Kubernetes 101

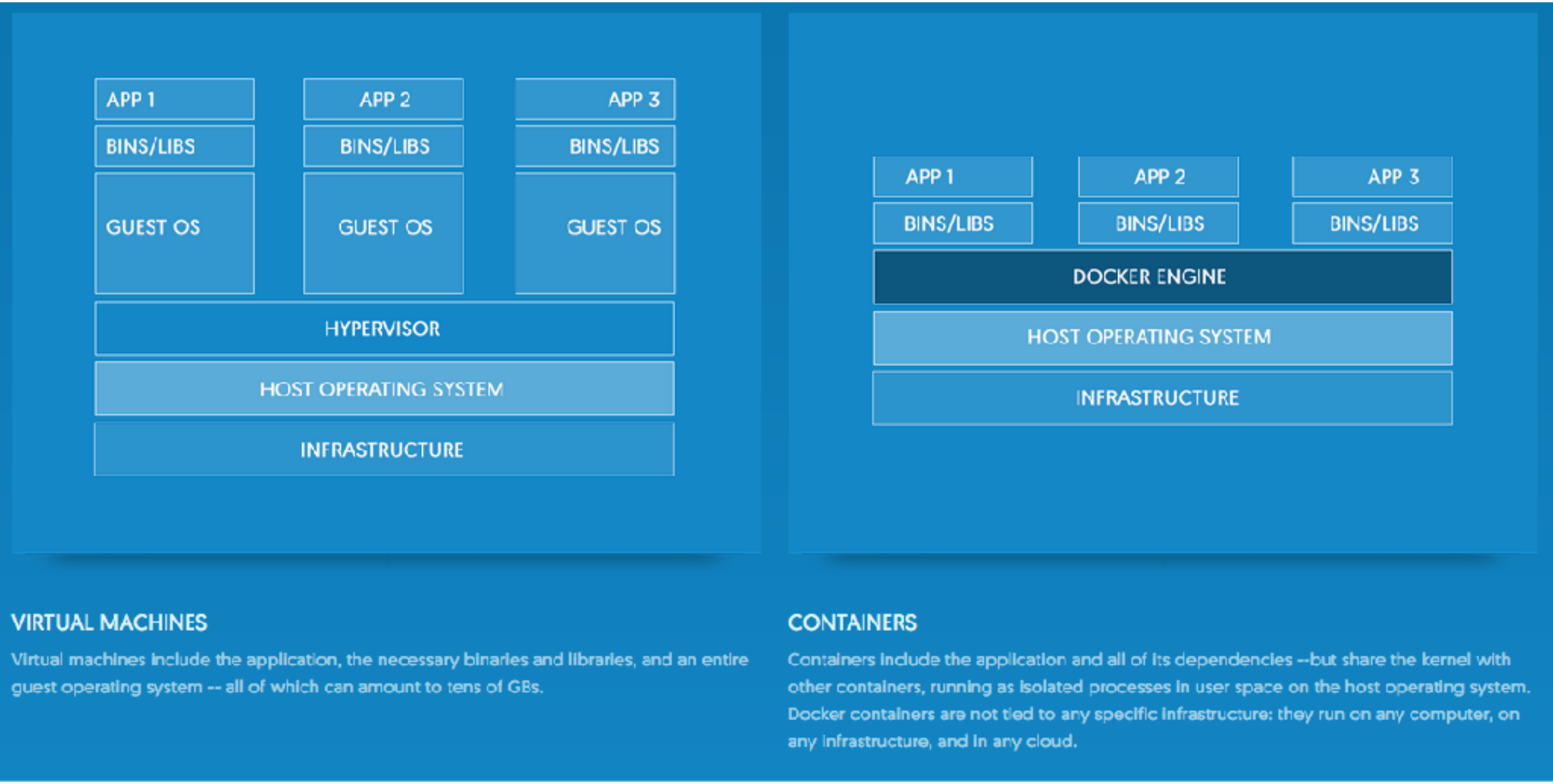
Tommy Chen @ TechCCU 2016

Tommy Chen

- a.k.a. SkyArrow
- Dcard / Full Stack Developer
- @tommy351

1

Recap Docker



VIRTUAL MACHINES

Virtual machines include the application, the necessary binaries and libraries, and an entire guest operating system -- all of which can amount to tens of GBs.

