# Distributed Systems

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

Distributed Systems

# Container

Chun-Feng Liao

廖峻鋒

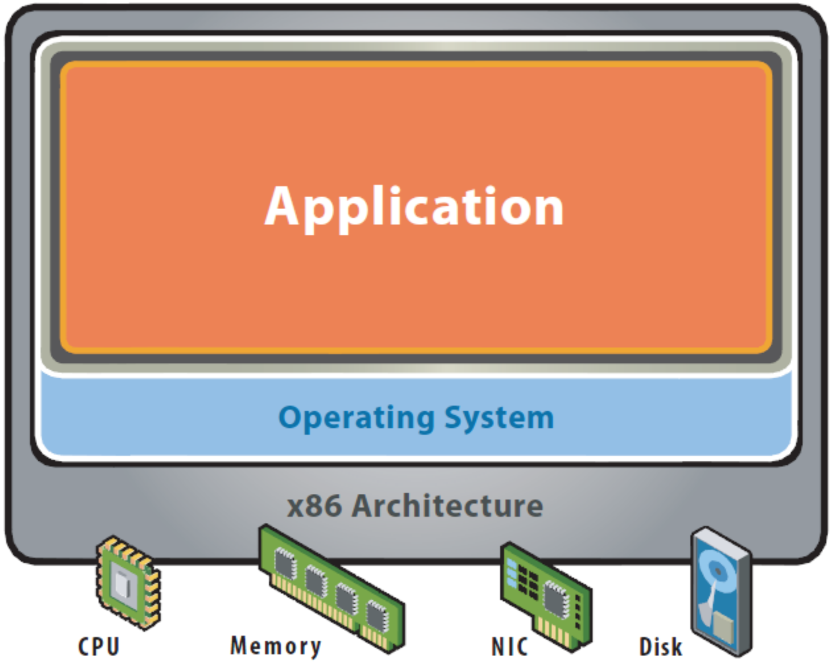
Dept. of Computer Science

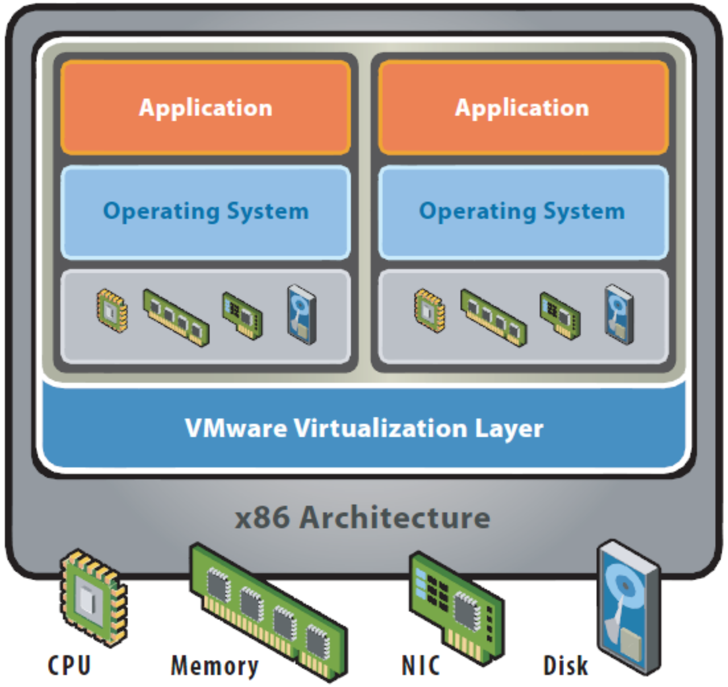
National Chengchi University

# Virtualization Technology

- Virtualization
  - To create a software-based version of something
    - Something = OS, Database, Server, Storage, Network…
- Virtual Machine
  - A software-based implementation of some real (hardware-based) computer
- Virtual Machine Monitor (VMM, or called Hypervisor)
  - The software that creates and manages the execution of virtual machines
  - Essentially an operating system

# Virtual Machines





## Before Virtualization:

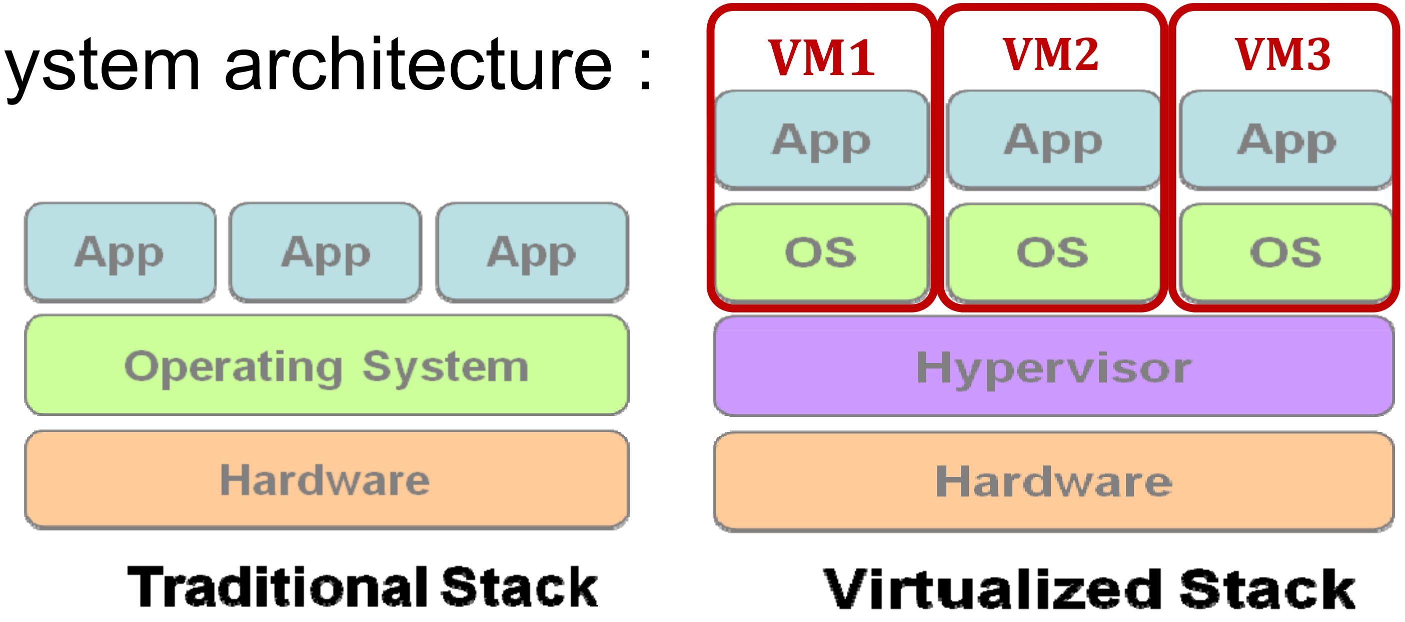
- Single OS image per machine
- Software and hardware tightly coupled
- Running multiple applications on same machine often creates conflict
- Underutilized resources
- Inflexible and costly infrastructure

## After Virtualization:

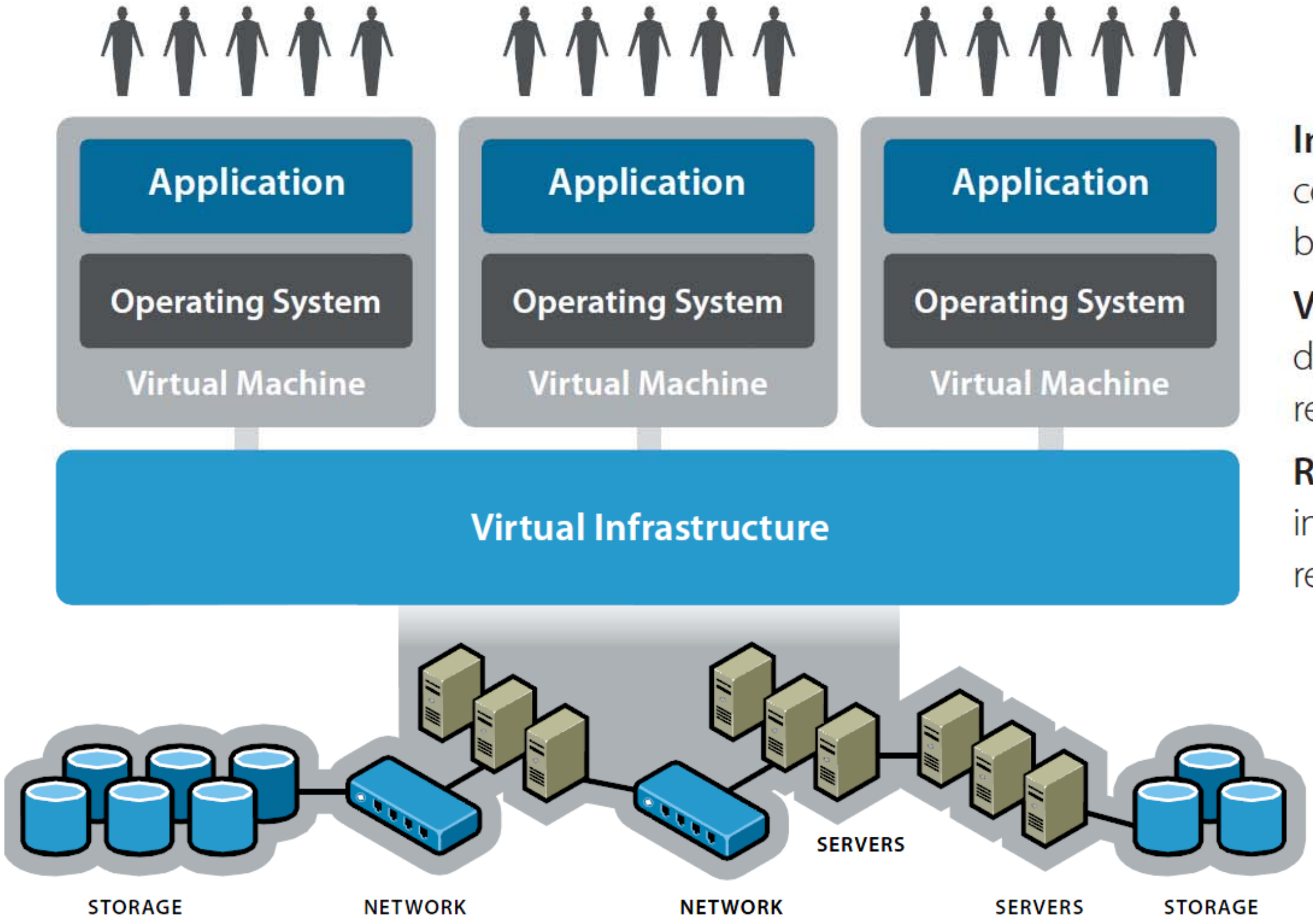
- Hardware-independence of operating system and applications
- Virtual machines can be provisioned to any system
- Can manage OS and application as a single unit by encapsulating them into virtual machines

# Virtual Machine Monitor

- What's Virtual Machine Monitor (VMM) ?
  - **VMM** or **Hypervisor** is the software layer providing the virtualization.
- System architecture :



# 任何資源都可以虛擬化

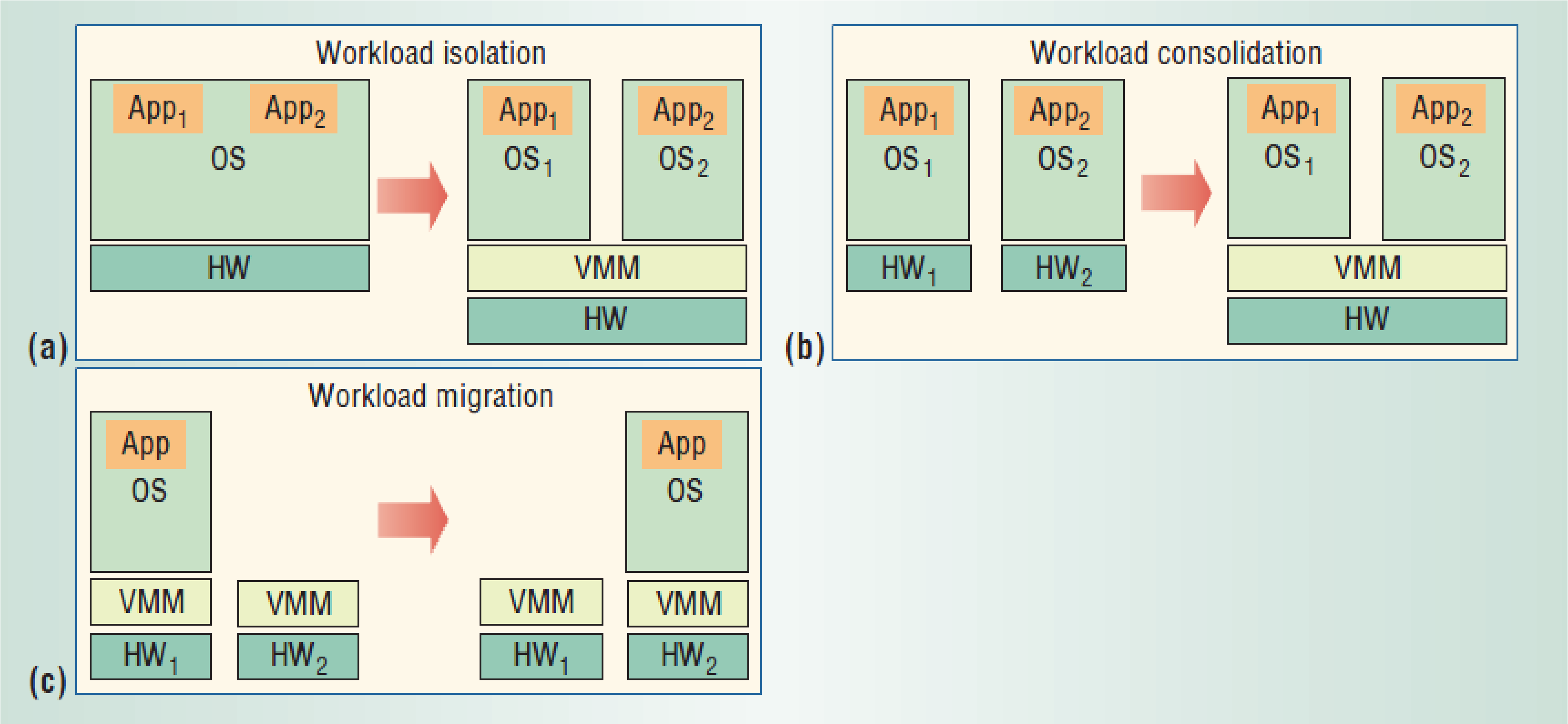


**Infrastructure** is what connects resources to your business.

**Virtual Infrastructure** is a dynamic mapping of your resources to your business.

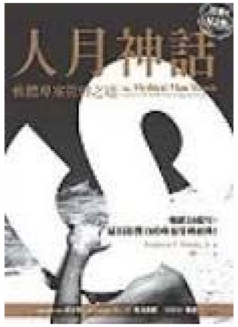
**Result**: decreased costs and increased efficiencies and responsiveness

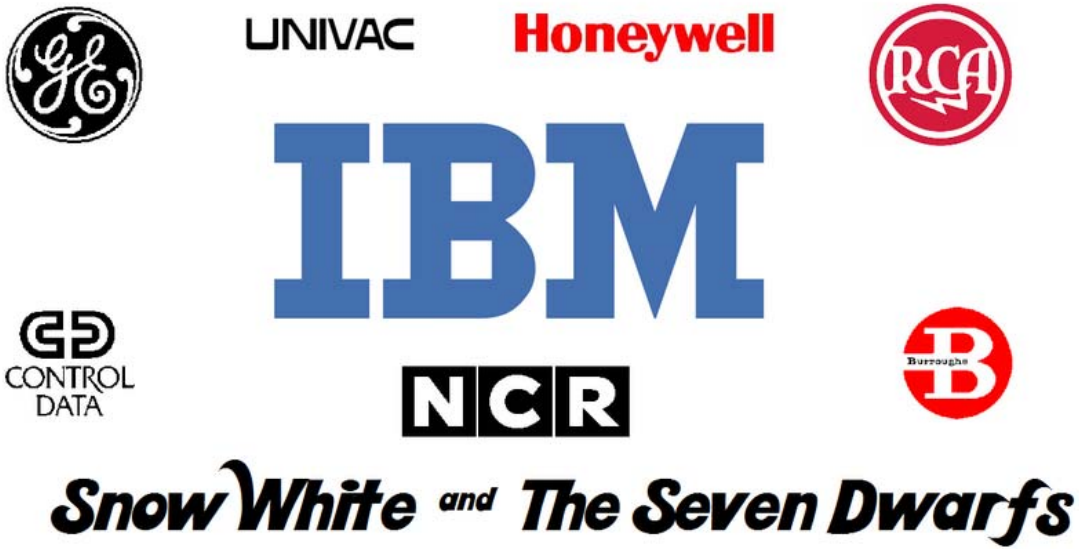
# 為何要虛擬化?


