KUBERNETES 101
HANDS-ON

- Pod
- Volume
- Secret
- Configmap
- Service
- Deployment
- HPA

Hello World

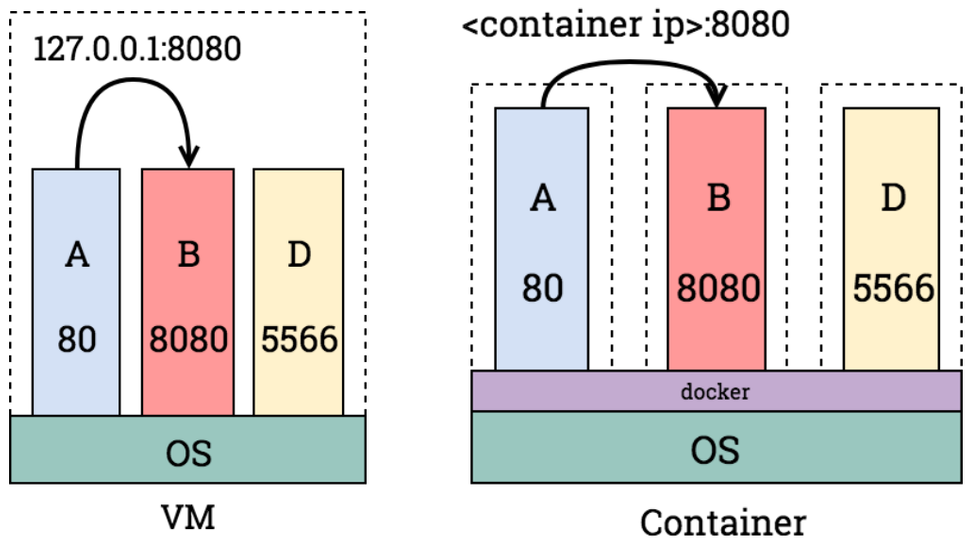
自我簡介

- Ta-Ching Chen (陳大慶)
- DevOps/Backend Engineer
- System Architecture Design/Implementation, Performance Tuning
- Blog: https://tachingchen.com/tw/
- Email: contact@tachingchen.com

101Repo

課程教材

- https://github.com/life1347/k8s-tutorial

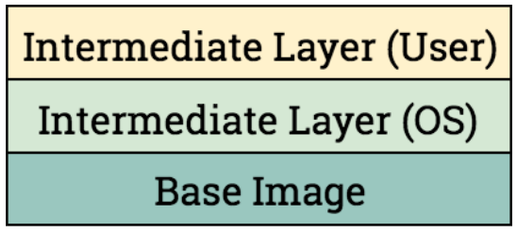



```python
# our base image
FROM alpine:3.5

# Install python and pip
RUN apk add --update py2-pip

# upgrade pip
RUN pip install --upgrade pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# tell the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "/usr/src/app/app.py"]
```

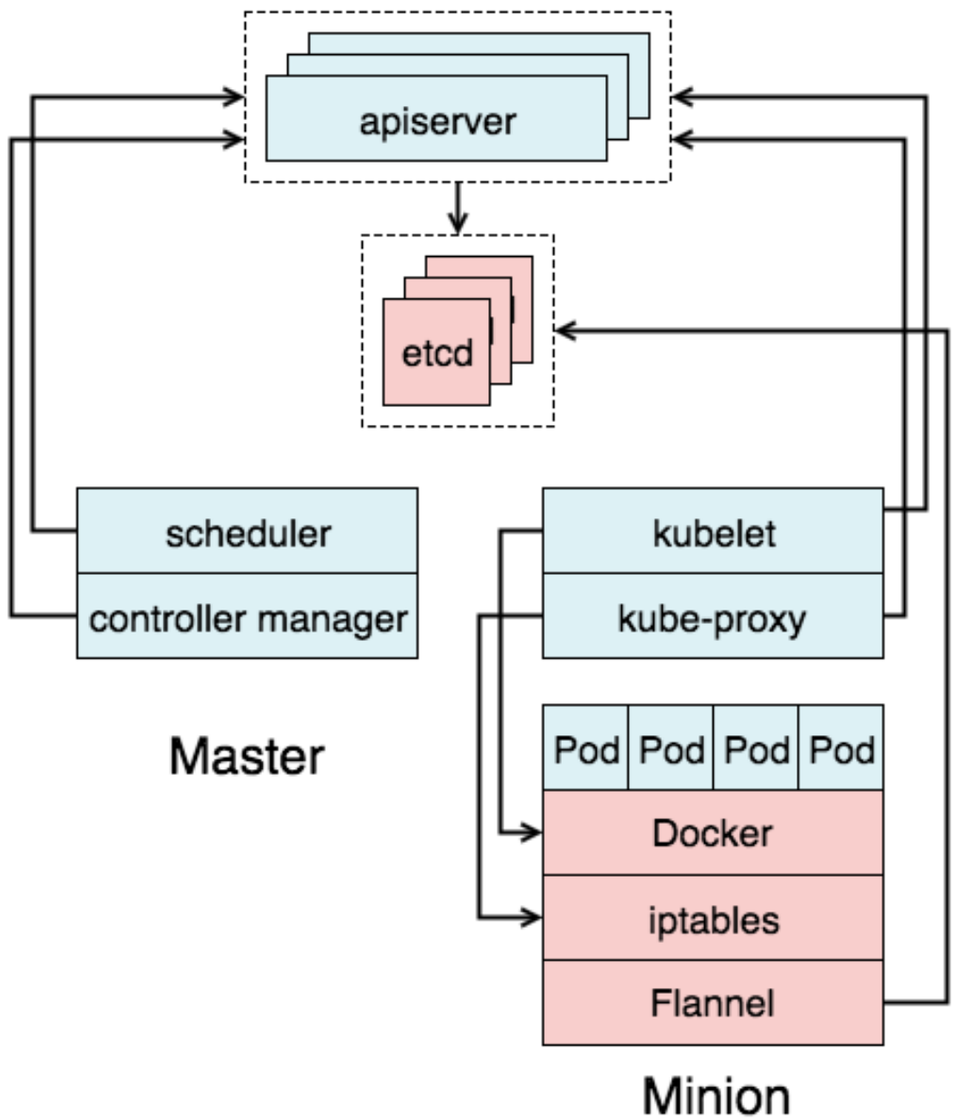
容器技術

原理

- 透過 Linux 功能將 process 隔離
  - cgroups: 資源限制
  - namespace: pid、網路、掛載點
- Image 組成要素



