МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Гордієнко Ю.Г., Таран В.І.

\

ХМАРНІ ОБЧИСЛЕННЯ

Конспект лекцій

Навчальний посібник
для здобувачів ступеня магістра
за освітньою програмою «Інженерія програмного забезпечення комп'ютерних систем»
спеціальності 121 «Інженерія програмного забезпечення»
за освітньою програмою «Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»
за освітньою програмою «Інформаційні управляючі системи та технології »
спеціальності 126 «Інформаційні системи та технології»

Електронне мережне навчальне видання

ЗАТВЕРДЖЕНО
на засіданні кафедри обчислювальної техніки
протокол № 10 від 25.05.2022

2022

****

Матеріали:

• Слайди лекцій
• Відеоверсії лекції
• Рекомендовані книги
• Теми для перспективних досліджень і розробок
• Екзаменаційні запитання
• Практичні запитання для самостійного навчання
• Підручник (конспект лекцій)

за посиланням:
https://cloud.comsys.kpi.ua/s/37mEqx2HdKz8oro

****

Хмарні обчислення

Конспект лекцій

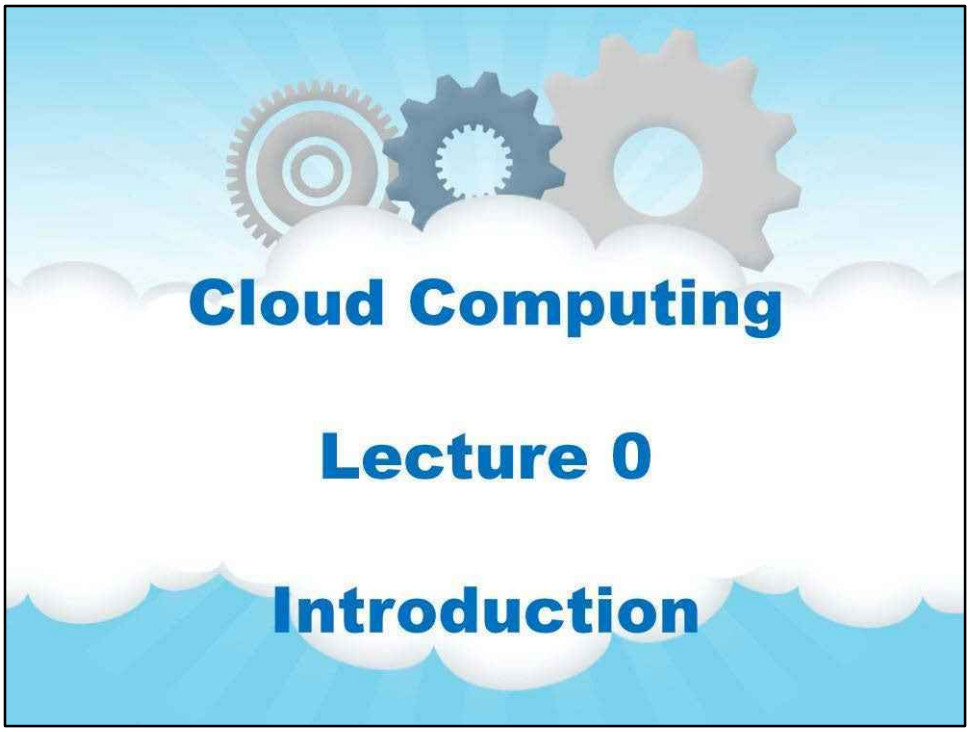
Том 0

Модуль 0

Вступ до контексту

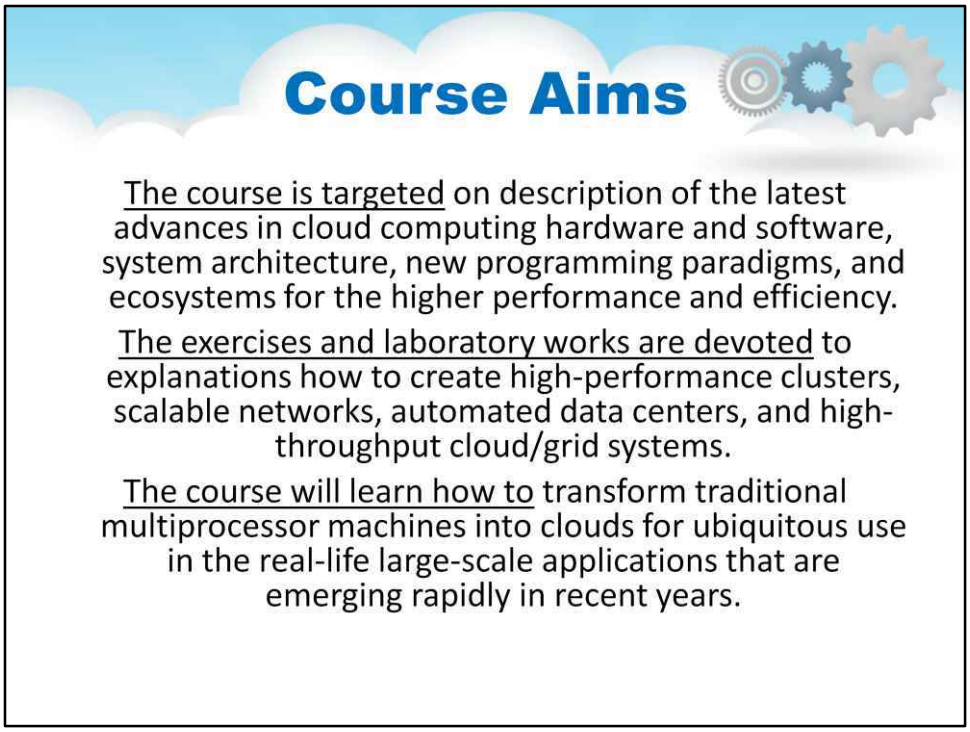
Хмарні обчислення

Зміст



Назва курсу «Хмарні обчислення»

Це вступна лекція № 0 про контекст хмарних обчислень.

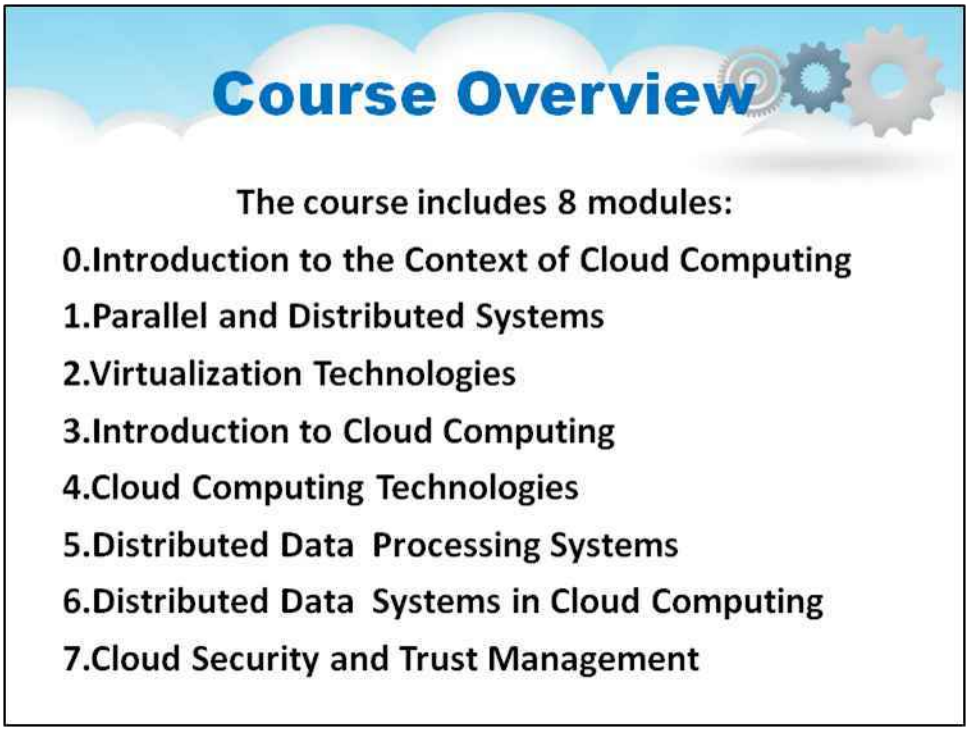


Розглянемо основні цілі цього курсу:

Курс спрямований на опис останніх досягнень апаратного та програмного забезпечення хмарних обчислень, системної архітектури, нових парадигм програмування та екосистем для підвищення продуктивності та ефективності.

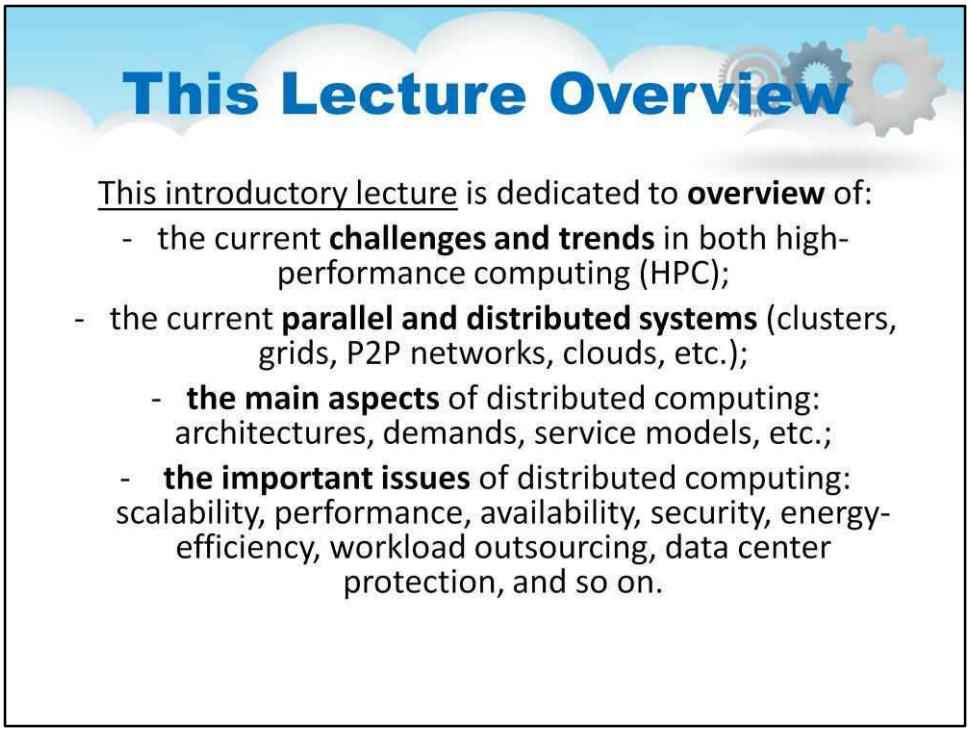
Вправи та лабораторні роботи присвячені поясненню того, як створювати високопродуктивні кластери, масштабовані мережі, автоматизовані центри обробки даних та високопродуктивні хмарні/грід-системи.

Курс навчить, як перетворити традиційні багатопроцесорні машини в хмари для повсюдного використання в реальних масштабних програмах, які швидко з’являються в останні роки.



Цей курс складається з наступних модулів.

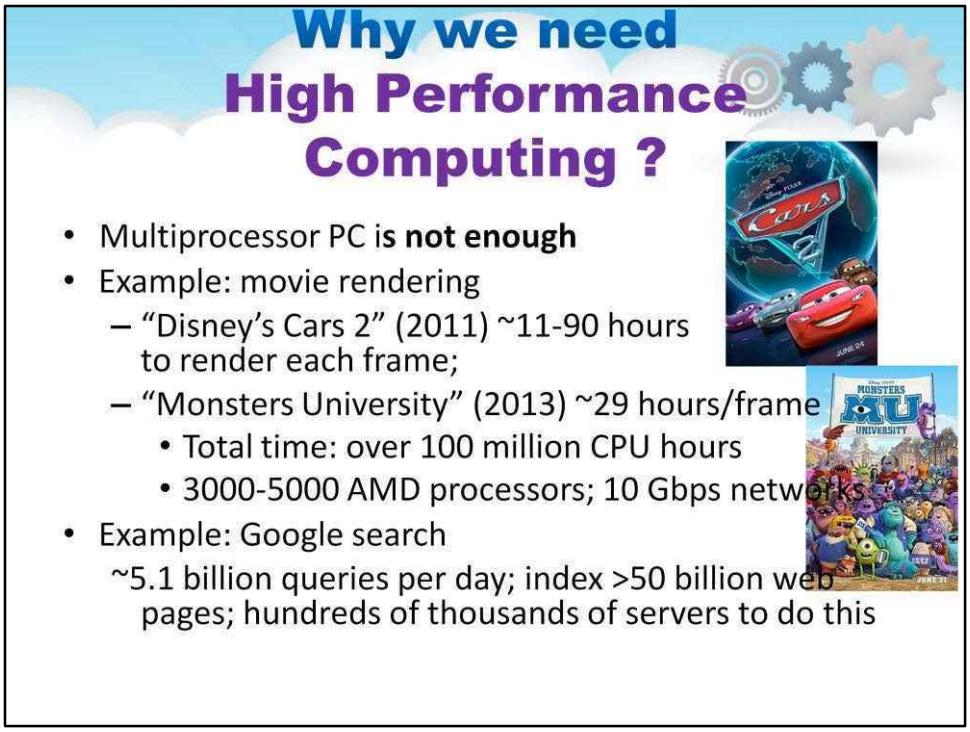
0. Вступ до контексту хмарних обчислень 1. Паралельні та розподілені системи
2. Технології віртуалізації
3. Вступ до хмарних обчислень
4. Технології хмарних обчислень
5. Розподілені системи обробки даних
6. Розподілені системи даних у хмарних обчисленнях
7. Хмарна безпека та управління довірою



Лекція 0. Введення в контекст хмарних обчислень

Поточні виклики

Огляд



У відповідь на запитання «Навіщо нам потрібні високопродуктивні обчислення» розглянемо кілька прикладів із нашого життя.

Для вирішення поточних обчислювальних проблем наявного багатопроцесорного персонального комп'ютера вже недостатньо.

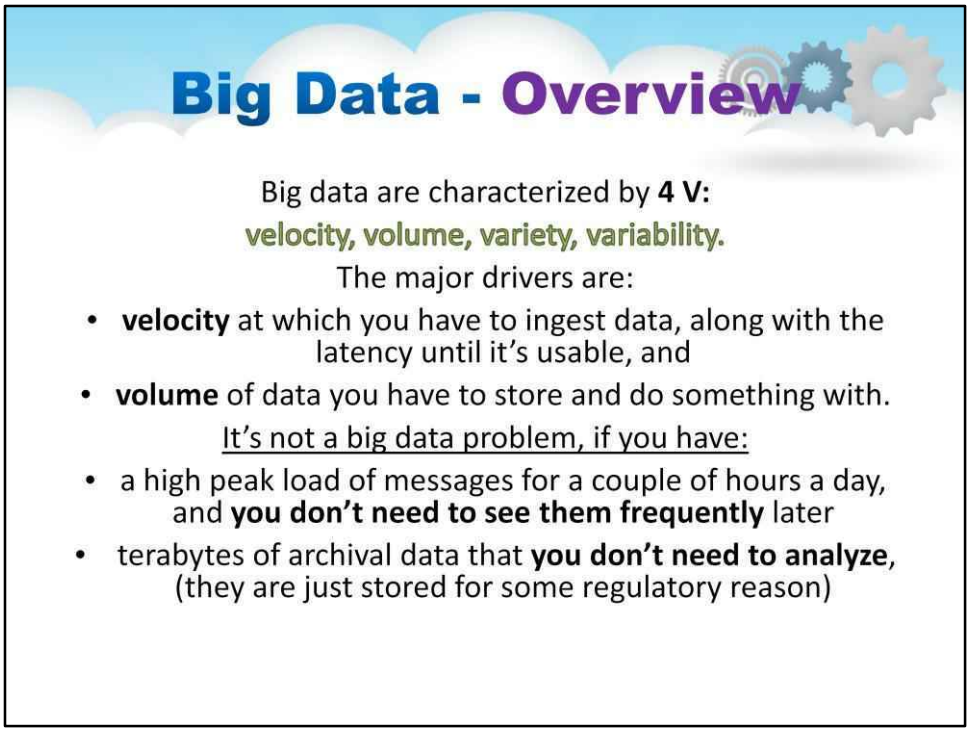
Наприклад, якщо ми хочемо створити якісний мультфільм. нам потрібно виконати операцію візуалізації. Він включає в себе розрахунок всіх елементів персонажів, об'єктів, їх рухів, кольорів, текстур, підсвічування і т.д.

Наприклад, для мультфільму «Тачки Діснея 2» у 2011 році обчислювальний час рендерингу кожного кадру становив ~11-90 годин.

Для іншого мультфільму «Університет монстрів» у 2013 році обчислювальний час рендерингу кожного кадру становив ~29 годин. А загальний обчислювальний час повинен перевищувати 100 мільйонів годин процесора! Відтворити такі фільми одним багатопроцесорним ПК, навіть найпотужнішим, неможливо.

Завдання рендерингу вирішувалася високопродуктивною обчислювальною системою з 3000-5000 процесорами AMD, з'єднаними високошвидкісними мережами 10 Гбіт/с.

Іншим прикладом є пошукова система Google, яка повинна обробляти ~5,1 мільярда запитів на день і індексувати >50 мільярдів веб-сторінок. Для цього повинні бути задіяні сотні тисяч серверів, а саме інфраструктура Cloud Computing повинна виконувати цю роботу!



Іншим прикладом є поточна проблема великих даних, пов'язана з обробкою великого обсягу даних, створених Інтернетом людей, Інтернетом речей та Інтернетом усього.

Детальніше ця проблема буде розглянута пізніше.

Але на цьому етапі, будь ласка, знайдіть наступний огляд проблеми великих даних.

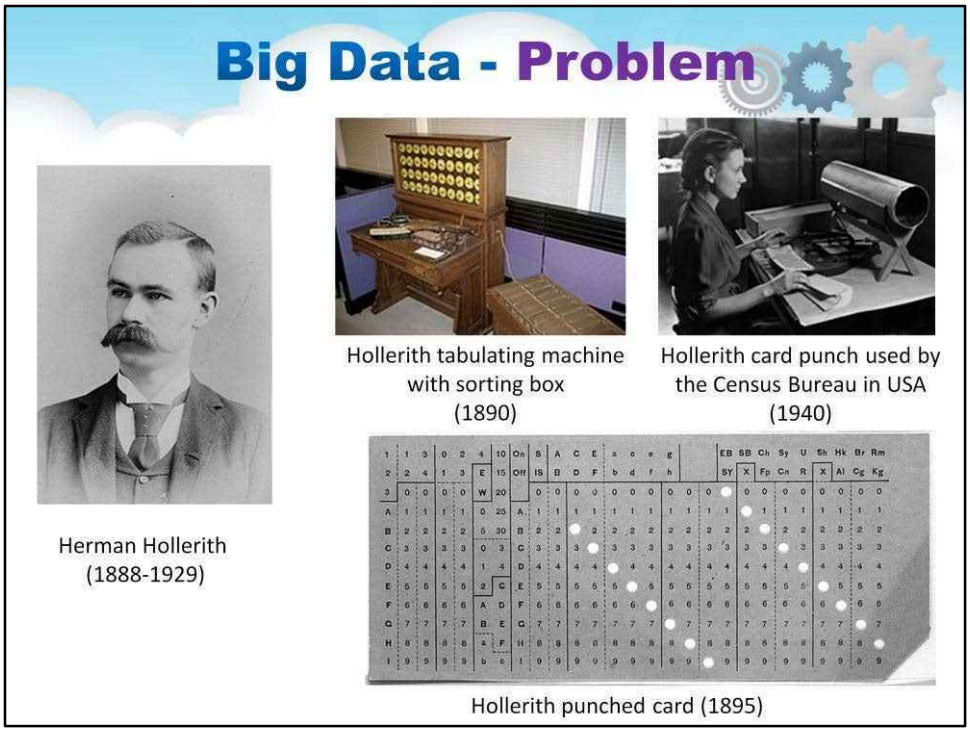
Великі дані характеризуються**4 В:** швидкість, обсяг, різноманітність, мінливість.

Основними драйверами є:

- **швидкість**коли ви повинні отримати дані разом із затримкою, поки вони не стануть придатними для використання, і
- **обсяг**даних, які потрібно зберігати та робити з ними. Але це не

велика проблема з даними, якщо у вас є:

- високе пікове навантаження повідомлень на пару годин на день, і**вам не потрібно бачити їх часто**пізніше
- терабайти архівних даних, які**не потрібно аналізувати**, (вони просто зберігаються з певних нормативних причин)



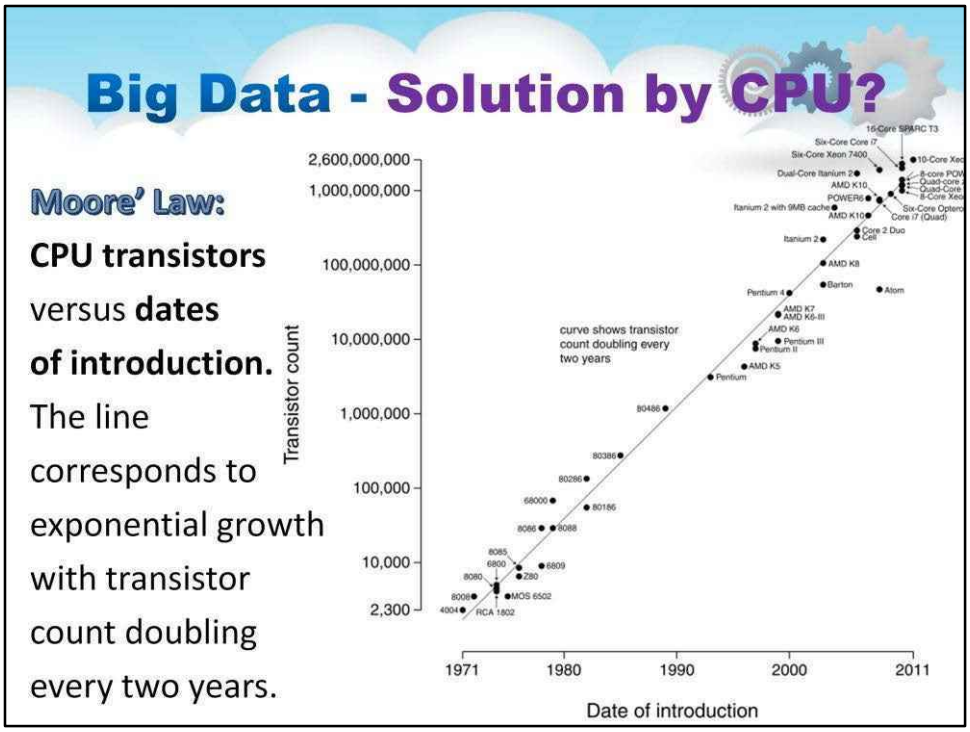
Коли виникла перша проблема великих даних?

Перша проблема великих даних виникла в 1880-х роках.

Наприкінці 1800-х років обробка даних перепису населення США почала займати близько 10 років. Проходить перепис**кожні 10 років** населення, а отже, кількість інформації збільшувалася —**проблема**!

У 1886 році Герман Холлеріт відкрив бізнес з оренди машин, які могли зчитувати та зводити дані перепису населення на перфокартах. Перепис 1890 року тривав менше 2 років, він охопив більшу кількість населення (62 мільйони людей) і більше даних, ніж перепис 1880 року.

Пізніше бізнес Холлеріта об'єднався з трьома іншими, щоб створити компанію IBM!



Тут ви можете побачити закон Мура:

Залежність кількості транзисторів всередині процесорапроти**дати впровадження.**

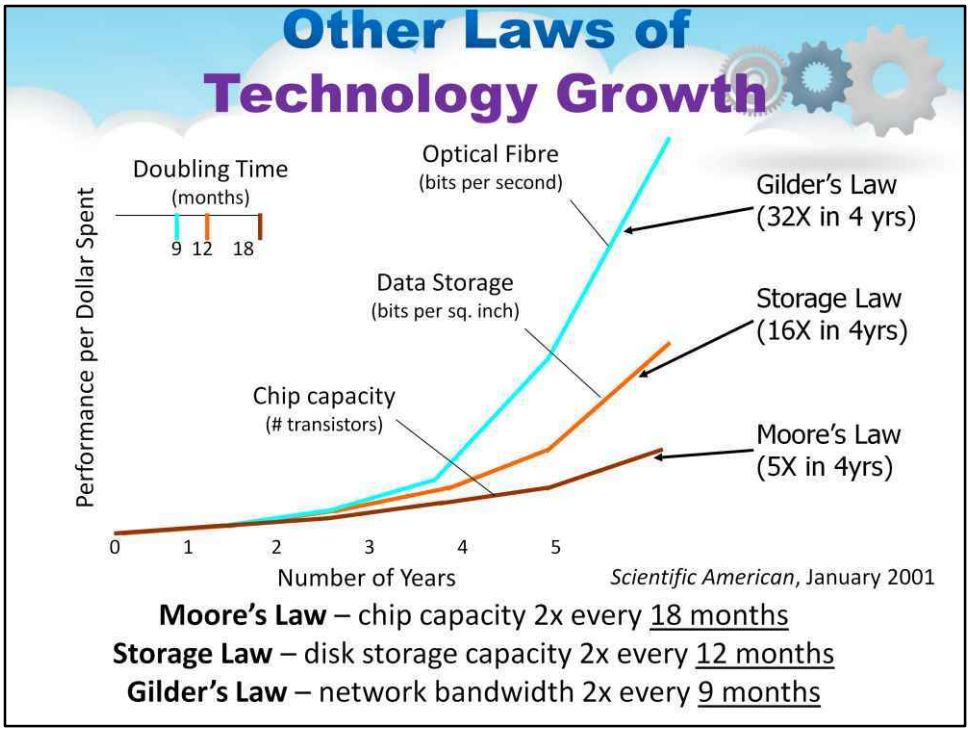
Лінія відповідає експоненційному зростанню з подвоєнням кількості транзисторів кожні два роки.

Ви бачите, що ця залежність насичується протягом останніх років через фізичні обмеження на розмір елементів.

Гордон Мур сказав у 2005 році:

«З точки зору розміру [транзисторів] ви бачите, що ми наближаємося до розміру атома, який є фундаментальним бар'єром, але мине два-три покоління, перш ніж ми зайдемо так далеко, але це так далеко, як ми» я коли-небудь міг бачити. У нас є ще 10-20 років, перш ніж ми досягнемо фундаментальної межі. До того часу вони зможуть виробляти більші чіпи та мати мільярдні бюджети транзисторів».

Але нинішнього розвитку єдиного центрального процесора (CPU) недостатньо для відповіді на поточні виклики.



Зараз існує кілька інших формулювань законів для характеристики тех~~нологічного~~ зростання:

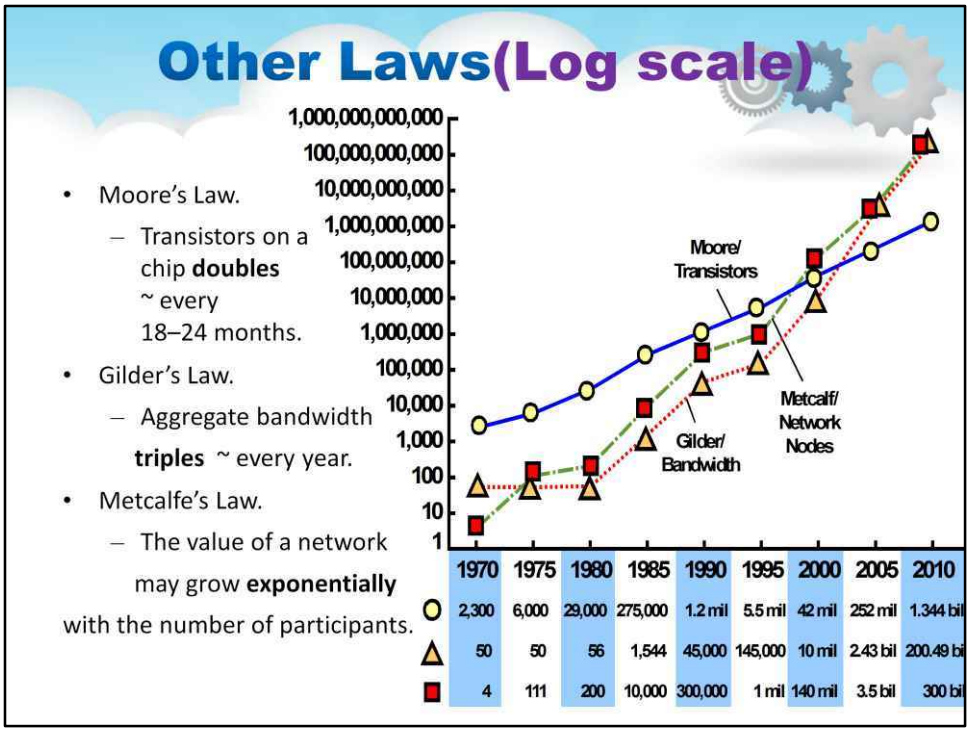
Закон Мура – Окремі комп'ютери подвоюються в профприпинення стор кожні 18 місяців Закон зберігання – ємність дискового сховища подвоюється кожні 12 місяців

Закон Гільдера – пропускна здатність мережі подвоюється кожні 9 місяців (але її важче встановити)

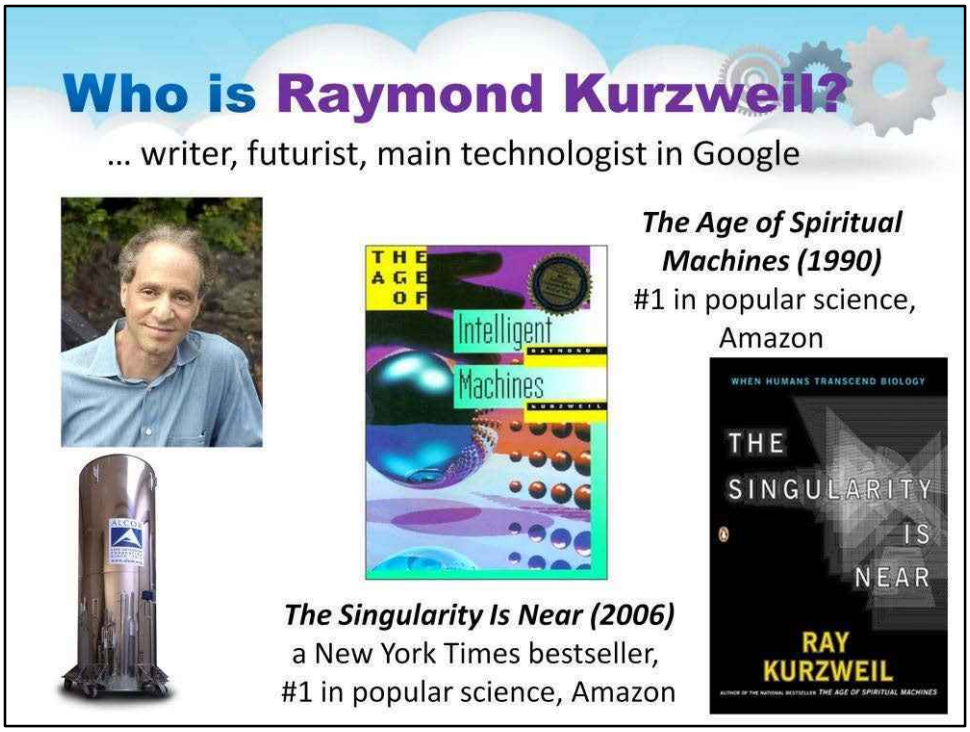
Це експоненційне зростання кардинально змінює ландшафт інформаційних технологій

Все більше і більше даних створюється (оскільки датчики менші, а процесори дешевші) і зберігається (оскільки диски дешевші й надійніші, ніж стрічки тощо), і до них здійснюється дистанційний доступ (оскільки мережа дешевша).

Зауважте, що час, необхідний для заповнення доступного сховища, зростає (тобто швидкість запису даних на диск зростає не так швидко, як ємність сховища)



Це логарифмічний графік цих законів.

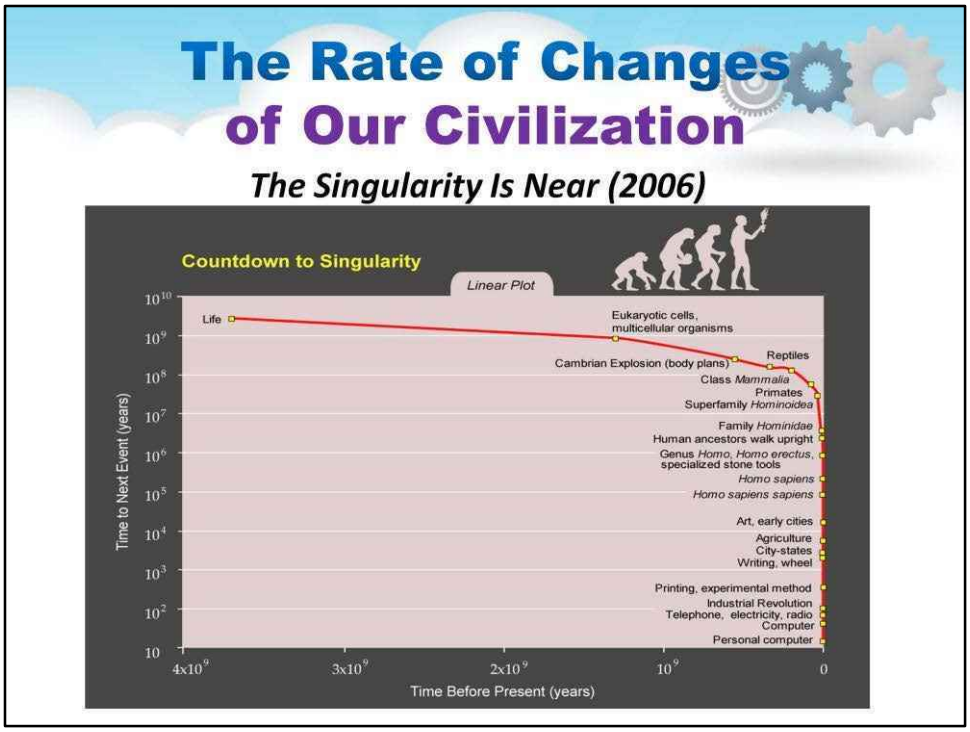


Реймонд Курцвейл - американський письменник, комп'ютерник, винахідник і футуролог.

Окрім футуризму, він займається такими галузями, як оптичне розпізнавання символів (OCR), синтез тексту в мову, технологія розпізнавання мови та електронні клавішні інструменти.

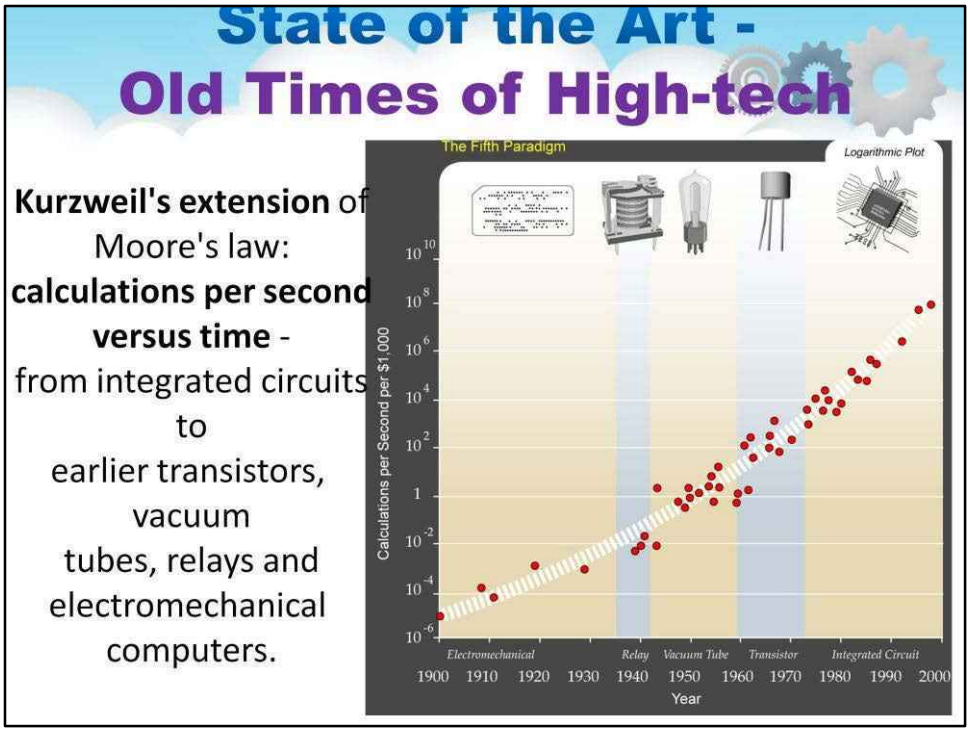
Він написав книги про здоров’я, штучний інтелект (ШІ), сингулярність.

У своїй першій книзі «Епоха розумних машин» Курцвейл представив свої ідеї щодо майбутнього в 1990 році. Курцвейл екстраполював тенденції у покращенні продуктивності програмного забезпечення комп’ютерних шахів, щоб передбачити, що комп’ютери переможуть найкращих гравців-людей «до 2000 року». У травні 1997 року чемпіон світу з шахів Гаррі Каспаров зазнав поразки від комп’ютера IBM Deep Blue у широко розголошеному шаховому матчі.



Він проаналізував еволюцію вирішальних подій нашої цивілізації та запропонував загальновизнану думку про те, що основною анатомічною відмінністю між людьми та іншими приматами, яка дозволяла вищі інтелектуальні здібності, була еволюція більшого неокортексу.

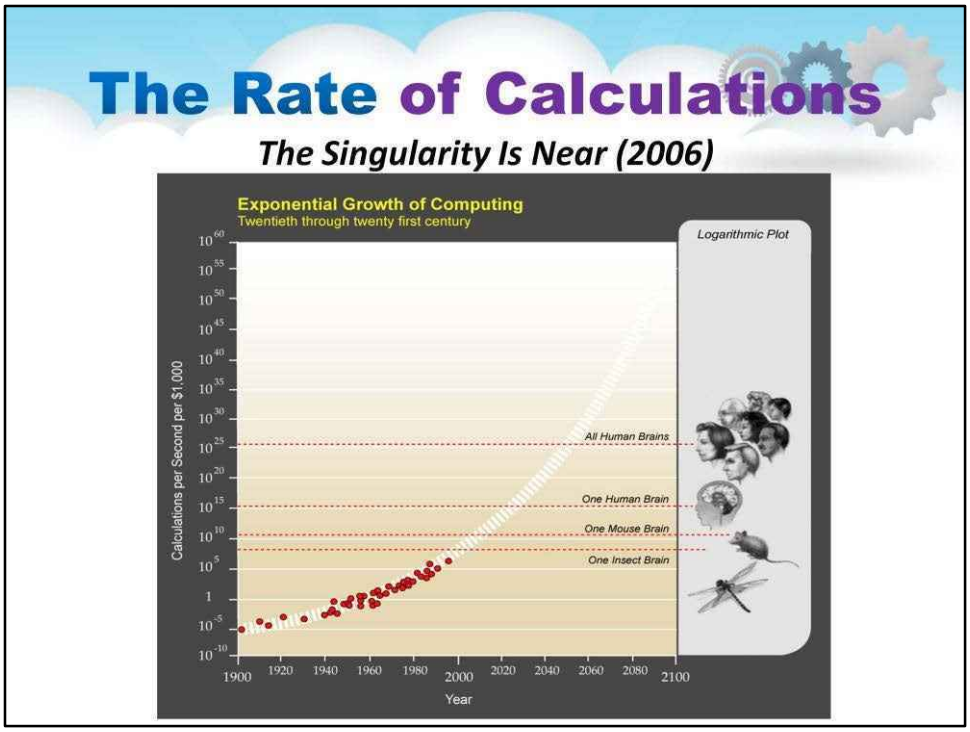
За словами Курцвейла, технологи створюватимуть синтетичні неокортекси на основі принципів роботи неокортекса людини з основною метою розширення наших власних неокортексів.



У своїй книзі 1999 року «Епоха духовних машин» Курцвейл запропонував «Закон прискореної віддачі», згідно з яким швидкість змін у різноманітних еволюційних системах (включно з розвитком технологій) має тенденцію до експоненціального зростання.

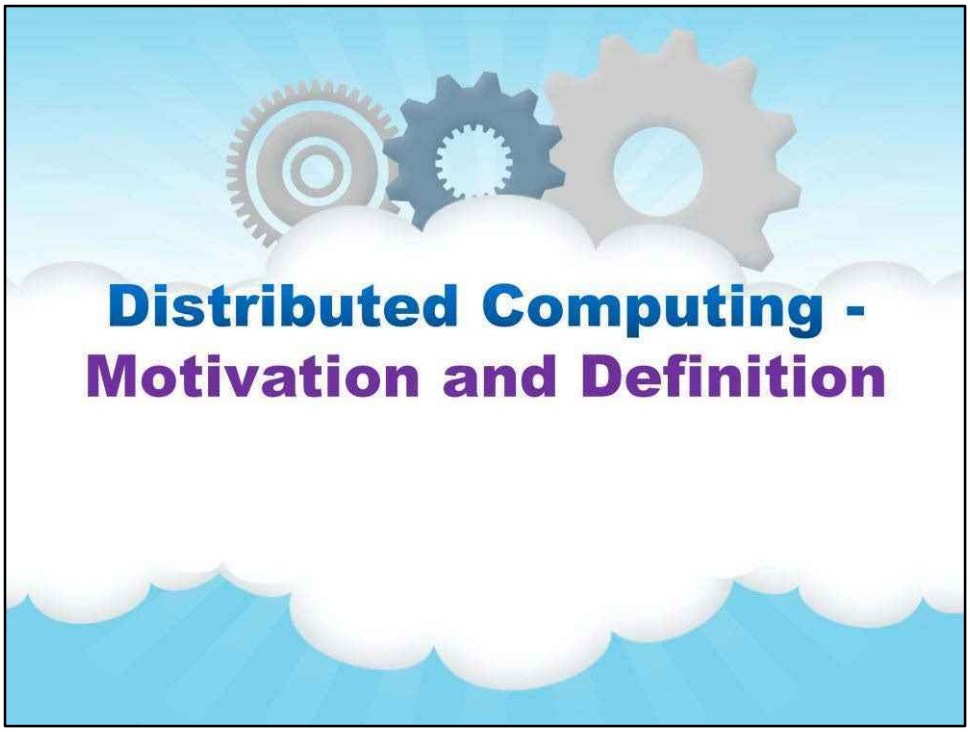
Він запропонував поширити закон Мура на широкий спектр технологій і використав це, щоб аргументувати на користь концепції технологічної сингулярності.

Можливо, найважливішим є те, що Курцвейл передбачив вибухове зростання всесвітнього використання Інтернету, яке почалося в 1990-х роках. На момент публікації «Епохи розумних машин» у світі було лише 2,6 мільйона користувачів Інтернету [62], а мережа була ненадійною, важкою у використанні та неповною. Він також заявив, що Інтернет вибухне не лише за кількістю користувачів, але й за вмістом, що зрештою надасть користувачам доступ «до міжнародних мереж бібліотек, баз даних та інформаційних служб». Крім того, Курцвейл стверджує, що правильно передбачив, що кращий спосіб доступу до Інтернету неминуче буде через бездротові системи, і він також мав рацію, коли оцінив, що останні стануть практичними для широкого використання на початку 21 століття.



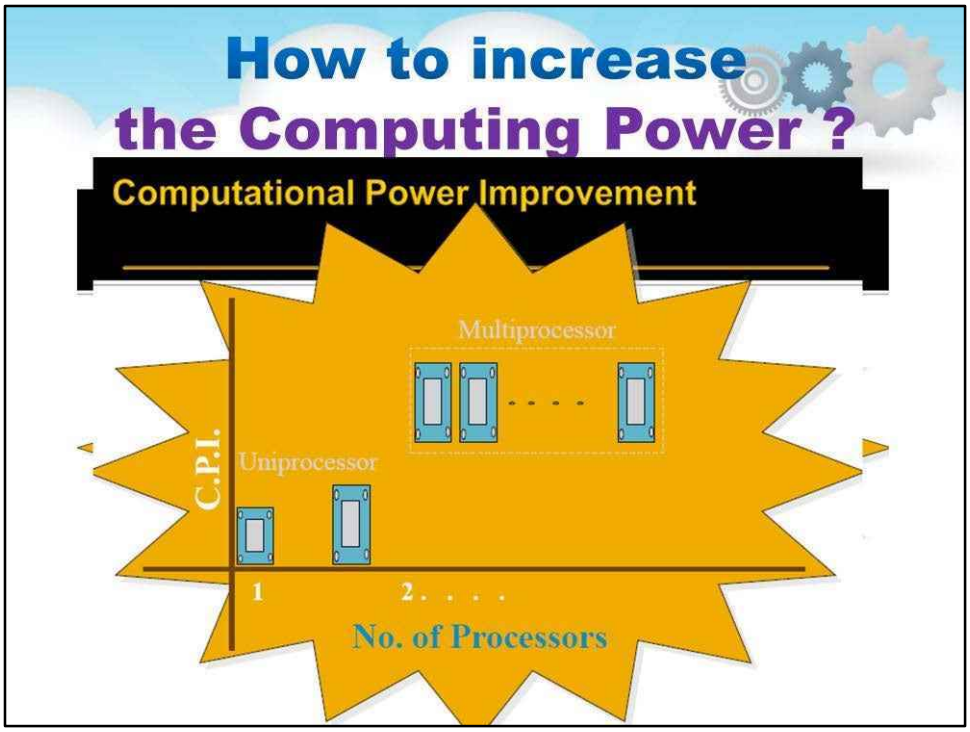
Курцвейл припускає, що це експоненційне технологічне зростання суперечить тому, як наш мозок сприймає світ, оскільки наш мозок був біологічно успадкований від людей, які живуть у світі, який був лінійним і локальним, і, як наслідок, він стверджує, що це заохочує великий скептицизм щодо його майбутнього прогнози.

У контексті згаданих вище викликів і проблем великих даних потреба в нових парадигмах для високопродуктивних обчислень є дуже високою та важливою.



Мотивація та визначення

для розподілених обчислень

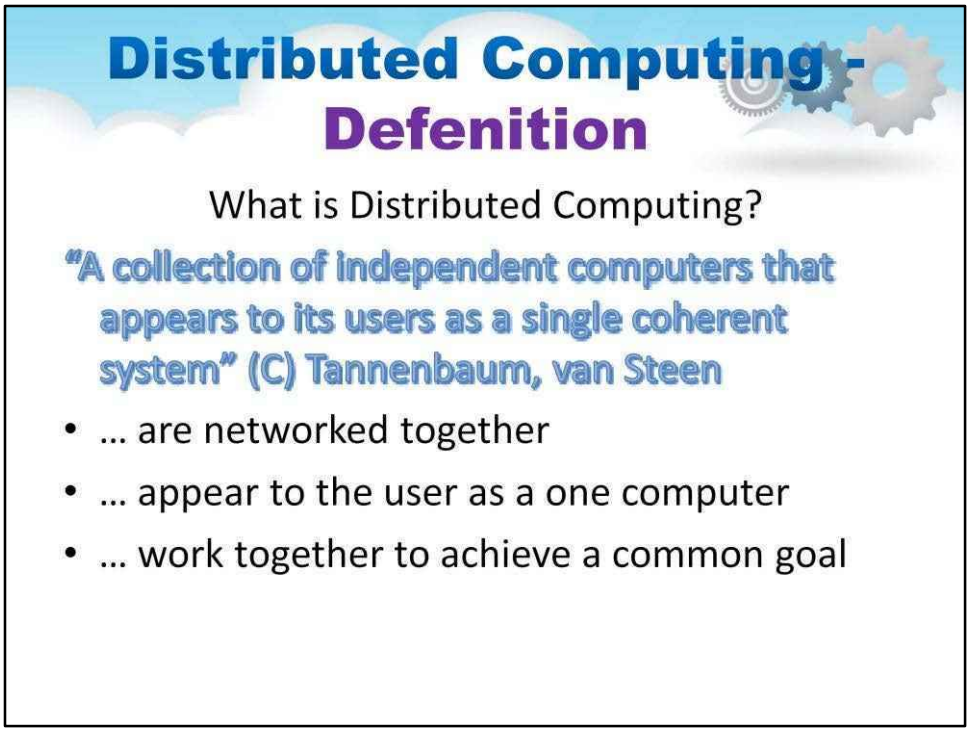


Як підвищити обчислювальну потужність?

Тут ви можете побачити аналогію зі збільшенням ваги людини.

Певною мірою будь-яка молода людина може збільшити**вага**зі збільшенням **висота**. Але після дитинства ця можливість закінчується, і дорослі люди можуть її збільшити**вага**зі збільшенням**ШИРИНА**тільки.

Така ж ситуація з обчислювальною потужністю.
Певною мірою ми можемо збільшити**обчислювальна потужність**зі збільшенням в **кількість транзисторів**. Але після обмеження Закону Мура ця можливість закінчується, і ми можемо збільшити**вага**зі збільшенням**кількість процесорів**тільки.



Що таке розподілене обчислення?

«Колекція незалежних комп'ютерів, які здаються користувачам єдиною цілісною системою» (С Танненбаум, ван Стін

… об'єднані в мережу

… здаються користувачеві одним комп'ютером …

працюють разом для досягнення спільної мети

Слід підкреслити такі важливі аспекти цього визначення:

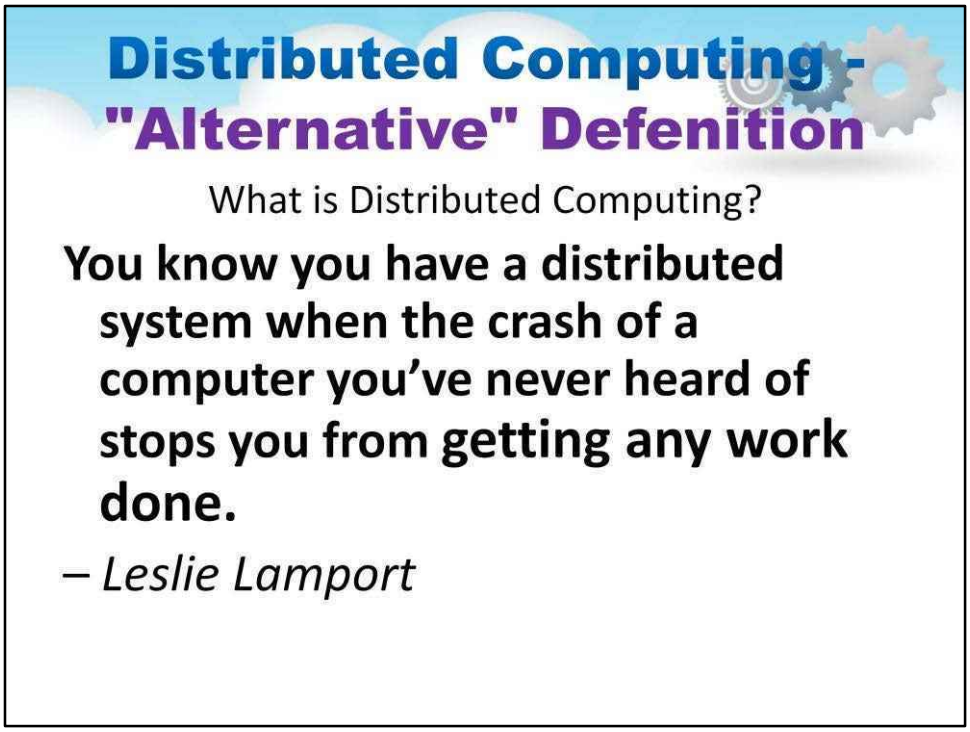
• з точки зору архітектури системи: машини є автономними; це означає, що це комп'ютери, які, в принципі, можуть працювати незалежно;

• з точки зору користувача: розподілена система сприймається як єдина система, що вирішує певну проблему (хоча насправді ми маємо кілька комп'ютерів, розміщених у різних місцях).

За допомогою програмного забезпечення розподіленої системи комп'ютери можуть:

- координувати свою діяльність

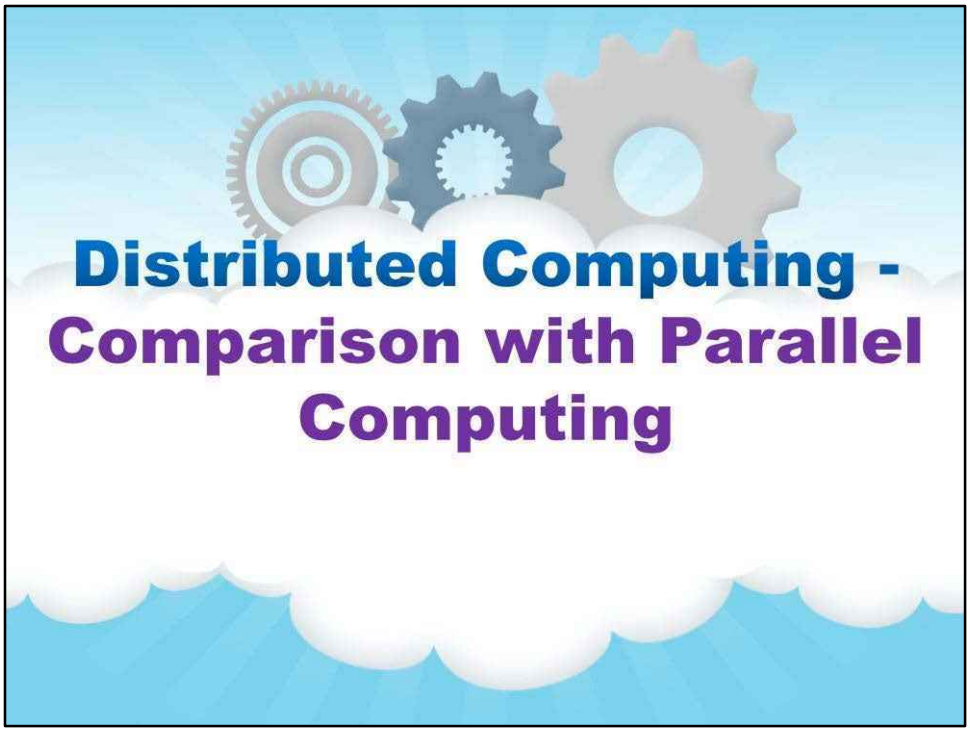
- спільно використовувати ресурси: обладнання, програмне забезпечення, дані.



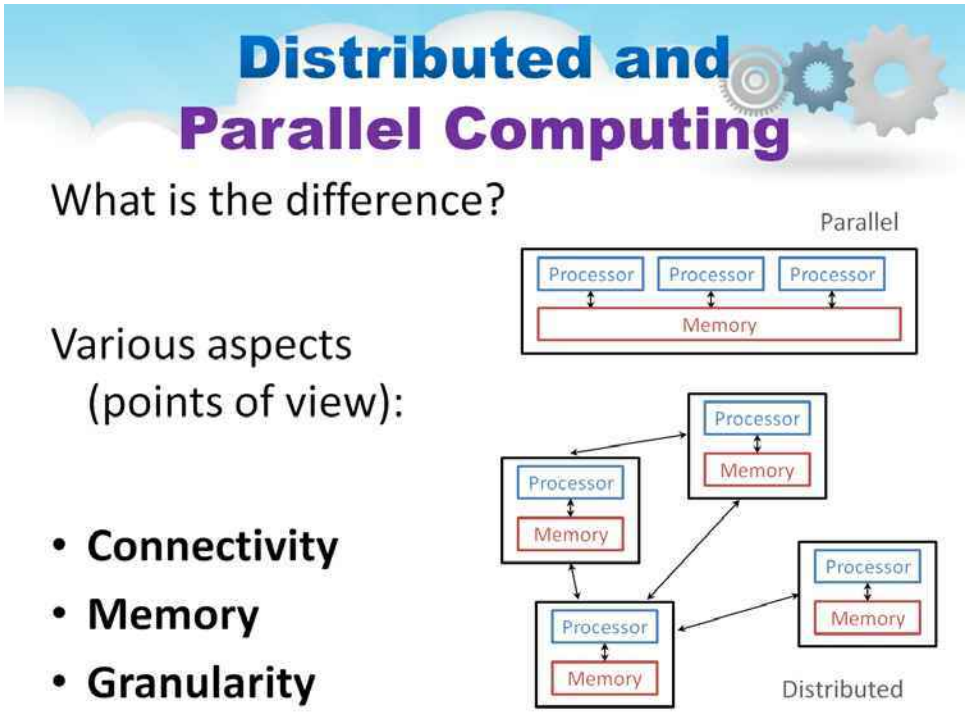
Це означає, що:

- відмінності між різними комп'ютерами, способи їх зв'язку та внутрішня організація розподіленої системи приховані від користувачів;

- користувачі та додатки можуть взаємодіяти з розподіленою системою узгодженим і рівномірним способом, незалежно від того, де і коли відбувається взаємодія.



Порівняння з паралельними обчисленнями



Підключення -Ступінь зв’язку між набором модулів, будь то апаратне чи програмне забезпечення, вимірюється з точки зору взаємозалежності, зв’язку та/або однорідності між модулями. Коли ступінь зв’язку високий (низький), модулі вважаються тісно (слабко) зв’язаними.

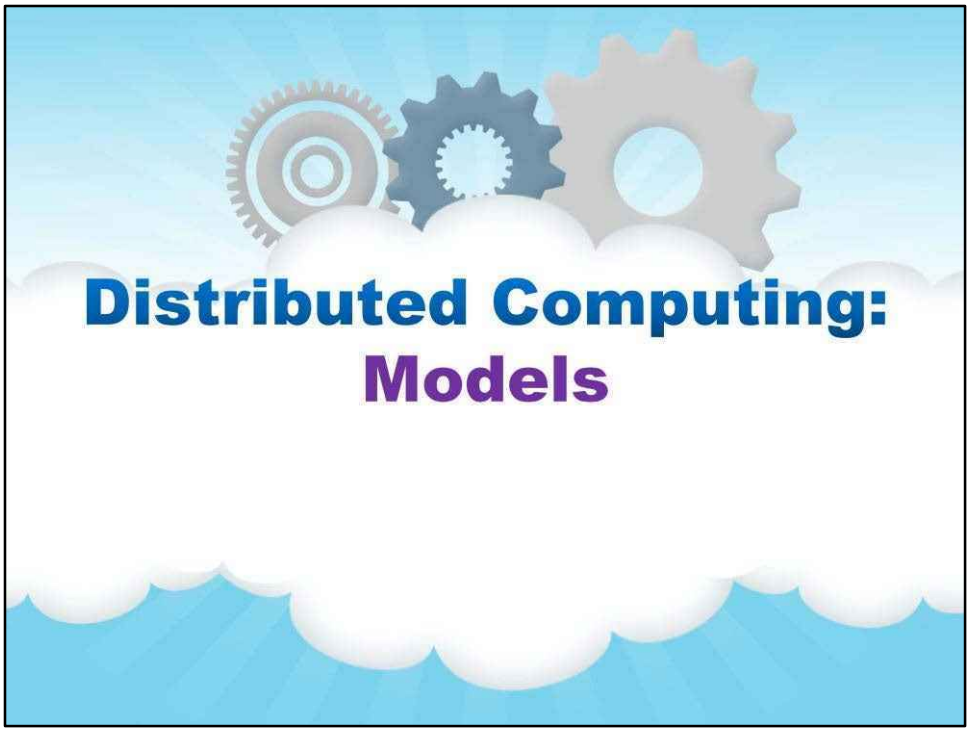
Пам'ять - Спільна пам'ятьсистеми — це ті, в яких існує (загальний) спільний адресний простір у всій системі. Відбувається зв'язок між процесорами через загальні змінні даних і змінні керування для синхронізації між процесорами. Семафори та монітори, які спочатку були розроблені для

Однопроцесорні та багатопроцесорні процесори зі спільною пам’яттю є прикладами того, як можна досягти синхронізації**системи спільної пам'яті**. Усі багатокомп’ютерні системи, які не мають спільного адресного простору, що забезпечується базовою архітектурою та апаратним забезпеченням –**системи розподіленої пам'яті**– обов'язково спілкуватися за допомогою передачі повідомлень. Абстракція спільна пам'ять може надається для імітації спільного адресного простору. Для розподіленої системи ця абстракція називається **системи розподіленої спільної пам'яті**

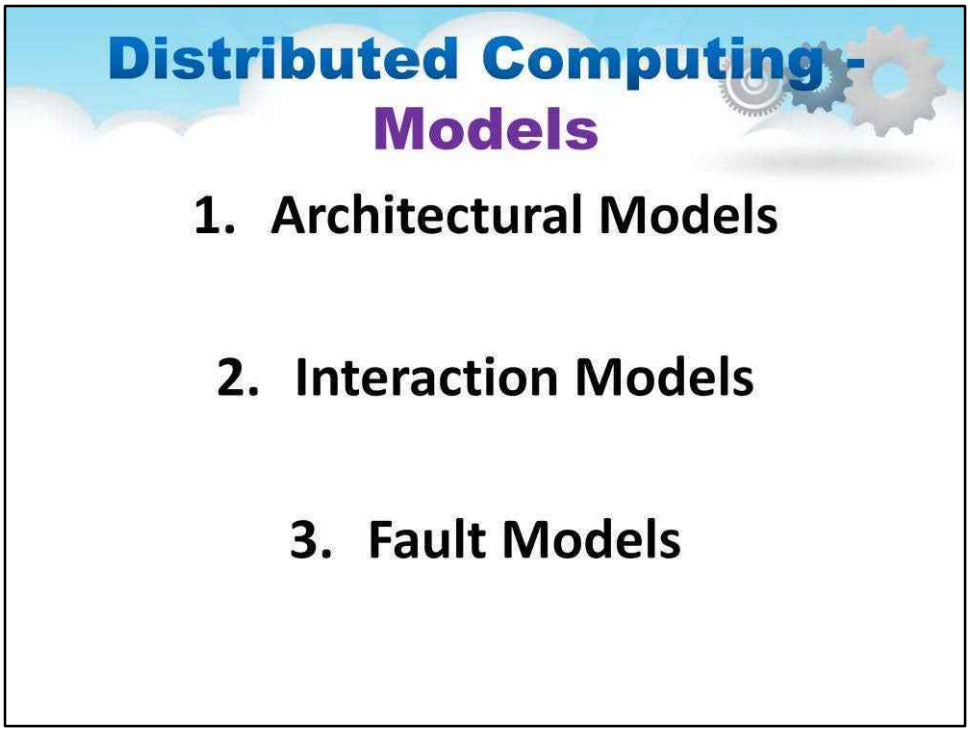
Зернистість–Відношення обсягу обчислень до обсягу зв’язку в рамках паралельної/ розподіленої програми називається зернистість. Якщо ступінь паралелізму є грубозернистим (дрібнозернистим), продуктивних інструкцій ЦП відносно набагато більше (менше) порівняно з кількістю

часів, коли процесори спілкуються або через спільну пам’ять, або через передачу повідомлень і чекають синхронізації з іншими процесорами. Програми з

дрібнозернистий паралелізм найкраще підходить для тісно пов’язаних систем.

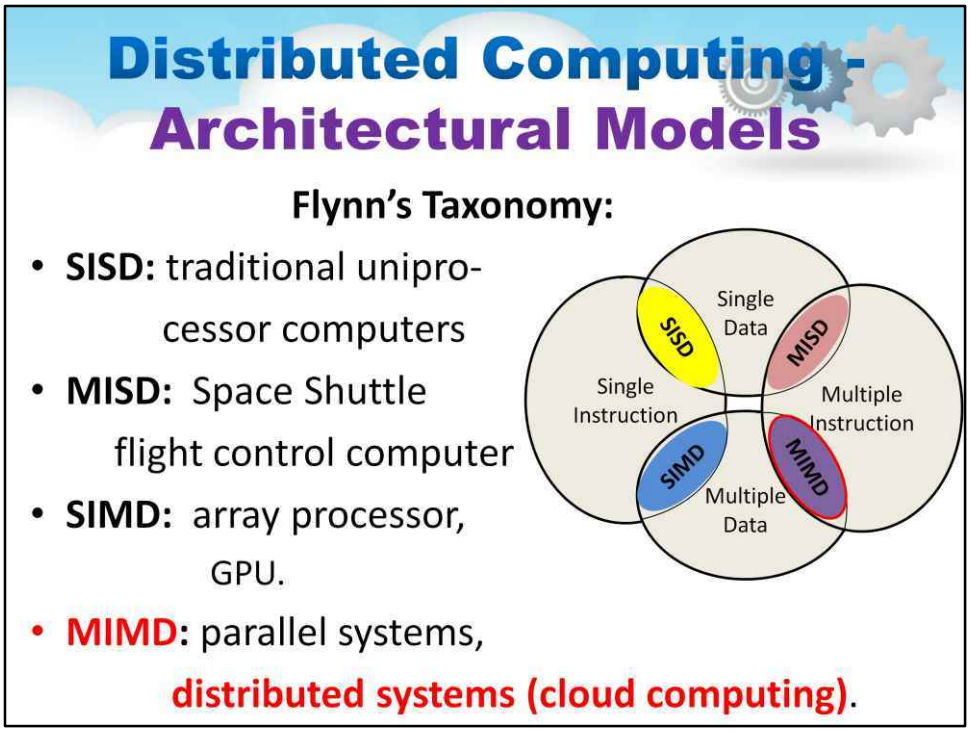


Моделі



Моделі розподілених обчислень поділяються на такі категорії:

1.**Архітектурні моделі**
2.**Моделі взаємодії**
3.**Моделі несправностей**

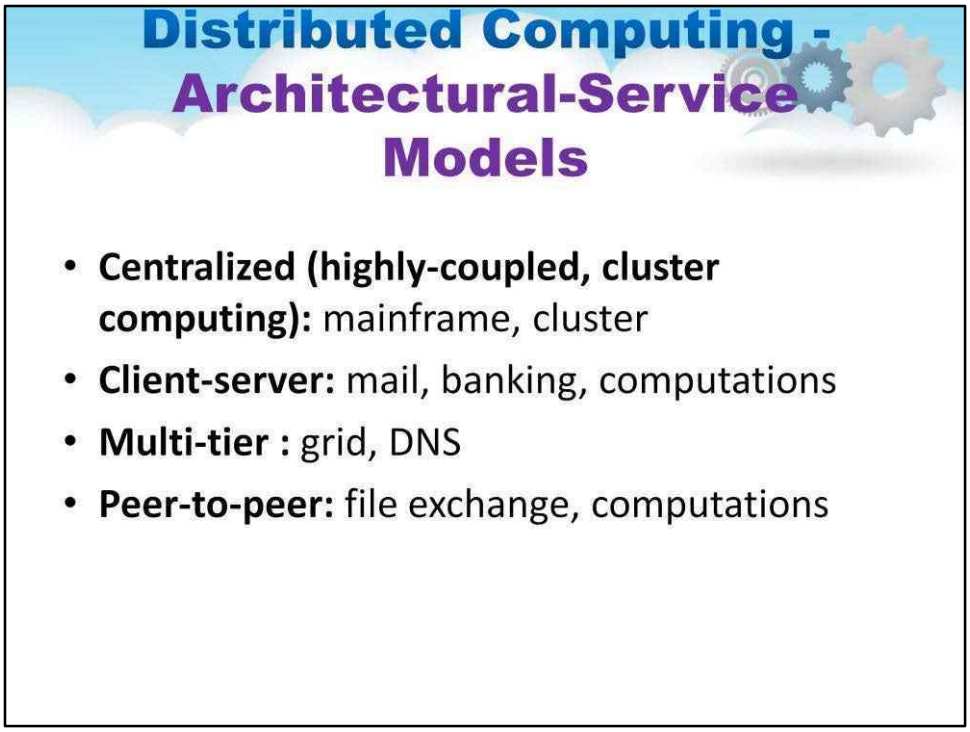


Таксономія Флінна — класифікація комп'ютерних архітектур, запропонована Майклом Дж. Флінном у 1966 році.

Ця класифікація заснована на кількості одночасних**Інструкція**потоки і **Дані** потоки, доступні в архітектурі:**неодруженийабомножинний**.

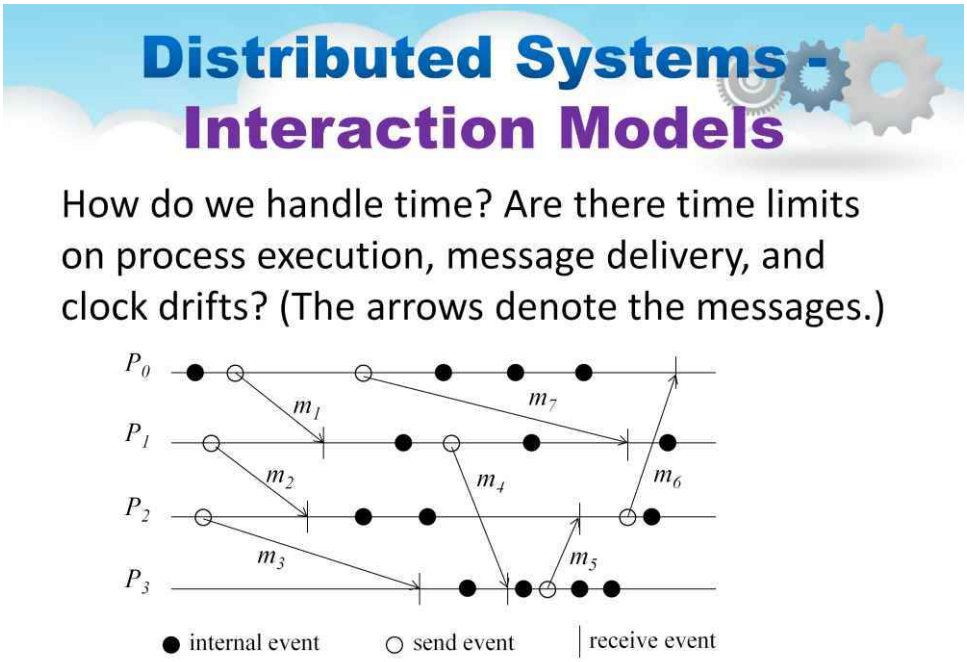
Таксономія Флінна включає:

- **SISD:**традиційні однопроцесорні комп'ютери
- **MISD:**Комп'ютер управління польотом космічного човника
- **SIMD:**масивний процесор, GPU.
- **MIMD:**паралельні системи,**розподілені системи (хмарні обчислення)**.



Архітектурно-сервісні моделі розподілених обчислень поділяються на такі категорії:

- Централізовані (високозв’язані, кластерні обчислення): мейнфрейм, кластер
- Клієнт-сервер: пошта, банкінг, наукові обчислення
- Багаторівневі: грід-системи, сервери доменних імен (DNS)
- Одноранговий: обмін файлами, наукові обчислення



• Асинхронне виконання — це виконання, у якому

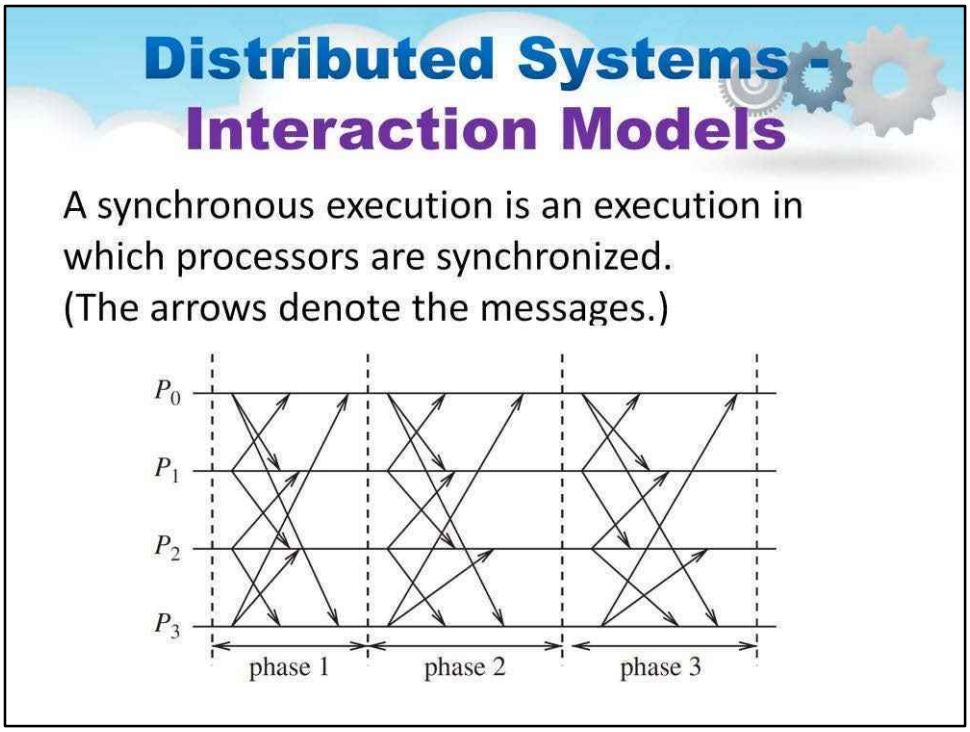
(i) немає синхронізації процесора та немає обмежень на швидкість дрейфу тактової частоти процесора,

(ii) затримки повідомлення (передача + час розповсюдження) є кінцевими, але необмеженими, і

(iii) немає верхньої межі часу, необхідного процесу для виконання кроку.

На цьому малюнку показано приклад асинхронного виконання з чотирма процесами від P0 до P3.

Стрілки позначають повідомлення; хвіст і головка стрілки позначають подію надсилання та отримання для цього повідомлення, позначені колом і вертикальною лінією відповідно. Події, пов’язані з відсутністю зв’язку, які також називають внутрішніми подіями, відображаються заштрихованими колами.



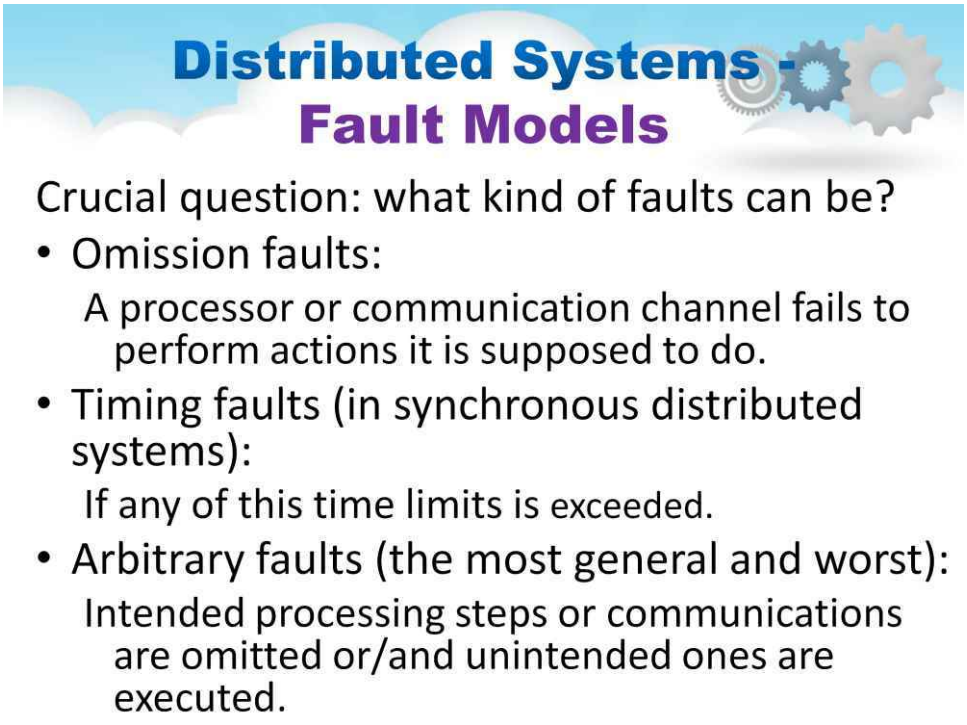
А синхронне виконання — це виконання, при якому

(i) процесори синхронізовані, і швидкість дрейфу тактової частоти між будь-якими двома процесорами обмежена,

(ii) час доставки повідомлення (передача + доставка) такий, що відбувається в один логічний крок або раунд, і

(iii) існує відома верхня межа часу, необхідного процесу для виконання кроку.

Приклад синхронного виконання з чотирма процесами від P0 до P3 показано на цьому малюнку.

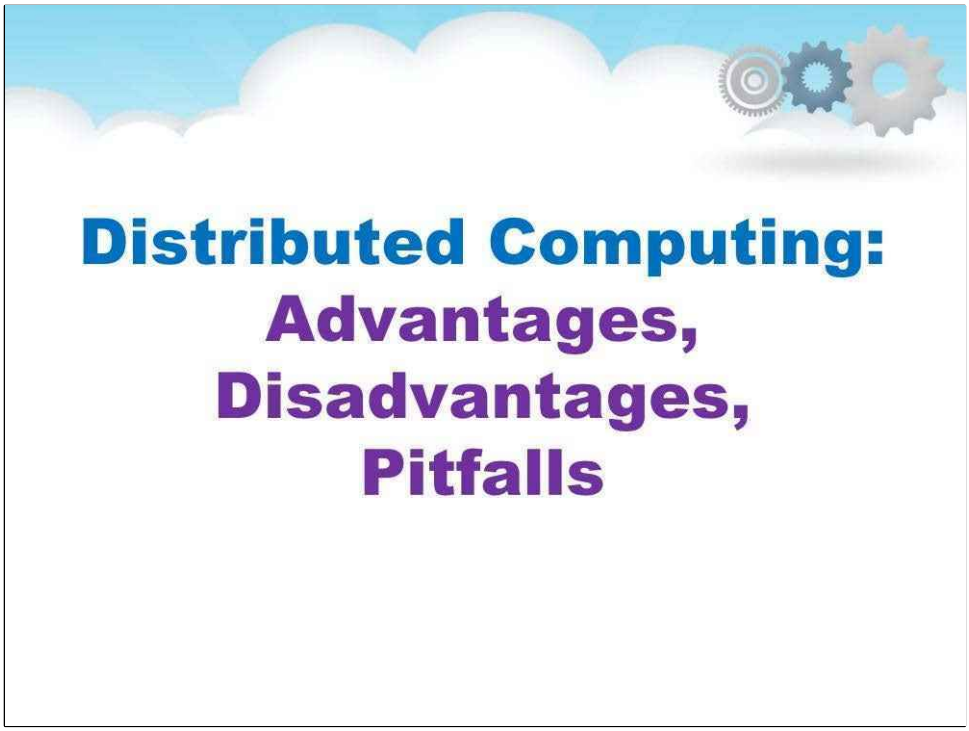


Модель несправностей визначає збої, які можуть виникнути в розподілених системах. Hadzilacos і Toueg пропонують таксономію моделей розломів.

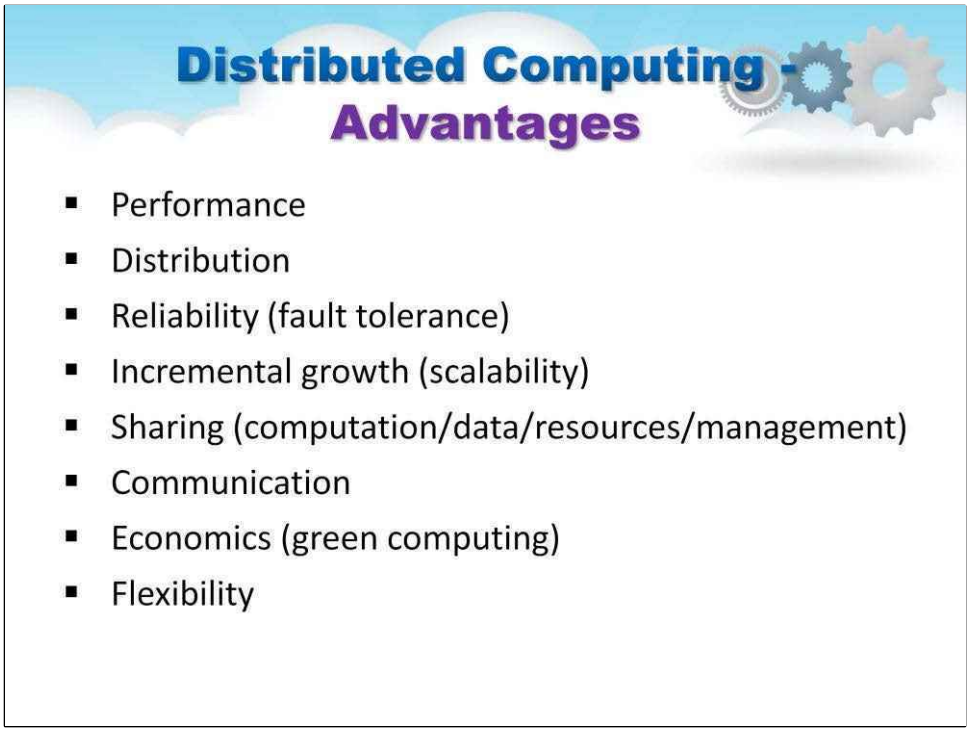
процес**пропуск**невдачі : Головна помилка процесу — збій. Коли ми говоримо, що процес вийшов з ладу, ми маємо на увазі, що він зупинився і більше не виконуватиме жодних кроків своєї програми.
спілкування**пропуск**невдачі: Канал зв'язку створює помилку пропуску, якщо він не транспортує повідомлення з буфер вихідних повідомлень p на буфер вхідних повідомлень q. Це відомий як «скидання повідомлень» і зазвичай спричинений браком буферного простору в приймачі чи на проміжному шлюзі, або через помилку передачі мережі, виявлену контрольною сумою, що передається з даними повідомлення.

Часневдачі: Вони застосовуються в синхронних розподілених системах, де встановлюються часові обмеження на час виконання процесу, час доставки повідомлень і швидкість дрейфу годинника. Будь-яка з цих помилок може призвести до того, що відповіді будуть недоступні для клієнтів протягом визначеного інтервалу часу.

Довільнийневдачі: Термін довільна або візантійська невдача використовується для опису найгіршої семантика відмови, в якій може виникнути помилка будь-якого типу. Наприклад, процес може встановити неправильні значення у своїх елементах даних або він може повернути неправильне значення у відповідь на виклик.



Переваги, недоліки, підводні камені



Продуктивність: дуже часто сукупність процесорів може забезпечити вищу продуктивність (і краще співвідношення ціна/продуктивність), ніж централізований комп'ютер. Підвищена продуктивність завдяки розподілу навантаження.

Розподіл: багато додатків за своєю суттю є розподіленими – включають, за своєю природою, просторово розділені машини (банківська, комерційна, автомобільна системи).

Надійність (відмовостійкість): немає єдиної точки відмови, якщо деякі з машин вийдуть з ладу, система може вижити.

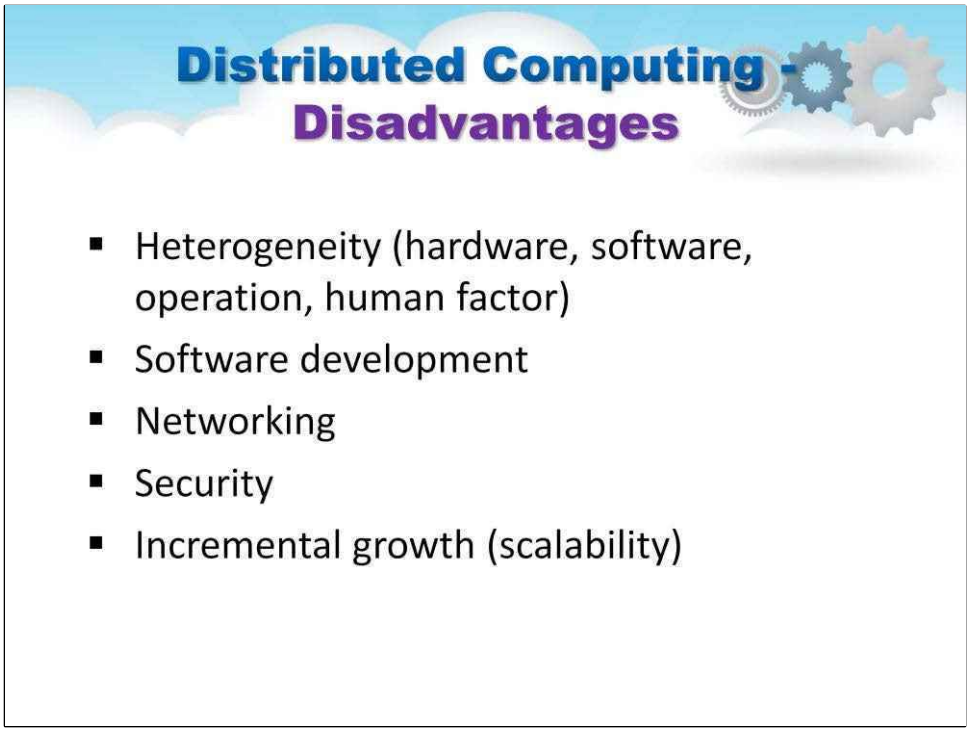
Поступове зростання (масштабованість): у міру зростання вимог до обчислювальної потужності можна поступово додавати нові машини.

Спільне використання обчислень/даних/ресурсів/управління: складні обчислення можуть бути спільними; спільні дані є важливими для багатьох програм (банківська справа, комп'ютерна підтримка спільної роботи, системи бронювання); інші ресурси також можуть бути спільними (наприклад, дорогі принтери).

Спілкування: полегшує спілкування між людьми.

Економічність (зелені обчислення): нижче співвідношення (ціна/продуктивність), ефективне енергоспоживання.

Гнучкість: балансування навантаження, розподіл навантаження на багато компонентів.

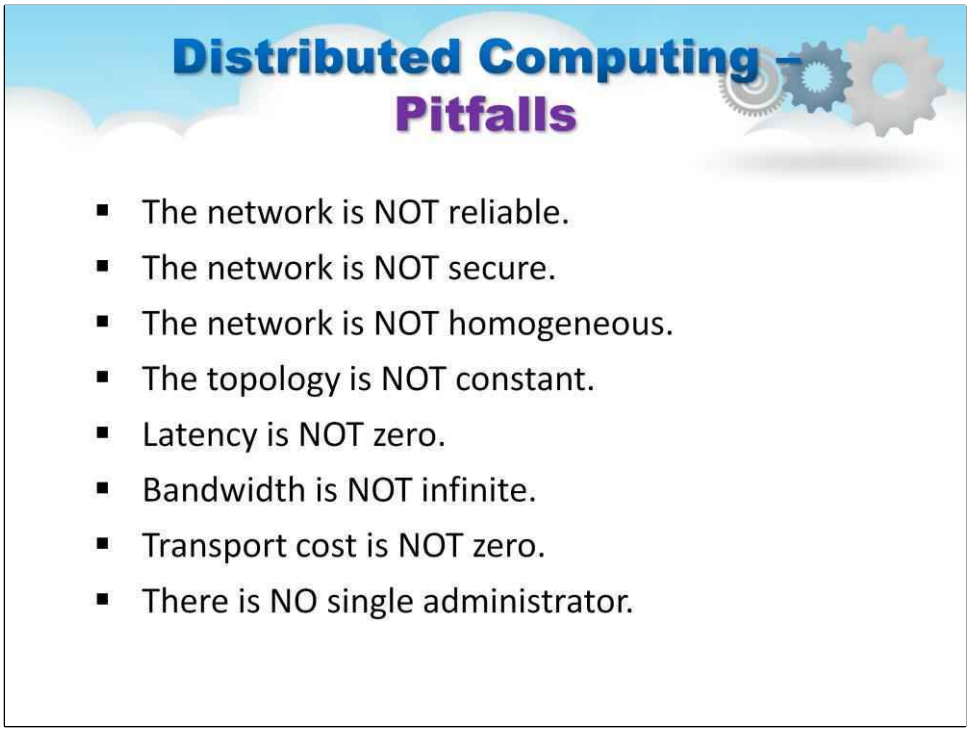


Труднощі розробки розподіленого програмного забезпечення: як мають виглядати операційні системи, мови програмування та програми? Розробити програмне забезпечення розподіленої системи важко.

Проблеми з мережею: мережева інфраструктура створює декілька проблем, які необхідно вирішити: втрата повідомлень, перевантаження тощо. Коли мережа перевантажена/втрачені повідомлення, перенаправлення/переналаштування мережі є дорогим/складним.

Проблеми безпеки: спільний доступ створює проблему безпеки даних. Більший обмін призводить до меншої безпеки, особливо в питаннях конфіденційності та цілісності.

Поступове зростання: важко на практиці через зміну апаратного та програмного забезпечення.



Розподілені системи відрізняються від традиційного програмного забезпечення тим, що компоненти розподілені по мережі. Неврахування цієї дисперсії під час проектування робить так багато систем непотрібно складними та призводить до помилок, які потрібно виправляти пізніше.

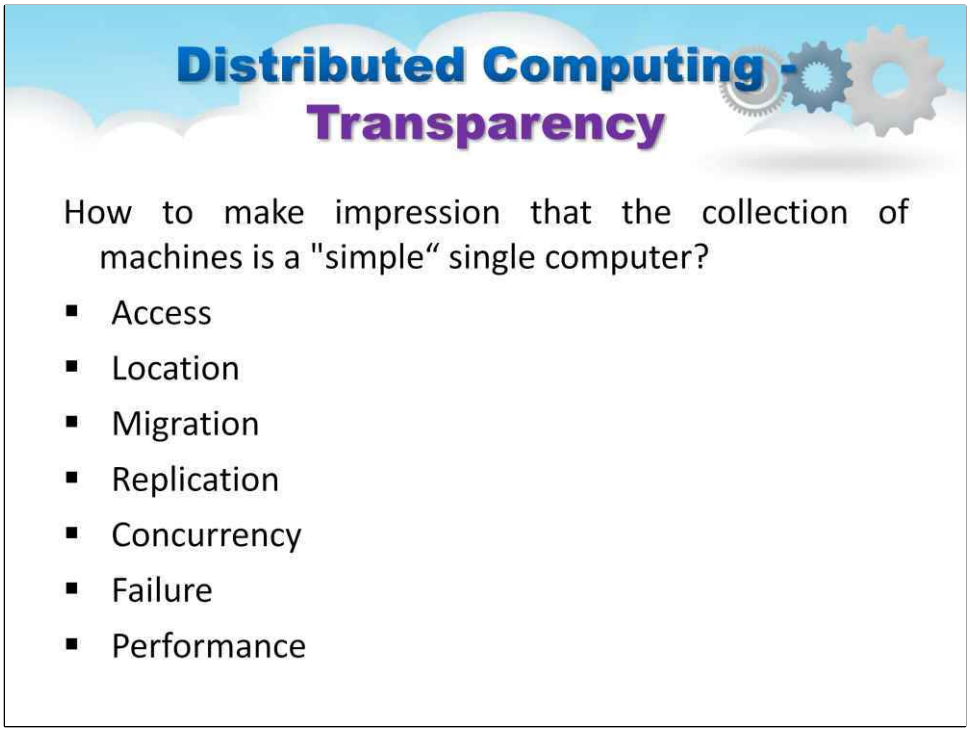
Пітер Дойч, коли він працював у компанії Sun Microsystems, сформулював ці помилки як наступні помилкові припущення, які кожен робить, коли розробляє розподілену програму вперше.

Дизайн

У цьому розділі ми зарахуємо кілька та розглянемо 4 важливі характеристики, які слід враховувати для проектування та створення ефективної розподіленої системи.

Розподілена система повинна

- зробити ресурси легкодоступними;
- розумно приховати той факт, що ресурси розподілені по мережі;
- бути відкритим;
- бути масштабованим;
- і інші.



Прозорість доступу: доступ до локальних і віддалених ресурсів здійснюється за допомогою ідентичних операцій.

Прозорість розташування: користувачі не можуть визначити, де розташовані апаратні та програмні ресурси (ЦП, файли, бази даних); назва ресурсу не повинна кодувати місце розташування ресурсу.

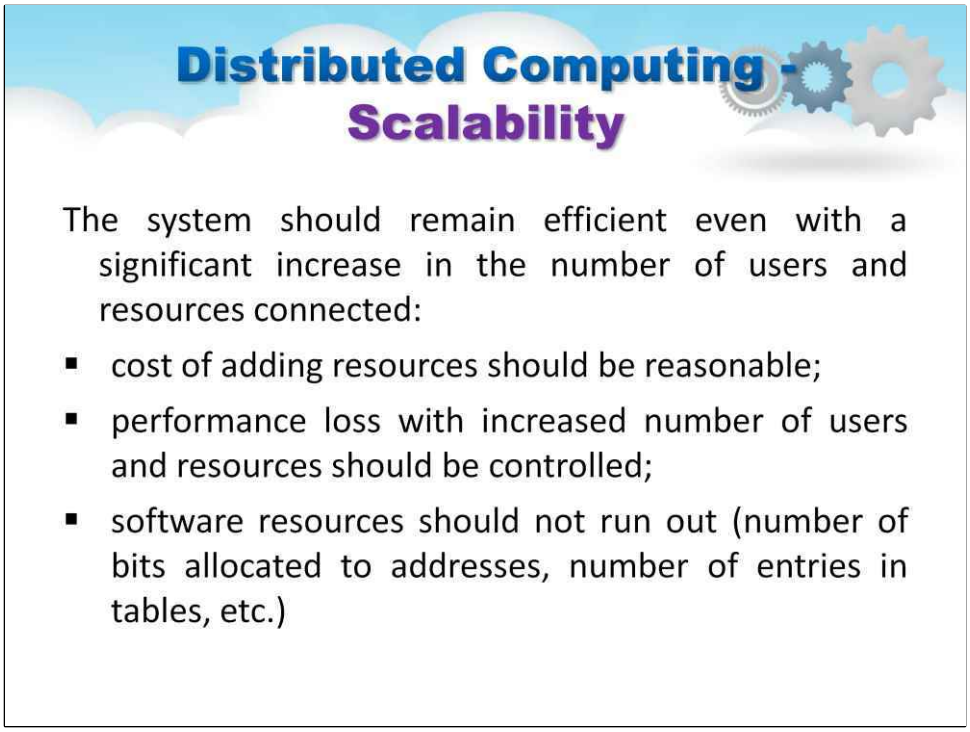
Прозорість міграції (мобільності): ресурси повинні вільно переміщатися з одного місця в інше без зміни назв.

Прозорість реплікації: система може вільно створювати додаткові копії файлів та інших ресурсів (з метою підвищення продуктивності та/або надійності), непомітно для користувачів. Приклад: декілька копій файлу; за певним запитом здійснюється доступ до тієї копії, яка є найближчою до клієнта.

Прозорість паралелізму: користувачі не помітять існування інших користувачів у системі (навіть якщо вони мають доступ до тих же ресурсів).

Прозорість збоїв: програми повинні мати можливість виконувати свої завдання, незважаючи на збої в певних компонентах системи.

Прозорість продуктивності: зміна навантаження не повинна призводити до погіршення продуктивності. Цього можна досягти шляхом автоматичного перенастроювання.



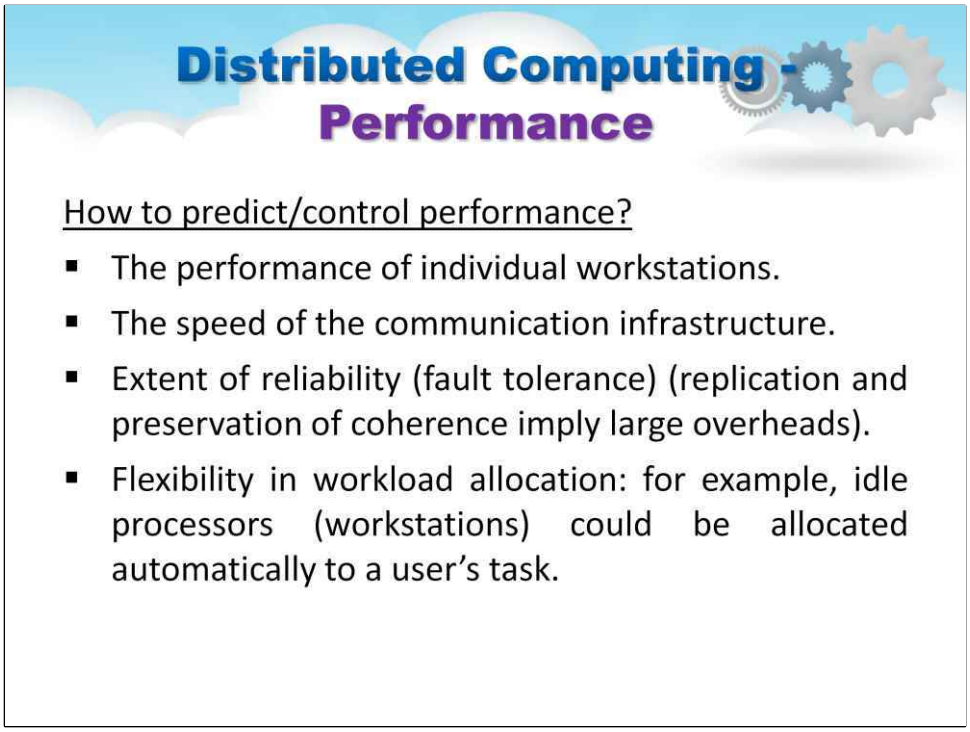
Масштабованість системи можна виміряти принаймні за трьома різними вимірами (Neuman, 1994):

1. По-перше, система може бути масштабованою щодо свого розміру, тобто ми можемо легко додати більше користувачів і ресурсів до системи.

2. По-друге, географічно масштабована система – це система, в якій користувачі та ресурси можуть лежати далеко один від одного.

3. По-третє, система може бути адміністративно масштабованою, тобто нею можна легко керувати, навіть якщо вона охоплює багато незалежних адміністративних організацій.

На жаль, система, яка масштабується в одному або кількох із цих вимірів, часто демонструє деяку втрату продуктивності під час масштабування системи.



Як передбачити/контролювати продуктивність?

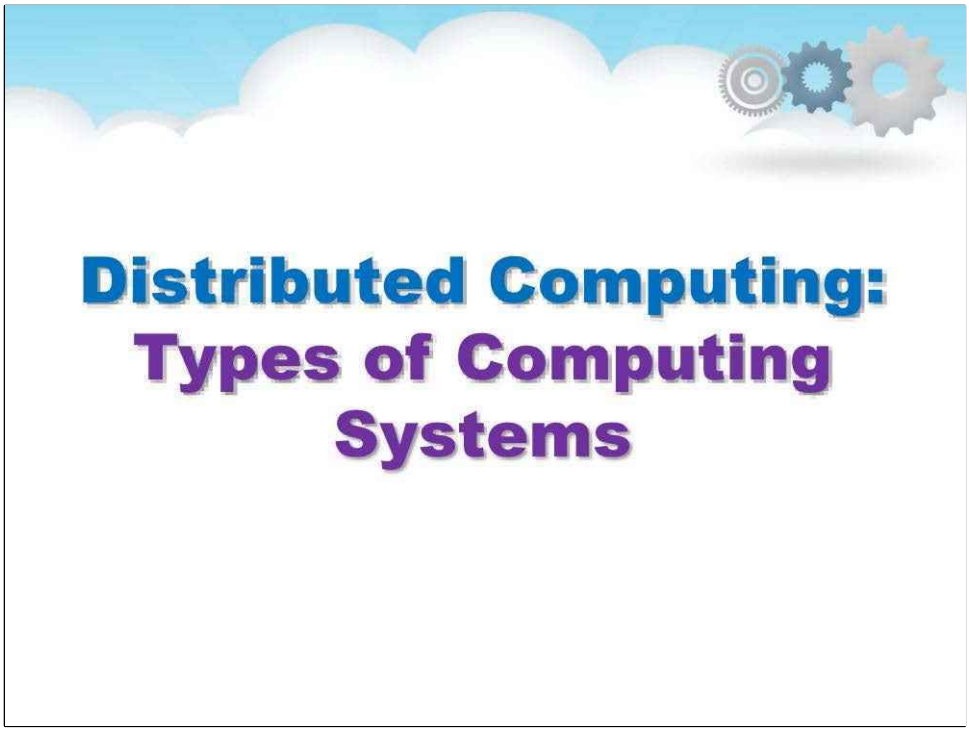
- Продуктивність окремих АРМ.
- Швидкість комунікаційної інфраструктури.
- Рівень надійності (відмовостійкість) (тиражування та збереження узгодженості потребують великих накладних витрат).
- Гнучкість у розподілі робочого навантаження: наприклад, неактивні процесори (робочі станції) можуть бути автоматично розподілені для завдання користувача.

Distributed Computing - Heterogeneity

- different hardware: mainframes, workstations, PCs, servers, etc.;
- different software: UNIX, Windows, OS/2, iOS, Android, Tizen, Real-time OSs, etc.;
- various devices: PCs, mobiles, ATM-machines, telephone switches, robots, sensors, etc.;
- diverse networks and protocols: Ethernet, FDDI, ATM, TCP/IP, Novell Netware, etc.

Distributed Computing - Applications

- Strategic Systems (Defence / Intelligence)
- Visualization and Graphics
- Economics and Finance
- Scientific Computing
  - Physics (LHC – Higgs boson!)
  - Bioinformatics (protein docking)
  - Geology (seismography)
  - Astronomy (simulation of galaxies)



Типи обчислювальних систем

Перш ніж почати обговорювати принципи розподілених систем, давайте спершу детальніше розглянемо різні типи розподілених систем. Далі ми розрізняємо розподілені обчислювальні системи, розподілені інформаційні системи та розподілені вбудовані системи.

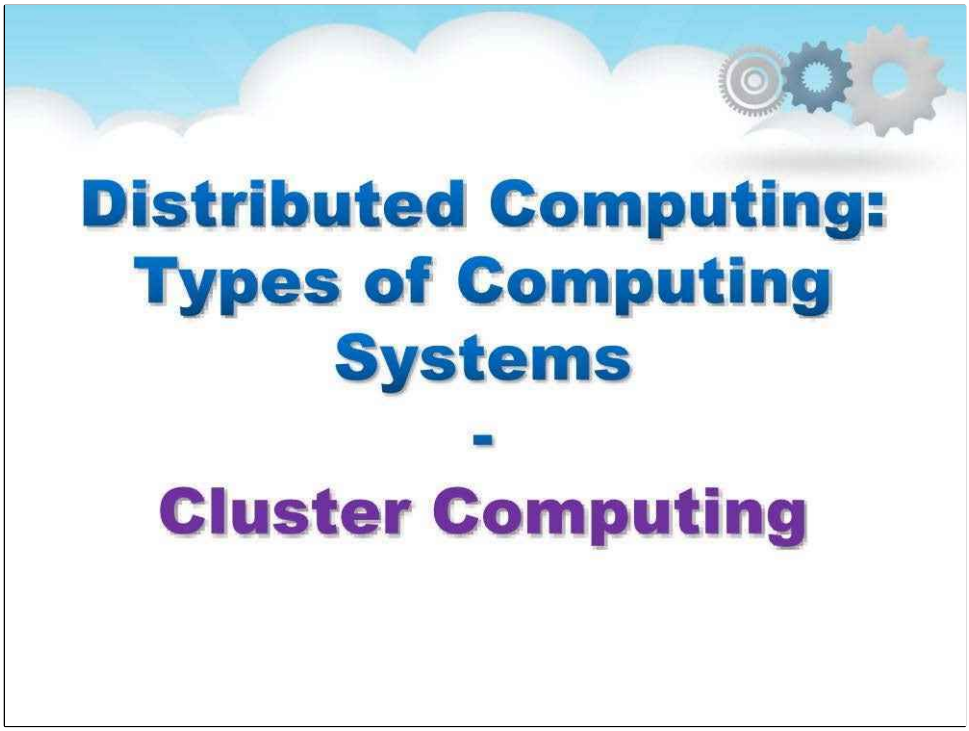
Можна зробити дуже грубе розмежування між двома підгрупами.

Тісно з’єднані комп’ютери:

наприклад, у кластерних обчисленнях базове обладнання складається з набору схожих робочих станцій або ПК, тісно пов’язаних за допомогою високошвидкісної локальної мережі, де на кожному вузлі працює одна операційна система.

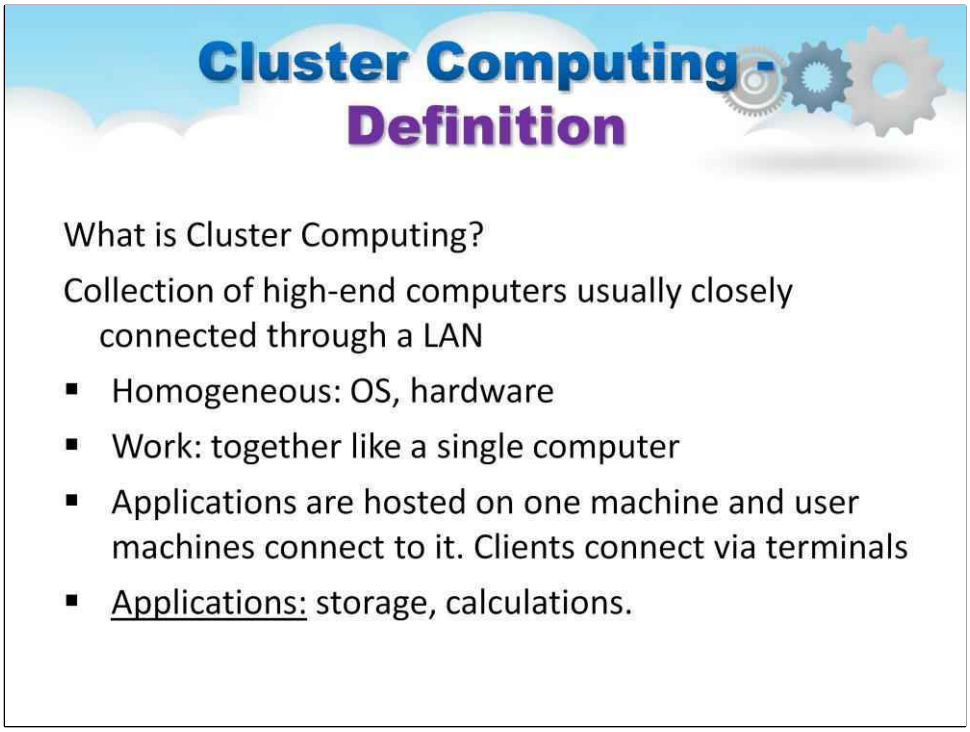
Слабо підключені комп’ютери:

наприклад, у мережевих обчисленнях базове апаратне забезпечення складається з об’єднання комп’ютерних систем, де кожна система може підпадати під окремий адміністративний домен і може сильно відрізнятися, коли йдеться про апаратне забезпечення, програмне забезпечення та розгорнуту мережеву технологію.

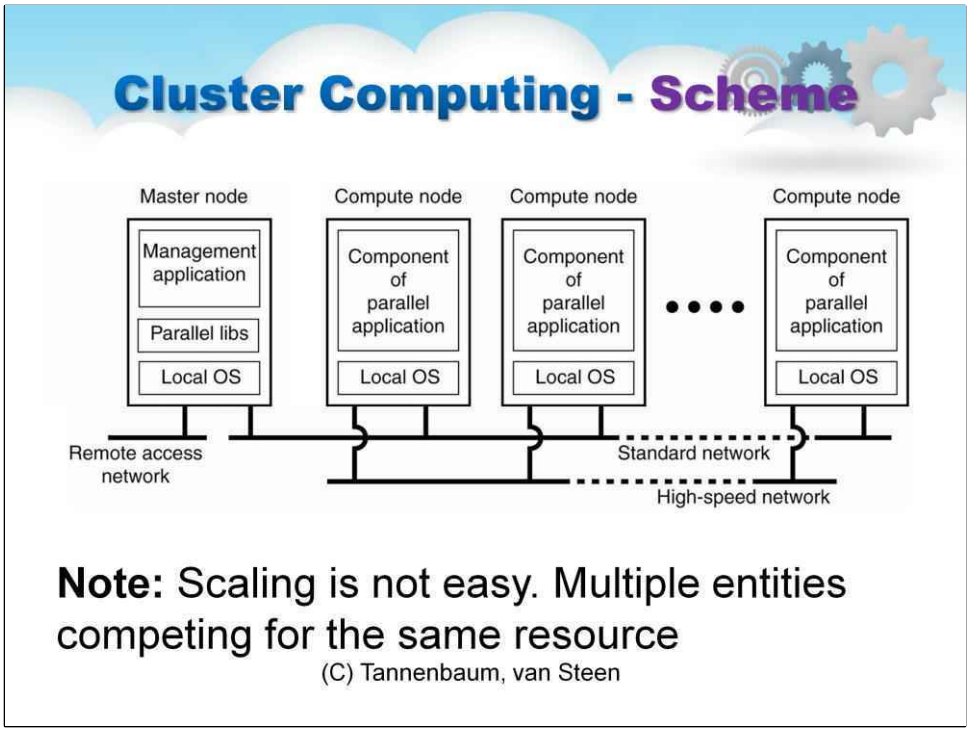


Кластерні обчислення

Давайте розглянемо парадигму кластерних обчислень...



Кластерні обчислювальні системи стали популярними, коли покращилося співвідношення ціна/ продуктивність персональних комп'ютерів і робочих станцій. У певний момент стало фінансово та технічно привабливим створити суперкомп'ютер з використанням стандартних технологій, просто об'єднавши набір відносно простих комп'ютерів у високошвидкісну мережу. Практично у всіх випадках кластерні обчислення використовуються для паралельного програмування, у якому одна (інтенсивна обчислювальна) програма виконується паралельно на кількох машинах.



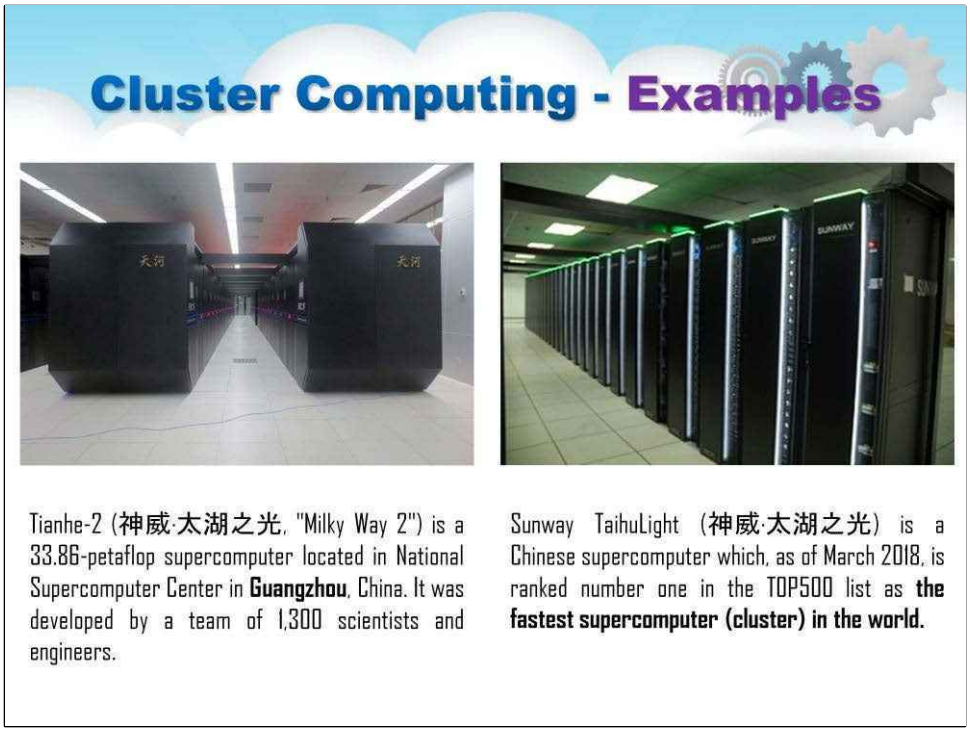
Типовий приклад кластерного комп'ютера показаний на цьому малюнку.

Кожен кластер складається з набору обчислювальних вузлів, які контролюються та доступні за допомогою одного головного вузла.

Головний вузол зазвичай обробляє розподіл вузлів для певної паралельної програми, підтримує пакетну чергу надісланих завдань і забезпечує інтерфейс для користувачів системи.

На головному вузлі фактично працює проміжне програмне забезпечення, тобто програмне забезпечення, необхідне для виконання програм і керування кластером, тоді як обчислювальним вузлам часто не потрібно нічого, крім стандартної операційної системи.

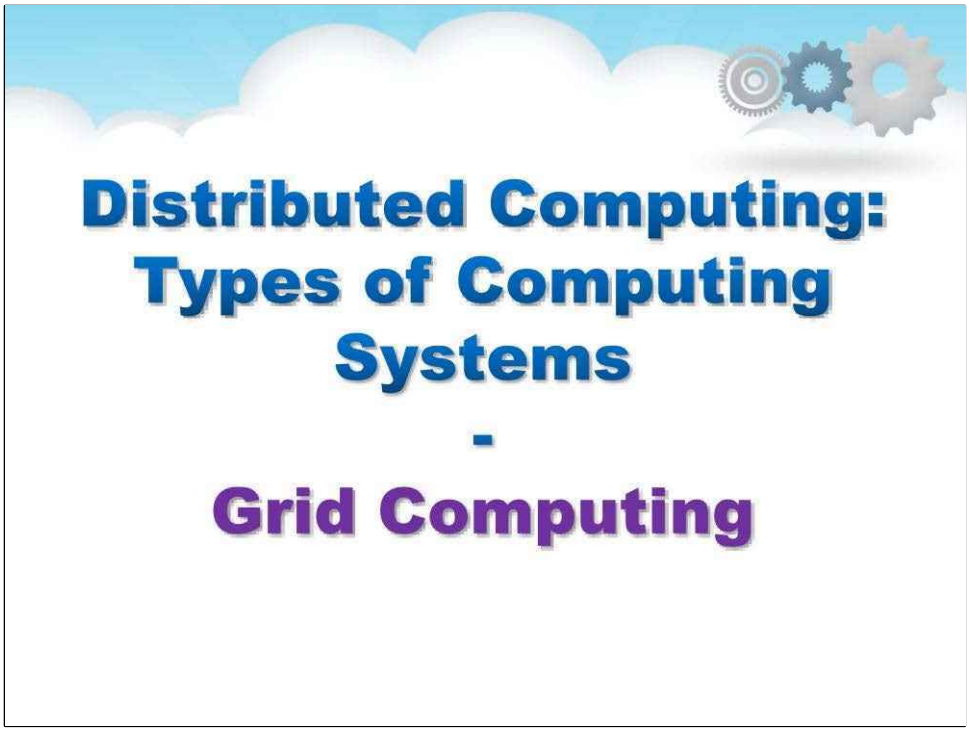
Слід зазначити, що важлива частина цього проміжного програмного забезпечення базується на бібліотеках для виконання паралельних програм, які ефективно забезпечують лише розширені засоби зв’язку на основі повідомлень, але не здатні обробляти несправні процеси, захист тощо.



Tianhe-2 або TH-2 («Чумацький Шлях 2») — це суперкомп'ютер із продуктивністю 33,86 петафлоп, розташований у Національному суперкомп'ютерному центрі в Гуанчжоу, Китай. Його розробила команда з 1300 вчених та інженерів.

Розробка Tianhe-2 була спонсорована Програмою високих технологій 863, ініційованою урядом Китаю, урядом провінції Гуандун і урядом міста Гуанчжоу. Він був створений Китайським національним університетом оборонних технологій (NUDT) у співпраці з китайською IT-компанією Inspur. Inspur виготовив друковані плати та допоміг із встановленням і тестуванням системного програмного забезпечення. Це був найшвидший суперкомп'ютер у світі згідно зі списками TOP500 з 2013 по 2015 рік. Рекорд був побитий у червні 2016 року Sunway TaihuLight.

The Sunway TaihuLight (китайська: 神威 · 太湖之光 ,Shénwēi·tàihú zhī guāng) — китайський суперкомп'ютер, який станом на березень 2018 року займає перше місце в списку TOP500 як найшвидший суперкомп'ютер (кластер) у світі з рейтингом LINPACK у 93 петафлопс. Це майже втричі швидше, ніж попередній рекордсмен Tianhe-2, який працював зі швидкістю 34 петафлопс.
Він був розроблений Національним дослідницьким центром паралельної комп'ютерної техніки та технологій (NRCPC) і розташований у Національному суперкомп'ютерному центрі в Усі в місті Усі, провінція Цзянсу, Китай.

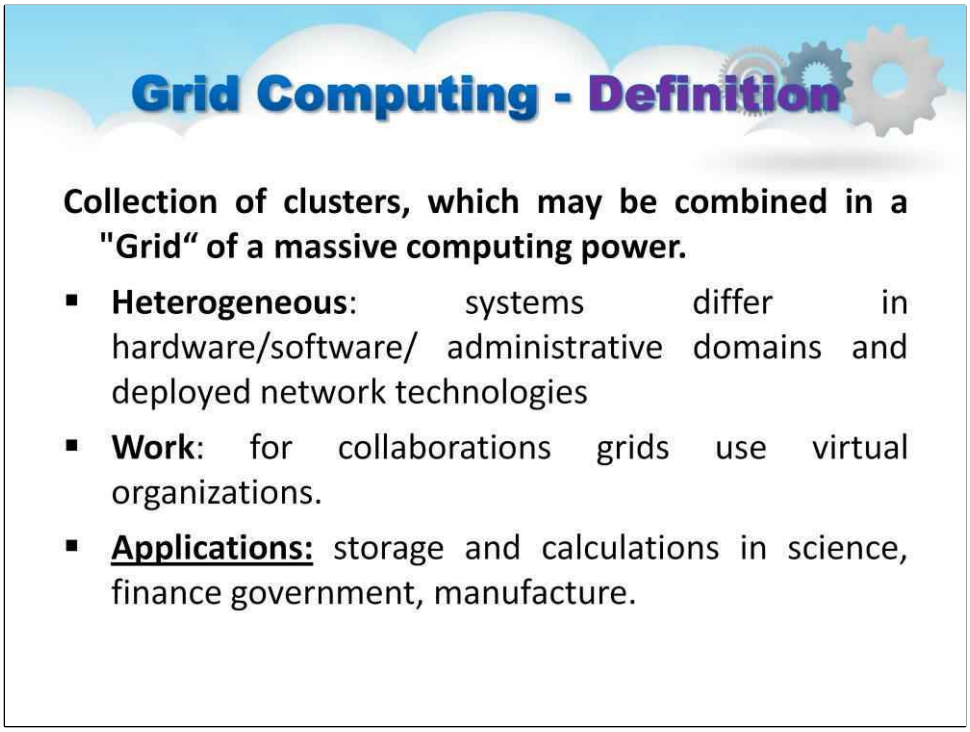


Грід-обчислення

Характерною особливістю кластерних обчислень є їх однорідність.

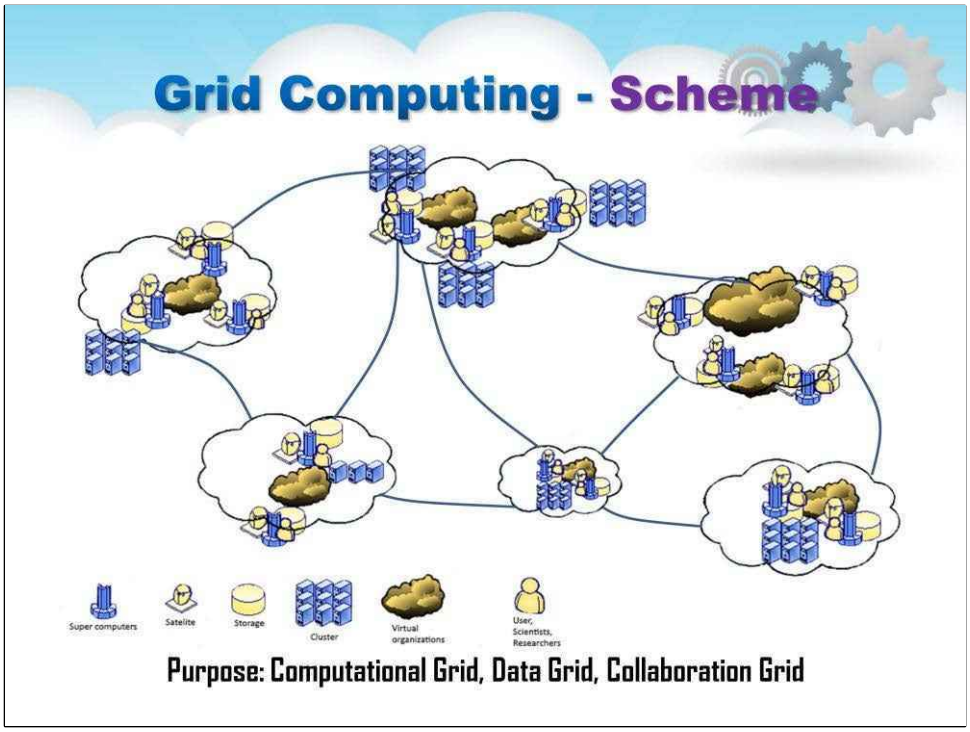
У більшості випадків комп’ютери в кластері здебільшого однакові, усі вони мають однакову операційну систему та під’єднані через одну мережу.

Навпаки, грід-обчислювальні системи мають високий ступінь неоднорідності: не робиться жодних припущень щодо апаратного забезпечення, операційних систем, мереж, адміністративних доменів, політики безпеки тощо.



Ключовою проблемою грид-обчислювальної системи є те, що ресурси різних організацій об'єднуються разом, щоб забезпечити можливість співпраці групи людей або установ. Така співпраця реалізується у формі віртуальної організації.

Люди, які належать до однієї віртуальної організації, мають права доступу до ресурсів, які надаються цій організації. Як правило, ресурси складаються з обчислювальних серверів (включаючи суперкомп'ютери, можливо реалізовані як кластерні комп'ютери), сховищ і баз даних. Крім того, можуть бути надані спеціальні мережеві пристрої, такі як телескопи, датчики тощо.

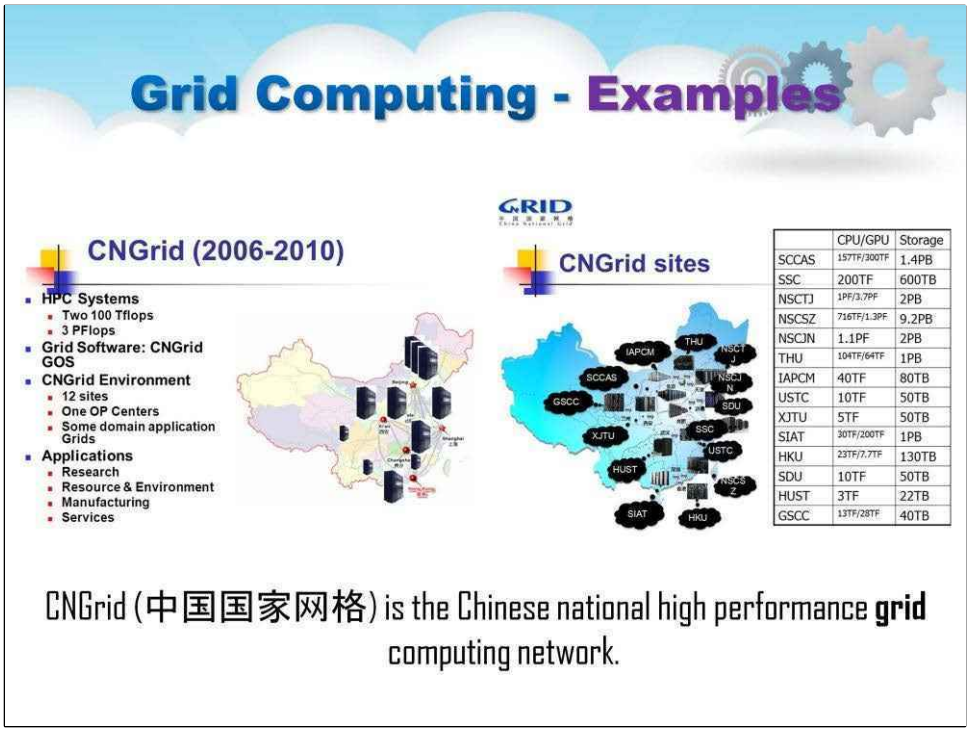


Типи сіток:

Обчислювальна сітка: спільні обчислювальні ресурси

Data Grid: доступ до великої кількості даних, розповсюджених на різних сайтах

Collaboration Grid: кілька систем співпраці для спільної роботи над спільною проблемою



CNGrid (китайська: 中国国家网格)це китайська національна високопродуктивна обчислювальна мережа, яка підтримується програмою 863.

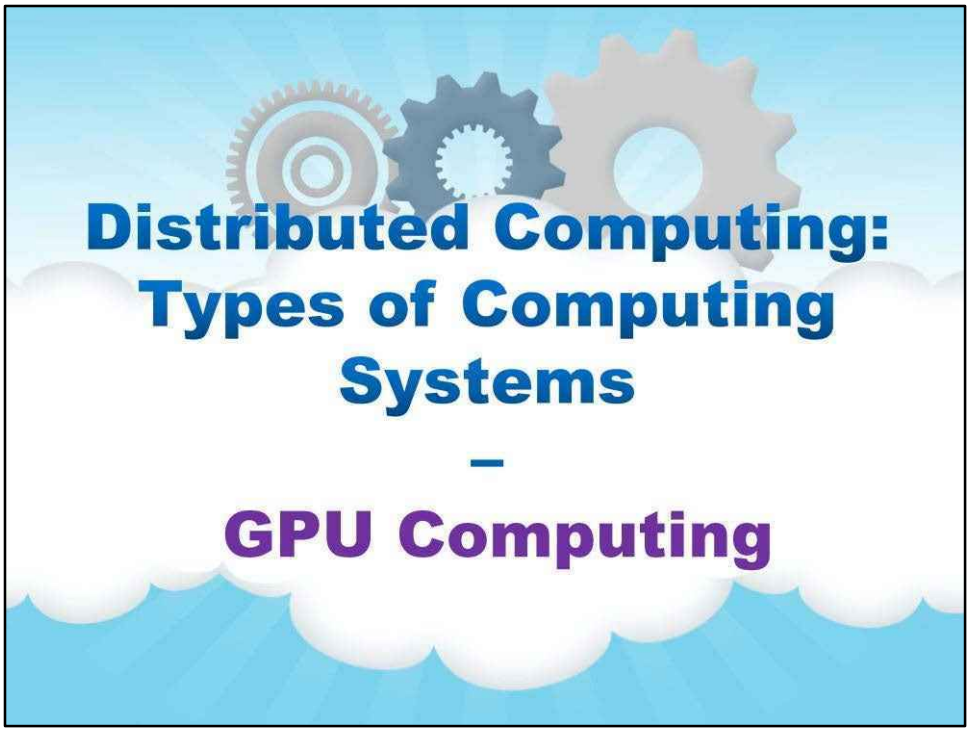
Китайська національна електромережа (CNGrid) — це великий проект, який підтримується Програмою досліджень і розвитку високих технологій (863) Китаю. CNGrid — це тестовий стенд нового покоління інформаційної інфраструктури, що об’єднує високопродуктивні обчислення та транзакції

можливості обробки. Завдяки спільному використанню ресурсів, узгодженій роботі та механізму обслуговування CNGrid ефективно підтримує багато додатків, таких як наукові дослідження, середовище ресурсів, передове виробництво та інформаційні послуги.

CNGrid сприяє побудові національної інформаційної індустрії та розвитку суміжних галузей шляхом технологічних інновацій. Програмне забезпечення China National Grid під назвою CNGrid GOS — це набір програмного забезпечення для мереж із незалежною інтелектуальною власністю, розроблене командою проекту CNGrid Software R&D.

CNGrid GOS в основному включає системне програмне забезпечення, систему керування сертифікатами CA та середовище тестування, три бізнес-версії підсистем (висока продуктивність

обчислювальний шлюз, сітка даних і робочий процес сітки) і система моніторингу. Цей проект здійснюється сімома організаціями, включаючи Інститут обчислювальної техніки Академії наук Китаю, Інститут обчислювальної технології Цзяньнань, Університет Цінхуа, Національний університет оборонних технологій, Університет Бейхан, Інформаційний центр комп’ютерних мереж Академії наук Китаю та Шанхайський суперкомп’ютерний центр.

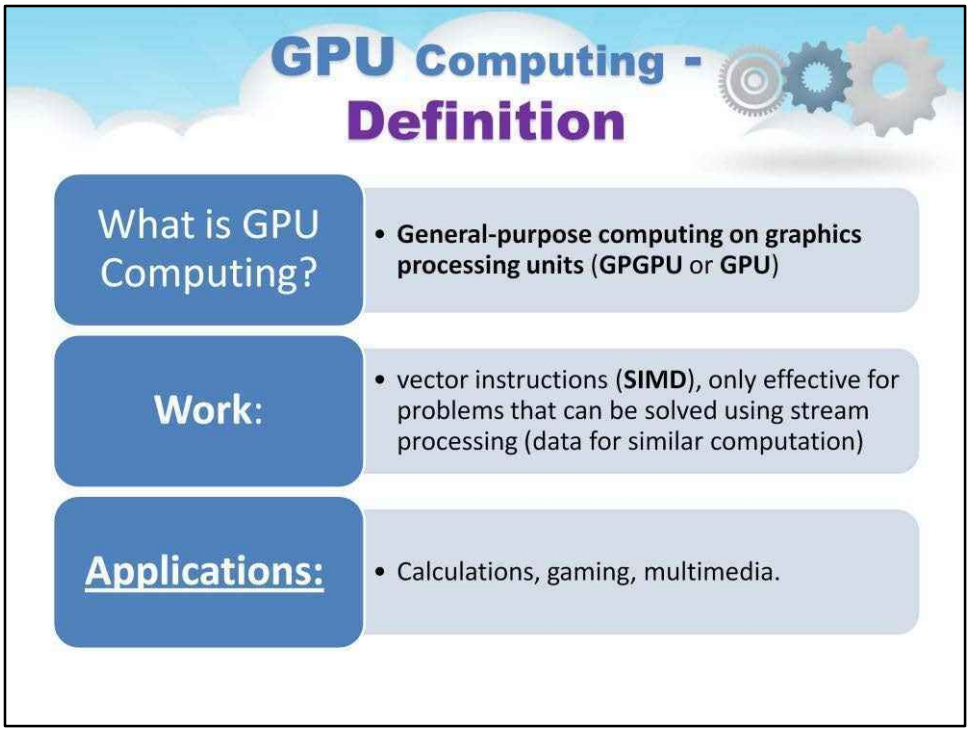


Обчислення GPU

GPU — це графічний співпроцесор або прискорювач, встановлений на графічній або відеокарті комп’ютера. Графічний процесор розвантажує ЦП від виснажливих графічних завдань у програмах для редагування відео.

Перший у світі графічний процесор GeForce 256 був випущений на ринок компанією NVIDIA в 1999 році. Ці чіпи GPU можуть обробляти щонайменше 10 мільйонів багатокутників за секунду та використовуються майже в кожному комп’ютері на ринку сьогодні. Деякі функції GPU також були інтегровані в певні процесори.

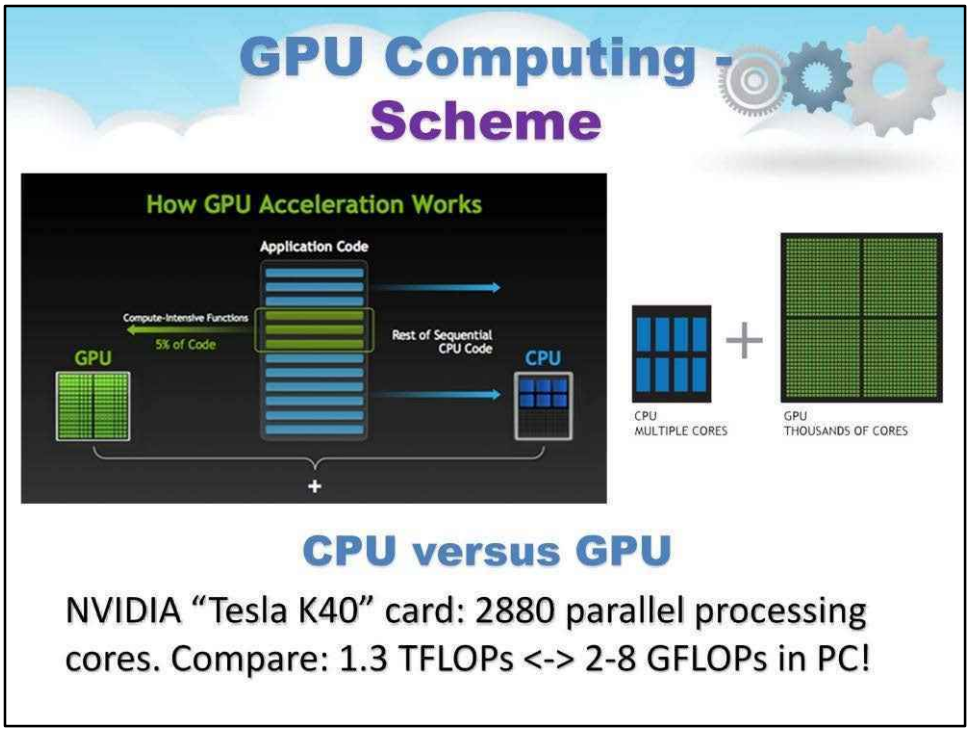
Традиційні процесори мають лише кілька ядер. Наприклад, процесор Xeon X5670 має шість ядер. Однак сучасний графічний чіп можна створити з тисячами процесорних ядер.



SIMD - чому він поширюється? – незалежно від центрального процесора, кілька графічних карт можуть бути інтегровані в ПК, кластери тощо

На відміну від ЦП, графічні процесори мають пропускну архітектуру, яка використовує масовий паралелізм, повільно виконуючи багато одночасних потоків замість того, щоб дуже швидко виконувати один довгий потік у звичайному мікропроцесорі. Останнім часом паралельні графічні процесори або кластери графічних процесорів привертають велику увагу проти використання процесорів з обмеженим паралелізмом.

Обчислення загального призначення на GPU, відомі як GPGPU, з’явилися в галузі HPC.



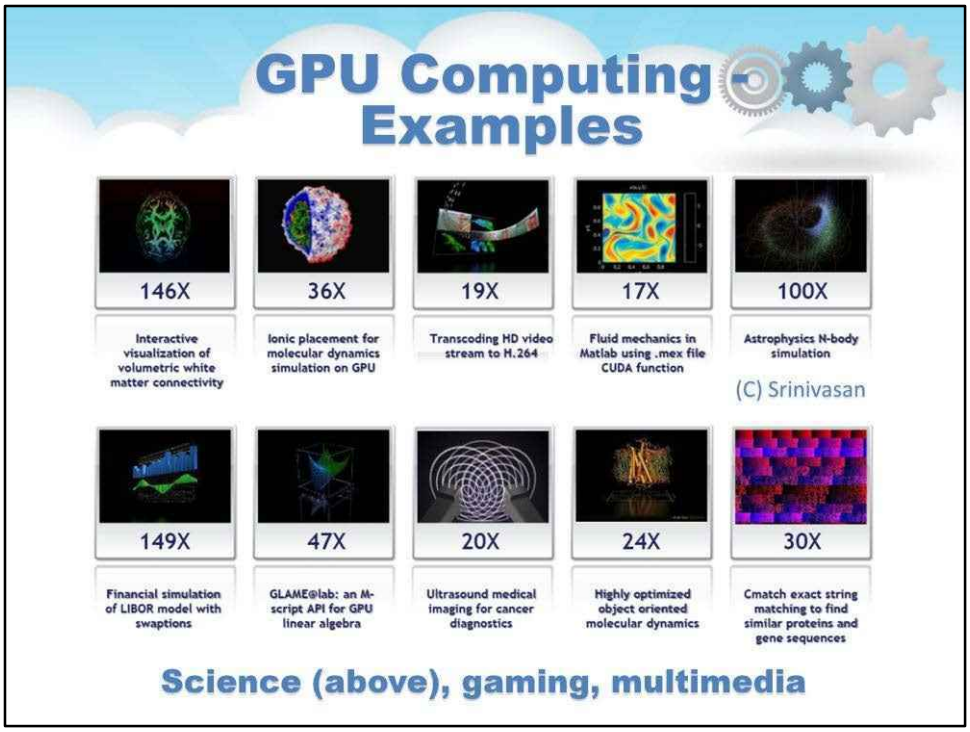
CPU ПРОТИ GPU

ЛІВО:

Простий спосіб зрозуміти різницю між ЦП і ГП — це порівняти, як вони обробляють завдання. Центральний процесор складається з кількох ядер, оптимізованих для послідовної обробки, тоді як графічний процесор складається з тисяч менших, ефективніших ядер, призначених для виконання кількох завдань одночасно.

ПРАВО:

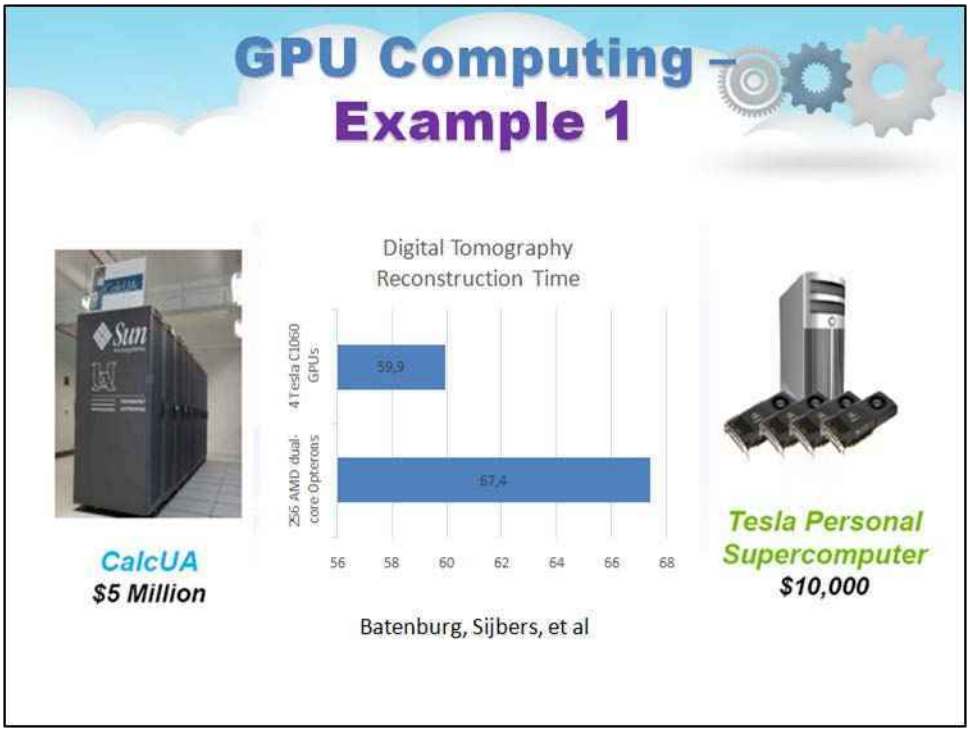
Графічні процесори мають тисячі ядер для ефективної обробки паралельних навантажень



Сучасні графічні процесори не обмежуються прискореною графікою чи кодуванням відео. Вони використовуються в системах HPC для живлення суперкомп'ютерів із масивним паралелізмом на багатоядерному та багатопоточному рівнях.

Графічні процесори розроблені для паралельної обробки великої кількості операцій із плаваючою комою. Певним чином графічний процесор розвантажує центральний процесор від усіх обчислень, що містять інтенсивні дані, а не лише від тих, які пов'язані з обробкою відео. Звичайні графічні процесори широко використовуються в мобільних телефонах, ігрових приставках, вбудованих системах, ПК та серверах.

Карти NVIDIA CUDA Tesla, Fermi або Volta використовуються в кластерах GPU або в системах HPC для паралельної обробки масивних даних із плаваючою точкою.



Оскільки сучасні стандартні методи скринінгу часто не дозволяють діагностувати рак молочної залози до розвитку метастазів, рання діагностика раку молочної залози все ще є серйозною проблемою. Тривимірна ультразвукова комп'ютерна томографія обіцяє високоякісні зображення молочної залози, але наразі обмежена трудомісткою реконструкцією зображення.

Медична візуалізація є одним із перших додатків, які використовують переваги графічного процесора для прискорення. Використання графічних процесорів у цій галузі досягло такого рівня, що зараз з графічним процесором Tesla від NVIDIA поставляється кілька медичних модальностей.

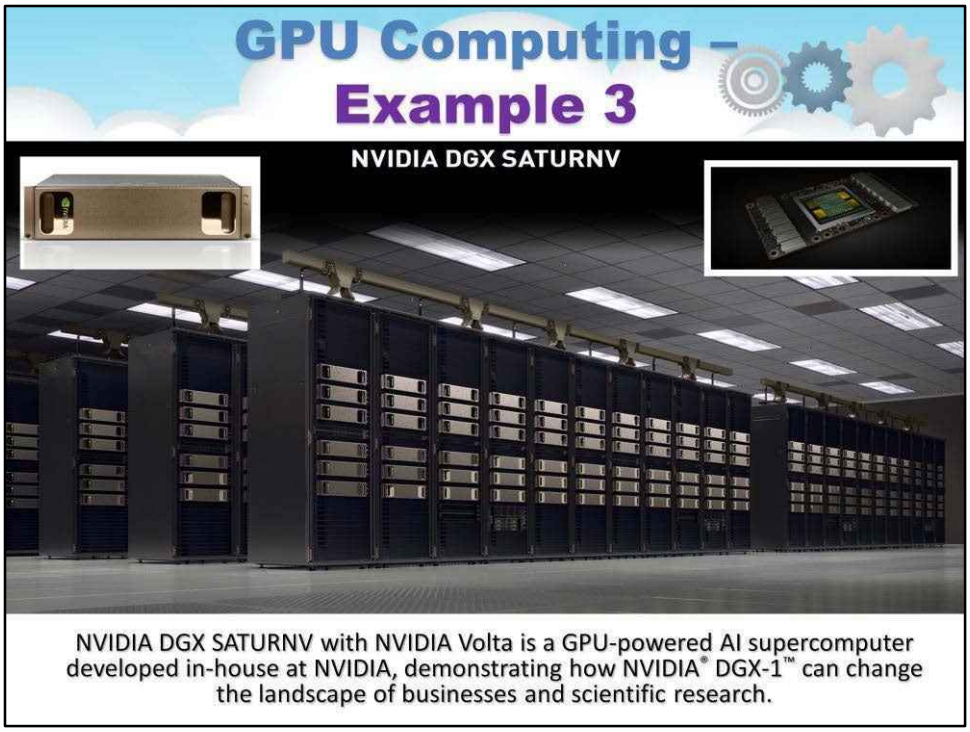
* CalcUA», суперкомп'ютер Антверпенського університету, який у березні 2005 року коштував 3,5 мільйона євро.

Volta використовує технологію високошвидкісного з'єднання NVIDIA NVLink™ нового покоління. Це забезпечує удвічі більшу пропускну здатність порівняно з попереднім поколінням NVLink.

Завдяки понад 21 мільярду транзисторів Volta є найпотужнішою у світі архітектурою графічного процесора, яка поєднує в собі NVIDIA CUDA® і ядра Tensor, що забезпечує продуктивність суперкомп'ютера зі штучним інтелектом у графічному процесорі.

Завдяки оптимізованим для Volta бібліотекам CUDA та NVIDIA Deep Learning SDK, таким як cuDNN, NCCL і TensorRT™, найкращі фреймворки та програми галузі можуть легко використовувати потужність Volta.

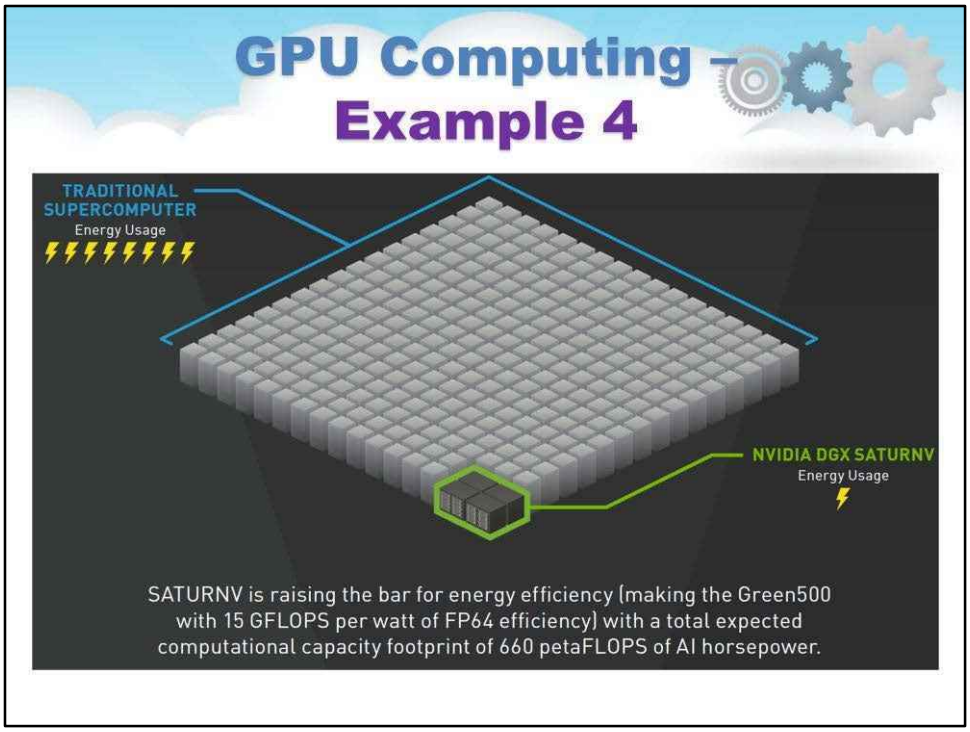
Оснащений 640 тензорними ядрами, Volta забезпечує продуктивність глибокого навчання понад 100 тераФЛОПС, що в 5 разів більше порівняно з архітектурою NVIDIA Pascal™ попереднього покоління.



NVIDIA DGX SATURNV базується на графічних процесорах NVIDIA TESLA® V100 і побудована на новітній архітектурі NVIDIA Volta GPU.

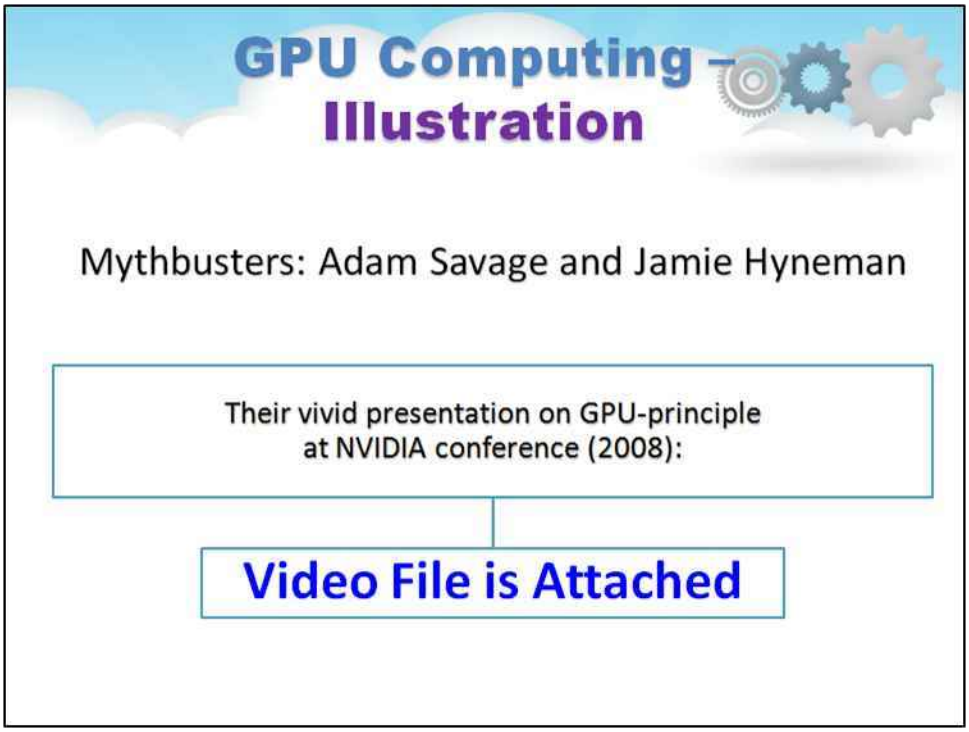
з NVIDIA Volta — це суперкомп’ютер зі штучним інтелектом на базі GPU, розроблений NVIDIA, який демонструє, як NVIDIA DGX-1™ може змінити ландшафт бізнесу та наукових досліджень.

Можливості для компаній, прискорених штучним інтелектом, зростають, оскільки світ починає покладатися на все більш складні продукти та послуги для створення кращого досвіду клієнтів, трансформації ланцюжків поставок і покращення якості продукції. NVIDIA лідирує в індустрії обчислень зі штучним інтелектом, а SATURNV створено для створення місячних знімків, що дає змогу найгеніальнішим умам світу виконувати роботу всього свого життя.



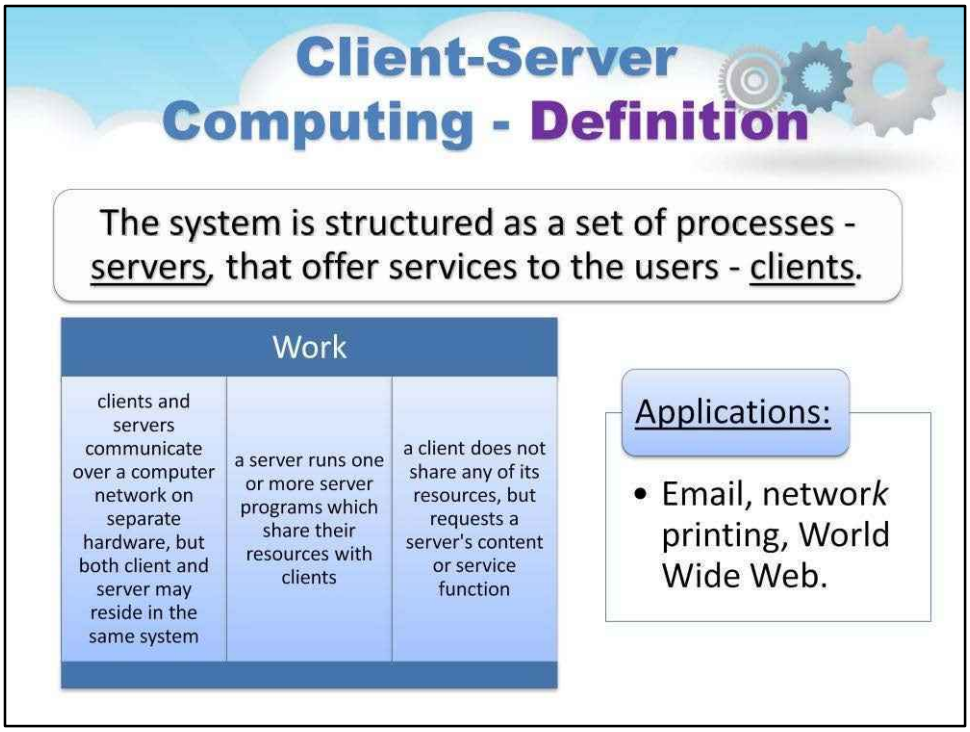
GPU Computing дуже енергоефективні.

Наприклад, DGX SATURNV піднімає планку енергоефективності (роблячи Green500 із 15 GFLOPS на ват ефективності FP64) із загальною очікуваною обчислювальна потужність 660 петаФЛОПС сил ШІ.



Давайте подивимося це відео з яскравою демонстрацією різниці між процесором і GPU.

Клієнт-серверні обчислення



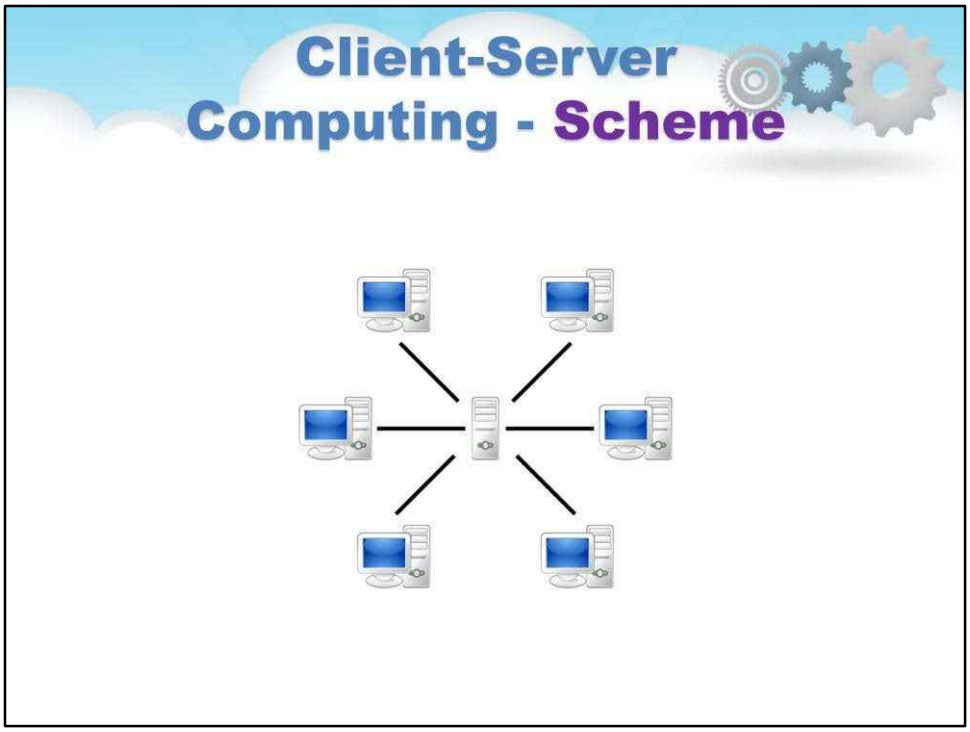
Прикладом добре налагодженої розподіленої системи є архітектура клієнт-сервер. У цьому сценарії клієнтські машини (ПК і робочі станції) підключаються до центрального сервера для обчислень, електронної пошти, доступу до файлів і додатків бази даних.

Система структурована як набір процесів -серверів, які пропонують послуги користувачам –клієнтів

- клієнти та сервери спілкуються через комп'ютерну мережу на окремому обладнанні, але і клієнт, і сервер можуть перебувати в одній системі
- сервер запускає одну або кілька серверних програм, які спільно використовують свої ресурси з клієнтами
- клієнт не надає спільний доступ до жодних своїх ресурсів, але запитує вміст або сервісну функцію сервера

Застосування:

Електронна пошта, мережа друк, Всесвітня павутина.



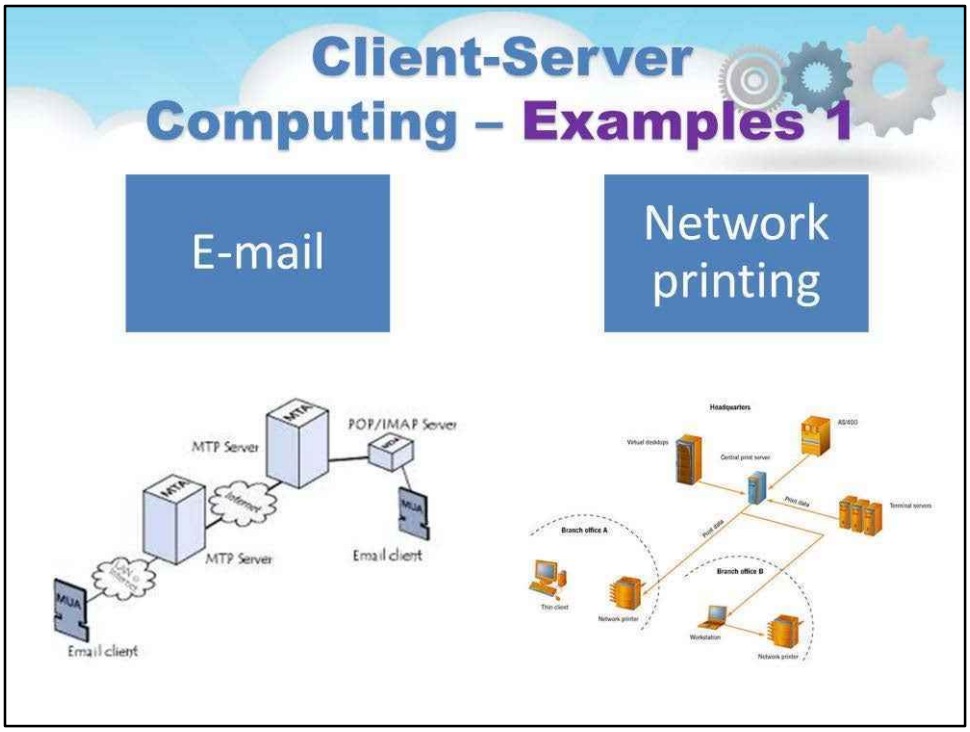
Система структурована як набір процесів - серверів, які пропонують послуги користувачам - клієнтам.

робота:

- клієнти та сервери спілкуються через комп'ютерну мережу на окремому обладнанні, але і клієнт, і сервер можуть знаходитися в одній системі

- сервер запускає одну або більше серверних програм, які спільно використовують свої ресурси з клієнтами

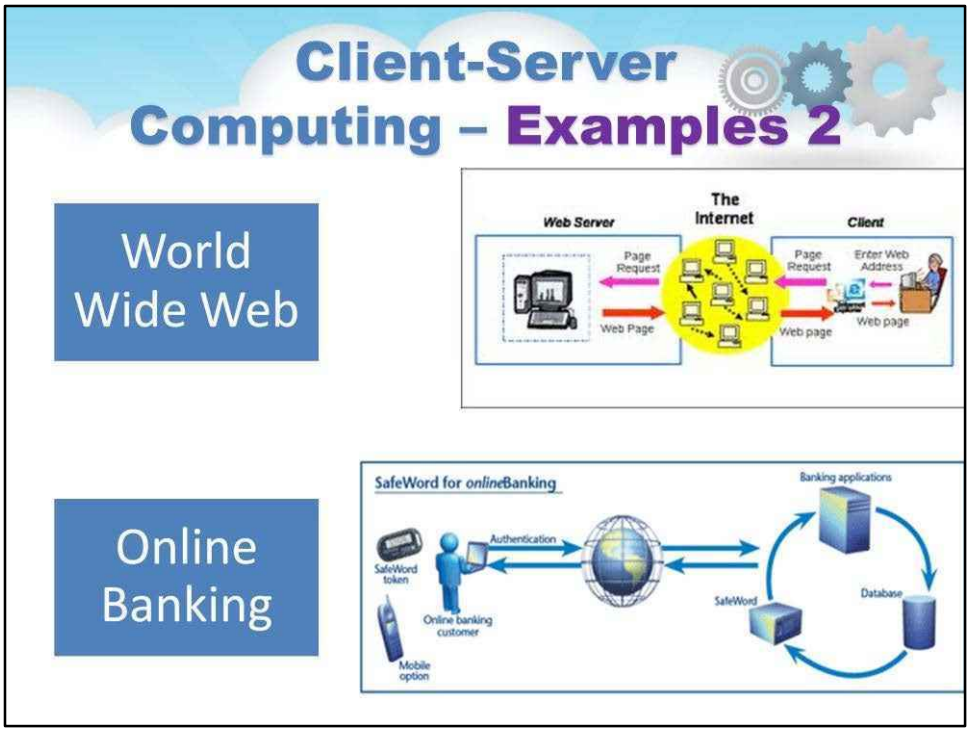
- клієнт не надає спільний доступ до будь-яких своїх ресурсів, але запитує вміст або сервісні функції сервера



Тут ви можете побачити кілька прикладів найпопулярніших клієнт-серверних систем:

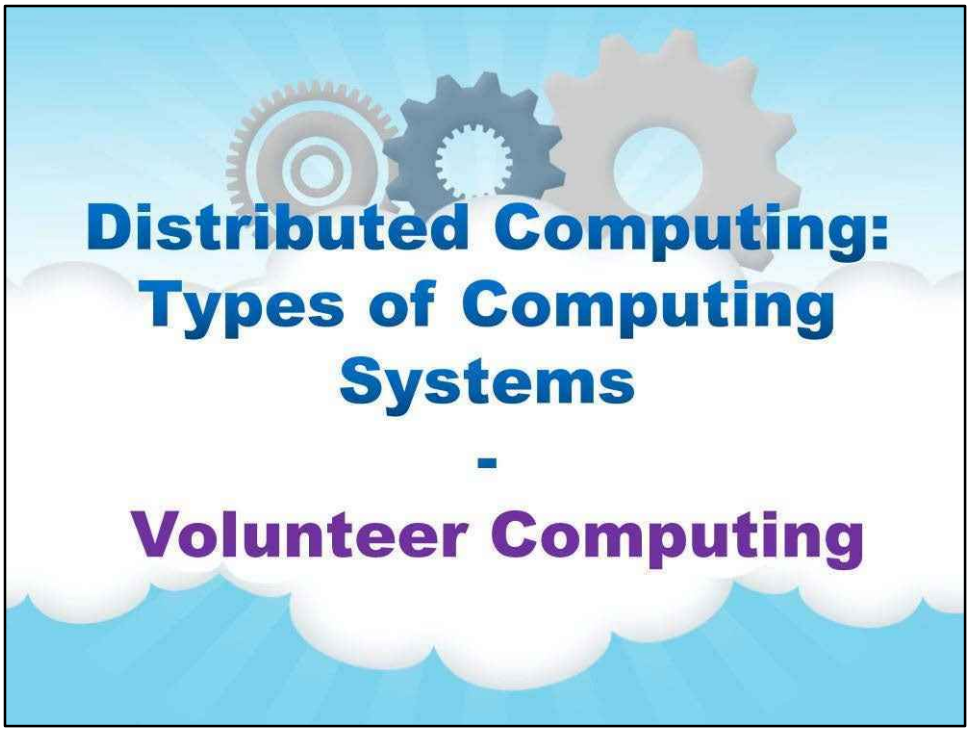
Електронна пошта, яку найчастіше називають електронною поштою або електронною поштою з бл. 1993, це метод обміну цифровими повідомленнями від автора до одного або кількох одержувачів. Сучасна електронна пошта працює через Інтернет або інші комп'ютерні мережі. Деякі перші системи електронної пошти вимагали, щоб і автор, і одержувач були в режимі онлайн одночасно, як і для обміну миттєвими повідомленнями. Сучасні системи електронної пошти засновані на моделі storeand-forward. Сервери електронної пошти приймають, пересилають, доставляють і зберігають повідомлення. Ні користувачі, ні їхні комп'ютери не зобов'язані одночасно бути онлайн; їм потрібно лише ненадовго підключатися, як правило, до поштового сервера, стільки часу, скільки потрібно для надсилання чи отримання повідомлень.

Мережевий друк: сервер друку або сервер принтера – це пристрій, який підключає принтери до клієнтських комп'ютерів через мережу. Він приймає завдання друку від комп'ютерів і надсилає їх на відповідні принтери, ставлячи завдання локально в чергу, щоб врахувати той факт, що робота може надходити швидше, ніж принтер може її впоратися. Допоміжні функції включають можливість перевіряти чергу завдань, які потрібно обробити, можливість змінювати порядок або видаляти завдання друку, що очікують, або здатність виконувати різні види обліку (наприклад, підрахунок сторінок принтера, який може передбачати зчитування даних, створених принтером). (s)).

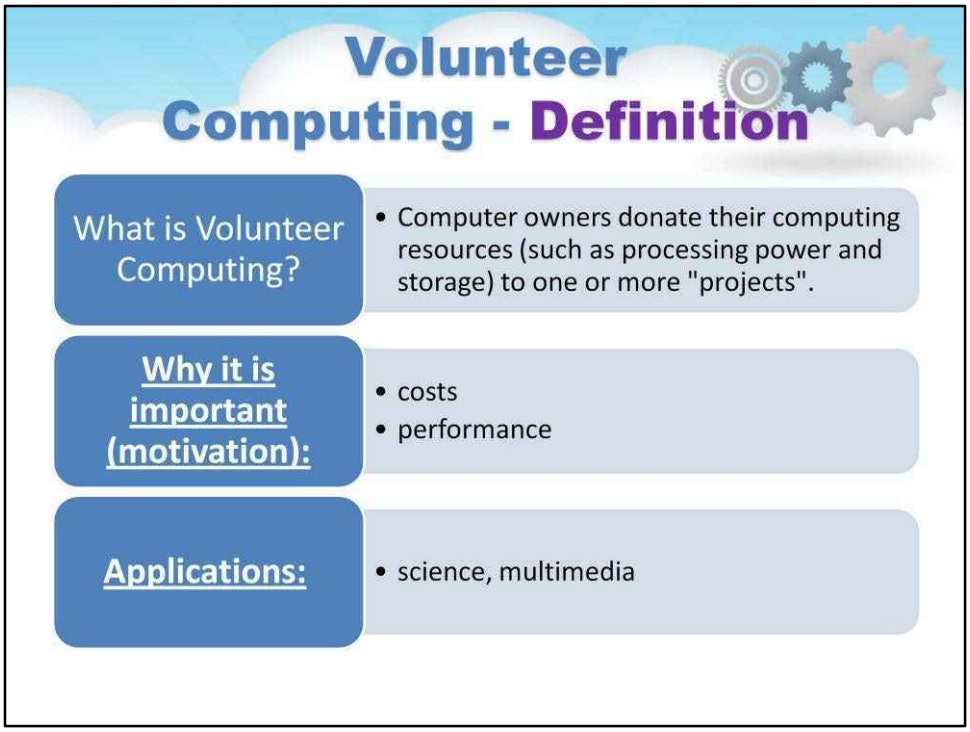


Всесвітня павутина – це система взаємопов’язаних гіпертекстових документів, доступ до яких здійснюється через Інтернет. За допомогою веб-браузера можна переглядати веб-сторінки, які можуть містити текст, зображення, відео та інші мультимедіа, і переходити між ними за допомогою гіперпосилань.

Онлайн-банкінг, також відомий як інтернет-банкінг, це електронна платіжна система, яка дозволяє клієнтам банку чи іншої фінансової установи проводити низку фінансових операцій через веб-сайт фінансової установи. Система онлайн-банкінгу, як правило, підключається до основної банківської системи, якою керує банк, або є її частиною, і вона відрізняється від банківських послуг у відділеннях, які були традиційним способом доступу клієнтів до банківських послуг.



Волонтерський комп’ютер



Що таке волонтерський комп’ютер?

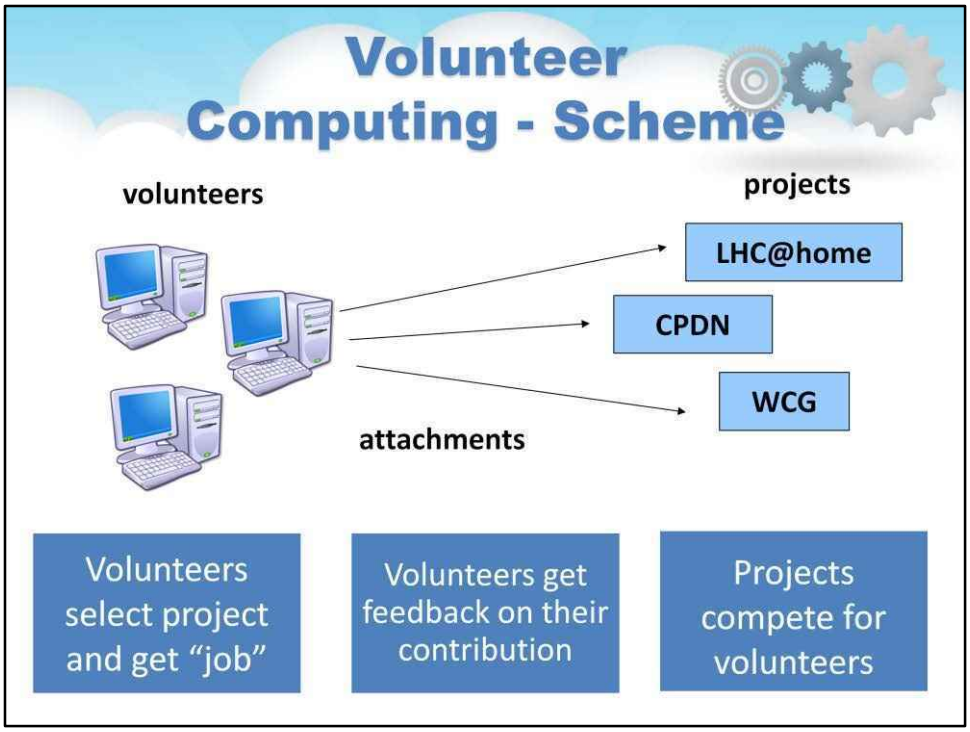
Власники комп’ютерів жертвують своїми обчислювальними ресурсами (такими як обчислювальна потужність і пам’ять) одному або кільком «проектам».

Чому це важливо (мотивація):

- низькі витрати
- висока продуктивність

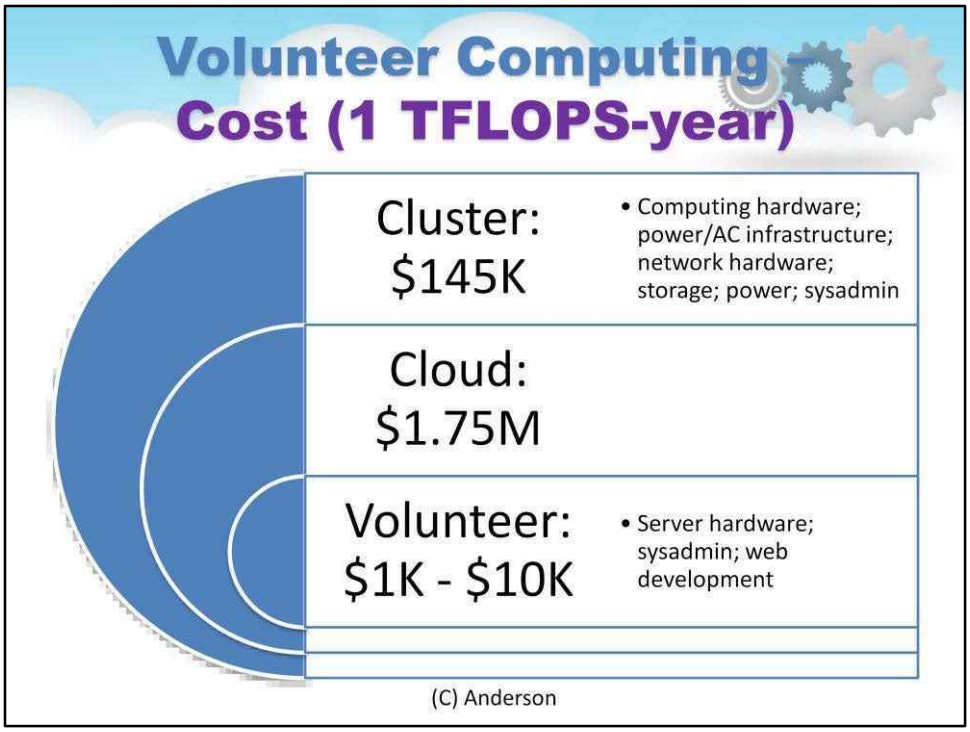
Застосування:

наука, мультимедіа



Основний робочий процес виглядає так:

- Волонтери обирають проект і отримують «роботу»
- Волонтери отримують відгуки про свій внесок
- Проекти змагаються за волонтерів



Типовий бюджет будівництва та експлуатації різних інфраструктур:

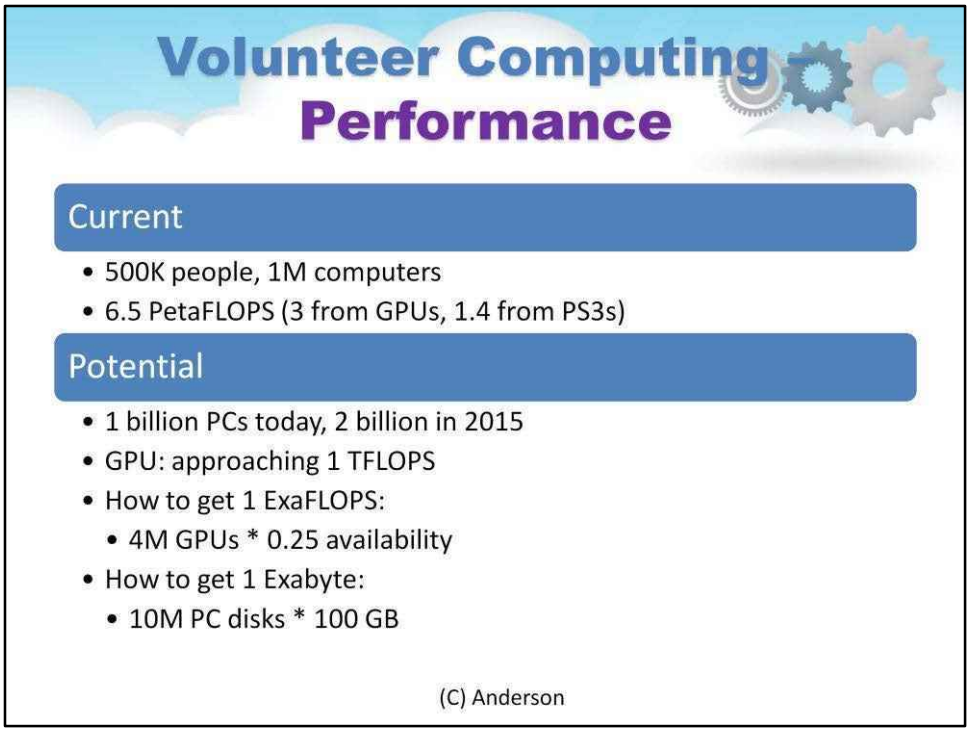
Кластер: $145 тис

Обчислювальне обладнання; інфраструктура живлення/змінного струму; мережеве обладнання; зберігання; потужність; системний адміністратор

Хмара: $1,75 млн

Волонтер: $1K - $10K

Серверне обладнання; системний адміністратор; веб-розробка



Сумарна продуктивність усіх добровільних обчислювальних систем така:

поточний

500 тисяч людей, 1 мільйон комп'ютерів

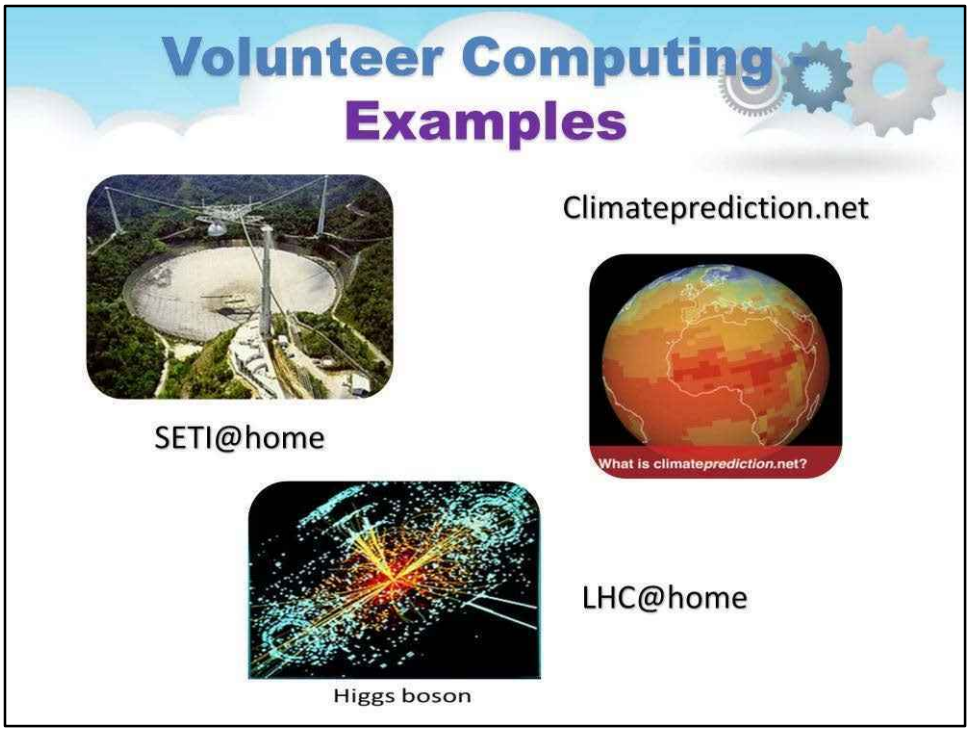
6,5 PetaFLOPS (3 з GPU, 1,4 з PS3)

потенціал

1 мільярд комп'ютерів сьогодні, 2 мільярди у 2015 році GPU: наближається до 1 TFLOPS

Як отримати 1 ExaFLOPS: 4M GPU * 0,25 доступності

Як отримати 1 ексабайт: 10 МБ дисків на ПК * 100 ГБ



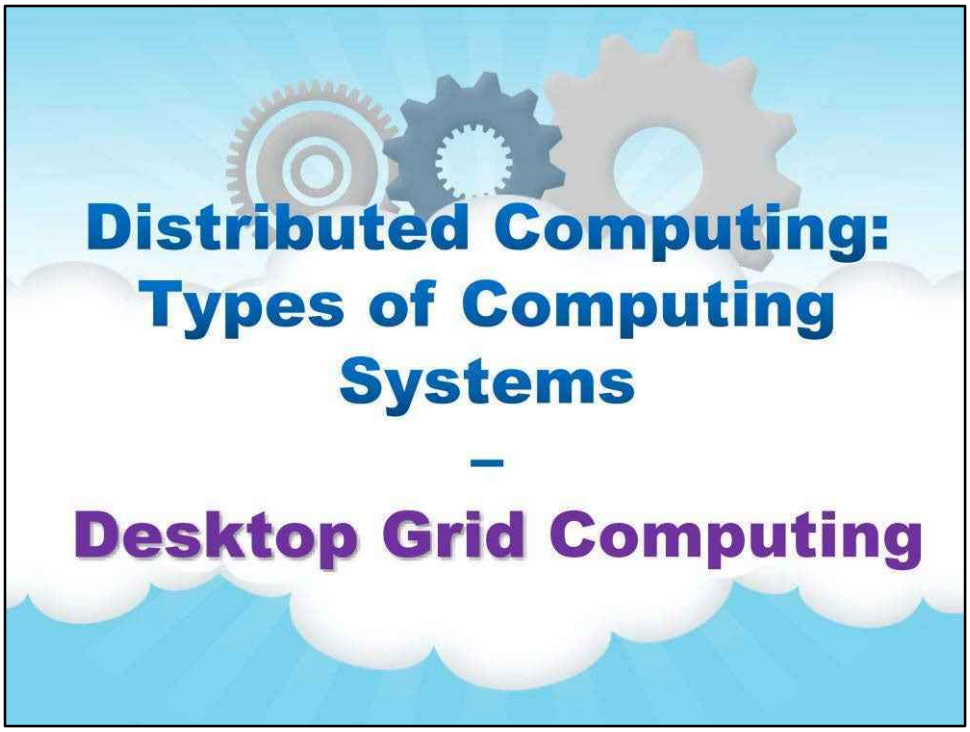
Волонтерський комп’ютер надзвичайно важливий, особливо для проектів з обмеженим фінансуванням.

SETI@home («SETI вдома») — це громадський волонтерський обчислювальний проект в Інтернеті, який використовує програмну платформу BOINC, створену Дослідницьким центром SETI у Берклі та розміщену в Лабораторії космічних наук Каліфорнійського університету в Берклі.

Його метою є аналіз радіосигналів, пошук ознак позаземного розуму, і тому він є одним із багатьох заходів, які проводяться в рамках всесвітньої роботи SETI.

SETI@home є найстарішим і третім широкомасштабним використанням розподілених обчислень в Інтернеті для дослідницьких цілей після Великого Інтернет-пошуку за числом Мерсенна (GIMPS) у 1996 році та distributed.net у 1997 році.

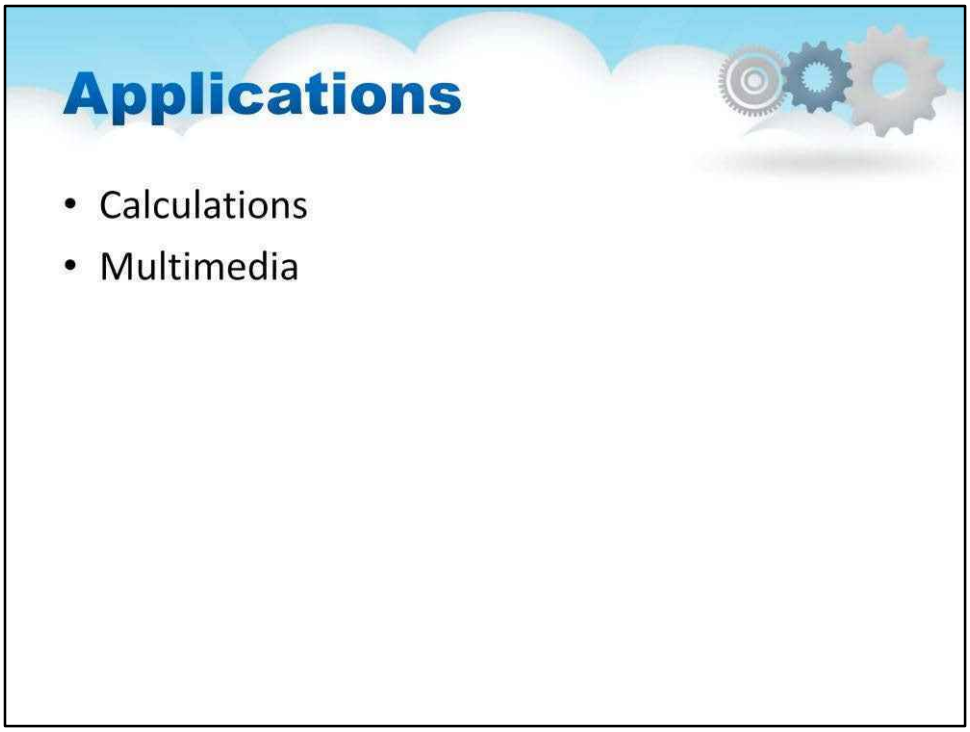
Разом із MilkyWay@home та Einstein@home це третій великий обчислювальний проект такого типу, основною метою якого є дослідження явищ у міжзоряному просторі.



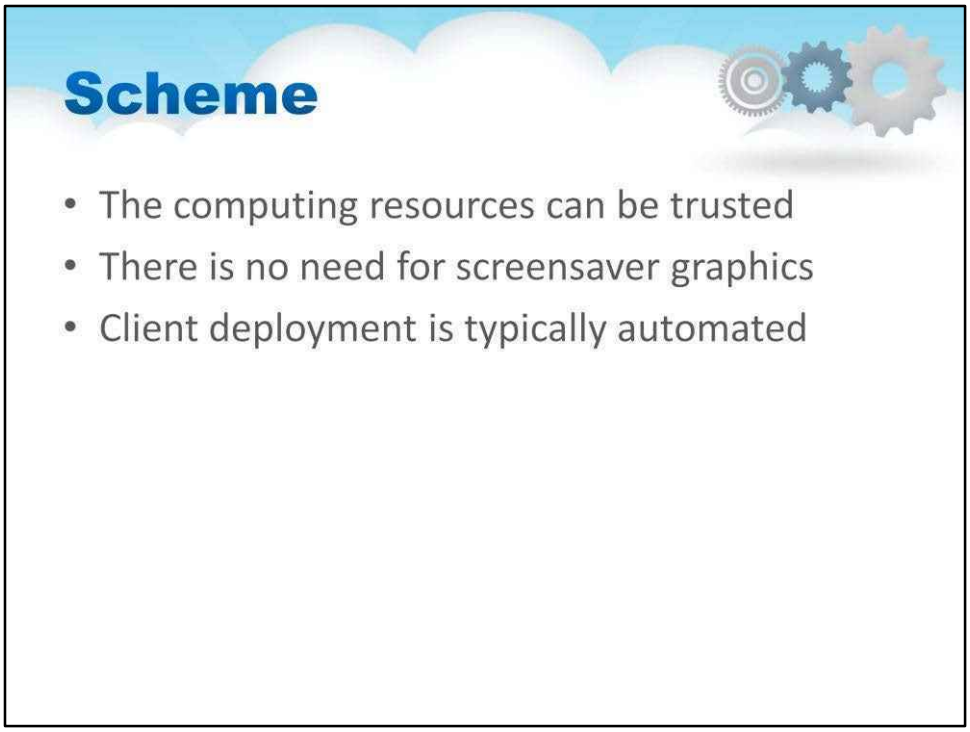
Настільні грід-обчислення

Що таке настільні мережеві обчислення?

Форма розподілених обчислень, у якій організація (бізнес, університет тощо) використовує наявні комп’ютери (настільний комп’ютер та/або вузли кластера) для виконання власних тривалих обчислювальних завдань.



Застосування настільних грід-обчислень: обчислення, мультимедіа

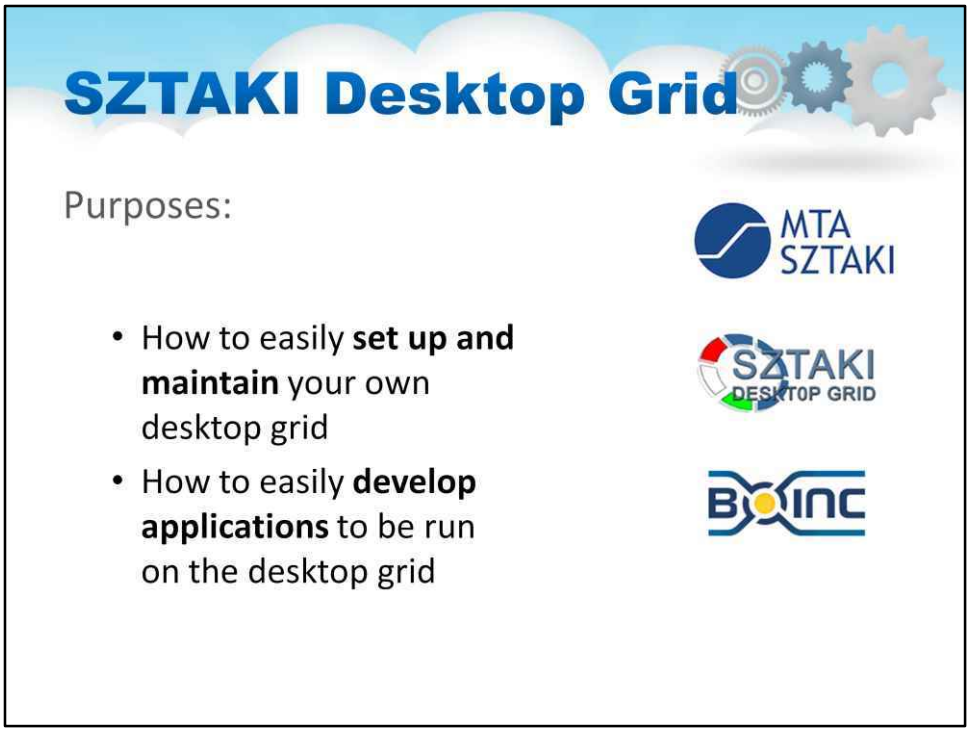


Це схоже на Volunteer Computing, але… відрізняється:

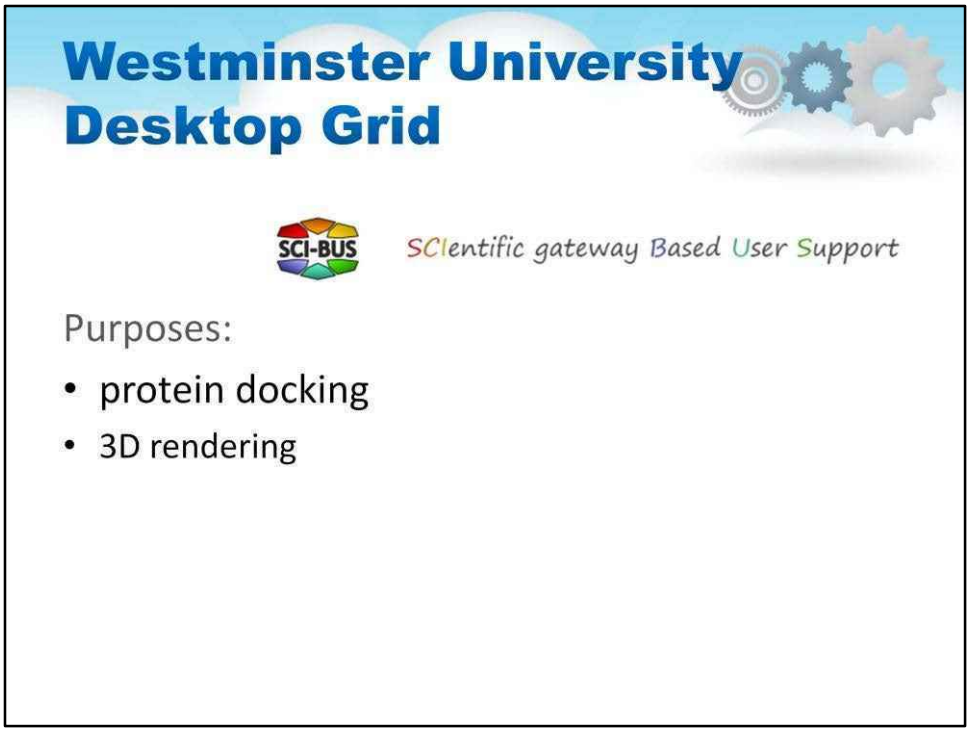
• Обчислювальним ресурсам можна довіряти; тобто можна припустити, що комп'ютери не повертають результати, які є навмисно неправильними або фальсифікованими

• Немає необхідності в графічних заставках; насправді може бути бажаним, щоб обчислення були повністю невидимими та поза контролем користувача ПК

• Розгортання клієнта зазвичай автоматизоване.



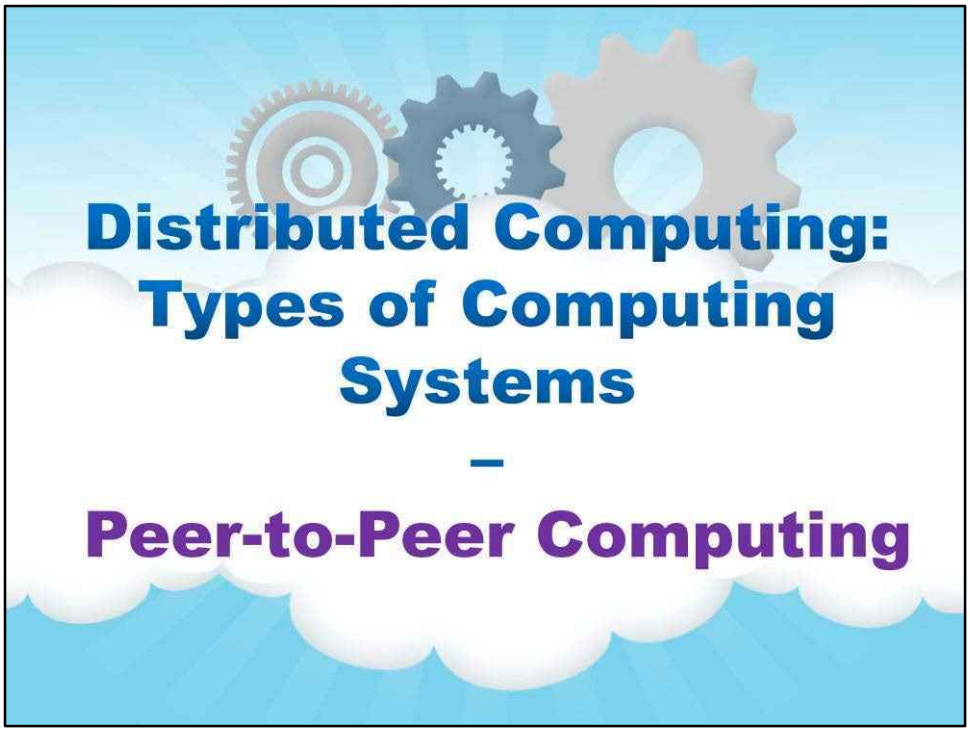
SZTAKI Desktop Grid (SzDG) — це проект BOINC, розташований в Угорщині, яким керує Інститут досліджень комп'ютерів і автоматизації (SZTAKI) Академії наук Угорщини.



Local Desktop Grid Вестмінстерського університету об’єднує лабораторні комп’ютери університету в інфраструктуру Desktop Grid на основі BOINC. Університет розташований у чотирьох основних кампусах і деяких додаткових менших місцях у центральній та північно-західній частині Лондона, кожен із яких пропонує різну кількість комп’ютерів на базі Windows для навчання.

Університет Вестмінстерського університету Local Desktop Grid наразі включає понад 1500 зареєстрованих машин. Ці машини доступні для настільних обчислень сітки, коли вони ввімкнені, але не використовуються студентами для навчання чи інших цілей.

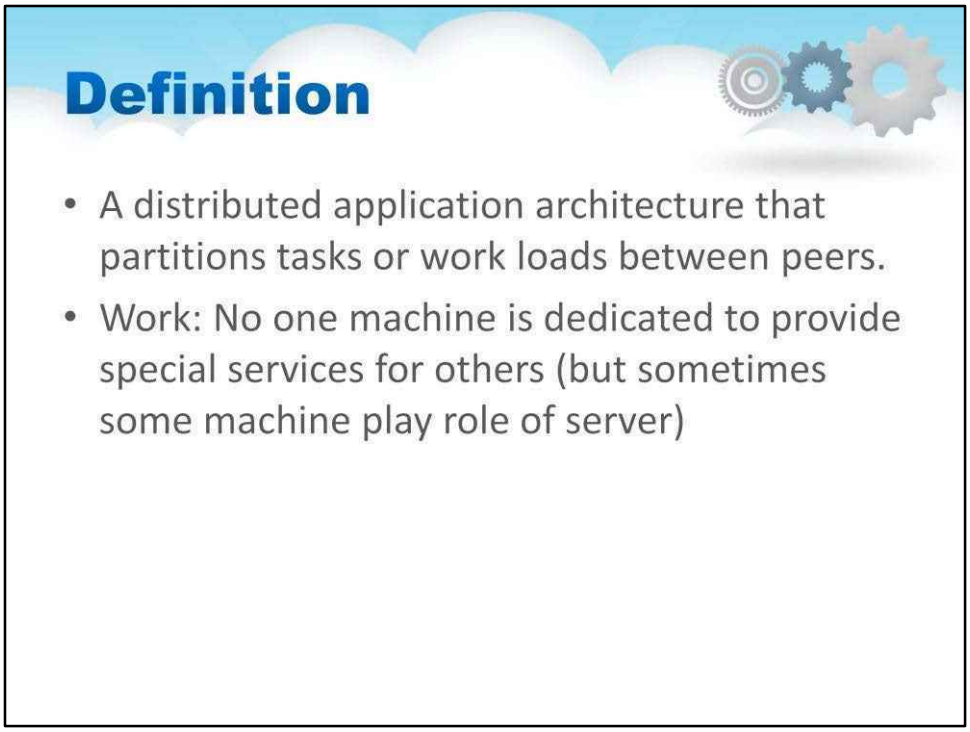
Метою Вестмінстерського шлюзу настільних мереж (UoW DG Gateway) є підтримка інтенсивних обчислень у дослідницьких і навчальних програмах у Вестмінстерському університеті на основі недорогих місцевих ресурсів.



Одноранговi обчислення

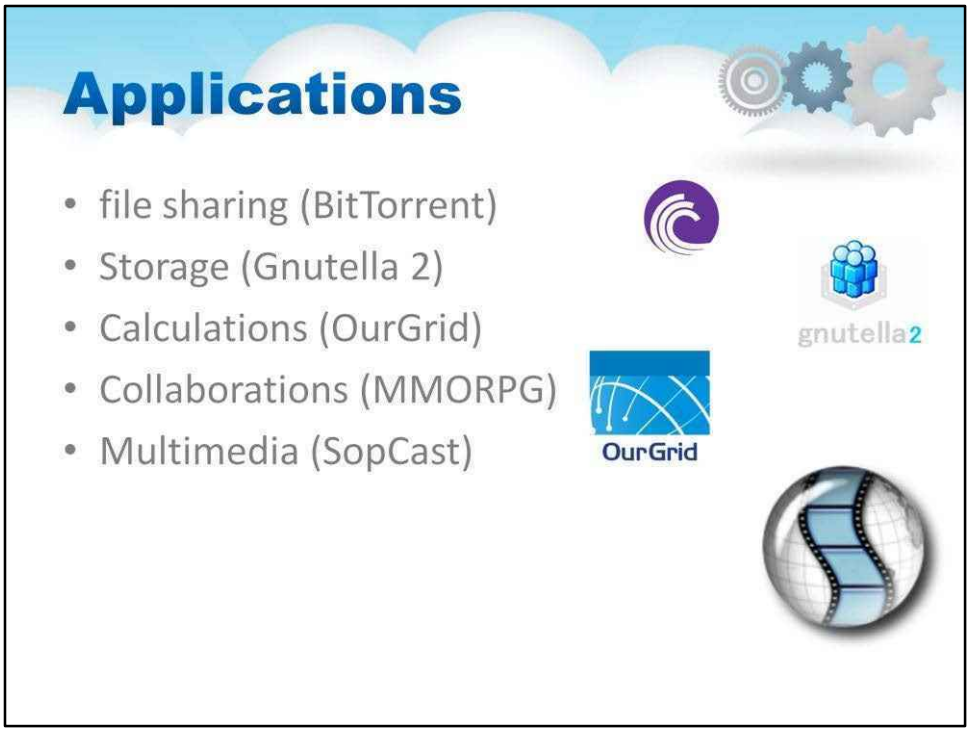
Архітектура P2P пропонує розподілену модель мережевих систем.

У цьому розділі системи P2P представлені на фізичному рівні та накладаються мережі на логічному рівні.



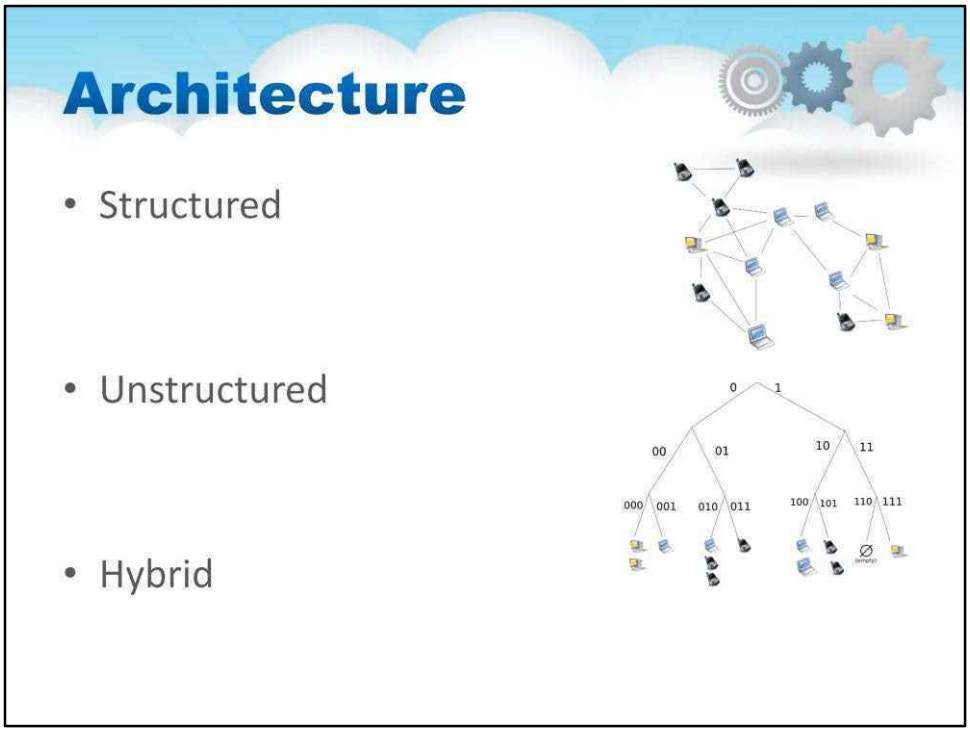
Однорангові (P2P) обчислення або мережа — це розподілена архітектура додатків, яка розподіляє завдання або навантаження між одноранговими користувачами. Ровесники порівну привілейовані, рівноправні учасники програми. Кажуть, що вони утворюють однорангову мережу вузлів.

Однорангові вузли роблять частину своїх ресурсів, таких як обчислювальна потужність, дискове сховище або пропускна здатність мережі, безпосередньо доступними для інших учасників мережі, без необхідності центральної координації серверами або стабільними хостами. Однорангові партнери є як постачальниками, так і споживачами ресурсів, на відміну від традиційної моделі клієнт-сервер, у якій споживання та постачання ресурсів розділено. Нові спільні P2P-системи виходять за рамки епохи однорангових пристроїв, які виконують подібні речі, спільними ресурсами, і шукають різноманітних однорангових систем, які можуть привнести унікальні ресурси та можливості у віртуальну спільноту, таким чином дозволяючи їй брати участь у більших завданнях, ніж ті, які можна виконати. окремими однолітками, але це вигідно всім одноліткам.



Застосування однорангових обчислень дуже широке:

- Обмін файлами (BitTorrent)
- Зберігання (Gnutella 2)
- Розрахунки (OurGrid)
- Співпраця (MMORPG)
- Мультимедіа (SopCast)

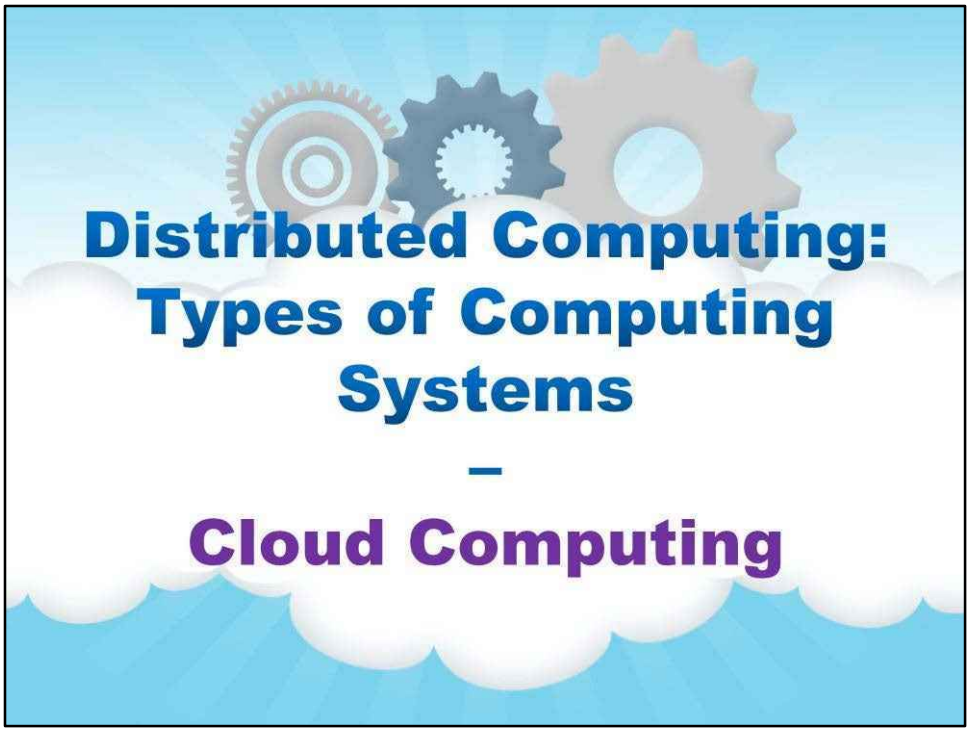


Одноранговa мережа розроблена навколо поняття рівноправних одноранговых вузлів, які одночасно функціонують як «клієнти» і «сервери» для інших вузлів мережі.

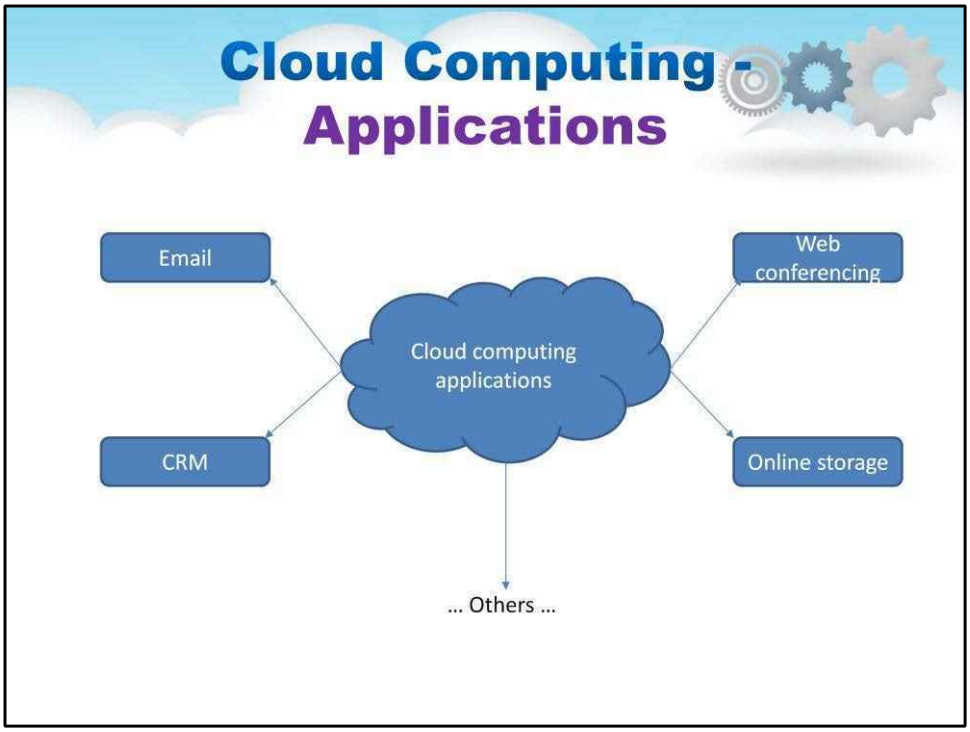
Неструктуровані однорангові мережі за проектом не нав’язують певну структуру накладеній мережі, а скоріше утворюються вузлами, які випадковим чином формують з’єднання один з одним.

У структурованих одноранговых мережах накладення організоване в певну топологію, а протокол гарантує, що будь-який вузол може ефективно шукати в мережі файл/ресурс, навіть якщо ресурс надзвичайно рідкісний.

Гібридні моделі — це комбінація одноранговых і клієнт-серверних моделей. Поширеною гібридною моделлю є наявність центрального сервера, який допомагає партнерам знаходити один одного.



Хмарні обчислення

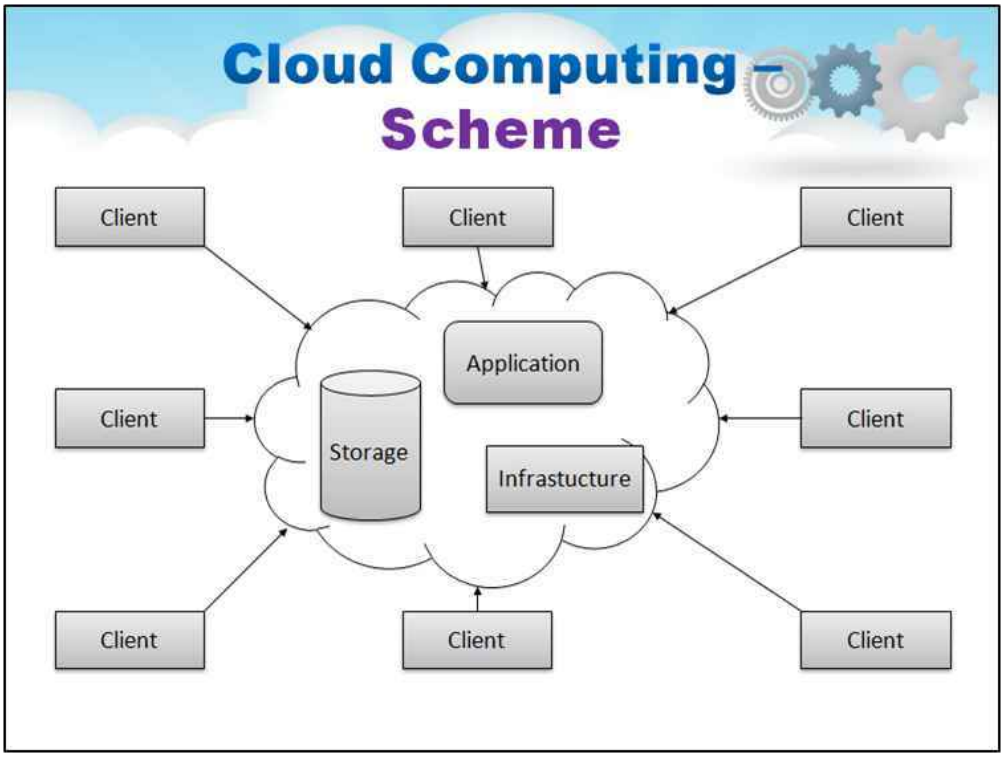


Що таке хмарні обчислення?

Це поняття буде детально пояснено нижче в усіх модулях цього курсу.

На цьому етапі ми розглянемо таке «жаргонне» визначення:

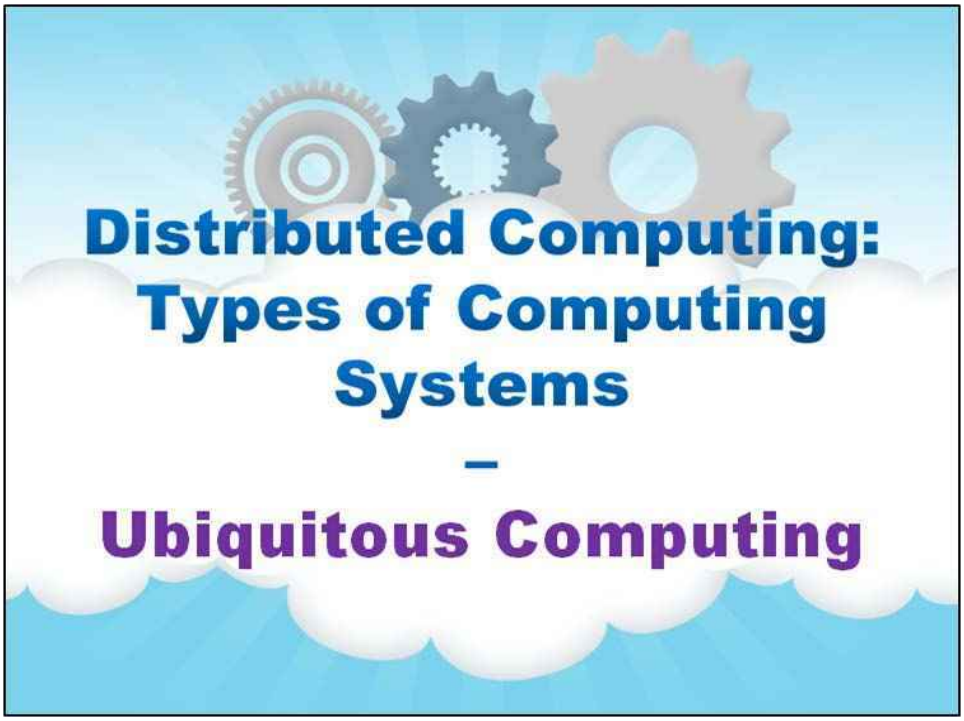
Постачання обчислювальної техніки як послуги, а не як продукту, який є спільним ресурси, програмне забезпечення та інформація надаються комп'ютерам та ін пристроїв як утиліту (наприклад, електричну мережу) через мережу (зазвичай Інтернет).



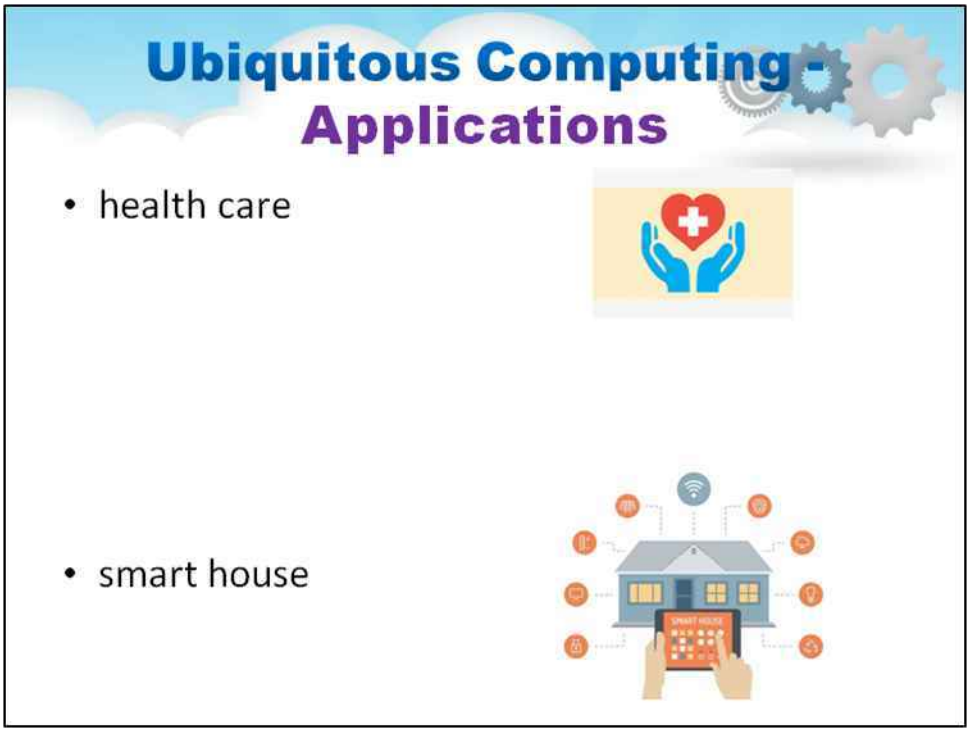
Нам не потрібно встановлювати частину програмного забезпечення на наш локальний ПК, і ось як хмарні обчислення долають проблеми залежності від платформи. Отже, хмарні обчислення роблять бізнес-додатки мобільними та придатними для співпраці.

Хмарні обчислення використовуються в багатьох програмах, таких як:

- Співпраця (Google Docs, Microsoft Office 365)
- Сховище (веб-служби Amazon)
- Обчислення (Google App, Amazon Web Services)



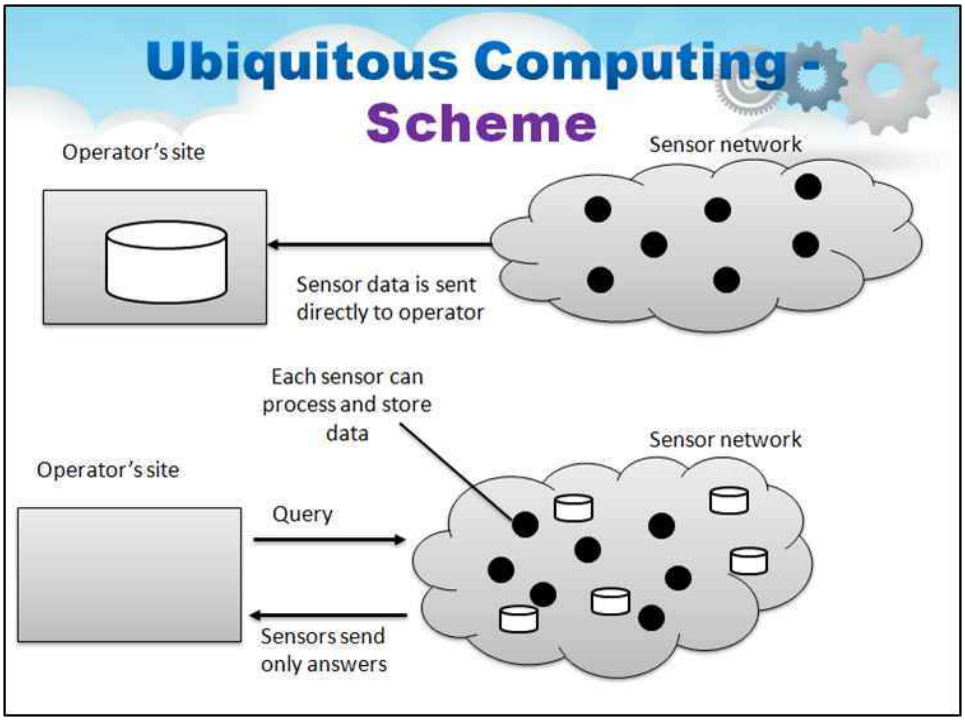
Повсюдне обчислення



Що таке Ubiquitous Computing?

На відміну від настільних комп’ютерів, повсюдне обчислення може відбуватися скрізь і будь-де, використовуючи будь-який пристрій, у будь-якому місці та в будь-якому форматі, включаючи ноутбуки, планшети та термінали в повсякденних об’єктах, таких як холодильник або пара окулярів.

Його використовують по-різному, але переважно в додатках для Інтернету речей і переносних комп’ютерів. Ці теми будуть розглянуті пізніше.

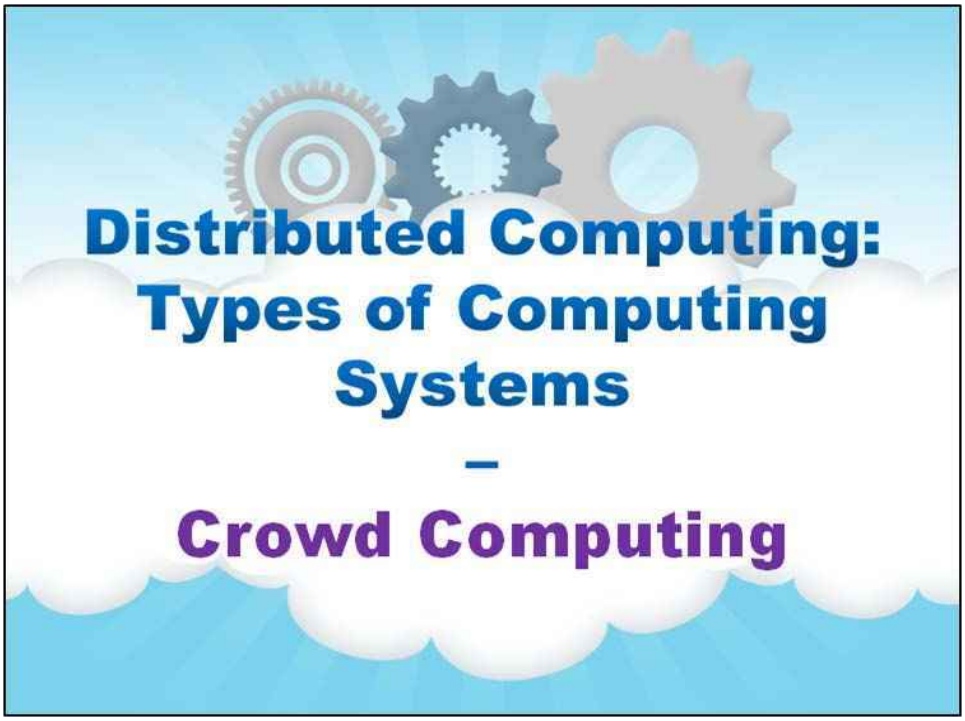


Попередня (стара) парадигма повсюдних обчислень (верхня частина малюнка):

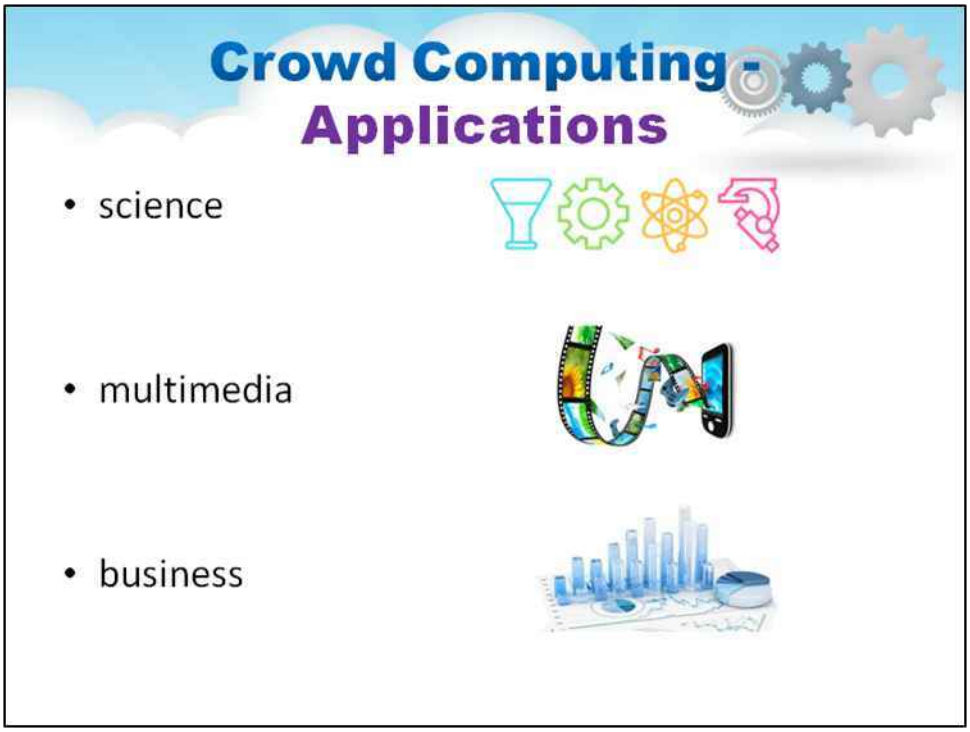
Організація бази даних сенсорної мережі, при цьому зберігання та обробка даних здійснюється тільки на майданчику оператора.

Хмара-Туман-Роса (нова) парадигма повсюдних обчислень (верхня частина малюнка):

Організація бази даних сенсорної мережі (Cloud), при цьому зберігаються та обробляються дані лише на контролерах поблизу датчиків (Fog) або датчиків (Dew)



Crowd Computing

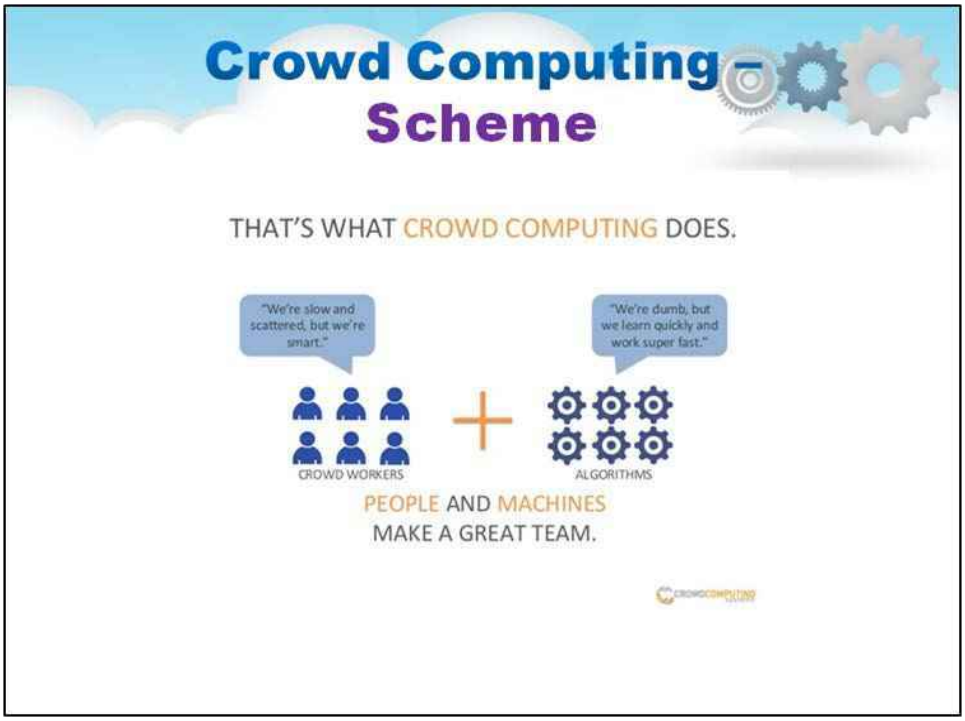


Що таке Crowd Computing (краудсорсинг)?

«Використання можливостей людей в Інтернеті для виконання важких завдань

окремим користувачам або комп'ютерам, які можна виконувати самостійно. Як хмарні обчислення, натовп

обчислення пропонують еластичні людські ресурси на вимогу, які можуть стимулювати нові програми та нові способи мислення про технології» (С) Роб Міллер



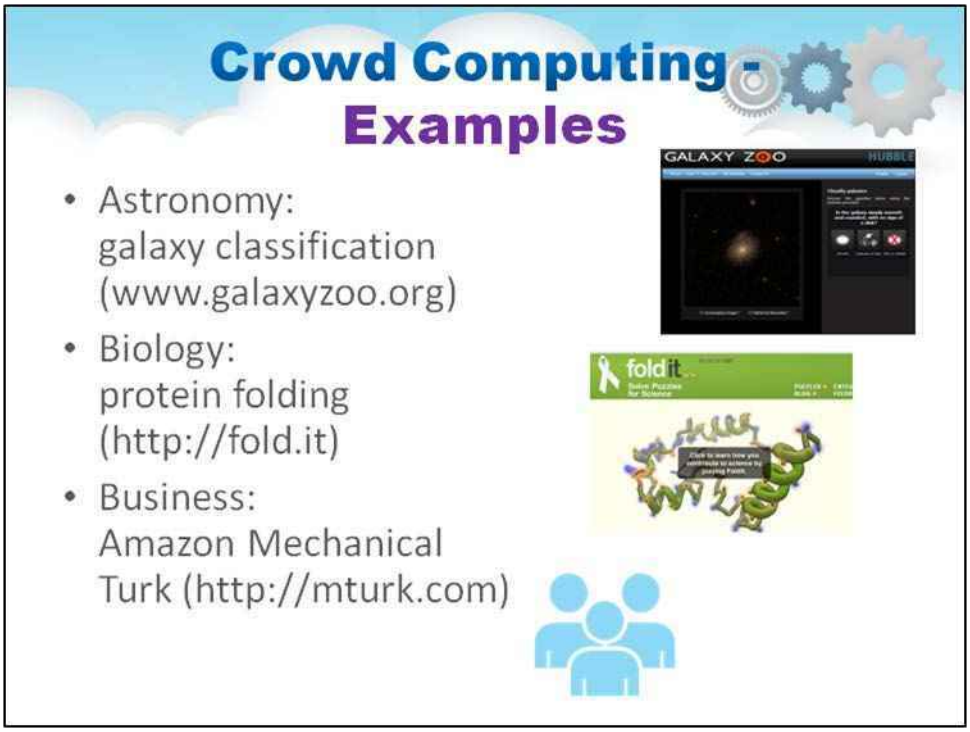
Це насправді так

волонтерський комп'ютерний

комп'ютер, але використовує обчислення натовпу

обчислення мозку волонтера

і поєднує людський інтелект (натовп) зі штучним інтелектом (хмара).



Приклади Crowd Computing є всюди:

Астрономія:

класифікація галактик (www.galaxyzoo.org)

Біологія:

згортання білка (http://fold.it)

Бізнес:

Amazon Mechanical Turk (http://mturk.com)

Хмарні обчислення

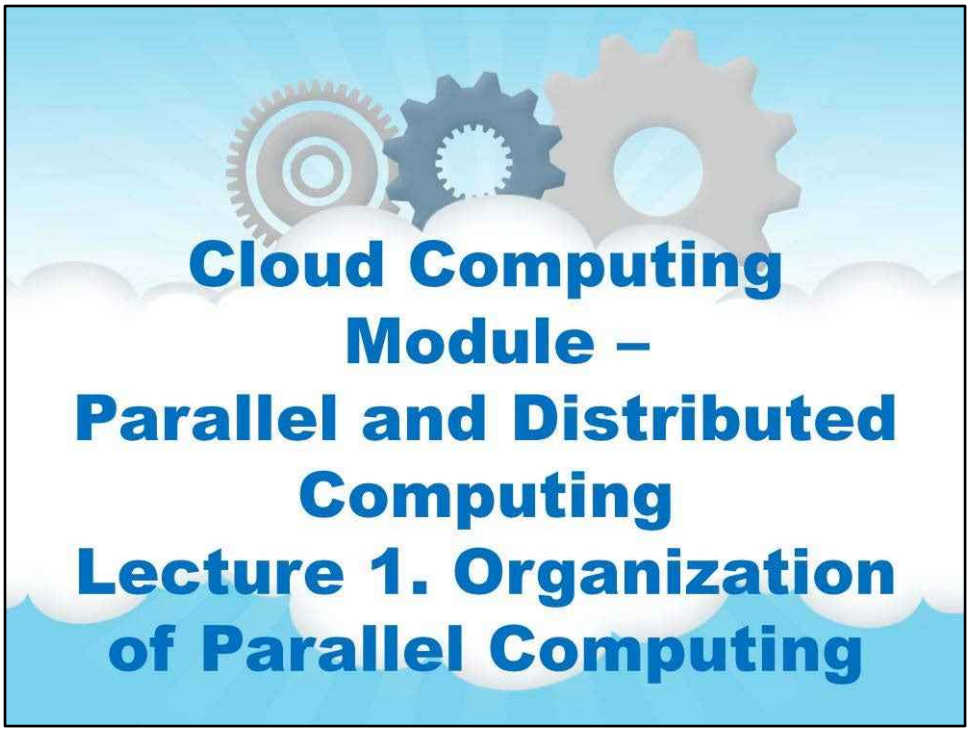
Лекційний посібник

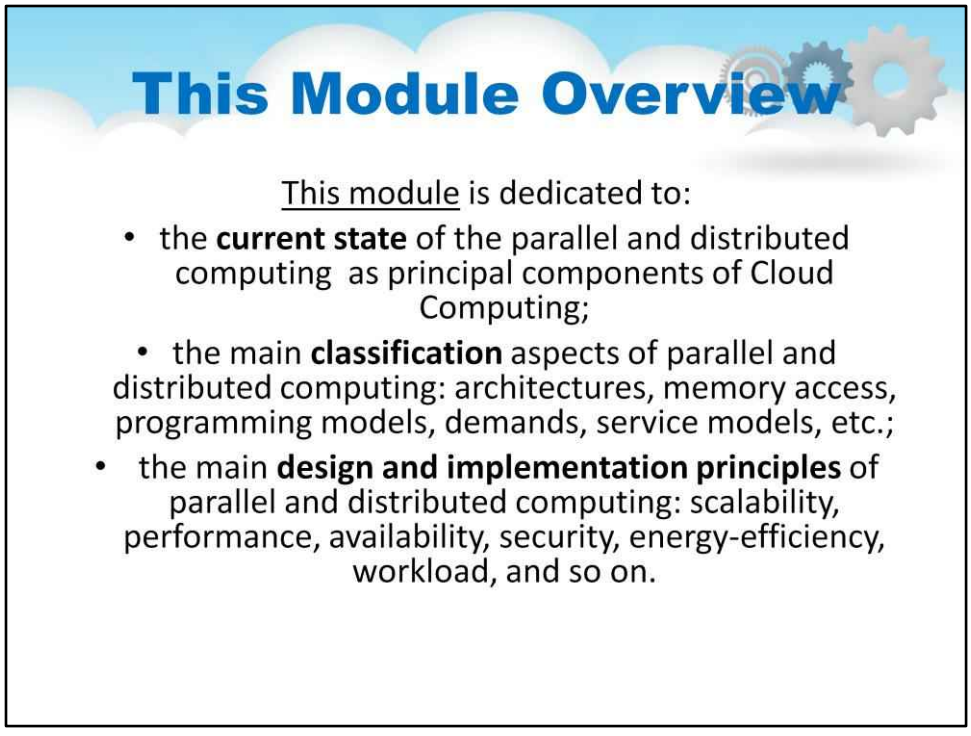
Том 1

Модуль 1

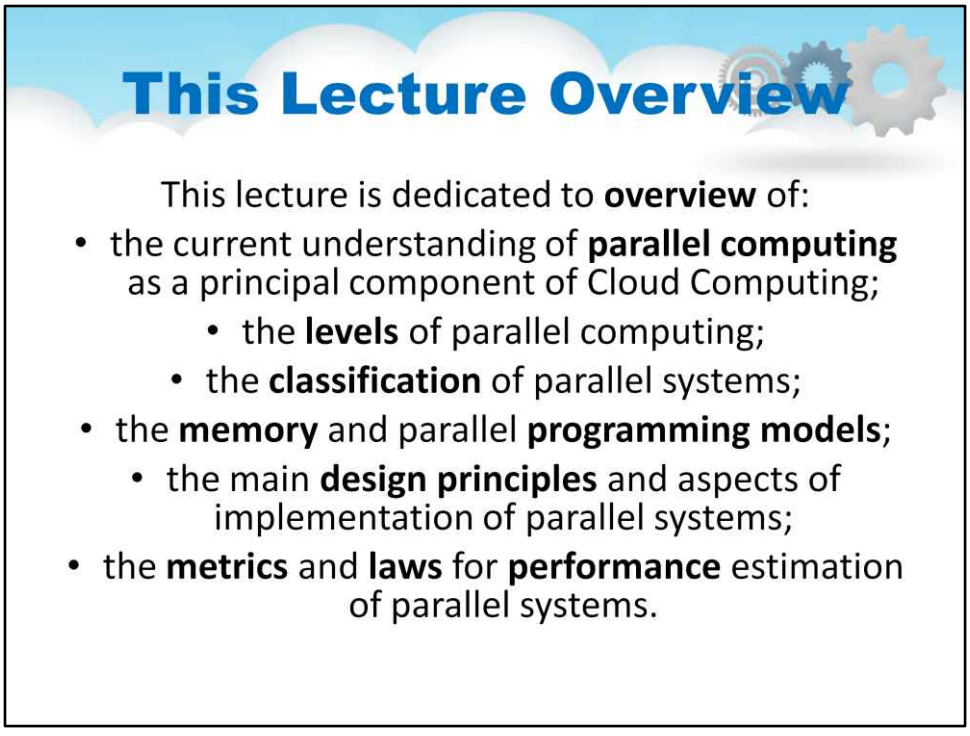
Паралельні і розподілені обчислення

Зміст


Cloud Computing
Module –
Parallel and Distributed
Computing
Lecture 1. Organization
of Parallel Computing



Модуль 1. Паралельні та розподілені обчислення



Лекція 1. Організація

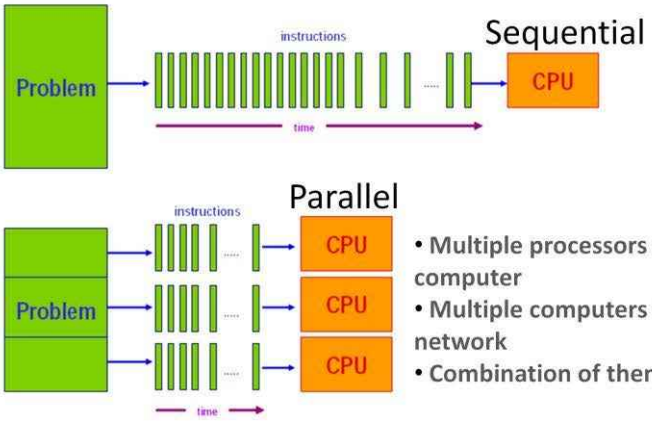
Паралельні обчислення

Паралельні обчислення

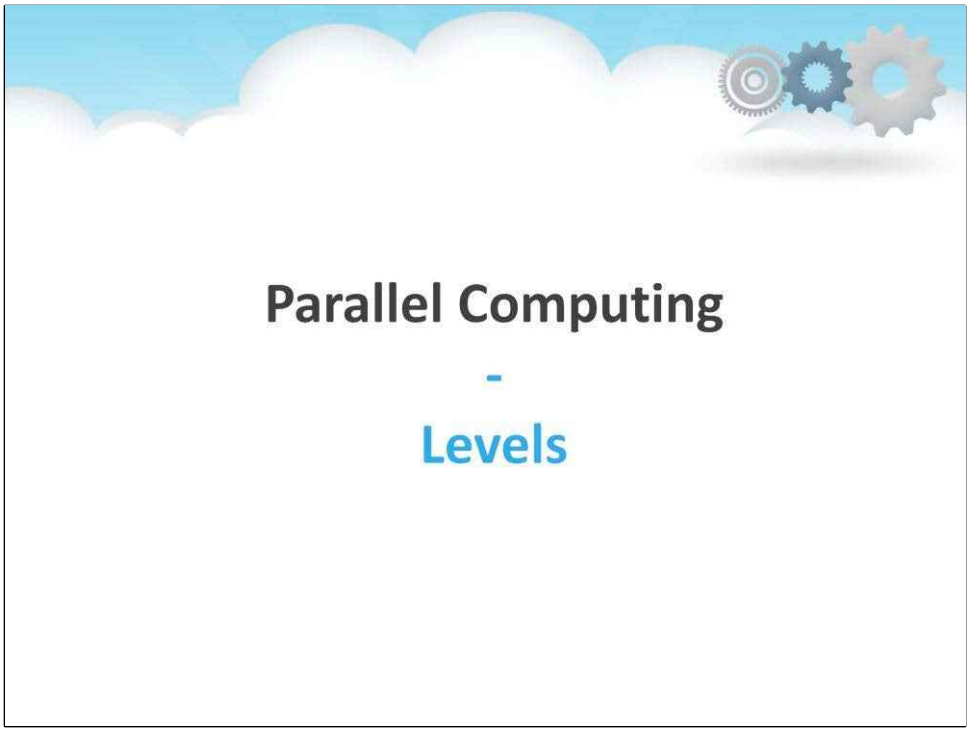
Огляд

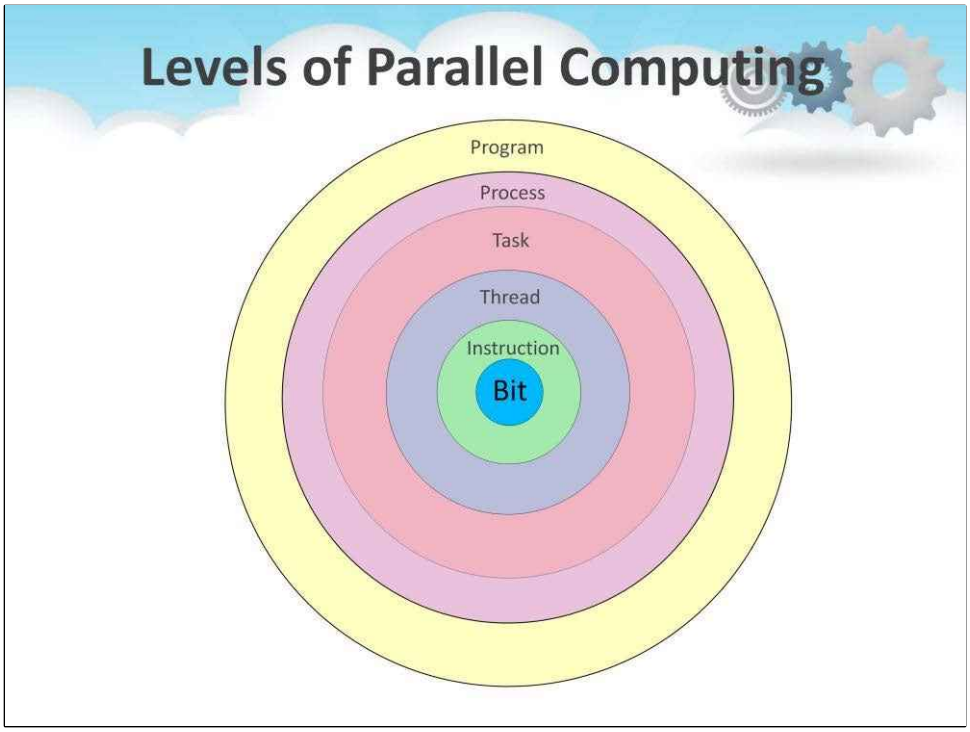
Parallel Computing – Definition:

… a form of computation, in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are solved concurrently ("in parallel“)



- Multiple processors in the same computer
- Multiple computers in the network
- Combination of them


Parallel Computing
-
Levels



Бітовий рівень: від бітової обробки до текстової обробки.

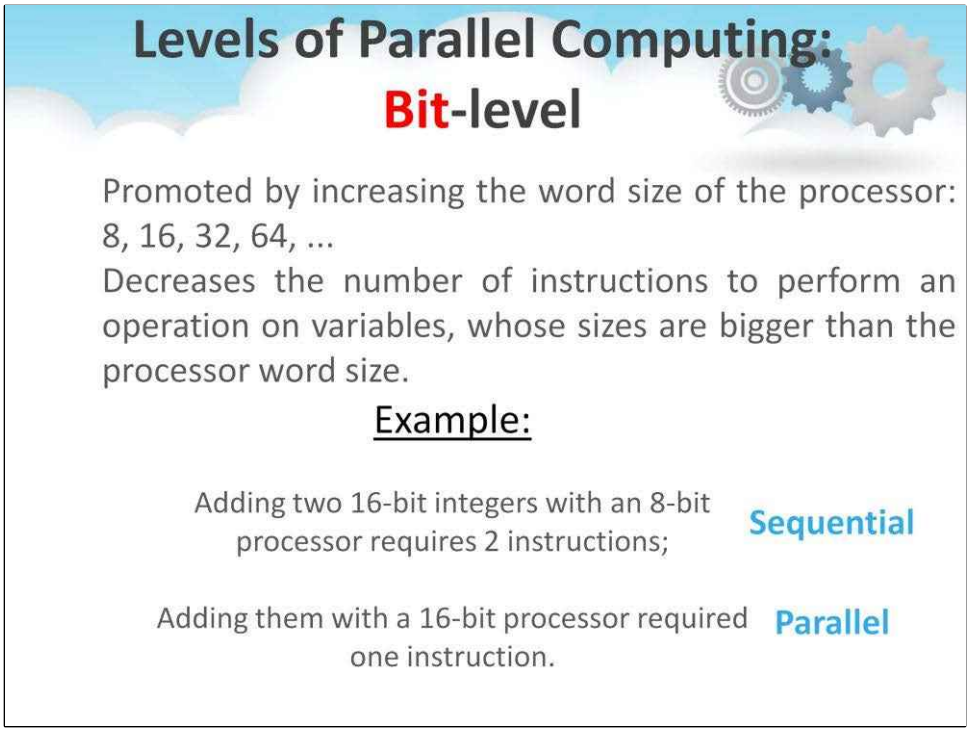
Інструкція рівня: одна операція процесора.

Потоковий рівень: потік виконання (має одну або декілька інструкцій).

Рівень завдання: шлях виконання через адресний простір, який містить багато інструкцій. (Іноді завдання та процес використовуються як взаємозамінні).

Процесний рівень: екземпляр програми у виконанні. Він має власний адресний простір і взаємодіє з іншими процесами лише через зв'язок, керований ОС. Процес може містити один або кілька потоків,
які мають однаковий адресний простір і взаємодіють безпосередньо.

Програмний рівень: виконуваний файл з одним або кількома завданнями.



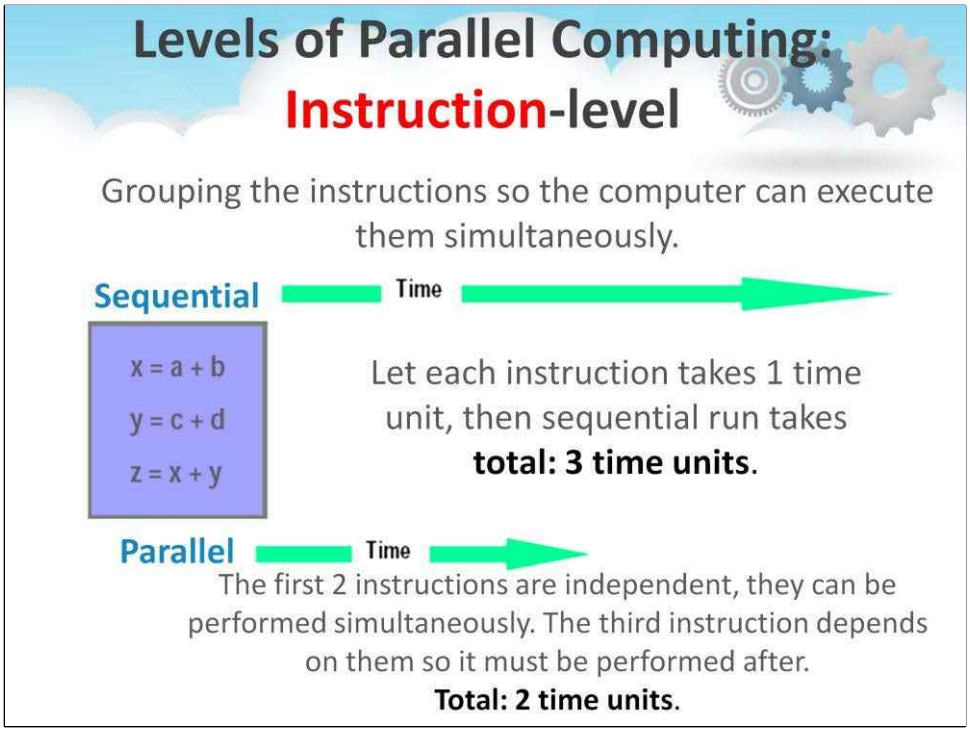
Більше шістдесяти років тому, коли апаратне забезпечення було громіздким і дорогим, більшість комп'ютерів проектувалися за принципом послідовного розряду.

У цьому сценарії**паралелізм на бітовому рівні (BLP)**поступово перетворює бітову послідовну обробку на обробку на рівні слова.

Протягом багатьох років користувачі переходили від 4-розрядних мікропроцесорів до 8-, 16-, 32- та 64-розрядних процесорів.

Паралелізм на бітовому рівні (BLP)привели нас до наступної хвилі вдосконалення, відомої як**паралелізм рівня інструкцій (ILP)**.

Розглянемо**паралелізм рівня інструкцій (ILP)**в деталях.

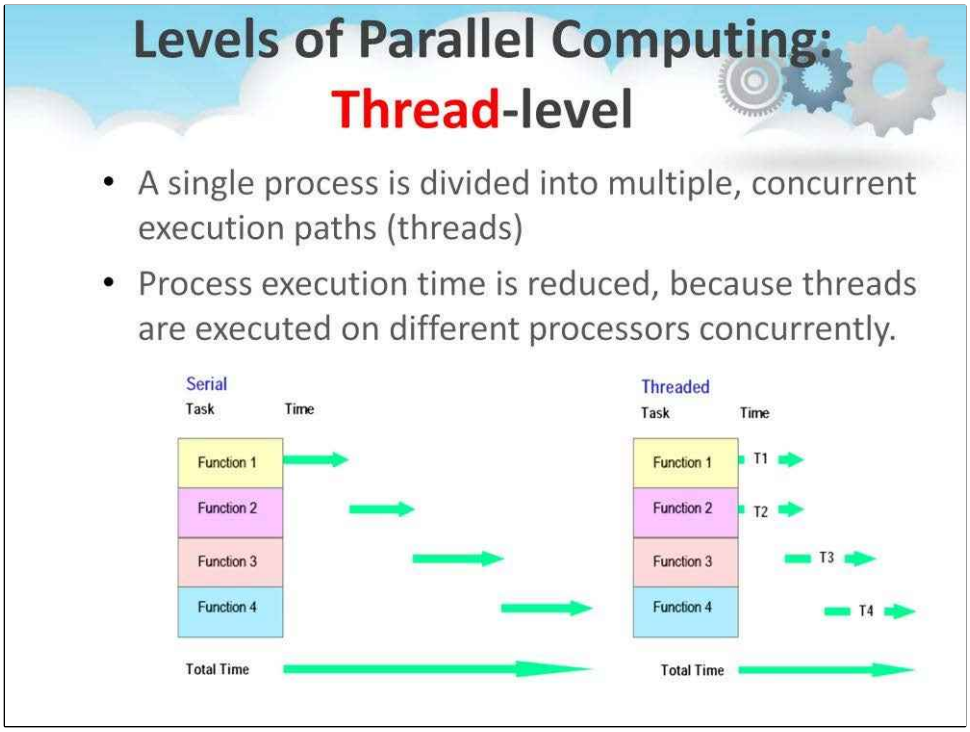


в**паралелізм рівня інструкцій (ILP)**
процесор виконує декілька інструкцій одночасно, а не лише одну інструкцію за раз.

За останні понад 40 років,
ILP суттєво еволюціонував із появою різноманітних технік, таких як:

- прогноз гілок,
- динамічне планування,
- спекуляції,
- конвеєрне,
- суперскалярні обчислення,
- Архітектури VLIW (дуже довге командне слово) і
- багатопотоковість.

Розглянемо**багатопотоковість**в деталях.



Багатопотоковість або**паралелізм на рівні потоку (TLP)**– це спільне використання функціональних блоків одного процесора декількома процесами або потоками, які беруть участь у накладеному виконанні. Метою може бути виконання:

- кілька програм на одному процесорі або
- виконувати одну програму як багатопотокову програму (реальна паралельна програма).

Процеси можуть належати різним користувачам (додаткам), але потоки належать одному користувачеві (додатку).

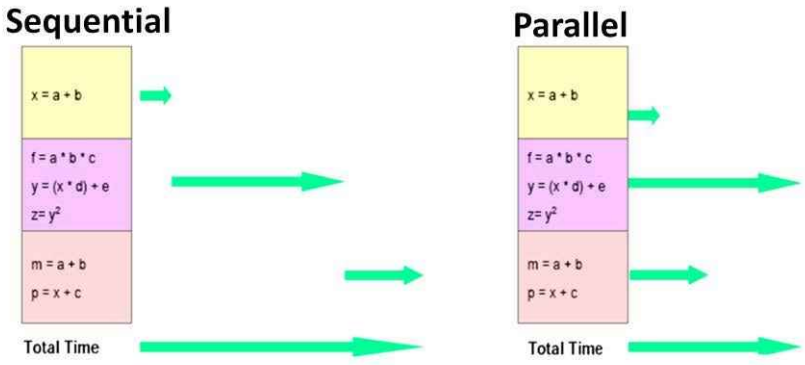
Це вищий рівень паралелізму, ніж паралелізм рівня інструкцій (ILP), оскільки виконання кожного окремого потоку може використовувати ILP.

Перевага(у порівнянні з розпаралелюванням рівня процесу):

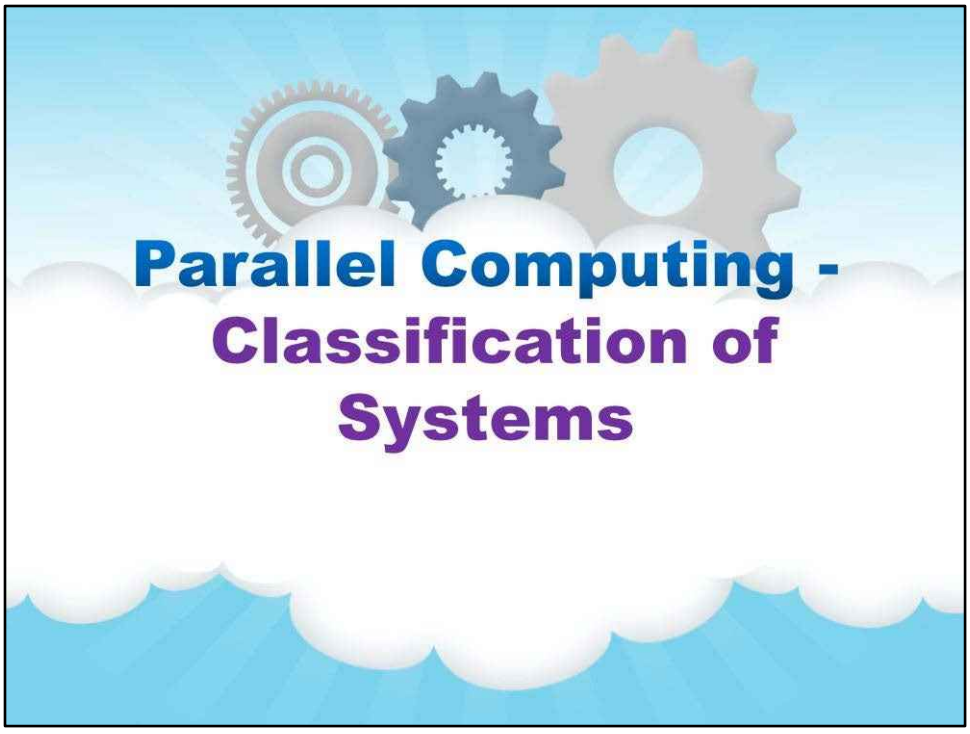
Перемикання між процесами, зазвичай позначається перемиканням контексту в термінології операційних систем, зазвичай може використовувати сотні або навіть тисячі тактів, в той час як є багатопотокові процесори, які можуть переключатися на інший потік протягом одного такту.

Levels of Parallel Computing: Task-level

The different operations are performed on the same (or different) data, i.e. each processor runs a different task, but it can communicate with other processors to exchange data.

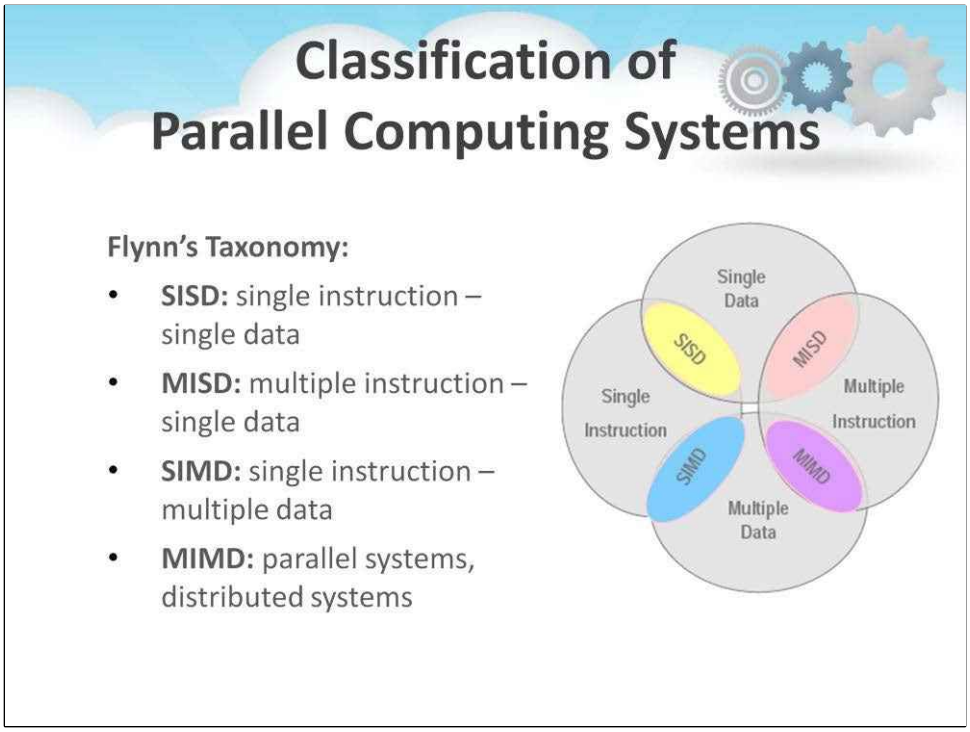


Example: add, multiply, … many parts of data



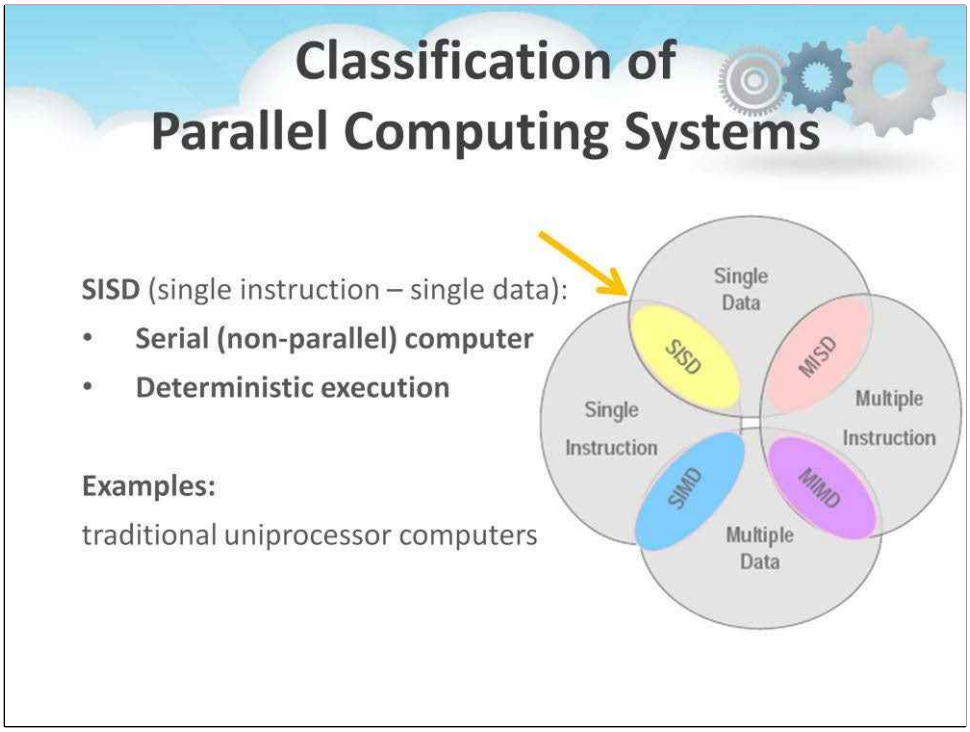
Класифікація систем

Тепер розглянемо класифікацію паралельних систем.



Флінн розділив мультипроцесори на чотири категорії на основі множинності **потоки інструкцій**і**потоки даних.**

Це стало відомо як знаменитеТаксономія Фліҥна

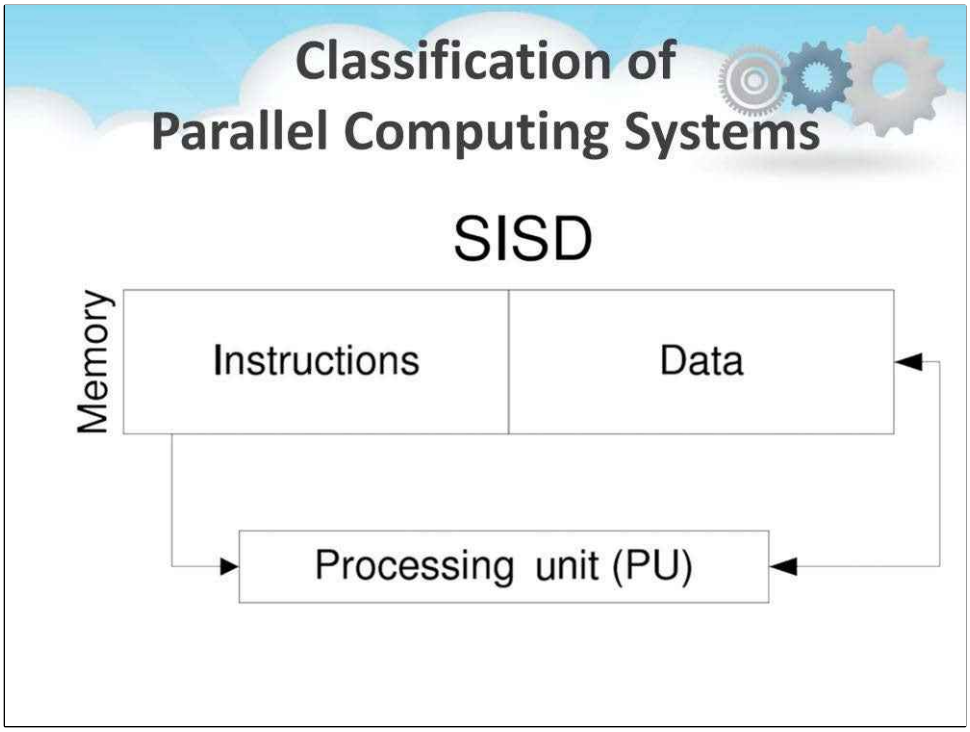


Звичайний комп’ютер (однопроцесорний або машина фон Неймана) називається а **Єдині дані однієї інструкції (SISD)**машина.

Тут ви можете побачити:

- його місце в таксономії Флінна
- його основні характеристики
- і приклади реалізації – в традиційних однопроцесорних комп'ютерах.

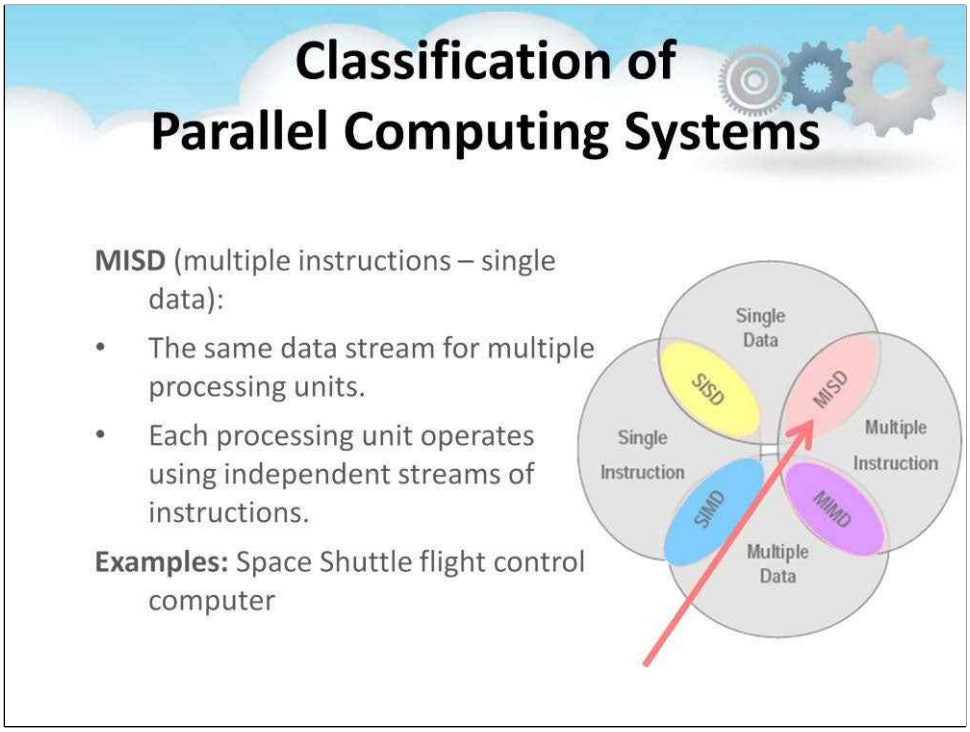
Розглянемо їх докладніше на наступному слайді…



Він має один блок виконання або обробки (PU).

Це**блок обробки**контролюється а**одна послідовність інструкцій**,
і він діє на а**одна послідовність даних**в пам'яті.

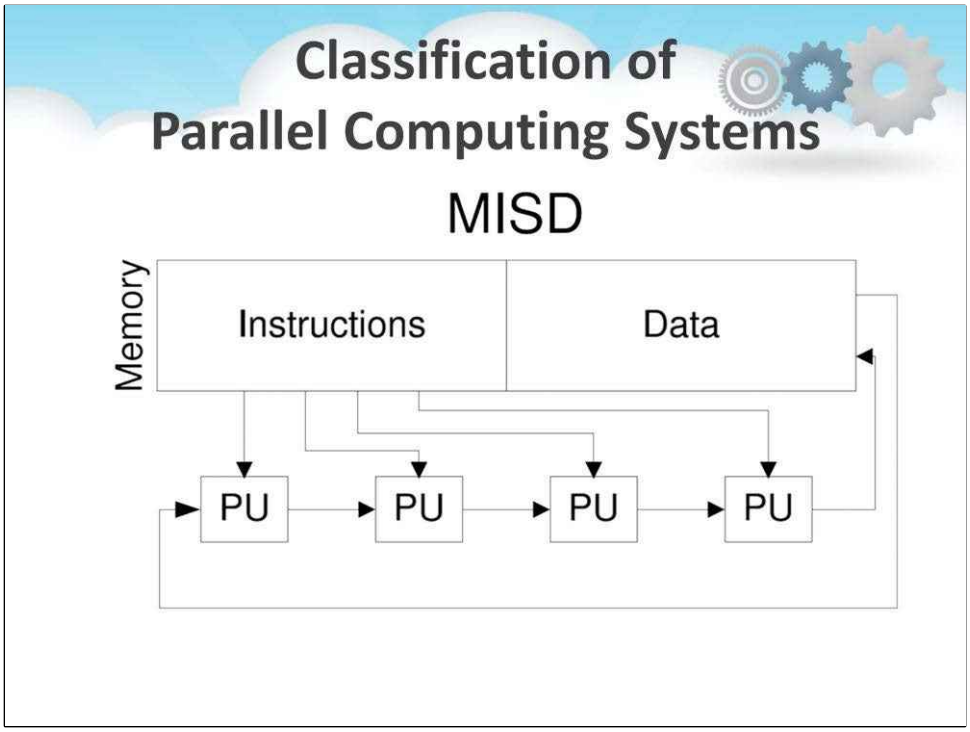
На початку розвитку обчислювальної техніки логіка управління мала декодувати інструкції в сигнали керування, які керують виконанням і
трафік даних у процесорі був дорогим компонентом.



The**Багатокомпонентні одиничні дані (MISD)**категорія машин була у літературі неоднозначно розглядається. У деяких підручниках просто сказано, що машин цієї категорії не створено, а в інших наводяться приклади. На наш погляд MISD є важливою категорією, що представляє різні паралельні архітектури.

Тут ви можете побачити:

- його місце в таксономії Флінна
- його основні характеристики
- та приклади реалізації – в комп'ютерах керування авіонікою, в спеціалізованих апаратних структурах –розглянемо їх детальніше на наступному слайді…



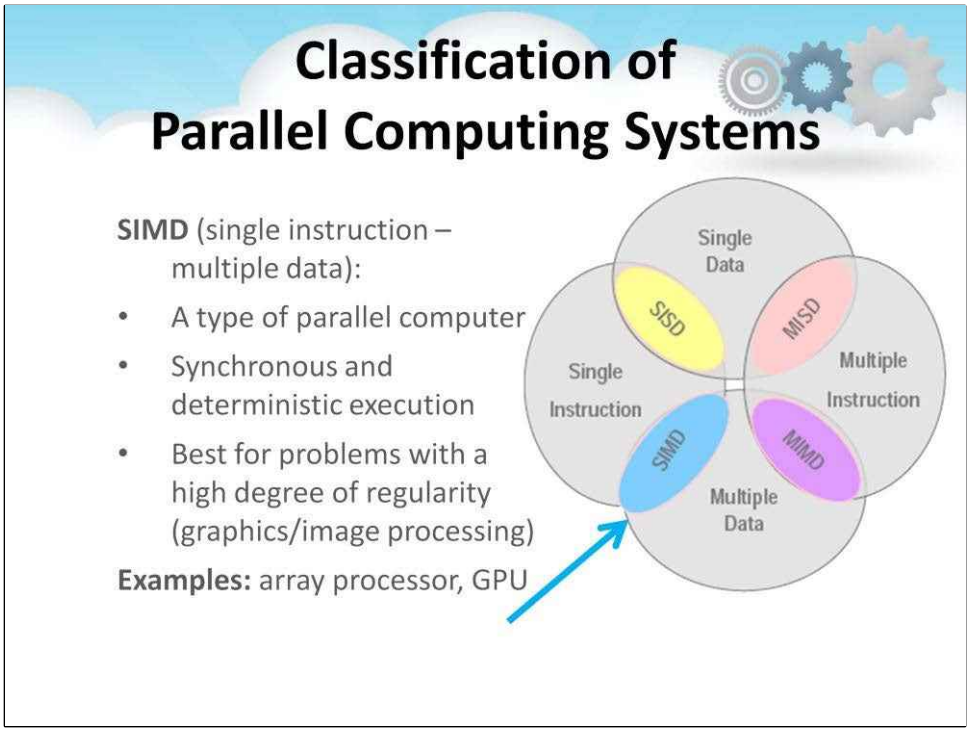
Один із прикладів архітектур, представлених у класичній статті Флінна, дуже схожий на варіант, показаний на цьому малюнку.

Ось вихідний потік даних

- відправляється з пам'яті на перший ПУ,
- потім отриманий потік даних надсилається до наступного PU, де він обробляється іншою програмою (потік інструкцій)
- і так далі, доки його не буде передано назад у пам'ять.

Цей тип обчислень деякі автори називають а**програмний конвеєр**. Це може бути ефективним для таких програм, як обробка в реальному часі потоку даних зображень (відео), де дані передаються через різні PU, які виконують різні функції обробки зображень (наприклад, фільтрація або вилучення функцій).

Ще один тип паралельних архітектур, які можна класифікувати як MISD **систолічні масиви**. Це спеціалізовані апаратні структури, часто реалізовані як **інтегральна схема для конкретного застосування**(ASIC), а також використовувати висококонвеєрне та паралельн виконання певних алгоритмів, таких як зіставлення шаблонів або сортування чи майнінг криптовалюти.

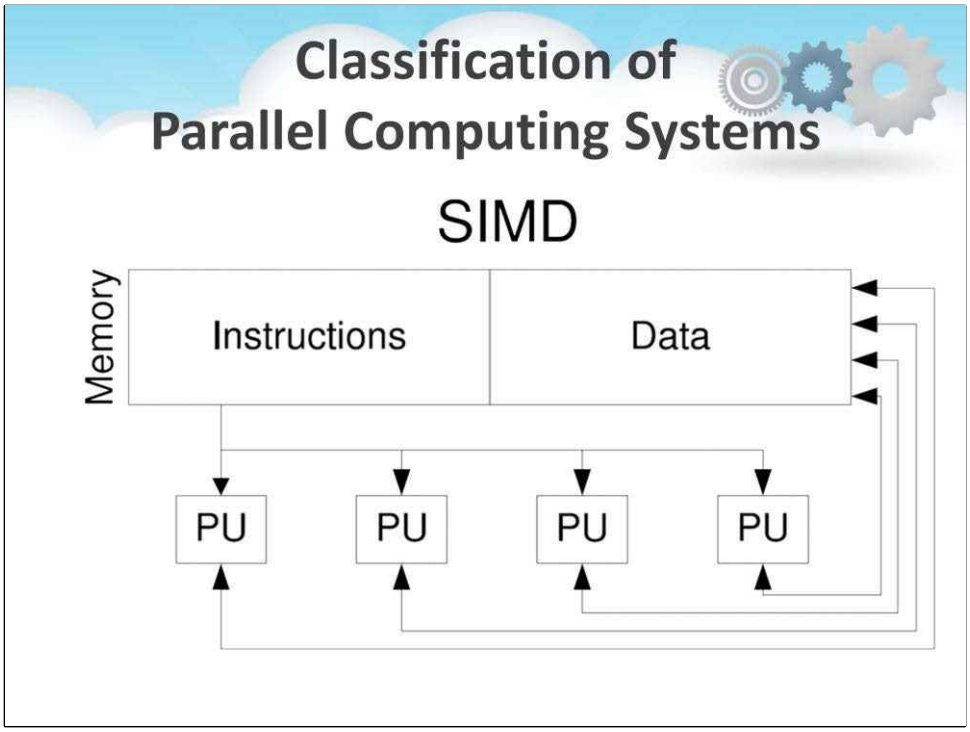


При введенні паралельної обробки,
тому було природно дозволити**кілька одиниць виконання**працювати з різними даними (**кілька даних**потоки).
Ними керував один і той же єдиний блок управління, тобто єдиний потік інструкцій.

Фундаментальний**обмеження**з них**Одна інструкція з кількома даними (SIMD)**архітектури полягає в тому, що різні PU**не може виконувати різні інструкції**і, в той же час, вони всі **прив'язаний до одного потоку інструкцій**.

Тут ви можете побачити:
- його місце в таксономії Флінна
- його основні характеристики
- та приклади реалізації – в масивних процесорах, GPU тощо.

Розглянемо їх докладніше на наступному слайді…



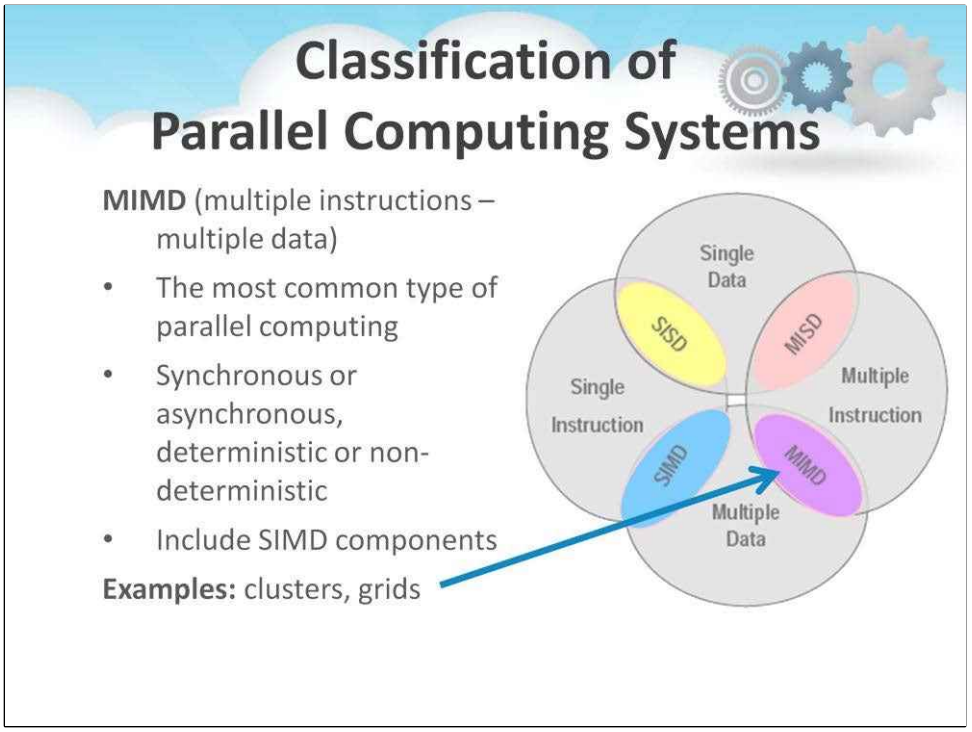
Машини SIMD розвивалися в багатьох варіантах.

Головна відмінність між
- SIMD с**спільна пам'ять**(як показано на цьому малюнку) і
- комп'ютери SIMD с**розподілена пам'ять**.

В останньому варіанті (**розподілена пам'ять**),
основна пам'ять розподіляється між різними PU.

The**перевага**цієї архітектури є те, що це багато**легше реалізувати**порівняно з кількома потоками даних в одну спільну пам’ять.

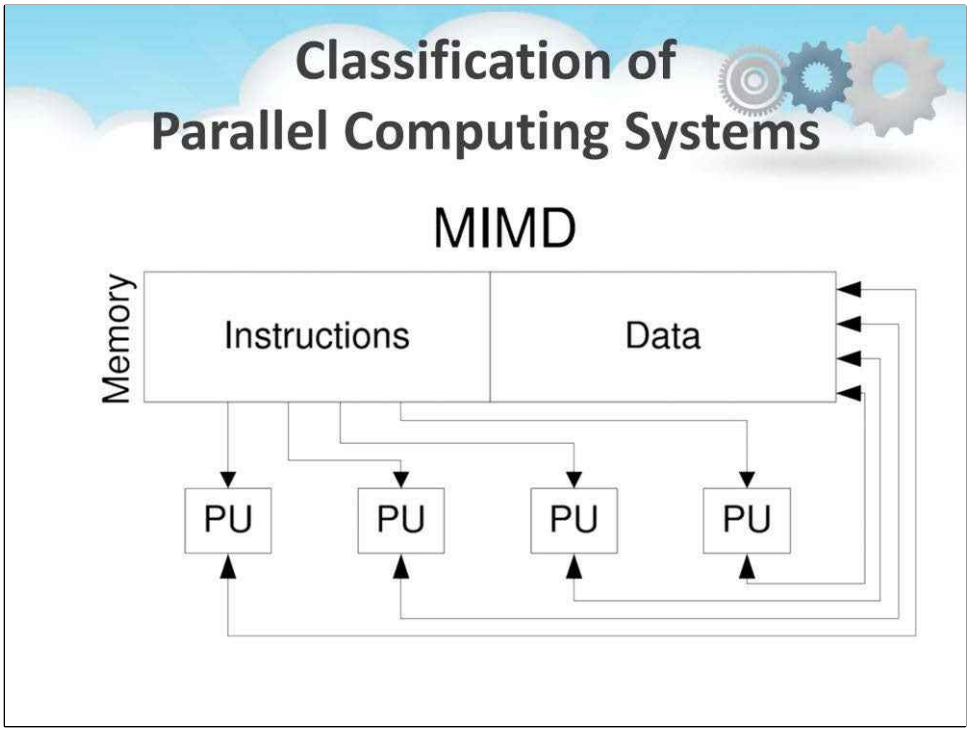
The**недолік**полягає в тому, що це створює потребу в певному механізмі, наприклад**спеціальні інструкції для спілкування**між різними PU.



Кілька інструкцій із кількома даними (MIMD)категорія включає в себе більшість сучасних паралельних комп'ютерних архітектур,
і його нездатність класифікувати їх
був джерелом для пропозиції**різні альтернативні таксономії**,
наприклад, описаний Куїном у його книзі:

MJ Quinn. Проектування ефективних алгоритмів для паралельних комп'ютерів. McGraw-Hill Book Company, Нью-Йорк, 1987.

Але це питання виходить за межі цього курсу.

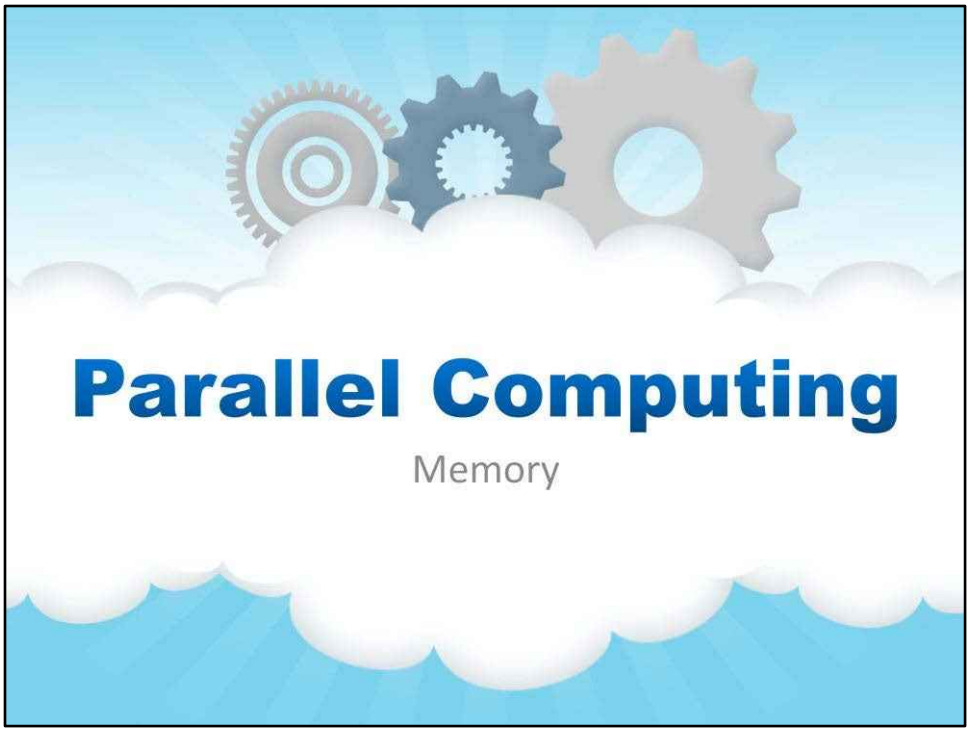


У комп’ютері MIMD кожен PU має власний блок керування, який читає, окремий потік інструкцій, що диктує виконання в його PU.

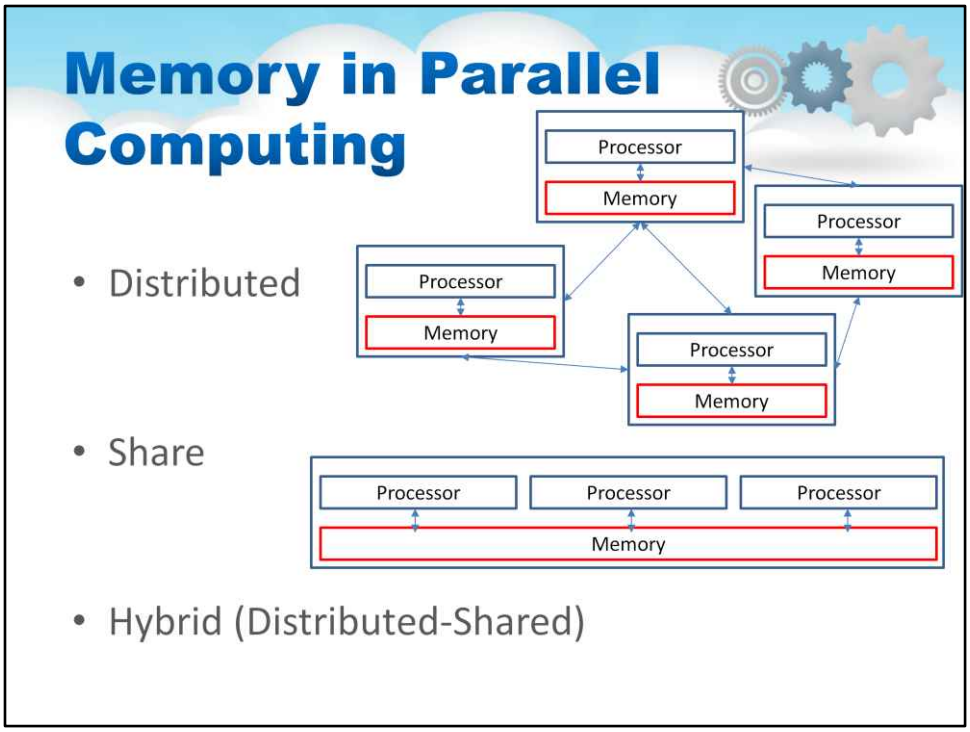
Як і для машин SIMD,
Основним підрозділом машин MIMD є машини, що мають
- спільна пам'ять або
- розподілена пам'ять.

В останньому варіанті (**розподілена пам'ять**) кожен PU може мати локальну пам’ять, де зберігаються як інструкції, так і дані. Це веде нас до іншої основної класифікації мультипроцесорів,
– спільна пам'ятьбагатопроцесорні і**передача повідомлень**багатопроцесорні.

Давайте розглянемо організацію пам'яті в НАСТУПНОМУ РОЗДІЛІ...



Пам'ять



Архітектура розподіленої пам'яті

Кожен процесор має свій адресний простір (локальна пам'ять).

Зміни, зроблені кожним процесором, не видно іншим.

Архітектура спільної пам'яті

Процесори виконують свої операції незалежно, але вони

спільно використовують ті самі ресурси

оскільки вони мають доступ до всієї пам'яті як глобального адресного простору.

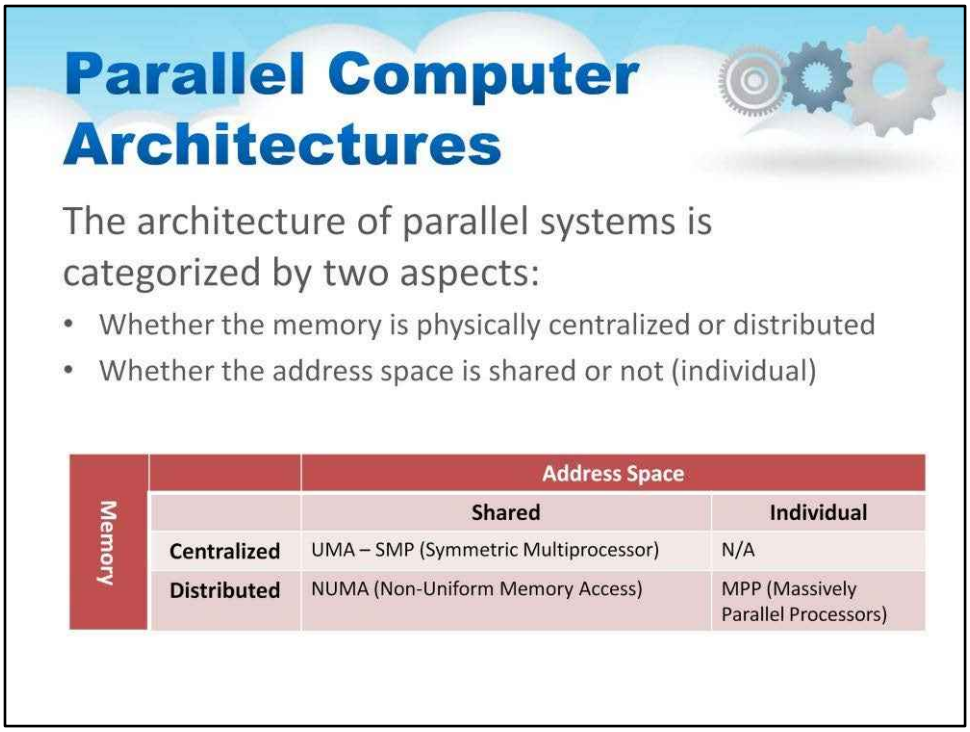
Гібридна (розподілена-спільна) архітектура пам'яті Це

містить

- компоненти спільної пам'яті і

- компоненти розподіленої пам'яті.

Він використовується в більшості сучасних швидких і великих паралельних комп'ютерів.

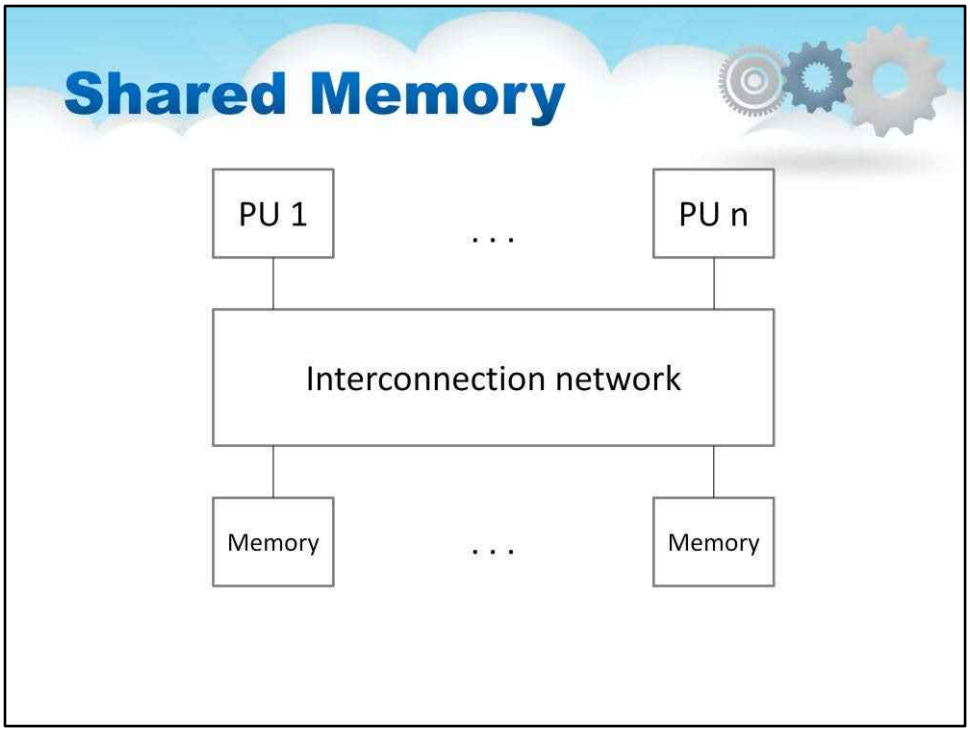


Архітектура паралельних систем класифікується за двома аспектами:
- Чи є пам'ять фізично централізованою чи розподіленою
- Спільний адресний простір чи ні

Якщо
- пам'ять є**централізований**l адресний простір є**спільний доступ**тоді маємо**рівномірний доступ до пам'яті (UMA)**архітектура;
- пам'ять є**поширюється**l адресний простір є**спільний доступ**тоді маємо**нерівномірний доступ до пам'яті (NUMA)**архітектура;
- пам'ять є**поширюється**l адресний простір є**НЕ поділився**тоді маємо**розподілена пам'ять (DM)**архітектура.

Іноді системи с
- **архітектура єдиного доступу до пам'яті (UMA).**називаються**симетричні мультипроцесори (SMP)**

і
- **архітектура розподіленої пам'яті (DM).**називаються**масивно паралельні процесори (MPP)**



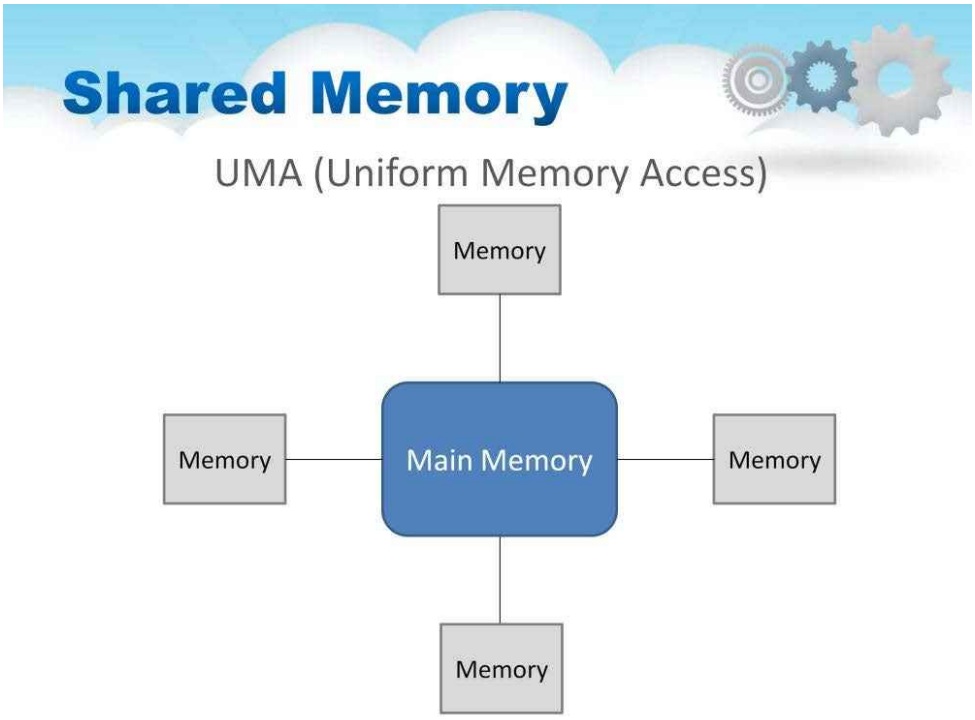
Процесоривиконують свої операції самостійно, але вони **використовувати однакові ресурси пам'яті** оскільки вони мають доступ до всього**пам'ять як глобальний адресний простір**. Якщо процесор**змінює значення в пам'яті**, тоді його спостерігатимуть усі інші процесори.

Якщо багато PU намагаються отримати доступ до одного модуля пам’яті через паралельне з’єднання мережі, пам'ять легко може стати вузьким місцем. Тому прийнято використовувати кілька модулів фізичної пам'яті**(банки пам'яті)** як показано на цьому малюнку.

Але ця архітектура називається а**централізована пам'ять**системи, тому що модулі (банк пам'яті) зібрані як одна підсистема, яка однаково доступна з усіх процесорів.

Наступне**основні класи**(перераховані на цьому малюнку) слід розглянути детально:

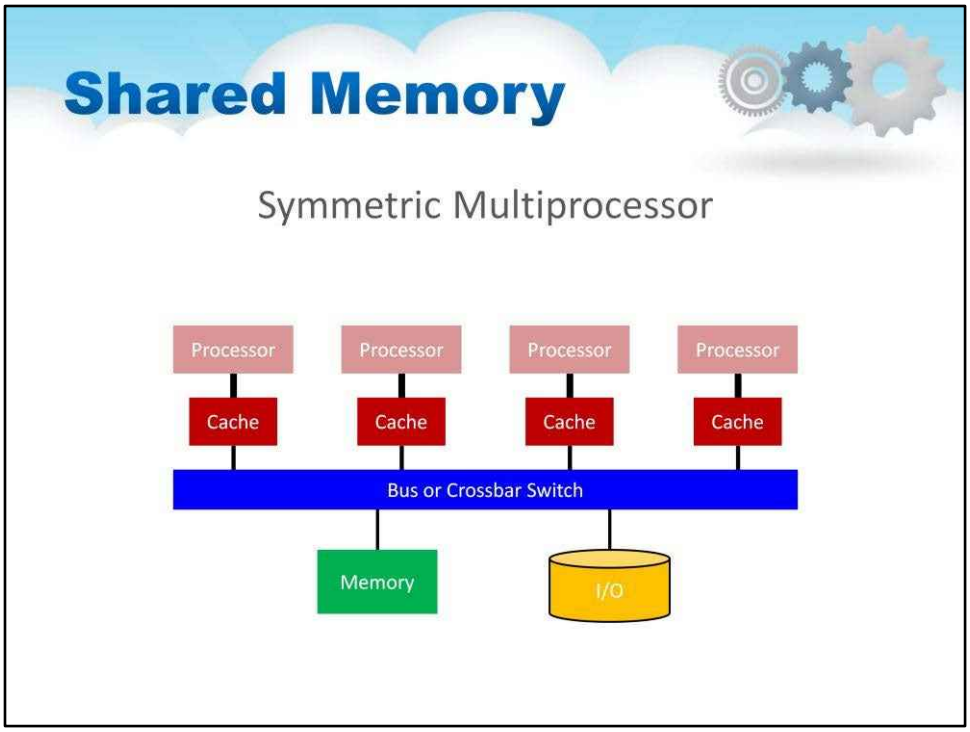
- UMA (уніфікований доступ до пам'яті)
- NUMA (нерівномірний доступ до пам'яті)



Це дає**рівний доступ**права та час доступу до пам'яті

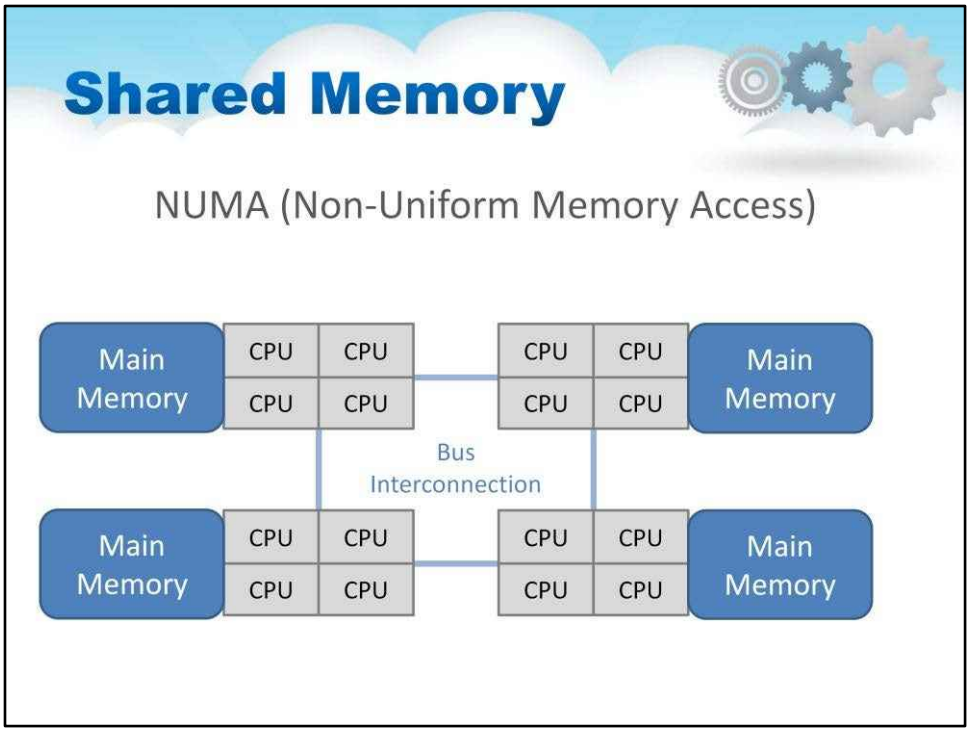
Завдяки цій однаковості доступу,
ці системи часто називають**симетричні мультипроцесори (SMP)** або **рівномірний доступ до пам'яті (UMA)**архітектури

Іноді називають**Cache Coherent UMA (CC-UMA)**



Архітектура симетричного багатопроцесорного процесора (SMP) використовує спільні системні ресурси, до яких можна отримати рівний доступ з усіх процесорів.

Зазвичай одна ОС керує машиною SMP і планує процеси та потоки на процесорах, щоб навантаження було збалансованим.

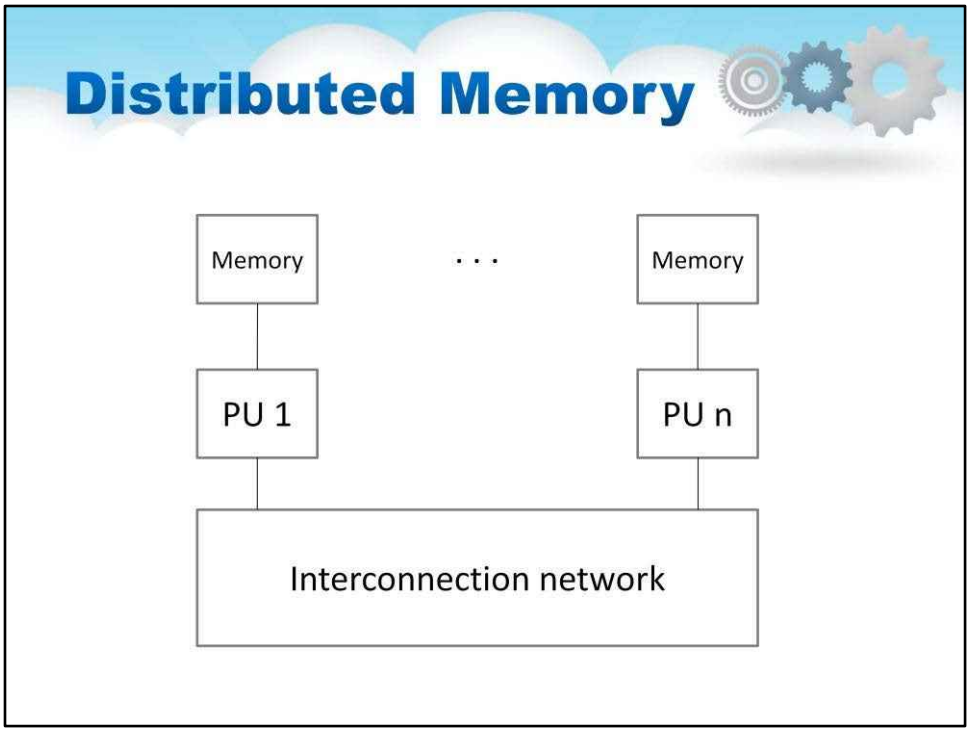


Зазвичай фізично пов’язані 2 або більше SMP,

щоб вони могли отримати прямий доступ до пам’яті один одного

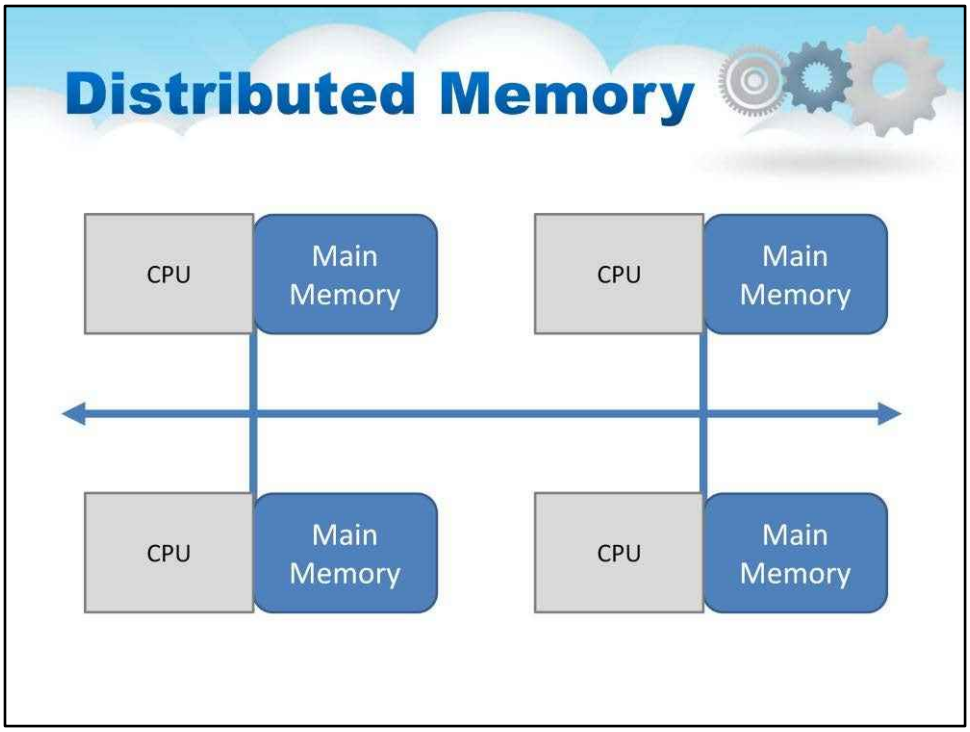
Не всі мають рівний доступчас для всіх спогадів

Доступ до пам'яті через канал набагато повільніший



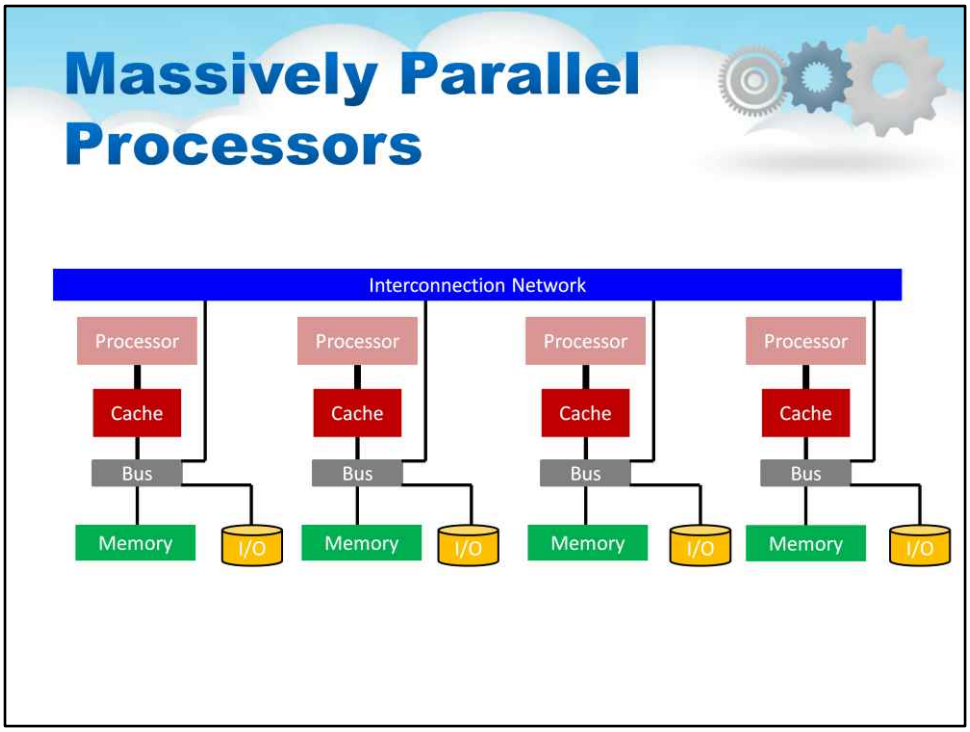
Основна альтернатива централізованій пам'яті називається**розподілена пам'ять**і показано на цьому малюнку.

Модулі пам'яті розташовані разом з процесорами.



- Кожен центральний процесор має власну локальну пам'ять.
- Зміни, внесені кожним процесором, не видно іншим.
- Процесори об'єднані мережею.
- Програма повинна визначити спосіб передачі даних між процесорами.

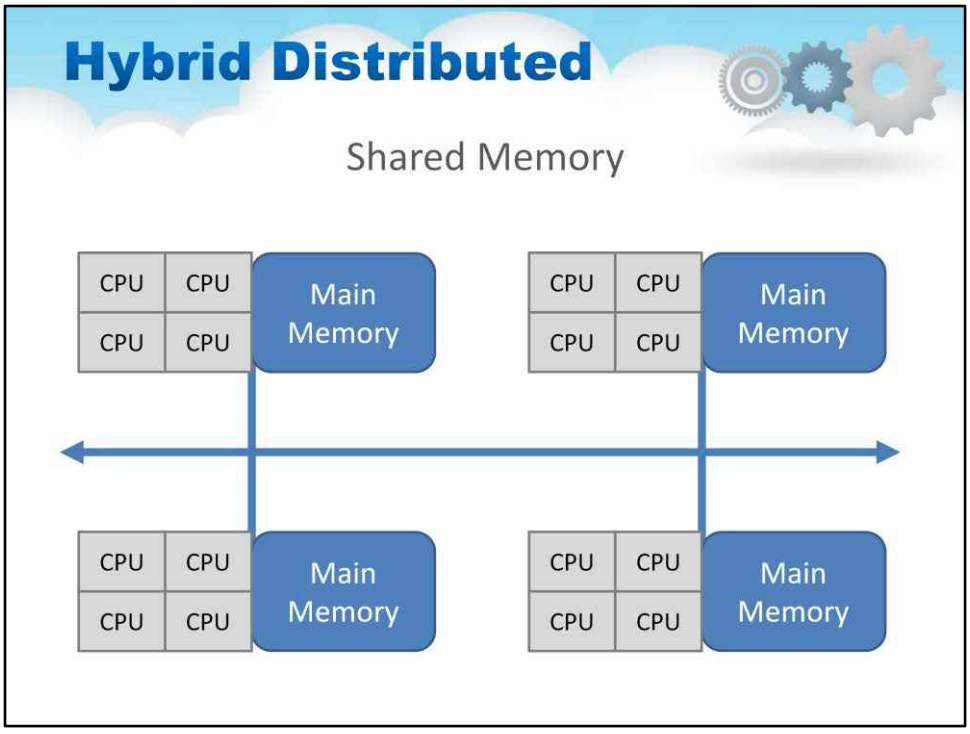
Такі системи с**архітектури розподіленої пам'яті (DM).**
називаються**масивні паралельні процесори (MPP).**



Архітектура масивних паралельних процесорів (MPP) складається з кількох вузлів.

Кожен вузол має власний процесор, пам'ять і підсистему введення-виведення.

На кожному вузлі працює незалежна ОС.



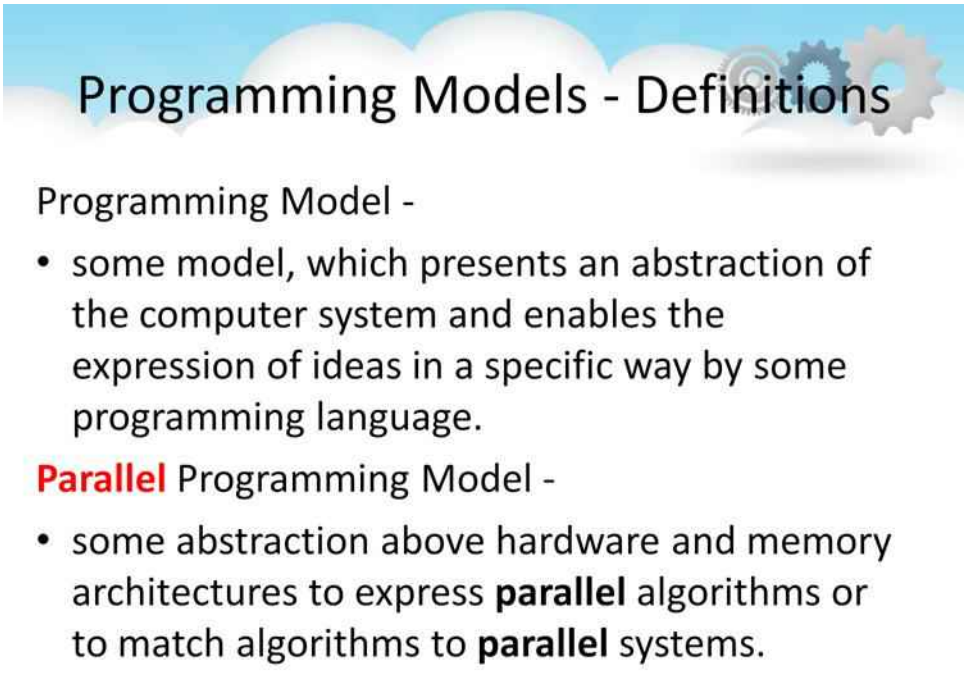
Використовується в більшості поточних паралельних систем

Компонентами спільної пам’яті є вузли SMP

Компонентом розподіленої пам'яті є мережа

Parallel Computing – Programming Models

Моделі програмування



Що таке модель програмування?

[Прочитайте 1ше речення з цього слайда]

Що таке модель паралельного програмування?

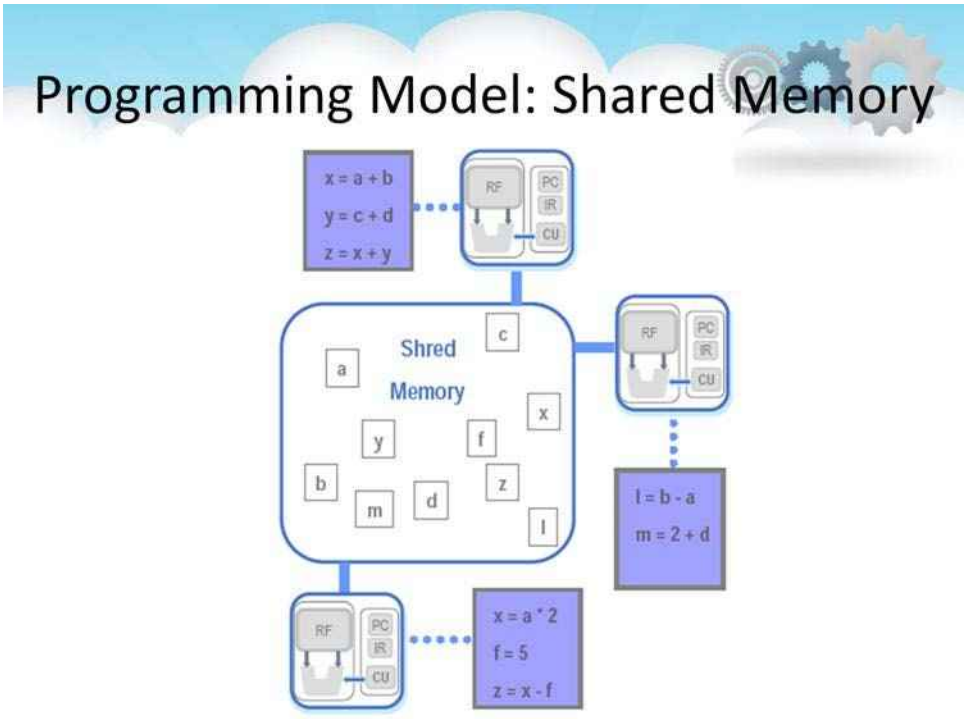
[Прочитайте 2ге речення з цього слайда]

Це визначає**як легко**програмісти можуть вказати свої алгоритми в паралельні одиниці обчислень (тобто завдання), які апаратне забезпечення розуміє

Це визначає**наскільки ефективно**на апаратному забезпеченні можна виконувати паралельні завдання

Основна мета:**використовувати все**процесори базової архітектури (наприклад, SMP, MPP, CMP) і**мінімізувати час**вашої програми

- Shared
- Message Passing



Процесори читають/записують змінні, що зберігаються в загальному адресному просторі, асинхронно.

Доступ до спільної пам'яті контролюється деякими механізмами (замки/семафори)

Перевага:

Розробку програми можна спростити,

оскільки жоден процес не володіє даними, що зберігаються в пам’яті.

Оскільки всі процесори можуть мати доступ до однакових змінних,

звернення до даних, що зберігаються в пам'яті, подібне до традиційних однопроцесорних програм.

-

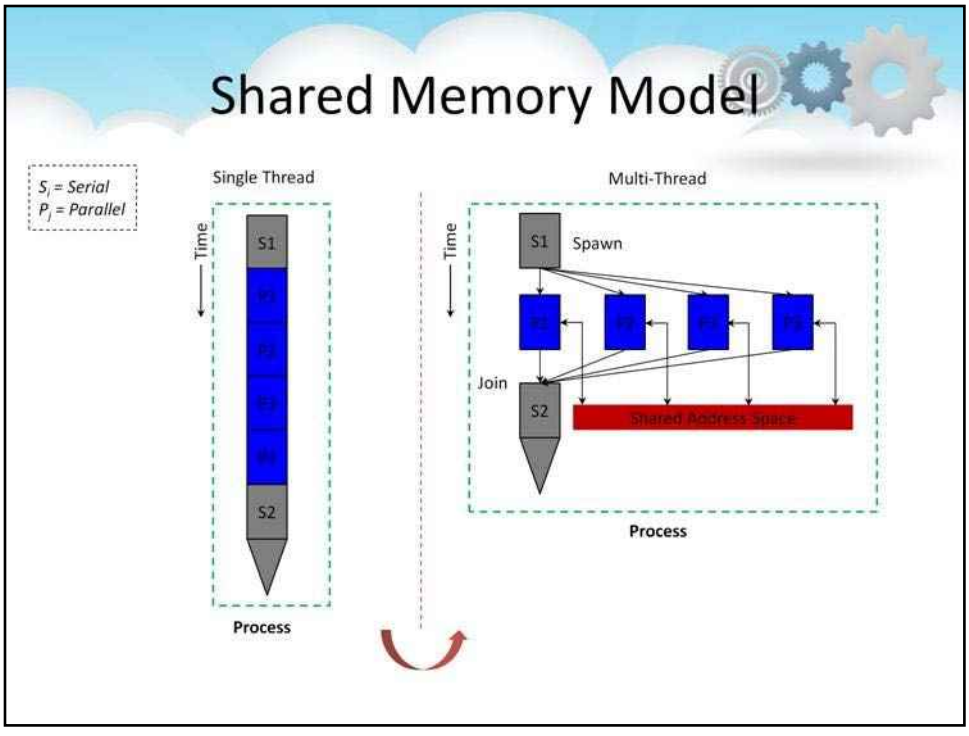
Недолік:

важко зрозуміти локальність даних і керувати нею.

Дані зберігаються локально в процесорі,

який працює над ним, зберігає доступ до пам’яті цього процесора.

Це викликає трафік шини, коли кілька процесорів намагаються отримати доступ до тих самих даних.

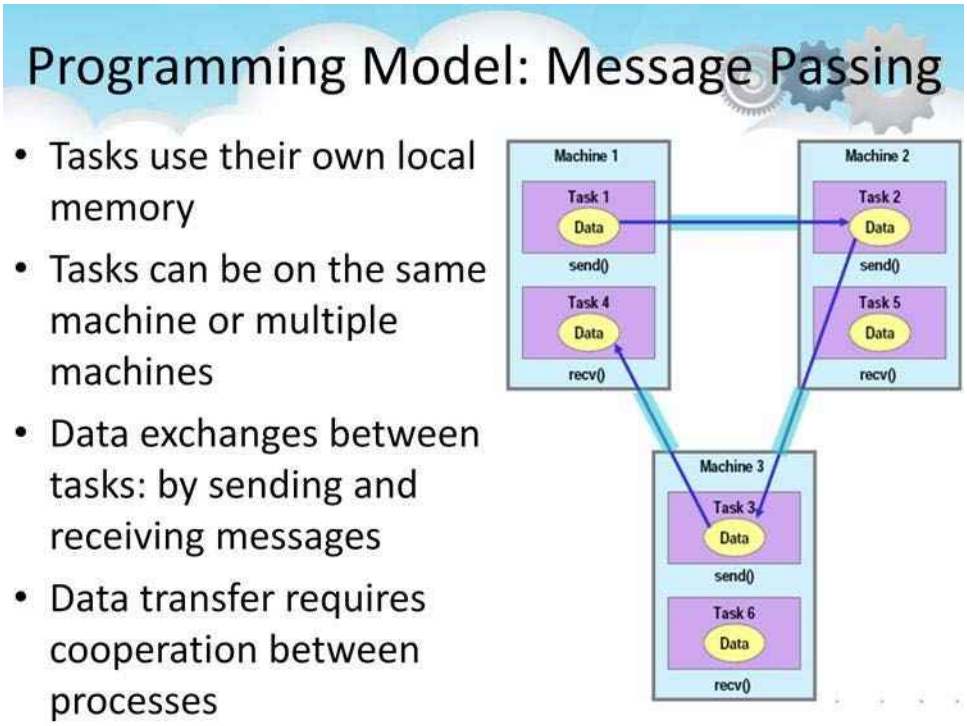


У моделі програмування спільної пам’яті абстракція полягає в тому, що паралельні завдання можуть отримати доступ до будь-якого місця пам’яті

Паралельні завдання можуть обмінюватися даними за допомогою читання та запису загальних місць пам’яті

Це схоже на потоки з одного процесу, які спільно використовують один адресний простір

Багатопотокові програми (наприклад, програми OpenMP) найкраще підходять до моделі програмування спільної пам’яті



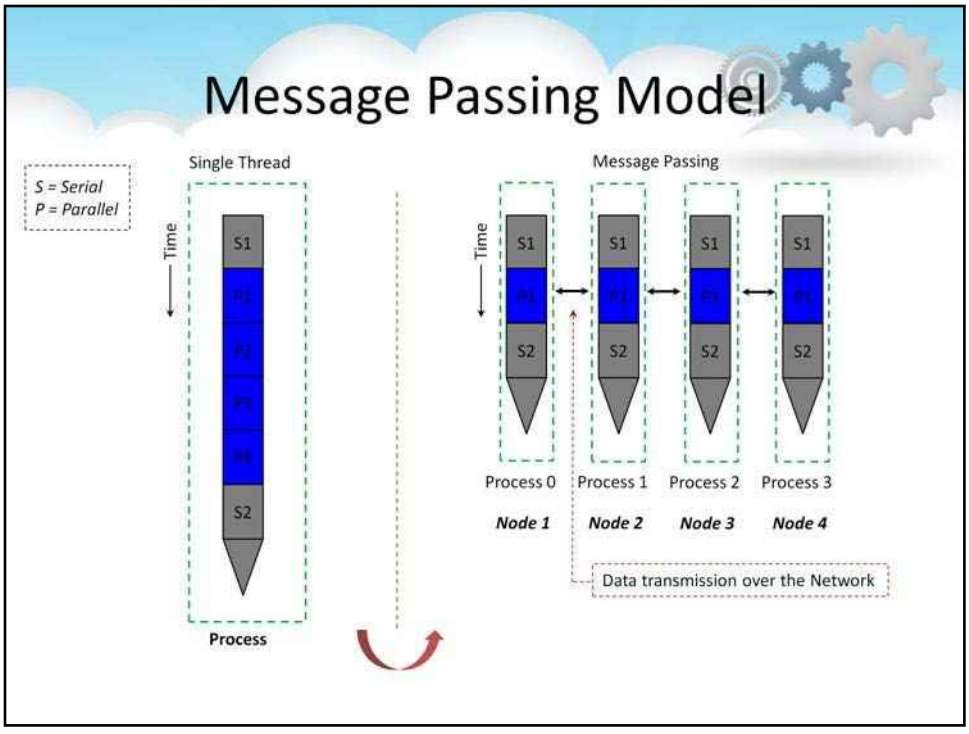
При передачі повідомлень паралельні завдання мають власну локальну пам’ять

Одне завдання не може отримати доступ до пам’яті іншого завдання

Отже, для передачі даних вони повинні покладатися на явні повідомлення, надіслані один одному

Це схоже на абстракцію процесів, які не мають спільного адресного простору

Програми інтерфейсу передавання повідомлень (MPI) найкраще відповідають моделі програмування передавання повідомлень


Message Passing Model
S = Serial
P = Parallel
Single Thread
Time
S1
P1
P2
P3
P4
S2
Process
Message Passing
Time
Process 0
Process 1
Process 2
Process 3
Node 1
Node 2
Node 3
Node 4
Data transmission over the Network

Shared Memory Vs. Message Passing

- Comparison between the shared memory and message passing programming models:

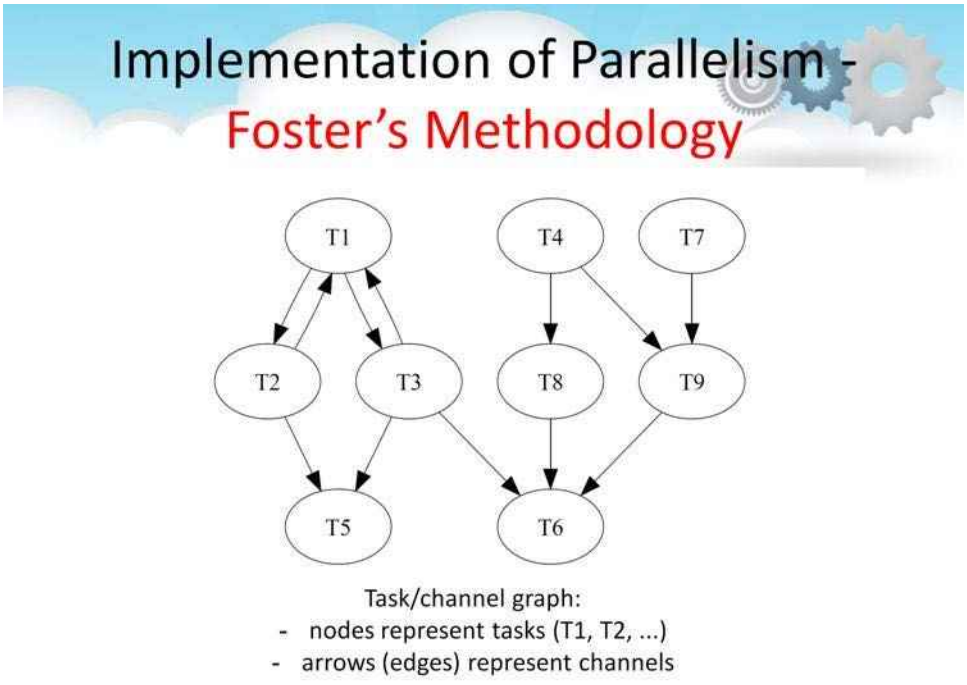
Aspect	Shared Memory	Message Passing
Communication	Implicit (via loads/stores)	Explicit Messages
Synchronization	Explicit	Implicit (Via Messages)
Hardware Support	Typically Required	None
Development Effort	Lower	Higher
Tuning Effort	Higher	Lower

Parallel Computing

-

Implementation (Design)

Проектування паралельних обчислень



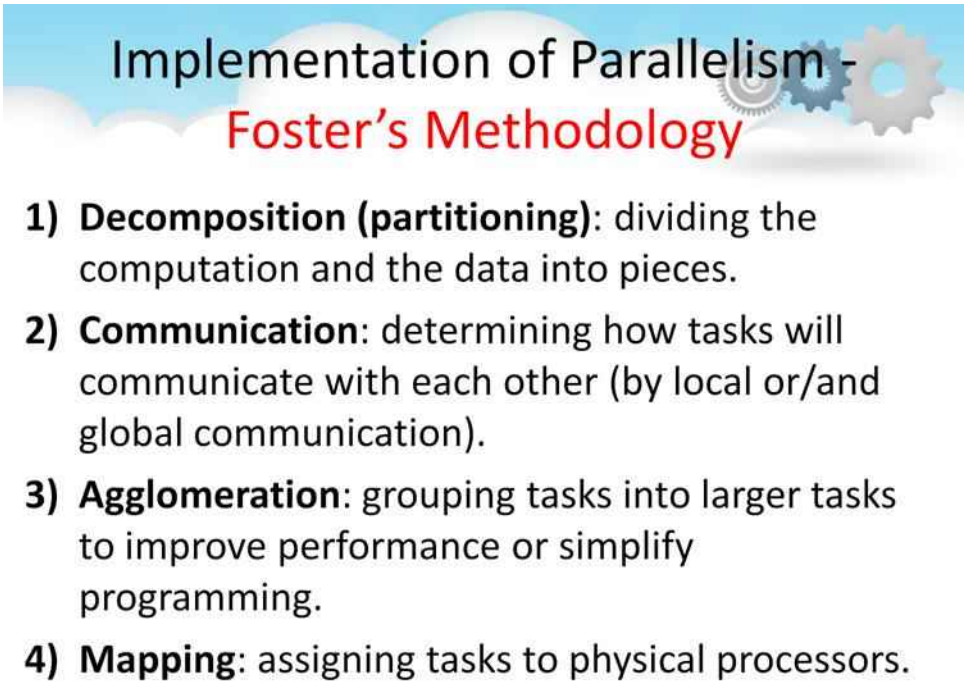
Методологію завдання/каналу, описану тут, запропонував Ян Фостер.

У цій методології паралельна програма розглядається як набір завдань, які спілкуються, надсилаючи повідомлення один одному через канали.

На цьому малюнку показано представлення завдання/каналу гіпотетичної паралельної програми.

завданняскладається з виконуваного блоку (вважайте це програмою)

Каналце черга повідомлень
який з'єднує вихідний порт одного завдання з вхідним портом іншого завдання.



Навіщо нам ця методологія?

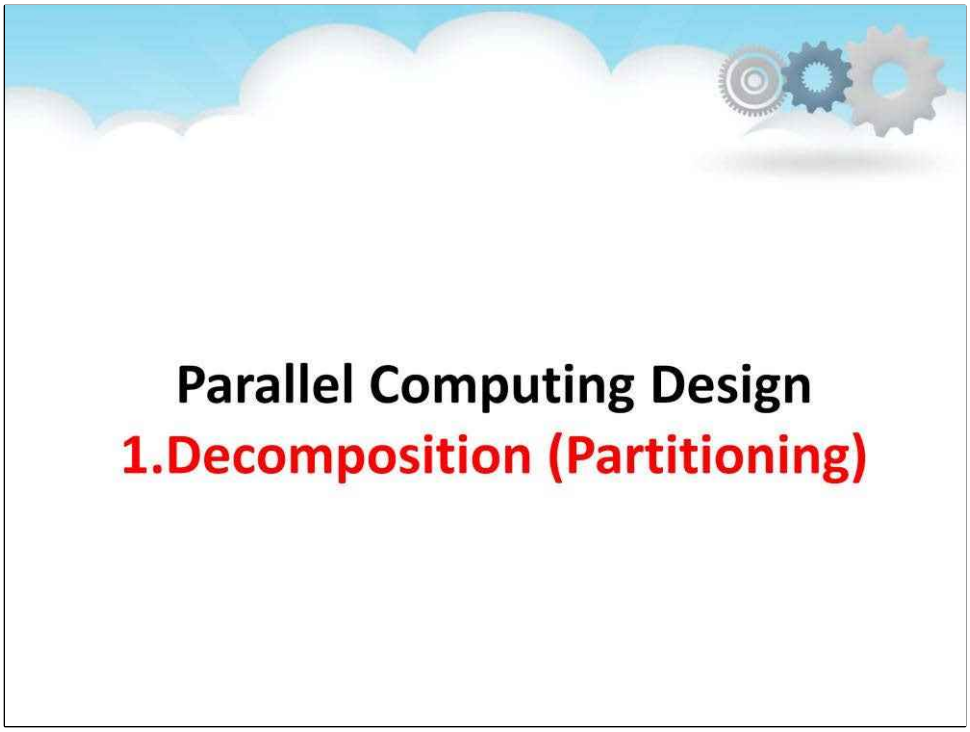
Розробити паралельну програму з нуля без певної методології нелегко.

Набагато краще використовувати перевірену методологію, яка є достатньо загальною і якої можна легко дотримуватися.

Інакше не навчишся на чужих помилках.

У 1995 році Ян Фостер запропонував таку методологію, яка
отримала назву методології дизайну Фостера.

Це чотириетапний процес проектування, як показано на цьому малюнку з їх короткими описами.



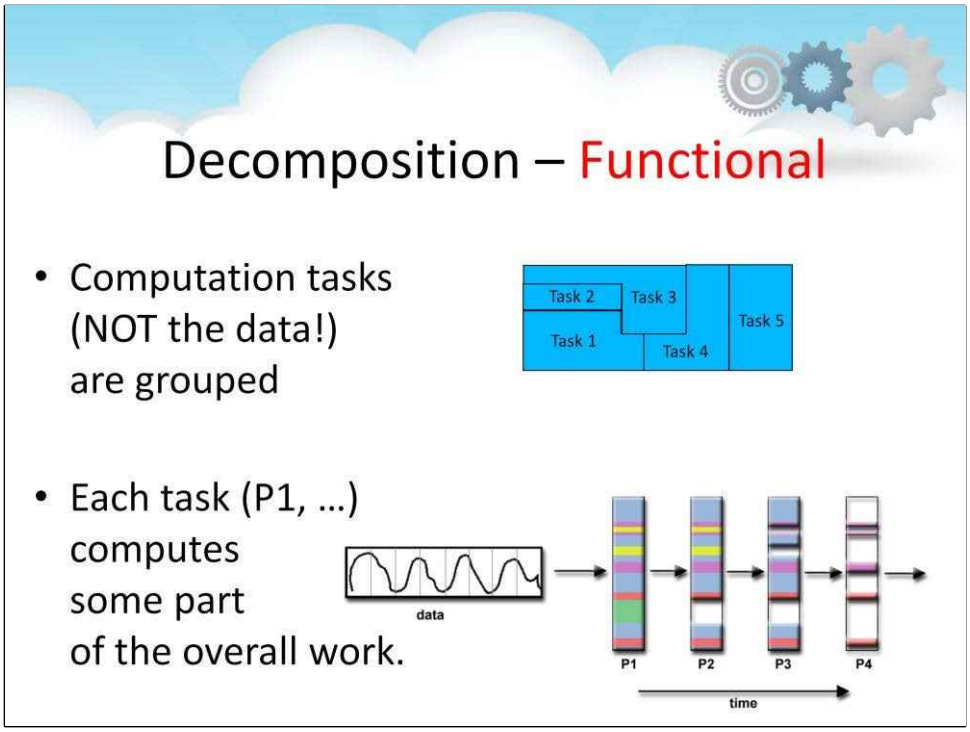
розкладання

Decomposition (Partitioning)

Definition:

Dividing the problem into chunks/parts of work that can be distributed to multiple tasks:

- Functional Decomposition
- Domain (Data) Decomposition

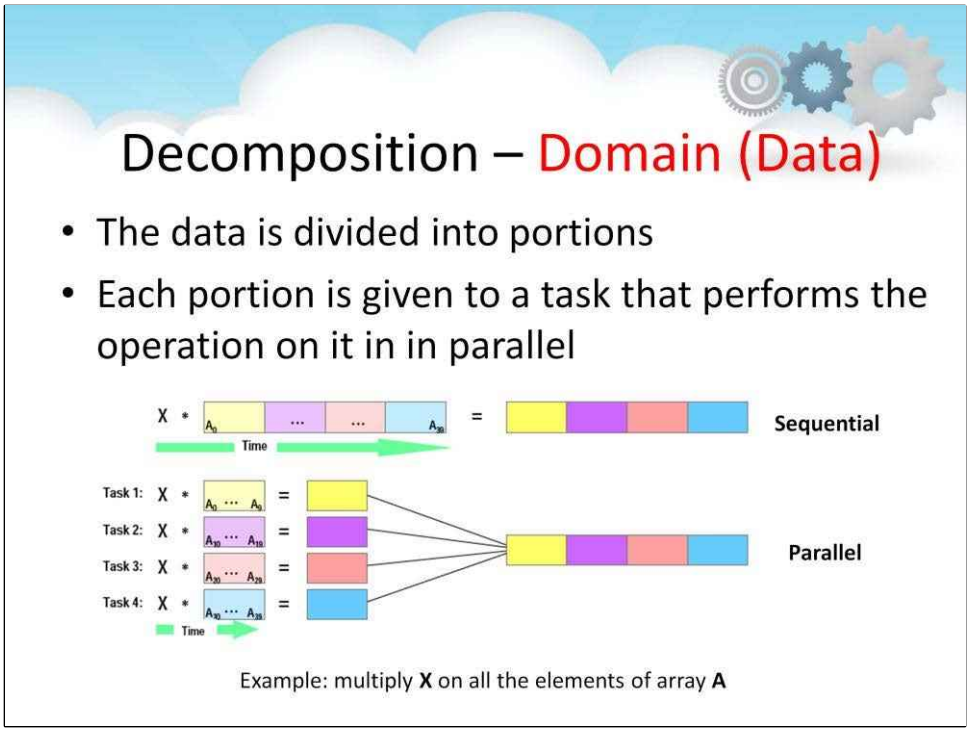


Функціональний паралелізм можливий при наявності окремих
операції, які можна застосовувати одночасно, зазвичай до різних частин набору даних.

Функціональна декомпозиція є парадигмою, в якій призначаються примітивні завдання
для окремих функцій, а потім ідентифікуються дані, до яких ці функції можна застосувати

Тут увага зосереджена на обчисленнях, які потрібно виконати, а не на даних. Проблема розкладається відповідно до роботи, яку необхідно виконати.
Потім кожне завдання виконує частину загальної роботияк показано тут**на верхньому зображенні**.

Іноді це призводить до конвеєрного підходу, як у прикладі обробки аудіосигналу, як показано тут **на нижньому зображенні** для цифрової обробки сигналу.
Сигнал пропускається через чотири різні фільтри. Кожен фільтр є окремим процесом.
Коли дані проходять через фільтр, вони надсилаються до наступного в рядку.
Це так званий а**обчислювальний конвеєр**.

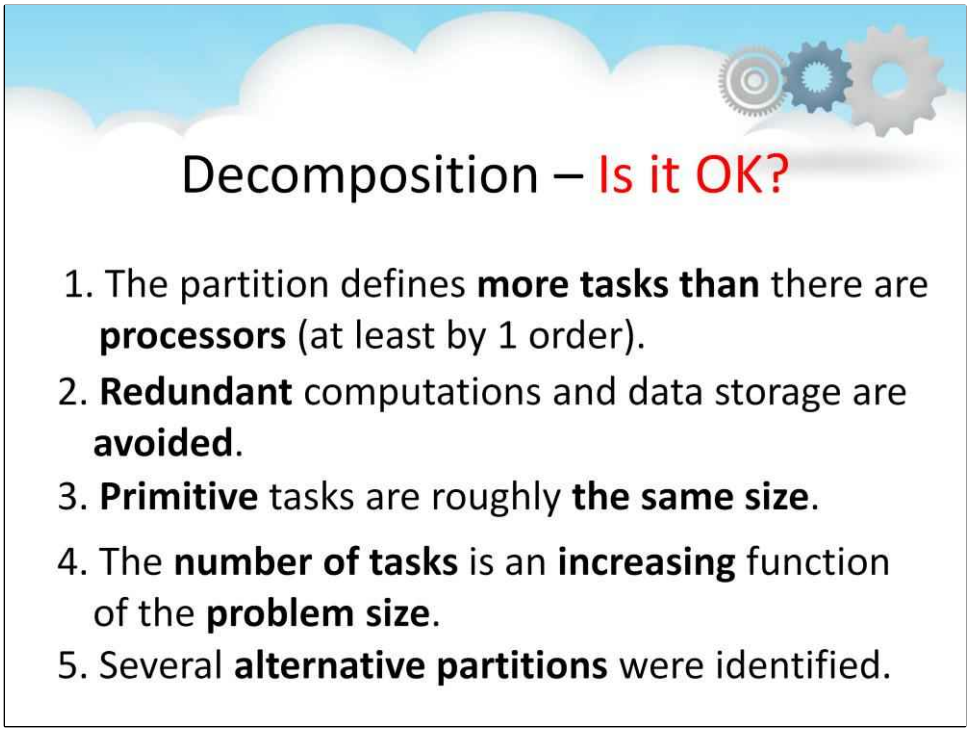


Одна і та ж операція виконується над різними частинами однієї структури даних, тобто кожен процесор виконує завдання своєї частини даних.

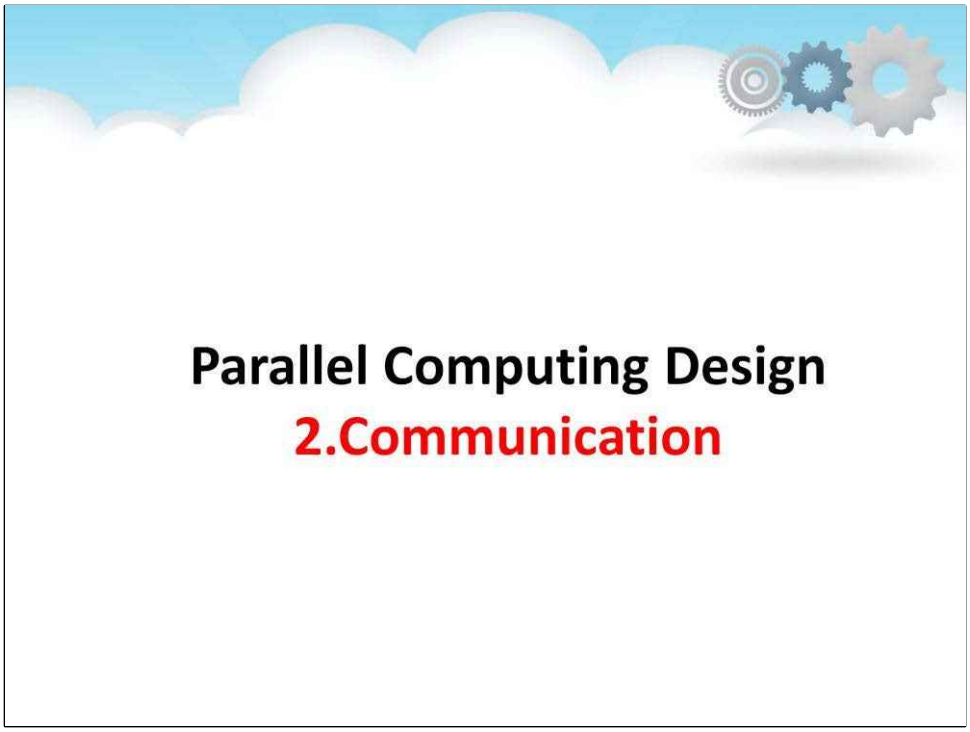
- Дані поділяються на фрагменти, над кожним фрагментом виконуються операції одночасно

- Набір завдань виконується на одній структурі даних,
але на іншій частині цієї структури

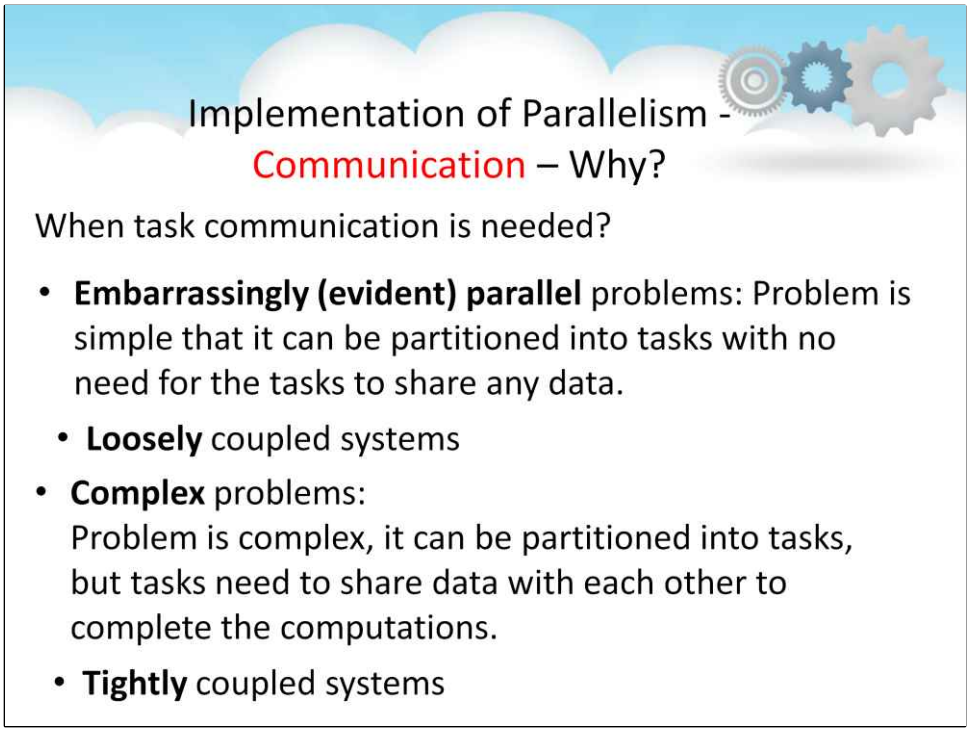
- Завдання в тій самій частині структури даних виконують однакові операції з кожним екземпляром цих даних.



Фостер надає контрольний список для оцінки етапу розділення.
Ваш розподіл має якомога більше відповідати таким критеріям: 1. Розділ визначає щонайменше на порядок більше завдань, ніж процесорів у вашому цільовому комп’ютері.
Якщо ні, у вас буде мало гнучкості на наступних етапах проектування.
2. Наскільки це можливо, уникають надлишкових обчислень і зберігання даних. Якщо ні, отриманий алгоритм може не впоратися з великими проблемами. 3. Примітивні завдання мають приблизно однаковий розмір.
Якщо ні, може бути важко розподілити для кожного процесора однакову кількість роботи, і це спричинить загальне зниження продуктивності.
4. Кількість завдань є зростаючою функцією розміру проблеми. В ідеалі збільшення розміру проблеми має збільшити кількість завдань, а не розмір окремих завдань. Якщо це не так, ваш паралельний алгоритм, можливо, не зможе вирішити більші проблеми, коли доступно більше процесорів. 5. Ви визначили кілька альтернативних розділів.
Ви можете максимізувати гнучкість на наступних етапах проектування, розглянувши альтернативи зараз.
Не забувайте досліджувати як доменне, так і функціональне розкладання.



спілкування



Коли все обчислення є однією послідовною програмою, усі дані доступні для всіх частин програми. Коли це обчислення розділено на незалежні завдання, які можуть виконуватися в окремих процесорах,

деякі дані, необхідні для виконання завдання, можуть зберігатися в його локальній пам'яті,

але деякі з них можуть знаходитися в пам'яті інших завдань.

У результаті цим завданням може знадобитися обмін даними один з одним. Цей інформаційний потік уточнюється на етапі комунікації проекту. Цей міжзадачний зв'язок не існує в послідовній програмі; це артефакт розпаралелювання обчислень. Тому вважається цілком**накладні витрати**.

Накладні витрати — це лише термін, який означає ціну, яку ми платимо за виробництво чогось,

але це не є безпосередньо частиною того, що ми виробляємо.

Ми хочемо мінімізувати ці накладні витрати в нашому

дизайні; тому важливо його ідентифікувати.

Ми також хочемо так розробити програму

щоб затримки, спричинені цим зв’язком, були мінімальними.

Implementation of Parallelism - Communication - Visibility

- Task communication is more visible in some models (for example, Message Passing Model), and is under the programmer's control.
- In other models (for example, Data Parallel Model), communication is often done transparently with no control of the programmer.

Implementation of Parallelism - Communication - Timing

Synchronous

- "Handshaking" between tasks that are sharing data is needed; it could be done implicitly or explicitly
- **Blocking communications**: some work must be held until the communications are done

Asynchronous

- Tasks can communicate with data independently from the work they are doing
- **Non-blocking communications**

Implementation of Parallelism - Communication – Range/Scope

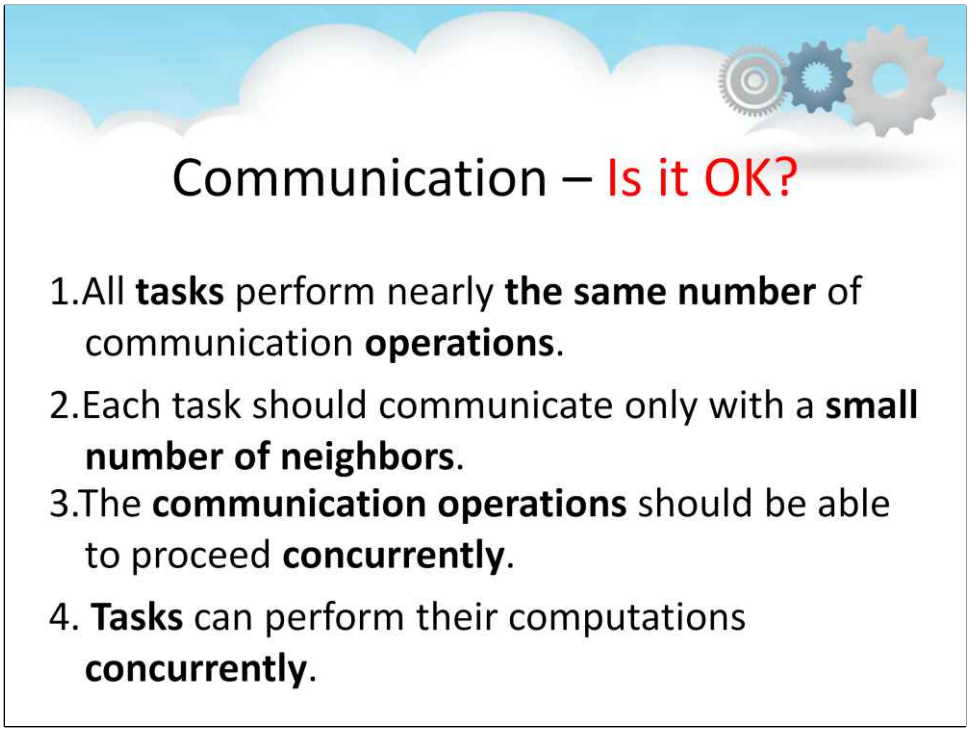
What tasks needs to communicate with each other?

Point-to-point

- Two tasks are communicating: one acts as the sender/producer of data, the other - as the receiver/consumer

Collective

- Data is communicated between more than two tasks (they are members of some common group)

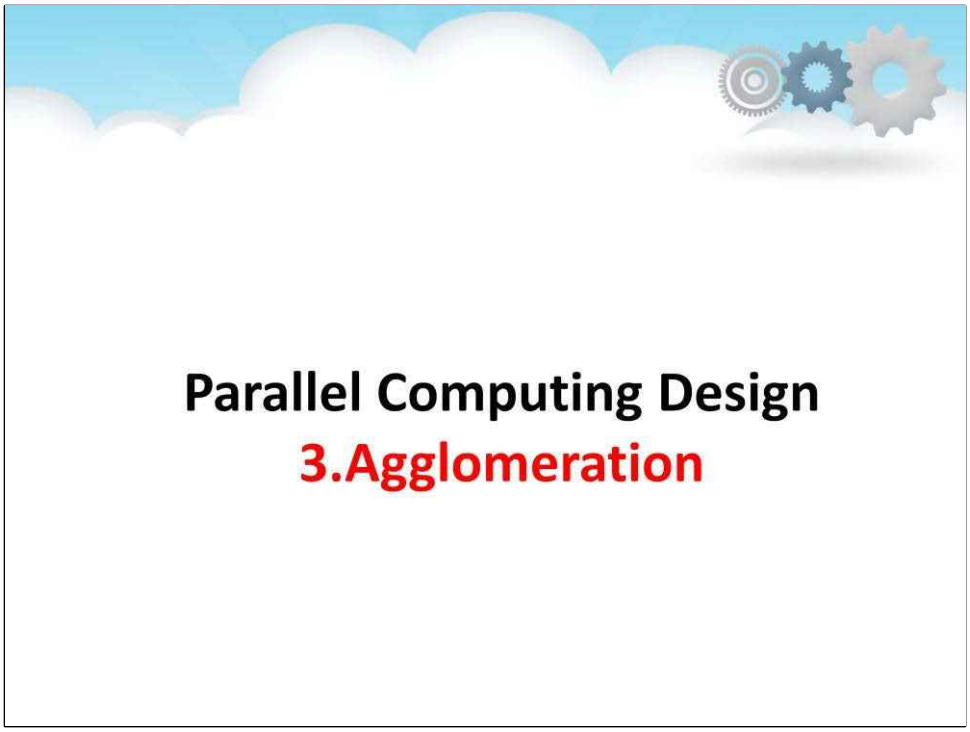


Оцінювати своє дизайнерське рішення слід за такими критеріями:

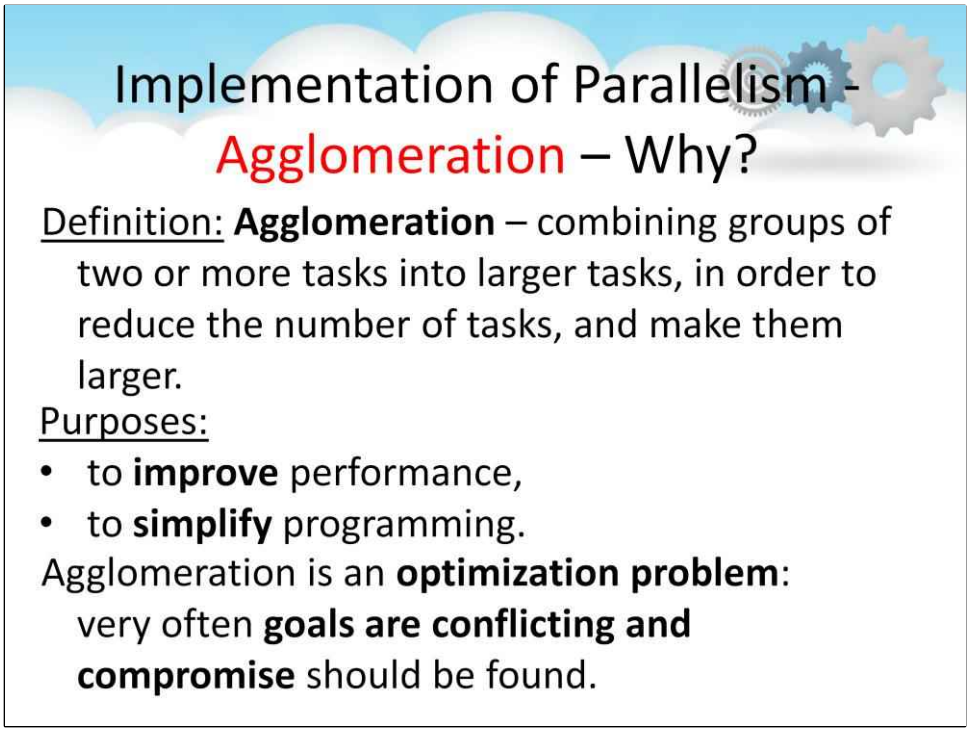
1. Усі завдання виконують приблизно однакову кількість операцій зв’язку. Незбалансовані комунікаційні вимоги передбачають дизайн що не буде масштабовано до більшого розміру екземпляра проблеми.

2. Кожне завдання має спілкуватися лише з невеликою кількістю сусідів. Якщо кожне завдання має взаємодіяти з багатьма іншими завданнями, це додасть занадто багато накладних витрат.

3. Комунікаційні операції повинні мати можливість виконуватися одночасно. Якщо ні, ваш алгоритм, ймовірно, буде неефективним і неможливим для вирішення більшої проблеми. 4. Завдання можуть виконувати свої обчислення одночасно. Якщо ні, ваш алгоритм, ймовірно, буде неефективним і неможливим для вирішення більшої проблеми.



Агломерація



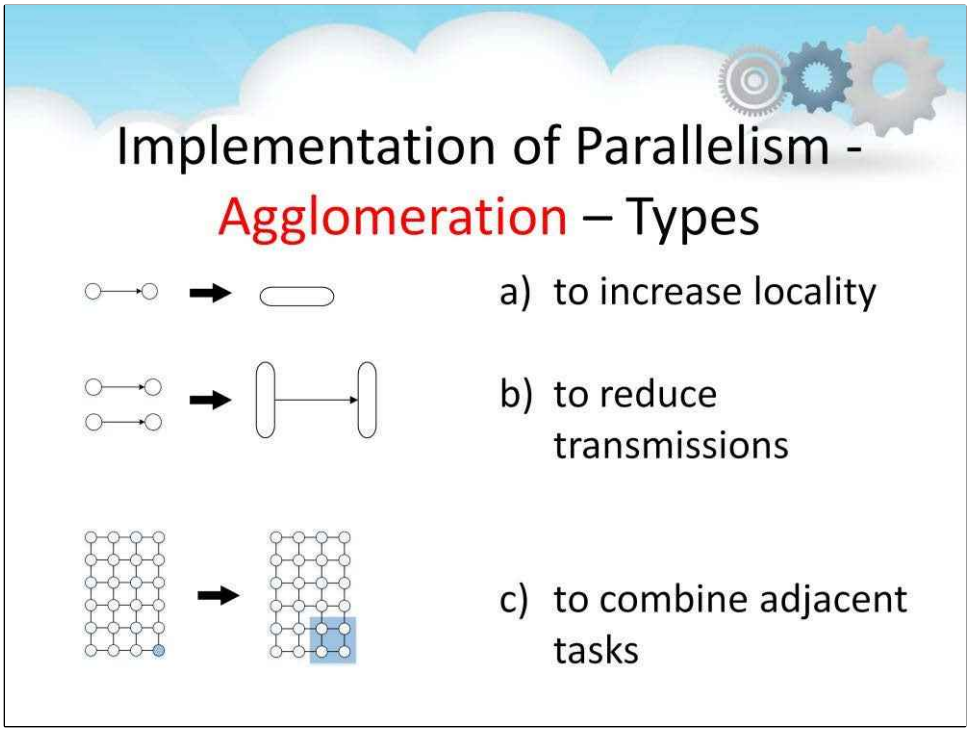
На перших двох етапах процесу проектування,
обчислення розділено для максимального паралелізму,
і зв’язок між завданнями вводиться, щоб завдання мали необхідні дані.
Отриманий алгоритм все ще є абстракцією,
оскільки він не призначений для виконання на будь-якому конкретному паралельному комп’ютері. Це також може бути дуже неефективним, особливо
якщо завдань набагато більше, ніж процесорів на цільовому комп’ютері.

Це відбувається тому:

- Створення завдання (процесу) зазвичай використовує накладні витрати, а планування завдань на процесорі використовує накладні витрати.
- Зв’язок між завданнями на одному процесорі створює штучні накладні витрати, тому що, якщо ці завдання об’єднати в одне завдання,
  такого спілкування не було б.

Навіть якщо процесорів стільки, скільки завдань,
може бути неефективність у проектуванні через зв’язок між завданнями.

На третьому етапі - агломерація,
переглядаються рішення, прийняті на етапах розділення та спілкування.



Одним із способів покращення продуктивності є збільшення деталізації, тобто зменшення накладних витрат на зв’язок.