(a)
Workload isolation
$\text{App}_1$
$\text{App}_2$
OS
HW
$\text{App}_1$
$\text{OS}_1$
$\text{App}_2$
$\text{OS}_2$
VMM
HW
(b)
Workload consolidation
$\text{App}_1$
$\text{OS}_1$
$\text{HW}_1$
$\text{App}_2$
$\text{OS}_2$
$\text{HW}_2$
$\text{App}_1$
$\text{OS}_1$
$\text{App}_2$
$\text{OS}_2$
VMM
HW
(c)
Workload migration
App
OS
VMM
$\text{HW}_1$
VMM
$\text{HW}_2$
VMM
$\text{HW}_1$
App
OS
VMM
$\text{HW}_2$

# 虛擬化的起源: IBM System/360

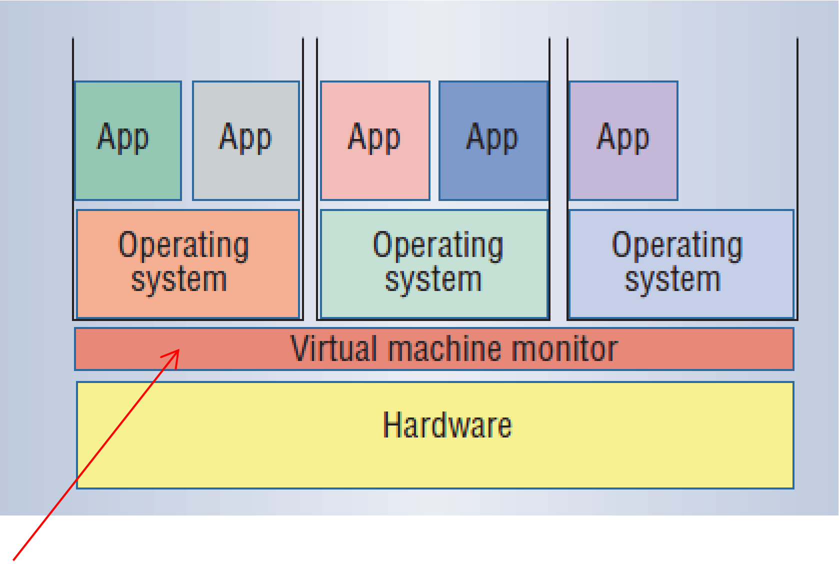
- IBM公司史上最大的豪賭
  - (當時) 人類史上最複雜的軟體系統
  - 開發過程徵召60,000員工、建立五座廠區
  - 1964/4/7公開後，IBM從此在大型主機奠定獨大地位





# 虛擬化的起源: IBM System/360

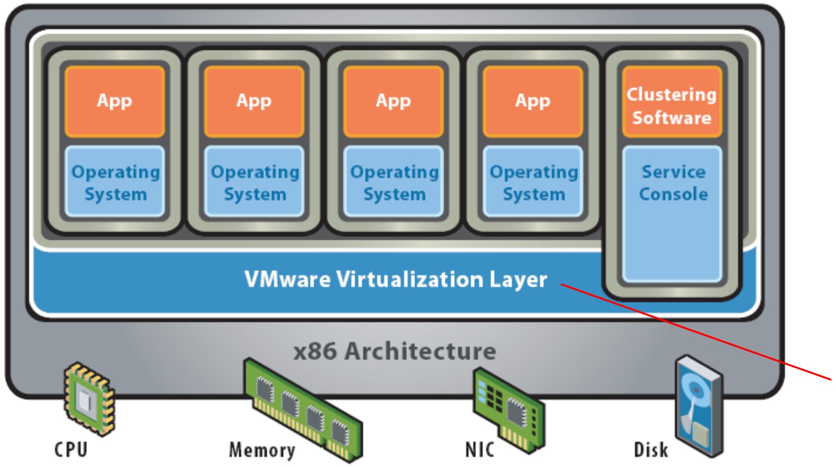
- 配合CP/CMS，成為史上第一個可虛擬化(Virtualization)的電腦
- CP (Control Program)
  - 相當於VMM
- CMS (Cambridge Monitor System)
  - 可在System/360上「同時」跑多種相容的作業系統
  - 也允許使用者自行創造作業系統



概念：提供一個(虛擬的)共同硬體介面

# System VM Virtualization Types

- Type 1 – Bare metal
  - 在硬體之上先建一個虛擬層(類似小的作業系統)，在虛擬層上再建作業系統，虛擬層完全控制硬體和資源分配，並直接分配給虛擬層上的作業系統
  - 例如:VMware ESX/ESXi Server

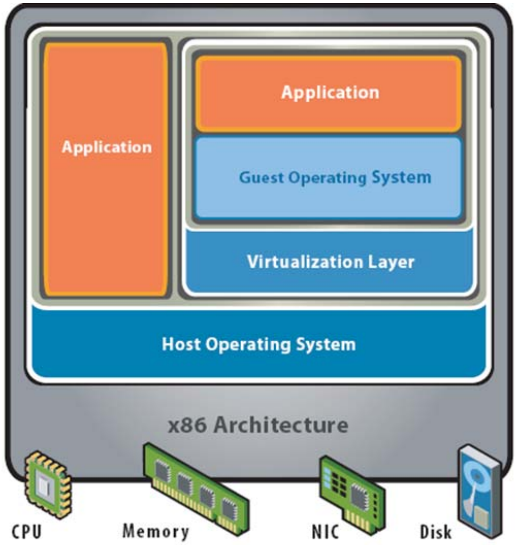


效能較高，但
Virtualization Layer要
implements所有driver!

Bare-Metal (Hypervisor) Architecture

# System VM Virtualization Types

- Type 2 – Hosted
  - 硬體已安裝了主作業系統，虛擬層被當做「應用程式」被安裝在主作業系統上，主作業系統直接存取硬體、控制和分配資源。
  - 虛擬層必需取得主作業系統所給予硬體資源，才能再分配給虛擬層上的寄居作業系統
  - Ex: VirtualBox



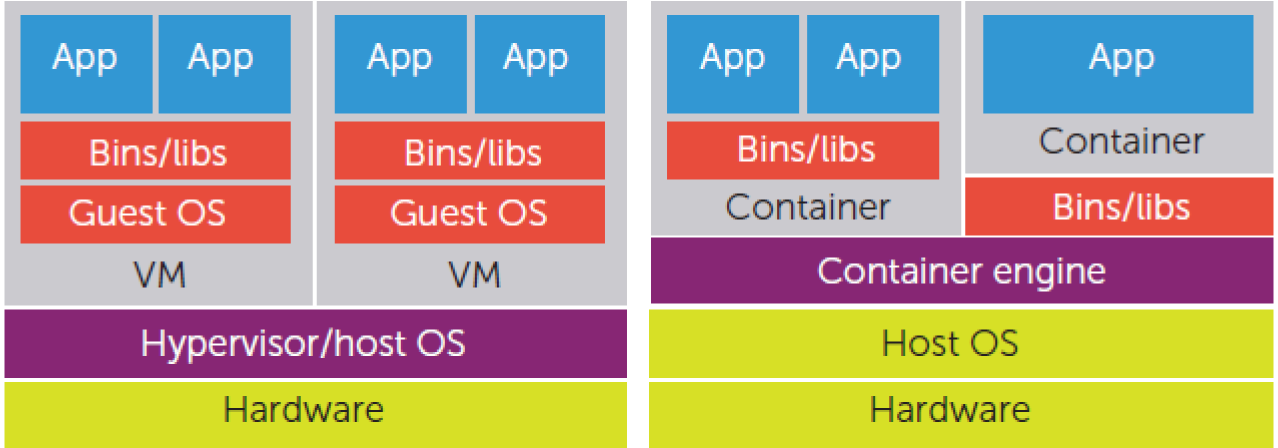
Hosted Architecture

docker 變成 container 的代名詞
但其為公司名稱

# Container and VM

VM 讓人以為獨占電腦，但其實是共享電腦
Container 讓人以為獨占 OS，但其實是共享 OS

- Full guest OS images are required for each VM
- Container
  - Holds packaged, self-contained, ready-to-deploy parts of applications
  - Containers share the same Host OS



App App
Bins/libs
Guest OS
VM

App App
Bins/libs
Guest OS
VM

Hypervisor/host OS
Hardware

App App
Bins/libs
Container

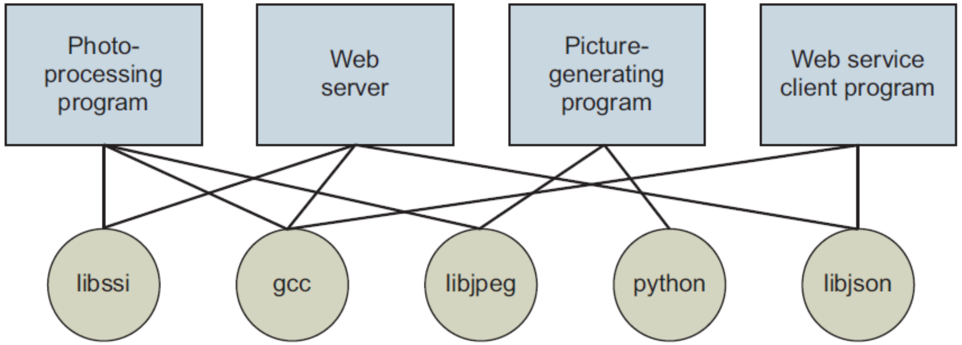
App
Container
Bins/libs

Container engine
Host OS
Hardware

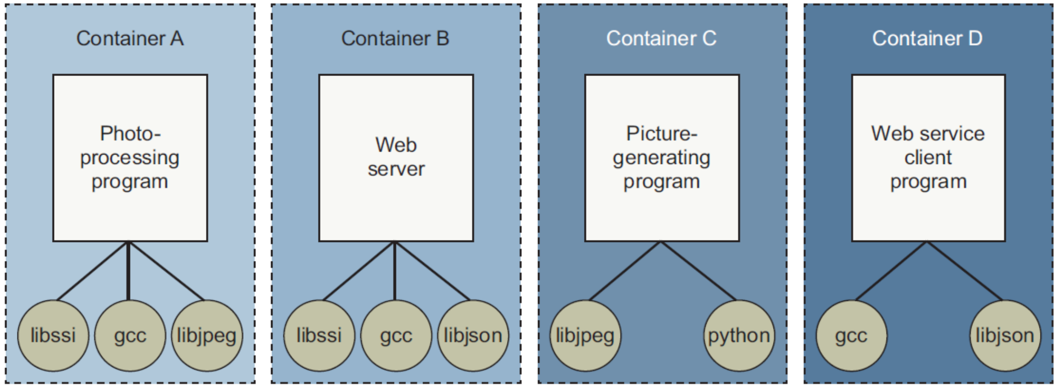
# Why Container?

- 自我包含的軟體部署
  - 易於隨時安裝、移除，不會互相影響

Without container



With container



# Portability Consideration

- Container instances should be OS-dependent
  - Container本質上是直接在OS跑
- The core reason that containers are portable
  - Containers were assumed to run on Linux
  - Linux has great binary portability among variants

# Container: 歷史觀點

- Using containers has been a best practice for a long time
  - UNIX chroot: 1979 Unix ver. 7
  - Jail: 1998 FreeBSD
  - Zones:2004 Solaris 10
  - Container:2010 Solaris 11

Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. IEEE Cloud Computing, 1(3), 81-84.

# Why Docker?

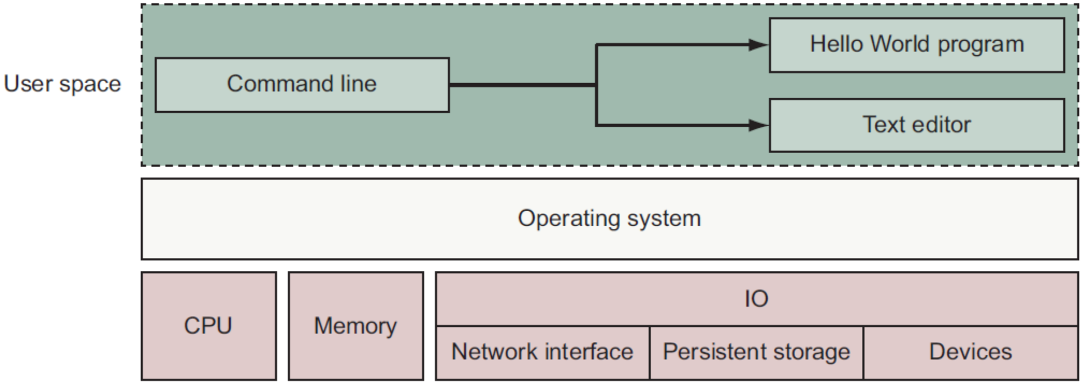
- Problem
  - Manually building containers can be challenging and easy to do incorrectly
- Solution
  - Provide a systematic way to automate fast package and deployment of Linux containers (LXC)
  - Docker uses existing container engines to provide consistent containers built according to best practices
- Benefits
  - Using LXC is easier and at lower cost
  - Provide a consistent way of using LXC

# What Docker Does?

- Docker在LXC之外做了些什麼
  - Provides kernel and application-level API
    - Takes care of Isolation
    - PID, File system, process tree, user space, CPU, network …
  - Image: the shipping container instance
    - Composed of a layered file system
    - Each action taken forms a new layer
    - Dockerfile: the script of constructing a new image

# Container, Docker and OS

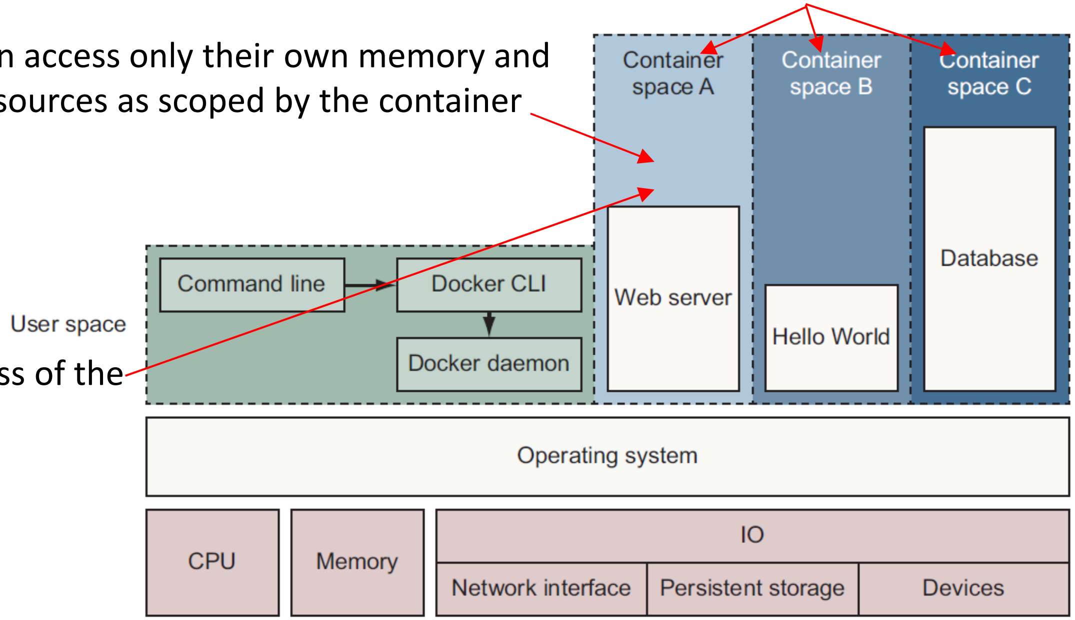
Without container



running in its own memory space

can access only their own memory and resources as scoped by the container

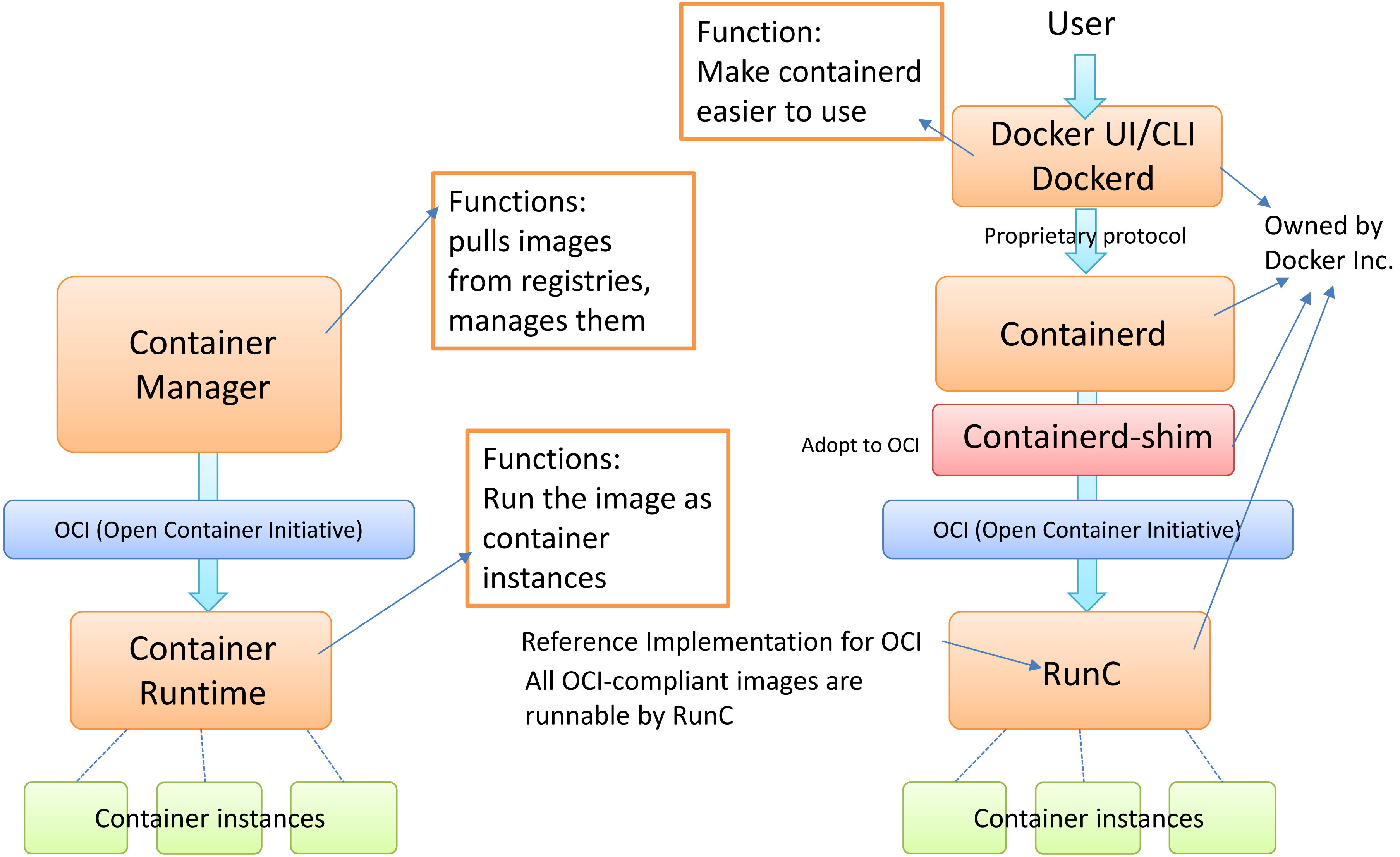
With container



Each container is a child process of the Docker engine

還沒執行是 image
執行起來就是 container

# The Architecture of Container



Without K8S
(Only Docker)