CONTAINERS

Containers include the application and all of its dependencies --but share the kernel with other containers, running as isolated processes in user space on the host operating system. Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud.

https://www.docker.com/what-docker

Dockerfile

```
FROM scratch
COPY app /
EXPOSE 4000
CMD ["/app"]
```

Build

```
$ docker build --tag tommy351/my-server .
Sending build context to Docker daemon 3.943 MB
Step 1 : FROM scratch
 --->
Step 2 : COPY app /
 ---> 4c25e1b50576
Removing intermediate container 22b9992b6f64
Step 3 : EXPOSE 4000
 ---> Running in 975456111f69
 ---> 03095e7edf87
Removing intermediate container 975456111f69
Step 4 : CMD /app
 ---> Running in 9bc9777f565e
 ---> f41f9b6ad7d0
Removing intermediate container 9bc9777f565e
Successfully built f41f9b6ad7d0

$ docker images
REPOSITORY           TAG                 IMAGE ID            CREATED
SIZE
tommy351/my-server   latest              f41f9b6ad7d0        6 seconds ago
3.939 MB
```

Run

```
$ docker run -p 4000:4000 -d tommy351/my-server
9eb06be69a532974e811aa89fdf44805fe63616e684c5c38e7fc
7d766b43dc50

$ docker ps
CONTAINER ID        IMAGE                COMMAND
CREATED             STATUS              PORTS
NAMES
9eb06be69a53        tommy351/my-server   "/app"
21 seconds ago      Up 20 seconds
0.0.0.0:4000->4000/tcp   fervent_heyrovsky

$ curl http://localhost:4000
Hello world
```

Push & Pull

```
$ docker push tommy351/my-server
The push refers to a repository [docker.io/tommy351/my-server]
8413560c067f: Pushed
latest: digest:
sha256:becf447f4dc2a950a10330a86d42209ed42c3a4f7d3636fd50cdd5eeaf3589
52 size: 528

$ docker pull tommy351/my-server
Using default tag: latest
latest: Pulling from tommy351/my-server
432847f70861: Already exists
Digest:
sha256:becf447f4dc2a950a10330a86d42209ed42c3a4f7d3636fd50cdd5eeaf3589
52
Status: Downloaded newer image for tommy351/my-server:latest

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
tommy351/my-server  latest              f41f9b6ad7d0        2
minutes ago      3.939 MB
```

2

What is Kubernetes?

Why Kubernetes?

- Docker 只能用在單機
- 必須要有軟體來處理多機之間的關係
    - Docker Swarm
    - Kubernetes
    - Marathon (Mesos + DCOS)
    - AWS ECS

功能

- 處理多機之間的關係
- 健康檢查
- 自動水平擴展
- 負載平衡
- 漸進更新（Rolling update）
- 服務探索
- 設定管理

3

Get Ready

Minikube

在本機架一個單節點的 Kubernetes cluster，方便進行測試。
https://github.com/kubernetes/minikube

```
$ curl -Lo minikube https://storage.googleapis.com/
minikube/releases/v0.10.0/minikube-darwin-amd64
$ chmod +x minikube
$ sudo mv minikube /usr/local/bin/
```

kubectl

安裝 Kubernetes 的管理工具。

透過 gcloud 安裝：

https://cloud.google.com/sdk/downloads

```
$ curl https://sdk.cloud.google.com | bash
$ exec -l $SHELL
$ gcloud components install kubectl
```

從 binary 安裝：

https://coreos.com/kubernetes/docs/latest/configure-kubectl.html

```
$ curl -O https://storage.googleapis.com/kubernetes-release/release/v1.3.6/bin/darwin/amd64/kubectl
$ chmod +x kubectl
$ mv kubectl /usr/local/bin/kubectl
```

啟動 minikube

```
$ minikube start
Starting local Kubernetes cluster...
Kubectl is now configured to use the cluster.
```

Hello World

```
$ kubectl run my-server --image tommy351/my-server --port 4000
deployment "my-server" created

$ kubectl get deployment
NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-server   1         1         1            1           17s

$ kubectl get replicaset
NAME                   DESIRED   CURRENT   AGE
my-server-1549122061   1         1         20s

$ kubectl get pod
NAME                         READY     STATUS    RESTARTS   AGE
my-server-1549122061-9io6v   1/1       Running   0          23s

$ kubectl expose deployment my-server --type NodePort
service "my-server" exposed

$ kubectl get service
NAME         CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
kubernetes   10.0.0.1     <none>        443/TCP    22m
my-server    10.0.0.133   <nodes>       4000/TCP   5s

$ curl $(minikube service my-server --url)
Hello world
```

