# Kubernetes入門實作
# 與
# TensorFlow 整合應用

# Today Topic

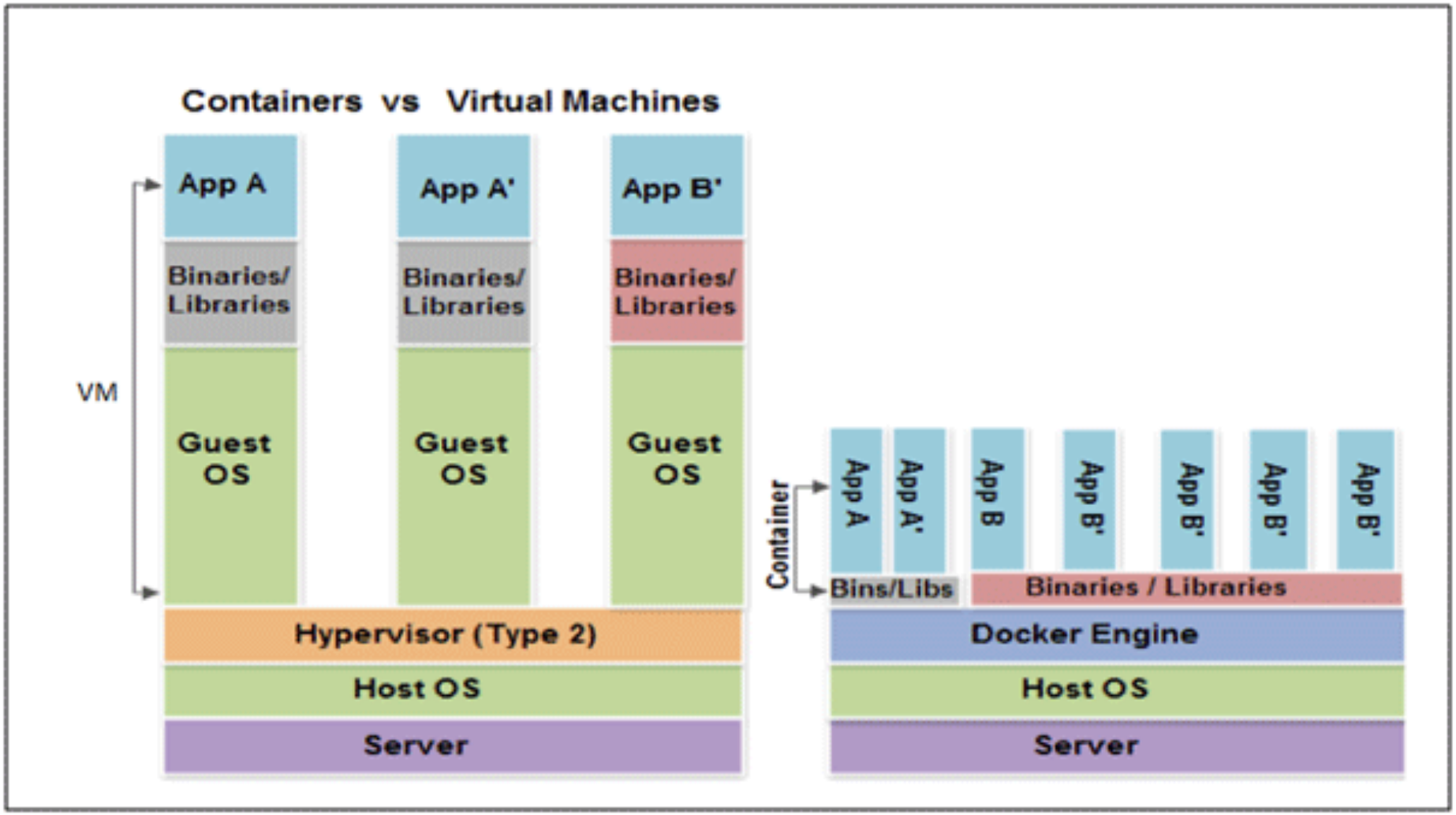
- Docker Review
- Kubernetes簡介
  - 什麼是 Kubernetes
  - 安裝 Kubernetes
  - Kubernetes 基本操作
- Kubernetes Basic Concept
  - 以WordPress + MySQL建立服務為例
- Kubernetes Advanced Topic

# Why Docker

## Shipping code to the server is very hard

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Static website | ? | ? | ? | ? | ? | ? | ? |
| Web frontend | ? | ? | ? | ? | ? | ? | ? |
| Background workers | ? | ? | ? | ? | ? | ? | ? |
| User DB | ? | ? | ? | ? | ? | ? | ? |
| Analytics DB | ? | ? | ? | ? | ? | ? | ? |
| Queue | ? | ? | ? | ? | ? | ? | ? |
| |  |  |  |  |  |  |  |

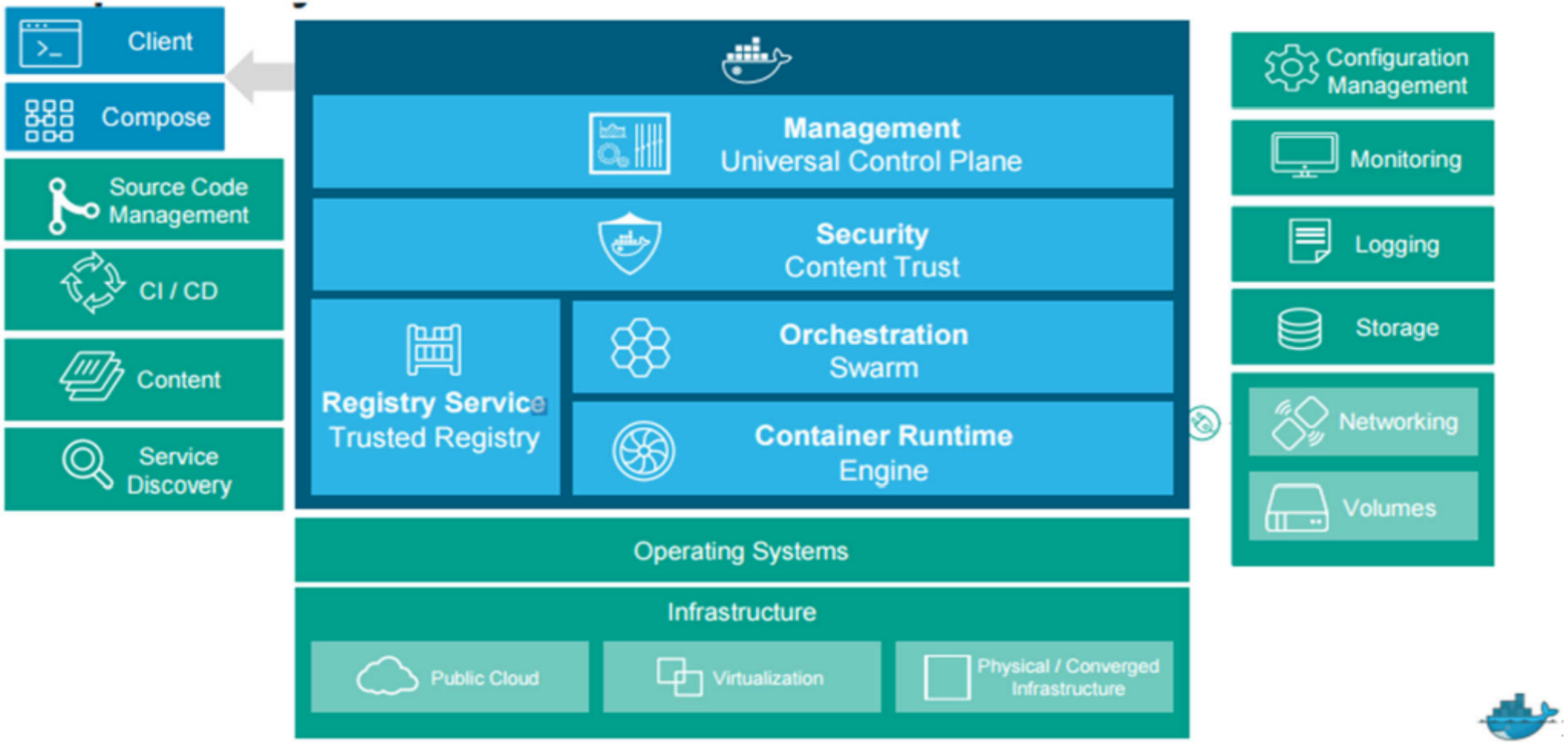
# VM v.s. Container



# How Docker Works

- Linux Kernel support the Namespaces mechanism to partition kernel resource to different process
- PID Namespace
- Network namespace
- Mount Namespace
- …

# Docker Roundup



# Today Topic

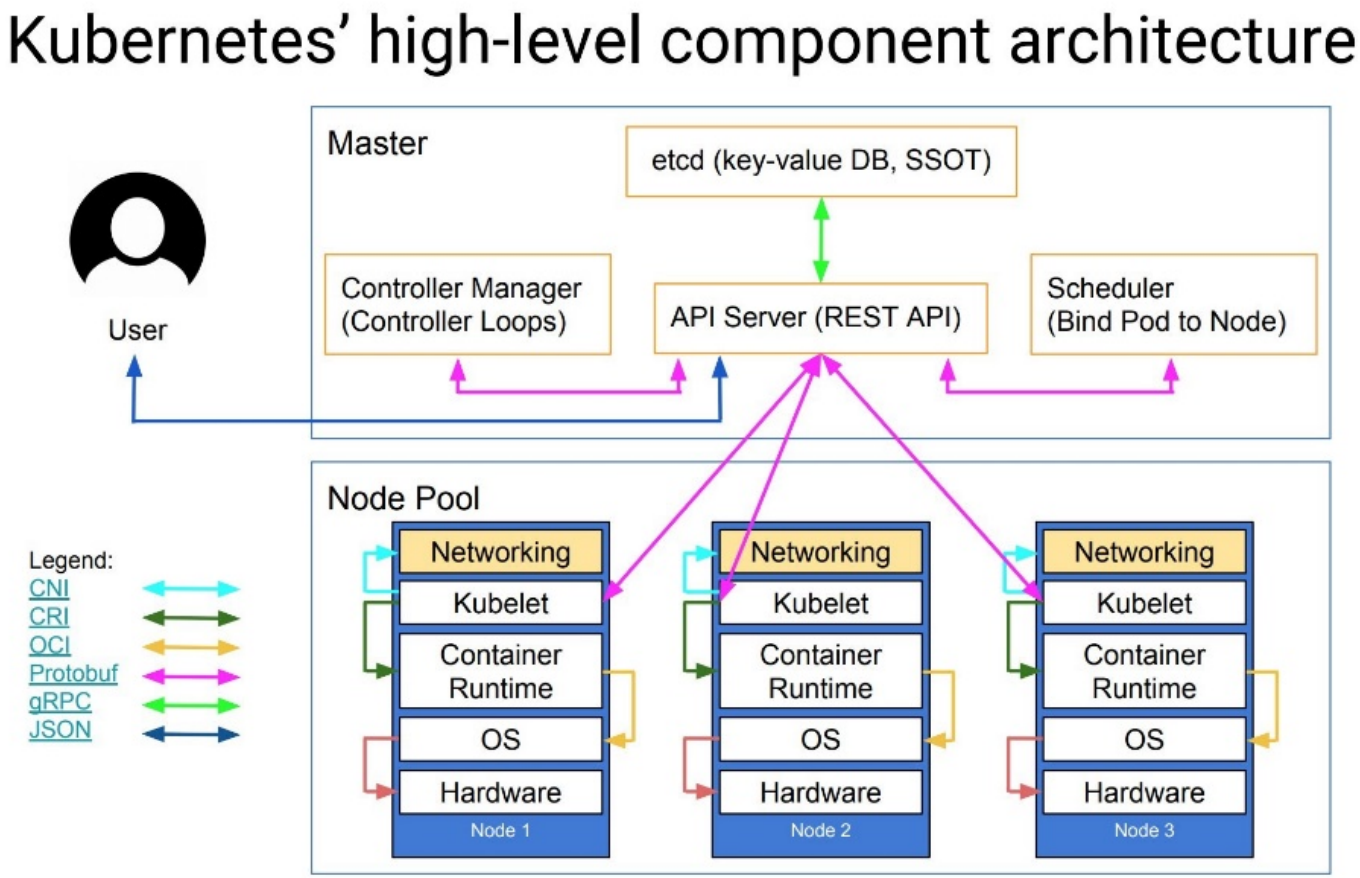
- Docker Review
- Kubernetes簡介
  - 什麼是 Kubernetes
  - 安裝 Kubernetes
  - Kubernetes 基本操作
- Kubernetes Basic Concept
  - 以WordPress + MySQL建立服務為例
- Kubernetes Advanced Topic

# 什麼是Kubernetes

- Kubernetes 是 Google 開源的Container分散式管理系統
- 基於OCI標準容器之上的容器叢集排程服務，簡稱為k8s(‘k’ + 8 letters + 's’)

# Kubernetes 基本架構

- Kubernetes 屬於分散式架構系統，主要由master與多個node組成



# Kubernetes - Master

- kube-apiserver
  - exposes the Kubernetes API
- Etcd
  - used as Kubernetes' backing store
- kube-scheduler
  - watches newly created pods that have no node assigned, and selects a node for them to run on.
- kube-controller-manager
  - A binary that runs controllers, which are the background threads that handle routine tasks in the cluster

# Kubernetes - Node

- Kubelet
  - the primary node agent
- kube-proxy
  - enables the Kubernetes service abstraction by maintaining network rules on the host and performing connection forwarding
- Docker
  - used for actually running containers.
- Flannel or other network plugin
  - give a subnet to each host for use with container runtimes

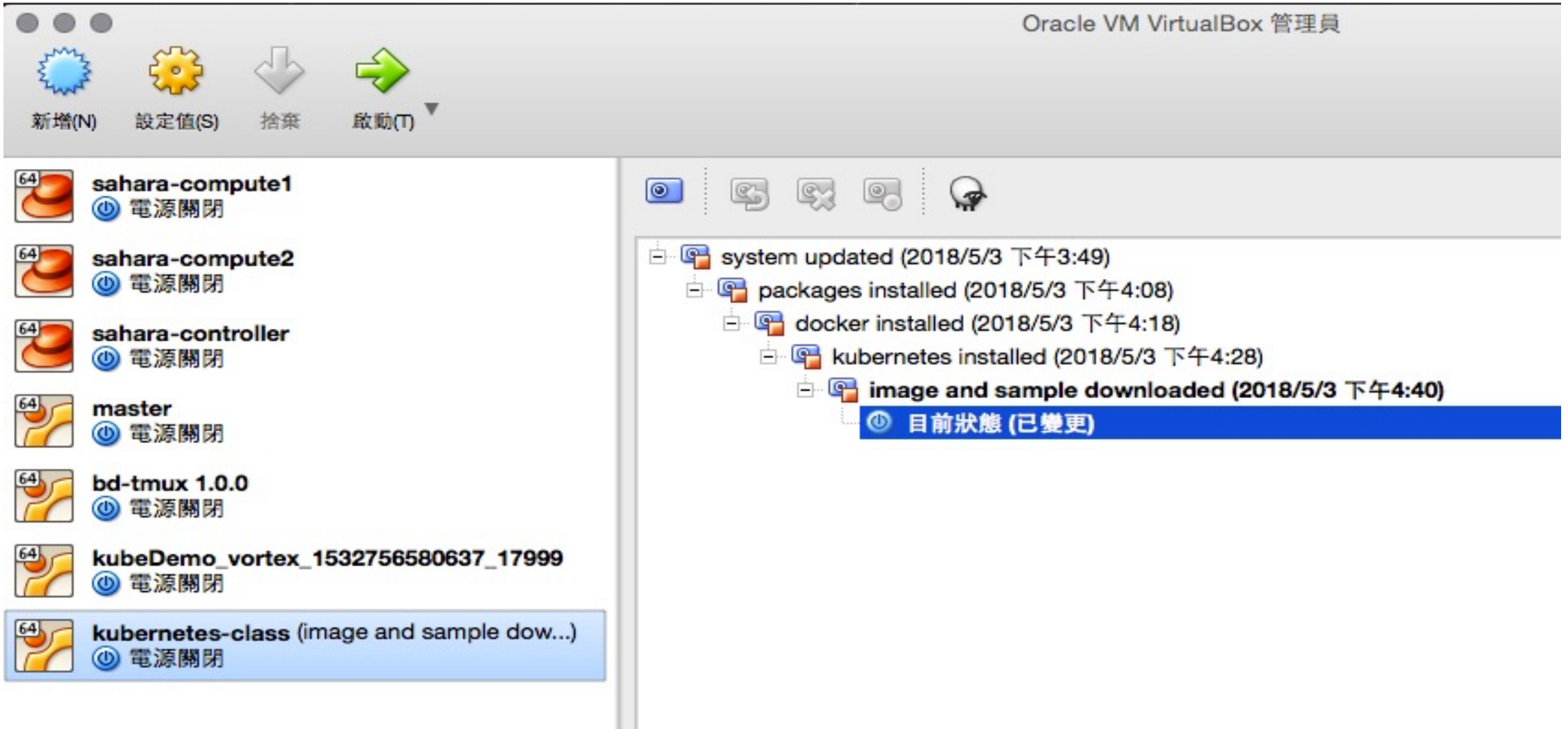
# Kubernetes基本機制

- configuration convergence:
  - 不斷的比較期望的配置和實際的配置，修訂實際配置以收斂到期望配置
- 寫入期望配置:
  - Kubernetes 裡的所有資源對象， Service、Deployment、等等，都是通過 api-server 檢查格式後，序列化並存入 etcd
- 收斂到期望配置:
  - controller 使用api-server 的 watch API 收取 etcd 裡資源的期望配置，和通過 kubelet 收集到的實際配置做對比並修正差異

# Today Topic

- Docker Review
- Kubernetes簡介
  - 什麼是 Kubernetes
  - 安裝 Kubernetes
  - Kubernetes 基本操作
- Kubernetes Basic Concept
  - 以WordPress + MySQL建立服務為例
- Kubernetes Advanced Topic

# 上機操作VM


Oracle VM VirtualBox 管理員
新增(N)
設定值(S)
捨棄
啟動(T)
sahara-compute1
電源關閉
sahara-compute2
電源關閉
sahara-controller
電源關閉
master
電源關閉
bd-tmux 1.0.0
電源關閉
kubeDemo_vortex_1532756580637_17999
電源關閉
kubernetes-class (image and sample dow...)
電源關閉
system updated (2018/5/3 下午3:49)
packages installed (2018/5/3 下午4:08)
docker installed (2018/5/3 下午4:18)
kubernetes installed (2018/5/3 下午4:28)
image and sample downloaded (2018/5/3 下午4:40)
目前狀態 (已變更)

# 安裝 Kubernetes – hard way

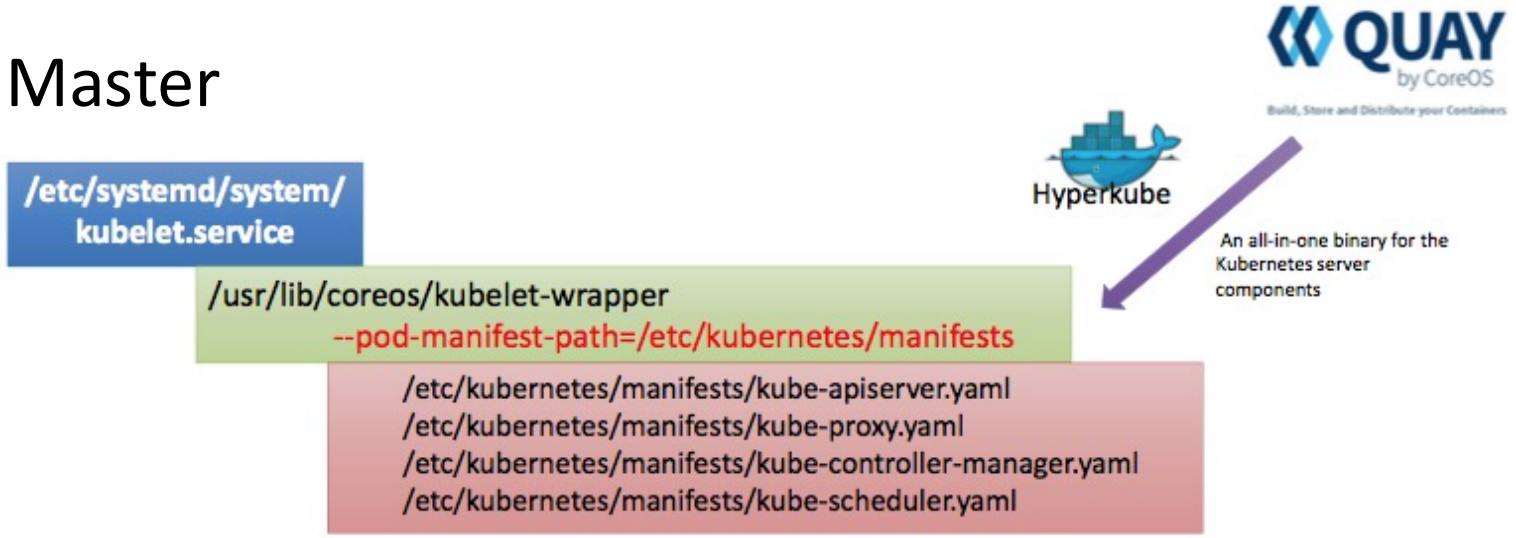
- 手動安裝
  - follow-me-install-kubernetes-cluster
- 步驟
  - Install OS
  - Setup an etcd cluster
  - Generate the certificates for Kubernetes components
  - Deploy master node
  - Deploy worker nodes
  - Configure kubectl to work with cluster
  - Deploy the add-ons
    - DNS
    - Dashboard