User

Function:
Make containerd easier to use

Functions:
pulls images from registries, manages them

Functions:
Run the image as container instances

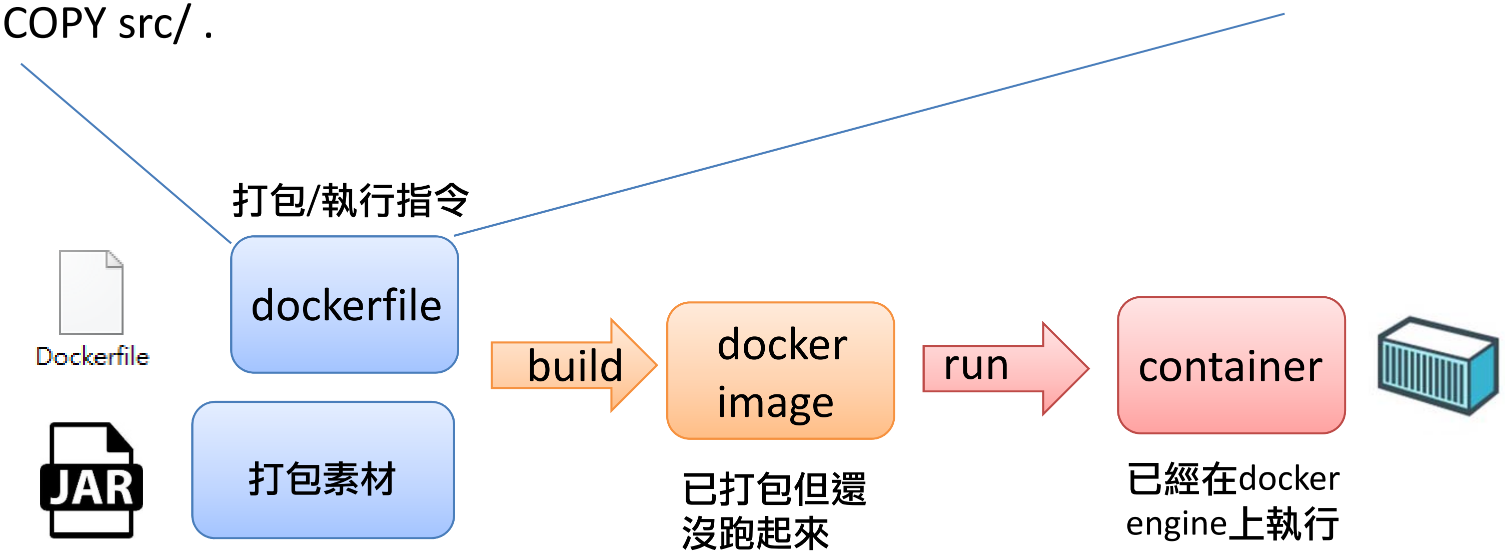
Reference Implementation for OCI
All OCI-compliant images are runnable by RunC

# Core technologies of Docker

- Linux namespace
  - PID namespace—Process identifiers and capabilities
  - UTS namespace—Host and domain name
  - MNT namespace—File system access and structure
  - IPC namespace—Process communication over shared memory
  - NET namespace—Network access and structure
  - USR namespace—User names and identifiers
- cgroups: resource isolation
- chroot: controls the location of the file system root
- unionfs

# 開發與佈署

```
FROM node:14-alpine
EXPOSE 80
cmd ["node", "server.js"]
WORKDIR /app
COPY --from=builder /src/node_modules/ /app/node_modules/
COPY src/ .
```



打包/執行指令

Dockerfile

dockerfile

JAR

打包素材

build

docker image

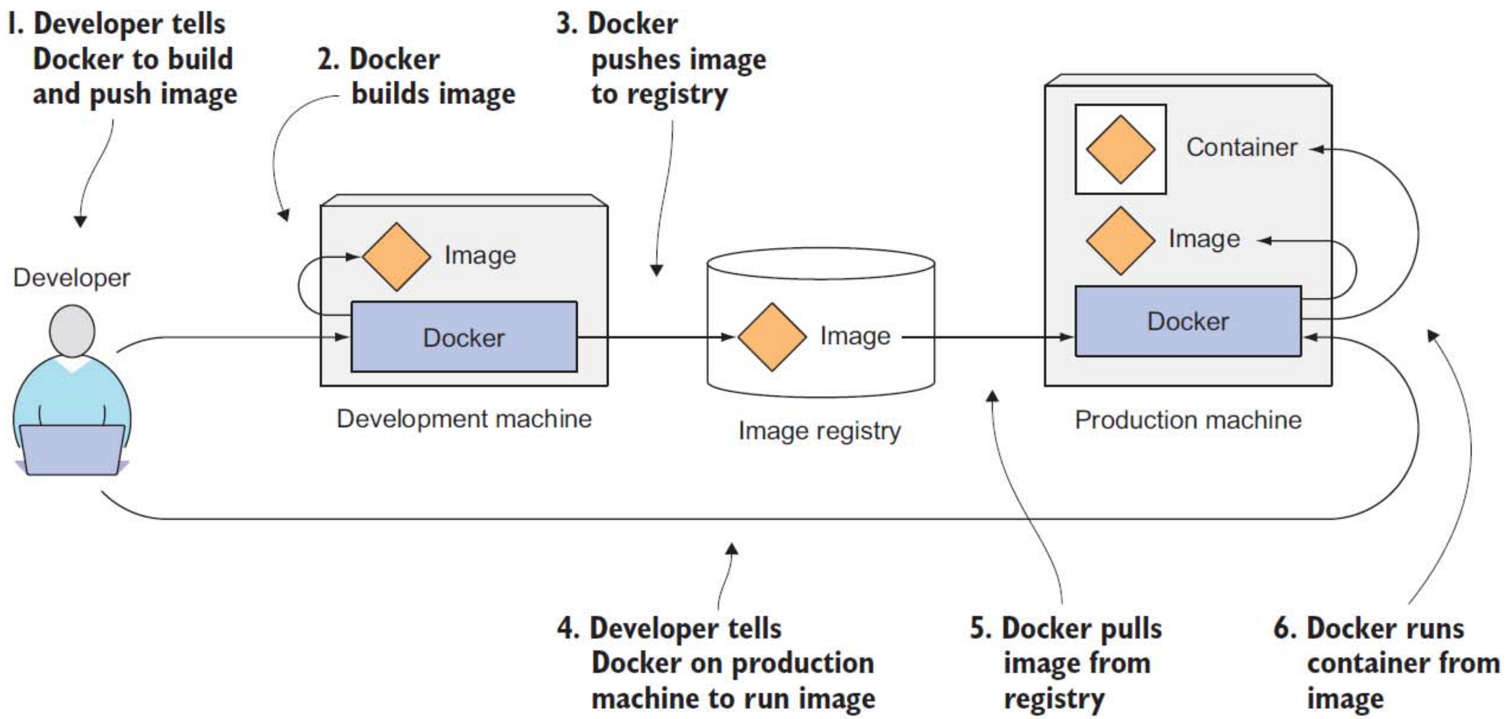
已打包但還沒跑起來

run

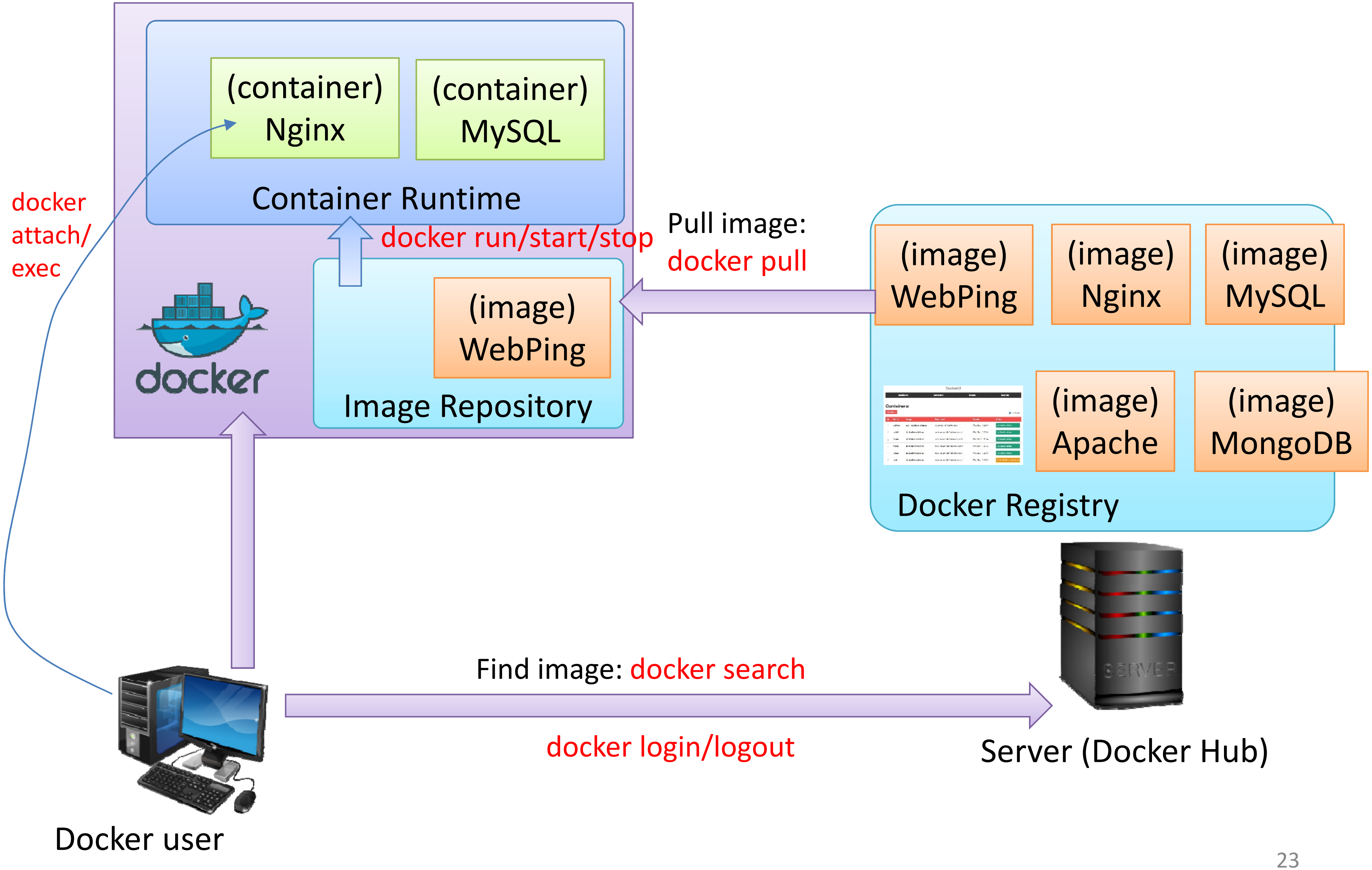
container

已經在docker engine上執行

# Docker的開發與佈署


1. Developer tells Docker to build and push image
2. Docker builds image
3. Docker pushes image to registry
Developer
Image
Docker
Development machine
Image
Image registry
Container
Image
Docker
Production machine
4. Developer tells Docker on production machine to run image
5. Docker pulls image from registry
6. Docker runs container from image

# Docker 基本操作



(container) Nginx

(container) MySQL

Container Runtime

docker attach/ exec

docker run/start/stop

(image) WebPing

Image Repository

Pull image: docker pull

(image) WebPing

(image) Nginx

(image) MySQL

(image) Apache

(image) MongoDB

Docker Registry

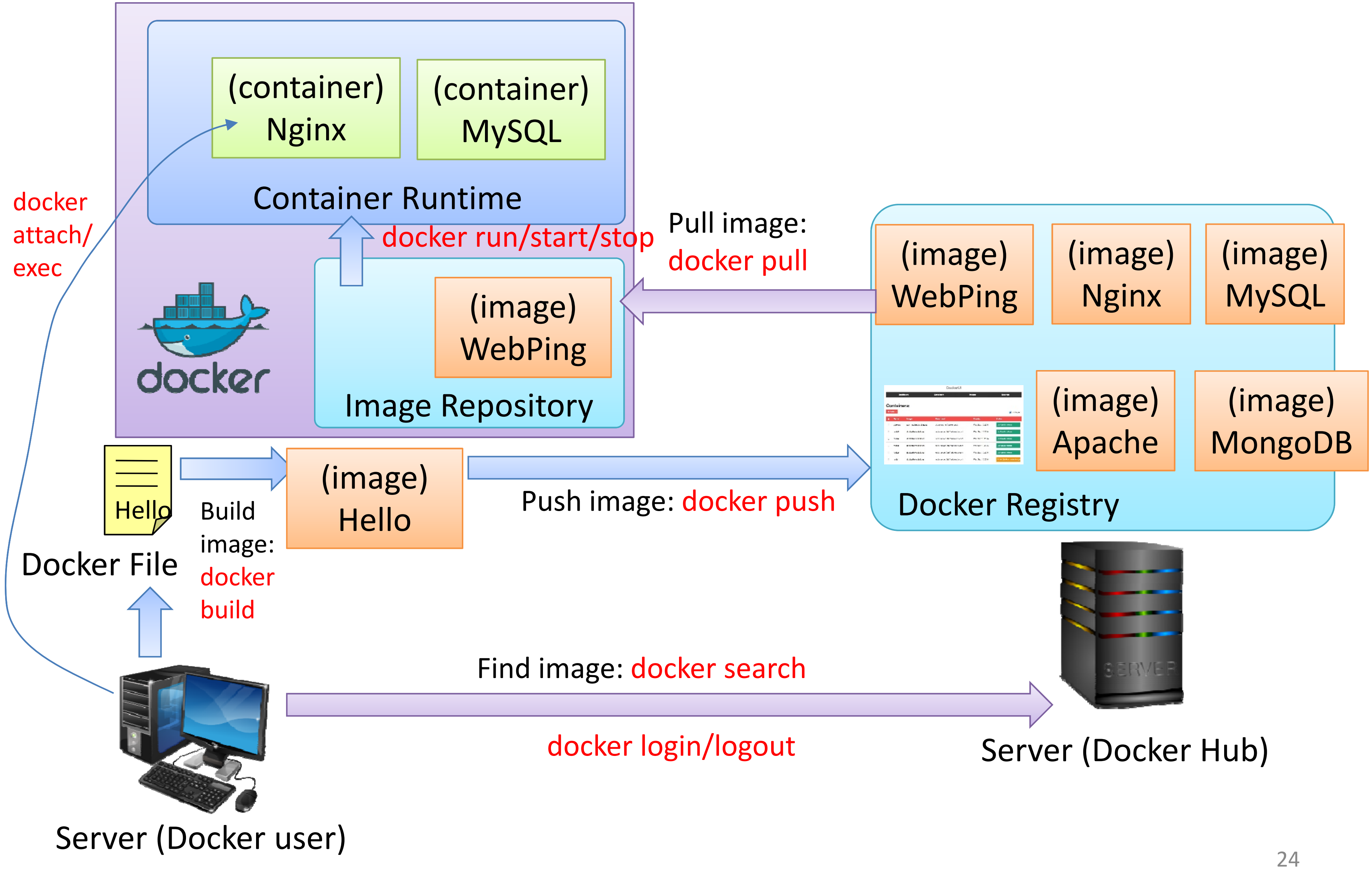
Find image: docker search

docker login/logout

Server (Docker Hub)

Docker user

# Docker 基本操作



(container) Nginx

(container) MySQL

Container Runtime

docker run/start/stop

(image) WebPing

Image Repository

docker attach/ exec

Pull image: docker pull

(image) WebPing

(image) Nginx

(image) MySQL

(image) Apache

(image) MongoDB

Docker Registry

Hello

Docker File

Build image: docker build

(image) Hello

Push image: docker push

Find image: docker search

docker login/logout

Server (Docker Hub)

Server (Docker user)

# Demo: container 操作

- docker container run hello-world
- docker image ls
- docker container run –it diamol/base
  - ps
  - hostname
  - ls
- docker container run –d diamol/ch02-hello-diamol-web
- docker container run –d –p 8080:80 diamol/ch02-hello-diamol-web
- 連接已執行的container
  - Docker exec –it <container-id> /bin/sh