剛剛做了什麼？

- kubectl run
    - 建立 Deployment → ReplicaSet → Pod
- kubectl expose
    - 建立 Service
    - 暴露服務給外界存取

4

Basic Concepts

常用指令

建立資源

```
$ kubectl create -f <filename>
```

顯示資源列表，type 可以是 pod, rc, service, etc.

```
$ kubectl get <type>
```

取得單一資源的資訊

```
$ kubectl get <type> <name>
```

刪除單一資源

```
$ kubectl delete <type> <name>
$ kubectl delete -f <filename>
```

在 pod 裡執行指令 (相當於 docker exec)

```
$ kubectl exec -it <pod> <command>
```

常用指令

顯示 pod 紀錄 (相當於 docker logs)

```
$ kubectl logs <pod>
```

更新資源

```
$ kubectl apply -f <filename>
```

顯示 kubernetes 的所有事件

```
$ kubectl get events
```

開啟 Kubernetes proxy，預設在 localhost:8001，
localhost:8001/ui 是 dashboard

```
$ kubectl proxy
```

連接埠轉發 (e.g. kubectl port-forward postgres 5433:5432)

```
$ kubectl port-forward <pod> <local port>:<remote port>
```

Pod

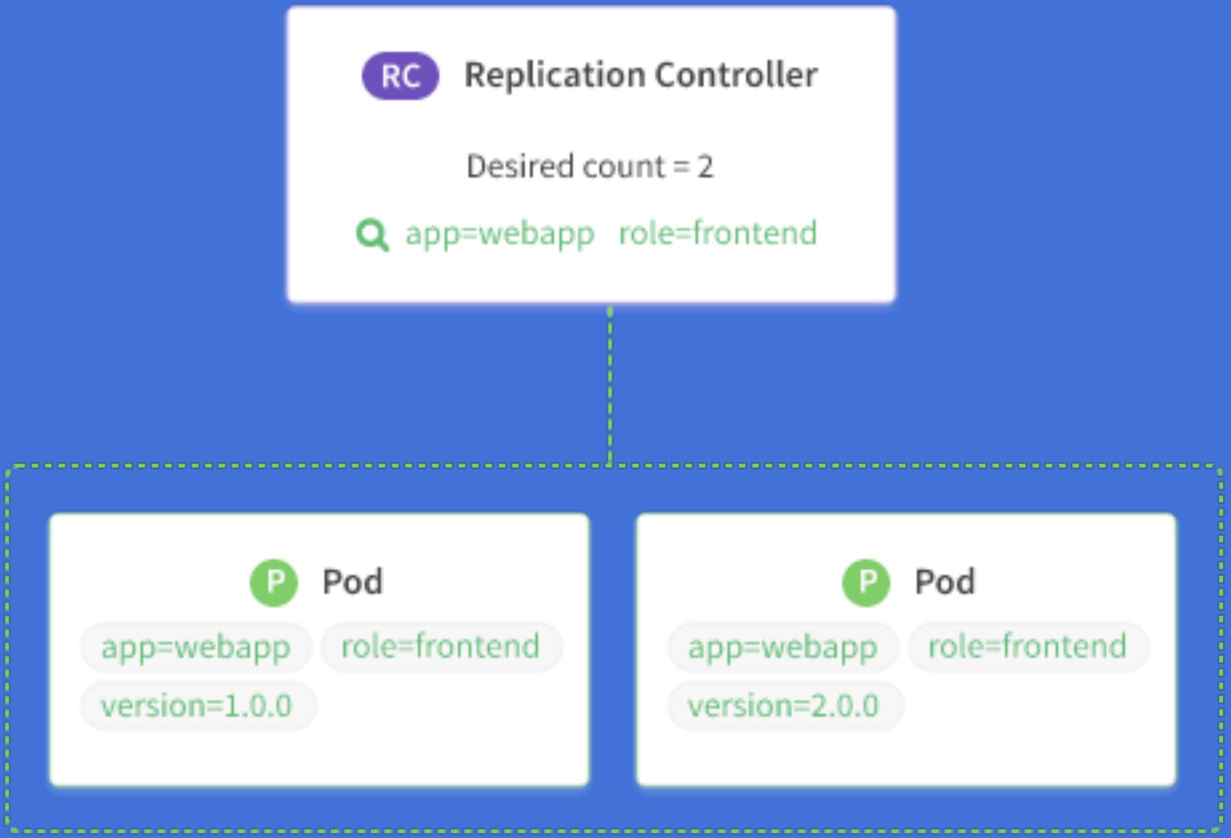
- 一個以上容器的集合
- Kubernetes 中基本的執行單位
- 每個 Pod 都會配一個 Cluster IP
- 死掉的話不會重啟，必須透過 Replication controller 或 Replica set 管理

Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-server
  labels:
    name: my-server
spec:
  containers:
    - name: my-server
      image: tommy351/my-server
      ports:
        - containerPort: 4000
```

Replication Controller

- 管理 Pod 的生命週期
- 確保指定數量的 Pod 在執行



https://coreos.com/kubernetes/docs/latest/replication-controller.html

Replication Controller

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: my-server
spec:
  replicas: 2
  selector:
    name: my-server
  template:
    metadata:
      name: my-server
      labels:
        name: my-server
    spec:
      containers:
        - name: my-server
          image: tommy351/my-server
          ports:
            - containerPort: 4000
```

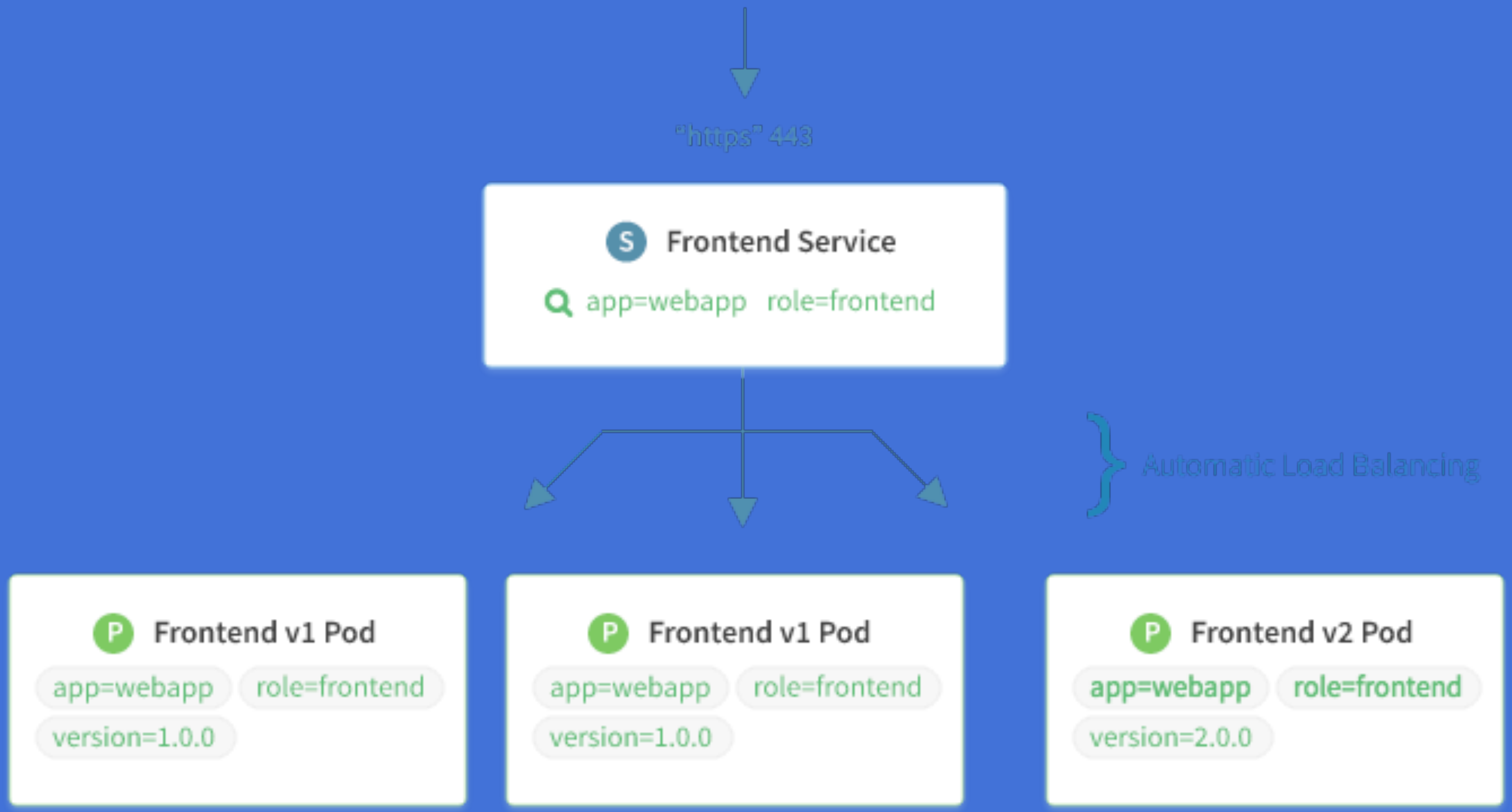
Pod 數量

Selector 會選中 Label 符合的 Pod

和剛剛的 Pod 內容一樣

Service

- 把指定的 Pod 暴露出來讓外部存取
- 如果有多個 Pod 的話可進行負載平衡



https://coreos.com/kubernetes/docs/latest/services.html

Service

- 三種服務類型：
  - ClusterIP
    - 預設的服務類型，只有 Cluster 內部能存取
  - NodePort
    - 暴露到某個 port
  - LoadBalancer
    - 暴露到外部 IP（需要雲端服務配合）

Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-server
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 4000
  selector:
    name: my-server
```

4000 (pod) → 80 (service)

透過 Pod labels 選擇指定的 Pod

服務探索

- Kubernetes 提供了兩種方式讓 Cluster 內部的 Pod 可以互相找到對方
- DNS
    - Kubernetes 內建了 kube-dns，可以透過 Service name 存取
- 環境變數

服務探索

例：有一個名為 redis 的服務

DNS

```
redis:6379
```

環境變數

```
REDIS_SERVICE_HOST=10.0.0.11
REDIS_SERVICE_PORT=6379
REDIS_PORT=tcp://10.0.0.11:6379
