

Red Hat OpenShift 4

OpenShift AP 開發進階

Michael Suen
Sr. Solution Architect , Red Hat
2024/03/28



前言

[https://tinyurl.com/
zb4yd2nz](https://tinyurl.com/zb4yd2nz)

Recap

- 容器基本概念: 什麼是容器(Container)、Containerfile(Dockerfile)、Podman。
- Kubernetes 開發基本概念: Kubernetes 基本元件(Pod、Deployment、Service、Route、Namespace、PVC)。
- OpenShift 開發基本概念: Operator & OperatorHub。
- 雲原生開發原則: 服務拆分、服務溝通、資料設計、可觀察性、可部署性。
- CI/CD 基本概念: OpenShift Pipeline - Tekton

建議延伸閱讀資料

O'REILLY®

Kubernetes Patterns

Reusable Elements for Designing
Cloud Native Applications

Second
Edition

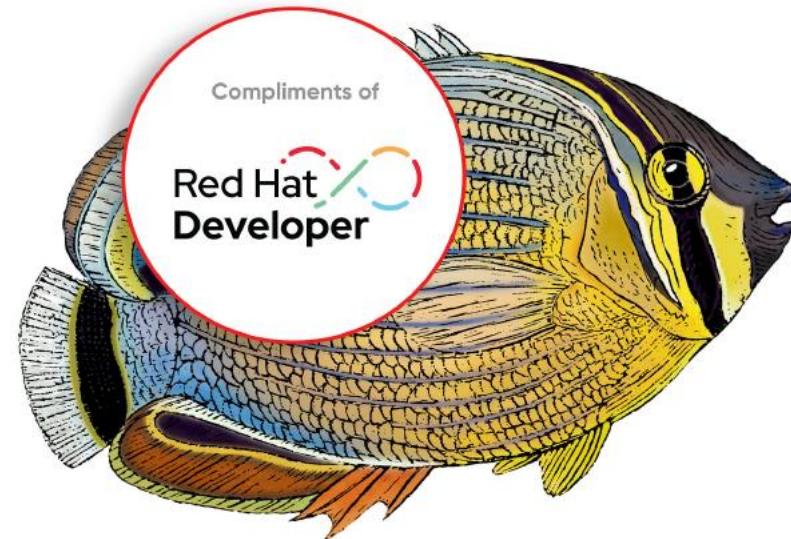


Bilgin Ibryam &
Roland Huß
Foreword by Brendan Burns

O'REILLY®

Operating OpenShift

An SRE Approach to Managing Infrastructure

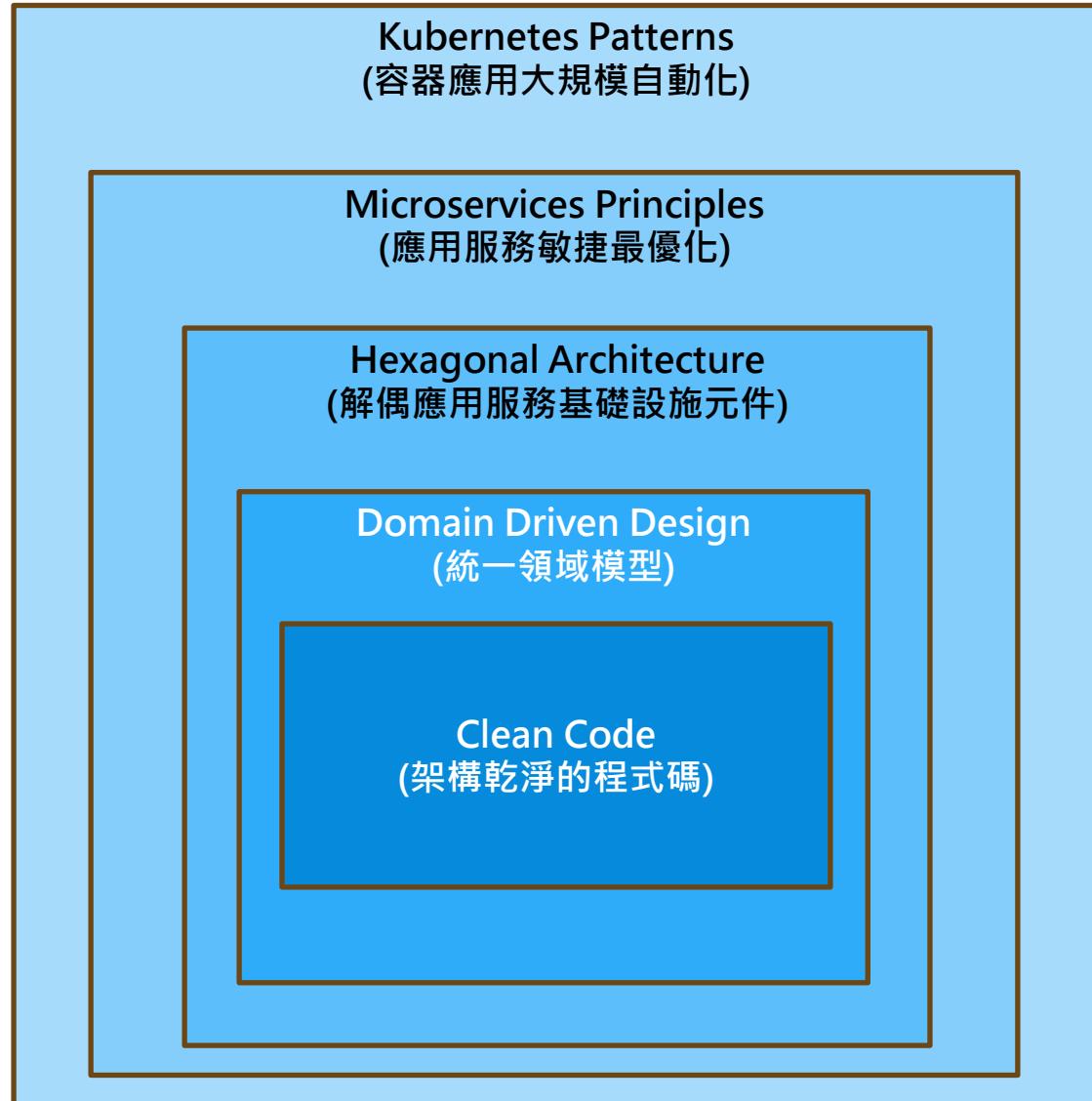


Rick Rackow
& Manuel Dewald



雲原生應用之路

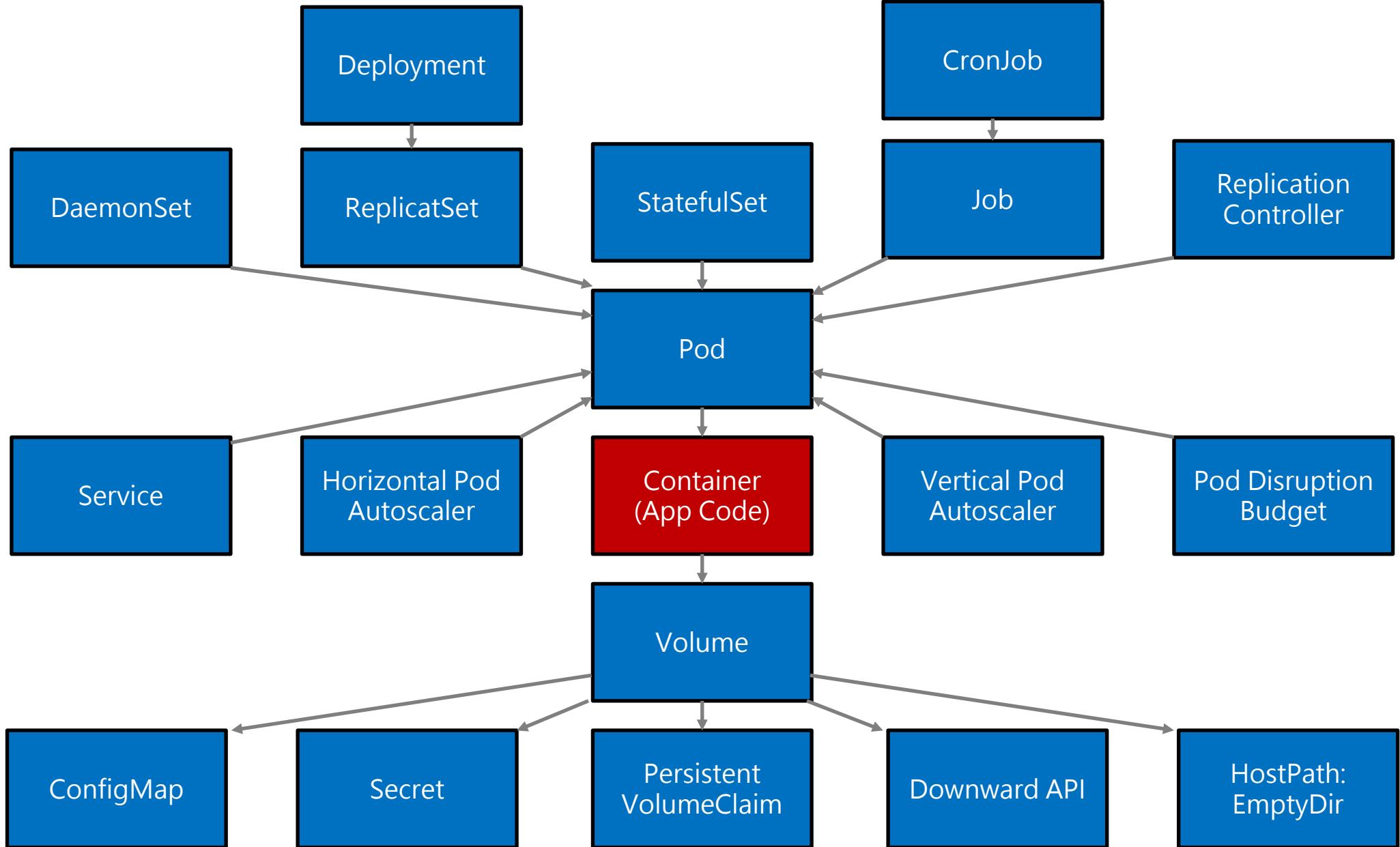
- Clean Code: 無關底層架構，應用程式面對變更及長期維運架構最優化。
- Domain Driven Design: 讓應用程式更貼近現實需求。
- Hexagonal Architecture: 又稱整潔架構，將核心邏輯和周邊基礎設施整合解偶的程式架構。
- Microservices Principles: 圍繞著 12 factors APP 原則建立應用服務。
- Kubernetes Patterns: 基於容器架構最大化微服務自動化維運設計



本地和分散式概念對應 (以 Java 為例)

概念	本地應用實作	分散式應用實作
行為封裝	Class	Container Image
行為實例	Object	Container
複用單位	.jar	Container Image
組合	Class A contains Class B	Sidecar Pattern
繼承	Class A extends Class B	Container FROM parent image
部署單位	.jar / .war	Pod
構建/運行時隔離	Module, package, class	Namespace, Pod, Container
啟動前提條件	Constructor	Init container
啟動後觸發	Init-method	postStart
銷毀前觸發	Destroy-method	prestop
非同步和平行處理	ThreadPoolExecutor, ForkJoinPool	Job
定期任務	Timer, ScheduledExecutorService	CronJob
背景任務	Daemon thread	DaemonSet
組態管理	Properties	Configmap, Secret

Kubernetes 重要物件



基礎 Pattern

透過 ConfigMap 注入依賴

- 應用組態是一種必要的依賴
- 透過 ConfigMap 或 Secert 實踐

```
kind: Pod
apiVersion: v1
metadata:
  name: random-generator
spec:
  container:
    - image: k8spatterns/random-generator:1.0
      name: random-generator
    env:
      - name: PATTERN
        valueFrom:
          configMapKeyRef:
            name: random-generator-config
            key: pattern
```

應用 Pod 參考的環境變數 (ConfigMap)

建立 config

- 透過文件建立 configmap

```
$ oc create configmap app-config --from-file=appconfig
```

- 透過文字建立 configmap

```
$ oc create configmap my-config --from-literal=foo=bar --from-literal=app=blu
```

- 透過文件建立 secret

```
$ oc create secret generic app-secret --from-file=appsecret
```

- 透過文件建立 secret

```
$ oc create secret generic my-secret --from-literal=secret.key=secretvalue
```

設定應用環境變數

- 將 configmap 設定至 deployment 物件

```
$ oc set env deployment/my-deployment --from configmap/my-config
```

- 將 secret 設定至 deployment 物件

```
$ oc set env deployment/my-deployment --from secret/my-secret
```

- 列出 pod 中的環境變數

```
$ oc set env pods <POD_NAME> --list
```



```
oc set env pods demo-56596dcb6c-757g6 --list
# pods/demo-56596dcb6c-757g6, container container
# APP from configmap my-config, key app
# FOO from configmap my-config, key foo
# APPCONFIG from configmap appconfig, key appconfig
```

fed

透過資源配置應用資源

	When it's Evaluated	Purpose
Quota	Request Time	限制單個或多個租戶可以請求的資源數量，以確保該租戶不會過分使用叢集資源
Request	Scheduling	Requests 用於排定最適合工作負載運行的節點 (連同其他的調度標準)
Limit	Runtime	Limits 用於限制容器在運行時可以使用的最大資源量

```
kind: Pod
apiVersion: v1
metadata:
  name: random-generator
spec:
  container:
  - image: k8spatterns/random-generator:1.0
    name: random-generator
    resources:
      requests:
        cpu: 100m
        memory: 200Mi
      limits:
        memory: 200Mi
```

設定應用配置

- 設定 deployment 資源上下限

```
$ oc set resources deployment <DEPLOYMENT_NAME> --limits=cpu=200m,memory=512Mi --requests=cpu=100m,memory=256Mi
```

- 設定 deployment readiness probe

```
$ oc set probe deployment <DEPLOYMENT_NAME> --readiness --get-url=http://:8080/healthz --initial-delay-seconds=10
```

- 設定 deployment liveness probe

```
$ oc set probe deployment <DEPLOYMENT_NAME> --liveness --get-url=http://:8080/healthz --initial-delay-seconds=10
```

應用的優先序 – Pod Priority

- Pod Priority 決定 Pod 相對的優先等級
- 當資源不足時，scheduler 可以移除優先等級較低的服務，調度優先度高的服務
- 可以透過 preemptionPolicy: Never 讓 Pod 不被移除

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000
globalDefault: false
description: This is a very high-priority Pod class
---
kind: Pod
apiVersion: v1
metadata:
  name: random-generator
  labels:
    env: random-generator
spec:
  container:
    - image: k8spatterns/random-generator:1.0
      name: random-generator
      priorityClassName: high-priority
```

value: 優先值
globalDefault: 該 PriorityClass 是否為全域設定

參考 PriorityClass 物件定義

Namespace 範疇的資源配置

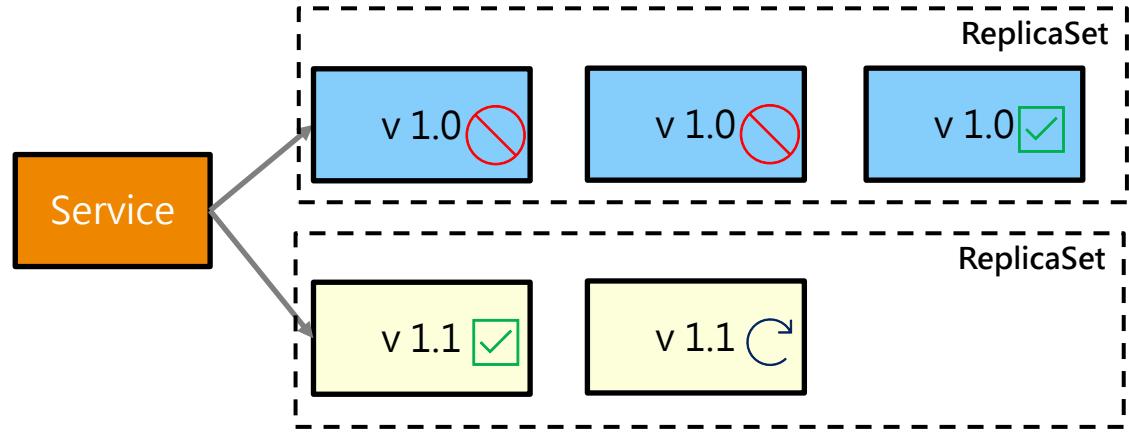
- Resource Quota: 可以限制 Namespace 內運算資源、儲存資源及物件資源的使用量
- LimitRanges: 可以控制 Container 或 Pod 資源配置不會超過叢集可提供的範圍外

```
kind: ResourceQuota
apiVersion: v1
metadata:
  name: object-counts
  namespace: default
spec:
  hard:
    pod: 4
    limits.memory: 5Gi
```

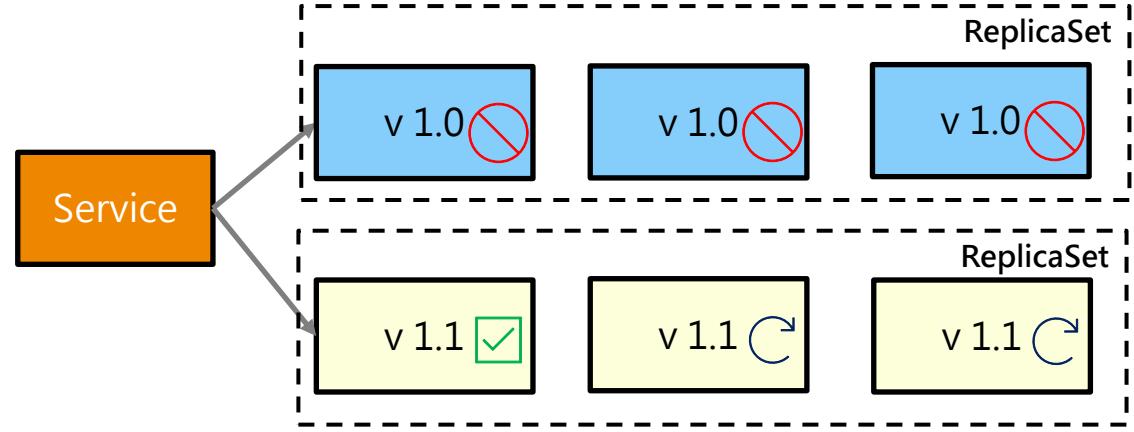
```
kind: LimitRange
apiVersion: v1
metadata:
  name: limits
  namespace: default
spec:
  limits:
  - min:
      memory: 250Mi
      cpu: 500m
    max:
      memory: 2Gi
      cpu: 2
  default:
    memory: 500Mi
    cpu: 500m
  defaultRequest:
    memory: 250Mi
    cpu: 250m
  maxLimitRequestRatio:
    memory: 2
    cpu: 4
  type: Container
```

部署策略

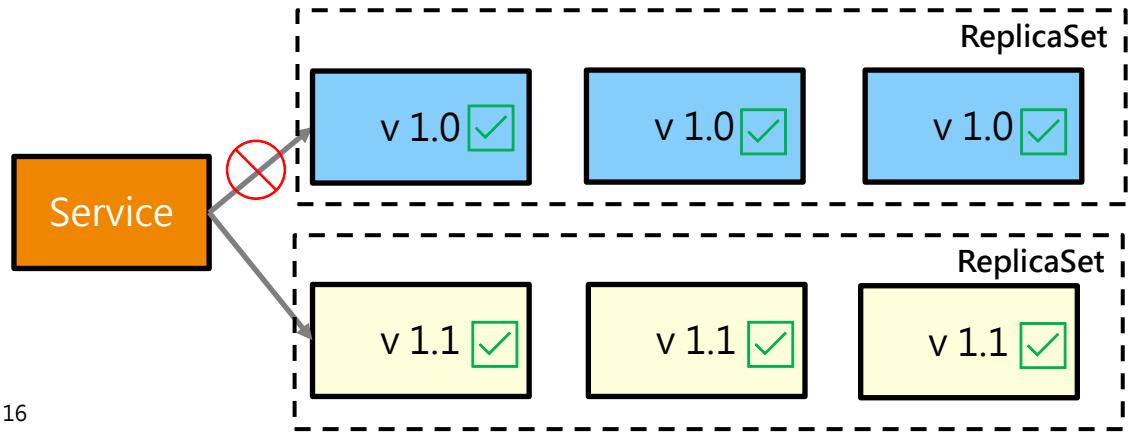
Rolling deployment



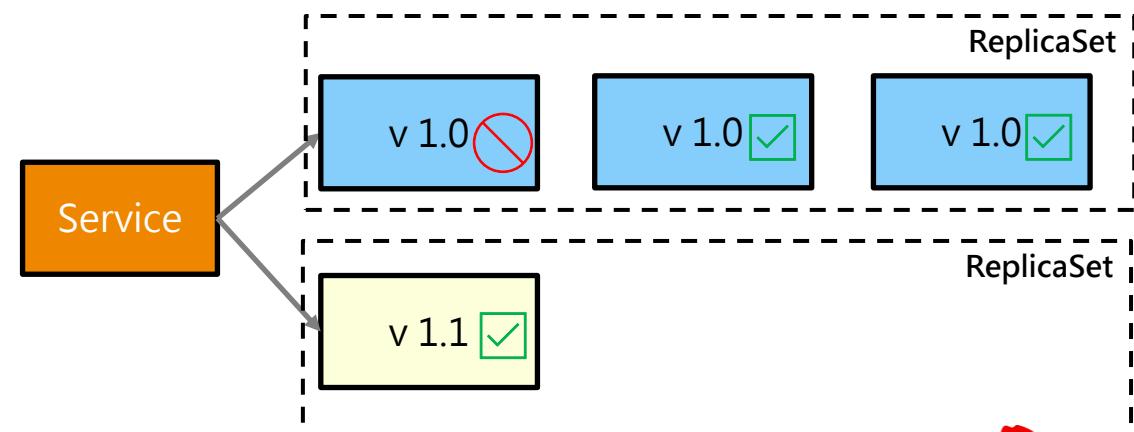
Fixes deployment



Blue Green Release

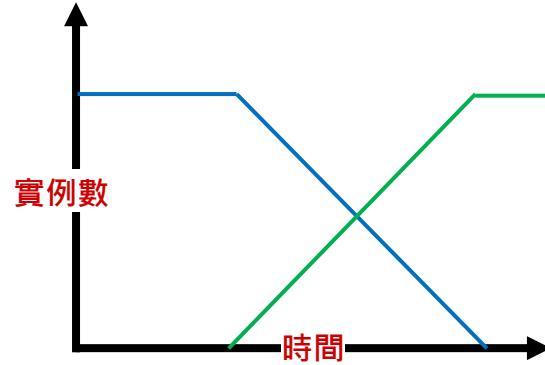


Canary Release

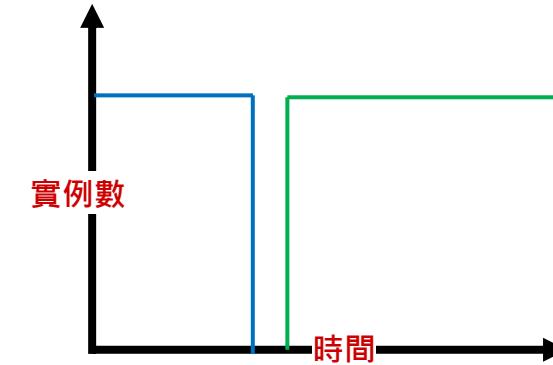


部署策略對應用的影響

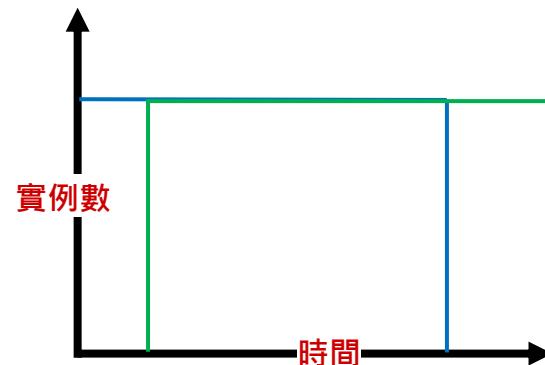
Rolling deployment



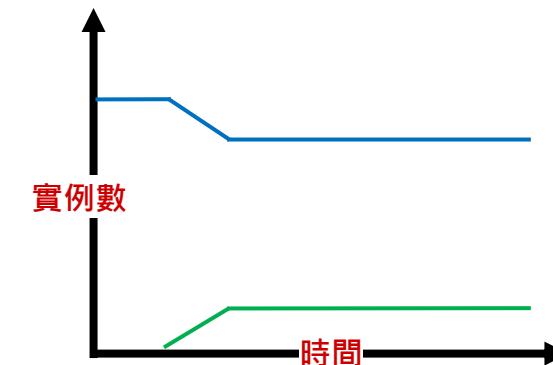
Fixes deployment



Blue Green Release

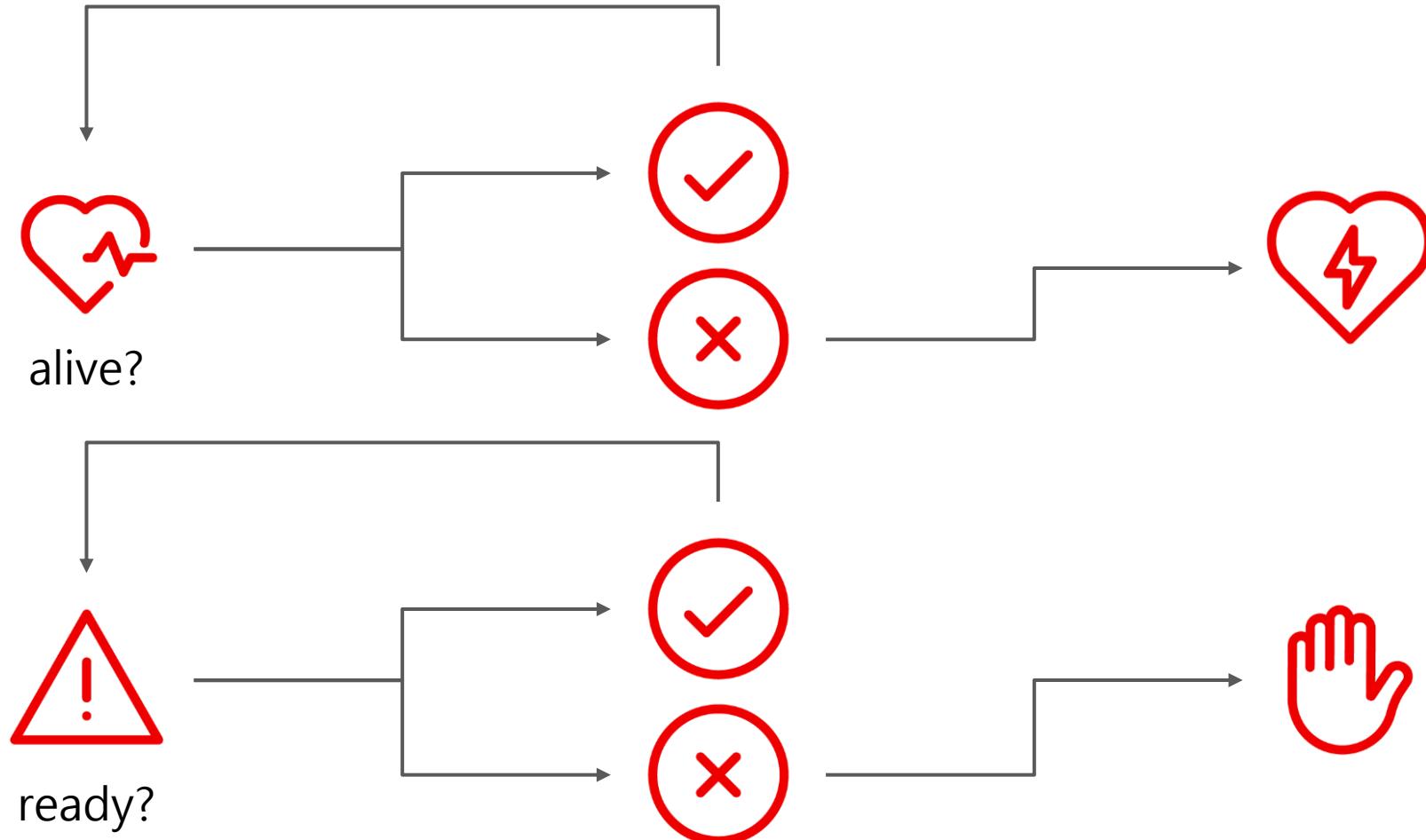


Canary Release

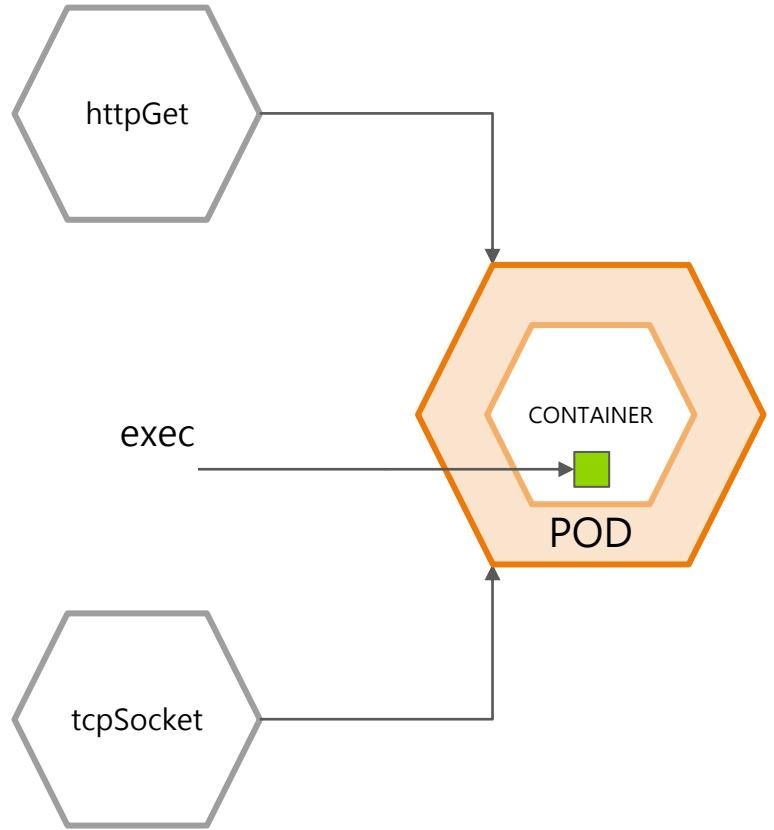


健康探針及應用生 命週期

Liveness / Readiness / Startup



三種方式，一個目標



initialDelaySeconds: Pod 啟動後等多久開始進行探針測試

timeoutSeconds: 花多久時間等一個成功的連線
(httpGet, tcpSocket only)

periodSeconds: 偵測頻率

failureThreshold: 判斷失敗的連續失敗數條件

Liveness Probe 及 Readiness Probe 配置

- Liveness Probe: 針對 http 端口進行探測，等候 30 秒避免應用尚未啟動
- Readiness Probe: 確認應用內存在的檔案，表示應用已經可以執行需求

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-with-liveness-check
spec:
  container:
    - image: k8spatterns/random-generator:1.0
      name: random-generator
      env:
        - name: DELAY_STARTUP
          value: "20"
    ports:
      - containerPort: 8080
    protocol: TCP
    livenessProbe:
      httpGet:
        path: /actuator/health
        port: 8080
      initialDelaySeconds: 30
```

