

Big data architecture

INTRODUCTION TO THE VIRTUALIZATION TECHNOLOGY

概述

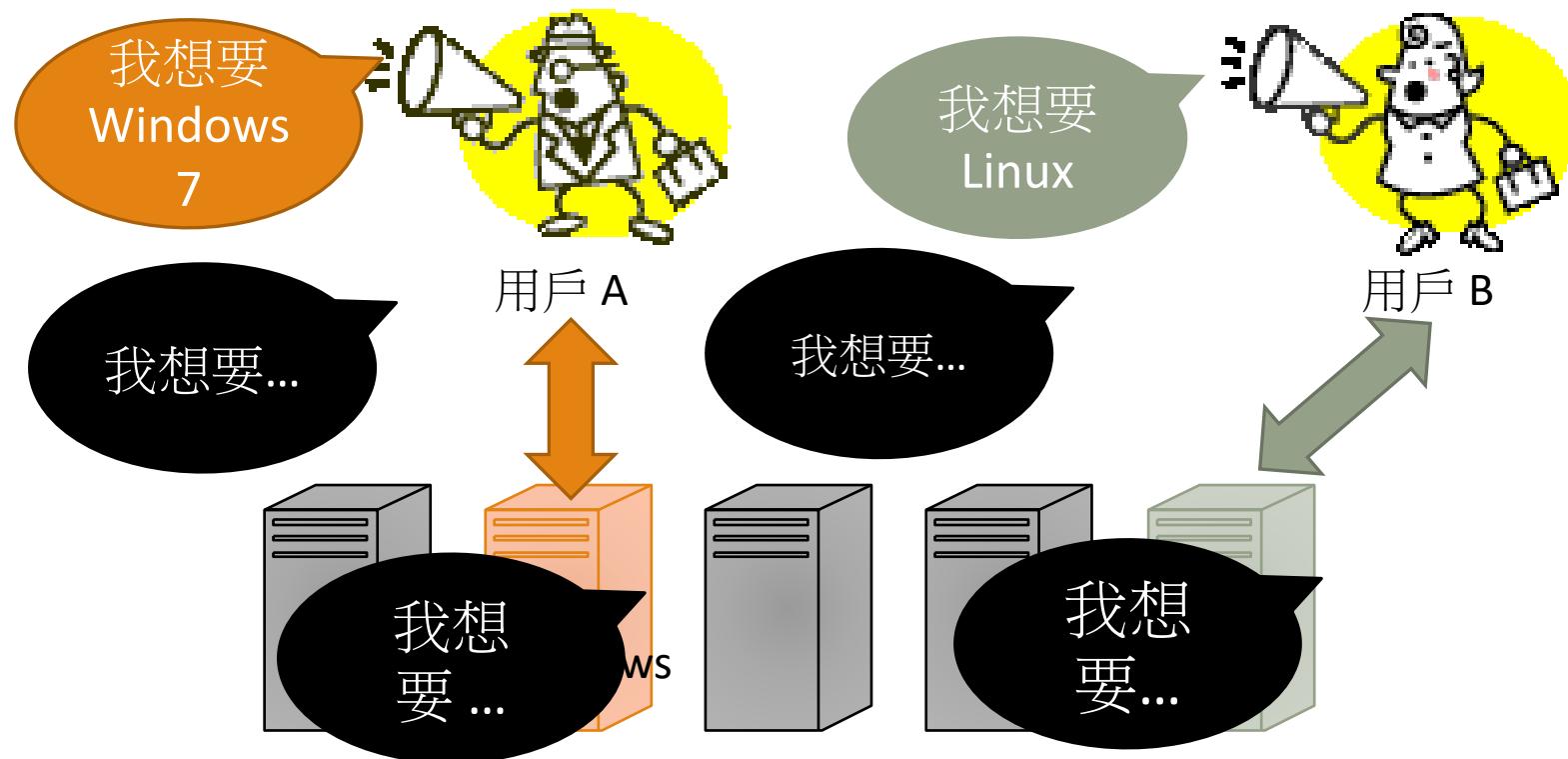
□ 傳統的資源管理方式會有什麼潛在的問題？

- 公司或企業的資訊部門大多是對於可能面臨的最大負載進行採購
- 資源管理方面缺乏靈活性
- 每個公司都存在基礎設施的維護成本
- 時常會面臨硬體故障的危機

□ 種種潛在的因素，對於一般公司和企業的資源使用而言，都是一項不願意面對的窘境

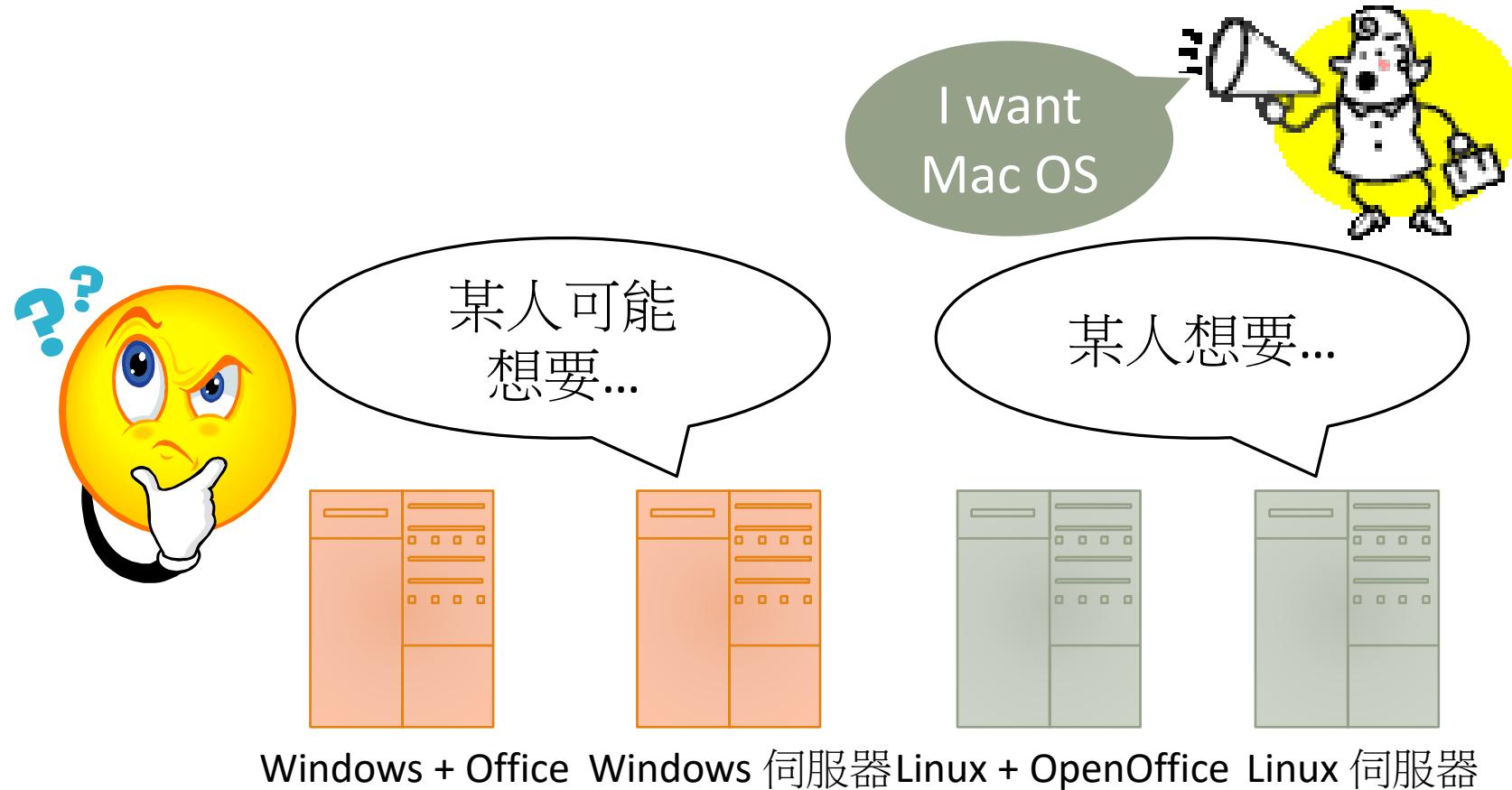
虛擬化 (2/4)

如果我們給每位新來的使用者都配置一個新的實體機器，那會發生什麼情況？

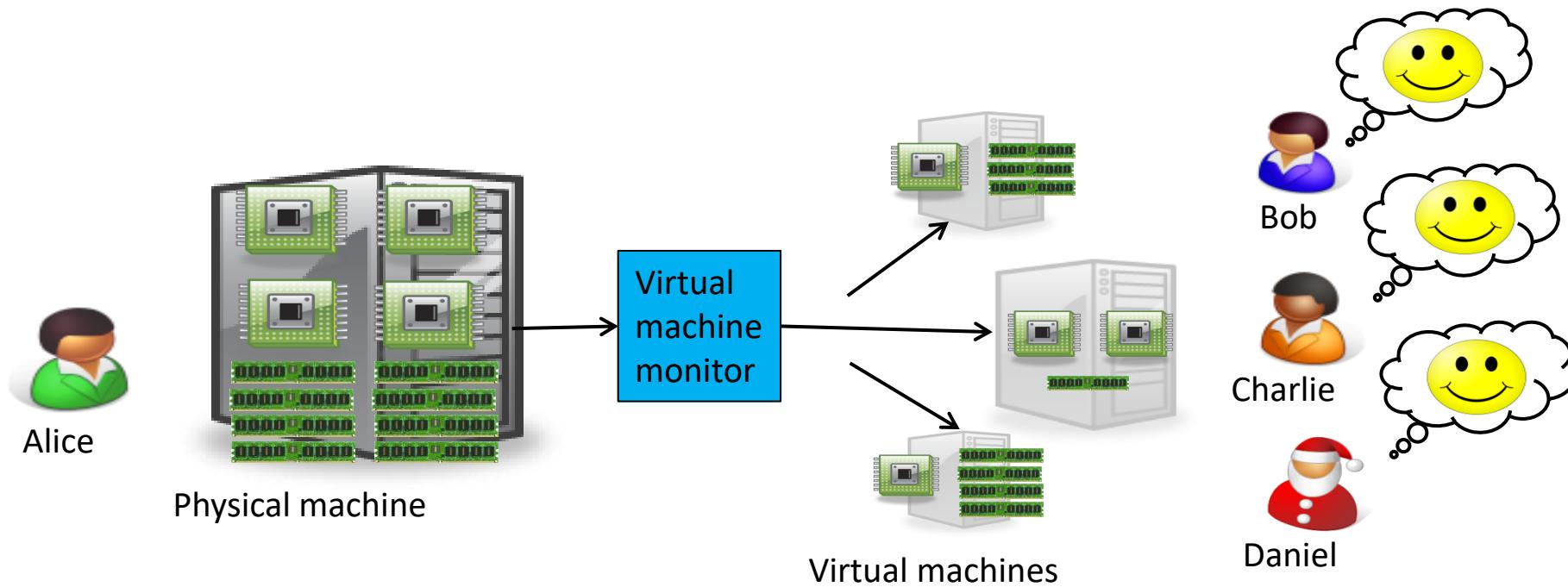


虛擬化 (3/4)

那如果準備好一堆事先安裝的實體機器應付各種需求，又會如何呢？



虛擬化 (3/4)



虛擬化 (4/4)

1. 很明顯地，上述所提的兩種策略都沒用
2. 我們需要更多強大的技術來處理這個問題
3. 虛擬化(virtualization)技術可以幫得上忙
 - 針對計算的資源
 - 伺服器虛擬化技術
 - 針對儲存資源
 - 儲存虛擬化的技術
 - 針對通訊
 - 網路虛擬化的技術

虛擬化概述

系統的抽象化

- 計算機系統是建立在抽象層之上
- 高的抽象層將實作細節隱藏在低的抽象層
- 每一個抽象層次的設計者，將使用底層所支援的功能，同時將自己的功能提供給上一層的抽象層

例如：

- 檔案是硬碟的一種抽象化

虛擬化概述

5G行動通訊系統將是一個全方位服務多技術融合的網路，來滿足未來包含廣泛資料和連接的各種業務的快速發展需要，關鍵就在於

軟體定義網路(Software-Defined Networking, SDN)