# 安裝 Kubernetes – easy way

- 佈署工具
  - Kubeadm
  - kubespray
  - kops
  - kube-aws
  - bootkube
  - kismatic
  - kube-ansible
  - kubeasz
- 每個component 是系統上的一個service
- 每個component 是kubernetes上的一個pod

# How kubeadm Initializes Your Kubernetes Master

- Master




- Node

# Today Topic

- Docker Review
- Kubernetes簡介
  - 什麼是 Kubernetes
  - 安裝 Kubernetes
  - Kubernetes 基本操作
- Kubernetes Basic Concept
  - 以WordPress + MySQL建立服務為例
- Kubernetes Advanced Topic

# Kubernetes Objects

- Kubernetes透過各種的objects來表示整個cluster的狀態
- 透過kubernetes api來建立、修改、刪除各種object
  - Kubectl: 一種CLI的api
  - 亦可使用library，用程式的方式來操作

| 類型 | 名稱 |
| --- | --- |
| resource object | **Pod**、ReplicaSet、**ReplicationController**、**Deployment**、StatefulSet、DaemonSet、Job、CronJob、HorizontalPodAutoscaling |
| Config object | Node、Namespace、**Service**、**Secret**、**ConfigMap**、**Ingress**、**Label**、**CustomResourceDefinition**、 ServiceAccount |
| Storage object | **Volume**、**Persistent Volume** |
| Policy Object | SecurityContext、 ResourceQuota、 LimitRange |

# 透過yaml or json表示 kubernetes object

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        imagePullPolicy: IfNotPresent
```

- 在yaml檔案裡描述要建立的object
  - apiVersion
    - object所使用的 Kubernetes API 的版本
  - Kind
    - Object的類型
  - metadata
    - 識別object唯一性的資訊，常用的為name
  - Spec:
    - 用來描述object的狀態
    - 不同的object有不同的spec定義

# 操作kubernetes - kubectl

- 透過kubectl操控所有的kubernetes objects
  - deployments / deployment / deploy
  - services / service / svc
  - pods / pod / po
  - daemonsets /daemonset / ds
  - …
- 建立objects
  - kubectl **create** –f /path/to/file
  - kubectl **apply** –f /path/to/file
- 刪除objects
  - Kubectl **delete** –f /path/to/file
  - Kubectl **delete** TYPE NAME

- 查看objects
  - kubectl **get** TYPE
  - kubectl **describe** TYPE NAME
  - Kubectl **logs** POD_NAME

- 更新object
  - Kubectl **edit** TYPE　NAME
  - kubectl **replace** -f /path/to/newfile

- 執行pod裡面的指令
  - kubectl **exec** -ti POD_NAME -- CMD

- 進行port-forward
  - Kubectl **port-forward** POD_NAME LOCAL:REMOTE

# KUBECONFIG

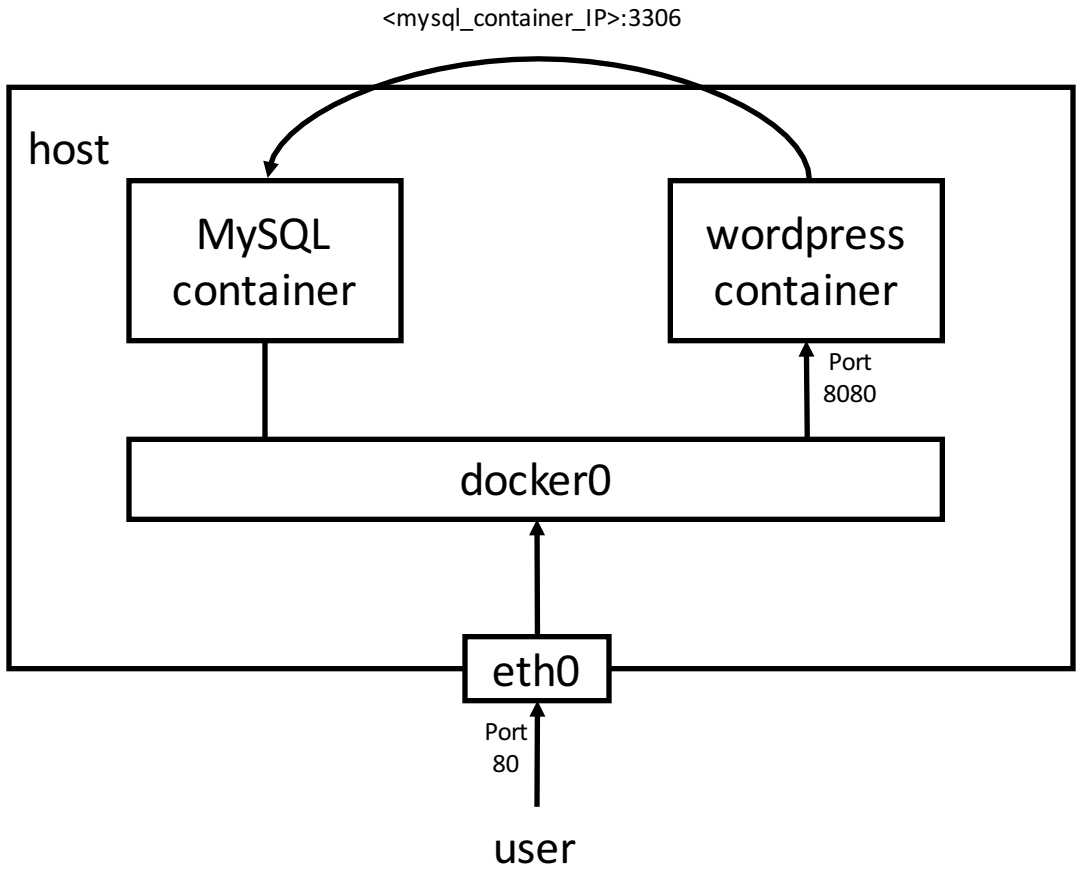
- 透過KUBECONFIG，設定要採用的context，
- Context = 以user身份 操作 cluster 上的 某個 namespace
- 透過RBAC設定user的細部權限

```
root@ubuntu:/etc/kubernetes# kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://10.0.2.5:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

# Today Topic

- Docker Review
- Kubernetes簡介
  - 什麼是 Kubernetes
  - 安裝 Kubernetes
  - Kubernetes 基本操作
- Kubernetes Basic Concept
  - 以WordPress + MySQL建立服務為例
- Kubernetes Advanced Topic

# Wordpress + MySQL的例子



# 用docker部署wordpress & mysql

- 缺點：
  - Wordpress 和 mysql要分別部署
  - 無法自動scale out
  - 必需手動升級與降級
  - Crash必需手動重啟
  - 無法做到 Service discovery
  - 無法做到 load balancing
- 試著用Kubernetes來解決

# Kubernetes Basic Concept

- Pod
- Volume & Persistence volume & Storage Class
- Label & Annotation & Selector
- Replication Controller
- Deployment
- Service
- Secret
- configMap
- Ingress

# Kubernetes - Pod

- Pod為 Kubernetes 中最小的部署單位 Pod
  - 由一至多個容器所組成
  - 通常將耦合性較高之container放置在同一個pod
  - 同一個pod的container會在同一個node
  - 即使node掛掉，Pod還是被排到同一個node
- 共享同樣的 network & Port space
  - Pod內的container可以用localhost溝通
  - 使用pause container設定共用網路
- 共享儲存空間 (volumes)
  - 在Pod定義volume，container可以掛載這些定義好的volume

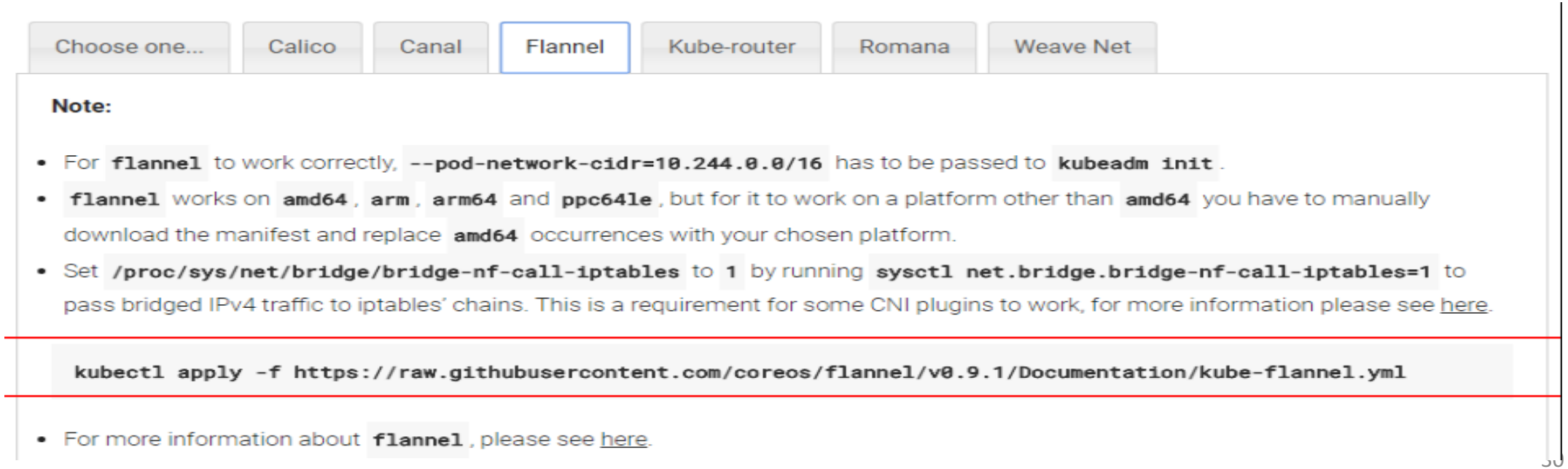
# Pod Life cycle

- Once scheduled to a node, pods do not move
  - Restart policy means restart in the same node
  - Pods are not rescheduled even if a node dies
- Pods phase
  - Pending
  - Running
  - Succeeded
  - Failed
  - Unknown

# Pod Networking

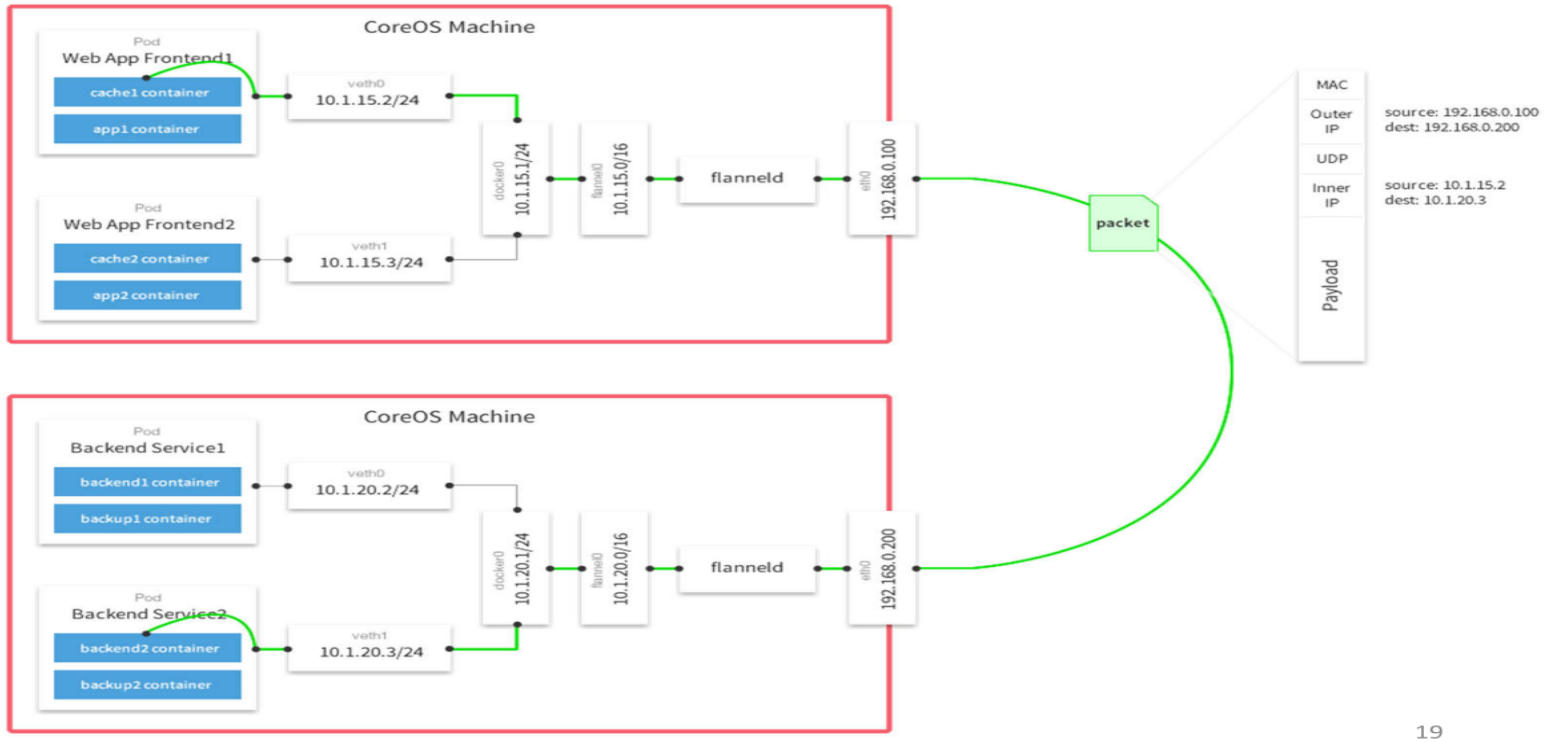
- Container network interface (CNI)
- 3rd party CNI

```
# Installing a pod network
## flannel
$ kubectl apply --namespace kube-system -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
```



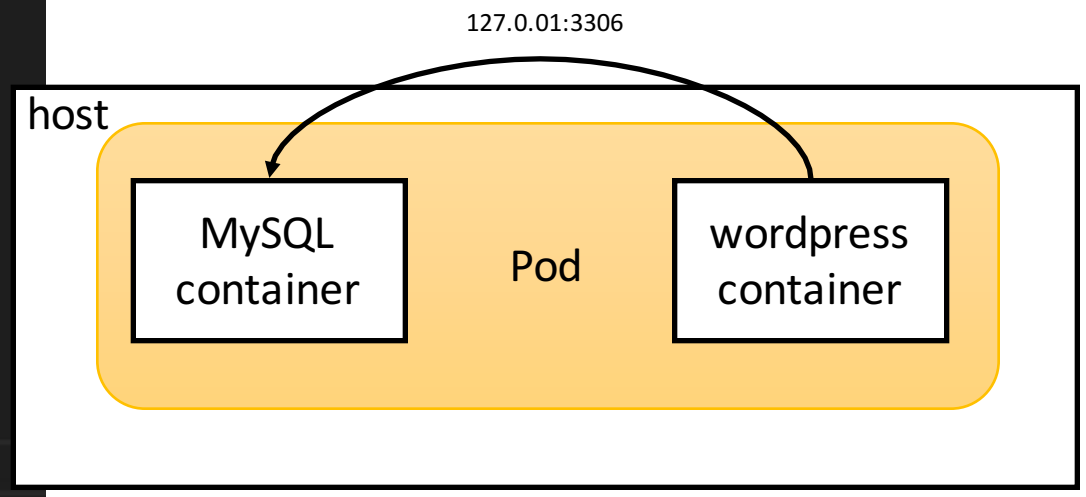
# Flannel

- Easy deploy,
  - only one yaml file, flaneeld run as a Pod in every host
- Provide Layer 2 overlay network
- Two way to encapsulate packet: UDP and VXLAN



# Wordpress + MySQL的例子

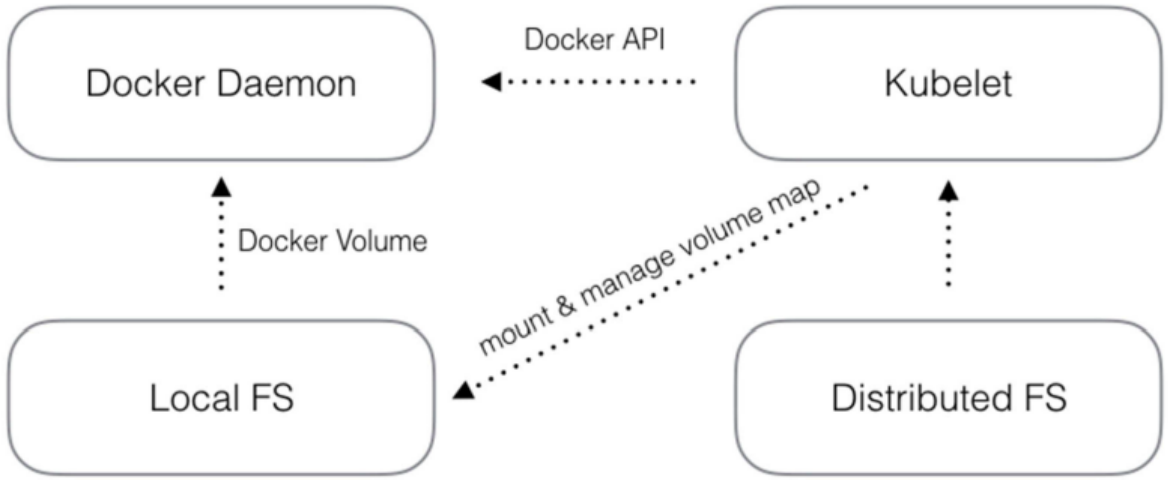
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-wordpress-pod
spec:
  containers:
  - name: mysql
    image: mysql:5.6
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: Password1234
  - name: wordpress
    image: wordpress:4.8-apache
    env:
    - name: WORDPRESS_DB_HOST
      value: 127.0.0.1
    - name: WORDPRESS_DB_PASSWORD
      value: Password1234
```



# Volume

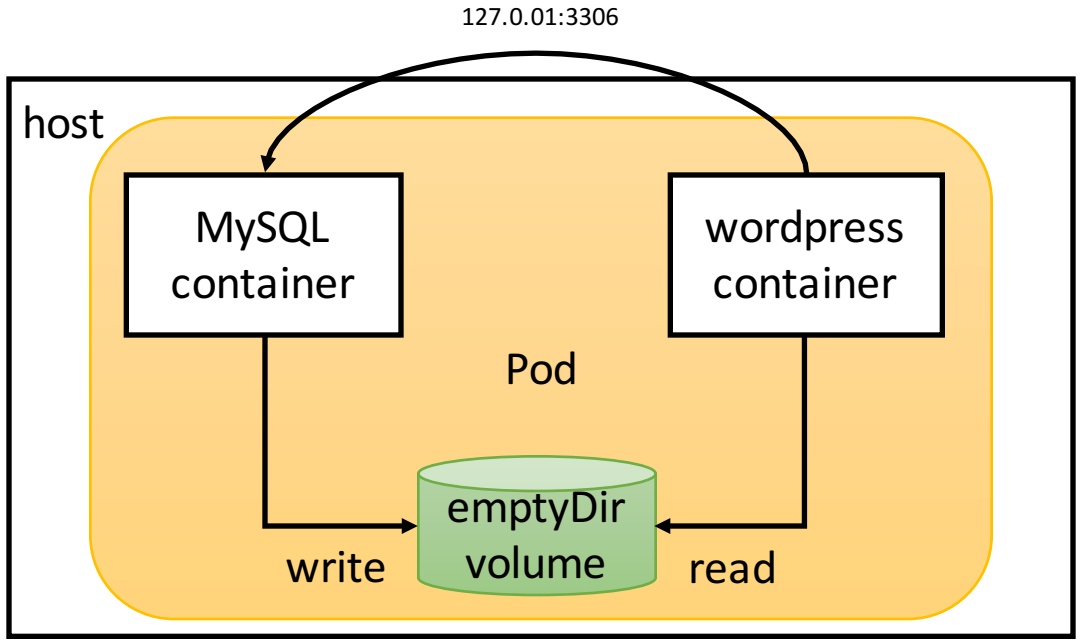
- Container裡的檔案是ephemeral
- Pod裡的container，可以共用volume
- Volume週期和pod一樣
- Kubernetes has different volume plugin

| Temp | Local | Network |
| --- | --- | --- |
| • emptyDir | • hostPath | • GlusterFS<br>• CephRBD<br>• gitRepo<br>• secret<br>• flocker<br>• gcePersistentDisk<br>• AWS ElasticBlockStore (EBS)<br>• NFS<br>• iSCSI<br>• Fibre Channel<br>• Cinder |



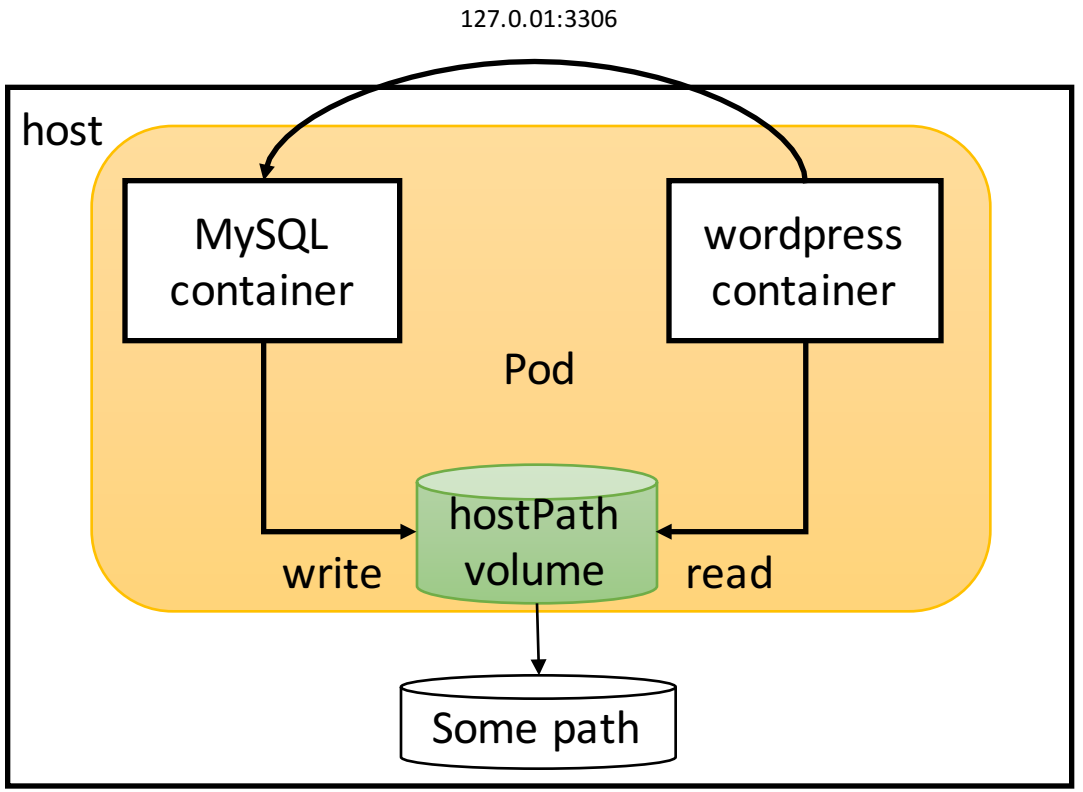
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-wordpress-pod
spec:
  containers:
  - name: mysql
    image: mysql:5.6
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: Password1234
    volumeMounts:
    - name: mysql-storage
      mountPath: /var/lib/mysql
  - name: wordpress
    image: wordpress:4.8-apache
    env:
    - name: WORDPRESS_DB_HOST
      value: 127.0.0.1
    - name: WORDPRESS_DB_PASSWORD
      value: Password1234
    volumeMounts:
    - name: mysql-storage
      mountPath: /var/lib/wordpress
  volumes:
  - name: mysql-storage
    emptyDir: {}
```