REDIS_PORT_6379_TCP=tcp://10.0.0.11:6379
REDIS_PORT_6379_TCP_PROTO=tcp
REDIS_PORT_6379_TCP_PORT=6379
REDIS_PORT_6379_TCP_ADDR=10.0.0.11
```

5

Pod Management

資料持久化

- Container 內的資料必須存在外部，否則 Container 終止後就會被刪除
- emptyDir
    - Pod 啟動時分配空間，終止後即被刪除
- hostPath
    - 掛載主機的路徑

資料持久化

- gcePersistentDisk, awsElasticBlockStore, AzureFileVolume, AzureDiskVolume
    - GCE, AWS, Azure 提供的磁碟
- nfs
- persistentVolumeClaim
- http://kubernetes.io/docs/user-guide/volumes/

資料持久化

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: redis
  labels:
    name: redis
spec:
  containers:
    - name: redis
      image: redis:3.2
      ports:
        - containerPort: 6379
      volumeMounts:
        - mountPath: /data
          name: data
  volumes:
    - name: data
      emptyDir: {}
```

設定管理

- 環境變數
- ConfigMap
- Secret
- 利用 ConfigMap 和 Secret 的方式：
  - 掛載磁碟
  - 環境變數

環境變數

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-server
  labels:
    name: my-server
spec:
  containers:
    - name: my-server
      image: tommy351/my-server
      ports:
        - containerPort: 4000
      env:
        - name: REDIS_ADDRESS
          value: redis:6379
        - name: SERVER_HOST
          value: :4000
```

REDIS_ADDRESS=“redis:6379”
SERVER_HOST=“:4000”

ConfigMap

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: redis
data:
  redis.conf: |
    maxmemory 2mb
    maxmemorypolicy allkeyslru
```

Secret

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

data 必須經過 base64 編碼
password=base64(“pass”)
username=base64(“admin”)