- Base Image
  - No FROM or FROM scratch
- Parent image:
  - FROM xxxxx
  - 包含 Linux 發行版檔案
  - (Optional) 使用者檔案
- Layer, Intermediate Layer:
  - 使用者檔案



KUBERNETES

架構與特點

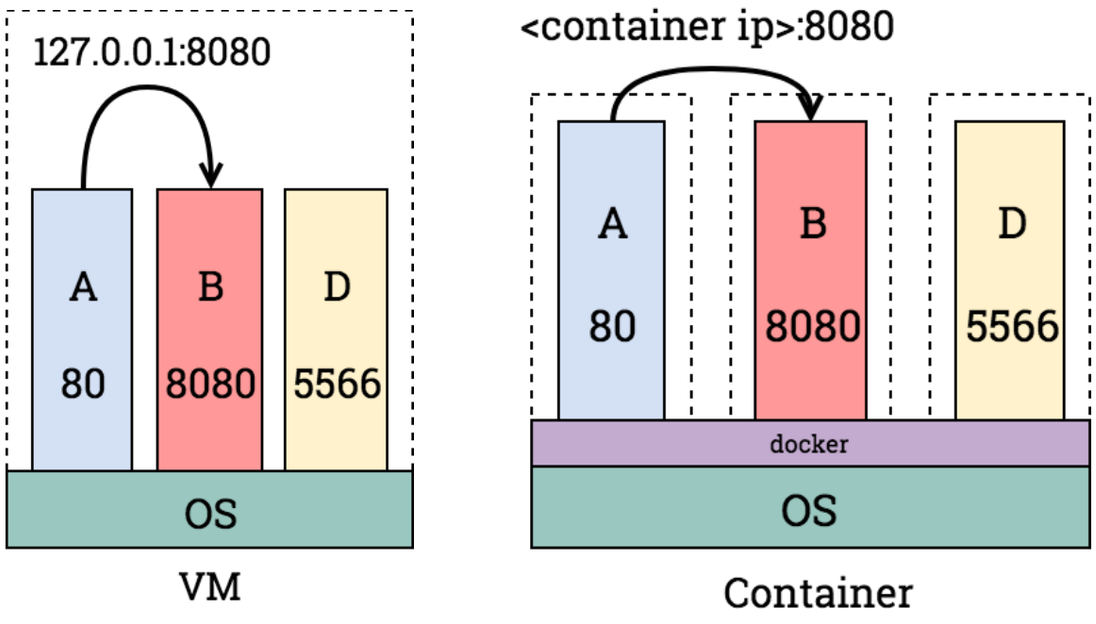
- 協助管理容器生命週期
- 服務損壞自動嘗試修復 (Auto healing)
- 流量自動負載平衡 (LB)
- 滾動升級與回滾 (Rolling update/Rollback)
- 簡易服務探索 (Service discovery)
- 容器可跨機器溝通
- 自動伸縮擴展 (Auto scaling)
- 可版控所有部署程式碼
- 資源控管

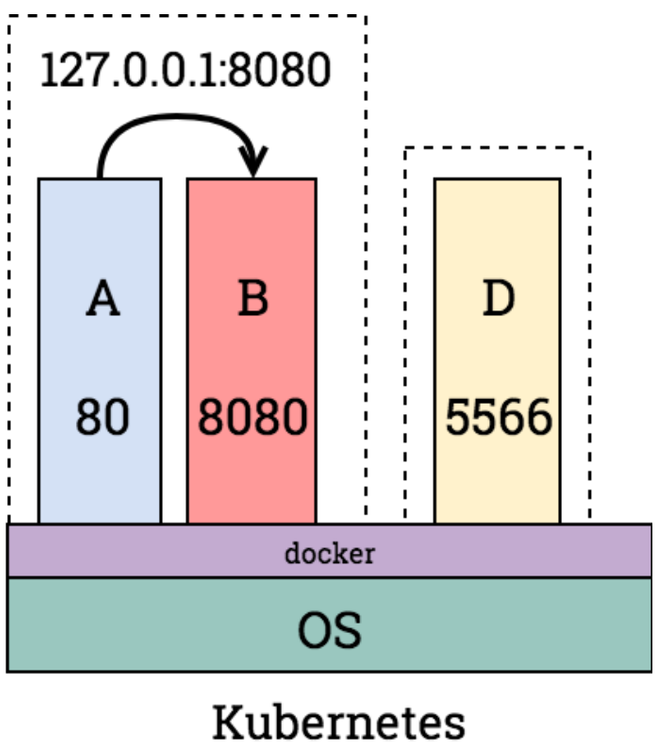
叢集管理指令應用

常用指令說明

- kubectl <verb> <resource> <optional>

```
Deploy Commands:
  rollout         Manage the rollout of a resource
  rolling-update  Perform a rolling update of the given ReplicationController
  scale           Set a new size for a Deployment, ReplicaSet, Replication Controller, or Job
  autoscale       Auto-scale a Deployment, ReplicaSet, or ReplicationController

Cluster Management Commands:
  top             Display Resource (CPU/Memory/Storage) usage.
  cordon          Mark node as unschedulable
  uncordon        Mark node as schedulable
  drain           Drain node in preparation for maintenance
  taint           Update the taints on one or more nodes

Troubleshooting and Debugging Commands:
  describe        Show details of a specific resource or group of resources
  logs            Print the logs for a container in a pod
  exec            Execute a command in a container
  port-forward    Forward one or more local ports to a pod
  proxy           Run a proxy to the Kubernetes API server
  cp              Copy files and directories to and from containers.

Advanced Commands:
  apply           Apply a configuration to a resource by filename or stdin
```