- 使用emptyDir
  - 生命週期和pod相同
    - 當pod被移除時，emptyDir也會被清空
    - 但Container 重啟，資料仍會保留
  - 同一個pod的container間，可以共享資料



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-wordpress-pod
spec:
  containers:
  - name: mysql
    image: mysql:5.6
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: Password1234
    volumeMounts:
    - name: mysql-storage
      mountPath: /var/lib/mysql
  - name: wordpress
    image: wordpress:4.8-apache
    env:
    - name: WORDPRESS_DB_HOST
      value: 127.0.0.1
    - name: WORDPRESS_DB_PASSWORD
      value: Password1234
    volumeMounts:
    - name: mysql-storage
      mountPath: /var/lib/wordpress
  volumes:
  - name: mysql-storage
    hostPath:
      path: /tmp/data
```

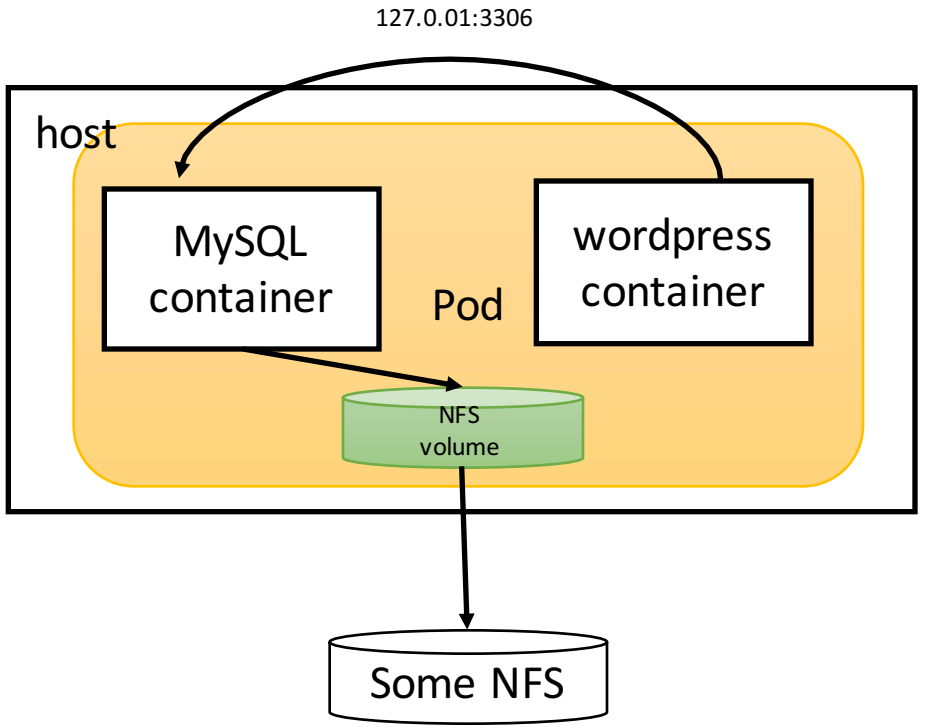
- 使用hostPath
  - 將host上的某個目錄掛載至Pod
  - 資料會保存在host上，重啟pod不會清空資料
  - 但container 移動到別的host上，資料遺失



# Wordpress + MySQL的例子

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-wordpress-pod
spec:
  containers:
  - name: mysql
    image: mysql:5.6
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: Password1234
    volumeMounts:
    - name: mysql-storage
      mountPath: /var/lib/mysql
  - name: wordpress
    image: wordpress:4.8-apache
    env:
    - name: WORDPRESS_DB_HOST
      value: 127.0.0.1
    - name: WORDPRESS_DB_PASSWORD
      value: Password1234
  volumes:
  - name: mysql-volumes
    nfs:
      server: 192.168.2.31
      path: /nfs-data
```

- 使用nfs



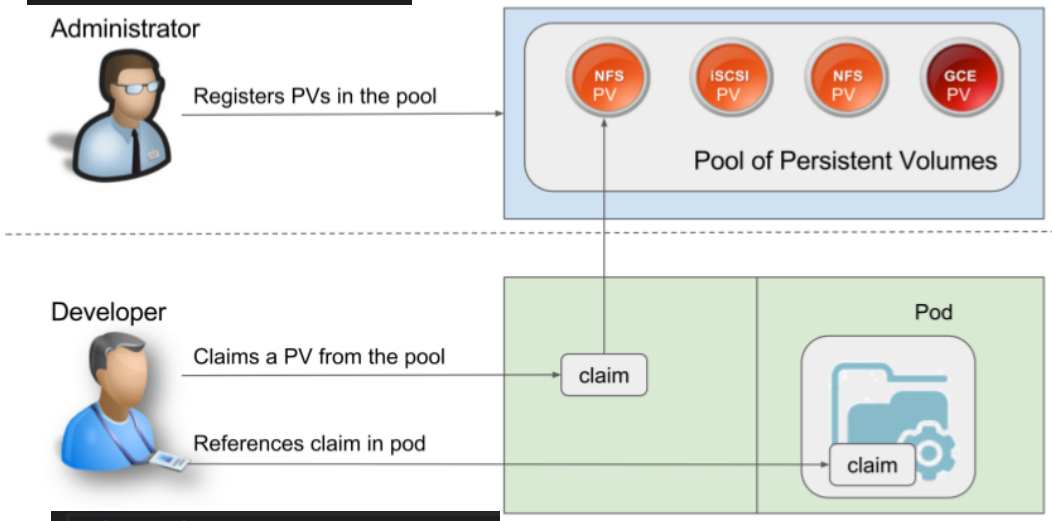
# Persistence Volume (PV) & Persistence Volume Claim(PVC)

- Abstraction of Physical storage
  - Separate **Dev**lopment & **Op**erator

- PersistentVolume (PV)
  - A networked storage provisioned **by an administrator**
  - Operator view

- PersistentVolumeClaim (PVC)
  - Request for storage **from a user**
  - Development view

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.2.31
    path: /nfs-data
```




```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1Gi
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-wordpress-pod
spec:
  containers:
  - name: mysql
    image: mysql:5.6
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: Password1234
    volumeMounts:
    - name: mysql-volumes
      mountPath: /var/lib/mysql
  - name: wordpress
    image: wordpress:4.8-apache
    env:
    - name: WORDPRESS_DB_HOST
      value: 127.0.0.1
    - name: WORDPRESS_DB_PASSWORD
      value: Password1234
  volumes:
  - name: mysql-volumes
    persistentVolumeClaim:
      claimName: nfs-pvc
```

# Storage Class(SC)

- Before User request a PVC, administrator need to create PV in advanced.
  - Static Provisioning

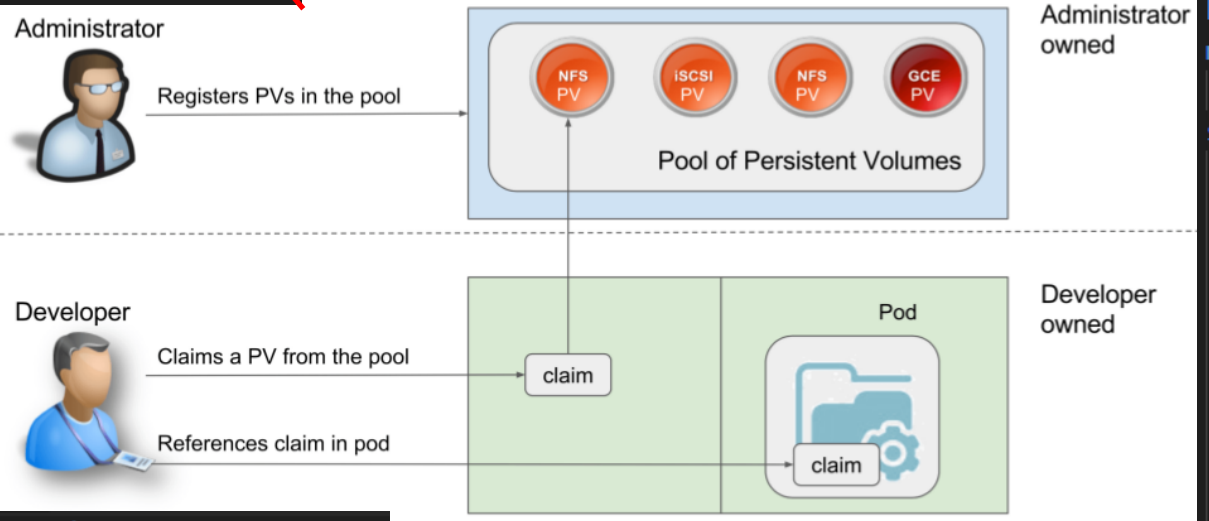
- If storage volumes can be created on-demand
  - Dynamic Provisioning
  - use storage class to enable Dynamic Provisioning
  - Provisioner is required for provisioning PVs
  - Specify SC in PVC
    - If SC is not specified in PVC, default SC is used
    - If there is no match SC, PVC is pending

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.2.31
    path: /nfs-data
```

Create PVs on-demand

Storage Class

provisioner




```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1Gi
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-wordpress-pod
spec:
  containers:
  - name: mysql
    image: mysql:5.6
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: Password1234
    volumeMounts:
    - name: mysql-volumes
      mountPath: /var/lib/mysql
  - name: wordpress
    image: wordpress:4.8-apache
    env:
    - name: WORDPRESS_DB_HOST
      value: 127.0.0.1
    - name: WORDPRESS_DB_PASSWORD
      value: Password1234
  volumes:
  - name: mysql-volumes
    persistentVolumeClaim:
      claimName: nfs-pvc
```

# Summary

- Pod透過volume持久化資料
- 透過pvc，提供volume的抽象概念，使用者不需要了解底層實作
- PVC需綁定至底層的一個PV, 且PV需事先建立
- 透過default SC, 可以自動產生PV

# Label & Annotations

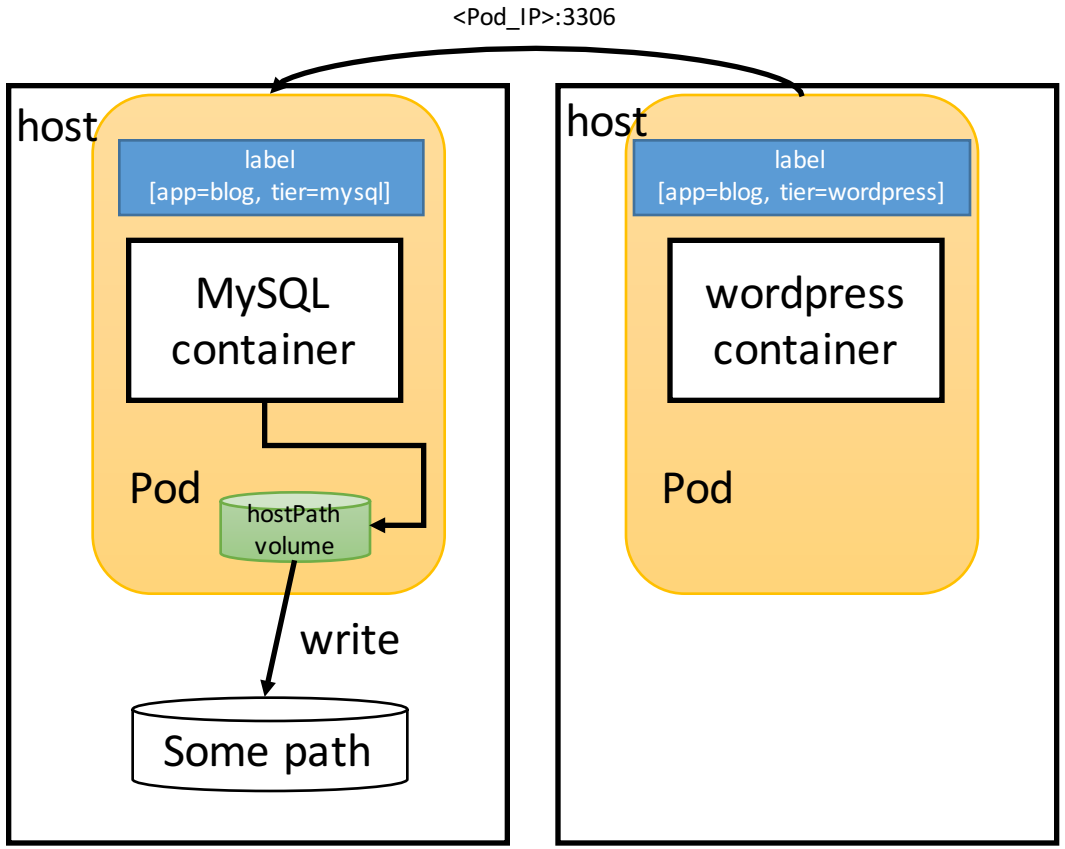
```
"environment" : "dev", "environment" : "qa", "environment" : "production"

"tier" : "frontend", "tier" : "backend", "tier" : "cache"

"partition" : "customerA", "partition" : "customerB"
```

- 相同點
  - 一對 key/value pair
  - 每個物件可以同時擁有許多個labels (or annotation)
  - 皆位於 object 的 metadata 欄位內

- 不同點
  - **Label** 可以透過selector進行篩選
  - annotation做為沒有識別用途的標籤
  - 可以讀取annotation的值
    - Eg: prometheus.io/path: “/metrics”

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-pod
  labels:
    app: blog
    tier: mysql
  annotations:
    compnay: 'NCHC'
spec:
  containers:
  - name: mysql
    image: mysql:5.6
```

# Wordpress + MySQL的例子



# Label的應用：讓Pod運行在特定節點上

- 在Pod Definition裡使用NodeSelector
- 在node上打上label

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
  nodeSelector:
    hardware: high-memory
```

# Selector

- Selector 透過label來查詢object
- 讓service/controller知道要處理那些Pod

- 目前有二類selector
  - Equality-based
  - Set-based requirements

```
environment = production
tier != frontend
```

```
environment in (production, qa)
tier notin (frontend, backend)
partition
!partition
```

- 若同時有多個selector，則要選擇全部都滿足的pod(即做and運算)

- For new resource:
  - Job、Deployment、Replica Set、Daemon Set

```
selector:
  matchLabels:
    app: blog
    tier: mysql
```

- For old resource:
  - Service、Replication Controller

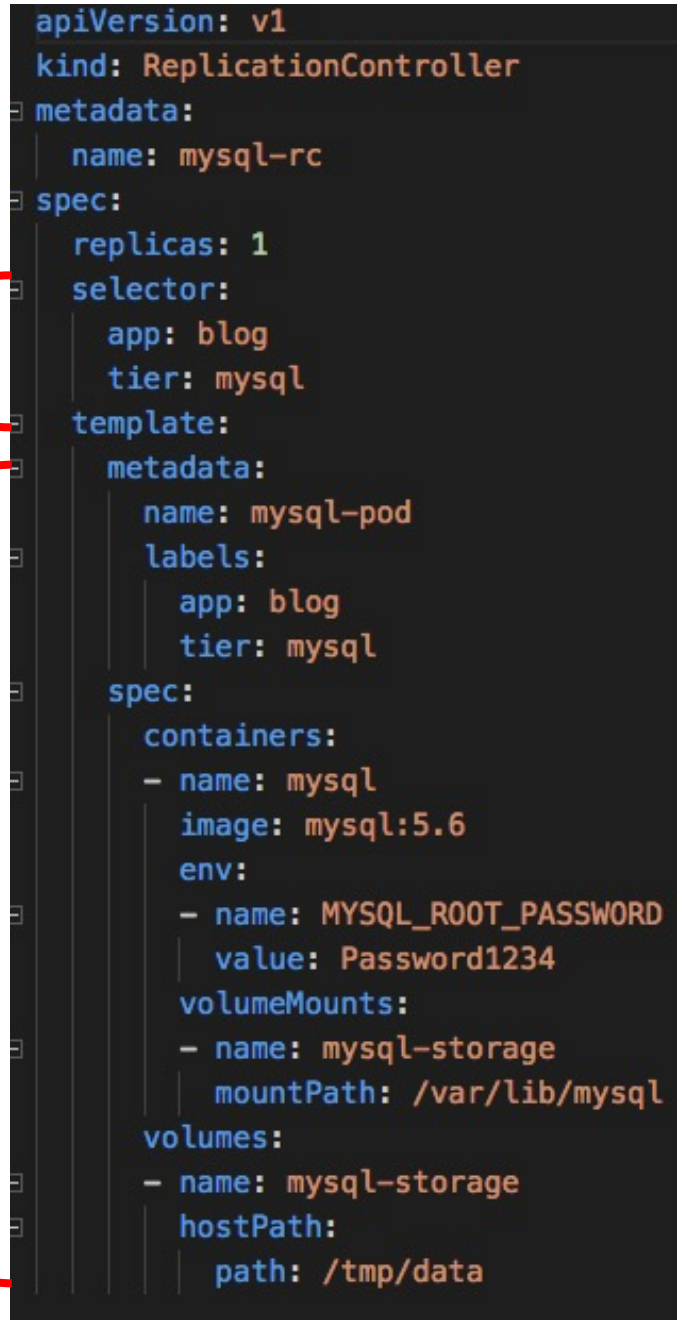
```
selector:
  app: blog
  tier: mysql
```