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-with-readiness-check
spec:
  container:
    - image: k8spatterns/random-generator:1.0
      name: random-generator
      readinessProbe:
        exec:
          command: [ "stat", "/var/run/random-generator-ready" ]
```

Startup Probe 配置

- 針對啟動時間較長的應用，配置 Startup Probe，與 liveness / readiness 相比其等待時間更長

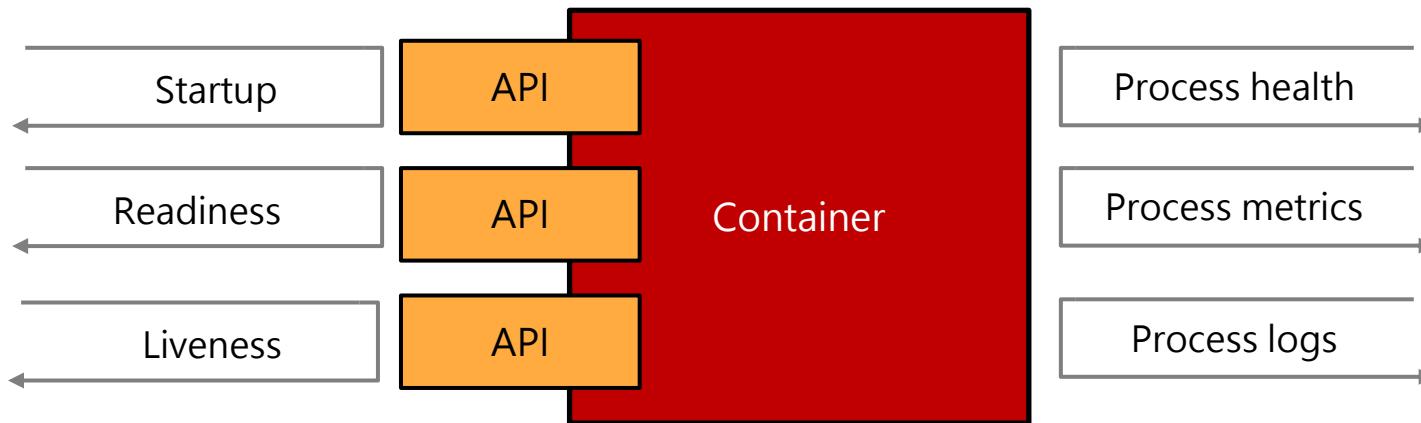
```
kind: Pod
apiVersion: v1
metadata:
  name: pod-with-startup-check
spec:
  container:
  - image: quay.io/wildfly/wildfly
    name: wildfly
    startupProbe:
      exec:
        command: [ "stat", "/opt/jboss/wildfly/standalone/tmp/startup-marker" ]
      initialDelaySeconds: 60
      periodSeconds: 60
      failureThreshold: 15
    livenessProbe:
      httpGet:
        path: /health
        port: 9990
        periodSeconds: 10
        failureThreshold: 3
```

Jboss 中的文件作為成功啟動的標記

啟動探針的時間參數

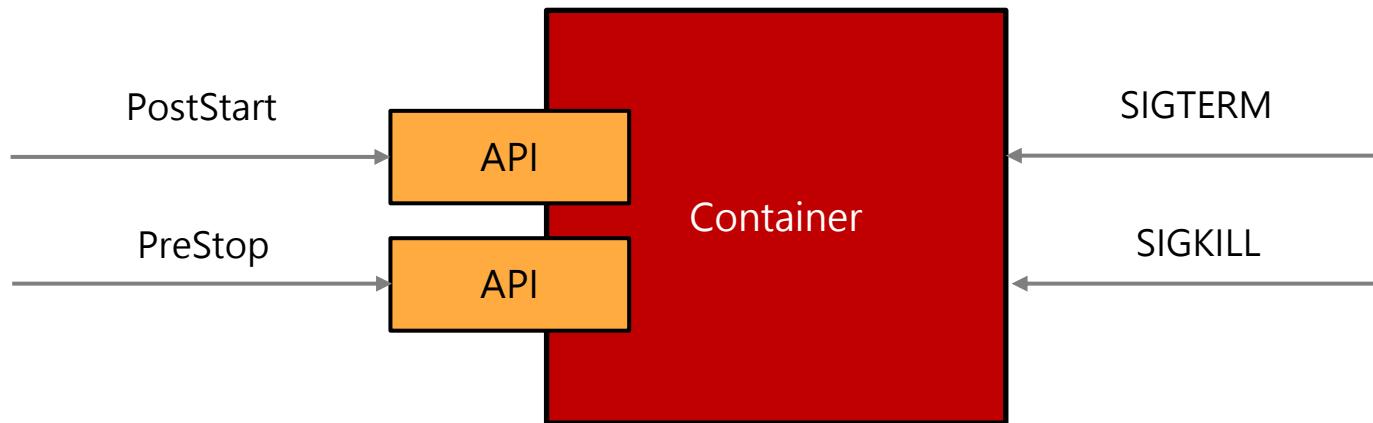
應用健康探針配置建議

- 應用透過實作 API 讓運行環境可以偵測其探針是雲原生應用的最佳實踐
- 應用框架實作健康探針範例: Quarkus SmallRye Health, Spring Boot Actuator, WildFly Swarm health check, Apache Karaf health check, or the MicroProfile spec for Java
- 透過將容器關閉的日誌紀錄在 /dev/termination-log 中，也能有效的透過日誌系統蒐集到應用被關閉的原因



SIGTERM 及 SIGKILL

- SIGTERM: 當 kubernetes 要通知應用關閉時，會發出 SIGTERM
- SIGKILL: 當容器沒有在 SIGTERM 通知發出後關閉時，系統會發出 SIGKILL 強制其關閉
- 預設 SIGKILL 會在 SIGTERM 30 秒後發出



PostStart Hook 和 PreStop Hook

- PostStart Hook 命令在容器建立後執行
- 容器狀態在 PostStart Hook 執行完畢前顯示 Pending 狀態
- PreStop Hook 用在當容器無法針對 SIGTERM 有正確反應時的 graceful shutdown 機制

```
kind: Pod
apiVersion: v1
metadata:
  name: post-start-hook
spec:
  container:
    - image: k8spatterns/random-generator:1.0
      name: random-generator
    lifecycle:
      postStart:
        exec:
          command:
            - sh
            - -c
            - sleep 30 && echo "Wake up!" > /tmp/postStart_done
```

```
kind: Pod
apiVersion: v1
metadata:
  name: pre-stop-hook
spec:
  container:
    - image: k8spatterns/random-generator:1.0
      name: random-generator
    lifecycle:
      preStop:
        httpGet:
          path: /shutdown
          port: 8080
```

Commandlet pattern

- 當 Pod 內的容器需要有特定的順序啟動
- 透過 entrypoint rewriting 機制達成更進階的控制

```
kind: Pod
apiVersion: v1
metadata:
  name: simple-random-generator
spec:
  containers:
    - image: k8spatterns/random-generator:1.0
      name: random-generator
      command:
        - "random-generator-runner"
      args:
        - "--seed"
        - "42"
```

當容器啟動時觸發該命令

可提供額外參數給 entrypoint 命令

應用調度最佳化

Node 資訊

透過 Node 相關指令可以了解到集群的狀態和健康

```
[maemmanu-redhat.com@clientvm 0 ~]$ oc get nodes -l node-role.kubernetes.io/worker
NAME                               STATUS    ROLES   AGE     VERSION
ip-10-0-136-144.us-east-2.compute.internal  Ready    worker  6d3h   v1.17.1
ip-10-0-156-199.us-east-2.compute.internal  Ready    worker  6d3h   v1.17.1
ip-10-0-168-78.us-east-2.compute.internal  Ready    worker  6d3h   v1.17.1
```

若 Node 的狀態是 not Ready 那它便無法和 OpenShift 控制平台溝通，對集群來說它就是無法服務狀態

```
[maemmanu-redhat.com@clientvm 0 ~]$ oc adm top nodes -l node-role.kubernetes.io/worker
NAME                           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
ip-10-0-136-144.us-east-2.compute.internal  163m        10%    1291Mi        18%
ip-10-0-156-199.us-east-2.compute.internal  438m        29%    2014Mi        29%
ip-10-0-168-78.us-east-2.compute.internal  347m        23%    1813Mi        26%
```

這是實際的使用量資訊，並不是請求資源量，控制平台是依據請求量去決定應用部署的使用空間

Node 資訊

```
[maemmanu-redhat.com@clientvm 1 ~]$ oc describe node ip-10-0-168-78.us-east-2.compute.internal
Name: ip-10-0-168-78.us-east-2.compute.internal
Roles: worker
output Omitted ...
Taints: <none>  Taints 手動或自動增加到 Node
Unschedulable: false
Lease:
  HolderIdentity: ip-10-0-168-78.us-east-2.compute.internal
  AcquireTime: <unset>
  RenewTime: Thu, 25 Jun 2020 11:46:59 +0000
Conditions:
  Type Status LastHeartbeatTime LastTransitionTime Reason Message
  ---- ----
  MemoryPressure False Thu, 25 Jun 2020 11:46:59 +0000 Fri, 19 Jun 2020 08:33:08 +0000 KubeletHasSufficientMemory kubelet has sufficient memory available
  DiskPressure False Thu, 25 Jun 2020 11:46:59 +0000 Fri, 19 Jun 2020 08:33:08 +0000 KubeletHasNoDiskPressure kubelet has no disk pressure
  PIDPressure False Thu, 25 Jun 2020 11:46:59 +0000 Fri, 19 Jun 2020 08:33:08 +0000 KubeletHasSufficientPID kubelet has sufficient PID available
  Ready True Thu, 25 Jun 2020 11:46:59 +0000 Fri, 19 Jun 2020 08:34:18 +0000 KubeletReady kubelet is posting ready status
output Omitted ...
Allocatable:
  attachable-volumes-aws-ebs: 39
  cpu: 1500m
  ephemeral-storage: 114381692328
  hugepages-1Gi: 0
  hugepages-2Mi: 0
  memory: 7010864Ki
  pods: 250
output Omitted ...
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource Requests Limits
  -----
  cpu 579m (38%) 300m (20%)
  memory 2829Mi (41%) 587Mi (8%)
  ephemeral-storage 0 (0%) 0 (0%)
  attachable-volumes-aws-ebs 0 0
Events: <none>
```

 Taints 手動或自動增加到 Node

 表示 Node 狀態

 Node 可以被分配的資源量

 Node 已經被分配的資源量

展示資源可用度，並且從部署調度的角度著手查找問題

Node Lifecycle: Reservations

Node Reservations for Static Pods and System Daemons (Kubelet, CRI-O)

除了定義驅逐閥值外，系統可以通過節點服務中
配置專門為節點服務和其他系統操作級別的服務
保留資源



kubeletArguments:

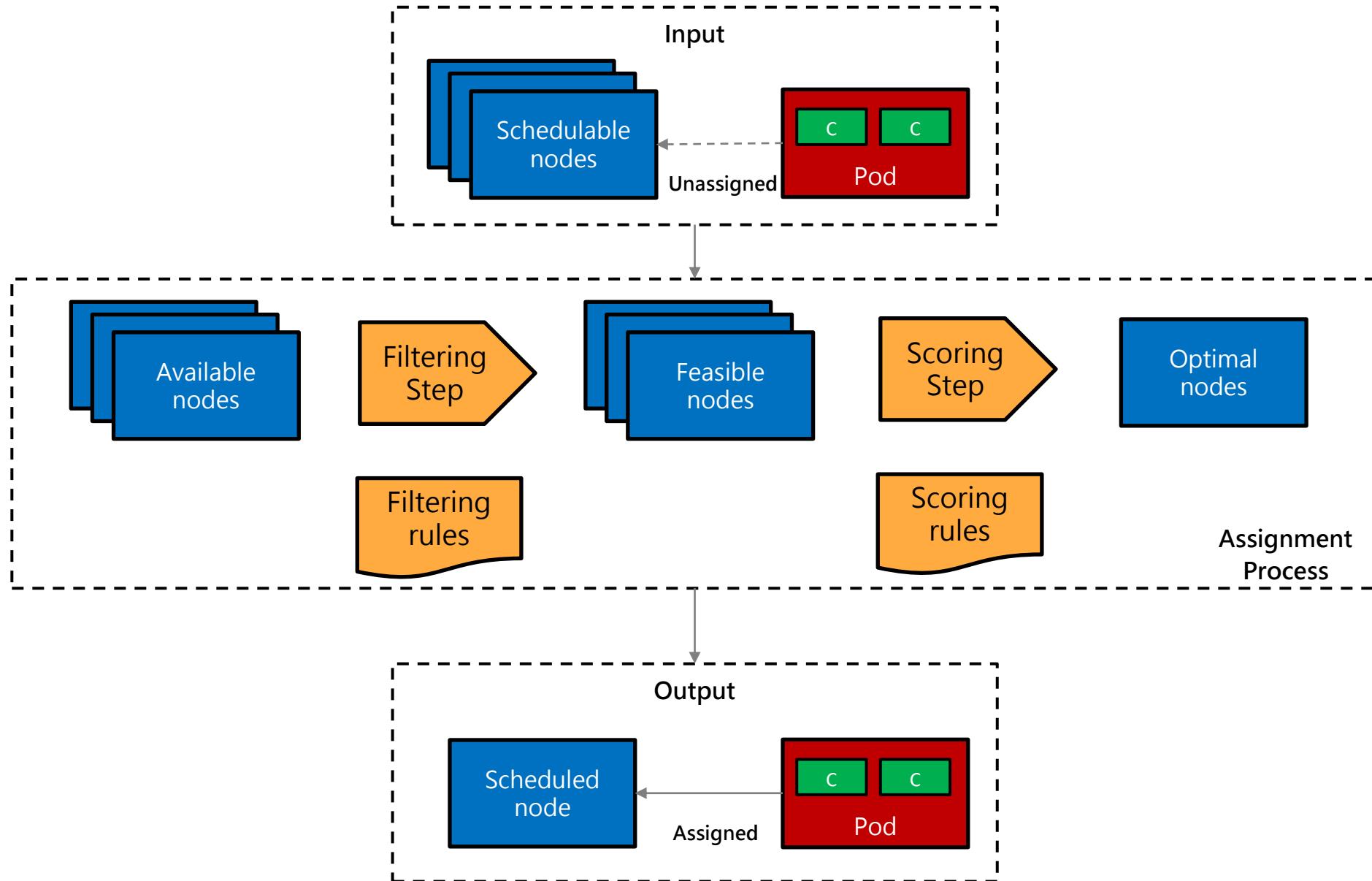
 kube-reserved:

- "cpu=<cpu>,memory=<mem>"

 system-reserved:

- "cpu=<cpu>,memory=<mem>"

容器調度流程



Node level 部署策略

- 透過指定 node selector 來指定 node label
- 透過 node affinity 紿予更詳細的規則
- required rules 是限定規則；preferred rules 是給予偏好權重

```
kind: Pod
apiVersion: v1
metadata:
  name: simple-random-generator
spec:
  container:
    - image: k8spatterns/random-generator:1.0
      name: random-generator
  nodeSelector:
    disktype: ssd
```

```
kind: Pod
apiVersion: v1
metadata:
  name: random-generator
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: numberCores
                operator: Gt
                values: [ "3" ]
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          preference:
            matchFields:
              - key: metadata.name
                operator: NotIn
                values: [ "control-plane-node" ]
  container:
    - image: k8spatterns/random-generator:1.0
      name: random-generator
```

Pod level 部署策略

- 透過 Pod Affinity 和 Anti-Affinity 能作到比 Node Affinity 更細緻化的配置策略
- Pod Affinity 可基於 topology key 作到基於現行 Pod 運行狀態的部署分布

```
kind: Pod
apiVersion: v1
metadata:
  name: random-generator
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              confidential: high
            topologyKey: security-zone
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchLabels:
                confidential: none
              topologyKey: kubernetes.io/hostname
    container:
      - image: k8spatterns/random-generator:1.0
        name: random-generator
```

基於現行 pod 運行狀態給予部署限制條件，並基於 label 確認如何與相關聯的 pod 部署

透過 anti-affinity 確保 pod 不會部署至錯誤的區域

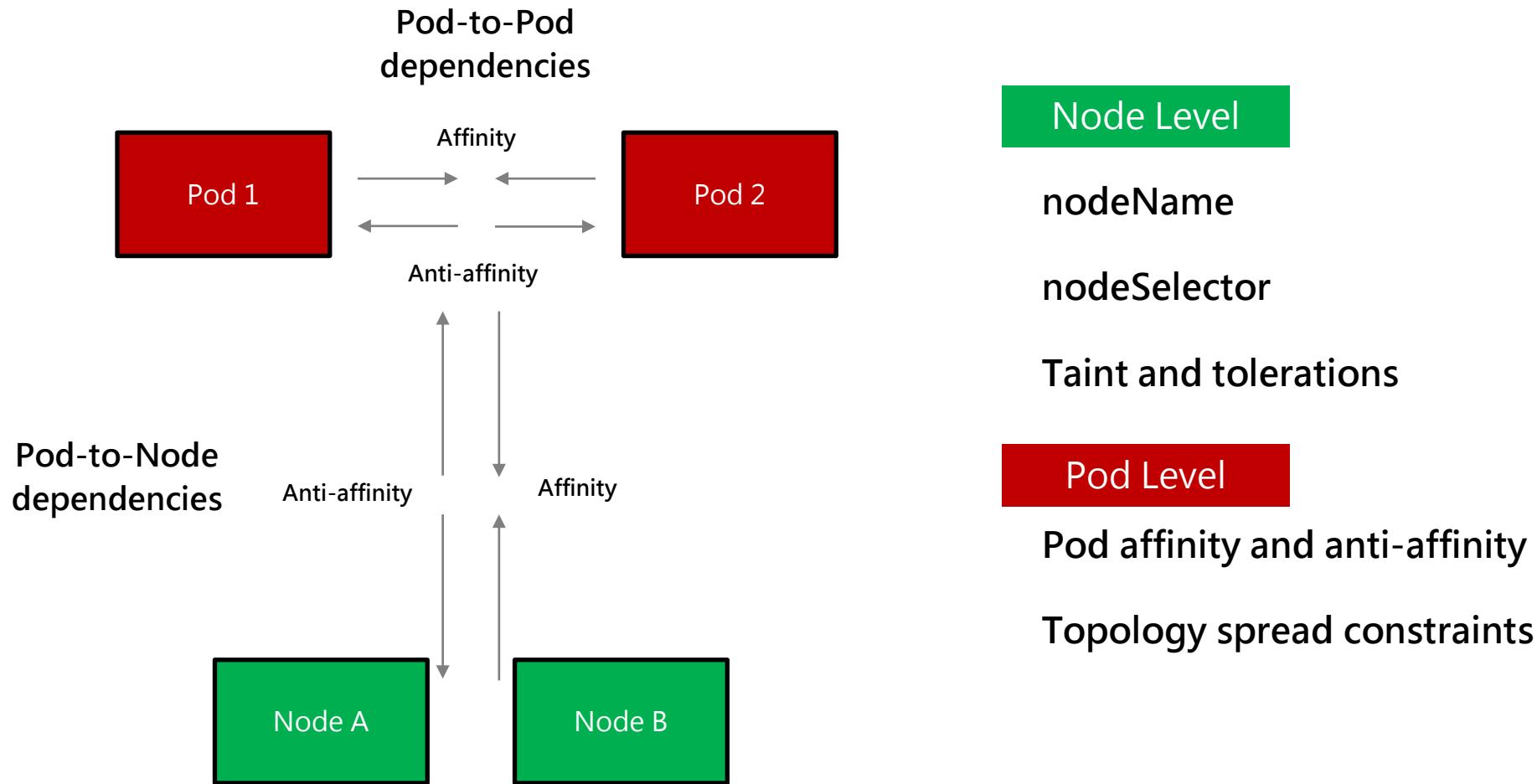
Taint 和 Toleration

- Node 可以透過 taint 來決定 Pod 是否可以部署於其上
- Pod 增加 Toleration 方可部署至有 taint 的 node

```
kind: Node
apiVersion: v1
metadata:
  name: control-plane-node
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/control-plane
      value: true
```