掛載 ConfigMap / Secret

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: redis
  labels:
    name: redis
spec:
  containers:
    - name: redis
      image: redis:3.2
      ports:
        - containerPort: 6379
      volumeMounts:
        - mountPath: /usr/local/etc/redis
          name: config
        - mountPath: /srv/redis/secret
          name: secret
  volumes:
    - name: config
      configMap:
        name: redis
    - name: secret
      secret:
        secretName: redis
```

/usr/local/etc/redis/redis.conf
maxmemory 2mb
maxmemorypolicy allkeyslru

/srv/redis/secret/username
admin

/srv/redis/secret/password
pass

從 ConfigMap / Secret 設定環境變數

```
apiVersion: v1
kind: Pod
metadata:
  name: redis
  labels:
    name: redis
spec:
  containers:
    - name: redis
      image: redis:3.2
      ports:
        - containerPort: 6379
      env:
        - name: REDIS_CONF
          valueFrom:
            configMapKeyRef:
              name: redis
              key: redisconf
        - name: REDIS_USERNAME
          valueFrom:
            secretKeyRef:
              name: redis
              key: username
        - name: REDIS_PASSWORD
          valueFrom:
            secretKeyRef:
              name: redis
              key: password
```

REDIS_CONF
maxmemory 2mb
maxmemorypolicy allkeyslru

REDIS_USERNAME
admin

REDIS_PASSWORD
pass

健康檢查

- 檢查 Pod 是否正常執行
- 兩種檢查時期：
  - livenessProbe
    - 檢查 Pod 是否還活著，每隔一段時間就會執行
  - readinessProbe
    - 確保 Pod 已經開啟並能正常運作，在 Pod 進入 Running 狀態前執行

健康檢查

- 三種檢查方式：
  - exec
    - 執行指令並檢查回傳值是否為 0
  - httpGet
    - 發送 HTTP 請求，並檢查回傳狀態為 2xx
  - tcpSocket
    - 確保 TCP socket 開啟

健康檢查

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-server
  labels:
    name: my-server
spec:
  containers:
    - name: my-server
      image: tommy351/my-server
      ports:
        - containerPort: 4000
      livenessProbe:
        initialDelaySeconds: 15
        periodSeconds: 10
        httpGet:
          path: /
          port: 4000
        timeoutSeconds: 5
```

Pod 啟動 15 秒後
每隔 10 秒
戳 http://localhost:4000/
如果回傳狀態不是 2xx
或超過 5 秒沒有回應
即判斷 Pod 不健康

Rolling Update

- 從舊的 image 漸進更新到新的 image
    - 建立新的 Replication controller
    - 用新的 image 開啟 container 並漸漸消滅舊的 container
    - 完成後重新命名新的 Replication controller 並消滅舊的

Rolling Update

```
$ kubectl rolling-update my-server –image=tommy351/my-server:v2
Created my-server-bc33bf3389402cd633ec4573695db4dc
Scaling up my-server-bc33bf3389402cd633ec4573695db4dc from 0 to
2, scaling down my-server from 2 to 0 (keep 2 pods available,
don't exceed 3 pods)
Scaling my-server-bc33bf3389402cd633ec4573695db4dc up to 1
Scaling my-server down to 1
Scaling my-server-bc33bf3389402cd633ec4573695db4dc up to 2
Scaling my-server down to 0
Update succeeded. Deleting old controller: my-server
Renaming my-server-bc33bf3389402cd633ec4573695db4dc to my-server
replicationcontroller "my-server" rolling updated

$ kubectl get rc -o wide
NAME        DESIRED   CURRENT   AGE       CONTAINER(S)
IMAGE(S)                SELECTOR
my-server   2         2         23s       my-server
tommy351/my-server:v2
deployment=bc33bf3389402cd633ec4573695db4dc,name=my-server
```

Rollback

Rolling update 途中有可能因為 crash 或中斷而失敗，可利用 —rollback 回退到上一個版本

```
$ kubectl rolling-update my-server —rollback
```

資源管理

- 可設定 CPU、記憶體限制
- requests
  - 必須有足夠資源才能啟動 Pod，否則會在 Pending 狀態等待
- limits
  - 超過資源上限的話，會使 Pod 終止
  - 如果沒有設定 requests 的話，requests = limit

資源管理

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-server
  labels:
    name: my-server
spec:
  containers:
    - name: my-server
      image: tommy351/my-server
      ports:
        - containerPort: 4000
      resources:
        requests:
          cpu: 200m
          memory: 100Mi
        limits:
          cpu: 500m
          memory: 200Mi
```

requests.cpu = 0.2 core
requests.memory = 100MB

limits.cpu = 0.5 core
limits.memory = 200MB

水平擴展

- 手動擴展
- 自動擴展
    - Pod 必須設定 CPU requests
    - 根據 CPU 使用量

手動擴展