Pod 應用

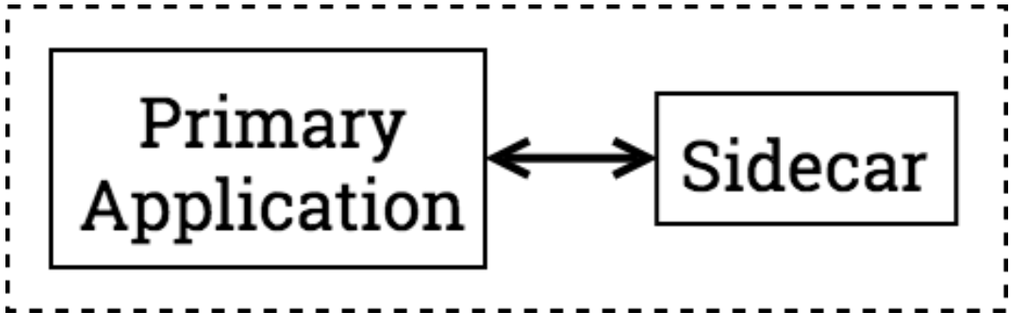
觀念說明

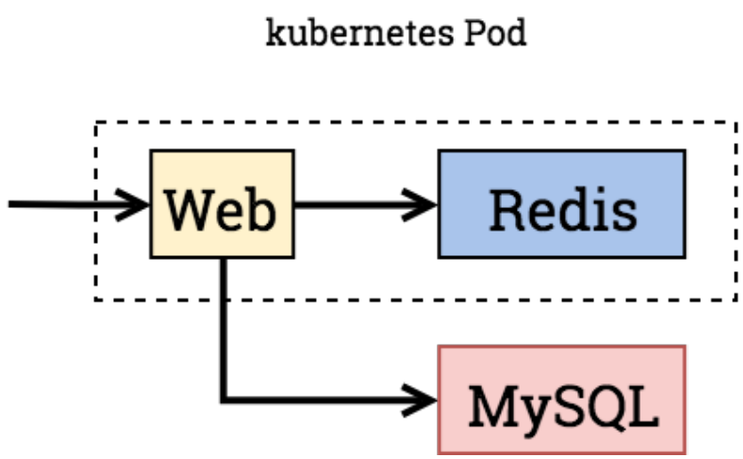
- Pod 內容器生死與共
  - Pod 內容器不會跨節點
- Pod 內容器共享同 Network Namespace
  - 獨立 IP、Port
  - 內部以 127.0.0.1 相互溝通
- 服務除錯、測試好幫手
  - kubectl logs -f -c
  - kubectl port-forward

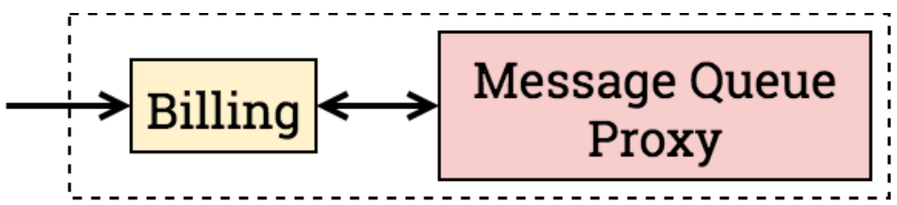
[1] https://tachingchen.com/tw/blog/kubernetes-pod/