```
kind: Pod
apiVersion: v1
metadata:
  name: random-generator
spec:
  containers:
    - image: k8spatterns/random-generator:1.0
      name: random-generator
  tolerations:
    - key: node-role.kubernetes.io/control-plane
      operator: Exists
      effect: NoSchedule
```

指派 Pod 策略思考



分散式服務機制

Batch Job

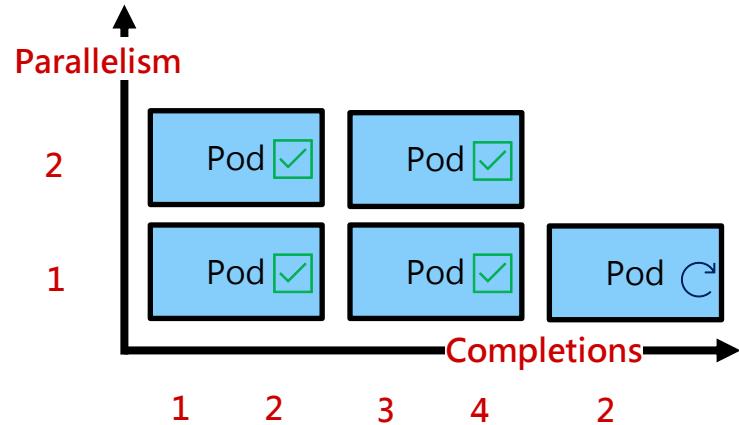
- Kubernetes Job 類似於 ReplicaSet，建立一個或多個 Pods 並確保其運行成功；不同點在於當其運作成功後，狀態會變成 complete 並不再建立新的 pod

```
kind: Job
apiVersion: batch/v1
metadata:
  name: random-generator
spec:
  completions: 5
  parallelism: 2
  ttlSecondsAfterFinished: 300
  template:
    metadata:
      name: random-generator
    spec:
      restartPolicy: OnFailure
      container:
        - image: k8spatterns/random-generator:1.0
          name: random-generator
          command: [ "java", "RandomRunner", "/numbers.txt", "10000" ]
```

- Job 不是一個無狀態的任務，而是在叢集重啟亦可持續存在
- Job 完成後，不會刪除，而是保留以便追蹤。這作為 Job 一部分建立的 Pod 也不會被刪除，便於用於檢查（例如，檢查容器日誌）
- 透過指定 *.spec.ttlSecondsAfterFinished* 指定特定時間便於將 Pod 在指定時間後刪除
- 使用 *.spec.completions* 欄位，可以指定 Pod 應成功完成的次數
- 透過 *.spec.parallelism* 欄位，可以指定同時起多個 Pod 以同步完成任務

Batch Job 配置策略

- `.spec.completions`: 定義多少 Pod 需運行來完成 Job
- `.spec.parallelism`: 定義多少 Pod replicas 平行運行



- Single Pod Jobs: `.spec.completions` 和 `.spec.parallelism` 設定為預設的 1
- Fix completion count Jobs: `.spec.completions` 設定到預定的數量 · `.spec.parallelism` 無限制；適合已知運行次數的任務
- Work queue Jobs: `.spec.completions` 不設定但 `.spec.parallelism` 大於一次；作為平行處理的任務

CronJob

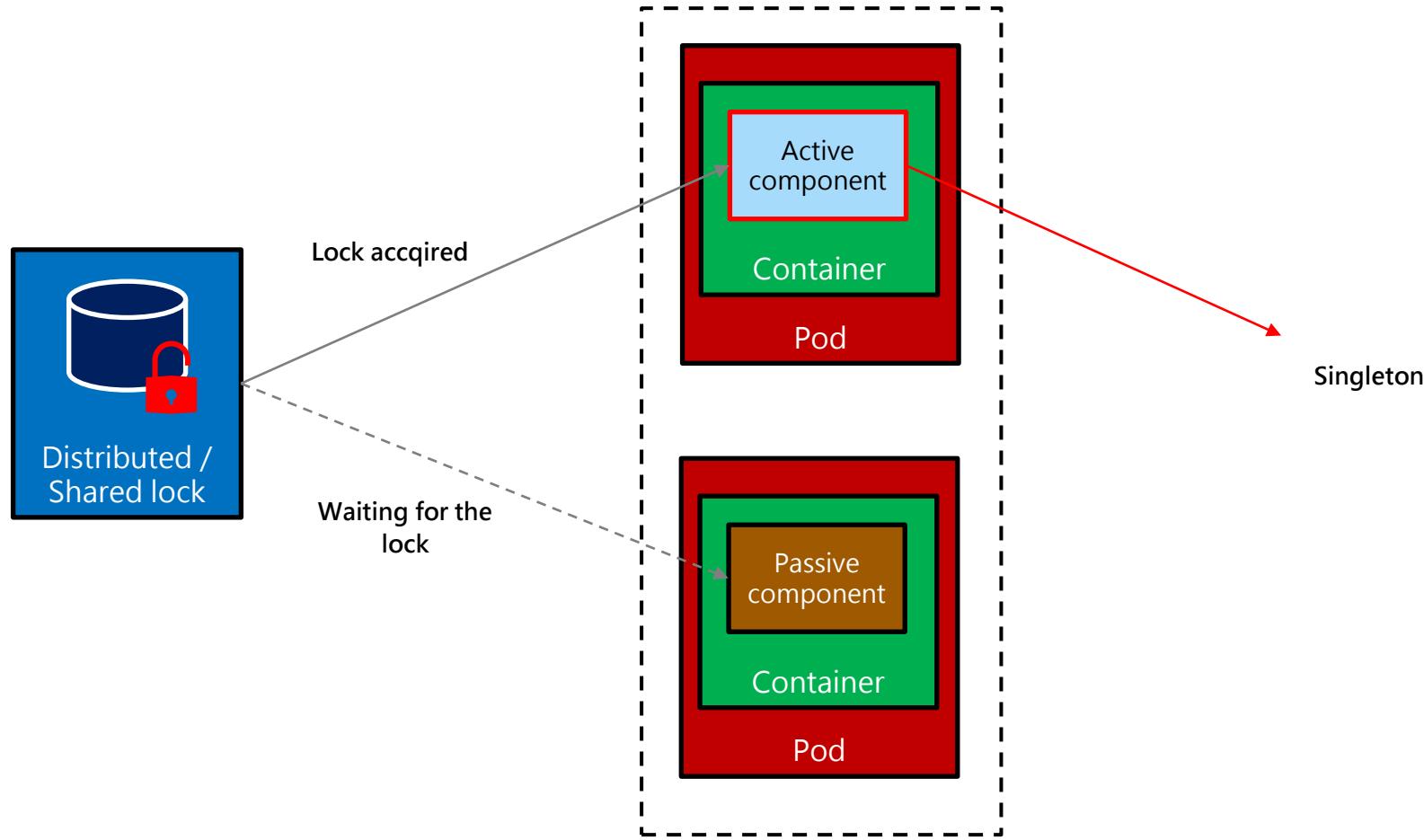
- 基於 Job 延伸，作為定期任務使用
- 類似於 Unix crontab

```
kind: CronJob
apiVersion: batch/v1
metadata:
  name: random-generator
spec:
  schedule: "*/3 * * * *"
  jobTemplate: <v1>
    spec:
      template:
        spec:
          restartPolicy: OnFailure
          container:
            - image: k8spatterns/random-generator:1.0
              name: random-generator
              command: [ "java", "RandomRunner", "/numbers.txt", "10000" ]
```

- .spec.schedule: 定義 Job 的排程
- .spec.startingDeadlineSeconds: 錯過排定時間的 deadline。建議不小於 10 秒
- .spec.concurrencyPolicy: 同一個 cronjob 同時管理 Job 的策略(eq. 即使前一個 Job 尚未完成，是否 Allow 建立新的 Job 實例)
- .spec.suspend: 在不影響已執行之 Job 前提下停止後續執行的 Job
- .spec.successfulJobsHistoryLimit 和 .spec.failedJobsHistoryLimit: 作為稽核用途保留的完成/失敗 Jobs

Singleton 服務

- Active-Passive 控制架構，實踐建議引入一些分散式應用運行框架（如 Dapr）
- 分散式狀態鎖工具: Apache ZooKeeper, Consul, Redis, etcd



PodDisruptionBudget

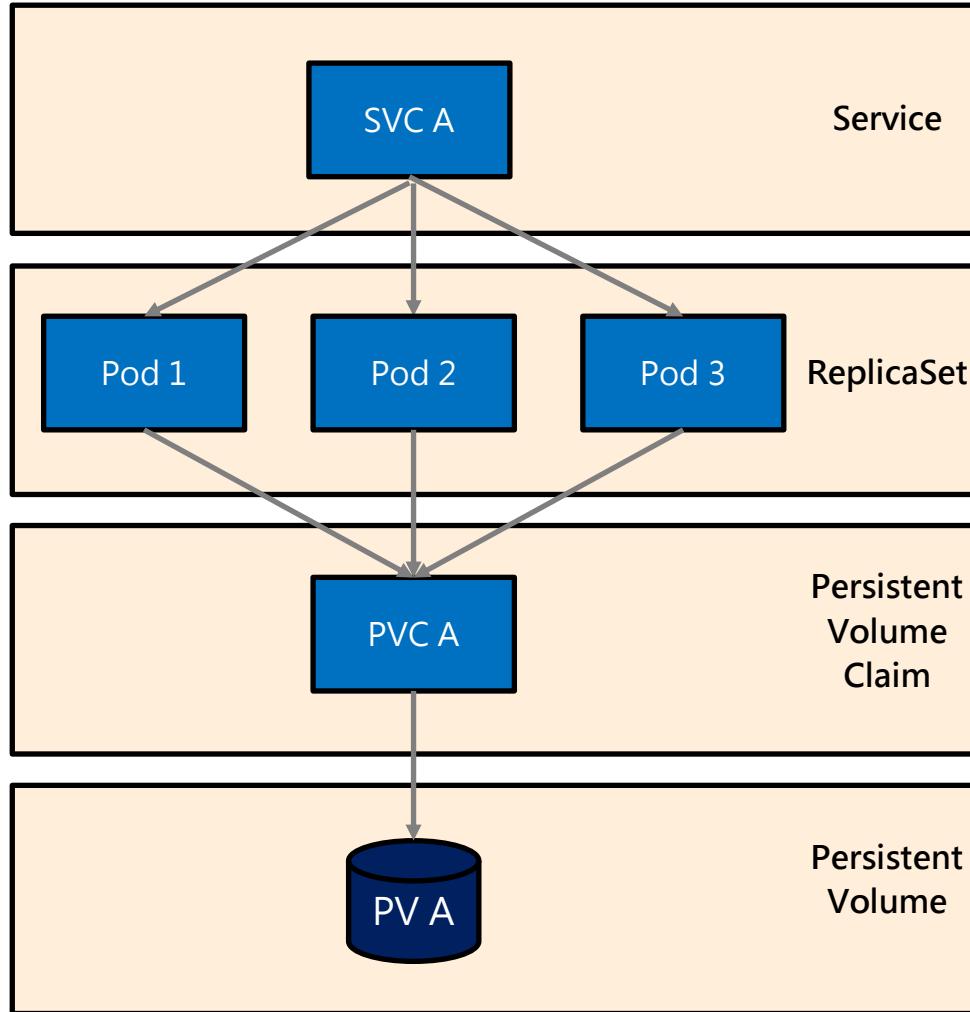
- 當系統需要確保「最少要多少應用活著」時，透過 PDB 設定能保證應用高可用
- 當系統是 Singleton 服務時，設定 PDB 亦可確保其被移除

```
kind: PodDisruptionBudget
apiVersion: policy/v1
metadata:
  name: random-generator-pdb
spec:
  selector:
    matchLabels:
      app: random-generator
  minAvailable: 2
```

透過 selector 決定需要配置的 Pod

指定最少可用數量 (亦可使用 %)

無狀態服務儲存本質



Persistent networking

- ReadWriteOnce: 每次在單一節點上 mount
- ReadOnlyMany: volume 在多個節點上 mount ; 僅讀
- ReadWriteMany: volume 在多個節點上 mount ; 讀寫
- ReadWriteOncePod: 單一個 Pod 有 access 保證 (使用這個模式會限制服務的擴充性)

Stateless service instances

Persistent storage

Storage provider

StatefulSet

```
kind: Service
apiVersion: v1
metadata:
  name: random-generator
spec:
  clusterIP: None
  selector:
    app: random-generator
  ports:
    - name: http
      port: 8080
```

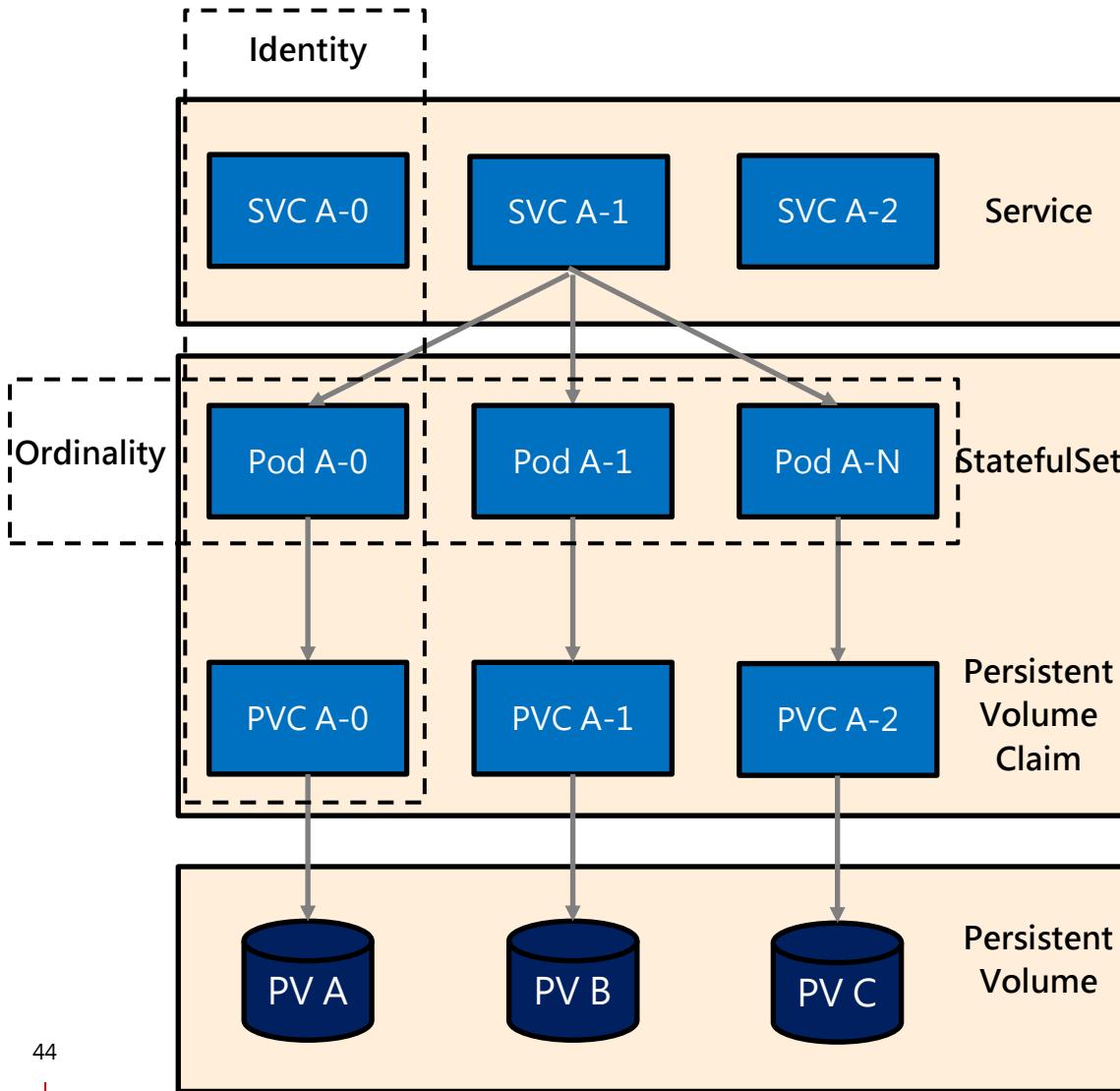
宣告為
headless 服務

```
kind: StatefulSet
apiVersion: apps/v1
metadata:
  name: random-generator
spec:
  serviceName: random-generator
  replicas: 2
  selector:
    matchLabels:
      app: random-generator
  template:
    metadata:
      labels:
        app: random-generator
    spec:
      containers:
        - image: k8spatterns/random-generator:1.0
          name: random-generator
          ports:
            - containerPort: 8080
              name: http
            volumeMounts:
              - name: logs
                mountPath: /logs
      volumeClaimTemplates:
        - metadata:
            name: logs
          spec:
            accessModes: [ "ReadWriteOnce" ]
            resources:
              requests:
                storage: 10Mi
```

參考對應的 Service 物件；產
生的 Pod 會給出 Identity 數字

為每個 Pod 建立的 PVC
的 template

有狀態服務儲存本質



Persistent networking

Persistent identity

Persistent storage

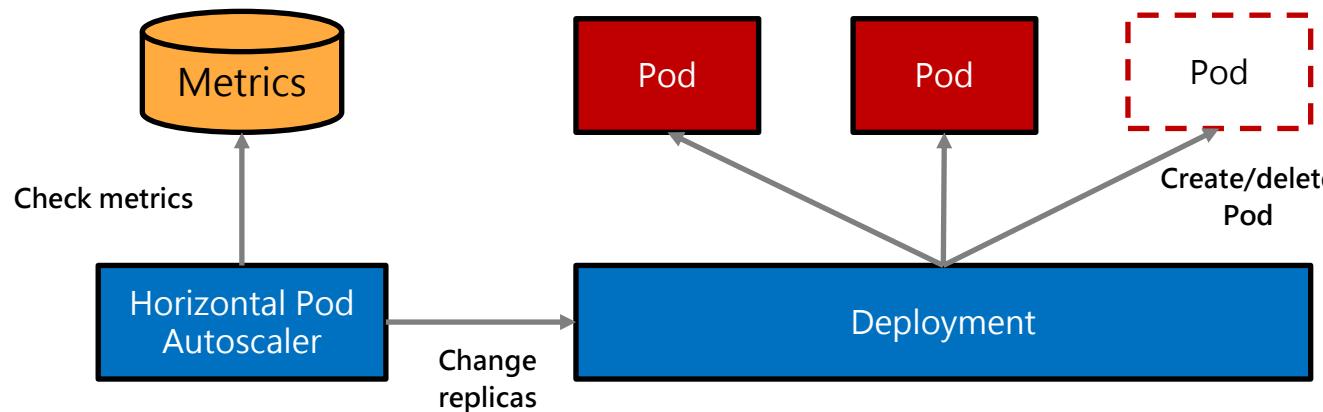
Storage provider

- **Identity:** 叢集有狀態服務很大程度上依賴每個實例都擁有其長期儲存和網絡 ID。這是因為在有狀態應用程式中，每個實例都是唯一的並且知道它自己的 ID，而該 ID 的主要成分是長期儲存和網路座標
- **Ordinality:** 除了唯一且長期存在的 ID 之外，叢集有狀態實例應用程式在實例集合中具有固定位置。這種排序通常影響實例放大和縮小的順序。此外該特性也可用於資料分發或存取，及叢集內行為定位(如 lock、singleton 或 leader)

Horizontal Pod Autoscaler 機制

- 透過 Kubernetes Metrics API 獲得的資訊，搭配 HPA 定義決定是否擴充應用
- 根據現狀指標和目標指標來計算 replicas 數量

$$desiredReplicas = \left[currentReplicas \times \frac{currentMetricValue}{desiredMetricValue} \right]$$



Horizontal Pod Autoscaler 配置

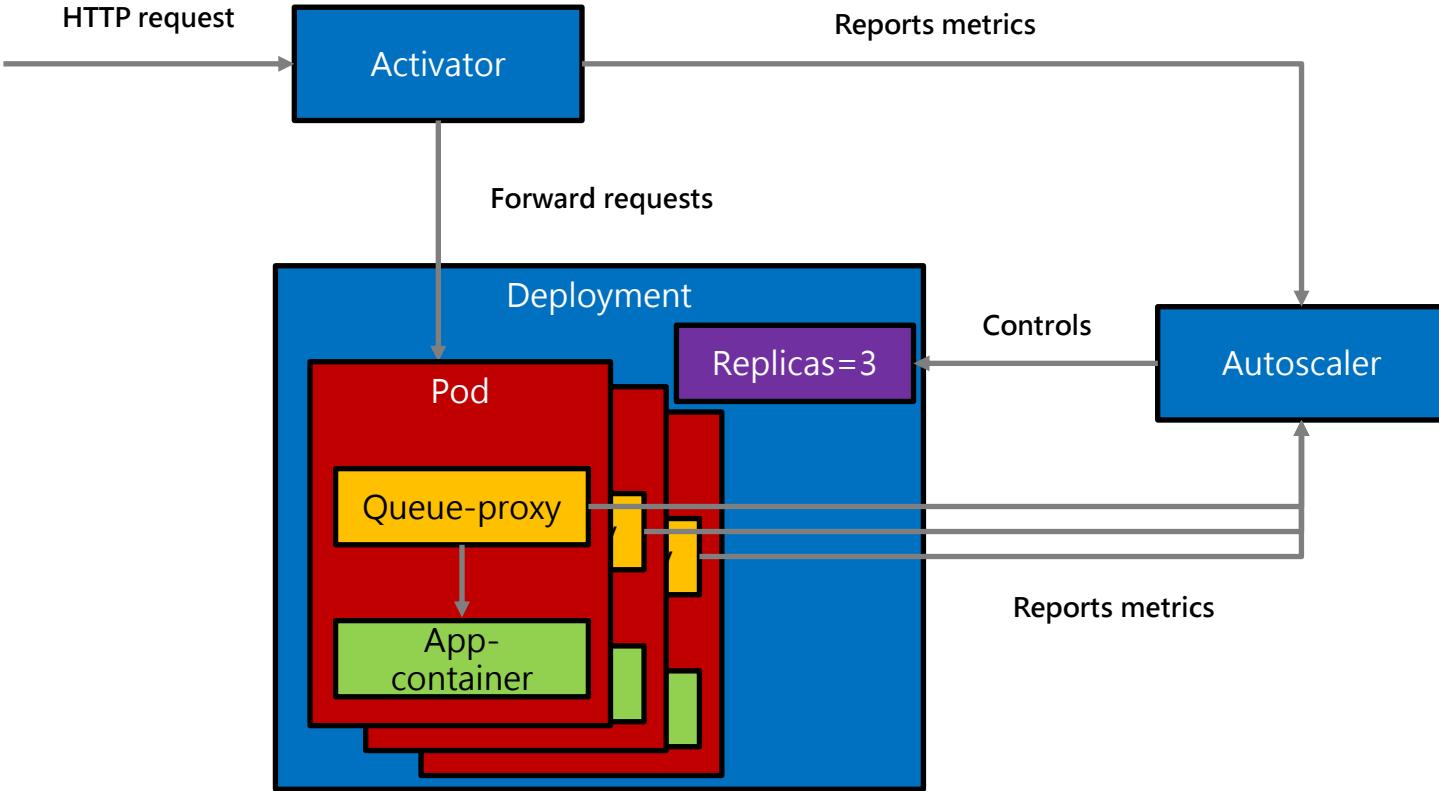
- 指標選擇: 原生 kubernetes 指標(cpu, memory)、客製指標(連線需求數、延遲長短...)
- 延遲執行: 避免應用因為指標波動而過度擴容、縮容

```
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
...
spec:
  ...
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
        - type: Percent
          value: 10
          periodSeconds: 60
    scaleUp:
      policies:
        - type: Pods
          value: 4
          periodSeconds: 15
```

縮容策略執行: 純予最少5分鐘的窗口
避免應用波動

擴容策略執行: 在15秒內最多擴4個
Pods

Knative 擴容策略



- Activator: 在應用前的 proxy，需求會先通過 Activator 直到 Pods 擴充到至少有1個。需求會 buffer 在 Activator 以避免丟失
- Queue proxy: 做為 sidecar 注入在應用 Pod 中，攔截擴容縮容所需之需求指標
- Autoscaler: 做為應用擴容縮容的判斷控制點

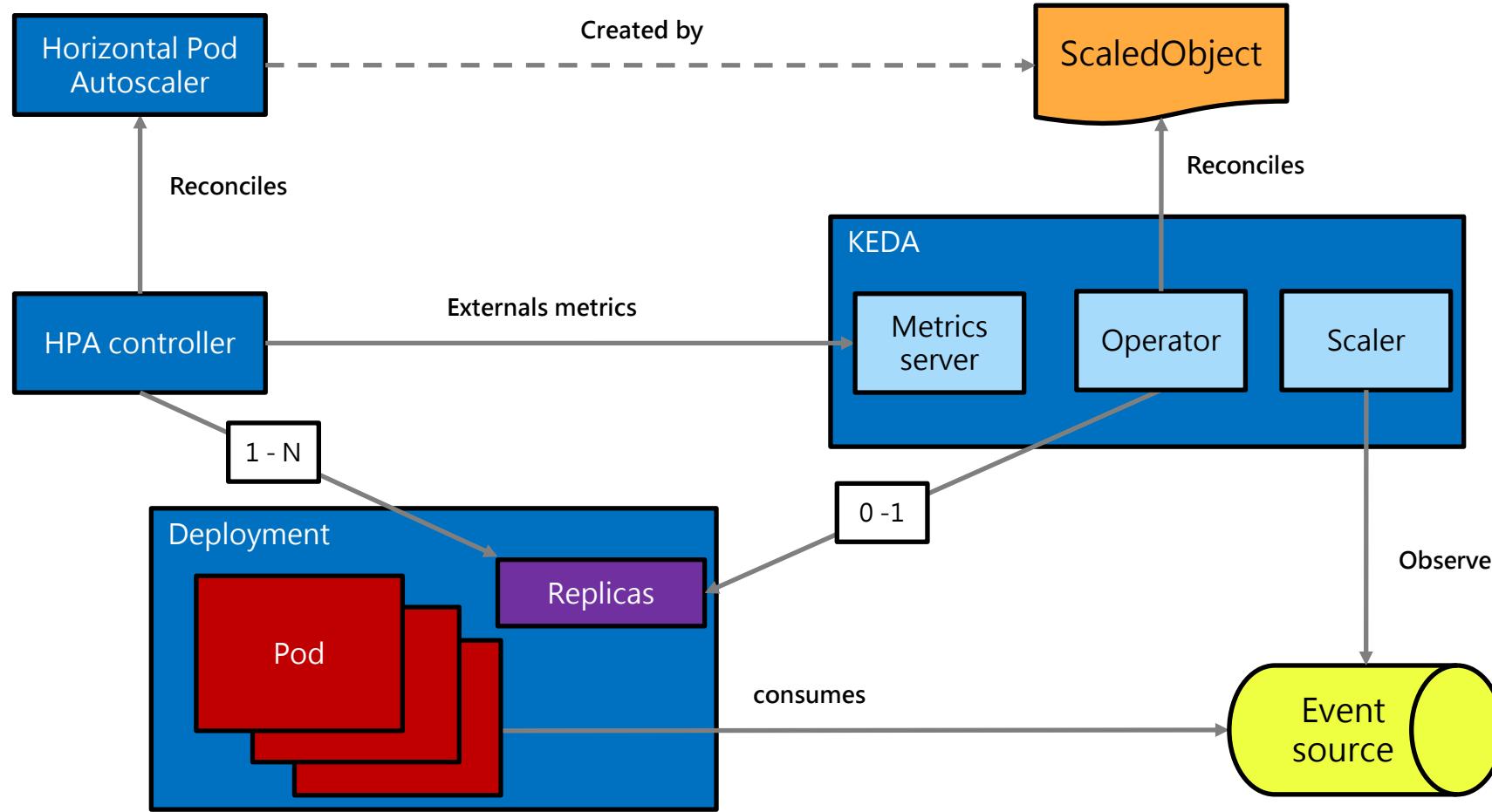
Knative 配置策略

- Knative service 做為 Knative Pods 對外暴露及擴充的控制物件