```
$ kubectl scale rc my-server –replicas=6
replicationcontroller "my-server" scaled

$ kubectl get rc
NAME        DESIRED   CURRENT   AGE
my-server   6         6         1m

$ kubectl get pod
NAME              READY     STATUS    RESTARTS    AGE
my-server-54m5f   1/1       Running   0           26s
my-server-9qxko   1/1       Running   0           26s
my-server-e9bch   1/1       Running   0           26s
my-server-tzjlm   1/1       Running   0           26s
my-server-yo6j8   1/1       Running   0           1m
my-server-zeh6e   1/1       Running   0           1m
```

自動擴展

```
$ kubectl autoscale rc my-server –cpu-percent=50 –
min=1 –max=10
replicationcontroller "my-server" autoscaled
```

HorizontalPodAutoscaler

```
apiVersion: extensions/v1beta1
kind: HorizontalPodAutoscaler
metadata:
  name: my-server
spec:
  scaleRef:
    kind: ReplicationController
    name: my-server
    subresource: scale
  minReplicas: 1
  maxReplicas: 10
  cpuUtilization:
    targetPercentage: 50
```

A

Appendix
Google Cloud

Google Container Engine

- 如果你懶得自己架 Kubernetes cluster 的話，Google Container Engine 大概是最輕鬆簡單的 solution
- Load balancer
- Cloud Logging
- Private Docker Registry
- Cloud Monitoring

Ingress

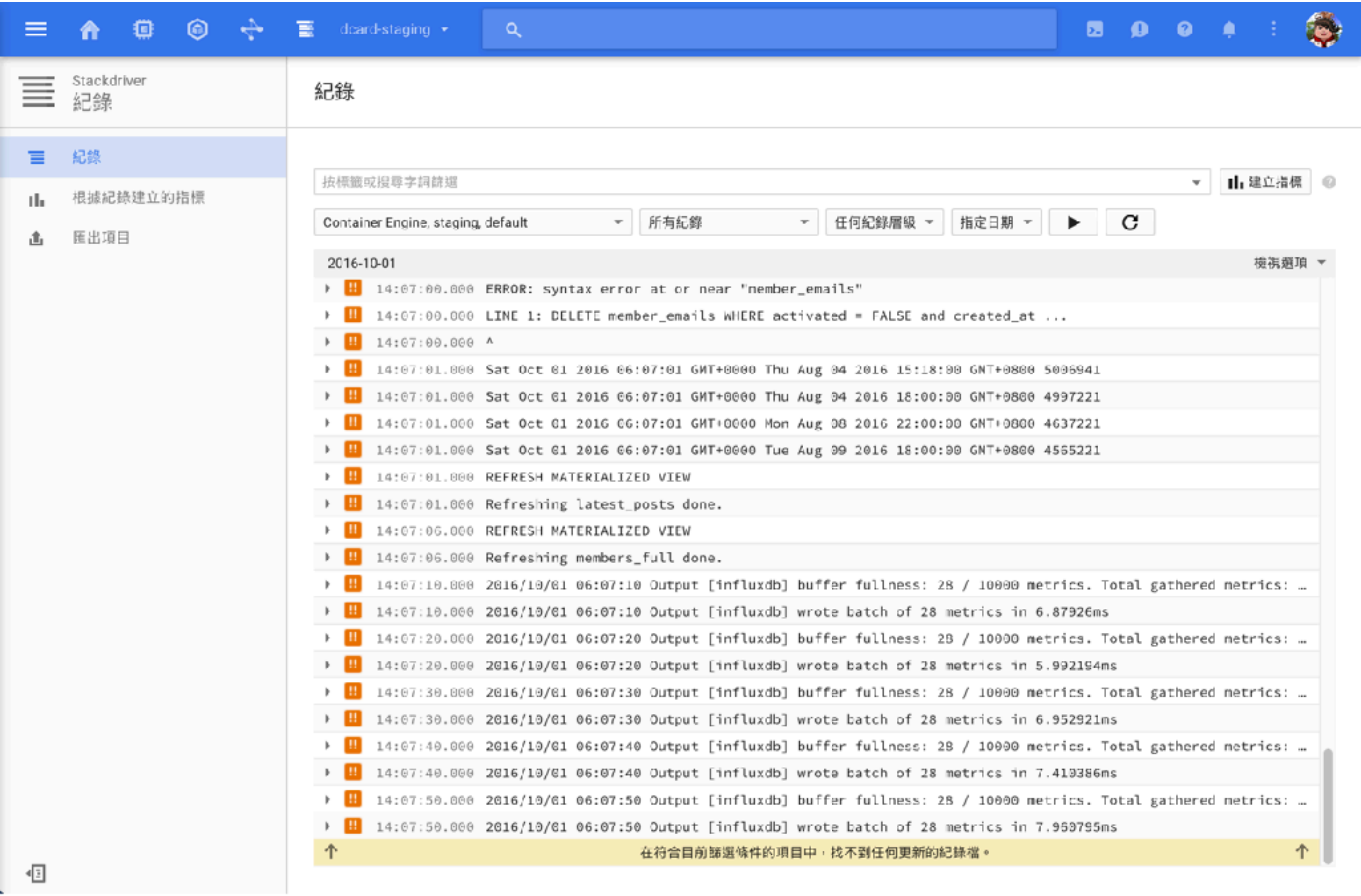
- 自動在 Google Cloud 上建立 Load balancer
- 必須使用 NodePort service
- Google Cloud 的 Load balancer 也有健康檢查，但是和 Kubernetes 不互通，要另外設定
- 必須暴露 30000-32767 TCP port（NodePort 的範圍）才能讓 Google Cloud 健康檢查

Ingress

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: s1
          servicePort: 80
      - path: /bar
        backend:
          serviceName: s2
          servicePort: 80
```