網路功能虛擬化(Network-Function Virtualization, NFV)。

網路虛擬化其實是通過在**流量層面**邏輯地劃分網路，以在現有網路中建立邏輯網段，這類似於硬碟的分割。

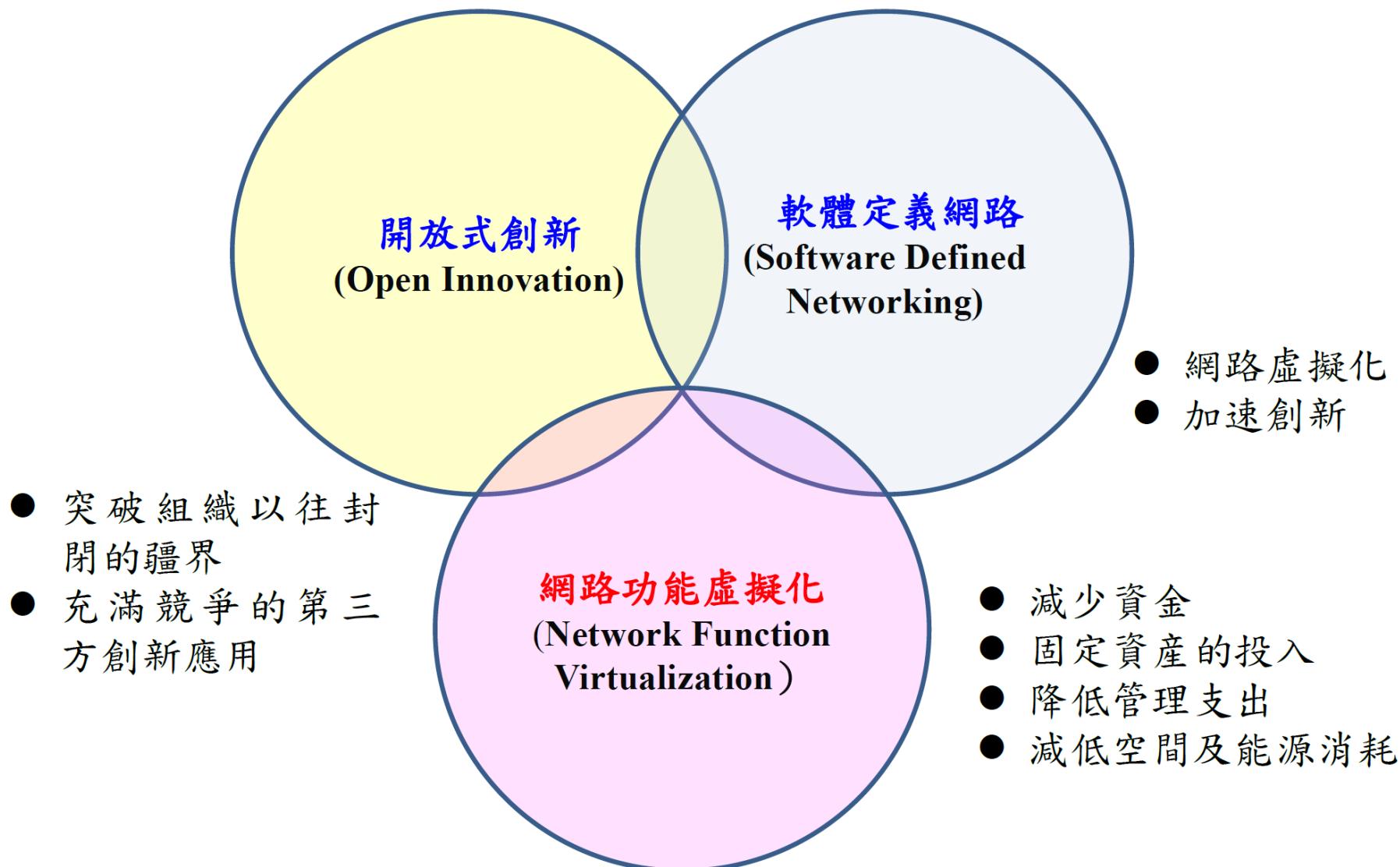
網路功能虛擬化 (Network Functions Virtualization, NFV)

- 將以往必須建基於硬件形式的各種技術虛擬化
 - 負載平衡(Load Balance)、防火牆(Firewall)、IPS (Intrusion Prevention System)
- 減少硬件成本，包括部署、硬件使用的電力、數據中心空間
- 一種可程式化控制的新型網路基礎配置，將原本的專屬硬體與功能，改為軟體功能，建置在通用高容量伺服器上
 - 企業標準
 - 多樣性網路服務



	傳統網路	NFV
作法	<ul style="list-style-type: none"> 分散式的架構 專屬功能有專屬硬體機器 	<ul style="list-style-type: none"> 集中式架構 利用虛擬化平台提供專屬機器的網路功能
優點	<ul style="list-style-type: none"> 從硬體設計來達成網路傳遞訊息的行為，並且提升交易的處理速度 	<ul style="list-style-type: none"> 從虛擬化平台來構成網路佈建網路傳遞處理伺服器，透過虛擬化平台彈性調整效能。
缺點	<ul style="list-style-type: none"> 當需要改變訊息傳遞行為時必須逐一進行設定或者更換原來的設備，如此將耗費非常多的人力和成本 	<ul style="list-style-type: none"> 初期軟硬體設備投資高，並受限於虛擬化平台的基礎硬體環境

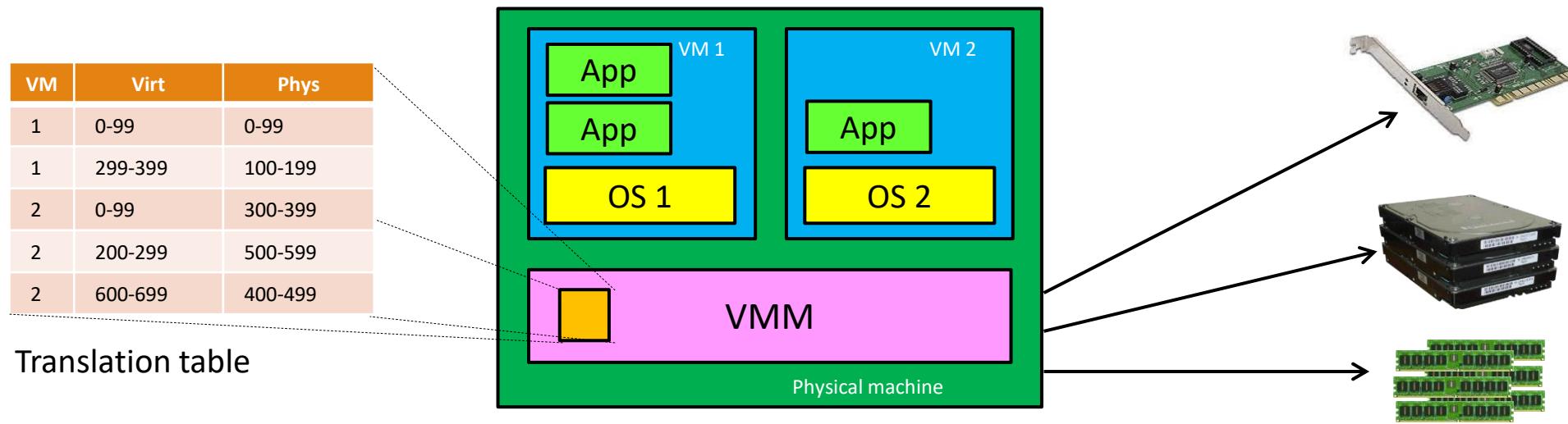
NFV V.S. SDN



NFV V.S. SDN

	SDN	NFV
作法	管理網路的控制與傳送封包的轉送分開，讓資訊技術對於網路行為與效能擁有更大的控制能力。	將那些傳統上與特殊化專屬硬體相關聯的網路程序轉換成可以在標準商用硬體執行的虛擬化軟體平台。
效用	集中化控制及可程式設計能解決快速成長和分散化網路在管理上所涉及的複雜性。	網路功能可依需求在網路內移動向上或向下擴展，不會因安裝新硬體裝置，而造成延誤與成本負擔。
好處	允許在軟體和硬體方面做出個別的採購決策。	促成快速且低成本的功能升級，達到快速的服務創新。

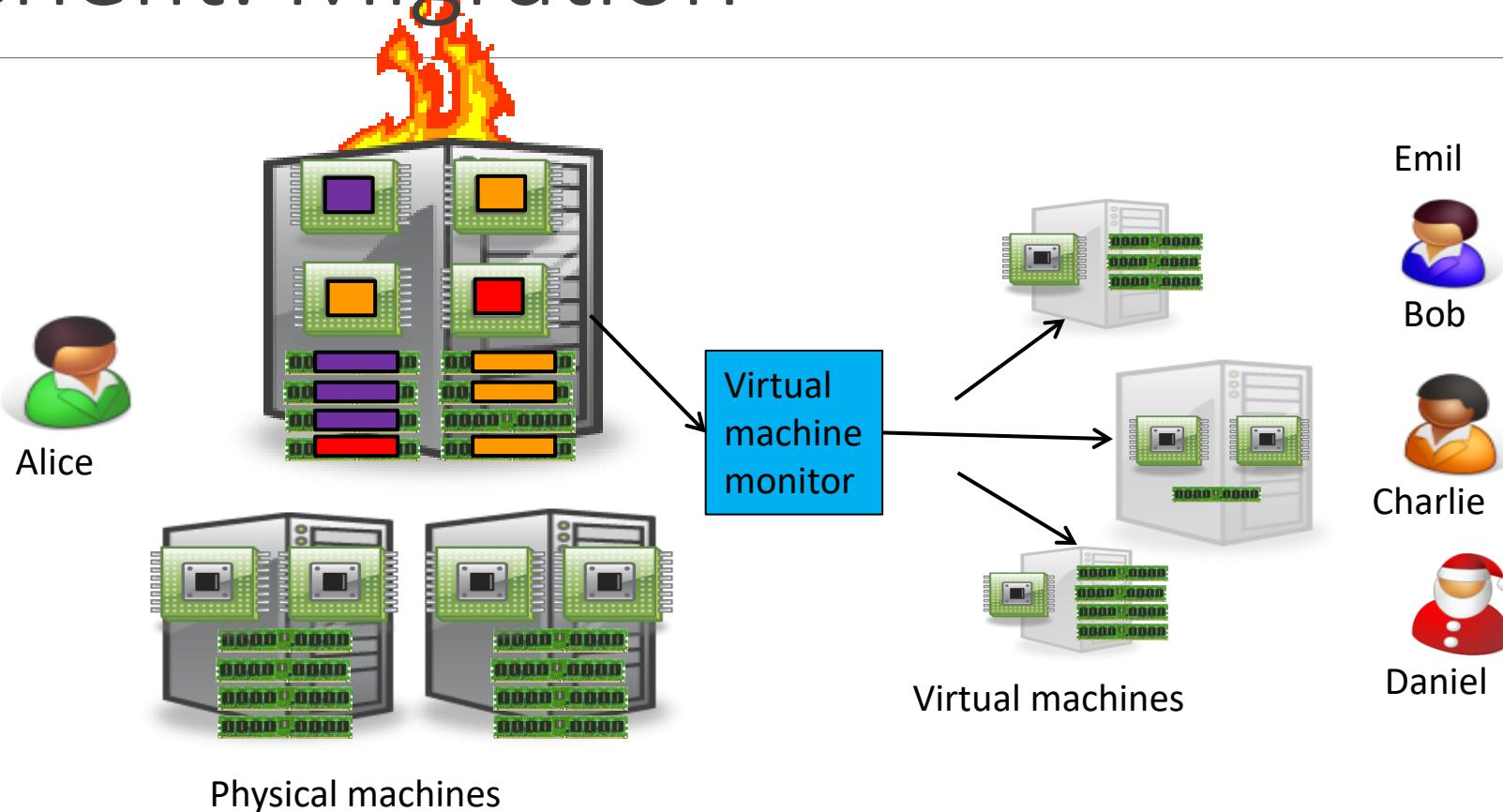
虛擬化技術？



Resources (CPU, memory, ...) are virtualized

- VMM ("Hypervisor") has translation tables that map requests for **virtual resources** to **physical resources**
- Example: VM 1 accesses memory cell #323; VM2 maps this to memory cell # 123.

Benefit: Migration



What if the machine needs to be shut down?

- e.g., for maintenance, consolidation, ...
- Alice can **migrate** the VMs to different physical machines without any customers noticing

常見虛擬機器軟體比較

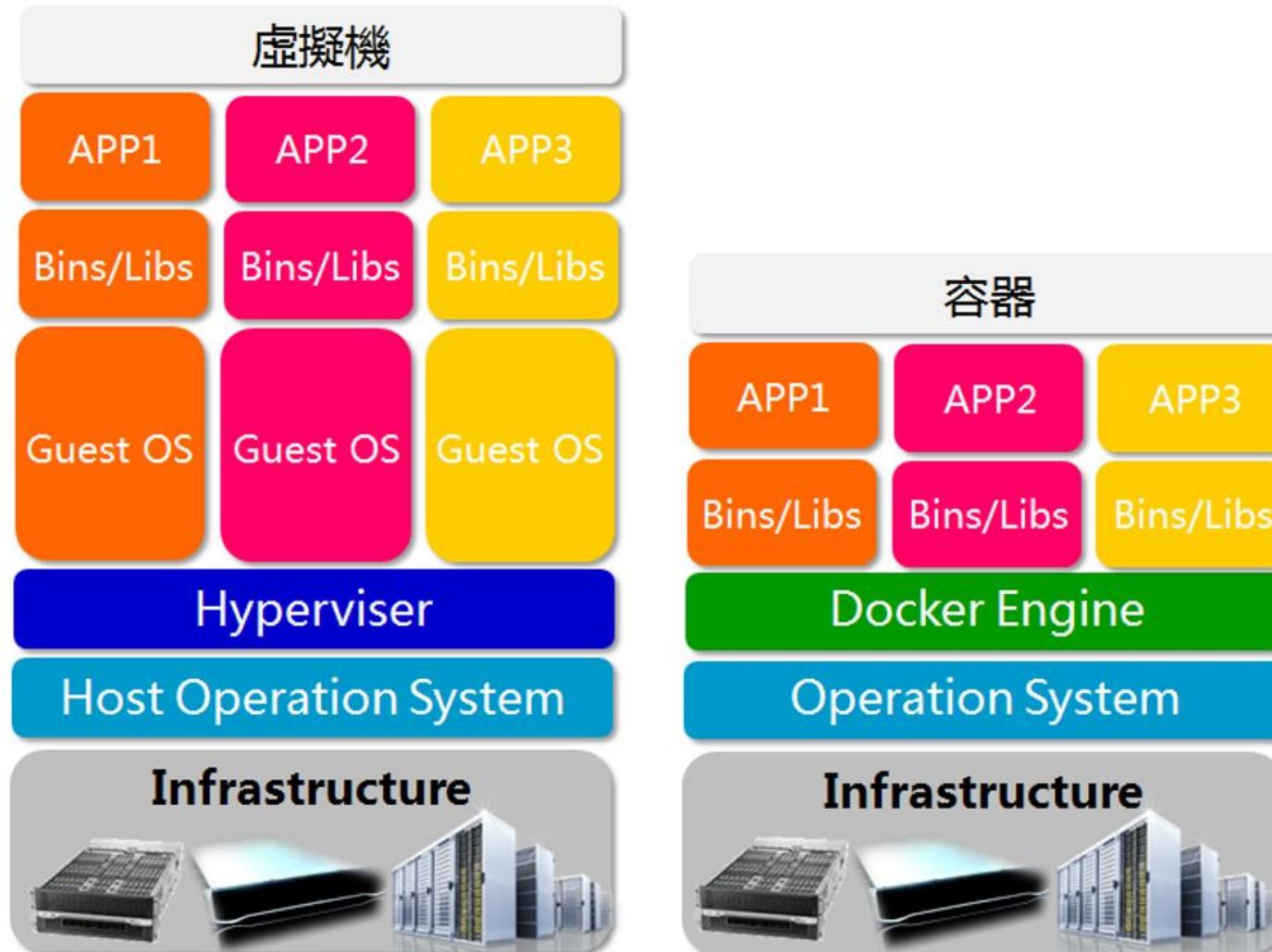
虛擬機器軟體 名稱	公司	虛擬硬碟 副檔名	其他
VMware 	VMware	VMDK	最早的虛擬機器 VMware Player → 免費版
VirtualBox 	甲骨文(Oracle)	VDI	開放原始碼、 體積小、速度快
VirtualPC 	微軟(Microsoft)	VHD VHDX	
Hyper-V			