а) до**збільшення локальності**

Коли два завдання, які обмінюються даними, об’єднуються в одне завдання, дані, якими обмінювалися через канал, є частиною одного завдання та цей канал і накладні витрати видаляються. Це називається збільшенням локальності. На рисунку А показано, як два завдання можна об’єднати в одне, щоб збільшити локальність. б) **щоб зменшити передачі**

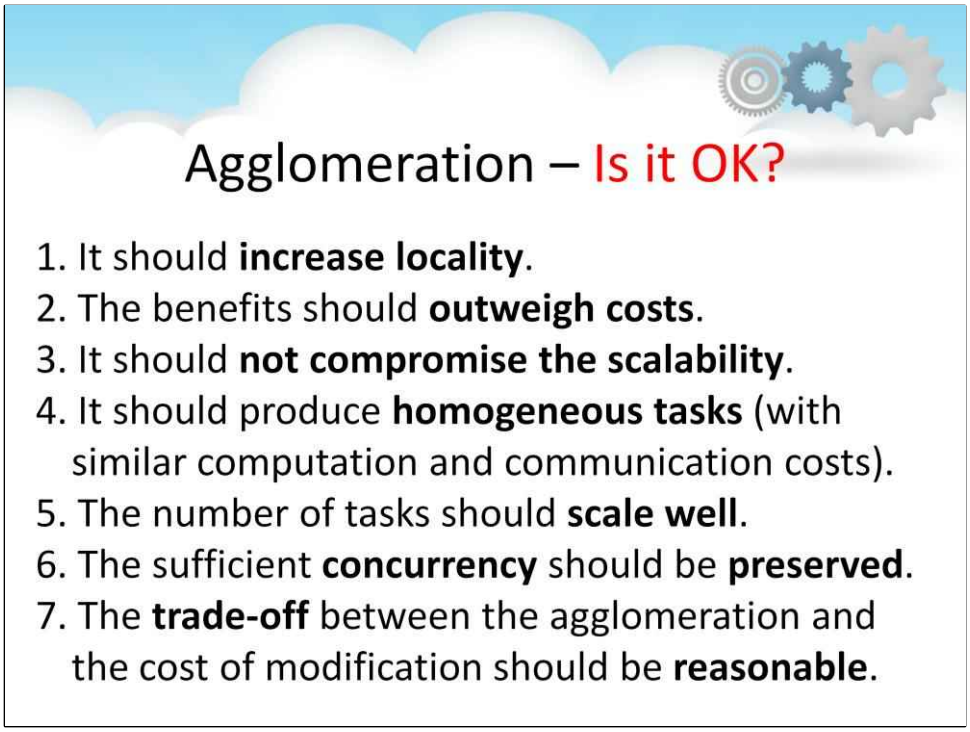
Другий спосіб зменшити витрати на зв’язок – об’єднати групи завдань, які надсилають, і групи завдань, які отримують дані одна від одної.

Вартість надсилання повідомлення складається з двох компонентів: початкового часу запуску, який називається затримкою, який не залежить від розміру повідомлення, і часу передачі, який є функцією кількості надісланих байтів. Час передачі не зменшується, але ми скорочуємо загальну затримку вдвічі. Рисунок В ілюструє цей тип агломерації.

в)**поєднувати суміжні завдання**

Третій спосіб агломерації полягає в об'єднанні суміжних завдань без зменшення розмірності рішення, але для збільшення гранулярності.

На рисунку С показано, як матриця 6*4 була розділена так, що кожне завдання має один матричний елемент, можна об’єднати, призначивши чотири елементи кожному завданню. Якби цільова архітектура мала лише шість процесорів, це могло б бути хорошим рішенням.



Ви повинні оцінити, наскільки добре ваша агломерація за такими критеріями: 1. Агломерація повинна зменшити витрати на комунікацію за рахунок збільшення місцевості.

2. Якщо агломерація повторила обчислення,

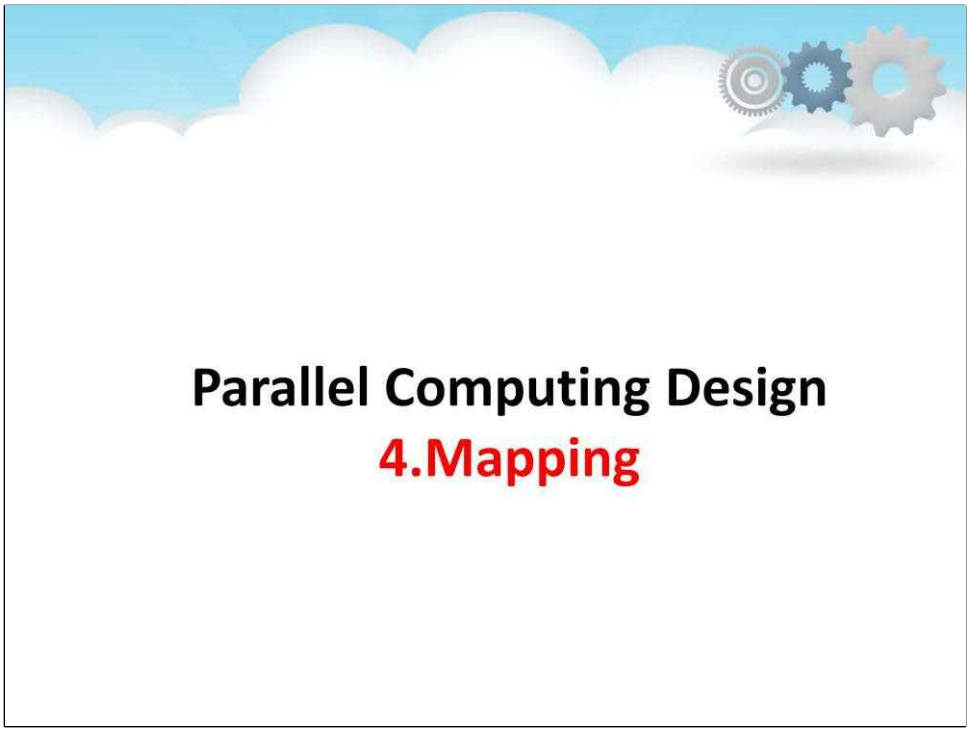
Переваги цієї реплікації повинні перевищувати її витрати для діапазону розмірів проблеми та кількості процесорів.

3. Якщо агломерація повторює дані, це не повинно шкодити масштабованості алгоритму, обмежуючи діапазон розмірів проблеми або процесорів, які він може вирішити. 4. Агломерація повинна створювати завдання з однаковими витратами на обчислення та комунікацію.

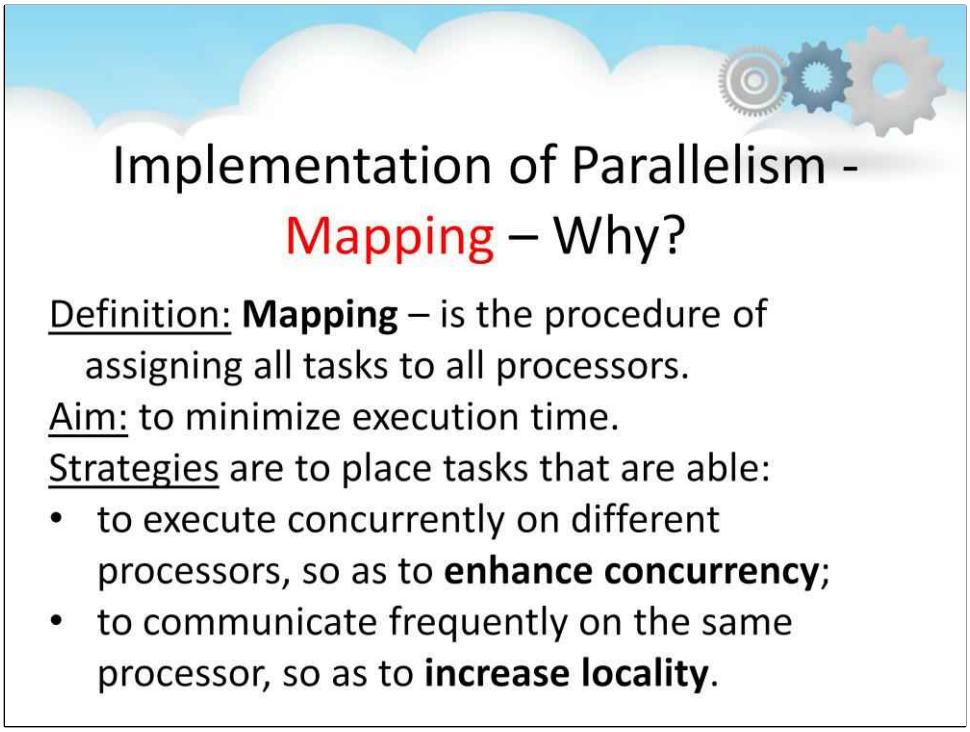
5. Кількість завдань все ще може масштабуватися залежно від розміру проблеми.

6. Є достатній паралелізм для поточних і майбутніх цільових комп’ютерів.

7. Компроміс між обраною агломерацією та вартість модифікації існуючого послідовного коду розумна.



Картографування



Відображення, завершальний етап методології Фостера, являє собою процедуру призначення кожного завдання процесору.

Звичайно, ця проблема відображення не виникає в однопроцесорних системах або на комп'ютерах зі спільною пам'яттю, операційні системи яких забезпечують автоматичне планування завдань

Тут ми припускаємо, що цільовою архітектурою є паралельний комп'ютер з розподіленою пам'яттю.

Метою розробки алгоритмів відображення є мінімізація загального часу виконання.

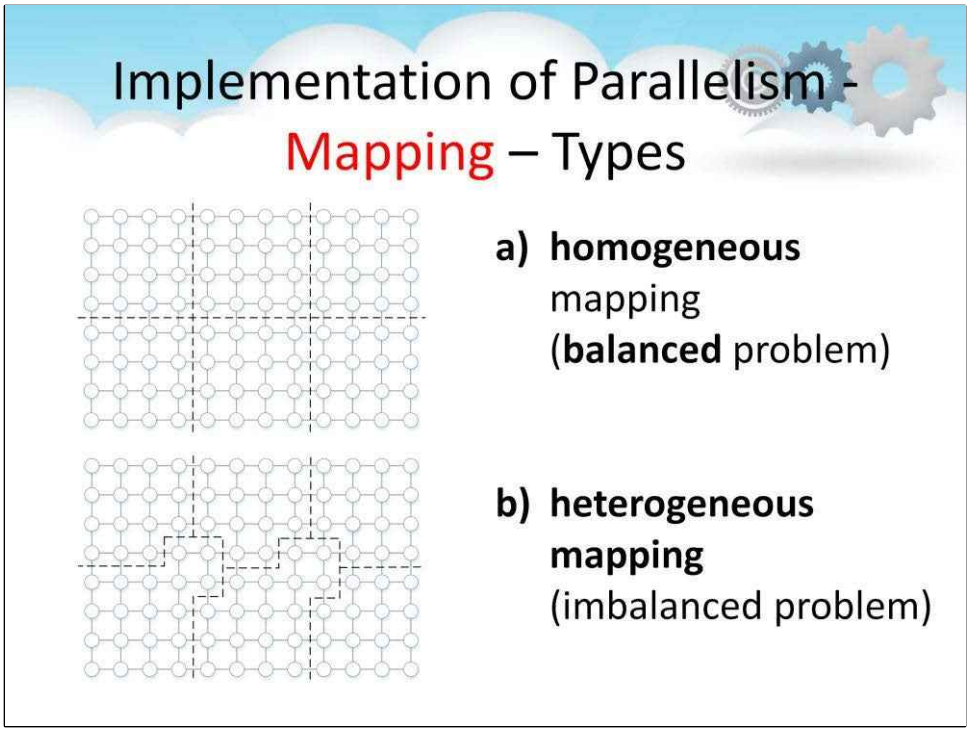
Зазвичай це досягається шляхом мінімізації міжпроцесорного зв'язку та максимального використання процесора.

Завантаження процесораце середній відсоток часу протягом які процесори комп'ютера активно виконують код, необхідний для вирішення проблеми. Він максимізується, коли обчислення збалансовано рівномірно,

тому що якщо є процесори, які простоюють, а інші зайняті, тоді це припускає, що інший дизайн міг би розвантажити роботу виконується більш зайнятими на неактивні, зменшуючи загальний час обчислення. Це не обов'язково так, але це загальна ідея.

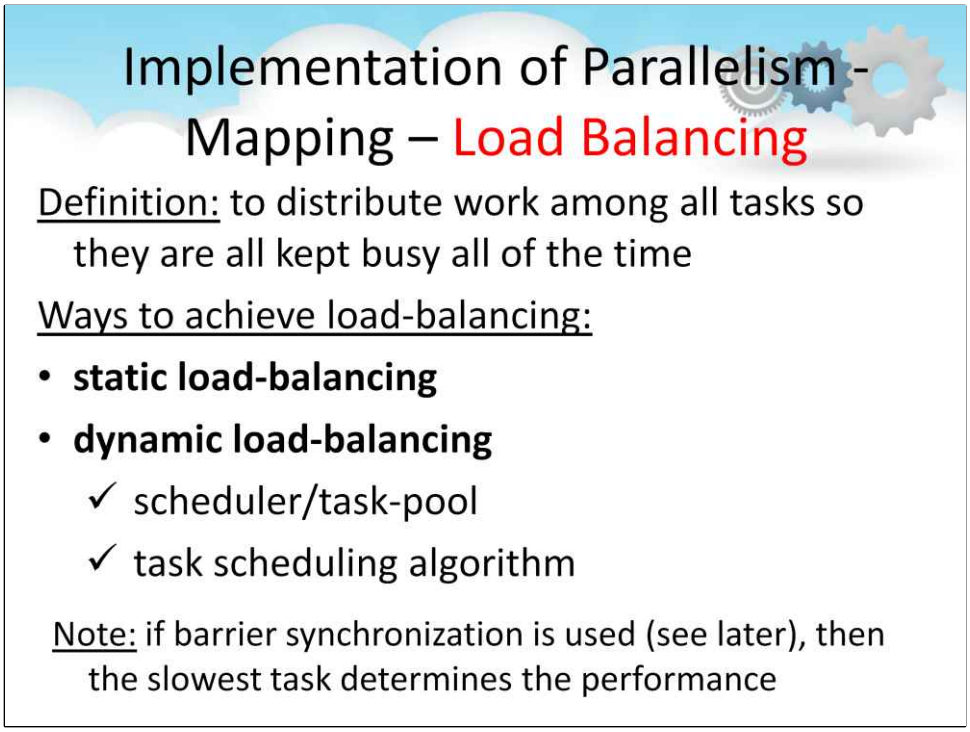
На малюнку показано дві різні стратегії мінімізації часу виконання. На жаль, іноді ці дві стратегії будуть конфліктувати, і в цьому випадку дизайн передбачатиме компроміси.

Крім того, обмеження ресурсів можуть обмежити кількість завдань, які можна розмістити на одному процесорі.



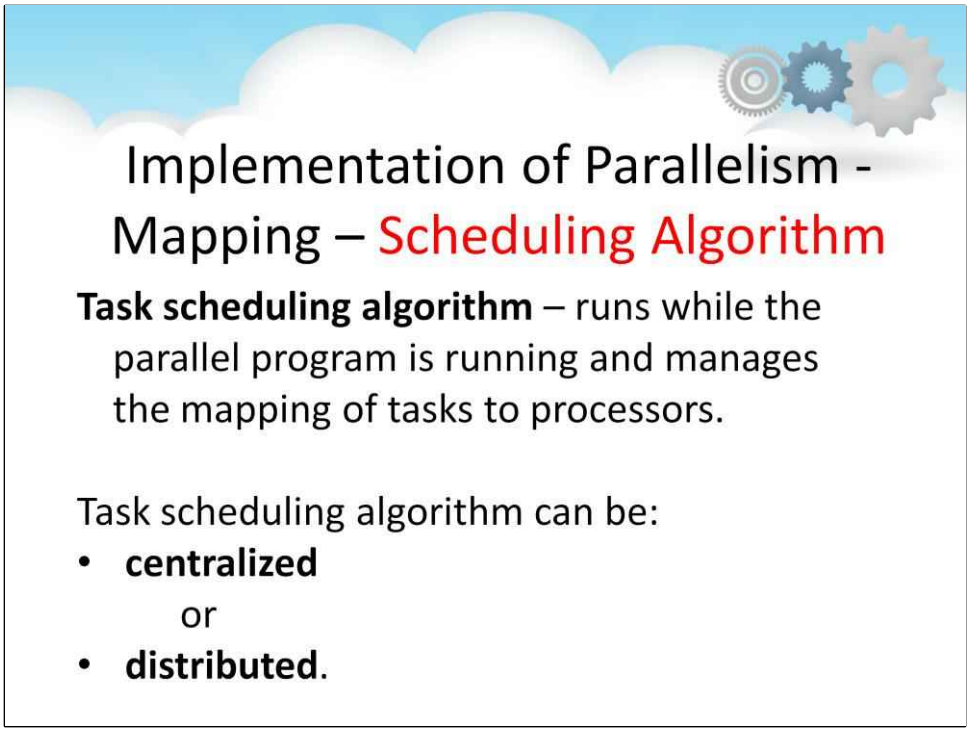
а)**однорідний**відображення
зазвичай після декомпозиції домену,
існує фіксована кількість завдань однакового розміру та подібних кроків обчислення. Природною стратегією є розподіл завдань між процесорами таким чином, щоб кожен процесор мав однакову кількість завдань,
і так, що суміжні завдання призначаються одному процесору**як показано на малюнку А**.

б)**неоднорідний**відображення
іноді деякі завдання можуть мати інше навантаження, ніж інші, наприклад, у кутах підсіток.
Це може створити а**дисбаланс навантаження**,
і а**неоднорідне відображення**може бути оптимальним (наприклад**як показано на малюнку В**).



У попередніх прикладах шаблон зв’язку був регулярним, тому рішення були регулярними.

Коли схема спілкування не регулярна, але є**відомо заздалегідь**, а**алгоритм статичного балансування навантаження**може бути використано.
Його можна застосувати під час компіляції для визначення стратегії відображення. Але іноді кількість завдань є**НЕ відомо заздалегідь**, або якщо так, то вимоги до зв’язку ні.
У будь-якому з цих випадків**динамічний алгоритм балансування навантаження**необхідно використовувати. Алгоритм динамічного балансування навантаження аналізує набір запущених завдань і генерує нове відображення завдань на процесори.



Іноді завдання нетривалі;
вони створюються на льоту для вирішення дрібних проблем, а потім їх знищують, і вони не спілкуються між собою.

У цьому випадку а**лгоритм планування завдань**виконується під час роботи паралельної програми та керує відображенням завдань на процесори.

Такий алгоритм може бути

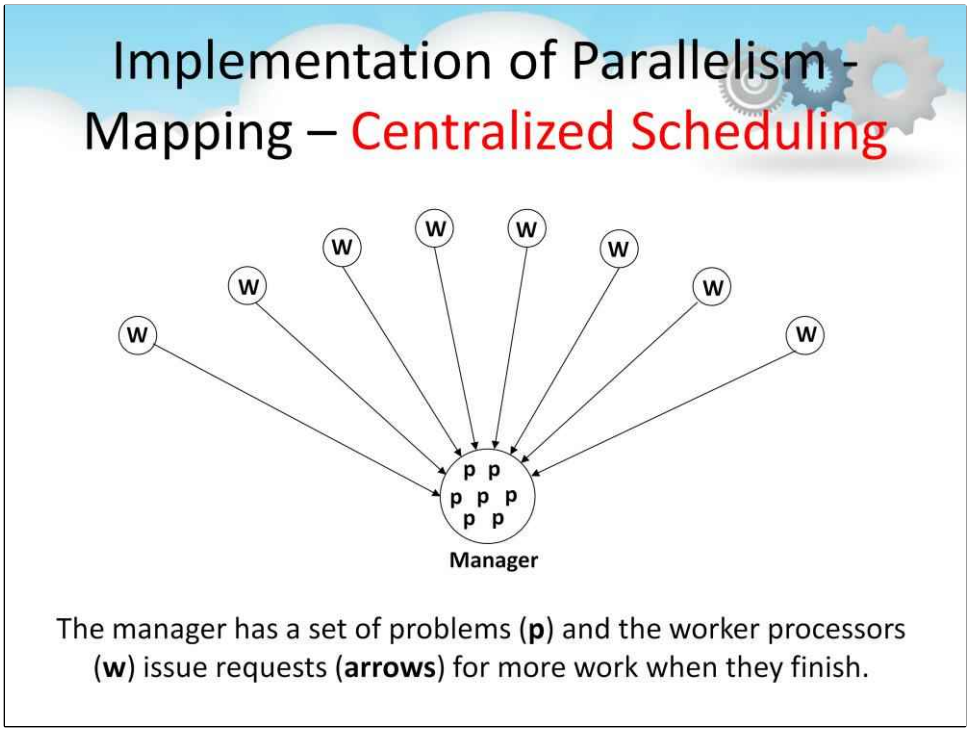
- централізований

або

- розподілений.

Загалом можна використовувати алгоритми планування завдань

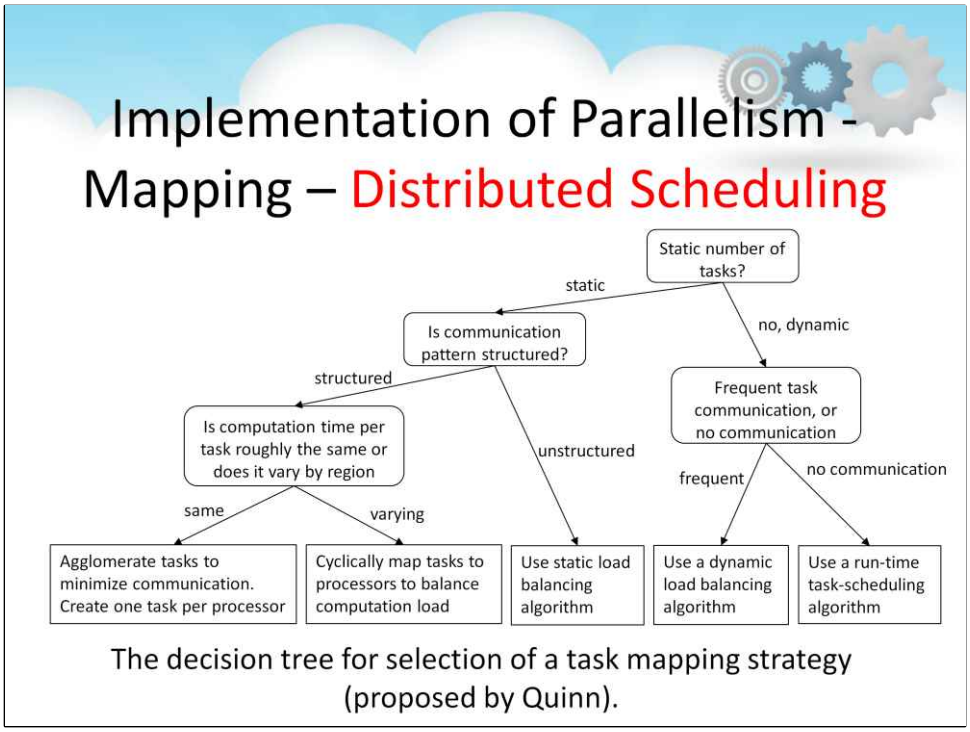
коли функціональна декомпозиція дає багато завдань, кожна з яких має слабкі вимоги до локальності.



У централізованому алгоритмі планування завдань:

- один процесор стає а **менеджер** і
- інші процесори використовуються для роботи **робітник** завдання.

Підтримується пул проблем, до якого поміщаються нові проблеми та з яких проблеми беруться для розподілу на робочі процесори. Кожен робочий процесор працює для вирішення проблеми,
і коли він закінчується, він повертає свої результати менеджеру та запитує нову проблему. **Цей малюнок ілюструє цю ідею.**



Однією з проблем централізованого планувальника є те, що менеджер стає вузьким місцем.

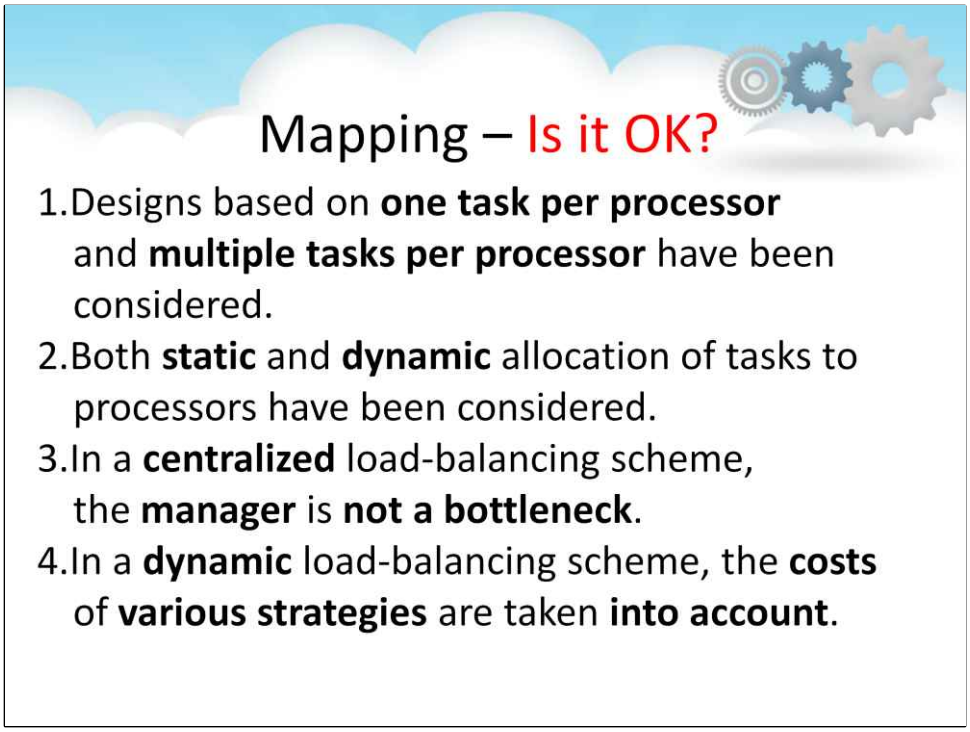
В**алгоритм розподіленого планування**, менеджера немає.

Натомість на кожному процесорі зберігається окремий пул завдань, і незадіяні працівники вимагають проблем від інших процесорів.

По суті, пул завдань — це розподілена структура даних, до якої різні завдання мають асинхронний доступ.

Це більш масштабоване рішення, ніж централізоване.

Квінн представляє а**дерево рішень**(показано на цьому малюнку), щоб вирішити, як зіставити завдання з процесорами.



Наступний контрольний список можна використовувати для неофіційної оцінки дизайну картографування:

1. Проекти на основі одного завдання на процесор
і кілька завдань на процесор були розглянуті.

2. Було розглянуто як статичний, так і динамічний розподіл завдань між процесорами.

3. Якщо вибрано централізовану схему балансування навантаження,
ви переконалися, що керівник не стане вузьким місцем.

4. Якщо вибрано схему динамічного балансування навантаження,
ви оцінили відносну вартість різних стратегій і повинні врахувати витрати на впровадження в аналізі.

Parallel Computing
-
Design
-
Synchronization

Синхронізація

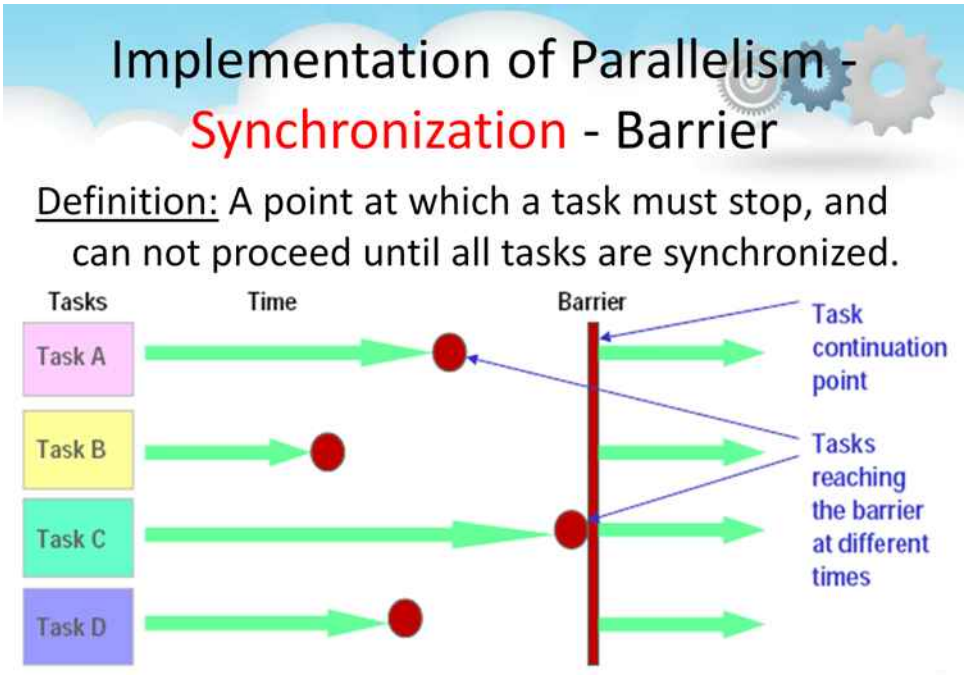
Implementation of Parallelism - Synchronization

Definition:

Reaching an agreement between simultaneous processes/tasks as to the sequence of steps to complete an action

Ways of Synchronization:

- Barriers
- Locks/ Semaphores
- Synchronous Communication Operations

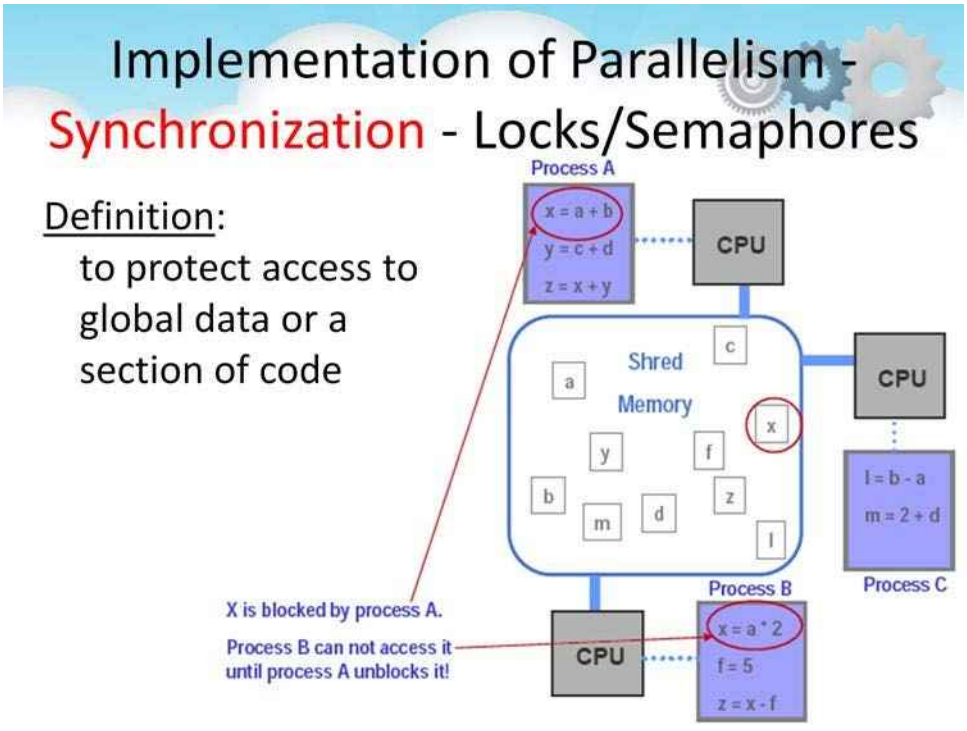


Кожне завдання продовжує працювати, поки не досягне бар'єру.

Потім він зупиняється й продовжує чекати, поки останнє завдання досягне бар’єру.

Коли останнє завдання досягає бар'єру, усі завдання синхронізуються.

З цього моменту завдання продовжують свою роботу.



Лише одне завдання одночасно може отримати доступ до блокування/семафора.

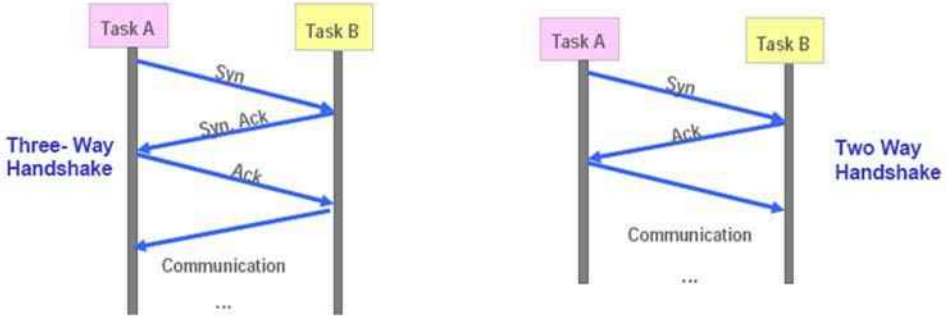
Перше завдання, яке отримує доступ до блокування, встановлює його як заблоковане та звільняє, коли завершить роботу з ним.

Коли інші завдання намагаються отримати доступ до блокування, вони не вдаються, доки завдання, яке володіє блокуванням, не звільнить його.

Може бути блокуючим або неблокуючим

Implementation of Parallelism - Synchronous Communication Oper-ns

Definition: Coordination is required between the task that is performing an operation and the other tasks performing the communication



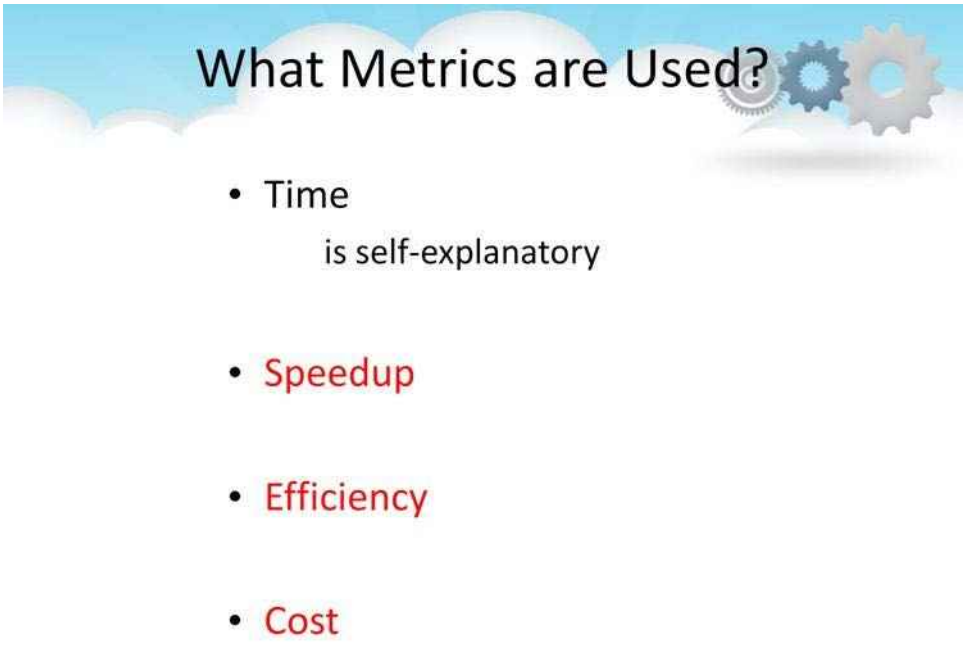
Parallel Computing
-
Performance

Продуктивність паралельних обчислень

Не менш важливо знати витрати та переваги розпаралелювання.

Метою цього розділу є розгляд середніх (показників):

- визначити, наскільки швидше працює паралельний алгоритм, ніж послідовний;
- прогнозувати продуктивність паралельного алгоритму;
- щоб вирішити, чи існують внутрішні обмеження
  які перешкоджають роботі паралельного алгоритму, і що це таке;
- щоб визначити, наскільки ефективним може бути паралельний алгоритм
  оскільки розмір проблеми та кількість доступних процесорів зростає.



Метрики

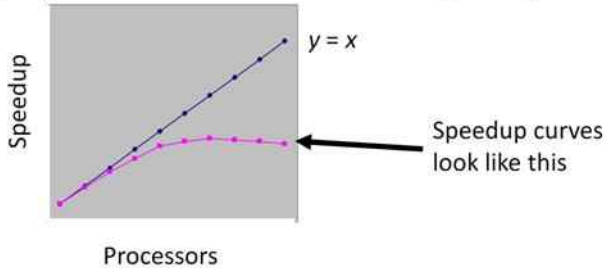
Наступні показники зазвичай використовуються для оцінки продуктивності паралельних програм...

Speedup - Definition

Definition: the ratio $\Psi(n,p)$ between sequential execution time and parallel execution time (for data size n and p processors):

$$\text{Speedup} = \Psi(n,p) = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$

Example: sequential program executes in 6 seconds and the parallel program executes in 2 seconds -> speedup is 3.



Ми використовуємо термін «прискорення», щоб вказати, наскільки паралельна програма швидша за послідовну.

Запитання: що це означає, якщо прискорення > 1?

Відповідь: це означає, що паралельна програма виконується швидше, ніж послідовна програма.

Питання: що це означає, якщо прискорення < 1?

Відповідь: це означає, що паралельна програма виконується повільніше, ніж послідовна програма.

Запитання: для заданого розміру проблеми прискорення часто зростає зі збільшенням кількості процесорів, але воно завжди «відступає» і починає знижуватися. чому

Відповідь: Накладні витрати, пов’язані зі створенням, керуванням і завершенням паралельних потоків, зростають із збільшенням кількості потоків. Обсяг роботи, яку повинен виконати кожен потік, стає меншим із збільшенням кількості потоків. У якийсь момент накладні витрати, пов’язані з новим потоком, перевищують економію часу, досягнуту додаванням нового потоку. У цей момент час паралельного виконання збільшиться, а крива прискорення опуститься.

Speedup - Notes

$$\Psi(n,p) = t_s/t_p$$

- In practice:
  - t_s is the execution time on a single processor, using the fastest known sequential algorithm
  - t_p is the execution time using *n* parallel processors.
- In theory:
  - t_s is the worst case running time for of the fastest known sequential algorithm for the problem
  - t_p is the worst case running time of the parallel algorithm using *n* processing units.

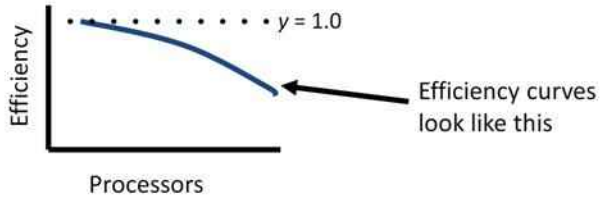
Efficiency - Definition

Definition: measure of processor utilization $\varepsilon(n,p)$ as the speedup divided by the number of processors p

$$\text{Efficiency} = \varepsilon(n,p) = \frac{\text{Speedup}}{\text{Processors}}$$

Example:

Program achieves speedup of 3 on 4 CPUs
Efficiency is 3 / 4 = 75%



Ми використовуємо слово «ефективність», щоб вказати, наскільки добре ми використовуємо доступні ресурси ЦП. Ефективність 100% означає, що кожен із процесорів у паралельному обчисленні витрачає весь свій час на виконання корисної роботи, порівняно з швидкість обчислення послідовної програми.

ДЛЯ ДАНОГО РОЗМІРУ ПРОБЛЕМИ ЕФЕКТИВНІСТЬ МАЙЖЕ ЗАВЖДИ ФУНКЦІЯ СПАДАННЯ КІЛЬКОСТІ ПРОЦЕСОРІВ, ЯКИХ ВИКОРИСТАНО ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ.

Запитання: що краще --- ефективність 80% на двох процесорах чи ефективність 50% на чотирьох процесорах?

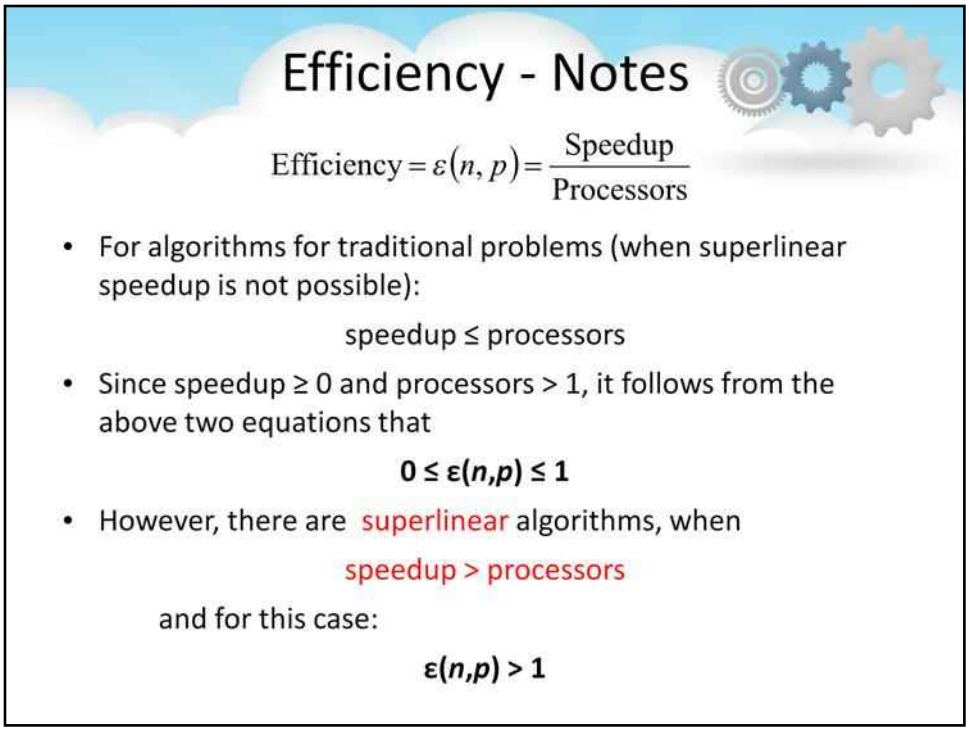
Відповідь: Ефективність = Прискорення / Кількість процесорів

Прискорення = Ефективність * Кількість процесорів⇒

Ефективність 80% на двох процесорах означає прискорення на 1,6.

Ефективність 50% на чотирьох процесорах означає прискорення 2,0.

Ефективність 50% на чотирьох процесорах краща, оскільки прискорення вище.



Це вірно, якщо чотирьом процесорам більше нічого робити. Але що, якщо у нас є два завдання, і кожне завдання має ефективність 80% на двох процесорах і 50% на чотирьох процесорах? У цьому випадку ми б ефективніше використовували процесори, якщо запускали обидва завдання одночасно і надавали кожному завданню два процесори. Це збільшить пропускну здатність системи (тобто швидкість виконання завдань).

Як це пов’язано з паралельною обробкою? Зрештою, ми сказали на початку заняття, що ми зосереджені на тому, щоб швидше виконувати певні завдання. Це актуально, якщо наша задача має потенційну декомпозицію завдання, а всередині кожної декомпозиції завдання є потенційна декомпозиція домену. Можливо, краще мати два завдання, які виконуються одночасно, кожне з двома процесорами, потім спочатку виконати одне завдання на чотирьох процесорах, а потім виконати друге завдання на чотирьох процесорах.

Cost - Definition

Cost = Parallel running time × processors

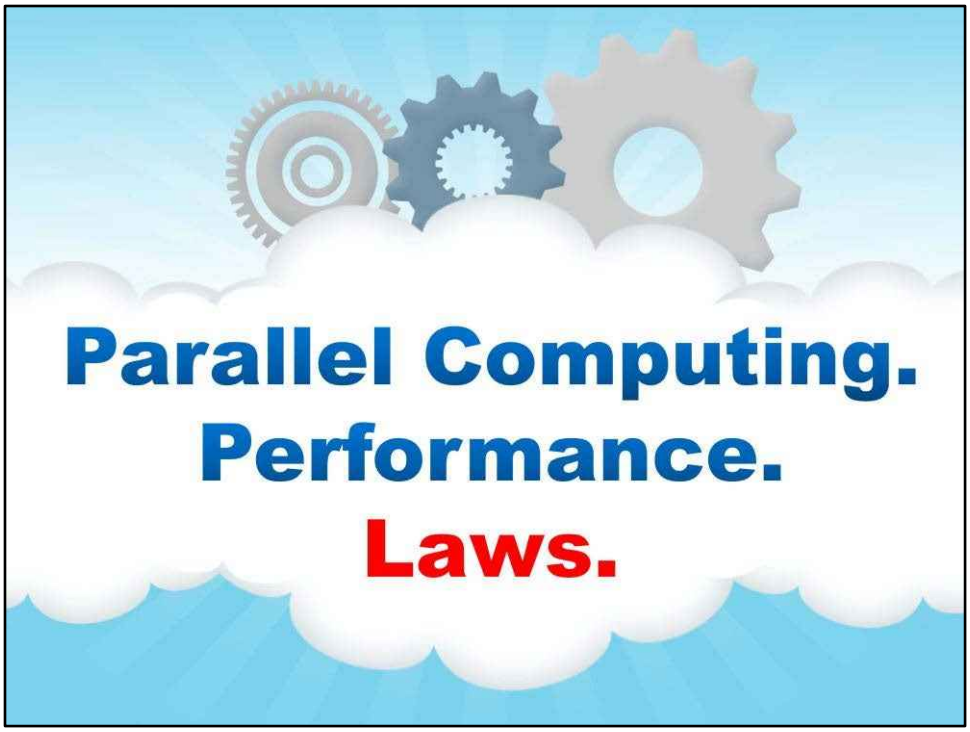
- "Cost" is a much overused word, the term "algorithm cost" is sometimes used for clarity.
- The cost of a parallel algorithm should be compared to the running time of a sequential algorithm.
  - Cost removes the advantage of parallelism by charging for each additional processor.
  - A parallel algorithm whose cost is growing "with similar rate" than the running time of an optimal sequential algorithm is called cost-optimal.

Speedup, Cost, Efficiency

$$\text{Efficiency} = \frac{\text{Sequential running time}}{\text{Processors} \times \text{Parallel running time}}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Processors}}$$

$$\text{Efficiency} = \frac{\text{Sequential running time}}{\text{Cost}}$$

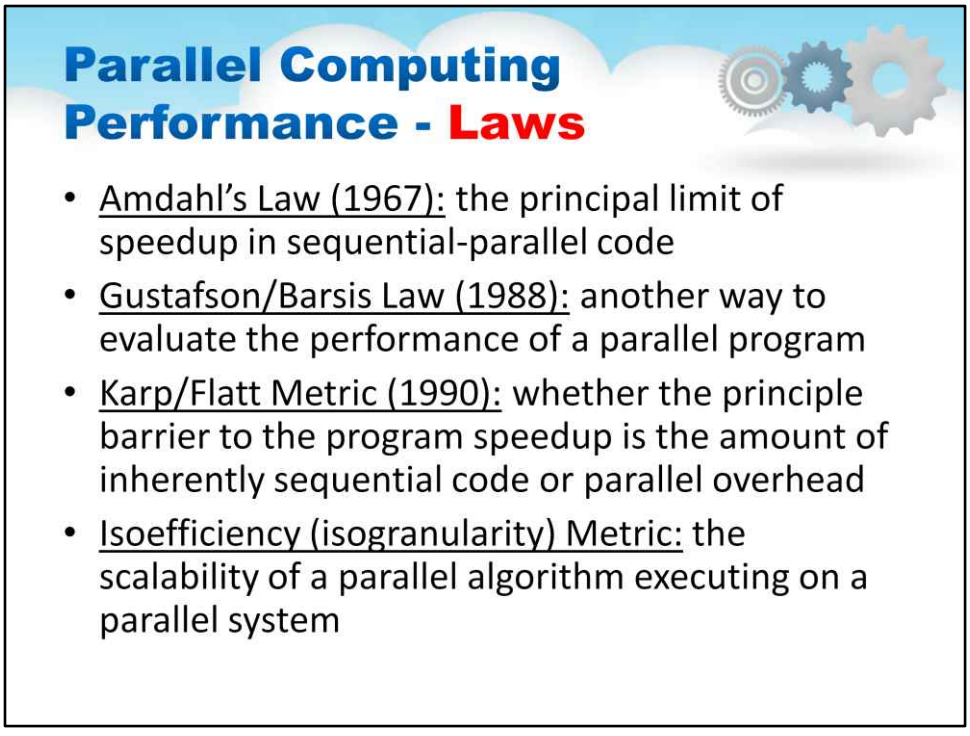


Закони

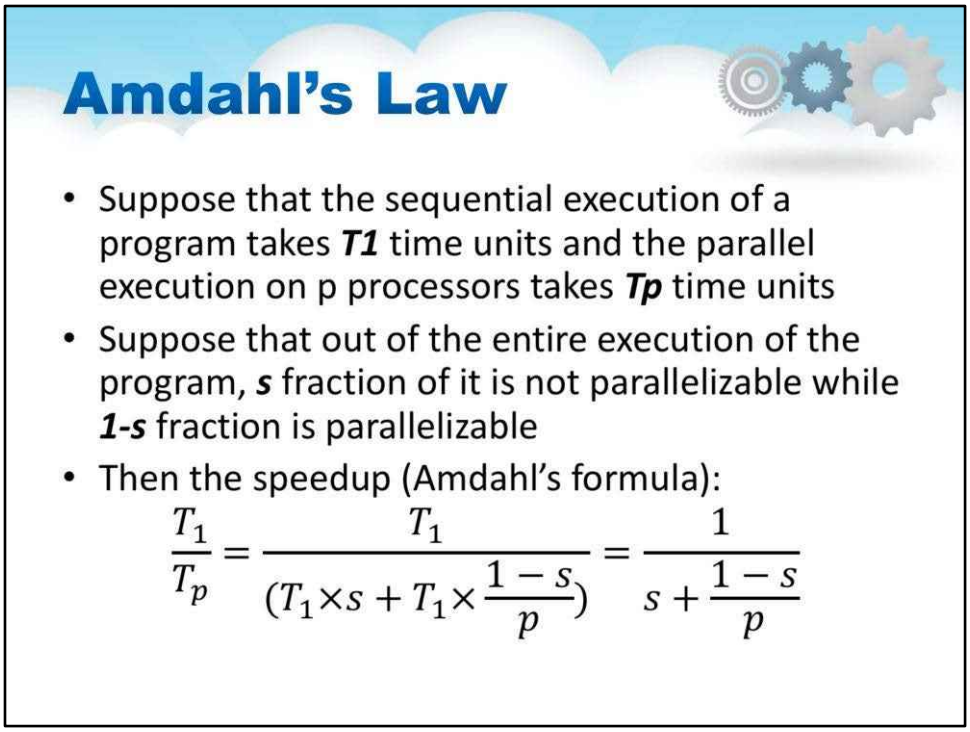
Не менш важливо знати витрати та переваги розпаралелювання.

Метою цього розділу є розгляд середніх (показників):

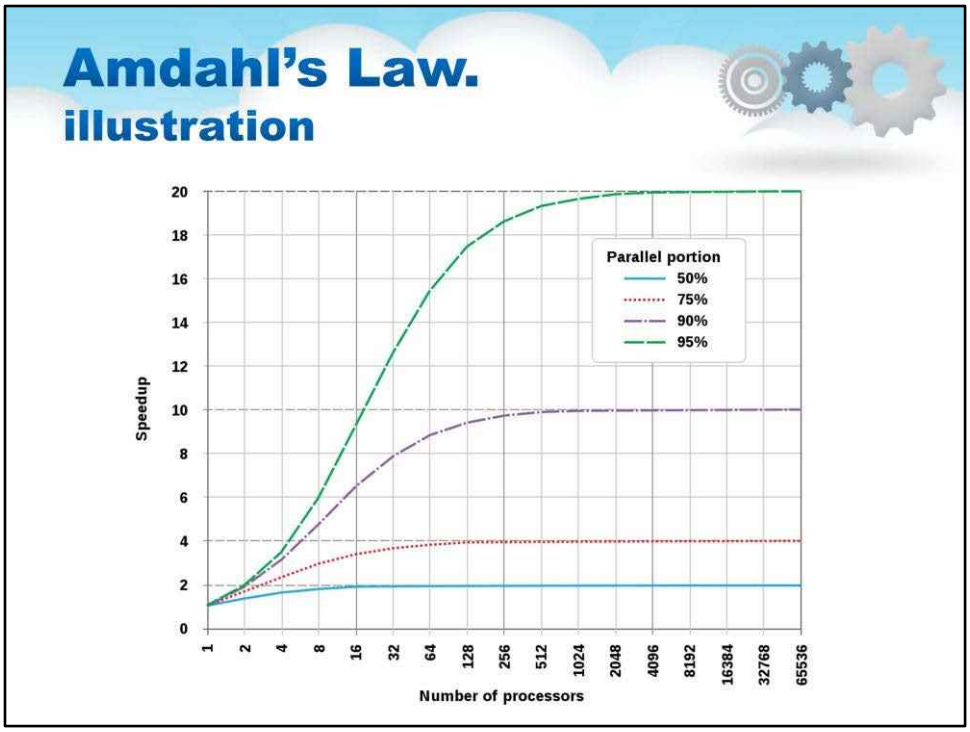
- визначити, наскільки швидше працює паралельний алгоритм, ніж послідовний;
- прогнозувати продуктивність паралельного алгоритму;
- щоб вирішити, чи існують внутрішні обмеження
  які перешкоджають роботі паралельного алгоритму, і що це таке;
- щоб визначити, наскільки ефективним може бути паралельний алгоритм
  оскільки розмір проблеми та кількість доступних процесорів зростає.



Сьогодні ми поговоримо проAmdahl's La w.



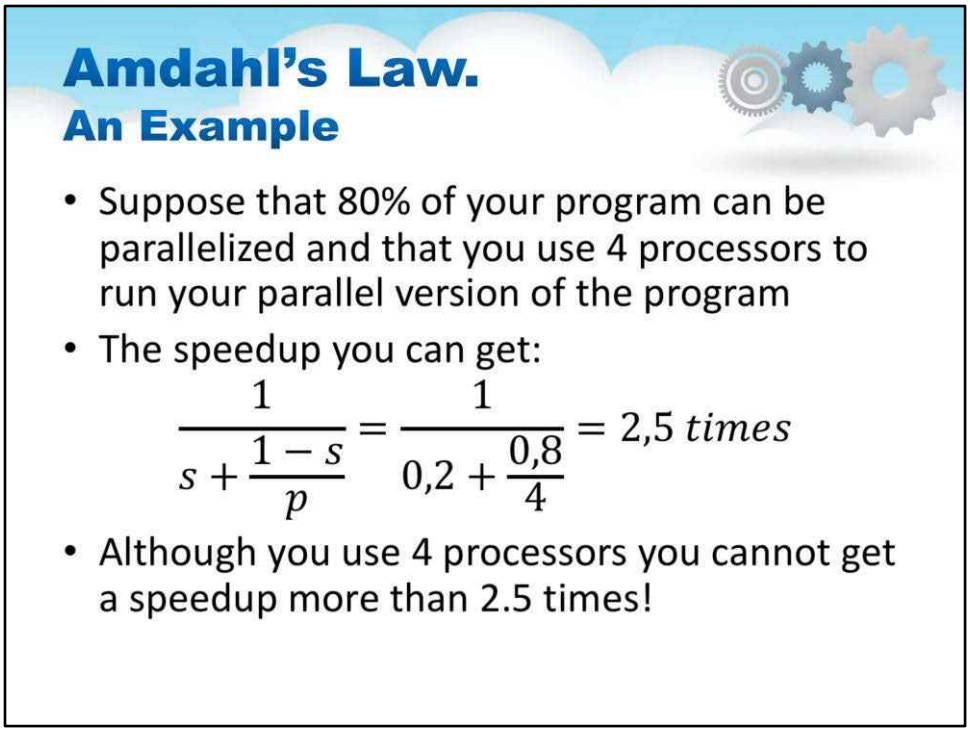
Припустимо, що послідовне виконання програми займаєТ1одиниць часу та паралельного виконання на р процесорахТродиниці часу



Закон Амдала часто використовується в паралельних обчисленнях для прогнозування теоретичного прискорення при використанні кількох процесорів.

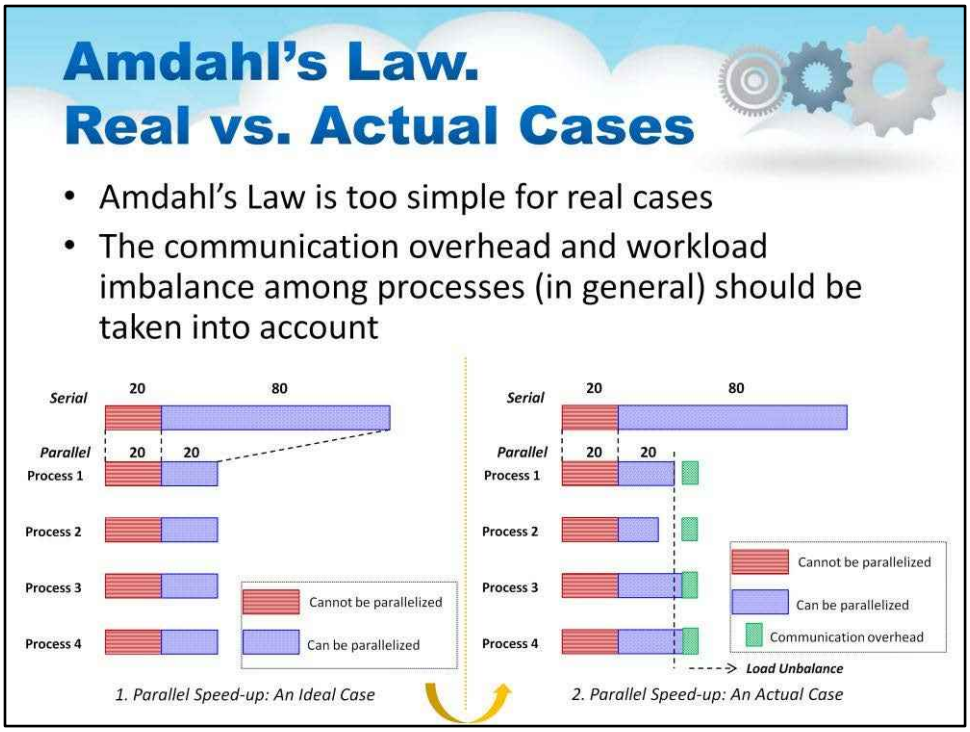
Наприклад, якщо програмі потрібно 20 годин, використовуючи одне ядро процесора, і певна частина програми, яка займає годину, не може бути розпаралелена, тоді як інші 19 годин ($p = 0{,}95$) часу виконання можуть бути розпаралелені, тоді Незалежно від того, скільки процесорів відведено для розпаралеленого виконання цієї програми, мінімальний час виконання не може бути меншим за одну критичну годину.

Отже, теоретичне прискорення обмежене не більше ніж у 20 разів ($1/(1 - p) = 20$). З цієї причини паралельні обчислення з багатьма процесорами корисні лише для програм, які добре розпаралелюються.

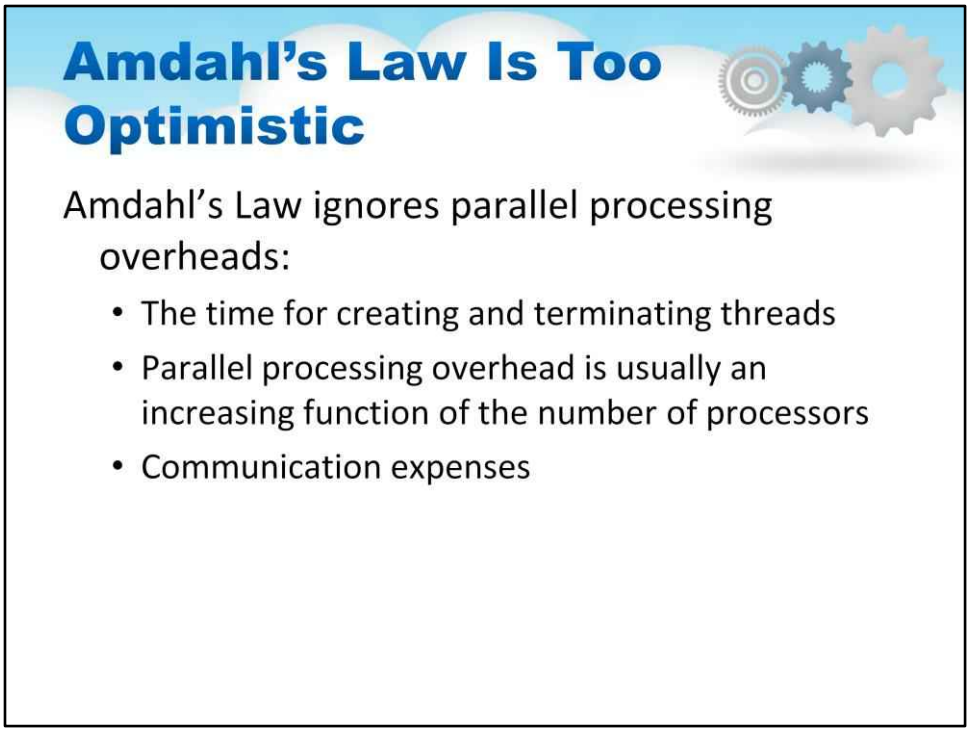


Припустімо, що 80% вашої програми можна розпаралелити, і ви використовуєте 4 процесори для виконання вашої паралельної версії програми.

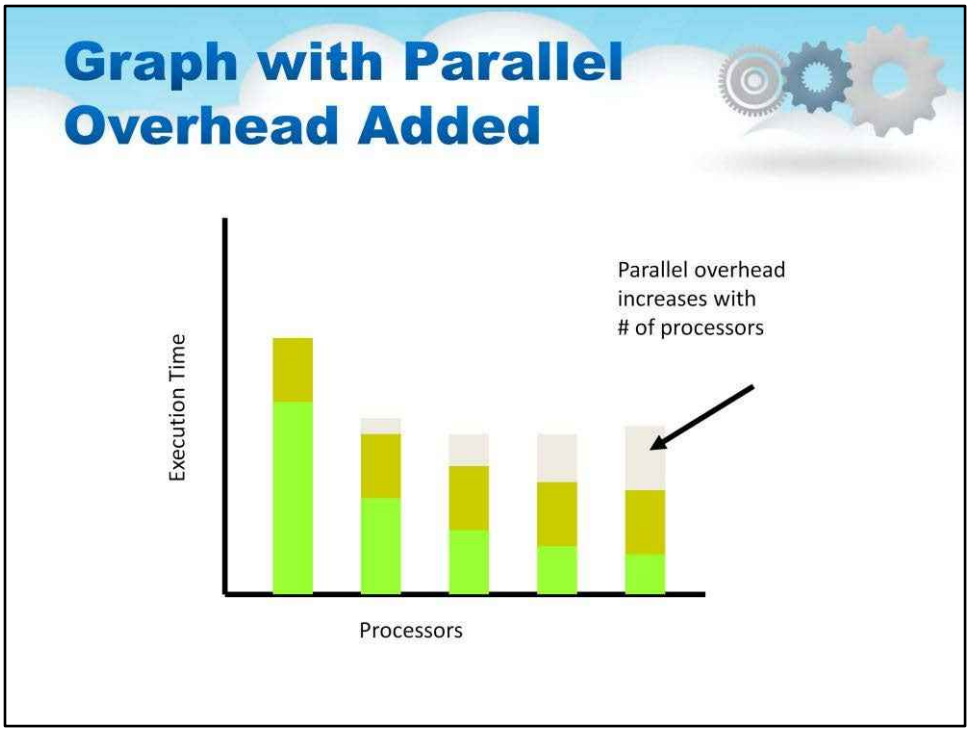
У результаті ви використовуєте 4 процесори і не можете отримати прискорення більш ніж у 2,5 рази!



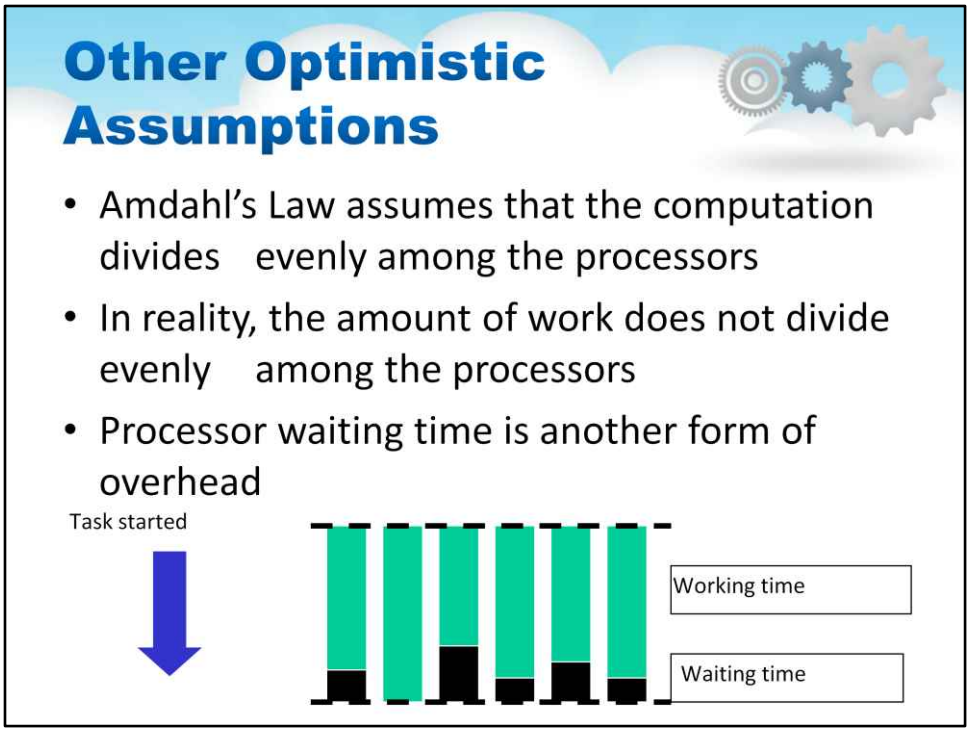
Закон Амдала занадто простий для реальних випадків. Накладні витрати на зв'язок і слід враховувати дисбаланс робочого навантаження між процесами (загалом).



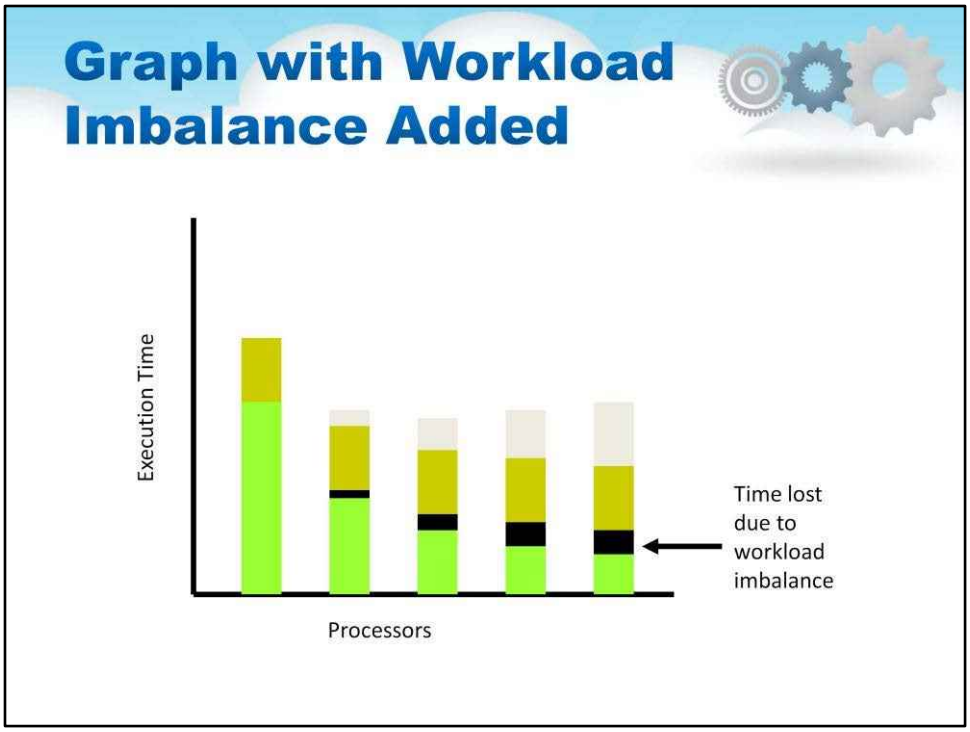
Рідко, коли паралельні обчислення дійсно досягають прискорення, передбаченого законом Амдала.



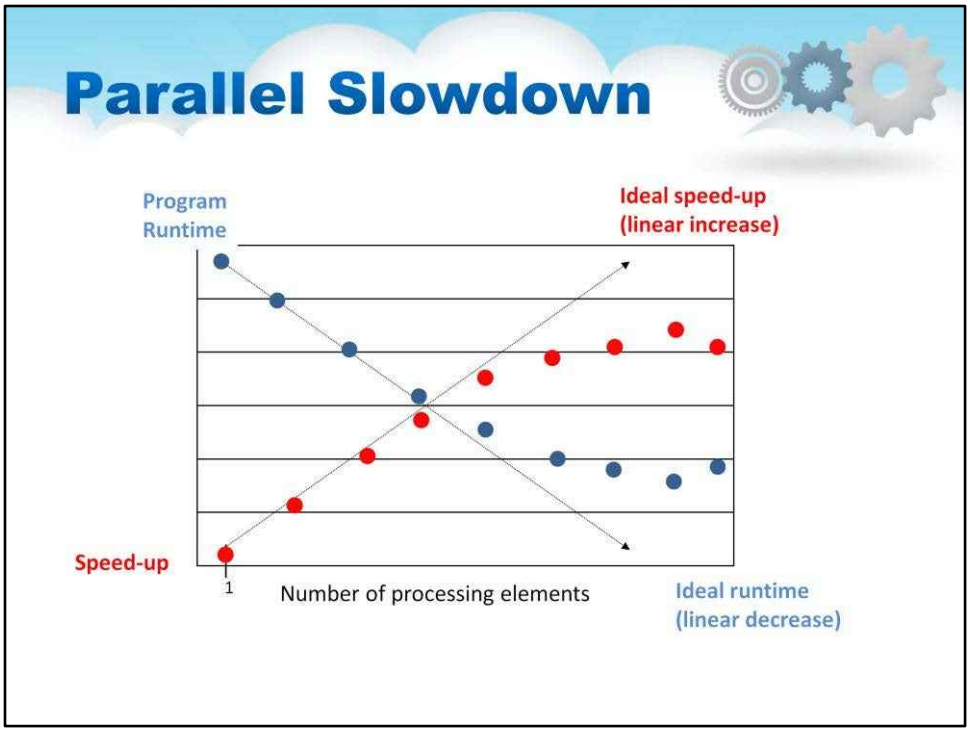
Врахування паралельних накладних витрат зменшує наші очікування щодо того, скільки процесорів можна вигідно використовувати для прискорення обчислень.



Завдання буде виконано лише тоді, коли завершить роботу останній процесор. Коли деякі процесори закінчити раніше за інших, час, який «ранні» процесори витрачають на очікування для останнього процесора, який закінчив, витрачається даремно. Це форма накладних витрат, оскільки один процесор ніколи не витрачає час на очікування, поки інший процесор закінчить роботу.



Коли ми додаємо час, втрачений через дисбаланс робочого навантаження, ми бачимо переваги розпаралелювання (у цьому гіпотетичному випадку) не виходить за межі трьох процесорів. Це набагато гірше, ніж передбачалося на початковому графіку.

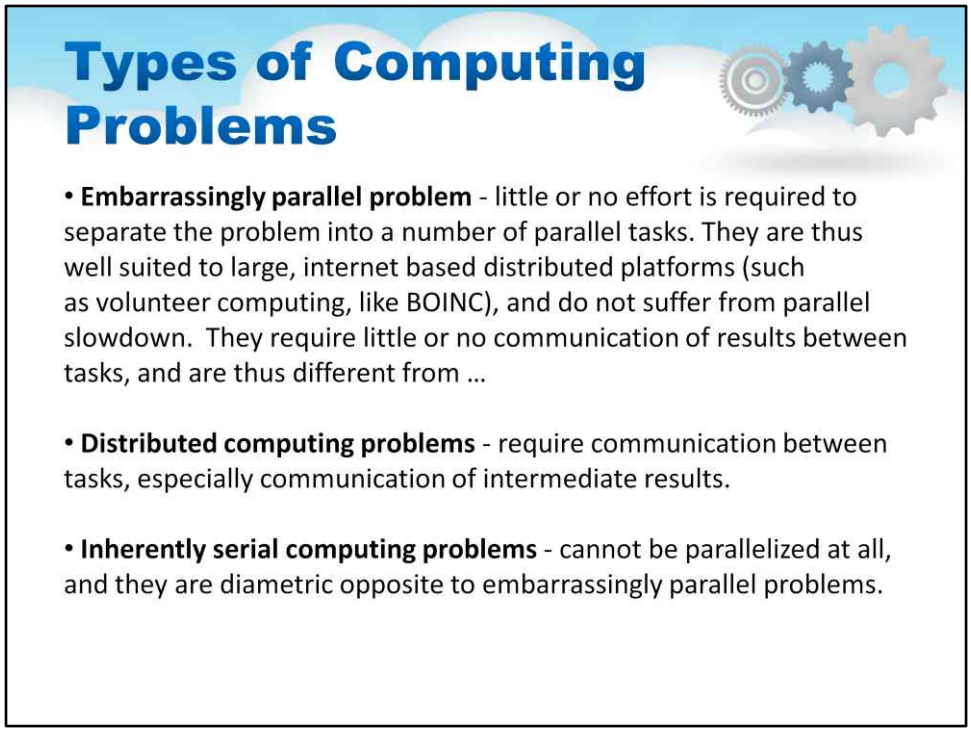


синій - показуєДіаграма часу виконання програми;

Червоний–шоупрограмне прискорення програми реального світу з субоптимальним розпаралелюванням.

Пунктирні лініївказують на оптимальне розпаралелювання – лінійне збільшення прискорення та лінійне зменшення часу виконання програми.

Ні: час виконання фактично збільшується з більшою кількістю процесорів (і прискорення так само зменшується) ->**це паралельне уповільнення.**



Надзвичайно паралельна проблема-щоб розділити проблему на кілька паралельних завдань, потрібно невеликі зусилля або зовсім не потрібні. Таким чином, вони добре підходять для великих розподілених платформ на основі Інтернету (таких як волонтерські обчислення, як-от BOINC), і не страждають від паралельного сповільнення. Вони вимагають незначного повідомлення результатів або взагалі не потребують їх між завданнями і, таким чином, відрізняються від...

Задачі розподілених обчислень-вимагають зв'язку між завданнями, особливо повідомлення проміжних результатів.

За своєю суттю послідовні обчислювальні проблеми-не можна паралелізувати взагалі, і вони є діаметрально протилежними до ганебно паралельних проблем

More General Speedup Formula

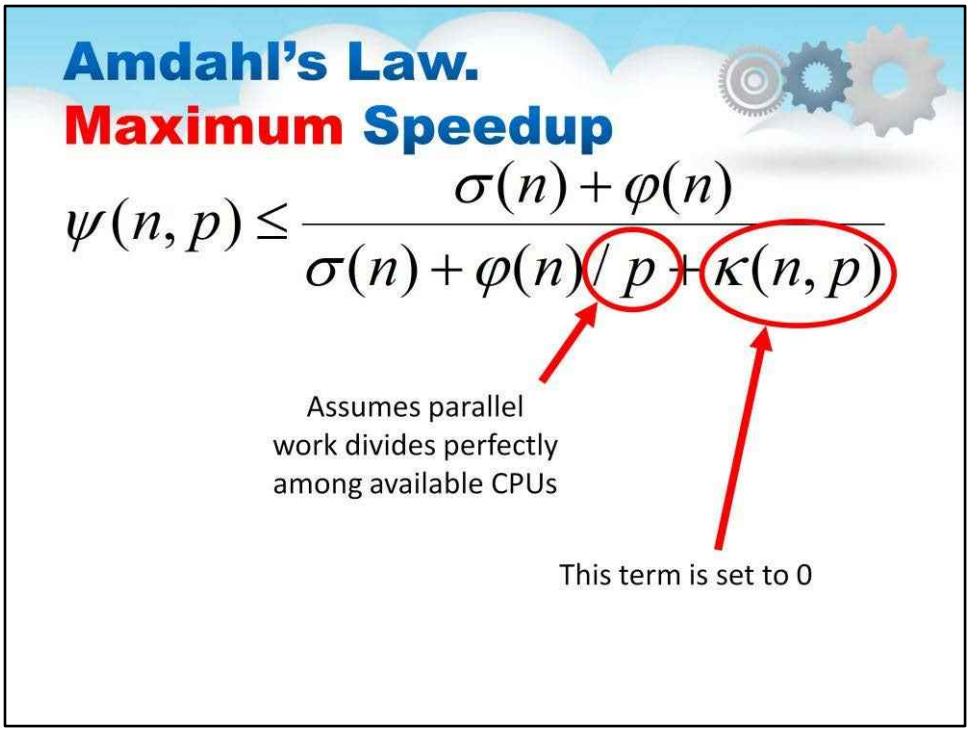
$\psi(n,p)$ - speedup for problem of size n on p CPUs

$\sigma(n)$ - time in sequential portion of code for problem of size n

$\varphi(n)$ - time in parallel portion of code for problem of size n

$\kappa(n,p)$ - parallel overheads

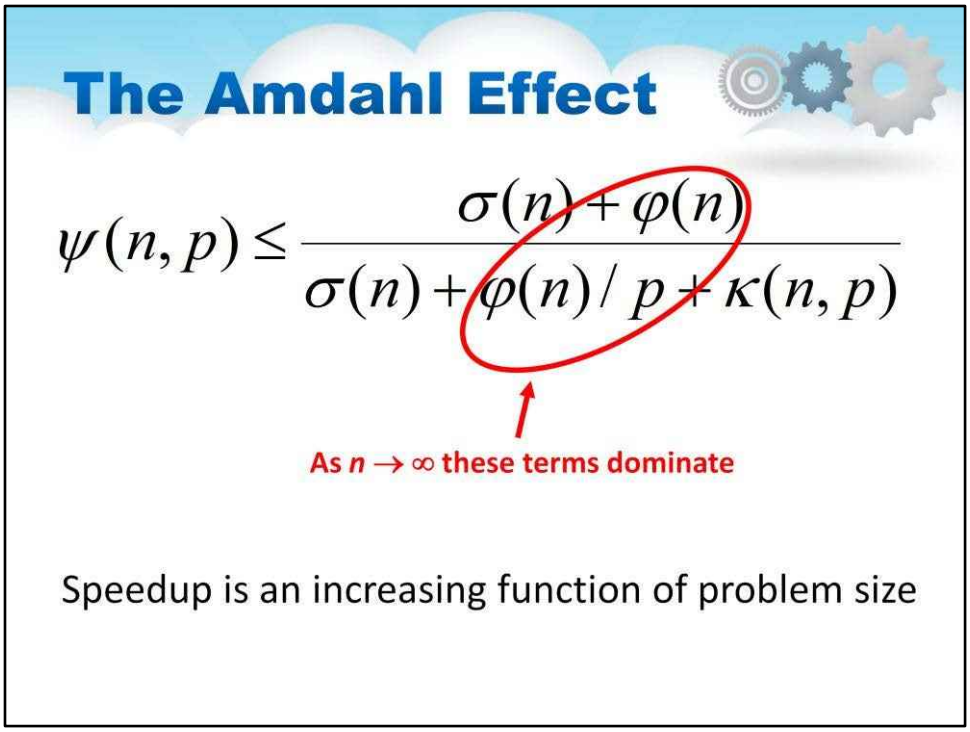
$$\psi(n,p) \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)}$$



Слайд висвітлює дві причини, чому ми повинні розглядати закон Амдала як забезпечення верхня межа прискорення, якого можна досягти, а не реалістичний прогноз.

Питання: Чому ми турбуємося про закон Амдала?

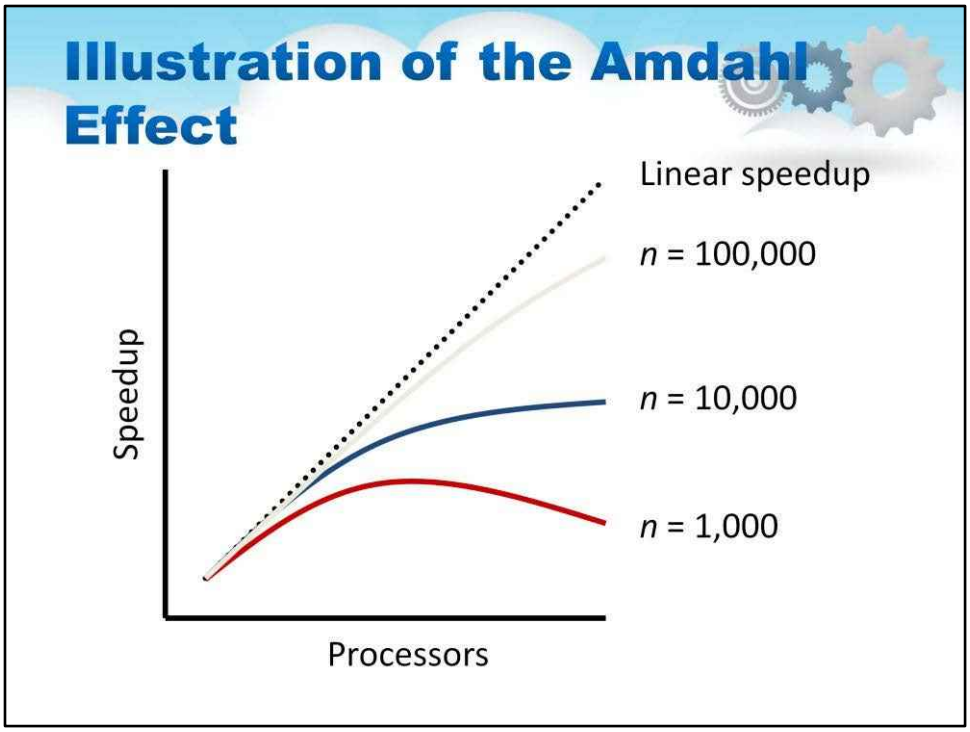
Відповідь: Тому що навіть занадто оптимістичний прогноз може бути корисним. Наприклад, Припустімо, ми перевіряємо програму, а потім використовуємо закон Амдала, щоб передбачити прискорення, якого ми досягнемо, якщо зробимо декілька ключових функцій паралельними. Якщо закон Амдала передбачає прискорення на 1,20 (тобто 20% покращення швидкості), ми можемо вирішити, що це не варте зусиль.



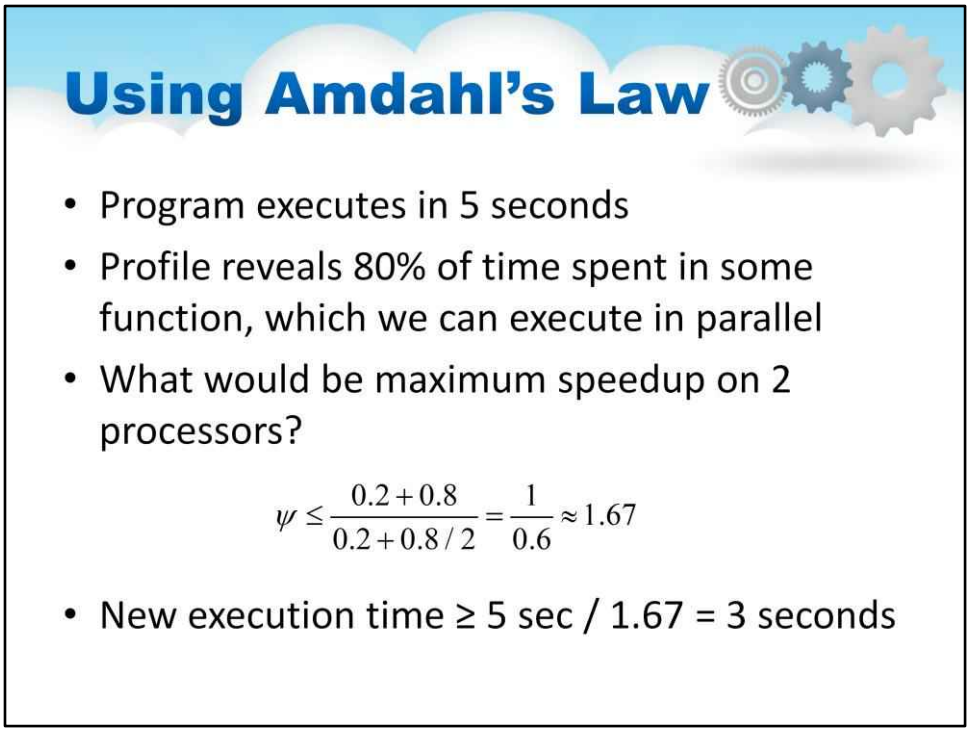
У більшості паралельних програм функції «sigma(n)» і «kappa(n,p)”мають нижчий клас складності, ніж функція “phi(n)”. Наприклад, при множенні матриці введення/виведення матриць має порядок «n» у квадраті, тоді як складність фактичного множення матриці дорівнює порядку «n» в кубі.

Отже, коли «n» стає більшим, домінують члени «phi(n)», а «psi(n,p)”наближається до «phi(n)/(phi(n)/p)» або «p».

Іншими словами, прискорення є зростаючою функцією розміру проблеми. Наприклад, коли розмір проблеми більший, може бути більше паралельних операцій на перешкоду синхронізації. Це робить накладні витрати на бар'єр відносно меншими.



Для даної кількості процесорів, коли ми збільшуємо розмір проблеми, відносна кількість часу, витраченого на виконання корисної роботи, збільшується, підвищуючи ефективність і прискорення паралельної програми.



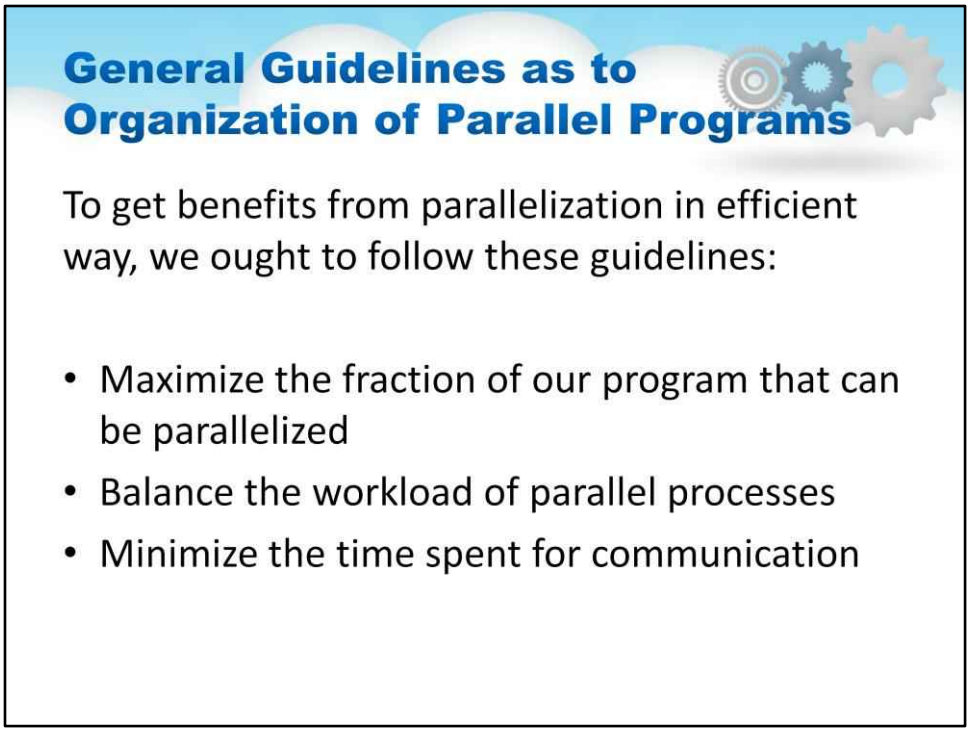
Запитання: якщо 25% часу програми витрачається на послідовний код, яка найбільшого прискорення, якого можна досягти, незалежно від кількості процесорів?

Відповідь: 4. Це тому, що обмеження, коли «p» прямує до нескінченності, становить 1 / (0,25 + (1 - .25)/p) = 1 / 0,25 = 4.

Gene Amdahl
(1922-2015)

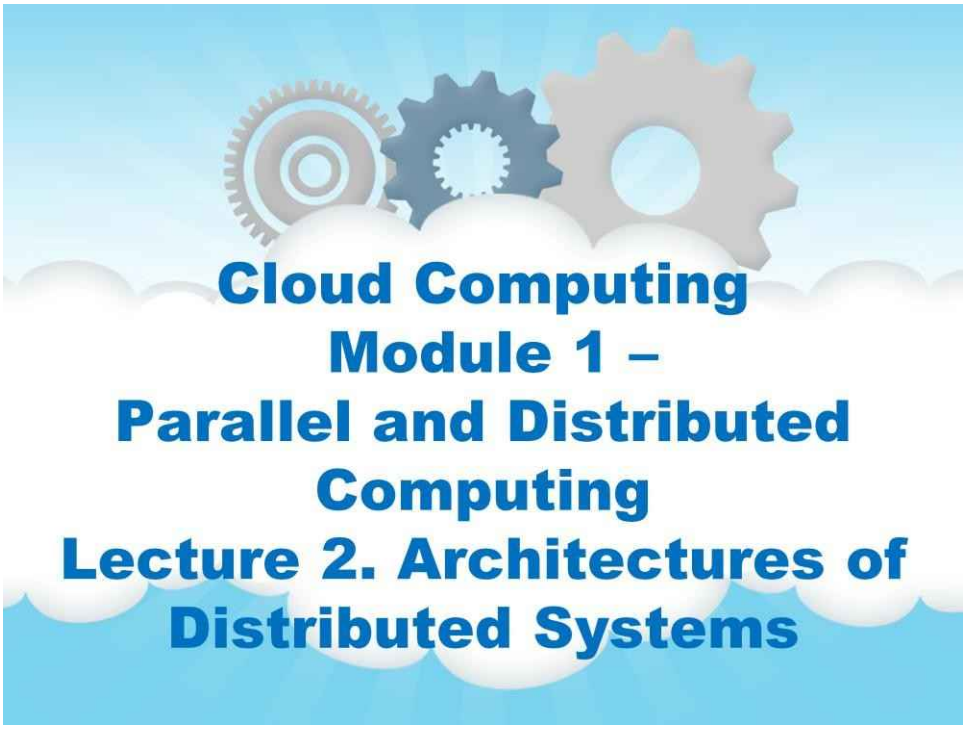
- He left IBM again in September 1970, after his ideas for computer development were rejected;
- **set up Amdahl Corporation** in Sunnyvale, California with aid from Fujitsu.
- Competing with IBM the company manufactured **"plug-compatible" mainframes**.
- **Amdahl's law**.

На початку 1960-х років Амдал був головним архітектором IBM System/360, яка поклала початок найприбутковішій лінії продуктів корпорації для мейнфреймів. Це був перший термінархітектура було застосовано до комп'ютерного дизайну.
System/360 не використовував паралельну обробку, але конкуруючий комп'ютер ILIAC IV мав дизайн SIMD із 64 процесорами.
У 1967 році Джин Амдал стверджував на весняній об'єднаній комп'ютерній конференції AFIPS, що оскільки операційна система ILIAC IV займала від 25 до 45 відсотків машинних циклів, паралельні програми могли досягти прискорення лише від 2 до 4.

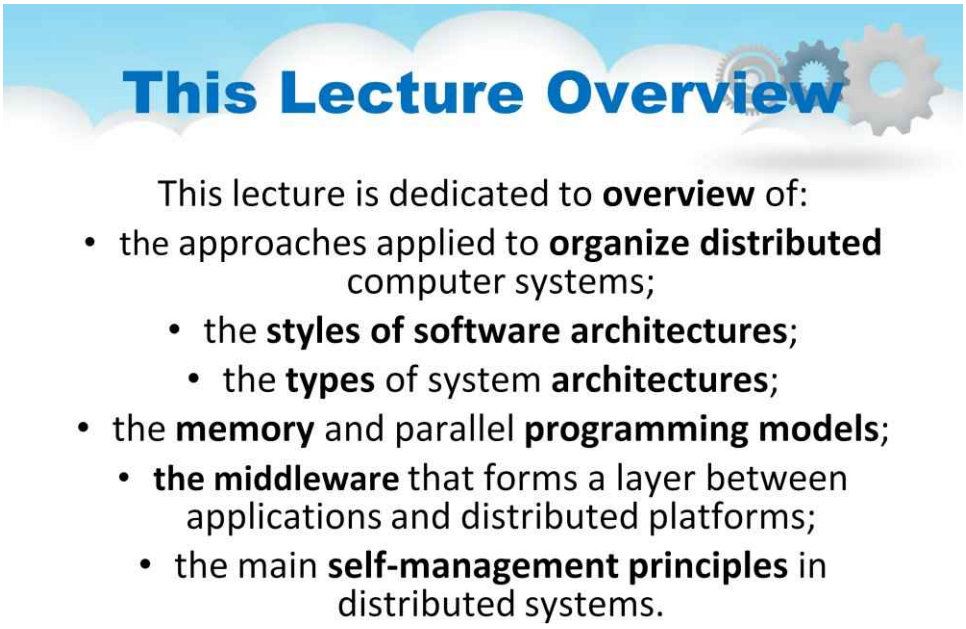


Щоб отримати переваги від розпаралелювання ефективним способом, ми повинні дотримуватися таких вказівок:

Збільште частку нашої програми, яку можна розпаралелити. Збалансуйте робоче навантаження паралельних процесів
Мінімізуйте час, витрачений на спілкування



Назва цього модуля – «Паралельні та розподілені обчислення». Це лекція 2 «Архітектури розподілених систем».

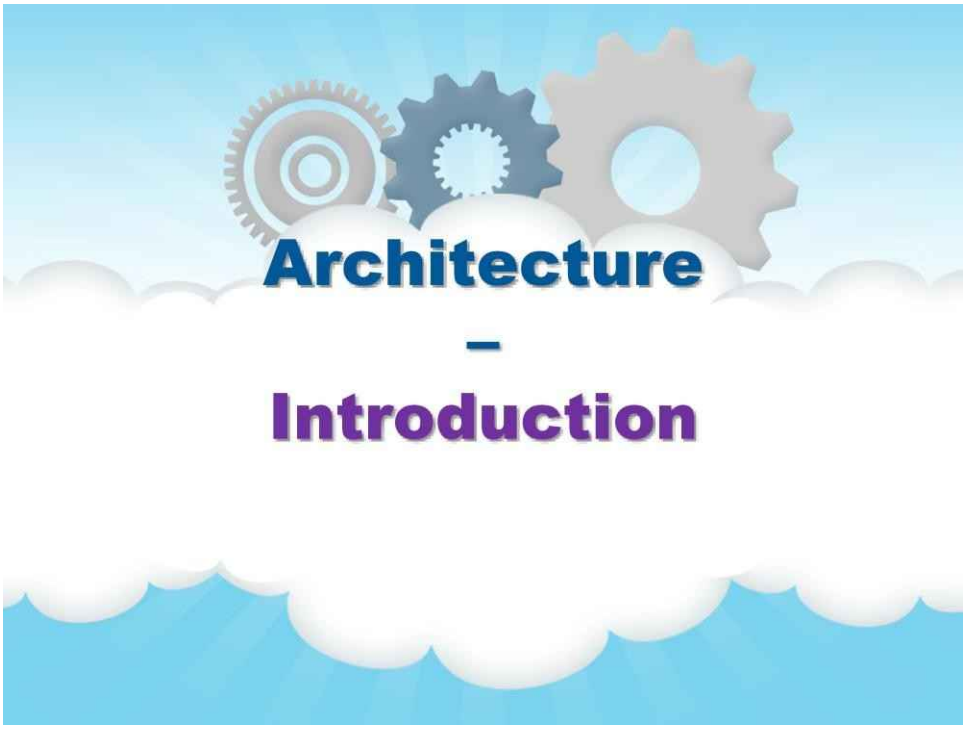


Лекція 2. Архітектури с

Розподілені системи

Ця лекція про:

- підходи, які використовуються для організації розподіленого комп'ютера системи; стилі програмних архітектур;
- типи системних архітектур;
- моделі пам'яті та паралельного програмування;
- Проміжне програмне забезпечення, а саме програмне забезпечення між програмами та розподіленими платформами;
- і важливі принципи самоуправління в розподілених системах.

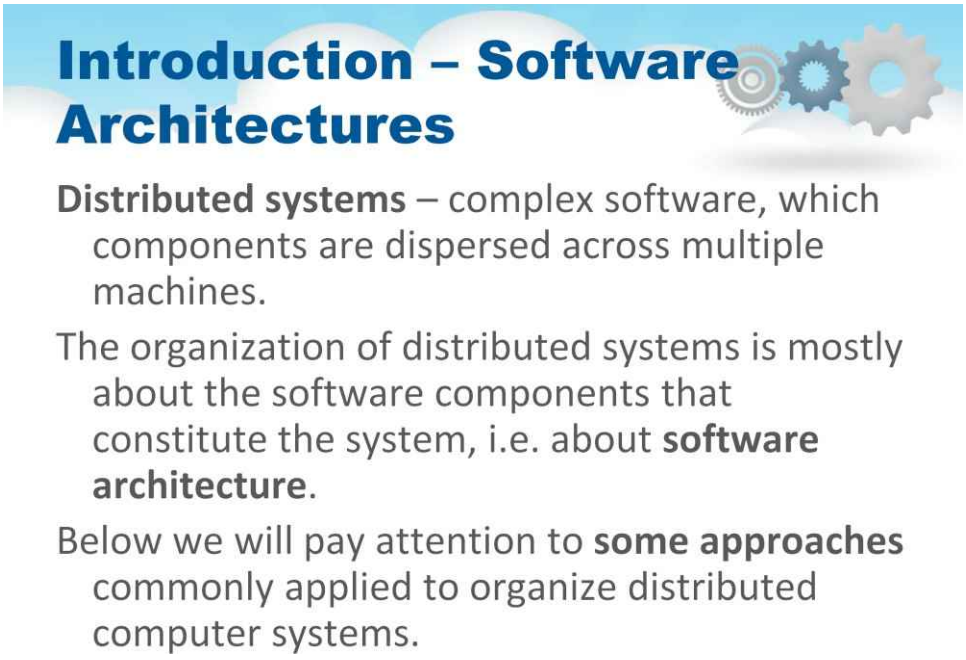


Архітектура

вступ

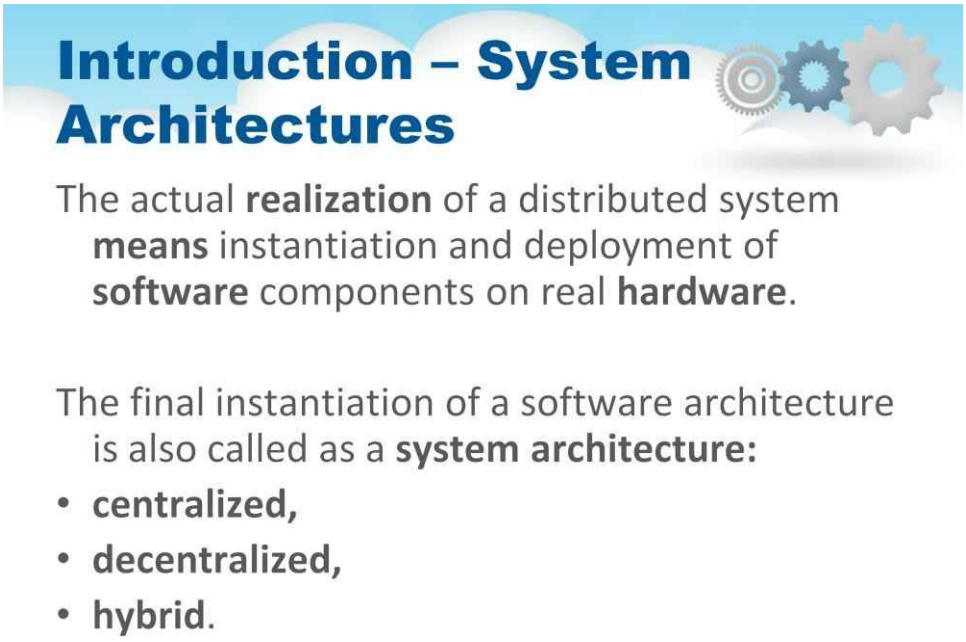
Ми починаємо наше обговорення архітектур з розгляду спочатку логічної організації розподілених систем у програмні компоненти, які також називають архітектурою програмного забезпечення.

Дослідження архітектур програмного забезпечення значно зріли, і тепер загальновизнано, що розробка або впровадження архітектури має вирішальне значення для успішної розробки великих систем.



Розподілені системи часто являють собою складні частини програмного забезпечення, компоненти якого за визначенням розосереджені на кількох машинах. Щоб подолати їх складність, надзвичайно важливо, щоб ці системи були належним чином організовані. Існують різні способи перегляду організації розподіленої системи, але очевидним є розрізнення між логічною організацією набору програмних компонентів і, з іншого боку, фактичною фізичною реалізацією.

Організація розподілених систем здебільшого стосується програмних компонентів, які складають систему. Ці архітектури програмного забезпечення говорять нам, як різні компоненти програмного забезпечення повинні бути організовані та як вони повинні взаємодіяти. У цій лекції ми звернемо увагу на деякі загальноприйняті підходи до організації (розподілених) комп'ютерних систем.



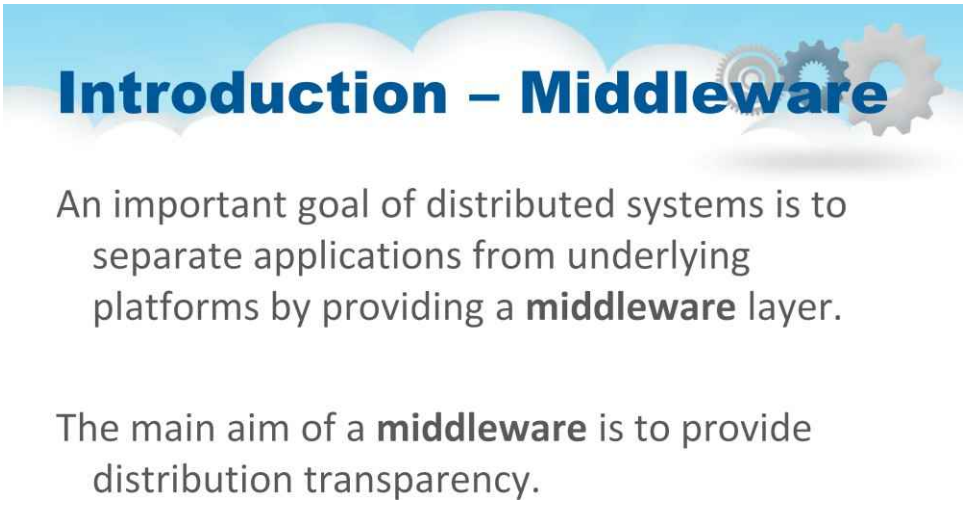
Фактична реалізація розподіленої системи вимагає, щоб ми створили екземпляр і розмістили компоненти програмного забезпечення на реальному обладнанні.

При цьому можна зробити багато різних варіантів. Остаточний екземпляр програмної архітектури також називають системною архітектурою.

У цій лекції ми розглянемо традиційні**централізований**архітектури, в яких один сервер реалізує більшість програмних компонентів (і, отже, функціональність), тоді як віддалені клієнти можуть отримати доступ до цього сервера за допомогою простих засобів зв’язку.

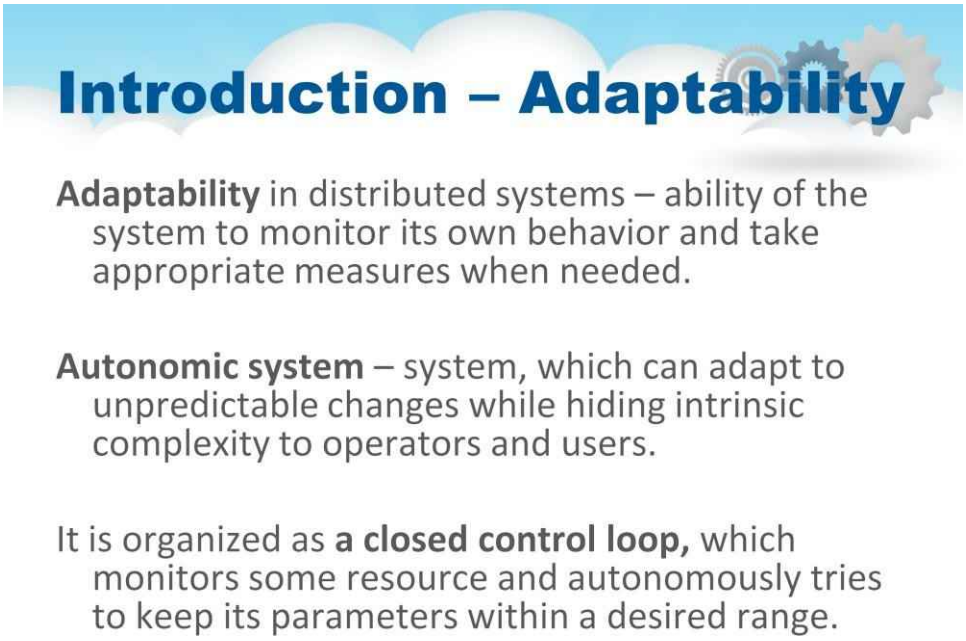
Крім того, ми розглядаємо**децентралізована**архітектури, в якій машини більш-менш відіграють однакові ролі.

Також розглядаємо як**гібрид**організацій.



Як ми пояснювали в попередній лекції, важливою метою розподілених систем є відокремлення додатків від базових платформ шляхом надання рівня проміжного програмного забезпечення.

Прийняття такого шару є важливим архітектурним рішенням, і його основна мета — забезпечити прозорість розподілу. Однак для досягнення прозорості необхідно досягти компромісів, що призвело до різних методів адаптації проміжного програмного забезпечення. У цій лекції ми обговорюємо деякі з них, які найчастіше застосовуються, оскільки вони впливають на організацію самого проміжного ПЗ.



Адаптивністьу розподілених системах також можна досягти, якщо система контролює свою власну поведінку та вживає відповідних заходів, коли це необхідно.

Це розуміння призвело до класу, який зараз називають**вегетативні системи**, які є системами, які адаптуються до непередбачуваних змін, приховуючи при цьому внутрішню складність для операторів і користувачів.

Ці розподілені системи часто організовані у формі зворотного зв'язку **контури керування**, які створюють важливий архітектурний елемент під час проектування системи.

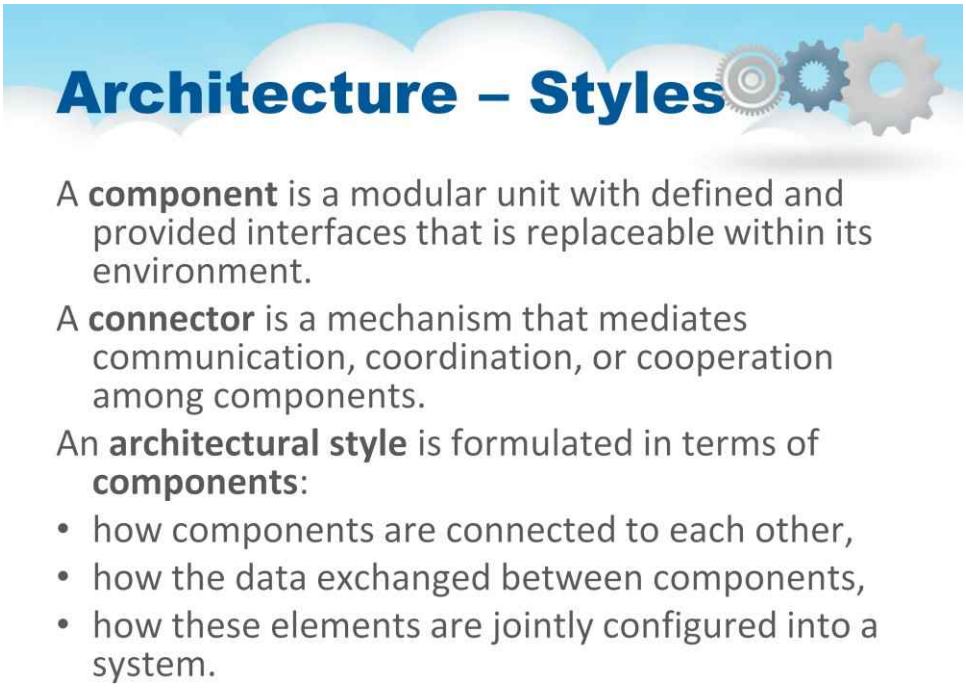
У цій лекції ми присвятимо розділ**автономні розподілені системи**.

Architecture – Styles

Стилі

Давайте почнемо наше обговорення архітектур, спочатку розглянувши логічну організацію розподілених систем у програмні компоненти, які також називають архітектурою програмного забезпечення.

Дослідження архітектур програмного забезпечення значно зріли, і тепер загальновизнано, що розробка або впровадження архітектури має вирішальне значення для успішної розробки великих систем.



Поняття архітектурного стилю є важливим і базується на компонентах і сполучниках.

Компонент — це модульна одиниця з чітко визначеними необхідними та наданими інтерфейсами, яку можна замінити в своєму середовищі. Як ми обговоримо нижче, важливою проблемою щодо компонента для розподілених систем є те, що його можна замінити, якщо ми поважаємо його інтерфейси.

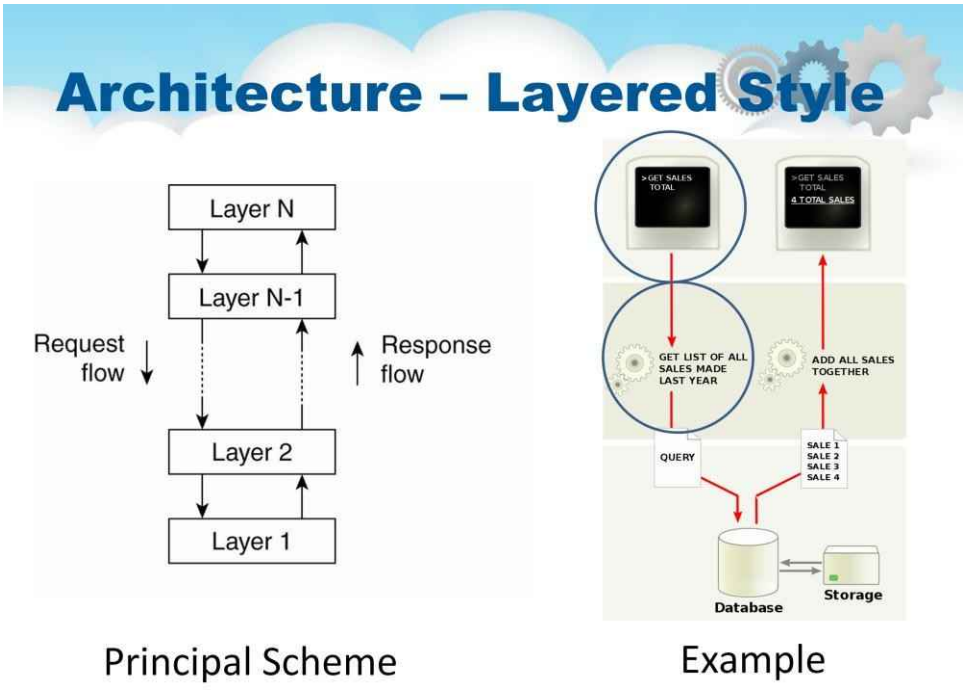
Дещо важче зрозуміти концепцію конектора, який зазвичай описується як механізм, що забезпечує зв’язок, координацію або співпрацю між компонентами. Наприклад, з’єднувач може бути утворений засобами для (віддаленого) виклику процедур, передачі повідомлень або потокових даних.

Архітектурний стиль формулюється в термінах компонентів, способу з’єднання компонентів один з одним, даних, якими обмінюються компоненти. і, нарешті, як ці елементи спільно налаштовані в систему.

Використовуючи компоненти та роз’єми, ми можемо прийти до різноманітних конфігурацій, які, у свою чергу, були класифіковані за архітектурними стилями.

На даний момент визначено декілька стилів, з яких найважливішими для розподілених систем є:

1. Багаторівневі архітектури
2. Об'єктно-орієнтовані архітектури
3. Архітектури, орієнтовані на дані
4. Архітектури на основі подій



[ЛІВА СТОРОНА]

Основна ідея для**багатошаровий стиль**простий:

компоненти організовані пошарово, де компонент на рівні i дозволено робити запити до компонентів на базовому рівні i-1, але не навпаки, як показано на цьому малюнку.

Ця модель була широко прийнята IT-спільнотою. Ключове зауваження полягає в тому, що управління зазвичай переходить від рівня до рівня: запити йдуть вниз по ієрархії, але результати, навпаки, йдуть вгору.

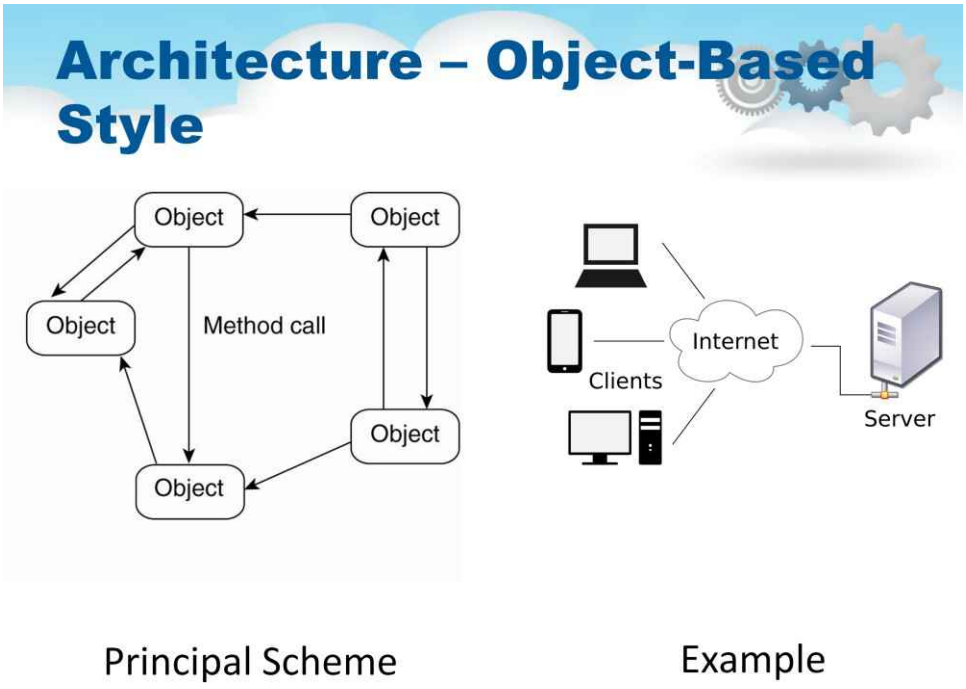
[ПРАВА СТОРОНА]

З правого боку ви можете побачити типове виконання цього стилю архітектури. The **найвищий**рівень (**Рівень презентації**у цьому прикладі) створює інтерфейс користувача. Його головна мета — перекладати запити користувача на машину і, навпаки, перекладати відповіді машини на користувача.

The**середина**рівень (**Логічний рівень**тут) координує застосування, команди, робить оцінки, виконує розрахунки та приймає рішення.

І він доставляє запити-відповіді та дані між іншими рівнями.

Нижній рівень (**Рівень даних**тут) містять інформацію в базі даних або файловій системі. Після запитів інформація з цього рівня даних передається на логічний рівень для обробки та на рівень презентації для користувачів.



[ЛІВА СТОРОНА]

Більш гнучку організацію пропонується в**об'єктний**архітектури, які зображені на цьому малюнку.

По суті, кожен об’єкт відповідає тому, що ми визначили як компонент,

і ці компоненти з’єднані через (віддалений) механізм виклику процедури.

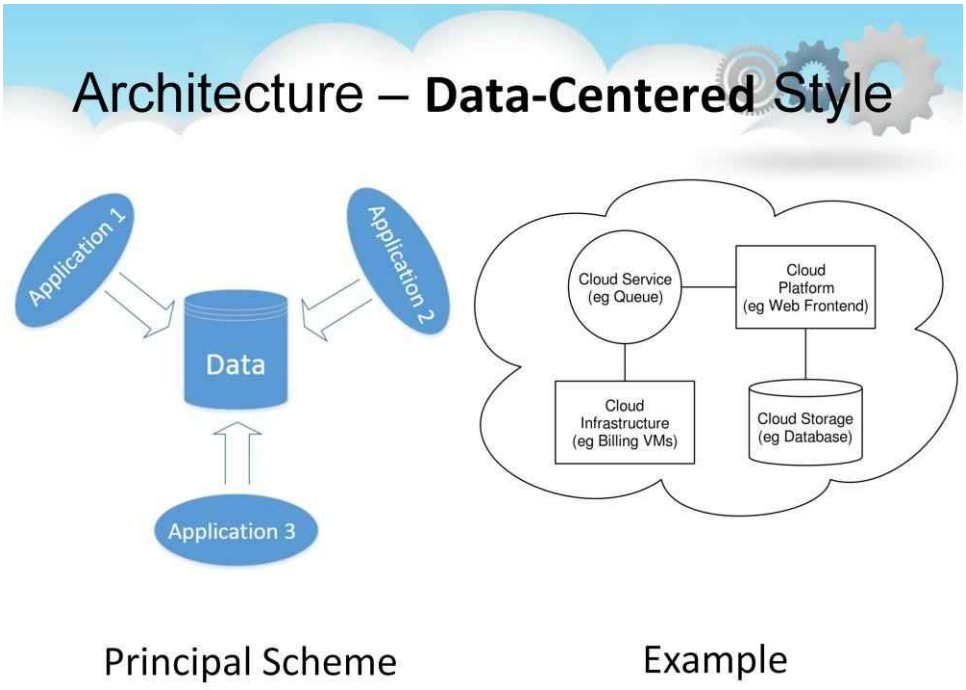
[ПРАВА СТОРОНА]

Цей стиль архітектури відповідає**клієнт-сервер**архітектура системи (детальніше описано нижче).

Багаторівнева та об’єктно-орієнтована архітектури все ще формують найважливіші стилі для великих програмних систем.

У цьому прикладі схеми комп’ютерної мережі клієнти спілкуються із сервером через Інтернет.

Сервер запускає одну або кілька серверних програм, які спільно використовують свої ресурси з клієнтами.



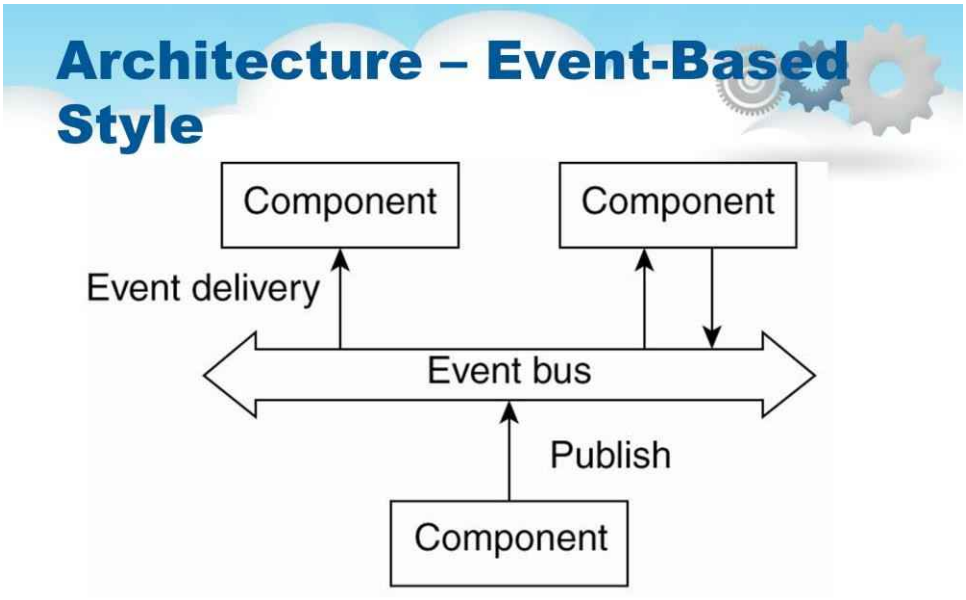
[ЛІВА СТОРОНА]

Перегляди архітектур, орієнтованих на дані**даних**як**найцінніша частина**програми, де процеси спілкуються через загальне (пасивне або активне) сховище даних.

[ПРАВА СТОРОНА]

У контексті розподілених систем ці архітектури є такими ж важливими, як і багаторівневі та об’єктно-орієнтовані архітектури.

Наприклад, багато мережевих додатків покладаються на спільну розподілену файлову систему, у якій фактично весь зв’язок відбувається через файли. Подібним чином системи хмарних обчислень, які ми докладно обговоримо пізніше, в основному орієнтовані на дані: процеси спілкуються за допомогою спільних служб даних.



Principal Scheme

В архітектурі, заснованій на подіях, процеси, по суті, спілкуються через поширення подій, які, за бажанням, також передають дані, як показано на цьому малюнку.

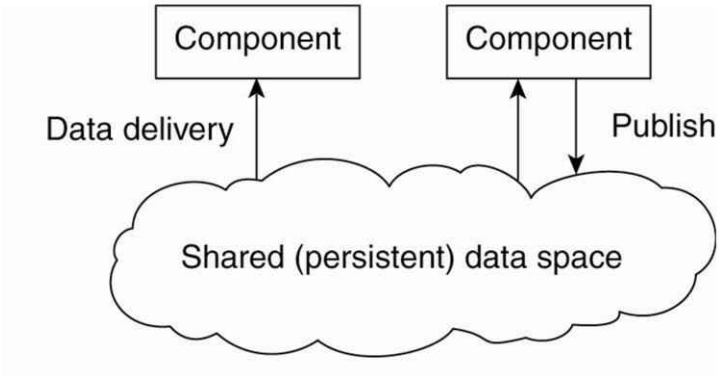
Для розподілених систем розповсюдження подій зазвичай асоціюється з так званими системами публікації/підписки.

Основна ідея полягає в тому, що процеси публікують події, після чого проміжне програмне забезпечення гарантує, що лише ті процеси, які підписалися на ці події, отримають їх.

Головна перевага систем, заснованих на подіях, полягає в тому, що процеси слабко пов’язані.

В принципі, вони не повинні явно посилатися один на одного.

Це також називають буттям**роз'єднані в просторі**, або**посилально відокремлений**.



Principal Scheme

Архітектури, засновані на подіях, можна поєднувати з архітектурами, орієнтованими на дані, які також називаються спільними просторами даних.

Суть спільних просторів даних полягає в тому, що процеси тепер також роз’єднані в часі: їм не обов’язково бути активними під час спілкування. Крім того, багато спільних просторів даних використовують SQL-подібний інтерфейс для спільного сховища в тому сенсі, що доступ до даних можна отримати за допомогою опису, а не явного посилання,
як у випадку з файлами.

Що робить ці програмні архітектури важливими для розподілених систем, так це те, що всі вони спрямовані на досягнення (на розумному рівні) прозорості розповсюдження. Однак прозорість розподілу вимагає компромісів між продуктивністю, відмовостійкістю, простотою програмування тощо.
Оскільки немає єдиного рішення, яке б відповідало вимогам для всіх можливих розподілених програм,
дослідники відмовилися від ідеї, що єдина розподілена система може бути використана для покриття 90% усіх можливих випадків.

System Architectures

Системні архітектури

Тепер, коли ми коротко обговорили деякі загальні архітектурні стилі, давайте подивимося, скільки розподілених систем насправді організовано, враховуючи, де розміщені програмні компоненти.

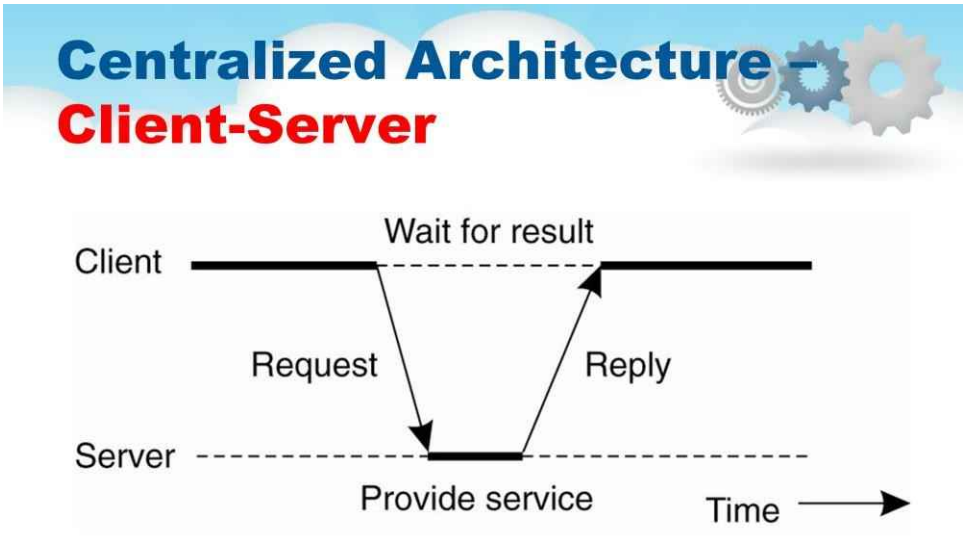
Програмні компоненти, їх взаємодія та їх розміщення призводять до створення екземпляра програмної архітектури, яку також називають системною архітектурою.

Ми обговоримо централізовані та децентралізовані організації та різні гібридні форми.

System Architectures – Centralized Architectures

Централізовані архітектури

Незважаючи на відсутність консенсусу щодо багатьох питань розподілених систем, є одна проблема, з якою погоджуються багато дослідників і практиків: мислення в термінах клієнтів, які запитують послуги від серверів, допомагає нам зрозуміти та керувати складністю розподілених систем, і це добре.



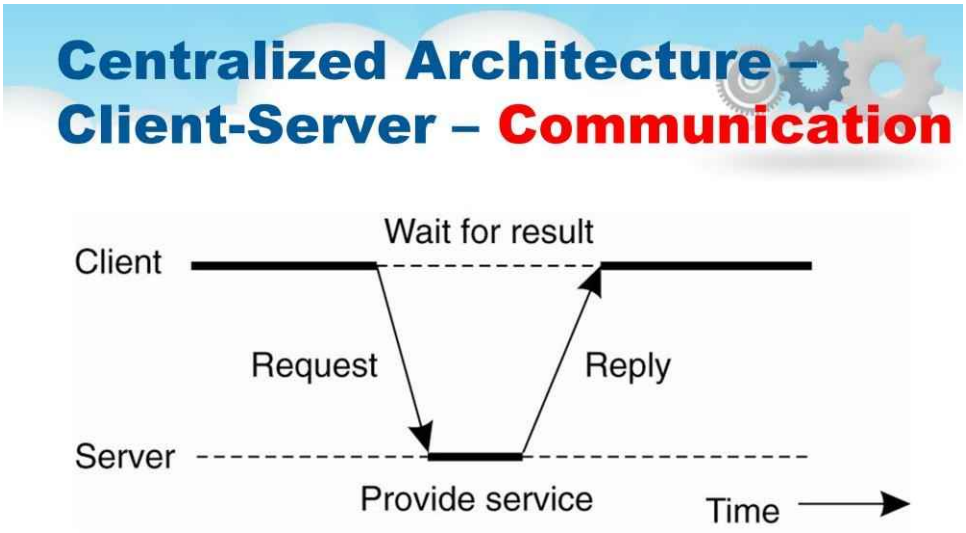
General **client-server** interaction (or **request-reply** behavior) between a client and a server

У базовій моделі клієнт-сервер процеси в розподіленій системі поділяються на дві групи (можливо, що перекриваються).

А**сервер**це процес, що реалізує певну службу, наприклад, службу файлової системи або службу бази даних.

А**клієнт**це процес, який запитує послугу від сервера, надсилаючи йому запит і згодом очікуючи відповіді сервера.

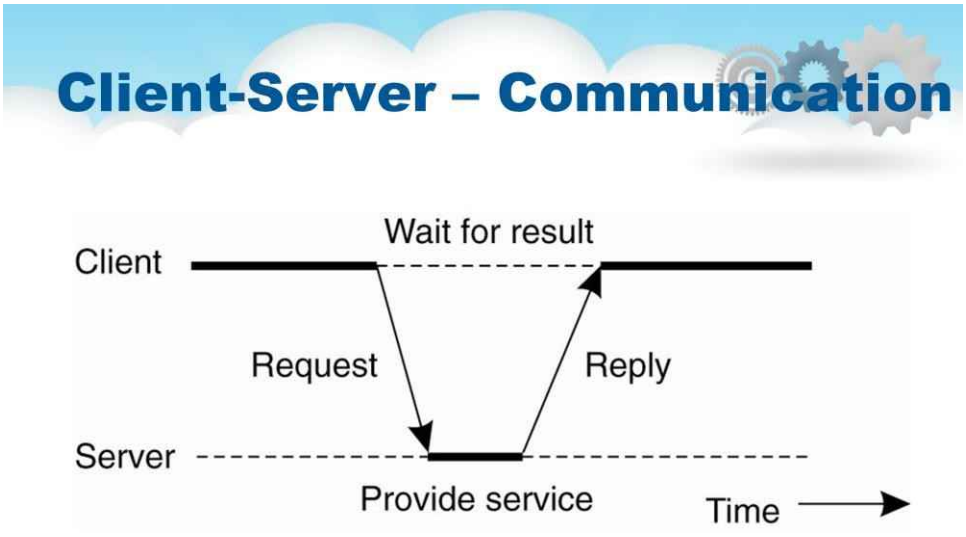
Ця взаємодія клієнт-сервер, також відома як**поведінка запит-відповідь**показано на цьому малюнку.



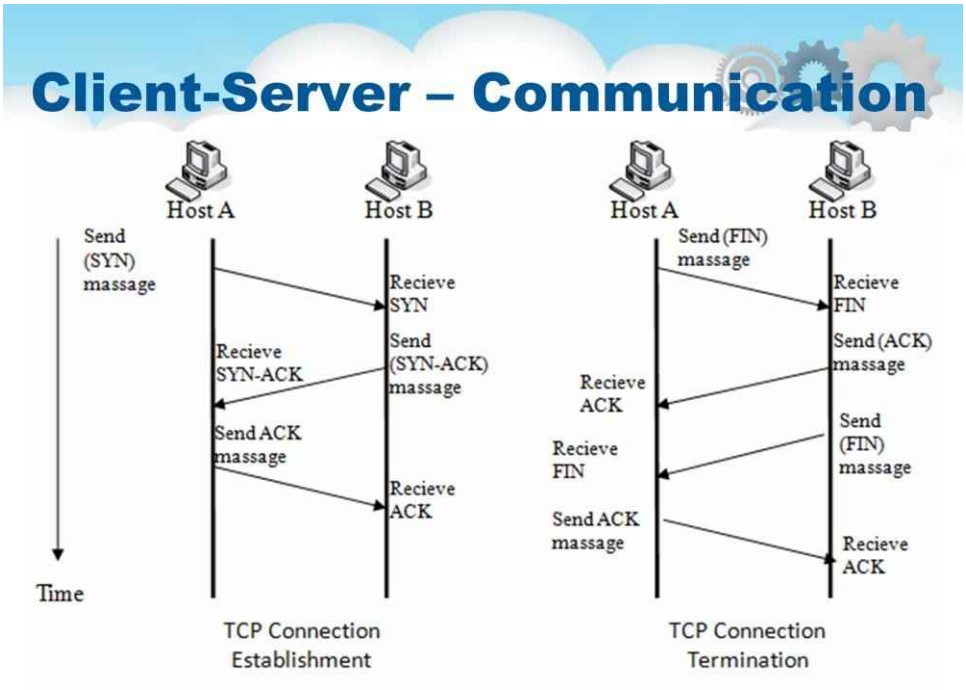
Зв'язок між клієнтом і сервером може бути реалізований за допомогою простого протоколу без з'єднання, коли базова мережа досить надійна, як у багатьох локальних мережах.

У цих випадках, коли клієнт запитує послугу, він просто пакує повідомлення для сервера, ідентифікуючи послугу, яку він хоче, разом із необхідними вхідними даними. Потім повідомлення надсилається на сервер.

Останній, у свою чергу, завжди чекатиме вхідного запиту, потім оброблятиме його та запаковуватиме результати у повідомлення-відповідь, яке потім надсилатиметься клієнту.



Використання протоколу без підключення має очевидну перевагу ефективності. Поки повідомлення не втрачаються чи не пошкоджуються, щойно намальований протокол запиту/відповіді працює нормально. На жаль, зробити протокол стійким до випадкових збоїв передачі не є тривіальним. Єдине, що ми можемо зробити, це, можливо, дозволити клієнту повторно надіслати запит, коли повідомлення відповіді не надійде. Проблема, однак, полягає в тому, що клієнт не може виявити, чи було втрачено вихідне повідомлення запиту, чи не вдалося передати відповідь. Якщо відповідь була втрачена, повторне надсилання запиту може призвести до виконання операції двічі. Якби операція була щось на кшталт «переказати 10 000 доларів США з мого банківського рахунку», тоді, очевидно, було б краще, щоб ми просто повідомили про помилку. З іншого боку, якщо операція була "



Як альтернатива, багато клієнт-серверних систем використовують надійний протокол, орієнтований на підключення. Хоча це рішення не зовсім доцільне в локальній мережі через відносно низьку продуктивність, воно ідеально працює в глобальних системах, у яких зв’язок за своєю суттю є ненадійним. Наприклад, практично всі протоколи додатків Інтернету базуються на надійних IP-з'єднаннях TCPI. У цьому випадку, коли клієнт запитує послугу, він спочатку встановлює з’єднання з сервером перед тим, як надсилати запит. Сервер зазвичай використовує те саме з’єднання для надсилання відповідного повідомлення, після чого з’єднання розривається. Проблема в тому, що встановлення та розірвання з'єднання є відносно дорогим, особливо коли повідомлення запиту та відповіді невеликі.

[ЛІВА СТОРОНА] Давайте розглянемо спілкування, орієнтоване на з’єднання, у «тристоронньому рукостисканні» **для встановлення з'єднання TCP**:

1. Перша стрілка - запит на синхронізацію (називається SYN)
2. Друга стрілка є підтвердженням синхронізації (називається SYN-ACK).
3. Третя стрілка також є підтвердженням, щоб повідомити одержувачу, що з’єднання встановлено (називається ACK).

[ПРАВА СТОРОНА] І аналогічна послідовність дій **для завершення з'єднання TCP:**

1. Хост A, якому необхідно припинити з’єднання, надсилає спеціальне повідомлення з прапором FIN (завершення), вказуючи, що він завершив надсилання даних.
2. Хост B, який отримує сегмент FIN, не розриває з’єднання, а переходить у стан «пасивного закриття» (CLOSE_WAIT) і надсилає ACK для FIN назад до хосту A. Тепер хост B переходить у стан LAST_ACK . У цей момент хост B більше не прийматиме дані від хоста A, але може продовжувати передавати дані хосту A. Якщо хост B не має даних для передачі хосту A, він також припинить з’єднання, надіславши FIN-сегмент. 3. Коли хост A отримує останній ACK від хосту B, він переходить у стан (TIME_WAIT) і надсилає ACK назад хосту B.
4. Хост B отримує ACK від хоста A та закриває з’єднання.

The previously mentioned layers of architectural style (in the context of distributed data):

- The user-interface level
- The processing level
- The data level

Протягом багатьох років модель клієнт-сервер була предметом багатьох дебатів і суперечок. Одним із головних питань було те, як провести чітке розмежування між клієнтом і сервером. Не дивно, що часто немає чіткого розмежування. Наприклад, сервер для розподіленої бази даних може постійно діяти як клієнт, оскільки він пересилає запити до різних файлових серверів, відповідальних за реалізацію таблиць бази даних. У такому випадку сам сервер бази даних, по суті, обробляє лише запити.

Однак, якщо клієнт-серверні програми націлені на підтримку доступу користувачів до баз даних, можна розглянути наступні три рівні:

1. Рівень інтерфейсу користувача – містить усе необхідне для безпосереднього взаємодії з користувачем, наприклад керування дисплеєм.

2. Рівень обробки - зазвичай містить програми.

3. Рівень даних - керує фактичними даними, які обробляються цими програмами для користувачів.

• Clients typically implement the user-interface level

• The simplest user-interface program can be like a character-based screen

• Modern user interfaces offer very rich functionality

• Many client-server applications are consists of roughly three different pieces:
- a part that interacts with a user,
- a part that operates on a database or file system,
- a middle part that contains the core functionality of an application.

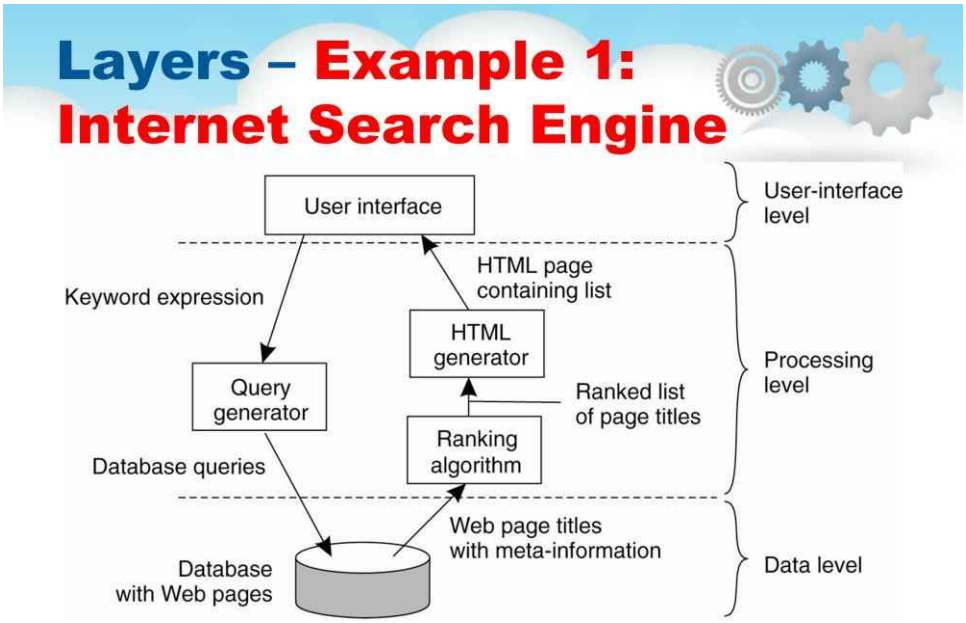
Клієнти зазвичай реалізують рівень інтерфейсу користувача. Цей рівень складається з програм, які дозволяють кінцевим користувачам взаємодіяти з програмами. Існує значна різниця в тому, наскільки складні програми з інтерфейсом користувача.

Найпростіша програма з інтерфейсом користувача — це не що інше, як символьний екран. Такий інтерфейс зазвичай використовується в середовищах мейнфреймів. У тих випадках, коли мейнфрейм контролює всю взаємодію, включаючи клавіатуру та монітор, навряд чи можна говорити про середовище клієнт-сервер. Однак у багатьох випадках термінал користувача виконує певну локальну обробку, таку як відтворення введених клавіш або підтримка інтерфейсів, подібних до форми, у яких повний запис потрібно відредагувати перед відправкою на головний комп’ютер.

Зараз навіть у середовищі мейнфреймів ми бачимо більш просунуті інтерфейси користувача. Як правило, клієнтська машина пропонує принаймні графічний дисплей, у якому використовуються спливаючі або розкривні меню, і в якому багато елементів керування екраном обробляються за допомогою миші замість клавіатури. Типові приклади таких інтерфейсів включають інтерфейси X-Windows, які використовуються в багатьох середовищах UNIX, і більш ранні інтерфейси, розроблені для ПК з MS-DOS і Apple Macintosh.

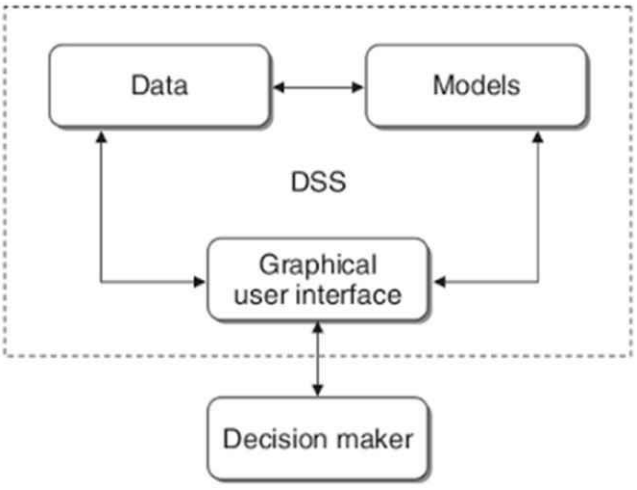
Сучасні інтерфейси користувача пропонують значно більше функціональних можливостей, дозволяючи програмам спільно використовувати одне графічне вікно та використовувати це вікно для обміну даними за допомогою дій користувача. Наприклад, щоб видалити файл, зазвичай можна перемістити піктограму, що представляє цей файл, на піктограму, що представляє кошик. Подібним чином багато текстових процесорів дозволяють користувачеві переміщувати текст у документі в іншу позицію, використовуючи лише мишу.

Багато клієнт-серверних програм можна створити приблизно з трьох різних частин: частини, яка обробляє взаємодію з користувачем, частини, яка працює з базою даних або файловою системою, і середньої частини, яка, як правило, містить основні функції програми. Ця середня частина логічно розміщена на рівні обробки. На відміну від користувальницьких інтерфейсів і баз даних, на рівні обробки не так багато спільних аспектів. Тому наведемо кілька прикладів, щоб зробити цей рівень більш зрозумілим.



Як перший приклад розглянемо пошукову систему в Інтернеті. Ігноруючи всі анімовані банери, зображення та інше химерне оформлення вітрин, інтерфейс користувача пошукової системи дуже простий: користувач вводить рядок ключових слів, а потім отримує список заголовків веб-сторінок. Внутрішня частина формується величезною базою даних веб-сторінок, які були попередньо вибрані та проіндексовані. Ядром пошукової системи є програма, яка перетворює рядок ключових слів користувача в один або кілька запитів до бази даних. Згодом він ранжує результати в список і перетворює цей список на серію HTML-сторінок. У моделі клієнт-сервер ця частина пошуку інформації зазвичай розміщується на рівні обробки.

Layers – Example 2: Decision Support System



Як другий приклад розглянемо систему підтримки прийняття рішень для біржової брокерської компанії. Подібно до пошукової системи, таку систему можна розділити на передню частину, що реалізує інтерфейс користувача, задню частину для доступу до бази даних із фінансовими даними та програми аналізу між цими двома. Аналіз фінансових даних може вимагати складних методів і технік зі статистики та штучного інтелекту. У деяких випадках ядро системи підтримки прийняття фінансових рішень може навіть знадобитися виконувати на високопродуктивних комп'ютерах, щоб досягти пропускної здатності та оперативності, які очікуються від її користувачів.

Загалом, система підтримки прийняття рішень розроблена трьома основними компонентами, а саме керування базою даних, базовою моделлю та програмною системою/інтерфейсом користувача.

1. Управління базами даних.

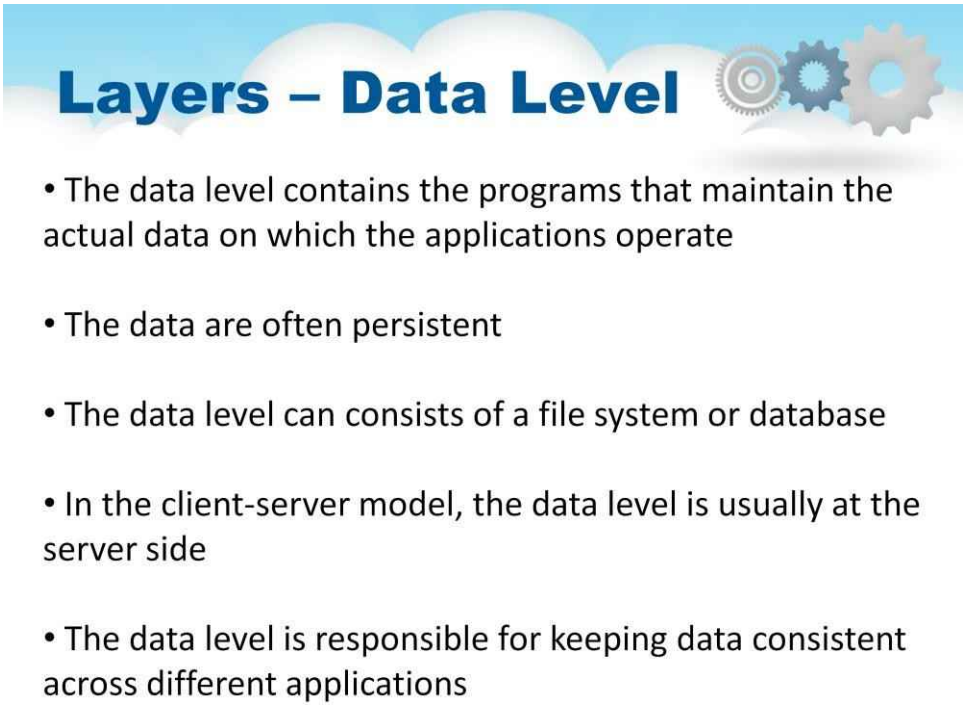
Це підсистема даних, організована в базі даних. Дані, які є системою підтримки прийняття рішень, можуть надходити ззовні та всередині середовища. Для цілей SPK необхідні дані, що мають відношення до проблеми, яку потрібно вирішити за допомогою моделювання.

2. База моделі.

Це модель, яка представляє проблему в кількісному форматі (математична модель як приклад) як основу для моделювання або прийняття рішень, включаючи мету, пов'язані компоненти, існуючі обмеження (обмеження) та пов'язані питання Інше. Базова модель дозволяє особам, які приймають рішення, аналізувати в цілому, розробляючи та порівнюючи альтернативні рішення

3. Інтерфейс користувача / Діалог керування.

Іноді називають підсистемою діалогу, злиття двох попередніх компонентів, а саме управління базою даних і бази моделей, включених у третій компонент (інтерфейс користувача), попередньо представлений у формі комп'ютерних моделей, які зрозумілі. Інтерфейс користувача відображає вивід системи для користувача та отримує вхідні дані від користувача в Систему підтримки прийняття рішень.



Рівень даних у клієнт-серверній моделі містить програми, які зберігають фактичні дані, на основі яких працюють додатки.

Важливою властивістю цього рівня є те, що дані часто є постійними, тобто навіть якщо жодна програма не запущена, дані десь зберігатимуться для наступного використання.

У своїй найпростішій формі рівень даних складається з файлової системи, але більш поширеним є використання повноцінної бази даних.

У моделі клієнт-сервер рівень даних зазвичай реалізується на стороні сервера.

Окрім простого зберігання даних, рівень даних, як правило, також відповідає за підтримку узгодженості даних у різних програмах. Коли використовуються бази даних, підтримка узгодженості означає, що такі метадані, як описи таблиць, обмеження введення та метадані, що стосуються програми, також зберігаються на цьому рівні. Наприклад, у випадку з банком ми можемо захотіти створити сповіщення, коли борг клієнта по кредитній картці досягає певної суми. Цей тип інформації можна підтримувати через тригер бази даних, який активує обробник для цього тригера у відповідний момент.

• In most business-oriented environments, the data level is organized as a relational database

• The data should be organized independently of the applications

• Using relational databases in the client-server model helps separate the processing level from the data level, as processing and data are considered independent

• The relational databases are not always the ideal choice

• If data operations are manipulations with objects - the data level is better to implement by an object-oriented or object-relational database

У більшості бізнес-орієнтованих середовищ рівень даних організований як реляційна база даних.

Тут важлива незалежність даних. Дані організовані незалежно від програм таким чином, що зміни в цій організації не впливають на програми, а програми також не впливають на організацію даних.

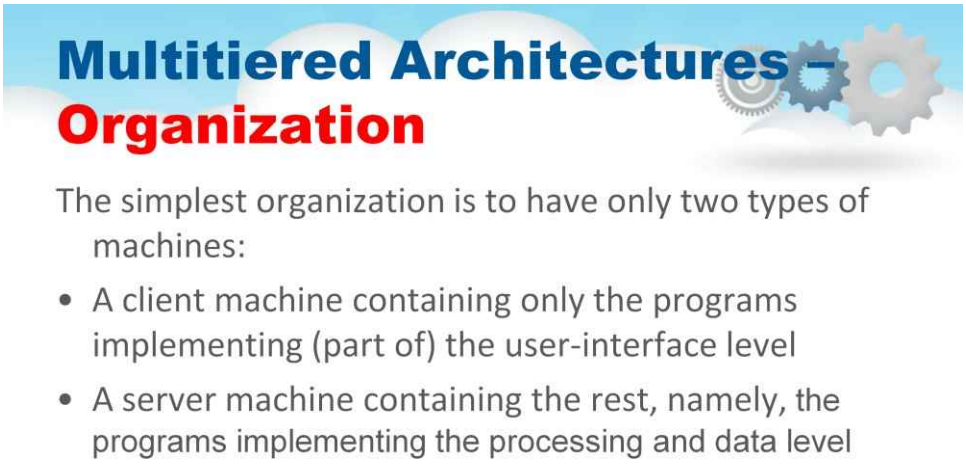
Використання реляційних баз даних у моделі клієнт-сервер допомагає відокремити рівень обробки від рівня даних, оскільки обробка та дані вважаються незалежними.

Однак реляційні бази даних не завжди є ідеальним вибором. Характерною особливістю багатьох програм є те, що вони працюють зі складними типами даних, які легше моделювати в термінах об’єктів, ніж у термінах відносин. Приклади таких типів даних варіюються від простих багатокутників і кіл до зображень конструкцій літаків, як у випадку з системами автоматизованого проектування (САПР).

У тих випадках, коли операції з даними легше виражаються в термінах маніпуляцій з об'єктами, має сенс реалізувати рівень даних за допомогою об'єктно-орієнтованої або об'єктно-реляційної бази даних. Примітно, що останній тип набув популярності, оскільки ці бази даних будуються на широко розсіяній моделі реляційних даних, водночас пропонуючи переваги, які дає об’єктно-орієнтована.

System Architectures – Multitiered Architectures

Багаторівневі архітектури

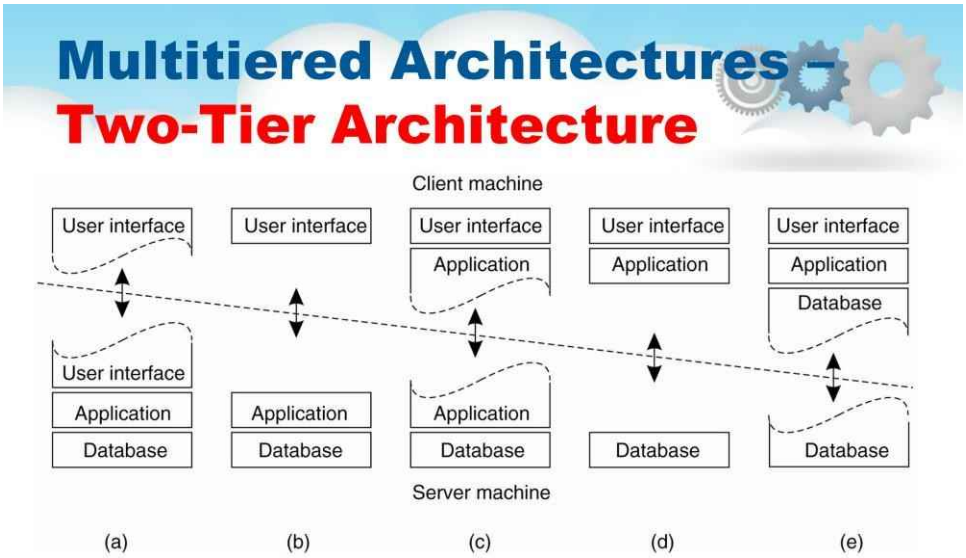


Поділ на три логічні рівні, як обговорювалося досі, передбачає низку можливостей для фізичного розподілу клієнт-серверної програми між кількома машинами.

Найпростіша організація - мати лише два типи машин:

1. Клієнтська машина, що містить лише програми, що реалізують (частину) рівень інтерфейсу користувача
2. Серверна машина, що містить решту, тобто програму, що реалізує обробку та рівень даних

У цій організації все обробляється сервером, тоді як клієнт, по суті, є не більш ніж дурним терміналом, можливо, з гарним графічним інтерфейсом. Є багато інших можливостей, деякі з яких ми розглянемо в цьому розділі.

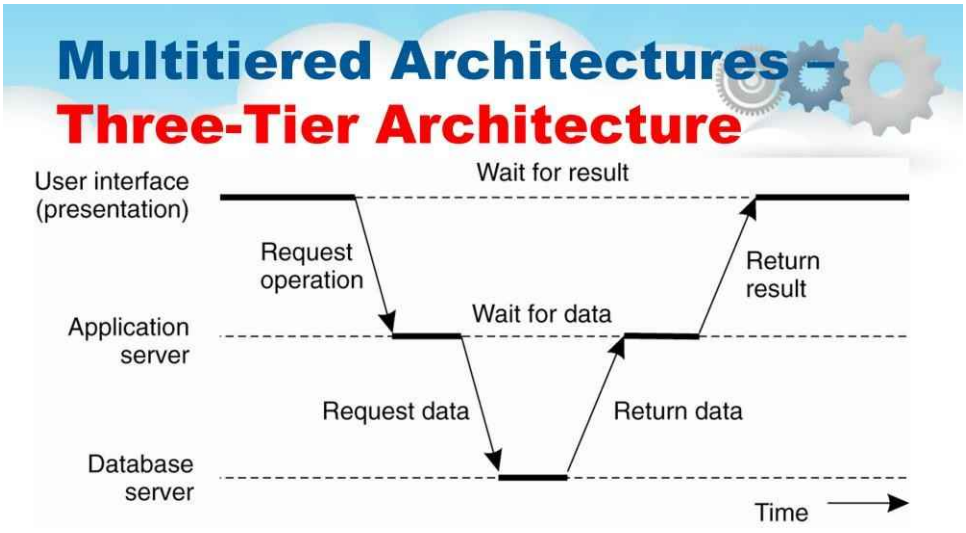


Один із підходів до організації клієнтів і серверів полягає в тому, щоб розподілити програми на прикладних рівнях попереднього розділу між різними машинами, як показано на цьому малюнку. Як перший крок, ми робимо різницю лише між двома типами машин: клієнтськими машинами та серверними машинами, що призводить до того, що також називається (фізично)**двоярусний** архітектура.

1) Одна з можливих організацій полягає в тому, щоб мати на клієнтській машині лише залежну від терміналу частину інтерфейсу користувача, як показано на малюнку (a), і надати додаткам віддалений контроль над представленням своїх даних.
2) Альтернативою є розміщення всього програмного забезпечення інтерфейсу користувача на стороні клієнта, як показано на малюнку (b). У таких випадках ми, по суті, поділяємо програму на графічну інтерфейсну частину, яка спілкується з рештою програми (розміщеною на сервері) за допомогою спеціального для програми протоколу. У цій моделі передній кінець (клієнтське програмне забезпечення) не виконує жодної обробки, крім необхідної для представлення інтерфейсу програми.
3) Продовжуючи цю лінію міркувань, ми також можемо перемістити частину програми до інтерфейсу, як показано на малюнку (c). Прикладом, коли це має сенс, є те, що програма використовує форму, яку потрібно повністю заповнити, перш ніж її можна буде обробити. Потім інтерфейс може перевірити правильність і узгодженість форми та, за необхідності, взаємодіяти з користувачем. Іншим прикладом організації на малюнку (c) є текстовий процесор, у якому основні функції редагування виконуються на стороні клієнта, де вони працюють з локально кешованими даними або даними в пам’яті. але де розширені інструменти підтримки, такі як перевірка орфографії та граматики, виконуються на стороні сервера.
4) У багатьох клієнт-серверних середовищах організації, показані на рисунках (d) і (e), є особливо популярними. Ці організації використовуються, коли клієнтською машиною є ПК або робоча станція, підключена через мережу до розподіленої файлової системи або бази даних.
По суті, більша частина програми працює на клієнтській машині, але всі операції з файлами або записами бази даних надходять на сервер. Наприклад, багато банківських програм працюють на машині кінцевого користувача, де користувач готує транзакції тощо. Після завершення програма зв’язується з базою даних на сервері банку та завантажує транзакції для подальшої обробки.

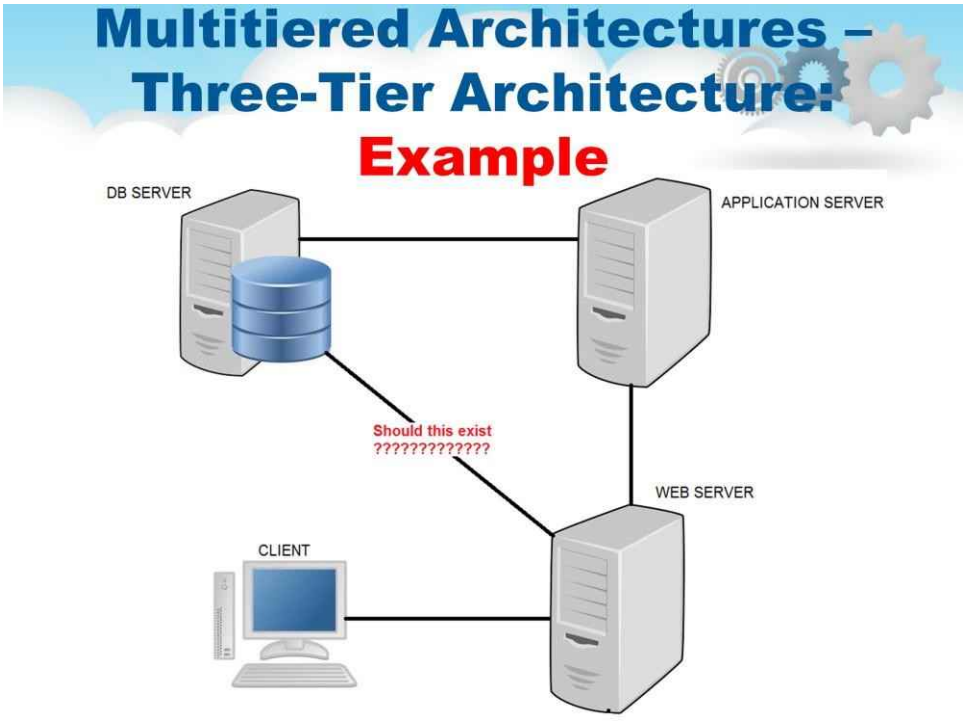
5) Рисунок (e) представляє ситуацію, коли локальний диск клієнта містить частину даних. Наприклад, під час перегляду веб-сторінок клієнт може поступово створити величезний кеш на локальному диску останніх перевірених веб-сторінок.

Ми зауважуємо, що протягом кількох років спостерігалася сильна тенденція до відмови від конфігурацій, показаних на рисунках (d) і (e), у випадку, коли клієнтське програмне забезпечення розміщується на машинах кінцевих користувачів. У цих випадках більша частина обробки та зберігання даних виконується на стороні сервера. Причина цього проста: хоча клієнтські машини роблять багато, ними також проблематичніше керувати. Наявність більшої функціональності на клієнтській машині робить клієнтське програмне забезпечення більш схильним до помилок і більш залежним від основної платформи клієнта (тобто операційної системи та ресурсів). З точки зору управління системою наявність так званих жирних клієнтів не є оптимальним. Натомість тонкі клієнти, представлені організаціями, показаними на рисунку (a)-(c), набагато простіші,



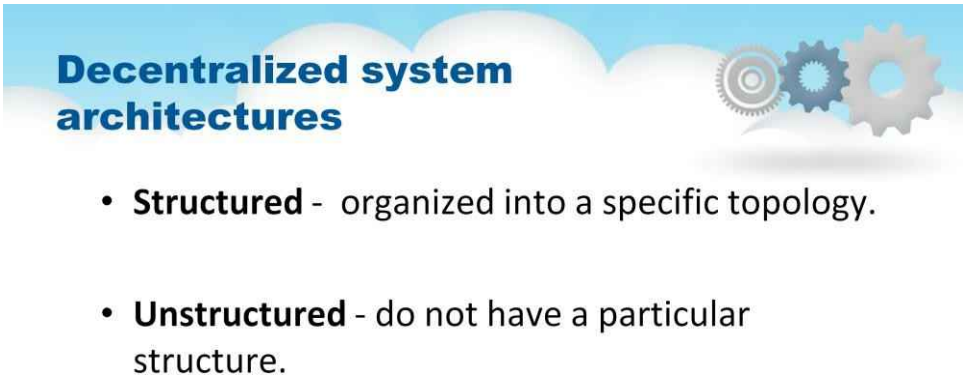
Зауважте, що ця тенденція не означає, що нам більше не потрібні розподілені системи. Навпаки, ми бачимо, що рішення на стороні сервера стають все більш розподіленими, оскільки один сервер замінюється декількома серверами, які працюють на різних машинах. Зокрема, розрізняючи лише клієнтські та серверні машини, як ми це робили досі, ми пропускаємо те, що сервер іноді може діяти як клієнт, як показано на цьому малюнку, що призводить до (фізично)**триярусний** архітектура.

У цій архітектурі програми, які є частиною рівня обробки, знаходяться на окремому сервері, але додатково можуть бути частково розподілені між клієнтською та серверною машинами.
Типовим прикладом використання трирівневої архітектури є обробка транзакцій, де окремий процес, який називається монітором обробки транзакцій, координує всі транзакції між, можливо, різними серверами даних.



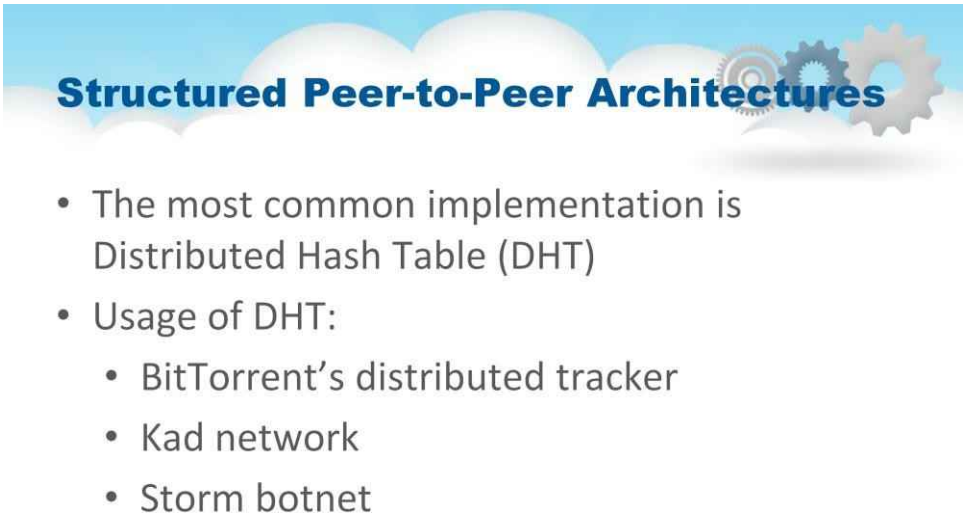
Ще один, але зовсім інший приклад, коли ми часто бачимо трирівневу архітектуру, – це організація веб-сайтів. У цьому випадку веб-сервер діє як точка входу на сайт, передаючи запити на сервер додатків, де відбувається фактична обробка. Цей сервер додатків, у свою чергу, взаємодіє з сервером бази даних. Наприклад, сервер додатків може бути відповідальним за запуск коду для перевірки наявного асортименту деяких товарів, які пропонує електронний книжковий магазин. Для цього може знадобитися взаємодія з базою даних, що містить необроблені дані інвентаризації.

Децентралізовані архітектури



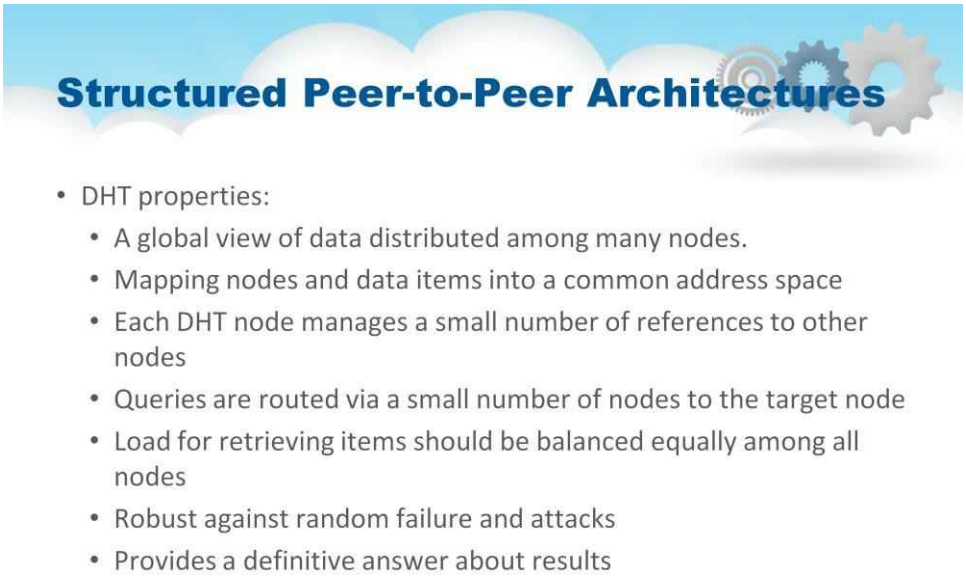
Структурований – організований у певну топологію, а протокол гарантує, що будь-який вузол може ефективно шукати в мережі файл/ресурс, навіть якщо ресурс надзвичайно рідкісний.

Неструктуровані – не накладають певну структуру на накладену мережу за проектом, а утворюються вузлами, які випадковим чином утворюють зв’язки один з одним.



У структурованій одноранговій архітектурі оверлейна мережа будується за допомогою детермінованої процедури.

Найбільш використовувана процедура полягає в організації процесів через розподілену хеш-таблицю (DHT). У системі на основі DHT елементам даних призначається випадковий ключ із великого простору ідентифікаторів, наприклад 128- або 160-бітовий ідентифікатор. Так само вузлам у системі також призначається випадкове число з того самого простору ідентифікаторів.



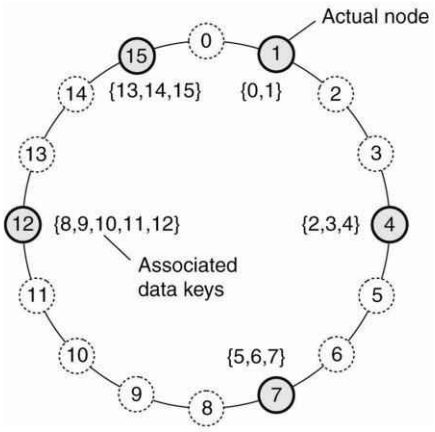
Тут наведено деякі властивості систем на основі DHT.

- Глобальний перегляд даних, розподілених між багатьма вузлами.
- Відображення вузлів і елементів даних у загальний адресний простір
- Кожен вузол DHT керує невеликою кількістю посилань на інші вузли
- Запити направляються через невелику кількість вузлів до цільового вузла
- Навантаження для отримання елементів має бути рівномірно збалансовано між усіма вузлами
- Стійкий проти випадкових збоїв і атак
- Надає остаточну відповідь про результати

Система на основі DHT повинна реалізовувати ефективну та детерміновану схему, яка однозначно відображає ключ елемента даних на ідентифікатор вузла на основі певної метрики відстані.

Найважливіше те, що під час пошуку елемента даних повертається мережева адреса вузла, відповідального за цей елемент даних. По суті, це досягається шляхом маршрутизації запиту на елемент даних до відповідального вузла.

Structured Peer-to-Peer Architectures



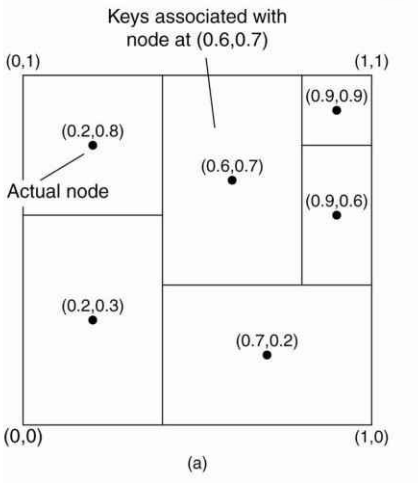
The mapping of data items onto nodes in Chord.

Наприклад, у системі Chord, запропонованій Stoica у 2003 році, вузли логічно організовані в кільце вузлів. Цей вузол називається наступником ключа k і позначається як succ(k).

Щоб знайти елемент даних, програма, що працює на довільному вузлі, викликає функцію LOOKUP(k), яка згодом повертає мережеву адресу succ(k). Крім того, програма може зв’язатися з вузлом, щоб отримати копію елемента даних.

Structured Peer-to-Peer Architectures

The mapping of data items onto nodes in CAN.



Інші підходи пропонуються в інших системах на основі DHT.

Як приклад, тут показано мережу з адресацією вмісту (CAN), яку запропонував Ratnasamy у 2001 році.

CAN використовує d-вимірний простір декартових координат, який повністю розділений між усіма вузлами, що беруть участь у системі.

Для простоти розглянемо лише двовимірний випадок.

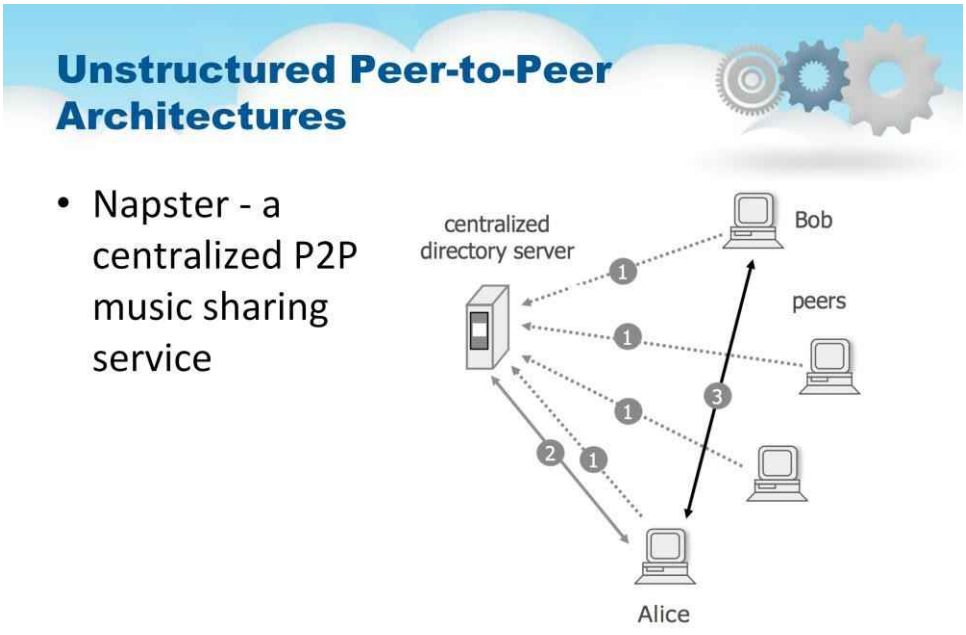
На цьому слайді показано, як поділяється двовимірний простір серед шести вузлів. Кожен вузол має пов’язану область. Кожному елементу даних у CAN буде призначено унікальну точку в цьому просторі, після чого також буде зрозуміло, який вузол відповідає за ці дані (не враховуючи елементи даних, які знаходяться на межі кількох регіонів, для яких використовується детермінізоване правило призначення) .

Неструктуровані однорангові системи в основному покладаються на рандомізовані алгоритми для побудови накладеної мережі.

Основна ідея полягає в тому, що кожен вузол підтримує список
сусідів, але цей список побудовано більш-менш випадковим чином.

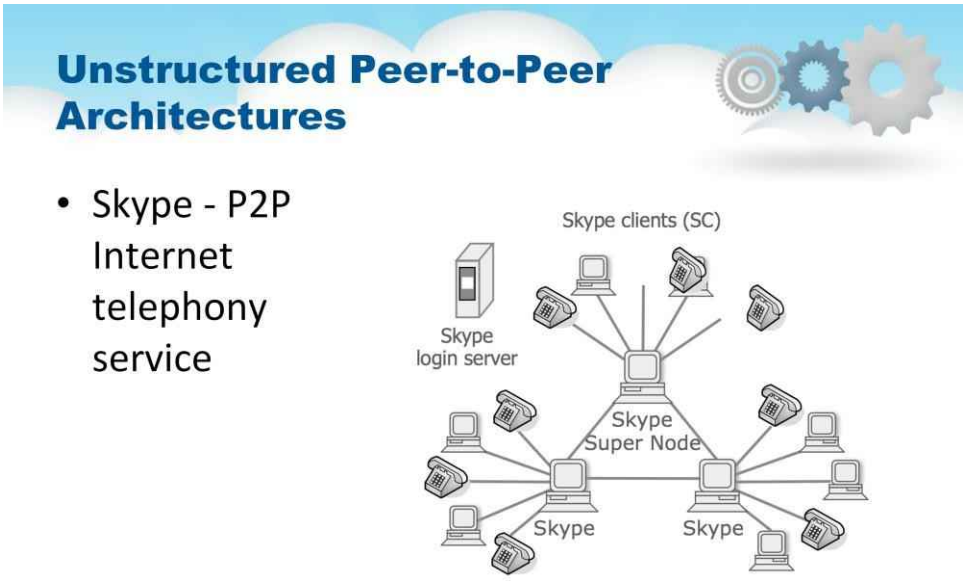
Передбачається, що елементи даних розміщуються на вузлах випадковим чином. Як наслідок, коли вузлу потрібно знайти певний елемент даних, єдине, що він може ефективно зробити, це заповнити мережу пошуковим запитом.

Неструктуровані мережі використовують затоплення та подібні опортуністичні методи, такі як випадкові блукання, розширене кільце, пошук за часом життя (TTL), щоб знайти однорангові мережі, які мають цікаві елементи даних.



Napster — це назва сумно відомих онлайн-сервісів, орієнтованих на музику. Він був заснований як піонерська однорангова (P2P) служба обміну файлами в Інтернеті для обміну цифровими аудіофайлами, як правило, аудіопіснями, закодованими у форматі MP3. Тут показана принципова схема Napster.

Його закрили за рішенням суду.
Пізніше компанії та проекти успішно наслідували приклад P2P обміну файлами, такі як Gnutella, Freenet, Kazaa, BearShare та багато інших.



Skype використовує власну мережу Інтернет-телефонії (VoIP), яка називається протоколом Skype.

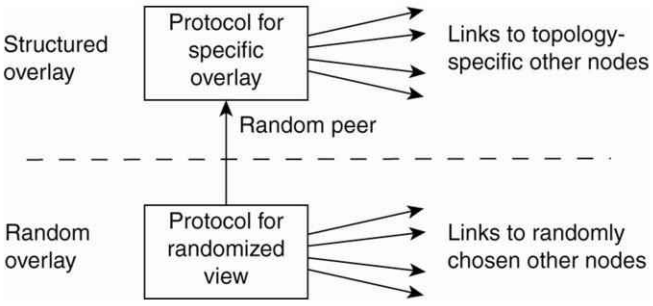
Частина технології Skype покладається на протокол P2P Global Index. Основна відмінність між Skype і стандартними VoIP-клієнтами полягає в тому, що Skype працює за одноранговою моделлю (спочатку на основі програмного забезпечення Kazaa), а не за звичнішою моделлю клієнт-сервер.

Тут показана принципова схема моделі Skype P2P.

Примітка. Дуже популярна модель протоколу ініціації сеансу (SIP) VoIP також є одноранговою, але реалізація зазвичай вимагає реєстрації на сервері, як і Skype.

Примітка. 20 червня 2014 року Microsoft оголосила про припинення підтримки старого протоколу Skype. Новий протокол Skype—Microsoft Notification Protocol 24.

Topology Management of Overlay Networks



A two-layered approach for constructing and maintaining specific overlay topologies using techniques from unstructured peer-to-peer systems.

Можна побудувати та підтримувати конкретні топології накладених мереж.

Таке управління топологією досягається шляхом застосування дворівневого підходу, як показано на цьому слайді.

Найнижчий рівень являє собою неструктуровану однорангову систему, в якій вузли періодично обмінюються записами своїх часткових переглядів з метою підтримки точного випадкового графа.

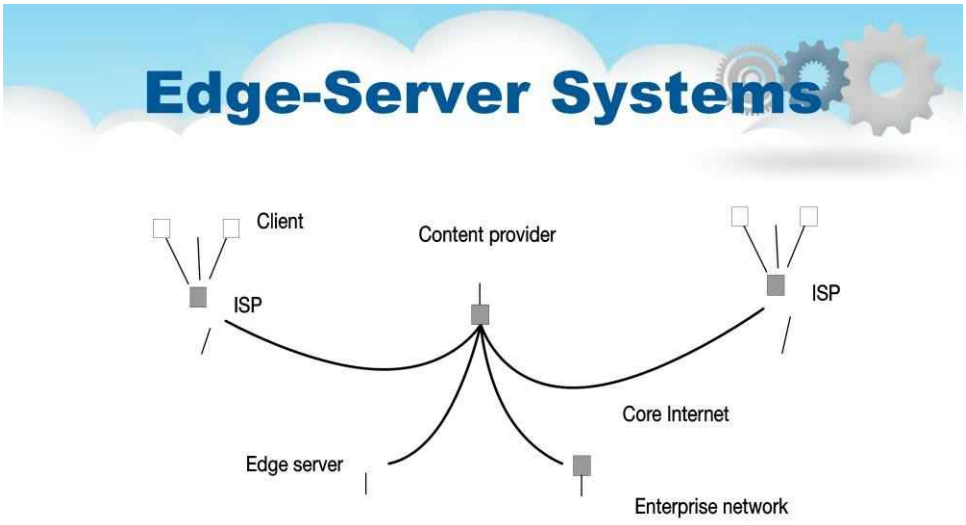
Найнижчий шар передає свій частковий вигляд на вищий рівень, де відбувається додатковий вибір записів. Потім це призводить до другого списку сусідів, що відповідає бажаній топології.

Гібридні архітектури

Поки що ми зосереджувалися на архітектурі клієнт-сервер і ряді одноранговых архітектур. Багато розподілених систем поєднують архітектурні особливості, як ми вже зустрічали в супероднорангових мережах. У цьому розділі ми розглянемо деякі конкретні класи розподілених систем, у яких клієнт-серверні рішення поєднуються з децентралізованою архітектурою.

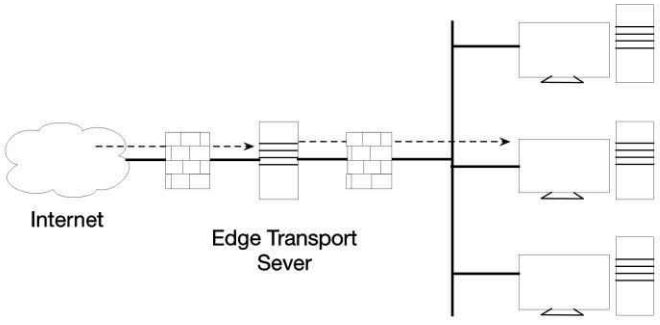
- Servers are placed ”at the edge” of the network.
- Purpose is to serve content after filtering and transcoding functions to data.

Важливий клас розподілених систем, організованих відповідно до гібридної архітектури, утворюють системи крайових серверів. Ці системи розгортаються в Інтернеті, де сервери розміщені «на краю» мережі.



- Figure 2-13. Viewing the Internet as consisting of a collection of edge servers.

Це межа утворена межею між корпоративними мережами та реальним Інтернетом, наприклад, як це надає постачальник послуг Інтернету (ISP). Подібним чином, якщо кінцеві користувачі вдома підключаються до Інтернету через свого провайдера, можна вважати, що провайдер перебуває на межі Інтернету. Це призводить до загальної організації, як показано на цьому слайді.

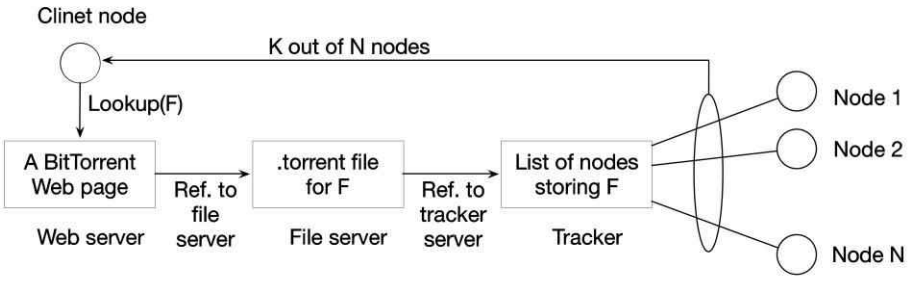


Кінцеві користувачі або клієнти загалом підключаються до Інтернету за допомогою периферійного сервера. Основна мета периферійного сервера — обслуговувати вміст, можливо, після застосування функцій фільтрації та перекодування. Більш цікавим є той факт, що набір периферійних серверів можна використовувати для оптимізації розповсюдження вмісту та програм. Основна модель полягає в тому, що для конкретної організації один периферійний сервер діє як вихідний сервер, з якого надходить увесь вміст. Цей сервер може використовувати інші периферійні сервери для копіювання веб-сторінок тощо.

- Hybrid structures are notably deployed in collaborative distributed systems.
- The main issue in many of these systems to first get started, for which often a traditional client-server scheme is deployed. Once a node has joined the system, it can use a fully decentralized scheme for collaboration.

Гібридні структури, зокрема, розгортаються в розподілених системах для спільної роботи. Основна проблема в багатьох із цих систем — спочатку почати роботу, для чого часто розгортається традиційна схема клієнт-сервер. Після приєднання вузла до системи він може використовувати повністю децентралізовану схему для співпраці.

Collaborative Distributed Systems (1)



The principal working of BitTorrent [adapted with permission from Pouwelse et al. (2004)].

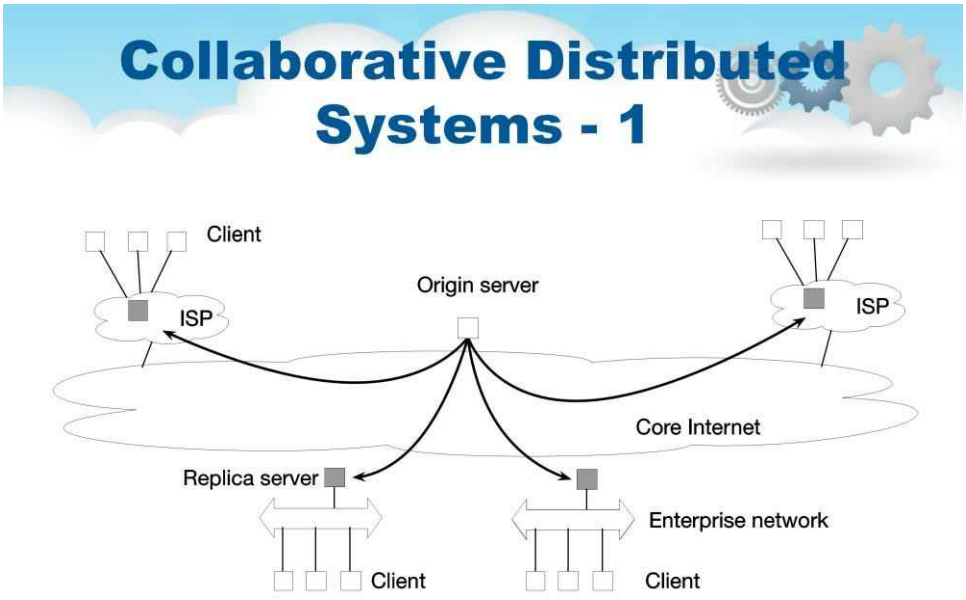
Розглянемо файлообмінну систему BitTorrent.

BitTorrent — це однорангова система завантаження файлів. Основна його робота показана на цьому слайді.

Основна ідея полягає в тому, що коли кінцевий користувач шукає файл, він завантажує фрагменти файлу від інших користувачів, доки завантажені фрагменти не можуть бути зібрані разом, щоб отримати повний файл. Важливою метою дизайну було забезпечення співпраці. У більшості систем обміну файлами значна частка учасників просто завантажує файли, але в іншому майже нічого не робить. З цією метою файл можна завантажити лише тоді, коли клієнт завантаження надає вміст комусь іншому.

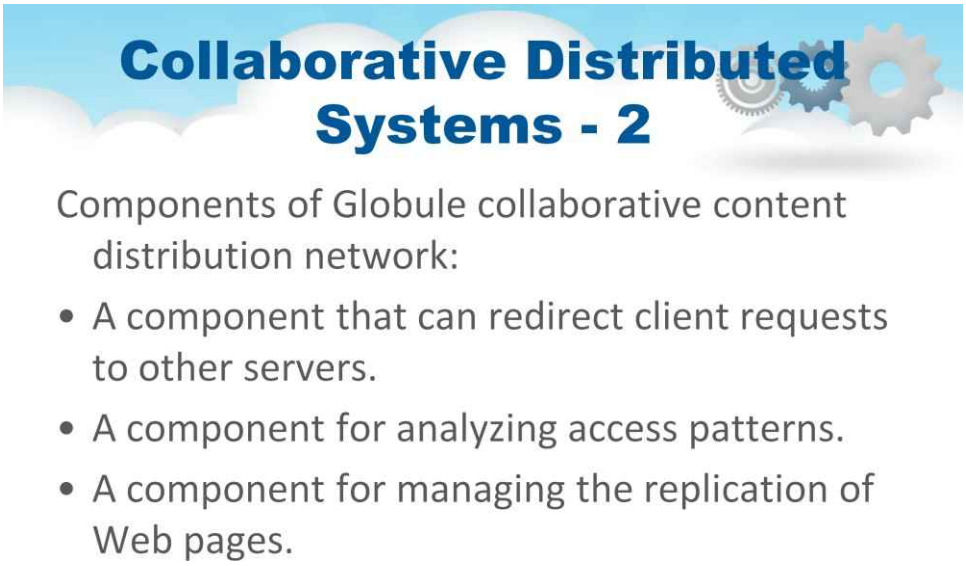
Щоб завантажити, користувач має отримати доступ до глобального каталогу, який є лише одним із небагатьох відомих веб-сайтів. Такий каталог містить посилання на файли .torrent. Файл .torrent містить інформацію, необхідну для завантаження певного файлу. Зокрема, це відноситься до того, що відомо як трекер, який є сервером, який веде точний облік активних вузлів, які мають (фрагменти) запитуваного файлу. Активний вузол – це вузол, який зараз завантажує інший файл.

Очевидно, що буде багато різних трекерів, хоча (як правило, буде лише один трекер на файл (або колекцію файлів). Після визначення вузлів, звідки можна завантажувати фрагменти, вузол завантаження фактично стає активним.



Очевидно, що BitTorrent поєднує централізовані та децентралізовані рішення. Як виявилося, вузьке місце системи, як не дивно, утворюють трекери. Як інший приклад розглянемо мережу спільного розподілу контенту Globule, запропоновану П’єром і ван Стіном у 2006 році.

Globule дуже нагадує згадану вище архітектуру edgeserver. У цьому випадку замість периферійних серверів кінцеві користувачі (а також організації) добровільно надають розширені веб-сервери, здатні співпрацювати в реплікації веб-сторінок.

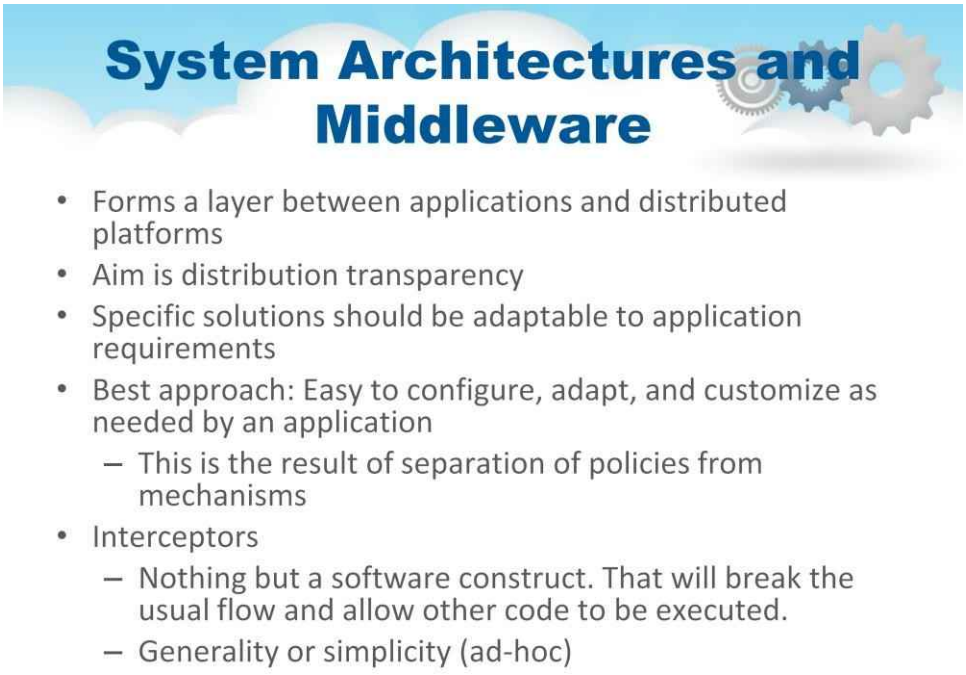


Globule — це децентралізована розподілена система.
Запити на веб-сайт спочатку пересилаються на сервер, після чого вони можуть бути перенаправлені на один з інших серверів. Також підтримується розподілене перенаправлення.

Globule також має централізований компонент у вигляді свого брокера. Посередник відповідає за реєстрацію серверів і надання інформації про ці сервери іншим. Сервери спілкуються з брокером повністю аналогічно тому, що можна очікувати в системі клієнт-сервер. З міркувань доступності брокер можна реплікувати, але цей тип реплікації широко застосовується для досягнення надійних обчислень клієнт-сервер.

Проміжне програмне забезпечення

Розглядаючи питання архітектури, які ми обговорювали досі, виникає питання про те, де вписується проміжне програмне забезпечення. Як ми обговорювали раніше, проміжне програмне забезпечення утворює прошарок між програмами та розподіленими платформами. Важливою метою є забезпечення певної прозорості розповсюдження, тобто до певної міри приховування розповсюдження даних, обробки та контролю від програм.



Системи проміжного програмного забезпечення фактично відповідають певному архітектурному стилю.

Наприклад, багато рішень проміжного програмного забезпечення прийняли об’єктно-орієнтований архітектурний стиль, такий як CORBA. Але інші, такі як TIB/Rendezvous, надають проміжне програмне забезпечення, яке відповідає архітектурному стилю, заснованому на подіях.

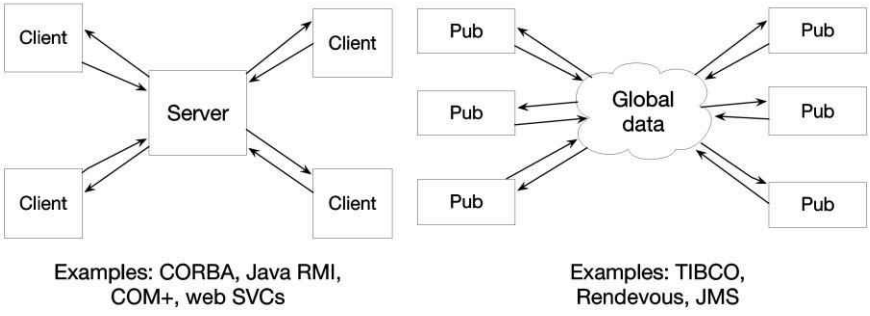
Якщо проміжне програмне забезпечення формується відповідно до певного архітектурного стилю, то це має перевагу в тому, що розробка програм може стати простішою. Однак очевидним недоліком є те, що проміжне програмне забезпечення більше не може бути оптимальним для того, що мав на увазі розробник програми.

Наприклад, деякі проміжні програми спочатку можуть пропонувати лише об’єкти, які можуть бути викликані віддаленими клієнтами. Пізніше ця форма взаємодії стала занадто обмежувальною, тому були додані інші моделі взаємодії, такі як обмін повідомленнями. Очевидно, що додавання нових функцій може легко призвести до незручних рішень проміжного програмного забезпечення. Крім того, проміжне програмне забезпечення має забезпечувати прозорість розповсюдження, а конкретні рішення мають адаптуватися до вимог програми.

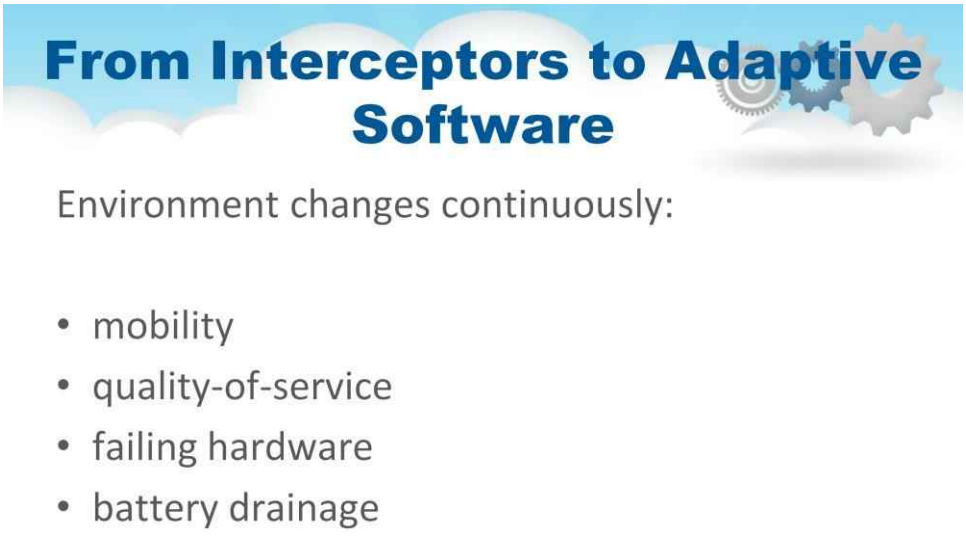
Одним із рішень цієї проблеми є створення кількох версій системи проміжного програмного забезпечення, де кожна версія адаптована до певного класу програм.

Але кращий підхід полягає в тому, щоб зробити системи проміжного програмного забезпечення такими, щоб їх було легко конфігурувати, адаптувати та налаштовувати відповідно до потреб програми. Як наслідок, зараз розробляються системи, в яких проводиться більш суворий розподіл між політиками та механізмами.

Давайте розглянемо деякі з поширених підходів.



Концептуально перехоплювач — це не що інше, як програмна конструкція, яка порушує звичайний потік керування та дозволяє виконувати інший (спеціальний для програми) код (як показано на цьому слайді). Для того, щоб зробити перехоплювачі загальними, може вимагатися значних зусиль щодо впровадження, і незрозуміло, чи слід у таких випадках віддавати перевагу загальності над обмеженою застосовністю та простотою. Крім того, у багатьох випадках наявність лише обмежених засобів перехоплення покращить керування програмним забезпеченням і розподіленою системою в цілому.



Насправді перехоплювачі пропонують засоби для адаптації проміжного ПЗ. Необхідність адаптації виникає через те, що середовище, в якому виконуються розподілені програми, постійно змінюється. Зміни, зокрема, викликані мобільністю, сильною різницею в якості обслуговування мереж, несправністю апаратного забезпечення та розрядкою акумулятора. Замість того, щоб додатки відповідали за реагування на зміни, це завдання поміщається в проміжне програмне забезпечення. Ці сильні впливи навколишнього середовища змусили багатьох розробників проміжного програмного забезпечення розглянути можливість створення адаптивного програмного забезпечення.

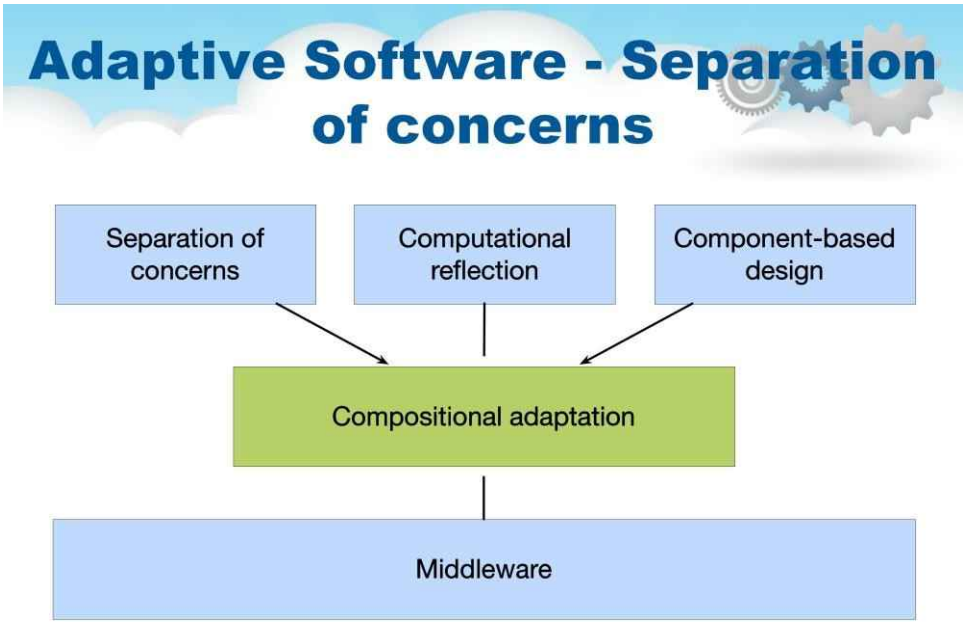
Three basic approaches to adaptive software:

- Separation of concerns
- Computational reflection
- Component-based design

Однак адаптивне програмне забезпечення виявилося не таким успішним, як очікувалося. Оскільки багато дослідників і розробників вважають це важливим аспектом сучасних розподілених систем, зупинимося на ньому коротко.

Для адаптації програмного забезпечення використовуються наступні три основні техніки:

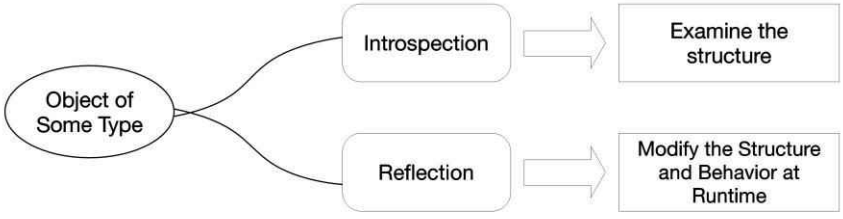
- Поділ турбот
- Обчислювальна рефлексія
- Компонентний дизайн



Розділення турбот

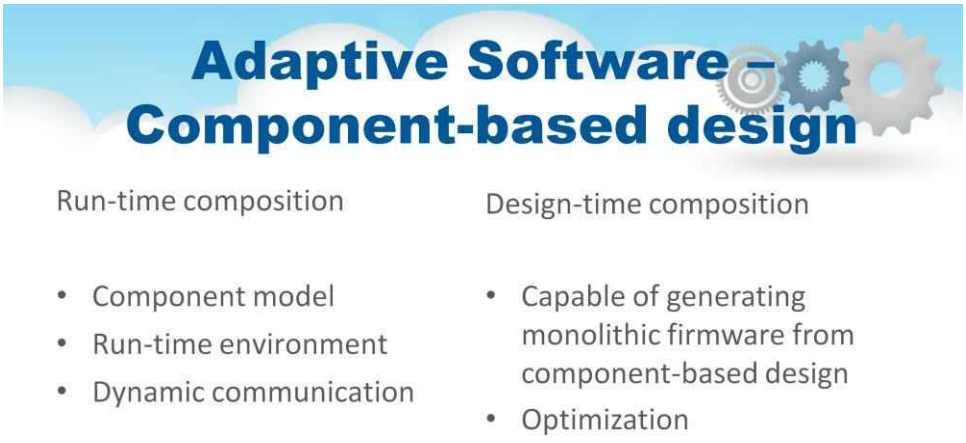
Це стосується традиційного способу модульної системи: відокремте частини, які реалізують функціональні можливості, від тих, які піклуються про інші речі (відомі як додаткові функції), такі як надійність, продуктивність, безпека тощо.

Але розробка проміжного програмного забезпечення для розподілених програм здебільшого пов’язана з обробкою додаткових функціональних можливостей незалежно від програм. Основна проблема полягає в тому, що ми не можемо легко відокремити ці додаткові функції за допомогою модулярізації. Наприклад, просте розміщення безпеки в окремому модулі не спрацює. Так само важко уявити, як відмовостійкість можна виділити в окрему коробку і продавати як окрему послугу. Розділення та подальше вплетення цих наскрізних проблем у (розподілену) систему є головною темою аспектно-орієнтованої розробки програмного забезпечення. Однак аспектна орієнтація ще не була успішно застосована для розробки великомасштабних розподілених систем, і можна очікувати, що попереду ще довгий шлях, перш ніж вона досягне цієї стадії.



Обчислювальна рефлексія

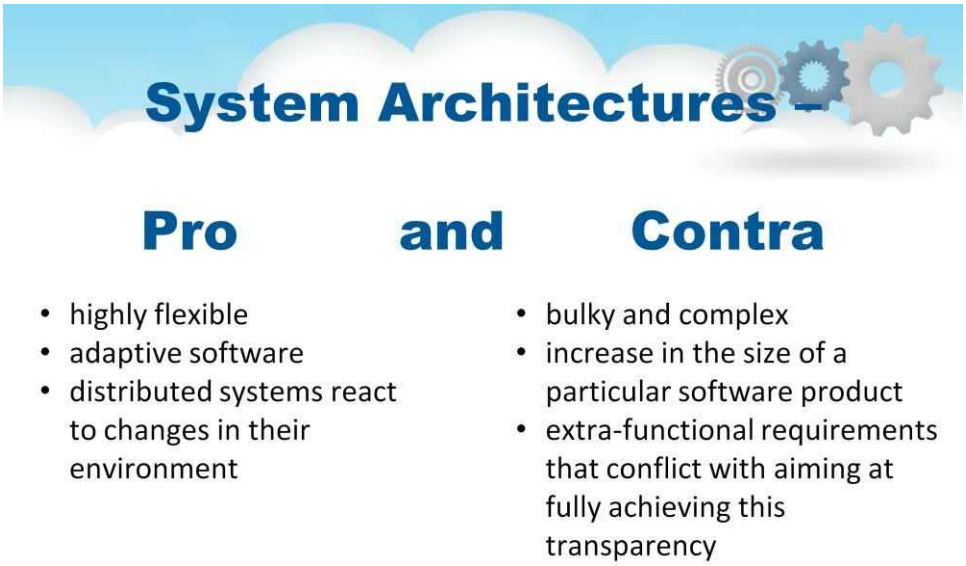
Це стосується здатності програми перевіряти себе та, якщо необхідно, адаптувати свою поведінку. Reflection вбудовано в мови програмування, включаючи Java, і пропонує потужний засіб для модифікацій під час виконання. Крім того, деякі системи проміжного програмного забезпечення забезпечують засоби для застосування рефлексивних методів. Однак, як і у випадку з орієнтацією на аспекти, рефлексивне проміжне програмне забезпечення ще має зарекомендувати себе як потужний інструмент для управління складністю великомасштабних розподілених систем.



Компонентний дизайн

Він підтримує адаптацію через композицію. Система може бути налаштована статично під час проектування або динамічно під час виконання. Для останнього потрібна підтримка пізнього зв’язування, техніки, яка успішно застосована в середовищах мов програмування, а також для операційних систем, де модулі можна завантажувати та вивантажувати за бажанням.

Зараз проводяться дослідження для автоматичного вибору найкращої реалізації компонента під час виконання, але, знову ж таки, процес залишається складним для розподілених систем, особливо якщо врахувати, що для заміни одного компонента потрібно знати, яким буде вплив такої заміни на інші компоненти. .

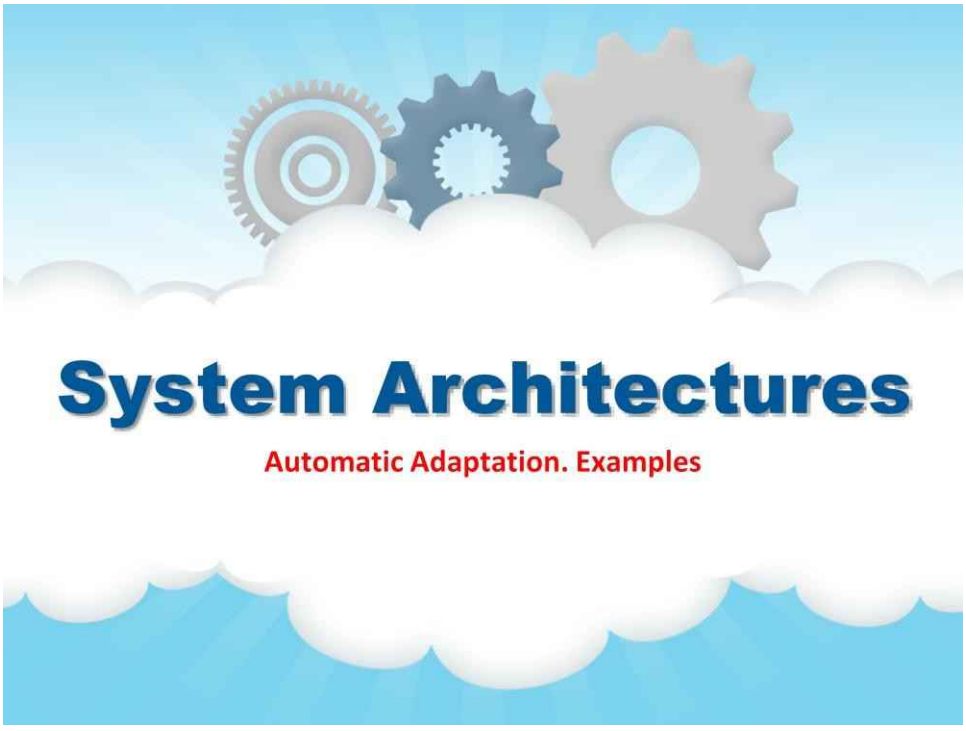


Тут наведено порівняння переваг і недоліків архітектури програмного забезпечення для розподілених систем.

Архітектура програмного забезпечення для розподілених систем (проміжне програмне забезпечення) є громіздкою та складною. Значною мірою ця громіздкість і складність виникають через необхідність бути загальними в тому сенсі, що необхідно забезпечити прозорість розподілу. Програми мають особливі додаткові функціональні вимоги, які суперечать прагненню досягти повної прозорості. Ці суперечливі вимоги до загальності та спеціалізації призвели до створення проміжного програмного забезпечення, яке є дуже гнучким.

Зараз усі великі програмні системи повинні працювати в мережевому середовищі, тому складність розподілених систем стає невід'ємною рисою та результатом спроб зробити розповсюдження прозорим. Основне припущення полягає в тому, що нам потрібне адаптивне програмне забезпечення в тому сенсі, що програмне забезпечення повинно змінюватися відповідно до змін середовища. Однак слід поставити питання, чи є адаптація до мінливого середовища вагомою причиною для зміни програмного забезпечення. Несправне апаратне забезпечення, атаки на безпеку, відведення енергії тощо, здається, є впливом навколишнього середовища, який може (і має) передбачатися програмним забезпеченням.

Найвагомішим і, безумовно, найвагомішим аргументом на користь адаптивного програмного забезпечення є те, що багато розподілених систем неможливо вимкнути. Це обмеження вимагає рішень для заміни та оновлення компонентів на льоту, але незрозуміло, чи якесь із запропонованих вище рішень є найкращим для вирішення цієї проблеми обслуговування.



Автоматична адаптація

Розподілені системи — і особливо пов’язане з ними проміжне програмне забезпечення — потребують загальних рішень для захисту від небажаних функцій, властивих мережам, щоб вони могли підтримувати якомога більше додатків. З іншого боку, повна прозорість розповсюдження — це не те, чого насправді хочуть більшість програм, що призводить до потреби підтримки окремих програмних рішень. Ми стверджували, що з цієї причини розподілені системи повинні бути адаптивними, але особливо коли мова йде про адаптацію їхньої поведінки виконання, а не програмних компонентів, які вони містять.

Насправді перехоплювачі пропонують засоби для адаптації проміжного ПЗ. Необхідність адаптації виникає через те, що середовище, в якому виконуються розподілені програми, постійно змінюється.

Зміни, зокрема, викликані мобільністю, сильною різницею в якості обслуговування мереж, несправністю апаратного забезпечення та розрядкою акумулятора.

Замість того, щоб додатки відповідали за реагування на зміни, це завдання поміщається в проміжне програмне забезпечення.

- Self-managing systems organize distributed systems as high-level feedback-control systems allowing automatic adaptions to changes.
- The automatic adaptations can be various:
  - self-managing,
  - self-configuring,
  - self-optimizing, and so on.

Розподілені системи повинні бути адаптивними, але особливо коли йдеться про адаптацію їхньої поведінки виконання, а не програмних компонентів, які вони містять. Нам потрібно організувати компоненти розподіленої системи таким чином, щоб можна було здійснювати моніторинг і коригування, а з іншого боку нам потрібно вирішити, де виконуватимуться процеси, що обробляють адаптацію

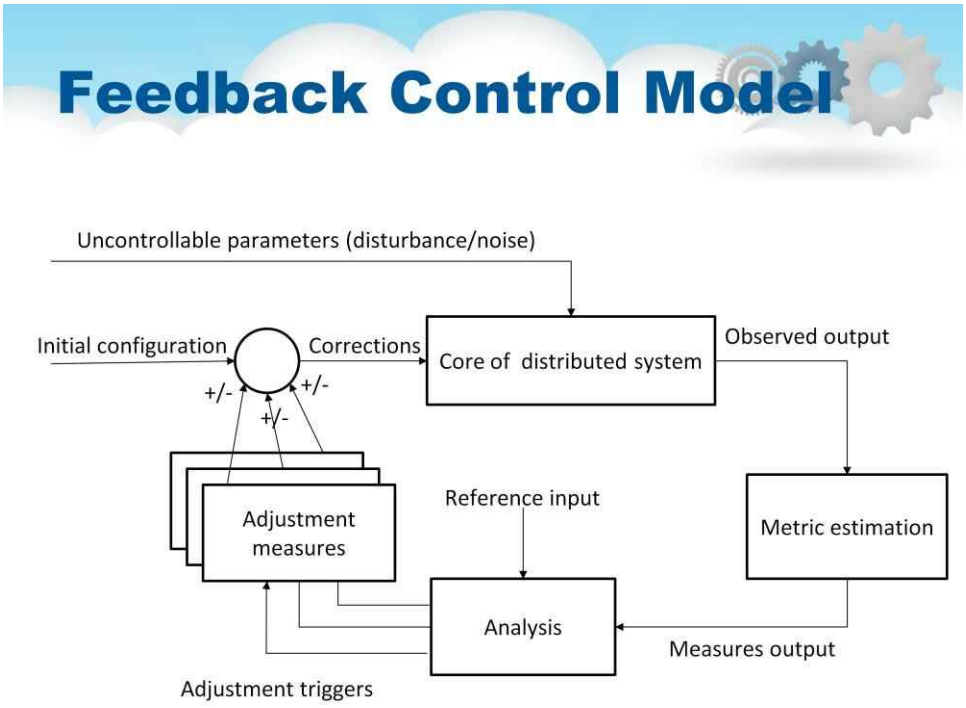
Самокеровані системи автономні обчислення або самокеровані системи організовують розподілені системи як високорівневі системи зворотного зв'язку та керування, що дозволяють автоматичну адаптацію до змін.

Різновиди, за якими вловлюються автоматичні адаптації, такі: самокерування,

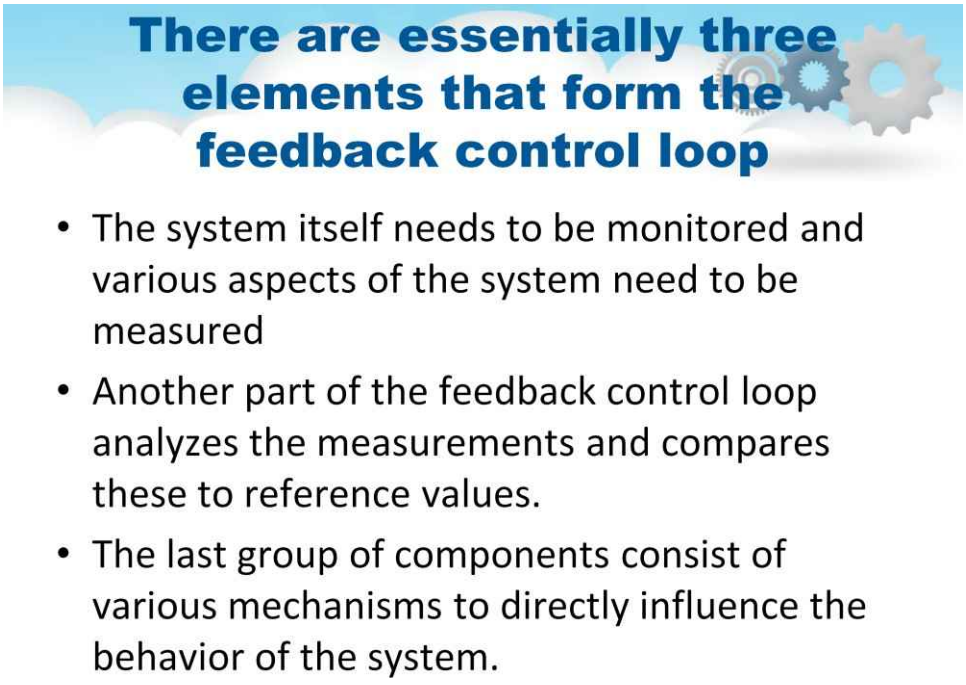
самовідновлення,

самоналаштування,

самооптимізація тощо.



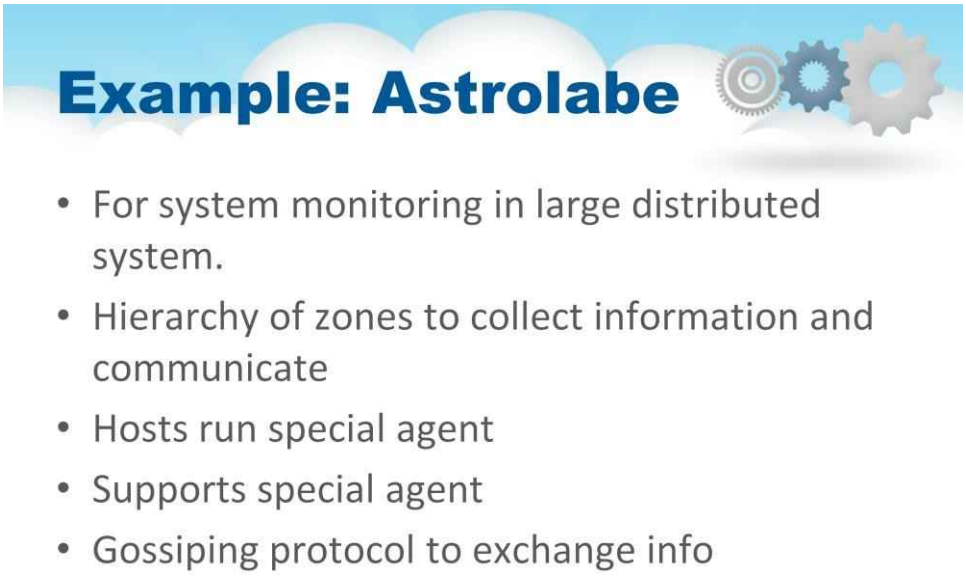
Ядро системи управління зі зворотним зв'язком утворюють компоненти, якими необхідно керувати. Передбачається, що ці компоненти керуються контрольованими вхідними параметрами, але на їх поведінку можуть впливати всі види неконтрольованих вхідних даних, також відомих як вхідні збурення або шуми. Хоча порушення часто надходять із середовища, в якому виконується розподілена система, цілком може статися так, що непередбачена взаємодія компонентів спричинить неочікувану поведінку.



Існують, по суті, три елементи, які утворюють контур керування зворотним зв’язком. 1. По-перше, необхідно контролювати саму систему, що вимагає вимірювання різних аспектів системи. У багатьох випадках вимірювання поведінки важко організувати. Наприклад, затримки в Інтернеті можуть сильно відрізнятися, а також залежати від того, що саме вимірюється. У таких випадках точна оцінка затримки може бути важкою, і з цих причин контур керування зворотним зв’язком зазвичай містить компонент оцінки логічної метрики.

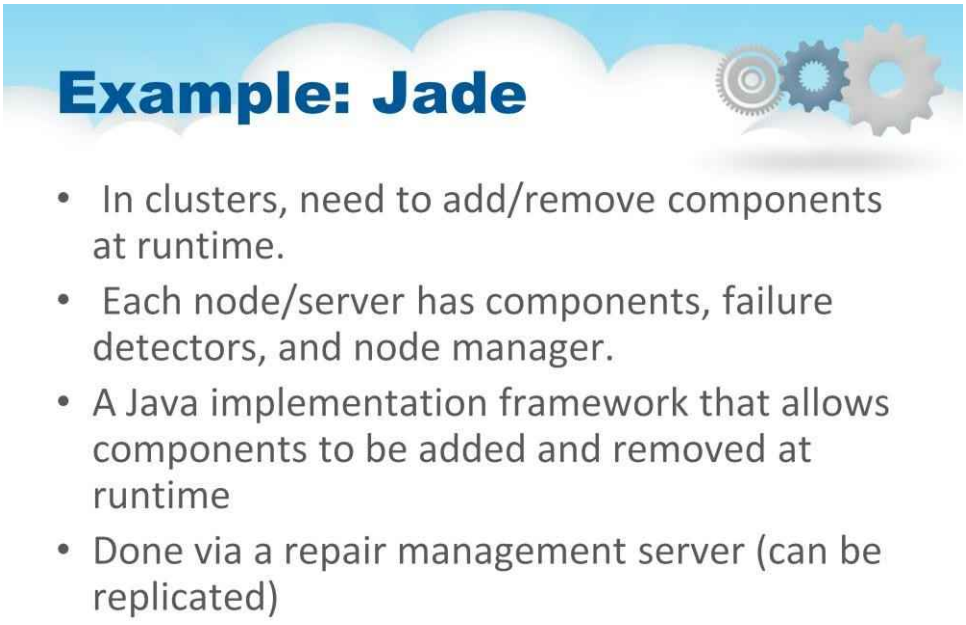
2. Інша частина контуру керування зворотним зв’язком аналізує вимірювання та порівнює їх із еталонними значеннями. Цей компонент аналізу зворотного зв’язку є серцевиною контуру керування, оскільки він міститиме алгоритми, які приймають рішення про можливі адаптації.

3. Остання група компонентів складається з різних механізмів безпосереднього впливу на поведінку системи. Може бути багато різних механізмів: розміщення реплік, зміна пріоритетів планування, перемикання служб, переміщення даних з міркувань доступності, перенаправлення запитів на різні сервери тощо. Компонент аналізу повинен знати про ці механізми та їхній (очікуваний) ефект на поведінку системи, тому він запустить один або кілька механізмів, щоб потім спостерігати ефект.



Для прикладу розглянемо систему Astrolabe (запропоновану Ван Ренессом у 2003 році), яка може підтримувати загальний моніторинг дуже великих розподілених систем. У контексті самокерованих систем Astrolabe слід позиціонувати як загальний інструмент для спостереження за поведінкою систем. Його вихідні дані можуть бути використані для передачі в компонент аналізу для прийняття рішення щодо коригувальних дій. Астролябія організовує велику колекцію хостів в ієрархію зон. Зони найнижчого рівня складаються лише з одного господаря, який згодом групується в зони збільшення розміру. Зона верхнього рівня охоплює всі хости. Кожен хост запускає процес Astrolabe, який називається агентом, який збирає інформацію про зони, в яких міститься цей хост. Агент також спілкується з іншими агентами з метою поширення інформації про зони по всій системі.

Кожен хост підтримує набір атрибутів для збору локальної інформації. Наприклад, хост може відстежувати конкретні файли, які він зберігає, використання ресурсів тощо. Доступні для запису лише ті атрибути, які підтримуються безпосередньо хостами, тобто на найнижчому рівні ієрархії. Кожна зона також може мати набір атрибутів, але значення цих атрибутів обчислюються зі значень зон нижчого рівня.

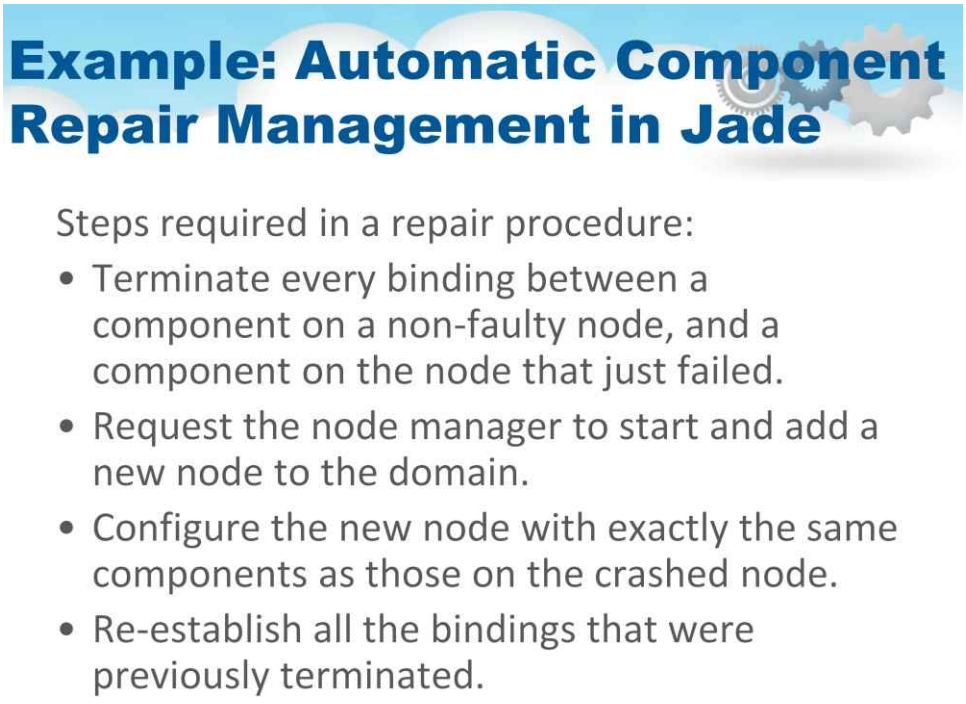


Під час підтримки кластерів комп’ютерів, на кожному з яких працюють складні сервери, стає важливим полегшити проблеми керування. Одним із підходів, який можна застосувати до серверів, побудованих із використанням компонентного підходу, є виявлення несправностей компонентів і їх автоматична заміна. Система Jade дотримується цього підходу.

Jade побудовано на моделі компонентів Fractal, реалізації фреймворку Java, яка дозволяє додавати та видаляти компоненти під час виконання.

Jade використовує поняття домену управління ремонтом. Такий домен складається з кількох вузлів, де кожен вузол представляє сервер разом із компонентами, які виконуються цим сервером. Існує окремий менеджер вузлів, який відповідає за додавання та видалення вузлів із домену. Менеджер вузла може бути відтворений для забезпечення високої доступності.

Кожен вузол оснащений детекторами збоїв, які контролюють працездатність вузла або одного з його компонентів і повідомляють про будь-які збої менеджеру вузла. Як правило, ці детектори враховують виняткові зміни в стані компонента, використання ресурсів і фактичну несправність компонента. Зауважте, що останнє насправді може означати, що машина вийшла з ладу.



При виявленні несправності починається процедура ремонту. Така процедура керується політикою відновлення, яка частково виконується менеджером вузла. Політики вказано чітко та виконуються залежно від виявленої помилки. Наприклад, припустимо, що було виявлено збій вузла. У цьому випадку політика ремонту може передбачати виконання наступних кроків:

1. Припиніть кожне зв’язування між компонентом на несправному вузлі та компонентом на вузлі, який щойно вийшов з ладу.
2. Попросіть менеджера вузлів запустити та додати новий вузол до домену.
3. Налаштуйте новий вузол за допомогою тих самих компонентів, що й на пошкодженому вузлі.
4. Повторно встановіть усі прив’язки, які були раніше розірвані.

У цьому прикладі політика відновлення проста і працюватиме лише тоді, коли не втрачено жодних важливих даних (повідомляється, що пошкоджені компоненти не мають стану).
Підхід, якого дотримується Jade, є прикладом самоконтролю: після виявлення збою автоматично виконується політика відновлення, щоб привести систему в цілому в стан, у якому вона була до збою. Будучи системою на основі компонентів, це автоматичне відновлення вимагає спеціальної підтримки, щоб дозволити додавати та видаляти компоненти під час виконання. Загалом, перетворити застарілі програми на самокеровані системи неможливо.

Розподілені системи можуть бути організовані різними способами. Ми можемо розрізнити архітектуру програмного забезпечення та архітектуру системи. Останній розглядає, де компоненти, які складають розподілену систему, розміщені на різних машинах. Перший більше стурбований логічною організацією програмного забезпечення: як компоненти взаємодіють, як вони можуть бути структуровані, як їх можна зробити незалежними тощо.

Ключовою ідеєю, коли йдеться про архітектуру, є архітектурний стиль. Стиль відображає основний принцип, який дотримується при організації взаємодії між програмними компонентами, що входять до складу розподіленої системи. До важливих стилів належать розшаровування, орієнтація об’єкта, орієнтація події та орієнтація простору даних.

Існує багато різних організацій розподілених систем. Важливим є клас, у якому машини поділяються на клієнти та сервери. Клієнт надсилає запит на сервер, який потім видає результат, який повертається клієнту. Архітектура клієнт-сервер відображає традиційний спосіб модульного програмного забезпечення, за якого модуль викликає функції, доступні в іншому модулі. Розміщуючи різні компоненти на різних машинах, ми отримуємо природний фізичний розподіл функцій у наборі машин.

Архітектури клієнт-сервер часто дуже централізовані.

У децентралізованих архітектурах ми часто бачимо рівну роль, яку відіграють процеси, які становлять розподілену систему, також відому як однорангові системи. У однорангових системах процеси організовані в оверлейну мережу, яка є логічною мережею, у якій кожен процес має локальний список інших однорангових вузлів, з якими він може спілкуватися. Накладну мережу можна структурувати, і в цьому випадку можна розгорнути детерміновані схеми для маршрутизації повідомлень між процесами. У неструктурованих мережах список однорангових вузлів є більш-менш випадковим, що означає, що алгоритми пошуку необхідно розгортати для визначення місцезнаходження даних або інших процесів.

В якості альтернативи були розроблені самокеровані розподілені системи. Ці системи певною мірою поєднують ідеї системної та програмної архітектур. Системи самокерування можуть бути загалом організовані як контури керування зі зворотним зв’язком. Такі цикли містять компонент моніторингу за допомогою вимірювання поведінки розподіленої системи, компонент аналізу, щоб побачити, чи потрібно щось налаштувати, і набір різноманітних інструментів для зміни поведінки. Контури зворотного зв'язку можуть бути інтегровані в розподілені системи в багатьох місцях.

Хмарні обчислення

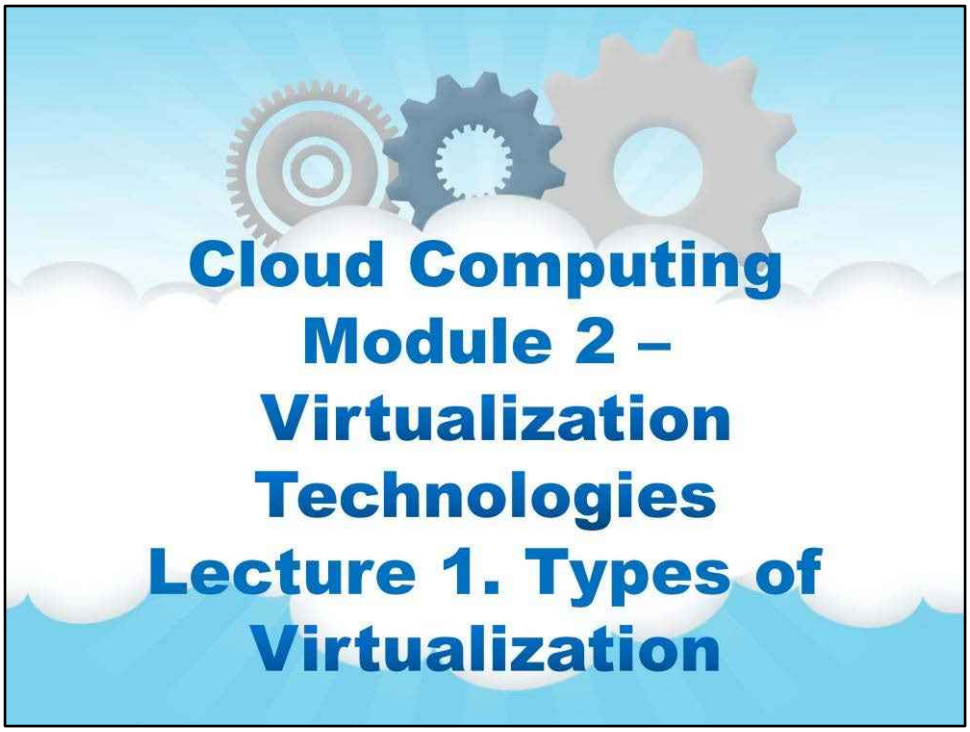
Лекційний посібник

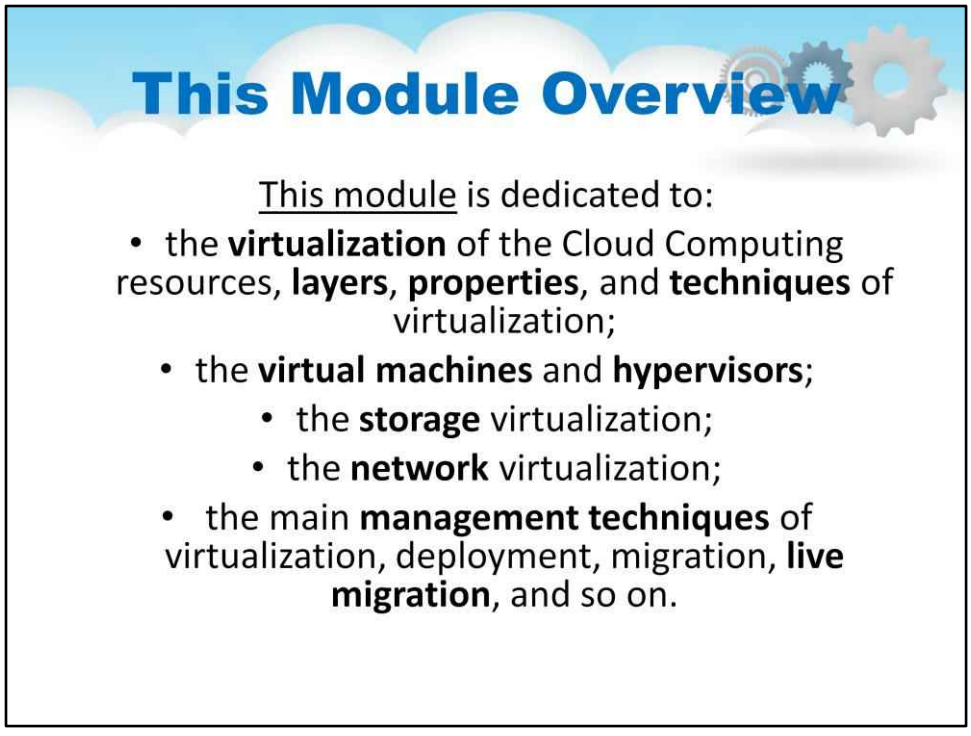
Том 2

Модуль 2

Технології віртуалізації

Зміст

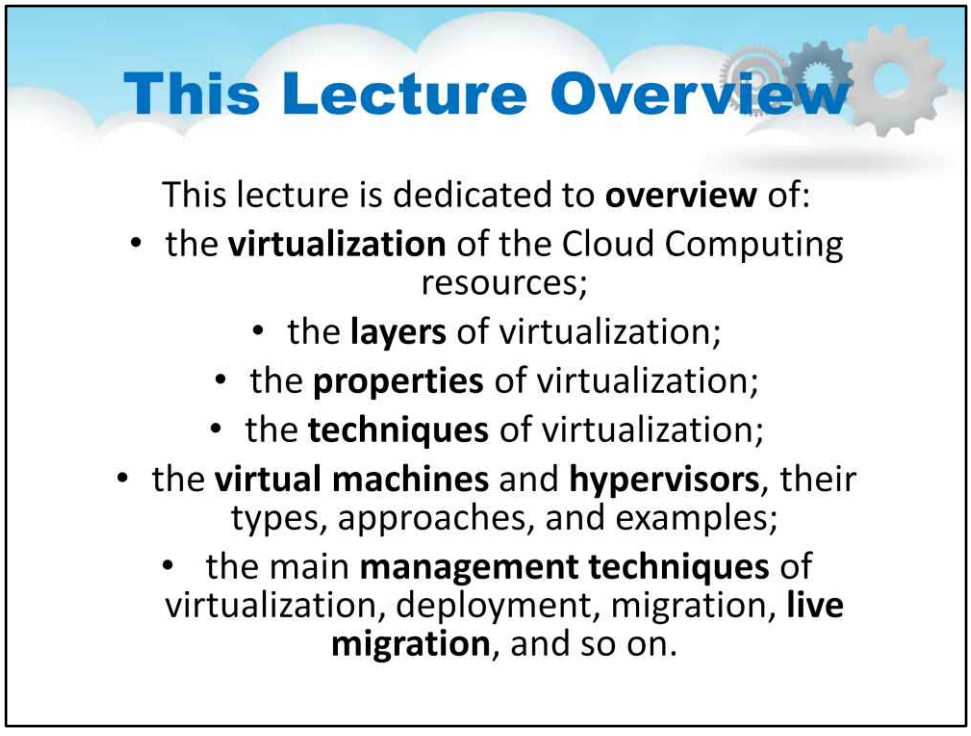

Cloud Computing
Module 2 –
Virtualization
Technologies
Lecture 1. Types of
Virtualization



Модуль 2. Віртуалізація Технології

Цей модуль присвячений:

- **в****іртуалізація**ресурсів хмарних обчислень,**шари**,**властивості**, і**техніки**з віртуалізація;
- **в****іртуальні машини****і****гіпервізори**;
- **в****зберігання**віртуалізація;
- **в****мережі**віртуалізація;
- головний**методи управління**віртуалізації, розгортання, міграції,**жива міграція**, і так далі.



Лекція 1. Види

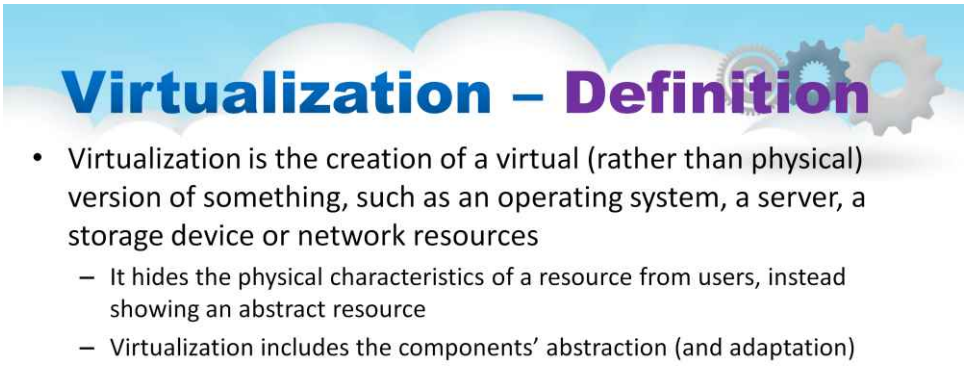
Віртуалізація

Ця лекція присвячена**огляд**з:

- в**віртуалізація**ресурсів хмарних обчислень;
- в**шари**віртуалізації;
- в**властивості**віртуалізації;
- в**техніки**віртуалізації;
- в**віртуальні машини**і**гіпервізори**, їх види, підходи та приклади;
- головний**методи управління**віртуалізації, розгортання, міграції,**жити міграція**, і так далі.

Технології віртуалізації

Огляд



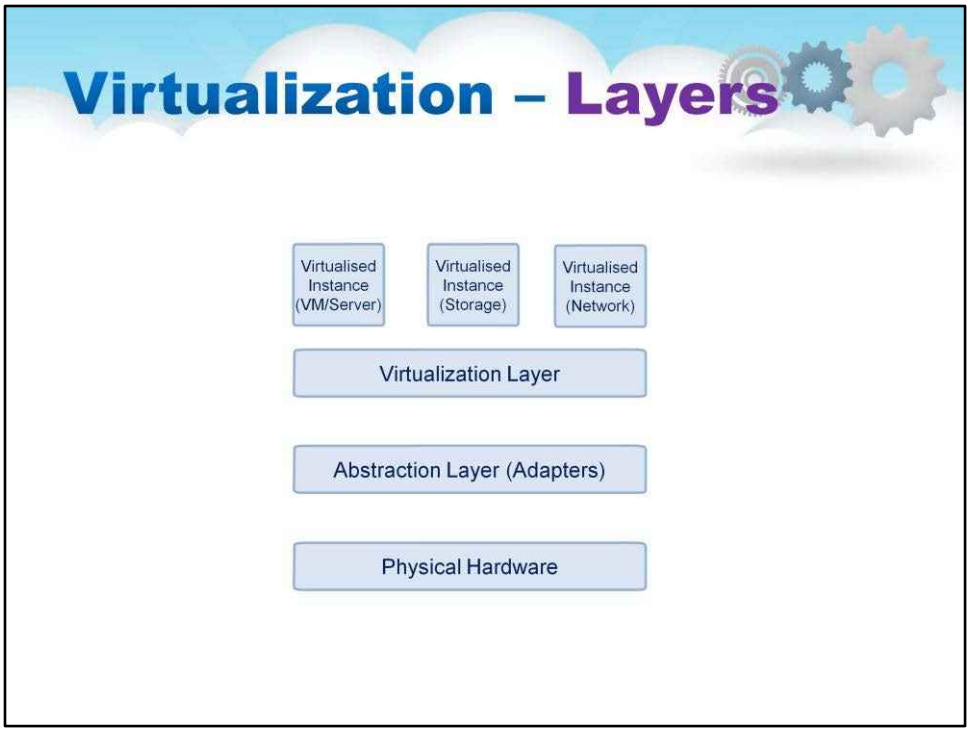
Цей слайд розповідає про «абстрактну» або теоретичну ідею віртуалізації.

Визначення: Віртуалізація — це створення віртуальної (а не фізичної) версії чогось, наприклад операційної системи, сервера, пристрою зберігання даних або мережевих ресурсів.

- Він приховує фізичні характеристики ресурсу від користувачів, натомість показуючи абстрактний ресурс
- Віртуалізація включає абстракцію (і адаптацію) компонентів

Або фізична річ може бути віртуалізована, як «диск», або більш концептуальна можливість, як «VLAN». Слайд говорить про те, як рівні віртуалізації пов'язані один з одним.

Зауважте, що іноді для моделювання певного типу процесора, накопичувача чи мережі потрібні спеціальні апаратні адаптери.



Шари

Загальні рівні віртуалізації:

- Віртуалізовані екземпляри з настроюваними характеристиками
- Рівень віртуалізації, реалізація програмної віртуалізації
- Рівень абстракції, включаючи апаратні адаптери
- Фізичні ресурси: різні види інфраструктурних ресурсів

Віртуалізація є важливою технікою для активації хмарних властивостей і важливих операцій, таких як оперативна міграція віртуальних машин.

Жива міграція віртуальних машин означає, що віртуальна машину можна перенести з однієї фізичної машини на іншу під час виконання з невеликим зниженням продуктивності.

Наведені нижче властивості хмари базуються на віртуалізації.

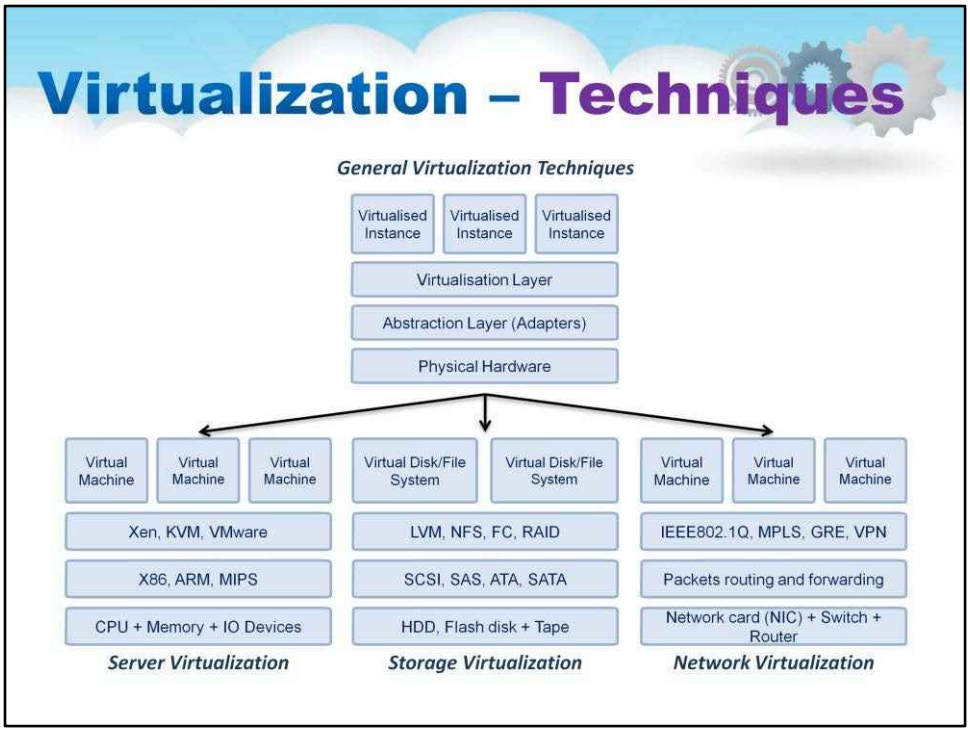
• Масштабованість – система віртуальної машини масштабується автоматично

• Доступність - відмовостійкість до апаратних і програмних збоїв

• Керованість/переносимість - автоматичне перетворення фізичної системи у віртуальну

• Продуктивність – балансування навантаження на рівні динамічної віртуальної машини

• Multi-tenancy - інфраструктура орендарів і ізоляція програм



Техніки

Цей слайд ілюструє методи віртуалізації.

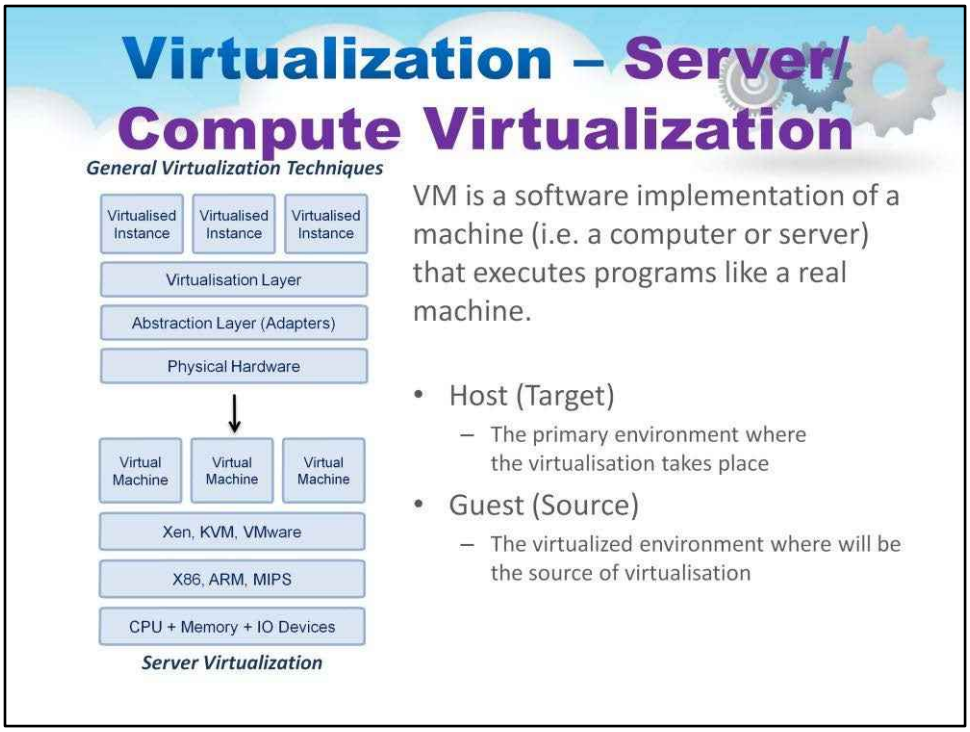
У верхній частині показано загальну техніку віртуалізації.

Нижче можна побачити, як загальна техніка застосовується до кількох проблем віртуалізації.

Для віртуалізації серверів широко використовуються гіпервізори.
Гіпервізори будуть пояснені пізніше.

Для віртуалізації сховищ використовується багато програмних методів, включаючи керування томами, файлові системи та реплікацію.

Для віртуалізації мережі існує багато різних функцій, включаючи агрегацію посилань, VPN, а також брандмауер, комутацію, маршрутизацію та фільтрацію додатків і балансування навантаження мають віртуальні можливості в багатьох сьогоднішніх хмарних реалізаціях.



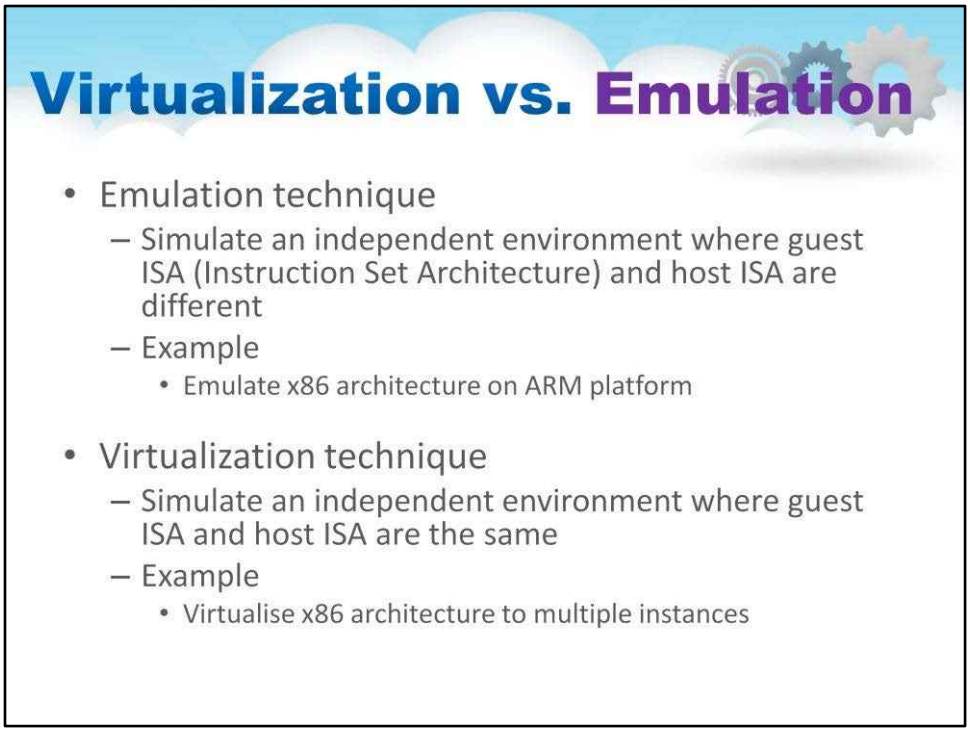
У цьому підручнику ми детальніше розглянемо віртуалізацію серверів або обчислювальних ресурсів. Спочатку ми розглянемо цю ситуацію в загальному вигляді.

Найважливіша концепція полягає в тому, що хост — це конкретна фізична інфраструктура, аж до типу процесора та конкретних пристроїв введення/виведення.

Гість — це операційна система, яка працює на «абстрактному» обладнанні платформа; ця абстрактна платформа може навіть не мати того самого процесора чи архітектури вводу/виводу, що й хост.

Це робота рівня віртуалізації та абстракції, щоб забезпечити цю проміжну функцію.

Це показано на слайді.



Емуляція проти віртуалізації

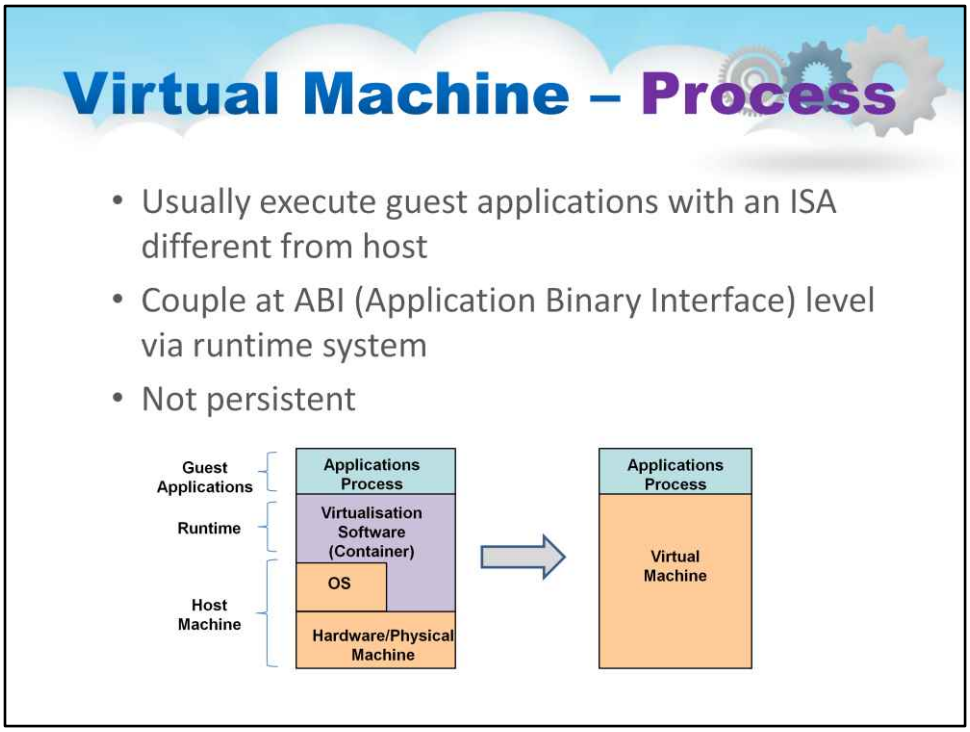
приклад

Емуляція архітектури x86 на платформі ARM

приклад

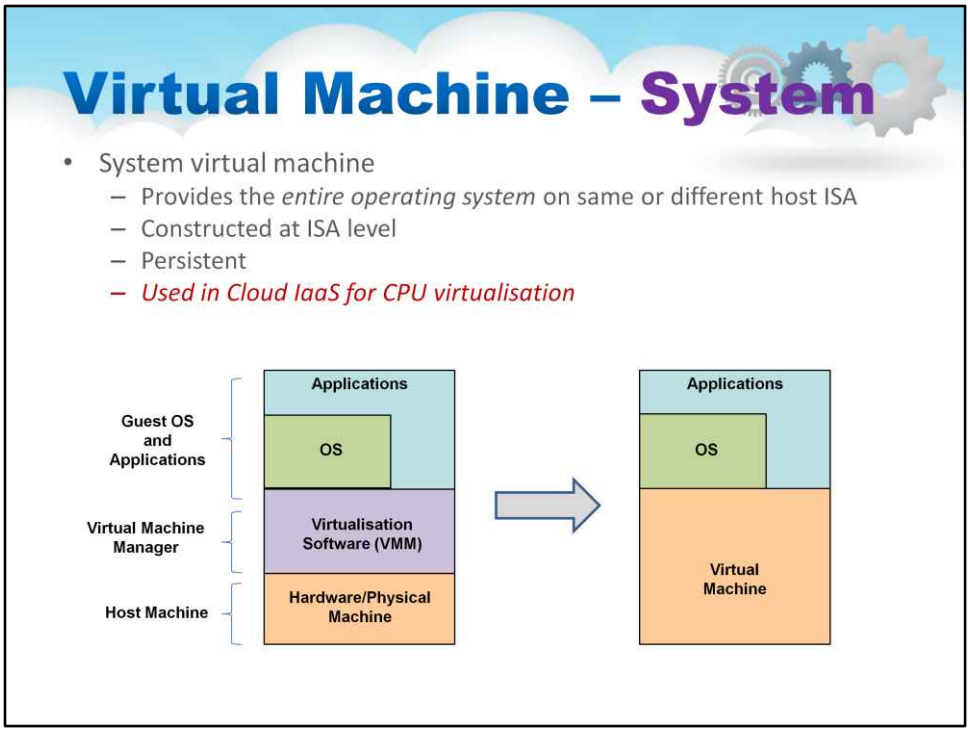
Віртуалізація архітектури x86 для кількох примірників

Віртуальні машини



Одним із підходів до віртуалізації є використання засобу на рівні процесу в операційній системі.Це називається «віртуальна машина процесу», що не те саме, що гіпервізор Техніка (наступний слайд). Це техніка ОС, інтерес до якої останнім часом відроджується.

Віртуальна машина процесу, яку іноді називають віртуальною машиною програми, працює як звичайна програма всередині головної ОС і підтримує один процес. Він створюється, коли цей процес запускається, і знищується, коли він виходить. Його мета полягає в тому, щоб забезпечити незалежне від платформи середовище програмування, яке абстрагує деталі базового обладнання чи операційної системи та дозволяє програмі виконуватися однаково на будь-якій платформі.



Найпоширеніший підхід віртуалізації використовує гіпервізор і називається «Системна віртуальна машина».

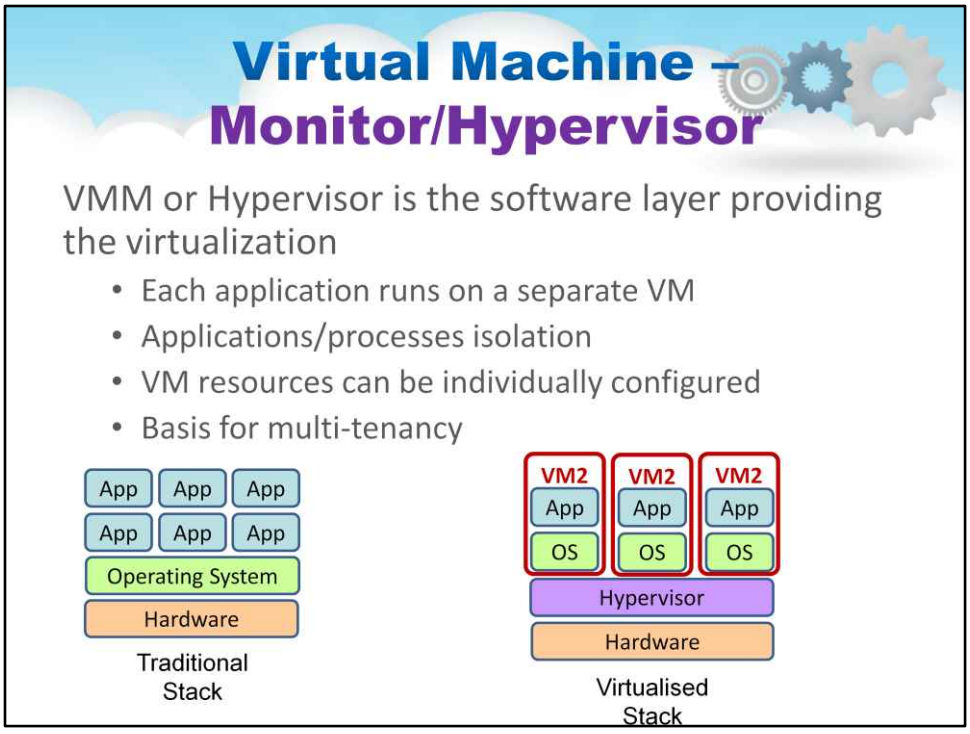
Забезпечує роботу всієї операційної системи на тому самому чи іншому хості.

ISA, створена на рівні ISA

Наполегливий

Використовується в Cloud IaaS для віртуалізації ЦП

Ось чому ми зосередимося на цій формі підходу



Щоб реалізувати віртуальну машину системного рівня, як описано раніше, потрібен монітор віртуальної машини. Це частіше називається гіпервізором.

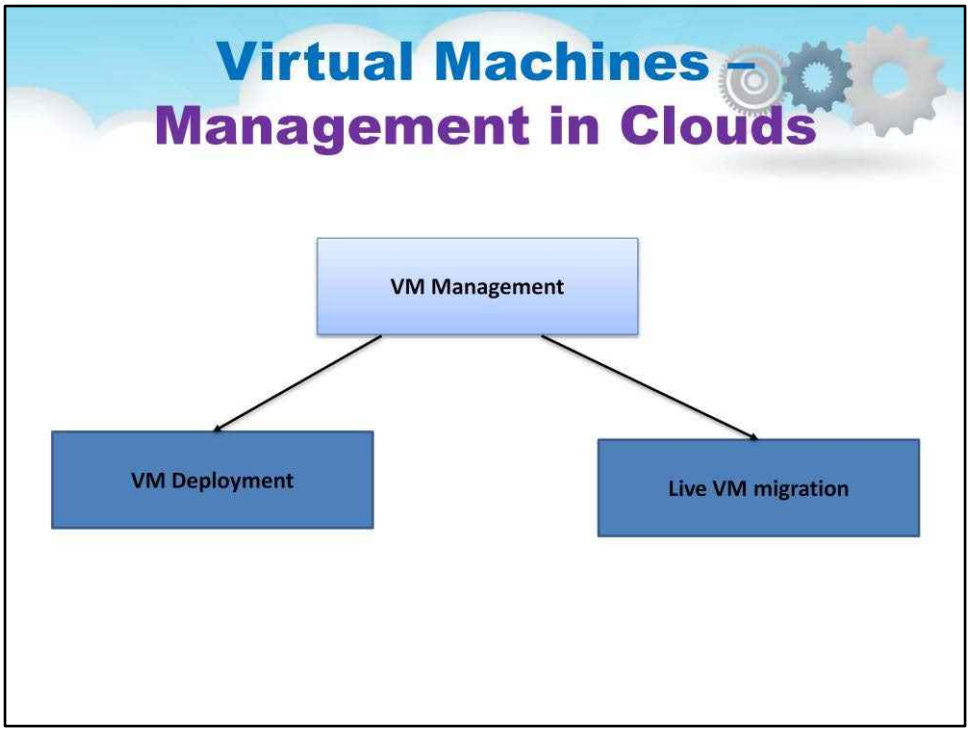
На слайді показано, як гіпервізор вписується в стек операційної системи.

Гіпервізор — це процес, який розділяє операційну систему комп’ютера та додатки від основного фізичного обладнання. Зазвичай це програмне забезпечення, хоча вбудовані гіпервізори можна створювати для таких речей, як мобільні пристрої.

Гіпервізор — це рівень програмного забезпечення, що забезпечує віртуалізацію

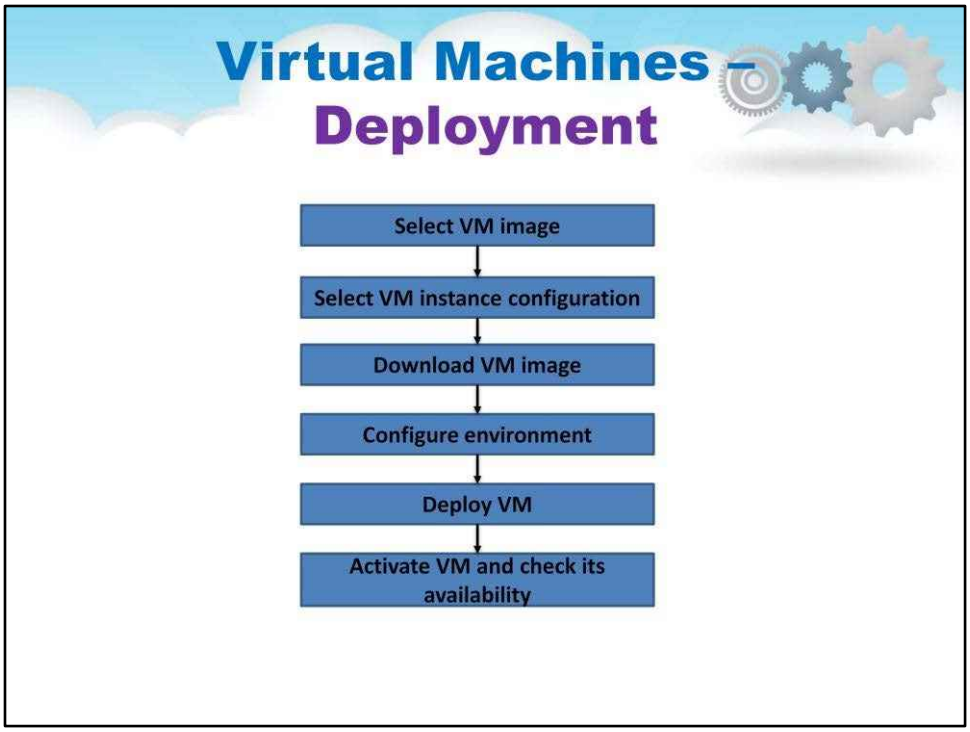
- Кожна програма працює на окремій віртуальній машині
- Ізоляція програм/процесів
- Ресурси віртуальної машини можна налаштувати індивідуально
- Це основа для багатоквартирного користування

Управління в хмарах



Наявність можливостей VMM/гіпервізора, встановлених на всіх машинах усередині кластера, є чудовим початком. Однак хмари насправді немає, але додається блок автоматизації, де розгортання віртуальних машин контролюється VMM у відповідь на програмні вхідні дані. VMM також може переорганізовувати або запускати та зупиняти Віртуальні машини для кращого використання ресурсів. Жива міграція віртуальної машини — це техніка, яку VMM використовує для перевпорядкування.

Загальна автоматизація забезпечується програмним керуванням віртуальних машин, яке а Надає VMM/Hypervisor. Давайте детально розглянемо цю важливу здатність.



Виберіть образ віртуальної машини. Використовуйте попередньо збережений образ віртуальної машини або знайдіть на ринку віртуальних машин, який зазвичай доступний у хмарного постачальника

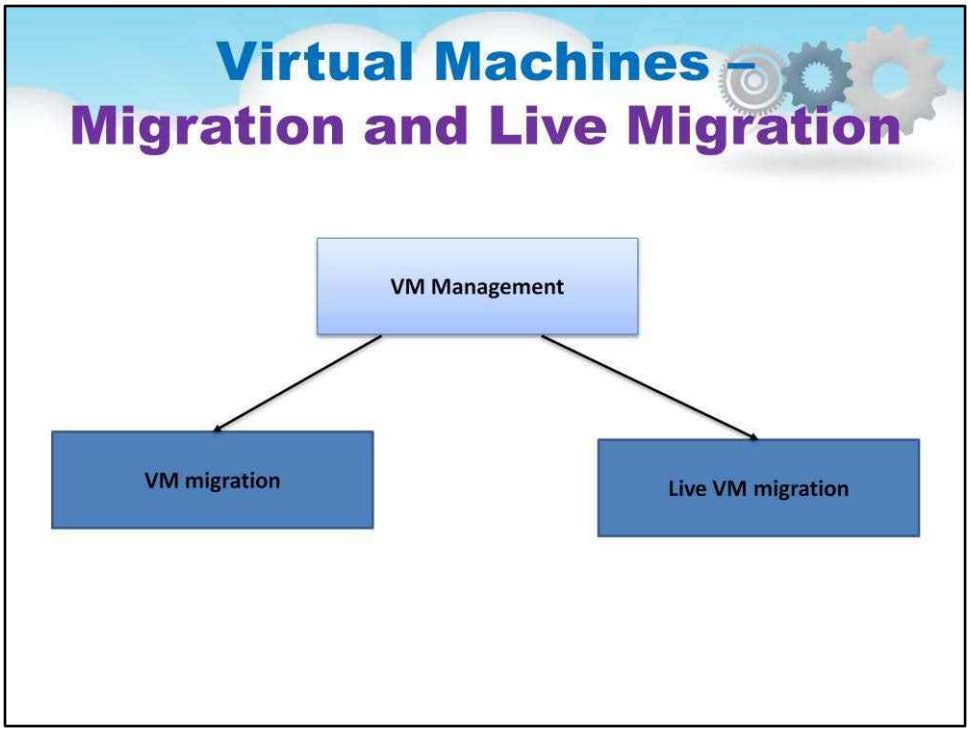
Виберіть конфігурацію екземпляра віртуальної машини – тип ЦП, # ядер, пам’ять, сховище

Завантажте образ віртуальної машини – у вибране розташування сервера/центру обробки даних (наприклад, виберіть зону доступності AWS)

Налаштувати середовище - IP-адреса (загальнодоступна чи приватна); шлюз, DNS, зовнішнє сховище, балансувальник навантаження (опціонально)

Розгорніть віртуальну машину за допомогою веб-інструменту або інтерфейсу командного рядка

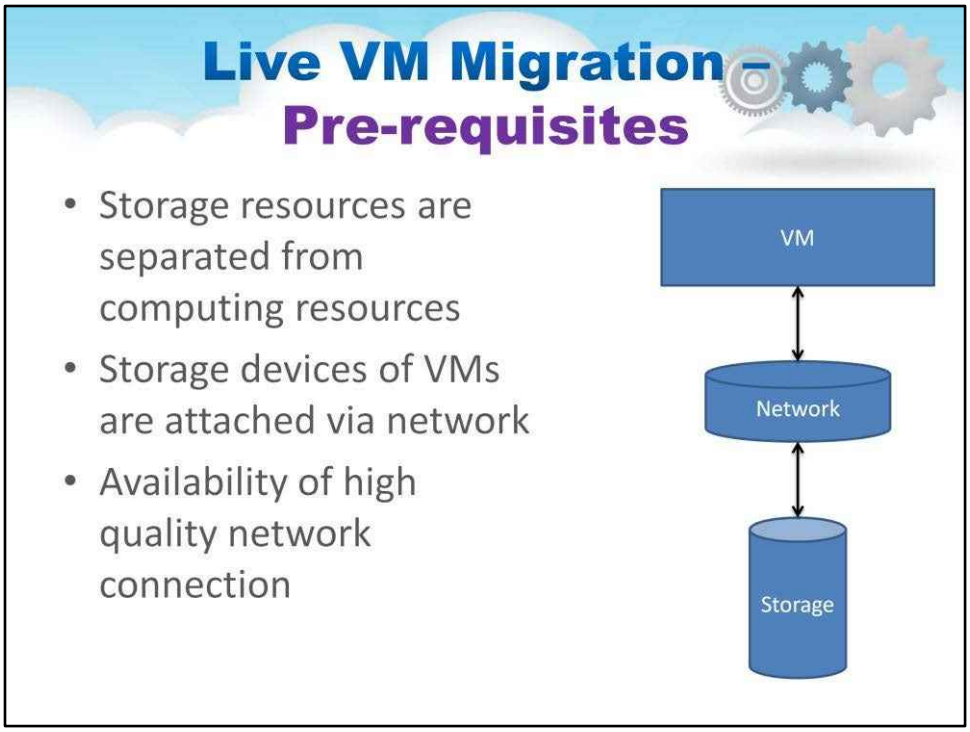
Активуйте віртуальну машину та перевірте її доступність, увійшовши до віртуальної машини та/або в каталозі гіпервізора



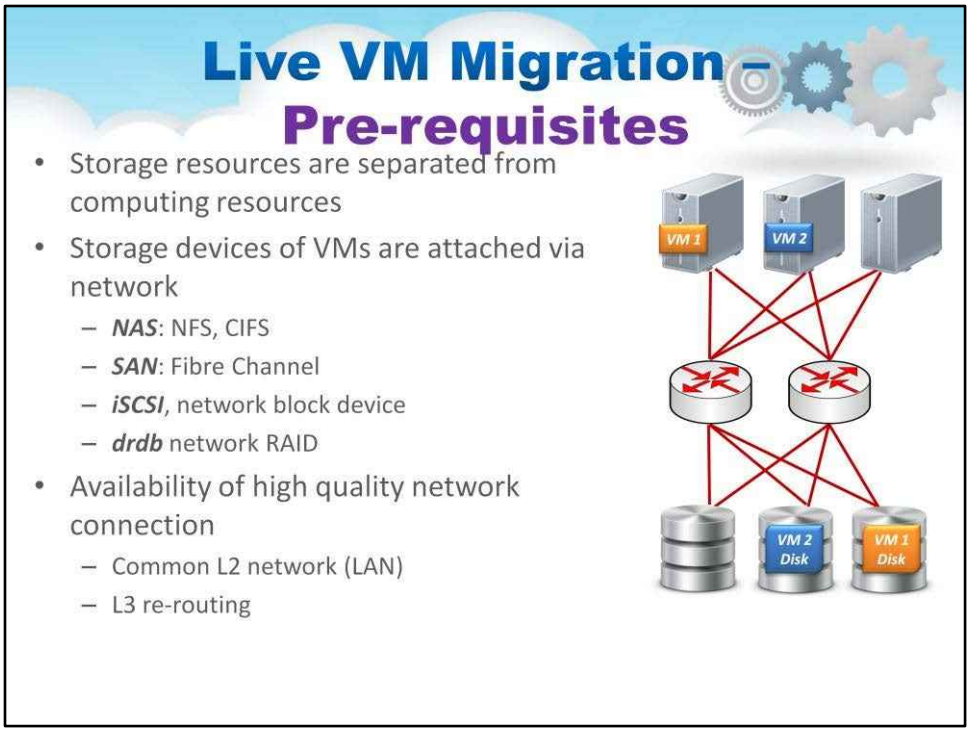
У контексті віртуалізації, де гостьова симуляція всього комп’ютера насправді є лише програмною віртуальною машиною (VM), що працює на головному комп’ютері під гіпервізором,**міграція**це процес, за допомогою якого працююча віртуальна машина переміщується з одного фізичного хоста на інший, з невеликим або без збоїв у роботі.

Жива міграціявідноситься до процесу переміщення запущеної віртуальної машини або програми між різними фізичними машинами без відключення клієнта або програми. Пам’ять, сховище та підключення до мережі віртуальної машини переносяться з початкової гостьової машини на цільову.

Жива міграція ВМ



Коли віртуальна машина запущена, вона підлягає подальшому налаштуванню Cloud OS / VMM. Може бути оптимізація, пов’язана з використанням, яка передбачає розміщення цієї запущеної віртуальної машини на іншому фізичному сервері.



Щоб перемістити віртуальну машину за допомогою Live Migration, ресурси зберігання потрібно відокремити від обчислювальних, щоб їх можна було повторнозіставлено з цільовим розташуванням віртуальної машини.

Пристрої зберігання віртуальних машин підключаються через мережу

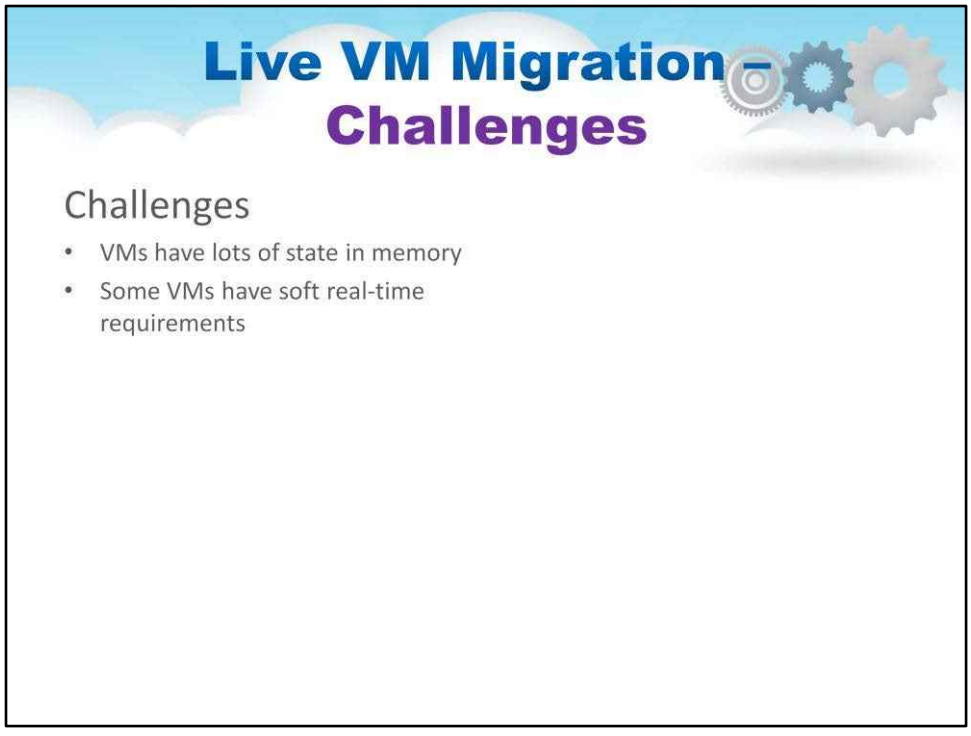
NAS: NFS, CIFS

SAN: Fibre Channel

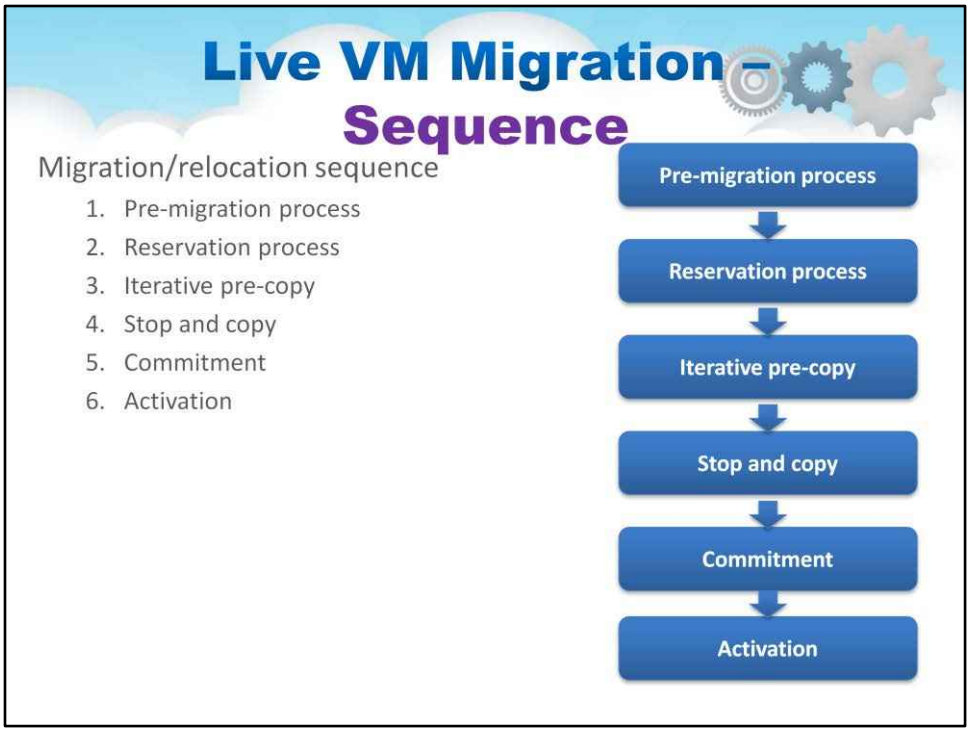
iSCSI, мережевий блоковий пристрій

мережевий RAID drdb

І оскільки ми виконуємо копіювання пам’яті (саме так працює Live Migration), мережа має підтримувати ці швидкості передачі через комутатори та маршрутизатори, якщо це необхідно.



Деякі програми мають багато-багато станів із великим обсягом пам’яті та великою кількістю транзакційних робочих навантажень. Жива міграція може стати складною гонкою між часовими сегментами підсистеми копіювання пам’яті та часовими сегментами виконання програми. Для більшості додатків алгоритми працюють добре, і Live Migration є хітом



Як насправді працює Live Migration?

Це показано на слайді.

Це спеціальна послідовність міграції/переміщення

1. Передміграційний процес

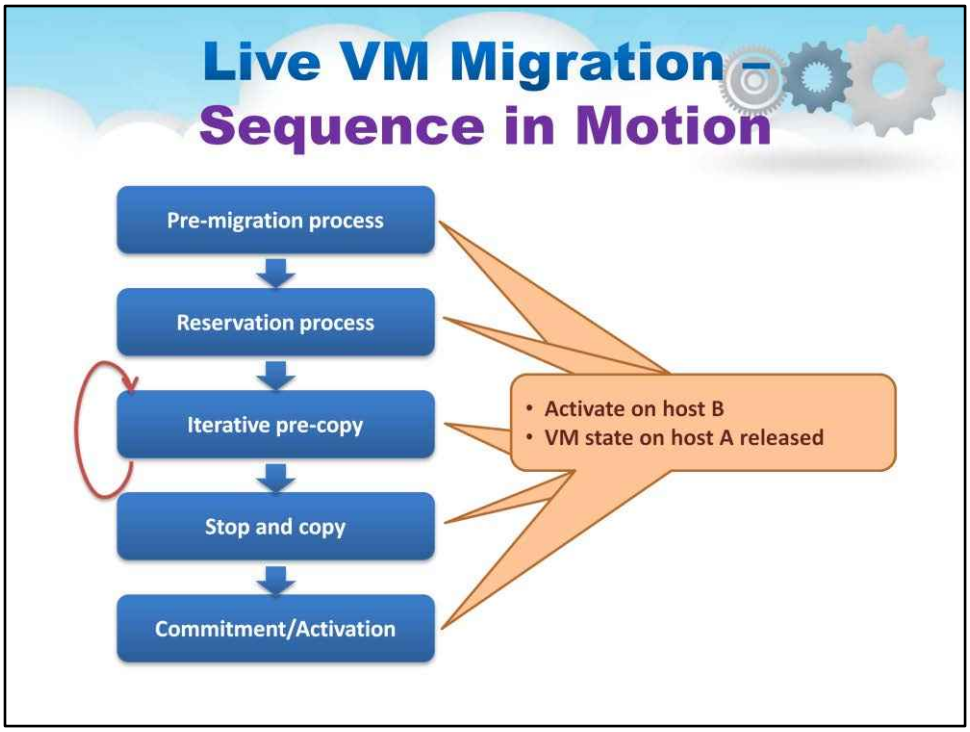
2. Процес резервування

3. Ітераційна попередня копія

4. Зупиніться та скопіюйте

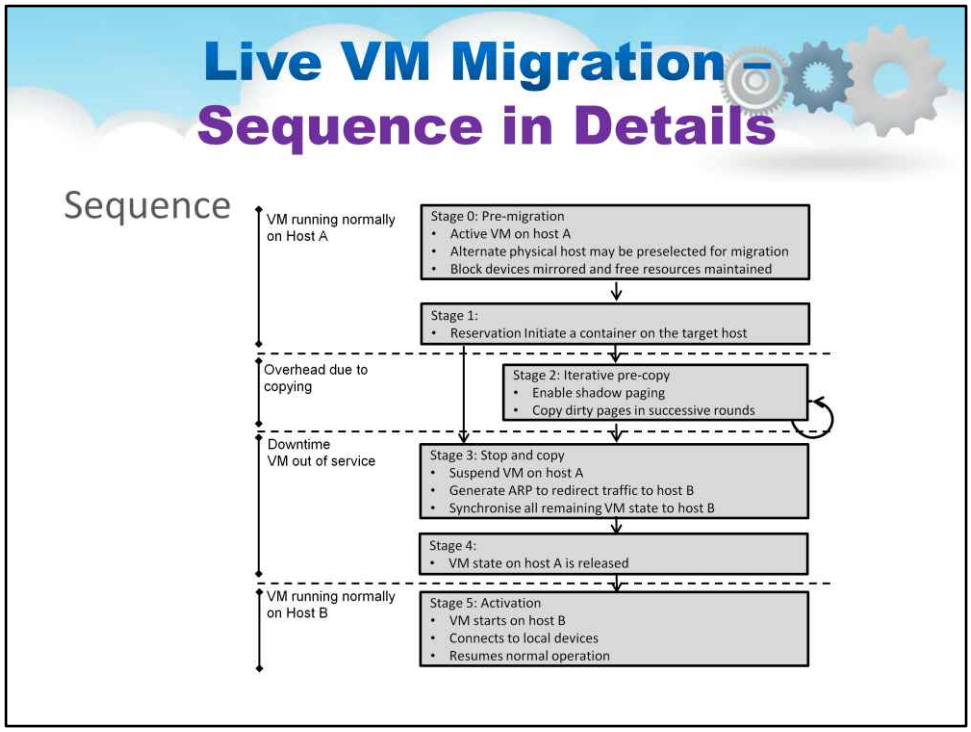
5. Прихильність

6. Активація



Цей слайд анімовано ілюструє весь процес міграції/ переміщення:

1. Передміграційний процес
2. Процес резервування
3. Ітераційна попередня копія
4. Зупиніться та скопіюйте
5. Зобов'язання/Активація



На цьому слайді детальніше розповідається про Live Migration.

Послідовність починається о**Етап 1: перед міграцією**

- Активна віртуальна машина на хості A
- Альтернативний фізичний хост може бути попередньо вибраний для міграції
- Віддзеркалення блокових пристроїв і збереження вільних ресурсів

Далі починається процес резервування**Етап 2: Резервування**

- Резервування Ініціювати контейнер на цільовому хості

Примітка. Протягом цих двох етапів віртуальна машина працює нормально на хості A.

Потім**Етап 3: Ітеративна попередня копія** починається:

- Увімкнути тіньовий пейджінг
- Копіюйте брудні сторінки послідовними раундами

Примітка: накладні витрати з’являються через копіювання.

Після цього**Етап 4: зупинка та копіювання**починається:

- Призупинити віртуальну машину на хості A
- Згенеруйте ARP для перенаправлення трафіку на хост B
- Синхронізувати весь стан віртуальної машини, що залишився, із хостом B

Потім**Етап 5: Зобов'язання**починається:

- Стан віртуальної машини на хості A звільнено

Примітка. Час простою віртуальної машини, який не працює, можна спостерігати на етапах 4 і 5.

Нарешті,**Етап 6: Активація**починається:

- Віртуальна машина запускається на хості B
- Підключається до локальних пристроїв
- Відновлює нормальну роботу

Примітка: зараз віртуальна машина працює нормально на хості B

Підсумок і на винос

- Cloud IaaS представляє всі загальні властивості хмарних обчислень і є найпоширенішим типом хмарних послуг
- Cloud IaaS Architecture включає функції для віртуалізації фізичних ресурсів (обчислення, зберігання, мережа), підтримку розгортання та керування наданими хмарними службами IaaS на вимогу
- Існує безліч хмарних платформ IaaS

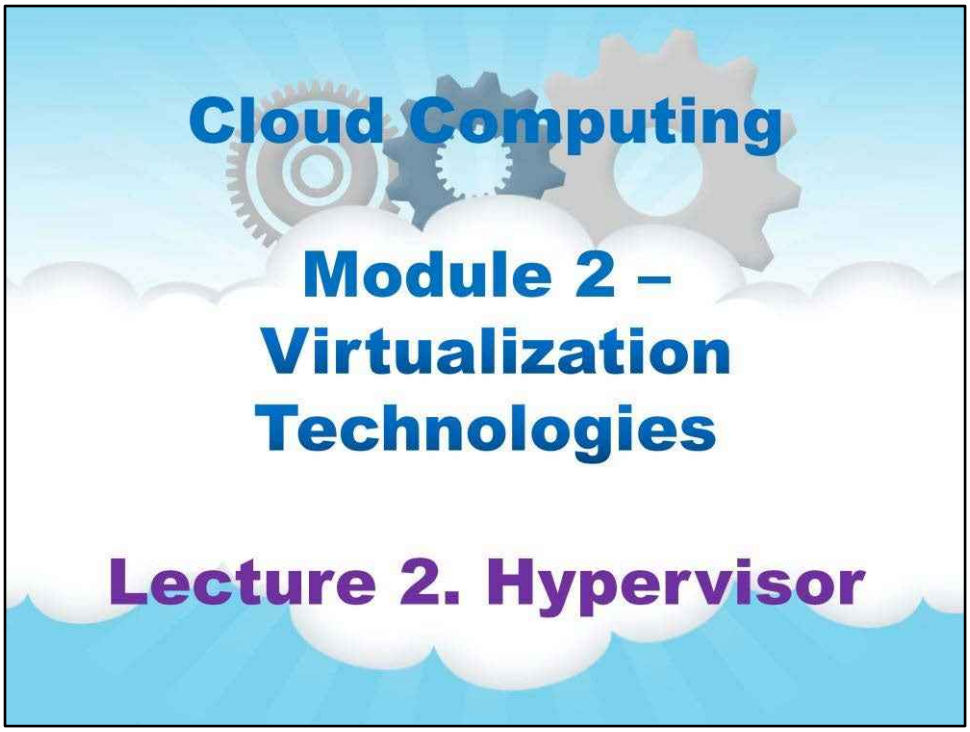
  Постачальники Big Cloud IaaS використовують власні власні платформи
- Існує безліч платформ керування хмарою з відкритим вихідним кодом,

  найбільш популярними є OpenStack, OpenNebula, Eucalyptus, Nimbus.
- Віртуалізація є основною сприятливою технологією для хмари IaaS Забезпечує

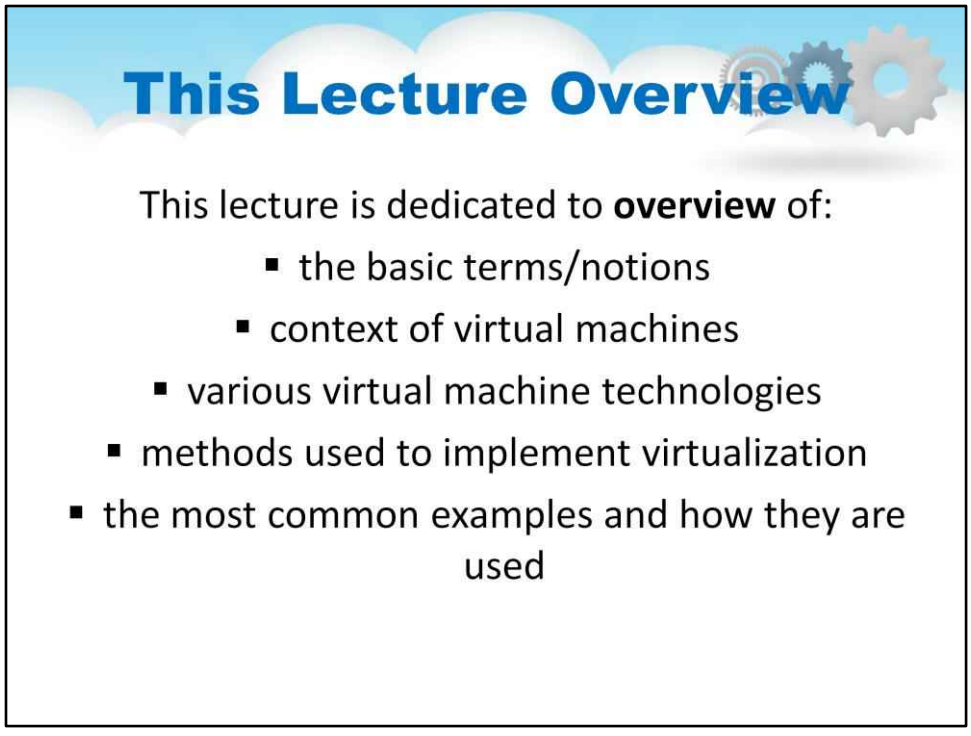
  масштабованість, доступність, керованість і продуктивність хмари
- Міграція віртуальної машини є функцією програмного забезпечення для керування Cloud IaaS

  Жива міграція віртуальної машини виконується за кілька кроків і може мінімізувати час простою служб

У цій лекції ми обговорювали Amazon Web Services Cloud та його основні функції. Історично спочатку поточна хмара AWS представляє всі загальні властивості хмари IaaS. Розуміння основних властивостей AWS Cloud стане гарною основою для розуміння інших хмарних платформ



Це модуль 2під назвою «ВіртуалізаціяТехнології».
Ця лекція 2є прohукерівники.



Лекція 2. Гіпервізор

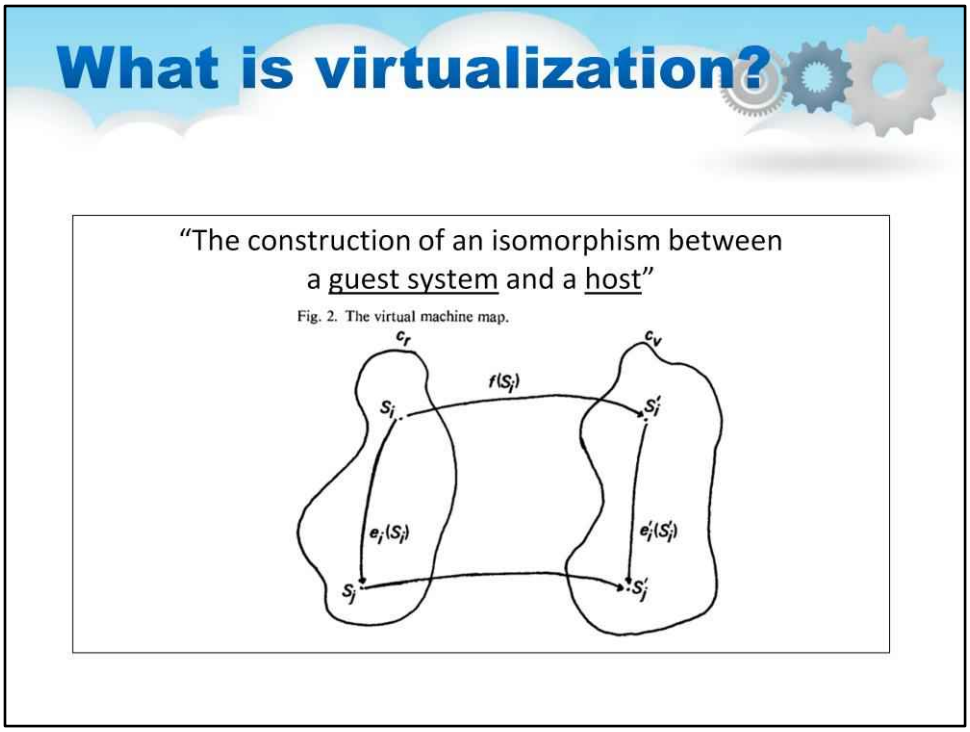
Ця лекція присвячена**огляд**з:

- основні терміни поняття в технологіях віртуалізації (що таке віртуалізація, визначення, компоненти, схема)
- загальний контекст віртуальних машин (що таке віртуалізація, визначення, компоненти, схема)
- різні технології віртуальних машин (основна ідея, мотивація та використання, еволюція)
- методи, що використовуються для реалізації віртуалізації (визначення, за і проти, класифікація)
- найпоширеніші приклади та способи їх використання

Основні поняття

У цьому розділі йдеться про основні поняття термінів у технологіях віртуалізації:

- Що таке віртуалізація
- визначення
- компоненти
- Схема



Віртуалізація — це створення віртуальної версії чогось (апаратного забезпечення, операційної системи, програми, мережі, пам'яті, сховища).

Відповідно до роботи Попека та Голдберга (у 1974 році), віртуалізація – це «побудова ізоморфізму між гостьовою системою та хостом».

Див. подробиці в Popek, Gerald J. і Robert P. Goldberg. «Формальні вимоги до віртуалізовані архітектури третього покоління». Комунікації ACM 17.7 (1974): 412-421.

What is virtualization – 2

- A way to run multiple operating systems and applications on the same hardware (virtual machines)
- Only virtual machine manager (a.k.a. hypervisor) has full system control
- Virtual machines completely isolated from each other (or so we hope)

Example – Virtual Disk

- Partition a single hard disk to multiple virtual disks
- Implement virtual disk by file
- Map between virtual disk and real disk contents
- Virtual disk write/read mapped to file write/read in host system

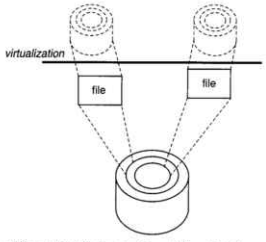


Figure 1.3 Implementing Virtual Disks. *Virtualization provides a different interface and/or resources at the same level of abstraction.*

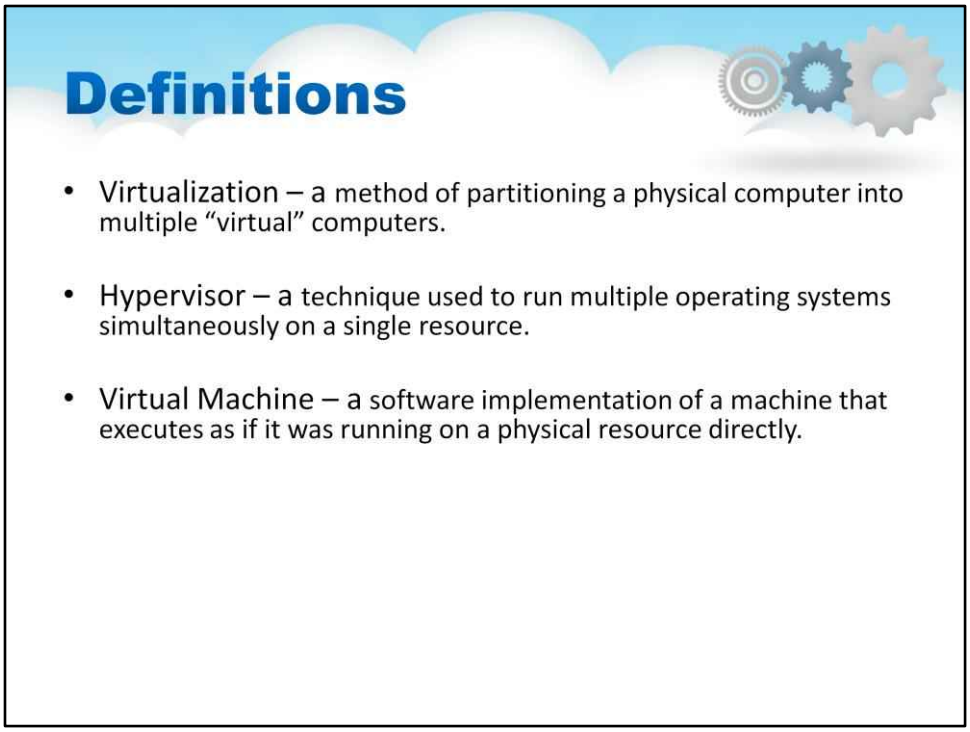
Розбийте один жорсткий диск на декілька віртуальних дисків

Віртуальний диск має віртуальні доріжки та сектори

Реалізація віртуального диска за файлом

Відображення між віртуальним диском і реальним вмістом диска

Запис/читання віртуального диска зіставляється із записом/читанням файлів у хост-системі



Що таке віртуалізація?

Це метод поділу фізичного комп’ютера на кілька «віртуальних» комп’ютерів, кожен з яких працює незалежно так, ніби він працює безпосередньо на апаратному забезпеченні.

Що таке гіпервізор?

Це техніка, яка використовується для одночасного запуску кількох операційних систем на одному ресурсі.

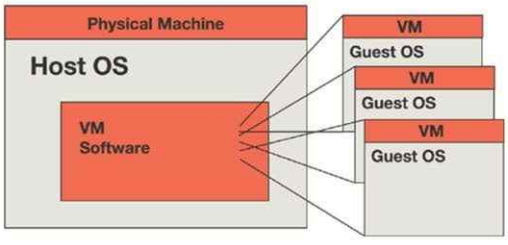
Також називається монітор віртуальної машини (VMM).

Що таке віртуальна машина?

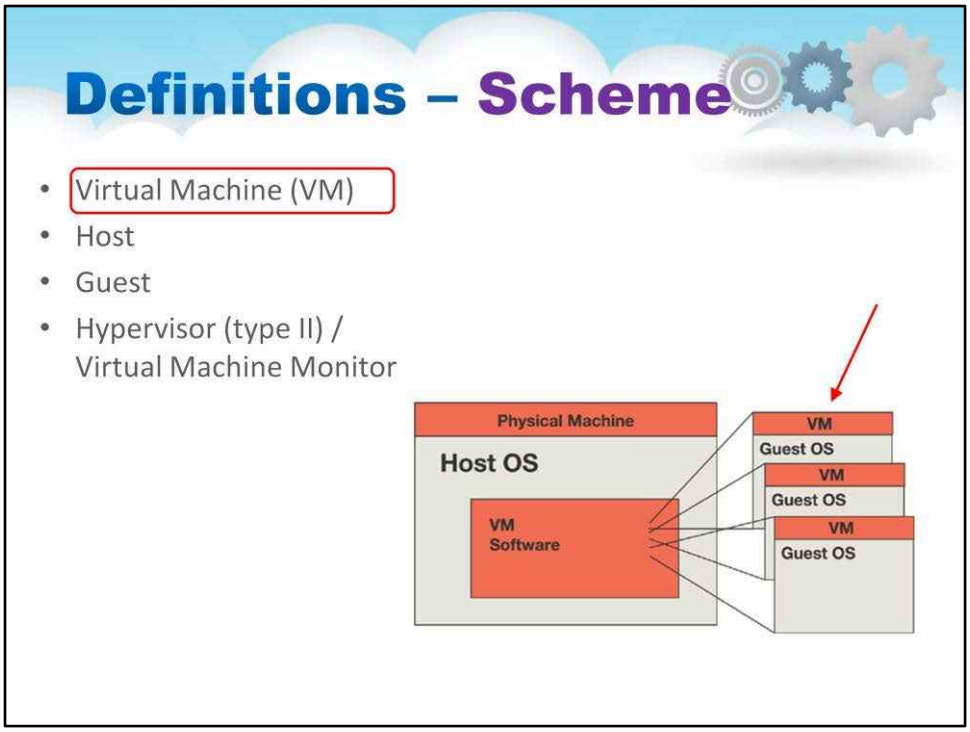
Це програмна реалізація машини, яка виконується так, ніби вона працює безпосередньо на фізичному ресурсі.

Definitions – Scheme

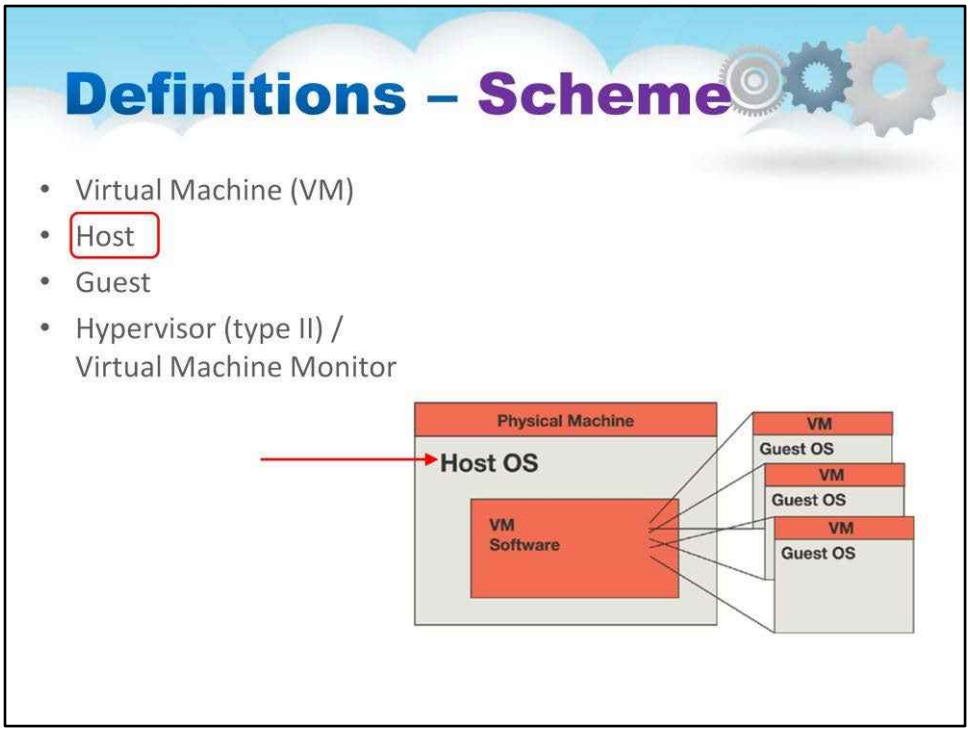
- Virtual Machine (VM)
- Host
- Guest
- Hypervisor (type II) /
  Virtual Machine Monitor



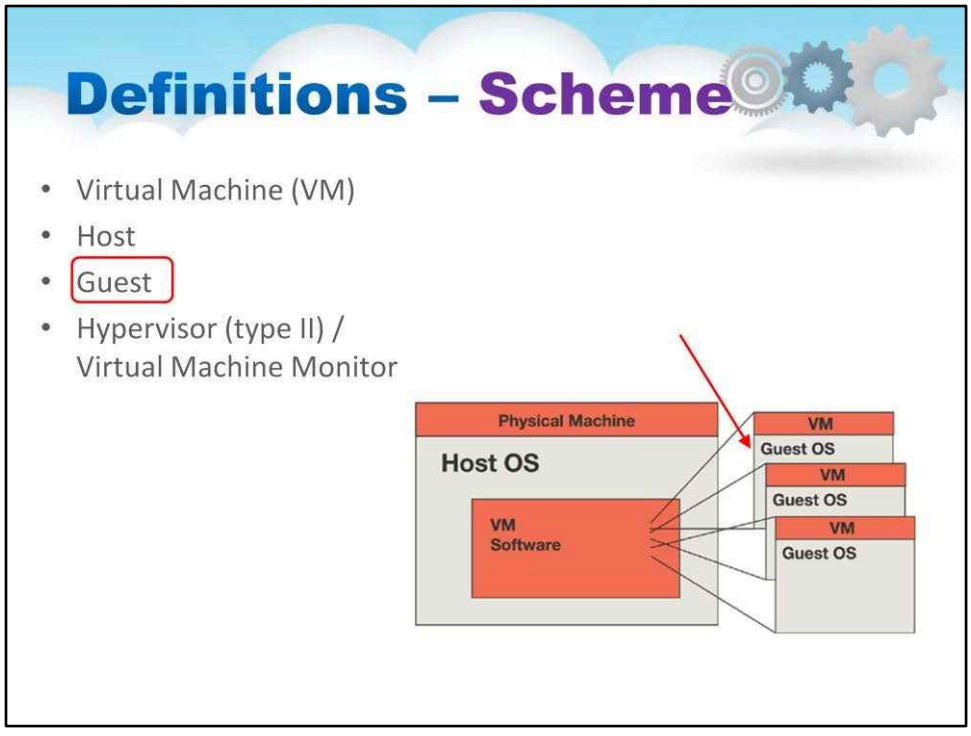
Розглянемо їх за загальною схемою...



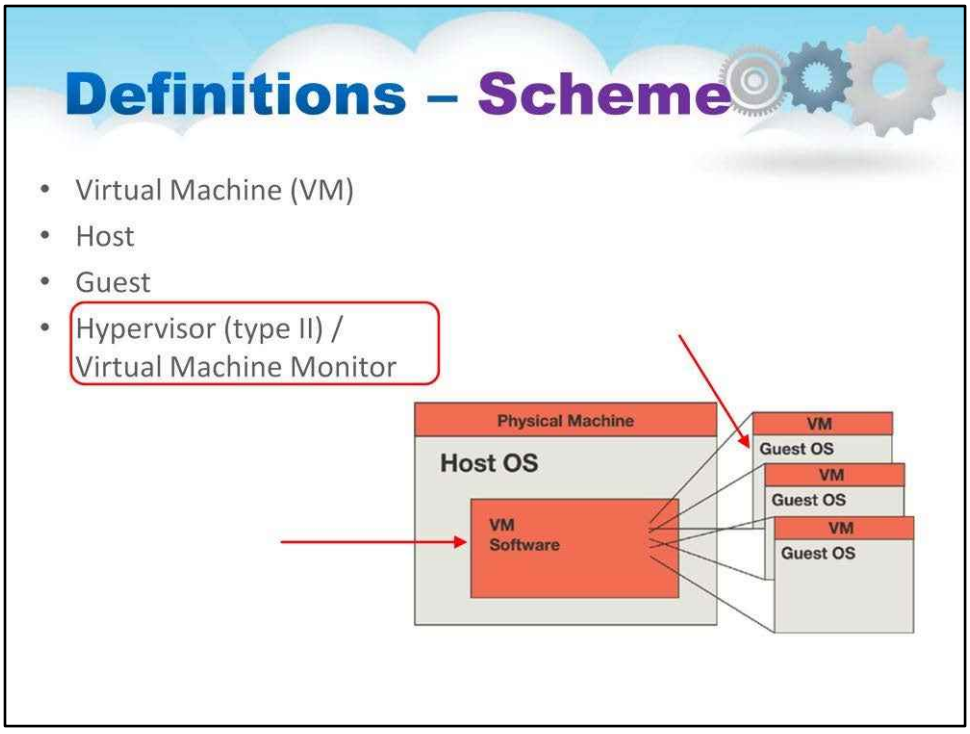
Віртуальна машина – програмна реалізація машини, яка виконується так, ніби вона працює безпосередньо на фізичному ресурсі.



Хост–базова апаратна система

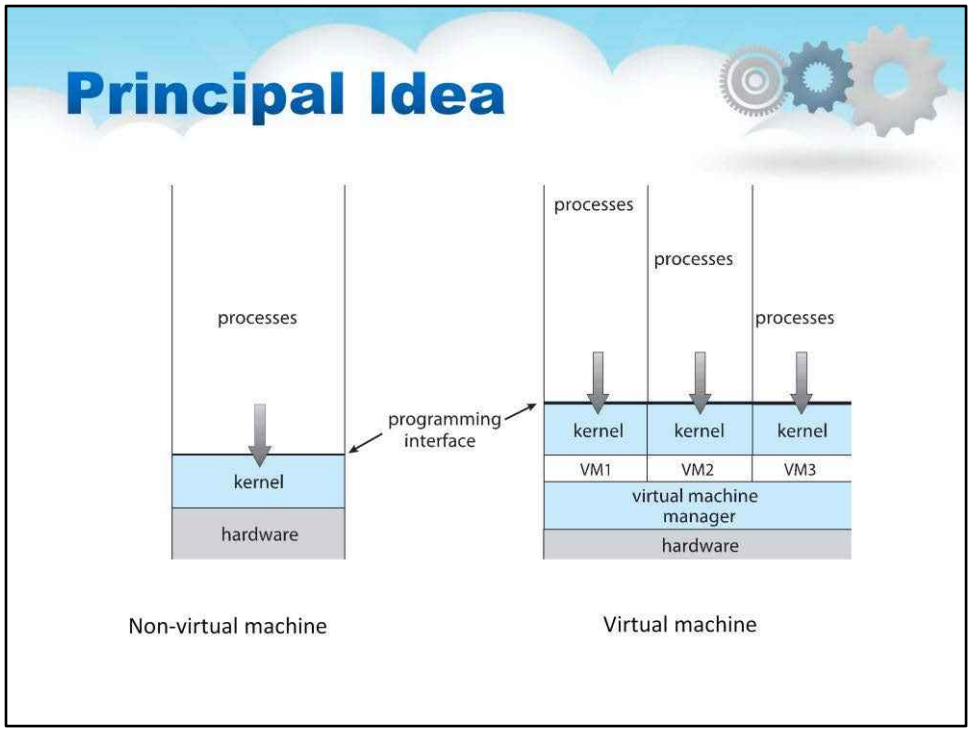


Гість–процес, забезпечений віртуальною копією хоста (зазвичай операційної системи).



Менеджер/монітор віртуальної машини(**VMM**) або**гіпервізор**–створює і запускає

віртуальні машини, надаючи інтерфейс, який єоднаковіна хост (окрім випадку паравіртуалізації)



Нарешті, фундаментальна ідея полягає в тому, щоб абстрагувати апаратне забезпечення одного комп'ютера в кілька різних середовищ виконання

Подібно до багаторівневого підходу

Але рівень створює віртуальну систему (**віртуальна машина**, або**В.М**), на яких операційні системи або програми можуть працювати

Кілька компонентів

Хост–базова апаратна система

Менеджер віртуальної машини(**VMM**) або**гіпервізор**–створює та запускає віртуальні машини, надаючи інтерфейс, який єоднаковідо господаря

(За винятком паравіртуалізації) **Гість**–процес забезпечений віртуальною копією хосту

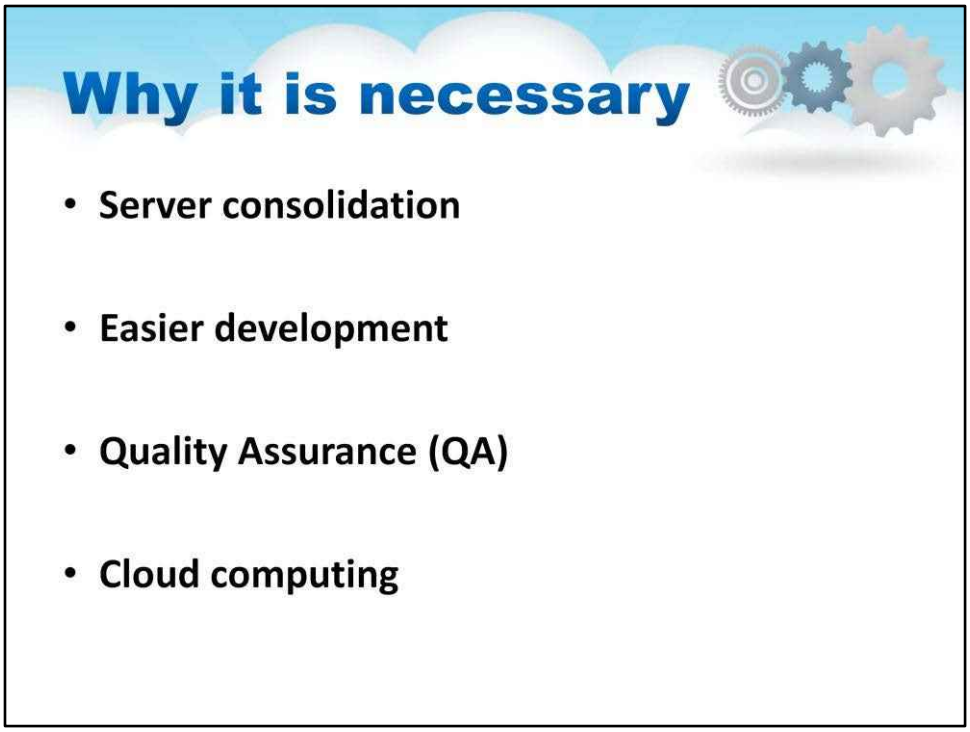
Зазвичай це операційна система

Одна фізична машина може запускати декілька операційних систем одночасно, кожна на своїй віртуальній машині.

Контекст

У цьому розділі йдеться про контекст технологій віртуалізації:

- Основна ідея
- Мотивація та використання
- Еволюція



Консолідація серверів

Запустити а**веб-сервер**і а**поштовий сервер**на**той же фізичний сервер**

Легший розвиток

Розвивати критичність**компоненти операційної системи**(файлова система, драйвер диска) без впливу**стабільність комп'ютера**

Забезпечення якості (QA) як спосіб запобігання помилкам і дефектам у вироблених продуктах і уникнення проблем під час надання рішень або послуг:

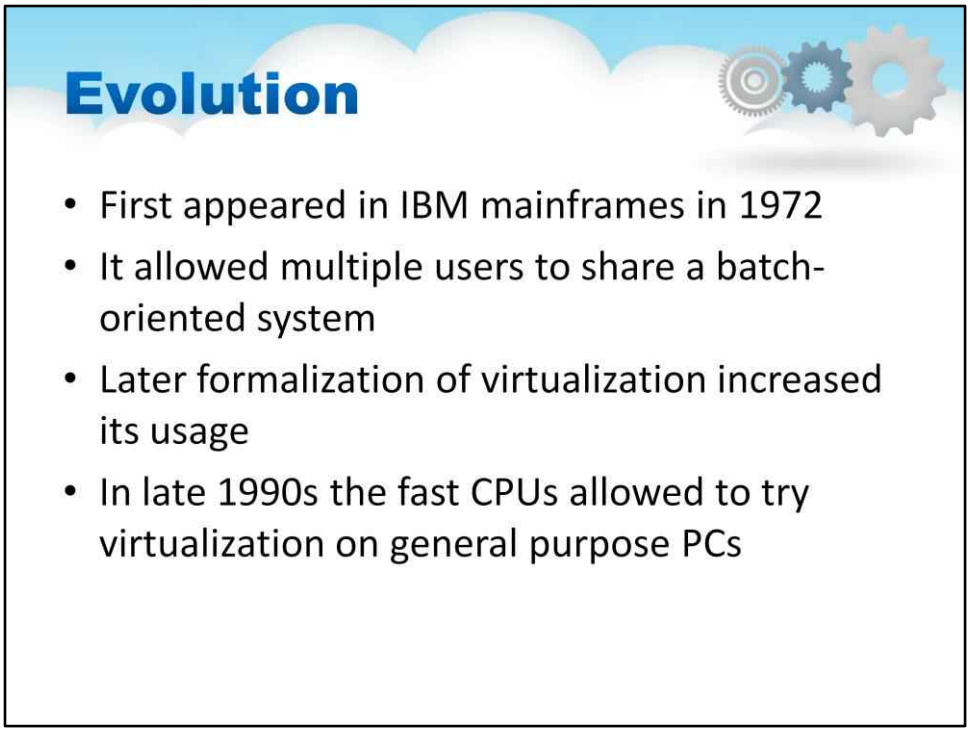
Тестування мережевого продукту (наприклад, брандмауера) може вимагати**десятки комп'ютерів**

Спробуйте ретельно перевірити продукт на кожній попереднійвипустити віху... і отримати прямим обличчям, коли ваш начальник показує вам**рахунок за світло**

Хмарні обчислення

Це вже не модне слово, а реальність

Треба продавати обчислювальну потужність і накопичувачі



Вперше з'явився в мейнфреймах IBM у 1972 році

Дозволяється кільком користувачам спільно використовувати пакетно-орієнтовану систему

Формальне визначення віртуалізації допомогло вийти за межі IBM

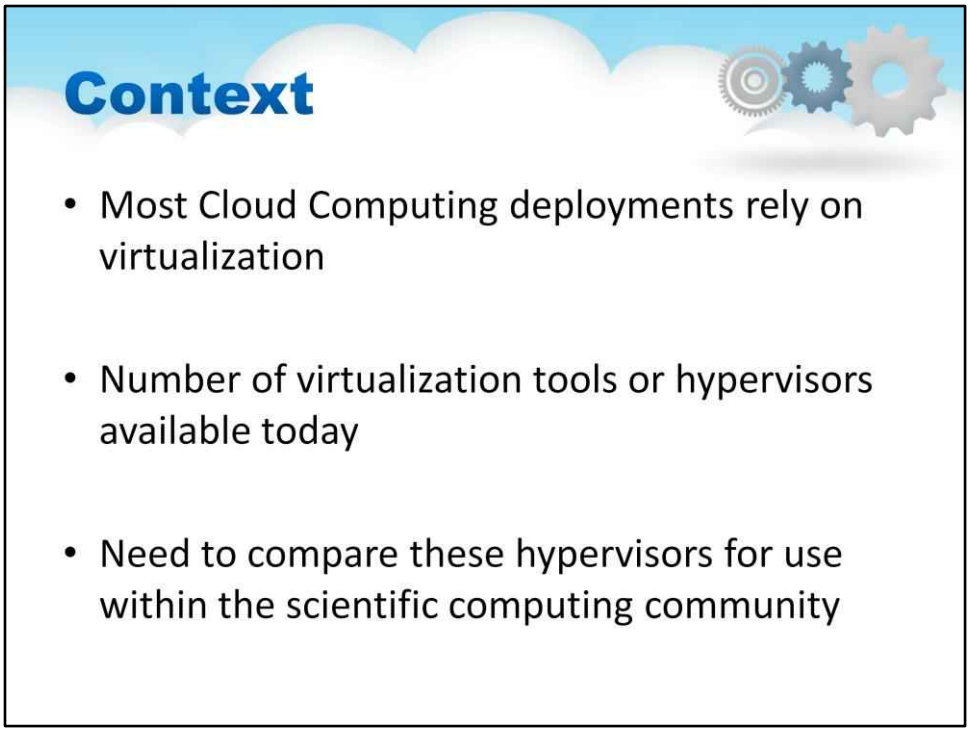
1. VMM забезпечує середовище для програм, яке, по суті, ідентичне оригінальній машині

2.Програми, що працюють у цьому середовищі, показують лише незначне зниження продуктивності

3. VMM повністю контролює системні ресурси

Наприкінці 1990-х процесори були досить швидкими, щоб дослідники спробували віртуалізувати на ПК загального призначення

Xenі**VMware**створені технології, які все ще використовуються сьогодні. Віртуалізація поширилася на багато ОС, ЦП, VMM



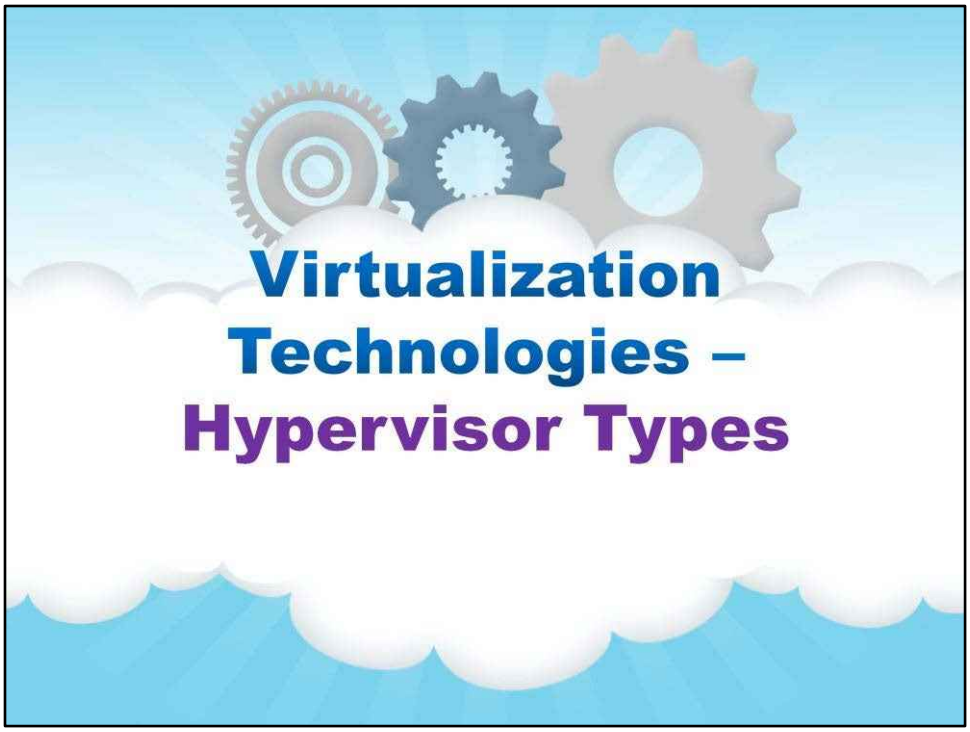
Більшість розгортань хмарних обчислень покладаються на віртуалізацію.

Amazon EC2, GoGrid, Azure, RackspaceХмара…
Nimbus, Eucalyptus, OpenNebula, OpenStack…

Кількість інструментів віртуалізації або гіпервізорів, доступних сьогодні.

Xen, KVM, VMWare, Virtualbox, Hyper-V …

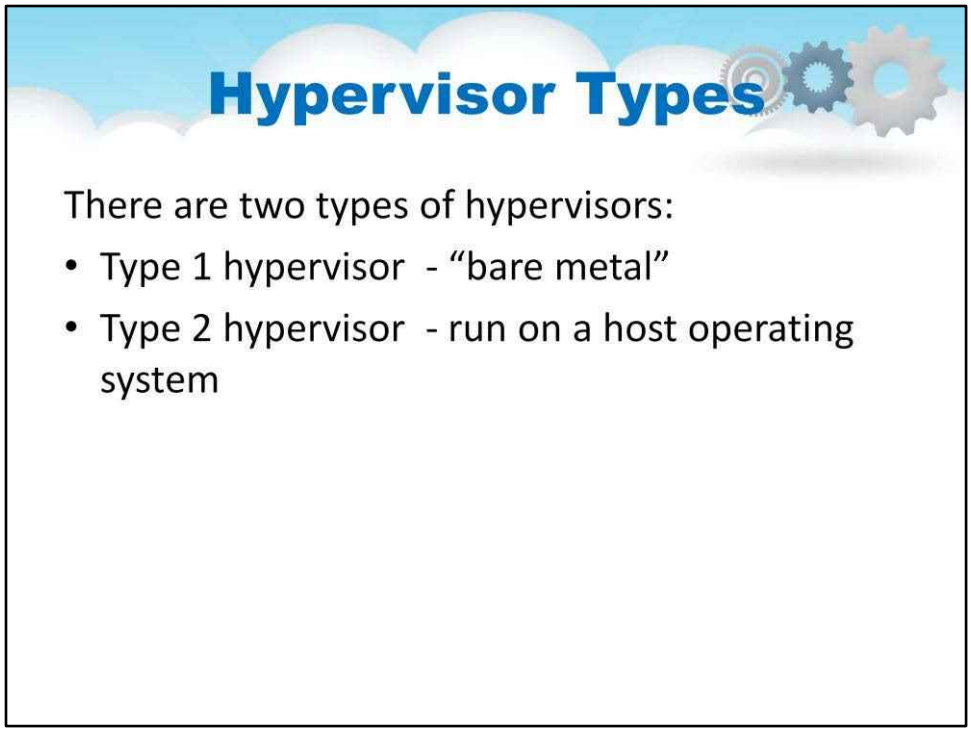
Необхідно порівняти ці гіпервізори для використання в науковому обчислювальному співтоваристві.



Типи гіпервізора

Цей розділ про**Типи гіпервізора**:

- визначення
- За і Проти
- Класифікація



У 1973 році Роберт Голдберг класифікував гіпервізори відповідно до їх близькості до апаратних інструкцій. Голдберг «Тип1”гіпервізор був визначений як такий, що один раз переводив фізичні ресурси у віртуальні. Голдберг «Тип2”гіпервізор двічі здійснив переклад ресурсу.

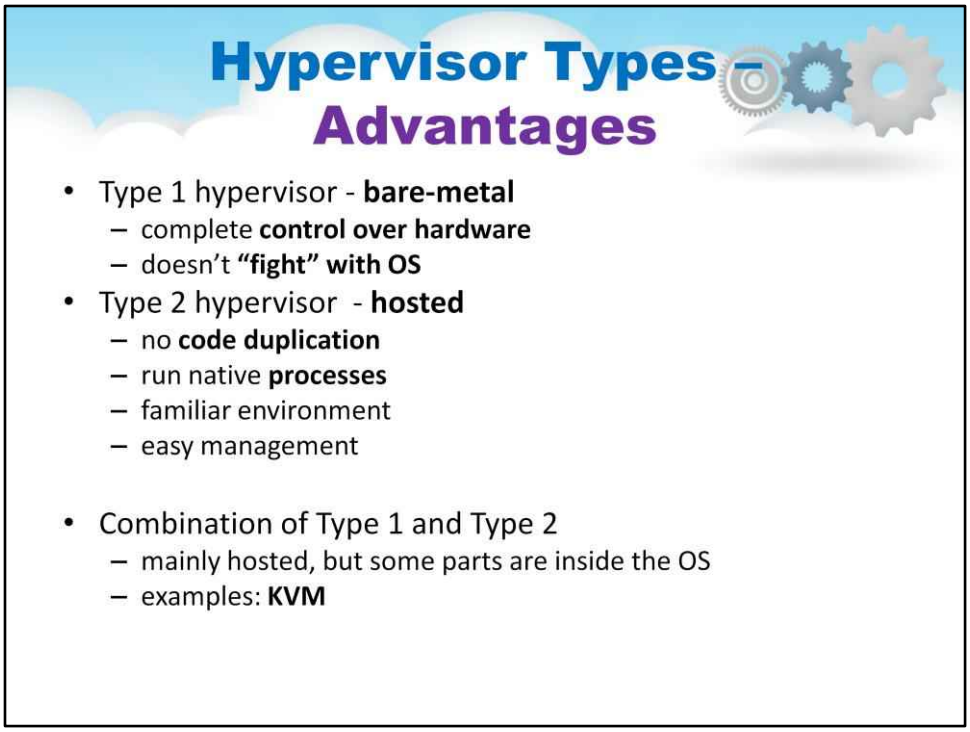
Нещодавно ці визначення були розширені до типу «голий метал».1 гіпервізор (працює безпосередньо на апаратному забезпеченні) проти «розміщеного»Гіпервізори типу 2 (працюють в операційній системі).

Пізніше почалася дискусія: чи є середовище традиційної операційної системи (ОС) частиною коду керування ресурсами гіпервізора, чи навіть традиційна ОС видима або прихована від адміністратора.

Нарешті, компроміс полягає в тому, що існує два типи гіпервізорів:
Гіпервізор типу 1: гіпервізори працюють безпосередньо на апаратному забезпеченні системи –Вбудований «голий метал». гіпервізор,
Гіпервізор типу 2: гіпервізори працюють на головній операційній системі, яка надає послуги віртуалізації, такі як підтримка пристроїв введення-виведення та керування пам’яттю.

Більшість гіпервізорів засновані на повна віртуалізація означає, що вони повністю емулюють усі апаратні пристрої для віртуальних машин. Гостьові операційні системи не вимагають жодних змін і поводитися так, ніби кожен з них має ексклюзивний доступ до всієї системи.

Повна віртуалізація часто має недоліки в продуктивності, оскільки зазвичай повна емуляція вимагає більше ресурсів обробки (і більше витрат) від гіпервізора. Xen базується на паравіртуалізація це вимагає модифікації гостьових операційних систем для підтримки операційного середовища Xen. Однак простір користувача програми та бібліотеки не вимагають модифікація.



Гіпервізор типу 1 -**голий метал:**

Він повний**контроль апаратних засобів** Не треба «воювати» з експлуатацією**система**

Гіпервізор типу 2 -**розміщено:**

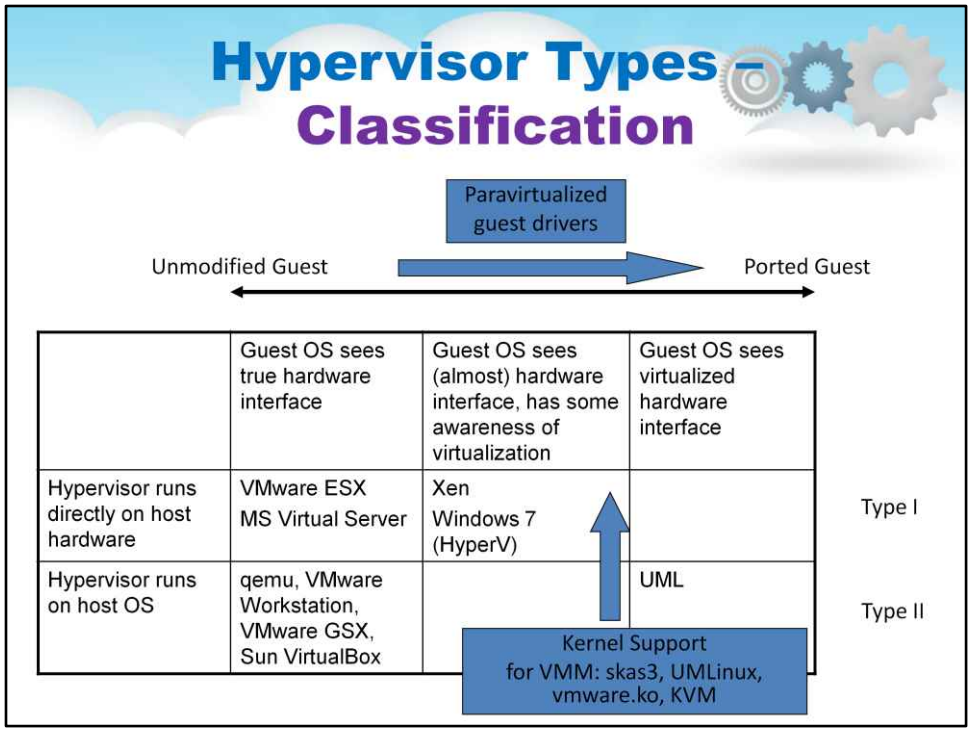
Уникайте**дублювання коду**: код а не потрібен**планувальник процесів**,**управління пам'яттю**система – в**ОС вже робить**що може запускати нативний**процеси поруч** віртуальні машини

Знайоме середовище –**скільки ЦП**і**пам'ять**приймає VM? використання **зверху**! Наскільки великий **віртуальний диск**?**ls–л** Просте керування – зупинити віртуальну машину? Звичайно, просто вбийте його!

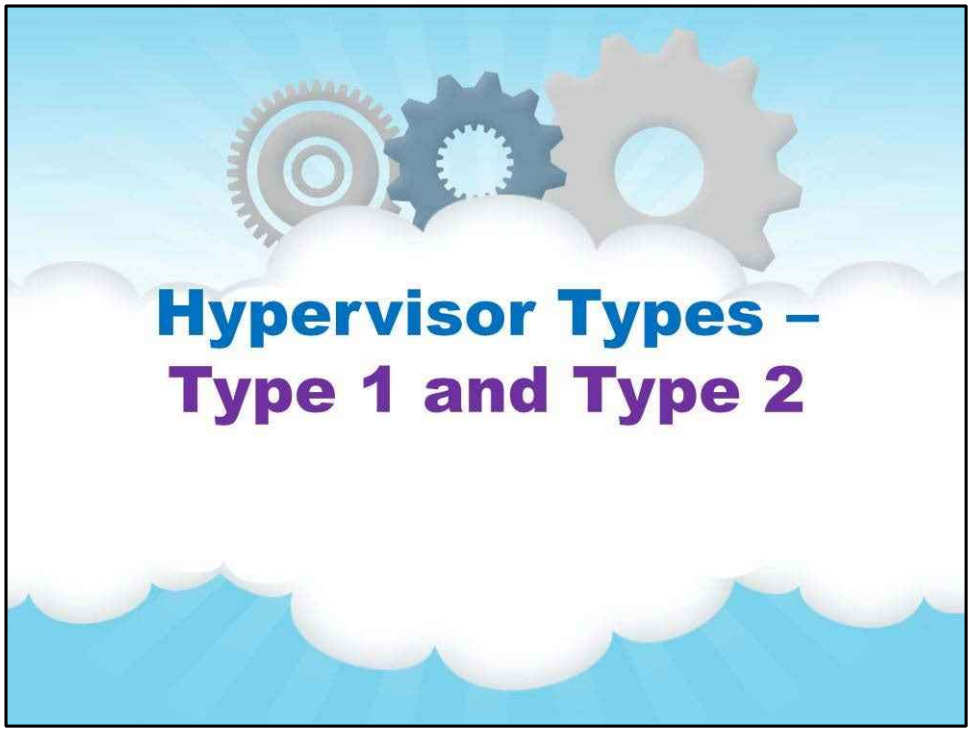
Поєднання типу 1 і типу 2:

Переважно розміщено, але деякі частини знаходяться всередині ядра ОС з міркувань продуктивності

наприклад,**KVM**



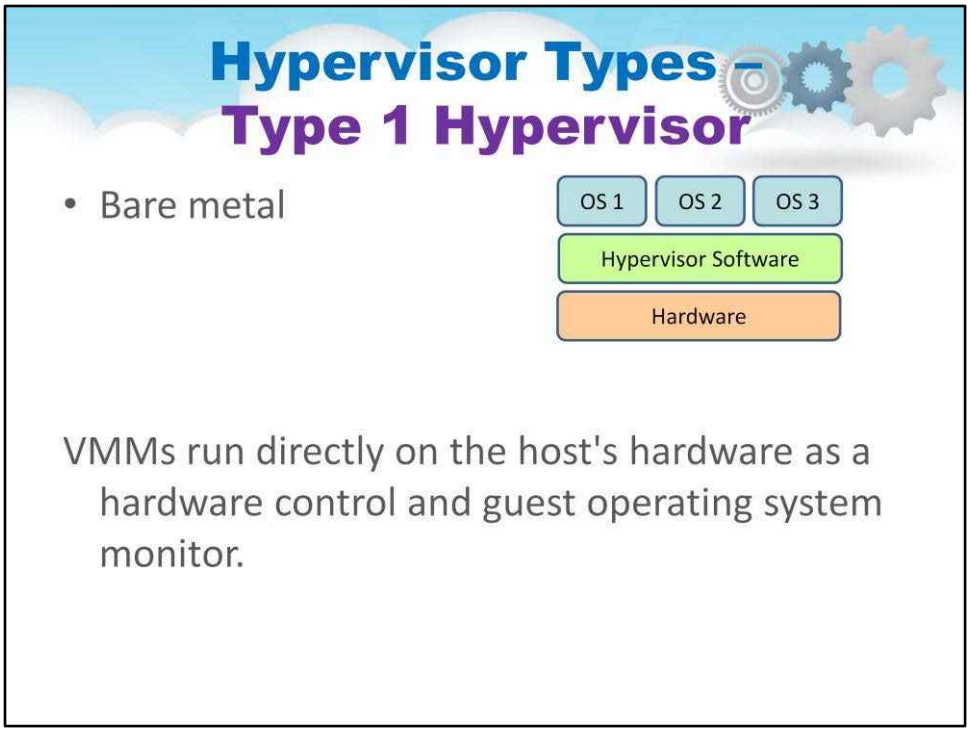
Тут класифікація гіпервізорів узагальнена в загальній порівняльній таблиці з деякими прикладами реалізації.



Тип 1 і тип 2 у деталях

Цей розділ присвячений більш детальному опису гіпервізорів типів 1 і 2:

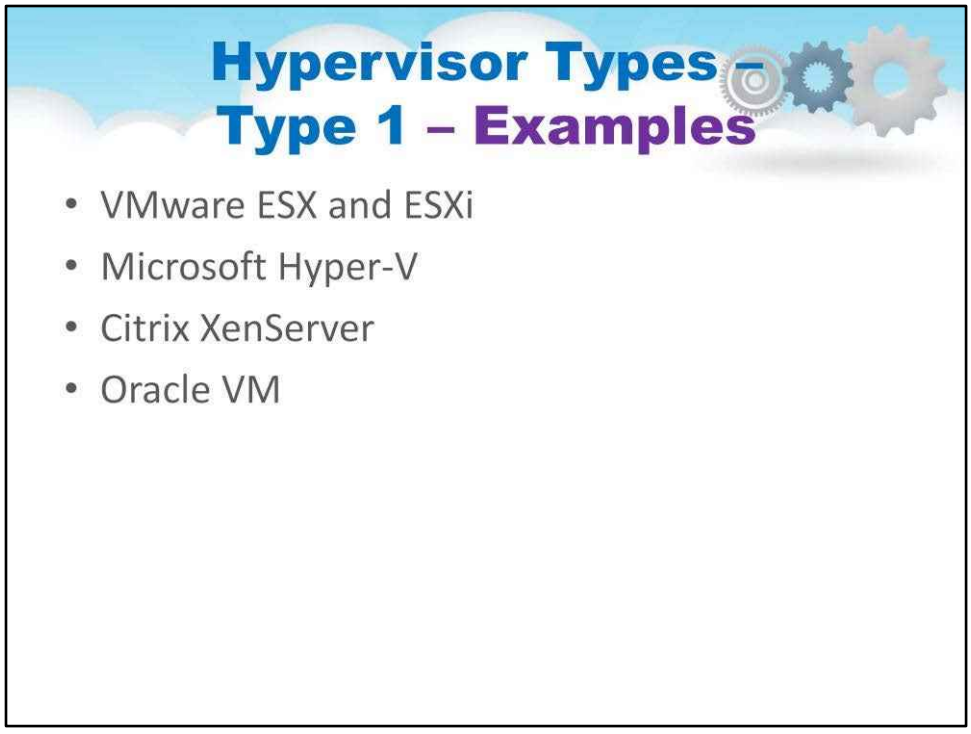
- Структура
- За і Проти
- приклади:
  - Розчин Xen
  - Рішення KVM



Гіпервізор типу 1 має на один рівень менше, ніж гіпервізор типу 2, і розташований безпосередньо на апаратному забезпеченні хоста (див. малюнок 2).добре Amazon-відома Elastic Compute Cloud (EC2) — це веб-сервіс, активований через гіпервізор типу 1. Використання гіпервізора Xen зверху потужних серверів, Amazon може запропонувати «шматочки» масштабованої обчислювальної потужності для своїх клієнтів, що дозволяє кожному сегменту працювати у віртуальному незалежному середовищі.

Гіпервізор типу 1 може бути тоншим за аналог типу 2, але зазвичай вимагає модифікації гостьової ОС. Таким чином, вбудований розробник часто повинен використовувати варіант гостьової ОС, створений спеціально для гіпервізора.

Багато типів 1гіпервізори покладаються на апаратне забезпечення, ніж на можливості віртуалізації. Intel VT,Технологія вбудованого гіпервізора Freescale та ARMTrustZone — це всі технології, які забезпечують основу для рішень віртуалізації. У результаті деякі постачальники апаратного забезпечення приєдналися до руху гіпервізорів і рекламують гіпервізори типу I, оптимізовані для їхніх апаратних платформ. Цей підхід пропонує привабливу альтернативу для систем, які потребують вищої продуктивності.



1. VMware ESX і ESXi

Ці гіпервізори пропонують розширені функції та масштабованість, але вимагають ліцензування, тому витрати вищі. Існує кілька недорогих пакетів, які пропонує VMware, і вони можуть зробити технологію гіпервізора більш доступною для малих інфраструктур.

VMware є лідером у виробництві гіпервізорів типу 1. Їхній продукт vSphere/ESXi доступний у безкоштовній версії та 5 комерційних версіях.

2. Microsoft Hyper-V

Гіпервізор Microsoft, Hyper-V не пропонує багато розширених функцій, які надають продукти VMware. однак,Завдяки XenServer і vSphere Hyper-V є одним із трьох найкращих гіпервізорів типу 1.

Спочатку він був випущений з Windows Server, але тепер Hyper-V значно покращено з Windows Server 2012 Hyper-V. Hyper-V доступний як у безкоштовній версії (без графічного інтерфейсу користувача та прав на віртуалізацію), так і в 4 комерційних версіях – Foundations (лише OEM), Essentials, Standard і Datacenter. Hyper-V

3. Citrix XenServer

Він починався як проект з відкритим кодом.

Основна технологія гіпервізора безкоштовна, але, як і VMware, безкоштовнаESXi, він майже не має додаткових функцій.

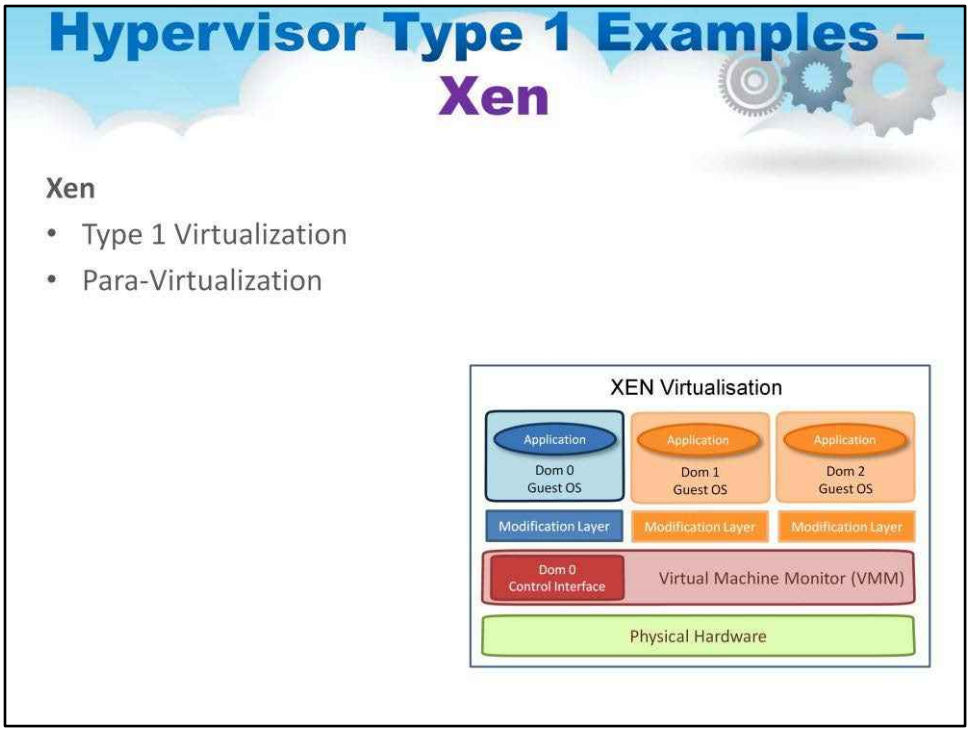
Xen — це голий металевий гіпервізор типу 1. Подібно до того, як Red Hat Enterprise Virtualization використовує KVM, Citrix використовує Xen у комерційному XenServer.

Сьогодні проекти та спільнота Xen з відкритим кодом знаходяться на Xen.org. Сьогодні XenServer — це комерційне рішення гіпервізора типу 1 від Citrix, яке пропонується в 4 версіях. Збентежує те, що Citrix також позначає свої інші пропрієтарні рішення, такі як XenApp і XenDesktop, назвою Xen.

4. Оракул В.М

Гіпервізор Oracle заснований на відкритому коді Xen.

Однак, якщо вам потрібна підтримка гіпервізора та оновлення продукту, це буде коштувати. Oracle VM не має багатьох розширених функцій, наявних в інших гіпервізорах віртуалізації.



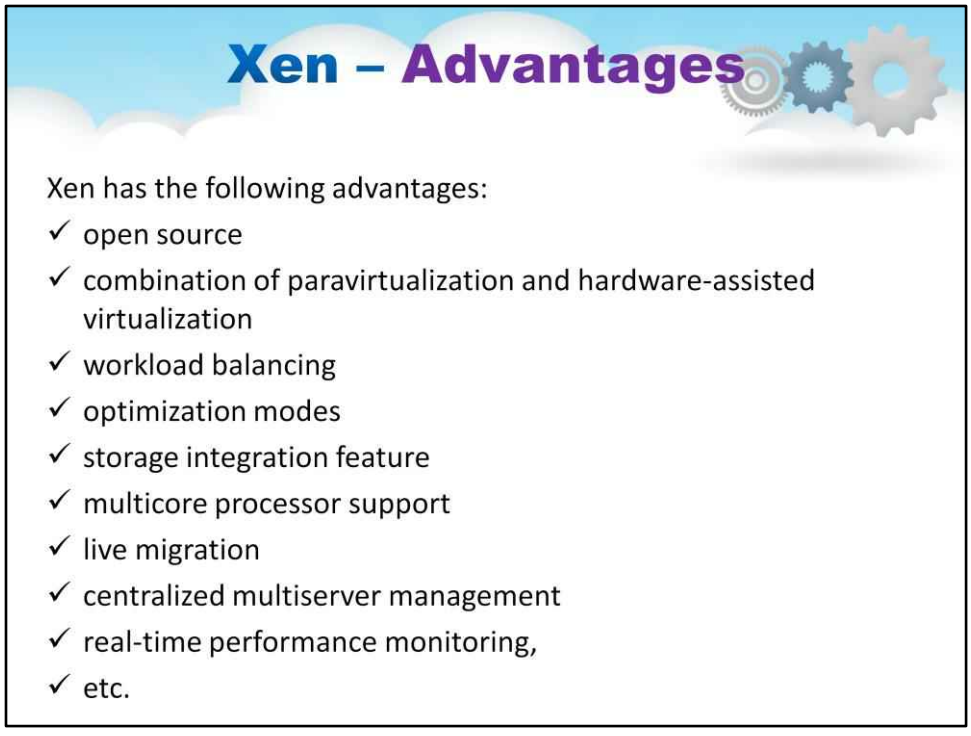
Ось кілька прикладів гіпервізорів, що демонструють різноманіття архітектур.

Перший зображений гіпервізор — XEN.

Можна побачити, що це віртуалізація типу 1, де VMM запускаються безпосередньо на апаратному забезпеченні хоста як апаратне керування та монітор гостьової операційної системи.

Також можна побачити, що це система паравіртуалізації, де VMM не обов'язково імітує апаратне забезпечення, а натомість пропонує спеціальний API, який може використовувати лише модифікована гостьова ОС.
Паравіртуалізація — це техніка віртуалізації, яка представляє програмний інтерфейс для віртуальних машин, схожий, але не ідентичний інтерфейсу основного апаратного забезпечення. Мета полягає в тому, щоб зменшити частину часу виконання гостьової операційної системи, яка витрачається на виконання операцій, які значно складніше виконувати у віртуальному середовищі порівняно з невіртуалізованим середовищем.

Xen може запускати кілька гостьових операційних систем, кожна з яких працює на власній віртуальній машині або домені. Коли Xen встановлюється вперше, він автоматично створює перший домен, домен 0 (або dom0).
Домен 0 є доменом керування та відповідає за керування системою. Він виконує такі завдання, як створення додаткових доменів (або віртуальних машин), керування віртуальними пристроями для кожної віртуальної машини, призупинення віртуальних машин, відновлення роботи віртуальних машин і міграція віртуальних машин. Домен 0 запускає гостьову операційну систему та відповідає за апаратні пристрої.



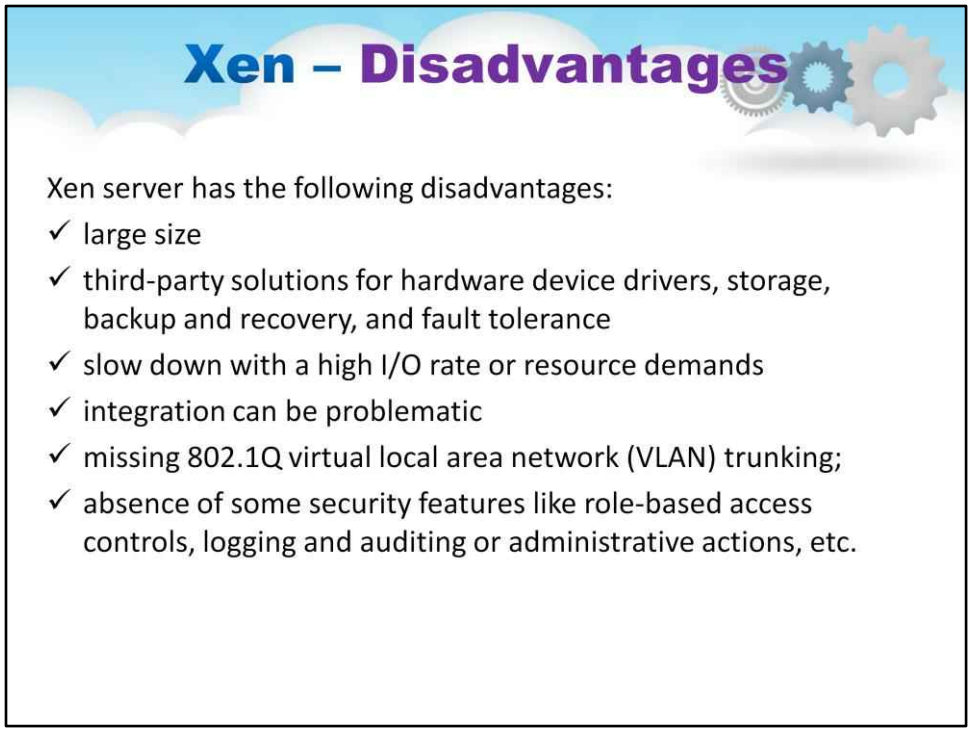
Xen має такі переваги:

- побудований на відкритому гіпервізорі Xen і використовує комбінацію паравіртуалізації та апаратної віртуалізації. Ця співпраця між ОС і платформою віртуалізації дозволяє розробити простіший гіпервізор, який забезпечує високооптимізовану продуктивність.

- забезпечує складне балансування робочого навантаження, яке фіксує дані ЦП, пам’яті, дискового вводу-виводу та мережевих даних вводу-виводу; він пропонує два режими оптимізації: один для продуктивності та інший для щільності.

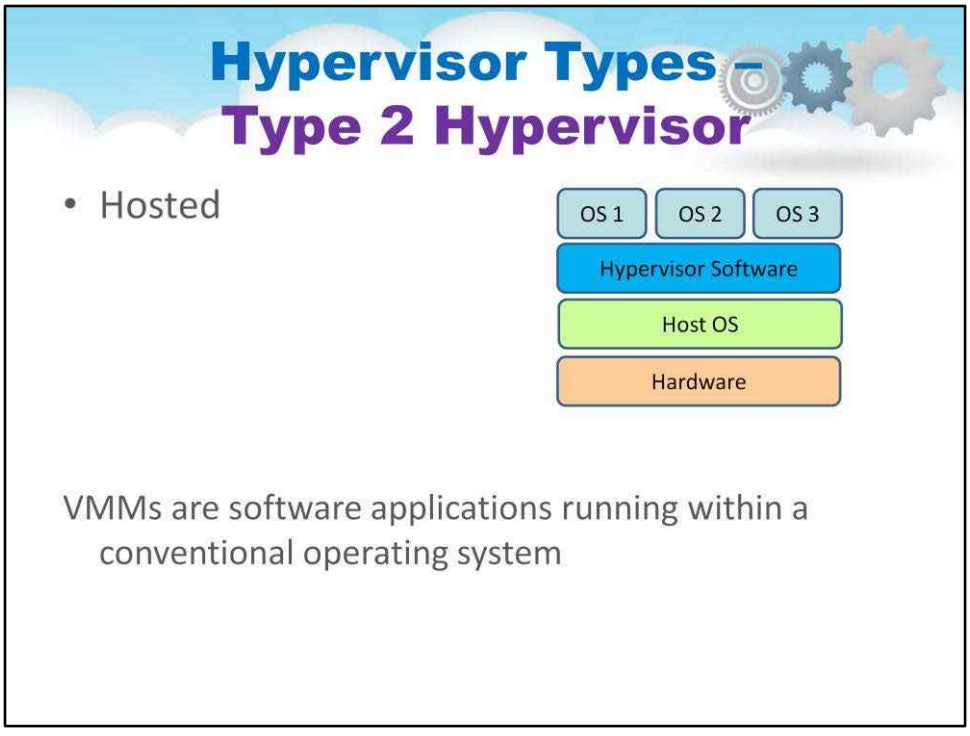
- використовує переваги унікальної функції інтеграції сховища під назвою Citrix Storage Link. За допомогою нього системний адміністратор може безпосередньо використовувати функції масивів таких компаній, як HP, Dell Equal Logic, NetApp, EMC та інших.

- включає підтримку багатоядерних процесорів, оперативну міграцію, інструменти перетворення фізичного сервера на віртуальну машину (P2V) і віртуальну на віртуальну (V2V), централізоване багатосерверне керування, моніторинг продуктивності в реальному часі та швидку продуктивність для Windows і Linux.

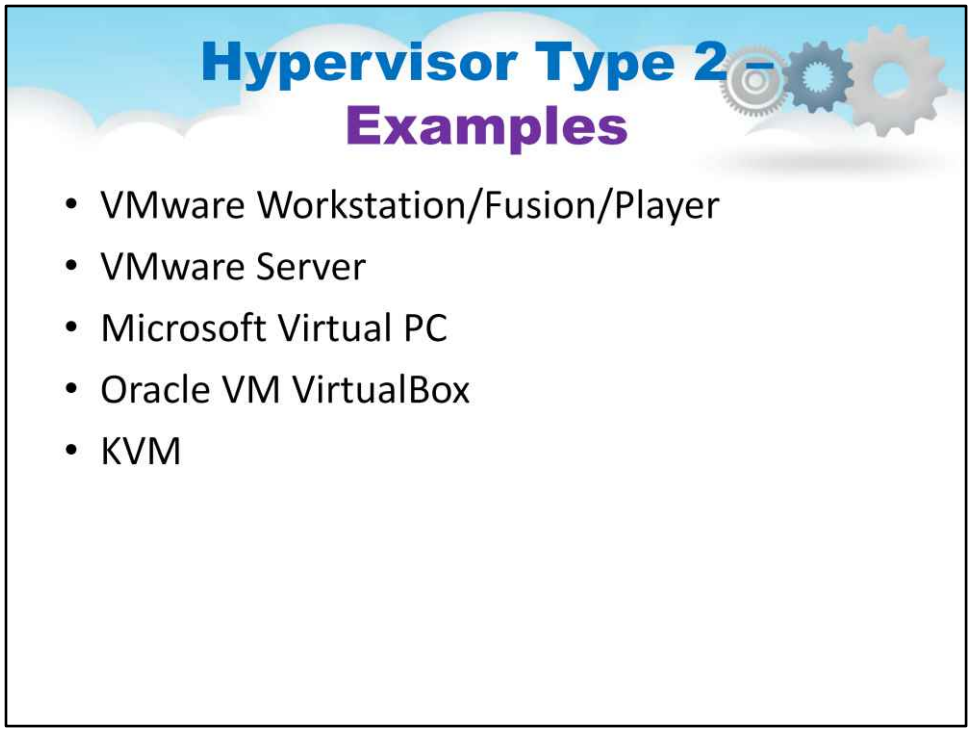


Сервер Xen має наступні недоліки:

- він має відносно великий слід
- він покладається на рішення сторонніх виробників для драйверів апаратних пристроїв, зберігання, резервного копіювання та відновлення, а також відмовостійкості
- він сповільнюється з будь-чим із високою швидкістю вводу-виводу або будь-чим, що забирає ресурси та голодує інші віртуальні машини
- його інтеграція може бути проблематичною; з часом це може стати тягарем для вашого ядра Linux
- відсутній транкінг віртуальної локальної мережі (VLAN) 802.1Q; що стосується безпеки, то він не пропонує інтеграцію служб каталогів, керування доступом на основі ролей або журнал безпеки та аудит або адміністративні дії.



Гіпервізори типу 2 прості у використанні — вони зазвичай не потребують модифікації гостьової ОС. У випадку VMWare програмне забезпечення робочої станції емулює базове апаратне забезпечення та надає гостьовій ОС повний набір віртуальних апаратних ресурсів (зіставляються гіпервізором на фізичні ресурси). Якщо гостьова ОС може працювати на машині x86, вона може працювати на робочій станції без змін. Зручність, яку забезпечує цей тип гіпервізора, має компроміс: додаткові накладні витрати на обробку, які можуть поставити під загрозу продуктивність у реальному часі.



1. VMware Workstation/Fusion/Player

VMware Player — безкоштовний гіпервізор віртуалізації. Він призначений для запуску лише однієї віртуальної машини (VM) і не дозволяє створювати віртуальні машини. VMware Workstation — це надійніший гіпервізор із деякими розширеними функціями, такими як підтримка запису та відтворення та знімків віртуальної машини. VMware Workstation має три основні варіанти використання:

для запуску кількох різних операційних систем або версій однієї ОС на одному робочому столі, для розробників, яким потрібні пісочниці та знімки, або

для лабораторних і демонстраційних цілей.

2. Сервер VMware

VMware Server — це безкоштовний розміщений гіпервізор віртуалізації, дуже схожий на VMware Робоча станція.