```
kind: Service
apiVersion: serving.knative.dev/v1
metadata:
  name: random
  annotations:
    autoscaling.knative.dev/target: "80"
    autoscaling.knative.dev/window: "120s"
spec:
  template:
    spec:
      containers:
        - image: k8spatterns/random
```

- target: 每個 replicas 可以處理的同時請求數，是軟限制；而 .spec.concurrencyLimit 是硬限制，無法超越。
- target-utilization-percentage: 做為 replicas 開始建立的判斷
- min-scale: 副本最小數；> 1代表不能縮到 0
- max-scale: 副本最大數；0代表無限制
- activation-scale: 從0開始擴容一次要啟動幾個副本
- scale-down-delay: 縮容要延遲多久才啟動
- Window: 做為指標平均判斷的時間窗口

KEDA 擴容策略

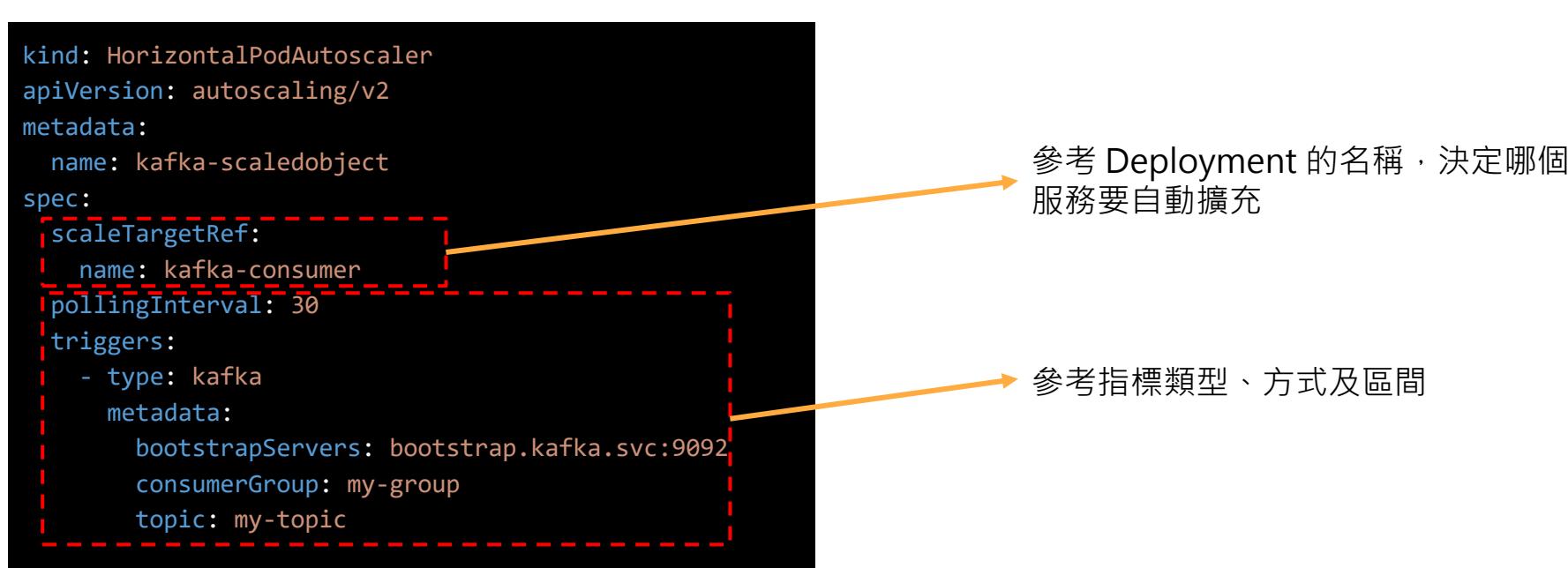


- 0 – 1 由 KEDA operator 接收指標判斷
- 1 – N 由啟動的工作負載，基於 KEDA 提供的指標交由 HPA controller 決定

ScaledObject 配置策略

- KEDA 核心的配置策略為 ScaledObject

```
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: kafka-scaledobject
spec:
  scaleTargetRef:
    name: kafka-consumer
  pollingInterval: 30
  triggers:
    - type: kafka
      metadata:
        bootstrapServers: bootstrap.kafka.svc:9092
        consumerGroup: my-group
        topic: my-topic
```



參考 Deployment 的名稱，決定哪個服務要自動擴充

參考指標類型、方式及區間

Kubernetes 上的自動擴容機制比較

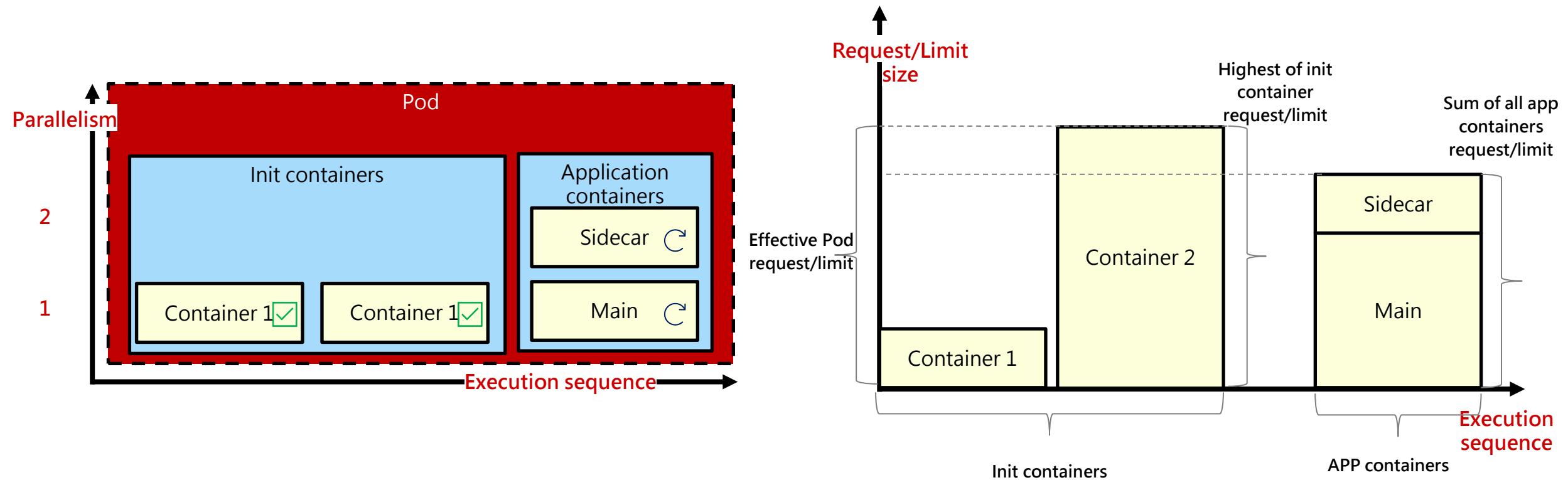
- Push model: 指標由一個控制去主動的推送给 autoscaler，決定是否擴容
- Pull model: 由應用或外部系統來獲取指標。通常是 autoscaler 無法直接訪問指標來源，或是指標儲存在外部系統。

	HPA	Knative	KEDA
擴充指標	Resource 使用量	HTTP 請求	外部客製指標
縮容到0	No	Yes	Yes
機制	Pull	Push	Pull
用例	穩定的 web 服務 Batch 類運行	Serverless 類型應用 Serverless functions	事件驅動微服務

結構 Pattern

Init container

- 將應用啟動和主應用本身分別
- init container 一般都比較輕量
- 相同點: 與 app container 歸屬同一個 Pod
- 相異點: 不享有共同的容器生命週期
- 由於 Init Container 的順序性，Pod 要求的資源需要依據最高的 init container 評估



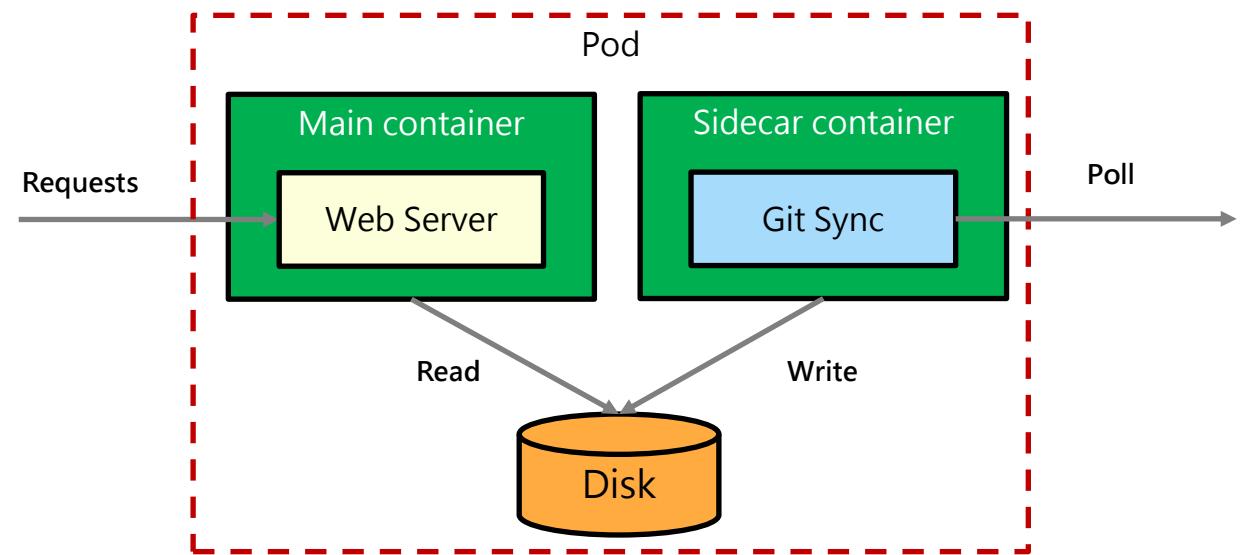
Init container 範例

```
kind: Pod
apiVersion: v1
metadata:
  name: www
  labels:
    app: www
spec:
  initContainers:
  - name: download
    image: axeclbr/git
    command:
      - git
      - clone
      - https://github.com/mdn/beginner-html-site-scripted
      - /var/lib/data
    volumeMounts:
      - mountPath: /var/lib/data
        name: source
  containers:
  - name: run
    image: docker.io/centos/httpd
    ports:
      - containerPort: 80
    volumeMounts:
      - mountPath: /var/www/html
        name: source
  volumes:
  - emptyDir: {}
    name: source
```

由 init container 複製 git 程式碼庫並放到 shared volume 讓 app container 共享

Sidecar container

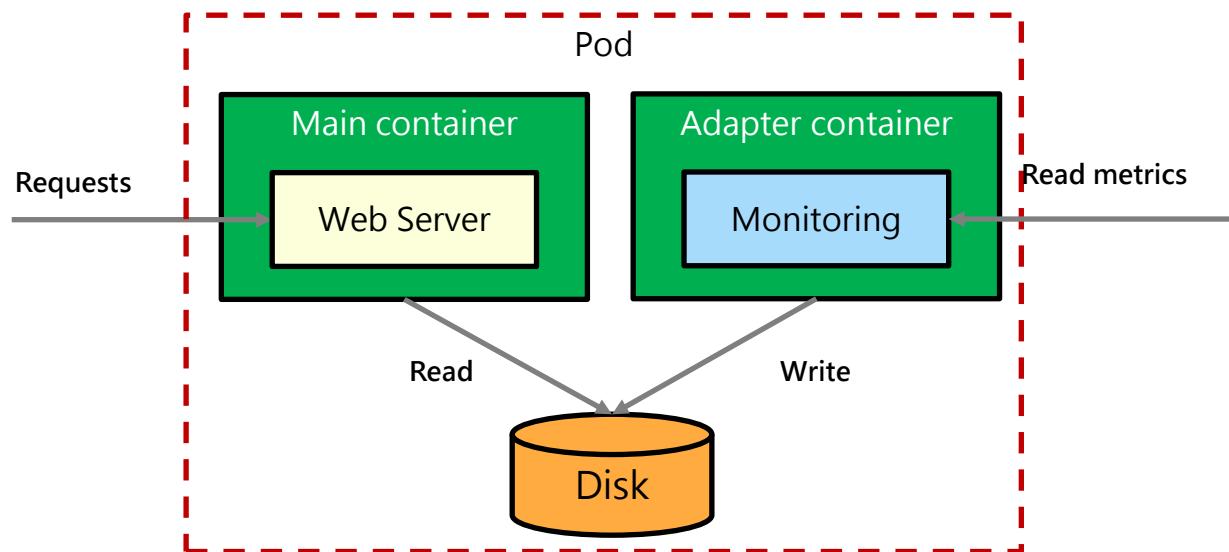
```
kind: Pod
apiVersion: v1
metadata:
  name: web-app
spec:
  containers:
    - name: app
      image: docker.io/centos/httpd
      ports:
        - containerPort: 80
      volumeMounts:
        - mountPath: /var/www/html
          name: git
    - name: poll
      image: axeclbr/git
      volumeMounts:
        - mountPath: /var/lib/data
          name: git
      env:
        - name: GIT_REPO
          value: https://github.com/mdn/beginner-html-site-scripted
      command:
        - "sh"
        - "-c"
        - "git clone $(GIT_REPO) . && watch -n 600 git pull"
      workingDir: /var/lib/data
  volumes:
    - emptyDir: {}
      name: git
```



Adapter pattern

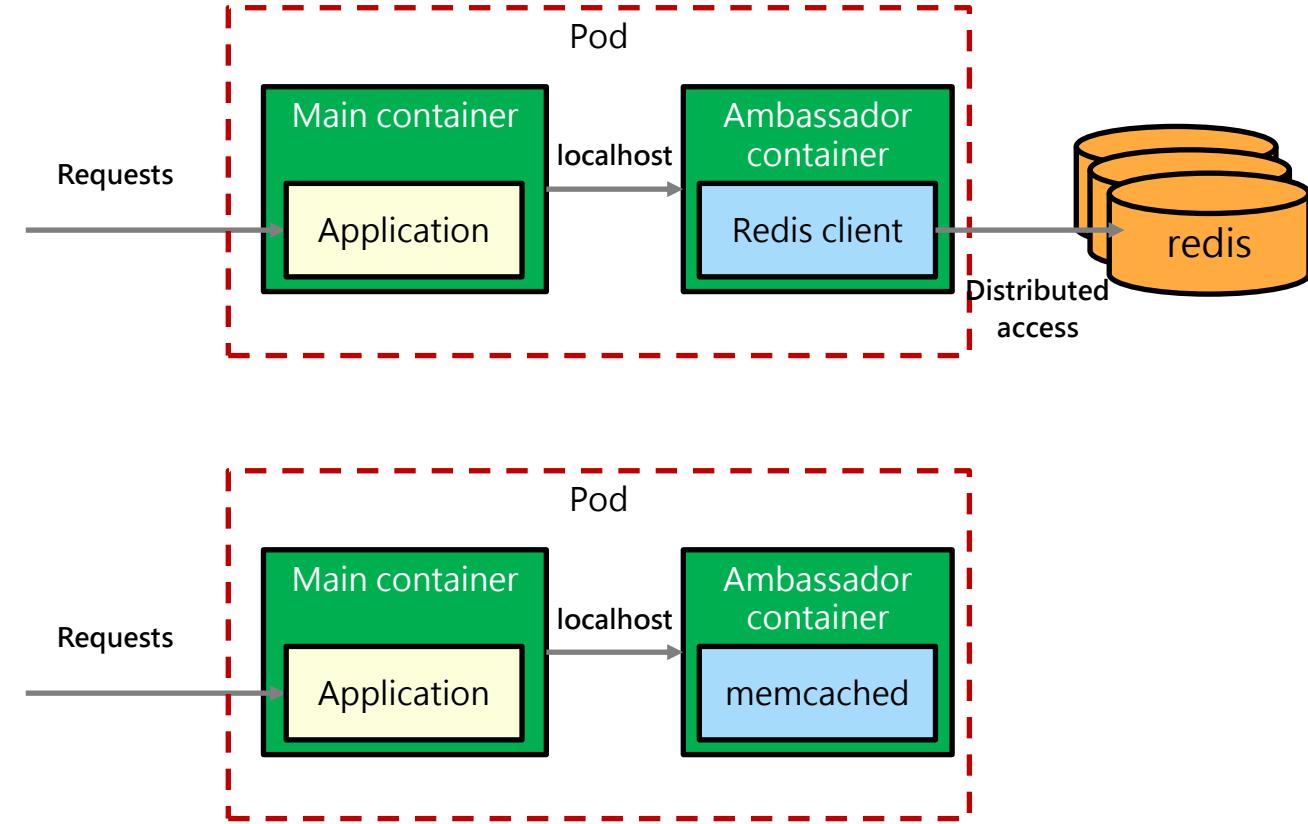
```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: random-generator
spec:
  replicas: 1
  selector:
    matchLabels:
      app: random-generator
  template:
    metadata:
      labels:
        app: random-generator
    spec:
      containers:
        - image: k8spatterns/random-generator:1.0
          name: random-generator
          env:
            - name: LOG_FILE
              value: /logs/random.log
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts:
            - mountPath: /logs
              name: log-volume
```

```
...
  - image: k8spatterns/random-generator-exporter
    name: prometheus-adapter
    env:
      - name: LOG_FILE
        value: /logs/random.log
    ports:
      - containerPort: 9889
        protocol: TCP
    volumeMounts:
      - mountPath: /logs
        name: log-volume
    volumes:
      - emptyDir: {}
        name: log-volume
```



Ambassador pattern