雲端運算下容器(Container)技術和應用

雲端運算下容器:Docker , openstack

- 容器(Container):傳統運輸領域的貨櫃，
- 承載貨櫃的港口:雲端服務提供商
- 貨櫃的輪船:雲端服務所提供的基礎架構即服務(Infrastructure as a Service, IaaS)服務。
- Docker 以容器為核心的資訊技術 (Information Technology, IT)交付與運營系統。
 - Docker Engine (容器的運營管理)、Docker Registry (容器的分發管理) 以及相關應用程式介面 (Application Programming Interface, API)。
- Docker 技術提供的是 CasS 服務 (Container-as-Service, CasS)
- Docker 容器可以提供平台即服務(Platform as a Service, PaaS)和軟體即服務(Software as a Service, SaaS)服務
 - CaaS 位於 IaaS 之上，在 PaaS 、SaaS 之下
 - Docker利用Linux容器 (Linux Containers, LXC) 技術來實現類似虛擬機(Virtual Machine, VM)的功能，從而利用更加節省的硬體資源提供給使用者更多的 運算資源。
 - 一台虛擬機可以運營多個容器，與虛擬機相比，容器輕量級、啟動速度快、隔離性好等優點。

雲端運算下容器(Container)



CONDA虛擬環境



Before



After



創建環境 `conda create -n environment_name python=3` (版本)

啟動環境 `conda activate environment_name`

退出環境 `deactivate`

列舉環境 `conda env list`

刪除環境 `conda env remove -n environment_name`

假設我們需要建立一個名為py27的python2.7版本環境，只需要在cmd中輸入

`conda create --name py27 python=2.7`

啟動py27這個環境之後，會發現開啟的是python2.7版本

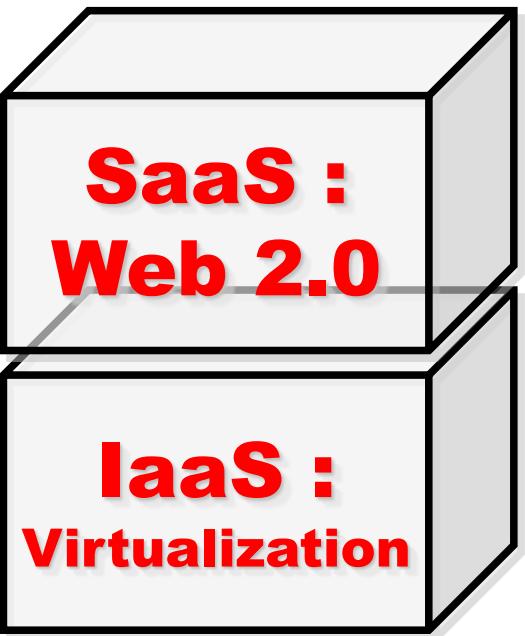
`conda activate py27`

退出虛擬環境

`conda deactivate`

Two Type of Cloud Architecture ?

雲端架構的兩大陣營？

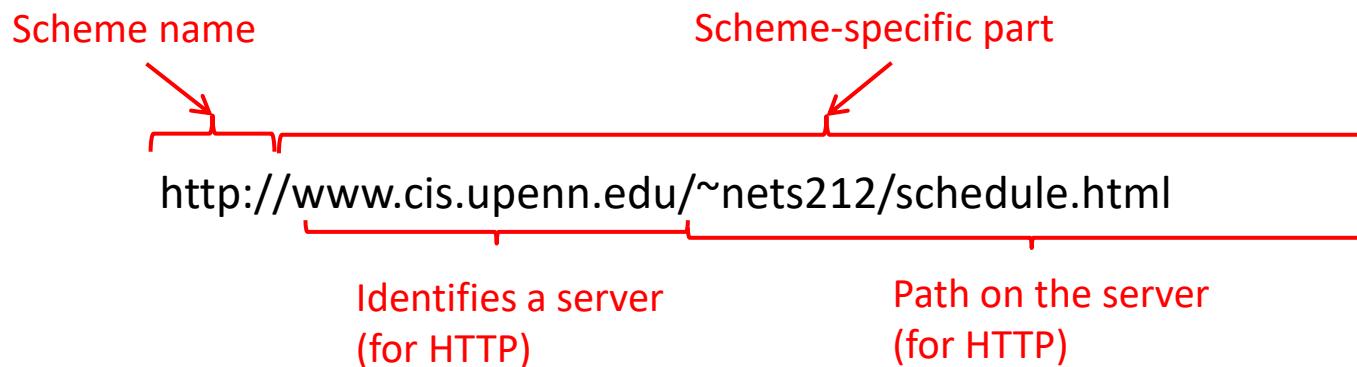


想盡辦法誘你用計算跟網路
Computing Intensive



想盡辦法誘你提供資料作分析
Data Intensive

URIs, URNs, and URLs



Uniform Resource Identifier (URI)

- Comes in two forms: URN and URL

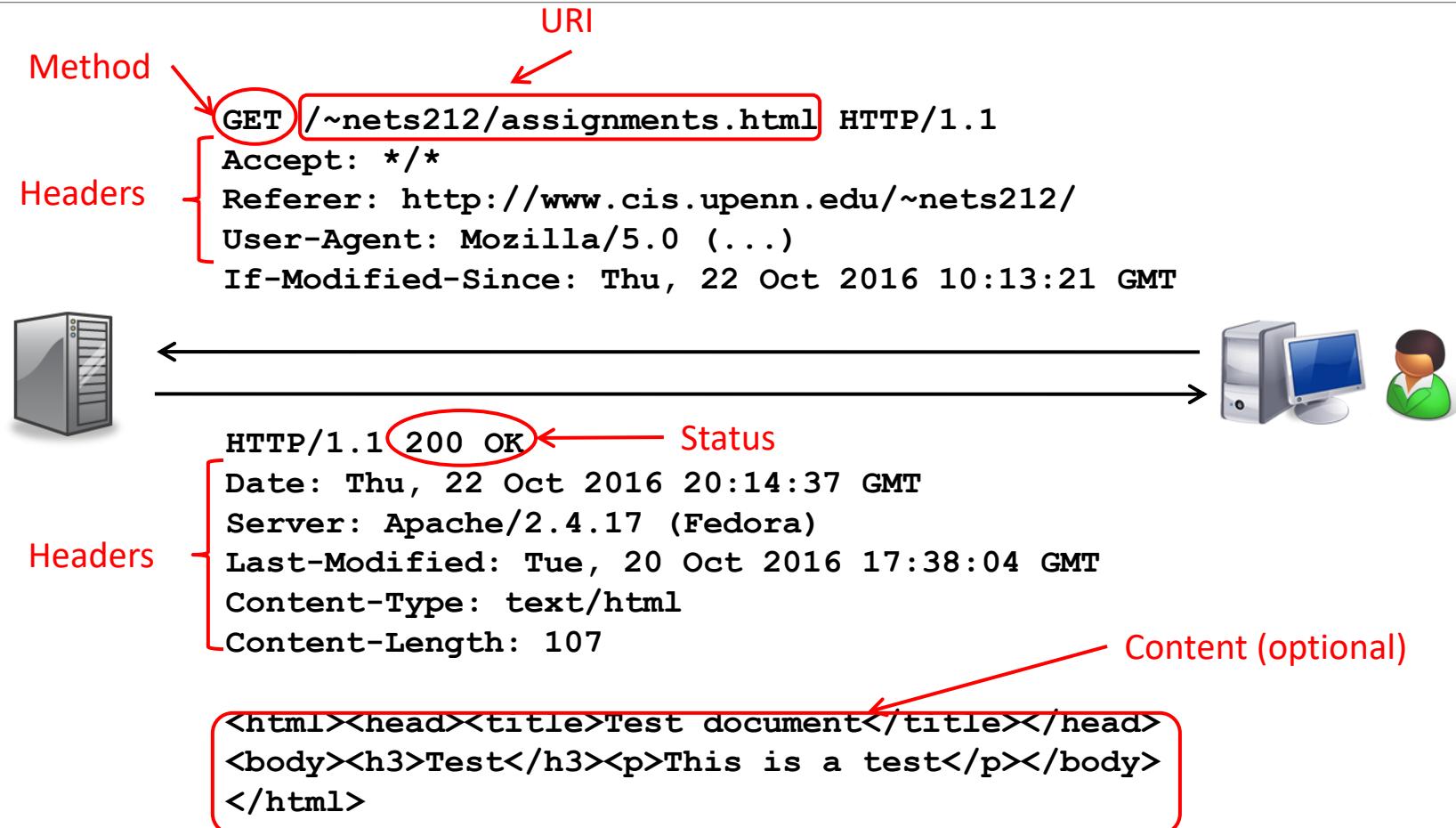
Uniform Resource Name (URN)

- Specifies *what* to find, independent of its location
- Example: `urn:isbn:1449311520` (for the course textbook)

Uniform Resource Locator (URL)

- Specifies *where* to find something
- `<scheme>://[user[:password]@]<server>[:port]/[path][/resource][?param1=value1¶m2=value2&...]`

Example: A simple HTTP request



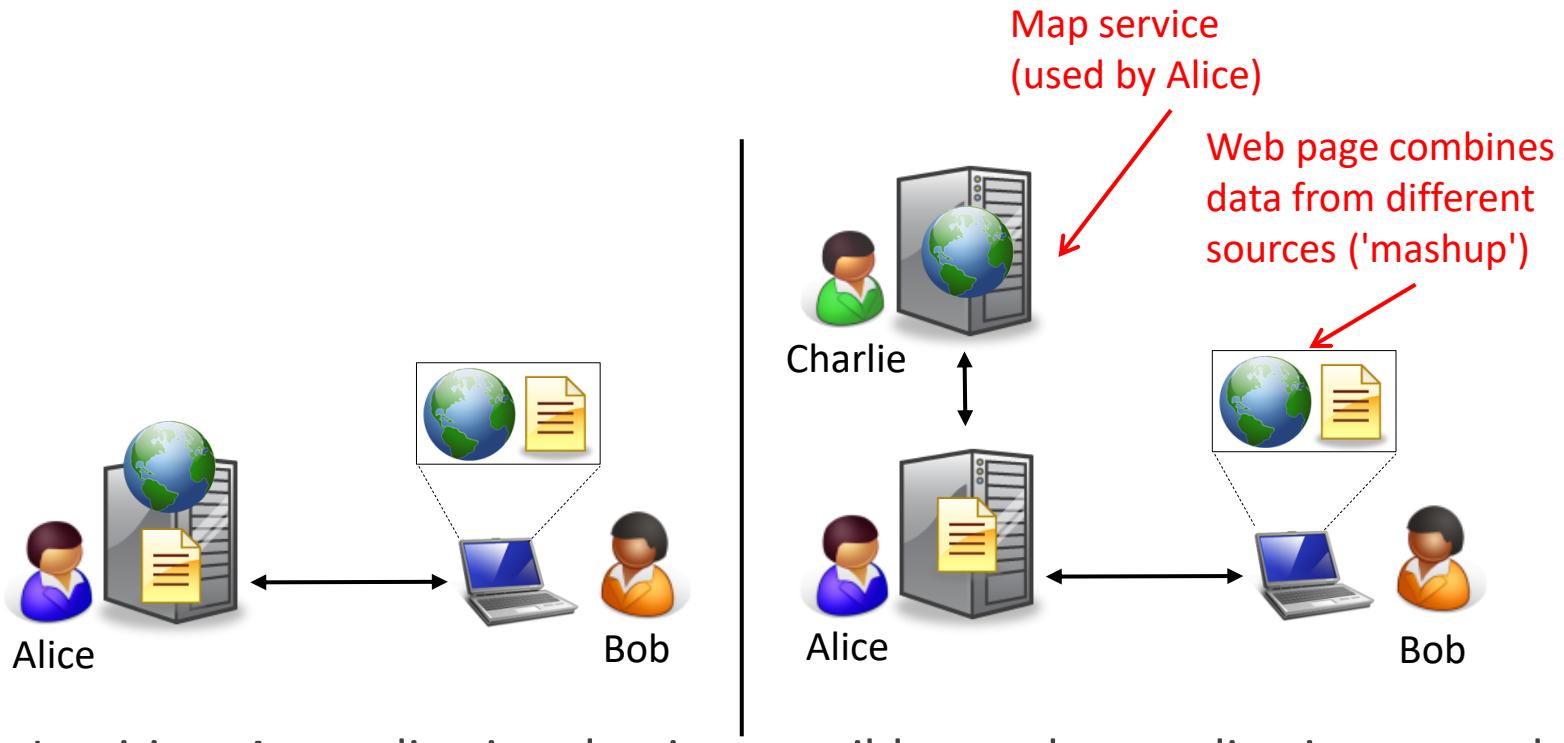
Status codes

Server sends back a status code to report how the request was processed

Common status codes:

- 200 OK
- 301 Moved Permanently
- 304 Not Modified
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

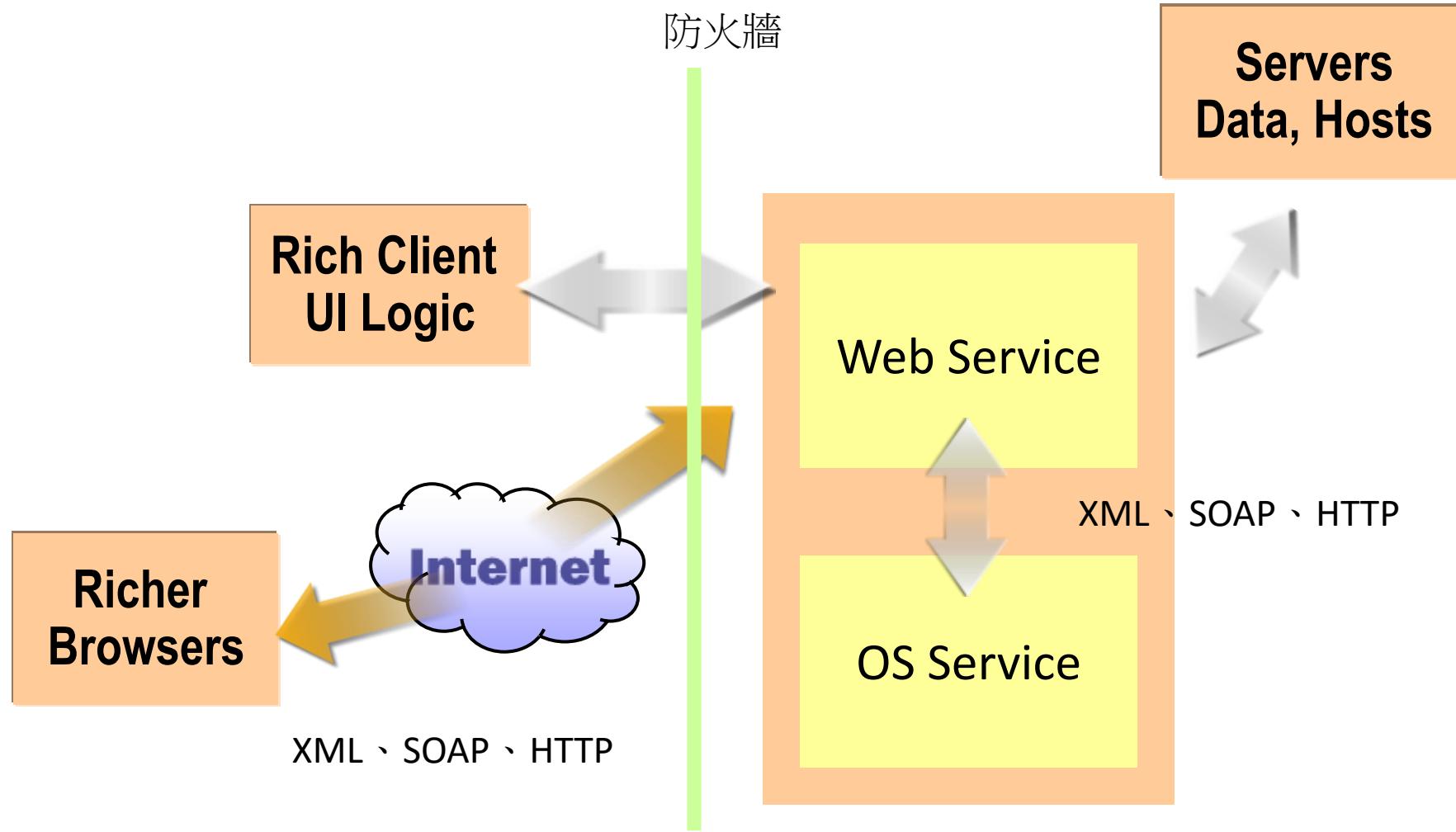
What is a web service?



Intuition: An application that is accessible to other applications over the web

- Examples: Google Maps API, Facebook Graph API, eBay APIs, Amazon Web Services APIs, ...

Web Services



Resources Requesting/Managing – Data

For web services, standardizing information exchanging helps simplifying communications between services.

In 1998, Microsoft and UserLand Software created the **XML-RPC** protocol, and supported by other companies.

SOAP is based on **XML-RPC**, with more flexibility, and become the part of W3C standard in 2000

JSON is another protocol describing data by using JavaScript

Resources Requesting/Managing – REST

OpenStack, AWS all support RESTful API

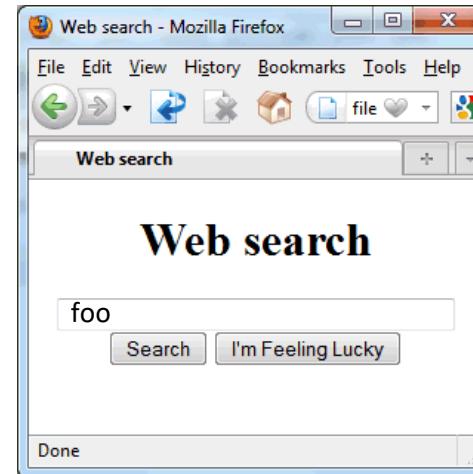
Example for OpenStack

- POST http://localhost/v1/AUTH_admin/mycontainer
 - Create a container “mycontainer”
- PUT file1 http://localhost/v1/AUTH_admin/mycontainer/
 - Upload a file “file1” to “mycontainer”
- GET http://localhost/v1/AUTH_admin/mycontainer/?prefix=fi
 - List files in “mycontainer” with prefix fi
- GET http://localhost/v1/AUTH_admin/mycontainer/file1
 - Download the file “file1”

The RESTful way makes better and scalable interface for a system

Forms and GET/POST

```
<html>
  <head><title>Web search</title></head>
  <body>
    <center><h1>Web search</h1>
    <form action="search.html" method="<u>GET</u>">
      <input type="text" size="40" name="term"><br>
      <input type="submit" value="Search">
      <input type="button" value="I'm Feeling Lucky">
    </form></center>
  </body>
</html>
```



What happens when we hit 'Search'?

With method="get":

```
GET /search.html?term=foo HTTP/1.1
Accept: text/html
```

With method="post":

```
POST /search.html HTTP/1.1
Accept: text/html
term=foo
```

IBM Visual recognition

Try the service

Choose a sample image or upload your own image to try out Visual Recognition.



Or paste an image URL

Watson sees...



JSON ↗

Classes	Score
tutti-frutti	0.58
ice cream	0.59
dessert	0.76

JSON ↗

Food	Score
strawberries and cream	0.72
fruit dish	0.72

Image recognition cloud api

Google Cloud Vision is Google's visual recognition API, based on the open-source TensorFlow framework and using a **REST API**. It detects individual objects and faces and contains a pretty comprehensive set of labels, including Optical Character Recognition (OCR)

IBM Watson Visual Recognition is part of IBM's Watson Developer Cloud.

Amazon Rekognition is Amazon's image recognition API

Microsoft Computer Vision API

Clarif.ai is an upstart image recognition service that also uses a REST API.

CloudSight seems to take a slightly different approach to image recognition. their service seems to offer some combination of algorithmic and manual (meaning human) tagging.

Image recognition cloud api example

front_door_bike crop.jpg



msft

[View raw json](#)

msft_captions

a bicycle leaning against a wall (.80)

msft_tags

bicycle (1.00) , outdoor (.95) , parked (.87) , red (.84) , sidewalk (.73) , transport (.66) , seat (.54) , rack (.53) , basket (.53) , bicycle rack (.12) , bicycling (.08)

ibm

[View raw json](#)

ibm_tags

vehicle (1.00) , bicycle (1.00) , bike (.80)

google

[View raw json](#)

google_tags

bicycle (.98) , racing bicycle (.94) , vehicle (.94) , cyclo cross bicycle (.85) , sports equipment (.81) , road bicycle (.78) , cyclo cross (.75) , land vehicle (.72) , mode of transport (.70) , mountain bike (.70)

cloudsight

[View raw json](#)

cloudsight_captions

black and white douglas road bike

clarifai

[View raw json](#)

clarifai_tags

wheel (1.00) , bike (1.00) , no person (.99) , transportation system (.93) , vehicle (.91) , spoke (.91) , old (.90) , cyclist (.90) , leisure (.90) , recreation (.90) , outdoors (.90) , bicycle (.89) , tire (.88) , saddle (.87) , brake (.86) , road (.86) , wheel (.85) , chain (.83) , vintage (.82) , travel (.82)

Image recognition cloud api example

binoculars.jpg



msft

[View raw json](#)

msft_captions

a vase sitting on a desk (.39)

msft_tags

tree (.99) , indoor (.95) , window (.93)

ibm

[View raw json](#)

ibm_tags

vegetation (1.00)

google

[View raw json](#)

google_tags

furniture (.58)

cloudsight

[View raw json](#)

cloudsight_captions

black binocular on window glass

clarifai

[View raw json](#)

clarifai_tags

window (.94) , people (.92) , two (.92) , light (.91) , backlit (.90) , tree (.90) , seat (.89) , glass (.87) , adult (.87) , wood (.87) , girl (.87) , flower (.86) , container (.86) , daylight (.86) , technology (.85) , room (.84) , furniture (.84) , outdoors (.84) , still life (.83) , portrait (.83)

Image recognition cloud api example

subway.jpg



msft

[View raw json](#)

msft_captions

person waiting at a train station (.34)

msft_tags

platform (.97) , station (.85) , subway (.44)

ibm

[View raw json](#)

ibm_tags

train (.60) , station (.57)

google

[View raw json](#)

google_tags

transport (.98) , train station (.90) , rapid transit (.88) , metro station (.73) , subway (.66) , public transport (.65) , tgv (.63)

cloudsight

[View raw json](#)

Cloud Sight seems to be too good to be true

cloudsight_captions

people walking near subway

clarifai

[View raw json](#)

clarifai_tags

train (1.00) , tube (1.00) , railway (1.00) , blur (1.00) , station (1.00) , train (1.00) , transportation system (1.00) , commuter (.99) , tunnel (.99) , fast (.99) , platform (.99) , train station (.98) , railway (.98) , escalator (.97) , travel (.97) , motion (.97) , rush (.97) , movement (.96) , speed (.96) , urban (.95)

Image recognition cloud api example

collie_in_a_park.jpg



msft

[View raw json](#)

msft_captions

a dog catching a frisbee in a grassy field (.81)

msft_tags

grass (1.00), outdoor (1.00), dog (1.00), tree (1.00), frisbee (.99), field (.78), green (.63), grassy (.44), lush (.13)

ibm

[View raw json](#)

ibm_tags

animal (1.00), mammal (1.00), dog (.71), cat (.62)

google

[View raw json](#)

google_tags

pet (.98), dog (.96), alaskan malamute (.93), mammal (.93), animal (.91), vertebrate (.86), grass (.82), carnivoran (.73), puppy (.52)

cloudsight

[View raw json](#)

cloudsight_captions

white black and brown long coated dog catching a yellow freesbie on green grass field during daytime

clarifai

[View raw json](#)

clarifai_tags

grass (.99), mammal (.98), dog (.98), lawn (.96), animal (.95), pet (.94), nature (.93), portrait (.93), canine (.93), summer (.92), outdoors (.91), cute (.91), hayfield (.89), no person (.89), field (.86), young (.86), looking (.84), outside (.83), domestic (.82), one (.82)

collie.jpg



msft

[View raw json](#)

msft_captions

a dog catching a frisbee in a grassy field (.81)

msft_tags

grass (1.00), outdoor (1.00), dog (1.00), tree (1.00), frisbee (.99), field (.78), green (.63), grassy (.44), lush (.13)

ibm

[View raw json](#)

ibm_tags

animal (1.00), mammal (1.00), dog (.71), cat (.62)

google

[View raw json](#)

google_tags

pet (.98), dog (.96), alaskan malamute (.93), mammal (.93), animal (.91), vertebrate (.86), grass (.82), carnivoran (.73), puppy (.52)

cloudsight

[View raw json](#)

cloudsight_captions

white brown rough collie

clarifai

[View raw json](#)

clarifai_tags

grass (.99), mammal (.98), dog (.98), lawn (.96), animal (.95), pet (.94), nature (.93), portrait (.93), canine (.93), summer (.92), outdoors (.91), cute (.91), hayfield (.89), no person (.89), field (.86), young (.86), looking (.84), outside (.83), domestic (.82), one (.82)

Different filenames, and got different captions, suggesting that human labeling is involved
it's not fair to compare them against these other vendors from a purely machine driven methods.
The cost of the cloudsight is significantly higher at scale due to there's more manual tagging going on behind