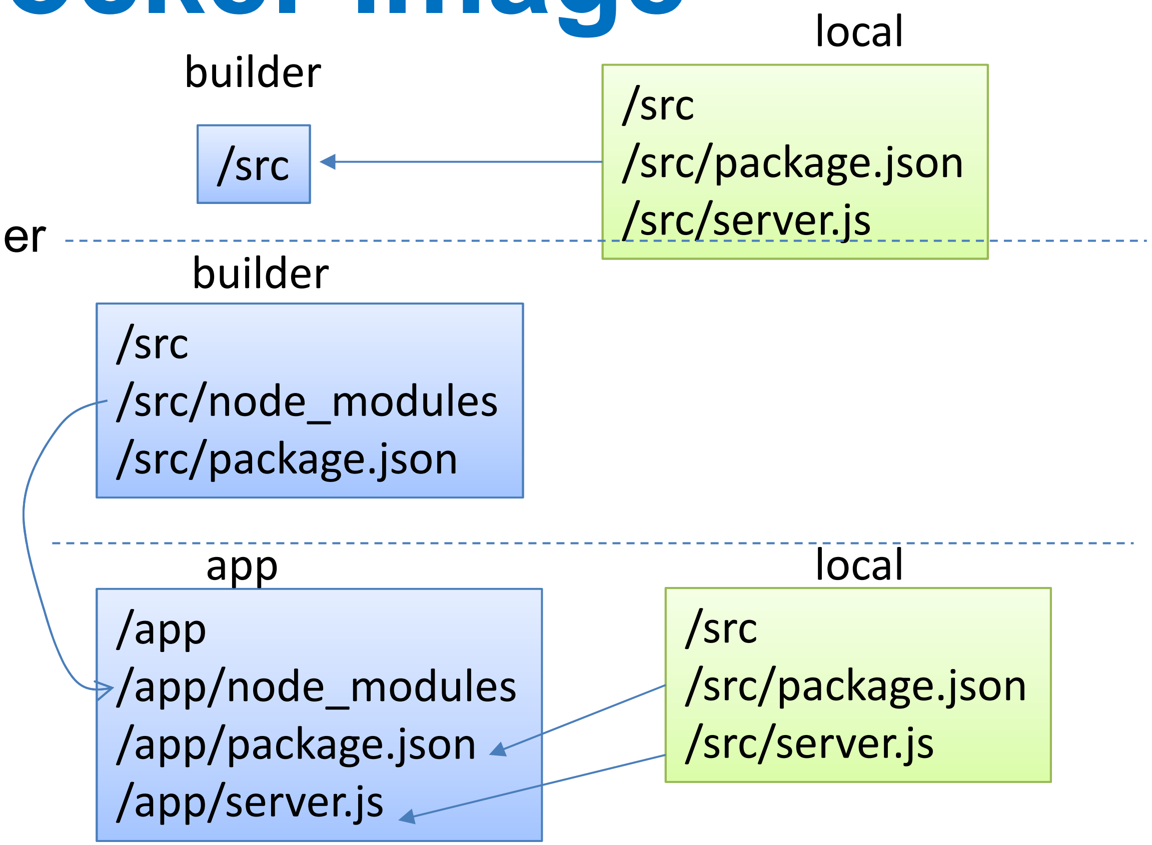
# Demo: 修改與建立docker images

FROM diamol/apache  COPY html/
/usr/local/apache2/htdocs/

- 建立
  - docker image build --tag myweb .
  - docker run -d -p 8080:80 myweb
- 發佈
  - docker login –uername xxx
    - 要登入到docker hub: new access token
  - docker image tag myweb xxx/myweb
  - docker image push xxx/myweb

# 建立Docker image

```
#builder
FROM node:14-alpine AS builder
WORKDIR /src
COPY src/package.json .
RUN npm install
#app
FROM node:14-alpine
EXPOSE 80
cmd ["node", "server.js"]
WORKDIR /app
COPY --from=builder /src/node_modules/ /app/node_modules/
COPY src/ .
```



local

/src
/src/package.json
/src/server.js

builder

/src

builder

/src
/src/node_modules
/src/package.json

app

/app
/app/node_modules
/app/package.json
/app/server.js

local

/src
/src/package.json
/src/server.js

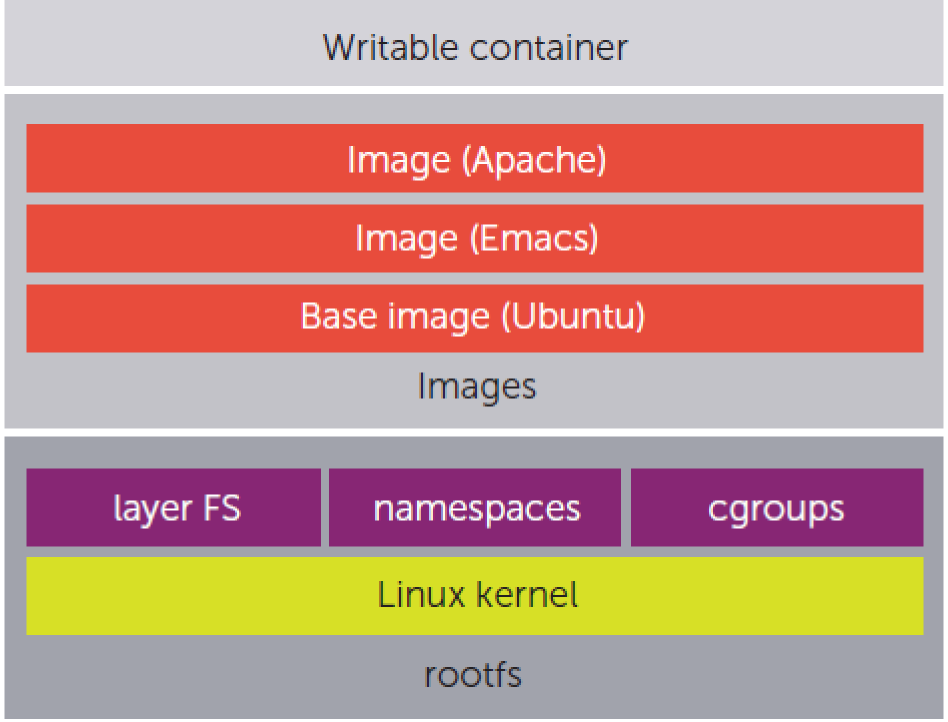
# Union FS

- UnionFS was originally developed by Prof. Erez Zadok and his team at Stony Brook University

https://unionfs.filesystems.org/

容器啟動後，其內應用程式對容器的所有改動，增刪，都只會發生在writable container這一層，不會動到原有的image

```
FROM ubuntu
RUN apt-get install emacs
RUN apt-get install apache2
CMD ["/bin/bash"]
```



Writable container

Image (Apache)

Image (Emacs)

Base image (Ubuntu)

Images

layer FS

namespaces

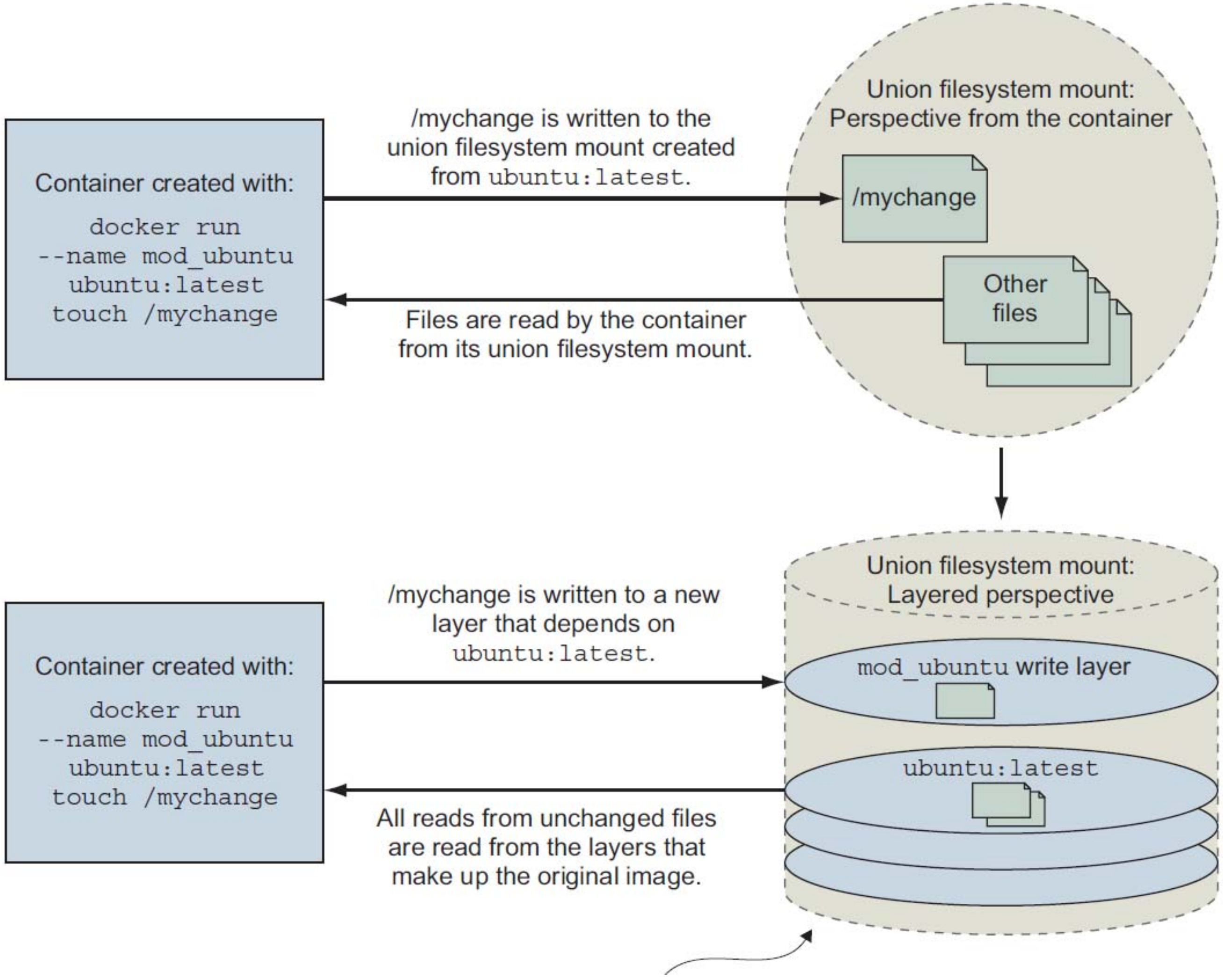
cgroups

Linux kernel

rootfs

Zadok, E., Iyer, R., Joukov, N., Sivathanu, G., & Wright, C. P. (2006). On incremental file system development. ACM Transactions on Storage (TOS), 2(2), 161-196.

# Union FS


Container created with:
docker run --name mod_ubuntu ubuntu:latest touch /mychange
/mychange is written to the union filesystem mount created from ubuntu:latest.
Union filesystem mount: Perspective from the container
/mychange
Other files
Files are read by the container from its union filesystem mount.
Container created with:
docker run --name mod_ubuntu ubuntu:latest touch /mychange
/mychange is written to a new layer that depends on ubuntu:latest.
Union filesystem mount: Layered perspective
mod_ubuntu write layer
ubuntu:latest
All reads from unchanged files are read from the layers that make up the original image.