# Replication Controller

- 在Kubernetes中通常是透過Controller來管理Pod, 很少直接使用Pod
  - Controller的作用是讓resource維持在想要的狀態
  - Replication Controller, Replication Set, Deployment是常用的controller
- Replication Controller是用來管理Pod的數量，確保運行中的Pod數量與設定的數量相同
- 只支援 equality-based selector requirements
- 主要的功能為rolling update 和scaling

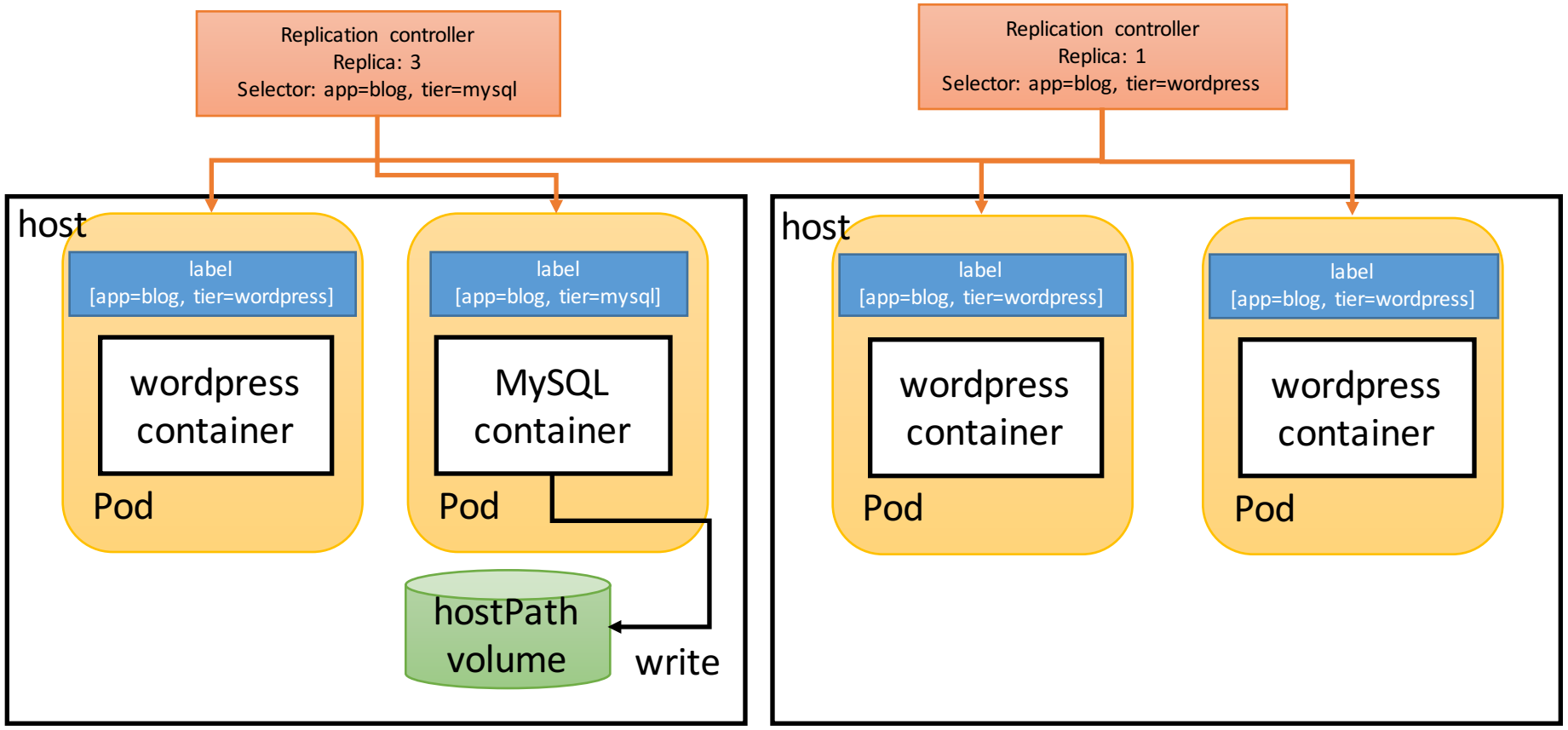
Selector

Pod Definition



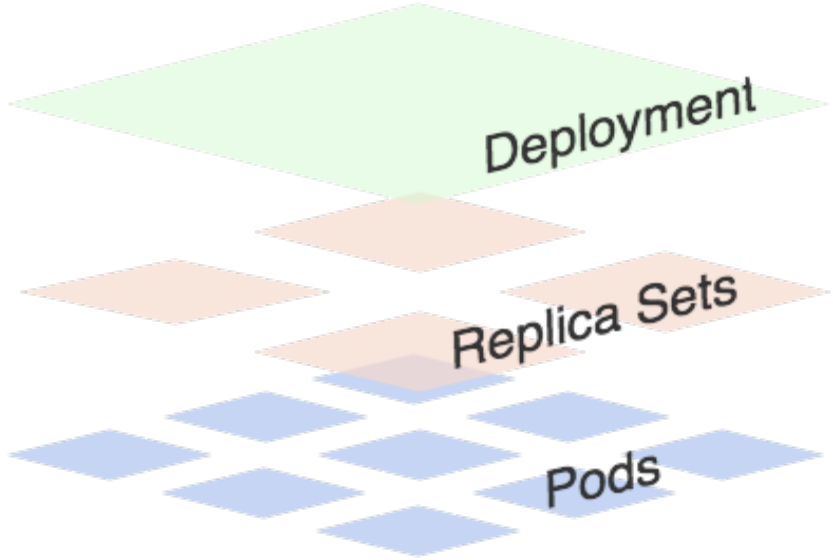
```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: mysql-rc
spec:
  replicas: 1
  selector:
    app: blog
    tier: mysql
  template:
    metadata:
      name: mysql-pod
      labels:
        app: blog
        tier: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:5.6
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: Password1234
        volumeMounts:
        - name: mysql-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-storage
        hostPath:
          path: /tmp/data
```

# Wordpress + MySQL的例子



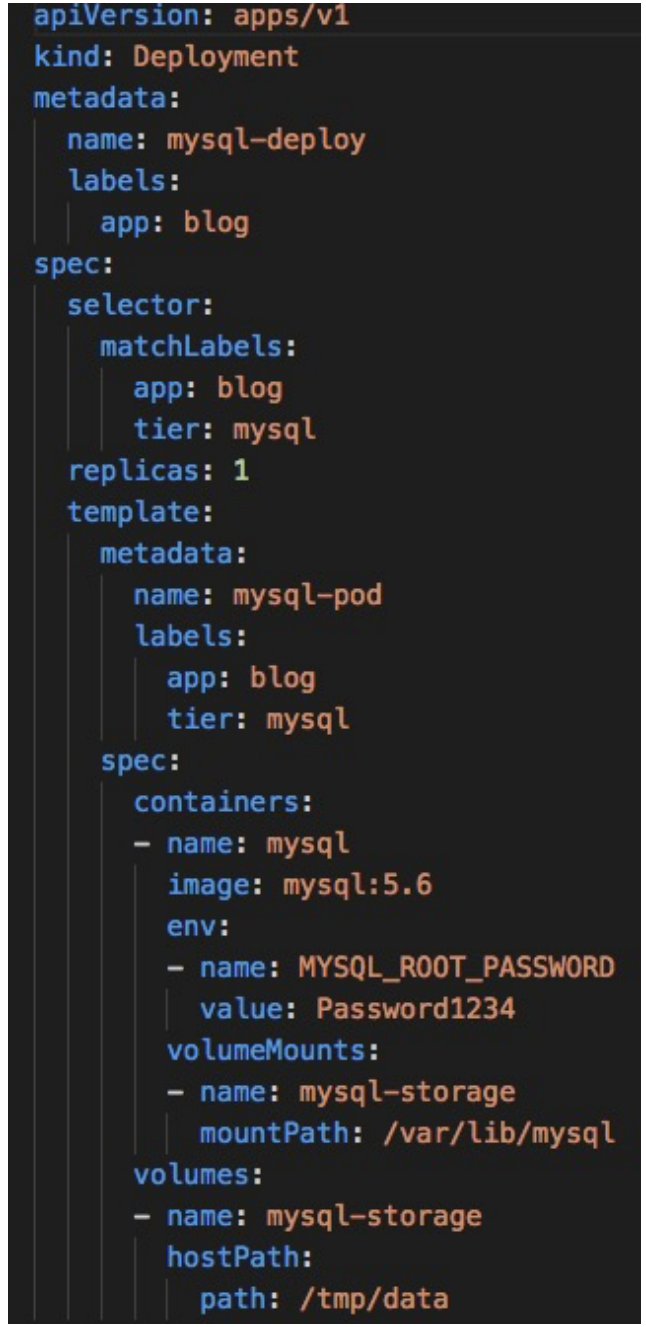
# Deployments

- Deployment 的功用
  - 確保 Pod 數量(replicas)滿足所設定的值
  - 變動Pod的數量(Scale)
  - 滾動升級(Rolling update)
  - 回滾(Roll back)的機制
- Deployment有多個replica set
  - 一個replica set控制一組pod的複本
- 官方推薦用 Deployment 取代 Replication Controller
  - 支援Equality-based & Set-based  requirements



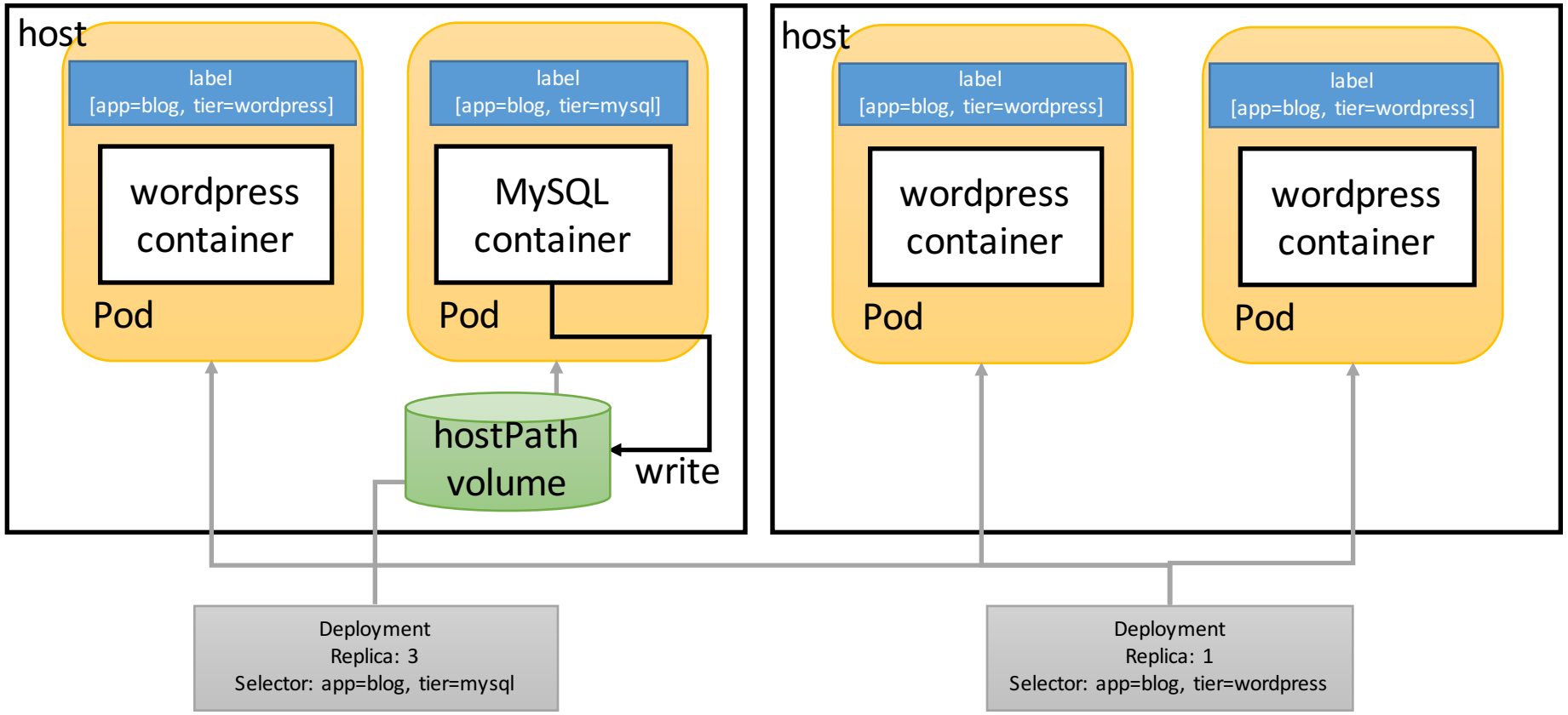
Selector

Pod Definition



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deploy
  labels:
    app: blog
spec:
  selector:
    matchLabels:
      app: blog
      tier: mysql
  replicas: 1
  template:
    metadata:
      name: mysql-pod
      labels:
        app: blog
        tier: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:5.6
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: Password1234
        volumeMounts:
        - name: mysql-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-storage
        hostPath:
          path: /tmp/data
```

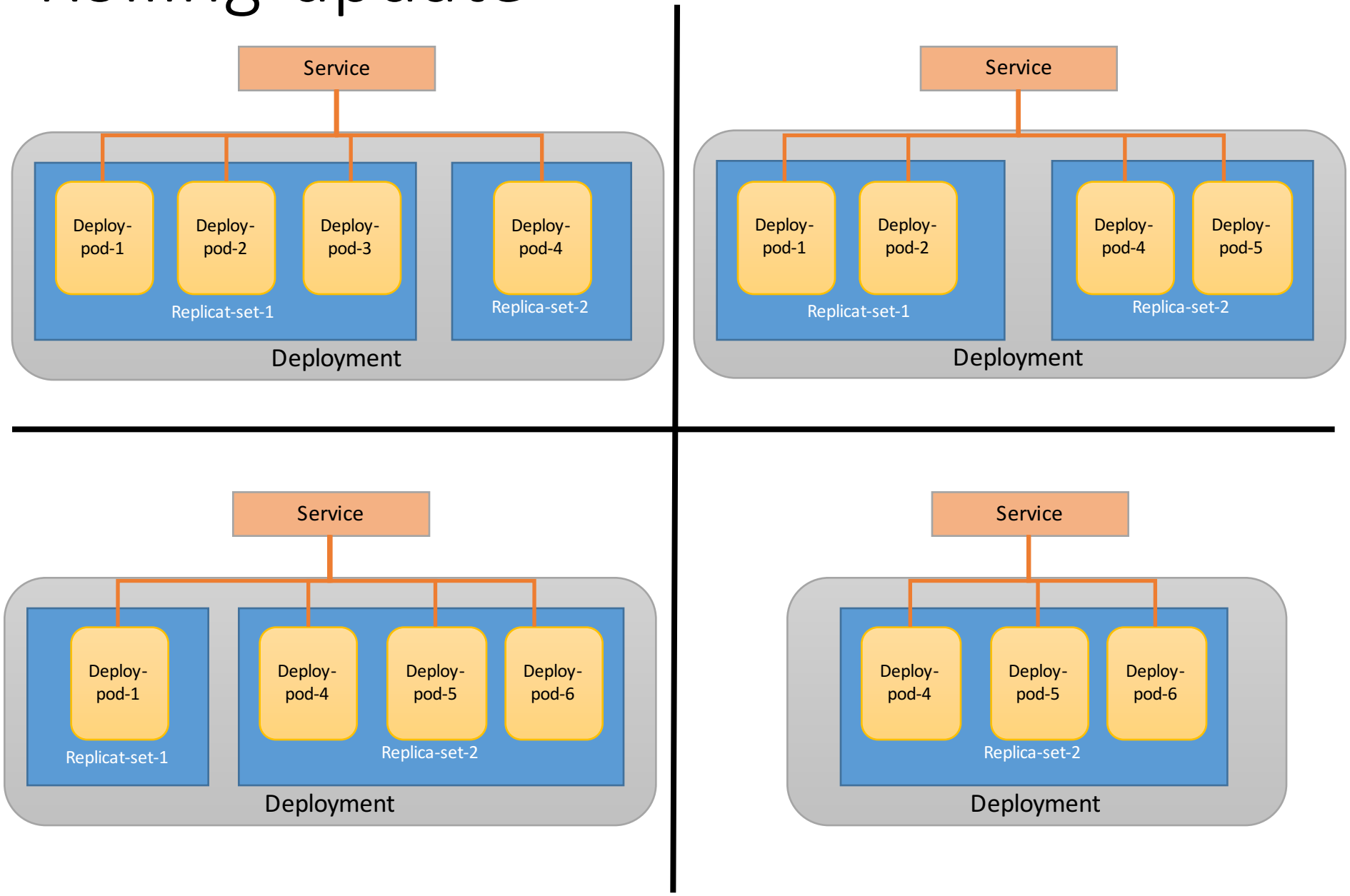
# Wordpress + MySQL的例子



# Deployment 的主要功能

- Scaling a Deployment
  - 擴展pod的數量
- Rolling update (zero down time update)
  - 可以達到無停機服務遷移
- Roll back
  - 當update 完後發現此次的升級造成服務發生不穩定的狀況，可以利用 rollback 來回復到先前的狀態。

# Rolling-update



# 常用 Deployment 指令

| Deployment相關指令 | 指令功能 |
| --- | --- |
| kubectl **scale** deploy  deploy <deployment>  --replicas=n | Scale deployment 物件的pod數 |
| kubectl **edit** deploy <deployment> | 編輯特定deployment物件 |
| kubectl **set image** deployment <deployment>  <container>=<image> | 將deployment管理的pod升級到特定image版本 |
| kubectl **rollout status** deploy <deployment> | 查詢目前某deployment升級狀況 |
| kubectl **rollout history** deploy <deployment> | 查詢目前某deployment升級的歷史紀錄 |
| kubectl **rollout undo** deploy <deployment>  --to-revision=n | 回滾Pod到某個特定版本 |
| kubectl **rollout undo** deploy <deployment> | 回滾Pod到先前一個版本 |

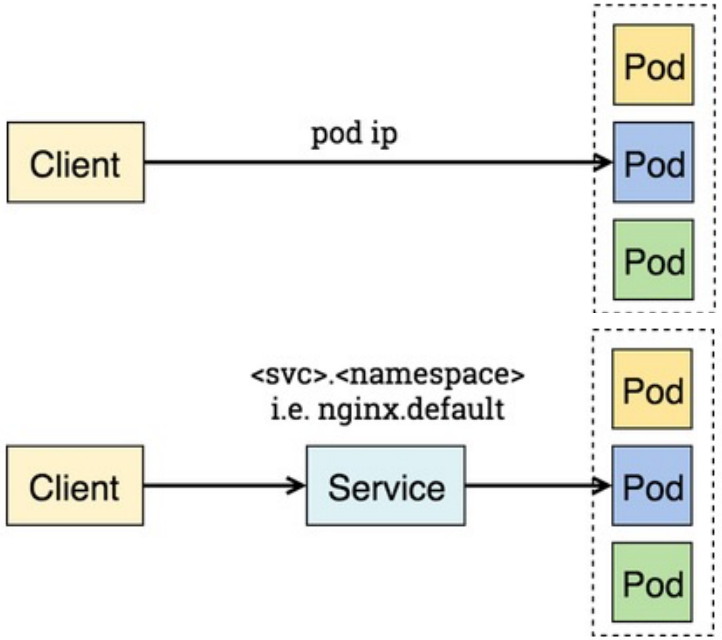
scale

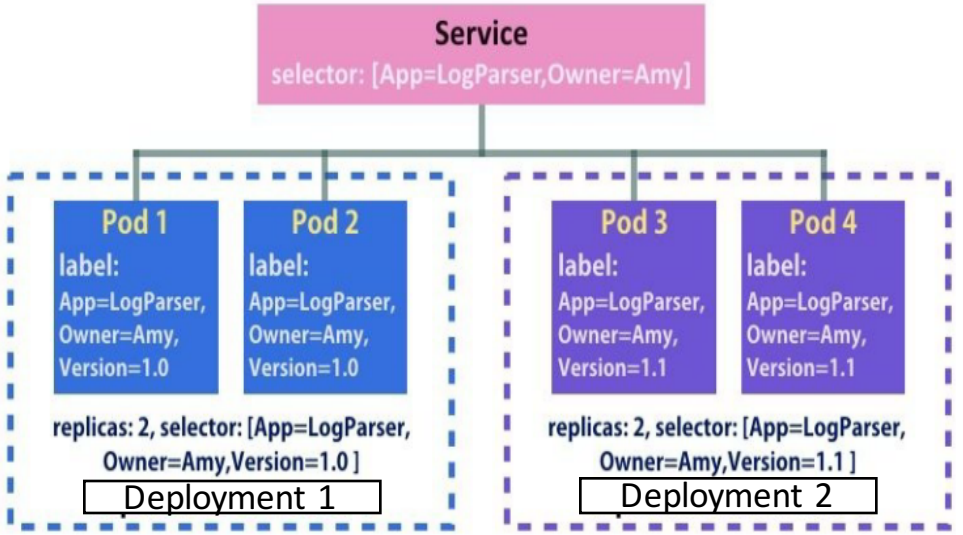
Rolling update

Roll back

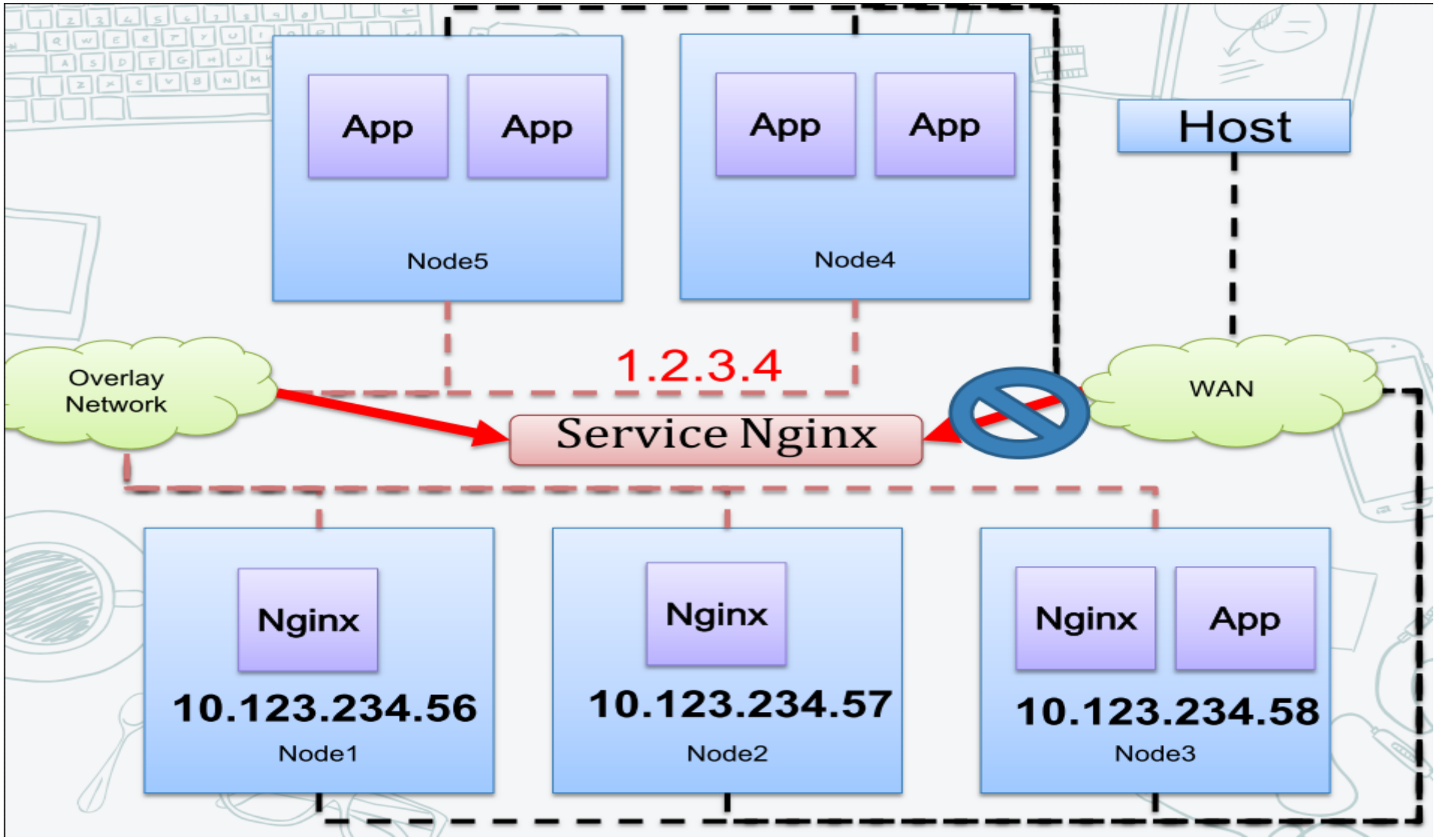
# Service

- Pod的數量與 IP 會變動， 不應該透過Pod IP進行存取
- 一群Pod指派給 service， 透過Service的cluster IP， service會將至cluster IP的流量導入不同的pod.
- 透過Label與selector決定service與pod的對應
- 可以透過svc.namespace來存取， 而不透過IP



- Cluster IP是一個virtual ip，只有kubernetes裡的object才能存取



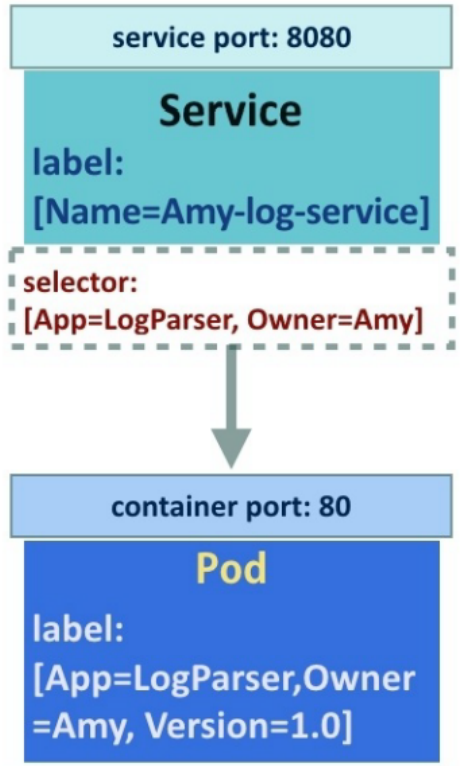
- 用expose建立Service
  - 可以用在pod、svc、rs、rc、deploy