Image recognition cloud api example

collie_in_a_park.jpg



msft

[View raw json](#)

msft_captions

a dog catching a frisbee in a grassy field (.81)

msft_tags

grass (.100) , outdoor (.100) , dog (.100) , tree (.100) , frisbee (.99) , field (.78) , green (.63) , grassy (.44) , lush (.13)

ibm

[View raw json](#)

ibm_tags

animal (.100) , mammal (.100) , dog (.71) , cat (.62)

google

[View raw json](#)

google_tags

pet (.98) , dog (.96) , alaskan malamute (.93) , mammal (.93) , animal (.91) , vertebrate (.86) , grass (.82) , carnivoran (.73) , puppy (.52)

cloudsight

[View raw json](#)

cloudsight_captions

white black and brown long coated dog catching a yellow freesbie on green grass field during daytime

clarifai

[View raw json](#)

clarifai_tags

grass (.99) , mammal (.98) , dog (.98) , lawn (.96) , animal (.95) , pet (.94) , nature (.93) , portrait (.93) , canine (.93) , summer (.92) , outdoors (.91) , cute (.91) , hayfield (.89) , no person (.89) , field (.86) , young (.86) , looking (.84) , outside (.83) , domestic (.82) , one (.82)

collie_in_a_park_rotated.jpg



msft

[View raw json](#)

msft_captions

a cat playing with a yellow frisbee (.46)

msft_tags

grass (.100) , cat (.88) , green (.72)

ibm

[View raw json](#)

ibm_tags

animal (.100) , mammal (.100) , cat (.50)

google

[View raw json](#)

google_tags

nature (.98) , color (.96) , white (.95) , green (.94) , wildlife (.93) , water (.91) , animal (.89) , vertebrate (.86) , grass (.82) , pet (.80)

cloudsight

[View raw json](#)

cloudsight_captions

brown black and white long coated dog playing frisbee rough collie

clarifai

[View raw json](#)

clarifai_tags

mammal (.97) , no person (.97) , outdoors (.97) , nature (.96) , animal (.96) , grass (.96) , water (.95) , wildlife (.91) , pet (.89) , summer (.88) , bird (.87) , daylight (.86) , park (.85) , lake (.84) , one (.84) , wild (.84) , landscape (.83) , color (.81) , river (.81) , pool (.79)

Image recognition cloud api

Feature Comparison

	Amazon	Google	Clarifai	Microsoft
Image Tagging	Yes	Yes	Yes	Yes
Video Tagging	No	No	Yes	Yes
Emotions detection	Yes	Yes	Yes	Yes
Logo detection	Yes	Yes	Yes	Yes
NSFW tagging	No	Yes	Yes	Yes
Dominant color	Yes	Yes	Yes	Yes
Feedback API	No	No	Yes	No

Pricing

Amazon Rekognition	\$1 / 1000 events	https://cl.ly/1e2R2d07lI2g
Google Vision API	\$1.5 / 1000 events	https://cl.ly/1W3h0P423N1J
Clarifai	\$19 / 20k, \$479 / 250k events	https://cl.ly/1M1P293n0C3E
Microsoft Computer Vision API	\$1.5 / 1000 Events	https://cl.ly/301X3i0q3U0W

Image Size Limits

Amazon Rekognition	5Mb / Image, 15Mb / Image from S3
Google Vision API	4 MB / Image
Clarifai	No data in documentation
Microsoft Computer Vision API	4 MB / Image
Rate limits	
Amazon Rekognition	Not defined in documentation
Google Vision API	10 Requests per second
Clarifai	Depends from plan (we get 30 rps for testing purposes)
Microsoft Computer Vision API	10 Requests per second

Performance Testing

MacBook Pro, Kraków, 1000 files, 10 at a time

	Average	Minimum	Maximum	90th percentile
Amazon	2.42s	1.03s	3.73s	3.21s
Google	1.23s	0.69s	1.68s	1.42s
Clarifai	4.69s	0.1s	58.16s	4.78s
Microsoft	1.11s	0.65s	5.07s	1.5s

Docker practice

資料庫多元性



最初表示“反SQL”運動
用新型的非關聯式資料庫取代關聯式資料庫

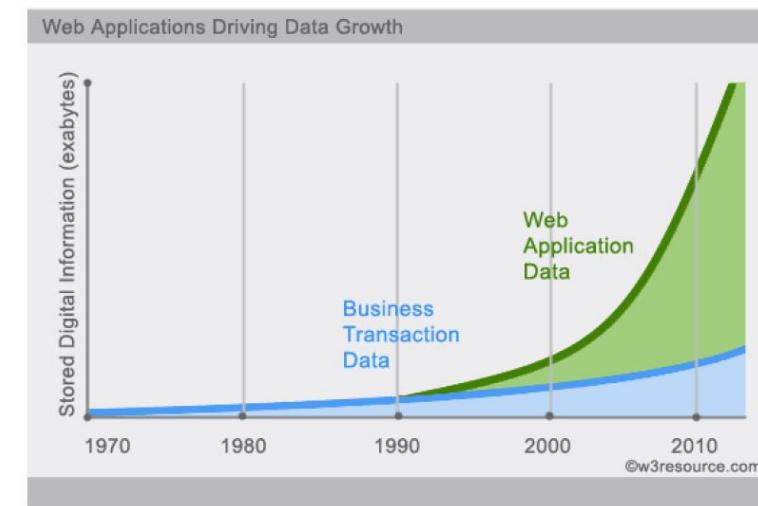
現在表示關聯式與非關聯式資料庫各有優缺點
彼此都無法互相取代

通常，NoSQL資料庫具有以下幾個特點：

- (1) 靈活的可擴展性：支援**橫向擴展**，不同於關聯式資料庫系統
- (2) 靈活的資料模型：關聯式資料庫是由代數理論發展，故其關聯式資料模型使用上有著嚴格的規範，如：在使用前需明確定義欄位有哪些、它們的資料型態...等，並需遵守多種限制條件，無法動態更改架構。而NoSQL則沒有太多限制
- (3) 與雲端運算緊密融合：可支援橫向擴展，故可結合雲端運算底層基礎架構

NOSQL 優勢

- 1、關聯式資料庫已經無法滿足Web2.0的需求。主要反應在以下方面：
- (1) 無法滿足**巨量資料的管理**需求
 - (2) 無法滿足**資料高動態即時**的需求
 - (3) 無法滿足**高可擴展性和高可用性**的需求



Web 2.0

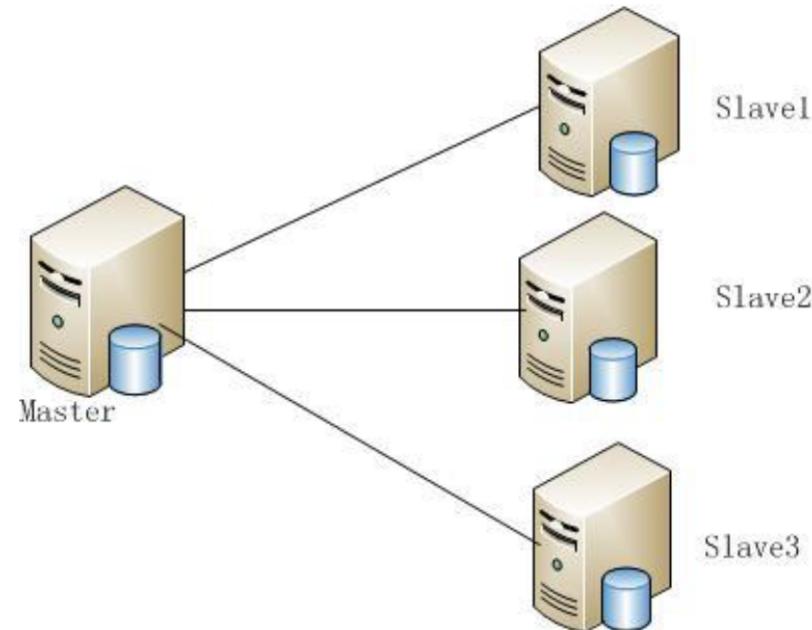
關聯式資料庫的關鍵特性包括**完善的交易機制**和**高效能的查詢機制**。但是，關聯式資料庫引以為傲的兩個關鍵特性，到了Web2.0時代卻成了雞肋，除了有一些額外成本外，主要表現在以下幾個方面：

- (1) Web2.0網站系統通常**不要求嚴格的資料庫交易** (如：部落格文章發表成功與否)
- (2) Web2.0**並不要求嚴格的讀寫即時性** (如：部落格文章發表是否要讓其它用戶立即看到)
- (3) Web2.0通常**不包含大量複雜的SQL查詢** (不需正規化方式設計表格，去除關聯式資料庫的嚴謹結構特性，以儲存空間**換取更好的查詢性能**)

關聯性資料庫群集

MySQL集群是否可以完全解決問題？

- **複雜性**：部署、管理、配置很複雜
- **資料庫複製**：MySQL主從之間採用複製方式，只能是**非同步複製**，當主資料庫壓力較大時可能產生較大延遲，主從切換可能會遺失最後一部分更新交易，這時往往需要人工介入，備份和恢復不方便
- **擴充問題**：如果系統壓力過大需要增加新的機器，這個過程涉及**資料重新劃分**，整個過程比較複雜，且容易出錯
- **動態資料遷移問題**：如果某個資料庫組壓力過大，需要將其中部分資料遷移出去，遷移過程需要總控節點整體協調，以及資料庫節點的配合。這個過程很難做到自動化，通常需要**人工處理**



RDBMS vs. NOSQL

比較標準	RDBMS	NoSQL	備註
資料庫原理	完全支援	部分支援	<ul style="list-style-type: none">RDBMS有關聯代數理論作為基礎NoSQL沒有統一的理論基礎
資料規模	大	超大	<ul style="list-style-type: none">RDBMS很難實現橫向擴展，縱向擴展的空間也比較有限，性能會隨著資料規模的增大而降低NoSQL可以很容易透過添加更多設備來支援更大規模的資料
資料庫模式	固定	靈活	<ul style="list-style-type: none">RDBMS需要定義資料庫模式，嚴格遵守資料定義和相關限制條件NoSQL不存在資料庫模式，可以自由靈活定義並儲存各種不同類型的資料
查詢效率	快	可以實現高效的簡單查詢，但是不具備高度結構化查詢等特性，複雜查詢的性能不盡人意	<ul style="list-style-type: none">RDBMS借助於索引機制可以實現快速查詢（包括記錄查詢和範圍查詢）很多NoSQL資料庫沒有以複雜查詢為主的索引，雖然NoSQL可以使用MapReduce來加速查詢，但是，在複雜查詢方面的性能仍然不如RDBMS

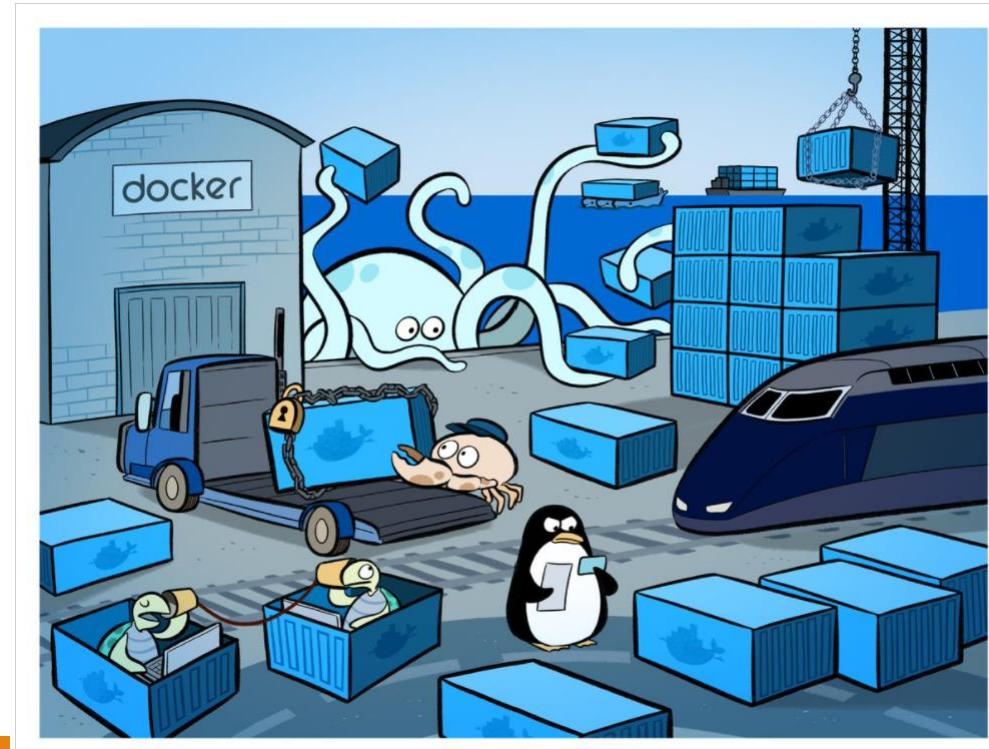
RDBMS vs. NOSQL

比較標準	RDBMS	NoSQL	備註
一致性	強一致性	弱一致性	<ul style="list-style-type: none">RDBMS嚴格遵守交易ACID模型，可以保證交易強一致性很多NoSQL資料庫放鬆了對交易ACID四特性的要求，而是遵守BASE模型，只能保證最終一致性
資料完整性	容易實現	很難實現	<ul style="list-style-type: none">任何一個RDBMS都可以很容易實現完整性限制，比如透過主鍵或者非空限制來實現個體完整性，透過主鍵、外來鍵來實現參考完整性，透過限制或者觸發器來實現用戶自訂完整性但是，在NoSQL資料庫卻無法實現
擴展性	一般	好	<ul style="list-style-type: none">RDBMS很難實現橫向擴展，縱向擴展的空間也比較有限NoSQL在設計之初就充分考慮了橫向擴展的需求，可以很容易透過添加廉價設備實現擴展
可用性	好	很好	<ul style="list-style-type: none">RDBMS在任何時候都以保證資料一致性為優先目標，其次才是最佳化系統性能。隨著資料規模的增大，RDBMS為了保證嚴格的一致性，只能提供相對較弱的可用性大多數NoSQL都能提供較高的可用性

DOCKER

Docker 字面上的意思是「碼頭工人」，碼頭上會有打包、運送...等服務，碼頭工人 (Docker) 可快速的用貨櫃 (Container) 將貨物 (Application) 裝上船。

Docker 是一種容器化技術 (輕量級的虛擬化技術)，可以把你的應用程式連同環境一起打包，部署的時候不用擔心環境的問題



Virtual machine (虛擬機器) vs. container(容器)