3. Microsoft Virtual PC

Це остання версія Microsoft цієї технології гіпервізора Windows Virtual

ПК і працює лише на Windows 7 і підтримує лише операційні системи Windows, що працюють на ньому.

4. Oracle VM VirtualBox

Технологія гіпервізора VirtualBox забезпечує розумну продуктивність і функції, якщо ви бажаєте віртуалізувати з обмеженим бюджетом. Незважаючи на те, що VirtualBox є безкоштовним розміщеним продуктом із дуже малими розмірами, він має багато спільних функцій із VMware vSphere та Microsoft Hyper-V.

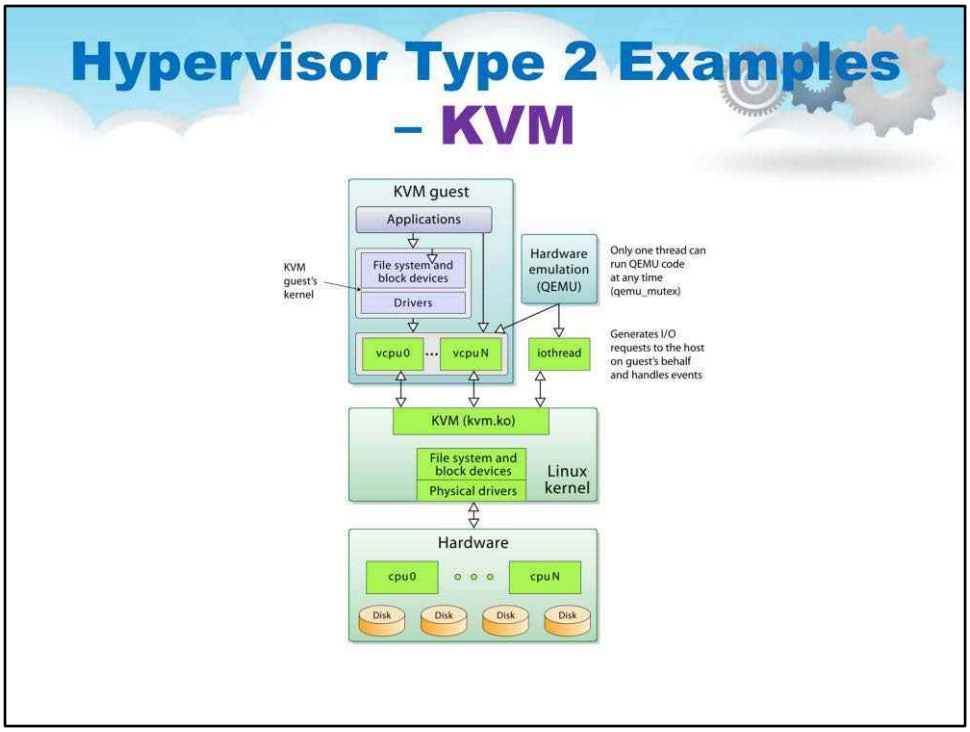
5. Віртуалізація Red Hat Enterprise

Ядро Red Hat -Віртуальна машина (KVM) має якості як розміщеного, так і простого гіпервізора віртуалізації. Він може перетворити саме ядро Linux на гіпервізор, щоб віртуальні машини мали прямий доступ до фізичного обладнання.

KVM

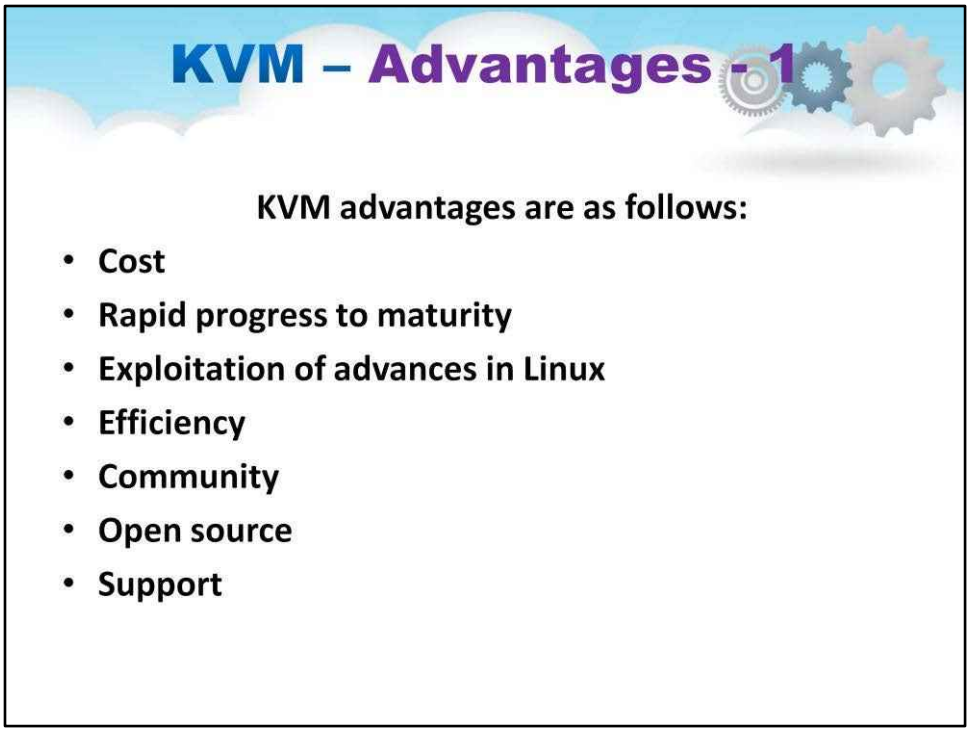
Це інфраструктура віртуалізації для ядра Linux. Він підтримує власну віртуалізацію на процесорах із розширеннями апаратної віртуалізації.

KVM з відкритим вихідним кодом (або віртуальна машина на основі ядра) — це гіпервізор типу 1 на основі Linux, який можна додати до більшості операційних систем Linux, включаючи Ubuntu, Debian, SUSE та Red Hat Enterprise Linux, а також Solaris і Windows .



KVM — це абревіатура від «Віртуальна машина на основі ядра» і є інфраструктурою віртуалізації для ядра Linux, яке перетворює його на гіпервізор. Основна архітектура KVM така.

Віртуальна машина на основі ядра (KVM) — це гіпервізор типу 2, який підтримується компанією Qumranet, Inc. KVM базується на емуляторі QEMU та отримує всі інструменти керування від QEMU.
Основна увага при розробці KVM полягає у використанні розширень x86 VT, які дозволяють віртуальним машинам здійснювати системні виклики.
Версії KVM, новіші за KVM-62, підтримують паравіртуалізовані гостьові системи Linux. KVM використовує набір модулів ядра Linux для забезпечення підтримки VT.
KVM може працювати на стандартному ядрі Linux, яке: (а) є достатньо новим і (б) має вбудовані модулі KVM.
KVM використовується з QEMU для емуляції деяких периферійних пристроїв, які називаються QEMU-KVM.
KVM підтримує формат образу диска QEMU Copy-on-write (QCOW), що дозволяє підтримувати режим моментального знімка для операцій введення-виведення диска.
У режимі знімка всі записи на диск спрямовуються до тимчасового файлу, а зміни не зберігаються у вихідному файлі образу диска.
Кілька віртуальних машин можна запускати з одного образу диска, дещо пом’якшуючи величезні вимоги до пам’яті, пов’язані з розміщенням мережі віртуальних машин.
Знищити віртуальний кластер так само просто, як надіслати SIGKILL до кожного гіпервізора та видалити зображення з диска.
KVM підтримує стандартну модель Linux TUN/TAP для мосту Ethernet. Використовуючи цю модель, кожна віртуальна машина отримує власні мережеві ресурси, завдяки чому її неможливо відрізнити від фізичної машини.



Переваги KVM наступні:

Вартість: враховуючи природу відкритого коду, KVM має нижчу загальну вартість володіння.

Швидкий прогрес до зрілості: спільнота експертів постійно вдосконалює KVM.

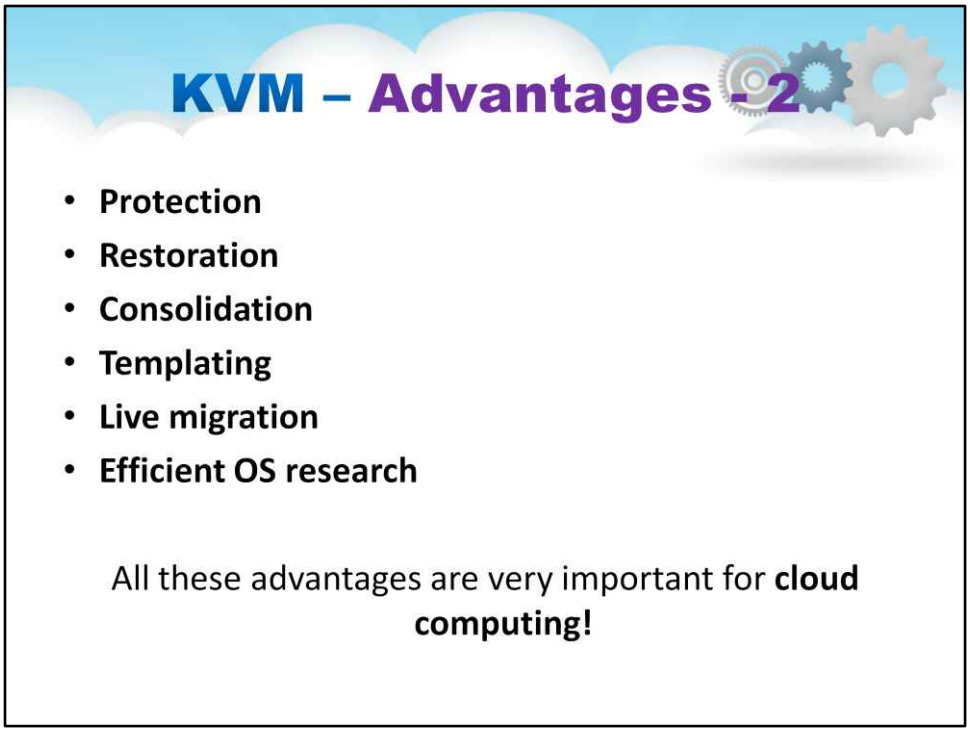
Використання досягнень у Linux: KVM вбудовано в Linux і приносить переваги всій спільноті Linux.

Ефективність: KVM використовує переваги сучасного апаратного забезпечення для безпечної роботи безпосередньо на центральному процесорі та розроблено для хорошої роботи навіть у середовищах з обмеженим обсягом пам’яті та процесора.

Спільнота: Вимоги клієнта до функцій і вразливості безпеки швидко усуваються дуже активною та чуйною спільнотою.

Відкритий вихідний код: код і дані його сховища доступні, постійно перевіряються та прозорі в обґрунтуванні модифікації протягом усього життєвого циклу продукту.

Підтримка: деякі великі гравці (наприклад, IBM) є активними членами спільноти KVM, і це впливає на встановлення пріоритетів розробки KVM



захист-хост-система, захищена від віртуальних машин, віртуальні машини, захищені одна від одної

Вірус має меншу ймовірність поширення

Спільний доступ забезпечується через спільний том файлової системи, мережевий зв'язок

Реставрація-можна заморозити,**призупинити**, запущений VM

Потім можна перемістити або скопіювати в інше місце**резюме**

Зробити знімок певного стану з можливістю відновлення до цього стану

Деякі VMM дозволяють робити кілька знімків на одну віртуальну машину **Клон**шляхом створення копії та запуску як оригіналу, так і копії

Консолідація–може запускати кілька різних ОС на одній машині

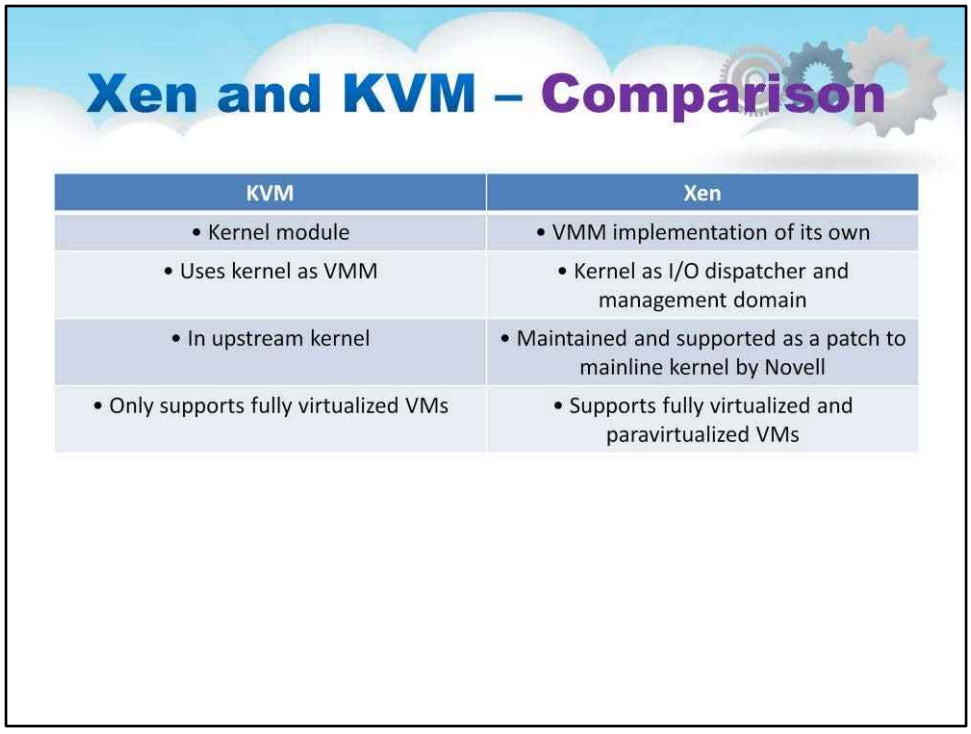
Розробник ОС,розробник програм, ...

шаблонування–щоб створити віртуальну машину OS + додаток, надати її клієнтам, використовувати для створення кількох екземплярів цієї комбінації

Жива міграція–перемістити запущену віртуальну машину з одного хоста на інший!

Ефективне дослідження ОС–дозволяють підвищити ефективність розробки операційної системи

Всі ці переваги дуже важливі для**хмарні обчислення, оскільки**Використовуючи API, програми можуть працювати в хмарній інфраструктурі (сервери, мережа, сховище) для створення нових гостей, віртуальних машин, віртуальних робочих столів!



Давайте порівняємоXen або KVM – будь ласка, цю таблицю з деякими зауваженнями щодо їхніх особливостей.
Який гіпервізор найкращий для вашої хмари: Xen чи KVM?

Xen

Xen почався приблизно в 2004 році, коли ще не було віртуалізації з відкритим кодом.
Xen,який є гіпервізором типу I, розташований на одному шарі над голим металом. Це як скорочена операційна система, і вона використовує функцію під назвою «pass-through» для прямого підключення до Пристрої PCI, такі як RAM/CPU/NIC.

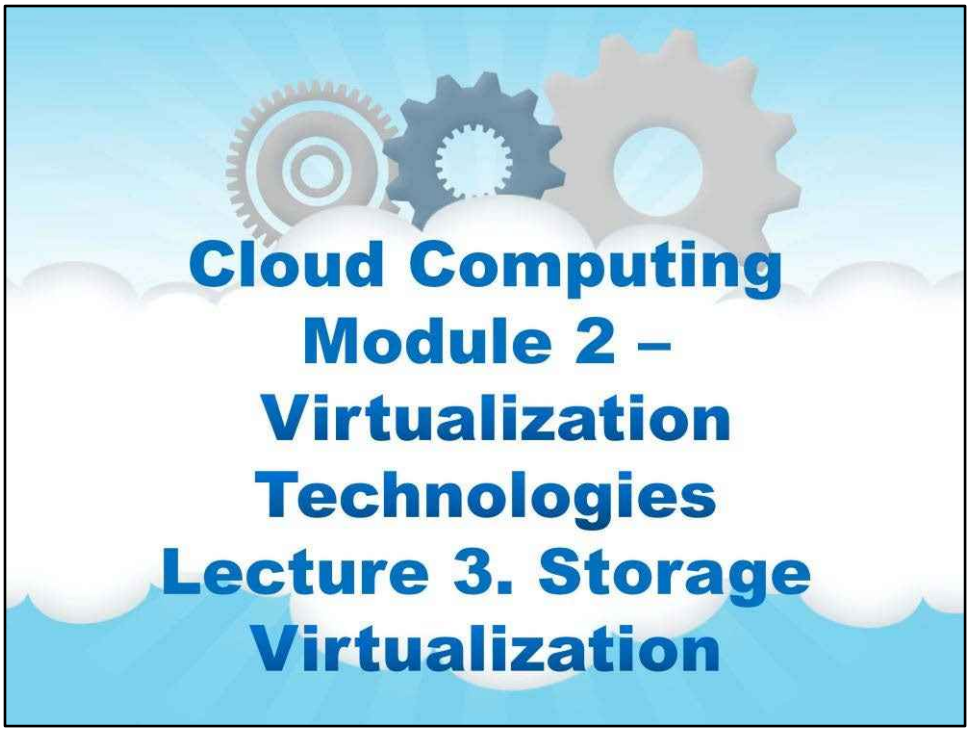
KVM

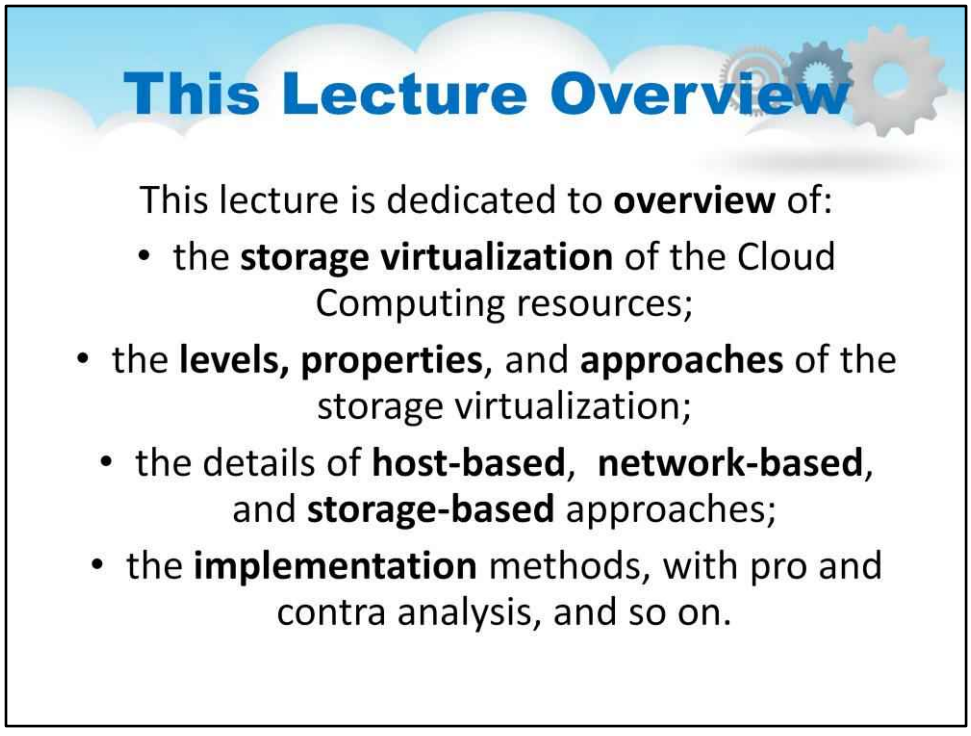
KVM (віртуальна машина на основі ядра) спочатку була написана та випущена приблизно в 2007 році.
KVM, який є гіпервізором типу II, розташований на один рівень над ОС. Якщо у вас машина на голому металі, вам потрібно встановити ОС, а потім інсталювати KVM.
У Linux він встановлюється у формі ядра, і це ядро потім перетворює голу машину на гіпервізор.

Вони обидва мають відкритий вихідний код і обидва підтримуються великими спільнотами та великими підприємствами.
Вони обидва виконують фундаментальну роботу віртуалізації.
Відмінності полягають у підтримці інновацій, інтеграції додатків, обізнаності та продуктивності додатків.
Xen, як правило, більш стабільний. Він старший і зріліший. Xen пропонує майже власні драйвери для таких ОС, як Microsoft Windows, тоді як KVM у цьому відношенні слабший. Однак KVM дуже добре працює з Linux.

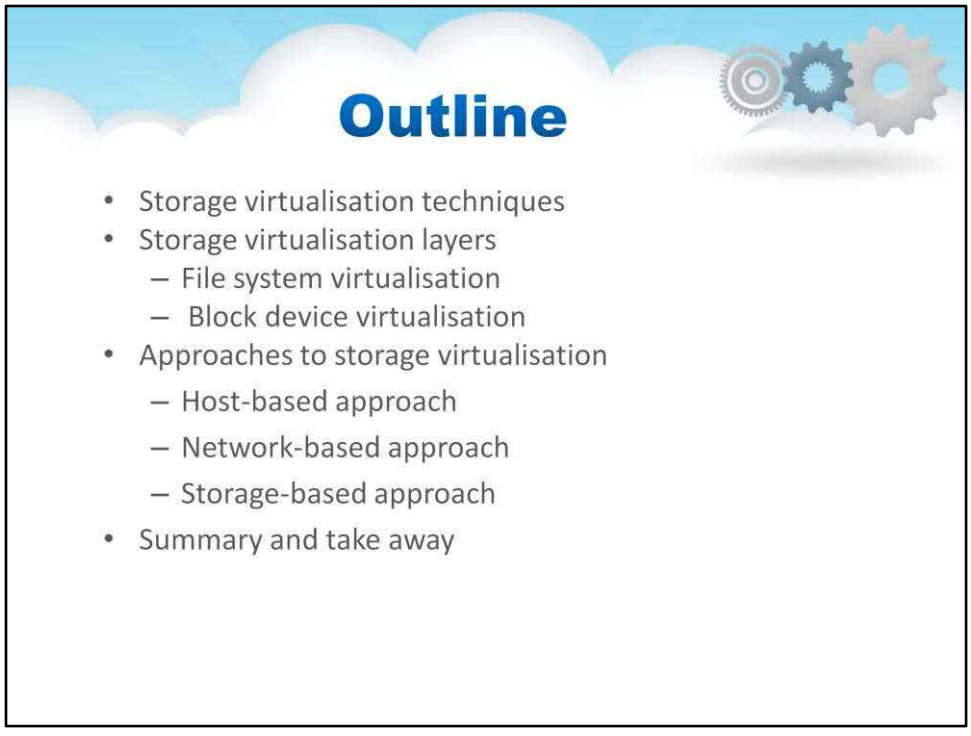
Загальні поради:

Якщо ви маєте справу з багатьма рішеннями на базі Linux, вам підійде KVM. Це не означає Xen погано. Просто є ймовірність, що ви побачите, що KVM запропонує кращу продуктивність із ОС Linux.
Якщо ви маєте справу з багатьма рішеннями на базі Windows, ви підете з нимиXen, оскільки хостинг на Xen має більше переваг завдяки майже рідним драйверам.


Cloud Computing
Module 2 –
Virtualization
Technologies
Lecture 3. Storage
Virtualization



Лекція 3. Віртуалізація сховищ



Як ви можете бачити на слайді, ми будемо охоплювати кілька сфер, пов’язаних зі сховищем Віртуалізація в Cloud IaaS

Методи віртуалізації сховищ

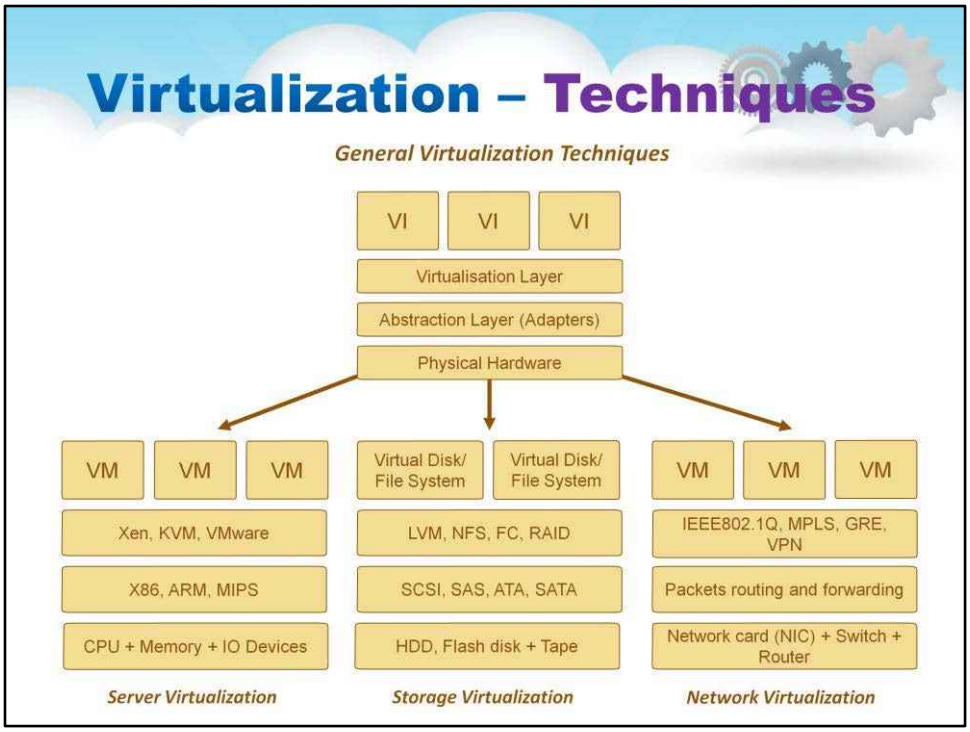
Рівні віртуалізації сховищ

Підходи до віртуалізації сховищ

Типи зберігання в хмарі

Віртуалізовані файлові системи для хмари

Огляд



VI -Віртуалізований екземпляр

ВМ -Віртуальна машина

Цей слайд ілюструє методи віртуалізації в Cloud IaaS

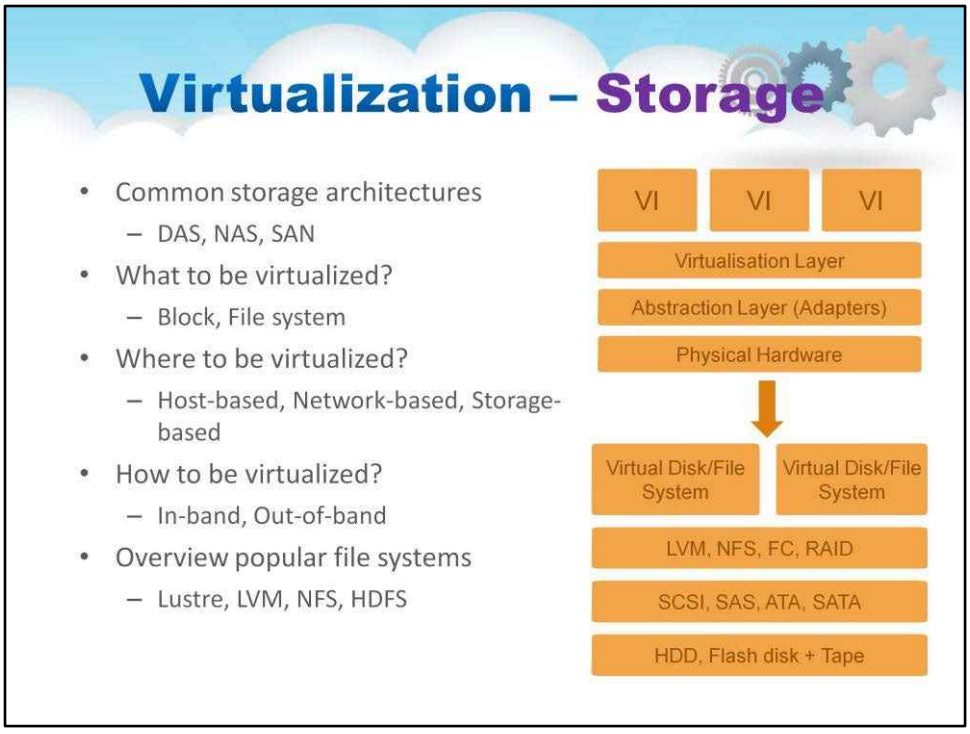
У верхній частині показано загальну техніку віртуалізації.

Нижче можна візуалізувати, як загальна техніка застосовується до кількох проблем віртуалізації.

Для віртуалізації серверів широко використовуються гіпервізори.

Для віртуалізації сховищ використовується багато програмних методів, включаючи керування томами, файлові системи та реплікацію.

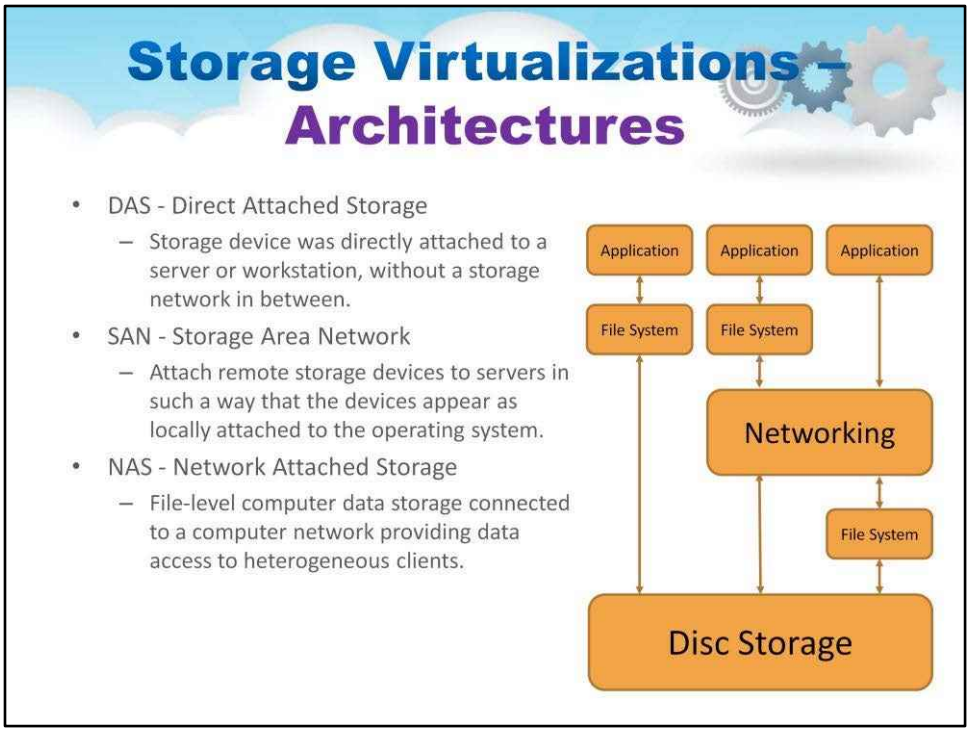
Для віртуалізації мережі існує багато різних функцій, включаючи агрегацію посилань, VPN, а також брандмауер, комутацію, маршрутизацію та фільтрацію додатків і балансування навантаження мають віртуальні можливості в багатьох сьогоднішніх хмарних реалізаціях.



VI -Віртуалізований екземпляр

Як ми бачили з попередніх уроків, віртуалізація будь-якого виду ресурсу слідує загальному плану. Фізичне обладнання поміщається під особливий вид програмного забезпечення елемент керування, який абстрагує фізичний рівень і представляє користувачеві те, що «виглядає» як фактичний ресурс, але насправді є «віртуальним» екземпляром цього ресурсу.

Зберігання відповідає цьому плану. Апаратне забезпечення та апаратні інтерфейси віртуалізуються програмним рівнем, як показано на ілюстрації, таким чином представляючи споживачеві примітиви віртуального зберігання (диски, файлові системи тощо).



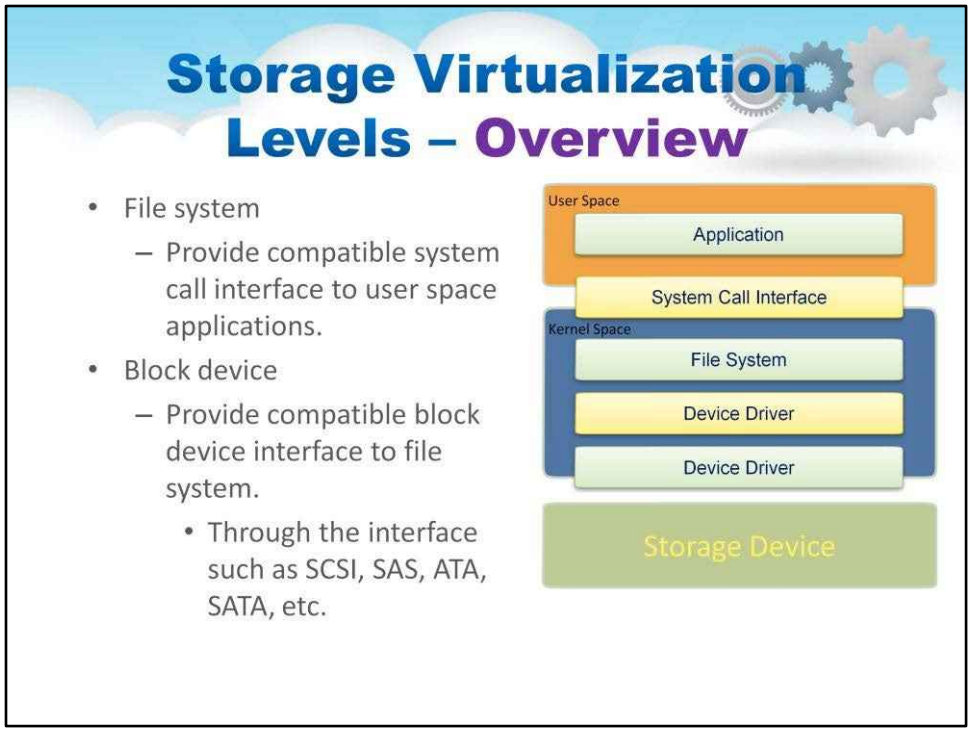
У хмарах існує багато варіантів реалізації фізичного сховища, оскільки інтерфейси віртуального сховища можна зберегти в невеликому наборі, а програмне забезпечення забезпечує загальні інтерфейси.

На схемі можна побачити, що базові архітектури цих різних схем досить різні. Інтерфейс програми (застарілий, не віртуальний) виглядає однаково – в основному програма отримує доступ до файлової системи. Під капотом підключення компонентів – і місце, де фактично працює код файлової системи – може бути в будь-якій кількості місць (як показано)

Рівень програмного забезпечення, який реалізує віртуалізоване сховище, також може покращити пропоновану модель сховища за межі того, що може виконати фізичне програмне забезпечення.

Віртуалізація сховищ включає об’єднання кількох фізичних ресурсів сховища (загалом гетерогенних), їх абстракцію та подальшу логічну композицію або сегментацію.

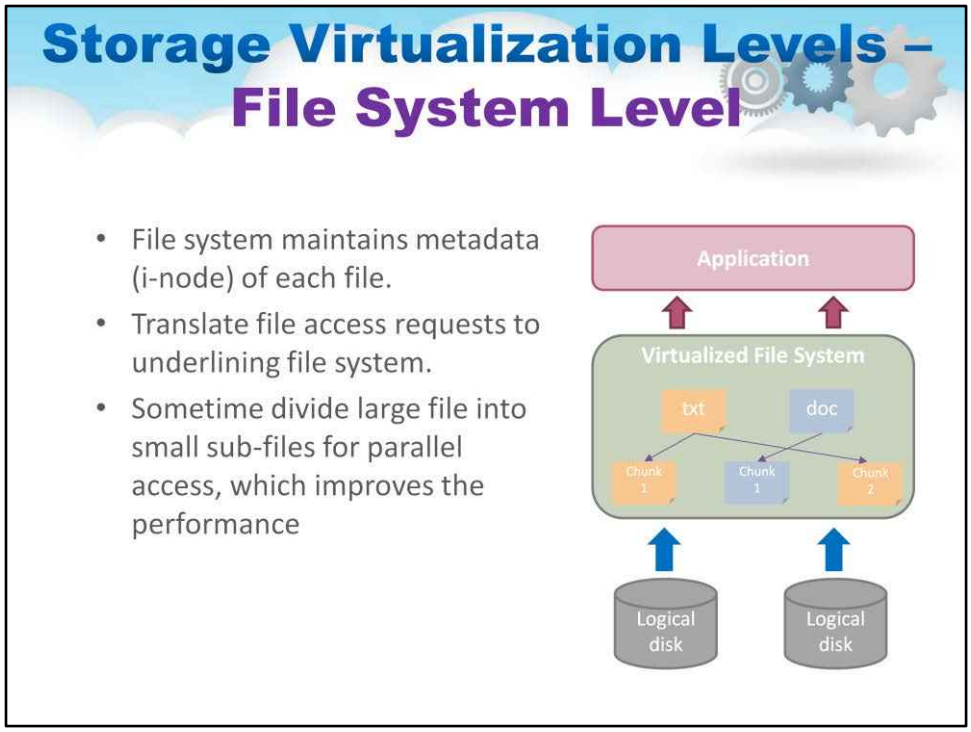
Рівні



Ілюстрація на цьому слайді детальніше пояснює, як працюють рівні віртуалізації для зберігання та яку модель зберігання вони представляють.

Рівень програмного забезпечення для зберігання знаходиться в просторі ядра в операційній системі, де він може перехоплювати примітиви дискової та файлової системи та вставляти можливість використання зовнішнього, мережевого сховища та сховища, створеного з реплікованих розподілених дисків.

Найпоширенішими моделями зберігання є «файлова система» та «блоковий пристрій». У той час як назви вказують на можливості моделей, хмарна ОС може не надавати таких самих можливостей, як стандарт, скажімо, файлова система Linux або блоковий пристрій. Ми обговоримо це пізніше.



Що таке файлова система

- Файлова система – це рівень програмного забезпечення, відповідальний за організацію та контроль створення, зміна та видалення файлів
- Файлові системи забезпечують ієрархічну організацію файлів у каталоги та підкаталоги
- Алгоритм B-дерева сприяє більш швидкому пошуку та пошуку файлів за назвою
- Цілісність файлової системи підтримується шляхом дублювання головних таблиць, журналів змін і негайного списання змін у файлі

Різні файлові системи

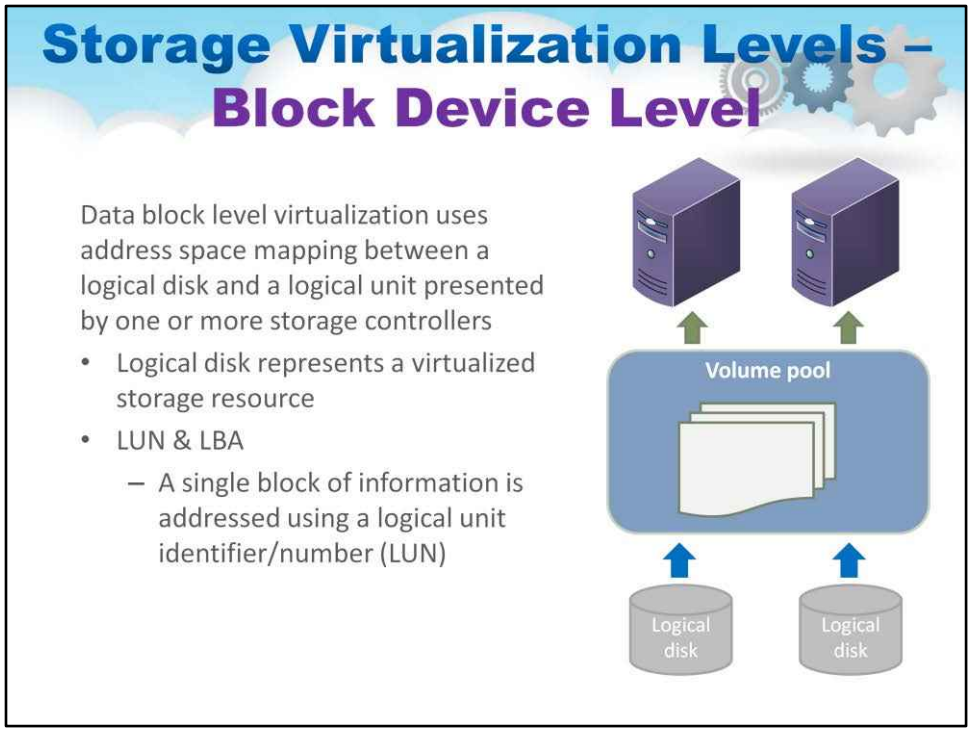
- В Unix суперблок містить інформацію про поточний стан файлової системи та її ресурсів.
- У Windows NTFS головна таблиця файлів містить інформацію про всі записи та стан файлів.

Метадані файлу

- Керуюча інформація для керування файлами відома як метадані.
- Метадані файлу містять атрибути файлу та вказівники на розташування вмісту файлу даних.
- Метадані файлу можуть бути відокремлені від вмісту даних файлу.
- Метадані про право власності на файл і дозволи використовуються для доступу до файлу.
- Метадані часових позначок файлу полегшують автоматизовані процеси, такі як резервне копіювання та керування життєвим циклом.

Різні файлові системи

- У системах Unix метадані файлів містяться в структурі i-node.
- У системах Windows метадані файлів містяться в записах атрибутів файлів.



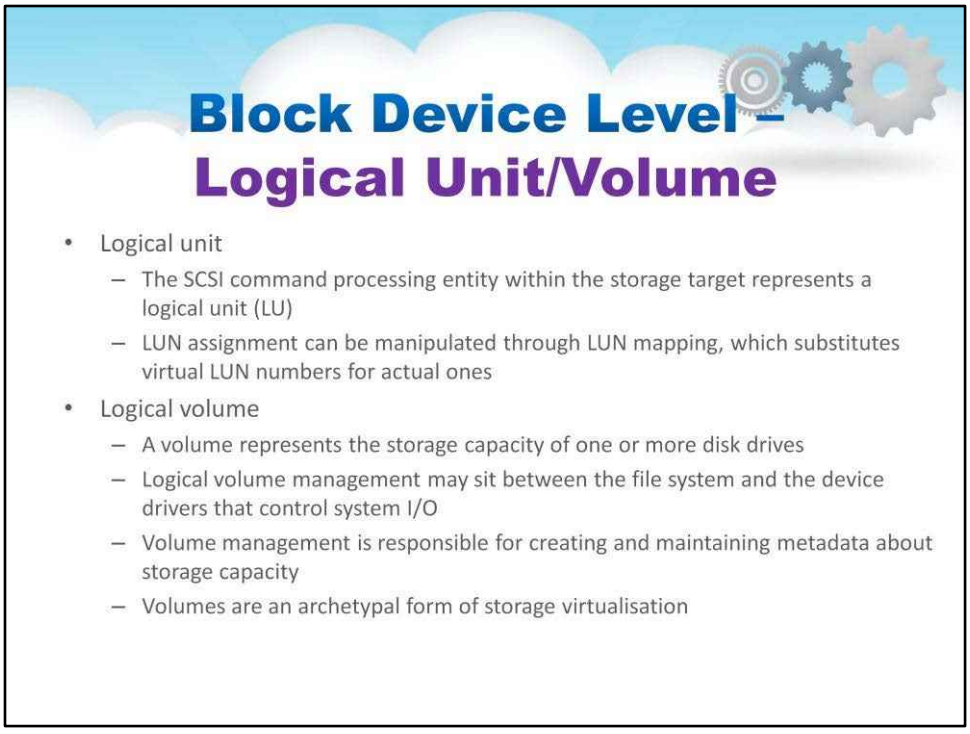
Віртуалізація на рівні блокових пристроїв — це техніка низького рівня, яка створює «пул томів» із колекції дисків. Він представляє віртуалізовані примітиви зберігання під назвою LUN

для ідентифікатора логічної одиниці та зсув у цьому LUN, який

відомий як адреса логічного блоку (LBA)

Це показано на слайді

Окремий блок інформації адресується за допомогою ідентифікатора/номера логічного блоку (LUN) і зсуву в межах цього LUN, відомого як адреса логічного блоку (LBA).



Дані рівня блоку

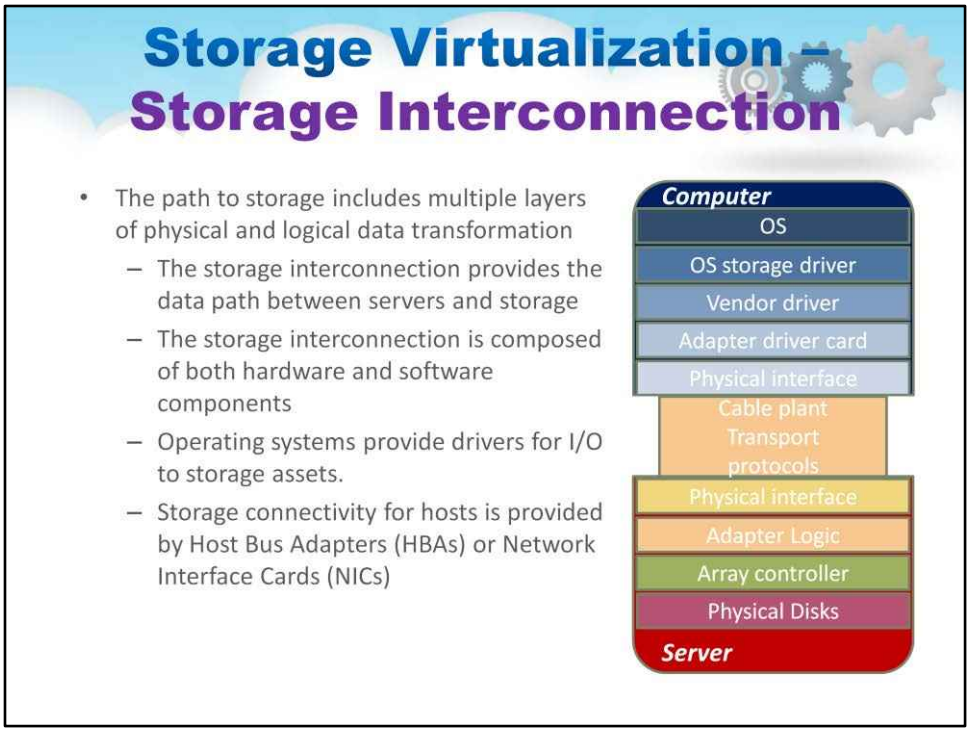
Блок файлової системи

- Атомною одиницею керування файловою системою є блок файлової системи.
- Дані файлу можуть охоплювати кілька блоків файлової системи.
- Блок файлової системи складається з послідовного діапазону адрес блоку диска. Дані на диску
- Дискові пристрої зчитують і записують дані на носій через циліндр, головку та секторну геометрію.
- Мікрокод на диску перетворює номери блоків дисків на розташування циліндрів/головок/секторів.
- Цей переклад є елементарною формою віртуалізації.

Інтерфейс рівня блокового пристрою: SCSI – інтерфейс малої комп'ютерної системи

- Обмін блоками даних між хост-системою та сховищем регулюється протоколом SCSI.

Об’єкт обробки команд SCSI у цільовому сховищі представляє логічну одиницю (LU) і їй присвоюється номер логічної одиниці (LUN) для ідентифікації хост-платформою



Диски не завжди є локальними для сервера, тому використовується взаємозв’язок зберігання.

Ця ілюстрація показує, що шлях до сховища включає кілька рівнів фізичного та логічного перетворення даних

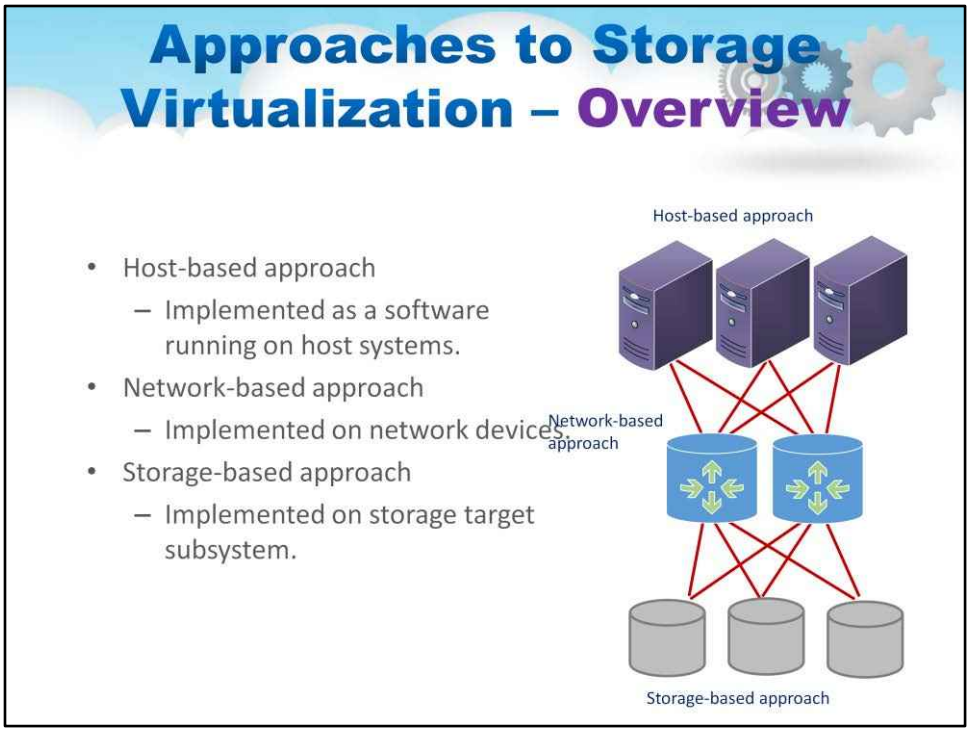
Взаємозв’язок зберігання забезпечує шлях даних

між серверами та сховищем

Взаємозв’язок зберігання складається з обох

апаратні та програмні компоненти

Підходи



Абстрагування фізичного зберігання

Від фізичного до віртуального

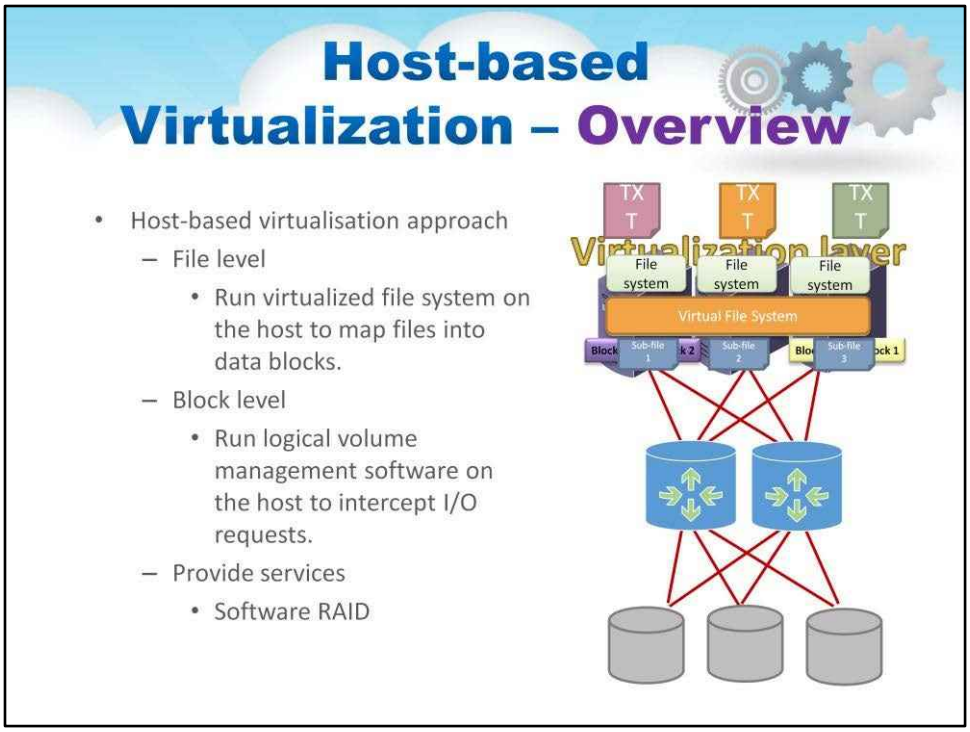
Геометрія циліндрів, головок і секторів окремих дисків віртуалізується в адреси логічних блоків (LBA). Для мереж зберігання фізична система зберігання ідентифікується парою мережева адреса/LUN. Поєднання ресурсів RAID і JBOD для створення віртуалізованого дзеркала має враховувати різницю в продуктивності.

Цілісність метаданих

Цілісність метаданих сховища вимагає резервування для відновлення після відмови або балансування навантаження.

Для забезпечення узгодженості даних інтелекту віртуалізації може знадобитися взаємодія з програмами верхнього рівня.

На основі хоста



Важливі питання

Сервери зберігання метаданих

Метадані зберігання можуть використовуватися кількома серверами.

Спільні метадані дозволяють переглядати файлову систему SAN для кількох серверів.

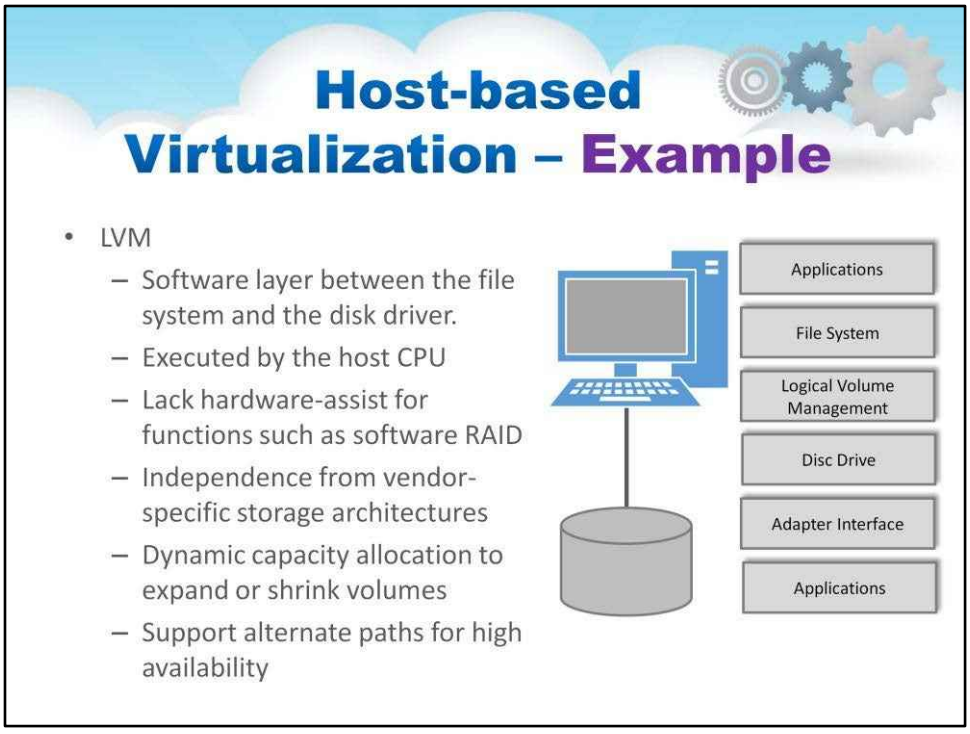
Забезпечує відображення адрес віртуального логічного блоку для клієнта.

Розподілена файлова система SAN потребує механізмів блокування файлів для збереження цілісності даних.

API зберігання на основі хоста

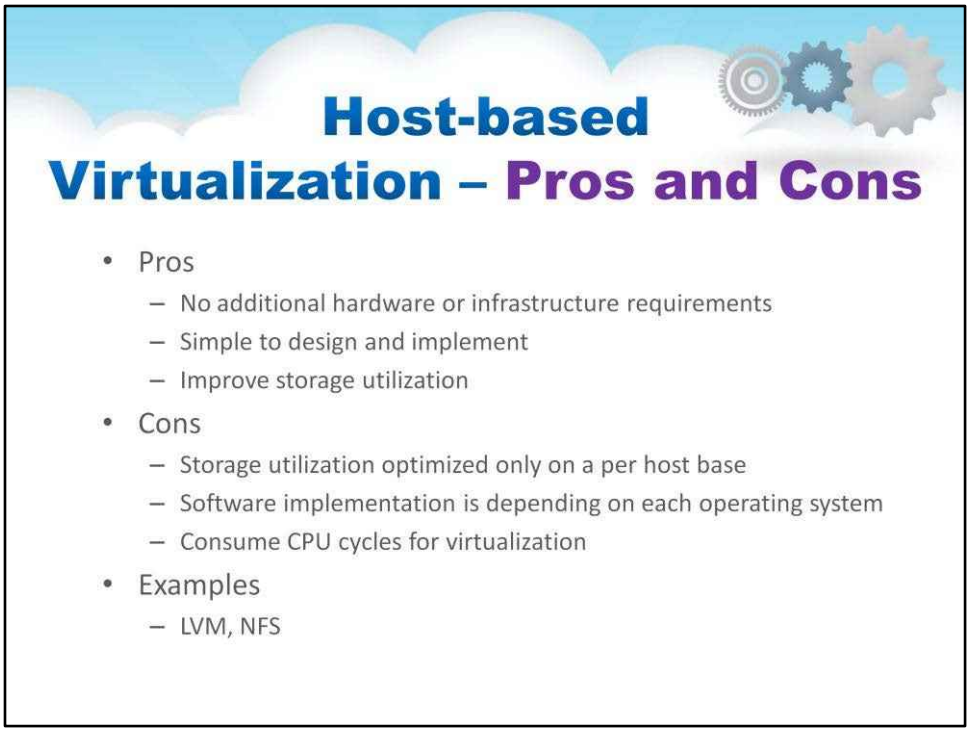
Може бути реалізований операційною системою для забезпечення спільного інтерфейсу для розрізнених віртуалізованих ресурсів.

Служба віртуальних дисків Microsoft (VDS) надає інтерфейс керування для динамічного створення віртуалізованого сховища.



На кожному хості можна запустити додатковий рівень абстракції та керування, який називається диспетчером логічних томів (LVM). Цей код працює на хості та зовнішніх ресурсах усіх типів внутрішніх ресурсів зберігання.

Варіанти використання та архітектура показані на слайді.



Віртуалізація сховища на основі хоста стала дуже популярною на серверних машинах, оскільки

Немає додаткових вимог до обладнання чи

інфраструктури

Простий у розробці та впровадженні

Покращте використання сховища

Проте

Використання пам’яті оптимізовано лише для кожного

хоста

Реалізація програмного забезпечення залежить від кожної

операційної системи

Споживайте цикли ЦП для віртуалізації

Як ми всі знаємо, NFS дуже популярна, і це форма віртуалізації сховища на основі хосту

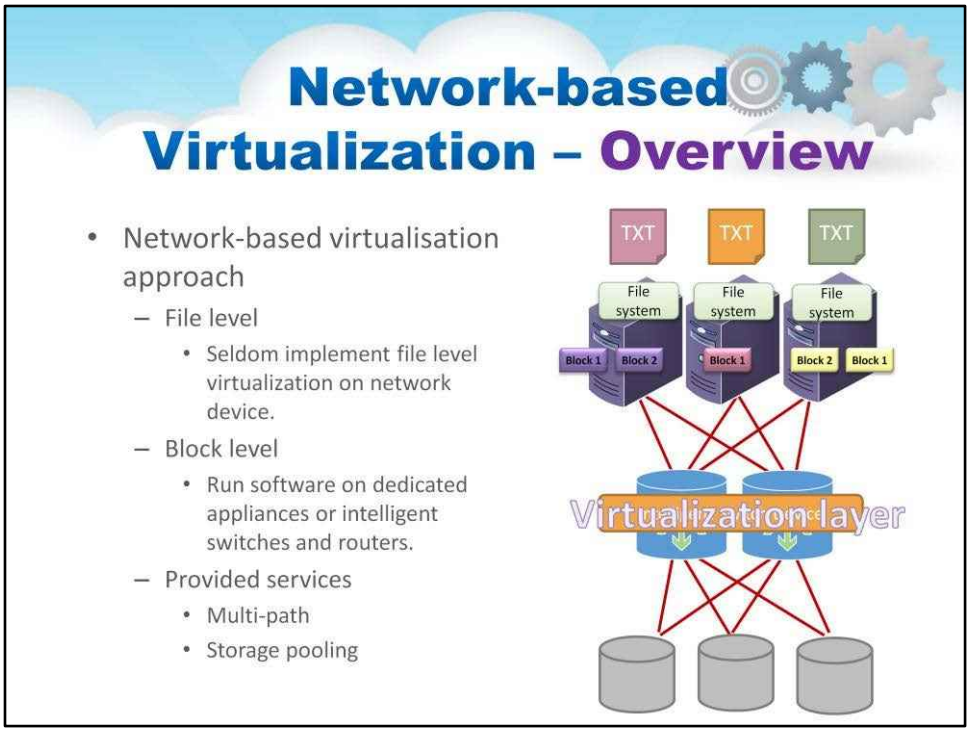
Рівень файлу

Запустіть віртуалізовану файлову систему на хості, щоб зіставити файли в блоки даних, які розподіляються між кількома пристроями зберігання.

Рівень блоку

Запустіть програмне забезпечення для керування логічним томом на хості, щоб перехоплювати запити вводу-виводу та перенаправляти їх на пристрої зберігання.

Мережевий



Анімація ілюструє…

Перемикач тканини повинен забезпечити

Підключення для всіх транзакцій зберігання.
Взаємодія між різними серверами, операційними системами та цільовими пристроями

FAIS (стандарт інтерфейсу додатків Fabric)

Визначте набір стандартних API для

інтеграції програм і комутаторів.

FAIS розділяє керуючу інформацію та шляхи

даних.

Процесор шляху керування (CPP) підтримує API

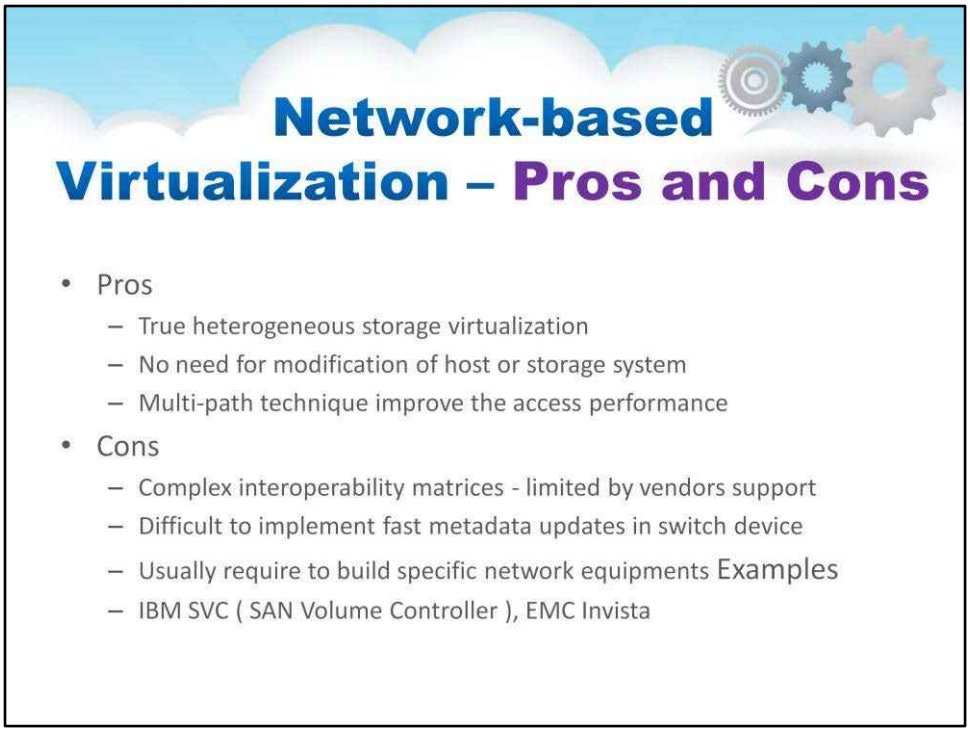
FAIS і програму віртуалізації сховища

верхнього рівня.

Контролер шляху даних (DPC) виконує

віртуалізований вхід/вихід SCSI під керуванням

одного або кількох CPP



Віртуалізація на основі мережі також має плюси та мінуси

Справжня віртуалізація гетерогенних сховищ

Немає необхідності модифікувати хост або сховище

система

Технологія Multi-path покращує доступ

продуктивність

Проте

Комплексні матриці взаємодії - обмежені

підтримка постачальників

Важко реалізувати швидке оновлення метаданих

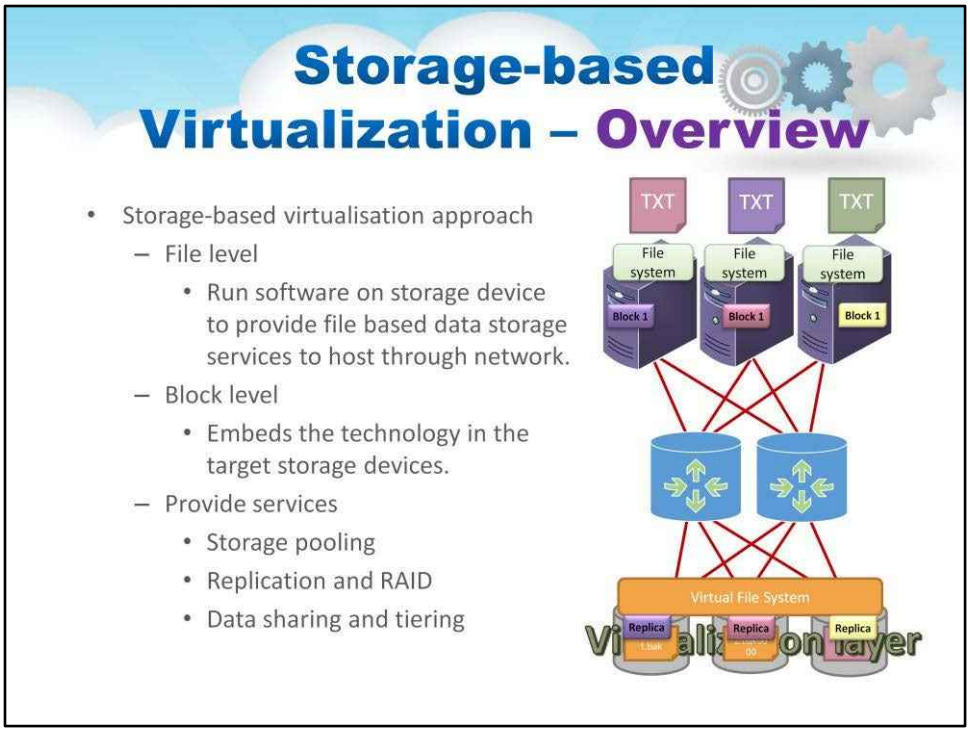
перемикач пристрою

Зазвичай потрібно створити спеціальну мережу

обладнання (наприклад, Fibre Channel)

Прикладом є IBM SVC (SAN Volume Controller).

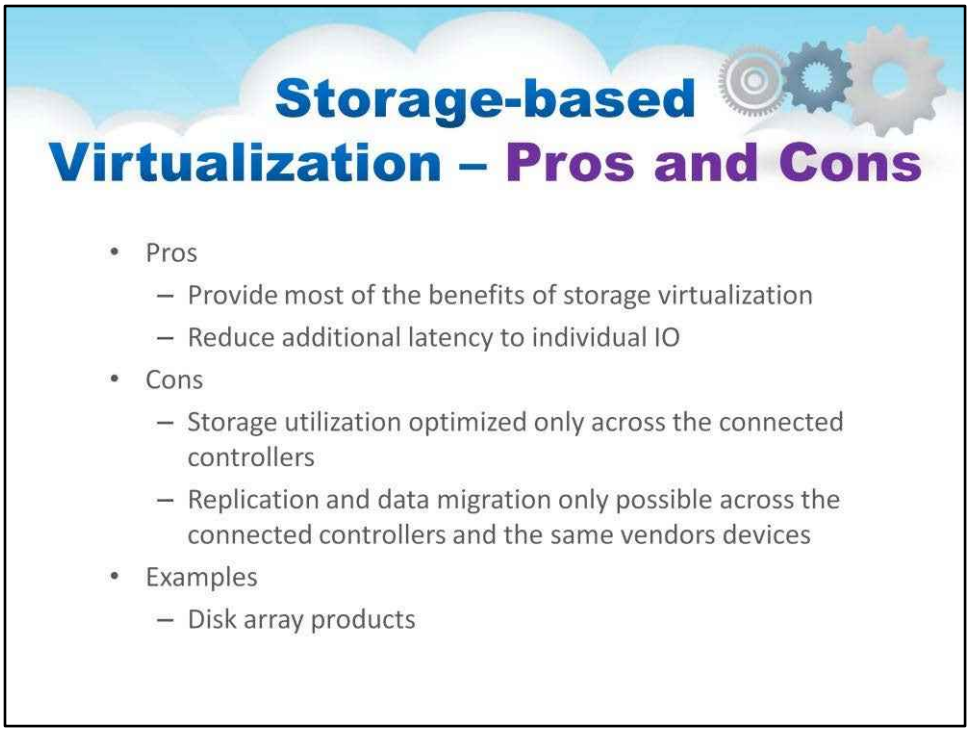
На основі зберігання



Ця анімація ілюструє, як різні рівні в системі зберігання виконують свої функції.

У першій частині анімації можна побачити режим, у якому базове віртуалізоване сховище забезпечує інтерфейс віртуальної файлової системи. Підключені операційні системи надсилають туди файли для збереження. Віртуалізоване сховище піклується про реплікацію файлу на фактичних дисках у хмарі для високої надійності.

У другій частині анімації можна побачити режим, у якому представлено базове віртуалізоване сховище та інтерфейс на рівні блоків. Тут програма працює на ОС, яка представляє інтерфейс локальної файлової системи. Операційна система розбиває за допомогою коду файлової системи збереження на серію блоків, які потрібно записати. У цьому випадку блоки переходять на рівень віртуалізації, який зберігає та реплікує на рівні блоків.



Віртуалізація сховищ надзвичайно корисна

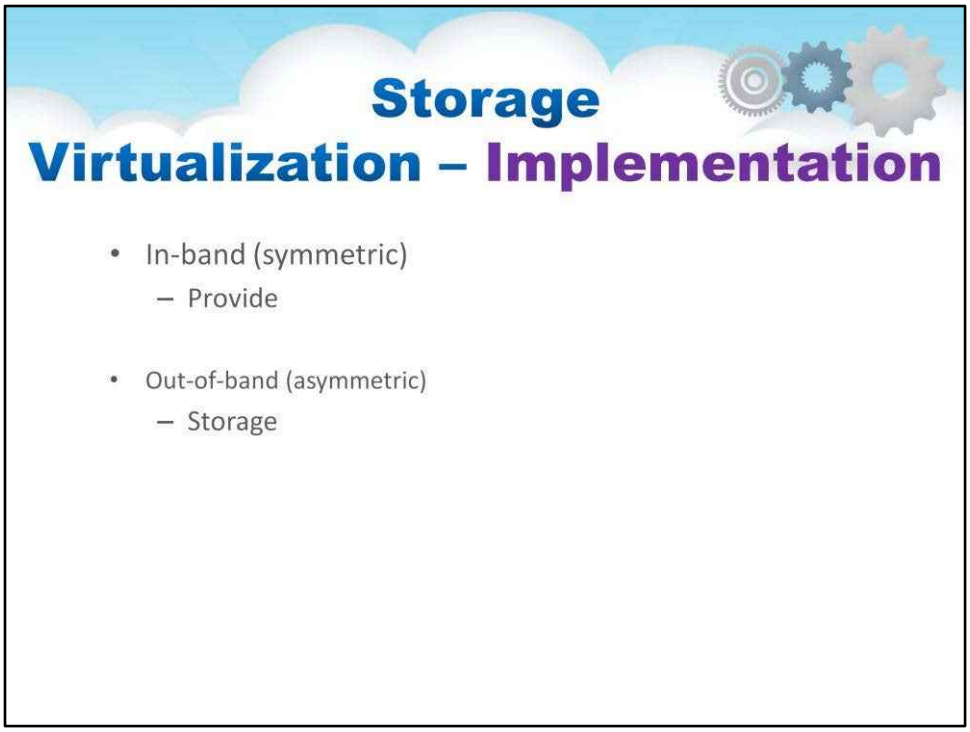
З одного боку

Забезпечте більшість переваг віртуалізації сховища. Зменште додаткову затримку для окремих операцій вводу-виводу

Проте

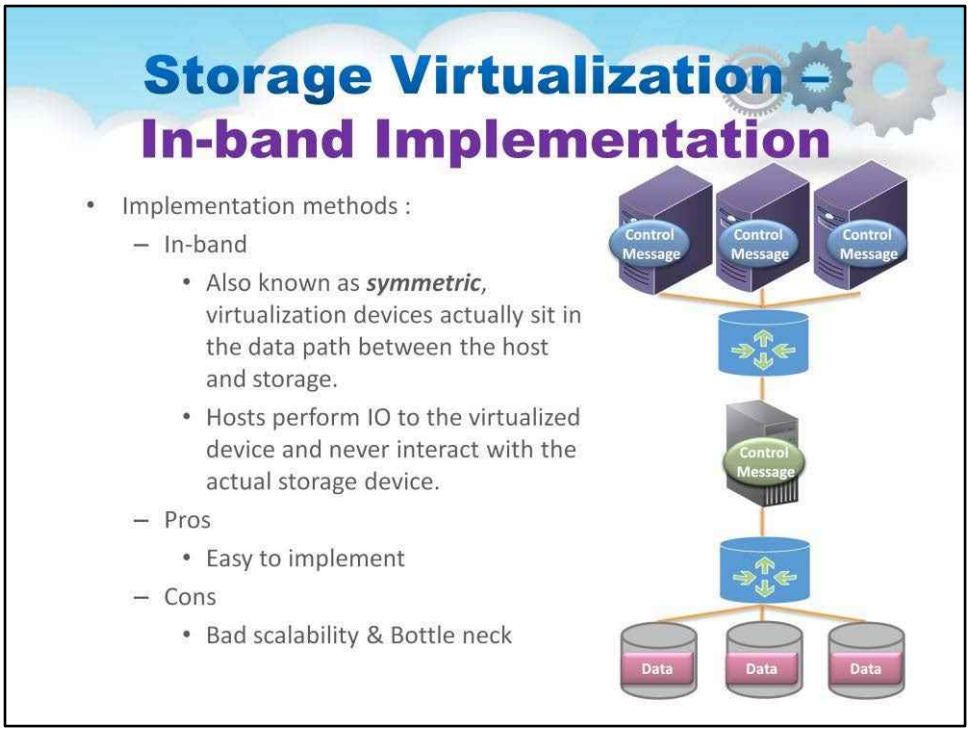
Використання сховища оптимізовано лише на підключених контролерах Реплікація та міграція даних можлива лише на підключених контролерах і пристроях тих самих постачальників

Методи реалізації



Віртуалізацію сховища можна реалізувати кількома способами.

- Внутрішньосмугова віртуалізація. Пристрої віртуалізації, також відомі як симетричні, фактично знаходяться на шляху передачі даних між хостом і сховищем.
- Позасмугова віртуалізація. Пристрої віртуалізації, також відомі як асиметричні, іноді називають серверами метаданих.

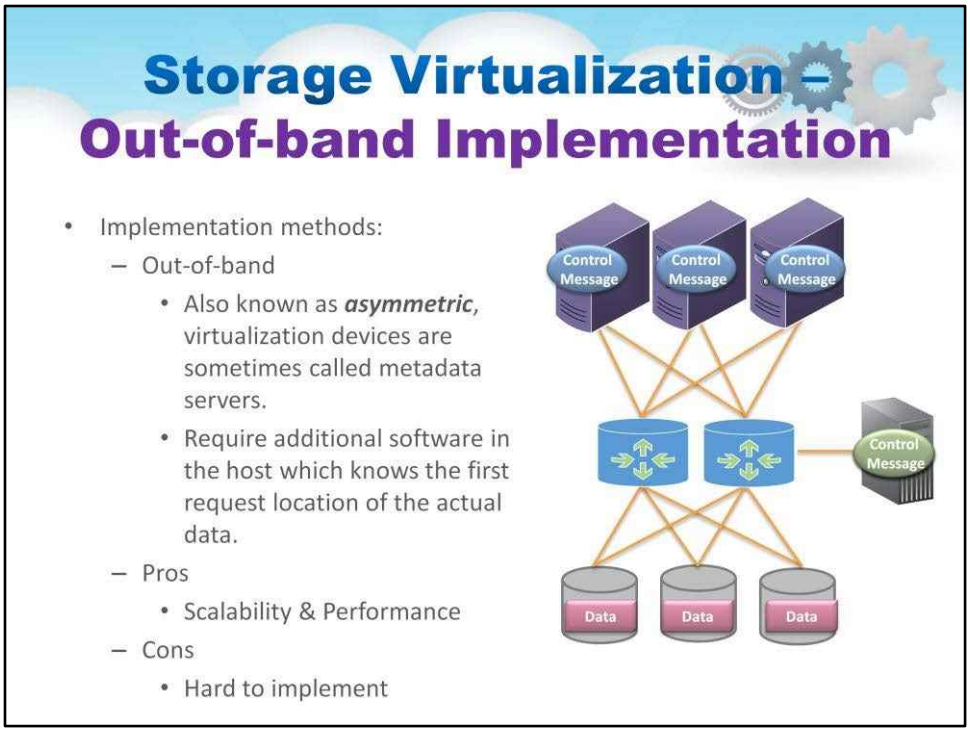


Анімація на цьому слайді показує так звану внутрішньосмугову віртуалізацію, також відому як симетрична, пристрої віртуалізації фактично знаходяться на шляху передачі даних між хостом і сховищем.

Хости виконують введення-виведення на віртуалізований пристрій і ніколи не взаємодіють із фактичним пристроєм зберігання.

Поки легко реалізувати

Він має погану масштабованість і характеристики вузького горловини



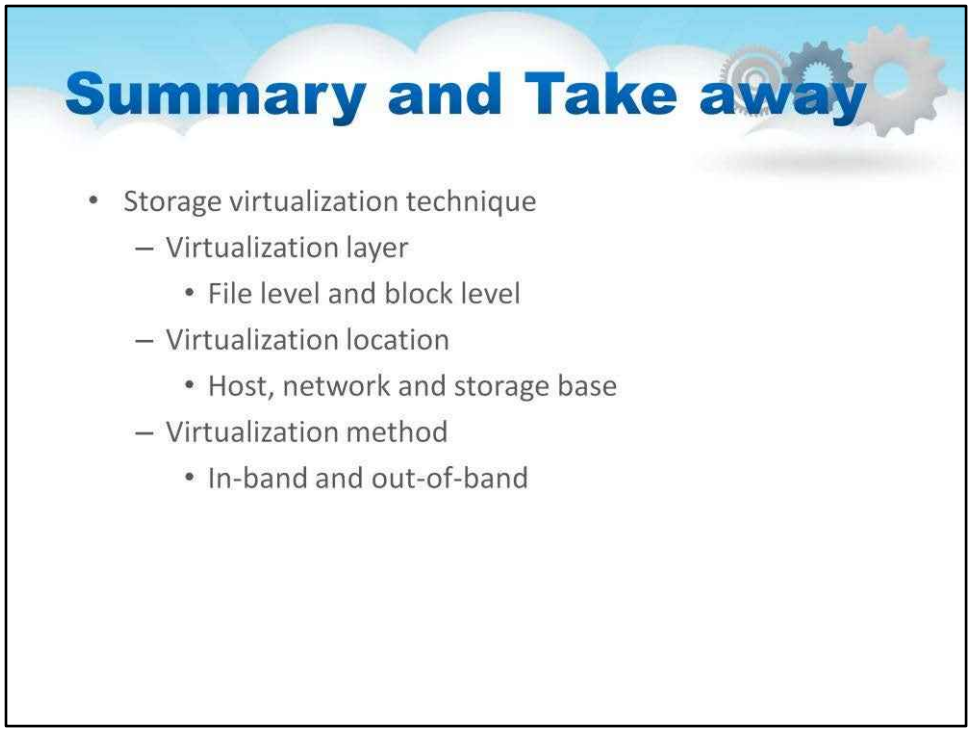
Анімація на цій ілюстрації показує позасмугову віртуалізацію

Пристрої віртуалізації, також відомі як асиметричні, іноді називають серверами метаданих.

Потрібне додаткове програмне забезпечення на хості, яке знає місце першого запиту фактичних даних.

Хоча гарна архітектура для масштабованості та продуктивності

Важко реалізувати

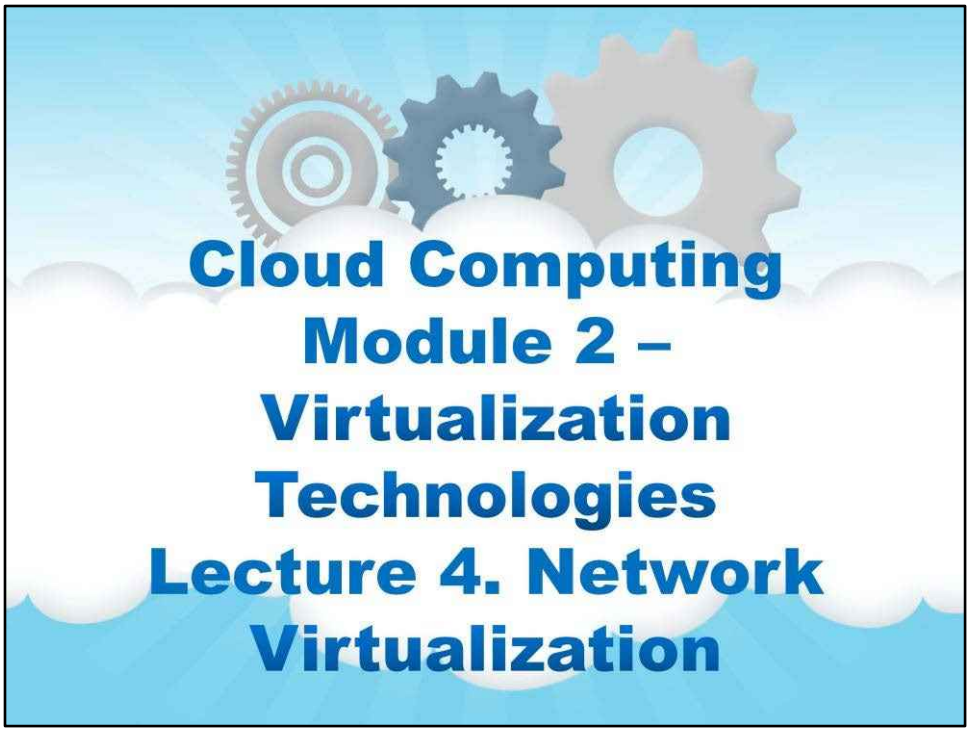


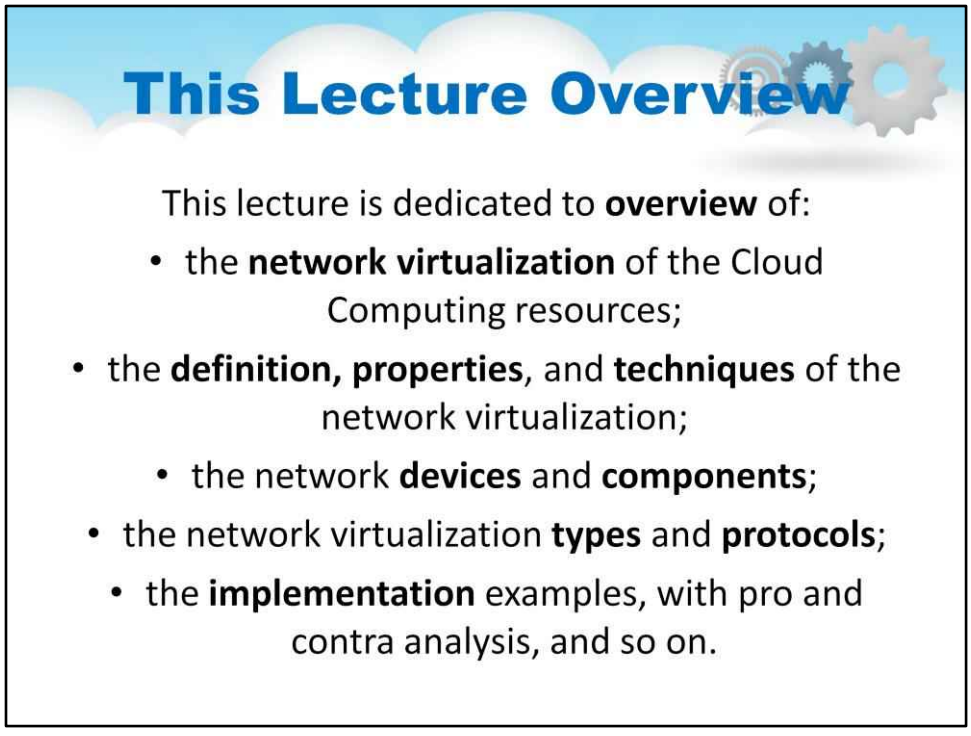
У цьому підручнику досліджено різні способи віртуалізації та впровадження сховища для великих розподілених систем, включаючи хмару.

Ми дослідили примітив зберігання, який був віртуалізований, і побачили, що деякі системи зосереджені на віртуалізації файлів, а деякі системи зосереджені на віртуалізації блоків.

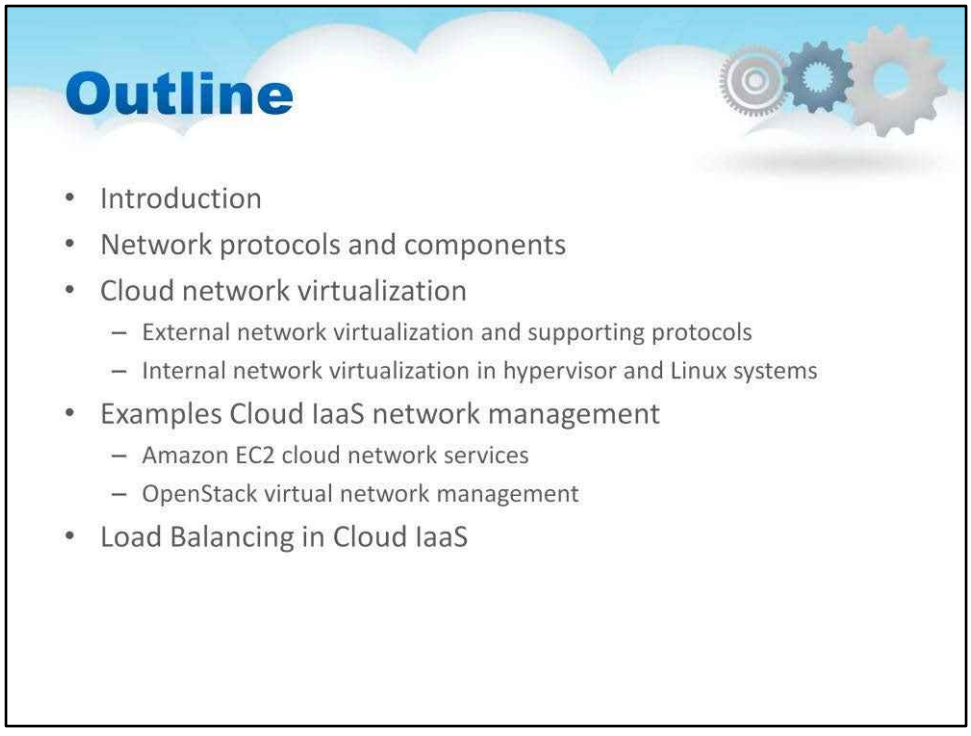
Ми побачили, що функція віртуалізації може працювати в різних місцях архітектури. Він може працювати на хості, у мережі або повністю назад, де знаходяться диски.

Ми побачили, що віртуалізацію можна розмістити «в смузі» операцій зберігання, а для масштабування зазвичай розміщують «поза смугою».


Cloud Computing
Module 2 –
Virtualization
Technologies
Lecture 4. Network
Virtualization



Лекція 4. Віртуалізація мережі



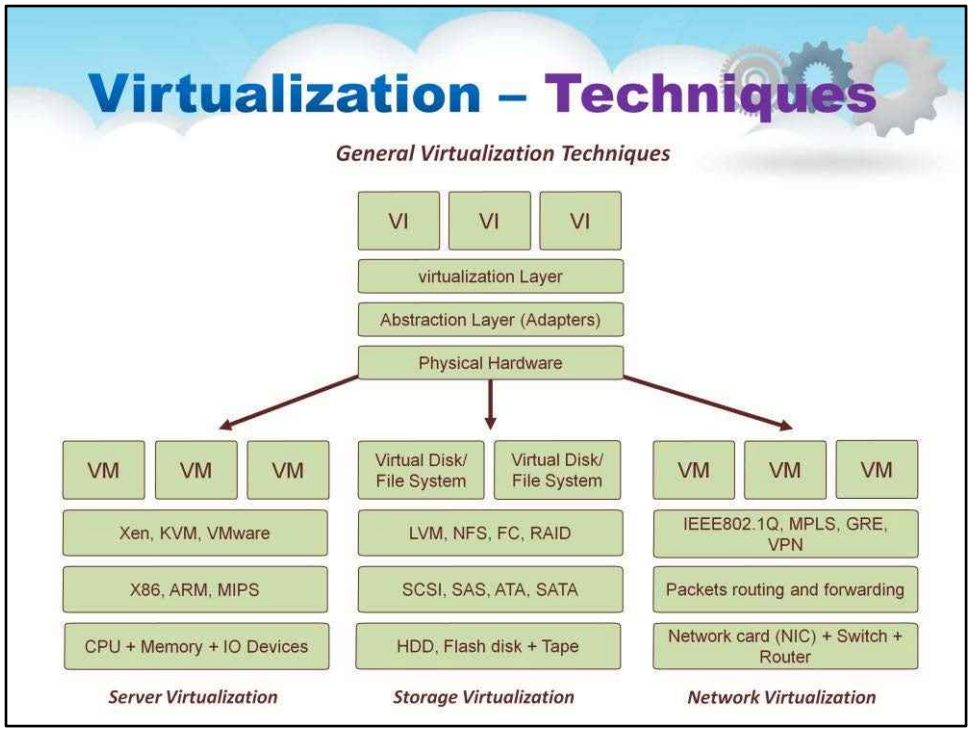
У цьому уроці ми збираємося глибше зануритися в хмарні мережі, зокрема так само, як ми вивчали віртуалізацію обчислень і віртуалізацію сховищ, ми збираємося вивчати віртуалізацію мережі, яка є, мабуть, найцікавішою та найновішою розробкою з трьох.

Ми розглянемо мережеві протоколи та компоненти та їхнє відношення до внутрішніх операційних систем у хмарі.

Ми розглянемо приклади мережевих сервісів у системах IaaS AWS та OpenStack.

Нарешті ми розглянемо тему балансування навантаження, яка є особливою мережевою функцією, надзвичайно корисною в хмарних обчисленнях.

Огляд

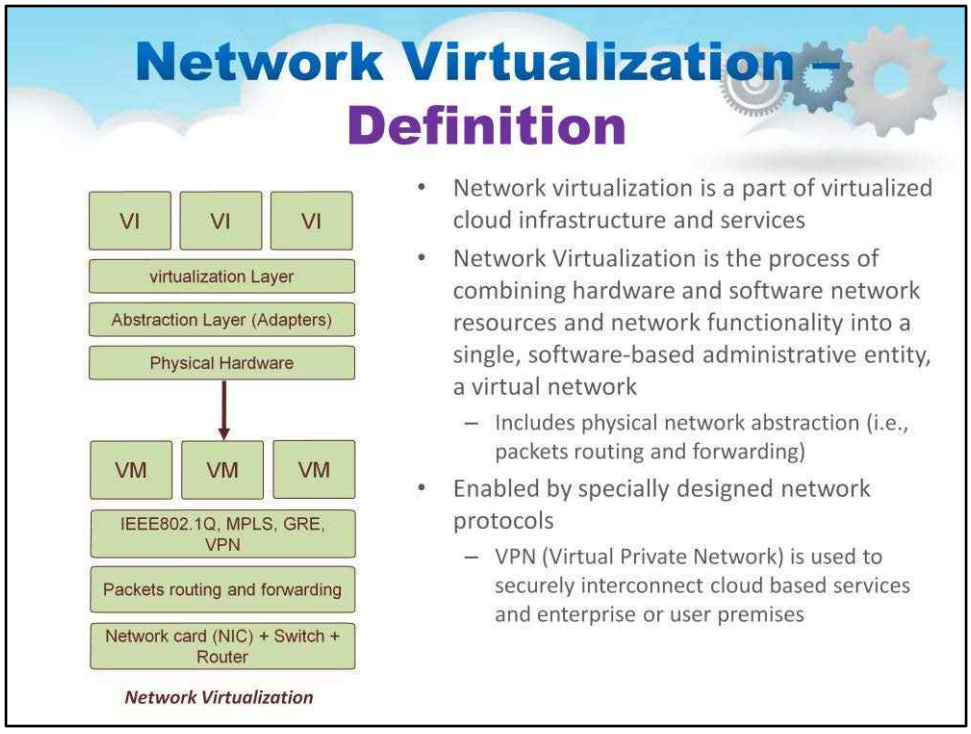


Ця діаграма була показана раніше для прикладу застосування узагальненого підходу віртуалізації до віртуалізації в окремих областях.

У попередньому уроці ми детально розглянули, як гіпервізор зазвичай активує віртуалізацію сервера (або comoute), як показано на середньому рівні частини ілюстрації для віртуалізації сервера.

Також у попередньому уроці ми докладно розглянули, як увімкнути віртуалізацію сховища. Як показано на середньому рівні частини віртуалізації сховища, ця віртуалізація зазвичай реалізується рівнями програмного забезпечення файлової системи, які можуть працювати будь-де на шляху до дисків, іншими словами, на хостах або в мережі в процесорах, де знаходяться диски. . Самі блокові пристрої також зазвичай віртуалізуються та об’єднуються за допомогою різних методів залежно від способу доступу до дисків.

Нарешті, на ілюстрації можна побачити, що існують аналогічні середні рівні у випадку віртуалізації мережі, на якій ми зараз зосередимося.

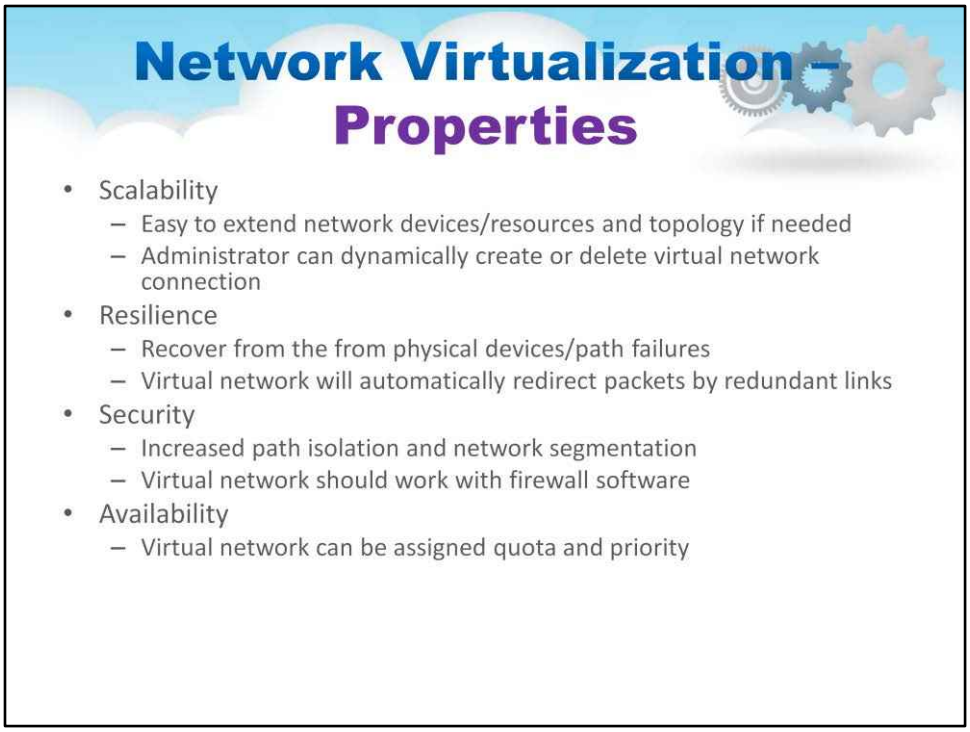


Що таке віртуалізація мережі? Віртуалізація мережі є частиною віртуалізованої хмарної інфраструктури та сервісів.

Звичайно, завжди існує фізична мережа, яка з'єднує всі фізичні частини хмари. Фактично, залежно від того, як хмара була фактично побудована, може існувати кілька мереж, що з'єднують фізичні частини хмари, щоб забезпечити більш високу продуктивність, кращу стійкість до відмов або кілька фізичних шляхів. У хмарному кластері, наприклад, скажімо, фізична хмара охоплює багато, багато стійки, цей/ці сегменти мережі можуть використовувати рівень 2 у стійці та рівень 3 від стійки до стійки; багато технік можливі. Цей фізичний рівень є дуже важливим і вимагає традиційної конфігурації комутаторів і маршрутизаторів.

Віртуалізація мережі створює «логічну» або віртуальну мережу поверх цієї фізичної мережі. Віртуалізація мережі використовує віртуальні комутатори, які присутні в гіпервізорах, щоб нашаровувати від віртуальної машини до віртуальної машини, а також вони накладають поверх фізичної мережі. Віртуалізація мережі — це процес об'єднання апаратних і програмних мережевих ресурсів і мережевих функціональних можливостей в єдину програмну адміністративну сутність — віртуальну мережу.

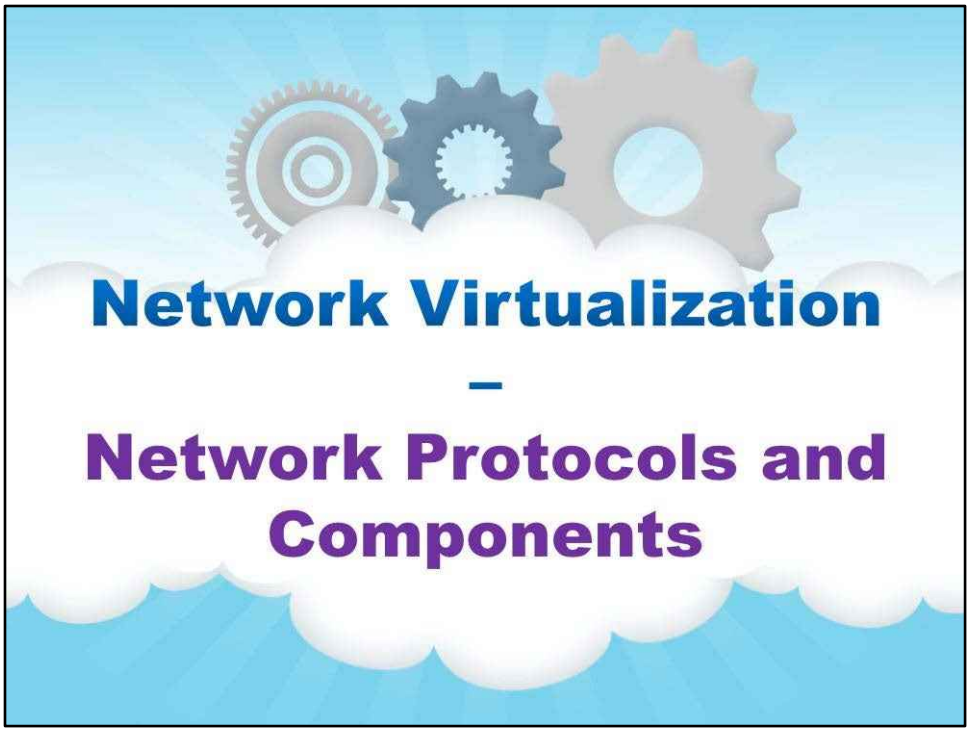
Ця віртуальна мережа підтримується спеціально розробленими мережевими протоколами та спеціальним програмним забезпеченням, яке контролює або визначає ці віртуальні мережі.,



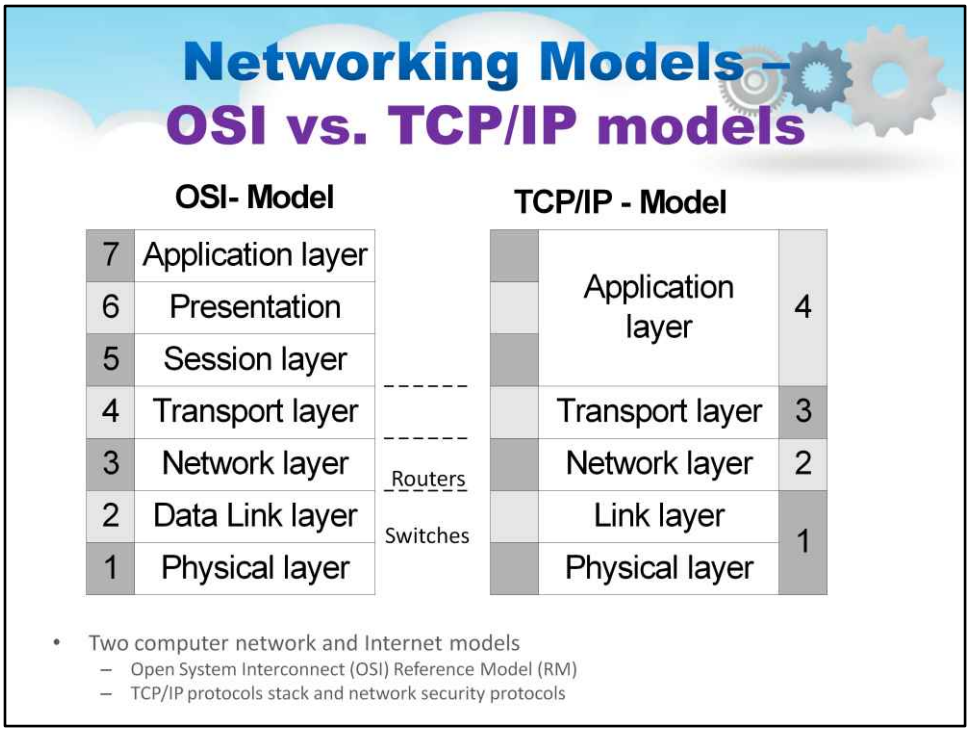
Заразми зосередимося на цьому рівні, який називається віртуалізованою мережею. Поки ми насправді цього не зробили пояснили, як це працює, концептуально ми знаємо, що це програмне забезпечення, яке знаходиться поверх фізичної мережі. Він поширений у хмарі та складається з кількох програмних модулів, які спілкуються один з одним, а також із фізичною мережею нижче.

З огляду на те, що віртуалізація мережі в основному є розподіленою програмною системою, якщо вона реалізована з використанням принципів розподіленого обчислення, таких як усвідомлення стану, використання реплікації, асинхронної роботи компонентів та всіх інших звичайних керівних принципів для побудови розподілених систем, віртуальна мережа демонструватиме характеристики добре розробленої розподіленої системи.

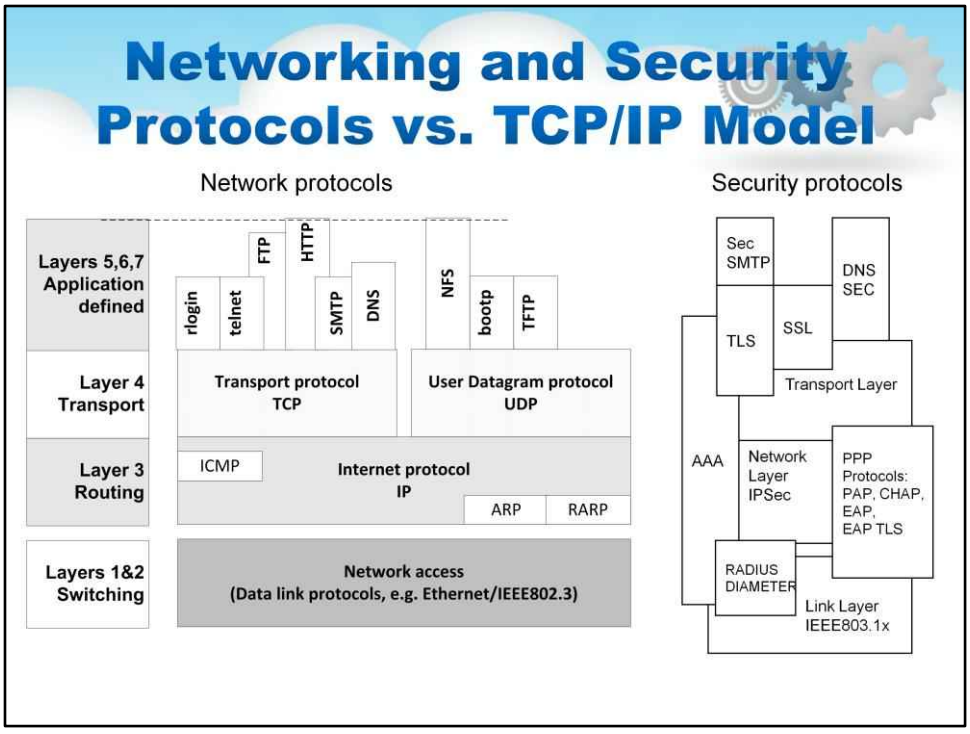
Як показано на слайді, масштабованість, відмовостійкість, безпека та доступність — усі властивості, які демонструватиме віртуальна мережачерез його розподілене програмне забезпечення заснована реалізація.



Мережеві протоколи та компоненти



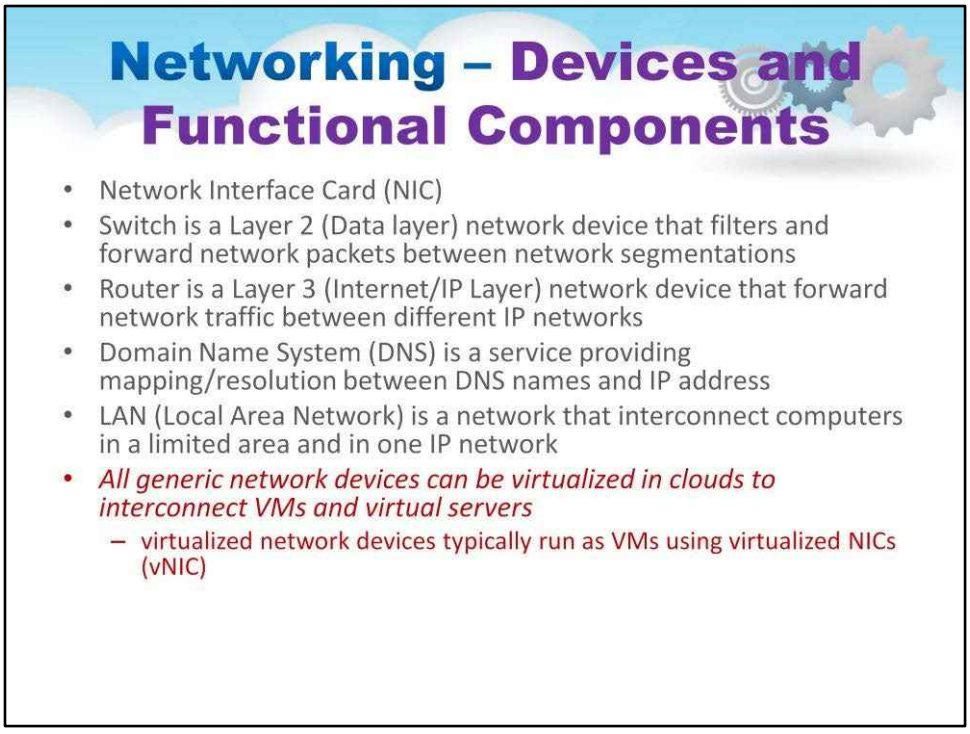
Як довідник, на цьому слайді представлено ілюстрацію, щоб оновити себе з мережевою моделлю OSI та мережевою моделлю TCP/IP. Ми будемо звертатися до цих шарів на постійній основі



Крім того, для довідки на цьому слайді представлено дві ілюстрації.

Перша ілюстрація показує загальні мережеві протоколи на основі IP і те, як вони відповідають рівням у моделі TCP/IP.

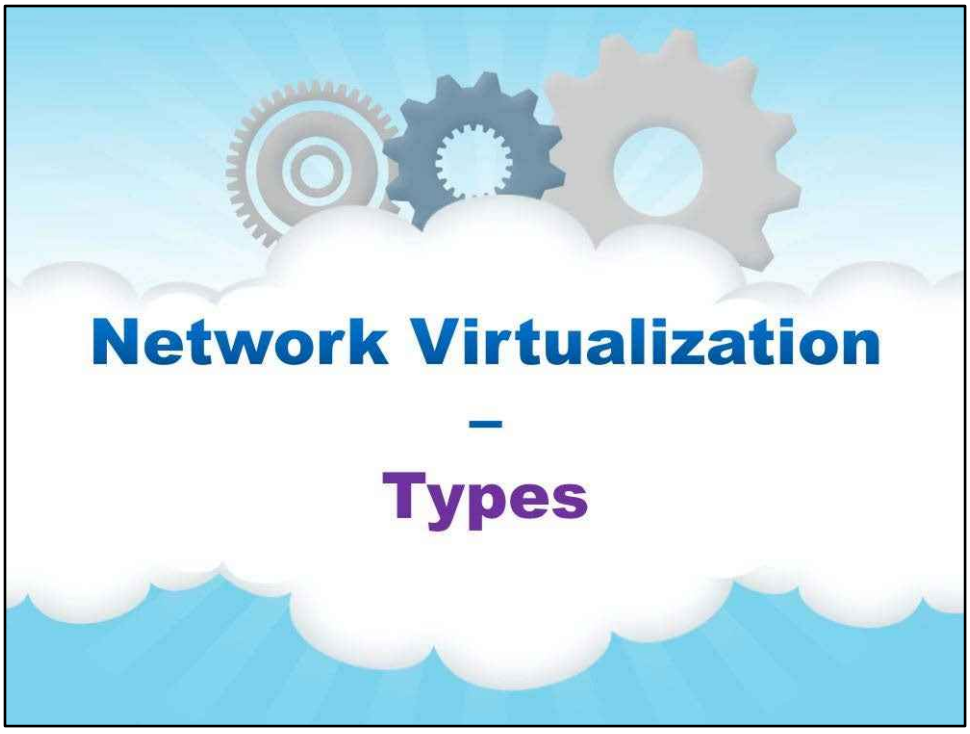
Друга ілюстрація показує загальні протоколи безпеки та те, як вони також відповідають рівням у моделі TCP/IP.



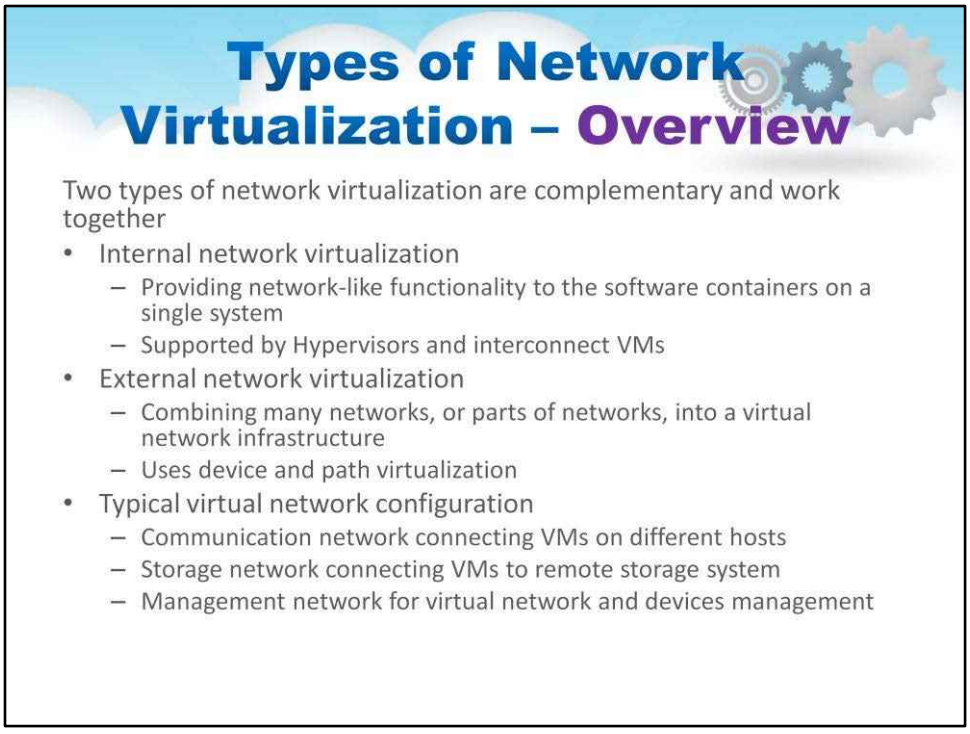
Як обговорювалося раніше, є фізичні компоненти мережі, які підключаються та налаштовуються. Це всі види мережевих пристроїв, від карт мережевого інтерфейсу (NIC) у серверах до комутаторів і маршрутизаторів, а також інфраструктура, від якої залежить мережа, наприклад система доменних імен (DNS) і, можливо, IP-адреса Система керування (IPAM) (наприклад, DHCP).

Поверх цього є програмний рівень, який реалізує віртуальну мережу. Віртуальна мережа включає віртуальні комутатори та віртуальні NIC (vNIC), які є частиною системи гіпервізора. Він також міститиме віртуальні комутатори (vSwitch), віртуальні маршрутизатори (vRouters) і навіть модулі, які постачають DNA та IPAM на рівні віртуалізованої мережі.

Іншими словами, усі загальні мережеві пристрої можуть бути у віртуалізованих формах і використовуватися в хмарах для з'єднання віртуальних машин і віртуальних серверів.



Типи



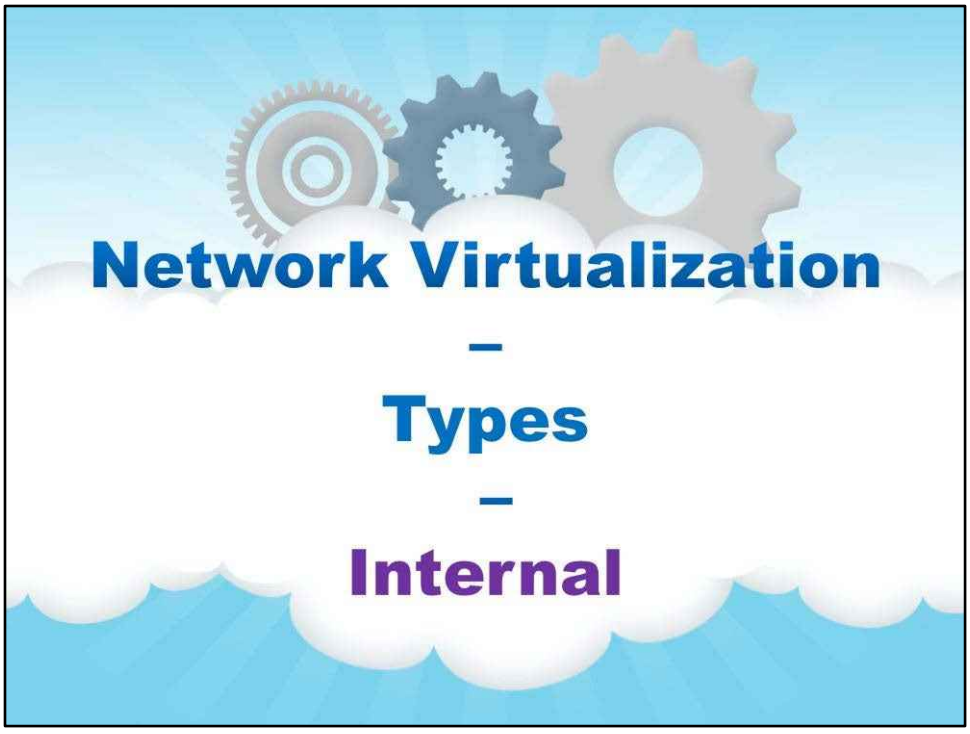
На попередньому слайді ми розрізняли фізичну та віртуальну мережі.

Насправді віртуальна мережа має два типи, як ви вже могли з’ясувати.

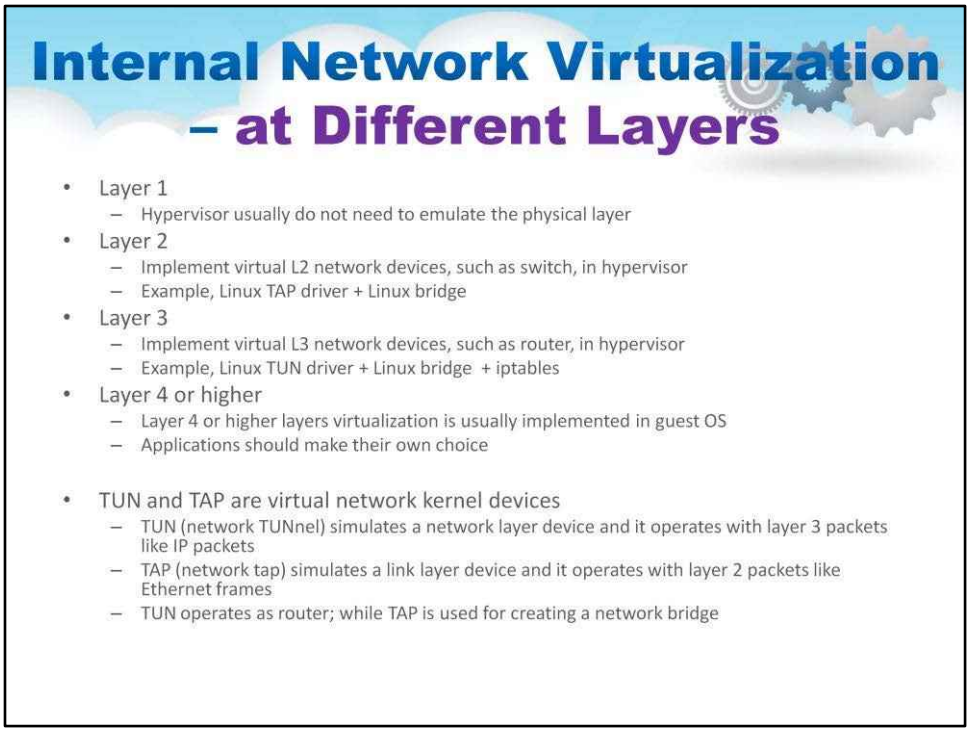
Перший тип називається віртуалізацією внутрішньої мережі, це vNIC і vSwitches, які підтримуються гіпервізорами та віртуальними машинами для з’єднання. Це надання мережевої функціональності програмним контейнерам в одній системі

Другий тип називається віртуалізацією зовнішньої мережі. Тут ми виходимо за межі окремого сервера та об’єднуємо багато мереж або частин мереж у віртуальну мережеву інфраструктуру. Це використовує віртуалізацію пристрою та шляху.

На слайді ми розглянемо типову конфігурацію віртуальної мережі.



внутрішній



Давайте тепер розглянемо віртуалізацію внутрішньої мережі на різних рівнях.

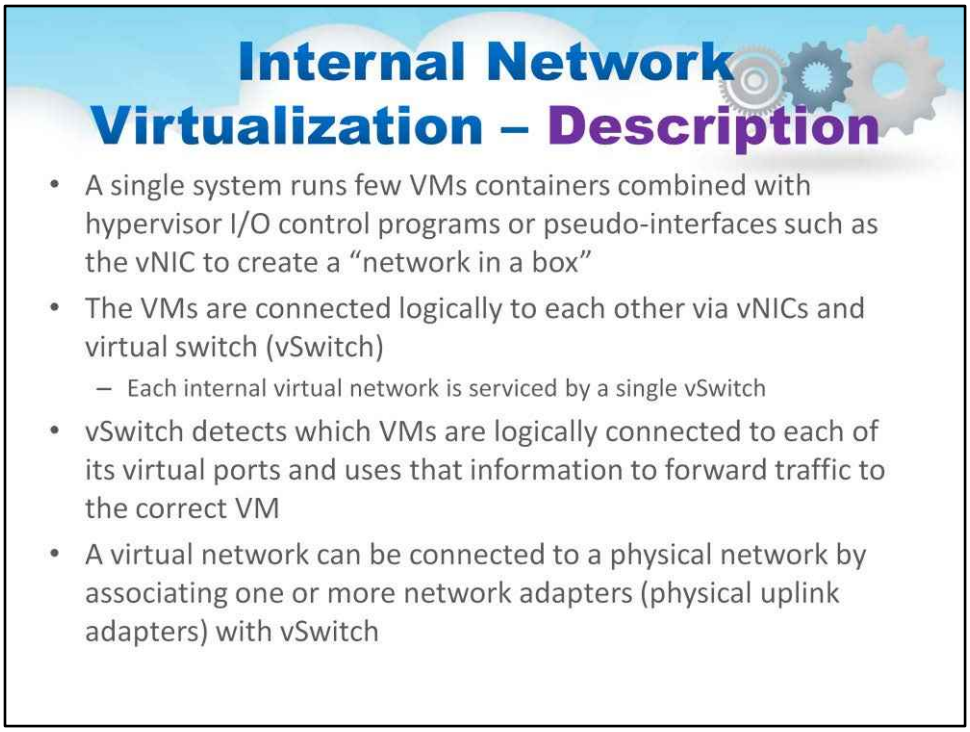
Як показано на слайді, рівень 1 не потребує емуляції, оскільки фізичний рівень не емулюється гіпервізором, драйвери абстрагуються від цього.

Рівень 2 – це місце, де програмне забезпечення вперше підключається до мережі, це місце, де vNIC відкриває Ethernet у гіпервізорі, або де vSwitch використовується для з'єднання або комутації рівня 2.

Рівень 3 Реалізує віртуальні мережеві пристрої L3, такі як маршрутизатор, у гіпервізорі

Віртуалізація рівня 4 або вище зазвичай реалізується в гостьовій ОС

У Linux можна побачити пристрої ядра віртуальної мережі TUN і TAP, як пояснено, TUN працює як маршрутизатор; тоді як TAP використовується для створення мережевого мосту



Тепер давайте детально поглянемо на віртуалізацію внутрішньої мережі

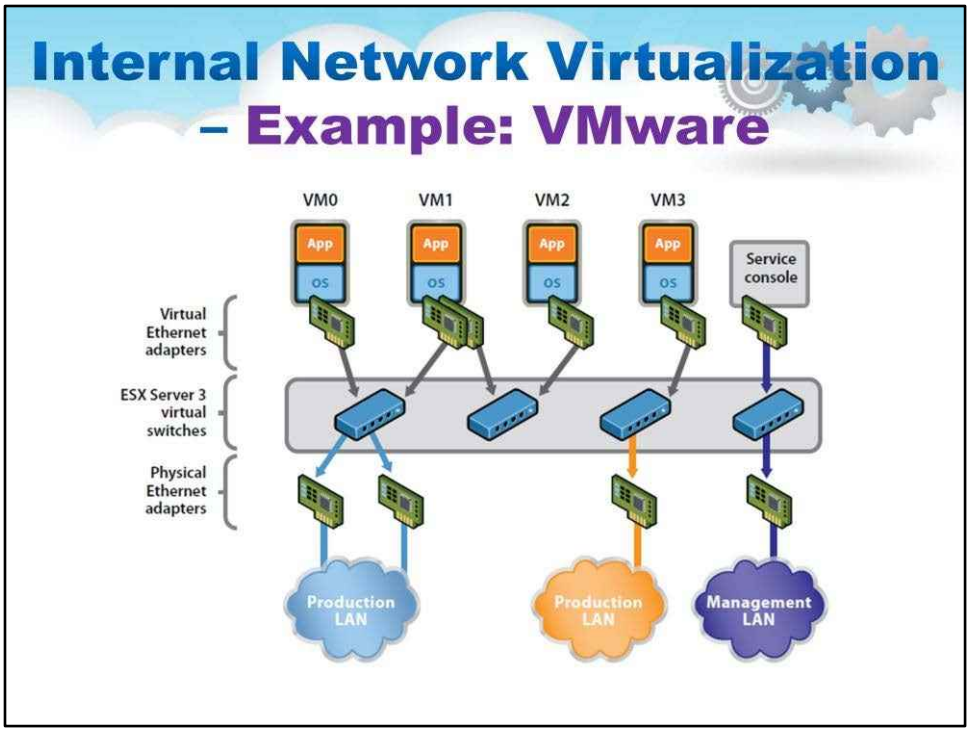
В основі цього полягає те, що кожна окрема система, яка запускає декілька контейнерів віртуальних машин, уже поєднана з програмами керування введенням/виведенням гіпервізора або псевдоінтерфейсами, такими як vNICстворити «мережу в коробці»

Віртуальні машини логічно підключені одна до одної через vNIC і віртуальний комутатор (vSwitch)

Кожна внутрішня віртуальна мережа обслуговується одним vSwitch

vSwitch визначає, які віртуальні машини логічно підключені до кожного з його віртуальних портів, і використовує цю інформацію для перенаправлення трафіку на правильну віртуальну машину.

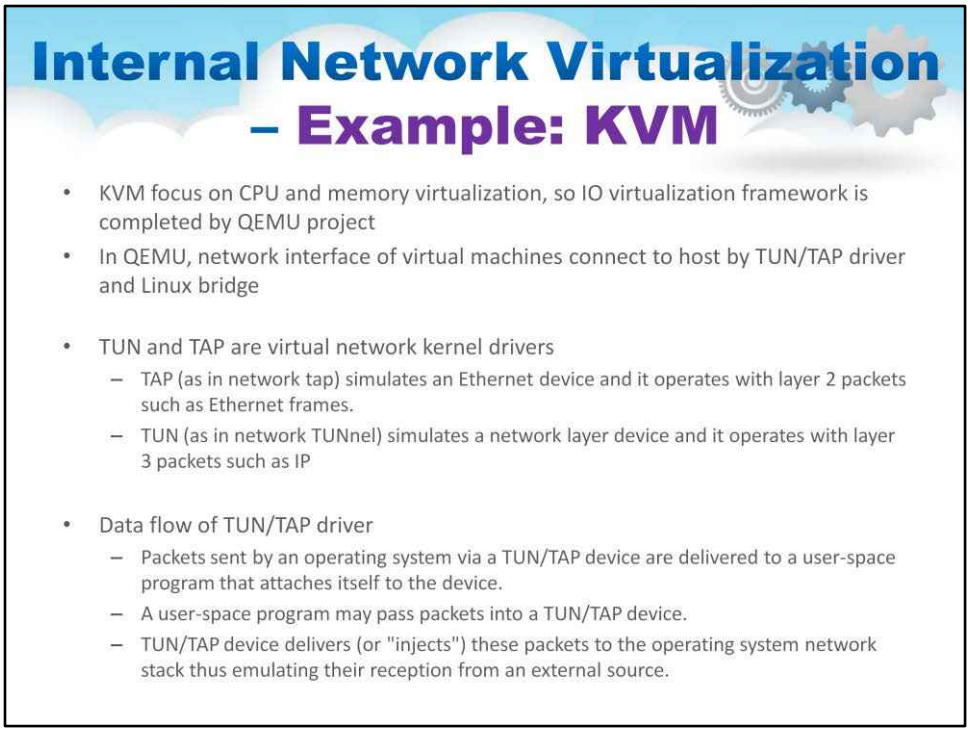
Віртуальну мережу можна підключити до фізичної мережі, зв’язавши один або кілька мережевих адаптерів (фізичних адаптерів висхідної лінії зв’язку) з vSwitch



Цей слайд чудово ілюструє приклад віртуалізації внутрішньої мережі від VMware.

Можна побачити vNIC і віртуальні комутатори

One підключає віртуальні машини до різних мереж через vSwitch і vNIC

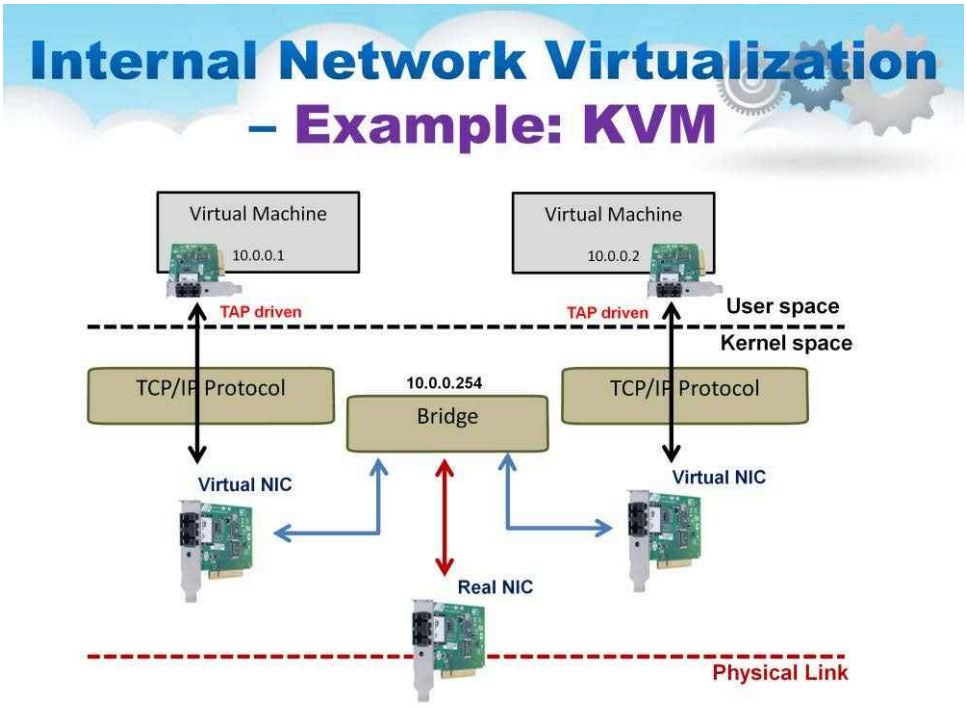


Тепер розглянемо віртуалізацію внутрішньої мережі в KVM

KVM зосереджується на віртуалізації процесора та пам’яті, тому структура віртуалізації вводу-виводу завершена проектом QEMU

У QEMU мережевий інтерфейс віртуальних машин підключається до хоста за допомогою драйвера TUN/TAP і мосту Linux

Робота TUN і TAP описана на слайді.

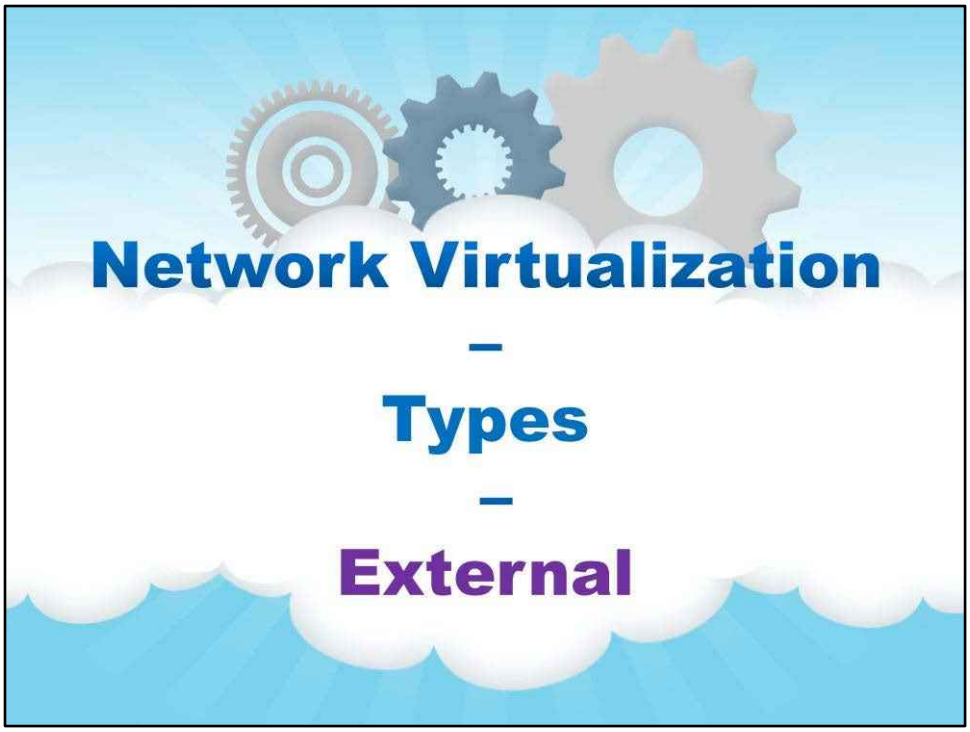


У мості KVM/Linux, як можна побачити на ілюстрації на цьому слайді, міст використовується для з’єднання vNIC із реальною фізичною мережевою картою,

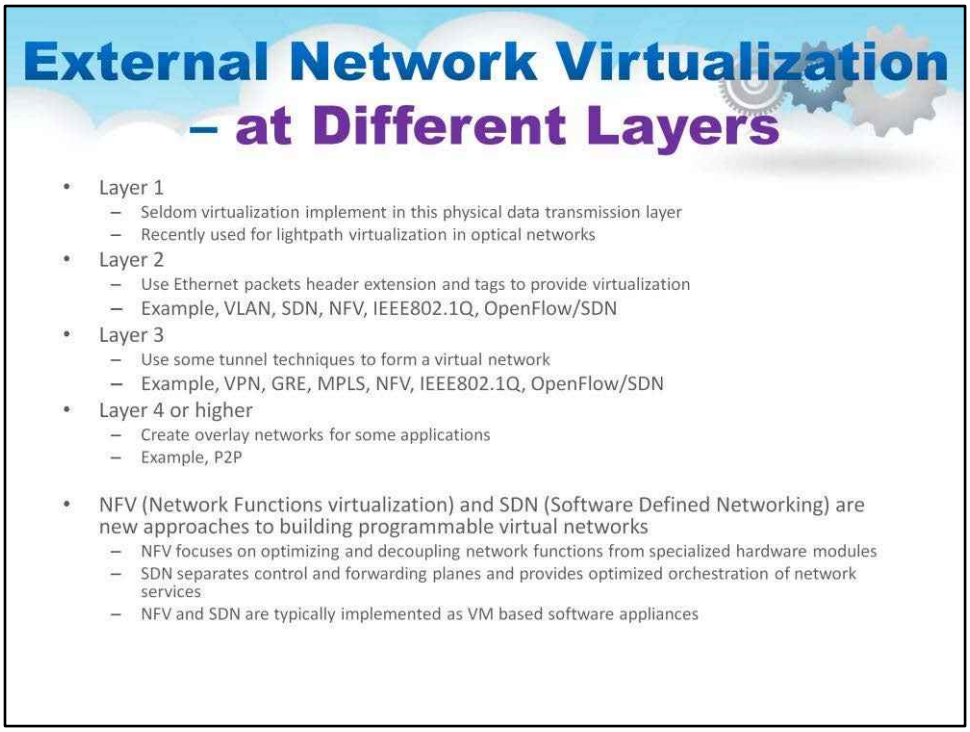
Мост — це метод пересилання, який використовується в комп’ютерних мережах з комутацією пакетів. На відміну від маршрутизації, з’єднання не робить припущень про те, де в мережі розташована конкретна адреса.

З’єднання мостів залежить від затоплення та перевірки адрес джерела в заголовках отриманих пакетів для визначення місцезнаходження невідомих пристроїв.

Мост з’єднує кілька сегментів мережі на канальному рівні (рівні 2) моделі OSI.



зовнішній



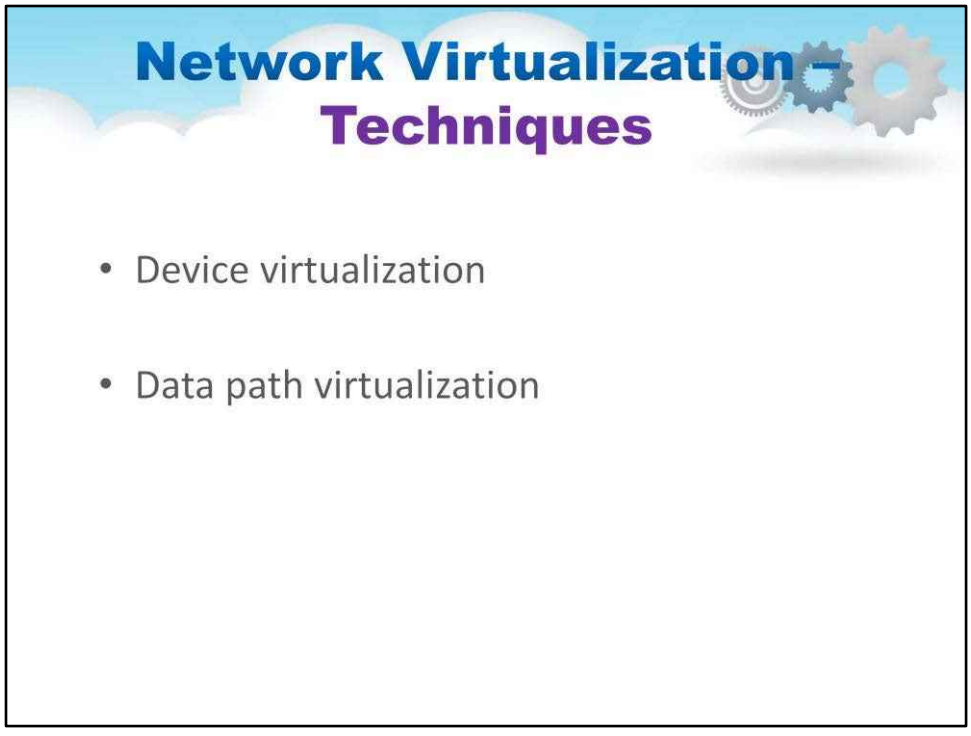
Ні, давайте розглянемо віртуалізацію зовнішньої мережі на різних рівнях

Знову ж таки, до рівня 1 рідко звертаються за програмним забезпеченням. Останнім часом на рівні 2 з’явилося багато технологій віртуалізації, ми приділимо більше часу віртуалізації рівня 2, наприклад VLAN, SDN, NFV, IEEE802.1Q, OpenFlow/SDN.

Усі методи рівня 2 використовують розширення заголовків пакетів Ethernet і теги для передачі по мережі інформації про віртуалізацію, і деякі з цих угод стандартизовано. Це активна зона.

Рівень 3 знаходиться в тій же ситуації, багато розробок, як правило, з використанням деяких тунельних методів для формування віртуальної мережі -

Наприклад, VPN, GRE, MPLS, NFV, IEEE802.1Q, OpenFlow/SDN.

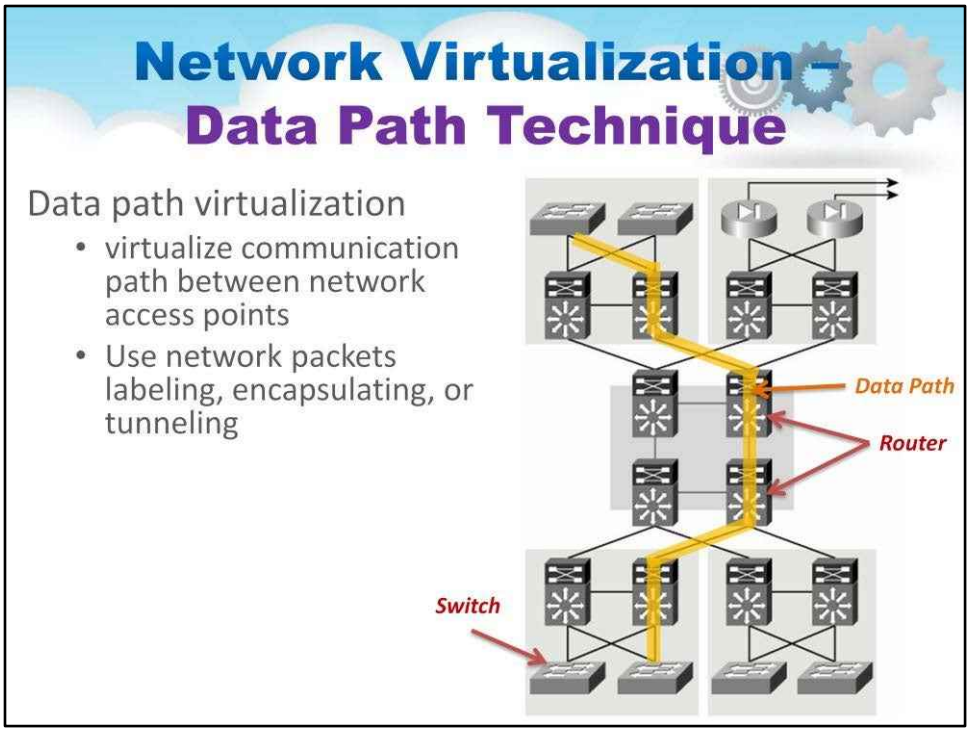


МережаВіртуалізація зазвичай використовує дві основні техніки для виконання своєї роботи це віртуалізація пристрою та шляху

Вони розміщують мережеву віртуалізацію в двох місцях: одне на пристрої, інше на шляху трафіку,

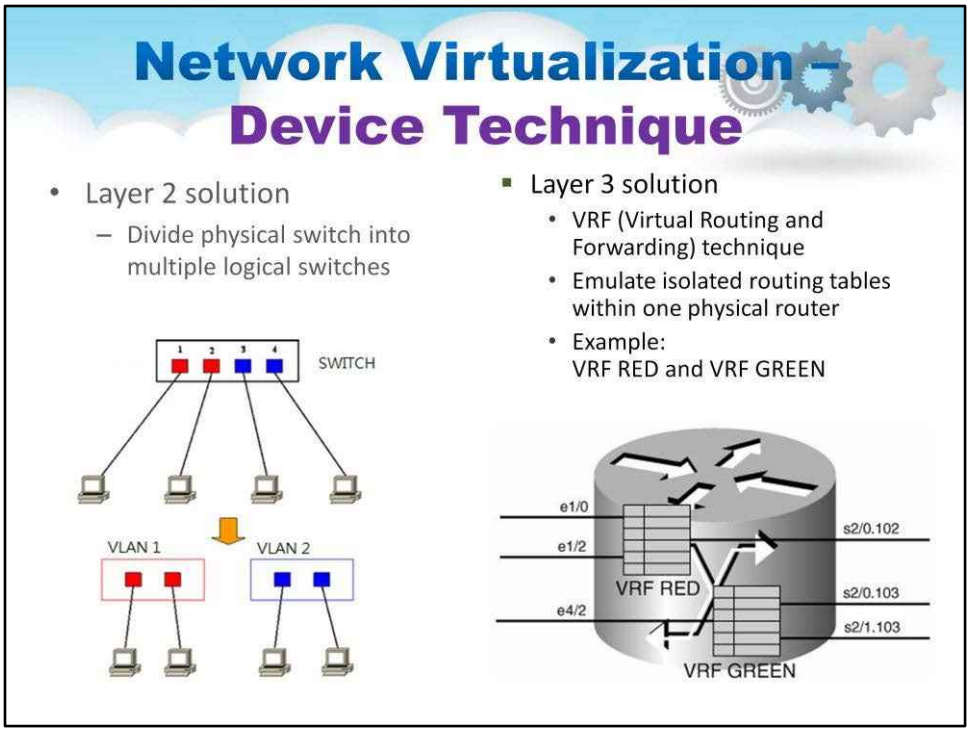
- Віртуалізація пристроїв віртуалізує фізичні пристрої в мережі
- Віртуалізація шляху даних віртуалізує шлях зв’язку між точками доступу до мережі

На ілюстрації можна побачити, як повну віртуалізовану мережу можна побудувати за допомогою віртуалізації пристрою та шляху



Віртуалізація шляху даних віртуалізує шлях зв’язку між точками доступу до мережі

На ілюстрації можна побачити, як повну віртуалізовану мережу можна побудувати за допомогою віртуалізації пристрою та шляху



Віртуалізація пристрою виконується по-різному для кожного типу мережевого пристрою, який потрібно створити

Рішення рівня 2 (рівень комутатора) створює кілька комутаторів, розділяючи трафік між комутаторами. У програмній реалізації це кілька vSwitches. У апаратному пристрої це робиться за допомогою ASIC і називається VLAN для віртуальної локальної мережі.

Рішення рівня 3 (рівень маршрутизатора) використовує таблиці маршрутизації, як можна було б підозрювати, створюючи окремі таблиці маршрутизації або, точніше, домени маршрутизації для кожної мережі рівня 3,

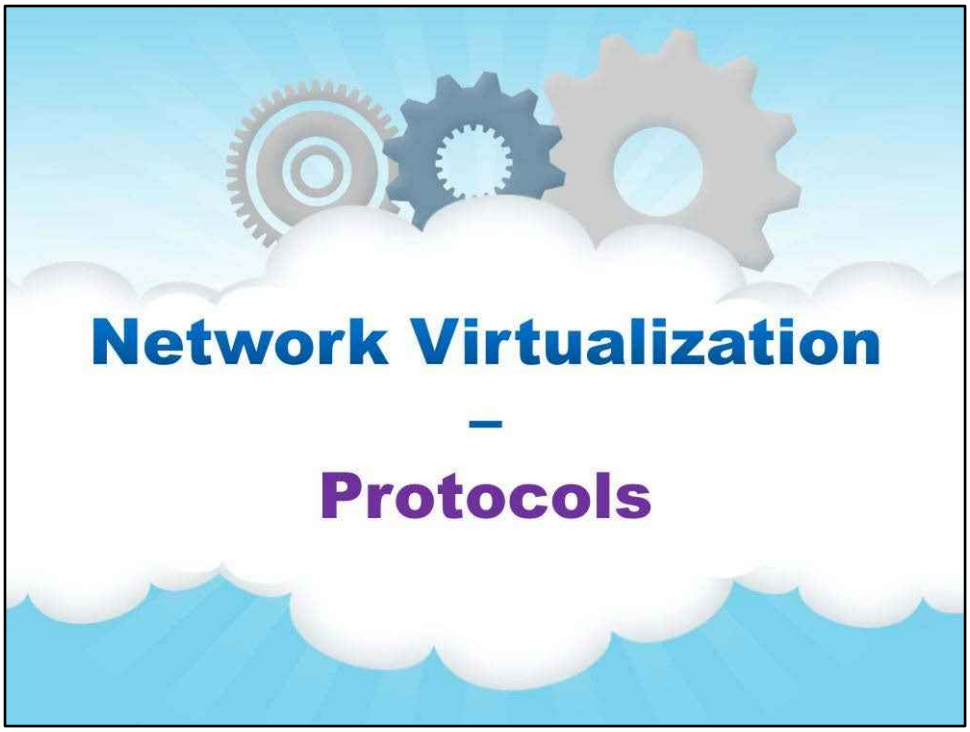
У програмній реалізації VRF (віртуальна маршрутизація та

Техніка пересилання) використовується в апаратному забезпеченні

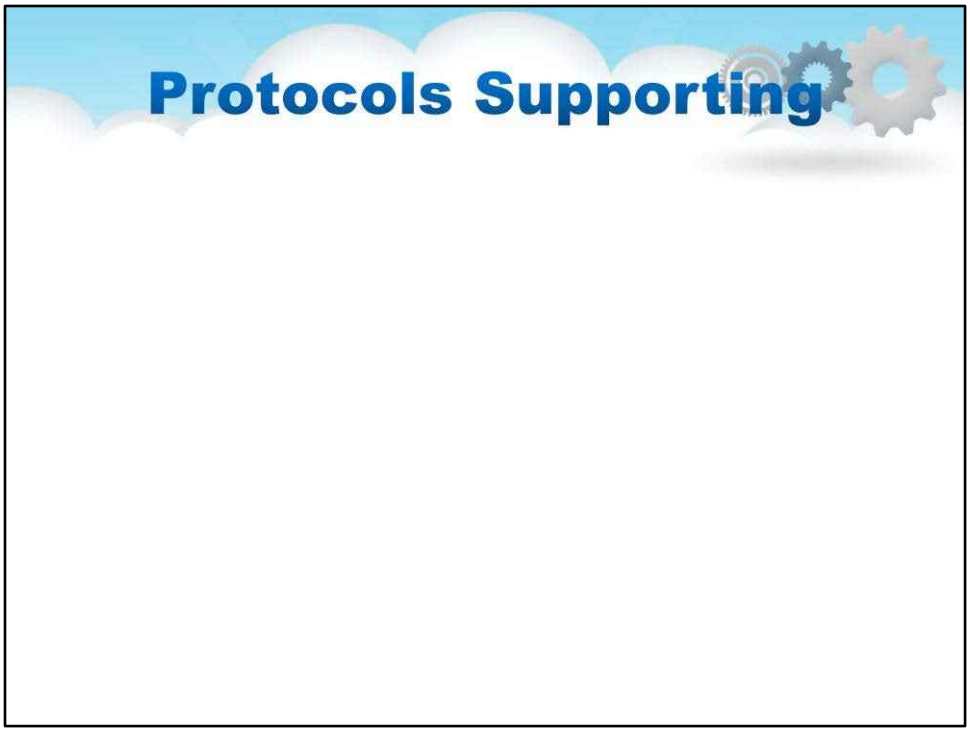
пристрій маршрутизатор створює окремі домени рівня 3

в межах одного маршрутизатора (з окремою маршрутизацією

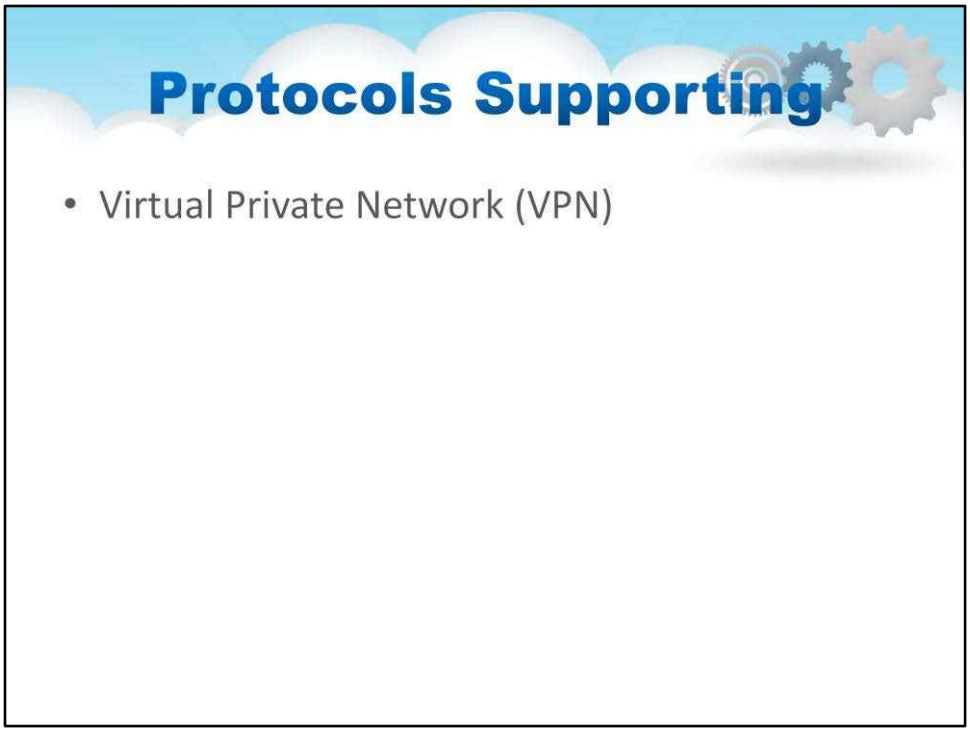
протоколи, процесори маршрутизації та таблиці).



Протоколи

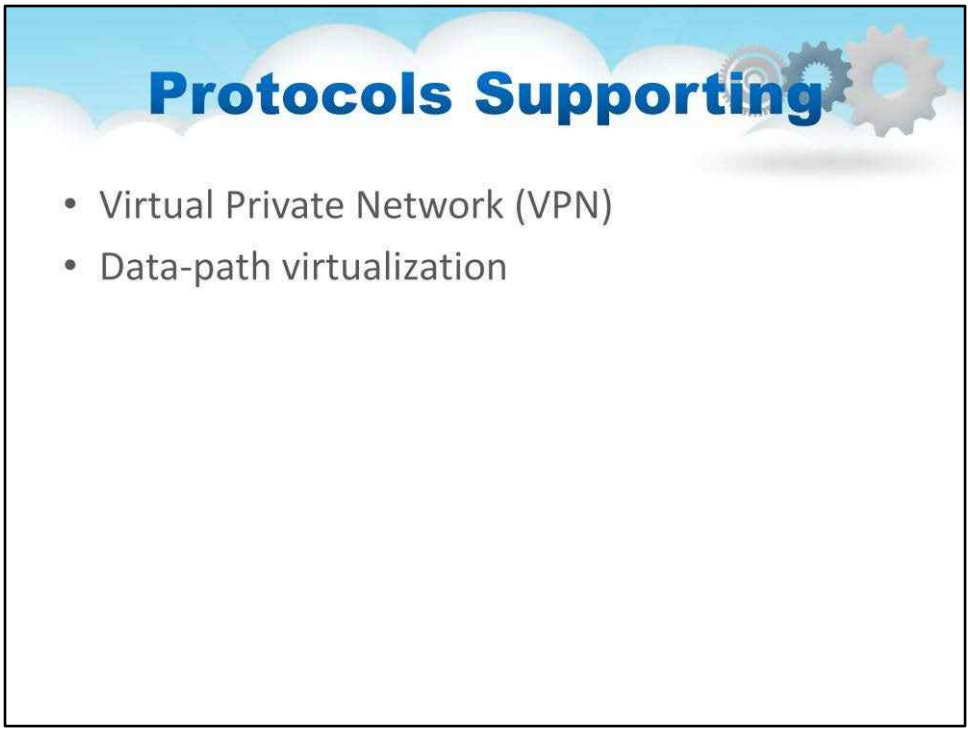


Протоколи підтримки віртуалізації мережі

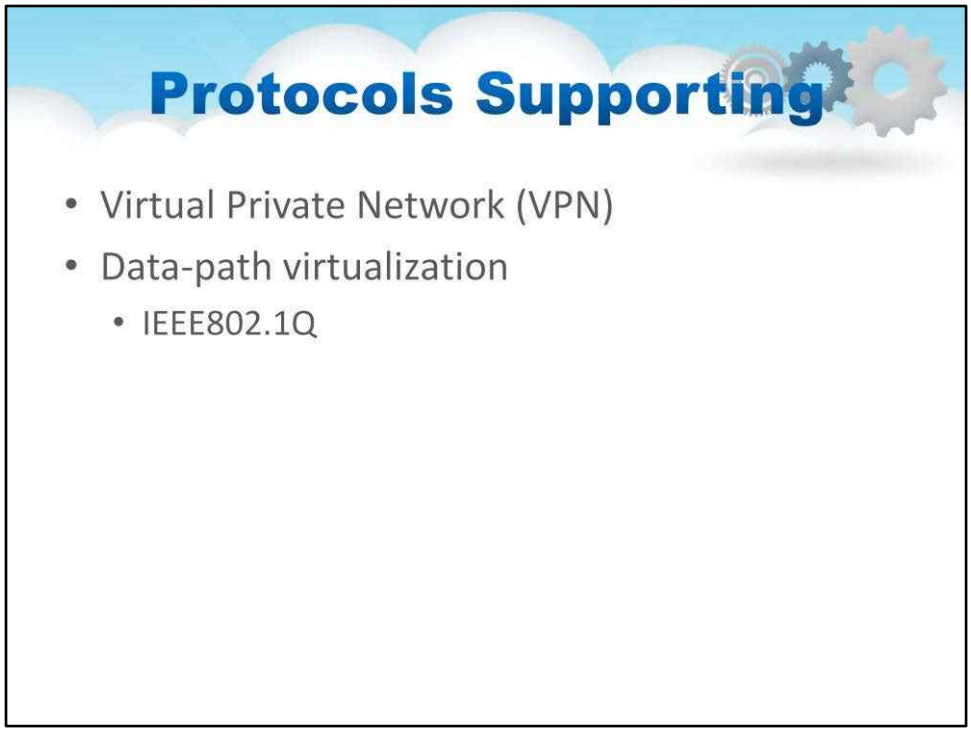


Віртуальна приватна мережа (VPN)

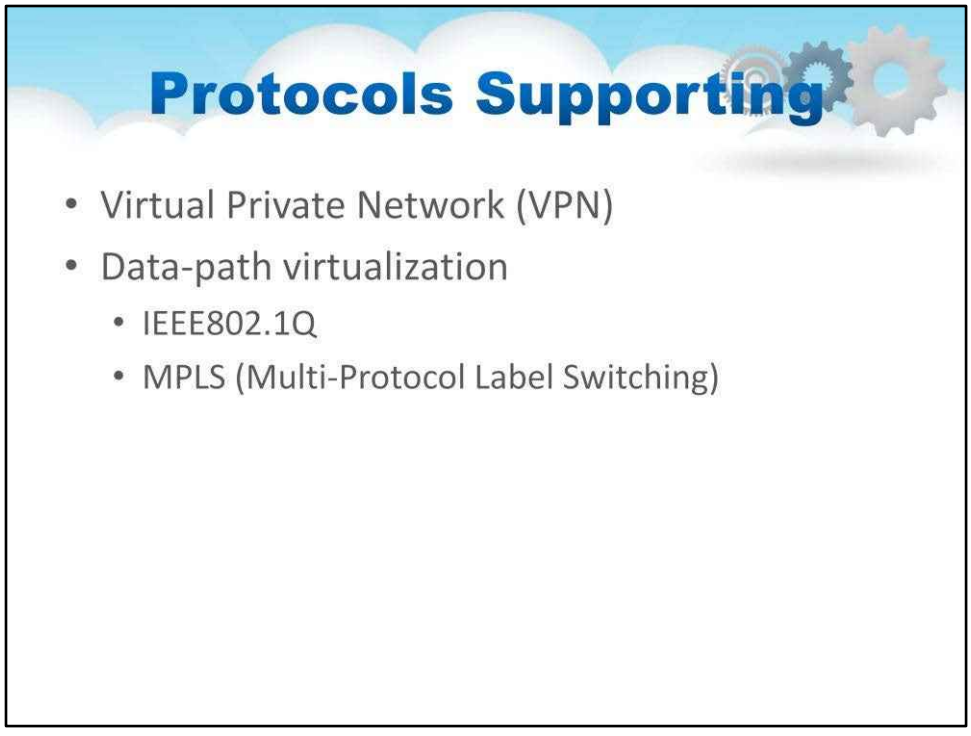
- **IPSec**заснований на протоколі тунелювання, що з’єднує дві мережі
- Найбільш поширене рішення для з’єднання віддалених офісів і користувачів у роумінгу



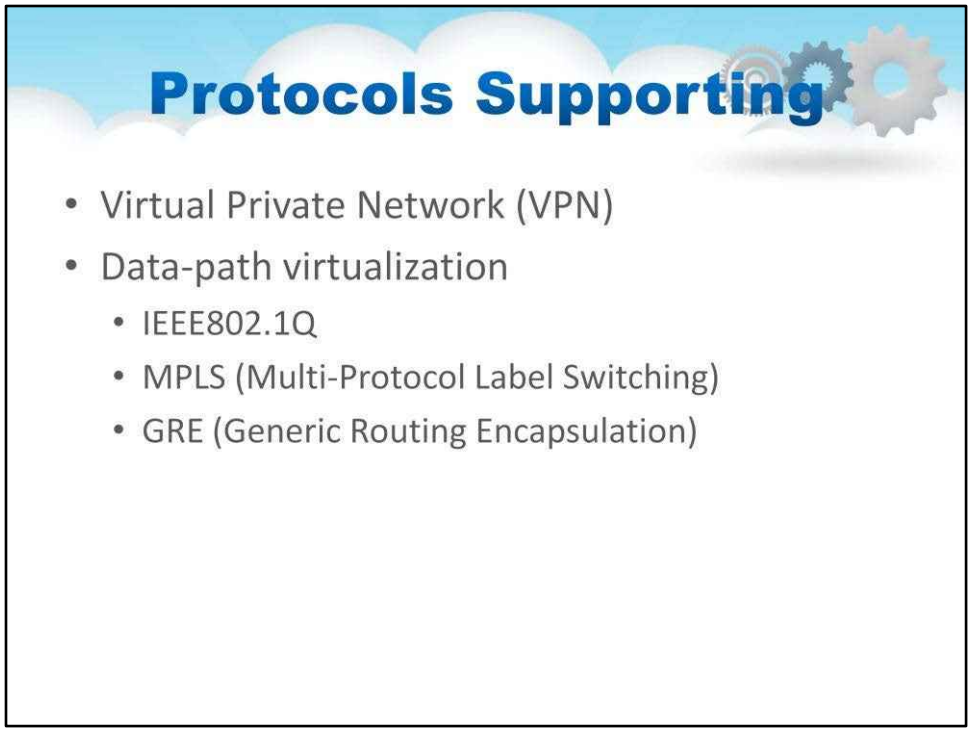
Віртуалізація шляху даних реалізована на основі спеціальних мережевих протоколів, таких як



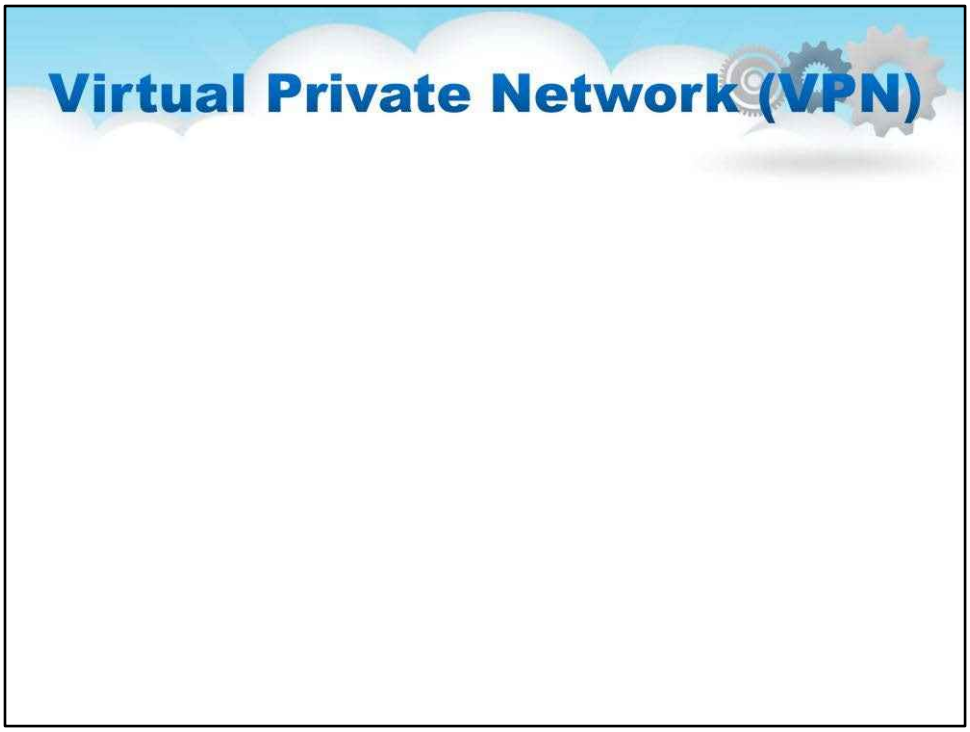
IEEE802.1Q — це протокол віртуалізації рівня 2, який реалізує віртуалізацію шляху передачі даних і дозволяє створювати віртуальні локальні мережі.

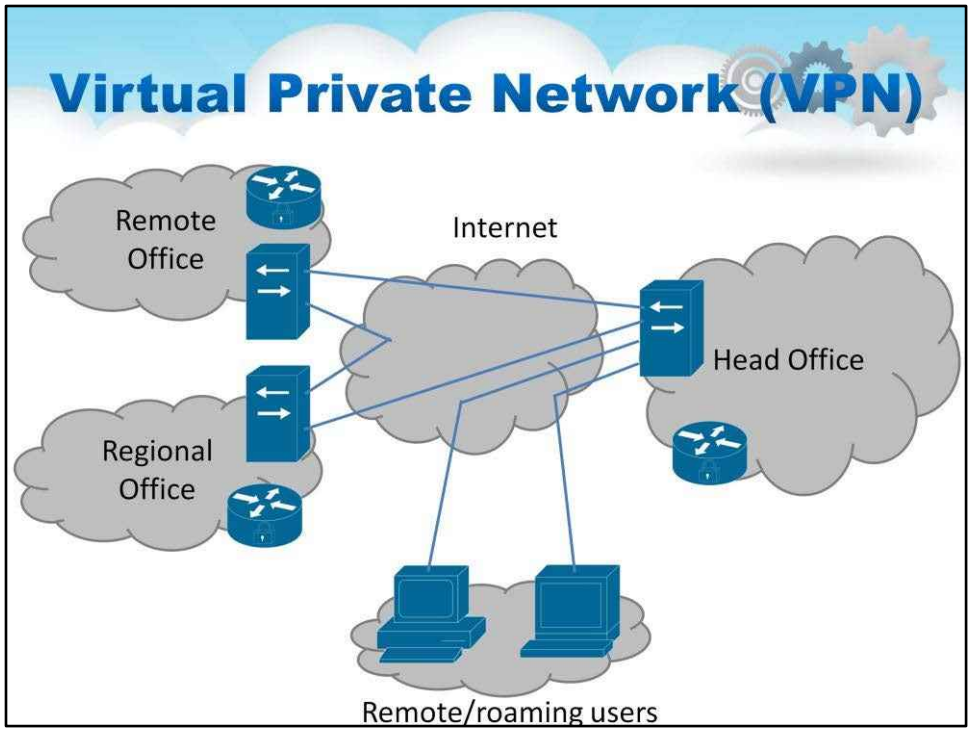


MPLS (Multi-Protocol Label Switching) — це протокол віртуалізації мережевого шляху рівня 3, який працює на рівні мережевих маршрутизаторів і комутаторів;

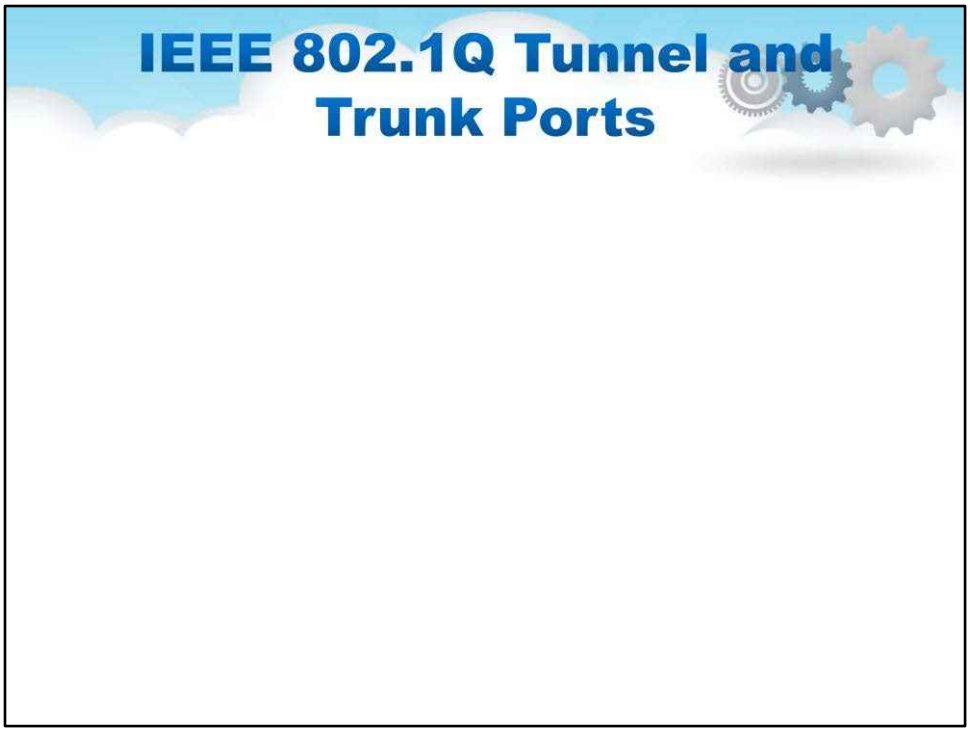


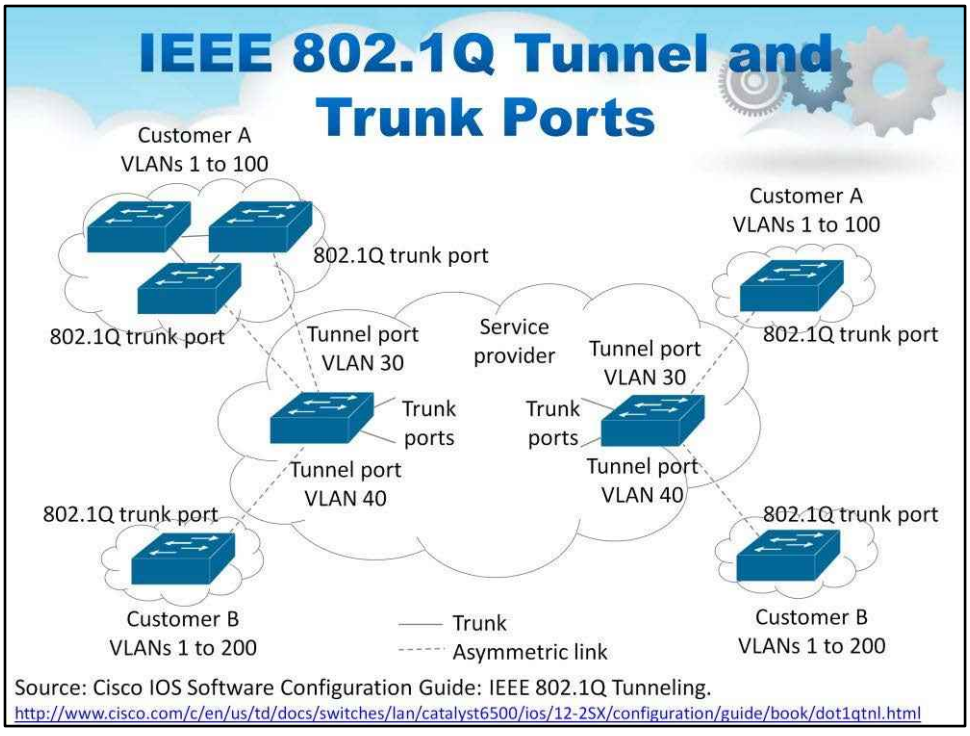
GRE (Generic Routing Encapsulation): реалізує віртуалізацію серед різноманітних мереж за допомогою техніки тунелювання.


Virtual Private Network (VPN)



- Віртуальна приватна мережа (VPN), яка використовується для розширення приватної мережі через загальнодоступний Інтернет до віддалених місць
  - Використовує наскрізний безпечний IP-протокол (IPSec)
- Спочатку VPN має один домашній/головний офіс
- Нові технології дозволяють багатодомну VPN

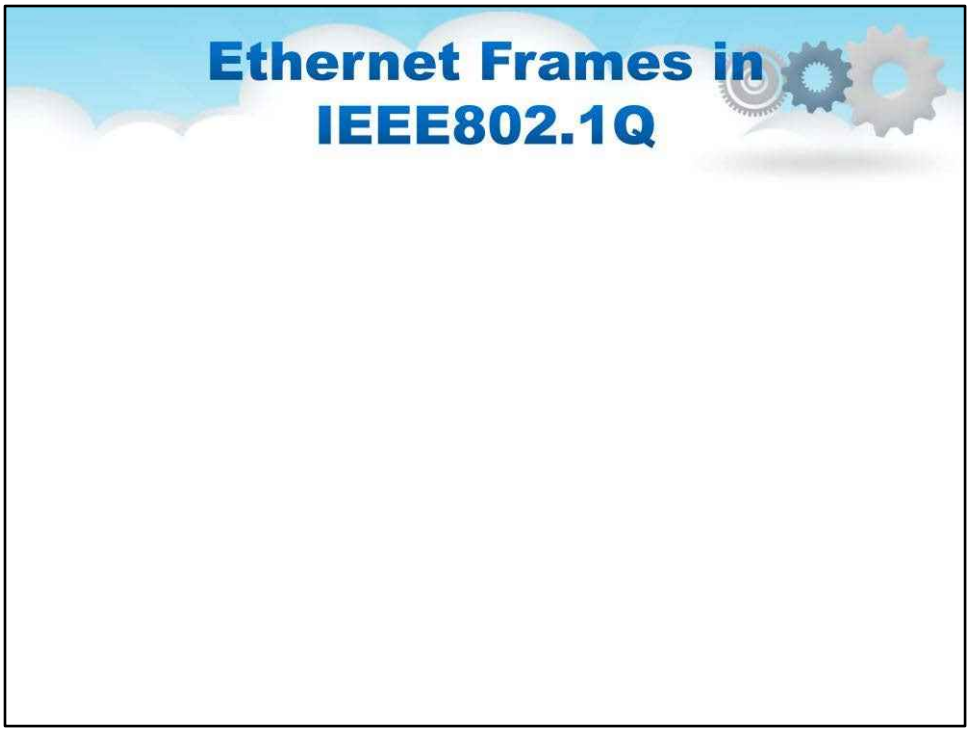

IEEE 802.1Q Tunnel and
Trunk Ports

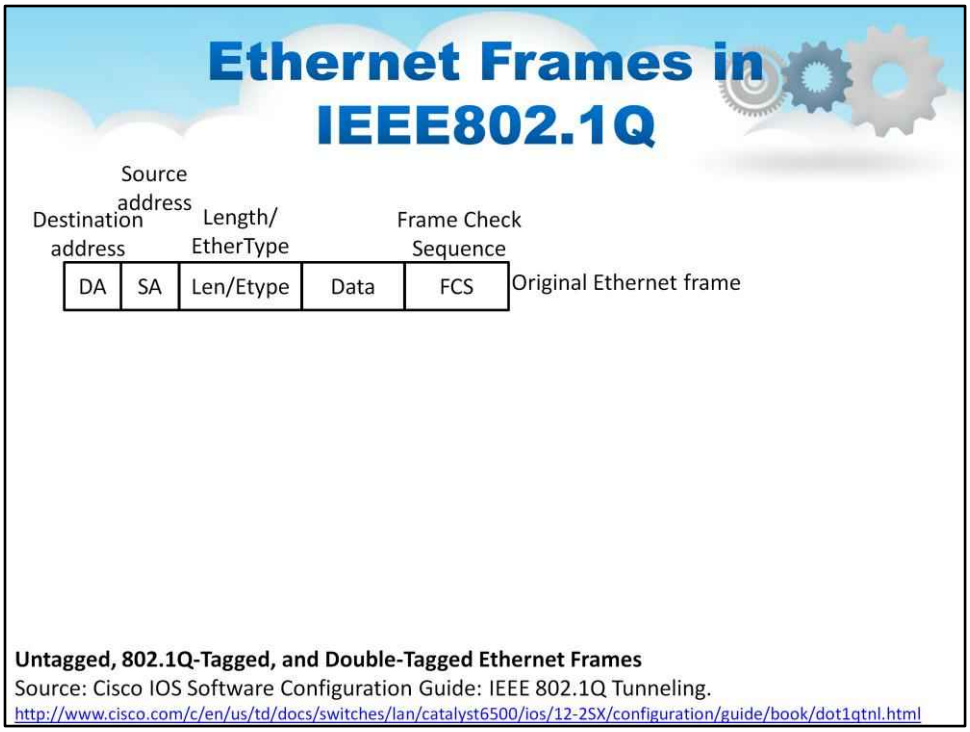


Тунельні порти IEEE 802.1Q знаходяться в мережі постачальника послуг

- Підтримуються кілька тунельних портів

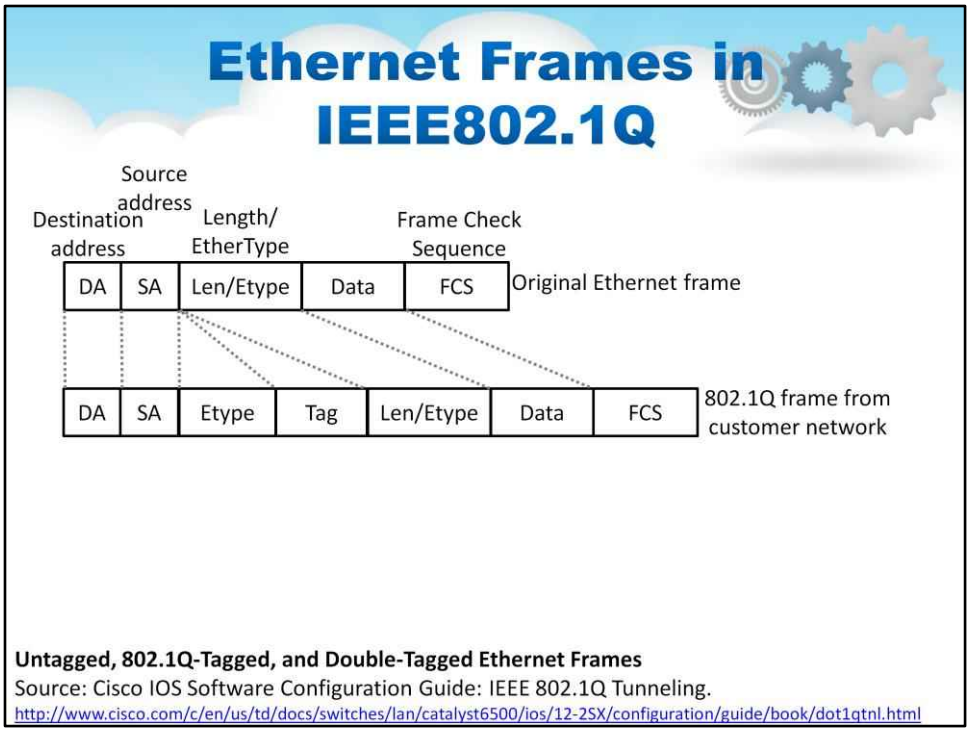
Магістральний порт IEEE 802.1Q у мережі клієнта


Ethernet Frames in
IEEE802.1Q



IEEE802.1Q додає 32-розрядне поле між MAC-адресою і EtherTypes поле

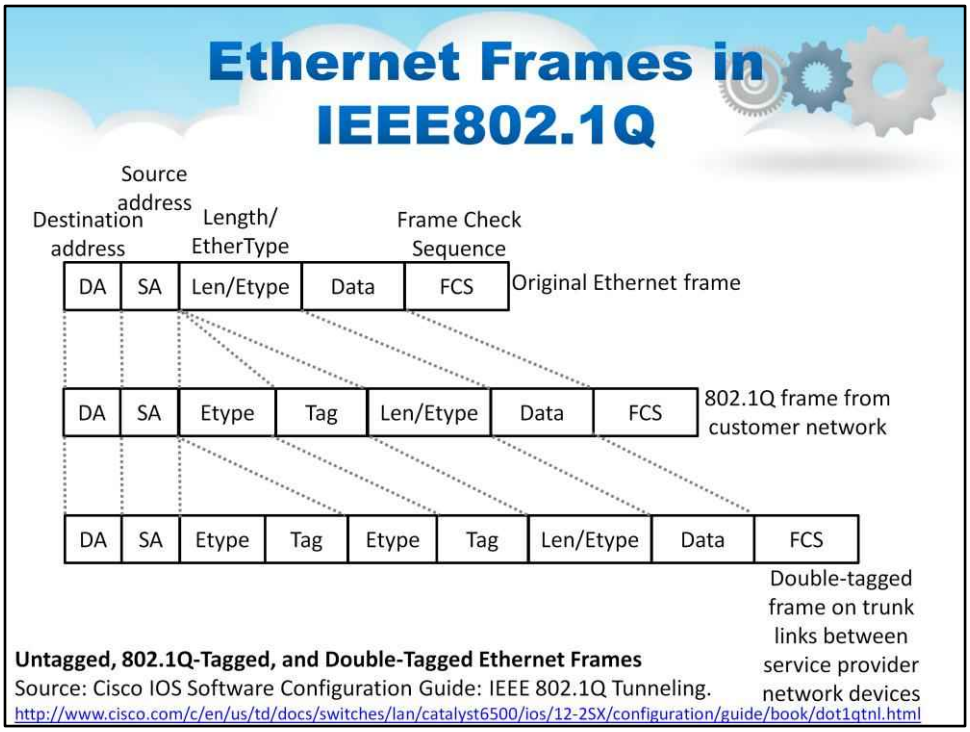
- ETYPE(2B): ідентифікатор протоколу
- Тег Dot1Q (2B): номер VLAN, код пріоритету



Untagged, 802.1Q-Tagged, and Double-Tagged Ethernet Frames
Source: Cisco IOS Software Configuration Guide: IEEE 802.1Q Tunneling.
http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/dot1qtnl.html

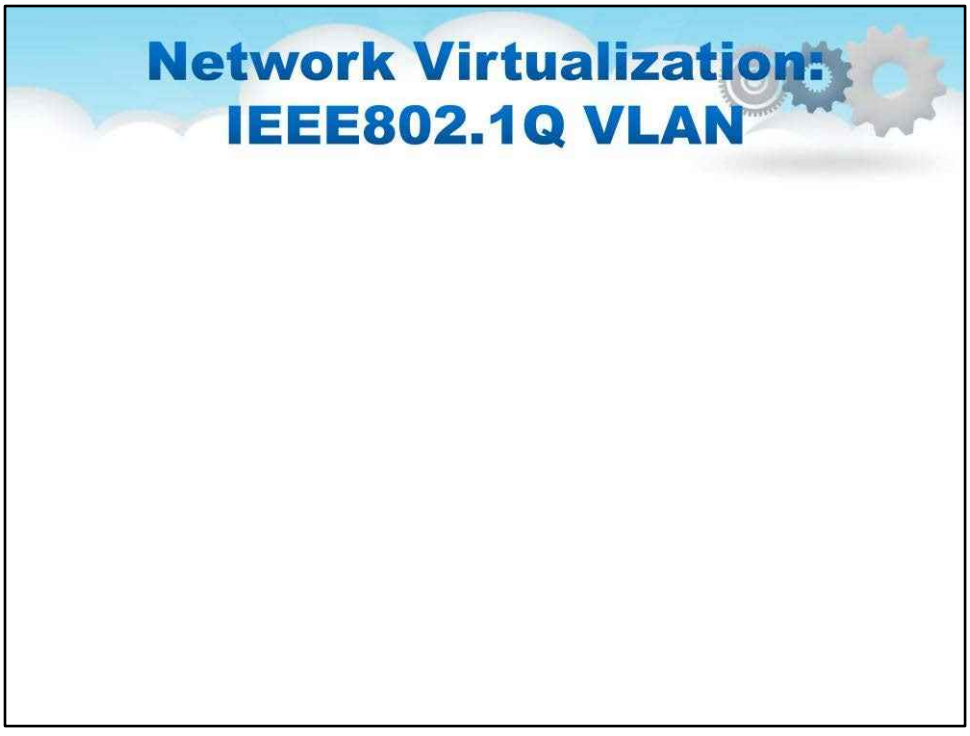
IEEE802.1Q додає 32-розрядне поле між MAC-адресами і EtherTypes поле

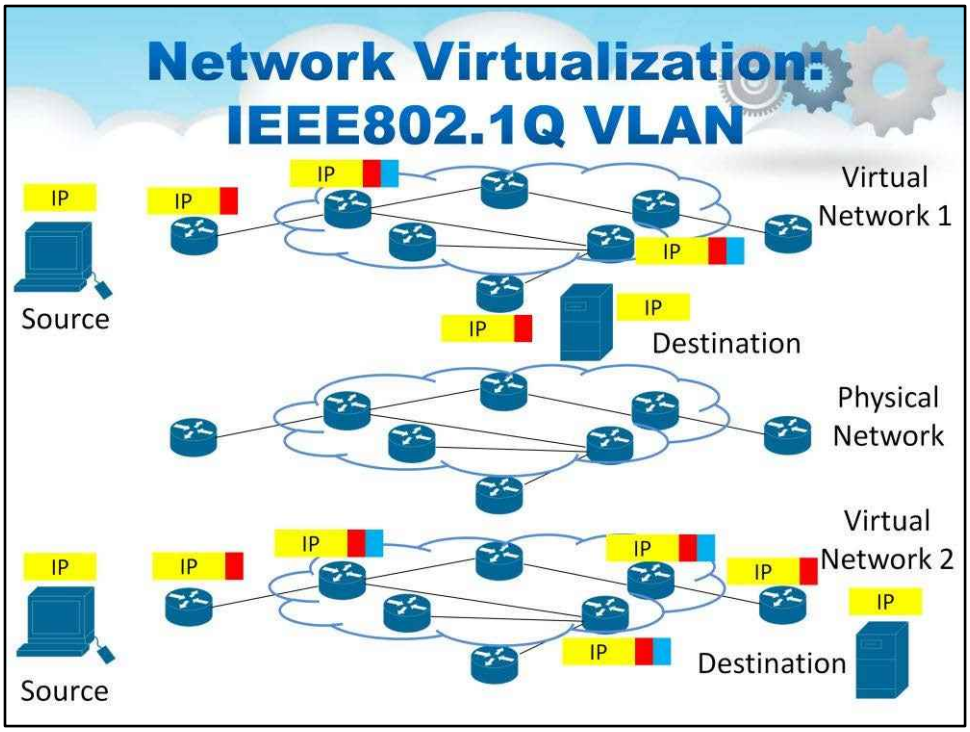
- ETYPE(2B): ідентифікатор протоколу
- Тег Dot1Q (2B): номер VLAN, код пріоритету

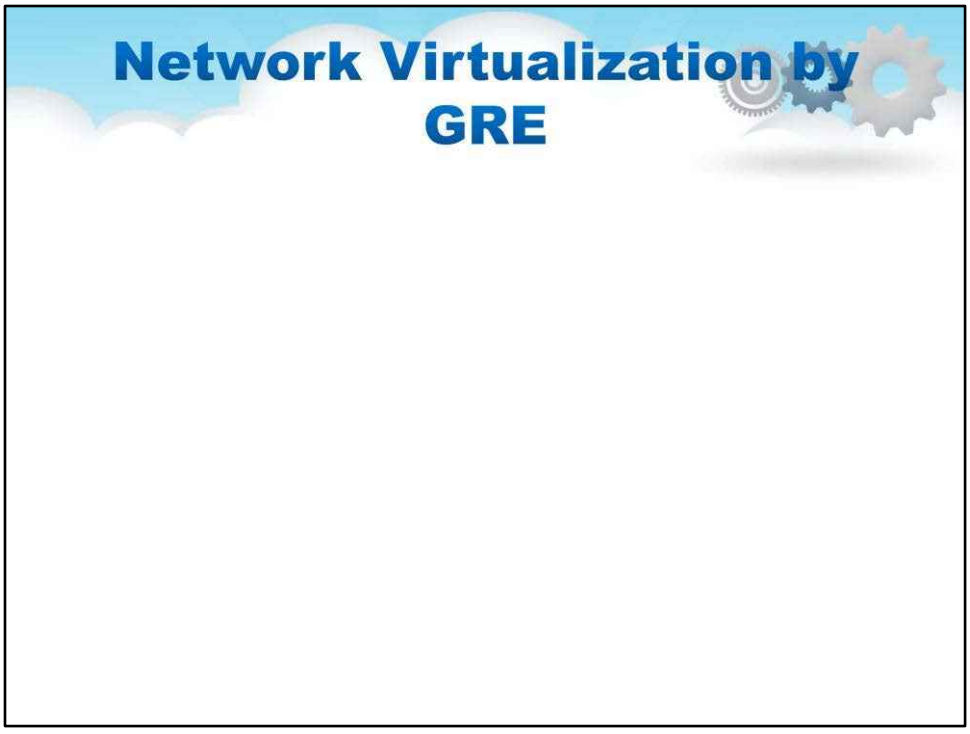


IEEE802.1Q додає 32-розрядне поле між MAC-адресами і EtherTypes поле

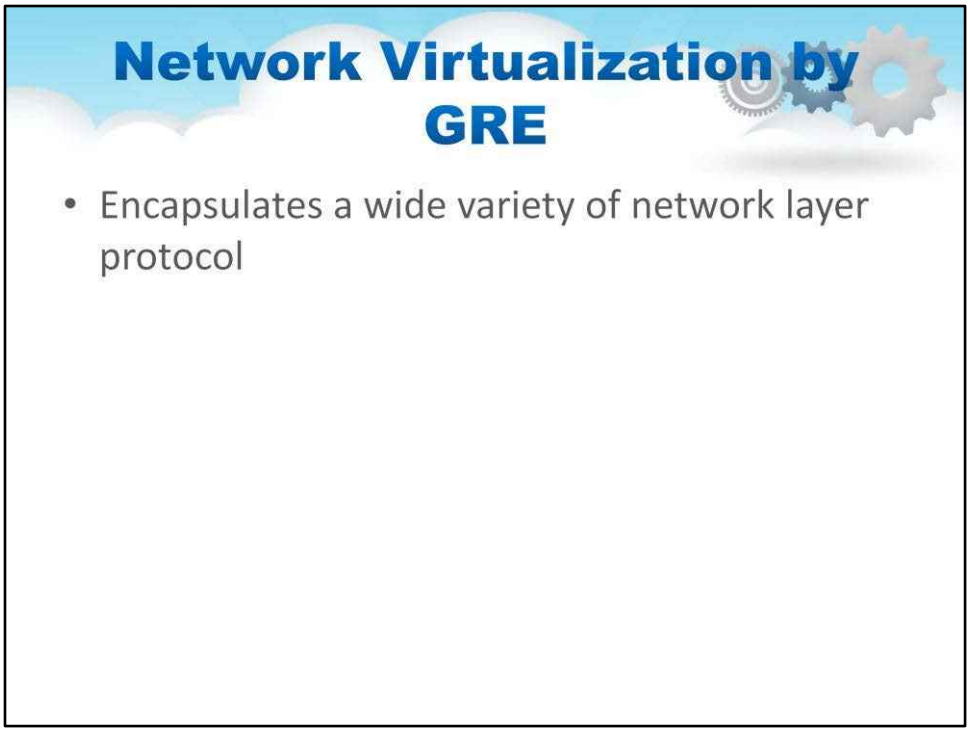
- ETYPE(2B): ідентифікатор протоколу
- Тег Dot1Q (2B): номер VLAN, код пріоритету


Network Virtualization:
IEEE802.1Q VLAN

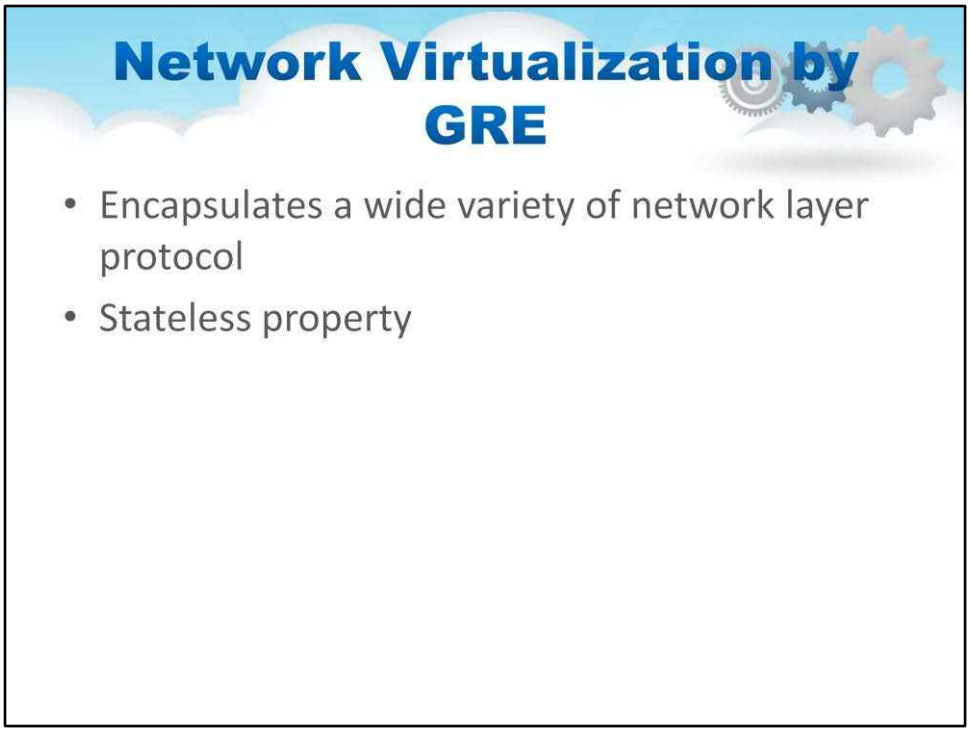

Network Virtualization:
IEEE802.1Q VLAN
IP
Source
IP
IP
IP
IP
IP
Destination
Virtual
Network 1
Physical
Network
Virtual
Network 2
IP
Source
IP
IP
IP
IP
IP
IP
Destination



GRE (Generic Routing Encapsulation) — це тунельний протокол, розроблений Cisco



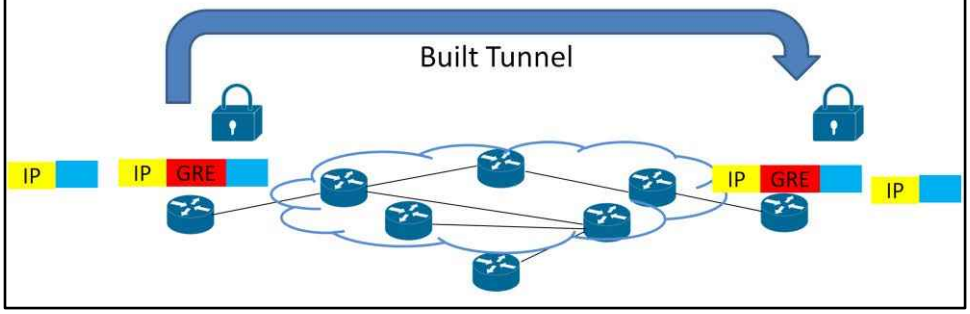
У поєднанні з IPSec забезпечує захищену багатодомну VPN

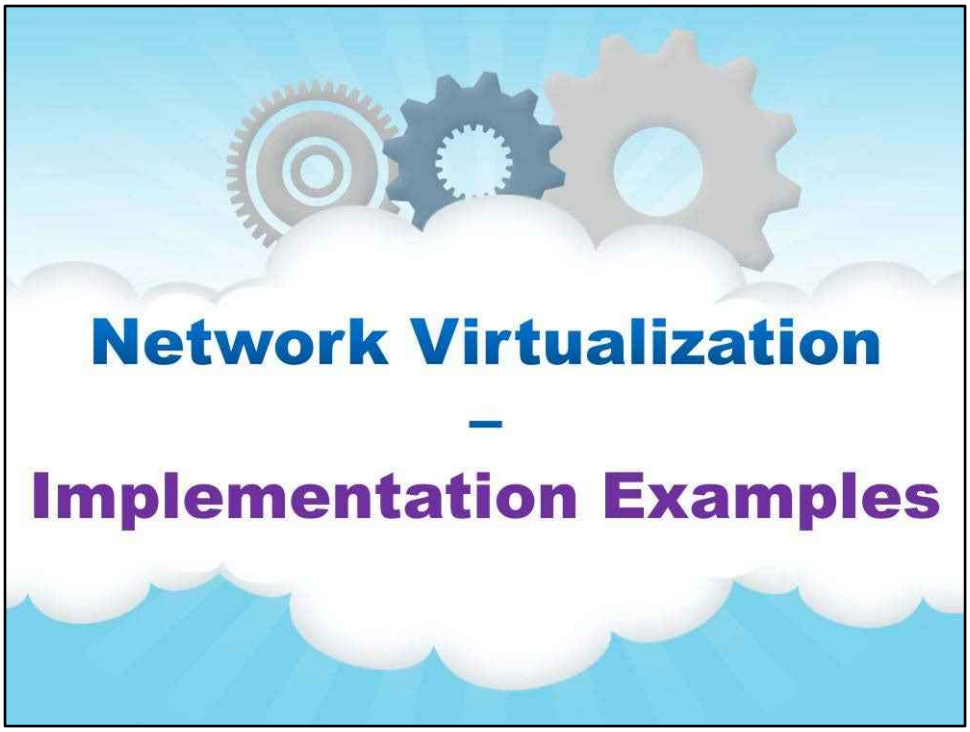


Це означає, що кінцева точка не зберігає інформацію про стан

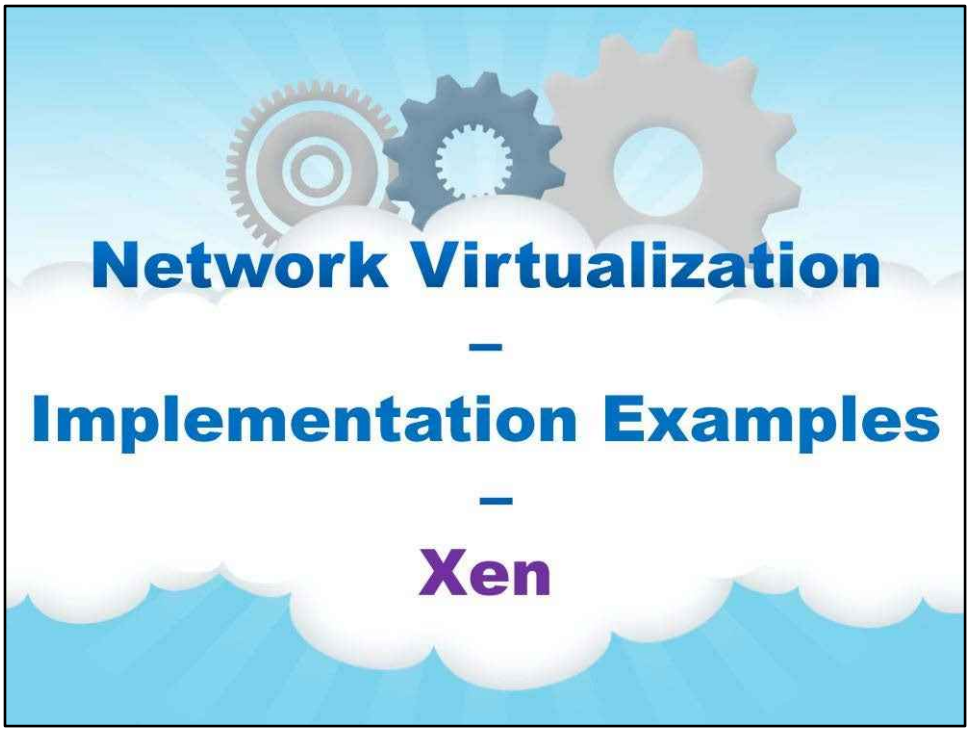
Network Virtualization by GRE

- Encapsulates a wide variety of network layer protocol
- Stateless property

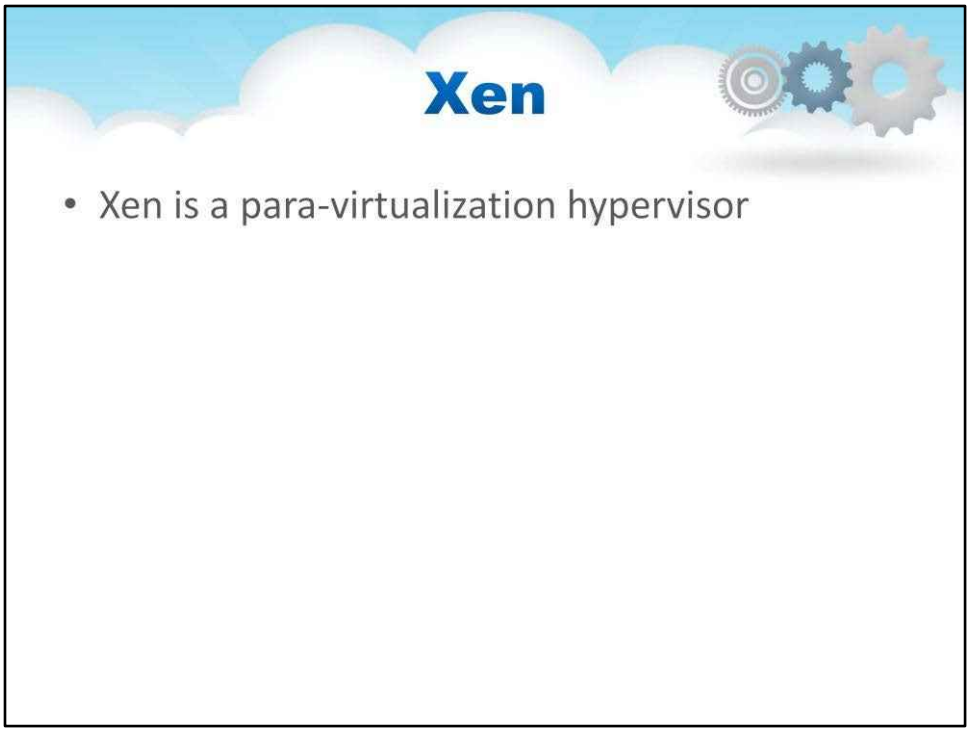




Приклади реалізації



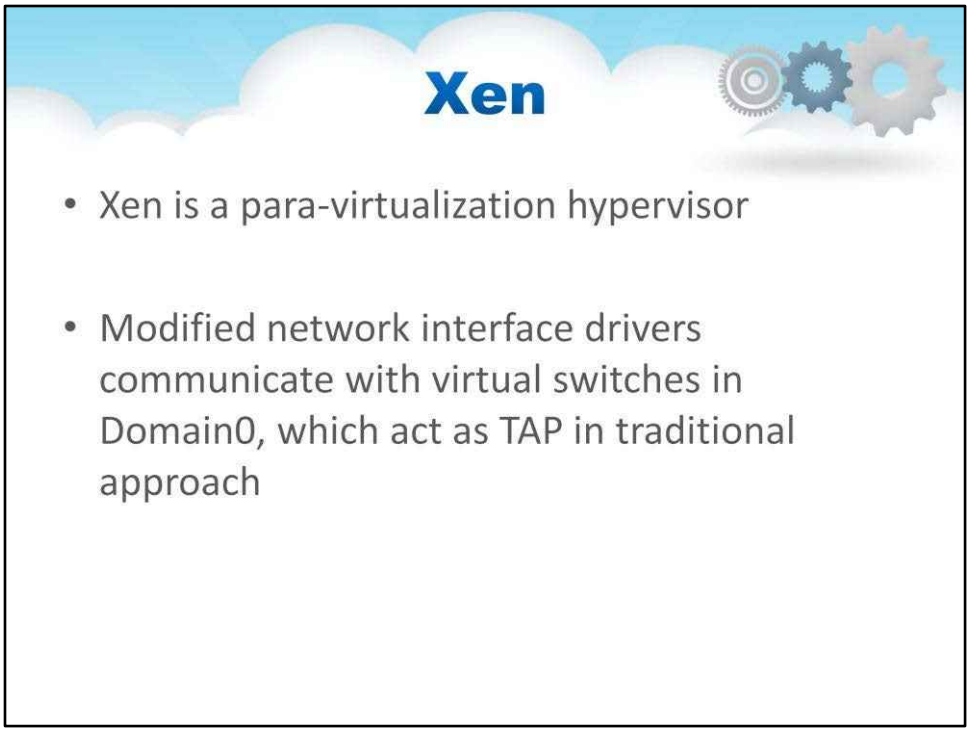
Xen



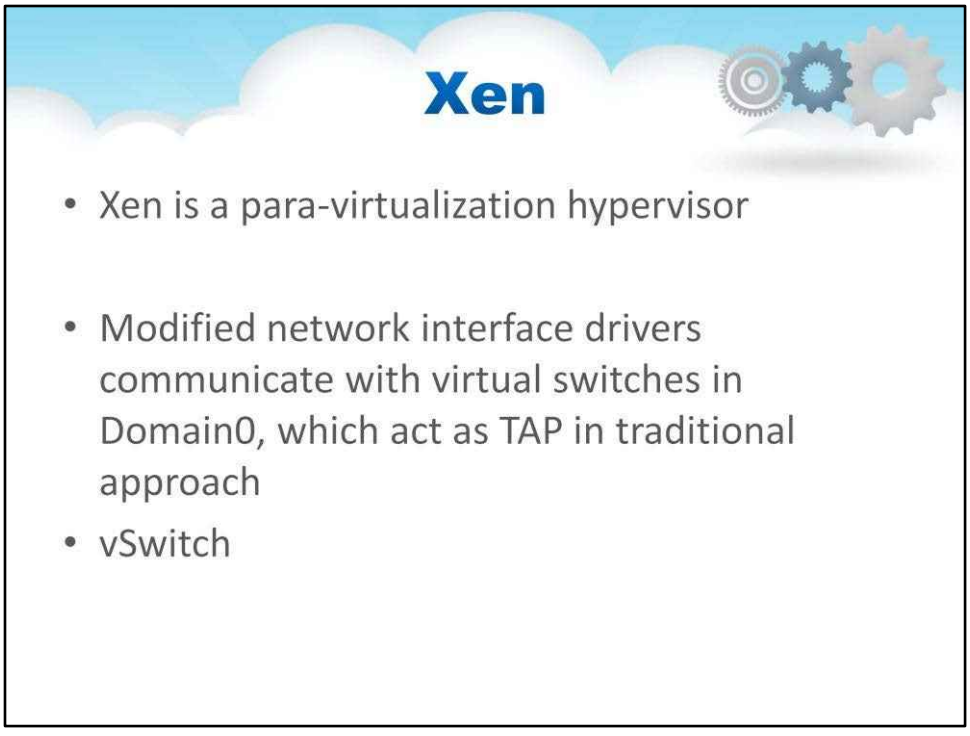
Далі розглянемо Xen. Xen — це гіпервізор паравіртуалізації, тому гостьова ОС використовує модифіковані драйвери мережевого інтерфейсу

Модифіковані драйвери мережевого інтерфейсу спілкуються з віртуальними комутаторами в Domain0, які діють як TAP у традиційному підході

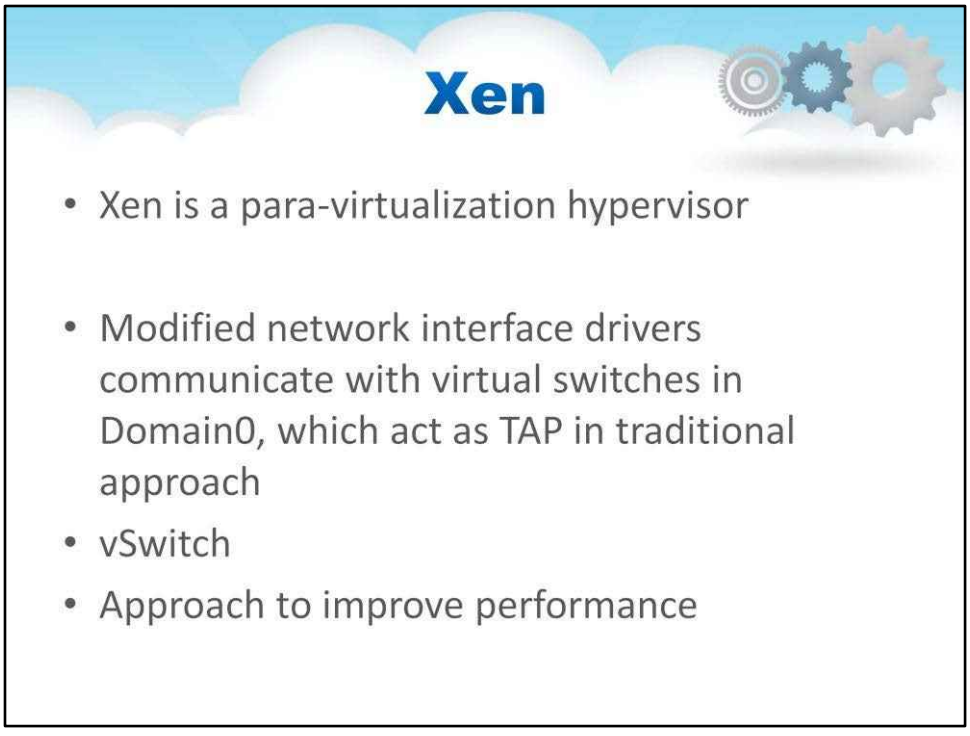
Xen — це гіпервізор паравіртуалізації, тому гостьова ОС використовує модифіковані драйвери мережевого інтерфейсу



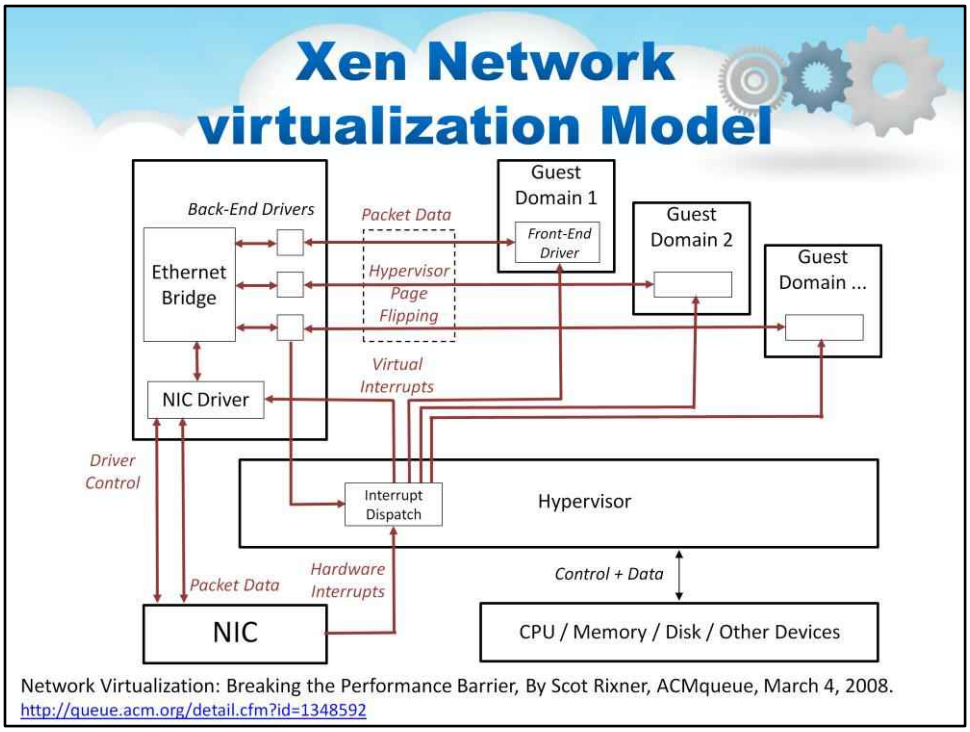
- Гіпервізор перепризначає сторінку пам’яті для MMIO (Memory Mapped I/O)
- Кожного разу, коли надсилаються пакети, індукуйте одне перемикання контексту з гостьової системи на домен 0, щоб керувати справжньою мережевою карткою



vSwitch у Xen може бути реалізований мостом Linux або працювати з іншими оптимізованими реалізаціями мосту



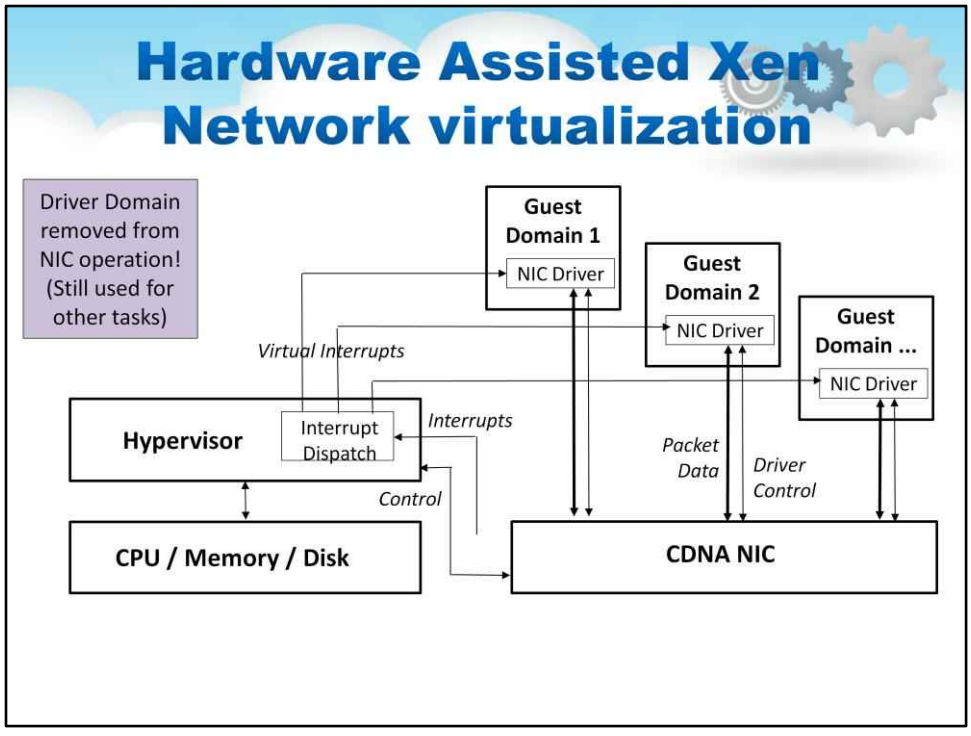
Апаратний адаптер CDNA (паралельний прямий доступ до мережі).



На цьому слайді представлено детальну ілюстрацію моделі віртуалізації мережі Xen.

Можна побачити, що в кожному з гостьових доменів є налаштований драйвер, який спілкується з гіпервізором, який потім є посередником з ОС, щоб з’єднати їх із фізичною мережевою карткою.

Здебільшого ця складність пов’язана з тим, що мережевий адаптер розроблено для використання з ним одна ОС за раз (концепція кількох ОС, які контролюють один мережевий адаптер через віртуалізації не було на момент розробки апаратного забезпечення).



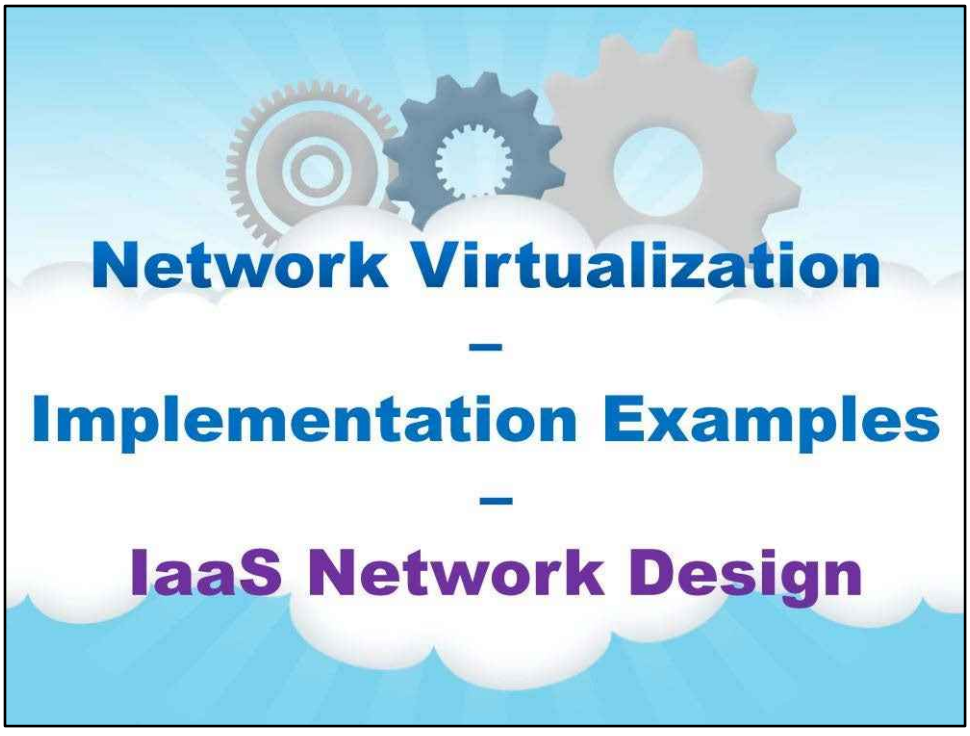
Цей слайд містить ілюстрацію, яка показує, як працює апаратна віртуалізація мережі Xen,

Основою цього є пере-архітектурний мережевий адаптер із підтримкою кількох операційних системЦе називається апаратним адаптером CDNA (паралельний прямий доступ до мережі).

Це значно покращує продуктивність апаратного забезпечення та видаляє домен драйвера з даних і переривань

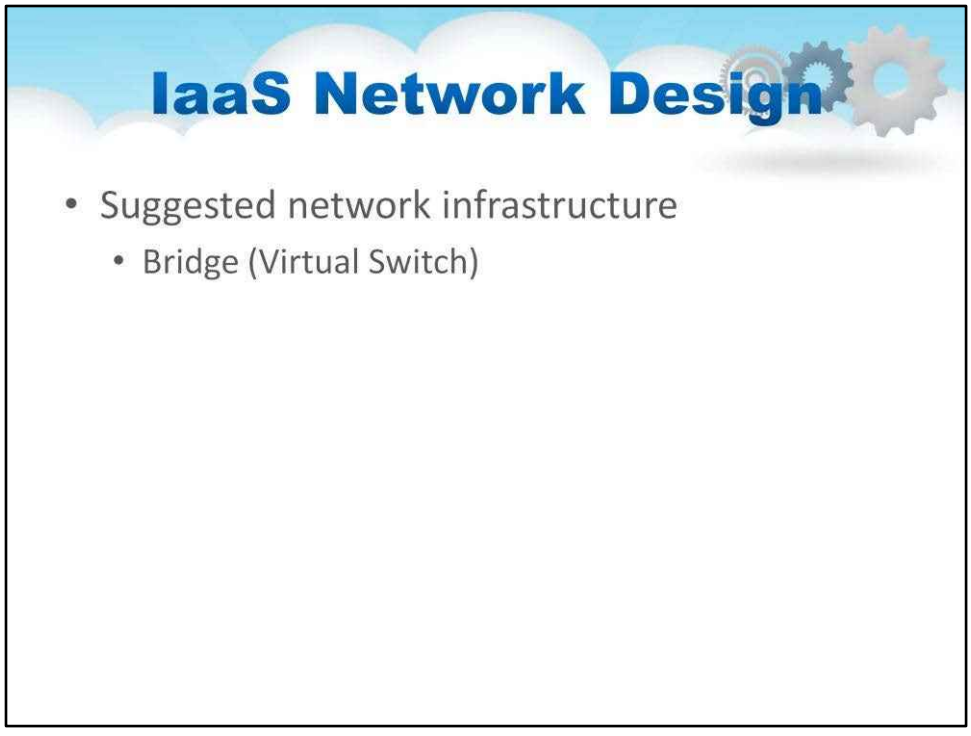
Апаратний адаптер CDNA (паралельний прямий доступ до мережі).

- Значно підвищує продуктивність апаратного забезпечення
- Видалити домен драйвера з даних і переривань
- Гіпервізор відповідає лише за віртуальні переривання та призначення контексту гостьовій ОС



Тепер давайте об’єднаємо деякі з цих концепцій і розглянемо мережевий дизайн хмари IaaS.

Дизайн мережі IaaS



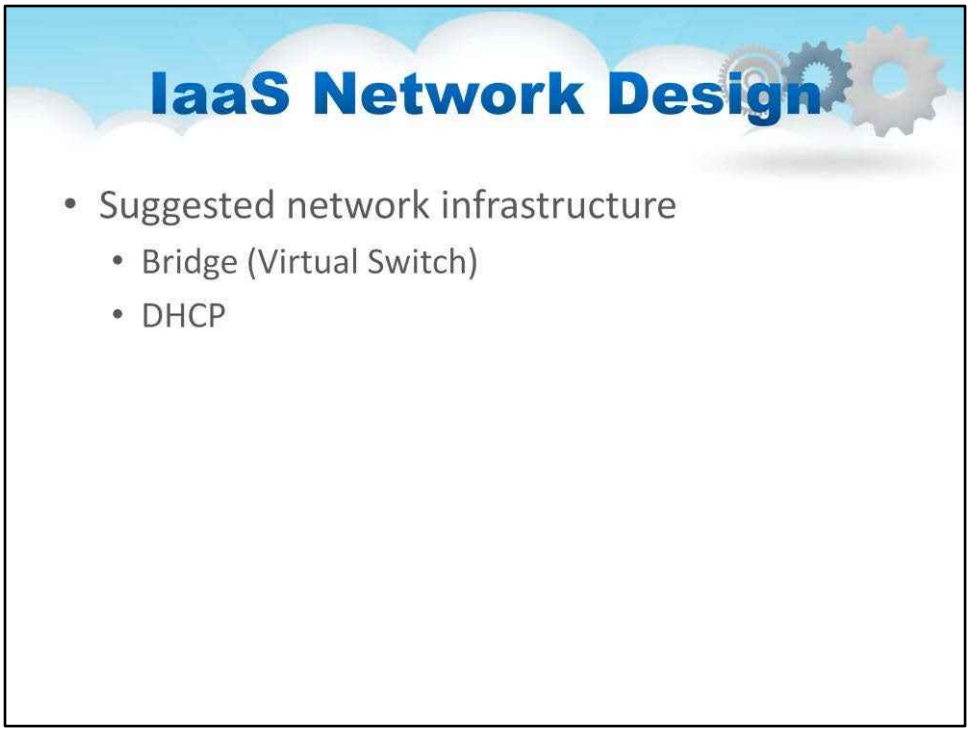
Фізичний сервер має підключитися до фізичної мережі, і він робить це за допомогою фізичних мережевих карток. Існує міст від цього фізичного рівня до віртуального рівня в гіпервізорі/на рівні операційної системи (vSwitch).

DHCP можна використовувати як частину стратегії IPAM (керування IP-адресами) із хмари назовні для зіставлення віртуальних MAC-адрес віртуальних машин із приватними IP-адресами в локальній мережі.

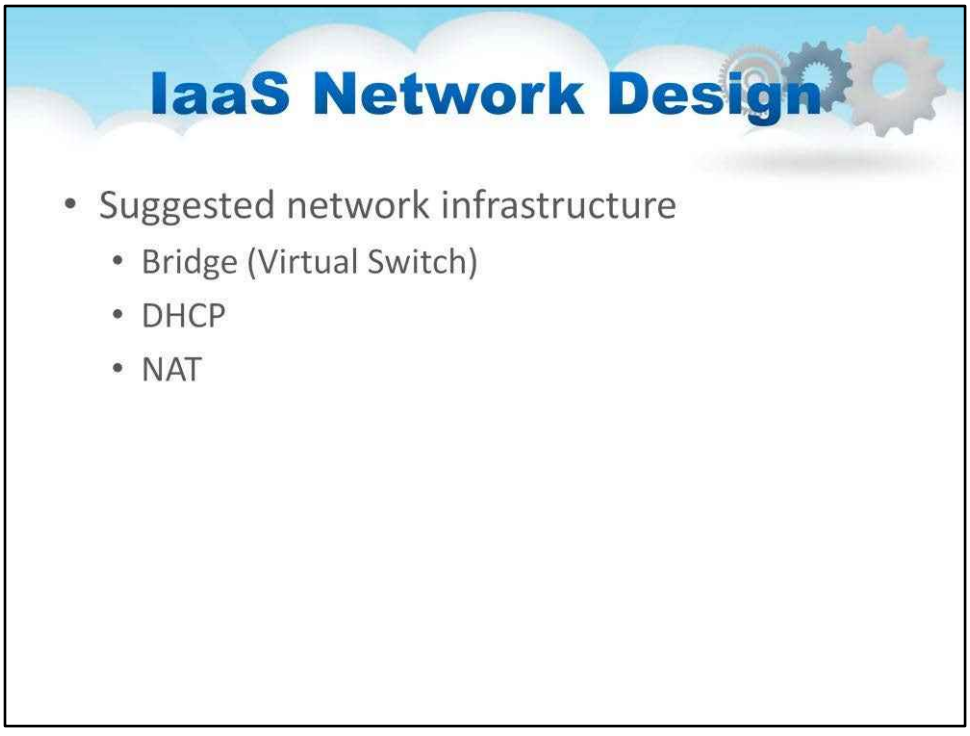
NAT Пересилає пакети до загальнодоступної мережі (WAN)

Всередині хмари IPAMобробляється самою хмарою зі своїми власнимиТаблиця зіставлення IP/MAC

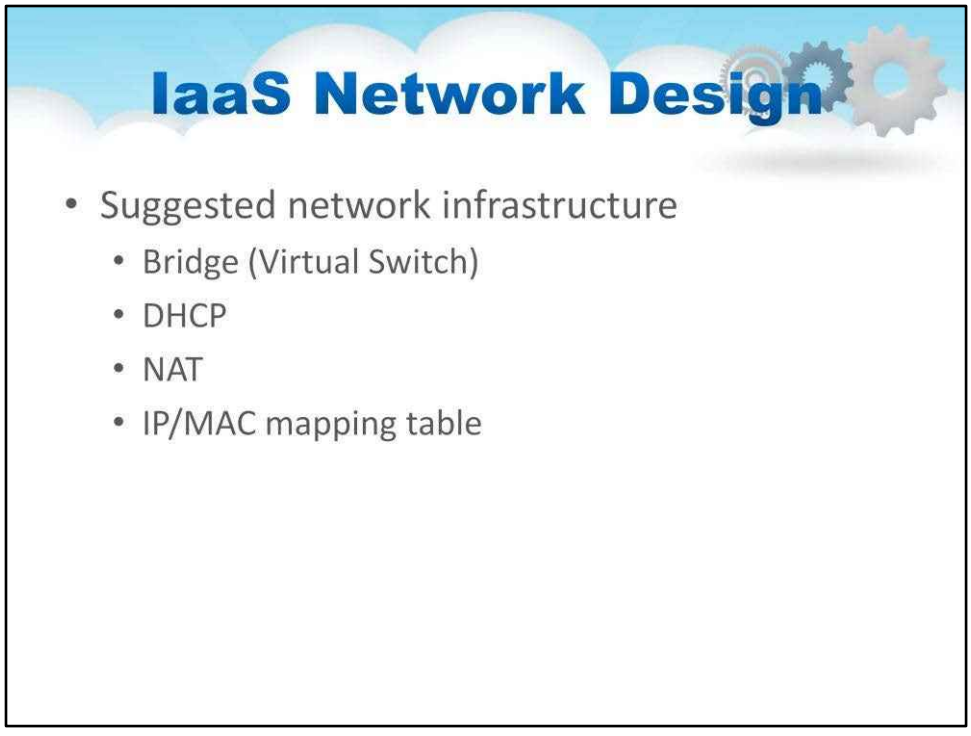
Зробіть так, щоб віртуальні машини на одному вузлі використовували фізичні мережеві карти.



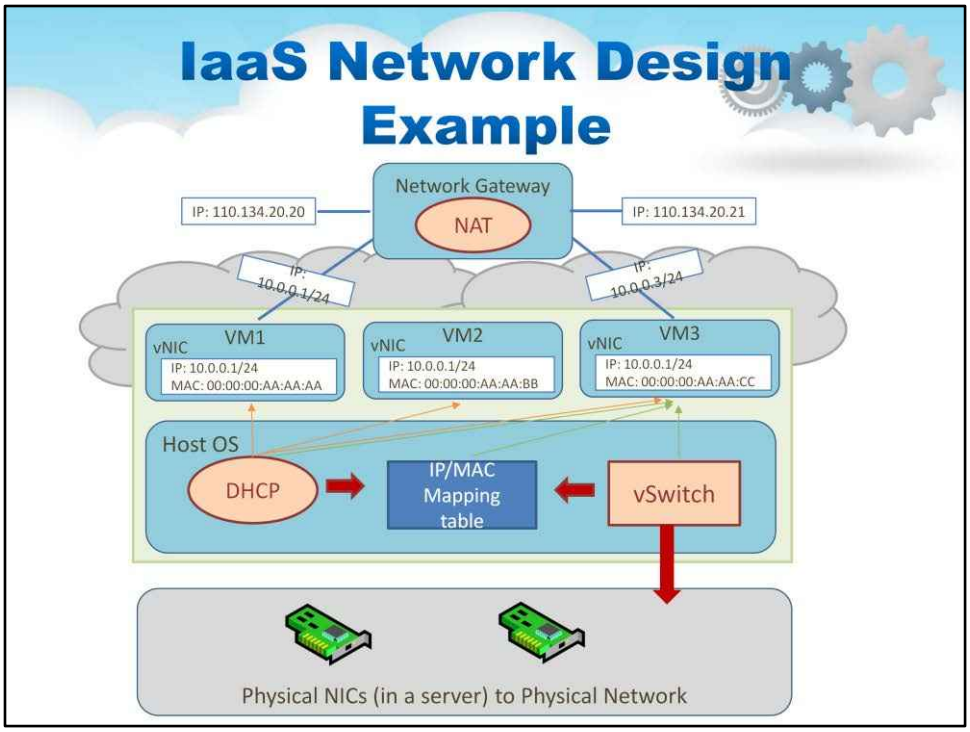
Зіставте віртуальні MAC-адреси віртуальних машин із приватними IP-адресами в локальній мережі.



Пересилайте пакети в загальнодоступну мережу (WAN).



- IP-адреси призначаються хмарою.
- MAC-адреси призначаються гіпервізором.
- Ця таблиця відображення підтримується хмарною системою.

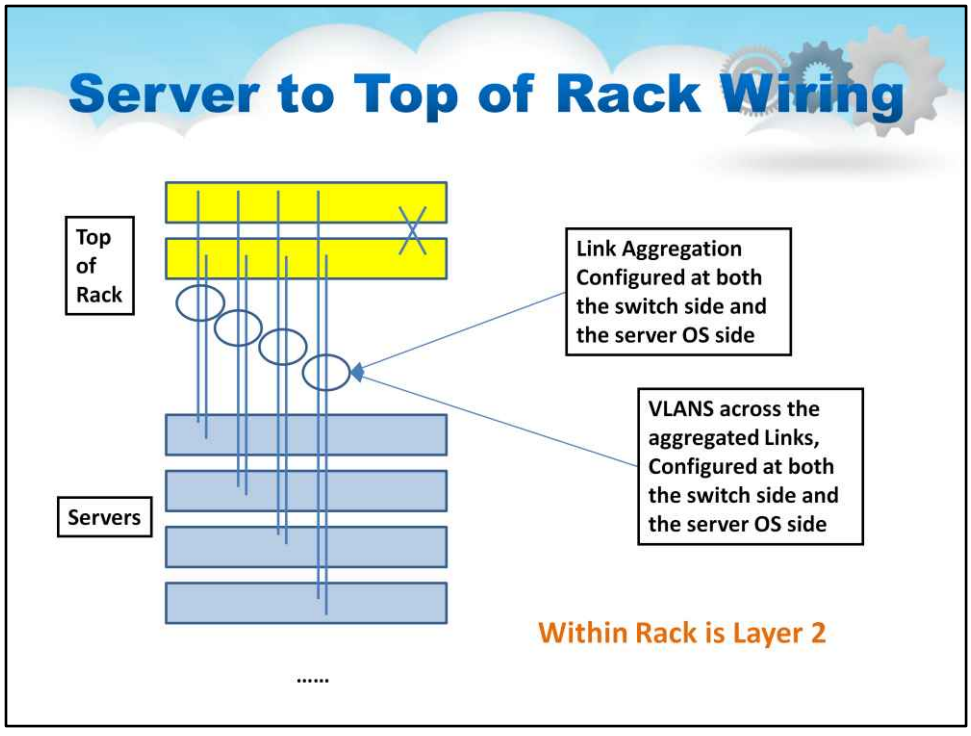


Малюнок на цьому слайді розширює попередні твердження та подає їх у діаграму. Він представляє гіпотетичний приклад дизайну віртуальної мережі IaaS. Три віртуальні машини налаштовані за допомогою vNIC і їхніх внутрішніх IP-адрес. Вони підключені до локальної мережі через фізичні мережеві карти та можуть отримувати доступ до зовнішнього або загальнодоступного Інтернету за допомогою протоколу трансляції мережевих адрес (NAT), який перетворює внутрішню IP-адресу в одну із зовнішніх публічних IP-адрес. Більш складну віртуальну мережеву інфраструктуру можна створити шляхом розгортання мережевих пристроїв на основі віртуальних машин або віртуалізації фізичних мережевих пристроїв. Нижче наведено компоненти створеної мережевої інфраструктури IaaS:

vSwitch: віртуальні машини на одному вузлі мають спільні фізичні мережеві карти.

DHCP: зіставлення віртуальних MAC-адрес віртуальних машин із приватними IP-адресами в локальній мережі. NAT: пересилання пакетів у загальнодоступну мережу.

Таблиця зіставлення IP/MAC: підтримує IP-адреси, призначені CMS, і MAC-адреси, призначені гіпервізором. Ця таблиця відображення підтримується CMS.

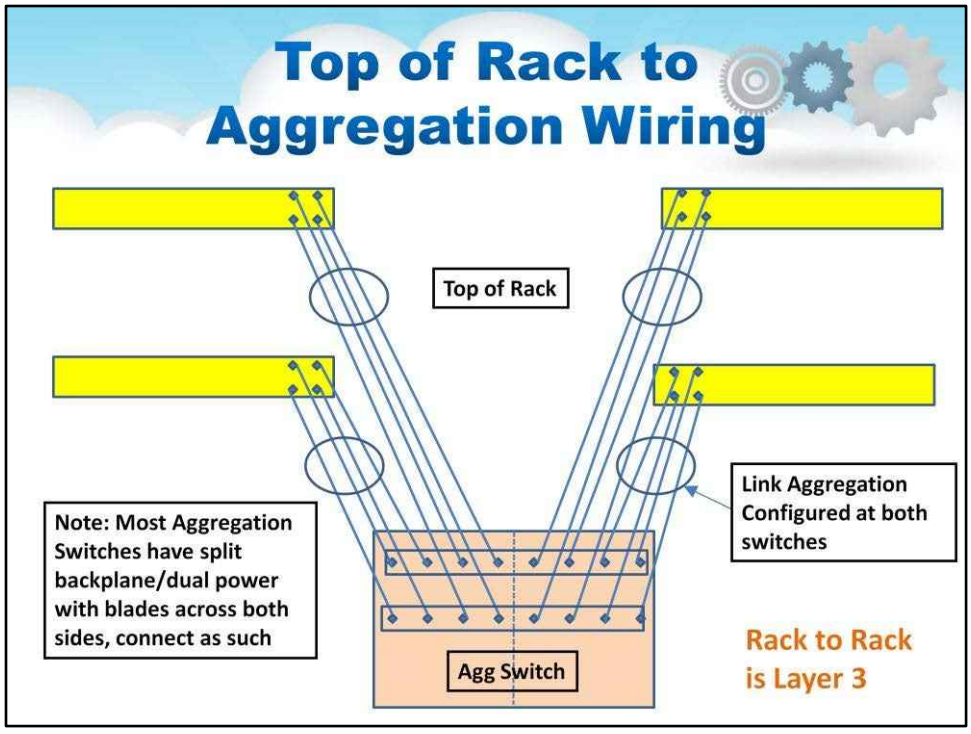


Тепер ми можемо взяти ідеї з попереднього середовища з одним сервером і показати, як побудувати більшу хмару

В основному сервери розташовані в стеках і підключені до фізичної верхньої частини стійки». мережі, як показано на ілюстрації на цьому слайді

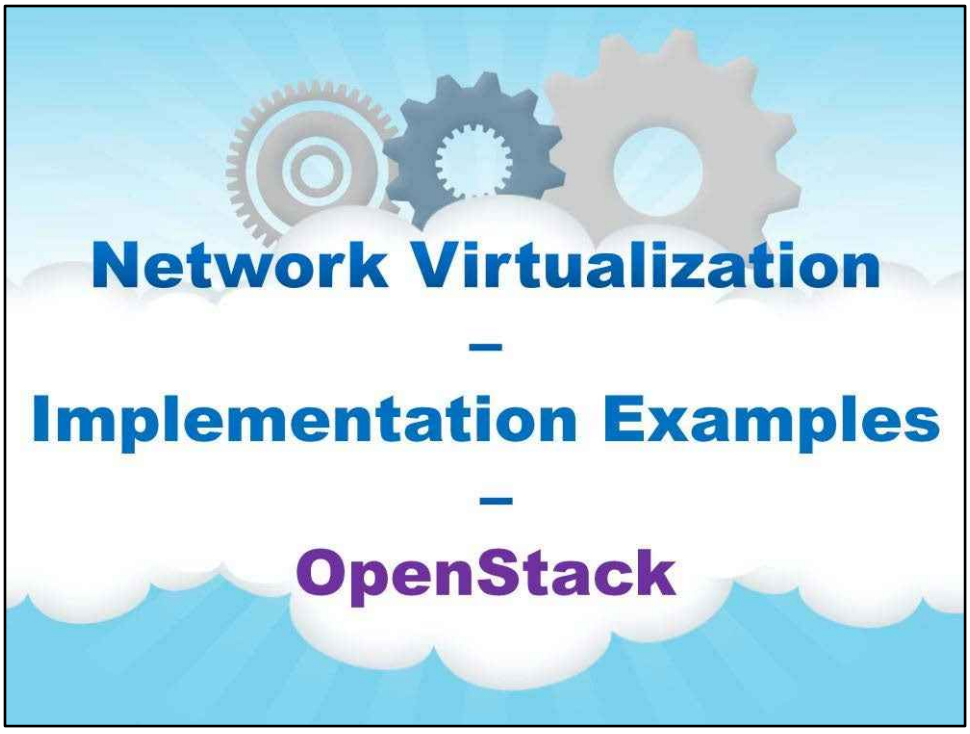
Зауважте, що дві фізичні мережеві карти на кожному сервері (з попереднього слайду) підключені до іншого комутатора у верхній частині стійки. Це підвищить надійність у разі збою зв’язку або комутатора. Вони налаштовані на використання агрегації посилань, щоб працювати так, ніби вони є «зв’язаний», щоб отримати використання всієї придбаної смуги пропускання.

Усередині стійки верхні комутатори стійки використовують комутацію рівня 2 для підключення всіх вузлів.

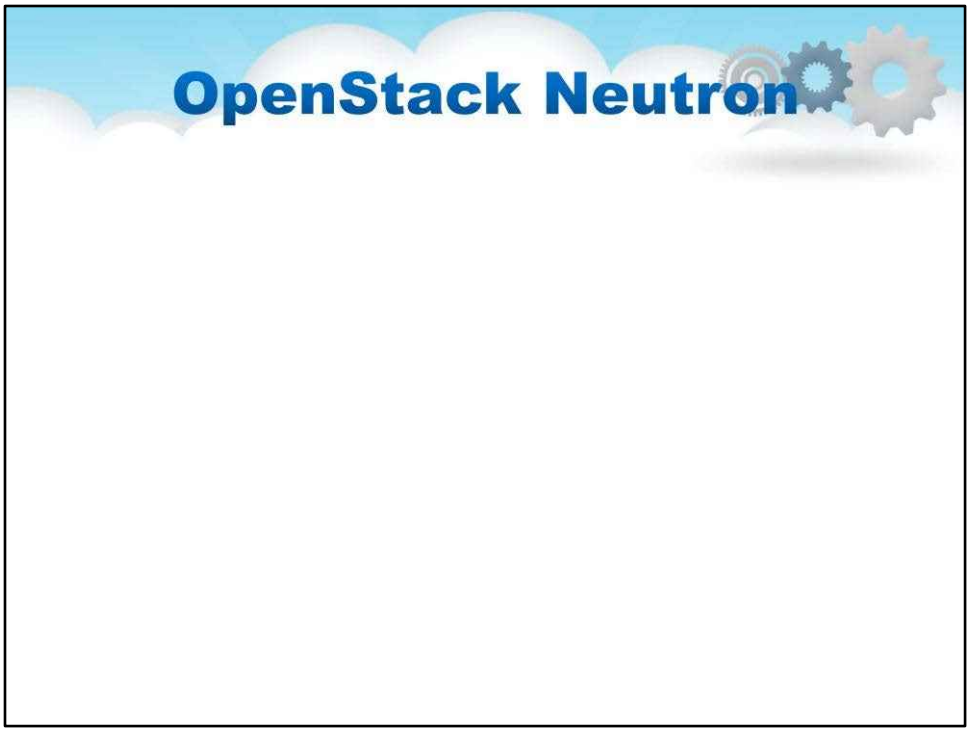


На цьому слайді показано, як кілька стійок підключаються до комутатора агрегації. Висхідні канали зв'язку є L3. У цьому прикладі ми показуємо 4 висхідні канали на комутатор у верхній частині стійки. Це забезпечує підхід високої доступності підключення кожного з двох комутаторів у верхній частині стійки до окремої задньої панелі та окремих блейд-серверів на комутаторі агрегації.

Знову агрегація каналів використовується для максимальної пропускної здатності та максимальної надмірності.

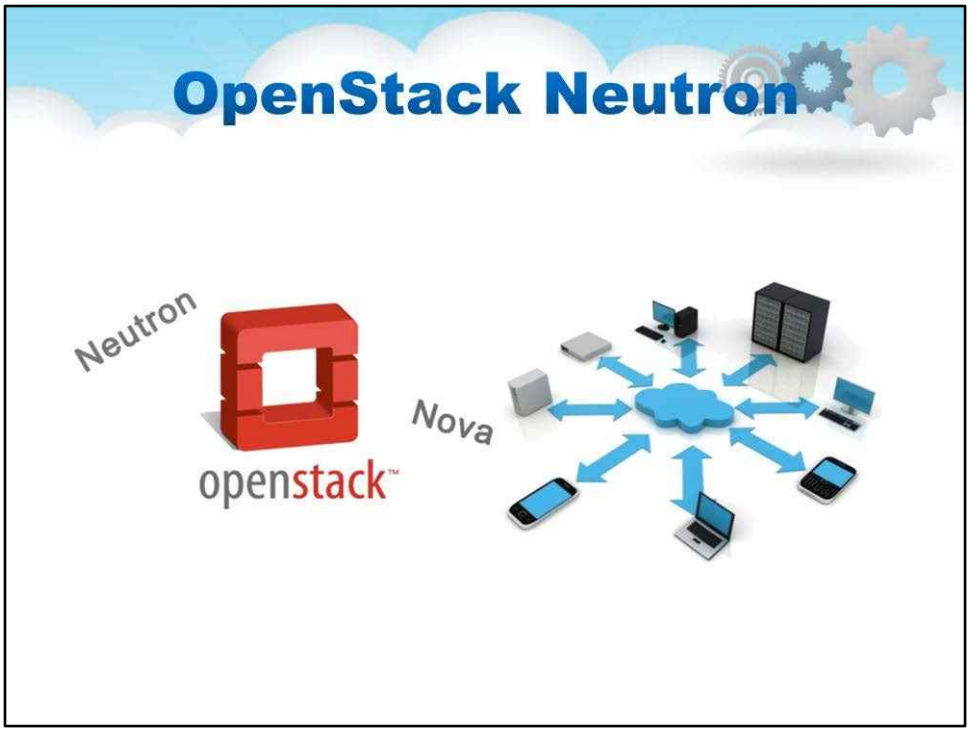

Network Virtualization
–
Implementation Examples
–
OpenStack

OpenStack



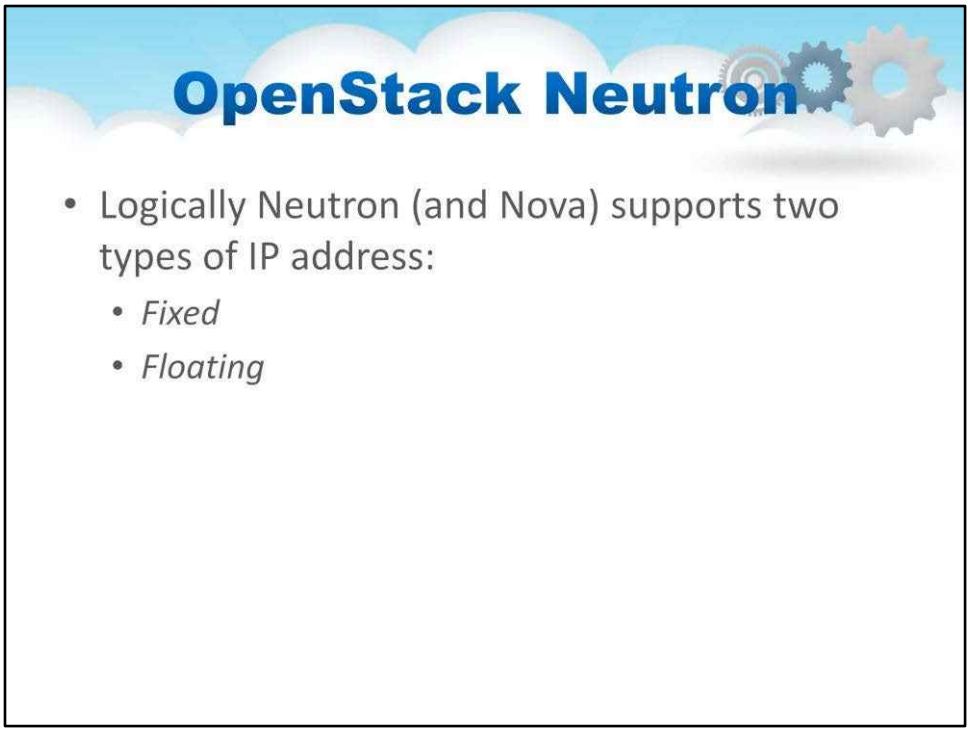
OpenStack Neutron — це компонент керування мережею в хмарній платформі OpenStack

- Працює разом із мережевим контролером у Nova (обчислювальний компонент)



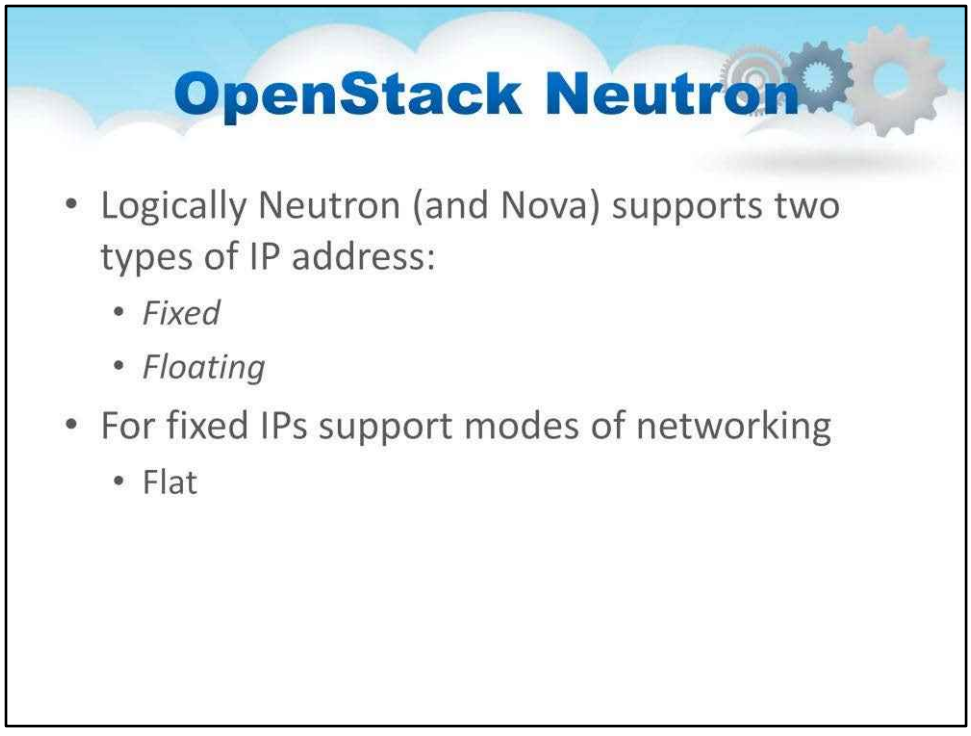
Менеджер мережі Neutron надає такі функції

- Надає API для створення розширених мережевих топологій і налаштування розширених мережевих політик
- Увімкніть розроблені користувачами плагіни, які надають розширені мережеві можливості; наприклад, NFV або SDN
- Дозволяє створювати розширені мережеві служби, які можна використовувати в орендаряІнфраструктура Openstack



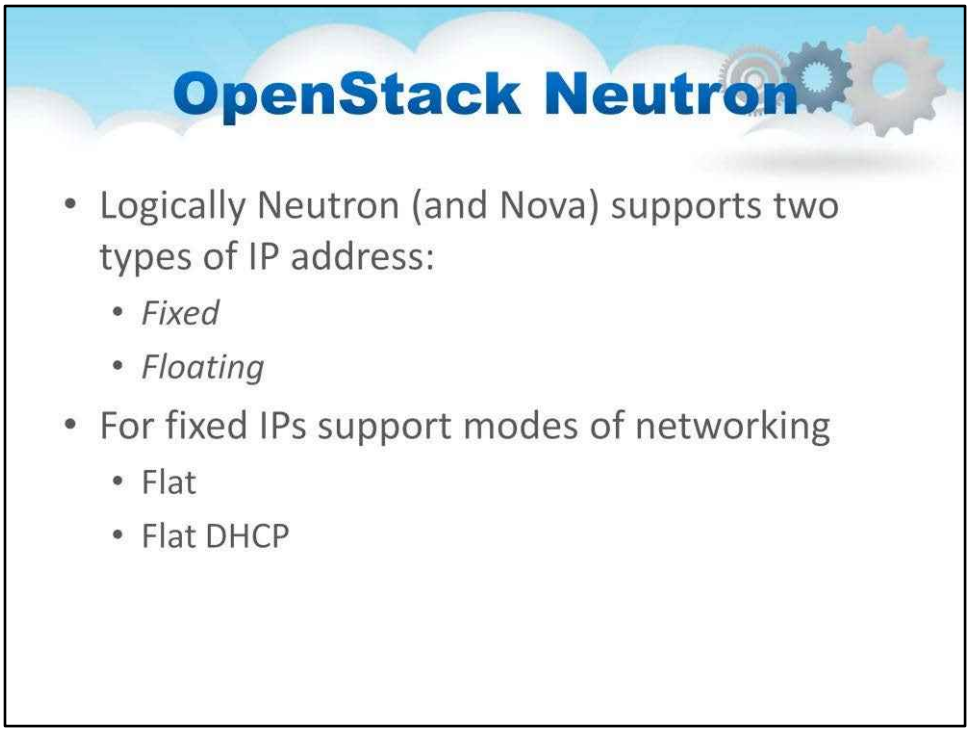
Логічно Neutron (і Nova) підтримує два типи IP-адрес:

- Виправлені, які асоціюються з екземпляром віртуальної машини під час створення та залишаються пов’язаними до припинення
- Плаваючий, який можна динамічно приєднувати/від’єднувати до/від запущеного екземпляра віртуальної машини під час виконання

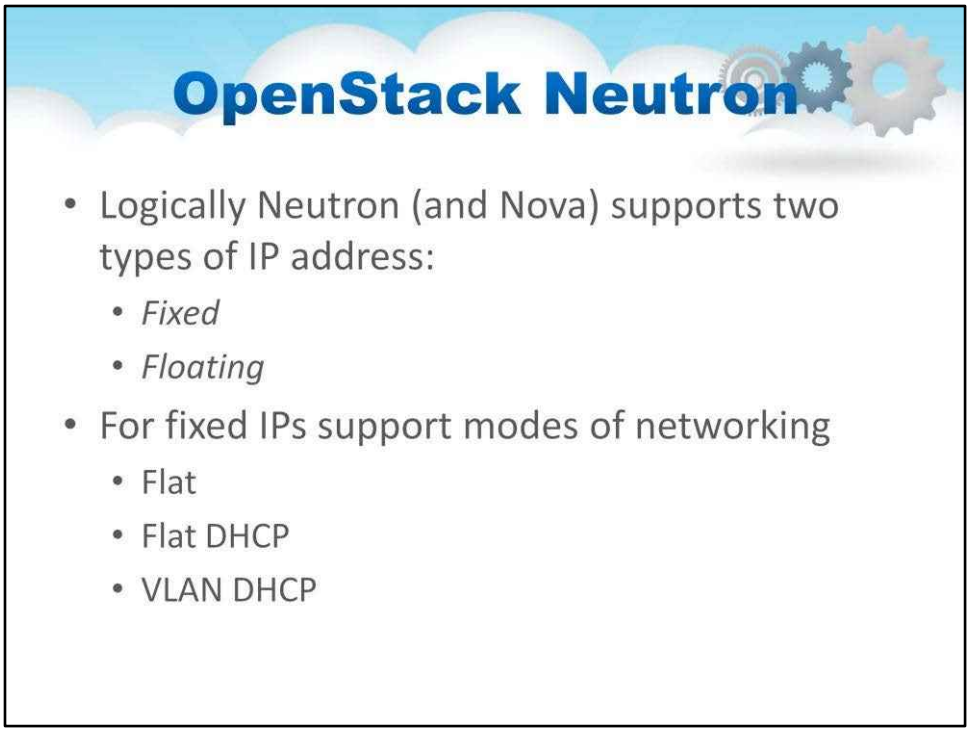


Для фіксованих IP-адрес Neutron (і Nova) підтримують такі три режими мережі

- Плоский режим надає кожному екземпляру віртуальної машини фіксовану IP-адресу, пов’язану з мережевим мостом за умовчанням, який потрібно налаштувати вручну



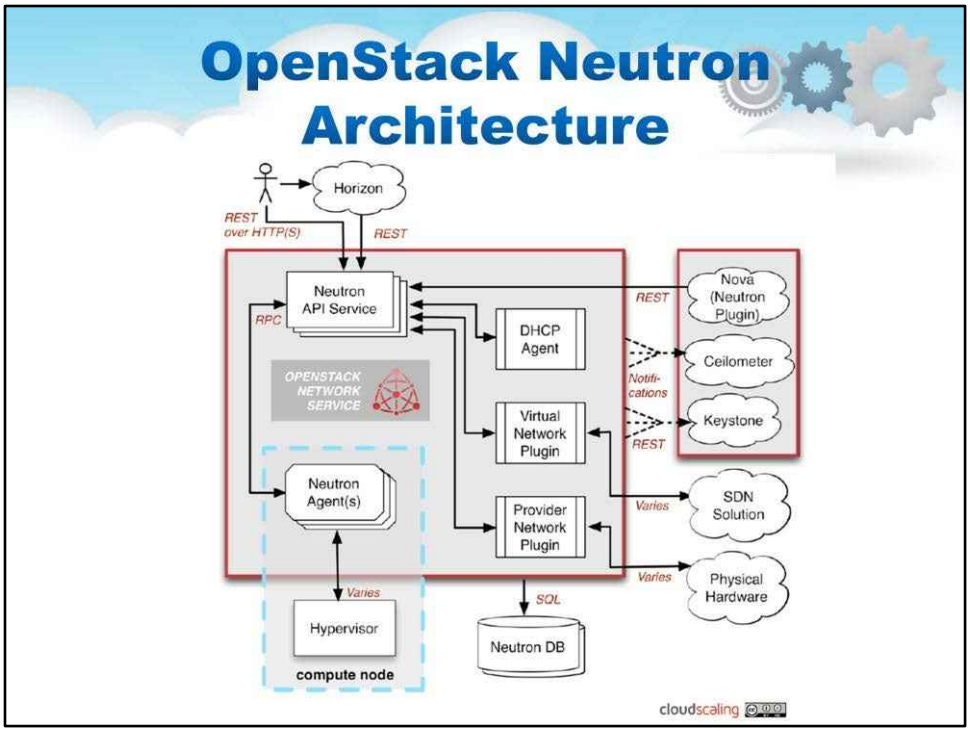
Плоский DHCP режим покращує режим Flat, створюючи сервер DHCP для надання фіксованих IP-адрес для екземплярів віртуальних машин



VLAN DHCP – режим мережі за замовчуванням, у якому Nova створює VLAN і

віртуальний міст для кожного проекту

- Примірникам віртуальної машини в проекті призначається приватна IP-адреса з діапазону IP-адрес.
- Користувачі можуть отримати доступ до цих екземплярів за допомогою спеціального екземпляра VPN під назвою «cloudpipe», який використовує сертифікат і ключ для створення VPN (віртуальної приватної мережі).



OpenStack Networking (Neutron, раніше Quantum) — це плагінована, масштабована та керована API система для керування мережами та IP-адресами. Як і інші аспекти хмарної операційної системи, її можуть використовувати адміністратори та користувачі для збільшення вартості наявних активів центру обробки даних. OpenStack Networking гарантує, що мережа не буде вузьким місцем або обмежуючим фактором у розгортанні хмари та надає користувачам справжнє самообслуговування навіть у їхніх конфігураціях мережі.

OpenStack Networking — це система для керування мережами та IP-адресами. Як і інші аспекти хмарної операційної системи, її можуть використовувати адміністратори та користувачі для збільшення вартості наявних активів центру обробки даних. OpenStack Networking гарантує, що мережа не буде вузьким місцем або обмежуючим фактором у розгортанні хмари та надає користувачам справжнє самообслуговування навіть у їхніх конфігураціях мережі.

OpenStack Neutron надає моделі мереж для різних програм або груп користувачів. Стандартні моделі включають плоскі мережі або VLAN для розділення серверів і трафіку. OpenStack Networking керує IP-адресами, дозволяючи використовувати виділені статичні IP-адреси або DHCP. Плаваючі IP-адреси дозволяють динамічно перенаправляти трафік до будь-якого з ваших обчислювальних ресурсів, що дає змогу перенаправляти трафік під час обслуговування або в разі збою. Користувачі можуть створювати власні мережі, контролювати трафік і підключати сервери та пристрої до однієї або кількох мереж. Адміністратори можуть скористатися перевагами програмно-визначеної мережевої технології (SDN), як-от OpenFlow, щоб забезпечити високий рівень мультитенантності та великого масштабу. OpenStack Networking має структуру розширення, яка надає додаткові мережеві служби, такі як системи виявлення вторгнень (IDS), балансування навантаження,

Мережеві можливості

OpenStack надає гнучкі моделі мереж, які відповідають потребам різних програм або груп користувачів. Стандартні моделі включають плоскі мережі або VLAN для розділення серверів і трафіку.

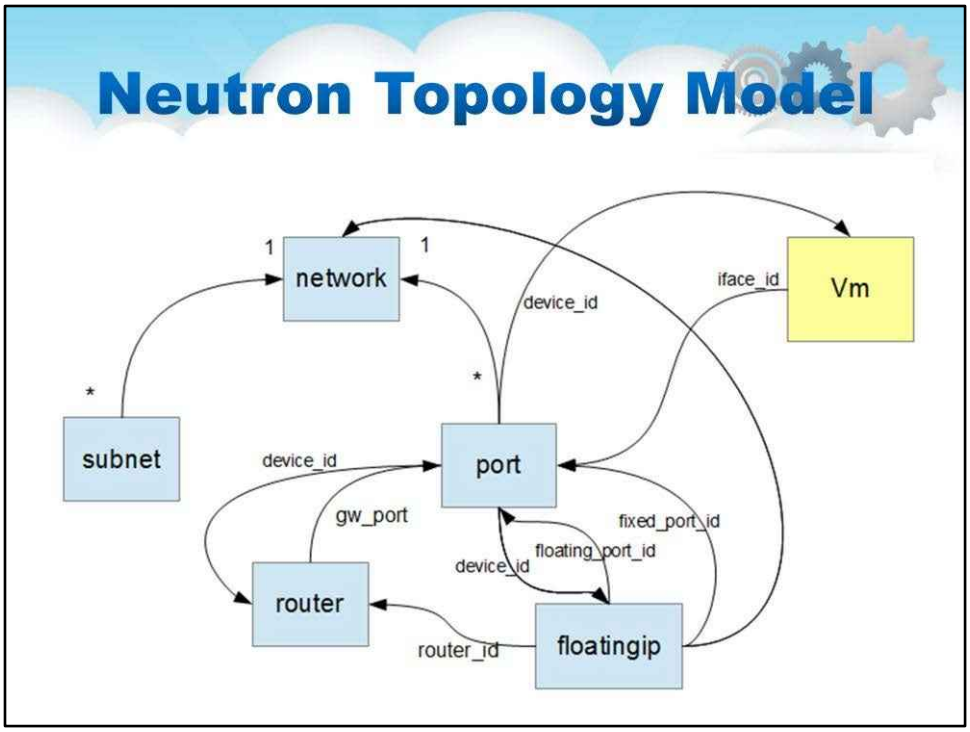
OpenStack Networking керує IP-адресами, дозволяючи використовувати виділені статичні IP-адреси або DHCP. Плаваючі IP-адреси дозволяють динамічно перенаправляти трафік до будь-якого з ваших обчислювальних ресурсів, що дає змогу перенаправляти трафік під час обслуговування або в разі збою.

Користувачі можуть створювати власні мережі, контролювати трафік і підключати сервери та пристрої до однієї або кількох мереж.

Підключена архітектура серверної частини дає користувачам змогу користуватися стандартним обладнанням або розширеними мережевими послугами від підтримуваних постачальників.

Адміністратори можуть скористатися перевагами програмно-визначеної мережевої технології (SDN), як-от OpenFlow, щоб забезпечити високий рівень мультитенантності та великого масштабу.

OpenStack Networking має структуру розширення, яка дозволяє розгортати та керувати додатковими мережевими службами, такими як системи виявлення вторгнень (IDS), балансування навантаження, брандмауери та віртуальні приватні мережі (VPN).



Neutron забезпечує «мережеве підключення як послугу» між інтерфейсними пристроями, керованими іншими службами OpenStack (швидше за все, Nova). Служба працює, дозволяючи користувачам створювати власні мережі, а потім приєднувати до них інтерфейси. Як і багато інших служб OpenStack, Neutron легко налаштовується завдяки своїй архітектурі плагінів. Ці модулі підключаються до різного мережевого обладнання та програмного забезпечення. Таким чином, архітектура та розгортання можуть кардинально відрізнятися.

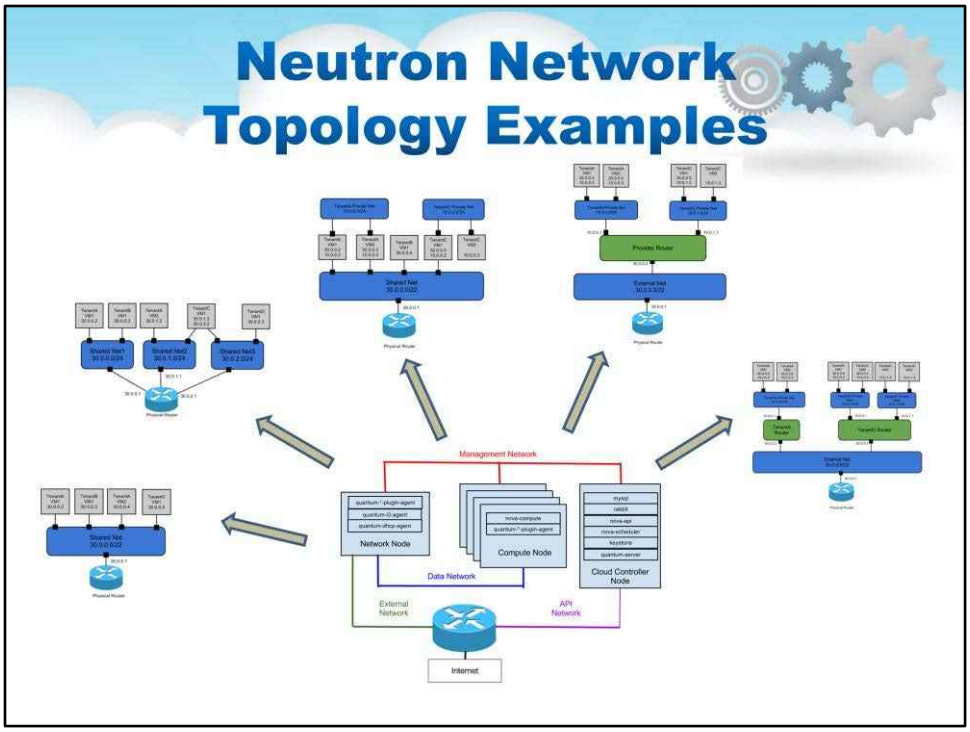
neutron-server приймає запити API, а потім направляє їх до відповідного плагіна Neutron для виконання.

Плагіни та агенти Neutron виконують фактичні дії, такі як підключення та відключення портів, створення мереж або підмереж і IP-адресування. Ці плагіни та агенти відрізняються залежно від постачальника та технологій, які використовуються в певній хмарі. Neutron поставляється з плагінами та агентами для: віртуальних і фізичних комутаторів Cisco, продуктів NEC OpenFlow, Open vSwitch, мостів Linux, мережевої операційної системи Ryu та VMware NSX.

Загальними агентами є L3 (рівень 3), DHCP (динамічна IP-адресація хоста) і спеціальний агент плагіна.

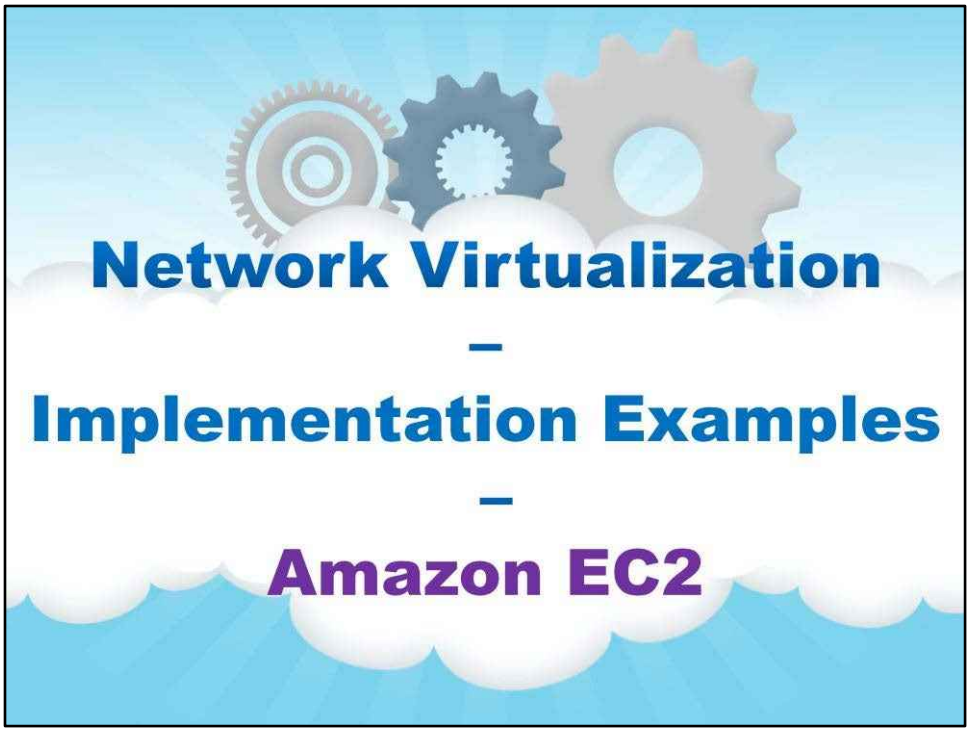
Більшість інсталяцій Neutron також використовуватиме чергу обміну повідомленнями для маршрутизації інформації між нейтрон-сервером і різними агентами, а також базу даних для зберігання стану мережі для окремих плагінів.

Neutron в основному взаємодіятиме з Nova, де він забезпечуватиме мережі та підключення для своїх примірників.

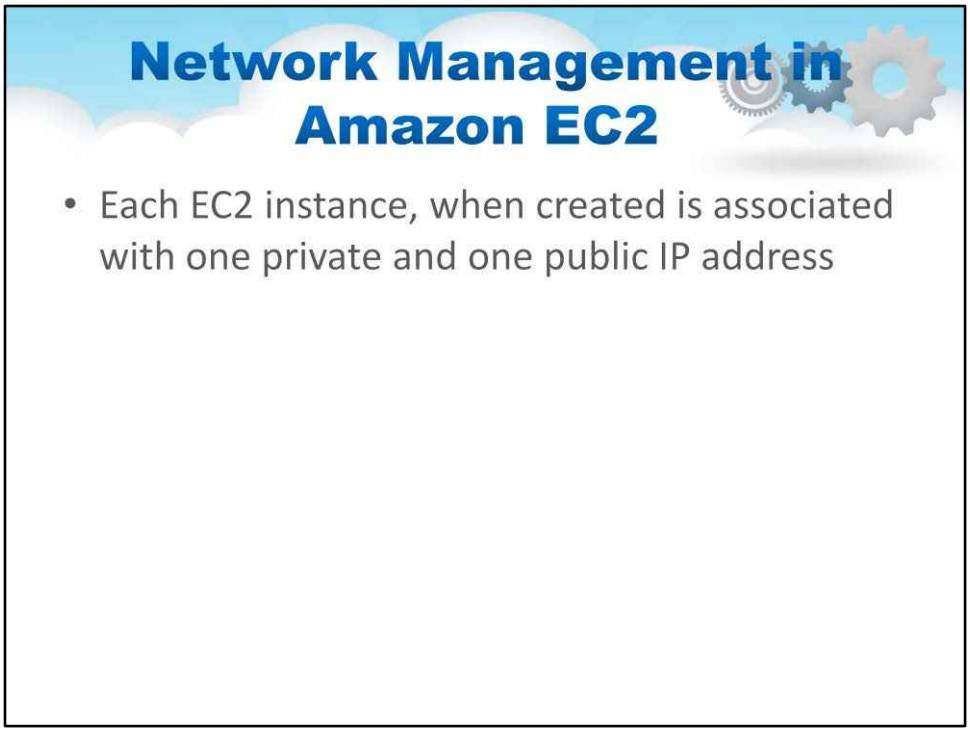


На цьому слайді показано, як мережеве програмне забезпечення на основі Neutron може бути застосоване для створення багатьох різних мережевих топологій для користувача

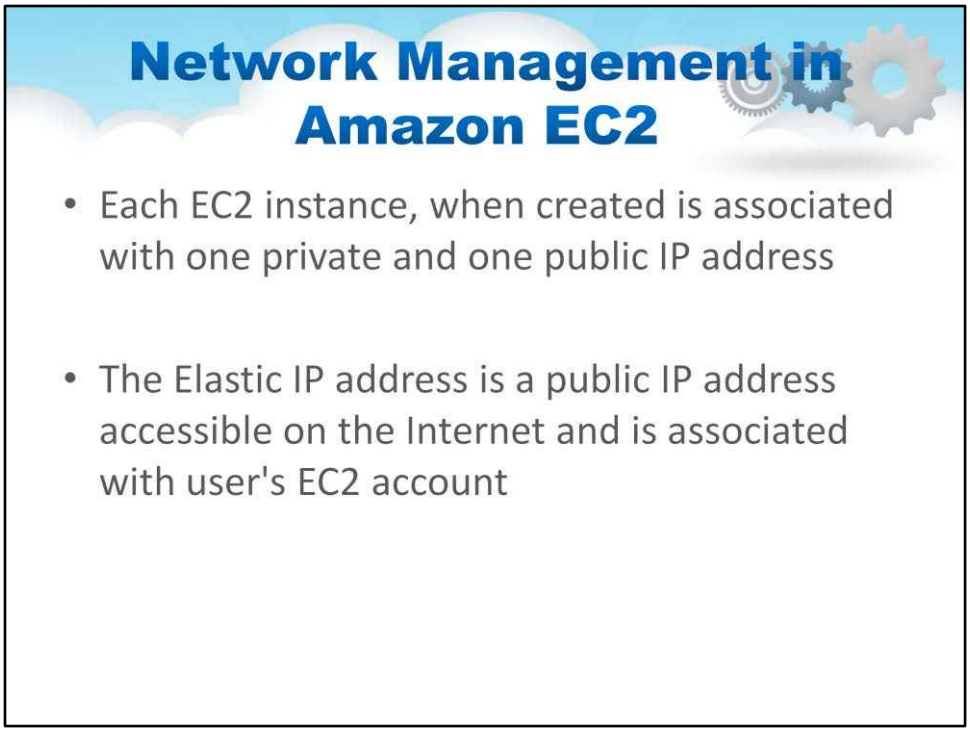
Зверніть увагу на можливість створення компонентів віртуального комутатора, а також кількох сегментів мережі.


Network Virtualization
–
Implementation Examples
–
Amazon EC2

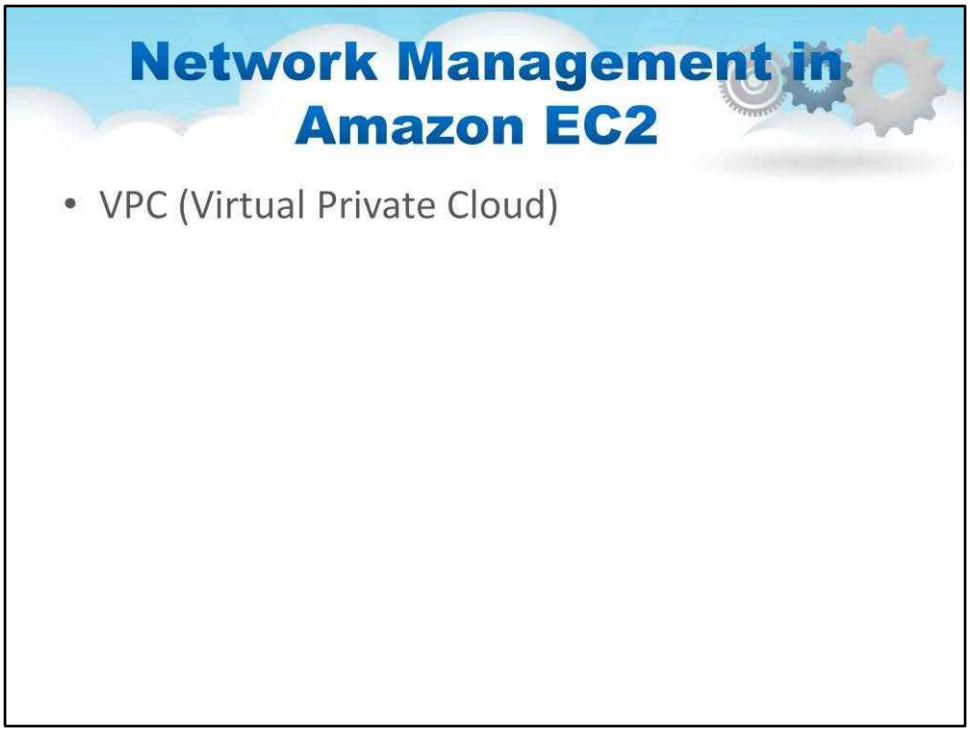
Amazon EC2



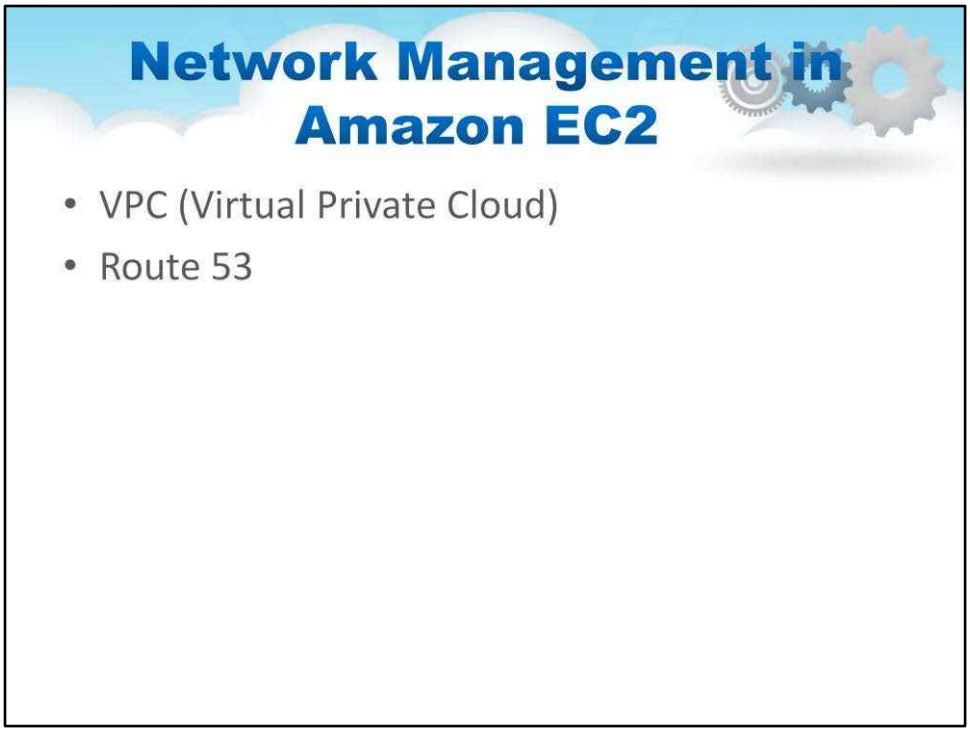
- Приватна IP-адреса використовується в EC2 LAN (локальна мережа)
- Загальнодоступна IP-адреса використовується для доступу до віртуальної машини користувача через Інтернет
- Загальнодоступна IP-адреса рекламується в Інтернеті та має відображення 1:1 NAT (перетворення мережевих адрес) на приватну IP-адресу екземпляра EC2



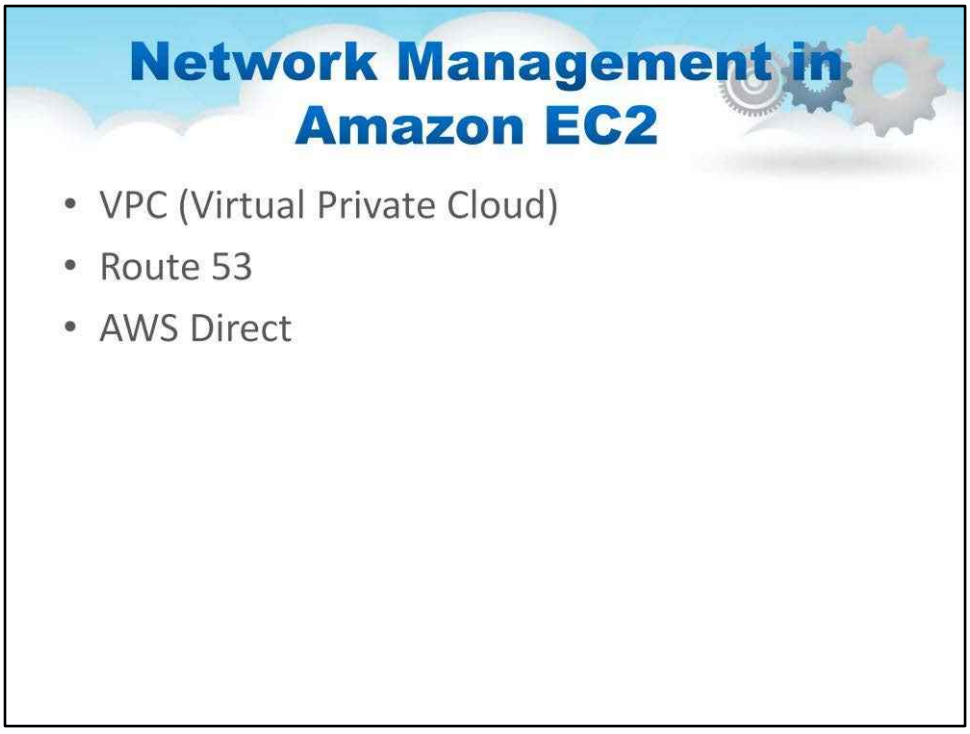
- Користувач може пов’язати цю IP-адресу з будь-яким конкретним екземпляром EC2, орендованим із цим обліковим записом
- Відображення можна динамічно змінювати на інший екземпляр EC2, зокрема на екземпляр для заміни у разі збою запущеної віртуальної машини.



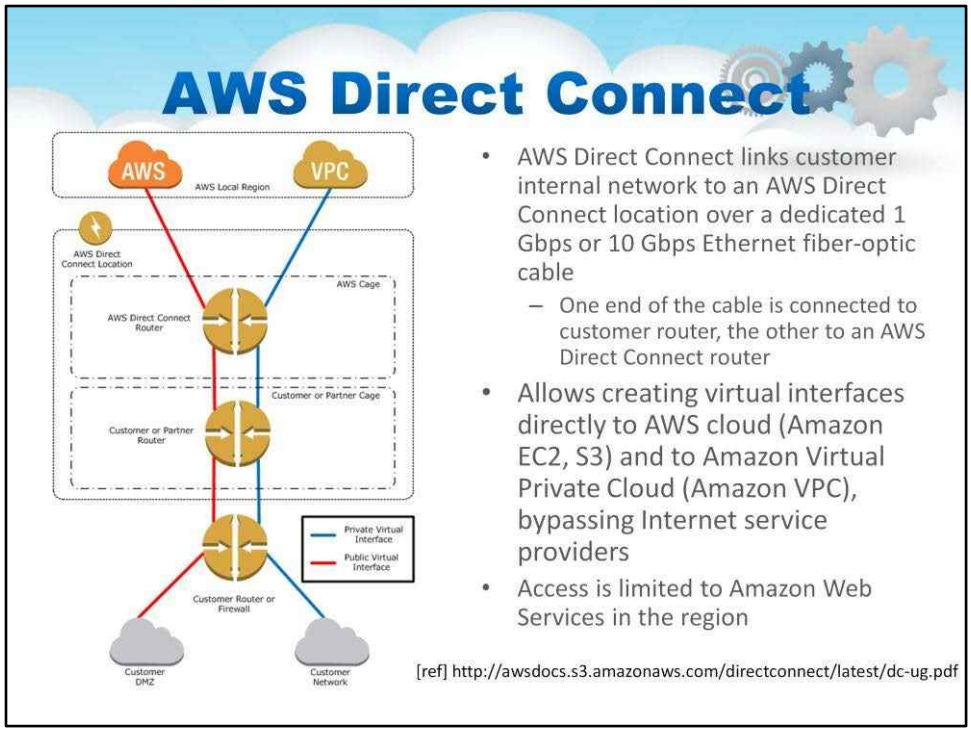
VPC (віртуальна приватна хмара) дозволяє організаціям використовувати ресурси AWS разом із наявною інфраструктурою, що з’єднує AWS VPC іорганізаціїмережі через VPN



Amazon Route 53 — це служба DNS (сервер доменних імен), яку надає Amazon для зіставлення IP-адрес екземплярів EC2 із доменними іменами.



AWS Direct Connect для спеціального підключення до хмари AWS

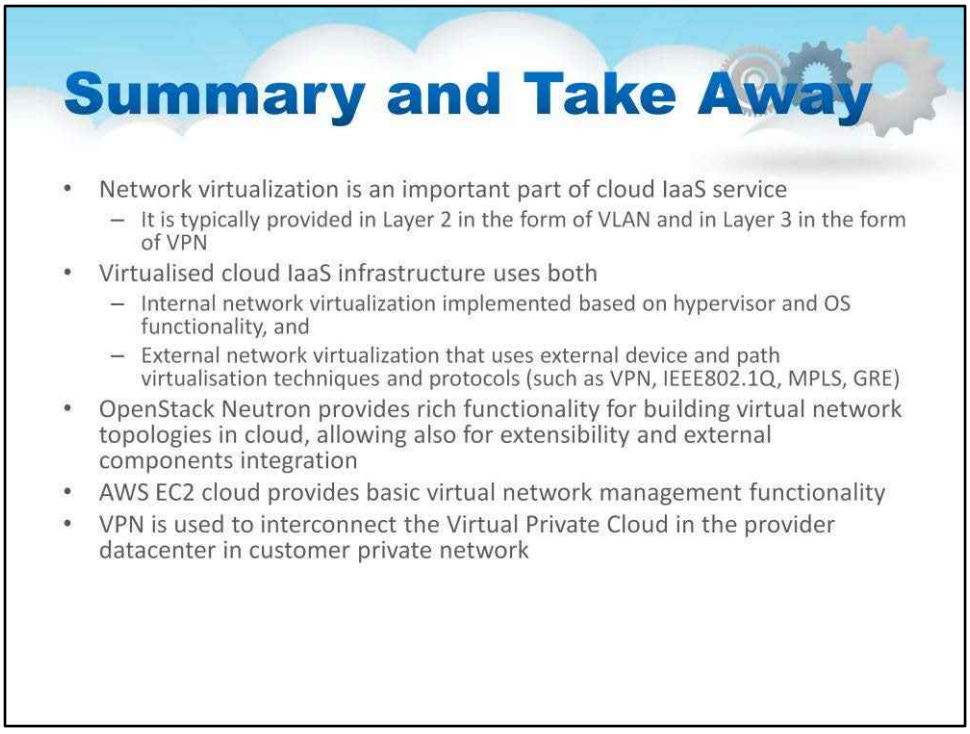


AWS пропонує ще один варіант для більш тісного підключення до хмари AWS. Припустімо, що ваш власний центр обробки даних компанії або принаймні частина вашої інфраструктури, яка потребує можливості переповнення хмарного типу, про яку говорилося на попередньому слайді, насправді є простором спільного розміщення в загальнодоступному центрі обробки даних. Далі припустимо, що AWS має частину своєї інфраструктури в тому самому центрі обробки даних. Це не випадково, тому що фірмові мегацентри обробки даних у таких місцях, як Нью-Йорк, Вашингтон, Кремнієва долина, Токіо та Лондон – це те місце, де варто бути, і це також місця, де є AWS. Отже, якщо ваша система знаходиться в тому самому центрі даних, що й AWS, чому

не просто підключатися до них безпосередньо? Ось що таке пряме з’єднання AWS. Один кінець кабелю під’єднано до маршрутизатора користувача, а інший – до маршрутизатора AWS Direct Connect. На рівні Ethernet VLAN встановлюється між двома маршрутизаторами. Рівень 3 — це з’єднання, яке використовує Direct Connect, і це схоже на взаємодію одного провайдера з іншим провайдером, тобто вони обидва мають номери автономної системи (AS) і використовують протокол BGP (Border Gateway Protocol) для IP-маршрутизації рівня 3. Загальнодоступний Інтернет (або решта IP-обміну в центрі обробки даних, якщо на те пішло) повністю обходиться.

AWS має Direct Connect у більшості регіонів

CoreSite NY1 і NY2 на сході США (Вірджинія)



У цьому уроці ми розглянули

Віртуалізація мережі є важливою частиною хмарної служби IaaS

Зазвичай він надається на рівні 2 у формі VLAN і на рівні 3 у формі VPN

Віртуалізована хмарна інфраструктура IaaS використовує обидва

Внутрішня віртуалізація мережі, реалізована на основі гіпервізора та функціональності ОС, а також

Віртуалізація зовнішньої мережі, яка використовує методи та протоколи віртуалізації зовнішніх пристроїв і шляхів (наприклад, VPN, IEEE802.1Q, MPLS, GRE). OpenStack Neutron надає багаті функціональні можливості для побудови топології віртуальної мережі в хмарі, а також забезпечує розширення та інтеграцію зовнішніх компонентів

Хмара AWS EC2 забезпечує базові функції керування віртуальною мережею VPN використовується для з’єднання віртуальної приватної хмари в центрі обробки даних постачальника в приватній мережі клієнта

Хмарні обчислення

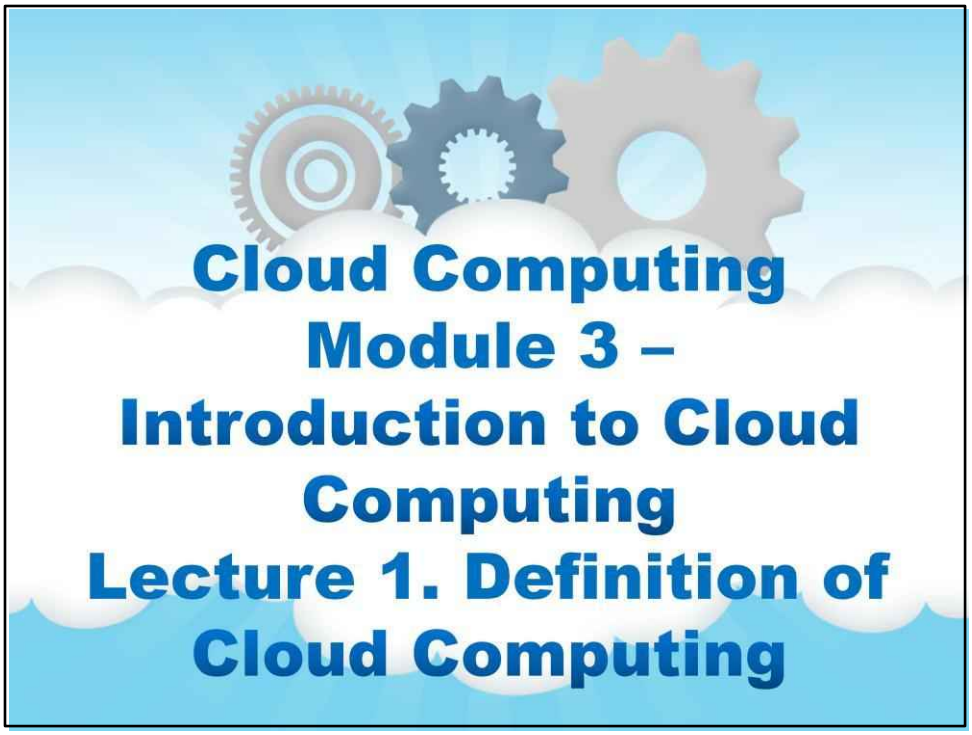
Лекційний посібник

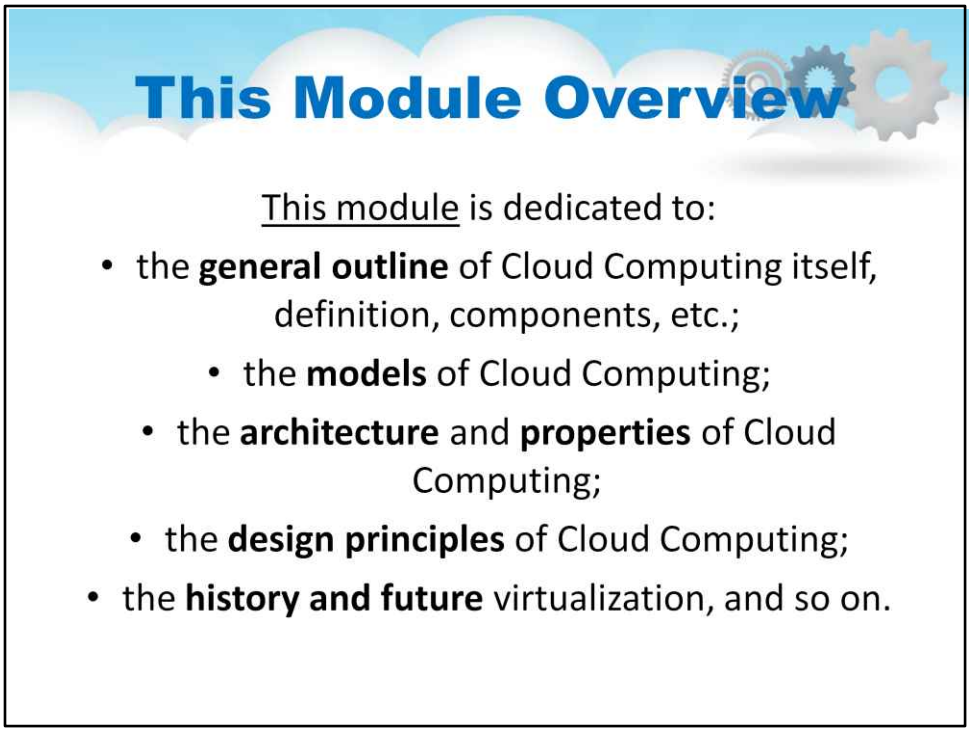
Том 3

Модуль 3

Вступ до хмарних обчислень

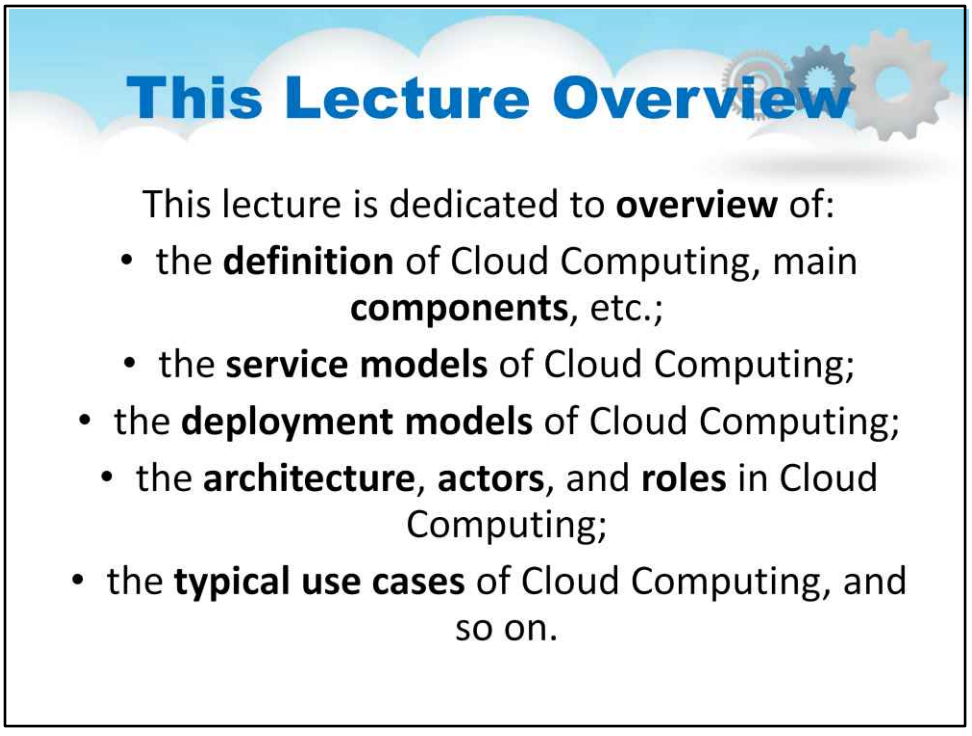
Зміст


Cloud Computing
Module 3 –
Introduction to Cloud
Computing
Lecture 1. Definition of
Cloud Computing



Модуль 3. Вступ до

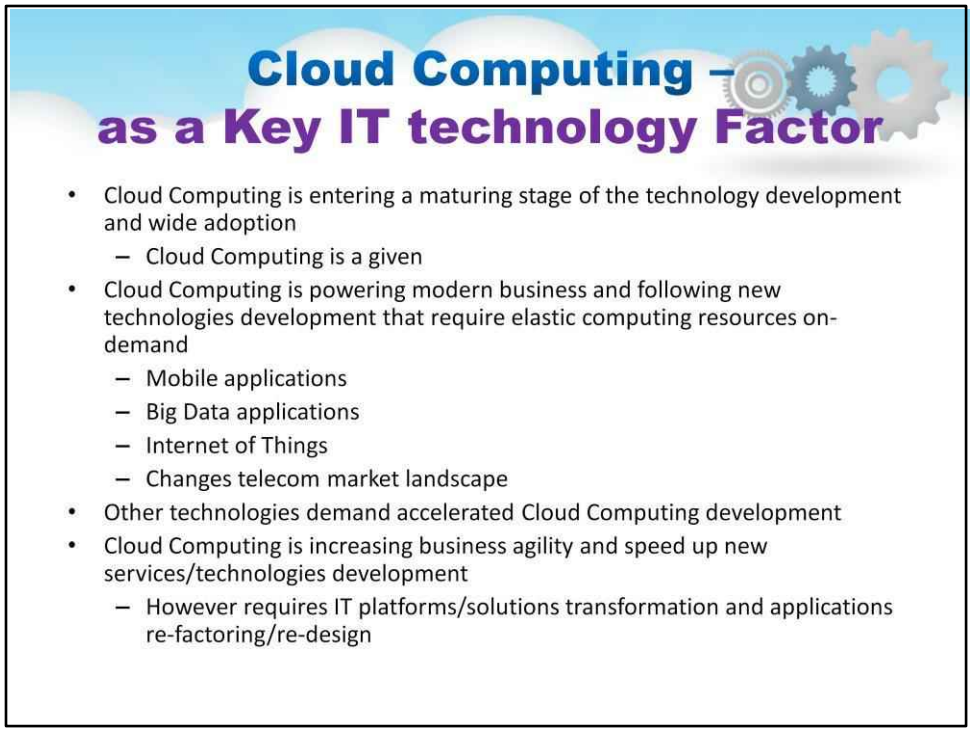
Хмарні обчислення



Лекція 1. Визначення хмари

обчислювальна техніка

Огляд



Перш за все, хмарні обчислення є ключовим технологічним фактором індустрії інформаційних технологій.

Хмарні обчислення вступають у стадію розвитку технологій і широкого впровадження

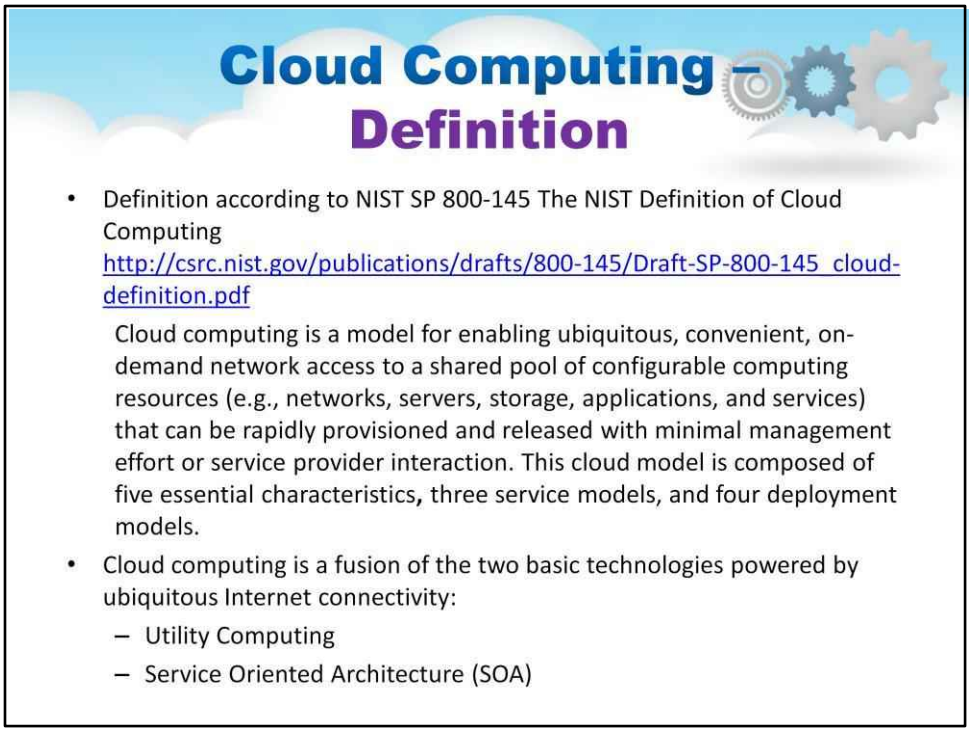
- Хмарні обчислення вже є даною сутністю.

Хмарні обчислення живлять сучасний бізнес і слідкують за розвитком нових технологій, які вимагають еластичних обчислювальних ресурсів на вимогу

- Мобільні додатки
- Програми Big Data
- Інтернет речей
- Змінює ландшафт телекомунікаційного ринку

Інші технології вимагають прискореного розвитку хмарних обчислень
Хмарні обчислення підвищують гнучкість бізнесу та прискорюють розвиток нових послуг/технологій

- Однак потрібна трансформація IT-платформ/рішень і рефакторинг/перепроектування програм



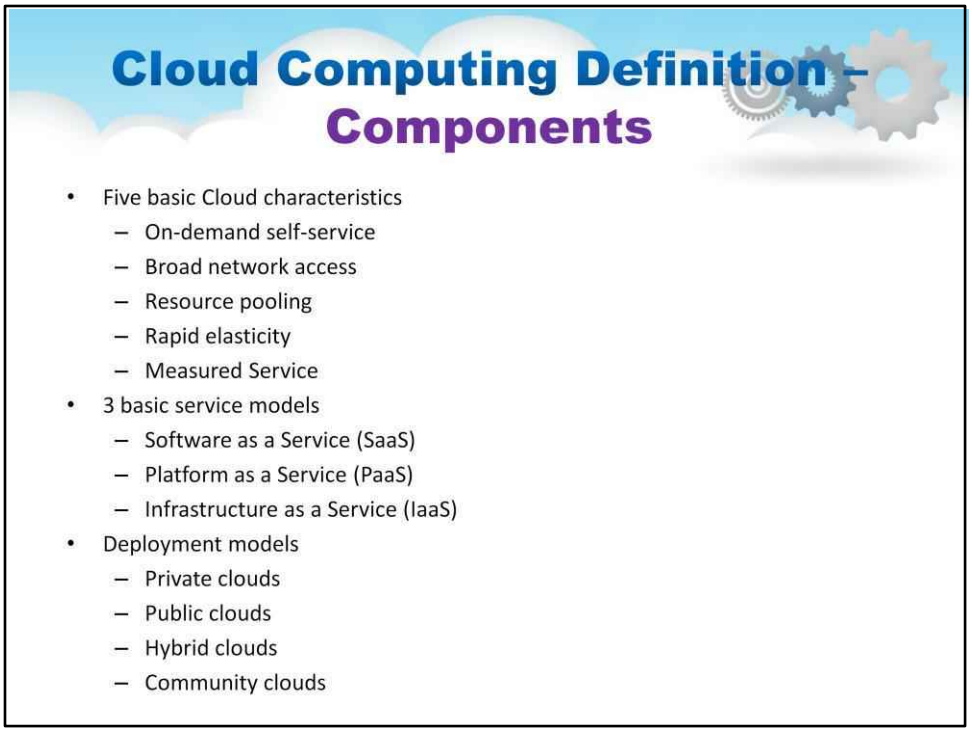
Визначення хмарних обчислень NIST (відповідно до NIST SP 800-145) (http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_clouddefinition.pdf)

Хмарні обчислення — це модель для забезпечення повсюдного, зручного мережевого доступу на вимогу до спільного пулу конфігурованих обчислювальних ресурсів (наприклад, мереж, серверів, сховищ, програм і служб), які можна швидко надати та вивільнити з мінімальними зусиллями керування або взаємодія постачальника послуг.

Ця хмарна модель складається з п’яти основних характеристик,три моделі обслуговування та чотири моделі розгортання.

Хмарні обчислення – це злиття двох основних технологій, що базуються на повсюдному підключенні до Інтернету:

- Службові обчислення
- Сервісно-орієнтована архітектура (SOA)



Цей слайд містить список основних компонентів парадигми хмарних обчислень:

П'ять основних характеристик Cloud

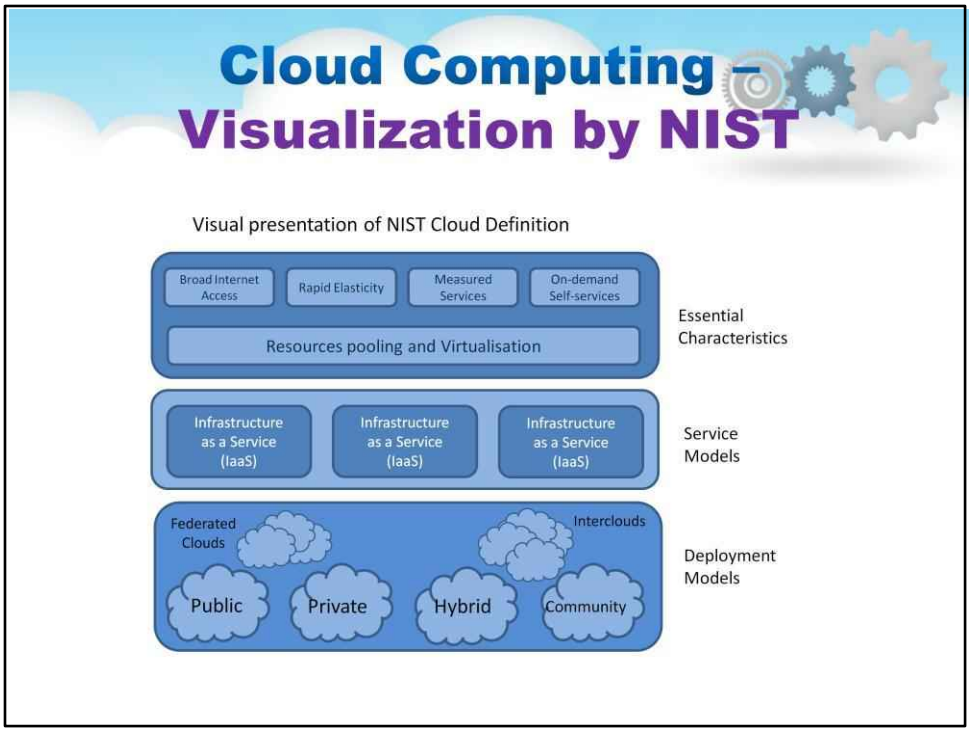
- Самообслуговування на вимогу
- Широкий доступ до мережі
- Об'єднання ресурсів
- Швидка еластичність
- Виміряне обслуговування

3 базові моделі обслуговування

- Програмне забезпечення як послуга
- (SaaS) Платформа як послуга (PaaS)
- Інфраструктура як послуга (IaaS)

Моделі розгортання

- Приватні хмари
- Громадські хмари
- Гібридні хмари
- Хмари спільноти



Цей слайд містить візуалізацію Cloud Computing та його основні компоненти:

Основні характеристики Cloud

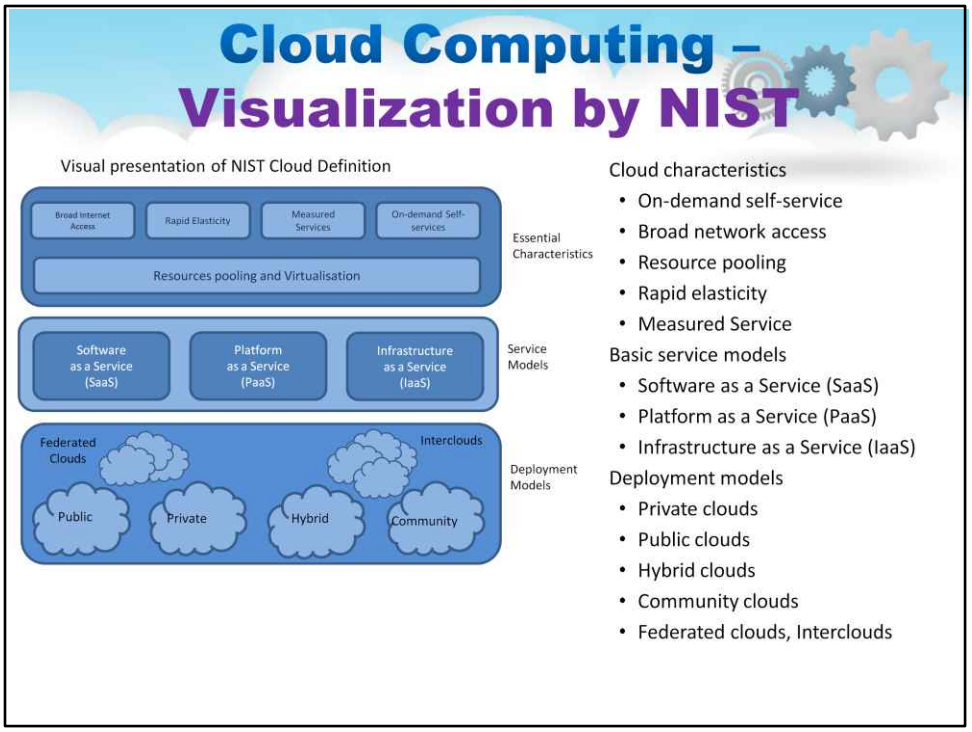
- Самообслуговування на вимогу
- Широкий доступ до мережі
- Об'єднання ресурсів
- Швидка еластичність
- Виміряне обслуговування

Основні моделі обслуговування

- Програмне забезпечення як послуга
- (SaaS) Платформа як послуга (PaaS)
- Інфраструктура як послуга (IaaS)

Найпопулярніші моделі розгортання

- Приватні хмари
- Громадські хмари
- Гібридні хмари
- Хмари спільноти



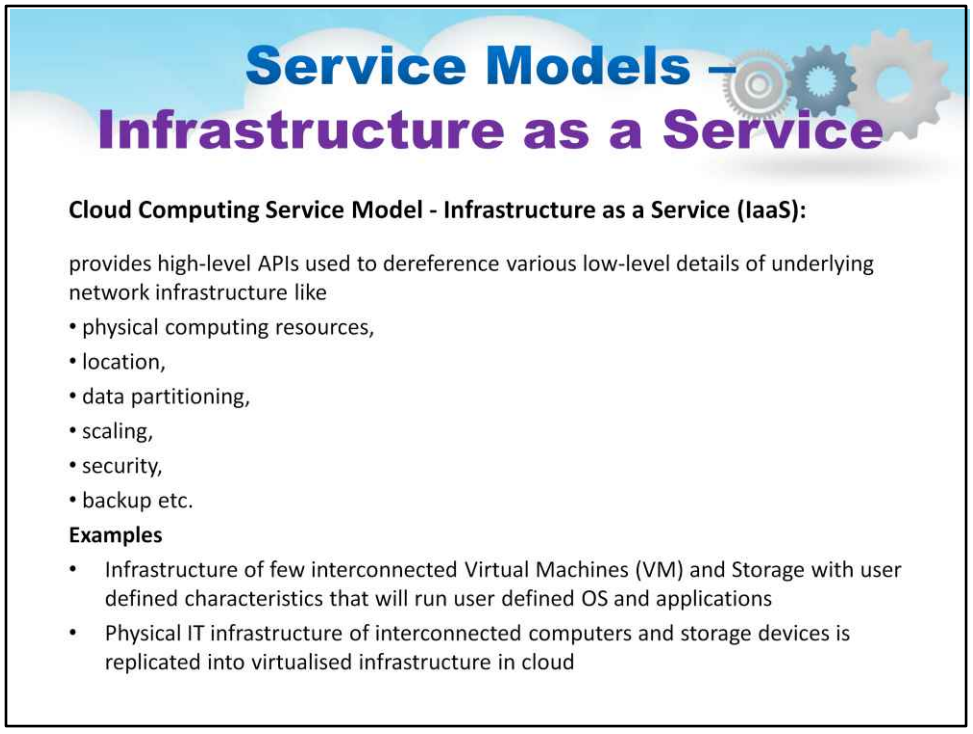
Відповідно до точки зору NIST:

Хмарні обчислення — це модель для забезпечення повсюдного, зручного мережевого доступу на вимогу до спільного пулу конфігурованих обчислювальних ресурсів (наприклад, мереж, серверів, сховищ, програм і служб), які можна швидко надати та вивільнити з мінімальними зусиллями керування або взаємодія постачальника послуг. Ця хмарна модель складається з п’яти основних характеристик**,**три моделі обслуговування та чотири моделі розгортання.

Тут ви можете побачити кумулятивний перегляд
- всі основні компоненти (в списку з правого боку) і
- візуалізації зв'язків між ними і (на схемі з правого боку).

Сервісні моделі

Тепер давайте розглянемо моделі обслуговування хмарних обчислень...



Модель служби хмарних обчислень – інфраструктура як послуга (IaaS):

надає API високого рівня, які використовуються для розіменування різних низькорівневих деталей базової мережевої інфраструктури, наприклад

- фізичні обчислювальні ресурси,
- Місцезнаходження,
- розділення даних,
- масштабування,
- безпека,
- резервне копіювання тощо

Приклади

- Інфраструктура кількох взаємопов’язаних віртуальних машин (VM) і сховищ із визначеними користувачем характеристиками, які запускатимуть визначені користувачем ОС і програми
- Фізична IT-інфраструктура взаємопов’язаних комп’ютерів і пристроїв зберігання даних копіюється у віртуальну інфраструктуру в хмарі

Можливість, що надається споживачу, полягає в забезпеченні обробки, зберігання, **мережі** та інші фундаментальні обчислювальні ресурси, де споживач може розгортати та запускати довільне програмне забезпечення, яке може включати операційні системи та програми.

Споживач не керує базовою хмарною інфраструктурою та не контролює її, але має контроль над операційними системами, сховищами, розгорнутими програмами та **можливо обмежений контроль вибраних мережевих компонентів (наприклад, брандмауери хоста)**

Споживач не керує базовою хмарою та не контролює її інфраструктури, але контролює операційні системи, сховище, розгорнуті програми таможливо обмежений контроль вибраних мережевих компонентів (наприклад, брандмауери хоста).

Модель обслуговування хмарних обчислень – платформа як послуга (PaaS):
надає платформу, яка дозволяє клієнтам розробляти, запускати та керувати програмами без складності створення та підтримки інфраструктури, яка зазвичай пов’язана з розробкою та запуском програми.

Приклади

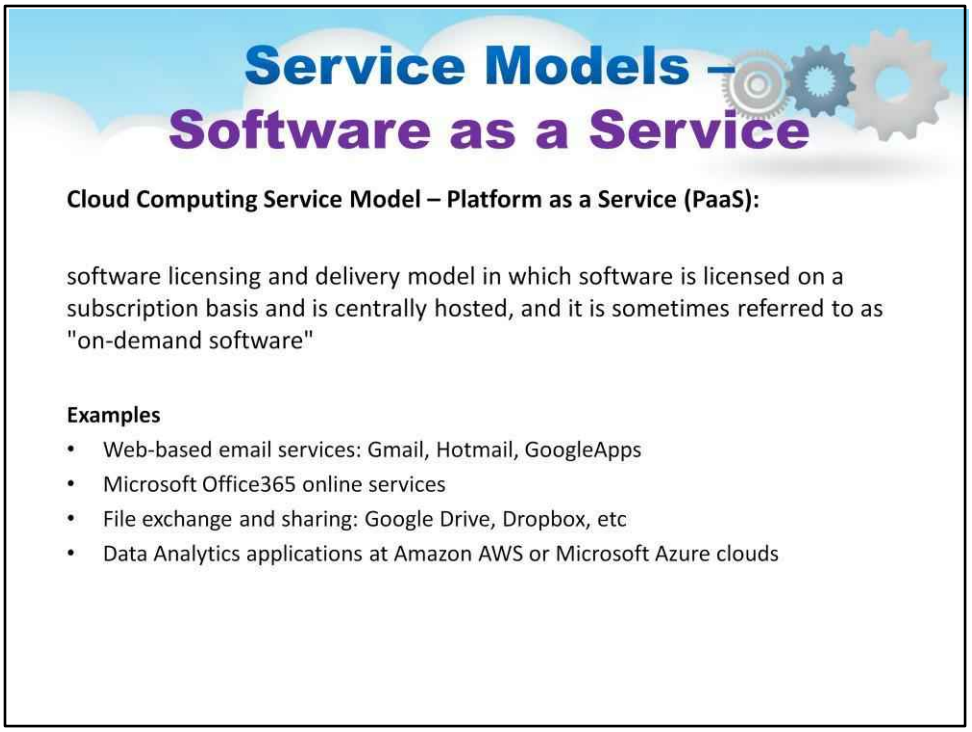
Широко використовувана платформа управління підприємством, наприклад CRM, надається корпоративним клієнтам на вимогу; його легко масштабувати залежно від робочого навантаження

Використання хмарної платформи PaaS для розробки та тестування нових програм

Використання хмарної платформи управління бізнес-процесами для запуску бізнес-процесів користувачів, наприклад, управління відносинами з клієнтами (CRM) або керування ланцюгом поставок (SCM)

Можливість, яка надається споживачеві, полягає в розгортанні в хмарній інфраструктурі створених споживачем або придбаних додатків, створених за допомогою мов програмування, бібліотеки, служби та інструменти, які підтримуються постачальником. Споживач не керує та не контролює базову хмарну інфраструктуру, включаючи мережу, сервери, операційні системи або сховище, але контроль над розгорнутими програмами та, можливо, параметрами конфігурації середовища розміщення програм PaaS можна доставити трьома способами:

- як загальнодоступна хмарна служба від постачальника, де споживач контролює розгортання програмного забезпечення з мінімальними параметрами конфігурації, а постачальник надає мережі, сервери, сховище, операційну систему (ОС), проміжне програмне забезпечення (наприклад, середовище виконання Java, середовище виконання .NET, інтеграцію тощо). .), бази даних та інші служби для розміщення програми споживача.
- як приватну службу (програмне забезпечення або пристрій) всередині брандмауера.
- як програмне забезпечення, розгорнуте в публічній інфраструктурі як послуга.



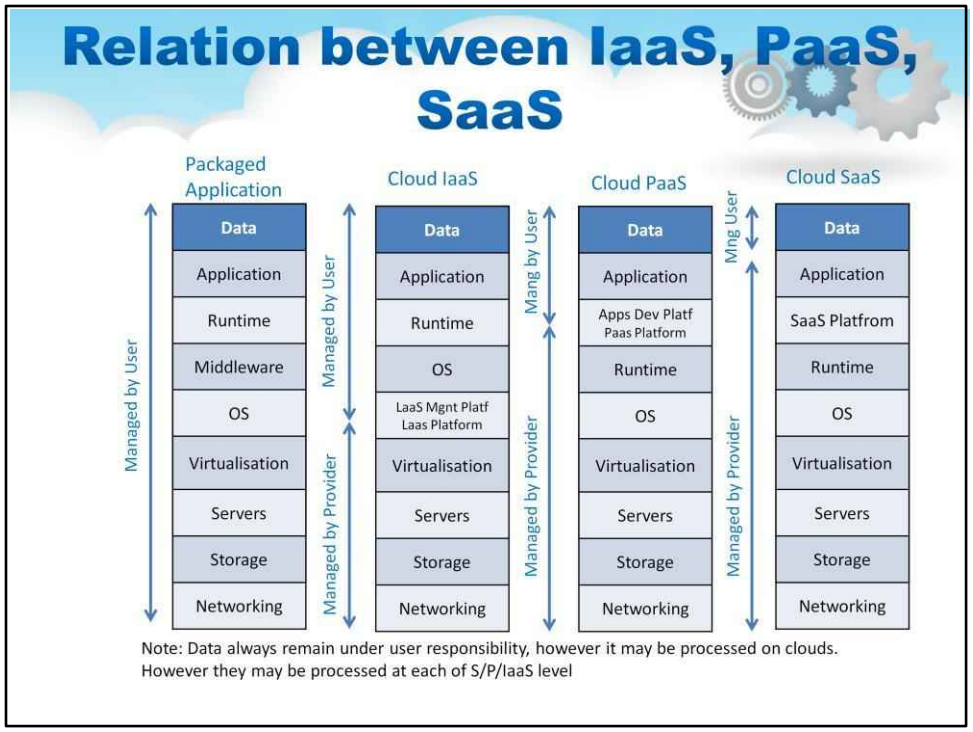
Модель обслуговування хмарних обчислень – платформа як послуга (PaaS):

забезпечує модель ліцензування та доставки програмного забезпечення, згідно з якою програмне забезпечення ліцензується на основі передплати та розміщується централізовано, і його іноді називають "програмним забезпеченням за вимогою"

Приклади

- Веб-служби електронної пошти: Gmail, Hotmail, GoogleApps
- Онлайн-сервіси Microsoft Office365
- Обмін та обмін файлами: Google Drive, Dropbox тощо
- Програми Data Analytics у хмарах Amazon AWS або Microsoft Azure

Можливість, яка надається споживачеві, полягає в тому, щоб використовувати програми провайдера, що працюють в хмарній інфраструктурі. Програми доступні з різних клієнтських пристроїв або через тонкий клієнтський інтерфейс, такий як веб-браузер (наприклад, веб-електронна пошта), або програмний інтерфейс. Споживач не керує базовою хмарною інфраструктурою, включаючи мережу, сервери, операційні системи, сховище або навіть окремі можливості програми, за винятком, можливо, обмежені параметри конфігурації програми для користувача



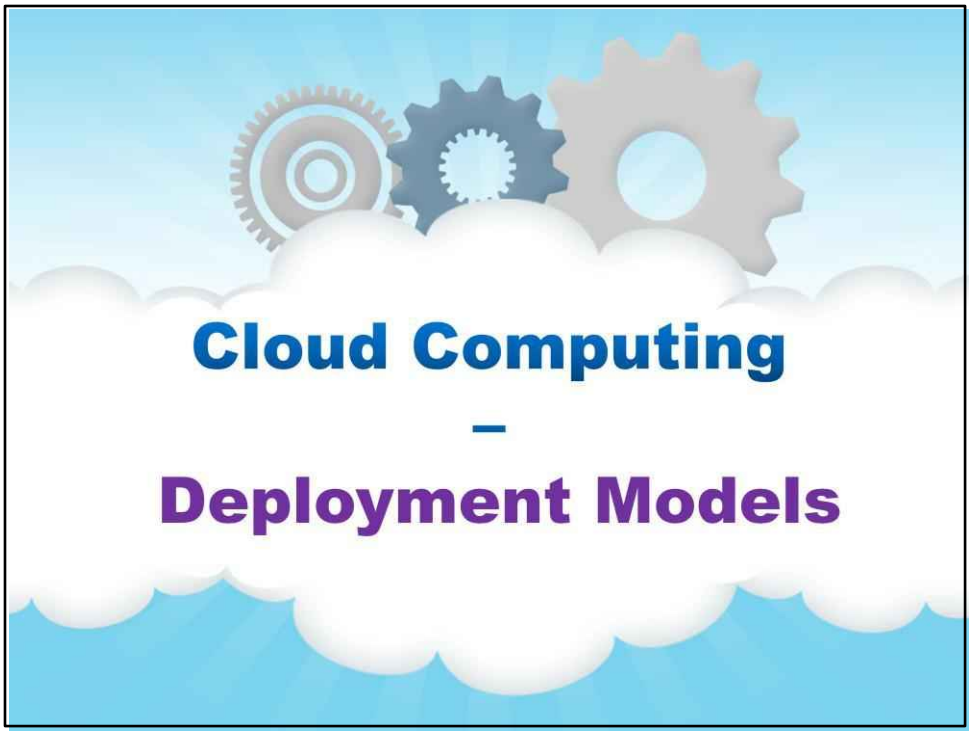
Моделі використання хмарних обчислень тісно пов’язані.

Для традиційної «фізично встановленої» упакованої програми **(це показано на 1 колонка з лівого боку)**, усім стеком розгортання керує користувач, як видно на першому блоці ілюстрації. Користувач несе відповідальність за надання повного сервера, включаючи мережу та сховище, а також програмне забезпечення операційної системи. Крім того, користувач надає необхідні бази даних і проміжне програмне забезпечення, будь-яке спеціальне середовище виконання (Java або Ruby) і, нарешті, програму. Дані програми також є відповідальністю (і власністю) користувача.

Модель Cloud IaaS **(це показано на 2 колонка з лівого боку)** це суттєво змінює, полегшуючи значну частину принаймні того, що вважалося б «фізичною» інфраструктурою, це показано на другому блоці ілюстрації. Програмне забезпечення для керування хмарою (іноді називається Cloud OS) показано на цій діаграмі як «IaaS Mngt Platf». Користувач отримує апаратне забезпечення у віртуалізованому вигляді від IaaS CSP і турбується лише про програмний стек, програму та дані.

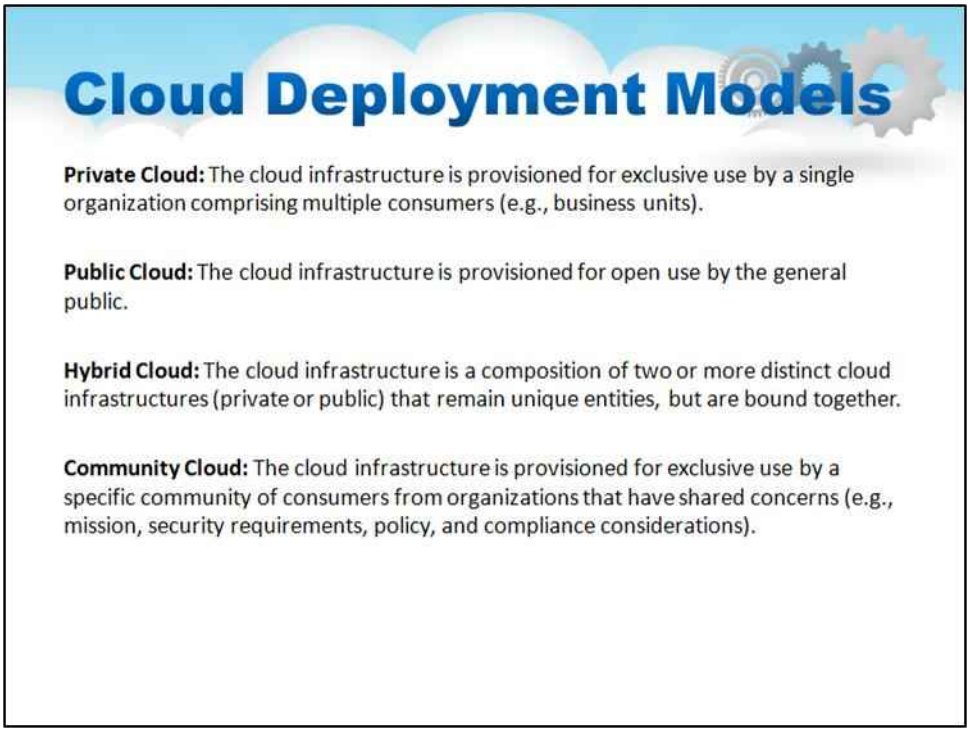
Модель Cloud PaaS **(це показано на 3 колонка з лівого боку)** далі пропонує проміжне програмне забезпечення, середовище виконання програми та якомога більше «програмної» інфраструктури, створюючи зручний контейнер для розробника програми. Хоча це вимагає від розробника реструктуризації, якщо не переписування програми на рівні вихідного коду для інтерфейсу з PaaS, це обмеження додасть нові переваги масштабованості та доступності додатку.

Модель Cloud SaaS **(це показано на 4 колонка з лівого боку)** це використання лише самої програми через браузер або веб-службу.



Моделі розгортання

Тепер давайте розглянемо моделі розгортання хмарних обчислень...



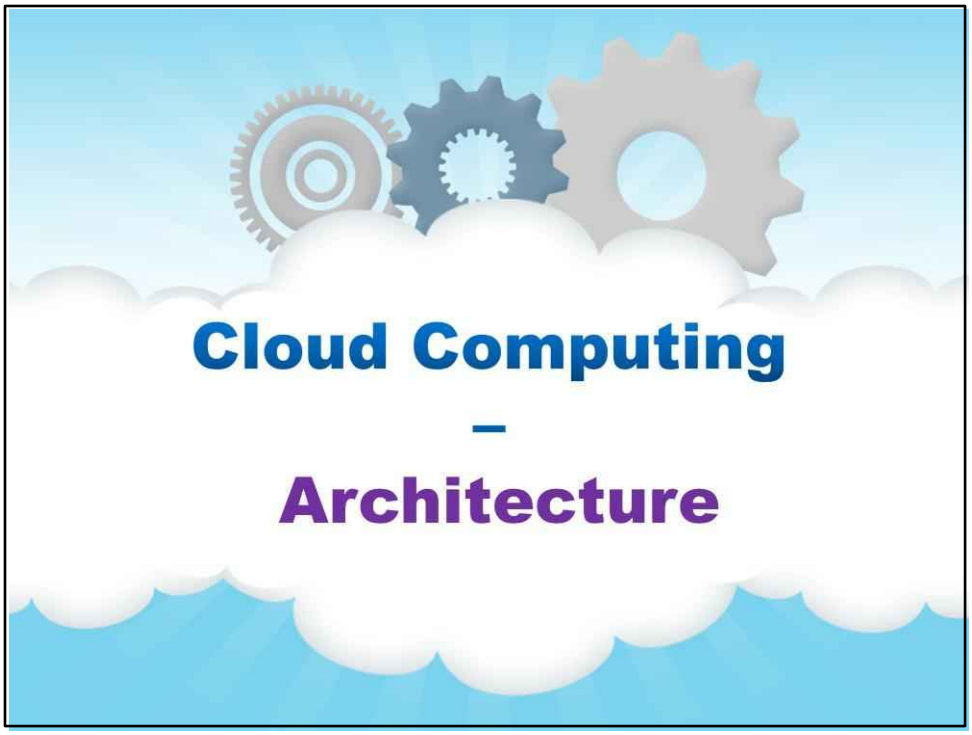
Цей слайд містить список основних моделей розгортання хмарних обчислень.

Приватна хмара:Хмарна інфраструктура надається для ексклюзивного використання однією організацією, що складається з кількох споживачів (наприклад, бізнес-підрозділів). Він може належати, управлятися та управлятися організацією, третьою стороною або деякою їх комбінацією, і він може існувати в приміщенні або за його межами.

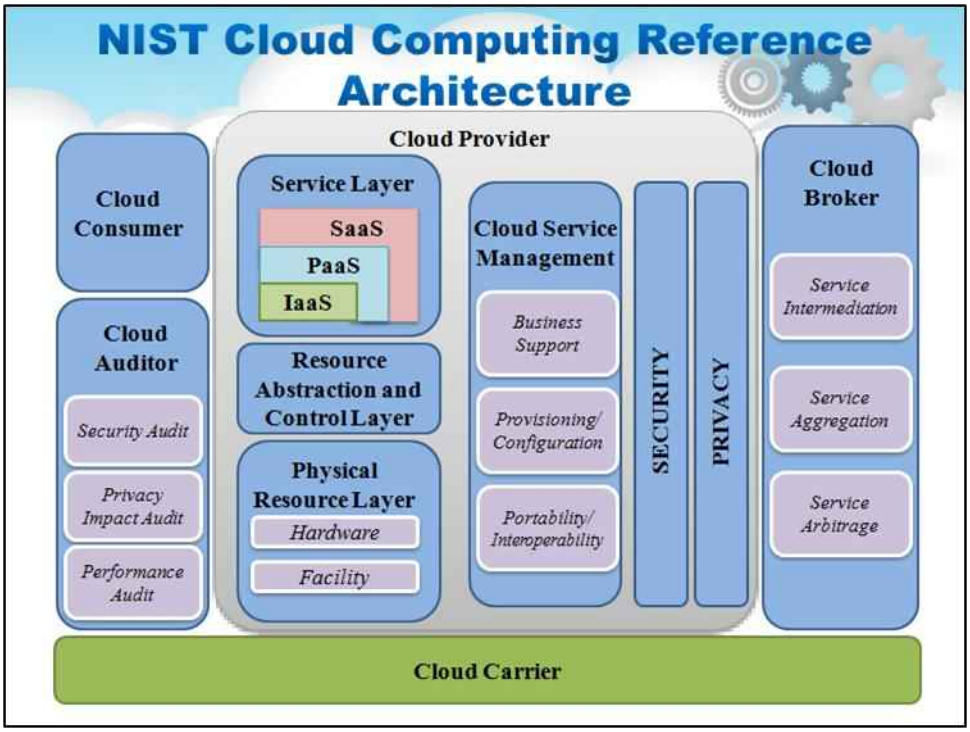
Громадська хмара:Хмарна інфраструктура надається для відкритого використання широким загалом. Воно може належати, управлятися та управлятися діловою, академічною чи державною організацією або деякою їх комбінацією. Він існує на території хмарного провайдера.

Гібридна хмара:Хмарна інфраструктура — це композиція з двох або більше різних хмарних інфраструктур (приватних, спільнотних або загальнодоступних), які залишаються унікальними сутностями, але пов’язані між собою стандартизованою або запатентованою технологією, яка забезпечує переносимість даних і програм (наприклад, розрив хмари для балансування навантаження між хмари).

Хмара спільноти:Хмарна інфраструктура надається для ексклюзивного використання певною спільнотою споживачів з організацій, які мають спільні інтереси (наприклад, місія, вимоги безпеки, політика та міркування щодо відповідності). Він може належати, управлятися та управлятися однією чи декількома організаціями в спільноті, третьою стороною або деякою їх комбінацією, і він може існувати в приміщеннях або поза ними.



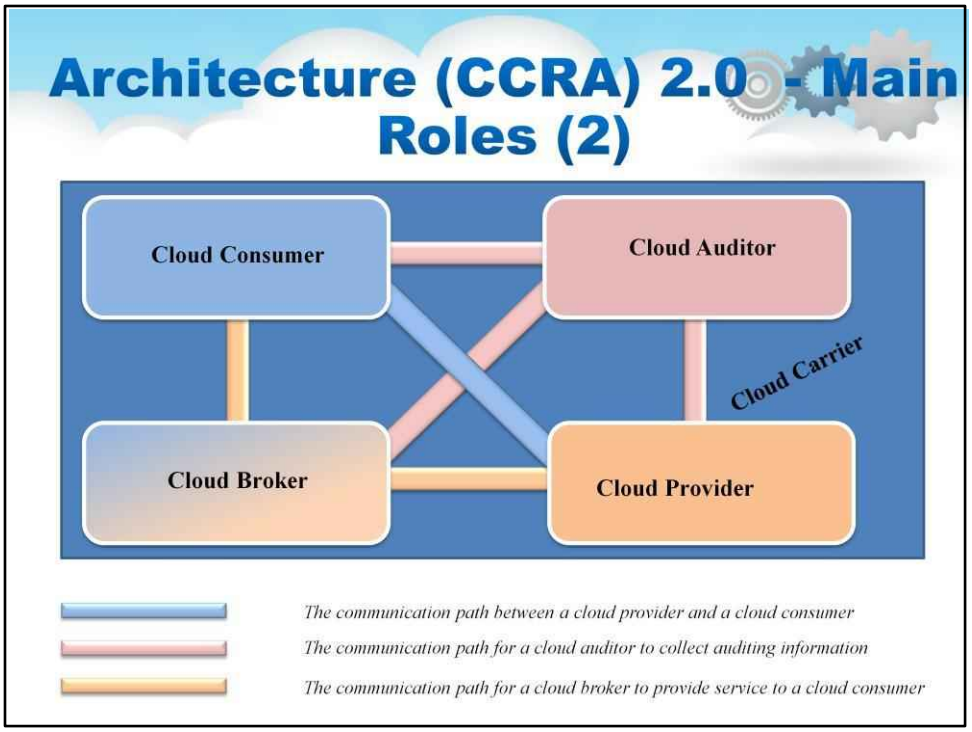
Архітектура



Стандартизація була дуже важливою з самого початку розвитку хмарних обчислень. І для постачальників хмарних послуг (через масштаб їхньої інфраструктури та засобів), і для споживачів хмарних послуг важливо дозволити їх взаємодію з іншими службами.

Тут ми посилаємося на відповідні стандарти Національного інституту стандартів і технологій США (NIST), які визначають технологію хмарних обчислень і еталонну архітектуру хмарних обчислень.

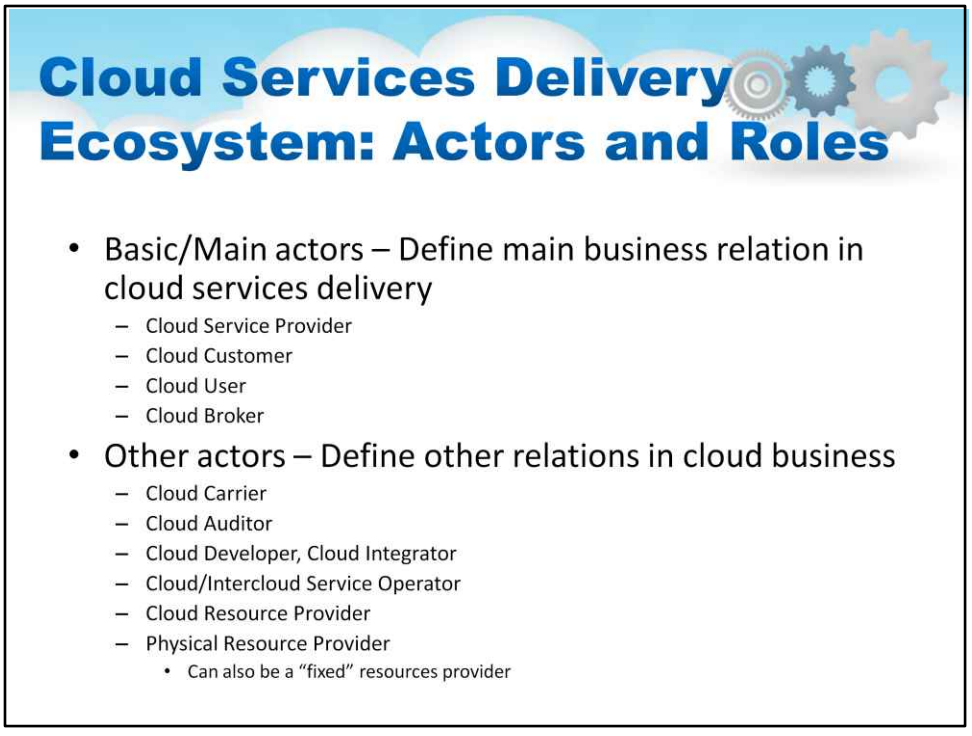
NIST активно сприяє розвитку практик хмарних обчислень, які підтримують сумісність, портативність і вимоги безпеки, які є відповідними та досяжними для важливих сценаріїв використання. З моменту першої публікації в даний час загальноприйнятого визначення хмари NIST у 2008 році NIST веде широку міжнародно визнану діяльність щодо визначення концептуальної та стандартної бази в хмарних обчисленнях, що призвело до публікації наступних документів, які створюють міцну основу для розробки хмарних сервісів і пропозиція: NIST SP 800-145, визначення NIST хмарних обчислень та іншу офіційну документацію.



На слайді представлено високорівневий огляд еталонної архітектури хмарних обчислень NIST (CCRA), який визначає основних учасників (споживач хмарних технологій, постачальник хмарних послуг, хмарний аудитор, хмарний посередник і хмарний оператор), їхні дії та функції в хмарних обчисленнях. Споживач хмари може запитувати хмарні послуги у хмарного постачальника безпосередньо або через хмарного брокера. Хмарний аудитор проводить незалежні перевірки та може зв’язуватися з іншими для збору необхідної інформації.

Запропонована архітектура підходить для багатьох цілей, де продуктивність мережі не є критичною, але її потрібно розширити за допомогою явного надання та керування мережевими службами, коли хмарні програми мають вирішальне значення для затримки мережі, наприклад у випадку корпоративних програм, бізнес-транзакцій, управління кризовими ситуаціями тощо

Еталонна архітектура хмарних обчислень NIST (CCRA) визначає низку зацікавлених сторін і учасників, які можуть бути розширені на основі базового аналізу варіантів використання. Слайд ілюструє деякі з тих, хто входить до такого списку, і показує відносини між зацікавленими сторонами та акторами.



Розглянемо весь список учасників екосистеми Cloud Computing для надання послуг:

Основні/головні учасники – визначте основні ділові відносини в наданні хмарних послуг

Постачальник хмарних послуг

Хмарний клієнт

Хмарний користувач

Хмарний брокер

Інші учасники – визначте інші відносини в хмарному бізнесі

Хмарний перевізник

Хмарний аудитор

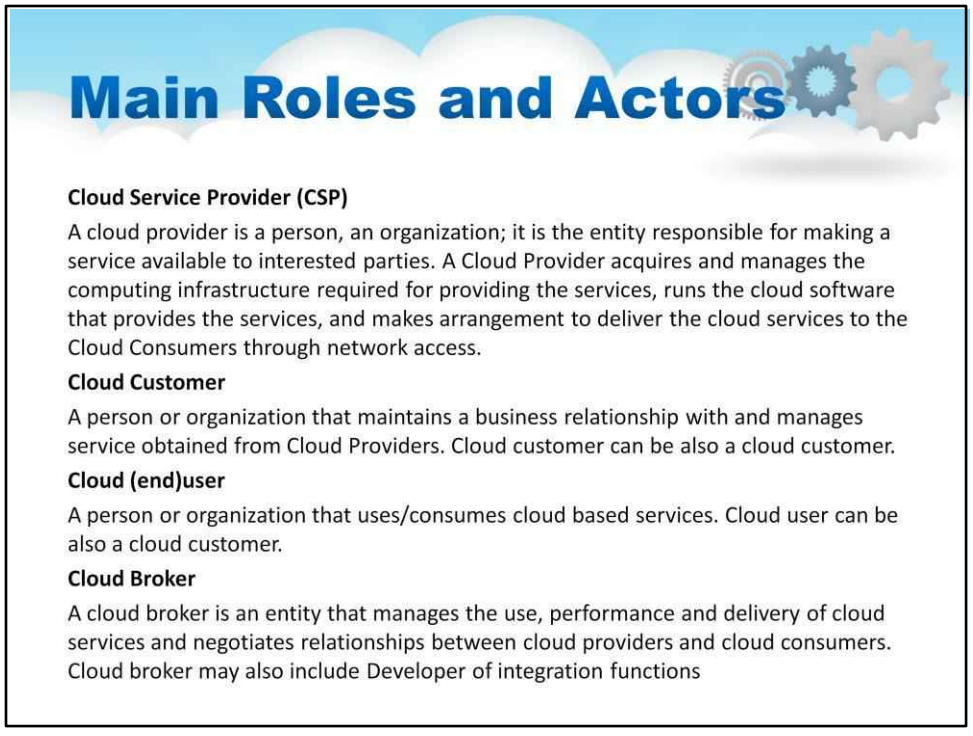
Хмарний розробник, хмарний інтегратор.

Оператор хмарних/інтерхмарних послуг.

Постачальник хмарних ресурсів

Постачальник фізичних ресурсів

Також може бути «фіксованим» постачальником ресурсів



Головні актори та їх ролі наступні:

Постачальник хмарних послуг (CSP)

Хмарний провайдер – це особа, організація; це організація, відповідальна за надання послуги доступній зацікавленим сторонам. Постачальник хмарних технологій отримує обчислювальну інфраструктуру, необхідну для надання послуг, і керує нею, запускає хмарне програмне забезпечення, яке надає послуги, і забезпечує надання хмарних послуг Споживачам хмарних послуг через доступ до мережі.

Хмарний клієнт

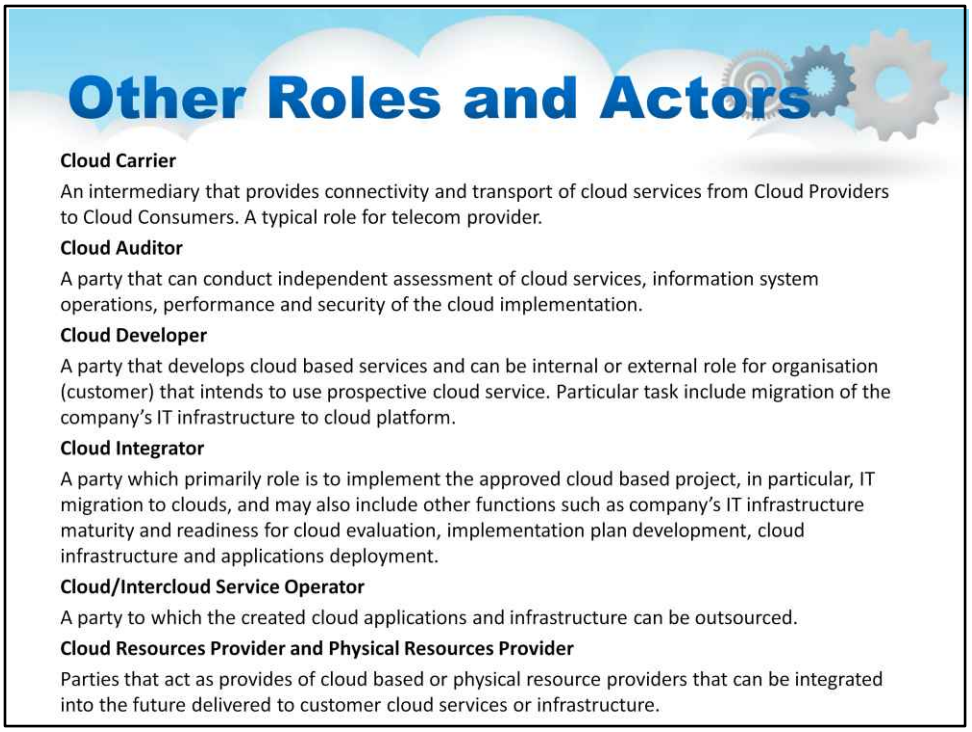
Особа чи організація, яка підтримує ділові відносини з послугами, отриманими від хмарних постачальників, і керує ними. Хмарний клієнт також може бути хмарним клієнтом.

Хмарний (кінцевий) користувач

Особа або організація, яка використовує/споживає хмарні послуги. Користувач хмари також може бути клієнтом хмари.

Хмарний брокер

Хмарний брокер — це організація, яка керує використанням, продуктивністю та доставкою хмарних послуг і веде переговори про відносини між постачальниками та споживачами хмарних технологій. Хмарний брокер також може включати розробника функцій інтеграції



Серед інших акторів і ролей:

Cloud Carrier

Посередник, який забезпечує підключення та передачу хмарних послуг від хмарних постачальників до хмарних споживачів. Типова роль оператора зв'язку.

Хмарний аудитор

Сторона, яка може проводити незалежну оцінку хмарних сервісів, роботи інформаційної системи, продуктивності та безпеки впровадження хмари. **Хмарний розробник**

Сторона, яка розробляє хмарні послуги та може виконувати внутрішню або зовнішню роль для організації (замовника), яка має намір використовувати перспективну хмарну службу. Серед окремих завдань – міграція IT-інфраструктури компанії на хмарну платформу.

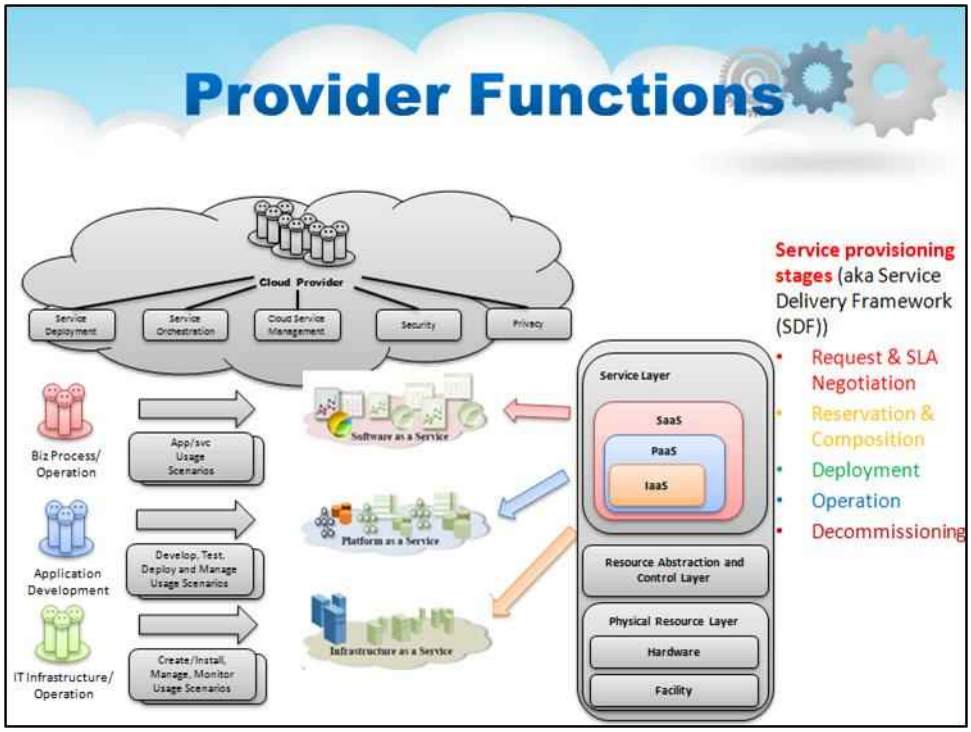
Хмарний інтегратор

Сторона, основна роль якої полягає у впровадженні схваленого хмарного проекту, зокрема міграції IT до хмар, а також може включати інші функції, такі як зрілість IT-інфраструктури компанії та готовність до оцінки хмари, розробка плану впровадження, хмарна інфраструктура та розгортання програм. **Оператор хмарного/міжхмарного сервісу**

Сторона, якій можуть бути передані створені хмарні програми та інфраструктура.

Постачальник хмарних ресурсів і постачальник фізичних ресурсів

Сторони, які діють як постачальники хмарних або фізичних ресурсів, які можуть бути інтегровані в майбутнє, що надається клієнтам хмарних послуг або інфраструктури.

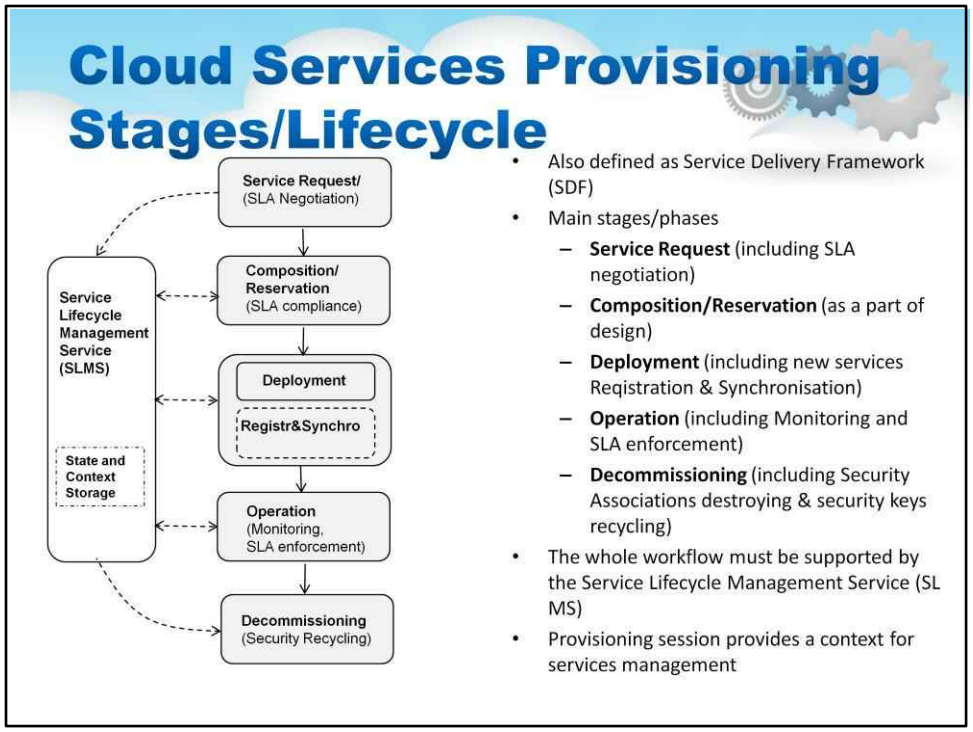


Постачальник хмарних послуг найбільш відомий своїми видимими функціями, які він надає, тобто вмикає можливості IaaS, PaaS і SaaS.

За лаштунками існує структура надання послуг (SDF), яка забезпечує механіку для надання послуг.

Надання послуги: відбувається в кілька етапів: запит і узгодження SLA, резервування та композиція, розгортання, операція, виведення з експлуатації.

Вони пояснюються на наступному слайді.



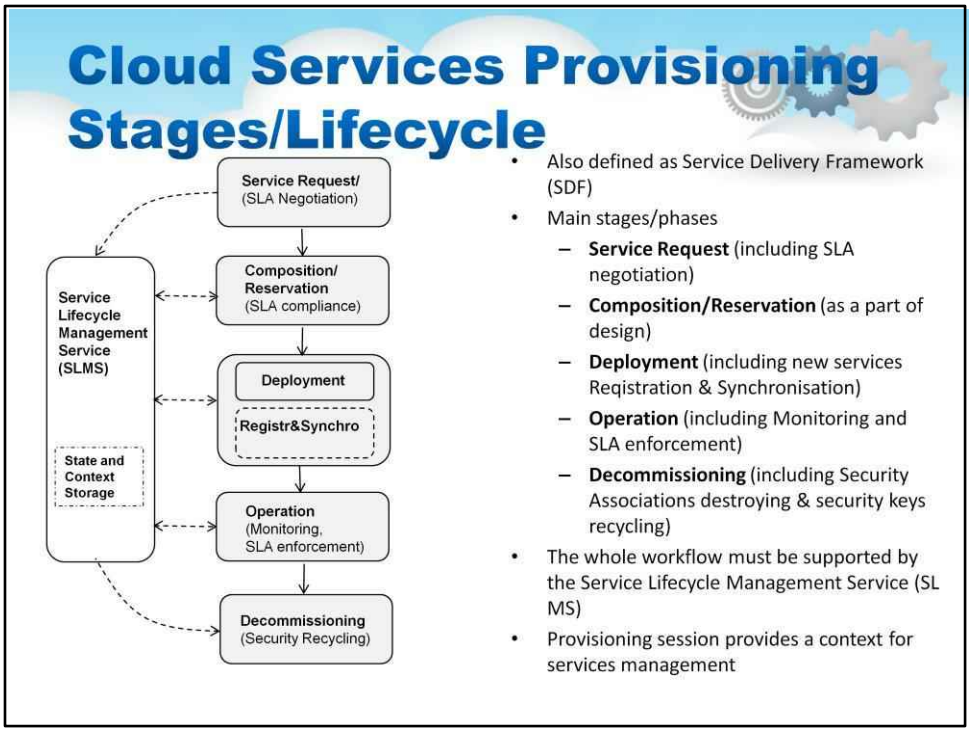
Слайд ілюструє основні етапи надання або доставки послуг, які відповідають конкретним вимогам до наданих віртуальних послуг на вимогу:

Етап запиту на обслуговування, включаючи переговори щодо угоди про рівень обслуговування (SLA). SLA може описувати якість обслуговування (QoS) і вимоги безпеки узгодженої інфраструктурної послуги, а також інформацію, яка полегшує автентифікацію запитів на обслуговування від користувачів. Цей етап також включає генерацію глобального ідентифікатора резервування (GRI), який слугуватиме ідентифікатором сеансу підготовки та зв’язуватиме всі інші етапи та відповідний контекст безпеки.

Етап композиції/резервування, який також включає зв’язування сеансу резервування з GRI, що забезпечує підтримку складних процесів резервування в багатодоменних середовищах із кількома постачальниками. На цьому етапі може знадобитися контроль доступу та застосування SLA/політики.

Стадія розгортання, включаючи реєстрацію та синхронізацію послуг. Етап розгортання починається після того, як усі ресурси компонентів були зарезервовані, і включає розповсюдження загального складеного контексту служби (включаючи контекст безпеки) і прив’язку зарезервованих ресурсів або послуг до GRI як загального ідентифікатора сеансу надання.

Етап реєстрації та синхронізації (які є необов’язковими) спеціально націлений на сценарії з міграцією або перепланування забезпеченої служби.



Етап експлуатації (включаючи моніторинг). Це основний робочий етап наданих хмарних сервісів за запитом. Моніторинг є важливою функціональністю на цьому етапі для забезпечення доступності служби та безпечної роботи, включаючи виконання SLA.

Етап виведення з експлуатації забезпечує припинення всіх сеансів, очищення даних і повторне використання контексту безпеки сеансу. Етап виведення з експлуатації також може надавати інформацію або ініціювати облік використання послуг.