```
$ kubectl expose pod <POD_NAME> --labels="Name=Amy-log-service"
--selector="App=LogParser,Owner=Amy"  --port=8080 --target-port=80
--name="my-service"
```

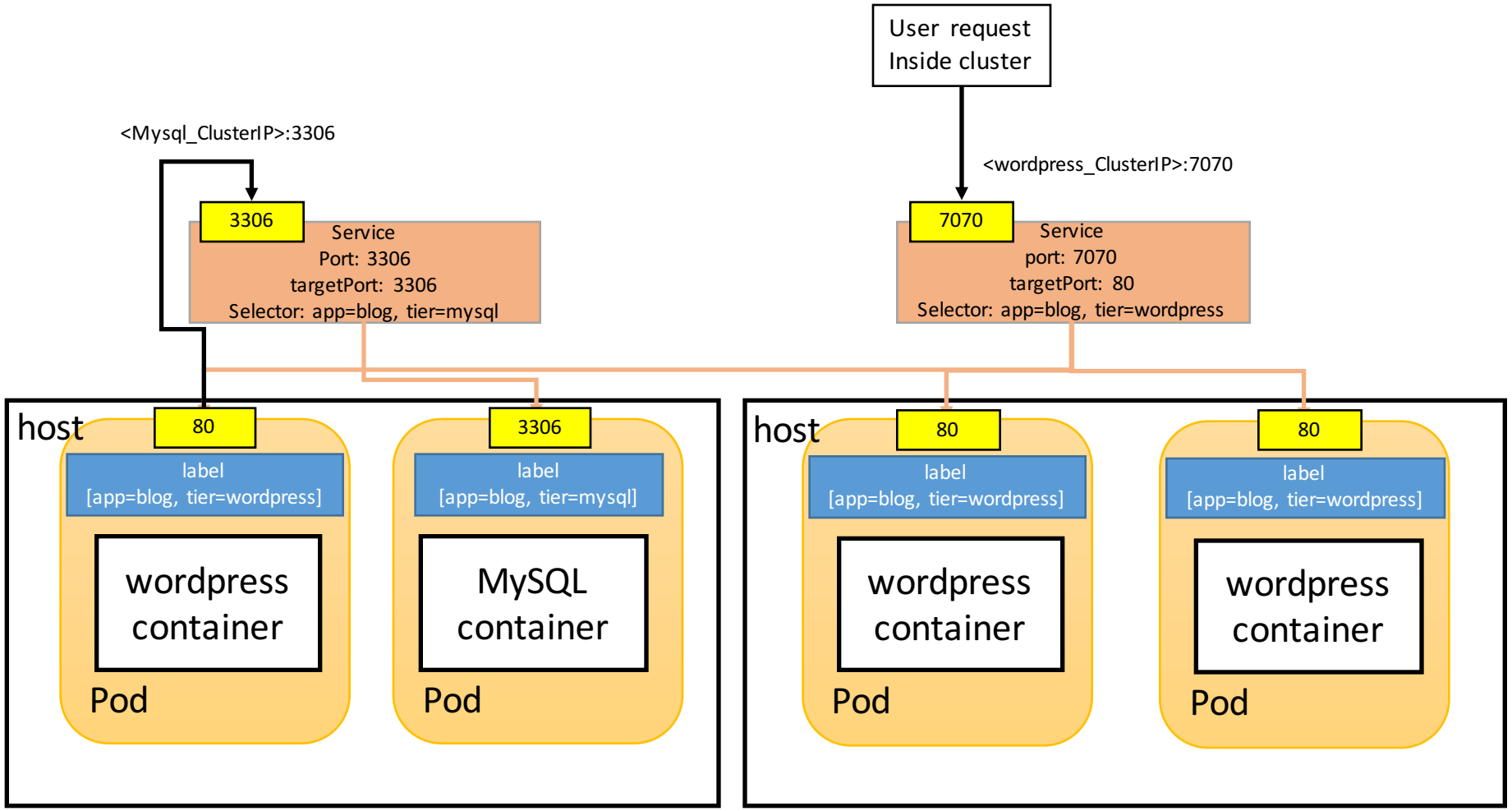
```
$ kubectl expose deploy nginx-deploy  --name="service-deploy"  --port 80
```

- 用YAML檔描述Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
  label:
    Name: Amy-log-service
selector:
  App: LogParser
  Owner: Amy
spec:
  type: ClusterIP
  ports:
  - port: 8080
    targetPort: 80
```

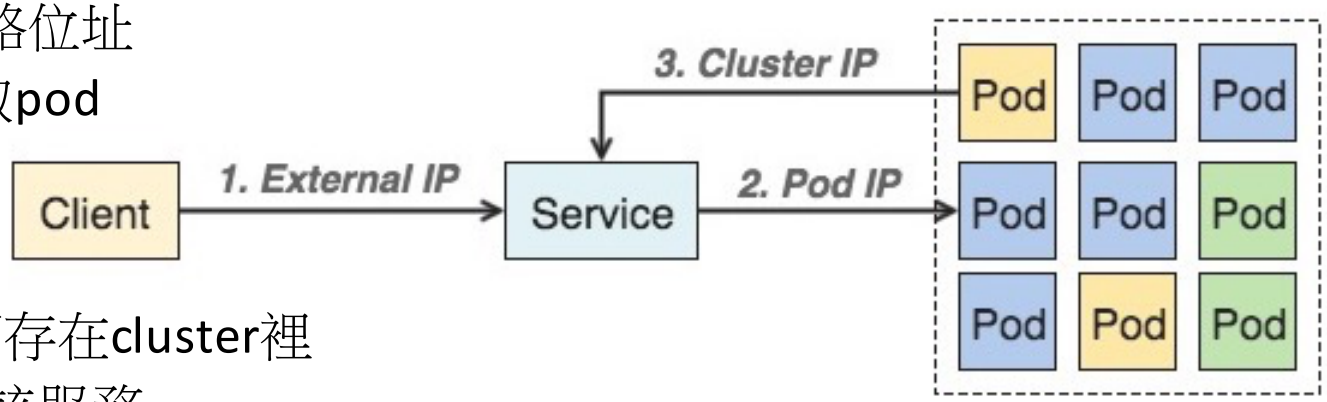


# Wordpress + MySQL的例子



# Kubernetes 的三種IP

- Pod IP
  - 每個 Pod 在overlay network上的網路位址
  - Cluster 內的 Pod/node 可用此ip存取pod
- Cluster IP
  - service在叢集內部的網路位址
  - 是一個virtual ip，不是real ip，且不存在cluster裡
  - 只有Cluster 內的 Pod 可用此ip存取該服務
- External IP
  - 透過cloud provider提供
  - Service暴露在外的位址，供外部使用者連線

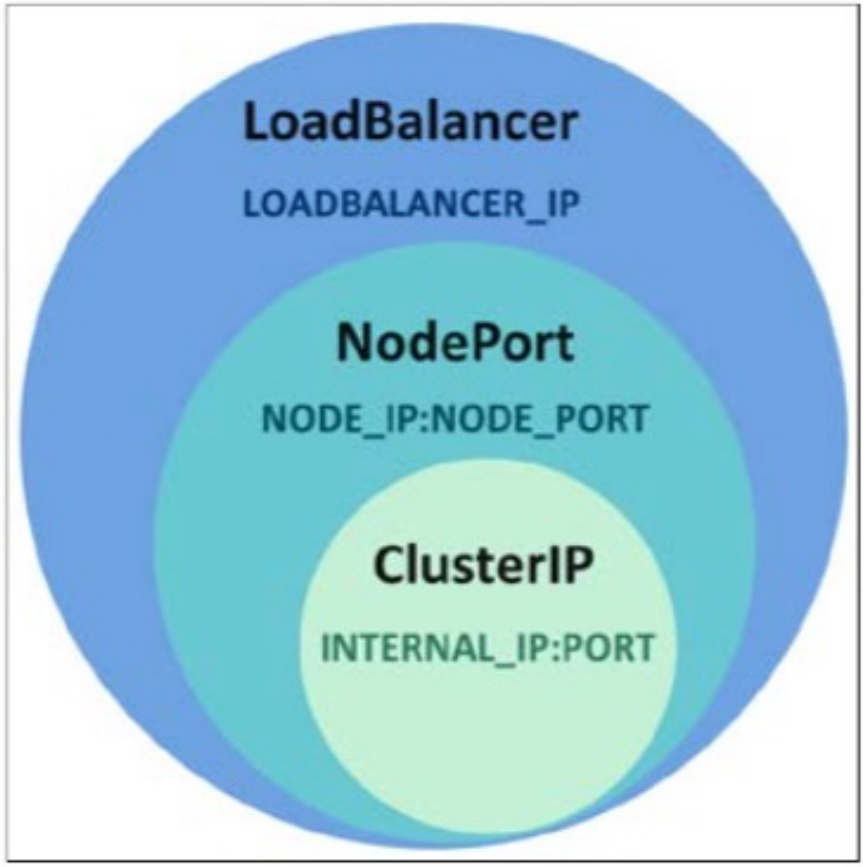



```
root@ubuntu:~# kubectl get pod -o wide
NAME                              READY     STATUS    RESTARTS   AGE       IP
nfs-provisioner-5bfd94c4b-86g7w   1/1       Running   0          11d       172.20.0.170
root@ubuntu:~# kubectl get svc -n=ingress-nginx
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
default-http-backend   ClusterIP   10.68.90.254    <none>        80/TCP
```

# 如何讓外部存取Pod

- Service
  - NodePort
  - Load Balancer
- HostNetwork
- HostPort
- Ingress

# Service Port type



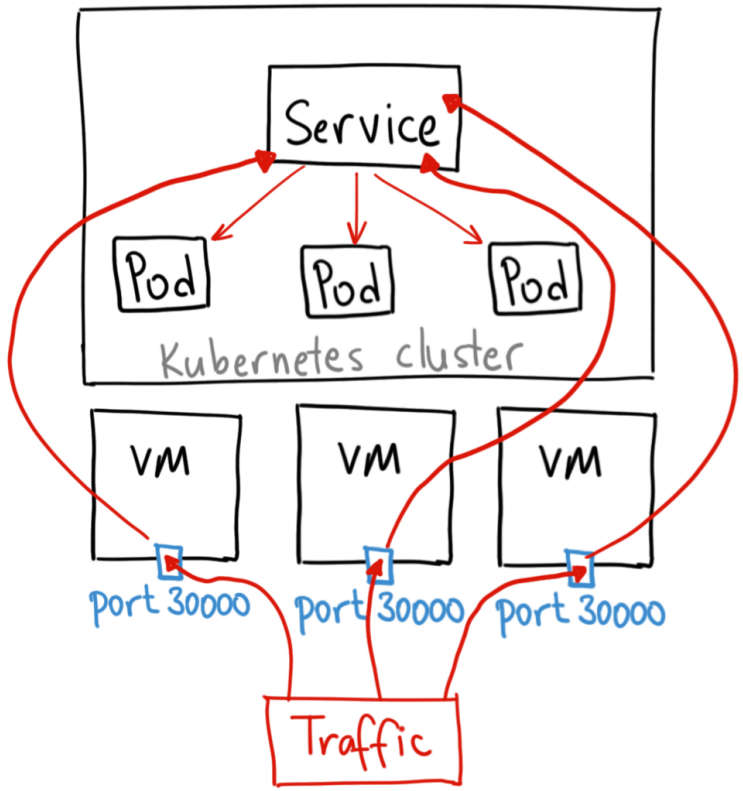

- ClusterIP: svc.Port→svc.targetPort
  1. Expose 一個cluster內部可存取的IP

- NodePort: nodePort→svc.Port→svc.targetPort
  1. Expose 一個cluster內部可存取的IP
  2. 在每個node的ip都expose相同的port

- Loadbalancer: LB→nodePort →svc.port→svc.targetPort
  1. Expose 一個cluster內部可存取的IP
  2. 在每個node的ip都expose相同的port
  3. Cloud provider提供一個LB的IP，此IP綁定到expose 的nodePort