```
kind: Pod
apiVersion: v1
metadata:
  name: random-generator
  labels:
    app: random-generator
spec:
  containers:
    - name: main
      image: k8spatterns/random-generator:1.0
      env:
        - name: LOG_URL
          value: http://localhost:9009
      ports:
        - containerPort: 8080
          protocol: TCP
    - name: ambassador
      image: k8spatterns/random-generator-log-ambassador
```

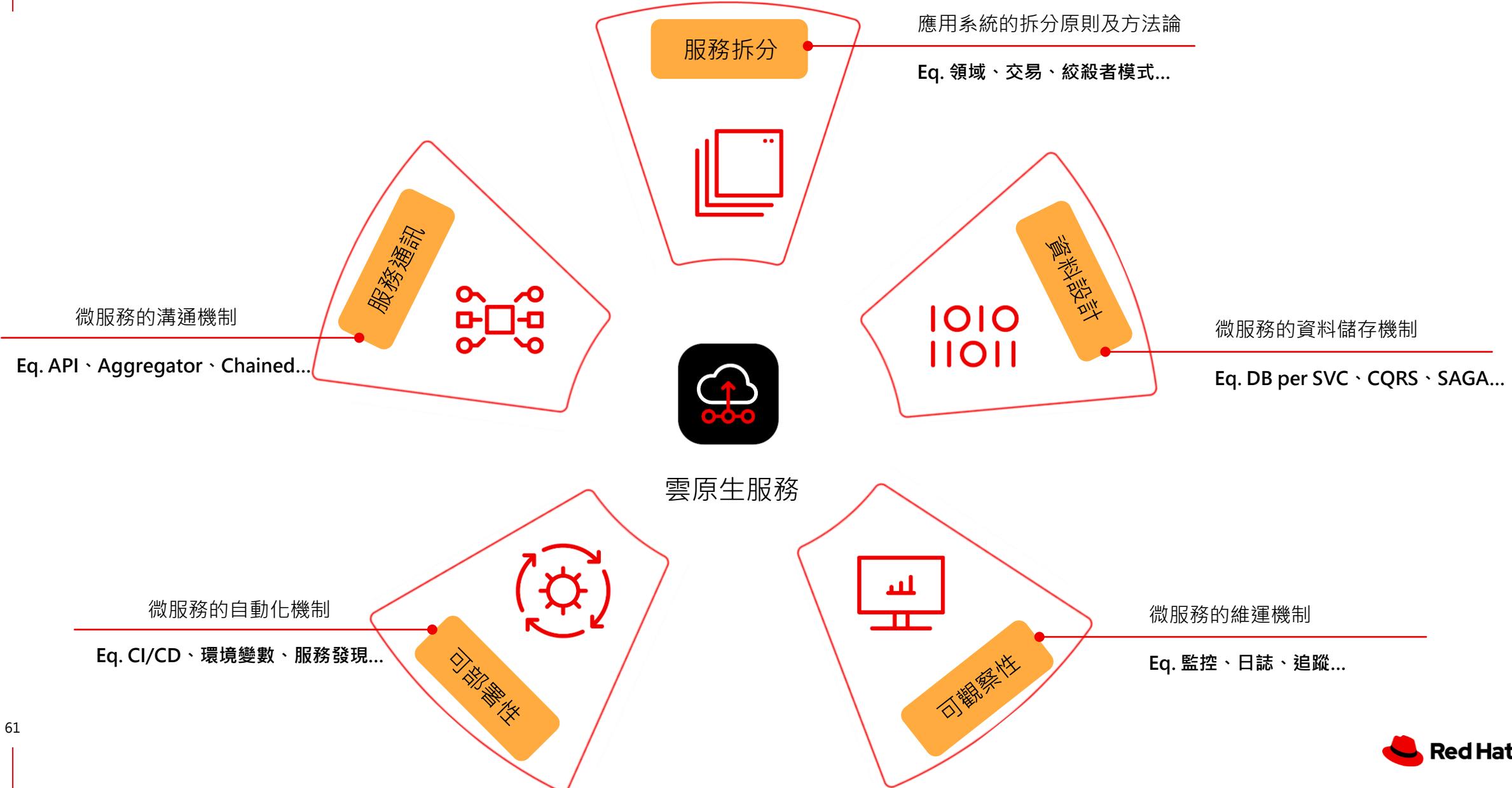


Demo 01 - Set health Probe

Lab 01 - Custom Autoscaler

Microservice Principles

從 12 法則實踐的微服務五大面向



容器微服務應用基本條件

1. 已建立了清晰的可自動化的編譯及構建流程

應用使用了如Maven、Gradle、Make或Shell等工具實現了構建編譯步驟的自動化。這將方便應用在容器平臺上實建。

2. 已實現應用配置參數外部化

應用已將配置參數外部化與設定檔或環境變數中，

3. 已提供合理可靠的健康檢查介面

容器平臺將通過健康檢查介面判斷容器狀態，對應用服務

4. 已實現狀態外部化，實現應用實例無狀態化

應用狀態資訊存儲於資料庫或緩存等外部系統，應用實例本身實現

5. 不涉及底層的作業系統依賴及複雜的網路通信機制

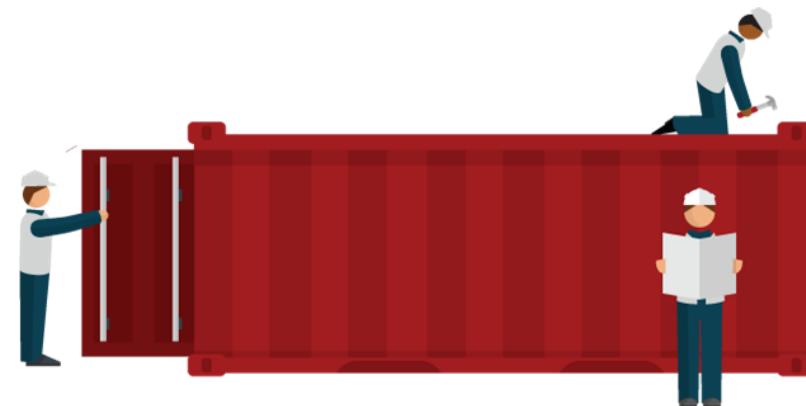
應用以處理業務為主，不強依賴於底層作業系統及組播等網路通信

6. 部署交付件及運行平臺的大小在2GB以內

羽量級的應用便於在大規模集群中快速傳輸分發，更符合容器敏捷的

7. 啟動時間在5分鐘以內

過長的啟動時間將不能發揮容器敏捷的特性。



容器化微服務應用最佳實踐

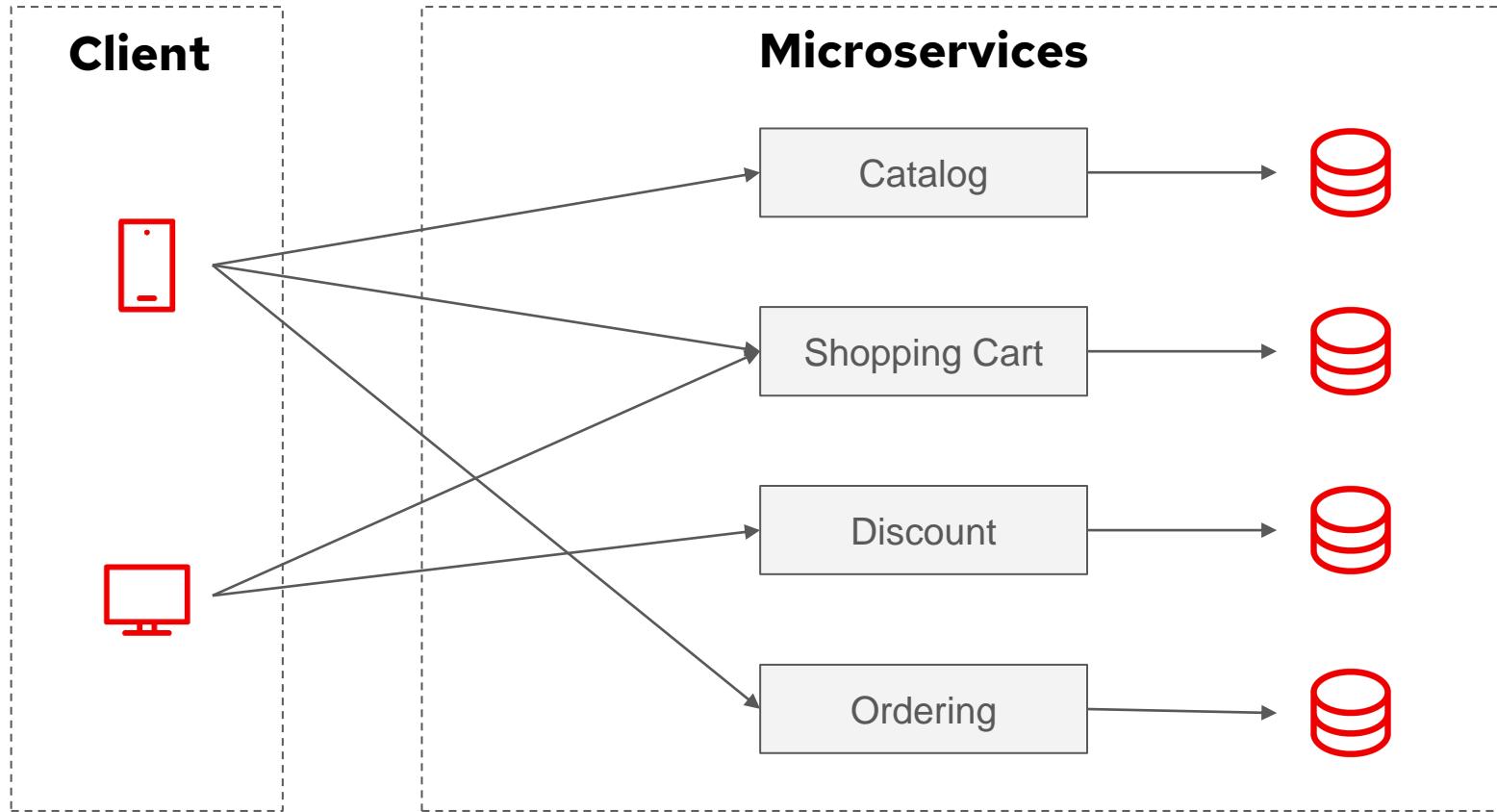
- 在pods定義中指定資源請求和資源限制。
- 使用Pod中斷預算保護應用程式（ pod disruption budgets. ）
- 每個容器運行一個進程。
- 實施應用程式監視和警報。
- 配置應用程式以將其日誌寫入stdout/stderr
- 使用受信任的base image。
- 使用最新版本的base image。
- 使用單獨的構建映射和運行時映射
- 盡可能遵守受限制的安全上下文約束(scc)
- 使用TLS保護應用程式元件之間的通信。



服務通訊原則及模 式

直接訪問模式

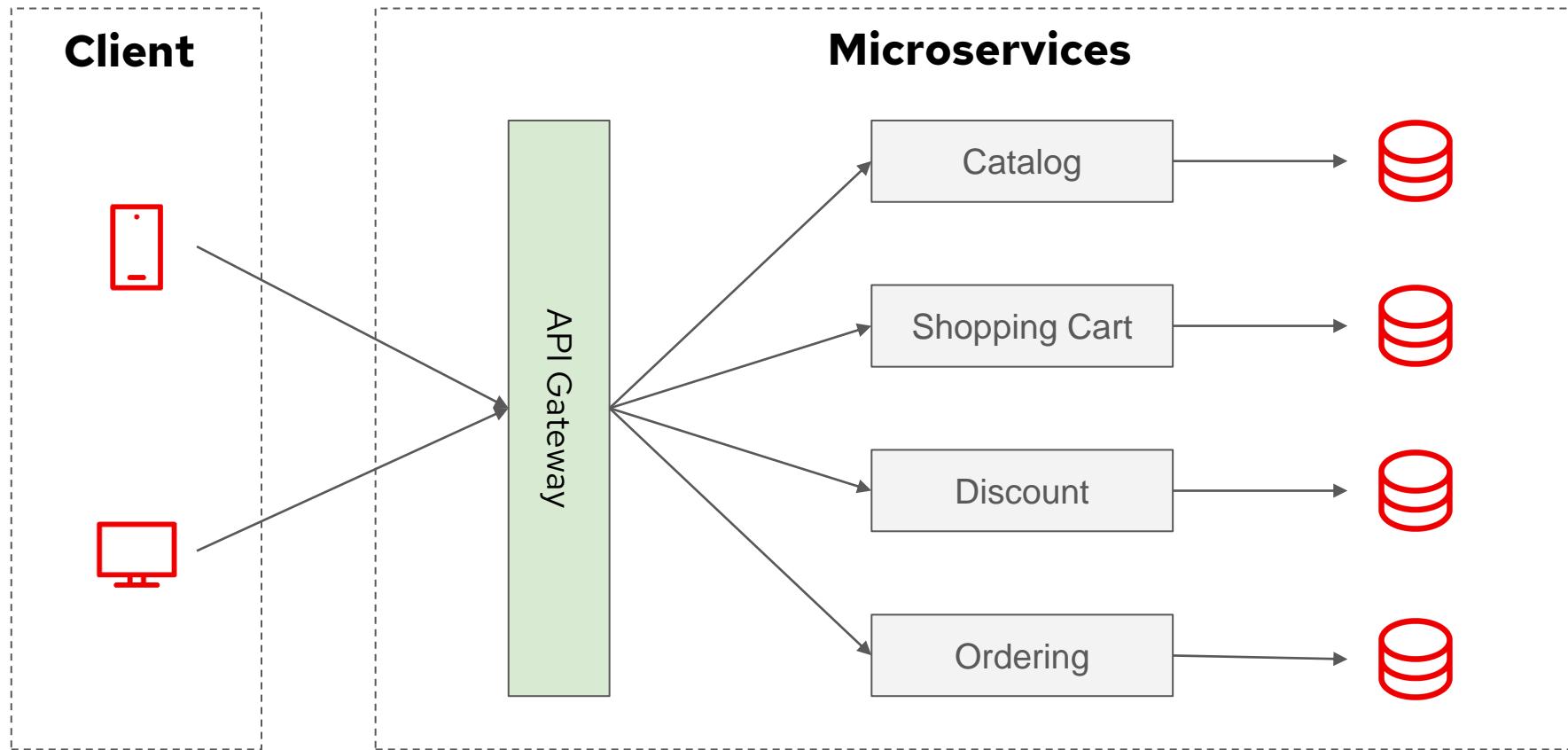
服務通訊



API Gateway 模式

服務通訊

API 網關位於客戶端 APP 和後端微服務之間充當 facade，它可以是反向代理，將客戶端請求路由到適當的後端微服務。它還支持將客戶端請求導到多個微服務，然後將響應聚合後返回給客戶端。它還支持必要的橫切關注點（SSL 終止、身份驗證、授權、節流、日誌記錄等）



API Gateway 模式



優點

- 在前端和後端服務之間提供松耦合
- 減少客戶端和微服務之間的調用次數
- 通過 SSL 終端、身份驗證和授權實現高安全性
- 集中管理的橫切關注點，例如，日誌記錄和監視、節流、負載平衡。

缺點

- 可能導致微服務架構中的單點故障
- 額外的網路調用帶來的延遲增加
- 如果不進行擴展，它們很容易成為整個企業應用的瓶頸。
- 額外的維護和開發費用

何時使用 API 網關

- 在複雜的微服務架構中，它幾乎是必須的。
- 在大型企業中，API 網關是中心化安全性和橫切關注點的必要工具。

何時不宜使用 API 網關

- 在安全和集中管理不是最優先要素的私人專案或小公司中。
- 如果微服務的數量相當少

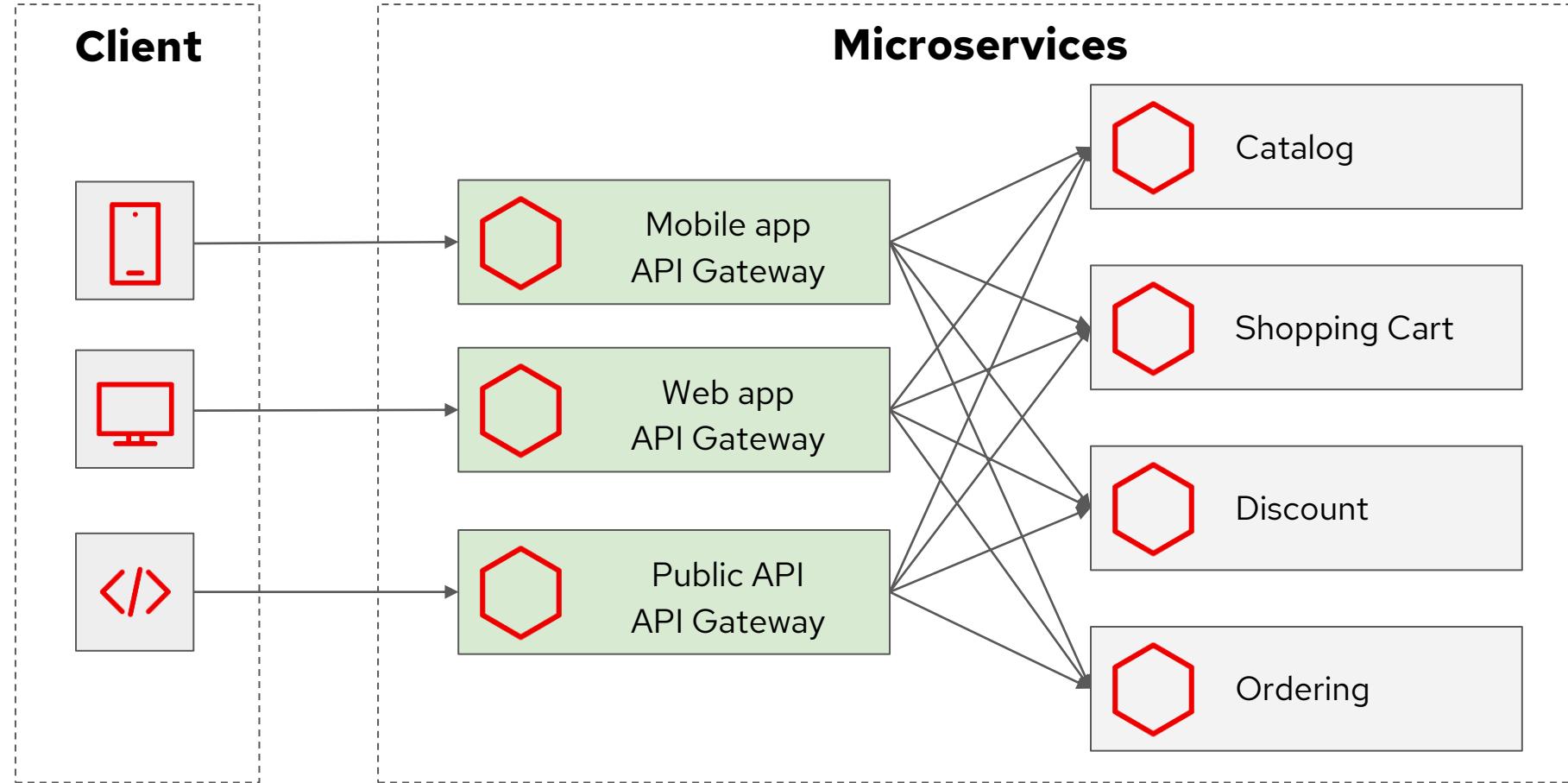
API Gateway 模式

模式	直接呼叫	透過 API Gateway
1.	效率差，須經過多次往返	效率好，只需一次往返
2.	不易管理，內部服務架構異動會影響 APP 設計	好管理，將內部服務架構的細節隔離在內部。必要時也能進行不同版本或格式的 API 轉譯
3.	難以最佳化	有統一的進出端口，容易進行最佳化，API Gateway 能妥善做好 output cache
4.	安全性差，跨服務的溝通細節暴露在外界	安全性佳，不需將不必要的細節傳遞到外面。API Gateway 甚至能負責認證等等問題。

Backends for Frontends (BFF) 模式

服務通訊

因為移動客戶端和 Web 客戶端有不同的螢幕尺寸、顯示器、性能、能耗和網路頻寬，它們的 API 需求不同。面向前端的後端模式適用於需要為特殊 UI 定製單獨後端的場景。它還提供了其他優勢，比如作為下游微服務的封裝，從而減少 UI 和下游微服務之間的頻繁通信。此外，在高安全要求的場景中，BFF 為部署在 DMZ 網路中的下游微服務提供了更高的安全性。





Backends for Frontends (BFF) 模式

優點

- 分離 BFF 之間的關注點，使得我們可以為具體的 UI 優化他們
- 提供更高的安全性
- 減少 UI 和下游微服務之間頻繁的通信

缺點

- BFF 之間代碼重複
- 大量的 BFF 用於其他用戶界面（例如，智能電視，Web，移動端，PC 桌面版）
- 需要仔細的設計和實現，BFF 不應該包含任何業務邏輯，而應只包含特定客戶端邏輯和行為。

何時使用 BFF

- 如果應用程式有多個不同 API 需求
- 應用程式可能潛在 N+1 問題時或 API 調用階層複雜
- 出於安全需要，UI 和下游微服務之間需要額外的隔離層。
- 如果在 UI 開發中使用微前端。

何時不宜使用 BFF

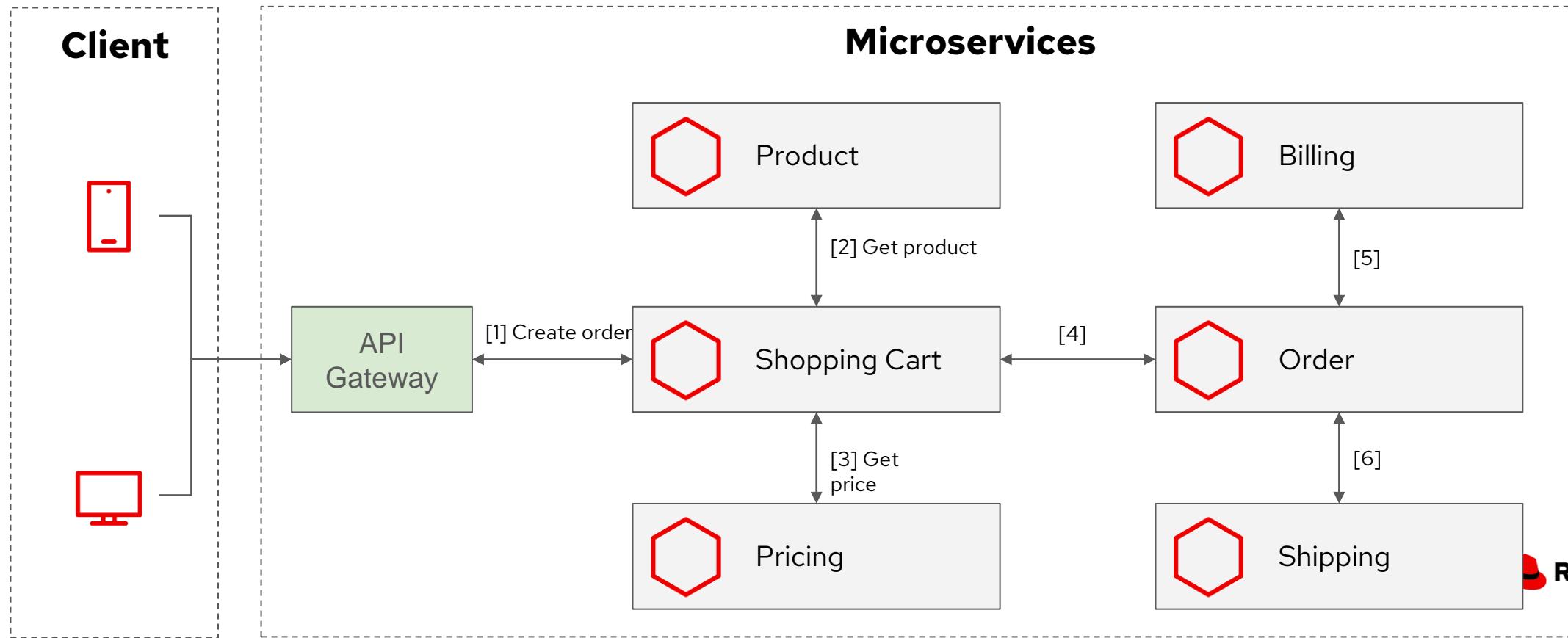
- 如果應用程式雖有多個 UI，但使用的 API 相同。
- 如果核心微服務不是部署在 DMZ 網路中。

鏈式 (Chained Microservice Design Pattern)

服務通訊



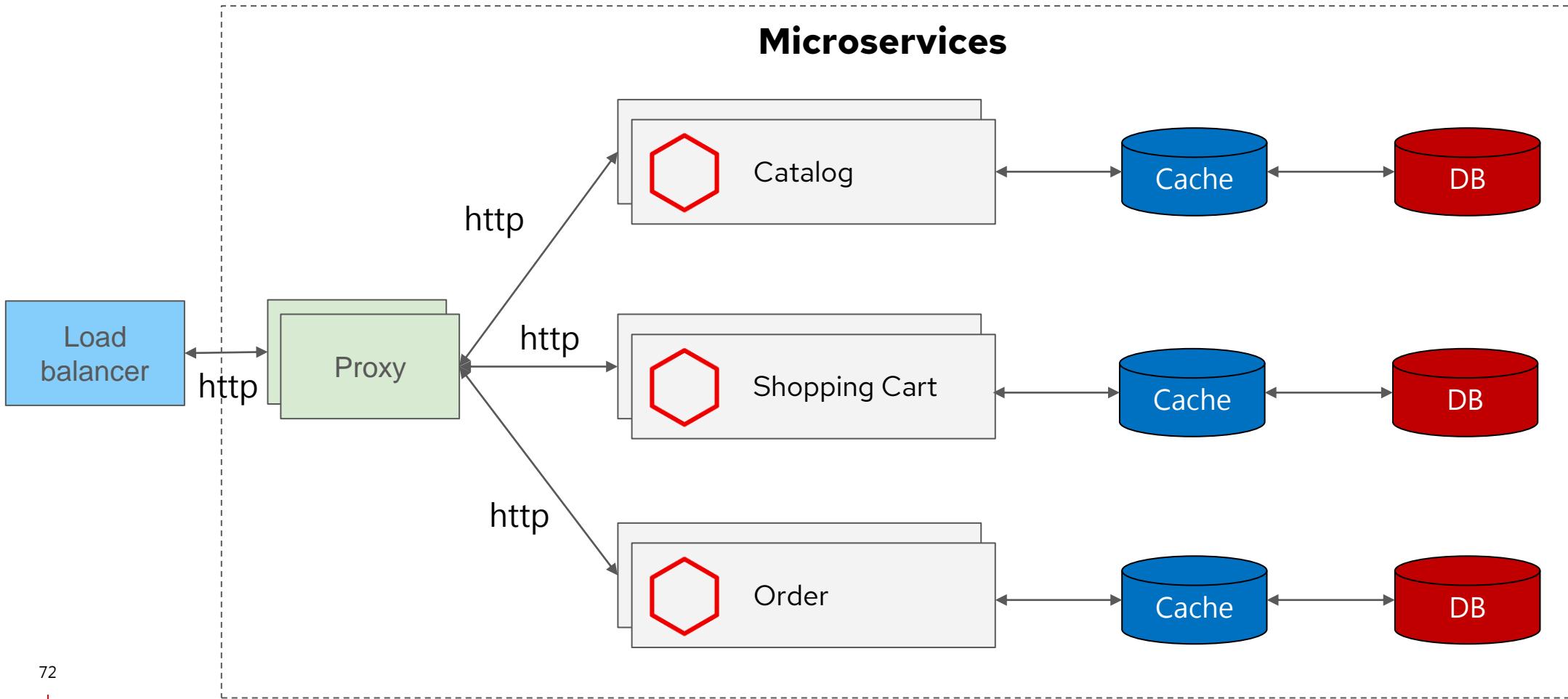
- 一個微服務在提供資料時，資料可以不只是由本身的資料來源所提供，還可以向其它微服務取得資料，而其它的微服務同樣可以再向另外的微服務取得資料，如此，就形成了Chained Microservice Pattern。
- 在這種情況下，服務 A 接收到請求後會與服務 B 進行通信，類似地，服務 B 會同服務 C 進行通信。所有服務都使用同步消息傳遞。在整個鏈式調用完成之前，客戶端會一直阻塞。因此，**服務調用鏈不宜過長**，以免客戶端長時間等待。



代理微服務設計模式 Proxy Pattern

服務通訊

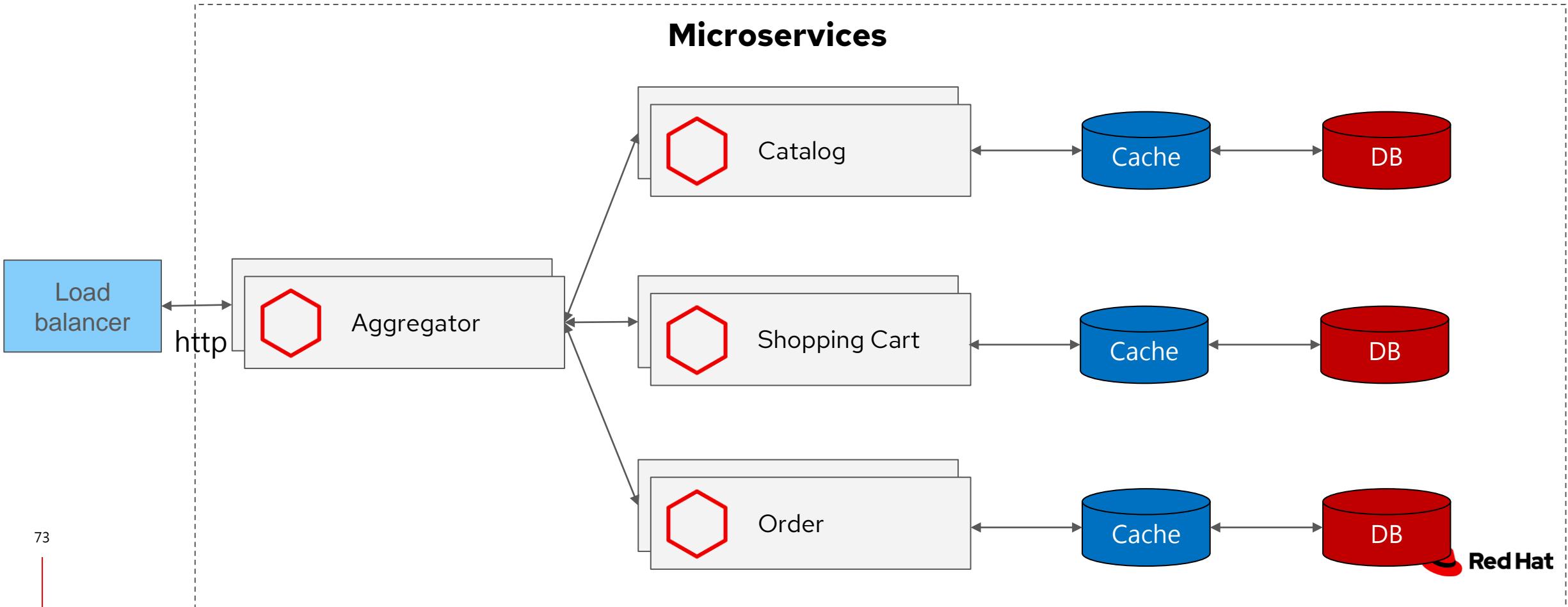
在這種情況下，客戶端並不聚合資料，但會根據業務需求的差別調用不同的微服務。代理可以僅僅委派請求，也可以進行資料轉換、協議轉換等工作。



聚合器微服務設計模式 Aggregator Pattern

服務通訊

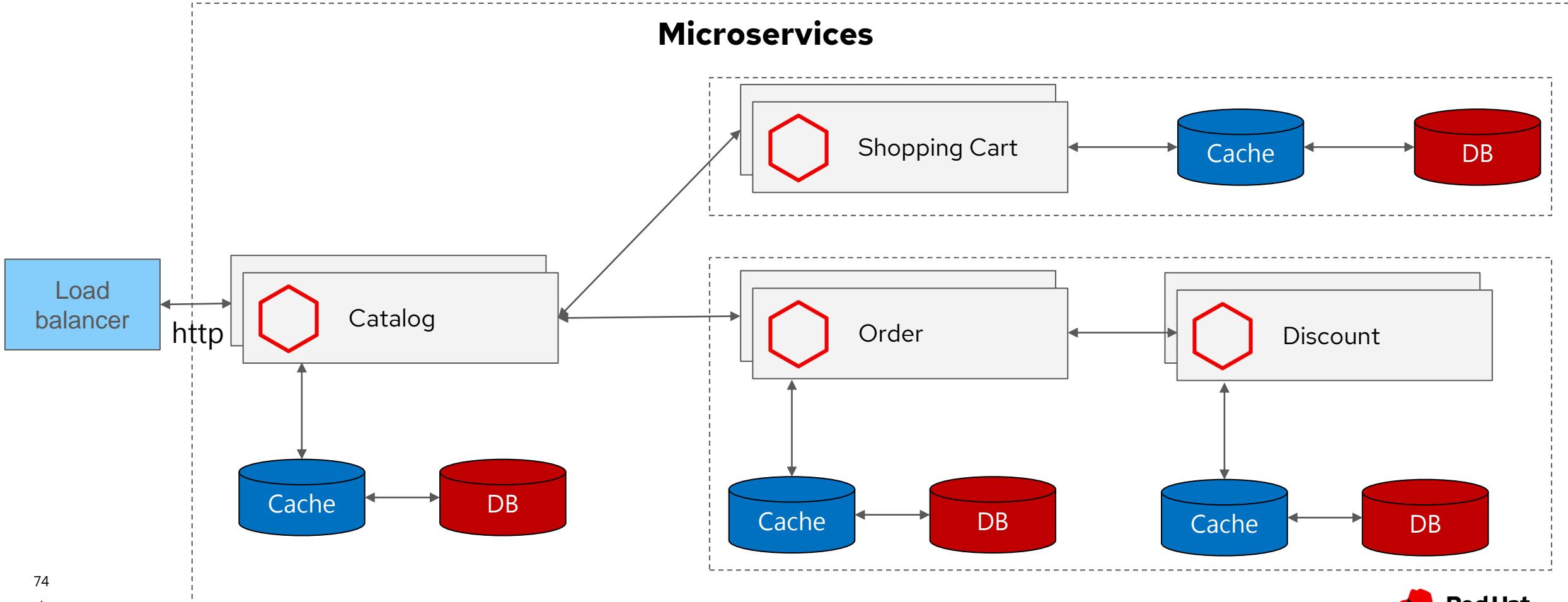
當客戶端需要由服務取得一些資料時(如一個頁面的渲染)，這些資料的資料來源可能包含到數個微服務，並且在取得資料後還需具有邏輯的判斷，此時若由客戶端一一向後端的各微服務發出請求然後再自行處理邏輯判斷，不只效能低下容易出錯，且當有複雜的邏輯需要處理時、資料取得的順序也是個問題，所以若能有一個微服務，聚合了所有需要的資料同時也處理好邏輯判斷，就可以避免可能發生的問題，而這就是Aggregator Pattern。



分支微服務設計模式 Branch Pattern

服務通訊

這種模式是聚合器模式的擴展，允許同時調用兩個微服務鏈，並將結果整合或發派。



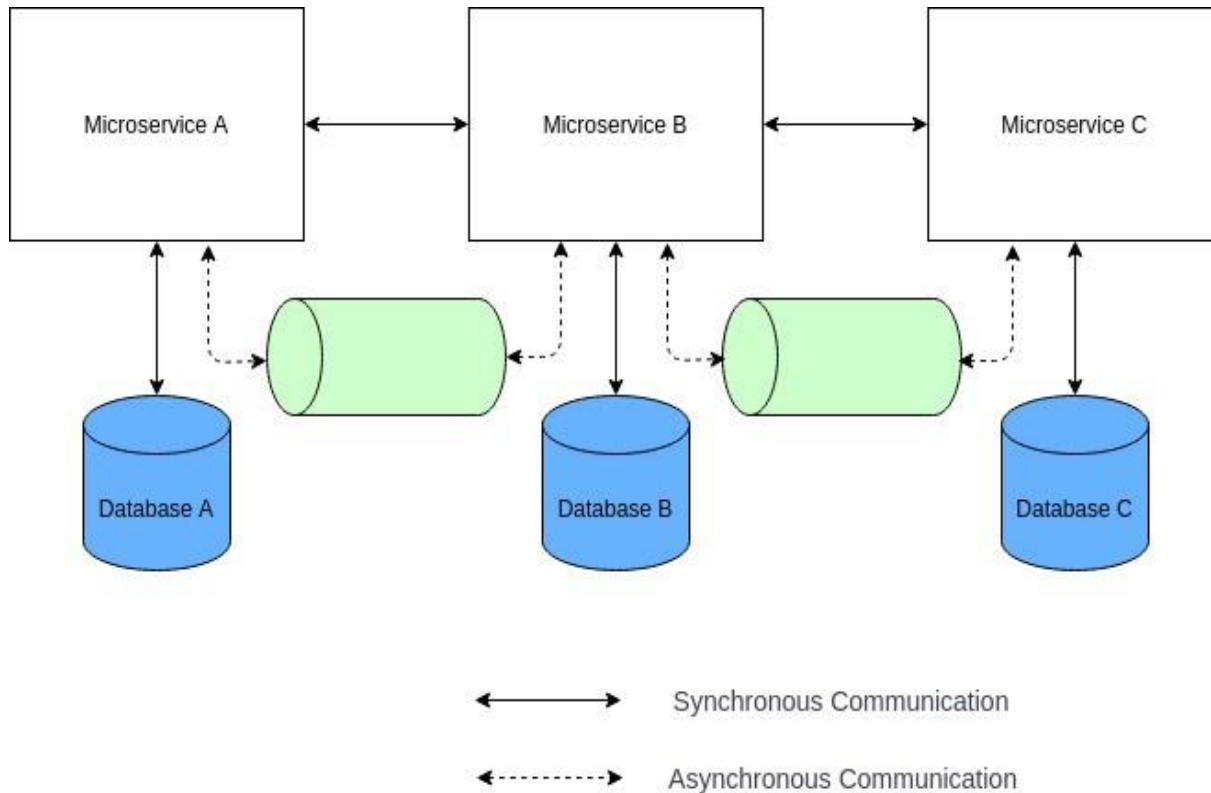
資料設計原則及模 式

獨享資料庫 (Database per Microservice)

資料設計

I O I O
I I O I I I

- 當將大型單體系統替換成一組微服務，首先要面臨的最重要決策是關於資料庫。
- 單體架構會使用大型中央資料庫。即使轉移到微服務架構許多架構師仍傾向於保持資料庫不變。雖然有一些短期收益，但長期下來會大有問題，特別是在大規模系統中，微服務將在資料庫層嚴重耦合，整個遷移到微服務的目標都將面臨失敗（例如，團隊授權、獨立開發等問題）。
- 更好的方法是為每個微服務提供自己的資料存儲，這樣服務之間在資料庫層就不存在強耦合。
- 這裏使用資料庫這一術語來表示邏輯上的資料隔離，也就是說微服務可以共享物理資料庫，但應該使用分開的資料結構、集合或者資料表，這還將有助於確保微服務是按照**領域驅動設計**的方法正確拆分的。



獨享資料庫 (Database per Microservice)