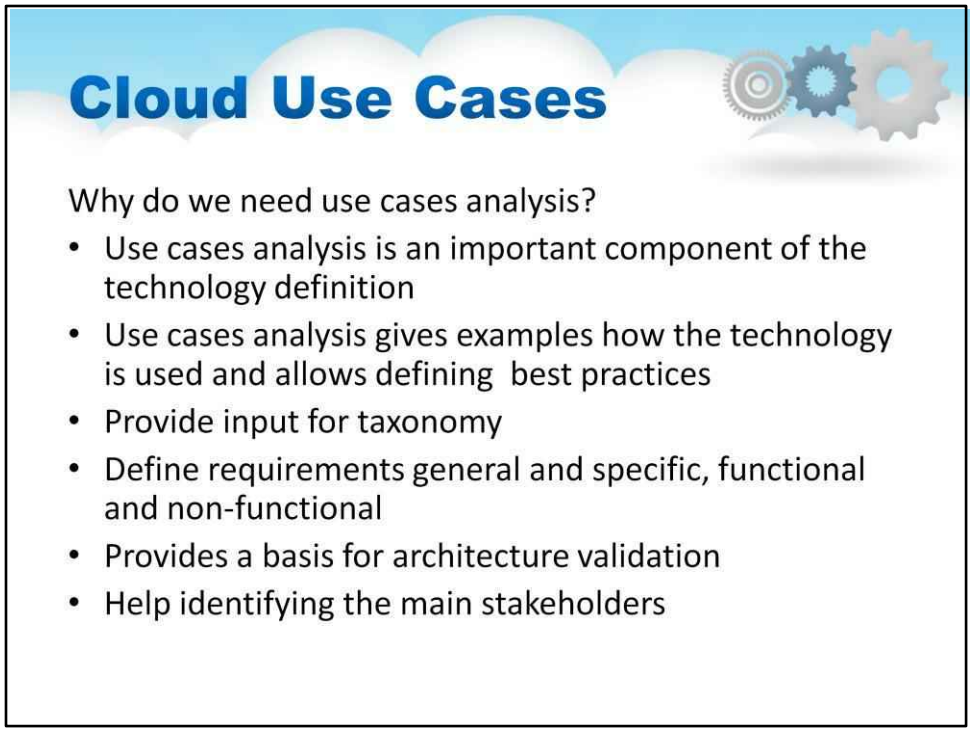
Два додаткових (під)етапи можуть бути ініційовані зі стадії «Операція» на основі запущеної служби або стану ресурсів:

Етап рекомпозиції або перепланування має дозволяти поступові зміни інфраструктури.

Етап відновлення/міграції може ініціювати користувач або постачальник. Цей процес може використовувати MD-SLC для ініціювання повної або часткової повторної синхронізації ресурсів, також може знадобитися повторна композиція.

Реалізація запропонованого SDF вимагає спеціального сховища метаданих життєвого циклу служби (MD SLC) для підтримки узгодженого управління життєвим циклом послуг. MD SLC зберігає метадані служб, які включають принаймні стан служби, властивості служби та інформацію про конфігурацію служб. Ця функція є частиною програмного забезпечення для керування хмарою та програмного забезпечення хмарної платформи.

Випадки використання



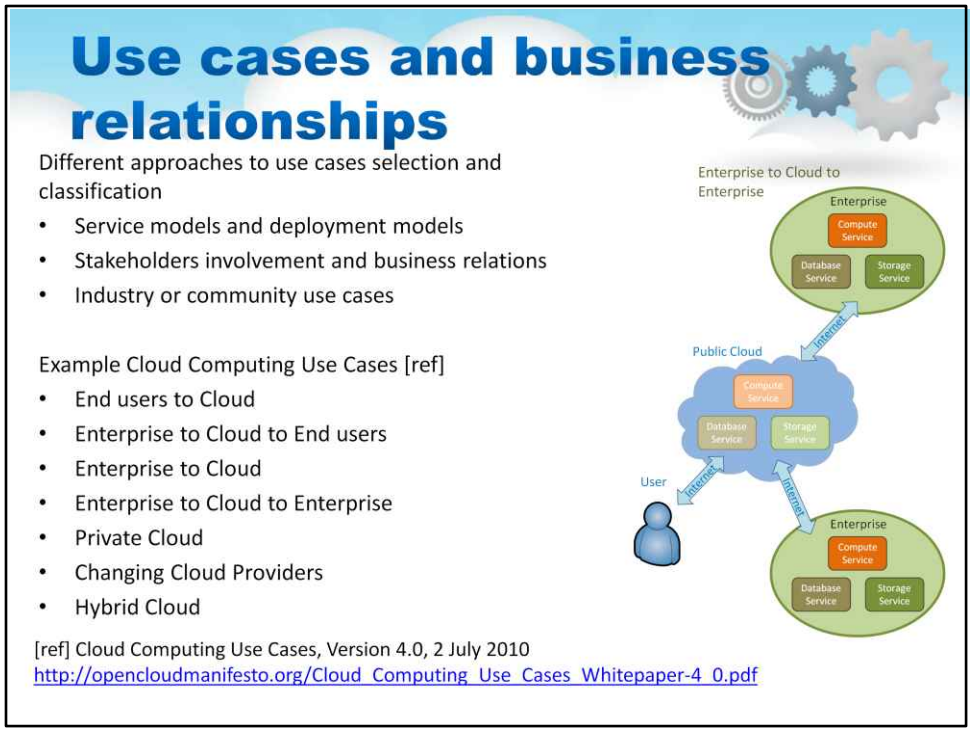
Навіщо нам аналіз варіантів використання?

- Аналіз варіантів використання є важливою складовою визначення технології
- Аналіз варіантів використання дає приклади використання технології та дозволяє визначити найкращі практики
- Введіть дані для таксономії
- Визначте вимоги загальні та специфічні, функціональні та нефункціональні
- Забезпечує основу для перевірки архітектури
- Допоможіть визначити головних зацікавлених сторін

Ми також не повинні виключати аналіз випадків використання як цінну інформацію для освіти та професійної підготовки. Що ми насправді робимо на цьому курсі.

З іншого боку, під час планування міграції IT-інфраструктури компанії в хмари відповідний хмарний сервіс і модель розгортання вибираються на основі низки факторів:

- Компанія, бізнес і програми повинні мати економічні або комерційні переваги
- рух до хмар
- Крім суто технічних, необхідно враховувати інші ділові, організаційні чи кадрові фактори
- Деякі (старіші) програми, можливо, потребуватимуть переробки
- Період переходу від внутрішніх до хмарних сервісів потребує часу та має бути ретельно спланований

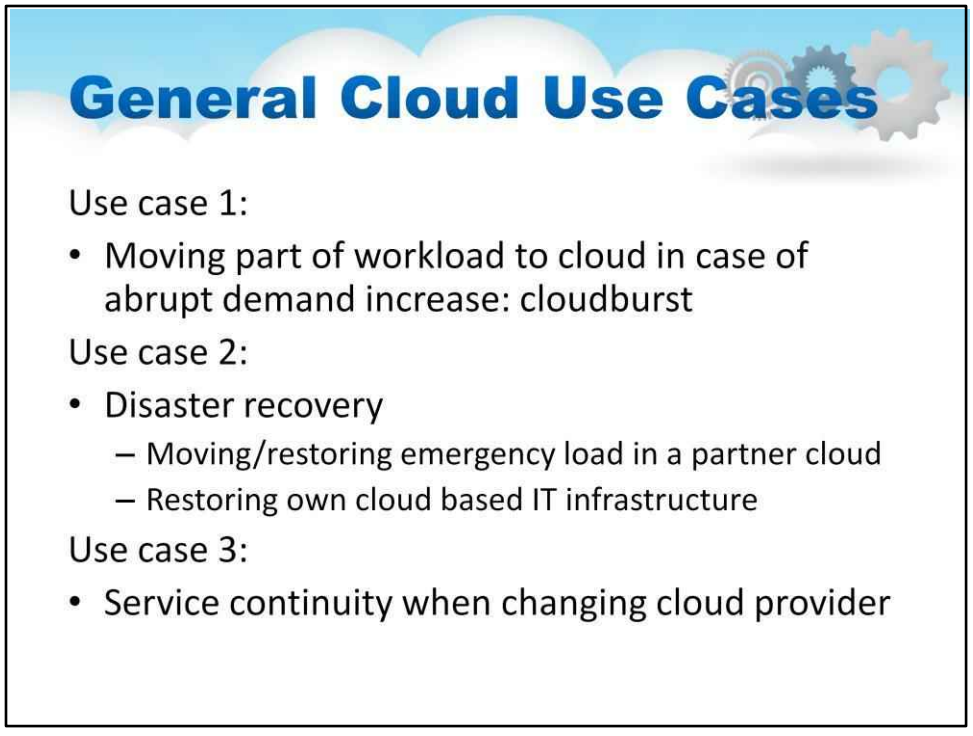


Існують різні підходи до вибору та класифікації випадків використання:

- подивитися на різноманітність моделей обслуговування та моделей розгортання;
- перерахувати можливості, засновані на залученні зацікавлених сторін і ділових відносинах; розгляну
- та прийняти загальні варіанти використання в галузі або спільноті.

Спільними зусиллями дослідників хмарних обчислень було запропоновано один із способів охарактеризувати приклади використання: Кінцеві користувачі з Хмари Підприємство в Хмару до Кінце користувачів З Підприємства в Хмару Від Підприємства до Хмари в Корпоративне Приватна Хмара Зміна Хмарних Провайдерів Гібридна Хмара Що важливо враховувати, це , що ці сценарії не раптово «трапляються», вони будуються, або, точніше, «вростають» через певну потребу підприємства. Подума про варіант використання, коли підприємство хоче перенести частину своєї IT-інфраструктури в хмару. не впевнено, який підхід йому потрібен. Але це той варіант використання, з яким стикаються всі компан та підприємства, коли вирішують перенести свою IT-інфраструктуру в хмари. Мотивація для цього поля в тому, щоб скористатися функціональними перевагами хмари, описаними вище, а також економічними та бізнес-цінностями.

Як ми зазначали, повна хмарна міграція не відбувається за один крок. Для великих організацій це зазвичай починається з впровадження приватної хмари та переміщення локальних IT-служб у хма Цей крок також призведе до повної зрілості IT та її готовності передати деякі послуги в публічну хмару. Це створює гібридну хмару. І наступним кроком буде переведення операційної IT-інфраструктури або деяких відділів повністю в хмару. Що таке виклики та як їх вирішувати, ми обговоримо в наступних випадках використання.



Ми можемо визначити наступні загальні випадки використання хмари, які ми детально обговорюємо нижче.

Випадок використання 1:

Переміщення частини робочого навантаження в хмару у разі різкого збільшення попиту: хмарна зрив

Випадок використання 2:

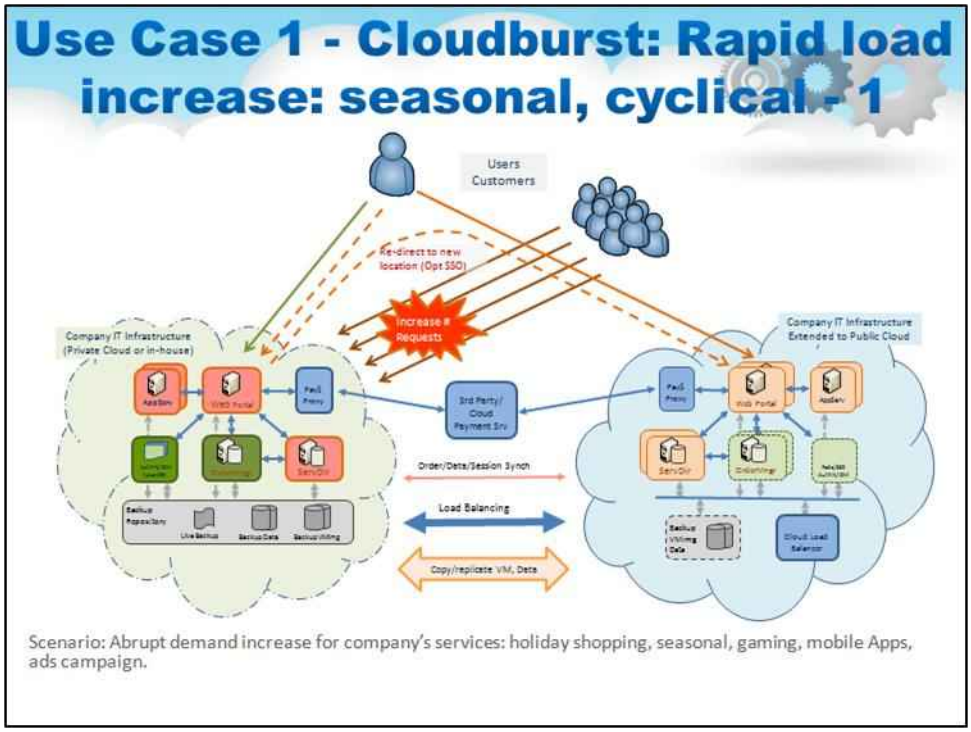
Аварійного відновлення

Переміщення/відновлення екстреного навантаження в партнерській

хмарі Відновлення власної хмарної IT-інфраструктури

Випадок використання 3:

Безперервність обслуговування при зміні хмарного провайдера



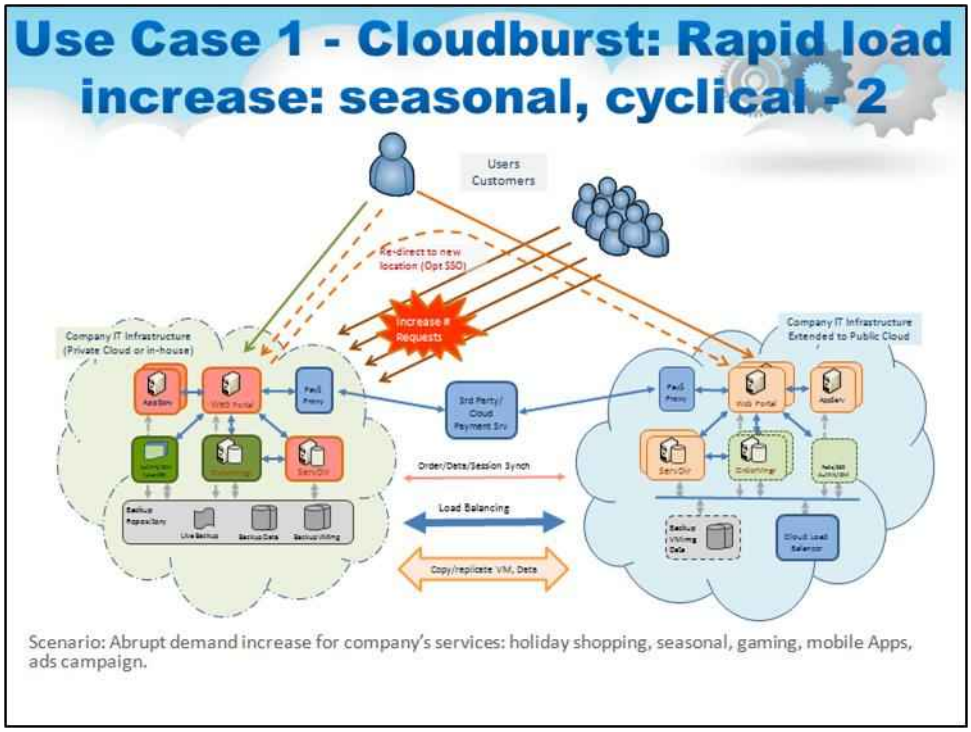
Це варіант використання 1: розширення послуг і можливостей у загальнодоступній хмарі у разі швидкого зростання попиту (сценарій «вибуху хмари»).

Термін «вибух хмари» — це неточний термін, який широко використовується компаніями для опису ситуацій, коли робоче навантаження тимчасово переноситься в хмару, розширюючи та відтворюючи ресурси приватної хмари та віртуальні машини (за формулою «купи базу, орендуй шип»).

Ми розглянемо один сценарій «вибуху хмари», оскільки він є одним із ключових випадків використання хмарних обчислень, що забезпечує важливі переваги для використання хмарних технологій малим і середнім бізнесом (також званим SMB — малим і середнім бізнесом).

Гіпотетичне МСП — це стартап, який уже працює, але розглядає можливість створення нового продукту чи послуги, що вимагатиме створення або аутсорсингу нової IT-інфраструктури та ресурсів. Основні передумови та вимоги:

Неперевірене/непередбачуване робоче навантаження - Зокрема, для веб-магазинів, соціальних сайтів, ігор і мобільних додатків. Бізнес хоче розгортати послуги та інфраструктуру «еластично», щоб їх можна було розширювати та не розширювати відповідно до фактичних потреб. Це дозволяє легко розширити (у разі успіху) і дешево відмовити (якщо обслуговування не вдасться).

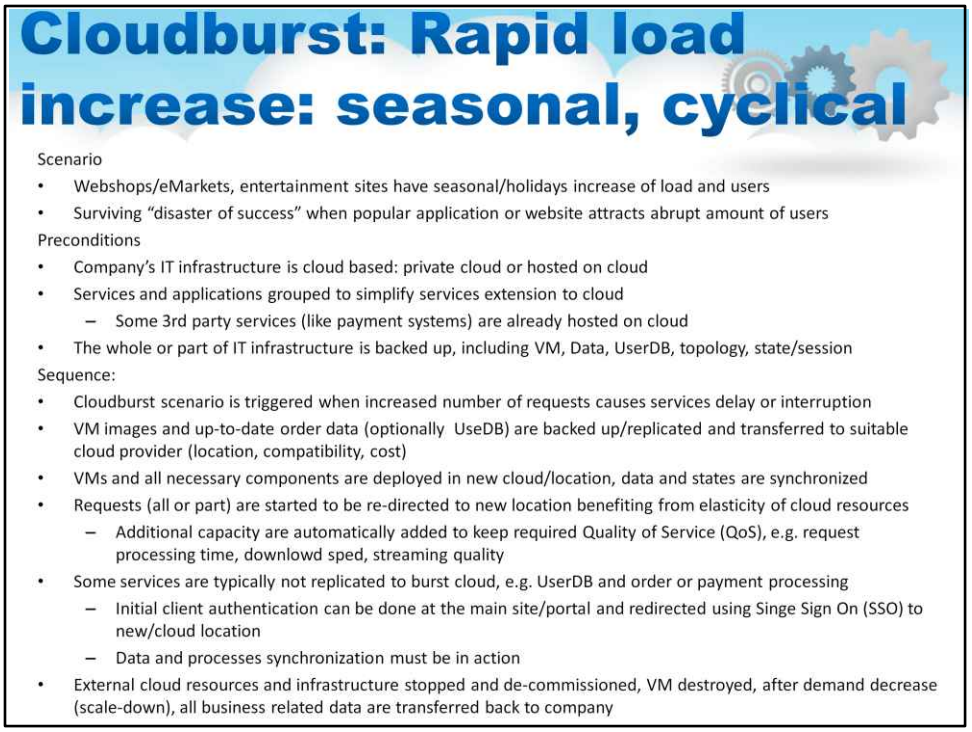


Іншою рушійною силою є очікуване розширення послуг у різних країнах і географічних зонах. Можливість створити інфраструктуру в інших місцях задовольняє цю потребу. Все одно хочеться мати кілька інфраструктур, щоб забезпечити балансування навантаження та мінімізацію затримок у різних регіонах.

Варіант використання SME також має розглядати потенційну ситуацію під назвою «катастрофа успіху високого рівня», що відбувається, коли популярність послуг або сайтів швидко зростає, що може бути у випадках із сучасними веб- і мобільними додатками. Відомі приклади/історії включають перевантаження служби святкових покупок BestBuy у 2012 році або збій служби Netflix у тому ж 2012 році через проблеми в AWS, яка розміщує сервіси Netflix. Сервіс Netflix також був нерегулярно доступний під час різдвяних свят у 2013 році в Європі. Усі ці випадки відмови в обслуговуванні були спричинені підвищеним попитом з боку клієнтів.

Рішення, засноване на хмарі, може ефективно вирішувати ситуації з припливом попиту, однак програми та послуги повинні бути розроблені таким чином, щоб дозволити їхнє легке розширення, тиражування та переміщення до зовнішньої інфраструктури хмарного постачальника. Важливо повторити, що не всі сервіси та операційні процедури підходять для переходу в хмари, зокрема ті, які мають справу з конфіденційними даними або вимагають критичної доступності.

Діаграма на слайді ілюструє цей варіант використання. Компанія може бути одного типу або запускати такі програми: веб-магазин або електронний ринок, розважальна або ігрова програма. Такі програми відомі тим, що мають сезонний або циклічний попит, і в разі успіху можуть залучити різко збільшену кількість користувачів.



Щоб завершити, давайте підкреслимо ключові моменти цього сценарію використання:

Сценарій

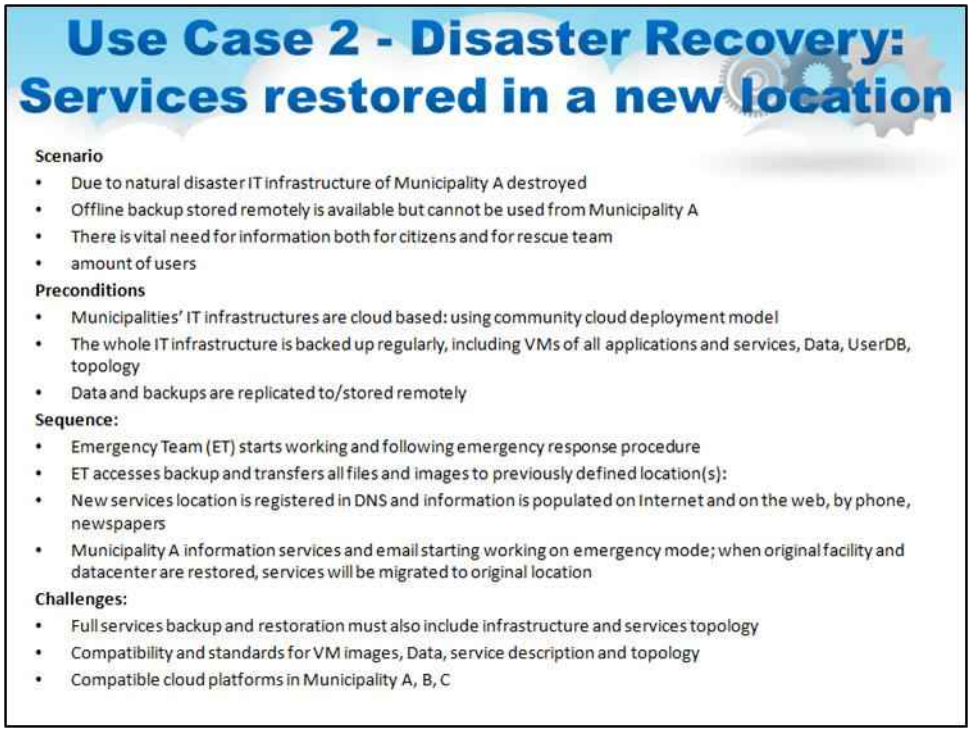
- Веб-магазини/електронні ринки, розважальні сайти мають сезонне/святкове збільшення навантаження та користувачів
- Пережити «катастрофу успіху», коли популярний веб-сайт приваблює різку кількість користувачів

Передумови

- IT-інфраструктура компанії заснована на хмарі: приватна хмара або розміщена в хмарі
- Служби та програми, згруповані для спрощення розширення послуг у хмарі
- Деякі сторонні сервіси (наприклад, платіжні системи) уже розміщені в хмарі
- Резервне копіювання всієї IT-інфраструктури або її частини, включаючи віртуальну машину, дані, базу даних користувача, топологію, стан/сесію

Послідовність:

- Сценарій Cloudburst запускається, коли збільшення кількості запитів спричиняє затримку або перериванн послуг
- Зображення віртуальної машини та оновлені дані про замовлення (опціонально UseDB) створюються резервні копії/тиражуються та передаються до відповідного хмарного постачальника (розташування, сумісність, вартість)
- Віртуальні машини та всі необхідні компоненти розгортаються в новій хмарі/розташуванні, дані та стани синхронізуються
- Запити (всі або частина) починають перенаправлятися в нове місце, що дає перевагу еластичності хмарних ресурсів
- Деякі служби, як правило, не копіюються в пакетну хмару, наприклад UserDB і обробка замовлень або платежів
- Зовнішні хмарні ресурси та інфраструктура зупинені, віртуальна машина знищена, після зменшення попиту (зменшення масштабу) усі пов'язані з бізнесом дані передаються назад до компанії.



Варіант використання 2: аварійне відновлення та масштабний збій постачальника

Сценарій

Через стихійне лихо IT-інфраструктура муніципалітету А зруйнована

Резервне копіювання в режимі офлайн, яке зберігається віддалено, доступне, але не може використовуватися з муніципалітету А Існує життєва потреба в інформації як для громадян, так і для кількості користувачів рятувальних команд

Передумови

IT-інфраструктури муніципалітетів базуються на хмарі: з використанням моделі розгортання хмари спільноти В

IT-інфраструктура регулярно створюється резервна копія, включаючи віртуальні машини всіх програм і служб, даних, UserDB, топології

Дані та резервні копії копіюються/зберігаються віддалено

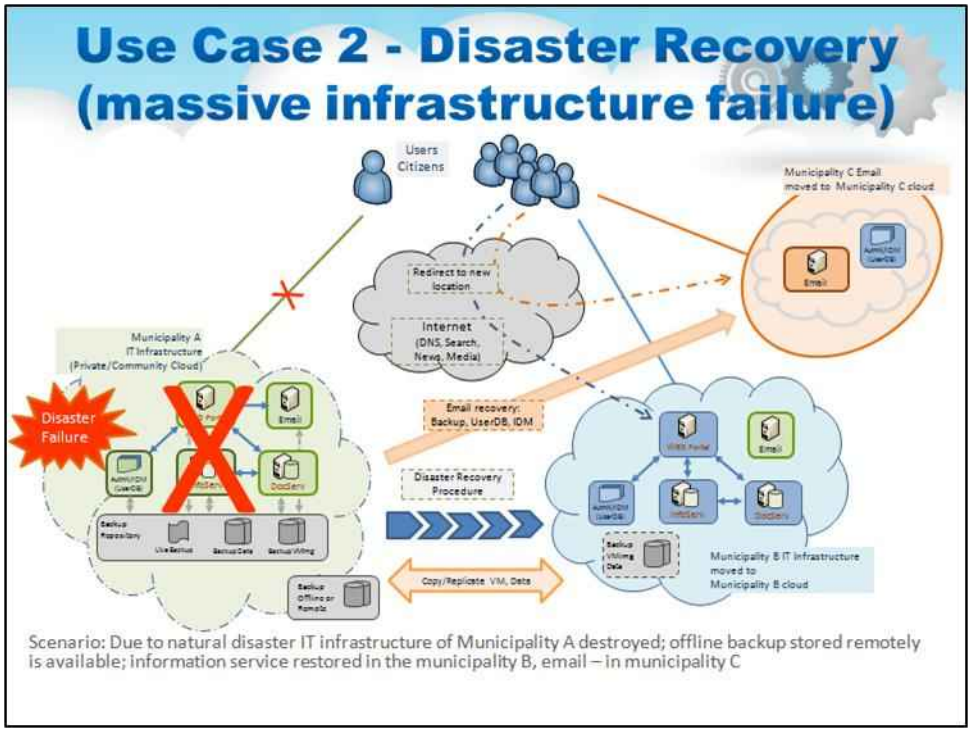
Послідовність:

Команда екстреної допомоги (ET) починає роботу та, дотримуючись процедури реагування на надзвичайні ситуації, ET отримує доступ до резервної копії та передає всі файли та зображення до попередньо визначеного місця(-й): нове розташування служби реєструється в DNS, а інформація заповнюється в Інтернеті та Інтернеті, по телефону, газети

Інформаційні служби муніципалітету А та електронна пошта працюють в екстреному режимі; після відновлення початкового об'єкта та центру обробки даних служби буде переміщено до початкового розташування **Виклики**:

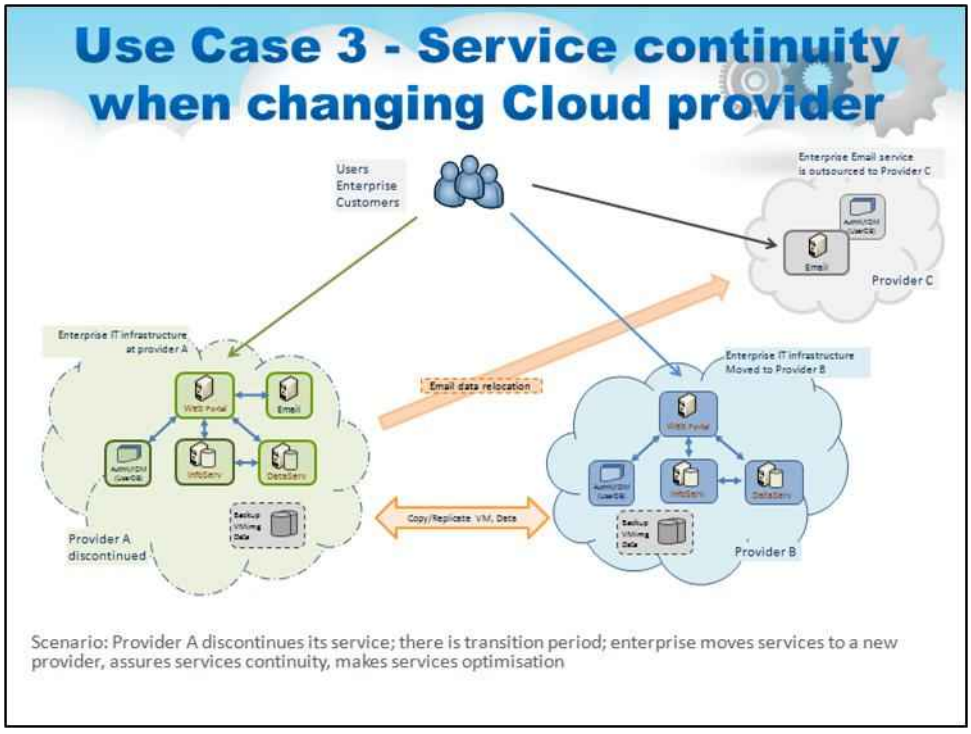
Повне резервне копіювання та відновлення служб також має включати інфраструктуру та топологію служб Сумісність і стандарти для зображень віртуальних машин, даних, опису послуги та топології

Сумісні хмарні платформи в муніципалітеті А, В, С



Для успішної роботи цього сценарію пропонуються наступні передумови: IT-інфраструктури муніципалітет базуються на хмарі, наприклад, використовують модель розгортання хмари спільноти. Вся IT-інфраструктура регулярно створює резервні копії, включаючи віртуальні машини всіх програм і служб, даних, UserDB і топології інфраструктури. Дані та резервні копії копіюються на/або зберігаються віддалено

Успіх описаного тут сценарію аварійного відновлення залежить від вирішення наступних проблем: Сумісність хмарних платформ у муніципалітеті A, B, C Сумісність і загальні стандарти для зображень, даних і опису послуг віртуальної машини. Повне резервне копіювання та відновлення служб також має включати інфраструктуру та топологію служб.



Варіант використання 3: безперервність обслуговування при зміні хмарного постачальника.

Цей приклад використання ілюструє основні завдання та проблеми під час переходу від одного постачальника хмарних послуг до іншого. Така ситуація може виникнути, коли поточний постачальник припиняє свою послугу або клієнт вирішив перейти до іншого постачальника через низку причин, наприклад вартість послуг, доступні послуги, нормативні вимоги, які можуть обмежувати розташування центру обробки даних постачальника.

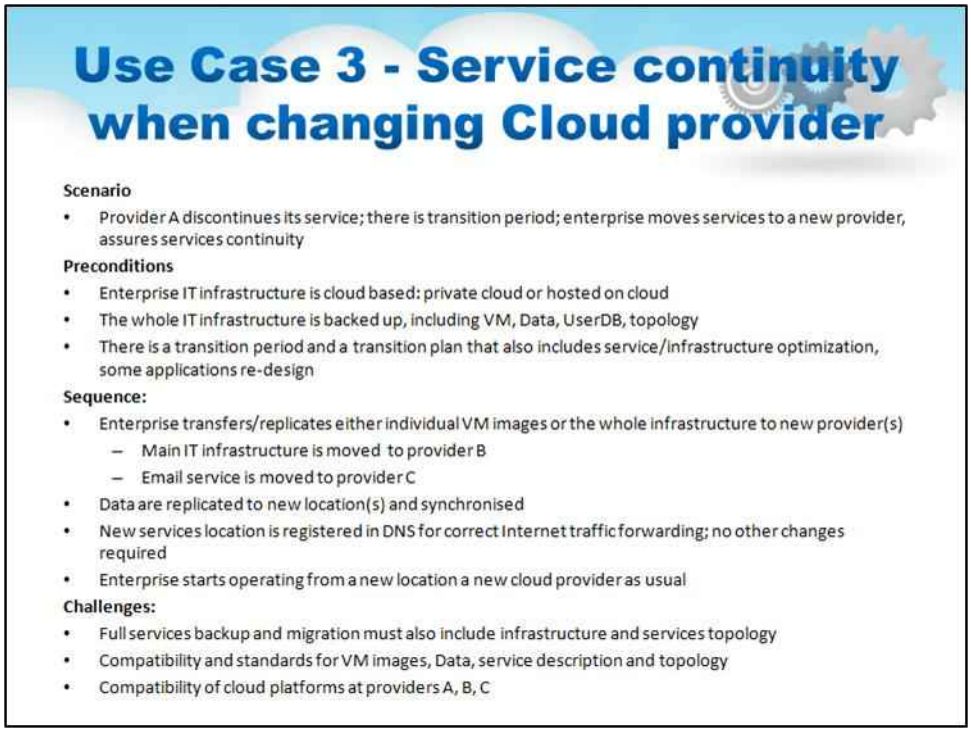
Власне, сценарій з переходом сервісу до іншого провайдера варто обговорювати при плануванні впровадження хмарних технологій на підприємстві, щоб уникнути можливих проблем з прив’язкою провайдера, що все ще характерно для хмарного бізнесу.

Слайд ілюструє сценарій міграції IT-інфраструктури. Наступні кроки описують процес міграції:

Підприємство передає/реплікує або окремі образи віртуальної машини, або всю інфраструктуру новому постачальнику(ам), у нашому випадку.

Основну IT-інфраструктуру переміщено до постачальника B. Службу електронної пошти переміщено до постачальника

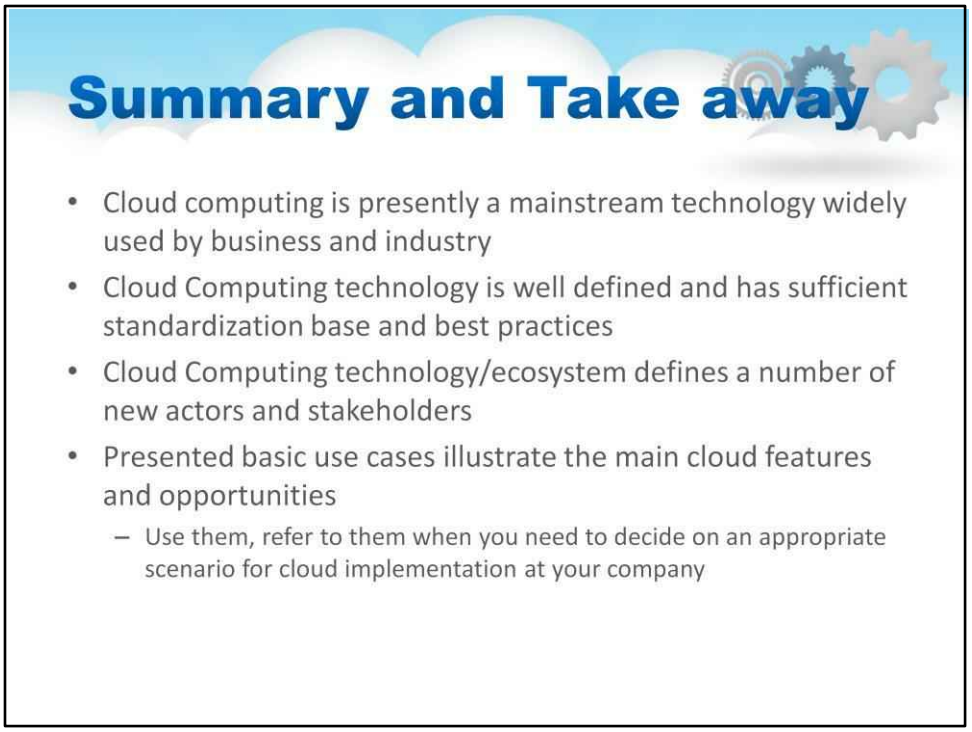
Дані копіюються в нове розташування та синхронізуються. Для коректної переадресації інтернет-трафіку в DNS реєструється нове розташування сервісів; жодних інших змін не потрібно Корпоративні служби починають працювати нових хмарних провайдерів, як зазвичай.



Щоб завершити, давайте підкреслимо ключові моменти цього випадку використання 3:

Процес міграції має бути добре спланований і буде перехідний період. Повинні бути забезпечені наступні передумови: корпоративна IT-інфраструктура базується на хмарі: приватна хмара або розміщена в хмарі. Резервна копія всієї IT-інфраструктури, включаючи віртуальну машину, дані, базу даних користувача, інфраструктуру або топологію служб. План переходу також може включати оптимізацію служб/інфраструктури, оновлений дизайн деяких програм.

Цей варіант використання має ті ж проблеми, що й у наших випадках: сумісність хмарних платформ у постачальників A, B, C, сумісність і стандарти для зображень віртуальних машин, дані, опис сервісу та топологія, повне та актуальне резервне копіювання сервісів, синхронізація даних на момент перехід служби на нове місце.



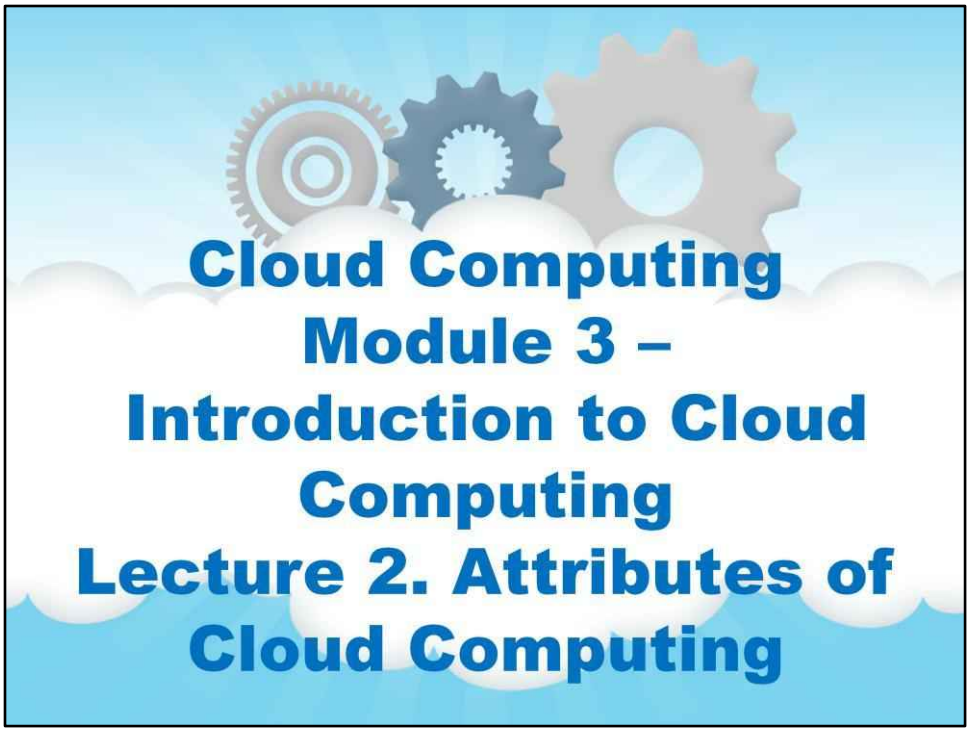
Хмарні обчислення зараз є основною технологією, яка широко використовується бізнесом і промисловістю

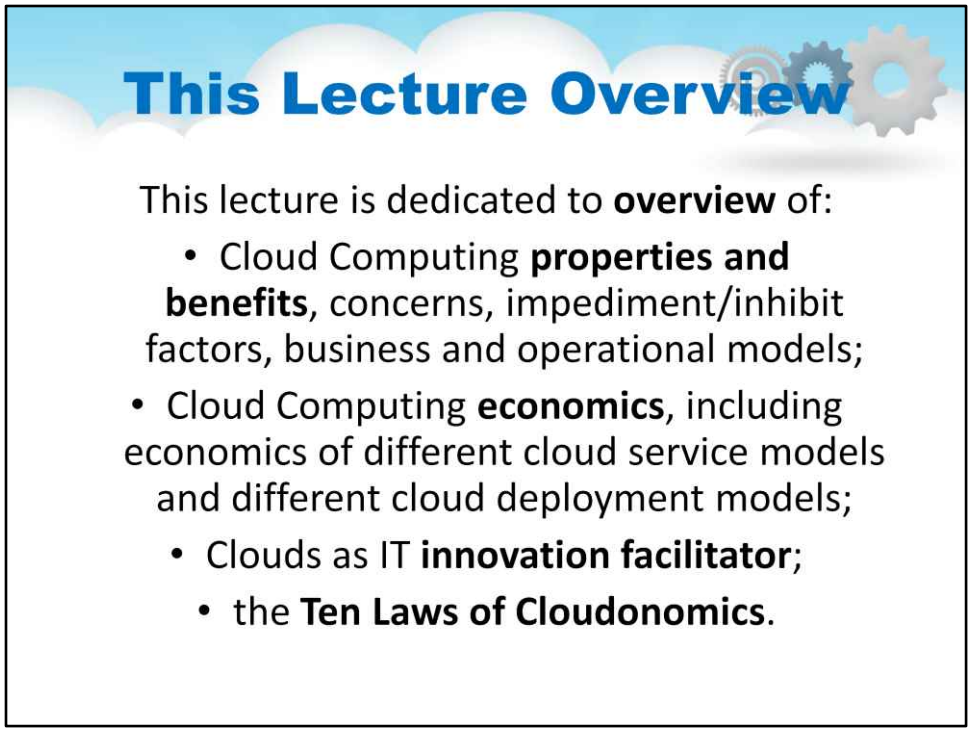
Технологія хмарних обчислень чітко визначена та має достатню базу стандартизації та передовий досвід

Технологія/екосистема хмарних обчислень визначає ряд нових акторів і зацікавлених сторін

Представлені базові випадки використання ілюструють основні функції та можливості хмари

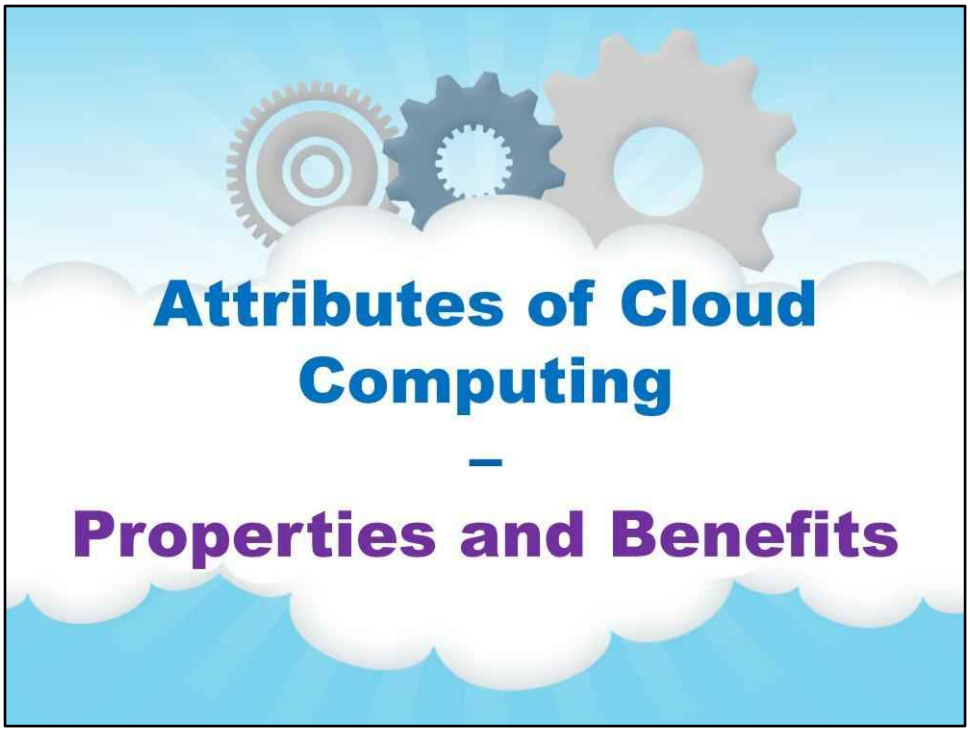
Використовуйте їх, зверніться до них, коли вам потрібно визначитися з відповідним сценарієм впровадження хмари у вашій компанії


Cloud Computing
Module 3 –
Introduction to Cloud Computing
Lecture 2. Attributes of Cloud Computing

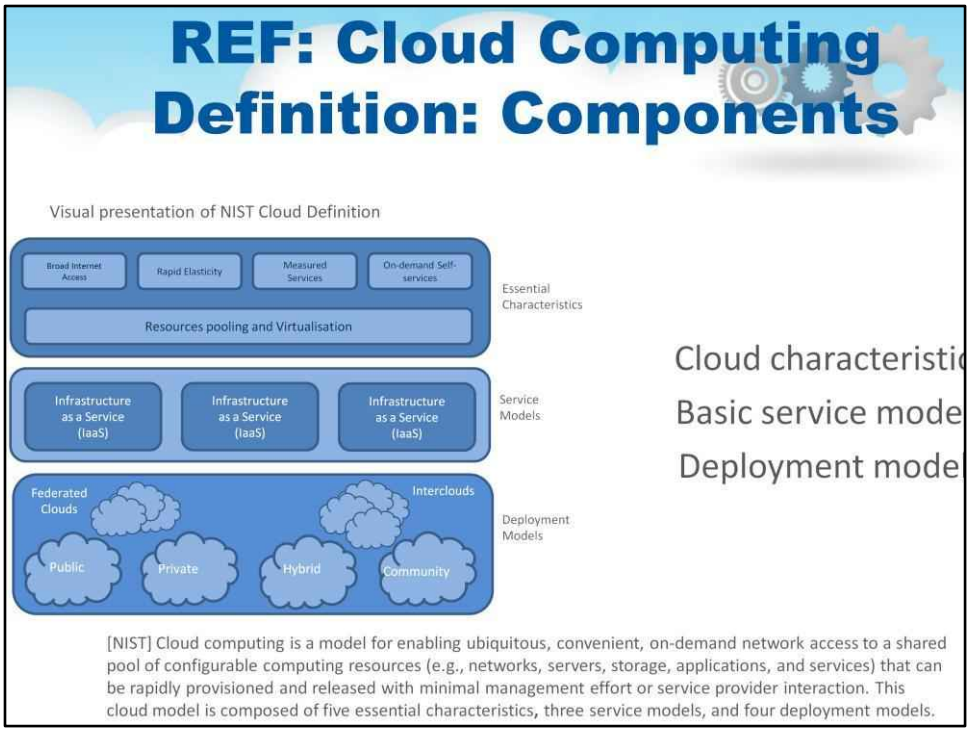


Лекція 2. Атрибути Хмари

обчислювальна техніка



Властивості та переваги



Хмарні характеристики

Самообслуговування на вимогу

Широкий доступ до мережі

Об'єднання ресурсів

Швидка еластичність

Виміряне обслуговування

Базові моделі обслуговування

Програмне забезпечення як послуга

(SaaS) Платформа як послуга (PaaS)

Моделі розгортання інфраструктури як

послуги (IaaS)

Приватні хмари

Громадські хмари

Гібридні хмари

Хмари спільноти

Федеративні хмари, міжхмарні хмари

Scalability and Elasticity

What do scalability and elasticity mean in IaaS?

- Clients should be able to dynamically increase or decrease the amount of infrastructure resources in need.
- Large amount of resources provisioning and deployment should be done in a short period of time, such as several hours or days.
- System behavior should remain identical in small scale or large one.

How to approach scalability and elasticity in IaaS?

- For computation resources:
  - Dynamically create or terminate virtual machines for clients on demand.
  - Integrate hypervisors among all physical machines to collaboratively control and manage all virtual machines.
- For storage resources:
  - Dynamically allocate or de-allocate virtual storage space for clients.
  - Integrate all physical storage resources in the entire IaaS system
  - Offer initial storage resources by thin provisioning technique.
- For communication resources :
  - Dynamically connect or disconnect the linking state of virtual networks for clients on demand.
  - Dynamically divide the network request flow to different physical routers to maintain access bandwidth.

Availability and Reliability

What do availability and reliability mean in IaaS?

- Clients should be able to access computation resources without considering the possibility of hardware failure.
- Data stored in IaaS cloud should be able to be retrieved when needed without considering any natural disaster damage.
- Communication capability and capacity should be maintained without considering any physical equipment shortage.

How to approach availability and reliability in IaaS?

- For computation resources :
  - Monitor each physical and virtual machine for any possible failure.
  - Regularly backup virtual machine system state for disaster recovery.
  - Migrate virtual machine among physical machines for potential failure prevention.
- For storage resources :
  - Maintain data pieces replication among different physical storage devices.
  - Regularly backup virtual storage data to geographical remote locations for disaster prevention.
- For communication resources :
  - Built redundant connection system to improve robustness.

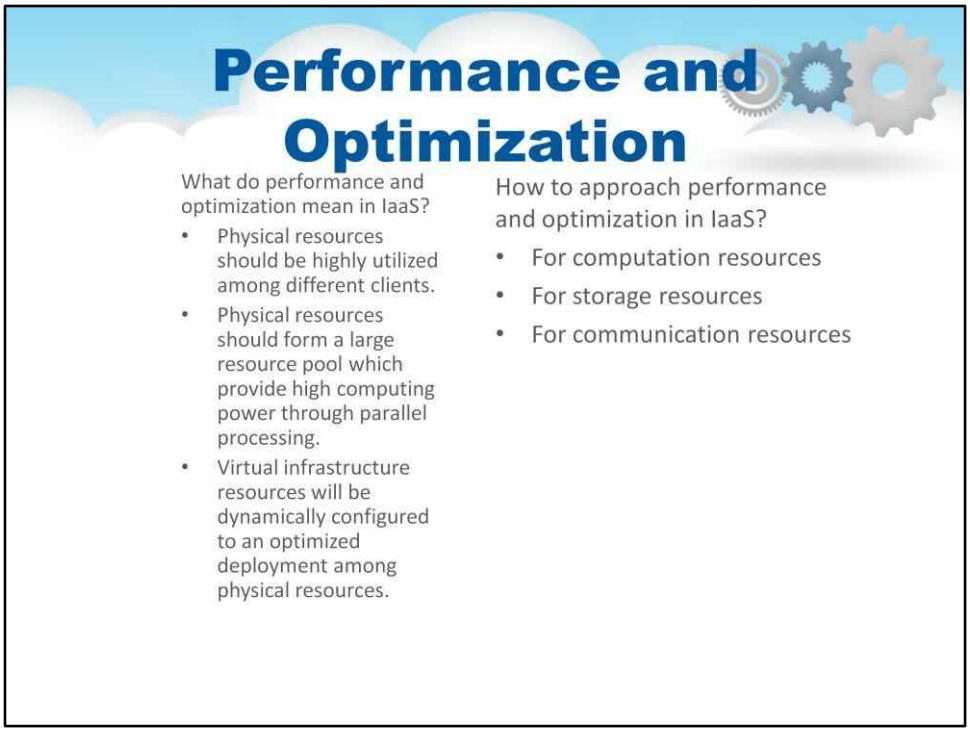
Manageability and Interoperability

What do manageability and interoperability mean in IaaS?

- Clients should be able to fully control the virtualized infrastructure resources which allocated to them.
- Virtualized resources can be allocated by means of system control automation process with pre-configured policy.
- States of all virtualized resource should be fully under monitoring.
- Usage of infrastructure resources will be recorded and then billing system will convert these

How to approach manageability and interoperability in IaaS?

- For computation resources :
  - Provide basic virtual machine operations, such as creation, termination, suspension, resumption and system snapshot.
  - Monitor and record CPU and memory loading for each virtual machine.
- For storage resources :
  - Monitor and record storage space usage and read/write data access from user for each virtual storage resource.
  - Automatic allocate/de-allocate physical storage according to space utilization.
- For communication resources :
  - Monitor and record the network bandwidth consumption for each virtual link.
  - Automatically reroute the data path when computation and storage are duplicated.



Для обчислювальних ресурсів:

Розгорніть віртуальну машину з урахуванням балансування навантаження.

Жива міграція віртуальних машин між фізичними, щоб збалансувати завантаження системи.

Для ресурсів зберігання:

Розгорніть віртуальне сховище з урахуванням доступу до гарячої точки.

Жива міграція віртуальних сховищ між фізичними з різним рівнем продуктивності.

Для комунікаційних ресурсів:

Враховуйте навантаження на пропускну здатність мережі під час розгортання віртуальних машин і сховищ.

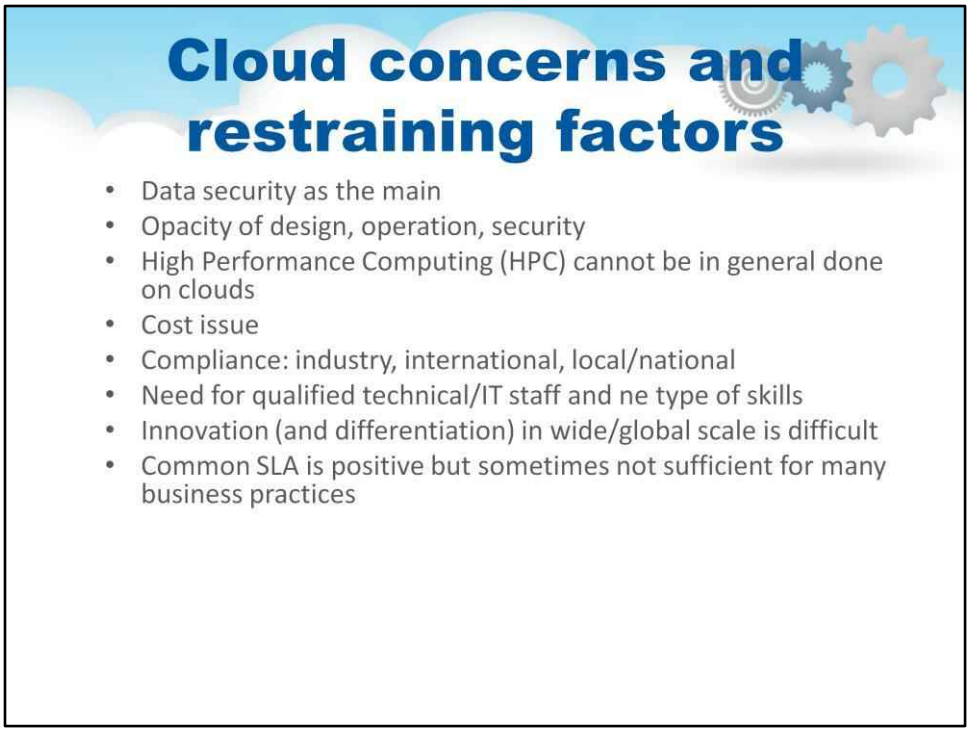
Accessibility and Portability

What do accessibility and portability mean in IaaS?

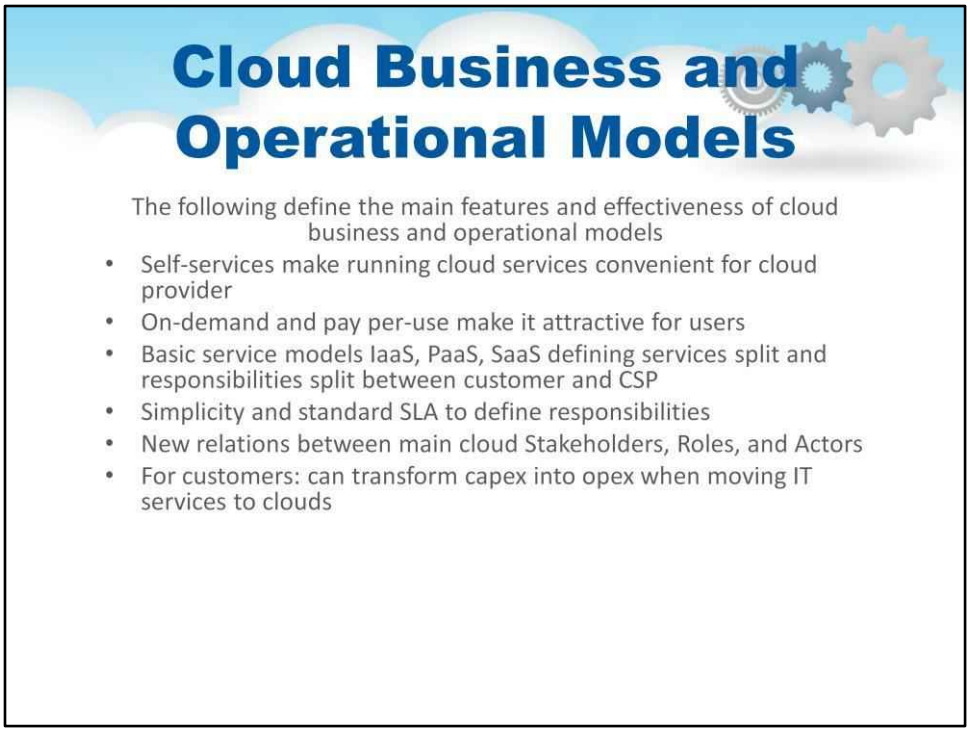
- Clients should be able to control, manage and access infrastructure resources in an easy way, such as the web-browser, without additional local software or hardware installation.
- Provided infrastructure resources should be able to be reallocated or duplicated easily.

How to approach accessibility and portability in IaaS?

- For computation resources :
  - Cloud provider integrates virtual machine management and access through web-based portal.
  - Comply the virtual machine standard for portability.
- For storage resources :
  - Cloud provider integrates virtual storage management and access through web-based portal.
- For communication resources :
  - Cloud provider integrates virtual network management and access through web-based portal.

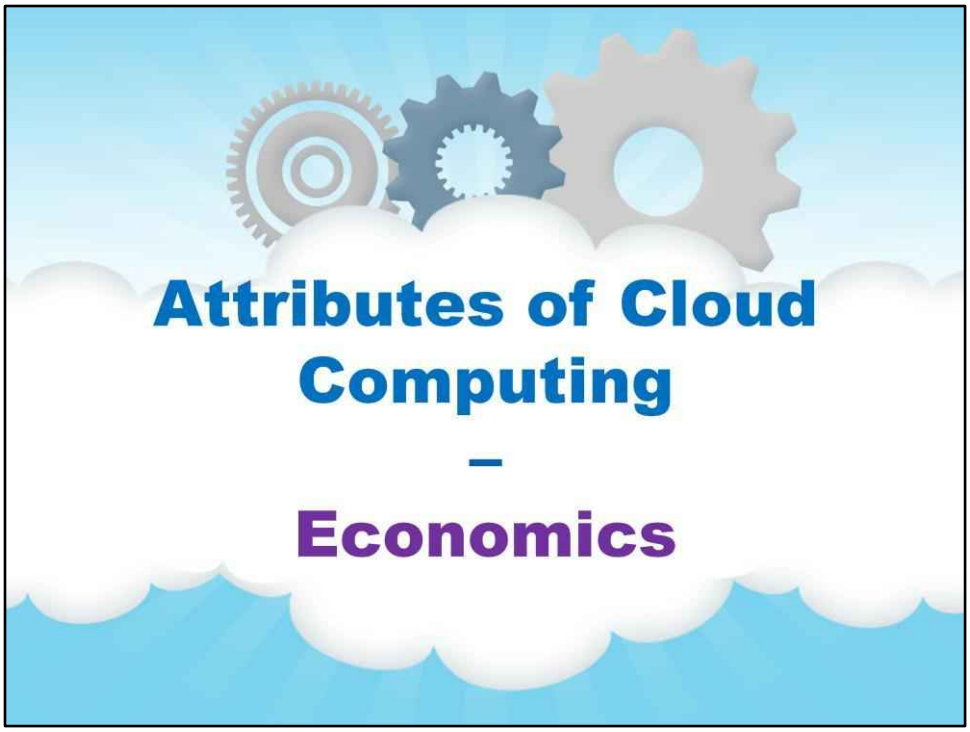


- Безпека даних як головний
  - о Де мої дані?
  - о Що відбувається з моїми даними? Хто має доступ до моїх даних? о Хмара eDiscovery і знищення даних
- Непрозорість дизайну, функціонування, безпеки о
  - Часто називають «хмарною завісою»
- Високопродуктивні обчислення (HPC) загалом не можна виконувати в хмарах о
  - Коефіцієнт підвищення через появу великих даних
- Проблема вартості
  - о Загальна вартість володіння (TOC) не завжди менша порівняно з традиційними IT
- Інновації (і диференціація) у широкому/глобальному масштабі є складними
  - о Темпи впровадження інновацій у країнах, що розвиваються, значно нижчі
- Загальний SLA є позитивним, але іноді недостатнім для багатьох бізнес-практик
  - о Деякі проблеми з хмарою можна вирішити, налаштований SLA, але може бути залучений підписавши NDA
    - о Зверніться до постачальника для модифікації та індивідуальних послуг



Для клієнтів: можна перетворити капітальні витрати на операційні витрати при перенесенні IT-послуг у хмари:

Крім того, зменште витрати на управління IT, резервне копіювання та IT-персонал

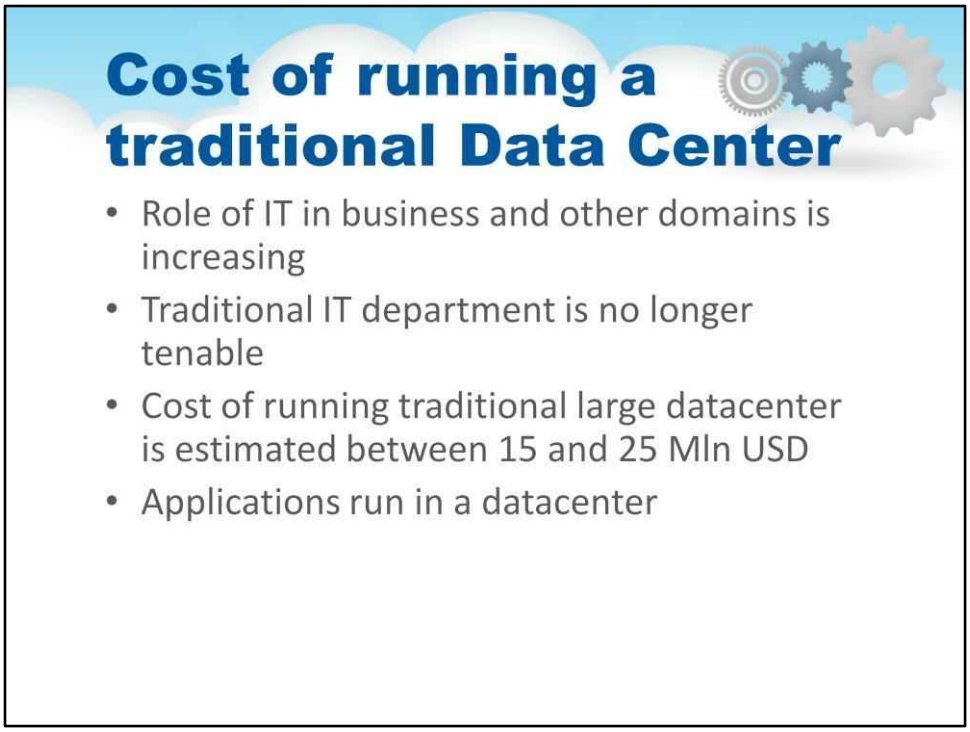


Економіка

Economics of Cloud Computing

Closer look at economical aspects of Cloud Computing

- Traditional Data Center
- Cloud Data Centers: economy of scale: Supply, demand, multi-tenancy
- Provider view, customer view, developer/integrator view



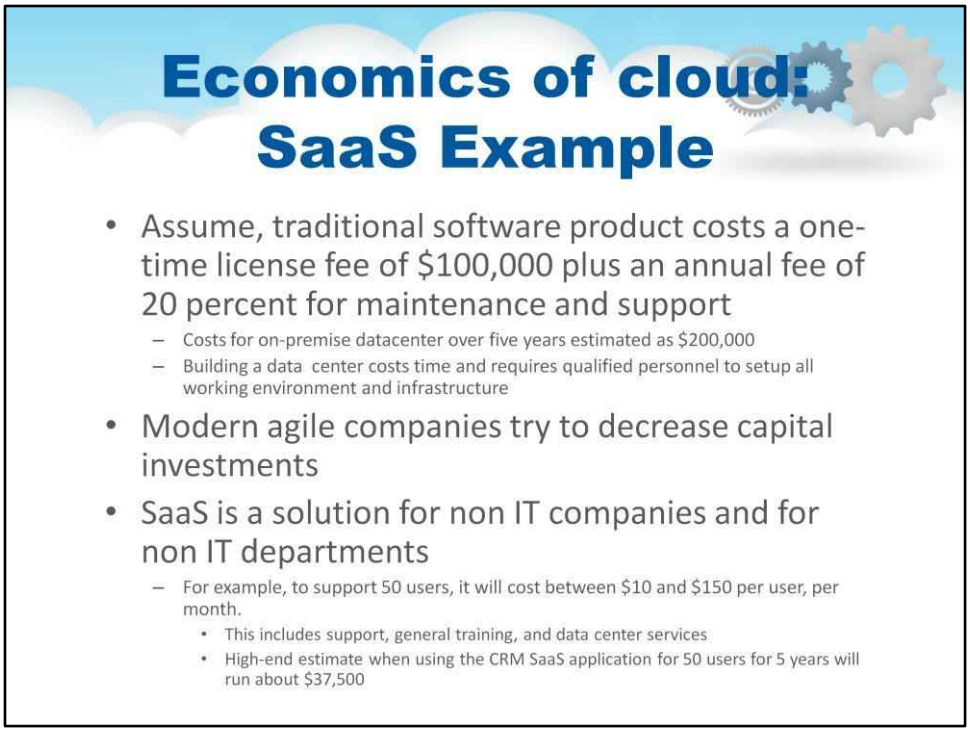
- Роль IT у бізнесі та інших сферах зростає
  - Це полегшує зміни в роботі/бізнесі та підвищує гнучкість бізнесу

- Традиційний IT-відділ більше не діє
  - Важко передбачити довгострокові (як раніше) потреби в IT із поточним ажіотажним зростанням цифрових програм як основного методу ведення бізнесу та взаємодії з клієнтами

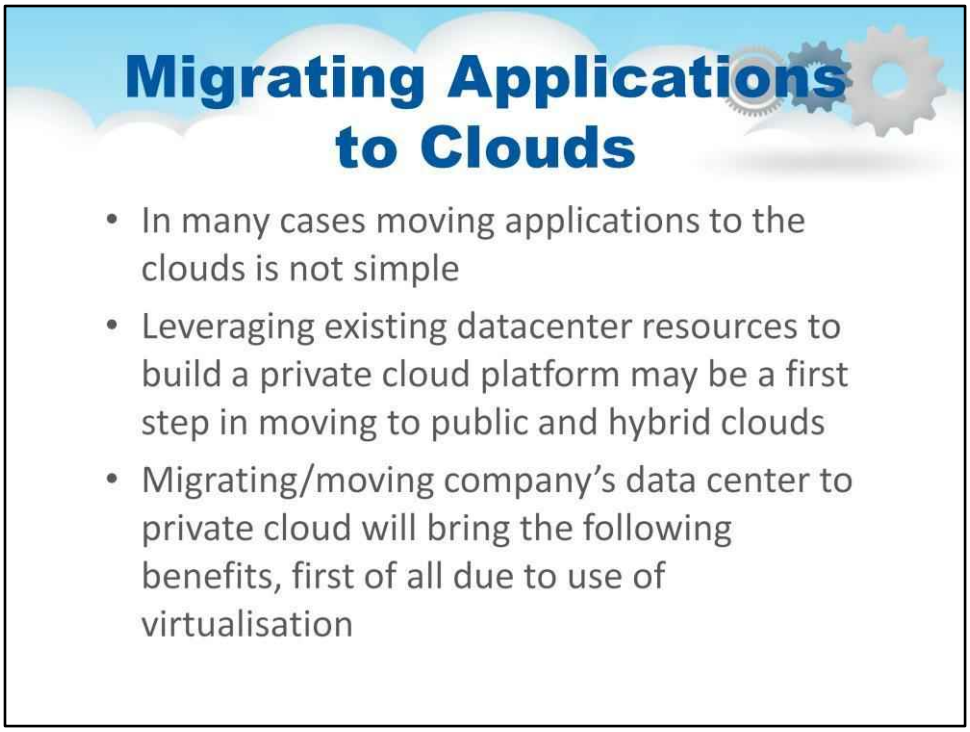
- Вартість експлуатації традиційного великого центру обробки даних оцінюється від 15 до 25 мільйонів доларів США
  - 42 відсотки: апаратне забезпечення, програмне забезпечення, механізми аварійного відновлення, джерела безперебійного живлення та мережі.
    - Витрати розподіляються в часі, амортизуються, оскільки вони являють собою комбінацію капітальних витрат і регулярних платежів.
  - 58 відсотків: опалення, кондиціонування повітря, податки на майно та продаж, а також витрати на оплату праці. (Насправді цілих 40 відсотків річних витрат становлять лише оплату праці)
- Програми працюють у центрі обробки даних
  - Більшість центрів обробки даних запускають багато різних програм і мають широкий спектр робочих навантажень.
  - Багато найважливіших програм, що працюють у центрах обробки даних, фактично використовуються лише відносно невеликою кількістю співробітників.
  - Деякі програми, які працюють на старих системах, уже видалено ринку (більше не продаються), але все ще необхідні для бізнесу.

Cost of running a Cloud Data Center

- Cloud data centers are different
  - Constructed for a different purpose
  - Perform different workloads than traditional data centers
  - Built to a different scale
  - Typically not constrained by the same limitations: space, hardware, staff (per server)
  - Leverage modern technologies and fully based on resources virtualisation
- The economics of a cloud data center is different and includes three cost factors
  - **Labor** costs estimated at 6 percent of the total costs of operating the cloud data center
  - **Power and cooling** estimated at 20 percent
  - **Computing** costs estimated at 48 percent



Для самостійної роботи знайдіть деталі в розділі «Основи послуг у хмарних обчисленнях» http://www.dummies.com/how-to/content/understanding-the-economics-of-saas-incloud-compu .html

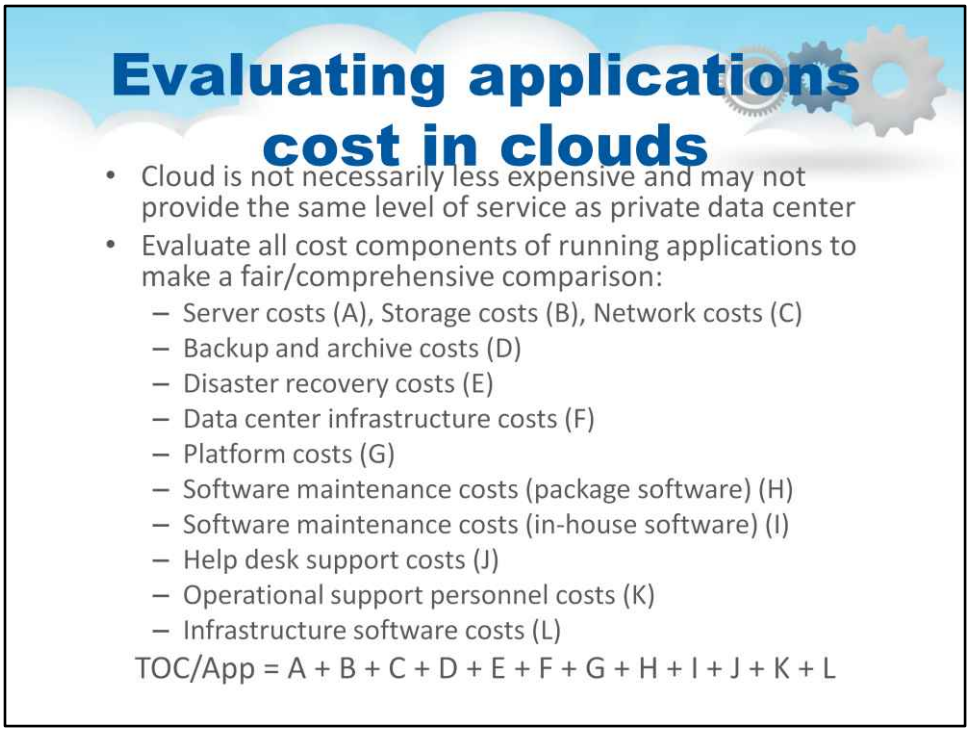


- У багатьох випадках переміщення програм у хмару непросте
  - Потрібно виконати певну роботу зі зміни конфігурації
  - Може знадобитися переробка програм і період тестування
  - Оцініть витрати, якщо програма недоступна для хмари
  - Для приватних і публічних хмар застосовуються однакові фактори вартості
  - Відповідність вимогам може бути ще одним фактором витрат, зокрема щодо безпеки та процедури віднов

Першим кроком може стати використання наявних ресурсів центру обробки даних для створення приватної хмарної платформи перехід до загальнодоступних і гібридних хмар Перенесення/перенесення центру обробки даних компанії до приватної хмари приносять такі переваги, перш за все завдяки використанню віртуалізації.

-

-



• Хмара не обов’язково є дешевшою та може не забезпечувати такий самий рівень обслуговування, як приватний центр обробки даних

• Власний центр обробки даних може мати угоду про рівень обслуговування з 99,999-відсотковим часом безвідмовної роботи.

• Чи запропонує хмарний провайдер той самий рівень обслуговування? Напевно ні.

• Компанія повинна зважити, наскільки критичним є рівень передбачуваного часу безвідмовної роботи для внутрішніх користувачів і клієнтів

Передбачте, що хмара не обов’язково буде дешевшою та не обов’язково забезпечуватиме той самий рівень обслуговування, що й ваш центр обробки даних.

Ваш власний центр обробки даних може мати угоду про рівень обслуговування з 99,999-відсотковим часом безвідмовної роботи. Чи запропонує ваш хмарний постачальник такий же рівень обслуговування? Напевно ні. Ви повинні зважити, наскільки цей рівень передбачуваної безвідмовної роботи є критичним для ваших внутрішніх клієнтів. Оцініть усі компоненти вартості запущених програм, щоб зробити справедливе/вичерпне порівняння:

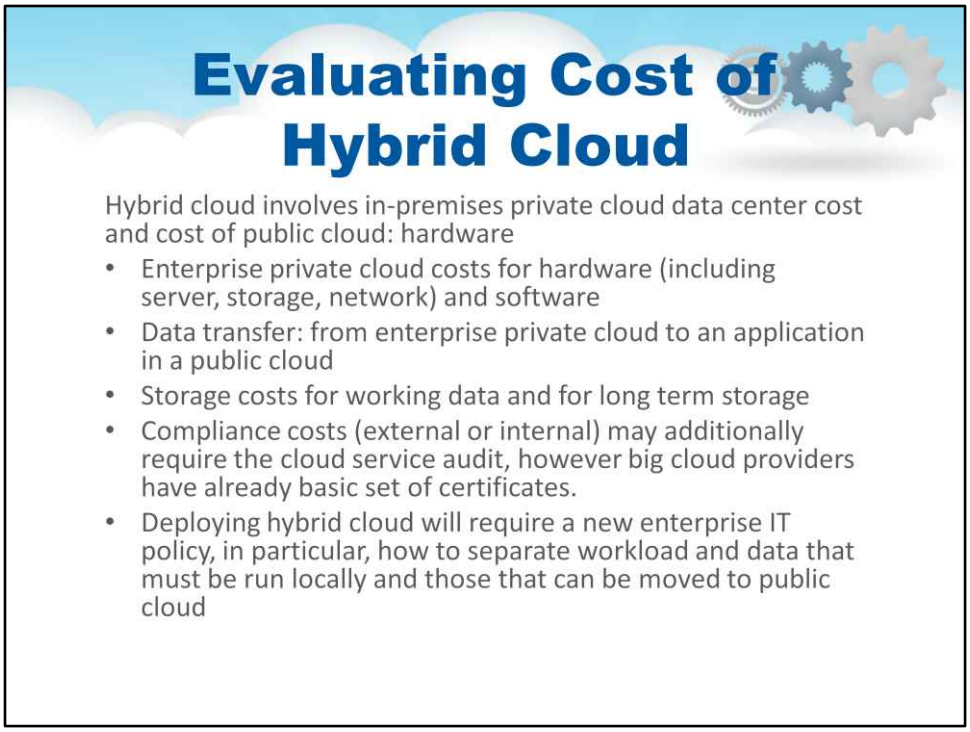
Вартість сервера (A):

З цим та всіма іншими апаратними компонентами вас особливо цікавить загальна річна вартість володіння, яка зазвичай складається з вартості підтримки апаратного забезпечення плюс деякі амортизаційні витрати на придбання апаратного забезпечення.

Витрати на зберігання (B):

У ситуаціях, коли для програми використовується мережа зберігання даних (SAN) або мережеве сховище (NAS), необхідно визначити пропорційну вартість усієї мережі SAN або NAS, включаючи витрати на управління та підтримку апаратного забезпечення.

Витрати на мережу (C):



• Витрати корпоративної приватної хмари на обладнання (включаючи сервер, сховище, мережу) та програмне забезпечення

• У внутрішньому центрі обробки даних також має працювати хмарне програмне забезпечення • Лише частина робочих навантажень буде переміщено в хмару

• Витрати на налаштування програм для ефективної роботи в хмарі

• Вартість інтеграції між зовнішніми та локальними програмами

• Витрати на оперативний допоміжний персонал зменшаться для приватного хмарного центру обробки даних

Витрати на відповідність (зовнішні чи внутрішні) можуть додатково потребувати аудиту хмарних служб, однак великі хмарні постачальники вже мають базовий набір сертифікатів.

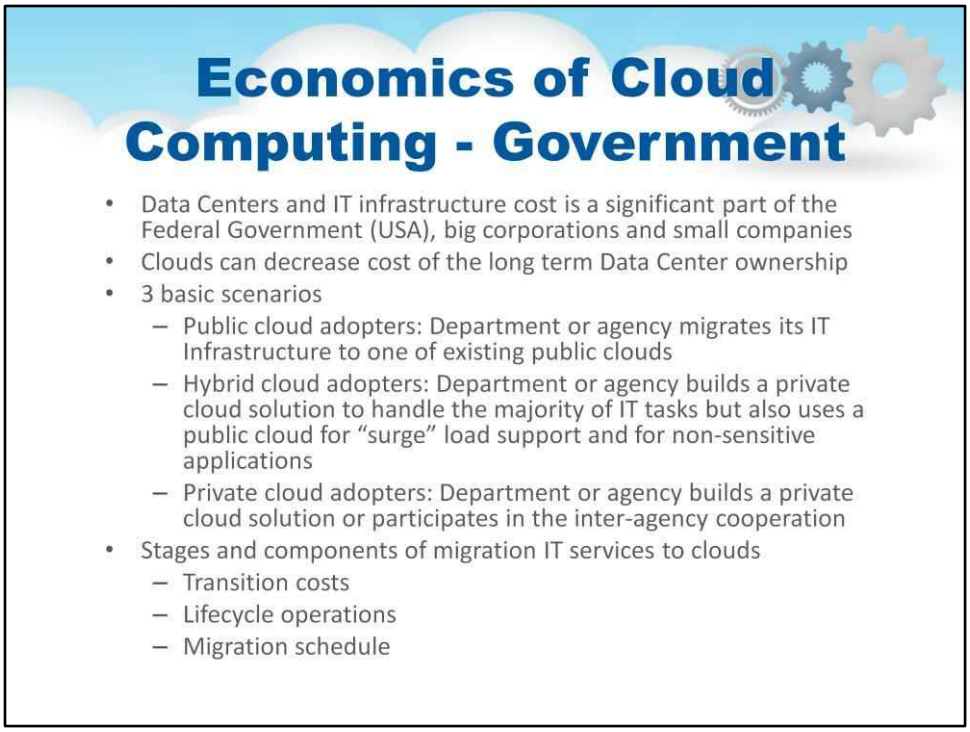
•

• Вимоги до відповідності можуть визначатися кількома органами. наприклад, PCI DSS, HIPAA, SOX, внутрішні компанії

Розгортання гібридної хмари вимагатиме нової корпоративної IT-політики, зокрема того, як розділити робоче навантаження та дані, які потрібно запускати локально, і ті, які можна перемістити в публічна хмара

•

• Безпекою даних, безпекою та конфіденційністю слід керувати по-різному для локальних і публічних хмар



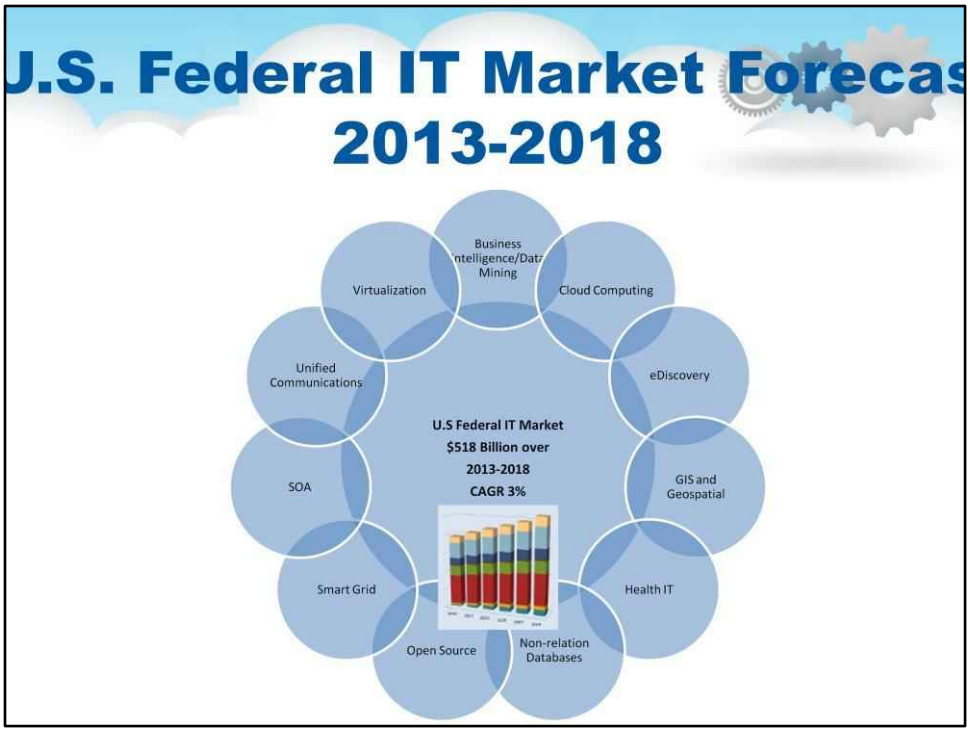
Витрати на центри обробки даних та IT-інфраструктуру становлять значну частину федерального уряду (США),

- великі корпорації та малі компанії
  - Витрати на IT зростають: $75,88 млрд у 2010 фінансовому році (2010 фінансовий рік); $88 млрд у 2011 фінансовому році
  - Оцінюється як 518 мільярдів доларів США протягом 2013-2018 років із CAGR 3% **)
  - Пошук довгострокового рішення для економії: тиск на скорочення федеральних IT-бюджетів без втрати ефективності та впровадження нових технологій: гнучкість, «оренда, а не купівля»

Хмари можуть знизити вартість довгострокового володіння ЦОД

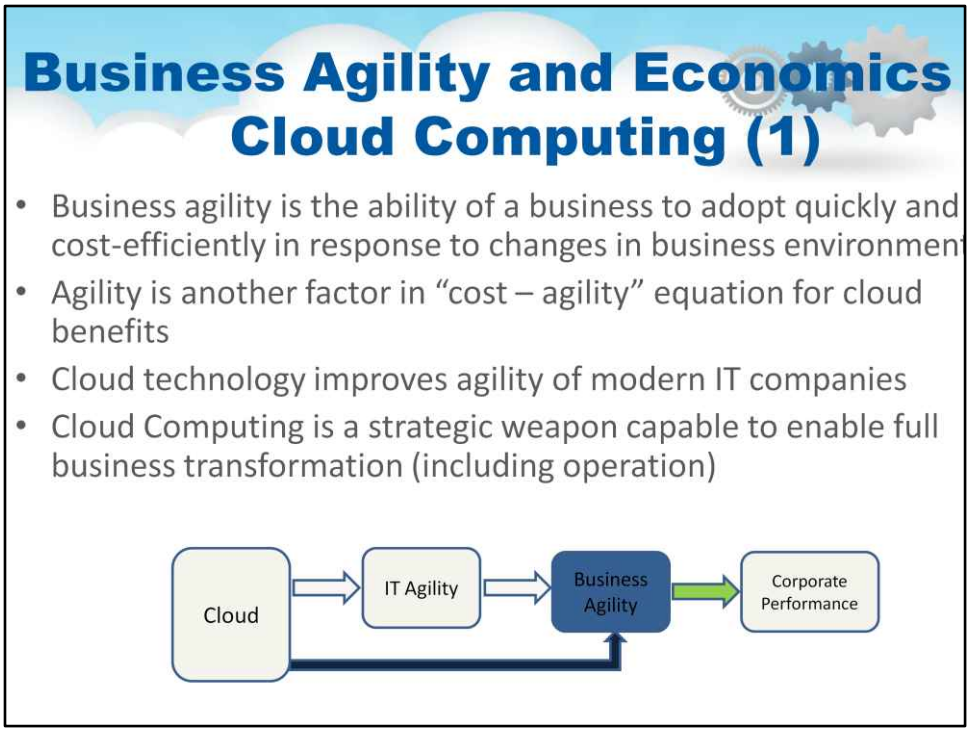
  - Впровадження та підтримка хмарного середовища протягом 13-річного циклу з економією на дві третини

- 3 базові сценарії
  - Користувачі публічної хмари: Департамент або агентство переносить свою IT-інфраструктуру до однієї з існуючих публічних хмар
    - Конкретних конфіденційних даних немає, перехідний період триває 1-2 роки
  - Адаптери гібридної хмари: Департамент або агентство створює приватне хмарне рішення для вирішення більшості IT-завдань, але також використовує загальнодоступну хмару для підтримки «стрибкового» навантаження та для нечутливих програм
    - Розрахункова частка між робочим навантаженням приватної та публічної хмари 75%/25%, приватна хмара побудована на існуючому IT-обладнанні, перехідний період триває 1-2 роки
  - Адаптанти приватної хмари: Департамент або агентство створює рішення для приватної хмари або бере участь у міжвідомчій співпраці
    - Пізні користувачі: конфіденційні дані місії, приватні, побудовані на існуючому IT-обладнанні, перехідний період триває 1-2 роки



Сектори, що швидко розвиваються

- Бізнес-аналітика
- Хмарні обчислення
- Кібербезпека
- Дедуплікація
- eDiscovery
- ГІС і геопростор
- Інформаційні технології охорони здоров'я (HIT)
- Високопродуктивне обчислення
- Нереляційні системи керування базами даних
- Відкрите джерело
- Розумна мережа
- SOA
- Віртуальні події
- Віртуалізація
- Бездротовий голос і дані.

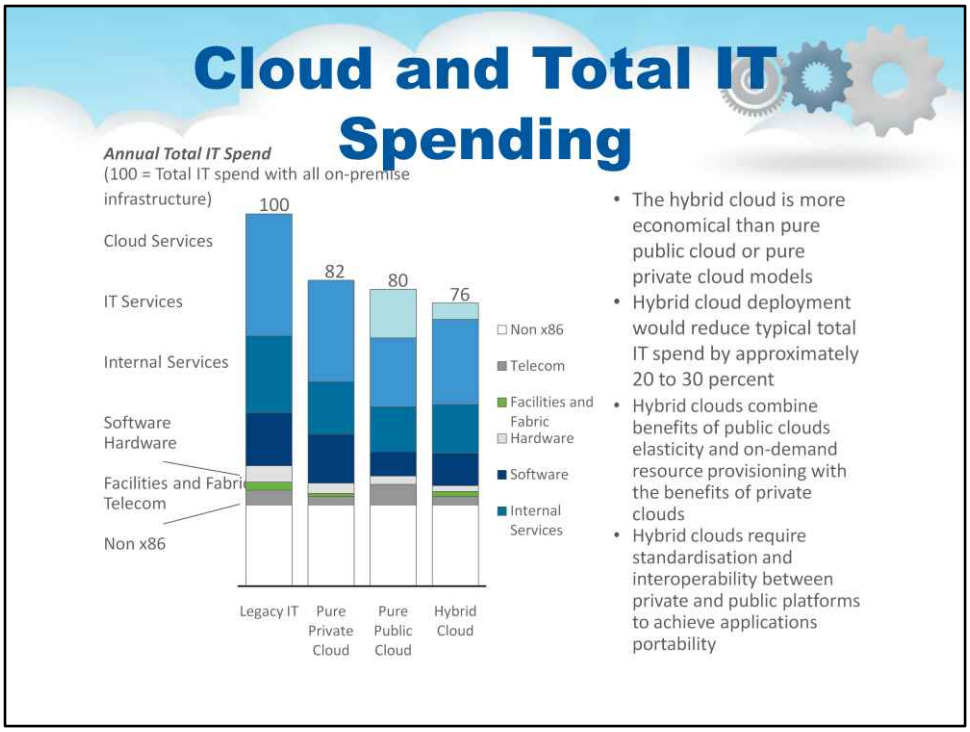


- Гнучкість є ще одним фактором у рівнянні «витрати – гнучкість» для переваг хмари
  - «Як хмара покращить конкурентоспроможність моєї компанії?»
  - Не просто «Скільки капітальних і операційних витрат можна скоротити в хмарі?»
- Хмарні технології покращують гнучкість сучасних IT-компаній
  - Тісний зв’язок між гнучкістю IT і гнучкістю бізнесу
  - Постійна трансформація бізнесу та розвиток технологій є актуальною тенденцією та новою економічною реальністю
    - Вимагає швидшої реакції
  - (Згідно з McKensey&Company) Переваги гнучкості включають швидше зростання доходу, більше та тривале скорочення витрат, ефективніше управління ризиками

Джерело: офіційний документ VMware Business

Business Agility and Economics of Cloud Computing (2)

- Clouds bring IT agility in economically efficient way
- On-demand services and pre-established cost of using compute and storage resources
- Cloud service model requires that IT departments do inventory of all cost components for specific services and applications
  - This is not a current approach in an environment of heterogeneous silos of technology
    - Cloud provides more homogeneous lower level IT platform
    - Cloud simple accountability model for using cloud resources, but running cloud based applications will involve more factors and cost components
  - Deployment models: public, hybrid, private clouds allow different split of responsibilities and costs between IT departments and cloud providers



• Гібридні хмари поєднують переваги еластичності загальнодоступних

хмар і надання ресурсів на вимогу з перевагами

приватні хмари

- Застарілі програми
- Оперативний контроль
- Конфіденційні програми та дані

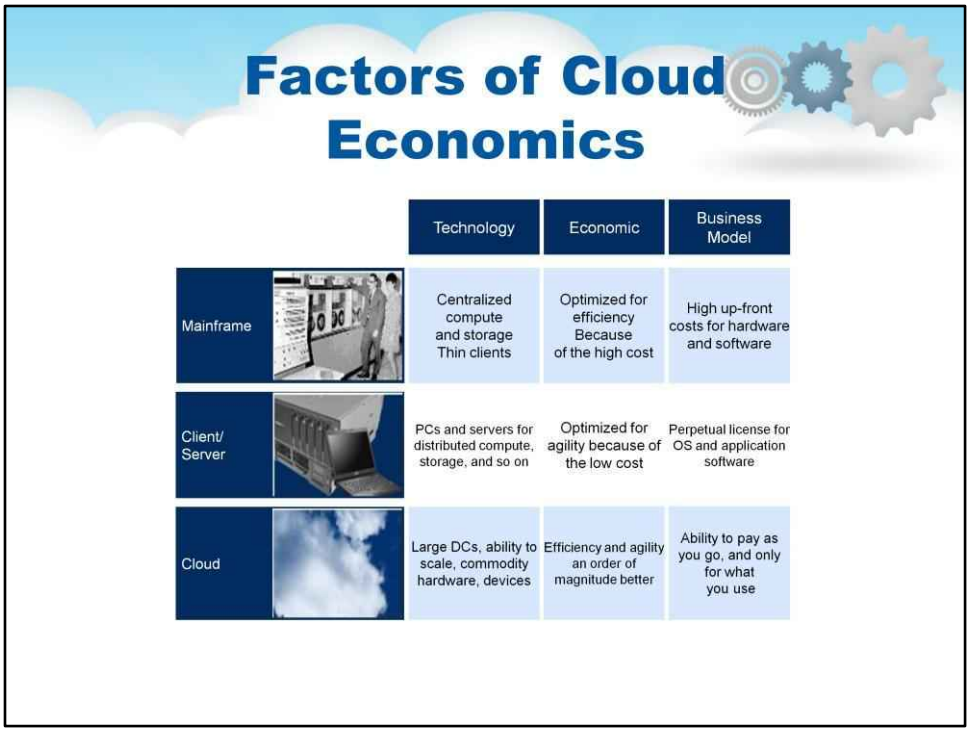
• Для досягнення гібридних хмар потрібна стандартизація та взаємодія

між приватними та публічними платформами

портативність програм

- Загальна платформа
- Загальний менеджер
- Безпека
- Відповідність

[ref] Гнучкість бізнесу та справжня економіка хмарних обчислень. Business White Paper, VMware 2011.



- Економія на стороні пропозиції: економія масштабу
    - Великі центри обробки даних (ЦОД) нижчі витрати на сервер.
- Агрегація на стороні попиту: агрегування попиту на обчислення згладжує загальну мінливість, дозволяючи підвищити рівень використання сервера.
- Ефективність роботи з кількома орендарями: при переході на модель багатокористувацької програми збільшення кількості орендарів (тобто клієнтів або користувачів) знижує витрати на керування програмою та сервер на кожного орендаря.

Економіка хмари.
Microsoft, 2010

Вартість електроенергії стрімко зростає і стає найбільшим елементом загальної вартості володіння (TCO), наразі становлячи 15%-20%.
Ефективність енергоспоживання, як правило, значно нижча у великих приміщеннях, ніж у менших.

- PUE = Загальна енергія об'єкта / Енергія об'єкта IT

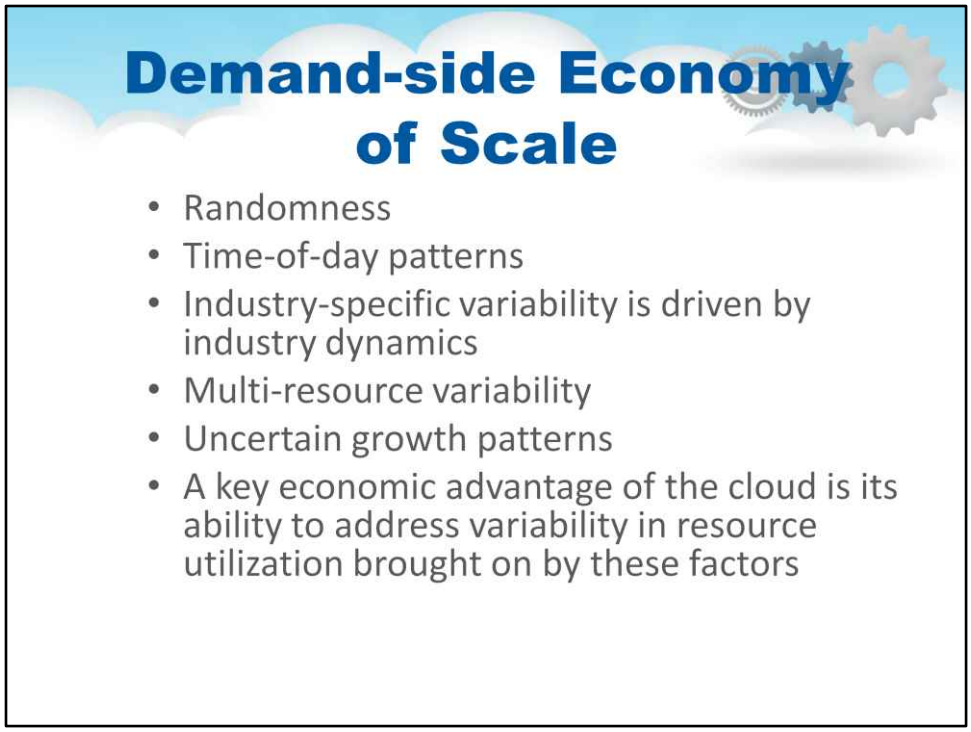
Хмарні обчислення значно знижують витрати на робочу силу завдяки автоматизації багатьох повторюваних завдань керування
Один системний адміністратор може обслуговувати приблизно 100+ серверів у традиційному підприємстві. У хмарному центрі обробки даних один адміністратор може обслуговувати тисячі серверів
Більше часу для діяльності з вищою доданою вартістю, як-от створення нових можливостей і реагування на запити клієнтів.

Загальна безпека IT-об’єктів, надійність і відповідність є поширеними проблемами традиційних і хмарних IT

- Часто згадується як потенційна перешкода для впровадження публічної хмари

Хмара знову приносить економію на масштабі завдяки переважно фіксованому рівню інвестицій, необхідних для досягнення операційної безпеки, надійності та відповідності.

- Великі комерційні хмарні постачальники можуть залучати глибокий досвід для вирішення цієї проблеми, таким чином фактично роблячи хмарні системи більш безпечними та надійними.
- Сучасні технології віртуалізації з підтримкою апаратного забезпечення дозволяють краще виконувати завдання та розділяти клієнтів у хмарному режимі з кількома клієнтами



Випадковість:

- Шаблони доступу кінцевого користувача містять певний ступінь випадковості, наприклад, перевірка електронної пошти або доступ до спільного сховища файлів.
- Щоб відповідати угодам про рівень обслуговування, створюються буфери потужності для врахування певної ймовірності попиту на послуги.

Шаблони часу доби:

- Щоденні повторювані цикли в поведінці людей: пік побутових послуг зазвичай припадає на вечір, тоді як пік послуг на робочому місці припадає на робочий день.
- Потужність має бути створена з урахуванням цих щоденних піків, але її можна оптимізувати шляхом балансування навантаження між різними схемами навантаження. Галузева мінливість залежить від динаміки галузі:
- Фірми роздрібної торгівлі спостерігають сплеск під час сезону святкових покупок, а податкові фірми США побачать пік до 15 квітня.

Різноманітність ресурсів:

- Обчислювальні ресурси, ресурси зберігання та введення/виведення (I/O) зазвичай купуються пакетами: сервер містить певну обчислювальну потужність (ЦП), зберігання та введення/виведення (наприклад, мережа або доступ до диска).
- Деякі робочі навантаження, як-от пошук, використовують багато ЦП, але відносно мало пам’яті або вводу-виводу, тоді як інші робочі навантаження, як-от електронна пошта, зазвичай використовують багато пам’яті, але мало ЦП.

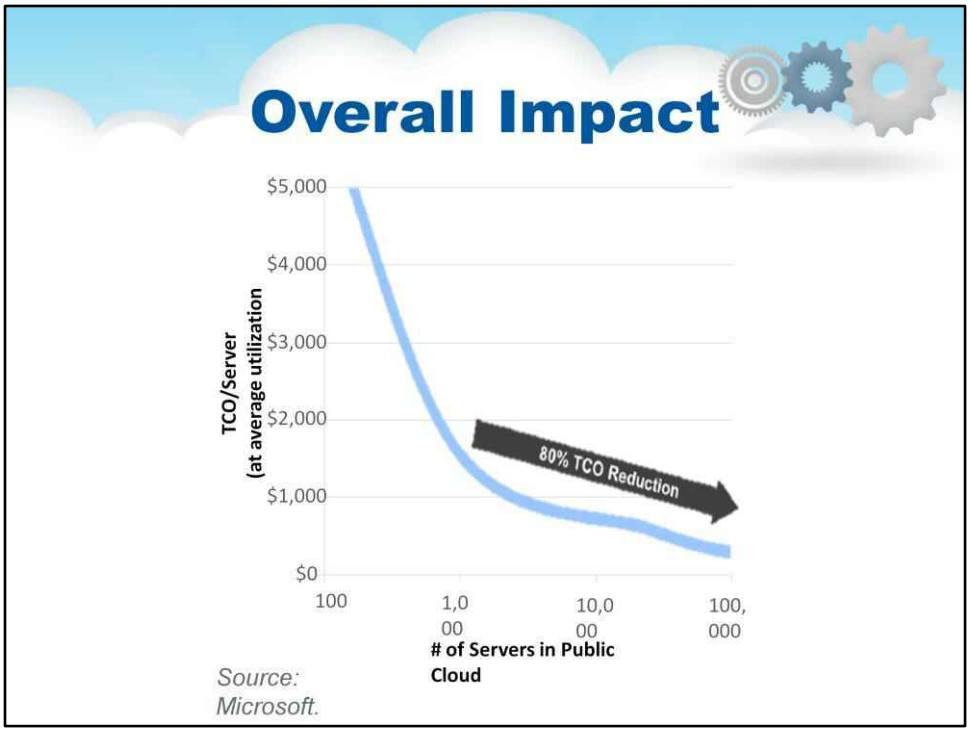
Невизначені моделі росту:

- Складність прогнозування майбутньої потреби в обчислювальних ресурсах і тривалий час для виведення потужності в режим онлайн є ще одним джерелом низького рівня використання

Ключовою економічною перевагою хмари є її здатність реагувати на мінливість у використанні ресурсів, спричинену цими факторами.

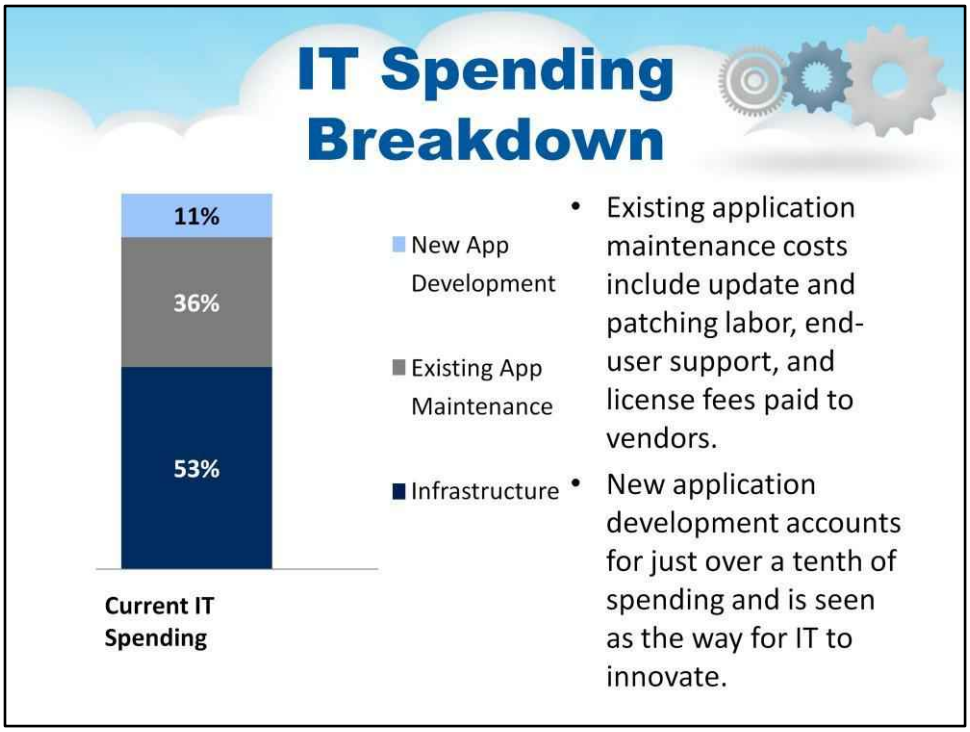
Multi-tenancy Economy of Scale

- To benefit from this source of economies of scale, the application must be written as a multitenant application, to allow isolated multi-instance operation
  - Fixed application maintenance cost/labor is amortized over a large number of customers
    - Costs associated with update and upgrade management, incident resolution, that cost is shared across a large set of customers, e.g. Microsoft Office 365 or Google Apps.
- Applications can benefit from using shared services provided by the cloud platform
  - The greater the use of such shared services, the greater the application will benefit from the multi-tenancy economies of scale.

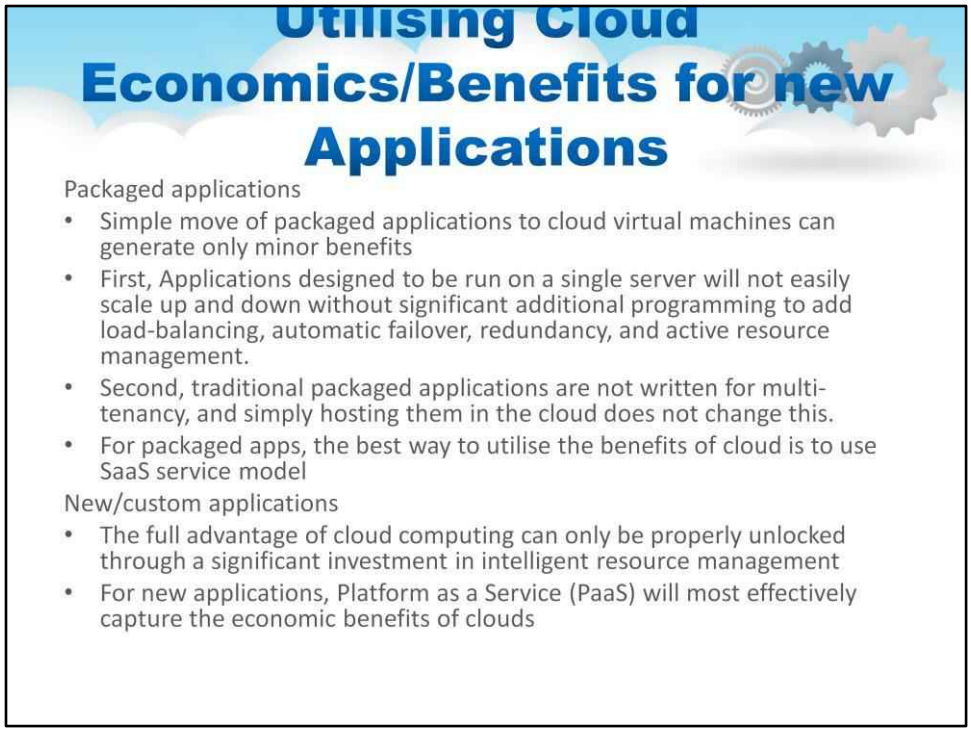


- Поєднання економії масштабу на серверній потужності з боку постачання (амортизація витрат на більшій кількості серверів), агрегації робочих навантажень на стороні попиту (зменшення мінливості) і багатокористувацької моделі додатків (амортизація витрат на кількох клієнтів) веде до значної економії. масштабу.

- Модель показує, що загальна вартість володіння (TCO) центру обробки даних із 100 000 серверів нижча на 80 % порівняно з центром обробки даних із 1 000 серверів.



• Існуючі витрати на технічне обслуговування програми включають роботу з оновлення та виправлення, підтримку кінцевих користувачів і ліцензійні збори постачальники.
    • На них припадає приблизно третина витрат, і вони орієнтовані на коефіцієнт ефективності багатоквартирних будинків.
• На розробку нових додатків припадає трохи більше десятої частини витрат і розглядається як шлях для IT-інновацій.
    • Економічні переваги хмарних обчислень дозволять збільшити зусилля та бюджет на інновації.



Просте переміщення упакованих програм до хмарних віртуальних машин може генерувати лише

• незначні переваги

По-перше, програми, призначені для роботи на одному сервері, нелегко розширити та розширити

• вимкнено без істотного додаткового програмування для додавання балансування навантаження, автоматичного перемикання після відмови, резервування та активного керування ресурсами.

- Це обмежує ступінь, до якого вони здатні агрегувати попит і збільшити використання сервера.
- Для ефективного використання в хмарах програми потрібно (пере)проектувати/ перебудувати

По-друге, традиційні упаковані програми не написані для багатокористувацького використання

• просте розміщення їх у хмарі не змінює цього.

Для пакетних додатків найкращий спосіб скористатися перевагами хмари – використовувати SaaS

• модель обслуговування

- Наприклад, Office365, який розроблено для масштабування та мультиарендації, щоб отримати всі переваги хмарних платформ Нові/ спеціальні програми

• Усі переваги хмарних обчислень можна належним чином розблокувати лише через а

• значні інвестиції в інтелектуальне управління ресурсами

- Менеджер ресурсів повинен розуміти як статус ресурсів (мережа, сховище та обчислення), так і активність програм. Для нових програм платформа як послуга (PaaS) найефективніше фіксуватиме економічні переваги хмар

•

- PaaS пропонує спільні послуги, розширене управління та автоматизацію функції, які дозволяють розробникам зосередитися безпосередньо на логіці програми, а не на розробці програми для масштабування.

Хмари як фасилітатор IT-новаторів

Cloud as IT Innovation facilitator: Possibilities

Cloud computing will free up significant resources that can be redirected to innovation. However, total cost of IT will increase as modern economy increasingly use new possibilities of computer based and data intensive technologies. Besides lower TCO the following factors will lead to a renewed level of innovation in IT:

- Elasticity and on-demand provisioning together with resources pooling and effective load balancing will enable users and organizations to rapidly accomplish complex tasks that were previously prohibited by cost or time constraints.
  - Being able to both scale up and scale down resource intensity nearly instantly enables a new class of experimentation and entrepreneurship.
- Elimination of capital expenditure will significantly lower the risk premium of projects, allowing for more experimentation.
  - This both lowers the costs of starting an operation and lowers the cost of failure or exit; if an application no longer needs certain resources, they can be decommissioned with no further expense or write-off.
- Self-service and a simple web portal based business model allows projects to be completed in less time with less risk and lower administrative overhead than previously.
- Reduction of complexity of writing and testing new application, e.g. with using PaaS based development platforms.

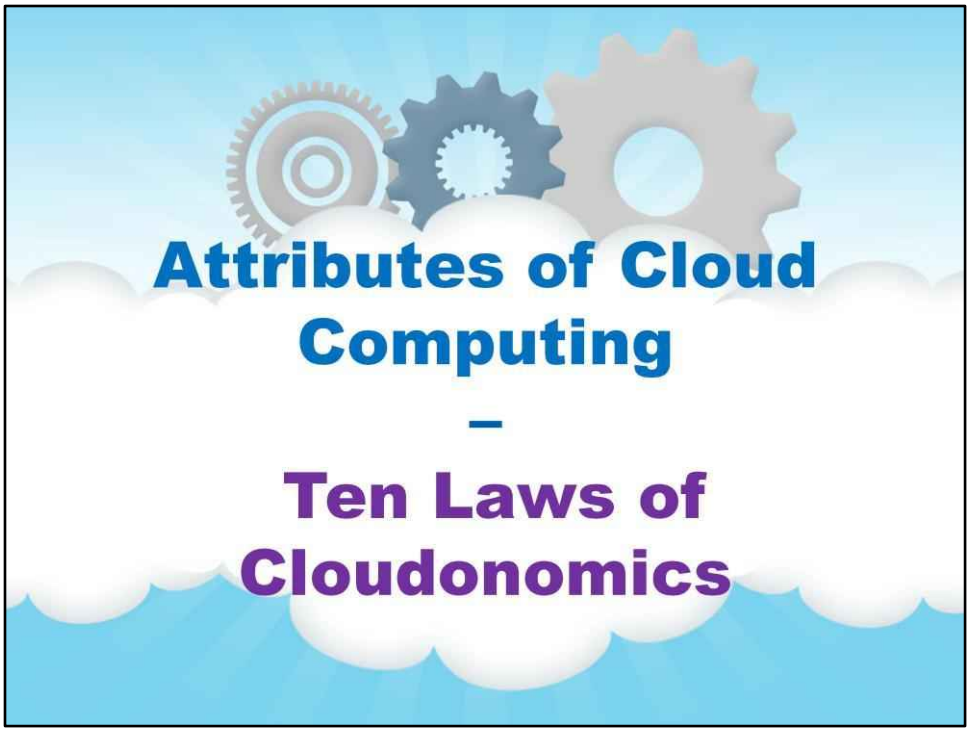
Cloud as IT Innovation facilitator: Obstacles

Security, privacy, maturity, and compliance are the top concerns, followed by concerns about legacy (backward) compatibility.

- It is often not straightforward to move existing applications to the cloud; once they moved there may be no way for backward compatibility.
- Financially and strategically important data and processes often are protected by complex security requirements.
  - Legacy systems have typically been highly customized to achieve these goals, and moving to a cloud architecture can be challenging.
  - Experience with the built-in, standardized security capabilities of cloud is still limited and many CIOs still feel more confident with legacy systems in this regard.
- Maturity: Moving services to cloud requires certain level of maturity of the company's on premises IT infrastructure to benefit from cloud opportunities
- Availability and Performance
  - Cloud requires CIOs/company to trust cloud provider to provide reliable and highly available services
- Compliance and Data Sovereignty
  - Enterprises are subject to audits and oversight, both internal and external (e.g. IRS, SEC).
  - Companies in many countries have data sovereignty requirements that severely restrict where they can host data services.
  - Many big cloud providers have their facilities certified against PCI DSS, HIPAA, SOX, others

What is the way to go?

- Traditional Data Center will definitely undergo transformation to private cloud platform keeping some legacy facility and applications.
- Private cloud will provide benefits and economy in simplifying IT infrastructure management and maintenance, including backup and disaster recovery, allowing also flexible models to integrate with external cloud based resources.
- Public cloud is a choice for smaller businesses and companies running massive/global web-scale businesses (e.g., social networks, mobile applications, gaming, content streaming, consulting and outsourcing services).
- Hybrid cloud provide benefits of both private and public clouds and are optimal for medium sized and large companies, who also owning own data centers. Hybrid clouds bring maximum benefits for modern agile companies.



Десять законів хмарономіки

The Ten Laws of Cloudonomics (1)

Created by Joe Weinman in 2008, then Strategic Solutions Sales VP for AT&T Global Business Services, are still the foundation for the economics of Cloud Computing.

Cloudonomics Law #1: Utility services cost less even though they cost more. Although utilities cost more when they are used, they cost nothing when they are not. Consequently, customers save money by replacing fixed infrastructure with Clouds when workloads are spiky, specifically when the peak-to-average ratio is greater than the utility premium.

Cloudonomics Law #2: On-demand trumps forecasting. Forecasting is often wrong, the ability to up and down scale to meet unpredictable demand spikes allows for revenue and cost optimalities.

Cloudonomics Law #3: The peak of the sum is never greater than the sum of the peaks. Enterprises deploy capacity to handle their peak demands. Under this strategy, the total capacity deployed is the sum of these individual peaks. However, since Clouds can reallocate resources across many enterprises with different peak periods, a Cloud needs to deploy less capacity.

Cloudonomics Law #4: Aggregate demand is smoother than individual. Aggregating demand from multiple customers tends to smooth out variation. Therefore, Clouds get higher utilization, enabling better economics.

Cloudonomics Law #5: Average unit costs are reduced by distributing fixed costs over more units of output. Larger Cloud providers can therefore achieve some economies of scale.

The Ten Laws of Cloudonomics (2)

Cloudonomics Law #6: Superiority in numbers is the most important factor in the result of a combat (Clausewitz). Service providers have the scale to fight rogue attacks.

Cloudonomics Law #7: Space-time is a continuum. Organizations derive competitive advantage from responding to changing business conditions faster than the competition. With Cloud scalability, for the same cost, a business can accelerate its information processing and decision-making.

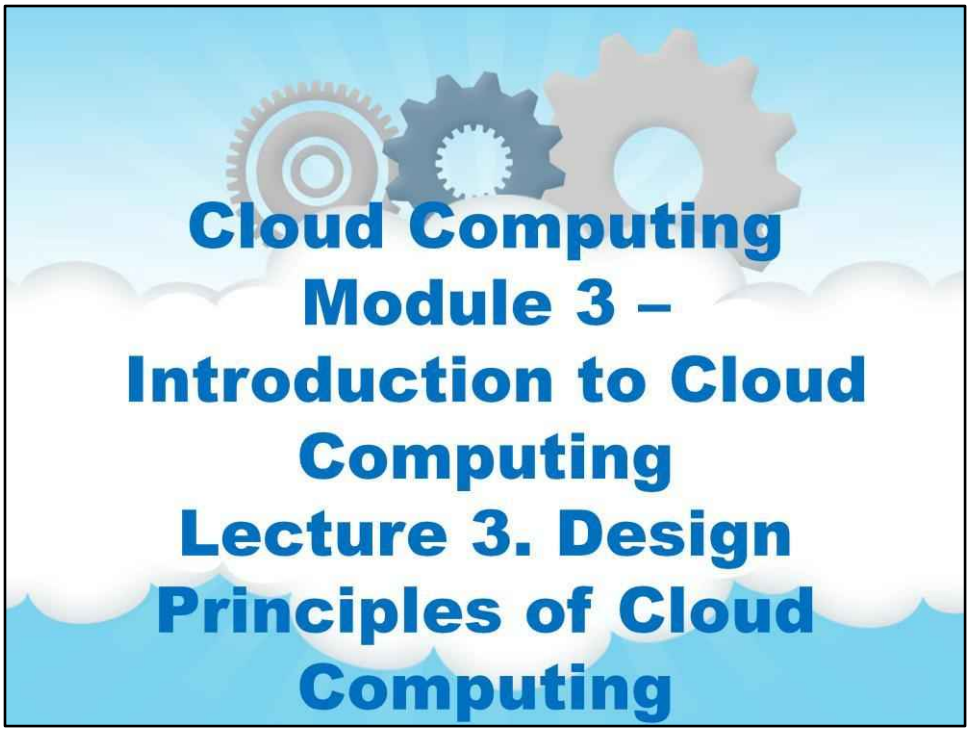
Cloudonomics Law #8: Dispersion is the inverse square of latency. Reduced latency is increasingly essential to modern applications. A Cloud Computing provider is able to provide more nodes, and hence reduced latency, than an enterprise would want to deploy.

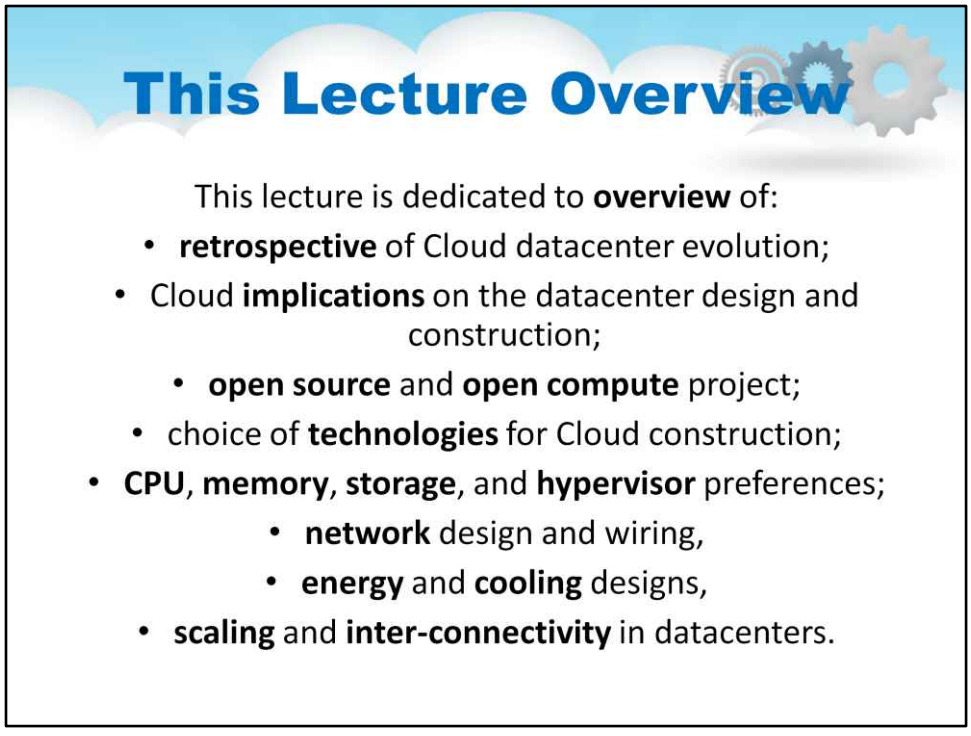
Cloudonomics Law #9: Don't put all your eggs in one basket. The reliability of a system increases with the addition of redundant, geographically dispersed components such as data centers. Cloud Computing vendors have the scale and diversity to do so.

Cloudonomics Law #10: An object at rest tends to stay at rest. A data center is a very large object. Private data centers tend to remain in locations for reasons such as where the company was founded, or where they got a good deal on property. A Cloud service provider can locate greenfield sites optimally.

Summary and Take away

- Cloud Computing has many benefits as a new technology and as IT infrastructure design and management transformation factor
- At the same time Cloud Computing and a number of restraining factors, main of which is security of data and services or infrastructure in clouds
- Cloud Total Ownership Cost (TOC) structure is complex and not such simple as it presented by the Cloud Service Providers
  - Besides costs of running services and applications in clouds, it includes also include transition costs, possible applications re-design, and in-premises IT management
  - For hybrid clouds TOC includes both cost of private cloud and services outsourcing to public cloud
- Cloud Economy of scale comprise of 3 factors: demand/customer side, supply/provider side, and multi-tenancy
- Moving enterprise IT infrastructure bring an agility both for IT infrastructure and to company itself
- Cloud as IT innovation facilitator can free significant resource in company which can be used for the company's main business


Cloud Computing
Module 3 –
Introduction to Cloud
Computing
Lecture 3. Design
Principles of Cloud
Computing



Лекція 3. Принципи проектування Хмарні обчислення

Контур

У цій лекції ми висвітлюємо

Еволюція хмарного центру обробки даних

Вплив хмари на проектування та будівництво центру обробки даних

Технології проектів з відкритим вихідним кодом і

Open Compute Вибір для хмарного будівництва

Параметри ЦП і гіпервізора

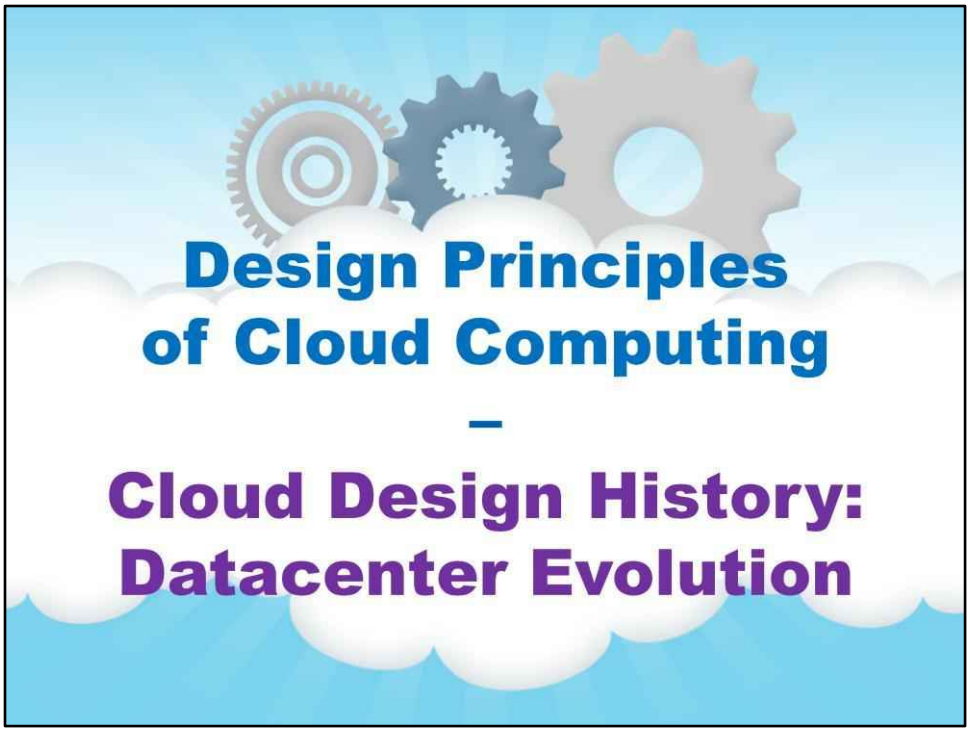
Пам'ять і сховище

Проектування мережі та електропроводки

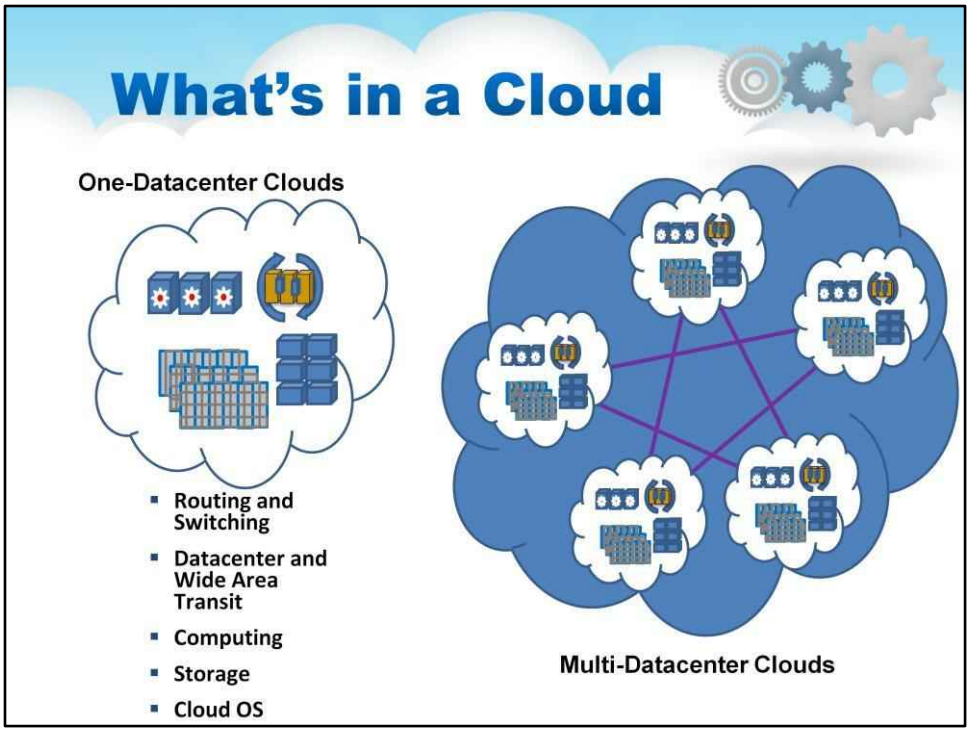
Проектування енергопостачання та охолодження

Масштабування та підключення між центрами обробки даних

Підведення підсумків і підсумок



Історія хмарного дизайну



Що в хмарі

Маршрутизація та комутація

Обчислення центрів обробки даних і

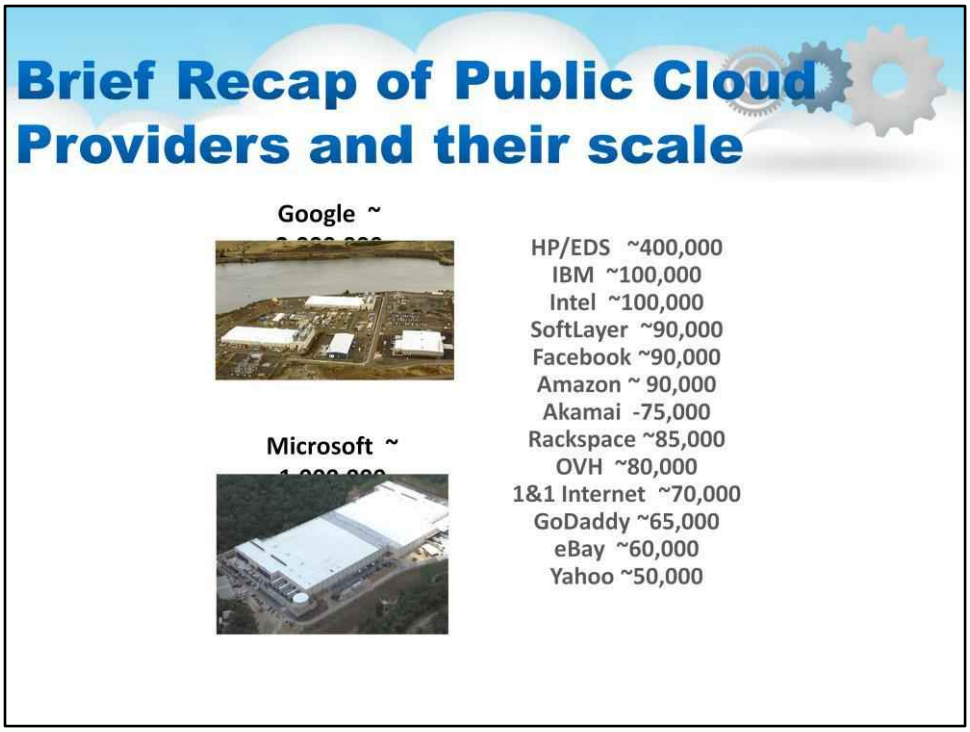
глобального транзиту

Зберігання

Хмарна ОС

Хмари з одним центром обробки даних

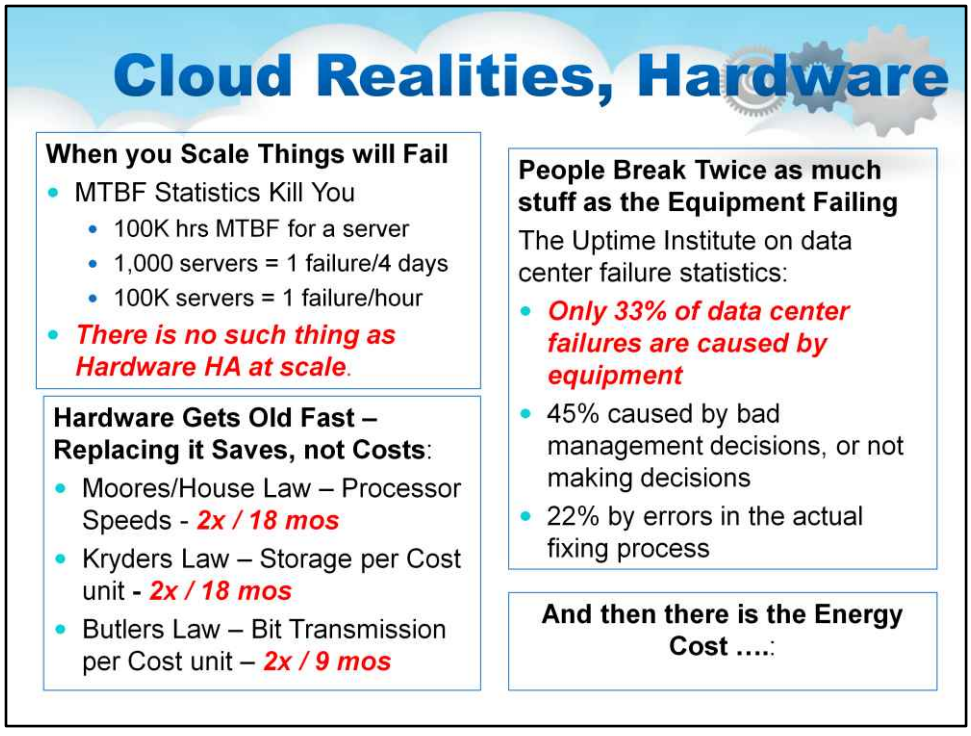
Хмари з кількома центрами обробки даних



Короткий огляд постачальників публічної хмари та їх масштаб

Найбільші публічні хмарні постачальники продовжують масове будівництво. Хоча фізичні масштаби цих центрів обробки даних одразу очевидні, програмне забезпечення та досягнення процесів для керування інфраструктурою такого розміру не менш вражаючі.

Як можна побачити, ці постачальники придумали, як запустити хмари буквально на мільйонах серверів. Як вони це роблять?



Хмарні реалії, обладнання

Щоб зрозуміти цей рівень масштабування, нам потрібно розглянути деякі холодні реалії апаратного забезпечення, статистику та динаміку відмов.

З точки зору статистики, цифри не на вашій стороні, що стосується підтримки роботи, коли у вас на роботі великі цифри.

Наприклад, розглянемо, як статистика MTBF вбиває вас. Якщо у вас є сервер із напрацюванням на відмову 100 тисяч годин для сервера

Якщо у вас 1000 серверів = 1 збій/4 дні. Якщо у вас 100K серверів = 1 збій/годину. У масштабі такого поняття, як апаратна HA, не існує.

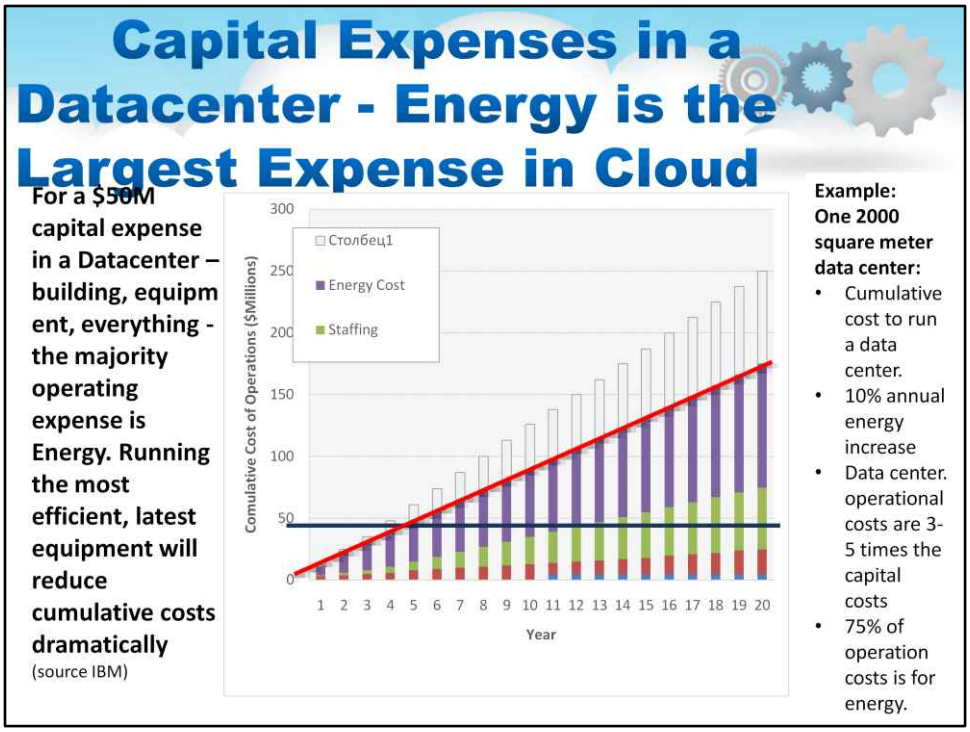
З боку апаратного забезпечення розгляньте «ефективність капіталу».умови збереження серверів і обладнання протягом навіть половини циклу амортизації

18-місячний сервер із використанням капіталу наполовину ефективнішим, ніж новий (Moores Law)

18 місстарий резерв пам’яті коштуватиме ½ дешевше, якщо придбати його сьогоднітоді (Закон Крайдера)

18 місстарий набір мережевих портів коштуватиме ½ вартості, якщо придбати його сьогодні (БатлерсЗакон) Це означає, що апаратне забезпечення швидко старіє – його заміна економить, а не коштує! Не існує

причина для ремонту старого, зламаного обладнання, це дуже неефективне використання капіталу



Енергія – найбільша витрата в хмарі

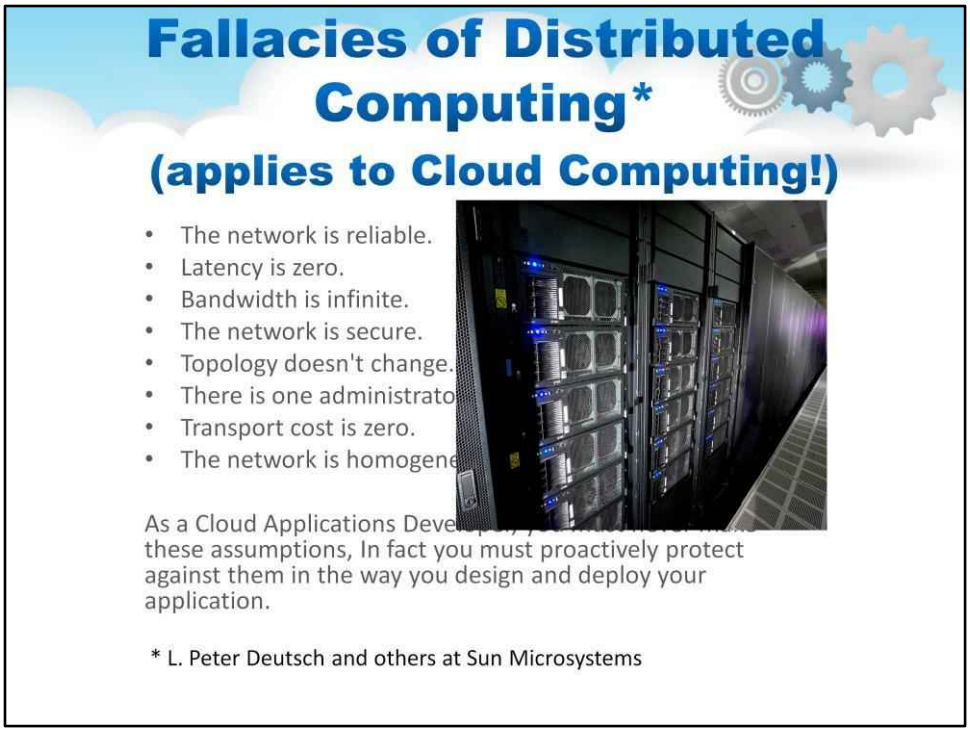
На цій діаграмі показано, що кумулятивно більшість операційних витрат становлять енергія

Для 50 мільйонів доларів капітальних витрат на центр обробки даних – будівля, обладнання, усе – більшість операційних витрат – це енергія. Використання найефективнішого найновішого обладнання значно зменшить сукупні витрати

Як бачимо, сукупні витрати на експлуатацію хмарного центру обробки даних у 3-5 разів перевищують капітальні витрати.

Це говорить нам про те, що найефективнішою умовою для використання хмарного центру обробки даних є використання максимально енергоємної конфігурації.

Огляд

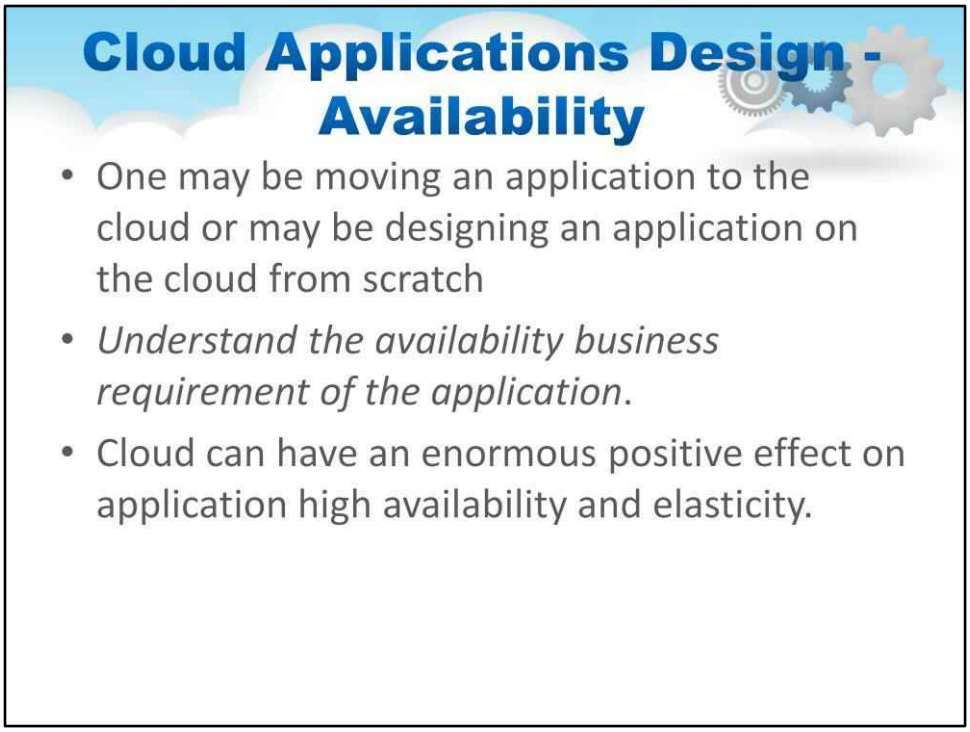


Хмара не є сервером. Це не спеціальна колекція серверів із лише потрібними функціями для автоматичного створення магічних речей вашої програми з масштабованістю чи доступністю,

Це розподілена обчислювальна система, і, ймовірно, досить велика. Отже, застосовуються правила розподілених обчислень. Що це за правила? Коли дослідник вперше почав розміщувати програми на розподілених обчислювальних платформах, програми зламалися кількома цікавими способами. Дослідники зрозуміли, що це тому, що вони робили припущення щодо платформи, які просто не відповідають дійсності. Окремо ці припущення часто були вірними для сервера розробки або двох. Але в світі а розподіленої системної платформи, як-от хмара, віра в будь-яку з цих «помилок розподілених обчислень», які ми тепер будемо називати «помилками хмари», зрештою виявляться фатальними для вашої програми.

«Помилки Cloud» описані тут, на цьому слайді.

Як розробник хмарних додатків, ви ніколи не повинні робити ці припущення. Насправді ви повинні завчасно захищати від них під час розробки та розгортання свого



Перш ніж приступити до написання або переміщення програми, спочатку слід розглянути дизайн хмарних програм щодо доступності. У цю категорію ми включили як «Стійкість до відмов», так і «Здатність обслуговувати в міру зростання попиту на масштаби». Зрозумійте бізнес-вимоги доступності програми:

Чи абсолютно важливо, щоб додаток мав фактично нульовий можливий час простою в будь-якій ситуації?

Іншими словами, чи поставлено під загрозу життя чи саме функціонування бізнесу залежить від цієї програми?

Або чи прийнятний для програми 5-хвилинний період «перезапуску» або «перемикання»?

Крім того, чи зазнає програма навантаження, яке сильно змінюється, наприклад, наприкінці кожного місяця навантаження на програму зростає від 1x до 100x або 1000x?

Який вплив на бізнес, якщо програма працює, але протягом певних коротких періодів працює повільно, чи це прийнятно?

Хмара може мати величезний позитивний вплив на високу доступність і еластичність програм.

Але ці можливості не з’являються автоматично, вони повинні бути розроблені та активовані/закодовані розробником/розробником

Щоб розробити правильну стратегію в хмарі, потрібно спочатку зрозуміти бізнес-вимоги доступності програми:

Чи абсолютно важливо, щоб будь-яка програма мала час простою практично нульовий?

Далі необхідно розглянути варіант розгортання програми в хмарі. Справа не в тому, щоб просто використовувати хмару як один великий сервер або групу серверів, як у фізичному світі.

Перш за все, ви отримаєте результат однієї або кількох «помилок хмарних обчислень». По-друге, ви отримаєте лише деякі переваги хмарних обчислень. Так що Варто приділити час, щоб зрозуміти деякі технічні деталі програми. На слайді детально описано кілька ключових питань щодо дизайну хмарних програм. Слід розуміти архітектуру програми.

Зрозумійте деякі технічні деталі програми:

Ви пишете цю нову програму на найновішій інфраструктурі? Ви змінюєте платформу відносно сучасної програми, яка працює на останній версії операційної системи та проміжного програмного забезпечення?

У вас є вихідний код?

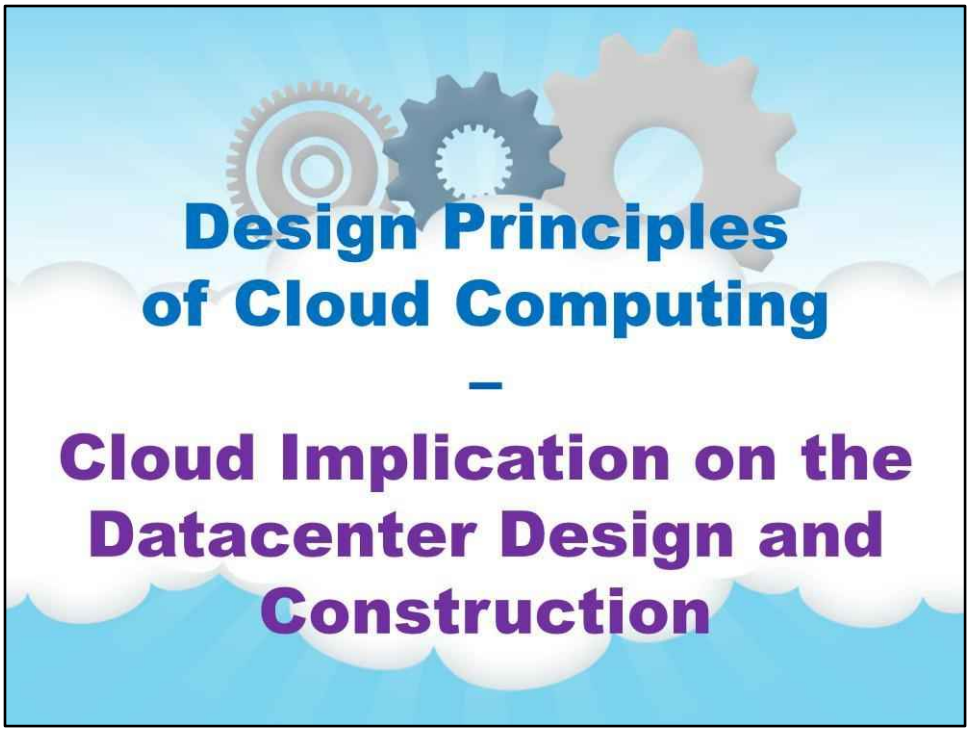
Чи підтримується ця версія постачальником програми?

Як різні частини програми взаємодіють між собою? Можливо, це старіша програма, яка мала застарілу платформу вимоги та не підтримує (і не підтримуватиме) останню інфраструктуру? Чи використовує він власну вбудовану базу даних чи її зовнішні стандарти компонент бази даних?

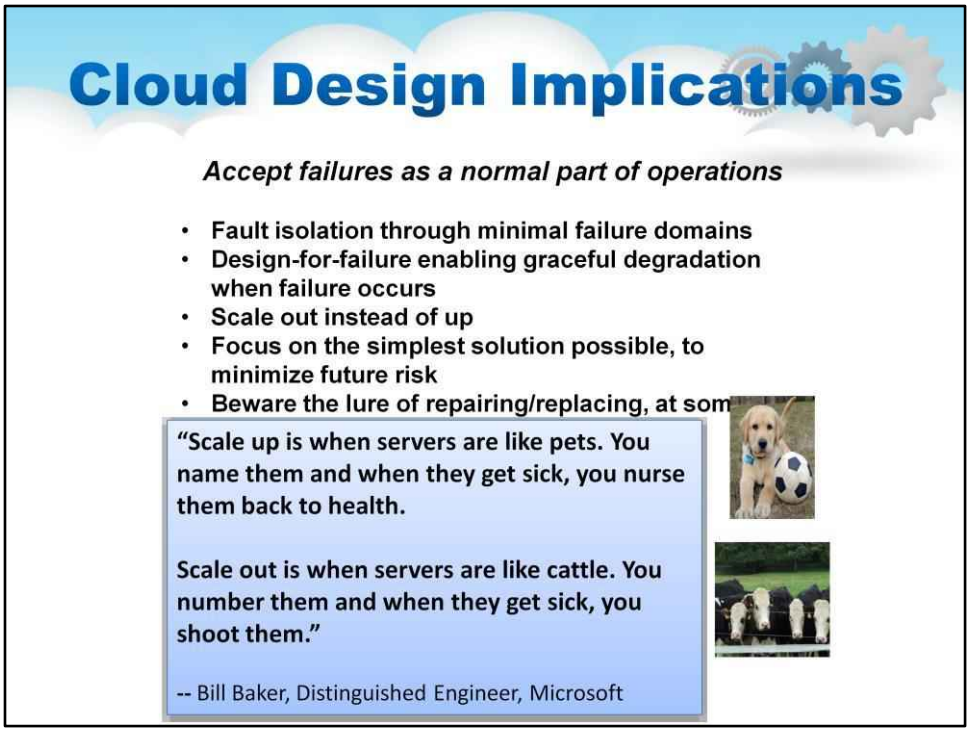
Чи програма керує швидко мінливими даними, які потребують постійного оновлення пам’яті, чи це інструмент, який отримує доступ до переважно статичної (або час від часу оновлюваної) інформації?

Як він записує в сховище?

Розуміння стану та стійкості є ключем до простої хмарної програми.



Вплив хмари на проектування та будівництво центру обробки даних



Наслідки дизайну хмари

Ці масштабні статистичні дані показують нам, що в хмарному дизайні потрібно сприймати збої як нормальну частину операцій

Ось філософія, щоб проілюструвати тезу

Цю досить відому цитату, яка існує в багатьох варіаціях, спочатку можна віднести до Білла Бейкера, заслуженого інженера Microsoft

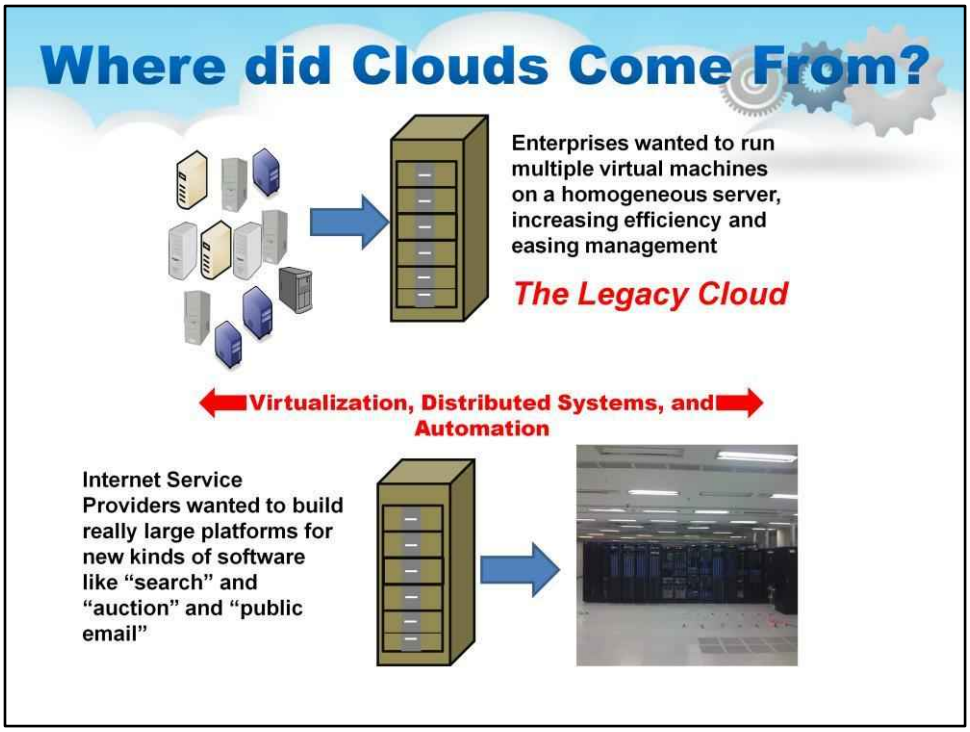
«Збільшення — це коли сервери схожі на домашніх тварин. Ви називаєте їх, а коли вони хворіють, ви доглядати за ними.

Масштабування — це коли сервери схожі на худобу. Ви їх пронумеруйте і коли вони отримають хворі, ти їх стріляєш».

Наслідки:

Ізоляція несправностей за допомогою мінімальних доменів відмов Проектування для відмов, що забезпечує плавну деградацію в разі виникнення несправностей Масштабування замість збільшення

Зосередьтеся на найпростішому можливому рішенні, щоб мінімізувати майбутній ризик. Остерігайтеся спокуси ремонту/заміни, у певному масштабі, просто вимкніть його. Оновіть, як тільки фінансовий директор дозволить вам



Звідки взялися хмари?

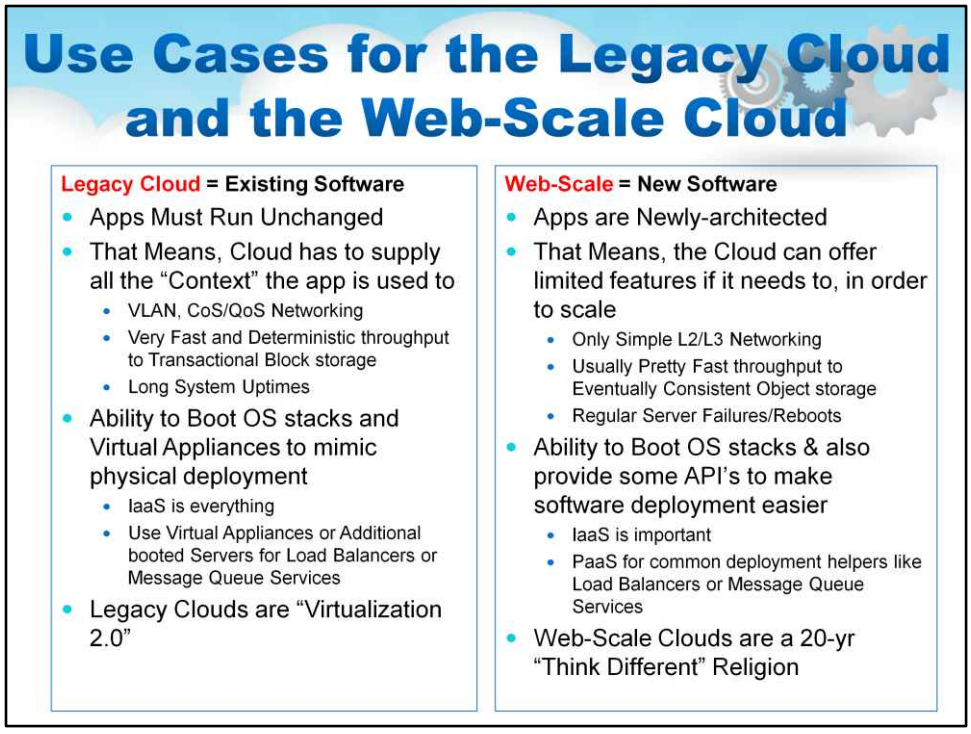
Щоб зрозуміти, як з’явилися перспективи створення хмар, потрібно розглянути, звідки з’явилися хмари.

З одного боку, Enterprises хотіли запускати кілька віртуальних машин на однорідному сервері, підвищуючи ефективність і полегшуючи керування

Ми називаємо це Legacy Cloud. Ці хмари не містять принципів проектування для досягнення великого масштабу та ефективного використання капіталу

З іншого боку, Інтернет-провайдери хотіли створити дійсно великі платформи для нових видів програмного забезпечення, таких як «пошук», «аукціон» і «публічна електронна пошта» (звучить як ІнтернетПрограми, якими ви користувалися раніше?)

Ми називаємо це Web-Scale Cloud. Ці хмари ДІЙСНО містять принципи проектування для масштабування. Ми розглянемо ці принципи дизайну докладніше.



Варіанти використання Legacy Cloud і Web-Scale Cloud

Застаріла хмара = Існуюче програмне забезпечення

Web-Scale = Нове програмне забезпечення

У застарілій хмарі можливість завантажувати стеки ОС і віртуальні пристрої для імітації фізичного розгортання

IaaS - це все

Використовуйте віртуальні пристрої або додаткові завантажені сервери для балансувальників навантаження або служб черги повідомлень

У хмарі Web Scale можливість завантажувати стеки ОС, а також надають деякі API для полегшення розгортання програмного забезпечення

IaaS є важливим

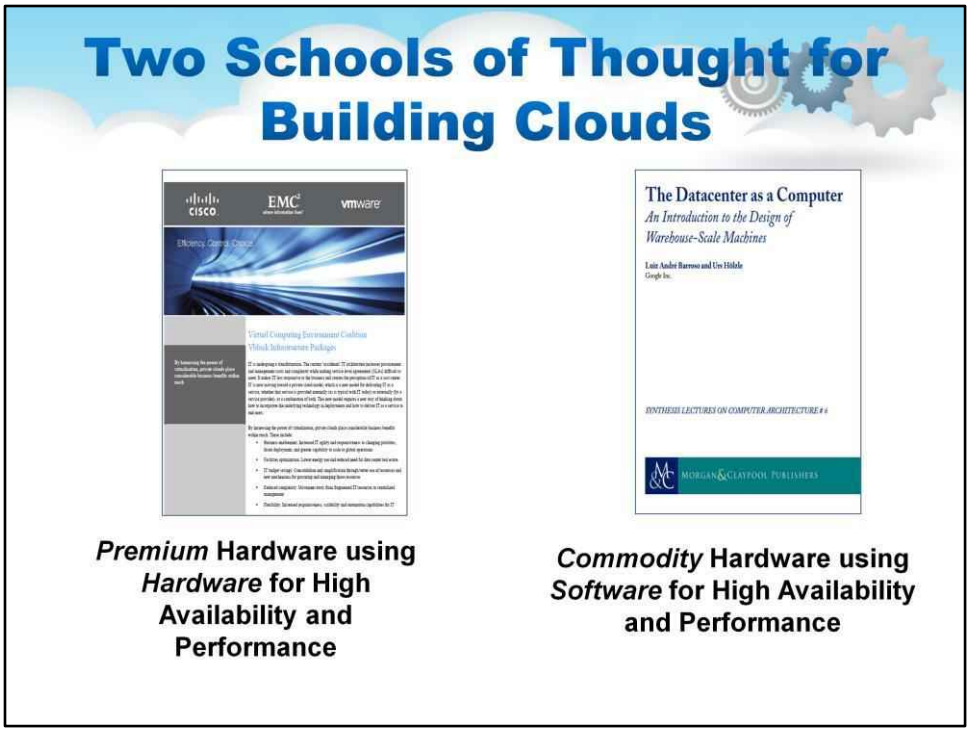
PaaS для поширених помічників розгортання, таких як балансувальники навантаження або служби черги повідомлень

Прочитайте інші області порівняння

Висновок

Застарілі хмари – це «віртуалізація 2.0»

Web-Scale Clouds — це 20-річна релігія «Думай по-іншому».



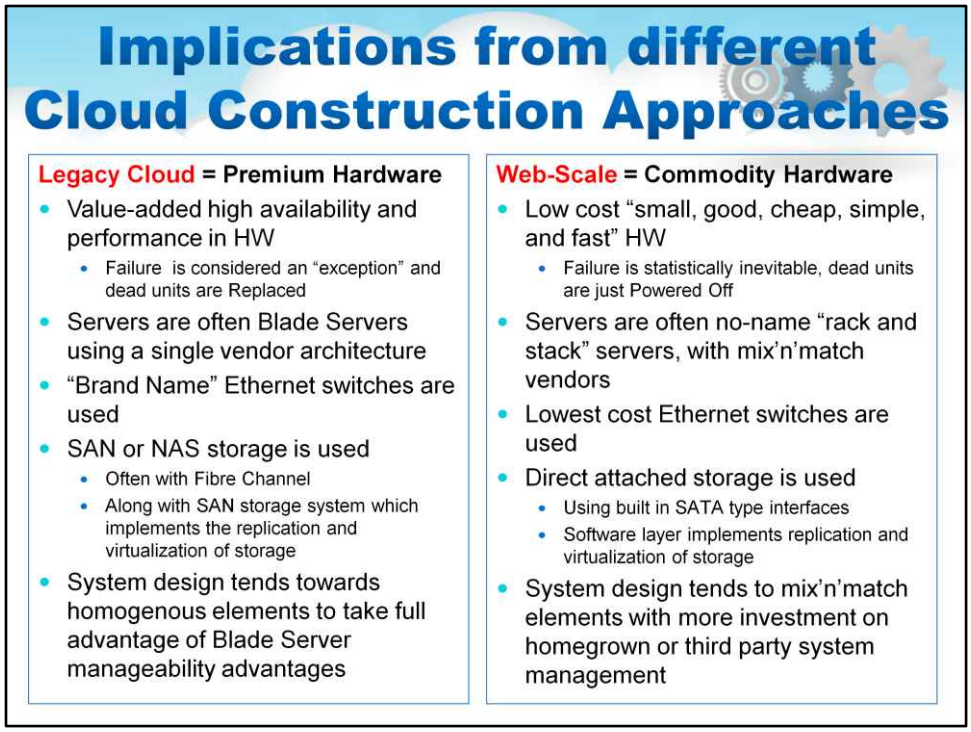
Дві школи думки для створення хмар

Існує дві школи думок щодо створення хмар: одна для застарілої хмари та інша для веб-масштабної хмари

Для Legacy Cloud класичний план передбачає використання обладнання для високої доступності та продуктивності

Для хмари Web Scale використовується програмне забезпечення для високої доступності та продуктивності

Література для цих двох схем проілюстрована на слайді



Наслідки різних підходів до створення хмари

Застаріла хмара = веб-масштаб апаратного забезпечення

преміум-класу = стандартне обладнання

Наприклад, на діаграмі показано багато порівнянь

Застаріла хмара, висока доступність і продуктивність із доданою вартістю в HW Failure вважається «винятком», і мертві блоки замінюються

Хмара веб-масштабу, низька вартість «маленький, хороший, дешевий, простий і швидкий» Збій HW статистично неминучий, мертві пристрої просто вимикаються

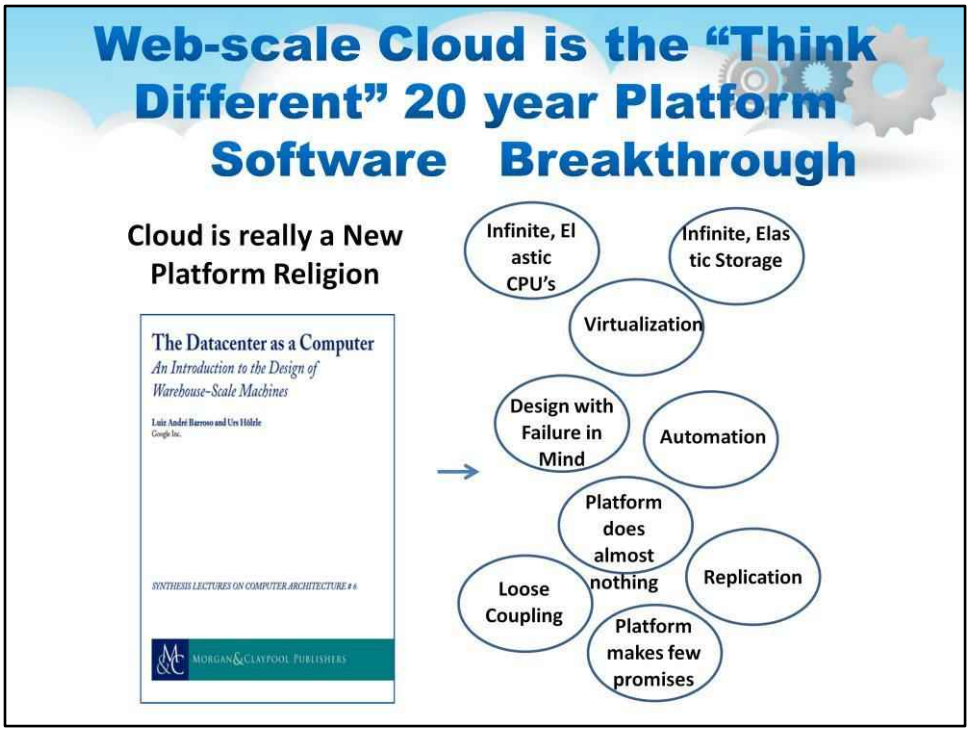
Застаріла хмара, сервери часто є блейд-серверами, які використовують архітектуру одного постачальника

Веб-масштабована хмара. Сервери часто є безіменними серверами «стійки та стека» з різними постачальниками

Прочитайте інші області порівняння

Висновок

У Legacy Cloud System дизайн має тенденцію до однорідних елементів. У Web Scale хмарі дизайн системи має тенденцію змішувати та збігати елементи



Веб-масштабована хмара — це 20-річний прорив програмного забезпечення платформи «Думай по-іншому».

Хмара – це дійсно релігія нової платформи

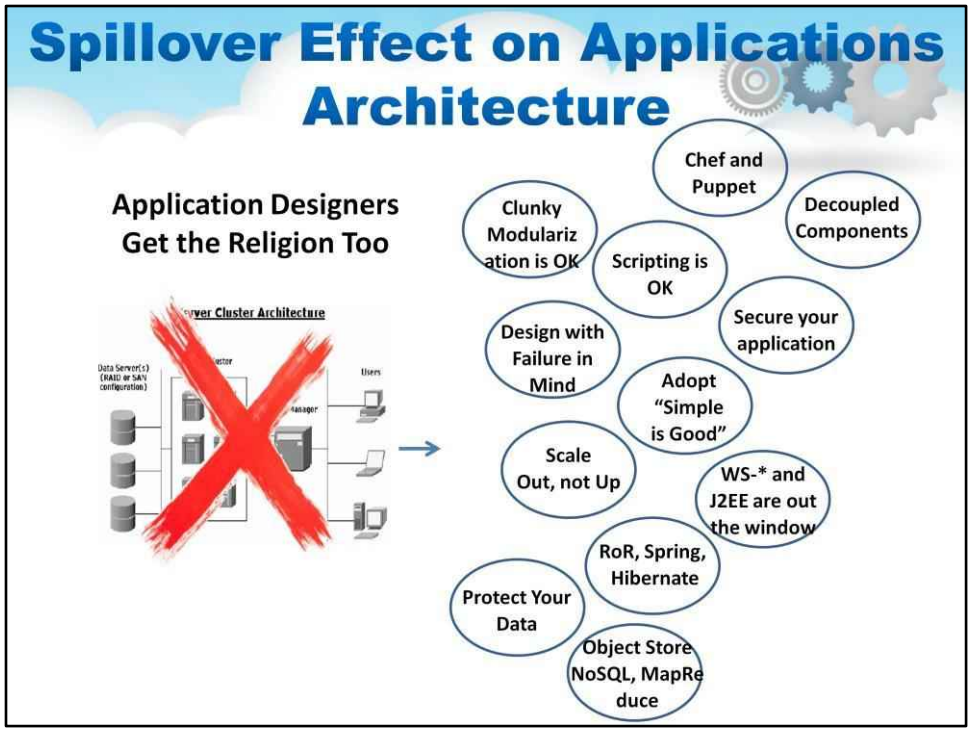
Платформа майже нічого не робить Платформа мало обіцяє

Віртуалізація

автоматизація

Нескінченне, еластичне сховище

Нескінченне, еластичне ЦП



Це має побічний ефект на архітектуру програм

Неможливо взяти всі шаблони проектування, які ви знаєте з корпоративної багаторівневої архітектури додатків або веб-архітектури, хоча вони значною мірою застосовні, багато ключових областей переосмислюються для хмари.

Програми тепер повинні самі піклуватися про: масштабування, а не збільшення, прийняття принципу «Просто – добре», дизайн з урахуванням невдач тощо.

Ключові технологічні прориви, що дозволяють створити хмару

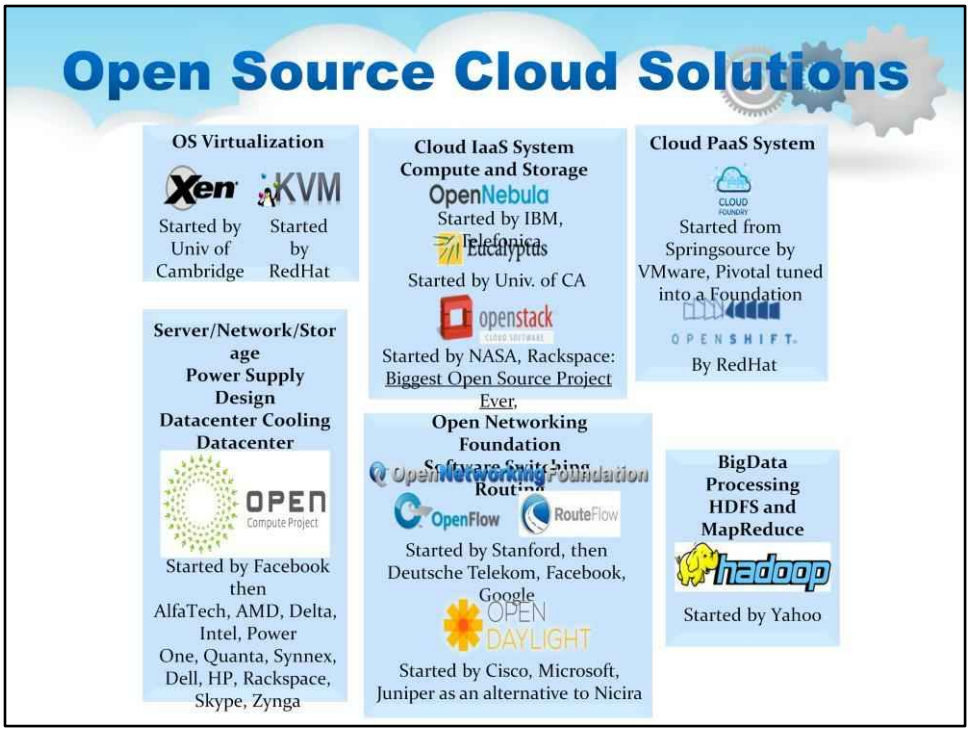
Певні технології допомогли створити ці великі веб-хмари

Як уже згадувалося, товарні сервери, які єназиваються «без надмірностей». що з’являються, зводячи до мінімуму химерні шафи, мерехтливі лампочки та додаткові USB-порти – це невеликі сервери з великим об’ємом пам’яті з прямим підключенням до сховища, ідеальні як хмарні сервери

Додайте до цієї мережі, яку легко налаштувати як конфігурацію «Spine and Leaf». для багатьох комутаторів у верхній частині стійки, які підключаються один до одного або через рівень агрегації, або безпосередньо як хребет.

Потім сам центр обробки даних розвинувся, щоб підтримувати дуже щільні стійки (з високою потужністю) і щільну електроенергію

Звичайно, є багато елементів з відкритим вихідним кодом, які мають вирішальне значення для запуску всього цього. Багато важливих модулів проілюстровано на слайді.



Рішення з відкритим кодом для хмари

Відкритий вихідний код був надзвичайно важливим у технологічному прогресі Cloud. Ця діаграма ілюструє всі сфери, де відкрите кодове джерело мало вирішальне значення. Дивовижно, що в багатьох областях існує більше ніж одна сфера з відкритим кодом, яка вирішує проблему.

З боку гіпервізора ми маємо Xen і KVM як альтернативи з відкритим кодом. Тепер вони є стандартними для дистрибутивів Linux.

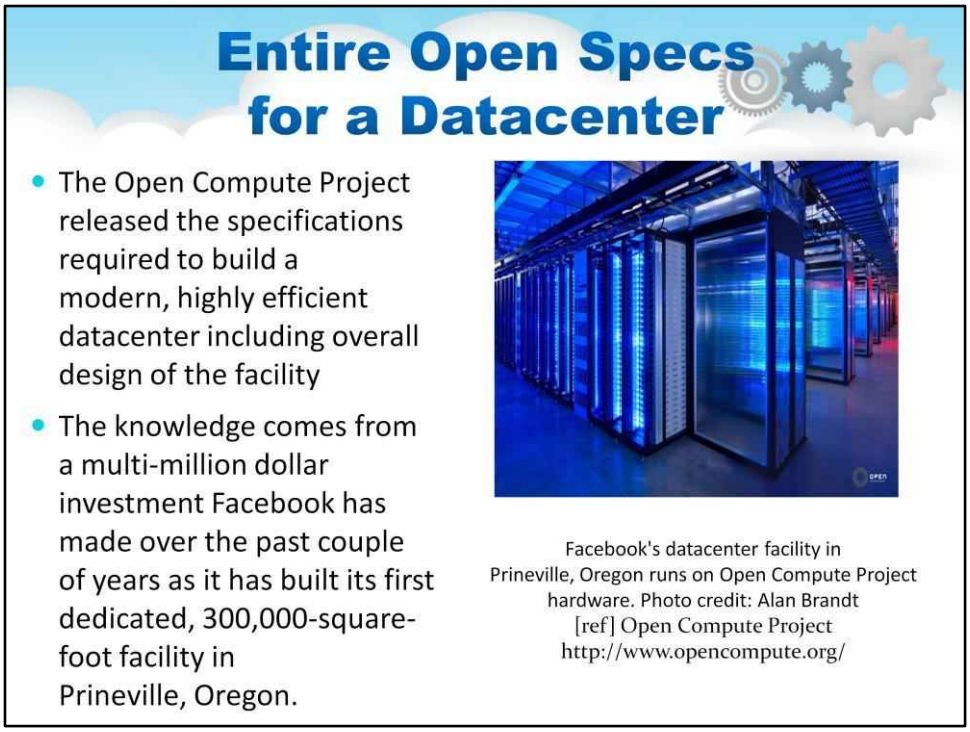
У системі IaaS є оригінальна хмарна система з відкритим кодом з Європи (OpenNebula), а також сам OpenStack, найбільший проект з відкритим кодом в історії людства.

Існують системи PaaS з відкритим кодом як від Pivotal, так і від RedHat.

Існує багато інструментів з відкритим кодом великих даних, починаючи зі спільноти Hadoop і продовжуючи звідти, з багатьма проектами в Apache.

Нарешті ви побачите декілька цікавих ініціатив, одна з яких називається проектом Open Compute, присвячена фізичним частинам побудови центру обробки даних, починаючи від електроживлення, охолодження та електропроводки, закінчуючи чистими серверами та модулями зберігання даних. Ви також побачите дві організації, які розглядають стандарти програмно-визначених мереж (SDN).

Насправді корисно зайти на кожен із цих сайтів і поглянути на кожну ініціативу. Зрозуміло, що відкрите кодове джерело є дуже важливим рушійним фактором у розвитку технології хмарних обчислень.



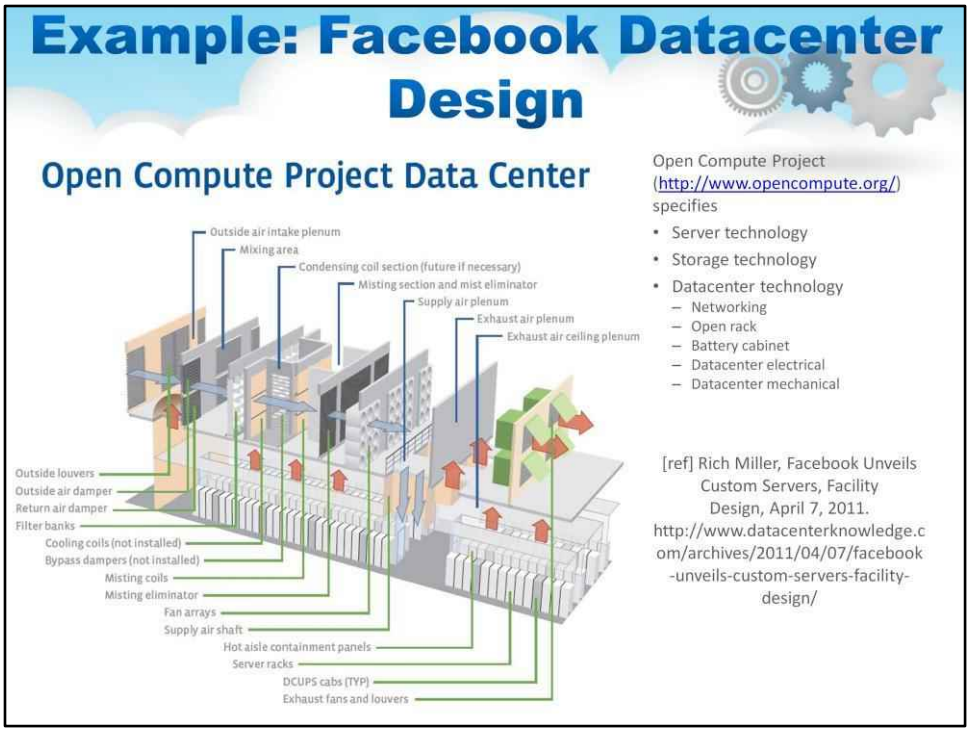
Повні відкриті специфікації для центру обробки даних

Від Open Compute Project (заснованого Facebook у 2011 році), який випустив специфікації, необхідні для створення сучасного, високоефективного центру обробки даних, включаючи загальний дизайн об’єкта

На слайді показано фотографію центру обробки даних Facebook у Прінвіллі, штат Орегон, який працює на обладнанні Open Compute Project.

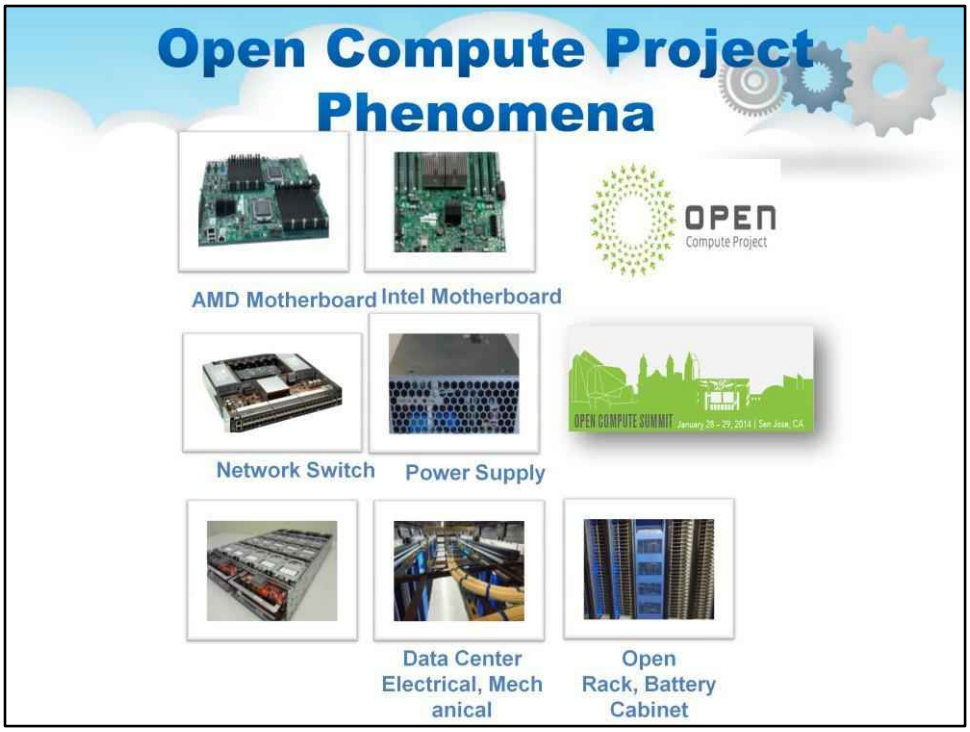
Ці відомості отримані завдяки багатомільйонним інвестиціям, які Facebook зробила за останні пару років, коли побудувала свій перший спеціальний об’єкт площею 300 000 квадратних футів у Прінвіллі, штат Орегон.

З тих пір багато інших компаній долучилися до цих специфікацій.



приклад

Цей слайд містить ілюстрацію, яка демонструє деталі дизайну центру обробки даних Facebook з акцентом на дизайні охолодження. Будь ласка, вивчіть схему.

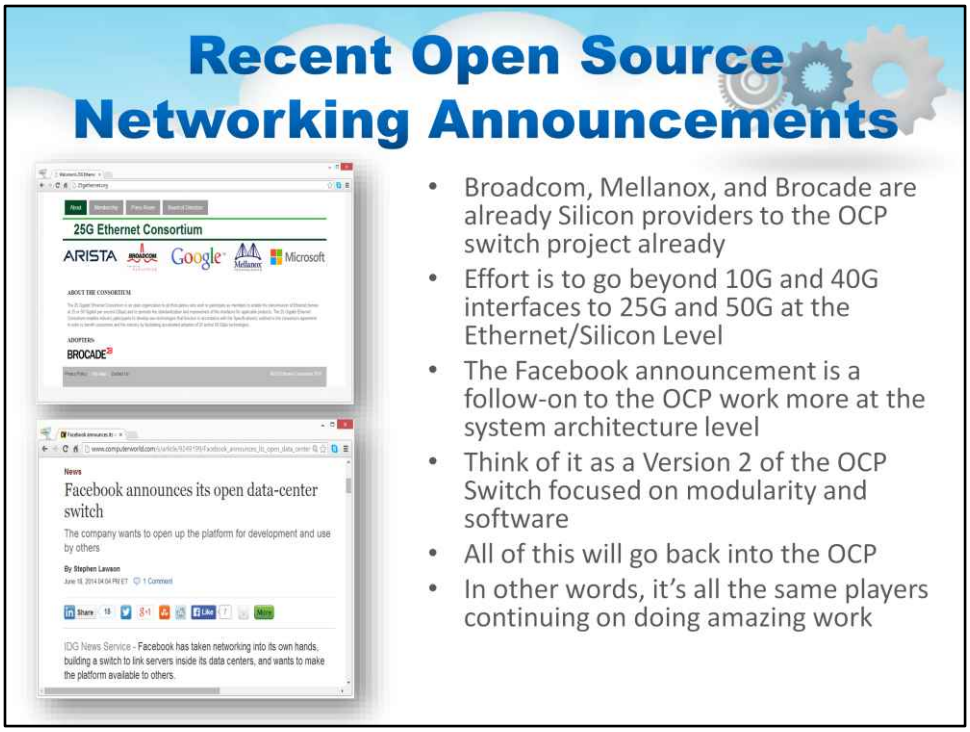


Явища Open Compute Project

У Open Compute Project насправді є багато підпроектів. Слайд містить ілюстрації до основних проектів.

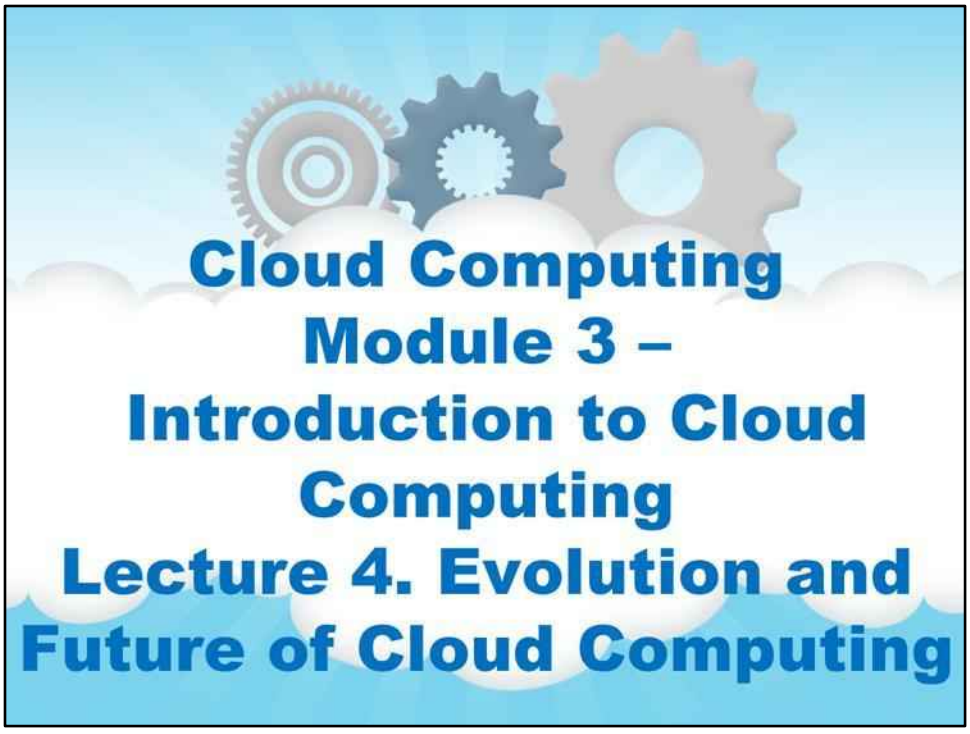
Ви побачите специфікації сервера (материнської плати), специфікації мережевого комутатора, специфікації блоку живлення, специфікації системи зберігання даних і специфікації електричних, механічних і навіть акумуляторних шаф.

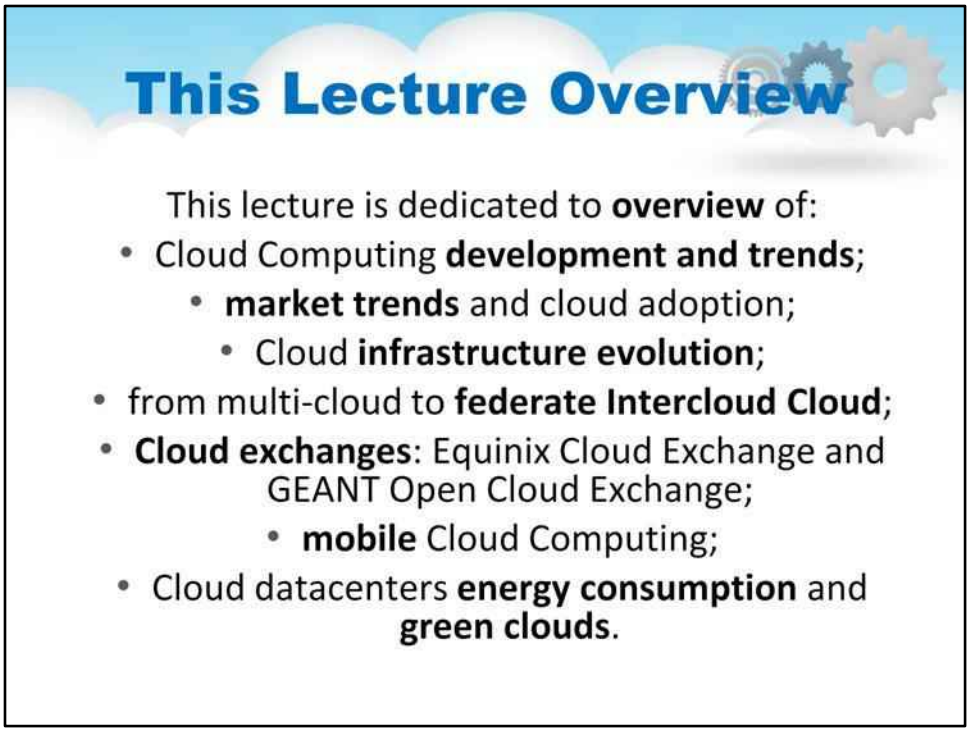
Проект Open Compute маєтепер це власна конференція в Силіконовій долині. Це дуже цікаві явища.



Нещодавно було зроблено оголошення з відкритим кодом щодо комутаторів центрів обробки даних. Спочатку це здавалося заплутаним, особливо тому, що оголошення були зроблені поза проектом Open Compute. При ближчому розгляді ми бачимо, що обидва оголошення, як показано на слайді, насправді були зроблені багатьма з тих самих компаній, які є частиною Open Compute Project.

Все це означає, що ера потреби в пропрієтарному мережевому обладнанні для доступу до величезної продуктивності та масштабованості закінчилася, стандартні рішення стають надзвичайно надійними.

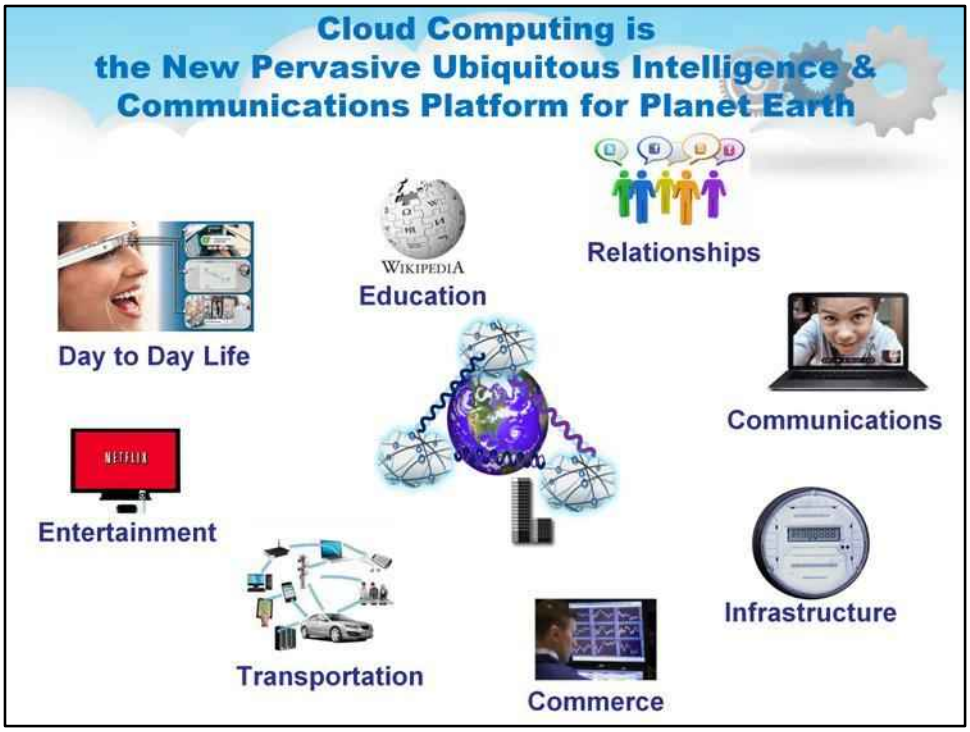

Cloud Computing
Module 3 –
Introduction to Cloud Computing
Lecture 4. Evolution and Future of Cloud Computing



Лекція 4. Еволюція та майбутнє

Хмарні обчислення

Огляд



Хмарні обчислення — це нова повсюдна інтелектуальна та комунікаційна платформа для планети Земля

Освіта

стосунки

Комунікації

Комерція інфраструктури

Інтернету речей

Транспорт

Розваги

День у день життя!

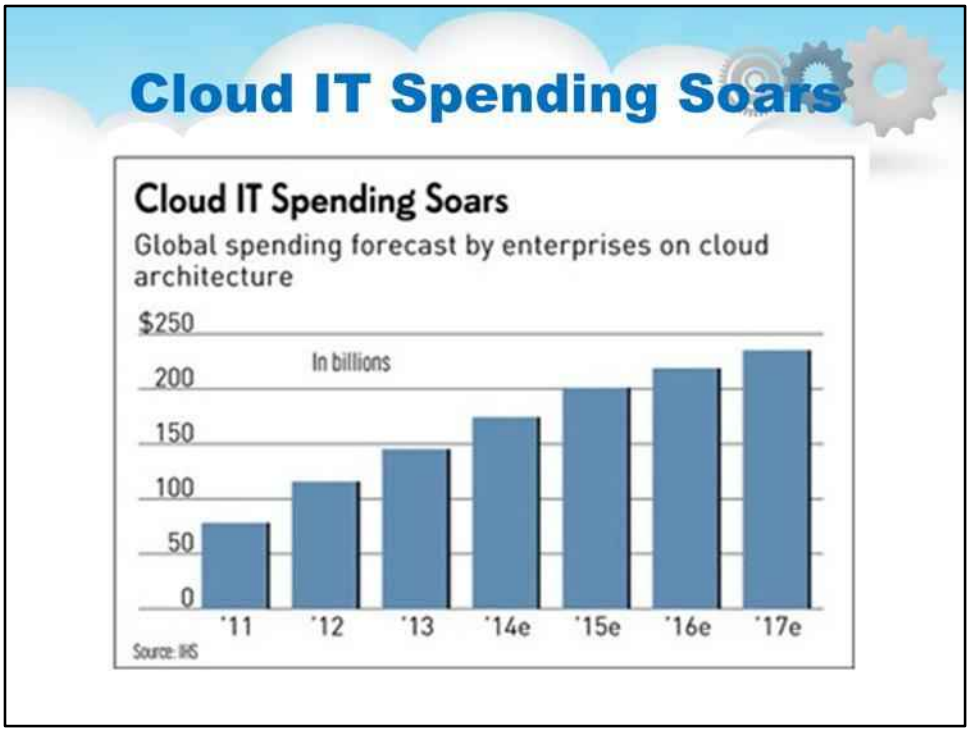
Хмари стають більш всюдисущими, ніж можна було б подумати. Постачальники хмарних послуг налічують від тисяч до понад мільйона серверів у хмарі. За даними datacentermap.com, зараз існують сотні таких хмар, які обслуговують більшу частину цивілізованого населення світу. Вони походять від таких компаній, як Google, Amazon, Microsoft, IBM і HP, а також від місцевого Telecom компаній, у тому числі в країні за вибором

Verizon, AT&T, NTT, Orange/FT, DT, BT тощо. Багато з цих телекомунікаційних компаній матимуть свої хмари, добре інтегровані з мобільними мережами, що полегшить використання можливостей великих і швидких даних для мобільних додатків, а також M2M/IoT.

http://www.datacentermap.com/cloud.html

Нині основні постачальники серверних операційних систем Linux або Windows — це пакети, які просто кажучи, приватні хмари з коробки. Встановіть останню версію весь Windows Server від Microsoft, і ви побачите, що це повна хмара IaaS. Подібним чином дистрибутиви Linux від Red Hat, Canonical або SUSE містять готову до використання версію OpenStack у приватній хмарі.

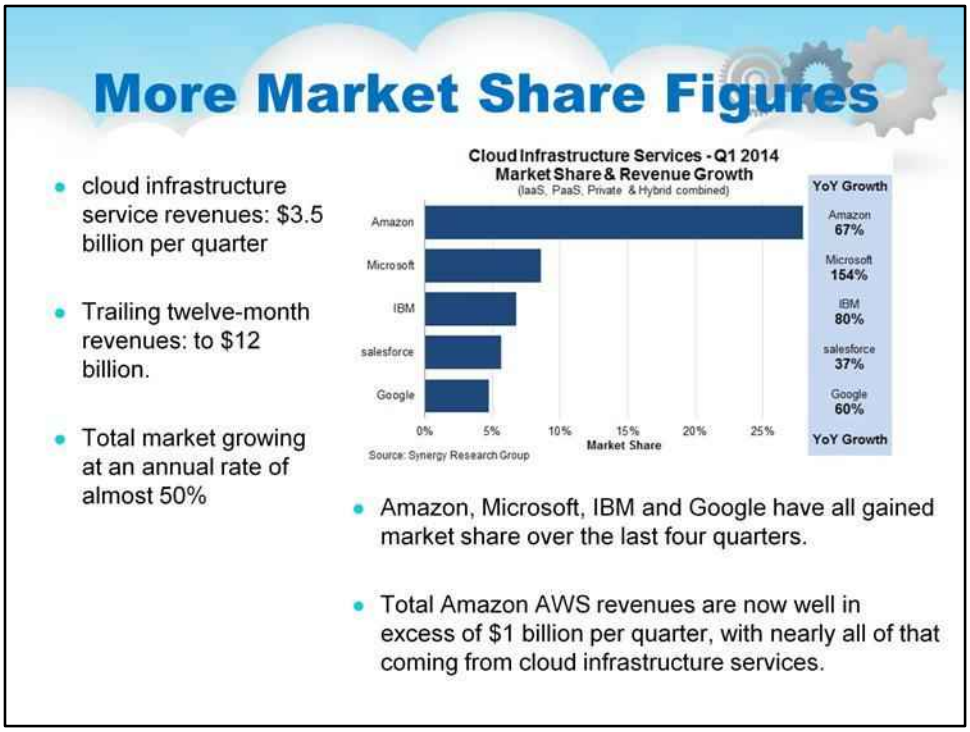
Деякі підприємства хочуть зберегти свої обчислення на приватній інфраструктурі з міркувань безпеки або відповідності, деякі з простих міркувань економії; економічно ефективні хмарні системи під ключ продаються як програмне забезпечення або комбінації апаратного/програмного забезпечення від усіх основних постачальників IT. Хмари можна розташувати якомога ближче до процесів, які вони допомагають оптимізувати, якщо це необхідно за допомогою приватних хмар.



Витрати на хмарні IT стрімко зростають

На слайді показано графік із загальними витратами на Cloud IT Globally

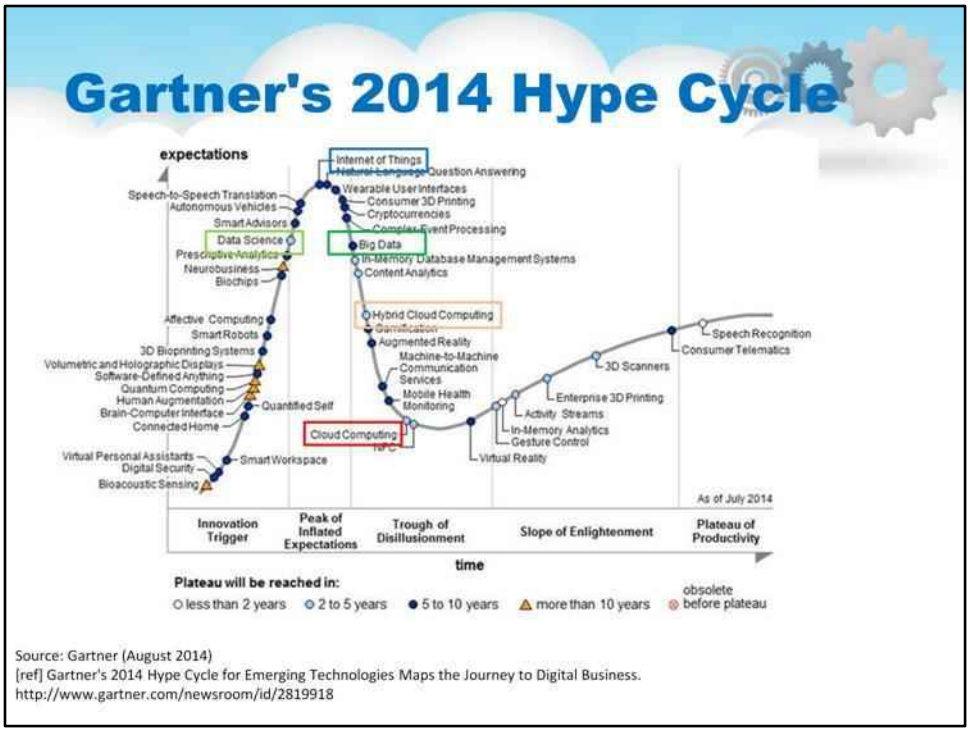
Як можна побачити, світові інвестиції в хмарні обчислення скоро перевищать 200 мільярдів доларів США



Вивчення додаткових показників частки ринку

- За оцінками, щоквартальний дохід від послуг хмарної інфраструктури (включаючи IaaS, PaaS і приватну та гібридну хмару) досяг 3,5 мільярда доларів США
- Дохід за дванадцять місяців склав 12 мільярдів доларів.
- Загальний ринок зростає щорічно майже на 50%
- Amazon, Microsoft, IBM і Google завоювали частку ринку за останні чотири квартали.
- Загальні доходи Amazon AWS зараз значно перевищують 1 мільярд доларів на квартал, причому майже всі вони надходять від сервісів хмарної інфраструктури.

Далі ми розглянемо кілька загальних досліджень тенденцій хмарних обчислень

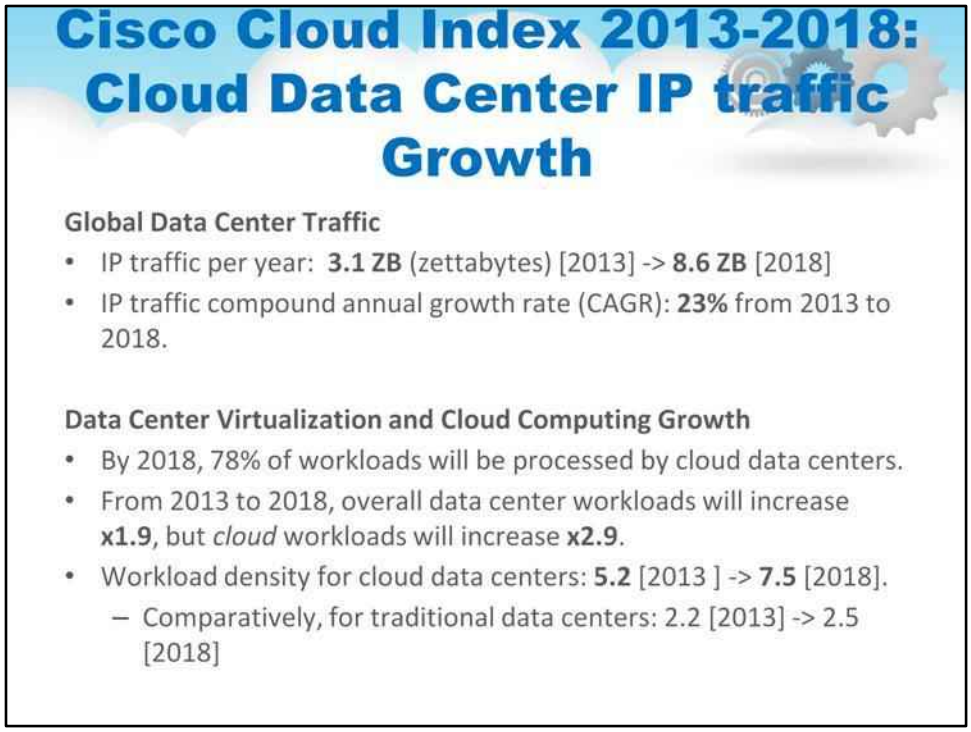


Source: Gartner (August 2014)
[ref] Gartner's 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business. http://www.gartner.com/newsroom/id/2819918

Хмарні обчислення наближаються до сфери продуктивності завдяки масовій реалізації, яка зустрічається з реальністю, а потім гібридним хмарним обчисленням, які адаптують хмари до трансформованого IT-середовища підприємства.

У той же час очевидно, що хмарні обчислення створюють міцну основу для багатьох технологій, які наразі розвиваються, і таких гучних слів, як Big Data, Data Science. Хмарні обчислення також надають нову багатофункціональну платформу для Інтернету речей (іноді його також називають ширшим терміном Інтернет усього).

Наступні слайди та дослідження досліджуватимуть різні числові оцінки різних аспектів розвитку хмарних обчислень.



Cisco Global Cloud Index (GCI) — це постійна спроба прогнозувати зростання глобального центру обробки даних і хмарного IP-трафіку. Прогноз включає тенденції, пов'язані з віртуалізацією центрів обробки даних і хмарними обчисленнями.
У цьому документі представлені деталі дослідження та методологія, що лежить в його основі.

Глобальний трафік центру обробки даних

- До кінця 2018 року річний глобальний IP-трафік центру обробки даних досягне 8,6 зетабайт (715 ексабайт [EB] на місяць), порівняно з 3,1 зетабайт (ZB) на рік (255 EB на місяць) у 2013 році.

- IP-трафік глобального центру обробки даних збільшиться майже втричі (у 2,8 раза) протягом наступних 5 років. Загалом IP-трафік центрів обробки даних з 2013 по 2018 рік зростатиме на 23 відсотки за рік.

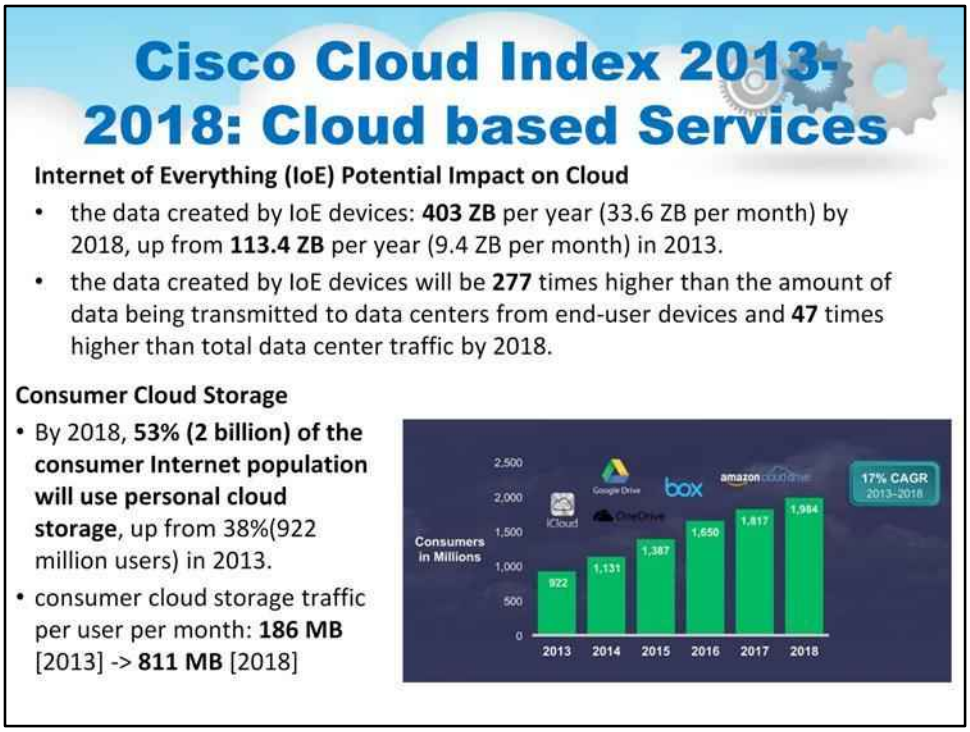
Віртуалізація центру обробки даних і розвиток хмарних обчислень

- До 2018 року понад три чверті (78 відсотків) робочих навантажень оброблятимуться хмарними центрами обробки даних; 22 відсотки оброблятимуть традиційні центри обробки даних.
- Загальне робоче навантаження центру обробки даних зросте майже вдвічі (в 1,9 раза) з 2013 по 2018 рік; однак хмарне робоче навантаження зросте майже втричі (у 2,9 раза) за той самий період.
- Щільність робочого навантаження (тобто навантаження на фізичний сервер) для хмарних центрів обробки даних становила 5,2 у 2013 році та зросте до 7,5 до 2018 року. Для порівняння, для традиційних центрів обробки даних щільність робочого навантаження становила 2,2 у 2013 році та зросте до 2,5 до 2018 року.

Важливо помітити зміни в робочому навантаженні, обробленому хмарою, різними моделями надання хмарних послуг, що вказуватиме на частку фінансового ринку

- До 2018 року 59 відсотків загального обсягу робочих навантажень у хмарі складатиме програмне забезпечення як послуга (SaaS), порівняно з 41 відсотком у 2013 році.
- До 2018 року 28 відсотків від загального обсягу хмарних робочих навантажень складатимуть робочі навантаження інфраструктури як послуги (IaaS), порівняно з 44 відсотками в 2013 році.
- До 2018 року 13 відсотків від загального обсягу хмарних робочих навантажень складатимуть робочі навантаження платформи як послуги (PaaS), порівняно з 15 відсотками у 2013 році.

Ця тенденція насправді відображає типове впровадження хмари та еволюцію на підприємствах: від початкового впровадження IaaS через PaaS і загальну платформу розробки програм до операційних послуг типу SaaS

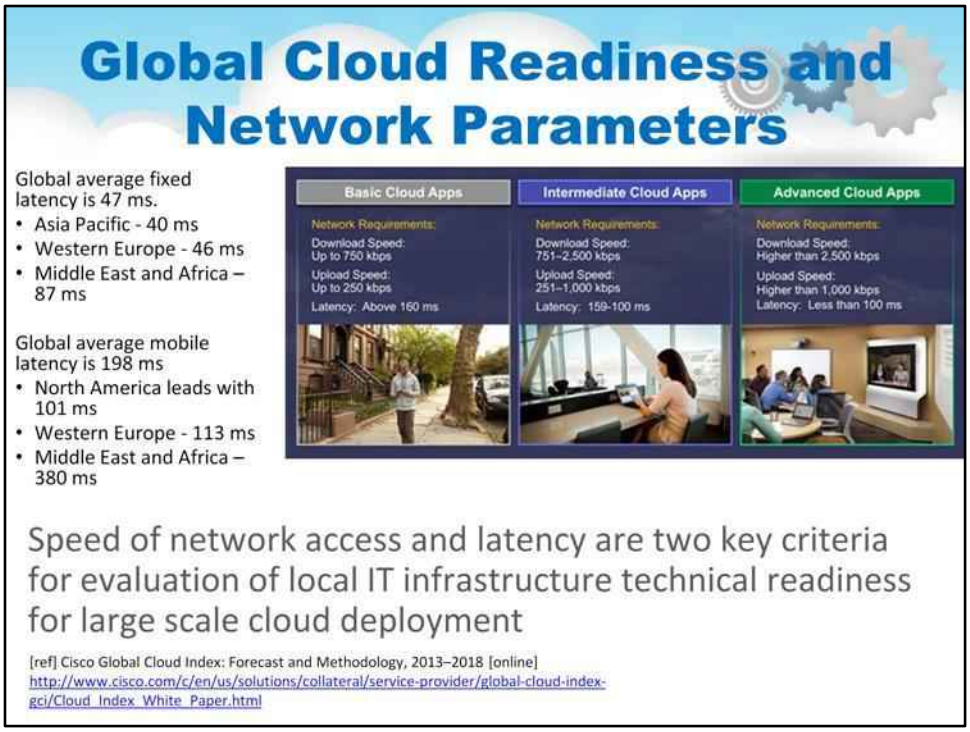


Потенційний вплив Інтернету всього (IoE) на хмару

- У всьому світі обсяг даних, створених пристроями IoE, досягне 403 ZB на рік (33,6 ZB на місяць) до 2018 року порівняно зі 113,4 ZB на рік (9,4 ZB на місяць) у 2013 році.
- У всьому світі дані, створені пристроями IoE, будуть у 277 разів більші, ніж обсяг даних, що передаються в центри обробки даних від пристроїв кінцевих користувачів, і в 47 разів перевищуватимуть загальний трафік центрів обробки даних до 2018 року.

Споживацьке хмарне сховище

- До 2018 року 53 відсотки (2 мільярди) споживачів Інтернету використовуватимуть персональні хмарні сховища, порівняно з 38 відсотками (922 мільйони користувачів) у 2013 році.
- У всьому світі трафік споживчого хмарного сховища на користувача становитиме 811 мегабайт на місяць до 2018 року порівняно зі 186 мегабайтами на місяць у 2013 році.



Затримка мережі має вирішальне значення для роботи хмарних програм, оскільки більшість надійних протоколів зв’язку вимагають зворотного зв’язку (зокрема TCP), а багато хмарних програм є інтерактивними та працюють майже в режимі реального часу (наприклад, розпізнавання голосу на смартфонах, бізнес-додатках або іграх).

Дослідження готовності до хмарних технологій пропонує регіональний погляд на вимоги до широкосмугових і мобільних мереж для надання хмарних послуг наступного покоління

Дослідження також вивчало здатність кожного глобального регіону (Азіатсько-Тихоокеанського регіону, Центральної та Східної Європи, Латинської Америки, Близького Сходу та Африки, Північної Америки та Західної Європи) підтримувати вибірку базового, середнього та розширеного бізнес-хмари та споживчої хмари. програми.

Швидкість доступу до мережі та затримка є двома ключовими критеріями для оцінки технічної готовності локальної IT-інфраструктури до масштабного хмарного розгортання

Готовність до хмарних технологій і їх прийняття в країнах, що розвиваються

Наступний слайд містить ілюстрацію з The Global Cloud Computing Scorecard BSA 2013

У цьому звіті оцінюється готовність країн підтримувати розвиток хмарних обчислень

Включено 24 країни, на які разом припадає 80 відсотків світового ринку інформаційних і комунікаційних технологій (IKT).

Економічне зростання та трансформація як підприємств, так і національних економік від широкого використання хмарних обчислень вимагають належної політики в кожній із семи сфер:

Забезпечення конфіденційності та регулювання захисту даних

Послідовні послуги безпеки та дотримання найкращих практик безпеки.

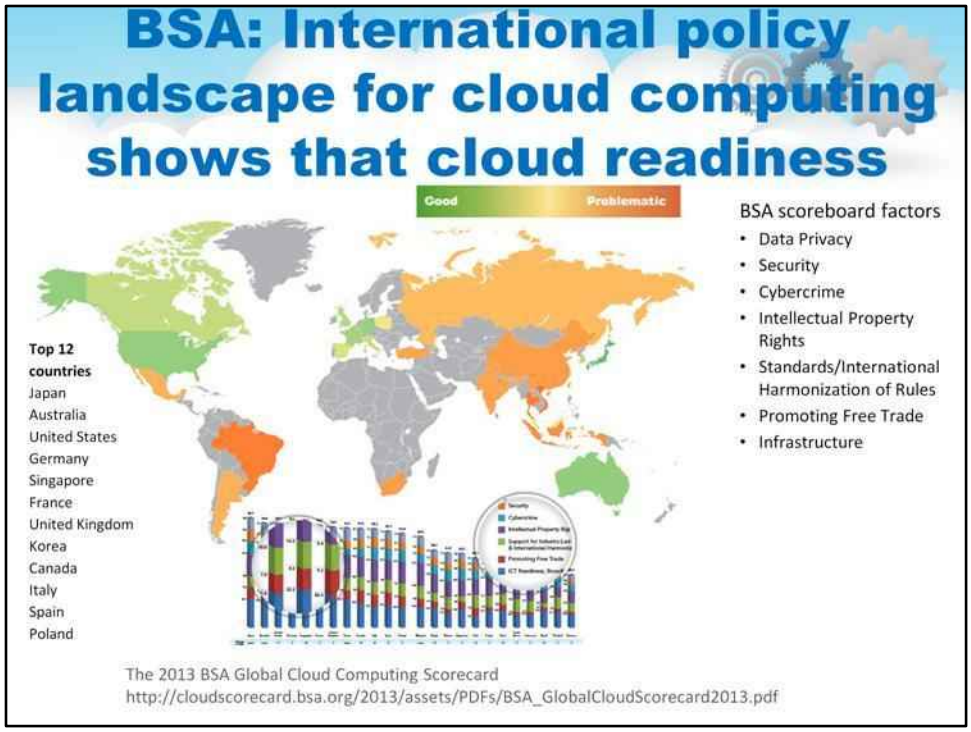
Правова система та заходи захисту від кіберзлочинності

Захист інтелектуальної власності для підтримки інновацій і транскордонного потоку контенту та між хмарними провайдерами

Забезпечення необмеженого або просто регламентованого переміщення даних, гармонізація міжнародних правил

Наявність необхідної IT-інфраструктури: хмарні обчислення вимагають надійного, повсюдного та доступного широкосмугового доступу.

Безпека та конфіденційність



Фактори табло BSA включають

Конфіденційність даних

Хмарні користувачі повністю приймуть і запровадять хмарні обчислення, лише якщо вони впевнені що конфіденційна інформація, яка зберігається в хмарі, будь-де у світі, не буде використана чи розкрита постачальником хмари несподіваним чином.

Безпека

Споживачі хмарних обчислень та інших цифрових послуг (включаючи приватні секторні та державні користувачі) потребують гарантії того, що постачальники хмарних послуг розуміють і належним чином керують ризиками безпеки, пов’язаними зі зберіганням їхніх даних і запуском їхніх програм у хмарних системах

Кіберзлочинність

Оскільки хмарні обчислення передбачають агрегацію величезних обсягів даних великих центрів обробки даних, це створює нові та дуже спокусливі цілі. Оскільки злочинці звертають свою увагу на ці сховища інформації, захистити такі центри обробки даних як від фізичних, так і від кібератак стане дедалі складніше.

Право інтелектуальної власності

Постачальники технологій і послуг хмарних обчислень і цифрової економіки, як с інші надзвичайно інноваційні продукти, покладаються на комбінацію патенти, авторські права, комерційні таємниці та інші форми захисту інтелектуальної власності.

Стандарти/Міжнародна гармонізація правил

Захист даних і конфіденційність у хмарі

Захист даних і конфіденційність залишатимуться головним обмежуючим фактором для зростання хмари та стимулюватимуть наступні дослідження та розробки

Національне та міжнародне законодавство щодо захисту даних, яке має підтримуватися відповідними процедурами сертифікації та оцінки, службами та органами

Нові вдосконалені технології підвищення конфіденційності (PET) повинні захистити конфіденційність користувачів щодо даних і активності, що зберігаються в хмарі

- Поява технологій великих даних вимагатиме нового PET, що запобігатиме (повторній) ідентифікації користувачів на основі кореляції між кількома джерелами даних

Спільна модель безпеки, яка зараз використовується в хмарі, підвищить рівень моніторингу безпеки (і конфіденційності), який контролюється клієнтом

Моделі контролю доступу до хмари продовжуватимуть використовувати федеративний контроль доступу та моделі керування ідентифікацією

- Незважаючи на те, що методи керування доступом базуються на початково перевіреній ідентифікації користувача, вони не повинні розкривати особу користувача постачальникам хмарних послуг і стороннім службам.

Шифрування даних і керування ключами

- Нові моделі обробки зашифрованих даних, такі як гомоморфне шифрування
- Нові моделі управління ключами

Надійність платформи хмарних обчислень і управління довірою

Довірена хмарна платформа включає принаймні два аспекти: надійність платформи та управління довірою

Надійність – це властивість хмарної платформи, яка включає численні фактори, включаючи принципи проектування, впровадження, технічні механізми та методи роботи

Надійність за проектом є ключем до створення надійної хмарної платформи

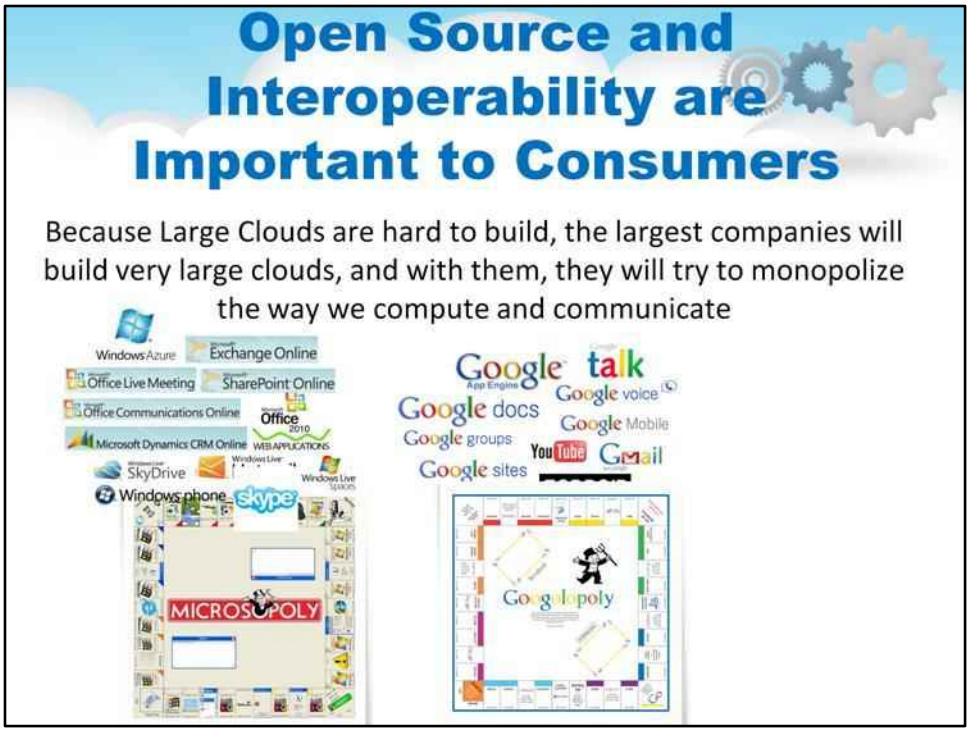
Хмарні постачальники приділяють велику увагу всім аспектам своєї платформи, щоб створити надійне середовище для послуг і даних користувачів

Управління довірою включає як технічні механізми, так і рішення для встановлення та керування довірчими стосунками між постачальником і клієнтом або адміністративними доменами та доменами безпеки.

Технічна довіра ґрунтується на довіреній платформі й зазвичай потребує налаштування вручну

Хмара стимулюватиме подальший розвиток динамічного встановлення довіри та механізмів управління для адаптації до динамічного та на вимогу характеру хмарних послуг

Архітектура довіреної обчислювальної платформи (TCP) від Trusted Computing Group (TCG) забезпечує технічну основу для побудови розподіленого надійного середовища

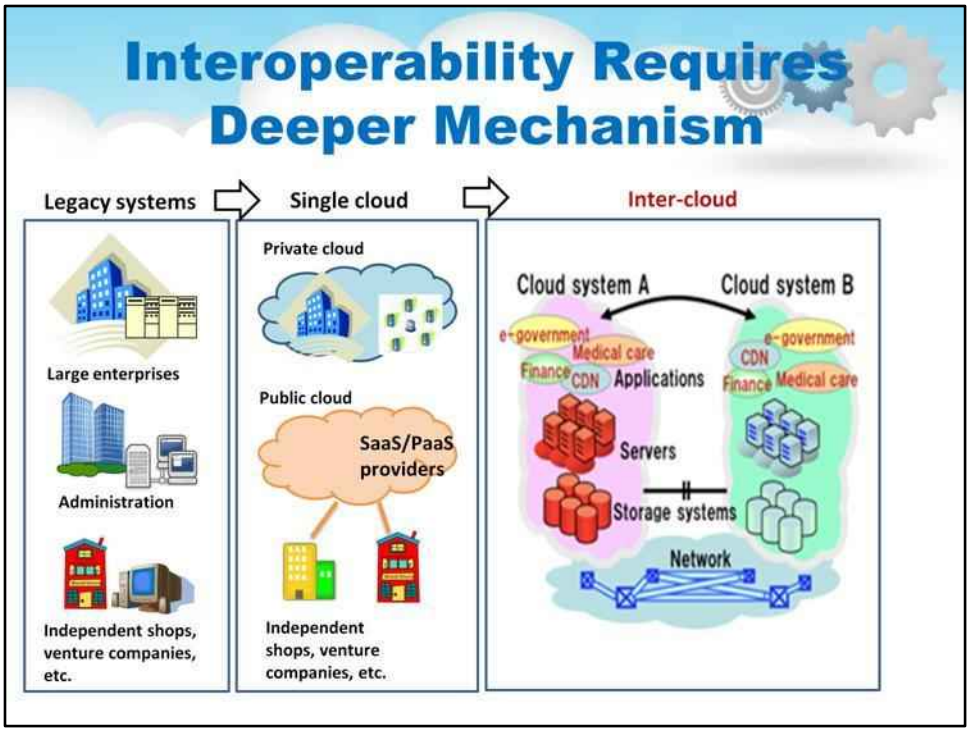


Відкритий код і сумісність важливі для споживачів

Оскільки великі хмари важко побудувати, найбільші компанії створюватимуть дуже великі хмари, і разом з ними вони намагатимуться монополізувати наш спосіб обчислення та спілкування

(? Не впевнений, чи правильно я інтерпретував оригінальний текст)

У часи до появи Інтернету домінували великі компанії. Коли відкритий вихідний код і стандарти взяли верх, Інтернет почав швидко розвиватися. Щось подібне буде і з хмарними технологіями.



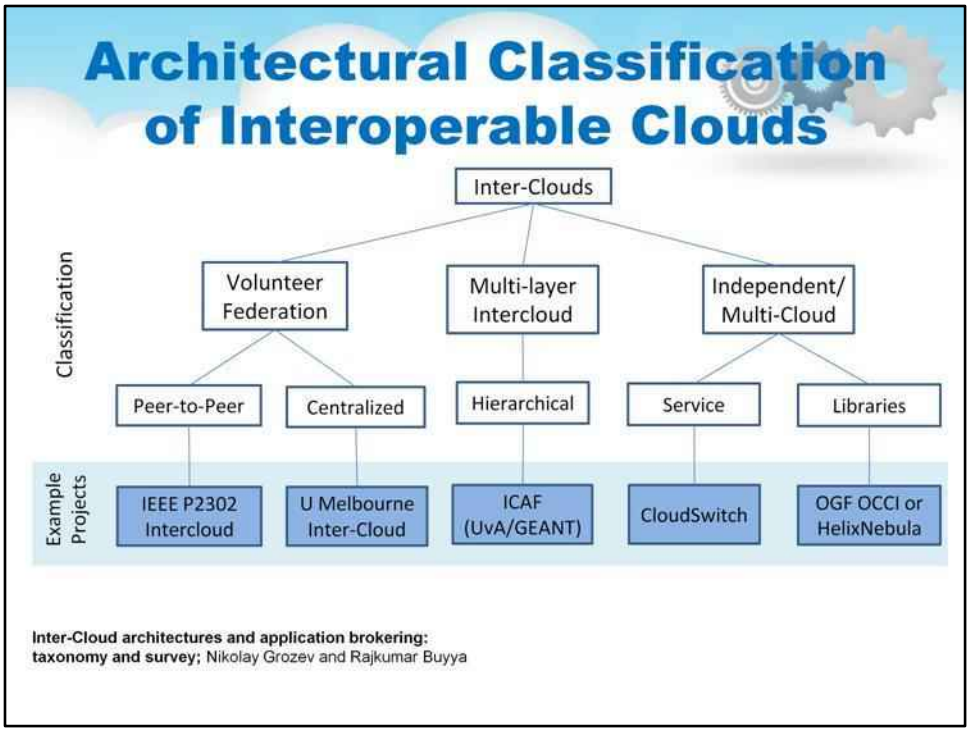
Цей слайд містить ілюстрацію, яка показує основні етапи еволюції IT. Можна побачити, що прикладне програмне забезпечення раніше було тісно пов’язане з набором серверів і сховищ і зазвичай встановлювалося на місці, на сайті компанії.

Невдовзі обчислювальні платформи стали більш гнучкими, з віртуалізацією та автоматизацією, щоб додатки могли спільно використовувати узагальнений кластер/платформу, яку ми тепер називаємо хмарою.

Ілюстрація показує, що за допомогою Intercloud компанії можуть працювати на платформі кількох центрів обробки даних, де хмари кількох постачальників співпрацюють і взаємодіють, утворюючи одну велику розподілену платформу.

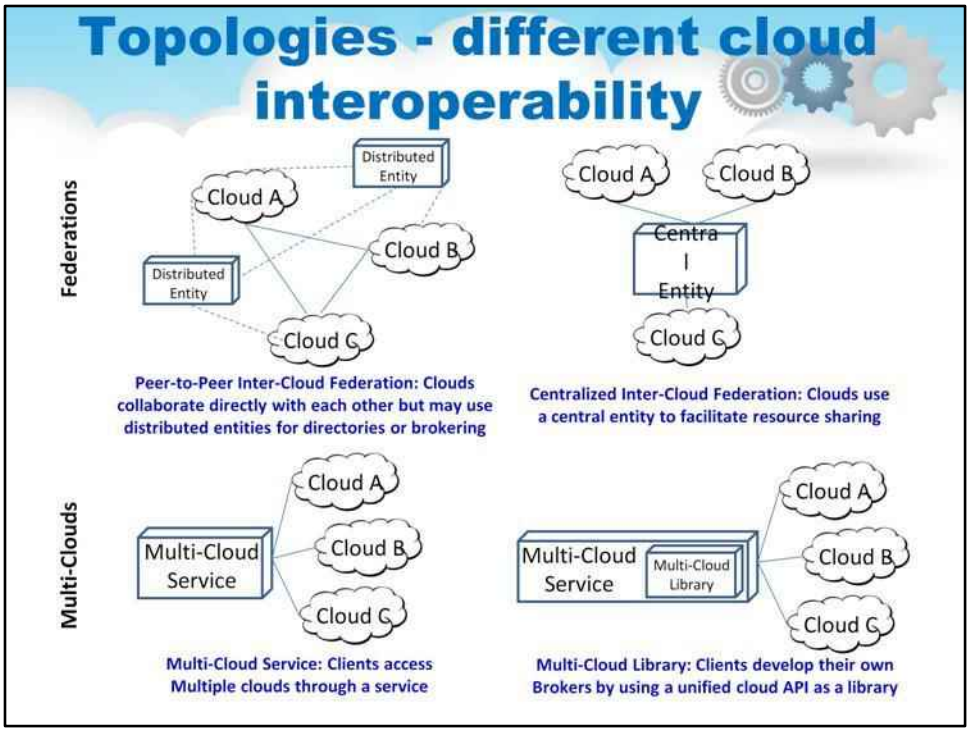
Ця ілюстрація люб’язно надана японською галузевою асоціацією Global Intercloud Technology Forum. Ця група виконала новаторську роботу у визначенні значення та вимог до Intercloud.

Еволюція хмарної інфраструктури



Існує багато способів взаємодії хмар. На цьому слайді використано ілюстрацію дослідницької групи з Університету Мельбурна, яка відшліфувала статтю про таксономію різних підходів до хмарної взаємодії. Будь ласка, вивчіть ілюстрацію.

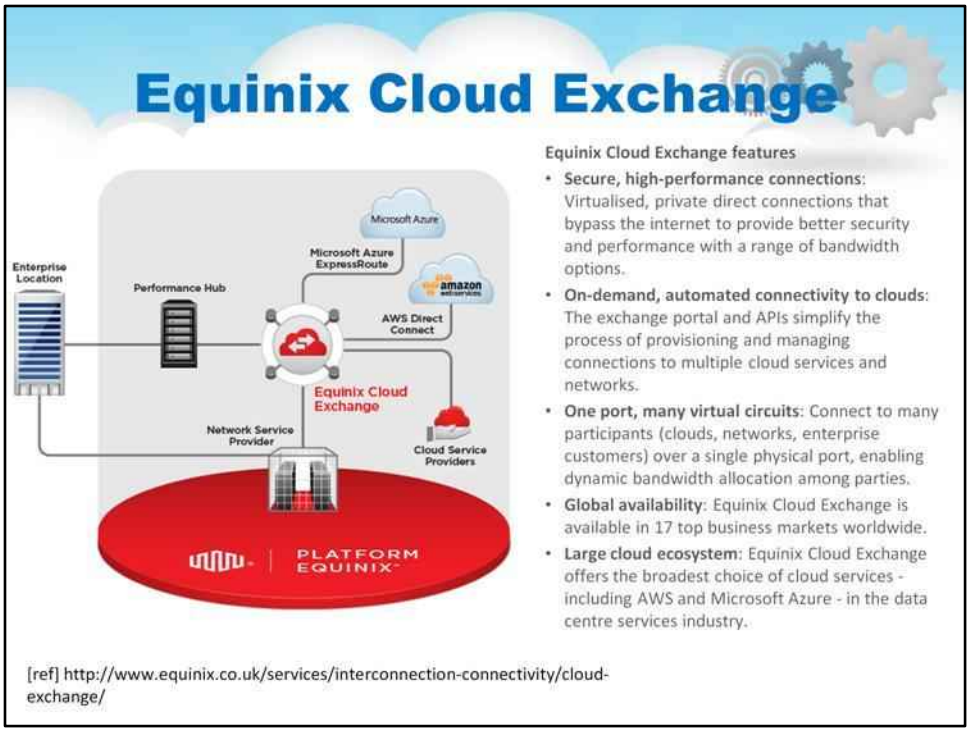
Intercloud Architecture Framework (ICAF) базується на багаторівневій моделі хмарних послуг (CMS) і розділяє контроль, управління, об’єднання та операційні аспекти в Intercloud. Розроблено компанією UvA, на даний момент реалізовано в європейському проекті GEANT.



Легше зрозуміти ці різні підходи до хмарної сумісності, якщо візуалізувати отримані топології. Ілюстрація на цьому слайді показує ті самі чотири категорії, що й на попередньому слайді, але у вигляді топології.

Легко побачити, що багатохмарний підхід буде працювати для користувача, який хоче отримати доступ до а невелику кількість хмар більш-менш вручну і потрібно «з'єднати». ресурси з кожної хмари. Цей підхід ідеально підходить для науковців, які хочуть отримати доступ до простих хмарних ресурсів (наприклад, віртуальних машин) для створення великих кластерів для Hadoop або Grid-обчислень. У цьому випадку відповідальність за розуміння варіацій між хмарами лягає на користувача.

Легко побачити, що підхід Федерації є набагато більш масштабованою та взаємопов'язаною архітектурою. Цей підхід ідеально підходить для систем, які можуть охоплювати кілька хмарних провайдерів або охоплювати приватні та публічні хмари. Це також покладає відповідальність за розуміння відмінностей між хмарами на інфраструктуру, а не на кінцевого користувача. Це важливий момент, який ми розглянемо пізніше.



Глобальна хмарна інфраструктура продовжує розвиватися та створює власну глобальну хмарну екосистему, яка включає глобальні інфраструктури хмарних провайдерів і точки обміну, подібні до поточної Інтернет-інфраструктури

Працює з 2014 року та має значне зростання

Equinix Cloud Exchange — це розширене рішення для взаємозв’язку, яке забезпечує безперебійне приватне підключення за запитом до багатьох хмар і багатьох мереж у великих міських районах по всьому світу. Це робить процес підключення до хмар таким же швидким і гнучким, як і сама хмара. Ось чому Equinix Cloud Exchange є найбезпечнішим, масштабованим і широко поширеним рішенням для підключення до хмари для центрів обробки даних.

Функції Equinix Cloud Exchange

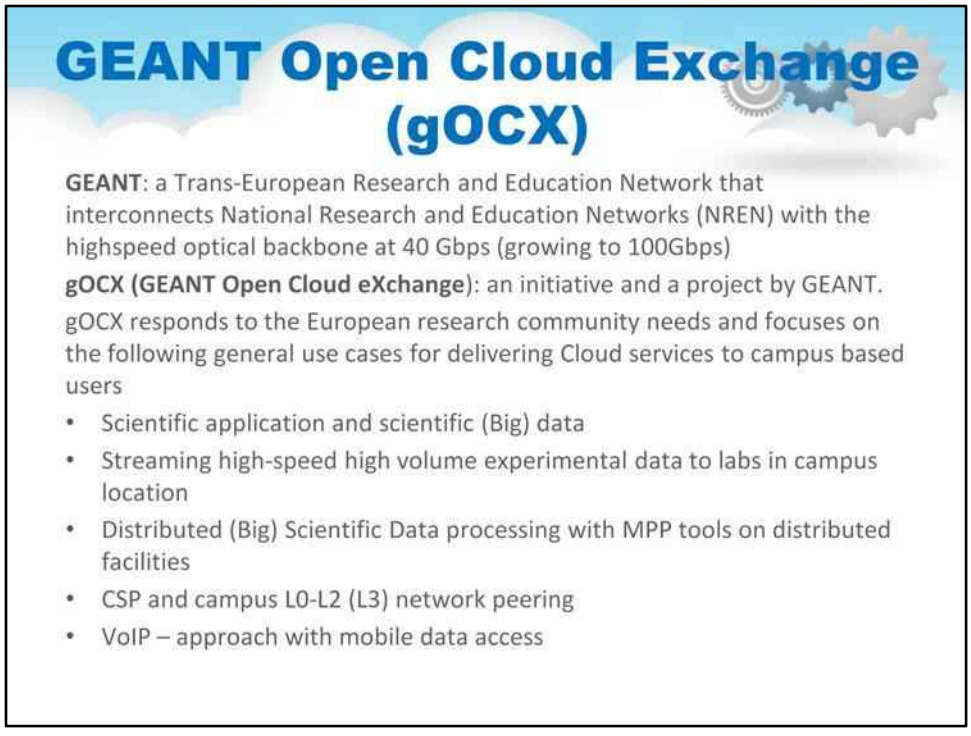
Безпечні високопродуктивні з’єднання: віртуалізовані приватні прямі з’єднання обійти Інтернет, щоб забезпечити кращу безпеку та продуктивність за допомогою низки параметрів пропускної здатності.

Автоматизоване підключення до хмар на вимогу: портал обміну та API спростіть процес надання та керування підключеннями до кількох хмарних служб і мереж.

Один порт, багато віртуальних каналів: підключення до багатьох учасників (хмари, мережі, корпоративні клієнти) через один фізичний порт, що забезпечує динамічний розподіл пропускної здатності між сторонами.

Глобальна доступність: Equinix Cloud Exchange доступний на 17 провідних бізнес-ринках світової.

Велика хмарна екосистема: Equinix Cloud Exchange пропонує найширший вибір хмар послуги, включаючи AWS і Microsoft Azure, в галузі послуг центрів обробки даних.



GEANT Open Cloud Exchange (gOCX) — це проект взаємодії для науки. GEANT — це транс'європейська високошвидкісна мережа, що об'єднує європейські університети та дослідницькі організації (через національні дослідницькі та освітні мережі (NREN)) із поточною швидкістю 10 Гбіт/с, яка в найближчі кілька років зросте до 40 і 100 Гбіт/с.

gOCX відповідає на потреби європейської дослідницької спільноти та зосереджується на наступних загальних випадках використання для надання хмарних послуг користувачам у кампусах:

Наукове застосування та наукові (великі) дані

- LHC/HEP, геноміка, астрономія, клімат, відео тощо (+ наука про довгий хвіст)

Високошвидкісна передача експериментальних даних великого обсягу в лабораторії в кампусі

- Прямі посилання через мережу кампусу

Обробка розподілених (великих) наукових даних за допомогою інструментів MPP на розподілених об'єктах

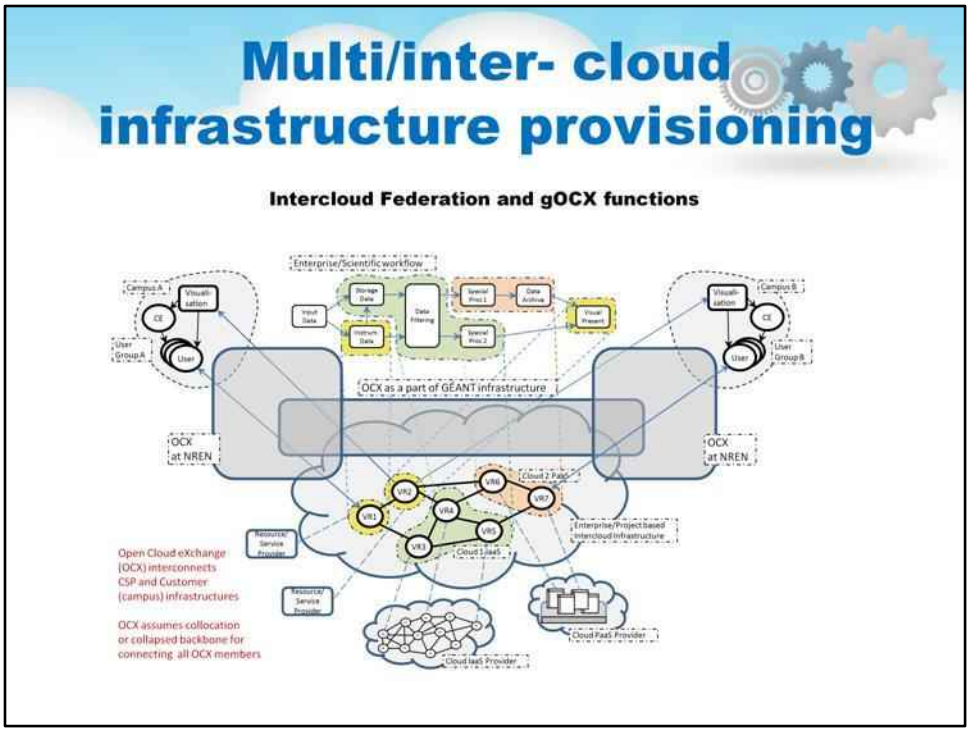
- Дані розподіляються між кількома місцями поруч із локальними центрами обробки даних

Піринг мережі CSP і кампусу L0-L2 (L3).

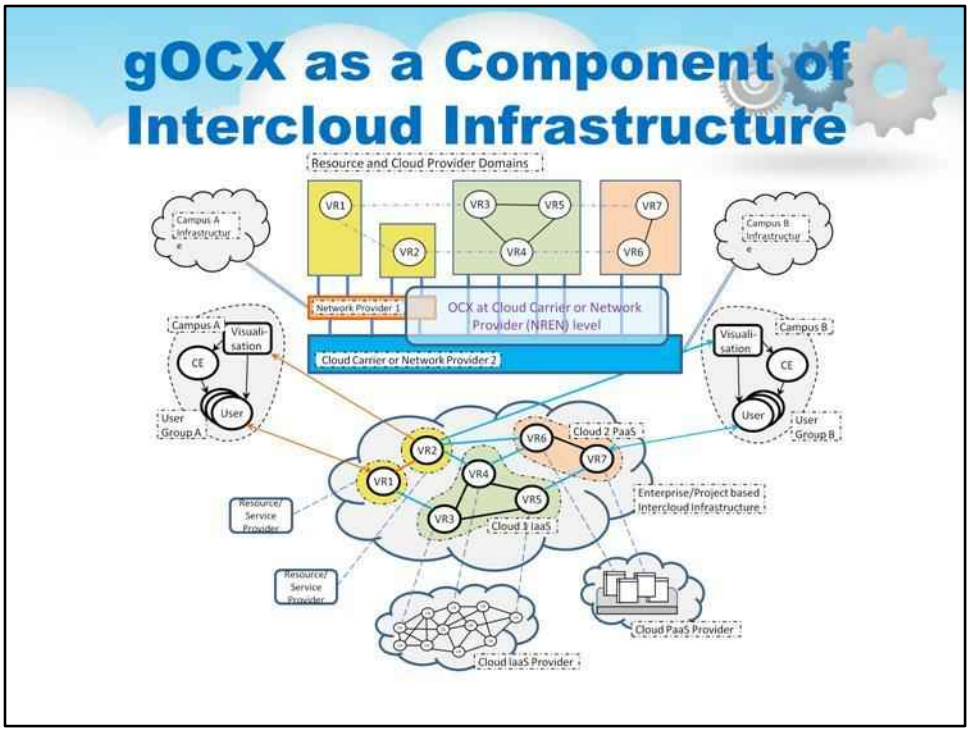
- Темне волокно з завершенням як кампусної мережі або мережі CSP

VoIP – підхід із мобільним доступом до даних

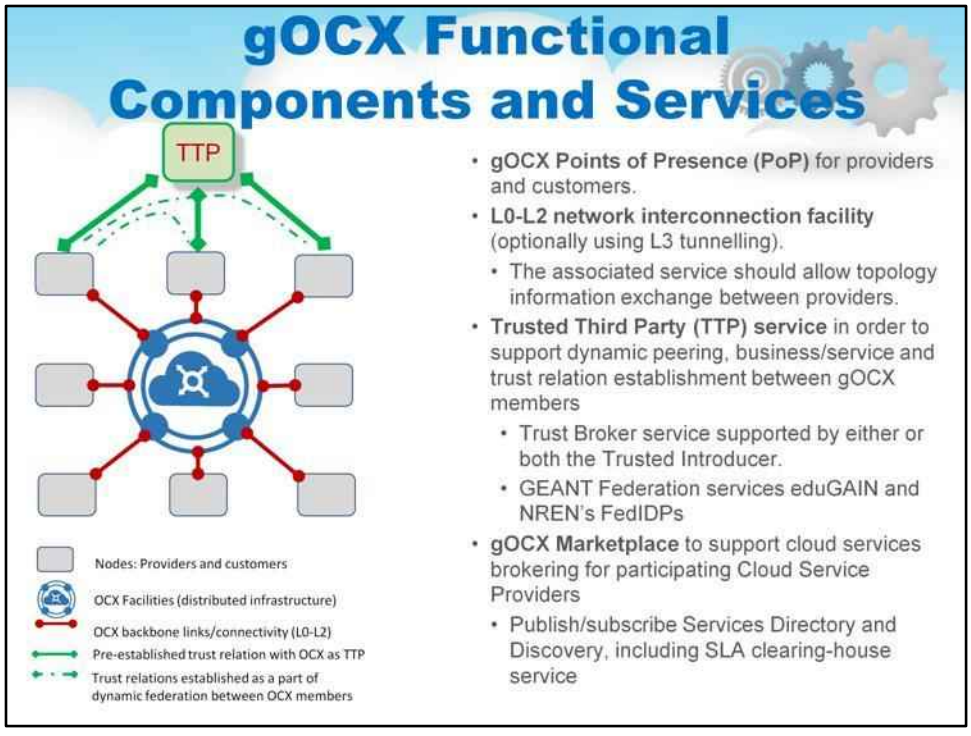
- Підтримка мережі мобільного доступу (LTE) і тунельного доступу до мережі кампусу



OCX може створити необхідний міст між користувачами та постачальниками хмарних послуг.



Слайд ілюструє систему gOCX



Архітектурно та функціонально gOCX включає такі служби та функціональні компоненти

Архітектурно та функціонально gOCX включає такі служби та функціональні компоненти

Точки присутності gOCX (PoP) для постачальників і клієнтів.

Можливість з'єднання між мережами L0-L2 (за бажанням також підключення за допомогою виділених оптичних каналів і тунелювання L3).

Служба довіреної третьої сторони (TTP) для підтримки динамічних
піринг, бізнес/послуги та встановлення довірчих відносин між членами gOCX

gOCX Marketplace для підтримки посередництва хмарних послуг для постачальників хмарних послуг-учасників

Мобільні хмарні обчислення

Мобільні хмарні обчислення (MCC) — ще одна тенденція, яка використовуватиме та стимулюватиме розвиток CC.

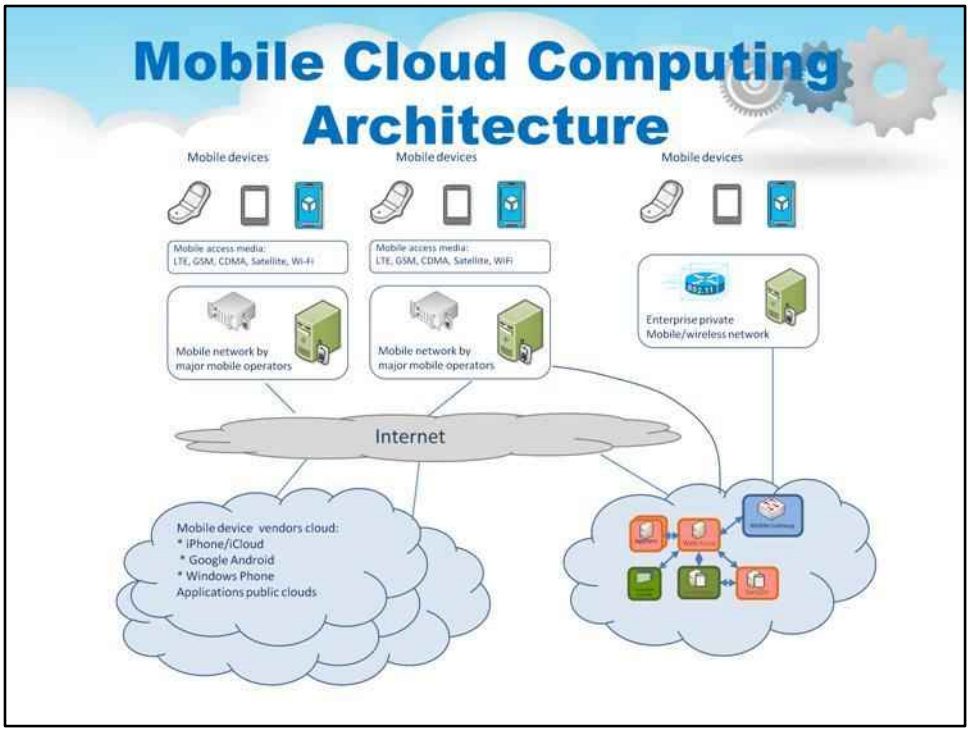
Смартфони, планшети та хмарні обчислення об’єднуються в нову багатовимірну технологічну область Mobile Cloud Computing (MCC). MCC може бути визначається як «розширена мобільна обчислювальна технологія, яка використовує уніфіковану еластичність ресурси різноманітних хмар і мережевих технологій для необмеженої функціональності, зберігання та мобільності для обслуговування безлічі мобільних пристроїв у будь-якому місці та в будь-який час через канал Ethernet або Інтернет, незалежно від гетерогенних середовищ і платформ на основі оплати за потреби.принцип використання».

Зростання використання мобільних пристроїв, їх зростаюча складність і функціональність роблять мобільні пристрої не просто мобільними клієнтами або терміналами для надання доступу до віддалених ресурсів і обчислювальної потужності, а й частиною екосистеми MCC, яка динамічно розподіляє й оптимізує робоче навантаження для вирішення таких загальних проблем мобільних пристроїв як обмежена обчислювальна здатність, обмежена пам’ять і сховище, безпека та конфіденційність.

Мобільні пристрої можуть відігравати багато ролей у нашому мережевому та комп’ютерному/хмарному світі, зокрема,

* Смартфони та планшети все частіше використовуються як мобільні клієнтські пристрої для доступу до локальних і хмарних служб із розширенням функціональності графічного інтерфейсу користувача (GUI). Хмарні програми SaaS надають спеціальний профіль для мобільних пристроїв.

* Смартфони та планшети можна використовувати як мобільну дистанційну візуалізацію



Слайд-шоу загальної архітектури МСС, включаючи мережу оператора мобільного зв’язку, хмару виробника мобільного пристрою/ОС, загальнодоступну хмару та корпоративний мобільний хмарний шлюз

ілюструє загальну архітектуру МСС, яка включає мережу оператора мобільного зв’язку, хмару постачальника мобільного пристрою/ОС, загальнодоступну хмару та корпоративну приватну хмару, яка може містити спеціальний шлюз до мереж операторів мобільного зв’язку для забезпечення оптимізованого доступу для мобільних пристроїв пристроїв. Важливо зазначити, що великі оператори мобільного зв’язку, а також постачальники мобільних пристроїв або мобільних ОС створюють власні хмари, які спеціально призначені для підтримки мобільних програм і послуг. Ви можете знайти багато таких інтелектуальних програм у свого провайдера або постачальника пристрою, які, очевидно, використовують хмарну обробку інтенсивних обчислювальних завдань, таких як побудова маршруту в програмах навігації (ці Додатки не працюють без підключення до мережі передачі даних отже, на віддалений сервер).

Підтримка мобільних додатків із серверними хмарами вимагає розподіленої та глобальної хмарної інфраструктури. Відстеження переміщення пристрою з деякими безперервними процесами вимагатиме міграції програми або віртуальної машини між центрами обробки даних. Така міграція потребує врахування контексту та політики розташування, що викликає низку технічних та дослідницьких питань.

Споживання енергії та зелені хмари

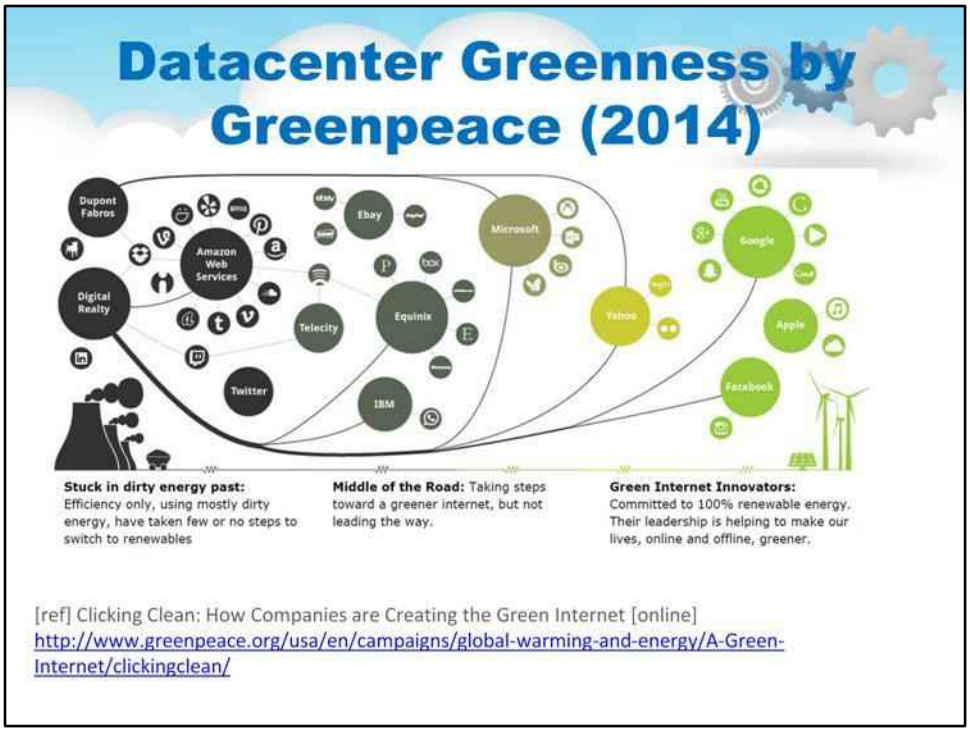
Зелені хмари та енергоспоживання центру обробки даних

Споживання енергії центром обробки даних стає глобальною проблемою, яка прискорюється з переміщенням все більшого навантаження в хмару

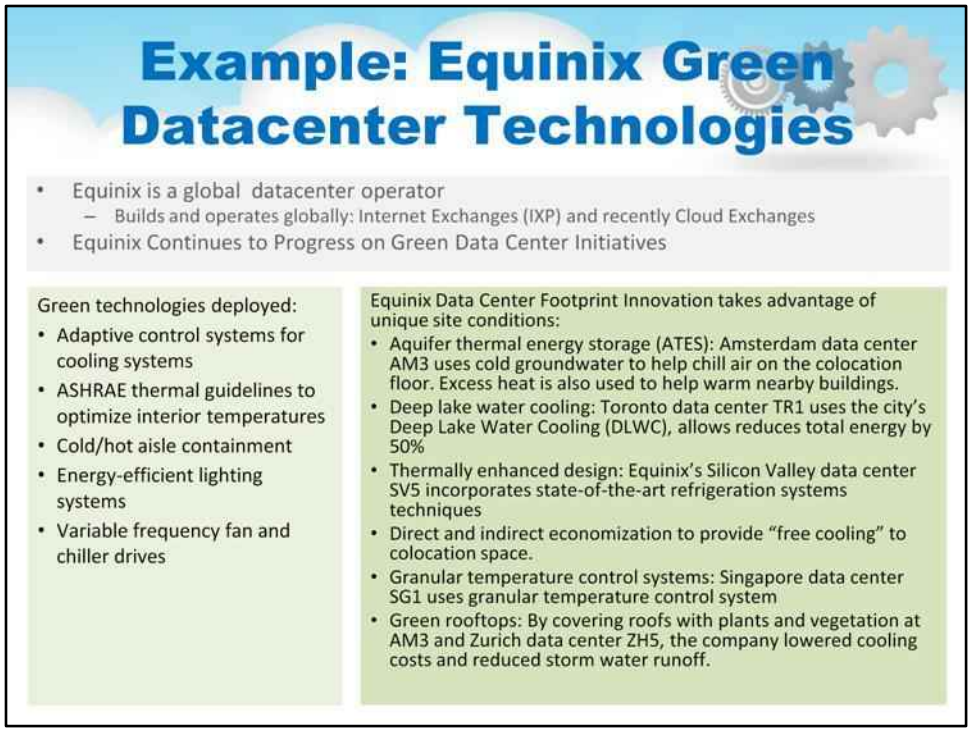
Низка ініціатив урядових, громадських і стандартизаційних органів зосереджена на оптимізації споживання енергії центрами обробки даних та Інтернетом загалом

Технічний комітет із зелених комунікацій та обчислень (TCGCC), IEEE Communications Society

Грінпіс про Зелений Інтернет



Слайд містить ілюстрацію Greenpeace Datacenter Greenness (2014)



Екологічні технології, впроваджені Equinix у всьому світі, включають:

Адаптивні системи керування, які зменшують енергоспоживання та збільшують потужність охолодження завдяки активному управлінню повітряним потоком за допомогою інтелектуальних розподілених датчиків та інноваційної політики керування.

Теплові рекомендації ASHRAE використовуються як еталон у наших найновіших приміщеннях для оптимізації внутрішньої температури. Це зменшує енергоспоживання на охолодження, зберігаючи при цьому безпечну робочу температуру для комп'ютерного обладнання.

Огородження холодного/гарячого проходу використовує фізичні бар'єри, щоб зменшити змішування холодного повітря в припливних проходах центру обробки даних із гарячим повітрям у витяжних проходах. Це призводить до меншого споживання енергії та більш ефективного охолодження.

Енергоефективні системи освітлення в наших центрах обробки даних використовують елементи керування, що активуються рухом, щоб зменшити споживання енергії та нагрівання навколишнього середовища від робочого освітлення.

Приводи змінної частоти використовуються в холодильних машинах, насосах і вентиляторах наших систем опалення, вентиляції та кондиціонування системи для економії енергії шляхом автоматичного зменшення швидкості двигуна та споживаної потужності щоб відповідати меншим навантаженням системи. Інновації в центрі обробки даних

Коли Equinix проектує та будує нові центри обробки даних, вона зменшує енергію, використовуючи унікальні умови на місці:

Охолодження води Deep Lake: центр обробки даних Equinix у Торонто використовує міське Deep Lake Водяне охолодження (DLWC). Цей новий підхід зменшує загальну потребу в енергії на 50 відсотків або більше.

У цій лекції ми розглянули:

- Хмара – це сфера, яка швидко розвивається, вперше з'явившись у 2008 році, і зараз (з 2015 року) перебуває в стані зрілості
- Хмарні обчислення стимулюють трансформацію багатьох технологій і забезпечують основу для нових технологій, таких як Big Data
- І самі великі дані стимулюють розробку хмарних технологій відповідно до обсягу та швидкості обробки даних
- Зміна моделі хмарних послуг від IaaS до SaaS свідчить про те, що з'являються нові можливості для розвитку бізнесу
- Хмара може взяти на себе всі функції маршрутизації управління інфраструктурою та дозволяє новаторам зосередитися на ключових і розумних ідеях
- Мобільні хмарні обчислення, які використовують все більшу кількість персональних кишенькових інтелектуальних пристроїв, використовують глобальну хмарну інфраструктуру та впливатимуть як на інфраструктуру, так і на розвиток програм
- Хмарна індустрія наполегливо працює над зменшенням споживання хмарних центрів обробки даних впровадження передових «зелених» технологій
- У хмарних обчисленнях і великих даних є чудова можливість побудувати кар'єру, і цей набір матеріалів має на меті забезпечити достатній обсяг знань від базового до просунутого рівня.

Хмарні обчислення

Лекційний посібник

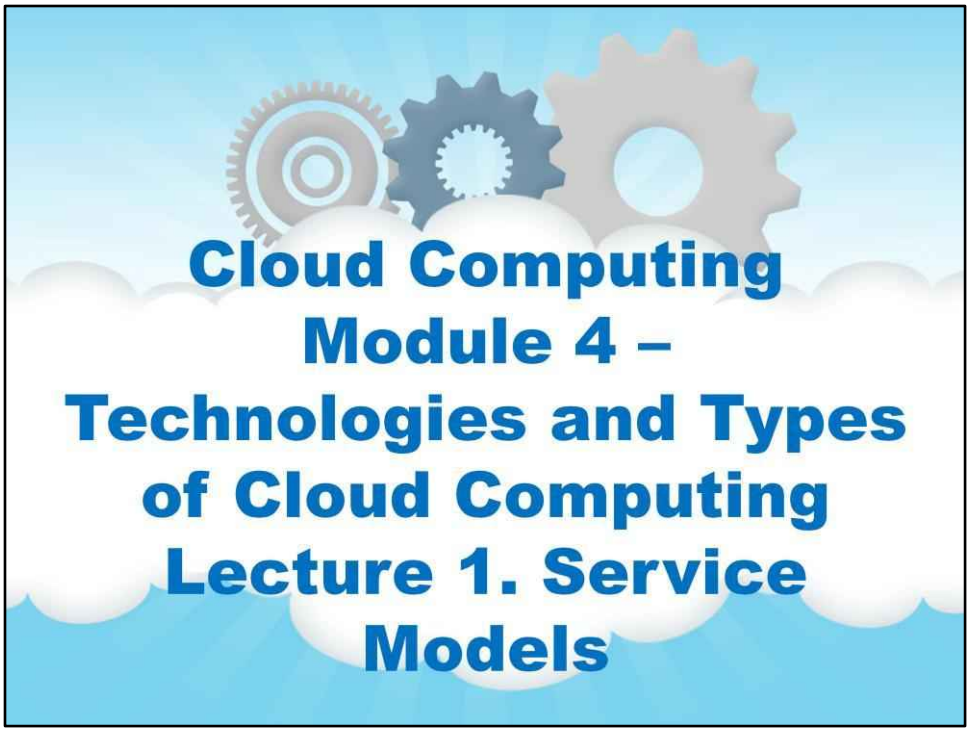
Том 4

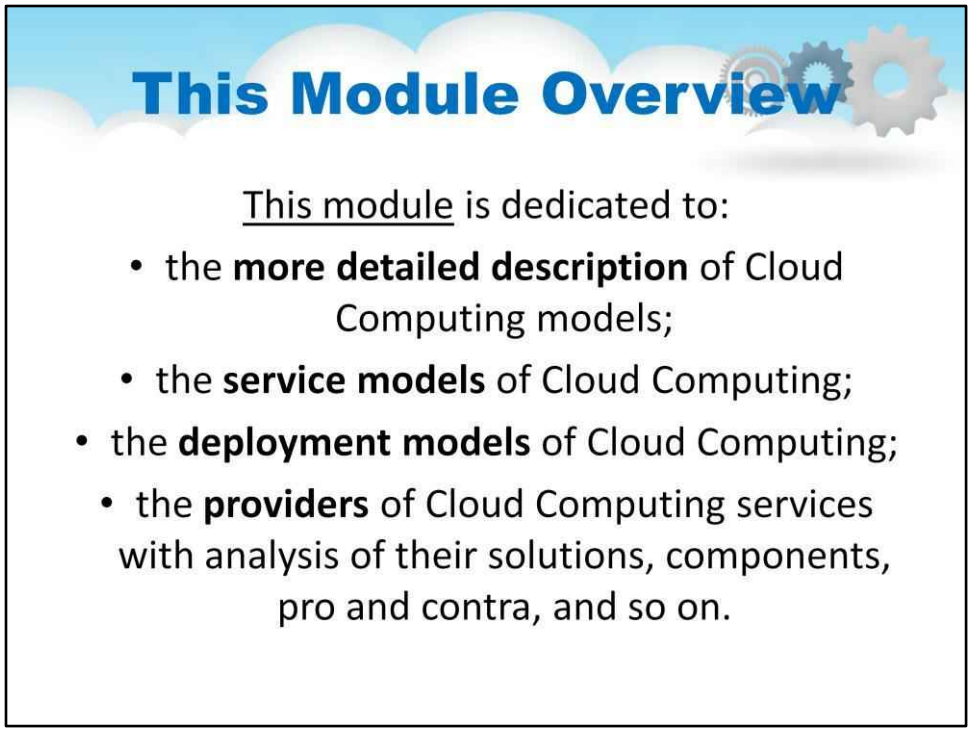
Модуль 4

Технології та види

Хмарні обчислення

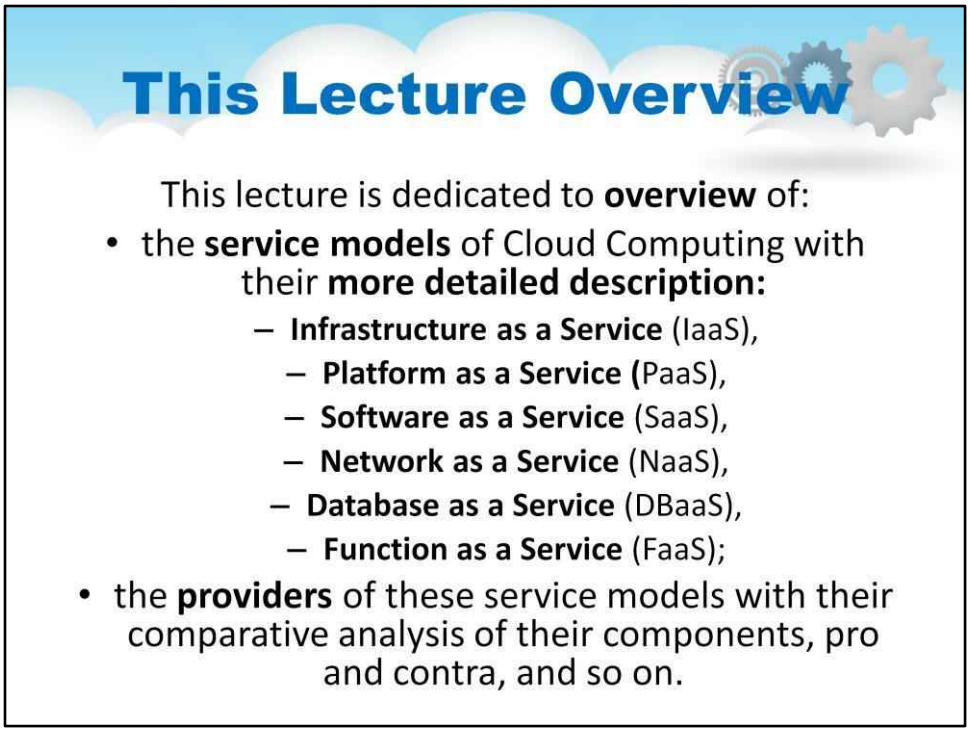
Зміст


Cloud Computing
Module 4 –
Technologies and Types
of Cloud Computing
Lecture 1. Service
Models



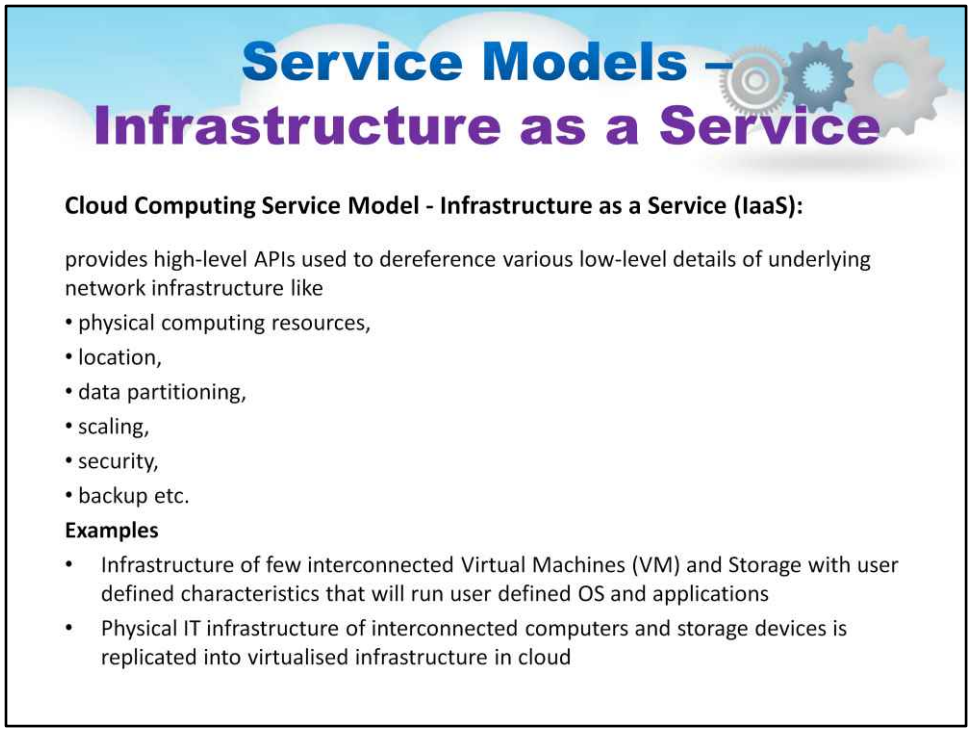
Модуль 4. Технології і типи хмар

обчислювальна техніка



Лекція 1. Сервісні моделі

Огляд

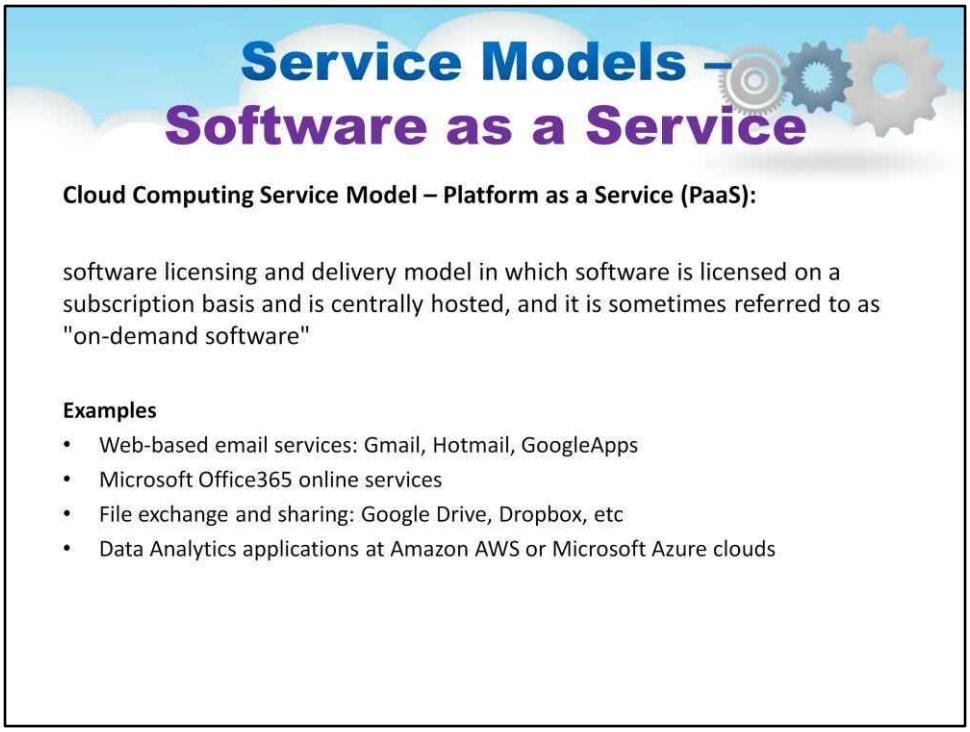


Можливість, що надається споживачу, полягає в забезпеченні обробки, зберігання, мережі,та інші фундаментальні обчислювальні ресурси, де споживач може розгортати та запускати довільне програмне забезпечення, яке може включати операційні системи та програми. Споживач не керує базовою хмарною інфраструктурою та не контролює її, але має контроль над операційними системами, сховищами, розгорнутими програмами таможливо обмежений контроль вибраних мережевих компонентів (наприклад, брандмауери хоста).

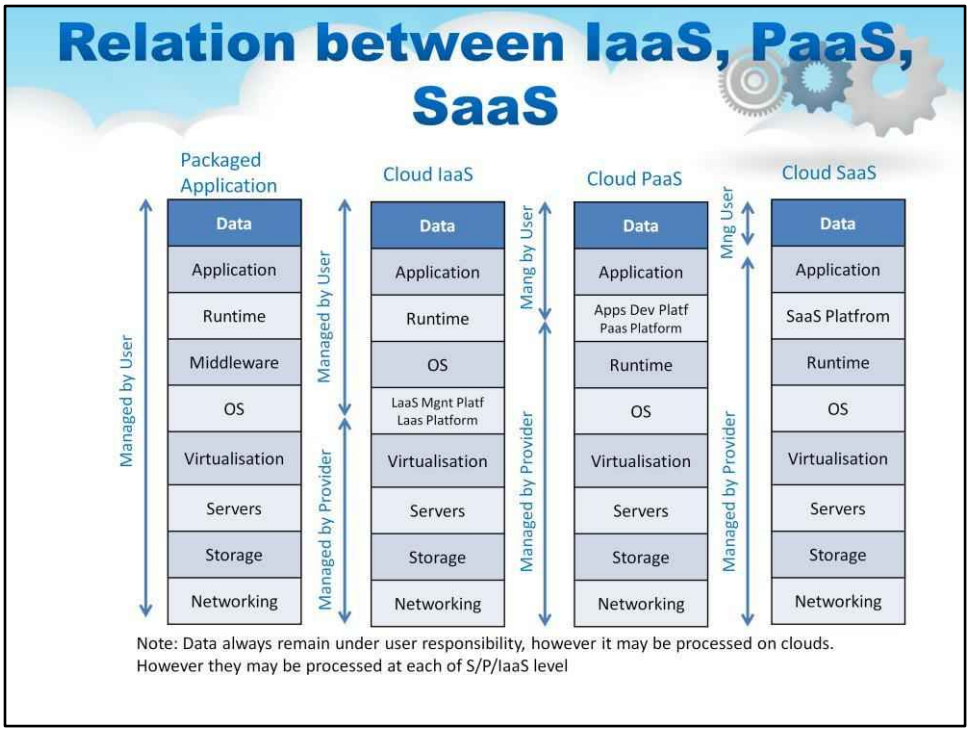
Можливість, яка надається споживачеві, полягає в розгортанні в хмарній інфраструктурі створених споживачем або придбаних додатків, створених за допомогою мови програмування, бібліотеки, служби та інструменти, які підтримуються постачальником. Споживач не керує та не контролює базову хмарну інфраструктуру, включаючи мережу, сервери, операційні системи або сховище, але має контроль над розгорнутими програмами та, можливо, параметрами конфігурації середовища розміщення програм.

PaaS можна доставити трьома способами:

- як загальнодоступна хмарна служба від постачальника, де споживач контролює розгортання програмного забезпечення з мінімальними параметрами конфігурації, а постачальник надає мережі, сервери, сховище, операційну систему (ОС), проміжне програмне забезпечення (наприклад, середовище виконання Java, середовище виконання .NET, інтеграцію тощо). .), база даних та інші послуги для розміщення споживача додаток.
- як приватну службу (програмне забезпечення або пристрій) всередині брандмауера.
- як програмне забезпечення, розгорнуте в публічній інфраструктурі як послуга.



Можливість, яка надається споживачеві, полягає в тому, щобвикористовувати запущені програми провайдера на хмарній інфраструктурі. Програми доступні з різних клієнтських пристроїв або через тонкий клієнтський інтерфейс, такий як веб-браузер (наприклад, веб-електронна пошта), або програмний інтерфейс. Споживач не управляє і не контролює базовий актив хмарна інфраструктура, включаючи мережу, сервери, операційні системи, сховища або навіть можливості окремих програм, за можливим винятком обмежена залежність від користувача налаштування конфігурації програми



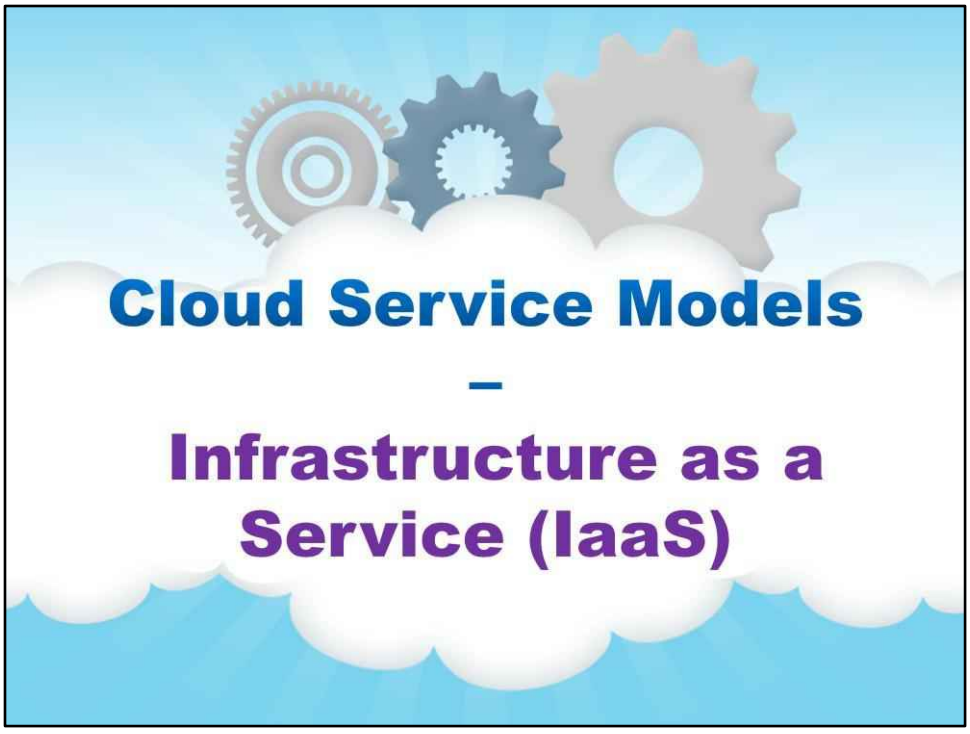
Моделі використання хмарних обчислень тісно пов’язані.

Для традиційної «фізично встановленої» упакованої програми — усе розгортання стеком керує користувач, як видно на першому блоці ілюстрації. Користувач несе відповідальність за надання повного сервера, включаючи мережу та сховище, а також програмне забезпечення операційної системи. Крім того, користувач надає необхідні бази даних і проміжне програмне забезпечення, будь-яке спеціальне середовище виконання (Java або Ruby) і, нарешті, програму. Дані програми також є відповідальністю (і власністю) користувача.

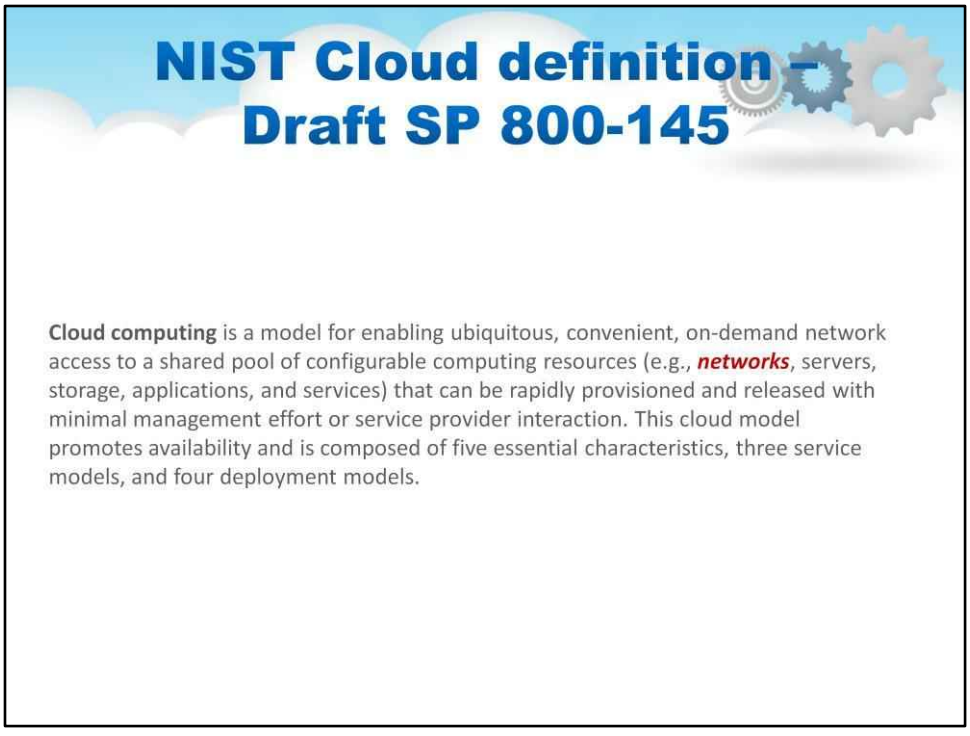
Модель Cloud IaaS суттєво змінює це, полегшуючи принаймні значну частину те, що вважатиметься «фізичною» інфраструктурою, показано у другому блоці ілюстрації. Програмне забезпечення для керування хмарою (іноді її називають Cloud OS) є показано на цій діаграмі як "IaaS Mngt Platf”.Користувач отримує обладнання в а віртуалізовану форму від IaaS CSP і турбується лише про програмний стек, програму та дані.

Модель Cloud PaaS також пропонує проміжне програмне забезпечення, час виконання програми тощо «програмної» інфраструктури, створюючи зручний контейнер для розробник додатків. Хоча це вимагає від розробника реструктуризації, якщо не переписування програми на рівні вихідного коду для інтерфейсу з PaaS, це обмеження додасть нові переваги масштабованості та доступності додатку.

Модель Cloud IaaS — це використання лише самої програми через браузер або веб-сервіс.

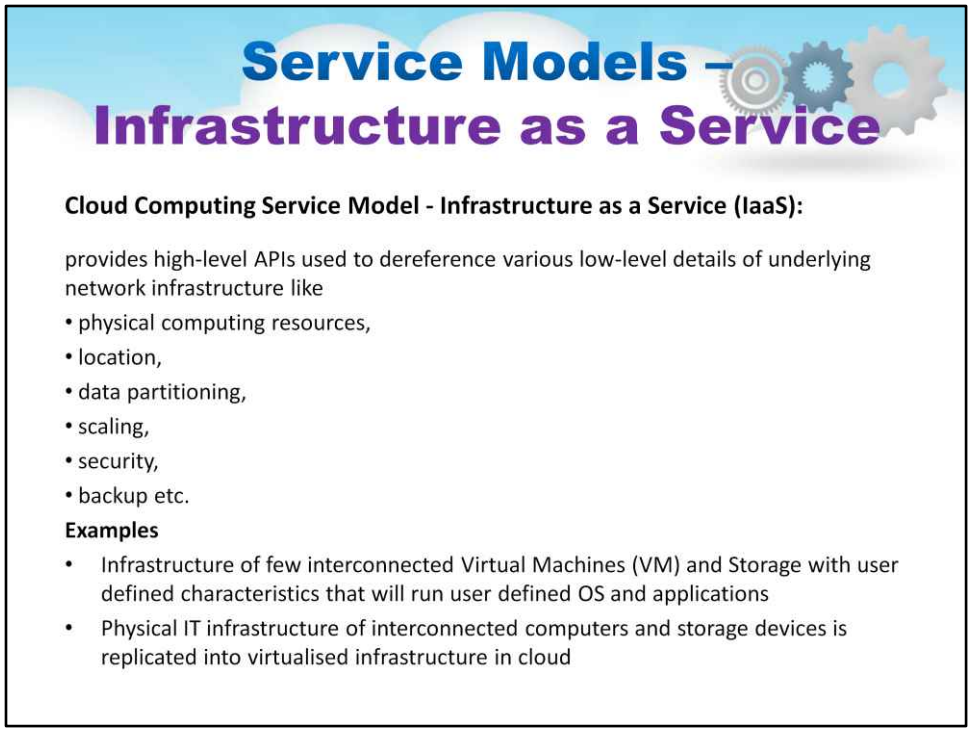


Інфраструктура як послуга (IaaS)

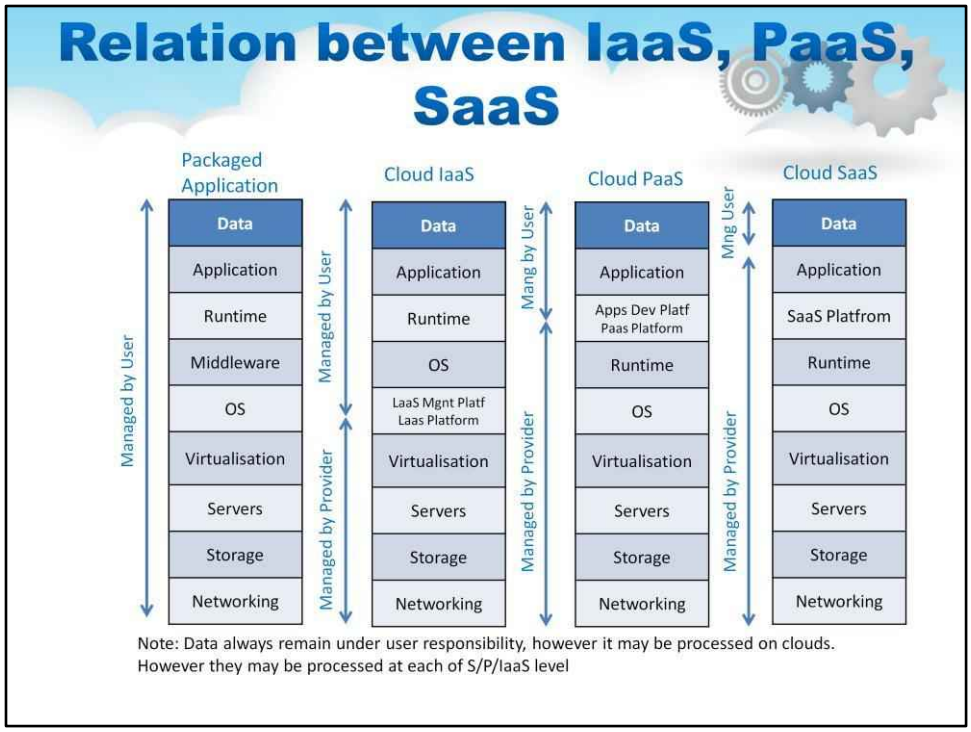


Ми розпочнемо урок із перегляду визначення хмари NIST.

Зокрема, визначення NIST є дуже конкретним щодо інфраструктури як послуги, яку ми спочатку дослідимо.



Можливість, що надається споживачу, полягає в забезпеченні обробки, зберігання, мережі,та інші фундаментальні обчислювальні ресурси, де споживач може розгортати та запускати довільне програмне забезпечення, яке може включати операційні системи та програми. Споживач не керує базовою хмарною інфраструктурою та не контролює її, але має контроль над операційними системами, сховищами, розгорнутими програмами таможливо обмежений контроль вибраних мережевих компонентів (наприклад, брандмауери хоста).



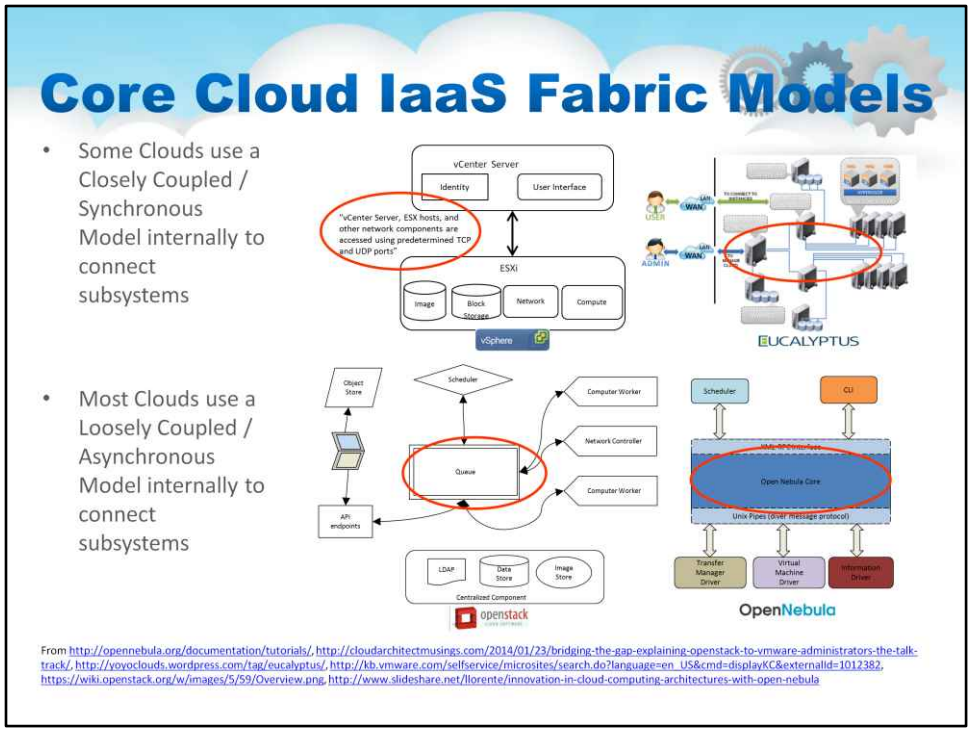
Моделі використання хмарних обчислень тісно пов’язані. Цю ілюстрацію було розглянуто в попередньому уроці, але її потрібно повторити.

Для традиційної «фізично встановленої» упакованої програми — усе розгортання стеком керує користувач, як видно на першому блоці ілюстрації. Користувач несе відповідальність за надання повного сервера, включаючи мережу та сховище, а також програмне забезпечення операційної системи. Крім того, користувач надає необхідні бази даних і проміжне програмне забезпечення, будь-яке спеціальне середовище виконання (Java або Ruby) і, нарешті, програму. Дані програми також є відповідальністю (і власністю) користувача.

Модель Cloud IaaS суттєво змінює це, полегшуючи принаймні значну частину те, що вважатиметься «фізичною» інфраструктурою, показано у другому блоці ілюстрації. Програмне забезпечення для керування хмарою (іноді її називають Cloud OS) є показано на цій діаграмі як "IaaS Mngt Platf”.Користувач отримує обладнання в а віртуалізовану форму від IaaS CSP і турбується лише про програмний стек, програму та дані.

Модель Cloud PaaS також пропонує проміжне програмне забезпечення, час виконання програми тощо «програмної» інфраструктури, створюючи зручний контейнер для розробник додатків. Хоча це вимагає від розробника реструктуризації, якщо не переписування програми на рівні вихідного коду для інтерфейсу з PaaS, це обмеження додасть нові переваги масштабованості та доступності додатку.

Модель Cloud IaaS — це використання лише самої програми через браузер або веб-сервіс.



Внутрішньо технологія в хмарних «операційних системах» серед різних альтернатив досить сильно відрізняється. Це безпосередньо впливає на фокус або цільовий варіант використання цієї системи.

Взагалі кажучи, одна з головних цілей CloudOS — організувати всі різні служби на серверах, які маніпулюють віртуальними машинами, керують розподіленим сховищем, налаштовують параметри мережі тощо. Таким чином, механізм зв’язку, який використовує CloudOS, є «хребтом» системи та є ключовим елементом архітектурного дизайну системи.

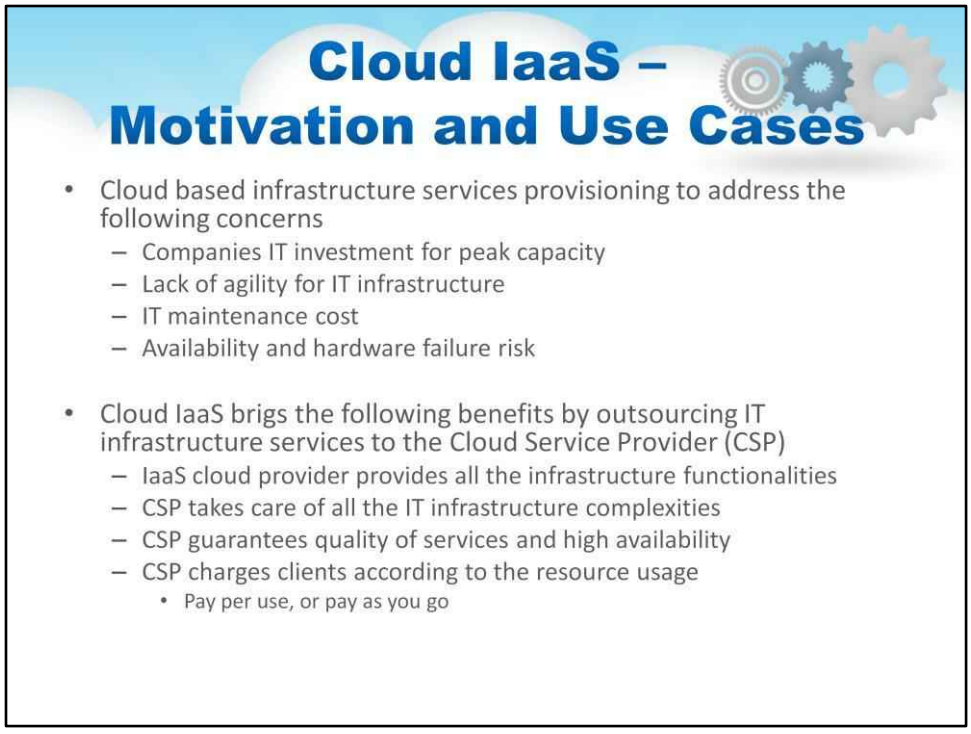
Ці механізми зв’язку можна розділити на дві великі категорії: тісно пов’язана/синхронна модель проти слабозв’язаної/асинхронної моделі.

Тісно пов’язана/синхронна модель є простішою архітектурно. Він призначений, перш за все, для забезпечення чудових характеристик, видимості та контролю. Зазвичай це включає з’єднання TCP/IP і Socke і, можливо, Unicast або Multicast на додаток до цього для зв’язку. Як побічний ефект цієї архітектури з’являються обмеження масштабованості, а також окремі точки відмови (центральний елемент оркестровк Ці архітектури чудово підходять для невеликих приватних хмар, де не потрібен великий масштаб і де потрібен жорсткий контроль запущених програм (наприклад, застаріле програмне забезпечення, що працю на вузлах).

Як приклад тісно пов’язаної/синхронної моделі – VMware vSphere і Eucalyptus, обидві використовують TCP/IP і сокети як структуру.

Слабозв’язана/асинхронна модель є складнішою архітектурно. Він призначений, перш за все, для забезпечення відмінної масштабованості та високих характеристик доступності. Зазвичай для зв’язку використовується система черги повідомлень. Як побічний ефект цієї архітектури може бути менш детальний контроль і менший контроль у реальному часі вузлів. Ці архітектури чудово підходять для великих хмар, особливо для найбільших публічних хмар, де потрібне масштабування та надмірність керуючих елементів.

Як приклад слабозв’язаної/асинхронної моделі – OpenStack і OpenNebula використовують механізм черги повідомлень як структуру.



Як вибрати серед цих різних підходів до використання хмари? Є багато рушійних факторів.

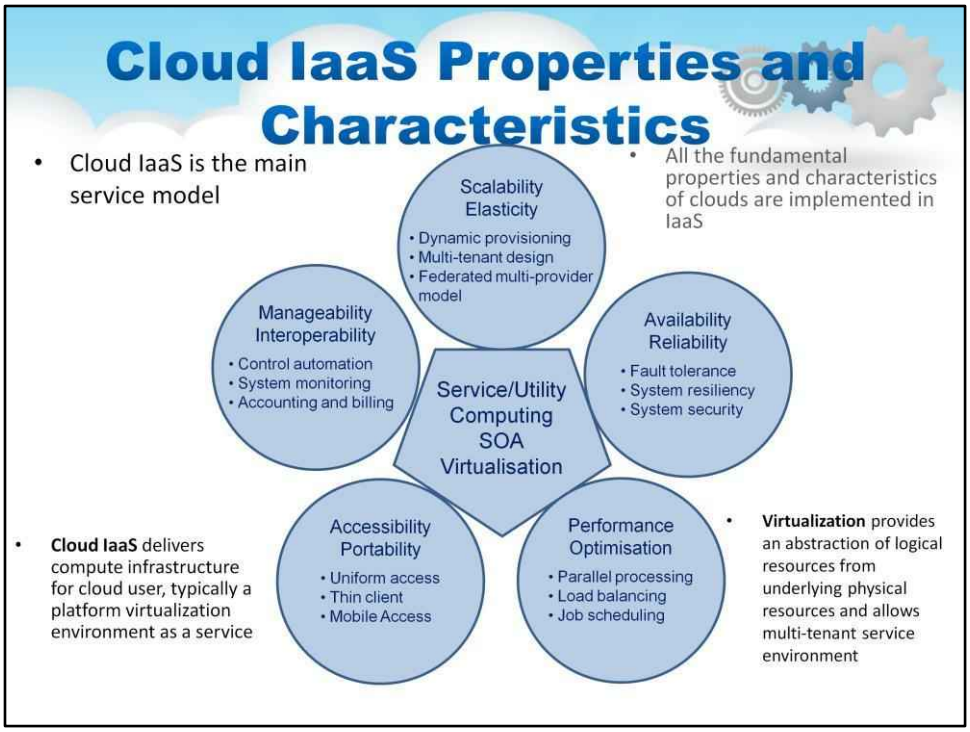
IaaS – це певним чином легший режим для прийняття, особливо якщо ви використовуєте комерційне програмне забезпечення, а не розробляєте PaaS в IT-магазині.

Цей слайд охоплює кілька причин, чому компанії звертаються до IaaS. Деякі з цих вимог виражені на концептуальному рівні «фізичного розгортання», іншими словами, «потрібно більше серверів»; деякі стурбовані операційними вдосконаленнями, такими як гнучкість або доступність.

IaaS може подбати про всі ці переваги, якщо програми переміщуються в IaaS відповідно до кількох найкращих практик. Щоб реалізувати переваги Хмари, слід деталізувати переваги, які очікуються від Постачальника Хмарних послуг:

CSP забезпечує заміну нашої власної фізичної інфраструктури.
CSP бере на себе всі складності IT-інфраструктури
CSP гарантує якість послуг і високу доступність CSP стягує
плату з клієнтів відповідно до використання ресурсів

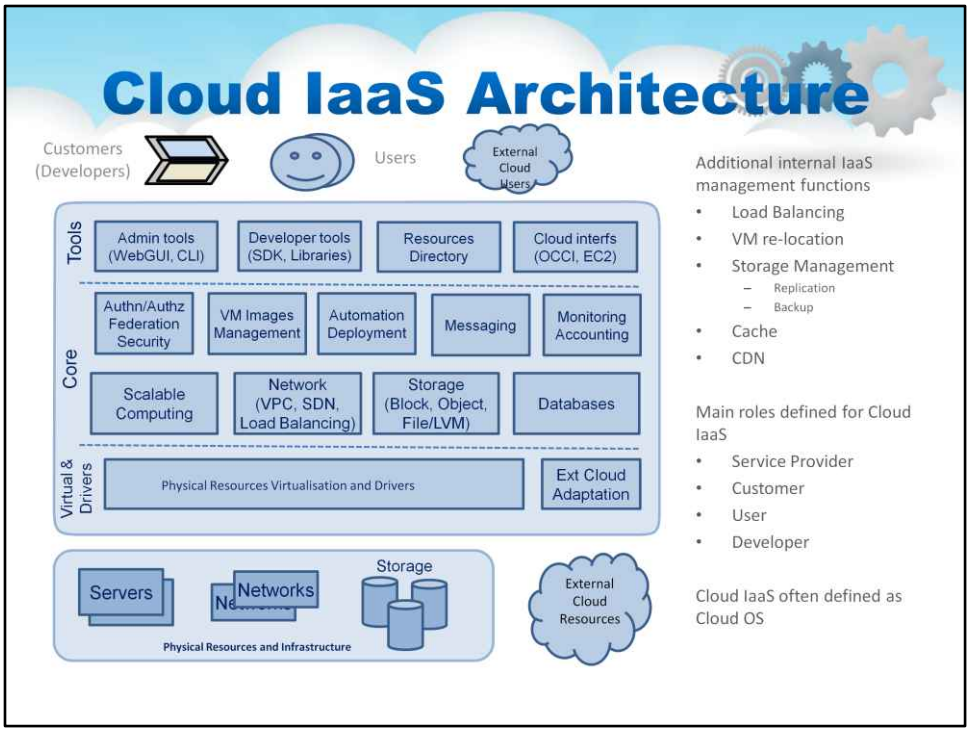
Переконайтеся, що проведено аналіз, щоб зрозуміти очікування та вплив кожного з них на відділ.



Cloud IaaS представляє повний набір характеристик хмари. Це більше, ніж просто використання віртуалізації або запуск чогось за межами вашого центру обробки даних. Усі фундаментальні властивості та характеристики хмар реалізовано в IaaS, і це створює синергію, яка дозволяє створювати нові характеристики.

Хмара поєднує в собі такі технології: обчислення сервісів/допоміжних засобів (автоматизація), SOA та віртуалізація, де віртуалізація є ключовою технологією для активації всіх властивостей хмари.

Хмара не надає програмі автоматично нові чарівні можливості, якщо її не розгорнути належним чином, щоб скористатися новими можливостями. Наприклад, масштабованість і еластичність можна досягти в хмарі, але потрібні можливості в модулях, які ви збираєтеся масштабувати вгору та вниз, а також вимагає певного коду керування чи автоматизації, який потрібно налаштувати для цієї програми. Хмара вмикає цю нову можливість, але не автоматично. Ми розглянемо масштабованість/ еластичність, доступність/надійність, оптимізацію продуктивності, доступність/ портативність, а також керованість і сумісність
оскільки ми більше вивчаємо IaaS і дізнаємося, як увімкнути функції в розгортанні програми.



Cloud IaaS часто визначають як хмарну ОС (наприклад, OpenStack, Microsoft), оскільки загалом вона забезпечує всі необхідні функції для керування ресурсами хмари та підтримки додатків.

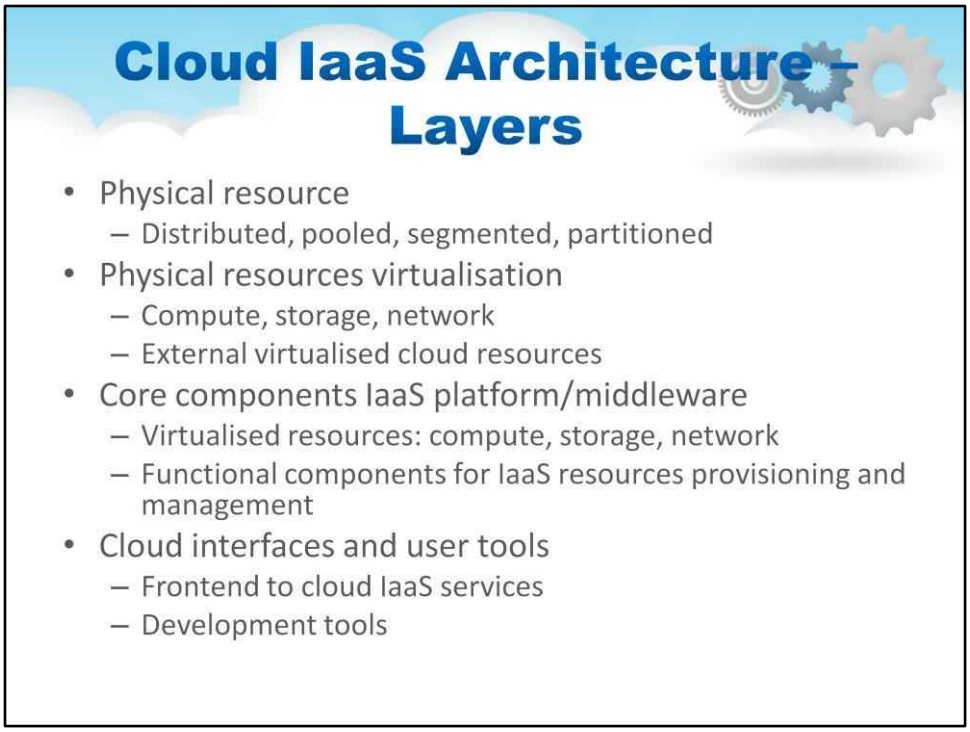
Зовнішні хмарні ресурси можуть бути різними віртуалізованими ресурсами (серверами, сховищем) або ресурсами, наданими іншими хмарами.

Зовнішні ресурси можуть використовувати хмарні API

Зверніть увагу на різницю між ролями: клієнт і користувач

Клієнт – це суб’єкт, який замовляє/купує хмарні послуги. Вони встановлюють і запускають/адмініструють хмарні програми або служби. Як сайт. Зазвичай розробники повністю контролюють хмарні ресурси.

Користувачі — це суб’єкти, які використовують хмарні сервіси, наприклад користувачі веб-сайтів.



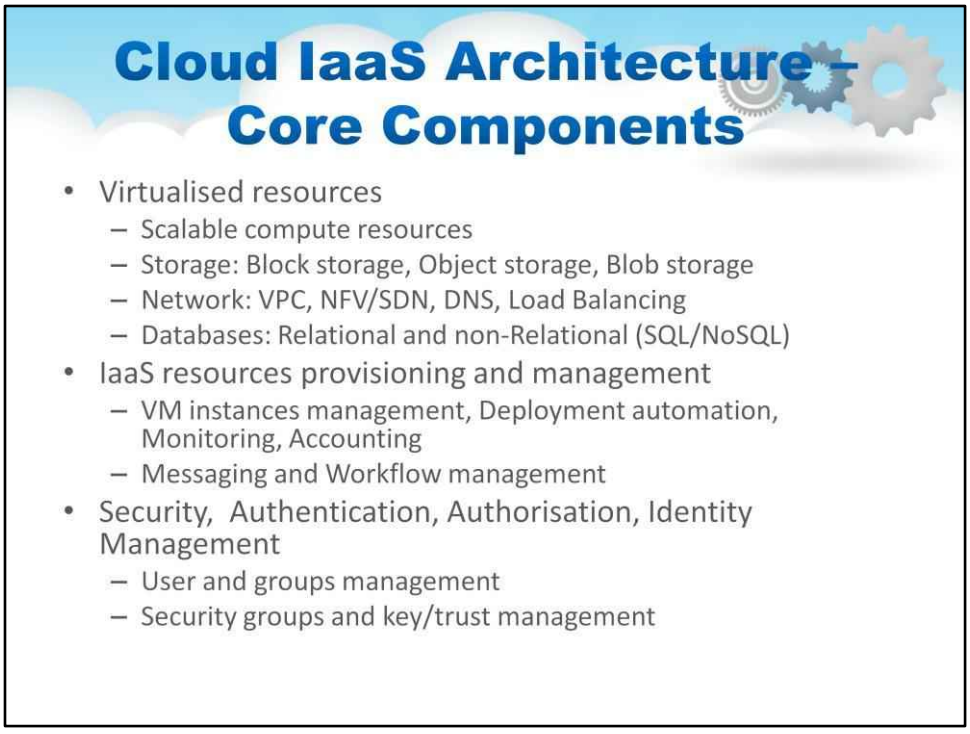
Як видно на попередній ілюстрації, Cloud IaaS має легко розрізнити рівні архітектури.

Рівень фізичних ресурсів — це апаратне та мікропрограмне забезпечення, з'єднане в певній топології з мережею та сховищем

Рівень віртуалізації віртуалізує та розміщує під програмним керуванням рівень фізичних ресурсів

Далі на віртуалізації працюють компоненти керування хмарою модулі «Cloud OS».

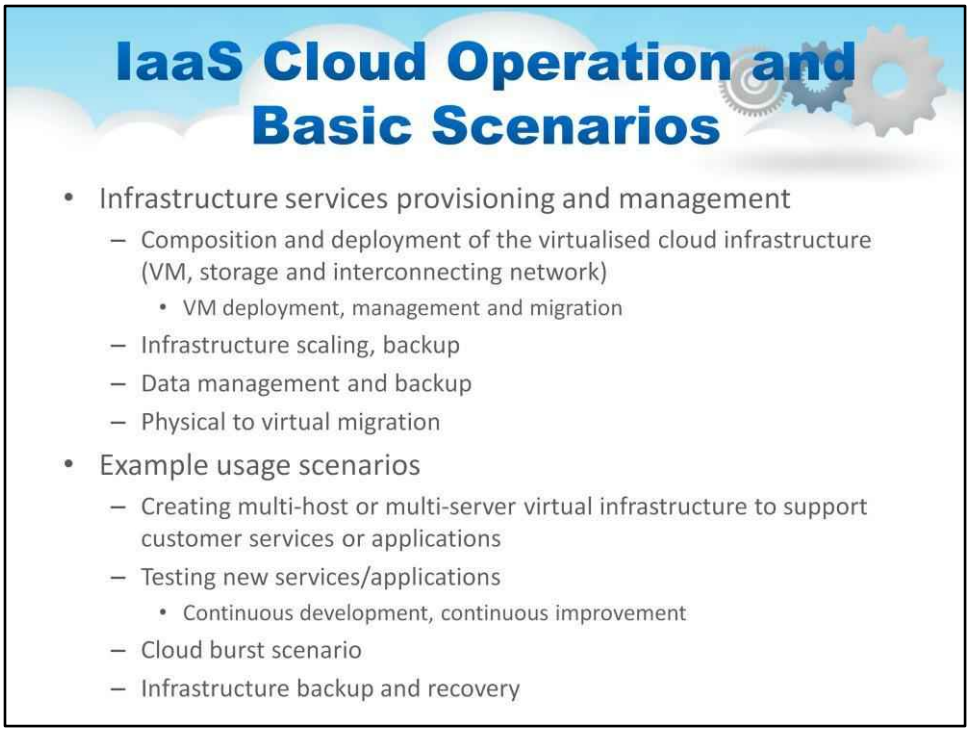
Нарешті, є інтерфейси та інструменти, які ми в кінцевому підсумку використовуємо для доступу до внутрішніх компонентів хмари



Хмарна ОС стала дуже повнофункціональною та продовжує зростати можливостями. Його робота спочатку відповідала за основну функціональність IaaS віртуалізованих ресурсів і автоматизацію ключових процесів у забезпеченні ресурсами.

Тепер хмарна ОС також надає кілька моделей зберігання, керовану програмою мережу з різними функціями та підтримку багатьох видів зберігання даних (бази даних).

Інші функції Cloud OS детально описано на слайді, де ви можете побачити механізми, зокрема автентифікацію, керування ключами, облік, робочий процес та інші речі.



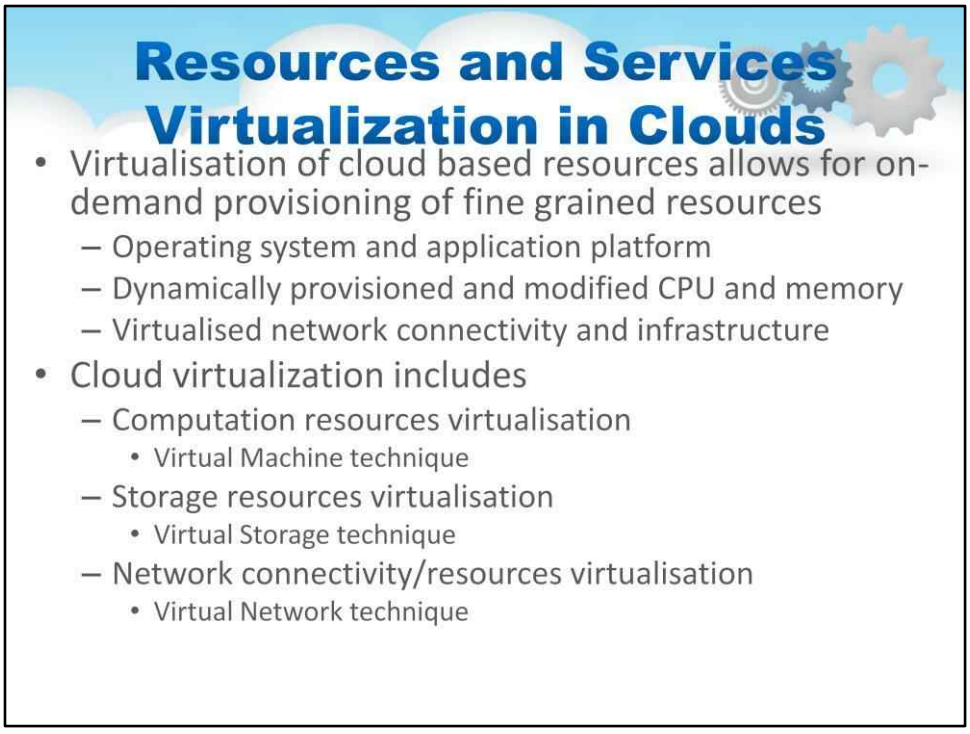
Давайте розглянемо пару сценаріїв базової хмарної роботи IaaS.

Теми, які ми розглянемо, включають створення та розгортання віртуальних машин, які нам знадобляться, масштабування інфраструктури, резервне копіювання

Керування даними та резервне копіювання, а потім фізична міграція до віртуальної

З цих базових кроків ми створимо на папері кілька типових сценаріїв розгортання

Ми розглянемо створення багатохостової системи, як ми проводимо тестування, як працює Cloud Burst і як щодо резервного копіювання та відновлення

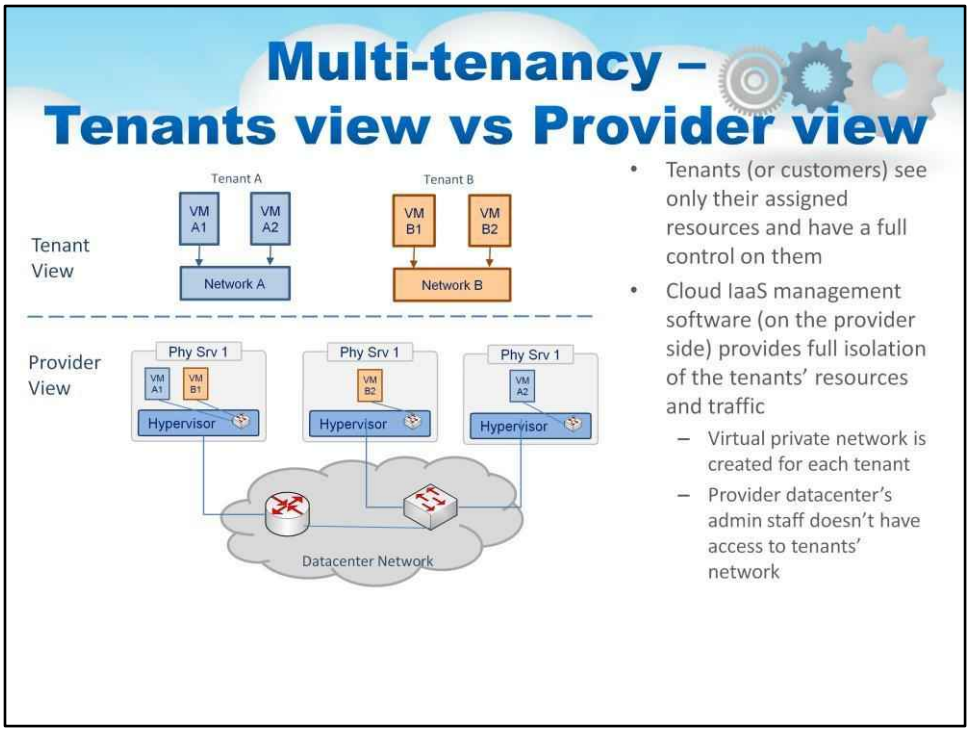


Для початку ми зосереджуємося на віртуалізації ресурсів і послуг у хмарах.

Ми збираємося розгорнути фактичні сервери, необхідні для запуску наших програм. Через чітко визначені інтерфейси до хмари, наприклад, веб-сервіси або REST, Cloud OS динамічно надає ЦП і пам’ять, організовує підключення до віртуалізованої мережі та інфраструктуру.

Включає хмарну віртуалізацію

- Метод віртуалізації обчислювальних ресурсів Virtual Machine
- Віртуалізація ресурсів зберігання, техніка Virtual Storage
- Віртуалізація підключення до мережі/ресурсів, техніка віртуальної мережі

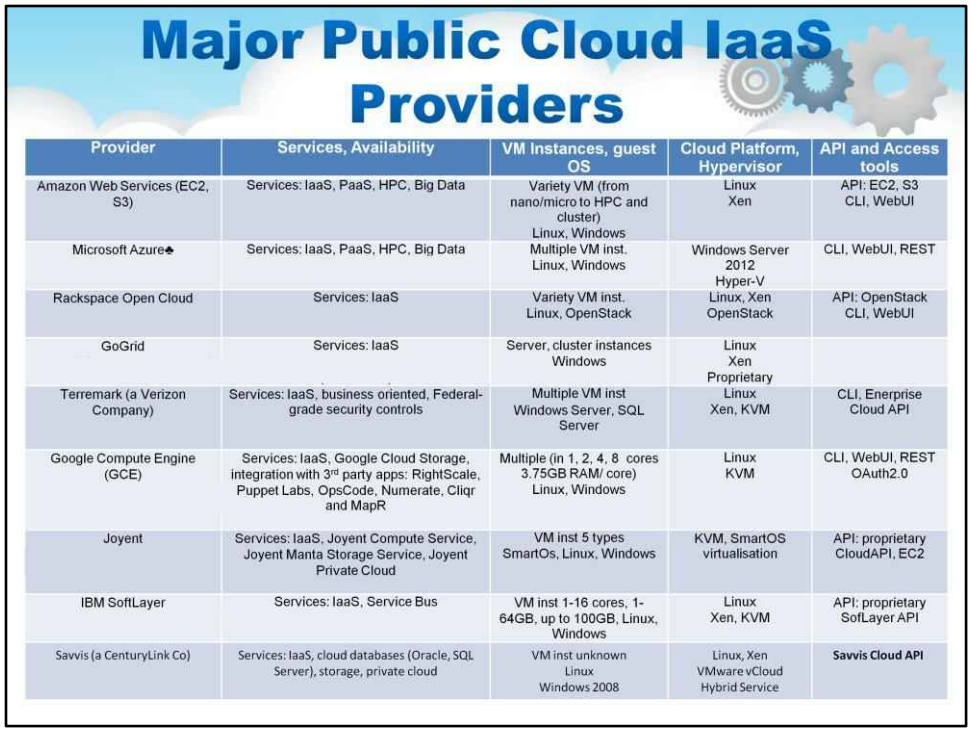


Як хмара фактично реалізує програмну доступність віртуальних машин, які обслуговуються різним клієнтам?

На ілюстрації показано «логічне» подання віртуальних машин і мереж. Зверніть увагу на кодування кольорів.

Провайдер «обслуговує» хмарні ресурси, використовуючи різноманітні параметри, враховуючи пул фізичних серверів, які він має в центрі обробки даних.

На цій ілюстрації показано, що Physical Server 1 запускає віртуальну машину для кожного клієнта, розділяючи ресурси машини між двома віртуальними машинами. Інші віртуальні машини розкидані на інших серверах у хмарі.

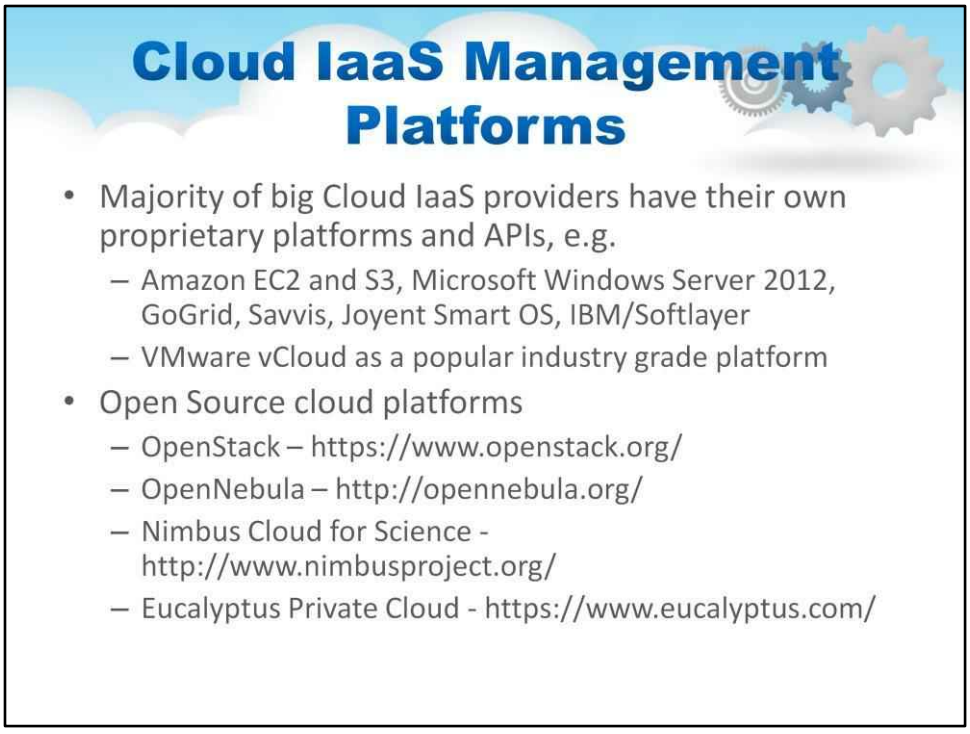


Хмарні постачальники IaaS відрізняються за платформою, API, місцем розташування, але більшість пропонують образи віртуальних машин Linux і Windows. Деякі CSP використовують власні платформи керування хмарою, деякі також пропонують платформи керування хмарою VMware.

Великі CSP пропонують послуги по всьому світу та мають кілька зон доступності.

(CSP1, 2104) Список постачальників IaaS: Порівняння та посібник за 2014 рік. [онлайн] http://www.tomsitpro.com/articles/iaas-providers,1-1560.html

(CSP2, 2014) Джо Панеттьєрі, список і дослідження 100 найкращих постачальників хмарних послуг (CSP) [онлайн] http://talkincloud.com/tc100

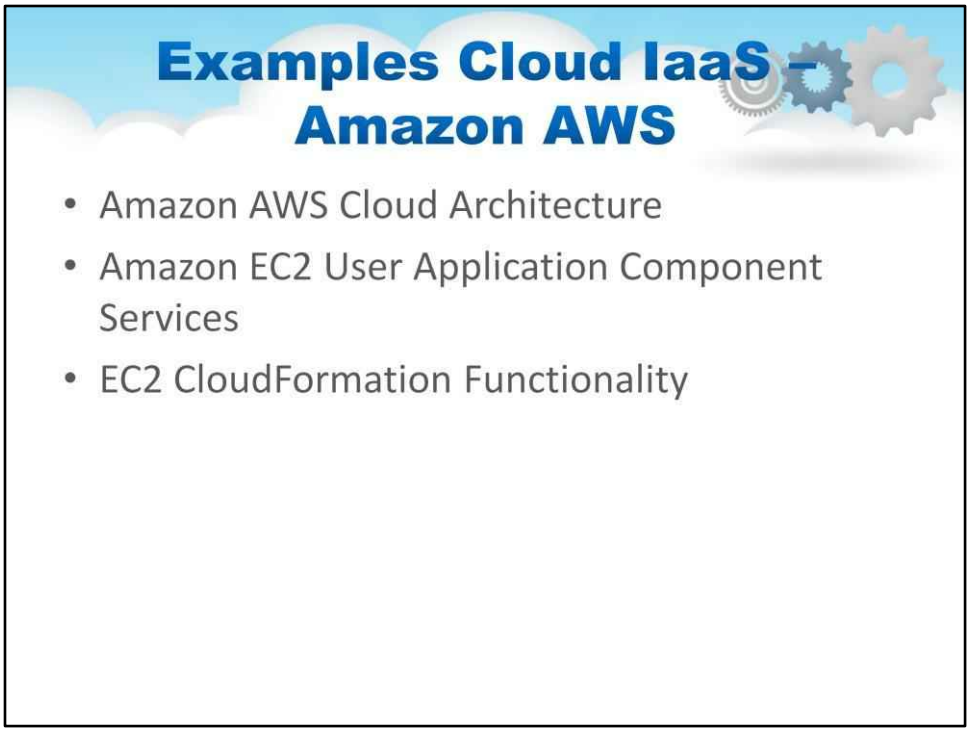


Проблема для розробника полягає в тому, щоб навіть IaaSAPI та конвенції різні між постачальниками хмарних послуг.

На слайді перелічено кілька постачальників хмарних послуг, а також альтернативи програмного забезпечення Cloud OS.

Незважаючи на те, що дизайн подібний, існує безліч хмарних платформ, які можна вибрати залежно від того, що зумовило ваш вибір.

Приклад: Amazon AWS



Найпопулярнішою хмарою IaaS є Amazon AWS. З цієї причини ми будемо посилатися на цю платформу, надаючи приклади рішень.

Архітектура AWS містить багато, багато сервісів. Перші і найосновніші є обчисленнями «EC2» і пов’язаними з ними основними частинами (S3 і EBS зберігання, наприклад).

Тому давайте досліджуватимемо AWS докладніше.

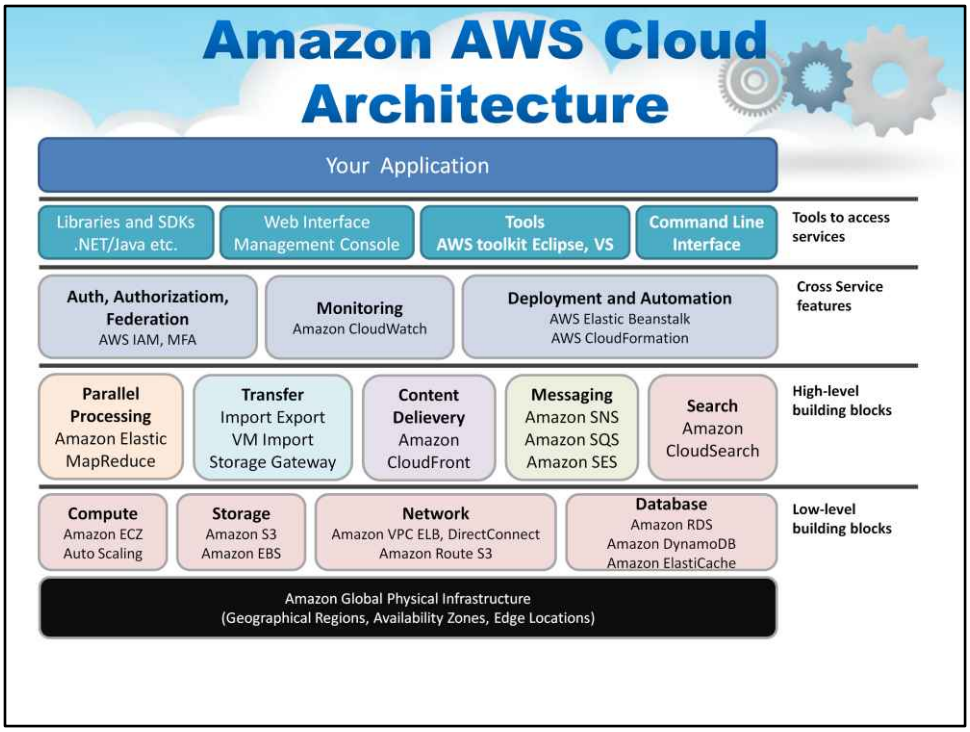
AWS пропонує дві основні послуги

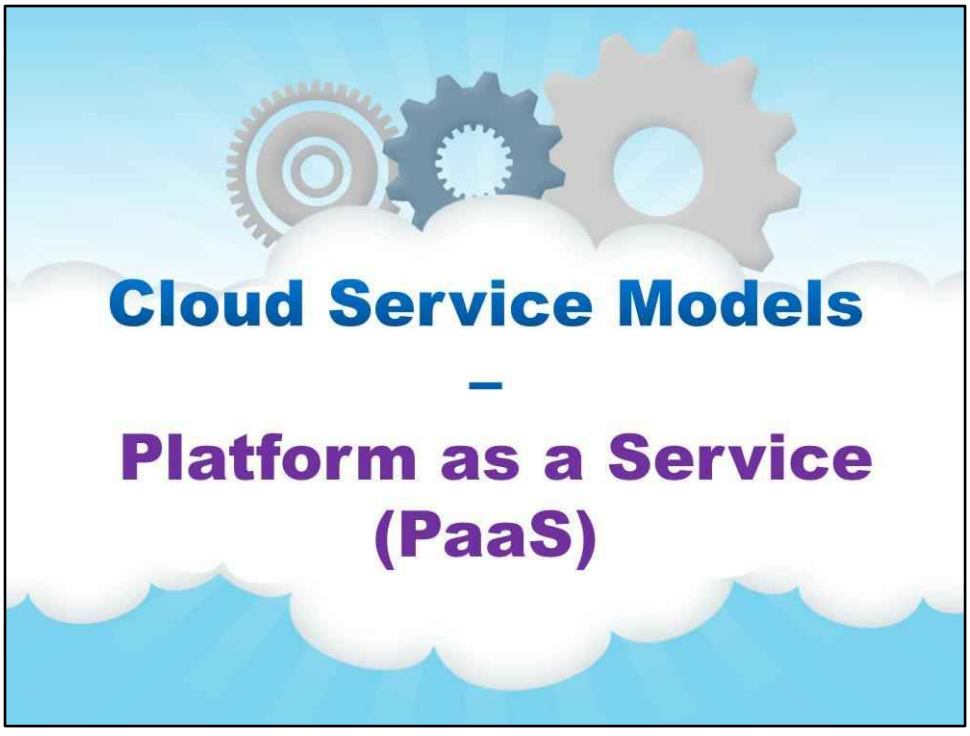
Elastic Compute Cloud (EC2), що забезпечує змінний розмір обчислювальної ємності Simple Storage Service (S3)

AWS також пропонує багато інших послуг, наприклад, усі види баз даних пропонує багато видів віртуальних машин і компонентів проміжного програмного забезпечення, все «як послуга».

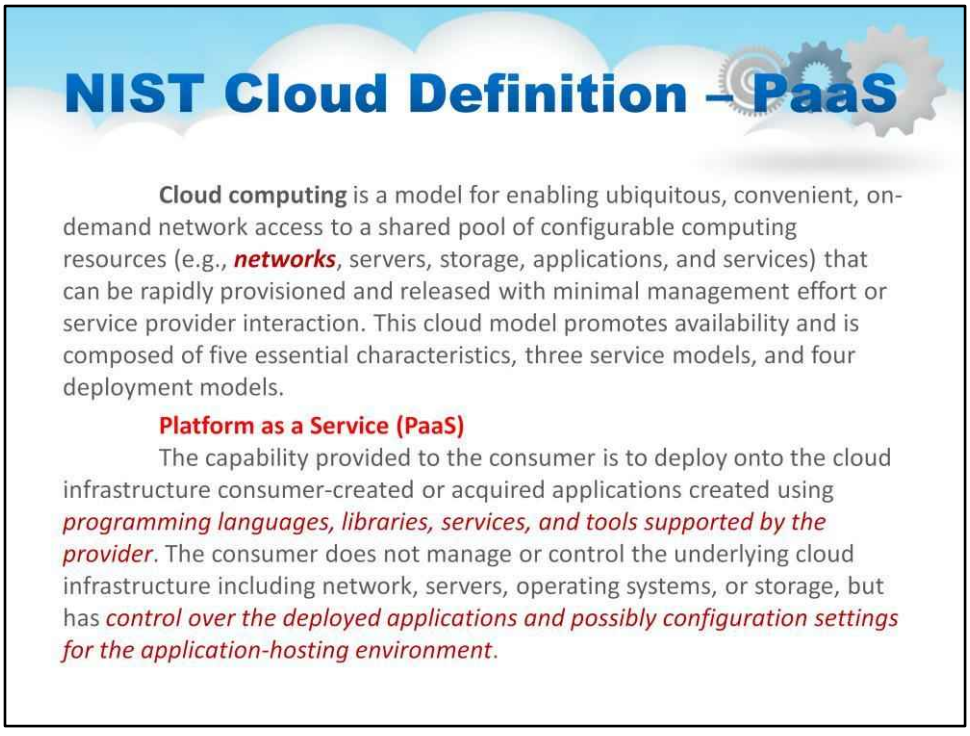
Amazon EC2 і S3 API стали стандартними де-факто інтерфейсами для доступу до хмарних сервісів і керування ними

AWS має 9 зон доступності на вибір:


Amazon AWS Cloud Architecture
Your Application
Libraries and SDKs .NET/Java etc.
Web Interface Management Console
Tools AWS toolkit Eclipse, VS
Command Line Interface
Tools to access services
Auth, Authorizatiom, Federation
AWS IAM, MFA
Monitoring
Amazon CloudWatch
Deployment and Automation
AWS Elastic Beanstalk
AWS CloudFormation
Cross Service features
Parallel Processing
Amazon Elastic MapReduce
Transfer
Import Export
VM Import
Storage Gateway
Content Delievery
Amazon CloudFront
Messaging
Amazon SNS
Amazon SQS
Amazon SES
Search
Amazon CloudSearch
High-level building blocks
Compute
Amazon ECZ
Auto Scaling
Storage
Amazon S3
Amazon EBS
Network
Amazon VPC ELB, DirectConnect
Amazon Route S3
Database
Amazon RDS
Amazon DynamoDB
Amazon ElastiCache
Low-level building blocks
Amazon Global Physical Infrastructure
(Geographical Regions, Availability Zones, Edge Locations)



Платформа як послуга (PaaS)



На цьому слайді цитується визначення платформи як послуги NIST Cloud Computing.

Спочатку слайд повторює визначення NIST хмарних обчислень у цілому.

Зверніть увагу на наголос на «всюдний, зручний мережевий доступ на вимогу до спільного пулу конфігурованих обчислювальних ресурсів»

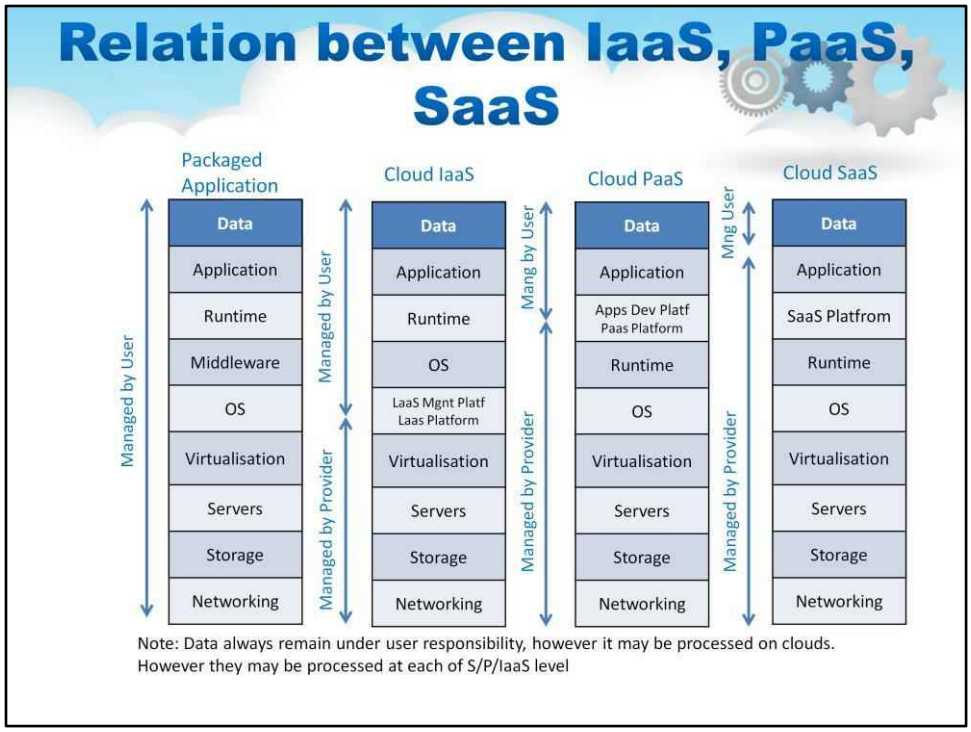
У визначенні NIST для платформи як послуги (PaaS) акцент робиться на «мови програмування, бібліотеки, служби та інструменти, які підтримуються постачальником. »

Тож PaaS — це середовище для розробників!

Можливість, яка надається споживачеві, полягає в розгортанні в хмарній інфраструктурі створених споживачем або придбаних додатків, створених за допомогою мови програмування, бібліотеки, служби та інструменти, які підтримуються постачальником. Споживач не керує та не контролює базову хмарну інфраструктуру, включаючи мережу, сервери, операційні системи або сховище, але має контроль над розгорнутими програмами та, можливо, параметрами конфігурації середовища розміщення програм.

PaaS можна доставити трьома способами:

- як загальнодоступна хмарна служба від постачальника, де споживач контролює розгортання програмного забезпечення з мінімальними параметрами конфігурації, а постачальник надає мережі, сервери, сховище, операційну систему (ОС), проміжне програмне забезпечення (наприклад, середовище виконання Java, середовище виконання .NET, інтеграцію тощо). .), база даних та інші послуги для розміщення споживача додаток.
- як приватну службу (програмне забезпечення або пристрій) всередині брандмауера.
- як програмне забезпечення, розгорнуте в публічній інфраструктурі як послуга.



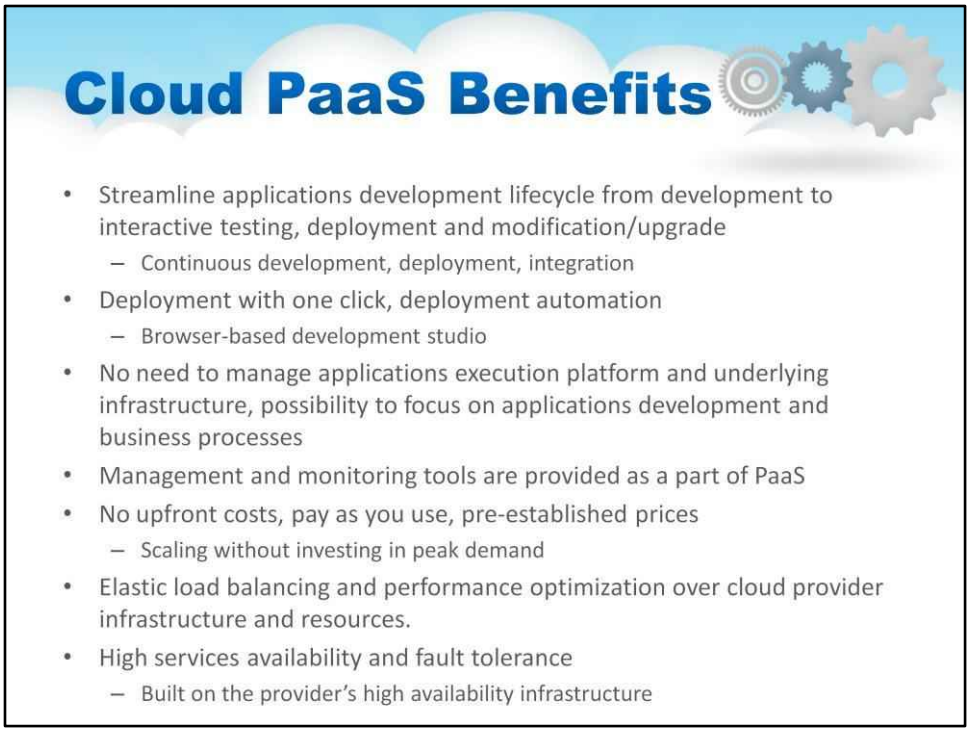
Моделі використання хмарних обчислень тісно пов’язані. Цю ілюстрацію було розглянуто в попередньому уроці, але її потрібно повторити.

Для традиційної «фізично встановленої» упакованої програми — усе розгортання стеком керує користувач, як видно на першому блоці ілюстрації. Користувач несе відповідальність за надання повного сервера, включаючи мережу та сховище, а також програмне забезпечення операційної системи. Крім того, користувач надає необхідні бази даних і проміжне програмне забезпечення, будь-яке спеціальне середовище виконання (Java або Ruby) і, нарешті, програму. Дані програми також є відповідальністю (і власністю) користувача.

Модель Cloud IaaS суттєво змінює це, полегшуючи принаймні значну частину те, що вважатиметься «фізичною» інфраструктурою, показано у другому блоці ілюстрації. Програмне забезпечення для керування хмарою (іноді її називають Cloud OS) є показано на цій діаграмі як "IaaS Mngt Platf”.Користувач отримує обладнання в а віртуалізовану форму від IaaS CSP і турбується лише про програмний стек, програму та дані.

Модель Cloud PaaS також пропонує проміжне програмне забезпечення, час виконання програми тощо «програмної» інфраструктури, створюючи зручний контейнер для розробник додатків. Хоча це вимагає від розробника реструктуризації, якщо не переписування програми на рівні вихідного коду для інтерфейсу з PaaS, це обмеження додасть нові переваги масштабованості та доступності додатку.

Модель Cloud IaaS — це використання лише самої програми через браузер або веб-сервіс.



PaaS, наданий хмарою, надає багато переваг розробнику в порівнянні з класичним проміжним програмним забезпеченням/пакетами програмування.

Він забезпечує спрощений життєвий цикл розробки програм із безперервною розробкою, розгортанням, інтеграцією

Він забезпечує розгортання одним клацанням миші, як правило, у студії розробки на основі браузера

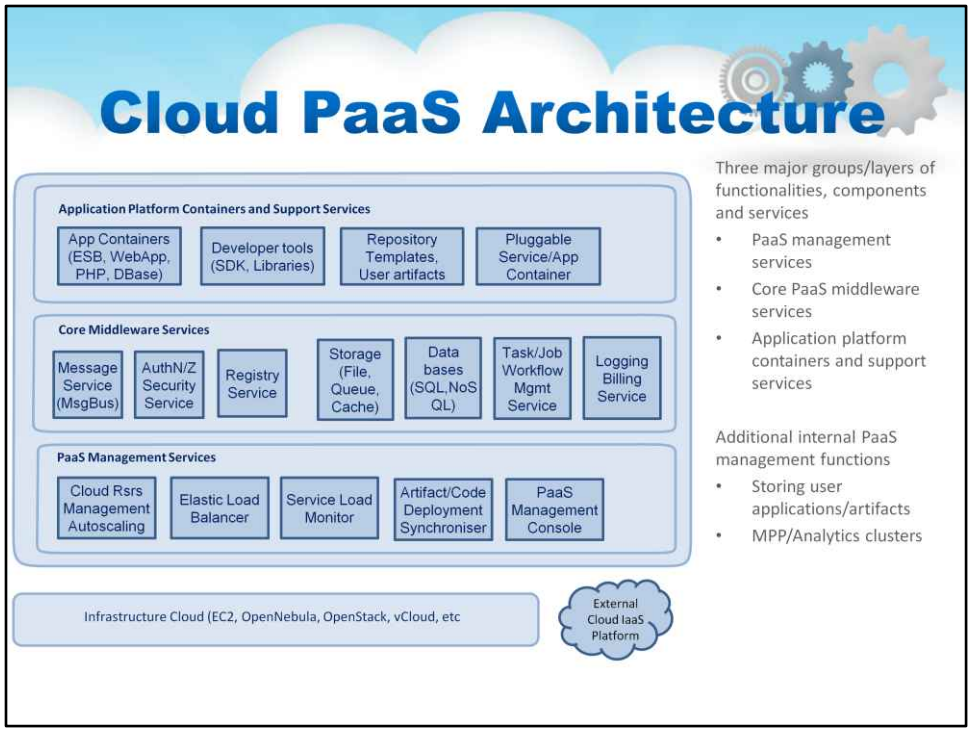
Не потрібно керувати платформою виконання додатків і базовою інфраструктурою

Надаються інструменти управління та моніторингу

Без попередніх витрат, платіть у міру використання

Більше автоматичного еластичного балансування навантаження та оптимізації продуктивності

Висока доступність послуг



Цей слайд містить ілюстрацію хмарної архітектури PaaS

Звичайно, під PaaS є інфраструктурна хмара (EC2, OpenNebula, OpenStack, vCloud тощо. Це може бути локальна або зовнішня хмарна платформа IaaS

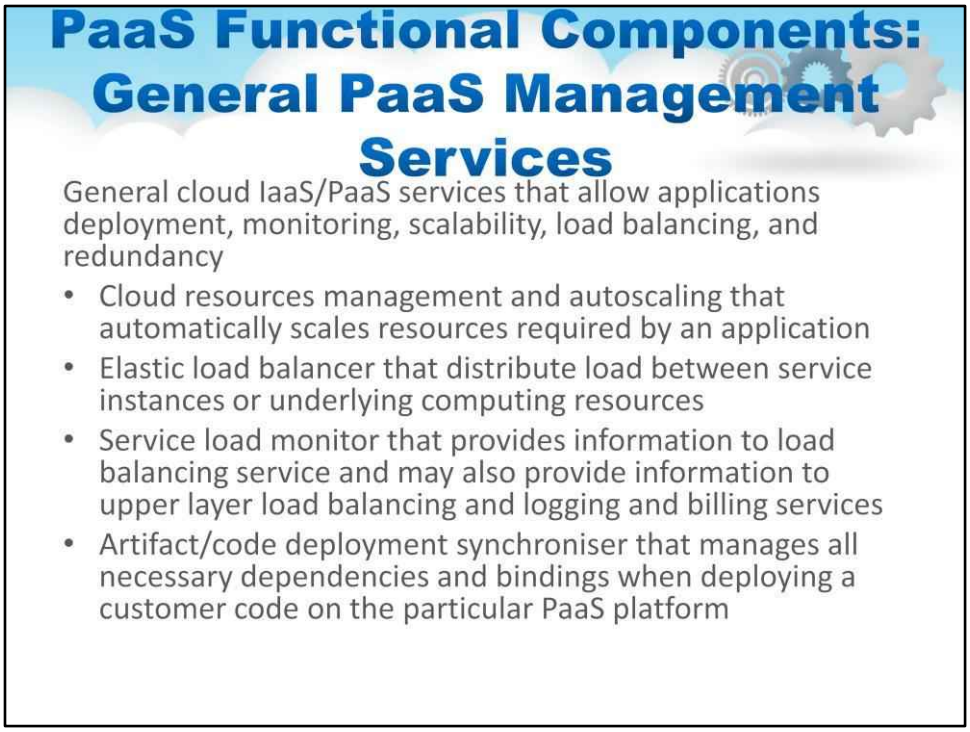
Є три основні групи/рівні функцій, компонентів і послуг

1. Служби керування PaaS, такі як Cloud Resources Management, Autoscaling, Elastic Load Balancer, Service Load Monitor і Artifact/Code Deployment, а також інші підсистеми залежно від різновиду PaaS.

2. Основні служби проміжного програмного забезпечення PaaS, такі як служби авторизації та безпеки, сховище (файл, черга, кеш), бази даних (SQL, NoSQL), служби керування робочим процесом завдань/робочих завдань та інші підсистеми залежно від різновиду PaaS

3. Контейнери платформи додатків і служби підтримки, такі як контейнери додатків (ESB, WebApp, PHP, DBase), інструменти розробника (SDK, бібліотеки), шаблони сховищ, Артефакти користувача та інші підсистеми залежно від різновиду PaaS

Додаткові внутрішні функції керування PaaS
Зберігання додатків/артефактів користувача
Кластери MPP/Analytics



Розглянемо ці елементи більш детально.

По-перше, функціональні компоненти PaaS

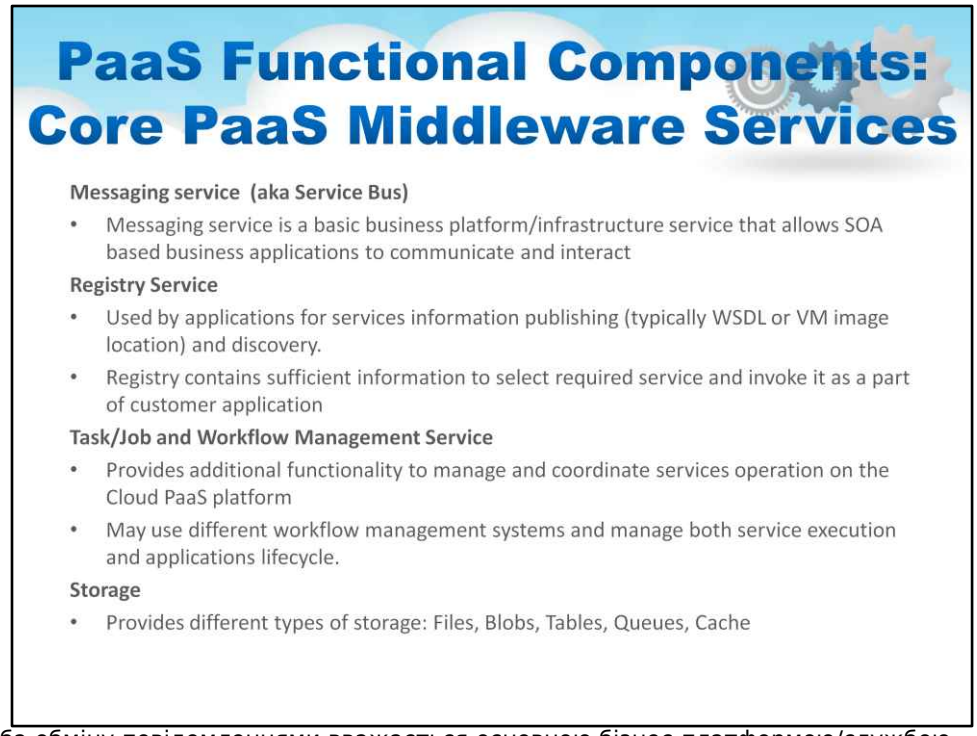
Загальні хмарні служби IaaS/PaaS дозволяють розгортати програми, моніторинг, масштабованість, балансування навантаження та резервування

Існує керування хмарними ресурсами та автомасштабування, яке автоматично масштабує ресурси, необхідні програмі

Існує еластичний балансувальник навантаження, який розподіляє навантаження між екземплярами служби або основними обчислювальними ресурсами

Існує монітор навантаження служби, який надає інформацію для служби балансування навантаження

Крім того, є синхронізатор розгортання артефакту/коду, який керує всіма необхідними залежностями та прив’язками під час розгортання коду користувача



Служба обміну повідомленнями вважається основною бізнес-платформою/службою інфраструктури, яка дозволяє бізнес-додаткам на основі SOA спілкуватися та взаємодіяти. SOA є основною архітектурою для бізнес-сервісів, які використовують повідомлення для зв'язку між сервісами та процесами.

Служба обміну повідомленнями має різні реалізації, де корпоративна службова шина (ESB) є найбільш широко використовуваною та стандартною де-факто для корпоративних програм на основі SOA. ESB має низку реалізацій з відкритим вихідним кодом та пропрієтарних реалізацій. В окремому посібнику ми обговоримо Azure Service Bus як хмарну службу обміну повідомленнями, яка реалізується Azure.

Реєстраційна служба

Веде реєстр сервісів, доступних на платформі, одному з ключових компонентів середовища на основі SOA

Використовується програмами для публікації та виявлення служб. Реєстр містить достатньо інформації для вибору необхідної служби та виклику її як частини клієнтської програми. Реєстр може містити або опис інтерфейсу служби (наприклад, WSDL), або посилання на пакет послуг або образ віртуальної машини.

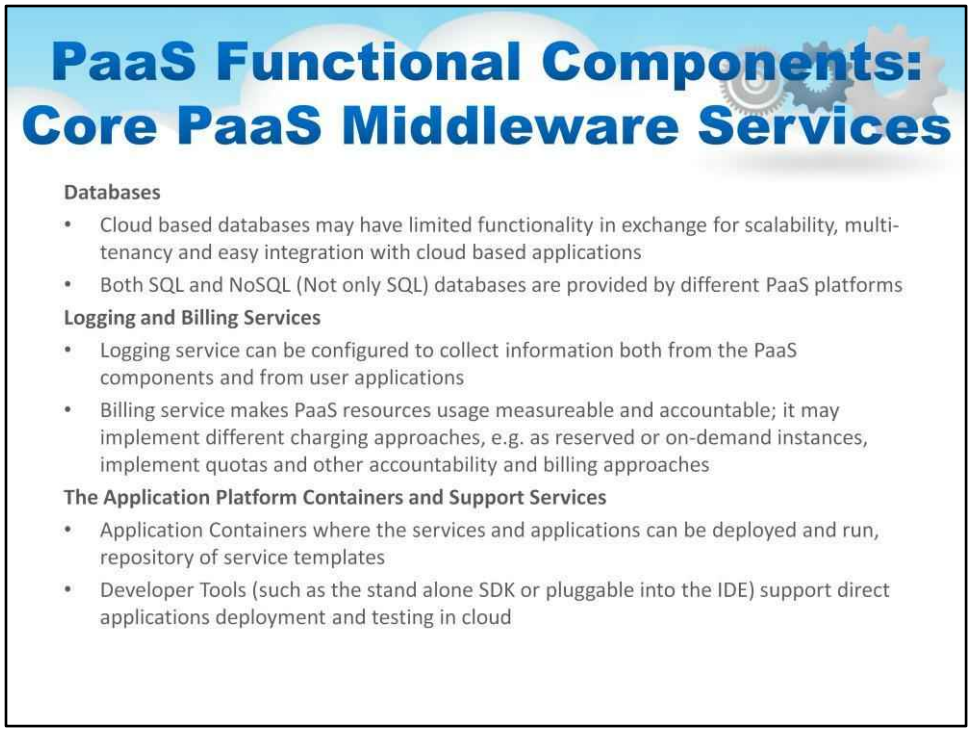
Служба керування завданнями/робочими процесами

Надає додаткові функції для керування та координації роботи сервісів на платформі Cloud PaaS

Може використовувати різні системи керування робочим процесом і керувати як виконанням послуг, так і життєвим циклом програм.

Зберігання

Забезпечує різні типи зберігання: файли, Blobs, таблиці, черги, кеш



Бази даних

Сервіси масштабованої бази даних є одним із ключових будівельних блоків програми

Хмарні бази даних можуть мати обмежену функціональність в обмін на масштабованість, мультиарендацію та легку інтеграцію з хмарними програмами. Обидва

Бази даних SQL і NoSQL (не тільки SQL) надаються різними платформами PaaS.

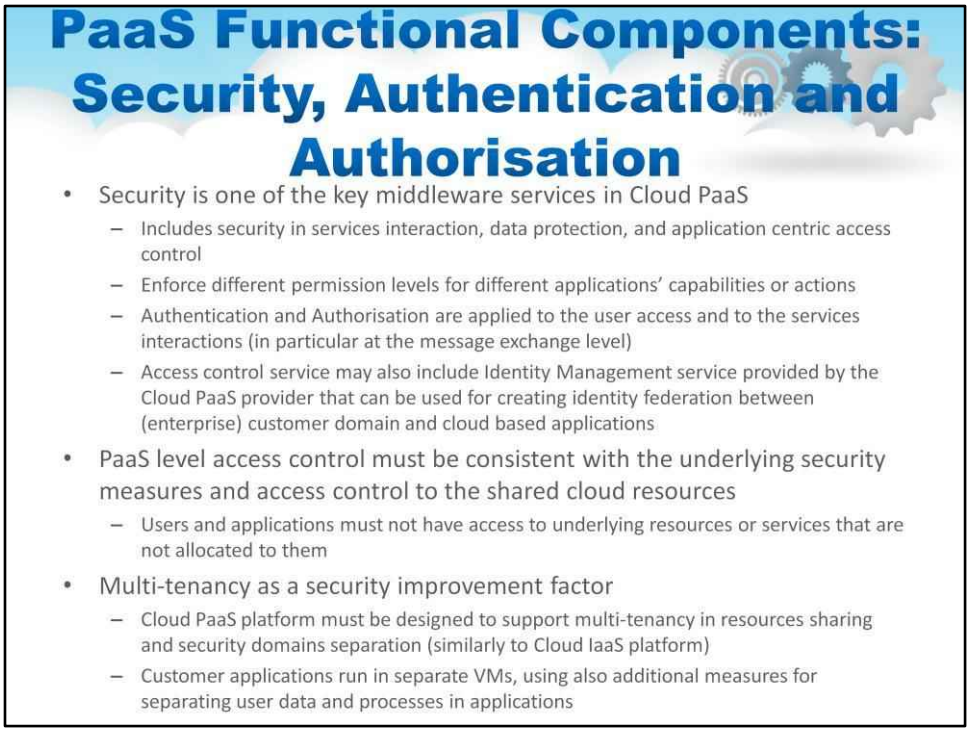
Послуги реєстрації та виставлення рахунків

Службу реєстрації можна налаштувати для збору інформації як з компонентів PaaS, так і з додатків користувача

Платіжний сервіс робить використання ресурсів PaaS вимірюваним і підзвітним; він може реалізовувати різні підходи до стягнення плати, наприклад, як зарезервовані або на вимогу екземпляри, реалізовувати квоти та інші підходи до звітності та виставлення рахунків.

Контейнери платформи додатків і служби підтримки

Включає контейнери додатків, де можна розгортати та запускати служби та додатки, сховище шаблонів служб та інструменти розробника, які включають комплекти розробки програмного забезпечення (SDK), які в більшості випадків можуть бути частиною інтегрованого середовища розробки (IDE) або підключатися до одного з популярні IDE, такі як Eclipse або Microsoft Visual Studio. Такі SDK/IDE підтримують пряме розгортання та тестування програм у хмарі. Кожен постачальник Cloud PaaS зазвичай підтримує одну або кілька мов програмування та відповідні контейнери програм і SDK. Він також може бути призначений для різних типів додатків, таких як веб-сайти, веб-додатки, управління бізнес-процесами тощо.



Безпека є однією з ключових служб проміжного програмного забезпечення в Cloud PaaS

Включає безпеку взаємодії служб, захист даних і орієнтований на програму контроль доступу, що дозволяє застосовувати різні рівні доступу/дозволів для можливостей різних програм або запитаних дій. Автентифікація та авторизація застосовуються до взаємодії служб (зокрема, на рівні обміну повідомленнями) і доступу користувачів.