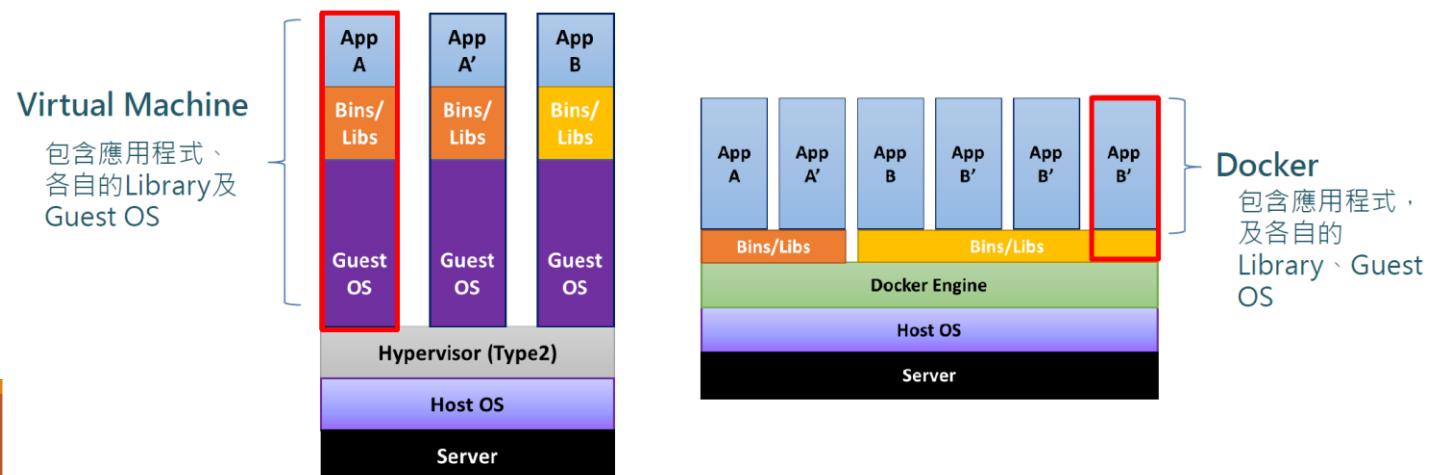
虛擬機器VM (以作業系統為中心)-在系統層上虛擬化

讓你在**實體作業系統 (Host OS)** VM上面再裝一個作業系統 (Guest OS) 獨立環境，就是一個可以然後讓兩個作業系統彼此不會打架的平台。

Hypervisor：Host OS 裡面負責管理 VM 的 Apps 稱為 Hypervisor 或 Virtual Machine Monitor (VMM，虛擬機器監視器)，Hypervisor 是在硬體層虛擬化。

容器 (以應用程式為中心)-在作業系統層上虛擬化

改善虛擬機器因為需要裝 Guest OS 導致啟動慢、佔較大記憶體的問題。將一個應用程式所需的程式碼、函式庫打包(想成一個資料夾就是一個執行環境)，不需要再另外安裝作業系統 (Guest OS) 也可以執行。



Docker願景

建置 image，然後使用 Registry 來管理 image，再使用 Docker Engine 將 container 和包含的 App 在任意平台(不管是實體機、虛擬機還是雲端)上執行



Docker優勢

更快速的交付和部署	更有效率的虛擬化	更輕鬆的遷移和擴展	更簡單的管理
<ul style="list-style-type: none">Developer快速建置開發環境DevOps可透過開發環境之容器快速部屬節省開發、測試、部屬時間	<ul style="list-style-type: none">不需額外的虛擬化支援，它是核心層級(應用程式層)的虛擬化	<ul style="list-style-type: none">平台相容性佳，包含實體機器、虛擬機、個人電腦、私有雲等	<ul style="list-style-type: none">應用程式更新、環境建置與部屬管理、叢集(Cluster)管理

特性	容器	虛擬機
開機載入速度	秒級	分鐘級
硬碟容量	一般為MB	一般為GB
效能	接近原生	比較慢
系統支援量	單機支援上千個容器	一般十幾個

Docker 三元素

映像檔 Image

映像檔是一個模板，用來重複產生容器實體。例如：一個映像檔裡可以包含一個完整的 MySQL 服務或是一個 Ubuntu 作業系統。

映像檔可以透過撰寫由命令行([linux指令](#))構成的 *Dockerfile* 輕鬆建立，或從公開的地方 ([DockerHub](#)) 下載已經做好的映像檔來使用。

容器 Container

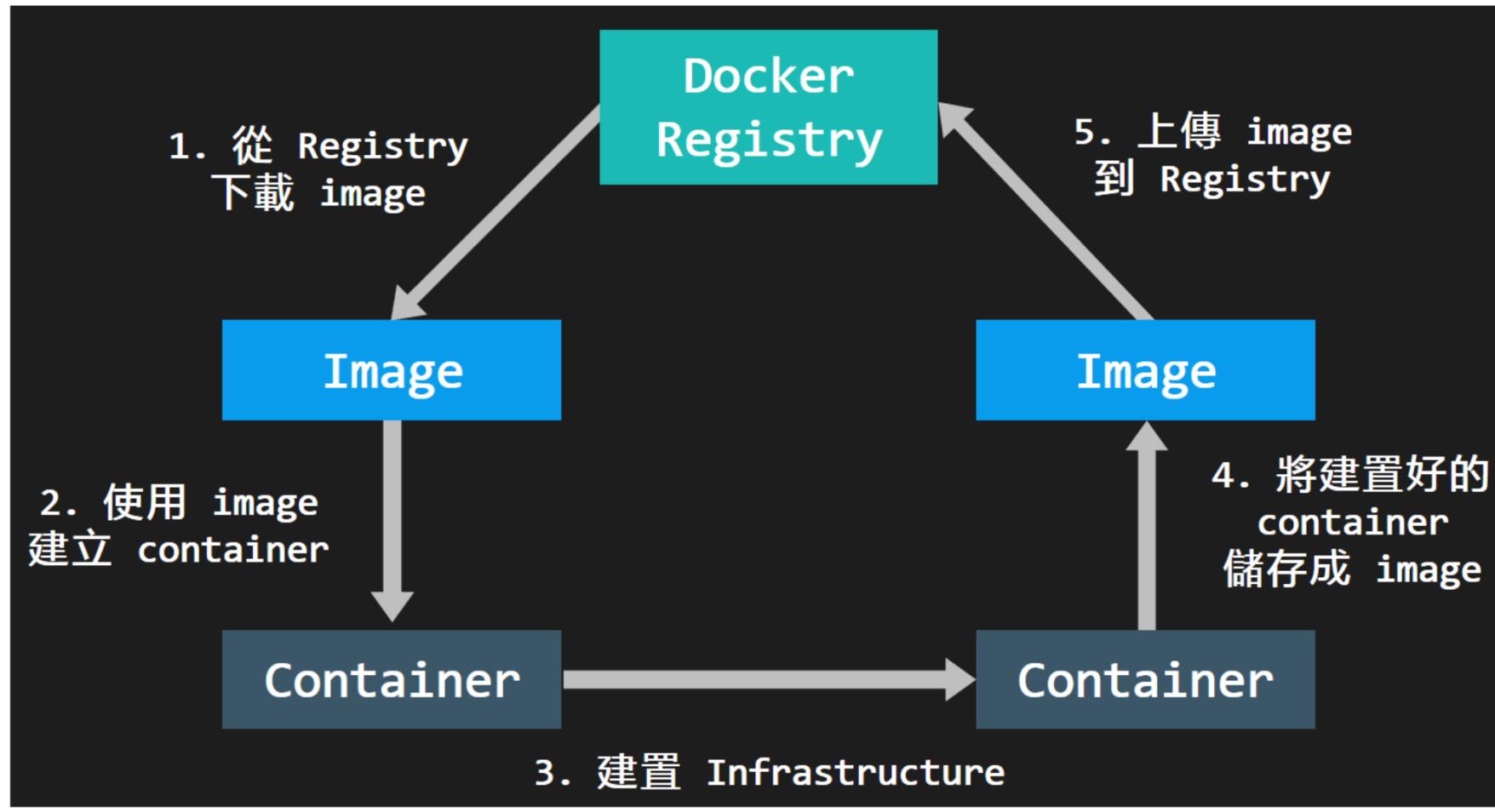
是用映像檔建立出來的執行實例，可以被啟動、開始、停止、刪除。每個容器都是相互隔離、保證安全的平台。

倉庫 Repository

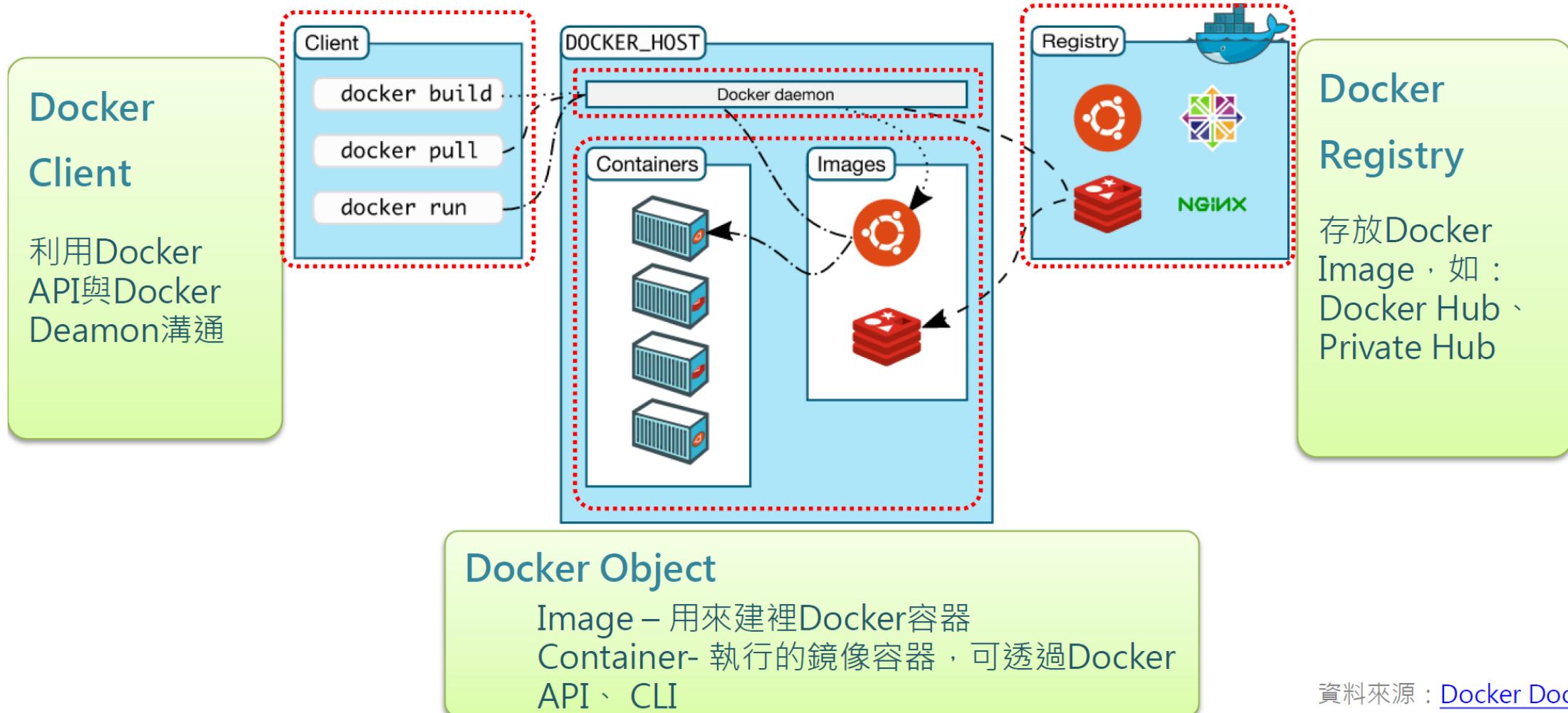
[Docker Hub](#)存放了數量龐大的映像檔供使用者下載，透過 push、pull 的方式上傳、存取。

Registry 上存放著多個 Repository，每個 Repository 中又包含了多個 image，每個 image 有多個不同的標籤 (tag)，標籤會以數字或英文的方式命名

Docker 是怎麼運作的？



Docker流程



資料來源：[Docker Doc](#)

安裝 Docker

在 Ubuntu Linux 中，使用 apt 安裝 Docker 比較方便

```
> sudo apt-get install docker.io
```

安裝好之後，查看一下 docker 服務是否有正常啟動

```
> service docker status
```

接著將自己的使用者帳號加入至 docker 群組

```
> sudo usermod -aG docker ubuntu
```

登出再重新登入之後，就可以開始使用 Docker 了。首先查看 Docker 的版本資訊：

```
> docker version
```

取得 Docker Container 映像檔

可以在 Docker Hub 有許多公開的 container 映像檔，利用 docker 的 search 指令來尋找自己需要的 container 映像檔，例如搜尋 Ubuntu Linux 的 container 映像檔：

> docker search [關鍵字]

> docker search Ubuntu //預設latest

> docker search ubuntu:15.10 //固定版本

若要下載 container 映像檔，可以使用 pull 並且指定 container 的名稱

> docker pull Ubuntu

docker 的 images 指令可以列出目前系統上所有的 container 映像檔

> docker images 或 docker image ls

> docker images [關鍵字] //只查詢某個關鍵字的映像檔

Docker Container 映像檔測試

下載 ubuntu container 映像檔，可以使用 **pull** 並且指定 container 的名稱

> docker **pull** Ubuntu

選擇 ubuntu 作為程式執行的環境，在這個環境中執行 'Hello world' 這個指令

> docker run ubuntu /bin/echo 'Hello world' 或

> docker run ubuntu echo 'Hello world'

離開 container 的互動式操作，則執行 exit 即可

> exit 或

Ctrl+D

Docker Container 映像檔背景測試

執行常駐程式使用 -d 參數，讓整個 container 放在背景執行，輸出一串很長的字串來表達常駐工作

```
> docker run -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 1; done"
```