Cloud Logging

- 不用設定
- 所有 Pod 的紀錄都會自動送到 Google Cloud



```
2016-10-01
14:07:00.000 ERROR: syntax error at or near "member_emails"
14:07:00.000 LINE 1: DELETE member_emails WHERE activated = FALSE and created_at ...
14:07:00.000 ^
14:07:01.000 Sat Oct 01 2016 06:07:01 GMT+0000 Thu Aug 04 2016 15:18:00 GMT+0800 5006941
14:07:01.000 Sat Oct 01 2016 06:07:01 GMT+0000 Thu Aug 04 2016 18:00:00 GMT+0800 4997221
14:07:01.000 Sat Oct 01 2016 06:07:01 GMT+0000 Mon Aug 08 2016 22:00:00 GMT+0800 4637221
14:07:01.000 Sat Oct 01 2016 06:07:01 GMT+0000 Tue Aug 09 2016 18:00:00 GMT+0800 4565221
14:07:01.000 REFRESH MATERIALIZED VIEW
14:07:01.000 Refreshing latest_posts done.
14:07:06.000 REFRESH MATERIALIZED VIEW
14:07:06.000 Refreshing members_full done.
14:07:10.000 2016/10/01 06:07:10 Output [influxdb] buffer fullness: 28 / 10000 metrics. Total gathered metrics: ...
14:07:10.000 2016/10/01 06:07:10 Output [influxdb] wrote batch of 28 metrics in 6.87926ms
14:07:20.000 2016/10/01 06:07:20 Output [influxdb] buffer fullness: 28 / 10000 metrics. Total gathered metrics: ...
14:07:20.000 2016/10/01 06:07:20 Output [influxdb] wrote batch of 28 metrics in 5.992194ms
14:07:30.000 2016/10/01 06:07:30 Output [influxdb] buffer fullness: 28 / 10000 metrics. Total gathered metrics: ...
14:07:30.000 2016/10/01 06:07:30 Output [influxdb] wrote batch of 28 metrics in 6.952921ms
14:07:40.000 2016/10/01 06:07:40 Output [influxdb] buffer fullness: 28 / 10000 metrics. Total gathered metrics: ...
14:07:40.000 2016/10/01 06:07:40 Output [influxdb] wrote batch of 28 metrics in 7.419386ms
14:07:50.000 2016/10/01 06:07:50 Output [influxdb] buffer fullness: 28 / 10000 metrics. Total gathered metrics: ...
14:07:50.000 2016/10/01 06:07:50 Output [influxdb] wrote batch of 28 metrics in 7.959795ms
```

在符合目前篩選條件的項目中，找不到任何更新的紀錄檔。

Thanks!