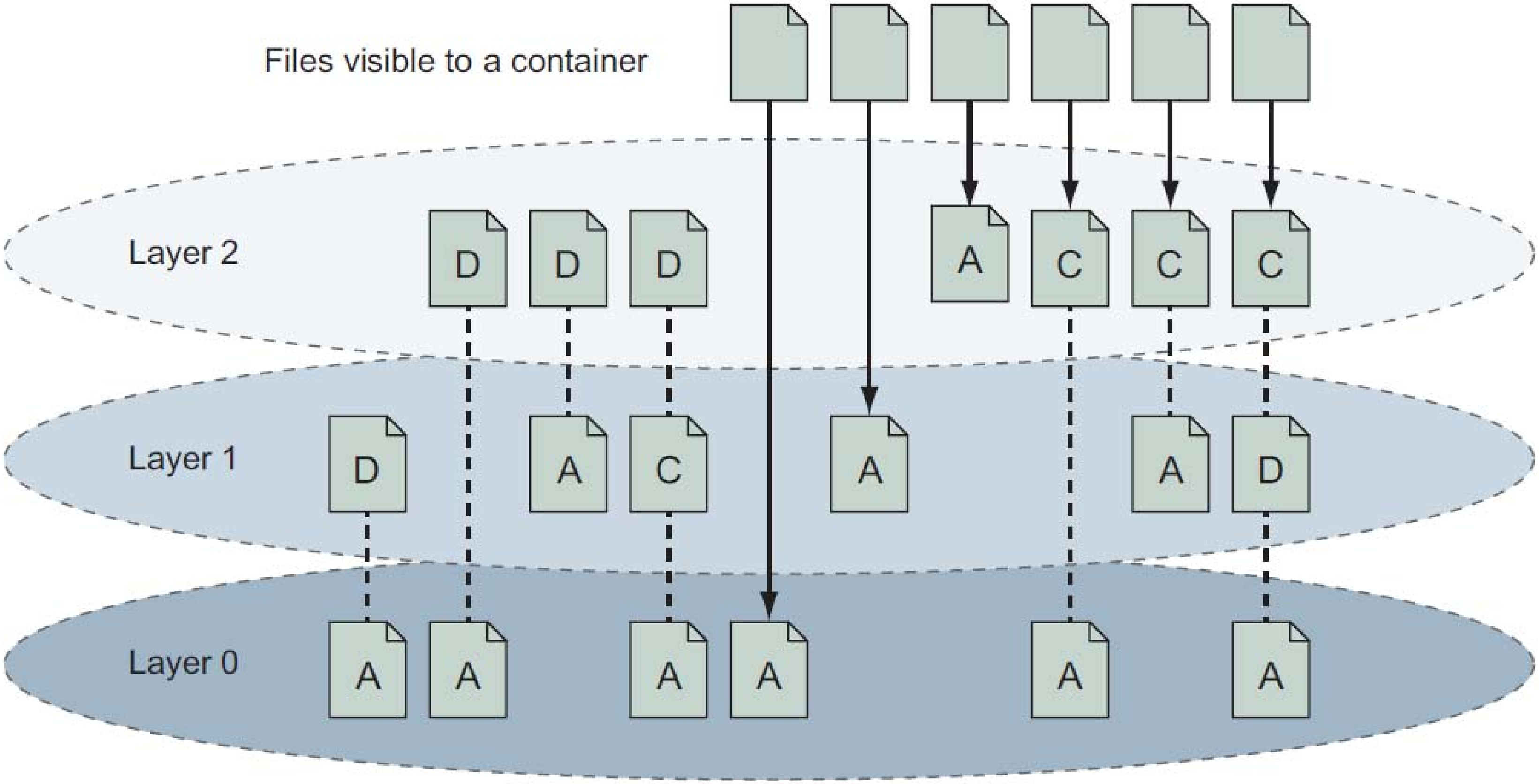
應用層觀點

檔案層觀點

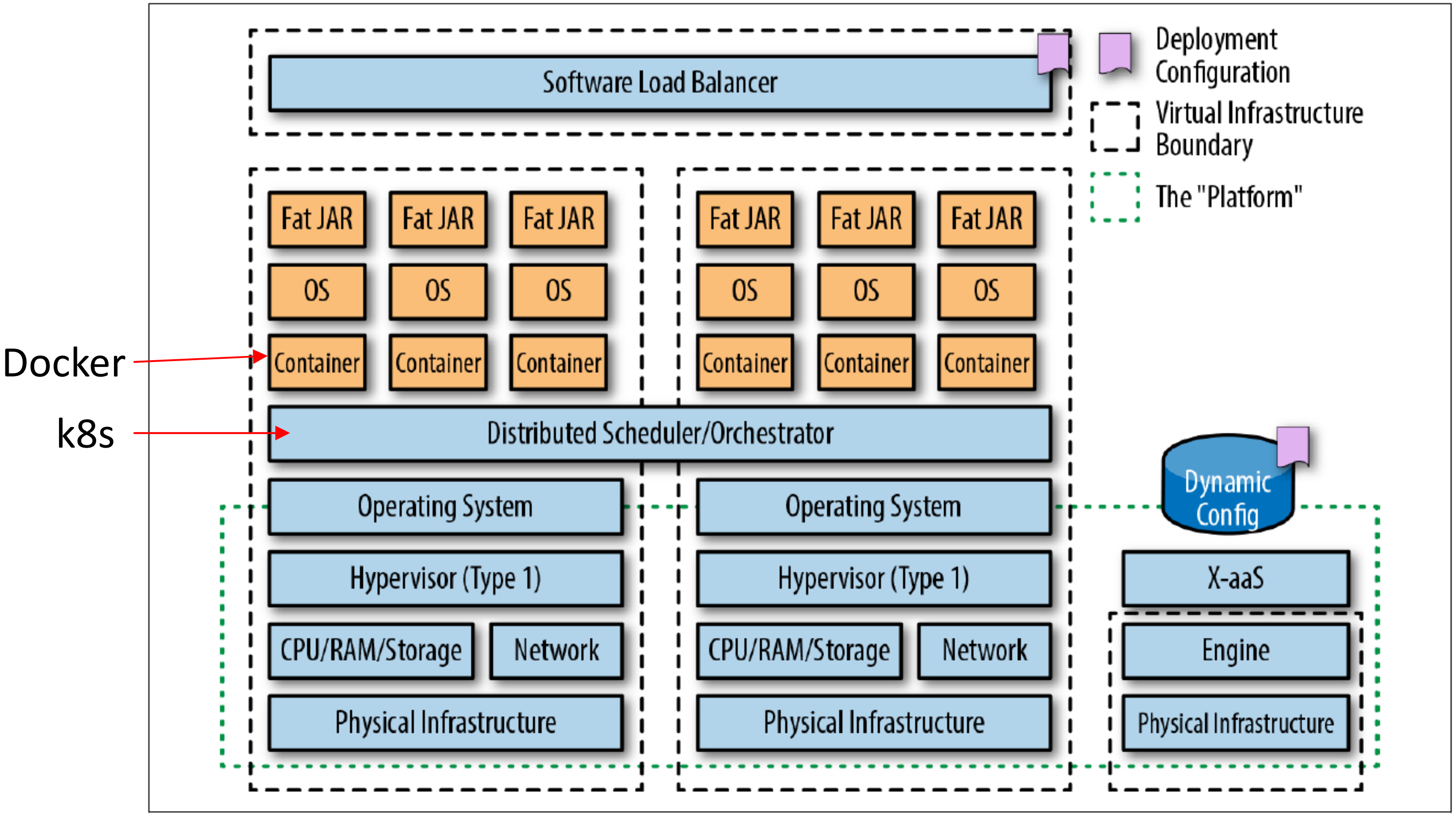
Writable container layer

原來下傳的 image

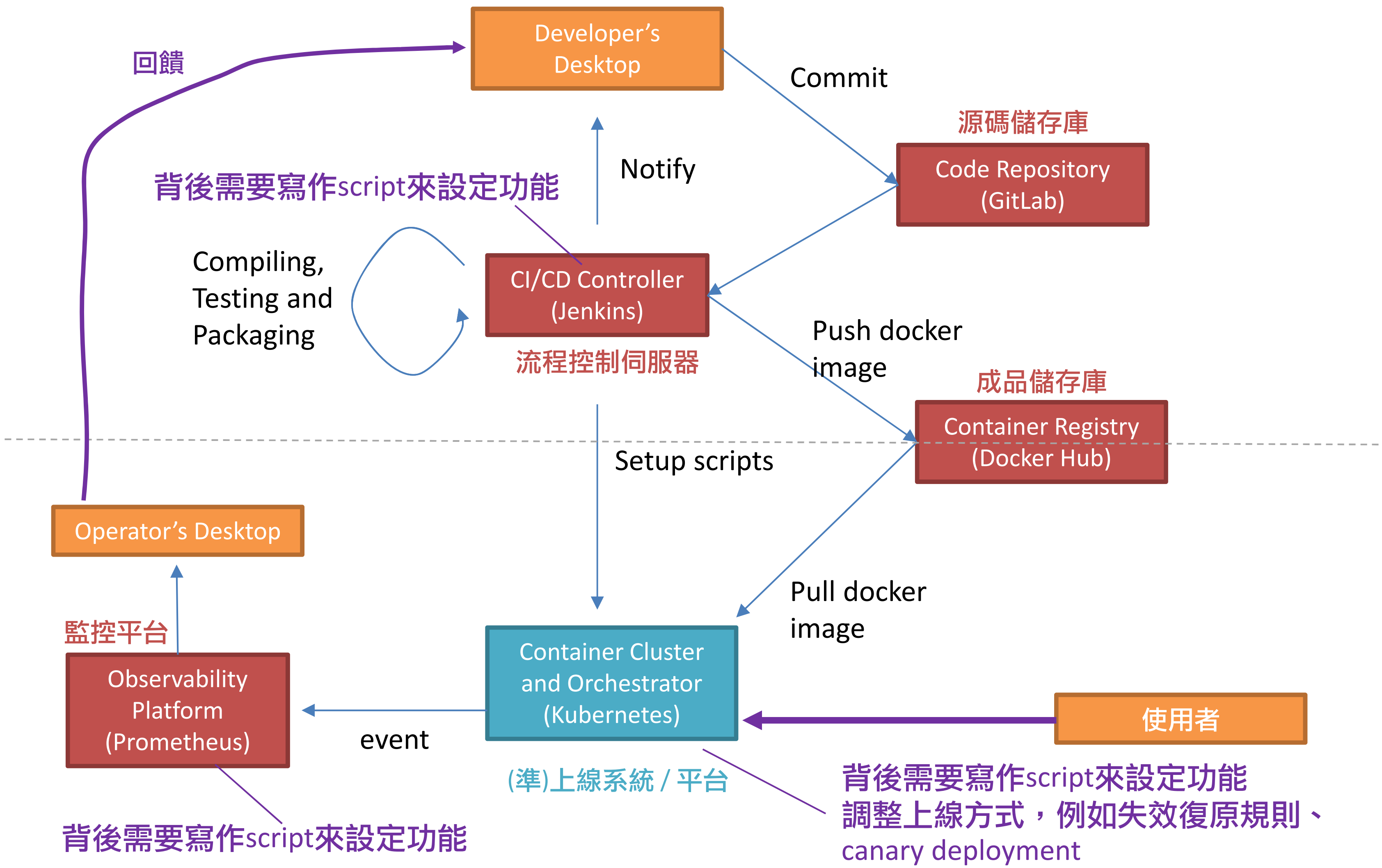
# Union FS：增刪修改的套用



# Cloud Native 系統服務的運行



# The "Cloud Native" CI/CD Pipeline


回饋
Developer's Desktop
Commit
源碼儲存庫
Code Repository (GitLab)
Notify
背後需要寫作script來設定功能
Compiling, Testing and Packaging
CI/CD Controller (Jenkins)
流程控制伺服器
Push docker image
成品儲存庫
Container Registry (Docker Hub)
Setup scripts
Operator's Desktop
監控平台
Observability Platform (Prometheus)
Pull docker image
Container Cluster and Orchestrator (Kubernetes)
使用者
event
(準)上線系統 / 平台
背後需要寫作script來設定功能
背後需要寫作script來設定功能
調整上線方式，例如失效復原規則、
canary deployment

# Container Orchestration

- Purpose
  - To manage the life cycle of containers at scale
- Tasks
  - Resource management
    - placing containers on nodes that provide sufficient resources
    - moving containers to other nodes if the resource limits is reached
    - Health monitoring  and restarting
    - Scaling in or out
  - Networking
    - Providing mappings for containers to connect to networking
    - Internal load balancing between containers

# Kubernetes

- An open source project for container orchestration
  - Contributed by Google in 2014
  - Borg: the platform behind Google Search, Gmail, and YouTube
  - Kubernetes leverages Borg’s innovations and lessons learned
- Competitors
  - Apache Mesos
  - Docker Swarm

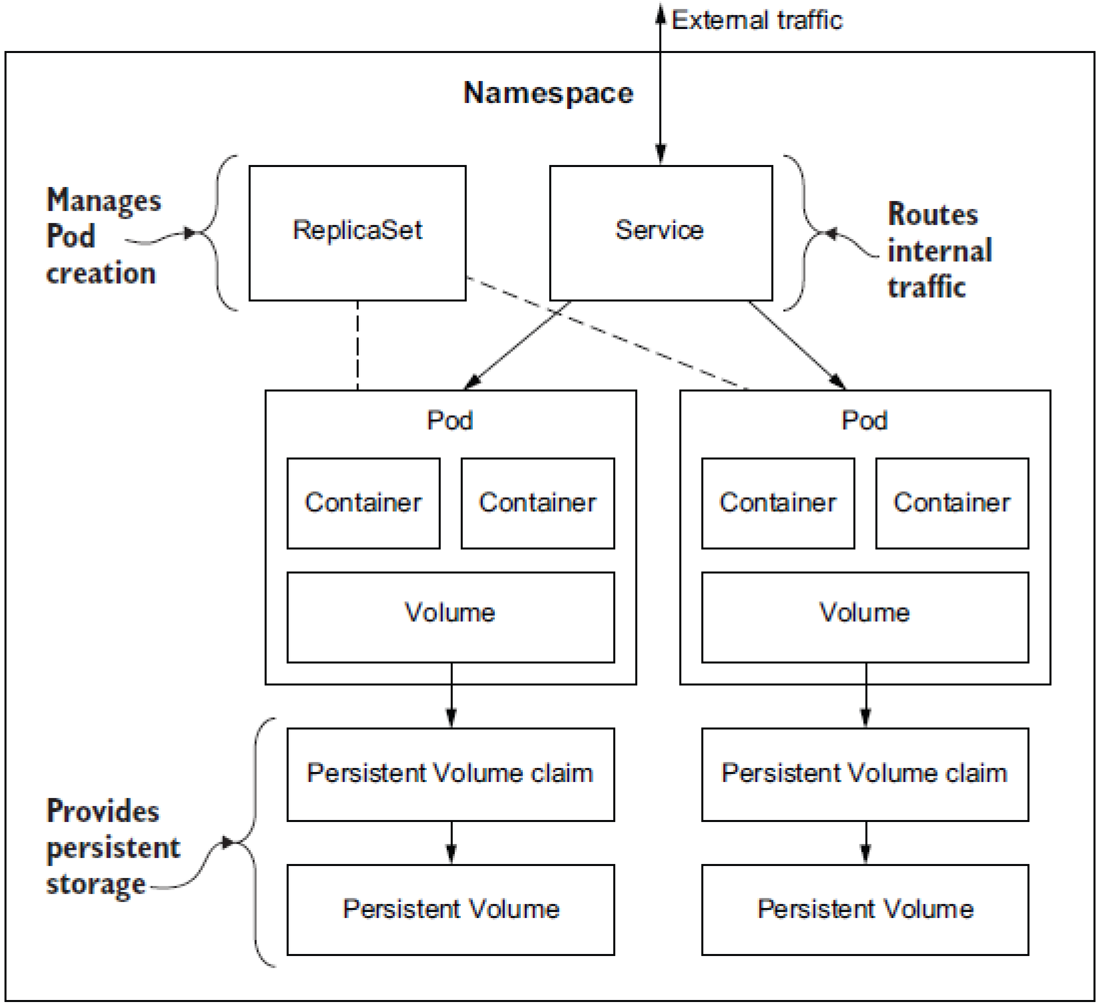
# Why we need Kubernetes?

- What Kubernetes does
  - Service-orientation
    - Service discovery via internal DNS; Runtime binding of address
  - Ad hoc cluster-wide LAN
  - Load balancing (horizontal scaling)
  - Self-healing (by restart)
  - Rollout and rollback
    - Change to specific version automatically and on demand

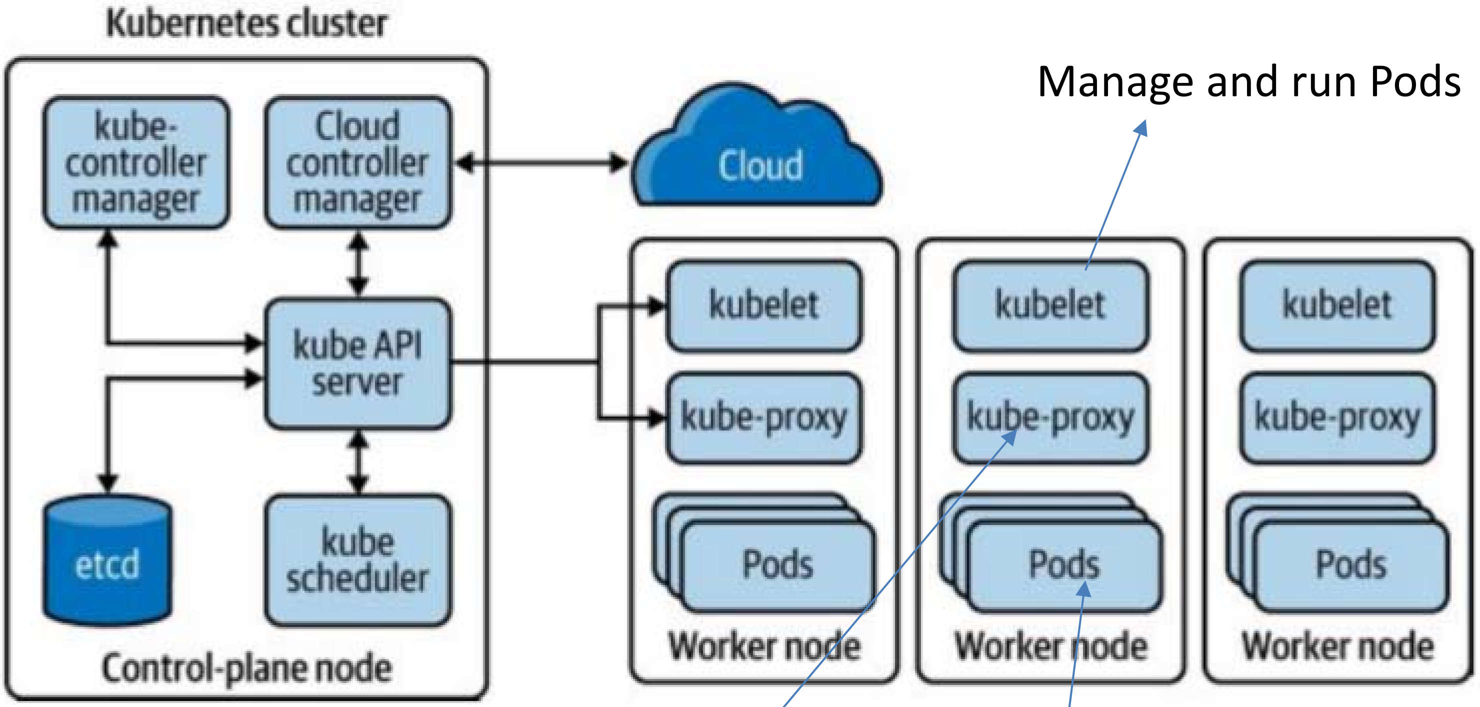
# Why we need Kubernetes?

- What Kubernetes doesn’t do
  - Diagnose the problems/bugs
  - Cross-cutting application-level services
    - Transactions, ORM, RPC….
  - Build container image
    - Pull OCI-compliant images from image registry
    - Should not build image in the cluster environment

# Logical View



# Kubernetes: a high-level view

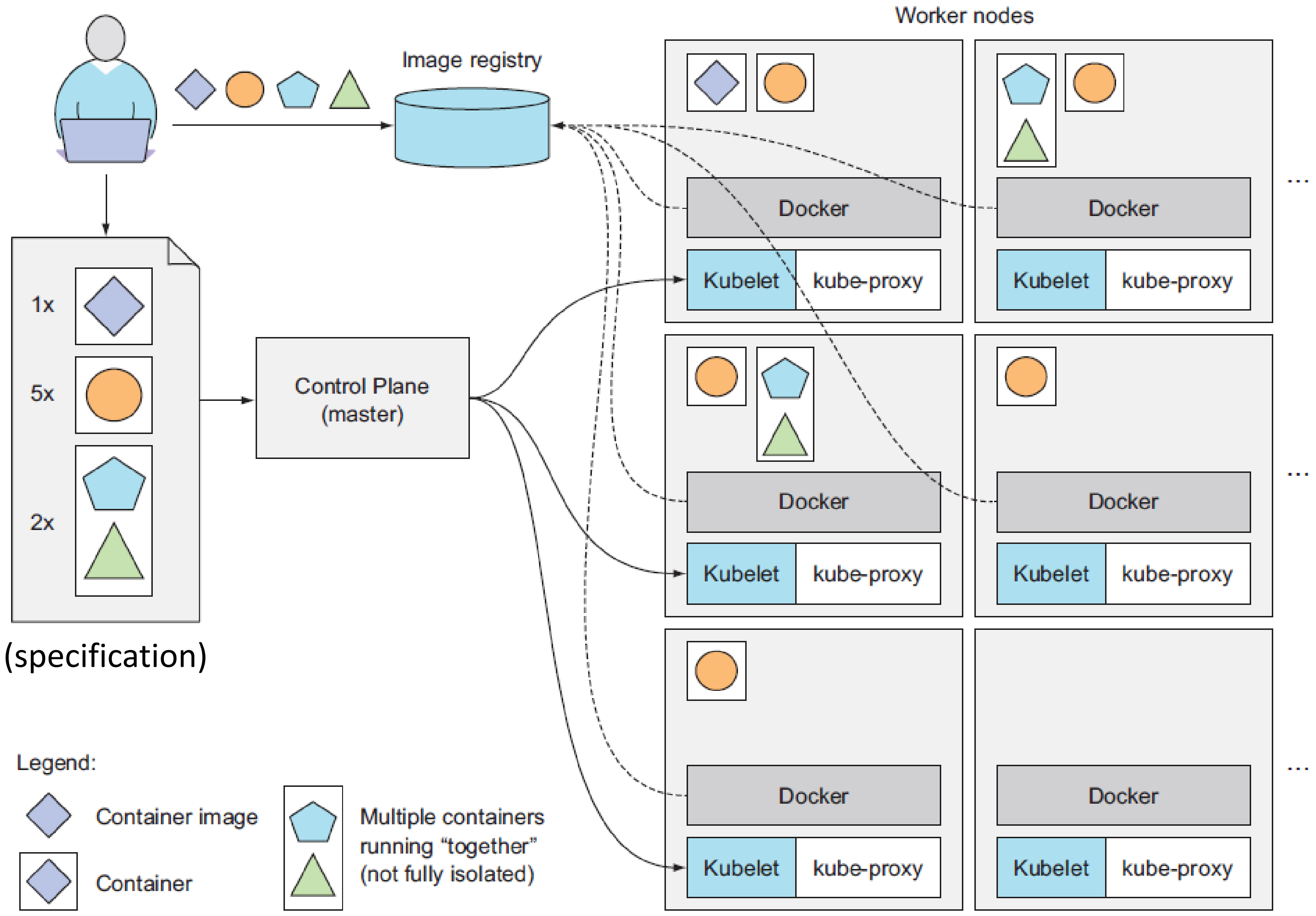


Manage and run Pods

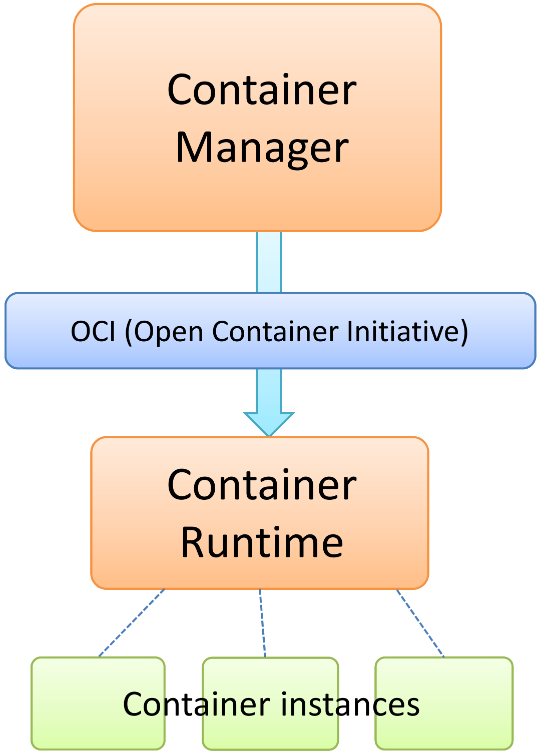
Create and manage ad hoc cluster networks