資料設計

IOIO
IIIOII

優點

- 資料由服務完全所有
- 服務的開發團隊之間耦合度降低

缺點

- 服務間的資料共享變得更有挑戰性
- 應用程式的交易如涉及到 ACID 時，難度會提升許多
- 細心設計如何拆分單體資料庫是一項極具挑戰的任務

何時使用獨享資料庫

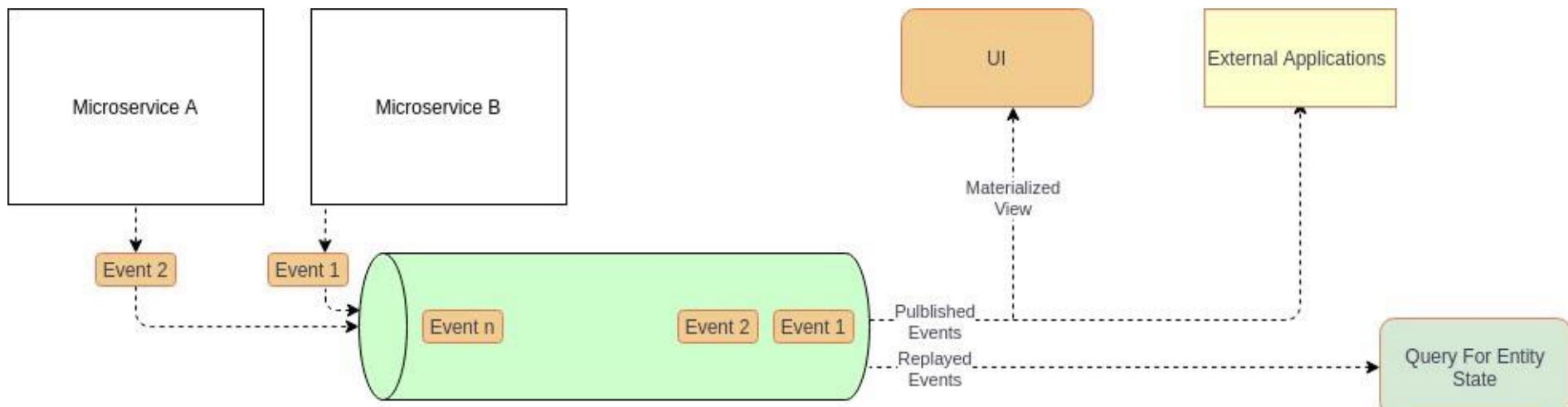
- 在大型企業應用程式中
- 當團隊需要完全把控微服務以實現開發規模擴展和速度提升

何時不宜使用獨享資料庫

- 在小規模應用中

事件溯源 (Event Sourcing)

- 在微服務架構中，特別使用獨享資料庫時，微服務之間需要進行資料交換。對於彈性高可伸縮的和可容錯的系統，它們應該通過**交換事件**進行**非同步**通信。
- 在這種情況，您可能希望進行類似更新資料庫併發送消息這樣的原子操作，如果在大數據量的分佈式場景使用關聯資料庫，您將無法使用兩階段鎖協議 (2 Phase Locking - 2PL)，因為它無法伸縮。而且如果像是NoSQL 資料庫這類大多不支持兩階段鎖協議甚至無法實現分散式交易。
- 在這些場景，可以基於事件的架構使用事件溯源模式。在傳統資料庫中，直接儲存的是業務實體的當前“狀態”，而在事件溯源中任何“狀態”更新事件或其他重要事件都會被儲存起來，而不是直接儲存實體本身。這意味着業務實體的所有改變將被保存為一系列不可變的事件。因為資料是作為一系列事件儲存的，而非直接更新儲存，所以各項服務可以通過重放事件儲存中的事件來計算出所需的資料狀態。



事件溯源 (Event Sourcing)

IOIO
IIIOII

優點

- 自動記錄實體變更歷史，方便進行溯源與歷史重現
- 能提供非常好的性能
- 鬆耦合和事件驅動的微服務

缺點

- 從事件存儲中讀取實體成為新的挑戰，通常需要額外的資料儲存（CQRS 模式）。
- 系統整體複雜性增加了，通常需要領域驅動設計協助規劃
- 系統需要處理事件重覆或丟失
- 變更事件結構成為新的挑戰。使用Event Sourcing模式，由於業務的變化，我們設計的事件，在結構上可能有一些改變，可能需要添加一些資料，或者刪除一些資料。那麼這時候，想要進行“歷史重現”就會有問題。這時我們就需要通過某種方式來提供相容。

何時使用事件溯源

- 使用 NoSQL 類型的資料庫
- 彈性高可伸縮微服務架構
- 方便使用資料分析
- 典型的消息驅動或事件驅動系統（電子商務、預訂和預約系統）

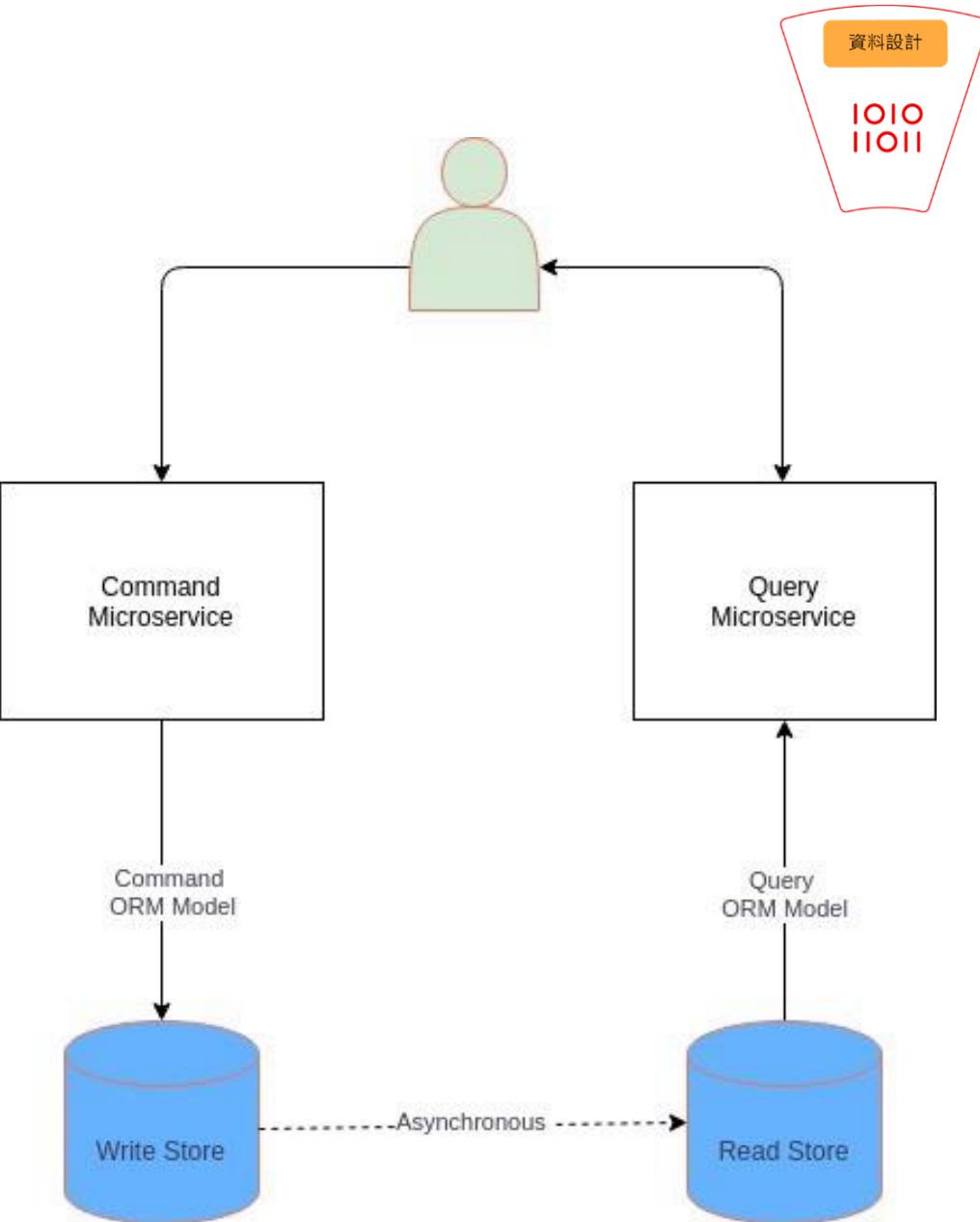
何時不宜使用事件溯源

- 傳統RDBMS需要考量效能和擴展性是否適合
- 如果你的系統有大量的CRUD的業務，那麼就不適合使用Event Sourcing模式
- 在服務可以同步交換資料（例如：通過 API ）的簡單微服務架構中。

命令和查詢職責分離 (CQRS)

資料設計

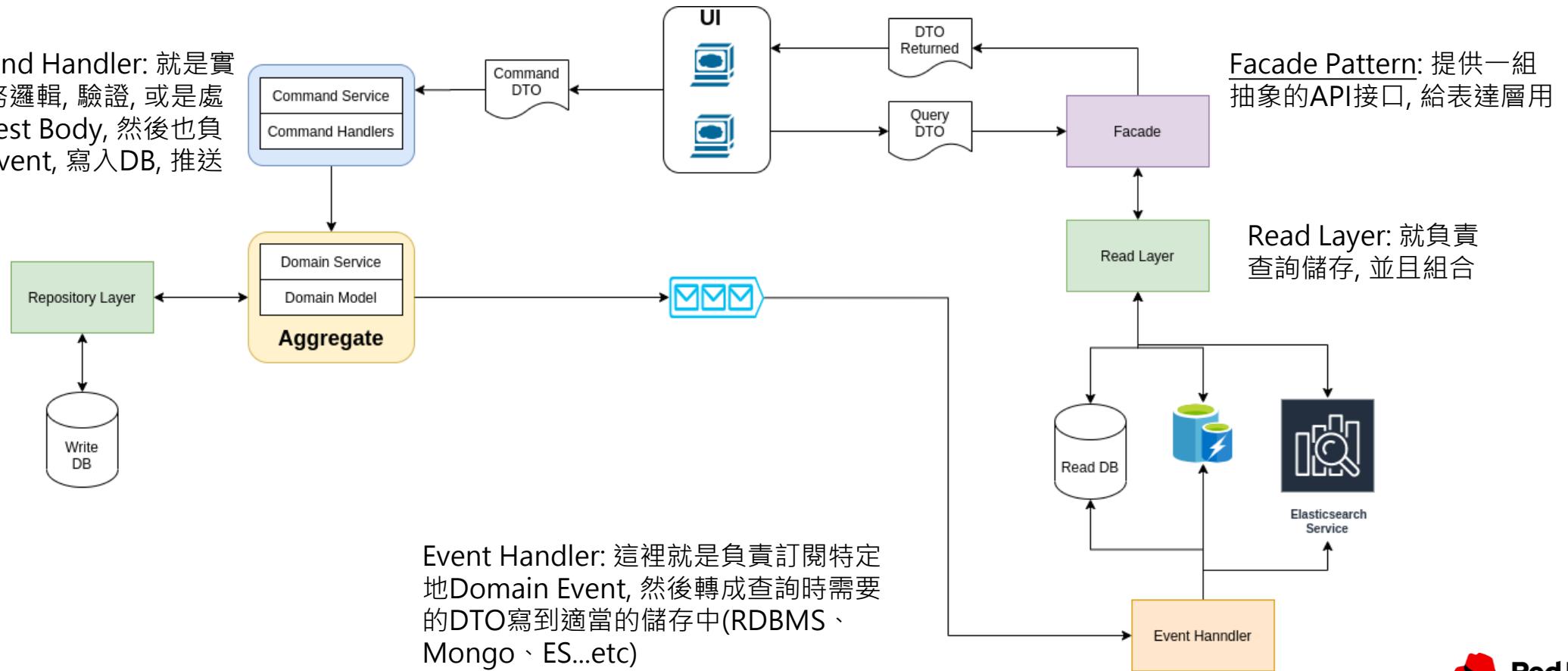
- 如果我們使用事件溯源，那麼從事件存儲中讀取資料就變得困難了。要從資料存儲中獲取實體，我們需要處理所有的實體事件。有時我們對讀寫操作還會有不同的一致性和吞吐量要求。
- 這種情況，我們可以使用 CQRS 模式。在該模式中，系統的資料修改部分（命令）與資料讀取部分（查詢）是分離的。而 CQRS 模式有兩種容易令人混淆的模式，分別是簡單的和高級的。
- 在簡單模式中，不同實體或 ORM 模型被用於讀寫操作



命令和查詢職責分離 (CQRS)

- 在進階模式中，會有不同的資料存儲用於讀寫操作。進階的 CQRS 通常結合事件溯源模式。根據不同情況，會使用不同類型的寫資料存儲和讀資料存儲。寫資料存儲是“記錄的系統”，也就是整個系統的核心源頭。

Command Handler: 就是實際的業務邏輯, 驗證, 或是處理Request Body, 然後也負責轉成Event, 寫入DB, 推送到MQ



命令和查詢職責分離 (CQRS)



優點

- 寫入性能更好(畢竟讀寫算是分離了)
- 讀取性能更好(因為會用適當的儲存做查詢)
- 歷史紀錄追蹤
- 滿足單一職責, 因為都滿足了對各自的業務做封裝
- 兩邊允許各自擴展
- 查詢端的儲存已經是存查詢所返回的DTO了, 不必再做Model->DTO的轉換

缺點

- 讀資料儲存是弱一致性的 (資料是從寫存儲非同步複製到讀存儲中的 , 所以**讀儲存和寫儲存之間會有延遲** , **但最終是一致的**。)
- 整個系統的複雜性增加了 , 混亂的 CQRS 會顯着危害整個專案。
- 很需要DDD的相關領域知識

何時使用 CQRS

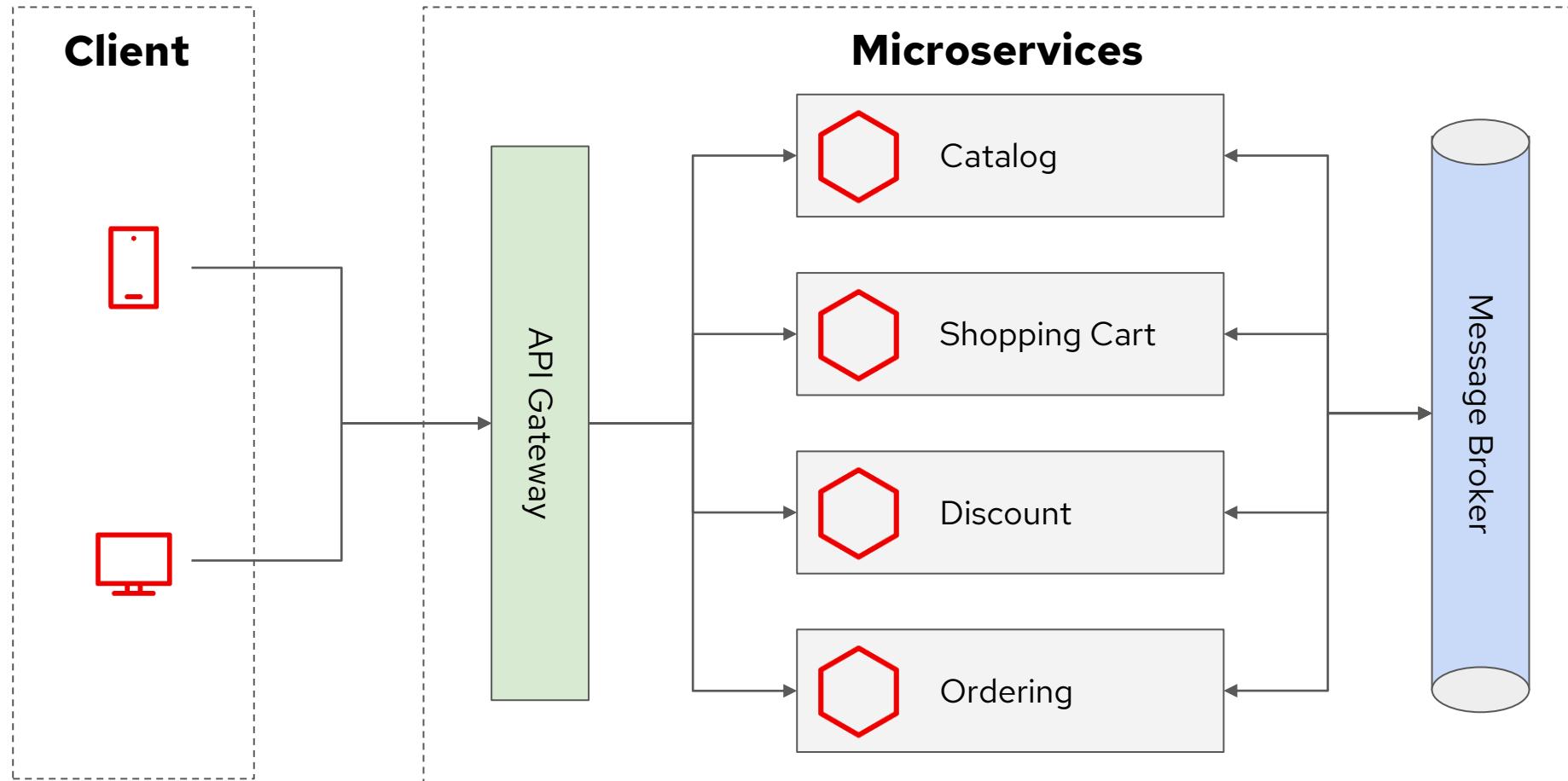
- 在高可擴展的微服務架構中使用事件溯源
- 在複雜領域模型中 , 讀操作需要同時查詢多個資料存儲。
- 在讀寫操作負載差異明顯的系統中

何時不宜使用 CQRS

- 在沒有必要存儲大量事件的微服務架構
- 在讀寫操作負載相近的系統中

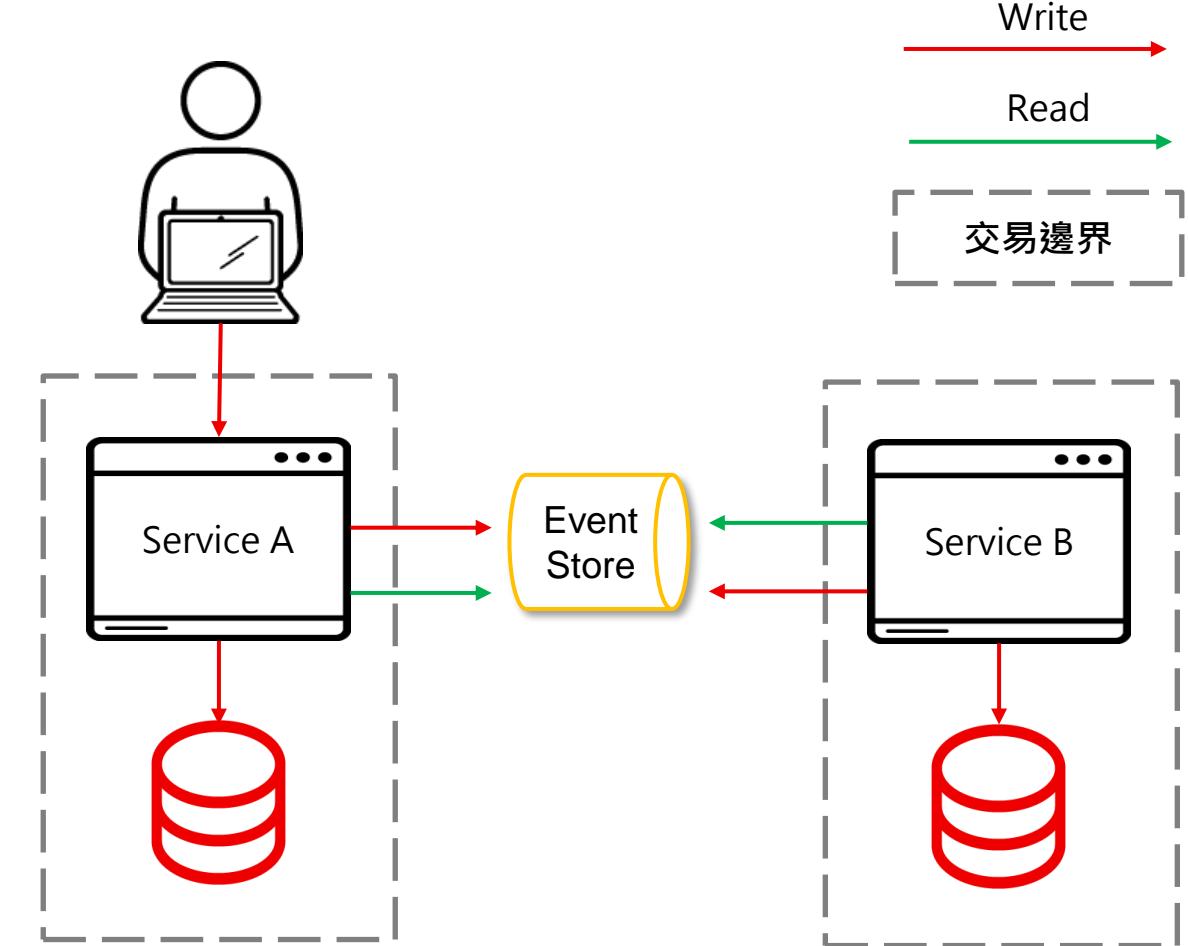
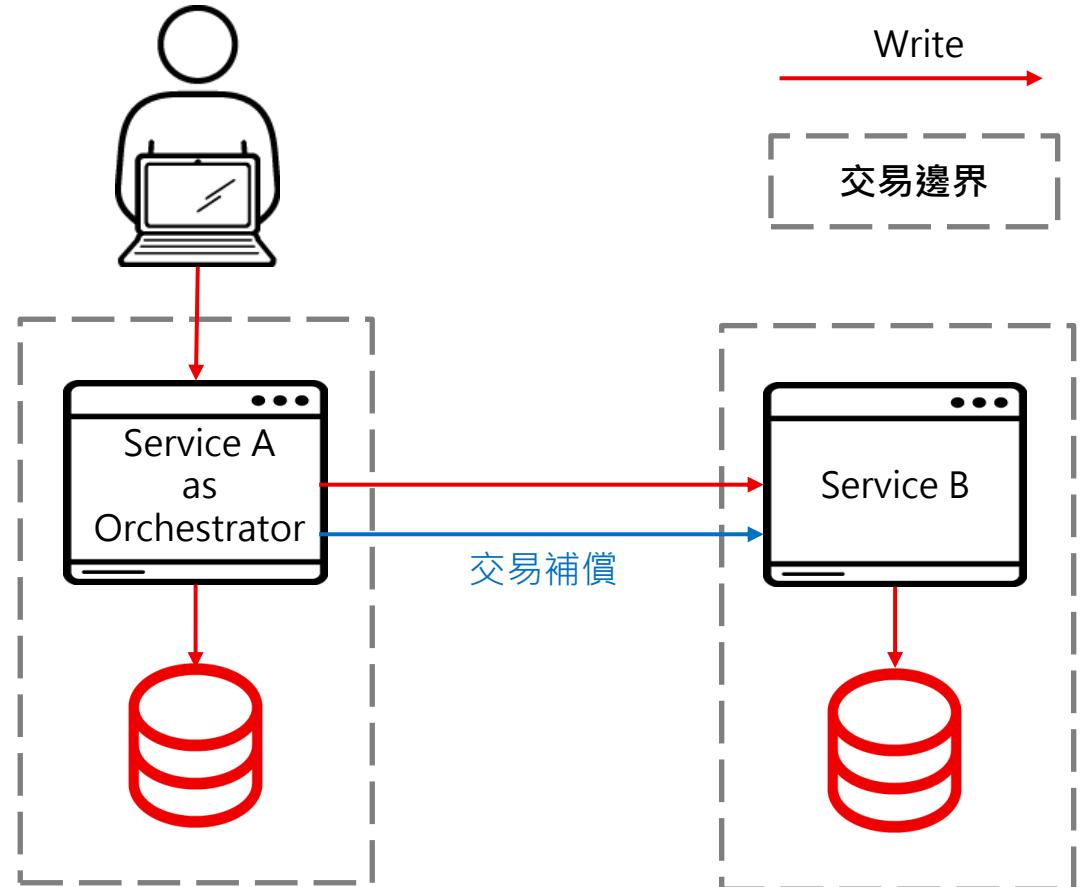
服務協同模式 (Choreography Pattern)

- 服務協同模式 (Service Choreography) 是一種微服務架構中的設計模式，它不使用一個中央的服務編排器 (Service Orchestrator) 來協調和管理其他服務的互動和流程，而是讓每個服務自行決定如何與其他服務溝通和合作。服務協同模式可以實現分散式的商務邏輯，並且提高服務的自治性和彈性。



Orchestration vs. Choreography

資料設計



Write

Read

交易邊界

Orchestration vs. Choreography

資料設計

IOIO
IIIOII

Orchestration

Choreography

資料源	異質資料源	異質資料源
控制節點	集中控制	分散控制
執行流	<ul style="list-style-type: none">連續性部分解偶	<ul style="list-style-type: none">連續性部分解偶
資料屬性	<ul style="list-style-type: none">ACDnon-blockingasyncsync	<ul style="list-style-type: none">ACDnon-blockingasyncsync
實踐方式	<ul style="list-style-type: none">SAGAOutbox	<ul style="list-style-type: none">SAGAOutboxEvent sourcing

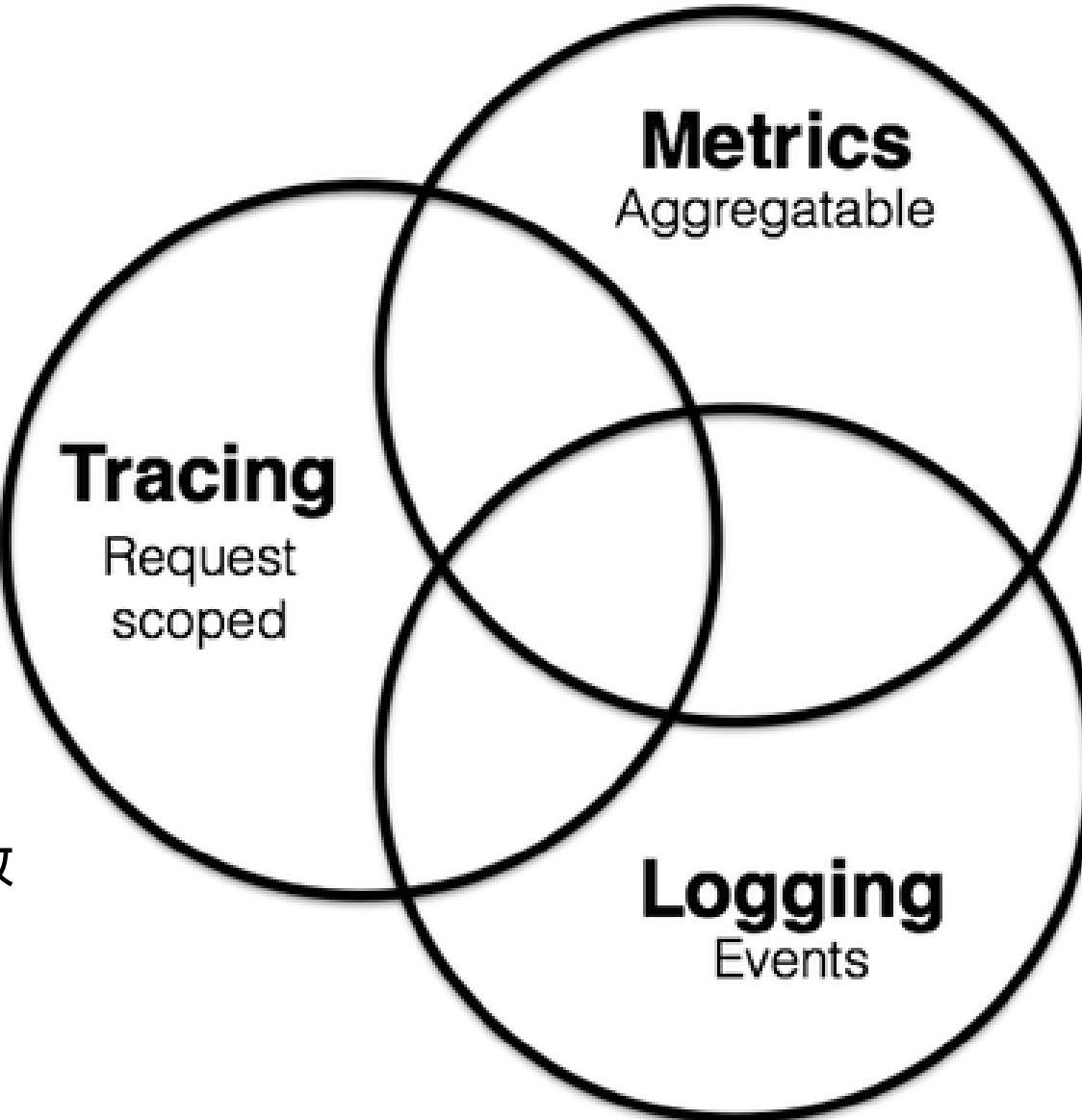
可觀察性實踐

容器應用的問題排查面向

可觀察性



- 宏觀
- 資料量少



- 採樣分析
- 適合問題排查
- 通常須配合程式修改

- 仔細記錄
- 資料量大

排查工具比較

可觀察性



Metrics

- 資料量最少，僅以數值的形式保存
- 適合用來進行報表產出、統計分析

Logging

- 資料量最大，格式長度、形式沒有統一的標準，但是仍建議進行格式正規化
- 常搭配 Parser 、 Log Rotate 來進行管理分析
- 資料完整，可以完全做到與其他兩項相同的功能 (但是效能低下)

Tracing

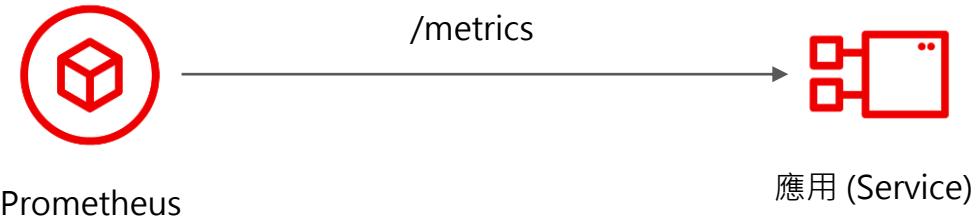
- 常採用採樣的方式，每筆 Trace 會有唯一的 ID 以進行追蹤 (常放於 HTTP Header 、 kafka Header)
- 透過採樣率 (Sampling Rate) ，控制數據量 (常見為 1/100 、 1/1000)
- 大部分情況，都需要修改程式才能使用 (或是用封包擷取的方式，但仍不精確)

可觀察性 - 指標監控

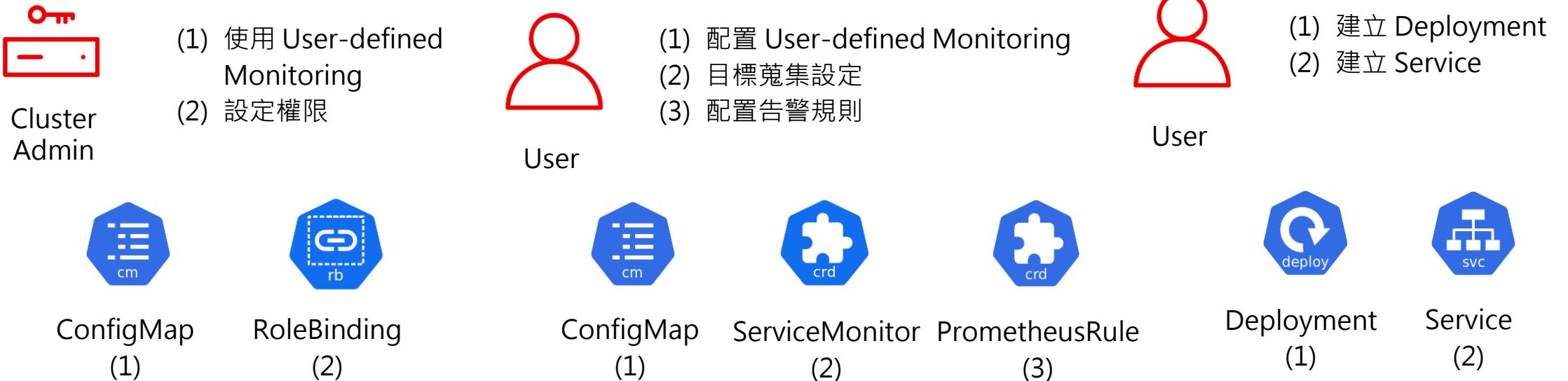
監控指標配置概覽

可觀察性

使用 Label 做為服務發現機制，偵測並
蒐集 Service 端口的 /metrics



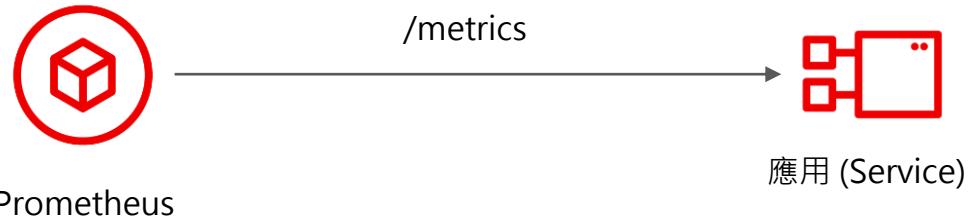
應用透過 **Prometheus Exporter** 的 HTTP
接口將指標 [1]



透過 ServiceMonitor 物件發現 Service

可觀察性

ServiceMonotor 中的 label 來偵測
Service 物件 [1]



在 Service 中定義與
ServiceMonitor 相同的 label



ServiceMonitor

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: prometheus-example-monitor
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
  - interval: 30s
    port: web
    scheme: http
  selector:
    matchLabels:
      app: prometheus-example-app
```