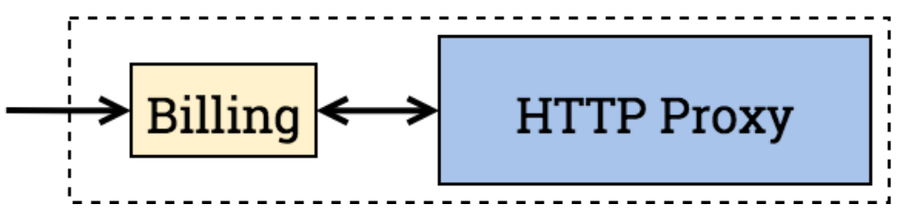
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod-1
  namespace: default
spec:
  containers:
  - image: nginx:latest
    imagePullPolicy: Always
    name: nginx
```

- kubectl logs -f -c <container> <pod>
- kubectl port-forward <pod> <local port>:<container port>
- kubectl run -i --rm --tty demo --image=life1347/ubuntu-kube-demo —restart=Never









Pod 應用

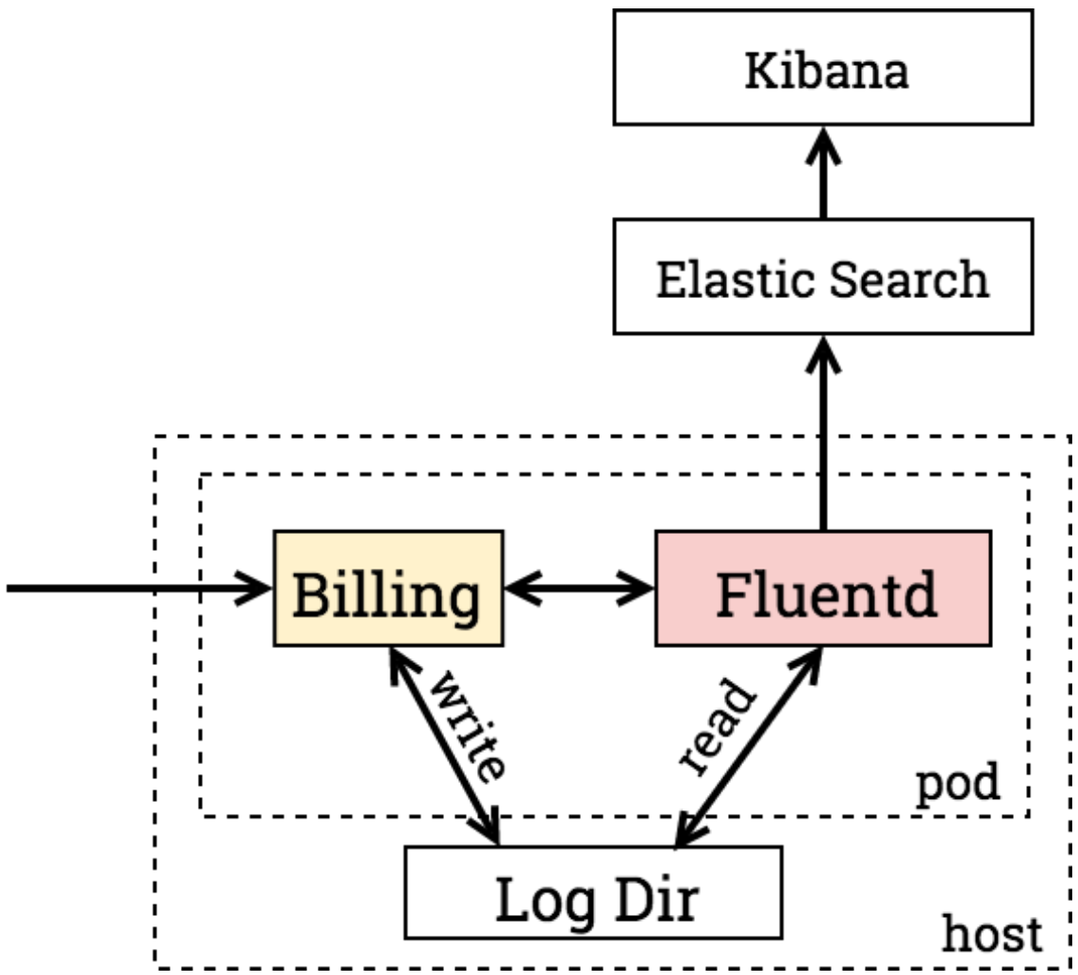
Sidecar Pattern 使用

- Sidecar Pattern [1]
  - 負責不包含在主應用的非核心功能
- 不更動主應用下增添新不同功能
  - 服務重用性增加，符合微服務概念
  - 靈活性提高
- 避免 Monolith Pod 出現
  - 1*主應用 + N*輔應用 (不要過多)
  - 緊密溝通的容器擺放在一起
  - 容器間沒有啟動先後順序
- 應用場景
  - 異質環境接口 (平台抽象化)
  - Remote Proxy
  - Logging

[1] https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar

Volume 應用

觀念說明



emptyDir

- 透過 Docker mount 將資料夾掛進去
  - 資訊分享與保存
- 考慮效能、服務特性選擇種類使用
- 種類
  - emptyDir: 跟 Pod 同生死
  - hostPath: 只存在該節點
  - nfs: host 需支援 NFS 功能
  - gitRepo: 適用 CI/CD
  - secret: 存放 cert、acc/pwd 等
  - configmap: Key-Value 設定值
- 應用場景
  - Distributed Log Collection
  - File Sharing
  - Sidecar Pattern

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod-1
  namespace: default
  labels:
    app: nginx
    backend: python
spec:
  containers:
  - image: nginx:latest
    imagePullPolicy: Always
    name: nginx
    volumeMounts:
    - mountPath: /share
      name: shared-volume
  - image: trinitronx/python-simplehttpserver
    name: python-server
    imagePullPolicy: Always
    volumeMounts:
    - mountPath: /share
      name: shared-volume
  volumes:
  - name: shared-volume
    emptyDir: {}
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    secret:
      secretName: mysecret
      items:
      - key: username
        path: my-group/my-username
```

Secret 應用

觀念說明

- 機密資訊與容器解耦合
- 以檔案形式掛載到 Pod 內
  - 當 secret 有更新時會跟著自動更新
  - Pod 需人為介入重啟或程式面要改動
- 以環境變數方式傳入 Pod
- 創建前須以 base64 將 data 編碼
- secret 其實不 secret
  - 有權限使用 kubectl 的人都可以看到
  - 每台上有執行 etcd 的節點都可以看到
  - 針對機器須做好安全管控 (非常基本)
  - 只掛載到需要使用的容器內
- 應用場景
  - 將機密資訊和容器映像檔分離
  - 統一的帳密管控

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: test
data:
  TEST_KEY: dGVzdHZhbHVl
type: Opaque
```

- kubectl create secret generic <secret> --from-literal=<key>="<value>"

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        # Define the environment variable
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              # The ConfigMap containing the va
              name: special-config
              # Specify the key associated wit
              key: special.how
  restartPolicy: Never
```