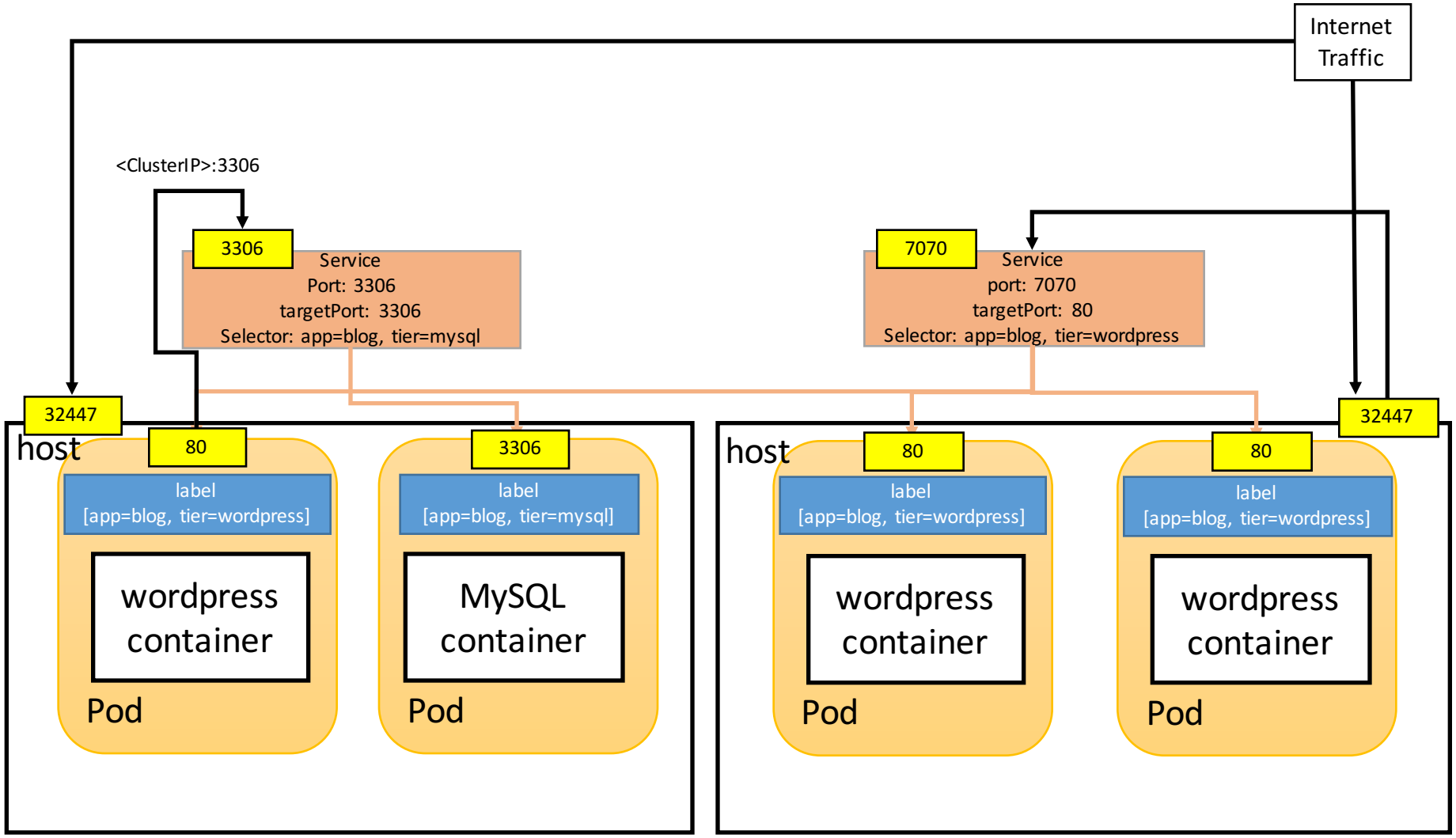
# Node Port

- Expose service to outside cluster
- One service uses one nodePort



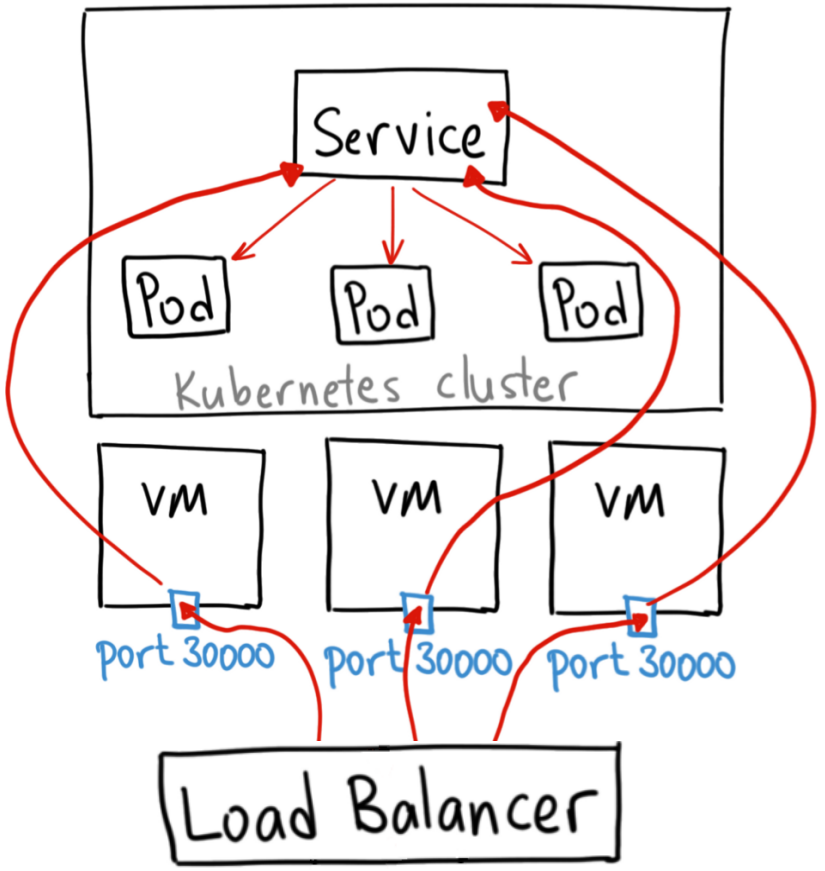

```
root@ubuntu:~# kubectl get svc -n=ingress-nginx
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
default-http-backend   ClusterIP   10.68.90.254    <none>        80/TCP                       11d
ingress-nginx          NodePort    10.68.163.140   <none>        80:32447/TCP,443:37619/TCP   10d
```

# Wordpress + MySQL的例子使用nodePort



# LoadBalancer

- Provided by cloud provider
- 若沒有Cloud provider, 則會一直pending

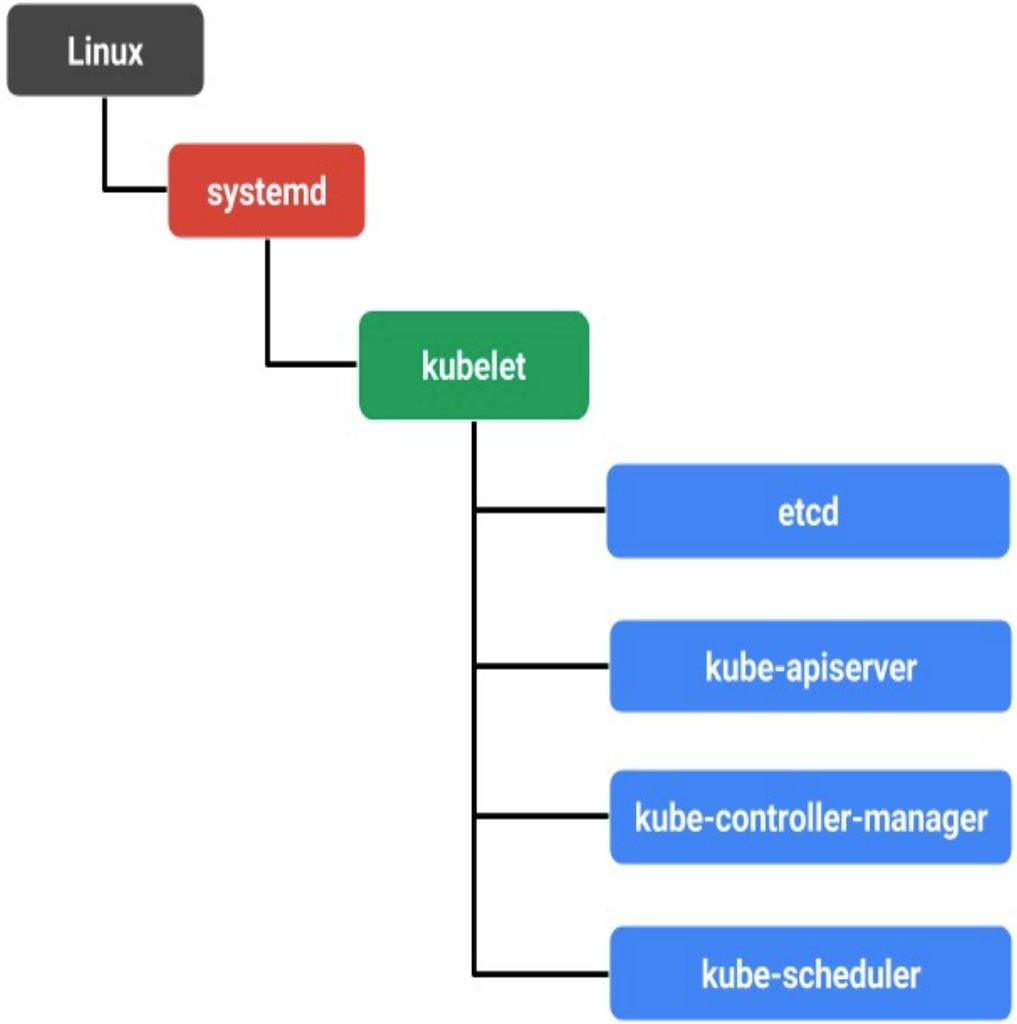



```
NAME          TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
hub           ClusterIP      10.68.169.105   <none>        8081/TCP                     12m
proxy-api     ClusterIP      10.68.242.208   <none>        8001/TCP                     12m
proxy-http    ClusterIP      10.68.144.45    <none>        8000/TCP                     12m
proxy-public  LoadBalancer   10.68.89.253    <pending>     80:24837/TCP,443:22383/TCP   12m
```

# 另二種存取Pod的方法

- Service
  - NodePort
  - Load Balancer
- HostNetwork
  - 定義在Pod裡，功用類似docker的host network 模式
  - Pod可以看到所在節點上的interface，而節點的所有interface都可以訪問該pod
- HostPort
  - 定義在Pod裡，功用類似docker的 expose port
  - 將container的port和所在節點的port做對應，可以透過節點ip + port來訪問container
- Ingress

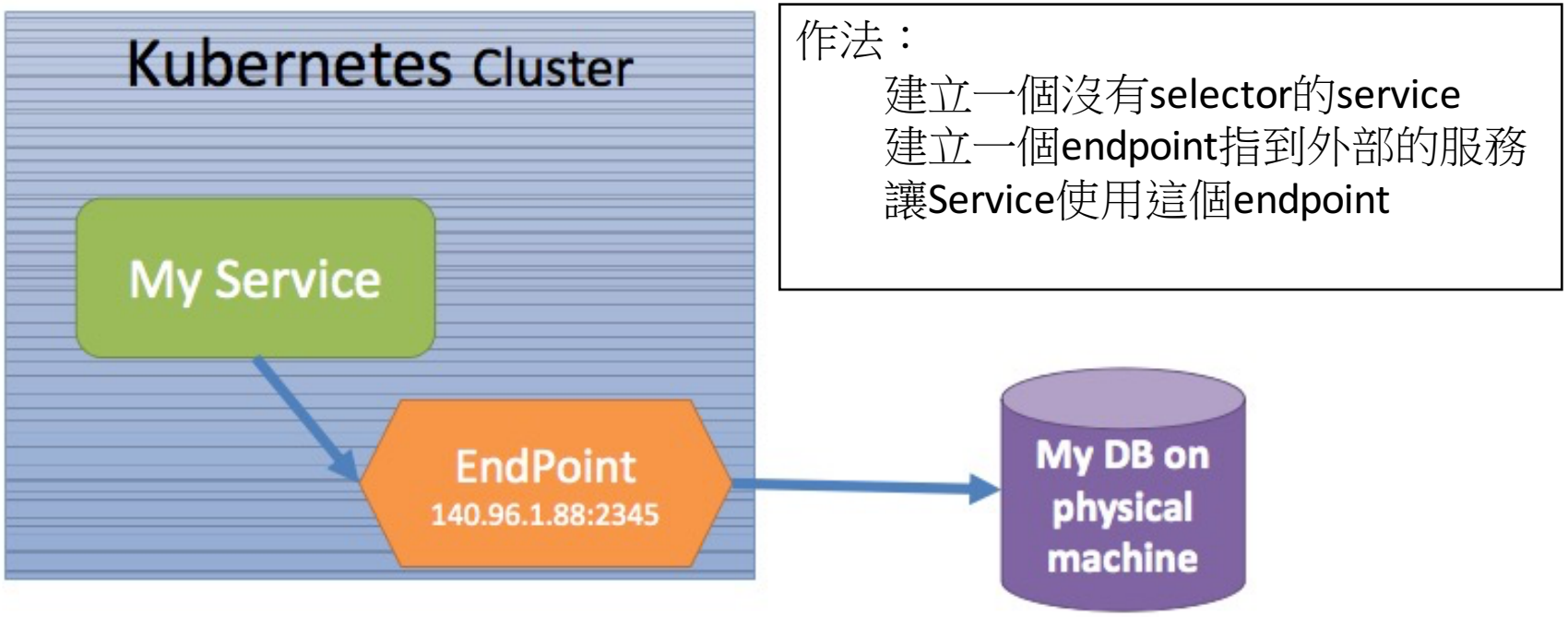
# 如何存取kubeadm上的control plane




```
{
  "kind": "Pod",
  "apiVersion": "v1",
  ...
  "spec": {
    "containers": [
      {
        "name": "kube-apiserver",
        "image": "gcr.io/google_containers/kube-apiserver-amd64:v1.4.0",
        "volumeMounts": [
          {
            "name": "certs",
            "mountPath": "/etc/ssl/certs"
          },
          {
            "name": "pki",
            "readOnly": true,
            "mountPath": "/etc/kubernetes/"
          }
        ],
        ...
      }
    ],
    "hostNetwork" true,
    ...
  }
}
```

# 沒有selector的Service

- Service是透過selector決定要把traffic導給pod
- 是否可以將traffic導到別的地方? 例如一個不是被kubernetes管理的server



# Summary

- Service有ClusterIP, NodPort, LoadBalancer三種
- 透過Service，可以用<SVC>.<NAMEPSACE>存取Pod
- 要讓外部存取pod，可以使用NodPort, LoadBalancer，Ingress, HostPort, HostNetork

# Secret

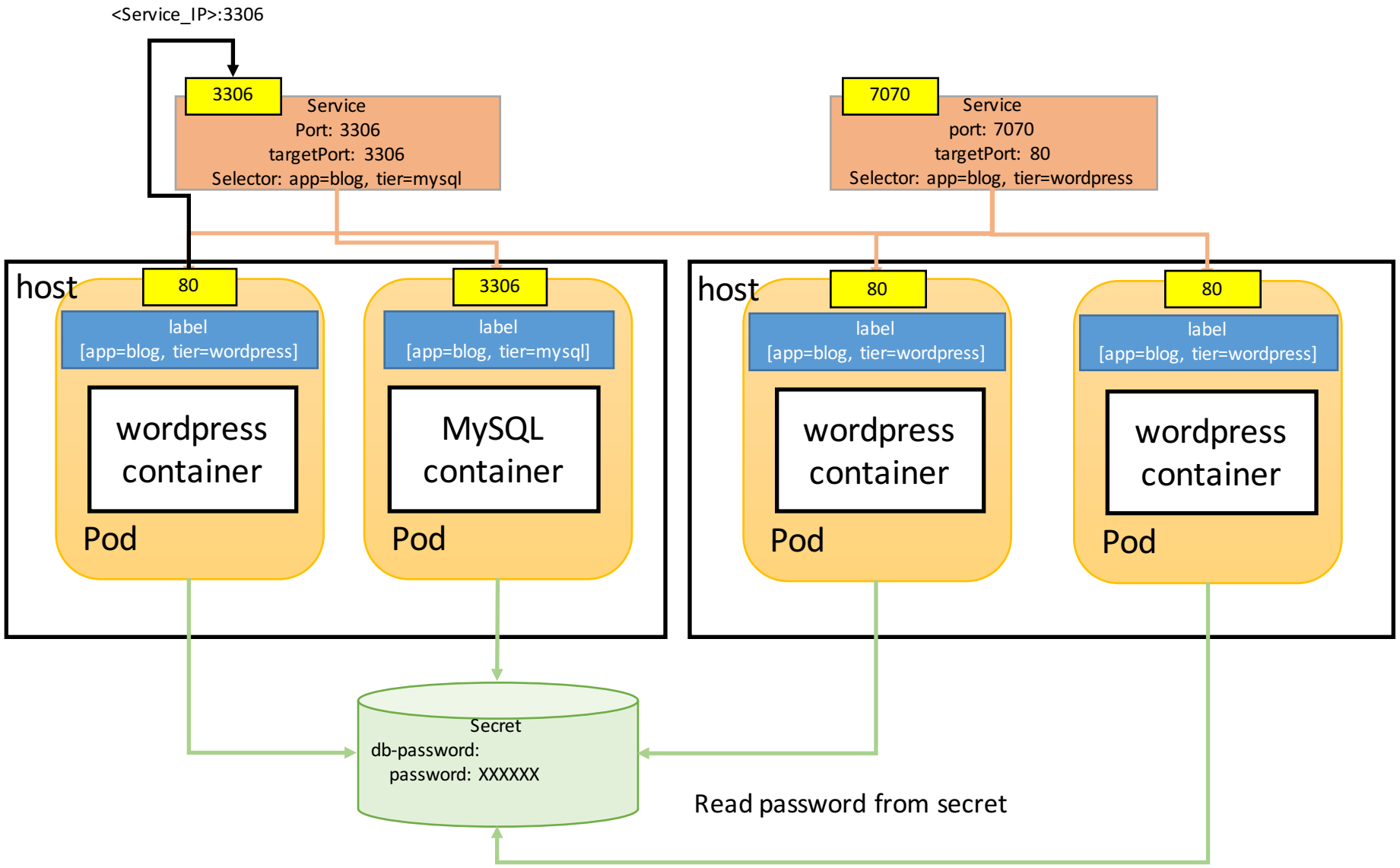
- 避免將機密資訊直接存放在Pod裡
- 透過mount / 環境變數 在 Pod裡使用
- 其實不安全
- 機密資料需先透過base64編碼
- 大小只能1MB

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-password
type: Opaque
data:
  password: UGFzc3dvcmQxMjM0
```

```yaml
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    volumeMounts:
    - name: db-secret
      mountPath: "/etc/db-secret"
      readOnly: true
  volumes:
  - name: db-secret
    secret:
      secretName: db-password
```

```yaml
spec:
  containers:
  - name: mysql
    image: mysql:5.6
    env:
    - name: MYSQL_ROOT_PASSWORD
      valueFrom:
        secretKeyRef:
          name: db-password
          key: password
    volumeMounts:
    - name: mysql-storage
      mountPath: /var/lib/mysql
  volumes:
  - name: mysql-storage
    hostPath:
      path: /tmp/data
```

# Wordpress + MySQL的例子



# ConfigMap

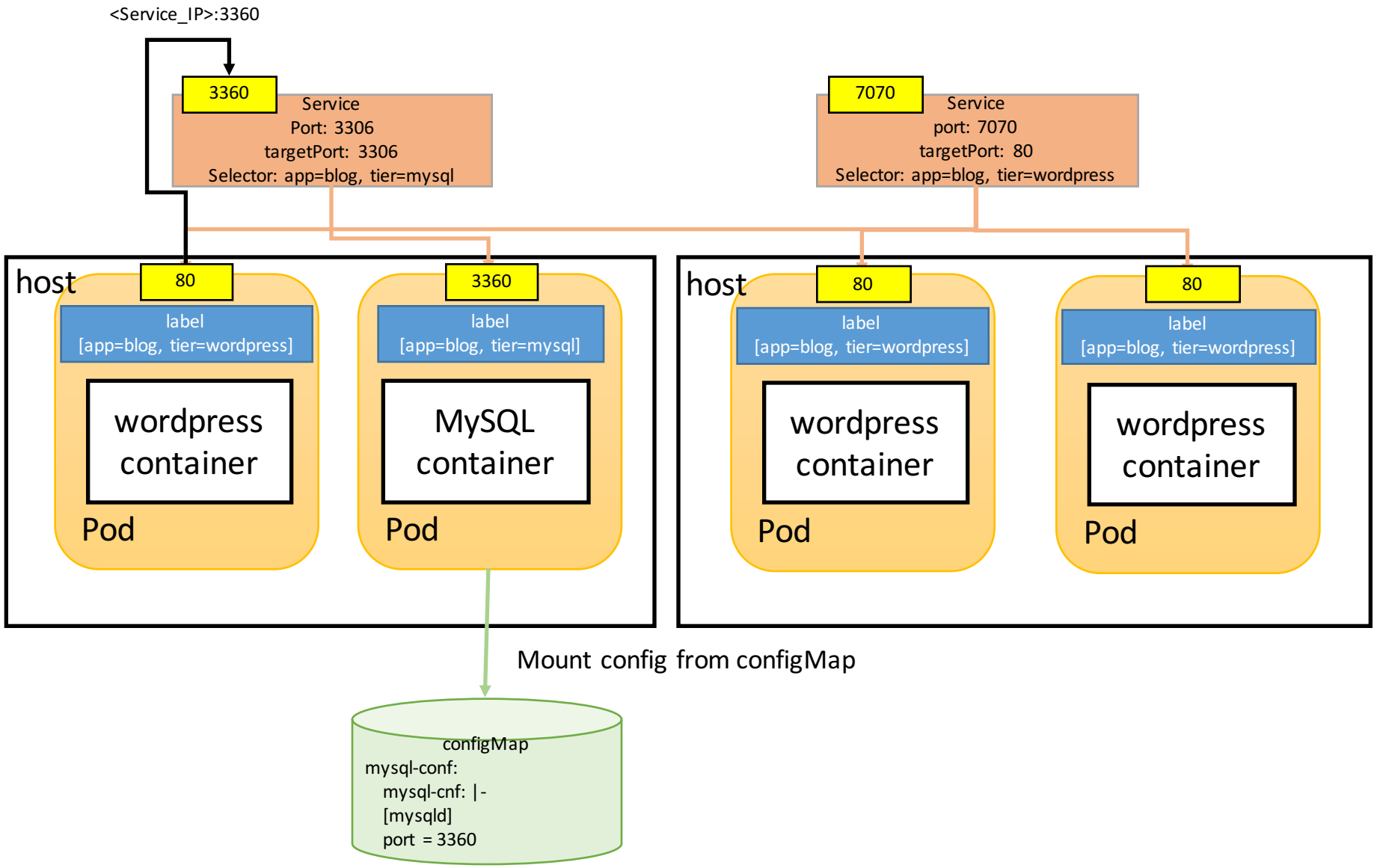
- 避免將非機密資訊直接存放在Pod裡
- 透過mount / 環境變數 在 Pod裡使用
- 其實不安全
- ~~機密資料需先透過base64編碼~~
- ~~大小只能1MB~~

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-conf
data:
  mysql-cnf: |-
    [mysqld]
    port = 3360
  mysql-port: "3360"
```

```yaml
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    env:
      - name: MYSQL_PORT
        valueFrom:
          configMapKeyRef:
            name: mysql-conf
            key: mysql-port
```

```yaml
spec:
  containers:
  - name: mysql
    image: mysql:5.6
    env:
    - name: MYSQL_ROOT_PASSWORD
      valueFrom:
        secretKeyRef:
          name: db-password
          key: password
    volumeMounts:
    - name: mysql-storage
      mountPath: /var/lib/mysql
    - name: mysql-conf
      mountPath: /etc/mysql/
  volumes:
  - name: mysql-storage
    hostPath:
      path: /tmp/data
  - name: mysql-conf
    configMap:
      name: mysql-conf
      items:
      - key: mysql-cnf
        path: my.cnf
```