Service

[2]

[3]

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: prometheus-example-app
  name: prometheus-example-app
  namespace: ns1
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    name: web
  type: ClusterIP
```

[1] 配置範例參考 <https://docs.openshift.com/container-platform/4.12/monitoring/managing-metrics.html>

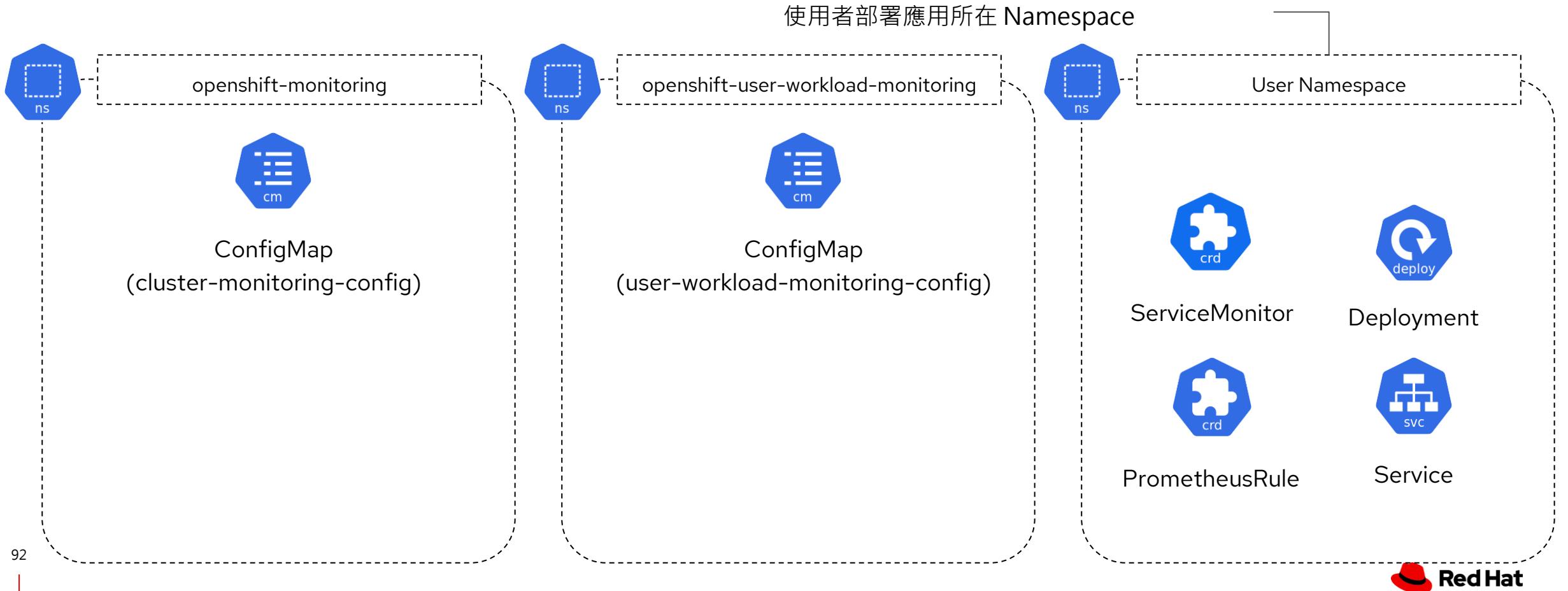
[2] 對應 ServiceMonitor 's spec.selector.matchLabels and Service' s metadata.labels 中的 Key/Value labels

[3] Specify the port name of the Service to which Prometheus connects. 當指定 port 時，使用 targetPort 而非 port

Configmap 和 resources 的對應關係

可觀察性

- ConfigMap 和 ServiceMonitor 所部署之 Namespace 需正確配置

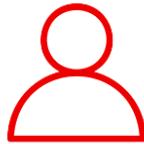


配置 User-defined Monitoring (ConfigMap)

可觀察性

- 定義Prometheus Operator, Prometheus, Thanos Ruler 資源
- 配置 Node Selector, Toleration, Request, Disk (PV), etc.

[1]



- (1) 配置 User-defined Monitoring
- (2) 目標蒐集設定
- (3) 配置告警規則

User



ConfigMap

(1)



ServiceMonitor

(2)



(3)

```
kind: ConfigMap
apiVersion: opentelemetry.io/v1alpha1
metadata:
  name: user-workload-monitoring-config
  name: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheusOperator:
      nodeSelector:
        <Key>: <value>
    prometheus:
      resources:
        requests:
          cpu: 200m
    thanosRuler:
```

目標蒐集設定

可觀察性



- (1) 配置 User-defined Monitoring
- (2) 目標蒐集設定
- (3) 配置告警規則

User



ConfigMap

(1)



ServiceMonitor

(2)

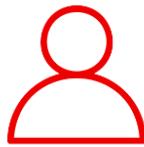


(3)

```
kind: ServiceMonitor
apiVersion: monitoring.coreos.com/v1
metadata:
  labels:
    k8s-app: prometheus-example-monitor
    name: prometheus-example-monitor
    namespace: ns1
spec:
  endpoints:
    - interval: 30s
      port: web
      scheme: http
  selector:
    matchLabels:
      app: prometheus-example-app
```

配置告警規則

可觀察性



User

- (1) 配置 User-defined Monitoring
- (2) 目標蒐集設定
- (3) 配置告警規則

可定義告警規則 [1]



ConfigMap

(1)



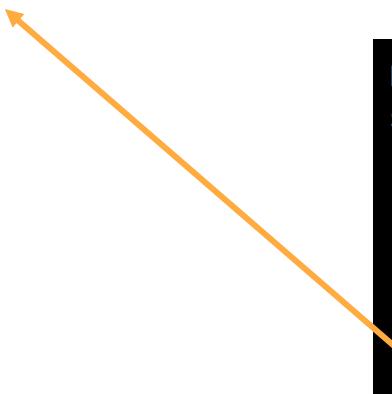
ServiceMonitor

(2)



PrometheusRule

(3)



```
kind: PrometheusRule
spec:
  groups:
    - name: general.rules
      rules:
        - alert: HTTPRequest
          annotations:
            message: '{{ printf "%.4g" $value }}...'
          expr: http_requests_total{job="example-app"} > 20
          for: 10m
          labels:
            severity: warning
```



PromQL

- PromQL 為查詢 Metrics 的語法，這些語法可以用於未來的視覺化以及告警使用，運算子與語法參照下表：

No	算數運算子	說明	No	比較運算子	說明	No	常用統計函數	說明
1	+	加	1	==	等於	1	sum()	取得 Metrics 合計值
2	-	減	2	!=	不等於	2	min()	取 Metrics 最小值。
3	*	乘	3	<	小於	3	max()	取 Metrics 最大值。
4	/	除	4	>	大於	4	count()	計算 Metrics 筆數。
5	%	求餘數	5	<=	小於等於	5	rate()	合計時間區間內的次數，平均成每秒幾次。
6	^	次方	6	>=	大於等於	6	increase()	計算時間區間內的變化量。

算數運算子

比較運算子

常用函數，更多請參照官網 [1]

- 範例：查詢 microservices-demo namespace 中 inventory-service pod 的 cpu 使用狀況

```
$ sum(namespace_pod_name_container_name:container_cpu_usage_seconds_total:sum_rate{namespace="microservices-demo",pod_name="inventory-service", resource="cpu"})
```

可觀察性 - 日誌搜尋

LogQL

可觀察性

- LogQL 查詢像是作為聚合日誌的一個分散式 “grep” 命令

```
{cluster="us-central1", namespace="dev"} |= "err" |~ "timeout|cancel" | json | response_latency > 10s
```



Label Matcher

OR

Log Stream Selector

Line Filter

OR

Log Pipeline

Label Filter

日誌轉發

- OpenShift 日誌子系統提供轉發日誌的功能，可以透過 ClusterLogForwarder 將日誌轉發至外部 Log Store (如：Elasticsearch、Kafka ... 等)，其原理為透過 CR的異動修改 Vector 的行為
- 在 CRD ClusterLogForwarder 中可自定義設定

可以指定多組外部的 Log Store

預設有 application、infrastructure、audit 三種日誌類型，亦可透過篩選的方式自定義某些 Project 下的日誌

可以任意組合 Input 與 Output，將 Input 的資料轉發至 Output

```
kind: ClusterLogForwarder
apiVersion: logging.openshift.io/v1
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputs:
    - name: elasticsearch
      type: "elasticsearch"
      url: http://elasticsearch.insecure.com:9200
    - name: kafka
      type: "kafka"
      url: tls://kafka.secure.com:9093/app-topic
  inputs:
    - name: my-app-logs
      application:
        namespaces:
          - my-project
  pipelines:
    - name: all-to-default
      inputRefs:
        - application
        - audit
        - infrastructure
      outputRefs:
        - default
    - name: my-project-forward
      inputRefs:
        - application
        - infrastructure
      outputRefs:
        - kafka
        - elasticsearch
```



日誌留存

可觀察性



- OpenShift 日誌子系統之 Log Store - Loki，包含自動清理日誌的功能
- 在 CRD LokiStack 中可自定義設定

全域設定，在沒設定保留天數的話保留多少天的日誌。圖中範例為預設所有日誌皆保留 20 天

依照日誌類別 (application、infrastructure、audit) 設定

依照規則設定，透過選擇日誌之標籤讓某些日誌的保留天數能夠覆蓋日誌類別、全域設定的設定值