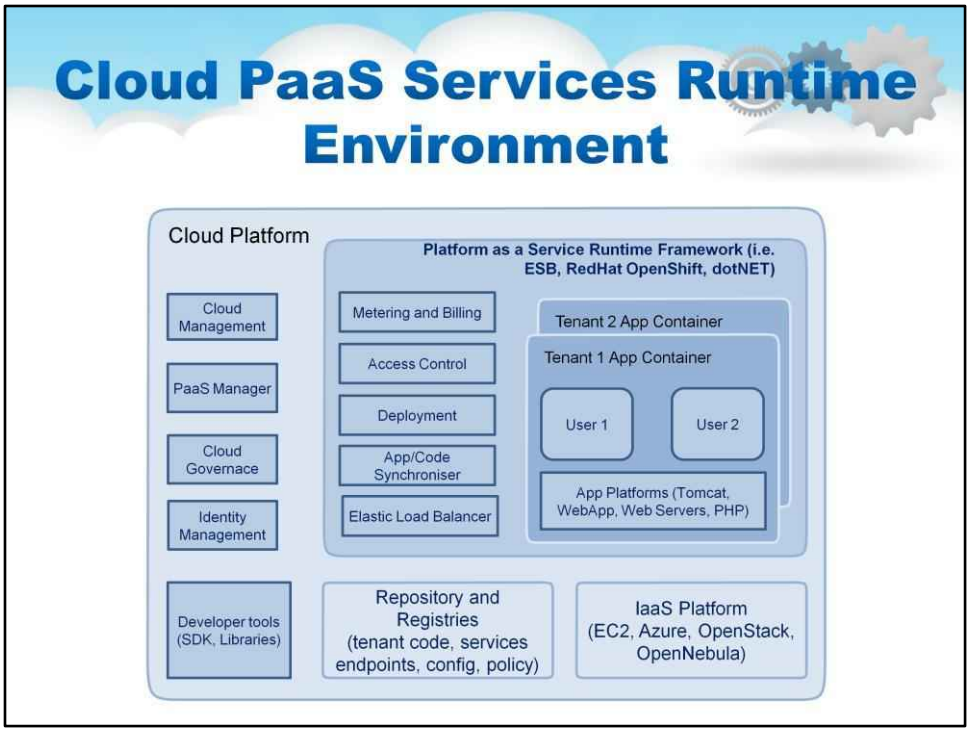
Служба контролю доступу може також включати службу керування ідентифікацією, надану постачальником Cloud PaaS, яку можна використовувати для створення об'єднання ідентифікацій між доменом користувача (корпоративного) та хмарними програмами.

Контроль доступу на рівні PaaS має відповідати основним заходам безпеки та контролю доступу до спільних хмарних ресурсів.

Користувачі та програми не повинні мати доступ до ресурсів або служб, які їм не призначені або рівень дозволу яких вищий за певний права доступу користувача.

Мультиоренда як фактор підвищення безпеки

Платформа Cloud PaaS повинна бути розроблена для підтримки мультиорендування в спільному використанні ресурсів і розділенні доменів безпеки, подібно до платформи Cloud IaaS. Додатки клієнта запускаються в окремих віртуальних машинах, використовуючи також додаткові заходи для розділення даних і процесів користувачів програми.

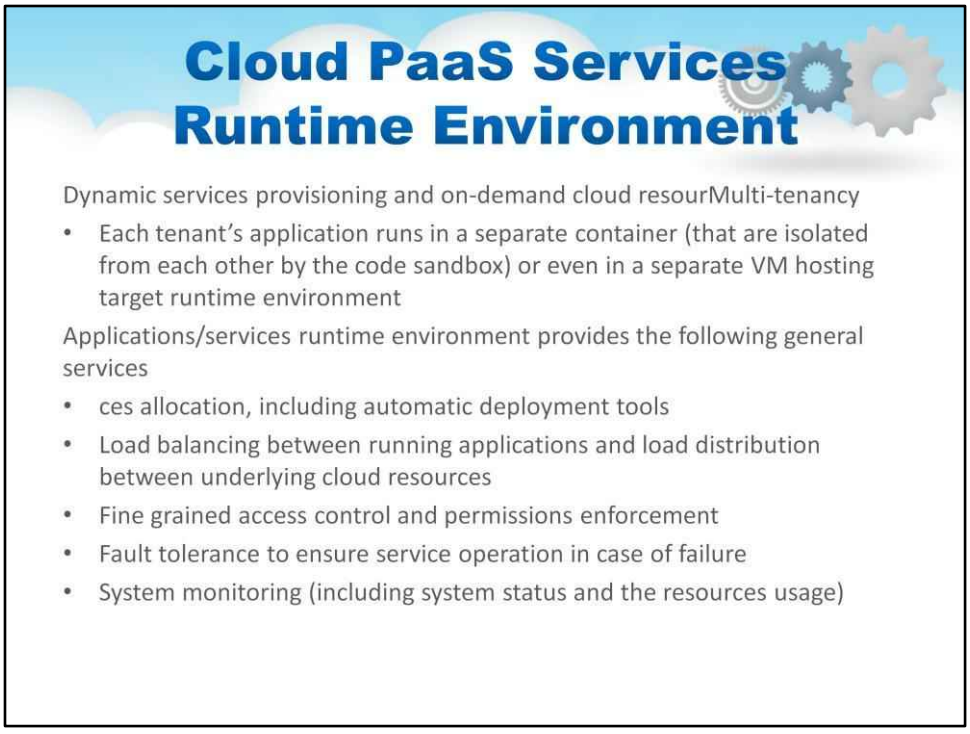


Цей слайд містить ілюстрацію середовища виконання Cloud PaaS Services.

Як видно, в основі лежить платформа IaaS. Існують інші засоби, невидимі для програміста, які забезпечують роботу хмарної платформи, наприклад керування хмарою, управління, ідентифікація тощо,

Крім того, є Платформа як Сервіс Runtime Framework (тобто Cloud Foundry, RedHat OpenShift, dotNET)

Зауважте, що в середовищі виконання PaaS мультитенантний рівень виникає на рівні контейнера програми, а не на рівні віртуальної машини. Немає видимості віртуальних машин для користувачів!



Як згадувалося на попередньому слайді, кожна програма клієнта працює в окремому контейнері (які ізольовані один від одного пісочницею коду) або навіть в окремому цільовому середовищі виконання віртуальної машини

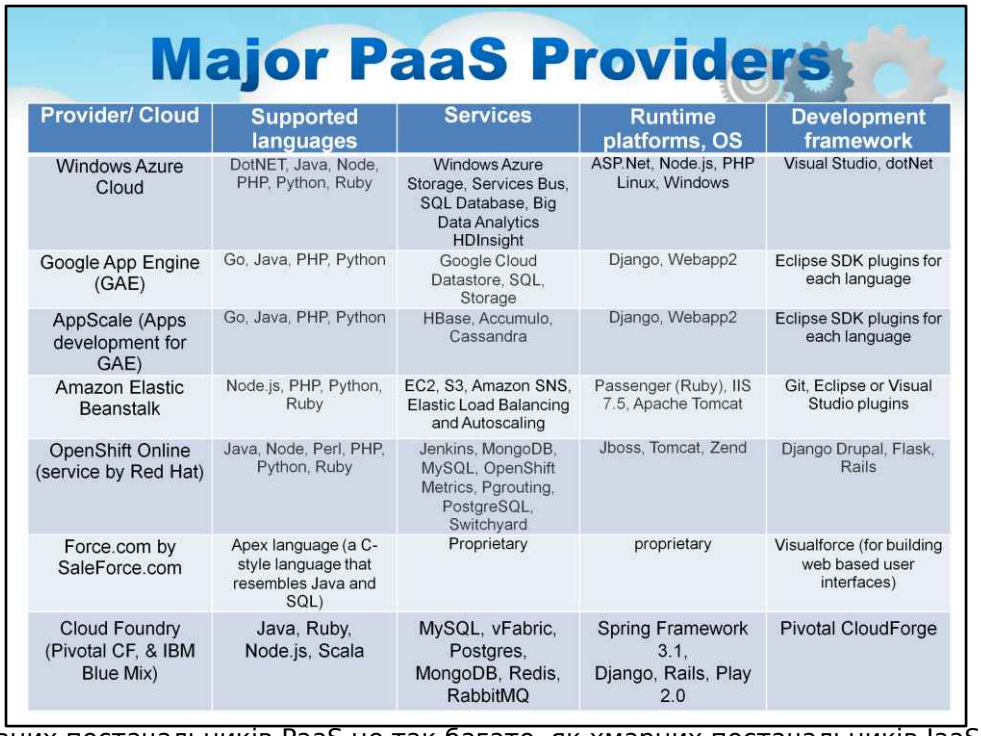
Середовище виконання програм/служб забезпечує такі загальні служби

Динамічне надання послуг, включаючи засоби автоматичного розгортання. Балансування навантаження

Точний контроль доступу

Відмовостійкість

Системний моніторинг



Major PaaS Providers

Provider/ Cloud	Supported languages	Services	Runtime platforms, OS	Development framework
Windows Azure Cloud	DotNET, Java, Node, PHP, Python, Ruby	Windows Azure Storage, Services Bus, SQL Database, Big Data Analytics HDInsight	ASP.Net, Node.js, PHP Linux, Windows	Visual Studio, dotNet
Google App Engine (GAE)	Go, Java, PHP, Python	Google Cloud Datastore, SQL, Storage	Django, Webapp2	Eclipse SDK plugins for each language
AppScale (Apps development for GAE)	Go, Java, PHP, Python	HBase, Accumulo, Cassandra	Django, Webapp2	Eclipse SDK plugins for each language
Amazon Elastic Beanstalk	Node.js, PHP, Python, Ruby	EC2, S3, Amazon SNS, Elastic Load Balancing and Autoscaling	Passenger (Ruby), IIS 7.5, Apache Tomcat	Git, Eclipse or Visual Studio plugins
OpenShift Online (service by Red Hat)	Java, Node, Perl, PHP, Python, Ruby	Jenkins, MongoDB, MySQL, OpenShift Metrics, Pgrouting, PostgreSQL, Switchyard	Jboss, Tomcat, Zend	Django Drupal, Flask, Rails
Force.com by SaleForce.com	Apex language (a C-style language that resembles Java and SQL)	Proprietary	proprietary	Visualforce (for building web based user interfaces)
Cloud Foundry (Pivotal CF, & IBM Blue Mix)	Java, Ruby, Node.js, Scala	MySQL, vFabric, Postgres, MongoDB, Redis, RabbitMQ	Spring Framework 3.1, Django, Rails, Play 2.0	Pivotal CloudForge

Основних постачальників PaaS не так багато, як хмарних постачальників IaaS. Загалом існує не так багато «сімейств» основних середовищ програмування. На найвищому рівні є «.NET (Microsoft)» і «Java (або, загалом кажучи, більш відкриті середовища розробки)».

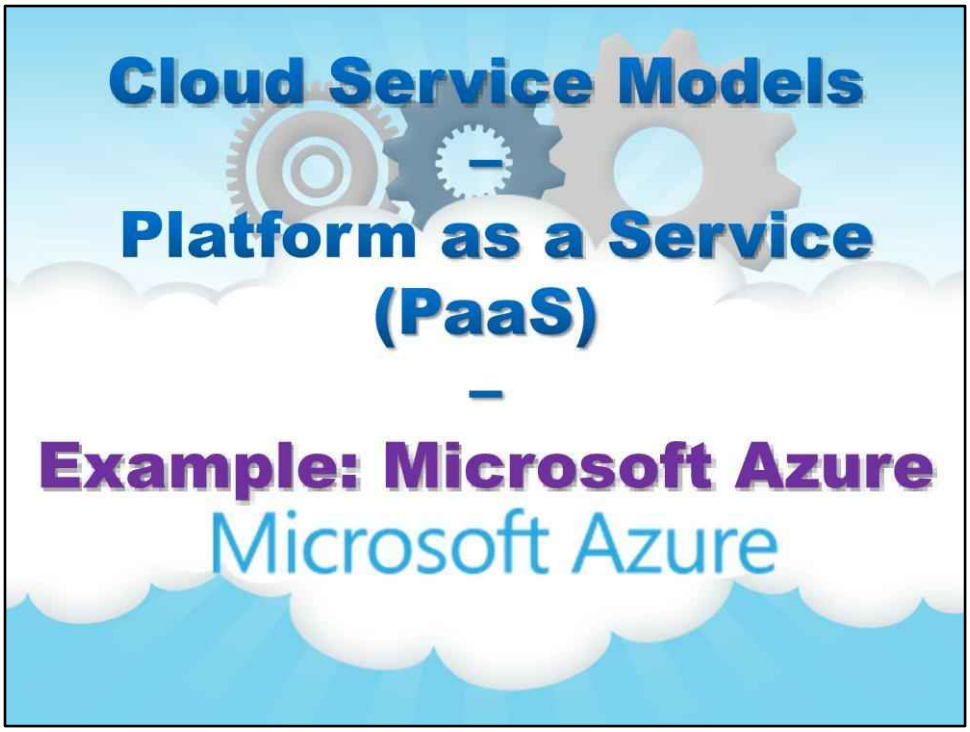
Microsoft була беззаперечним піонером PaaS. Під час початкового запуску Windows Azure взагалі не було режиму IaaS, це була пропозиція PaaS-pure-play. Корпорація Майкрософт дійсно випередила час, вважаючи, що користувачі ніколи не повинні маніпулювати віртуальними машинами, і що вони можуть забезпечити набагато кращу хмару та набагато більше безпеки та можливостей, відкриваючи середовище контейнера з чистим кодом. Незважаючи на те, що Azure додав можливість IaaS, він залишається одним із найбільш добре архітектурних і повних середовищ PaaS. Варто зазначити, що Microsoft має чудову підтримку для багатьох мов, включаючи PHP, Ruby та Java, тому думати, що Azure або .NET PaaS є суто середовищем C#, є абсолютно неправильним.

Google також випустив PaaS до того, як вони підтримали пропозицію IaaS, GAE була дуже простою.

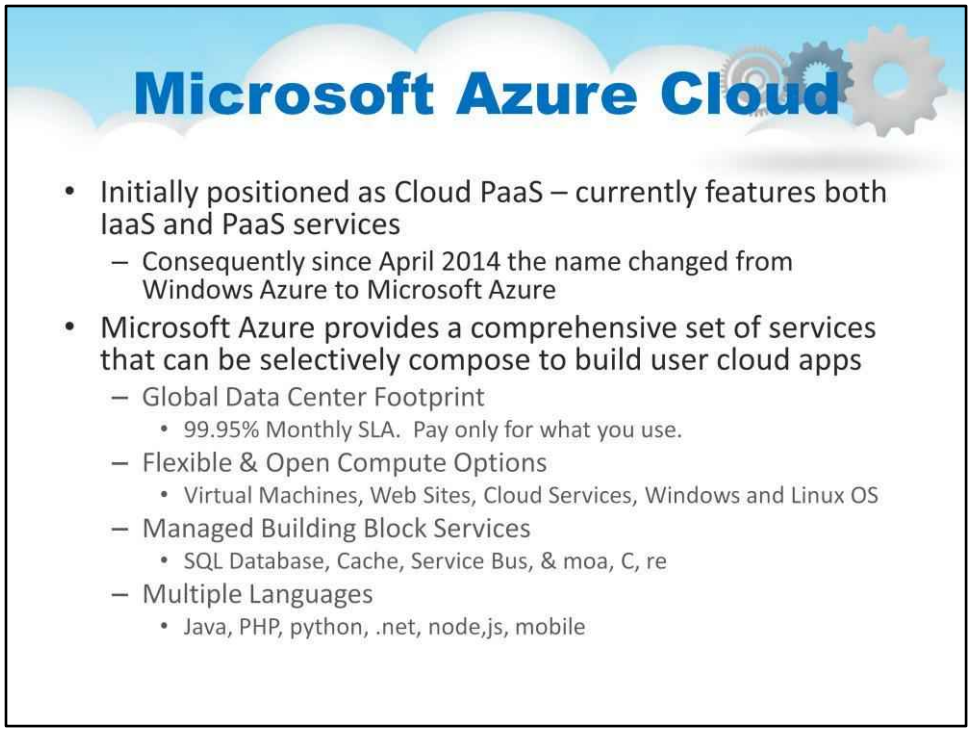
Хоча люди стверджують, що AWS насправді не є PaaS, це успадковане сприйняття, оскільки воно містить усілякі можливості для запуску коду, як зазначено в таблиці

RedHat об’єднав/повторно адаптував свої пропозиції проміжного програмного забезпечення до OpenShift, але це було менш сприйнято спільнотою, ніж Cloud Foundry (див. нижче), оскільки воно, здається, сильно контролюється RedHat,

Salesforce,com має середовище «налаштування та розширення» для своїх програм CRM і бази даних, яке називається Force,com.



Приклад: Microsoft Azure



The Microsoft Azure Cloud was first released as Windows Azure and was strictly a PaaS platform aimed at providing a Cloud home to Windows developers. After adding IaaS abilities including the ability to run Linux VM's, Azure has become a hue force in Cloud.

Microsoft Azure provides a comprehensive set of services that can be selectively compose to build user cloud apps

Глобальний центр обробки даних

99,95% щомісячного SLA. Платіть лише за те, що використовуєте.

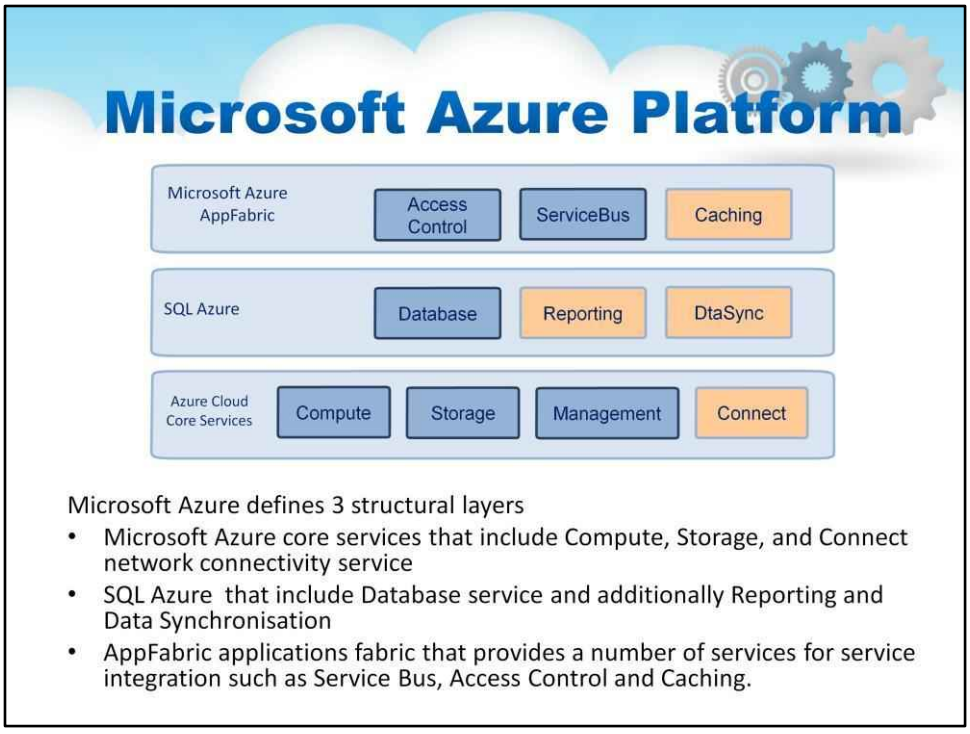
Гнучкі та відкриті параметри обчислень

Віртуальні машини, веб-сайти, хмарні служби, служби керованих будівельних блоків ОС Windows і Linux

SQL Database, Cache, Service Bus, & moa, C, re

Multiple Languages

Java, PHP, python, .net, node, js, mobile



Платформа Azure за своєю структурою майже схожа на інші хмарні платформиза винятком того, що служби даних і додатків дуже тісно інтегровані з основними хмарними службами, що лежать в основі.

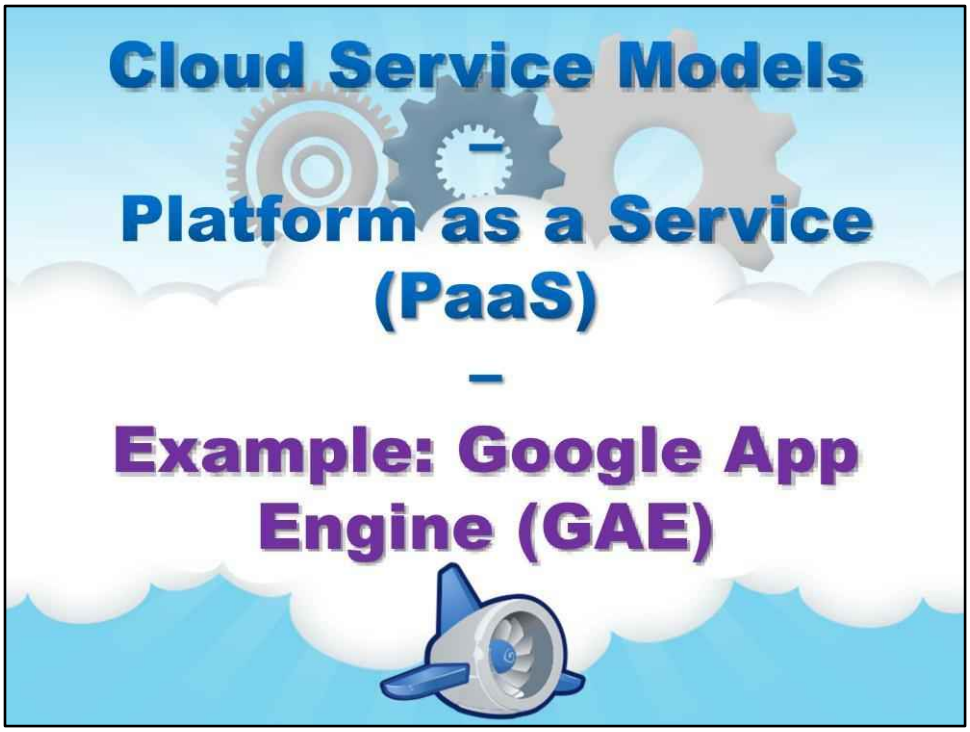
Це артефакт того, як Azure створювало чисте середовище PaaS на початку

Слайд ілюструє структурні рівні Microsoft Azure 3

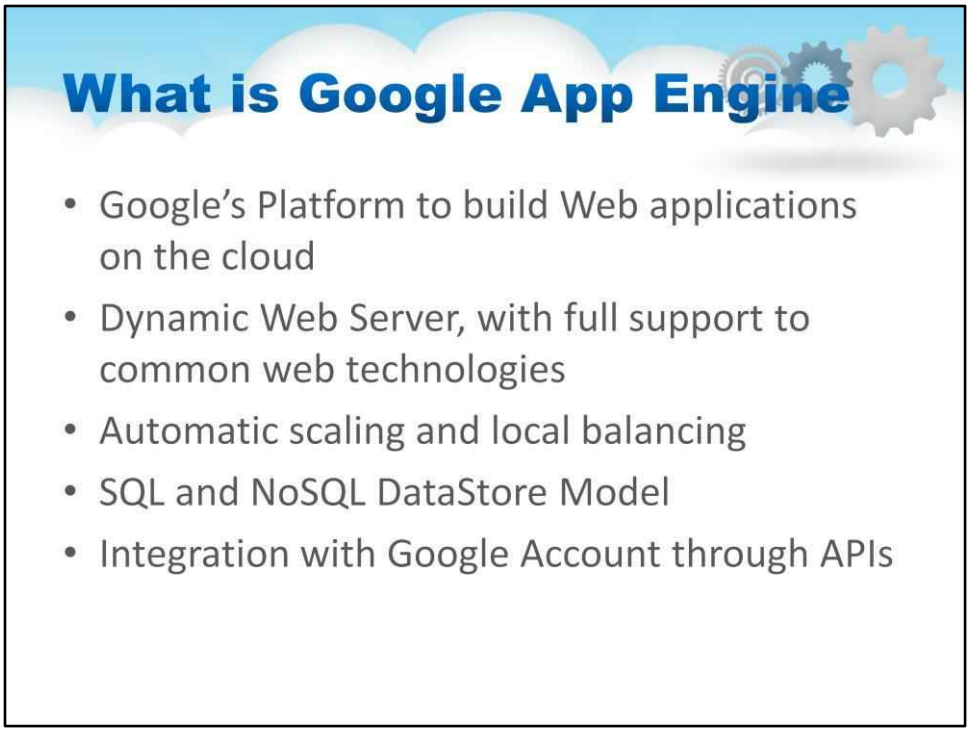
Основні служби Microsoft Azure, які включають службу підключення до мережі Compute, Storage та Connect

SQL Azure, що включає службу бази даних, а також звітування та синхронізацію даних

Структура додатків AppFabric, яка надає низку служб для інтеграції послуг, таких як шина обслуговування, контроль доступу та кешування.



Приклад: Google App Engine (GAE)



Google App Engine Google App Engine (часто називають GAE або просто App Engine) — це платформа хмарних обчислень (PaaS) для розробки та розміщення веб-додатків у центрах обробки даних, якими керує Google.

Програми знаходяться в ізольованому програмному середовищі та працюють на кількох серверах. додаток Engine пропонує автоматичне масштабування для веб-додатків — коли кількість запитів до програми збільшується, App Engine автоматично виділяє більше ресурсів для веб-додатків, щоб впоратися з додатковим попитом. Google App Engine є безкоштовним до певного рівня споживаних ресурсів. Плата стягується за додаткове сховище, пропускну здатність або години екземплярів, необхідні програмі.

Вперше він був випущений як попередній перегляд у квітні 2008 року та вийшов попереднього перегляду у вересні 2011 року.

Автоматичне масштабування - немає потреби в надлишковому забезпеченні

Статичні файли -Статичні файли використовують CDN Google

Easy Logs - перегляд журналів у веб-консолі

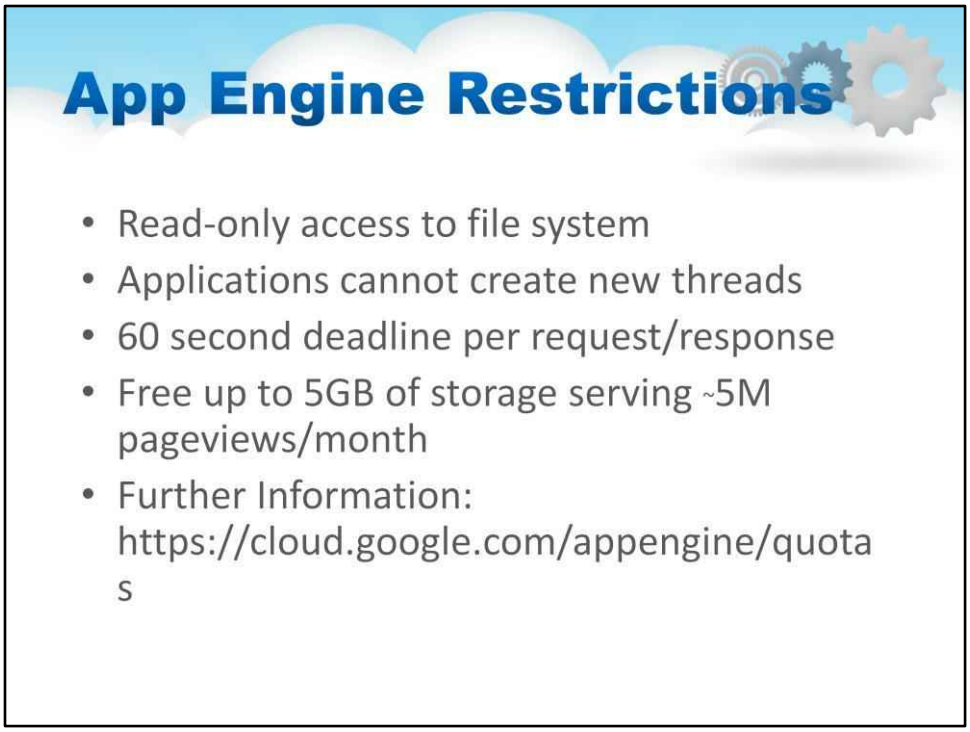
Просте розгортання - розгортання буквально в 1 клік

Безкоштовна квота - 99% додатків не платять нічого

Доступне масштабування – Google App Engine недешеве, але має дуже конкурентоспроможну структуру витрат, враховуючи масштабованість і простоту використання.

Немає конфігурації - немає необхідності налаштовувати ОС або сервери

Проста безпека -Вся інфраструктура App Engine значно відрізняється від «звичайної» хостингові платформи. Додаток працює в невеликій пісочниці з дуже обмеженими можливостями дозволи. Оскільки програма не може писати вфайлова система,це неможливо змінити код програми з самої програми. Це запобігає більшості цілеспрямованих атак WordPress.



Програма App Engine може споживати ресурси до певних квот.

Ви можете переглянути щоденне використання та споживання квоти App Engine ресурсів для вашого проекту на сторінці подробиць квоти Google Cloud Platform Console.

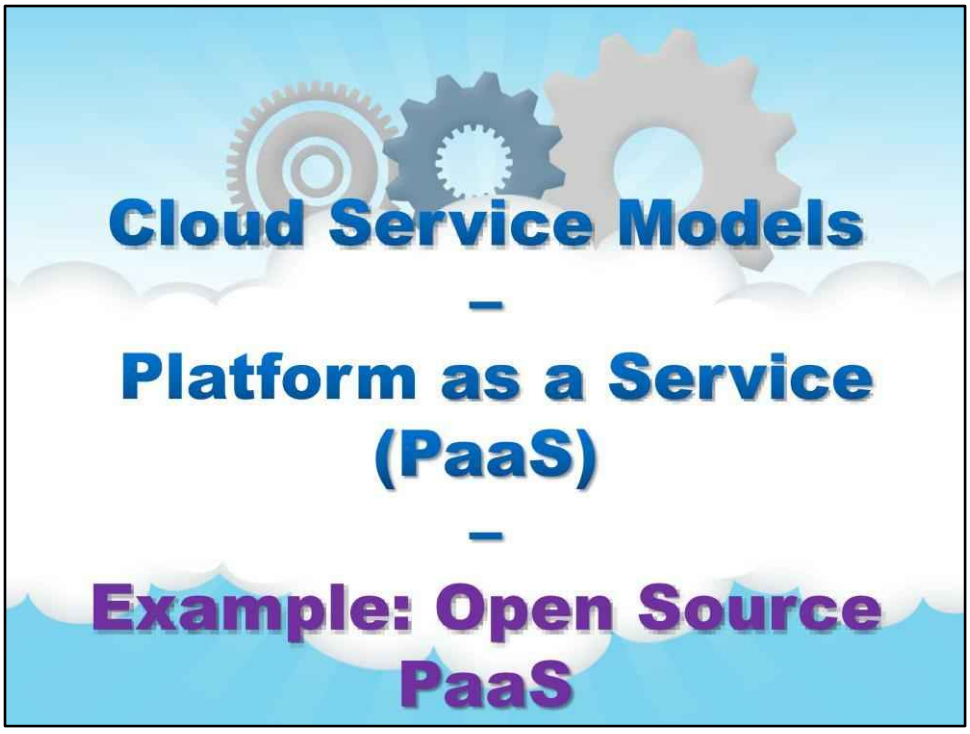
Оплачувані ліміти та безпечні ліміти

App Engine має три типи квот або обмежень:

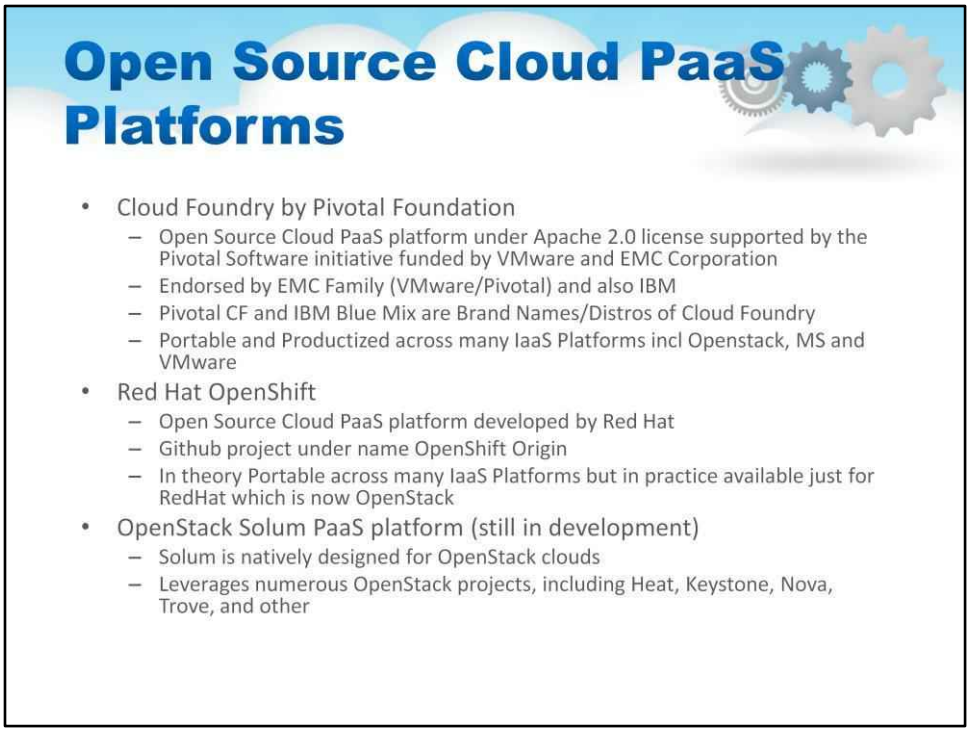
Безкоштовні квоти: Кожна програма отримує суму кожного ресурсу для безкоштовно. Безкоштовні квоти можуть бути перевищені лише платними програмами, до ліміту витрат програми або безпечного ліміту, залежно від того, що настане раніше.

Обмеження витрат: Якщо ви є власником проекту та адміністратором із виставлення рахунків, ви можете встановити ліміт витрат, щоб керувати витратами на програми в Google Cloud Platform Console у налаштуваннях App Engine. Ліміти витрат можуть бути трохи перевищені, оскільки програму вимкнено.

Межі безпеки: Обмеження безпеки встановлює Google для захисту цілісності система App Engine. Ці квоти гарантують, що жодна програма не може надмірно споживати ресурси на шкоду іншим програмам. Якщо ви перевищите ці ліміти, ви отримаєте повідомлення про те, платно чи безкоштовно.



Приклад: PaaS з відкритим кодом



Триває «битва» щодо того, яка платформа PaaS з відкритим вихідним кодом буде найбільш прийнятною

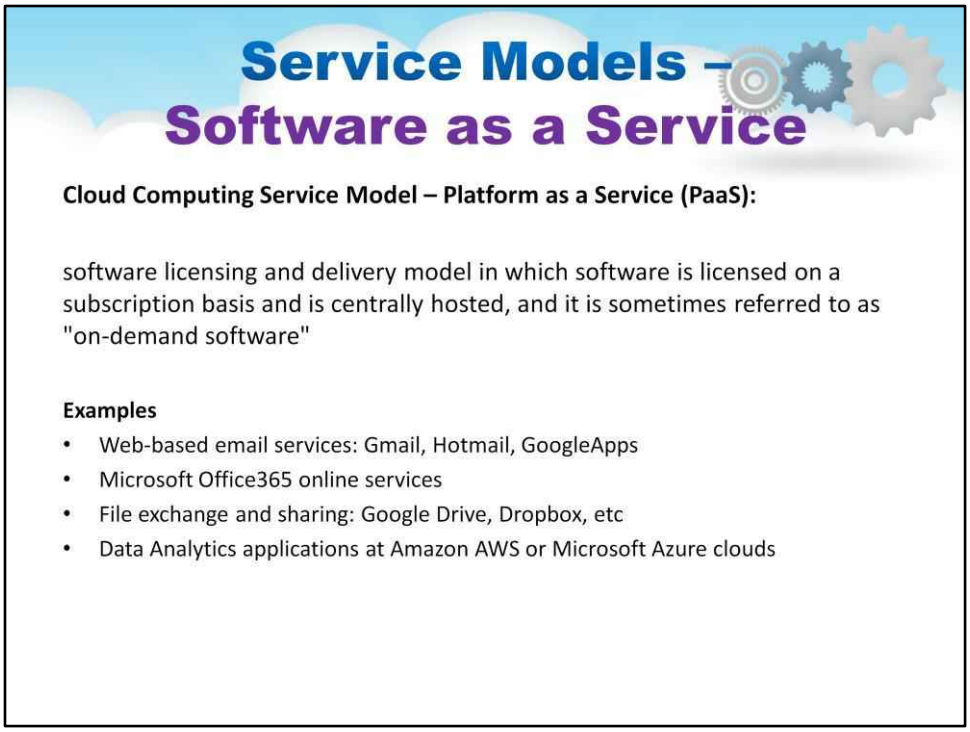
Cloud Foundryспочатку був зібраний із Spring і Hibernate, і було додано кілька компонентів, розроблених як у Open Source, так і VMware, і створених VMware/EMC Pivotal. Pivotal запустив фонд, створений VMware і EMC з початковою підтримкою IBM. Pivotal CF і IBM Bluemix є брендами/ дистрибутивами Cloud

Ливарний цех. Одразу Pivotal Foudation отримав більше підтримку з боку бізнес-орієнтованих IT-компаній, таких як IBM, HP і SAP. Схоже, Pivotal стає «новим J2EE»

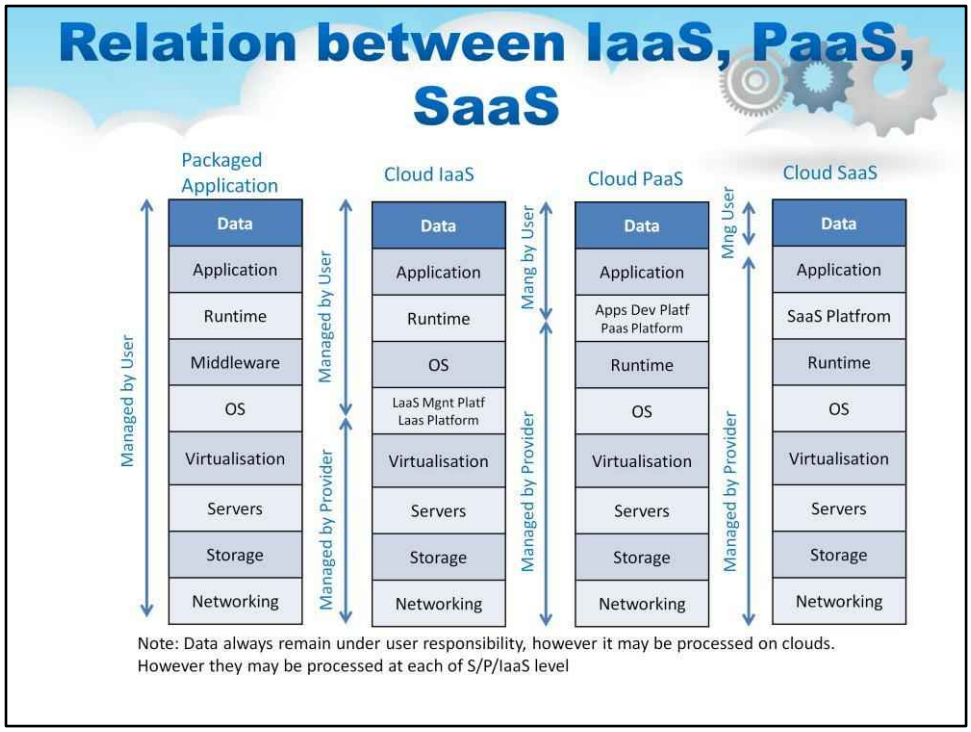
Червоний капелюхоб’єднав OpenShift, який походить від його великої лінії продуктів проміжного програмного забезпечення (усі з відкритим кодом). Набираючи початкових обертів, OpenShift ставить RedHat у протиріччя зі своїм зазвичай найбільшим партнером IBM. Майбутнє OpenShift під питанням.

OpenStackSolum ще в розробці.

Програмне забезпечення як послуга (SaaS)



Можливість, яка надається споживачеві, полягає в тому, щобвикористовувати запущені програми провайдера на хмарній інфраструктурі. Програми доступні з різних клієнтських пристроїв або через тонкий клієнтський інтерфейс, такий як веб-браузер (наприклад, веб-електронна пошта), або програмний інтерфейс. Споживач не управляє і не контролює базовий актив хмарна інфраструктура, включаючи мережу, сервери, операційні системи, сховища або навіть можливості окремих програм, за можливим винятком обмежена залежність від користувача налаштування конфігурації програми



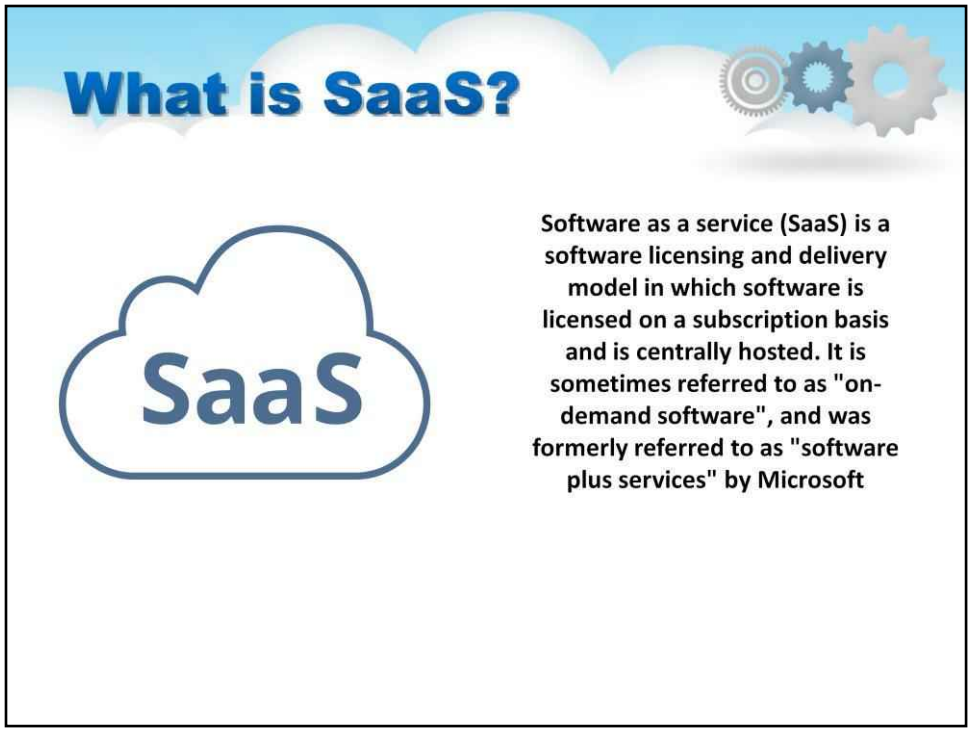
Моделі використання хмарних обчислень тісно пов’язані. Цю ілюстрацію було розглянуто в попередньому уроці, але її потрібно повторити.

Для традиційної «фізично встановленої» упакованої програми — усе розгортання стеком керує користувач, як видно на першому блоці ілюстрації. Користувач несе відповідальність за надання повного сервера, включаючи мережу та сховище, а також програмне забезпечення операційної системи. Крім того, користувач надає необхідні бази даних і проміжне програмне забезпечення, будь-яке спеціальне середовище виконання (Java або Ruby) і, нарешті, програму. Дані програми також є відповідальністю (і власністю) користувача.

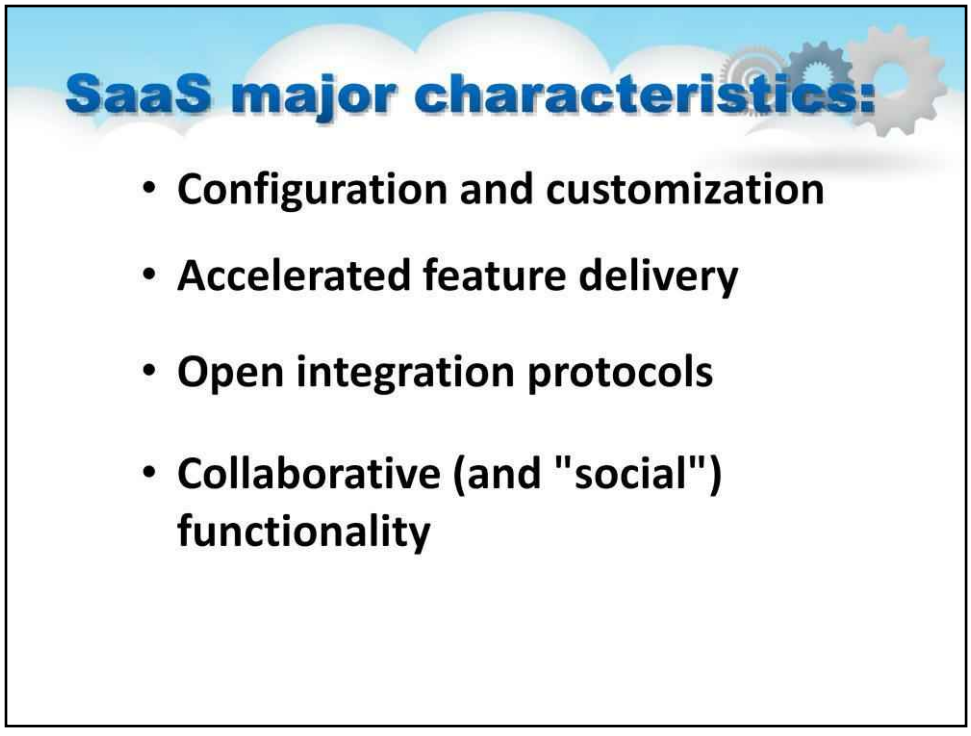
Модель Cloud IaaS суттєво змінює це, полегшуючи принаймні значну частину те, що вважатиметься «фізичною» інфраструктурою, показано у другому блоці ілюстрації. Програмне забезпечення для керування хмарою (іноді її називають Cloud OS) є показано на цій діаграмі як "IaaS Mngt Platf”.Користувач отримує обладнання в а віртуалізовану форму від IaaS CSP і турбується лише про програмний стек, програму та дані.

Модель Cloud PaaS також пропонує проміжне програмне забезпечення, час виконання програми тощо «програмної» інфраструктури, створюючи зручний контейнер для розробник додатків. Хоча це вимагає від розробника реструктуризації, якщо не переписування програми на рівні вихідного коду для інтерфейсу з PaaS, це обмеження додасть нові переваги масштабованості та доступності додатку.

Модель Cloud IaaS — це використання лише самої програми через браузер або веб-сервіс.



Програмне забезпечення як послуга (SaaS)це модель ліцензування та доставки програмного забезпечення, згідно з якою програмне забезпечення ліцензується на основі передплати та розміщується централізовано. Його іноді називають «програмним забезпеченням на вимогу», а раніше Microsoft називало «програмне забезпечення плюс служби». Користувачі, як правило, отримують доступ до SaaS за допомогою тонкого клієнта через веб-браузер. SaaS став загальноприйнятою моделлю доставки для багатьох бізнес-додатків, включаючи офісне програмне забезпечення, програмне забезпечення для обміну повідомленнями, програмне забезпечення для обробки заробітної плати, програмне забезпечення СУБД, програмне забезпечення для керування, програмне забезпечення САПР, програмне забезпечення для розробки, гейміфікацію, віртуалізацію, облік, співпрацю, управління взаємовідносинами з клієнтами (CRM), Інформаційні системи управління (MIS), планування ресурсів підприємства (ERP), виставлення рахунків, управління людськими ресурсами (HRM), залучення талантів, системи управління навчанням, управління контентом (CM), і управління службою обслуговування. SaaS включено в стратегію майже всіх провідних компаній, що займаються корпоративним програмним забезпеченням.



Конфігурація та налаштування

Програми SaaS так само підтримують те, що традиційно називають конфігурацією програми.

Прискорене надання функцій

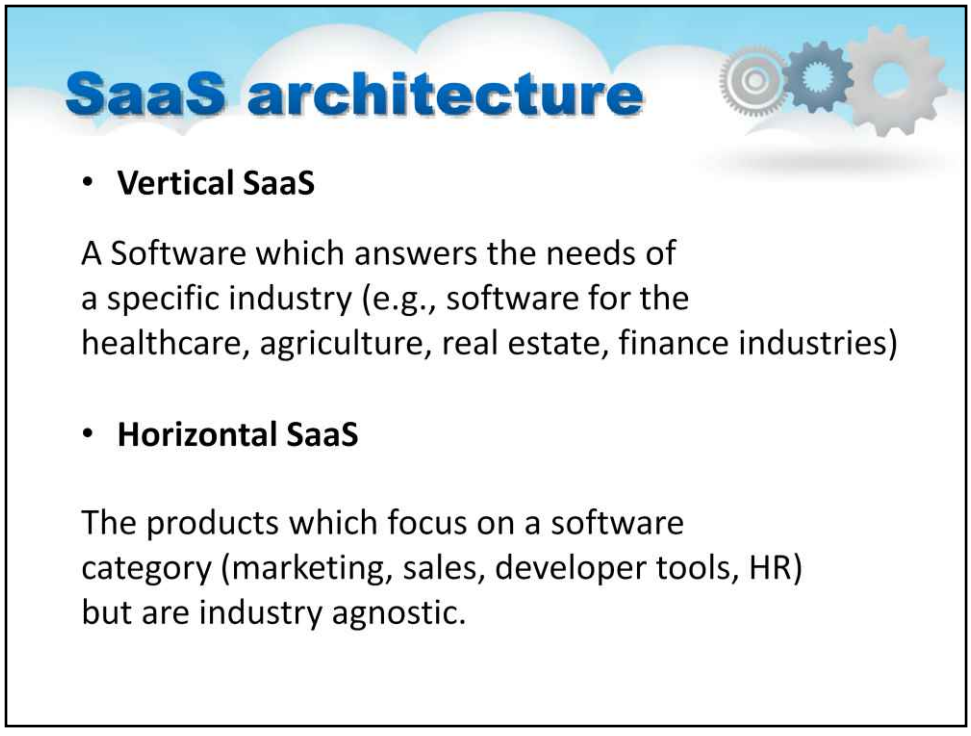
Програми SaaS часто оновлюються частіше, ніж традиційне програмне забезпечення, у багатьох випадках щотижня або щомісяця

Відкриті протоколи інтеграції

Оскільки програми SaaS не мають доступу до внутрішніх систем компанії (баз даних або внутрішніх служб), вони переважно пропонують протоколи інтеграції та інтерфейси прикладного програмування (API), які працюють у глобальній мережі. Як правило, це протоколи на основі HTTP, REST і SOAP.

Спільна (і «соціальна») функціональність

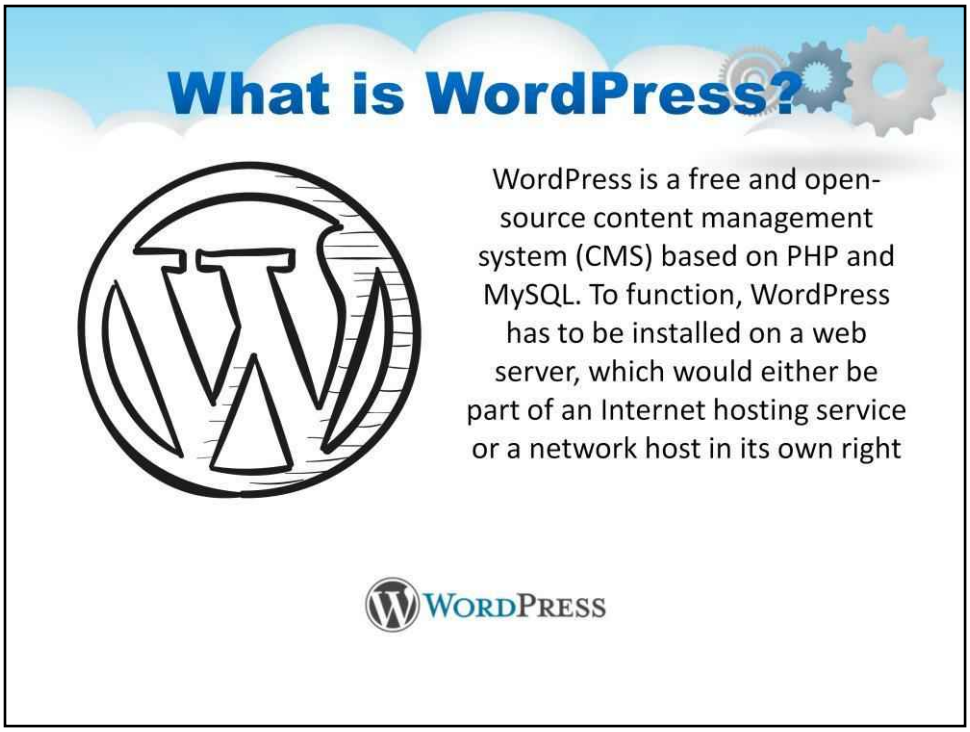
Натхненні успіхом онлайн-соціальних мереж та інших так званих функцій Web 2.0, багато програм SaaS пропонують функції, які дозволяють користувачам співпрацювати та обмінюватися інформацією.



Переважна більшість рішень SaaS базується на мультитенантній архітектурі. У цій моделі єдина версія програми з єдиною конфігурацією (апаратне забезпечення, мережа, операційна система) використовується для всіх клієнтів («орендарів»). Для підтримки масштабованості програму встановлюють на кількох машинах (так зване горизонтальне масштабування). У деяких випадках друга версія програми налаштована, щоб запропонувати вибраній групі клієнтів доступ до попередніх версій програм (наприклад, бета-версії) з метою тестування. Це відрізняється від традиційного програмного забезпечення, де кілька фізичних копій програмного забезпечення — кожна потенційно має іншу версію, з потенційно іншою конфігурацією та часто налаштовано — встановлюються на різних сайтах клієнтів. У цій традиційній моделі

Є два основних різновиди SaaS (поточний слайд)

Приклад: WordPress



WordPress — це безкоштовна система керування вмістом (CMS) із відкритим вихідним кодом на основі PHP і MySQL. Щоб функціонувати, WordPress має бути встановлено на веб-сервері, який буде або частиною служби Інтернет-хостингу, або власним мережевим хостом. Прикладом першого сценарію може бути такий сервіс, як WordPress.com, а другим випадком може бути комп’ютер із пакетом програмного забезпечення WordPress.org. Локальний комп’ютер можна використовувати для тестування та навчання для одного користувача. Особливості включають архітектуру плагінів і систему шаблонів. Станом на квітень 2018 року WordPress використовують 30,6% із 10 мільйонів найкращих веб-сайтів. Таким чином, WordPress є найпопулярнішою системою керування веб-сайтами або блогами, що використовується в Інтернеті, яка підтримує понад 60 мільйонів веб-сайтів. WordPress також використовувався для інших областей додатків, таких як pervasive display systems (PDS).

Простота

Простота дає змогу швидко вийти в Інтернет і опублікувати. Ніщо не повинно заважати вам створити свій веб-сайт і опублікувати вміст.

Гнучкість

За допомогою WordPress ви можете створити веб-сайт будь-якого типу: особистий блог або веб-сайт, фотоблог, бізнес-сайт, професійне портфоліо, державний веб-сайт, журнал або веб-сайт новин, онлайн-спільнота, навіть мережа веб-сайтів. Ви можете зробити свій веб-сайт красивим за допомогою тем і розширити його за допомогою плагінів. Ви навіть можете створити власну програму.

Публікуйте з легкістю

Якщо ви коли-небудь створювали документ, ви вже є фахівцем у створенні вмісту за допомогою WordPress. Ви можете створювати дописи та сторінки, легко їх форматувати, вставляти медіафайли, і одним натисканням кнопки ваш вміст б опубліковано в мережі.

Інструменти публікації

WordPress полегшує вам керування вмістом. Створюйте чернетки, плануйте публікацію та переглядайте редакції своїх публікацій. Зробіть свій вміст загальнодоступним або приватним і захистіть публікації та сторінки паролем

Керування користувачами

Не всім потрібен однаковий доступ до вашого веб-сайту. Адміністратори керують сайтом, редактори працюють із контентом, автори та учасники пишуть цей контент, а передплатники мають профіль, яким вони можуть керувати Це дає змогу мати різноманітних співавторів на вашому веб-сайті, а іншим просто бути частиною вашої спільноти

Медіа менеджмент

Вони кажуть, що зображення говорить тисячу слів, тому для вас важливо мати можливість швидко та легко завантажувати зображення та медіа до WordPress. Перетягніть медіафайли в засіб завантаження, щоб додати їх на свій веб-сайт. Додайте альтернативний текст, підписи та заголовки, а також вставте зображення та галереї у свій вміст. Ми навіть додали кілька інструментів для редагування зображень, якими ви можете весело провести час.

Повна відповідність стандартам

Кожна частина створеного WordPress коду повністю відповідає стандартам, установленим W3C. Це означа що ваш веб-сайт працюватиме в сучасному веб-переглядачі, зберігаючи сумісність із веб-переглядачами нового покоління. Ваш веб-сайт — це прекрасна річ зараз і в майбутньому.

Проста система тем

WordPress постачається в комплекті з двома темами за замовчуванням, але якщо вони вам не підходять, є катало тем із тисячами тем, за якими ви зможете створити чудовий веб-сайт. Жоден із них не до смаку? Завантажте вла тему одним натисканням кнопки. Вам знадобиться лише кілька секунд, щоб повністю оновити свій веб-сайт.

Розширення за допомогою плагінів

WordPress пропонує повну кількість функцій для кожного користувача, для кожної іншої функції є каталог із тисячами плагінів. Додайте складні галереї, соціальні мережі, форуми, віджети соціальних мереж, захист від спаму, календарі, налаштуйте елементи керування для оптимізації пошукової системи та форми. **Вбудовані коментарі**

Ваш блог – це ваш дім, а коментарі надають вашим друзям і підписникам простір для взаємодії з вашим вмістом. Інструменти для коментарів WordPress надають вам все, що вам потрібно, щоб стати форумом для обговорення та модерувати це обговорення.

Оптимізовано для пошукових систем

WordPress оптимізований для пошукових систем одразу після виходу з коробки. Для більш точного контролю SEO є багато плагінів SEO, які подбають про це за вас.

Багатомовний

WordPress доступний понад 70 мовами. Якщо ви або особа, для якої ви створюєте веб-сайт, бажаєте використовувати WordPress не англійською мовою, це легко зробити.

Легке встановлення та оновлення

WordPress завжди легко встановлювати та оновлювати. Якщо вам подобається програма FTP, ви можете створити базу даних, завантажити WordPress за допомогою FTP і запустити інсталятор.

Володійте своїми даними

Розміщені послуги приходять і йдуть. Якщо ви коли-небудь користувалися послугою, яка зникла, ви знаєте, наскільки це може бут травматично. Якщо ви коли-небудь бачили рекламу, яка з’являється на вашому веб-сайті, ви, ймовірно, були дуже роздратовані. Використання WordPress означає, що ніхто не має доступу до вашого вмісту. Володійте всіма своїми даними — веб-сайтом, вмістом даними.

Свобода

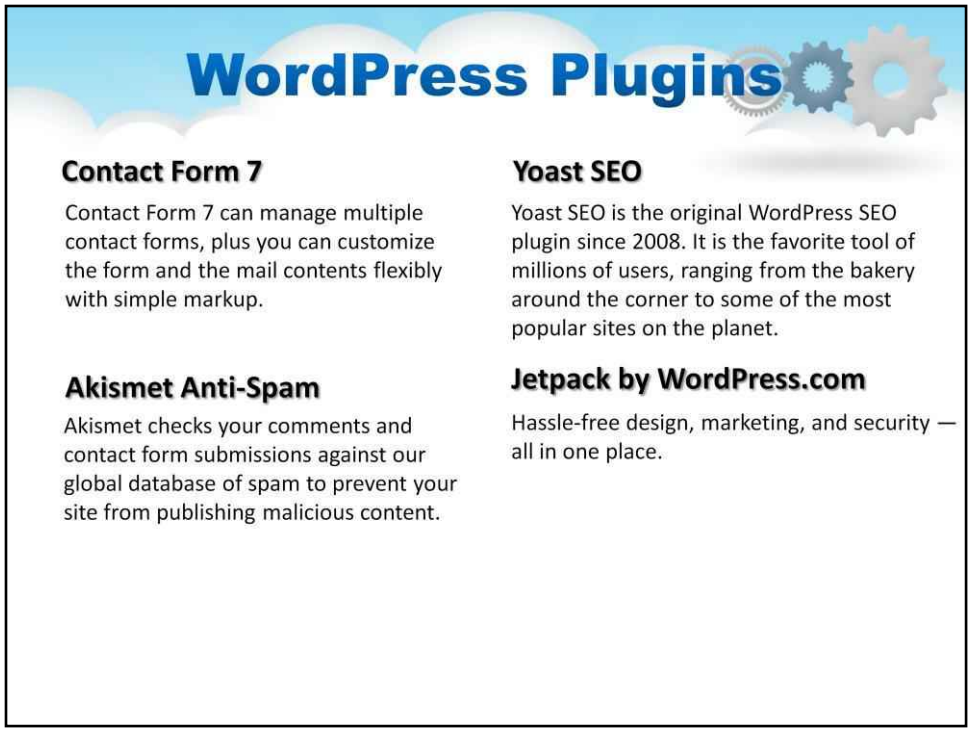
WordPress має ліцензію GPL, яка була створена для захисту ваших свобод. Ви можете вільно використовувати WordPres будь-який спосіб: встановлювати, використовувати, змінювати, поширювати. Свобода програмного забезпечення — основа, на якій побудований WordPress.

Спільнота

Будучи найпопулярнішою CMS з відкритим кодом в Інтернеті, WordPress має активну спільноту, яка підтримує її. Постав запитання на форумах підтримки та отримайте допомогу від волонтера, відвідайте WordCamp або Meetup, щоб дізнати більше про WordPress, прочитайте публікації в блогах і навчальні посібники про WordPress. Спільнота є серцевиною WordPress, що робить його таким, яким він є сьогодні.

Внести свій внесок

Ви також можете бути WordPress! Допомагайте створювати WordPress, відповідайте на запитання на форумах підтримки, пишіть документацію, перекладайте WordPress на вашу мову, виступайте на WordCamp, пишіть про WordPress у своєму блозі. Незалежно від ваших навичок, ми будемо раді вам!



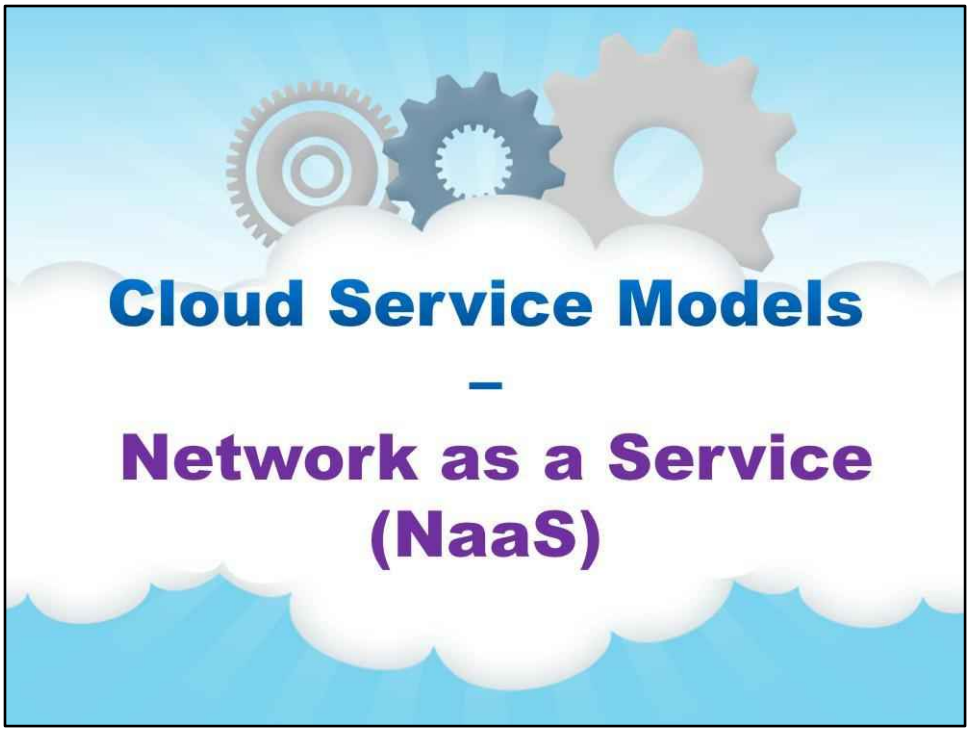
Контактна форма 7можна керувати кількома контактними формами, а також ви можете гнучко налаштовувати форму та вміст електронної пошти за допомогою простої розмітки. Форма підтримує надсилання на базі Ajax, CAPTCHA, фільтрацію спаму Akismet тощо.

Yoast SEO улюбленим інструментом мільйонів користувачів, від пекарні за рогом до деяких із найпопулярніших сайтів на планеті. Завдяки Yoast SEO ви отримуєте надійний набір інструментів, як допоможе вам посісти перше місце в результатах пошуку. Yoast: SEO для всіх. Yoast SEO робить усе, його силах, щоб догодити як відвідувачам, так і павукам пошукових систем.

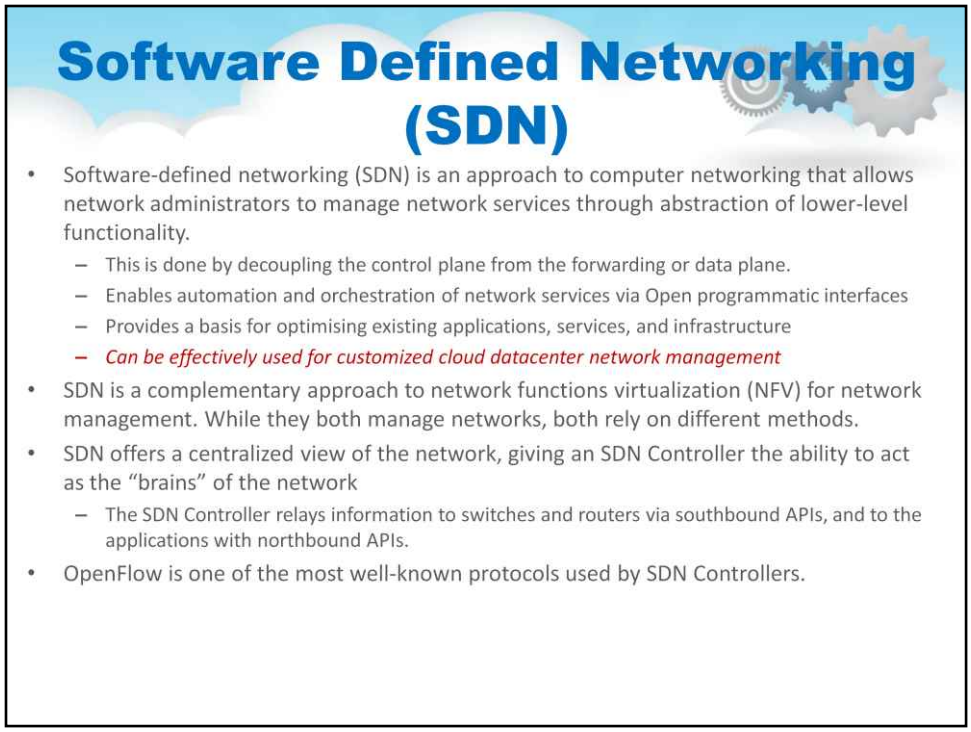
Акісметперевіряє ваші коментарі та надіслані контактні форми з нашою глобальною базою спаму, щоб запобігти публікації шкідливого вмісту на вашому сайті. Ви можете переглянути спам у коментарях, який уловлює, на екрані адміністратора «Коментарі» вашого блогу. Основні функції Akismet: Автоматично перевіряє всі коментарі та відфільтровує ті, які виглядають як спам. Кожен коментар має історію статусу, ви можете легко побачити, які коментарі були перехоплені або видалені Akismet, а які модератор відправ спамом або не спамом. URL-адреси відображаються в тексті коментаря, щоб виявити приховані чи оманли посилання. Модератори можуть бачити кількість схвалених коментарів для кожного користувача. Функція відхилення, яка повністю блокує найгірший спам, заощаджуючи місце на диску та пришвидшуючи роботу сайту.

Jetpack від WordPress.com

Безпроблемний дизайн, маркетинг і безпека — все в одному місці. Створіть і налаштуйте свій сайт WordPress від початку до кінця. Jetpack допоможе вам із: сотнями професійних тем для будь-якого сайту, інтуїтивно зрозуміли та потужними інструментами налаштування, необмеженою та високошвидкісною мережею доставки зображень відеовмісту, відкладеним завантаженням зображень для швидшого мобільного досвіду, інтеграцією з офіційним мобільними програмами WordPress



Мережа як послуга (NaaS)



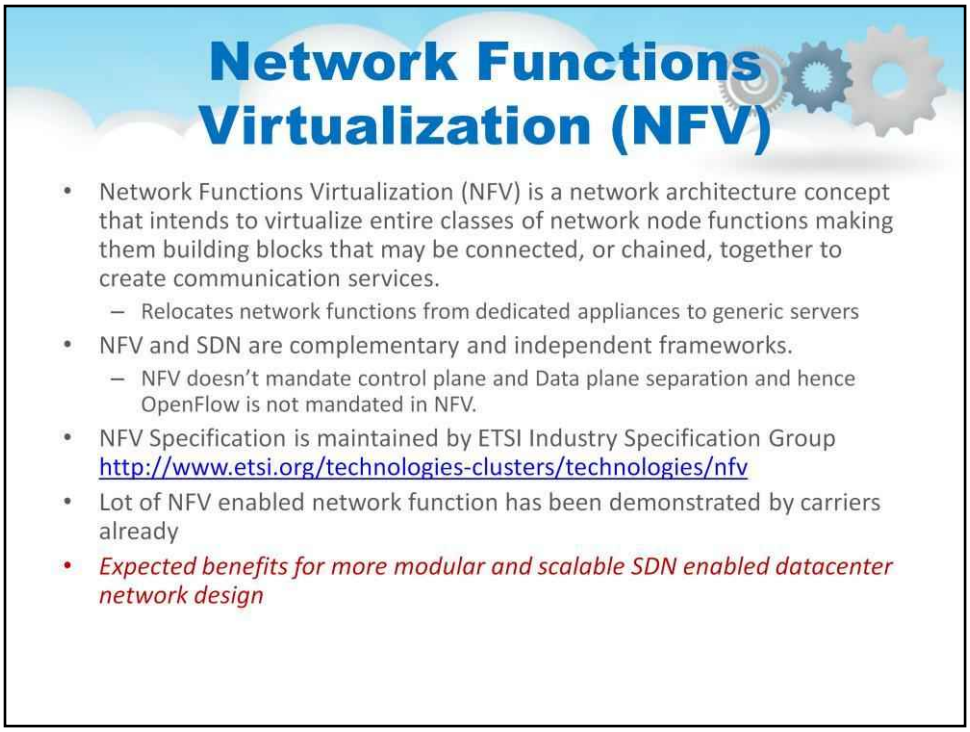
Програмно визначена мережа (SDN)

Програмно-визначена мережа (SDN) — це підхід до комп'ютерної мережі, який дозволяє мережевим адміністраторам керувати мережевими службами через абстракцію функціональності нижчого рівня.

Це робиться шляхом відокремлення площини керування від площини пересилання або даних. Дозволяє автоматизувати та оркеструвати мережеві служби через відкриті програмні інтерфейси
Забезпечує основу для оптимізації існуючих програм, служб та інфраструктури

SDN — це додатковий підхід до віртуалізації мережевих функцій (NFV) для керування мережею. Хоча вони обидва керують мережами, обидва покладаються на різні методи.

SDN пропонує централізований перегляд мережі, надаючи контролеру SDN можливість виступають «мозком» мережі



Віртуалізація мережевих функцій (NFV)

Віртуалізація мережевих функцій (NFV) — це концепція мережевої архітектури, яка має на меті віртуалізувати цілі класи функцій мережевих вузлів, роблячи їх будівельними блоками, які можна з'єднувати або об'єднувати разом для створення комунікаційних послуг.

Переміщує мережеві функції з виділених пристроїв на загальні сервери

Подумайте про брандмауери, балансувальники навантаження, оптимізацію WAN тощо.

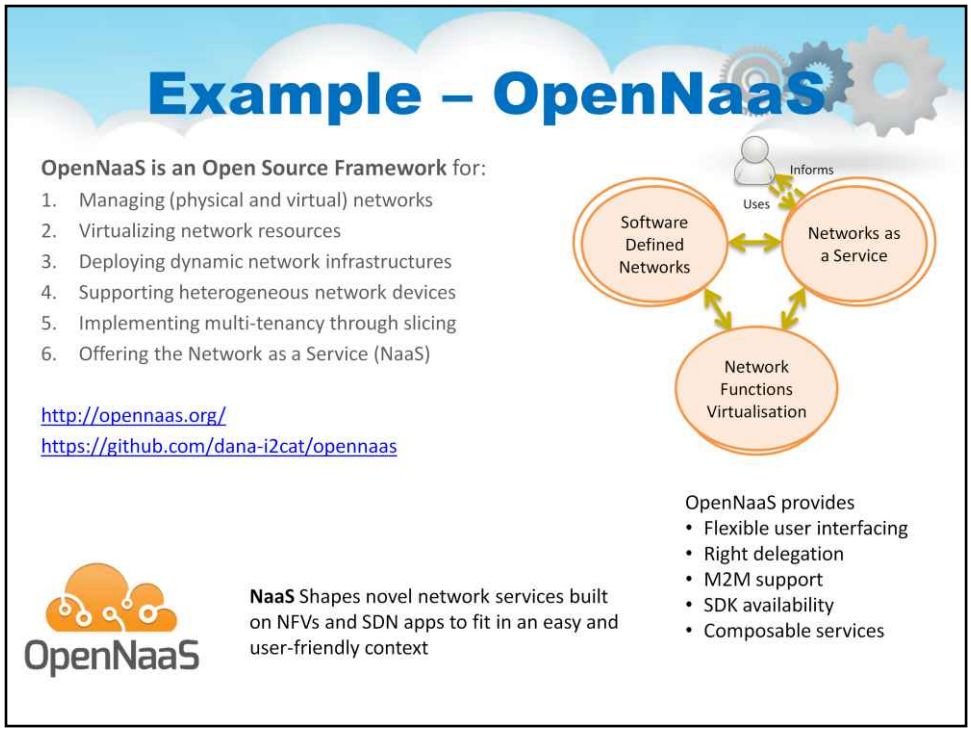
NFV також включає телекомунікаційні функції, такі як IP/PBX, SMS/MMS, IP/TV та інші функції Telco, які зазвичай є на спеціальних пристроях

NFV і SDN є взаємодоповнювальними та незалежними структурами.

Специфікація NFV підтримується ETSI Industry Specification Group

Багато телекомунікаційних функцій відповідають специфікаціям ETSI після перетворення в NFV

Багато продуктів IT-мережевого обладнання не відповідають специфікаціям ETSI



Концепція, тісно пов’язана з SDN і NFV, — це NaaS («Мережа як послуга»).

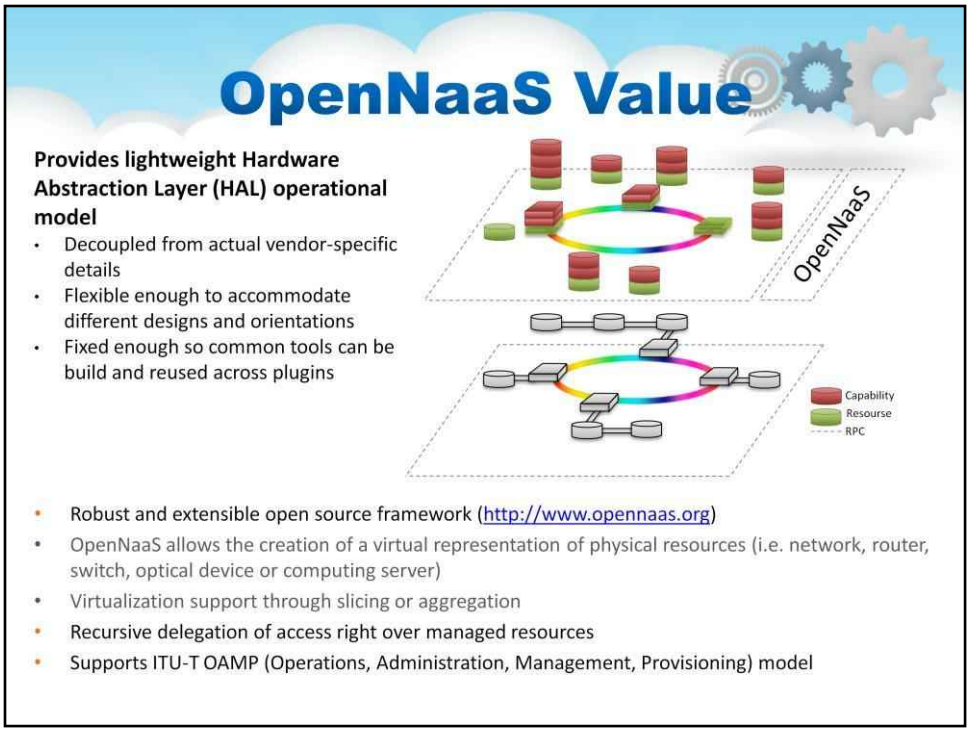
Слайд містить ілюстрацію, яка демонструє цей зв’язок.

NaaS формує нові мережеві служби, створені на основі VNF і SDN-додатків, щоб вписатися в простий і зручний контекст

OpenNaaS — це організація, яка просуває реалізацію NaaS з відкритим кодом

OpenNaaS — це платформа з відкритим вихідним кодом для:

Управління (фізичними та віртуальними) мережами
Віртуалізація мережевих ресурсів
Розгортання динамічних мережевих інфраструктур. Підтримка різнорідних мережевих пристроїв. Реалізація мультитенантності через нарізку. Пропонування мережі як послуги (NaaS)

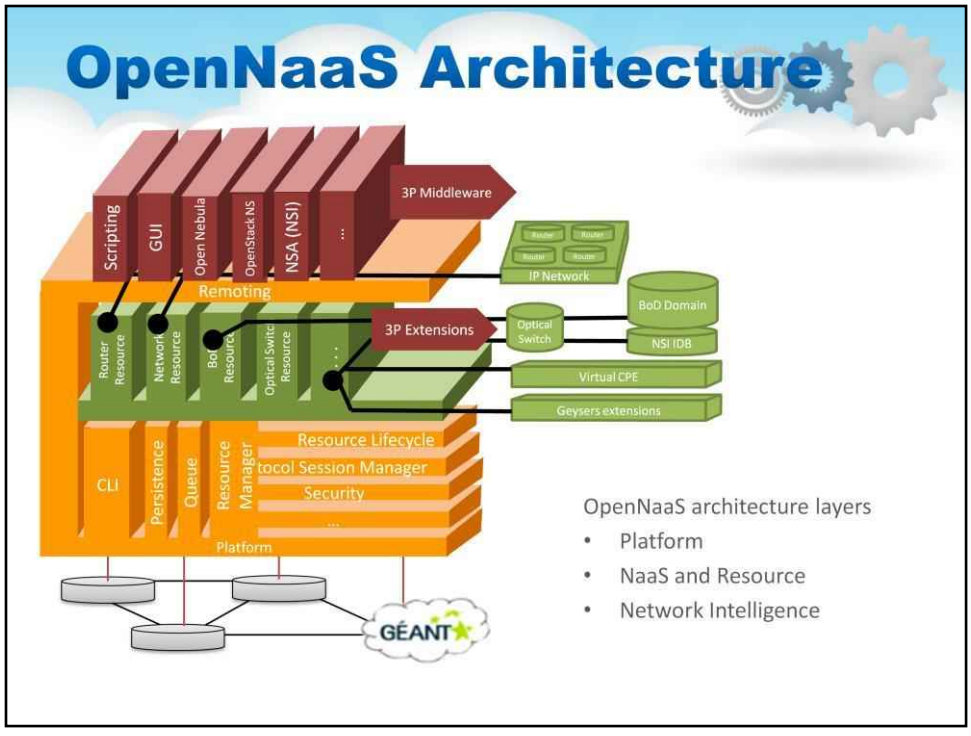


Значення OpenNaaS

OpenNaaS Надає спрощену операційну модель апаратного рівня абстракції

Відокремлено від фактичних деталей постачальника
Достатньо гнучкий, щоб адаптувати різні дизайни та орієнтації. Досить фіксований, щоб можна було створювати загальні інструменти та повторно використовувати їх у плагінах

OpenNaaS дозволяє створювати віртуальне представлення фізичних ресурсів (тобто мережі, маршрутизатора, комутатора, оптичного пристрою або обчислювального сервера)
Підтримка віртуалізації через нарізку або агрегацію.
Рекурсивне делегування прав доступу до керованих ресурсів
Підтримує модель ITU-T OAMP (Operations, Administration, Management, Provisioning).



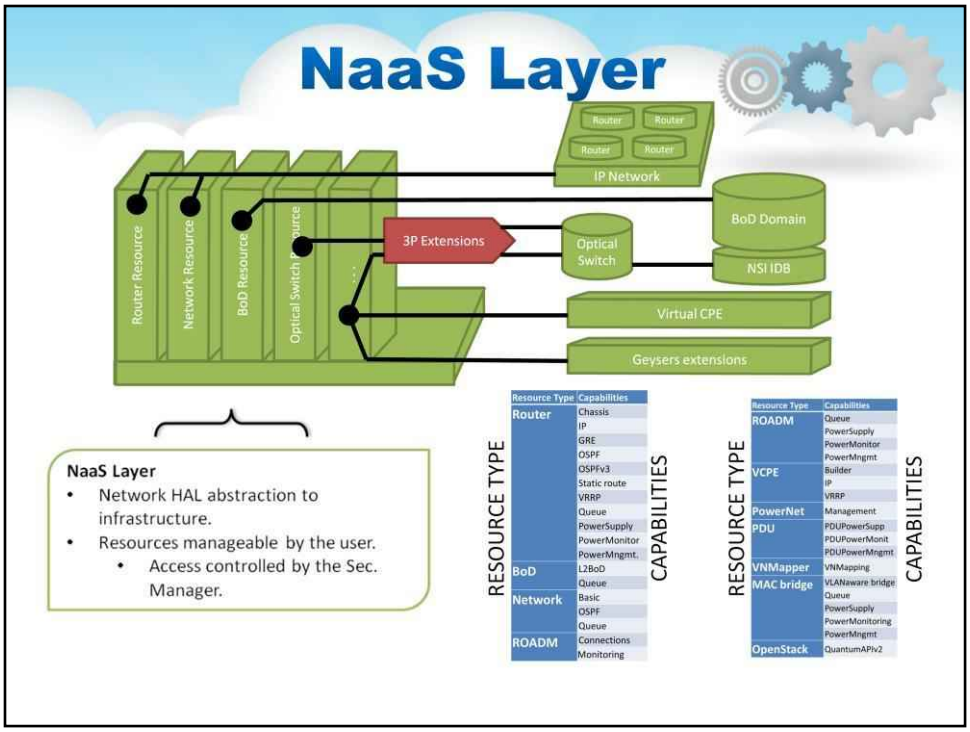
Слайд містить ілюстрацію, яка демонструє архітектуру OpenNaaS

Зверніть увагу на рівні архітектури OpenNaaS

Платформа

NaaS і ресурс

Мережевий інтелект



NaaS в деталях

Цей рівень в основному складається з розширень, що використовують базові компоненти абстрактної платформи. Саме тут додається підтримка певних типів ресурсів, можливостей і пристроїв.

Ресурс

Представляє собою керовану одиницю всередині концепції NaaS

Це може бути комутатор, маршрутизатор, посилання, логічний маршрутизатор, мережа тощо...

Розкладається на:

Модель.

Масив**Можливості.**

Можливість

Інтерфейс до заданого**Ресурс**функціональність.

Для маршрутизатора:

OSPF, IPv6, створення/керування логічними маршрутизаторами тощо...

Цей інтерфейс, як і модель, абстрактний і нейтральний від постачальника

внутрішньо,**Можливості**потрібен спосіб абстрактної реалізації деталей пристроїв, ми їх назвали**Дії**.

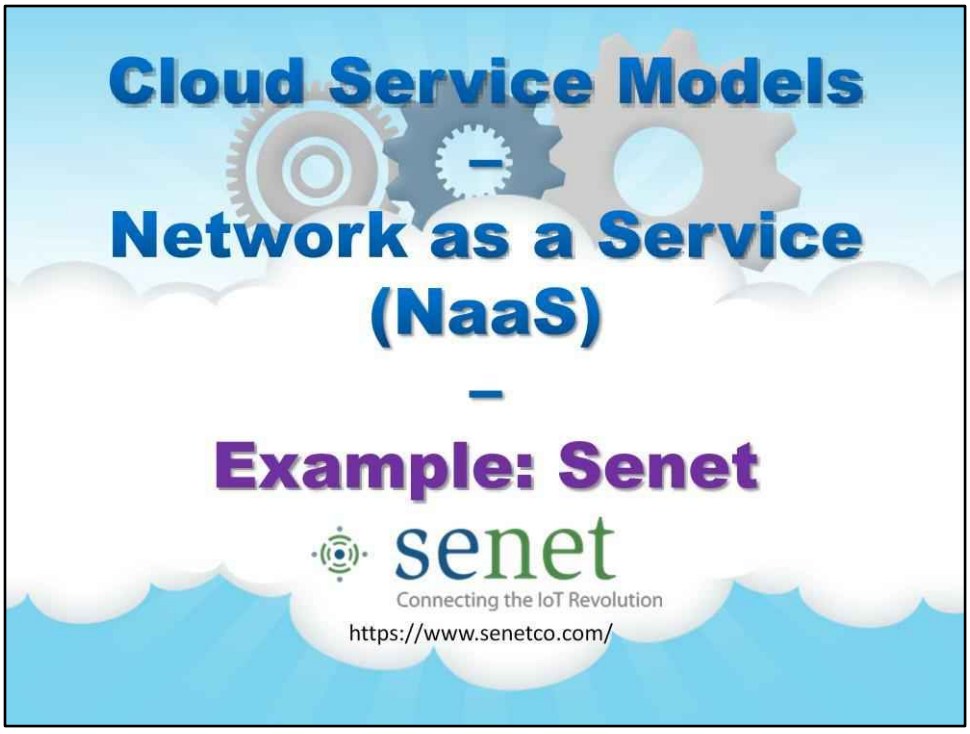
Дії

Спеціальна реалізація модифікації конфігурації від постачальника (і протоколу).

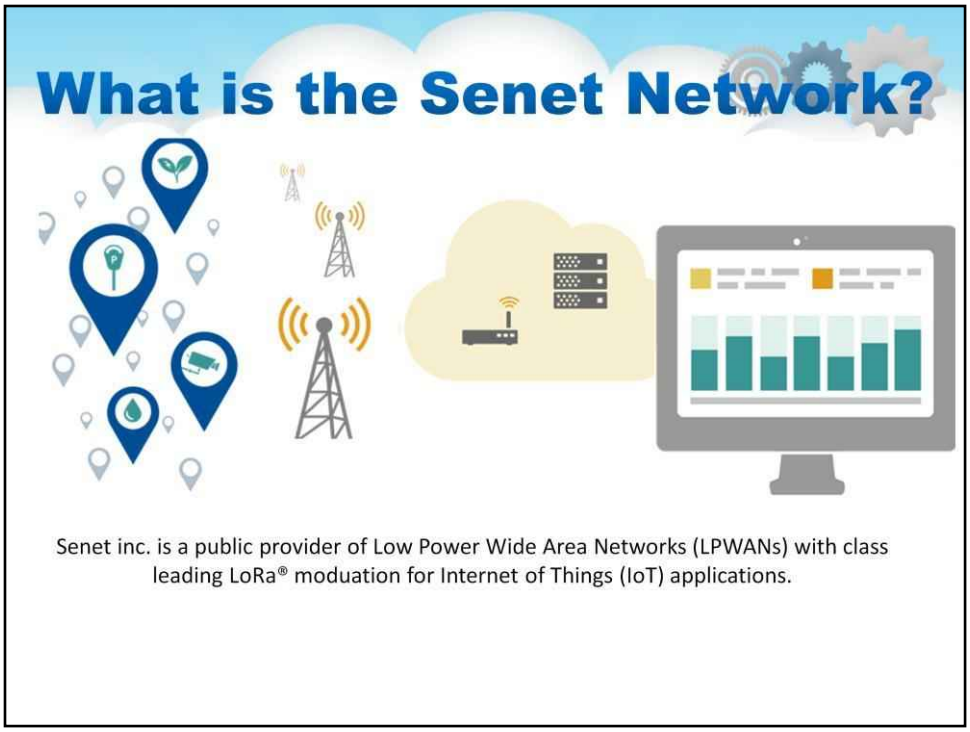
Його можна поставити в чергу.

Його можна скасувати (відкотити).

QueueManager використовується для укладання всіх**Дії**для виконання (підтримує відкат).

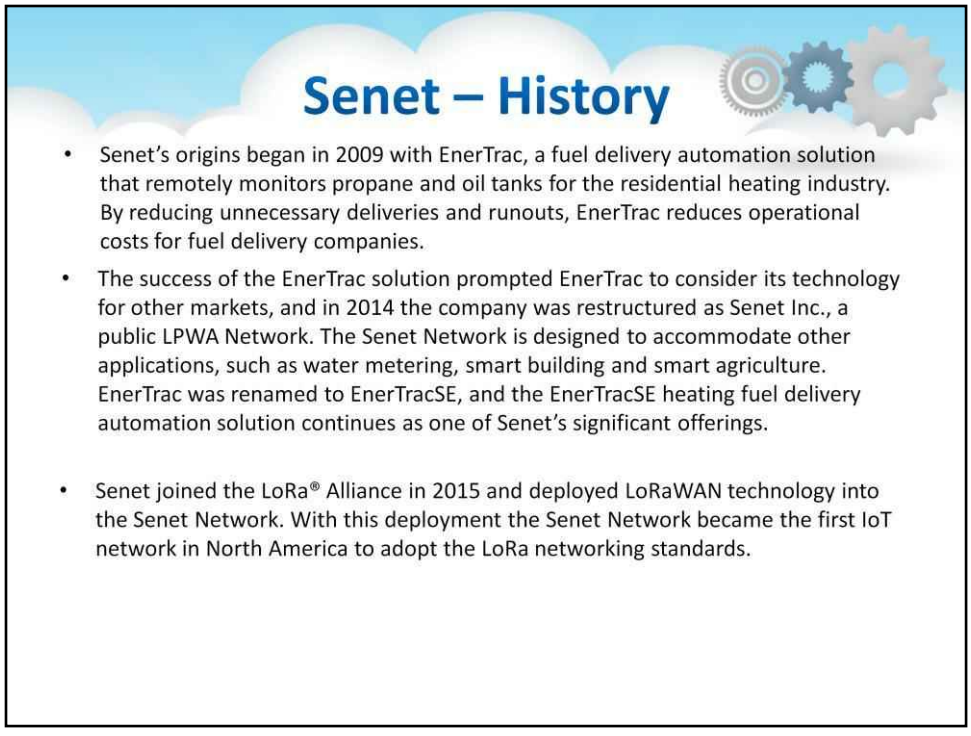


Приклад: Сенет



Senet Inc. – американським постачальником глобальної мережі з низьким енергоспоживанням (LPWAN) для програм IoT/M2M. Мережа Senet описана як «перший і єдиний державний постачальник мереж LPWA з найкращою в своєму класі модуляцією LoRa® для додатків IoT/M2M у Північній Америці». Її платформа відповідає потребам зростаючого «Інтернету речей» (IoT).) екосистема.

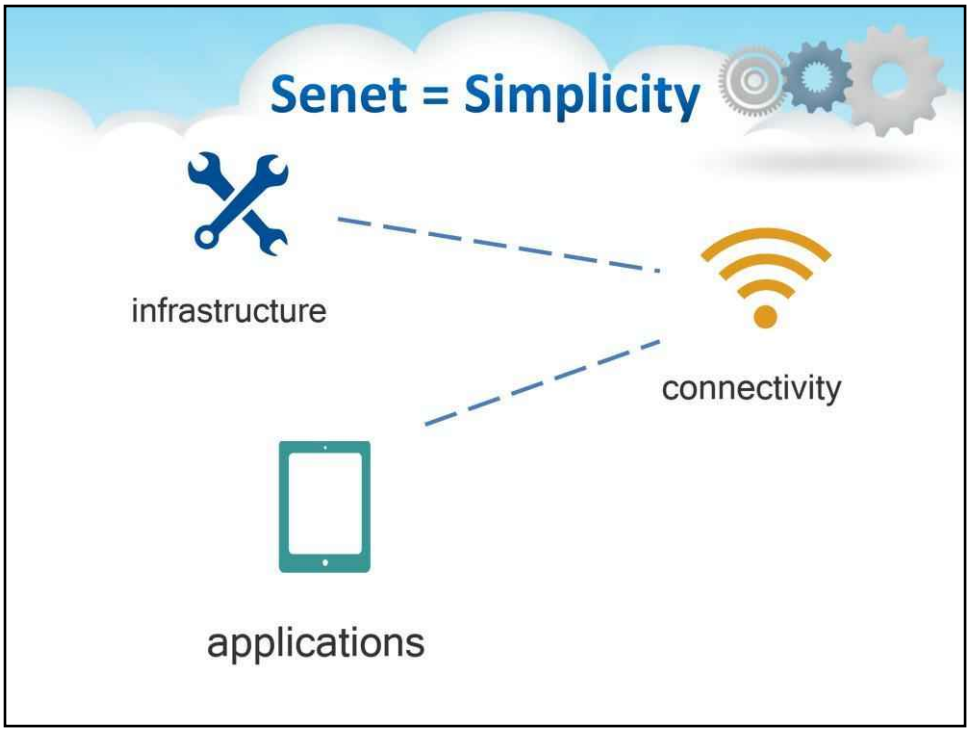
Мережа Senet дотримується відкритих глобальних мережевих стандартів, запропонованих LoRa® Alliance, і забезпечує низьке енергоспоживання, підключення між машинами (M2M) у широких географічних зонах. Він розроблений, щоб усунути бар’єри, які традиційно пов’язані з керуванням віддаленими пристроями, такі як надійність, радіус дії, час автономної роботи, можливість двонаправленого зв’язку, мобільність і вартість.



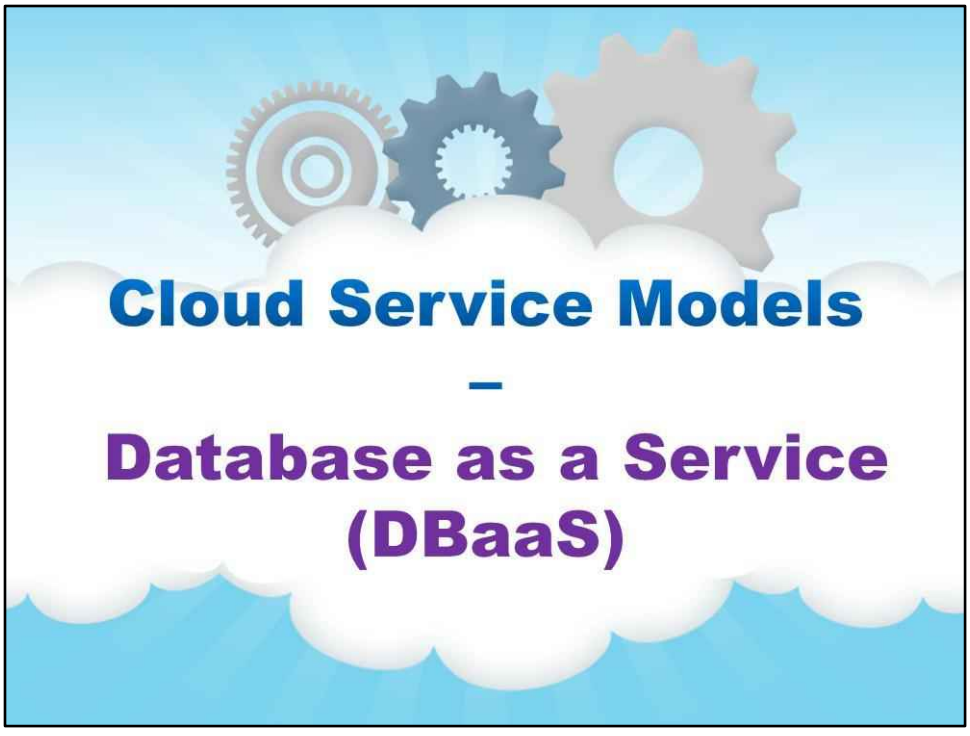
Виникнення Senet почалося в 2009 році з EnerTrac, рішенням для автоматизації доставки палива, яке дистанційно контролює баки з пропаном і паливом для житлової опалювальної галузі. Зменшуючи непотрібні доставки та розбиття, EnerTrac зменшує операційні витрати для компаній, що постачають паливо.

Успіх рішення EnerTrac спонукав EnerTrac розглянути свою технологію для інших ринків, і в 2014 році компанія була реструктуризована в Senet Inc., публічну мережу LPWA. Мережа Senet розроблена для інших додатків, таких як вимірювання води, розумне будівництво та розумне сільське господарство. EnerTrac було перейменовано на EnerTracSE, а рішення для автоматизації доставки палива для опалення EnerTracSE залишається однією з важливих пропозицій Senet.

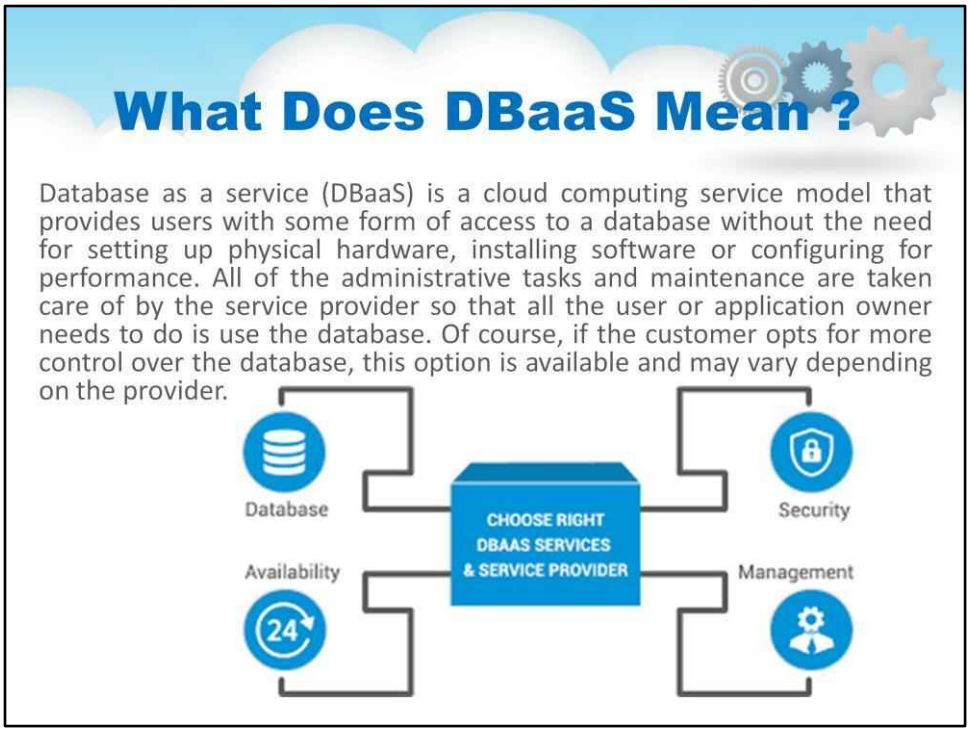
Senet приєднався до LoRa® Alliance у 2015 році та розгорнув технологію LoRaWAN у мережі Senet. Завдяки цьому розгортанню мережа Senet стала першою мережею IoT у Північній Америці, яка прийняла мережеві стандарти LoRa.



До Senet вартість і складність були перешкодами для підключення IoT. Тепер ми полегшуємо вам підключення за допомогою нашої повної наскрізної мережі та Інтернету речей послуги платформи. Все, що вам потрібно зробити, це підписатися. Решту зробимо ми.



База даних як послуга (DaaS)



Більше інформації тут:
https://searchcloudapplications.techtarget.com/definition/cloud-database

Database as a Service

Database as a Service (DBaaS) is an architectural and operational approach enabling DBAs to deliver database functionality as a service to internal and/or external customers.

Database as a Service architectures support the following required capabilities:

- Customer side provisioning and management of database instances using on-demand, self-service mechanisms
- Automation of monitoring with provider-defined service definitions, attributes and quality SLAs
- Fine-grained metering of database usage enabling showback reporting or charge-back for both internal and external functionality for each individual consumer

Why DBaaS?

DBaaS standardizes and optimizes the platform requirements which eliminates the need to deploy, manage and support dedicated database hardware and software for each project's multiple development, testing, production, and failover environments.

DBaaS architectures are inherently designed for elasticity and resource pooling. They deliver production and non-production database services that support average daily workload requirements and are not impacted by:

- Resource Limitations
- Time Sensitive Projects
- Hardware limitations/budgets

Benefits of DBaaS

DBaaS solution provides an organization a number of benefits, the chief among them being:

- Developer agility
- DBA productivity
- Application reliability, performance and security.

Benefits of DBaaS – Developer Agility

•When a developer wishes to provision a database, the steps involved include provisioning compute, storage and networking components, configuring them properly and then installing database software. Finally, the database software must be configured properly to utilize the underlying infrastructure components.

•This multi-step process leaves many opportunities for errors, omissions and non-standard modes of operation. When the thing that is being provisioned (a database) is the "system of record", this is unacceptable.

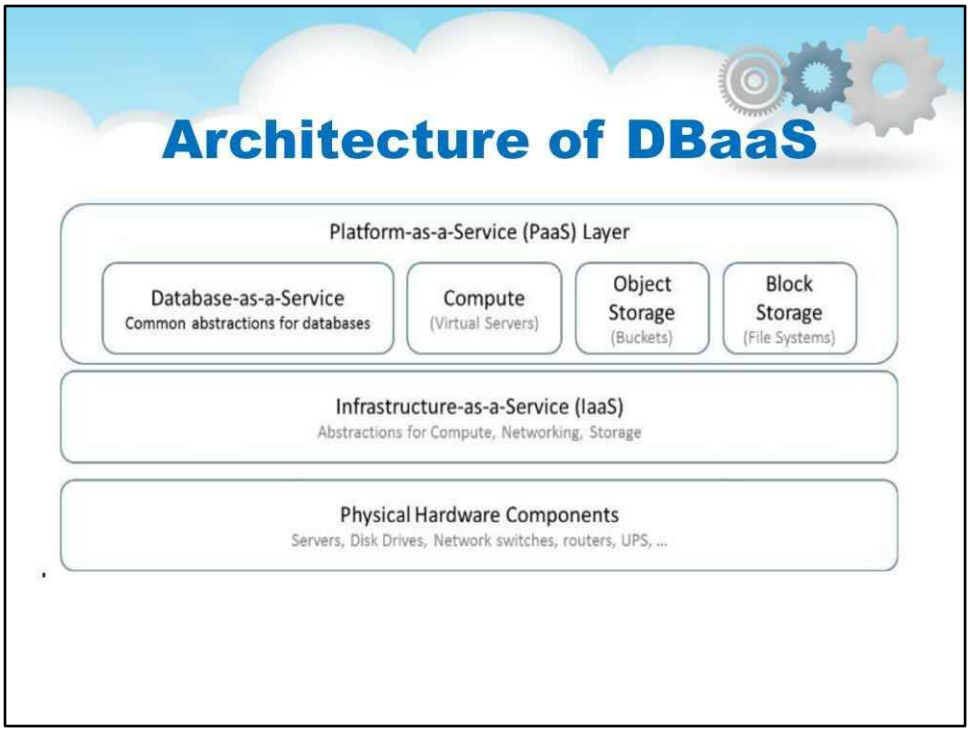
Benefits of DBaaS – DBA Productivity

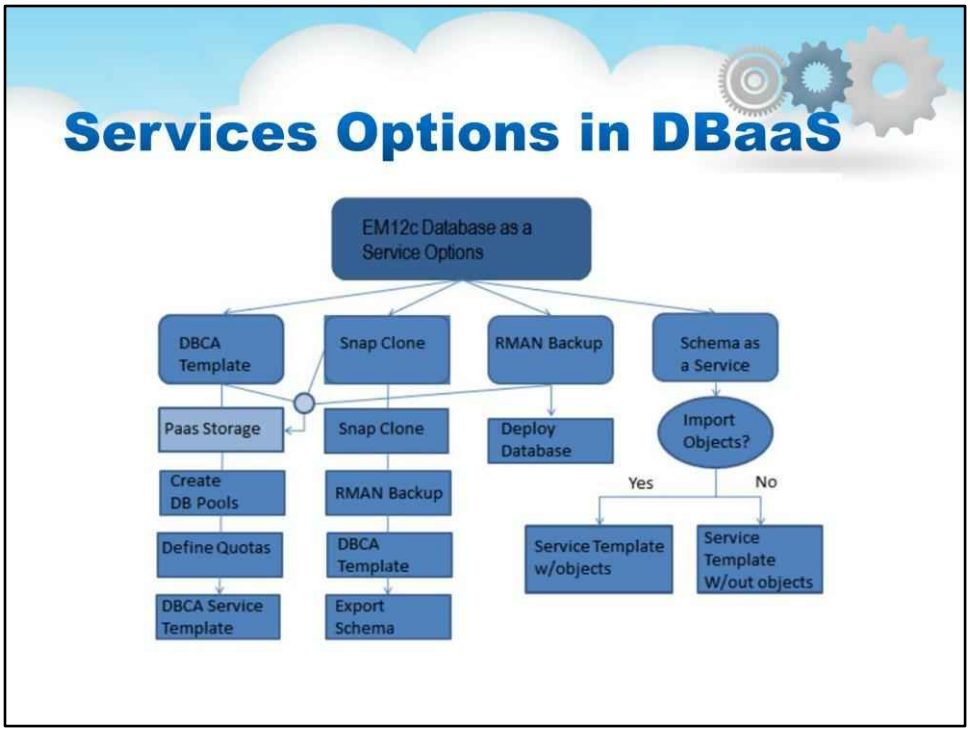
•When an enterprise operates hundreds of instances of many different databases, a considerable resources get consumed on maintenance and upkeep. This includes things like tuning, configuration, patching, periodic backups, and so on; all the things that DBAs have to do to keep databases in proper working order.

•DBaaS solutions provide abstractions that allow DBAs to manage groups of databases and perform operations like upgrades and configuration changes on a fleet of databases in a simplified way.

Benefits of DBaaS – Application Reliability, Performance and Security

- Databases are often the "system of record" and are the repository of valuable information in the organization. A database outage could have catastrophic impact. Through automation and standardization, DBaaS ensures that all common workflows involved in the provisioning, configuration, management, and operation of databases are consistent.
- Through this standardization, a DBaaS ensures that all databases are operated in the same way, and in keeping with the best practices established by the IT organization. This , frees up the developer and DBA to work on more important things like the application and innovation rather than the boring minutiae of running a database.

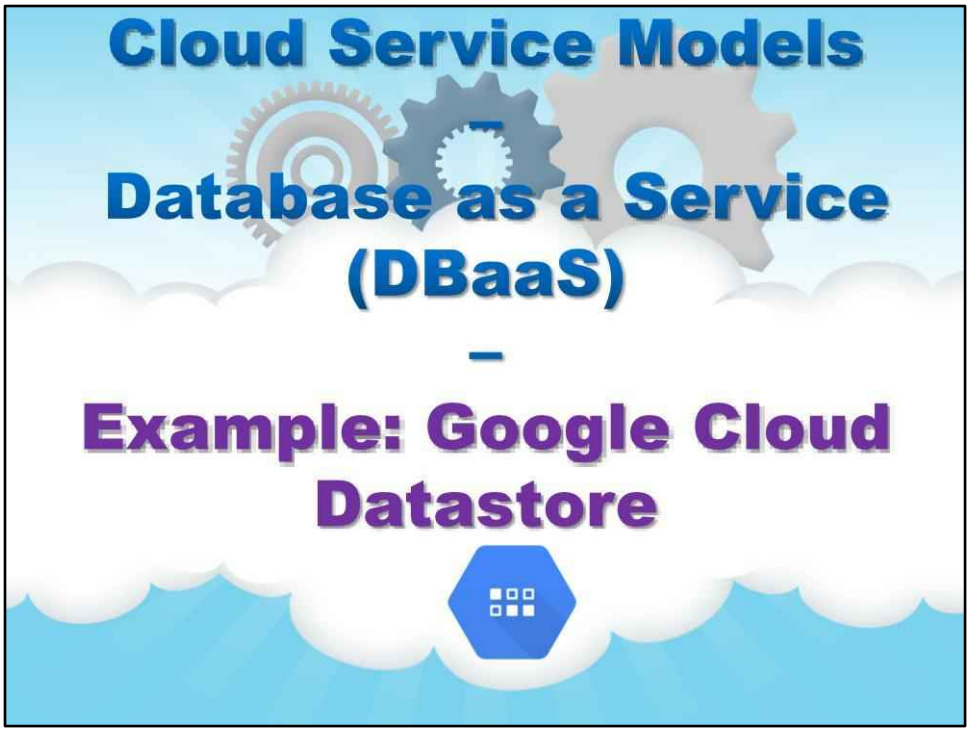

Architecture of DBaaS
Platform-as-a-Service (PaaS) Layer
Database-as-a-Service
Common abstractions for databases
Compute
(Virtual Servers)
Object Storage
(Buckets)
Block Storage
(File Systems)
Infrastructure-as-a-Service (IaaS)
Abstractions for Compute, Networking, Storage
Physical Hardware Components
Servers, Disk Drives, Network switches, routers, UPS, ...



На цьому слайді показано основні параметри служби в DBaaS

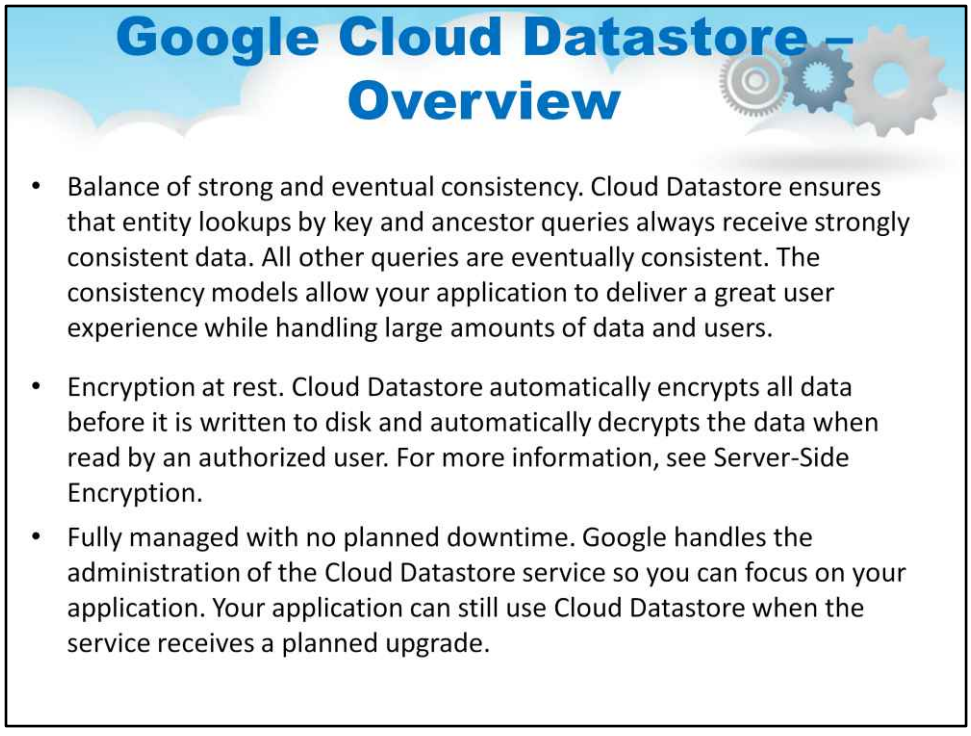
Conclusion

- Database-as-a-Service provides a framework within which enterprises can operate all of their databases.
- It provides end users (developers and application deployers) with improved agility and simplified provisioning and operation, and the flexibility to choose from amongst a number of pre-configured options established by the IT organization.
- DBaaS improves the operation (IT and DBA) of fleets of diverse database through automation and standardization.
- DBaaS allows IT organizations to cost-effectively offer their users with a number of database choices.
- DBaaS ensures that these databases are operated in a safe and secure way and in compliance with established best practices.



Приклад: Google Cloud Datastore

Офіційна документація: https://cloud.google.com/datastore/docs/concepts/overview



Офіційна документація: https://cloud.google.com/datastore/docs/concepts/overview

Comparison with other traditional databases

While the Cloud Datastore interface has many of the same features as traditional databases, as a NoSQL database it differs from them in the way it describes relationships between data objects. Here's a high-level comparison of Cloud Datastore and relational database concepts:

Concept	Cloud Datastore	Relational database
Category of object	Kind	Table
One object	Entity	Row
Individual data for an object	Property	Field
Unique ID for an object	Key	Primary key

Comparison with other traditional databases

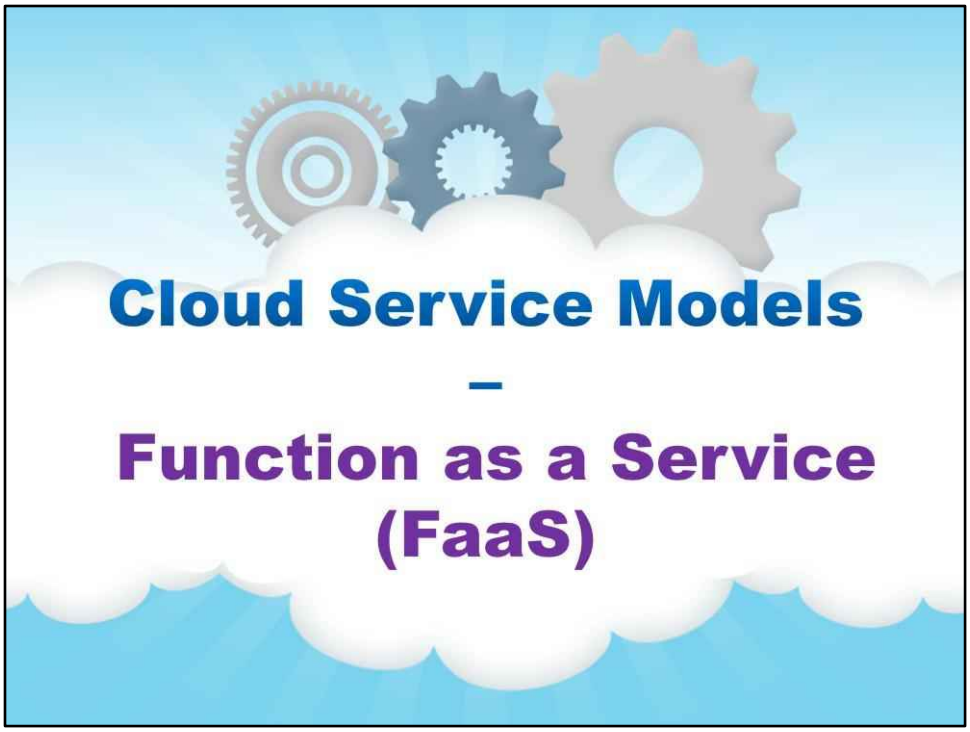
In particular, Cloud Datastore differs from a traditional relational database in the following important ways:

- designed to automatically scale to very large data sets, allowing applications to maintain high performance as they receive more traffic
- writes scale by automatically distributing data as necessary
- reads scale because the only queries supported are those whose performance scales with the size of the result set (as opposed to the data set). This means that a query whose result set contains 100 entities performs the same whether it searches over a hundred entities or a million. This property is the key reason some types of queries are not supported
- Unlike traditional relational databases which enforce a schema, it doesn't require entities of the same kind to have a consistent property set (although you can choose to enforce such a requirement in your own application code).

Google Cloud Datastore – Applications

Cloud Datastore is ideal for applications that rely on highly available structured data at scale. You can use Cloud Datastore to store and query all of the following types of data:

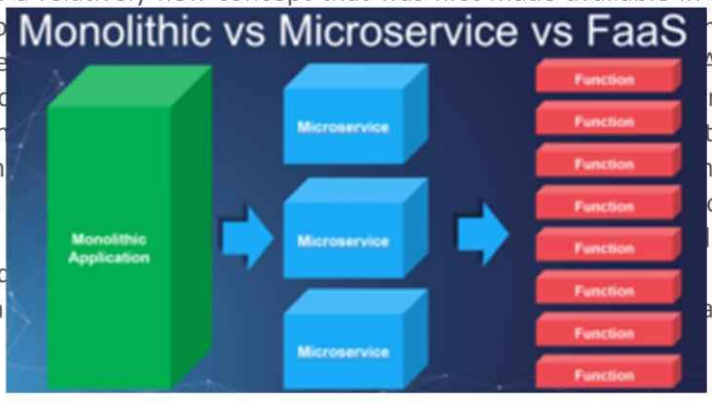
- Product catalogs that provide real-time inventory and product details for a retailer.
- User profiles that deliver a customized experience based on the user's past activities and preferences.
- Transactions based on ACID properties, for example, transferring funds from one bank account to another.



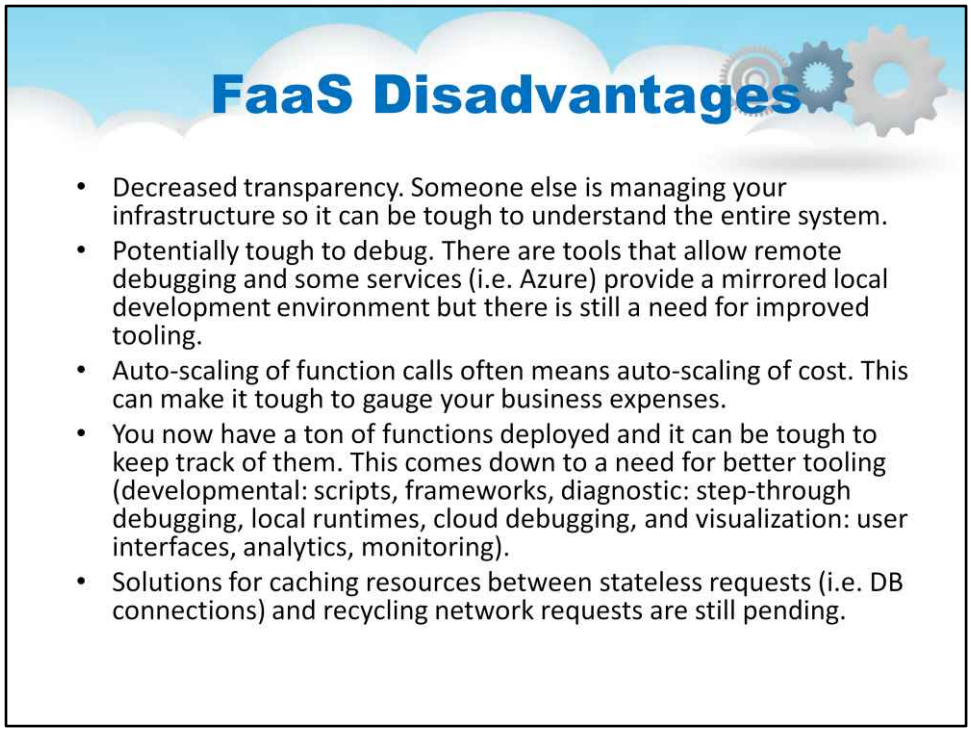
Функція як послуга (FaaS)

What is a purpose of FaaS?

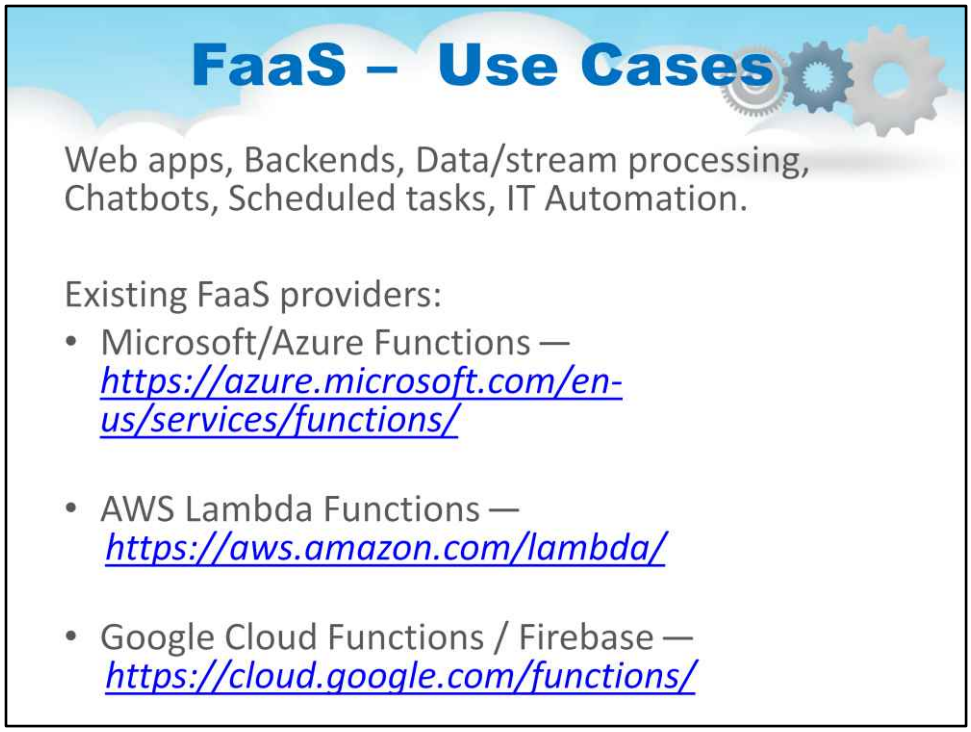
FaaS is a relatively new concept that was first made available in 2014 by hoo hbda, Google Azure Functio ream allowin thout buildin eans is that o the cloud ities! Instead load, you ca an be scaled



Чудова стаття, яка описує, що таке FaaS і що це означає: https://stackify.com/function-as-a-service-serverless-architecture/



Чудова стаття, яка описує, що таке FaaS і що це означає: https://stackify.com/function-as-a-service-serverless-architecture/



Існуючі постачальники FaaS:

Microsoft/AzureФункції—https://azure.microsoft.com/en-us/services/functions/ пропонує

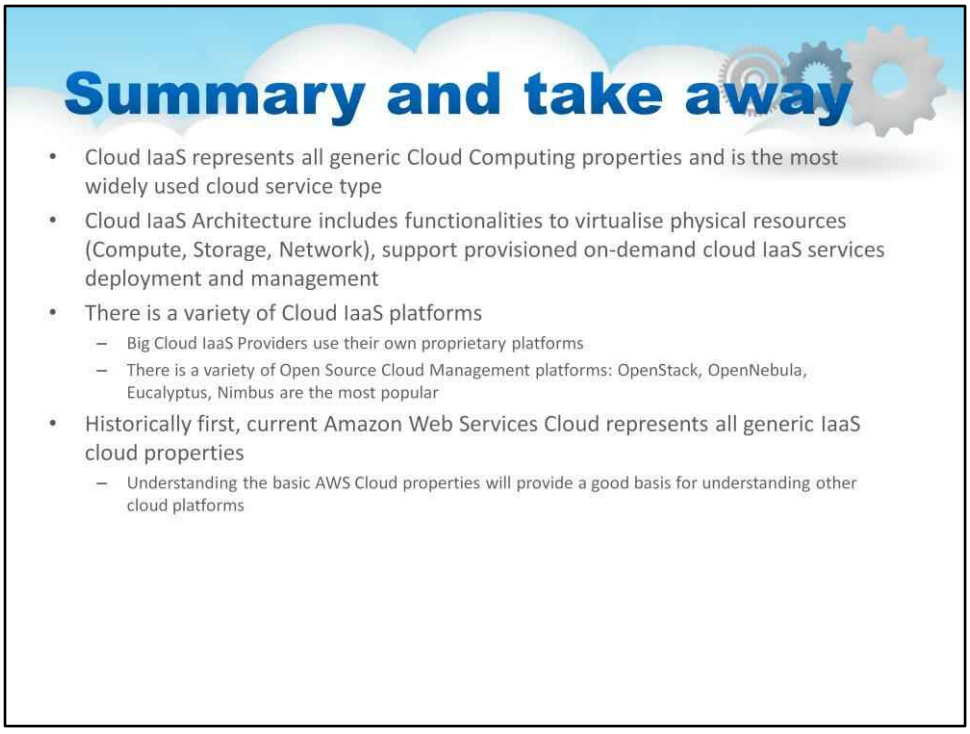
- Інструменти, прив’язки та тригери
- Логікапрограми—візуальнідизайнер із 25+ роз’ємами та оркестровкою функцій
- Сітка подій
- Локальне налагодження

AWS ЛямбдаФункції—https://aws.amazon.com/lambda/ пропонує

- функції,
- API,
- Таблиці

Функції Google Cloud /Firebase—https://cloud.google.com/functions/ пропонує

- Хмарне сховище,
- Cloud pub/sub,
- HTTPS,
- Журнали Stackdriver



Підсумок і на винос

Cloud IaaS представляє всі загальні властивості хмарних обчислень і є найпоширенішим типом хмарних послуг

Cloud IaaS Architecture включає функції для віртуалізації фізичних ресурсів (обчислення, зберігання, мережа), підтримку розгортання та керування наданими хмарними службами IaaS на вимогу

Існує безліч хмарних платформ IaaS

Постачальники Big Cloud IaaS використовують власні пропрієтарні платформи Існує різноманітність платформ керування хмарою з відкритим вихідним кодом

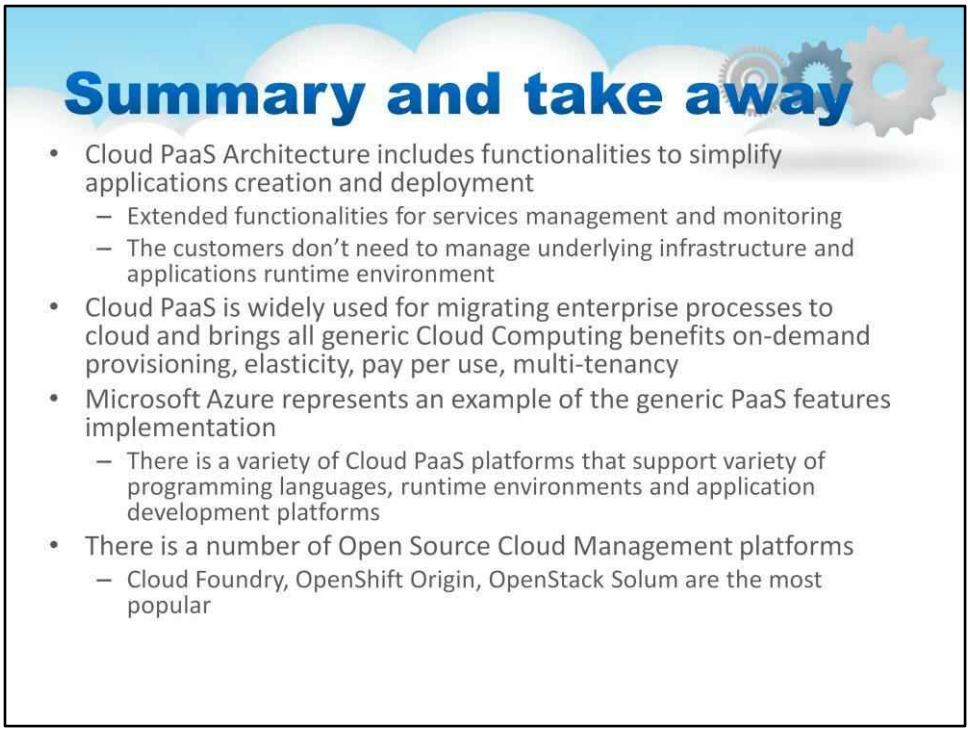
Найпопулярнішими є OpenStack, OpenNebula, Eucalyptus, Nimbus

І, нарешті, у цьому посібнику ми обговорили Amazon Web Services Cloud та його основні функції

По-перше, нинішня AWS Cloud представляє все

загальні властивості хмари IaaS. Розуміння основних AWS

Властивості хмари стануть хорошою основою для розуміння інших хмарних платформ



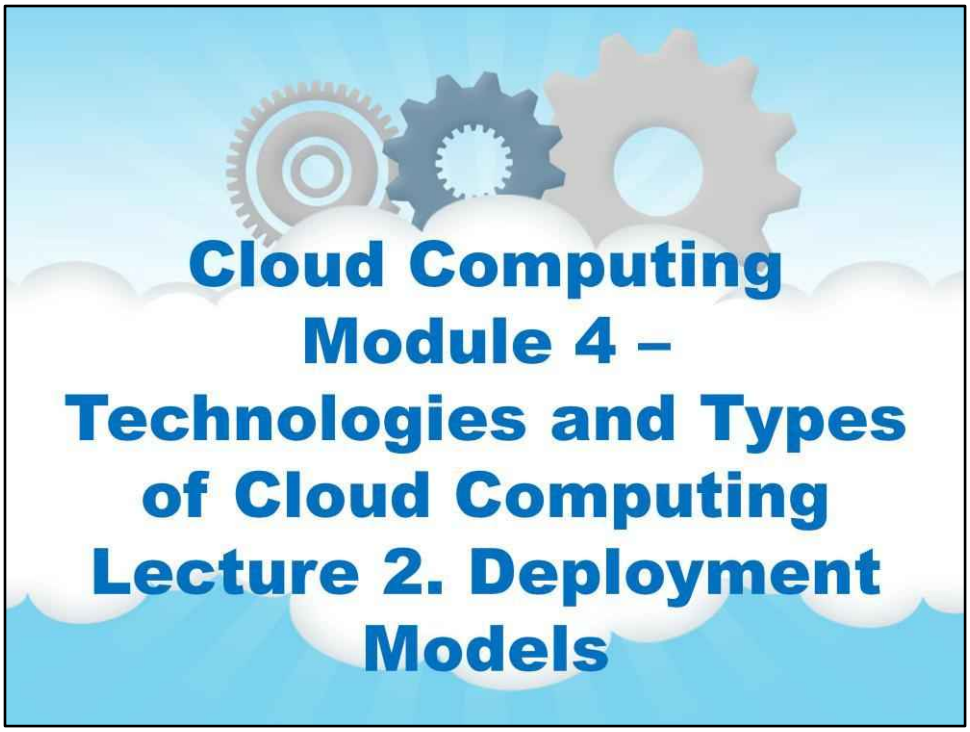
У цьому уроці ми розглянули

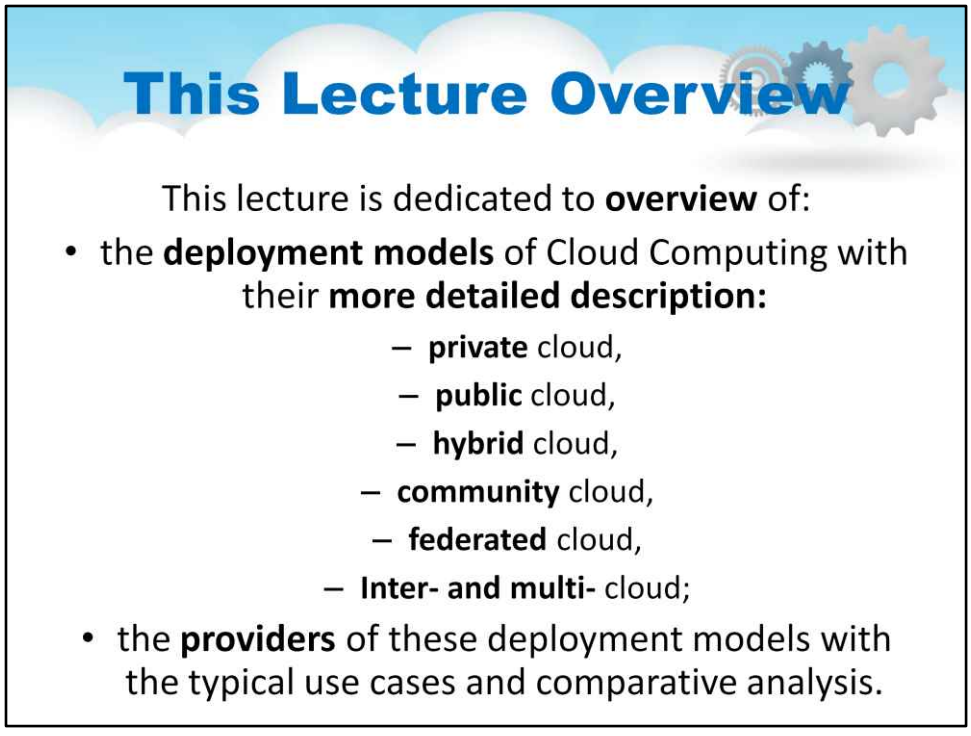
Cloud PaaS Architecture включає функції для спрощення створення та розгортання програм

Cloud PaaS широко використовується для перенесення корпоративних процесів у хмару та забезпечує всі загальні переваги хмарних обчислень, зокрема надання послуг за вимогою, еластичність, оплата за використання, багатокористувацький доступ

Microsoft Azure є прикладом реалізації загальних функцій PaaS

Існує кілька платформ керування хмарою з відкритим вихідним кодом

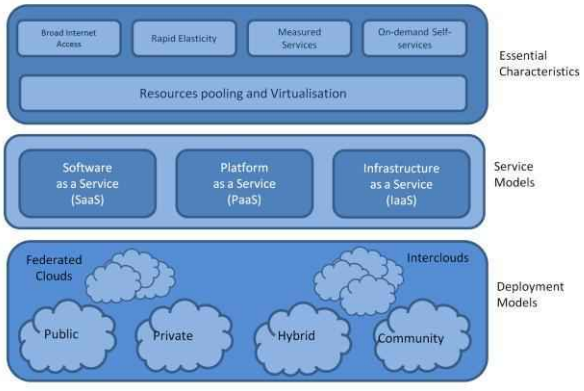

Cloud Computing
Module 4 –
Technologies and Types
of Cloud Computing
Lecture 2. Deployment
Models



Лекція 2. Моделі розгортання

Огляд

Cloud Computing – Visualization by NIST



Cloud characteristics

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured Service

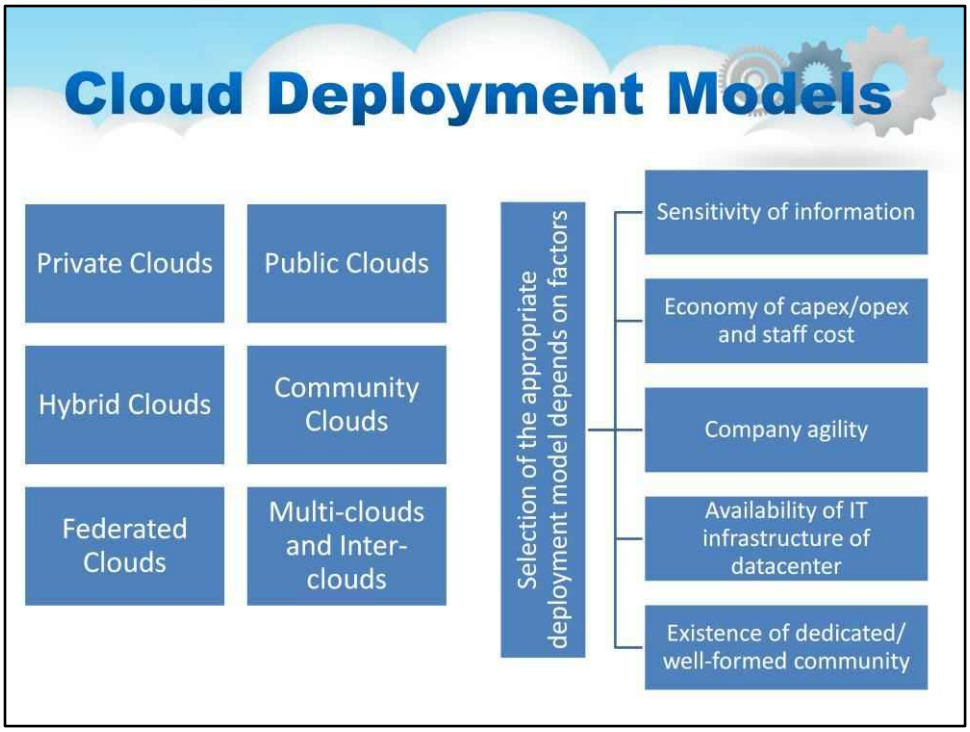
Basic service models

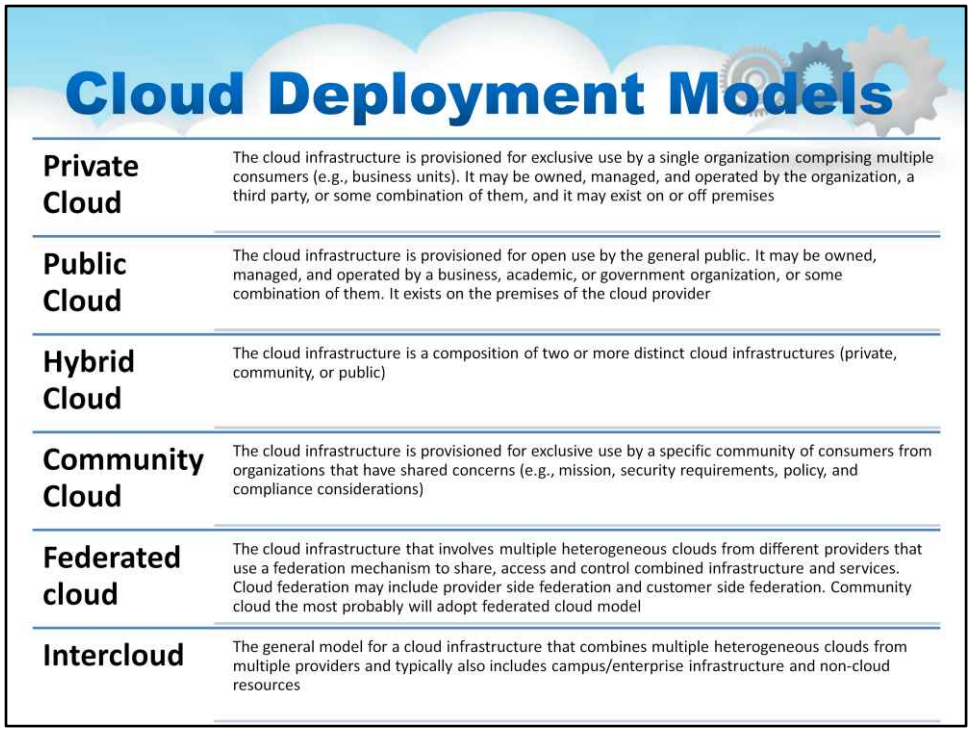
- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

Deployment models

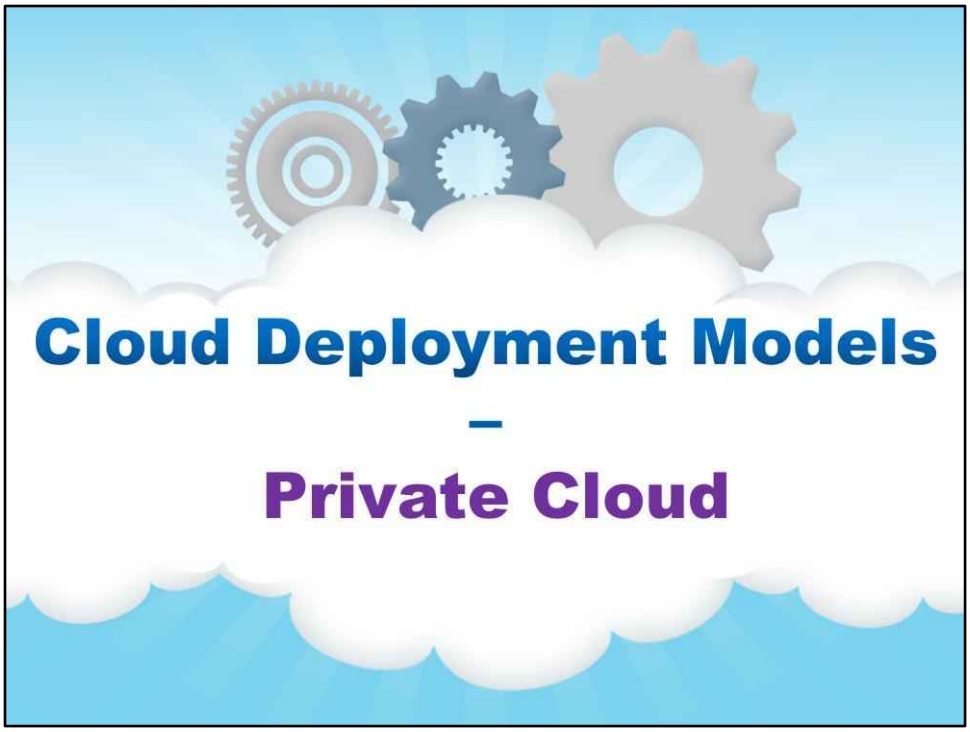
- Private clouds
- Public clouds
- Hybrid clouds
- Community clouds
- Federated clouds, Interclouds

[NIST] Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

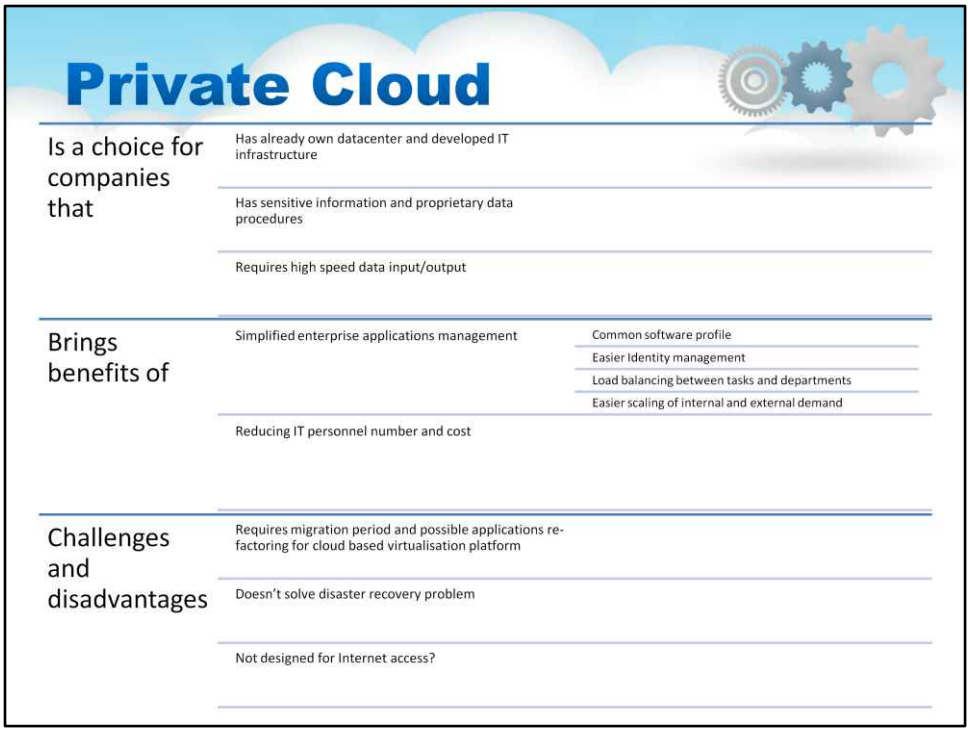

Cloud Deployment Models
Private Clouds
Public Clouds
Hybrid Clouds
Community Clouds
Federated Clouds
Multi-clouds and Inter-clouds
Selection of the appropriate deployment model depends on factors
Sensitivity of information
Economy of capex/opex and staff cost
Company agility
Availability of IT infrastructure of datacenter
Existence of dedicated/ well-formed community



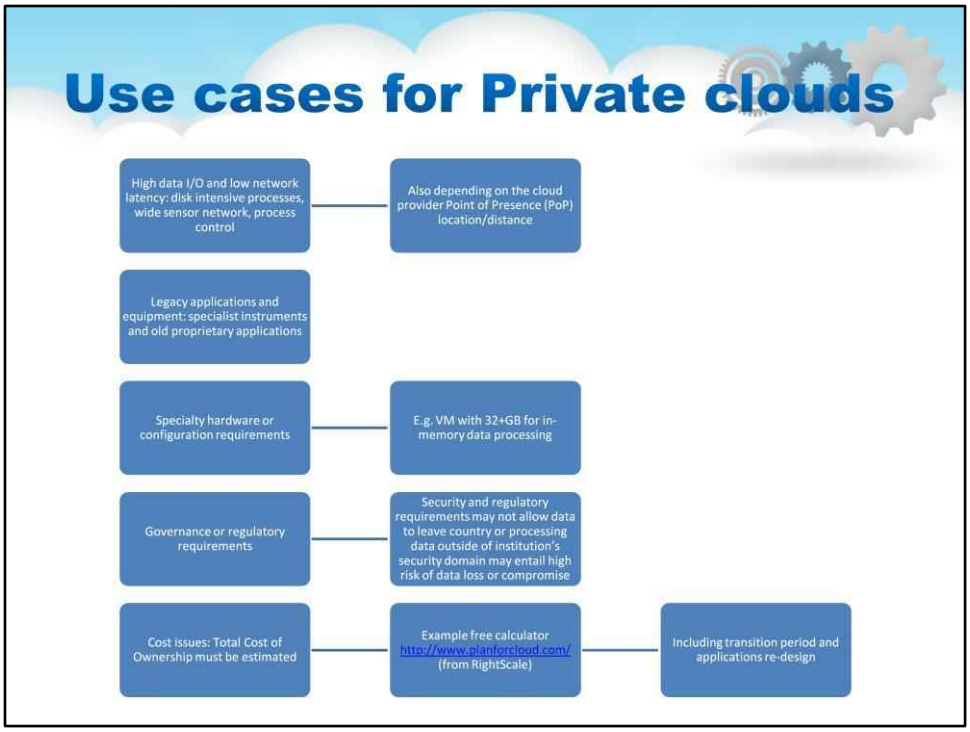
На цьому слайді показано чотири моделі розгортання Cloud IaaS, визначені в стандарті NIST SP 800-145: приватні, публічні, гібридні хмари та хмари спільноти. Нещодавно розвинулися ще два: об’єднана хмара та Intercloud.

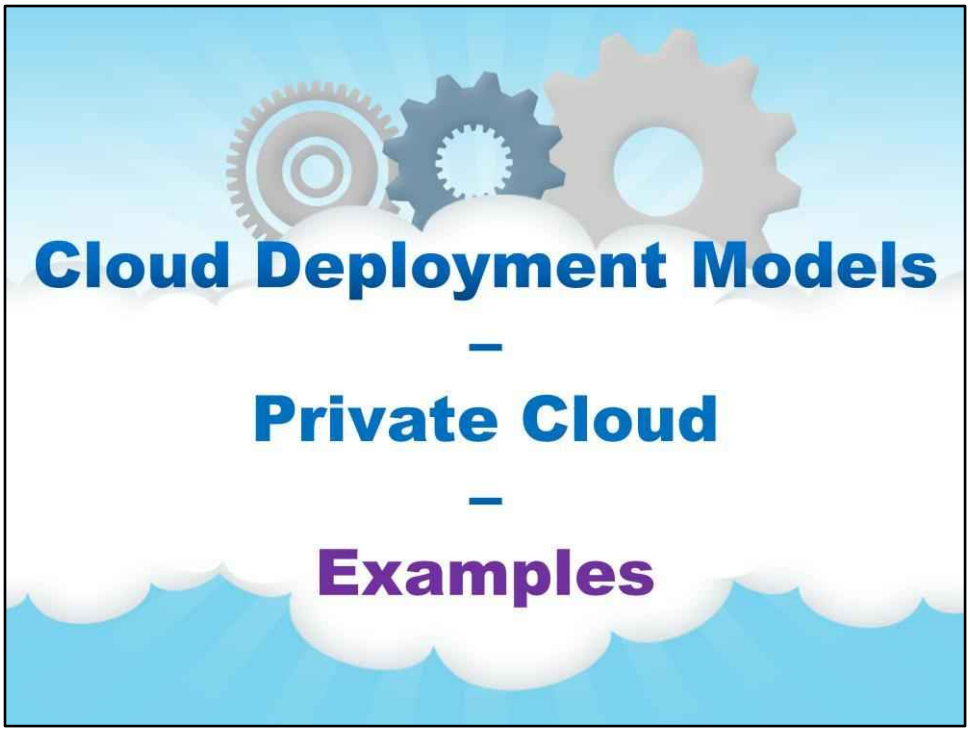


Приватна хмара



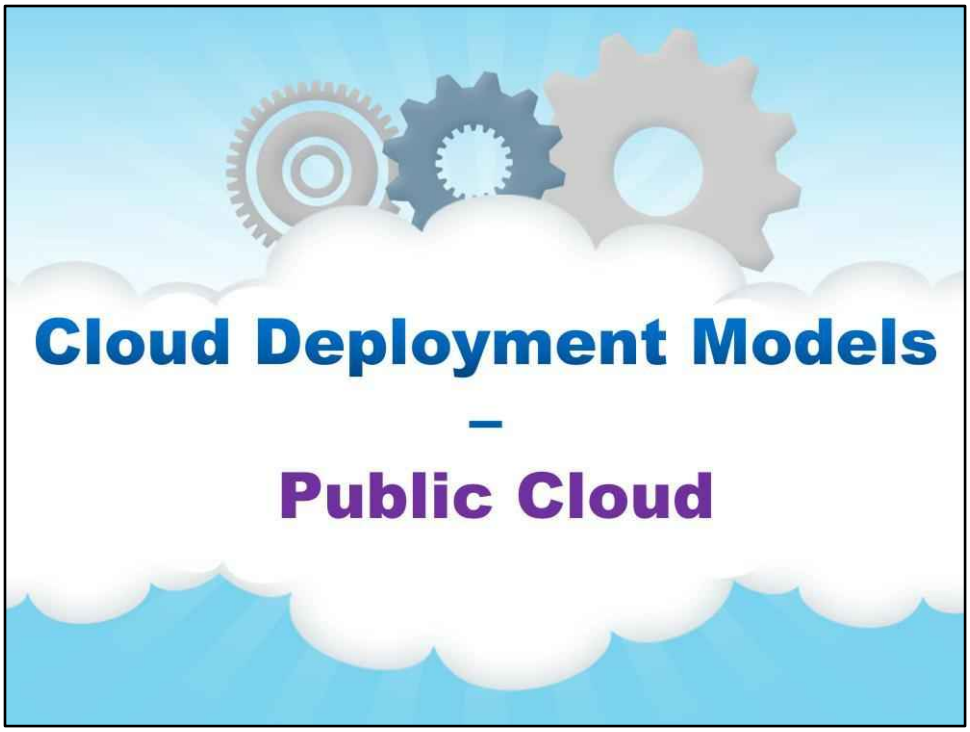
Хмарна інфраструктура надається для ексклюзивного використання однією організацією, що складається з кількох споживачів (наприклад, бізнес-підрозділів). Він може належати, управлятися та управлятися організацією, третьою стороною або деякою їх комбінацією, і він може існувати в приміщенні або за його межами


Use cases for Private clouds
High data I/O and low network latency: disk intensive processes, wide sensor network, process control
Also depending on the cloud provider Point of Presence (PoP) location/distance
Legacy applications and equipment: specialist instruments and old proprietary applications
Specialty hardware or configuration requirements
E.g. VM with 32+GB for in-memory data processing
Governance or regulatory requirements
Security and regulatory requirements may not allow data to leave country or processing data outside of institution's security domain may entail high risk of data loss or compromise
Cost issues: Total Cost of Ownership must be estimated
Example free calculator http://www.planforcloud.com/ (from RightScale)
Including transition period and applications re-design

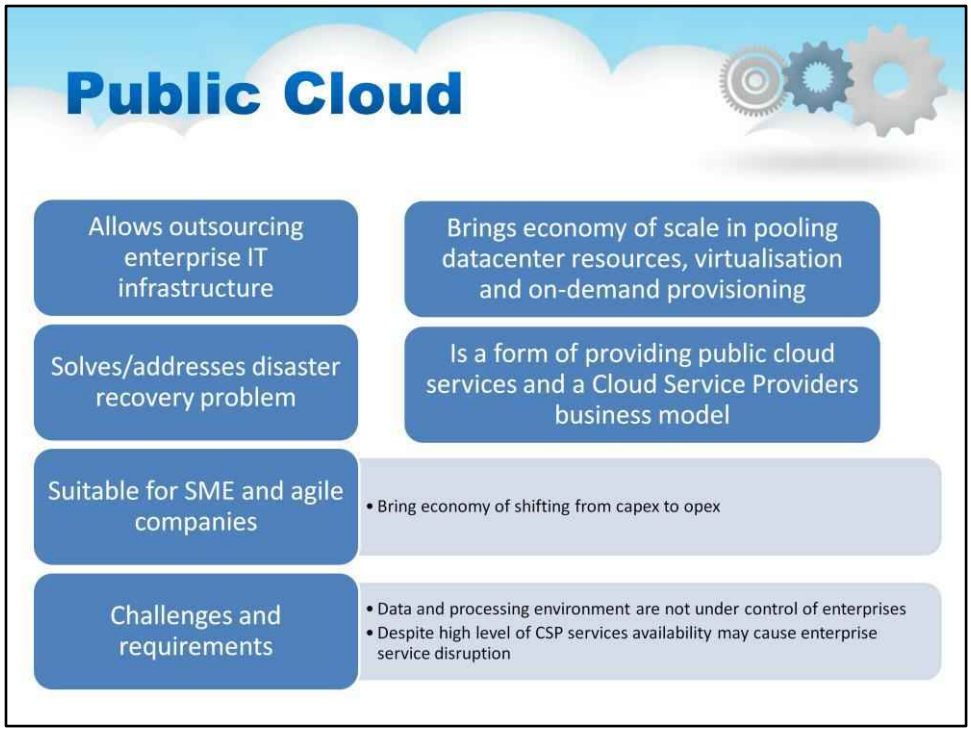


Приклади


Examples
hp
NetIQ
ubuntu
ORACLE
cisco
GOGRID
amazon web services
Amazon VPC
openstack
apachecloudstack
SUSE
DELL
Microsoft
rackspace
the open cloud company
CITRIX
vmware
Eucalyptus
nebula
Piston



Громадська хмара



Хмарна інфраструктура надається для відкритого використання широким загалом. Воно може належати, управлятися та управлятися діловою, академічною чи державною організацією або деякою їх комбінацією. Він існує на території хмарного провайдера

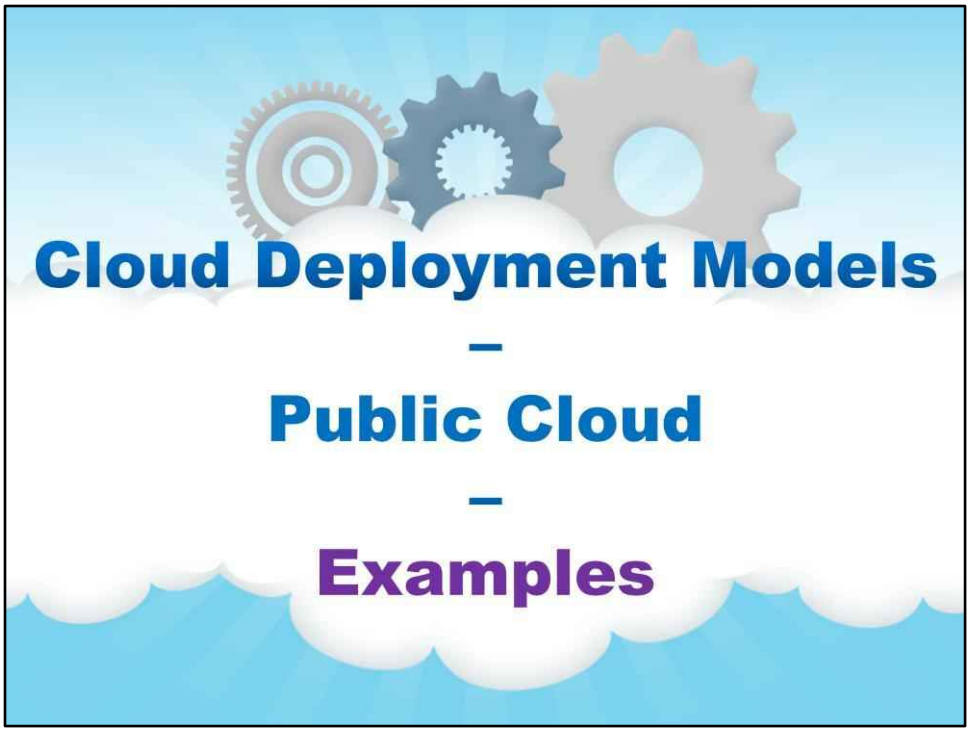
Use cases for Public Clouds

Application workloads characterised by

- Unpredictable growth: game or social websites, marketing campaigns
  - Benefit from on-demand and elastic character of cloud resources
- Cyclical: applications with regular daily or seasonal traffic fluctuation such as financial markets or eCommerce
  - Benefit from cloud elasticity and auto-scaling to keep utilisation high
- Easily parallelised: applications using batch processing, data analytics, media encoding

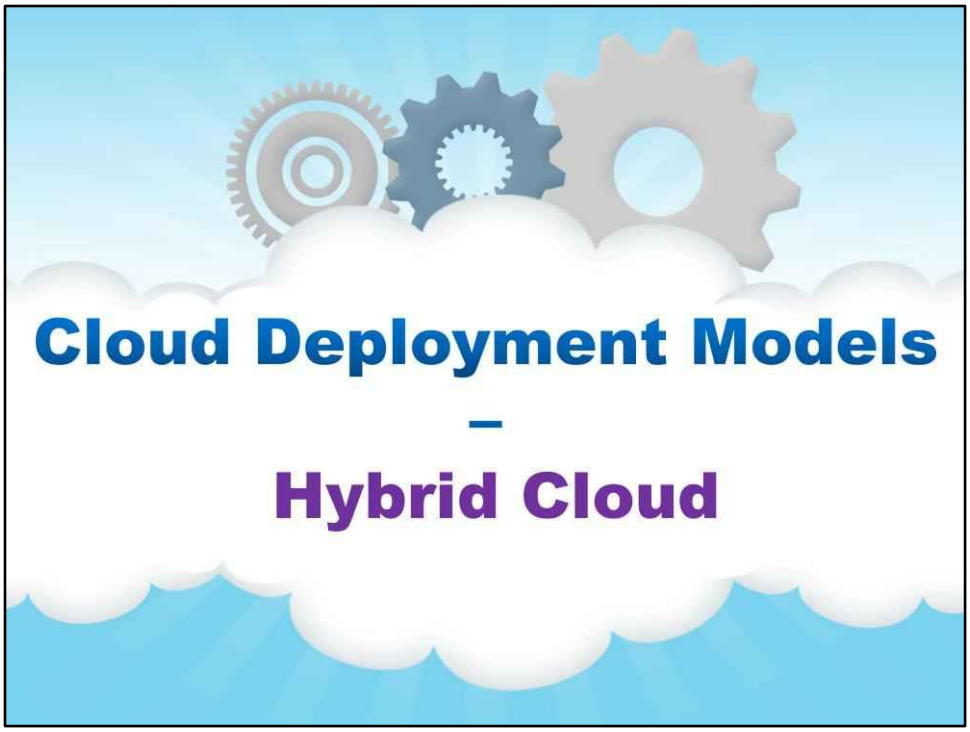
Example use cases

- Web/social web and mobile applications
- Development and test, proof of concept
- Big Data analytics, business reporting

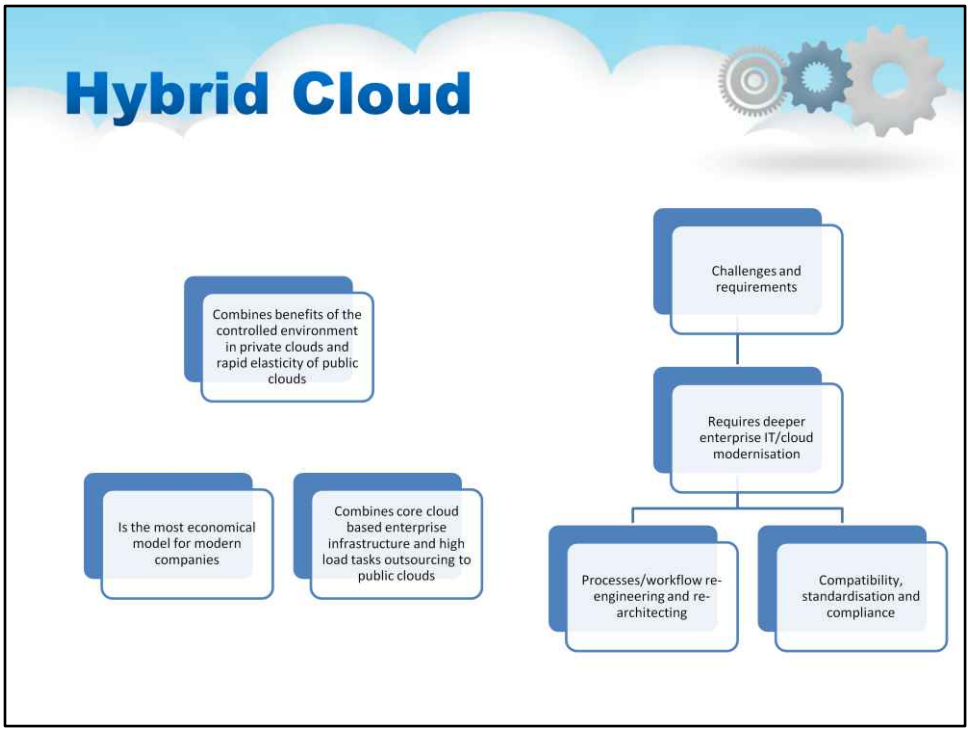


Приклади

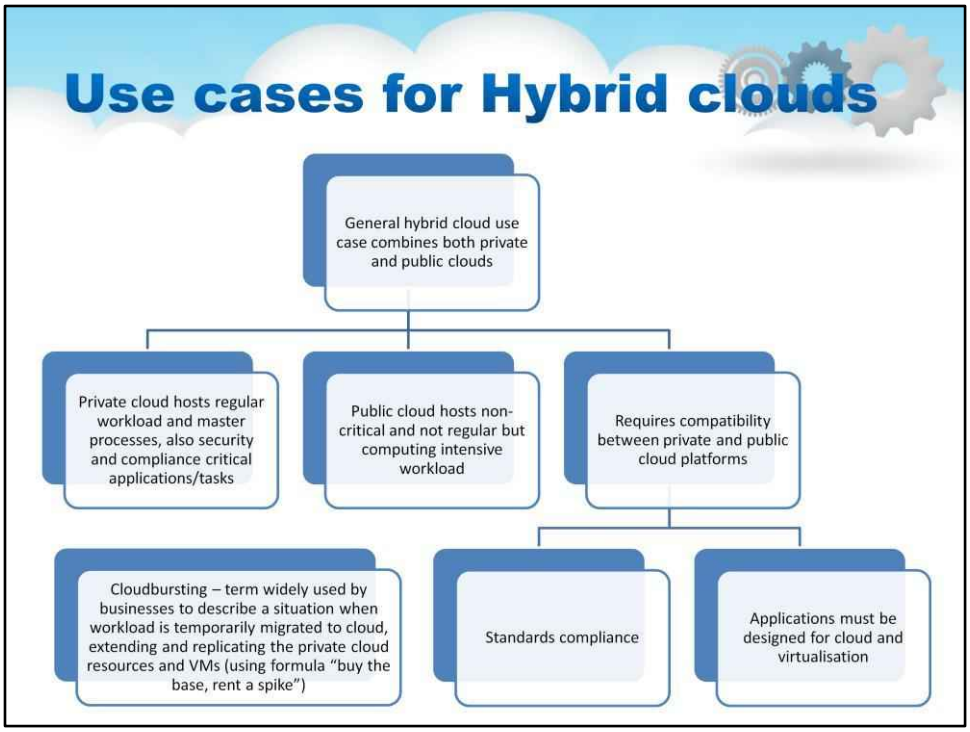

Examples
Microsoft Azure
IBM Cloud
Google Cloud Platform
amazon
web services

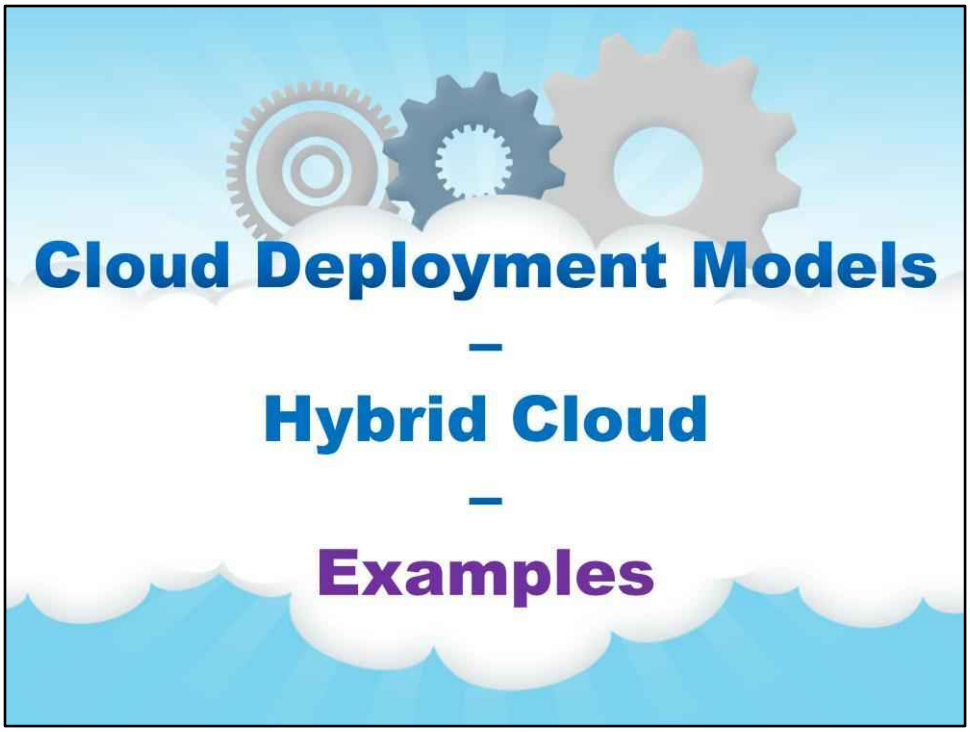


Гібридна хмара



Хмарна інфраструктура — це композиція з двох або більше різних хмарних інфраструктур (приватних, спільнотних або загальнодоступних), які залишаються унікальними сутностями, але пов’язані між собою стандартизованою або запатентованою технологією, яка забезпечує переносимість даних і програм (наприклад, розрив хмари для балансування навантаження між хмари)


Use cases for Hybrid clouds
General hybrid cloud use case combines both private and public clouds
Private cloud hosts regular workload and master processes, also security and compliance critical applications/tasks
Public cloud hosts non-critical and not regular but computing intensive workload
Requires compatibility between private and public cloud platforms
Cloudbursting – term widely used by businesses to describe a situation when workload is temporarily migrated to cloud, extending and replicating the private cloud resources and VMs (using formula "buy the base, rent a spike")
Standards compliance
Applications must be designed for cloud and virtualisation



Приклади

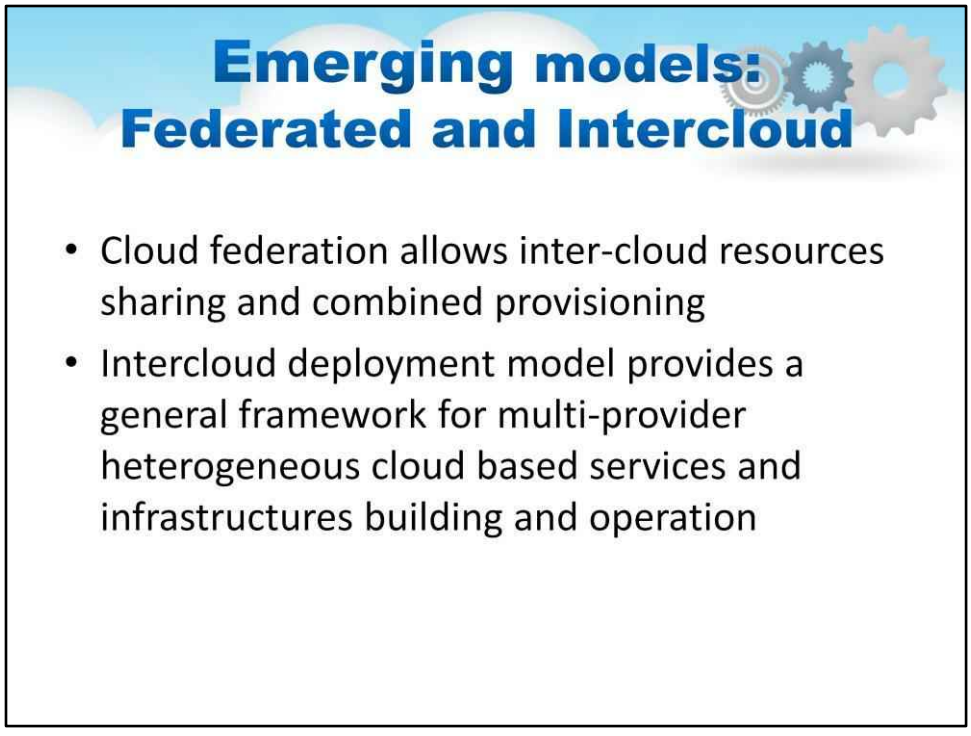

Examples
Microsoft
Hyper-V
DELL EMC
vmware
amazon
web services
rackspace

Хмара спільноти

[NIST SP800-145] Хмарна інфраструктура надається для ексклюзивного використання певною спільнотою споживачів з організацій, які мають спільні інтереси (наприклад, місія, вимоги безпеки, політика та міркування щодо відповідності). Він може належати, управлятися та управлятися однією чи декількома організаціями в спільноті, третьою стороною або деякою їх комбінацією, і він може існувати в приміщеннях або поза ними.

- Хмарна платформа спрощує створення та керування спільними ресурсами

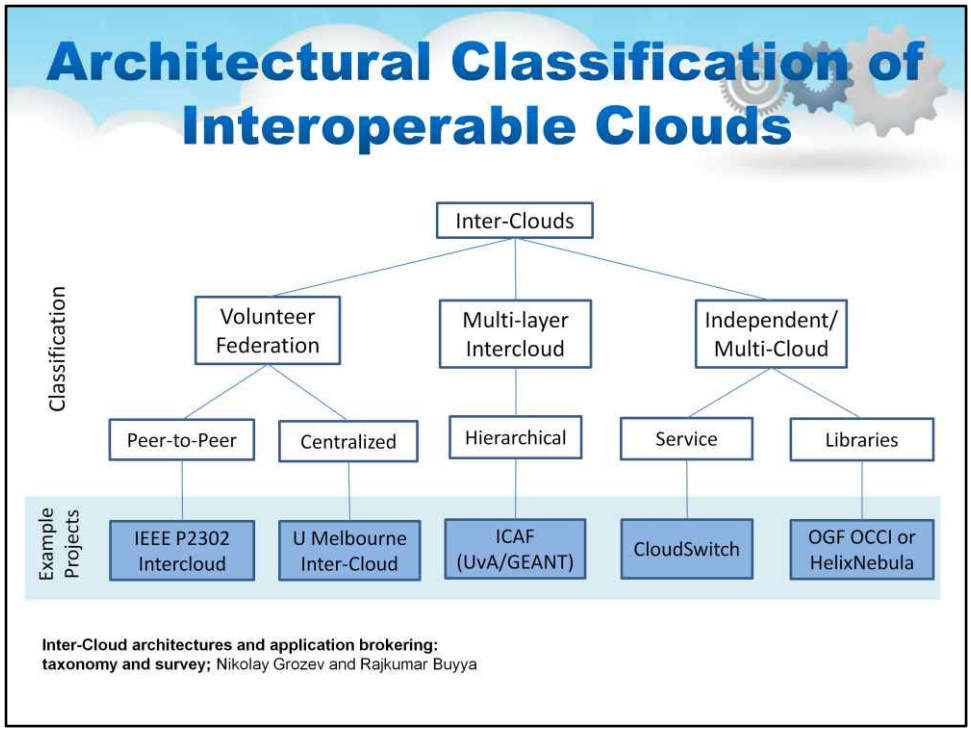
- Зазвичай вирішується за допомогою Federated Clouds і Federated Identity



Два типи хмарної федерації

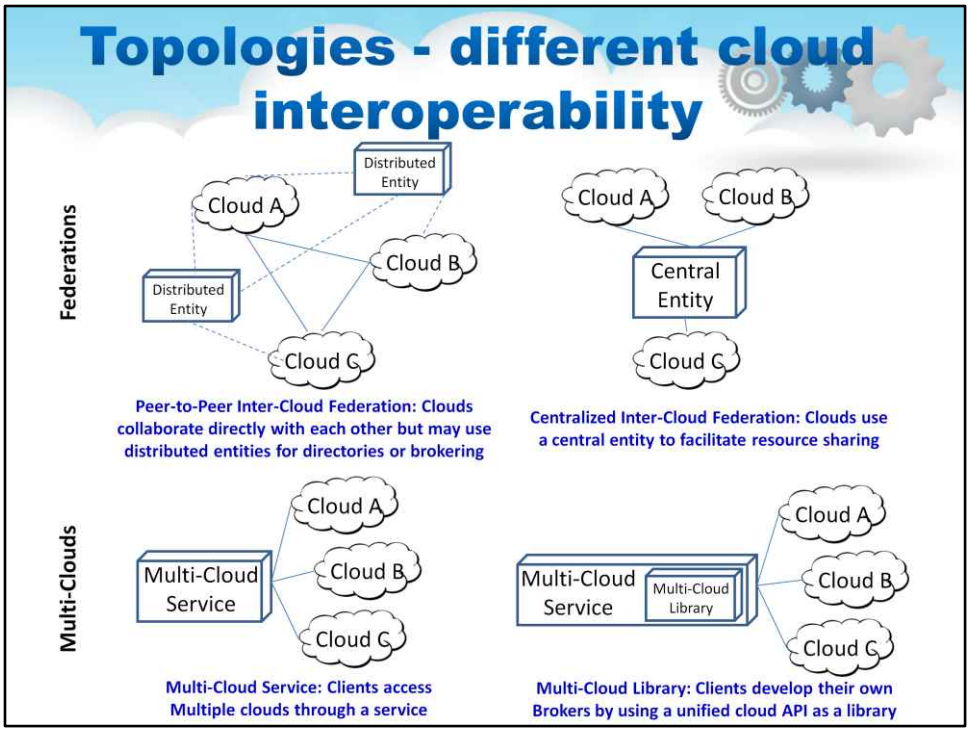
- Федерація на стороні постачальника для спільного використання ресурсів і надання
- Федерація на стороні користувача/клієнта, що дозволяє створювати різнорідні хмарні інфраструктури з кількома провайдерами

- Дозволяє ієрархічну хмарну інтеграцію IaaS-PaaS-SaaS
- Вимагає та можливо при використанні добре визначених хмарних стандартів
- Має розглядати питання: контроль і управління, управління міжхмарними об'єднаннями, питання операцій



Існує багато способів взаємодії хмар. На цьому слайді використано ілюстрацію дослідницької групи з Університету Мельбурна, яка відшліфувала статтю про таксономію різних підходів до хмарної взаємодії. Будь ласка, вивчіть ілюстрацію.

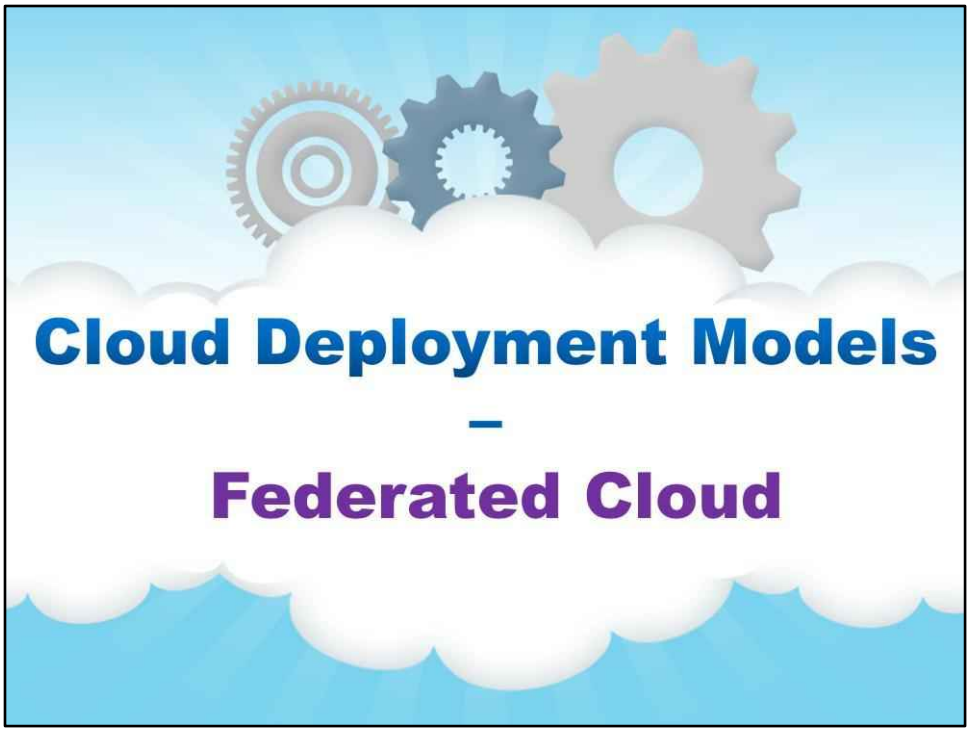
Intercloud Architecture Framework (ICAF) базується на багаторівневій моделі хмарних послуг (CMS) і розділяє контроль, управління, об'єднання та операційні аспекти в Intercloud. Розроблено компанією UvA, на даний момент реалізовано в європейському проекті GEANT.



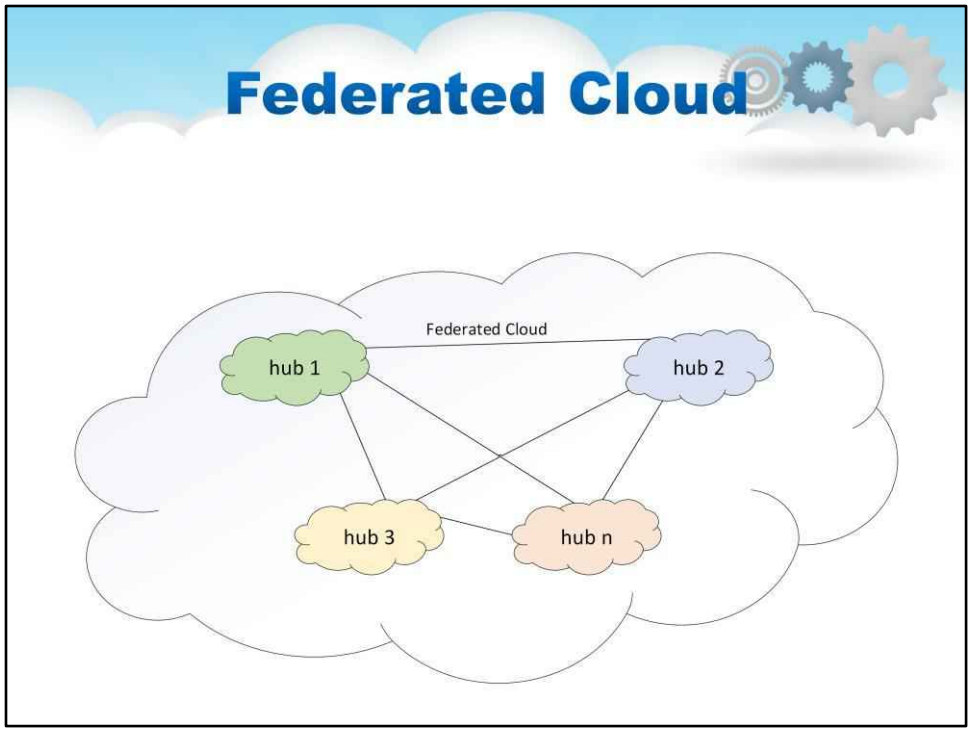
Легше зрозуміти ці різні підходи до хмарної сумісності, якщо візуалізувати отримані топології. Ілюстрація на цьому слайді показує ті самі чотири категорії, що й на попередньому слайді, але у вигляді топології.

Легко побачити, що багатохмарний підхід буде працювати для користувача, який хоче отримати доступ до а невелику кількість хмар більш-менш вручну і потрібно «з'єднати». ресурси з кожної хмари. Цей підхід ідеально підходить для науковців, які хочуть отримати доступ до простих хмарних ресурсів (наприклад, віртуальних машин) для створення великих кластерів для Hadoop або Grid-обчислень. У цьому випадку відповідальність за розуміння варіацій між хмарами лягає на користувача.

Легко побачити, що підхід Федерації є набагато більш масштабованою та взаємопов'язаною архітектурою. Цей підхід ідеально підходить для систем, які можуть охоплювати кілька хмарних провайдерів або охоплювати приватні та публічні хмари. Це також покладає відповідальність за розуміння відмінностей між хмарами на інфраструктуру, а не на кінцевого користувача. Це важливий момент, який ми розглянемо пізніше.

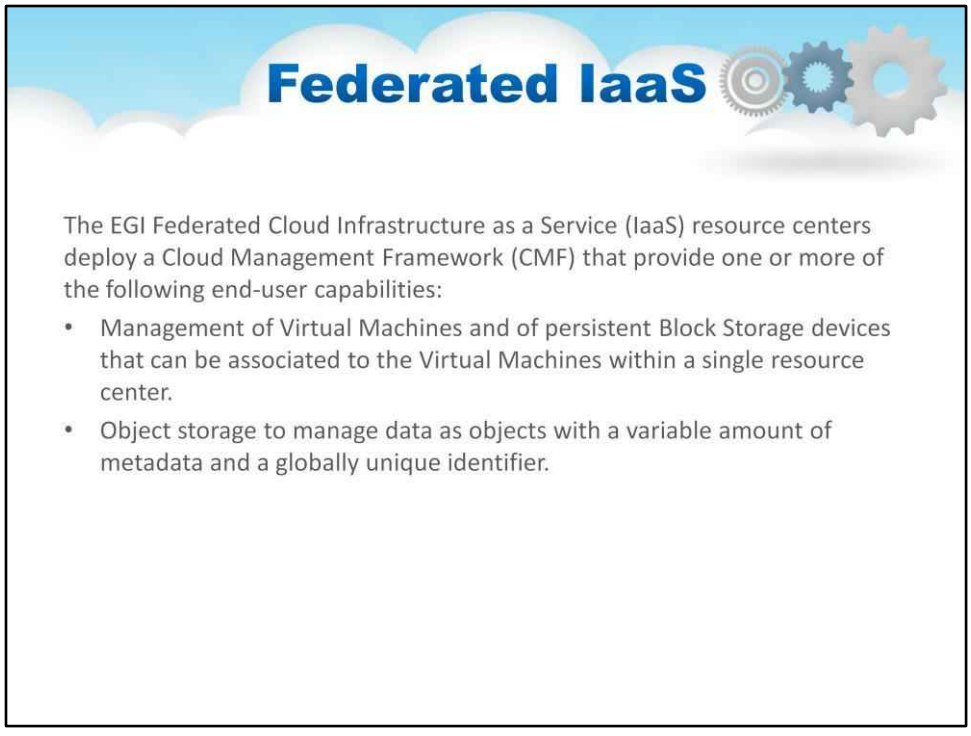


Федеративна хмара



Об’єднана хмара відноситься до бізнес-моделі, в якій кілька зовнішніх і внутрішніх хмарних обчислювальних служб розгортаються та досягають спільної мети.

Об’єднана хмара (також називається хмарною федерацією) — це розгортання зовнішніх і внутрішніх і керування нимипослуги хмарних обчислень відповідати потребам бізнесу. Федерація - цеоб'єднання кількох дрібн частини, які виконують спільну дію.



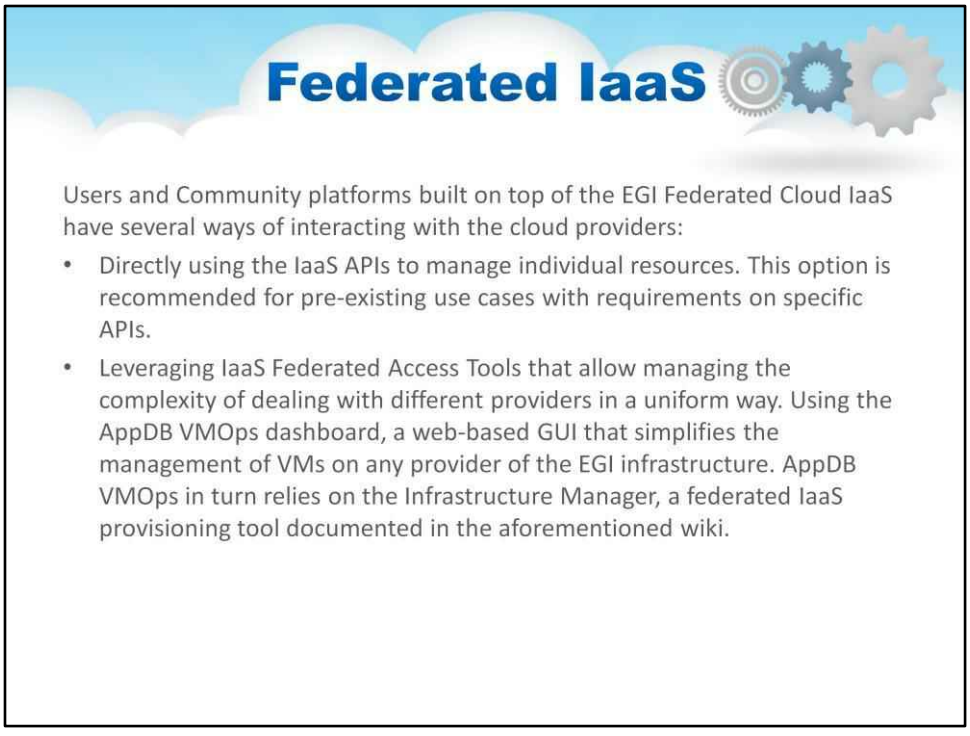
Ці можливості кінцевого користувача мають надаватися через узгоджені спільнотою API, які можна інтегрувати з такими службами EGI:

AAI для забезпечення єдиного входу для автентифікації та авторизації по всій хмарній федерації.

База даних конфігурації для запису інформації про топологію електронна інфраструктура.

Облік для збору, узагальнення та відображення інформації про використання.

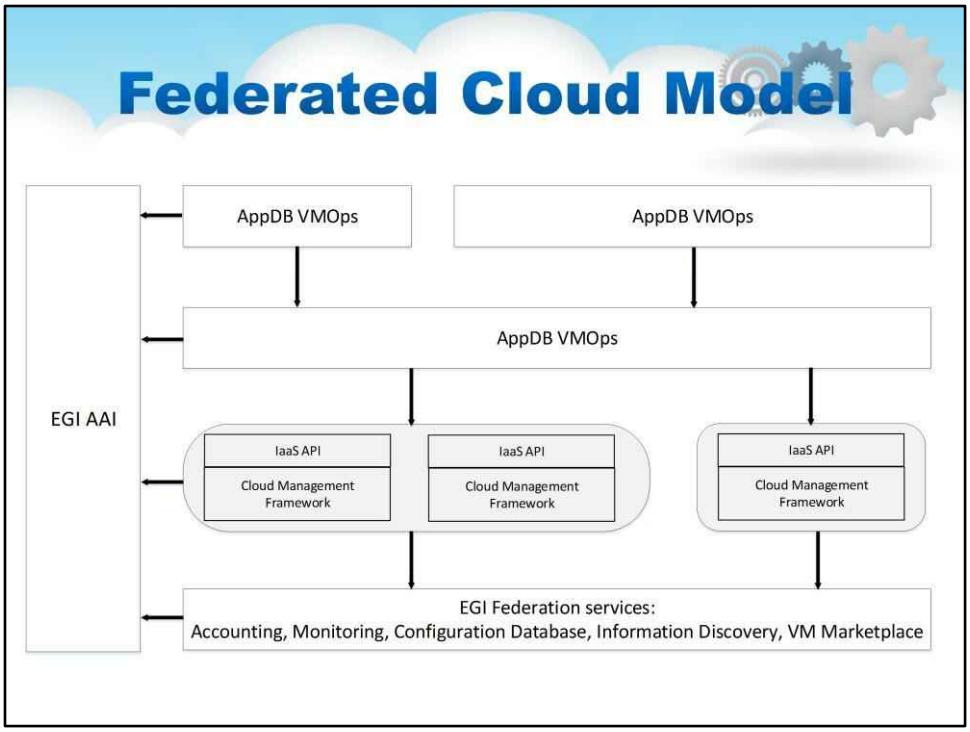
Моніторинг для моніторингу доступності об’єднаної служби та звітування про кінцеві точки розподіленої хмарної служби та отримання цієї інформації програмним шляхом. Інтеграція з моніторингом є пасивною діяльністю центру ресурсів, моніторинг виконується за допомогою API кінцевого користувача з обліковими даними звичайного користувача від EGI AAI.



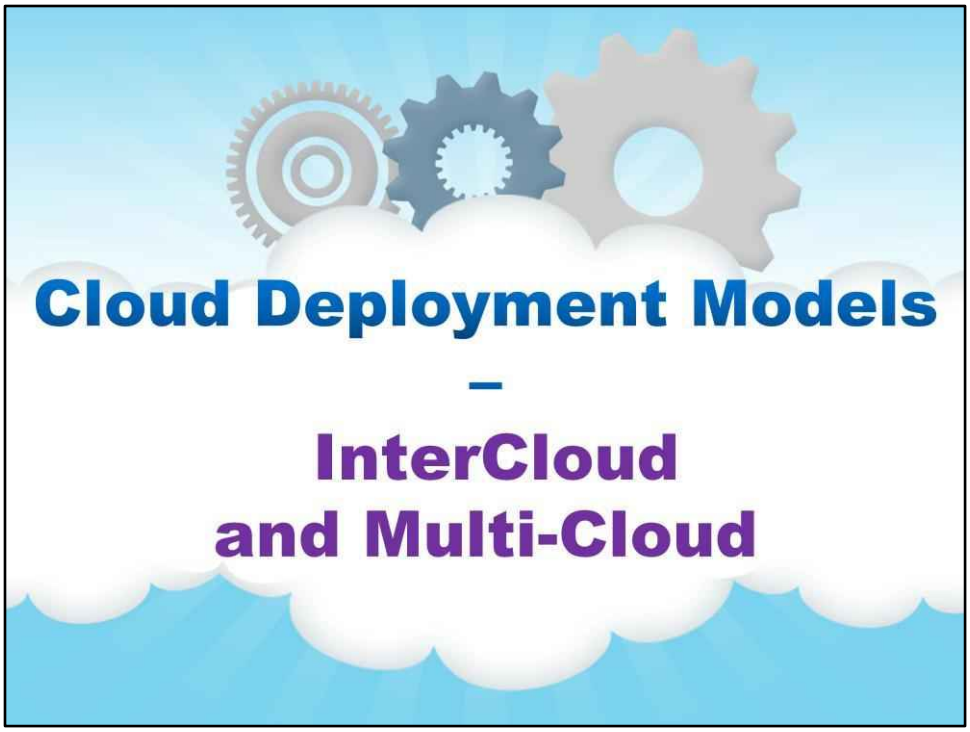
Ці інструменти включають

Системи забезпечення IaaS, які дозволяють визначати інфраструктуру як код, а також керувати та поєднувати ресурси від різних постачальників, таким чином забезпечуючи переносимість розгортання додатків між ними (наприклад, IM або Terraform); Хмарні брокери, які забезпечують підбір робочих навантажень для доступних постачальників (наприклад, INDIGO-DataCloud Orchestrator); і

Програмне забезпечення для керування хмарою, яке надає уніфіковану консоль для доступу до ресурсів і розгортання робочих навантажень відповідно до набору визначених користувачем встановлених політик (наприклад, Scalr або RightScale)



EGI не зобов’язує розгортати будь-яку конкретну або специфічну структуру керування хмарою; Ресурсний центр несе відповідальність за дослідження, визначення та розгортання рішення, яке найкраще відповідає їхнім індивідуальним потребам, одночасно гарантуючи, що запропоновані послуги реалізують необхідні інтерфейси та доменні мови сфер об’єднання, членами яких вони є.

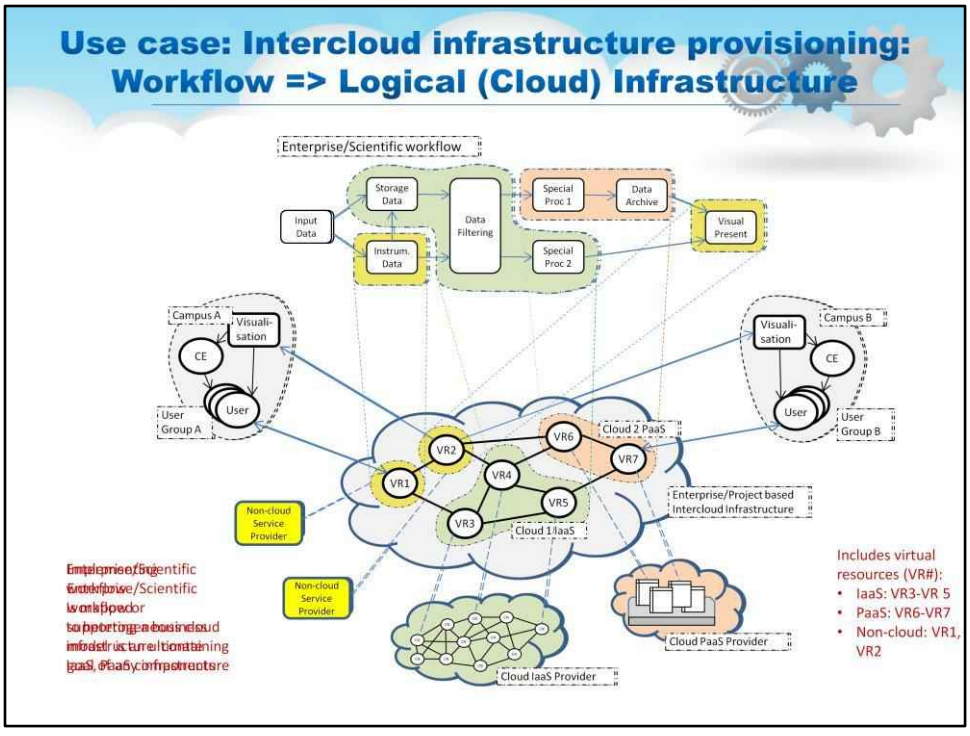


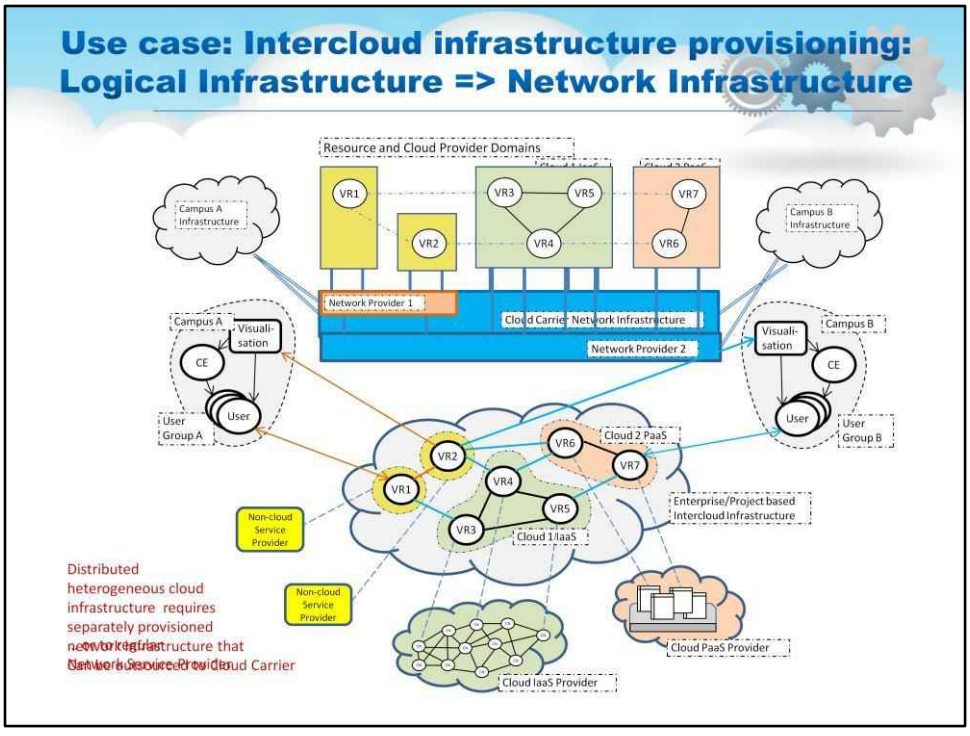
Inter Cloud i Multi-Cloud

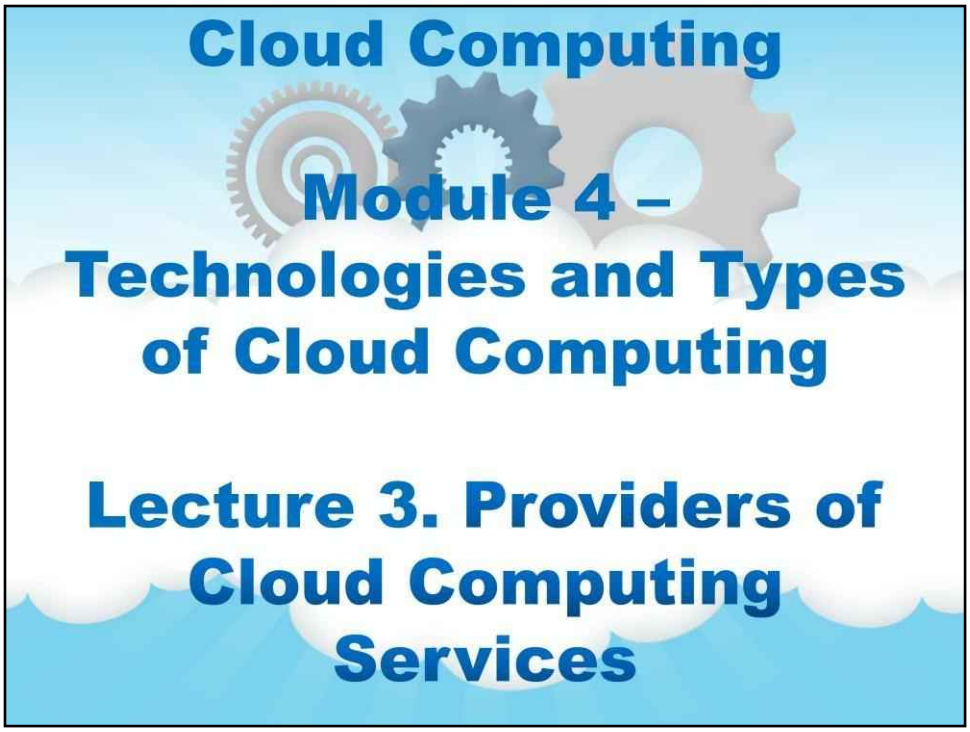
Use cases for Intercloud and Multi-cloud

Intercloud deployment model provides a basis for provisioning heterogeneous multi-provider cloud based project oriented infrastructures on-demand

- Hybrid multi-cloud project oriented collaborative environment
- Federated multi-cloud multi-provider infrastructure
  - Including legacy and non-cloud resources integration
- Enterprise/campus cloud infrastructure evolution and migration/portability
  - Business continuity when moving to a new provider
- Community clouds that may unite heterogeneous resources from multiple community members
- Infrastructure disaster recovery

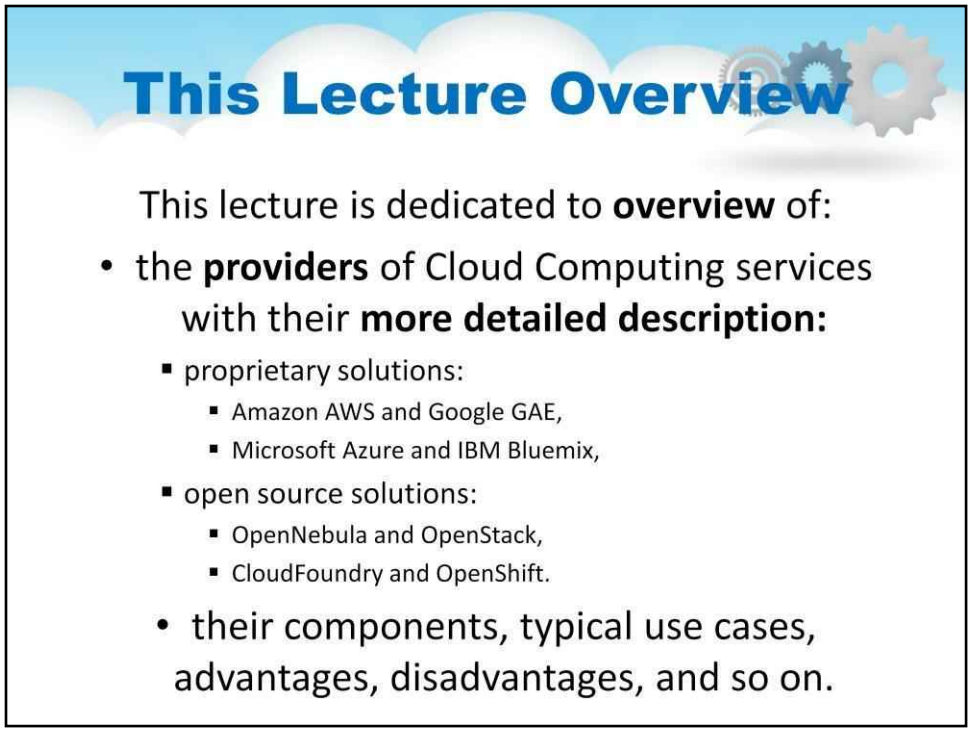

Use case: Intercloud infrastructure provisioning:
Workflow => Logical (Cloud) Infrastructure
Enterprise/Scientific workflow
Storage Data
Input Data
Instrum. Data
Data Filtering
Special Proc 1
Special Proc 2
Data Archive
Visual Present
Campus A
Visuali-sation
CE
User
User Group A
Campus B
Visuali-sation
CE
User
User Group B
VR1
VR2
VR3
VR4
VR5
VR6
VR7
Cloud 2 PaaS
Cloud 1 IaaS
Enterprise/Project based Intercloud Infrastructure
Non-cloud Service Provider
Non-cloud Service Provider
Cloud PaaS Provider
Cloud IaaS Provider
Includes virtual resources (VR#):
• IaaS: VR3-VR 5
• PaaS: VR6-VR7
• Non-cloud: VR1, VR2


Use case: Intercloud infrastructure provisioning:
Logical Infrastructure => Network Infrastructure
Resource and Cloud Provider Domains
Campus A Infrastructure
Campus B Infrastructure
VR1
VR2
VR3
VR4
VR5
VR6
VR7
Network Provider 1
Cloud Carrier Network Infrastructure
Network Provider 2
Campus A
Visuali-sation
CE
User
User Group A
Visuali-sation
Campus B
CE
User
User Group B
Cloud 2 PaaS
Enterprise/Project based Intercloud Infrastructure
Cloud 1 IaaS
Non-cloud Service Provider
Non-cloud Service Provider
Distributed heterogeneous cloud infrastructure requires separately provisioned network infrastructure that
Cloud Resource Provider
Cloud Carrier
Cloud PaaS Provider
Cloud IaaS Provider



Назва цього модуля: «Технології та види хмарних обчислень”.

Ця лекція 3 присвячена «Постачальникам послуг хмарних обчислень».



Лекція 3. Хмарні провайдери

Обчислювальні послуги

Ця лекція про:

в**провайдерів**послуг хмарних обчислень із**більш детальний опис**з**:**

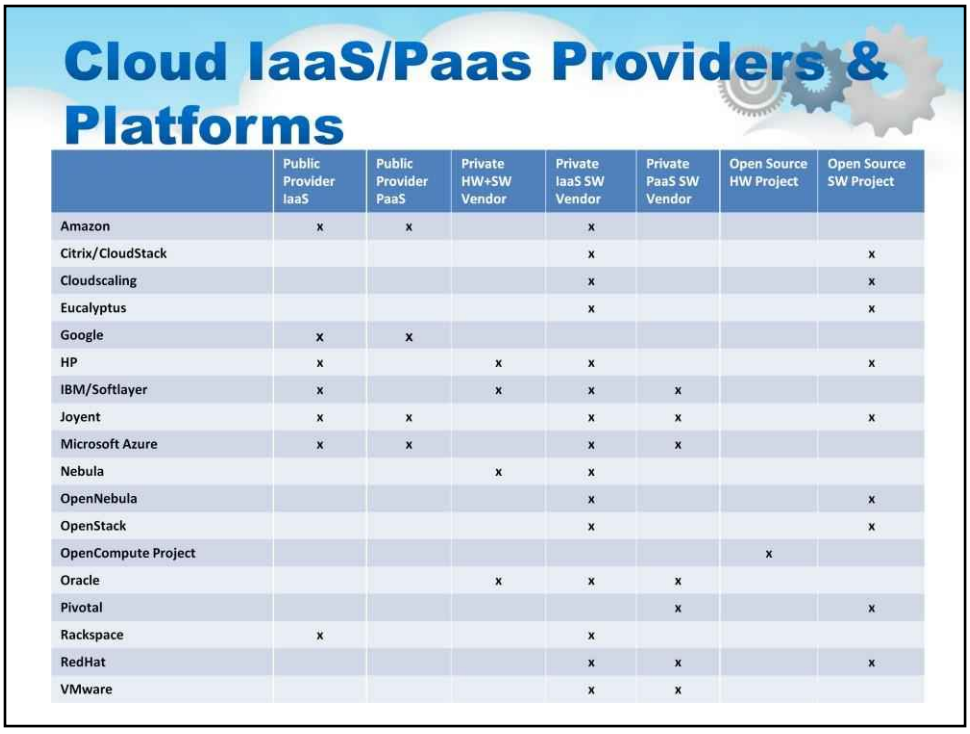
-власні (комерційні) рішення:
- Amazon AWS і Google GAE,
- Microsoft Azure і IBM Bluemix,

-рішення з відкритим кодом:
- OpenNebula і OpenStack,
- CloudFoundry і OpenShift.

Тут описані їхні компоненти, типові випадки використання, переваги, недоліки тощо в деталях.

Огляд

Почнемо з огляду постачальників хмарних послуг…



	Public Provider IaaS	Public Provider PaaS	Private HW+SW Vendor	Private IaaS SW Vendor	Private PaaS SW Vendor	Open Source HW Project	Open Source SW Project
Amazon	x	x		x			
Citrix/CloudStack				x			x
Cloudscaling				x			x
Eucalyptus				x			x
Google	x	x					
HP	x		x	x			x
IBM/Softlayer	x		x	x	x		
Joyent	x	x		x	x		x
Microsoft Azure	x	x		x	x		
Nebula			x	x			
OpenNebula				x			x
OpenStack				x			x
OpenCompute Project						x	
Oracle			x	x	x		
Pivotal					x		x
Rackspace	x			x			
RedHat				x	x		x
VMware				x	x		

У цій діаграмі перелічено основних гравців у сфері технологій хмарних обчислень. Незважаючи на те, що це неповно, очевидно, що існує велика кількість великих компаній, які інвестують значні дослідження та розробки в хмарні пропозиції.

На діаграмі показано різні способи доставки (загальнодоступні хмарні служби або програмне забезпечення для приватної хмари).

На діаграмі також показано рівні хмарних технологій (апаратне забезпечення, IaaS або PaaS)

Нарешті, діаграма вказує, чи надає постачальник свою пропозицію з відкритим вихідним кодом як ключову стратегію.

Примітка:

деякі постачальники SaaS із спеціальними середовищами PaaS, наприклад Salesforce.com, Netsuite або Workday, були виключені.

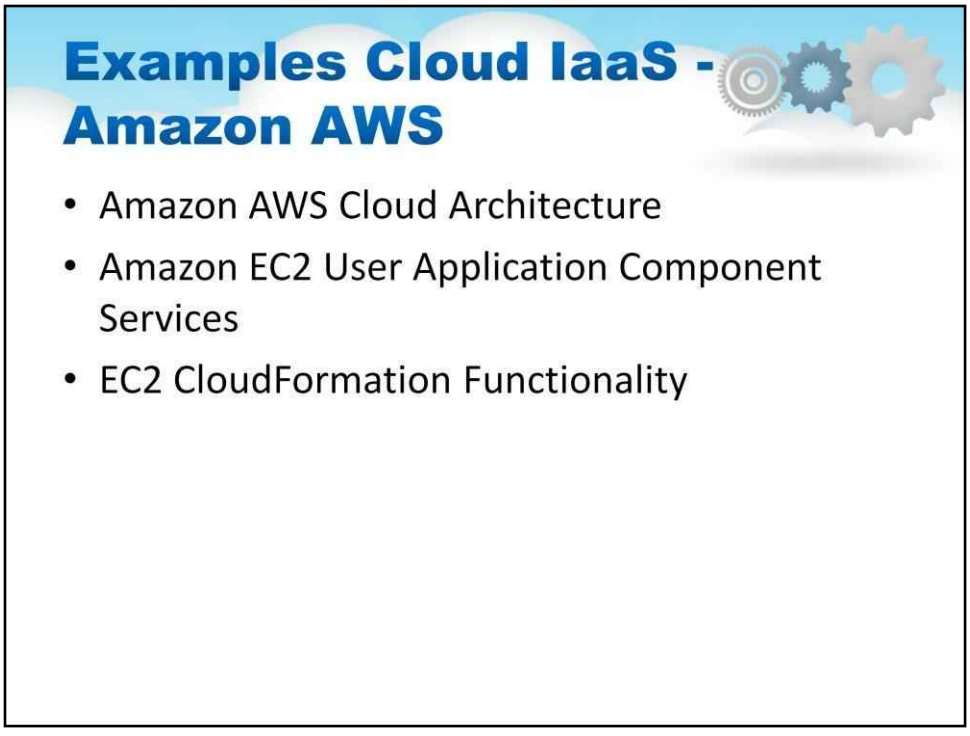
Тому вибір оптимального технологічного напрямку в цьому контексті може бути ускладнений!

Запатентовані рішення

Розглянемо деякі комерційні рішення...

Приклад: Amazon AWS

Перший приклад присвячено Amazon AWS, який є прикладом інфраструктури як послуги.



Найпопулярнішою хмарою IaaS є Amazon AWS. З цієї причини ми будемо посилатися на цю платформу, надаючи приклади рішень.

Архітектура AWS містить багато, багато сервісів. Перші та найосновніші з них — це обчислення «EC2» і відповідні основні частини, які постачаються з ними (наприклад, сховище S3 і EBS).

Тому давайте досліджуватимемо AWS докладніше.

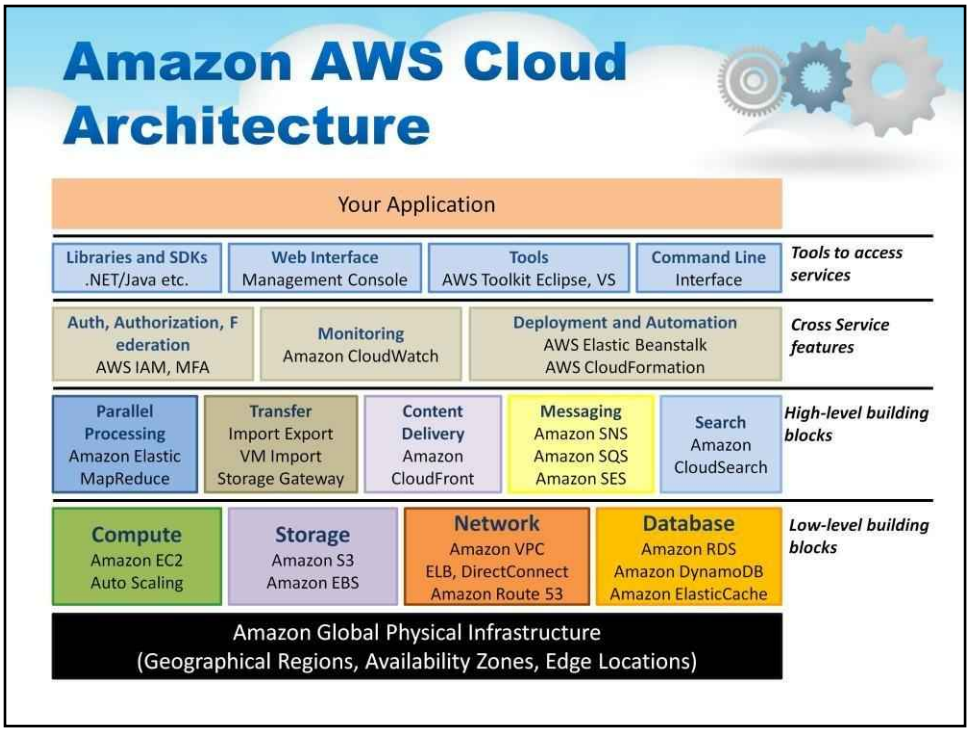
AWS пропонує дві основні послуги

Elastic Compute Cloud (EC2), що забезпечує змінний розмір обчислювальної ємності Simple Storage Service (S3)

AWS також пропонує багато інших послуг, наприклад, пропонуються всі види баз даних, багато видів віртуальних машин і компонентів проміжного програмного забезпечення, все «як послуга».

Amazon EC2 і S3 API стали стандартними де-факто інтерфейсами для доступу до хмарних сервісів і керування ними

AWS має кілька зон доступності на вибір.



Це «концептуальна архітектура» AWS.
Ця ілюстрація є зручним способом отримати враження про рівні та групи різних сервісів AWS.

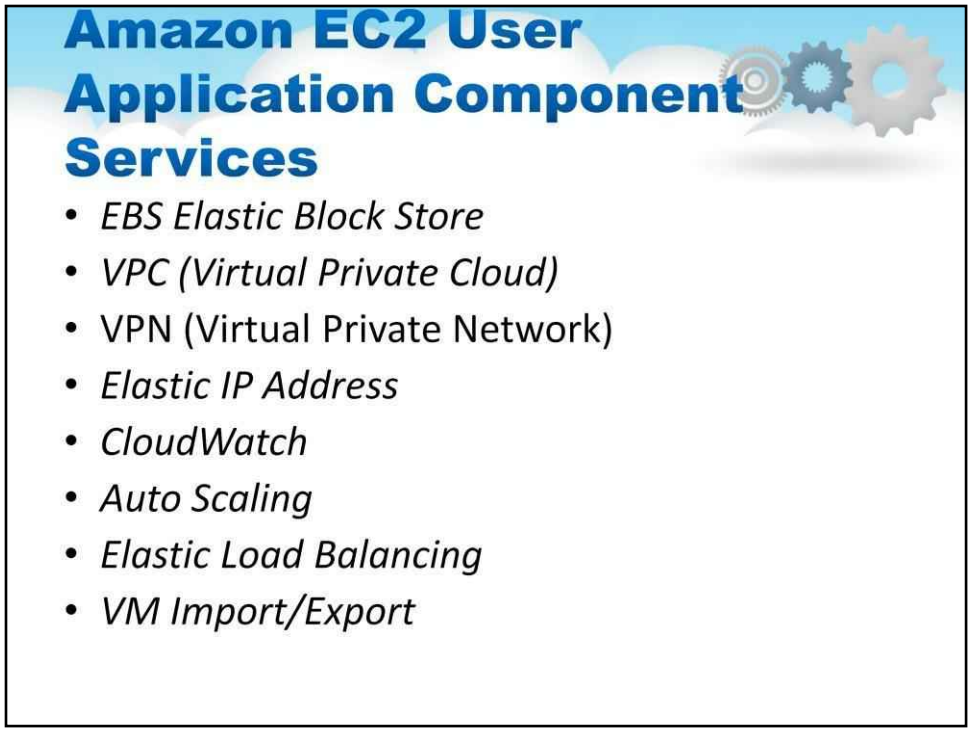
Примітка:

Будівельні блоки низького рівнямістять основні функції AWS, а також параметри в інших основних функціях, таких як DNS, мережа та кеш.

Будівельні блоки вищого рівнянещодавно побудовані та реалізов

функції вищого рівня, такі як пошук і доставка вмісту.
Перехресні сервісні функціївключають механізми безпеки, такі як автентифікація, моніторинг і контроль програм.

AWS дозволяє отримати до нього доступ через оригінальні низькорівневі команди та інтерфейси до наданих інструментів.



Цей слайд ілюструє хмарну архітектуру AWS, яку можна розглядати як еталонну хмарну реалізацію IaaS. Він містить усі основні функціональні компоненти загальної архітектури Cloud IaaS.

Магазин еластичних блоків (EBS)забезпечує високодоступні та надійні томи зберігання, які є незалежними та постійними протягом усього терміну служби віртуальної машини/AMI

Ці томи можна використовувати як завантажувальний розділ для певного екземпляра або як простір для зберігання з серверною реплікацією для довгострокової надійності

Віртуальна приватна хмара (VPC)дозволяє організаціям використовувати ресурси AWS разом із наявною інфраструктурою в а**VPN (віртуальна приватна мережа)**збільшити обчислювальні можливості за межі локальних ресурсів.

Еластична IP-адресаце постійна IP-адреса, яку можна призначити певному обліковому запису користувача та дає змогу користувачеві динамічно призначати її будь-якій віртуальній машині користувача.

Може бути налаштований для перемикання на замінний екземпляр у разі збою запущеної віртуальної машини

CloudWatchце сервіс для моніторингу використання ресурсів AWS, операційної продуктивності та загальних моделей попиту

Автоматичне масштабуваннядозволяє користувачам динамічно надавати та відмовлятися від ресурсів, які використовуються їхніми програмами, залежно від вимоги під час виконання

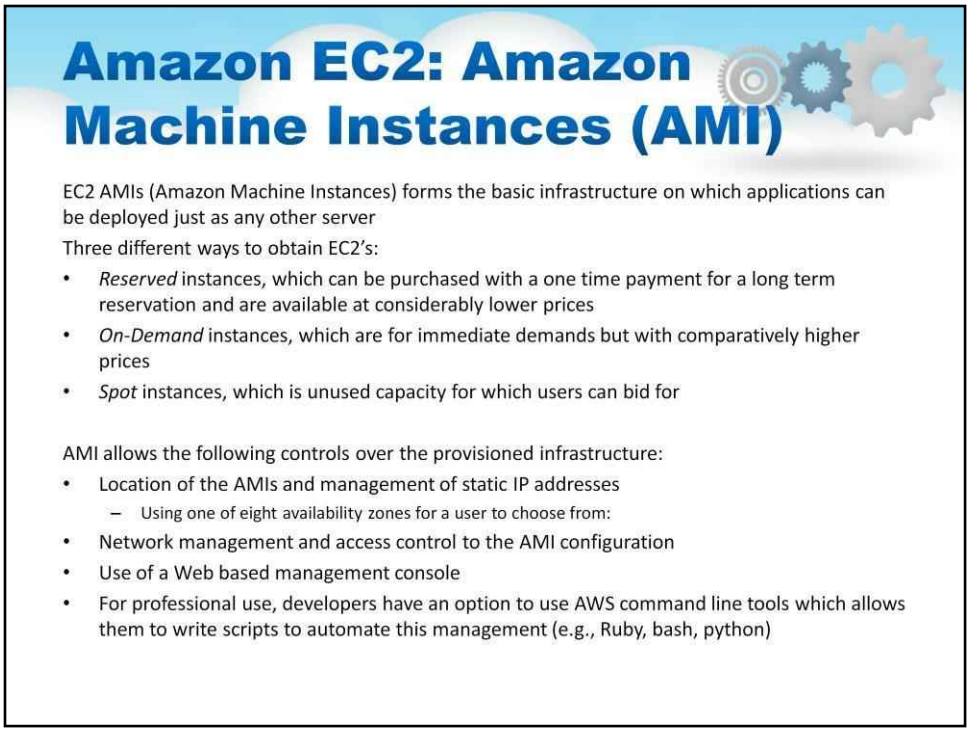
Щоб впоратися з раптовими стрибками попиту без необхідності попереднього надлишку ресурсів

Еластичний баланс навантаженняможе збалансувати навантаження між кількома віртуальними машинами, розташованими в одній зоні доступності або кількох зонах

Можна використовувати для створення відмовостійкої системи, яка відстежує працездатність кожної віртуальної машини та розподіляє навантаження між активними екземплярами

Імпорт/експорт VMце служба, яка надає функціональні можливості для завантаження користувацьких зображень віртуальної машини та збереження живого екземпляра як зображення

Економить зусилля для створення спеціального екземпляра віртуальної машини (включаючи компоненти інфраструктури, програми, функції безпеки та відповідності тощо).

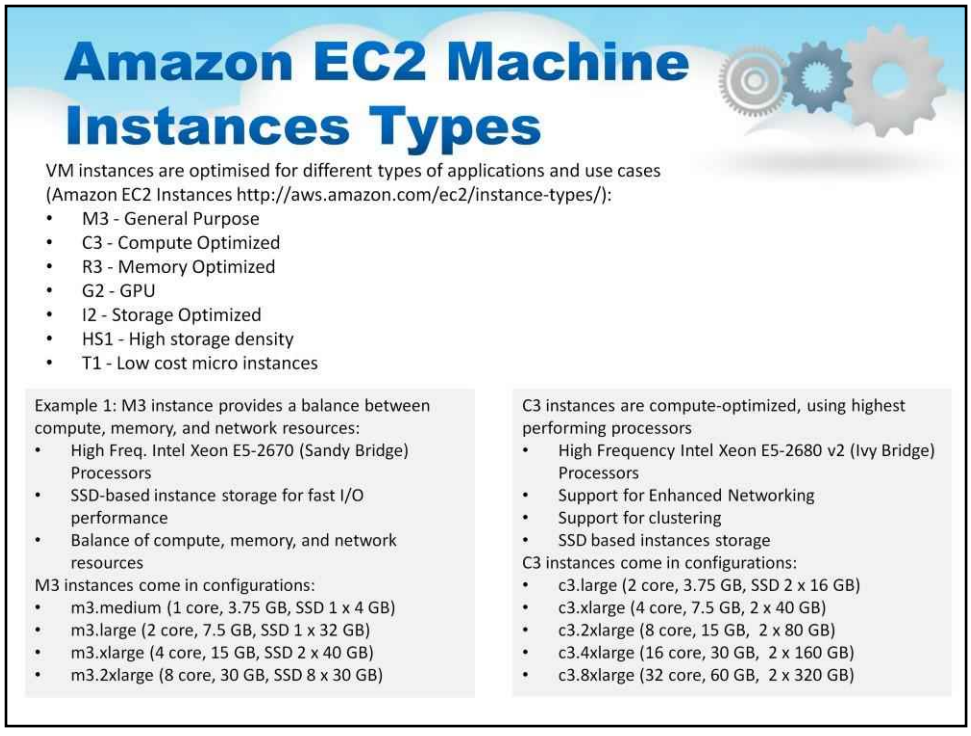


Давайте тепер уважніше розглянемо EC2.

EC2 AMI (екземпляри машини Amazon) формує базову інфраструктуру, на якій можна розгортати програми, як і будь-який інший сервер

На слайді наведено три різні способи отримання EC2. Щоб було зрозуміло, щойно ви отримуєте певний розмір EC2, усі вони «однакові», немає ніякої суттєвої різниці між «маленьким», отриманим за допомогою Spot або Reserved.

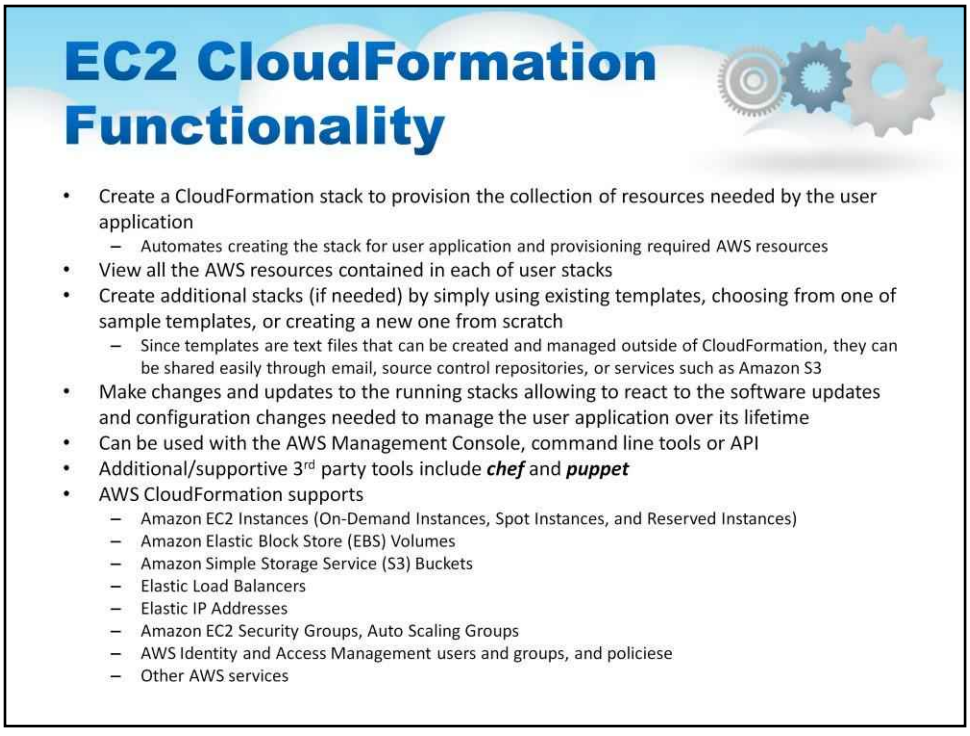
AMI дозволяють використовувати кілька елементів керування над наданою інфраструктурою. На слайді перелічено ці елементи керування. Їх повинно бути мінімум для «максимальної простоти» у використанні.



Amazon має багато типів машин.
Це «типи» та розміри атрибутів EC2, які можна «замовити».

Примітка: правила іменування дозволяють зрозуміти конфігурацію, що стоїть за іменами.

Примітка: не існує «стандартної пружної обчислювальної одиниці» або іншого реального вимірювання ємності. Незважаючи на те, що різні типи машин досить чітко описують те, з чого вони складаються, будь-яке поняття похідної продуктивності чи стандартизованого тесту не повинно бути видно.

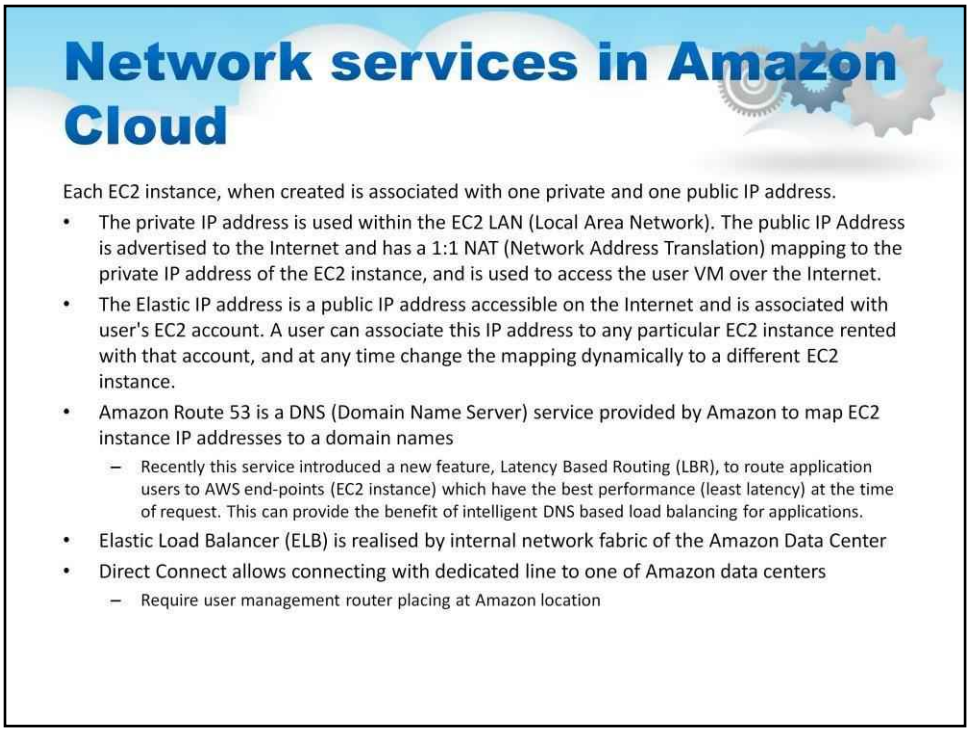


CloudFormation — це система (або насправді інструмент), яку AWS надає як опцію, щоб допомогти з автоматизацією ініціалізації. Він потребує специфікації ресурсів, необхідних для кожного елемента в колекції елементів, які складають додаток.

На цьому слайді детально описано деякі функції Cloud Formation.

Це дуже зручний набір інструментів для створення та підтримки розгортань AWS.
Він є власністю Amazon.

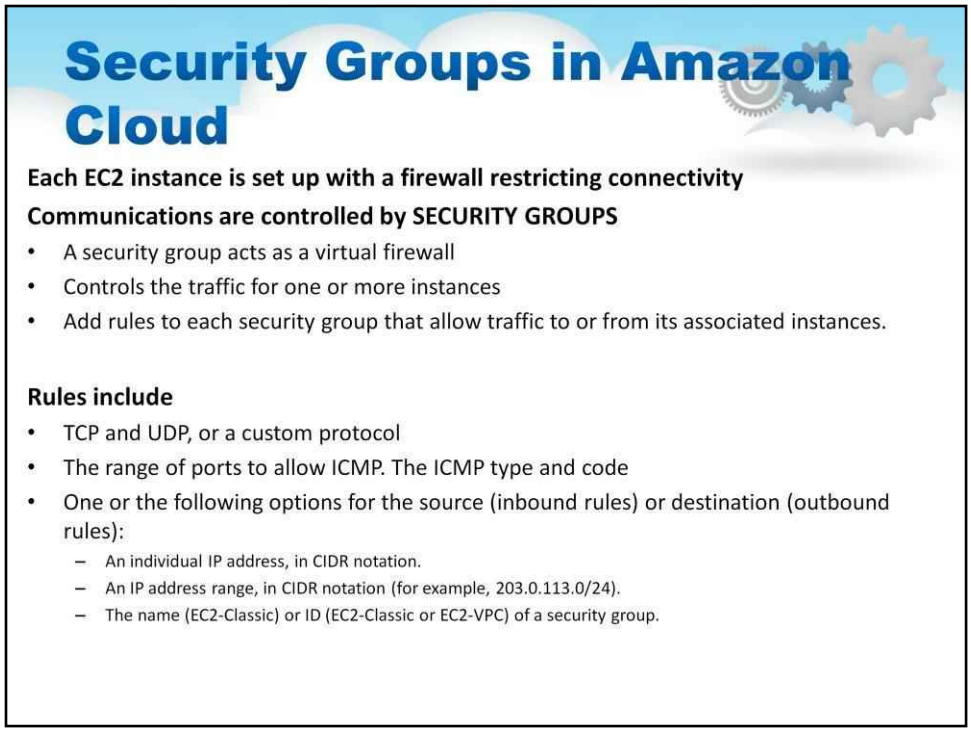
Хоча є певні зусилля («тепловий» проект OpenStack) відтворити сумісну систему як портативну реалізацію з відкритим вихідним кодом, розробник повинен розуміти, що цей набір інструментів недоступний ніде, крім AWS. З цієї причини багато розробників використовують Chef або Puppet якомога частіше, а потім інтерфейсують їх із CloudFormation.



Кожна система IaaS має особливу «особистість» щодо того, як вона працює, і одним із головних аспектів цієї особистості є те, як налаштовано мережу для віртуальної машини.

AWS використовує дуже специфічні мережеві налаштування, у яких відсутні багато мережевих функцій (наприклад, багатоадресна передача або VLAN).

На цьому слайді детально описано мережевий контекст AWS.



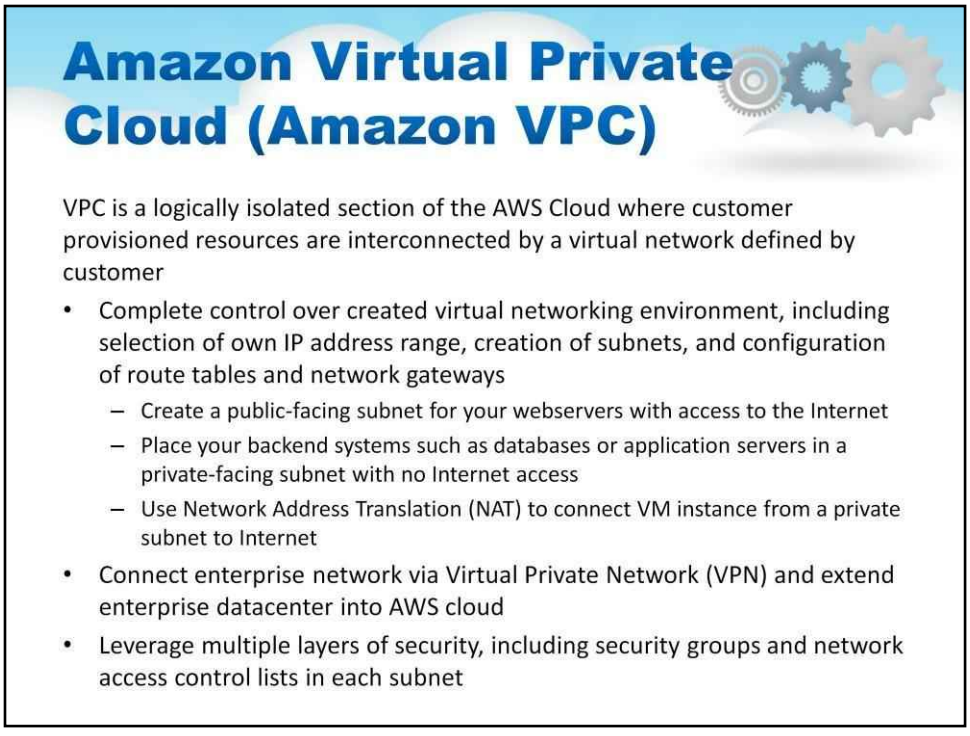
Правила групи безпеки контролюють вхідний трафік, якому дозволено досягати екземплярів, пов’язаних із групою безпеки, і вихідний трафік, якому дозволено залишати їх.

За замовчуванням групи безпеки дозволяють весь вихідний трафік.

- Ви можете будь-коли додавати та видаляти правила.
- Через короткий проміжок часу ваші зміни автоматично застосовуються до екземплярів, пов’язаних із групою безпеки.
- Ви можете змінити існуюче правило в групі безпеки або видалити його та додати нове правило.
- Ви можете скопіювати правила з існуючої групи безпеки до нової групи безпеки.
- Ви не можете змінити вихідні правила для EC2-Classic.

Правила групи безпеки завжди є дозвільними; ви не можете створити правила, які забороняють доступ. Для кожного правила ви вказуєте наступне:

- Дозволений протокол (наприклад, TCP, UDP або ICMP): TCP і UDP або спеціальний протокол.
- Діапазон дозволених портів ICMP, тип і код ICMP
- Один або наведені нижче параметри для джерела (правила вхідних повідомлень) або призначення (правила вихідних повідомлень), як описано на цьому слайді.



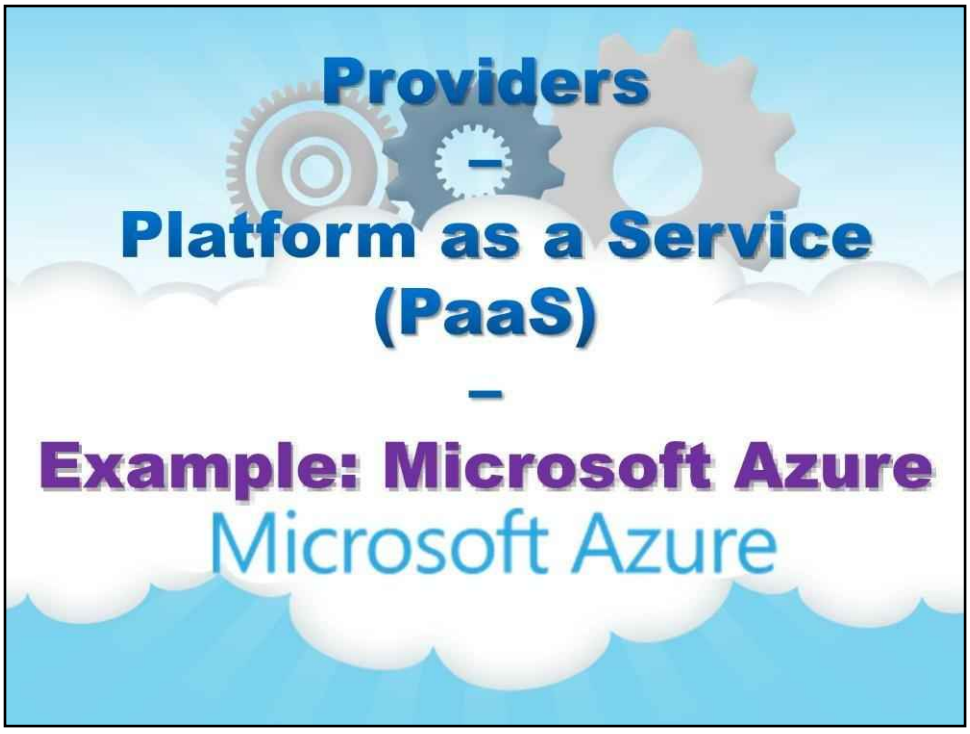
Віртуальна приватна хмара Amazon (Amazon VPC) — це логічно ізольований розділ хмари AWS, де ви можете запускати ресурси AWS у визначеній вами віртуальній мережі.

VPC дозволяє**повний контроль над створеним віртуальним мережевим середовищем**, включаючи вибір власного діапазону IP-адрес, створення підмереж і налаштування таблиць маршрутів і мережевих шлюзів.

> Наприклад. Ви можете створити загальнодоступну підмережу для своїх веб-серверів із доступом до Інтернету. Тоді ви можете створити приватну підмережу, де ви зможете розмістити свої серверні системи, такі як бази даних або сервери додатків, без доступу до Інтернету.

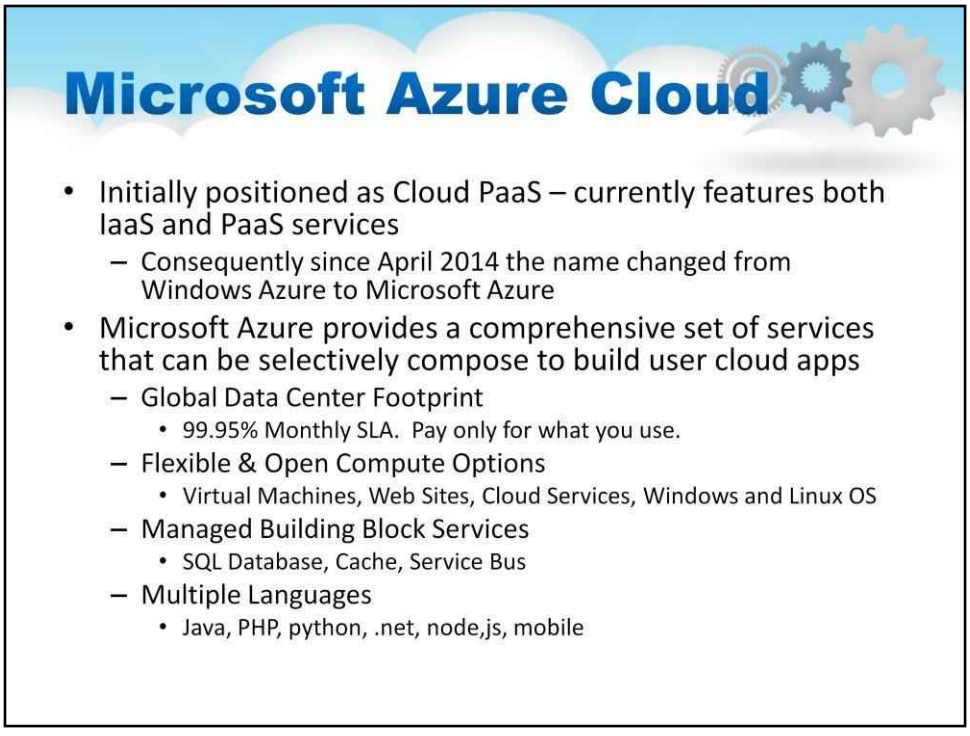
VPC дозволяє**підключити корпоративну мережу через віртуальну приватну мережу (VPN)**і розширити корпоративний центр обробки даних у хмару AWS

VPC дозволяє**використовувати кілька рівнів безпеки**включаючи групи безпеки та списки контролю доступу до мережі в кожній підмережі.



Приклад: Microsoft Azure

Другий приклад присвячено Microsoft Azure, який представляє приклад платформи як послуги (PaaS).



Хмара Microsoft Azure була спочатку випущена як Windows Azure і була суто платформою PaaS, спрямованою на надання хмарних послуг розробникам Windows. Після додавання можливостей IaaS, включаючи можливість запускати віртуальні машини Linux, Azure став важливим гравцем у Cloud.

Microsoft Azure надає повний набір служб, які можна вибірково створювати для створення хмарних програм користувача, наприклад:

Глобальний центр обробки даних

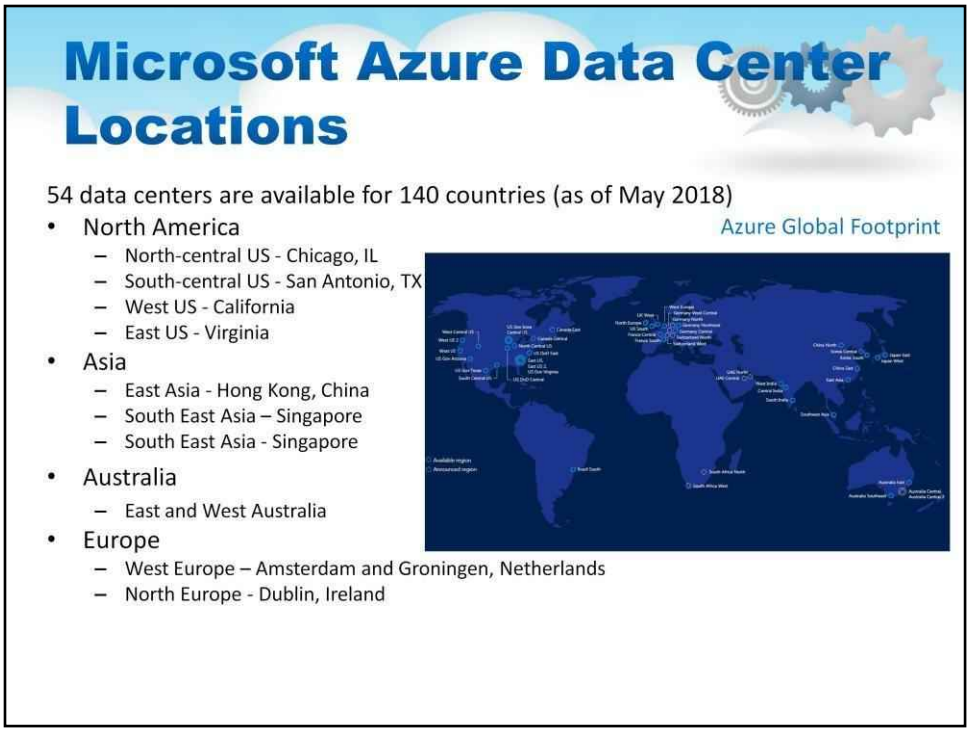
99,95% щомісячного SLA. Платіть лише за те, що використовуєте.

Гнучкі та відкриті параметри обчислень

Віртуальні машини, веб-сайти, хмарні служби, служби керованих будівельних блоків ОС Windows і Linux

База даних SQL, кеш, шина обслуговування Кілька мов

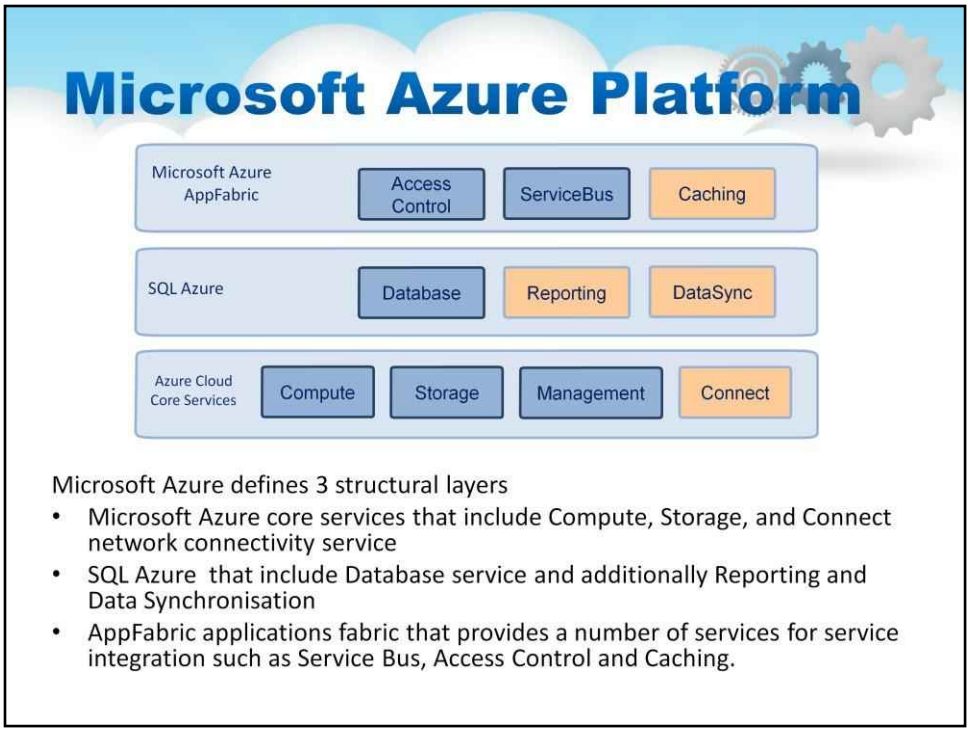
Java, PHP, python, .net, node, js, mobile



Корпорація Майкрософт займає одну з трьох найбільших обчислювальних машин, враховуючи всіх постачальників хмарних технологій (Microsoft, Google і Amazon).

Зараз у всьому світі встановлено 54 масивні центри обробки даних, які доступні в 140 країнах. Корпорація Майкрософт надає багато послуг зі своєї хмари, починаючи від Azure, Xbox Live, Outlook.com, Windows Update і будь-яку програму SaaS між ними.

Отже, вони справді інвестували в глобальну інфраструктуру.



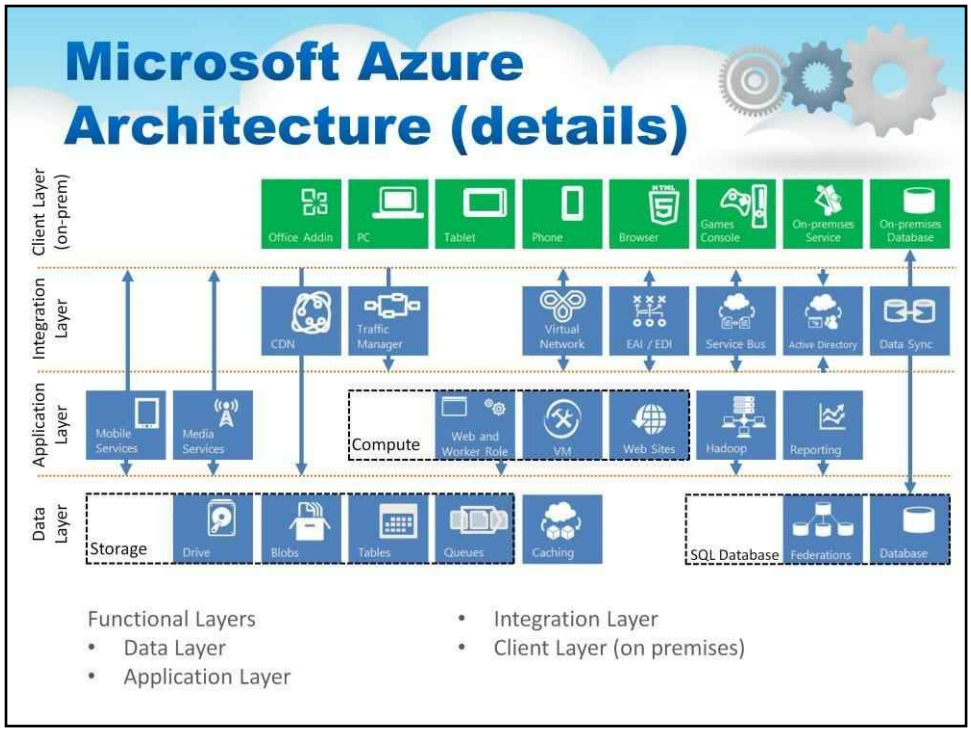
Платформа Azure за своєю структурою майже схожа на інші хмарні платформи, за винятком того, що служби даних і додатків дуже тісно інтегровані з основними хмарними службами, що лежать в основі.

Слайд ілюструє структурні рівні Microsoft Azure 3:

Основні служби Microsoft Azureвключає службу підключення до мережі Compute, Storage та Connect.

SQL Azureвключає службу бази даних, а також звітність і синхронізацію даних.

Програми Microsoft AppFabricнадає низку служб для інтеграції послуг, таких як шина обслуговування, контроль доступу та кешування.

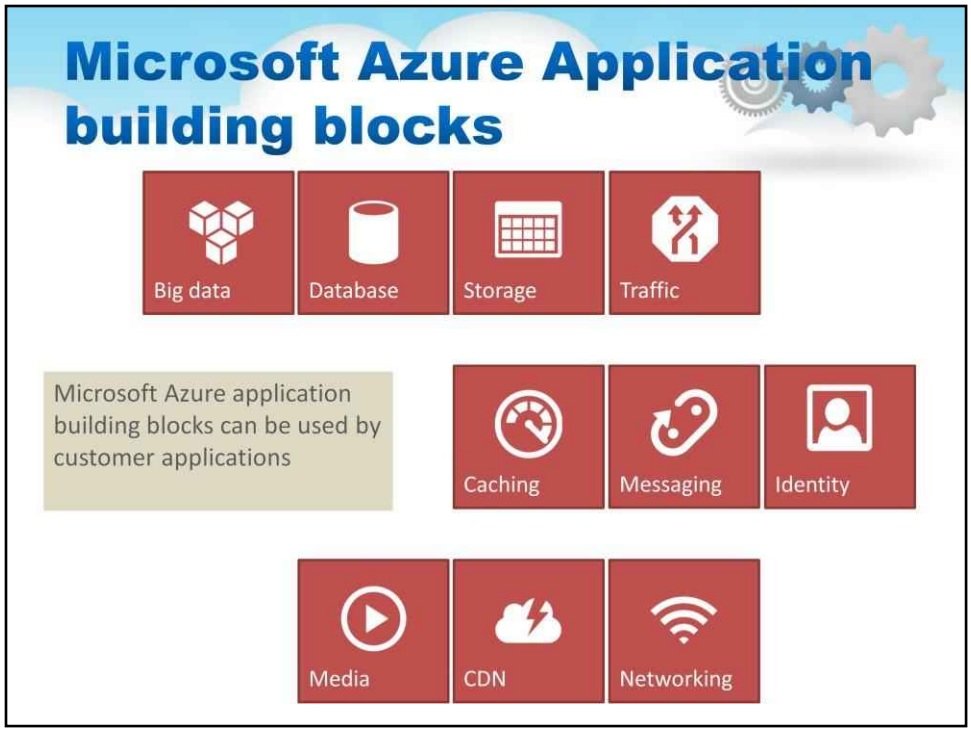


Цей слайд ілюструє «ринкову архітектуру», тобто ринкову архітектуру Microsoft Azure.

Можна побачити, що він організований у кілька функціональних рівнів:

-Рівень даних-
-Рівень програми-
-Інтеграційний рівень-
-Клієнтський рівень (локальний)-

Як видно з цього зображення, кожен рівень містить значну кількість функціональних можливостей.



Однією з інших переваг Microsoft Azure є можливість використання ряду будівельних блоків додатків.

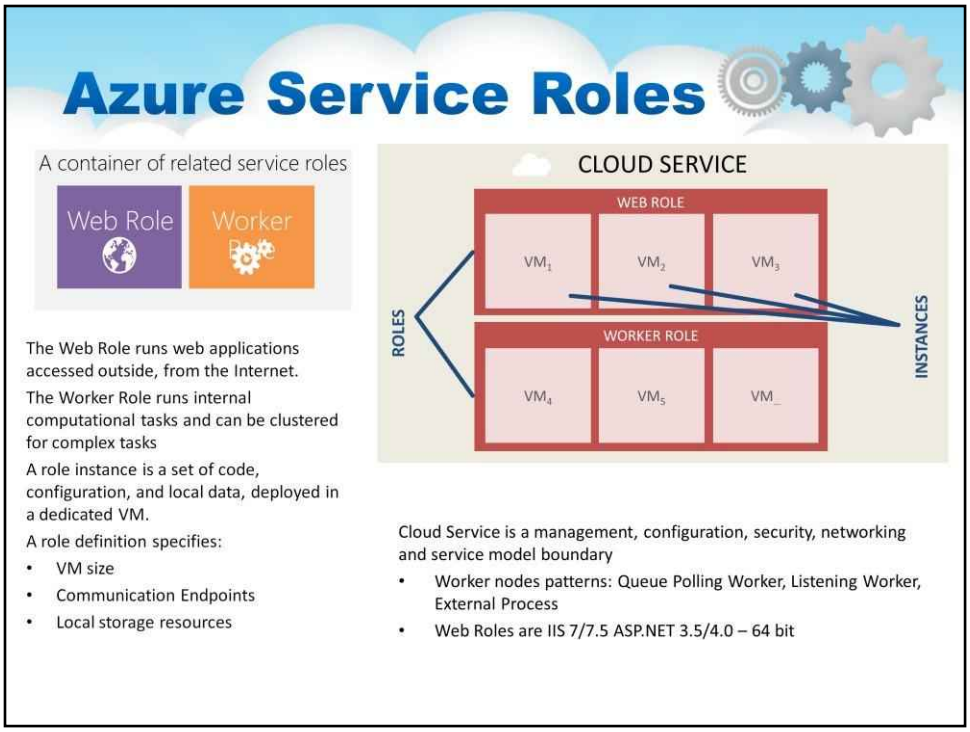
Будівельні блоки програми– це керовані служби, які Microsoft запускає, щоб забезпечити високу цінність, щоб розробник міг уникнути підтримки інфраструктури для загальних можливостей.

Розробник може створювати віртуальні машини і розміщувати туди все, що завгодно.

У багатьох випадках розробник виявить, що корпорація Майкрософт має вбудовані служби, які надаються безпосередньо або надаються її партнерами.

Розробник може використовувати будь-яку з цих служб із віртуальною машиною, веб-сайтом або хмарною службою, тому є гнучкість у способах їх використання.

Слайд ілюструє різноманітність цих будівельних блоків програми.



Ключовим поняттям у середовищі Azure PaaS є роль служби.

Службові роліздебільшого визначають, який тип трафіку спрямовано на цю службу,

Наприклад,**Веб-роль**запускає веб-програми, доступ до яких здійснюється ззовні, з Інтернету. Усі зовнішні з’єднання проходять через сутності ролі веб-служби. Балансування навантаження (властиве Azure) розподіляє вхідний трафік між об’єктами ролей веб-служб.

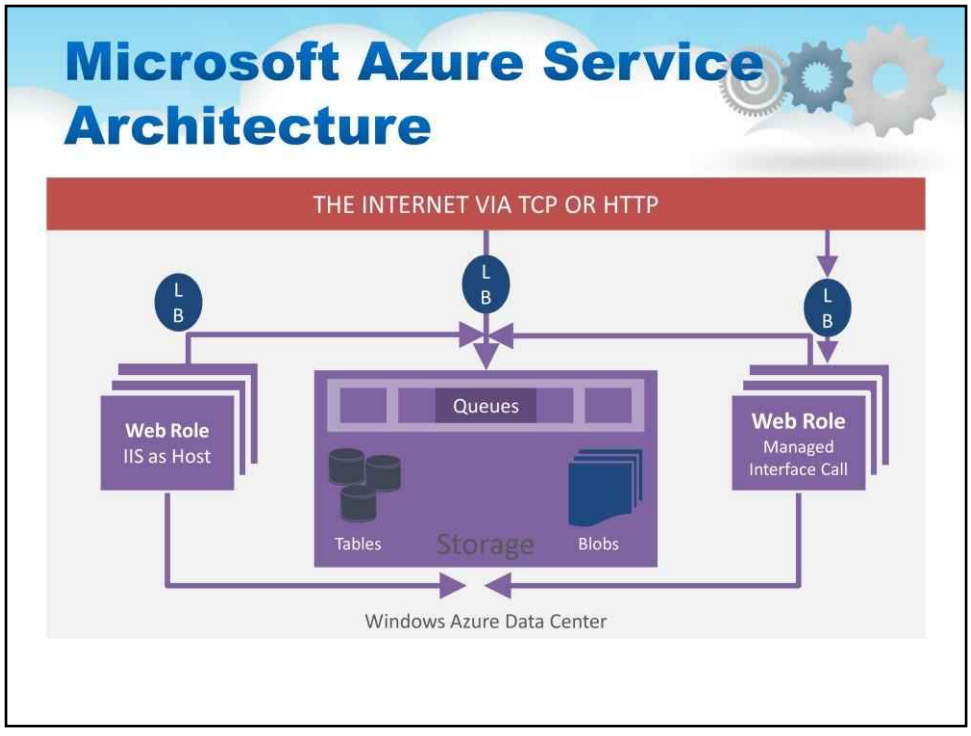
Але інше**Роль працівника**виконує внутрішні обчислювальні завдання та може бути кластеризовано для складних завдань. Підключення об’єктів робочої ролі обмежено лише внутрішнім трафіком (до/від інших об’єктів робочої ролі або до/від об’єктів веб-ролі)

А**рольовий екземпляр**це набір коду, конфігурації та локальних даних, розгорнутих у виділеній віртуальній машині.

Визначення ролі визначає:

- Розмір VM
- Кінцеві точки зв'язку
- Локальні ресурси зберігання

Хмарна служба – це межа моделі керування, конфігурації, безпеки, мережі та обслуговування, що складається з групи ролей служби.

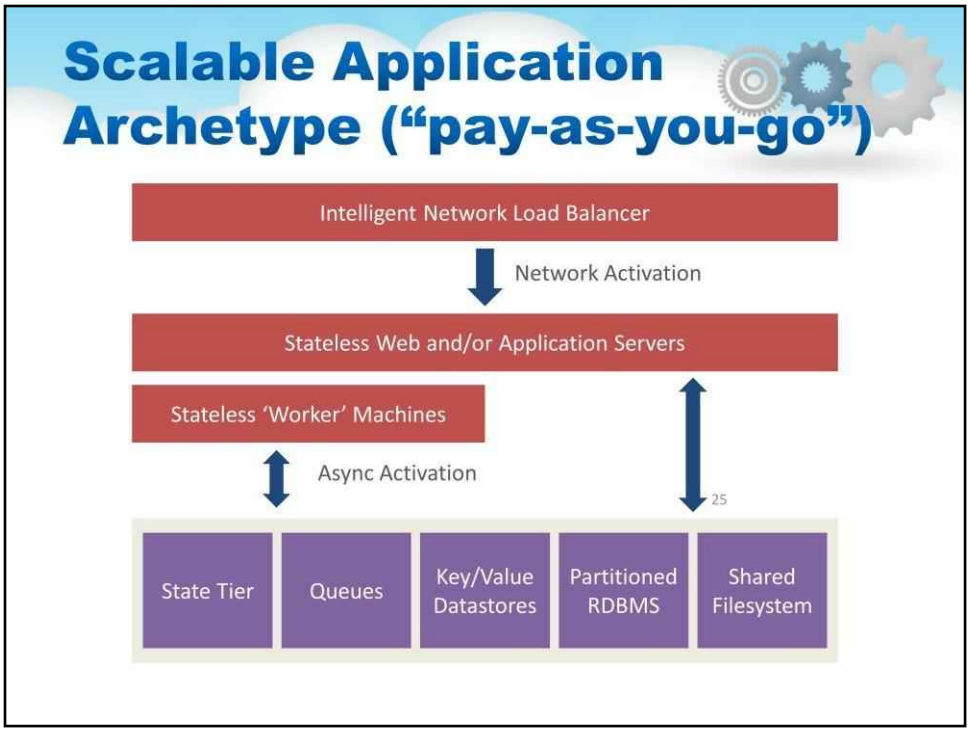


Цей слайд містить ілюстрацію масштабного перегляду архітектури служби Microsoft Azure:

Ось ключові моменти

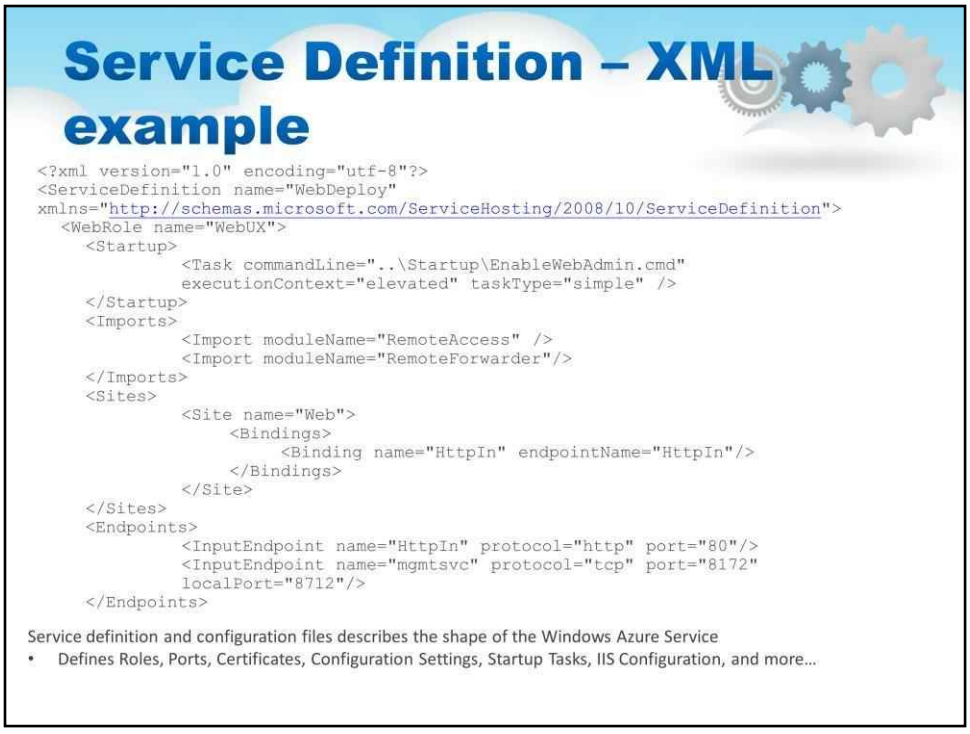
- усі зовнішні підключення проходять через балансир навантаження (LB)
- міжрольовий зв’язок (зверніть увагу, що балансувальника навантаження немає) і порти TCP безпосередньо до робочих ролей (або веб-ролей)
- сховище використовується для асинхронного та надійного зв’язку через черги для багатьох варіантів
- міжрольове спілкування заповнює, коли вам потрібна пряма синхронна комунікація.

Балансувальники навантаження є ключем до Windows Azure.



Цей слайд містить ілюстрацію програми масштабування

- Високомасштабні додатки часто слідують такому шаблону
- Вхідне підключення здійснюється через балансувальник навантаження
  - Запити маршрутизуються циклічним способом
  - Балансувальник навантаження зазвичай знає про стан веб-серверів
- Є один або кілька рівнів або груп веб-серверів або серверів програм без збереження стану
  - Під відсутністю стану ми маємо на увазі, що вони не зберігають стан між запитами клієнта
  - Без збереження стану означає, що працює просте балансування навантаження – немає потреби у закріплених сеансах
  - Без стану означає, що збій веб-сервера не спричиняє серйозних проблем для програми — його просто видаляють із балансувальника навантаження
- Рівень збереження стану або зберігання
  - Зазвичай це включатиме певний підхід масштабування для великих програм
  - Часто використовують розділені бази даних
  - Часто якийсь механізм черги
- Програми часто виконують обробку у фоновому режимі.
  - Покращує час відповіді для користувачів
  - Дозволяє буферизувати піки навантаження в чергах



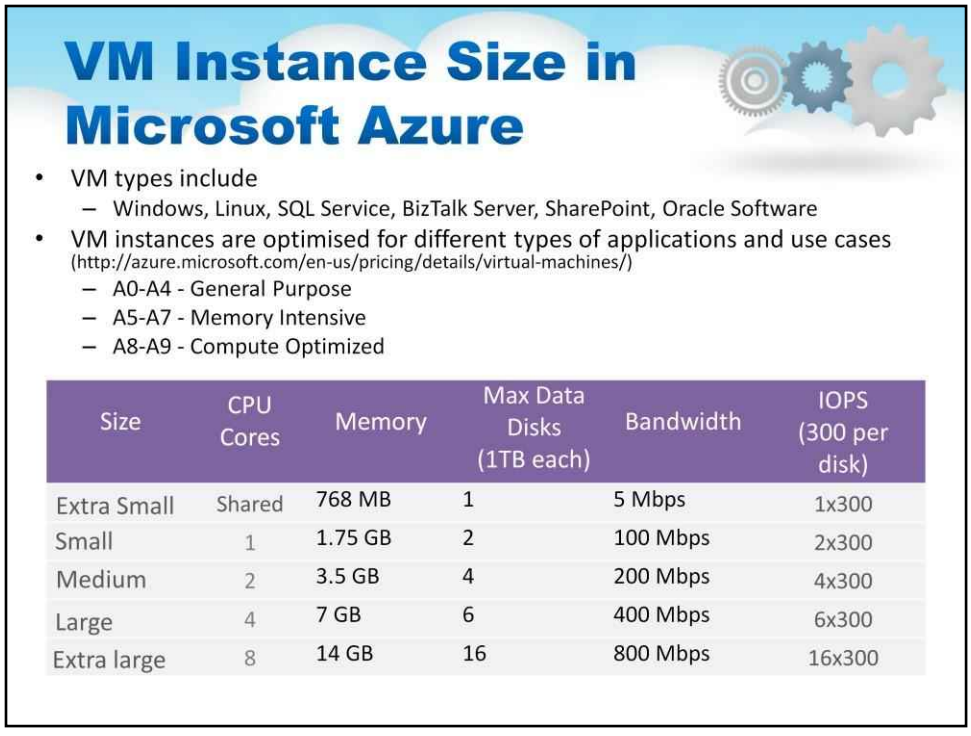
Компоненти та конфігурацію служби Microsoft Azure визначено у формі документа XML.

Визначення послуг

- описує форму служби Microsoft Azure
- визначає ролі, порти, сертифікати, параметри конфігурації, завдання запуску, конфігурацію IIS тощо…

На цьому слайді показано XML-приклад визначення служби.

У цьому прикладі файл конфігурації служби визначає ім’я облікового запису та інформацію, пов’язану з доступом: ім’я користувача, зашифрований пароль і відповідний відбиток сертифіката.



Цей слайд містить таблицю, спрямовану на розуміння того, як і навіщо змінювати розмір віртуальної машини для ролі Windows Azure.

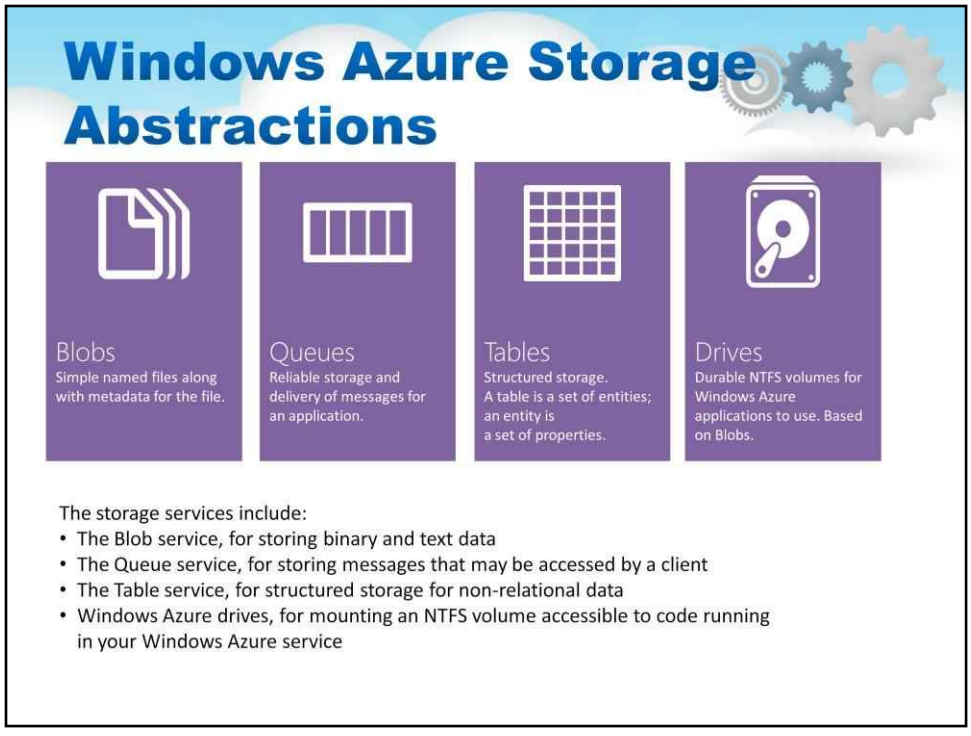
Коли ви створюєте свою модель обслуговування, ви можете вказати розмір віртуальної машини (VM), на якому розгортати екземпляри вашої ролі, залежно від вимог до ресурсів.

Визначає розмір ВМ

- кількість ядер ЦП
- ємність пам'яті
- розмір локальної файлової системи, призначений для запущеного екземпляра

Кожна фізична машина в Windows Azure містить 8 процесорних ядер. Тобі потрібно вкажіть примірник XL, щоб зарезервувати всю машину

- Мережа спільна, але розривна
- Може вибухнути за вашу 1/8 виділення при використанні невеликої віртуальної машини
- Може обмежуватися лише вашим розподілом
- Для гарантованої високої пропускної здатності мережі використовуйте XLVM



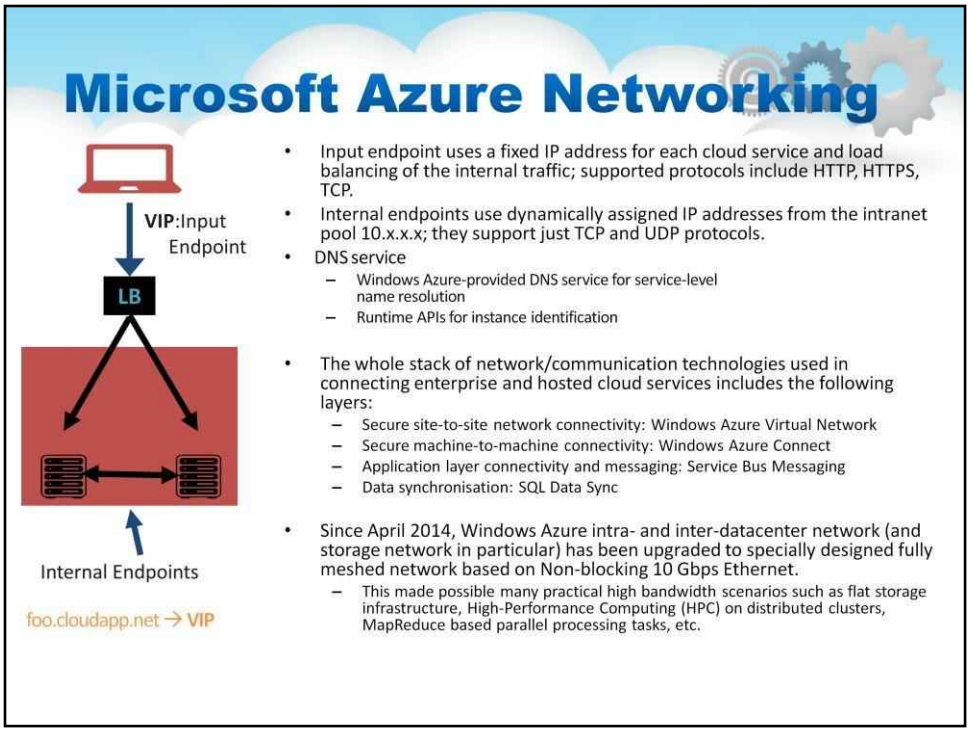
Ілюстрація на цьому слайді розглядає абстракції сховища Windows Azure і дозволяє зрозуміти кожен тип сховища на високому рівні.

Служби зберігання Windows Azure забезпечують зберігання двійкових і текстових даних, повідомлень і структурованих даних у Windows Azure.

Послуги зберігання включають:

- Сервіс Blob для зберігання двійкових і текстових даних
- Служба черги для зберігання повідомлень, до яких може отримати доступ клієнт
- Служба таблиць для структурованого зберігання нереляційних даних
- Диски Windows Azure для монтування тому NTFS, доступного для коду, запущеного у вашій службі Windows Azure

Для розробників доступ до служб Blob, Queue і Table доступний через керовану бібліотеку Windows Azure і REST API служб зберігання Windows Azure.



Тепер ми розглянемо мережу, включену в хмарну службу

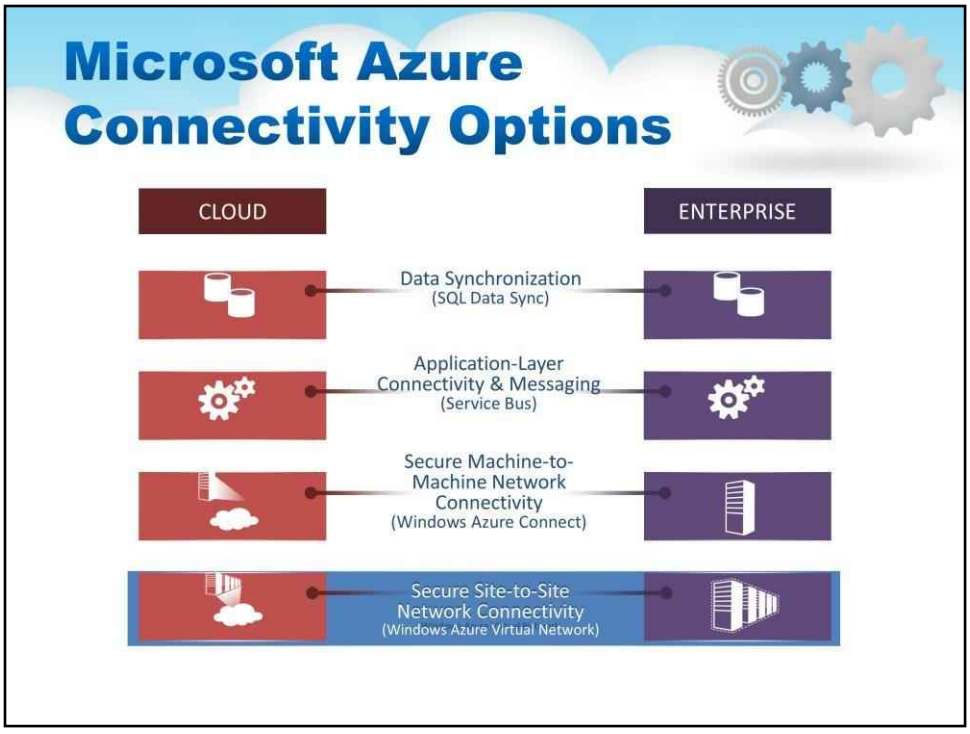
1. Хмарна служба отримує віртуальну IP-адресу, призначену для слота розгортання
   - Жоден порт не відкривається за замовчуванням
   - Необхідно визначити кінцеві точки, щоб відкрити порти
2. Вхідна кінцева точка – це порт
   - Це збалансоване навантаження
   - Зіставляється для всіх екземплярів ролі
   - Підтримується mapipng порту
3. Внутрішня кінцева точка забезпечує зв’язок між ролями та екземплярами
   - Порти для зв’язку між віртуальними машинами закриті за умовчанням
   - Потрібно визначити внутрішню кінцеву точку для зв’язку
   - Внутрішні кінцеві точки можуть бути діапазонами портів
4. Розділення DNS
   - Підтримується розпізнавання імен лише на рівні служби
   - Необхідно використовувати API середовища виконання для розпізнавання імен примірників.

Весь набір мережевих/комунікаційних технологій, що використовуються для підключення корпоративних і розміщених хмарних служб, включає такі рівні:

- Захищене підключення до мережі «сайт-сайт»: Windows Azure VirtualNetwork
- Безпечне міжмашинне підключення: Windows AzureConnect
- Підключення рівня додатків і обмін повідомленнями: Service BusMessaging
- Синхронізація даних: SQL Data Sync

З квітня 2014 року внутрішня та міжцентрова мережа Windows Azure (зокрема мережа зберігання даних) була оновлена до спеціально розробленої повністю комірчастої мережі на основі неблокуючої мережі Ethernet 10 Гбіт/с.

Це стало можливим для багатьох практичних сценаріїв із високою пропускною здатністю, таких як плоска інфраструктура зберігання, високопродуктивні обчислення (HPC) у розподілених кластерах, завдання паралельної обробки на основі MapReduce тощо.



На цьому слайді показано структуру стеку Microsoft для забезпечення зв’язку між локальною та хмарною мережею.

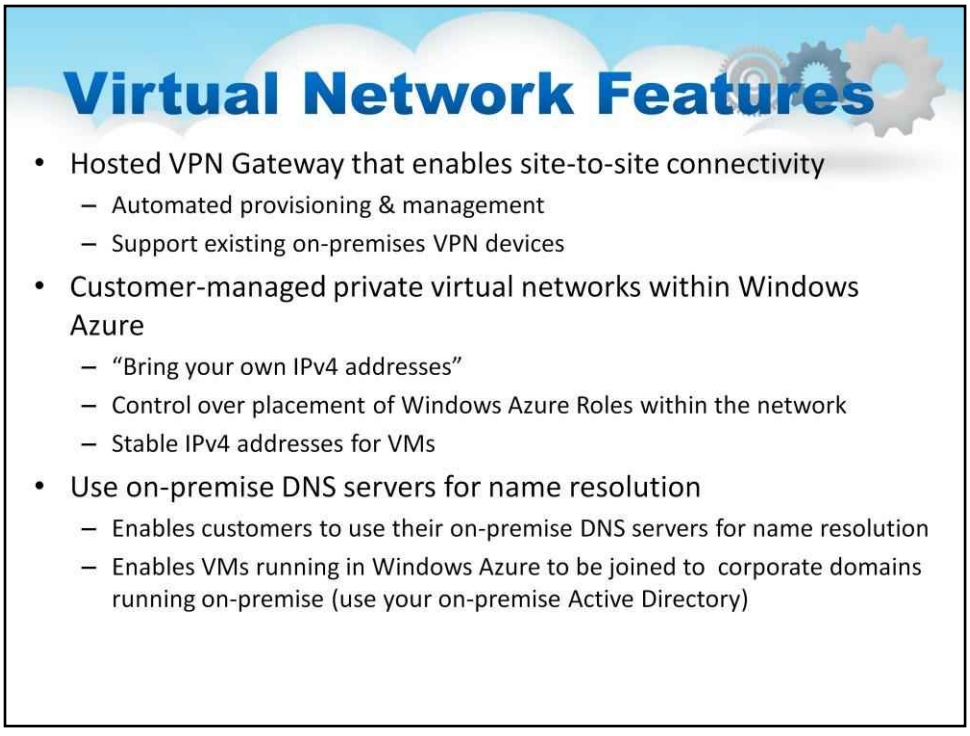
Існує кілька можливостей, які можна використовувати залежно від мети підключення

На найнижчому рівні підключення на основі VPN можна досягти за допомогою віртуальної мережі Azure. Це аналог AWS Virtual Private Cloud, воно засноване на тунелюванні IPSEC.

Azure Connect є більш конкретним, з’єднуючи сервер (VM) із сервером, як показано на малюнку

Існує багато варіантів підключення на рівні програми, але найпоширеніший реалізує надійний транспорт повідомлень (pub/sub) Service Bus (протокол MS MQ)

Нарешті, нова функція SQL Server під назвою SQL Sever Always On синхронізує екземпляри AQL Server у мережі.



На цьому слайді описуються функції віртуальної мережі.

Для VNET потрібен Hosted VPN Gateway, який забезпечує підключення між сайтами

- Автоматизоване надання та керування
- Підтримка наявних локальних пристроїв VPN

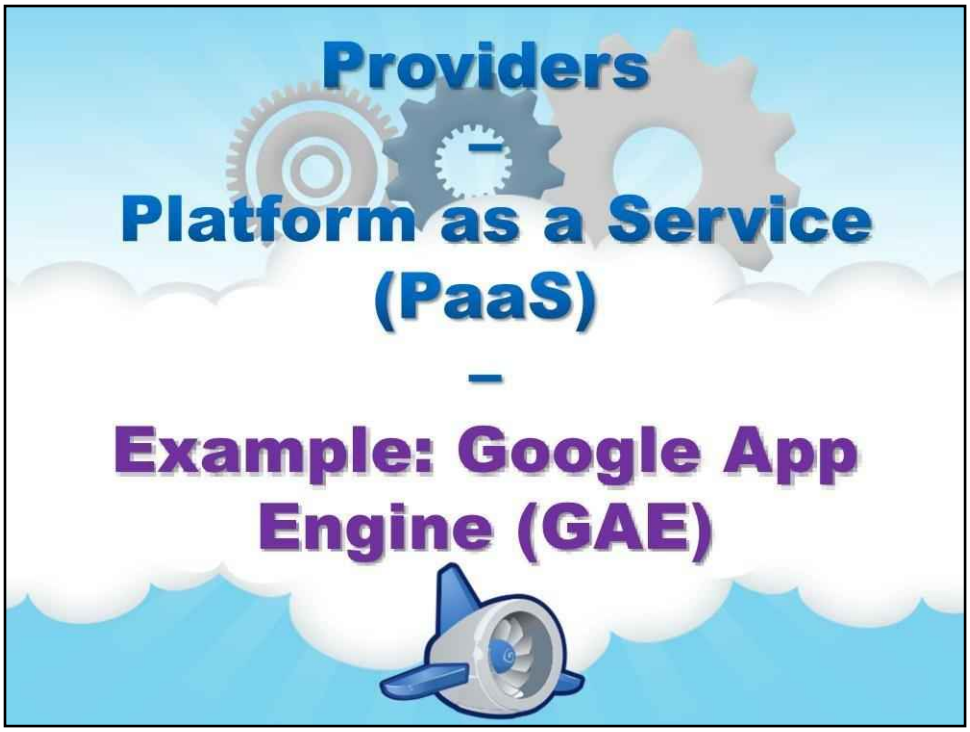
Він забезпечує приватні віртуальні мережі, якими керує клієнт, у Windows Azure

- «Принесіть власні адреси IPv4"
- Контроль за розміщенням ролей Windows Azure в мережі
- Стабільні адреси IPv4 для віртуальних машин
- Надає лише IP-адреси у віртуальній мережі
- Це також дозволяє виділяти IP-підмережі за допомогою віртуальної мережі
- Застереження: підмережі, що перекриваються, заборонені, і IP-адреса залишається у віртуальній машині протягом усього життя

Це дозволяє клієнтам використовувати локальні DNS-сервери для розпізнавання імен

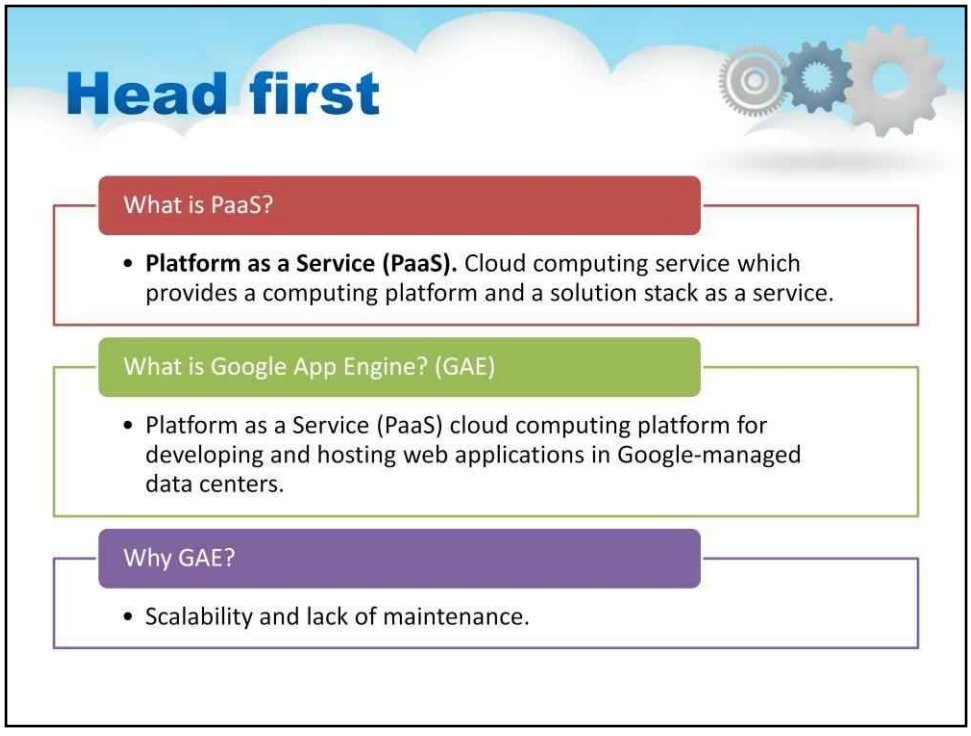
- Дозволяє клієнтам використовувати свої локальні DNS-сервери для розпізнавання імен
- Дозволяє приєднувати віртуальні машини, що працюють у Windows Azure, до корпоративних доменів, які працюють локально (використовуйте ваш локальний Active Directory)

Microsoft керує шлюзом на сайті клієнта, який є програмним забезпеченням і працює в активному/пасивному режимі для високої доступності.



Приклад: Google App Engine (GAE)

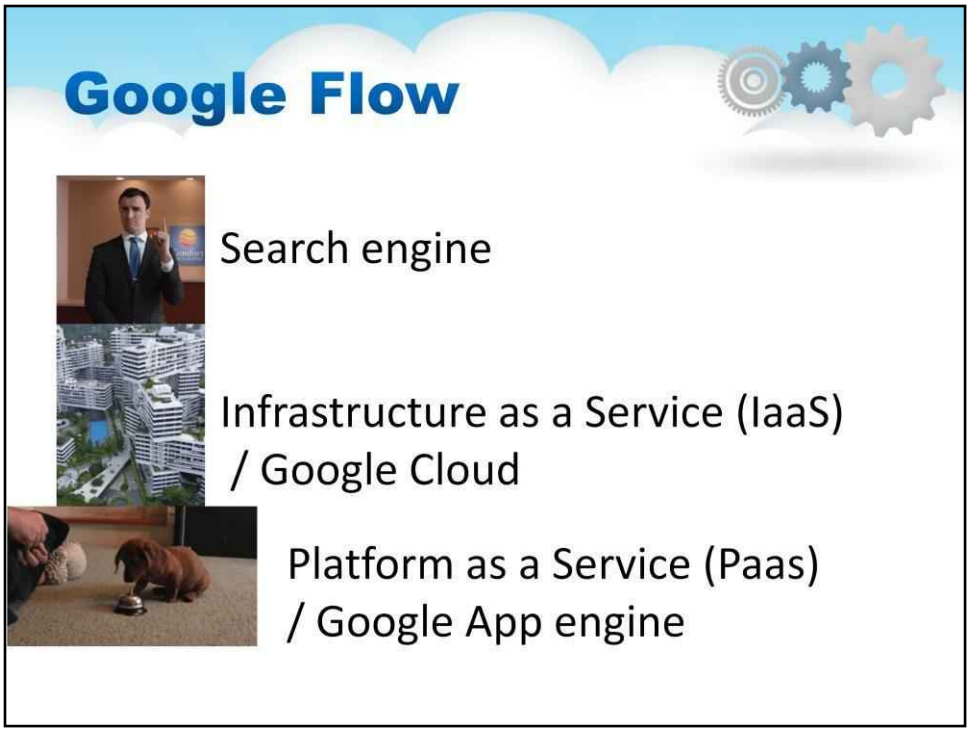
Другий приклад присвячено Google App Engine (GAE), який представляє приклад платформи як послуги (PaaS).



Звичайні хмарні сервери (наприклад, Amazon AWS) надають вам віртуальний сервер у хмарі. Ви отримуєте абстракцію фізичної машини, а все інше ви повинні зробити самі. Щоб створити веб-сайт тут, вам потрібно буде встановити ОС (наприклад, Linux), потім встановити веб-сервер (як-от Apache), потім додати інтерпретатор для бажаної мови на веб-сервер (наприклад, modpython для Python) і встановити базу даних (як MySQL). Звичайно, якщо ви хочете використовувати кілька серверів, або кілька баз даних, або кілька веб-серверів для масштабованості, ви повністю самі. І ви платите за сервер на місяць.

На один рівень вище цього — такі гравці, як Heroku або Webfaction. У них встановлена ОС, веб-сервер, інтерпретатор і база даних. Вам просто потрібно написати свою програму та з’єднати всі вищезазначені частини разом. Як і раніше, якщо ви хочете використовувати кілька серверів, або кілька баз даних, або кілька веб-серверів для масштабованості, ви повністю самі. Зазвичай у таких спільних хостах ви платите за «кількість екземплярів», які ви запускаєте, будь-якої програми та розмір кожної (з точки зору споживаної пам’яті чи пропускної здатності).

Google AppEngine на один рівень вищий за цей. Тут ви не маєте доступу до веб-сервера чи бази даних. Ви просто отримуєте платформу, на якій починаєте писати код на Python чи Java (або ціла низка інших мов тепер підтримується), і є API для зберігання даних, який ви використовуєте безпосередньо, не турбуючись про те, у якій базі даних вони зберігаються. І тут , ви платите окремо за кожен спожитий ресурс (як-от пам’ять, пропускна здатність, ГБ пам’яті, МБ даних у базі даних, кількість надісланих електронних листів тощо).



Як ми всі знаємо, Google знаходиться в авангарді того, що можна назвати Інтернет-революцією.

Окрім того, що компанія є пошуковою системою номер 1, компанія також використовує Cloud і розробила різні продукти, щоб допомогти розробникам запускати нові масштабовані веб- та мобільні програми.

Серед різноманітних продуктів на основі хмарних технологій Google app engine став досить популярним.

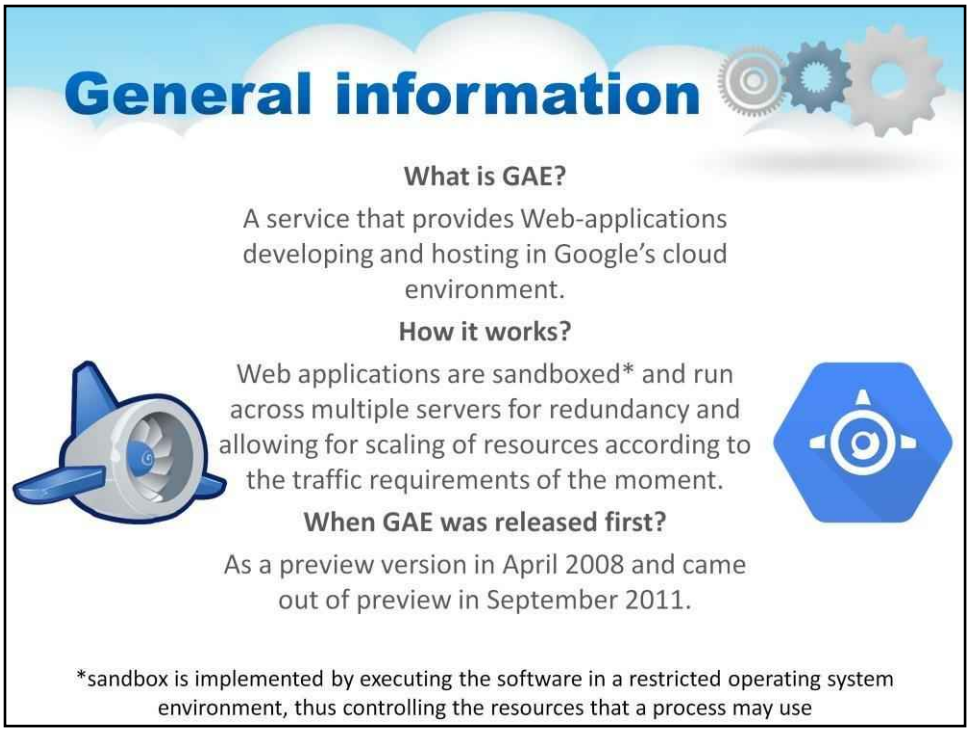
Двигун додатків — це платформа на основі хмари, вона є досить комплексною та поєднує в собі інфраструктуру як послугу (IaaS), платформу як послугу (PaaS) і програмне забезпечення як послугу (SaaS). Механізм додатків підтримує доставку, тестування та розробку програмного забезпечення на вимогу в середовищі хмарних обчислень, яке підтримує мільйони користувачів і має високу масштабованість.

Компанія розширює свою платформу та інфраструктуру до хмари за допомогою механізму додатків. Він представляє платформу для тих, хто хоче розробити рішення SaaS за конкурентоспроможними цінами.

Це платформа хмарних обчислень типу «платформа як послуга» (PaaS), яка повністю керується та використовує вбудовані служби для запуску ваших програм. Ви можете почати розробку майже миттєво після цього

завантаження комплекту розробки програмного забезпечення (SDK). Ви можете перейти до посібника розробника

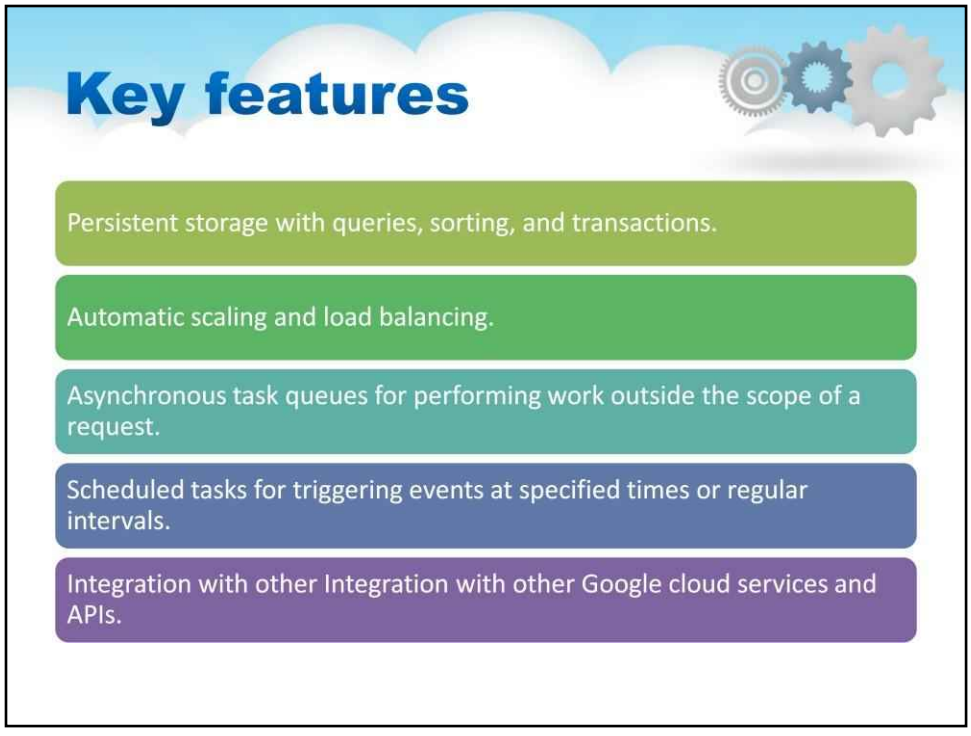
відразу, коли ви клацаєте мову, на якій хочете розробити свою програму.



Google App Engine (часто відомий як GAE) — це веб-фреймворк і платформа хмарних обчислень для розробки та розміщення веб-додатків у центрах обробки даних, якими керує Google.

Програми знаходяться в ізольованому програмному середовищі та працюють на кількох серверах.

GAE був вперше випущений як версія попереднього перегляду в квітні 2008 року та вийшов з попереднього перегляду у вересні 2011 року.

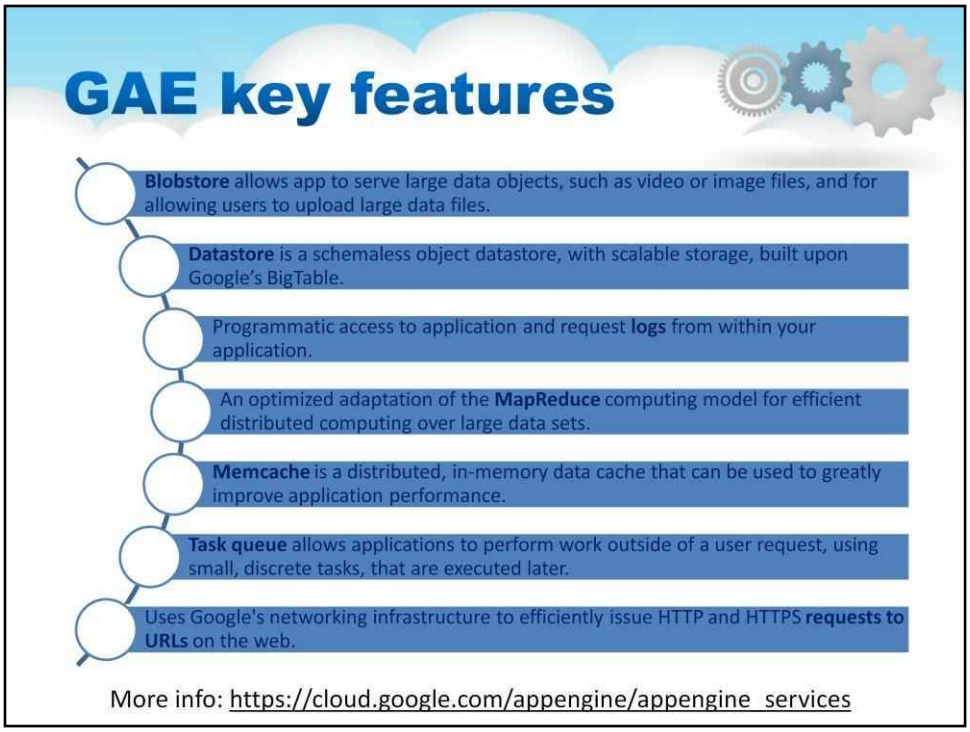


Стандартне середовище App Engine базується на примірниках контейнерів, що працюють в інфраструктурі Google. Контейнери попередньо налаштовані на одну з кількох доступних середовищ виконання (Java 7, Java 8, Python 2.7, Go та PHP). Кожне середовище виконання також містить бібліотеки, які підтримують стандартні API App Engine. Для багатьох програм стандартне середовище виконання та бібліотеки можуть бути всім, що вам потрібно.

Стандартне середовище App Engine дозволяє легко створювати та розгортати програму, яка надійно працює навіть за великого навантаження та великих обсягів даних. Він містить такі функції, які показано на цьому малюнку:

- Постійне зберігання із запитами, сортуванням і транзакціями.
- Автоматичне масштабування та балансування навантаження.
- Асинхронні черги завдань для виконання роботи поза межами arequest.
- Заплановані завдання для запуску подій у визначений час або через регулярні проміжки часу.
- Інтеграція з іншими хмарними службами та API Google.

Програми працюють у безпечному середовищі ізольованого програмного середовища, що дозволяє стандартному середовищу App Engine розподіляти запити між декількома серверами та масштабувати сервери відповідно до потреб трафіку. Ваша програма працює у власному безпечному та надійному середовищі, яке не залежить від апаратного забезпечення, операційної системи чи фізичного розташування сервера.



На цьому слайді ви можете побачити список основних функцій GAE.

Зберігання даних.Сховище Google App Engine складається з DataStore і BlobStore. BlobStore обслуговує об'єкти даних, більші за допустимі для Datastore.

Журнали.Стандартне середовище App Engine підтримує два типи журналів: журнали програми містять довільні повідомлення з міткою часу та рівнем журналу, журнали запитів містять записи для кожного запиту, обробленого вашою програмою.

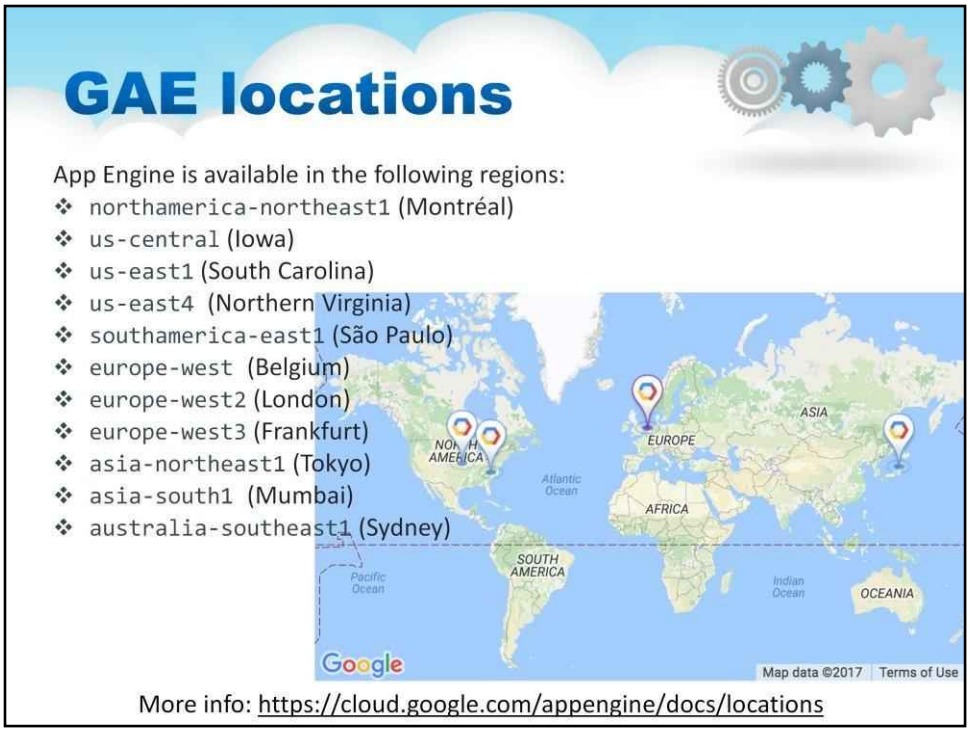
MapReduce. MapReduce — це бібліотека з відкритим вихідним кодом, створена на основі служб App Engine. Він надає модель програмування для великомасштабної розподіленої обробки даних, автоматичного розпаралелювання та розподілу в межах існуючої кодової бази та інших корисних функцій. MapReduce доступний на GitHub для Java і Python. Це описано в окремому модулі та лекціях.

Memcache.Стандартне середовище App Engine підтримує два класи служби memcache: спільний memcache забезпечує ємність кешу на основі найкращих зусиль і залежить від загального попиту всіх програм, які обслуговує App Engine; виділений кеш пам'яті забезпечує фіксовану ємність кешу, призначену виключно клієнтській програмі.

Черга завдань.Якщо якійсь програмі потрібно виконати фонову роботу, вона може використовувати Task Queue API, щоб організувати цю роботу в невеликі окремі блоки, які називаються завданнями. Програма додає завдання до черги завдань, щоб виконати їх пізніше.

спілкування.Програми стандартного середовища App Engine можуть обмінюватися даними з іншими програмами або отримувати доступ до інших ресурсів в Інтернеті за допомогою отримання URL-адрес.

Більше інформації можна отримати за посиланням, наведеним тут.



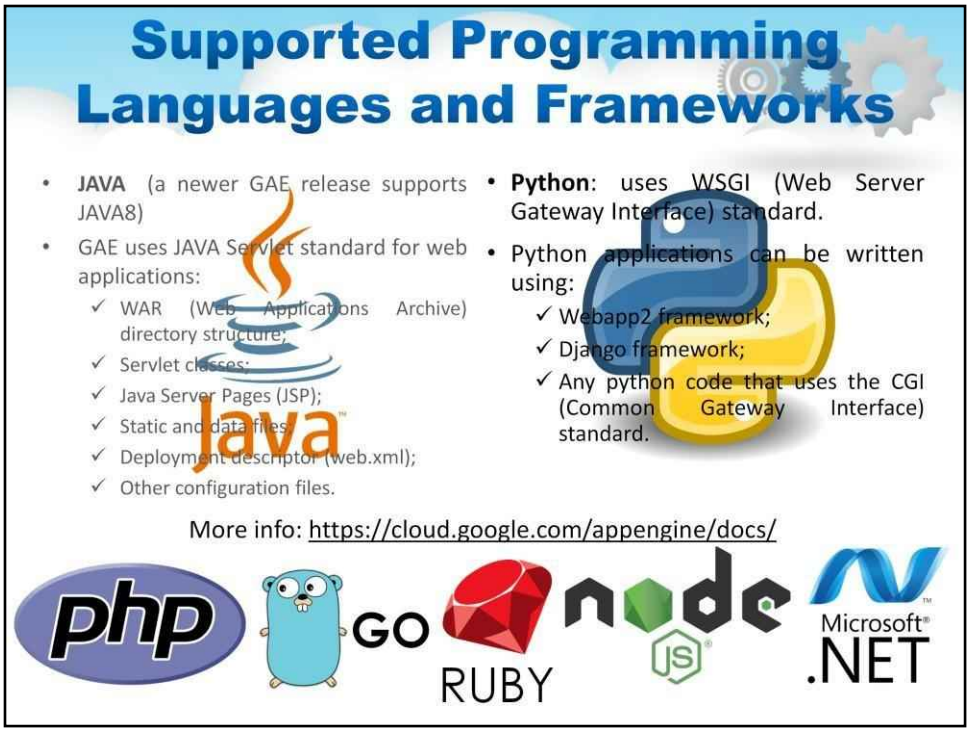
App Engine є регіональним, що означає, що інфраструктура, яка запускає ваші програми, розташована в певному регіоні та керується Google, щоб бути надлишковим доступним у всіх зонах цього регіону.

Клієнт може вибрати, де розташувати програми, щоб відповідати вимогам до затримки, доступності та довговічності.

Регіони — це самостійні географічні області, які складаються із зон.

Служби Google Cloud Platform доступні в країнах Північної Америки, Південної Америки, Європи, Азії та Австралії.

Більше інформації можна отримати за посиланням, наведеним тут.



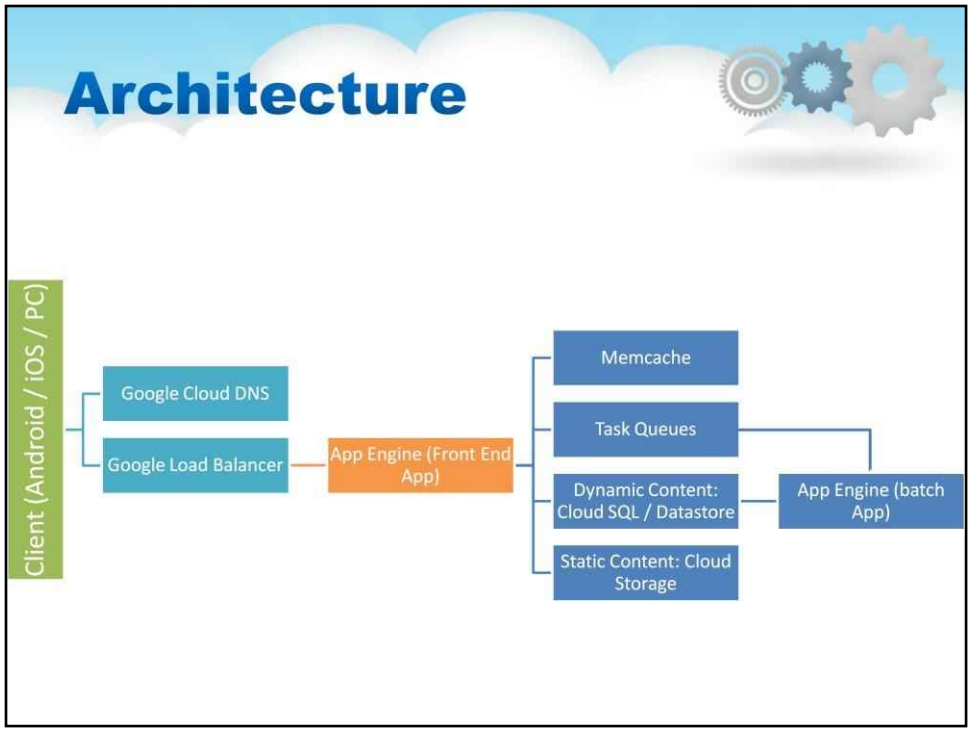
Для GAE підтримувані мови програмування включають Java (і, відповідно, інші мови JVM, такі як Kotlin, Groovy, JRuby, Scala, Clojure), Python, PHP, Go та Ruby. Node.js і .NET Framework також доступні в гнучкому середовищі.

Google App Engine підтримує багато стандартів і фреймворків Java. Основою цього є технологія сервлетів 2.5, яка використовує веб-сервер Jetty з відкритим кодом разом із супутніми технологіями, такими як JSP. Новіший випуск App Engine Standard Java у бета-версії підтримує Java8, Servlet 3.1 і Jetty9.

Веб-фреймворки Python, які працюють на Google App Engine включають Django, CherryPy, Pyramid, Flask, web2py і webapp2. Для створення програми можна використовувати будь-яку структуру Python, яка підтримує WSGI за допомогою адаптера CGI; рамку можна завантажити разом із розробленою програмою. Бібліотеки сторонніх розробників, написані на чистому Python, також можна завантажувати.

Розробники можуть використовувати Go, Java, PHP або Python для написання додатків механізму додатків. Вони можуть розробляти та тестувати програму локально за допомогою SDK, що містить інструменти для розгортання програм. Кожна мова має власний SDK і середовище виконання.

Більше інформації можна отримати за посиланням, наведеним тут.



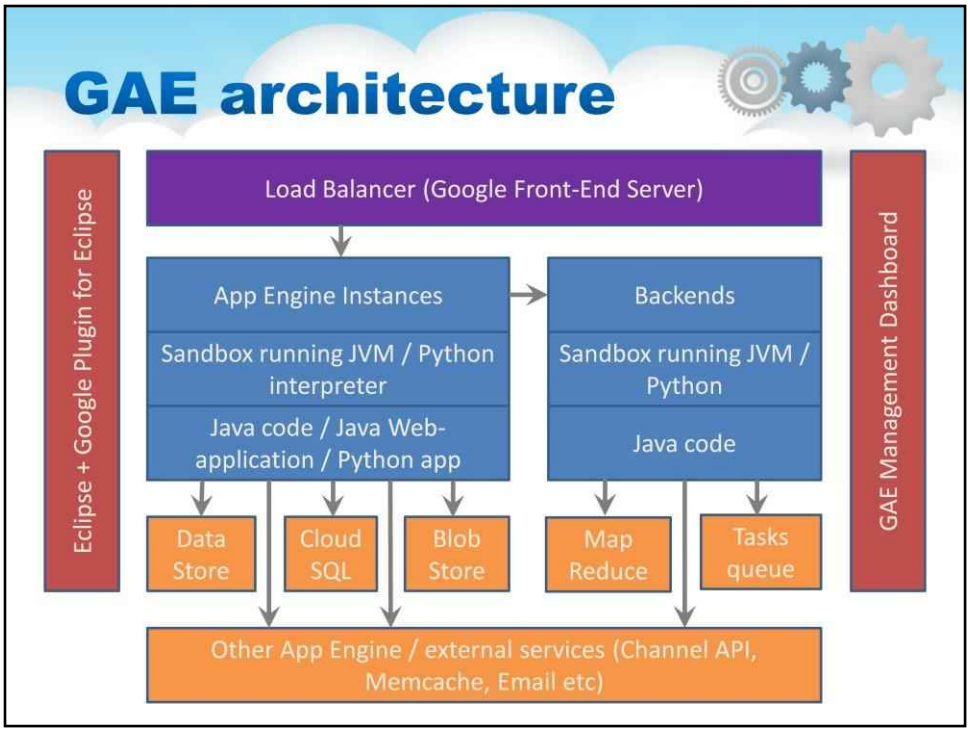
App Engine — це платформа Google PaaS і надійне середовище розробки. SDK для App Engine підтримує розробку та розгортання програми в хмарі.

App Engine підтримує кілька версій програми, що дозволяє легко розгортати нові функції програми та розподіляти трафік для підтримки A/B-тестування.

Сервіси Memcache і Task Queue інтегровані в стандартне середовище App Engine. Memcache — це кеш у пам’яті, спільний для екземплярів App Engine. Це забезпечує надзвичайно високу швидкість доступу до інформації, яка кешується веб-сервером (наприклад, автентифікація або інформація облікового запису).

Черги завдань забезпечують механізм для перевантаження довготривалих завдань на внутрішні сервери, звільняючи передні сервери для обслуговування нових запитів користувачів.

Нарешті, App Engine має вбудований балансувальник навантаження (надається Google Load Balancer), який забезпечує прозоре балансування навантаження рівнів 3 і 7 для програм.



На цьому слайді описано, як працює GAE.

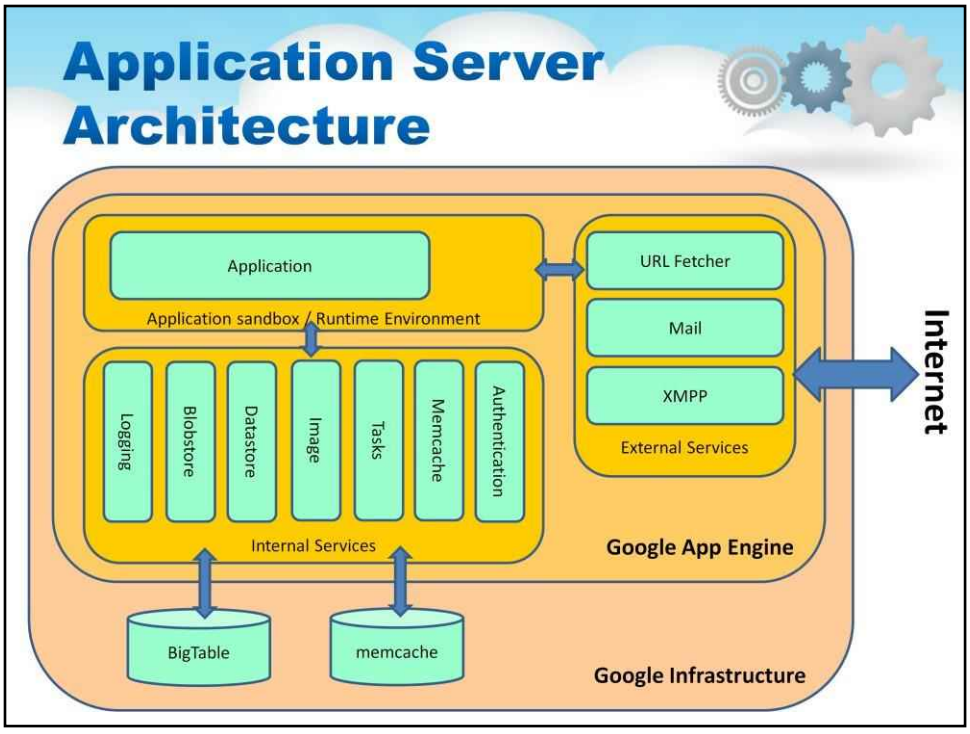
Front-End Server, який балансує програму між екземплярами App Engine і їхніми серверними частинами.

Запуск програм, які отримують доступ до сховищ даних, як Datastore, Blobstore і Cloud SQL.

Запуск коду, що працює з бібліотекою MapReduce і чергами завдань.

І програми, і код можуть використовувати інші служби, такі як кеш пам’яті, електронна пошта тощо.

Розробники контролюють процеси через плагін IDE та інформаційну панель керування GAE.



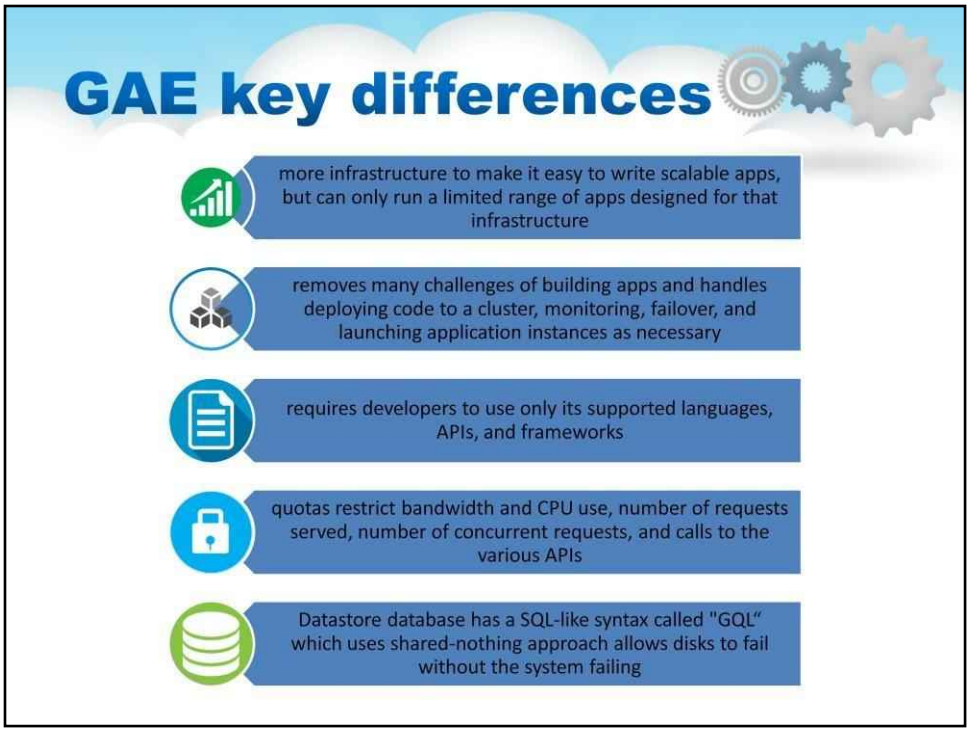
На цьому слайді описано, як працює сервер додатків.

Програма працює на сервері в пісочниці.

Google App Engine, що з’єднує запущену програму з внутрішніми та зовнішніми службами.

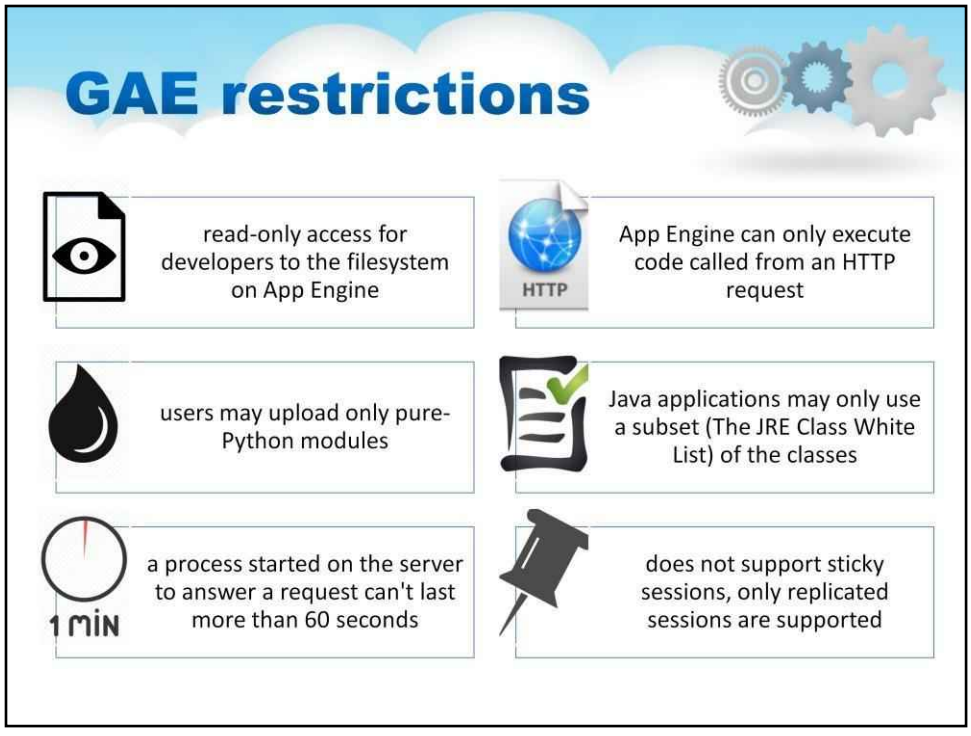
Внутрішні служби використовують інфраструктуру Google із технологіями Memcache і BigTable.

Зовнішні сервіси забезпечують підключення до мережі Інтернет.



Основні відмінності Google App Engine від інших постачальників:

- Порівняно з іншими масштабованими службами хостингу, такими як Amazon EC2, GAE надає більше інфраструктури, щоб спростити написання масштабованих програм, але може запускати лише обмежений діапазон програм, розроблених для цієї інфраструктури.
- Інфраструктура App Engine усуває багато проблем із системним адмініструванням і розробкою створення додатків для масштабування до сотень запитів на секунду та більше. Google займається розгортанням коду в кластері, моніторингом, відновленням після збоїв і запускає екземпляри програми, якщо це необхідно.
- У той час як інші служби дозволяють користувачам встановлювати та налаштовувати майже будь-яке програмне забезпечення, сумісне з *NIX, App Engine вимагає від розробників використання лише підтримуваних мов, API та фреймворків. Google Cloud SQL можна використовувати для додатків App Engine, яким потрібна реляційна база даних, сумісна з MySQL.
- Щоденні та хвилинні квоти обмежують пропускну здатність і використання ЦП, кількість обслуговуваних запитів, кількість одночасних запитів і викликів різних API, а окремі запити припиняються, якщо вони займають більше 60 секунд або повертають більше 32 МБ даних .
- Інтегрована база даних Google Cloud Datastore Google App Engine має SQL-подібний синтаксис під назвою "GQL". GQL не підтримує оператор Join. Натомість зв’язки «один до кожного» та «багато до багатьох» можна досягти за допомогою ReferenceProperty(). Цей підхід без спільного використання дозволяє дискам виходити з ладу без збою системи.



Розробникам слід враховувати наступні обмеження:

- Розробники мають доступ лише для читання до файлової системи в App Engine. Програми можуть використовувати лише віртуальні файлові системи, наприклад GAE-filestore.
- App Engine може виконувати лише код, викликаний із запиту HTTP (заплановані фонові завдання дозволяють самостійно викликати запити HTTP).
- Користувачі можуть завантажувати довільні модулі Python, але лише якщо вони чисті на Python; Модулі C не підтримуються.
- Програми Java можуть використовувати лише підмножину (зазначену в білому списку класів JRE) класів зі стандартної версії JRE. Це обмеження не існує для середовища виконання App Engine Standard Java8.
- Процес, запущений на сервері для відповіді на запит, не може тривати більше 60 секунд (у версії 1.4.0 це обмеження більше не поширюється на фонові завдання).
- Не підтримує закріплені сеанси (так звану спорідненість сеансів), підтримуються лише репліковані сеанси, включаючи обмеження кількості даних, що серіалізуються, і часу для серіалізації сеансу.

Google App Engine є безкоштовним до певного рівня споживаних ресурсів.

На цьому слайді ви можете побачити таблицю з ресурсами та відповідними безкоштовними обмеженнями за умовчанням.

Плата стягується за додаткове сховище, пропускну здатність або години екземплярів, необхідні програмі тощо.

Більше інформації можна отримати за посиланням, наведеним тут.

Цей слайд описує численні**переваги**двигуна програми:

Інфраструктура безпеки

Інтернет-інфраструктура Google є, мабуть, найбезпечнішою у світі. Ви можете бути впевнені, що ваша програма буде доступною для користувачів у всьому світі в будь-який час, оскільки Google має кілька сотень серверів у всьому світі. Політика безпеки та конфіденційності Google поширюється на програми, розроблені з використанням інфраструктури Google.

Масштабованість

Для успіху будь-якої програми це є одним із вирішальних факторів. Google створює власні програми за допомогою GFS, Big Table та інших подібних технологій. Вам потрібно лише написати код програми, а Google подбає про тестування завдяки функції автоматичного масштабування, яку має механізм програми. Незалежно від обсягу даних або кількості користувачів, які зберігає ваша програма, механізм програми може задовольнити ваші потреби шляхом масштабування вгору або вниз за потреби.

Продуктивність і надійність

За останні 15 років компанія створила нові еталонні показники на основі продуктивності своїх послуг і продуктів. Механізм програми забезпечує таку ж надійність і продуктивність, як і будь-який інший продукт Google.

Економія коштів

Вам не потрібно наймати інженерів для керування серверами або робити це самостійно. Ви можете інвестувати заощаджені гроші в інші частини свого бізнесу.

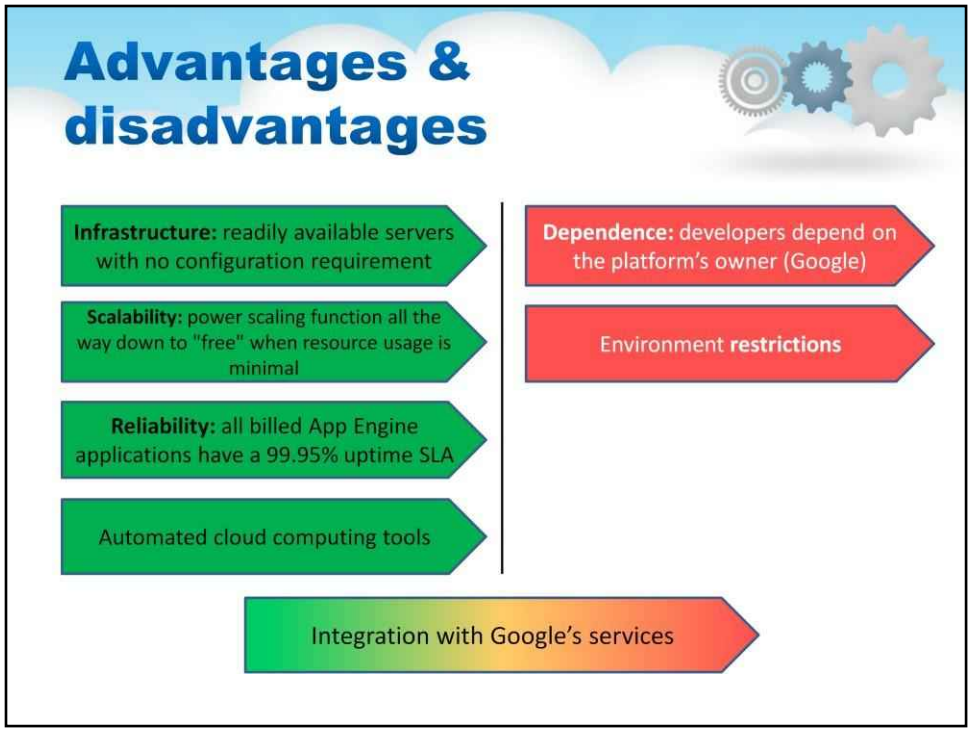
Цей слайд описує деякі**недоліки**двигуна програми:

Ви обмежені мовами: Java, PHP, Go та Python

Ви можете використовувати лише служби AppEngine (черги, пошук, кеш пам’яті, журналювання тощо). Якщо вам потрібно щось понад це - це буде важко.

Ви обмежені часом виконання AppEngine (наприклад: у Python ви не можете мати будь-які сторонні залежності, які знаходяться в C... обробка зображень тощо - не може бути й мови)

Суть полягає в тому, що ви обмежені – у вас є набір послуг, які надає AppEngine, і ви заблоковані в цих службах. Все, що виходить за межі цього, вимагатиме певного злому, щоб обійти його обмеження.



Нарешті, давайте порівняємо переваги та недоліки Google App Engine Usage.

The**переваги**включають готову інфраструктуру, масштабованість, надійність і широкий спектр інструментів.

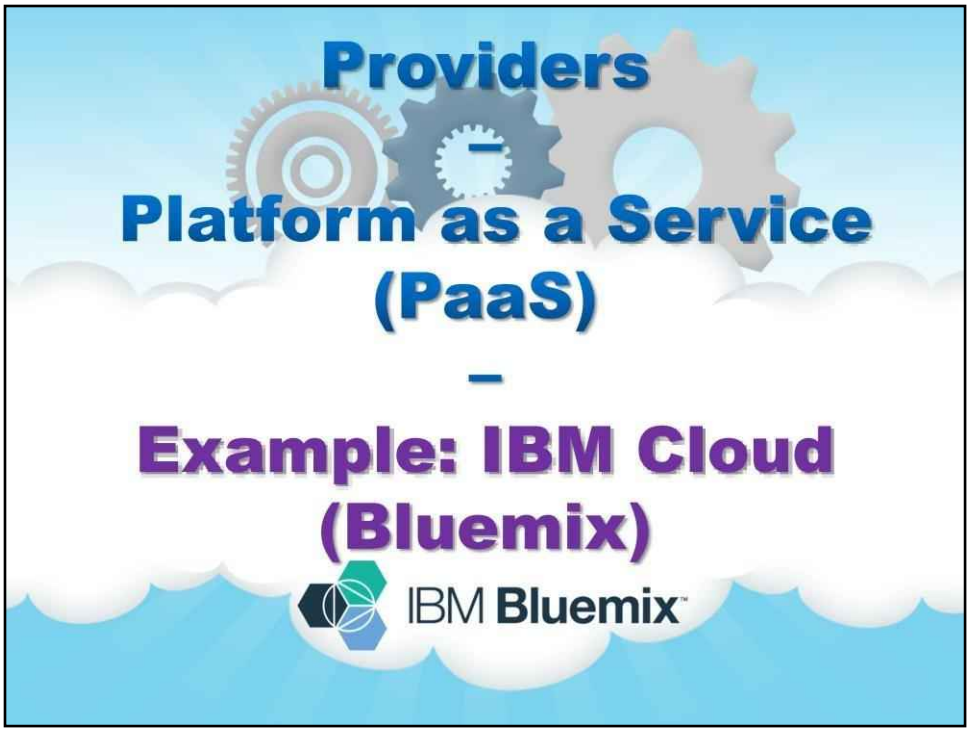
The**недоліки**включають залежність від платформи Google і обмеження, описані раніше.

Інтеграція з сервісом Google може бути і перевагою, і недоліком одночасно.

Google App Engine дозволяє створювати веб-програми для вашого бізнесу, використовуючи інфраструктуру Google.

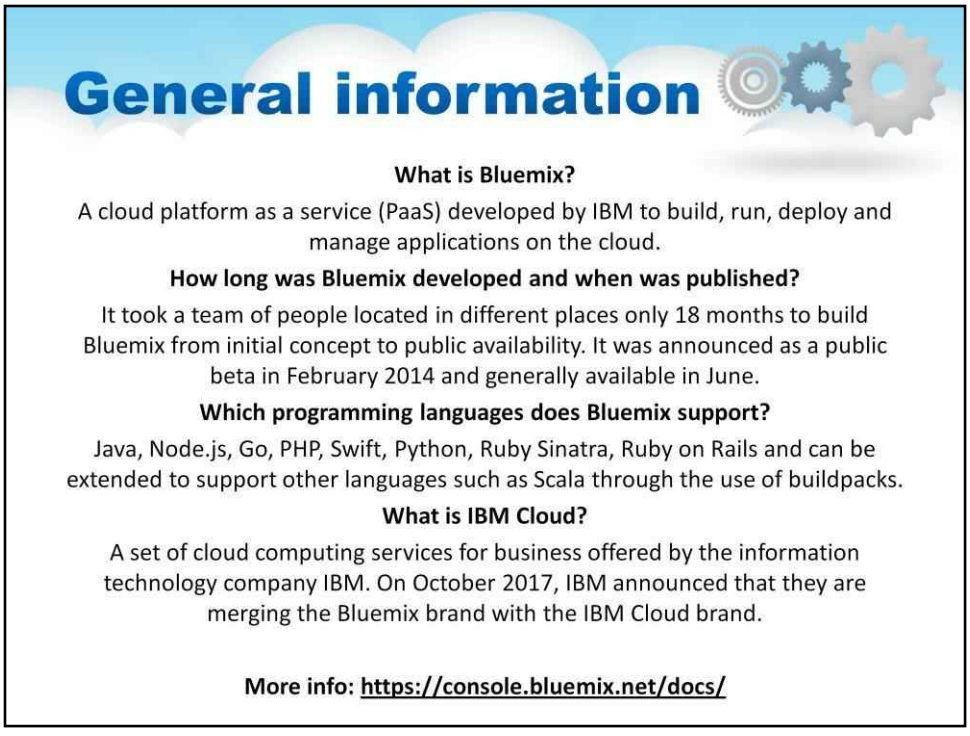
Програми App Engine легко розробляти, підтримувати та масштабувати відповідно до зростання потреб у трафіку та зберіганні даних.

З App Engine ви не платите за великі серверні простори, а потім витрачаєте ресурси на їх підтримку. Ви просто завантажуєте свою програму, і вона готова до обслуговування ваших користувачів. Про все інше подбає Google Cloud.



Приклад: IBM Cloud (Bluemix)

Наступний приклад присвячений IBM Cloud (попередня назва IBM Bluemix), який представляє приклад платформи як послуги (PaaS).



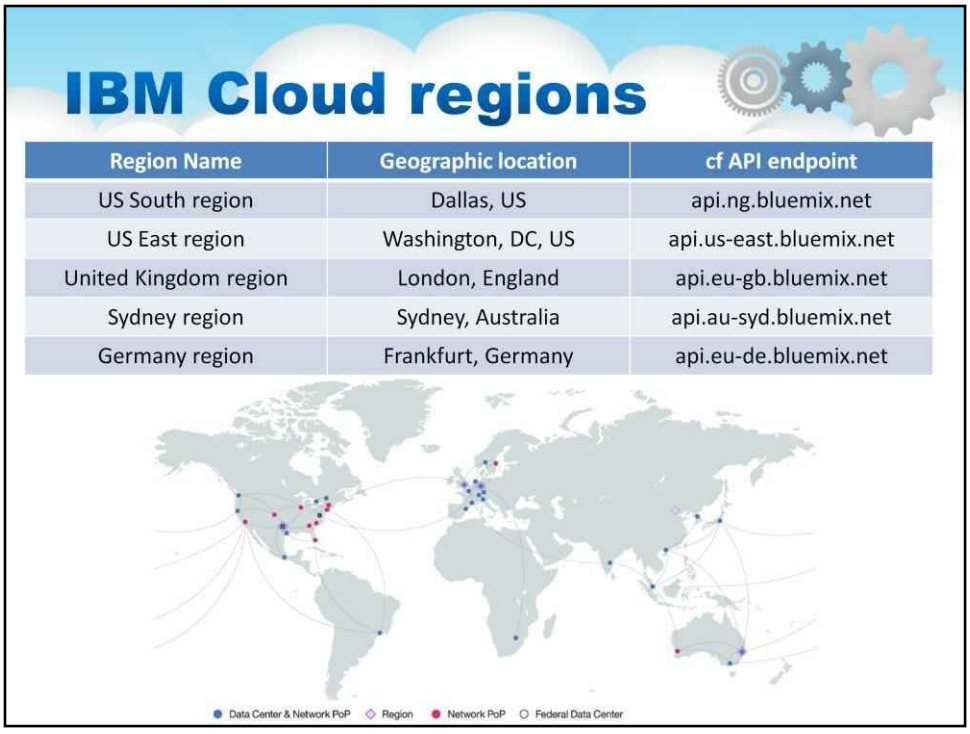
IBM Bluemix — це хмарна платформа як послуга (PaaS), розроблена IBM для створення, запуску, розгортання та керування програмами в хмарі.

Команді людей, розташованих у різних місцях, знадобилося лише 18 місяців, щоб створити Bluemix. Його було оголошено як публічну бета-версію в лютому 2014 року та загальнодоступним у червні.

Bluemix підтримує Java, Node.js, Go, PHP, Swift, Python, Ruby Sinatra, Ruby on Rails і може бути розширений для підтримки інших мов, таких як Scala, за допомогою пакетів збірки.

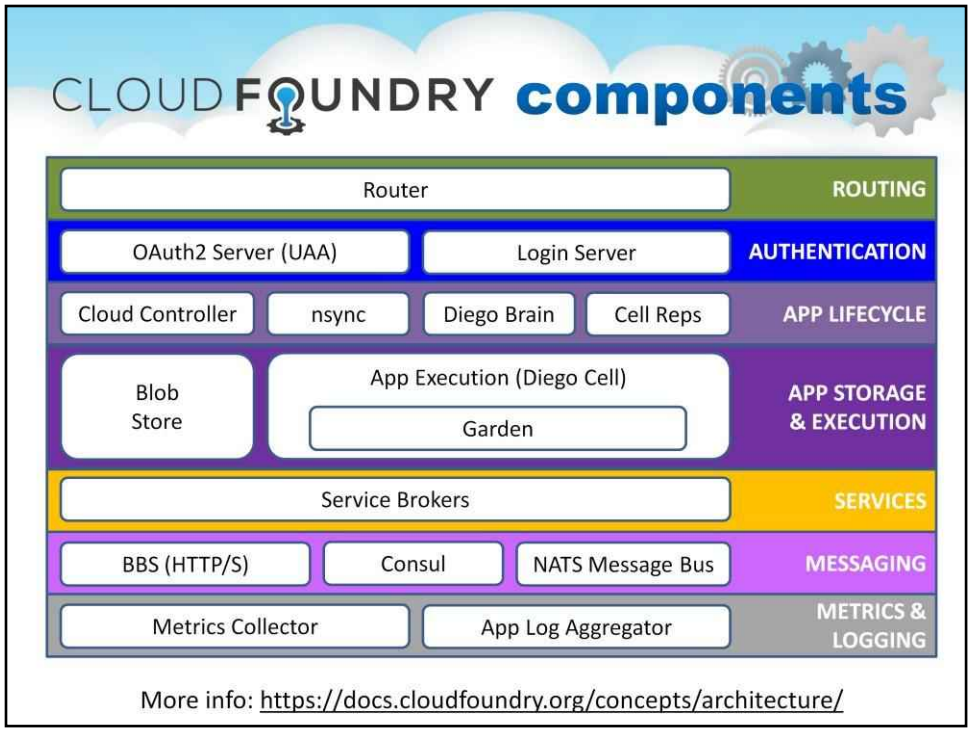
У жовтні 2017 року IBM оголосила про об’єднання бренду Bluemix із брендом IBM Cloud.
Ось чому ми дослідимо документацію IBM Cloud для Bluemix.

Більше інформації можна отримати за посиланням, наведеним тут.



IBM Cloud пропонує майже 60 центрів обробки даних у 6 регіонах і 18 зонах доступності по всьому світу. по всьому світу, щоб допомогти вам задовольнити потреби вашого глобального бізнесу.

Можна розгортати програми в різних регіонах з огляду на затримку або безпеку, з можливістю розгортання в одному регіоні або в кількох регіонах.



IBM Bluemix — це пропозиція PaaS на основі проекту з відкритим кодом Cloud Foundry. Платформа Cloud Foundry складається з наступних компонентів.

> **Примітка**: тут Diego є системою керування контейнерами, а Diego Cell безпосередньо керує та обслуговує завдання та запити. Diego Brain розподіляє завдання та запити в Diego Cells.

Маршрутизаторнаправляє вхідний трафік до відповідного компонента, компонента Cloud Controller або розміщеної програми, що працює на Diego Cell. **Сервер OAuth2**(Обліковий запис користувача та сервер автентифікації**-**UAA) і сервер входу

працювати разом, щоб забезпечити керування ідентифікацією.

Хмарний контролерпотім керує Diego Brain через компоненти CC-Bridge для координації окремих комірок Diego для створення та запуску програм.

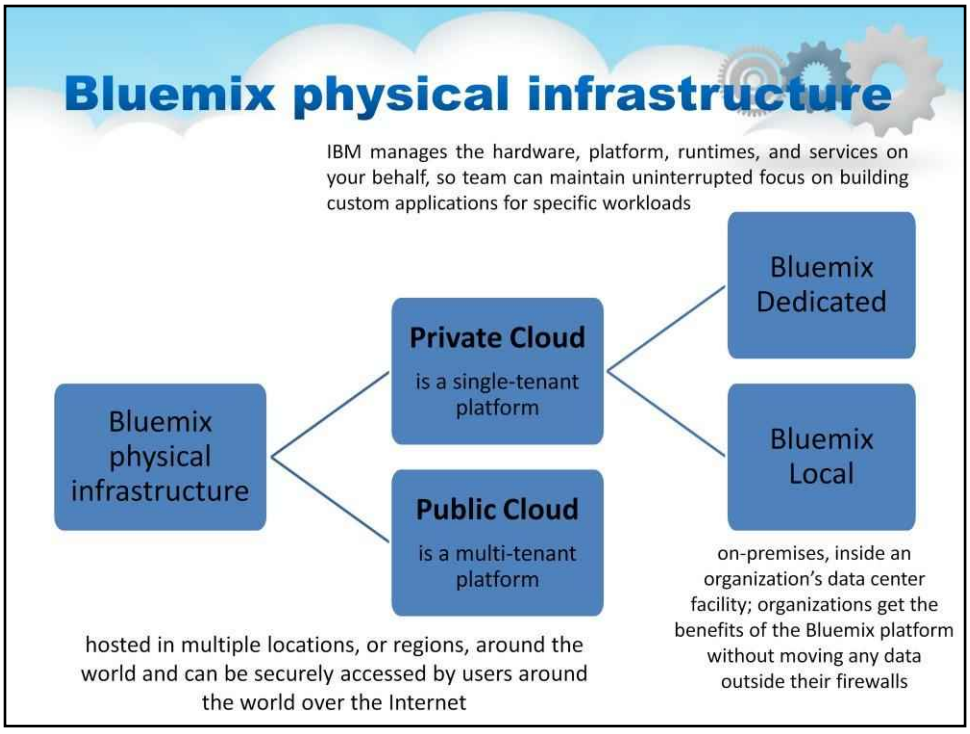
Blobstoreє репозиторієм для великих бінарних файлів.

Садові контейнери.Екземпляри додатків, завдання додатків і проміжні завдання виконуються як контейнери Garden на віртуальних машинах Diego Cell.

Сервісний брокервідповідає за надання конкретного екземпляра послуги. **Система дощок оголошень Дієго (BBS)**зберігає часто оновлювані та одноразові дані. **Шина повідомлень NATS** використовує протокол NATS для трансляції останніх таблиць маршрутизації на маршрутизатори.

Консульський серверзберігає довготривалі контрольні дані.

The**збирач метрик**збирає показники та статистику з компонентів. The **Агрегатор журналів програм**передає журнали програми розробникам.



Все більш прагматичним і практичним способом для підприємств є а**гібридна хмарна стратегія**.

Це дозволяє їм використовувати дані та IT-ресурси, які вже існують, і запроваджувати нові хмарні сервіси, а також переносити деякі наявні функції в хмару.

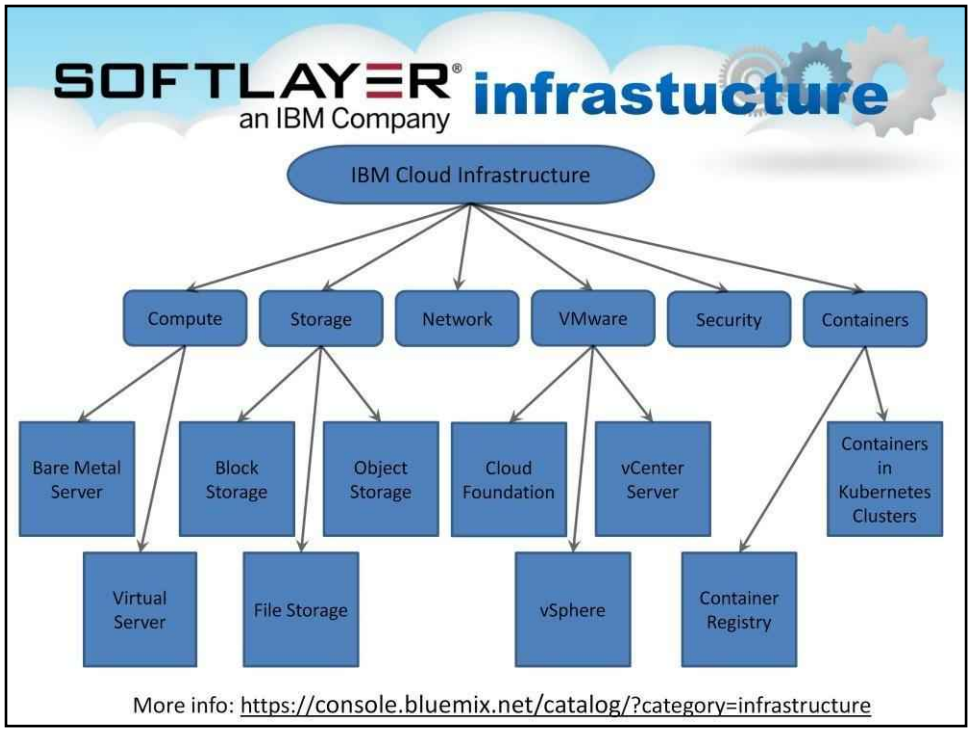
Для підтримки цього IBM пропонує публічні та приватні хмарні платформи для створення, запуску та керування програмами.

Публічна хмарає багатокористувачем і розміщено в центрі обробки даних постачальника.

Приватна хмарає одним орендарем та/або розміщено в корпоративному центрі обробки даних.

Забезпечуючи додаткову гнучкість, середовища Bluemix можна розгорнути в приватному хмарному середовищі з одним клієнтом (**Bluemix Dedicated**) або

приватний локальний центр обробки даних клієнтів (**Місцевий Bluemix**).



Bluemix працює на інфраструктурі SoftLayer. На цьому слайді ви можете побачити його основні компоненти. **Голі сервери** забезпечують необхідну потужність, необхідну для робочих навантажень із інтенсивним використанням процесора та дискового введення/виведення.

Віртуальні сервериможна розгорнути за лічені хвилини з образів віртуальних серверів у географічному регіоні, який є доцільним для певних робочих навантажень.

Блок зберіганняце постійне сховище на основі iSCSI з високою продуктивністю та ємністю до 12 ТБ.

Зберігання файлівце швидке та гнучке сховище файлів на основі NFS із варіантами ємності від 20 ГБ до 12 ТБ.

Зберігання об'єктівце високомасштабована служба хмарного зберігання даних, призначена для зберігання, керування та доступу до даних через портал самообслуговування та RESTful API.

VMware**Cloud Foundation**на IBM Cloud об’єднує VMware vSphere, vSAN і NSX у вбудований стек віртуальних обчислень, віртуальних сховищ і віртуальних мереж, побудованих на виділеній інфраструктурі IBM Cloud compute.

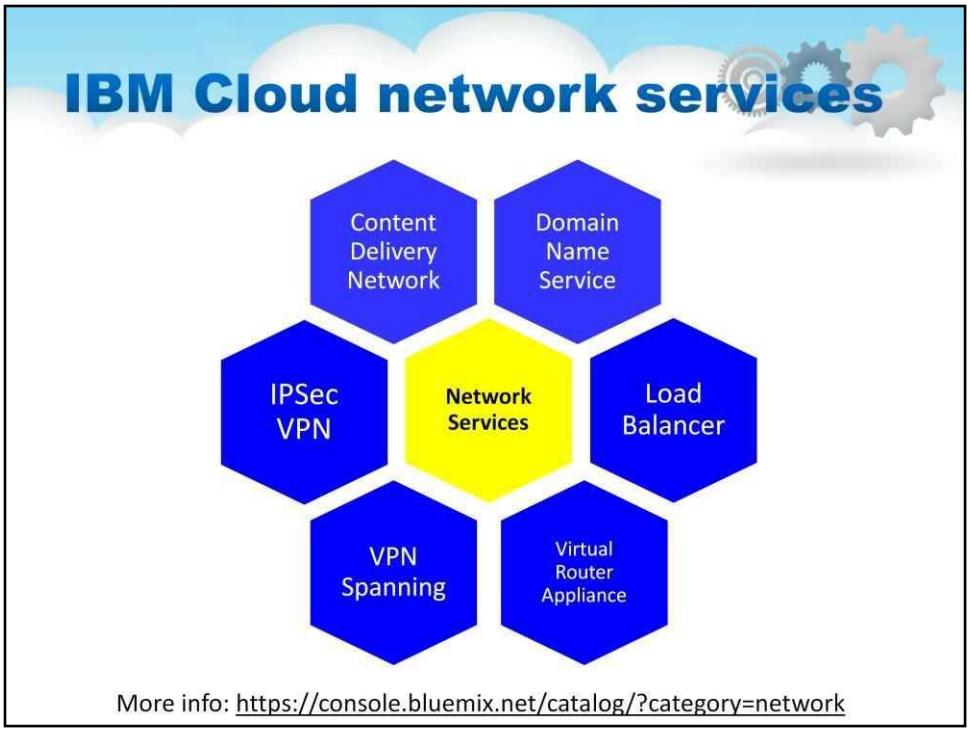
VMware**vSphere**на IBM Cloud забезпечує максимальну гнучкість для створення середовища, розміщеного на сервері IBM, використовуючи апаратне забезпечення, сумісне з VMware, і правильний набір компонентів VMware, які відповідають бізнес-потребам і знанням.

VMware**vCenter**Сервер у IBM Cloud — це розміщена приватна хмара, яка надає стек VMware vSphere як послугу.

Кластер Kubernetesдозволяє безпечно керувати ресурсами, необхідними для швидкого розгортання, оновлення та масштабування програм.

Реєстр контейнерівможна від’єднати від IBM Cloud Kubernetes Service, щоб зберігати та розповсюджувати образи Docker через серверну програму без збереження стану.

Доступні мережеві служби будуть описані на наступному слайді.



Мережа доставки вмістусервіс поширює контент там, де це потрібно.

Коли вміст запитується вперше, він завантажується з хост-сервера в мережу та залишається там, щоб інші користувачі могли отримати до нього доступ.

Служба доменних імен.IBM Cloud пропонує послуги реєстрації доменів разом із спеціальним персоналом підтримки, кваліфікованим обслуговуванням клієнтів і розумними цінами, і все це надається через безпечну мережу.

Віртуальна приватна мережа (VPN).Цей доступ призначений для того, щоб дозволити користувачам віддалено керувати всіма серверами та службами, пов’язаними з їхнім обліковим записом, через нашу приватну мережу.

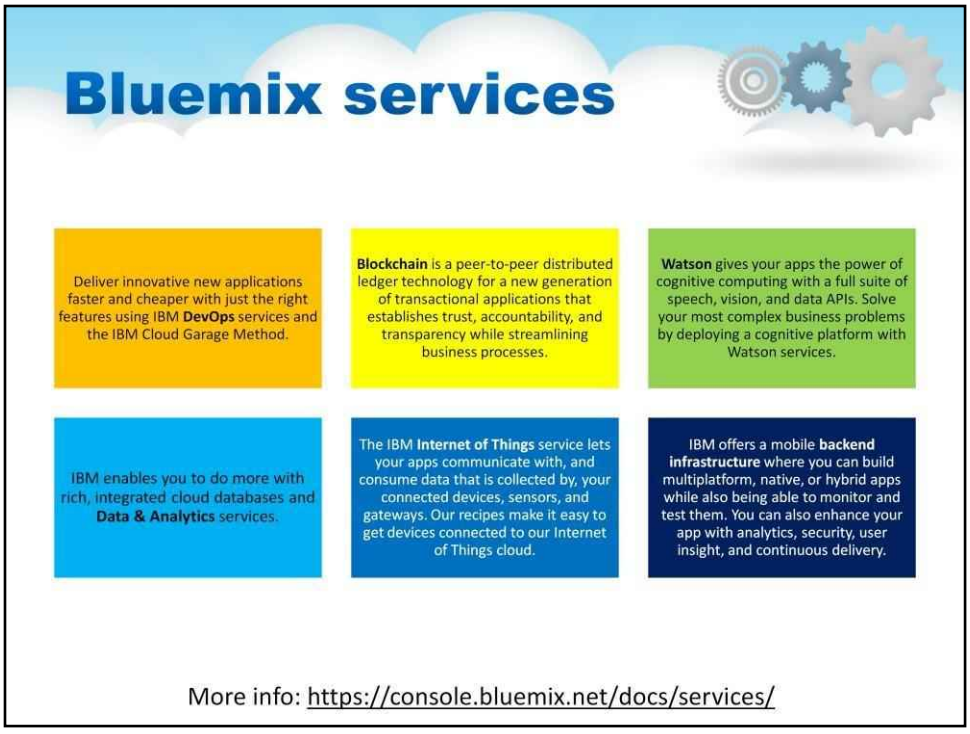
Балансувальник навантаженняСлужба розподіляє трафік між серверами додатків, розташованими локально в центрі обробки даних.

VPN Spanning. Він з’єднує всі приватні мережеві VLAN в обліковому записі, дозволяючи пристроям в окремих VLAN спілкуватися один з одним.

Віртуальний маршрутизатор (VRA)надає найновішу мережеву операційну систему Vyatta для серверів x86.

Можна створювати віртуальні маршрутизатори, брандмауери та VPN, які відповідають унікальним вимогам програми.

Більше інформації можна отримати за посиланням, наведеним тут.

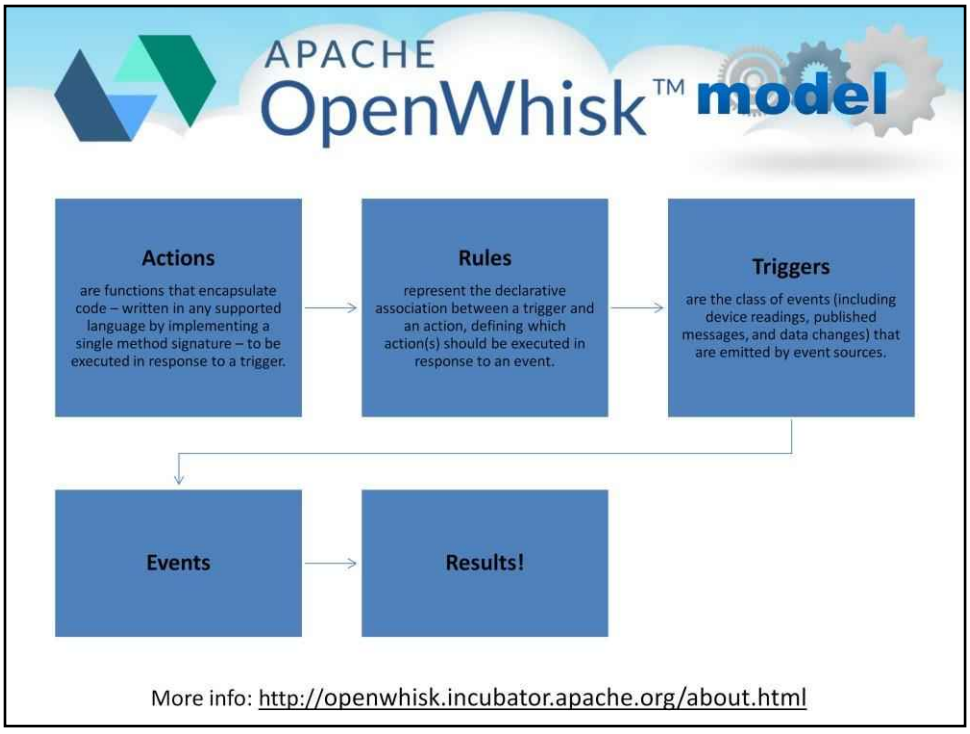


Інформаційна панель IBM Cloud забезпечує доступ до служб IBM Cloud, доступних від IBM і сторонніх постачальників.