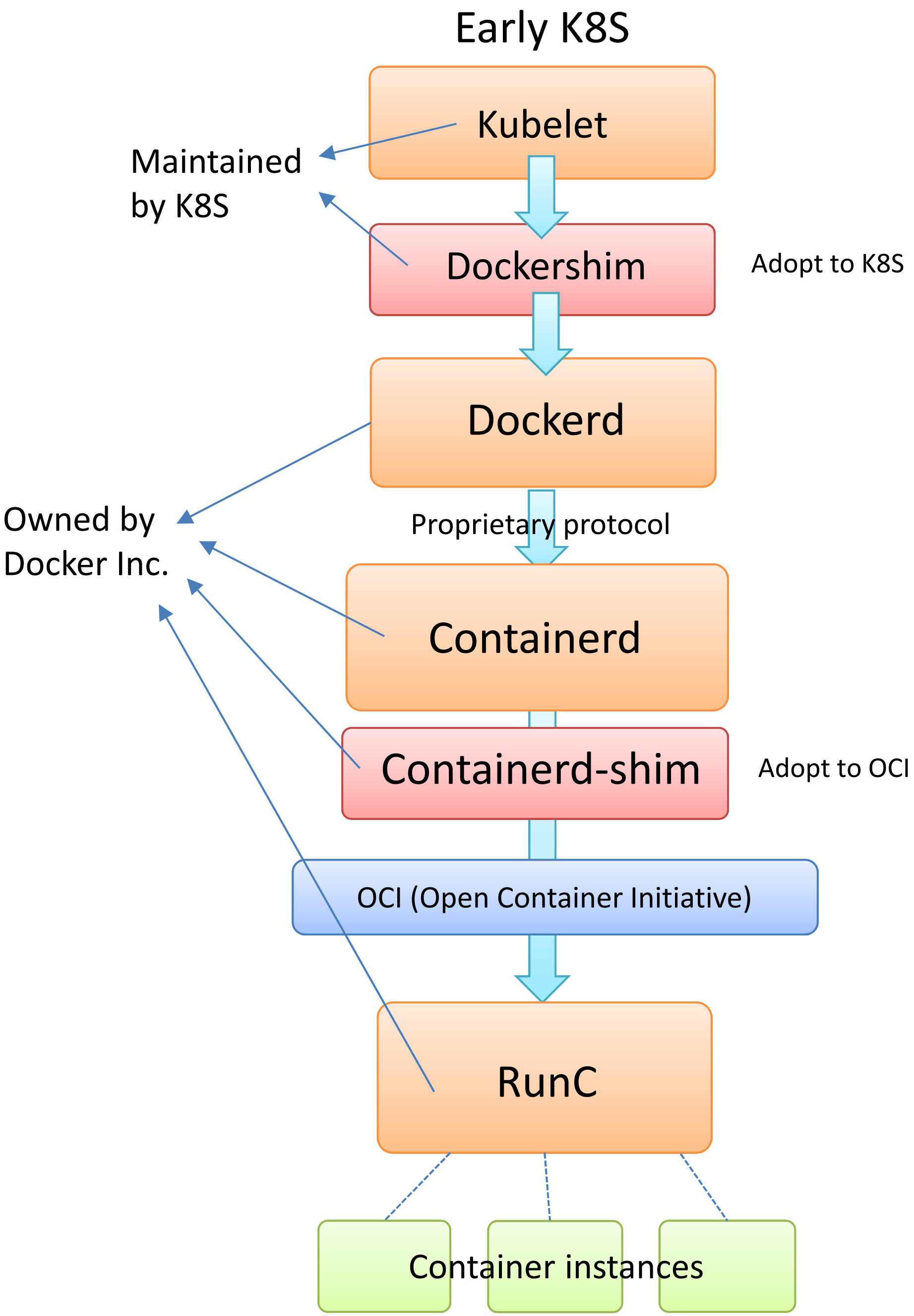
A pod may includes one or more container instances
Pod is the unit of deployment in K8S

# Kubernetes運行架構

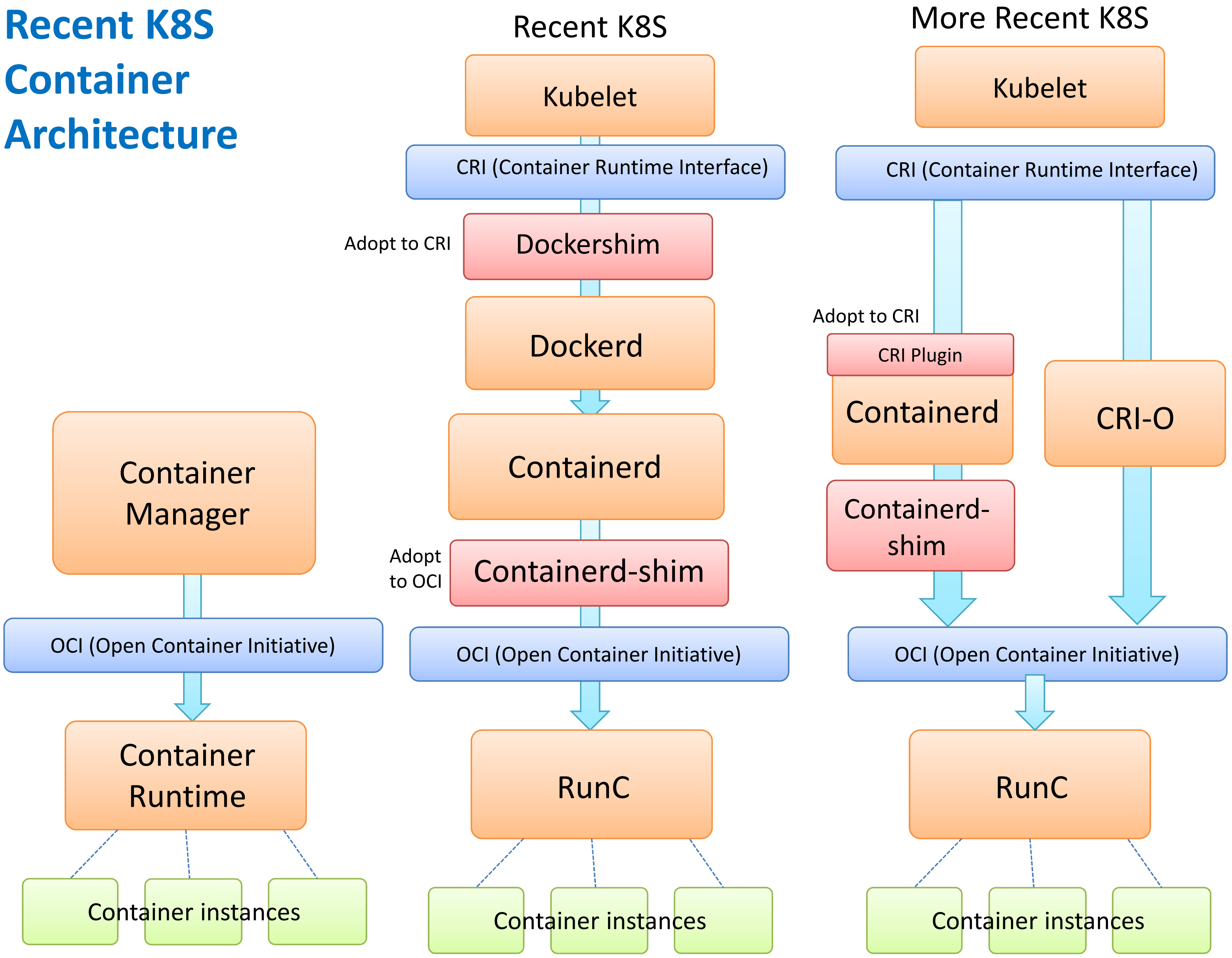


# Early K8S Container Architecture





# Recent K8S Container Architecture

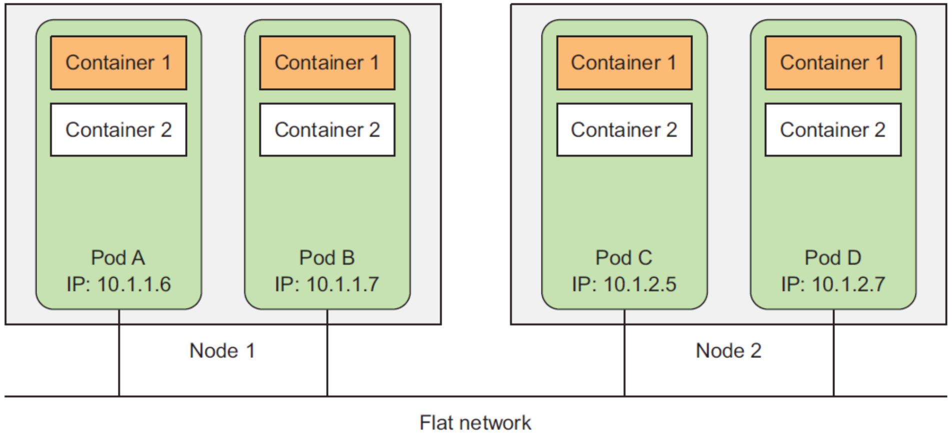


# (單機測試用) minikube安裝

- 安裝
  - https://minikube.sigs.k8s.io/docs/start/
  - 有各平台詳細說明; windows較複雜
- Windows
  - 下傳minikube Windows installer
  - windows features-> enable hyperv
    - 控制台/程式和功能/開啟或關閉windows功能
  - bcdedit /set hypervisor launchtype auto (要有系統管理員權限)
  - minikube config set driver hyperv
  - minikube 指令
    - start/stop
    - status
    - ssh
    - service <service-name> (用來取得可連上的測試網址外部IP)
    - dashboard

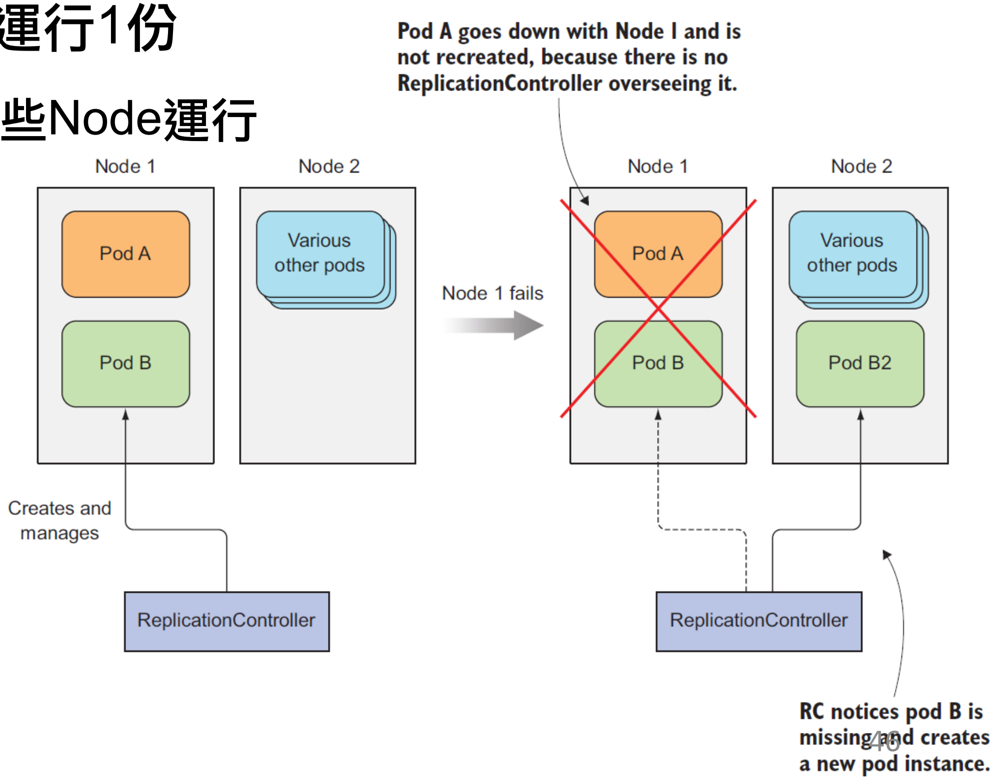
# 重要元素

- Pod
  - 一個Pod可放置1到多個container instances (通常是1個)
    - 同pod中的containers只有partial isolation
    - 同Pod中的containers共享資源
      - PID (disable by default), UTS, MNT, IPC, NET
      - Ex: share the same IP address (containers 可透過localhost:port來互相存取)
  - Pod是k8s做整體調控的基本單位
  - 屬於同k8s cluster中的Pods，概念上同屬於一個網路(no NAT)
    - 彼此可透過IP直接相互存取
- Node
  - 主機 (實體或虛擬)
  - 一個Node可內含多個Pods



# 重要元素

- ReplicaSet
  - 確保Pods (至少n個副本)的運行
  - Replication Controller的後繼者
- DaemonSet
  - 確保某個Pod在每個Node均運行1份
    - 也可以透過進階設定只在某些Node運行



Pod A goes down with Node 1 and is not recreated, because there is no ReplicationController overseeing it.

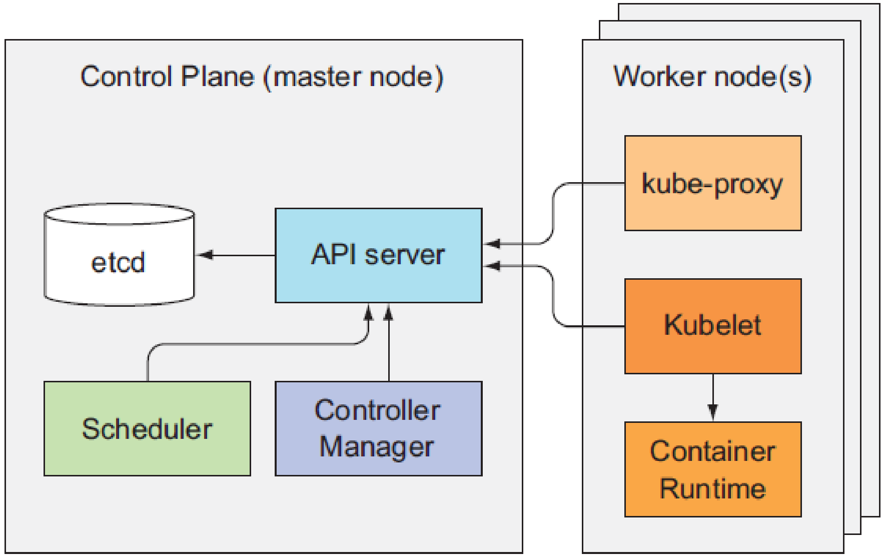
RC notices pod B is missing and creates a new pod instance.