使用 ps 指令來查看目前所有 **正在執行** 的 container

```
> docker ps
```

在結果中最後一個 NAMES 欄位是 container 執行個體的名稱。

因為常駐運行在後端，可以使用以下指令進入

```
> docker attach [常駐程式名稱] 或
```

```
> docker exec -it [常駐程式名稱] /bin/bash
```

或是透過 Docker 的 logs 紀錄檔功能來查閱常駐程式的結果

```
> docker logs [常駐程式名稱]
```

透過 Docker 的 stop 來停止常駐程式

```
> docker stop [常駐程式名稱]
```

Docker images and container 移除

查詢

docker 的 images 指令可以列出目前系統上所有的 container 映像檔

> docker images 或 docker image ls

docker 的 container 指令可以列出目前系統上所有在執行當中的容器

> docker container ls

停止執行(才能刪除)

要移除所有已經停止的 Container 可以使用以下指令

> docker container stop [容器ID或是名字] (可在查詢階段看到)

> docker container prune

刪除映像檔與容器

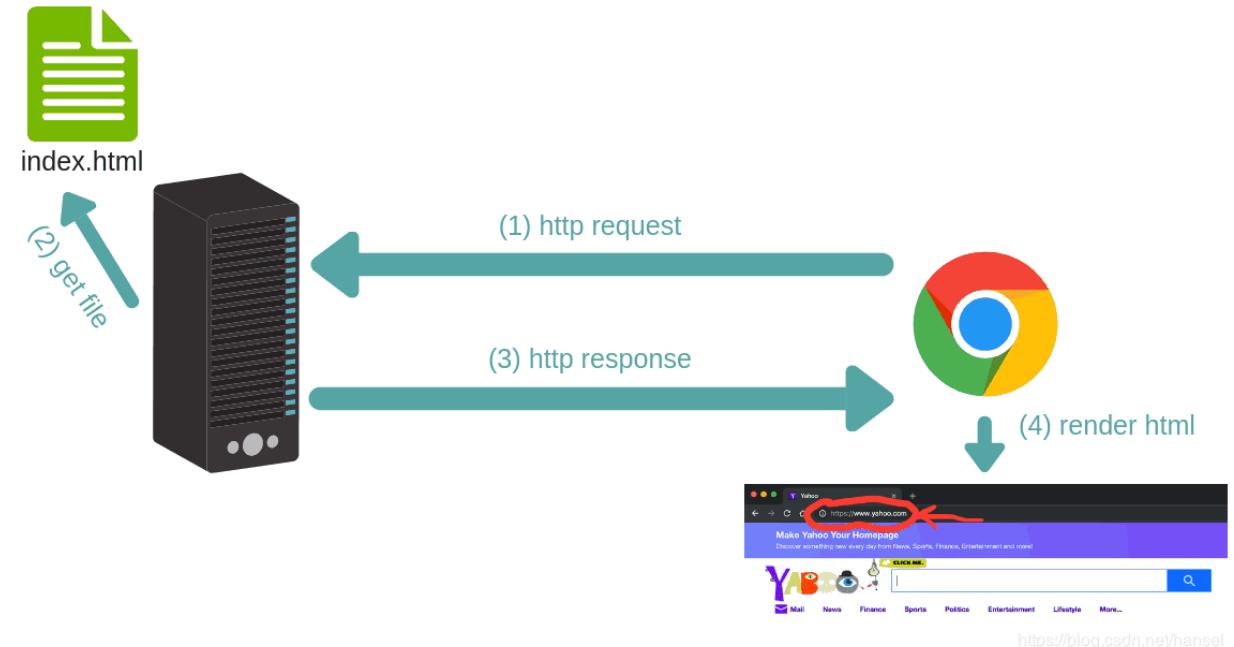
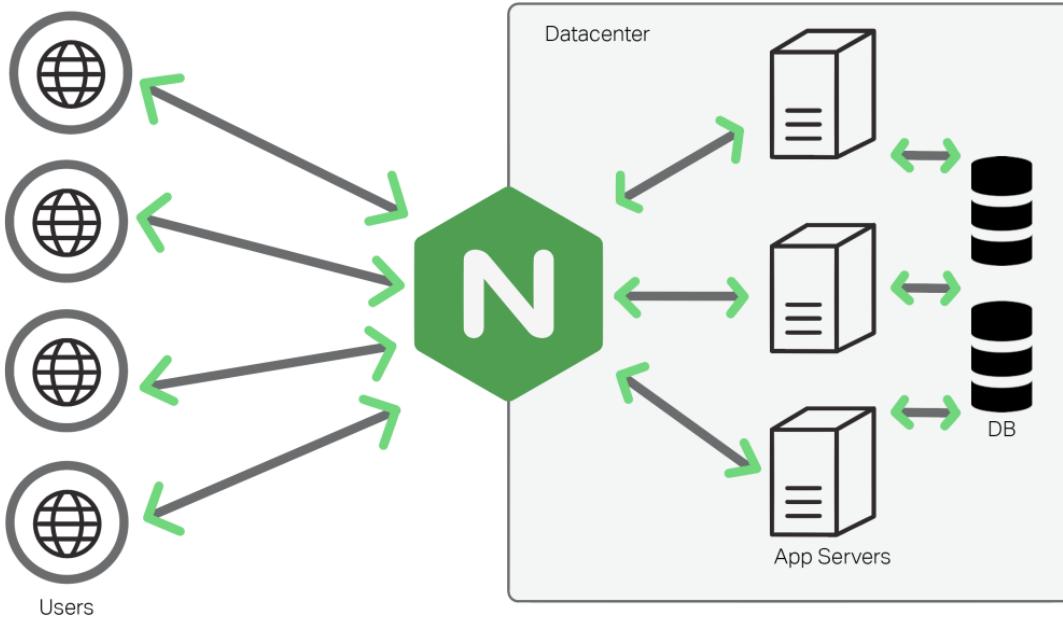
可以刪除目前系統上的容器

> docker container rm [容器ID] (容器ID可在查詢階段看到)

可以刪除目前系統的某個映像檔

> docker image rm[映像檔名字] (映像檔名字可在查詢階段看到"REPOSITORY")

Web server



Web Server：如 **Nginx**、**Apache**，只能拿來處理靜態資源，負載平衡、代理。動態資源會把需求轉發到程式語言起的 Application Server，由 Application Server 處理完後，再丟 response 回去，由 Web Server 進行回應，最後才回到 Client 端。

Web Server 主要功能則是負載平衡（在高流量的狀況下，只靠一個 Application Server 是肯定會炸掉的，所以需要開起多個 Application Server 來分擔流量，負載平衡就是負責分發，決定 request 要被分到哪一個 Application Server 處理）

Dockerhub start (web server)

我們在google 或是把dockerhub找到我們要的docker images

如何建置web Server (nginx) 並觀看頁面，將此 images 下載pull下來

```
> docker run -it -d -p 8080:80 --name web[任意名字] nginx
```

-i , --interactive : 讓 Container 的標準輸入保持打開

-t , --tty : 讓 Docker 分配一個虛擬終端 (pseudo-tty) 並綁定到 Container 的標準輸入上

-d , --detach : 讓 Container 處於背景執行狀態並印出 Container ID

--name : 指定 Container 名稱

-p , --publish : 將 Container 發布到指定的port號

利用瀏覽器看一下網址: [AWS EC2的IP]:8080

放自己的網頁

```
> docker run -it -d -p 8080:80 --name web[任意名字] -v 自己網頁資料夾:/usr/share/nginx/html -d nginx
```

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Docker nginx</title>
</head>
<body> Hello world! </body>
</html>
```

Dockerhub start (mysql+phpmyadmin)

用 phpmyadmin 資料庫管理工具串接 Mysql 服務在瀏覽器上管理資料庫，將 images 下載pull下來

> docker run -itd --name mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=[密碼] mysql:5.7.24

> docker run --name phpmyadmin -d --link mysql -e PMA_HOST="mysql" -p 8080:80
phpmyadmin/phpmyadmin

-i , --interactive : 讓 Container 的標準輸入保持打開

-t , --tty : 讓 Docker 分配一個虛擬終端 (pseudo-tty) 並綁定到 Container 的標準輸入上

-d , --detach : 讓 Container 處於背景執行狀態並印出 Container ID

--name : 指定 Container 名稱

-p , --publish : 將 Container 發布到指定的port號

> docker start mysql

> docker start phpmyadmin

建立 Docker Image

可以建立屬於自己的映像檔，需要*Dockerfile*來建立

Dockerfile 裡是由一行行指令所組成，並且支援以#為開頭的註解行，腳本裡大致會有這四項配置：

基底映像檔 (FROM): 整個 Dockerfile 的地基，在指令中為 FROM，例如以 ubuntu 為基底。空白鏡像為基底則寫”FROM scratch”

維護者(MAINTAINER)

操作指令

RUN:

Dockerfile 裡為重要的部份，安裝、複製、引入..等指令全部都在這個地方運行。

EXPOSE:

聲明對外開放的端口

容器執行時指令(CMD):

當容器執行後所要執行的動作，一個 Dockerfile 裡只能有一個 CMD，當有多個 CMD 時，只會執行最後一個 CMD 指令。

FROM ubuntu:16.04

MAINTAINER richard

RUN apt-get update -y

RUN apt-get install nginx -y

RUN apt-get install mysql -y

或

RUN apt-get update -y \

&& apt-get install nginx -y \

&& apt-get install mysql -y \

EXPOSE 80

CMD echo 'Hello world!'

CMD ["nginx", "-g", "daemon off;"]

建立 Docker Image

建立存放 Dockerfile 的資料夾

```
> mkdir Dockerfile
```

在Dockerfile資料夾內撰寫 Dockerfile 檔案

在Dockerfile資料夾內Build Dockerfile 檔案

```
> docker build -t nginx-build .
```

-t 是為了給這個 Image 一個 Tag

執行Dockerfile 檔案

```
> docker run -itd --name nginx -p 8080:80 nginx-build
```

```
FROM ubuntu:16.04
```

```
MAINTAINER richard
```

```
RUN apt-get update -y
```

```
RUN apt-get install nginx -y
```

```
EXPOSE 80
```

```
CMD ["nginx", "-g", "daemon off;"]
```

建立 Docker for nginx

建立一個存放網頁的資料夾

```
> mkdir my_web
```

建立一個首頁的index.html內容如右

在資料夾內撰寫 Dockerfile 檔案

```
> vi Dockerfile
```

在Dockerfile資料夾內Build Dockerfile 檔案

```
> docker build -t image名字:image版本 ..
```

```
> docker build -t my_web . (不寫版本預設latest)
```

-t 是為了給這個 Image 一個 Tag

輸入docker images 查看是否存在新的my_web

執行Dockerfile 檔案

```
> docker run -itd --name container名字 -p 8000:80 -d image名字
```

```
> docker run -itd --name web -p 8000:80 -d my_web
```

FROM nginx

COPY index.html /usr/share/nginx/html

Dockerfile

!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Docker nginx</title>

</head>

<body>

Hello world!

</body>

</html>

index.html

來衝一波pig吧!

首先在網路上找到一個比較輕量級的pig docker

```
> docker run --name pig -it pkeropen3/pig_hadoop_debian:0.15.0 bash
```

恭喜!你已經灌好JAVA, HADOOP, PIG啦!

實作PIG

1. 網路上先下載我們需要的檔案並存成sample_1.txt

```
> curl https://bigdataprogrammers.com/wp-content/plugins/download-  
attachments/includes/download.php?id=1279 > sample_1.txt
```

2. 進入pig

```
> pig -x local
```

出現grunt>

成功了~

來衝一波pig吧!

首先在網路上找到一個比較輕量級的pig docker

```
> docker run --name pig -it pkeropen3/pig_hadoop_debian:0.15.0 bash
```

恭喜!你已經灌好JAVA, HADOOP, PIG啦!

實作PIG

1. 網路上先下載我們需要的檔案並存成sample_1.txt

```
> curl https://bigdataprogrammers.com/wp-content/plugins/download-  
attachments/includes/download.php?id=1279 > sample_1.txt
```

2. 進入pig

```
> pig -x local
```

出現grunt>

成功了~

```
root@35971f9d0ff4:/# cat sample_1.txt  
18282782|NW  
1929SEGH2|BSTN  
172u8562|PLA  
121232|JHK  
3443453|AG  
198WS238|AGSroot@35971f9d0ff4:/#
```

```
root@35971f9d0ff4:/# pig -x local  
21/05/15 07:57:32 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL  
21/05/15 07:57:32 INFO pig.ExecTypeProvider: Picked LOCAL as the ExecType  
2021-05-15 07:57:32,786 [main] INFO org.apache.pig.Main - Apache Pig version 0.15.0 (r1682971) compiled Jun 01 2015, 11:  
:44:35  
2021-05-15 07:57:32,787 [main] INFO org.apache.pig.Main - Logging error messages to: //pig_1621065452785.log  
2021-05-15 07:57:32,820 [main] INFO org.apache.pig.impl.util.Utils - Default bootup file /root/.pigbootup not found  
2021-05-15 07:57:33,445 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. I  
nstead, use fs.defaultFS  
2021-05-15 07:57:33,448 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated  
. Instead, use mapreduce.jobtracker.address  
2021-05-15 07:57:33,457 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hado  
op file system at: file:///br/>2021-05-15 07:57:33,602 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is depreca  
ted. Instead, use dfs.bytes-per-checksum  
grunt>
```

Pig example: load & store

1. 首先將檔案讀成pig表格

```
grunt> MyData = LOAD './sample_1.txt' using PigStorage('|') AS (id:chararray, code:chararray);
```

2. 處理完了，把資料全部DUMP出來

```
grunt> DUMP MyData; 或是 grunt> illustrate MyData;
```

3. 儲存成逗點分格的檔案

```
grunt> STORE MyData INTO './psv_to_comma' using PigStorage(',');
```

4. 呈現結果

跳開PIG利用Ctrl+C