Серед них Watson, Internet of Things, Data & Analytics, Mobile і DevOps.

На цьому слайді ви можете побачити короткий опис кожної послуги.

Більше інформації можна отримати за посиланням, наведеним тут.



IBM Bluemix включає систему IBM Function as a Service (FaaS) або пропозицію безсерверних обчислень, створену з відкритим вихідним кодом із проекту інкубатора Apache OpenWhisk, який в основному приписується IBM для заповнення.

OpenWhisk — це розподілена безсерверна обчислювальна платформа з відкритим кодом, здатна виконувати логіку додатків (**Дії**) у відповідь на події (**Тригери**) із зовнішніх джерел (**Канали**) або запити HTTP, керовані умовною логікою (**правила**).

Він забезпечує середовище програмування, яке підтримується інтерфейсом командного рядка (CLI) на основі REST API, а також інструменти для підтримки служб пакування та каталогів.

Більше інформації можна отримати за посиланням, наведеним тут.

Watson — це хмарний сервіс IBM для когнітивних обчислень із такими компонентами: **Студія IBM Watson** прискорює робочі процеси машинного та глибокого навчання, необхідні для впровадження ШІ в бізнес для стимулювання інновацій.

Каталог знань IBM Watsonзабезпечує інтелектуальне самообслуговування виявлення даних, моделей тощо, активуючи їх для штучного інтелекту, машинного та глибокого навчання.

Watson DiscoveryФункція виконує когнітивний пошук і аналіз вмісту, щоб виявити закономірності, тенденції та корисну інформацію для прийняття кращих рішень.

Новини Discoveryозначає виявлення тенденцій і закономірностей у настроях за допомогою сукупного аналізу, щоб побачити нові перспективи того, як новини розгортаються по всьому світу.

Розуміння природної мови. Це дозволяє аналізувати текст для отримання метаданих із вмісту, таких як поняття, сутності, ключові слова, категорії, емоції, відносини, семантичні ролі.

Служба перетворення мовлення в текстперетворює людський голос на письмове слово.

Служба перетворення тексту в мовленняобробляє текст і природну мову для створення синтезованого аудіовиходу з відповідною частотою та інтонацією.

Watson Language Translatorдозволяє динамічно перекладати текст і миттєво публікувати вміст кількома мовами.

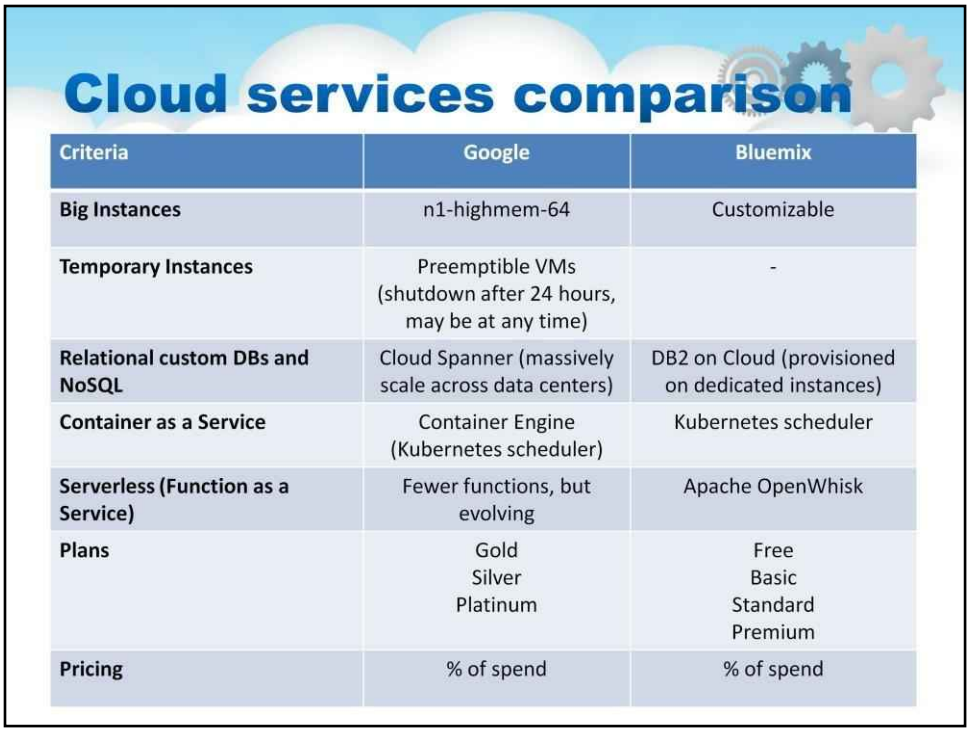
Служба класифікатора природної мовизастосовує методи когнітивного обчислення, щоб повернути найкращі відповідні класи для речення чи фрази.

Особистість Insightsотримує інформацію з даних транзакцій і соціальних мереж, щоб визначити психологічні риси, які визначають рішення про купівлю, наміри та поведінкові риси. **Аналізатор тону**використовує когнітивний лінгвістичний аналіз для виявлення різноманітних тонів як на рівні речення, так і на рівні документа.

Watson Assistantзабезпечує інтерфейс природною мовою для автоматизації взаємодії з користувачами.

Візуальне розпізнаваннядозволяє аналізувати зображення для сцен, об’єктів, облич та іншого вмісту.

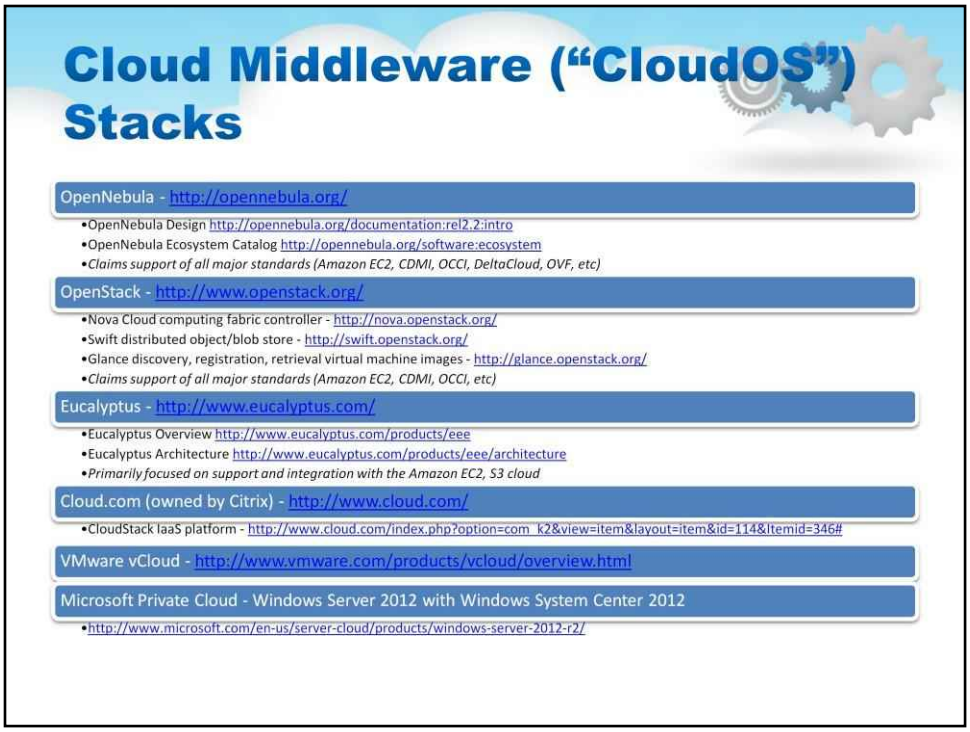
Більше інформації можна отримати за посиланням, наведеним тут.



На останньому слайді ви можете побачити відмінності Google та IBM Clouds у деяких ключових моментах.

Рішення з відкритим кодом

Давайте розглянемо деякі рішення з відкритим кодом...

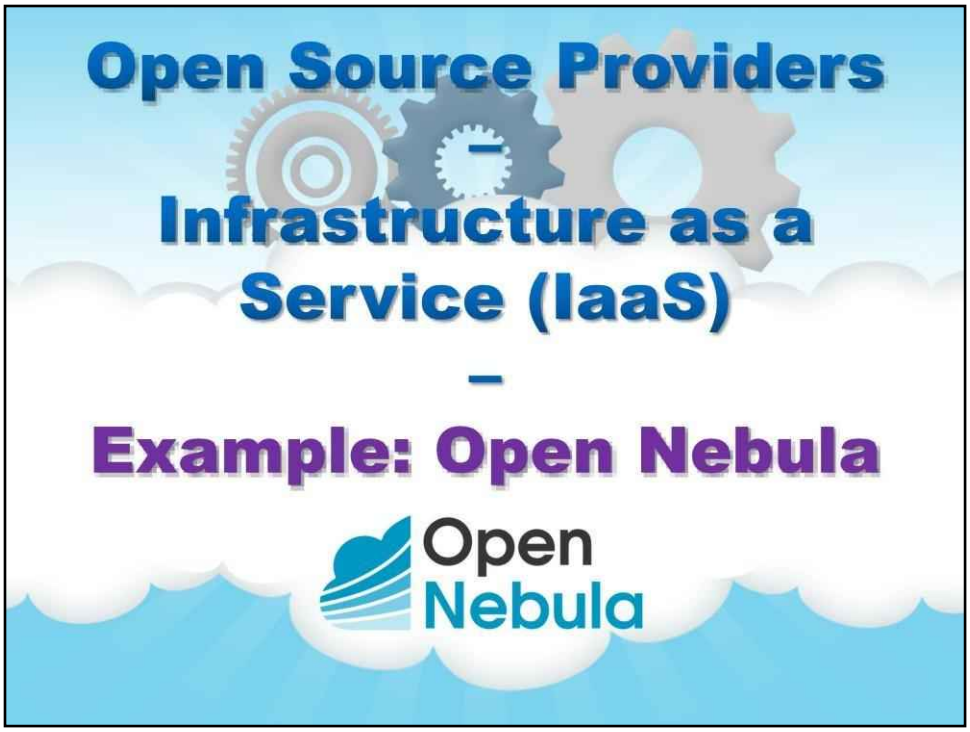


Існує кілька різних стеків Cloud Middleware.

Ці «стеки» часто називають «CloudOS», хоча насправді вони є проміжним програмним забезпеченням.

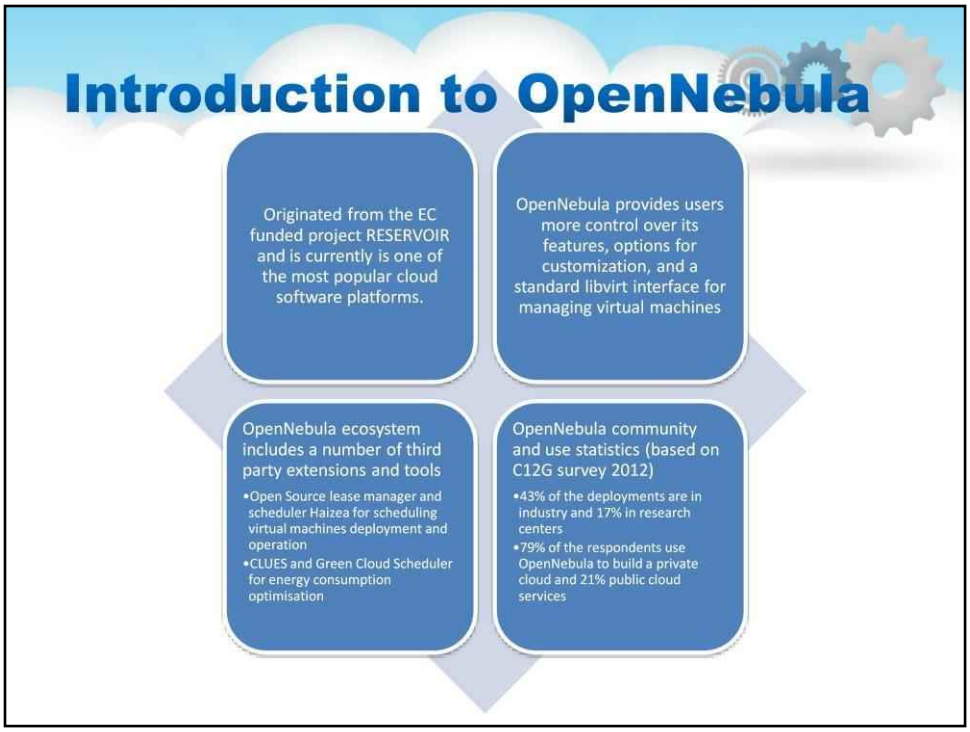
Кожен із серверів у хмарі фактично працює під керуванням операційної системи, такої як Linux або Windows, і включає компонент віртуалізації, такий як Xen, KVM, ESX або Hyper-V.

У цій частині лекції ми починаємо з OpenStack...



Приклад: OpenNebula

Перший приклад присвячений OpenNebula, який представляє приклад інфраструктури як послуги (IaaS).



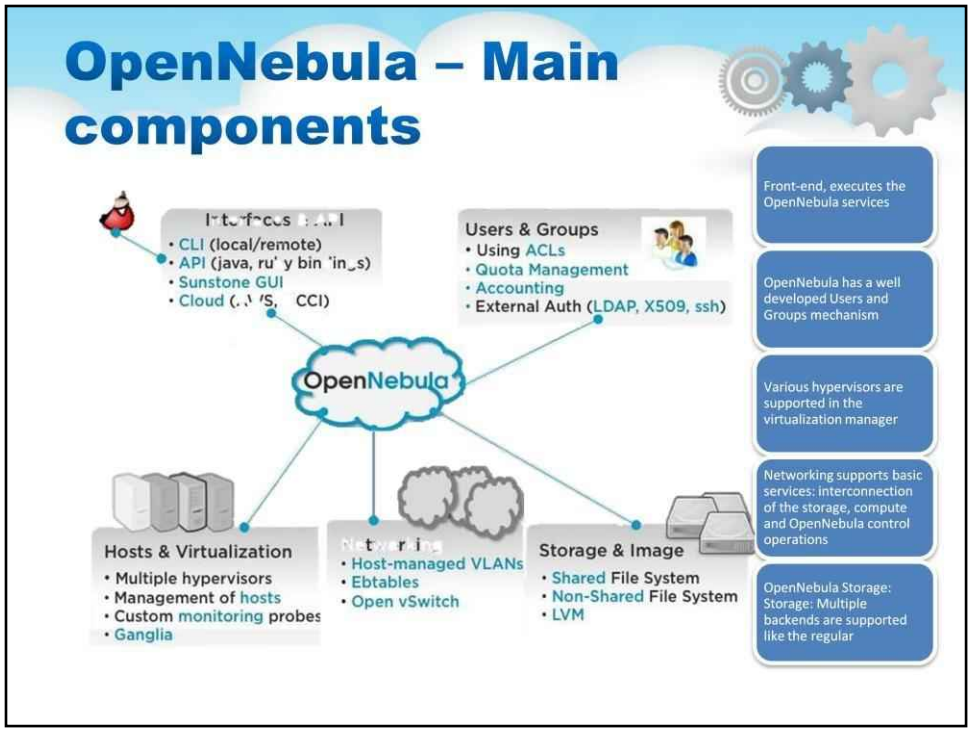
OpenNebula вперше було створено як дослідницький проект ще в 2005 році в рамках проекту RESERVOIR, що фінансується ЄС. З моменту свого першого публічного випуску програмного забезпечення в березні 2008 року воно розвивалося через випуски з відкритим кодом і зараз працює як проект з відкритим кодом.

> Перша версія 1.0 була випущена в 2008 році, і з тих пір проект спонсорується Дослідницькою групою архітектури розподілених систем (http://dsaresearch.org) в Мадридському університеті Computense.

Технологія OpenNebula стала дорослішою завдяки активній та залученій спільноті користувачів і розробників. Його програмне забезпечення завантажувалося з сайту кілька тисяч разів на місяць. Окрім експоненційного зростання кількості користувачів, різні проекти, дослідницькі групи та компанії створили нові компоненти віртуалізації та хмари, щоб доповнити та покращити функціональні можливості цього широко використовуваного набору інструментів з відкритим кодом для хмарних обчислень.

У березні 2010 року головні автори OpenNebula заснували C12G Labs, щоб надавати додаткові професійні послуги, які потрібні багатьом корпоративним IT-магазинам для внутрішнього впровадження, і щоб проект OpenNebula не був прив'язаний виключно до державного фінансування, сприяючи його довгостроковій стабільності. У вересні 2013 року OpenNebula організувала свою першу конференцію спільноти, на якій виступили провідні організації з усього світу.

OpenNebula надає користувачам більше контролю над своїми функціями, параметрами налаштування та стандартним інтерфейсом libvirt для керування віртуальними машинами. Використовує NFS для зберігання образів дисків. OpenNebula має планувальник, який може виконувати основні завдання планування віртуальних машин.



OpenNebula має розширену мультитенантність із потужним керуванням користувачами та групами, детальним розподілом ресурсів і керуванням квотами ресурсів для відстеження та обмеження використання обчислень, сховища та мережі.

Керування користувачами: OpenNebula може перевіряти користувачів за допомогою власної внутрішньої бази даних користувачів на основі паролів або зовнішніх механізмів, таких як ssh, x509, ldap або Active Directory **Віртуалізація OpenNebula:**У диспетчері віртуалізації підтримуються різні гіпервізори з можливістю контролювати повний життєвий цикл віртуальних машин і кількох гіпервізорів в одній хмарній інфраструктурі.

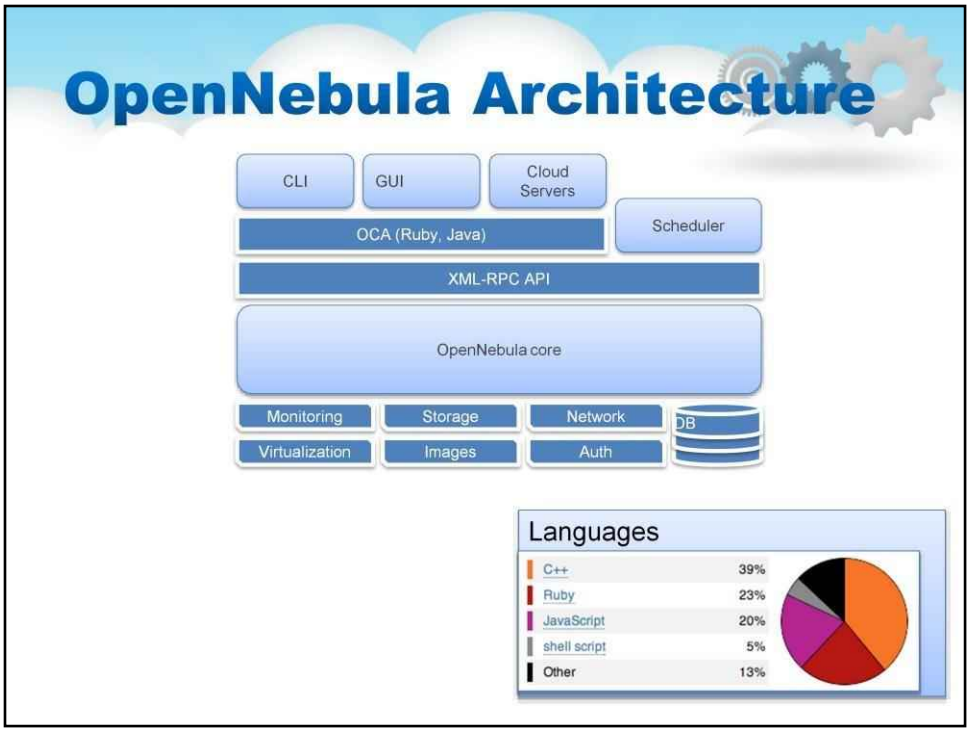
Віртуалізація:Кілька технологій гіпервізора повністю підтримуються, як-от Xen, KVM і VMware.**Моніторинг:** OpenNebula надає власну налаштовану систему моніторингу, а також інтегрується з інструментами моніторингу центру обробки даних, такими як Ganglia.

Хости OpenNebula:Менеджер хостів надає повну функціональність для керування фізичними хостами в хмарі.

OpenNebula має потужну мережу. Динамічне створення кластерів як пулів хостів, які спільно використовують сховища даних і віртуальні мережі для балансування навантаження, високої доступності та високопродуктивних обчислень.

Мережа OpenNebula:Мережева підсистема, яка легко адаптується та настроюється, присутня в OpenNebula для кращої інтеграції з конкретними мережевими вимогами існуючих центрів обробки даних і забезпечення повної ізоляції між віртуальними машинами, які складають віртуалізований сервіс.

Сховище OpenNebula:Підтримуються кілька серверних модулів, як-от звичайне (спільне чи ні) сховище даних файлової системи, що підтримує популярні розподілені файлові системи, такі як NFS, Lustre, GlusterFS; сховище даних VMware, спеціалізоване для гіпервізора VMware, сховище даних iSCSI/LVM для зберігання образів дисків у формі блокових пристроїв; і Ceph для розподіленого блокового пристрою.



Цей слайд містить загальний вигляд архітектури OpenNebula з такими основними компонентами:

1.CLI - інтерфейс командного рядка

2.GUI – графічний інтерфейс користувача

3.CloudServer

4. OCA - OpenNebula Cloud API

5. Шедулер

6. API XML_RPC

7. Ядро OpenNebula

8. Моніторингові послуги

9. Складські послуги

10. Мережеві служби

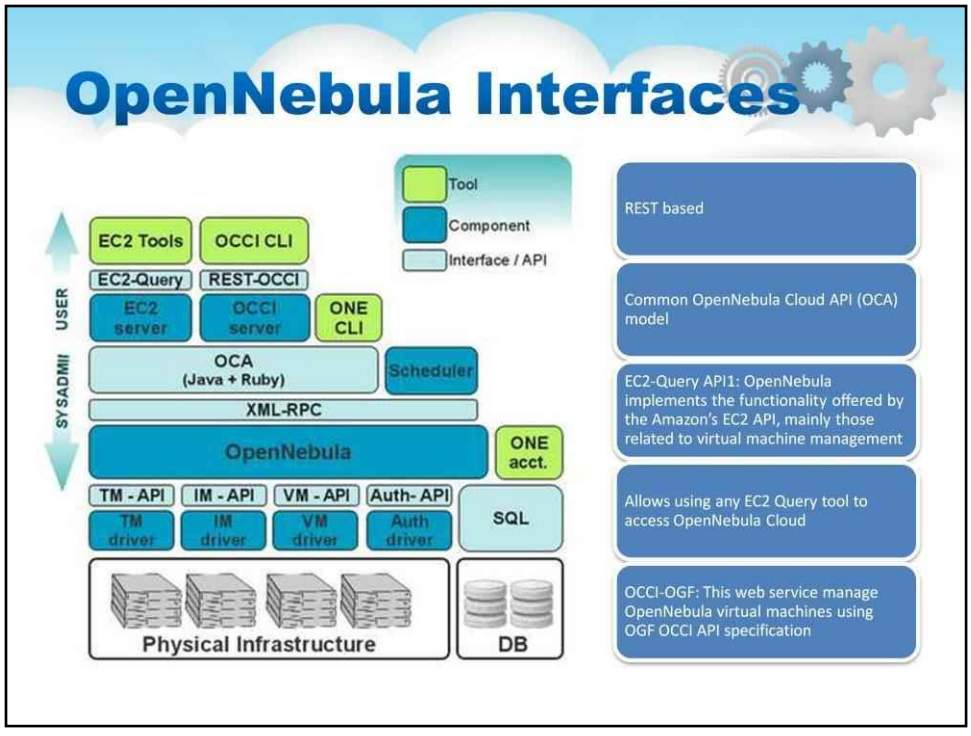
11. Послуги віртуалізації

12. Зображення

13. Послуги автентифікації

14.DB – служби баз даних

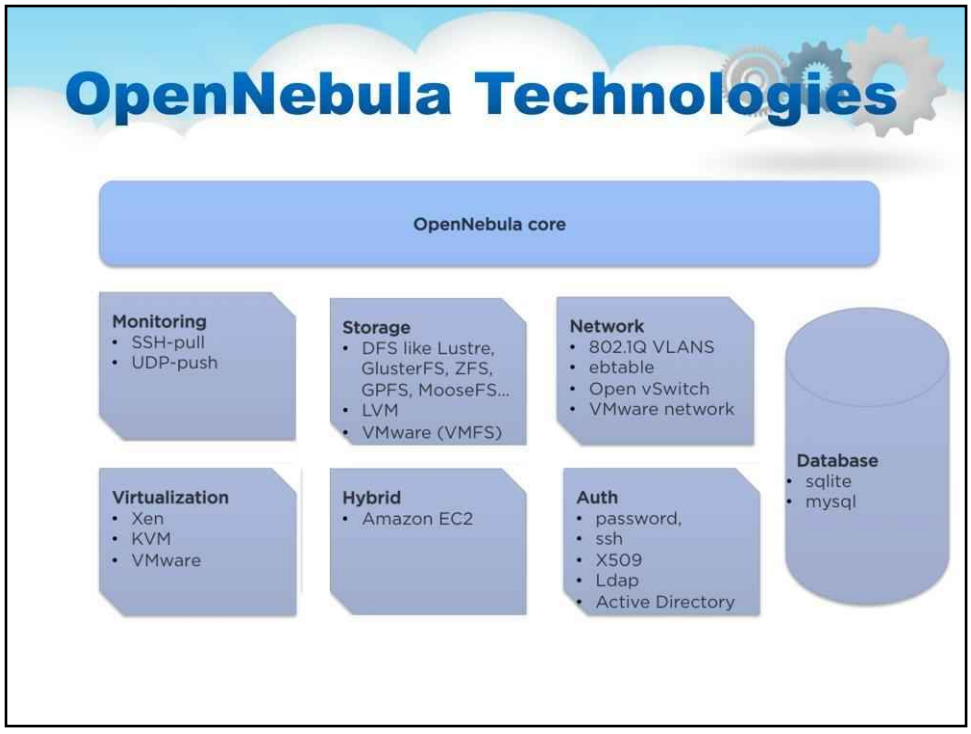
Тут також показано підтримувані мови та відсоток їх користувачів у всьому світі.



Хмарні інтерфейси дозволяють керувати віртуальними машинами, мережами та зображеннями за допомогою простого та легкого у використанні REST API. Хмарні інтерфейси приховують більшу частину складності хмари та спеціально підходять для кінцевих користувачів. OpenNebula реалізує два різні інтерфейси, а саме:

-EC2-Query API. OpenNebula реалізує функціональні можливості, які пропонує Amazon EC2 API, в основному пов'язані з керуванням віртуальними машинами. Таким чином, ви можете використовувати будь-який інструмент запиту EC2 для доступу до своєї хмари OpenNebula.-

-OCCI-OGF. Ця веб-служба дозволяє запускати та керувати віртуальними машинами у вашій інсталяції OpenNebula за допомогою останньої чернетки специфікації API OGF OCCI.-



На цьому слайді описуються основні технології OpenNebula, що лежать в основі його компонентів.

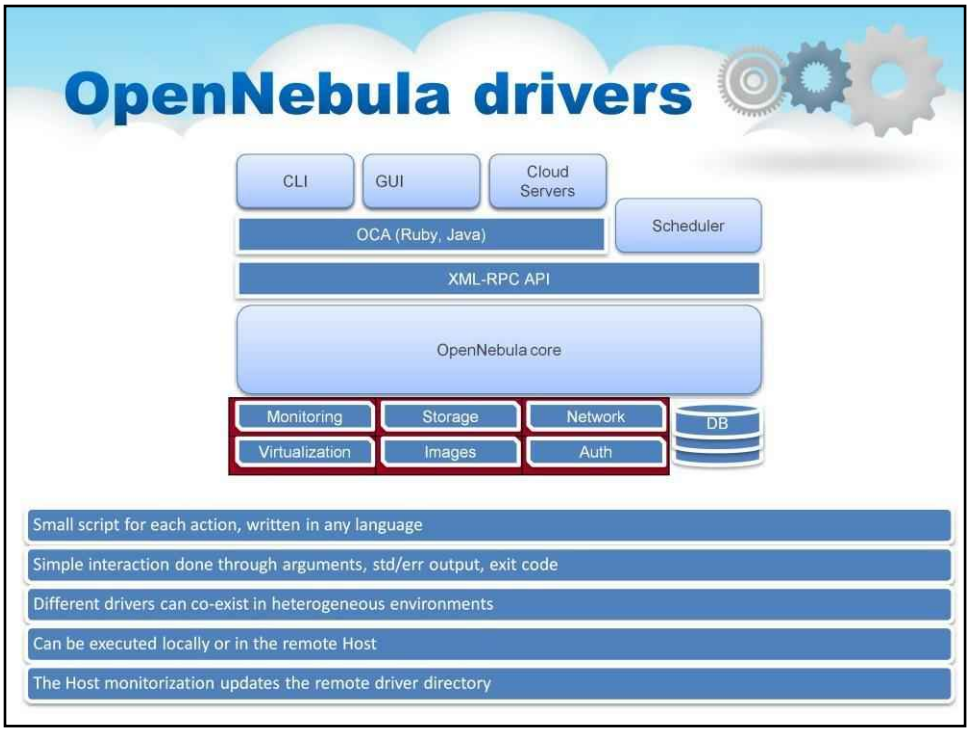
Розширений контроль і моніторинг віртуальної інфраструктури

Підсистема сховища зображеньз каталогом і повною функціональністю для керування зображеннями віртуальної машини. Підсистема сховища шаблонів із каталогом і повною функціональністю для керування шаблонами віртуальних машин. Повний контроль життєвого циклу екземпляра віртуальної машини та повна функціональність для керування екземпляром віртуальної машини. Розширені функції для динамічного керування віртуальною машиною, як-от знімок системи та диска, зміна розміру ємності або гаряче підключення мережевих карт. Програмовані операції віртуальної машини, що дозволяє користувачам планувати дії

Підсистема зберіганняз підтримкою кількох сховищ даних, щоб збалансувати операції вводу/виводу між серверами зберігання або визначити різні політики SLA (наприклад, резервне копіювання) і характеристики продуктивності для різних типів віртуальних машин або користувачів. Він підтримує будь-яку серверну конфігурацію з різними типами сховищ даних.

Гнучка мережева підсистема.

Агностична віртуалізація гіпервізораіз широкою підтримкою гіпервізорів (Xen, KVM і VMware), централізованим керуванням середовищами з кількома гіпервізорами та підтримкою кількох гіпервізорів в одній фізичній коробці.
Гібридні хмарні обчислення та Cloudbursting.Розширення локальної приватної інфраструктури ресурсами з віддалених хмар.
Рамка авторизації. Безпечна та ефективна підсистема користувачів і груп для автентифікації та авторизації запитів із повною функціональністю для керування користувачами.
Розширена мультиоренда з груповим керуванням. Адміністратори можуть групувати користувачів в організації, які можуть представляти різні проекти, підрозділи.
База даних.Високомасштабована база даних із підтримкою MySQL і SQLite



На цьому слайді описуються інтерфейси драйверів OpenNebula.

Взаємодія між OpenNebula та хмарною інфраструктурою виконується спеціальними драйверами, кожен з яких стосується певної області:

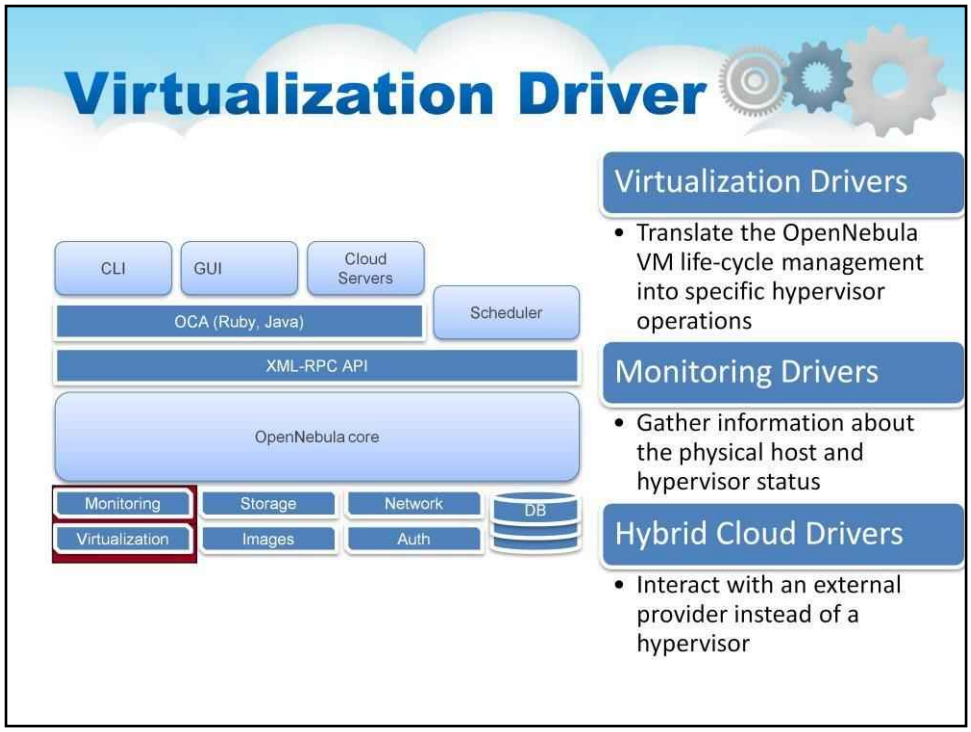
Зберігання.Ядро OpenNebula випускає абстрактні операції зберігання (наприклад, клонування або видалення), які реалізуються спеціальними програмами, які можна замінити або модифікувати для інтерфейсу спеціальних систем зберігання та файлових систем.

Віртуалізація.Взаємодія з гіпервізорами також реалізована за допомогою спеціальних програм для завантаження, зупинки або міграції віртуальної машини. Це дозволяє вам спеціалізувати кожну операцію віртуальної машини, щоб виконувати спеціальні операції.

Моніторинг.Інформація моніторингу також збирається зовнішніми датчиками. Ви можете додати додаткові зонди, щоб включити користувацькі показники моніторингу, які пізніше можна використовувати для розподілу віртуальних машин або для цілей обліку

Авторизація. OpenNebula також можна налаштувати на використання зовнішньої програми для авторизації та автентифікації запитів користувачів. Таким чином можна реалізувати будь-яку політику доступу до хмарних ресурсів.

База даних. OpenNebula зберігає свій стан і багато облікової інформації в постійній базі даних. OpenNebula може використовувати базу даних MySQL або SQLite, яку можна легко поєднати з будь-яким інструментом БД.



У OpenNebula можлива додаткова інтеграція з інструментами моніторингу центру обробки даних, такими як Ganglia.

Примітка. Ganglia — це масштабований розподілений інструмент моніторингу для високопродуктивних обчислювальних систем, кластерів і мереж.

Якщо у вашому кластері вже розгорнуто ганглії, ви можете використовувати драйвери гангліїв, надані OpenNebula, щоб отримати з нього інформацію про хости та віртуальні машини. Ці драйвери мають зробити моніторинг ефективнішим у великій установці, оскільки вони не використовують з'єднання ssh з вузлами для отримання інформації. З іншого боку, вони вимагають додаткової роботи від системного адміністратора, оскільки ганглії повинні бути правильно налаштовані, а завдання cron повинні бути встановлені на вузлах, щоб надавати інформацію про віртуальну машину системі гангліїв.

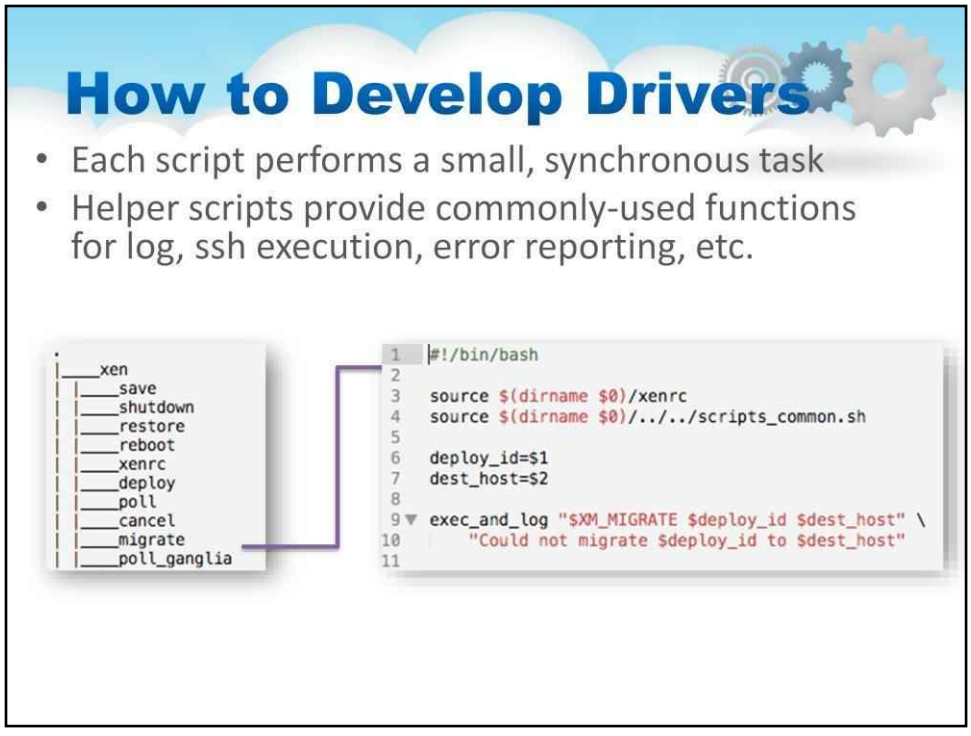
Компонент, який працює з гіпервізором для створення, керування та отримання інформації про об'єкти віртуальної машини, називається Virtual Machine Manager (скорочено VMM).

Наступні драйвери для основних гіпервізорів вже написані:

-Драйвер KVM 4.0 - KVM (віртуальна машина на основі ядра) є повною технікою віртуалізації для Linux. Він пропонує повну віртуалізацію, коли кожна віртуальна машина взаємодіє зі своїм власним віртуалізованим обладнанням.-

-Драйвер Xen 4.0 - гіпервізор XEN пропонує потужний, ефективний і безпечний набір функцій для віртуалізації x86, IA64, PowerPC та інших архітектур ЦП. Він забезпечує як паравіртуалізацію, так і повну віртуалізацію.-

- Драйвери VMware 4.0 – драйвери VMware дозволяють керувати хмарою OpenNebula на основі гіпервізорів VMware ESX та/або VMware Server. Вони використовують функцію простого процесу налаштування, який використовуватиме стабільність, продуктивність і набір функцій будь-якої існуючої хмари OpenNebula на основі VMware.-



Цей слайд містить опис того, як написані драйвери віртуалізації.

Цей компонент складається з двох частин:

- перший знаходиться в ядрі та містить більшість загальної функціональності, спільної для всіх драйверів (і деяких специфічних),
- другий — драйвер, здатний транслювати базові дії VMM на гіпервізор.

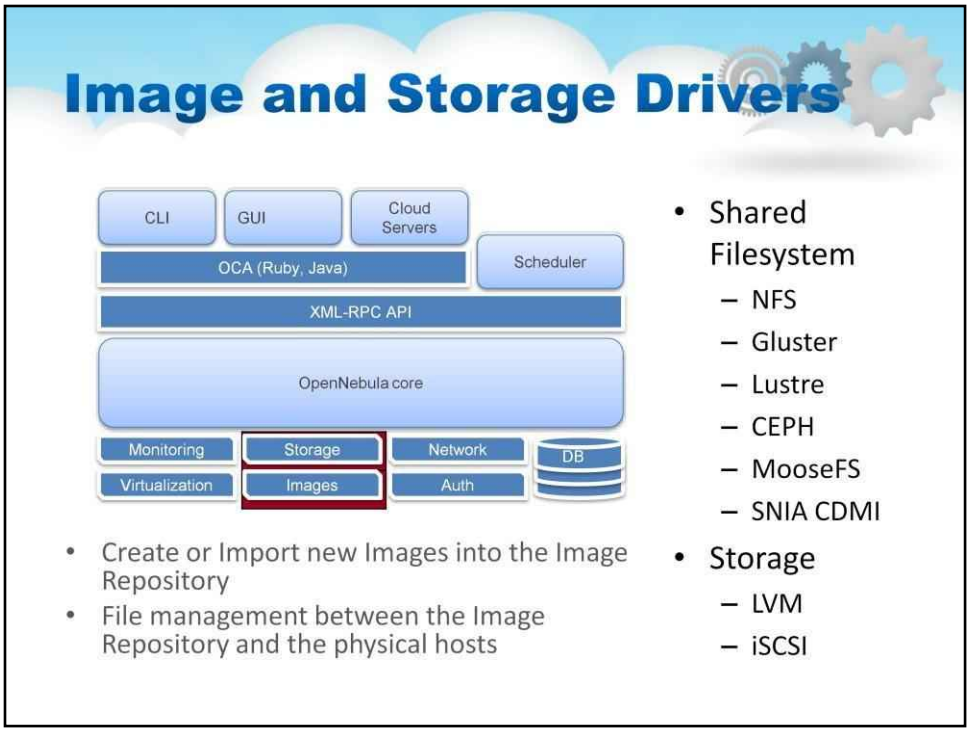
Можливі дії:

DEPLOY - повідомляє гіпервізору створити нову віртуальну машину. SHUTDOWN - надсилає сигнал завершення роботи на віртуальну машину. CANCEL - знищує віртуальну машину.

SAVE – зберігає стан віртуальної машини (призупинення) RESTORE – відновлює віртуальну машину до попереднього збереженого стану MIGRATE – виконує живу міграцію віртуальної машини

POLL - отримує інформацію про віртуальну машину MIGRATE - виконує живу міграцію віртуальної машини

Кожна дія повинна мати виконувану програму (головним чином сценарії), розташовану у віддаленому каталозі, яка виконує бажану дію.



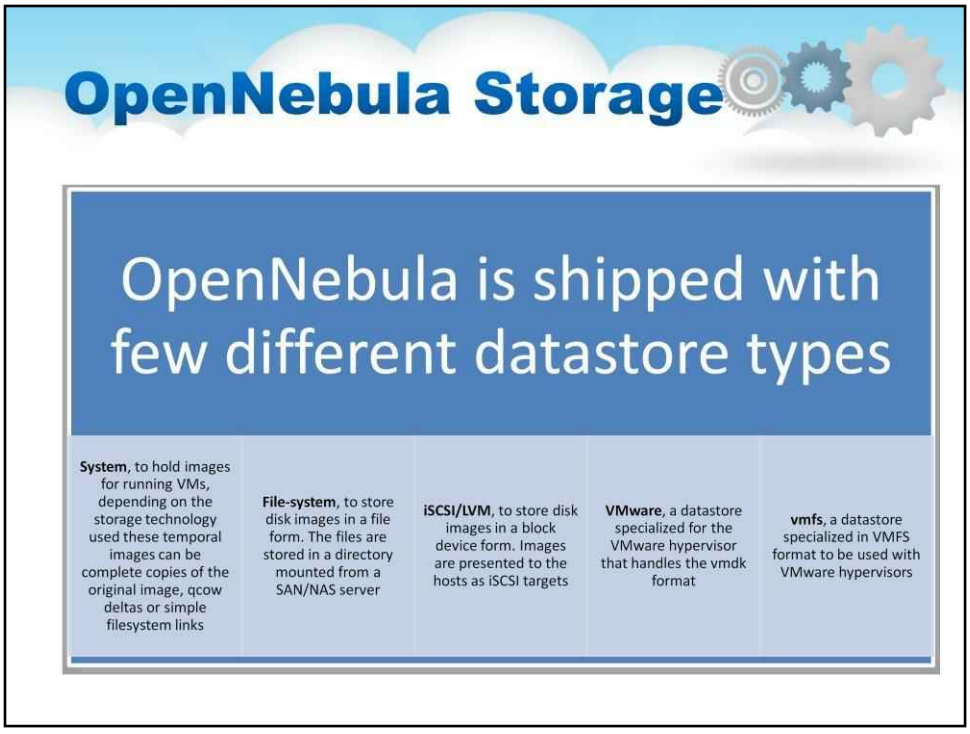
Підсистема зберігання є високомодульною.

Ці драйвери розділені на дві логічні групи:

DS: драйвери сховища даних. Вони служать для керування зображеннями: реєструють, видаляють і створюють порожні блоки даних.

TM: драйвери Transfer Manager. Вони керують зображеннями, пов’язаними зі створеними віртуальними машинами.

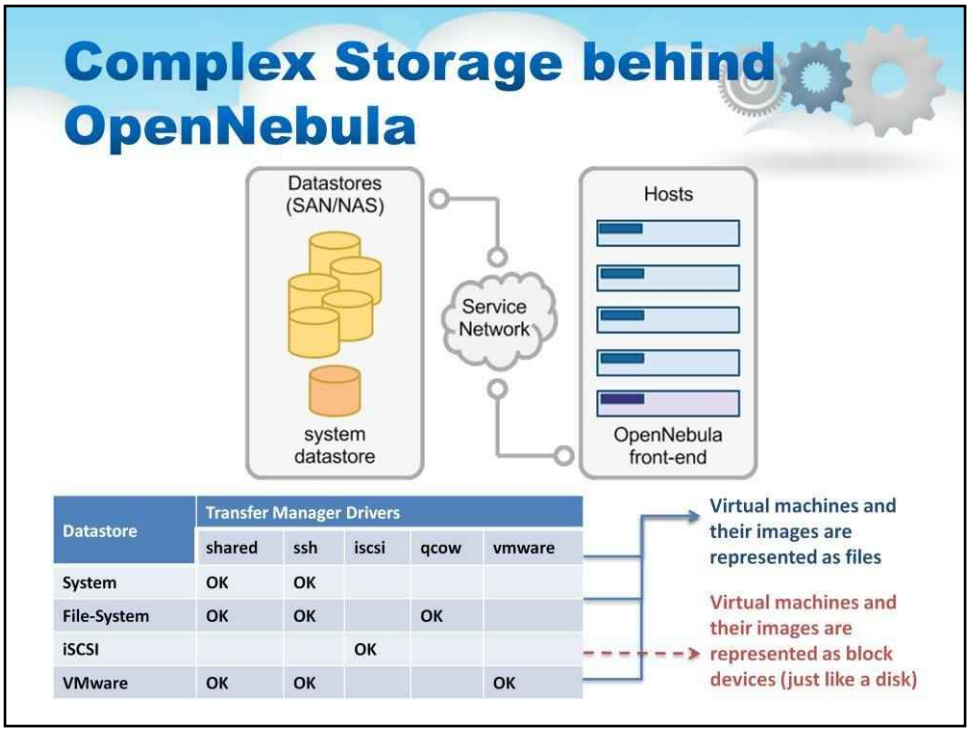
Образи дисків, зареєстровані в сховищі даних, передаються на хости драйверами диспетчера передавання (TM). Ці драйвери є спеціалізованими частинами програмного забезпечення, які виконують низькорівневі операції зберігання (наприклад, налаштування цілей iSCSI або переміщення файлів).



Механізм передачі визначається для кожного сховища даних. Таким чином один хост може одночасно отримати доступ до кількох сховищ даних, які використовують різні драйвери передачі. Зауважте, що хости мають бути налаштовані для належного доступу до кожного типу сховища даних (наприклад, монтування спільних ресурсів FS).

OpenNebula постачається з різними типами сховищ даних:

1. Система для зберігання зображень для запущених віртуальних машин, залежно від використовуваної технології зберігання.
2. Файлова система для зберігання образів дисків у файловій формі.
3. iSCSI/LVM, для зберігання образів дисків у формі блокових пристроїв.
4. VMware, спеціалізоване сховище даних для гіпервізора VMware.
5. vmfs, спеціалізоване сховище даних у форматі VMFS для використання з гіпервізорами VMware.
6. Ceph, для зберігання образів дисків за допомогою блокових пристроїв Ceph.
7. Файли, це спеціальне сховище даних, яке використовується для зберігання звичайних файлів, а не образів дисків. Звичайні файли можна використовувати як ядра, RAM-диски або контекстні файли.



Datastore — це будь-який носій даних, який використовується для зберігання образів дисків для віртуальних машин, попередні версії OpenNebula називають цю концепцію Image Repository.

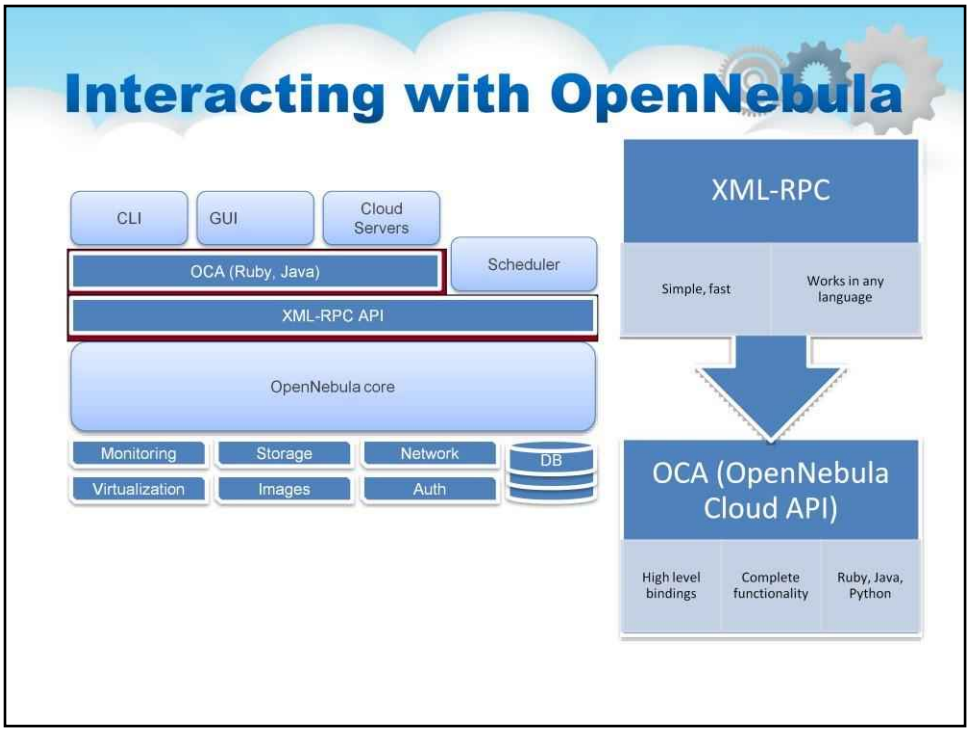
Інсталяція OpenNebula може мати кілька сховищ даних кількох типів для зберігання образів дисків. OpenNebula також використовує спеціальне сховище даних, системне сховище даних, щоб зберігати зображення запущених віртуальних машин. Ви можете скористатися багатьма функціями сховища даних OpenNebula, щоб краще масштабувати сховище для своїх віртуальних машин, зокрема:

Балансування операцій введення-виведення між серверами зберігання

Різні типи віртуальних машин або користувачі можуть використовувати сховища даних із різними функціями продуктивності.

Різні політики SLA (наприклад, резервне копіювання) можна застосовувати до різних типів віртуальних машин або користувачів. Легко додайте нове сховище до хмари

Існують певні обмеження та функції залежно від механізму передачі, який ви вибрали для сховищ даних вашої системи та зображень (перевірте посібник із кожного сховища даних для отримання додаткової інформації). У таблиці наведено дійсні комбінації драйверів Datastore і передачі.



На цьому слайді описано інтерфейси OpenNebula

Інтерфейс XML-RPC

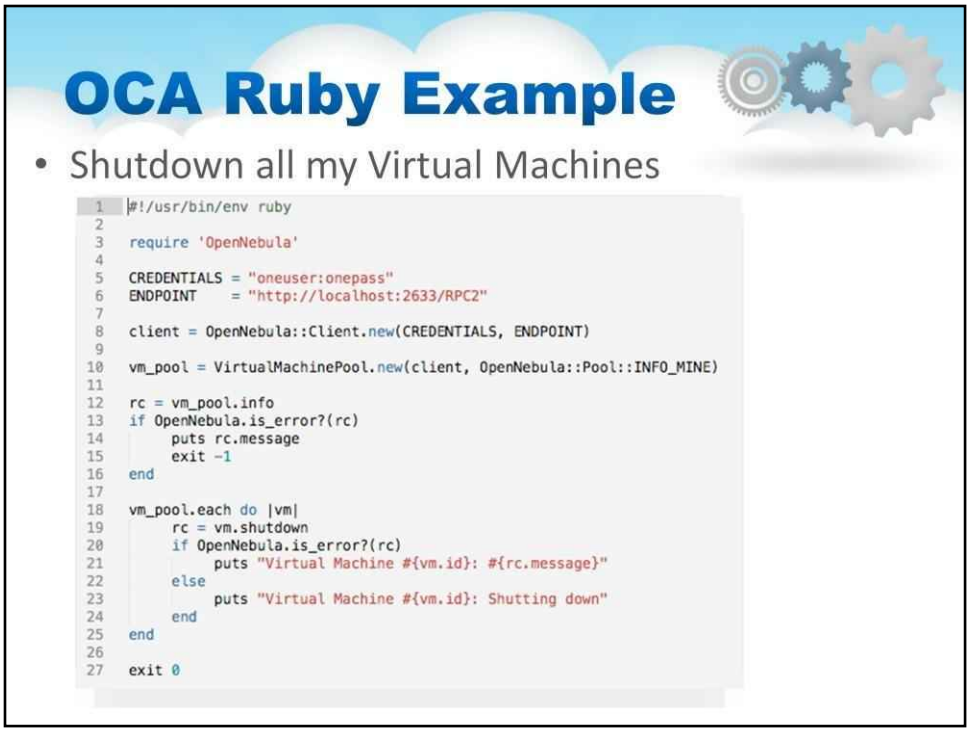
Це основний інтерфейс для OpenNebula, і він надає всі функції для інтерфейсу демона OpenNebula. За допомогою інтерфейсу XML-RPC ви можете керувати будь-яким ресурсом OpenNebula, включаючи віртуальні машини, мережі, зображення, користувачів, хости та кластери.

Інтерфейс XML-RPC слід використовувати, якщо ви розробляєте спеціалізовані бібліотеки для хмарних програм або вам потрібен низькорівневий інтерфейс з ядром OpenNebula.

OpenNebula Cloud API (OCA)

Він забезпечує спрощений і зручний спосіб інтерфейсу ядра OpenNebula. Інтерфейси OCA надають ту саму функціональність, що й інтерфейс XML-RPC. OpenNebula включає дві мовні прив’язки для OCA: Ruby та JAVA.

Інтерфейс OCA слід використовувати, якщо ви розробляєте розширені інструменти IaaS, яким потрібен повний доступ до функцій OpenNebula.



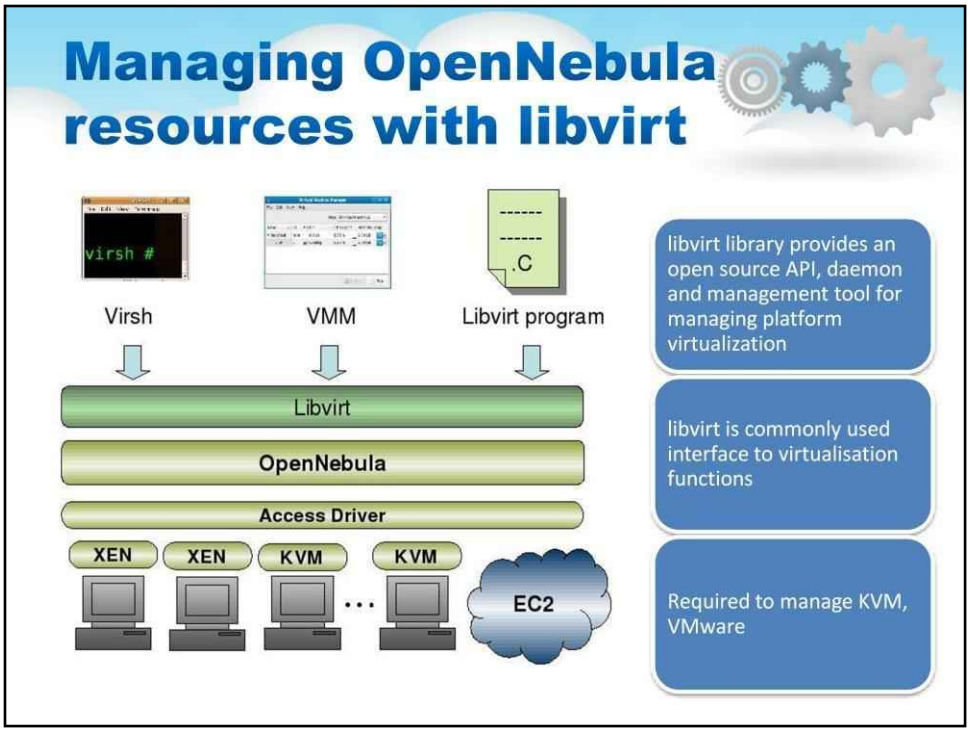
Специфікація OpenNebula Cloud API для Ruby була розроблена як оболонка для методів XML-RPC з деякими базовими помічниками.

Тут представлено зразок коду для завершення роботи всіх віртуальних машин пулу. Потік коду буде таким:

-Створіть новий клієнт, налаштувавши рядок авторизації. Вам потрібно створити його лише один раз.-

- Отримайте пул VirtualMachine, який містить віртуальні машини, якими володіє цей користувач.-

- Ви можете негайно виконувати «дії» над цими об’єктами, наприклад myVNet.delete();. У цьому прикладі всі віртуальні машини будуть вимкнені.

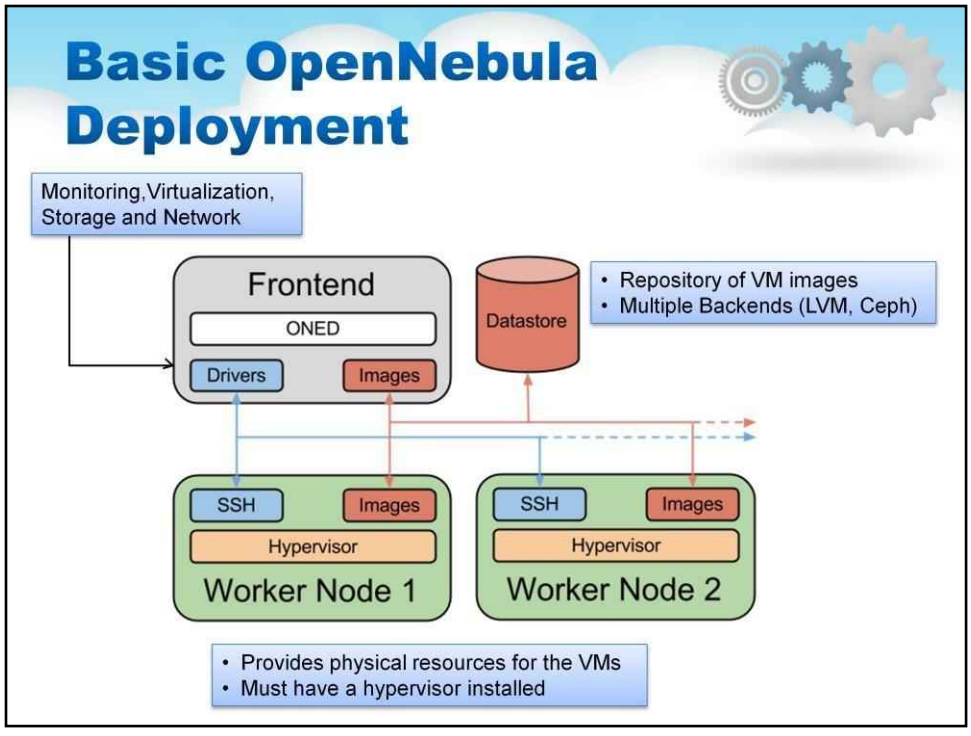


API віртуалізації libvirt

Його можна використовувати як інтерфейс для приватних хмарних обчислень. версія libvirt включає драйвер OpenNebula, який надає інтерфейс libvirt для розподіленої віртуальної інфраструктури, що складається з локальних ресурсів, на яких працює VMware, KVM або Xen; і хмарні ресурси на Amazon EC2 або ElasticHosts.

libvirt перетворюється на дуже багатий і широко використовуваний інтерфейс для керування можливостями віртуалізації сервера, включаючи віртуальну мережу, сховище та керування доменом. Отже, libvirt може бути дуже ефективним інтерфейсом адміністрування для приватної хмари, відкриваючи повний набір операцій віртуальної машини та фізичних вузлів. Таким чином libvirt

\+ OpenNebula надає потужну абстракцію для вашої приватної хмари

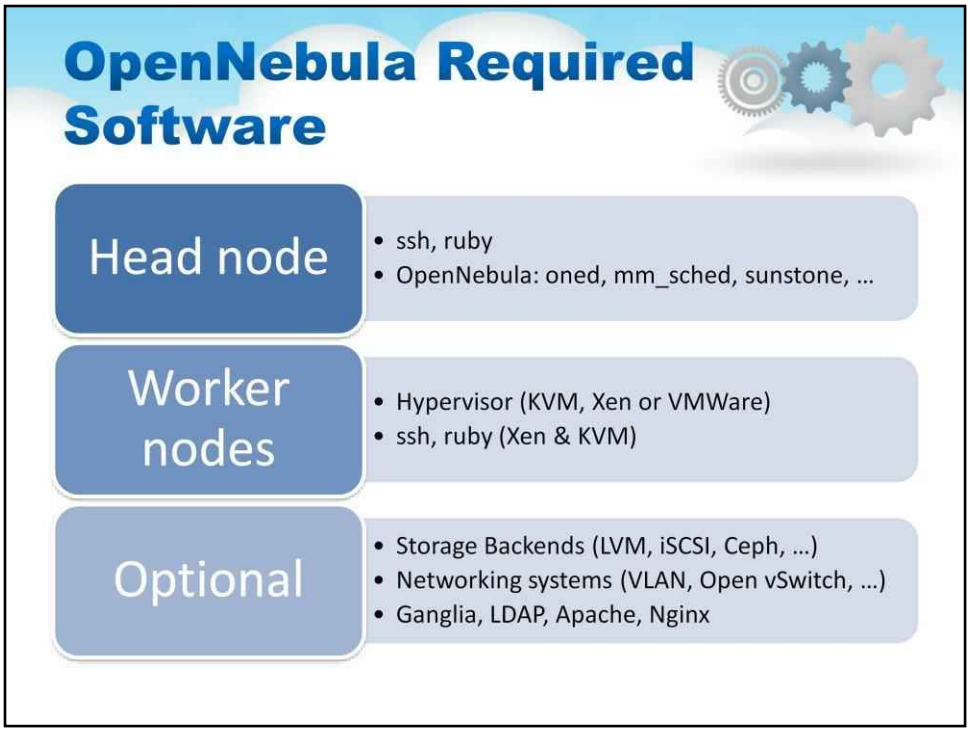


Розгортання OpenNebula

OpenNebula припускає, що ваша фізична інфраструктура використовує класичну кластерну архітектуру з інтерфейсом і набором хостів, де будуть виконуватися віртуальні машини (VM). Існує принаймні одна фізична мережа, яка об'єднує всі хости за допомогою інтерфейсу. Основними компонентами системи OpenNebula є:

- Інтерфейс, виконує служби OpenNebula.
- Хости, або Noes, хости з підтримкою гіпервізора, які надають ресурси, необхідні віртуальним машинам.
- Сховища даних містять базові образи віртуальних машин.
- Сервісна мережа, фізична мережа, яка використовується для підтримки основних служб: взаємозв'язку серверів зберігання та операцій керування OpenNebula
- Фізична мережа VM Networks, яка підтримуватиме VLAN для віртуальних машин.

Front-End.Машина, на якій встановлено OpenNebula, називається інтерфейсом. Ця машина повинна мати доступ до сховищ даних (наприклад, пряме монтування або мережу) і підключення до мережі для кожного хоста. Послуги OpenNebula включають: **Господарі.**Хости — це фізичні машини, на яких запускатимуться віртуальні машини. Під час інсталяції вам потрібно буде налаштувати обліковий запис адміністратора OpenNebula, щоб мати можливість SSH для хостів, і залежно від вашого гіпервізора вам доведеться дозволити цьому обліковому запису виконувати команди з привілеями root або зробити його частиною певної групи. **Зберігання.**OpenNebula використовує Datastores для обробки зображень дисків VM. Зображення VM зареєстровано або створено (порожні томи) у сховищі даних. Загалом, кожне сховище даних має бути доступним через інтерфейс за допомогою будь-якої відповідної технології.



Тут перелічено програмне забезпечення, необхідне для OpenNebula:

Головний вузол

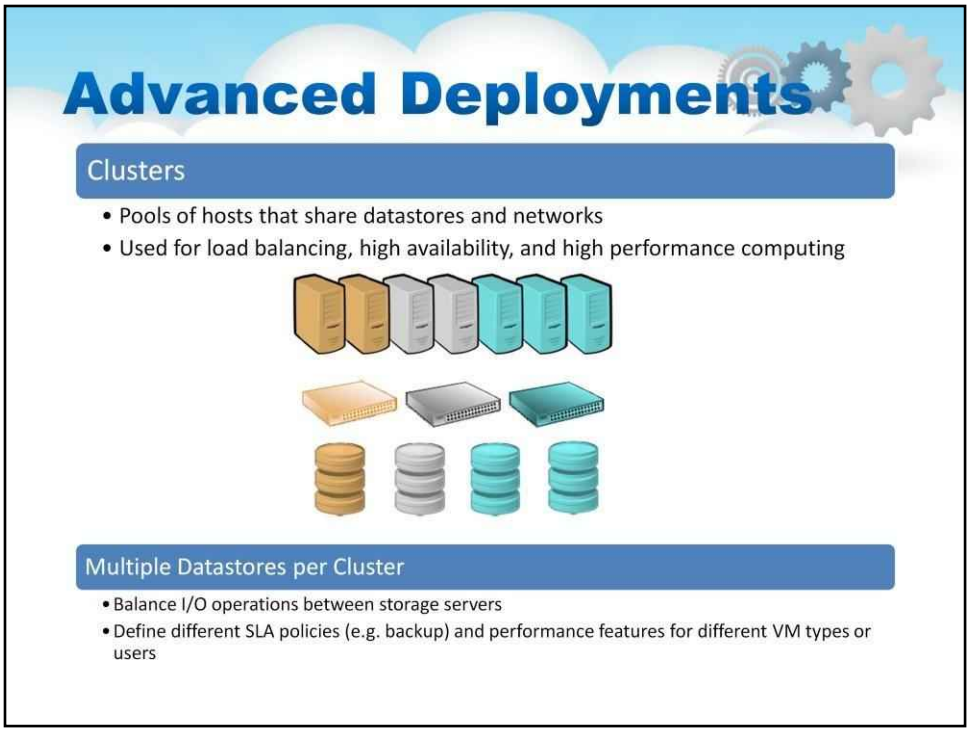
- ssh, рубін
- OpenNebula: oned, mm_sched, sunstone, ...

Робочі вузли

- Гіпервізор (наприклад, KVM, Xen або VMWare)
- ssh, ruby (для Xen і KVM)

Додатково

- Сервери зберігання (наприклад, LVM, iSCSI, Ceph, ...)
- Мережеві системи (наприклад, VLAN, Open vSwitch, ...)
- Ganglia, LDAP, Apache, Nginx

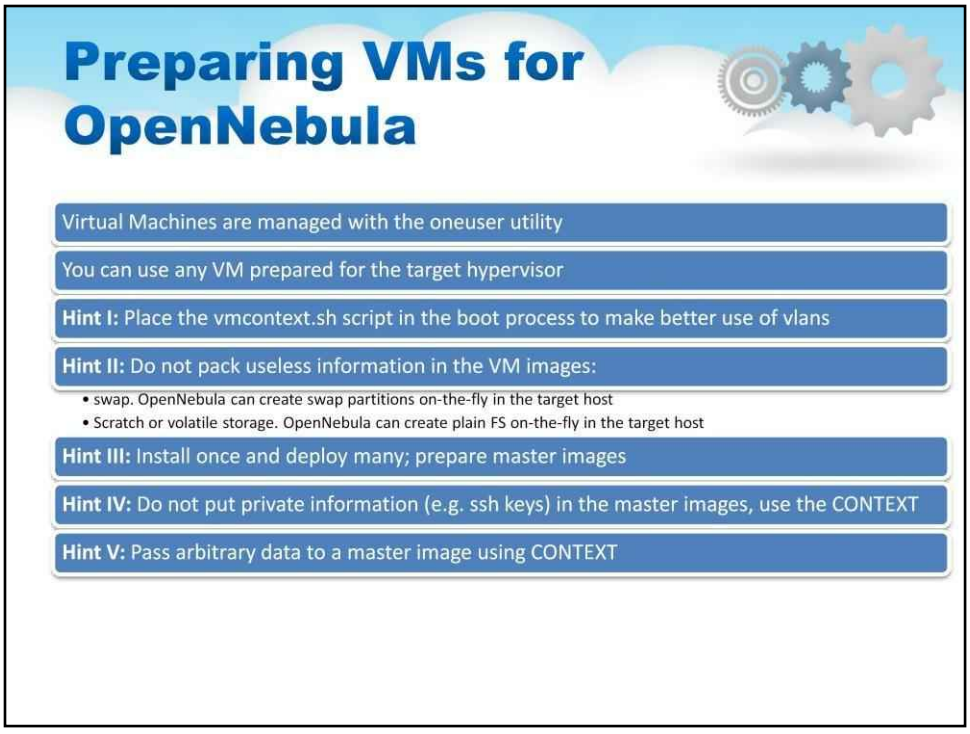


Кластери — це пули хостів, які спільно використовують сховища даних і віртуальні мережі. Кластери використовуються для балансування навантаження, високої доступності та високопродуктивних обчислень. Хости можуть бути згруповані в кластери.

Хости можуть бути створені безпосередньо в кластері. Хости можуть бути лише в одному кластері одночасно.

Сховища даних і віртуальні мережі можна додати до одного кластера. Це означає, що будь-який хост у цьому кластері належним чином налаштований для запуску віртуальних машин за допомогою зображень із сховищ даних або використовує оренду віртуальних мереж.
Наприклад, якщо у вас є кілька хостів, налаштованих на використання драйверів сховища даних iSCSI та мереж Open vSwitch, ви повинні згрупувати їх в одному кластері.

Системне сховище даних для кластера. Ви можете пов’язати певне системне сховище даних із кластером, щоб покращити його продуктивність (наприклад, збалансувати введення/виведення віртуальної машини між різними серверами) або використовувати різні типи системного сховища даних (наприклад, спільне та ssh). Щоб використовувати з кластером певне системне сховище даних замість стандартного, просто створіть його та пов’яжіть, як і будь-яке інше сховище даних.



Система зберігання дозволяє адміністраторам і користувачам OpenNebula налаштовувати зображення, які можуть бути оперативними системами або даними, для легкого використання у віртуальних машинах. Ці зображення можуть використовуватися кількома віртуальними машинами одночасно, а також надаватися іншим користувачам.

Існують різні види зображень:

ОС: образ ОС містить робочу операційну систему. Кожен шаблон віртуальної машини повинен визначати один ДИСК, який посилається на образ цього типу.

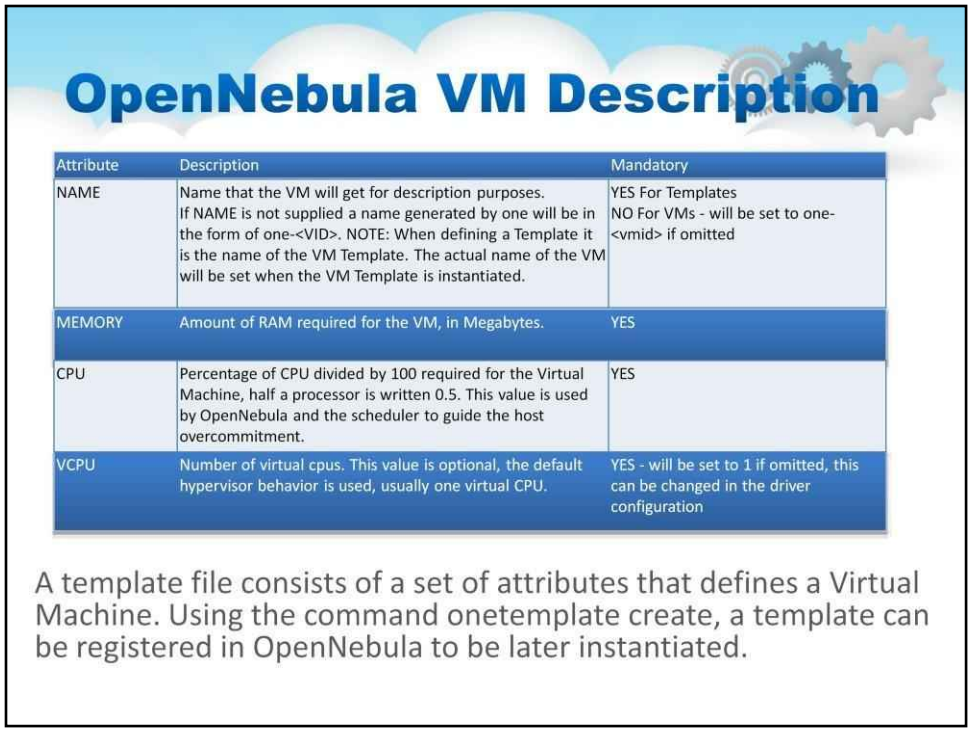
CD-ROM: ці зображення призначені лише для читання. У кожному шаблоні віртуальної машини можна використовувати лише одне зображення цього типу.

БЛОК ДАНИХ: образ блоку даних — це сховище для даних, до яких можна отримати доступ і змінити їх з різних віртуальних машин. Ці зображення можна створити з попередніх існуючих даних або як порожній диск.

ЯДРО: звичайний файл для використання як ядро. Зауважте, що образи файлів KERNEL можна зареєструвати лише у сховищах файлів.

RAMDISK: звичайний файл для використання як ramdisk. Зауважте, що образи файлів RAMDISK можна зареєструвати лише у сховищах файлів.

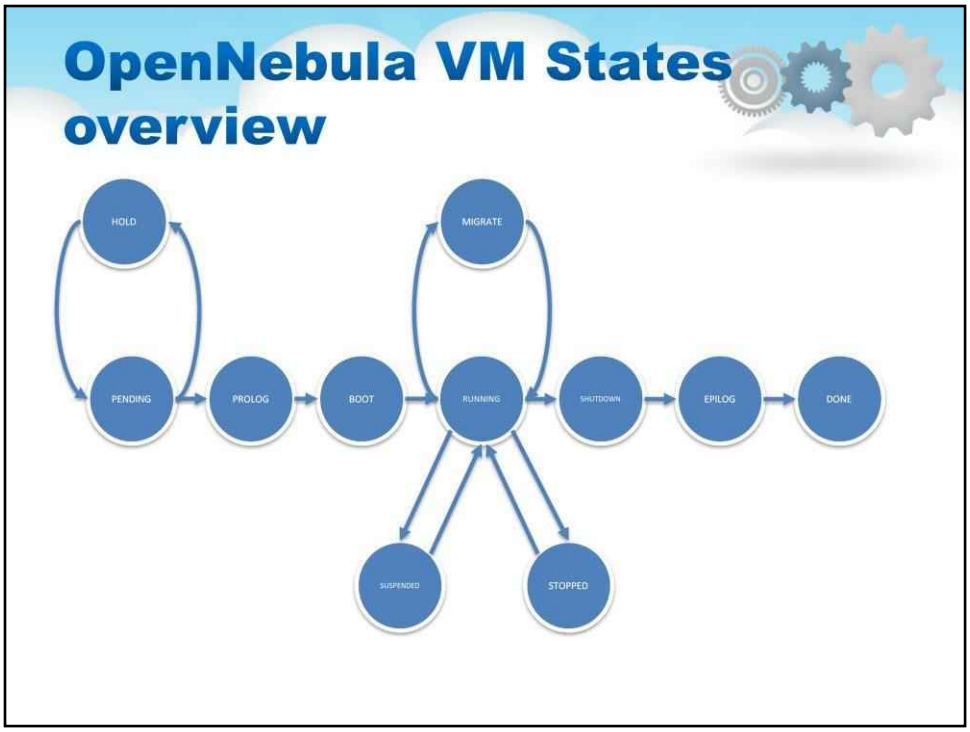
CONTEXT: звичайний файл, який буде включено до контекстного компакт-диска. Зауважте, що зображення файлів CONTEXT можна зареєструвати лише у сховищах даних файлів.



Цей слайд містить пояснення до опису OpenNebula Virtula Machine.

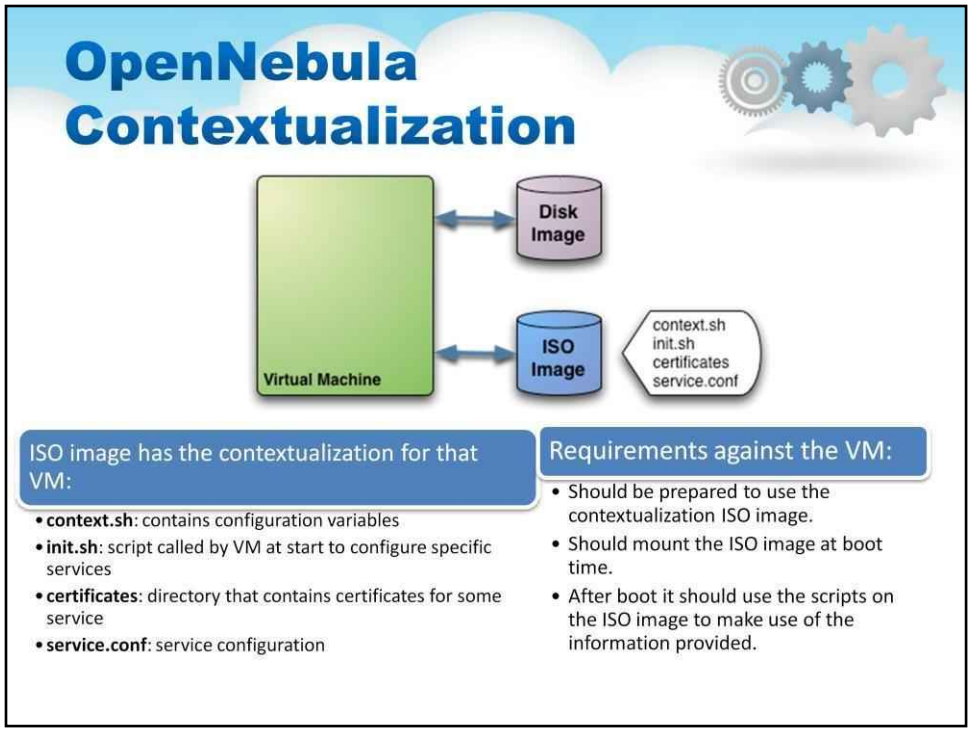
Файл шаблону складається з набору атрибутів, які визначають віртуальну машину.

Для сумісності з попередніми версіями ви також можете створити нову віртуальну машину безпосередньо з файлу шаблону.



Одним із найскладніших аспектів ефективного використання будь-якої інфраструктури хмарних обчислень є необхідність адекватного розуміння життєвого циклу, який проходитиме керована хмарою віртуальна машина.

Це принципова схема такого життєвого циклу.



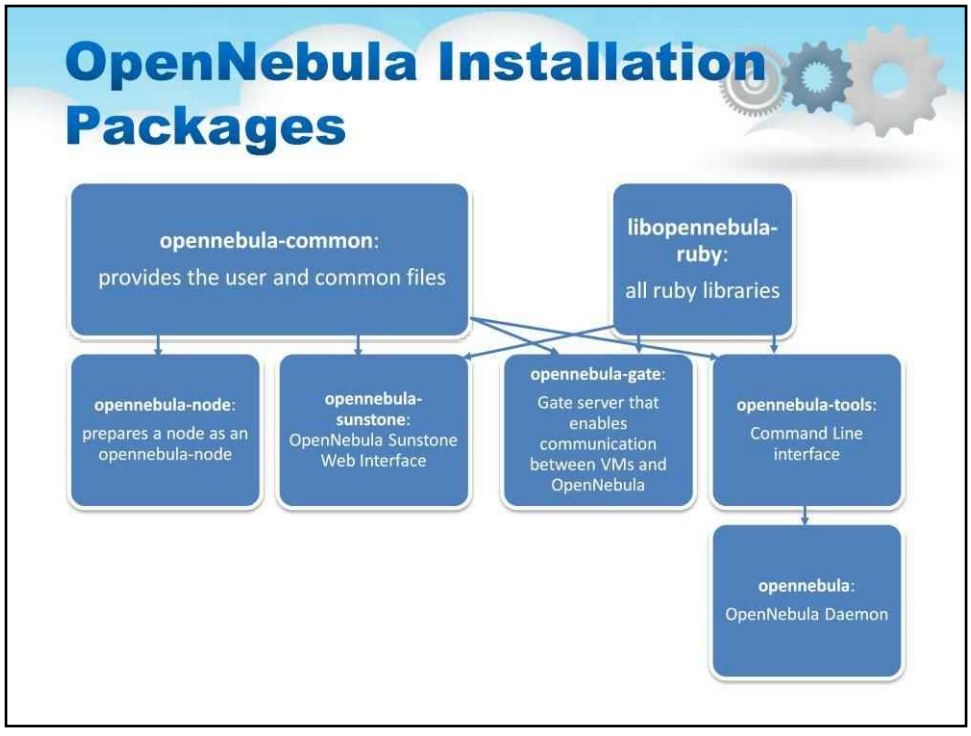
У OpenNebula доступні два механізми контекстуалізації:

- автоматичне призначення IP-адреси та
- більш загальний спосіб надання будь-яких файлів і параметрів конфігурації.

Автоматичне призначення IP. За допомогою OpenNebula ви можете отримати IP-адресу, призначену віртуальній машині, з MAC-адреси за допомогою правила MAC_PREFFIX:IP. Для цього існують контекстні сценарії для систем на базі Debian, Ubuntu, CentOS і openSUSE.

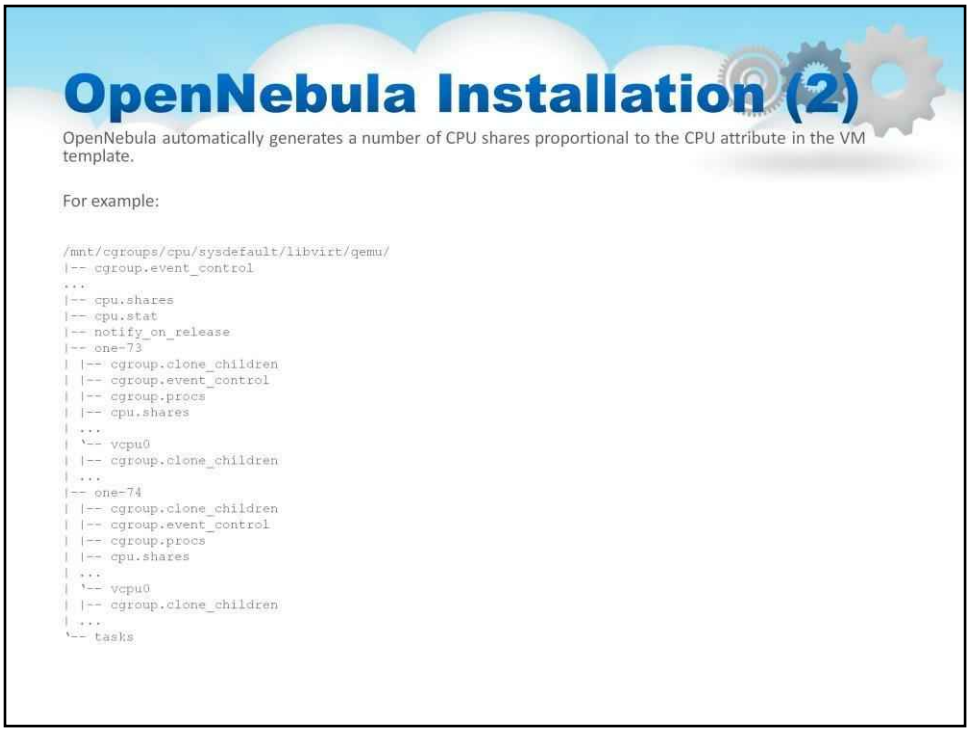
Загальна контекстуалізація. Метод надається для надання параметрів конфігурації нещодавно запущеній віртуальній машині за допомогою образу ISO (рекомендація OVF). Цей метод не залежить від мережі, тому його можна також використовувати для налаштування мережевих інтерфейсів. У файлі опису віртуальної машини ви можете вказати вміст файлу iso (файли та каталоги), повідомити пристрою, що образ ISO буде доступним, і вказати параметри конфігурації, які будуть записані у файл для подальшого використання у віртуальній машині.

У цьому прикладі ми бачимо віртуальну машину з двома асоційованими дисками. Образ диска містить файлову систему, з якої запускатиметься операційна система. Образ ISO має контекстуалізацію для цієї віртуальної машини.



Інсталяційні пакети OpenNebula містять такі частини:

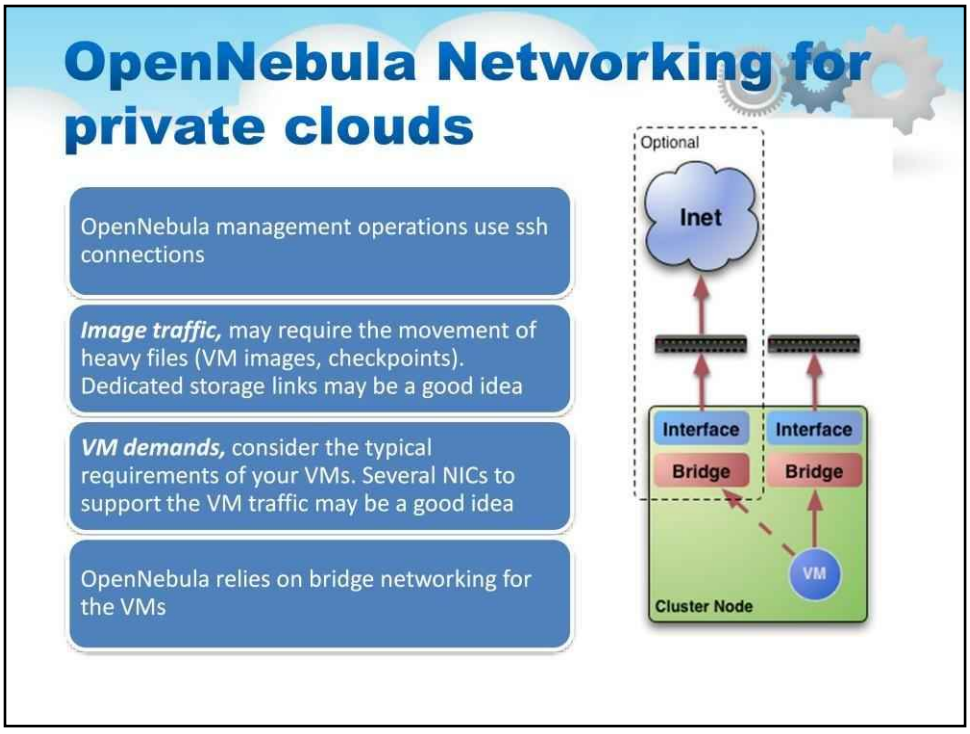
opennebula-common: надає файли користувача та спільні файли
libopennebula-ruby: усі бібліотеки ruby
opennebula-node: готує вузол як opennebula-node
opennebula-sunstone: веб-інтерфейс OpenNebula Sunstone
opennebula-tools: інтерфейс командного рядка
opennebula-gate: сервер Gate, який забезпечує зв’язок між віртуальними машинами та OpenNebula
opennebula-flow: Керує службами та еластичністю
opennebula: OpenNebula Daemon



OpenNebula автоматично генерує кількість ресурсів ЦП, пропорційну атрибуту ЦП у шаблоні віртуальної машини.

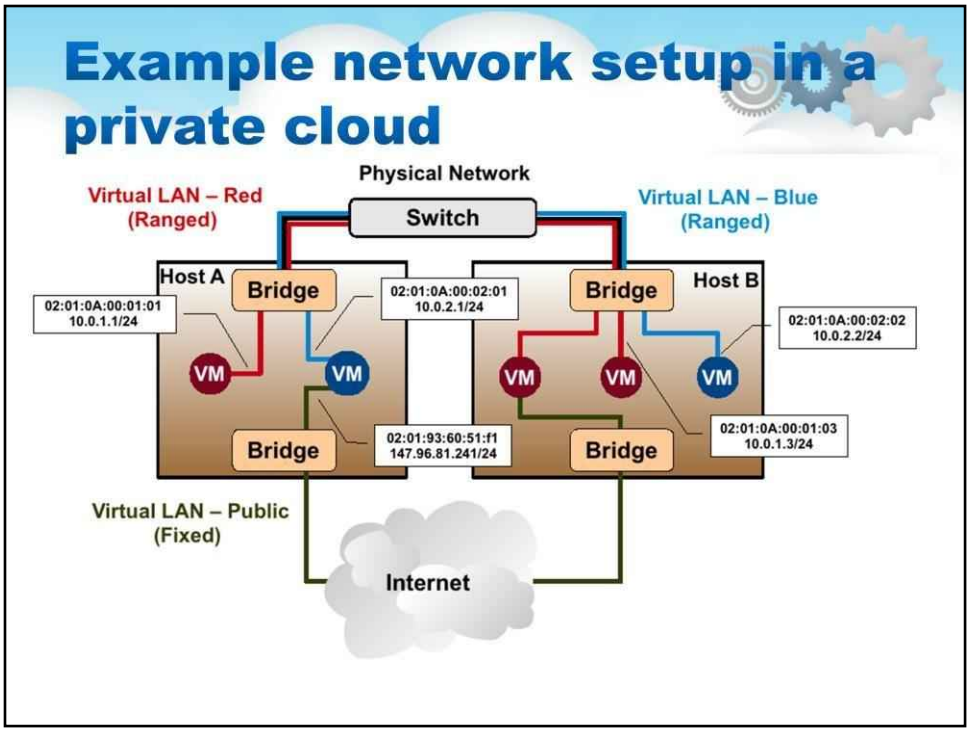
Наприклад, розглянемо хост, на якому запущено 2 віртуальні машини (73 і 74) з різними частками ЦП для них.

Якщо все правильно налаштовано, ви повинні побачити:



Коли запускається нова віртуальна машина, OpenNebula з’єднає свої мережеві інтерфейси (визначені в розділі NIC шаблону) до мосту або фізичного пристрою, указаного у визначенні віртуальної мережі. Це дозволить віртуальній машині мати доступ до різних мереж, публічних або приватних. Ця функція надається через драйвери Virtual Network Manager.

Щоб ефективно використовувати розгортання віртуальних машин, вам, імовірно, потрібно буде зробити одну або кілька фізичних мереж доступними для них. Наприклад, типовий хост із двома фізичними мережами, одна для загальнодоступних IP-адрес (приєднана до NIC eth0), а інша для приватних віртуальних локальних мереж (NIC eth1), повинен мати два мости (як показано на ілюстрації).



Адміністратор OpenNebula може пов’язати один із таких драйверів з кожним хостом:

манекен:Стандартний драйвер, який не виконує жодних мережевих операцій. Правила брандмауера також ігноруються.

fw:Правила брандмауера застосовуються, але ізоляція мережі ігнорується.

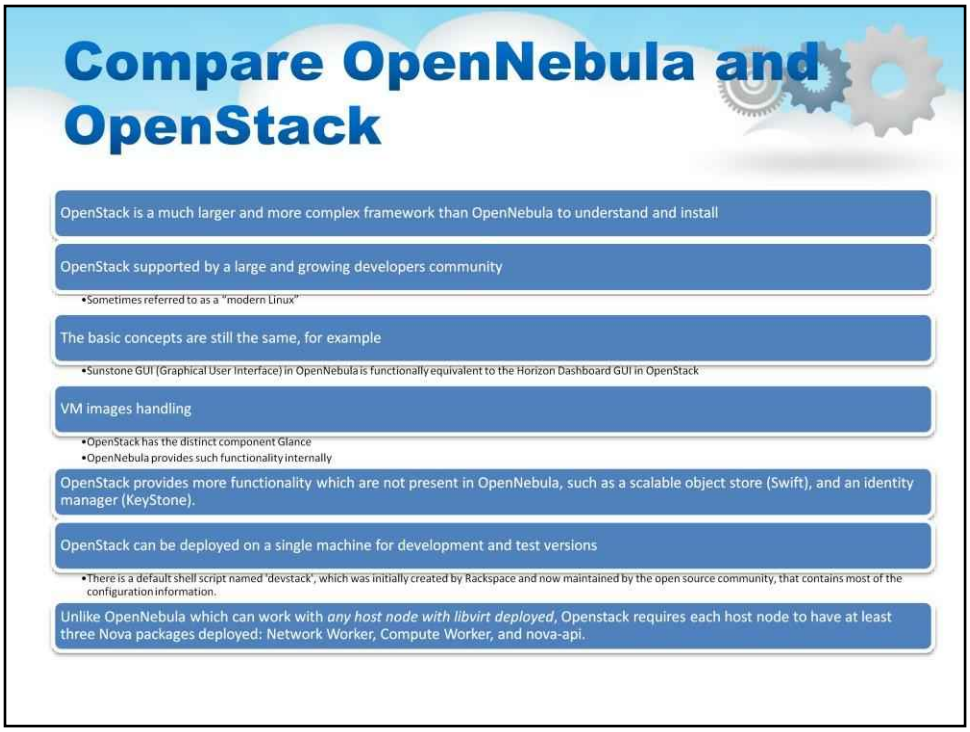
802.1Q:обмежити доступ до мережі через тегування VLAN, для чого також потрібна підтримка апаратних комутаторів.

ebtables:обмежити доступ до мережі за допомогою правил Ebtables. Спеціальна апаратна конфігурація не потрібна.

ovswitch:обмежте доступ до мережі за допомогою Open vSwitch Virtual Switch.

VMware:використовує мережеву інфраструктуру VMware для забезпечення ізольованої та сумісної з 802.1Q мережі для віртуальних машин, запущених за допомогою гіпервізора VMware. Зверніть увагу, що деякі з цих драйверів також створюють мостовий пристрій у хостах.

Мережа потрібна зовнішнім демонам OpenNebula для доступу до хостів для керування та моніторингу гіпервізорів; і перемістити файли зображень. Для цього настійно рекомендується встановити спеціальну мережу. Щоб запропонувати мережеве підключення до віртуальних машин на різних хостах, конфігурація за замовчуванням підключає мережевий інтерфейс віртуальної машини до мосту на фізичному хості.



Тепер давайте порівняємо OpenNebula і OpenStack, які будуть розглянуті в наступному розділі (і залишено для самостійної роботи).

OpenStack — це набагато більший і складніший фреймворк, ніж OpenNebula для розуміння та встановлення.

OpenStack підтримується великою та зростаючою спільнотою розробників

Іноді його називають «сучасним Linux»

Наприклад, основні поняття залишаються незмінними

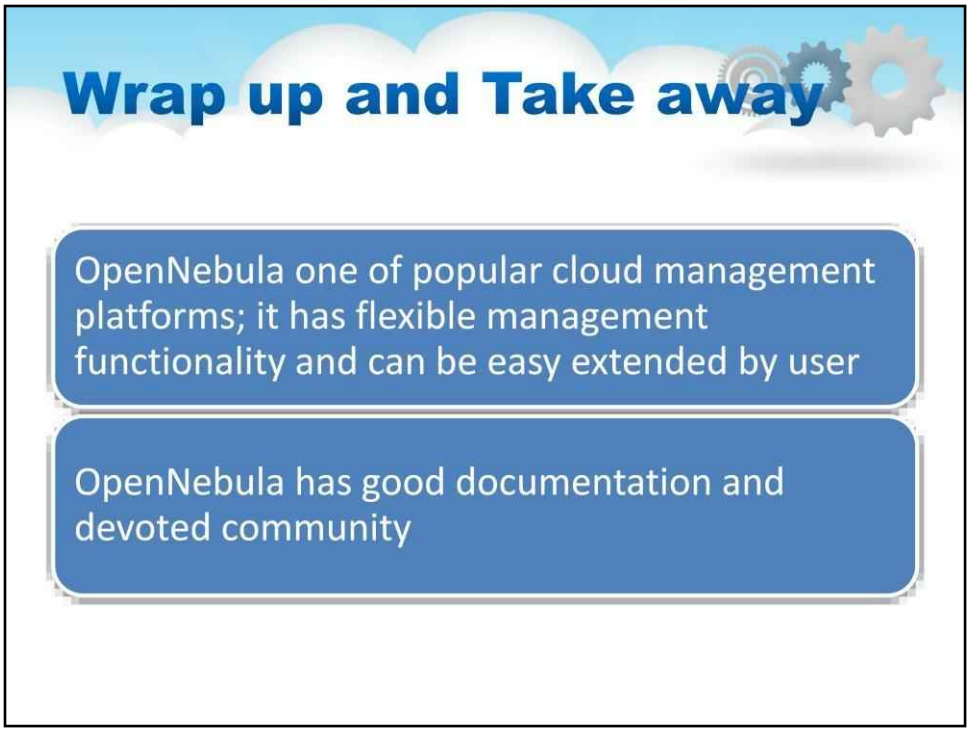
Sunstone GUI (графічний інтерфейс користувача) в OpenNebula функціонально еквівалентний графічному інтерфейсу Horizon Dashboard в OpenStack

Обробка зображень VM

OpenStack має окремий компонент Glance. OpenNebula забезпечує таку функціональність усередині

OpenStack надає більше функцій, яких немає в OpenNebula, наприклад, масштабоване сховище об’єктів (Swift) і менеджер ідентичності (KeyStone).

OpenStack можна розгорнути на одній машині для розробки та тестування версій



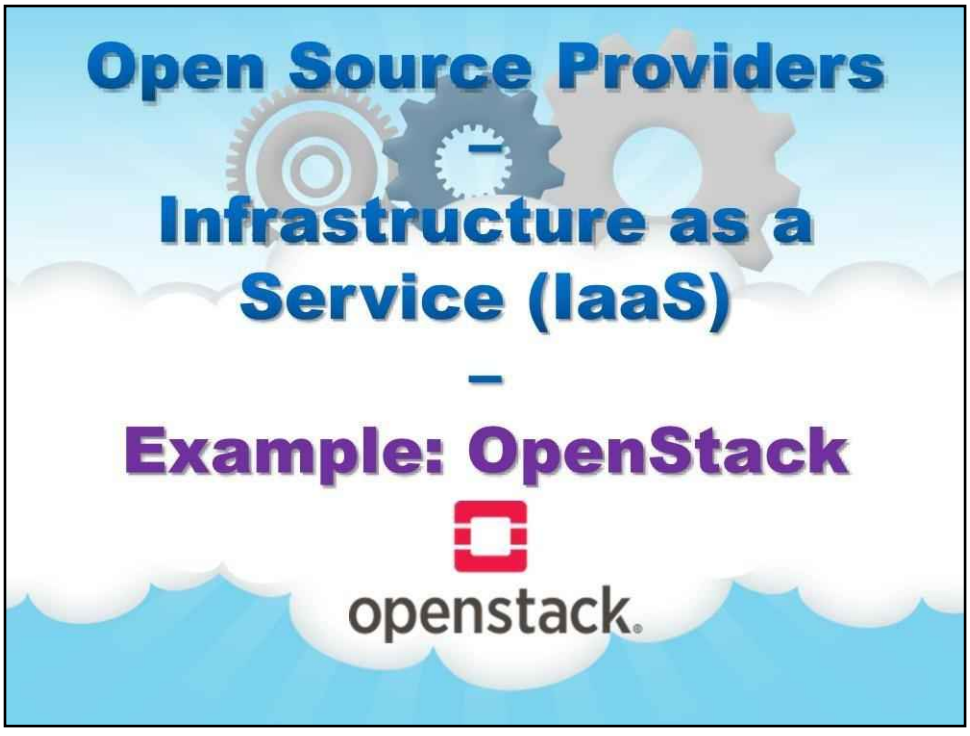
Стосовно OpenNebula слід зробити такі останні примітки:

-OpenNebula є однією з популярних хмарних платформ керування-

-OpenNebula має гнучкі функції керування-

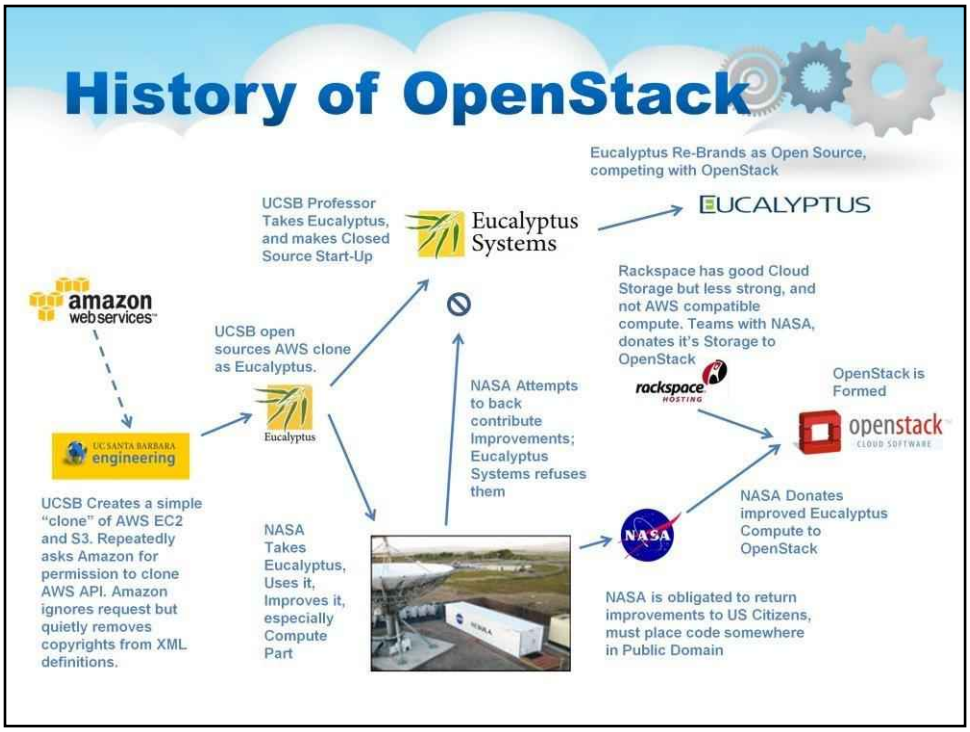
-OpenNebula може бути легко розширена користувачем-

-OpenNebula має гарну документацію та віддану спільноту-



Приклад: OpenStack

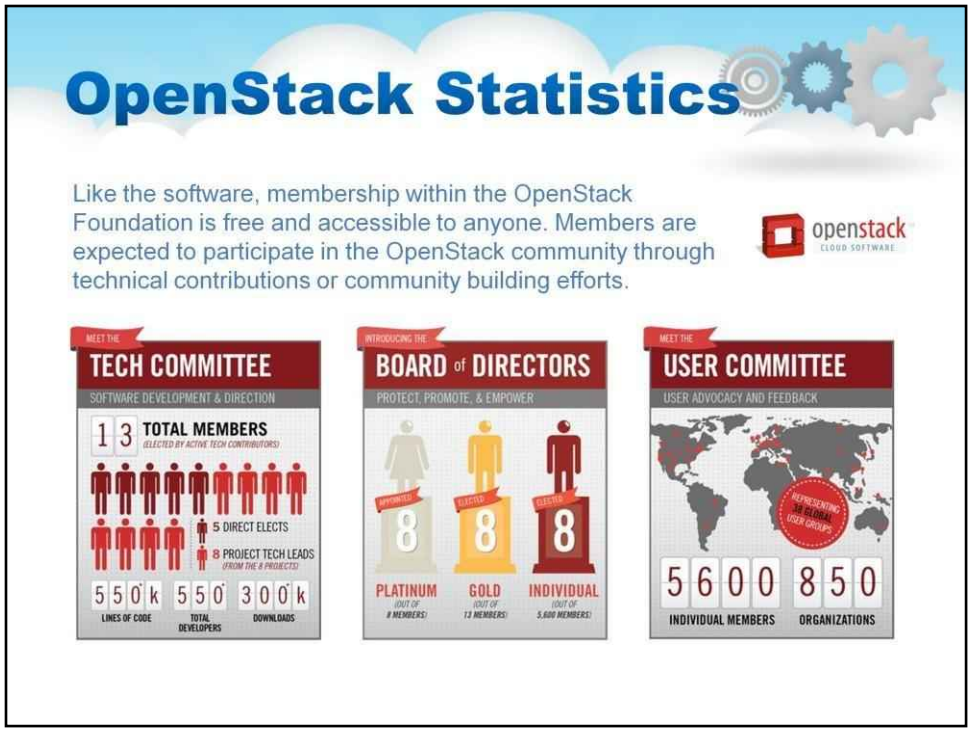
Цей розділ і наступні залишено для самостійної роботи.



Як показано на ілюстрації, OpenStack з'явився в дуже обхідний спосіб. Насправді це маловідома історія.

Натхненням для створення OpenStack став Каліфорнійський університет Санта-Барбара Евкаліптовий проект, над яким NASA провело значну роботу. Однак NASA не змогло повернути свої зміни до проекту Eucalyptus, оскільки він уже став допоміжною компанією та дотримується стратегії «закритого коду». Оскільки за законом NASA зобов'язувалося якось зробити доступними вдосконалення (вони належали громадянам Сполучених Штатів, оскільки NASA фінансується за гроші платників податків), коли Rackspace звернулися з проханням розмістити репозиторій, вони вирішили додати свій код як ну і створіть OpenStack, яким ми його знаємо сьогодні.

У липні 2010 року Rackspace Hosting і NASA спільно запустили ініціативу хмарного програмного забезпечення з відкритим кодом, відому як OpenStack. Проект OpenStack покликаний допомогти організаціям, які пропонують послуги хмарних обчислень, що працюють на стандартному обладнанні. Перший офіційний реліз під кодовою назвою Austin з'явився через чотири місяці з планами випускати регулярні оновлення програмного забезпечення кожні кілька місяців. Перший код надійшов з платформи NASA Nebula та платформи Rackspace Cloud Files. У липні 2011 року розробники Ubuntu Linux прийняли OpenStack.



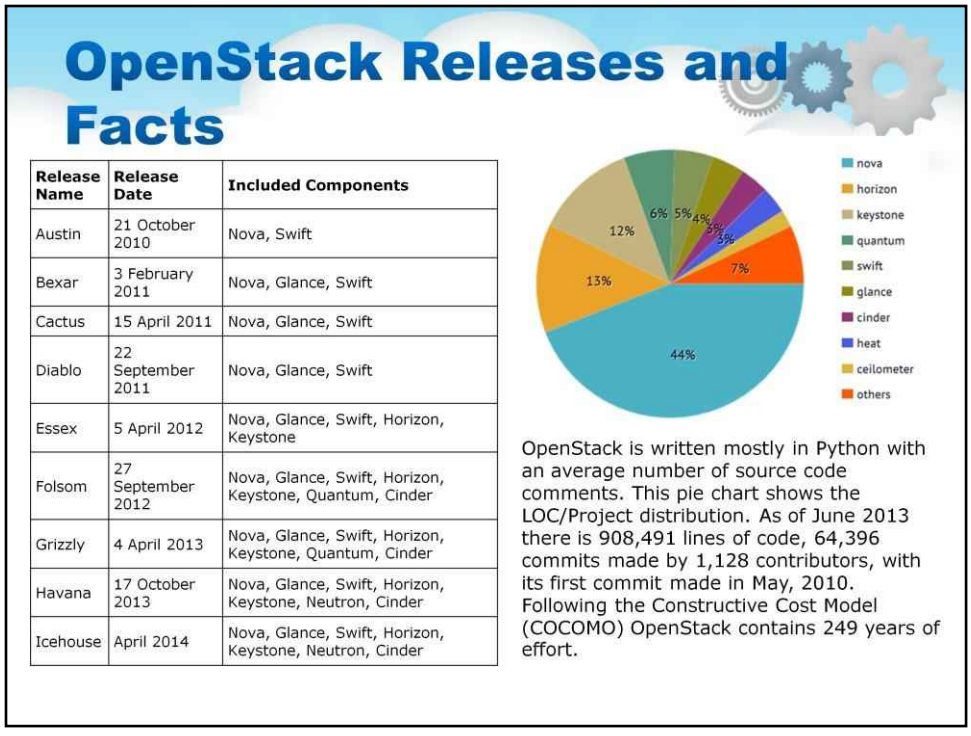
OpenStack Foundation, створена у вересні 2012 року, є незалежною організацією, яка надає спільні ресурси для досягнення місії OpenStack шляхом захисту, розширення можливостей і просування програмного забезпечення OpenStack і спільноти навколо нього. Це стосується користувачів, розробників і всієї екосистеми.

OpenStack, заснований Rackspace Hosting і NASA, перетворився на глобальну спільноту розробників програмного забезпечення, які співпрацюють над стандартною хмарною операційною системою з відкритим вихідним кодом, яка широко масштабується.

OpenStack Foundation сприяє розробці, розповсюдженню та прийняттю хмарної операційної системи OpenStack. Будучи незалежною домівкою для OpenStack, Фонд уже залучив понад 7000 окремих членів із 100 країн і 850 різних організацій. Він також забезпечив понад 10 мільйонів доларів фінансування.

Весь код для OpenStack є у вільному доступі за ліцензією Apache 2.0.

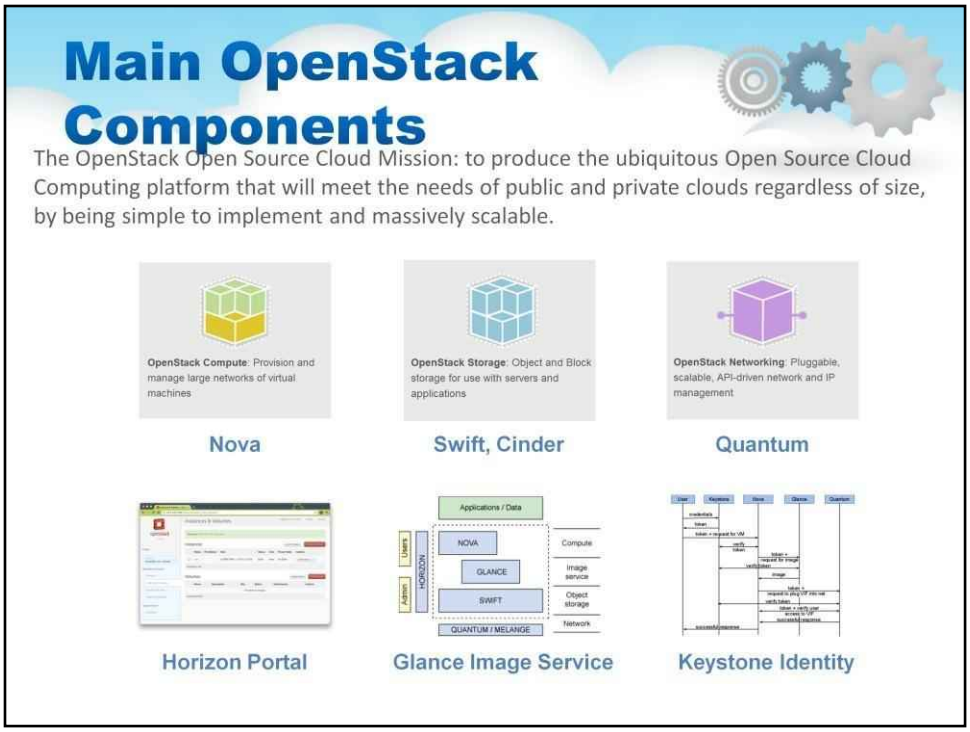
Деякі користувачі OpenStack включають: PayPal / eBay, NASA, CERN, Yahoo! Rackspace Cloud HP Public Cloud, Mercado, Libre.com, AT&T, KT (раніше Korea Telecom), Deutsche Telekom, Wikimedia Labs, Hostalia of Telef nica Group, рішення SUSE Cloud, рішення Red Hat OpenShift PaaS Zadara Storage тощо.



OpenStack базується на скоординованому 6-місячному циклі випуску з частими етапами розробки.

Створення OpenStack зайняло приблизно 249 років зусиль (модель COCOMO).

У двох словах, OpenStack має:
64 396 комітів, зроблених 1128 учасниками, з першим комітом, зробленим у травні 2010 року.
908 491 рядок коду. OpenStack написаний переважно на Python із середньою кількістю коментарів до вихідного коду.
Кодова база з довгою історією джерел. Збільшення
зобов’язань у порівнянні з минулим роком.
Дуже велика команда розробників, що складається з людей з усього світу.



OpenStack має модульну архітектуру з різними кодовими назвами для своїх компонентів. OpenStack має кілька спільних служб, які охоплюють три стовпи обчислення, зберігання та мережі, що полегшує впровадження та керування вашою хмарою. Ці служби, включаючи ідентифікацію, керування зображеннями та веб-інтерфейс, інтегрують компоненти OpenStack один з одним, а також із зовнішніми системами, щоб забезпечити уніфікований досвід для користувачів під час взаємодії з різними хмарними ресурсами.

Це

Nova: OpenStack Compute: створення та керування великими мережами віртуальних машин

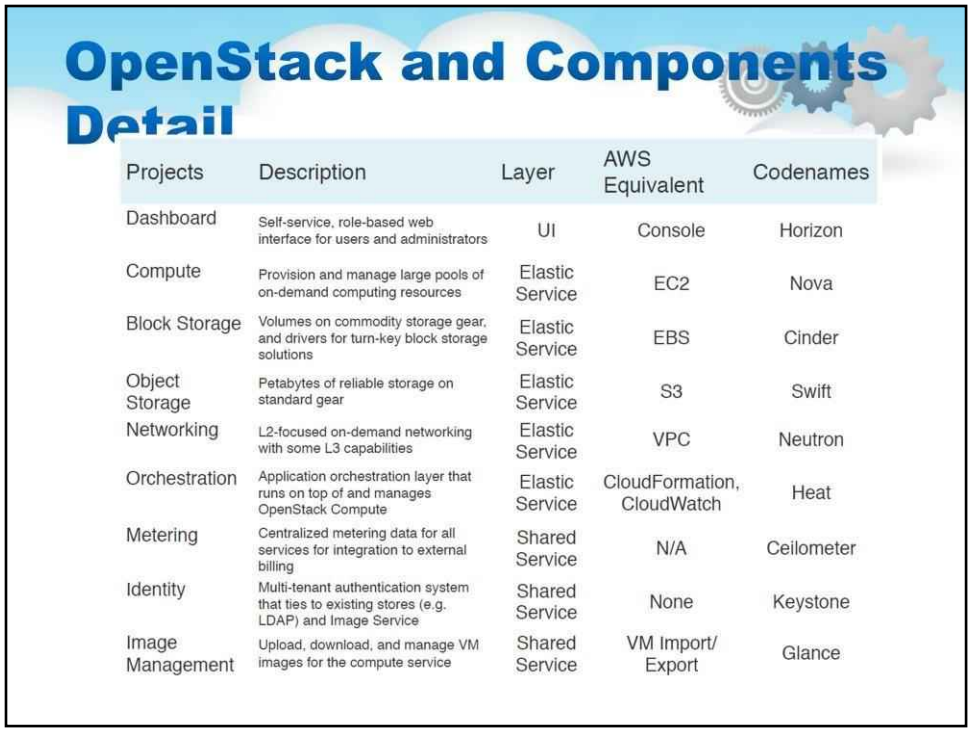
Swift і Cinder: сховище OpenStack: сховище об’єктів і блоків для використання з серверами та програмами

Neutron: Мережа OpenStack: підключається, масштабована мережа та IP-керування на основі API

Горизонт: інформаційна панель

Keystone: Послуги ідентифікації

Glance: Image Services



Панель приладів (Горизонт)надає адміністраторам і користувачам графічний інтерфейс для доступу, наданн та автоматизації хмарних ресурсів.

Обчислення (Нова). Хмарна операційна система OpenStack дозволяє підприємствам і постачальникам послуг пропонувати обчислювальні ресурси на вимогу, забезпечуючи і керуючи великими мережами віртуальних машин. Обчислювальні ресурси доступні через API для розробників, що створюють хмарні програми, і через веб-інтерфейси для адміністраторів і користувачів. Обчислювальна архітектура розроблена для горизонтального масштабування на стандартному обладнанні.

Блок зберігання (Cinder)забезпечує постійні пристрої зберігання на рівні блоку для використання з екземплярами обчислень OpenStack. Блокова система зберігання керує створенням, підключенням і від'єднанням блокових пристроїв до серверів.

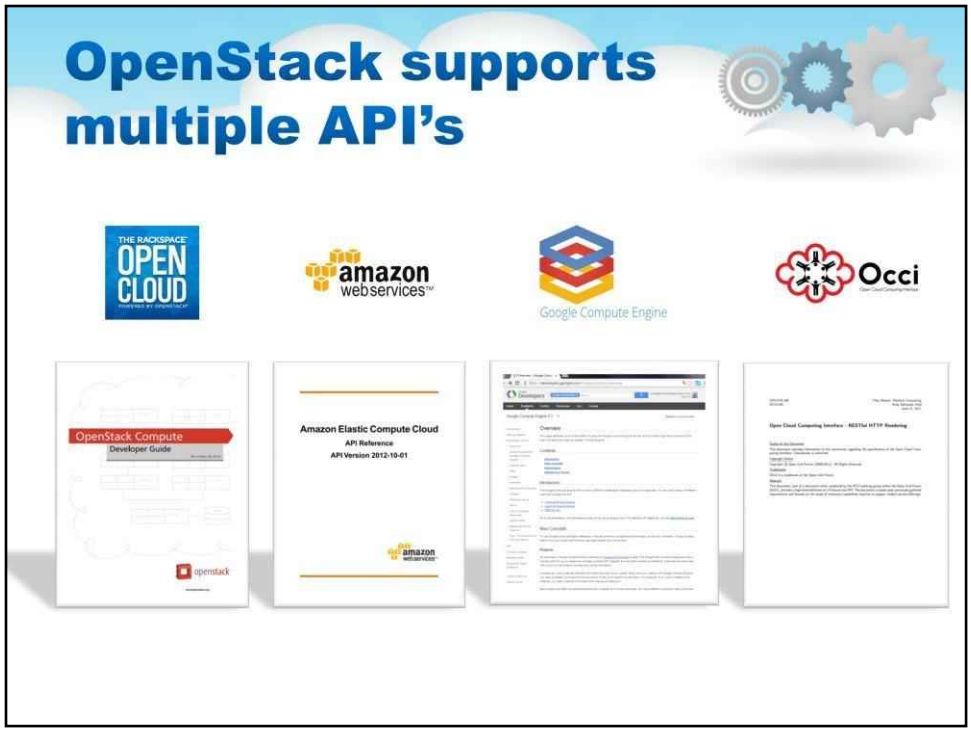
Об'єктне сховище (Swift).На додаток до традиційної технології зберігання корпоративного класу, багато організацій тепер мають різноманітні потреби в сховищах із різними вимогами до продуктивності та ціни. OpenStack підтримує як Object Storage, так і Block Storage, з багатьма варіантами розгортання для кожного залежно від варіанту використання.

Мережа (Нейтрон).Сучасні мережі центрів обробки даних містять більше пристроїв, ніж будь-коли раніше. Від серверів, мережевого обладнання, систем зберігання та пристроїв безпеки, багато з яких далі поділяються на віртуальні машини та віртуальні мережі. Кількість IP-адрес, конфігурацій маршрутизації та правил безпеки мож швидко зрости до мільйонів. Традиційні методи керування мережами не можуть забезпечити дійсно масштабований, автоматизований підхід до керування цими мережами нового покоління. У той же час користувачі очікують більшого контролю та гнучкості завдяки швидшій ініціалізації.

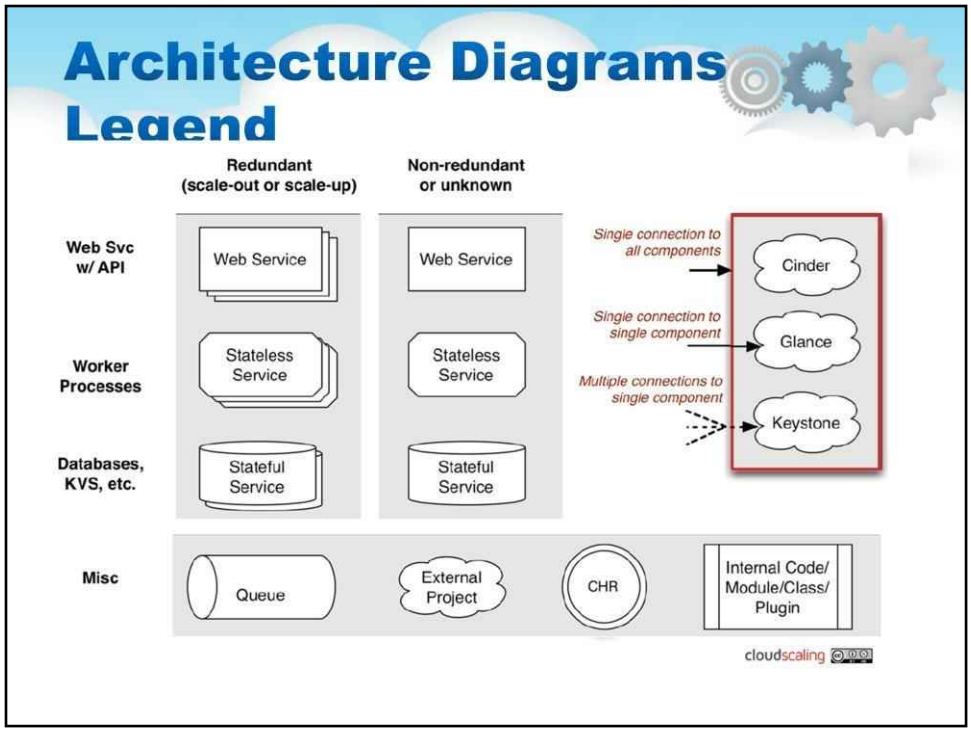
Служба ідентифікації (Keystone)надає центральний каталог користувачів, зіставлених із службами OpenStack, до яких вони мають доступ. Він діє як звичайна система автентифікації в хмарній операційній системі та може інтегруватися з існуючими серверними службами каталогів, такими як LDAP.

Служба зображень (Погляд)

Служба зображень (Погляд)надає послуги виявлення, реєстрації та доставки образів дисків і серверів.

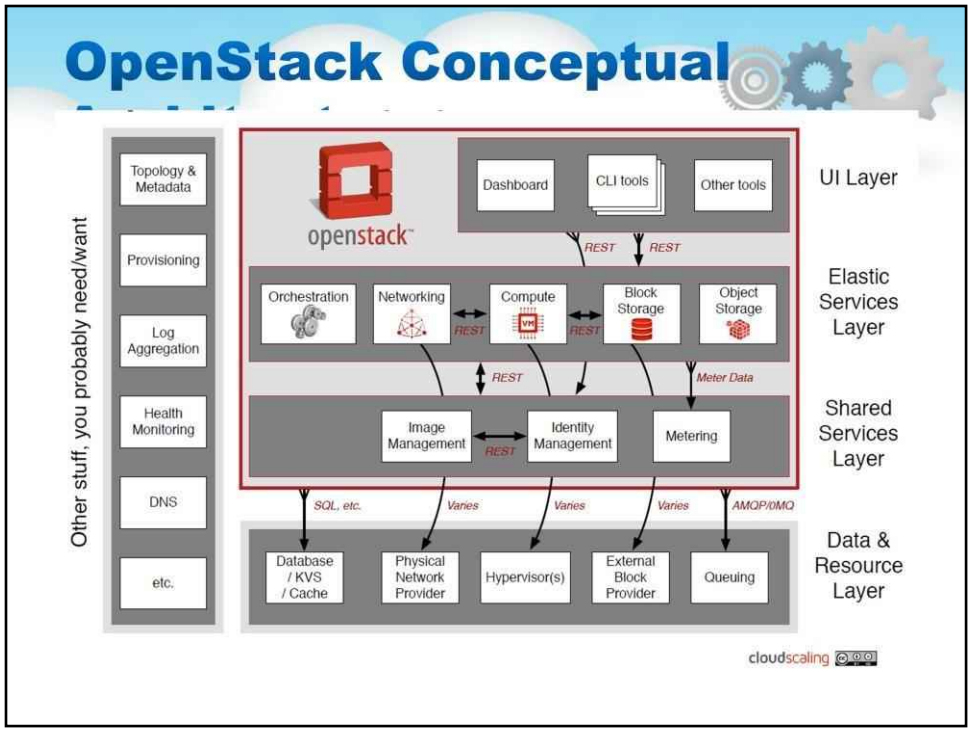


API OpenStack сумісні з Amazon EC2 і Amazon S3, тому клієнтські програми, написані для Amazon Web Services, можна використовувати з OpenStack з мінімальними зусиллями щодо портування.



Наступні слайди містять архітектурні діаграми внутрішніх елементів OpenStack.

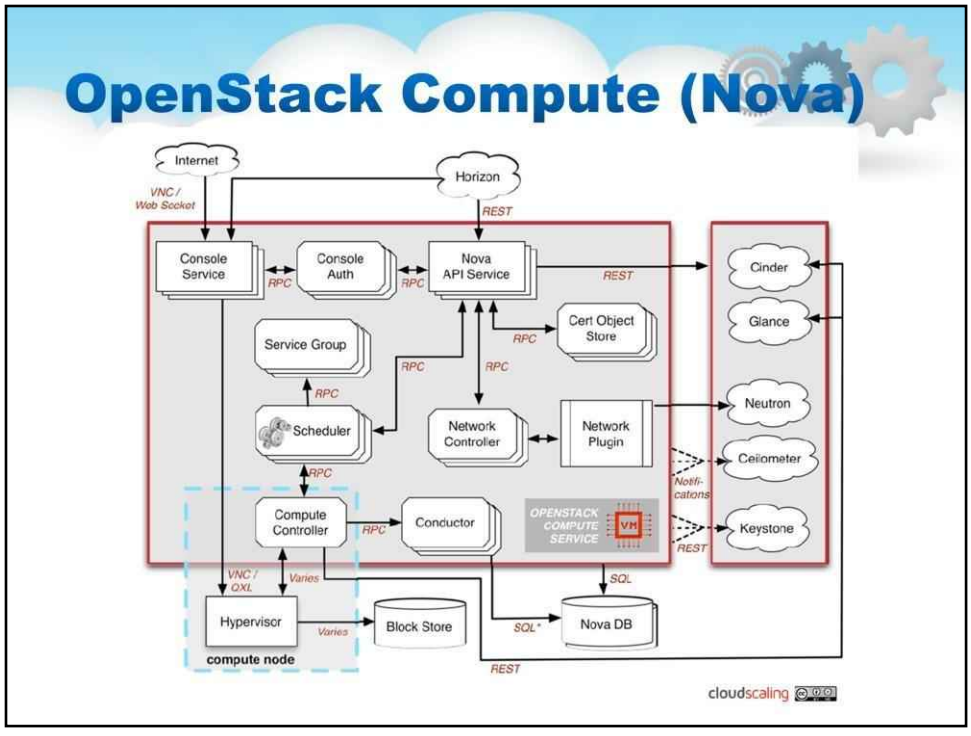
Для позначення типу елемента використовуються дуже специфічні умовні позначення. Будь ласка, зверніться до цієї схеми під час вивчення наступних діаграм.



Концептуальна архітектура

Проект OpenStack загалом розроблено для створення широкомасштабованої хмарної операційної системи. Щоб досягти цього, кожна зі складових служб розроблена для спільної роботи, щоб забезпечити повну інфраструктуру як послугу (IaaS). Цю інтеграцію сприяють загальнодоступні інтерфейси прикладного програмування (API), які пропонує кожна служба (і, у свою чергу, може використовувати). Хоча ці API дозволяють кожній із служб використовувати іншу службу, це також дозволяє розробнику вимикати будь-яку службу, якщо вони підтримують API. Це (переважно) ті самі API, які доступні кінцевим користувачам хмари.

Dashboard ("Horizon") забезпечує веб-інтерфейс для інших служб OpenStack Compute ("Nova"), які зберігають і отримують віртуальні диски ("зображення") і пов'язані метадані в Image ("Glance")
Мережа («Neutron») забезпечує віртуальну мережу для Compute.
Block Storage («Cinder») надає обсяги сховища для Compute.
Зображення («Glance») може зберігати фактичні файли віртуального диска в Object Store («Swift»).
Усі служби автентифікуються за допомогою Identity («Keystone»)
Кінцеві користувачі можуть взаємодіяти через загальний веб-інтерфейс (Horizon) або безпосередньо до кожної служби через їхній API
Усі служби автентифікуються через спільне джерело (за допомогою трапецієвого спотворення) Окремі служби взаємодіють одна з одною через свої загальнодоступні API (за винятком випадків, коли потрібні привілейовані команди адміністратора)



OpenStack Compute (Нова)це контролер тканини хмарних обчислень (основна частина системи IaaS). Він написаний на Python і використовує багато зовнішніх бібліотек, таких як Eventlet (для паралельного програмування), Kombu (для зв'язку AMQP) і SQLAlchemy (для доступу до бази даних). Архітектура Nova розроблена для горизонтального масштабування на стандартному апаратному забезпеченні без власних вимог до апаратного чи програмного забезпечення та забезпечує можливість інтеграції із застарілими системами та сторонніми технологіями. Він призначений для керування та автоматизації пулів комп'ютерних ресурсів і може працювати з широко доступними технологіями віртуалізації, а також із конфігураціями «голого металу» та високопродуктивних обчислень (HPC). KVM і XenServer є доступними варіантами для технології гіпервізора разом із технологією контейнерів Hyper-V і Linux, такою як LXC. Крім різних гіпервізорів,

Популярні випадки використання:

Постачальники послуг, що пропонують обчислювальну платформу IaaS або послуги вище стеку IT-відділи, що діють як постачальники хмарних послуг для бізнес-підрозділів і проектних груп Обробка великих даних за допомогою таких інструментів, як Hadoop

Масштабування обчислень угору та вниз для задоволення попиту на веб-ресурси та програми. Високопродуктивні обчислювальні середовища (HPC) обробляють різноманітні та інтенсивні робочі навантаження

Nova є найскладнішим і поширеним компонентом OpenStack. Велика кількість процесів співпрацює, щоб перетворити запити API кінцевого користувача на запущені віртуальні машини. Нижче наведено список цих процесів і їхні функції:

nova-api приймає та відповідає на виклики обчислювального API кінцевого користувача. Він підтримує API OpenStack Compute, API Amazon EC2 і спеціальний API адміністратора (для привілейованих користувачів для виконання адміністративних дій). Він також ініціює більшість дій оркестровки (наприклад, запуск екземпляра), а також забезпечує дотримання певної політики (переважно перевірки квот).

Процес nova-compute — це насамперед робочий демон, який створює та завершує роботу екземплярів віртуальної машини через API гіпервізора (XenAPI для XenServer/XCP, libvirt для KVM або QEMU, VMwareAPI для VMware тощо). Процес, за допомогою якого він це робить, досить складний, але основи прості: прийняти дії з черги, а потім виконати серію системних команд (наприклад, запуск екземпляра KVM), щоб виконати їх під час оновлення стану в базі даних.

nova-volume керує створенням, приєднанням і від'єднанням z-томів від обчислювальних екземплярів (подібна функціональність Amazon Elastic Block Storage). Він може використовувати томи від різних постачальників, таких як iSCSI або Rados Block Device у Ceph. Новий проект OpenStack, Cinder, з часом замінить функціональність nova-volume. У випуску Folsom novavolume і служба Block Storage матимуть схожу функціональність.

Демон nova-network worker дуже схожий на nova-compute і nova-volume. Він приймає мережеві завдання з черги, а потім виконує завдання для маніпулювання мережею (наприклад, налаштування інтерфейсів мосту або зміна правил iptables). Ця функція переноситься до Neutron, окремого проекту OpenStack. У випуску Folsom велика частина функціональності буде дубльована між nova-network і Neutron.

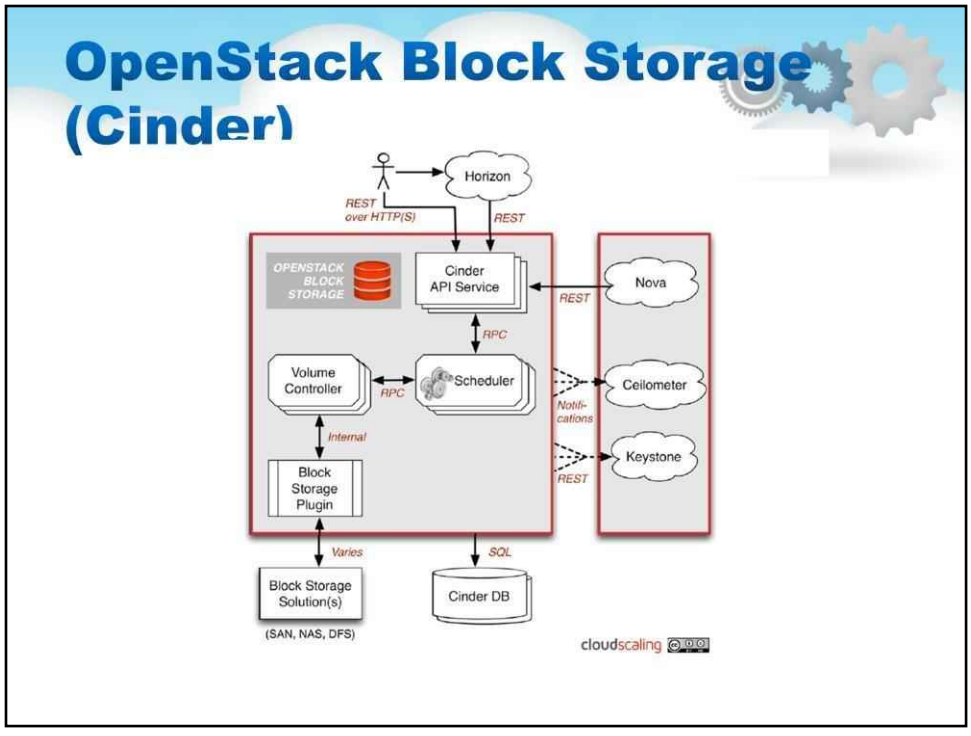
Процес nova-schedule концептуально є найпростішим фрагментом коду в OpenStack Nova: він бере запит екземпляра віртуальної машини з черги та визначає, де він має виконуватися (зокрема, на якому хості обчислювального сервера він має працювати).

Черга є центральним центром для передачі повідомлень між демонами. Сьогодні це зазвичай реалізовано за допомогою RabbitMQ, але це може бути будь-яка черга повідомлень AMQP (наприклад, Apache Qpid). Новим у випуску Folsom є підтримка Zero MQ.

База даних SQL зберігає більшу частину часу збирання та стану виконання для хмарної інфраструктури. Це включає типи екземплярів, які доступні для використання, екземпляри, що використовуються, доступні мережі та проекти. Теоретично OpenStack Nova може підтримувати будь-яку базу даних, що підтримується SQL-Alchemy, але єдиними базами даних, які зараз широко використовуються, є SQLite3 (підходить лише для тестування та розробки), MySQL і PostgreSQL.

Nova також надає консольні послуги, щоб дозволити кінцевим користувачам отримувати доступ до консолі свого віртуального екземпляра через проксі. Це стосується кількох демонов (nova-console, nova-novncproxy і nova-consoleauth).

Nova взаємодіє з багатьма іншими службами OpenStack: Keystone для автентифікації, Glance для зображень і Horizon для веб-інтерфейсу. Взаємодія Glance є центральною. Процес API може завантажувати та надсилати запити Glance, тоді як nova-compute завантажуватиме зображення для використання під час запуску зображень.



Блокові томи сховища повністю інтегровані в OpenStack Compute і панель приладів, що дозволяє користувачам хмари керувати власними потребами в сховищі. Окрім локального серверного сховища Linux, він може використовувати такі платформи зберігання, як Ceph, CloudByte, Coraid, EMC (VMAX і VNX), GlusterFS, IBM Storage (сімейство Storwize, SAN Volume Controller і XIV Storage System), Linux LIO, NetApp, Nexenta, Scality, SolidFire і HP (сімейства Store Virtual і StoreServ 3Par). Блокове сховище підходить для чутливих до продуктивності сценаріїв, таких як сховище бази даних, розширювані файлові системи або надання серверу доступу до необробленого сховища на блочному рівні. Керування знімками забезпечує потужні функції для резервного копіювання даних, що зберігаються на блокових томах зберігання. Моментальні знімки можна відновити або використати для створення нового блокового тома сховища.

Кілька моментів щодо блокового сховища OpenStack:

OpenStack надає постійні пристрої зберігання на блочному рівні для використання з екземплярами обчислень OpenStack.

Блокова система зберігання керує створенням, підключенням і від’єднанням блокових пристроїв до серверів. Блокові томи сховища повністю інтегровані в OpenStack Compute і панель приладів, що дозволяє користувачам хмари керувати власними потребами в сховищі.

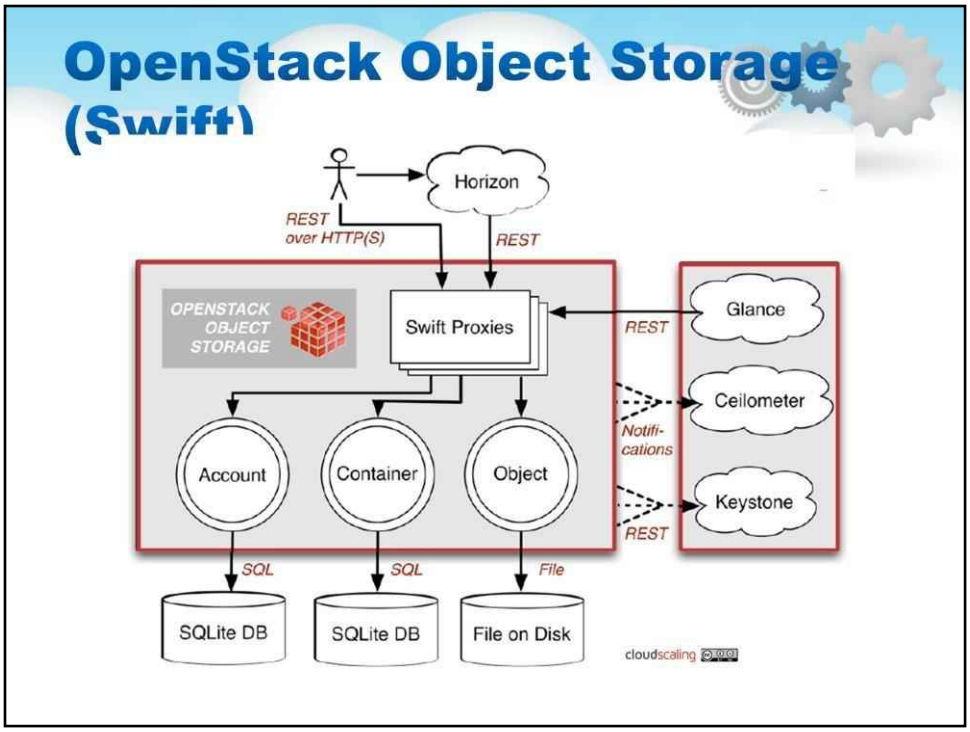
На додаток до використання простого серверного сховища Linux, він має уніфіковану підтримку зберігання для багатьох платформ зберігання, включаючи Ceph, NetApp, Nexenta, SolidFire і Zadara.

Блокове сховище підходить для чутливих до продуктивності сценаріїв, таких як сховище бази даних, розширювані файлові системи або надання серверу доступу до необробленого сховища на блочному рівні.

Керування знімками забезпечує потужні функції для резервного копіювання даних, що зберігаються на блокових томах зберігання. Моментальні знімки можна відновити або використати для створення нового блокового тома сховища.

Cinder відокремлює функцію постійного блокового зберігання, яка раніше була частиною OpenStack Compute (у формі nova-volume), у власну службу. OpenStack Block Storage API дозволяє маніпулювати томами, типами томів (подібно до обчислень) і знімками томів.
cinder-api приймає запити API та направляє їх на cinder-volume для виконання.

cinder-volume виконує запити, зчитуючи або записуючи в базу даних Cinder для підтримки стану, взаємодіючи з іншими процесами (як-от cinder-scheduler) через чергу повідомлень і безпосередньо через сховище блоків, що забезпечує апаратне або програмне забезпечення. Він може взаємодіяти з різними постачальниками сховищ через архітектуру драйвера. Наразі існують драйвери для IBM, SolidFire, NetApp, Nexenta, Zadara, linux iSCSI та інших постачальників сховищ.
Подібно до nova-scheduler, демон cinder-scheduler вибирає оптимальний вузол постачальника блокового сховища для створення тому.
Розгортання Cinder також використовуватиме чергу повідомлень для маршрутизації інформації між процесами cinder, а також базу даних для зберігання стану томів.
Як і Neutron, Cinder в основному взаємодіятиме з Nova, забезпечуючи томи для своїх екземплярів.



OpenStack Object Storage (Swift) — це масштабована резервована система зберігання. Об’єкти та файли записуються на кілька дисків, розподілених по серверах у центрі обробки даних, при цьому програмне забезпечення OpenStack відповідає за забезпечення реплікації та цілісності даних у кластері. Кластери сховищ масштабуються горизонтально, просто додаючи нові сервери. У разі збою сервера або жорсткого диска OpenStack копіює свій вміст з інших активних вузлів у нові місця в кластері. Оскільки OpenStack використовує програмну логіку для забезпечення реплікації та розподілу даних між різними пристроями, можна використовувати недорогі стандартні жорсткі диски та сервери.

Object Storage ідеально підходить для економічного, масштабованого зберігання. Він забезпечує повністю розподілену платформу зберігання, доступну через API, яку можна інтегрувати безпосередньо в програми або використовувати для резервного копіювання, архівування та збереження даних. Block Storage дозволяє відкривати блокові пристрої та підключати їх до обчислювальних екземплярів для розширення сховища, кращої продуктивності та інтеграції з корпоративними платформами зберігання, такими як NetApp, Nexenta та SolidFire.

Декілька деталей про сховище об’єктів OpenStack

OpenStack забезпечує резервне, масштабоване сховище об’єктів за допомогою кластерів стандартизованих серверів, здатних зберігати петабайти даних

Object Storage — це не традиційна файлова система, а скоріше розподілена система зберігання статичних даних, таких як зображення віртуальних машин, сховище фотографій, сховище електронної пошти, резервні копії та архіви. Відсутність центрального «мозку» або головної точки керування забезпечує більшу масштабованість, резервування та довговічність.

Об’єкти та файли записуються на кілька дисків, розподілених по серверах у центрі обробки даних, при цьому програмне забезпечення OpenStack відповідає за забезпечення реплікації та цілісності даних у кластері.

Кластери сховищ масштабуються горизонтально, просто додаючи нові сервери. У разі збою сервера або жорсткого диска OpenStack копіює свій вміст з інших активних вузлів у нові місця в кластері. Оскільки OpenStack використовує програмну логіку для забезпечення реплікації та розподілу даних між різними пристроями, замість дорожчого обладнання можна використовувати недорогі стандартні жорсткі диски та сервери. Швидка архітектура дуже розподілена, щоб запобігти будь-якій точці відмови, а також масштабуватись горизонтально.

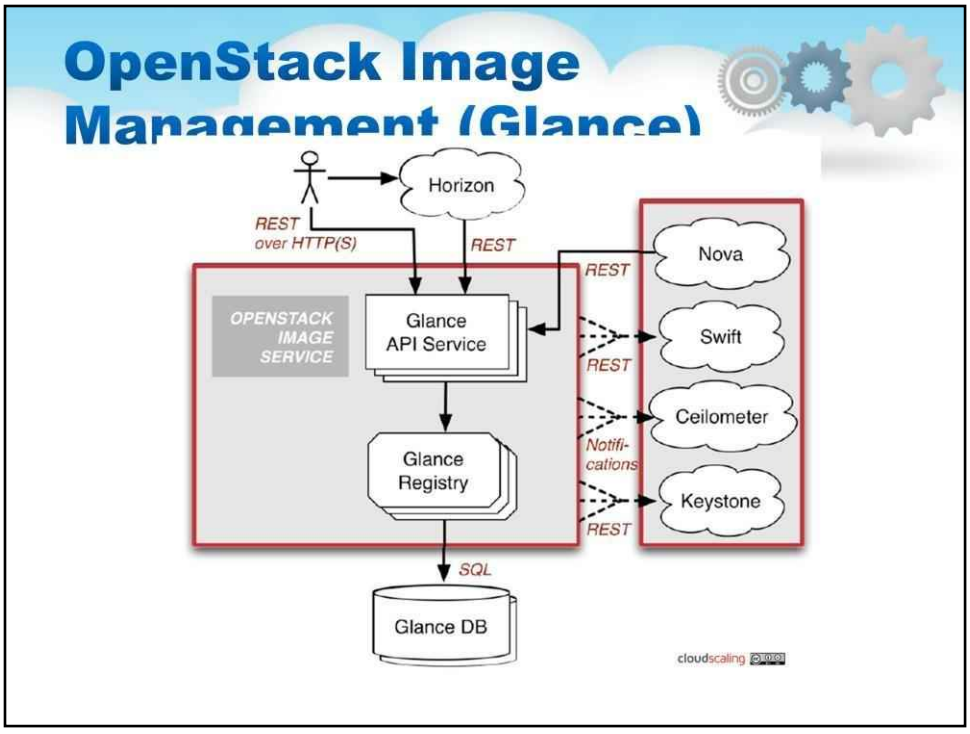
Він включає наступні компоненти:

Проксі-сервер (swift-proxy-server) приймає вхідні запити через OpenStack Object API або просто необроблений HTTP. Він приймає файли для завантаження, зміни метаданих або створення контейнера. Крім того, він також надаватиме файл або список контейнерів веб-переглядачам. Проксі-сервер може використовувати додатковий кеш (зазвичай розгортається з memcache) для підвищення продуктивності.

Сервери облікових записів керують обліковими записами, визначеними за допомогою служби зберігання об’єктів.

Сервери контейнерів керують відображенням контейнерів (тобто папок) у службі зберігання об’єктів. Сервери об’єктів керують фактичними об’єктами (тобто файлами) на вузлах зберігання.

Існує також низка періодичних процесів, які виконуються для виконання службових завдань у великому сховищі даних. Найважливішою з них є служби реплікації, які забезпечують послідовність і доступність через кластер. Ін періодичні процеси включають аудиторів, оновлювачів і женців. Автентифікація здійснюється за допомогою настроюваного проміжного програмного забезпечення WSGI (яке зазвичай є Keystone).



Служба зображень OpenStack (Glance) надає послуги пошуку, реєстрації та доставки образів дисків і серверів. Збережені зображення можна використовувати як шаблон. Їх також можна використовувати для зберігання та каталогізації необмеженої кількості резервних копій. Служба зображень може зберігати образи дисків і серверів у різноманітних серверних системах, включаючи OpenStack Object Storage. Image Service API надає стандартний інтерфейс REST для запиту інформації про образи дисків і дозволяє клієнтам передавати зображення на нові сервери.

Можливості Image Service включають:

- Адміністратори можуть створювати базові шаблони, з яких їхні користувачі можуть запускати нові екземпляри обчислень. Користувачі можуть вибирати з доступних зображень або створювати власні з наявних серверів

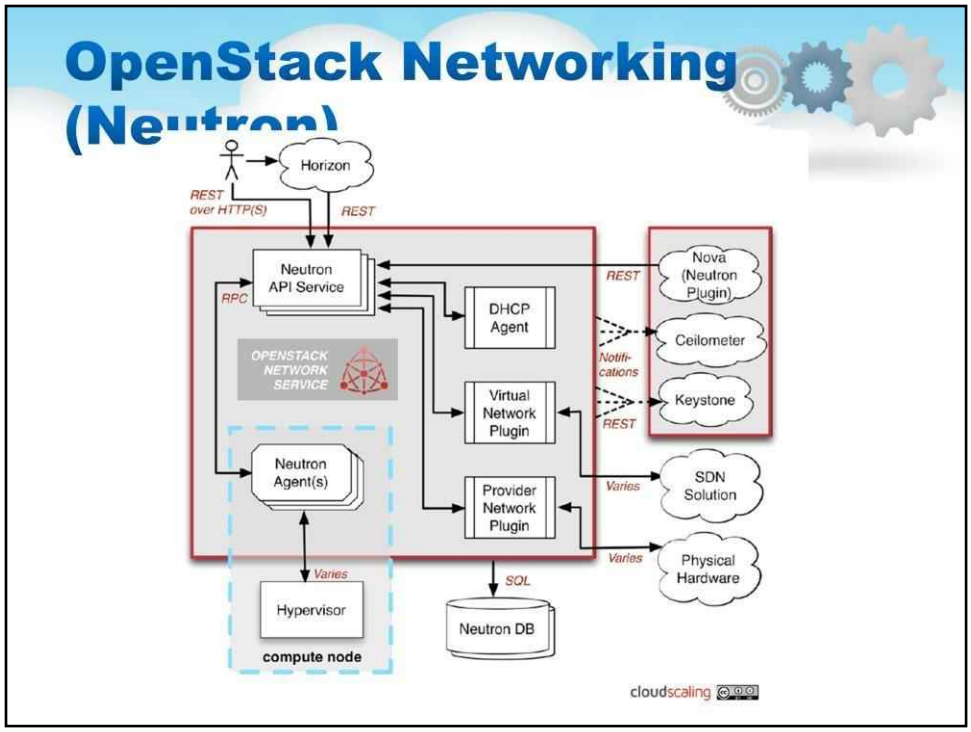
- Знімки також можна зберігати в Image Service, щоб можна було швидко створити резервні копії віртуальних машин
- Багатоформатний реєстр зображень, служба зображень дозволяє завантажувати приватні та публічні зображення в різних форматах, зокрема:
  - Сирий
  - Машина (ядро/рамдиск поза образом, також відомий як AMI) VHD (Hyper-V)
  - VDI (VirtualBox) qcow2 (Qemu/KVM) VMDK (VMWare)
  - OVF (VMWare, інші)

Архітектура Glance залишалася відносно стабільною після випуску Cactus. Найбільшою архітектурною зміною стало додавання автентифікації, яка була додана у випуску Diablo. Коротко нагадаю, що Glance складається з чотирьох основних частин:

1) glance-api приймає виклики Image API для виявлення, пошуку та зберігання зображень.
2) glance-registry зберігає, обробляє та отримує метадані про зображення (розмір, тип тощо).
3) База даних для зберігання метаданих зображення. Як і Nova, ви можете вибрати свою базу даних залежно від ваших уподобань (але більшість людей використовують MySQL або SQLite).
4) Репозиторій для фактичних файлів зображень. На схемі вище Swift показано як сховище зображень, але це можна налаштувати. Окрім Swift, Glance підтримує звичайні файлові системи, блокові пристрої RADOS, Amazon S3 і HTTP. Майте на увазі, що деякі з цих варіантів обмежено використанням лише для читання.

Існує також ряд періодичних процесів, які запускаються на Glance для підтримки кешування. Найважливішою з них є служби реплікації, які забезпечують послідовність і доступність через кластер. Інші періодичні процеси включають аудиторів, оновлювачів і женців.

Як видно на діаграмі в розділі «Концептуальна архітектура», Glance відіграє центральну роль у загальній картині IaaS. Він приймає запити API для зображень (або метаданих зображень) від кінцевих користувачів або компонентів Nova і може зберігати свої дискові файли в службі зберігання об’єктів Swift.



OpenStack Networking (Neutron, раніше Quantum) — це плагінована, масштабована та керована API система для керування мережами та IP-адресами. Як і інші аспекти хмарної операційної системи, її можуть використовувати адміністратори та користувачі для збільшення вартості наявних активів центру обробки даних. OpenStack Networking гарантує, що мережа не буде вузьким місцем або обмежуючим фактором у розгортанні хмари та надає користувачам справжнє самообслуговування навіть у їхніх конфігураціях мережі.

OpenStack Networking — це система для керування мережами та IP-адресами. Як і інші аспекти хмарної операційної системи, її можуть використовувати адміністратори та користувачі для збільшення вартості наявних активів центру обробки даних. OpenStack Networking гарантує, що мережа не буде вузьким місцем або обмежуючим фактором у розгортанні хмари та надає користувачам справжнє самообслуговування навіть у їхніх конфігураціях мережі.

OpenStack Neutron надає моделі мереж для різних програм або груп користувачів. Стандартні моделі включають плоскі мережі або VLAN для розділення серверів і трафіку. OpenStack Networking керує IP-адресами, дозволяючи використовувати виділені статичні IP-адреси або DHCP. Плаваючі IP-адреси дозволяють динамічно перенаправляти трафік до будь-якого з ваших обчислювальних ресурсів, що дає змогу перенаправляти трафік під час обслуговування або в разі збою. Користувачі можуть створювати власні мережі, контролювати трафік і підключати сервери та пристрої до однієї або кількох мереж. Адміністратори можуть скористатися перевагами програмно-визначеної мережевої технології (SDN), як-от OpenFlow, щоб забезпечити високий рівень мультитенантності та великого масштабу. OpenStack Networking має структуру розширення, яка надає додаткові мережеві служби, такі як системи виявлення вторгнень (IDS), балансування навантаження,

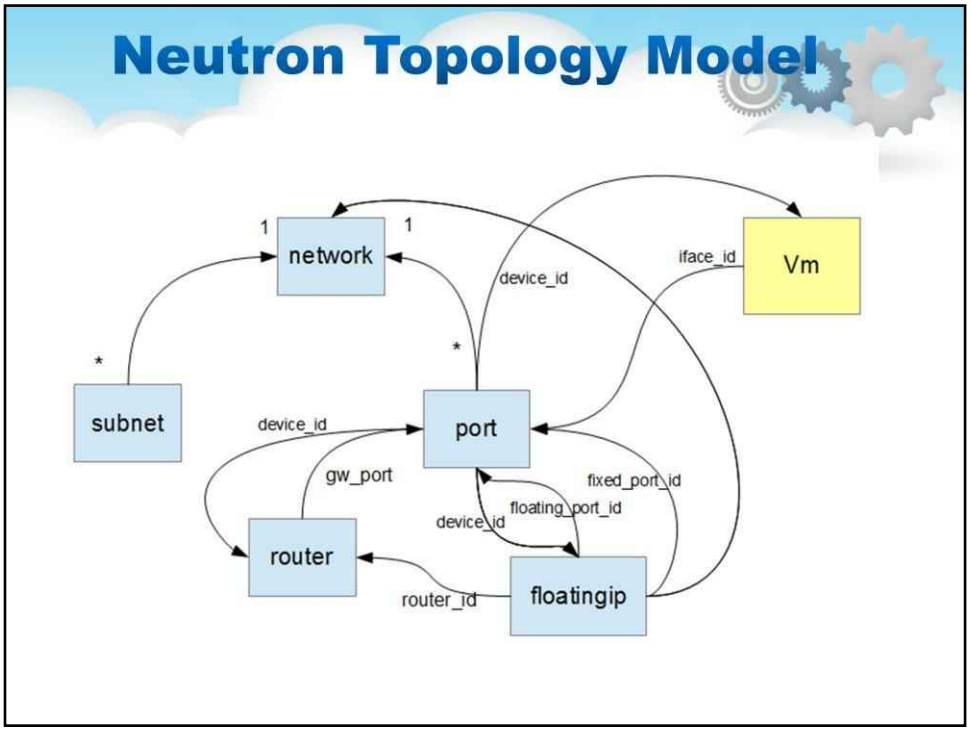
Мережеві можливості

OpenStack надає гнучкі моделі мереж, які відповідають потребам різних програм або груп користувачів. Стандартні моделі включають плоскі мережі або VLAN для розділення серверів і трафіку.

OpenStack Networking керує IP-адресами, дозволяючи використовувати виділені статичні IP-адреси або DHCP. Плаваючі IP-адреси дозволяють динамічно перенаправляти трафік до будь-якого з ваших обчислювальних ресурсів, що дає змогу перенаправляти трафік під час обслуговування або в разі збою.
Користувачі можуть створювати власні мережі, контролювати трафік і підключати сервери та пристрої до однієї або кількох мереж.
Підключена архітектура серверної частини дає користувачам змогу користуватися стандартним обладнанням або розширеними мережевими послугами від підтримуваних постачальників.

Адміністратори можуть скористатися перевагами програмно-визначеної мережевої технології (SDN), як-от OpenFlow, щоб забезпечити високий рівень мультитенантності та великого масштабу.

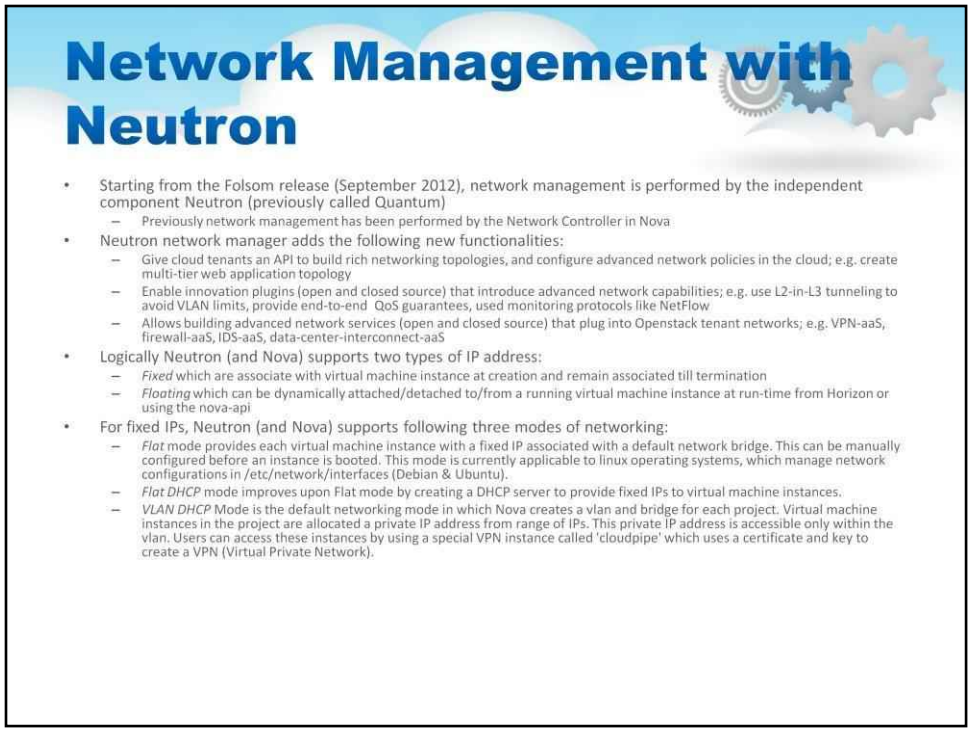
OpenStack Networking має структуру розширення, яка дозволяє розгортати та керувати додатковими мережевими службами, такими як системи виявлення вторгнень (IDS), балансування навантаження, брандмауери та віртуальні приватні мережі (VPN).



Neutron забезпечує «мережеве підключення як послугу» між інтерфейсними пристроями, якими керує інший нейтрон-сервер, приймає запити API, а потім направляє їх до відповідного плагіна Neutron для плагінів Neutron, а агенти виконують фактичні дії, такі як підключення та відключення портів, кре
Загальними агентами є L3 (рівень 3), DHCP (динамічна IP-адресація хоста) і спеціальний плагін ag

Більшість установок Neutron також використовуватимуть чергу повідомлень для маршрутизації інформації між ними

Neutron в основному взаємодіятиме з Nova, де він забезпечуватиме мережі та підключення для своїх інстаграм



Починаючи з випуску Folsom (вересень 2012 р.), управління мережею виконується незалежним компонентом Neutron (раніше називався Quantum).

Раніше керування мережею здійснювалося за допомогою мережевого контролера в Nova

Менеджер мережі Neutron додає такі нові функції:

Надайте хмарним клієнтам API для створення розширених мережевих топологій і налаштування розширених мережевих політик у хмарі; наприклад створити багаторівневу топологію веб-додатків

Увімкніть інноваційні плагіни (з відкритим і закритим кодом), які надають розширені мережеві можливості; наприклад, використовуйте тунелювання L2-in-L3, щоб уникнути обмежень VLAN, надайте наскрізні гарантії QoS, використовуйте протоколи моніторингу, такі як NetFlow

Дозволяє створювати розширені мережеві служби (з відкритим і закритим кодом), які підключаються до мереж орендарів Openstack; наприклад, VPN-aaS, firewall-aaS, IDS-aaS, data-center- interconnect-aaS

Логічно Neutron (і Nova) підтримує два типи IP-адрес:

Виправлено, які асоціюються з екземпляром віртуальної машини під час створення та залишаються пов’язаними до припинення

Плаваючий, який можна динамічно приєднувати/від’єднувати до/від запущеного екземпляра віртуальної машини під час виконання з Horizon або за допомогою nova-api

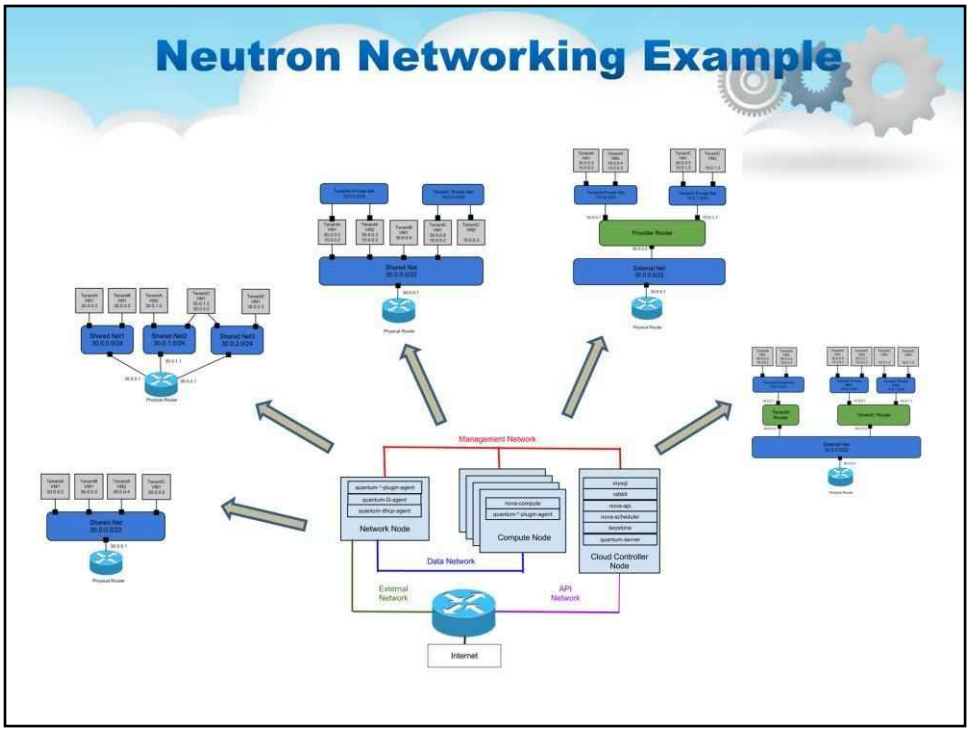
Для фіксованих IP-адрес Neutron (і Nova) підтримує такі три режими мережі:

Плоский режим надає кожному екземпляру віртуальної машини фіксовану IP-адресу, пов’язану з мережевим мостом за умовчанням. Це можна налаштувати вручну перед завантаженням примірника. Цей режим зараз застосовний до операційних систем Linux, які керують конфігураціями мережі

/etc/network/interfaces (Debian і Ubuntu).

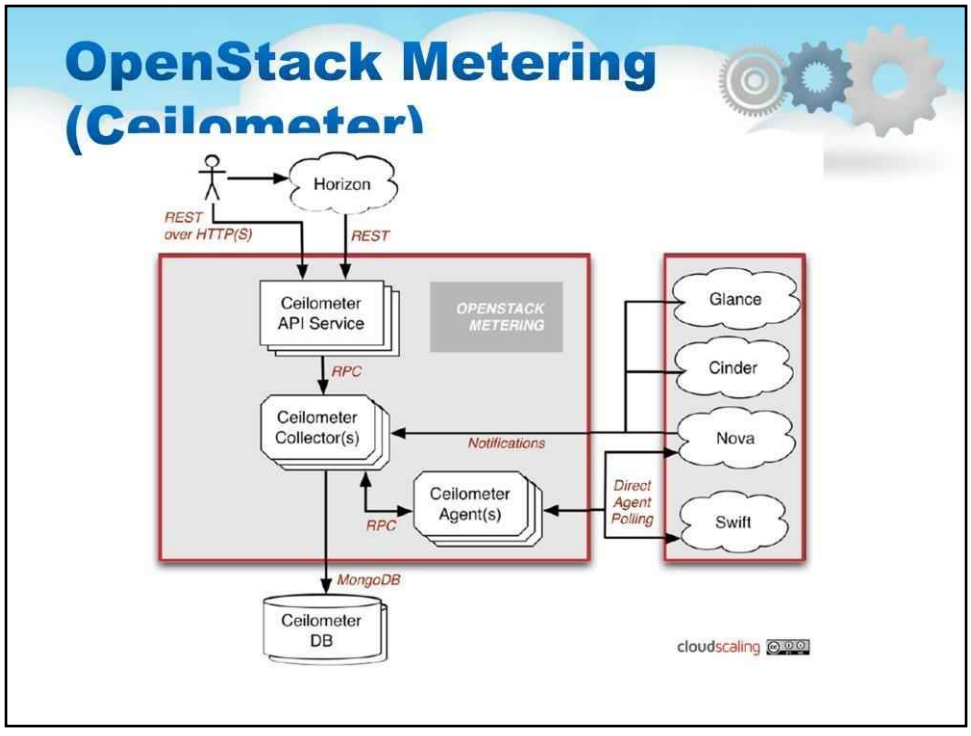
Режим Flat DHCP покращує режим Flat, створюючи сервер DHCP для надання фіксованих IP-адрес для екземплярів віртуальних машин.

Режим VLAN DHCP — це мережевий режим за замовчуванням, у якому Nova створює vlan і міст для кожного проекту. Примірникам віртуальної машини в проекті призначається приватна IP-адреса з діапазону IP-адрес. Ця приватна IP-адреса доступна лише в межах vlan. Користувачі можуть отримати доступ до цих екземплярів за допомогою спеціального екземпляра VPN під назвою «cloudpipe», який використовує сертифікат і ключ для створення VPN (віртуальної приватної мережі).



На цьому слайді показано, як мережеве програмне забезпечення на основі Neutron може бути застосоване для створення багатьох різних мережевих топологій для користувача

Зверніть увагу на можливість створення компонентів віртуального комутатора, а також кількох сегментів мережі.



Проект Ceilometer має на меті стати інфраструктурою для збору вимірювань у OpenStack, щоб не потрібно було писати двох агентів для збору однакових даних. Його головними цілями є моніторинг і вимірювання, але структуру слід легко розширювати для збору для інших потреб. Для цього Ceilometer повинен мати можливість обмінюватися зібраними даними з різними споживачами.

Агент запускається на кожному вузлі OpenStack (машина Bare Metal) і збирає дані локально

Якщо лічильник доступний з наявного компонента OpenStack, його слід використовувати. Автономний агент ceilometer реалізує лічильники, які ще не доступні з існуючих компонентів OpenStack

Демон зберігання спілкується з агентами, щоб збирати їхні дані та агрегувати їх

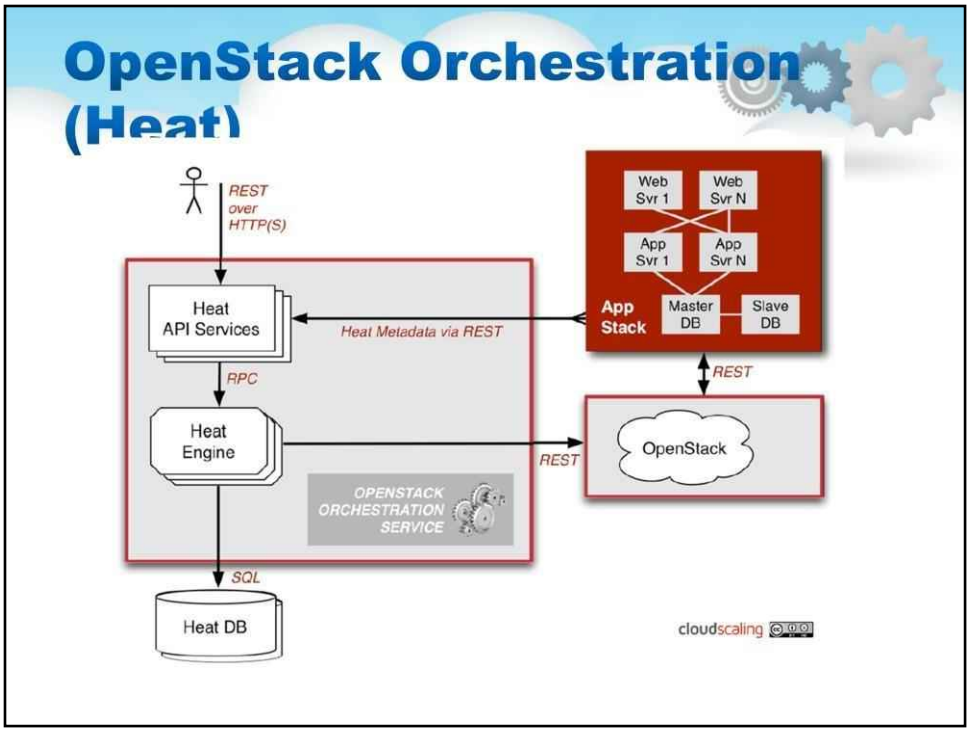
Агенти, які збирають дані, проходять аутентифікацію, щоб уникнути забруднення послуги вимірювання

Дані надсилаються від агентів до демона зберігання через довірену систему обміну повідомленнями (RabbitMQ?)

Дані/повідомлення, якими обмінюються агенти та демон зберігання, використовують загальний формат повідомлень

Вміст сховища стає доступним через REST API, що забезпечує агрегацію. Черга повідомлень відокремлена від інших черг (наприклад, черга nova)

Повідомлення в черзі підписані та не підлягають відкиданню



Heat є основним проектом у програмі OpenStack Orchestration. Він реалізує механізм оркестровки для запуску кількох складених хмарних програм на основі шаблонів у формі текстових файлів, які можна розглядати як код. Власний формат шаблону Heat розвивається, але Heat також намагається забезпечити сумісність із форматом шаблону AWS CloudFormation, щоб багато існуючих шаблонів CloudFormation можна було запускати на OpenStack.

Heat надає як OpenStack-native ReST API, так і CloudFormation-сумісний Query API.

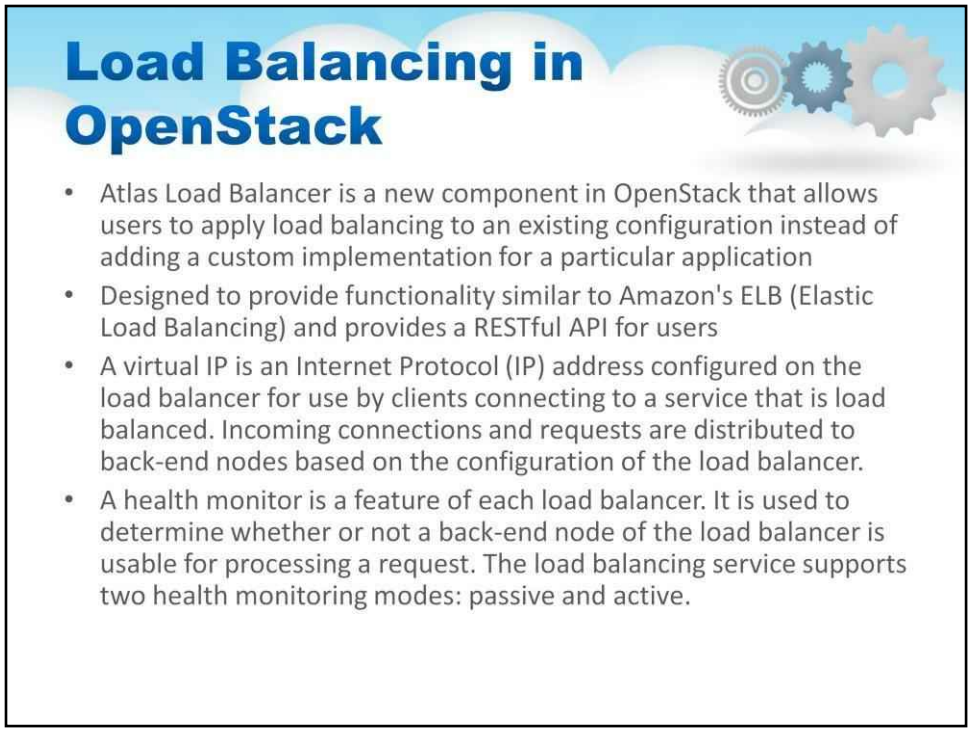
Шаблон Heat описує інфраструктуру для хмарної програми в текстовому файлі, доступному для читання та запису людьми, і який можна перевіряти в системі керування версіями, змінювати тощо. Ресурси інфраструктури, які можна описати, включають: сервери, плаваючі IP-адреси, томи, групи безпеки, користувачів тощо.

Heat також надає службу автоматичного масштабування, яка інтегрується з Ceilometer, тому ви можете включити групу масштабування як ресурс у шаблон.

Шаблони також можуть визначати зв'язки між ресурсами (наприклад, цей том підключено до цього сервера). Це дає змогу Heat звертатися до OpenStack API, щоб створити всю вашу інфраструктуру в правильному порядку для повного запуску програми.

Heat керує всім життєвим циклом програми - коли вам потрібно змінити інфраструктуру, просто змініть шаблон і використовуйте його для оновлення наявного стеку. Тепло знає, як внести необхідні зміни. Він також видалить усі ресурси, коли ви завершите роботу з програмою.

Heat переважно керує інфраструктурою, але шаблони добре інтегруються з інструментами керування конфігурацією програмного забезпечення, такими як Puppet і Chef. Команда Heat працює над забезпеченням ще кращої інтеграції між інфраструктурою та програмним забезпеченням.



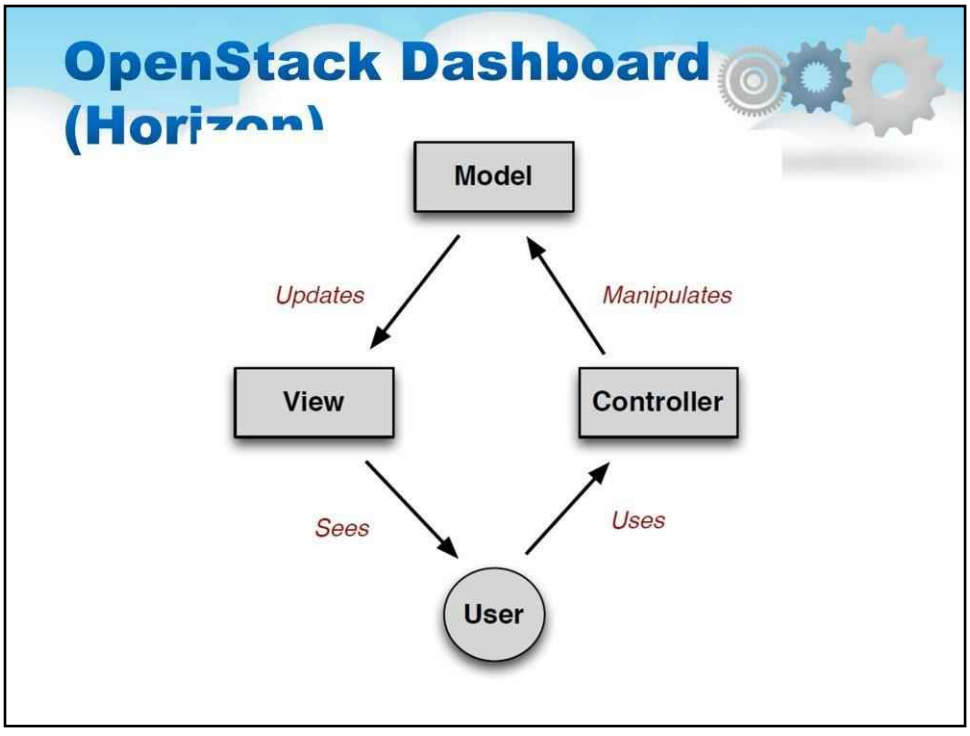
Балансувальник навантаження - це логічний пристрій. Він використовується для розподілу робочого навантаження між декількома внутрішніми системами або службами, які називаються вузлами, на основі критеріїв, визначених як частина його конфігурації.

Atlas Load Balancer — це новий компонент OpenStack, який дозволяє користувачам застосовувати балансування навантаження до наявної конфігурації замість додавання спеціальної реалізації для конкретної програми

Розроблено для забезпечення функціональності, подібної до ELB (пружного балансування навантаження) Amazon, і забезпечує RESTful API для користувачів

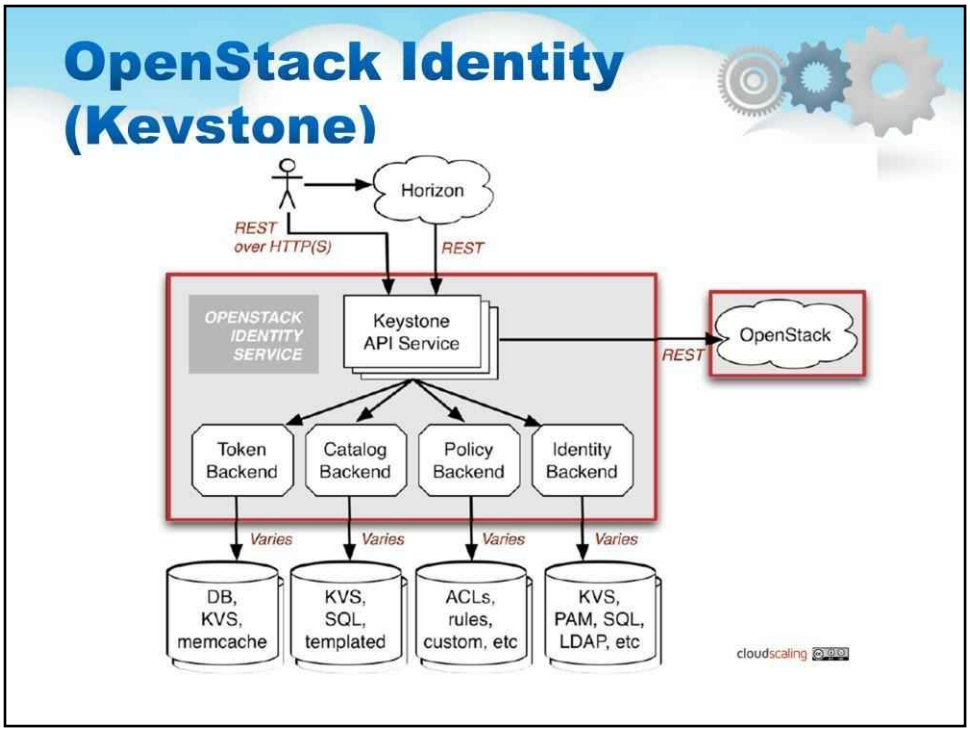
Віртуальний IP – це адреса Інтернет-протоколу (IP), налаштована на балансирі навантаження для використання клієнтами, які підключаються до служби з балансуванням навантаження. Вхідні підключення та запити розподіляються між серверними вузлами на основі конфігурації балансувальника навантаження.

Монітор справності є функцією кожного балансувальника навантаження. Він використовується для визначення того, чи можна використовувати внутрішній вузол балансувальника навантаження для обробки запиту. Служба балансування навантаження підтримує два режими моніторингу працездатності: пасивний і активний.



Конструкція дозволяє використовувати сторонні продукти та послуги, такі як виставлення рахунків, моніторинг і додаткові інструменти керування. Постачальники послуг та інші комерційні постачальники можуть налаштувати інформаційну панель під власний бренд.

Інформаційна панель — це лише один із способів взаємодії з ресурсами OpenStack. Розробники можуть автоматизувати доступ або створити інструменти для керування своїми ресурсами за допомогою рідного API OpenStack або API сумісності EC2.



OpenStack Identity (Keystone) надає центральний каталог користувачів, зіставлений із службами OpenStack, до яких вони мають доступ. Він діє як звичайна система автентифікації в хмарній операційній системі та може інтегруватися з існуючими серверними службами каталогів, такими як LDAP. Він підтримує різні форми автентифікації, включаючи стандартні облікові дані для імені користувача та пароля, системи на основі маркерів і облікові дані для входу в Amazon Web Services, такі як ті, що використовуються для EC2.

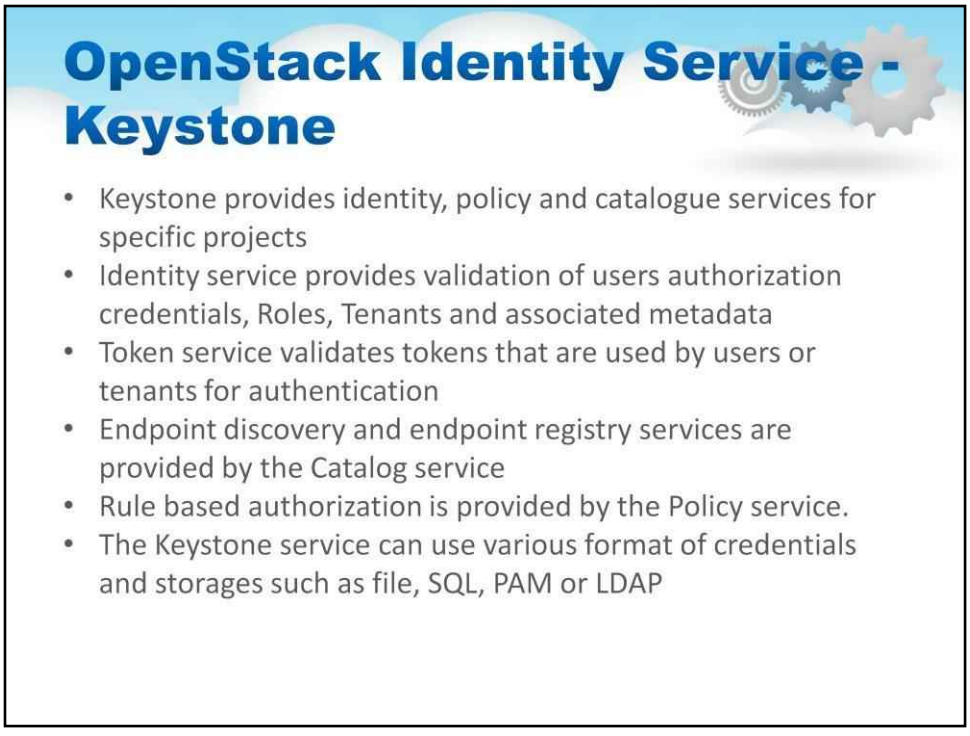
Крім того, каталог надає доступний для запиту список усіх служб, розгорнутих у хмарі OpenStack, в одному реєстрі. Користувачі та інструменти сторонніх розробників можуть програмно визначати, до яких ресурсів вони мають доступ.

Служба ідентифікації OpenStack дозволяє адміністраторам:

Налаштуйте централізовані політики для користувачів і систем
Створюйте користувачів і орендарів і визначте дозволи для обчислювальних ресурсів, сховищ і мережевих ресурсів за допомогою функцій керування доступом на основі ролей (RBAC). Інтеграція з наявним каталогом, наприклад LDAP, для забезпечення єдиного джерела автентифікації в масштабах підприємства

Служба ідентифікації OpenStack дозволяє користувачам:

Перелічіть служби, до яких вони мають
доступ. Робіть запити API
Увійдіть на веб-панель, щоб створити ресурси, які належать їхнім обліковим записам



Keystone забезпечує єдину точку інтеграції політики, каталогу, маркера та автентифікації OpenStack.

Keystone обробляє запити API, а також надає настроювані послуги каталогу, політики, маркера та ідентифікації.

Служба ідентифікації забезпечує перевірку облікових даних авторизації користувачів, ролей, орендарів і пов’язаних метаданих

Служба маркерів перевіряє маркери, які використовуються користувачами або орендарями для автентифікації

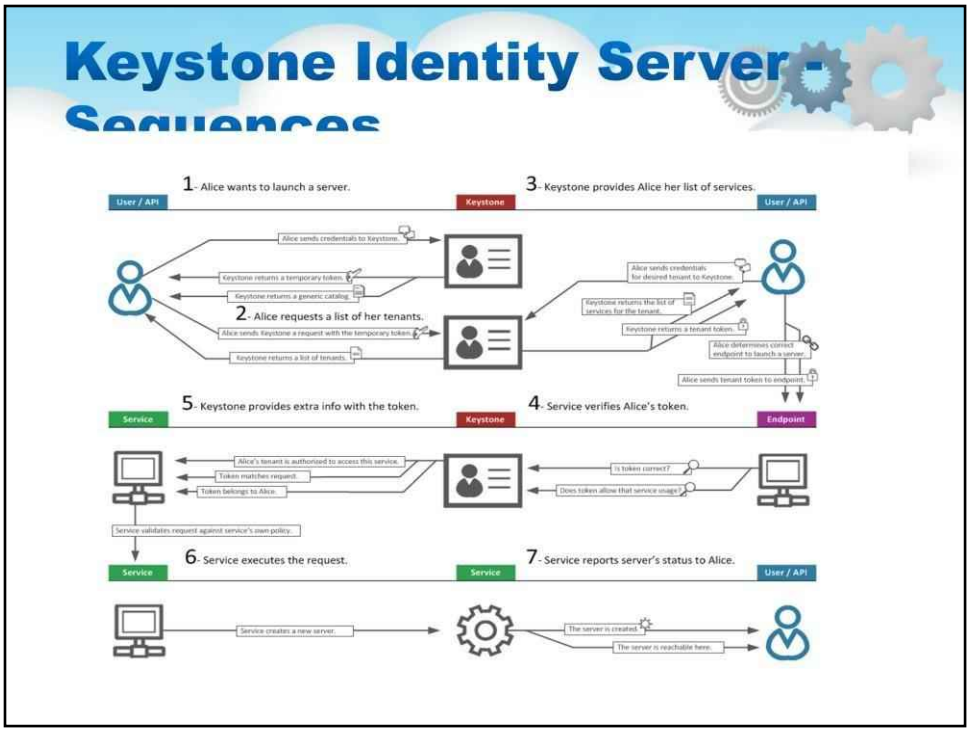
Послуги виявлення кінцевих точок і реєстру кінцевих точок надаються службою Catalog

Авторизація на основі правил забезпечується службою політики.

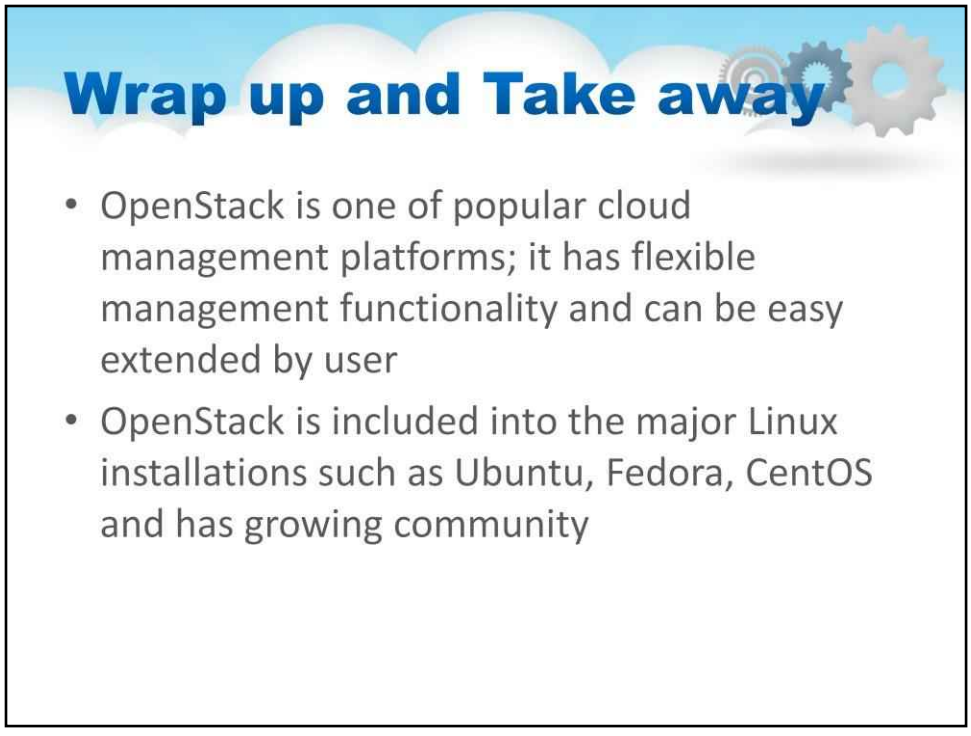
Служба Keystone може використовувати різні формати облікових даних і сховищ, як-от файл, SQL, PAM або LDAP

Кожна функція Keystone має вбудований сервер, який дозволяє різними способами використовувати певну службу. Більшість підтримує стандартні серверні модулі, такі як LDAP або SQL, а також сховища ключових значень (KVS).

Більшість людей використовуватимуть це як пункт налаштування своїх поточних служб автентифікації.

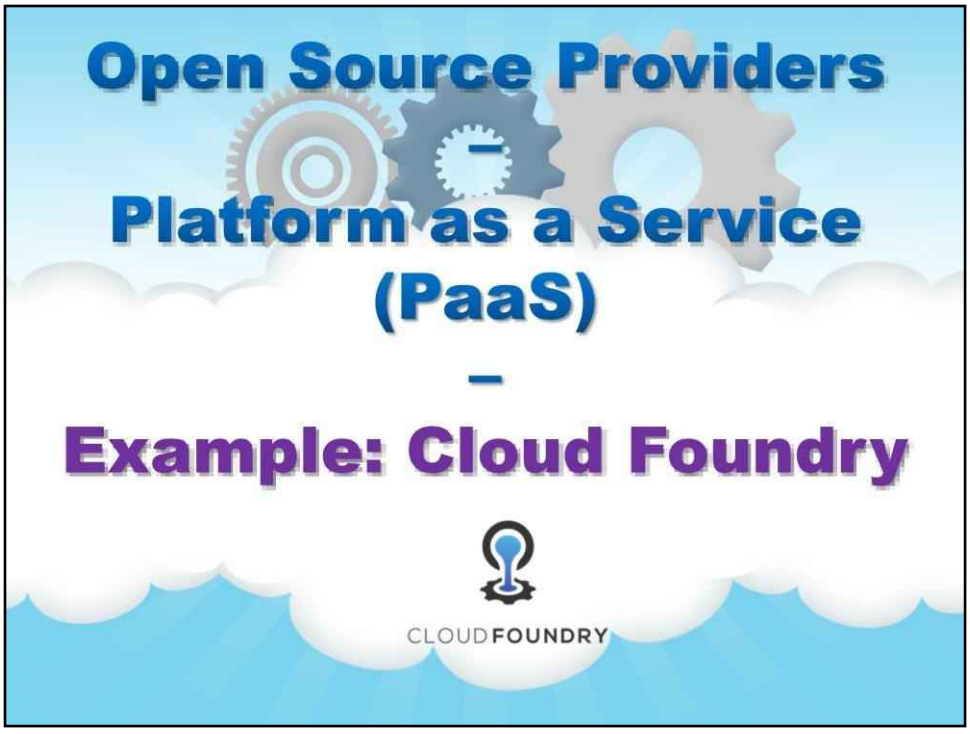


Діаграма на цьому слайді показує послідовність того, як використовується трапеція.

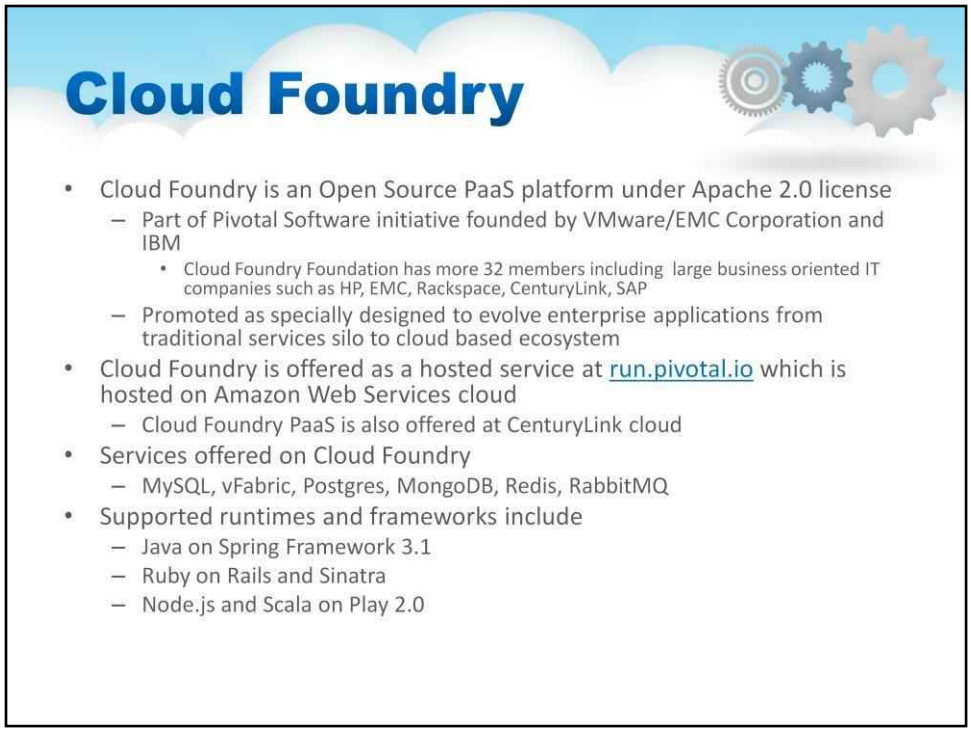


OpenStack — одна з популярних хмарних платформ керування; він має гнучкі функції керування та може бути легко розширений користувачем

OpenStack включено до основних установок Linux, таких як Ubuntu, Fedora, CentOS, і має зростаючу спільноту



Приклад: Cloud Fondry



Cloud Foundry — це платформа PaaS з відкритим кодом під ліцензією Apache 2.0, Apache 2.0 є найменш обмежувальною ліцензією. Розроблений код можна використовувати як в комерційних, так і в некомерційних цілях. Частина ініціативи Pivotal Software, заснованої VMware/EMC Corporation і IBM Cloud Foundry Foundation, налічує понад 32 члени, включаючи великі IT-компанії, орієнтовані на бізнес, такі як HP, EMC, Rackspace, CenturyLink, SAP

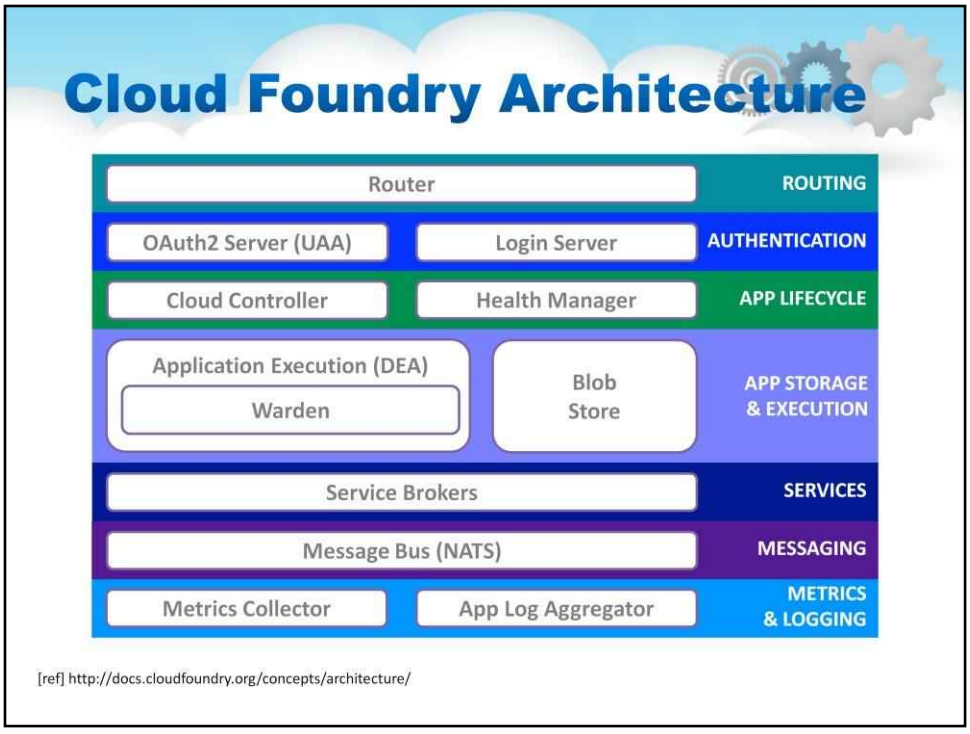
Рекламується як спеціально розроблений для розвитку корпоративних додатків від традиційних служб до хмарної екосистеми

Послуги, які пропонує Cloud Foundry:

- MySQL - реляційна база даних з відкритим кодом
- Postgres - реляційна база даних на основі PostgreSQL
- MongoDB - масштабована, відкрита база даних на основі документів
- Redis - відкритий сервер структури даних ключ-значення
- RabbitMQ - надійний, масштабований і портативний обмін повідомленнями для програм

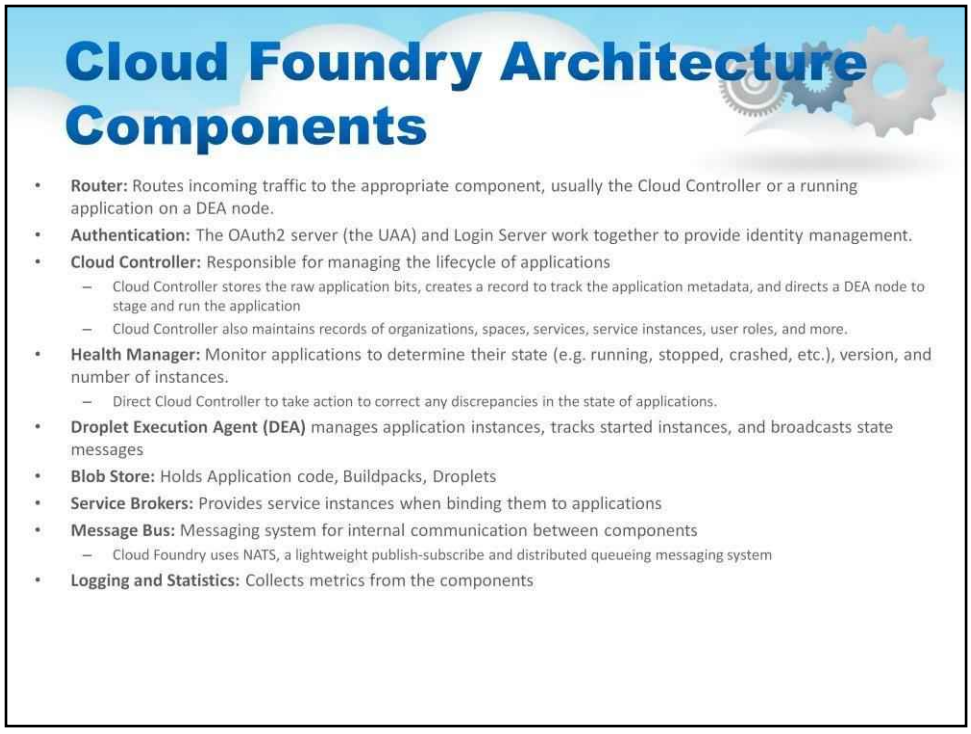
Підтримувані середовища виконання та фреймворки включають:

- Java на Spring Framework 3.1
- Ruby on Rails і Сінатра
- Node.js і Scala на Play 2.0



На цьому слайді показано ілюстрацію «Marketechture» Cloud Foundry.

Наступний слайд буде окремо для кожної області.



Маршрутизатор:Направляє вхідний трафік до відповідного компонента, зазвичай Cloud Controller або запущеної програми на вузлі DEA.

Автентифікація:Сервер OAuth2 (UAA) і сервер входу працюють разом, щоб забезпечити керування ідентифікацією.

Хмарний контролер:Відповідає за управління життєвим циклом програм

Хмарний контролер зберігає необроблені біти програми, створює запис для відстеження метаданих програми та направляє вузол DEA для створення та запуску програми

Cloud Controller також зберігає записи організацій, просторів, служб, примірників служб, ролей користувачів тощо.

Менеджер охорони здоров'я:Відстежуйте програми, щоб визначити їх стан (наприклад, запущено, зупинено, збій тощо), версію та кількість екземплярів.

Направте Cloud Controller вжити заходів для виправлення будь-яких невідповідностей у стані програм.

Droplet Execution Agent (DEA)керує екземплярами програми, відстежує запущені екземпляри та транслює повідомлення про стан.

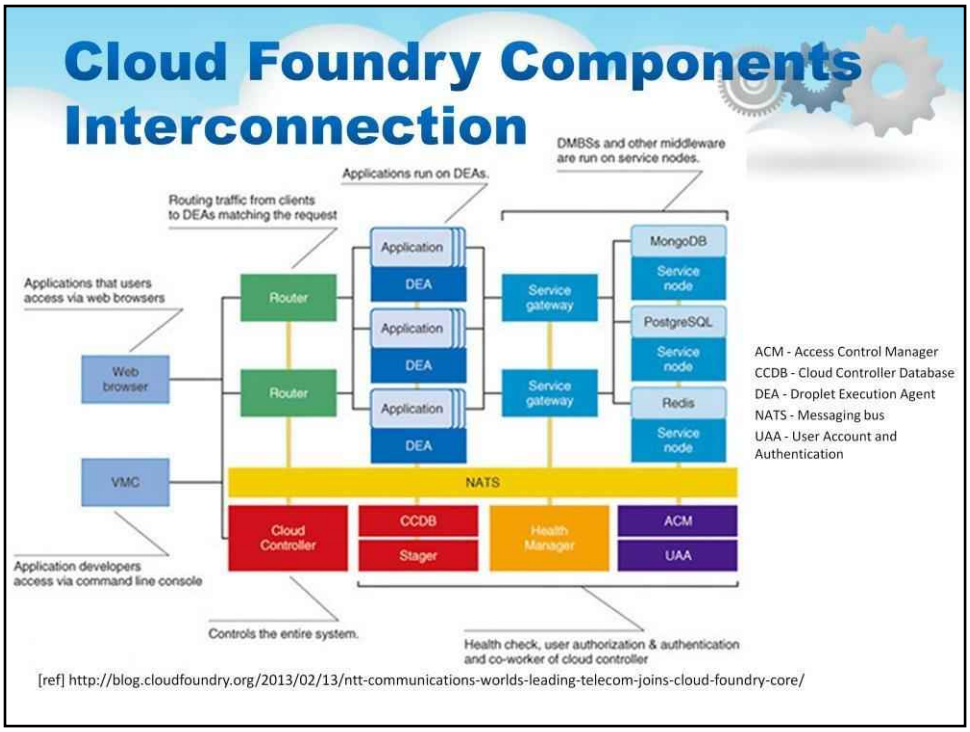
Магазин крапель:Зберігає код програми, пакети збірки, дроплети

Сервісні брокери:Надає екземпляри служби під час прив’язки до програм

Шина повідомлень:Система обміну повідомленнями для внутрішнього зв'язку між компонентами

Cloud Foundry використовує NATS, легку систему публікації-підписки та розподіленої системи обміну повідомленнями в черзі

Логування та статистика:Збирає показники з компонентів



Ці слайди ілюструють більш технічно точну структуру Cloud Foundry,

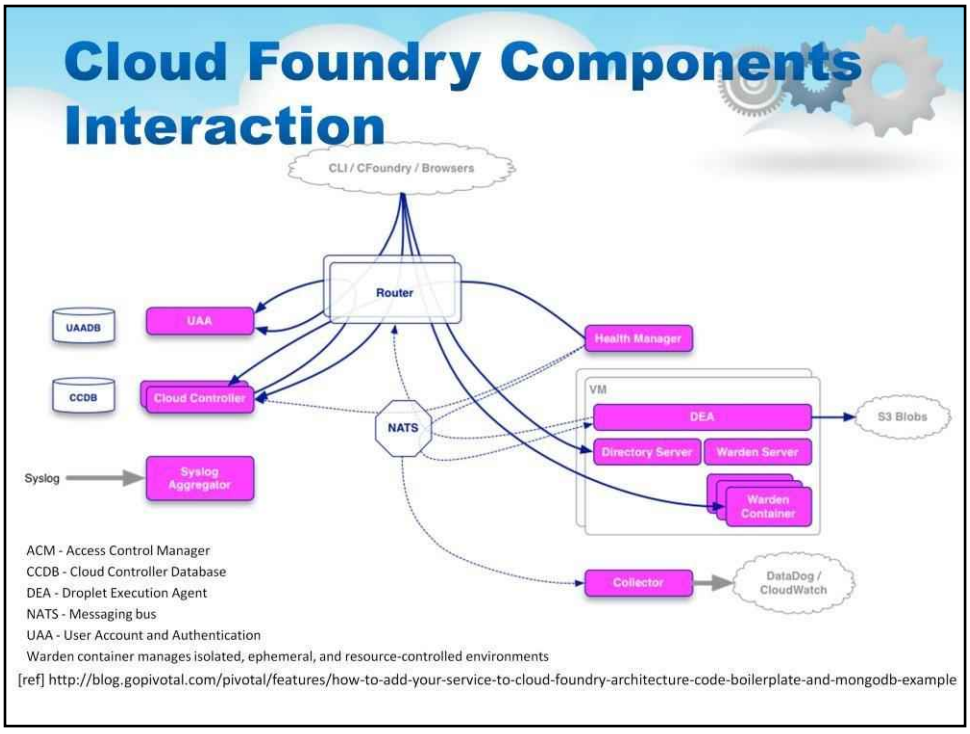
У центрі видно Droplet Execution Engine, який керує поняттям «програми» в Cloud Foundry.

Попереду цих програм є програмні маршрутизатори, які переміщують трафік з Інтернету (браузера чи веб-служб) до програм.

З іншого боку, бек-енд додатків, оскільки додаткам потрібні такі послуги, як база даних або сховище, шлюз служб підключає додатки та DEA до потрібного типу послуги.

Шина повідомлень NATS організовує ці різні рівні для спілкування один з одним. NATS керується хмарним контролером, який контролює всю систему. Інші модулі включають керування, авторизацію та автентифікацію та інші модулі, типові для середовища сервера додатків,

Зверніть увагу на розподілену архітектуру Cloud Foundry. Це не схоже на тісно пов’язані багаторівневі сервери додатків на основі віддаленого виклику процедур минулих років. Ідея, згідно з якою всі частини об’єднані в чергу транзакційних повідомлень – та сама техніка, яку використовують хмари, щоб підтримувати себе разом на рівні IaaS – розроблена для розподіленого середовища. Для хмар також ідеально підходить реплікація DEA і, отже, програм. Це масштабується за допомогою хмарної моделі розпродажу.



Ілюстрація на цьому слайді демонструє потік даних у Cloud Foundry. Суцільні лінії позначають трафік програми, наприклад, HTTP. Пунктирні лінії позначають трафік керуючих повідомлень, наприклад, від NATS.

Можна побачити центральну роль, яку відіграє маршрутизатор у трафіку додатків. Усім трафіком додатків керує маршрутизатор.

Компанією Cloud Foundry є віртуальна машина, де працює Drople Execution Engine. Кожна віртуальна машина запускає Droplet Execution Agent і кілька контейнерів служби Warden.

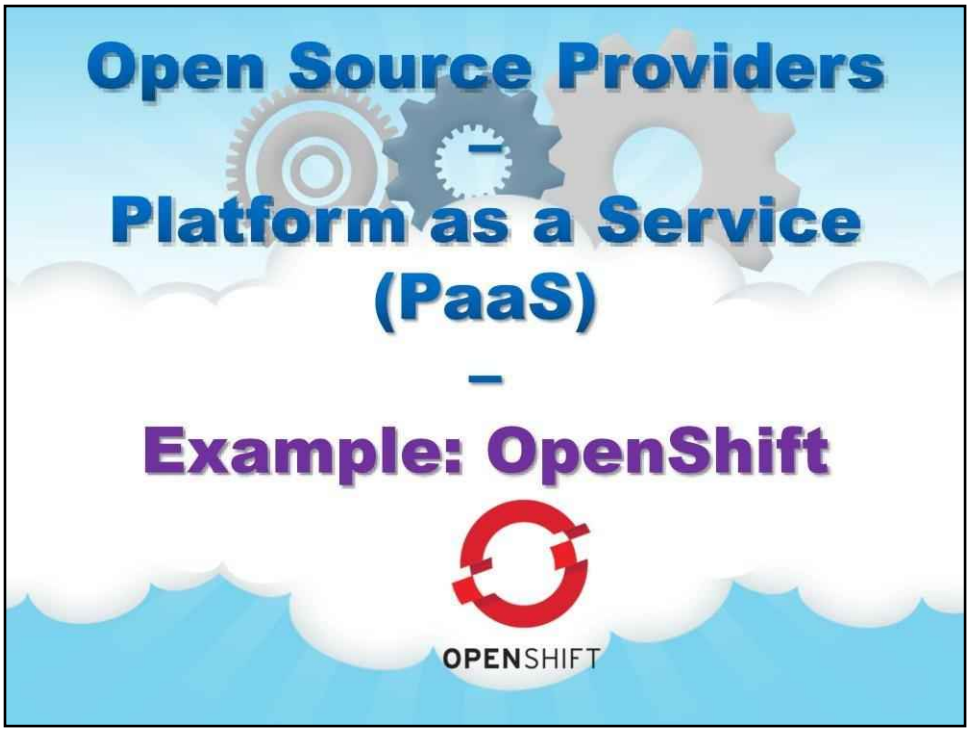
Контейнер Warden керує ізольованими, ефемерними та контрольованими ресурсами середовищами.

NOSH — це інструмент, створений для створення самого Cloud Foundry. У звичайних ситуаціях розробники додатків Cloud Foundry майже не побачать BOSH.

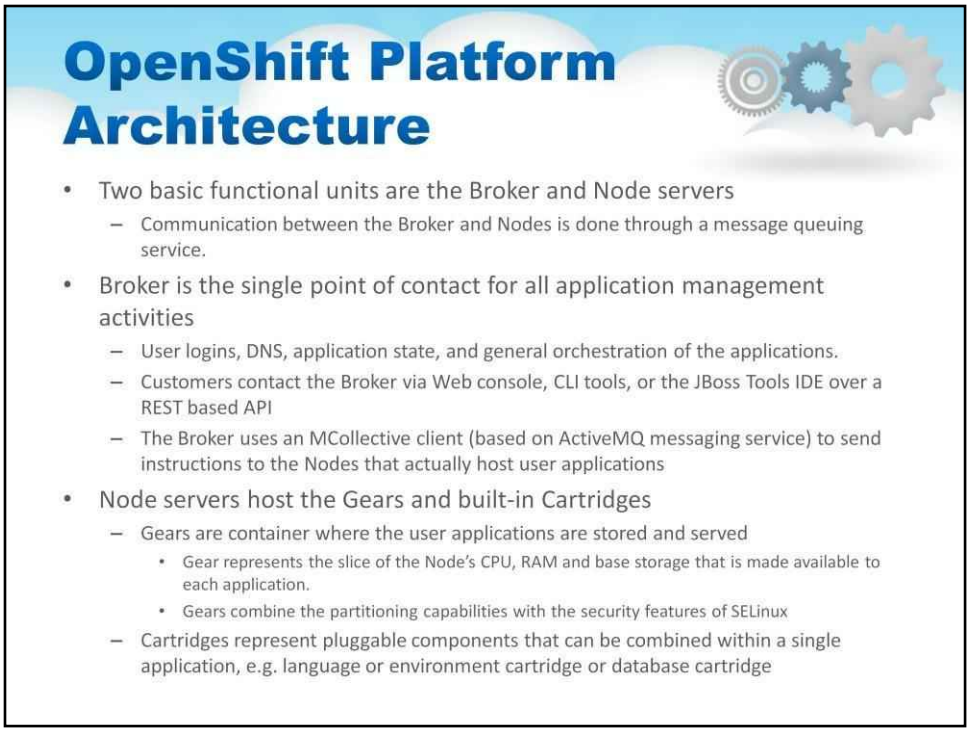
Розробники сервісів і додатків до Cloud Foundry використовуватимуть BOSH,

Незважаючи на те, що BOSH можна застосувати для створення інших фреймворків додатків, оскільки він переноситься практично на будь-яку платформу IaaS, він не знаходить реального застосування за межами розробки Cloud Foundry.

У решті цього слайда детально описано, як BOSH використовується для створення та під час виконання Cloud Foundry.



Приклад: OpenShift

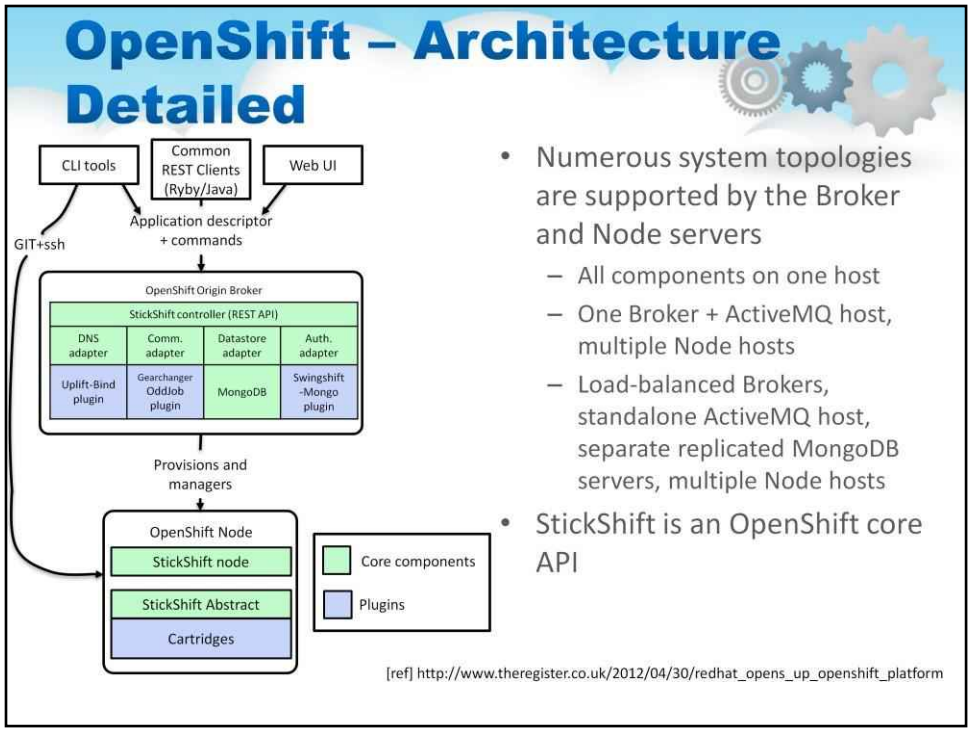


Тепер ми звернемо нашу увагу на розуміння RedHat OpenShift

OpenShift має два основних функціональних блоки - сервери Broker і Node. Зв'язок між Брокером і вузлами здійснюється через службу черги повідомлень, як і в CloudFoundry.

Посередник є єдиною точкою контакту для всіх дій з керування програмами, включаючи вхід користувачів, DNS, стан програми та загальну оркестровку програм. Клієнти зв'язуються з Брокером через веб-консоль, інструменти CLI або IDE JBoss Tools через API на основі REST. Посередник використовує клієнт MCollective (на основі служби обміну повідомленнями ActiveMQ), щоб надсилати інструкції до вузлів, які фактично розміщують програми користувача

Сервери вузлів розміщують Gears і вбудовані картриджі. Gears — це контейнер, де зберігаються та обслуговуються користувацькі додатки. Gear представляє частину процесора, оперативної пам'яті та основного сховища Node, які доступні для кожної програми. Gears поєднує можливості розділення з функціями безпеки SELinux. Картриджі представляють компоненти, що підключаються, які можна комбінувати в одній програмі, наприклад, картридж мови чи середовища або картридж бази даних



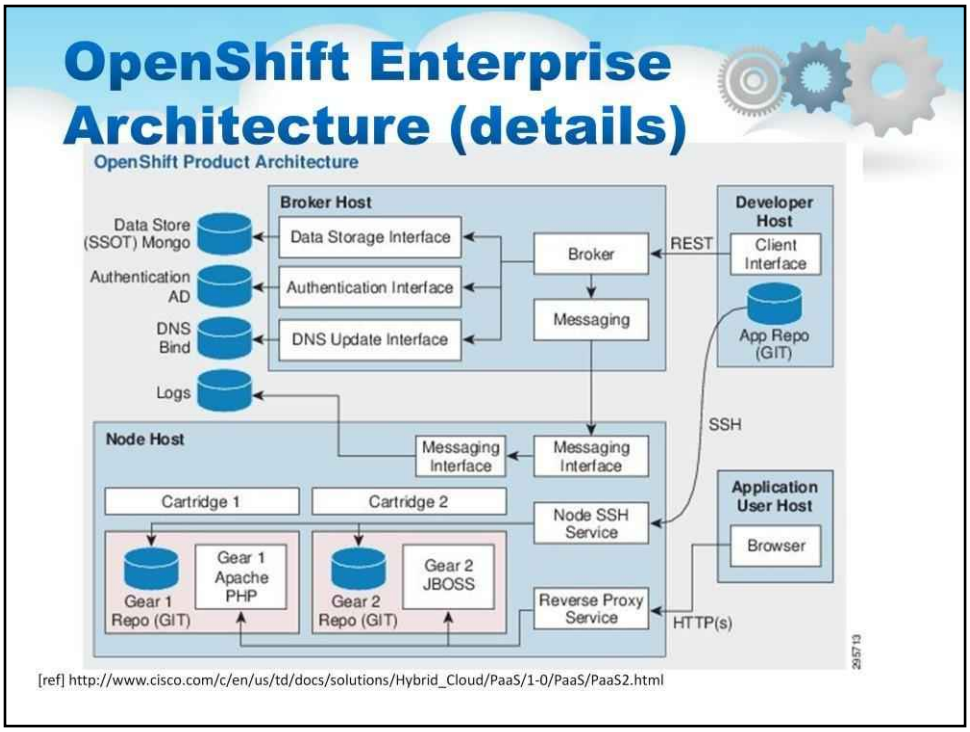
На цьому слайді є детальна ілюстрація архітектури, яка демонструє компоненти OpenShift.

У найпростішій топології OpenShift розділений, як показано, де Origin Broker знаходиться на одному вузлі, а вузол OpenSHift із картриджем — на іншому вузлі.

Однак численні системні топології підтримуються серверами Broker і Node

> Всі компоненти на одному хості
>
> Один брокер + хост ActiveMQ, кілька хостів Node. Брокери із збалансованим навантаженням, автономний хост ActiveMQ, окремі репліковані сервери MongoDB, кілька хостів Node

StickShift — внутрішній API ядра OpenShift; можна було б використовувати Stickshift, щоб додати картриджі до системи.



На цьому слайді показано більш детальну архітектуру середовища розміщення, розробки та керування програмами на основі OpenShift.

Зокрема, він показує клієнт на основі браузера користувача, хост розробника, хост-посередник і компоненти хост-вузла.

Вузол, що один хост вузла (на віртуальній машині) може розмістити багато передач і картриджів,

Хмарні обчислення

Лекційний посібник

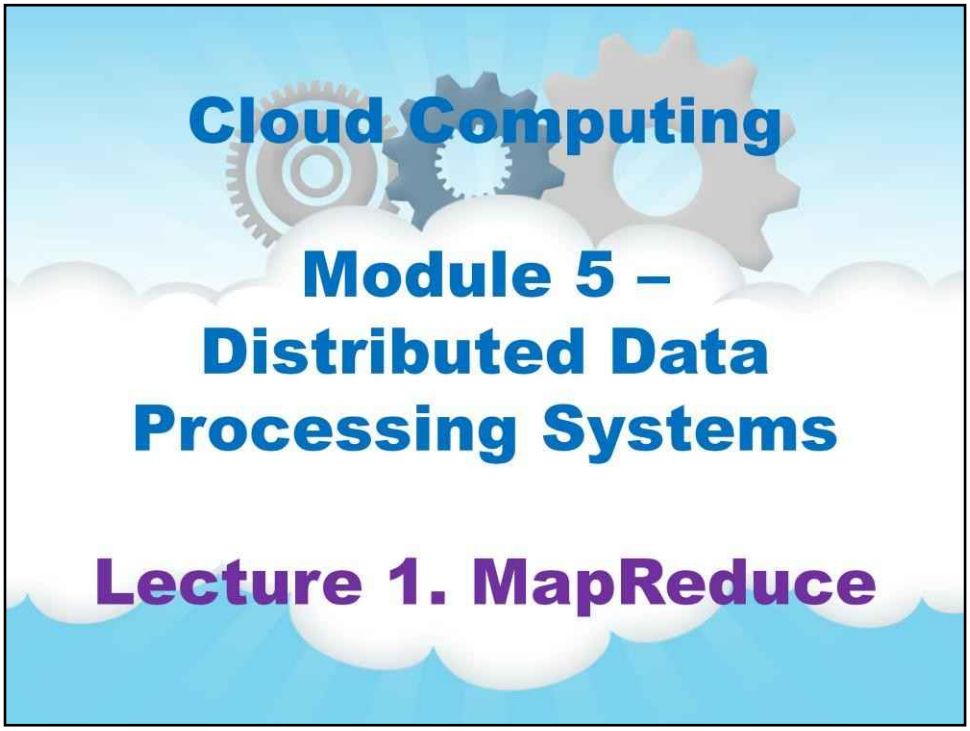
Том 5

Модуль 5

Розподілені системи обробки даних

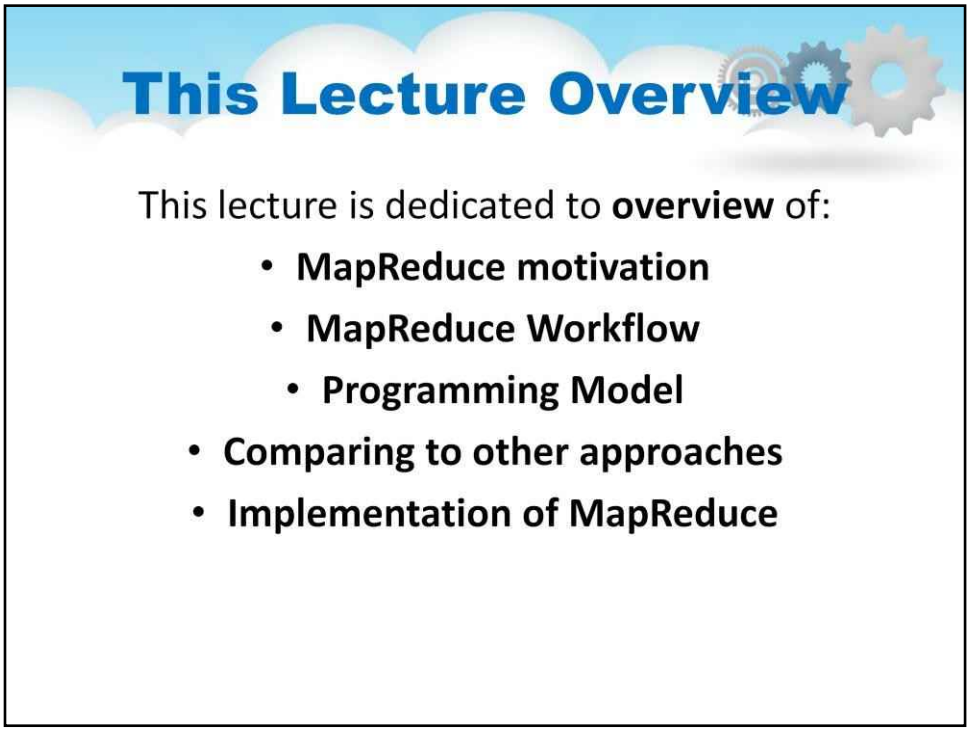
в хмарних обчисленнях

Зміст



У цій лекції ми розглянемо модель розподіленої обробки великих наборів даних під назвою MapReduce.

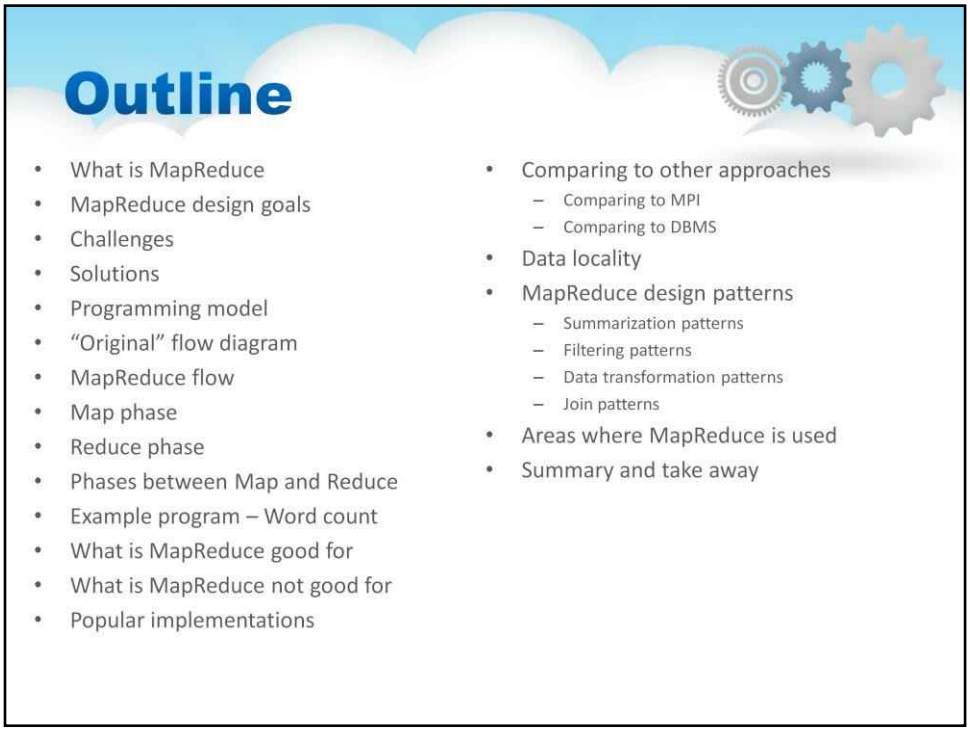
Фреймворки, які реалізують цю модель, забезпечують гарне горизонтальне масштабування та приховують від розробника багато технічних деталей, пов’язаних із розпаралелюванням, плануванням, передачею даних, обробкою помилок і відновленням після апаратних збоїв.



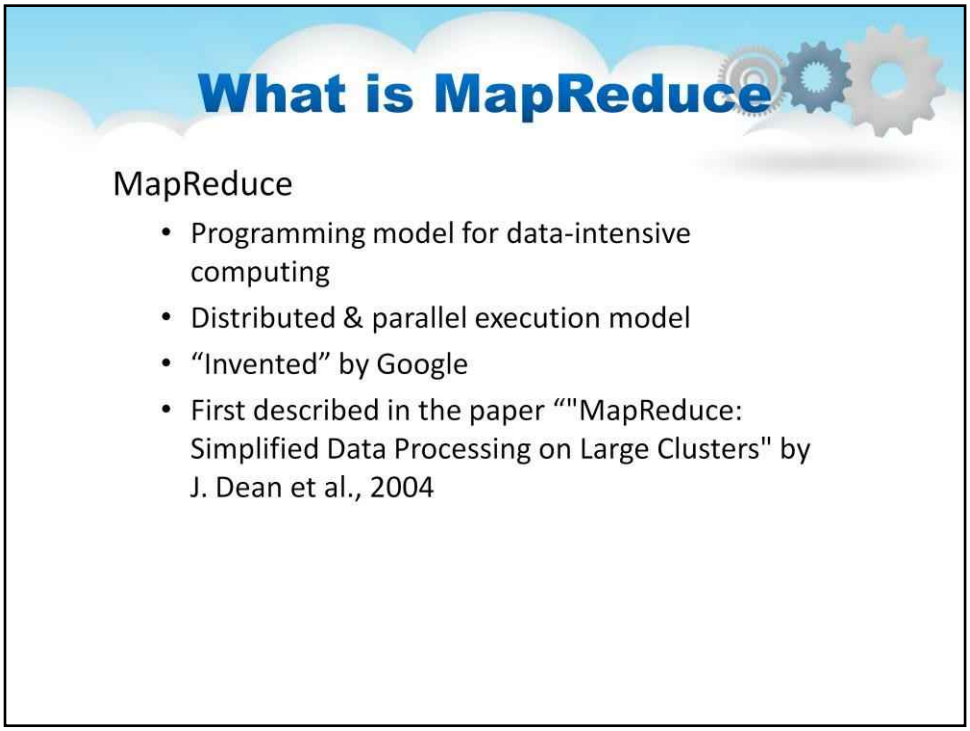
Лекція 1. MapReduce

Ця лекція присвячена огляду:

- Мотивація MapReduce
- Робочий процес MapReduce
- Модель програмування
- MapReduce порівняно з іншими підходами великомасштабної обробки даних
- Найпопулярніші реалізації MapReduce



Що таке MapReduce
Цілі дизайну MapReduce
Виклики
Рішення
Модель програмування
«Оригінальна» схема
потоку MapReduce
Фаза карти
Зменшити фазу
Фази між Map і Reduce
Приклад програми – кількість
слів Чим корисний MapReduce
Чим не підходить MapReduce
Популярні реалізації
Порівняння з іншими підходами
Порівняно з MPI
Порівняння з СУБД
Локальність даних
Шаблони проектування MapReduce
Зразки узагальнення
Шаблони фільтрації
Шаблони перетворення даних
Шаблони з'єднання
Області, де використовується
MapReduce Підсумок і підсумки



Огляд

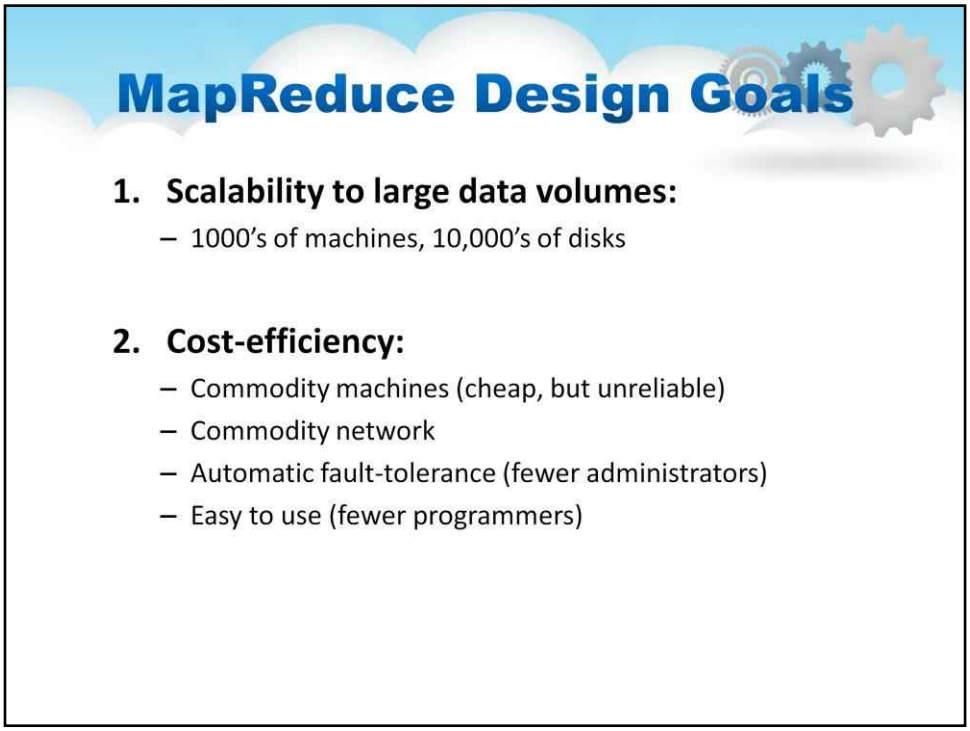
MapReduce — це модель програмування та відповідна реалізація для обробки великих наборів даних.

MapReduce виник із функціонального програмування та був представлений Google у статті під назвою «MapReduce: спрощена обробка даних у великих кластерах».

Одним із перших завдань Google було з’ясувати, як індексувати зростаючий обсяг вмісту в Інтернеті.

Щоб вирішити цю проблему, Google «винайшов» новий стиль обробки даних, відомий як MapReduce, щоб керувати великомасштабною обробкою даних у великих кластерах товарних серверів.

Реалізація MapReduce від Google є запатентованим рішенням і ще не опублікована.



Основними цілями дизайну були:

Спочатку**, масштабованість до великих обсягів даних**.

Мова йде не про кластер з кількох чи кількох десятків комп'ютерів. Йдеться про
тисячі і більше комп'ютерів з дисками.

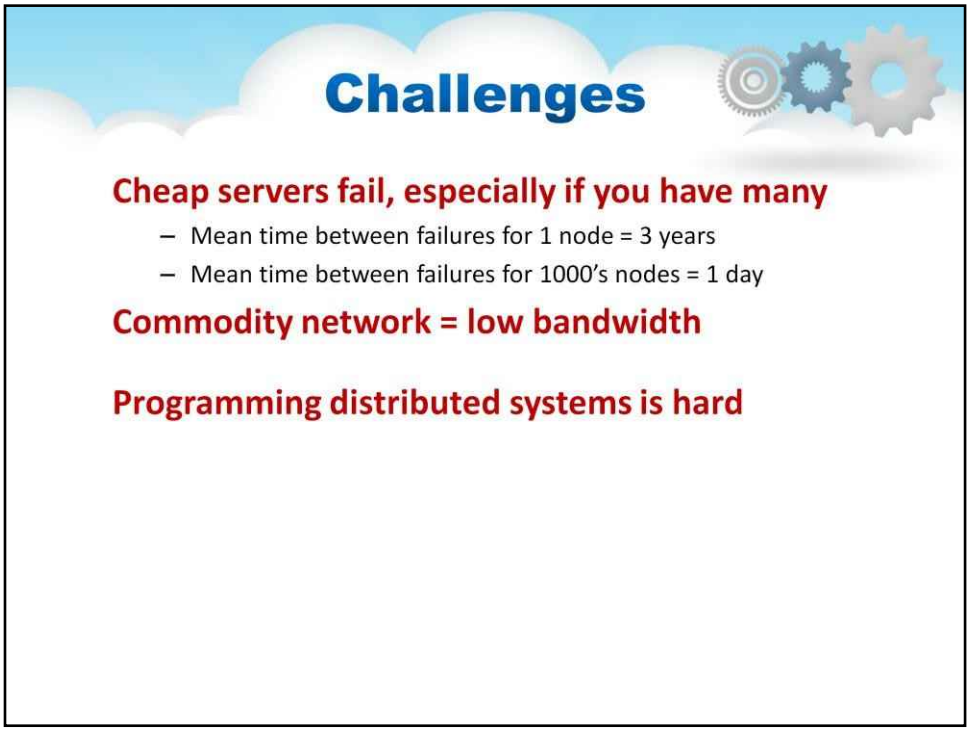
по-друге,**економічність**.

Слід використовувати товарні комп'ютери та мережеве обладнання. Це означає, що апаратні збої є радше правилом, ніж винятком.

Отже, на рівні програмного забезпечення повинні бути вбудовані функції відновлення після відмови та відновлення. Деякі абстракції повинні приховувати від розробника складні технічні деталі, пов'язані з розпаралелюванням і плануванням, міжмашинними зв'язками, обробкою апаратних збоїв.

Ці спеціальні знання не повинні вимагатися.

Це дозволяє програмістам без будь-якого досвіду роботи з паралельними та розподіленими системами легко використовувати ресурси великої розподіленої системи.

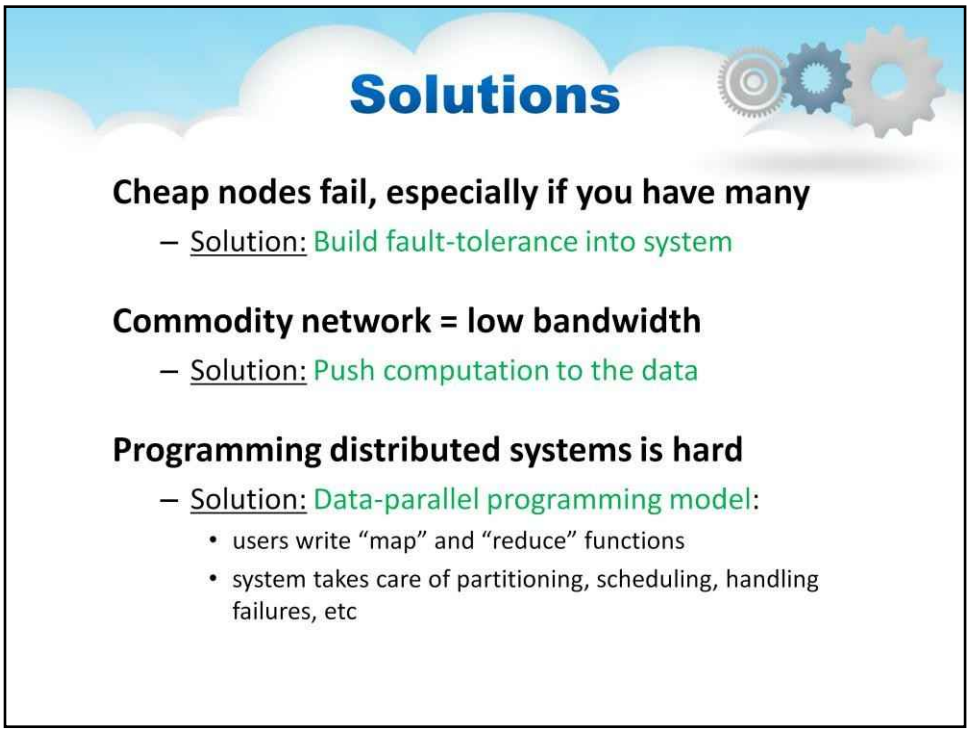


Розглянемо проблеми, з якими можна зіткнутися при розробці такої системи.

Перше завдання – це середній час між невдачами. Для одного вузла це три роки, а для тисячі вузлів - близько одного дня, тому що будь-який з вузлів може вийти з ладу, і вся система вийде з ладу.

Другий виклик – це товарна мережа. Якщо між вузлами немає спеціального високошвидкісного з'єднання, пропускна здатність низька, а затримка висока.
Інтенсивний обмін даними між процесами на різних вузлах знизить загальну продуктивність системи.

По-третє, програмувати розподілені системи важко. Це вимагає специфічних знань.

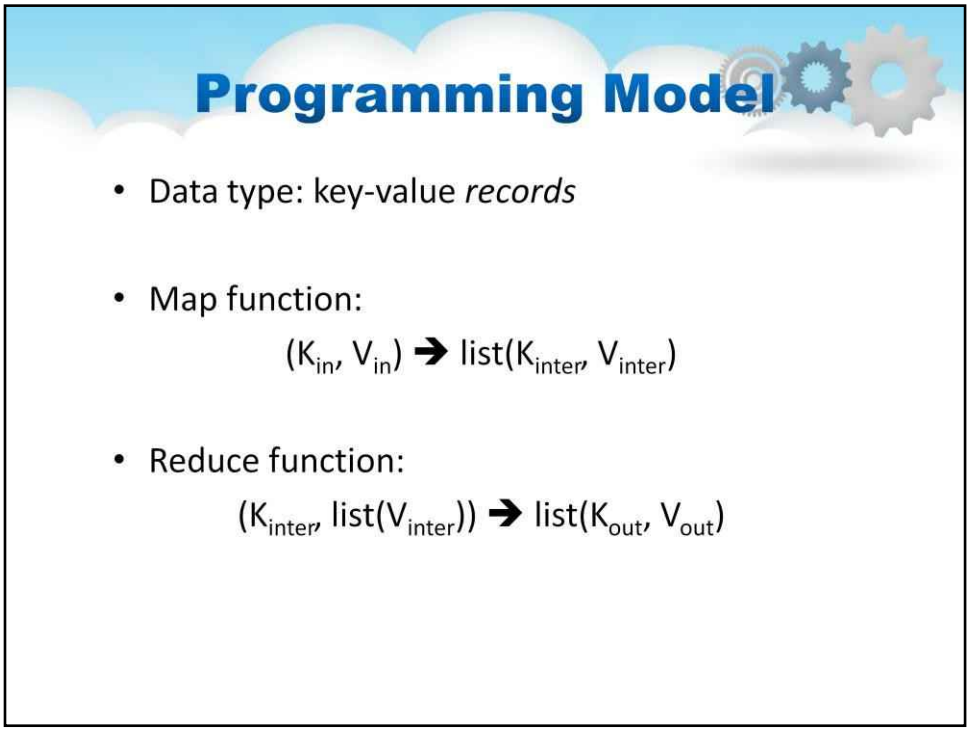


Розглянемо можливі рішення

Перше завдання – це середній час між невдачами. Рішення полягає в тому, щоб створити в системі відмовостійкість.

Друга проблема – це товарна мережа. Рішення полягає в тому, щоб перенести обчислення на дані.

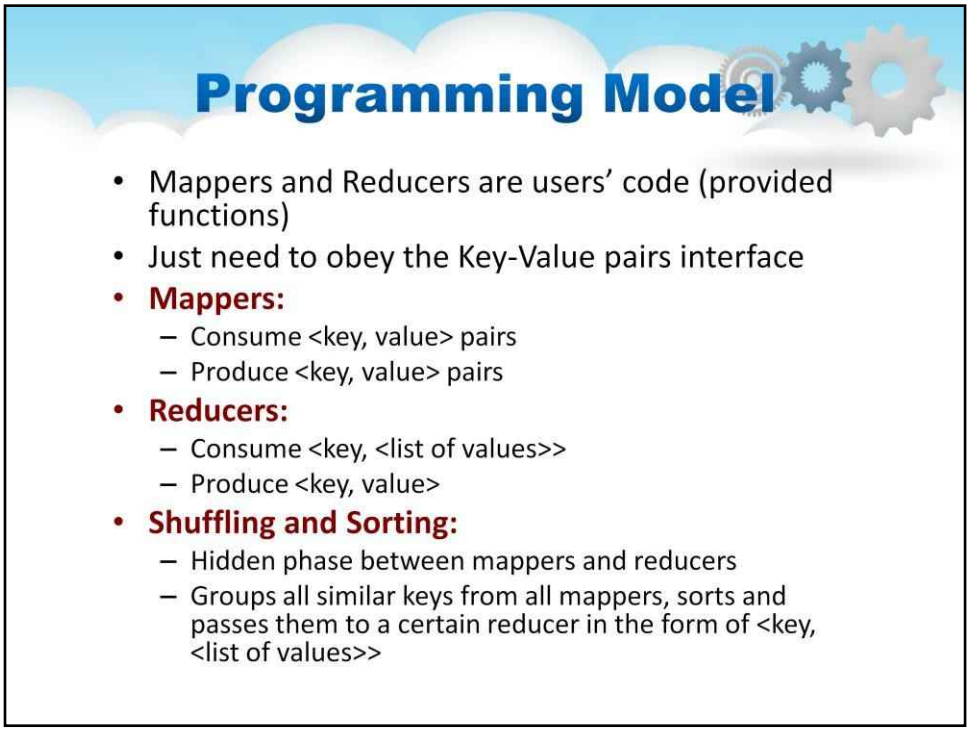
По-третє, програмування розподілених систем ishard.
Рішенням є створення простої моделі програмування. Тут ми маємо лише дві функції - Map і Reduce. Framework подбає про решту.



Модель програмування

У загальному вигляді модель виглядає так:

Картафункція відображає вхідні пари ключ/значення на набір проміжних пар ключ/значення. Карти — це окремі завдання, які перетворюють вхідні записи на проміжні записи. Перетворені проміжні записи не обов'язково мають бути того самого типу, що й вхідні записи. **Зменшити**функція зменшує набір проміжних значень, які мають спільний ключ, до меншого набору значень.



КОНЦЕПЦІЇ ФУНКЦІОНАЛЬНОГО ПРОГРАМУВАННЯ

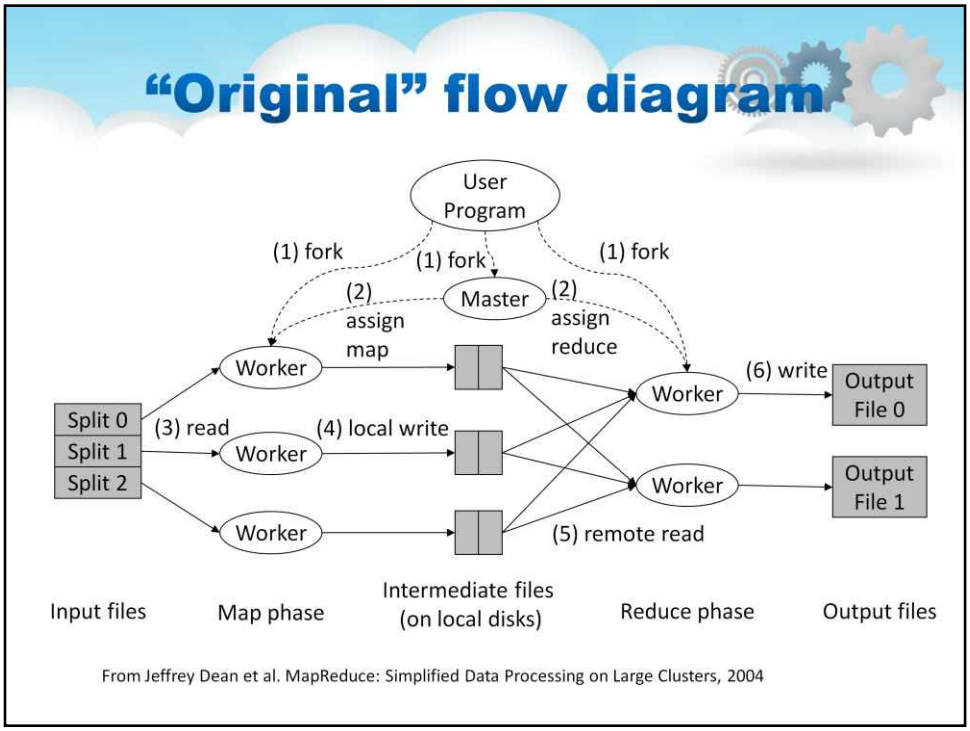
Програми MapReduce призначені для паралельного обчислення великих обсягів даних. Це вимагає розподілу робочого навантаження між великою кількістю машин.

Ця модель не буде масштабована до великих кластерів (сотень або тисяч вузлів), якщо

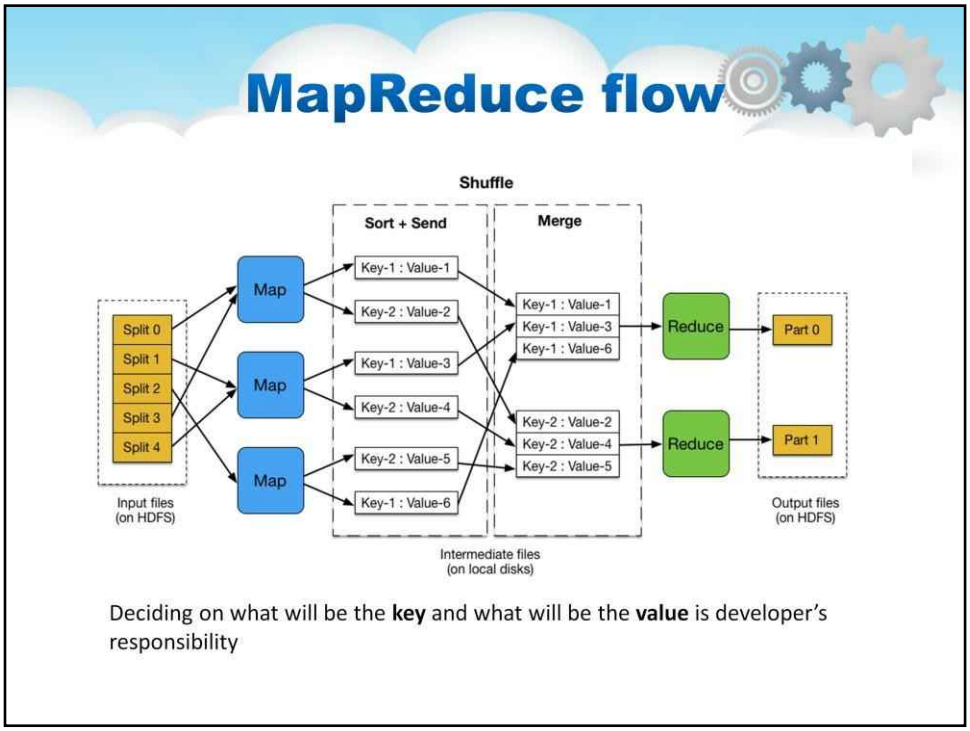
компонентам було дозволено обмінюватися даними довільно.

Накладні витрати на зв'язок, необхідні для постійної синхронізації даних на вузлах, завадять системі працювати надійно чи ефективно у великих масштабах.

Натомість усі елементи даних у MapReduce є**незмінний**, тобто їх неможливо оновити. Якщо в завданні зіставлення ви змінюєте вхідну пару (ключ, значення), це не відображається у вхідних файлах; Зв'язок відбувається лише шляхом генерації нових вихідних пар (ключ, значення), які потім пересилаються системою на наступну фазу виконання.



Це оригінальна блок-схема зі статті «MapReduce: спрощена обробка даних у великих кластерах» Джеффрі Діна, опублікованої в 2014 році.

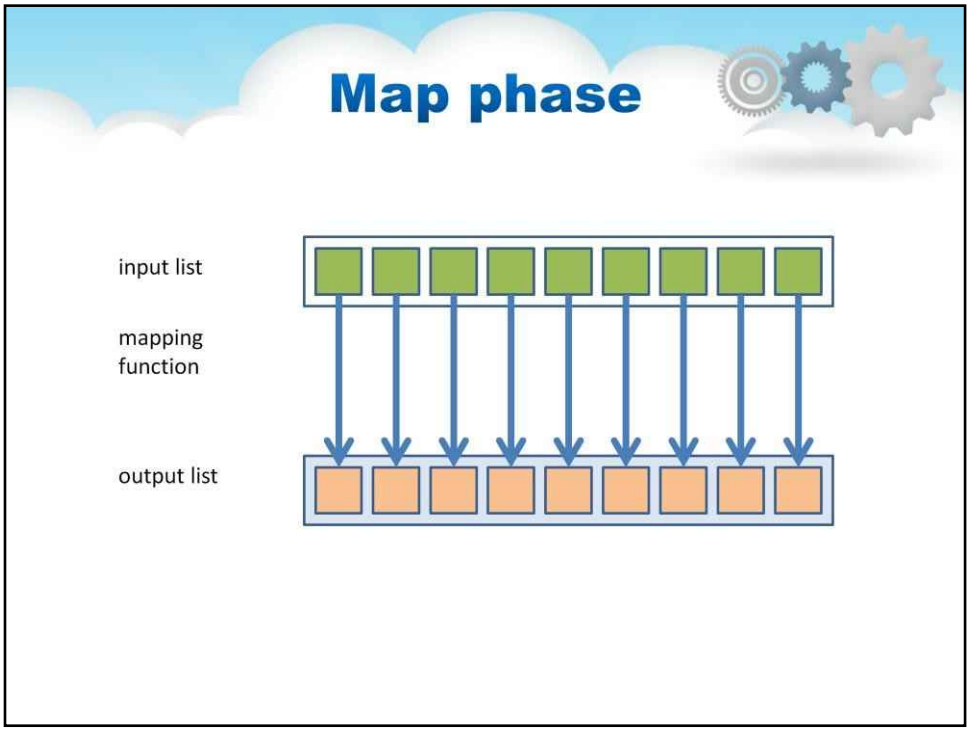


Концептуально програми MapReduce перетворюють списки елементів вхідних даних у списки елементів вихідних даних.

Програма MapReduce зробить це двічі, використовуючи дві різні обробки списку

ідіоми: **карта** і **зменшити**. Ці терміни взято зі списку таких мов обробки

як LISP.

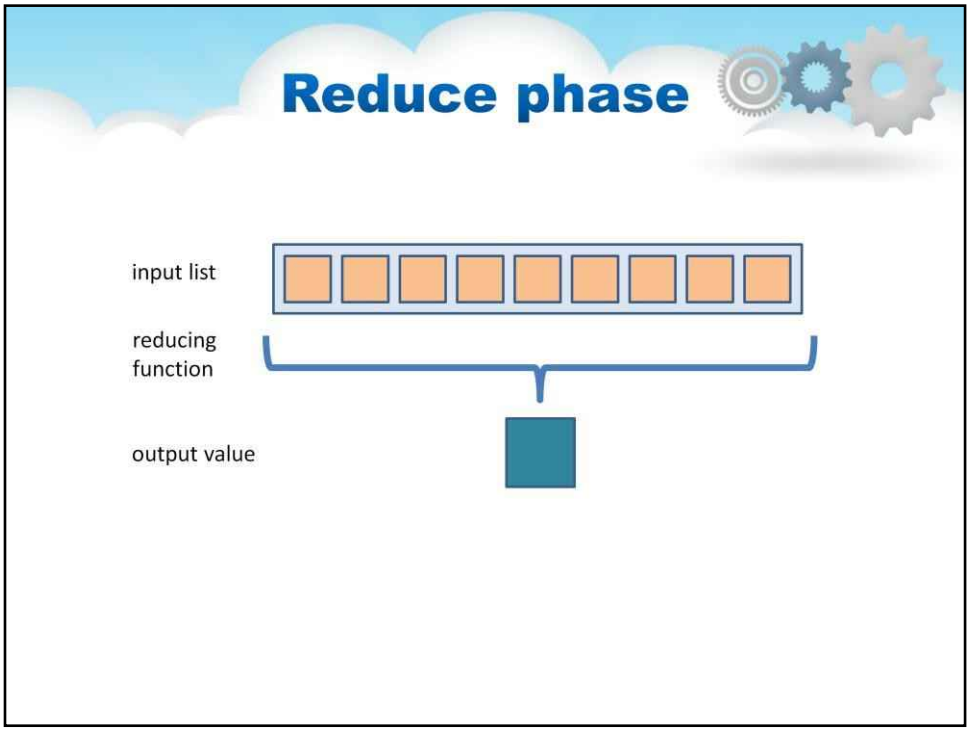


Викликається перший етап програми MapReduce. **відображення** Список елементів даних надається по одному для функції під назвою **Картограф**, який перетворює кожен елемент окремо в елемент вихідних даних. Відображення створює новий вихідний список шляхом застосування функції до окремих елементів

вхідний список.

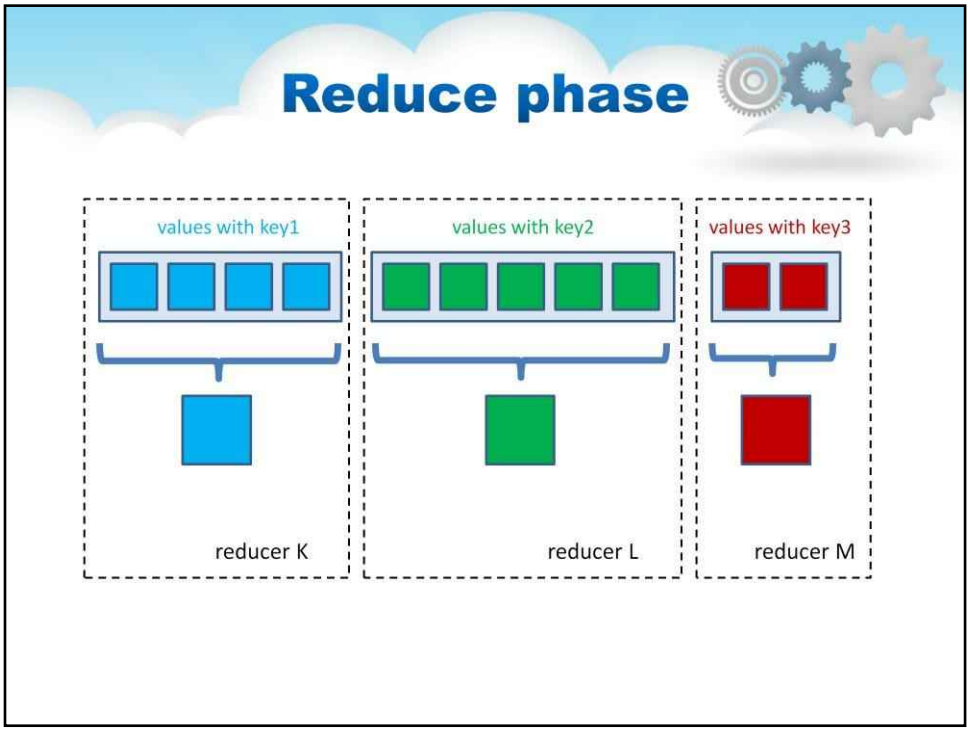
Як приклад корисності map: припустімо, що у вас є функція toUpper(str), яка повертає версію свого вхідного рядка у верхньому регістрі. Ви можете скористатися цією функцією

з map, щоб перетворити список рядків на список рядків у верхньому регістрі. Зауважте, що тут ми не змінюємо вхідний рядок: ми повертаємо новий рядок, який буде частиною нового вихідного списку.



Зменшення дозволяє агрегувати значення разом. Â **редукторі** функція отримує ітератор вхідних значень зі списку вхідних даних. Потім він поєднує ці значення разом, повертаючи одне вихідне значення.

Зменшення часто використовується для отримання «зведених» даних, перетворюючи великий обсяг даних на менший підсумок. Наприклад, «+» можна використовувати як функцію скорочення, щоб повернути суму списку вхідних значень.



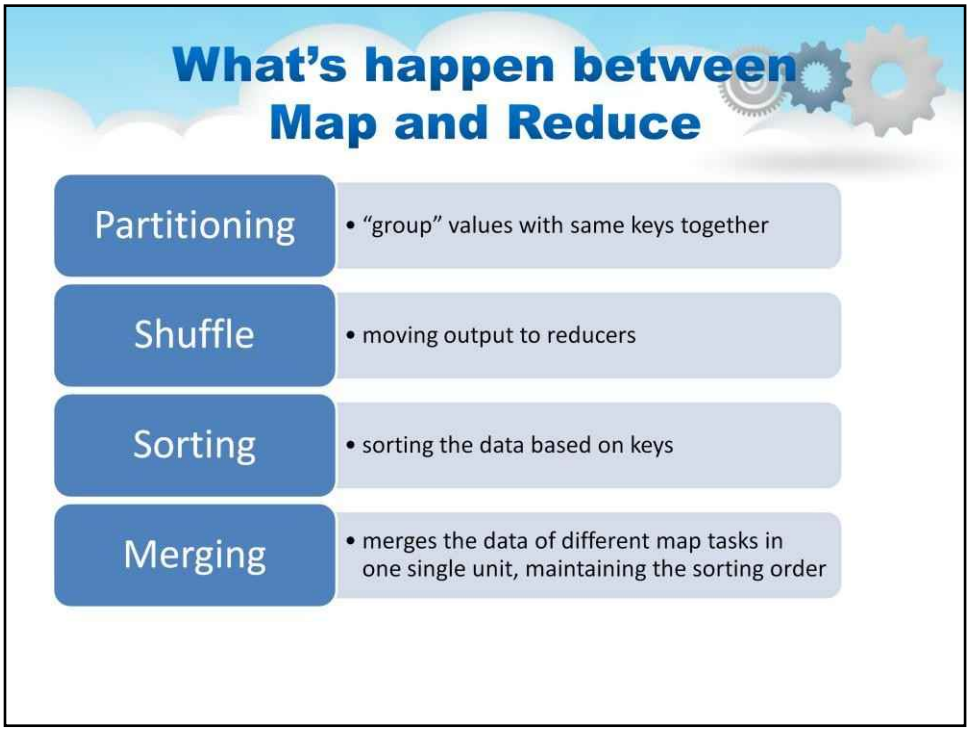
Фреймворк MapReduce бере ці концепції та використовує їх для обробки великих обсягів інформації. Програма MapReduce складається з двох компонентів: один реалізує відображувач, а інший реалізує редюсер. Ідіоми Mapper і Reducer, описані вище, трохи розширені для роботи в цьому середовищі, але основні принципи ті самі.

Ключі та значення: У MapReduce жодне значення не стоїть окремо. Кожне значення має а **ключ** ов'язані з ним. Ключі ідентифікують пов’язані значення.

Функції відображення та скорочення отримують не просто значення, а пари (ключ, значення). Результат кожної з цих функцій однаковий: і ключ, і значення мають бути передані до наступного списку в потоці даних.

Ключі поділяють зменшений простір: Зменшувальна функція перетворює великий список значень на одне (або кілька) вихідних значень. У MapReduce усі вихідні значення зазвичай не зменшуються разом. Усі цінності з тим же ключем представлені одному редуктору разом. Це виконується незалежно від будь-яких операцій скорочення, що відбуваються з іншими списками значень з різними приєднаними ключами.

На схемі різні кольори позначають різні клавіші. Усі значення з однаковим ключем представлені в одному завданні скорочення.



Розбиття та перемішування:Після завершення перших завдань карти кожен з вузлів може все ще виконувати ще кілька завдань карти.

Але вони також починають обмінюватися проміжними виходами із завдань карти туди, де вони потрібні редукторам.

Цей процес переміщення виходів карти до редукторів відомий як **перетасування**

Для кожного скорочувального вузла призначається різна підмножина проміжного ключового простору; ці підмножини (відомі як «розділи») є входами для завдань скорочення.

Кожне завдання карти може видавати пари (ключ, значення) будь-якому розділу; усі значення для одного ключа завжди зводяться разом незалежно від того, який картограф є його джерелом.

Таким чином, всі вузли карти повинні домовитися про те, куди надсилати різні фрагменти проміжних даних.

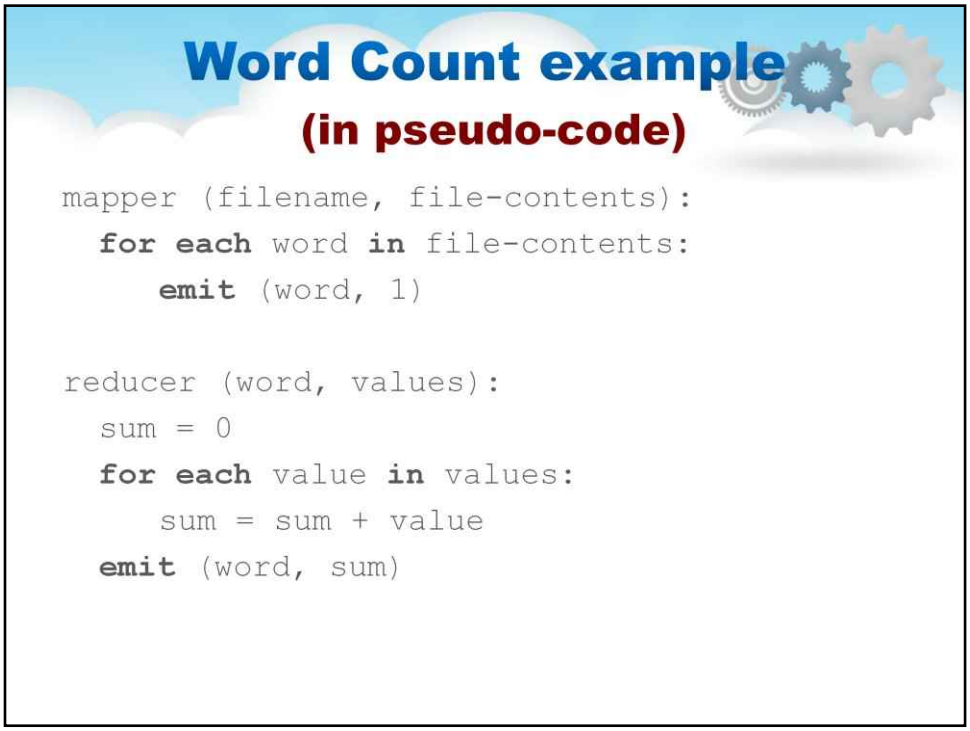
Розділювач визначає, до якого розділу піде задана пара (ключ, значення). Розділ за замовчуванням обчислює хеш-значення для ключа та призначає розділ на основі цього результату.

Сортування

Це просто сортування даних на основі ключів

Об'єднання:

Це відбувається на стороні редуктора. Reducer може отримувати дані з кількох завдань карти та шляхом об'єднання об'єднує дані різних завдань карти в один блок, зберігаючи порядок сортування.



приклад

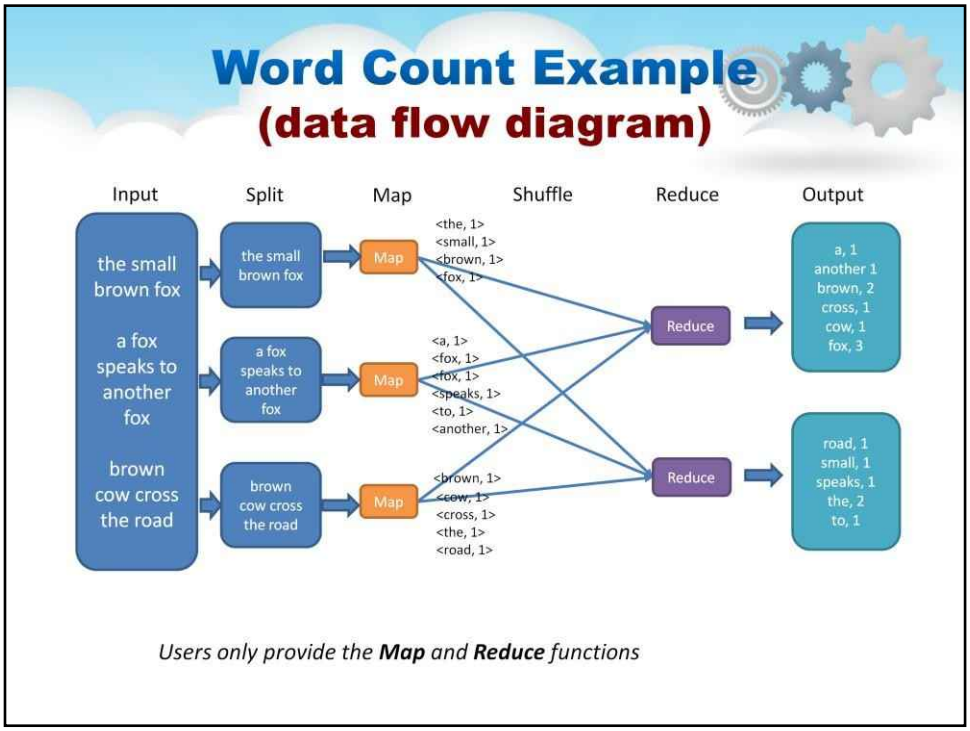
Тепер подивіться на приклад програми в псевдокоді. Він підраховує повторення слів у тексті.

Кілька випадків**картограф**створюються на різних машинах у нашому кластері. Кожен екземпляр отримує інший вхідний файл (передбачається, що таких файлів багато).

Відповідачі виводять (слово, 1) пари, які потім пересилають редукторам. Кілька екземплярів методу редуктора також створено на різних машинах.

Кожен редуктор відповідає за обробку списку значень, пов’язаних з іншим словом.

Список значень буде складатися з одиниць; редуктор підсумовує ці одиниці в остаточну кількість, пов’язану з одним словом. Потім редуктор видає остаточний (слово, кількість) вихід, який записується у вихідний файл.



Давайте розберемося з кожним із етапів, зображених на схемі вище. **введення:**Це вхідні дані для обробки.

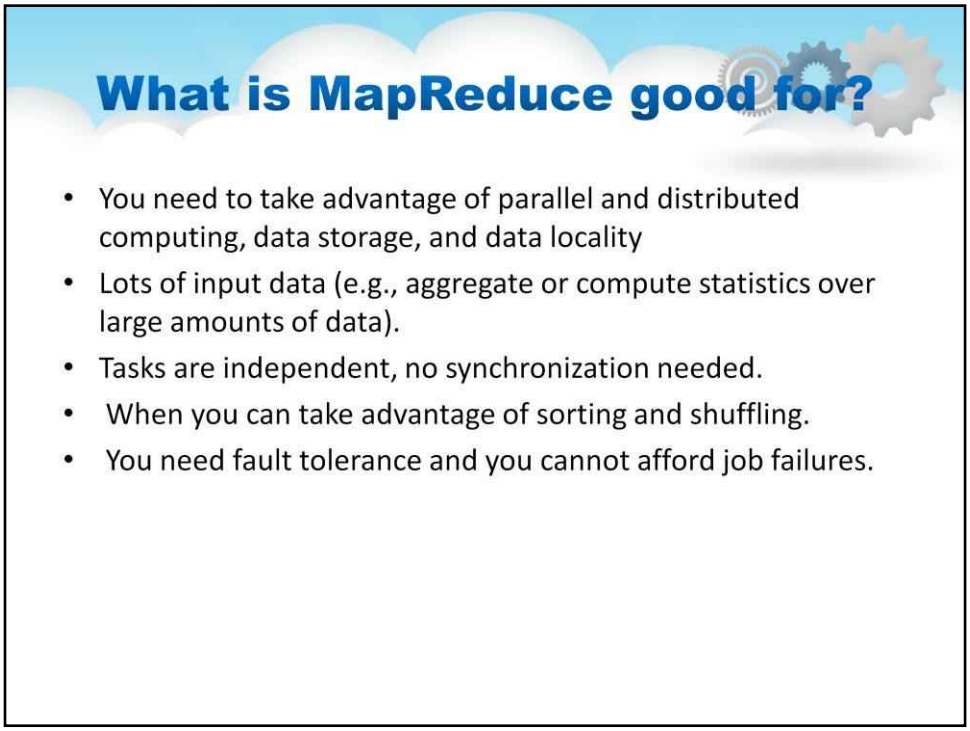
Розділити: Framework розбиває вхідні дані на менші частини, які називаються «сплітами».

Карта:На цьому кроці MapReduce обробляє кожне розділення відповідно до логіки, визначеної у функції map(). Один картограф працює над одним розбиттям за раз. Кожен картограф розглядається як завдання.

Перемішати та сортувати:На цьому кроці результати всіх картографів перемішуються, сортуються, щоб упорядкувати їх, і групуються перед тим, як відправити їх до наступного кроку.

Зменшити:Цей крок використовується для агрегування результатів картографів за допомогою функції reduce(). Вихід редуктора надсилається на наступний і останній крок. Кожен редуктор розглядається як завдання.

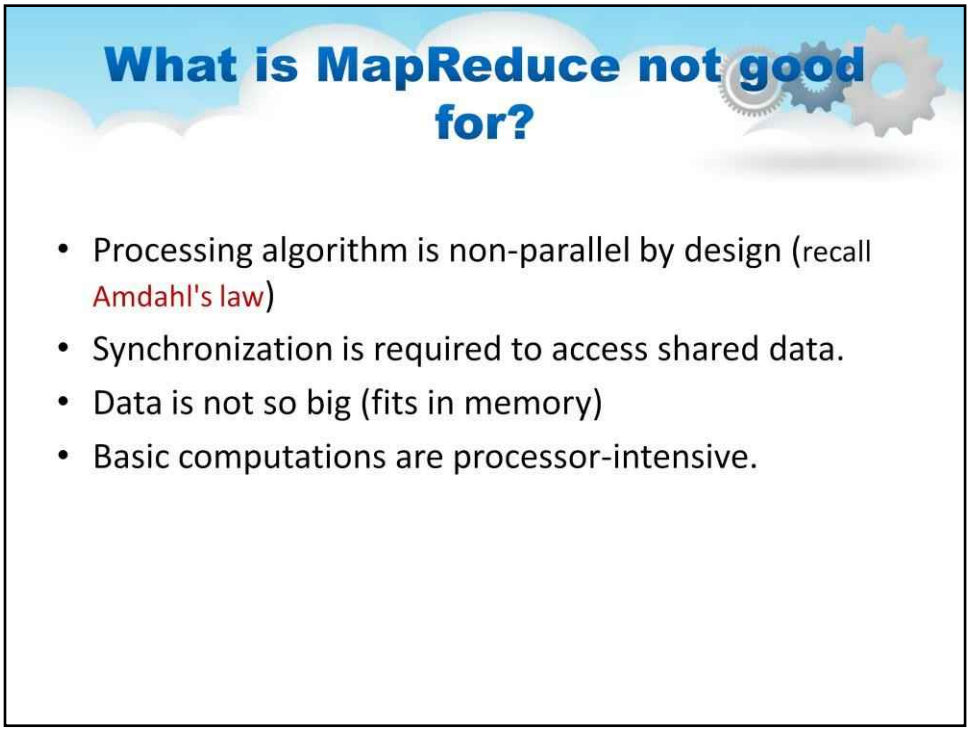
Вихід:Нарешті результат кроку зменшення записується у файл.



Використання

Є багато випадків, коли MapReduce підходить, наприклад:

- Коли вам потрібно обробляти велику кількість вхідних даних (наприклад, агрегувати чи обчислювати статистику для великих обсягів даних).
- Коли вам потрібно скористатися перевагами паралельних і розподілених обчислень, даних зберігання та локальність даних.
- Коли ви можете виконувати багато завдань самостійно без синхронізації.
- Коли ви можете скористатися перевагами сортування та перемішування.
- Коли вам потрібна відмовостійкість і ви не можете дозволити собі невдач.

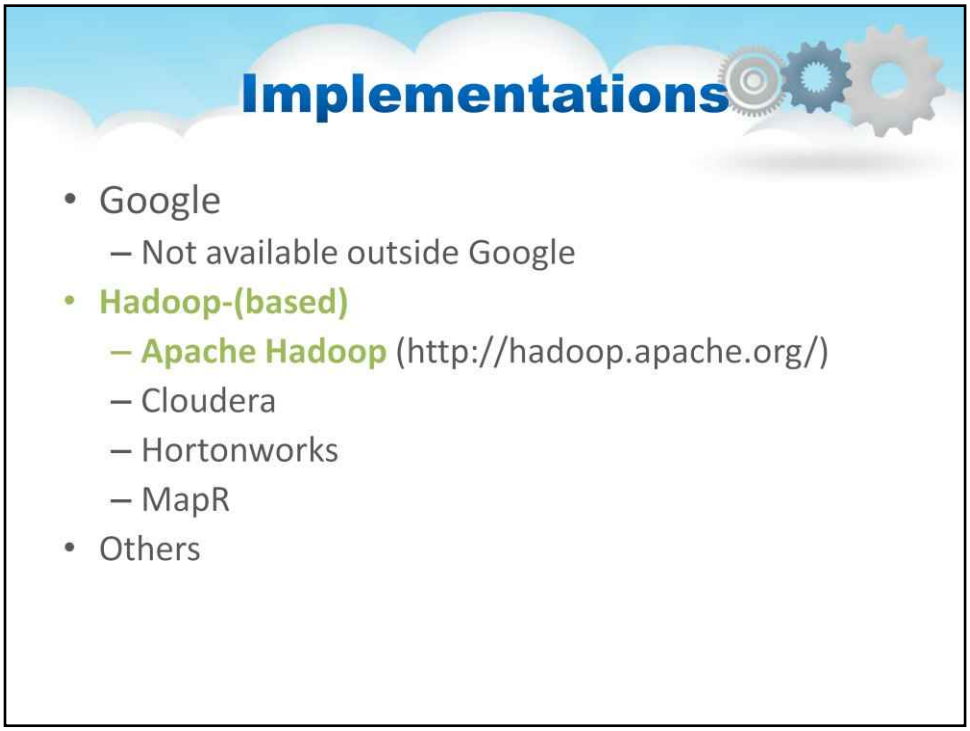


Чи MapReduce підходить для всього? Проста відповідь - ні. Коли у нас є великі дані, якщо ми можемо розділити їх і кожен розділ можна обробляти незалежно, тоді ми можемо почати думати про алгоритми MapReduce. Але ось інші сценарії, коли MapReduce не слід використовувати:

- Якщо обчислення значення залежить від попередньо обчислених значень. Хорошим прикладом є ряд Фібоначчі, де кожне значення є сумою двох попередніх значень:

$F(k + 2) = F(k + 1) + F(k)$

- Якщо набір даних достатньо малий для обчислення на одній машині.
- Якщо для доступу до спільних даних потрібна синхронізація.
- Якщо всі введені дані поміщаються в пам’яті.
- Якщо одна операція залежить від інших операцій.
- Якщо базові обчислення потребують процесора.



Реалізація MapReduce від Google недоступна за межами Google. Інші відомі реалізації засновані на Apache Hadoop.

Клаудера, заснована в 2008 році, була першою компанією, яка розробляла та розповсюджувала програмне забезпечення на основі Apache Hadoop, і досі має найбільшу базу користувачів із найбільшою кількістю клієнтів.

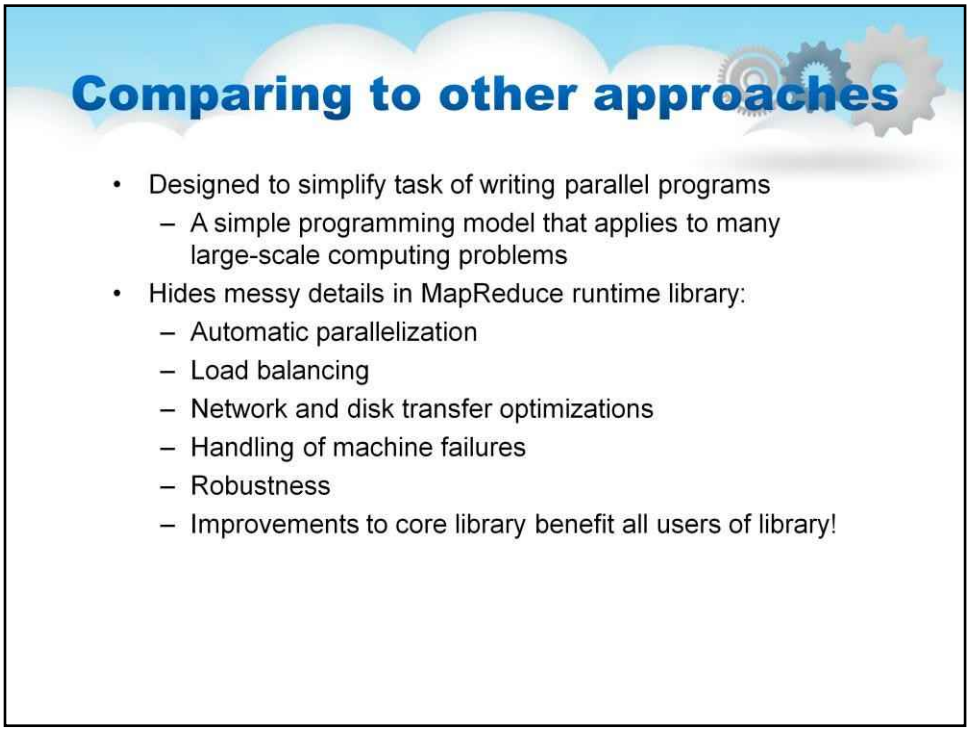
Хоча ядро дистрибутива базується на Apache Hadoop, він також надає власні сервіси та інструменти.

Hortonworks, заснований у 2011 році, швидко став одним із провідних постачальників Hadoop.

Дистрибутив надає платформу з відкритим кодом на основі Apache Hadoop для аналізу, зберігання та керування великими даними.

Hortonworks є єдиним комерційним постачальником, який розповсюджує Apache Hadoop з відкритим вихідним кодом без додаткового пропрієтарного програмного забезпечення.

MapRзамінює компонент HDFS (буде описано пізніше) і натомість використовує власну власну файлову систему під назвою MapR FS.

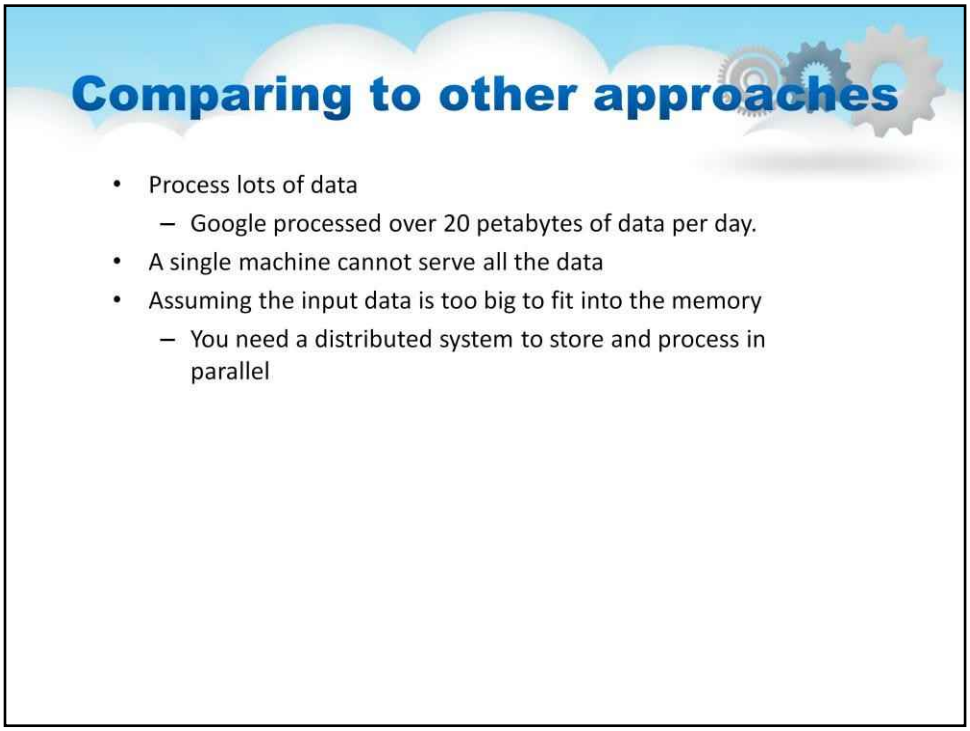


Порівняння

Натхненна функціональним програмуванням обчислювальна парадигма MapReduce проста й зрозуміла.

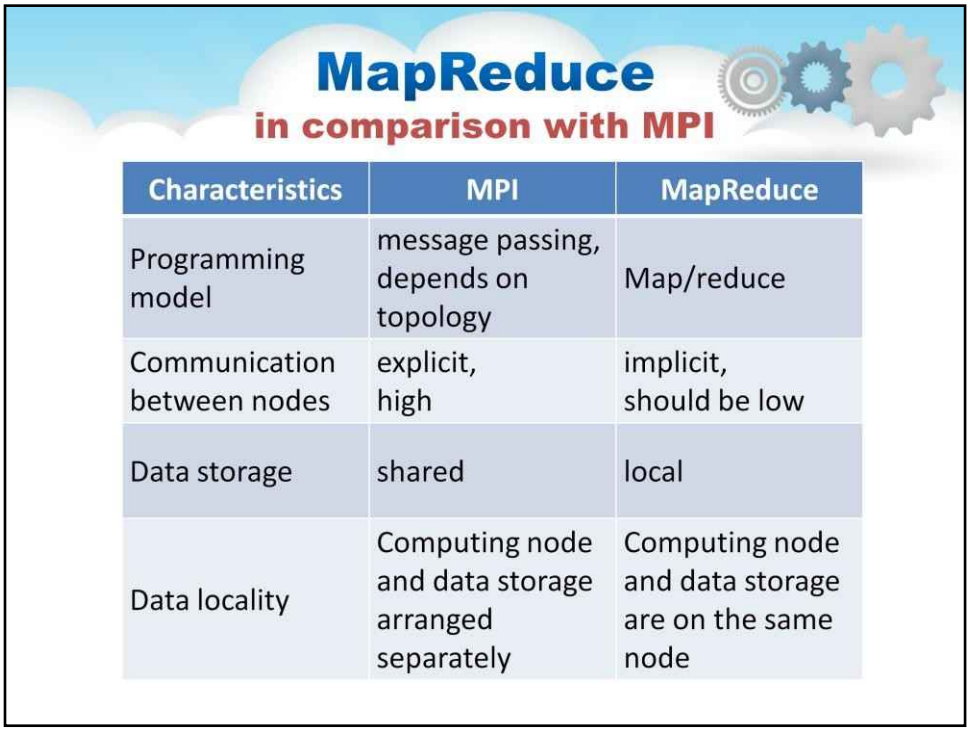
Завдяки автоматичному розпаралелюванню, відсутності явної обробки передачі даних і синхронізації в програмах і відсутності взаємоблокувань ця модель дуже приваблива. Тим часом цього достатньо для простих обчислень, згаданих раніше. Навпаки, крива навчання набагато більш комплексного MPI є крутою.

Відмовостійкість. Оскільки апаратні збої є поширеними у великих кластерах звичайних ПК, MapReduce приділяє велику увагу відмовостійкості. У MPI головним завданням розробників програм є створення відмовостійкості програми.



Припускаючи, що вхідні дані занадто великі, щоб поміститися в пам’ять (об’єднані з усіх вузлів), MapReduce використовує модель потоку даних, яка також забезпечує простий інтерфейс вводу/виводу для доступу до великої кількості даних у розподіленій файловій системі.

Він також використовує локальність даних для ефективності. У більшості випадків нам взагалі не потрібно турбуватися про введення-виведення. Хоча це звучить тривіально, насправді це важливе вдосконалення для аналітики великих даних.



Characteristics	MPI	MapReduce
Programming model	message passing, depends on topology	Map/reduce
Communication between nodes	explicit, high	implicit, should be low
Data storage	shared	local
Data locality	Computing node and data storage arranged separately	Computing node and data storage are on the same node

MapReduce і MPI

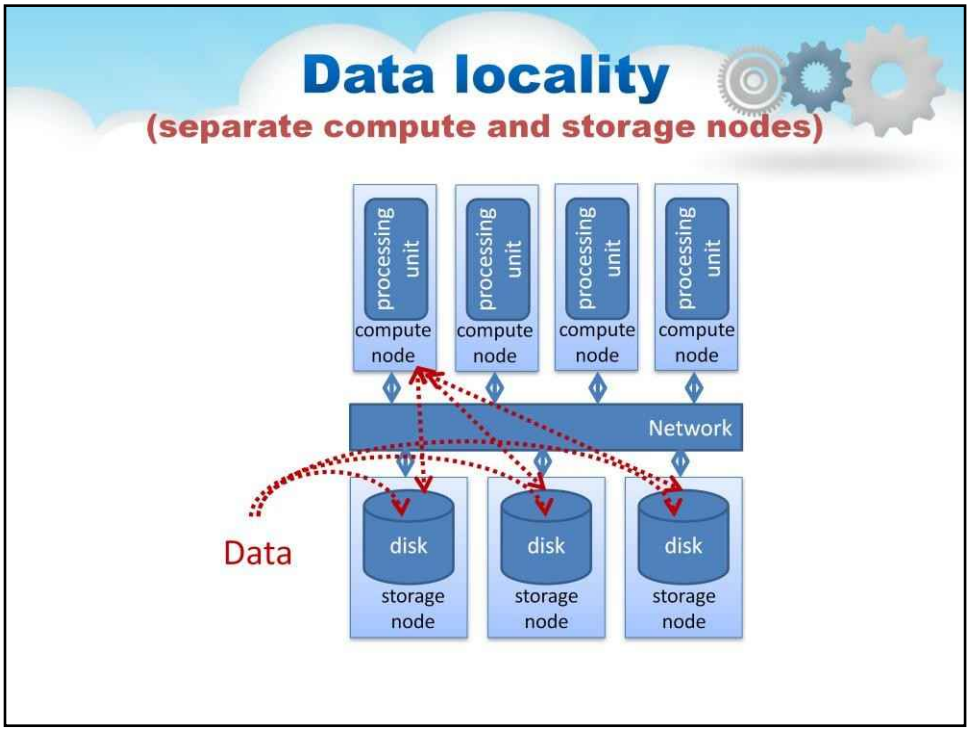
Розглянемо деякі відмінності між MPI і MapReduce.

У моделі програмування MPI обчислення складається з одного або кількох процесів, які спілкуються, викликаючи підпрограми бібліотеки для надсилання та отримання повідомлень іншим процесам.

Модель MapReduce, заснована на двох функціях – map і reduce. MPI дозволяє явний зв'язок між процесами під час виконання. Розробник вирішує, коли і куди надсилати повідомлення.

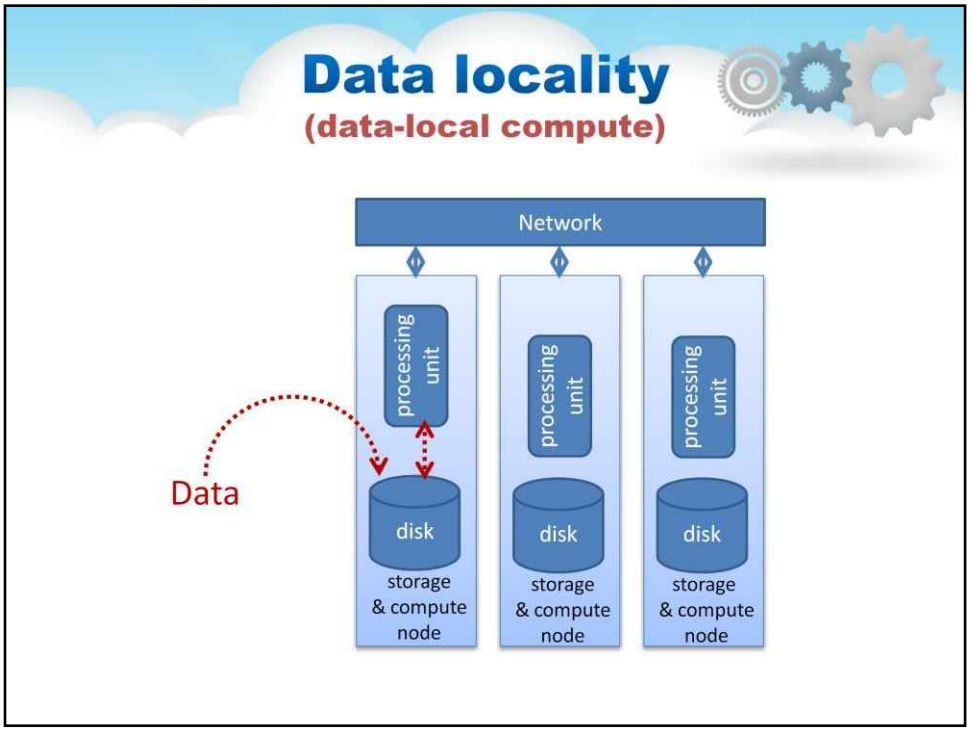
У MapReduce комунікація неявна. Передбачається, що обмін даними відбувається лише на певних етапах виконання.

У MPI зазвичай усі вузли кластера підключаються до спільного сховища. Обчислювальний вузол і сховище даних є різними вузлами. Навпаки, у MapReduce зазвичай обчислювальні вузли та вузли зберігання однакові.



У багатьох традиційних системах обробки даних зберігання (де дані перебувають у стані спокою) і обчислення (де дані обробляються) зберігаються окремо.

У цих системах дані будуть переміщуватися через дуже швидку мережу до комп’ютерів, які потім їх оброблятимуть. Переміщення даних по мережі дуже дороге з точки зору часу.



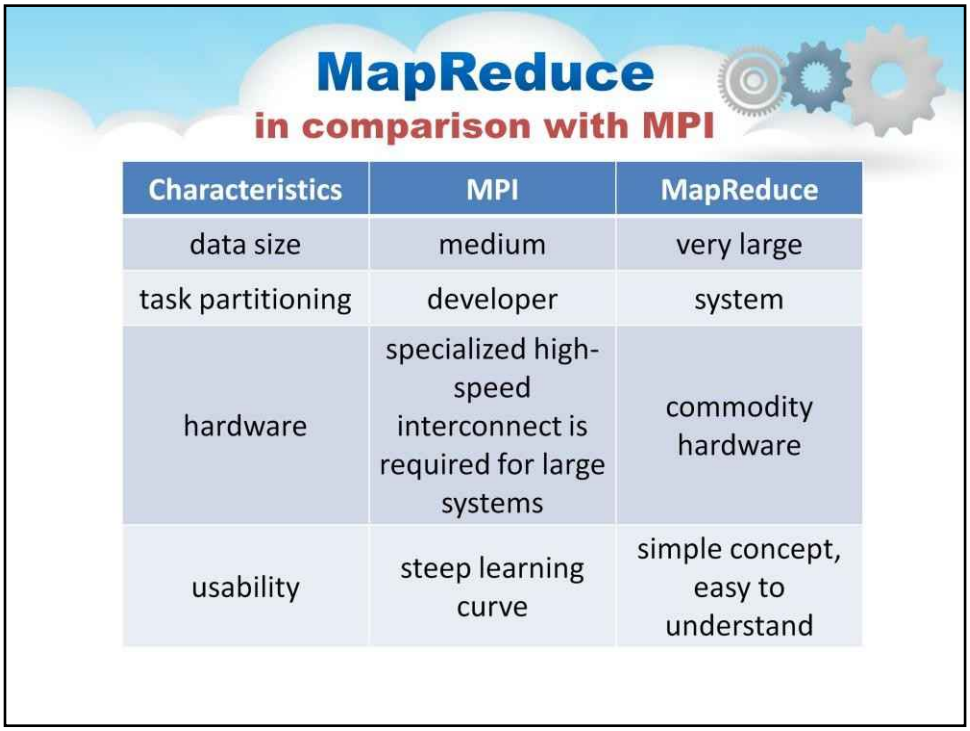
Локальність даних здається дійсно простою ідеєю, але концепція не переміщення даних через мережу та їх обробки на місці насправді є досить новою.

Фреймворки MapReduce були одними з перших універсальних і широко доступних

системи, які застосували цей підхід.

Виявляється, передача інструкцій для обчислення (тобто скомпільована програма, що містить відносно невеликий код для функцій відображення та зменшення) через мережу відбувається набагато швидше, ніж передача петабайта даних туди, де знаходиться програма.

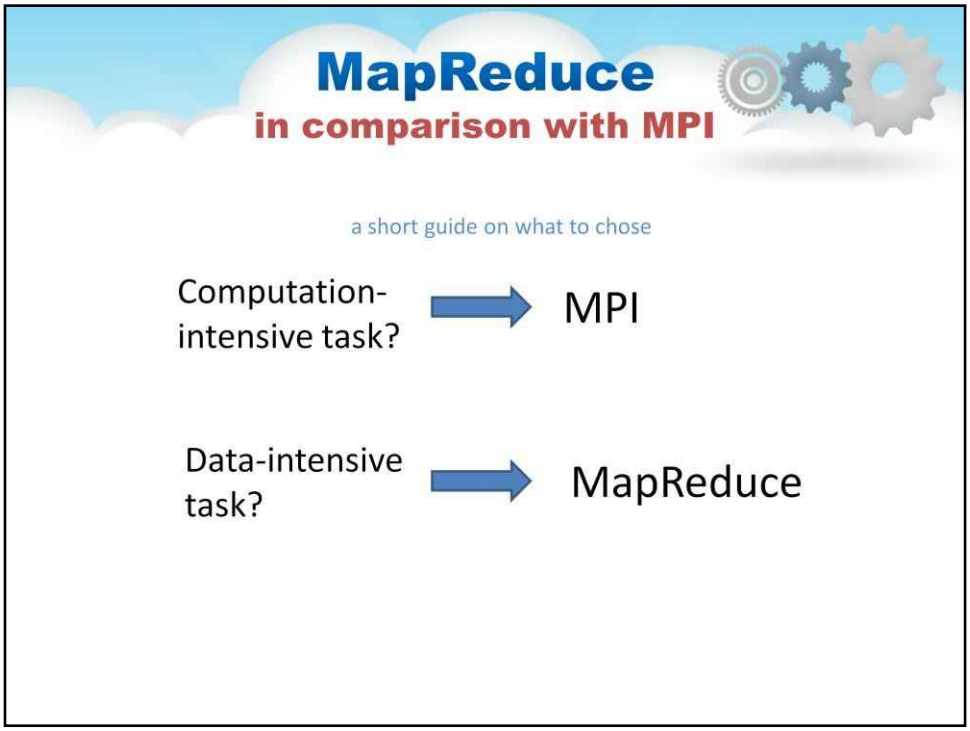
Локальність даних принципово усуває мережу як вузьке місце для лінійної масштабованості.



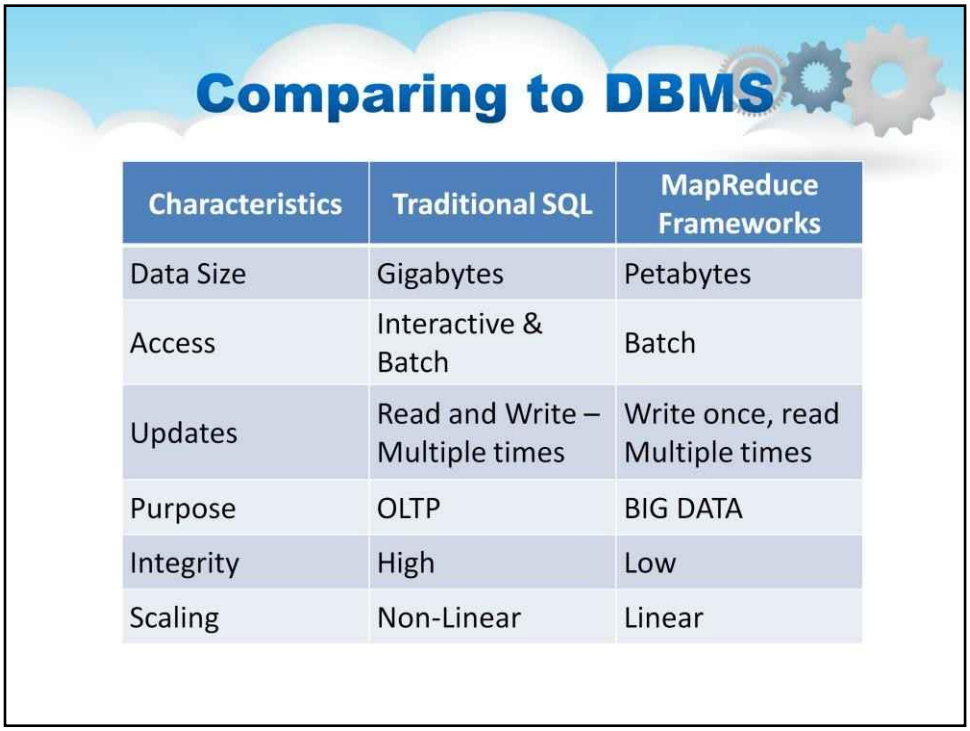
Characteristics	MPI	MapReduce
data size	medium	very large
task partitioning	developer	system
hardware	specialized high-speed interconnect is required for large systems	commodity hardware
usability	steep learning curve	simple concept, easy to understand

Зазвичай MapReduce використовується для обробки величезних обсягів даних. У MPI розділення завдань є відповідальністю розробника. У MapReduce розділення відбувається автоматично.

Масштабний кластер MPI не працюватиме добре без високошвидкісного з’єднання. Map-reduce легше освоїти, тоді як MPI є значно складнішим із великою кількістю функцій.



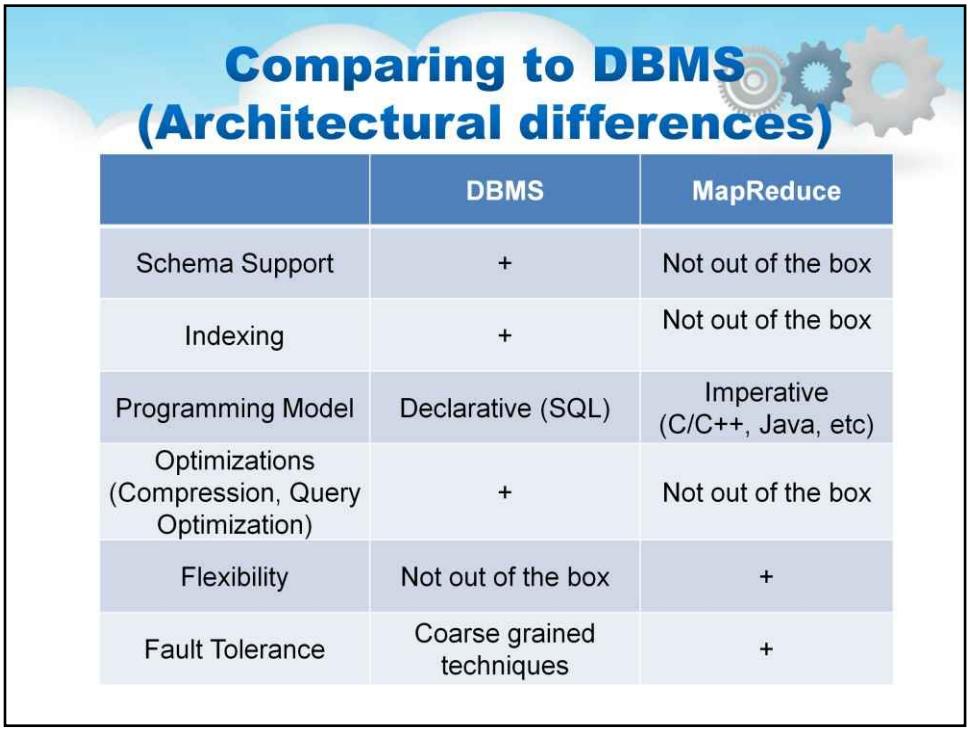
Підсумовуючи вищесказане, map-reduce більше підходить для завдань, що потребують інтенсивних даних, тоді як MPI більше підходить для завдань, що потребують інтенсивних обчислень.



MapReduce і СУБД

Перша відмінність - це розмір даних. Системи керування базами даних зберігають і обробляють гігабайти даних, тоді як MapReduce має справу з петабайтами.

По-друге, традиційні системи керування базами даних дозволяють як інтерактивні, так і пакетні дії
доступ, а фреймворк MapReduce – лише пакетний доступ.
Запис, читання та зміна одних і тих же даних кількох типів є звичайними діями для СУБД, тоді як MapReduce слідує принципу WORM (записати один раз, прочитати багато разів).
Далі, СУБД підходить для OLTP (обробка онлайн-транзакцій), а фреймворки MapReduce підходять для BigData.
У СУБД цілісність даних висока. Згадайте ACID (атомарність, консистенція, ізоляція та довговічність).
Масштабування є перевагою MapReduce. В ідеальному випадку фреймворк MapReduce забезпечує лінійне масштабування.

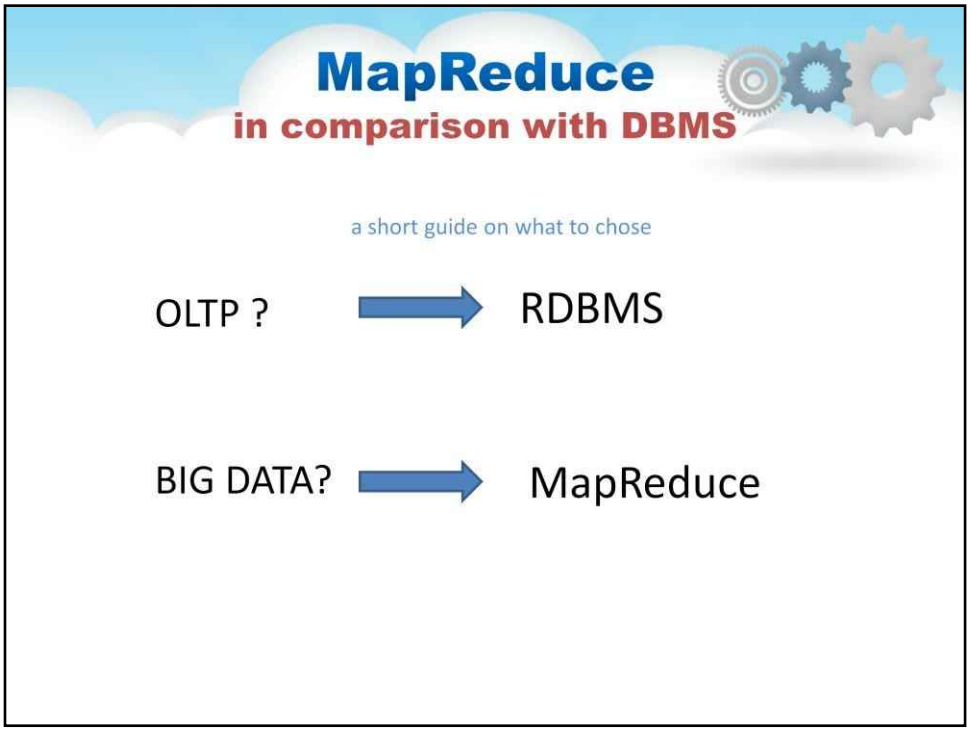


Schema-on -rite був стандартом протягом багатьох років у реляційних базах даних. Перед записом будь-яких даних у базу даних структура цих даних чітко визначається.

Для MapReduce схема не визначена під час зберігання даних. Коли хтось готовий

щоб використовувати ці дані, вони визначають, які частини є важливими для їхньої мети. СУБД підтримують індексування з коробки, тоді як фреймворки MapReduce ні. Модель програмування є декларативною в СУБД (використовується мова SQL).

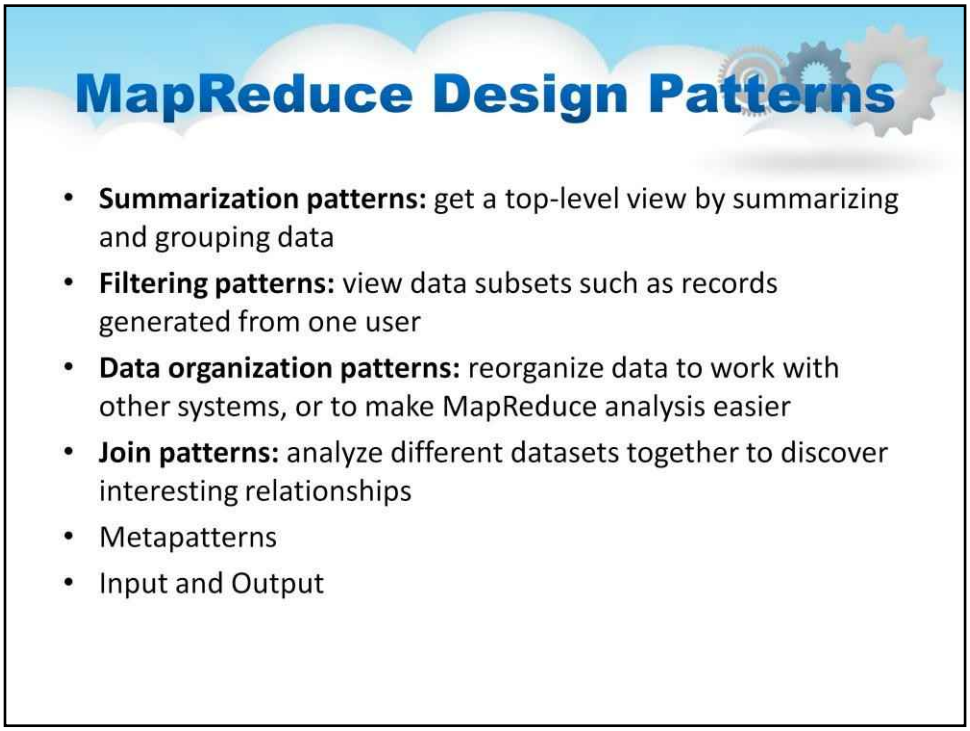
У MapReduce модель програмування є обов’язковою. Ви повинні визначити функції відображення та скорочення (наприклад, код Java).



Підсумовуючи вищесказане, система керування базами даних більше підходить для обробки онлайн-транзакцій, а MapReduce — для аналізу великих даних.

Шаблони проектування

Парадигма MapReduce є простою та потужною, але вона не надає загального рішення, як вирішити будь-яку проблему в полі великих даних.
Шаблон проектування (загалом) — це багаторазове рішення поширеної проблеми.
Отже, давайте подивимося, як розв’язувати деякі поширені типи завдань у рамках парадигми MapReduce.



Візерунки можна розділити на такі категорії:

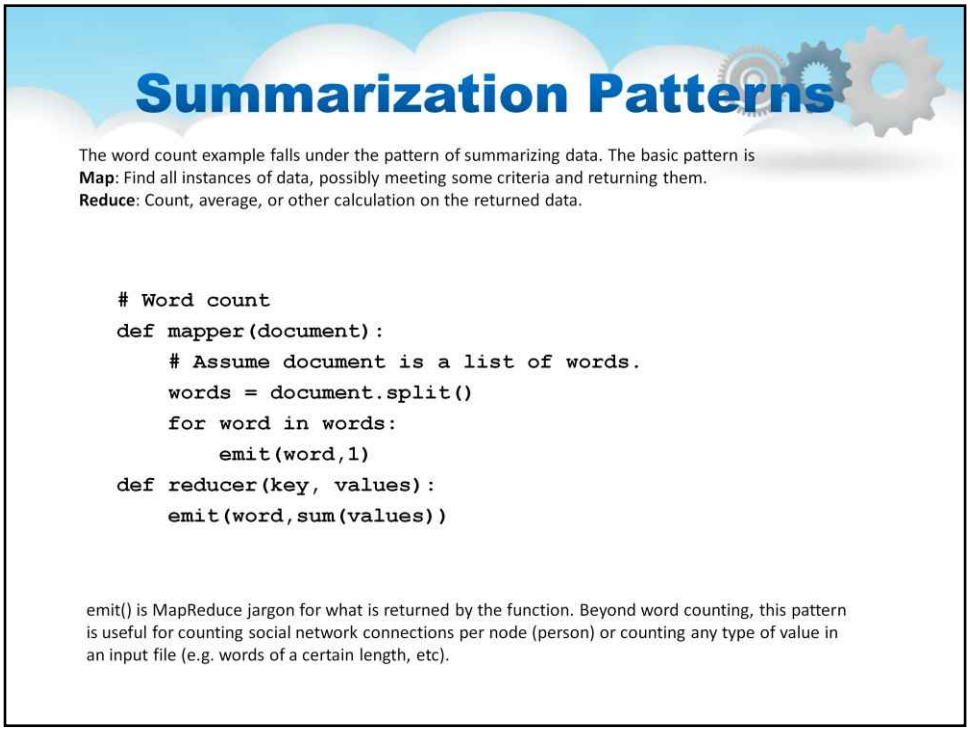
Шаблони підсумовування:отримати представлення верхнього рівня шляхом узагальнення та групування даних

Шаблони фільтрації:переглядати підмножини даних, наприклад записи, створені одним користувачем

Шаблони організації даних:реорганізувати дані для роботи з іншими системами або зробити

Простіший аналіз MapReduce

Шаблони з'єднання:разом аналізувати різні набори даних, щоб виявити цікаві зв'язки**мета-шаблони**і**введення-виведення**візерунки.



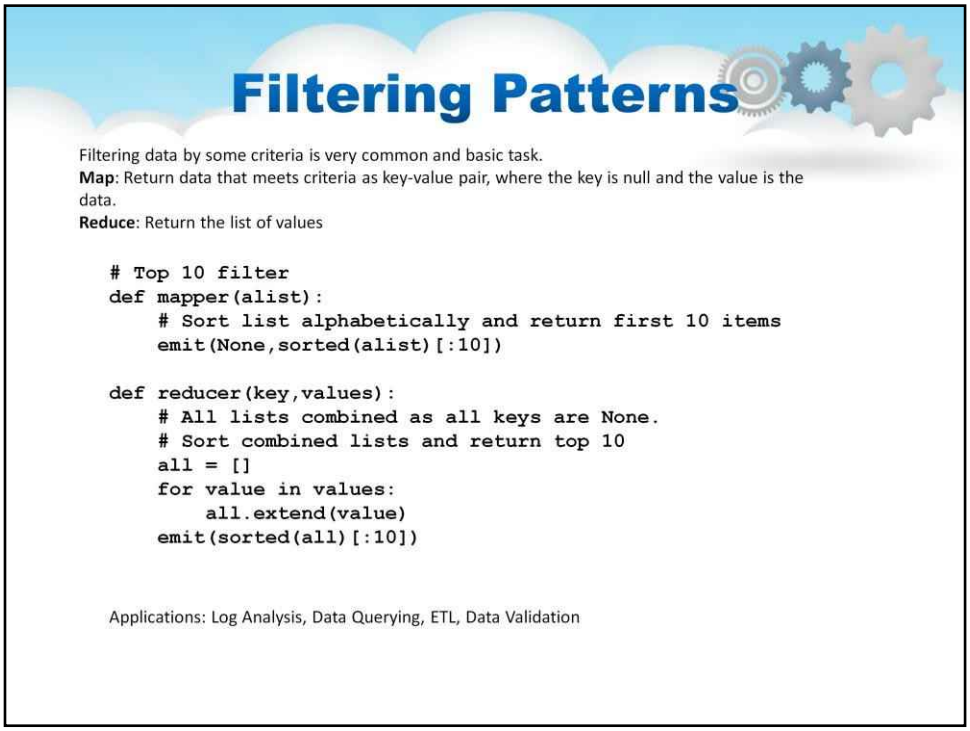
Зразки узагальнення.

Приклад підрахунку слів (описаний раніше) підпадає під шаблон **підведення підсумків**

даних. Основний шаблон:

Карта: знайти всі екземпляри даних, які, можливо, відповідають певним критеріям, і повернути їх.

Зменшити: підрахунок, середнє або інший обчислення на основі повернутих даних. Крім підрахунку слів, цей шаблон корисний для підрахунку підключень до соціальних мереж на людину або підрахунку будь-якого типу значення у вхідному файлі (наприклад, слова певної довжини тощо).



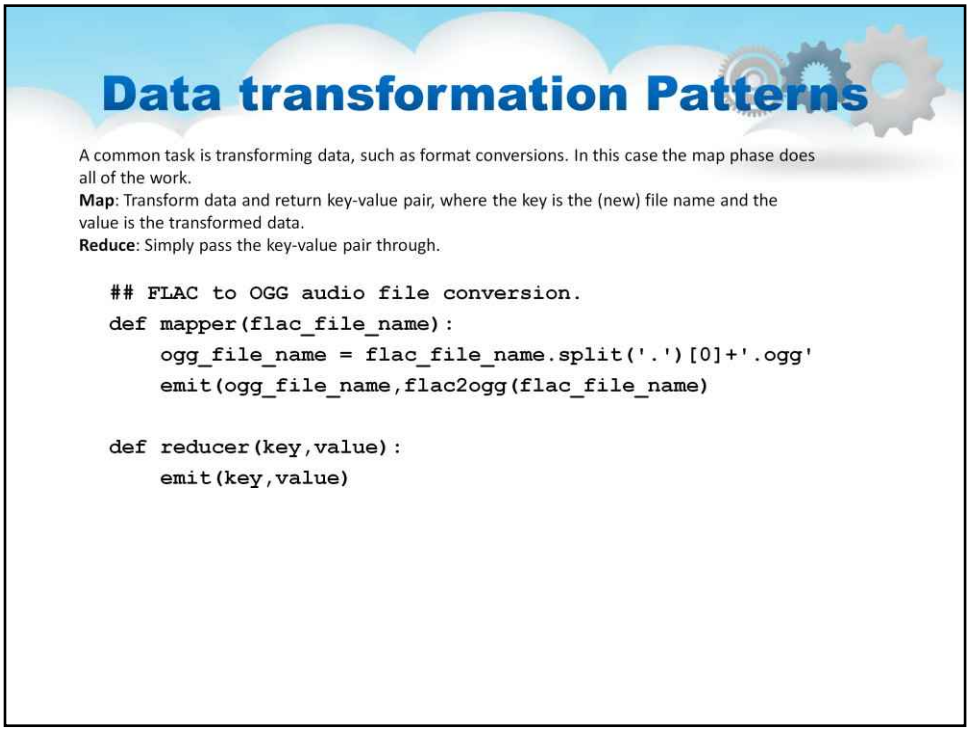
Шаблони фільтрації

Фільтрування даних за деякими критеріями є дуже поширеним і базовим завданням. **Карта**: повертає дані, які відповідають критеріям, як пару ключ-значення, де ключ дорівнює null і

цінність - це дані.

Зменшити: повертає список значень

Приклади програм: аналіз журналу, запит даних, ETL, перевірка даних



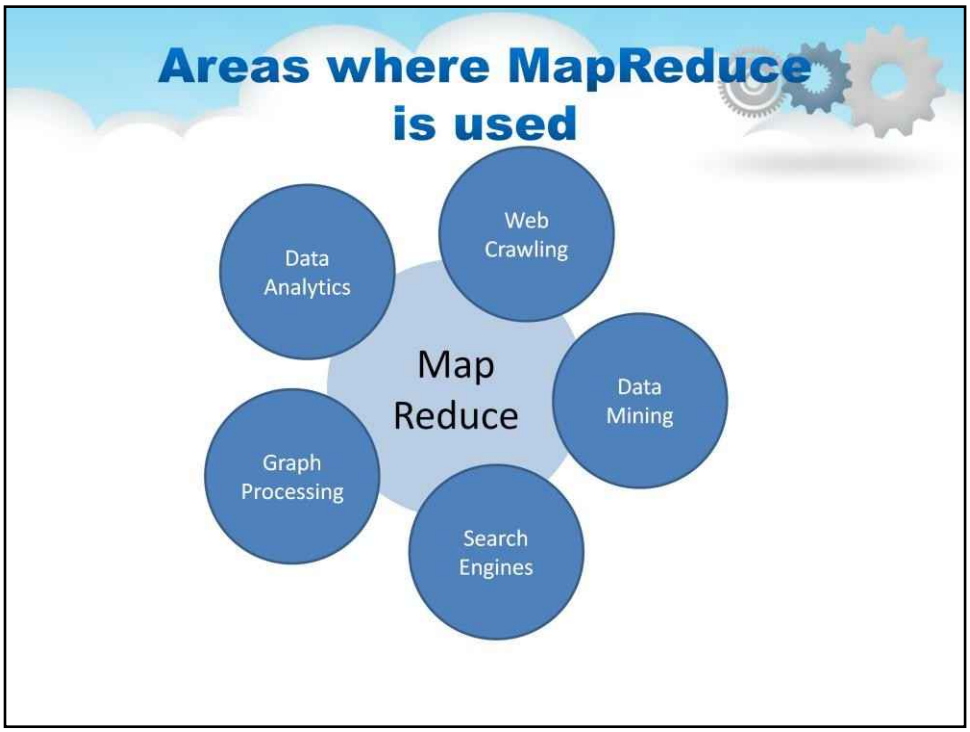
Поширеним завданням є перетворення даних, наприклад перетворення форматів. У цьому випадку етап карти виконує всю роботу.

Карта: перетворює дані та повертає пару ключ-значення, де ключ є (новою) назвою файлу

а значенням є перетворені дані. **Зменшити**:

просто передайте пару ключ-значення.

Об’єднання дуже важливі в системах керування реляційними базами даних, але є одними з найскладніших операцій у MapReduce.
MapReduce добре обробляє набори даних, розглядаючи кожен запис ізольовано, але
об’єднання/комбінування наборів даних не вписується в парадигму MapReduce.
Детальний розгляд цієї теми виходить за рамки цього курсу. Щоб дізнатися більше, зверніться, наприклад, до цієї книги **«Шаблони проектування MapReduce: створення ефективних алгоритмів і аналітики для Hadoop та інших систем»**



Додатки

Аналітика даних:

MapReduce є одним із найпопулярніших рішень для аналізу даних. Зараз найпоширеніша аналітика даних включає такі речі, як підрахунок і аналіз унікальних кліків або відвідувачів веб-сайту, пошук найбільш відвідуваних/пошукових продуктів за місяць на веб-сайті електронної комерції.

Сканування даних:

Існують сервіси та проекти, які сканують дані в Інтернеті, наприклад програми, які сканують дані Твіттера, щоб знайти деякі факти чи дійти певних висновків. Такі додатки мають працювати з великою кількістю даних. MapReduce є хорошим рішенням для сканування та обробки такого роду даних.

Видобуток даних:

Інтелектуальний аналіз даних — ще один вид проблеми, де використовується MapReduce. Наприклад, нам потрібно видобути велику кількість даних (скажімо, цілий архів газетної компанії) і створити певну класифікацію тексту або кластеризацію цих архівів. Знову ж таки, якщо обсяг даних великий, який неможливо обробити в одній системі, MapReduce використовується поверх кластера Hadoop.

Пошукові системи:

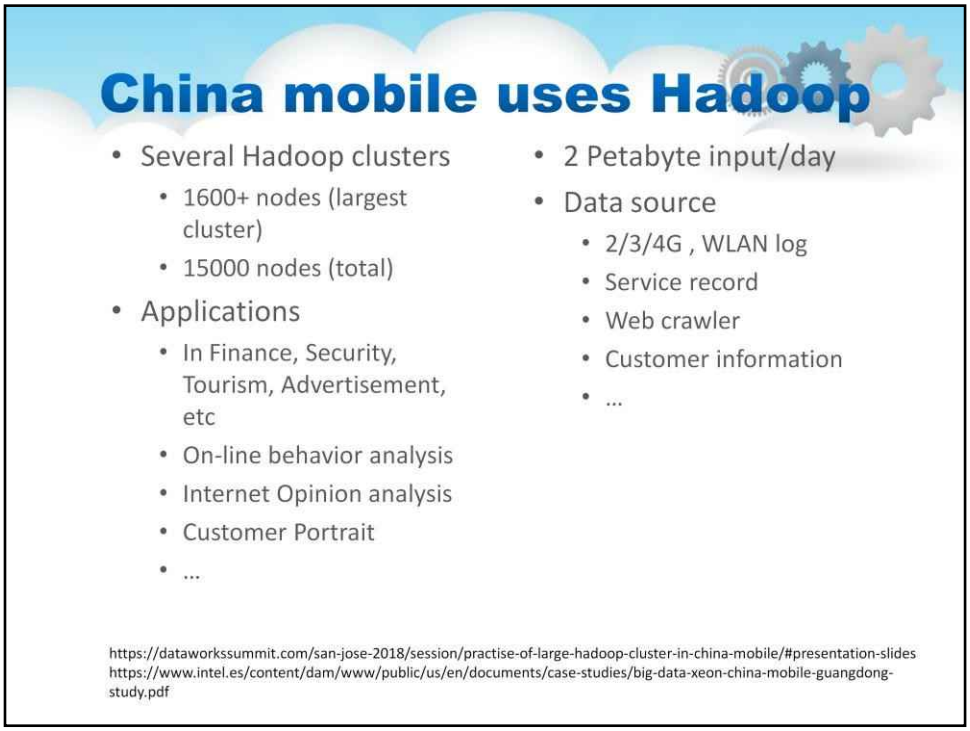
Google була компанією, яка спочатку розробила MapReduce, а потім на основі наукової статті Google була розроблена версія з відкритим кодом під назвою Hadoop MapReduce. Швидше за все, Google розробив MapReduce для індексування тексту. Тепер очевидно, що Google або будь-яка подібна компанія потребує отримання інформації з великої кількості даних/документів, які вони мають, і результати мають бути швидкими. Тому такі дані потрібно індексувати, щоб покращити продуктивність. MapReduce використовується для створення інвертованих індексів цих документів тощо. Інвертований індекс — це структура даних, яка зберігає карту від вмісту до його розташування у файлі/документі. MapReduce використовується для створення таких інвертованих індексів.

Обробка графів:

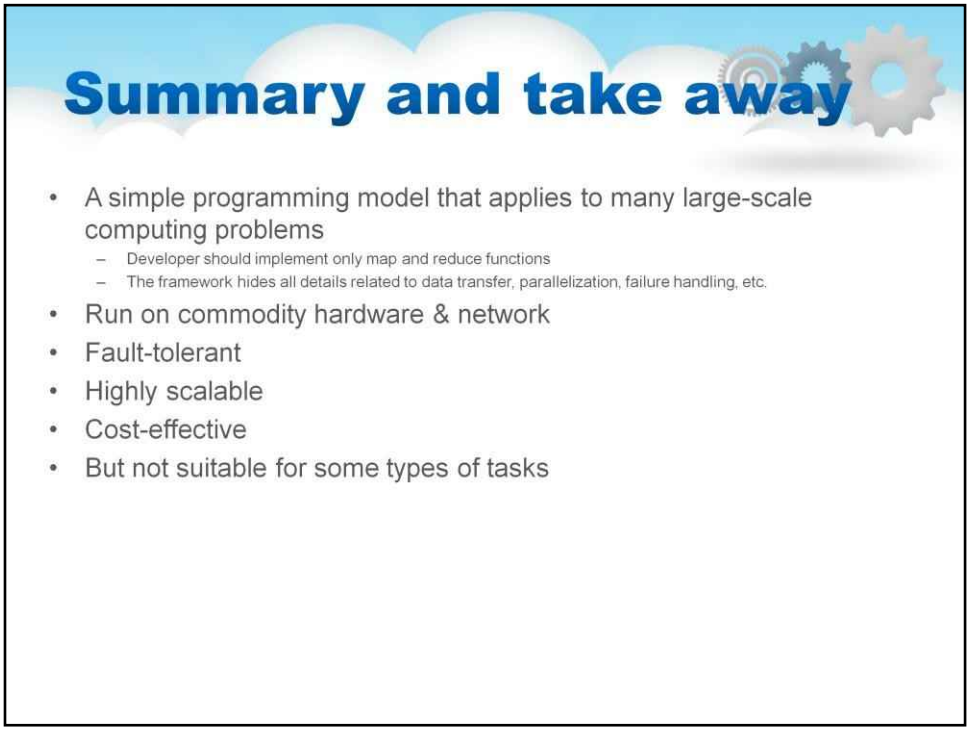
Припустимо, що існує мережа сутностей і зв’язків між ними. Тепер нам потрібно обчислити стан кожної сутності на основі властивостей сусідніх сутностей. Цю проблему також називають обробкою графіка або аналізом графіка, який найчастіше використовується в аналізі соціальних мереж. MapReduce можна використовувати для такого типу обробки графіків, оскільки графік соціальних мереж зазвичай досить великий

Alibaba Cloud Elastic MapReduce (E-MapReduce) — це рішення для обробки великих обсягів даних для швидкої обробки величезних обсягів даних.
Базуючись на Apache Hadoop і Apache Spark з відкритим кодом, E-MapReduce гнучко керує вашими випадками використання великих даних, такими як аналіз тенденцій, сховище даних і аналіз безперервних потокових даних. E-MapReduce спрощує обробку великих даних, роблячи легким, швидким, масштабованим і економічно ефективним для вас створення розподілених кластерів Hadoop і обробку ваших даних.

Це допоможе вам оптимізувати свій бізнес за допомогою кращих рішень на основі масивного аналізу даних у режимі реального часу.



У деяких звітах і матеріалах конференцій згадується, що China Mobile розгорнула великий кластер Hadoop.
Під час розгортання інженери компанії зіткнулися з деякими труднощами
успішно подолав його.
Найближчим часом кількість вузлів кластера збільшиться до 14 000.

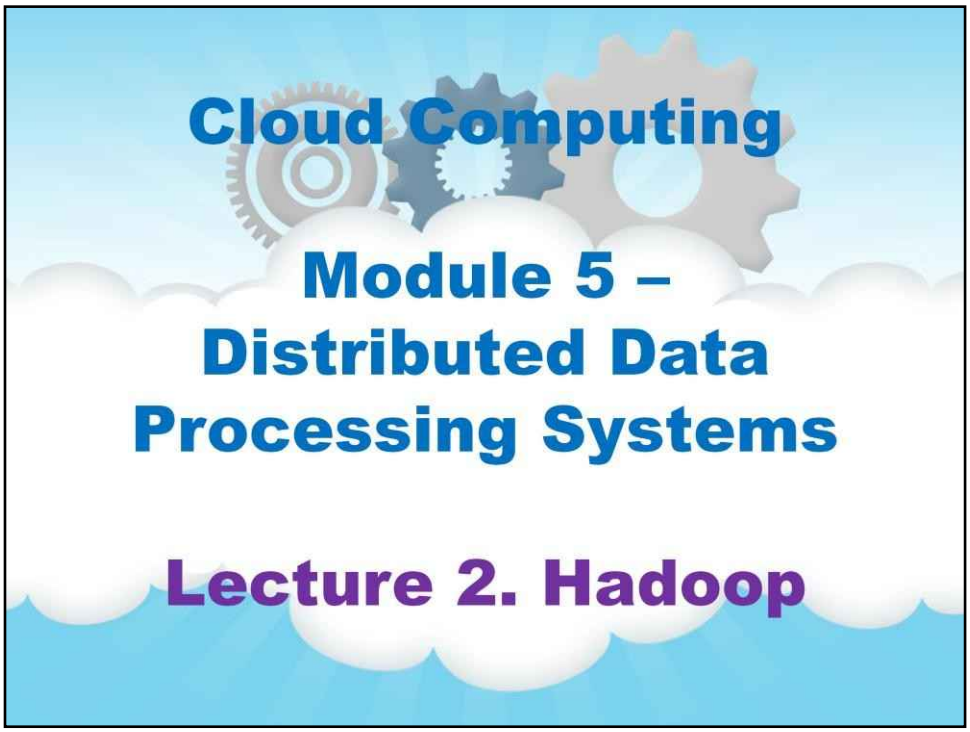


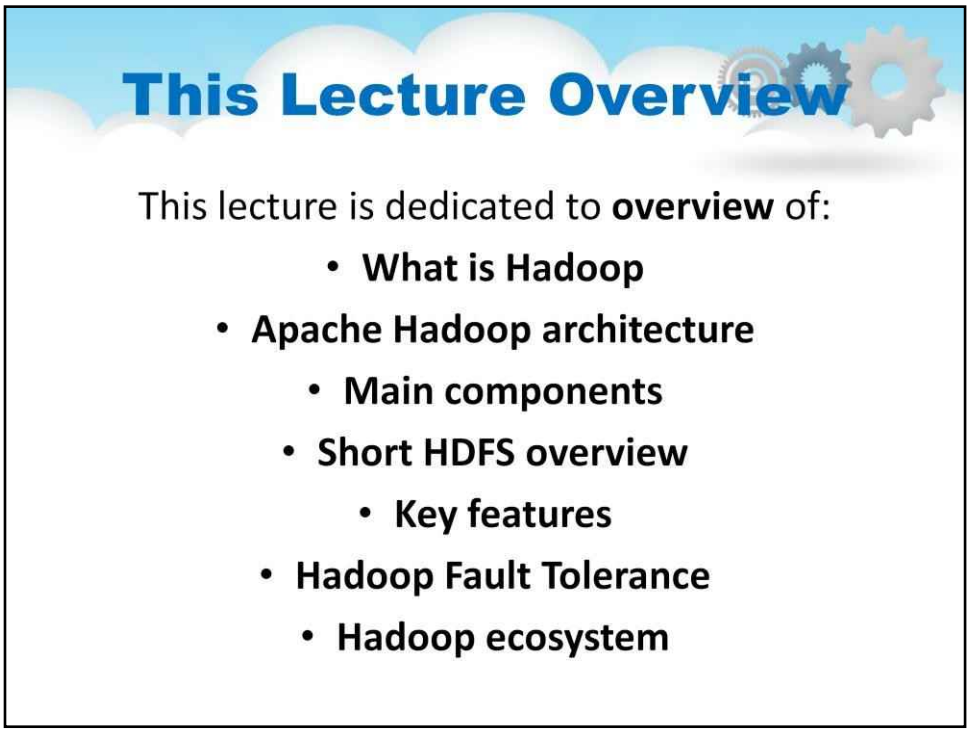
У цьому уроці ми розглянули

MapReduce — це проста модель програмування та відповідна реалізація для

обробка та генерація великих наборів даних.

Розробник повинен реалізувати тільки дві функції - Map і Reduce. Сучасні фреймворки MapReduce мають високу масштабованість (до 10000 вузлів і більше), відмовостійкі, можуть працювати на стандартному (дешевому) обладнанні. MapReduce не є «срібною кулею». У деяких випадках MapReduce працює не дуже добре.


Cloud Computing
Module 5 –
Distributed Data
Processing Systems
Lecture 2. Hadoop



Лекція 2. Hadoop

Ця лекція присвячена огляду:
- Що таке Hadoop
- Архітектура Apache Hadoop
- Основні компоненти
- Короткий огляд HDFS
- Ключові риси
- Відмовостійкість Hadoop
- Екосистема Hadoop

Контур

Що таке Hadoop

Історія Hadoop

Ключ Hadoop

особливості

Основний Hadoop

компоненти

Архітектура

ПРЯЖА

Файлова система HDFS

оболонка HDFS

команди HDFS

тиражування

Оптимізація локалізації даних

Hadoop 1.x проти 2.x проти 3.x

порівняння Hadoop

екосистема

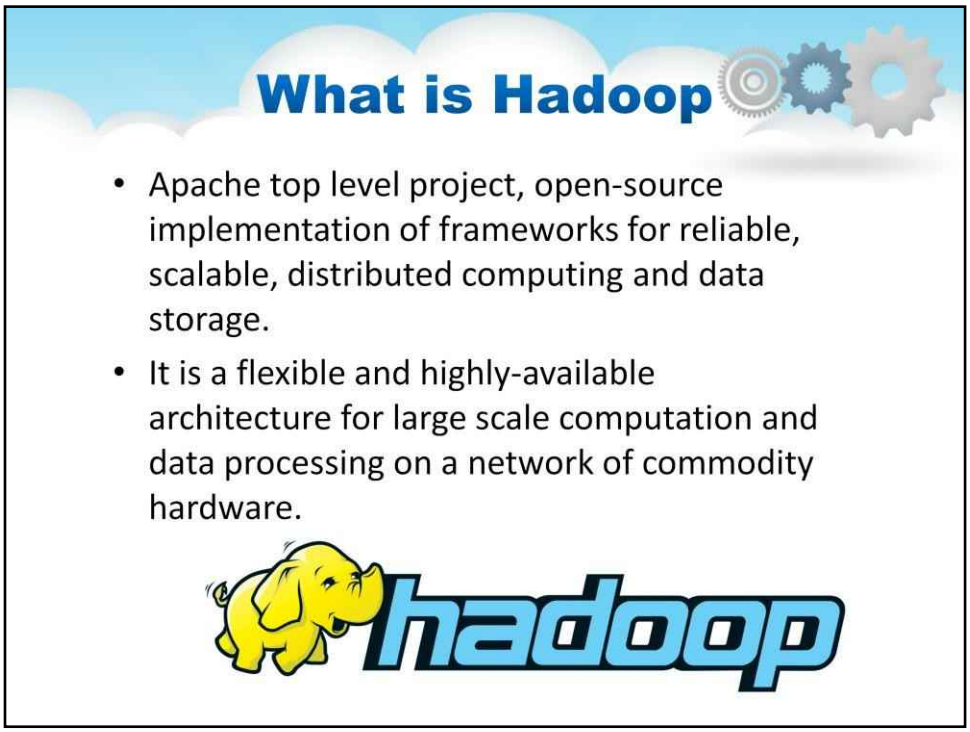
Свиня Апачі

Apache

Вулик

Apache

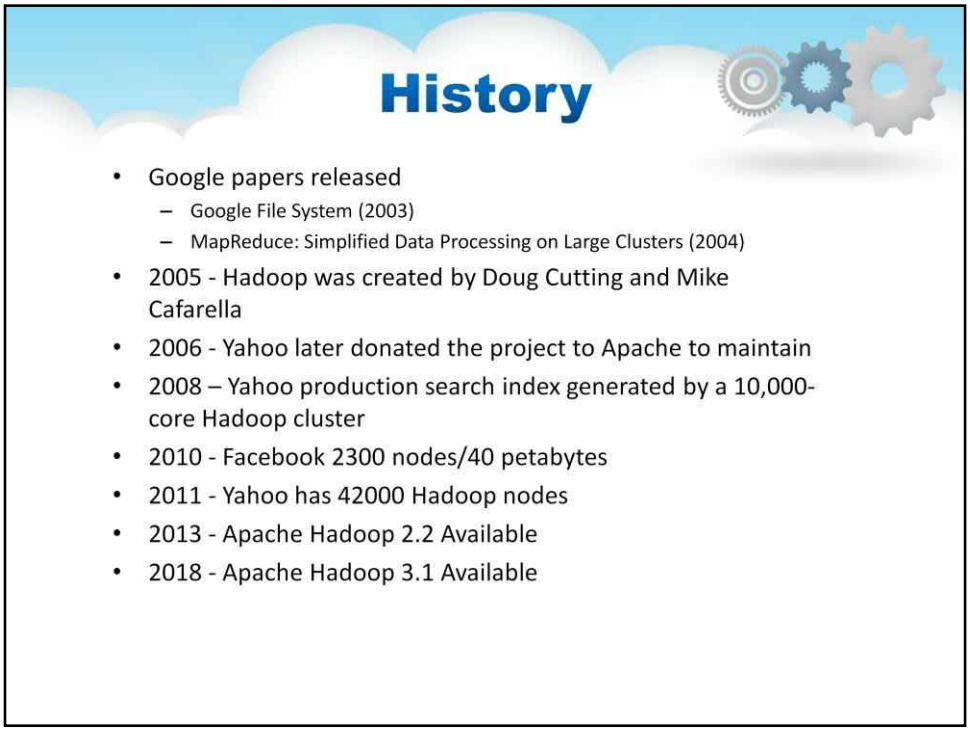
HBase



Огляд

Apache Hadoop — це проект верхнього рівня, реалізація інфраструктури з відкритим кодом для надійних, масштабованих, розподілених обчислень і зберігання даних.

Це гнучка та високодоступна архітектура для великомасштабних обчислень і обробки даних у мережі стандартного обладнання.



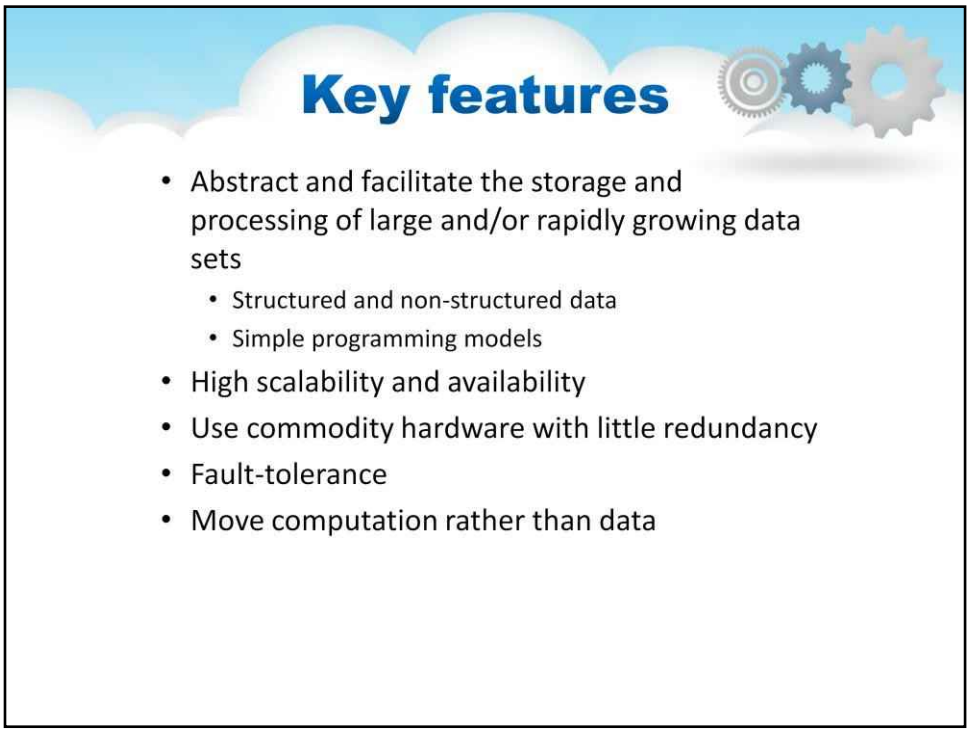
Даг Каттінг і Майк Кафарелла, засновники Hadoop, надихалися архітектурою Google GFS і MapReduce.

Google представив систему GFS (файлова система Google) для розподіленого зберігання величезних наборів даних у кластері стандартного обладнання та технологію MapReduce для обробки наборів даних, наявних у цих розподілених системах.

У 2004 році Google опублікував технічні документи GFS і MapReduce. Даг Каттінг і Майк Кафарелла, які тоді працювали в Nutch, надихнулися технологією Google і почали створювати власну пошукову систему під назвою Nutch на основі файлової системи Google і технології MapReduce.

Спочатку Hadoop був розроблений для підтримки розповсюдження для проекту пошукової системи Nutch.

А в 2006 році Yahoo! Найняв Дага Каттинга та представив світові фреймворк Hadoop, назвавши проект на честь іграшкового слона свого сина. Зараз, за допомогою цього фреймворку, багато організацій почали аналізувати величезні набори даних, які залишалися невирішеними протягом багатьох десятиліть.



Основні функції Apache Hadoop

• Абстрагувати та полегшувати зберігання та обробку великих та/або швидко зростаючих наборів даних

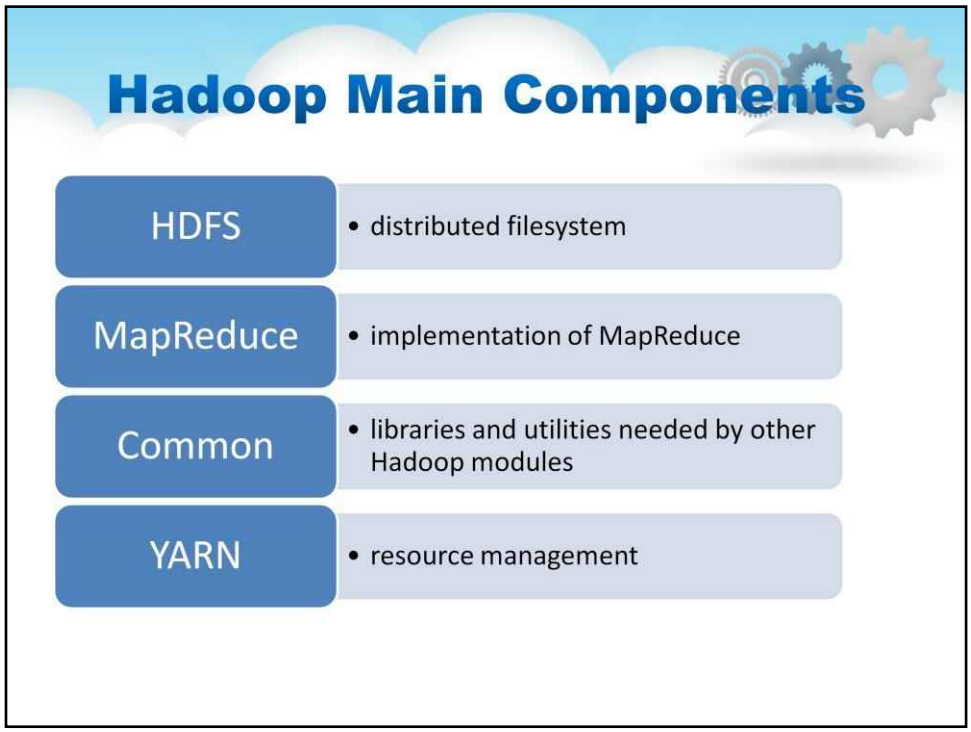
- Структуровані та неструктуровані дані
- Прості моделі програмування

• Висока масштабованість і доступність

• Використовуйте стандартне (дешеве) обладнання з невеликим резервуванням

• Відмовостійкість

• Перемістіть обчислення, а не дані

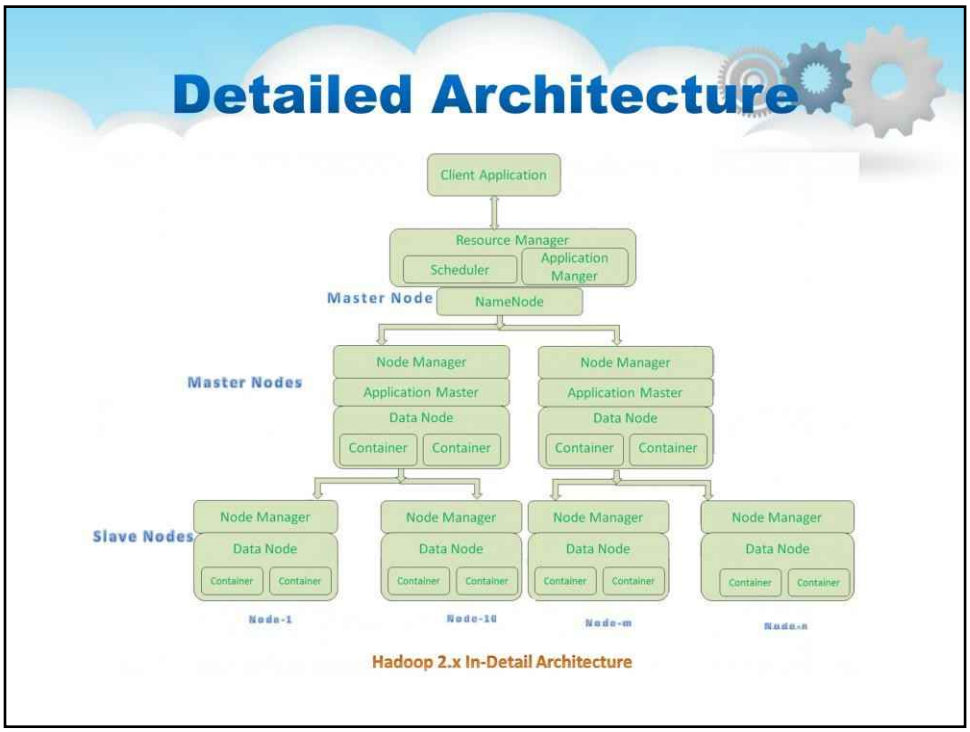


Компоненти Hadoop

Основними компонентами в першій ітерації Hadoop були MapReduce, розподілена файлова система Hadoop (HDFS) і Hadoop Common, набір спільних утиліт і бібліотек. Як видно з назви, MapReduce використовує функції відображення та зменшення, щоб розділити завдання обробки на кілька завдань, які виконуються на вузлах кластера, де зберігаються дані, а потім об'єднати те, що виробляють завдання, у послідовний набір результатів. Спочатку MapReduce функціонував і як механізм обробки Hadoop, і як менеджер ресурсів кластера, що прив'язувало HDFS безпосередньо до нього та обмежувало користувачів запускати пакетні програми MapReduce.

Це змінилося в Hadoop 2.0, який став загальнодоступним у жовтні 2013 року, коли було випущено версію 2.2.0. Він представив Apache Hadoop YARN, нову технологію управління ресурсами кластера та планування завдань, яка взяла ці функції від MapReduce. YARN — скорочення від Yet Another Resource Negotiator, але зазвичай згадується лише абревіатурою — припинило сувору залежність від MapReduce і відкрило Hadoop для інших механізмів обробки та різноманітних програм, окрім пакетних завдань.

Серія випусків Hadoop 2.0 також додала функції високої доступності (HA) і об'єднання для HDFS, підтримку запуску кластерів Hadoop на серверах Microsoft Windows та інші можливості, призначені для розширення універсальності інфраструктури розподіленої обробки для управління великими даними та аналітики.
Hadoop 3.0.0 була наступною основною версією Hadoop. Випущений Apache у грудні 2017 року, він не розширив набір основних компонентів Hadoop. Однак він додав функцію YARN Federation, розроблену для того, щоб YARN підтримував десятки тисяч або більше вузлів в одному кластері порівняно з попереднім обмеженням у 10 000 вузлів. Нова версія також включала підтримку графічних процесорів і кодування стирання, альтернативу реплікації даних, яка вимагає значно менше місця для зберігання.



Архітектура

Менеджер ресурсівє компонентом на рівні кластера. Менеджер ресурсів розділений на два компоненти: планувальник і менеджер додатків.
Планувальник ресурсів менеджера відповідає за планування необхідних ресурсів для додатків (тобто для кожної програми).
Він виконує лише планування та піклується про моніторинг або відстеження цих програм.

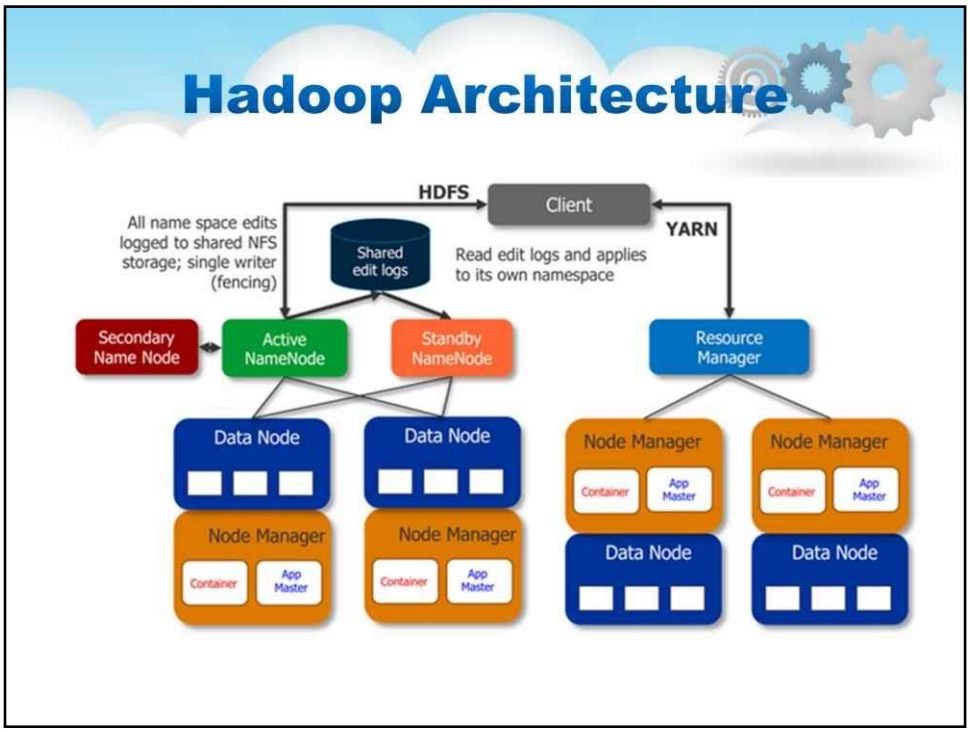
Майстер застосуванняє компонентом рівня програми. Він відповідає за:

- Управління призначеним життєвим циклом програми.
- Він взаємодіє як із планувальником ресурсів, так і з диспетчером вузлів
- Він взаємодіє з Планувальником для отримання необхідних ресурсів.
- Він взаємодіє з диспетчером вузлів для виконання призначених завдань і моніторингу стану цих завдань.

Менеджер вузлівє компонентом рівня кожного вузла та відповідає за:

- Управління життєвим циклом контейнера.
- Контроль використання ресурсів кожного контейнера.

Кожен головний або підлеглий вузол містить набір**Контейнери**.
Контейнер — це частина пам’яті в HDFS (вузол імен або вузол даних).



У Hadoop 2.0 Job Tracker у YARN в основному залежить від 3 важливих компонентів

1. Компонент менеджера ресурсів:

Цей компонент розглядається як узгоджувач усіх ресурсів у кластері. Менеджер ресурсів далі класифікується як менеджер додатків, який керуватиме всіма завданнями користувачів за допомогою кластера та планувальника, що підключається. Це невпинна служба YARN, призначена для отримання та запуску програм у кластері Hadoop. У Hadoop 2.0 завдання MapReduce розглядатиметься як додаток.

2. Компонент диспетчера вузлів:

Це серверний компонент історії завдань YARN, який надаватиме інформацію про всі виконані завдання. NM відстежує всі завдання користувачів і їхній робочий процес на будь-якому конкретному вузлі.

3. Головний компонент програми (він же Диспетчер життєвого циклу роботи користувача):

Це компонент, у якому фактично розміщується завдання, а компонент Application Master відповідає за керування кожним завданням Map Reduce і завершується після завершення обробки завдання.

Менеджер ресурсів RM

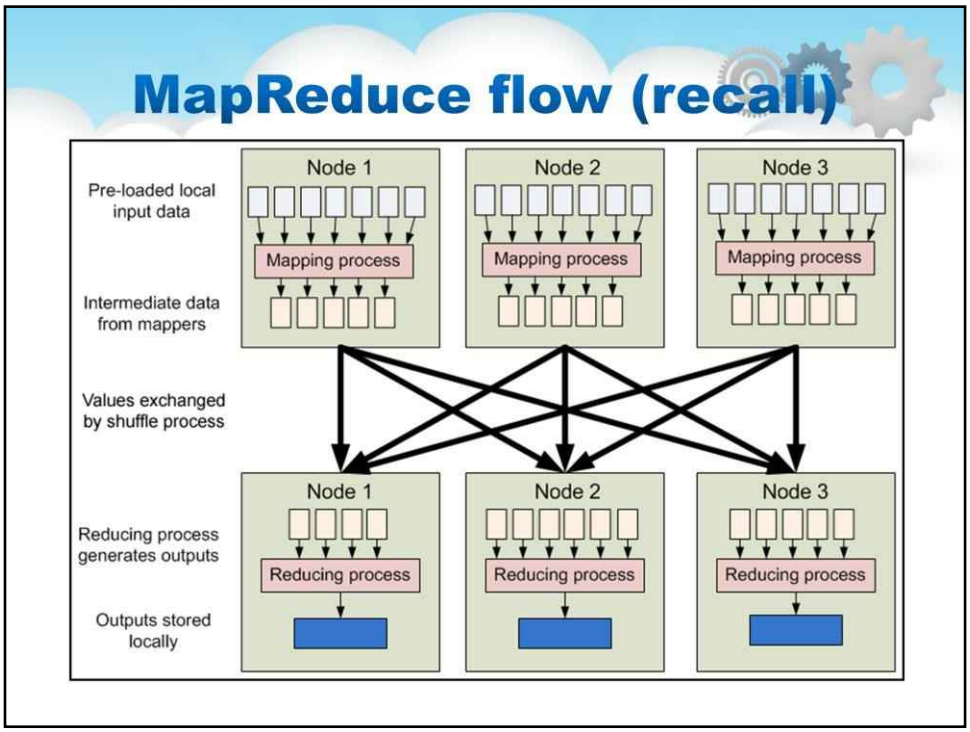
1. Це глобальний планувальник ресурсів
2. Він працює на головному вузлі кластера
3. Він відповідає за узгодження ресурсів системи між конкуруючими програмами.
4. Він відстежує серцебиття від Node Manager

NM-Node Manager

1. Менеджер вузлів спілкується з менеджером ресурсів. 2. Він працює на підлеглих вузлах кластера

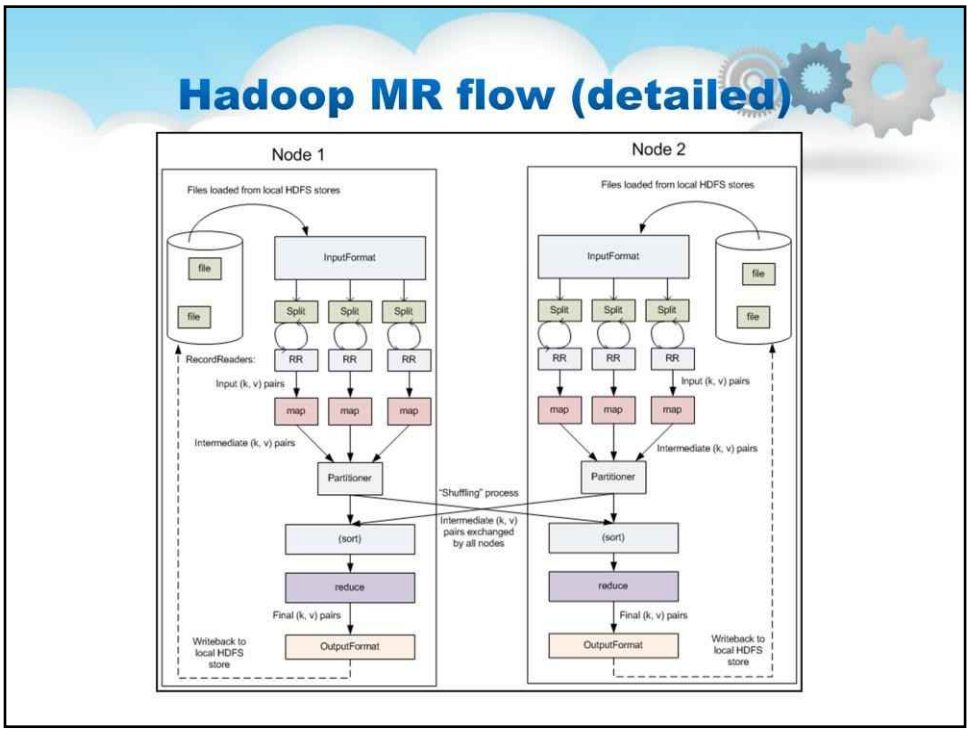
AM-Application Master

1. На кожну програму існує одна AM, яка залежить від програми або фреймворку. 2. AM працює в контейнерах, створених менеджером ресурсів за запитом.