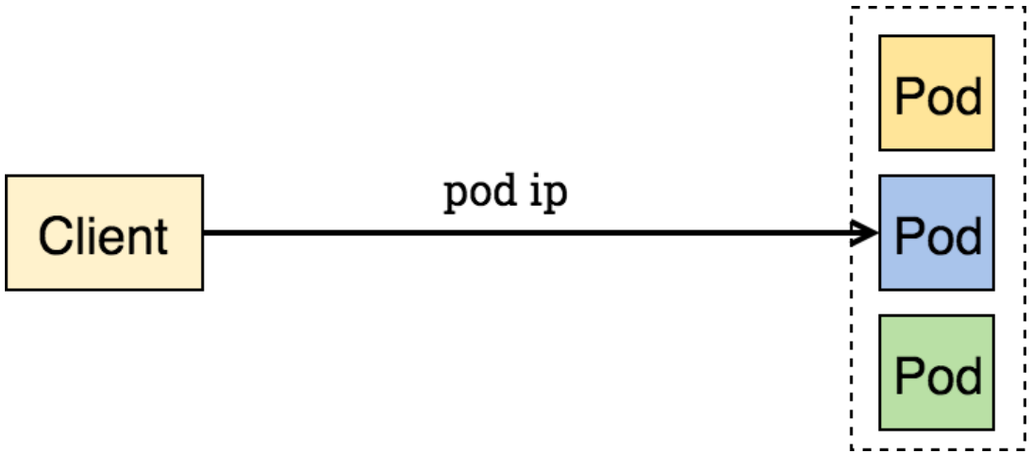
Configmap 應用

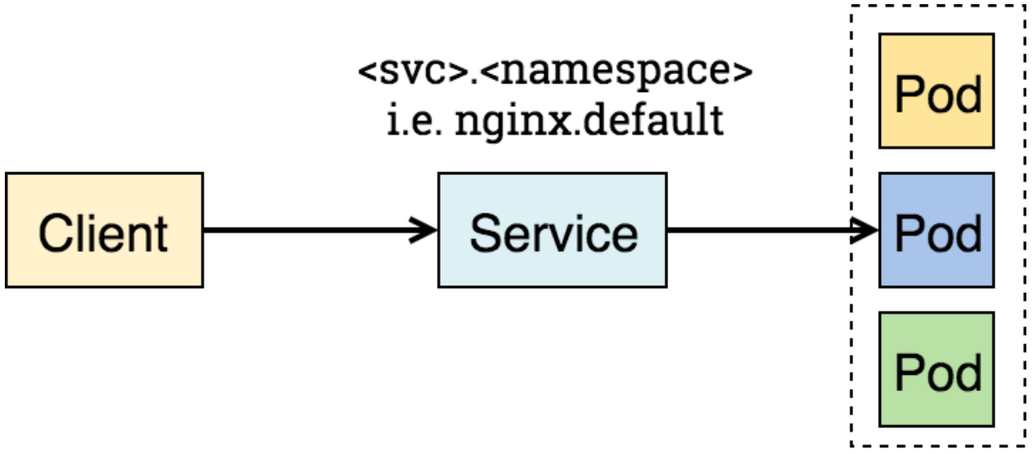
觀念說明

- 非機密資訊與容器解耦合
- 以檔案形式掛載到 Pod 內
  - 當 config 有更新時會跟著自動更新
  - Pod 需人為介入重啟或程式面要改動
- 以環境變數方式傳入 Pod
- 以 Key-Value 形式創建無需 base64
- configmap 和 secret 一樣很赤裸
  - 有權限使用 kubectl 的人都可以看到
  - 每台上有執行 etcd 的節點都可以看到
  - 針對機器須做好安全管控 (非常基本)
  - 只掛載到需要使用的容器內
- 應用場景
  - 將設定和容器映像檔分離
  - 集中式的設定管理

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: bar
data:
  TEST_KEY: TESTVALUE
```

- kubectl -n default create configmap <name> --from-literal=<key>=<value>





Service 應用

觀念說明

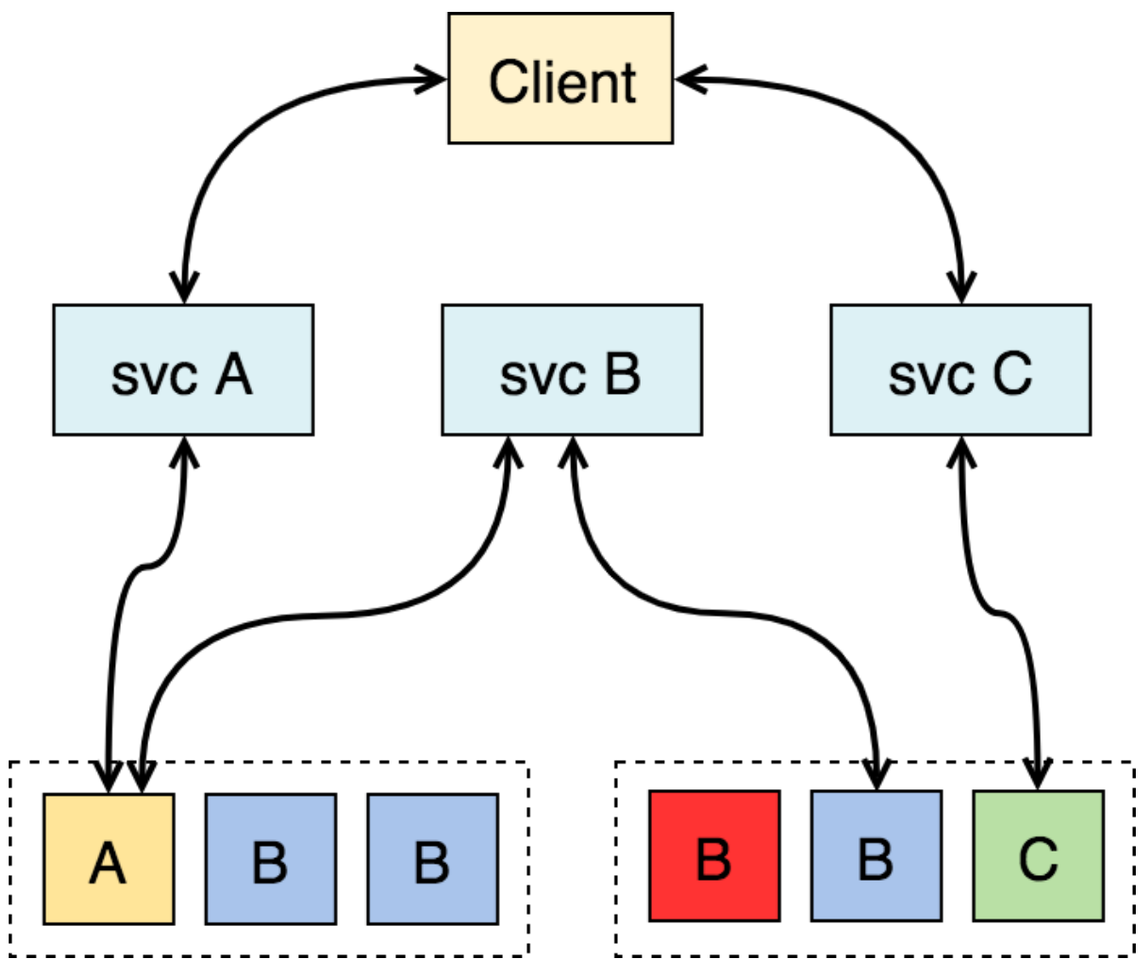
- 邏輯上的一群 Pod 以及存取他們的規則
  - Service 與 Pod 為鬆散耦合
  - 服務抽象化
- <svc>.<namespace>.svc.cluster.local
  - 存取服務的 DNS A records
  - 非 Pod IP，隱藏背後請求處理細節
- 除特別需求，否則不應該直接使用 Pod IP
  - 利用 DNS 降低程式複雜度
  - 高可用性、避免強耦合
  - K8S 有內建服務探索功能

[1] https://tachingchen.com/tw/blog/kubernetes-service/

Service 與 Label 應用

觀念說明 (cont.)