```
> cat ./psv_to_comma/*
```

```
root@35971f9d0ff4:/# cat sample_1.txt
18282782|NW
1929SEGH2|BSTM
172u8562|PLA
121232|JHK
3443453|AG
198WS238|AGS
root@35971f9d0ff4:/#
```

Pig example: filter, select

1. 網路上先下載我們需要的檔案並存成sample_2.txt

```
> curl https://bigdataprogrammers.com/wp-content/plugins/download-  
attachments/includes/download.php?id=597 > sample_2.txt
```

2. 進入pig

```
> pig -x local
```

3. 將檔案讀成pig表格

```
grunt> MyData = LOAD './sample_2.txt' using PigStorage('|') AS (id:chararray, code:chararray);
```

4. 處理完了，把資料全部DUMP出來

```
grunt> DUMP MyData;
```

```
2017-04-27 17:37:51 [main] INFO org.apache.pig.  
          (1A58252, POSTP)  
          (73VS543, )  
          (HD52836, PREP)  
          (8SH356H, POSTP)  
          (L62HJS5, PEND)  
          (Q672821, PREP)  
          (M672GS6, CLS)  
          (DGSW522, POSTP)  
          (RAR1729, PEND)  
          (YE52863, )  
          (GHS5281, )  
          (SYY8373, CLS)  
grunt> █
```

Pig example: filter, select

5. 篩選資料

```
grunt> SubData = FILTER INPUT_RECORDS BY (contype=='POSTP' OR contype=='PREP' OR contype IS NULL OR contype== "");
```

6. 處理完了，把資料全部DUMP出來

```
grunt> DUMP SubData;
```

7. 缺值補NA

```
grunt> OUTPUT_RECORDS = FOREACH SubData GENERATE id,((contype IS NULL OR contype==")?'NA':contype) AS contype;
```

8. 處理完了，把資料全部DUMP出來

```
grunt> DUMP OUTPUT_RECORDS;
```

```
2017-04-27 17:38:56,429 [main] INFO org.apache.pig.backe
(1A58252, POSTP)
(73VS543, )
(HD52836, PREP)
(8SH356H, POSTP)
(Q672821, PREP)
(DGSW522, POSTP)
(YE52863, )
(GHS5281, )
grunt> █
```

```
2017-04-27 17:37:23,974 [main] INFO org.apache.pig.backe
(1A58252, POSTP)
(73VS543, NA)
(HD52836, PREP)
(8SH356H, POSTP)
(Q672821, PREP)
(DGSW522, POSTP)
(YE52863, NA)
(GHS5281, NA)
grunt> █
```

Pig example: inner join

1. 網路上先下載我們需要的檔案並存成sample_2.txt

```
> curl https://bigdataprogrammers.com/wp-content/plugins/download-attachments/includes/download.php?id=399 > A.txt
```

```
> curl https://bigdataprogrammers.com/wp-content/plugins/download-attachments/includes/download.php?id=400 > B.txt
```

2. 進入pig

```
> pig -x local
```

3. 將檔案讀成pig表格

```
grunt> A = LOAD './A.txt' using PigStorage(',') AS (id :int,type:chararray);
```

```
grunt> B = LOAD './B.txt' using PigStorage(',') AS (id :int,type:chararray);
```

4. Inner join

```
grunt> INNER_JOIN = JOIN A BY id, B BY id;
```

5. 處理完了，把資料全部DUMP出來

```
grunt> DUMP INNER_JOIN ;
```

A		B	
Id	Type	Id	Type
1	accounts	1	accounts
2	science	3	science
3	statistics	4	statistics
4	computer	8	agriculture
5	computer	9	finance

Inner Join			
A		B	
Id	Type	Id	Type
1	accounts	1	accounts
3	statistics	3	science
4	computer	4	statistics

Pig example: inner join

4. Left join

```
grunt> LEFT_JOIN = JOIN A BY id LEFT, B BY id;
```

5. 處理完了，把資料全部DUMP出來

```
grunt> DUMP LEFT_JOIN;
```

6. Right join

```
grunt> RIGHT_JOIN = JOIN A BY id RIGHT ,B BY id;
```

7. 處理完了，把資料全部DUMP出來

```
grunt> DUMP RIGHT_JOIN;
```

8. full join

```
grunt> FULL_JOIN = JOIN A BY id FULL, B BY id;
```

9. 處理完了，把資料全部DUMP出來

```
grunt> DUMP FULL_JOIN;
```

A		B	
Id	Type	Id	Type
1	accounts	1	accounts
2	science	3	science
3	statistics	4	statistics
4	computer	8	agriculture
5	computer	9	finance

Left Join			
A		B	
Id	Type	Id	Type
1	accounts	1	accounts
2	science	NULL	NULL
3	statistics	3	science
4	computer	4	statistics
5	computer	NULL	NULL

Right Join			
A		B	
Id	Type	Id	Type
1	accounts	1	accounts
3	statistics	3	science
4	computer	4	statistics
NULL	NULL	8	agriculture
NULL	NULL	9	finance

Full Join			
A		B	
Id	Type	Id	Type
1	accounts	1	accounts
2	science	NULL	NULL
3	statistics	3	science
4	computer	4	statistics
5	computer	NULL	NULL
NULL	NULL	8	agriculture
NULL	NULL	9	finance

Pig example: match, order by

1. 網路上先下載我們需要的檔案並存成檔案

```
> curl https://raw.githubusercontent.com/jiankaidang/Pig-Latin-and-MapReduce/master/tweets.csv > tweets.csv
```

2. 進入pig

```
> pig -x local
```

3. 將檔案讀成pig表格

```
grunt> tweets = LOAD './tweets.csv' using PigStorage(',') AS (id:long, content:chararray, user:chararray);
```

4. 篩選相似內容

```
grunt> tweets_include_favorite = FILTER raw BY (content matches '.*favorite.*');
```

5. 排序資料

```
grunt> tweets_include_favorite_ordered_by_id = ORDER tweets_include_favorite BY id;
```

6. 處理完了，把資料全部DUMP出來

```
grunt> DUMP tweets_include_favorite_ordered_by_id;
```

Pig example: group by, count

1. 網路上先下載我們需要的檔案並存成檔

```
> curl https://raw.githubusercontent.com/jiankaidang/Pig-Latin-and-MapReduce/master/tweets.csv > tweets.csv
```

```
> curl https://raw.githubusercontent.com/jiankaidang/Pig-Latin-and-MapReduce/master/users.csv > users.csv
```

2. 進入pig

```
> pig -x local
```

3. 將檔案讀成pig表格

```
grunt> tweets = LOAD './tweets.csv' using PigStorage(',') AS (id:long, content:chararray, user:chararray);
```

```
grunt> users = LOAD './users.csv' using PigStorage(',') AS (login:chararray, name:chararray, state:chararray);
```

Pig example: group by, count

4. 以users為主合併tweets

```
grunt> users_join_tweets = JOIN users BY login LEFT, tweets BY user;
```

5. 依照name來群組

```
grunt> name_group = GROUP users_join_tweets BY name;
```

6. 計算每個name所發表的文章數

```
grunt> number_of_tweets = FOREACH name_group GENERATE group, COUNT(users_join_tweets.id);
```

7. 處理完了，把資料全部DUMP出來

```
grunt> DUMP number_of_tweets;
```

7. 存起來

```
grunt> STORE number_of_tweets INTO '3a.result' USING PigStorage (',');
```

執行方式

我們經常會使用 "sh" 這個指令來執行一些副檔名為 .sh 的檔案，這些 .sh 的檔案都是所謂的 **Shell script**，也就是在 Linux 系統當中常見的腳本檔案

```
> pig -x local test_1.pig
```

```
/**  
 * Write a Pig Latin query that outputs the login of all users in NY state. (local mode)  
 *  
 * As input it will read from a CSV file that contains descriptions about users.  
 * Each line in the user collection contains: login, name and state from a specific user.  
 */  
  
-- Use the PigStorage function to load the user collection file into the raw bag as an array of records.  
-- Input: (login, name, state)  
raw = LOAD 'users.csv' USING PigStorage(',') AS (login:chararray, name:chararray, state:chararray);  
  
-- Use the FILTER command to remove all records with a state which is not NY.  
users_in_NY = FILTER raw BY state eq 'NY';  
  
-- Use the FOREACH-GENERATE command to project login field from relation users_in_NY to relation login_of_users_in_NY.  
login_of_users_in_NY = FOREACH users_in_NY GENERATE login;  
  
-- Use the default Pig output function to store the results.  
STORE login_of_users_in_NY INTO '1a.result';
```

test_1.pig

Docker compose

快速建立多個 container 的工具

利用 docker-compose.yml YAML 檔來管理多個 Docker container

使用 docker-compose 指令來啟動、停止和重啟應用，以及應用中的服務和所有依賴服務的 container

如何安裝 docker compose 在 ubunut 上

```
> sudo curl -L "https://github.com/docker/compose/releases/download/1.28.4/docker-compose-  
$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

```
> sudo chmod +x /usr/local/bin/docker-compose
```

Docker compose- LAMP

docker-compose.yml

build : 指定要使用的 Dockerfile 所在目錄，Docker Compose 會自動幫你執行 docker build 指令的動作

image : 直接使用某 Docker image 來建立該 container

ports: [hostPort]:[ContainerPort] : 設定 port mapping , 指定一個 port , host 對外開 [hostPort] port , container 對內開 [ContainerPort] port

depends_on : 依據依賴順序啟動服務

volumes: [hostPath]:[containerPath] : 掛載主機上的指定 [hostPath] 目錄到 container 的指定目錄 [containerPath] 上

environment : 設定環境變數

```
version: '3.3'

services:
  phpapache:
    #image: titangene/php-apache-mysql:v1.0
    build: ./php
    ports:
      - "80:80"
      - "443:443"
    depends_on:
      - mysql
    volumes:
      - ./www:/var/www/html
  mysql:
    build: ./mysql
    ports:
      - "3306:3306"
    volumes:
      - ./mysql/data:/var/lib/mysql
  environment:
    MYSQL_ROOT_PASSWORD: admin
    #MYSQL_DATABASE: testdb
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    ports:
      - "8080:80"
    depends_on:
      - mysql
    environment:
      PMA_HOST: mysql
      PMA_PORT: 3306
```

Docker compose- LAMP

利用 Docker Compose 啟動服務

```
> docker-compose up -d
```

利用瀏覽器看[AWS EC2的IP]/db.php

停止執行、刪除服務

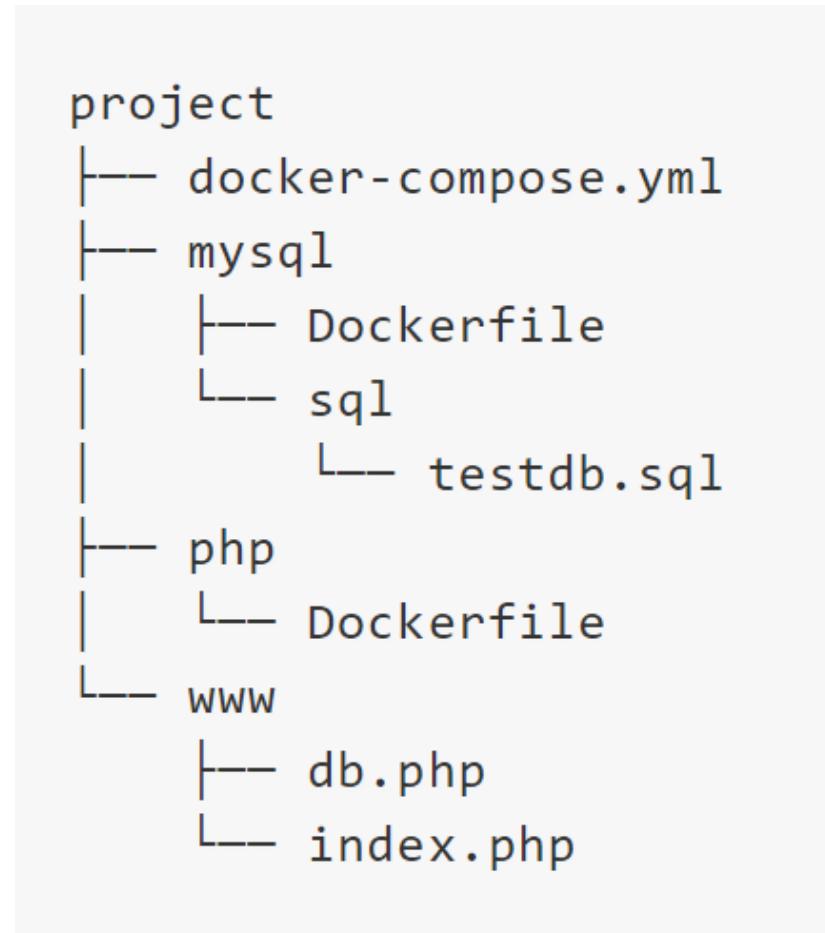
```
> docker-compose stop
```

```
> docker-compose rm
```

```
> docker-compose rm -sf
```

-s : 會先讓 container 停止執行，然後再刪除 container

-f : 會自動確認刪除 container (原本會問 Y or N)



PIG Homework (deadline: 6/11)

1. 將上課所教users.txt篩選出state的欄位是等於'NY'的資料，並且選取login欄位輸出。
2. 將上課所教tweets.txt篩選出content的欄位包含'love'的資料輸出。
3. 將上課所教users.txt檔案的state欄位選出不重複state並輸出。
4. 將上課所教的tweets.txt以及tweets.txt檔案，以users為主來合併tweets，利用name欄位來群組，了解每個群組的發文數並將發文數的欄位寫成number，最後依照新的number欄位進行遞減排序。
5. 將上課所教的users.txt 以及tweets.txt檔案進行inner join，利用login, name兩個欄位來群組，了解每個群組的發文數並將發文數的欄位寫成number，篩選出發文數大於1的資料輸出。