У попередній лекції ми обговорювали модель програмування MapReduce.
Нагадаємо основні моменти потоку даних MapReduce.
Hadoop обмежує обсяг зв’язку, який можуть виконувати процеси, оскільки кожен окремий запис обробляється завданням ізольовано один від одного.

Хоча спочатку це звучить як серйозне обмеження, це робить всю структуру набагато надійнішою. Hadoop не запускатиме будь-яку програму та не розподілятиме її по кластеру.
Програми мають бути написані відповідно до певної моделі програмування під назвою "MapReduce". У MapReduce записи обробляються ізольовано викликаними завданнями Картографи.
Вихідні дані Mappers об’єднуються в другий набір завдань, що називається редуктори, де результати від різних картографів можна об’єднати.

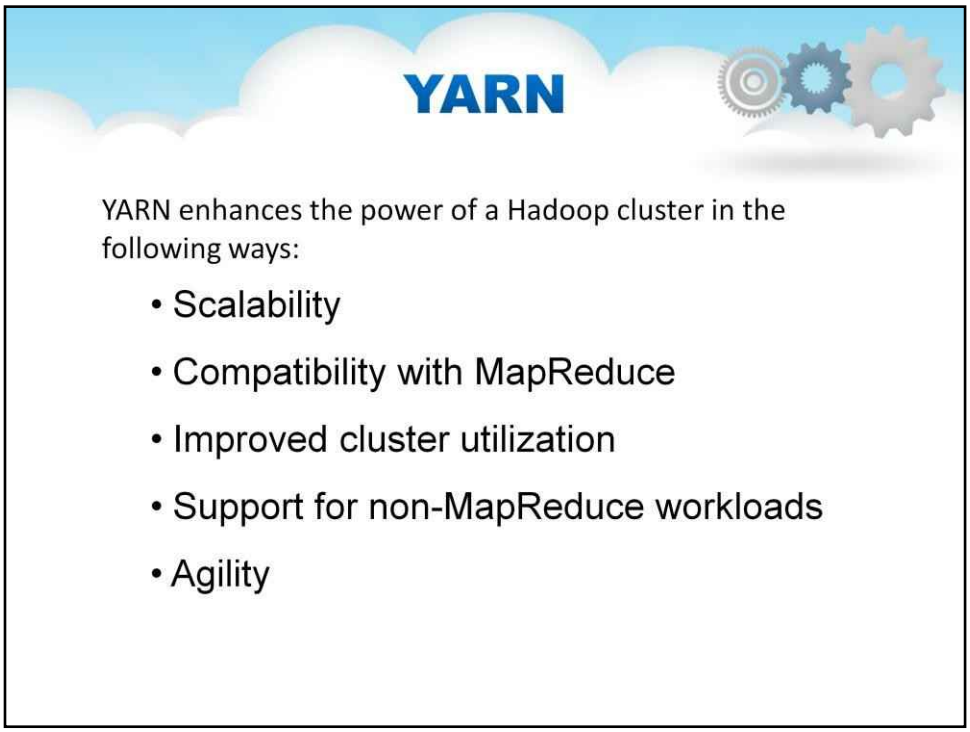


На цій детальній діаграмі ви можете побачити компоненти прикладу додатка «Підрахунок слів».
Вхідні файли: тут спочатку зберігаються дані для завдання (зазвичай у HDFS).
InputFormatвизначає спосіб розділення та читання вхідних файлів. Кілька форматів введення надаються з Hadoop.**InputSplits**: описує одиницю роботи, яка складається з одного завдання карти в програмі MapReduce. Завдання карти можуть читати цілий файл або лише частину файлу.

The**InputSplit**визначає частину роботи, але не описує, як до неї отримати доступ. The **RecordReader**фактично завантажує дані з джерела та перетворює їх у пари (ключ, значення), придатні для читання Картографом.
The**Картограф**виконує визначену користувачем роботу на етапі карти. Маючи ключ і значення, метод map() випромінює пару(и) (ключ, значення), які пересилаються до **редуктори**.
Після завершення перших завдань карти кожен з вузлів може все ще виконувати ще кілька завдань карти. Але вони також починають обмінюватися проміжними виходами із завдань карти туди, де вони потрібні редукторам. Цей процес переміщення виходів карти до редукторів відомий як**перетасування**.
Кожне завдання зменшення відповідає за зменшення значень, пов’язаних із кількома проміжними ключами. Набір проміжних ключів на одному вузлі автоматично сортується Hadoop перед тим, як вони будуть представлені**Редуктор**.**Редуктор**це екземпляр наданого користувачем коду, який виконує фазу зменшення.
Спосіб запису пар (ключ, значення) визначається за допомогою**OutputFormat**.
Функції OutputFormat схожі на клас InputFormat, описаний раніше.
The**OutputFormat**клас є фабрикою для**RecordWriter**об'єкти; вони використовуються для запису окремих записів у файли, як визначено**OutputFormat**.
The**вихідні файли**написані Reducers потім зберігаються в HDFS.

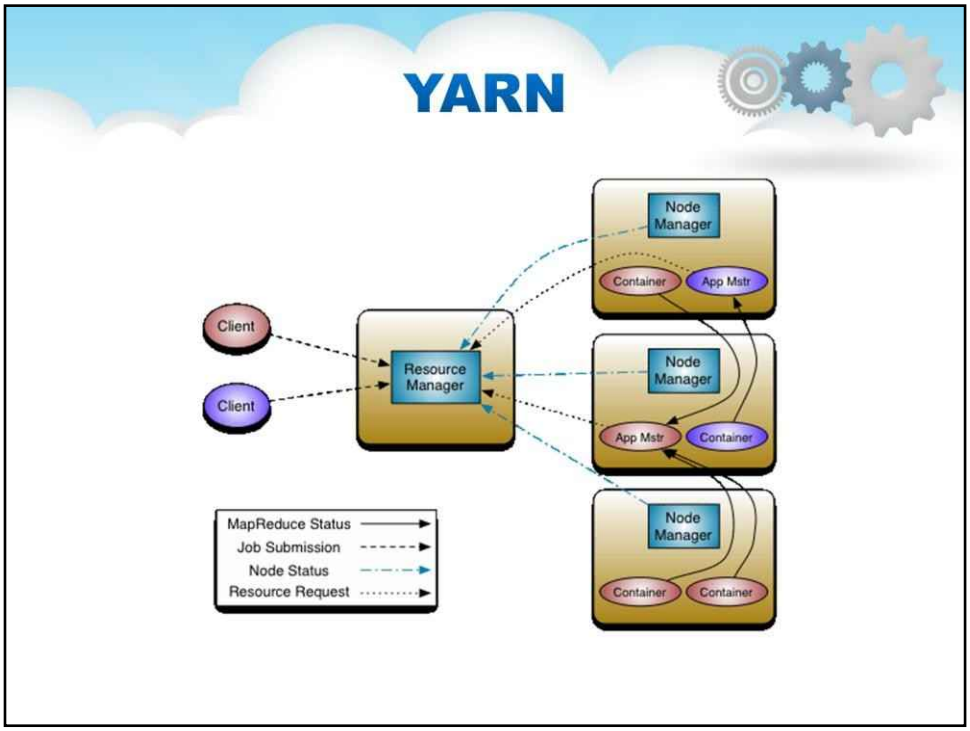
ПРЯЖА

Apache Hadoop YARN — це відкрита технологія керування ресурсами та планування завдань
вихідна структура розподіленої обробки Hadoop. Один із основних компонентів Apache Hadoop, YARN, відповідає за розподіл системних ресурсів для різних програм, що працюють у кластері Hadoop, і планування завдань, які потрібно виконати на різних вузлах кластера.



Що робить YARN:
YARN підвищує потужність обчислювального кластера Hadoop такими способами:
Масштабованість: Обчислювальна потужність у центрах обробки даних продовжує швидко зростати. Оскільки YARN ResourceManager зосереджується виключно на плануванні, він може набагато легше керувати великими кластерами.
Сумісністьз MapReduce: наявні програми MapReduce і користувачі можуть працювати поверх YARN без переривання існуючих процесів.
Покращене використання кластера: ResourceManager — це чистий планувальник, який оптимізує використання кластера відповідно до таких критеріїв, як гарантії пропускної здатності, справедливість і SLA. Крім того, на відміну від раніше, немає іменованих карт і слотів зменшення, що допомагає краще використовувати ресурси кластера.
Підтримка для**навантаження, відмінні від MapReduce**: для обробки даних тепер можливі додаткові моделі програмування, такі як обробка графіків та ітераційне моделювання.

Спритність: Коли MapReduce стає бібліотекою для користувачів, вона може розвиватися незалежно від основного рівня менеджера ресурсів і набагато гнучкіше.



Фундаментальна ідея YARN полягає в тому, щоб розділити функції управління ресурсами та планування/моніторинг завдань на окремі демони.

Ідея полягає в тому, щоб мати глобальний ResourceManager (RM) і ApplicationMaster для кожної програми (AM). Програма — це або окреме завдання, або група завдань.

ResourceManager і NodeManager утворюють структуру обчислення даних.

ResourceManager — це головний орган, який розподіляє ресурси між усіма програмами в системі. NodeManager — це агент фреймворка для кожного комп'ютера, який відповідає за контейнери, відстежує використання їхніх ресурсів (процесор, пам'ять, диск, мережа) і повідомляє про це ResourceManager/Scheduler.

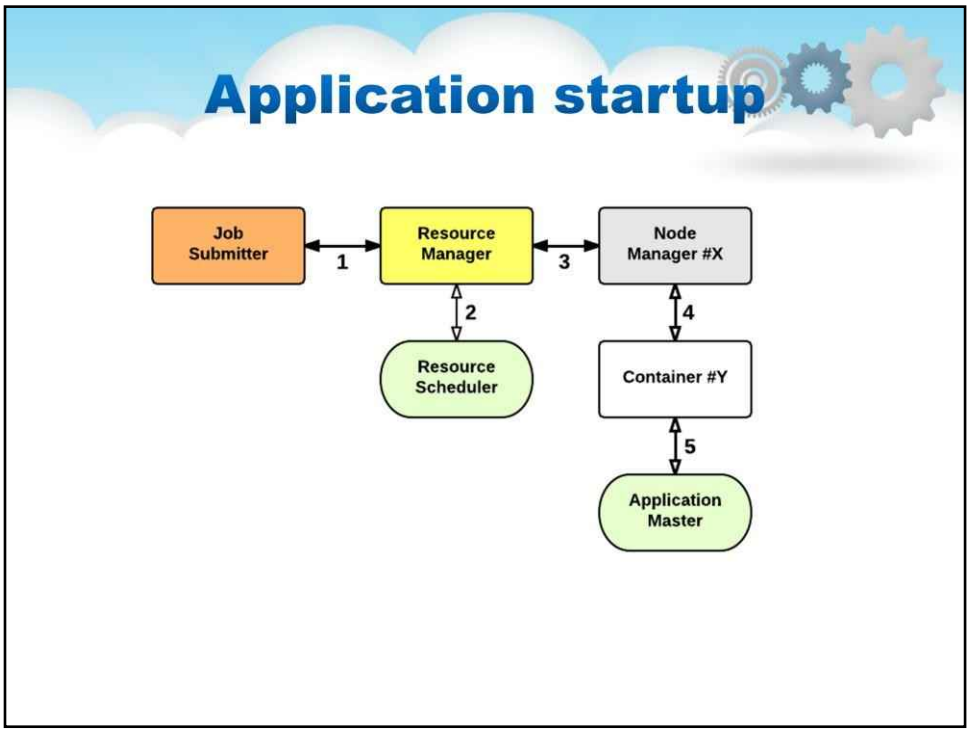
ApplicationMaster для кожного додатка — це, по суті, бібліотека для конкретного фреймворку, і їй доручено узгоджувати ресурси з ResourceManager і працювати з NodeManager(ами) для виконання та моніторингу завдань. ResourceManager має два основні компоненти: Scheduler і ApplicationsManager.

Планувальник відповідає за розподіл ресурсів між різними запущеними програмами відповідно до звичних обмежень потужності, черги тощо. Планувальник є чистим планувальником у тому сенсі, що він не здійснює моніторинг або відстеження статусу програми. Крім того, він не дає жодних гарантій щодо перезапуску невдалих завдань через збій додатків або апаратних збоїв. Планувальник виконує свою функцію планування на основі вимог до ресурсів програм; це робиться на основі абстрактного поняття ресурсу Контейнер, який включає такі елементи, як пам'ять, процесор, диск, мережа тощо.

Планувальник має підключену політику, яка відповідає за розподіл ресурсів кластера між різними чергами, програмами тощо. Поточні планувальники, такі як CapacityScheduler і FairScheduler, можуть бути деякими прикладами плагінів.

ApplicationsManager відповідає за прийняття повідомлень про завдання, узгодження першого контейнера для виконання конкретного ApplicationMaster і надає послугу для перезапуску контейнера ApplicationMaster у разі помилки. ApplicationMaster для кожної програми несе відповідальність за узгодження відповідних контейнерів ресурсів із Планувальника, відстеження їх стану та моніторинг прогресу.

MapReduce у hadoop-2.x підтримує сумісність API із попередньою стабільною версією (hadoop-1.x). Це означає, що всі завдання MapReduce мають виконуватися без змін поверх YARN лише за допомогою перекомпіляції.



У YARN принаймні три актори:

- в**Надсилає роботу**(клієнт)
- в**Менеджер ресурсів**(Майстер)
- в**Менеджер вузлів**(раб)

Процес запуску програми такий:

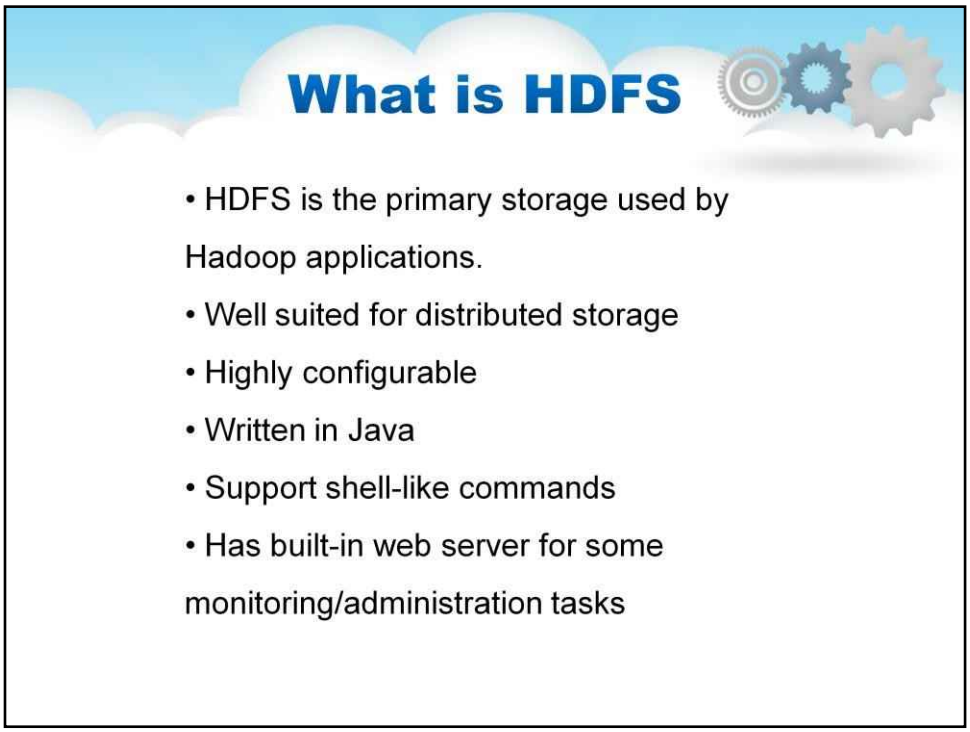
1. клієнт подає заявку менеджеру ресурсів
2. менеджер ресурсів виділяє контейнер
3. менеджер ресурсів зв’язується з відповідним менеджером вузла
4. диспетчер вузлів запускає контейнер
5. Контейнер виконує**Майстер застосування**

Менеджер ресурсів є єдиною точкою відмови в YARN. Використовуючи Application Masters, YARN поширює по кластеру метадані, пов’язані з запущеними програмами. Це зменшує навантаження на диспетчер ресурсів і робить його швидко відновлюваним.

HDFS

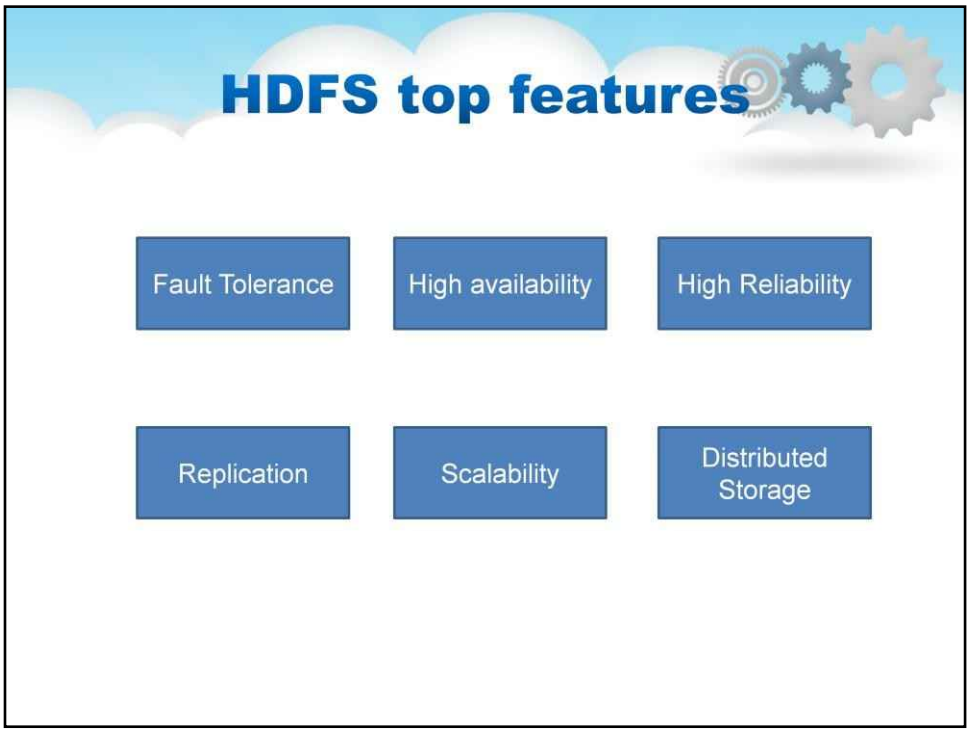
HDFS — це система зберігання фреймворку Hadoop.

Це розподілена файлова система, яка може зручно працювати на звичайному обладнанні для обробки неструктурованих даних.



HDFS — це основне розподілене сховище, яке використовується програмами Hadoop. HDFS добре підходить для розподіленого зберігання та розподіленої обробки з використанням стандартного обладнання. Він відмовостійкий, масштабований і надзвичайно простий у розширенні. HDFS має широкі можливості налаштування з конфігурацією за замовчуванням, яка добре підходить для багатьох установок. У більшості випадків конфігурацію потрібно налаштувати лише для дуже великих кластерів.

Hadoop написаний на Java і підтримується на всіх основних платформах. Hadoop підтримує shell-like команди для безпосередньої взаємодії з HDFS.
Мають вбудовані веб-сервери, які спрощують перевірку поточного стану кластера.



Найкращі функції HDFS

1.Відмовостійкість

Відмовостійкість у Hadoop HDFS — це ефективність роботи системи в несприятливих умовах. Він має високу відмовостійкість. Фреймворк Hadoop ділить дані на блоки. Після цього створює кілька копій блоків на різних машинах у кластері. Отже, коли будь-яка машина в кластері виходить з ладу, клієнт може легко отримати доступ до своїх даних з іншої машини, яка містить ту саму копію блоків даних.

2.Висока доступність

Hadoop HDFS — це високодоступна файлова система. У HDFS дані реплікуються між вузлами в кластері Hadoop шляхом створення копії блоків на інших підлеглих серверах, присутніх у кластері HDFS. Отже, щоразу, коли користувач хоче отримати доступ до цих даних, він може отримати доступ до своїх даних із підлеглих пристроїв, які містять його блоки. Під час несприятливих ситуацій, таких як збій вузла, користувач може легко отримати доступ до своїх даних з інших вузлів. Оскільки дублікати блоків присутні на інших вузлах у кластері HDFS.

3.Висока надійність

HDFS забезпечує надійне зберігання даних. Він може зберігати дані в діапазоні 100 петабайт. HDFS надійно зберігає дані в кластері. Він розбиває дані на блоки. Фреймворк Hadoop зберігає ці блоки на вузлах, присутніх у кластері HDFS. HDFS надійно зберігає дані, створюючи копію кожного блоку в кластері. Таким чином, забезпечується відмовостійкість. Якщо вузол у кластері, що містить дані, виходить з ладу, користувач може легко отримати доступ до цих даних з інших вузлів. HDFS за замовчуванням створює 3 репліки кожного блоку, що містить дані у вузлах. Тому,

дані швидко доступні користувачам. Таким чином, користувач не стикається з проблемою втрати даних. Таким чином, HDFS є високонадійним.

4.тиражування

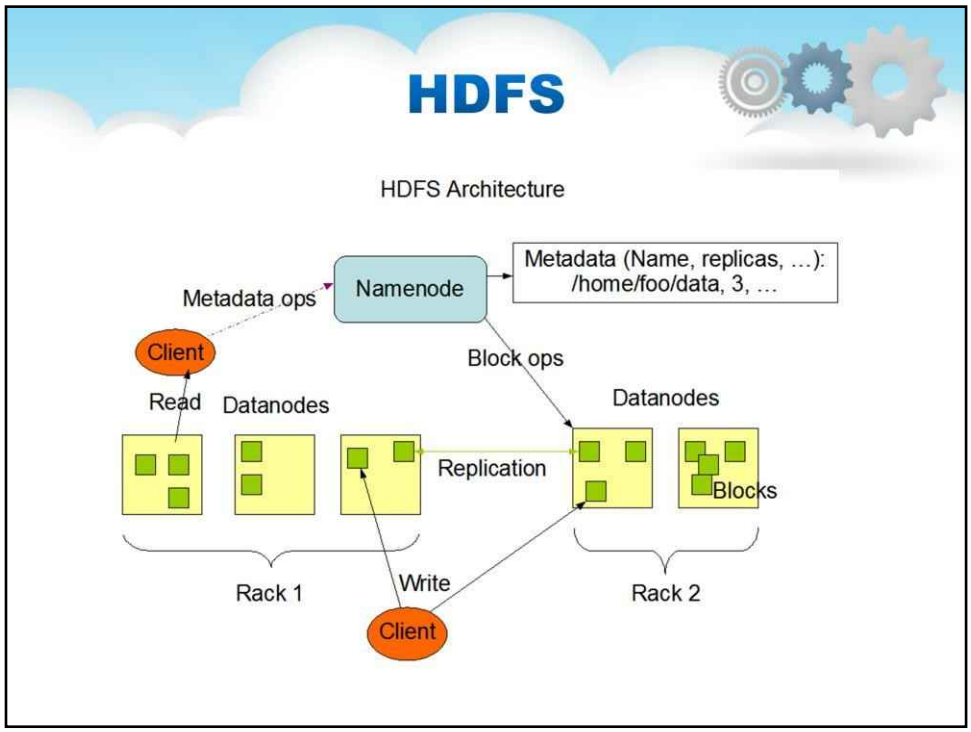
Реплікація даних — унікальна функція HDFS. Реплікація вирішує проблему втрати даних у несприятливих умовах, таких як збій обладнання, збій вузлів тощо. HDFS підтримує процес реплікації через регулярні проміжки часу. HDFS також продовжує створювати копії даних користувача на різних машинах, присутніх у кластері. Отже, коли будь-який вузол виходить з ладу, користувач може отримати доступ до даних з інших машин. Таким чином, немає ймовірності втрати даних користувача.

5.Масштабованість

Hadoop HDFS зберігає дані на кількох вузлах у кластері. Отже, щоразу, коли вимоги збільшуються, ви можете масштабувати кластер. У HDFS доступні два механізми масштабованості: вертикальна та горизонтальна масштабованість.

6.Розподілене сховище

Усі функції HDFS досягаються за допомогою розподіленого зберігання та реплікації. HDFS зберігає дані розподіленим способом між вузлами. У Hadoop дані поділяються на блоки та зберігаються на вузлах, присутніх у кластері HDFS. Після цього HDFS створює репліку кожного блоку та зберігає на інших вузлах. Коли одна машина в кластері виходить з ладу, ми можемо легко отримати доступ до наших даних з інших вузлів, які містять її репліку.

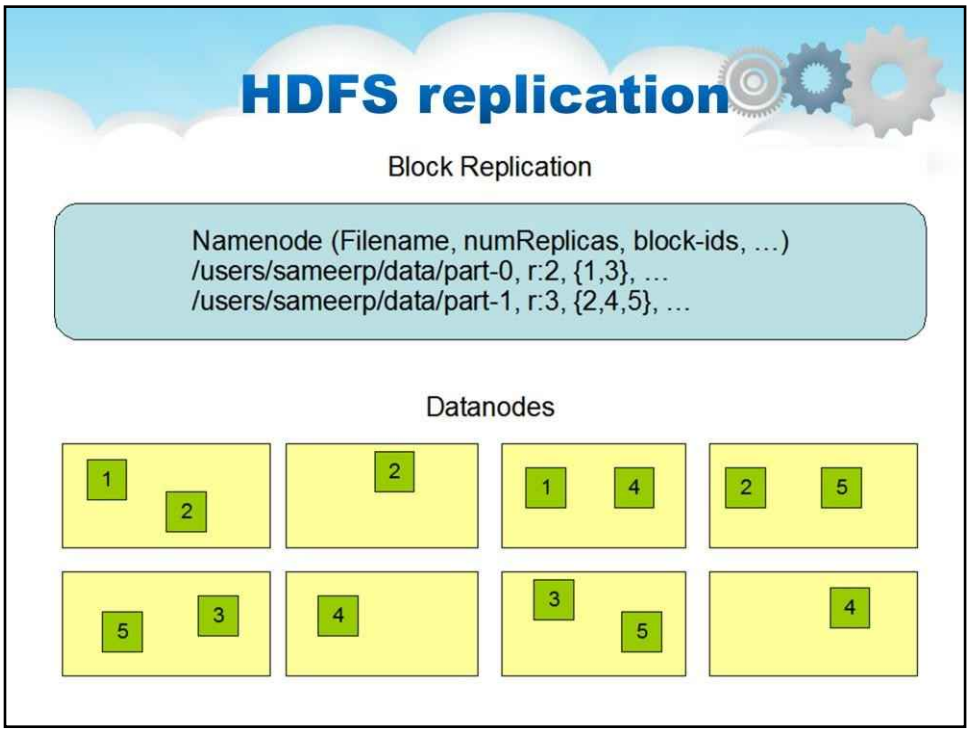


NameNode і DataNodes

HDFS має архітектуру головний/підлеглий. Кластер HDFS складається з одного NameNode, головного сервера, який керує простором імен файлової системи та регулює доступ до файлів клієнтами. Крім того, існує кілька DataNodes, зазвичай по одному на вузол у кластері, які керують сховищем, підключеним до вузлів, на яких вони працюють. HDFS відкриває простір імен файлової системи та дозволяє зберігати дані користувача у файлах. Внутрішньо файл розбивається на один або кілька блоків, і ці блоки зберігаються в наборі DataNodes. NameNode виконує такі операції простору імен файлової системи, як відкриття, закриття та перейменування файлів і каталогів. Він також визначає відображення блоків у DataNodes. DataNodes відповідають за обслуговування запитів на читання та запис від клієнтів файлової системи. DataNodes також виконує створення блоків, видалення,

NameNode і DataNode — це частини програмного забезпечення, призначені для роботи на машинах, що випускаються товарами. На цих машинах зазвичай працює операційна система (ОС) GNU/Linux. HDFS побудовано з використанням мови Java; будь-яка машина, яка підтримує Java, може запускати програмне забезпечення NameNode або DataNode. Використання мови Java з високою портативністю означає, що HDFS можна розгортати на широкому діапазоні машин. Типове розгортання має виділену машину, на якій працює лише програмне забезпечення NameNode. На кожній іншій машині в кластері працює один екземпляр програмного забезпечення DataNode. Архітектура не перешкоджає запуску кількох DataNodes на одній машині, але в реальному розгортанні, що рідко трапляється.

Наявність одного NameNode у кластері значно спрощує архітектуру системи. NameNode є арбітром і сховищем для всіх метаданих HDFS. Система розроблена таким чином, що дані користувача ніколи не проходять через NameNode.



Простір імен файлової системи

HDFS підтримує традиційну ієрархічну організацію файлів. Користувач або програма може створювати каталоги та зберігати файли в цих каталогах. Ієрархія простору імен файлової системи подібна до більшості інших існуючих файлових систем; можна створювати та видаляти файли, переміщувати файл з одного каталогу в інший або перейменовувати файл. HDFS підтримує квоти користувачів і права доступу. HDFS не підтримує жорсткі та м'які посилання. Однак архітектура HDFS не перешкоджає реалізації цих функцій.

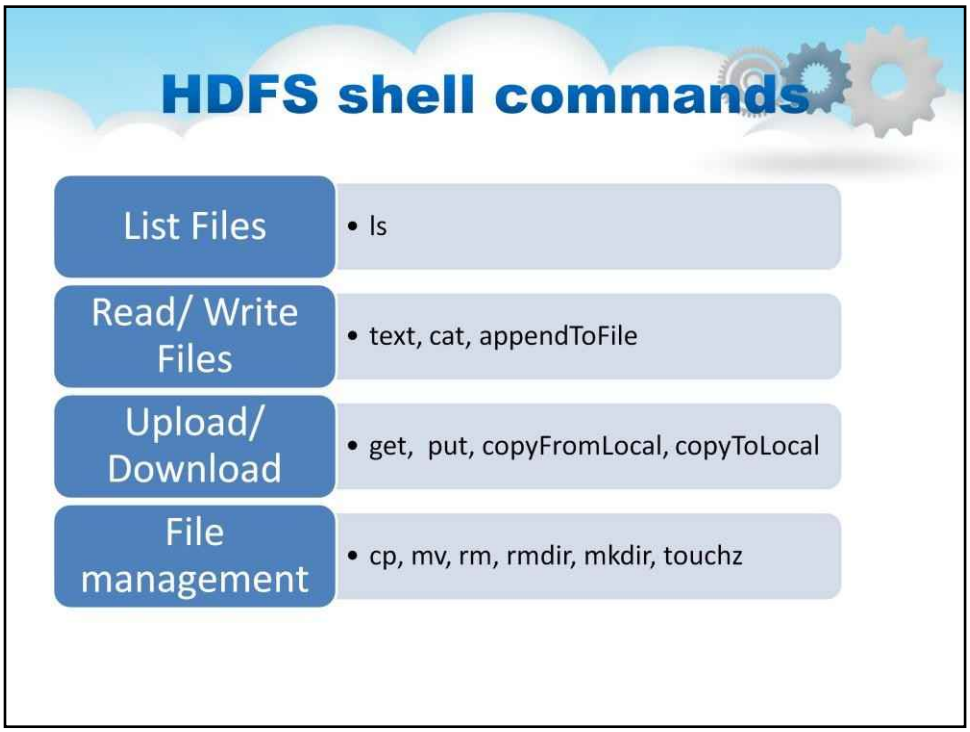
NameNode підтримує простір імен файлової системи. Будь-яка зміна простору імен файлової системи або її властивостей записується NameNode. Програма може вказати кількість реплік файлу, які має підтримувати HDFS. Кількість копій файлу називається коефіцієнтом реплікації цього файлу. Цю інформацію зберігає NameNode.

Реплікація даних

HDFS розроблено для надійного зберігання дуже великих файлів на машинах у великому кластері. Він зберігає кожен файл як послідовність блоків. Блоки файлу копіюються для відмовостійкості. Розмір блоку та коефіцієнт реплікації можна налаштувати для кожного файлу.

Усі блоки у файлі, крім останнього блоку, мають однаковий розмір, тоді як користувачі можуть почати новий блок, не заповнюючи останній блок до налаштованого розміру блоку після того, як для додавання та hsync додано підтримку блоку змінної довжини. Програма може вказати кількість реплік файлу. Фактор реплікації можна вказати під час створення файлу та змінити пізніше. Файли в HDFS записуються одноразово (за винятком додавання та скорочення) і мають строго одного записувача в будь-який час.

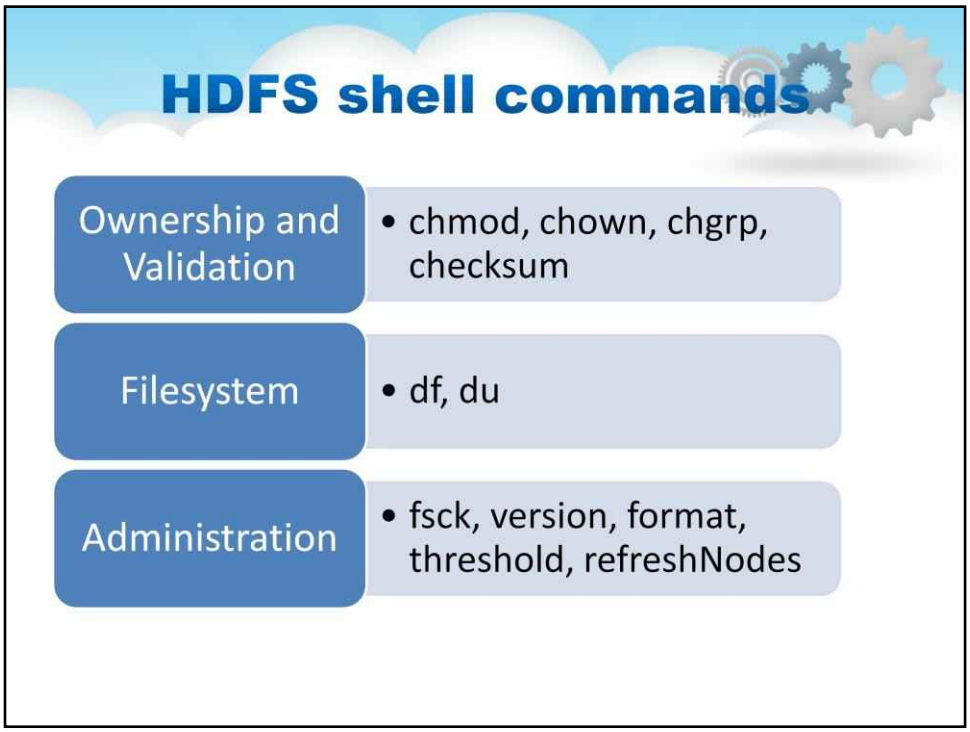
NameNode приймає всі рішення щодо реплікації блоків. Він періодично отримує Heartbeat і Blockreport від кожного з DataNodes у кластері. Отримання Heartbeat означає, що DataNode функціонує належним чином. Blockreport містить список усіх блоків на DataNode.



Оболонка файлової системи містить різні команди, подібні до оболонки, які безпосередньо взаємодіють із розподіленою файловою системою Hadoop (HDFS), а також іншими файловими системами, які підтримує Hadoop.

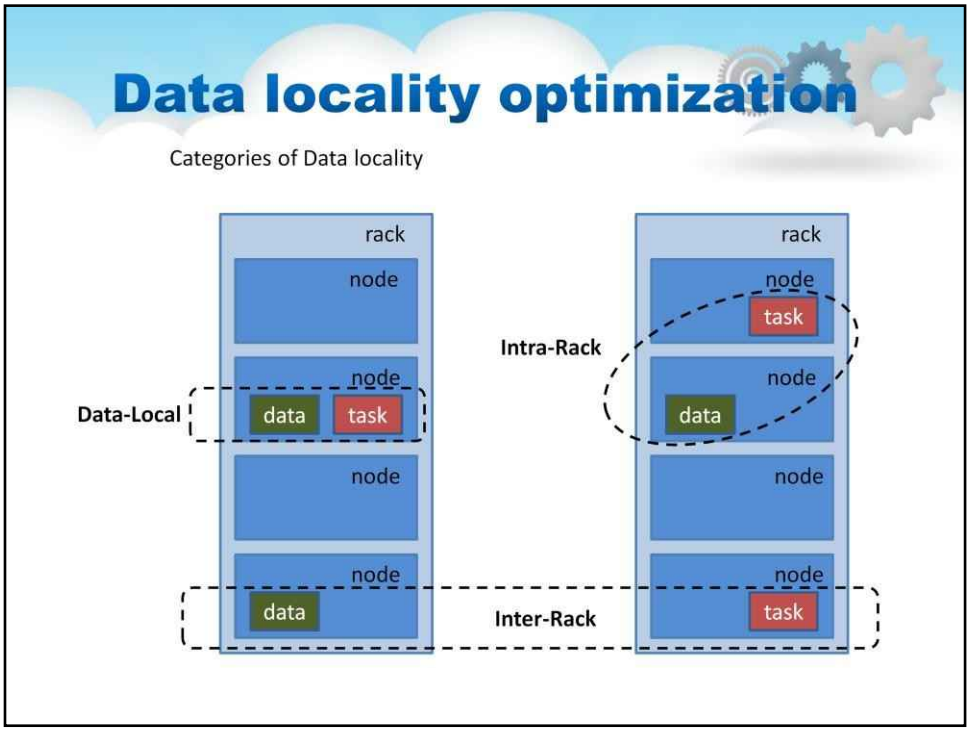
Усі команди поділяються на такі категорії:

- Список файлів (наприклад,**ls**команда)
- Читання/запис файлів (**текст**,**кіт**,**appendToFile**)
- Завантажити/завантажити (**отримати**,**поставити**,**copyFromLocal**,**copyToLocal**)
- Керування файлами (**ср**,**мв**тощо)



- Команди власності та перевірки (наприклад,**chmod**,**chown**тощо)
- Команди файлової системи (**ду**і**df**)
- Команди адміністрування (**fsck**,**формат**тощо)

Команди та способи їх використання будуть детально розглянуті в лабораторній роботі



Локальність даних

У Hadoop це означає переміщення обчислень близько до даних, а не переміщення даних до обчислень. Hadoop зберігає дані в**HDFS**, який розбиває файли на блоки та розподіляє між різними вузлами даних. Коли надсилається завдання mapReduce, воно поділяється на завдання карти та завдання зменшення. Завдання Map призначається вузлу даних відповідно до доступності даних, тобто воно призначає завдання вузлу даних, який знаходиться ближче або зберігає дані на своєму локальному диску. Локальність даних стосується процесу розміщення обчислень поруч із даними, що сприяє високій пропускній здатності та швидшому виконанню даних.

Категорії локальності даних**:**

1. Місцеві дані

Якщо завдання карти виконується на вузлі, який має вхідний блок для обробки, воно називається локальними даними.

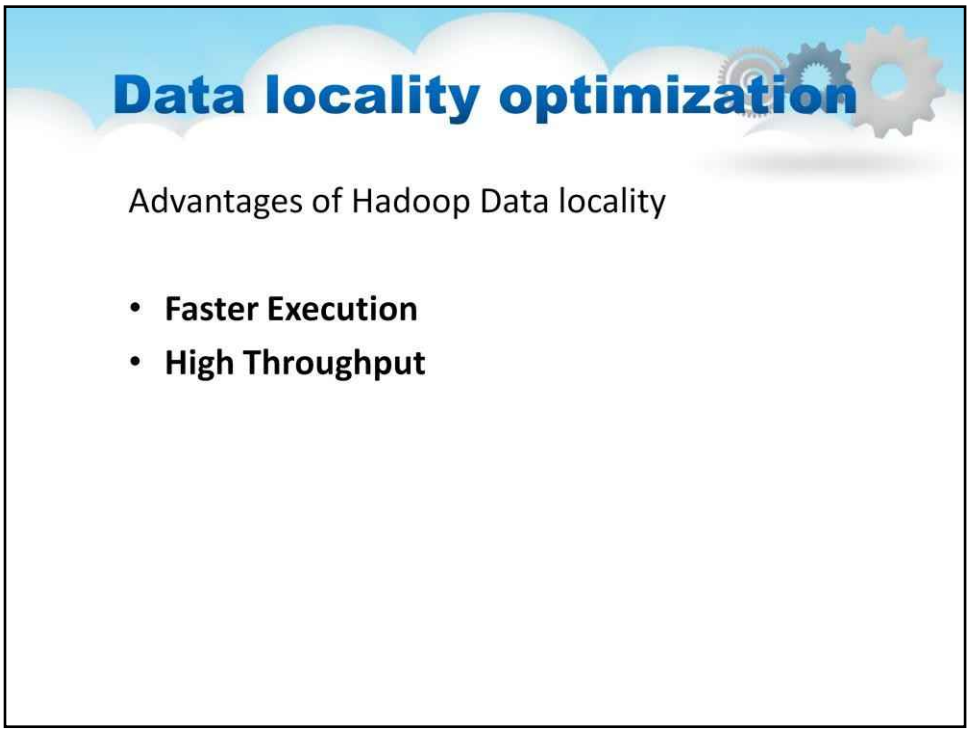
2. Внутрішньостійковий

Не завжди можливо запустити завдання карти на тому самому вузлі, де розташовані дані, через мережеві обмеження. У цьому випадку Mapper працює на іншій машині, але на тій же стійці. Отже, дані потрібно переміщувати між вузлами для виконання.

3. Міжстійка

У деяких випадках Intra-Rack local також неможливий. У таких випадках програма відображення виконуватиметься з іншої стійки. Щоб виконати програму відображення, дані потрібно скопіювати з вузла, який зберігає дані, до вузла, який виконує програму відображення між стійками.

Завдання карти зчитують дані з вхідних блоків і генерують проміжні результати. Оскільки завдання карти працюють на блоках із HDFS і є паралельними даними, локальність даних важлива для кращої продуктивності та швидшого виконання завдань карти.



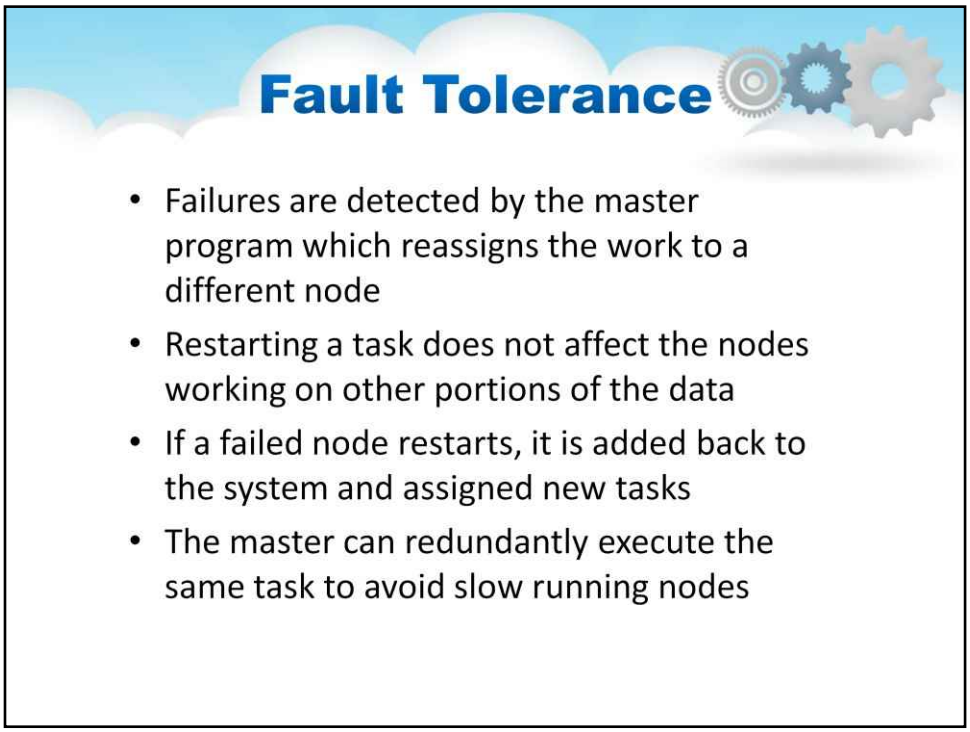
Є дві переваги даних Locality у MapReduce. Давайте обговоримо їх один за одним -

Швидше виконання

У локальності даних програма переміщується до вузла, де зберігаються дані, замість переміщення великих даних до вузла, це робить Hadoop швидшим. Оскільки розмір програми завжди менший за розмір даних, переміщення даних є вузьким місцем передачі по мережі.

Висока пропускна здатність

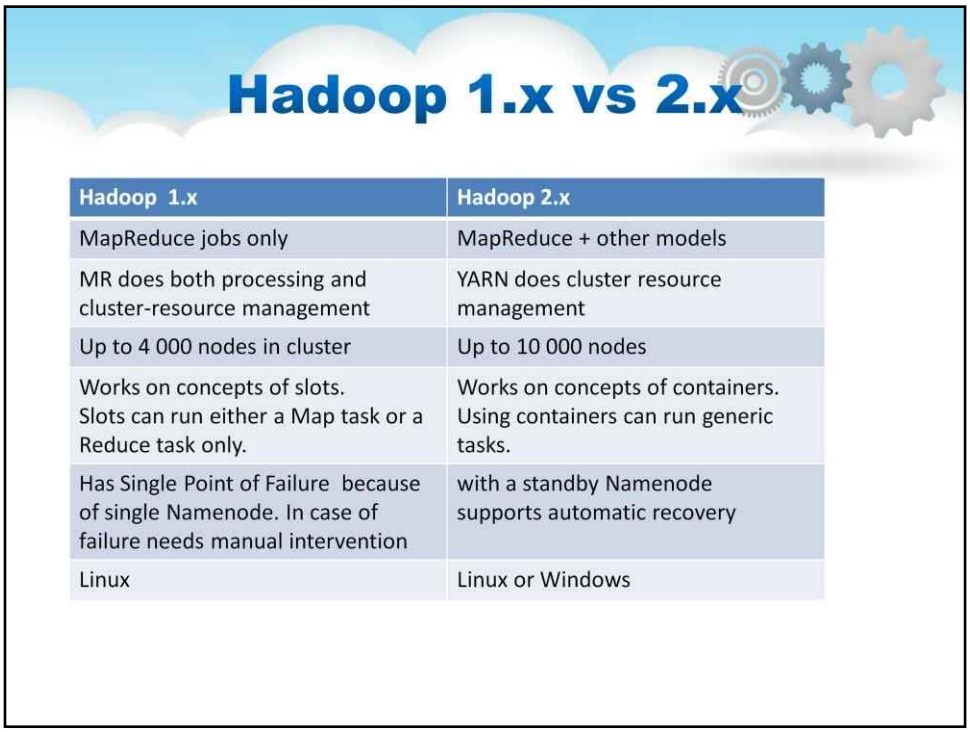
Локальність даних збільшує загальну пропускну здатність системи.



Відмовостійкість

Збої виявляються головною програмою, яка перепризначає роботу іншому вузлу. Перезапуск завдання не впливає на вузли, що працюють з іншими частинами даних.

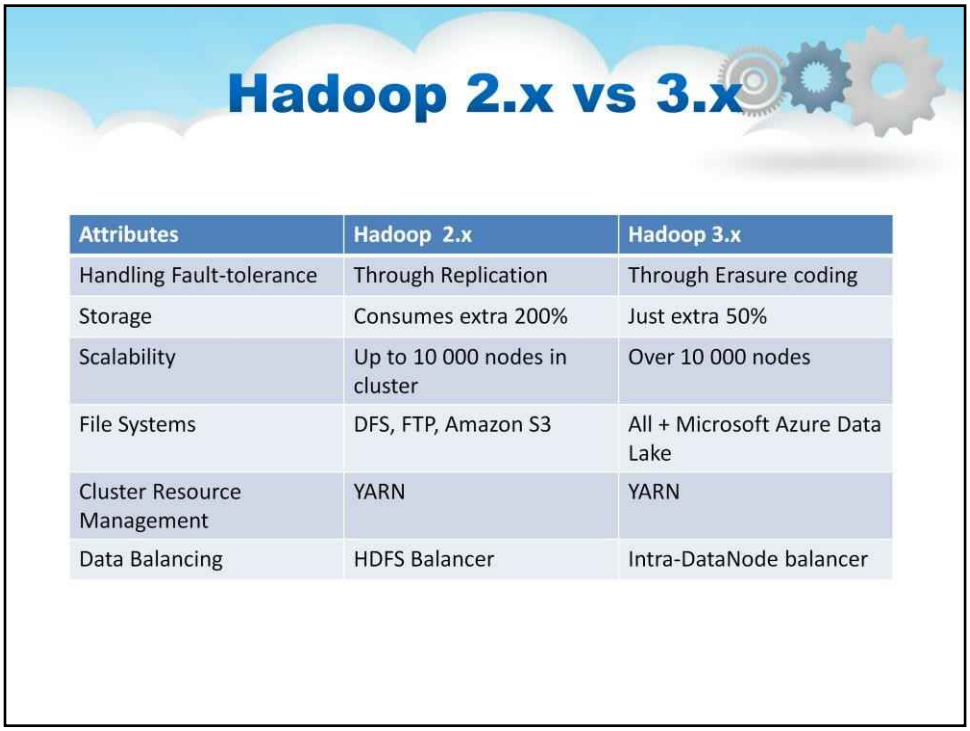
Якщо невдалий вузол перезапускається, він повертається до системи та призначається нові завдання. Майстер може резервно виконувати те саме завдання, щоб уникнути повільної роботи вузлів.



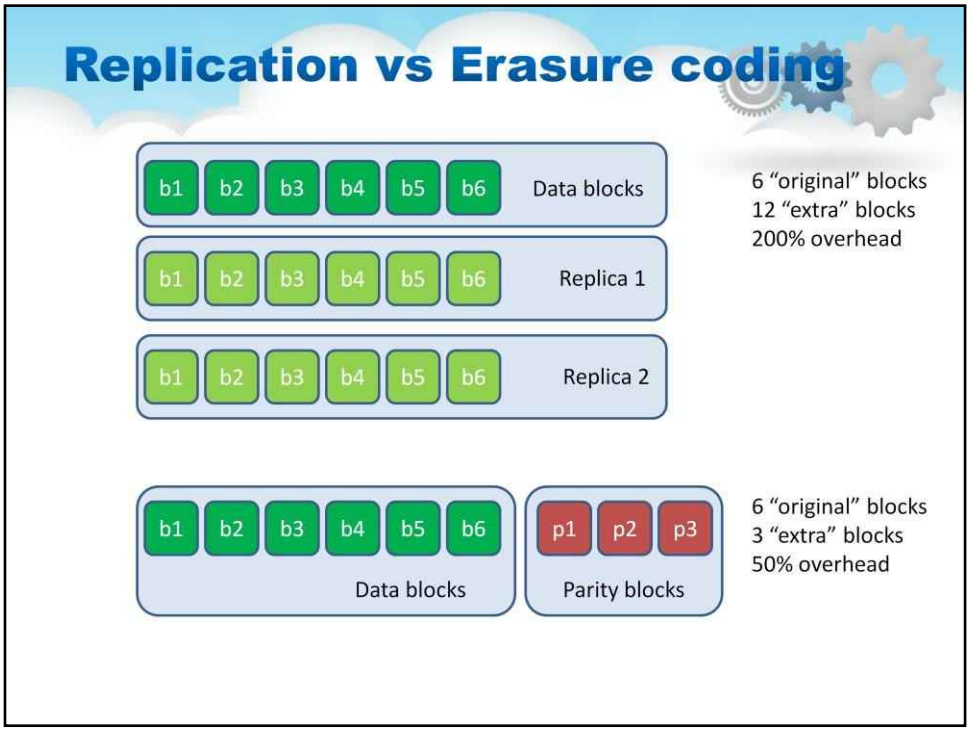
Відмінності версій

Ось порівняння Hadoop 1.x із таблицею Hadoop 2.x.

Основні відмінності полягають у тому, що Hadoop 2 використовує YARN для керування ресурсами, підтримує завдання як MapReduce, так і не-MapReduce, більш масштабований (до 10000 вузлів) і більш відмовостійкий. Він також може працювати в Windows.



Порівняно з 2.x, Hadoop 3.x вимагає менше місця для зберігання тих самих даних (через кодування стирання, а не дублювання), є більш масштабованим і має балансувальник внутрішнього вузла даних.



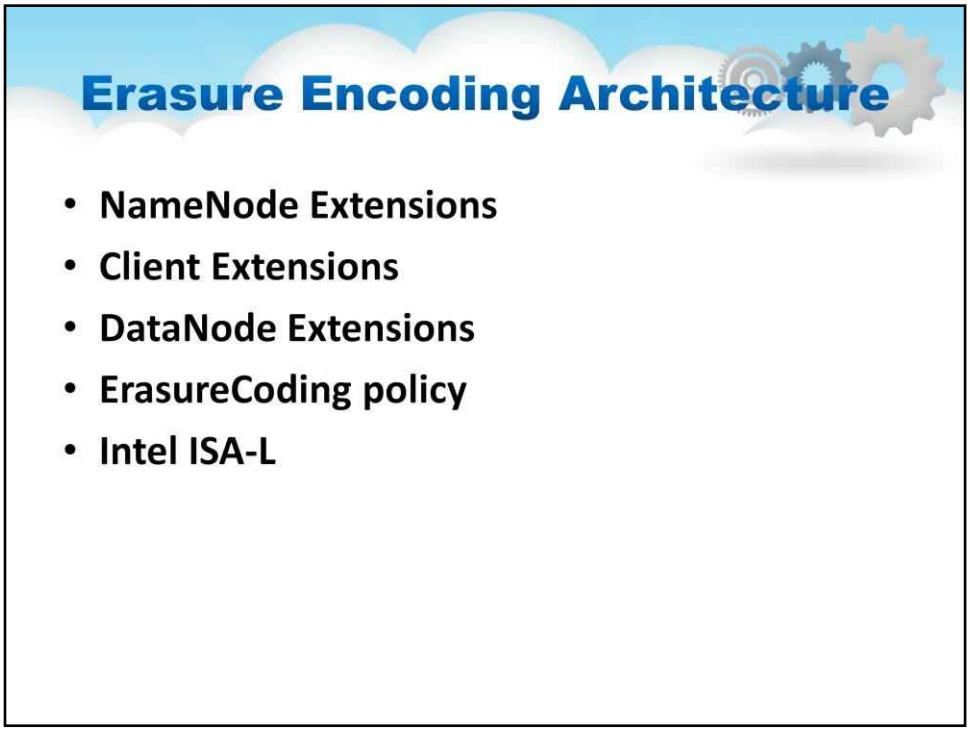
тиражування

Реплікація дорога – стандартна схема реплікації 3x у HDFS має 200% витрат на простір для зберігання та інші ресурси (наприклад, пропускну здатність мережі). Однак для теплих і холодних наборів даних із відносно низькою активністю введення-виведення додаткові репліки блоків рідко доступні під час звичайних операцій, але споживають таку ж кількість ресурсів, як і перша репліка.

Тому природним удосконаленням є використання Erasure Coding (EC) замість реплікації, що забезпечує той самий рівень відмовостійкості зі значно меншим обсягом пам’яті. У типових налаштуваннях Erasure Coding (EC) накладні витрати на зберігання не перевищують 50%. Фактор реплікації файлу EC не має сенсу. Він завжди дорівнює 1 і не може бути змінений за допомогою команди -setrep.

У системах зберігання найбільш помітним використанням EC є резервний масив недорогих дисків (RAID). RAID реалізує EC через смугу, яка розділяє логічно послідовні дані (наприклад, файл) на менші одиниці (наприклад, біт, байт або блок) і зберігає послідовні одиниці на різних дисках. У решті цього посібника ця одиниця розподілу смуги називається коміркою (або коміркою) смуги. Для кожної смуги оригінальних комірок даних обчислюється та зберігається певна кількість комірок парності – процес цього називається кодуванням. Помилка в будь-якій комірці смуги може бути відновлена шляхом обчислення декодування на основі збережених даних і клітинок парності.

Інтеграція EC із HDFS може підвищити ефективність зберігання, забезпечуючи при цьому таку ж довговічність даних, як традиційні розгортання HDFS на основі реплікації. Як приклад, трикратний реплікований файл із 6 блоками займатиме 6*3 = 18 блоків дискового простору. Але з розгортанням EC (6 даних, 3 паритету) він займатиме лише 9 блоків дискового простору.



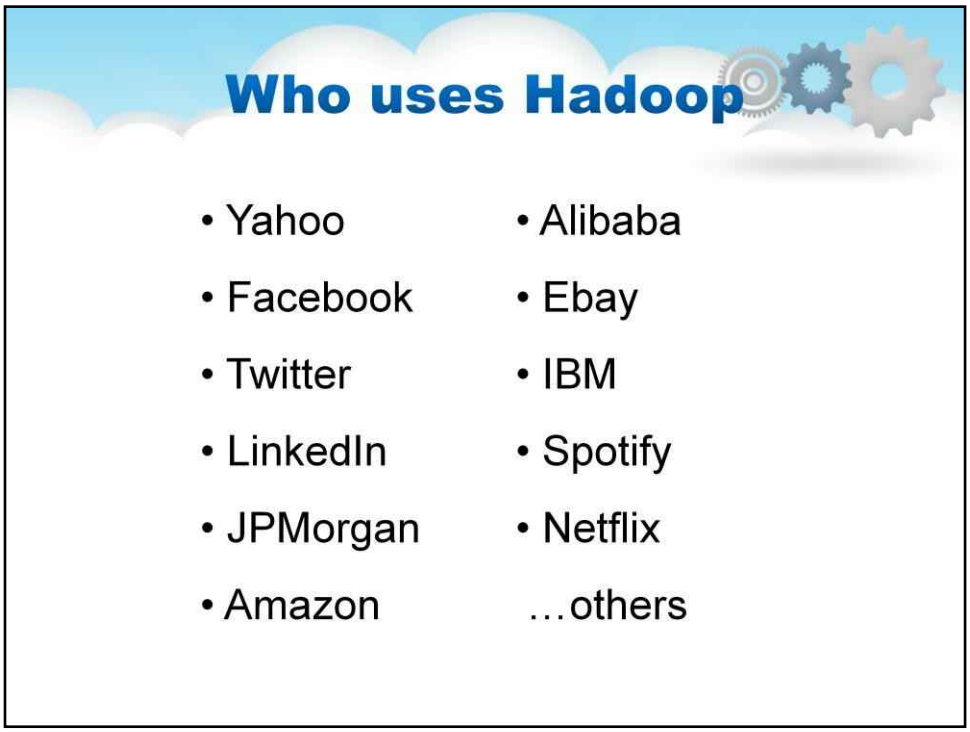
Архітектура кодування стирання HDFS

Розширення NameNode–Файли HDFS розбиті на групи блоків, які мають певну кількість внутрішніх блоків. Тепер, щоб зменшити споживання пам'яті NameNode з цих додаткових блоків, створено нову ієрархічну **протокол іменування блоків**, що введено. Ідентифікатор групи блоків можна вивести з ідентифікатора будь-якого з її внутрішніх блоків. Це дозволяє керувати на рівні групи блоків, а не блоку.

Клієнтські розширення–Після впровадження Erasure Encoding у HDFS, NameNode працює на рівні групи блоків, а шляхи читання та запису клієнта були покращені для роботи з кількома внутрішніми блоками в групі блоків паралельний. На шляху виведення/запису DFSStripedOutputStream керує набором потоків даних, по одному для кожного DataNode, що зберігає внутрішній блок у поточній групі блоків. Координатор бере на себе відповідальність за операції над усією групою блоків, включаючи завершення поточної групи блоків, виділення нової групи блоків тощо. Під час введення/читання шлях,DFSStripedInputStream перетворює запитуваний логічний діапазон байтів даних як діапазони у внутрішні блоки, що зберігаються на DataNodes. Потім він паралельно надсилає запити на читання. У разі збою він видає додаткові запити на читання для декодування.

Розширення DataNode –DataNode запускає додаткове завдання ErasureCodingWorker (ECWorker) для фонового відновлення кодованих блоків, які не вдалося видалити. Невдалі блоки EC виявляються NameNode, який потім вибирає DataNode для виконання роботи з відновлення. Реконструкція виконує три ключові завдання: читає дані з вузлів джерела та зчитує лише мінімальну кількість вхідних блоків і блоків парності для реконструкції. Нові дані та блоки парності декодуються з вхідних даних. Усі відсутні дані та блоки парності декодуються разом. Після завершення декодування відновлені блоки передаються до цільових DataNodes.

Політика ErasureCoding–Щоб адаптувати різнорідні робочі навантаження, ми дозволяємо файлам і каталогам у кластері HDFS мати різні політики реплікації та EC. Інформація про кодування та декодування файлів міститься в класі ErasureCodingPolicy. Він містить 2 частини інформації, тобто Схема ECS і розмір комірки очищення.**Intel ISA-L**Intel ISA-L означає Intelligent Storage Acceleration Library. ISA-L — це колекція оптимізованих низькорівневих функцій із відкритим вихідним кодом, розроблена для програм зберігання даних. Він містить швидкі блокові коди стирання типу Ріда-Соломона, оптимізовані для наборів інструкцій Intel AVX і AVX2. Кодування стирання HDFS може використовувати ISA-L для прискорення обчислень кодування та декодування. ISA-L підтримує більшість основних операційних систем, включаючи Linux і Windows. ISA-L не ввімкнено за замовчуванням. Перегляньте інструкції нижче, щоб увімкнути ISA-L



Використання

Yahoo:Використовується для тестів масштабування.

Facebook:Використовується як джерело для звітності та машинного навчання.

Twitter:Щоб зберігати та обробляти твіти, файли журналів за допомогою LZO, яка є портативною бібліотекою стиснення даних без втрат, написаною мовою ANSI C. Це швидко та

також допомагає звільнити ЦП для інших завдань.

LinkedIn:Дані LinkedIn передаються через кластери Hadoop. Діяльність користувачів, показники сервера, зображення, журнали транзакцій, що зберігаються в HDFS, використовуються даними

аналітики для бізнес-аналітики, як виявлення людей, яких ви можете знати.

JPMorgan:Аналітика операцій клієнтів.

Amazon:Обробка даних шляхом аналізу відгуків і вимог клієнтів. **Adobe:**Соціальні сервіси для структурованого зберігання даних. **Ebay:**З 300+ мільйонами користувачів, які переглядають понад 350 мільйонів продуктів, перелічених на їхньому веб-сайті, eBay має один із найбільших кластерів Hadoop

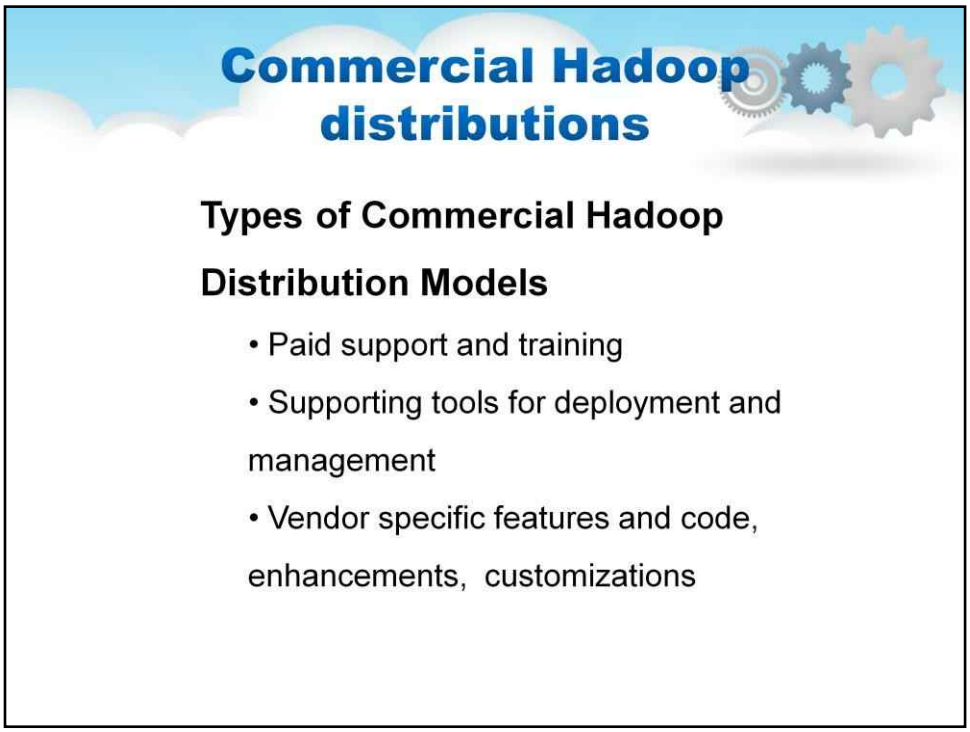
в галузі, яка займає важливе місце в MapReduce Jobs.
Hadoop використовується eBay для пошукової оптимізації та досліджень.

Netflix:Для прийняття рішень.

Aol:Націлено на машини та подвійні процесори.

Alibaba:Аналізує вертикальну пошукову систему.

IBM:Клієнтські проекти у сфері фінансів, телекомунікацій та роздрібної торгівлі, машинне навчання з Watson Analytics.
Infosys:Клієнтські проекти у сфері фінансів, телекому та роздрібної торгівлі. **TCS:** Клієнтські проекти у сфері фінансів, телекому та роздрібної торгівлі. **Spotify:** Використовується для створення вмісту та агрегації даних.



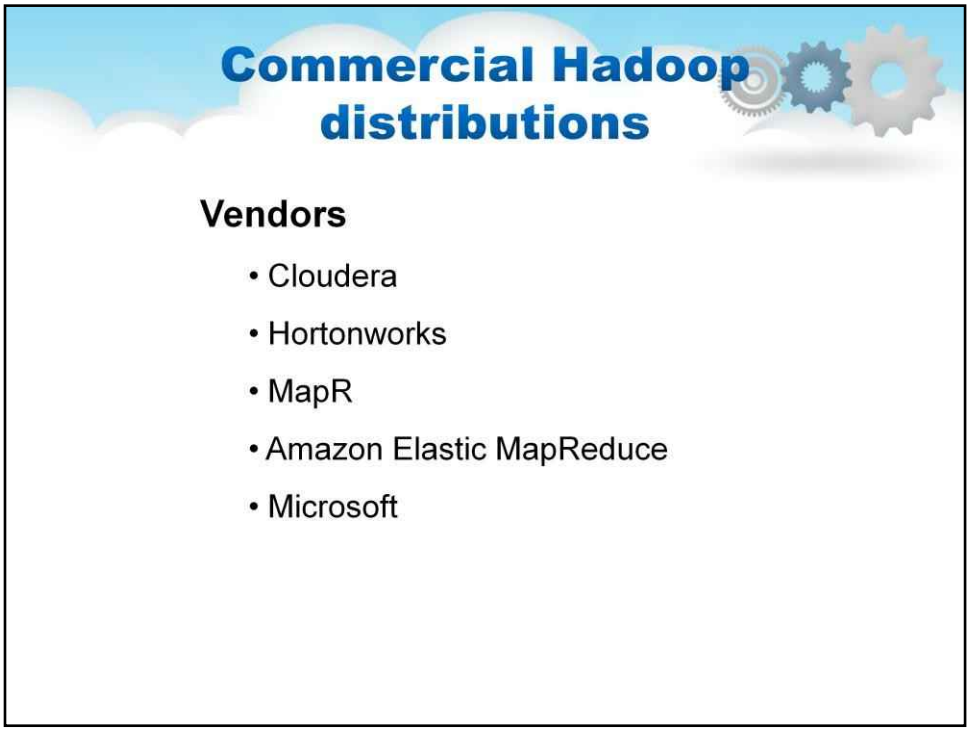
Типи комерційних моделей розповсюдження Hadoop

Щоб задовольнити потреби підприємств у розгортанні Hadoop для приборкання великих даних, кілька компаній розробили комерційні моделі розповсюдження Hadoop. Комерційні дистрибутиви Hadoop в основному поділяються на три основні види. Вони такі:

Дистрибутиви, які надають платну підтримку та навчання для Apache Hadoop (наприклад, Cloudera, HortonWorks, MapR, IBM тощо).

Дистрибутиви, які пропонують набір допоміжних інструментів для розгортання та керування Apache Hadoop як альтернативу (наприклад, Cloudera, HortonWorks, MapR).

Дистрибутиви, які дозволяють додавати специфічні функції та код постачальника, платні вдосконалення, щоб покращити або налаштувати розгортання Apache Hadoop і узгодити його з потребами бізнесу (наприклад, Cloudera, HortonWorks, MapR, IBM тощо). Тепер головне питання полягає в тому, як ви вибрати дистрибутив Hadoop з численних варіантів, доступних на ринку? Давайте розглянемо деякі критерії, які можуть допомогти вам вибрати відповідний для вас дистрибутив Hadoop.



Клаудера

Cloudera, американський постачальник програмного забезпечення та рішень для технології Apache Hadoop і перший постачальник, який запропонував Hadoop як пакет, продовжує залишатися лідером на ринку дистрибутивів Hadoop. CDH від Cloudera, який містить усі компоненти з відкритим кодом, націлений на розгортання корпоративного класу та є одним із найпопулярніших комерційних дистрибутивів Hadoop.

Відома своїми інноваціями Cloudera першою запропонувала SQL-for-Hadoop із механізмом запитів Impala. Інші доповнення Cloudera включають безпеку, інтерфейс користувача та інтерфейси для інтеграції з програмами сторонніх розробників. Cloudera підтримує його розповсюдження через службу передплати Cloudera Enterprise.

Hortonworks

Hortonworks розробляє та підтримує Apache Hadoop для розподіленої обробки великих наборів даних у комп'ютерних кластерах. Платформа даних Hortonworks (HDP) — це платформа з повністю відкритим кодом, призначена для маневрування даними з багатьох джерел і форматів. Платформа включає різні технології Hadoop, такі як Hadoop Distributed File System, MapReduce, Zookeeper, HBase, Pig і Hive, а також додаткові компоненти.

Hortonworks відома тим, що купує інші компанії з корисним кодом і випускає код у спільноту з відкритим кодом. Нова тенденція до консолідації на ринку призвела до зростання популярності продукції Hortonworks. Нещодавно і Amazon, і IBM почали пропонувати Hortonworks як опцію на своїх власних платформах поряд із власними дистрибутивами Hadoop. HDP також є ядром Open Data Platform Initiative, групи, яка прагне спростити та стандартизувати специфікації в екосфері великих даних.

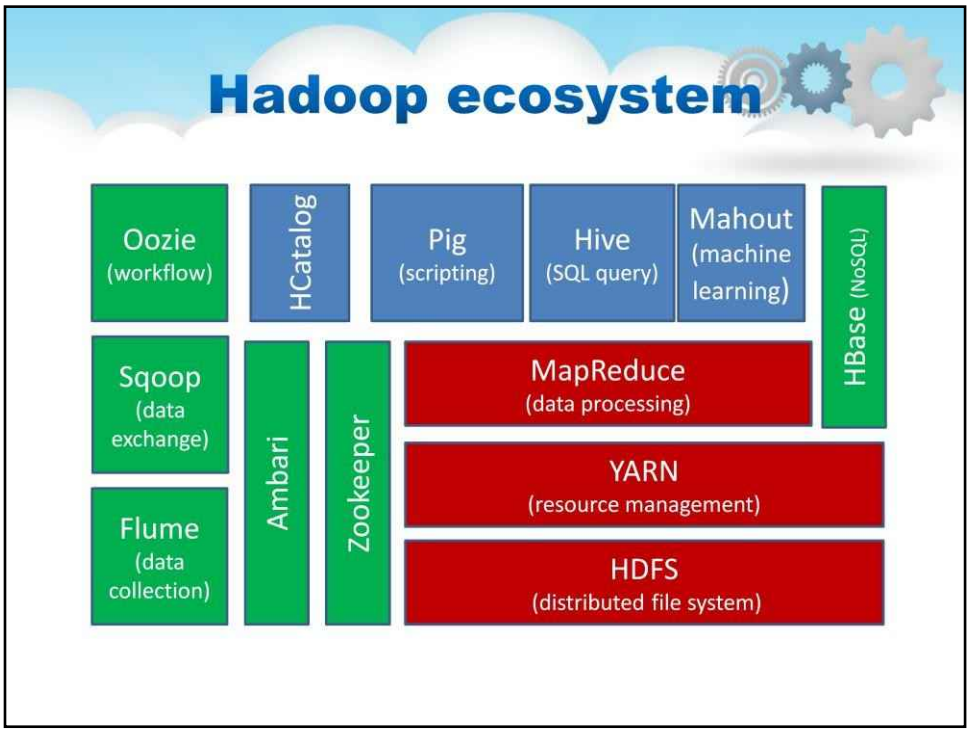
MapR

Замість керованого постачальника послуг, як Amazon і Microsoft, MapR є орієнтованим на платформу постачальником рішень Hadoop, як Hortonworks і Cloudera. MapR інтегрує власну систему баз даних, відому як MapR-DB, одночасно пропонуючи послуги розповсюдження Hadoop. Стверджується, що MapR-DB працює в чотири-сім разів швидше, ніж базова база даних Hadoop, HBase, яка працює в інших дистрибутивах. Завдяки своїй швидкості MapR часто вважають кращим вибором для великих проектів Big Data. **Amazon Elastic MapReduce**

Amazon пропонує модель оплати за використання на хмарній платформі. Він надає платформу Hadoopas-a-Service через підрозділ Amazon Web Services. Ключовою перевагою розрахункової моделі є масштабованість. Ця модель дозволяє збільшувати або зменшувати масштаб у міру зміни вимог. Amazon Elastic MapReduce також легко підключається до іншої інфраструктури хмарних сервісів Amazon, як-от Amazon S3 і DynamoDB для зберігання, EC2 для хмарної обробки та AWS IoT для збору даних із пристроїв із підтримкою Інтернету речей.

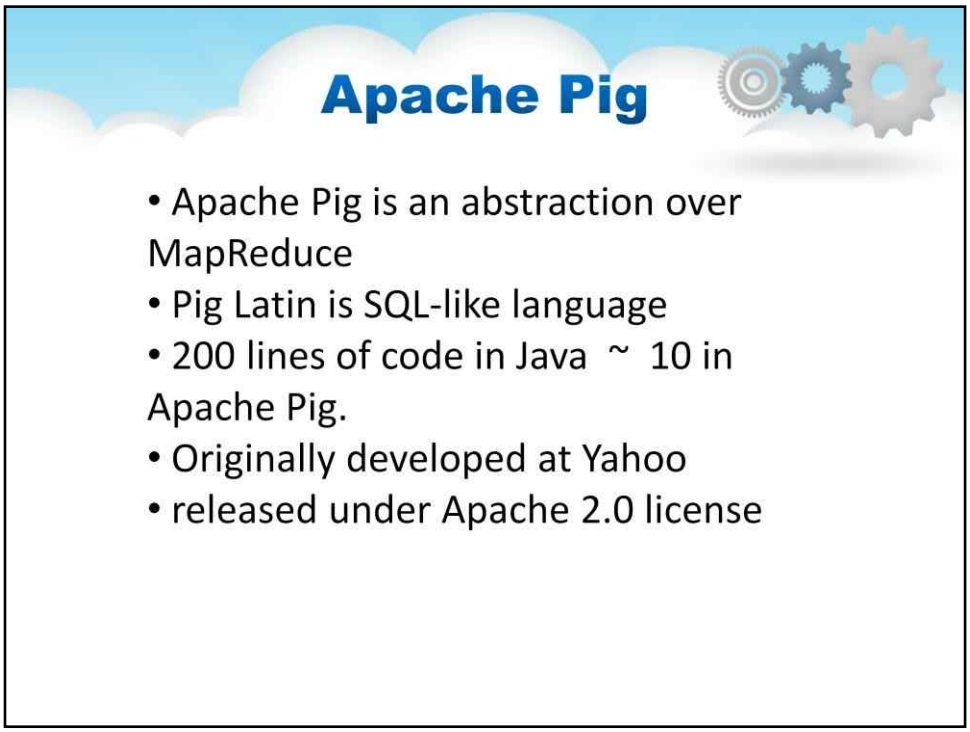
Microsoft

Microsoft також пропонує лише хмарну службу у формі платформи Azure HDInsight, яка пропонує керовані інсталяції кількох відкритих дистрибутивів Hadoop, зокрема Cloudera, Hortonworks і MapR. HDInsight інтегрує різні дистрибутиви Hadoop із власною платформою Azure Data Lake, щоб забезпечити повне рішення для хмарного зберігання та аналітики. Крім того, HDInsights надає хмарні сервіси Hive, Spark, Kafka і Storm разом із власною хмарною системою безпеки. Вибір правильного розповсюдження Hadoop повністю залежить від перешкод і проблем, з якими стикається організація під час впровадження Hadoop на підприємстві. Кожен комерційний дистрибутив Hadoop має свої плюси та мінуси.
Тому важливо враховувати ризик і вартість разом із додатковою цінністю, яку пропонує кожен дистрибутив Hadoop, щоб дистрибутив виявився корисним для потреб вашого бізнесу.



Hadoop HDFS – рівень розподіленого сховища для Hadoop.
Yarn Hadoop – рівень керування ресурсами,
представлений у Hadoop 2.x. Hadoop Map-Reduce – рівень
паралельної обробки для Hadoop.
HBase – це база даних, орієнтована на стовпці, яка працює поверх HDFS. Це база даних NoSQL, яка не розуміє структурований запит. Для розрідженого набору даних це добре підходить.

Hive – Apache Hive – це інфраструктура сховища даних на основі Hadoop, яка дозволяє легко узагальнювати дані за допомогою запитів SQL.
Pig – мова сценаріїв верхнього рівня. Оскільки ми використовуємо його з Hadoop. Pig дозволяє писати складну обробку даних без програмування на Java.
Flume – це надійна система для ефективного збору великих обсягів даних журналу з різних джерел у режимі реального часу.
Sqoop – це інструмент для передачі величезних обсягів даних між Hadoop і RDBMS.
Oozie – це веб-програма Java, яка використовується для планування завдань Apache Hadoop. Він об'єднує декілька завдань послідовно в одну логічну одиницю роботи.
Zookeeper – централізована служба для підтримки конфігураційної інформації, іменування, забезпечення розподіленої синхронізації та надання групових послуг. Mahout – бібліотека масштабованих алгоритмів машинного навчання, реалізована на основі Apache Hadoop і використовує парадигму MapReduce

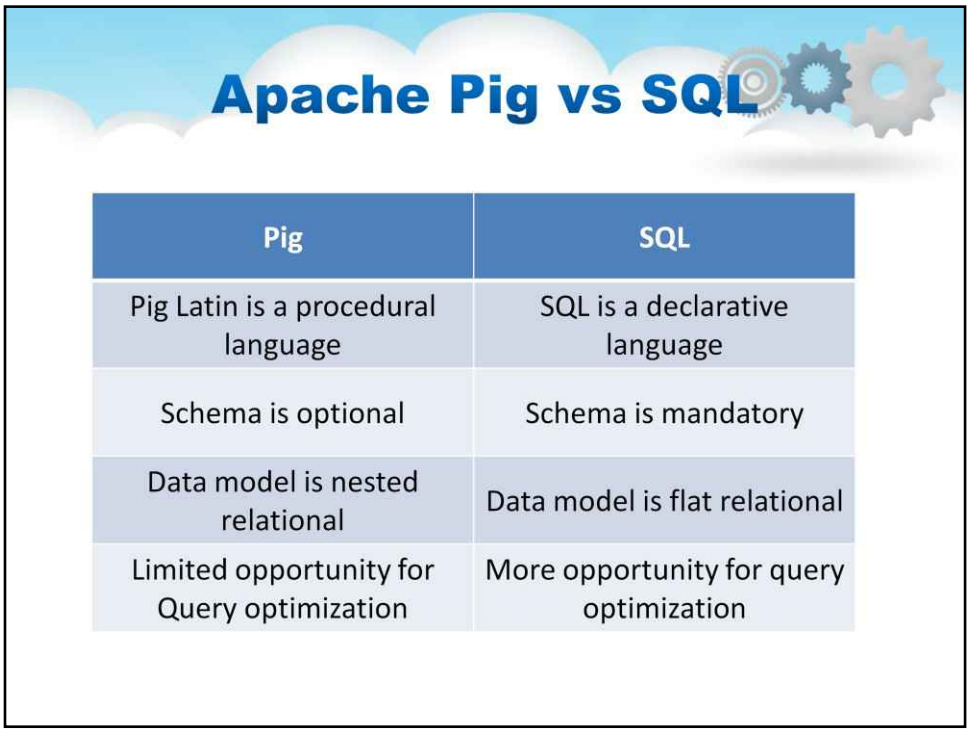


Свиня Апачі

Apache Pig — це платформа для аналізу великих наборів даних, абстракція над MapReduce. Для написання програм аналізу даних Pig надає мову високого рівня, відому як **Свиняча латинська**.

Pig Latin дозволяє вказати послідовність перетворень даних, наприклад об’єднання наборів даних, їх фільтрування та застосування функцій до записів або груп записів. Pig має багато вбудованих функцій, але ви також можете створювати власні функції, визначені користувачем, для обробки спеціального призначення.

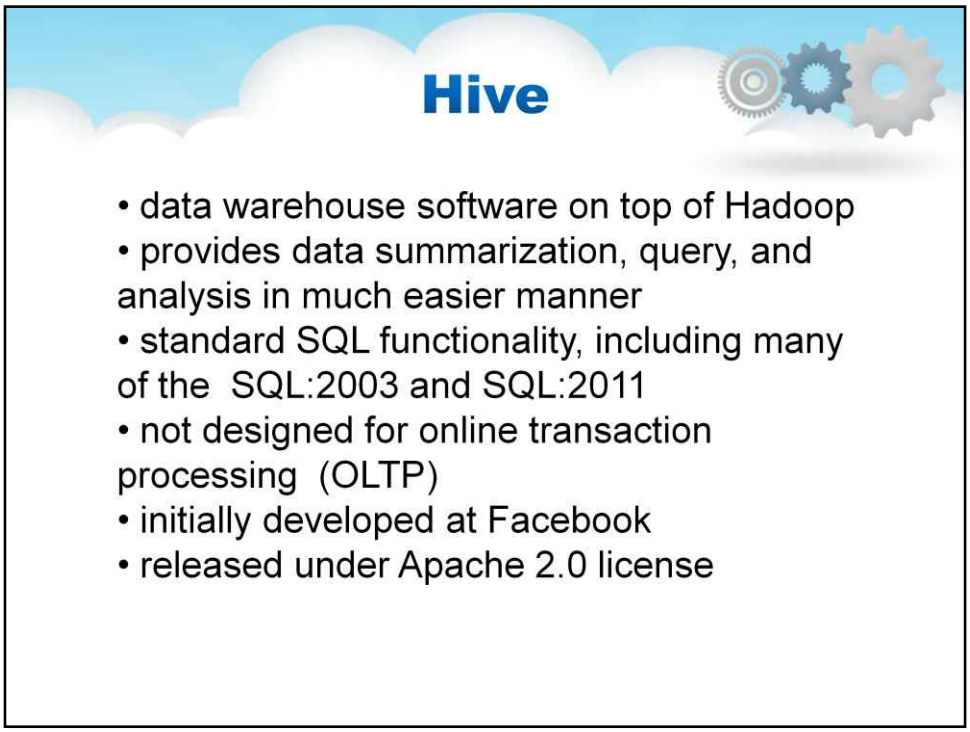
Програми Pig Latin виконуються розподіленим способом у кластері (програми поєднуються в завдання Map/Reduce і виконуються за допомогою Hadoop).



Apache Pig — це платформа для аналізу великих наборів даних, абстракція над MapReduce. Для написання програм аналізу даних Pig надає мову високого рівня, відому як **Свиняча латинська**.

Pig Latin дозволяє вказати послідовність перетворень даних, наприклад об’єднання наборів даних, їх фільтрування та застосування функцій до записів або груп записів. Pig має багато вбудованих функцій, але ви також можете створювати власні функції, визначені користувачем, для обробки спеціального призначення.

Програми Pig Latin виконуються розподіленим способом у кластері (програми поєднуються в завдання Map/Reduce і виконуються за допомогою Hadoop).



Apache Hive

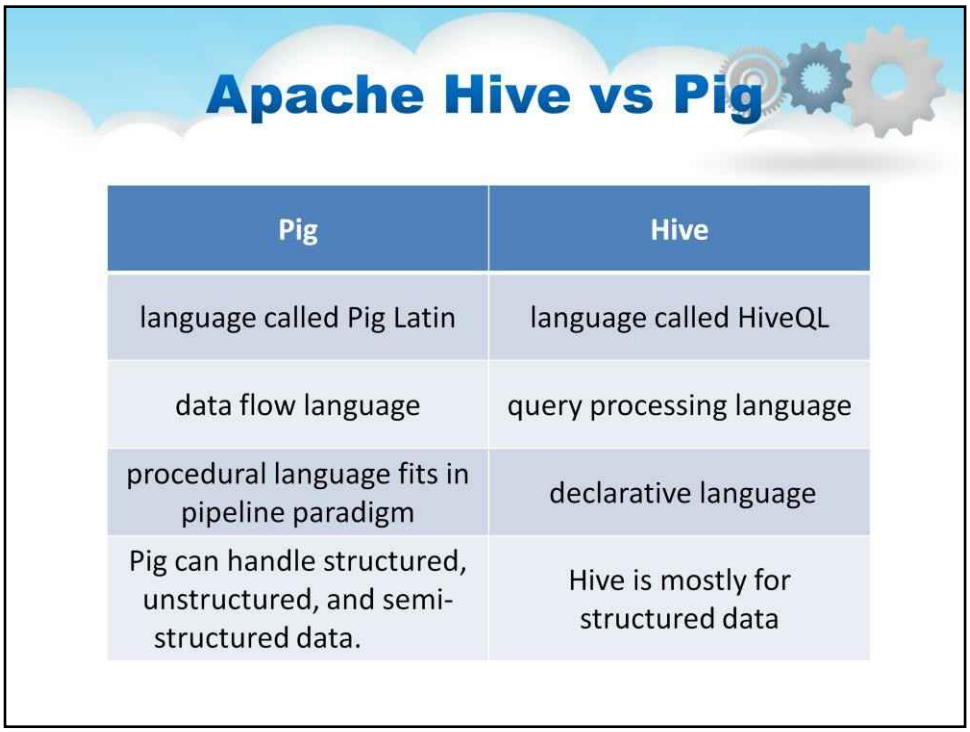
Програмне забезпечення сховища даних Apache Hive полегшує читання, запис і керування великими наборами даних, які зберігаються в розподіленому сховищі та запитуються за допомогою синтаксису SQL.

забезпечує підсумовування даних, запити та аналіз у набагато простіший спосіб, підтримує зовнішні таблиці, які дають змогу обробляти дані без фактичного зберігання в HDFS.

Використання HiveQL не вимагає знання мови програмування, достатньо знання базових запитів SQL. Hive надає стандартну функціональність SQL, включаючи багато пізніших функцій SQL:2003 і SQL:2011 для аналітики.

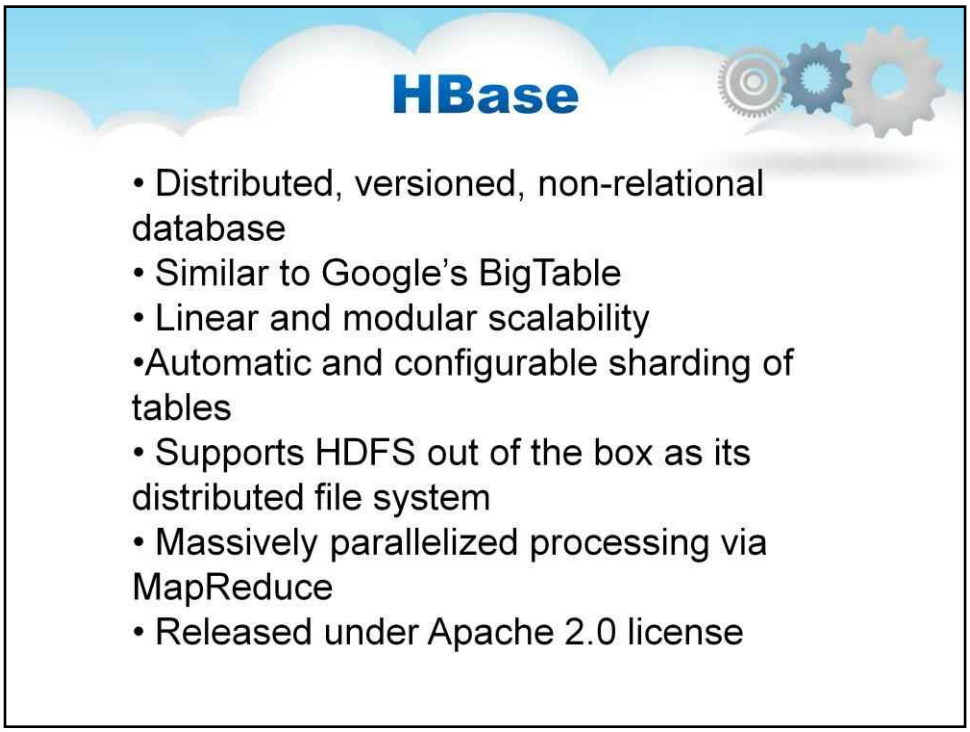
SQL Hive також можна розширити за допомогою коду користувача за допомогою визначених користувачем функцій Hive не розроблено для робочих навантажень обробки онлайнових транзакцій (OLTP). Його найкраще використовувати для традиційних завдань зі сховищ даних.

Hive розроблено для максимального збільшення масштабованості (розширення з більшою кількістю машин, які динамічно додаються до кластера Hadoop), продуктивності, розширюваності, відмовостійкості та слабкого зв’язку з форматами введення.



Ось порівняльна таблиця Apache Pig і Hive. Apache Pig використовує мову під назвою Pig Latin. Спочатку він був створений в Yahoo. Hive використовує мову HiveQL.
Спочатку він був створений у Facebook.
Pig Latin — мова потоку даних. HiveQL — це мова обробки запитів. Pig Latin — це процедурна мова, яка відповідає конвеєрній парадигмі. HiveQL є декларативною мовою.

Apache Pig може обробляти структуровані, неструктуровані та напівструктуровані дані. Вулик здебільшого призначений для структурованих даних.



HBase

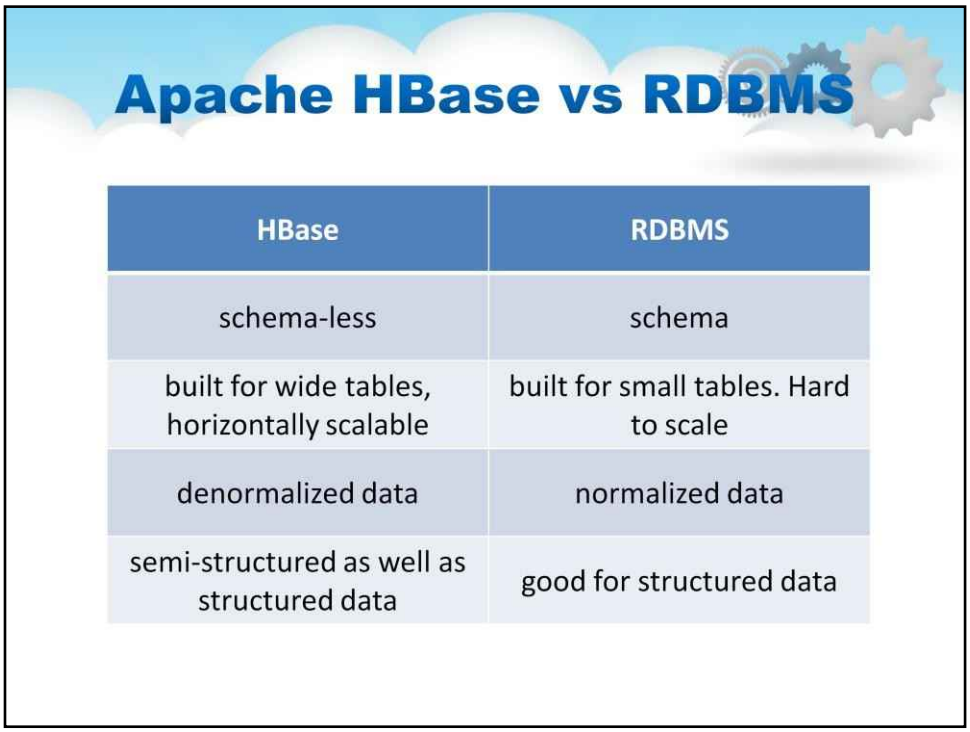
HBase — це розподілена нереляційна база даних із відкритим вихідним кодом, керована версіями, створена за зразком статті Google Bigtable. Подібно до того, як Bigtable використовує розподілене сховище даних, яке забезпечує файлова система Google, Apache HBase надає можливості, подібні до Bigtable, на додаток до Hadoop і HDFS.

Лінійна та модульна масштабованість.

Автоматичне та настроюване шардування таблиць

Інтеграція Hadoop/HDFS: HBase підтримує HDFS із коробки як свою розподілену файлову систему.

MapReduce: HBase підтримує масову паралелізовану обробку через MapReduce для використання HBase як джерела, так і приймача.

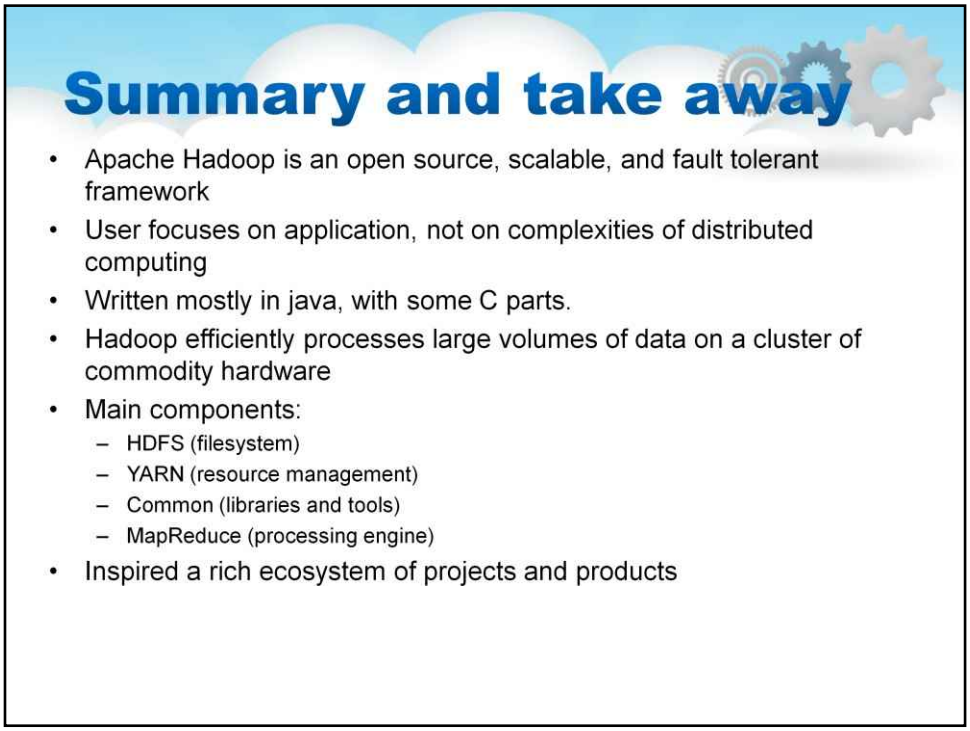


Ось порівняльна таблиця HBase та RDBMS.

HBase не має схем, він не має концепції схеми фіксованих стовпців; визначає лише сімейства стовпців. РСУБД керується своєю схемою, яка описує всю структуру таблиць.

HBase створений для широких таблиць і горизонтально масштабований. RDBMS тонка і створена для невеликих таблиць. Важко масштабувати. У HBase немає транзакцій, але більшість RDBMS є транзакційними.

HBase має денормалізовані дані, але RDBMS має нормалізовані дані. HBase підходить як для напівструктурованих, так і для структурованих даних, а RDBMS — для структурованих даних.



Apache Hadoop — це масштабована та відмовостійка

структура з відкритим кодом

Hadoop в основному написаний на java. Деякі його частини написані мовою C.

Користувач зосереджується на застосуванні, а не на складності

розподілені обчислення

Hadoop ефективно обробляє великі обсяги даних на кластері

стандартного обладнання Основні компоненти:

HDFS (файлова система)

YARN (керування ресурсами) Common

(бібліотеки та інструменти) MapReduce

(процесор)

Apache Hadoop надихнув багату екосистему проектів і продуктів, які приносять користь усім, хто цікавиться великими даними

Хмарні обчислення

Лекційний посібник

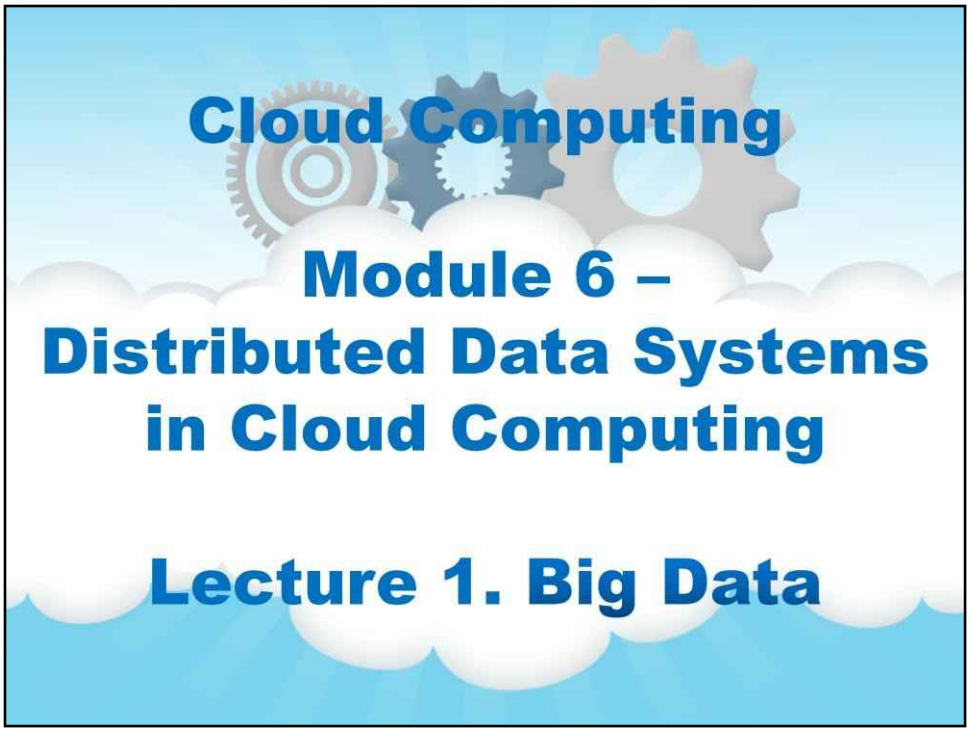
Том 6

Модуль 6

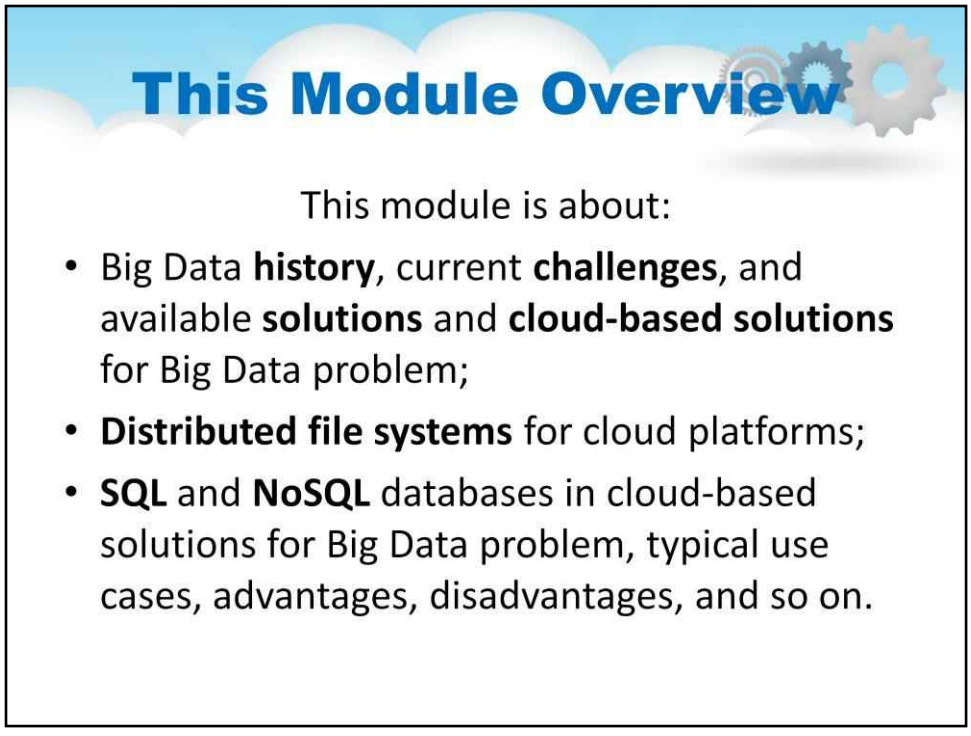
Розподілені системи даних у хмарі

обчислювальна техніка

Зміст



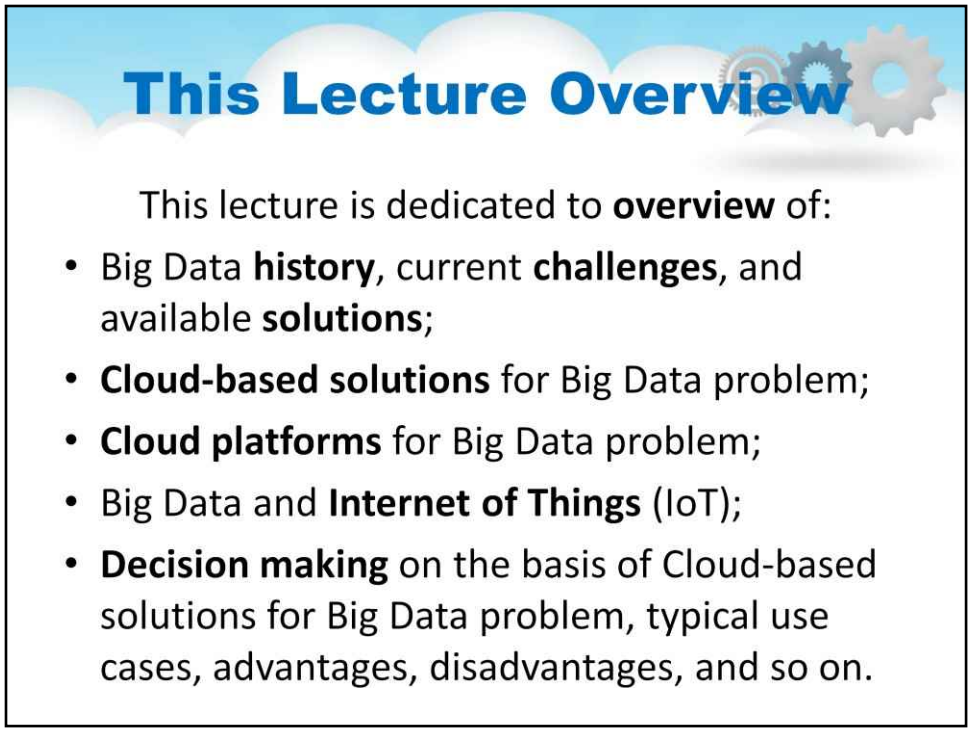
Назва цього модуля: «Розподілені системи даних у хмарних обчисленняхЦе лекція №1 про проблему Big Data та її контекст.



Модуль 6. Розподілені системи даних у хмарних обчисленнях

Цей модуль про:

- Великі дані **історії**, поточний **виклики**, і в наявності **рішення** і **хмарні рішення** для проблеми великих даних;
- **Розподілені файлові системи** для хмарних платформ;
- **SQL** і **NoSQL** бази даних у хмарних рішеннях для проблеми великих даних, типові випадки використання, переваги, недоліки тощо.



Лекція 1. Великі дані

Ця лекція присвячена**огляд**з:

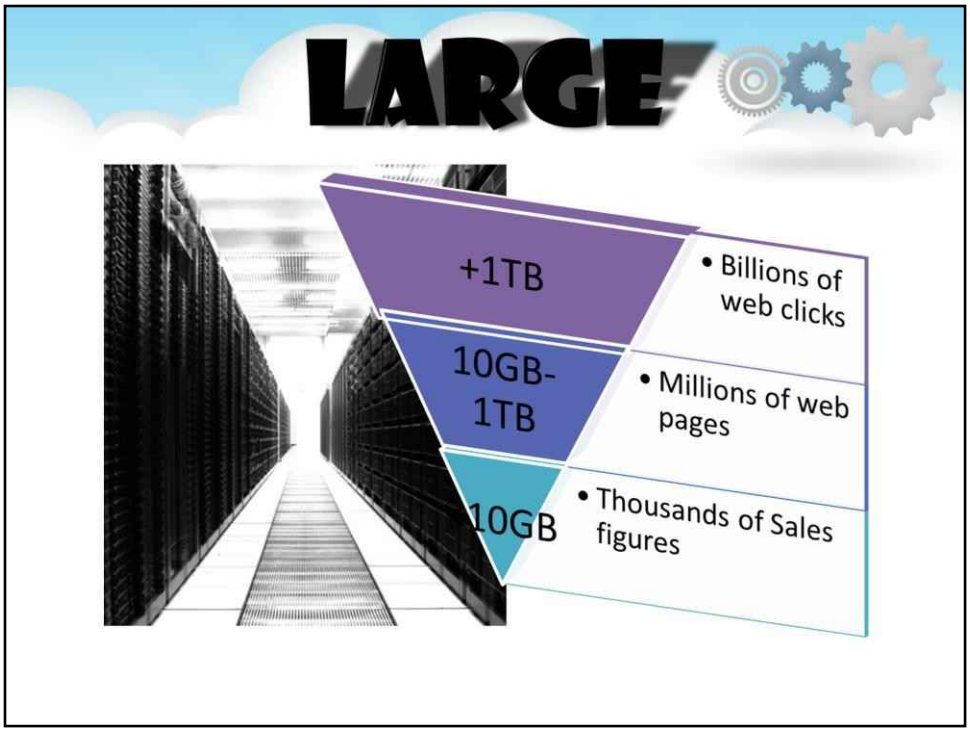
•Великі дані**історії**, поточний**виклики**, і в наявності**рішення**;
•**Хмарні рішення**для проблеми великих даних;
•**Хмарні платформи**для проблеми великих даних;
•Великі дані і**Інтернет речей**(IoT);
•**Прийняття рішень**на основі хмарних рішень для проблеми великих даних, типові випадки використання, переваги, недоліки тощо.

Огляд

Почнемо із загального погляду на проблему Big Data...

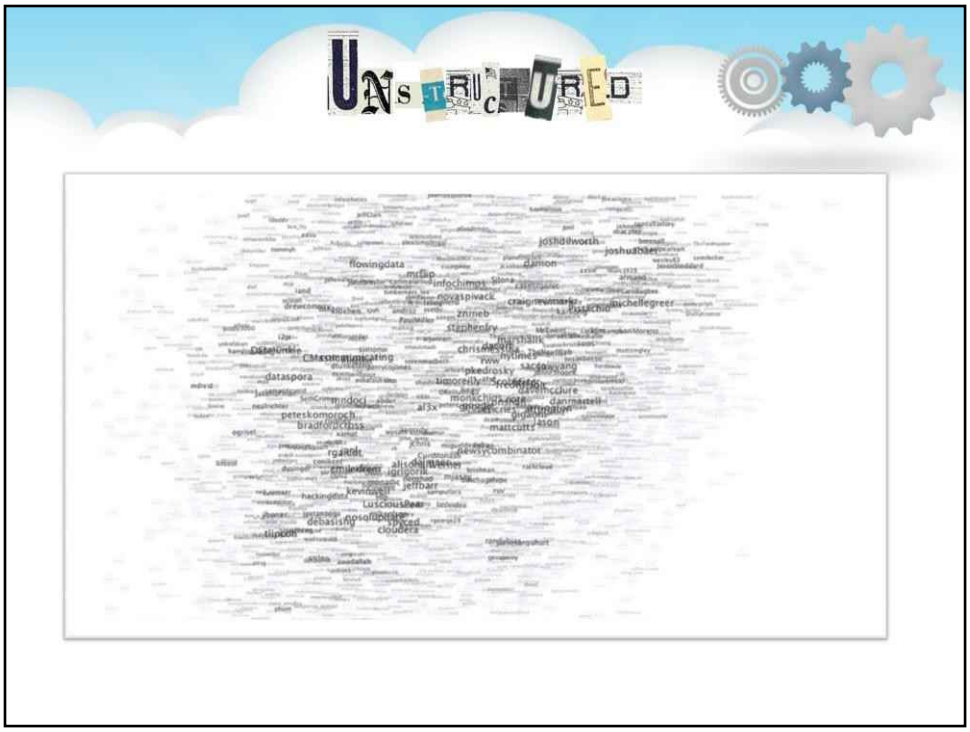
Великі дані зазвичай характеризуються як деяка маса даних, яка є:

- Великий
- Неструктурований
- Реальний час



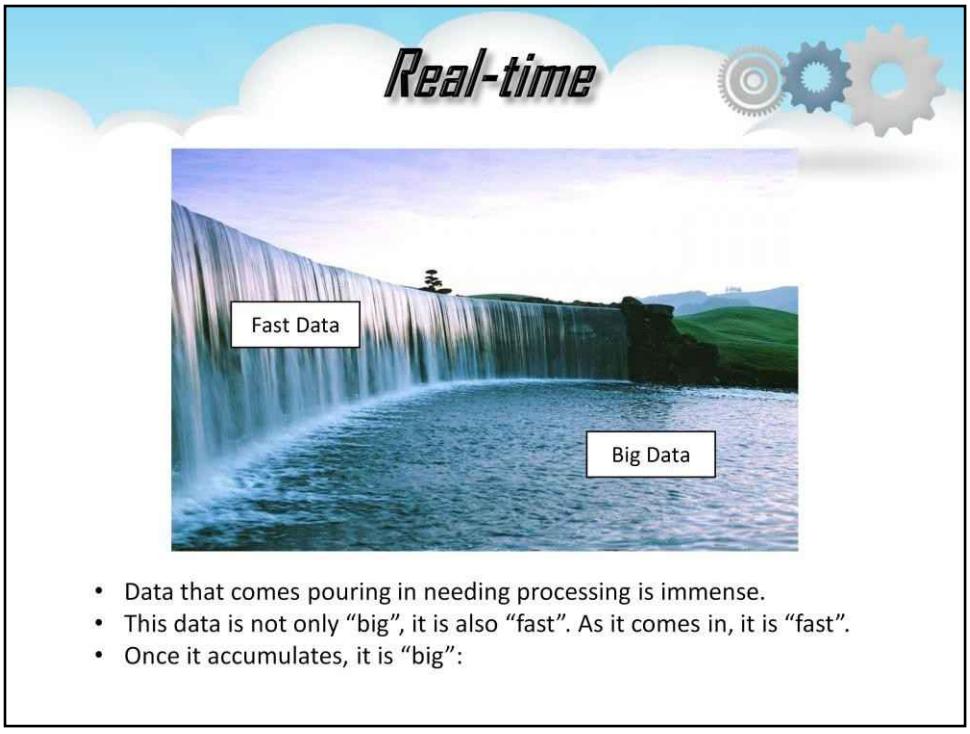
Великі дані трансформувалися з:

- Тисячі: бази даних із тисячами цифр продажів
- Мільйони: мільйони веб-сторінок
- Мільярди веб-клаців



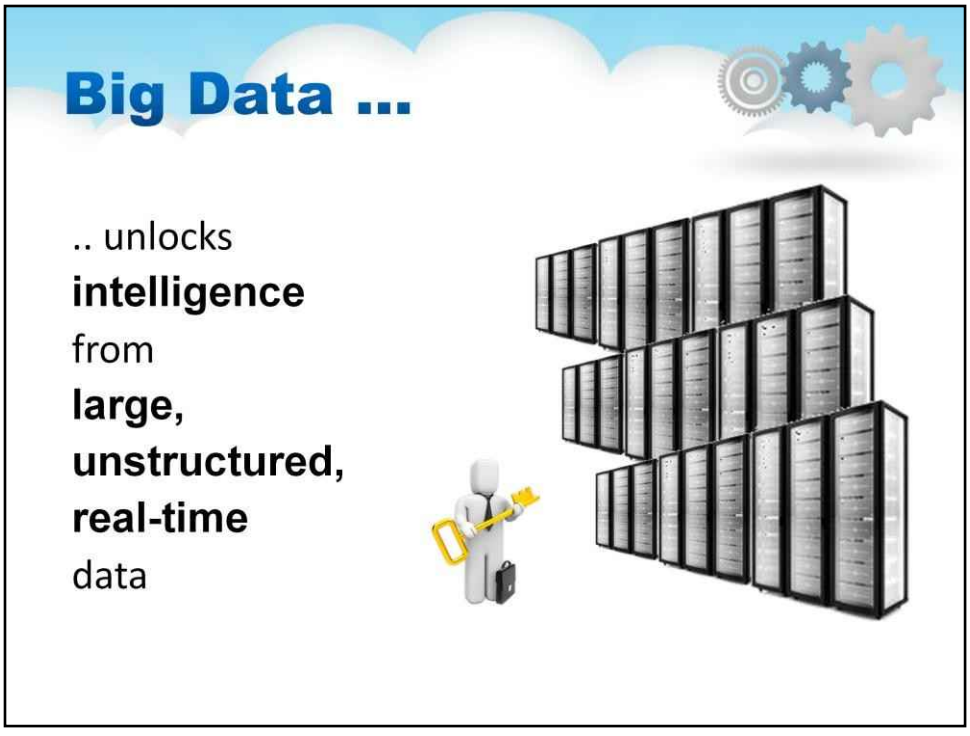
Великі дані неструктуровані

Слайд демонструє ілюстрацію веб-вмісту, який зберігається та представлений у формах, що споживаються людиною, які ми неструктуровані, що стосується машин



Великі дані працюють у реальному часі

Швидкі дані можуть бути інформацією, згенерованою програмним або апаратним забезпеченням. Це можуть бути файли журналів, набір даних у пам’яті, який швидко змінюється, дані датчиків з Інтернету речей (IoT), геопросторові дані з популяцій мобільних додатків, повідомлення електронної пошти, Twitter або SMS, інформація про погоду чи фондовий ринок або розвідувальні відомості з поля бою.



Що таке великі дані?

Великі дані відкриваються

інтелект

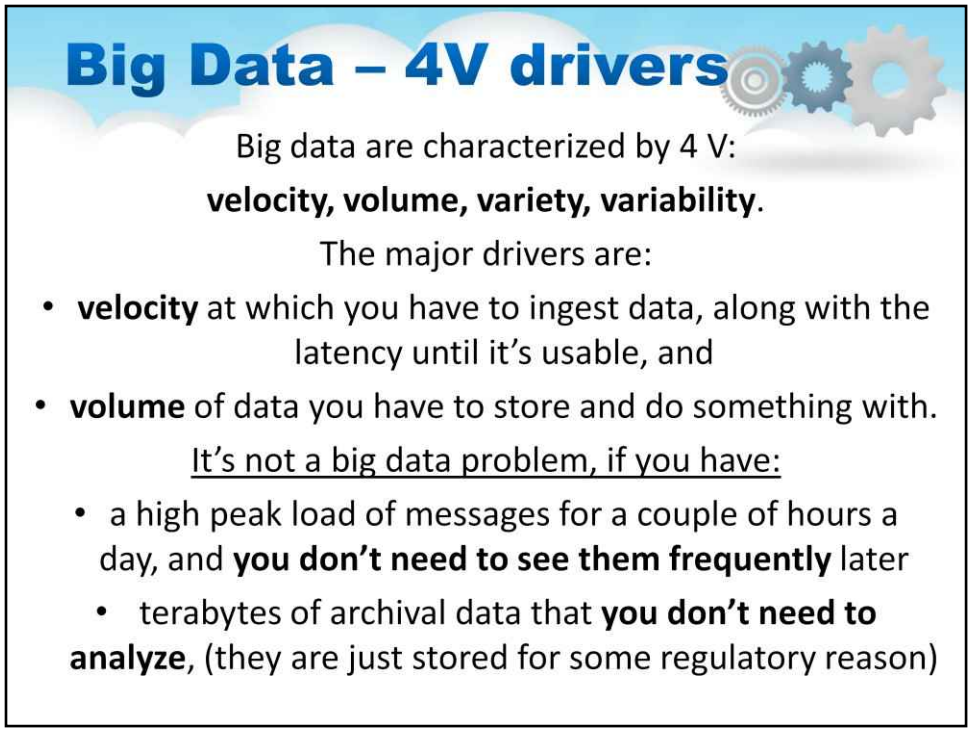
від

великий,

неструктурований,

реальний час

даних



Як було зазначено у вступній лекції 0, поточна проблема великих даних пов'язана з обробкою великого обсягу даних, створених Інтернетом людей, Інтернетом речей та Інтернетом усього.

Великі дані характеризуються**4 В:**

velocity, volume, variety, variability.

Основними драйверами є:

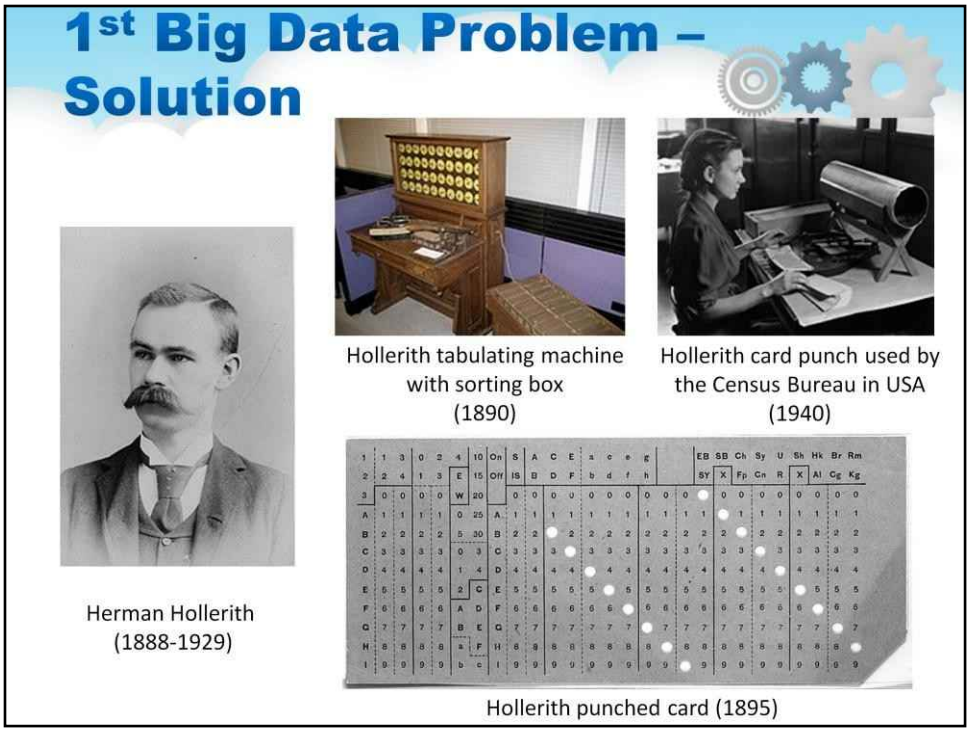
- **швидкість**коли ви повинні отримати дані разом із затримкою, поки вони не стануть придатними для використання, і
- **обсяг**даних, які потрібно зберігати та робити з ними.

Але це не велика проблема з даними, якщо у вас є:

- високе пікове навантаження повідомлень на пару годин на день, і**вам не потрібно бачити їх часто**пізніше
- терабайти архівних даних, які**не потрібно аналізувати**, (вони просто зберігаються з певних нормативних причин)

історія

Давайте розглянемо історію та еволюцію проблеми великих даних...



Згадаймо вступну лекцію 0, де ми розглядали перші великі дані проблема.

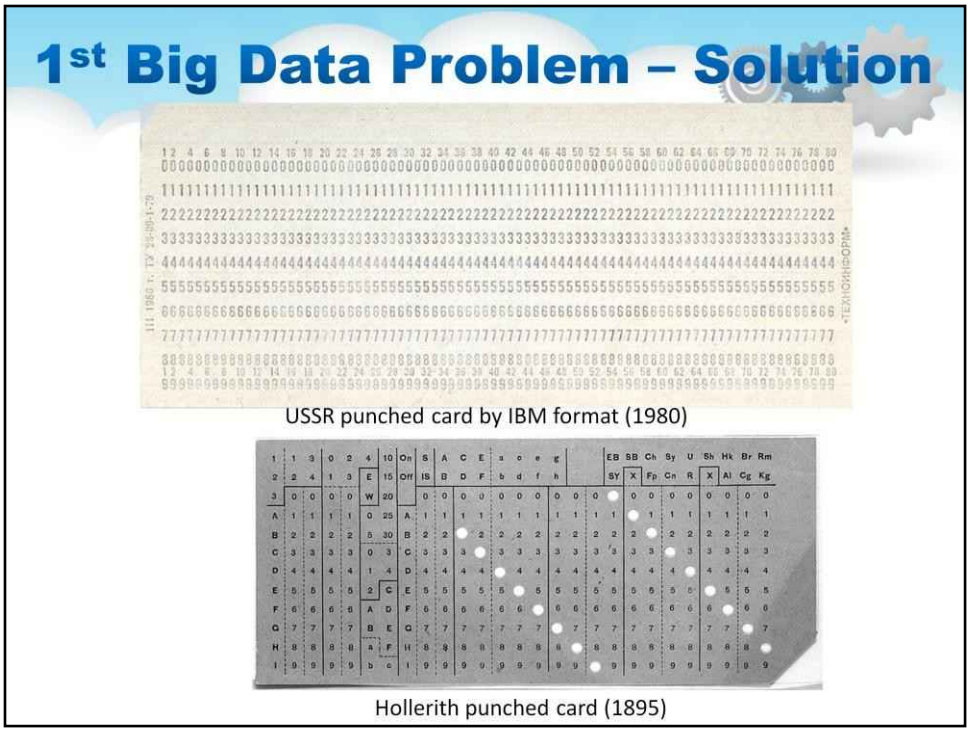
Це сталося в 1880-х роках.

Наприкінці 1800-х років обробка перепису населення США почала займати близько 10 років. Проходить перепис**кожні 10 років**і населення, а отже, і кількість інформації збільшувалося —**проблема**!

У 1886 році Герман Холлеріт відкрив бізнес з оренди машин, які могли зчитувати та зводити дані перепису населення на перфокартах. Для проведення перепису 1890 року знадобилося менше двох років, і він охопив більшу кількість населення (62 мільйони осіб) і більше даних, ніж перепис 1880 року.

перепис населення.

Пізніше бізнес Холлеріта об'єднався з трьома іншими, щоб створити компанію IBM!



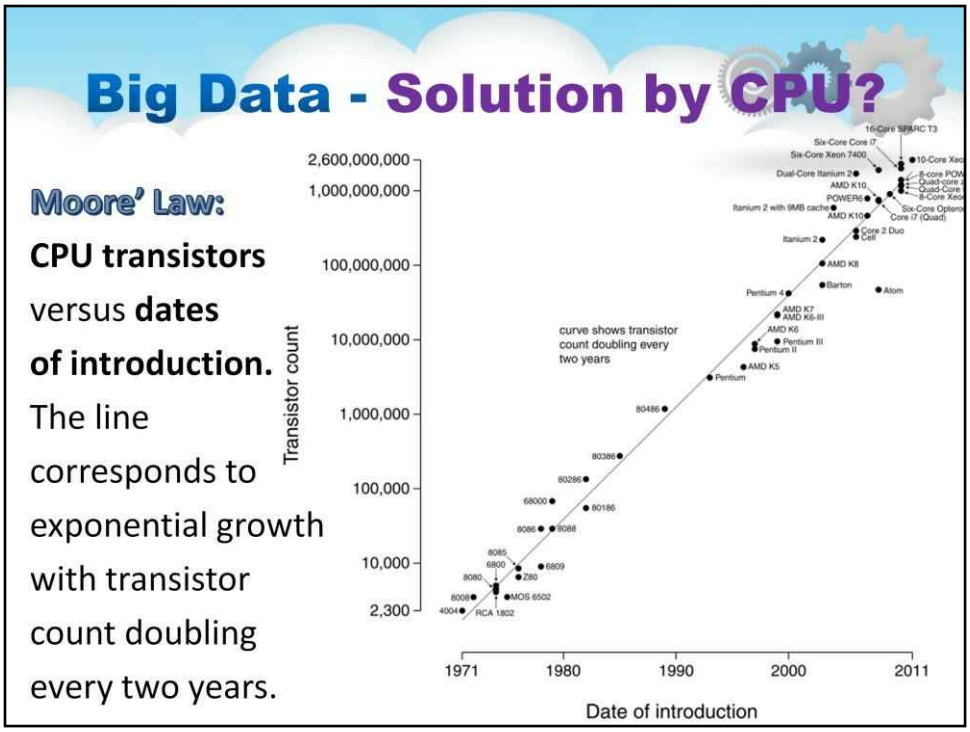
У 1886 році Герман Холлеріт відкрив бізнес з оренди машин, які могли зчитувати та зводити дані перепису населення на перфокартах. Для проведення перепису 1890 року знадобилося менше двох років, і він охопив більшу кількість населення (62 мільйони осіб) і більше даних, ніж перепис 1880 року.

перепис населення.

Пізніше бізнес Холлеріта об'єднався з трьома іншими, щоб створити компанію IBM!

Поточні виклики

А тепер згадаймо найгостріші виклики сучасності...



Тут ви можете побачити закон Мура:

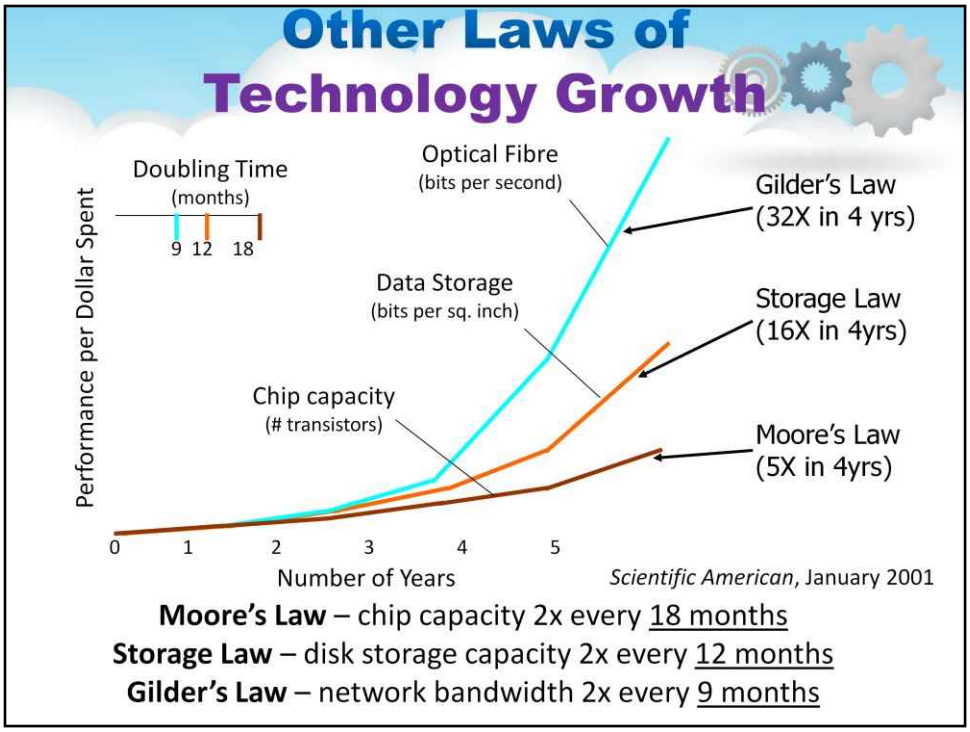
Залежність кількості транзисторів всередині процесорапроти**дати впровадження.**

Лінія відповідає експоненційному зростанню з подвоєнням кількості транзисторів кожні два роки.

Ви бачите, що ця залежність насичується протягом останніх років через фізичні обмеження на розмір елементів.

Гордон Мур сказав у 2005 році:
«Що стосується розміру *транзисторів+, ви бачите, що ми наближаємося до розміру атома, який є фундаментальним бар'єром, але мине два-три покоління, перш ніж ми зайдемо так далеко, але це настільки далеко, наскільки ми досягли колись міг побачити. У нас є ще 10-20 років, перш ніж ми досягнемо фундаментальної межі. До того часу вони зможуть виробляти більші чіпи та мати мільярдні бюджети транзисторів».

Але нинішнього розвитку єдиного центрального процесора (CPU) недостатньо для відповіді на поточні виклики.



Зараз існує кілька інших формулювань законів для характеристики технологічного зростання:

Закон Мура – обчислювальна потужність кожного комп'ютера подвоюється18 місяців Закон зберігання – ємність дискового накопичувача подвоюється12 місяців

Закон Гільдера – пропускна здатність мережі подвоюється щоразу9 місяців (але його важче встановити) Це експоненціальне зростання глибоко змінює ландшафт інформаційних технологій

Все більше і більше даних створюється (оскільки датчики менші, а процесори дешевші) і зберігається (оскільки диски дешевші й надійніші, ніж стрічки тощо), і до них здійснюється дистанційний доступ (оскільки мережа дешевша).

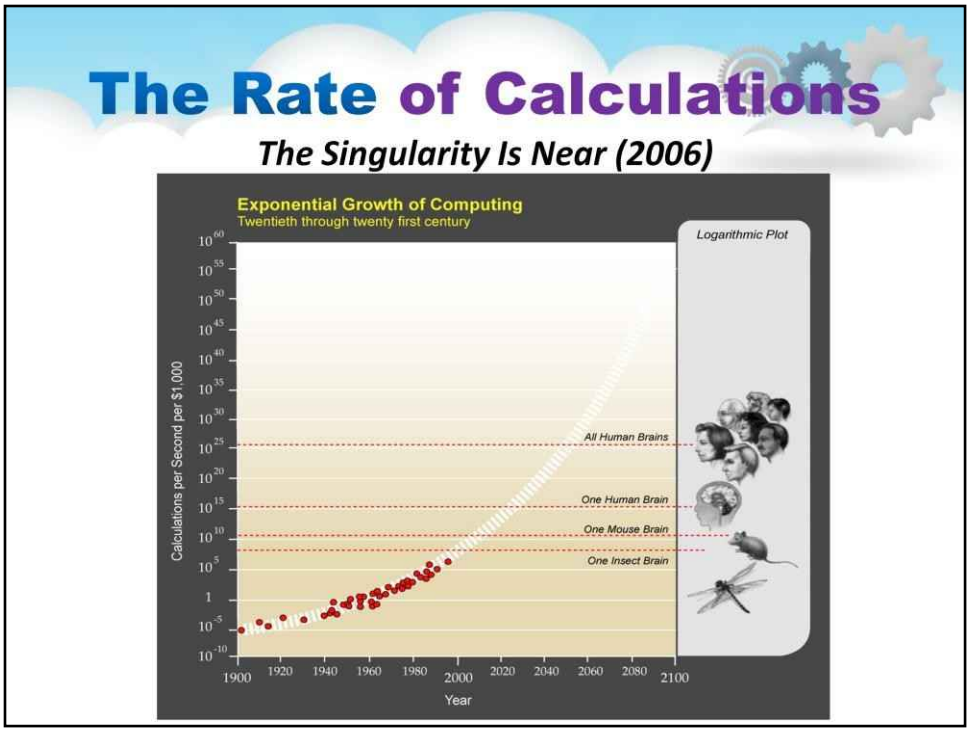
Зауважте, що час, необхідний для заповнення доступного сховища, зростає (тобто швидкість запису даних на диск зростає не так швидко, як ємність сховища)

Реймонд Курцвейл - американський письменник, комп'ютерник, винахідник і футуролог.

Окрім футуризму, він займається такими галузями, як оптичне розпізнавання символів (OCR), синтез тексту в мову, технологія розпізнавання мови та електронні клавішні інструменти.

Він написав книги про здоров’я, штучний інтелект (ШІ), сингулярність.

У своїй першій книзі «Епоха розумних машин» Курцвейл представив свої ідеї щодо майбутнього в 1990 році. Курцвейл екстраполював тенденції у покращенні продуктивності програмного забезпечення комп’ютерних шахів, щоб передбачити, що комп’ютери переможуть найкращих гравців-людей «до 2000 року». У травні 1997 року чемпіон світу з шахів Гаррі Каспаров зазнав поразки від комп’ютера IBM Deep Blue у широко розголошеному шаховому матчі.

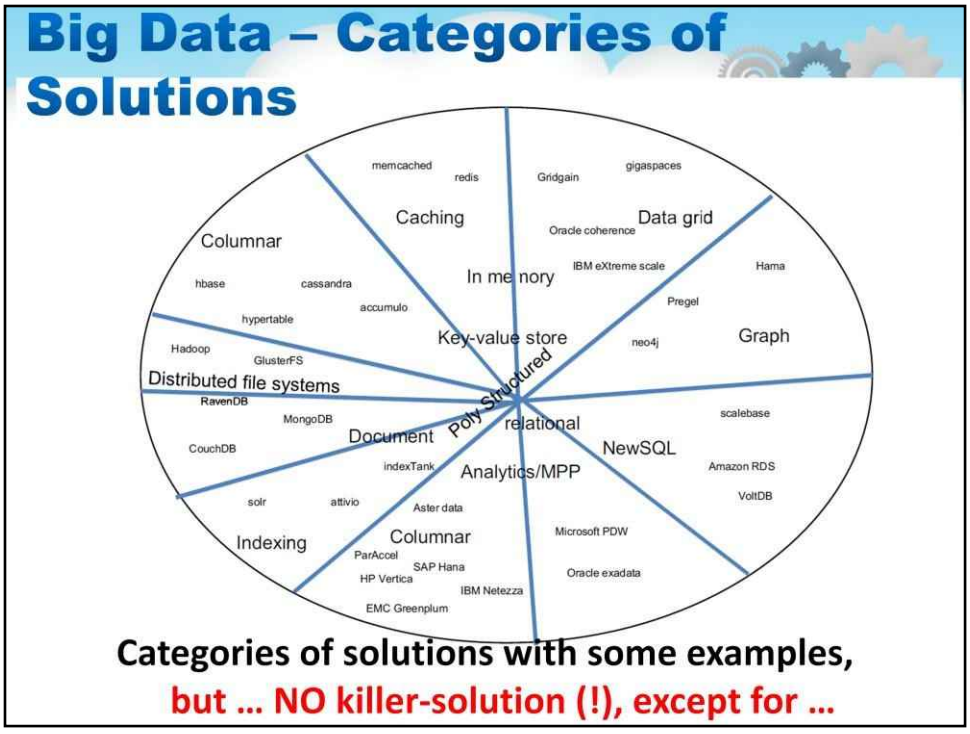


Курцвейл припускає, що це експоненційне технологічне зростання суперечить інтуїції способу, у який наш мозок сприймає світ, оскільки наш мозок був біологічно успадкований від людей, які живуть у світі, який був лінійним і локальним, і, як наслідок, він стверджує, що це сприяло великій скептицизм у своїх прогнозах на майбутнє.

У контексті згаданих вище викликів і проблем великих даних потреба в нових парадигмах для високопродуктивних обчислень є дуже високою та важливою.

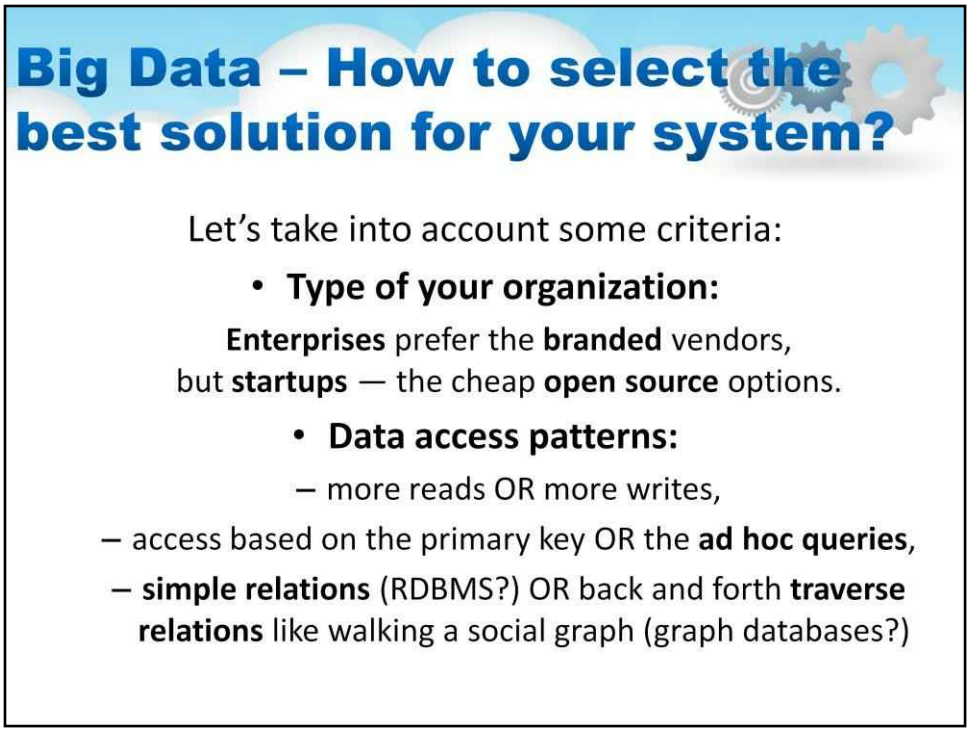
Рішення

І які рішення можна запропонувати для проблеми великих даних...



Цей слайд містить карту багатьох доступних інструментів і засобів для вирішення проблеми з даними про болото.

Швидкий висновок полягає в тому, що ми не маємо ЄДНОГО рішення для цієї проблеми.



Щоб вибрати найкраще рішення для вашої системи, ми повинні взяти до уваги наступні критерії:

- **Тип вашої організації:**

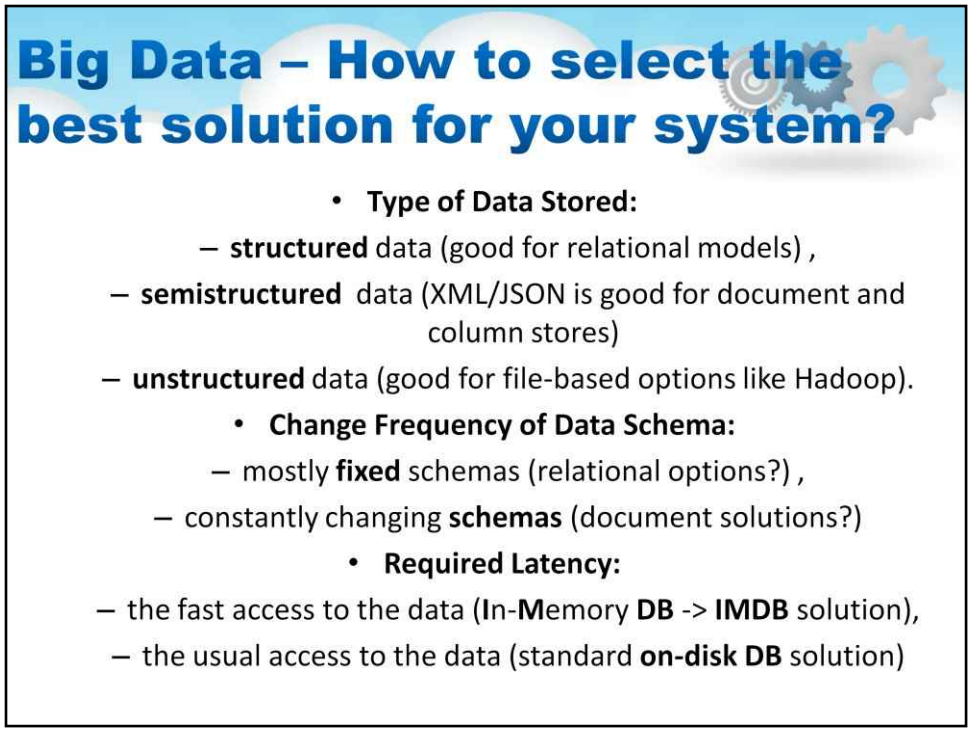
  Підприємствавіддають перевагу**фірмовий**продавці,

  але**стартапи**—дешевий**відкрите джерело**параметри.

- **Шаблони доступу до даних:**

  більше читає АБО більше пише,
  доступ на основі первинного ключа АБО**спеціальні запити**,

  прості відносини(RDBMS?) АБО вперед і назад**поперечні відносини**як прогулянка соціальним графом (базами даних графів?)



(продовження з попереднього слайду)

- **Тип збережених даних:**

  структурованийдані (добре для реляційних моделей),

  напівструктурованийдані (XML/JSON добре підходить для зберігання документів і стовпців)

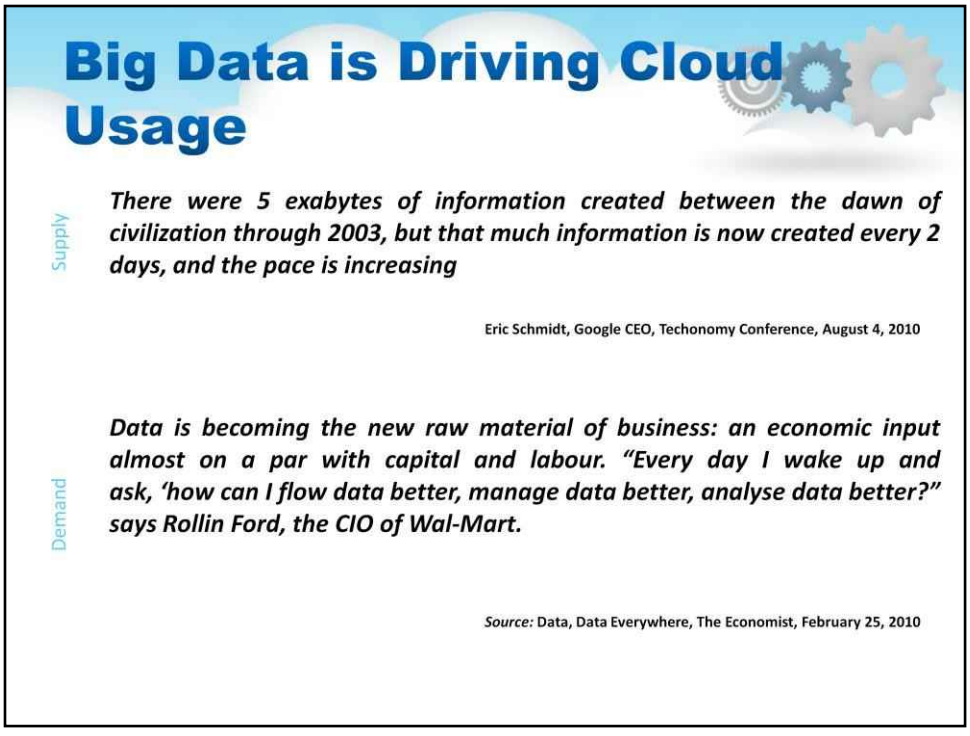
  неструктурованийданих (добре для параметрів на основі файлів, таких як Hadoop).

- **Частота зміни схеми даних:**

  в основному**фіксований**схеми (параметри відношення?), що
  постійно змінюються**схеми**(рішення документів?)

- **Необхідна затримка:**

  швидкий доступ до даних (**я**п-**М**Еморі**БД**->**IMDB**рішення),
  звичайний доступ до даних (стандарт**на диску БД**рішення)

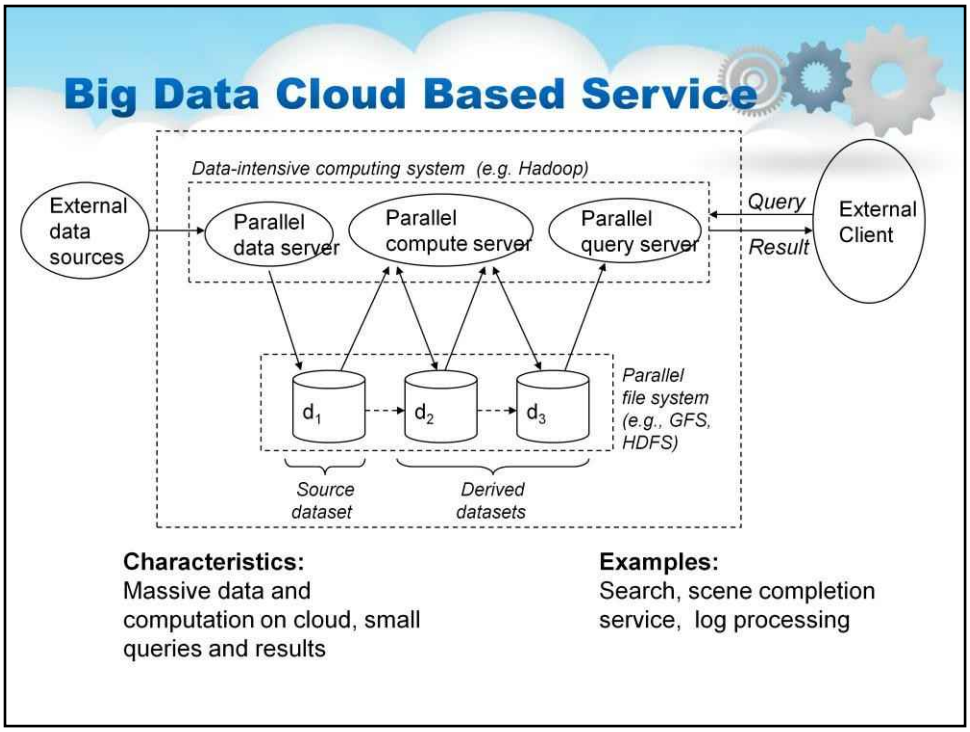


Великі дані стимулюють використання хмари

За останнє десятиліття в обчислювальній техніці виникли дві глибокі тенденції. Підприємства та телекомунікаційні компанії перетворюють свою IT-інфраструктуру з наборів слабко інтегрованих програм, кожна з яких працює на власних серверах і має власні бази даних, у стратегічні, гнучкі високопродуктивні платформи, здатні зберігати кожен біт інформації, до якої підприємство має доступ, з достатньою кількістю даних. обчислення на вимогу, щоб забезпечити можливості, які відповідають уяві найкреативнішого IT-директора.

Хмарні рішення

Візьмемо до уваги доступні хмарні рішення...

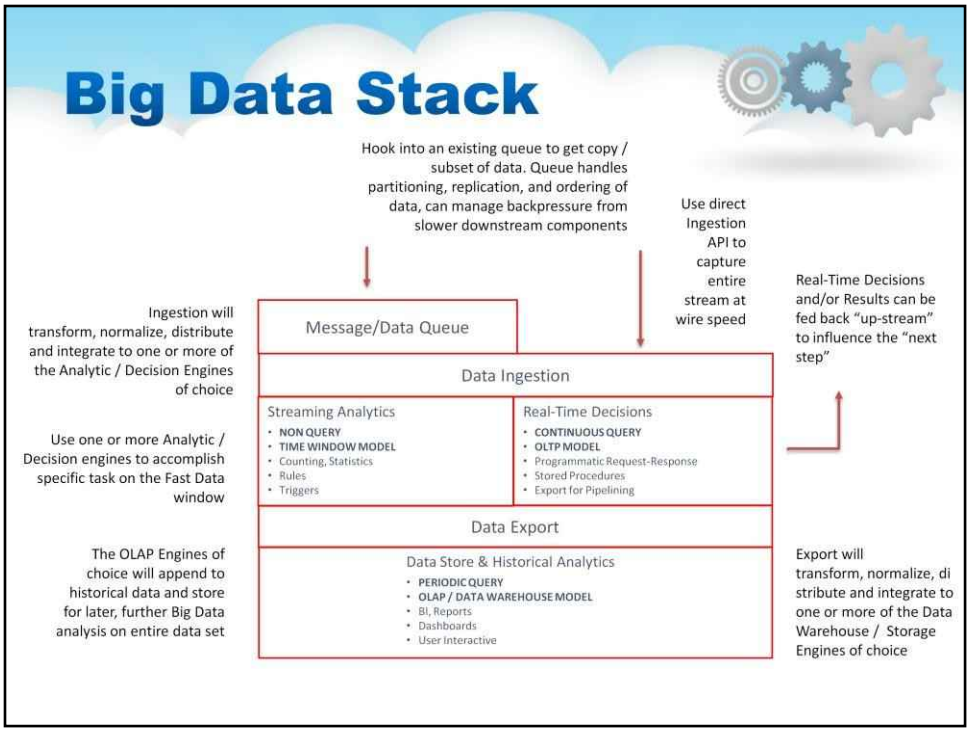


На цьому слайді показано, як працює хмарна служба великих даних

Прикладами є пошук, служба завершення сцени, обробка журналу

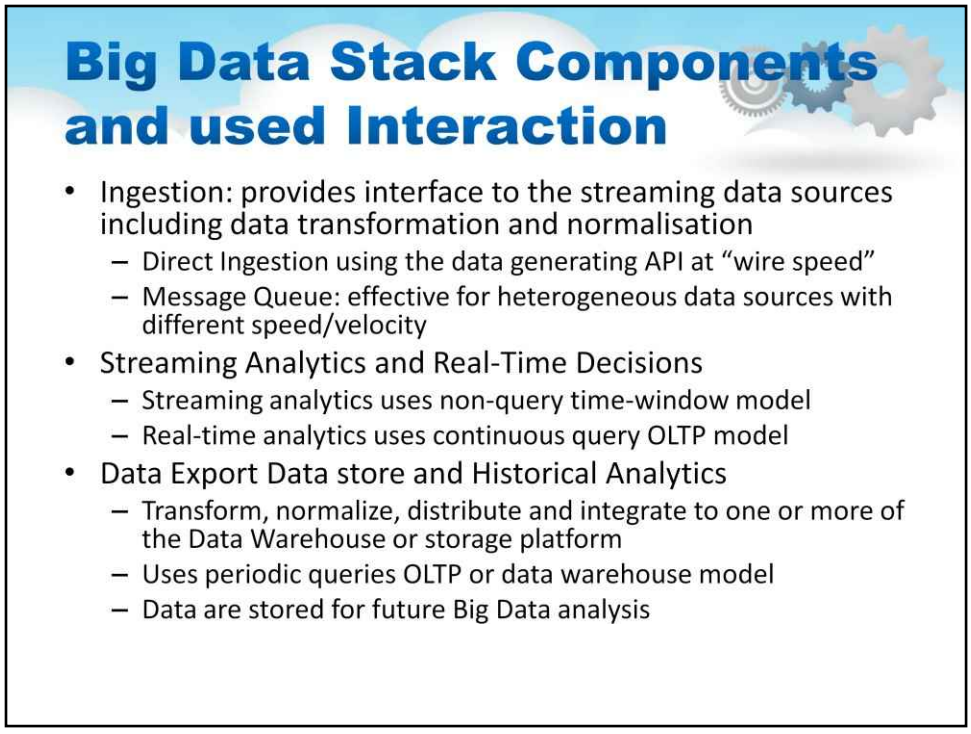
Характеристики: масив даних і обчислень у хмарі, але керуються невеликими запитами та повертають невеликі результати

Великі дані обробляються в хмарі за допомогою конвеєра даних із системою керування конвеєром, наприклад Hadoop.



Стек великих даних

Цей «стек швидких даних/великих даних» включає прийом, аналіз і зберігання. Але є кілька схем і комбінацій, які використовуються для різних випадків використання.



YD: Це комбіновані нотатки з кількох слайдів

У стеку швидких і великих даних прийом є першим етапом. Робота прийому полягає в тому, щоб підключатися до джерел потокових даних і, за потреби, трансформувати та нормалізувати дані. Тоді передавання даних потенційно розділить потік даних на один або кілька вибраних механізмів аналізу/прийняття рішень.

Є два варіанти прийому. Першим вибором є використання «Прямого прийому», якщо це можливо, де модуль прямого коду може підключитися безпосередньо до API, що генерує дані, захоплюючи безпосередньо весь потік зі швидкістю, з якою працюватиме API та мережа, наприклад, на « швидкість дроту». У цьому випадку механізми аналітики/прийняття рішень матимуть «адаптер» прямого прийому, і за допомогою певної кількості кодування механізми аналітики/прийняття рішень можуть обробляти потоки даних із конвеєра API і не потребують розміщення чи кешування будь-яких даних на диску.

Якщо доступ до API генерації даних недоступний, зазвичай альтернативою є розміщення даних у черзі повідомлень. У цьому випадку необхідна система прийому, щоб підключитися до наявної черги, щоб отримати копію/підмножину даних. Черга прийому обробляє розділення, реплікацію та впорядкування даних, може керувати зворотним тиском від повільніших компонентів нижче.

Коли дані передаються, вони в кінцевому підсумку передаються одному або кільком механізмам аналізу/прийняття рішень для виконання конкретного завдання щодо потокової передачі даних. На цьому етапі дані є швидкими даними, і завдання для механізмів аналізу/прийняття рішень полягає в тому, щоб споживати швидкість потоку даних.

Рішення в реальному часізабезпечити достатній аналіз вчасно для того, щоб рішення були передані «вгорі» та вплинули на «наступний крок» обробки. Рішення в реальному часі виконують багато роботи, оскільки вони споживають швидкість потоку даних і водночас зазвичай обробляють складну логіку, усе вчасно для завершення циклу зворотного зв’язку Рішення в реальному часі. Рішення в реальному часі мають такі характеристики:

Режим безперервного запиту
Аналітична модель OLTP

Програмні збережені
процедури запит-відповідь

Експорт для трубопроводів

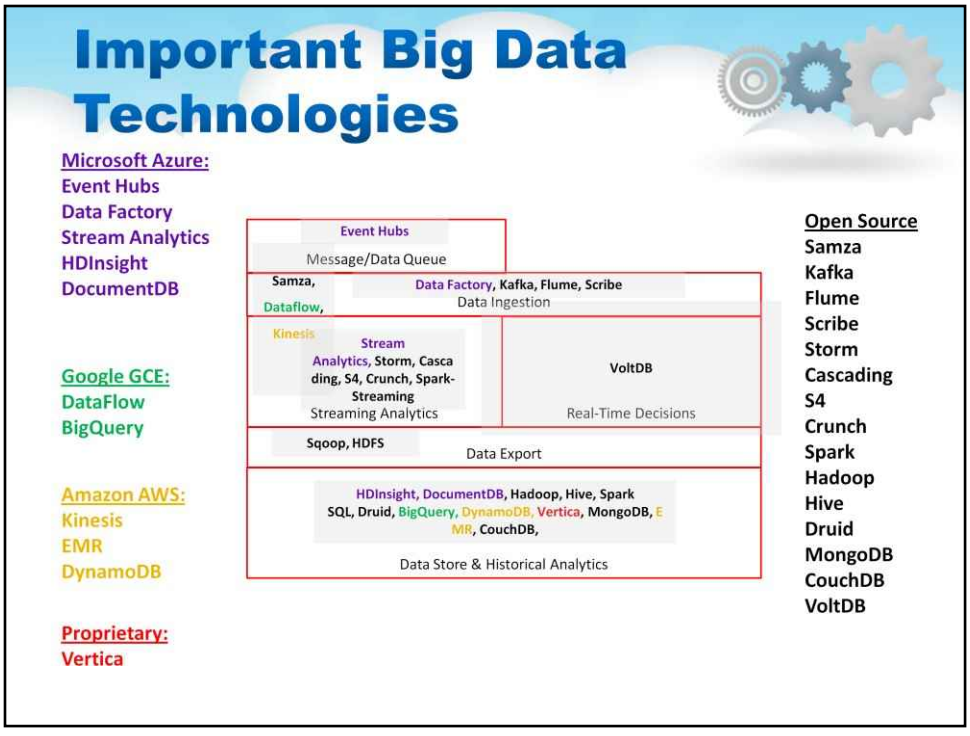
Потокова аналітикавсе ще потрібно споживати швидкість потоку даних, але обробляється менш складна логіка, де програми не вимагають зворотного зв’язку в реальному часі в програмах. Потокова аналітика має такі характеристики:

Режим без запитів

Обробка вікна часу
Підрахунок, статистика

правила
Тригери

Після завершення швидкого аналізу даних дані можуть продовжувати переміщатися в конвеєр для подальшої обробки. Зазвичай Streaming Analytics покладається на чергу прийому, яка просто продовжується на етапі експорту. Для рішень у реальному часі, які обробляють швидкі дані в режимі безперервного запиту, необхідна функція експорту для перетворення та розповсюдження даних у вибрані механізми сховища/сховища великих даних (OLAP).

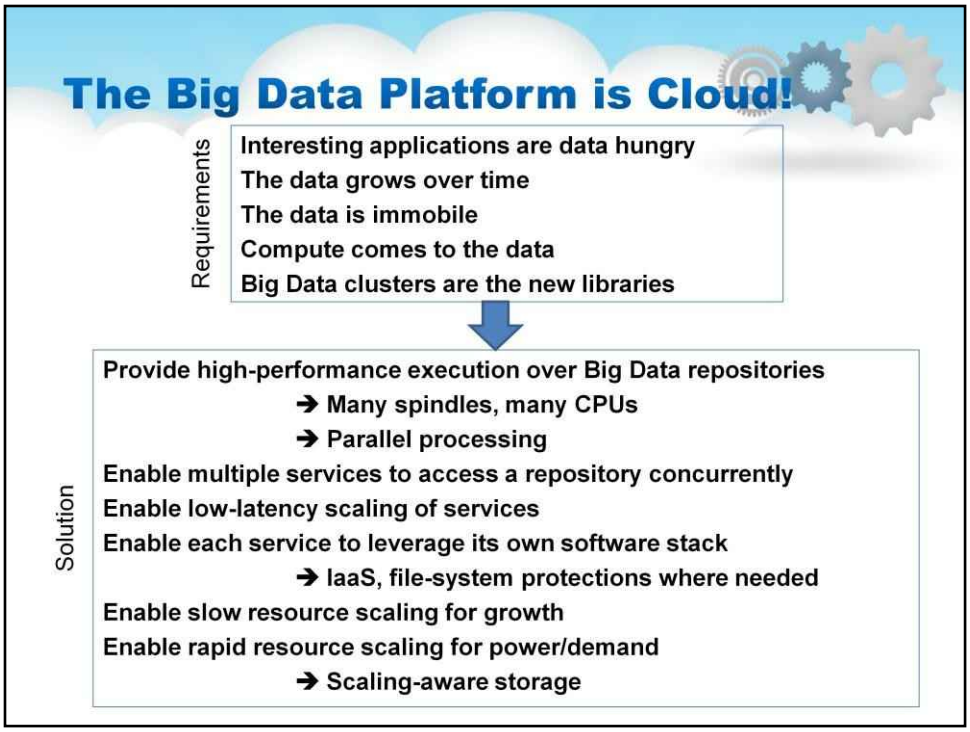


Цей слайд ілюструє важливі технології в просторі великих даних. Зауважте, що деякі з них надійшли від постачальників, а деякі – з відкритим кодом

Зверніть також увагу на деякі охоплюючі функціональні області

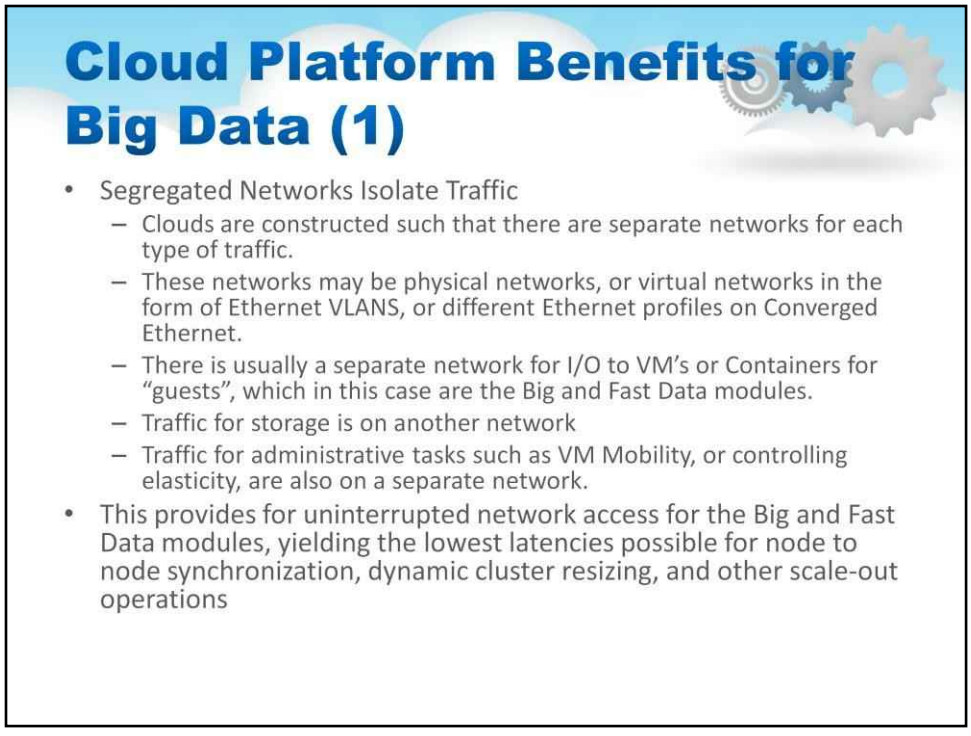
Хмарна платформа

Тепер ми переходимо до хмарної платформи для проблеми великих даних...



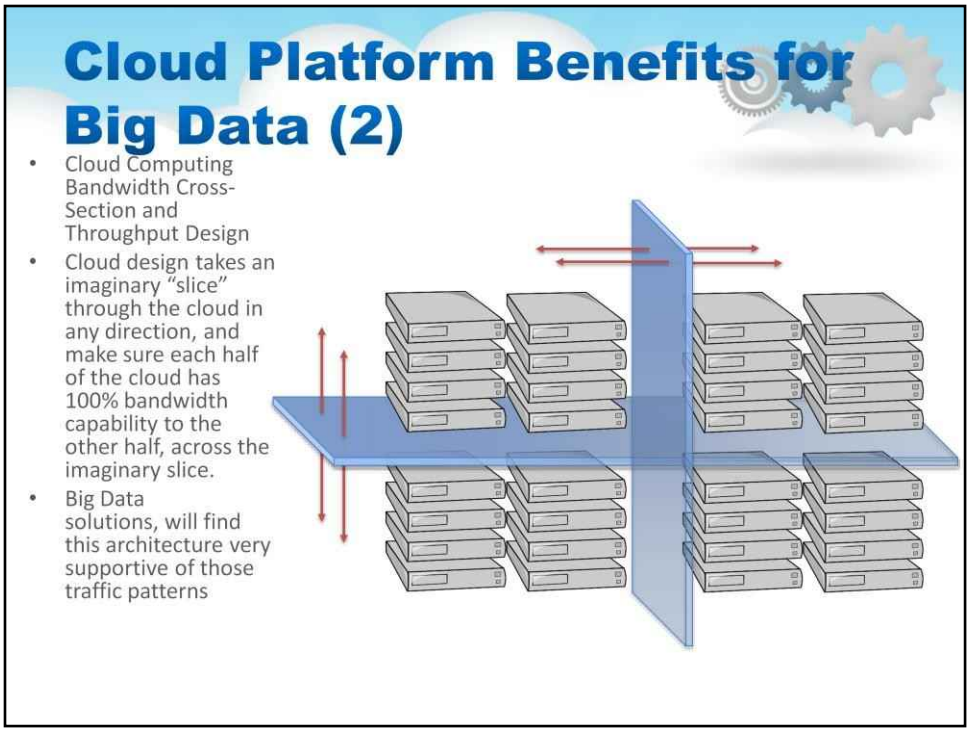
Платформа великих даних – це хмара!

Хмарні обчислення стали повсюдною, по суті нескінченною обчислювальною платформою, яка задовольняє потреби компаній і інтелектуальних програм для обробки даних – незалежно від того, наскільки швидко вони надходять і якими б розмірами не стають. Хмари розроблені таким чином, щоб мати неймовірну здатність переміщувати дані, динамічно використовувати потужність обробки даних і зберігати ці дані. Ми зрозуміли, що перегляд даних протягом усього життєвого циклу – у міру надходження та накопичення – дає потужну інформацію. Філософія масштабування, навколо якої розроблені хмари, дозволяє використовувати паралельні високомасштабовані механізми для запуску мережевих, обчислювальних елементів і елементів зберігання для обробки всіх аспектів цієї обробки. У міру того, як Інтернет перетворився на цілий світ настільних браузерів і мобільних пристроїв, для яких доступні інтелектуальні програми для будь-яких завдань, Хмара стала механізмом, який виконує всю масову обробку, що стоїть за цими програмами. Місцезнаходження людини, історія, уподобання та «що є в тренді» використовуються для забезпечення абсолютно нового онлайн-досвіду в режимі реального часу за допомогою хмари.

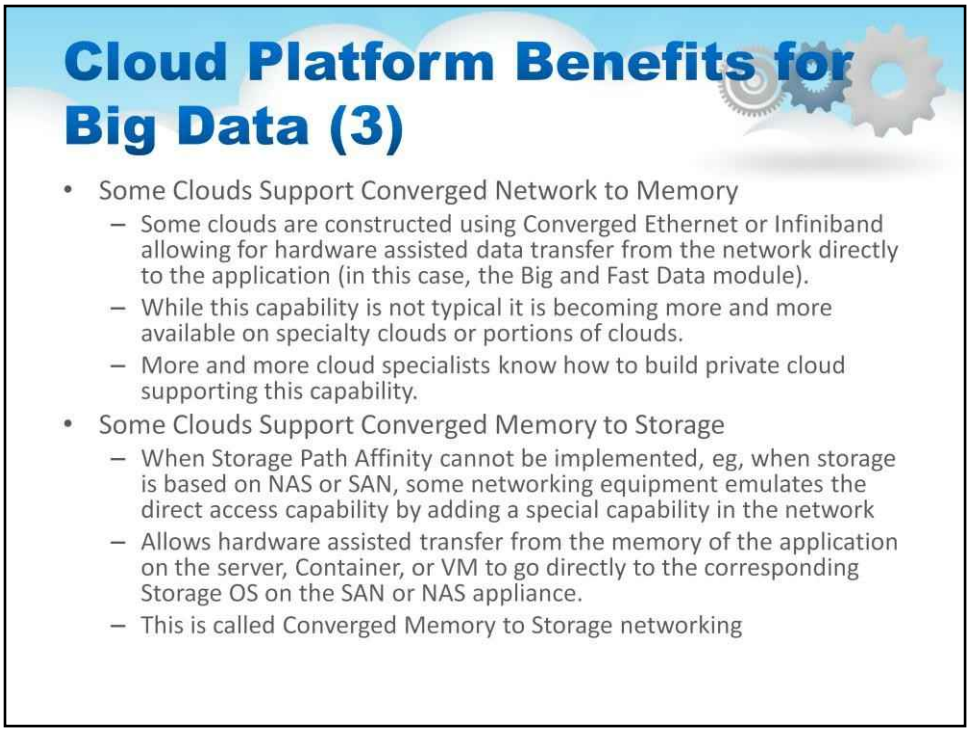


Отже, у наступних розділах буде розглянуто деякі конкретні внутрішні структури хмари та досліджено, щоб зрозуміти, наскільки синергетичними є атрибути платформи для рішень для великих і швидких даних, що працюють у хмарі.

Ці коментарі стосуються способу створення загальнодоступних або приватних хмар. Деякі з більш езотеричних функцій, згаданих тут, можна знайти лише в спеціальних «високопродуктивних обчисленнях» («HPC») або частинах хмар (як приклад загальнодоступної хмари), або можна цілеспрямовано розробити як таку, придбавши конкретне програмне та апаратне забезпечення який надає ці можливості (як приклад приватної хмари).



Хмарні екземпляри — це в основному стійки серверів, з'єднаних між собою ефективним чином. На малюнку нижче показано примірник Cloud із 4 стійками серверів, по 8 серверів у кожній стійці. Кожен сервер міститиме кілька чіпів ЦП, і кожен чіп ЦП міститиме кілька ядер, кожне ядро здатне виконувати кілька потоків. Під час створення хмарного екземпляра потрібно бути обережним, щоб не перепідписати мережу в «вищих» точках агрегації. Один із методів, який використовується, полягає в тому, щоб зробити уявний «зріз» хмари в будь-якому напрямку та переконатися, що кожна половина хмари має 100% пропускну здатність до іншої половини через уявний зріз. У результаті цього аналізу типові корпоративні типи архітектур Top of Rack/Aggregation були замінені архітектурами «Leaf and Spine».



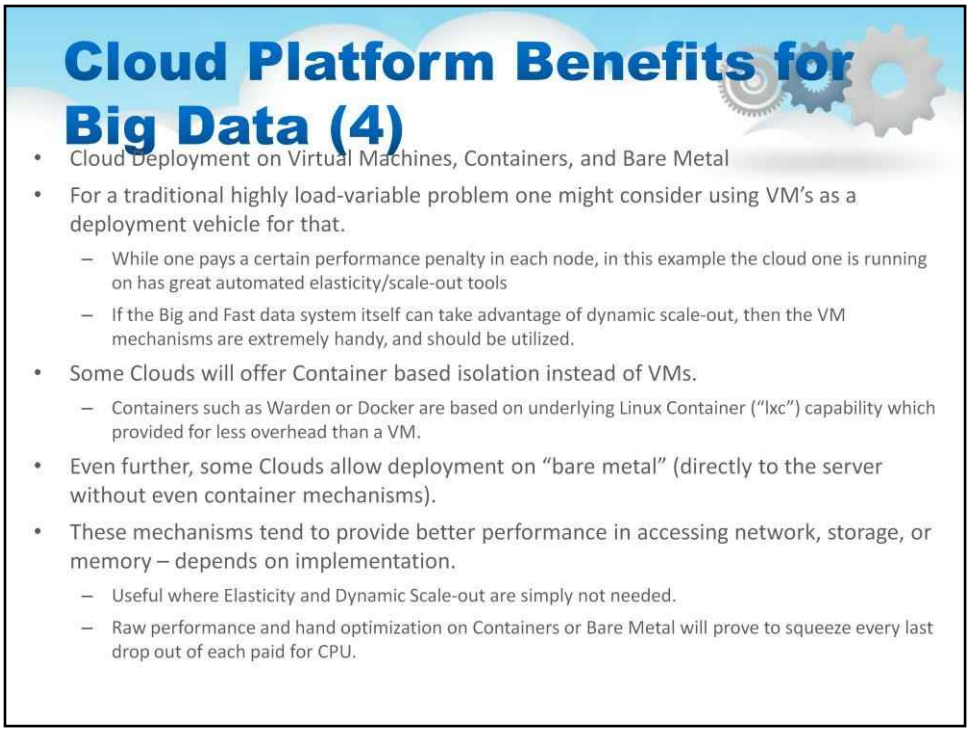
Деякі хмари підтримують конвергентну мережу в пам’ять

Дивлячись на моделі хмарних мереж, можна знайти певні хмари, які використовують спеціальні мережеві можливості, призначені для значного збільшення межвузлової мережі. Як і більшість драйверів операційної системи, стандартні мережеві драйвери є функціями ядра в операційних системах хмарних серверів, які зазвичай копіюють дані з мережі у віртуальні машини або контейнери, на яких запущені модулі програми. Деякі хмари побудовані за допомогою конвергентного Ethernet або Infiniband, що дозволяє апаратно передавати дані з мережі безпосередньо до програми (у цьому випадку модуль Big and Fast Data). Хоча ця можливість не є типовою, вона стає все більш доступною для спеціальних хмар або частин хмар.

Все більше фахівців з хмарних технологій знають, як створити приватну хмару, що підтримує цю можливість. Це справжній «турбонаддув» для певних класів розподілених програм. Big and Fast Data буде одним з перших, хто скористається цією можливістю, оскільки вона стає все більш поширеною.

Деякі хмари підтримують конвергентну пам’ять для зберігання

Якщо Storage Path Affinity не може бути реалізовано, наприклад, коли сховище базується на NAS або SAN, деяке мережеве обладнання емулює можливість прямого доступу, додаючи спеціальну можливість у мережі, що дозволяє апаратно передавати з пам’яті програми на сервері, Контейнер або віртуальна машина, щоб перейти безпосередньо до відповідної ОС зберігання на пристрої SAN або NAS. Це називається мережею Converged Memory to Storage і є ще однією з езотеричних оптимізацій, які можна знайти в хмарі, які ідеально підходять для проблем великих і швидких даних.



Моделі хмарного розгортання – віртуальні машини, контейнери та Bare Metal

Щоразу, коли продуктивність програми є проблемою, потрібно враховувати компроміси у вартості, керованості та специфічній продуктивності (наприклад, затримки) для різних доступних варіантів розгортання. Наприклад, розглянемо задачу з великим навантаженням. Для цього можна розглянути можливість використання віртуальних машин як засобу розгортання. Хоча кожен вузол сплачує певний штраф за продуктивність, у цьому прикладі хмара, на якій він працює, має чудові автоматизовані інструменти еластичності/масштабування, і якщо сама система великих і швидких даних може скористатися перевагами динамічного масштабування, то Механізми VM надзвичайно зручні, і їх слід використовувати.

Деякі хмари пропонують ізоляцію на основі контейнерів замість віртуальних машин. Контейнери, такі як Warden або Docker, базуються на можливостях основного контейнера Linux («lxc»), які забезпечують менше витрат, ніж віртуальна машина. Крім того, деякі хмари дозволяють розгортати на «голому металі» (безпосередньо на сервері навіть без контейнерних механізмів). Ці механізми, як правило, забезпечують кращу продуктивність під час доступу до мережі, сховища чи пам'яті – залежить від реалізації. Розглянемо досить статичну пропускну здатність і швидкість набору проблем великих і швидких даних, які можуть не потребувати будь-якої змінності інфраструктури
– Еластичність і динамічне масштабування просто не потрібні. Тому масштабна автоматизація на основі віртуальних машин також може не знадобитися. Висока продуктивність і ручна оптимізація на Containers або Bare Metal дозволять вичавити до останньої краплі з кожного платного ЦП.

Інтернет речей

Наступним ДІЙСНО великим аспектом є Інтернет речей як джерело набагато більших даних, ніж будь-коли...

IoT (Інтернет речей) — це набір взаємопов’язаних пристроїв, які генерують та обмінюються даними на основі спостережень, фактів та іншими даними, роблячи їх доступними для будь-кого. Рішення Інтернету речей розроблені, щоб зробити наші знання про навколишній світ більш своєчасними та актуальними, дозволяючи отримувати дані про що завгодно з будь-якого місця та в будь-який час. Очевидно, що кількість пристроїв IOT може перевищити кількість людей на планеті.

Рішення IOT – це просто набір пристроїв, призначених для виробництва, споживання або представлення даних про певну подію чи серію подій чи спостережень. Це може включати пристрої, які генерують дані, такі як датчики, пристрої, які об’єднують дані, щоб зробити висновок, пристрої або служби, призначені для таблиць і зберігання даних, а також пристрої або системи, призначені для представлення даних. Будь-який або всі з них можуть бути підключені до Інтернету.

Рішення IOT можуть включати одну або всі ці якості, незалежно від того, об’єднані вони в одному пристрої, такому як веб-камера, використовують пакет датчиків і блок моніторингу, такий як метеостанція, або використовують складну систему спеціальних датчиків, агрегаторів, зберігання даних , а також презентація, наприклад повна система домашньої автоматизації.

Це популярне зображення показує футуристичне зображення всіх пристроїв, повсюдно підключених до Інтернету через бази даних, збирачі даних або інтегратори, служби відображення та навіть інші пристрої.

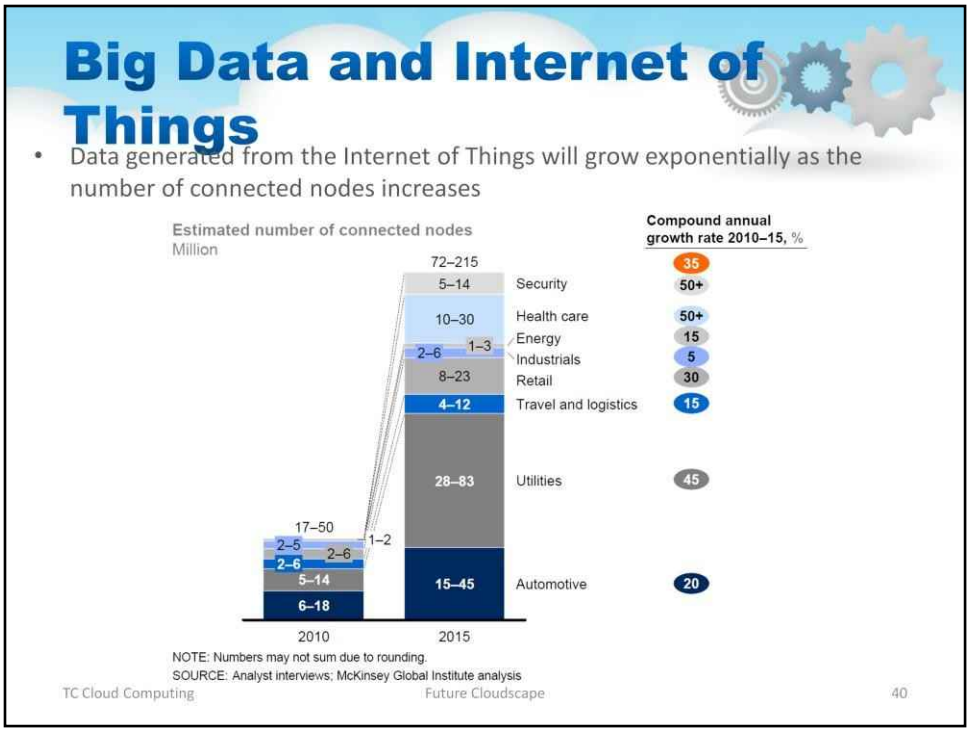
Збір і розуміння всіх цих даних ускладнюється тим, як дані зберігаються або більш відповідним чином вилучаються.

Наприклад, що сталося б, якби кожен пристрій у вашому домі, машині, офісі тощо створював дані?

Додайте до цього погляд на зростаючий інтерес до переносних датчиків і подібних пристроїв, і ви отримаєте потенціал генерувати більше даних, ніж може впоратися чи навіть розшифрувати будь-яка людина.

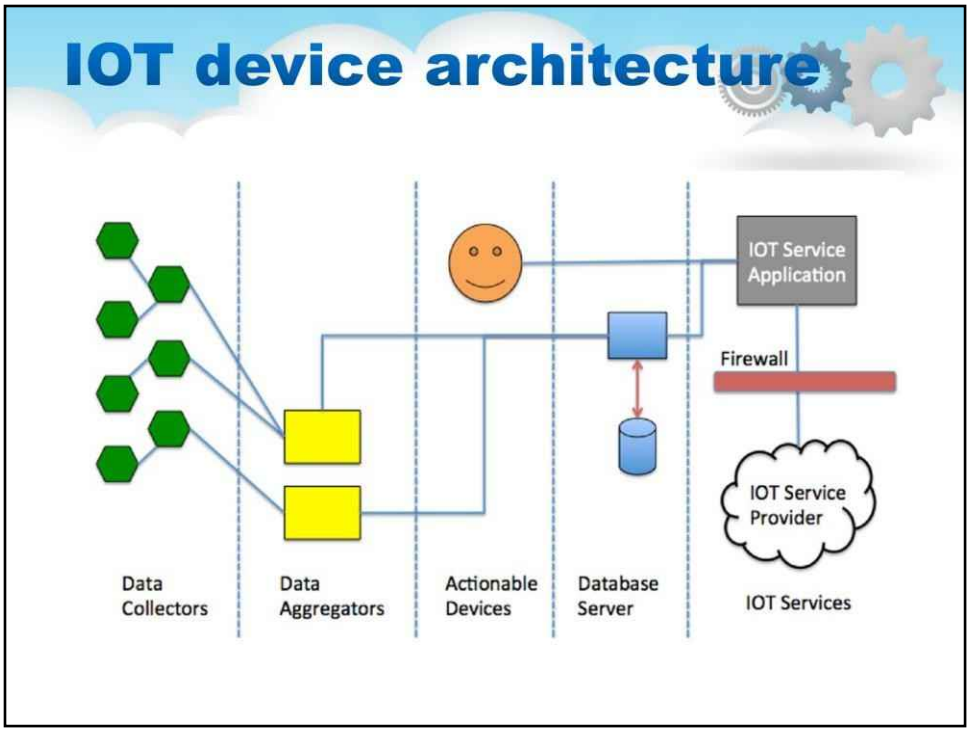
Давайте розглянемо коротку історію носимих комп’ютерів на цьому слайді.

Це Стів Манн і короткий опис його діяльності як піонера в області носимих комп'ютерів і мультимедійних даних, створених носимими гаджетами.



Великі дані ще більше прискорюються завдяки Інтернету речей

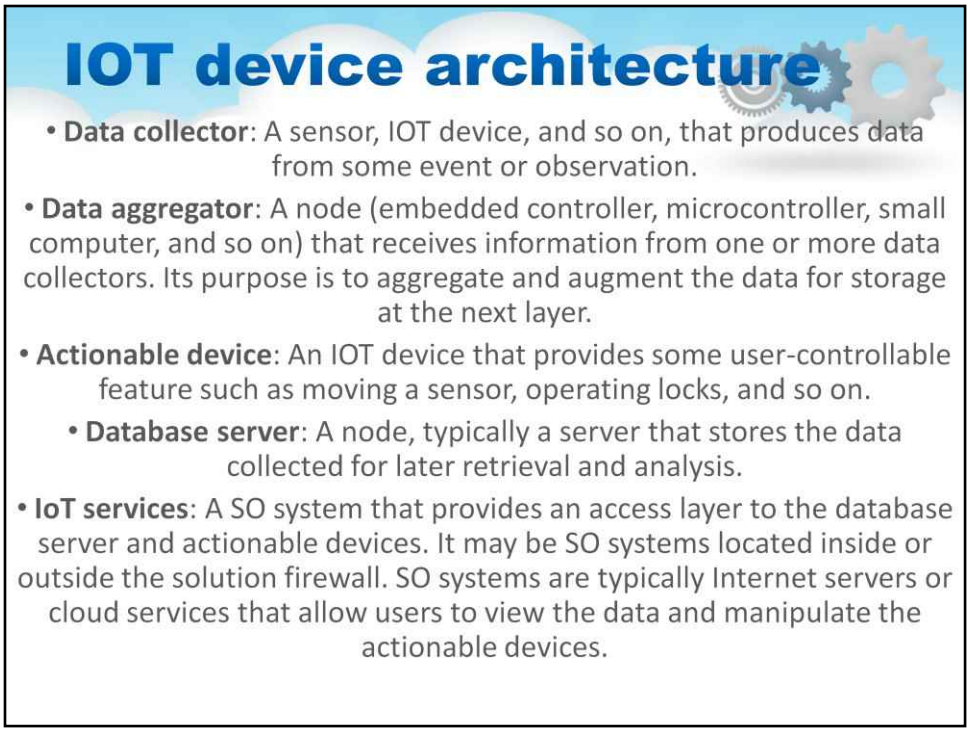
Слайд ілюструє, як дані, згенеровані з Інтернету речей, зростатимуть експоненціально зі збільшенням кількості підключених вузлів



Ліворуч від цього слайда розташовані пристрої IOT. Це може бути простий датчик, ціла сенсорна мережа, пристрій з одним або декількома датчиками, вбудований мікроконтролер із датчиками, більш складне рішення на основі мікропроцесора або навіть такий пристрій, як мікрохвильова піч, будильник або телевізор. . Ці пристрої є більш складними пристроями з вбудованими мережевими можливостями.

У центрі знаходиться користувач, який отримує доступ до даних через Інтернет або хмарний сервіс. Звичайно, це може бути будь-який тип пристрою, наприклад ноутбук, настільний комп'ютер, планшет, телефон, годинник або інший смарт-пристрій або пристрій (включаючи інший пристрій IOT).

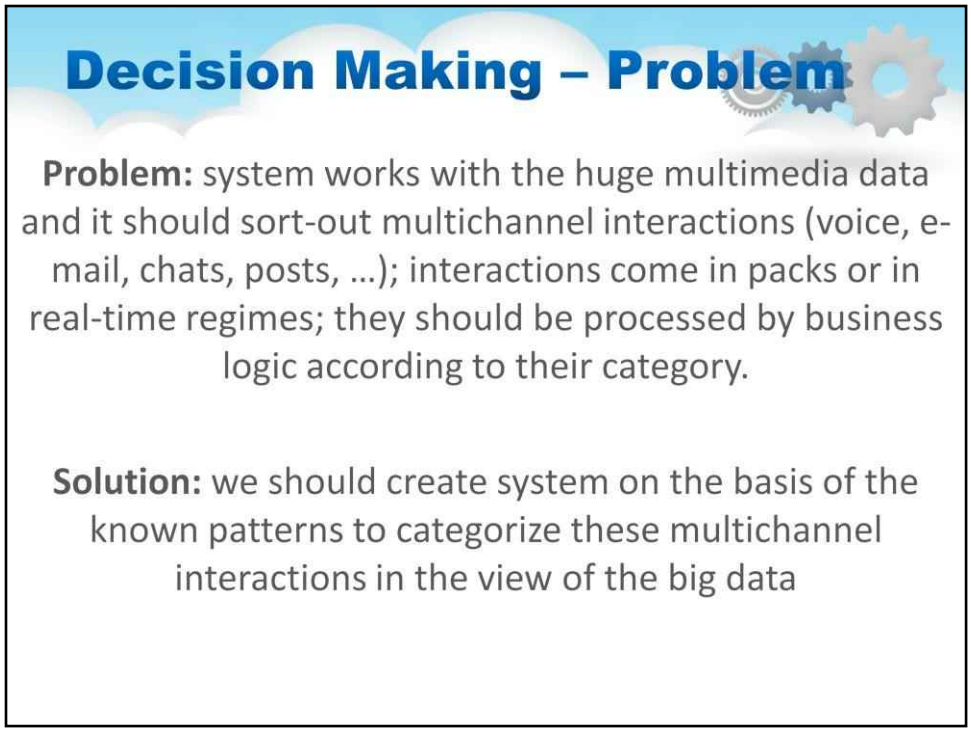
На наступному слайді давайте розглянемо ролі в архітектурі пристроїв IoT, представлені внизу цього слайда.



Давайте розглянемо ролі в архітектурі пристроїв IoT, представлені на попередньому слайді...

Прийняття рішень

Тепер давайте розглянемо, як обробка великих даних може допомогти нам у реальному житті...



З цієї формулювання проблеми:

> **проблема:**система працює з величезною кількістю мультимедійних даних і повинна сортувати багатоканальні взаємодії (голос, електронна пошта, чати, публікації, ...); взаємодії відбуваються пакетами або в режимах реального часу; вони повинні оброблятися бізнес-логікою відповідно до їх категорії.

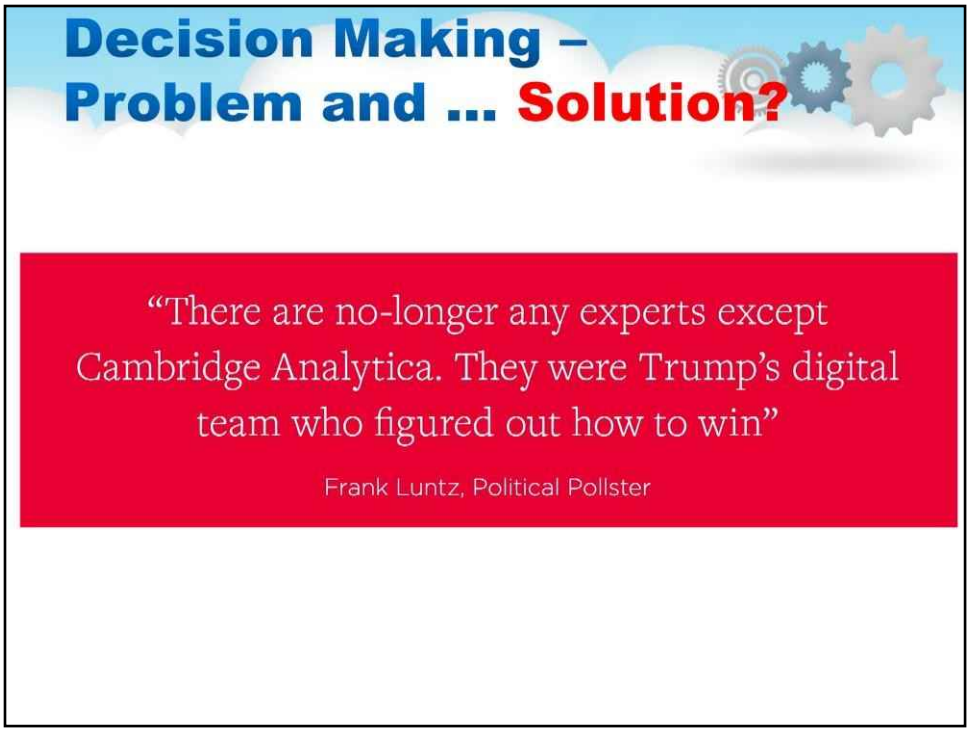
ми повинні знайти деяке рішення, яке можна формалізувати наступним чином:

> **рішення:**ми повинні створити систему на основі відомих шаблонів, щоб класифікувати ці багатоканальні взаємодії з точки зору великих даних

У зв'язку з цим,**машинне навчання**можна використовувати, яка є галуззю інформатики, яка використовує статистичні методи, щоб надати комп’ютерним системам здатність «навчатися» з даними без явного програмування.

Термін «машинне навчання» був запропонований Артуром Самуелем у 1959 році.
Він почався з вивчення теорії розпізнавання образів і обчислювального навчання в штучному інтелекті.

Тепер це стосується дослідження алгоритмів, які можуть навчатися з даних і робити прогнози на основі даних – такі алгоритми долають дотримання строго статичних програмних інструкцій, роблячи керовані даними передбачення або рішення, створюючи модель із зразків вхідних даних.



Розглянемо останній приклад участі Cambridge Analytica в обробці великих даних під час виборів у США.

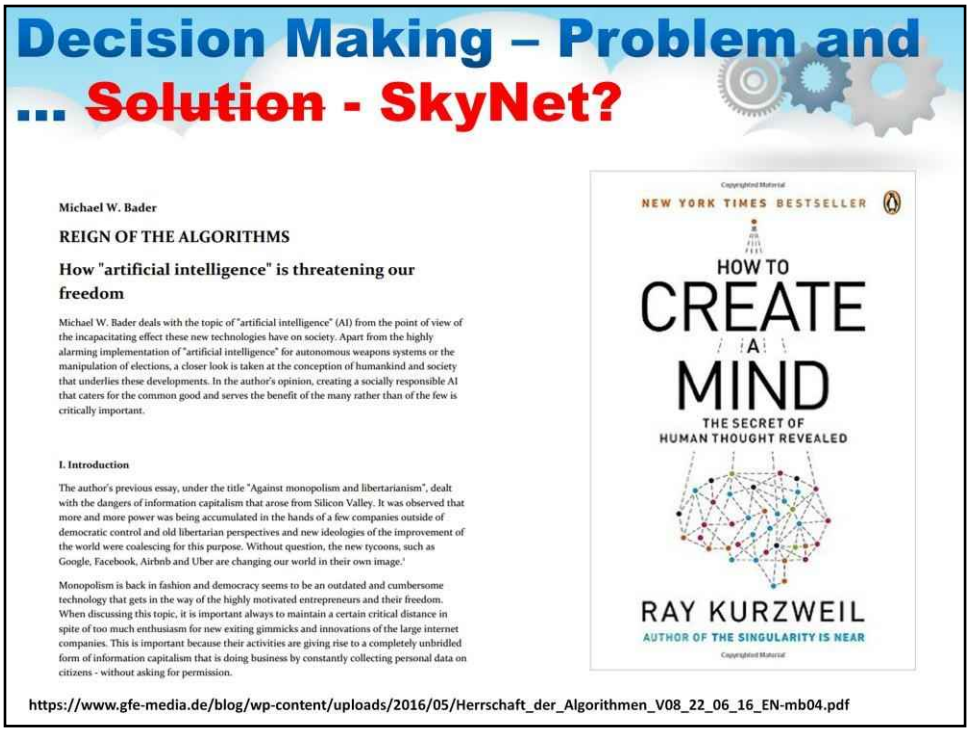
Cambridge Analytica Ltd (Каліфорнія) була британською політичною консалтинговою компанією, яка поєднувала аналіз даних, брокерство та аналіз даних (машинне навчання) зі стратегічною комунікацією під час виборчих процесів.

Особисті дані приблизно 87 мільйонів користувачів Facebook були отримані через 270 000 користувачів Facebook, які використовували додаток Facebook під назвою «This Is Your Digital Life».

Вони розробили систему профілювання, використовуючи загальні онлайн-дані, лайки у Facebook і дані смартфонів. Вони показали, що за допомогою обмеженої кількості «лайків» людей можна аналізувати краще, ніж це можуть робити друзі чи родичі, і що індивідуальне психологічне таргетування є потужним інструментом впливу на людей.

Для кожного політичного клієнта CA може звузити сегменти виборців із 32 різних стилів особистості, які вона приписує кожному дорослому жителю Сполучених Штатів. Персональні дані вказуватимуть на тон мови, що використовується в рекламних повідомленнях або сценаріях контактів з виборцями, тоді як додаткові дані використовуються для визначення позиції виборців щодо конкретних питань.

CA працювала на президентську кампанію Дональда Трампа, і CA вплинула на президентську кампанію Трампа в США в 2016 році - фактично допомагає Трампу перемогти на подив багатьох політологів.



У 2012 році винахідник і футуролог Рей Курцвейл (його згадували на початку цього

лекція) опублікував свою книгу «Як створити розум: розкрита таємниця людської думки» про мозок, як людський, так і штучний. Вона стала бестселером за версією New York Times.[

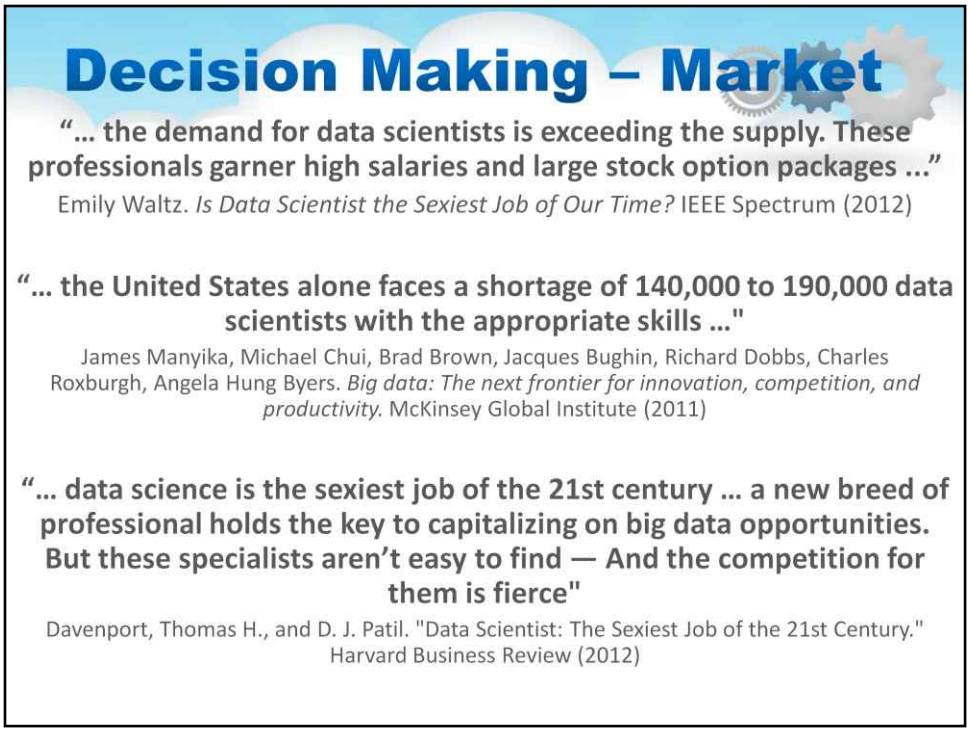
Курцвейл припускає, що мозок містить ієрархію розпізнавачів шаблонів, і мозок є «рекурсивним імовірнісним фракталом», рядок коду якого представлено в межах 30-100 мільйонів байт стисненого коду в геномі.

Потім Курцвейл пояснює, що комп’ютерна версія цього дизайну може бути використана для створення штучного інтелекту, здатнішого за людський мозок. Він використовуватиме такі методи, як приховані моделі Маркова та генетичні алгоритми, стратегії, які Курцвейл успішно використовував у роки свого комерційного розробника програмного забезпечення для розпізнавання мови.

Штучний мозок потребуватиме величезної обчислювальної потужності (яку на даний момент можуть забезпечити хмарні обчислення), тому Курцвейл переглядає свій закон прискорення віддачі, який пояснює, як комплексні ефекти експоненціального зростання забезпечать необхідне обладнання лише за кілька десятиліть.

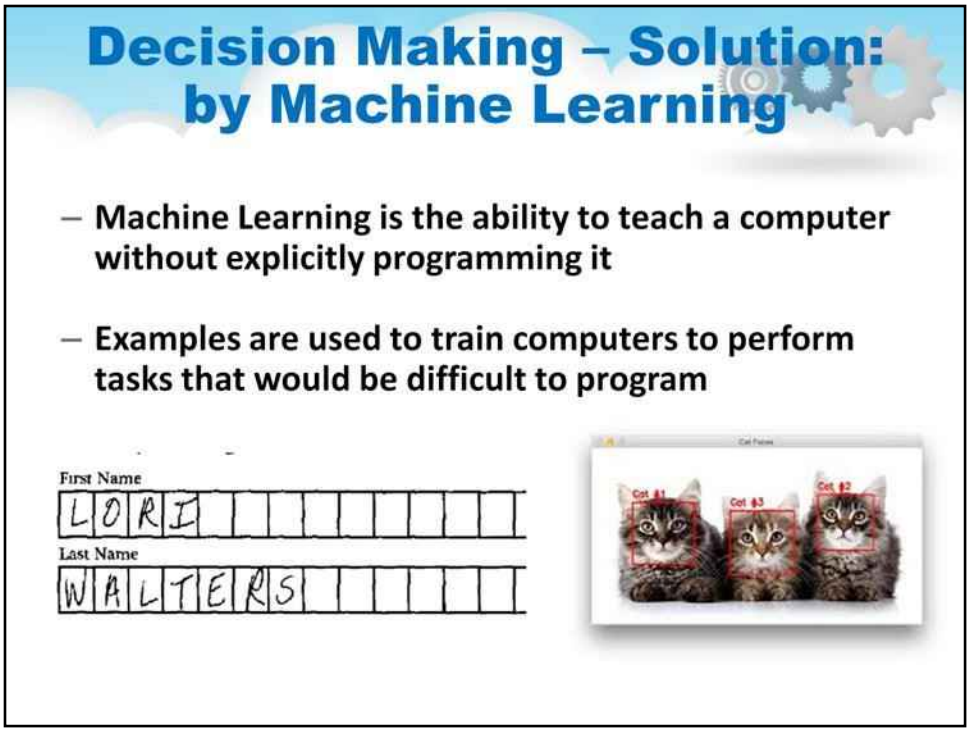
Пам'ятаєте SkyNet з фільму "Термінатор"? Ця вигадана система штучного інтелекту (AI) займає центральне місце у фільмі та є справжнім головним антагоністом.

Можливо, ми стикаємося з передумовами справжнього Skynet як можливої загрози, яку достатньо розвинений ШІ може становити для людства. Ілон Маск раніше згадував Skynet, коли говорив про таку загрозу.

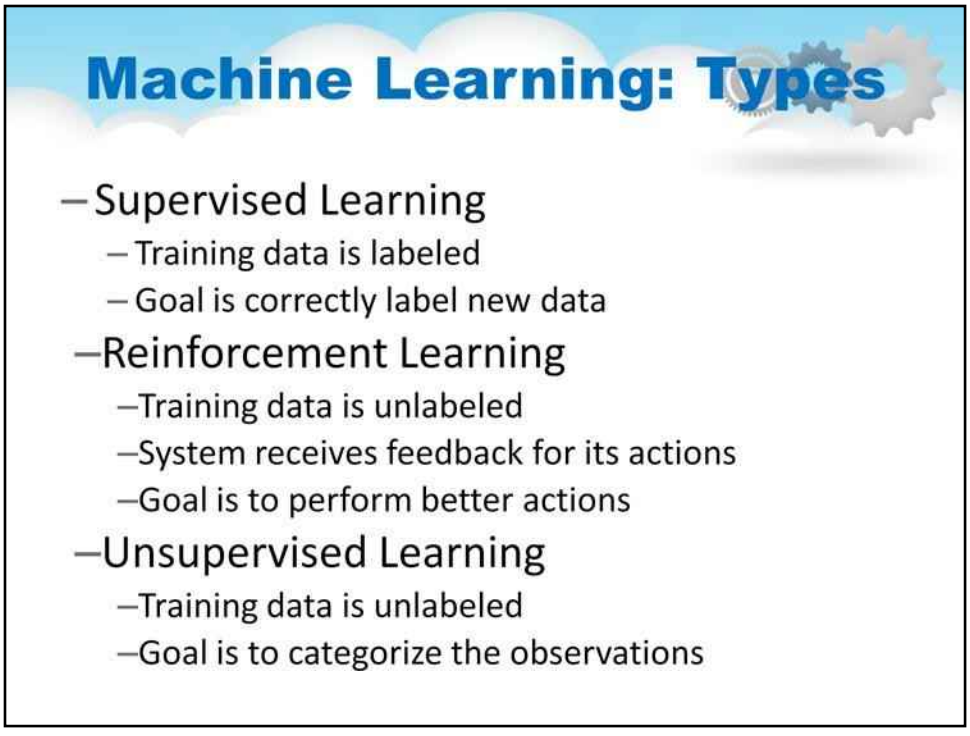


Потенційний ринок для таких систем прийняття рішень ВЕЛИЧЕЗНИЙ!

Давайте коротко розглянемо їхні основи машинного навчання на наступних слайдах...

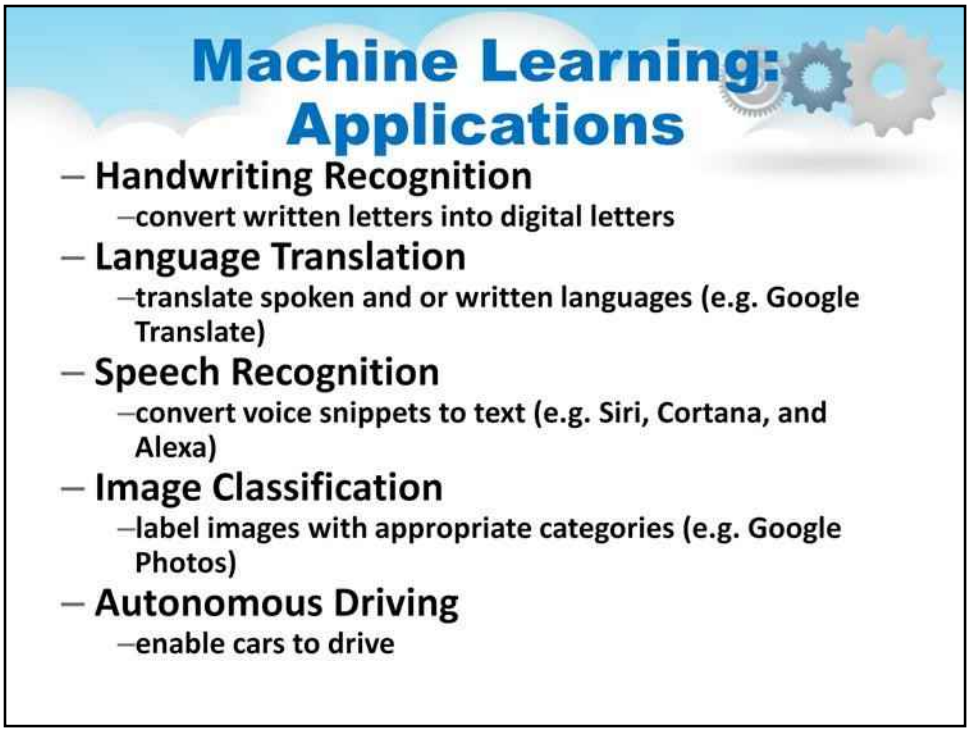


- Машинне навчання покладається на велику кількість даних для навчання комп’ютерів (OCR та виявлення об’єктів на зображеннях)
- Комп’ютери можуть вивчати складні завдання, програмування яких потребуватиме величезних зусиль
- Машинне навчання обмежене доступністю прикладів і обчислювальних ресурсів
- Маючи достатню кількість прикладів і обчислювальну потужність, комп’ютери можуть вивчати майже будь-які завдання
- Моделі є основним компонентом машинного навчання
- Модель — це метод використання відомих прикладів для прогнозування або створення висновків на основі нових даних.
- Роботи піддаються впливу шумного та динамічного середовища, тому запрограмувати робота на будь-які ситуації неможливо
- Тому інженери часто використовують машинне навчання, щоб навчити роботів виконувати завдання, подібно до того, як навчаються діти.



- Контрольоване навчання є найпоширенішим типом машинного навчання.
- Прикладом навчання під наглядом є класифікація електронних листів як СПАМ.
- Навчальні дані – це електронні листи, які позначені як СПАМ або НАМ.
- Потім створюється модель, яка фіксує зв’язок між вмістом електронної пошти та міткою електронної пошти.
- Потім модель може передбачити категорію нових електронних листів.
- Навчання з підкріпленням зазвичай використовується в робототехніці, оскільки там зазвичай немає позначених даних
- Прикладом навчання з підкріпленням є навчання робота підніматися сходами.
- Робот отримує «винагороду» за кожну сходинку, на яку він піднімається, тому він дізнається, які дії є корисними
- Неконтрольоване навчання використовується в інтелектуальному аналізі даних, щоб виявити інформацію про немарковані дані

- Прикладом неконтрольованого навчання є групування квітів на основі їхніх характеристик без знання виду квітки

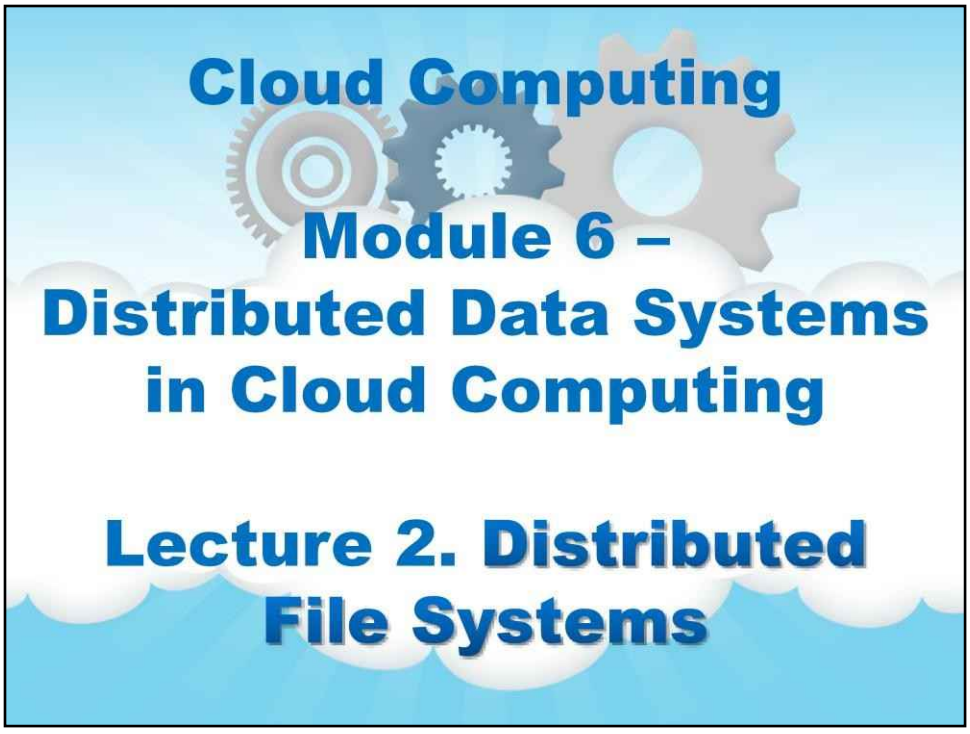


- За останні роки машинне навчання дало вражаючі результати в різних дисциплінах.
- Існує багато доступних мобільних додатків, які дозволяють користувачам фотографувати рукописні символи та перетворювати їх на цифровий текст.
- Більшість програм для перекладу зараз використовують машинне навчання для розуміння мовного перекладу, оскільки часто немає однозначної відповідності між словами в різних мовах
- До машинного навчання розпізнавання мовлення було неприємним і неточним, тепер машинне навчання забезпечує надійне розпізнавання мовлення на різноманітних пристроях
- Споживчі програми, такі як Google Photos і Apple Photos, автоматично групують фотографії за людьми або місцями, щоб зробити пошук простим та інтуїтивно зрозумілим
- Було б непрактично запрограмувати автомобіль на будь-яку ситуацію, яка може виникнути, але машинне навчання дозволило автомобілям і безпілотним автомобілям вчитися на їхньому досвіді.

У цій лекції ми розглянули:

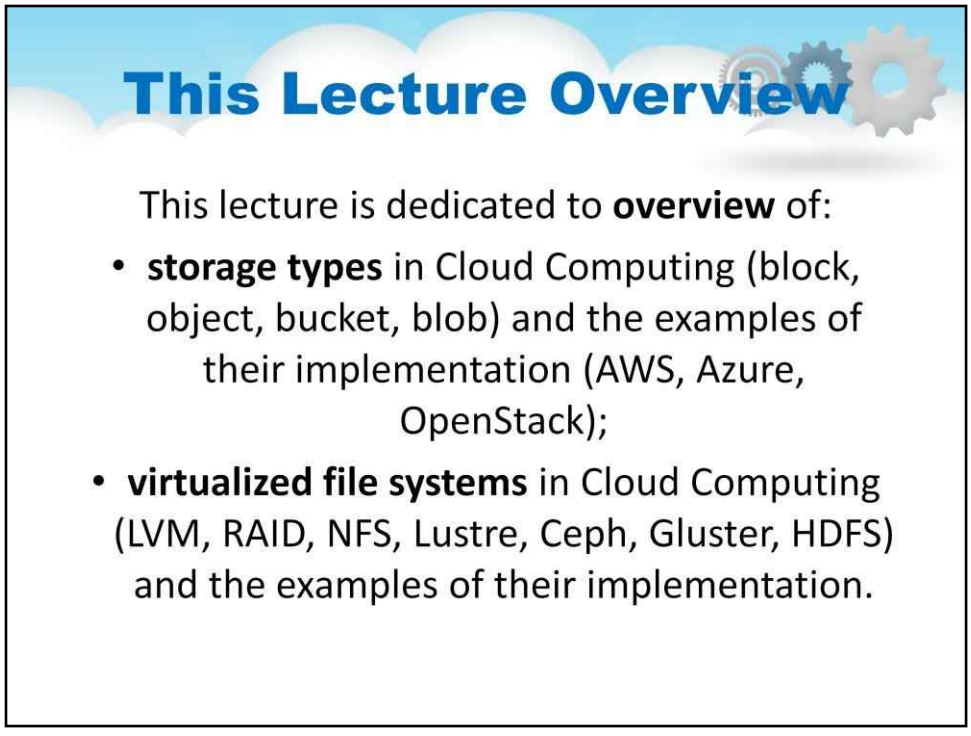
Хмарні обчислення стимулюють трансформацію багатьох технологій і забезпечують основу для нових технологій, таких як Big Data

І самі великі дані стимулюють розробку хмарних технологій відповідно до обсягу та швидкості обробки даних



Назва цього модуля: «Системи розподілених даних у хмарних обчисленнях».

Це лекція 2 про розподілені файлові системи.



Лекція 2. Розподілені файлові системи

Ця лекція описує наступні основні компоненти розподілених файлових систем:

- види зберігання в хмарних обчисленнях

- блок,
- об'єкт,
- відро,
- крапка

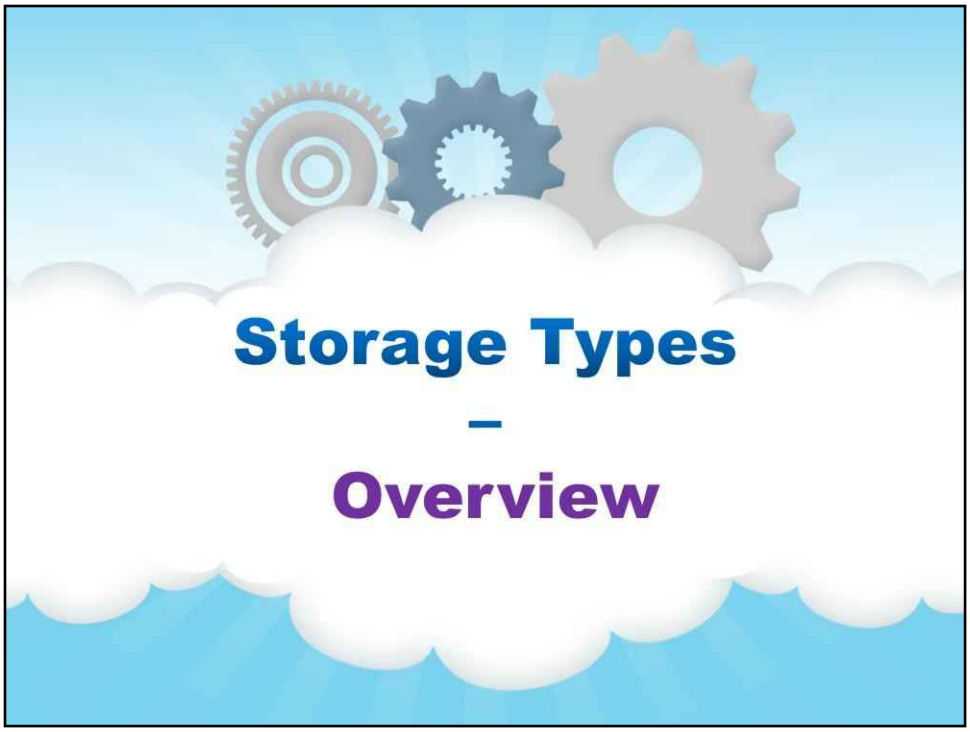
з прикладами їх реалізації:

- AWS,
- Лазурний,
- OpenStack

- віртуалізовані файлові системи у хмарних обчисленнях:

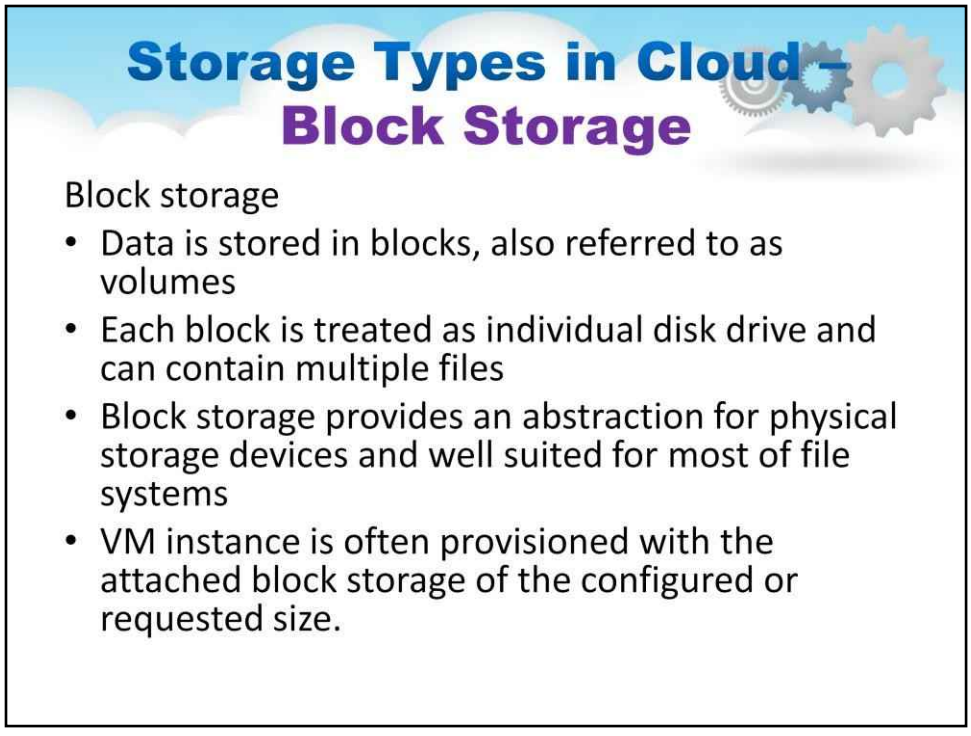
- LVM,
- RAID,
- NFS,
- блиск,
- Цеф,
- Глустер,
- HDFS

з прикладами їх реалізації.



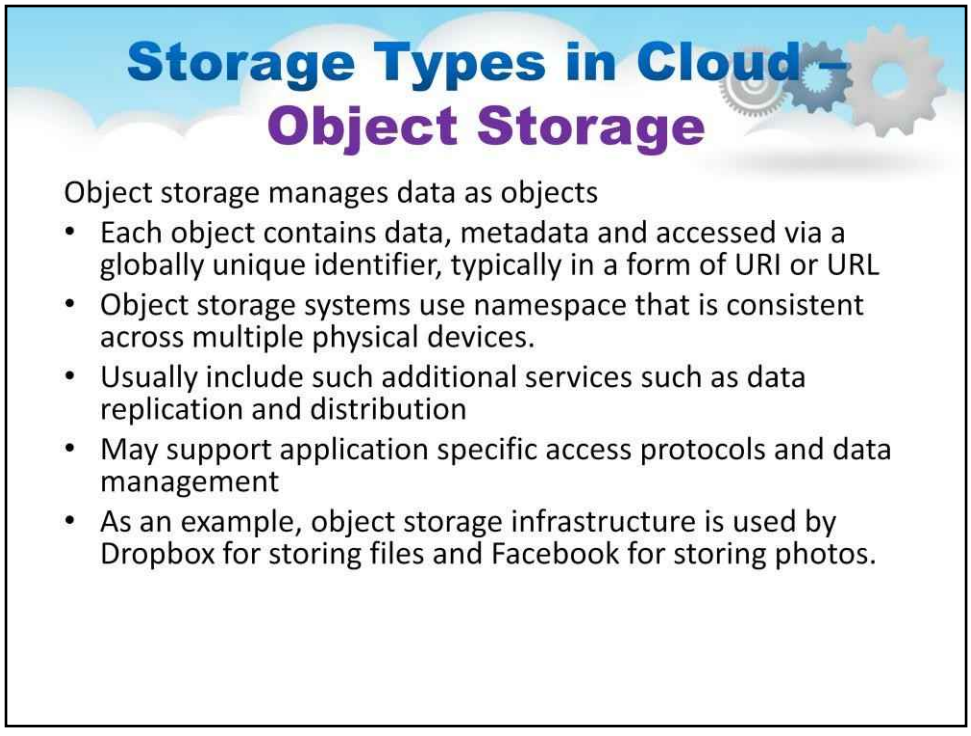
Типи зберігання

Огляд



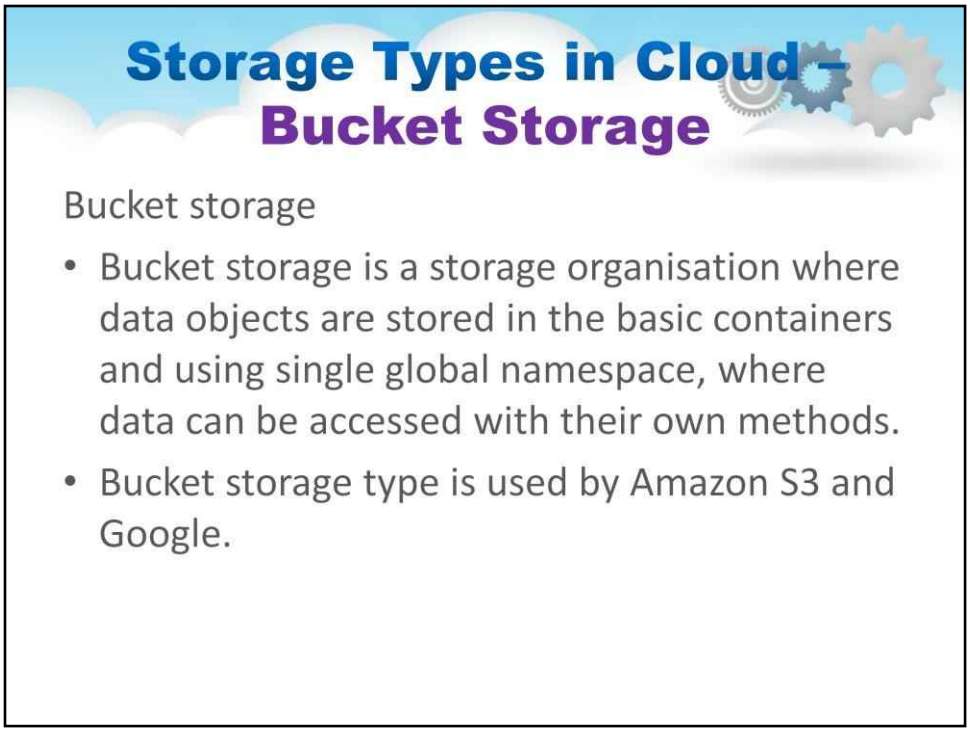
Блок зберігання

Блокове зберігання — це тип зберігання даних, у якому дані зберігаються в блоках, які також називаються томами. Кожен блок розглядається як окремий диск і може містити декілька файлів. Таким чином, блочне сховище забезпечує гарну абстракцію для фізичних пристроїв зберігання та добре підходить для більшості файлових систем. У хмарі екземпляр віртуальної машини часто забезпечується прикріпленим блоковим сховищем налаштованого або запитаного розміру.



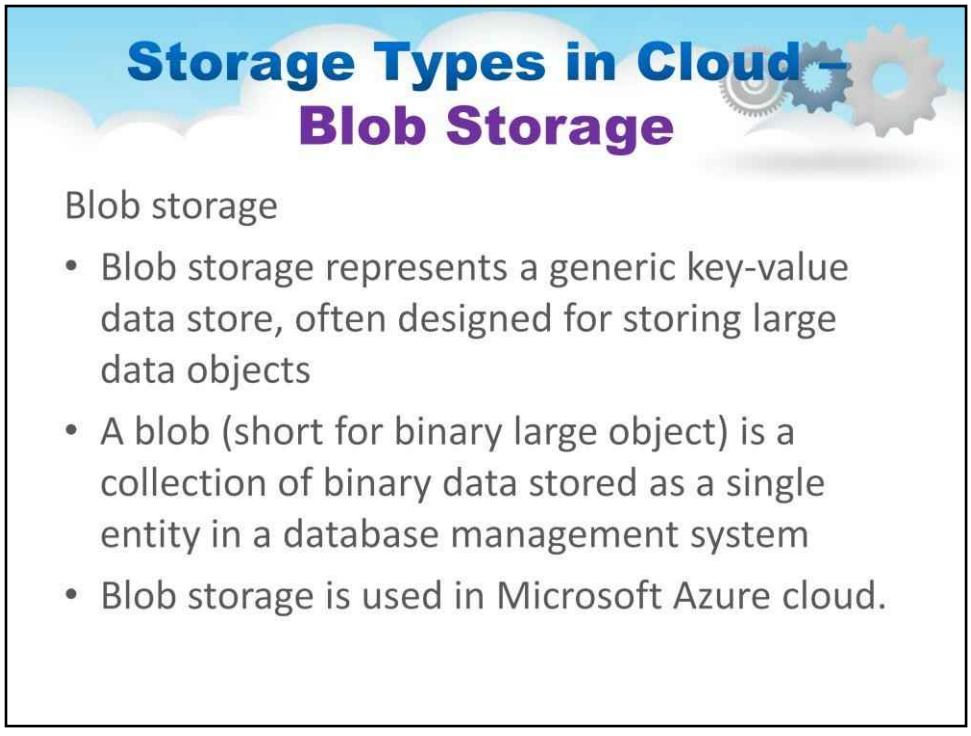
Об'єкт зберігання

Архітектура зберігання, яка керує даними як об’єктами. Кожен об’єкт містить дані, метадані, доступ до яких здійснюється через глобальний унікальний ідентифікатор, зазвичай у формі URI або URL. Системи зберігання об’єктів використовують простір імен, узгоджений для кількох фізичних пристроїв. Системи зберігання об'єктів зазвичай включають такі додаткові послуги, як реплікація та розподіл даних, а також можуть підтримувати протоколи доступу до програм і управління даними. Як приклад, інфраструктура зберігання об’єктів використовується Dropbox для зберігання файлів і Facebook для зберігання фотографій.



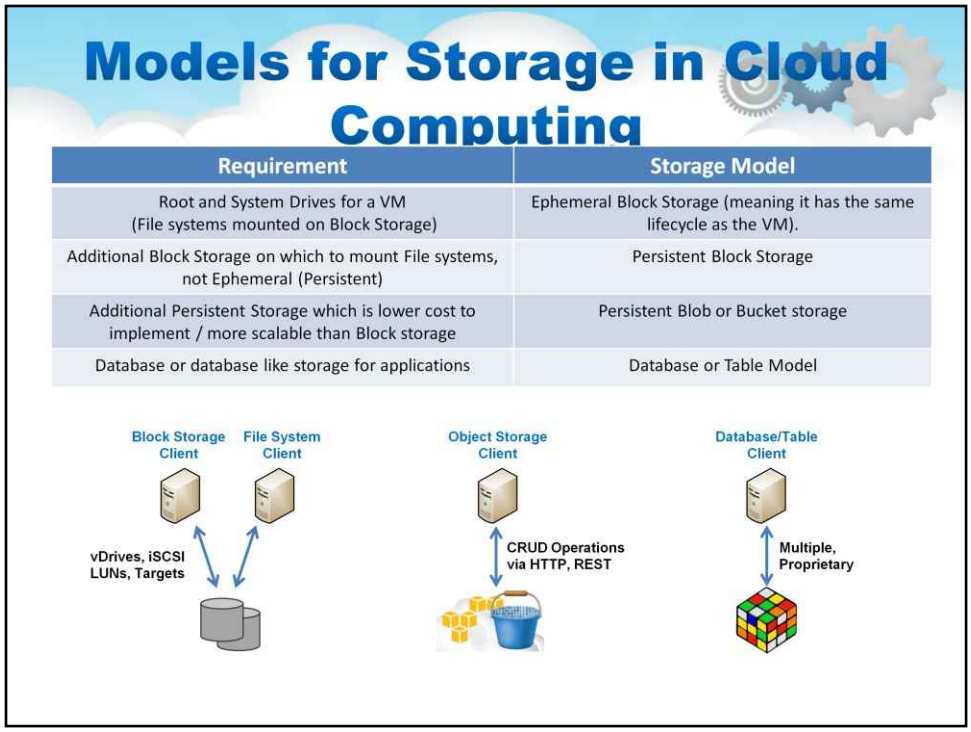
Зберігання відра

Бакетне сховище — це організація зберігання, де об’єкти даних зберігаються в основних контейнерах і використовують єдиний глобальний простір імен, де доступ до даних можна отримати за допомогою власних методів. Тип зберігання відра використовується Amazon S3 і Google.



Зберігання blob

Сховище BLOB-об’єктів представляє загальне сховище даних «ключ-значення», яке часто призначене для зберігання великих об’єктів даних. Blob (скорочення від binary large object) — це набір двійкових даних, які зберігаються як єдине ціле в системі керування базою даних. Сховище BLOB-об’єктів використовується в хмарі Microsoft Azure.



Хмарні обчислення включають кілька моделей зберігання. Деякі з них є необхідними для запуску існуючого програмного забезпечення. У Хмарі також з'явилася велика кількість абсолютно нових моделей зберігання даних, які стали можливими завдяки архітектурі Хмари або стали необхідними (там, де раніше не було потреби) через безпрецедентно великі обсяги даних, знайдені в Хмарі. , якого просто не існувало раніше.

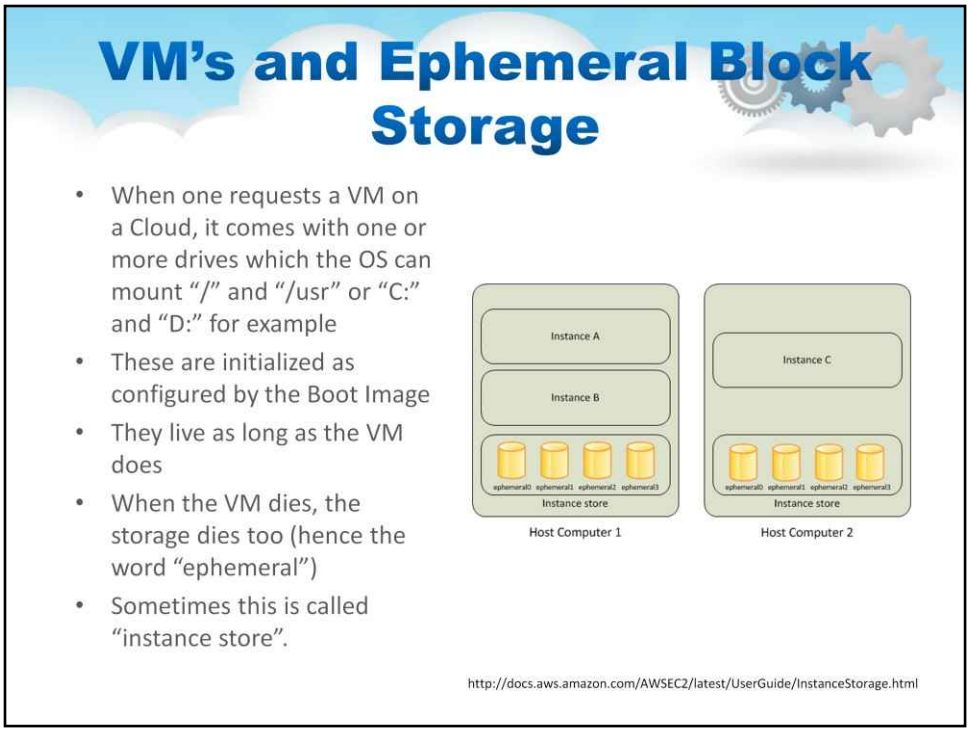
Перший тип сховища виникає через необхідність кореневого та системного дисків для віртуальної машини. Це Файл системи, змонтовані на Block Storage. Їх потрібно доставити разом із створеною віртуальною машиною, і, навпаки, вони не повинні існувати після зникнення віртуальної машини. Тому цей тип сховища називається Ephemeral Block Storage (це означає, що він має той самий життєвий цикл, що й віртуальна машина).

Другий тип — це додаткове блочне сховище, на якому монтуються файлові системи, а не тимчасове (постійне). Це блочне сховище залишатиметься в хмарі незалежно від будь-яких віртуальних машин. Імовірно, віртуальні машини після створення монтуватимуть ці диски та матимуть до них доступ таким чином. Структура блокового сховища повинна підтримуватися фактичною реалізацією сховища. Загалом блочне сховище реалізується простим приєднанням до наявних пристроїв зберігання даних SAN або NAS, які завжди забезпечують блочний інтерфейс. Або це може бути реалізовано в програмному забезпеченні CloudOS, як ми побачимо нижче.

Наступний тип сховища справді виник у хмарі за моделлю стрічкового архіву. У застарілому використанні стрічок можна було мати кілька «архівів», один за одним, на стрічці. Архів не мав структури, це була послідовність бітів. Можна, наприклад, безпосередньо помістити образ диска на стрічковий архів. Або файл «tar» (який у певному сенсі схожий на сучасний ZIP-файл) можна помістити на стрічковий архів (насправді tar означає Tape ARchive). Стрічкові архіви являли собою «відра бітів» на послідовних магнітних носіях. На хмарі ми маємо

те саме поняття, яке ми називаємо об'єктом, або BLOB, або відерним зберіганням. Об'єктне зберігання реалізовано в хмарі розумними способами, щоб забезпечити реплікацію даних для високої доступності. Часто це найнижчий варіант зберігання в хмарі.

Нарешті, багато програм потребують високоструктурованого сховища, як база даних, для своїх програм. У той час як багато програм використовують системи RDBM із дуже складними запитами SQL, багато програм використовують дуже прості бази даних, без складних об'єднань або збережених процедур, їм потрібен якийсь простий спосіб виконувати пошук на основі стовпців або таблиць. Як ми побачимо пізніше, реалізація баз даних на розподіленій архітектурі створює певні проблеми, і щоб відповісти на ці проблеми, використовуючи різні компроміси, існує багато різних типів баз даних і варіантів, подібних до них.



Першою моделлю хмарного сховища є Ephemeral Block Storage.

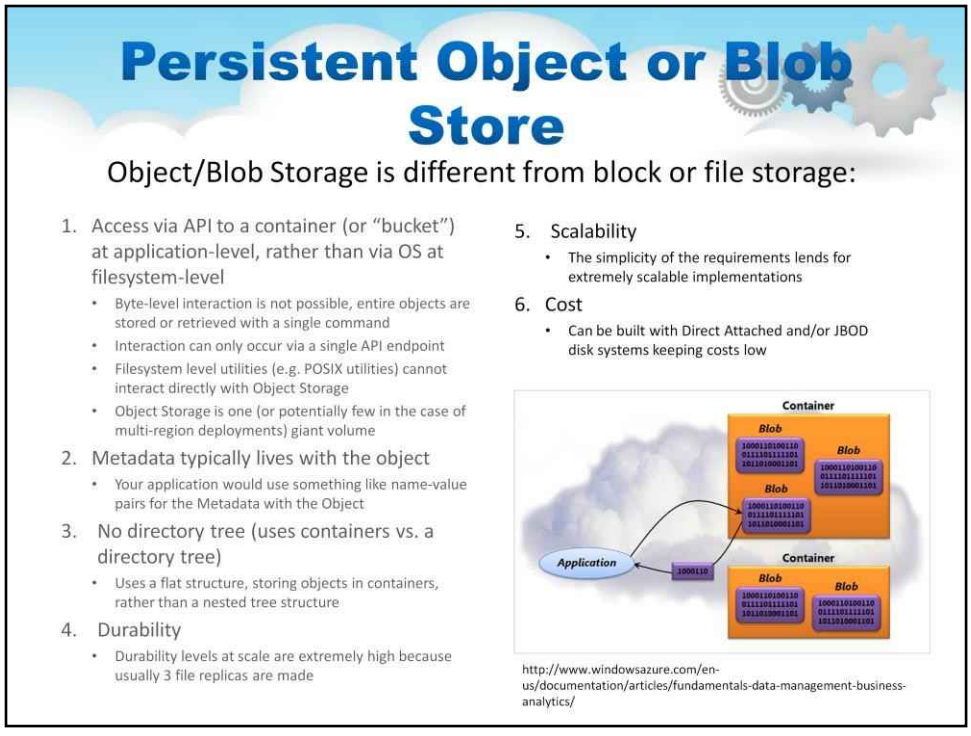
Коли хтось запитує віртуальну машину в хмарі, вона постачається з одним або декількома дисками, які ОС може підключити, наприклад, «/» і «/usr» або «C:» і «D:».

Вони ініціалізуються відповідно до конфігурації завантажувального образу

Вони живуть стільки ж, скільки і ВМ

Коли віртуальна машина вимикається, сховище також вмирає (звідси слово «ефемерний»).

Іноді це називається «сховище екземплярів».



Постійне сховище об'єктів або блобів відрізняється від сховища блоків або файлів.

Знову ж таки, подумайте про це як про те, що ми називаємо «файлом» або «архівом» на стрічці. Це потік бітів між маркерами архіву. Ви можете транслювати зі стрічки або на стрічку, ви не можете шукати всередині архіву.

Доступ до Object Storage здійснюється через API до контейнера (або «відра») на рівні програми, а не через ОС на рівні файлової системи

Взаємодія на рівні байтів неможлива, цілі об'єкти зберігаються або витягуються за допомогою однієї команди

Утиліти рівня файлової системи (наприклад, утиліти POSIX) не можуть безпосередньо взаємодіяти з Object Storage

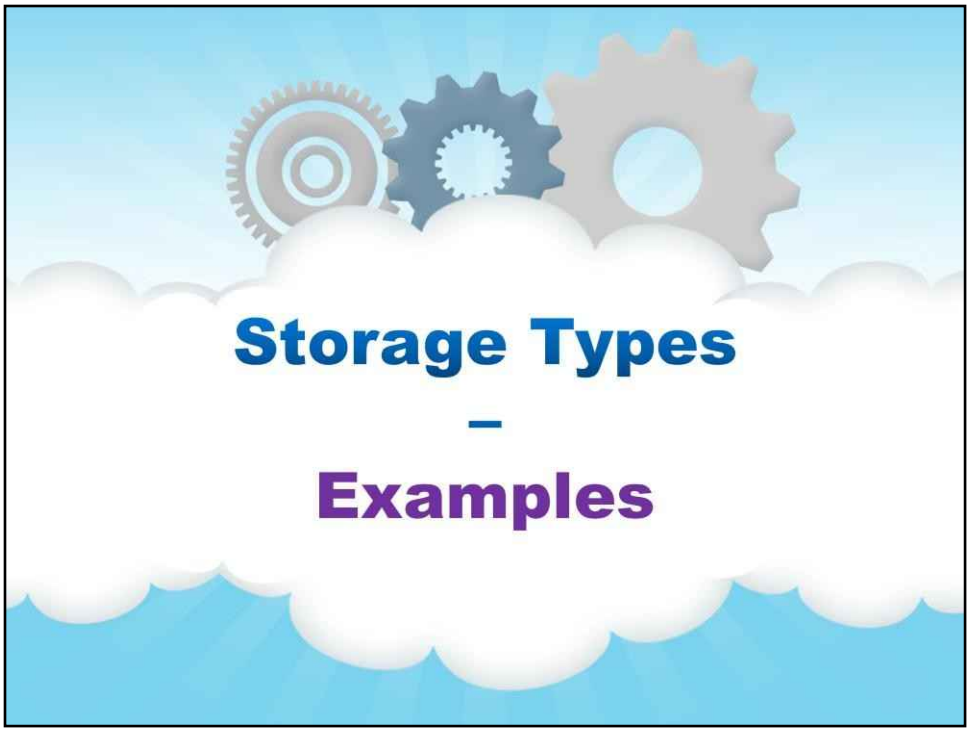
Об'єктне сховище – це один (або потенційно кілька у випадку розгортання в кількох регіонах) гігантський обсяг. Метадані зазвичай живуть разом з об'єктом

Ваша програма використовуватиме щось на зразок пар ім'я-значення для метаданих з об'єктом

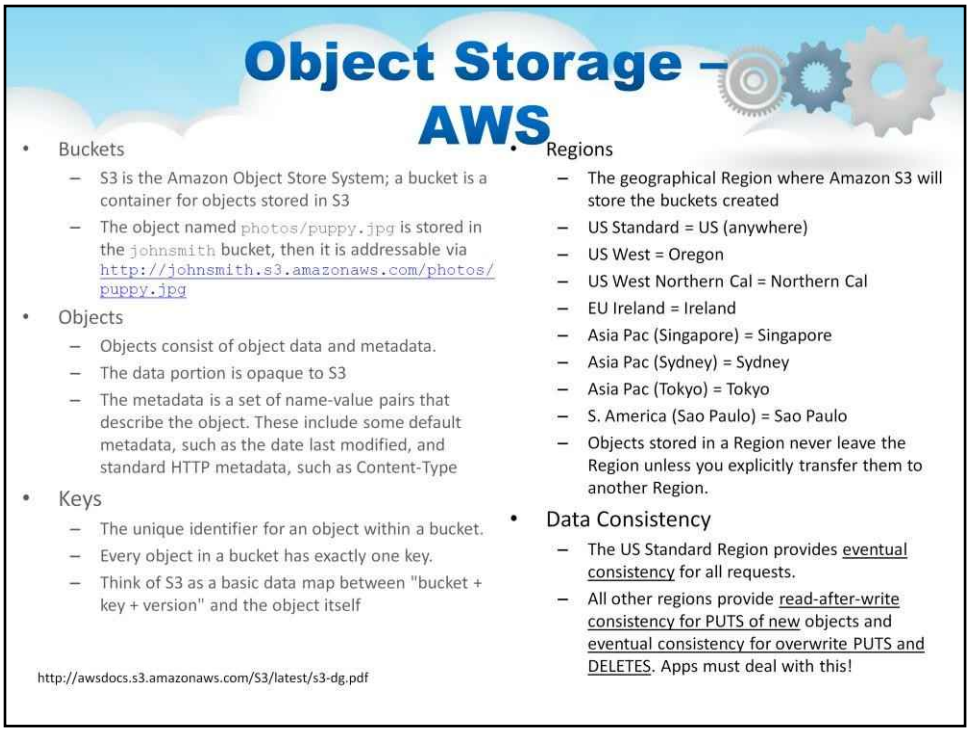
У сховищі об'єктів немає ні структури, ні дерева каталогів. Він використовує плоску структуру, зберігаючи об'єкти в контейнерах, а не вкладену структуру дерева

Рівні довговічності в масштабі надзвичайно високі, оскільки зазвичай створюються 3 репліки файлів

Простота вимог забезпечує надзвичайно масштабовані реалізації з недорогими накопичувачами та впровадженням чистого програмного забезпечення, зберігаючи низькі витрати



Приклади



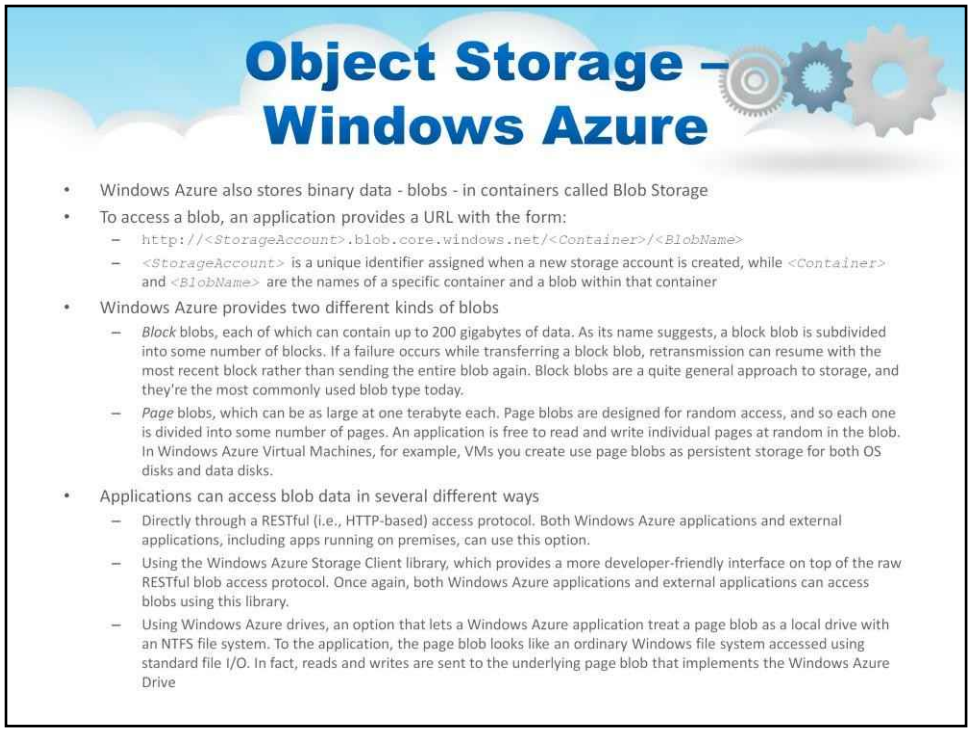
Давайте розглянемо характеристики реалізації Object Storage в Amazon AWS.

S3 — це назва системи зберігання об’єктів Amazon; відро — це контейнер для об’єктів, що зберігаються в S3. Об’єкт під назвою photos/puppy.jpg зберігається у відрі johnsmith, тоді його можна адресувати через http://johnsmith.s3.amazonaws.com/photos/puppy.jpg

Об’єкти складаються з даних об’єкта та метаданих. Частина даних є непрозорою для S3. Метадані — це набір пар ім’я-значення, які описують об’єкт. До них належать деякі метадані за замовчуванням, такі як дата останньої зміни, і стандартні метадані HTTP, такі як Content-Type

Ключі — це унікальний ідентифікатор об’єкта в сегменті.
Кожен об’єкт у відрі має рівно один ключ.
Подумайте про S3 як про базову карту даних між "відром + ключом + версією" та самим об'єктом

Що стосується зберігання об’єктів, треба знати, що вони роблять! Як показано на слайді, різні регіони AWS поводяться по-різному з Object Storage.



Windows Azure також зберігає двійкові дані – blob-об’єкти – у контейнерах під назвою Blob Storage

Windows Azure надає два різних типи blob-файлів

Блокувати blobs, кожна з яких може містити до 200 гігабайт даних. Як випливає з назви, блочна крапля підрозділяється на певну кількість блоків. Якщо під час передачі блочного blob-об’єкта сталася помилка, повторна передача може відновитися з останнього блоку, а не знову надсилати весь blob-об’єкт. Блокові blob-об’єкти є досить загальним підходом до зберігання, і вони є найбільш часто використовуваним типом blob-об’єктів сьогодні.

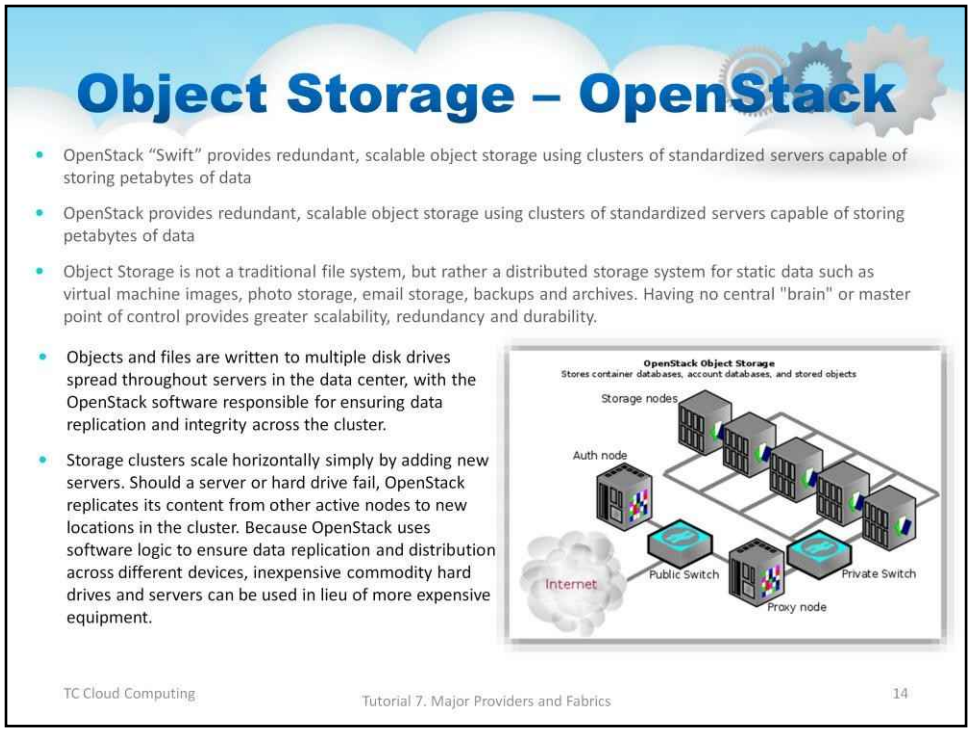
Сторінка blobs, розмір яких може становити один терабайт кожен. Блоки сторінок призначені для довільного доступу, тому кожен розділений на певну кількість сторінок. Програма може вільно читати та записувати окремі сторінки випадковим чином у blob. Наприклад, у віртуальних машинах Windows Azure віртуальні машини, які ви створюєте, використовують блоби сторінок як постійне сховище як для дисків ОС, так і для дисків із даними.

Програми можуть отримувати доступ до даних blob кількома різними способами

Безпосередньо через протокол доступу RESTful (тобто на основі HTTP). Цю опцію можуть використовувати як програми Windows Azure, так і зовнішні програми, включно з локальними програмами.

Використання клієнтської бібліотеки Windows Azure Storage, яка надає більш зручний для розробників інтерфейс на основі необробленого протоколу доступу до blob RESTful. Знову ж таки, як програми Windows Azure, так і зовнішні програми можуть отримувати доступ до blob-файлів за допомогою цієї бібліотеки.

Використання дисків Windows Azure, параметр, який дозволяє програмі Windows Azure розглядати сторінковий blob як локальний диск із файловою системою NTFS. Для програми blob сторінки виглядає як звичайна файлова система Windows, доступ до якої здійснюється за допомогою стандартного файлового введення-виведення. Фактично, читання та запис надсилаються до основної сторінки, яка реалізує Windows Azure Drive



Проект OpenStack Object Store, відомий як Swift, пропонує хмарне програмне забезпечення для зберігання та отримання великої кількості даних за допомогою простого API. Його створено для масштабування та оптимізовано для довговічності, доступності та паралельності всього набору даних. Swift ідеально підходить для зберігання неструктурованих даних, які можуть рости без обмежень.

Swift забезпечує резервне, масштабоване сховище об’єктів за допомогою кластерів стандартизованих серверів, здатних зберігати петабайти даних

Об’єкти та файли записуються на кілька дисків, розподілених по серверах у центрі обробки даних, при цьому програмне забезпечення OpenStack відповідає за забезпечення реплікації та цілісності даних у кластері.

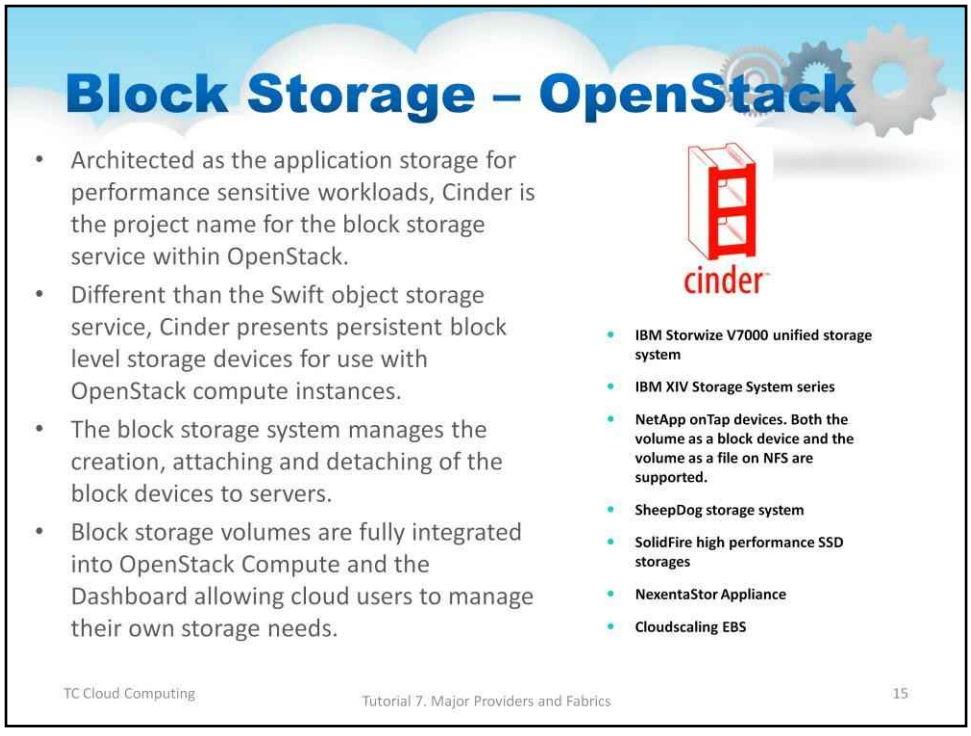
Swift має багато функцій як для кінцевих користувачів, так і для системних адміністраторів, які керують системою.

Версійний пише

CORS
ACL

Як завгодно великі об'єкти
Статичний хостинг сайтів
Підписані URL-адреси, термін дії яких закінчується
Спеціальні метадані

Масові операції
Багатодіапазонні запити



Cinder — це служба блокового зберігання для OpenStack. Його розроблено, щоб дозволити використання будь-якої еталонної реалізації (LVM) для представлення ресурсів зберігання кінцевим користувачам, які можуть використовуватися OpenStack Compute Project (Nova).

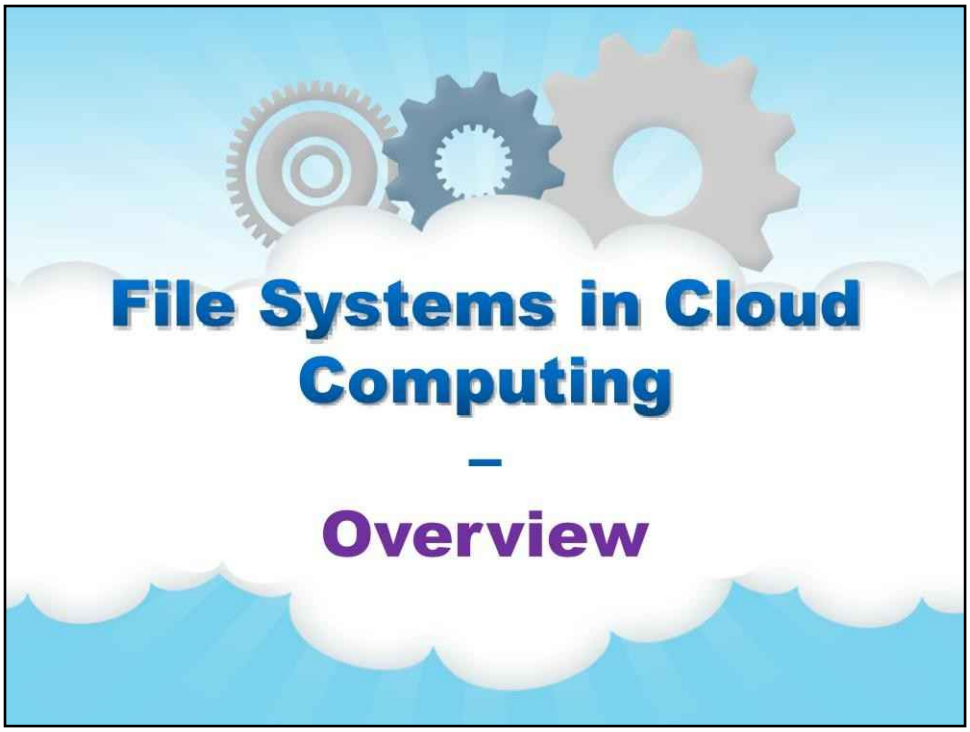
Короткий опис Cinder полягає в тому, що він віртуалізує пули блокових пристроїв зберігання даних і надає кінцевим користувачам API самообслуговування для запиту та використання цих ресурсів

Розроблений як сховище додатків для чутливих до продуктивності робочих навантажень, Cinder — це назва проекту служби зберігання блоків у OpenStack.

На відміну від служби зберігання об’єктів Swift, Cinder представляє пристрої зберігання постійних блоків для використання з обчислювальними екземплярами OpenStack.

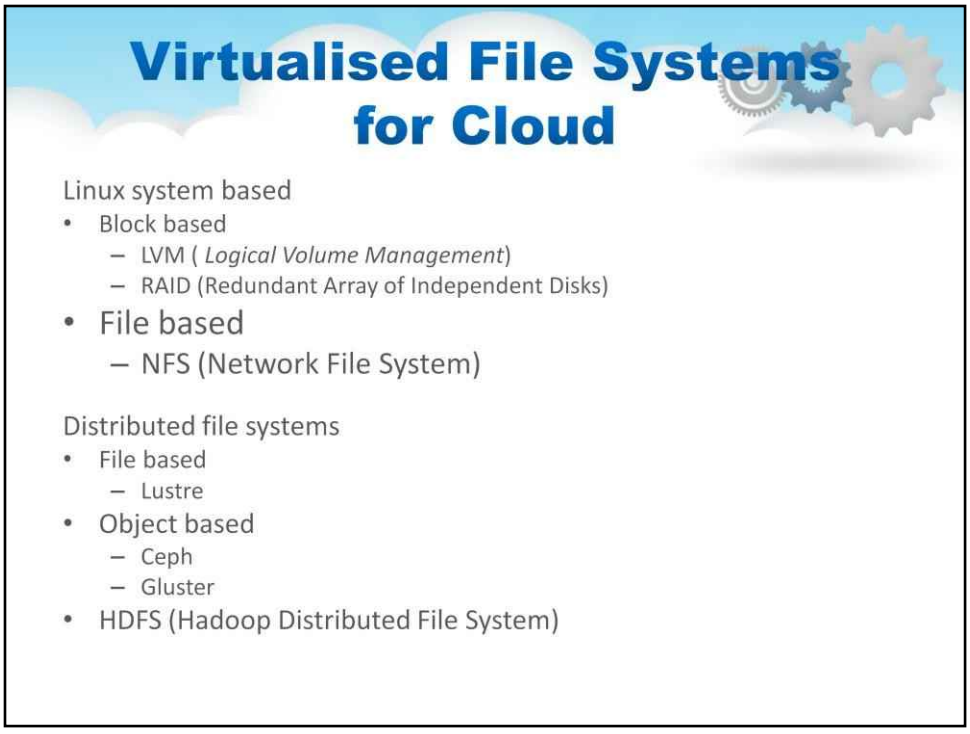
Блокова система зберігання керує створенням, підключенням і від’єднанням блокових пристроїв до серверів.

Блокові томи сховища повністю інтегровані в OpenStack Compute та інформаційну панель, що дозволяє користувачам хмари керувати власними потребами в сховищі.



Файлові системи в хмарних обчисленнях

Огляд



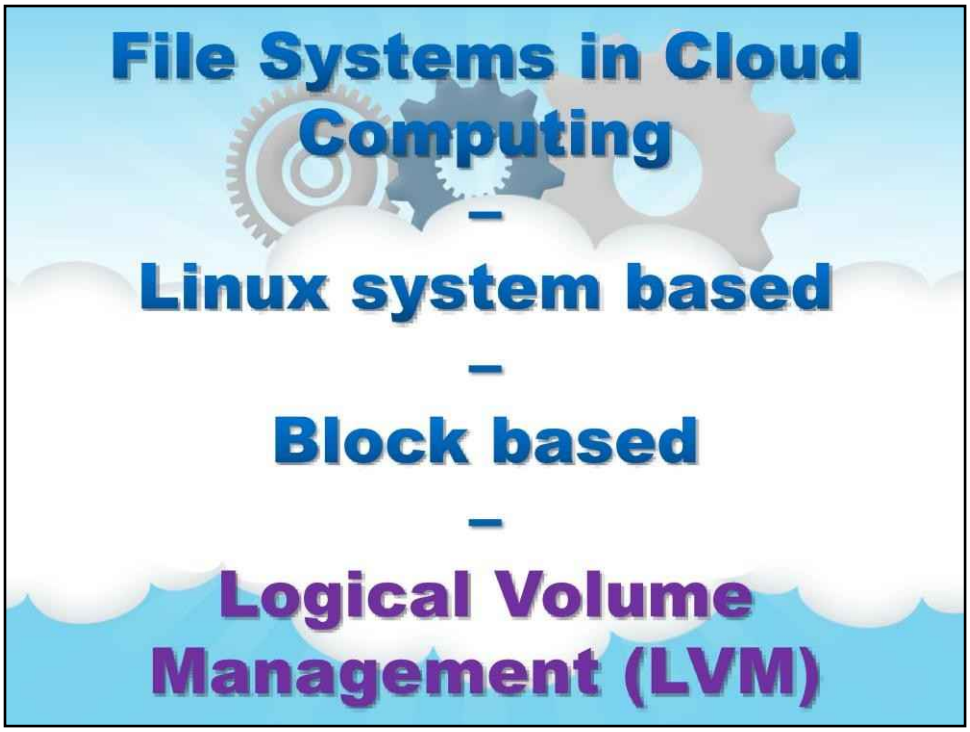
Ми розглянемо деякі популярні файлові системи для хмари.
Кожен з них має низку переваг при реалізації в конкретному хмарному середовищі.

Більшості програм потрібні файлові системи, оскільки сервери мають файлові системи. Більшість файлових систем потребують блокового сховища для монтування. Тому не дивно, що файлові системи з базовим блоковим сховищем є популярними варіантами хмарного сховища.

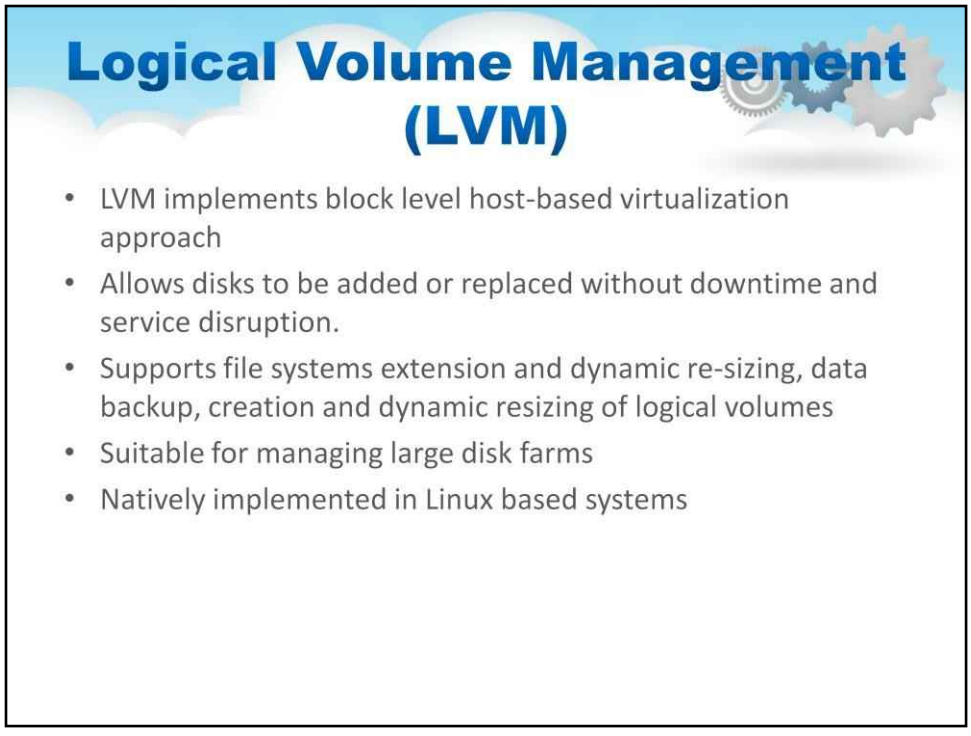
Простий підхід полягає у використанні технологій на базі сервера або NAS, розширених безпосередньо до хмари для блокових файлових систем. Це означає зв'язування кількох дисків і, можливо, доступ через мережу. У списку слайдів LVM, RAID і NFS є прикладами віртуалізованих файлових систем у блочному сховищі, які зазвичай зустрічаються в невеликих хмарах.

Більші хмари використовують розподілені файлові системи, які мають високий ступінь резервування в хмарі, яку вони обслуговують. Як показано на слайді, вони можуть бути файловими або об'єктними, і є багато прикладів обох, популярних у хмарних реалізаціях.

HDFS (розподілена файлова система Hadoop) спеціально розроблена для великомасштабної обробки даних у масивних паралельних кластерах. Може використовуватися в хмарі для високопродуктивного введення/виведення даних, зокрема для CDN (мережі розподілу вмісту)


File Systems in Cloud Computing
–
Linux system based
–
Block based
–
Logical Volume Management (LVM)

LVM

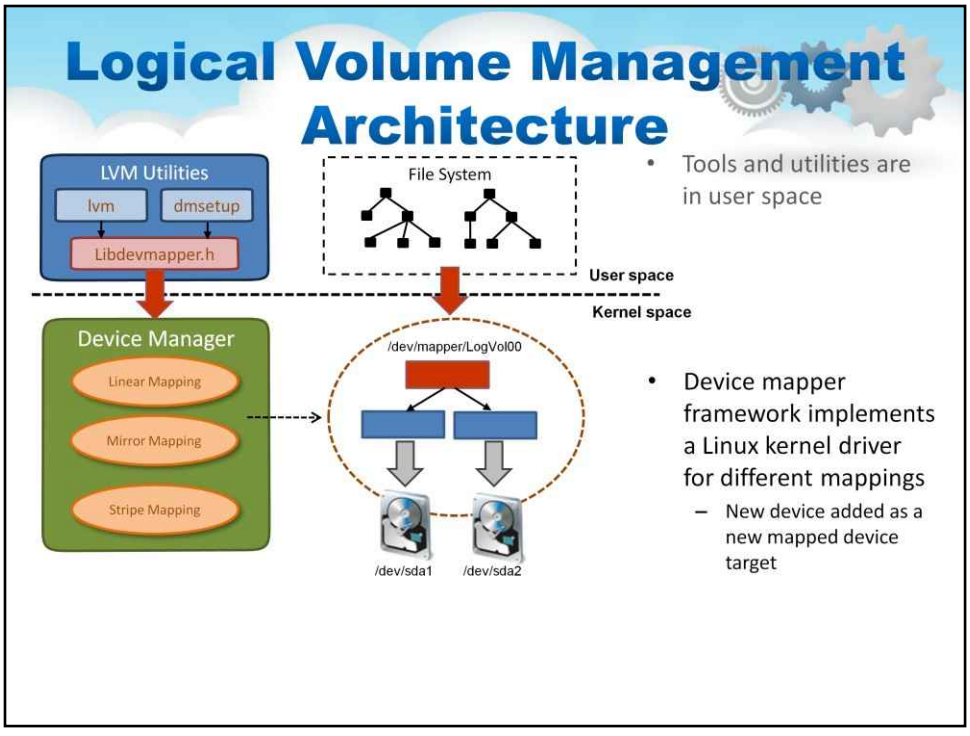


Менеджер логічних томів дуже популярний, оскільки він зазвичай зустрічається в Linux. Він реалізує підхід віртуалізації на основі хосту на блочному рівні

Дозволяє додавати або замінювати диски без простоїв і збоїв у роботі.

Підтримує розширення файлових систем і динамічну зміну розміру, резервне копіювання даних, створення та динамічну зміну розміру логічних томів

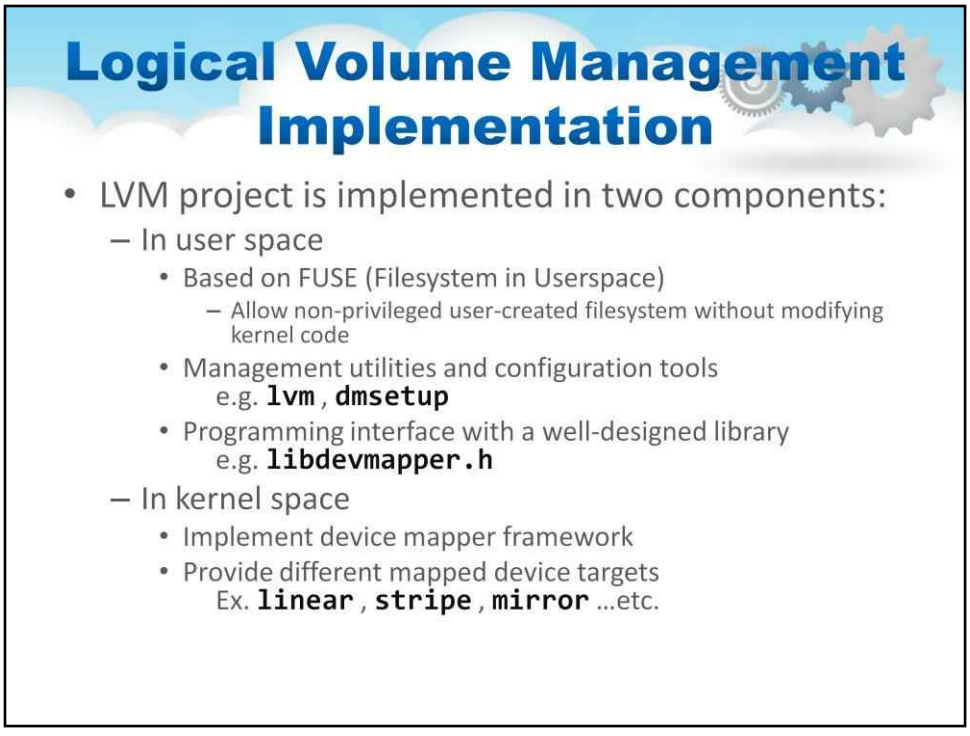
Підходить для керування великими дисковими фермами



Цей слайд ілюструє архітектуру керування логічним томом

Інструменти та утиліти знаходяться в просторі користувача

Структура відображення пристроїв реалізує драйвер ядра Linux для різних відображень

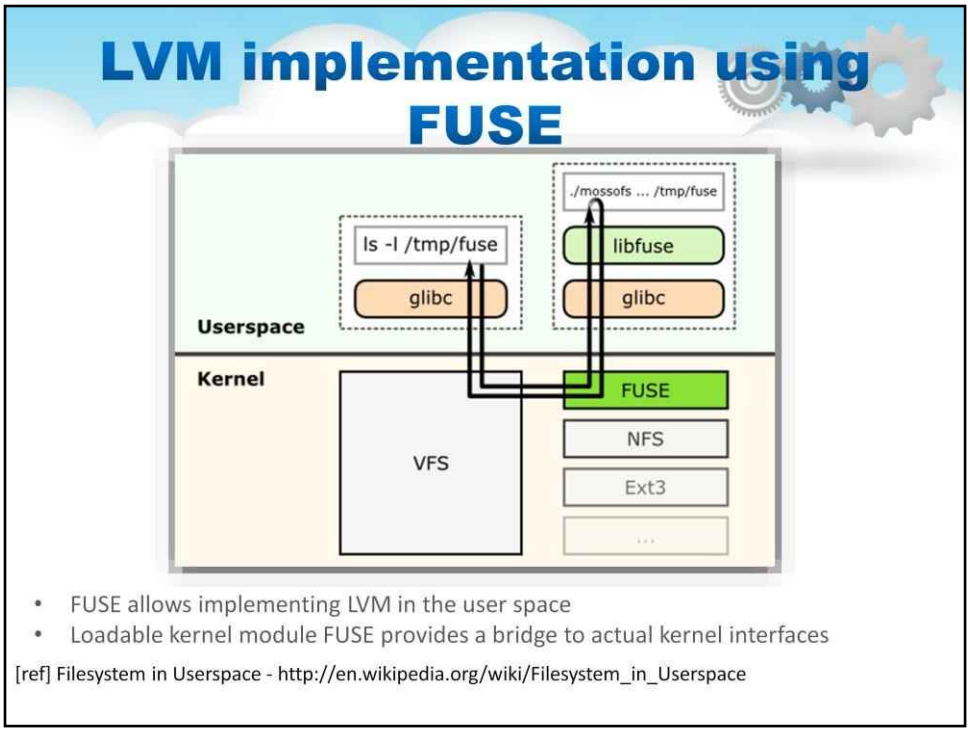


Впровадження керування логічним томом

Проект LVM реалізований у двох компонентах:

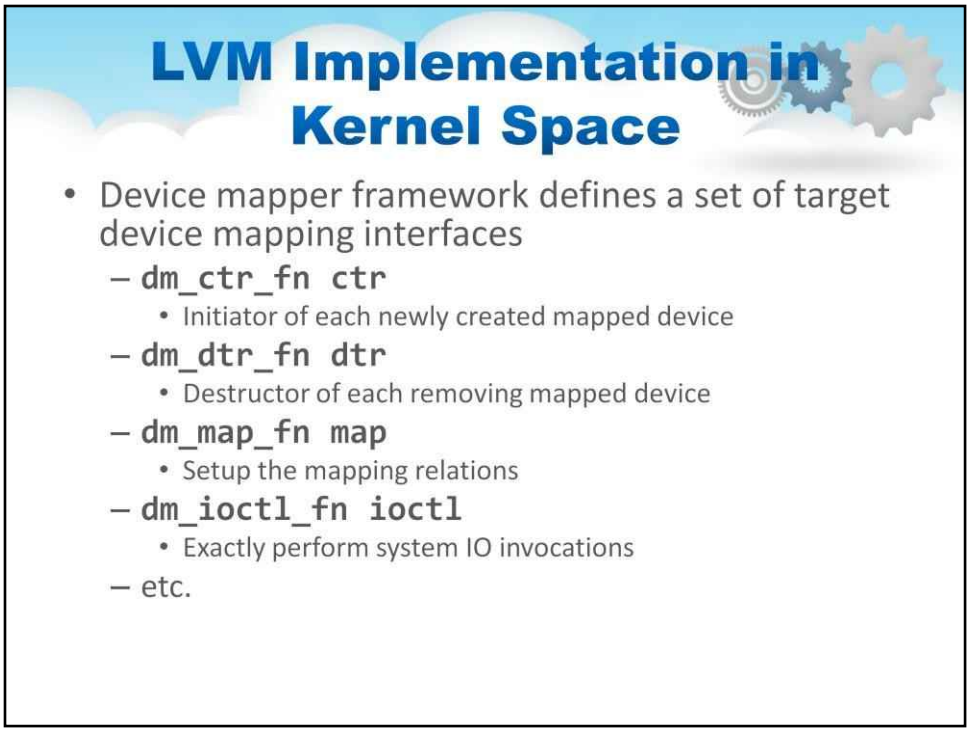
У просторі користувача На основі FUSE (файлова система в просторі користувача)

У просторі ядра, який реалізує структуру відображення пристроїв



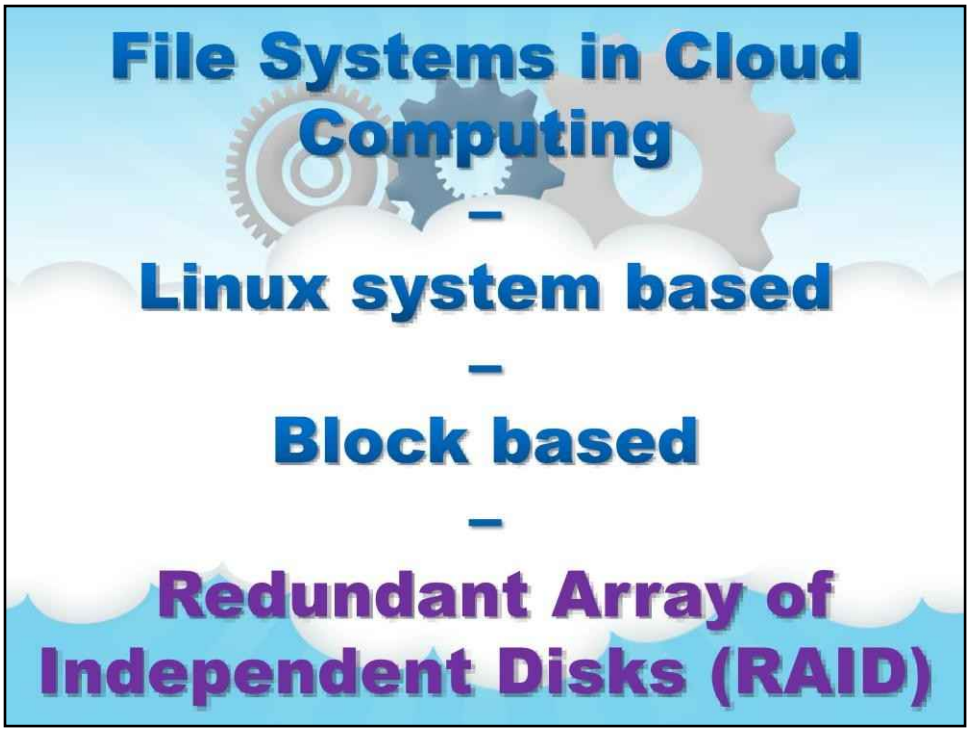
На цьому слайді зображено диспетчер логічних томів за допомогою файлової системи в просторі користувача

Як видно з діаграми, ключем є те, що завантажуваний модуль ядра FUSE забезпечує міст до реальних інтерфейсів ядра

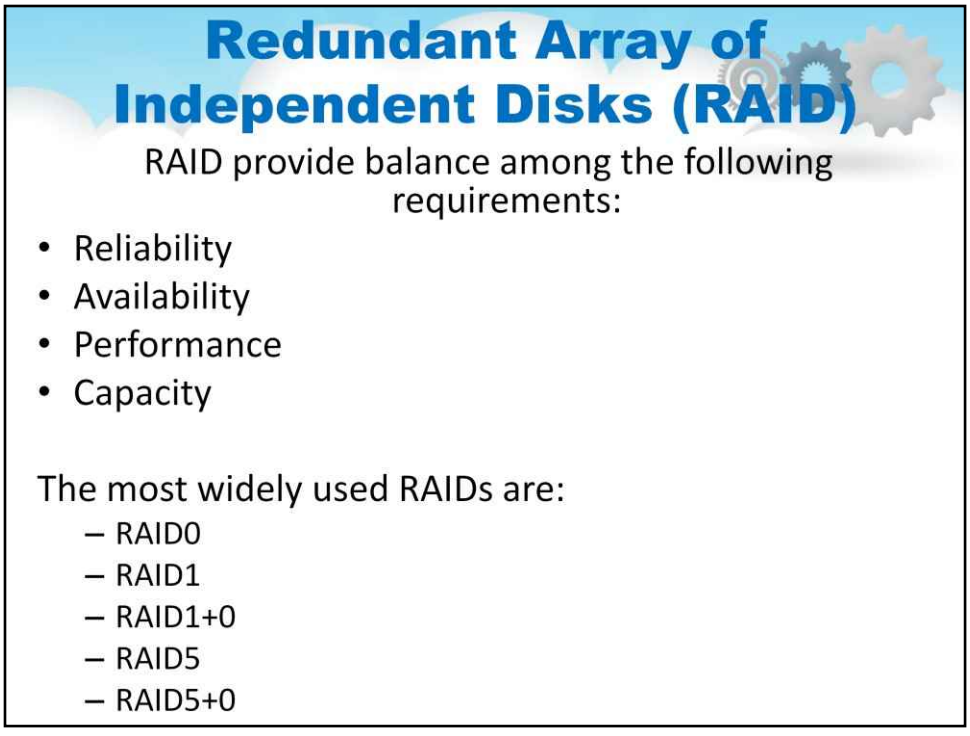


З міркувань продуктивності доступна реалізація LVM у просторі ядра

Цей слайд говорить про системні виклики, реалізовані таким чином


File Systems in Cloud Computing
–
Linux system based
–
Block based
–
Redundant Array of Independent Disks (RAID)

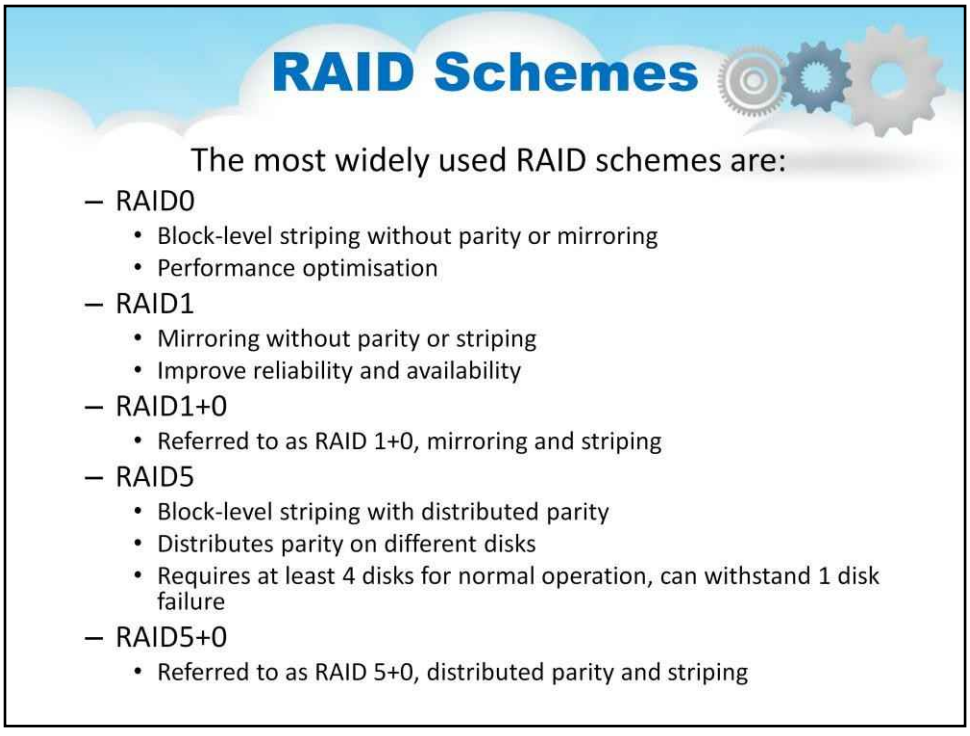
RAID



Іншою поширеною схемою віртуалізації сховища є RAID (надлишковий масив незалежних дисків).

RAID — це програмний рівень, який об’єднує диски та реалізує різні рівні реплікації та розподілу даних між дисками.

Схеми RAID забезпечують різний баланс між ключовими цілями: надійність, доступність, продуктивність, ємність



Слайд описує відмінності в поширених схемах RAID:

RAID0

Смуга на рівні блоку без паритету чи дзеркального відображення

Оптимізація продуктивності

RAID1

Дзеркалювання без паритету чи смуги

Покращення надійності та доступності

RAID1+0

Позначається як RAID 1+0, дзеркальне відображення та чередування

RAID5

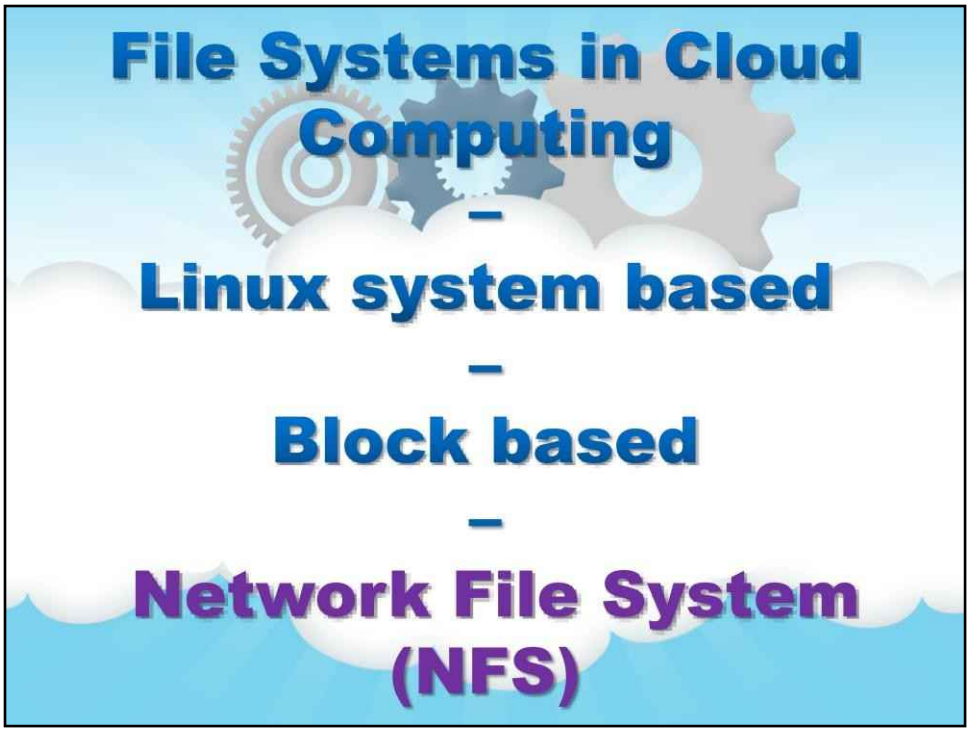
Черекування на рівні блоку з розподіленою

парністю Розподіляє паритет на різних дисках

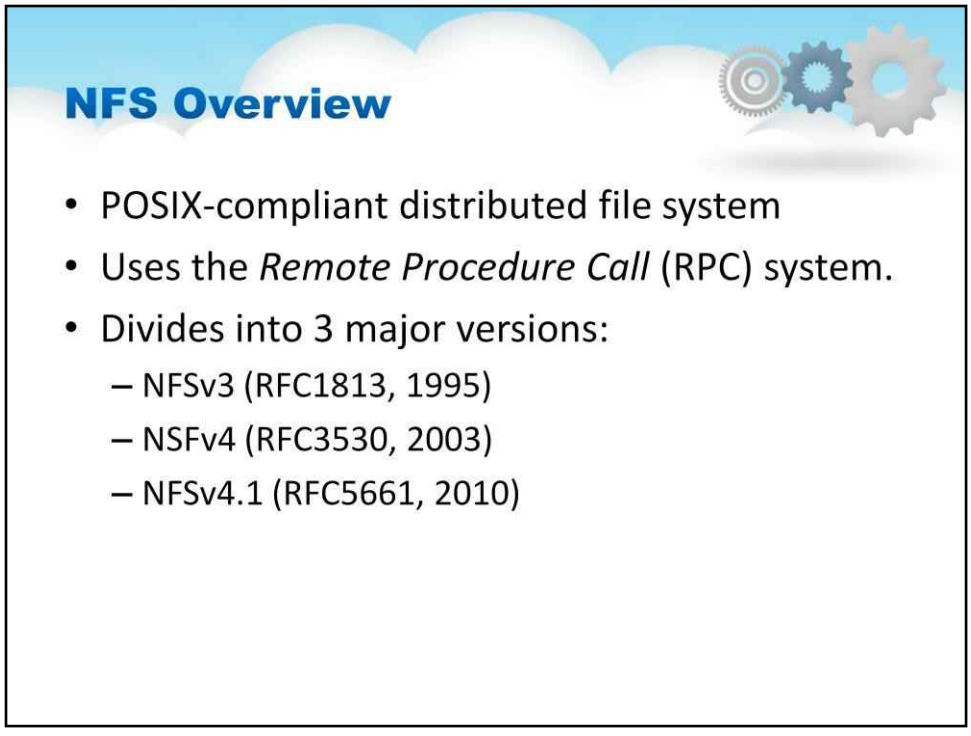
Для нормальної роботи потрібно не менше 4 дисків, витримує 1 збій диска

RAID5+0

Називається RAID 5+0, розподілений паритет і чередування


File Systems in Cloud Computing
–
Linux system based
–
Block based
–
Network File System (NFS)

NFS



Мережна файлова система є відкритим стандартом, визначеним у RFC.

NFS є звичайним модулем ядра Linux.

NFSv3 забезпечує:

Підтримка 64-розрядних розмірів файлів і зсувів для обробки файлів розміром понад 2 ГБ Підтримка протоколу TCP і асинхронного запису на сервері

NSFv4 і NFSv4.1 забезпечують:

Покращення продуктивності, забезпечення надійної безпеки та запровадження протоколу з відстеженням стану

Підтримує розгортання кластерних серверів, включаючи масштабований паралельний доступ до файлів, розподілених між кількома серверами (розширення pNFS)

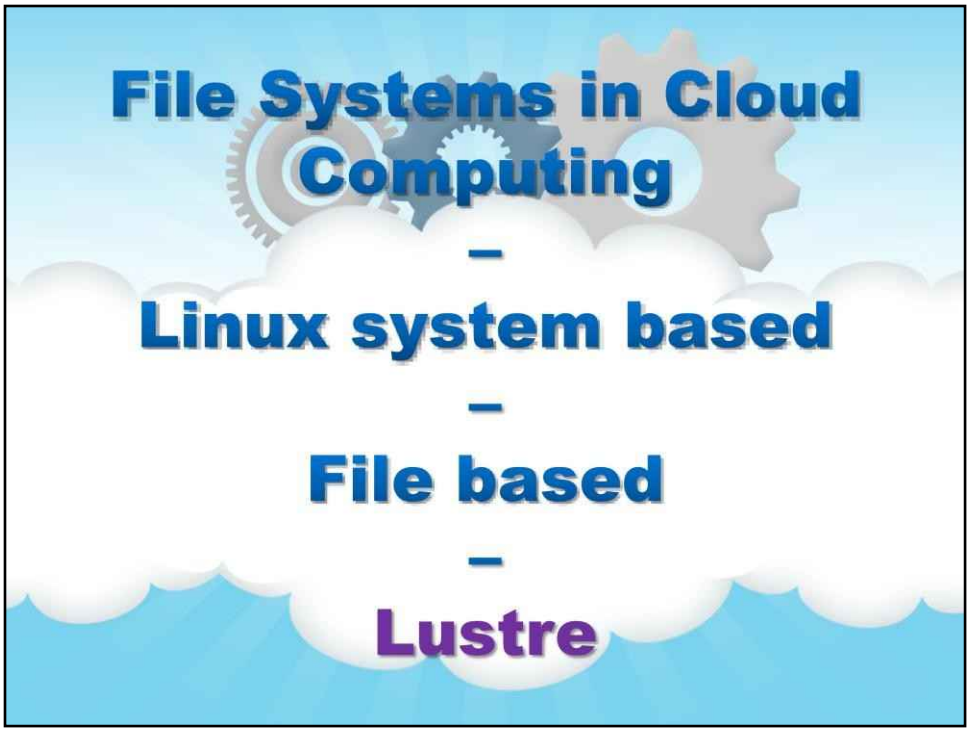
Файлові системи мають особливості, на які покладаються розробники програм. Наприклад, коли виклик «write» повертається від ядра до програми користувача, користувач

програма припускає, що дані фактично записані, або принаймні, що наступне читання тих самих даних поверне те, що було щойно записане. Ці поведінкові припущення є частиною специфікації POSIX.

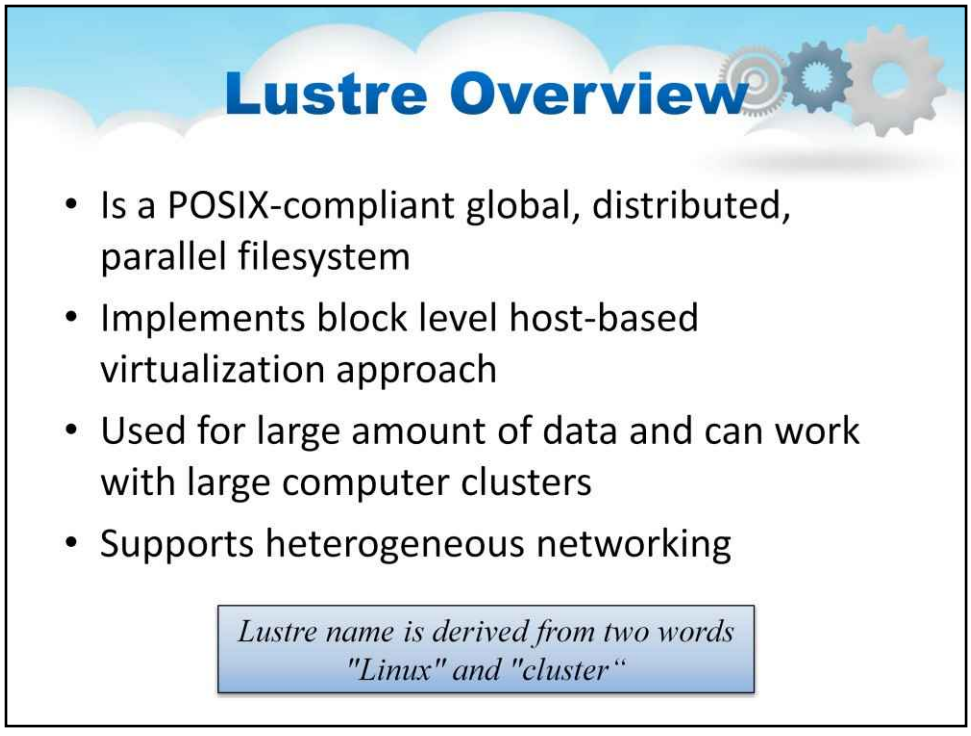
Коли використовуються кластери дисків і коли файлові системи експортуються через мережу, стає складно відповідати всім вимогам файлової системи POSIX. Дуже популярними стали мережеві файлові системи, які мали правильну поведінку. Мережева файлова система SUN (NFS) була однією з перших і найнадійніших POSIX-сумісних розподілених файлових систем.

Як показано на слайді, NFS визначено низкою RFC, а протоколи, які використовуються для реалізації NFS, добре відомі та зрозумілі. З роками NFS стала мережевою файловою системою «перехід до».

Зокрема, NFS було розширено для підтримки розгортання кластерних серверів, включаючи масштабований паралельний доступ до файлів, розподілених між кількома серверами (розширення pNFS). Ця технологія є ключовою частиною, наприклад, реалізації приватної хмари IBM.


File Systems in Cloud Computing
–
Linux system based
–
File based
–
Lustre

Блиск



Lustre — це тип паралельної розподіленої файлової системи, яка використовується для великих обсягів даних і може працювати з великими комп’ютерними кластерами.

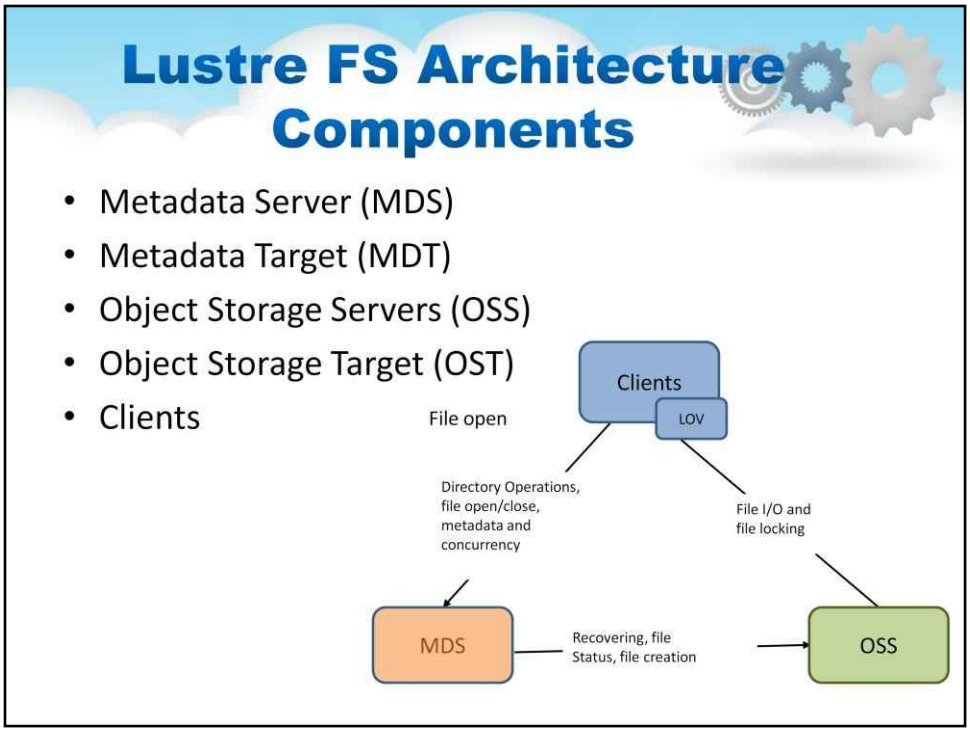
Назва Luster походить від двох слів «Linux» і «cluster».

Lustre часто використовується як файлова система для суперкомп'ютерів і багатосайтових комп'ютерних кластерів.

Кластер зберігання Lustre може містити тисячі вузлів і петабайт обсягу зберігання.

Архітектура Lustre включає три основні компоненти: сервери метаданих, які зберігають інформацію про файлову систему (файли та каталоги), а також права доступу, сервери зберігання об’єктів і клієнти, які отримують доступ і використовують дані.

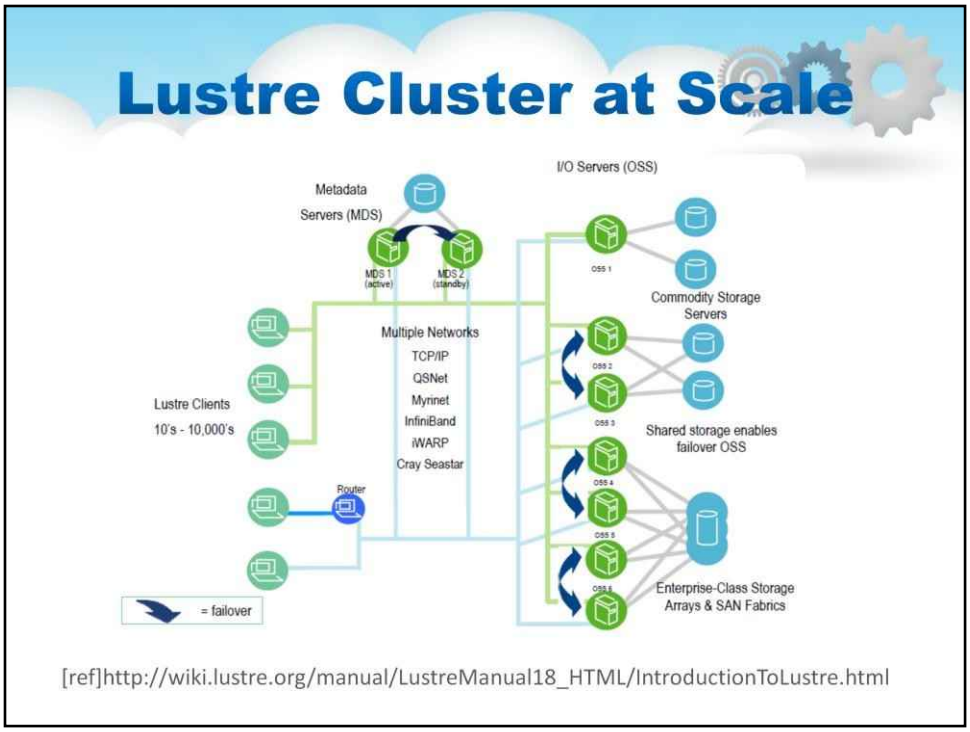
Lustre використовує уніфікований простір імен, сумісний із семантикою POSIX.



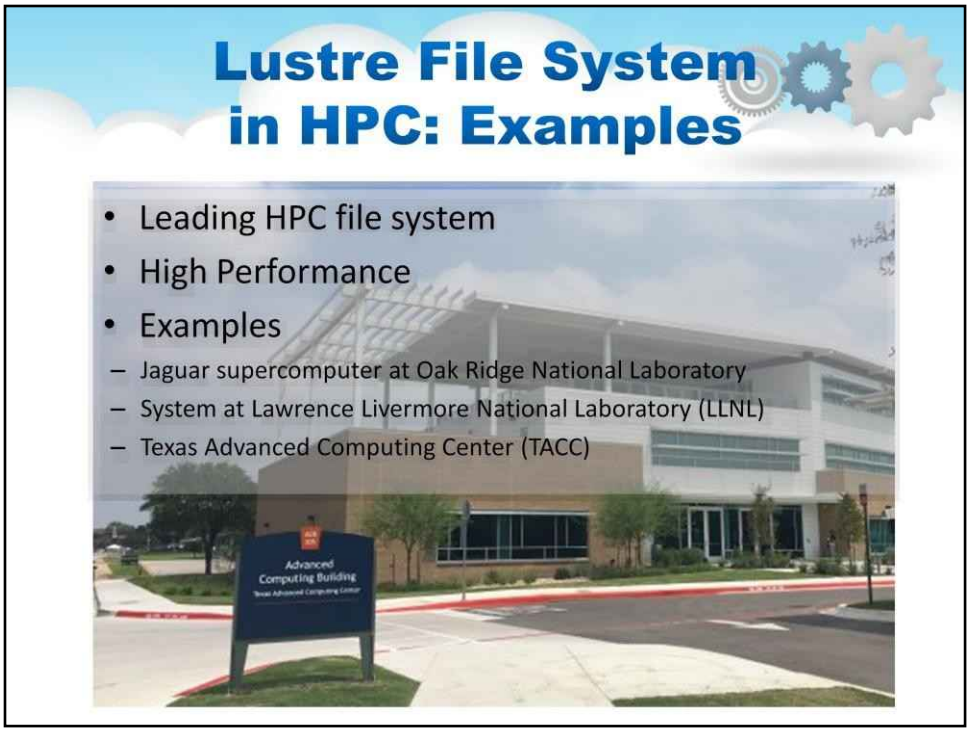
Сервер MDS створює метадані, що зберігаються в одному або кількох MDT.
MDT зберігає метадані (такі як імена файлів, дозволи) на MDS.
OSS забезпечує службу введення/виведення файлів і обробку мережевих запитів для одного або кількох локальних OST

OST зберігає дані файлів як об’єкти даних в одному або кількох OSS. Групи OSS обернуті в логічний том об’єкта (LOV).

На цьому слайді описано основні компоненти файлової системи Luster. Зауважте, що одним із найважливіших елементів дизайну є поняття багатьох серверів зберігання об’єктів. Це забезпечує масштабованість дизайну.



Цей слайд ілюструє дизайн масштабованості, представлений на попередньому слайді. Загалом у великих розгортаннях використовуються концепції високої доступності та високої масштабованості. Тут показано розгортання Luster у масштабі, де показано кілька мереж між клієнтами та кластером Luster, а також кількість серверів вводу-виводу (об'єднаних у групи, що перешкоджають збоям).



1. 7 з 10 найкращих

Понад 40% із Top100

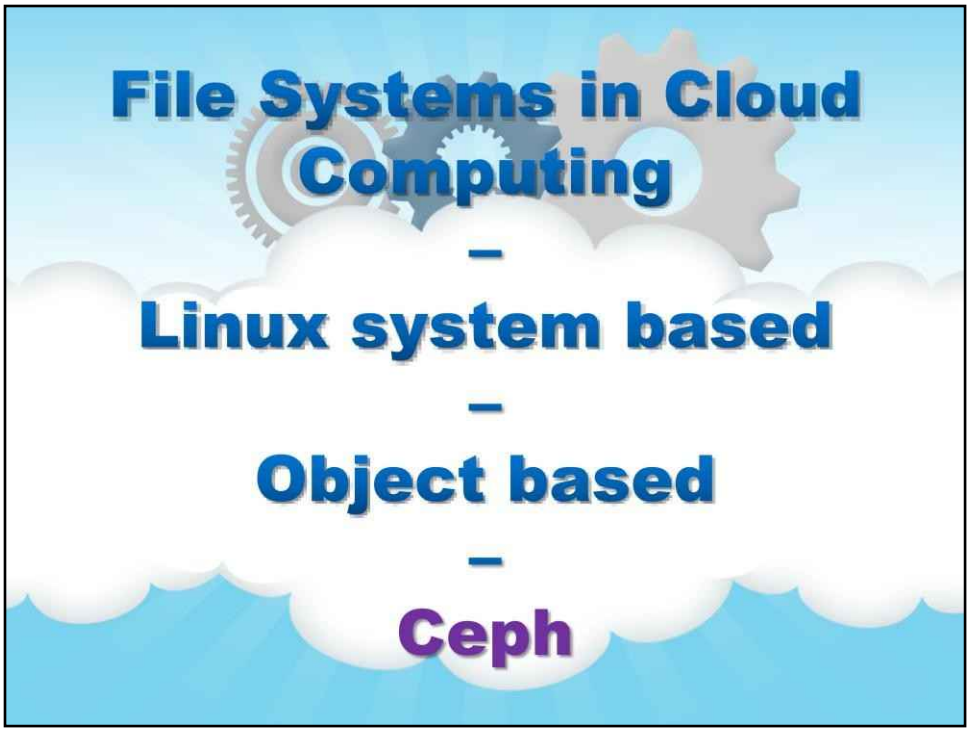
продемонстрували масштабованість

2. 190 ГБ/с введення-виведення

26 000 клієнтів

Системи з понад 1000 вузлами

Спільнота високопродуктивних обчислювальних машин працює над розширенням меж продуктивності та масштабованості, і Luster досяг популярності в цій спільноті. Luster має значний імпульс у спільноті HPC і фактично є основною розподіленою файловою системою для цих систем, як показано на слайді.
Ви можете побачити вражаюче масштабування та високу продуктивність.


File Systems in Cloud Computing
–
Linux system based
–
Object based
–
Ceph

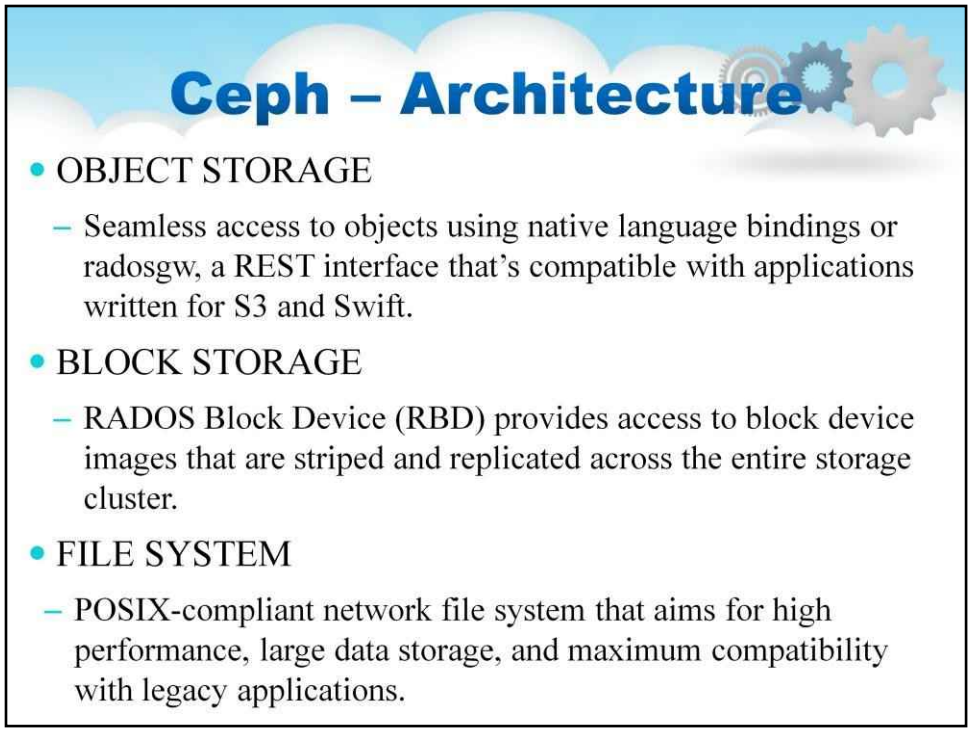
цеф

Ceph (DreamHost) – Overview

Ceph is a distributed file system that offers basic network storage to high-performance clusters for big data.

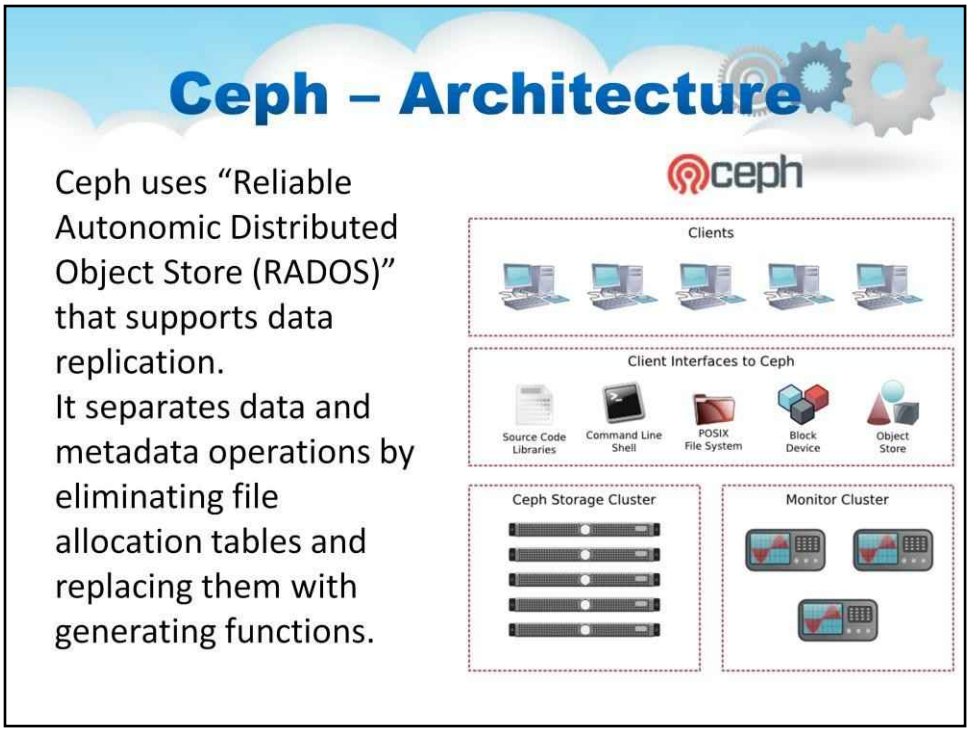
Anywhere there was a need for NFS can now be fulfilled with CFS.

This will remedy all the file lock issues experienced using the Linux native file system.



Сервіси Ceph Object Storage та/або Ceph Block Device — це модулі, які потрібно інтегрувати в хмарні платформи, розгортати файлову систему Ceph або використовувати Ceph з іншою метою. Усі розгортання кластера зберігання Ceph починаються з налаштування кожного вузла Ceph, вашої мережі та кластера зберігання Ceph .
Для кластера зберігання Ceph потрібен принаймні один монітор Ceph і принаймні два демони Ceph OSD. Сервер метаданих Ceph необхідний під час роботи клієнтів файлової системи Ceph.

Ceph зберігає дані клієнта як об'єкти в межах пулів зберігання. Використовуючи алгоритм CRUSH, Ceph обчислює, яка група розміщення має містити об'єкт, і далі обчислює, який Ceph OSD Daemon має зберігати групу розміщення. Алгоритм CRUSH дає змогу кластеру зберігання даних Ceph динамічно масштабуватися, балансувати та відновлюватися.



1. Призначений для об'єднання серверів зберігання об'єктів, блоків і файлів з єдиного розподіленого комп'ютерного кластера

2. Ceph зіставляє імена файлів і каталоги в кластері RADOS

Блокове сховище Ceph можна безпосередньо підключити до віртуальної машини та забезпечує автоматичну реплікацію даних.

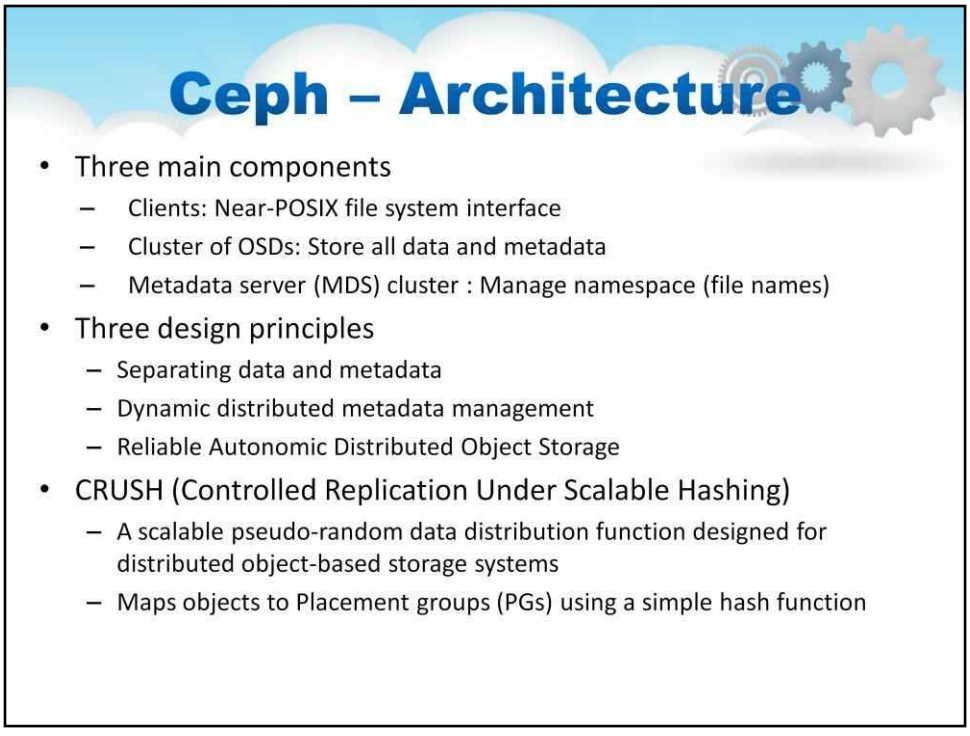
3. Ceph використовує високоадаптивний розподілений кластер метаданих для покращення масштабованості

Його придбала компанія Red Hat у 2014 році

цефє прикладом повністю розподіленої архітектури зберігання (без централізованого керування) і файлової системи, призначеної для інтеграції серверів зберігання об’єктів, блоків і файлів з єдиного розподіленого комп’ютерного кластера.

Розподілене сховище об’єктів Ceph побудовано навколо надійного автономного сховища розподілених об’єктів (RADOS), яке підтримує реплікацію даних. Блокове сховище Ceph можна безпосередньо підключити до віртуальної машини та забезпечує автоматичну реплікацію даних у кластері сховищ.

Файлова система Ceph працює поверх об’єктного або блокового сховища та відображає імена файлів і каталоги в кластері RADOS.



Цеф складається з трьох компонентів

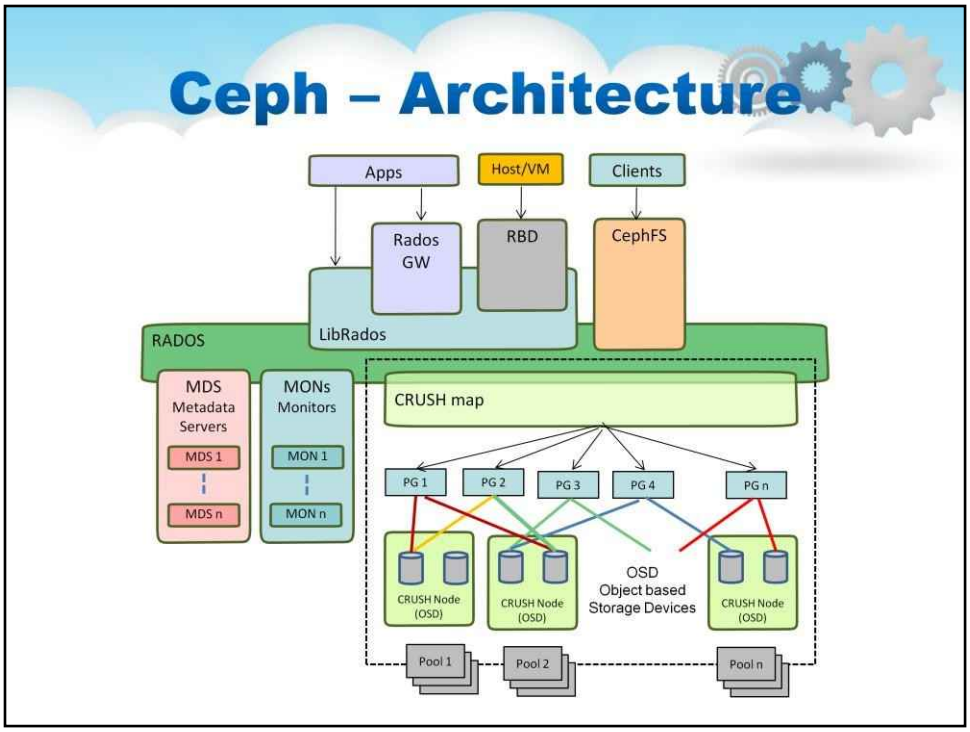
Клієнти: інтерфейс файлової системи Near-POSIX

Кластер OSD: зберігає всі дані та метадані, використовує функцію CRUSH для призначення об’єктів на запам’ятовуючі пристрої

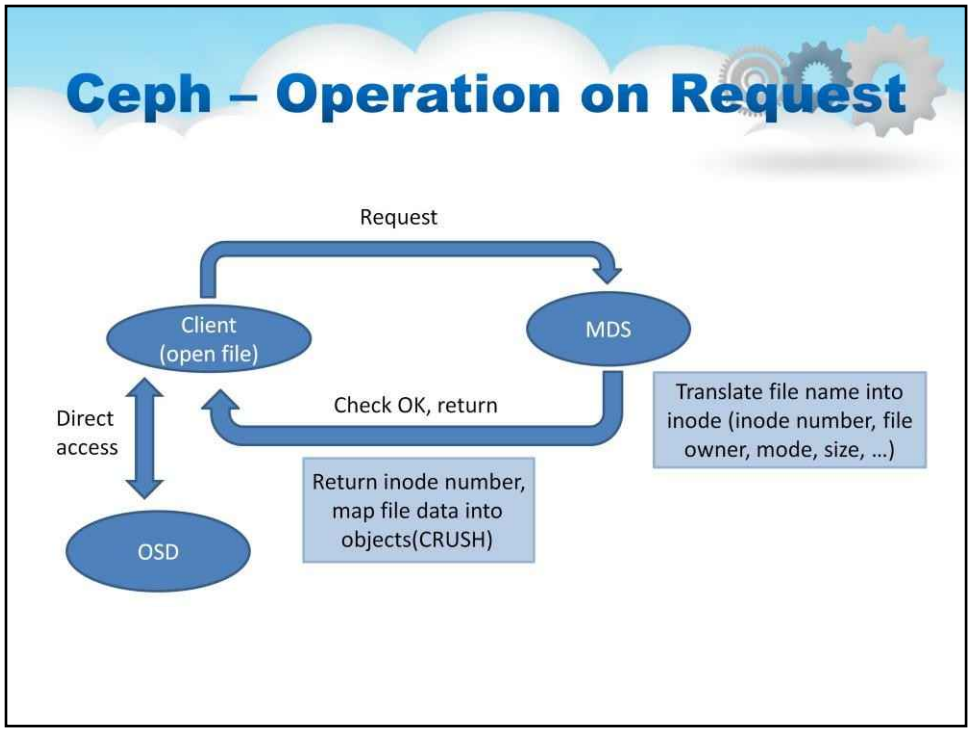
Кластер сервера метаданих (MDS): Керування простором імен (імена файлів) Він розроблений для високої доступності та масштабованості за допомогою ключових шаблонів проектування:

Розділення даних і метаданих Динамічне керування розподіленими метаданими Надійне автономне розподілене сховище об’єктів

PG призначаються OSD за допомогою CRUSH



Ceph розділяє операції з даними та метаданими Запит даних/ файлу включає запит до MDS для отримання розташування компонентів/вузлів файлу та метаданих



Синхронізація клієнта

- Якщо кілька клієнтів (читачів і записувачів) використовують один і той самий файл, MDS відкликає всі попередні запити на читання та запис, доки їх не підтвердить OSD
- Примусова синхронізація клієнтів

Ceph використовує ефективну модель синхронізації клієнта.

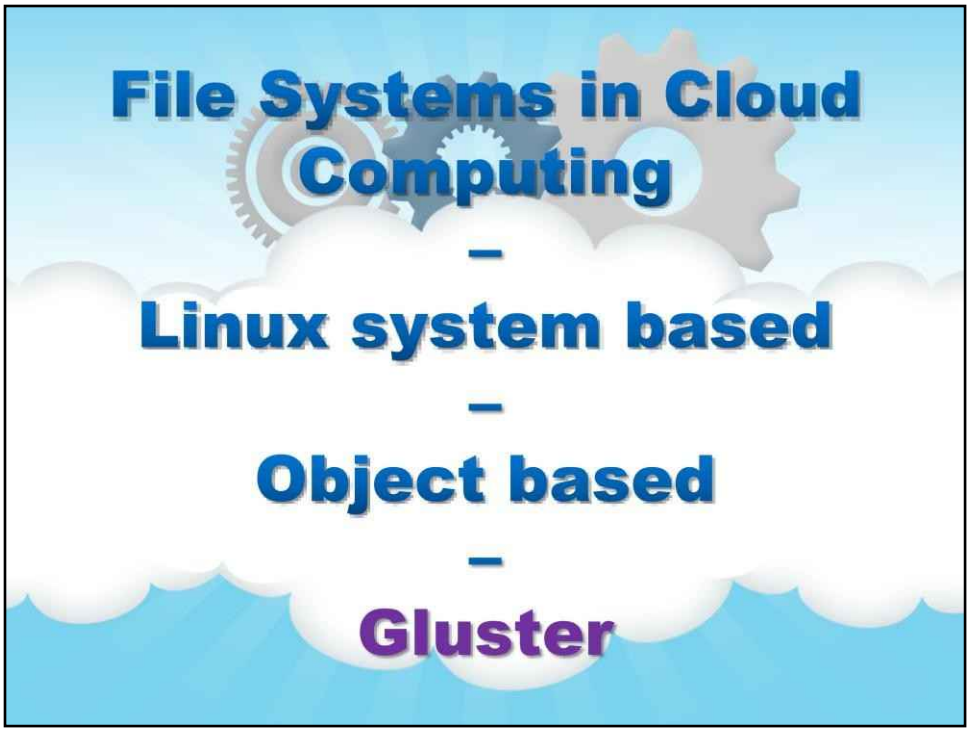
Клієнт робить запит до сервера метаданих, який перетворює ім’я файлу в inode (номер inode, власник файлу, режим, розмір, ...)

Потім починає працювати модуль CRUSH (контрольована реплікація під масштабованим хешуванням).

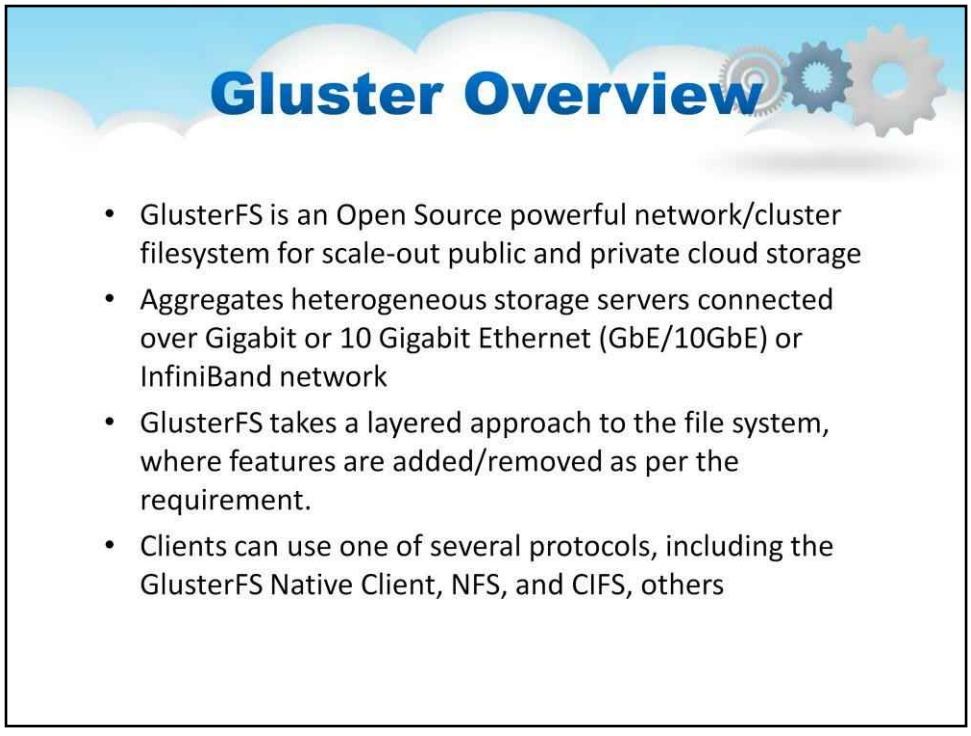
CRUSH — це масштабована функція розподілу псевдовипадкових даних, розроблена для розподілених об’єктних систем зберігання

Зіставляє об’єкти на групи розміщення (PG) за допомогою простої хеш-функції. Вона повертає номер inode, зіставляє дані файлу з об’єктами.

Потім клієнт отримує доступ до пристрою зберігання об’єктів, як видно на ілюстрації.


File Systems in Cloud Computing
–
Linux system based
–
Object based
–
Gluster

Глюстер



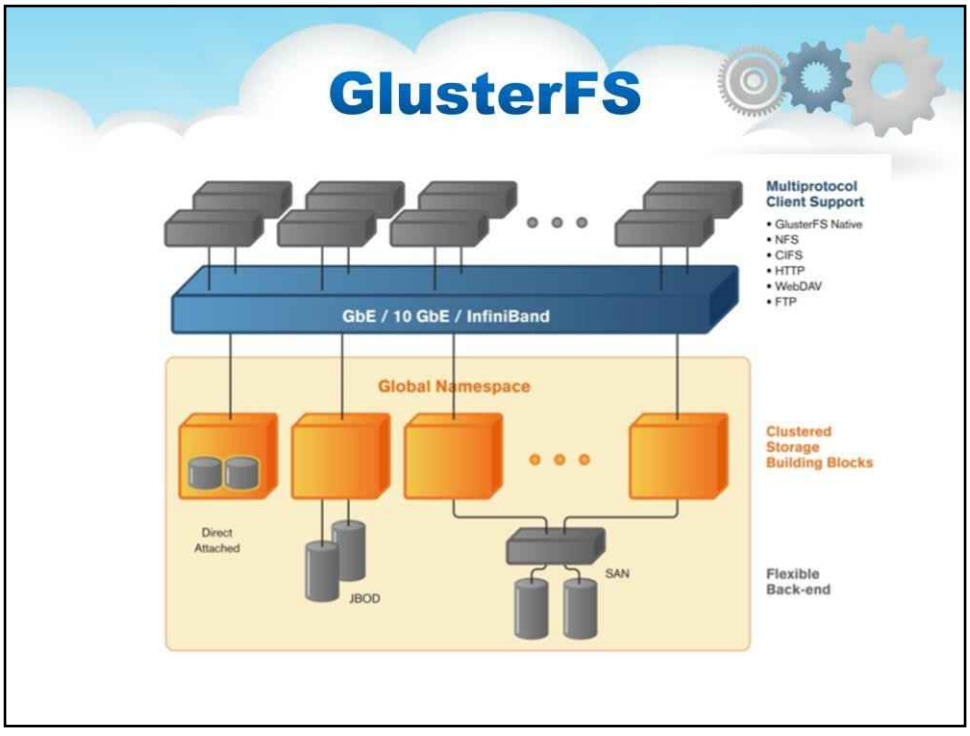
1. Надає прості функції та залишає всі функції керування файлами клієнтам

2. Може створити глобальний простір імен із набору блоків кластерного сховища, які включають пряме підключене сховище, JBOD (просто купу дисків) і SANfabrics 3. Написано в просторі користувача та використовує FUSE для підключення до VFS (віртуальна файлова система).) шар

Gluster використовує існуючі дискові файлові системи, такі як ext3, ext4, xfs тощо, для зберігання даних Масштабування до петабайт пам’яті, яка доступна в одній точці монтування Придбано Red Hat восени 2011 року

Система зберігання та файлів Gluster — це платформа з відкритим кодом для масштабованого публічного та приватного хмарного сховища. Подібно до Lustre, назва Gluster походить від двох слів «GNU» і «cluster». Gluster агрегує гетерогенний сервер зберігання, підключений через мережу Ethernet або Infiniband.

Файлова система Gluster забезпечує просту функціональність і залишає всі функції керування файлами клієнтам.



- GlusterFS — це розподілена файлова система для звичайного обладнання

- Кожен сервер і приєднане товарне сховище (налаштоване як DAS, JBOD або SAN) вважаються вузлом.

- Ємність масштабується шляхом додавання додаткових вузлів або додаткового сховища до кожного вузла.

- Продуктивність підвищується завдяки розгортанню сховища на більшій кількості вузлів.

- Висока доступність досягається шляхом багатосторонньої реплікації даних між вузлами.

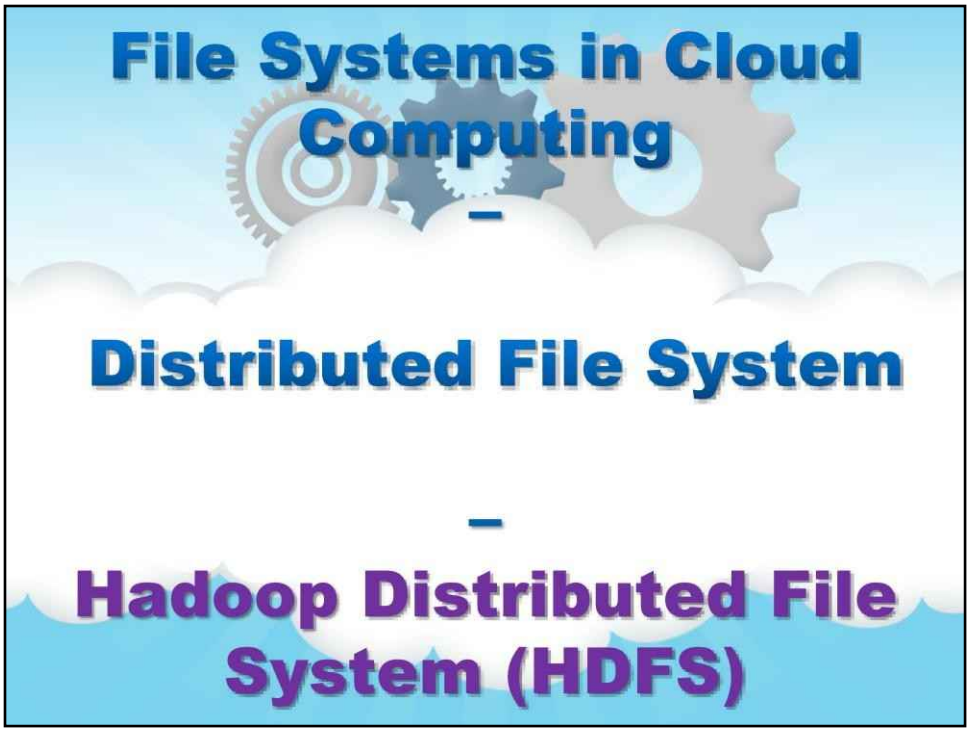
Gluster — це масштабована мережева файлова система зберігання. Він знайшов застосування, включаючи хмарні обчислення, потокові медіа-сервіси та мережі доставки контенту. GlusterFS спочатку була розроблена компанією Gluster, Inc., потім компанією Red Hat, Inc., після придбання Gluster у 2011 році.

Подумайте про Gluster як про альтернативу Swift і Cinder у OpenStack.

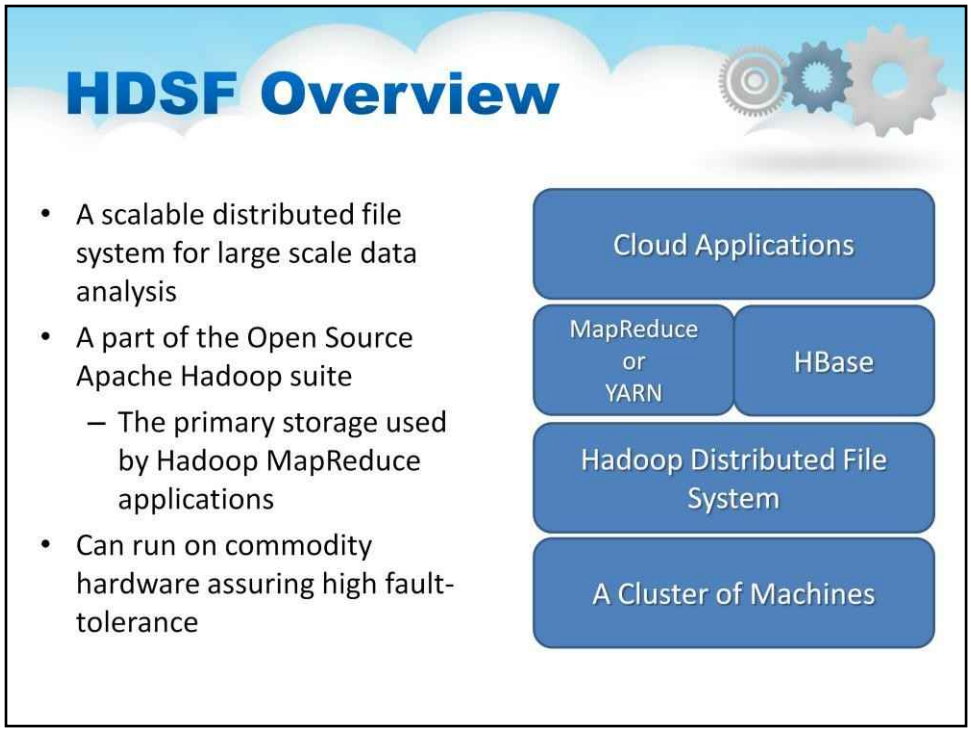
GlusterFS об’єднує різні сервери зберігання даних через мережу Ethernet або Infiniband в одну велику паралельну мережеву файлову систему.

Gluster зберігає дані у вигляді файлів і папок і використовує маркери для ідентифікації розташування файлу в кластері. Маркери, які зберігаються як розширені атрибути файлу, самі по собі розподіляються між каталогами, таким чином покращуючи балансування навантаження, уникаючи потреби у спеціальному сервері метаданих. Коли клієнт отримує доступ до файлу, Gluster перетворює запитане ім’я файлу в маркер і отримує прямий доступ до файлів.

Файлова система Gluster дає змогу налаштувати кластер для реплікації файлів на різних пристроях зберігання, що забезпечує високу доступність файлів і даних у вашому середовищі зберігання.


File Systems in Cloud Computing
–
Distributed File System
–
Hadoop Distributed File System (HDFS)

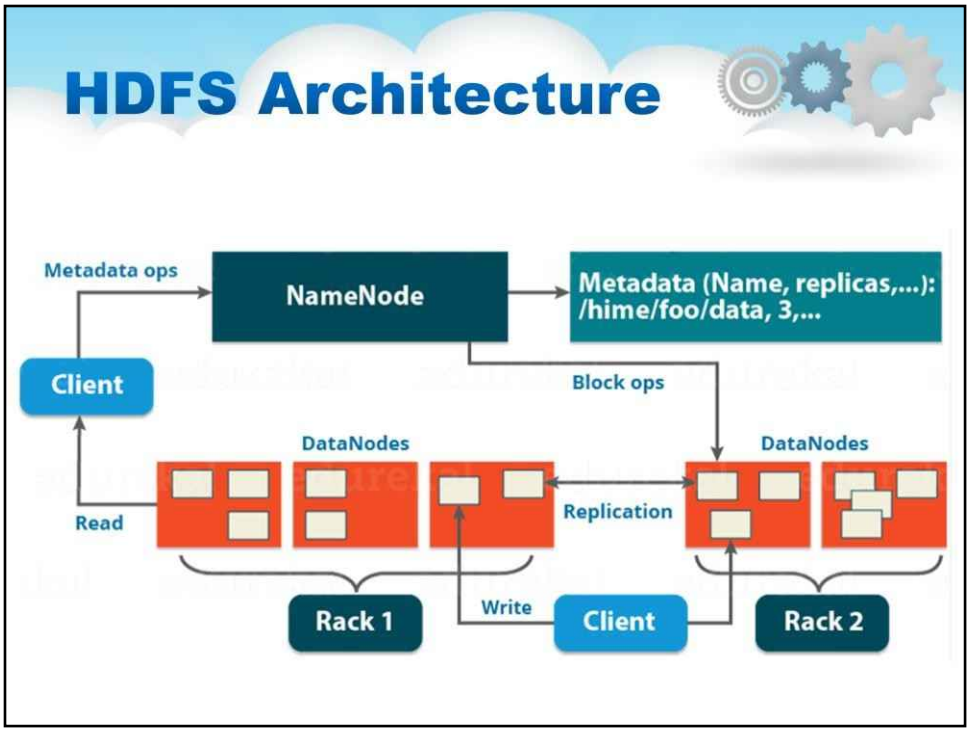
HDFS



Розподілена файлова система Hadoop (HDFS) — це зовсім інший вид «файлової системи», оптимізованої для певного класу програм, як-от Map Reduce і подібні системи «великих даних», як-от бази даних «без SQL».

Це масштабована розподілена файлова система для великомасштабного аналізу даних. Частина набору Apache Hadoop з відкритим кодом

Основне сховище, яке використовується програмами Hadoop MapReduce Може працювати на звичайному обладнанні, що забезпечує високу відмовостійкість

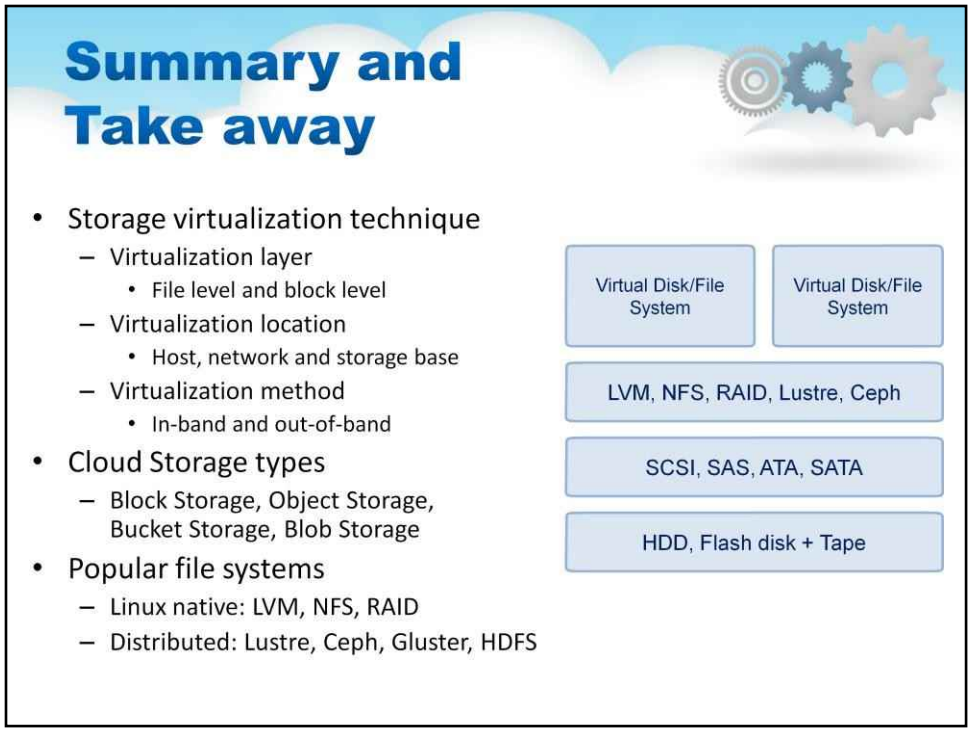


Вміст файлу розбивається на блоки (за замовчуванням 128 МБ, 3 репліки).

NameNode підтримує дерево простору імен і відображення блоків файлів у DataNodes.

Файли та каталоги представлені на NameNode за допомогою inodes (дозволи, час модифікації та доступу, простір імен і квоти дискового простору). Простір імен — це ієрархія файлів і каталогів.

Кластер HDFS складається з одного головного вузла/сервера, який запускає NameNode і кількох DataNodes, зазвичай по одному на фізичний вузол у кластері Hadoop. Дані користувача зберігаються у файлах, зовні вони доступні через простір імен, яким керує NameNode. Щоб отримати доступ до файлу, клієнт-користувач повинен запитати розташування файлу або метадані від NameNode, а після цього він може надіслати запит на читання або запис безпосередньо до DataNode. DataNodes створюють блоки даних і виконують реплікацію на основі інструкцій з NameNode.



У цій лекції досліджено різні способи віртуалізації та реалізації сховища для великих розподілених систем, включаючи хмару.

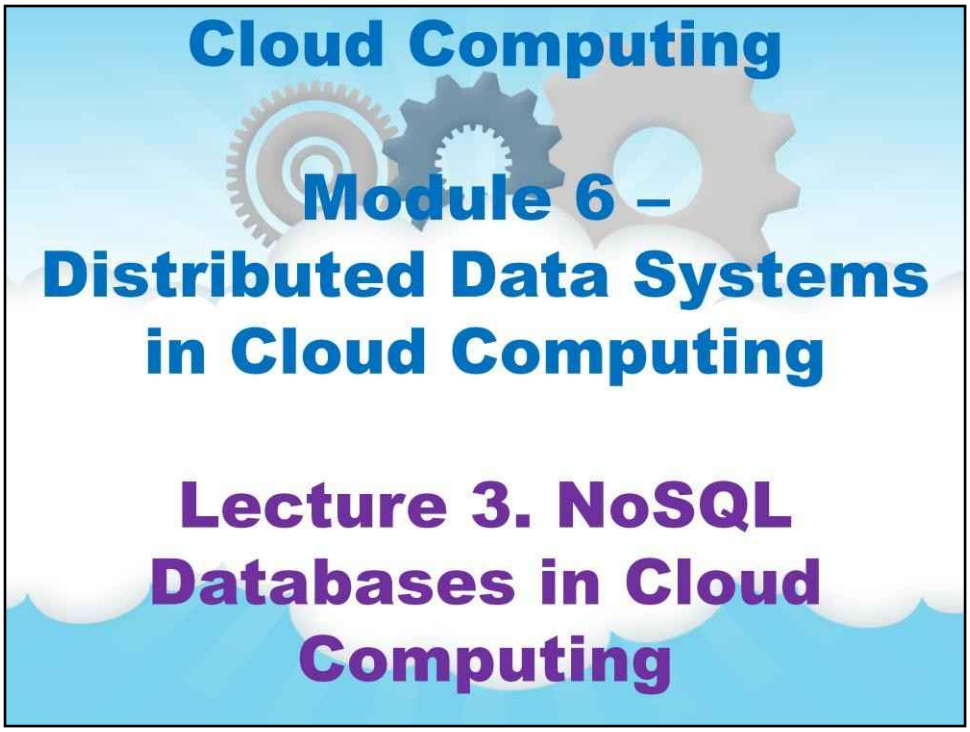
Ми дослідили примітив зберігання, який був віртуалізований, і побачили, що деякі системи зосереджені на віртуалізації файлів, а деякі системи зосереджені на віртуалізації блоків. Ми побачили, що функція віртуалізації може працювати в різних місцях архітектури. Він може працювати на хості, у мережі або повністю назад, де знаходяться диски.

Ми побачили, що віртуалізацію можна розмістити «в смузі» операцій зберігання, а для масштабування зазвичай розміщують «поза смугою».

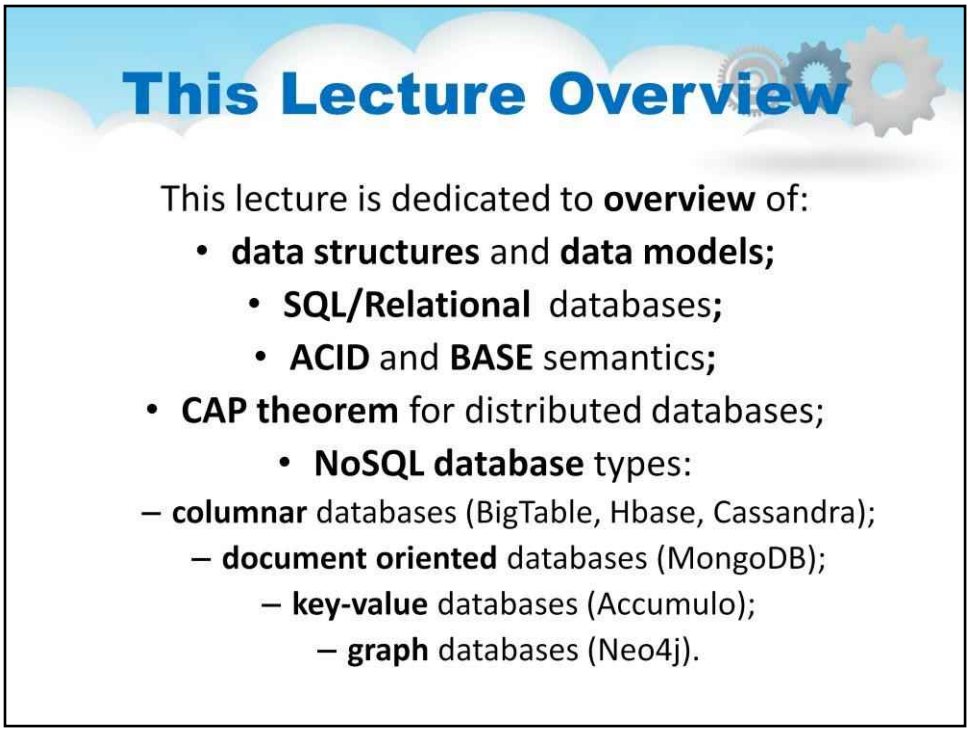
Існує багато типів примітивів зберігання, які після завершення віртуалізації стають доступними для програм. Об’єкти (бакети, блоки), блоки або файлові системи

Існує багато способів розшарування файлової системи за допомогою віртуалізації та реплікації в кластері. Можливості можуть бути близькі до операційної системи, наприклад LVM, або через мережу, як NFS.

Нарешті, ми детально розглянули декілька нових файлових систем, оптимізованих для керування гетерогенними фермами хмарних сховищ.



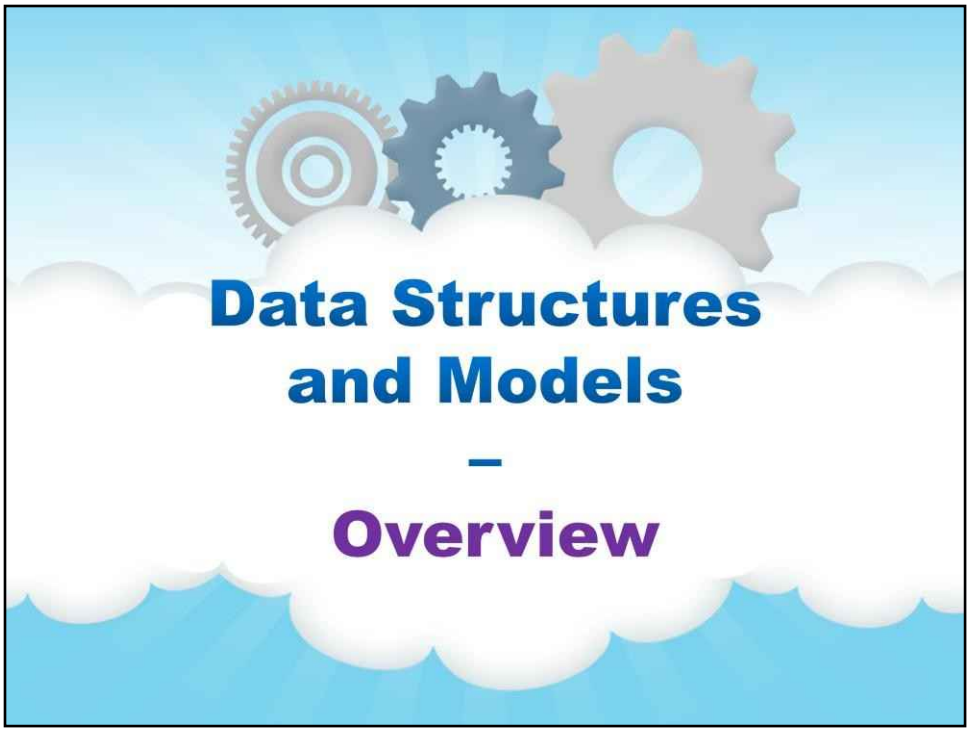
У цій лекції 3 йдеться про бази даних NoSQL у хмарних обчисленнях.



Лекція 3. Бази даних NoSQL в Хмарні обчислення

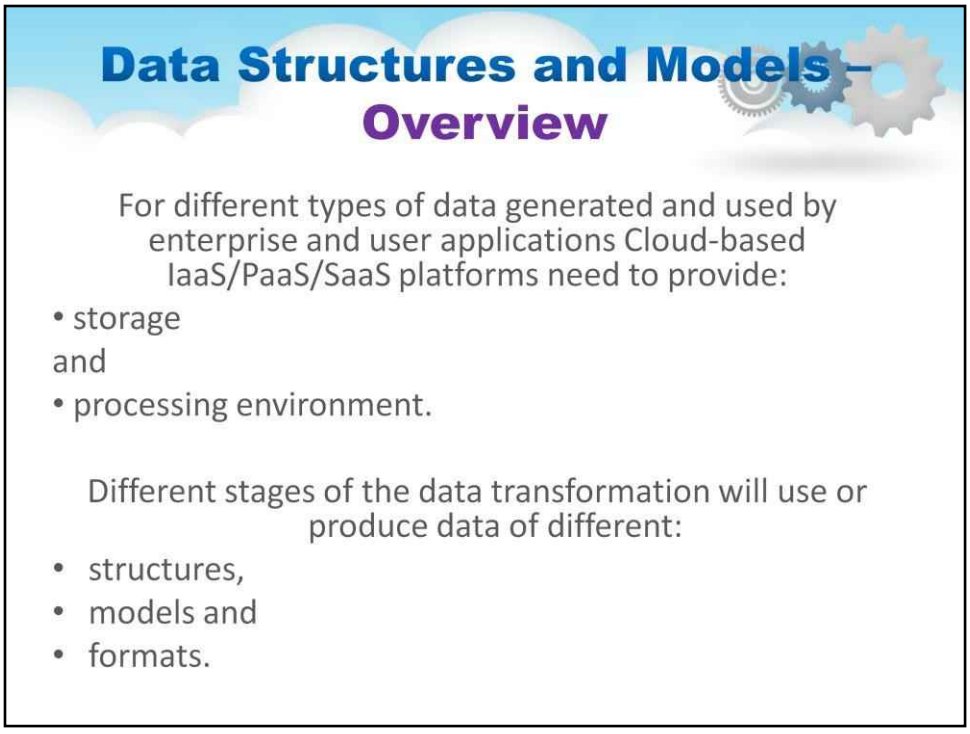
Ця лекція присвячена:

- **структури даних**і**моделі даних;**
- **SQL/реляційний**бази даних**;**
- **КИСЛОТА**і**БАЗА**семантика**;**
- **CAP теорема**для розподілених баз даних;
- **База даних NoSQL**види:
  - **стовпчастий**бази даних (BigTable, Hbase, Cassandra);
  - **орієнтований на документ**бази даних (MongoDB);
  - **ключ-значення**бази даних (Accumulo);
  - **графік**бази даних (Neo4j).

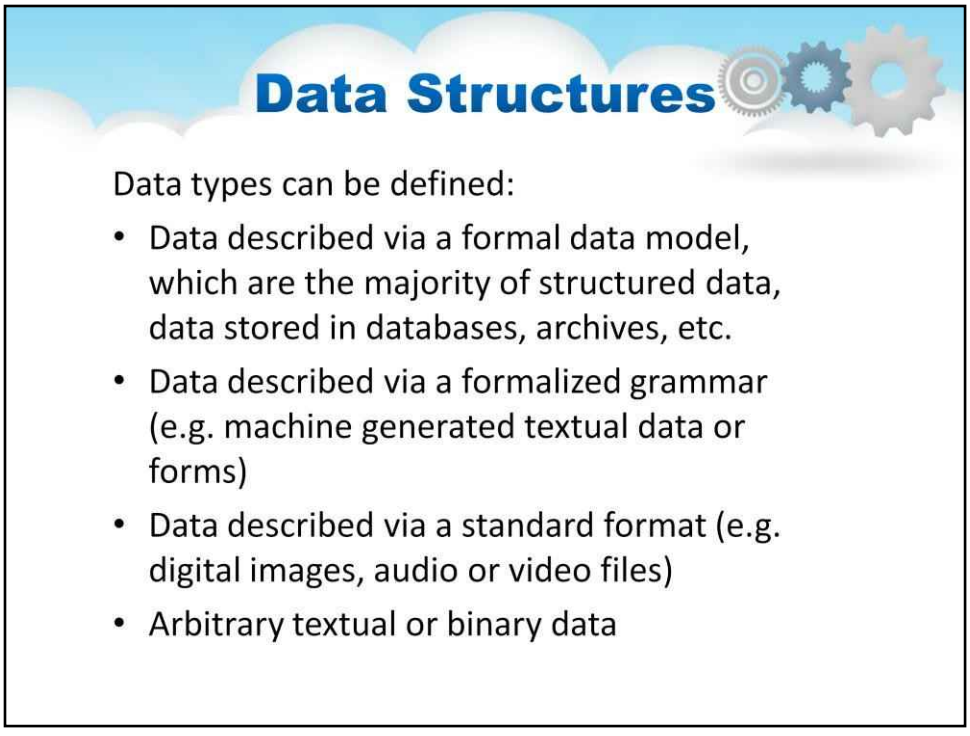


Огляд

Почнемо з огляду структур і моделей даних...