# 在Wordpress + MySQL的例子中



# Ingress

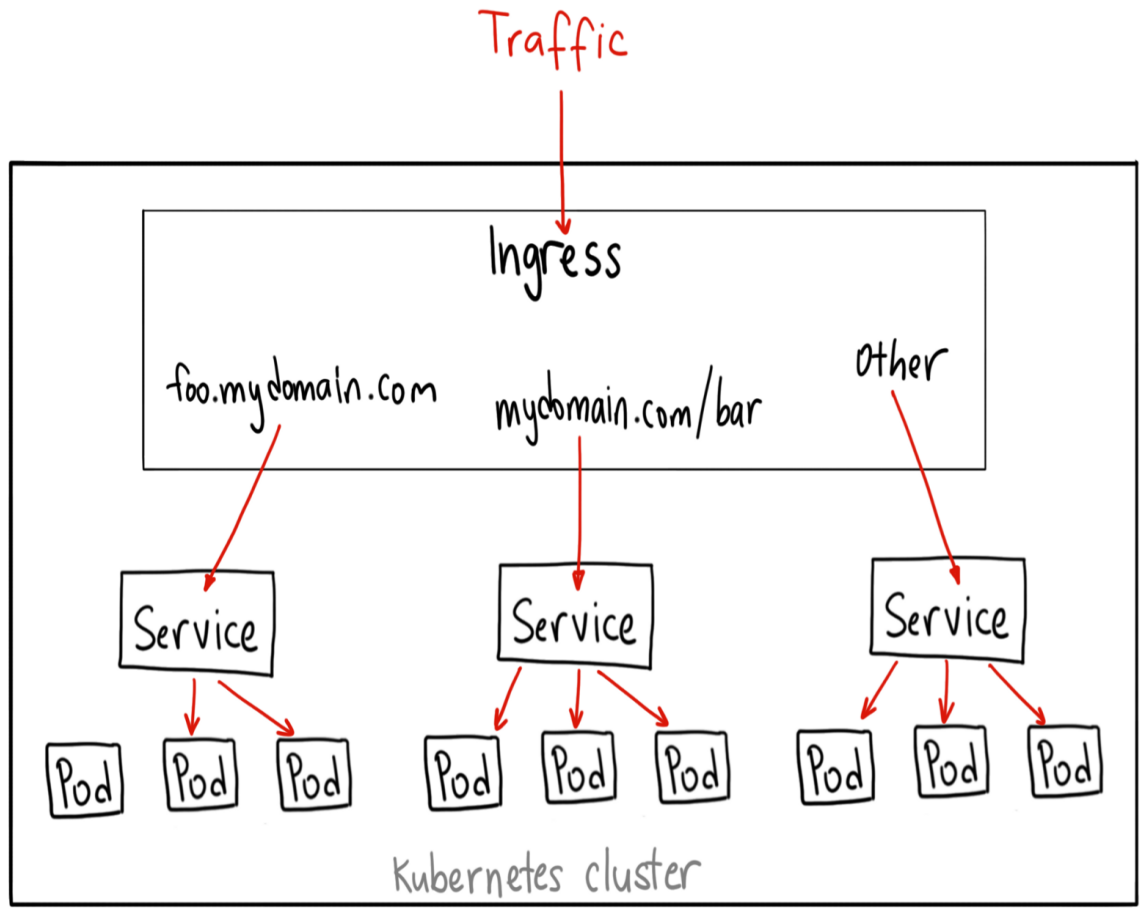
- 如何 expose service給外部？
  - NodePort: 每個服務要對應一個nodePort
  - Load Balancer: 必需有cloud provider支援
  - Ingress: 支援L7 & L4

- Ingress 包括
  - Ingress
    - 設定轉發的規則
  - Ingress-controller
    - 有不同的實作方法，nginx-controller是用用nginx當proxy進行轉發
  - Service for ingress-controller
    - 使用nodePort讓外面存取不同服務的request都進入ingress-controller

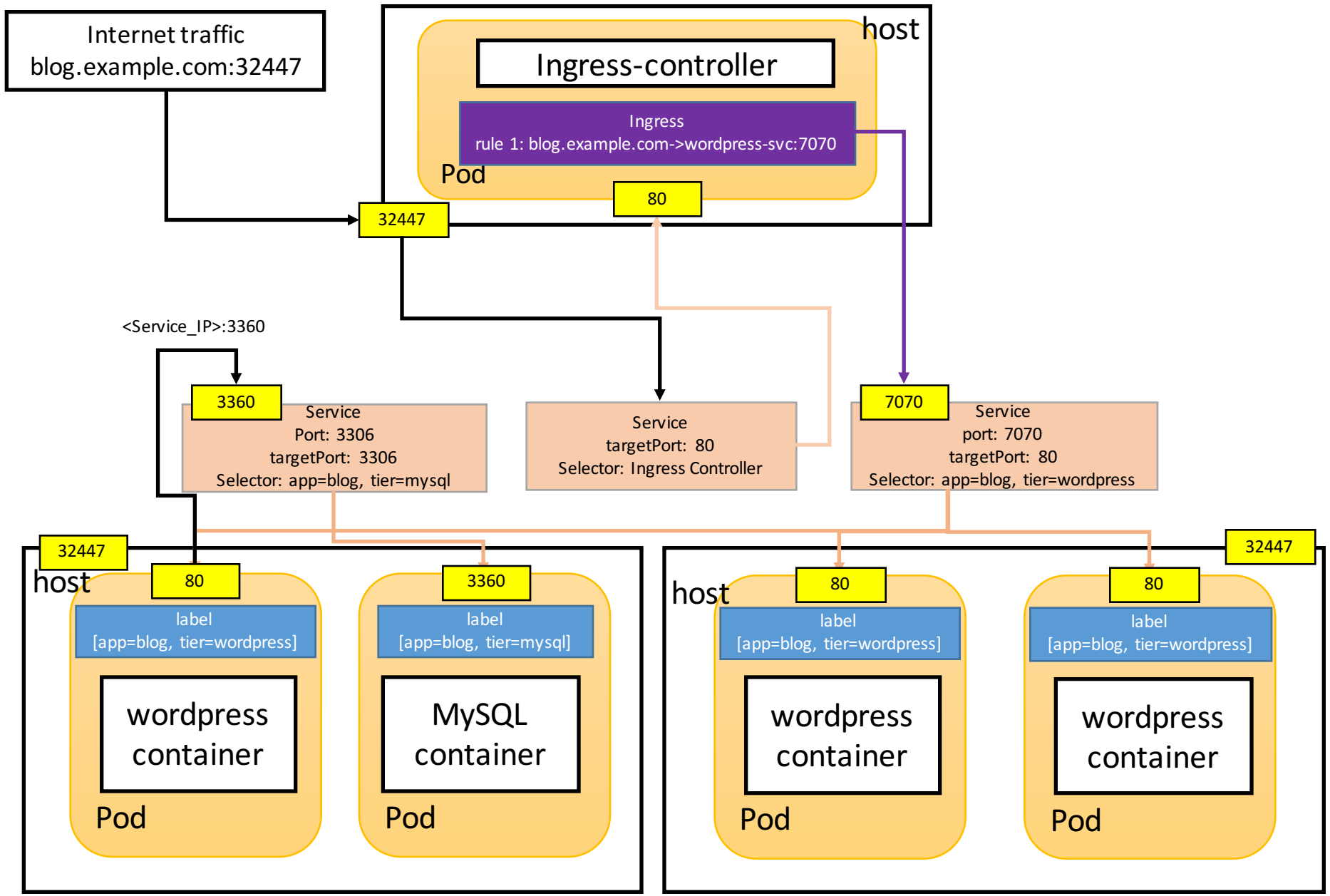
```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: wordpress-ingress
  namespace: jimmy
spec:
  rules:
  - host: blog.example.com
    http:
      paths:
      - backend:
          serviceName: wordpress-svc
          servicePort: 80
  - host: jupyter.example.com
    http:
      paths:
      - backend:
          serviceName: proxy-public
          servicePort: 80
```

Rule-1

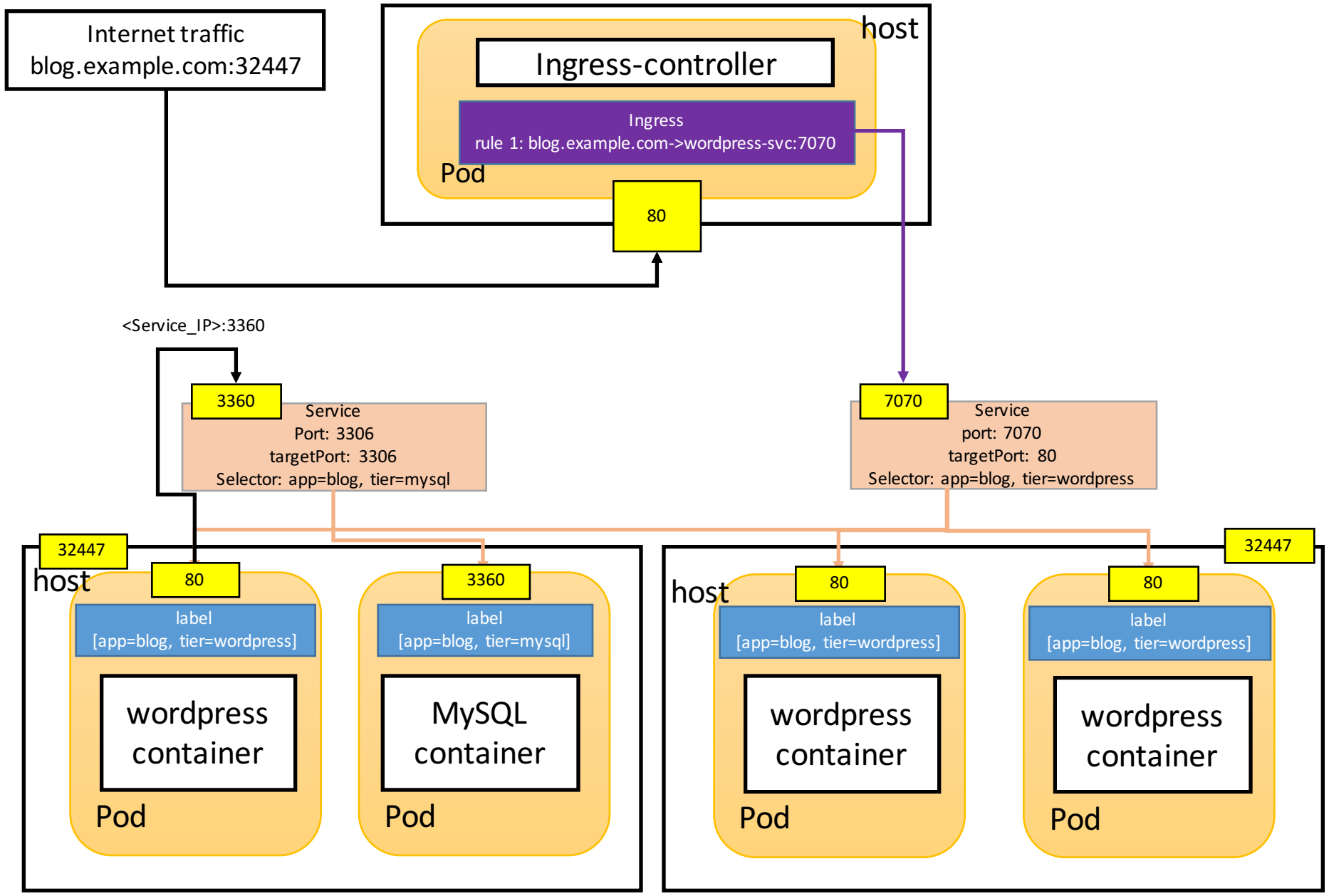
Rule-2



# Wordpress + MySQL的例子



# Ingress controller 採用hostPort



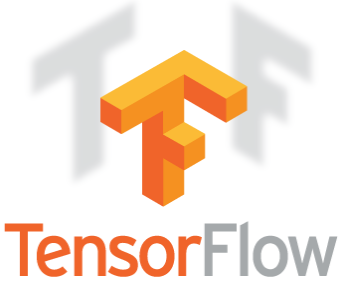
# Today Topic

- Docker Review
- Kubernetes簡介
  - 什麼是 Kubernetes
  - 安裝 Kubernetes
  - Kubernetes 基本操作
- Kubernetes Basic Concept
  - 以WordPress + MySQL建立服務為例
- Kubernetes Advanced Topic

# Kubernetes advanced Topic

- TensorFlow on Kubernetes
- Distributed TensorFlow on Kubernetes
- Support GPU in Kubernetes

# Tensorflow

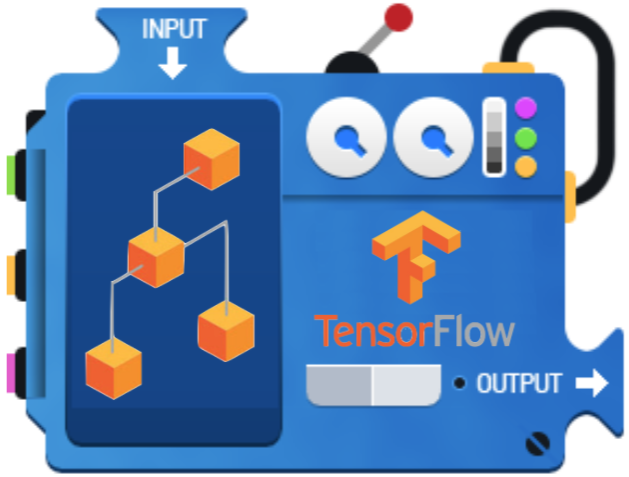



- TensorFlow™ is an open source software library for numerical computation using data flow graphs.
- Tensor:
  - multidimensional data arrays, 1-D is array, 2-D is matric, etc
- Dataflow
  - Nodes in the graph represent mathematical operations
  - Edges represent the tensors communicated between node

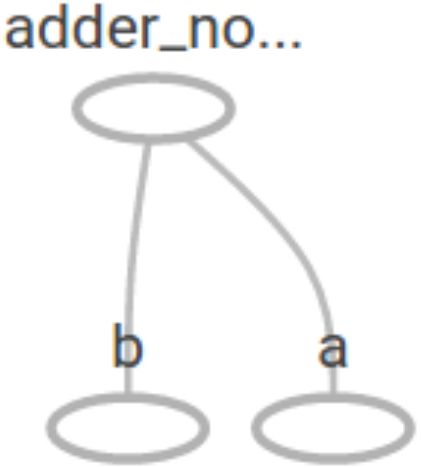
# Data flow graphs

2 feed data and run graph (operation)
***sess.run (op)***

1 Build graph using
TensorFlow operations




3 update variables
in the graph
(and return values)




```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = a + b  # + provides a shortcut for tf.add(a, b)

print(sess.run(adder_node, feed_dict={a: 3, b: 4.5}))
print(sess.run(adder_node, feed_dict={a: [1,3], b: [2, 4]}))
```

1 2 3

```
7.5
[ 3.  7.]
```

# Tensorflow on Kubernetes

- Tensorflow是python library, 因此只要將tensorflow 安裝至image裡即可

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/tools/docker/Dockerfile

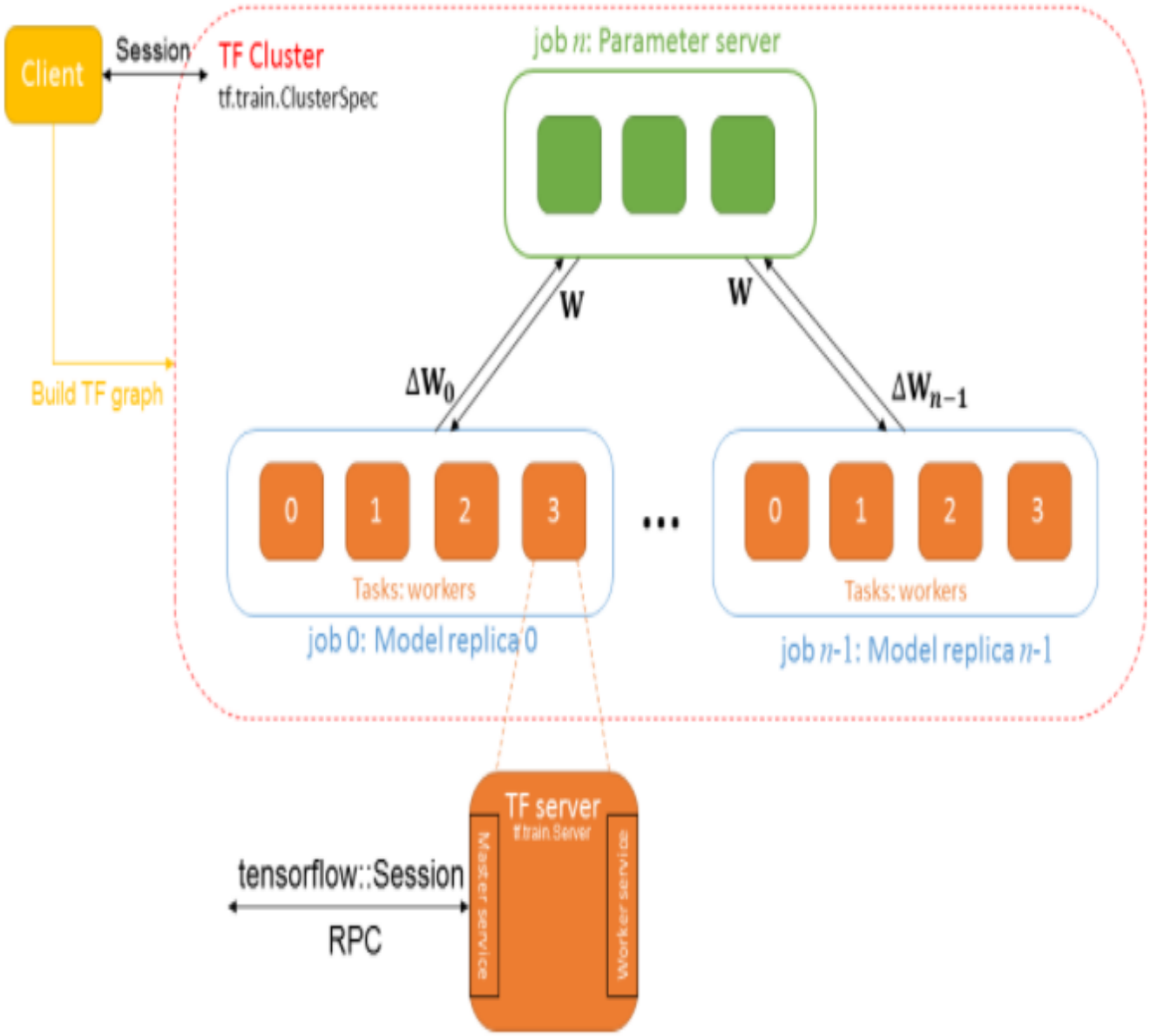
- Tensorflow可以透過Jupyter進行web-based存取，在kubernetes裡透過Service/ingress讓外部使用

# Kubernetes advanced Topic

- TensorFlow on Kubernetes
- Distributed TensorFlow on Kubernetes
- Support GPU in Kubernetes

# What is Distributed TF

- Cluster
  - A set of "tasks" that participate in the distributed execution of a TensorFlow graph
  - Define multiple Jobs by ClusterSpec
  - Composed by multiple processes (i.e. tasks)
- Job
  - A cluster has multiple jobs
  - Two types of job: worker job or parameter server job.
  - Each Job has multiple tasks
- Task
  - Each task has worker and master
  - worker executes operations in the graph
  - Master is used to create sessions