```
REDIS_1_PORT_6379_TCP=tcp://10.3.240.101:6379
REDIS_1_PORT_6379_TCP_PORT=6379
REDIS_1_SERVICE_PORT_REDIS=6379
REDIS_1_PORT=tcp://10.3.240.101:6379
REDIS_1_PORT_6379_TCP_PROTO=tcp
REDIS_1_SERVICE_HOST=10.3.240.101
REDIS_1_SERVICE_PORT=6379
REDIS_1_PORT_6379_TCP_ADDR=10.3.240.101
```



- 可透過 DNS 或環境變數取得服務資訊
- 底層實作利用 iptabels 做流量分散
  - 網路延遲問題
  - 請求可能會被送到不同 Pod、節點
  - 長時間下來才會是均勻分散

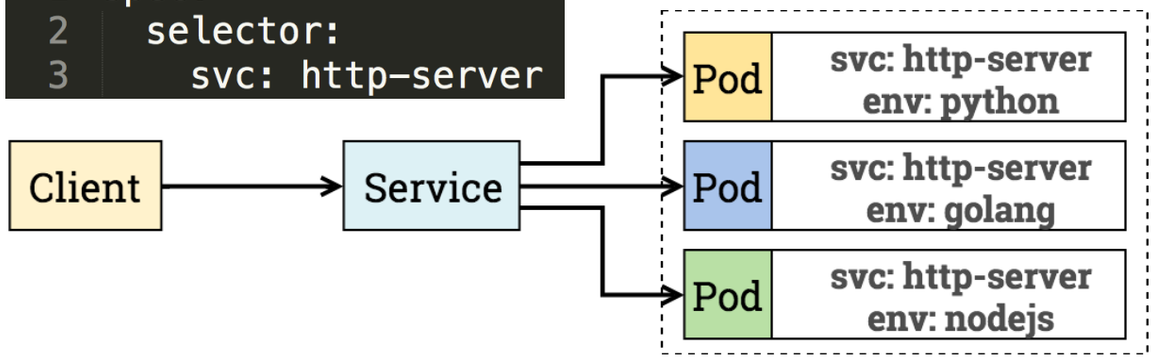
- kubectl expose pod <pod> —type=<svc type> —name=<svc name> --port <expose port> —target-port=<container port>

```
error: couldn't retrieve selectors via --selector flag or introspection: the pod has no labels and cannot be exposed
See 'kubectl expose -h' for help and examples.
```

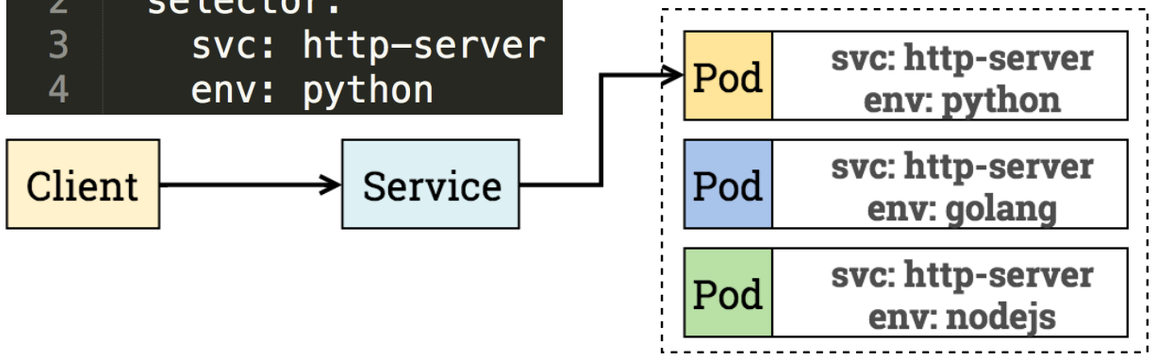
Service 與 Label 搭配應用

標籤組合，精密分流