# 重要元素

- Service
  - 做為存取Pods的統一入口
    - Expose pods at a single and stable IP/Port
    - 如果是外部，需要配合定義對外接口
    - 內部可直接存取
- Volume
  - 用來bind pod外部檔案系統

# Kubernetes Infrastructure

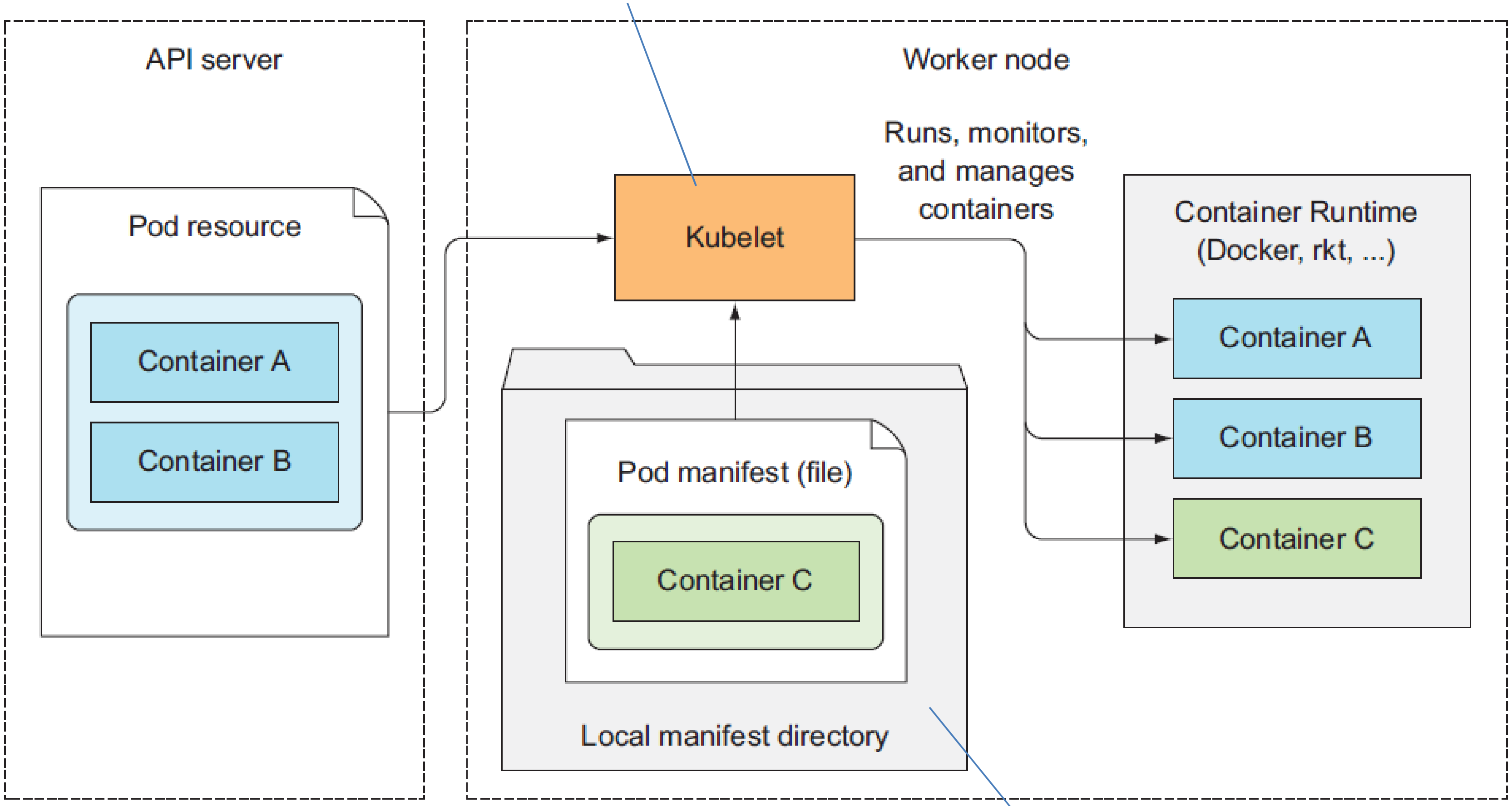
- Master Node: container cluster control pane
  - Kube-apiserver
  - etcd (KV store)
  - kube-scheduler (deploy new container to pods)
  - Kube-controller-manager
- Worker Node
  - Kubelet (local manager of pods)
  - Kube-proxy (network mapping)
  - Container runtime



K8s上所有的系統管理元件不直接溝通; 而是透過API Server溝通
Etcd也只被API Server維護
Master node上的系統元件，也可以變成pod方式運行
(此時master node上也要運行kubelet)

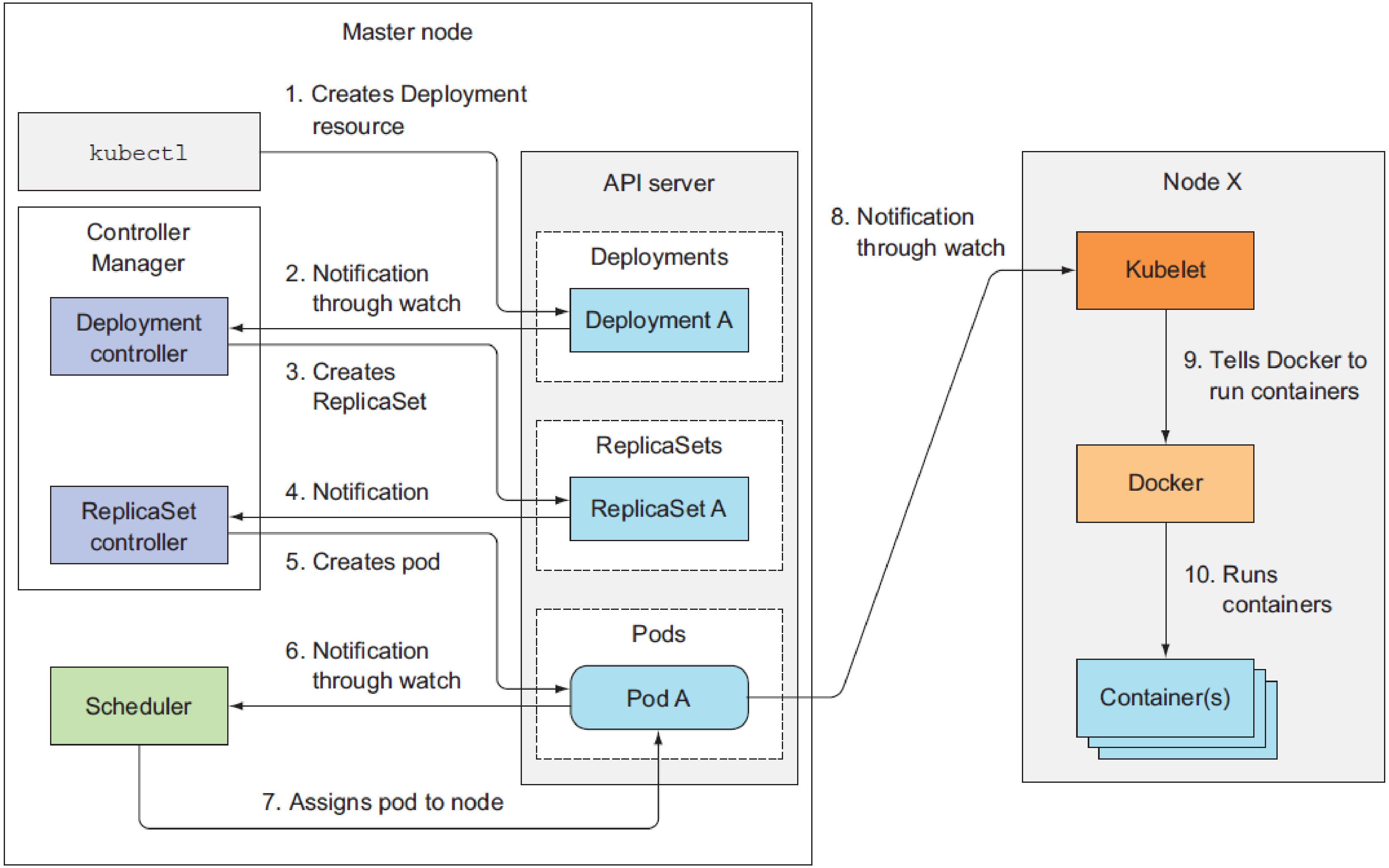
# Kubelet功能

Kubelet不斷詢問API Server是否有要新加的pods，若有，就新建pod並下傳並執行裡面的containers
Pod(與裡面的containers)建好後，kubelet向API server註冊，並持續向API Server回報pod狀況



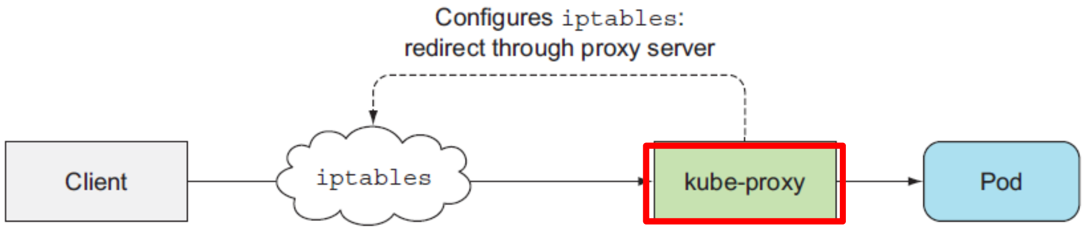
運行系統pod: 不透過分派，直接在local目錄建manifest，讓
kubelet直接跑pod (bootstrapping)

# 整體部署流程

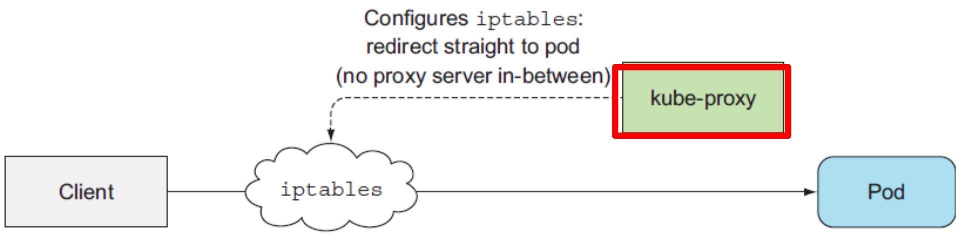

Master node
1. Creates Deployment resource
kubectl
API server
Controller Manager
2. Notification through watch
Deployments
Deployment A
Deployment controller
3. Creates ReplicaSet
ReplicaSets
4. Notification
ReplicaSet A
ReplicaSet controller
5. Creates pod
Pods
6. Notification through watch
Pod A
Scheduler
7. Assigns pod to node
8. Notification through watch
Node X
Kubelet
9. Tells Docker to run containers
Docker
10. Runs containers
Container(s)

# Kube-proxy

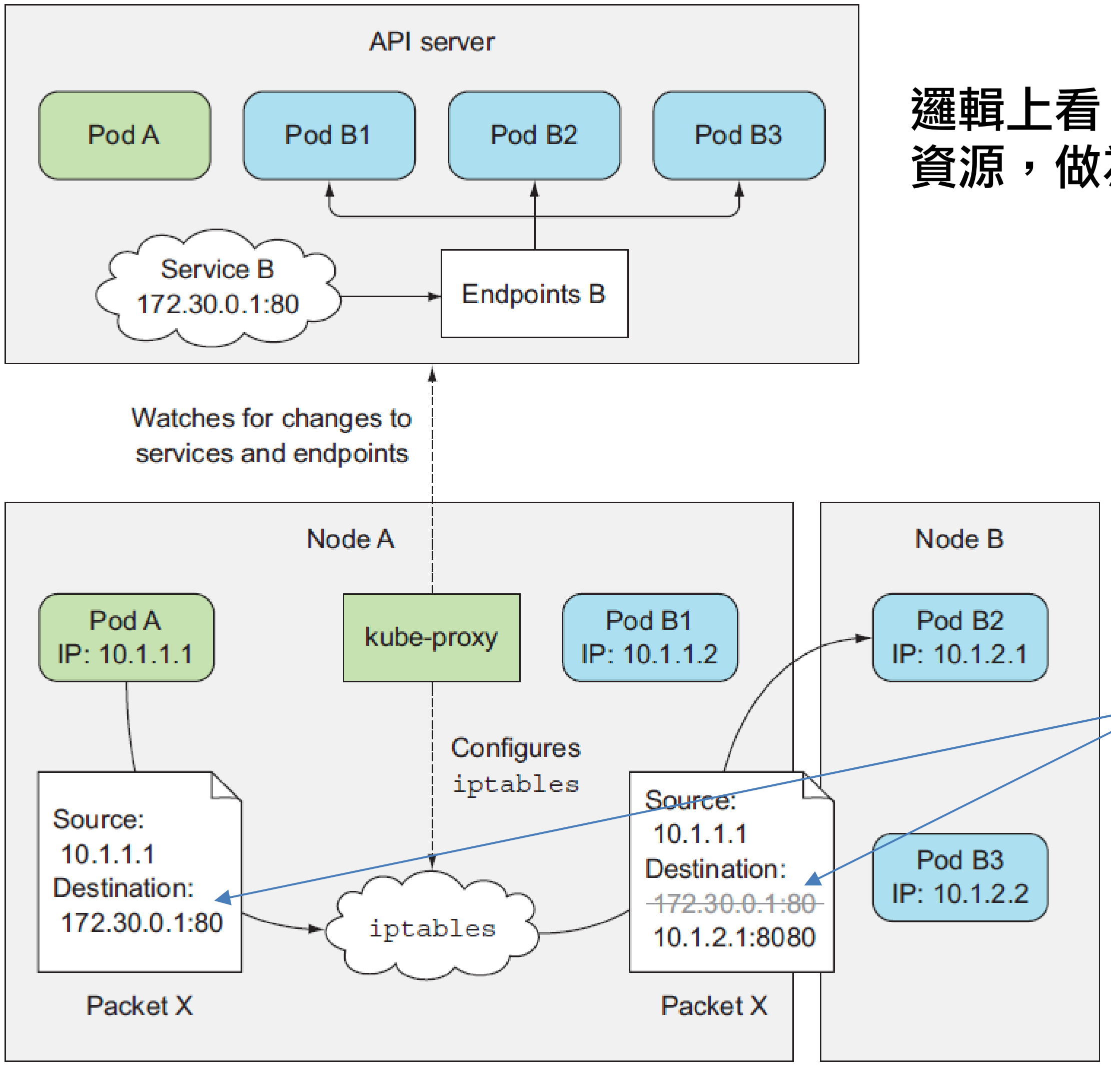
- 外部clients如何連接到直正的pods?
  - 外部clients 只知道 service的IP/Port
  - 後台的Pods只有private IPs
  - 如何找到、連到真正的pods?
- 二種方式
  - userspace proxy mode
    - 重導所有要求到kube-proxy
    - Kube-proxy as a proxy server



  - iptables proxy mode
    - kube-proxy動態設定iptables設定繞送路徑



# Kube-proxy

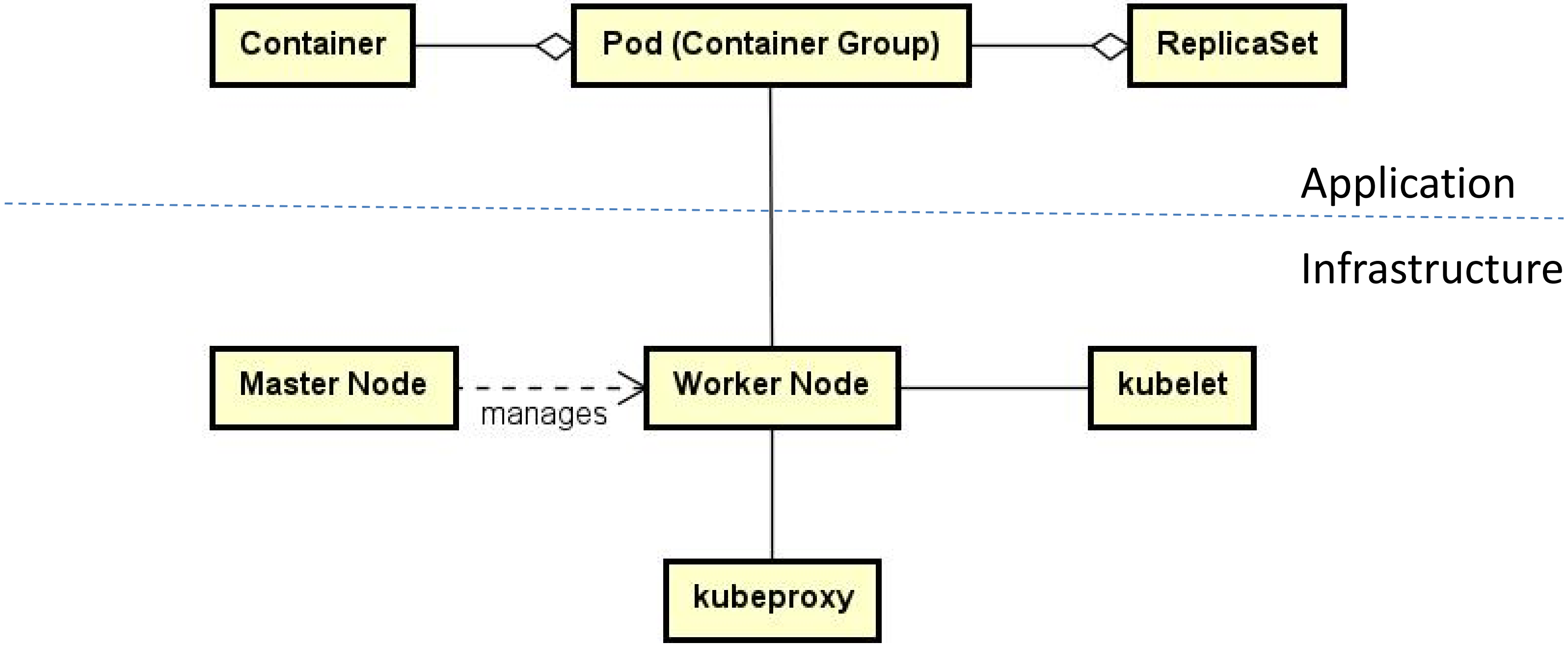


邏輯上看，ServiceB是一個有特定IP/Port的資源，做為Pod B1-Pod B3存取入口

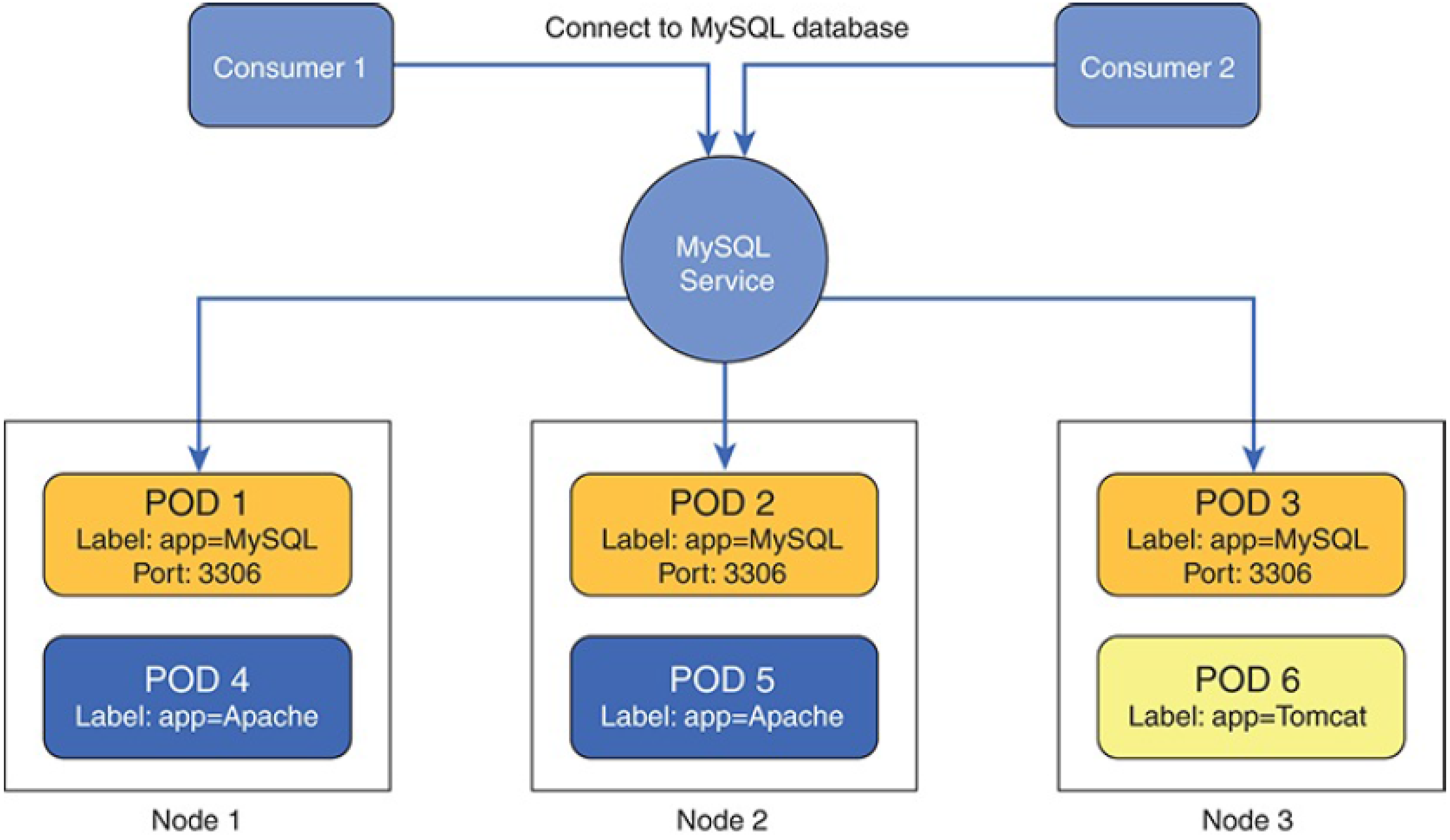
- 假設現Pod A是client要存取Service B
- Node 上的kube proxy修改iptables規則，將172.30.0.1:80 (Service B)對應到10.1.2.1:80 (Pod B2)