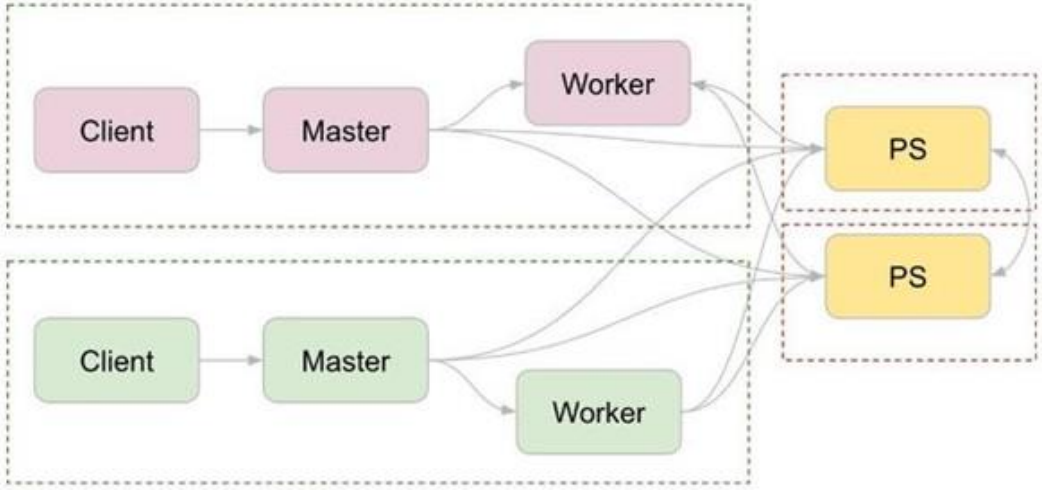

- 1 cluster has
  - 1 parameter server Job
    - 3 tasks
  - 2 worker jobs
    - Worker0 has 4 tasks
    - Worker1 has 4 tasks

```
cluster = tf.train.ClusterSpec({
    "worker0": ["worker0-0.example.com:2222",
                "worker0-1.example.com:2222",
                "worker0-2.example.com:2222",
                "worker0-3.example.com:2222"],
    "worker1": ["worker1-0.example.com:2222",
                "worker1-1.example.com:2222",
                "worker1-2.example.com:2222",
                "worker1-3.example.com:2222"],
    "ps": ["ps0.example.com:2222",
           "ps1.example.com:2222",
           "ps2.example.com:2222"]
})
```

# How to run Distributed tensorflow

- 手動啟動多個 PS task 和多個 worker task
  - 1 ps job has two ps tasks
  - 1 worker job has two worker tasks
- 傳入不同參數決定每個 process 是worker task 還是 PS task
- https://www.tensorflow.org/deploy/distributed

```
# On ps0.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=ps --task_index=0
# On ps1.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=ps --task_index=1
# On worker0.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=worker --task_index=0
# On worker1.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=worker --task_index=1
```



- 透過參數決定ClusterSpec

```python
def main(_):
  ps_hosts = FLAGS.ps_hosts.split(",")
  worker_hosts = FLAGS.worker_hosts.split(",")

  # Create a cluster from the parameter server and worker hosts.
  cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})

  # Create and start a server for the local task.
  server = tf.train.Server(cluster,
                           job_name=FLAGS.job_name,
                           task_index=FLAGS.task_index)
```

```python
if __name__ == "__main__":
  parser = argparse.ArgumentParser()
  parser.register("type", "bool", lambda v: v.lower() == "true")
  # Flags for defining the tf.train.ClusterSpec
  parser.add_argument(
      "--ps_hosts",
      type=str,
      default="",
      help="Comma-separated list of hostname:port pairs"
  )
  parser.add_argument(
      "--worker_hosts",
      type=str,
      default="",
      help="Comma-separated list of hostname:port pairs"
  )
  parser.add_argument(
      "--job_name",
      type=str,
      default="",
      help="One of 'ps', 'worker'"
  )
  # Flags for defining the tf.train.Server
  parser.add_argument(
      "--task_index",
      type=int,
      default=0,
      help="Index of task within the job"
  )
  FLAGS, unparsed = parser.parse_known_args()
  tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

- 透過參數決定每個task要計算的graph

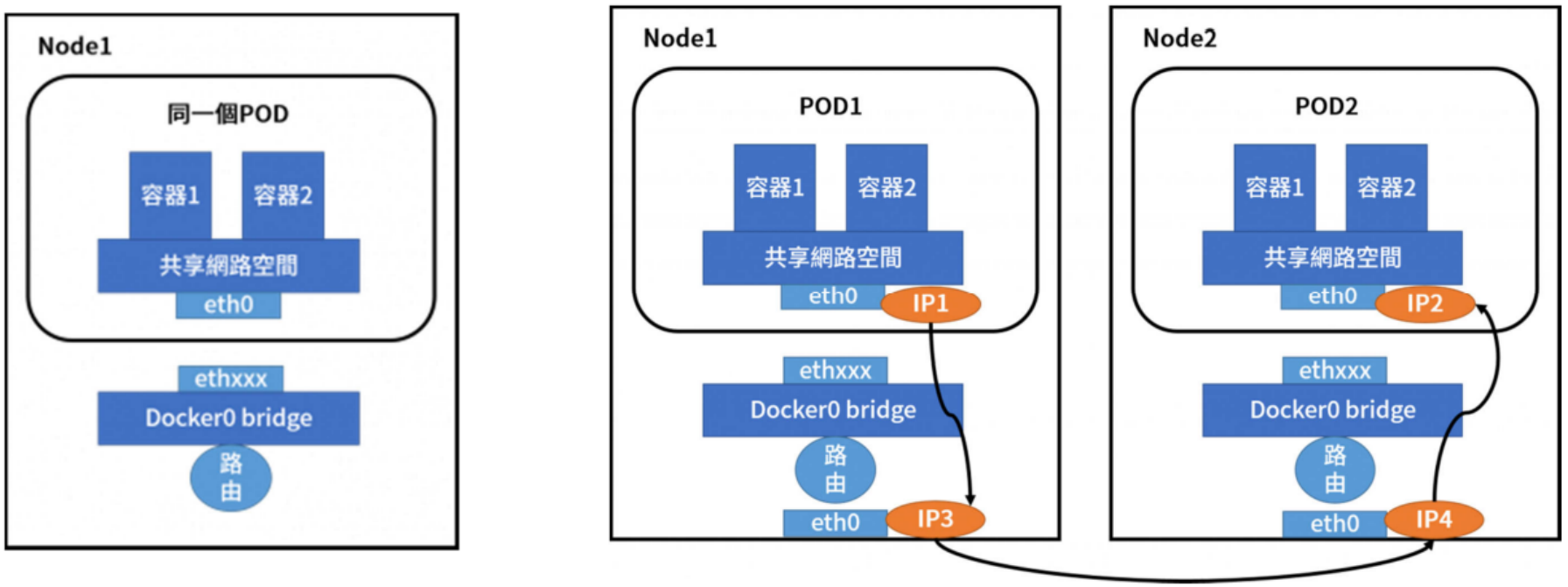
```
if FLAGS.job_name == "ps":
  server.join()
elif FLAGS.job_name == "worker":

  # Assigns ops to the local worker by default.
  with tf.device(tf.train.replica_device_setter(
      worker_device="/job:worker/task:%d" % FLAGS.task_index,
      cluster=cluster)):

    # Build model...
    loss = ...
    global_step = tf.contrib.framework.get_or_create_global_step()

    train_op = tf.train.AdagradOptimizer(0.01).minimize(
        loss, global_step=global_step)
```

# Distributed TensorFlow的缺陷

**Note:** Manually specifying these cluster specifications can be tedious, especially for large clusters. We are working on tools for launching tasks programmatically, e.g. using a cluster manager like Kubernetes. If there are particular cluster managers for which you'd like to see support, please raise a GitHub issue.

| Distributed tensorflow | Tensorflow on Kubernetes |
| --- | --- |
| TensorFlow各個Task資源無法隔離 | 提供ResourceQuota, LimitRanger等多種資源管理機制，能做到任務之間很好的資源隔離 |
| 缺乏調度能力，需要用戶手動配置和管理任務的計算資源 | 支持任務的計算資源的配置和調度 |
| 集群規模大時，訓練任務的管理很麻煩，要跟蹤和管理每個任務的狀態 | 訓練任務以容器方式運行，因此任務狀態的管理很方便 |
| 用戶要查看各個Task的訓練日誌需要找出對應的服務器，並ssh過去 | 透過kubectl log或建立elk等，能方便的查看任務日誌 |

- Parameter server and workers are containers in the same Pod
  - Communicate between different ports in localhost
- Parameter server and workers are in different Pod
  - Communication between Pods



# GPU Support in Kubernetes

- v1.7之前，使用Accelerators
- v1.8之後，使用Device Plugins
  - Containers do not share GPUs.
  - container can request one or more GPUs.
    - But It is not possible to request a fraction of a GPU.
  - Can not differentiate between types of GPUs

```
containers:
- name: worker0
  image: tensorflow/tensorflow:1.0.1-gpu
  ports:
  - containerPort: 2222
  command: ["/bin/sh", "-c"]
  args: ["curl -L https://raw.githubusercontent.com/ogre0403/Distrib
         python /opt/bg_dist.py --job_name=worker --task_index=0"]
  resources:
    limits:
      alpha.kubernetes.io/nvidia-gpu: 1 # requests one GPU
```

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod-restart
spec:
  containers:
    - name: cuda-container
      image: gcr.io/tensorflow/tensorflow:latest-gpu
      command: ["python"]
      args:
      - -u
      - -c
      - from tensorflow.python.client import device_lib;
      resources:
        limits:
          nvidia.com/gpu: 1 # requesting 2 GPUs
```

# Use different types of GPUs

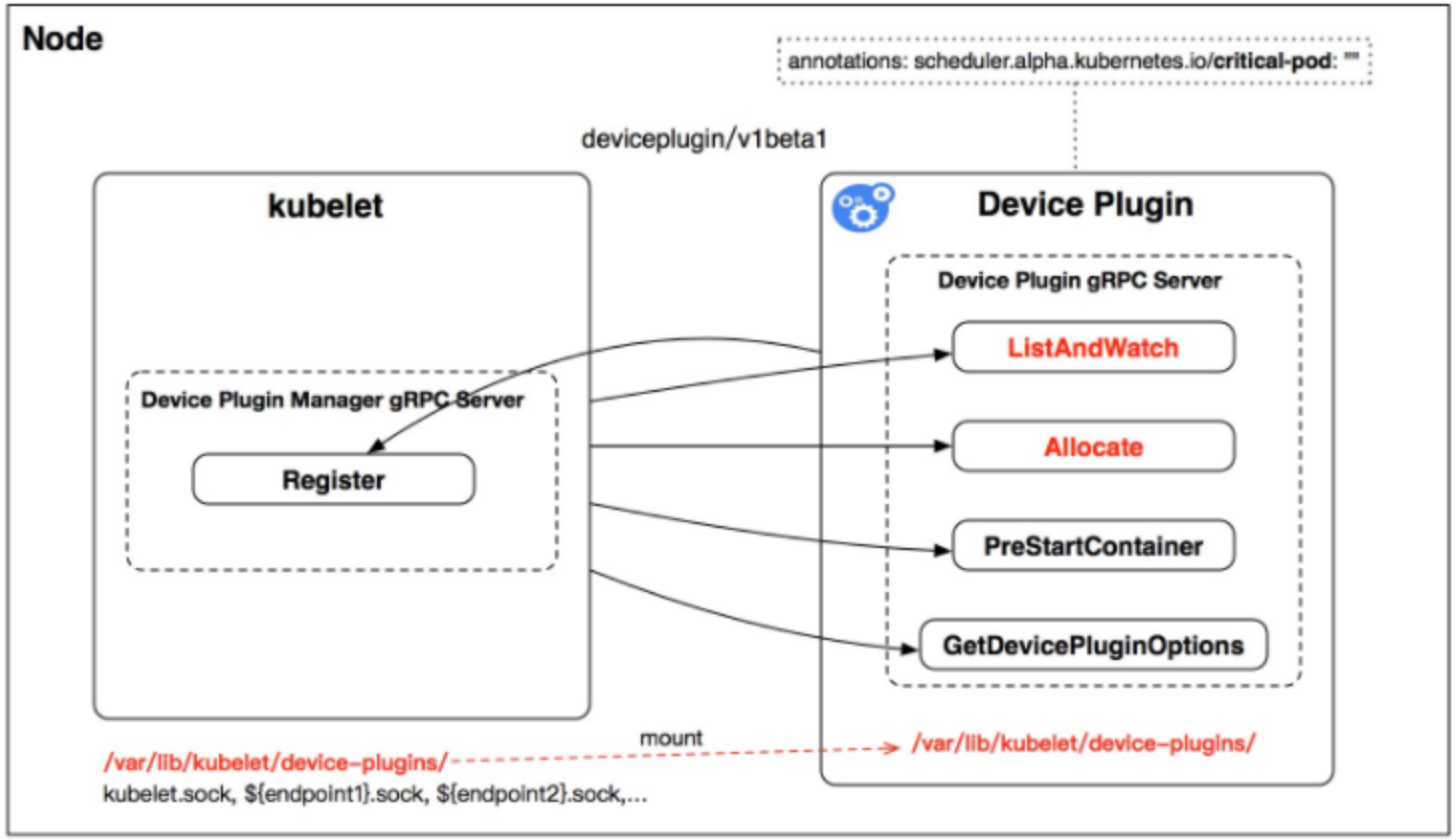
- If your nodes have different types of GPUs,
  - Use NodeSelector

# Limits & requests

- requests <= limits
  - Requests: 作為pod調度時的參考依據
  - Limits: 限制每個容器使用資源的最大值

- 申請所需要的GPU數量
  - 透過limits申請GPU
  - Requests在申請GPU資源不需設定，若有設定，必需和limits相同(ie. Limits = requests)
  - 目前若limits=0或沒設定則會有問題
    - Issue #59631
    - Issue #59629

# Nvidia GPU device plugin



# Tensorflow using GPU

- 若同時有cpu與gpu，會優先使用gpu
- 透過 tf.device(“/gpu:0”) 設定使用的gpu設備

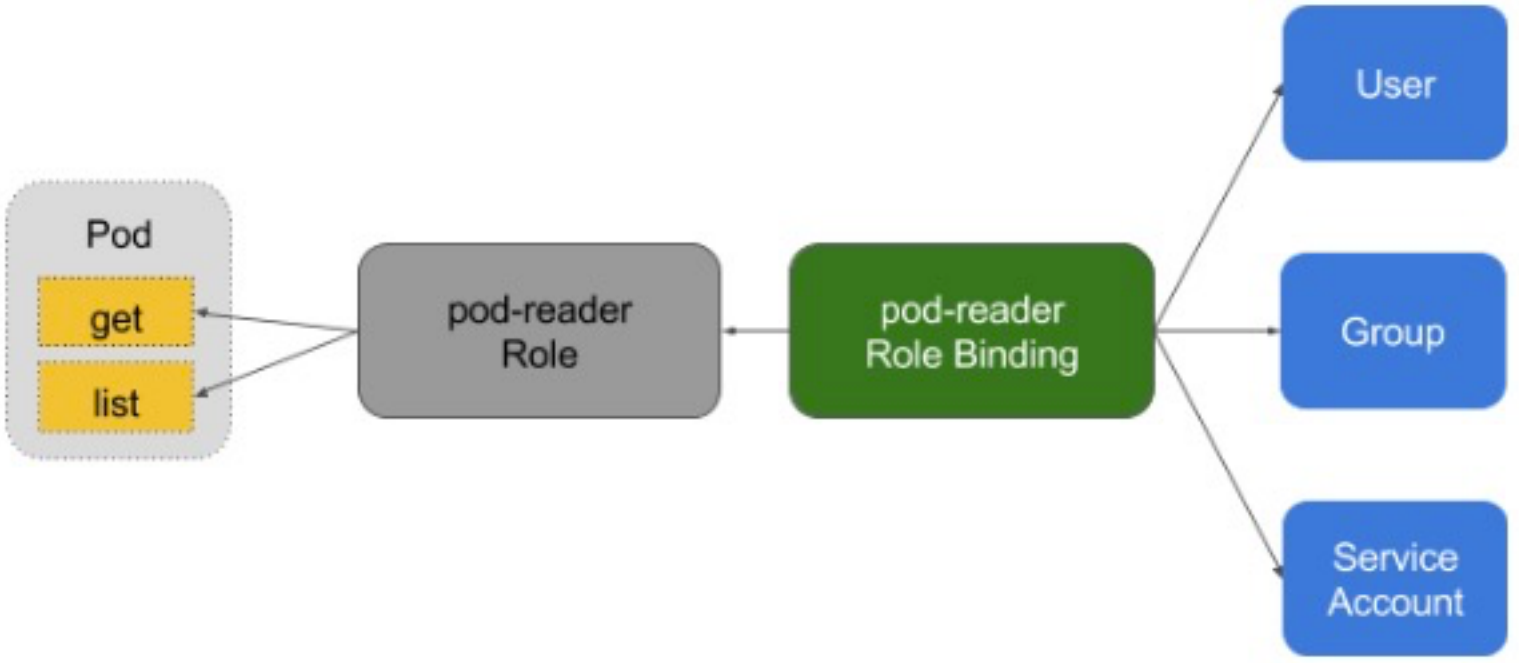
```
 import tensorflow as tf


with tf.device('/gpu:2'):
  a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
  b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)
sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True, log_device_placement=True))
print sess.run(c)
```

# GPU 環境

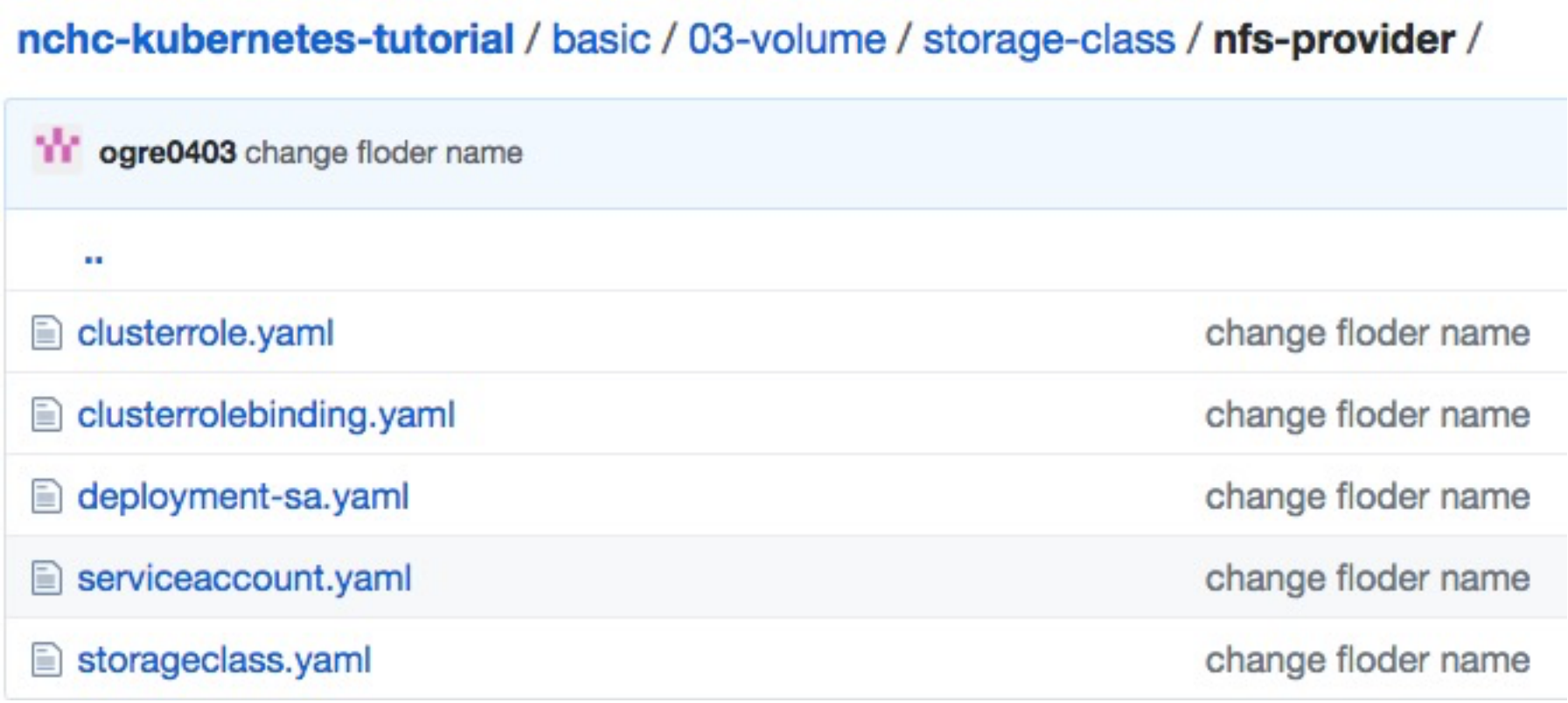
- cht只能存取cht namespace

```
jimmy@cht-guest-node:~$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://172.16.0.50:6443
  name: default-cluster
contexts:
- context:
    cluster: default-cluster
    namespace: cht-demo
    user: cht
  name: cht-demo-ctx
current-context: cht-demo-ctx
kind: Config
preferences: {}
users:
- name: cht
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

# Introduction to RBAC



# RBAC example: NFS-provisioner



# RBAC Example: User permission

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
  name: cht-demo-rolebinding
  namespace: cht-demo
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: edit
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: cht
```

# Thank you !!

ogre0403@gmail.com

課程資料