```
1 spec:
2   selector:
3     svc: http-server
```




```
1 spec:
2   selector:
3     svc: http-server
4     env: python
```



- 過去要設定 LB 將流量分散，太過耗時
- 善用選擇器，精密控流
- 鬆散耦合 + 唯一標籤組合
  - 提高粒度，不影響服務高可用性
- 應用場景
  - 流量管控，可以隨時切分流量
  - 簡易 A/B 測試

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod-1
  namespace: default
  labels:
    app: nginx
spec:
  containers:
  - image: nginx:latest
    imagePullPolicy: Always
    name: nginx
```

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx
  name: nginx-svc
  namespace: default
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  sessionAffinity: None
  type: NodePort
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: gcr.io/google_containers/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

Service 與 Label 應用

探針實現服務探索，提高應用可用性

- Readiness: 是否準備接受流量
- Liveness: 服務是否正常運作
- 三種檢測方式: CMD、HTTP、TCP
- kubectl describe pod <pod>
- 應用場景
  - 避免 Service 把流量送到不健康 Pod
  - 自動修復，提高可用性

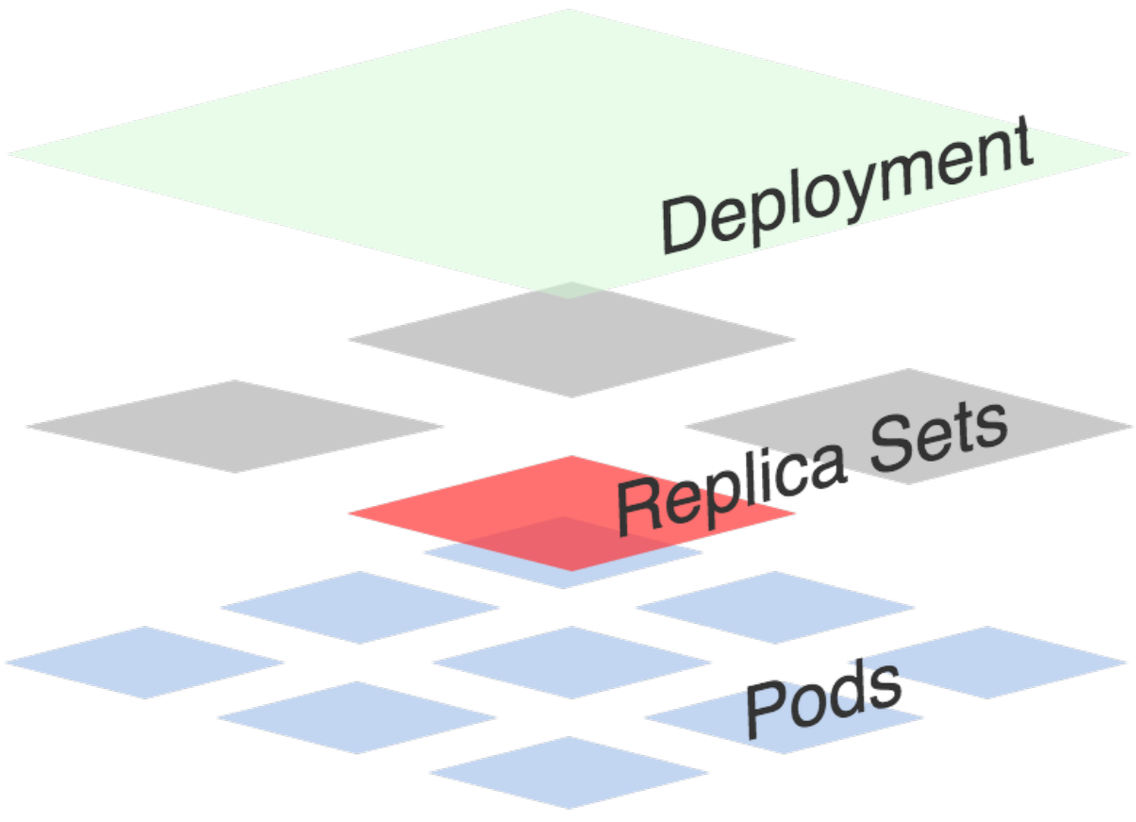
```
Events:
  Type     Reason
  ----     ------
  Normal   Scheduled
  Normal   SuccessfulMountVolume
  Normal   Pulled
  Normal   Created
  Normal   Started
  Warning  Unhealthy
  Normal   Pulling
  Normal   Killing
```

[1] https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod-2
  namespace: default
  labels:
    app: nginx
    backend: go
spec:
  containers:
  - image: nginx:latest
    imagePullPolicy: Always
    name: nginx
    livenessProbe:
      httpGet:
        path: /dummy-healthz-url
        port: 80
      initialDelaySeconds: 3
      periodSeconds: 3
```

Deployment 應用

觀念說明




```
Deployment: <name>
└ Replica Set: <name>-<rs>
    └ Pod: <name>-<rs>-<randomString>
```

- Scale by cloning
  - 在 Pod 之上更高層次的抽象化
- Pod 命名為隨機字串
- 降低人力需求與失誤
  - 便利升級、Pod 數量確保、自我修復
- 提供各種強大功能
  - 滾動升級 (Rolling update)
  - 回滾 (Rollback)
  - 暫停/繼續升級
- 應用場景
  - 適合無狀態服務使用
  - 服務升級、回滾

[1] https://tachingchen.com/tw/blog/kubernetes-rolling-update-with-deployment/

```yaml
# for old k8s cluster:
# apiVersion: extensions/v1beta1
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-1
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx-pod-1
      namespace: default
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: Always
        name: nginx
```

- kubectl run <depl name> --image=<img> --labels="<key>=<value>" --expose --port=<port>

```yaml
spec:
  replicas: 10
  selector:
    matchLabels:
      service: http-server
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  minReadySeconds: 5
```