# Summary

- Key concepts

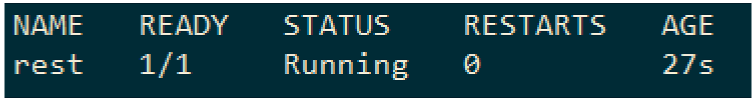


# Example:
# Accessing MySQL Service



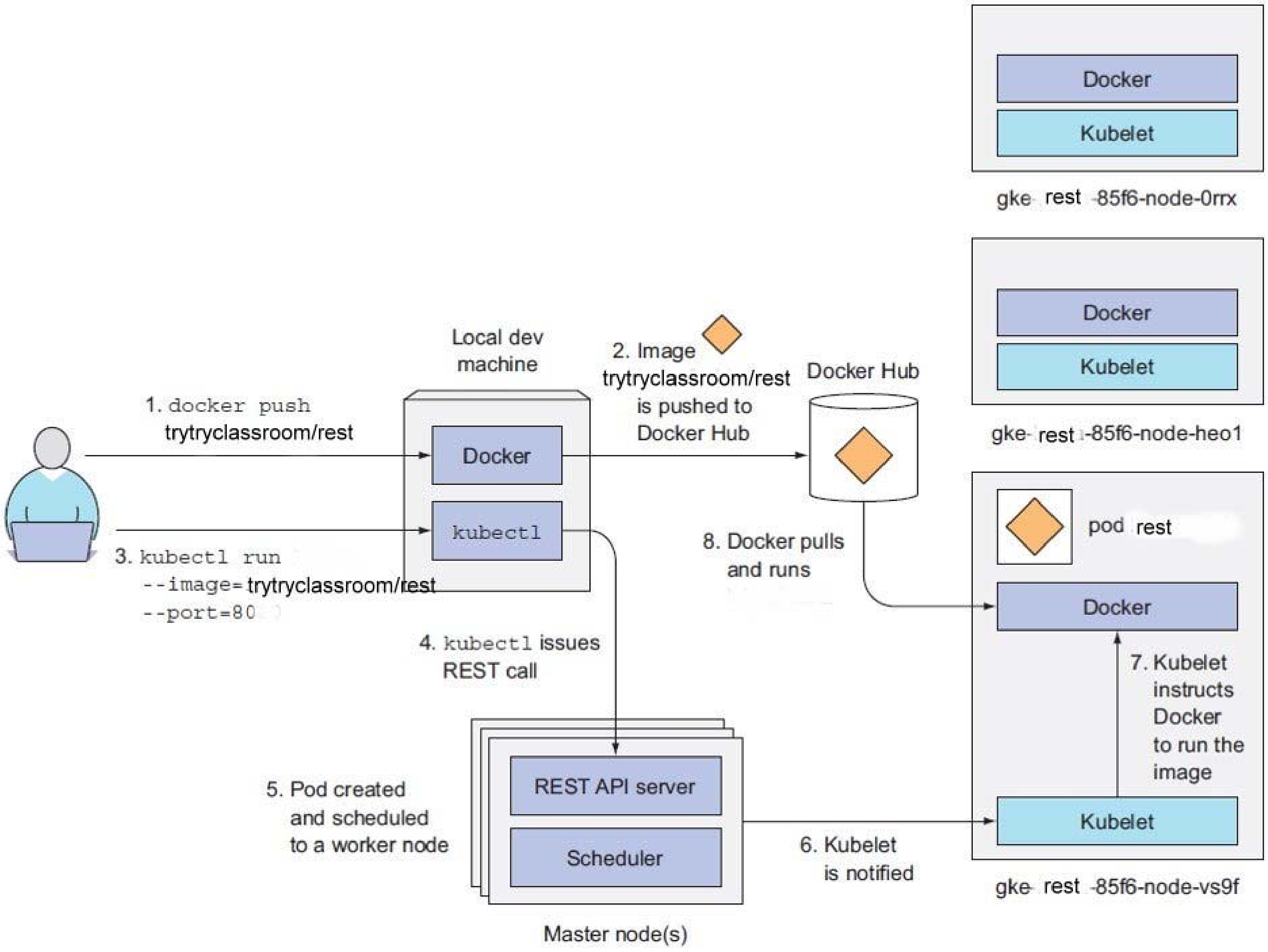
# Example

- 佈署image到k8s上
  - 準備Image，傳到docker hub: trytryclassroom/rest (80)
  - 抓取image並放到pod中
    - kubectl run rest --image=trytryclassroom/rest –port 80
    - kubectl get pods



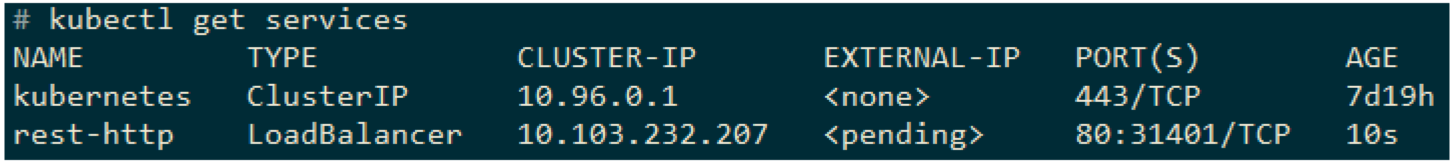
```
NAME    READY   STATUS    RESTARTS   AGE
rest    1/1     Running   0          27s
```

# Example


Docker
Kubelet
gke- rest -85f6-node-0rrx
Docker
Kubelet
gke- rest -85f6-node-heo1
Local dev machine
2. Image trytryclassroom/rest is pushed to Docker Hub
Docker Hub
1. docker push trytryclassroom/rest
Docker
kubectl
3. kubectl run --image= trytryclassroom/rest --port=80
8. Docker pulls and runs
pod rest
Docker
4. kubectl issues REST call
7. Kubelet instructs Docker to run the image
5. Pod created and scheduled to a worker node
REST API server
Scheduler
6. Kubelet is notified
Kubelet
gke- rest -85f6-node-vs9f
Master node(s)

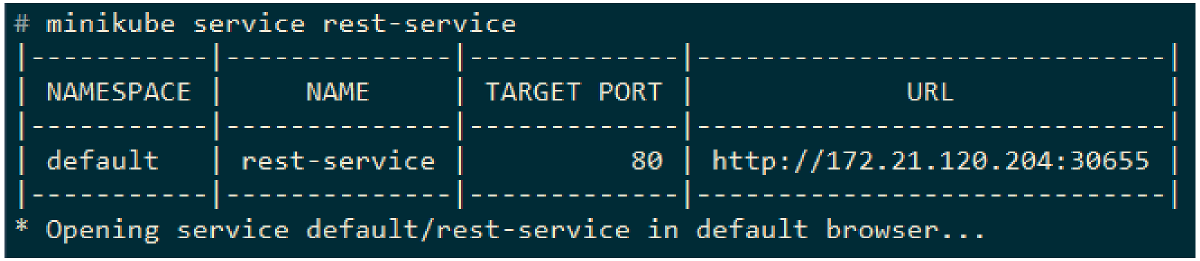
# Example

- Exposing Pod
  - kubectl expose pod rest –type=LoadBalancer –name rest-service

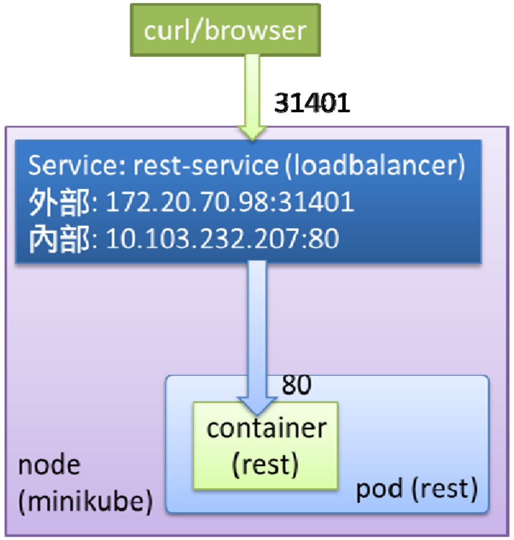


```
# kubectl get services
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP      10.96.0.1        <none>        443/TCP        7d19h
rest-http    LoadBalancer   10.103.232.207   <pending>     80:31401/TCP   10s
```

  - 找到存取點
    - minikube service rest-service

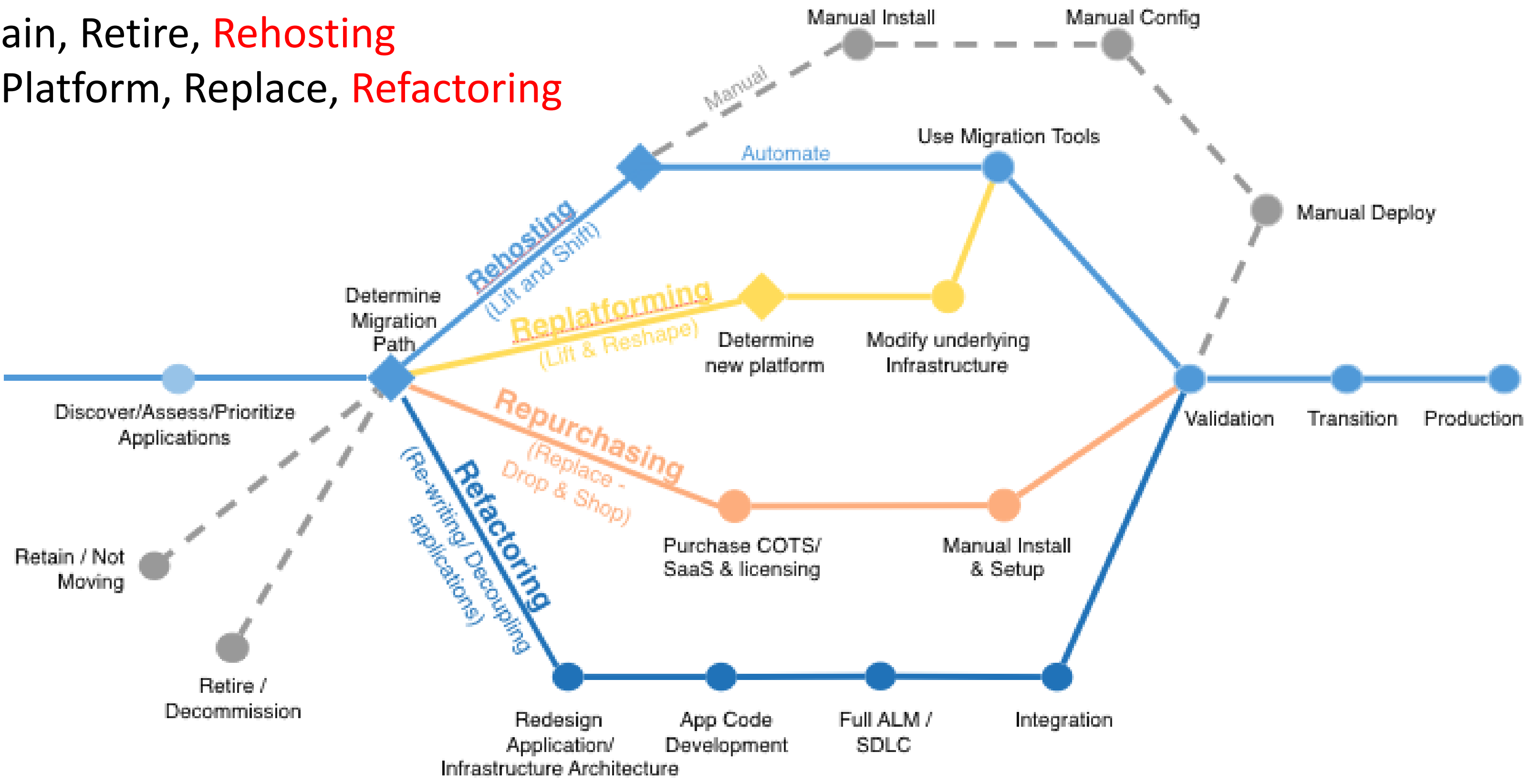


```
# minikube service rest-service
|-----------|--------------|-------------|-----------------------------|
| NAMESPACE |     NAME     | TARGET PORT |             URL             |
|-----------|--------------|-------------|-----------------------------|
| default   | rest-service |          80 | http://172.21.120.204:30655 |
|-----------|--------------|-------------|-----------------------------|
* Opening service default/rest-service in default browser...
```



# The 6 Rs: Strategies for Migrating Applications to the Cloud

Retain, Retire, Rehosting
Re-Platform, Replace, Refactoring



https://aws.amazon.com/de/blogs/enterprise-strategy/6-strategies-for-migrating-applications-to-the-cloud/

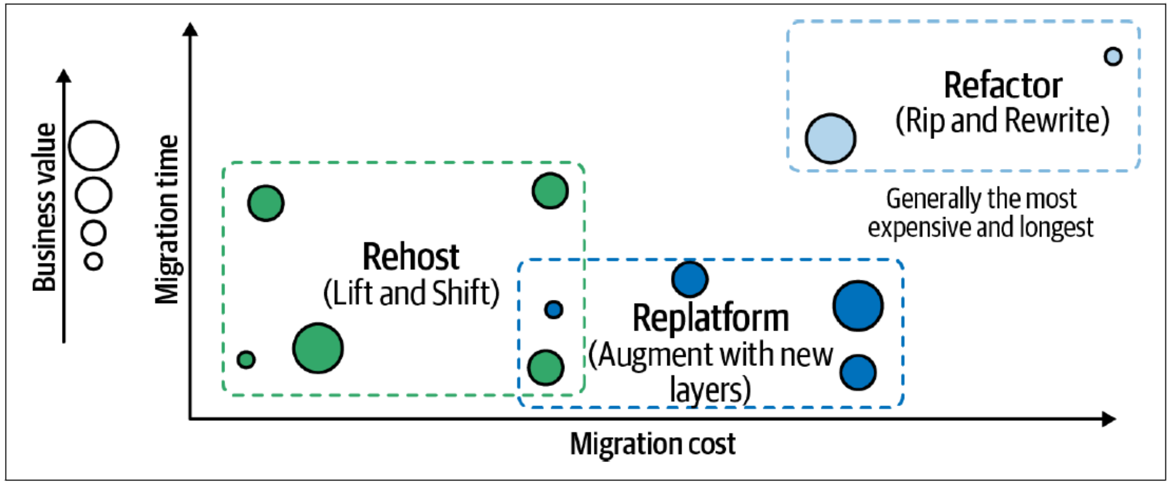
# Rehost: Lift and Shift

- Approach
  - Simple port existing app as-is to run inside a container
- Main challenges
  - JVM optimizing
  - Storage
    - Transaction
    - In memory session data
    - Persistent volume mapping
  - Need to perform sufficient research and testing

Eisele, M. & Vinto, N., (2021) Modernizing Enterprise Java: A Concise Cloud Native Guide for Developers, Oreilly Media Inc.

# Re-platform and Refactoring

- Re-platform
  - Ex: weblogic to tomcat
  - Ex: Java EE to pure spring framework
- Refactoring
  - Monolithic to microservices



# Q & A