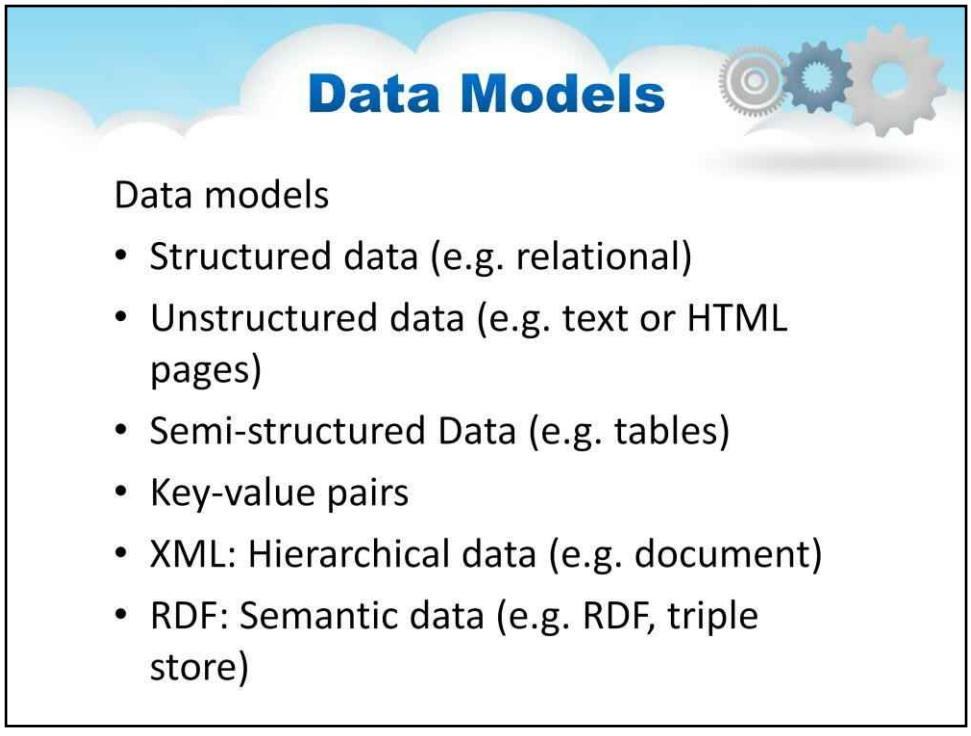


Системи та програми великих даних використовуватимуть та/або створюватимуть різні типи даних, які визначаються їх походженням або цільовим використанням. У свою чергу, різні етапи перетворення даних також використовуватимуть або вироблятимуть дані різних структур, моделей і форматів.



Можна визначити такі типи даних:

- дані, описані за допомогою формальної моделі даних, якими є більшість структурованих даних, дані, що зберігаються в базах даних, архівах тощо.
- дані, описані через формалізовану граматику (наприклад, машинно згенеровані текстові дані або форми)
- дані, описані в стандартному форматі (багато прикладів цифрових зображень, аудіо- та відеофайлів, а також відформатованих двійкових даних)
- довільні текстові або двійкові дані



Наступні слайди пояснюють приклади різних моделей даних

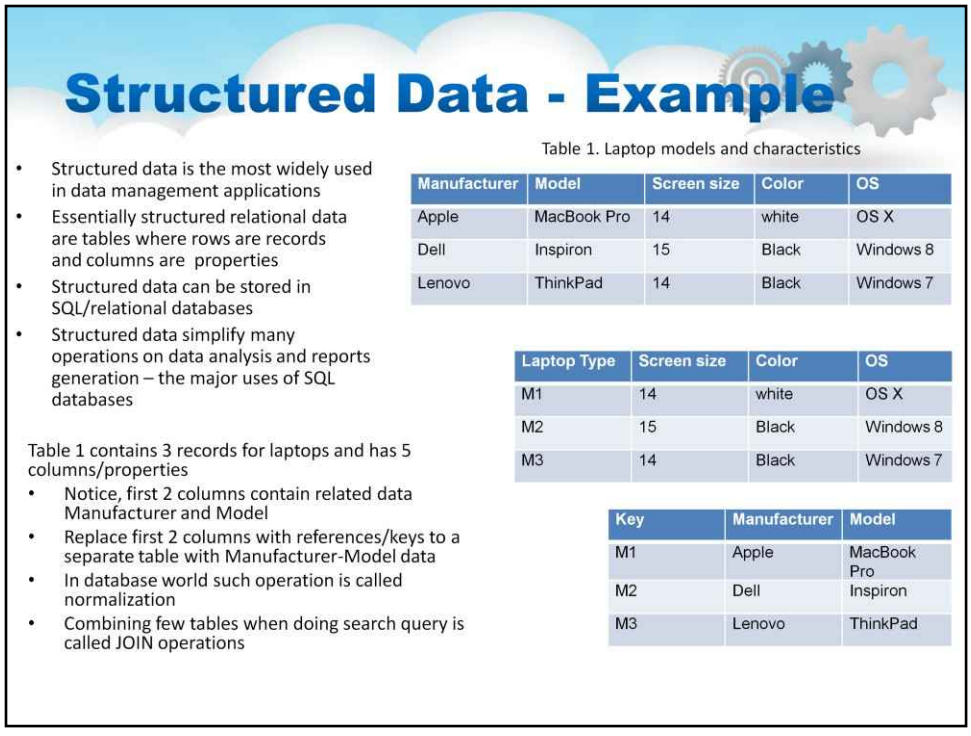
- Структуровані дані (наприклад, реляційні)
- Неструктуровані дані (наприклад, текст або HTML-сторінки)
- Напівструктуровані дані (наприклад, таблиці)
- Пари ключ-значення
- XML: ієрархічні дані (наприклад, документ)
- RDF: семантичні дані (наприклад, RDF, потрійне зберігання)

Далі ми розглянемо кілька прикладів

Тут просто згадати про неструктуровані дані, прикладом яких є текстові дані

Незважаючи на те, що текстові дані широко використовуються всюди, перш за все, для інтелектуального звітування та спілкування між людьми природною мовою, вони майже не мають явної структури. Також можливо, що створений машиною текст має певну формальну граматику, але досить часто текст, створений людьми, міститиме багато граматичних порушень.

Веб-сторінки або HTML-документи являють собою ще один приклад неструктурованих текстових даних. В даний час операції запитів над текстовими даними досить прості, прикладом яких є операції пошукової системи. Ви можете використовувати прості логічні команди керування пошуком у формі пошуку (наприклад, перевірити розширений пошук Google для цього), але в кінці це просто пошук у величезній, але все ще рівній базі даних індексу.



Тепер давайте подивимося детальніше на те, як дані структуровані (чи ні)

Тепер поглянемо на структуровані дані

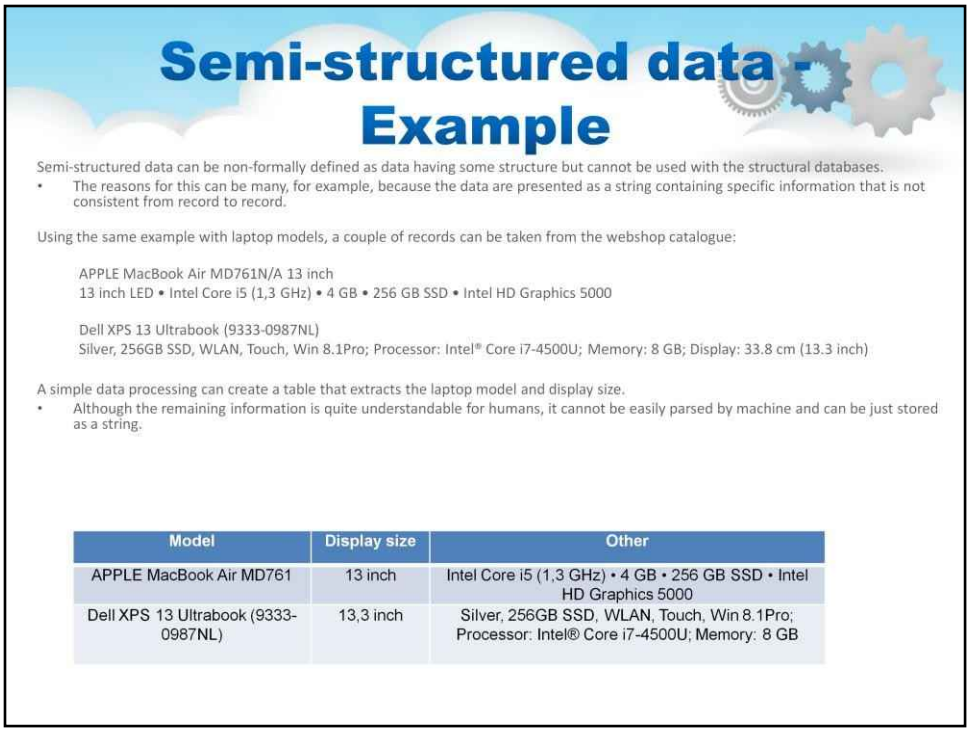
Структуровані дані найбільш широко використовуються в програмах керування даними.

По суті, структуровані реляційні дані – це таблиці, де рядки є записами, а стовпці – властивостями

Структуровані дані можна зберігати в SQL/реляційних базах даних

Структуровані дані спрощують багато операцій з аналізу даних і генерації звітів – основних застосувань баз даних SQL.

Слайд ілюструє структуровані дані. Не те, як дані ідеально вписуються в таблиці. Зверніть увагу, що існує одна таблиця, яка «об’єднує» дві інші таблиці за допомогою ключа. Усе це дуже поширені типи методів у структурованих даних.



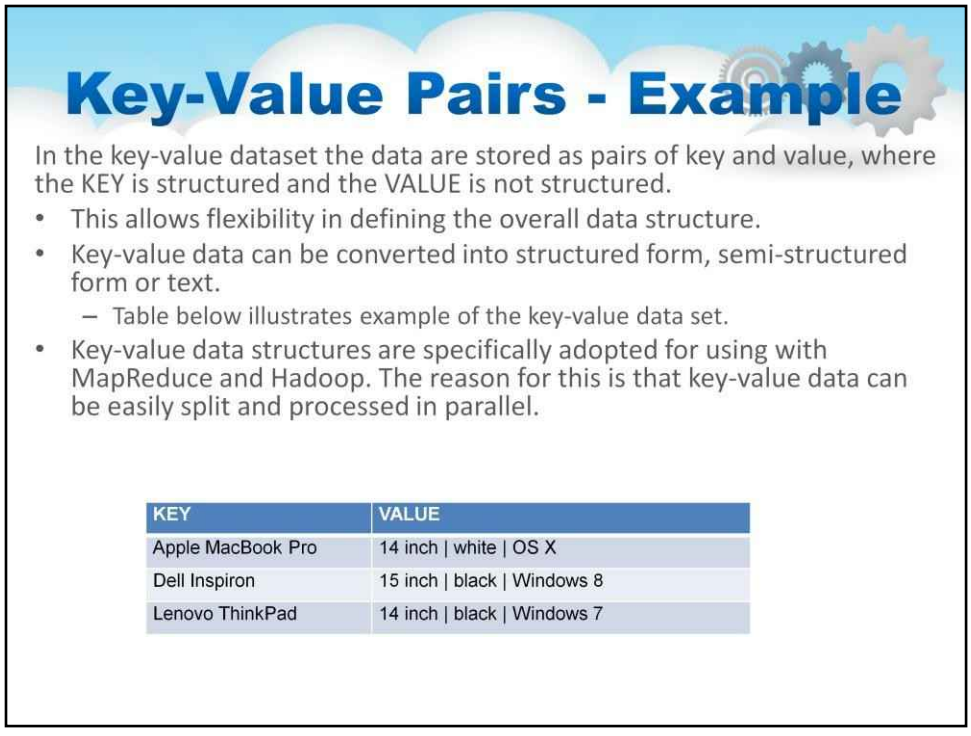
Далі ми розглянемо напівструктуровані дані.

Подумайте про напівструктуровані дані, визначені як дані, що мають певну структуру, але не можуть використовуватися зі структурними базами даних.

Причин для цього може бути багато. На слайді представлено приклад, коли напівструктуроване джерело даних має структуру з двох частин, що є очевидним для будь-якого програмного забезпечення. Перша частина призначена для моделі продукту, за якою йде друга частина, яка призначена для опису. Аспект «двох частин» — це «структура» в цьому прикладі.

«Напівструктурована» частина полягає в тому, що текстовий вміст — модель і опис — є текстом «звичайної мови» — хоча вони ідеально читаються людиною, вони не мають достатньої структури, щоб бути машиночитаними.

Хоча дані легко вписуються в таблицю, вони не такі корисні, як могли б бути.



Часто можна почути про те, що дані структуровані в «пари ключ-значення». Ми це визначимо.

У наборі даних ключ-значення дані зберігаються як пари ключа та значення, де КЛЮЧ є структурованим, а ЗНАЧЕННЯ не структурованим.

Це забезпечує гнучкість у визначенні загальної структури даних.

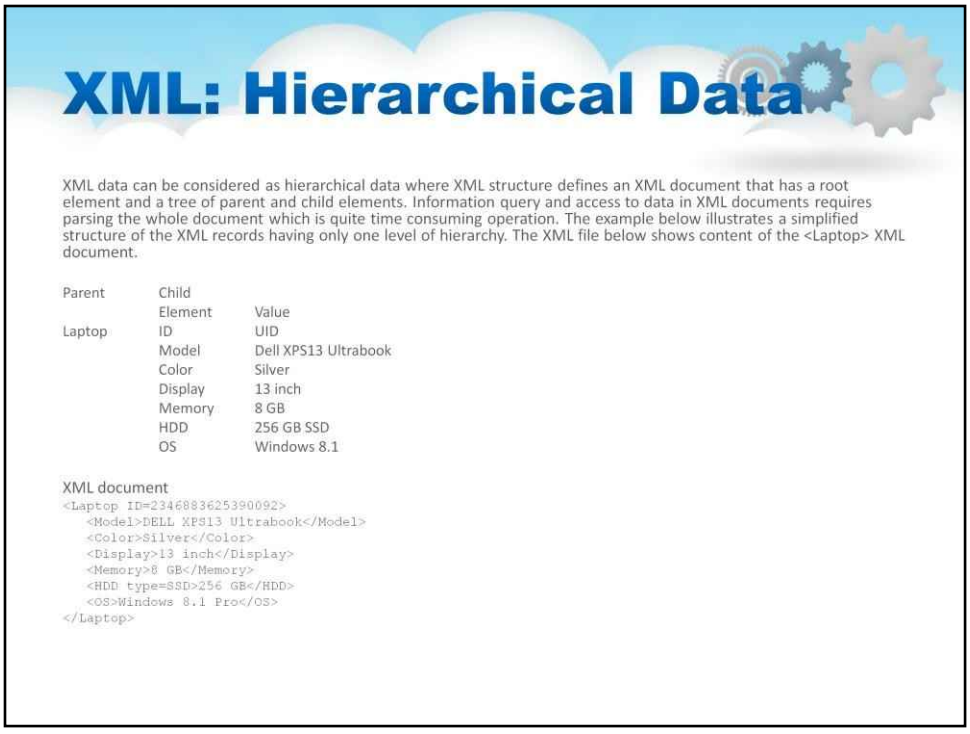
Дані ключ-значення можна перетворити в структуровану форму, напівструктуровану форму або текст.

Таблиця на слайді ілюструє приклад набору даних "ключ-значення".

Можна побачити, що це покращення порівняно з напівструктурованим набором даних, оскільки перша частина даних була поміщена в жорсткий формат, де її можна використовувати як «ключ», який зіставляється з відповідним набором дані, які можуть мати невелику або велику структуру. Справа в тому, що ключ дозволяє легко впорядковувати ці дані, щоб розділити їх і працювати над визначенням набору даних, з яким поєднані ключі.

Структури даних ключ-значення спеціально адаптовані для використання з MapReduce і Hadoop. Причина цього полягає в тому, що дані про ключ-значення можна легко розділити та обробляти паралельно.

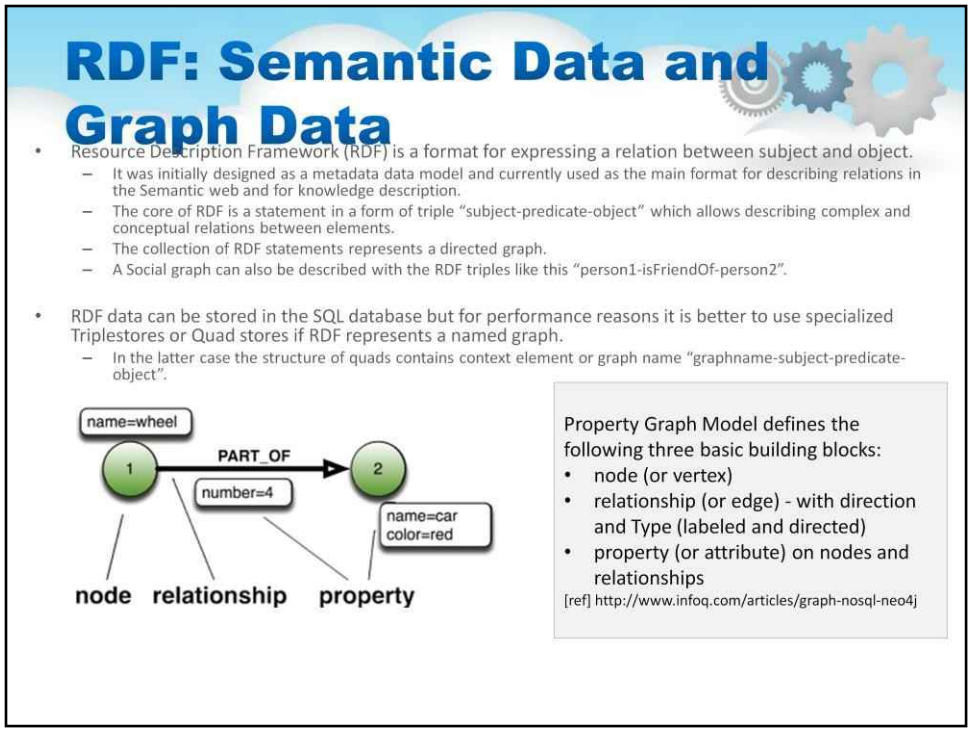
Будь ласка, перегляньте більш детальну лекцію про MapReduce і Hadoop. Наразі має бути логічним і очевидним, що структура ключ-значення призводить до свого роду алгоритму «розділяй і володарюй», один із найпопулярніших з яких називається MapReduce.



XML (розширена мова розмітки) визначає набір правил для кодування документів у форматі, який читається як людиною, так і машиною. Формат XML-структури визначається іншим файлом, який називається XML-схемою. Після того, як схема відома, можна читати або писати XML-документ, який відповідає цій схемі. Схема XML, яка визначає модель даних і на яку посилається в документі, може бути окремим файлом або частиною самого документа в окремому розділі.

Для комп’ютерів XML є дуже некомпактною формою для структурування даних, документи XML додають багато накладних витрат на дані, закодовані за допомогою XML. Якщо помістити двійкові дані в структуру XML – від стиснутої фотографії до об’єкта Java – вони будуть закодовані за допомогою текстових символів і стануть дуже великими та громіздкими. Модулі, які кодують і декодують XML, а також серіалізують/десеріалізують XML для передачі та прийому відповідно, хоча й не є складними, можуть потребувати великого бюджету обчислювальних ресурсів і ресурсів пам’яті.

Для людей, хоча теоретично XML доступний для читання та запису, він не дуже зручний для людини, і рідко взаємодія з XML здійснюється безпосередньо, використання інструментів як шару, що представляє набагато більш зручний інтерфейс, є дуже поширеним явищем.



Більш складна структура даних називається «Семантичні дані». Семантичні дані представлені у форматі під назвою Resource Description Framework (RDF).

RDF — це формат для вираження зв’язку між суб’єктом і об’єктом.

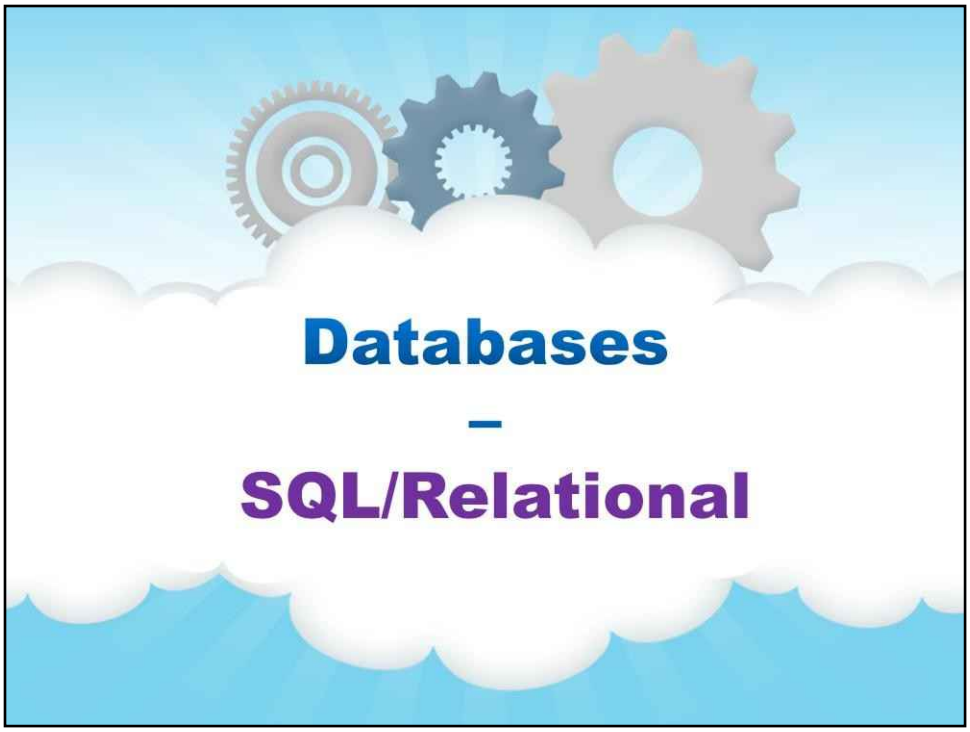
Спочатку він був розроблений як модель даних метаданих і зараз використовується як основний формат для опису відносин у семантичній мережі та для опису знань.

Ядром RDF є висловлювання у формі потрійного «суб’єкт-предикат-об’єкт», яке дозволяє описати складні та концептуальні зв’язки між елементами.

Набір операторів RDF представляє орієнтований граф.

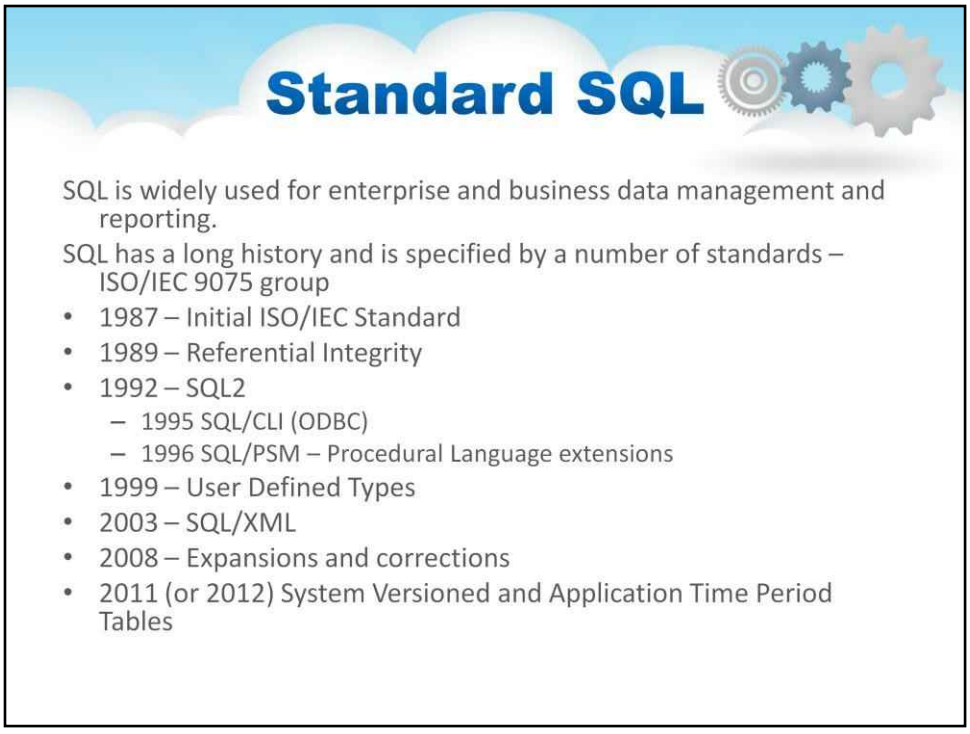
Соціальний граф також можна описати за допомогою трійок RDF, наприклад «person1-isFriendOf-person2».

Ці колекції суб’єкт-предикат-об’єкт і засновані на графах сутності суб’єкт-предикат-об’єкт додають значну інформацію в опис даних. У рамках проекту Semantic Web було проведено багато досліджень щодо того, що інформація, додана до опису даних за допомогою цих зв’язків, дозволяє використовувати інформацію як базу знань, а не просто базу даних. Питання на кшталт «що з цього належить разом» і «яка річ найбільше схожа на іншу». Можна зрозуміти більше, що це за дані, якщо



Бази даних

SQL/реляційний



SQL (вимовляється як "ess-que-el") означає мову структурованих запитів. SQL використовується для зв’язку з базою даних.

Згідно з ANSI (Американський національний інститут стандартів), це стандартна мова для систем управління реляційними базами даних.

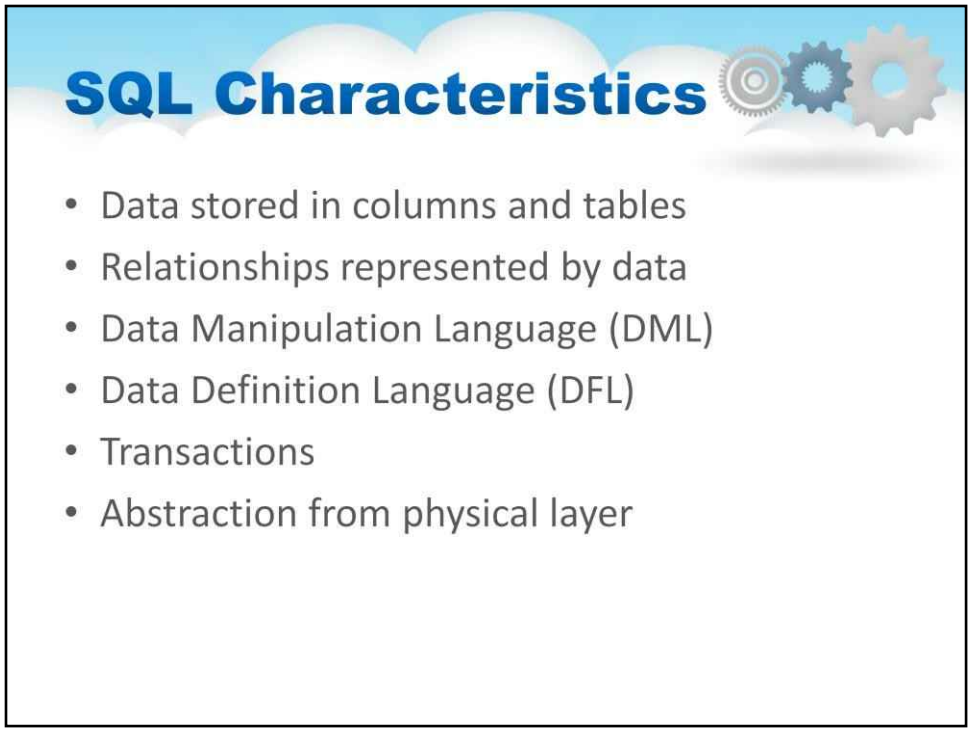
Зрештою, це мова програмування спеціального призначення

SQL за своєю суттю покладається на здатність взаємодіяти з даними реляційним способом. Існує багато команд, таких як JOIN, SELECT, UPDATE, DELETE, INSERT, WHERE, які припускають наявність зв’язків у даних (або базі даних).

SQL широко використовується для керування корпоративними та бізнес-даними та звітності.

SQL має довгу історію і специфікується низкою стандартів

Слайд показує багато віх розвитку SQL.



База даних SQL досить складна.

Мова обробки даних (DML)

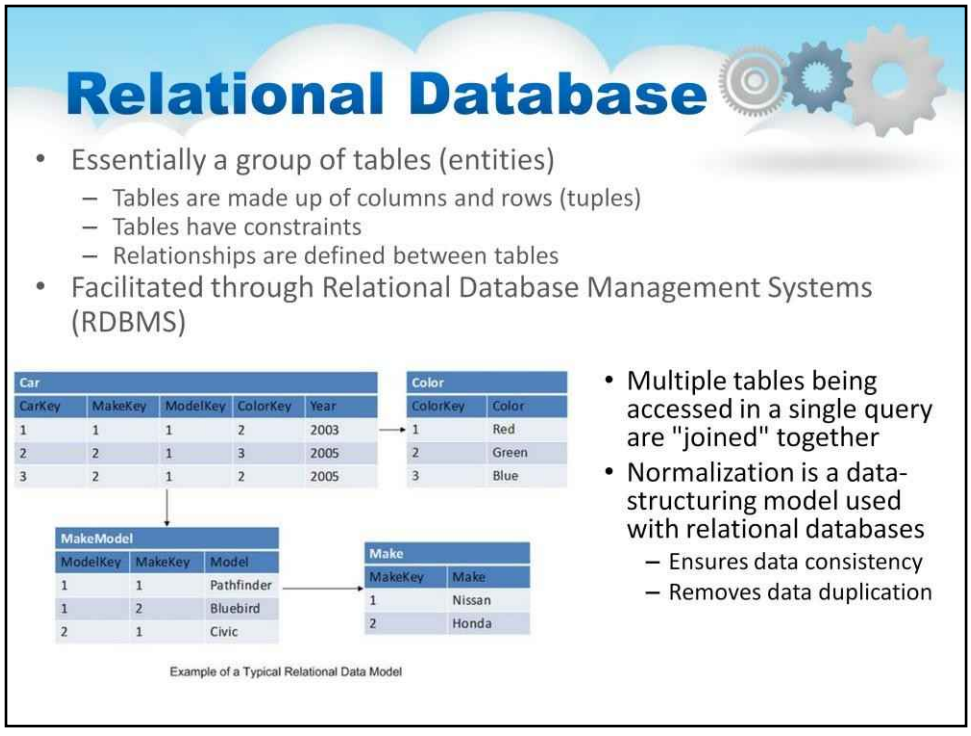
У SQL мова обробки даних містить оператори зміни даних SQL, які змінюють збережені дані, але не схему або об’єкти бази даних.

Дані, якими керують оператори Select, Insert, Update та Delete
Агрегація даних
Складені висловлювання
Функції та процедури Явний
контроль транзакцій

Мова визначення даних (DFL)

Маніпулювання постійними об’єктами бази даних, наприклад, таблицями або збереженими процедурами, за допомогою операторів схеми SQL, а не даних, що зберігаються в них.

Схема, визначена на початку, як: Створити таблицю (Стовпець1 Тип даних1, Стовпець2 Тип даних 2, ...)
Обмеження для визначення та забезпечення дотримання відносин
Первинний ключ
Зовнішній ключ
тощо
Тригери для відповіді на вставлення, оновлення та видалення
збережених модулів



Традиційно базою даних є система, яка програмується за допомогою SQL і підтримує транзакції.

Базовою архітектурою, яка робить це можливим, є реляційна структура даних. Relational підтримує те, що конструкції (наприклад, SELECT, JOIN і WHERE) мають бути реалізовані.

Relational — це дуже структурована група таблиць.

Таблиці (також звані сутностями) складаються зі стовпців і рядків (кортежів)

Таблиці мають обмеження

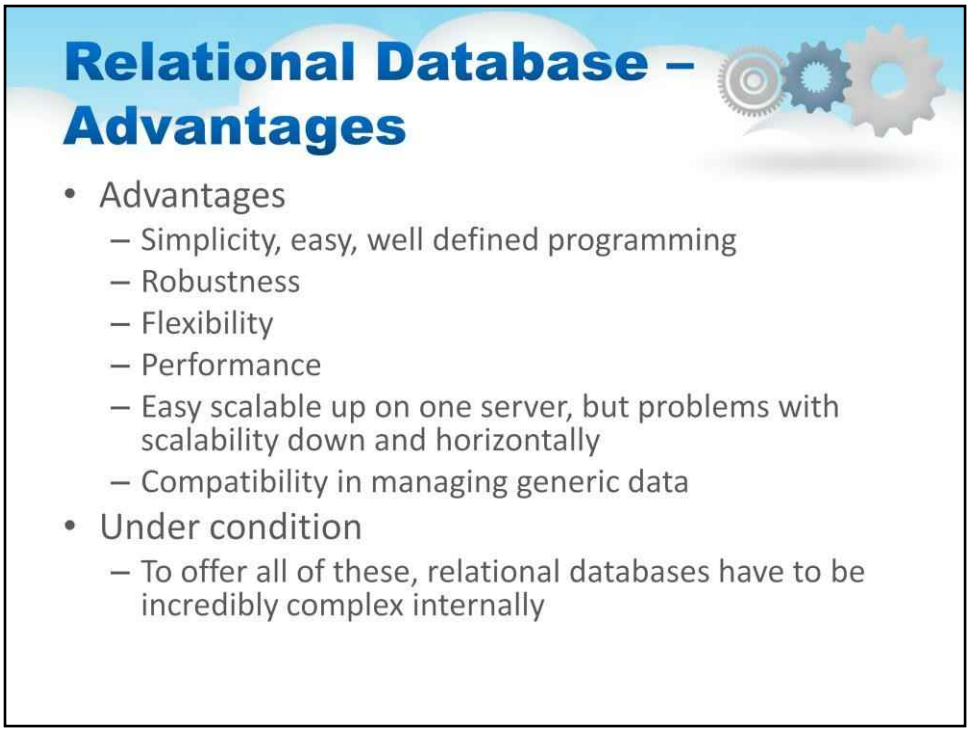
Між таблицями визначаються зв’язки

Це показано на слайді.

Кілька таблиць, доступ до яких здійснюється в одному запиті, «з’єднуються» разом

Нормалізація — це модель структурування даних, яка використовується з реляційними базами даних

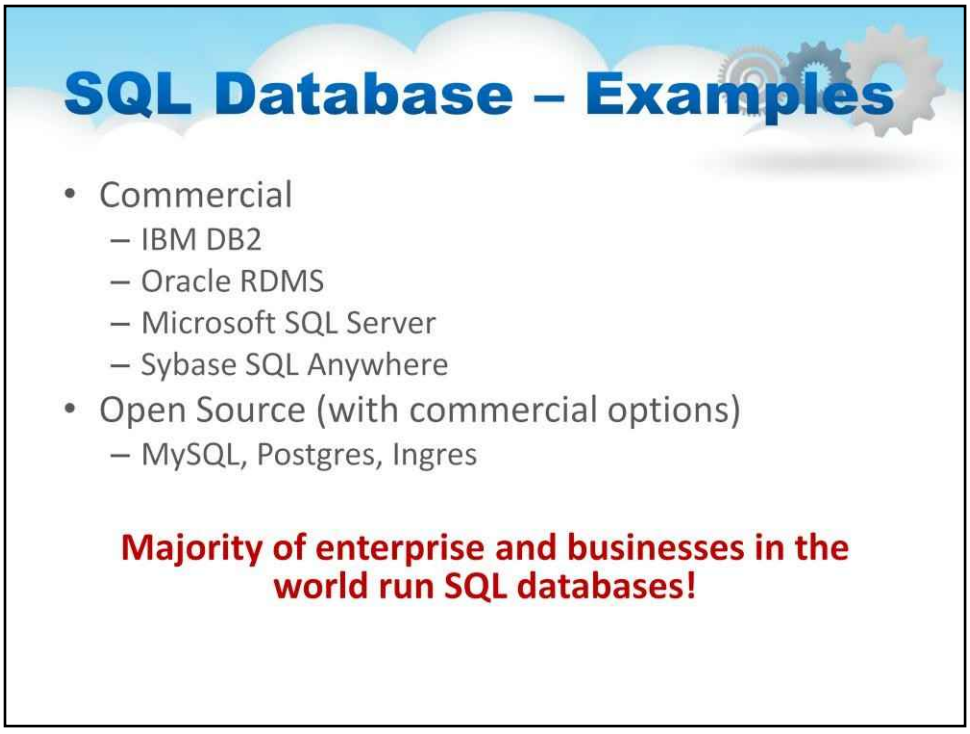
Забезпечує узгодженість даних - усуває дублювання даних



Реляційна база даних має багато переваг:

- Простота, легкість, добре визначене програмування
- Міцність
- Гнучкість
- Продуктивність
- Легке масштабування на одному сервері, але проблеми з масштабуванням вниз і по горизонталі
- Сумісність в управлінні загальними даними

Проте! Щоб запропонувати все це, реляційні бази даних мають бути неймовірно складними внутрішньо



Знайти приклади баз даних SQL неважко

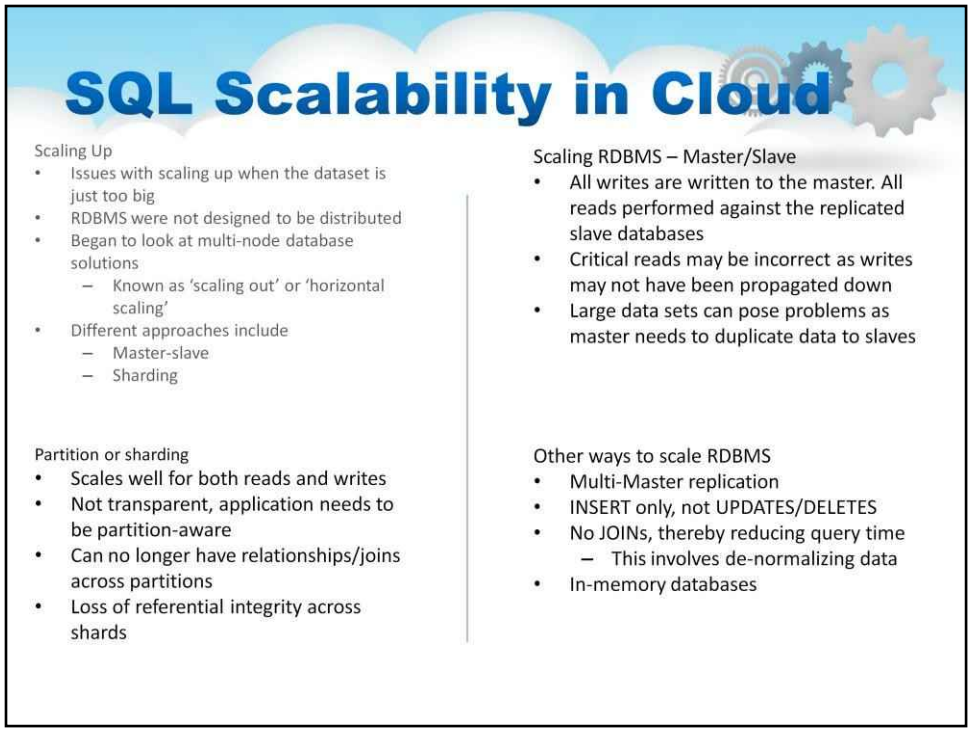
Комерційний

•IBM DB2
•Oracle RDMS
•Microsoft SQL Server
•Sybase SQL Anywhere

Відкритий код (з комерційними опціями)

•MySQL,
•Postgres,
Енгр

Більшість підприємств і компаній у світі використовують бази даних SQL!



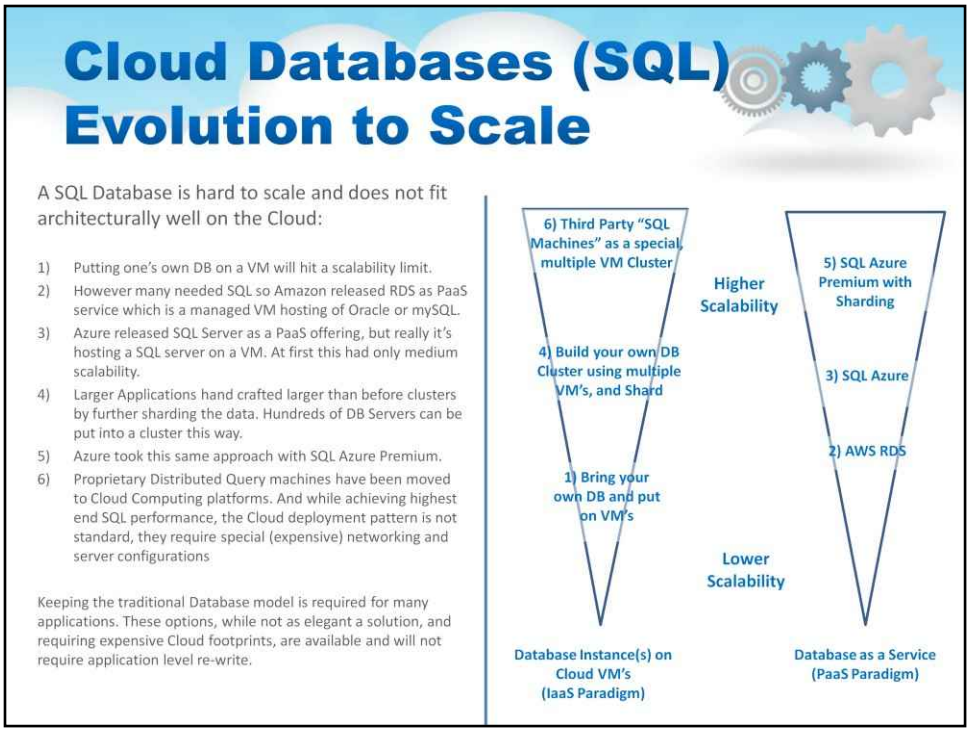
- > Найкращий спосіб забезпечити ACID і розширену модель запиту – мати набір даних на одній машині.

- > Однак існують обмеження щодо збільшення (вертикальне масштабування).

- > Після певного моменту організація визнає, що масштабування (горизонтальне масштабування) дешевше та доцільніше шляхом додавання менших, дешевших (відносно) серверів замість того, щоб інвестувати в один більший сервер.

- > низка різних підходів до масштабування (горизонтальне масштабування).

- > Адміністратори баз даних почали дивитися на головний-підлеглий і шардинг як на стратегію подолання деяких із цих проблем.



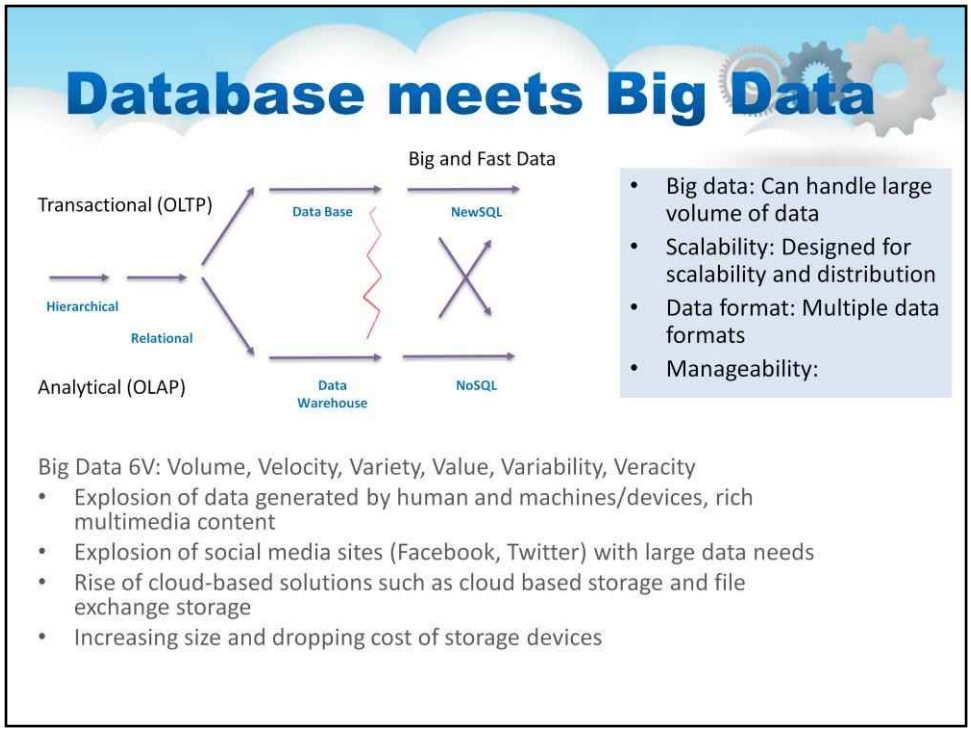
Базу даних SQL важко масштабувати, і архітектура її погано вписується в хмару

Ця діаграма досліджує два різні сценарії. Один екземпляр(и) бази даних на хмарних віртуальних машинах (парадигма IaaS) і один екземпляр бази даних як послуга (парадигма PaaS).

При переході від вимог нижчої масштабованості до вищої масштабованості класичні системи СУБД стикаються зі складними проблемами.

Ця діаграма вказує на те, що саме роблять люди, коли їхні потреби в масштабованості зростають, і все одно намагаються використовувати профіль СУБД

По суті, сфера великих даних була створена, коли системи транзакцій і сховища даних зіткнулися з проблемами з великими даними та середовищем обробки, налаштованим у розподіленій системі, як-от хмара.



Цей слайд ілюструє еволюцію баз даних і пропонує простий спосіб подумати про OLTP і OLAP і про те, що відбувається з базами даних у хмарі

Де значення:

OLTP – онлайнова обробка транзакцій
OLAP – онлайнова аналітична обробка

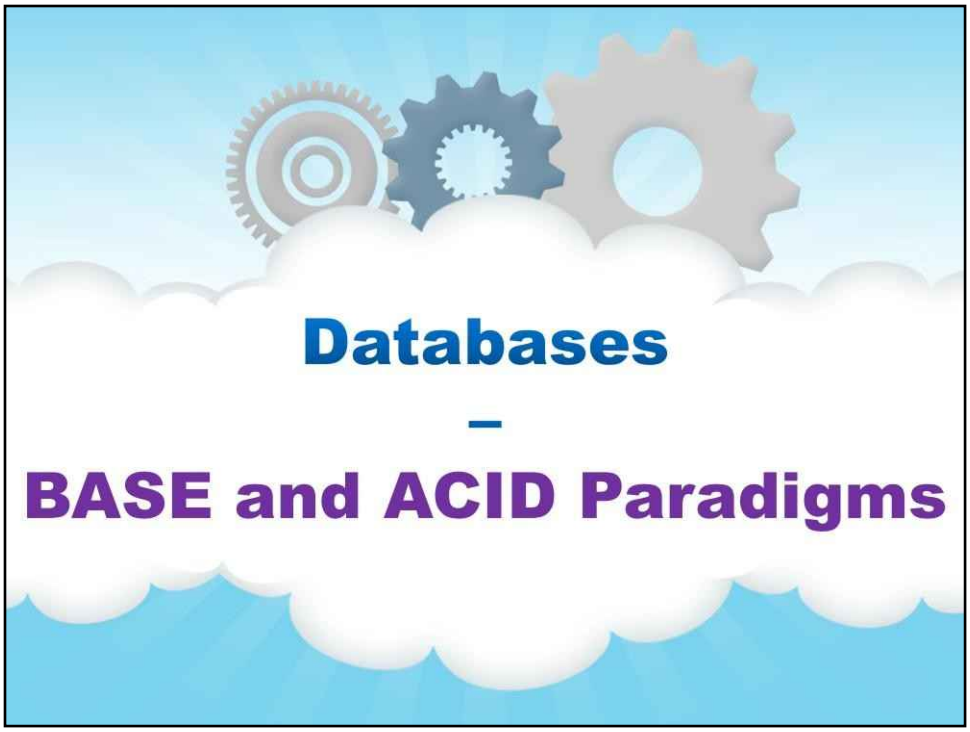
- > Бази даних можна збільшити, розрізаючи (або «розбиваючи») набори даних.

- > Однак люди з найбільшими наборами даних (терабайт/петабайт) почали усвідомлювати, що шардинг накладає пов’язку на їхні проблеми. Більш агресивні лідери думок (Google, Facebook, Twitter) почали досліджувати альтернативні способи зберігання даних. Особливо це стало актуальним у 2008/2009 н.

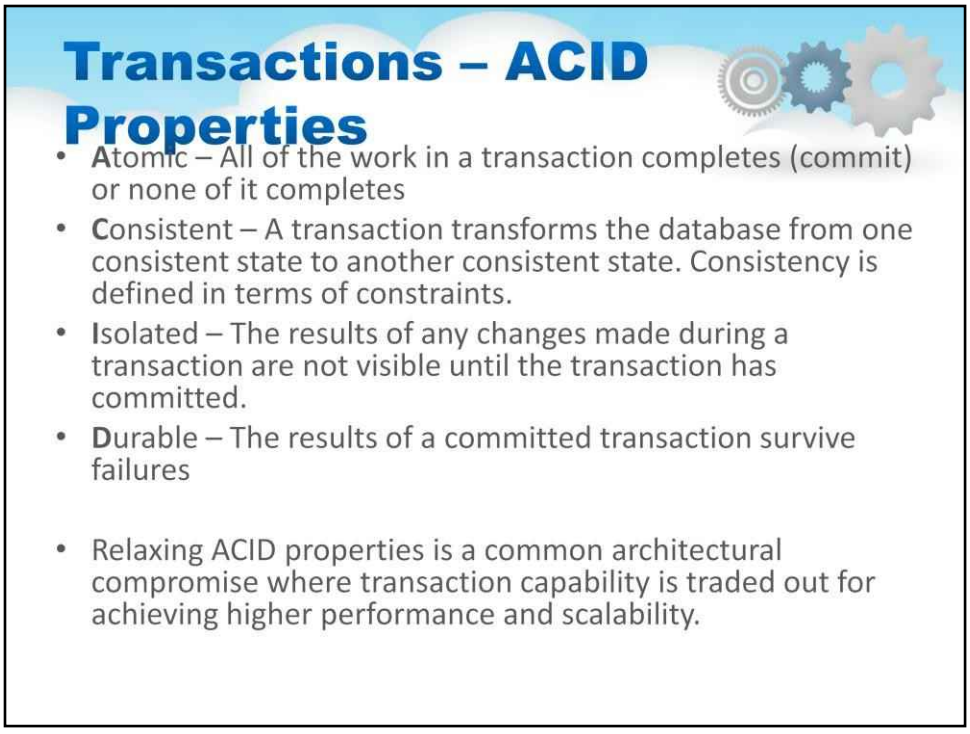
- > Ці набори даних мають високі швидкості читання/запису.

- > З появою Amazon S3 великий поважний постачальник заявив, що, можливо, було б добре розглянути альтернативні рішення для зберігання, крім реляційних.

- > Усі параметри NoSQL, за винятком Amazon S3 (Amazon Dynamo), є рішеннями з відкритим кодом. Це забезпечує недорогу точку входу, щоб «скинути шини».



Парадигми BASE і ACID



Більшість баз даних підтримують поняття транзакції. Сказавши це, багато «баз даних NoSQL» не підтримують транзакції, у кластері NoSQL вони можуть мати модель «можливої узгодженості».

Отже, що означає підтримувати транзакцію?

Трансакції демонструють властивості ACID, де ACID є абревіатурою:

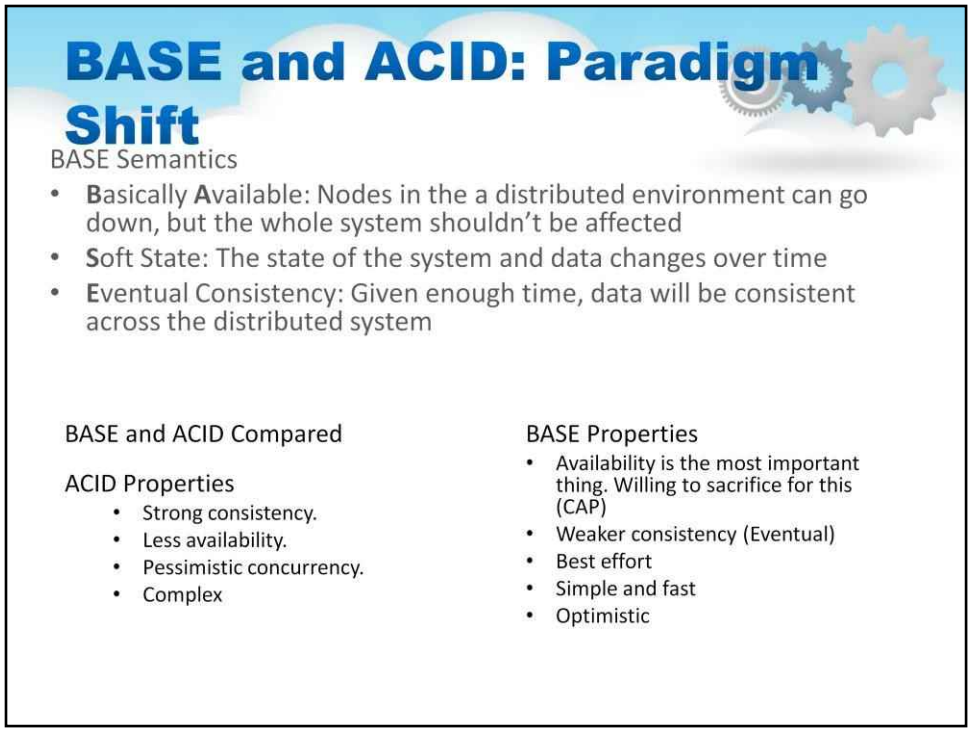
Atomic – уся робота в транзакції завершена (фіксована) або жодна з них не завершена

Consistent – транзакція перетворює базу даних з одного узгодженого стану в інший узгоджений стан. Узгодженість визначається в термінах обмежень.

яsolated – результати будь-яких змін, внесених під час транзакції, не видно, доки транзакція не буде зафіксована.

Дпридатний – результати здійсненої транзакції переживають невдачі

Послаблення властивостей ACID — це спосіб, у який деякі системи компрометують транзакції для досягнення вищої продуктивності та масштабованості.



Тепер подивимося уважніше на зміну парадигми BASE та ACID

Ми вивчали властивості ACID для реалізації транзакцій. Акронім BASE не випадково названий як протилежність ACID, а визначається як:

•**Б**азіально**А**доступний: вузли в розподіленому середовищі можуть вийти з ладу, але це не повинно вплинути на всю систему

•**С**часто стан: стан системи та даних змінюється з часом

•**Е**випадкова узгодженість: за достатньо часу дані будуть узгодженими по всій розподіленій системі

Це має на увазі «іншу поведінку» бази даних, щоб забезпечити більшу продуктивність, масштаб, географічний розподіл даних і так далі

Давайте порівняємо BASE і ACID

КИСЛОТА характеризується такими властивостями:

•Сильна консистенція.

•Менша доступність.

•Песимістична збіг.

•Комплекс

BASE характеризується іншими властивостями:

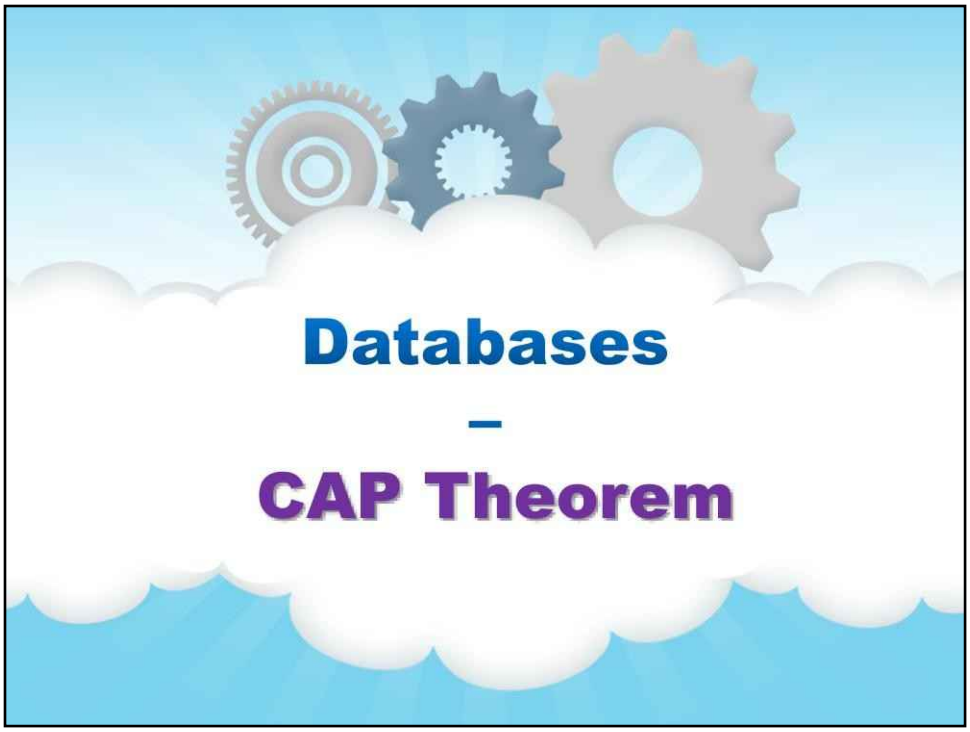
•Доступність - це найголовніше

•Слабша консистенція (можлива)

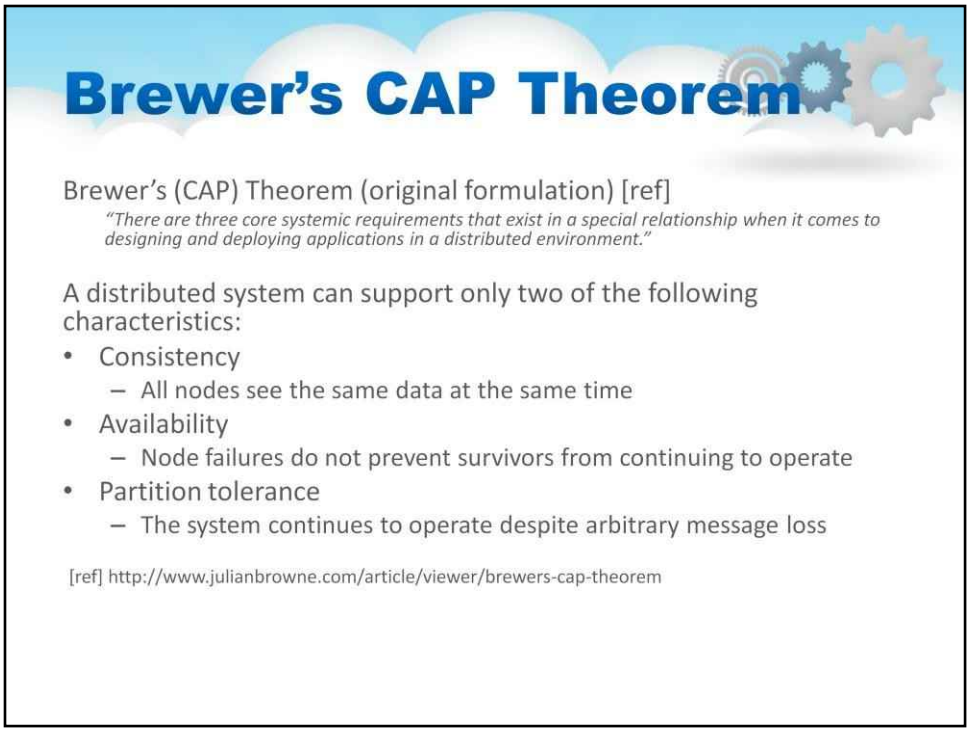
•Найкращі зусилля

•Просто і швидко

•Оптимістичний



Теорема CAP



Чому важко реалізувати такі речі, як транзакції в базах даних у хмарі? Оскільки хмара — це розподілена система.

Професор на ім'я Брюер вивчав цю область впровадження розподілених баз даних і опублікував теорему CAP

Він стверджував, що

«Існують три основні системні вимоги, які існують в особливому зв'язку, коли йдеться про розробку та розгортання програм у розподіленому середовищі».

Тоді він стверджував, що

Розподілена система може підтримувати лише дві з наведених нижче характеристик (у будь-якій конструкції):

Послідовність

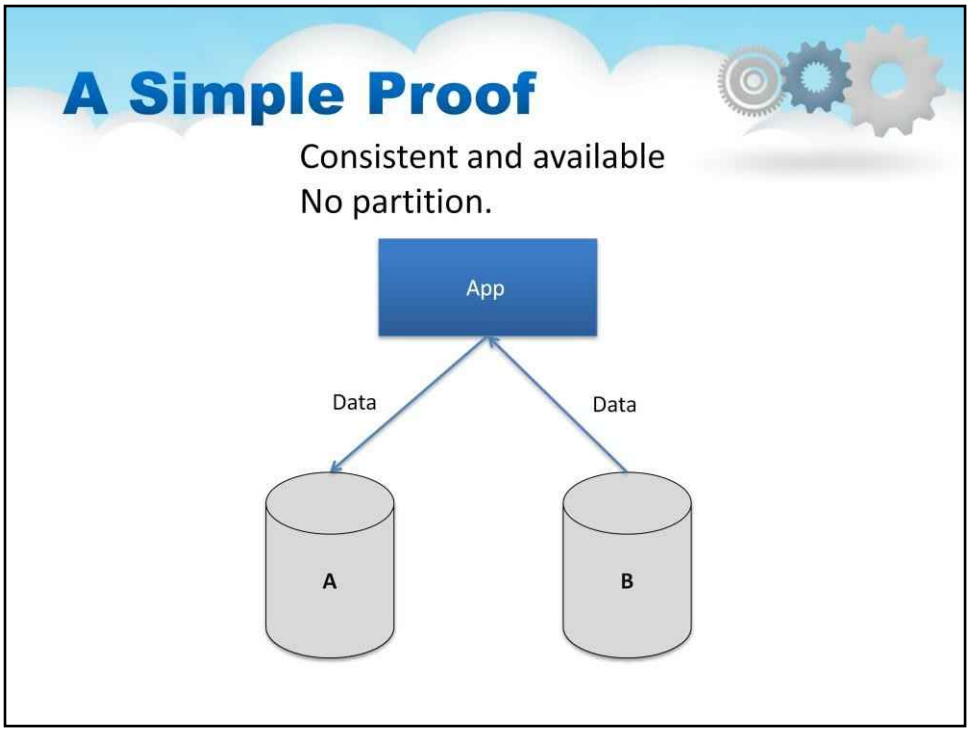
Усі вузли бачать однакові дані одночасно.

Доступність

Збої вузлів не заважають тим, хто вижив, продовжувати працювати з допуском розділу

Система продовжує працювати, незважаючи на довільну втрату повідомлення

Акронім для цього - CAP

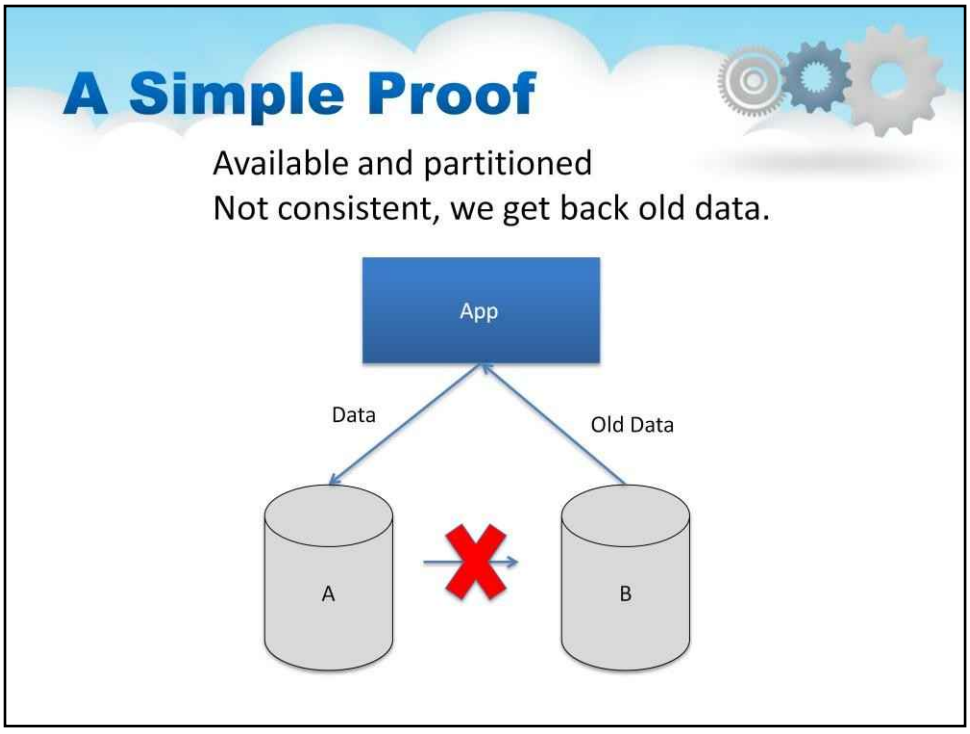


Давайте розглянемо кілька прикладів, як показано на слайді.

Спочатку ми розглянемо програму, підключену до даних, які знаходяться в двох службах зберігання. Всі дані є в обох місцях.

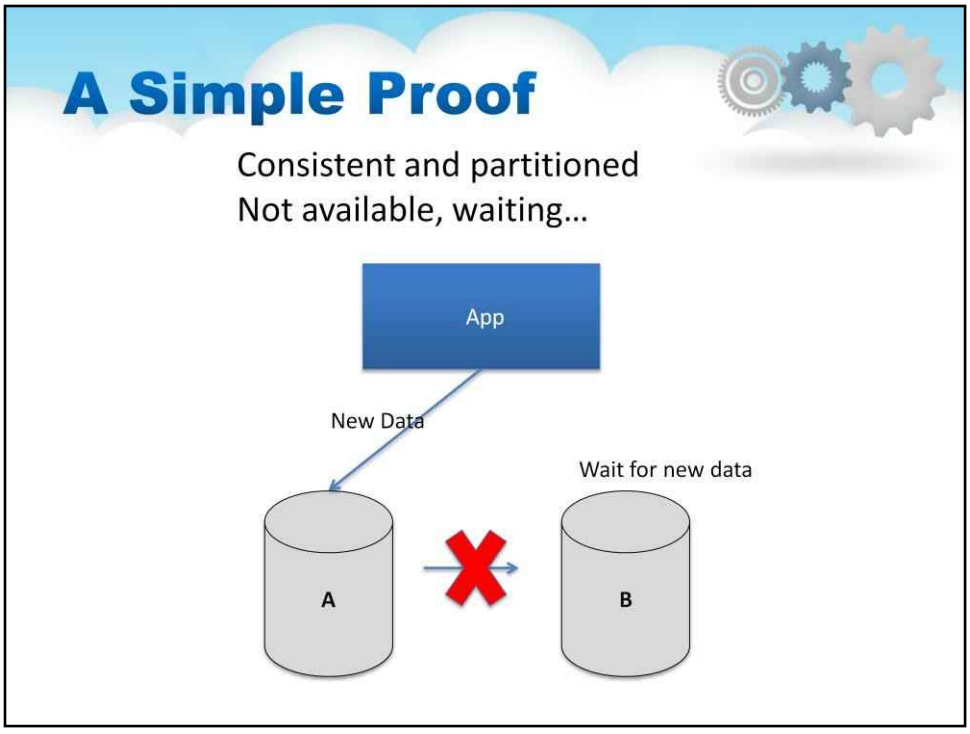
Оскільки дані онлайн і підключені до програми в обох місцях, програма бачить дані узгодженими та доступними незалежно від того, яка служба зберігання виконує дії читання чи запису

Мені не потрібно розділяти дані, оскільки вони доступні онлайн для мого додатка.



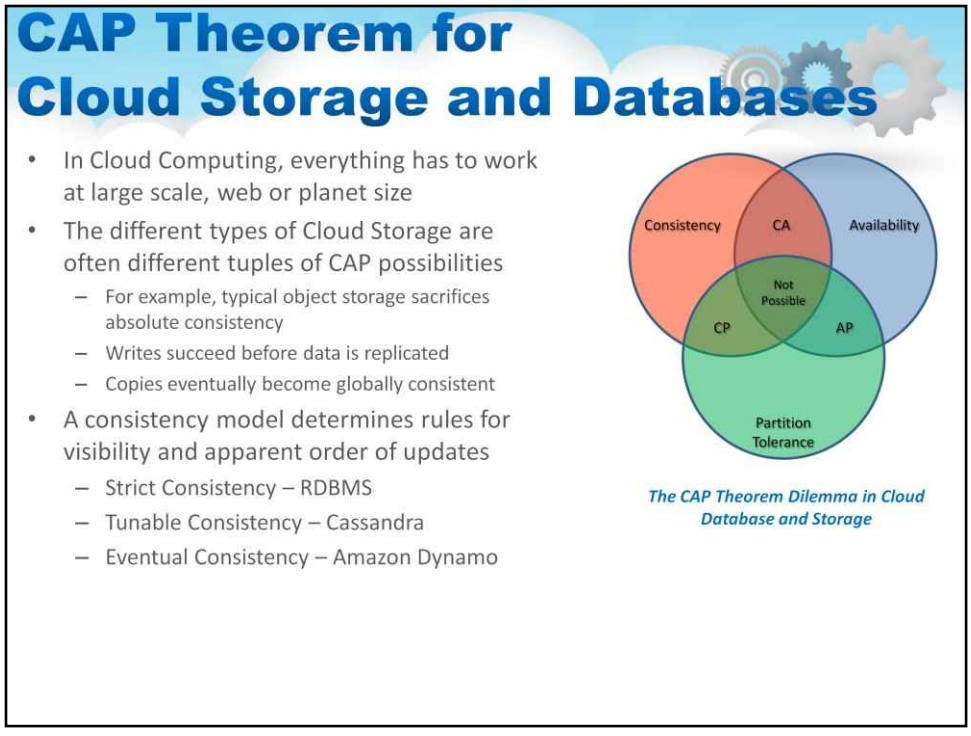
У цьому прикладі зроблено розділ, деякі дані вже старі. Якщо ви отримаєте доступ до служби, яка випадково має старі дані, і її обслуговуватиметься, це буде те, що ви отримаєте – старі дані

Ми намагалися бути доступними та розділеними, але це нам не вдалося.



У цьому випадку, як показано на слайді, якщо хтось наполягає на узгодженості, потрібно дочекатися синхронізації всіх служб зберігання. Це означає, що під час процесу синхронізації дані недоступні (наприклад, вони очікують).

Тепер легко зрозуміти, чому узгодженість, розділення та доступність пов’язані.



Це суть проблеми, яку мають бази даних на хмарних платформах із великими наборами даних. Вона називається теоремою про розподілені (хмарні) бази даних, а точніше теоремою CAP.

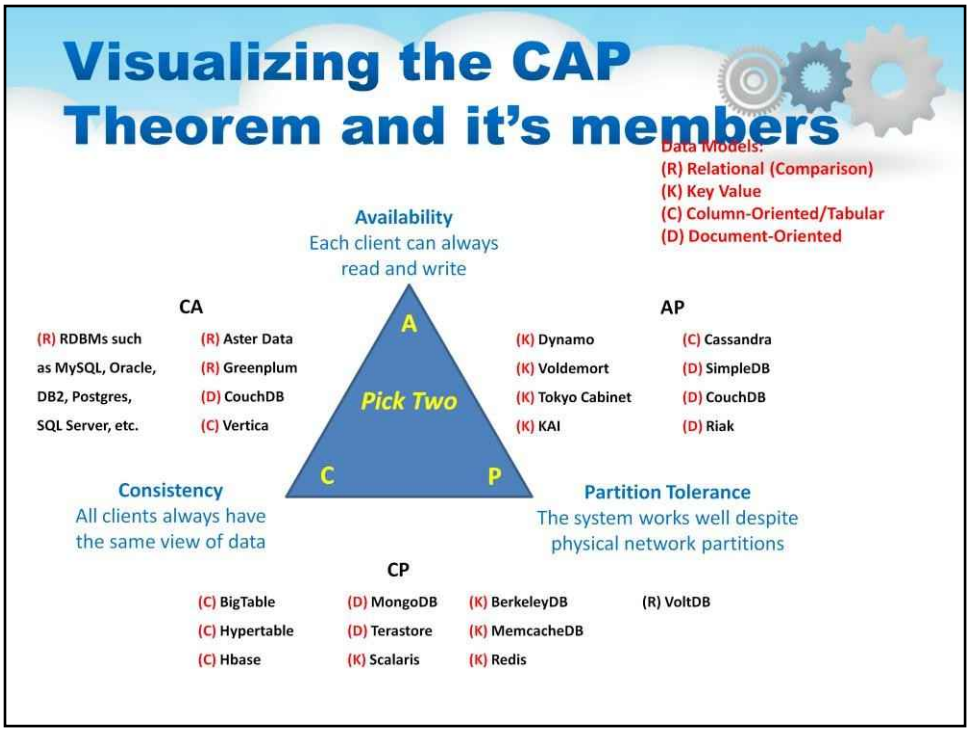
У ньому стверджується, що в хмарних обчисленнях все має працювати у великих масштабах планети. Області, які, здається, суперечать одна одній у розгортанні великої бази даних:

- Узгодженість: усі клієнти мають бачити поточні дані незалежно від оновлень чи видалень
- Доступність: система продовжує працювати належним чином навіть із збоями вузлів
- Допуск до розділів: система продовжує працювати належним чином, незважаючи на збої мережі або системи повідомлень

Багато розробників задавалися питанням, як вирішити цю проблему.

- Можливість поділу: розділіть вузли на невеликі групи, які можуть бачити інші групи, але не можуть бачити всіх.
- Узгодженість: напишіть значення, а потім прочитайте значення, і ви отримаєте те саме значення. У розділеній системі є вікна, де це не так.
- Доступність: не завжди можна писати чи читати. Система скаже, що ви не можете писати, тому що вона хоче підтримувати послідовність системи. Для масштабування вам потрібно розділити, тому вам залишається вибрати або високу послідовність, або високу доступність для конкретної системи. Ви повинні знайти правильне перекриття доступності та послідовності.

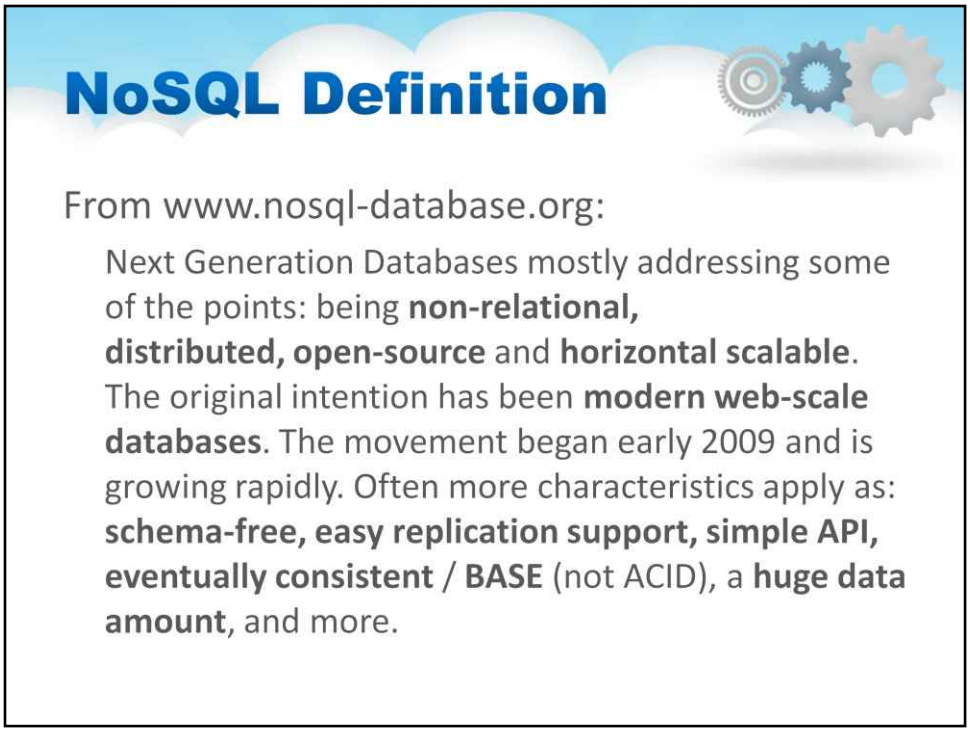
Вибирайте конкретний підхід, виходячи з потреб послуги.



Діаграма на цьому слайді наочно ілюструє теорему CAP. Він також класифікує комерційні реалізації та реалізації з відкритим кодом в одну з трьох категорій CAP Theorem.

NoSQL

Огляд



Визначення NoSQL

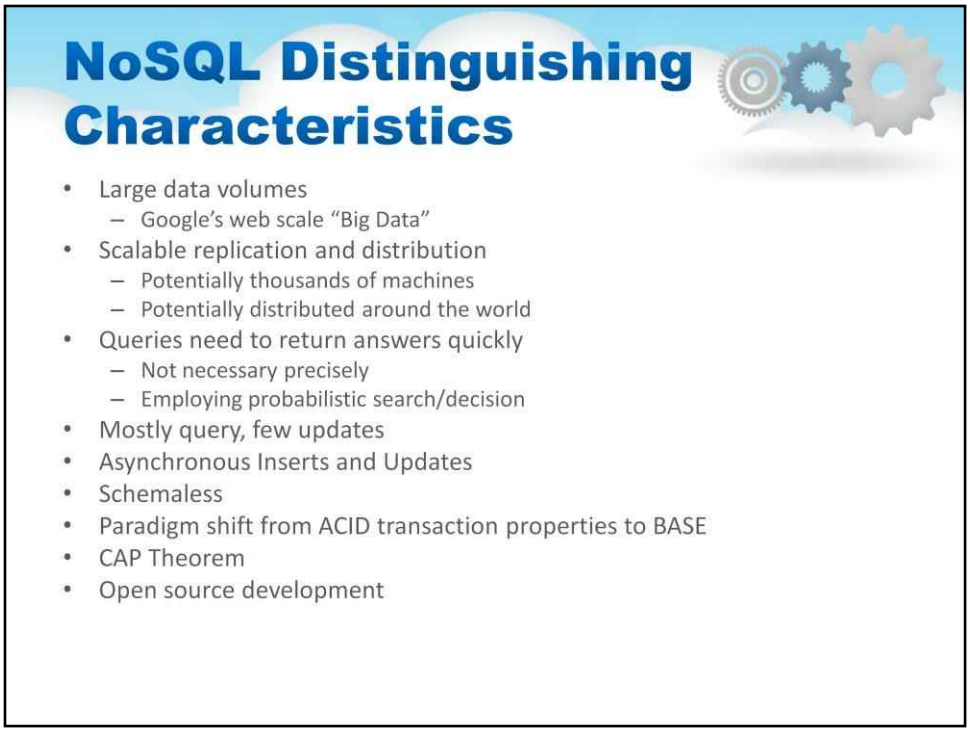
Бази даних наступного покоління здебільшого стосуються деяких питань:

буття**нереляційний, розподілений, з відкритим кодом**і**горизонтально масштабований**.

Початковий намір був**сучасні бази даних веб-масштабу**.

Рух розпочався на початку 2009 року і швидко розвивається.

Часто застосовуються такі характеристики, як:**без схем, проста підтримка реплікації, простий API, зрештою послідовний**/**БАЗА**(не КИСЛОТА), а**величезний обсяг даних**, і більше.



Інший спосіб поглянути на NoSQL – це розрізнити характеристики

Великі обсяги даних

Веб-шкала Google "Big Data"

Масштабована реплікація та розподіл

Потенційно тисячі машин Потенційно розподілені по всьому світу

Запити повинні швидко повертати відповіді

Не обов'язково точно

Використання імовірнісного пошуку/рішення

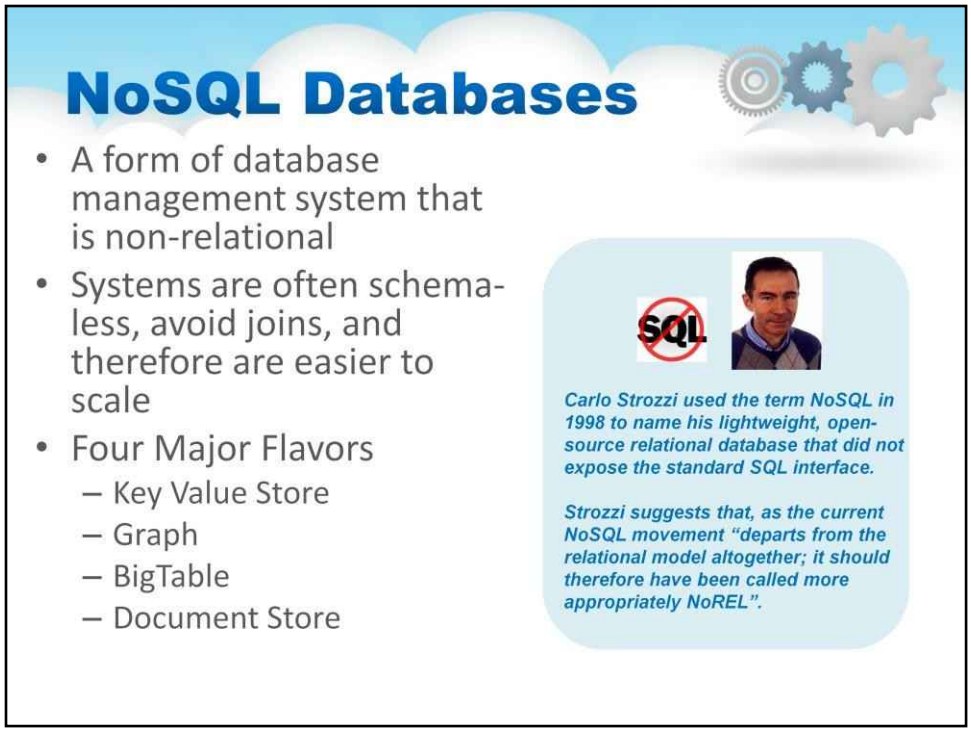
Переважно запити, кілька оновлень

Асинхронні вставки та оновлення без схеми

Зміна парадигми від властивостей транзакцій ACID до теореми BASE CAP

Розробка з відкритим кодом

Далі ми розглянемо деякі з цих тем більш детально



Ви зауважите, що багато баз даних у кількісній оцінці теореми CAP називаються «Базами даних NoSQL».

Цей цікавий термін відноситься до форми системи керування базами даних, яка не є реляційною. Ці системи часто не містять схем, уникають об’єднань і тому їх легше масштабувати.

Як трохи історії, Карло Строцці використав термін NoSQL у 1998 році, щоб назвати свою легку реляційну базу даних з відкритим кодом, яка не розкривала стандартний інтерфейс SQL.

Строцці припускає, що, оскільки поточний рух NoSQL «повністю відходить від реляційної моделі; тому його слід було називати більш доцільно NoREL».

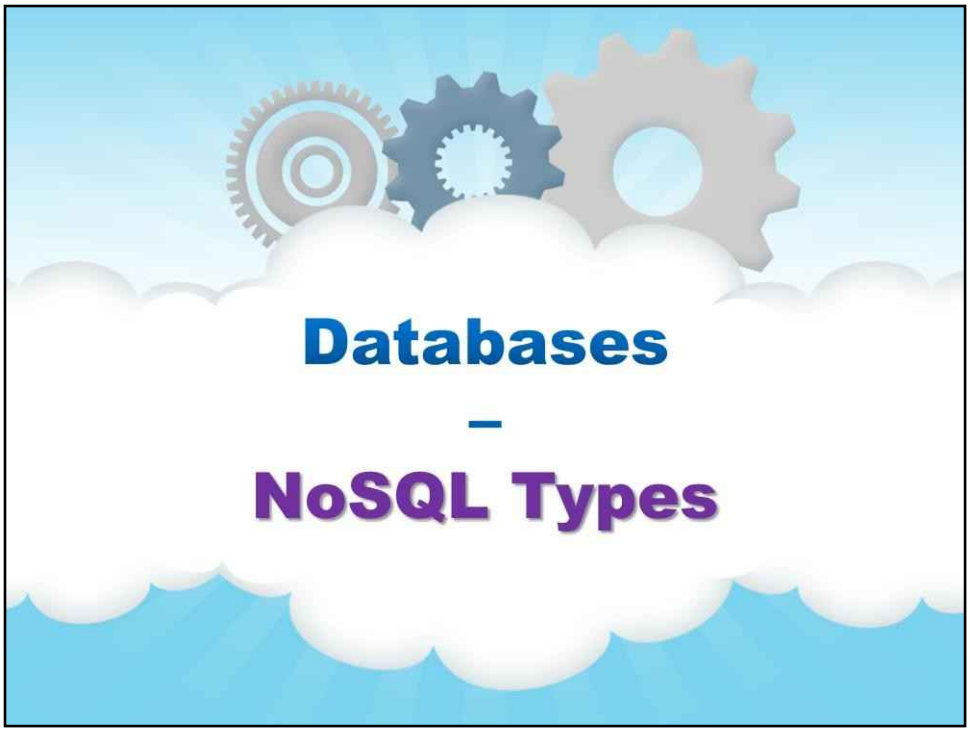
Є чотири основні смаки

Key Value Store

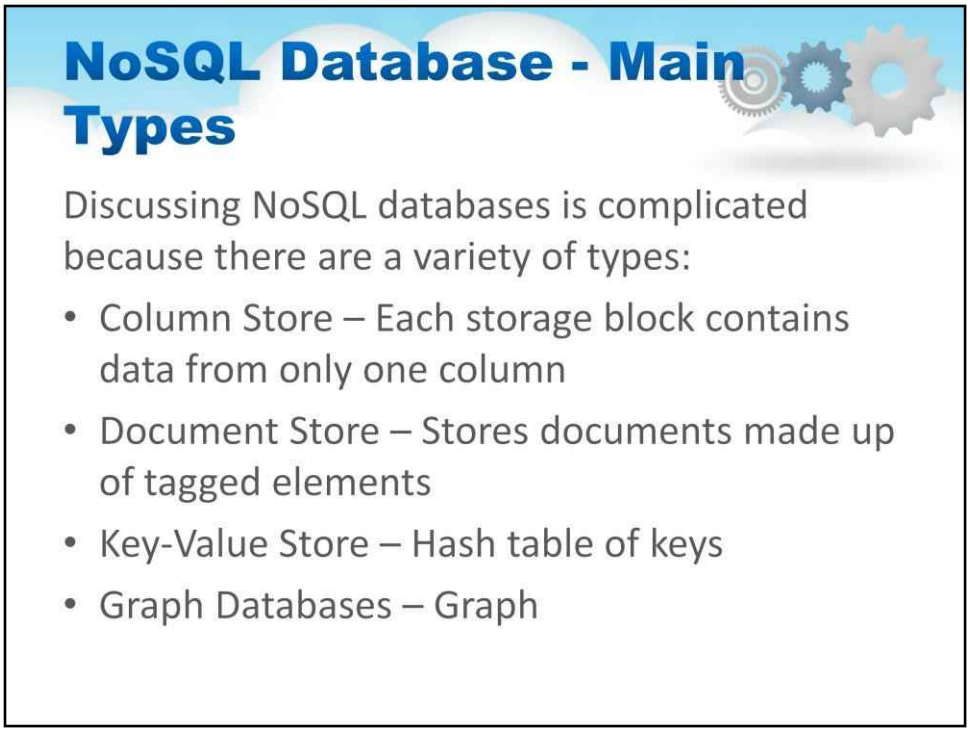
Графік

BigTable

Магазин документів



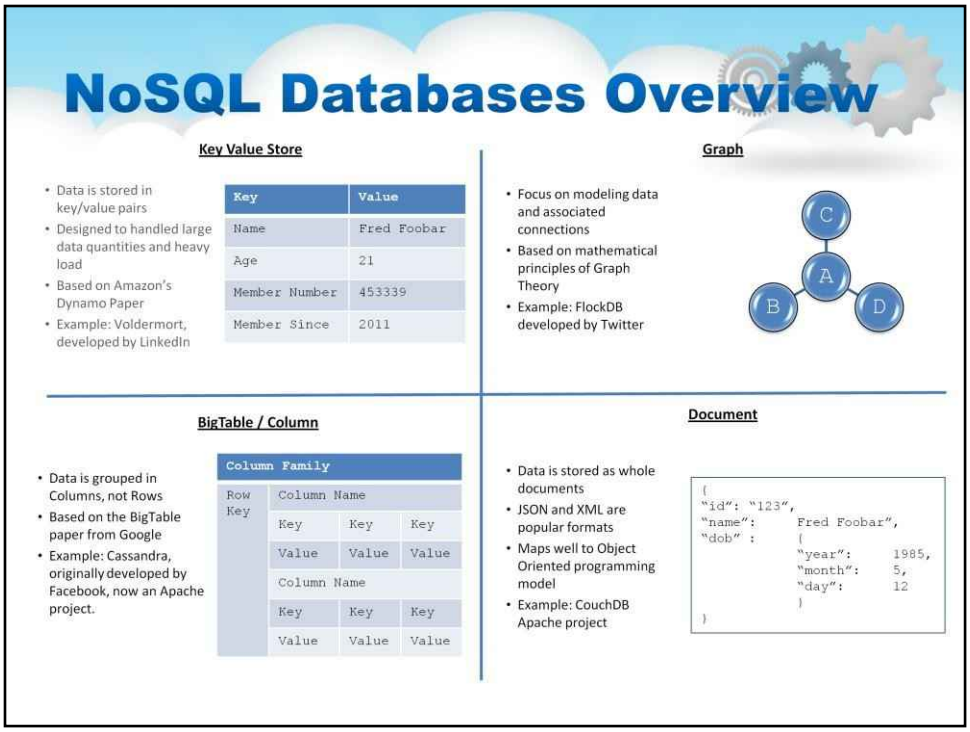
Типи NoSQL



Далі ми зануримося в бази даних NoSQL – спочатку розглянемо основні типи

Обговорювати бази даних NoSQL складно, оскільки існує безліч типів:

Column Store – кожен блок зберігання містить дані лише з одного стовпця
Document Store – зберігає документи, що складаються з тегованих елементів Key-Value Store – хеш-таблиця ключів
Графові бази даних – Graph



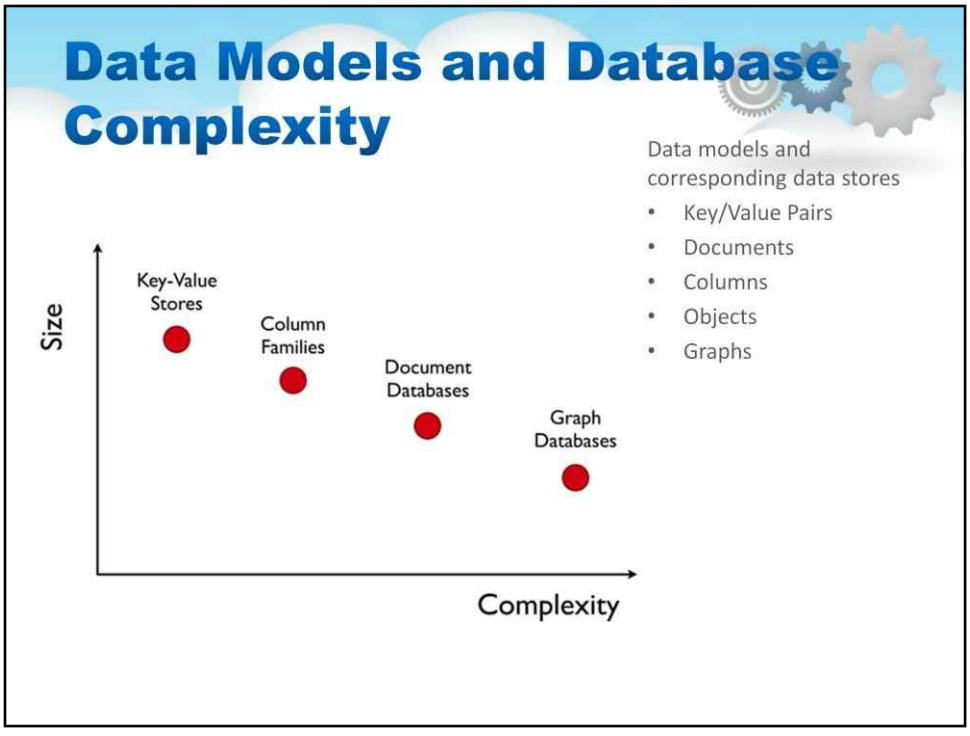
На цьому слайді показано чотири основні типи баз даних noRel разом із тим, як виглядають структури даних у них.

У Key Value Store Дані зберігаються в парах ключ/значення
Розроблено для обробки великої кількості даних і великого навантаження На основі Amazon Dynamo Paper
Приклад: Voldermort, розроблений LinkedIn

У Graph основна увага приділяється моделюванню даних і пов'язаних з ними зв'язків. На основі математичних принципів теорії графів
Приклад: FlockDB, розроблений Twitter

У BigTable / Дані стовпців групуються в стовпці, а не в рядки.
На основі документа BigTable від Google
Приклад: Cassandra, спочатку розроблений Facebook, тепер проект Apache.

Нарешті, у документі Дані зберігаються як цілі документи.
JSON і XML є популярними форматами
Добре відповідає моделі об'єктно-орієнтованого програмування. Приклад: проект CouchDB Apache

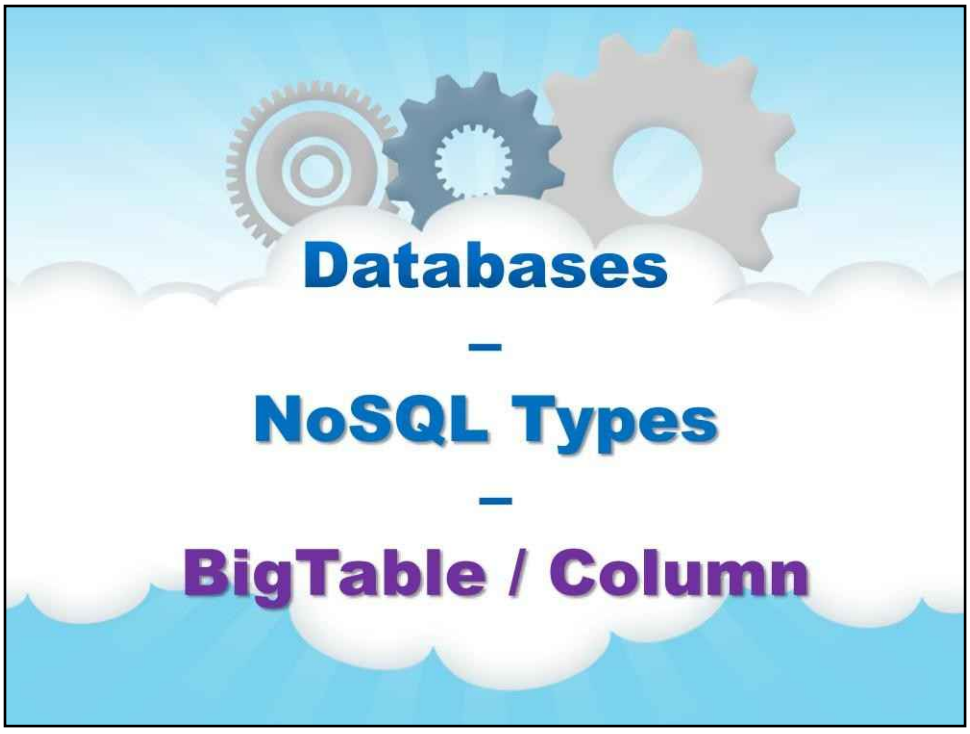


Один із способів класифікації цих різних підходів проілюстровано на слайді.

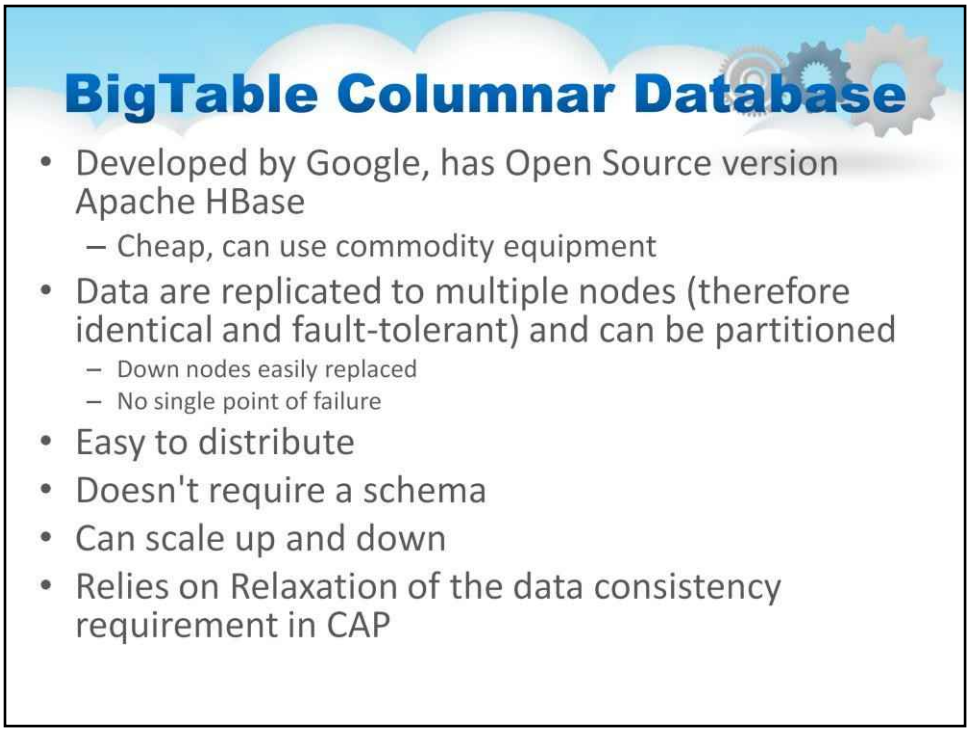
Кожна з різних систем має дещо інший розмір даних і компроміс щодо складності.

Це означає, що чим складніший аналіз потрібно виконати на наборі даних, тим меншим буде набір даних, який може зрозуміти ця база даних.

Кожна техніка відповідає певній точці компромісу – це показано на ілюстрації.



BigTable / Колонка



Спочатку ми зосередимося на одному з перших рішень NoSQL, заснованому на ранніх роботах Google. Він називається BigTable.

Існує версія з відкритим вихідним кодом під назвою Apache HBase

Дані реплікуються на кілька вузлів (отже, ідентичні та відмовостійкі) і можуть бути розділені

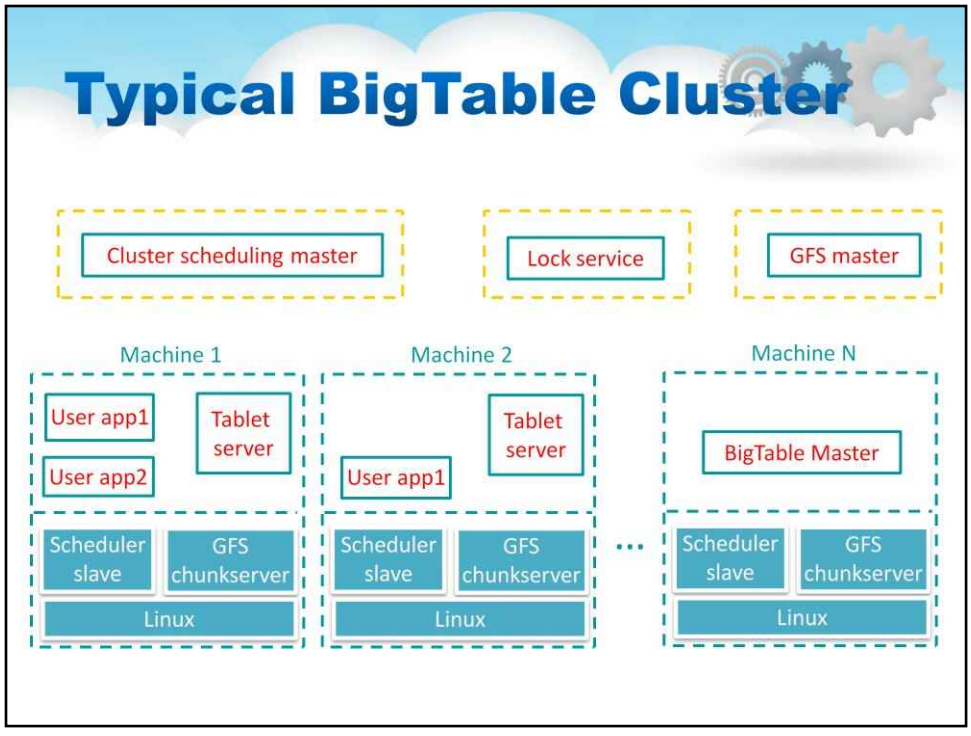
Несправні вузли легко замінюються.
Немає єдиної точки відмови

BigTable легко розповсюджувати.
Не потребує схеми. Можна
масштабувати вгору та вниз

BigTable покладається на послаблення вимог узгодженості даних у CAP

- > Оскільки дані записуються, остання версія принаймні на одному вузлі. Потім дані версії/реплікація на інші вузли в системі.

- > Зрештою, однакова версія на всіх вузлах.

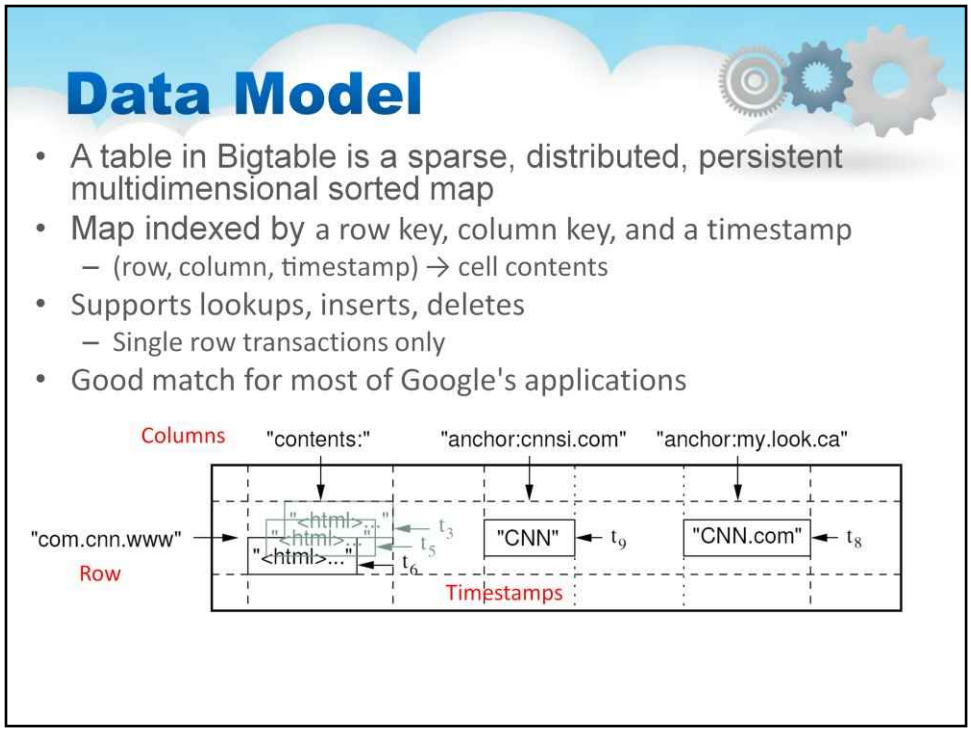


Ось приклад того, як BigTable реалізовано в кластері. Перегляньте ілюстрацію на слайді.

Зауважте, що є один або кілька вузлів керування (чи машин, чи віртуальних машин), на яких запущено планування, блокування та файлову систему

Інші N вузлів (або машин, або віртуальних машин) налаштовано на роботу з планувальником і файловою системою, кожен з яких працює над частиною («фрагментом») даних

Планшети пояснюються трохи пізніше.



Таблиця в Bigtable — це розріджена, розподілена, постійна багатовимірна відсортована карта

Ця карта індексована ключем рядка, стовпця та міткою часу

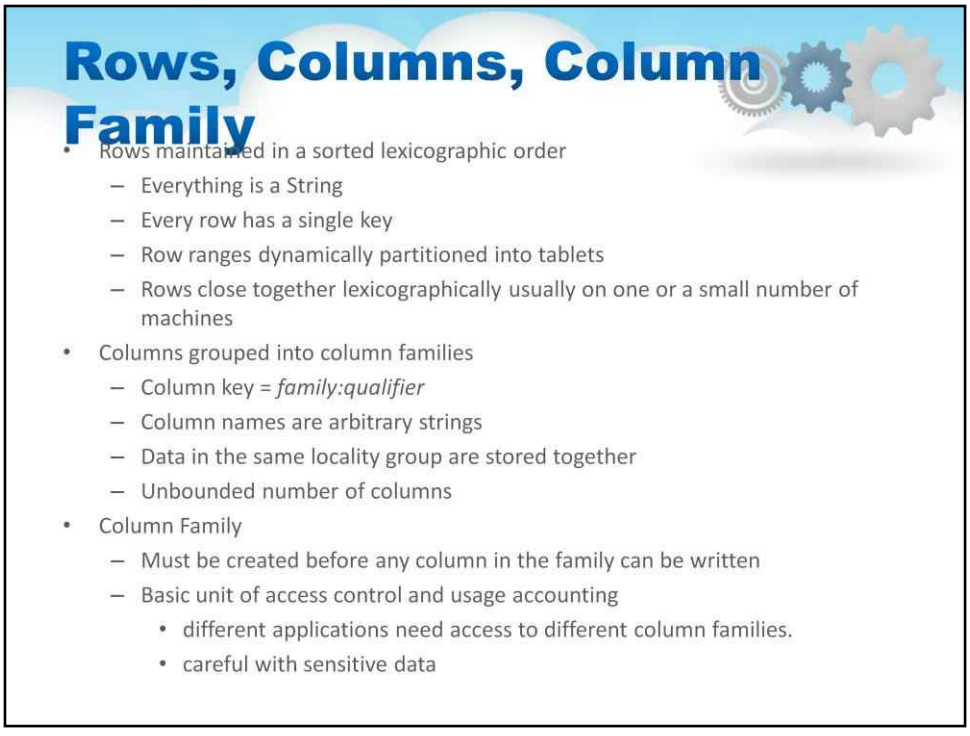
Мітки часу використовуються для зберігання різних версій даних у клітинці

Для нових записів за замовчуванням використовується поточний час, але мітки часу для записів також можуть бути встановлені клієнтами явно

Існує кілька варіантів пошуку. Приклади:

«Повернути останні значення K»

«Повернути всі значення в діапазоні позначок часу (або всі значення)»



У BigTable рядки, стовпці, сімейство стовпців мають особливу структуру

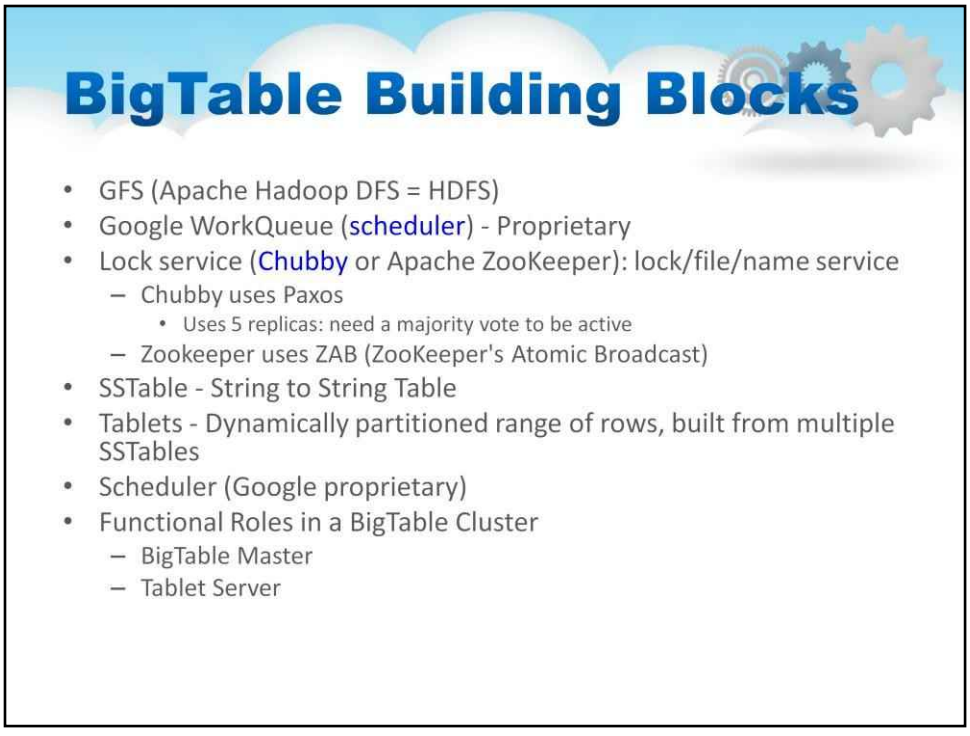
Рядки зберігаються в упорядкованому лексикографічному порядку

Усе є рядком Кожен
рядок має один ключ
Діапазони рядків, динамічно розділені на таблетки
Рядки зближуються лексикографічно зазвичай на одній або невеликій кількості машин

Стовпці групуються в сімейства стовпців

Ключ стовпця = родина: кваліфікатор Назви стовпців є довільними рядками
Дані в одній групі локалізації зберігаються разом у необмеженій кількості стовпців

Сімейства стовпців Необхідно створити, перш ніж можна буде записати будь-який стовпець у сімействі
Вони є базовою одиницею контролю доступу та обліку використання



BigTable потребує набору модулів, щоб створити повне рішення

Цеглинки є

Файлова система – Google GFS (Apache Hadoop DFS = HDFS) Робоча черга – Google WorkQueue (планувальник) – власність

Служба блокування (Chubby або Apache ZooKeeper): служба блокування/файлів/імен

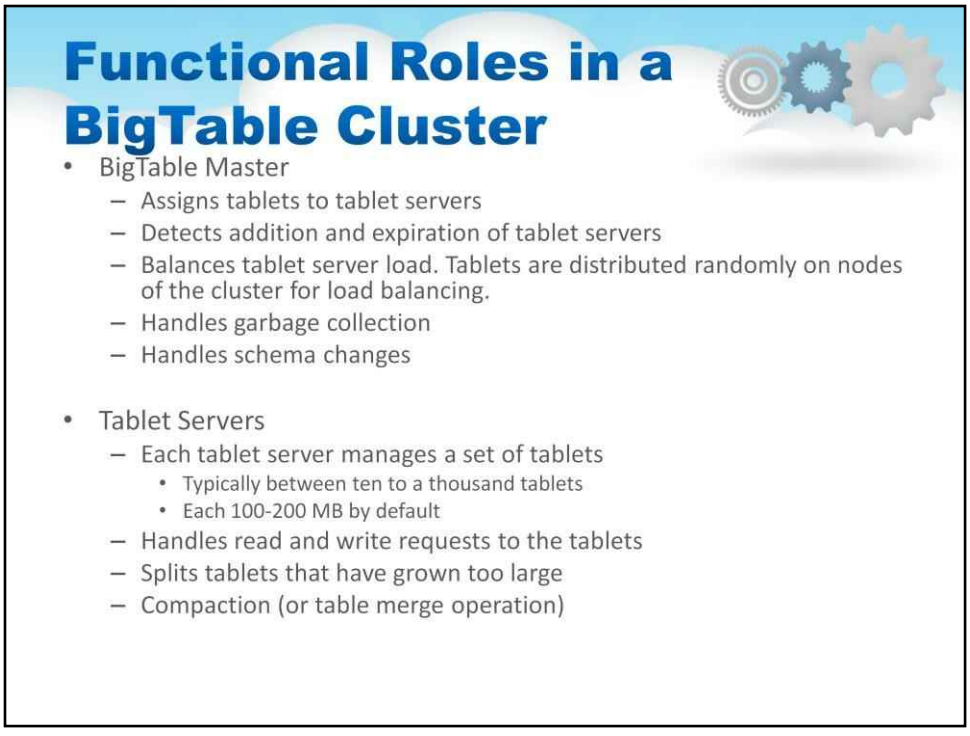
Chubby використовує Paxos

Використовує 5 реплік: потрібна більшість голосів, щоб бути активним Zookeeper використовує ZAB (ZooKeeper's AtomicBroadcast)

SSTable - таблиця рядків до рядків

Планшети – динамічно розділений діапазон рядків, створений із кількох SSTables

Планувальник (власний Google)



Функціональні ролі в кластері BigTable такі:

Майстер BigTable

Призначає планшети до планшетних серверів

Виявляє додавання та закінчення терміну дії планшетних серверів

Балансує навантаження на сервер планшета. Планшети розподіляються випадковим чином по вузлах кластера для балансування навантаження.

Обробляє збирання сміття

Обробляє зміни схеми

Сервери планшетів

Кожен планшетний сервер керує набором планшетів

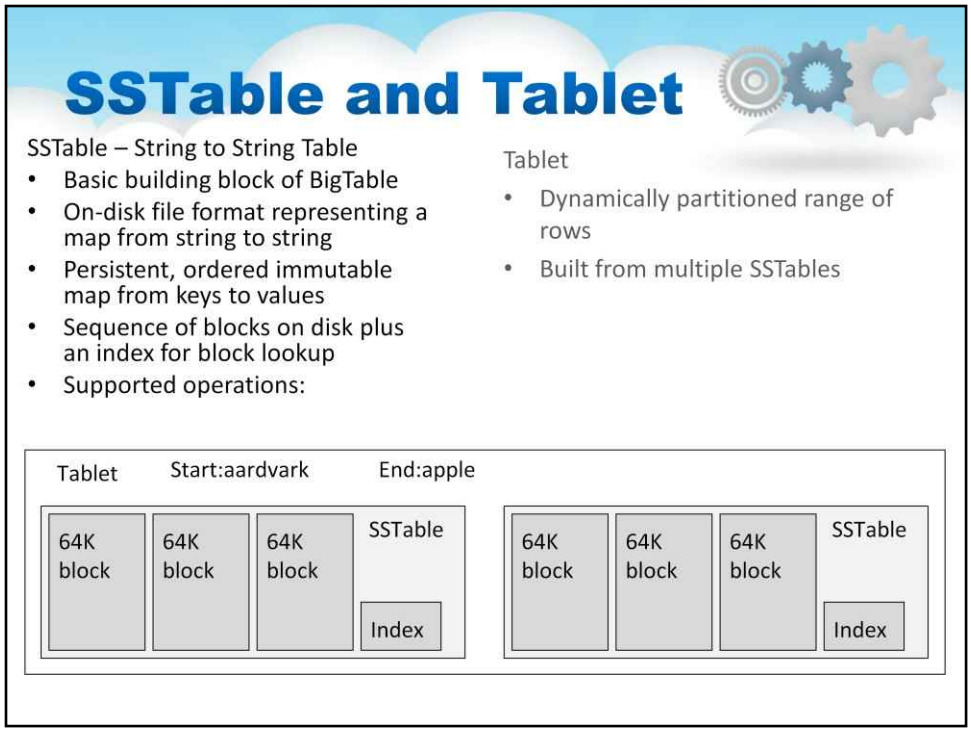
Зазвичай від десяти до тисячі планшетів кожен по

100-200 МБ за замовчуванням

Обробляє запити на читання та запис до планшетів

Розділяє планшети, які виросли занадто великими

Стискання (або операція злиття таблиць)



Детальніше про SSTable і планшет

SSTable–Рядок до таблиці рядків Основний будівельний блок BigTable

Формат файлу на диску, що представляє карту від рядка до рядка Постійна, впорядкована незмінна карта від ключів д значень

Зберігається в GFS

Послідовність блоків на диску плюс індекс для пошуку блоків Може бути повністю відображено в пам'яті

Підтримувані операції:

- Пошук значення, пов’язаного з ключем
- Ітерація пар ключ/значення в діапазоні ключів

планшет

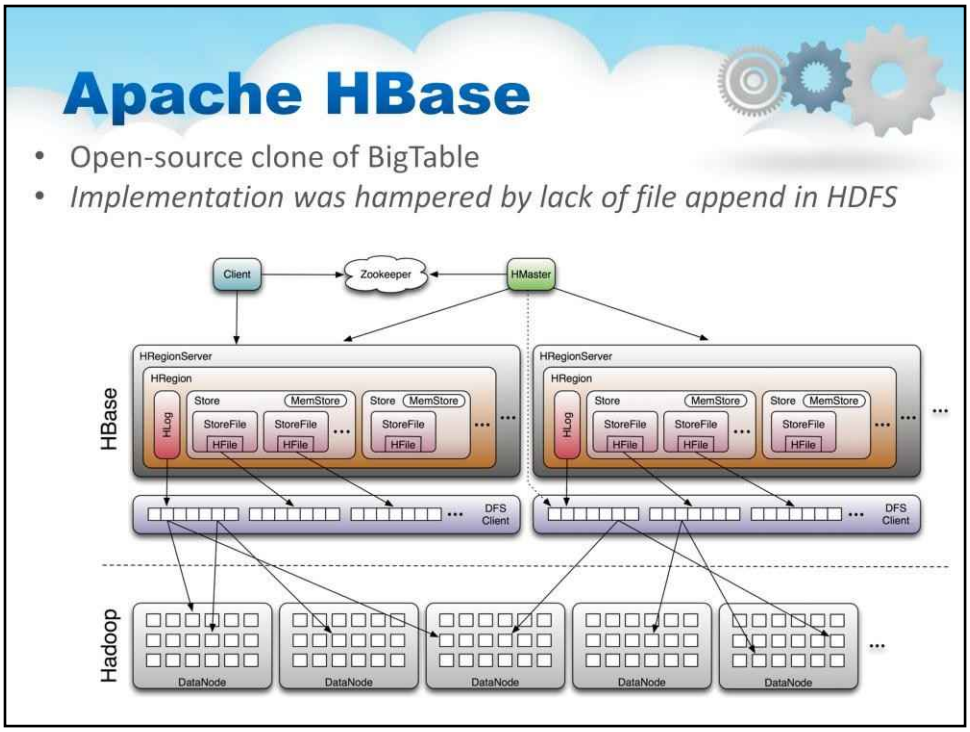
Динамічно розділений діапазон рядків, створений із кількох SSTables

Розповсюджується на планшетних серверах Блок балансування навантаження

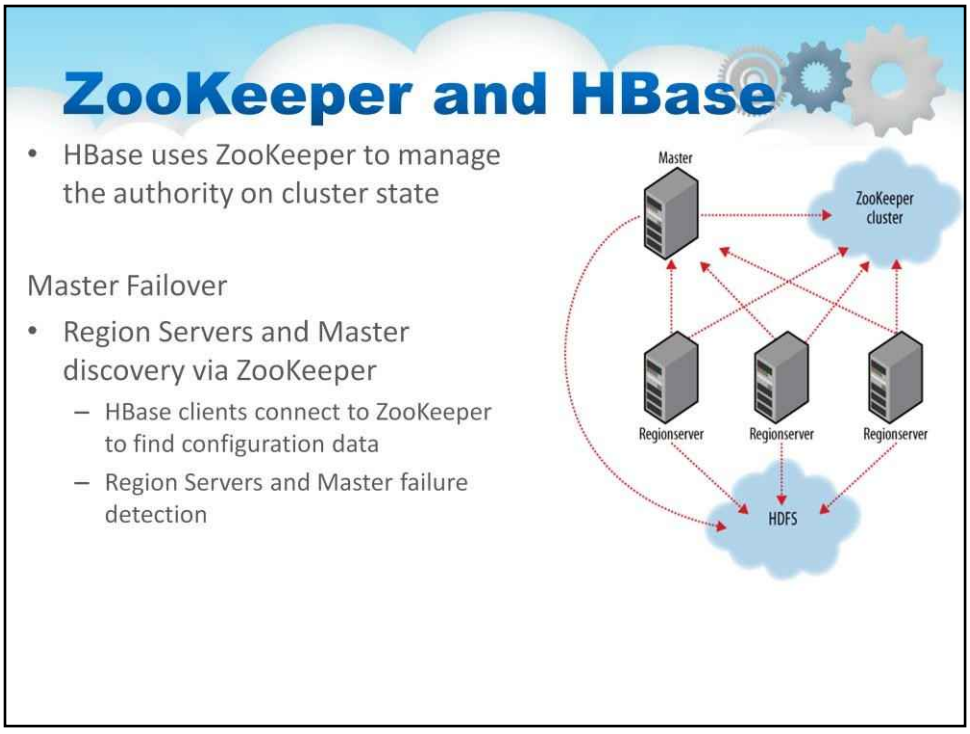
Планшети розділяють і об’єднують автоматично залежно від розміру та завантаження або вручну

Клієнти можуть вибрати ключі рядків для досягнення локальності

На слайді є докладна ілюстрація, яка показує деталі руди на цих компонентах, а також те, як вони поєднуються.



Як згадувалося, Apache Hbase є клоном BigTable з відкритим вихідним кодом. Спочатку його впровадження ускладнювалося відсутністю додавання файлів у HDFS. Для цієї проблеми опубліковано обхідні шляхи.



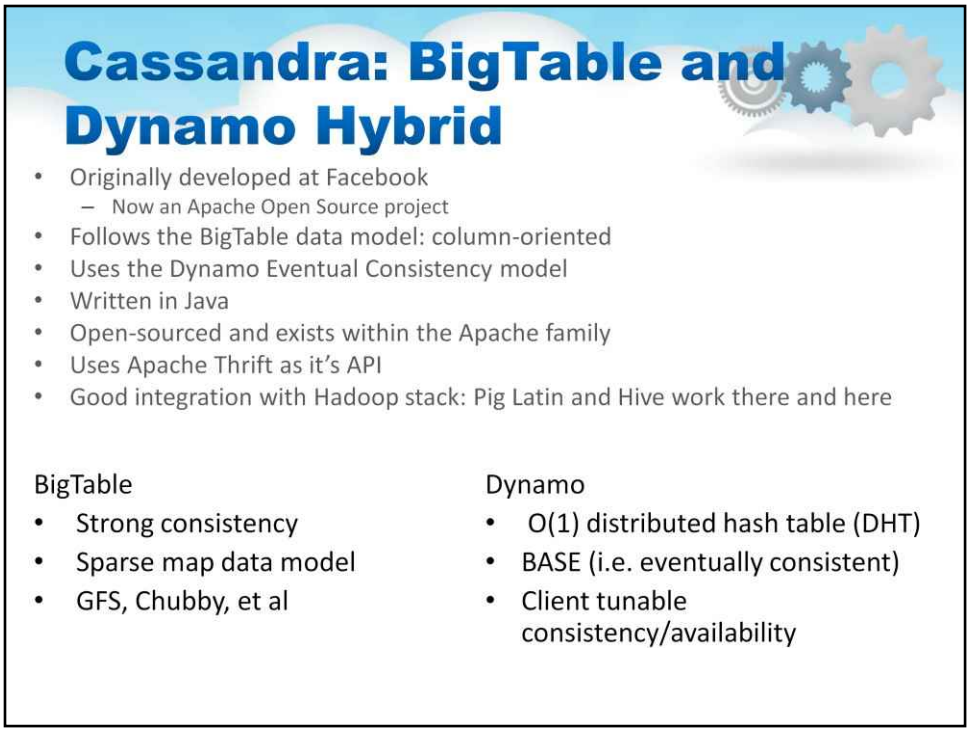
ZooKeeper — інструмент для кластеризації Hbase

На слайді зображено кластер ZooKeeper із регіональними серверами та HDFS.

HBase використовує ZooKeeper для керування повноваженнями щодо стану кластера

ZooKeeper керує регіональними серверами та основним виявленням

Клієнти HBase підключаються до ZooKeeper, щоб знайти конфігураційні дані



Cassandra набула дуже великої популярності як реалізація BigTable

Він використовує систему на основі DHT, схожу на систему Dynamo, описану Amazon, яка контролює багато модулів AWS

Кассандра спочатку була розроблена у Facebook

Тепер це проект Apache з відкритим кодом

Він відповідає моделі даних BigTable: орієнтований на стовпці

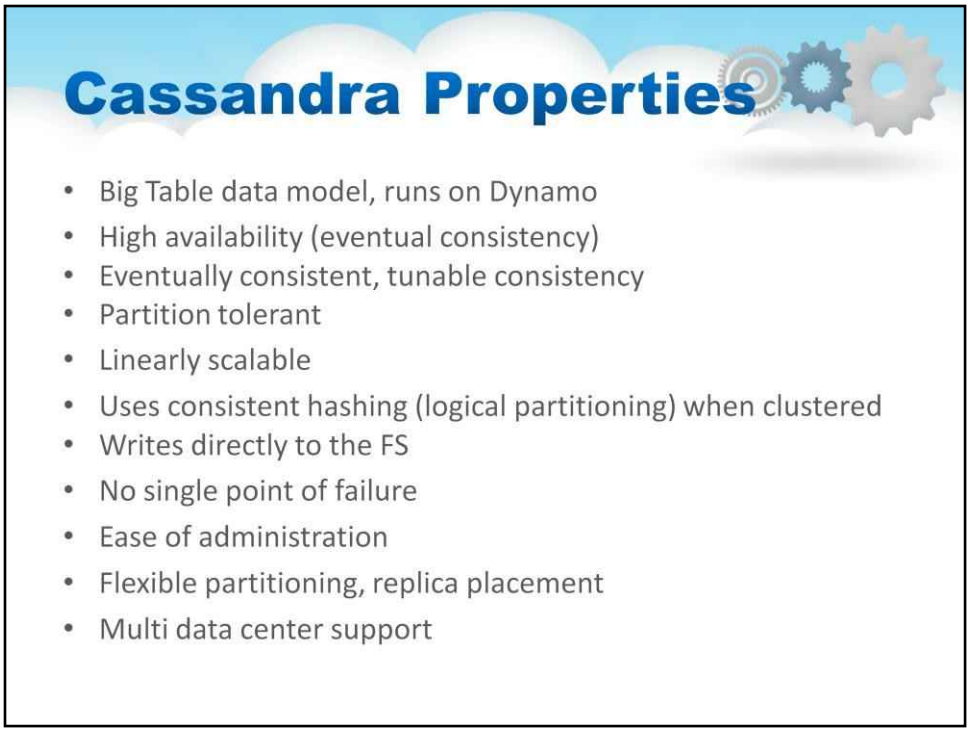
Використовує модель Dynamo Eventual Consistency

Він дуже портативний, написаний на Java

Він відкритий і існує в сімействі Apache

Cassandra використовує Apache Thrift як свій API

Крім того, він має хорошу інтеграцію зі стеком Hadoop: Pig Latin і Hive працюють там і тут



Властивості Кассандри більш детально

Кассандра дуже здібна. Він заснований на моделі даних Big Table

Він реалізує:

Висока доступність (можлива узгодженість)
Зрештою узгоджена, настроювана узгодженість
Толерантність до розділів

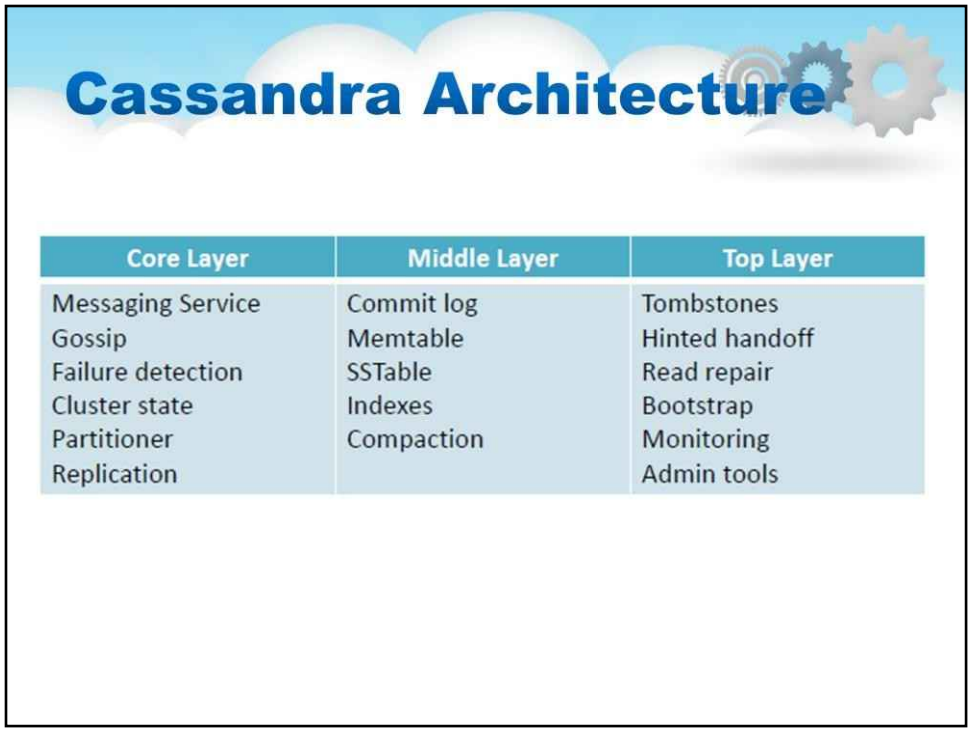
Лінійно масштабована

Використовує узгоджене хешування (логічне розділення) у кластері
Записує безпосередньо до FS

Жодної точки відмови

Гнучке розділення, розміщення реплік Підтримка

кількох центрів обробки даних



Архітектуру Cassandra можна описати так:

Верхній рівень розроблено для забезпечення ефективного, узгодженого читання та запису за допомогою простого API. API Cassandra складається з простих методів отримання та встановлення та не має посилання на розподілену природу бази даних.

Ще один елемент у верхньому шарі — Hinted hand-off. Це відбувається, коли вузол виходить з ладу - вузол-наступник стає координатором (тимчасово) з деякою інформацією (`підказкою') про невдалий вузол.

Середній рівень містить функції для обробки даних, що записуються в базу даних. Ущільнення намагається поєднати ключі та стовпці для підвищення продуктивності системи. Тут також обробляються різні способи зберігання даних, наприклад Memtable і SSTable.

Основний рівень стосується розподіленої природи бази даних і містить функції для зв’язку між вузлами, стану кластера в цілому (включаючи виявлення збоїв) і реплікації між вузлами.

Глибший погляд на HintedHandoff

Cassandra Hinted hand off — це техніка оптимізації для запису даних у репліки, коли запис зроблено, а вузол репліки для ключа не працює

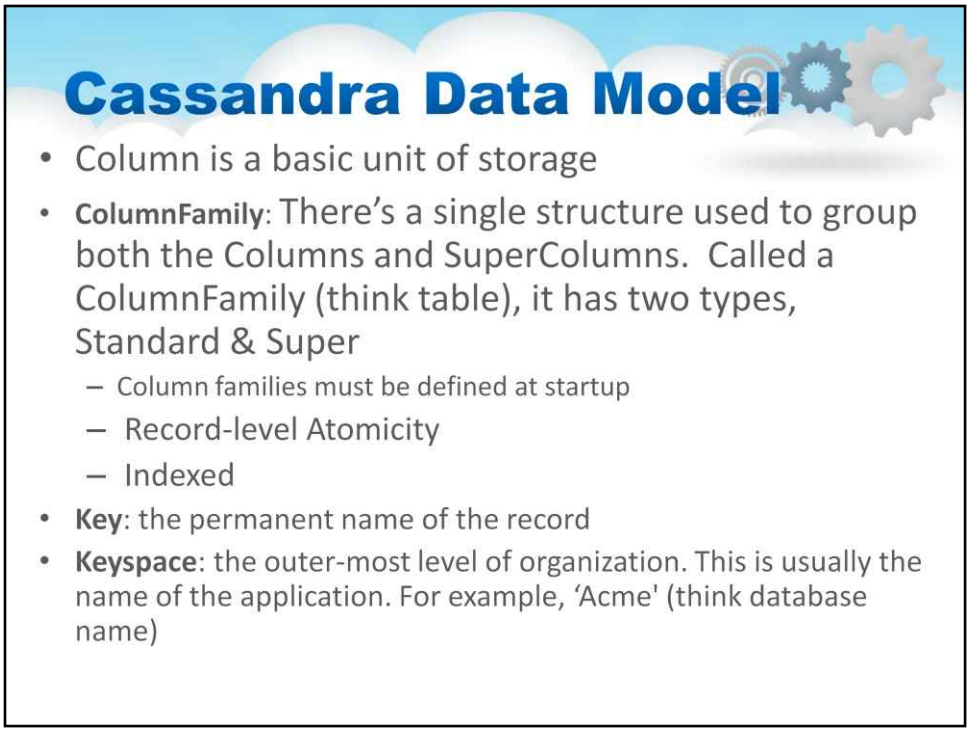
Кассандра напише підказку до живого вузла репліки

Цей вузол-репліка буде нагадувати збитому вузлу про зміни, коли він повернеться до мережі

HintedHandoff зменшує затримку запису, коли репліка тимчасово не працює.
HintedHandoff забезпечує високу доступність запису ціною узгодженості

Запис із підказкою НЕ зараховується до вимог ConsistencyLevel для ONE, QUORUM або ALL

Якщо для цього ключа немає активних вузлів-реплік і було вказано ConsistencyLevel.ANY, вузол-координатор запише підказку локально



Тепер розглянемо модель даних Cassandra

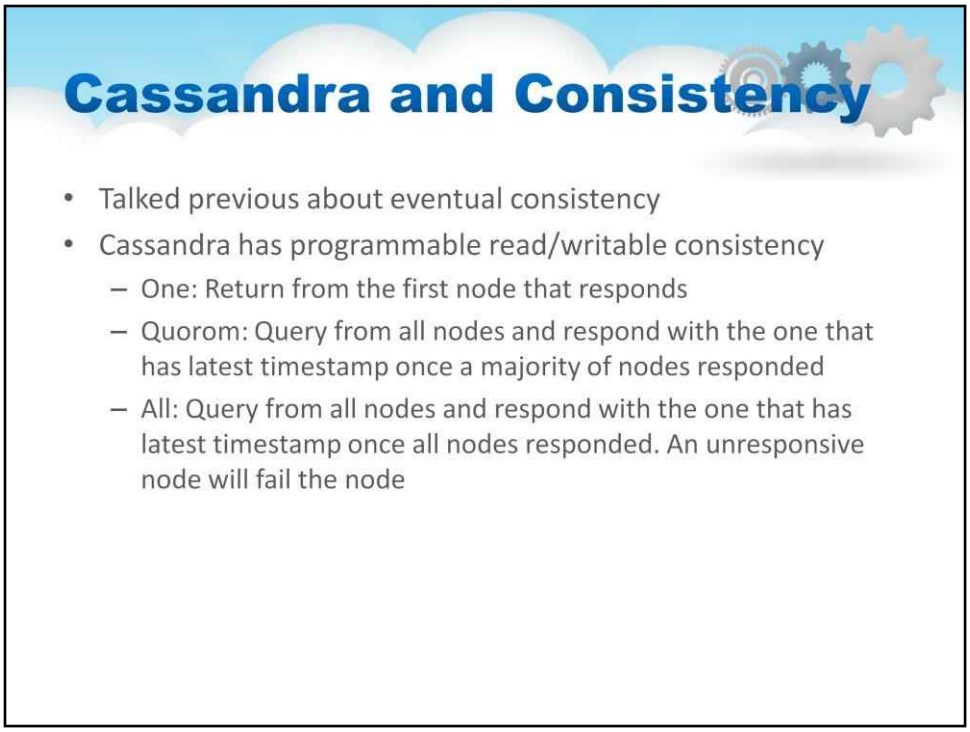
The**Колонка**є основною одиницею зберігання
Є єдина структура, яка використовується для групування стовпців і надстовпців. Називається ColumnFamily (таблицею) і має два типи: стандартний і супер

The**ключ**: постійна назва запису

The**Простір клавіш**: це зовнішній рівень організації.
Зазвичай це назва програми. Наприклад, "Acme" (уявіть назву бази даних)

Що потрібно знати про Кассандру:

- > Ключі мають різну кількість стовпців, тому база даних може масштабуватися нерівномірно.
- > Простий і супер: суперколонки – це колонки в колонках.
- > Зверніться до дати за**простір клавіш**, а**родина стовпців**, а**ключ**ова**супер колонка**, і а**колонка**.



Кассандра і консистенція

Попередня охоплена можлива консистенція, яка є режимом, у якому працює Кассандра.

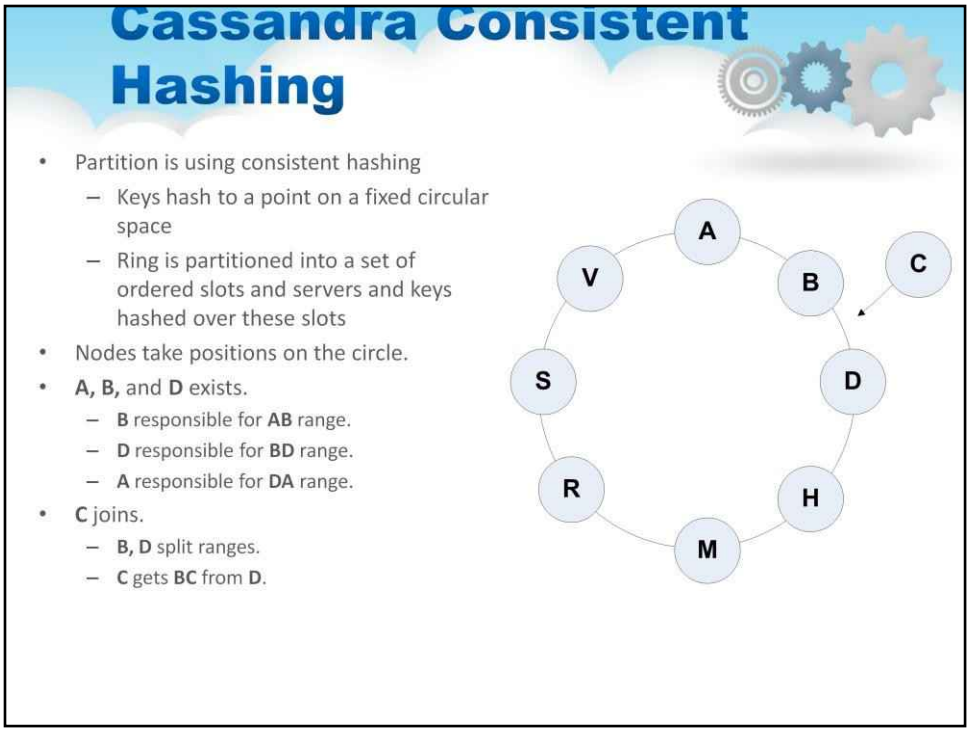
Cassandra має програмовану послідовність читання/запису

«Один»: повернення від першого вузла, який відповідає

«Quorom»: надсилайте запити з усіх вузлів і надсилайте відповідь тим, який має останню позначку часу, коли більшість вузлів відповіли

«Усі»: надсилайте запити з усіх вузлів і надсилайте відповідь тим, який має останню позначку часу, коли всі вузли відповідають. Вузол, який не відповідає, виведе з ладу вузол

- > http://www.slideshare.net/julesbravo/cassandra-3125809
- > Майте хороший простий тест, щоб продемонструвати різницю між Cassandra та MySQL



Cassandra Scales завдяки своїй узгодженій архітектурі хешування (Dynamo DHT Style)

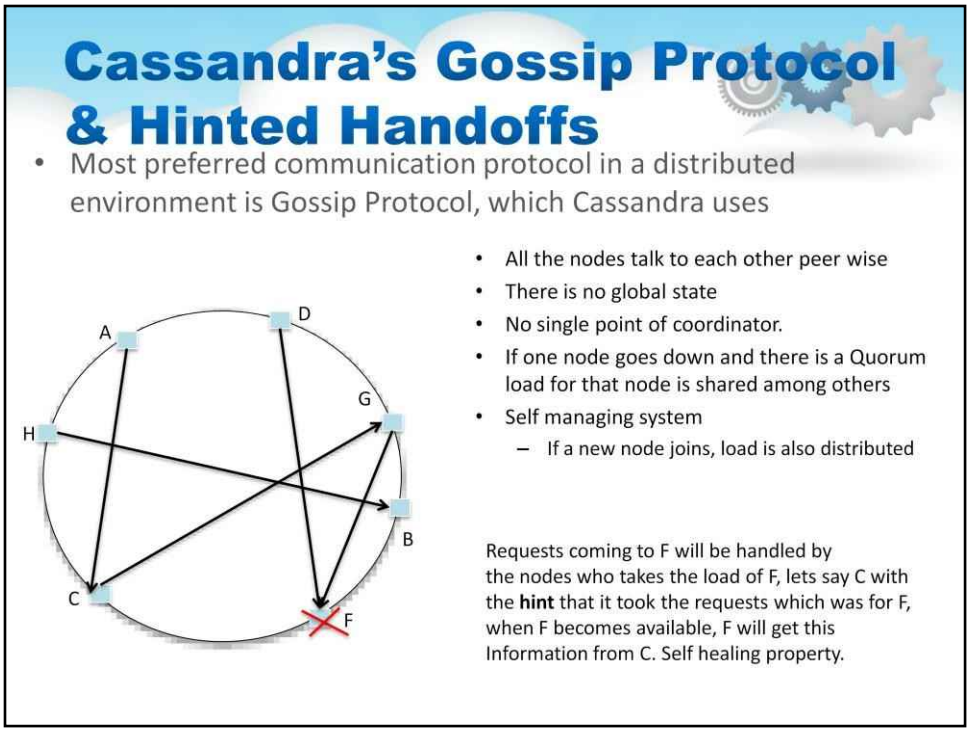
Розділ використовує послідовне хешування

Ключі хешу до точки на фіксованому круговому просторі

Кільце розділене на набір упорядкованих слотів і серверів, а ключі хешуються над цими слотами

Вузли займають положення на колі.

На слайді є ілюстрація та приклад діапазонів кругового простору.



Gossip Protocol — це метод вирішення цього комунікаційного хаосу

Cassandra використовує цей найбільш бажаний протокол зв'язку в розподіленому середовищі

У плітках:

Усі вузли взаємодіють між собою однорангово.

Глобального стану немає

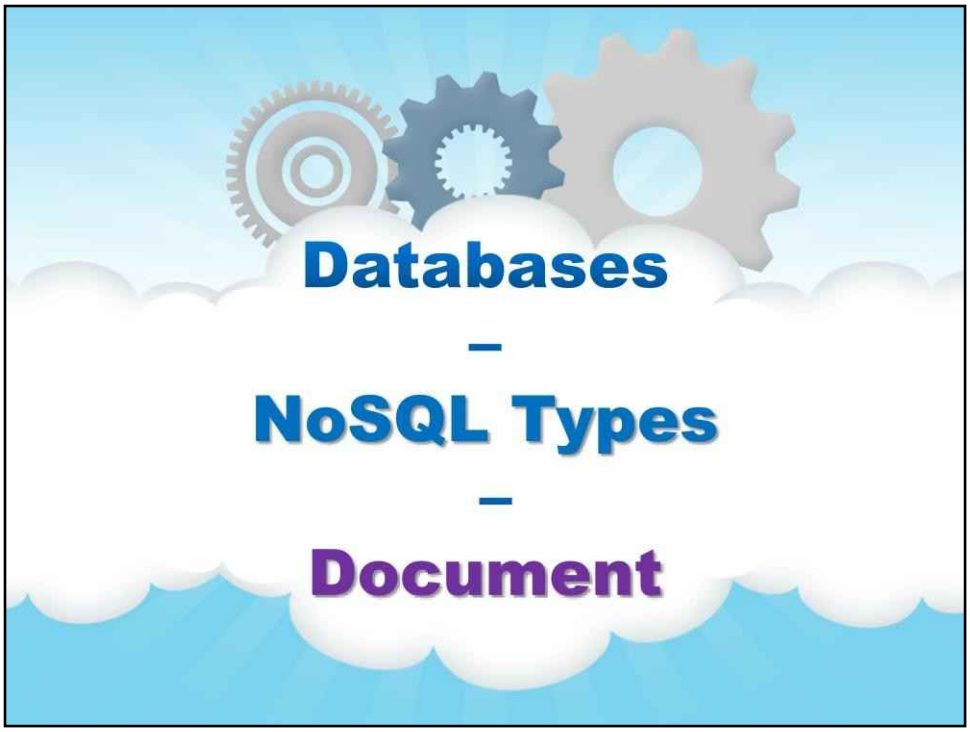
Немає єдиного координатора.

Якщо один вузол виходить з ладу та є навантаження кворуму для цього вузла, розподіляється між ними інші

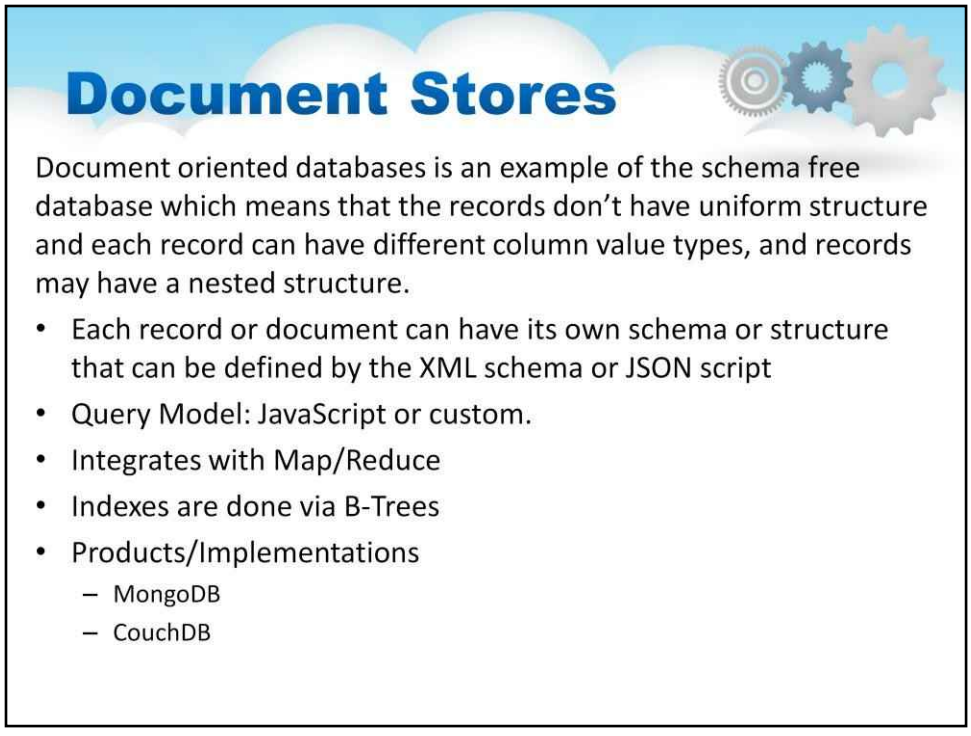
Це система самокерування

Якщо приєднується новий вузол, навантаження також розподіляється

На слайді є ілюстрація, яка показує протокол, а також містить приклад як це працює.



документ



Документно-орієнтовані бази даних є іншою архітектурою бази даних.

Вони є прикладом бази даних без схем, що означає, що записи не мають однорідної структури, і кожен запис може мати різні типи значень стовпця, а записи можуть мати вкладену структуру.

Кожен запис або документ може мати власну схему або структуру, яка може бути визначена схемою XML або сценарієм JSON

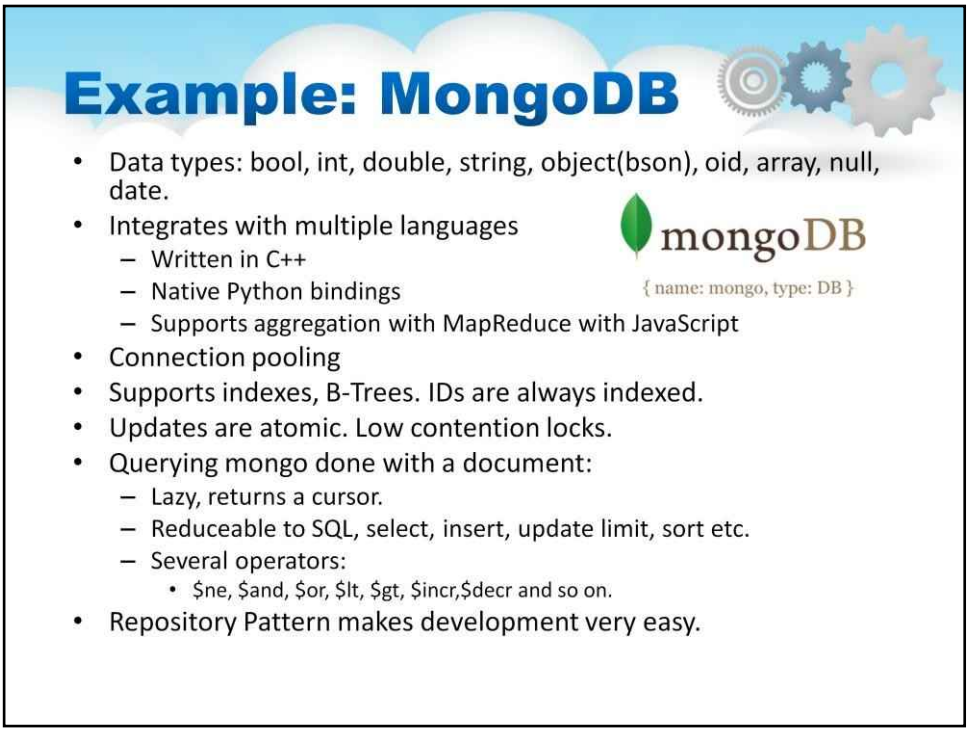
Модель запиту: JavaScript або спеціальна.

Інтегрується з Map/Reduce

Індекси виконуються за допомогою B-дерев

Продукти/впровадження

MongoDB
CouchDB



Щоб детально ознайомитися з MongoDB

Він підтримує багато типів даних: bool, int, double, string, object(bson), oid, array, null, date.

Він інтегрується з кількома мовами

Написано мовою C++

Власні прив’язки Python

Підтримує агрегацію за допомогою MapReduce з JavaScript

Він реалізує пул підключень

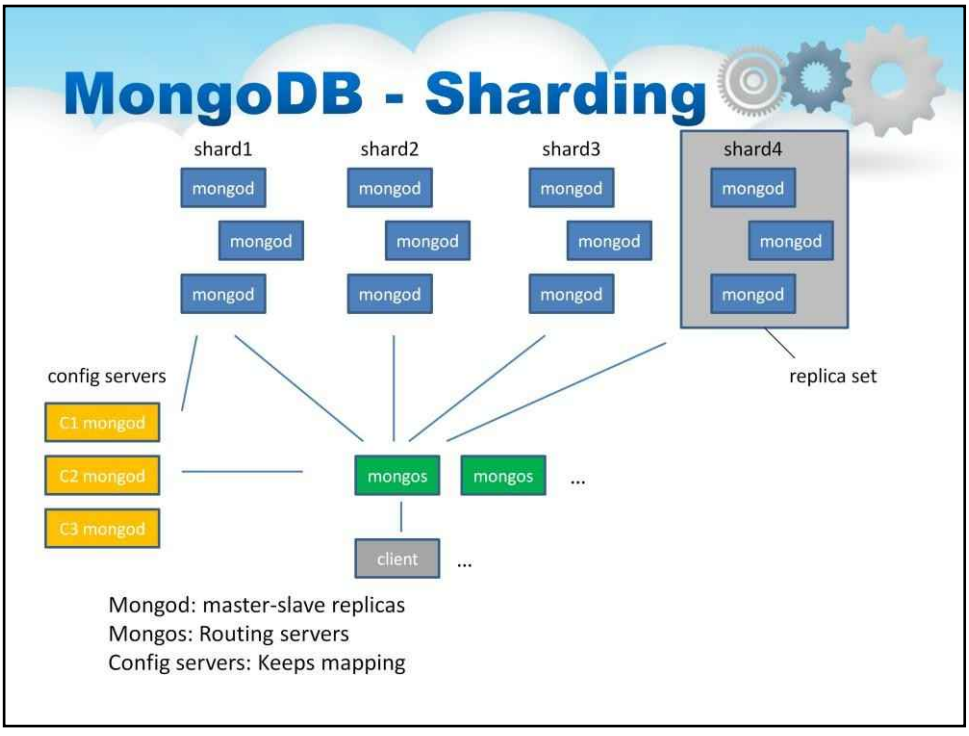
Він підтримує індекси, B-дерева. Ідентифікатори завжди індексуються.

Оновлення MongoDB є атомарними. Є блокування з низьким рівнем конкуренції.

Запит mongo виконано за допомогою документа:

Це ледачий запит, що означає, що результати не повертаються, а курсор у місці результату.

Дещо можна звести до підмножини SQL, вибору, вставки, обмеження оновлення, сортування тощо. Об’єднань немає, як це типово для систем NoSQL.

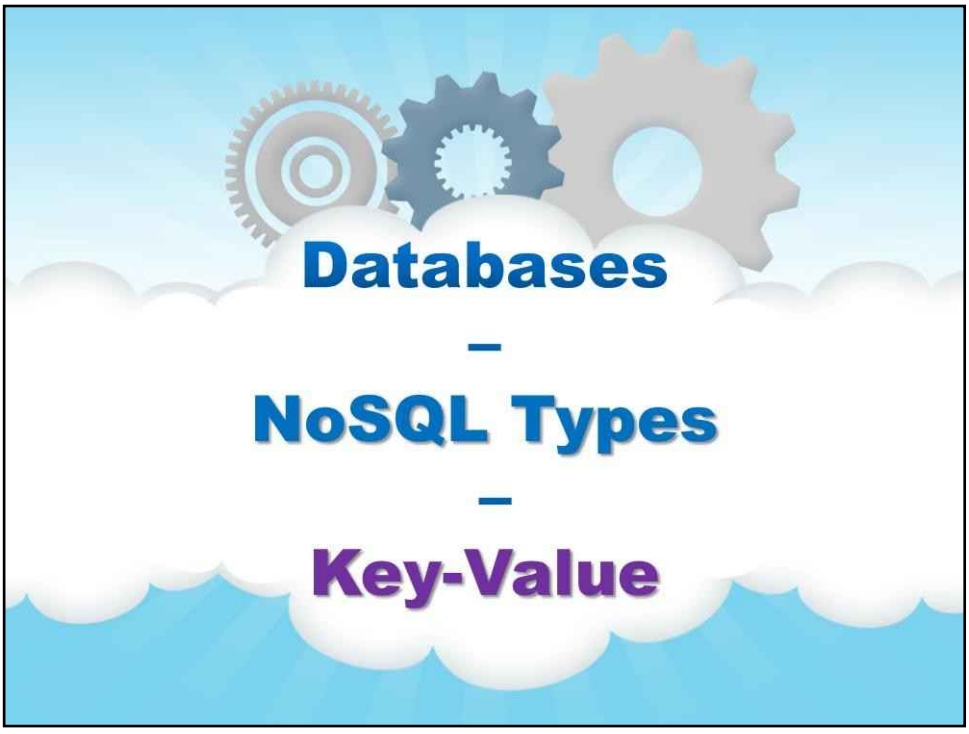


MongoDB реалізує шардинг.

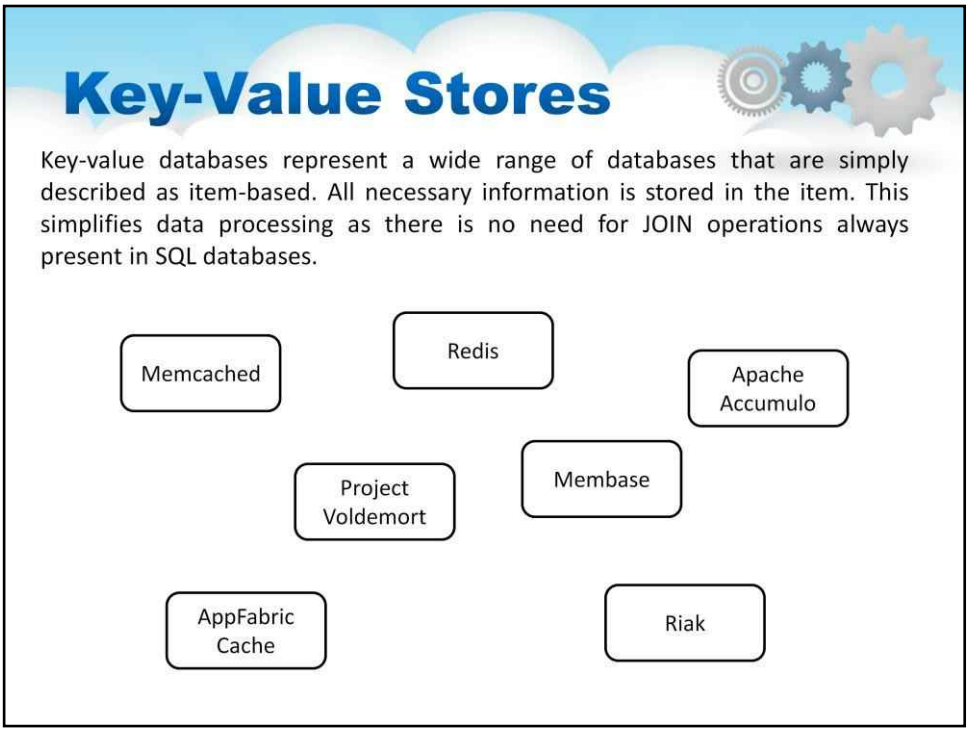
Шардований кластер MongoDB складається з таких компонентів:

- шард: кожен шард містить підмножину сегментованих даних. Кожен шард можна розгорнути як набір реплік.
- mongos: mongos діє як маршрутизатор запитів, надаючи інтерфейс між клієнтськими програмами та сегментованим кластером.
- сервери конфігурації: сервери конфігурації зберігають метадані та параметри конфігурації для кластера. Починаючи з MongoDB 3.4, конфігураційні сервери мають бути розгорнуті як набір реплік (CSRS).

Слайд містить ілюстрацію, яка показує, як це працює.



Ключ-значення



Далі ми розглянемо магазини ключ-вартість

Бази даних "ключ-значення" представляють широкий спектр баз даних, які просто описуються як бази даних на основі елементів. Вся необхідна інформація зберігається в пункті. Це спрощує обробку даних, оскільки немає необхідності в операціях JOIN, які завжди присутні в базах даних SQL.

Приклади:

Memcached – сховища ключових значень

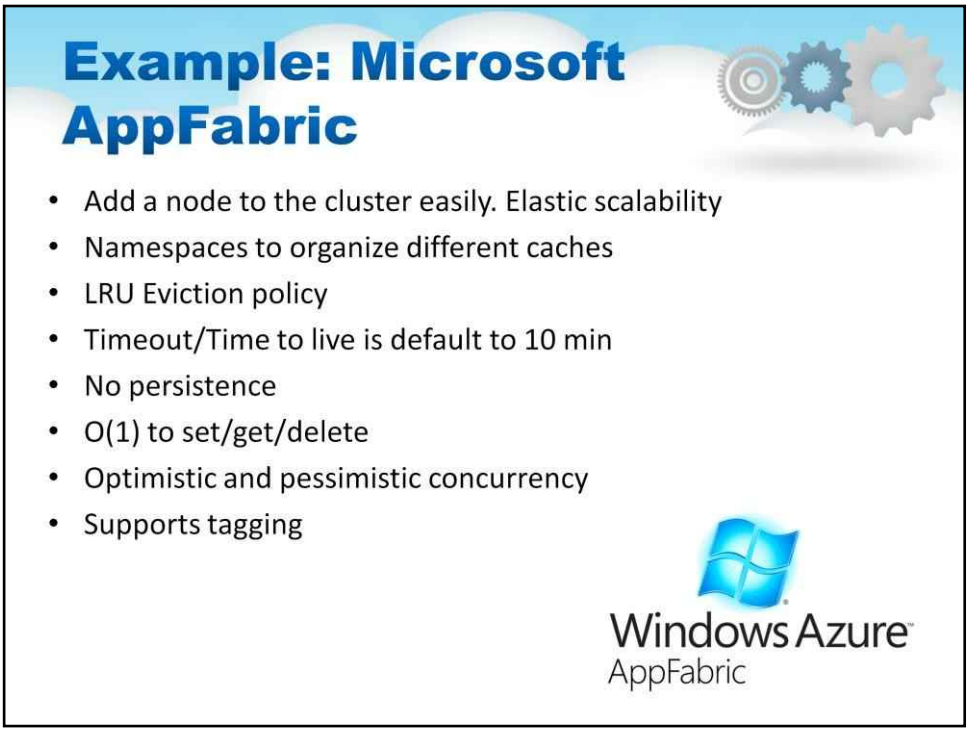
Membase – Memcached із стійкістю та покращеним узгодженим хешуванням

AppFabric Cache – Multi region Cache

Redis – Сервер структури даних Riak –

На основі Dynamo від Amazon

Project Voldemort – можливі узгоджені сховища значень ключів, автоматичне масштабування Apache Accumulo



Інший приклад: Microsoft AppFabric

Подібно до memcached:

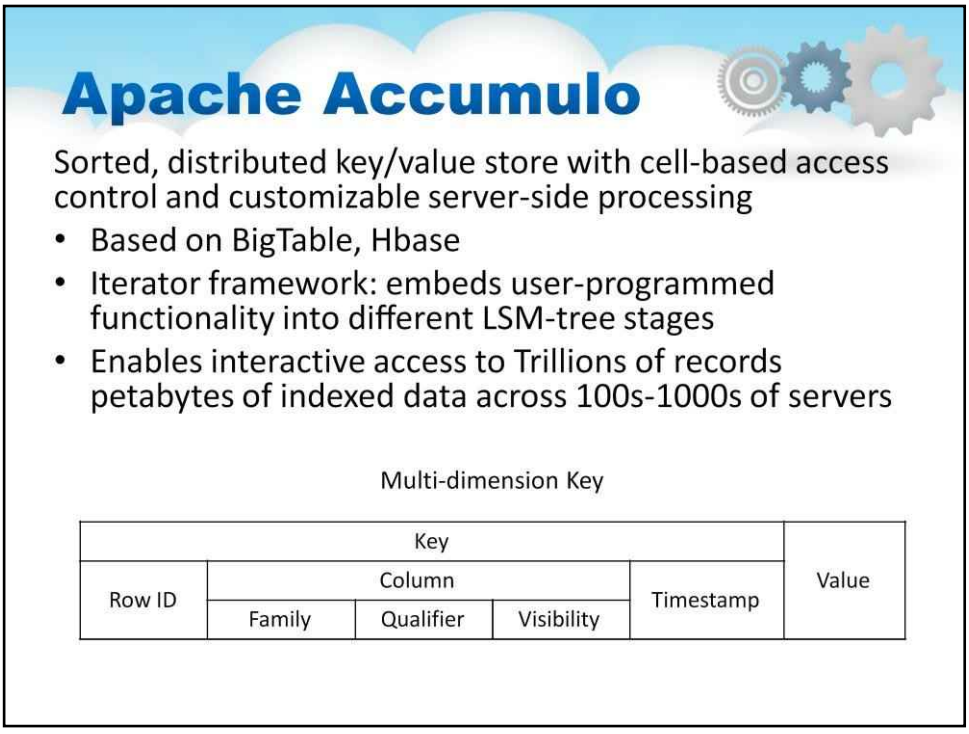
Легко додайте вузол до кластера. Еластична масштабованість

Простори імен для організації різних кешів

Політика виселення LRU

Тайм-аут/Час життя за умовчанням становить 10 хвилин

Жодної наполегливості



Apache Accumulo — ще один приклад сховища ключів,

Apache Accumulo — це відсортоване розподілене сховище ключів і значень із керуванням доступом на основі клітинок і настроюваною обробкою на стороні сервера

Розроблено NSA (Агентство національної безпеки), також відкритий код

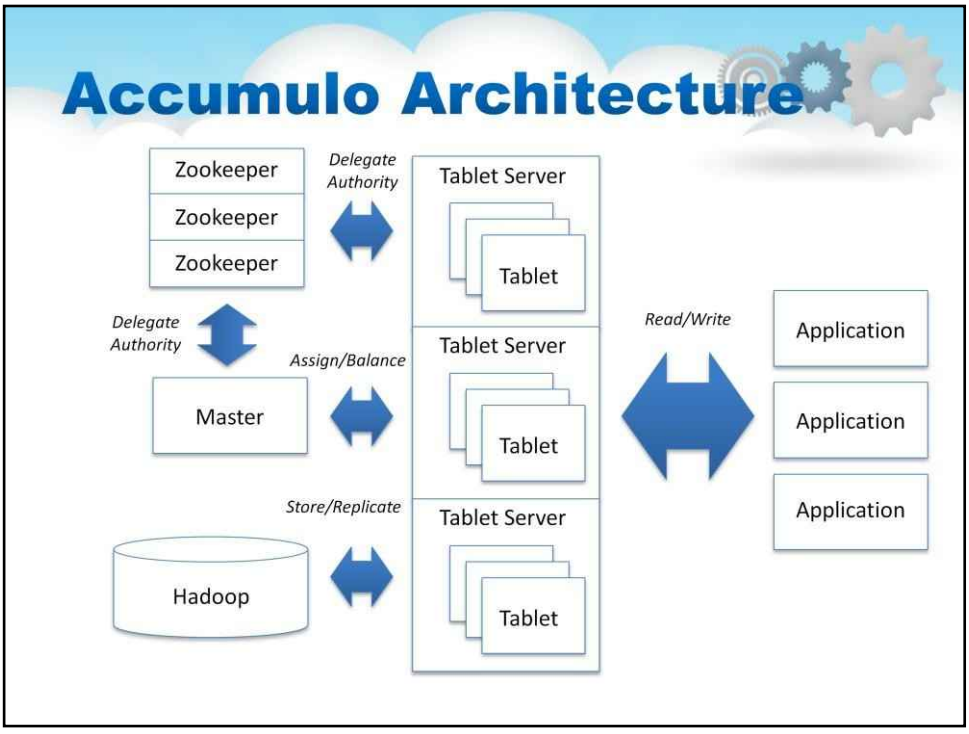
На основі BigTable, Hbase

Структура ітератора: вбудовує запрограмовану користувачем функціональність у різні етапи дерева LSM

Наприклад, ітератори можуть працювати під час незначного ущільнення, використовуючи дані memstore як вхідні дані для створення файлів зберігання на диску, які складаються з певних перетворень вхідних даних, таких як статистика або додаткові індекси

Забезпечує інтерактивний доступ до трильйонів записів петабайт індексованих даних на 100-1000 серверах

Слайд містить ілюстрацію, яка показує спосіб організації ключів, стовпців і рядків у Accumulo.



Слайд містить ілюстрацію, яка показує архітектуру Accumulo

Архітектура Accumulo складається з таких компонентів:

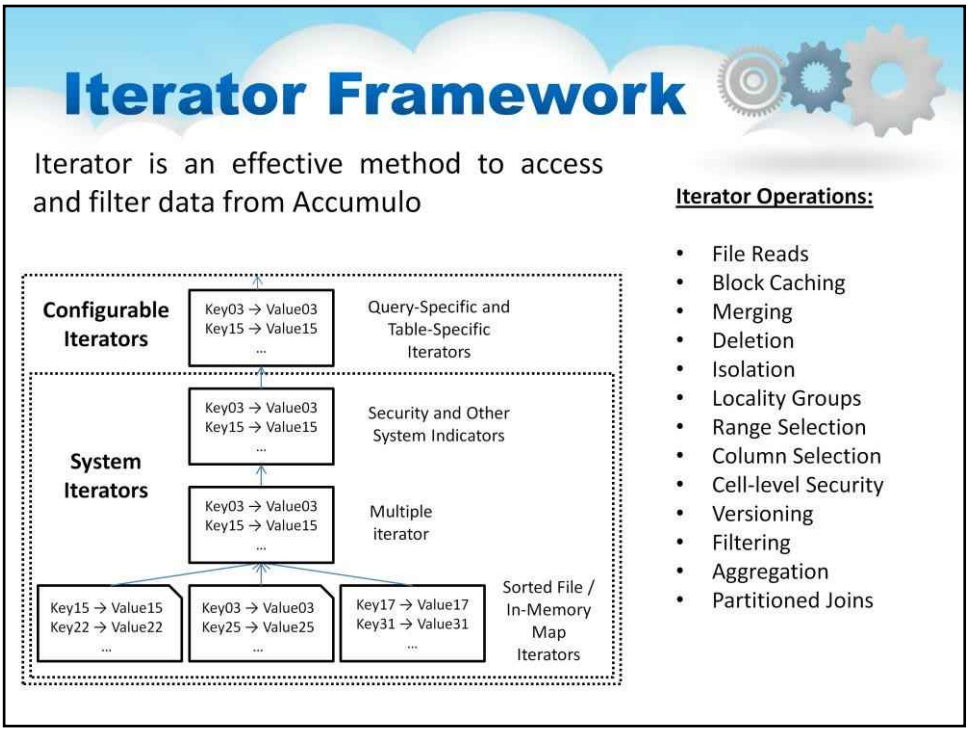
таблетки:Розділи таблиць, що складаються з відсортованих пар ключ/значення.

Сервери планшетів:Керуйте планшетами, включно з отриманням записів від клієнтів, постійними записами в журнал попереднього запису, сортуванням нових пар ключів і значень у пам'яті, періодичним використанням відсортованих пар ключів і значень до нових файлів у HDFS і реагуванням на читання від клієнтів. Під час читання сервери планшета забезпечують відсортований злиттям перегляд усіх ключів і значень зі створених ним файлів і відсортованого сховища в пам'яті.

майстер:Відповідає за виявлення збою сервера планшета та реагування на нього. Майстер намагається збалансувати навантаження на планшетні сервери, ретельно призначаючи планшети та наказуючи планшетним серверам перенести планшети, коли це необхідно. Майстер гарантує, що кожен планшет призначено точно одному серверу планшетів, і обробляє багато різноманітних запитів адміністрування бази даних. Головний також координує запуск, плавне завершення роботи та відновлення журналів попереднього запису в разі збою серверів планшета.

ZooKeeper:Розподілений запірний механізм без єдиної точки відмови. Zookeeper відповідає за підтримку конфігураційної інформації, іменування та забезпечення розподіленої синхронізації. (http://www.sqrrl.com/whitepaper/)

В основі Accumulo лежить механізм Tablet, який одночасно оптимізує низьку затримку між довільним записом і сортованим читанням (підтримка запитів у реальному часі) та ефективне використання дискового сховища.

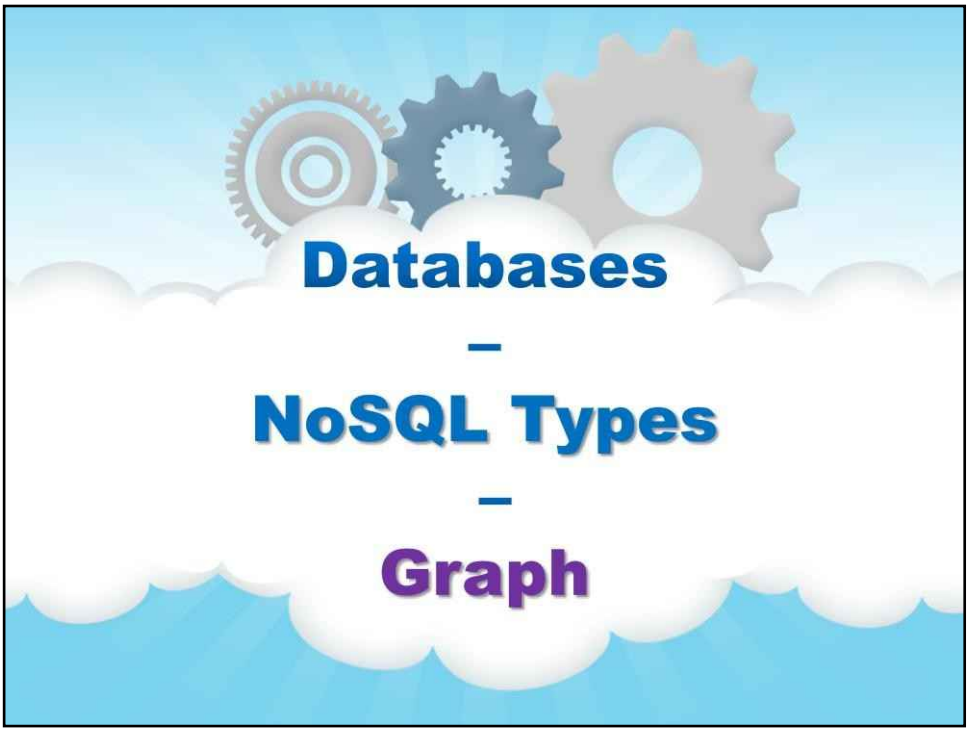


Accumulo підтримує більш розширену обробку даних, ніж просто сортування ключів і ефективний пошук.

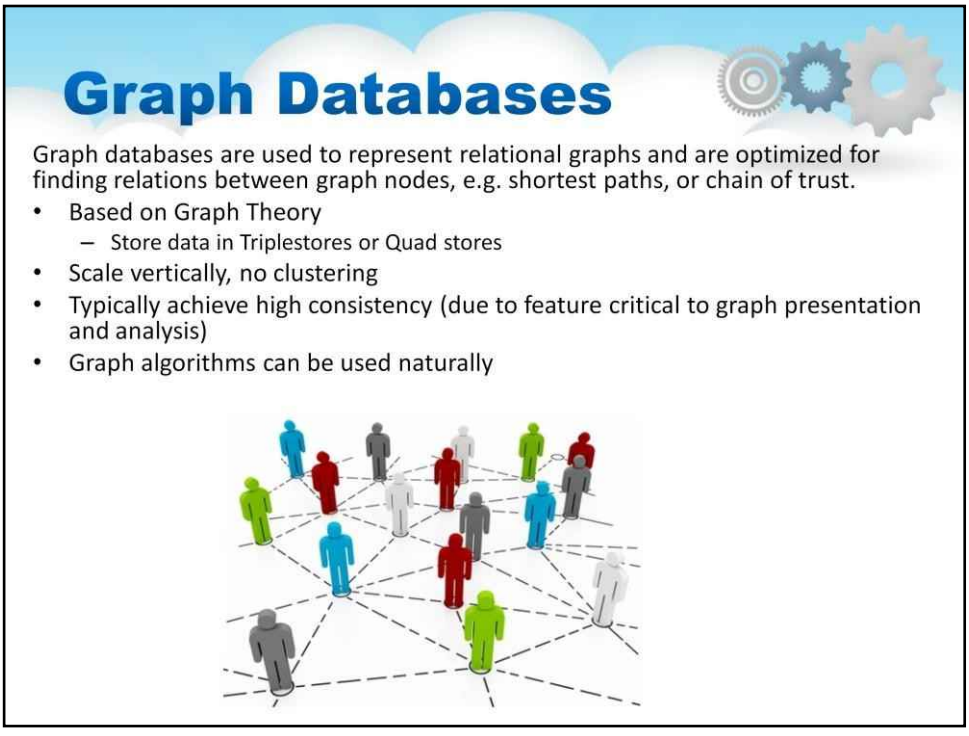
Аналітику можна розробити за допомогою MapReduce та ітераторів у поєднанні з таблицями Accumulo.

Ітератор — це ефективний метод доступу та фільтрації даних із Accumulo

Слайд містить ілюстрацію, яка показуєIterator Frameworkв Accumulo



Графік



Графові бази даних

Бази даних графів використовуються для представлення реляційних графів і оптимізовані для пошуку зв’язків між вузлами графа, наприклад, найкоротших шляхів або ланцюжка довіри.

На основі теорії графів

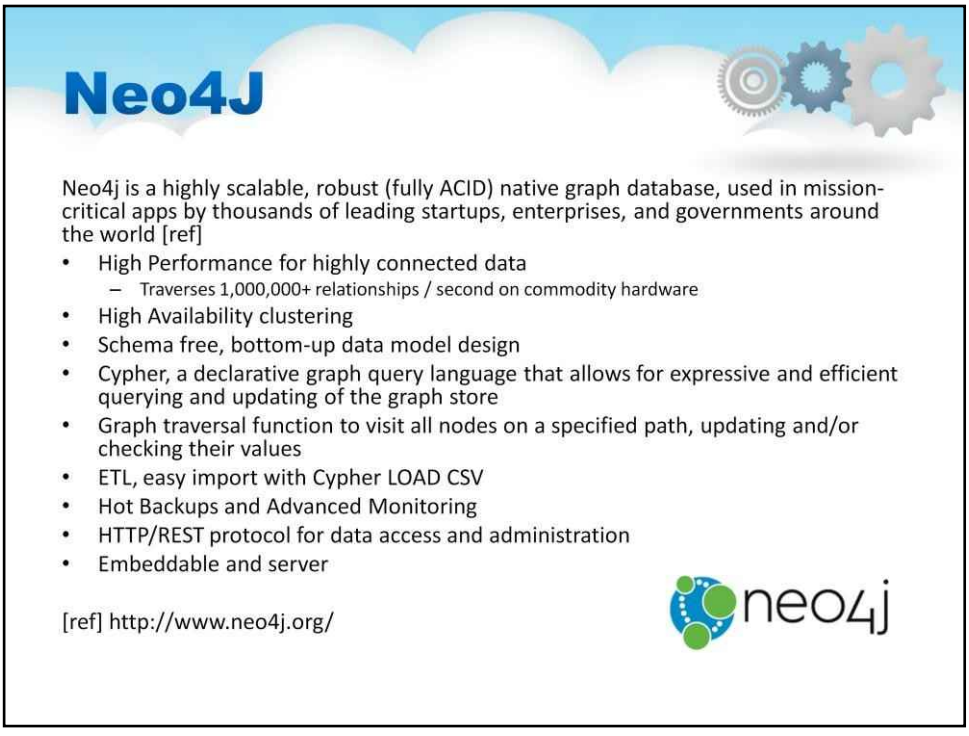
Зберігайте дані в магазинах Triplestores або Quad

Масштаб вертикально, без групування

Зазвичай досягається висока узгодженість (завдяки функції, критичній для представлення та аналізу графіка)

Графові алгоритми можна використовувати природно

Наприклад, відмінно підходить для соціальних мереж



Прикладом графової бази даних є Neo4J

Neo4j — це високомасштабована, надійна (повністю ACID) рідна база даних графів

Висока продуктивність для високопідключених даних

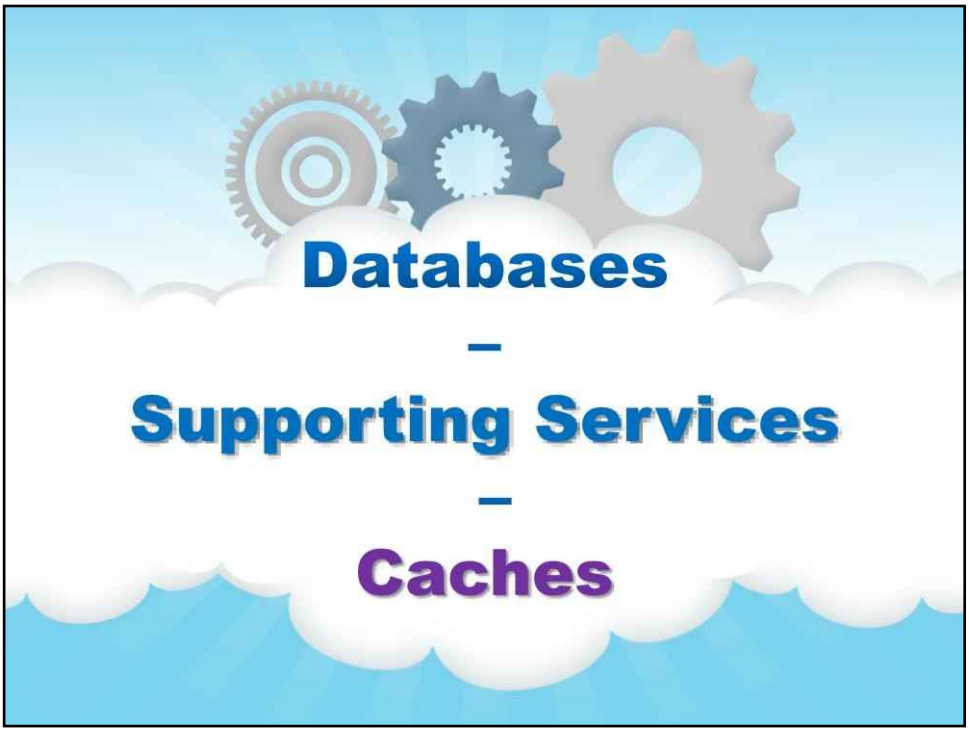
Проходить понад 1 000 000 зв’язків на секунду на стандартному обладнанні

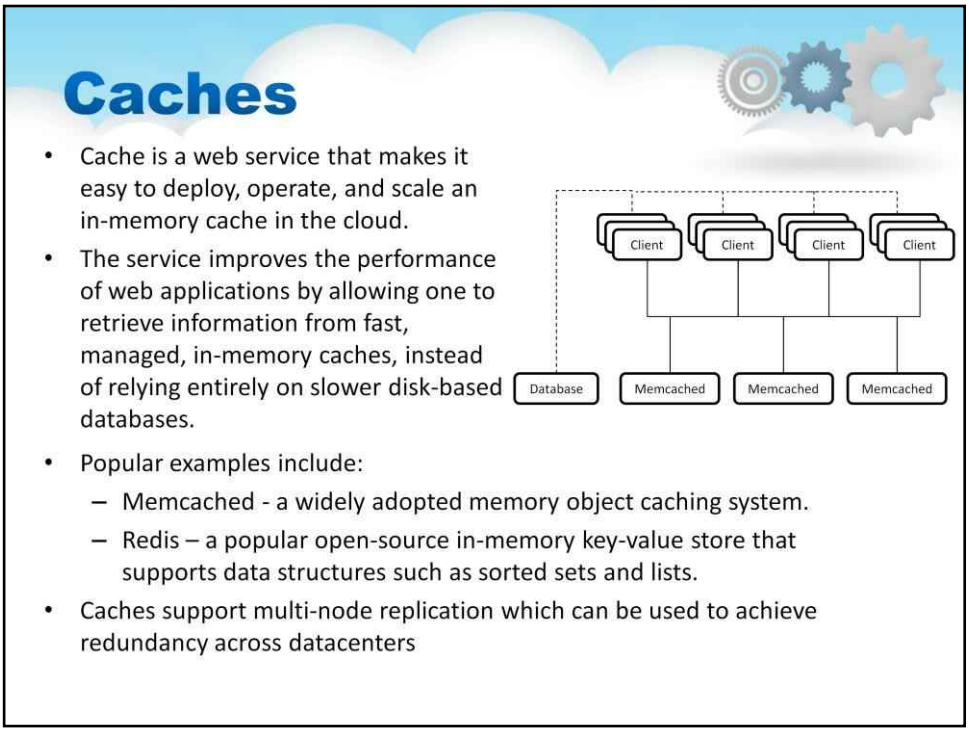
Кластеризація високої доступності

Дизайн моделі даних без схем, знизу вгору

Cypher, декларативна мова графових запитів, яка дозволяє виконувати експресивні та ефективні запити та оновлювати сховище графів

Функція обходу графіка для відвідування всіх вузлів на визначеному шляху, оновлення та/або перевірки їхніх значень


Databases
–
Supporting Services
–
Caches



Ще один важливий інструмент зберігання даних у хмарі, який за архітектурою дуже близький до бази даних, — це кеш.

Кеш — це веб-сервіс, який спрощує розгортання, роботу та масштабування кешу пам’яті в хмарі.

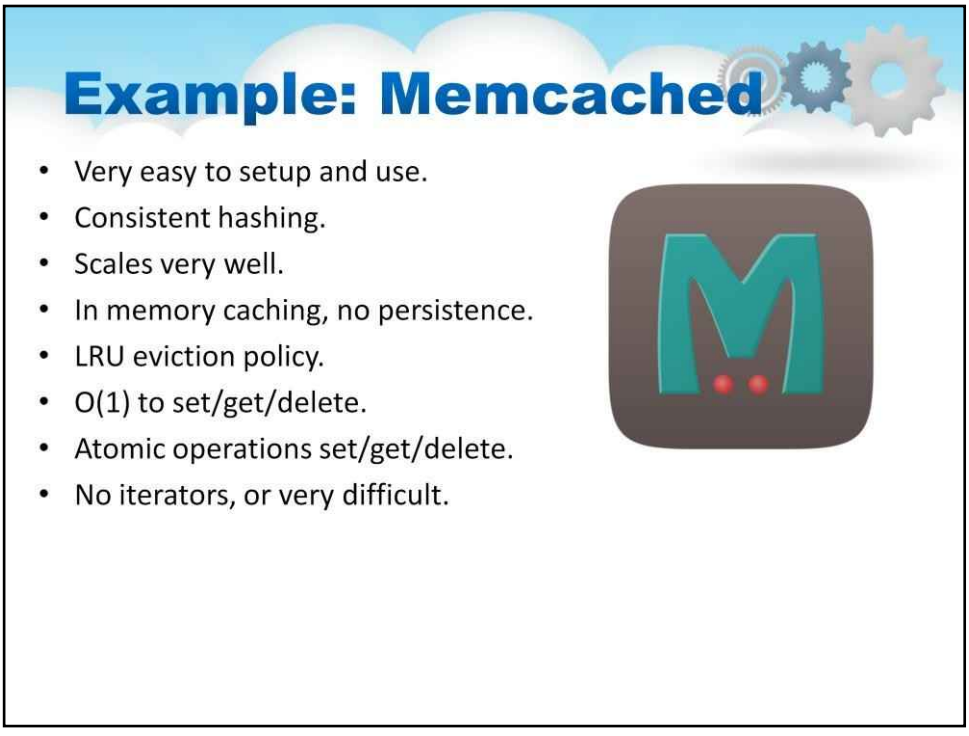
Служба покращує продуктивність веб-додатків, дозволяючи отримувати інформацію зі швидких, керованих кеш-пам’яті, замість того, щоб повністю покладатися на повільніші бази даних на диску.

Популярні приклади:

Memcached - широко поширена система кешування об'єктів пам'яті.

Redis – популярне сховище ключів і значень із відкритим кодом у пам’яті, яке підтримує такі структури даних, як відсортовані набори та списки.

Кеші підтримують багатовузлову реплікацію, яка може бути використана для досягнення надмірності в центрах обробки даних



Memcahced дуже популярний і поставляється як попередньо налаштований сервіс у багатьох хмарах.

Memcached це:

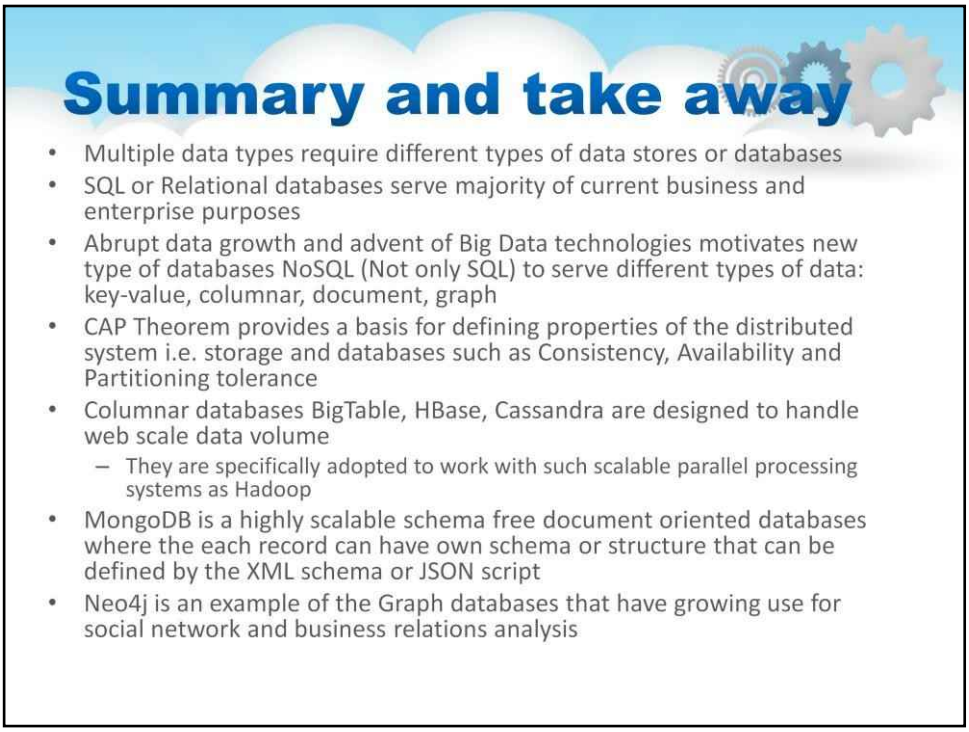
Дуже простий у налаштуванні та використанні.

Використовує послідовне хешування.

Ваги дуже добре.

Реалізує кешування пам’яті, без збереження.

Політика виселення LRU



У цій лекції ми розглянули наступні аспекти:

Кілька типів даних вимагають різних типів сховищ даних або баз даних

Бази даних SQL або реляційні бази даних служать для більшості поточних цілей бізнесу та підприємства

Різке зростання обсягу даних і поява технологій великих даних мотивує новий тип баз даних NoSQL (не тільки SQL) для обслуговування різних типів даних: ключ-значення, стовпці, документ, графік

Теорема CAP забезпечує основу для визначення властивостей розподіленої системи, тобто зберігання та баз даних, таких як узгодженість, доступність і допуск до розділення

Стовпчасті бази даних BigTable, HBase, Cassandra призначені для обробки обсягів даних веб-масштабу. Вони спеціально розроблені для роботи з такими масштабованими системами паралельної обробки, як Hadoop

MongoDB — це документально-орієнтована база даних із високою масштабованістю без схем, де кожен запис може мати власну схему чи структуру, яку можна визначити за допомогою схеми XML або сценарію JSON

Neo4j є прикладом баз даних Graph, які все частіше використовують для аналізу соціальних мереж і ділових відносин

Хмарні обчислення

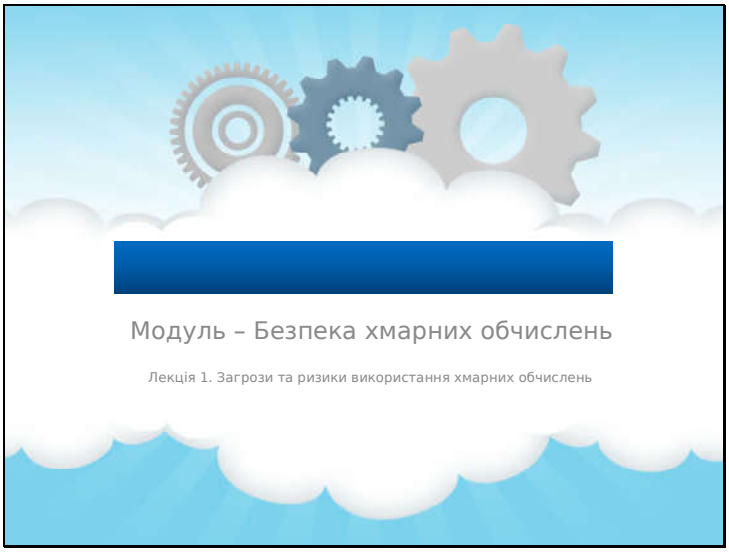
Лекційний посібник

Том 7

Модуль 7

Безпека хмарних обчислень

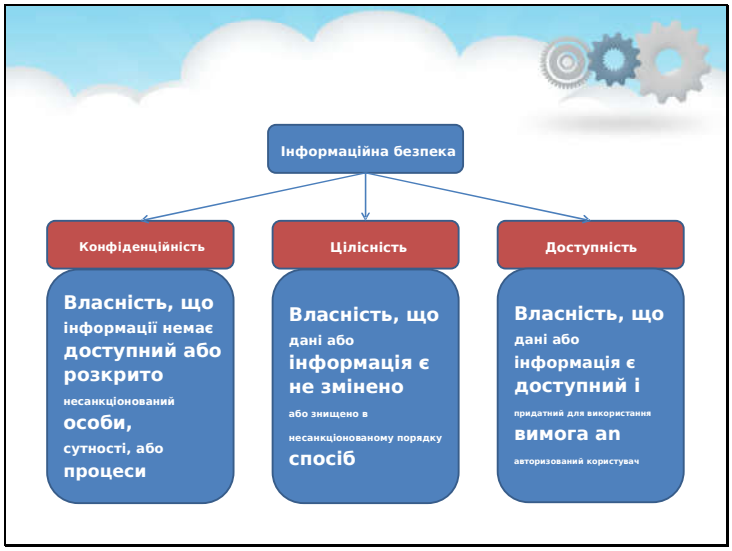
Зміст



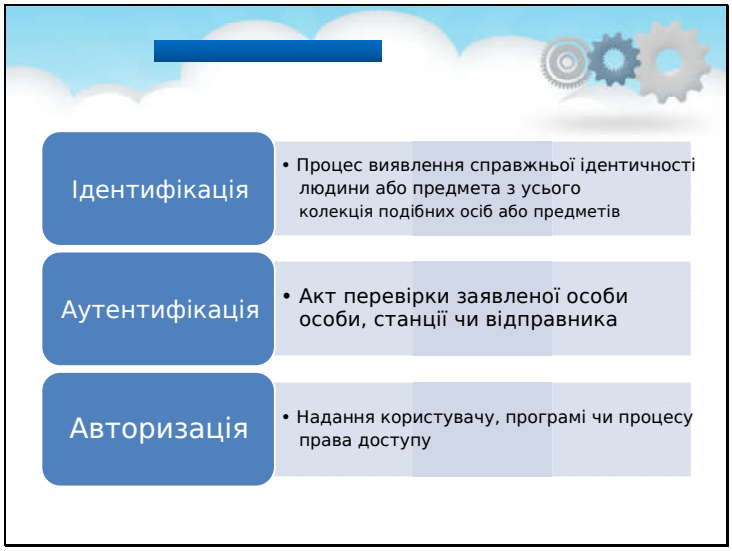
Лекція 1. Загрози і ризики використання хмарних обчислення

Огляд цієї лекції

- Ця лекція присвячена **огляд** з:
  - **базова інформація про кібербезпеку;**
  - **принципи безпеки;**
  - **загрози інформаційній безпеці;**
  - **види атак на інформаційні системи;**
  - **механізми захисту від атак.**



Основні відомості про кібербезпеку


Ідентифікація
• Процес виявлення справжньої ідентичності людини або предмета з усього колекція подібних осіб або предметів
Аутентифікація
• Акт перевірки заявленої особи особи, станції чи відправника
Авторизація
• Надання користувачу, програмі чи процесу права доступу

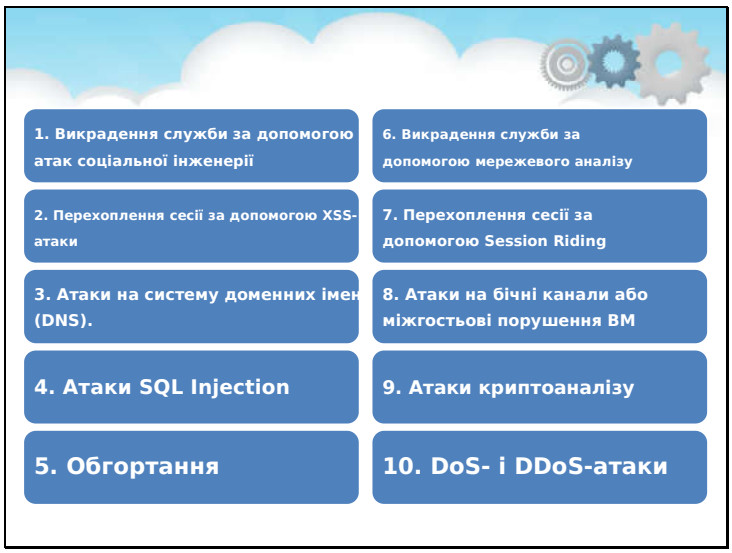

Загроза
• Потенційна причина небажаного інциденту, який може завдати шкоди інформаційній системі
Вразливість
• Слабкі місця в інформаційній системі, процедурах безпеки системи, внутрішньому контролі або реалізації, які можуть бути використані або спровоковані джерелом загрози
Атака
• Реалізація певної конкретної загрози, яка впливає на конфіденційність, цілісність, доступність обчислювального ресурсу


Інформація безпеки
• Збереження конфіденційності, цілісності та доступності інформації
Не відмова
• Можливість заборонити відправникам заперечувати, що вони надіслали повідомлення, а одержувачам — заперечувати, що вони отримали повідомлення
Ризик
• Імовірність того, що певна загроза безпеці використає вразливість системи

Загрози інформаційній безпеці


Природні загрози
• Натуральний катастрофи
• Повені
• Землетруси
• Урагани
• Пожежа
• Обладнання руйнування
фізичний Загрози безпеці
• Втрата або пошкодження система ресурси
• Фізичний вторгнення
• диверсія, шпигунство і помилки
Людські загрози
• Слабкий паролі
• Соціальний інженерія
• Відсутність знання і обізнаність
• Атака шляхом хакери


Мережеві загрози
• Спуфінг
• Нюхання або Підслуховування
• Підключення викрадення
• Людина в середня атака
• Атака на відмову в обслуговуванні
• SQL ін'єкція
• Отруєння АРП
Погрози хоста
• Атака паролів
• Шкідливе ПЗ Віруси, черви, Трояни, назад двері
• Збій обладнання через несправність
• Привілей ескалація
• Несанкціонований доступу
Програмні загрози
• Дані/Введення перевірка
• Аутентифікація і авторизація напади
• Переповнення буфера
• Криптографія напади
• Параметр маніпуляції
• Аудит і проблеми реєстрації



Атаки на хмарні обчислення

Викрадення служби за допомогою атак соціальної інженерії

Використовуючи методи соціальної інженерії, атака може спробувати вгадати пароль. Результатом атак соціальної інженерії є несанкціонований доступ із розкриттям конфіденційної інформації відповідно до рівня привілеїв скомпрометований користувач.

Викрадення служби за допомогою мережевого аналізу

Використовуючи інструменти аналізу пакетів, помістивши себе в мережу, зловмисник може захопити конфіденційна інформація, така як паролі, ідентифікатор сеансу, файли cookie та інша інформація, пов’язана з веб-службою, як-от UDDI, SOAP і WSDL

Викрадення сесії за допомогою XSS-атаки

Запустивши міжсайтовий сценарій (XSS), зловмисник може викрасти файли cookie, вставивши шкідливий код на веб-сайт.

Викрадення сесії за допомогою Session Riding

Session Riding призначений для викрадення сесії. Зловмисник може використати це, намагаючись підробка міжсайтового запиту. Зловмисник використовує поточний активний сеанс і проходить його виконання таких запитів, як зміна даних, видалення даних, онлайн-транзакції тощо

зміна пароля шляхом відстеження користувача за натисканням шкідливого посилання.

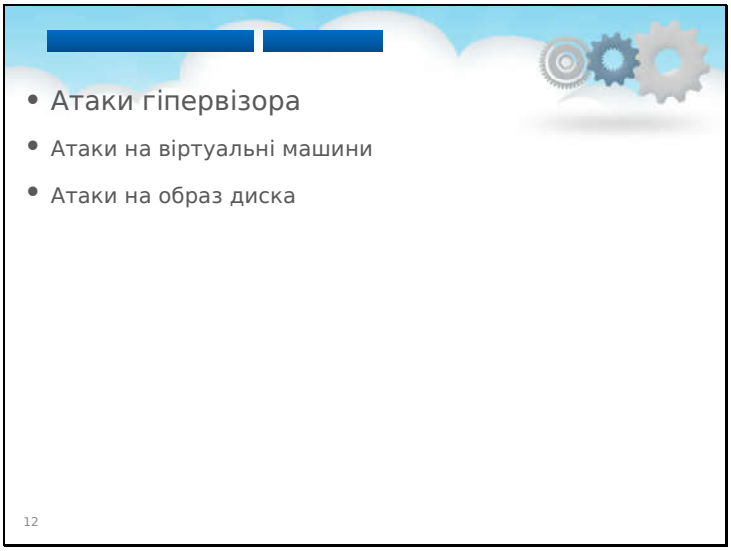
Атаки на систему доменних імен (DNS).
Атаки на систему доменних імен (DNS) включають DNS Poisoning, Cybersquatting, Domain
викрадення та розділення домену. Зловмисник може спробувати підробити, отруївши DNS
сервер або кеш для отримання облікових даних внутрішніх користувачів. Викрадення домену передбачає крадіжку
доменне ім’я хмарної служби. Подібним чином через фішингові шахрайства користувачі можуть бути перенаправлені на
фальшивий сайт.

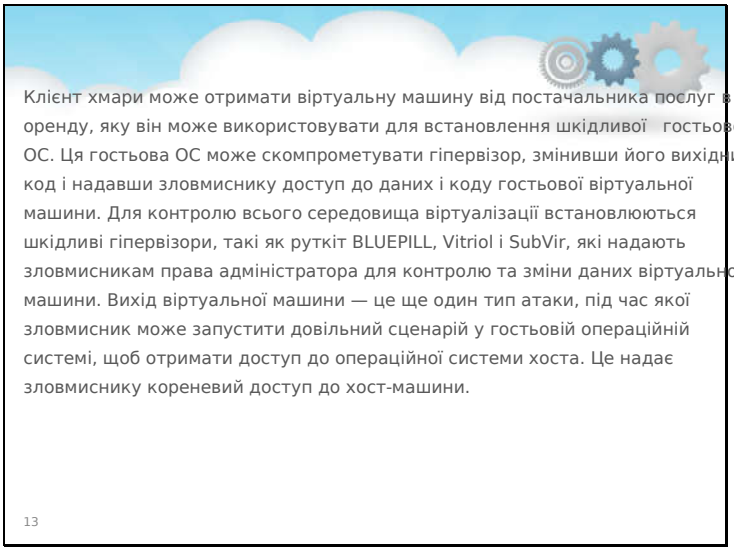
Атаки на боковий канал або злами міжгостьових віртуальних машин
Атаки на бічні канали або Cross-Guest VM Breach — це атака, яка потребує розгортання зловмисної віртуальної машини на тому самому хості. Наприклад, фізичний хост
розміщує віртуальну машину, яка пропонує хмарні послуги, отже, мета
нападник. Зловмисник встановить шкідливу віртуальну машину на той самий хост, щоб взяти його
перевага спільного використання ресурсів одного хоста, таких як кеш процесора, криптографічний
ключі тощо. Встановлення може здійснити зловмисний інсайдер або зловмисник, видавши себе за іншу особу
законний користувач.

Подібним чином є й інші зловмисники, про яких йшлося раніше, які також уразливі до Cloud
Обчислення, наприклад атака SQL Injection (введення зловмисних операторів SQL для вилучення
інформація), атаки криптоаналізу (слабке або застаріле шифрування) атаки загортання
(дублювання тіла повідомлення), атаки на відмову в обслуговуванні (DoS) і атаки на розподілену відмову в обслуговуванні (DDoS).

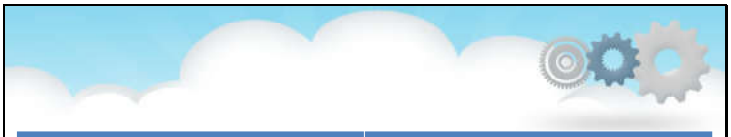
Хмарні обчислення стають популярними серед IT-бізнесу завдяки гнучким, гнучким і економічно ефективним послугам. Віртуалізація є ключовим аспектом хмарних обчислень і основою надання послуг рівня інфраструктури орендарям.

Ми описуємо різні типи віртуалізації та проблеми безпеки в хмарних компонентах віртуалізації, таких як гіпервізор, віртуальні машини та образи гостьових дисків. Аналіз безпеки віртуалізації охоплює атаки на компоненти віртуалізації в хмарі, рішення безпеки для компонентів віртуалізації, надані в літературі, і рекомендації безпеки для віртуалізації середовище, яке може бути корисним для хмарних адміністраторів.

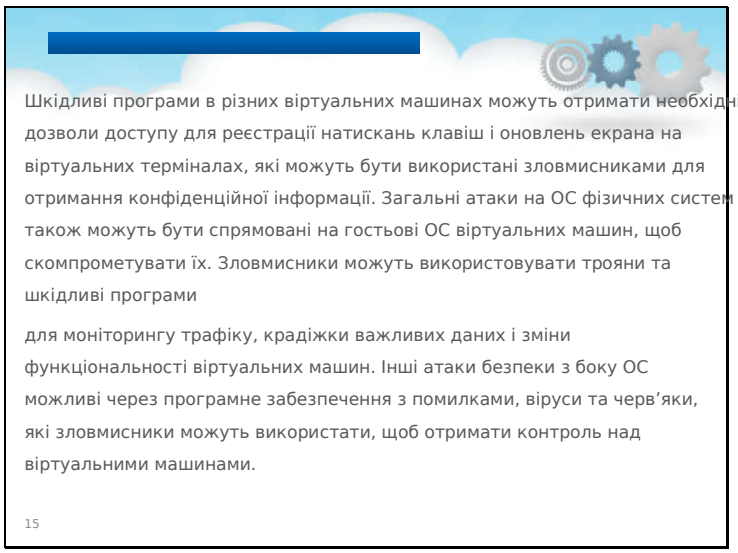

• Атаки гіпервізора
• Атаки на віртуальні машини
• Атаки на образ диска
12



Клієнт хмари може отримати віртуальну машину від постачальника послуг в оренду, яку він може використовувати для встановлення шкідливої гостьової ОС. Ця гостьова ОС може скомпрометувати гіпервізор, змінивши його вихідний код і надавши зловмиснику доступ до даних і коду гостьової віртуальної машини. Для контролю всього середовища віртуалізації встановлюються шкідливі гіпервізори, такі як руткіт BLUEPILL, Vitriol і SubVir, які надають зловмисникам права адміністратора для контролю та зміни даних віртуальної машини. Вихід віртуальної машини — це ще один тип атаки, під час якої зловмисник може запустити довільний сценарій у гостьовій операційній системі, щоб отримати доступ до операційної системи хоста. Це надає зловмиснику кореневий доступ до хост-машини.

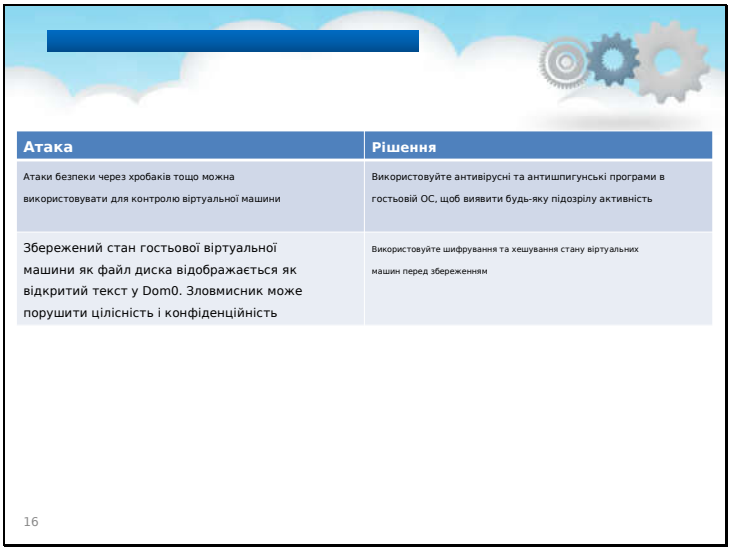


Атака	Рішення
Атака VM Escape	Правильно налаштуйте взаємодію між гостьовими машинами та віртуальною машиною хоста
Клієнти можуть орендувати гостьову віртуальну машину для встановлення шкідливої гостьової ОС	Віртуальні машини можна захистити від скомпрометованого гіпервізора шляхом шифрування VMsR
Атаки HyperJacking, BLUEPILL, Vitriol, SubVirt і DKSM	Віртуальні машини можна захистити від скомпрометованого гіпервізора шляхом шифрування VMsR
Збільшення розміру коду призвело до вразливостей дизайну та впровадження	Hypersafe — це система, яка підтримує цілісність гіпервізора

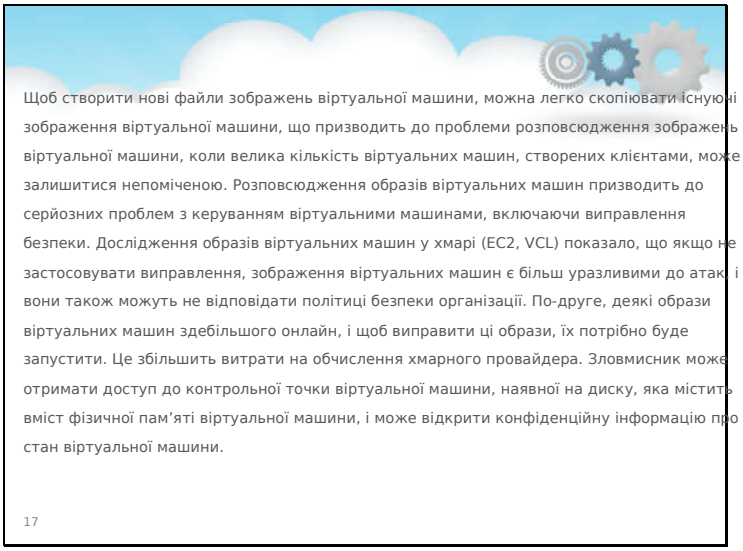


Шкідливі програми в різних віртуальних машинах можуть отримати необхідні дозволи доступу для реєстрації натискань клавіш і оновлень екрана на віртуальних терміналах, які можуть бути використані зловмисниками для отримання конфіденційної інформації. Загальні атаки на ОС фізичних систем також можуть бути спрямовані на гостьові ОС віртуальних машин, щоб скомпрометувати їх. Зловмисники можуть використовувати трояни та шкідливі програми

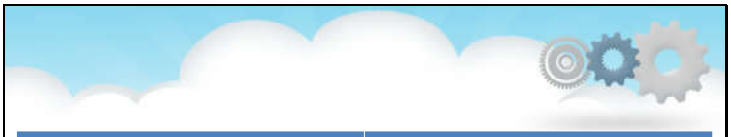
для моніторингу трафіку, крадіжки важливих даних і зміни функціональності віртуальних машин. Інші атаки безпеки з боку ОС можливі через програмне забезпечення з помилками, віруси та черв’яки, які зловмисники можуть використати, щоб отримати контроль над віртуальними машинами.



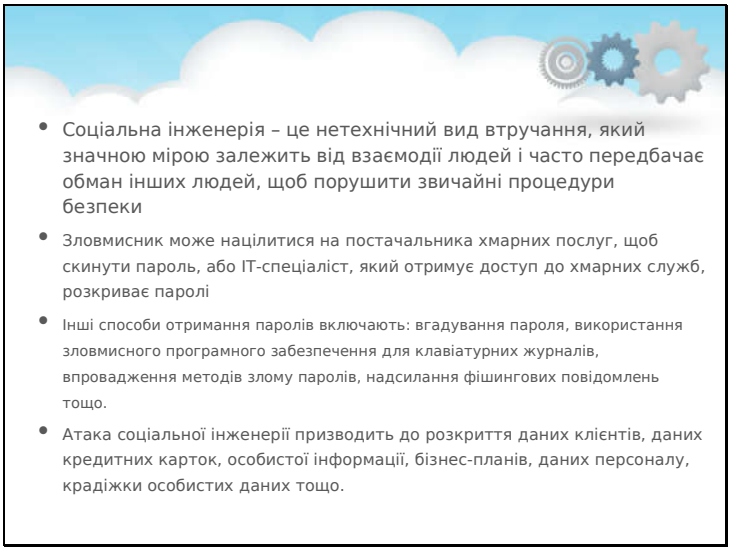
Атака	Рішення
Атаки безпеки через хробаків тощо можна використовувати для контролю віртуальної машини	Використовуйте антивірусні та антишпигунські програми в гостьовій ОС, щоб виявити будь-яку підозрілу активність
Збережений стан гостьової віртуальної машини як файл диска відображається як відкритий текст у Dom0. Зловмисник може порушити цілісність і конфіденційність	Використовуйте шифрування та хешування стану віртуальних машин перед збереженням



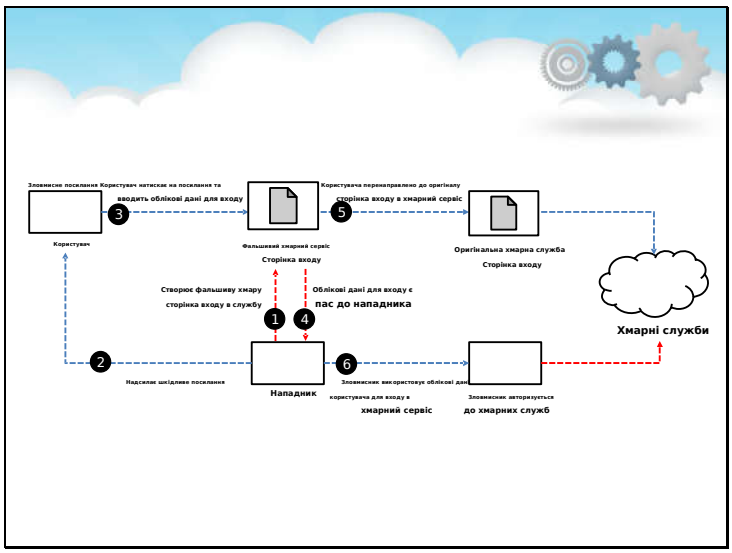
Щоб створити нові файли зображень віртуальної машини, можна легко скопіювати існуючі зображення віртуальної машини, що призводить до проблеми розповсюдження зображень віртуальної машини, коли велика кількість віртуальних машин, створених клієнтами, може залишитися непоміченою. Розповсюдження образів віртуальних машин призводить до серйозних проблем з керуванням віртуальними машинами, включаючи виправлення безпеки. Дослідження образів віртуальних машин у хмарі (EC2, VCL) показало, що якщо не застосовувати виправлення, зображення віртуальних машин є більш уразливими до атак, і вони також можуть не відповідати політиці безпеки організації. По-друге, деякі образи віртуальних машин здебільшого онлайн, і щоб виправити ці образи, їх потрібно буде запустити. Це збільшить витрати на обчислення хмарного провайдера. Зловмисник може отримати доступ до контрольної точки віртуальної машини, наявної на диску, яка містить вміст фізичної пам’яті віртуальної машини, і може відкрити конфіденційну інформацію про стан віртуальної машини.

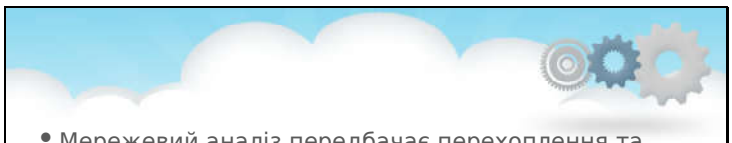


Атака	Рішення
Атаки на контрольні точки ВМ	Атаки на контрольні точки можна запобігти шляхом шифрування контрольних точок або використання SPARCR
Непотрібні зображення можуть призвести до порушення безпеки	Організації, які використовують віртуалізацію, повинні мати політику, яка керує проблемами непотрібних зображень



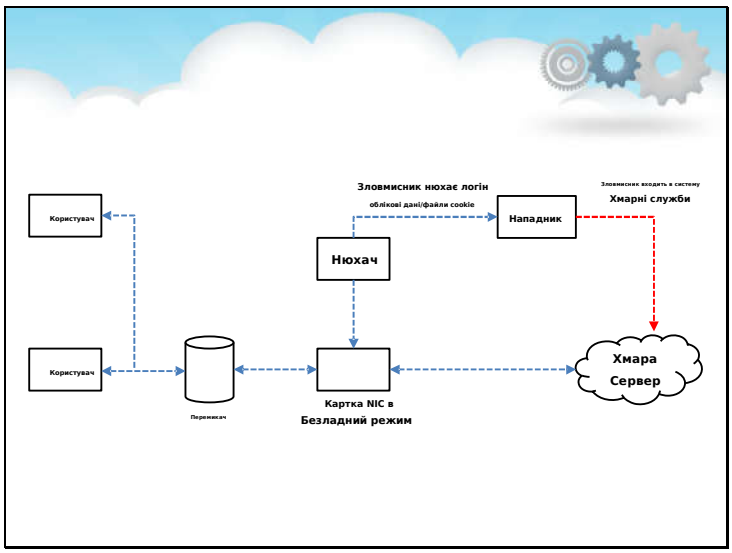
- Соціальна інженерія – це нетехнічний вид втручання, який значною мірою залежить від взаємодії людей і часто передбачає обман інших людей, щоб порушити звичайні процедури безпеки
- Зловмисник може націлитися на постачальника хмарних послуг, щоб скинути пароль, або IT-спеціаліст, який отримує доступ до хмарних служб, розкриває паролі
- Інші способи отримання паролів включають: вгадування пароля, використання зловмисного програмного забезпечення для клавіатурних журналів, впровадження методів злому паролів, надсилання фішингових повідомлень тощо.
- Атака соціальної інженерії призводить до розкриття даних клієнтів, даних кредитних карток, особистої інформації, бізнес-планів, даних персоналу, крадіжки особистих даних тощо.

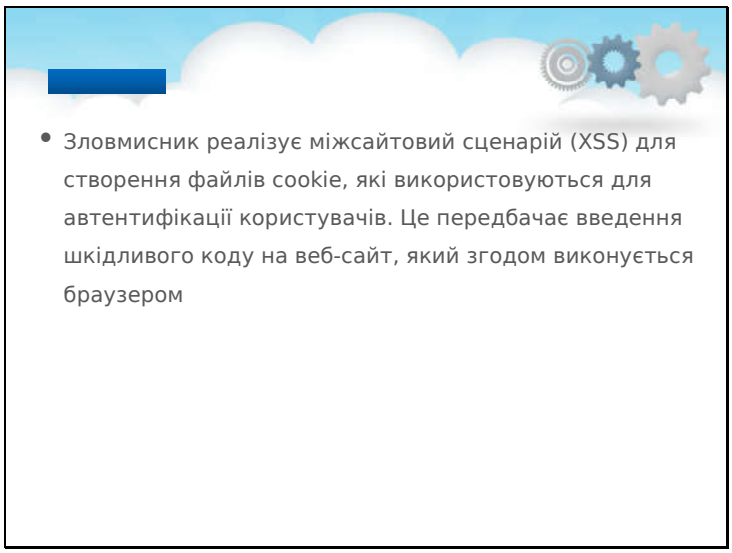

Зловмисне посилання
Користувач натискає на посилання та
вводить облікові дані для входу
Користувача перенаправлено до оригіналу
сторінка входу в хмарний сервіс
Користувач
Фальшивий хмарний сервіс
Сторінка входу
Оригінальна хмарна служба
Сторінка входу
Створює фальшиву хмару
сторінка входу в службу
Облікові дані для входу є
пас до нападника
Хмарні служби
Надсилає шкідливе посилання
Нападник
Зловмисник використовує облікові дані
користувача для входу в
хмарний сервіс
Зловмисник авторизується
до хмарних служб



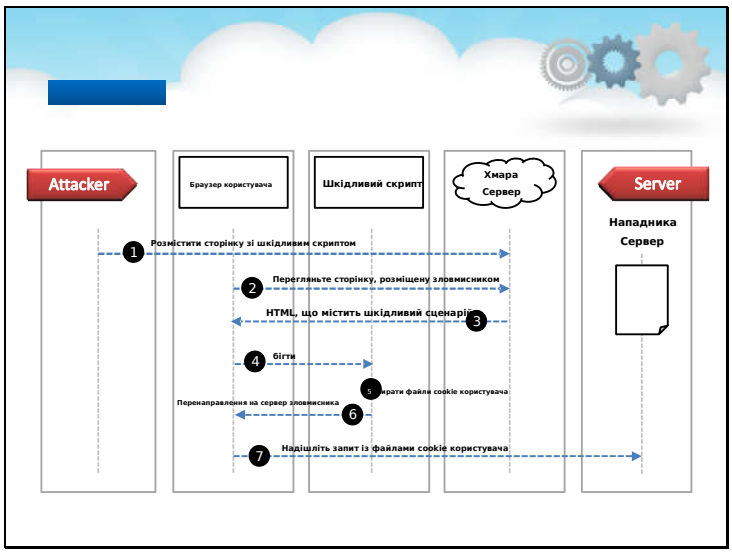
- Мережевий аналіз передбачає перехоплення та моніторинг мережевого трафіку, який надсилається між двома хмарними вузлами
- Зловмисник використовує сніфери пакетів для захоплення конфіденційних даних, таких як паролі, файли cookie сеансу та інших конфігурацій безпеки, пов’язаних із веб-службами, таких як UDDI (універсальний опис).

Файли Discovery and Integrity), SOAP (Simple Object Access Protocol) і WSDL (Web Service Description Language).

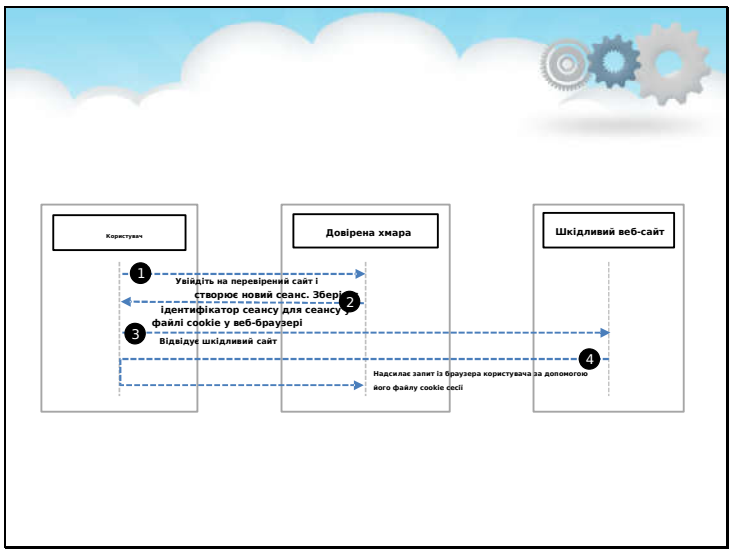

Користувач
Користувач
Перемикач
Нюхач
Зловмисник нюхає логін
облікові дані/файли cookie
Нападник
Зловмисник входить в систему
Хмарні служби
Картка NIC в
Безладний режим
Хмара
Сервер

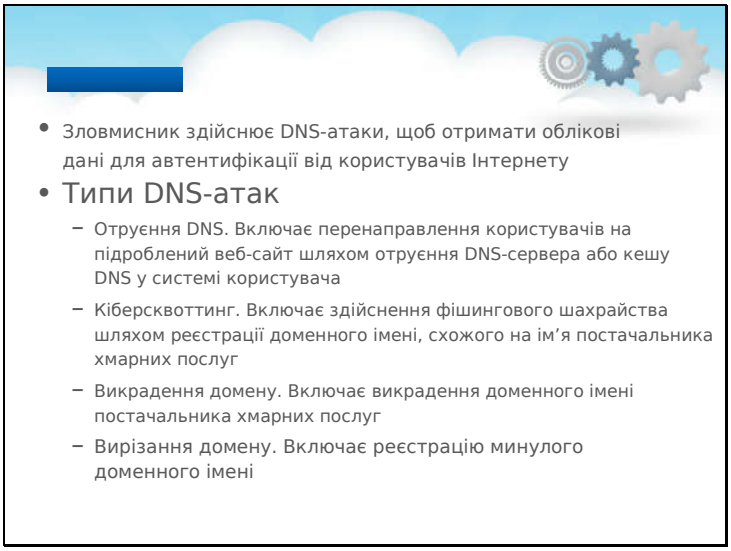


- Зловмисник реалізує міжсайтовий сценарій (XSS) для створення файлів cookie, які використовуються для автентифікації користувачів. Це передбачає введення шкідливого коду на веб-сайт, який згодом виконується браузером

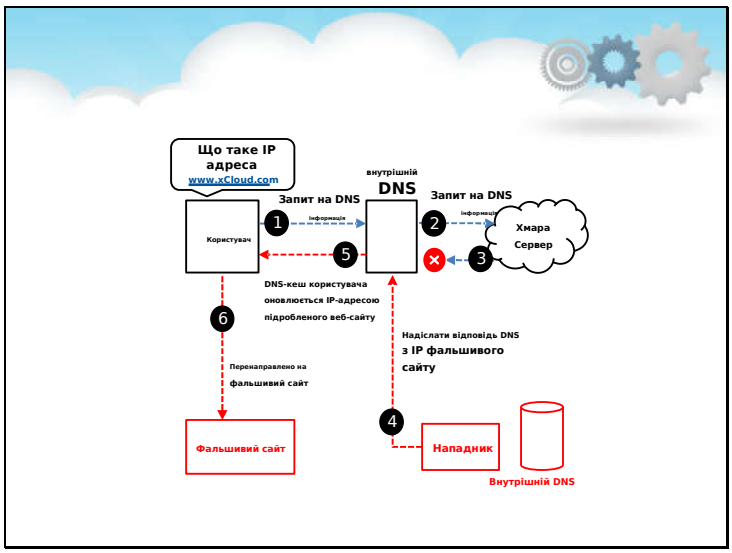

Attacker
Браузер користувача
Шкідливий скрипт
Хмара
Сервер
Server
Нападника
Сервер
1
Розмістити сторінку зі шкідливим скриптом
2
Перегляньте сторінку, розміщену зловмисником
HTML, що містить шкідливий сценарій
3
4
бігти
5
крати файли cookie користувача
Перенаправлення на сервер зловмисника
6
7
Надішліть запит із файлами cookie користувача

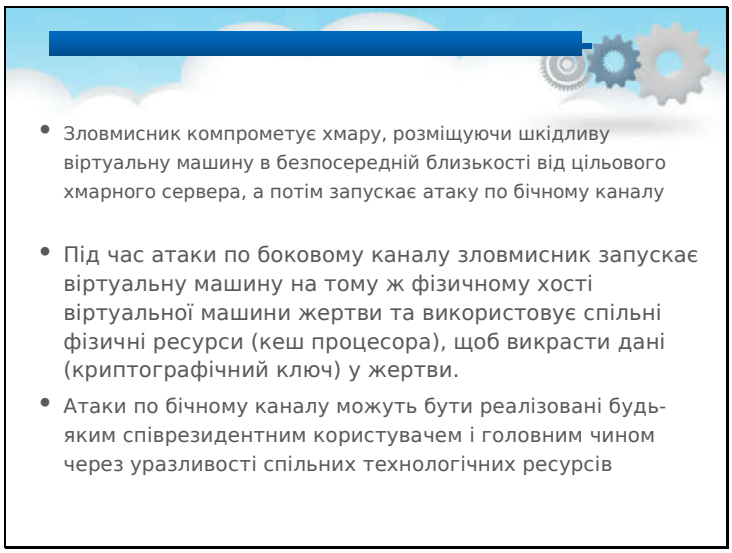
- Зловмисник використовує веб-сайт, застосовуючи підробку міжсайтових запитів для передачі неавторизованих команд
- Під час керування сеансом зловмисник керує активним сеансом комп'ютера, надсилаючи електронний лист або обманом змушуючи користувача відвідати шкідливу веб-сторінку, коли він увійшов на цільовий сайт.
- Коли користувач натискає шкідливе посилання, веб-сайт виконує запит, оскільки користувач уже автентифікований
- Використовувані команди включають: змінювати або видаляти дані користувача, виконувати онлайн-транзакції, скидати паролі тощо.


Користувач
Довірена хмара
Шкідливий веб-сайт
1
Увійдіть на перевірений сайт і
створює новий сеанс. Збер
2
ідентифікатор сеансу для сеансу у
3
файлі cookie у веб-браузері
Відвідує шкідливий сайт
4
Надсилає запит із браузера користувача за допомогою
його файлу cookie cecii

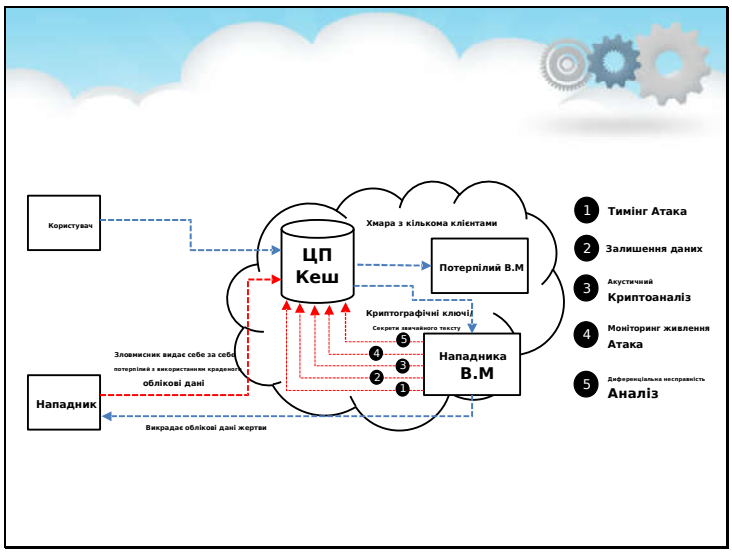


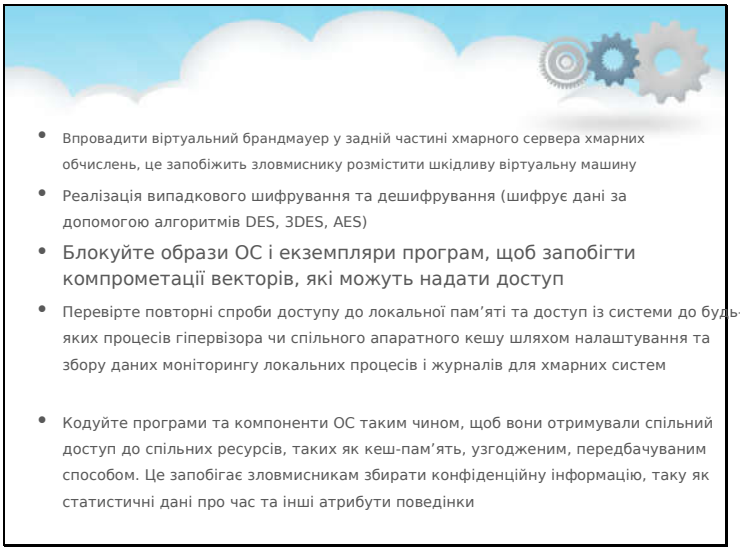
- Зловмисник здійснює DNS-атаки, щоб отримати облікові дані для автентифікації від користувачів Інтернету
- Типи DNS-атак
  - Отруєння DNS. Включає перенаправлення користувачів на підроблений веб-сайт шляхом отруєння DNS-сервера або кешу DNS у системі користувача
  - Кіберсквоттинг. Включає здійснення фішингового шахрайства шляхом реєстрації доменного імені, схожого на ім’я постачальника хмарних послуг
  - Викрадення домену. Включає викрадення доменного імені постачальника хмарних послуг
  - Вирізання домену. Включає реєстрацію минулого доменного імені


Що таке IP адреса
www.xCloud.com
внутрішній
DNS
Запит на DNS
Запит на DNS
Користувач
Інформація
Інформація
Хмара
Сервер
1
2
3
5
6
DNS-кеш користувача оновлюється IP-адресою підробленого веб-сайту
Надіслати відповідь DNS з IP фальшивого сайту
Перенаправлено на фальшивий сайт
4
Фальшивий сайт
Нападник
Внутрішній DNS



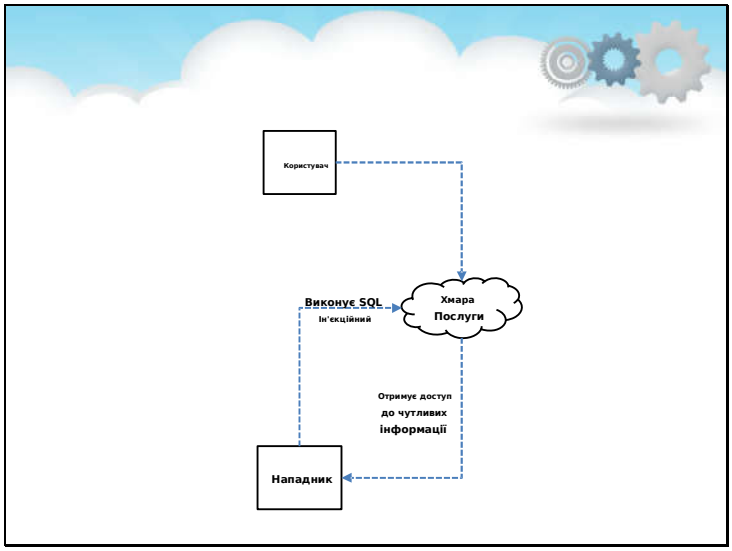
- Зловмисник компрометує хмару, розміщуючи шкідливу віртуальну машину в безпосередній близькості від цільового хмарного сервера, а потім запускає атаку по бічному каналу
- Під час атаки по боковому каналу зловмисник запускає віртуальну машину на тому ж фізичному хості віртуальної машини жертви та використовує спільні фізичні ресурси (кеш процесора), щоб викрасти дані (криптографічний ключ) у жертви.
- Атаки по бічному каналу можуть бути реалізовані будь-яким співрезидентним користувачем і головним чином через уразливості спільних технологічних ресурсів


Користувач
Нападник
ЦП
Кеш
Хмара з кількома клієнтами
Потерпілий В.М
Криптографічні ключі
Секрети звичайного тексту
Нападника
В.М
Зловмисник видає себе за себе
потерпілий з використанням крадених
облікові дані
Викрадає облікові дані жертви
1 Тимінг Атака
2 Залишення даних
3 Акустичний
Криптоаналіз
4 Моніторинг живлення
Атака
5 Диференціальна несправність
Аналіз

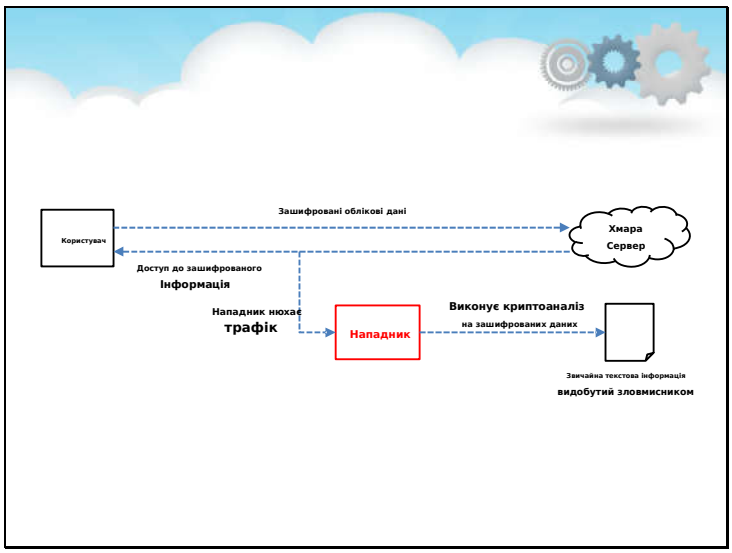


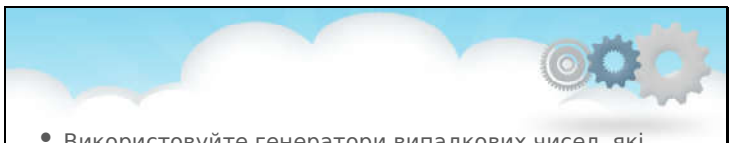
- Впровадити віртуальний брандмауер у задній частині хмарного сервера хмарних обчислень, це запобіжить зловмиснику розмістити шкідливу віртуальну машину
- Реалізація випадкового шифрування та дешифрування (шифрує дані за допомогою алгоритмів DES, 3DES, AES)
- Блокуйте образи ОС і екземпляри програм, щоб запобігти компрометації векторів, які можуть надати доступ
- Перевірте повторні спроби доступу до локальної пам’яті та доступ із системи до будь-яких процесів гіпервізора чи спільного апаратного кешу шляхом налаштування та збору даних моніторингу локальних процесів і журналів для хмарних систем
- Кодуйте програми та компоненти ОС таким чином, щоб вони отримували спільний доступ до спільних ресурсів, таких як кеш-пам’ять, узгодженим, передбачуваним способом. Це запобігає зловмисникам збирати конфіденційну інформацію, таку як статистичні дані про час та інші атрибути поведінки

- Зловмисники атакують SQL-сервери, на яких працюють уразливі програми баз даних
- Зазвичай це відбувається, коли програма використовує вхідні дані для створення динамічних операторів SQL
- У цій атаці зловмисники вставляють шкідливий код (створений за допомогою спеціальних символів) у стандартний код SQL, щоб отримати несанкціонований доступ до бази даних.
- Далі зловмисники можуть маніпулювати вмістом бази даних, отримувати конфіденційні дані, віддалено виконувати системні команди або навіть взяти під контроль веб-сервер для подальшої злочинної діяльності

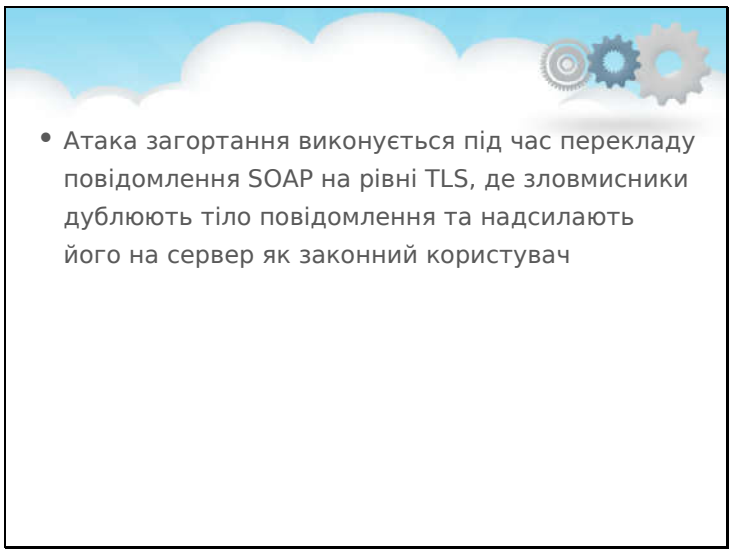

Користувач
Виконує SQL
Ін'єкційний
Хмара
Послуги
Отримує доступ
до чутливих
інформації
Нападник

- Ненадійне або застаріле шифрування робить хмарні служби чутливими до криптоаналізу
- Дані, наявні в хмарі, можуть бути зашифровані, щоб запобігти їх читанню в разі доступу зловмисників. Однак критичні недоліки в реалізації криптографічного алгоритму (наприклад: слабка генерація випадкових чисел) можуть перетворити надійне шифрування на слабке або зламане, також існують нові методи зламу криптографії
- Часткову інформацію також можна отримати із зашифрованих даних шляхом моніторингу шаблонів доступу до запитів клієнтів і аналізу доступних позицій

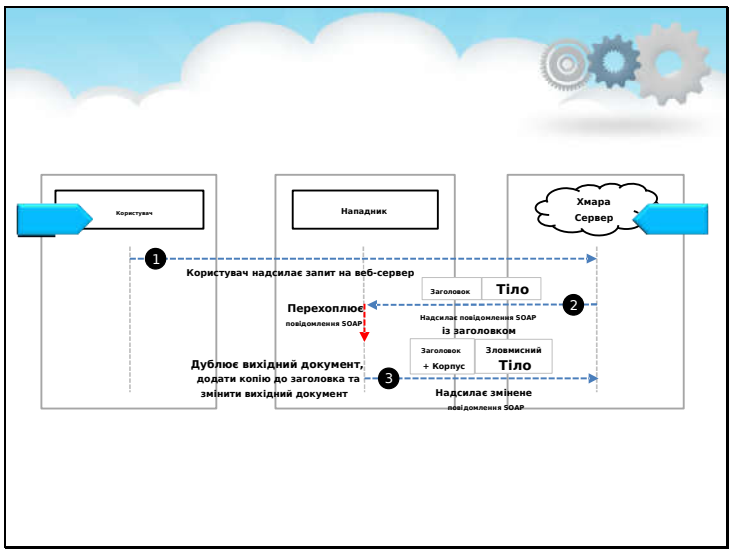

Користувач
Зашифровані облікові дані
Хмара
Сервер
Доступ до зашифрованого
Інформація
Нападник нюхає
трафік
Нападник
Виконує криптоаналіз
на зашифрованих даних
Звичайна текстова інформація
видобутий зловмисником

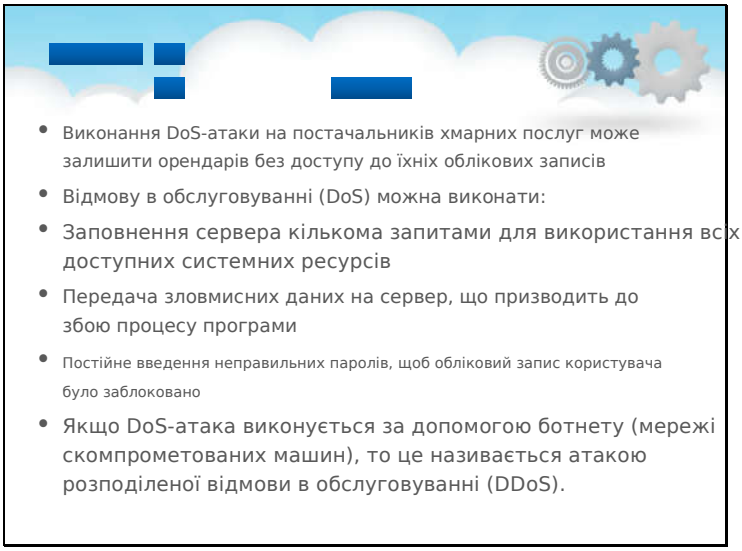


- Використовуйте генератори випадкових чисел, які генерують криптографічно надійні випадкові числа, щоб забезпечити надійність криптографічних матеріалів, таких як ключі безпечної оболонки (SSH) і розширення безпеки системи доменних імен (DNSSEC).
- Не використовуйте несправні криптографічні алгоритми

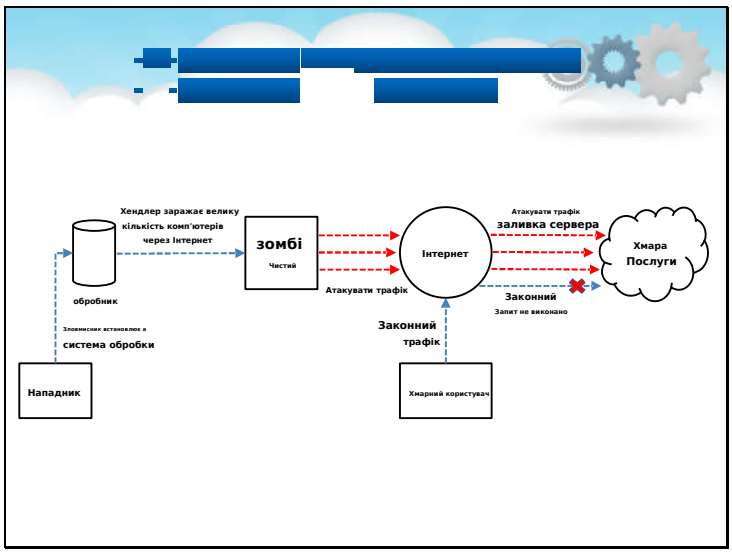


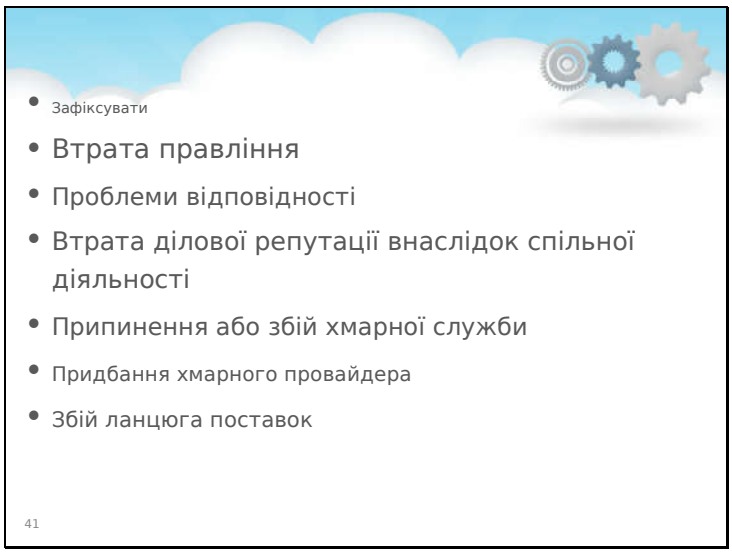
- Атака загортання виконується під час перекладу повідомлення SOAP на рівні TLS, де зловмисники дублюють тіло повідомлення та надсилають його на сервер як законний користувач


Користувач
Нападник
Хмара
Сервер
1
Користувач надсилає запит на веб-сервер
Заголовок
Тіло
2
Надсилає повідомлення SOAP
із заголовком
Перехоплює
повідомлення SOAP
Заголовок
+ Корпус
Зловмисний
Тіло
Дублює вихідний документ,
додати копію до заголовка та
змінити вихідний документ
3
Надсилає змінене
повідомлення SOAP



- Виконання DoS-атаки на постачальників хмарних послуг може залишити орендарів без доступу до їхніх облікових записів
- Відмову в обслуговуванні (DoS) можна виконати:
- Заповнення сервера кількома запитами для використання всіх доступних системних ресурсів
- Передача зловмисних даних на сервер, що призводить до збою процесу програми
- Постійне введення неправильних паролів, щоб обліковий запис користувача було заблоковано
- Якщо DoS-атака виконується за допомогою ботнету (мережі скомпрометованих машин), то це називається атакою розподіленої відмови в обслуговуванні (DDoS).


Хендлер заражає велику
кількість комп'ютерів
через Інтернет
обробник
Зловмисник встановлює
систему обробки
Нападник
зомбі
Чистий
Атакувати трафік
Інтернет
Законний
трафік
Хмарний користувач
Атакувати трафік
заливка сервера
Законний
Запит не виконано
Хмара
Послуги



Зафіксувати

Потенційна залежність від конкретного хмарного постачальника, залежно від зобов’язань постачальника, може призвести до катастрофічної бізнес-провалу, якщо хмарний постачальник збанкрутує або шлях міграції вмісту та програми до іншого постачальника буде надто дорогим. У постачальників хмарних послуг мало або зовсім немає стимулів полегшувати перехід до іншого постачальника, якщо це не передбачено контрактом.

Втрата управління

Використовуючи хмарні інфраструктури, клієнт обов’язково передає контроль постачальнику хмарних технологій у низці питань, які можуть вплинути на безпеку. Це може потенційно серйозно вплинути на стратегію організації, а отже, на здатність досягати її місії та цілей. Втрата контролю та керування може призвести до неможливості дотримання вимог безпеки, недостатньої конфіденційності, цілісності та доступності даних, а також до погіршення продуктивності та якості обслуговування, не кажучи вже про запровадження проблем із відповідністю.

Виклики відповідності

Певним компаніям, які переходять на хмару, може знадобитися відповідність певним галузевим стандартам або нормативним вимогам, як-от індустріальні дані платіжних карток

Стандарт безпеки (PCI DSS). Перехід до хмари може поставити під загрозу ці бізнес-потреби, якщо хмарний постачальник не може надати докази власної відповідності відповідним вимогам або якщо постачальник не дозволяє клієнту проводити аудит.

Втрата ділової репутації через спільну діяльність

Спільне використання ресурсів може спричинити проблеми, коли репутація спільних ресурсів стає заплямованою діяльністю поганого сусіда. Це також включатиме вживання певних заходів для пом’якшення, таких як блокування адреси Інтернет-протоколу (IP) і конфіскація обладнання.

Припинення або збій хмарної служби

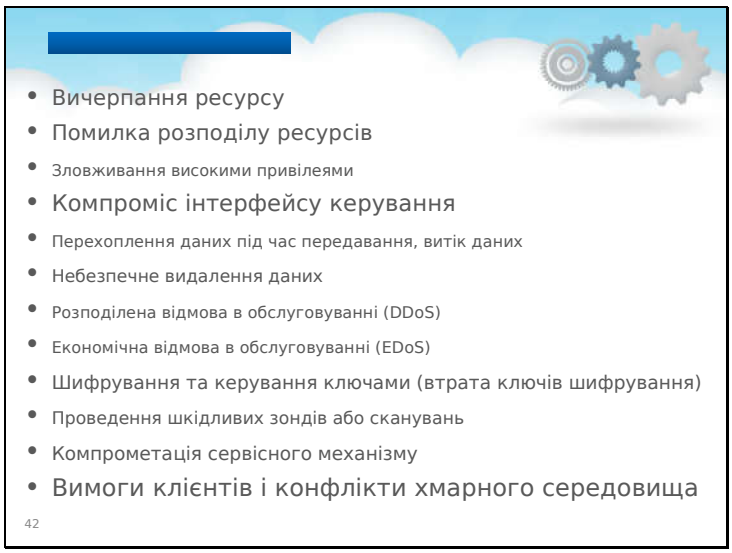
Якщо хмарний постачальник стикається з ризиком припинення діяльності через фінансові, юридичні чи інші причини, клієнт може постраждати від втрати або погіршення продуктивності надання послуг і якості послуг, а також втрати інвестицій.

Придбання хмарного постачальника

Придбання хмарного постачальника може збільшити ймовірність стратегічних змін і може поставити під загрозу попередні угоди. Це може унеможливити дотримання існуючих вимог безпеки. Остаточний вплив може завдати шкоди ключовим активам, таким як репутація організації, довіра клієнтів або пацієнтів, а також лояльність і досвід співробітників.

Збій ланцюга поставок

Постачальник хмарних обчислень може передати певні спеціалізовані завдання своєї інфраструктури третім сторонам. У такій ситуації рівень безпеки хмарного постачальника може залежати від рівня безпеки кожного з посилань і рівня залежності хмарного постачальника від третьої сторони. Загалом, відсутність прозорості в договорі може стати проблемою для всієї системи.



Вичерпання ресурсів

Неточне моделювання запитів клієнтів хмарним постачальником може призвести до недоступності послуг, компрометації контролю доступу та економічних і репутаційних втрат через виснаження ресурсу. Клієнт бере на себе певний рівень ризику, розподіляючи всі ресурси хмарної служби, оскільки ресурси розподіляються відповідно до статистичних прогнозів.

Помилка сегрегації ресурсу

Цей клас ризиків включає збій механізмів, що розділяють сховище, пам’ять, маршрутизацію та навіть репутацію між різними орендарями спільної інфраструктури (атаки з переходом на гостьову систему, атаки з ін’єкціями SQL, що викривають дані багатьох клієнтів, і атаки на бокових каналах). Імовірність цього сценарію інциденту залежить від хмарної моделі, прийнятої замовником. Це менш імовірно для клієнтів приватної хмари порівняно з клієнтами публічної хмари.

Зловживання високими привілеями

Потенційно можуть зловмисні дії інсайдера впливають на конфіденційність, цілісність і доступність усіх типів даних, IP, послуг, а отже, опосередковано на репутацію організації, довіру клієнтів,

та досвід співробітників. Це можна вважати особливо важливим у випадку хмарних обчислень через те, що хмарні архітектури вимагають певних ролей, які є надзвичайно ризикованими. Приклади таких ролей включають системних адміністраторів і аудиторів постачальника хмарних послуг і постачальників керованих послуг безпеки, які мають справу зі звітами про виявлення вторгнень і реагування на інциденти.

Компроміс інтерфейсу керування

Інтерфейси керування клієнтами постачальників загальнодоступних хмар є доступними через Інтернет і забезпечують доступ до більших наборів ресурсів (ніж традиційні постачальники хостингу). Вони також становлять підвищений ризик, особливо в поєднанні з віддаленим доступом і вразливістю веб-браузера.

Перехоплення даних під час передавання, витік даних

Хмарні обчислення, будучи розподіленою архітектурою, передбачають більше даних у передачі, ніж традиційні інфраструктури. Сніфінг, спуфінг, людина посередині, бічний канал і повтор атаки слід розглядати як можливі джерела загрози.

Небезпечне видалення даних

Щоразу, коли змінюється постачальник, ресурси зменшуються, фізичне обладнання перерозподіляється, і дані можуть бути доступними після закінчення терміну служби, зазначеного в безпеці політики. Якщо потрібне справжнє видалення даних, необхідно дотримуватися спеціальних процедур, які можуть не підтримуватися хмарний постачальник.

Розподілена відмова в обслуговуванні

Поширений метод атаки передбачає насичення цільового середовища зовнішніми запитами зв’язку, таким чином воно не може відповідати на законний трафік або відповідає настільки повільно, що стає фактично недоступним. Це може призвести до фінансових та економічних втрат.

Економічна відмова в обслуговуванні

EDoS знищує економічні ресурси; найгіршим сценарієм буде банкрутство клієнта або серйозні економічні наслідки. Можливі такі сценарії: Зловмисник може використовувати обліковий запис і використовувати ресурси клієнта для власної вигоди або для того, щоб завдати клієнту економічної шкоди. Замовник не встановив діючий обмежує використання платних ресурсів і відчуває несподіване навантаження на них

ресурси. Зловмисник може використовувати а
загальнодоступний канал для виснаження виміряних ресурсів клієнта. Наприклад, коли клієнт платить за HTTP-запит, DDoS-атака може мати такий ефект.

Encrypt ion і керування ключами (втрата ключів Encrypt ion)

Цей ризик включає розкриття секретних ключів (SSL, шифрування файлів, особистих ключів клієнтів) або паролів зловмисникам. Це також включає втрату або пошкодження цих ключів або їх несанкціоноване використання для автентифікації та неспростування (цифровий підпис).

Проведення шкідливих зондувань або сканувань

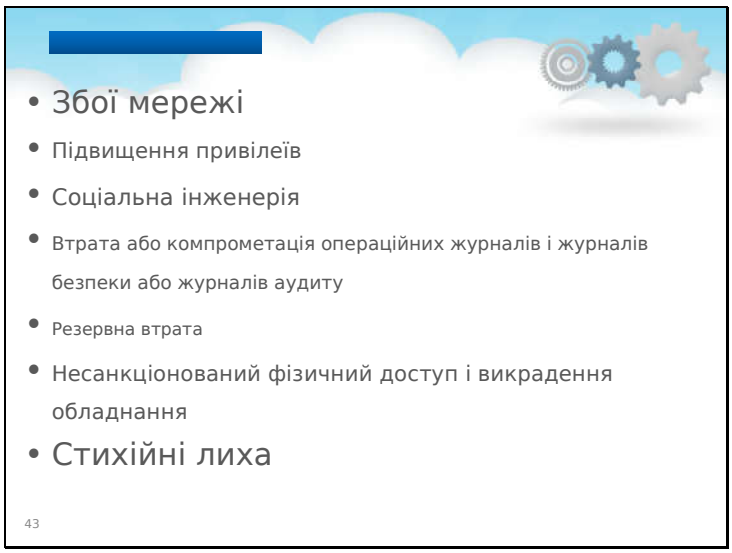
Шкідливі зонди чи сканування, а також відображення мережі є непрямими загрозами для активів, які розглядаються. Їх можна використовувати для збору інформації в контексті злому спроба . Можливим впливом може бути втрата конфіденційності, цілісності та доступності послуг і даних.

Компрометація Service Engine

Кожна хмарна архітектура спирається на вузькоспеціалізовану платформу d і механізм обслуговування. Механізм обслуговування знаходиться над фізичними апаратними ресурсами та керує ресурсами клієнтів на різних рівнях абстракції. Наприклад, у хмарах IaaS цим програмним компонентом може бути гіпервізор. Як і будь-який інший рівень програмного забезпечення, код механізму обслуговування може мати вразливості та схильний до атак або несподіваних збоїв. Постачальники хмарних послуг повинні встановити чіткий розподіл обов’язків із визначенням мінімальних дій, які мають виконати клієнти.

Вимоги клієнта та конфлікти хмарного середовища

Постачальники хмарних послуг повинні встановити чіткий розподіл обов’язків із визначенням мінімальних дій, які мають виконати клієнти. Нездатність клієнтів належним чином захистити своє середовище може стати вразливою для хмарної платформи, якщо постачальник хмарних послуг не вжив необхідних заходів для забезпечення ізоляції. Постачальники хмарних послуг повинні додатково сформулювати свої механізми ізоляції та надати вказівки щодо передової практики, щоб допомогти клієнтам захистити свої ресурси.



Збої мережі

Цей ризик є одним із найвищих ризиків, оскільки він безпосередньо впливає на надання послуг. Він існує через неправильну конфігурацію мережі, вразливість системи, відсутність ізоляції ресурсів і погані або неперевірені плани безперервності бізнесу (BC) і аварійного відновлення (DR). Модифікація мережевого трафіку також може становити ризик для клієнта та хмарного постачальника, якщо надання не виконується належним чином або немає шифрування трафіку чи оцінки вразливості.

Підвищення привілеїв

Хоча існує низька ймовірність використання, підвищення привілеїв може спричинити втрату даних клієнта та контролю доступу. Таким чином, зловмисник може отримати контроль над великими частинами хмарної платформи. Ризик проявляється через автентифікацію, авторизацію та інші вразливості контролю доступу, уразливості гіпервізора (вибухи хмари) і неправильну конфігурацію.

Соціальна інженерія

Цей ризик є одним із найбільш ігнорованих, оскільки більшість технічного персоналу зосереджується на нелюдських аспектах своїх платформ. Використання цього ризику призвело до втрати репутація постачальників хмарних послуг, таких як Amazon і Apple, завдяки публічності подій. Цей ризик може бути

легко звести до мінімуму за допомогою тренінгів з питань безпеки, належного забезпечення користувачів, ізоляції ресурсів, шифрування даних і належних процедур фізичної безпеки.

Втрата або компрометація операційних журналів і журналів безпеки або журналів аудиту

Операційні журнали можуть бути вразливими через відсутність політики або погані процедури збору журналів. Це також включало б уразливості утримання, керування доступом, уразливості деініціалізації користувача, відсутність судової готовності та вразливості ОС.

Резервне копіювання втрати

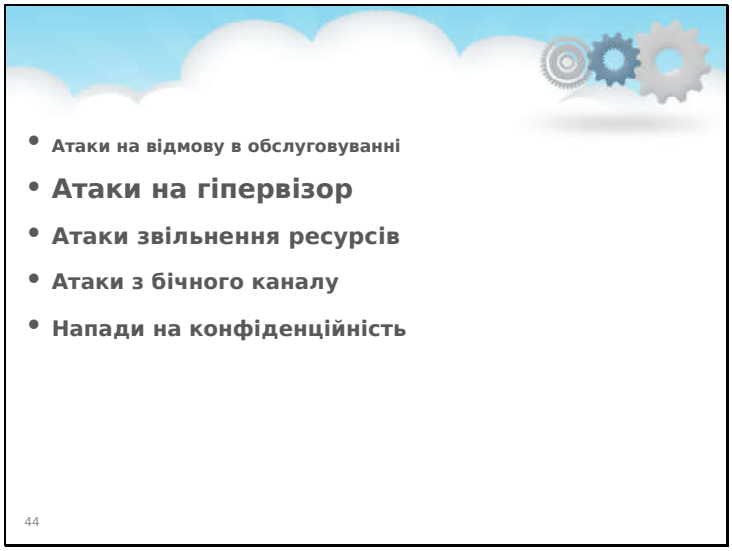
Цей високий ризик впливу впливає на репутацію компанії, і все це підтримується отримання даних і надання послуг. Це також відбувається через неадекватні процедури фізичної безпеки, управління доступом уразливості та вразливості деініціалізації користувача.

Несанкціонований фізичний доступ і викрадення обладнання

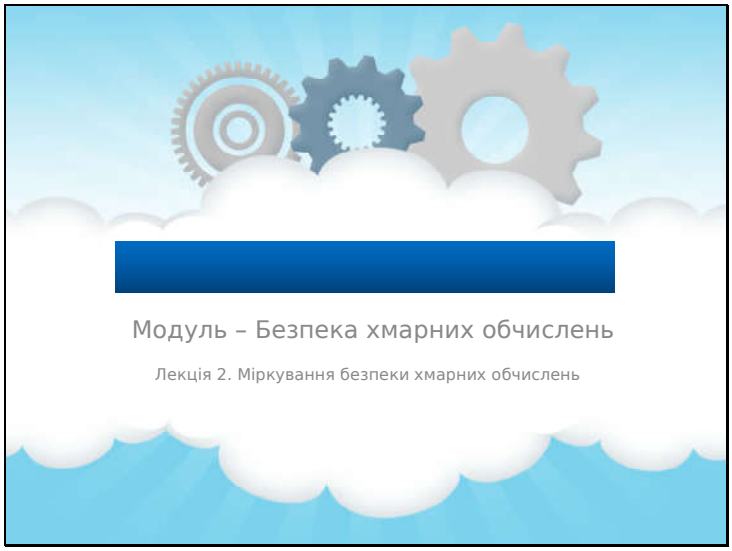
Імовірність того, що зловмисники отримають доступ до фізичного місцезнаходження, дуже низька, але в разі такого Вплив на хмарного провайдера та його клієнтів дуже великий. Це може вплинути на репутацію компанії та дані, розміщені в приміщеннях, і загроза безпеці, яку вони створюють, пов’язана з неадекватними процедурами фізичної безпеки.

Стихійні лиха

Цей ризик часто ігнорується, але він може мати значний вплив на бізнес, задіяний у разі його виникнення. Якщо бізнес має низьку чи неперевірену безперервність і план ліквідації наслідків або його немає, його репутація, дані та надання послуг можуть бути серйозно скомпрометовані.



- Атаки на відмову в обслуговуванні
- **Атаки на гіпервізор**
- Атаки звільнення ресурсів
- Атаки з бічного каналу
- Напади на конфіденційність



Лекція 2. Хмара

Обчислювальна безпека

міркування

Огляд цієї лекції

- Ця лекція присвячена **огляд** з:
  - **рівні контролю безпеки хмари;**
  - **відповідальність споживача та провайдера хмари;**
  - **найкращі практики захисту хмари;**
  - **рекомендації NIST щодо хмарної безпеки;**
  - **основи криптографії**
  - **шифрування хмарного сховища.**

Хмарні рівні контролю безпеки

Рівні керування хмарною безпекою

Рівень програми

Існує кілька механізмів безпеки, пристроїв і політик, які забезпечують підтримку в

різні рівні керування хмарною безпекою. На прикладному рівні

брандмауери веб-програм

розгортаються для фільтрації трафіку та спостереження за його поведінкою. Аналогічно

Системи

Життєвий цикл розробки (SDLC), аналіз двійкового коду, забезпечення безпеки транзакцій

безпека онлайн-транзакцій, аналіз сценаріїв тощо.

Інформація

У хмарних обчисленнях, щоб забезпечити конфіденційність і цілісність інформації, яка знаходиться

обмінюються даними між клієнтом і сервером, для моніторингу налаштовано різні політики

втрата даних. Ці політики включають запобігання втраті даних (DLP) і керування вмістом

Каркас (CMF). Запобігання втраті даних (DLP) — це функція, яка пропонує запобігти

витік інформації за межі мережі. Традиційно ця інформація може включати

конфіденційну інформацію компанії чи організації, службову, фінансову та іншу таємницю
інформації. Функція запобігання втраті даних також забезпечує дотримання вимог
з правилами та положеннями з використанням політик запобігання втраті даних, щоб запобігти користувачеві
навмисне чи ненавмисне надсилання цієї конфіденційної інформації.

управління
Безпека Cloud Computing щодо управління здійснюється різними підходами
таких як управління, управління ризиками та відповідність (GRC), ідентифікація та доступ
Керування (IAM), керування виправленнями та налаштуваннями. Ці підходи допомагають
контролювати безпечний доступ до ресурсів і керувати ними.

Мережевий рівень
Є деякі доступні рішення для захисту мережевого рівня в хмарних обчисленнях, наприклад
розгортання пристроїв IDS/IPS наступного покоління, брандмауерів наступного покоління, DNSSec,
Анти-DDoS, OAuth і Deep Packet Inspection (DPI) тощо. Вторгнення нового покоління
Система запобігання, відома як NGIPS, є одним із ефективних проактивних компонентів у
Інтегроване рішення захисту від загроз. NGIPS забезпечує міцніший рівень безпеки з глибиною

видимість, розширені дані безпеки та розширений захист від нової загрози
захистити складні інфраструктури мереж.

Надійні обчислення
Корінь довіри (RoT) встановлюється шляхом перевірки кожного компонента обладнання та

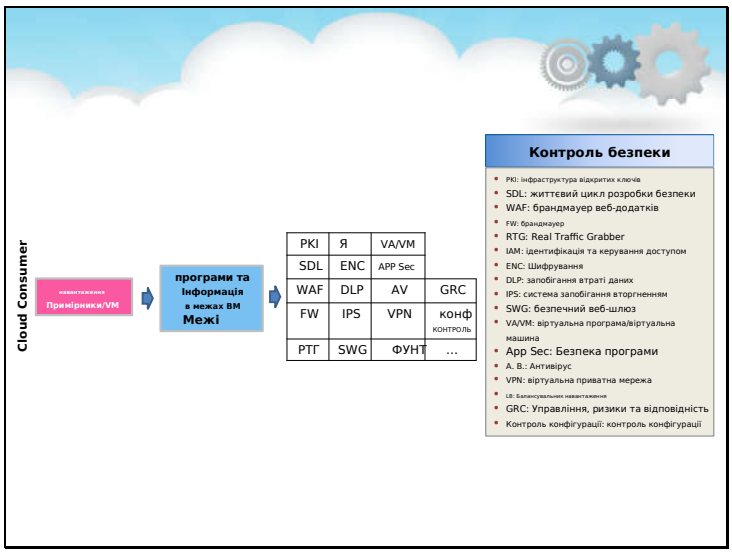
програмне забезпечення від кінцевої сутності до кореневого сертифіката. Це призначено лише для того, щоб забезпечити
можна використовувати надійне програмне та апаратне забезпечення, зберігаючи при цьому гнучкість.

Комп'ютер і зберігання
Обчислення та зберігання в хмарних обчисленнях можна захистити шляхом впровадження на основі хосту
Системи виявлення або запобігання вторгненням HIDS/HIPS. Налаштування перевірки цілісності, файл

моніторинг системи та аналіз файлів журналу, аналіз підключень, виявлення рівня ядра,
Шифрування сховища тощо. IPS/IDS на основі хосту зазвичай розгортається для захисту
конкретної хост-машини, і вона тісно співпрацює з ядром операційної системи хоста
машина. Він створює рівень фільтрації та відфільтровує будь-які виклики шкідливих програм до ОС.

Фізична безпека

Фізична безпека завжди потрібна в пріоритеті, щоб захистити будь-що. Оскільки він теж перший
рівень моделі OSI, якщо пристрій фізично не захищений, будь-яка конфігурація безпеки
не буде ефективним. Фізична безпека включає захист від рукотворних атак, наприклад
як крадіжка, пошкодження, несанкціонований фізичний доступ, а також вплив на навколишнє середовище, наприклад
дощ, пил, відключення електроенергії, пожежа тощо.



Відповідальність споживача та постачальника хмари

Обов’язки споживача хмарних послуг включають дотримання наступної безпеки

елементи керування:

Інфраструктура відкритих ключів (PKI). Життєвий цикл розробки безпеки (SDLC). Брандмауер веб-додатків (WAF). Брандмауер

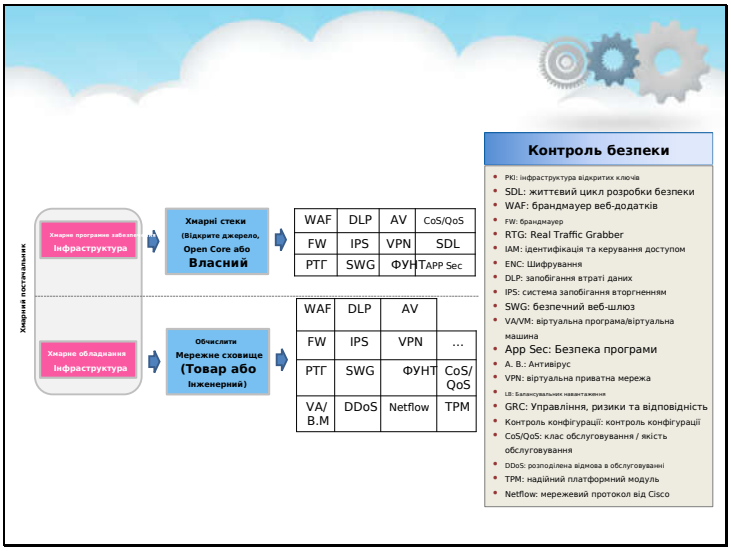
Шифрування.

Системи запобігання вторгненням

Безпечний веб-шлюз

Безпека програми

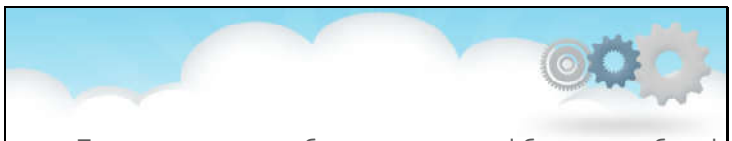
Віртуальна приватна мережа (VPN) та інші.



Постачальник хмарних послуг

Обов’язки постачальника хмарних послуг включають дотримання таких заходів безпеки:

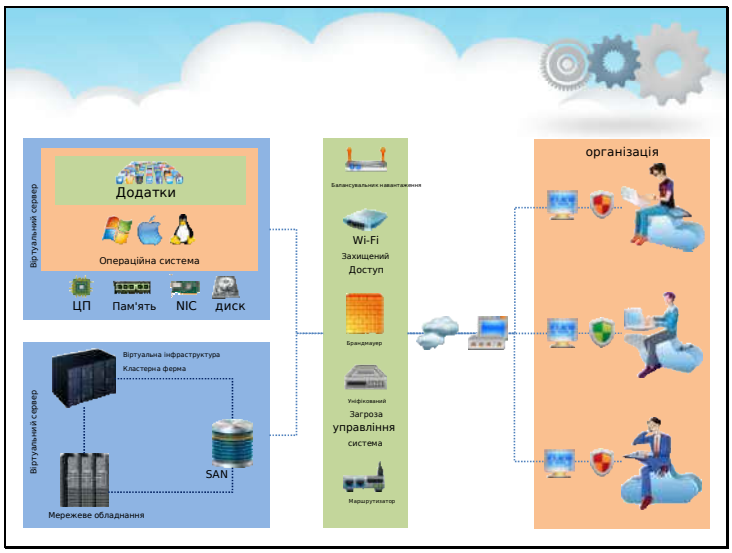
Брандмауер веб-додатків (WAF).
Брандмауер Real Traffic Grabber (RTG).
Запобігання втраті даних (DLP)
Системи запобігання вторгненням
Захищений веб-шлюз (SWG) Безпека додатків (App Sec) Віртуальна приватна мережа (VPN)
Балансування навантаження
CoS/QoS
Модуль Trusted Platform Module Netflow та інші.

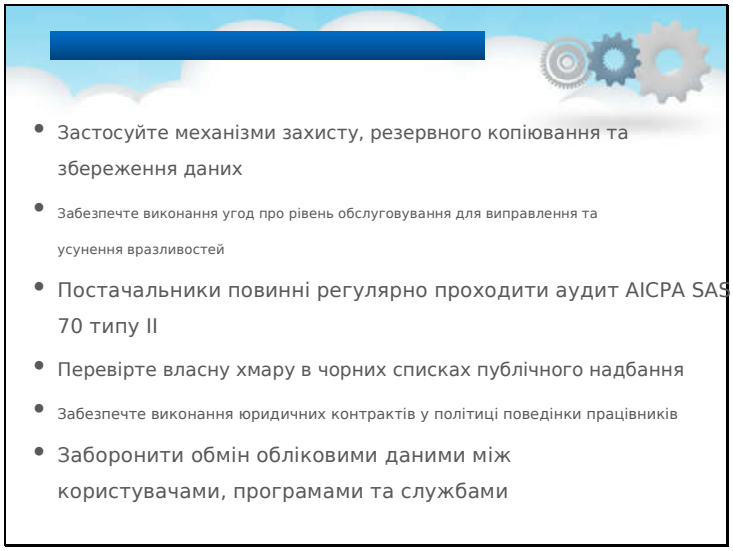


- Послуги хмарних обчислень повинні бути розроблені постачальником відповідно до вимог безпеки клієнтів
- Постачальники хмарних послуг повинні забезпечувати більш високий рівень багатокористування, що дозволяє оптимально використовувати ресурси хмари та захищати дані та програми
- Хмарні сервіси повинні впроваджувати план аварійного відновлення для збережених даних, що дозволяє отримувати інформацію в неочікуваних ситуаціях
- Для підтримки угод про рівень обслуговування між споживачами та постачальниками послуг необхідний постійний моніторинг якості обслуговування (QoS).

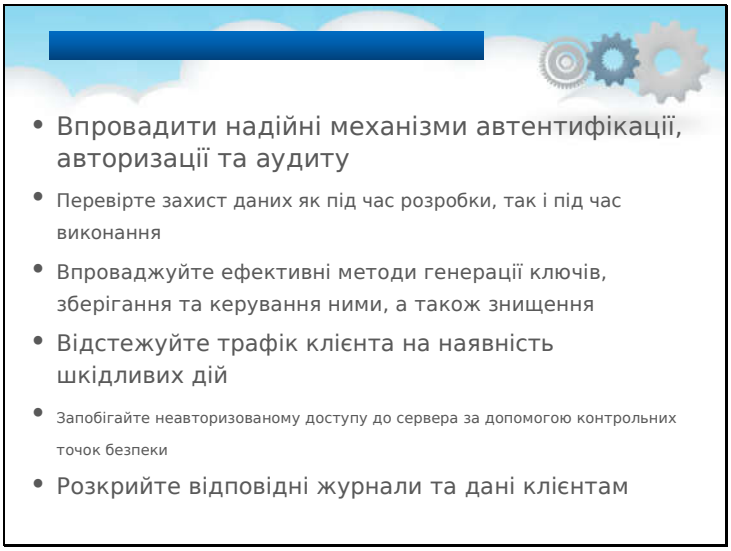
- Дані, що зберігаються в хмарних службах, повинні бути реалізовані безпечно, щоб забезпечити цілісність даних
- Служба хмарних обчислень має бути швидкою, надійною та повинна забезпечувати швидкий час відповіді на нові запити
- Симетричні та асиметричні криптографічні алгоритми повинні бути реалізовані для оптимальної безпеки даних у хмарних обчисленнях

- Операційний процес хмарних служб має бути спроектований, керований та безпечно інтегрований до організаційного управління безпекою
- Балансування навантаження має бути включено в хмарні служби для полегшення мереж і ресурсів для покращення часу відповіді завдання з максимальною пропускною здатністю

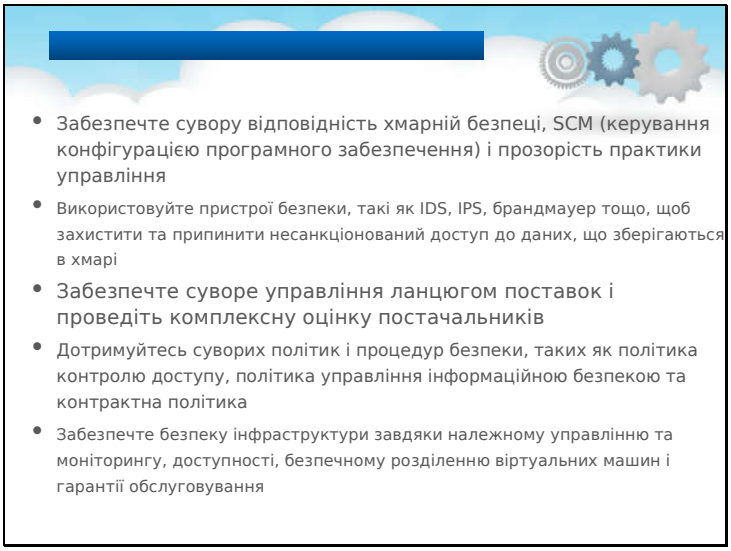

Віртуальний сервер
Додатки
Операційна система
ЦП
Пам'ять
NIC
диск
Віртуальний сервер
Віртуальна інфраструктура
Кластерна ферма
SAN
Мережеве обладнання
Балансувальник навантаження
Wi-Fi
Захищений
Доступ
Брандмауер
Уніфікований
Загроза
управління
система
Маршрутизатор
організація



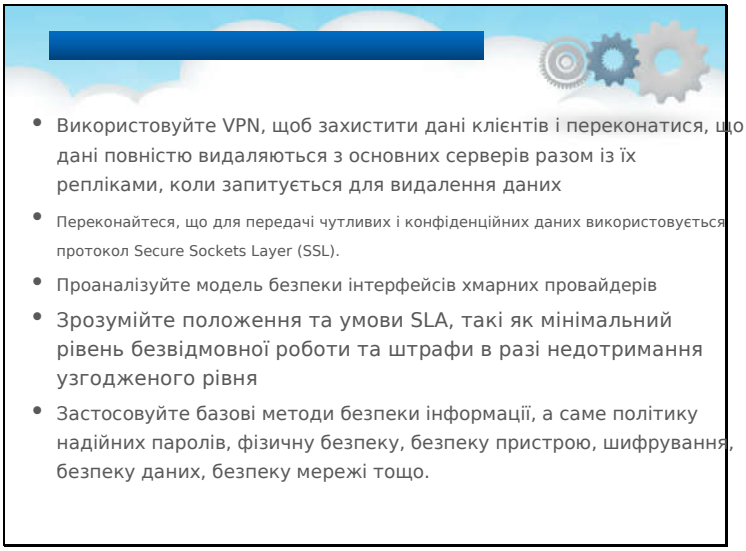
Найкращі практики захисту хмари



- Впровадити надійні механізми автентифікації, авторизації та аудиту
- Перевірте захист даних як під час розробки, так і під час виконання
- Впроваджуйте ефективні методи генерації ключів, зберігання та керування ними, а також знищення
- Відстежуйте трафік клієнта на наявність шкідливих дій
- Запобігайте неавторизованому доступу до сервера за допомогою контрольних точок безпеки
- Розкрийте відповідні журнали та дані клієнтам

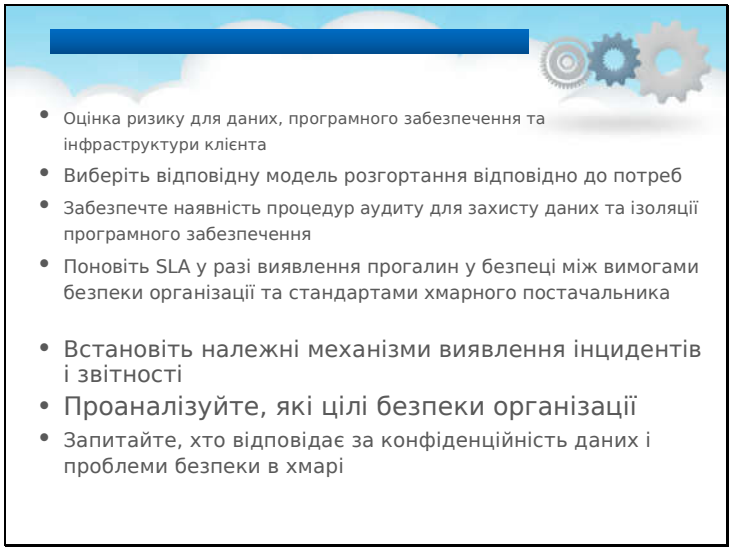


- Забезпечте сувору відповідність хмарній безпеці, SCM (керування конфігурацією програмного забезпечення) і прозорість практики управління
- Використовуйте пристрої безпеки, такі як IDS, IPS, брандмауер тощо, щоб захистити та припинити несанкціонований доступ до даних, що зберігаються в хмарі
- Забезпечте суворе управління ланцюгом поставок і проведіть комплексну оцінку постачальників
- Дотримуйтесь суворих політик і процедур безпеки, таких як політика контролю доступу, політика управління інформаційною безпекою та контрактна політика
- Забезпечте безпеку інфраструктури завдяки належному управлінню та моніторингу, доступності, безпечному розділенню віртуальних машин і гарантії обслуговування



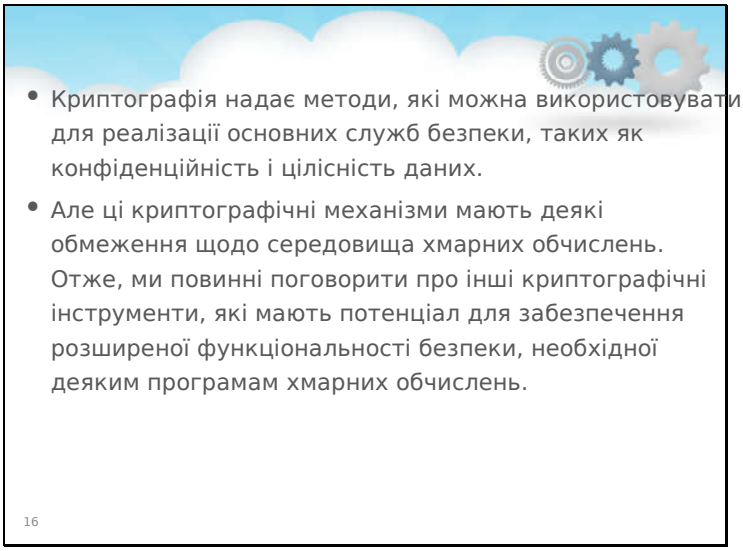
- Використовуйте VPN, щоб захистити дані клієнтів і переконатися, що дані повністю видаляються з основних серверів разом із їх репліками, коли запитується для видалення даних
- Переконайтеся, що для передачі чутливих і конфіденційних даних використовується протокол Secure Sockets Layer (SSL).
- Проаналізуйте модель безпеки інтерфейсів хмарних провайдерів
- Зрозумійте положення та умови SLA, такі як мінімальний рівень безвідмовної роботи та штрафи в разі недотримання узгодженого рівня
- Застосовуйте базові методи безпеки інформації, а саме політику надійних паролів, фізичну безпеку, безпеку пристрою, шифрування, безпеку даних, безпеку мережі тощо.


Аналізуйтеполітики безпеки хмарних провайдерів і SLA
Оцінити безпекухмарні APIа також зареєструвати клієнтамережевий трафік
Переконайтеся, що хмара регулярно проходить перевірки безпеки та оновлення
Переконайтеся, що фізична безпека є а24 x 7 x 365роман
Примусовостандарти безпекив установці/конфігурації
Переконайтеся, що пам'ять, сховище та доступ до мережі єізольований
Сильне кредитне плечедвофакторна аутентифікаціятехніки, де це можливо
Базовий рівеньсповіщення про порушення безпеки процес
АналізуйтеПрограмне забезпечення ланцюга залежностей APIмодулі
Дотримуватись суворихпроцес реєстрації та перевірки
Виконайте вразливість і конфігураціюоцінка ризику
Розкрити інформацію про інфраструктуру, виправлення безпекиі відомості про брандмауер



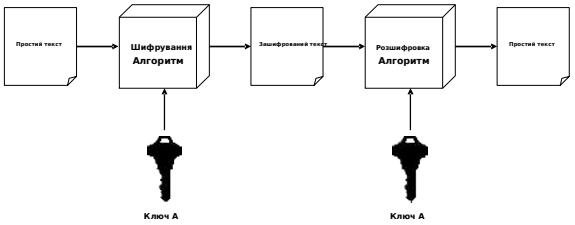
Рекомендації NIST щодо безпеки в хмарі

управління	організація	Провайдер
Чи кожен знає про свої обов'язки щодо безпеки хмари?		
Чи існує механізм оцінки безпеки хмарного сервісу?		
Чи пом'якшує бізнес-управління ризики безпеці, які можуть виникнути через хмарні «тіньові IT»?		
Чи знає організація, в межах яких юрисдикцій можуть зберігатися її дані?		
Чи існує механізм управління ризиками, пов'язаними з хмарою?		
Чи розуміє організація архітектуру даних, необхідну для належної безпеки на всіх рівнях?		
Чи може організація бути впевненою в безперервності наскрізного обслуговування кількох постачальників хмарних послуг?		
Чи дотримується постачальник усіх відповідних галузевих стандартів (наприклад, Закону Великобританії про захист даних)?		
Чи розуміє функція відповідності конкретні нормативні питання, пов'язані з впровадженням організацією хмарних послуг?		

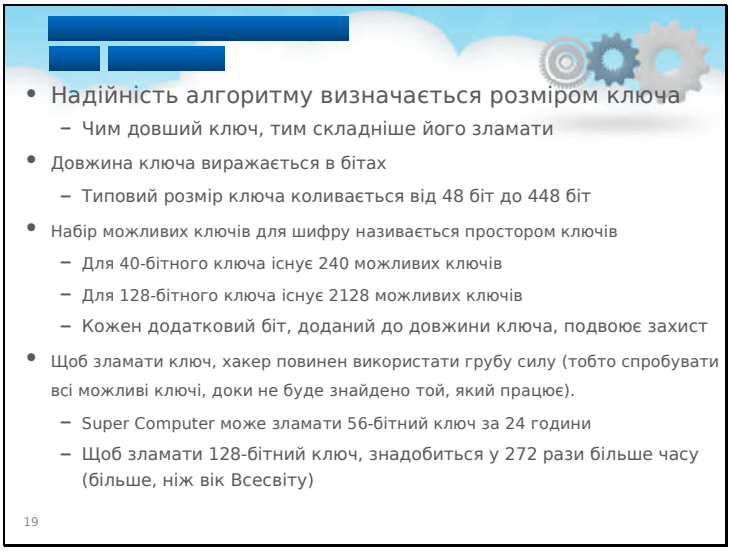


Основи криптографії

- Алгоритми із симетричним ключем — це алгоритми криптографії, які використовують однакові криптографічні ключі як для шифрування відкритого тексту, так і для дешифрування зашифрованого тексту. Ключі, на практиці, являють собою спільний секрет між двома або більше сторонами. Ця вимога, щоб обидві сторони мали доступ до секретного ключа, є одним із головних недоліків шифрування з симетричним ключем.

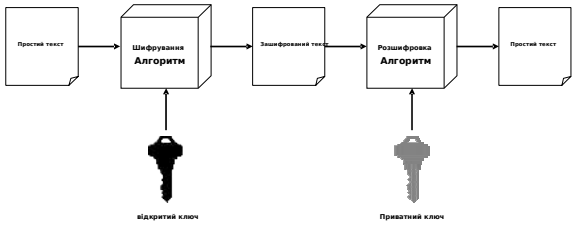


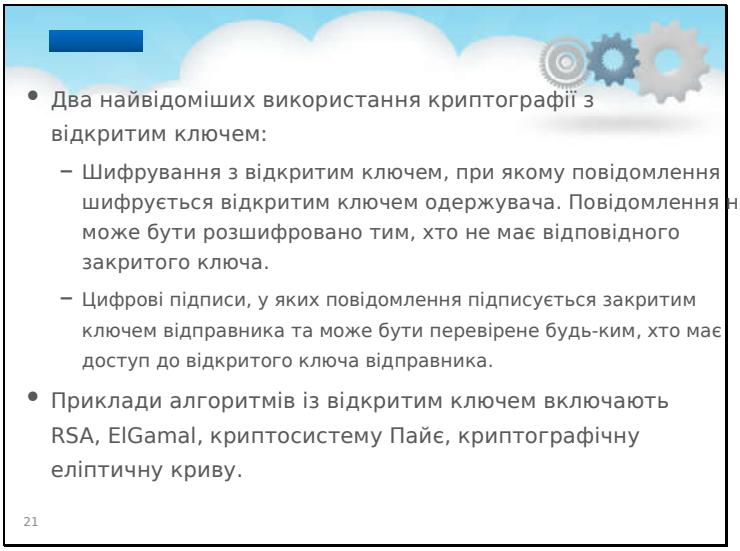
- Шифрування із симетричним ключем може використовувати або потокові шифри, або блочні шифри
  - Потокові шифри шифрують цифри (зазвичай байти) або літери (у шифрах заміни) повідомлення по одній.
  - Блокові шифри беруть кілька бітів і шифрують їх як єдине ціле, доповнюючи відкритий текст так, щоб він був кратним розміру блоку. Зазвичай використовувалися блоки з 64 бітів.
- Приклади популярних алгоритмів із симетричним ключем включають Twofish, Serpent, AES (Rijndael), Blowfish, CAST5, Kuznyechik, RC4, 3DES, Skipjack, IDEA.



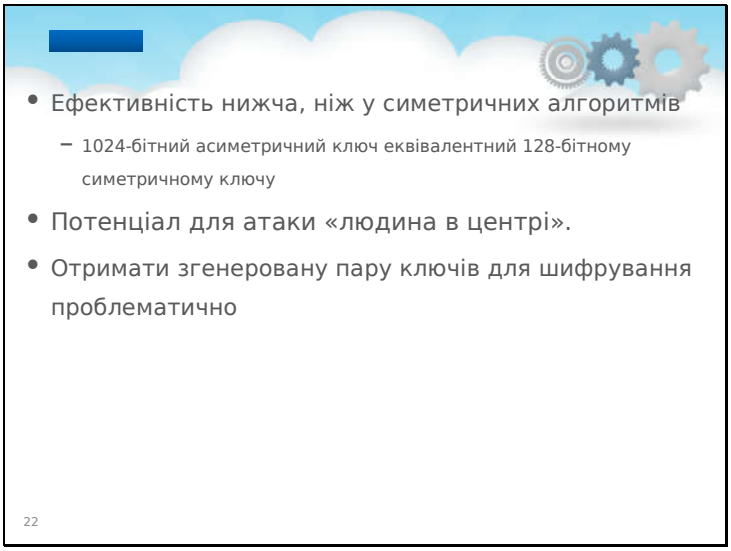
- Надійність алгоритму визначається розміром ключа
  - Чим довший ключ, тим складніше його зламати
- Довжина ключа виражається в бітах
  - Типовий розмір ключа коливається від 48 біт до 448 біт
- Набір можливих ключів для шифру називається простором ключів
  - Для 40-бітного ключа існує 240 можливих ключів
  - Для 128-бітного ключа існує 2128 можливих ключів
  - Кожен додатковий біт, доданий до довжини ключа, подвоює захист
- Щоб зламати ключ, хакер повинен використати грубу силу (тобто спробувати всі можливі ключі, доки не буде знайдено той, який працює).
  - Super Computer може зламати 56-бітний ключ за 24 години
  - Щоб зламати 128-бітний ключ, знадобиться у 272 рази більше часу (більше, ніж вік Всесвіту)

- Криптографія з відкритим ключем, або асиметрична криптографія, — це будь-яка криптографічна система, яка використовує пари ключів: відкритих ключів, які можуть широко поширюватися, та закритих ключів, які відомі лише власнику. Це виконує дві функції: автентифікацію, коли відкритий ключ підтверджує, що власник парного закритого ключа надіслав повідомлення, і шифрування, коли лише парний власник закритого ключа може розшифрувати повідомлення, зашифроване відкритим ключем.

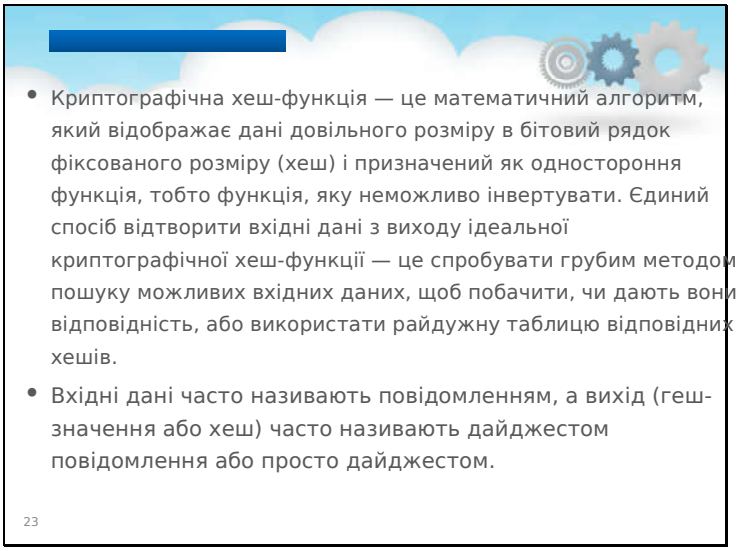




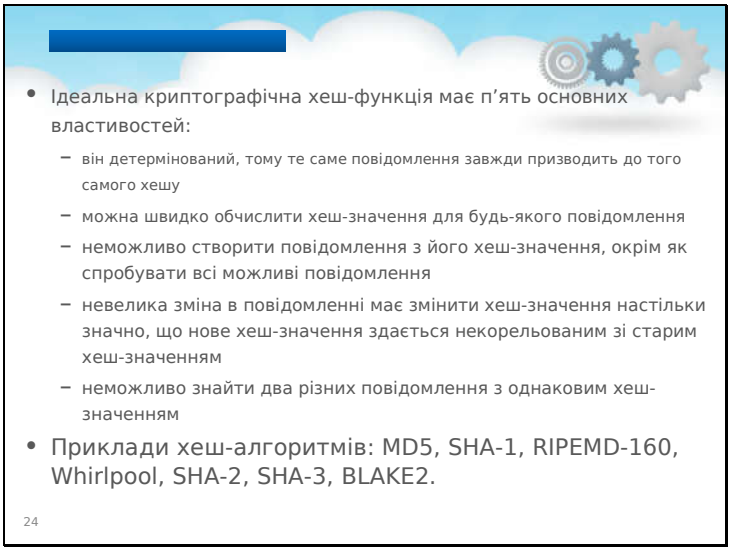
- Два найвідоміших використання криптографії з відкритим ключем:
  - Шифрування з відкритим ключем, при якому повідомлення шифрується відкритим ключем одержувача. Повідомлення не може бути розшифровано тим, хто не має відповідного закритого ключа.
  - Цифрові підписи, у яких повідомлення підписується закритим ключем відправника та може бути перевірене будь-ким, хто має доступ до відкритого ключа відправника.
- Приклади алгоритмів із відкритим ключем включають RSA, ElGamal, криптосистему Пайє, криптографічну еліптичну криву.



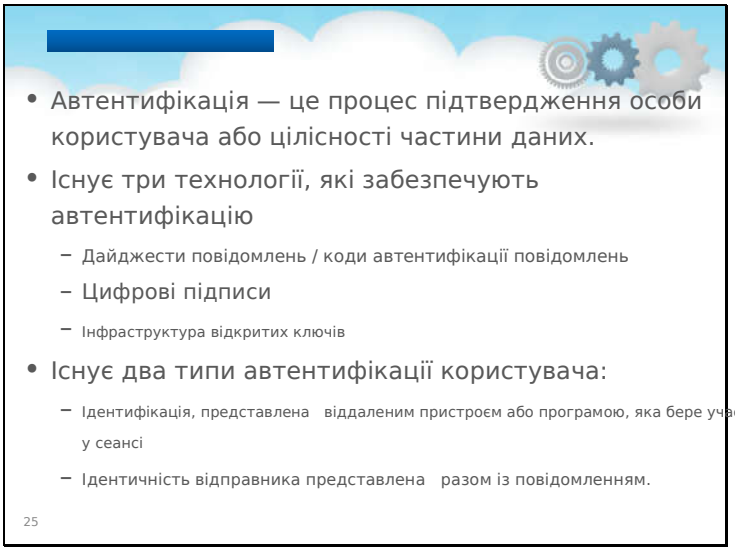
- Ефективність нижча, ніж у симетричних алгоритмів
  - 1024-бітний асиметричний ключ еквівалентний 128-бітному симетричному ключу
- Потенціал для атаки «людина в центрі».
- Отримати згенеровану пару ключів для шифрування проблематично



- Криптографічна хеш-функція — це математичний алгоритм, який відображає дані довільного розміру в бітовий рядок фіксованого розміру (хеш) і призначений як одностороння функція, тобто функція, яку неможливо інвертувати. Єдиний спосіб відтворити вхідні дані з виходу ідеальної криптографічної хеш-функції — це спробувати грубим методом пошуку можливих вхідних даних, щоб побачити, чи дають вони відповідність, або використати райдужну таблицю відповідних хешів.
- Вхідні дані часто називають повідомленням, а вихід (геш-значення або хеш) часто називають дайджестом повідомлення або просто дайджестом.



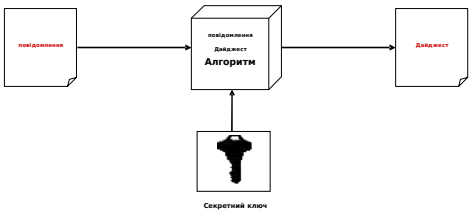
- Ідеальна криптографічна хеш-функція має п’ять основних властивостей:
  - він детермінований, тому те саме повідомлення завжди призводить до того самого хешу
  - можна швидко обчислити хеш-значення для будь-якого повідомлення
  - неможливо створити повідомлення з його хеш-значення, окрім як спробувати всі можливі повідомлення
  - невелика зміна в повідомленні має змінити хеш-значення настільки значно, що нове хеш-значення здається некорельованим зі старим хеш-значенням
  - неможливо знайти два різних повідомлення з однаковим хеш-значенням
- Приклади хеш-алгоритмів: MD5, SHA-1, RIPEMD-160, Whirlpool, SHA-2, SHA-3, BLAKE2.

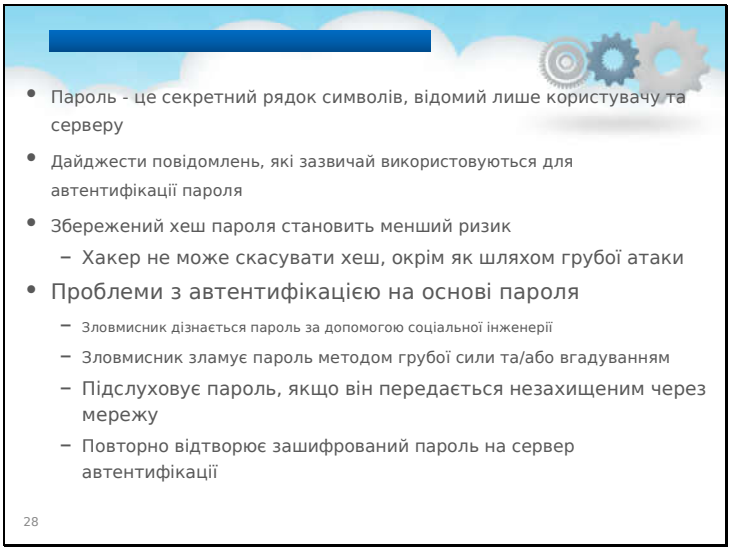


- Автентифікація — це процес підтвердження особи користувача або цілісності частини даних.
- Існує три технології, які забезпечують автентифікацію
  - Дайджести повідомлень / коди автентифікації повідомлень
  - Цифрові підписи
  - Інфраструктура відкритих ключів
- Існує два типи автентифікації користувача:
  - Ідентифікація, представлена віддаленим пристроєм або програмою, яка бере участь у сеансі
  - Ідентичність відправника представлена разом із повідомленням.

- Дайджест повідомлення – це відбиток документа
- Метою дайджесту повідомлення є підтвердження того, що дані не змінено
- Процес створення дайджесту повідомлення з даних називається хешуванням

- Дайджест повідомлення, створений за допомогою ключа
- Створює безпеку, вимагаючи, щоб обидві сторони мали секретний ключ, щоб отримати повідомлення

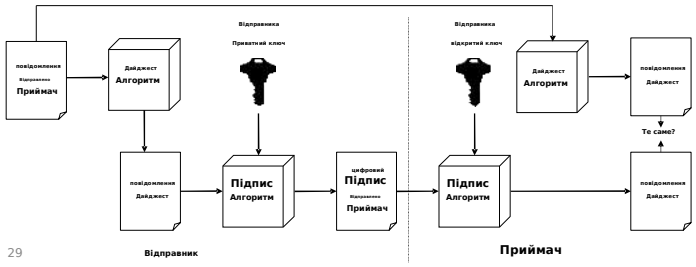




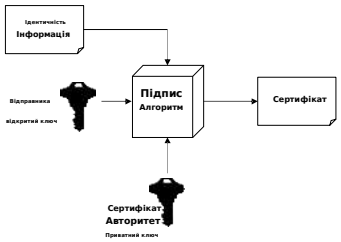
- Пароль - це секретний рядок символів, відомий лише користувачу та серверу
- Дайджести повідомлень, які зазвичай використовуються для автентифікації пароля
- Збережений хеш пароля становить менший ризик
  - Хакер не може скасувати хеш, окрім як шляхом грубої атаки
- Проблеми з автентифікацією на основі пароля
  - Зловмисник дізнається пароль за допомогою соціальної інженерії
  - Зловмисник зламує пароль методом грубої сили та/або вгадуванням
  - Підслуховує пароль, якщо він передається незахищеним через мережу
  - Повторно відтворює зашифрований пароль на сервер автентифікації

- Цифровий підпис — це елемент даних, який супроводжує або логічно пов’язаний із повідомленням у цифровому кодуванні.

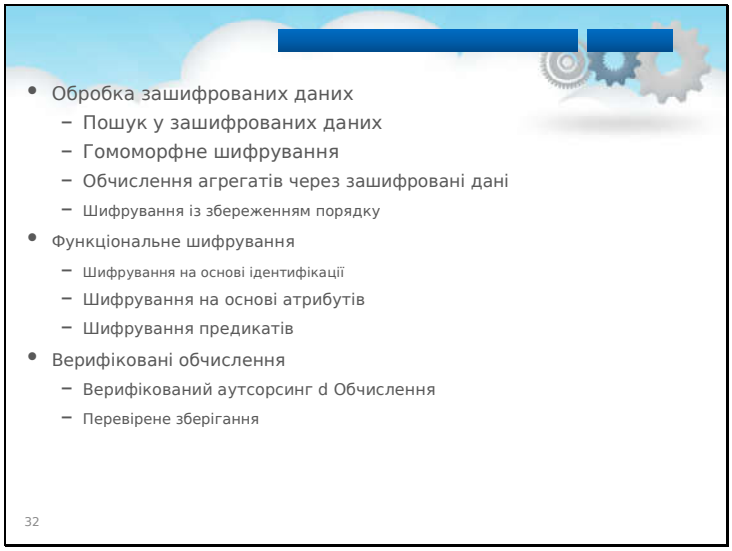
- Він має дві мети
  - Гарантія джерела даних
  - Доказ того, що дані не були підроблені



- Цифровий сертифікат — це підписана заява довіреної сторони про те, що відкритий ключ іншої сторони належить їй.
  - Це дозволяє одному центру сертифікації бути авторизованим іншим органом (кореневим ЦС)
- Сертифікат верхнього рівня має бути самопідписаним

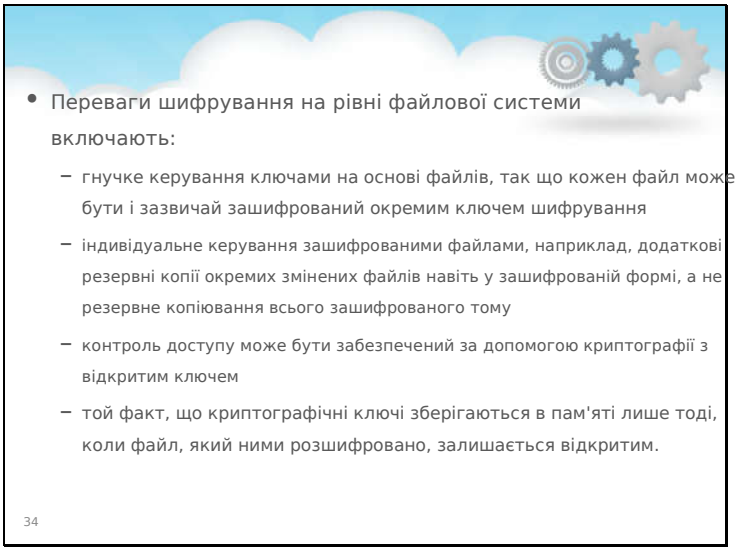


- Хмарні середовища створюють ряд проблем, які не вирішуються звичайними криптографічні механізми. Нижче наведено три основні обмеження традиційної криптографії, які застосовуються до налаштувань хмари:
  - Неможливість виконувати обробку зашифрованих даних
  - Включення політик доступу до даних
  - Надійність власника зашифрованих даних

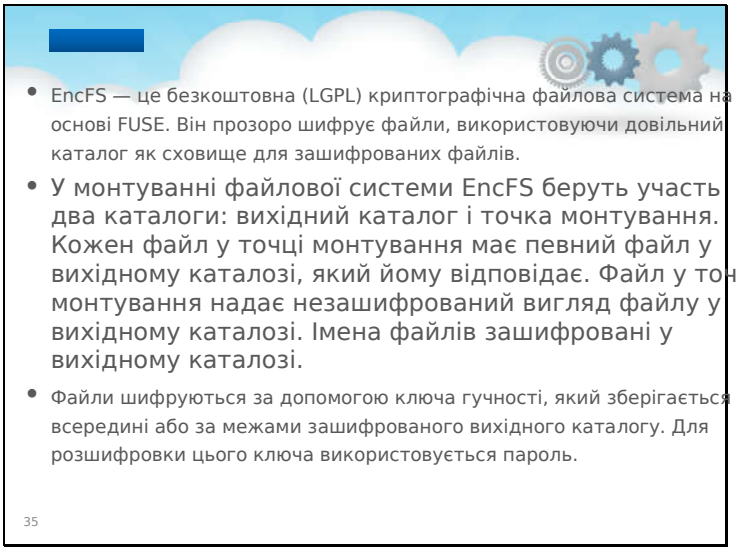


- Обробка зашифрованих даних
  - Пошук у зашифрованих даних
  - Гомоморфне шифрування
  - Обчислення агрегатів через зашифровані дані
  - Шифрування із збереженням порядку
- Функціональне шифрування
  - Шифрування на основі ідентифікації
  - Шифрування на основі атрибутів
  - Шифрування предикатів
- Верифіковані обчислення
  - Верифікований аутсорсинг d Обчислення
  - Перевірене зберігання

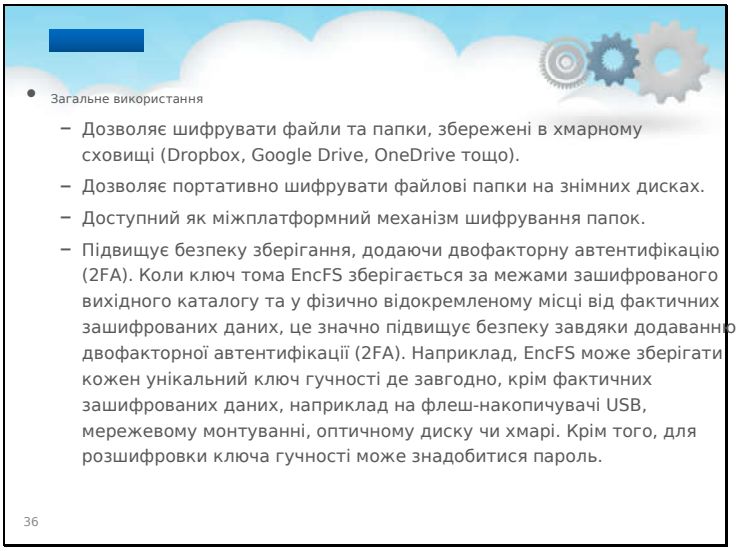
Шифрування хмарного сховища



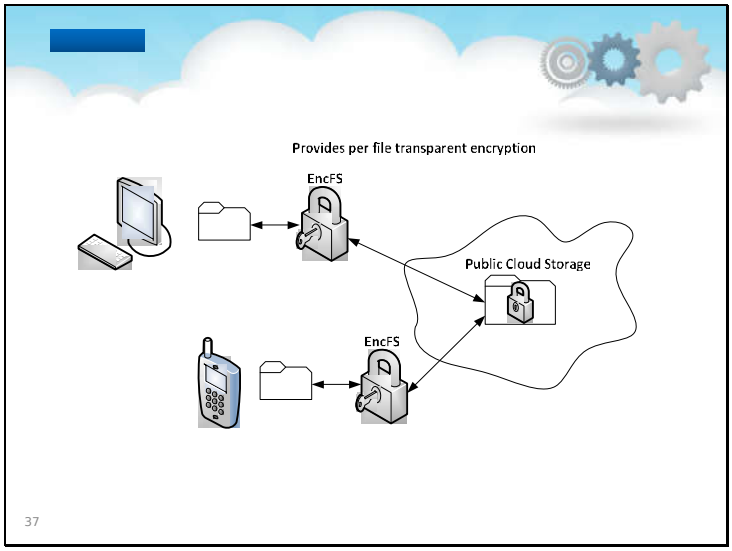
- Переваги шифрування на рівні файлової системи включають:
  - гнучке керування ключами на основі файлів, так що кожен файл може бути і зазвичай зашифрований окремим ключем шифрування
  - індивідуальне керування зашифрованими файлами, наприклад, додаткові резервні копії окремих змінених файлів навіть у зашифрованій формі, а не резервне копіювання всього зашифрованого тому
  - контроль доступу може бути забезпечений за допомогою криптографії з відкритим ключем
  - той факт, що криптографічні ключі зберігаються в пам'яті лише тоді, коли файл, який ними розшифровано, залишається відкритим.



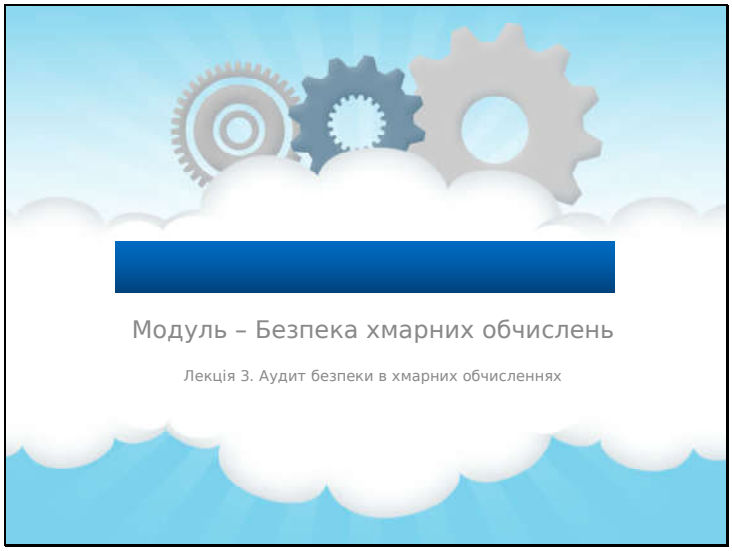
- EncFS — це безкоштовна (LGPL) криптографічна файлова система на основі FUSE. Він прозоро шифрує файли, використовуючи довільний каталог як сховище для зашифрованих файлів.
- У монтуванні файлової системи EncFS беруть участь два каталоги: вихідний каталог і точка монтування. Кожен файл у точці монтування має певний файл у вихідному каталозі, який йому відповідає. Файл у точці монтування надає незашифрований вигляд файлу у вихідному каталозі. Імена файлів зашифровані у вихідному каталозі.
- Файли шифруються за допомогою ключа гучності, який зберігається всередині або за межами зашифрованого вихідного каталогу. Для розшифровки цього ключа використовується пароль.



- Загальне використання
  - Дозволяє шифрувати файли та папки, збережені в хмарному сховищі (Dropbox, Google Drive, OneDrive тощо).
  - Дозволяє портативно шифрувати файлові папки на знімних дисках.
  - Доступний як міжплатформний механізм шифрування папок.
  - Підвищує безпеку зберігання, додаючи двофакторну автентифікацію (2FA). Коли ключ тома EncFS зберігається за межами зашифрованого вихідного каталогу та у фізично відокремленому місці від фактичних зашифрованих даних, це значно підвищує безпеку завдяки додаванню двофакторної автентифікації (2FA). Наприклад, EncFS може зберігати кожен унікальний ключ гучності де завгодно, крім фактичних зашифрованих даних, наприклад на флеш-накопичувачі USB, мережевому монтуванні, оптичному диску чи хмарі. Крім того, для розшифровки ключа гучності може знадобитися пароль.



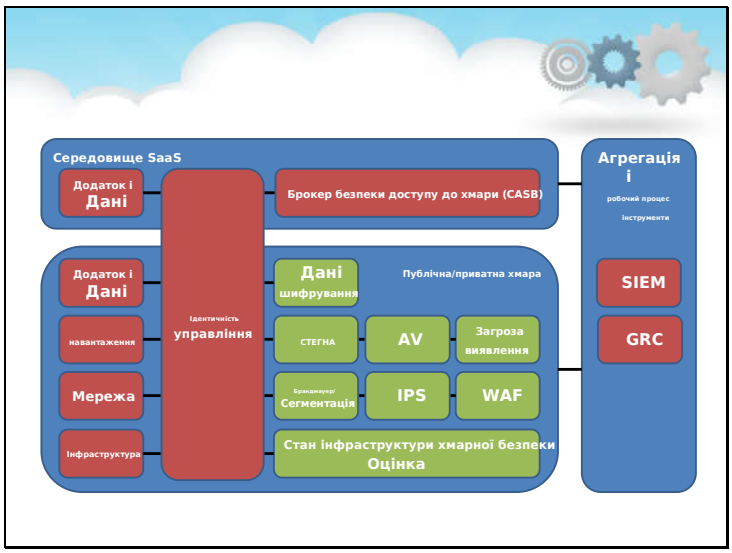
EncFS забезпечує прозоре шифрування кожного файлу, коли дані передаються в хмару, і розшифровку, коли дані повертаються. Цю функцію можна використовувати для забезпечення конфіденційності в публічному хмарному сховищі.



Лекція 3. Аудит безпеки в Хмарні обчислення

Огляд цієї лекції

- Ця лекція присвячена**огляд**з:
  - **хмарний ландшафт безпеки;**
  - **інструменти хмарної безпеки;**
  - **брокер безпеки доступу до хмари (CASB);**
  - **брандмауер веб-додатків.**



Хмарний ландшафт безпеки

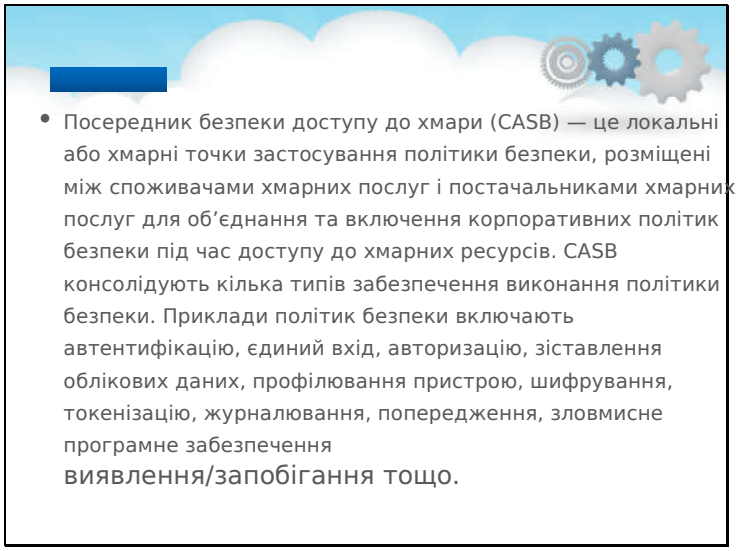
Ефективна хмарна безпека має достатньо рівнів, тому навіть досвідченим професіоналам у сфері безпеки іноді важко впоратися з багатьма компонентами надійної хмарної стратегії безпеки – і для цього потрібна стратегія.

На жаль, ми неодноразово бачимо організації з підходами, які є сильними в сферах, які вони охоплюють, але пропускають один (або кілька) рівнів, що наражає організації на серйозні загрози. Завдяки цьому ми склали діаграму, яка висвітлює основні рівні та постачальників у ландшафті хмарної безпеки. Вони варіюються від добре відомих компонентів, таких як антивірус і брандмауери, до критично важливих нових рішень, таких як інструменти оцінки стану безпеки хмарної інфраструктури (CISPA), ефективні стратегії безпеки хмари:

- Прийняти спільну модель безпеки, описану основними постачальниками IaaS
- Переконайтеся, що кожен шар враховано

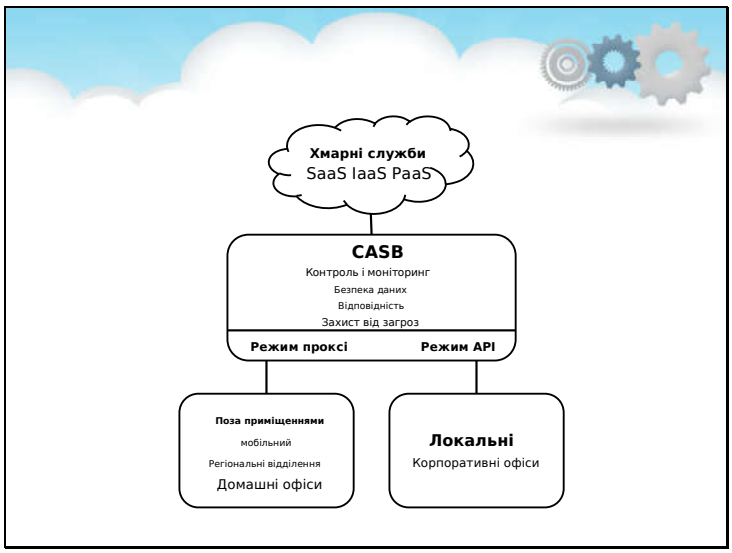
Хмарні інструменти безпеки

- Logz.io – користувачі можуть створювати проактивні сповіщення про вибрані події та відповідні інформаційні панелі, щоб агрегувати та переглядати тенденції даних і відстежувати загрози безпеці, включаючи виявлення паролів грубою форсацією, контроль доступу та доступ до мережі
- Metasploit – приймає хмарну IP-адресу та перевіряє проникнення, щоб переконатися, що безпека діє
- MyPermissions – надсилає сповіщення щоразу, коли програми чи служби намагаються отримати доступ до особистих даних
- Nessus – працює як інструмент оцінки вразливості з відкритим кодом
- Netskope – виявляє будь-які хмарні програми та тіньові IT, які використовуються у вашій мережі
- Okta – керує входом у всі хмарні програми, включаючи Google Apps, Salesforce, Workday, Box, SAP, Oracle і Office 365
- Proofpoint – фокусується саме на електронній пошті для захисту вхідних і вихідних даних
- Qualys – сканує будь-які та всі використовувані веб-програми на наявність уразливостей у інструментах SaaS, IaaS та PaaS
- SilverSky – пропонує моніторинг електронної пошти та захист мережі для відповідності HIPAA та PCI
- Skyhigh Networks – виявляє, аналізує та захищає хмарні програми за допомогою журналів із існуючих брандмауерів, проксі та шлюзів
- Vaultive – шифрує будь-які дані, що надходять із мережі до програми
- Zscaler – відстежує весь трафік, який надходить і виходить з вашої мережі, а також захищає пристрої iOS і Android



Брокер безпеки доступу до хмари (CASB)

Програмне забезпечення Cloud Access Security Broker (CASB) з’явилося, щоб допомогти IT-спеціалістам впоратися з повною ситуацією безпеки в хмарі. CASB — це точки застосування політики безпеки між користувачами хмарної служби та одним або кількома постачальниками хмарної служби. Вони можуть перебувати на території підприємства або хмарний провайдер може розмістити їх. У будь-якому випадку CASB надають фахівцям з інформаційної безпеки критичну точку контролю для безпечного та сумісного використання хмарних служб у кількох хмарних постачальників. Вони забезпечують виконання багатьох рівнів політики безпеки підприємства, коли користувачі, пристрої та інші хмарні об’єкти намагаються отримати доступ до хмарних ресурсів.



CASB надає підприємствам критично важливу контрольну точку для безпечного використання хмарних служб у багатьох хмарних постачальників. Додатки програмного забезпечення як послуги (SaaS) стають всепоширеними на підприємствах, що посилює розчарування команд безпеки, які шукають видимість і контроль над цими програмами.

Продажі CASB різко зросли, оскільки зросли занепокоєння щодо хмарної безпеки, особливо використання хмарних служб Shadow IT, про які не знають команди IT-безпеки.

Рішення CASB заповнюють багато прогалин у безпеці в окремих хмарних службах і дозволяють фахівцям з інформаційної безпеки робити це через хмарні служби, включаючи постачальників інфраструктури як послуги (IaaS) і платформи як послуги (PaaS). Таким чином, CASB задовольняють критичні вимоги підприємства щодо встановлення політики, моніторингу поведінки та управління ризиками для всього набору корпоративних хмарних сервісів, які використовуються.

Що таке CASB

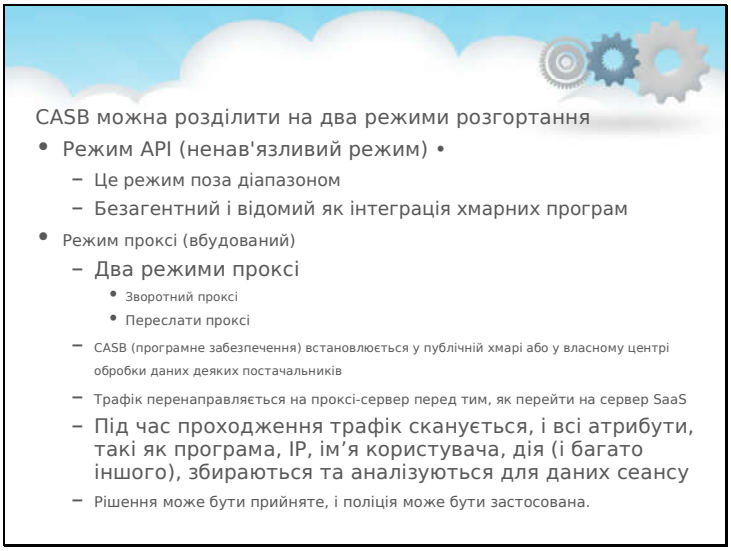
CASB може консолідувати кілька типів застосування політики безпеки. Приклади політик безпеки, які застосовуються CASB, включають автентифікацію, єдиний вхід, авторизацію, відображення облікових даних, профілювання пристрою, шифрування, токенізацію, ведення журналів, сповіщення, а також виявлення та запобігання зловмисному програмному забезпеченню.

Постачальник CASB також надає підприємствам доступ до авторизованого та неавторизованого використання хмари. Він може перехоплювати та контролювати трафік даних між корпоративною мережею та хмарною платформою, допомагати у вирішенні проблем відповідності, пропонувати дані

застосування політики безпеки та запобігання доступу неавторизованих пристроїв, користувачів і програм до хмарних служб.

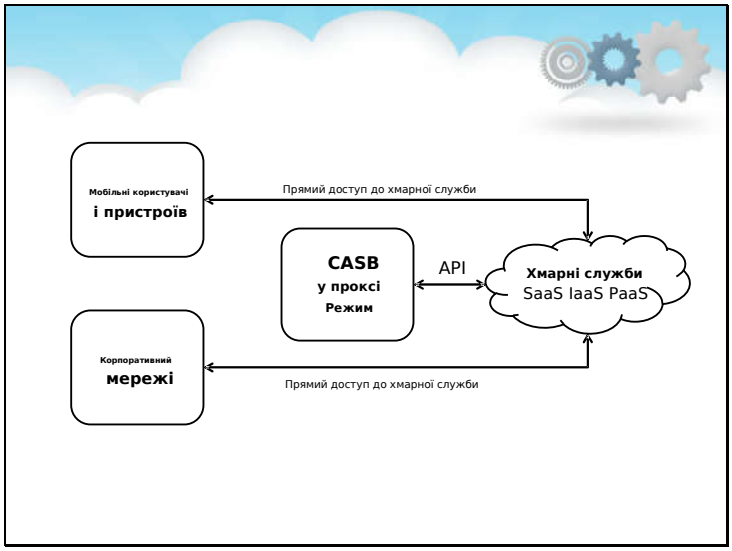
У надзвичайно важливій сфері безпеки даних провайдер CASB забезпечує дотримання корпоративних політик безпеки даних, щоб запобігти небажаній активності на основі класифікації даних, виявлення даних і моніторингу активності користувачів. Політики застосовуються за допомогою елементів керування, таких як перевірки, сповіщення, блокування, карантин, видалення та шифрування на рівні поля та файлу в службах хмарного хостингу.

Рішення CASB включають контроль і моніторинг, управління ризиками та відповідністю, захист від загроз і безпеку хмарних даних.



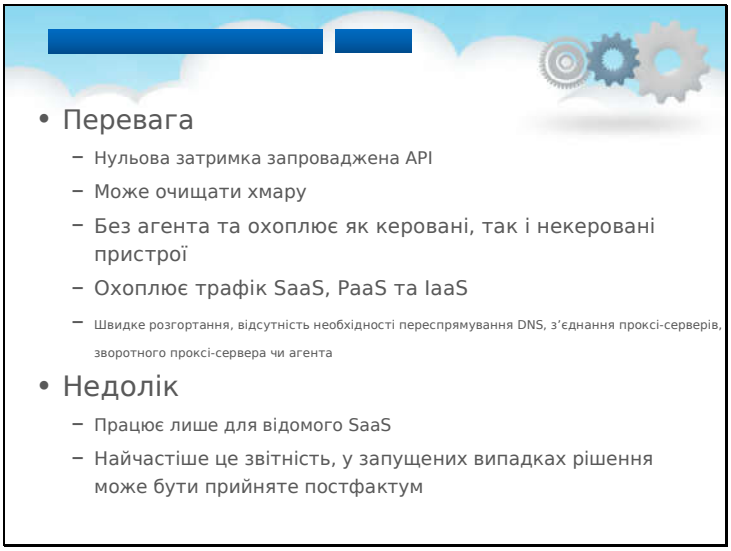
CASB можна розділити на два режими розгортання

- Режим API (ненав'язливий режим) •
  - Це режим поза діапазоном
  - Безагентний і відомий як інтеграція хмарних програм
- Режим проксі (вбудований)
  - Два режими проксі
    - Зворотний проксі
    - Переслати проксі
  - CASB (програмне забезпечення) встановлюється у публічній хмарі або у власному центрі обробки даних деяких постачальників
  - Трафік перенаправляється на проксі-сервер перед тим, як перейти на сервер SaaS
  - Під час проходження трафік сканується, і всі атрибути, такі як програма, IP, ім’я користувача, дія (і багато іншого), збираються та аналізуються для даних сеансу
  - Рішення може бути прийняте, і поліція може бути застосована.

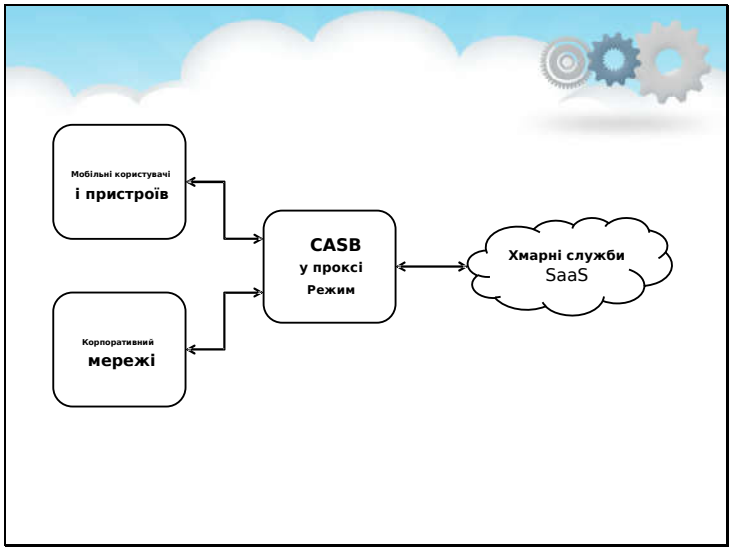


CASB на основі API – це позасмугове рішення, яке не має того самого мережевого шляху, що й дані. Оскільки рішення інтегрується безпосередньо з хмарними службами, рішення на основі API не мають зниження продуктивності, і вони забезпечують як керований, так і некерований трафік між хмарними службами Saas, IaaS і PaaS.

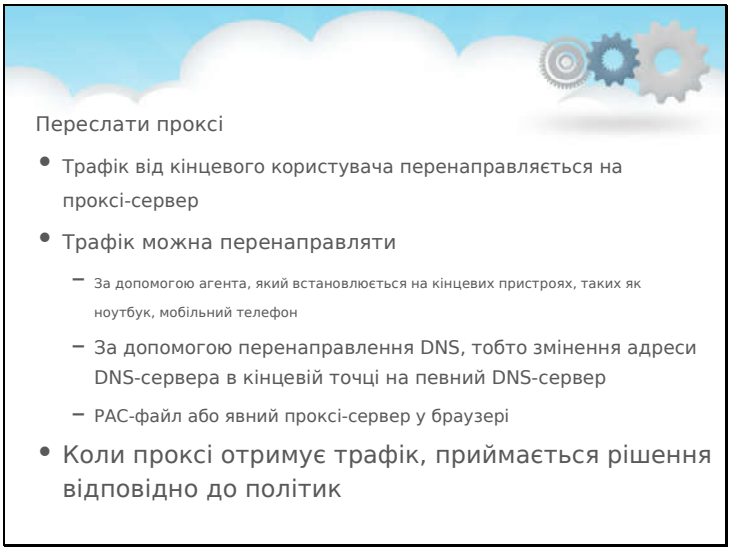
- Позасмугове розгортання
- Найкраще використовувати для очищення хмари
- Інтеграція API для відомих програм SaaS
- API сканує хмару для історичних даних для програми SaaS і застосовує політику для DLP, недійсного спільного доступу або виявлення зловмисного програмного забезпечення
- Контроль можна застосувати до будь-якої майбутньої дії
- На основі опитувань
  - Коли працівник спостерігає за хмарою, система сповістить про будь-які зміни
  - Зміну буде відскановано та застосовано правила •
- Режим зворотного дзвінка
  - Деякі хмарні програми підтримують API, у такому випадку SaaS повідомляє про будь-які значні зміни



- Перевага
  - Нульова затримка запроваджена API
  - Може очищати хмару
  - Без агента та охоплює як керовані, так і некеровані пристрої
  - Охоплює трафік SaaS, PaaS та IaaS
  - Швидке розгортання, відсутність необхідності переспрямування DNS, з'єднання проксі-серверів, зворотного проксі-сервера чи агента
- Недолік
  - Працює лише для відомого SaaS
  - Найчастіше це звітність, у запущених випадках рішення може бути прийняте постфактум

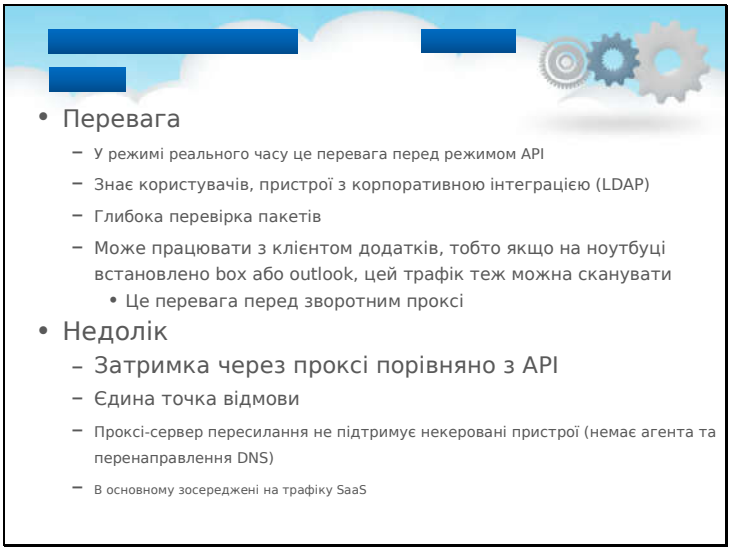


Вбудоване проксі-рішення перевіряє та фільтрує відомих користувачів і пристрої через єдиний шлюз. Оскільки весь трафік проходить через одну контрольну точку, проксі-сервер може вживати заходів безпеки в режимі реального часу. На жаль, єдина контрольна точка також означає, що вона сповільнює продуктивність мережі та захищає лише відомих користувачів. Крім того, рішення на основі проксі захищають лише хмарні служби SaaS, залишаючи хмари IaaS і PaaS уразливими.

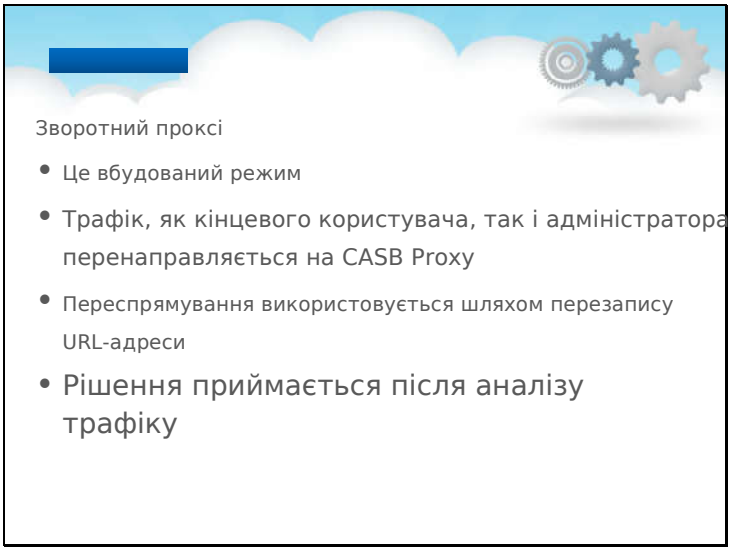


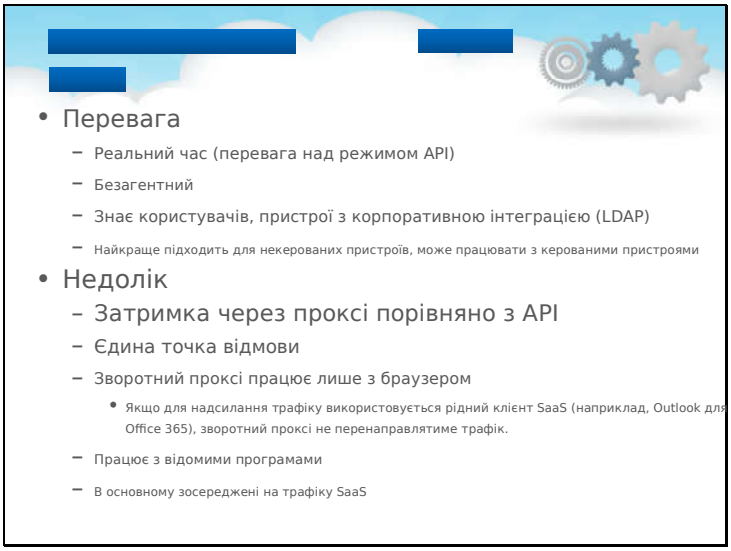
Переслати проксі

- Трафік від кінцевого користувача перенаправляється на проксі-сервер
- Трафік можна перенаправляти
  - За допомогою агента, який встановлюється на кінцевих пристроях, таких як ноутбук, мобільний телефон
  - За допомогою перенаправлення DNS, тобто змінення адреси DNS-сервера в кінцевій точці на певний DNS-сервер
  - PAC-файл або явний проксі-сервер у браузері
- Коли проксі отримує трафік, приймається рішення відповідно до політик

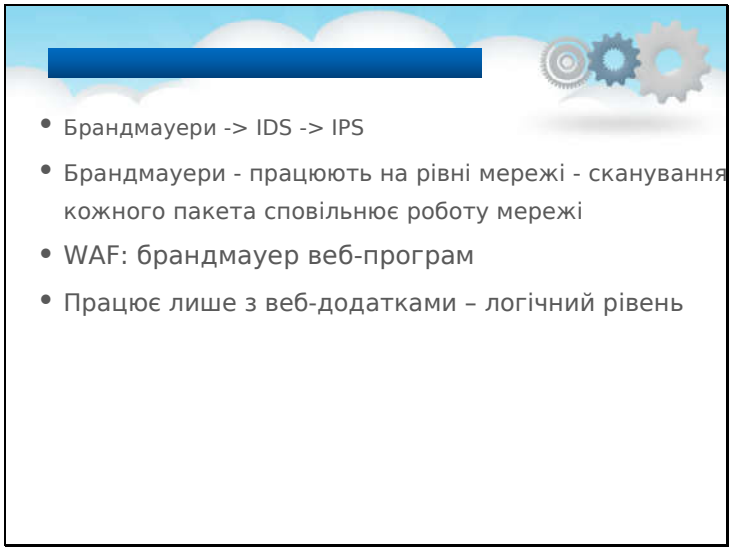


- Перевага
  - У режимі реального часу це перевага перед режимом API
  - Знає користувачів, пристрої з корпоративною інтеграцією (LDAP)
  - Глибока перевірка пакетів
  - Може працювати з клієнтом додатків, тобто якщо на ноутбуці встановлено box або outlook, цей трафік теж можна сканувати
    - Це перевага перед зворотним проксі
- Недолік
  - Затримка через проксі порівняно з API
  - Єдина точка відмови
  - Проксі-сервер пересилання не підтримує некеровані пристрої (немає агента та перенаправлення DNS)
  - В основному зосереджені на трафіку SaaS


Зворотний проксі
• Це вбудований режим
• Трафік, як кінцевого користувача, так і адміністратора, перенаправляється на CASB Proxy
• Переспрямування використовується шляхом перезапису URL-адреси
• Рішення приймається після аналізу трафіку



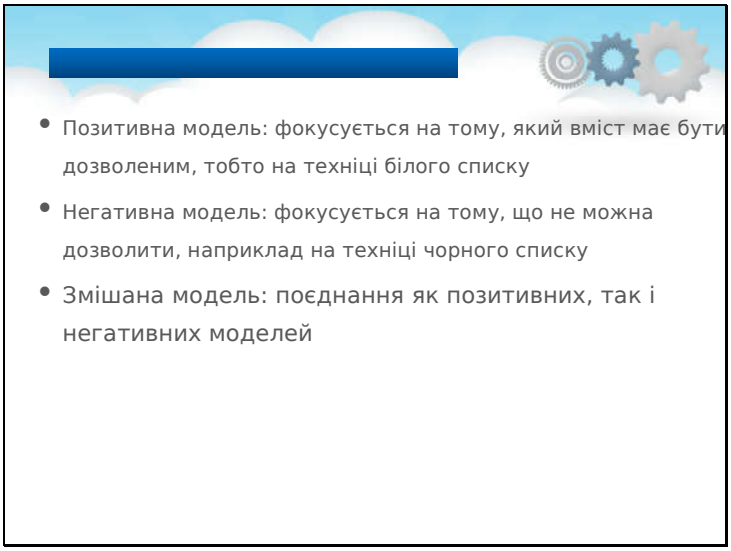
- Перевага
  - Реальний час (перевага над режимом API)
  - Безагентний
  - Знає користувачів, пристрої з корпоративною інтеграцією (LDAP)
  - Найкраще підходить для некерованих пристроїв, може працювати з керованими пристроями
- Недолік
  - Затримка через проксі порівняно з API
  - Єдина точка відмови
  - Зворотний проксі працює лише з браузером
    - Якщо для надсилання трафіку використовується рідний клієнт SaaS (наприклад, Outlook для Office 365), зворотний проксі не перенаправлятиме трафік.
  - Працює з відомими програмами
  - В основному зосереджені на трафіку SaaS



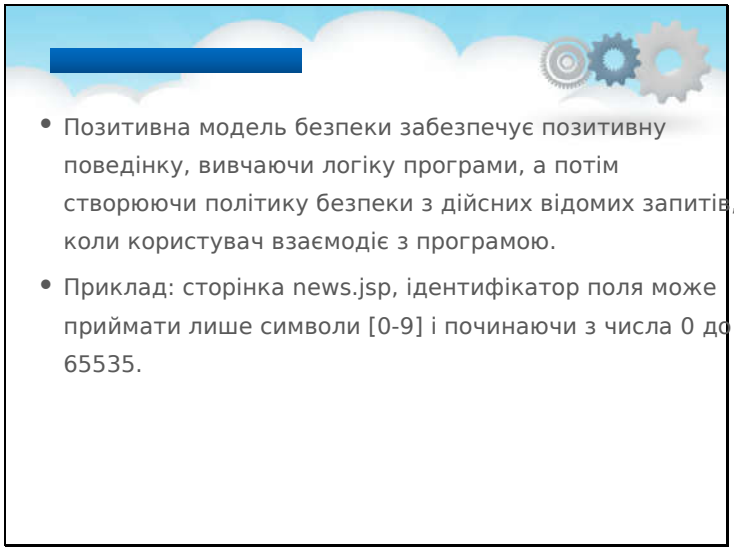
Брандмауер веб-додатків

Брандмауер веб-програми (або WAF) фільтрує, відстежує та блокує трафік HTTP до та з веб-програми. WAF відрізняється від звичайного брандмауера тим, що WAF може фільтрувати вміст певних веб-додатків, тоді як звичайні брандмауери служать воротами безпеки між серверами. Перевіряючи HTTP-трафік, він може запобігти атакам, спричиненим недоліками безпеки веб-додатків, такими як впровадження SQL, міжсайтовий сценарій (XSS), включення файлів і неправильні конфігурації безпеки.

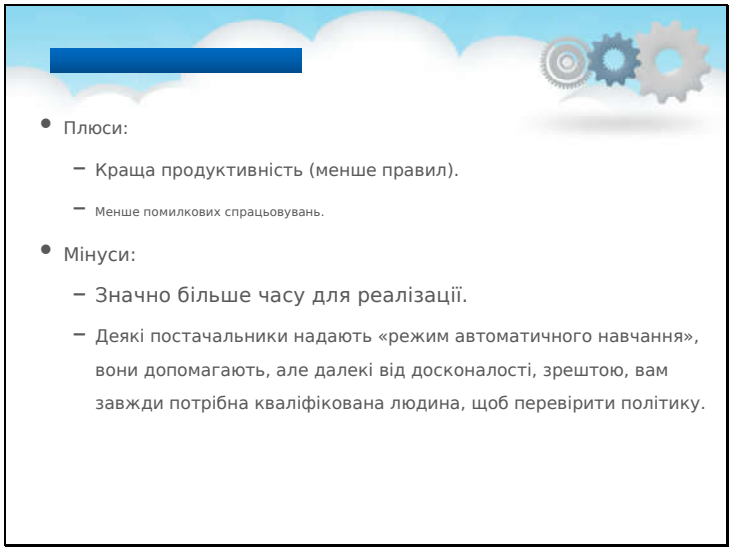
- Брандмауери веб-додатків на основі пристроїв: переважно апаратне забезпечення
  - Приклади: Netscaler MPX WAF від Citrix
- Хмарні та гібридні брандмауери веб-додатків: уся інфраструктура спільно з постачальниками WAF, захист від DDoS. Гібридні рішення чудово підходять для розподіленому середовищі (наприклад, кілька офісів) або коли віртуальне розгортання має сенс для організації.
  - Приклади: Cloud WAF: провідна в галузі служба WAF Incapsula, продукт WAF від Qualys є прикладом гібридного підходу віртуального пристрою/хмари.



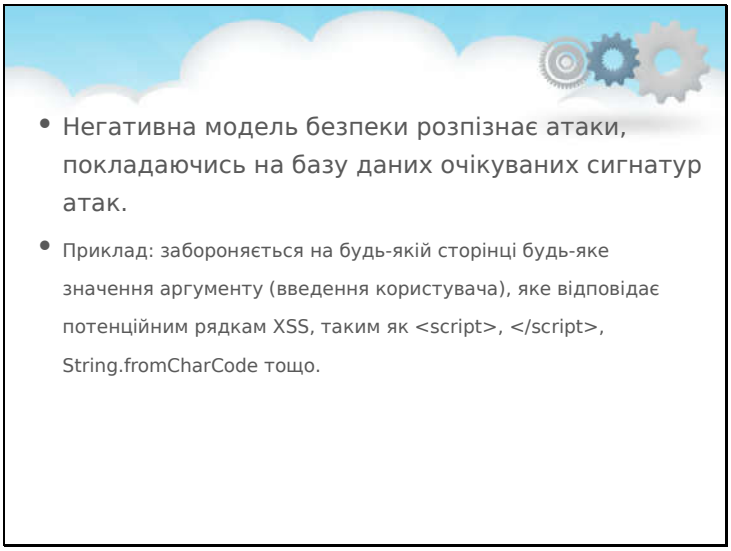
- Позитивна модель: фокусується на тому, який вміст має бути дозволеним, тобто на техніці білого списку
- Негативна модель: фокусується на тому, що не можна дозволити, наприклад на техніці чорного списку
- Змішана модель: поєднання як позитивних, так і негативних моделей



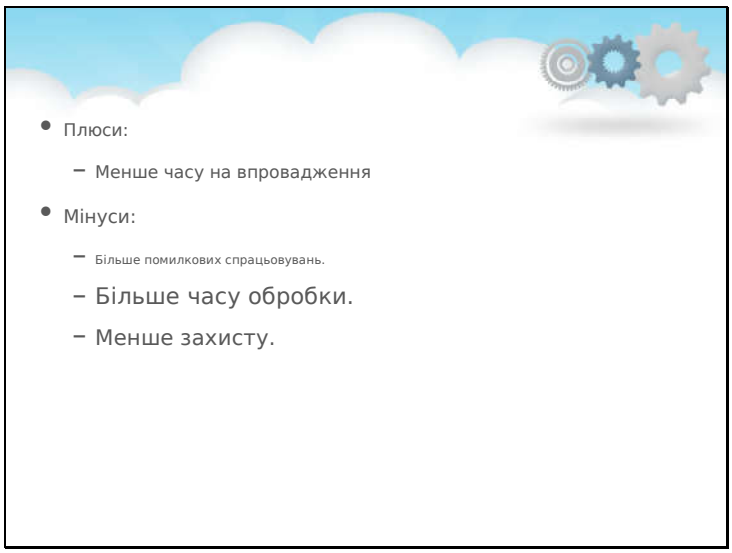
- Позитивна модель безпеки забезпечує позитивну поведінку, вивчаючи логіку програми, а потім створюючи політику безпеки з дійсних відомих запитів, коли користувач взаємодіє з програмою.
- Приклад: сторінка news.jsp, ідентифікатор поля може приймати лише символи [0-9] і починаючи з числа 0 до 65535.



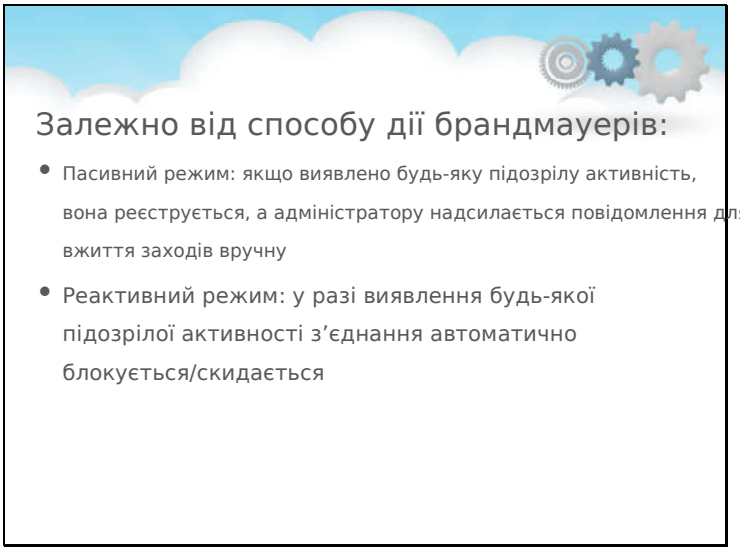
- Плюси:
  - Краща продуктивність (менше правил).
  - Менше помилкових спрацьовувань.
- Мінуси:
  - Значно більше часу для реалізації.
  - Деякі постачальники надають «режим автоматичного навчання», вони допомагають, але далекі від досконалості, зрештою, вам завжди потрібна кваліфікована людина, щоб перевірити політику.



- Негативна модель безпеки розпізнає атаки, покладаючись на базу даних очікуваних сигнатур атак.
- Приклад: забороняється на будь-якій сторінці будь-яке значення аргументу (введення користувача), яке відповідає потенційним рядкам XSS, таким як <script>, </script>, String.fromCharCode тощо.

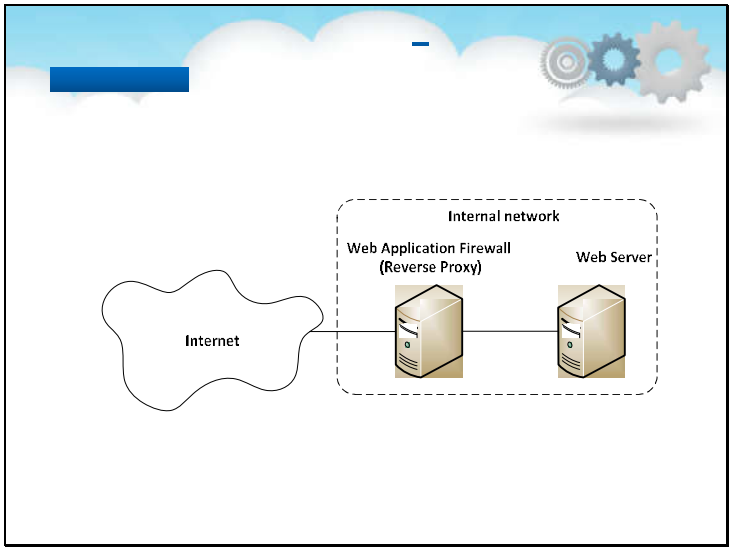


- Плюси:
  - Менше часу на впровадження
- Мінуси:
  - Більше помилкових спрацьовувань.
  - Більше часу обробки.
  - Менше захисту.



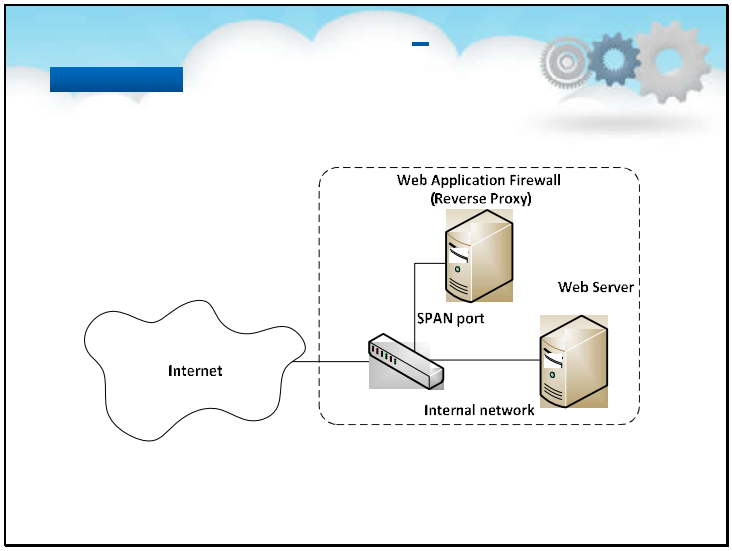
Залежно від способу дії брандмауерів:

- Пасивний режим: якщо виявлено будь-яку підозрілу активність, вона реєструється, а адміністратору надсилається повідомлення для вжиття заходів вручну
- Реактивний режим: у разі виявлення будь-якої підозрілої активності з’єднання автоматично блокується/скидається



У комп’ютерних мережах зворотний проксі — це тип проксі-сервера, який отримує ресурси від імені клієнта з одного або кількох серверів. Потім ці ресурси повертаються клієнту, виглядаючи так, ніби вони походять із самого проксі-сервера. На відміну від прямого проксі, який є посередником для зв’язаних із ним клієнтів для зв’язку з будь-яким сервером, зворотний проксі є посередником для зв’язаних із ним серверів, з якими може зв’язатися будь-який клієнт.

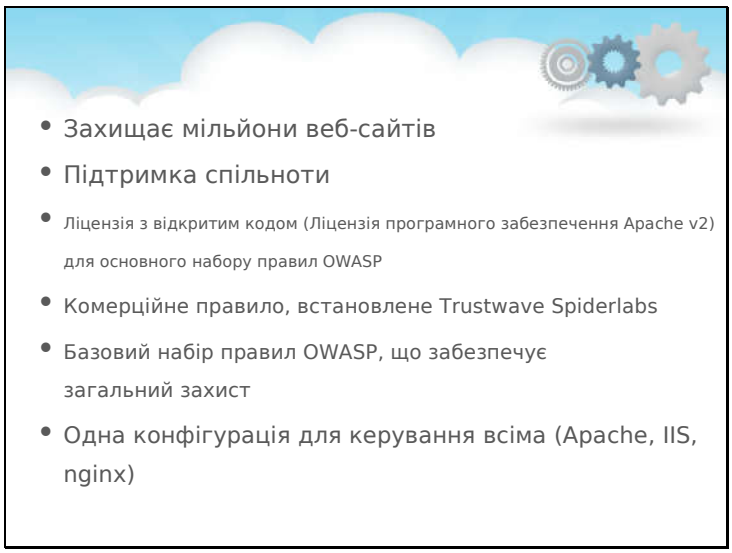
Досить часто популярні веб-сервери використовують функцію зворотного проксі-сервера, захищаючи рамки програм від слабших можливостей HTTP.



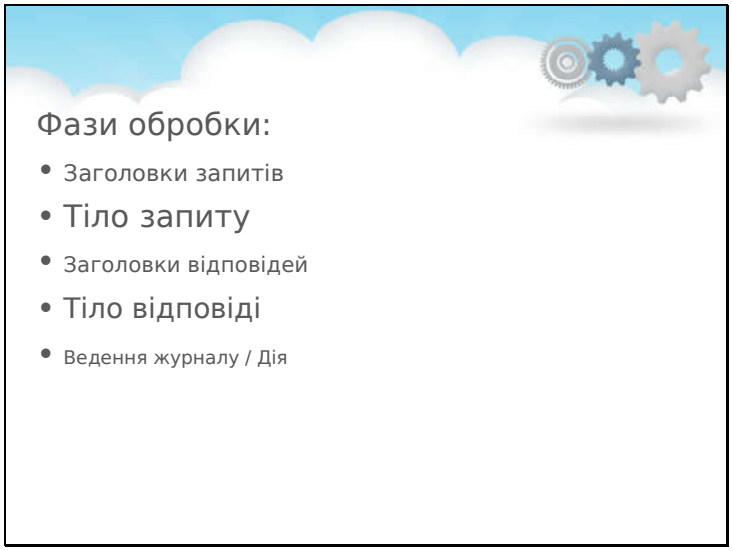
WAF у цьому режимі розгортання забезпечує лише функції моніторингу, він використовує порт SPAN на комутаторі мережі для отримання всього трафіку.

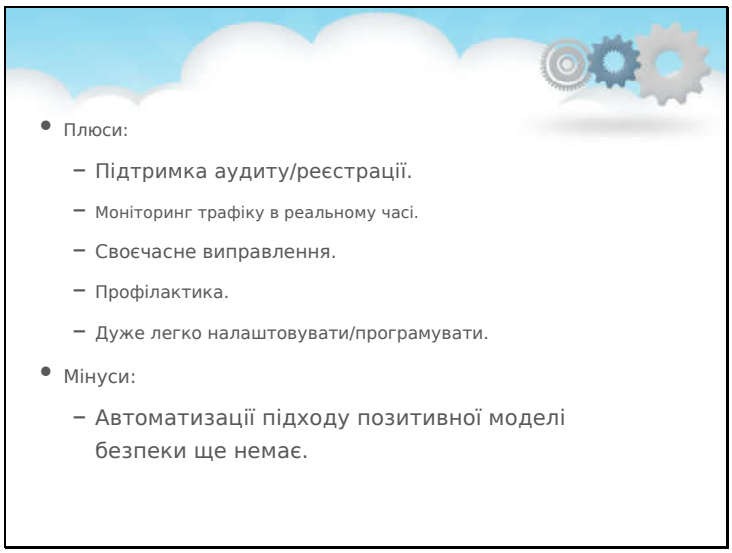
Дзеркалювання портів, також відоме як SPAN (Switched Port Analyzer), — це метод моніторингу мережевого трафіку. Якщо ввімкнуто дзеркальне відображення портів, комутатор надсилає копію всіх мережевих пакетів, які бачать один порт (або цілу VLAN), до іншого порту, де пакет можна проаналізувати.

Функція Port Mirroring підтримується майже всіма комутаторами корпоративного класу (керованими комутаторами).



- Захищає мільйони веб-сайтів
- Підтримка спільноти
- Ліцензія з відкритим кодом (Ліцензія програмного забезпечення Apache v2) для основного набору правил OWASP
- Комерційне правило, встановлене Trustwave Spiderlabs
- Базовий набір правил OWASP, що забезпечує загальний захист
- Одна конфігурація для керування всіма (Apache, IIS, nginx)


Фази обробки:
• Заголовки запитів
• Тіло запиту
• Заголовки відповідей
• Тіло відповіді
• Ведення журналу / Дія



- Плюси:
  - Підтримка аудиту/реєстрації.
  - Моніторинг трафіку в реальному часі.
  - Своєчасне виправлення.
  - Профілактика.
  - Дуже легко налаштовувати/програмувати.
- Мінуси:
  - Автоматизації підходу позитивної моделі безпеки ще немає.