```
20        <none>
21        <none>
23        kubectl set image deployment ad-targeter
24        kubectl set image deployment ad-targeter
25        kubectl set image deployment ad-targeter
26        kubectl set image deployment ad-targeter
27        kubectl set image deployment ad-targeter
28        kubectl set image deployment ad-targeter
29        kubectl set image deployment ad-targeter
30        kubectl edit deployment ad-targeter
31        kubectl edit deployment ad-targeter
```

Deployment 應用

觀念說明 (cont.)

- 不要隨意改動 selector 的值
  - 造成孤兒 RS, Pod 增加維運困擾
- 設定 Rolling Strategy，才支援滾動升級
  - minReadySeconds
  - maxSurge
  - maxUnavailable
- 利用 —record 紀錄每次操作
  - 使用 kubectl set image 更新 image，確保操作歷史清楚易懂
  - 設定 .spec.revisionHistoryLimit，保留定量 revision 即可

[1] https://tachingchen.com/tw/blog/kubernetes-rolling-update-with-deployment/

```yaml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 50
```

Horizontal Pod Autoscaler 應用

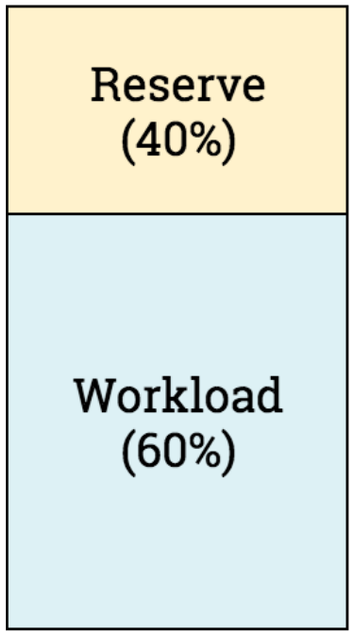
觀念說明

- 依據 Pod 負載進行 Deployment 伸縮擴展
- Pod 需要做資源管控，以計算使用量
- 預設每 30s 做 utilization 計算
- HPA 採保守機制
  - 開很快、關很慢
  - 避免顛簸流量造成 Pod 重複開關
- 可搭配 GKE Cluster Autoscaler
  - Pod 出現 Pending 狀態才會觸發
  - 但是該怎麼做才能確保服務正常？

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
[redacted]	Deployment/[redacted]	6% / 30%	20	120	20
[redacted]	Deployment/[redacted]	16% / 40%	3	15	3
[redacted]	Deployment/[redacted]	16% / 30%	50	250	50
[redacted]	Deployment/[redacted]	27% / 40%	5	20	5
[redacted]	Deployment/[redacted]	0% / 30%	30	120	30
[redacted]	Deployment/[redacted]	6% / 40%	10	30	10
[redacted]er	Deployment/[redacted]er	3% / 40%	20	30	20
[redacted]	Deployment/[redacted]	5% / 40%	10	60	10

Horizontal Pod AutoScaling 應用

一場價值十六萬的課 (教訓)



pod normal cpu usage: 100m cpu
server resource reserve: 60%
server core: 4
max Pod: 200

Q: pod request cpu:

4 * 1000m / 100m = 40 pod

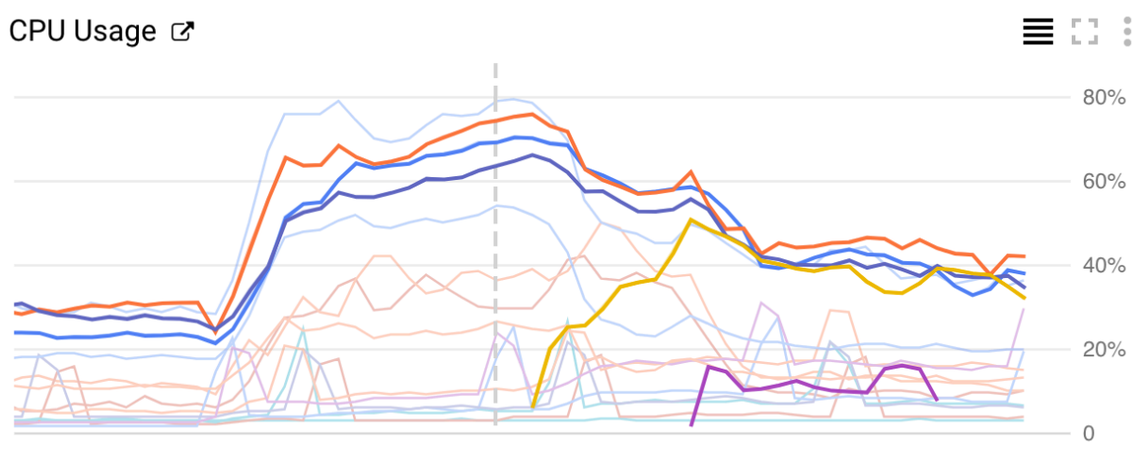
40 * 0.6 = 24 pod

4 * 1000m / 24 = 167m

167m = 100m (pod) + 67m (reserve) #

Q: max node when peak

167m * 200 / 4000m = 8.35 #



- K8S 會將節點所有資源拿來開 Pod (危險)
- 每個節點 K8S 元件會佔去 0.2 ~ 0.5 core
- 服務穩定關鍵
  - 節點核心數約在 4 ~ 6 core 為佳
    - 擴展快、元件資源佔比小
  - 總節點負荷在六或七成為佳
    - 保留系統資源與應付瞬間突波
  - 單一 Pod 資源不宜過多
    - 難分配到其他節點 (容易 Pending)
    - 以水平擴展為主要手段
- 面對穩定流量
  - 高 Threshold
- 面對瞬間不穩定流量
  - 低 Threshold，讓 HPA 容易被觸發

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  targetCPUUtilizationPercentage: 50
```

- kubectl autoscale deployment <depl> --cpu-percent=<percentage> --min=<min> --max=<max>

Q&A