```
kind: LokiStack
apiVersion: loki.grafana.com/v1
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  limits:
    global:
      retention:
        days: 20
    tenants:
      application:
        retention:
          days: 14
      streams:
        - days: 14
        selector: '{Kubernetes_namespace_name="my-project"}'
    infrastructure:
      retention:
        days: 7
    audit:
      retention:
        days: 7
  managementState: Managed
```

可觀察性 - 分散式 追蹤

調用鏈基本概念

可觀察性

Trace :

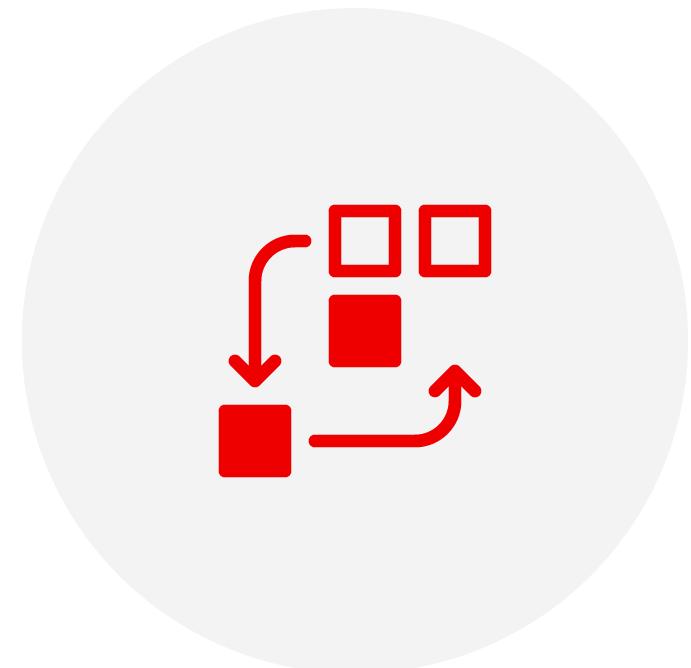
- 跨分散式系統記錄請求
- 表示為跨度樹或 DAG(Directed Acyclic Graph) (有向無環圖 - 與 git commit tree 一樣)
- 一條 Trace (調用鏈) 可以被認為是一個由多個 Span 組成的

進階操作 :

- Trace 中的連續工作段，每個Span就是一次服務調用資訊
- 有名字和時間
- 通過父子關係等與其他 Span 進行交互。

進階操作 :

- Trace 的第一個跨度
- Root Span 時間代表或非常接近整個 Trace 的時間



Trace and Span

可觀察性

分散式服務鏈路追蹤的基本概念

Trace : 關聯使用者從請求發動到結束後所經過的操作

Span : 表示一個工作單元或操作單元

Root Span



分散式追蹤的概念

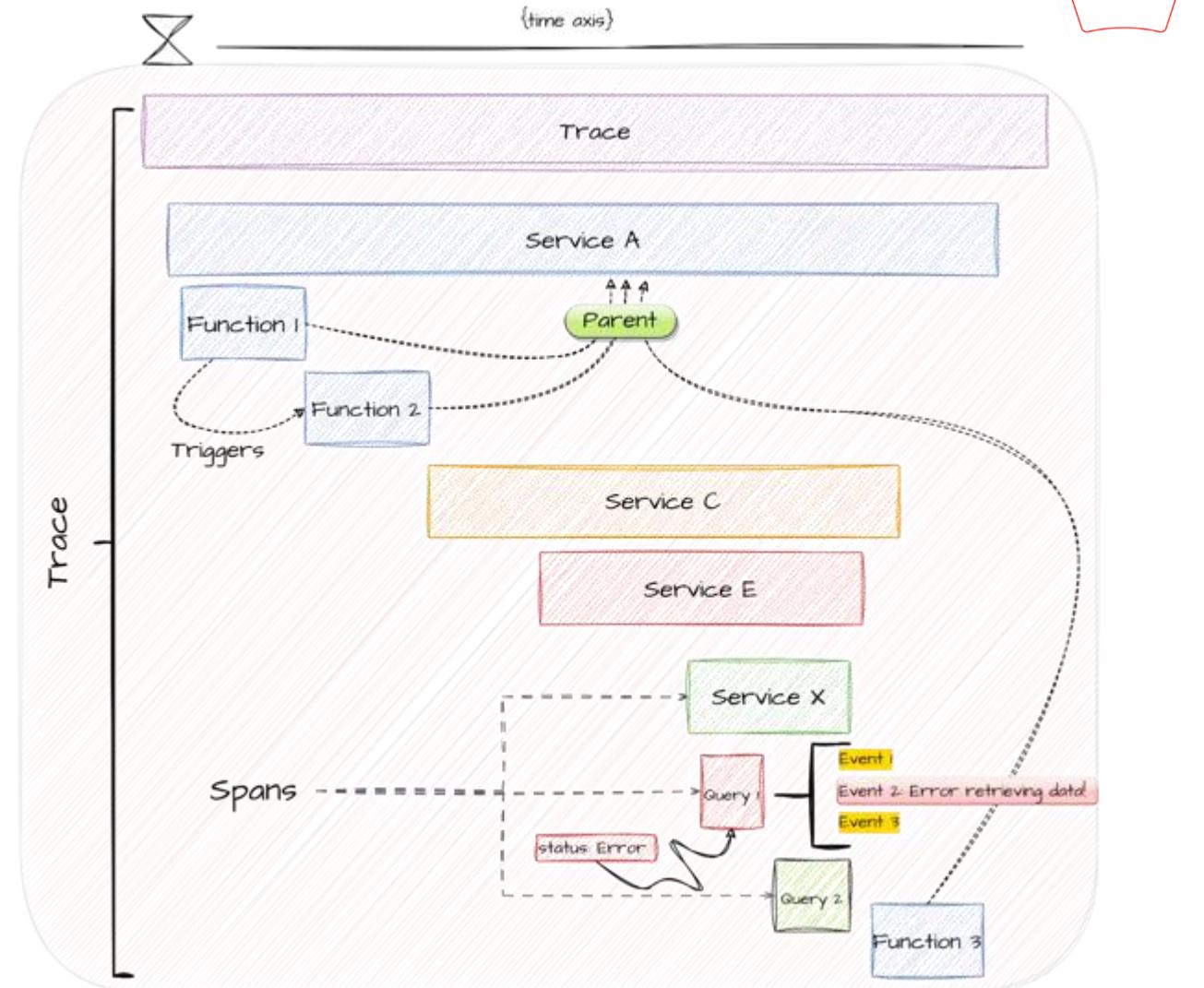
可觀察性

Root Span :

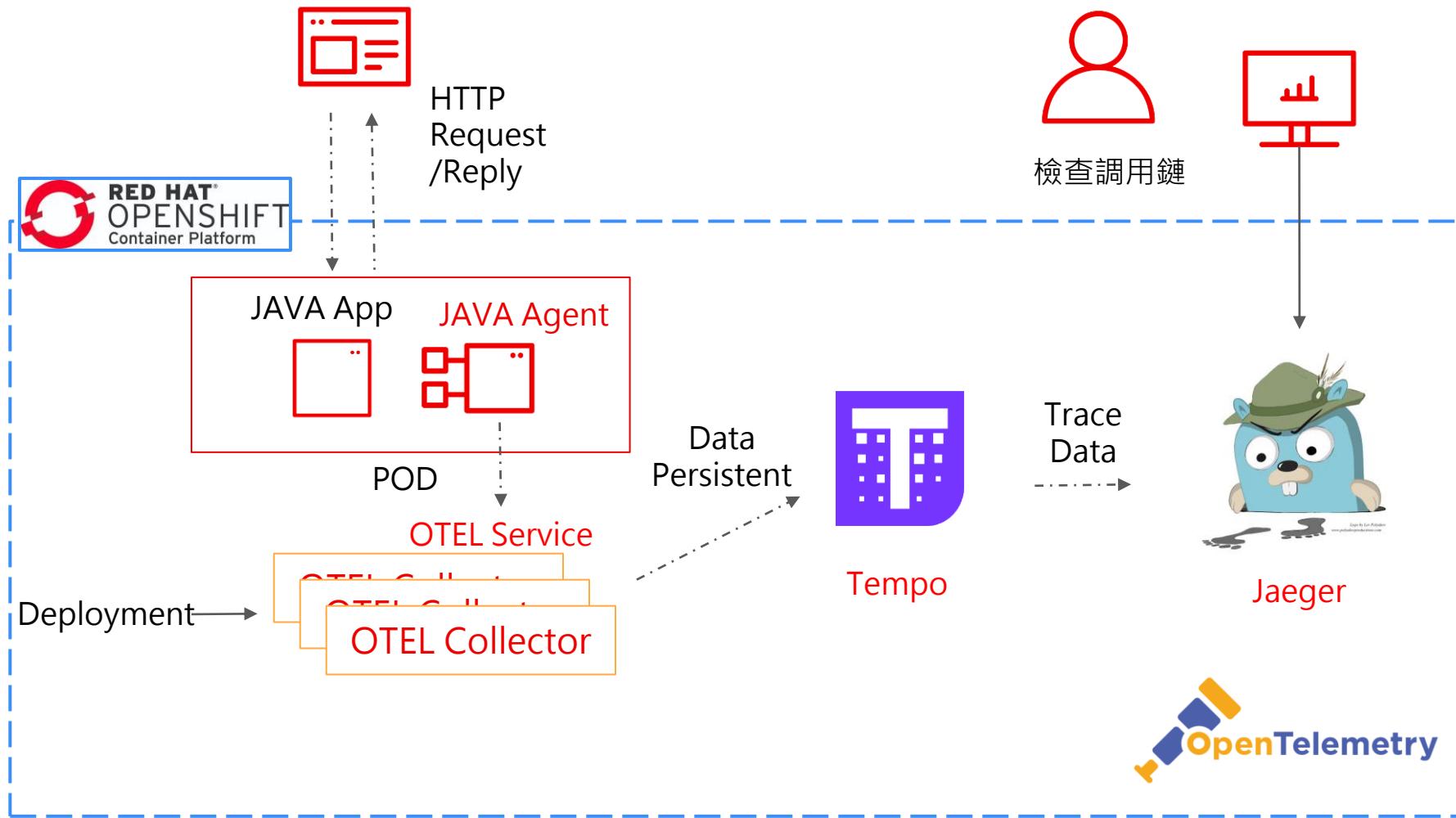
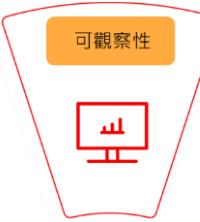
- Trace 第一個Span
- Root Span 時間代表整個交易的時間

Context Propagation :

- Tracing由上下文傳播組成在跨服務請求中，路徑上唯一的識別標識



Deployment Mode



建立 OpenTelemetry 範例

可觀察性



```
kind: OpenTelemetryCollector
apiVersion: opentelemetry.io/v1alpha1
metadata:
  name: otel
spec:
  mode: deployment
  observability:
    metrics:
      enableMetrics: true
  config: |
    receivers:
      otlp:
        protocols:
          grpc:
            endpoint: 0.0.0.0:4317
    exporters:
      otlp:
        endpoint: tempo-sample-distributor:4317
        tls:
          insecure: true
    service:
      pipelines:
        traces:
          receivers: [otlp]
          processors: []
          exporters: [otlp]
```

Collector 連線位置

```
kind: Instrumentation
apiVersion: opentelemetry.io/v1alpha1
metadata:
  name: java-instrumentation
spec:
  env:
    - name: OTEL_EXPORTER_OTLP_TIMEOUT
      value: "20"
  exporter:
    endpoint: http://otel-collector:4317
  propagators:
    - tracecontext
  sampler:
    type: parentbased_traceidratio
    argument: "1"
  java:
    env:
      - name: OTEL_JAVAAGENT_DEBUG
        value: "true"
```

Tempo 儲存連線位置

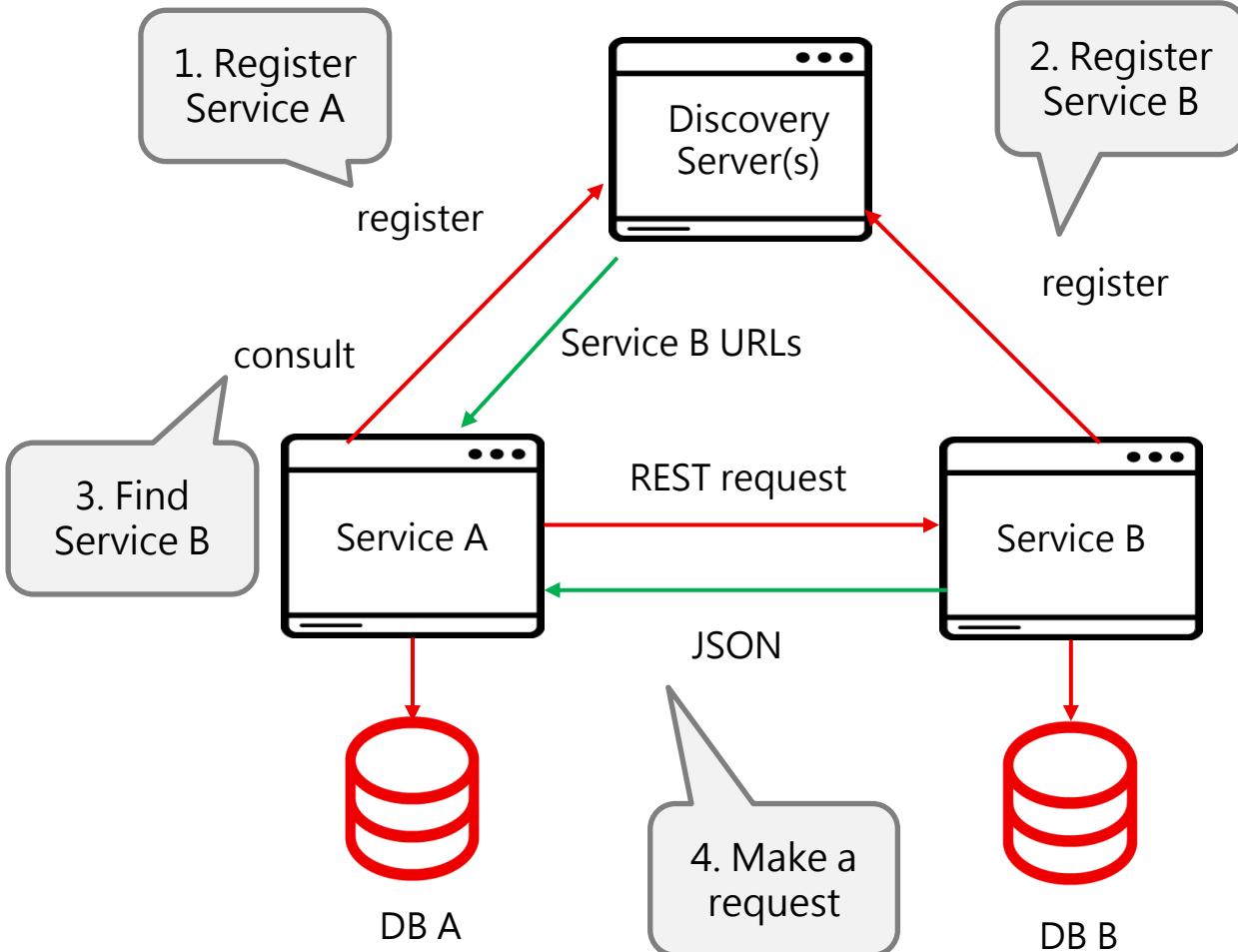
Demo 02 - Microservices OpenTelemetry

可部署性原則及模 式

服務發現模式

可部署性

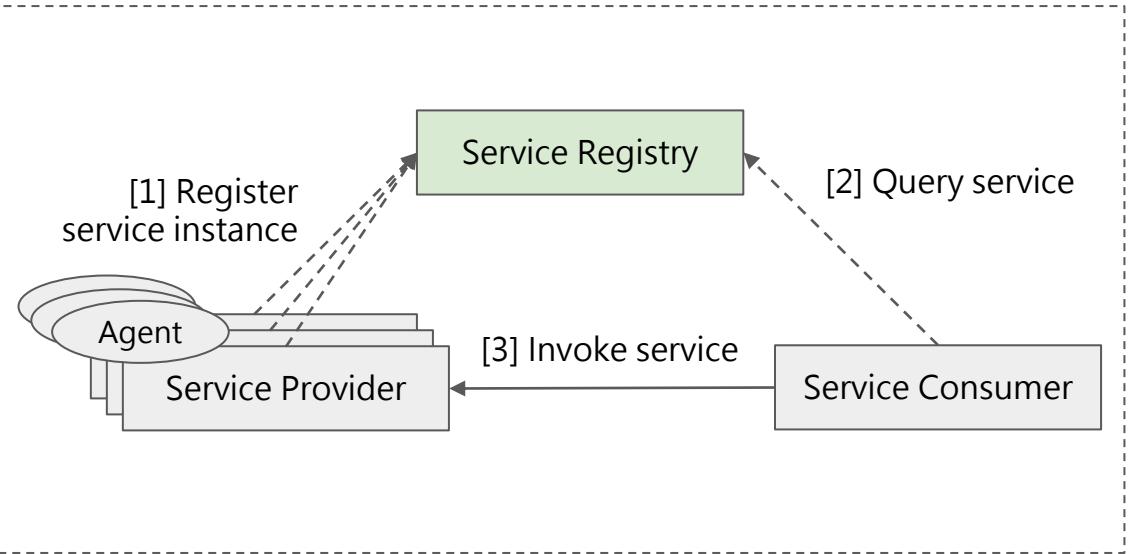
- 在微服務出現時，調用服務方面成為一個必要技術問題。藉助容器技術，IP 地址可以動態地分配給服務實例。每次地址更改時，消費端服務都會中斷並且需要手動更改。
- 對於消費端服務來說，它們必須記住每個上游服務的 URL，這就變成緊耦合了。
- 為此，需要創建一個服務註冊中心，該註冊表將保留每個生產者服務的元資料和每個服務的配置。服務實例在啟動時應當註冊到註冊中心，而在關閉時應當註銷。



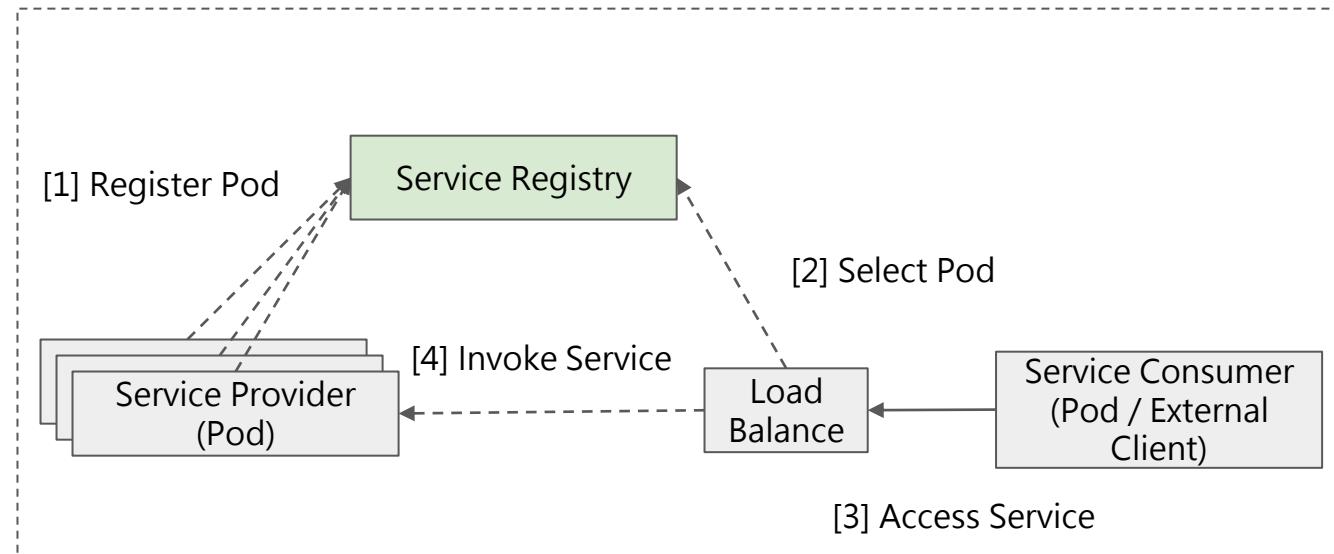
兩種服務發現模式



Client-side Service Discovery



Server-side Service Discovery (Kubernetes Example)

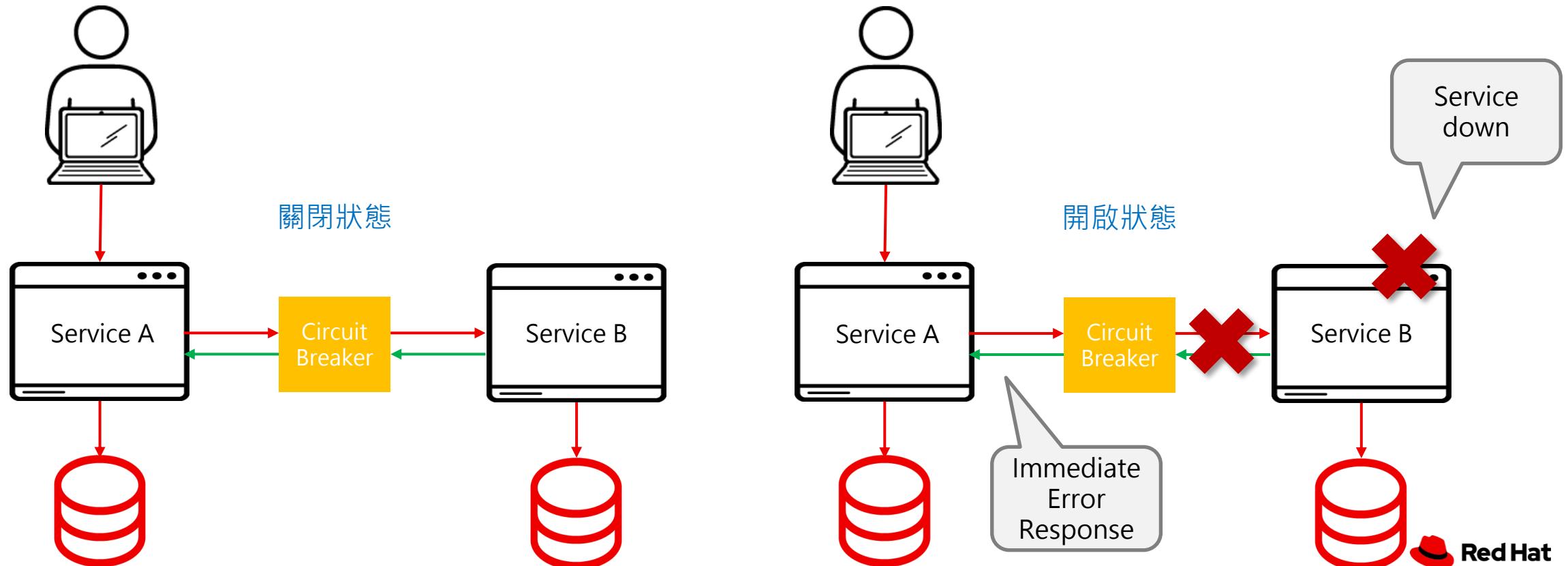


斷路器模式

可部署性



- 同步微服務調用會由於短暫的故障導致失敗，這種情況重試可以解決問題。然而，如果出現了嚴重問題，那麼微服務將長時間不可用，這時重試沒有意義且浪費寶貴的資源（線程被阻塞，CPU 被浪費）。此外，一個服務的故障還會引發整個應用系統的級聯故障，造成雪崩效應。這時快速失敗是一種更好的方法。
- 在這種情況，可以使用斷路器模式挽救。一個微服務通過代理請求另一個微服務，其工作原理類似於電器斷路器，代理通過統計最近發生的故障數量，並使用它來決定是繼續請求還是單純直接返回異常。



斷路器模式

可部署性



斷路器可以有以下三種狀態：

1. **關閉**：斷路器將請求路由到微服務，並統計給定時段內的故障數量，如果超過閾值，它就會觸發並進入打開狀態。
2. **打開**：來自微服務的請求會快速失敗並返回異常。在超時後，斷路器進入半開啟狀態。
3. **半開**：只有有限數量的微服務請求被允許通過並進行調用。如果這些請求成功，斷路器將進入閉合狀態。如果任何請求失敗，斷路器則會進入開啟狀態。

優點

- 提高微服務架構的容錯性和彈性
- 阻止引發其他微服務的連鎖故障

缺點

- 需要複雜的異常處理
- 日誌和監控
- 仍有須人工介入復原的可能

何時使用斷路器

- 在微服務間使用同步通信的緊耦合的微服務架構中
- 如果微服務依賴多個其他微服務

何時不宜使用斷路器

- 鬆耦合、事件驅動的微服務架構
- 如果微服務不依賴於其他微服務

Service Mesh 模式

可部署性

流量管理

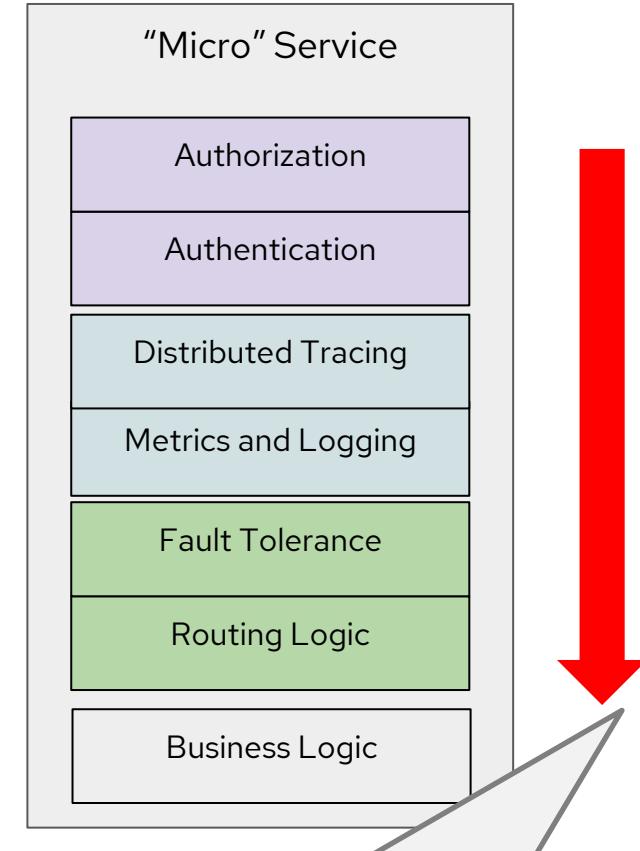
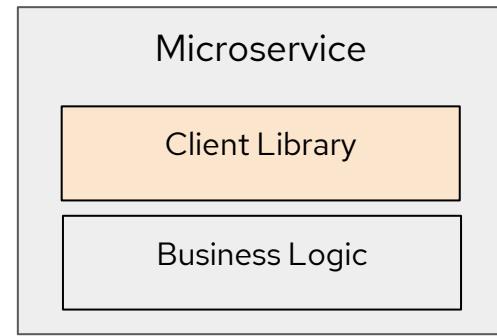
- 服務發現和負載均衡
- 服務分流：金絲雀部署、A/B 部署、百分比分流、Header 分流、流量鏡像
- 故障注入：延遲和中斷
- 服務 timeout 和 retry 機制
- 服務融斷

安全控制

- 身分驗證：服務間驗證、終端用戶驗證
- 授權：Mesh 網路間服務的授權

策略和遙測

- 監控
- 日誌
- 追蹤
- 速率限制



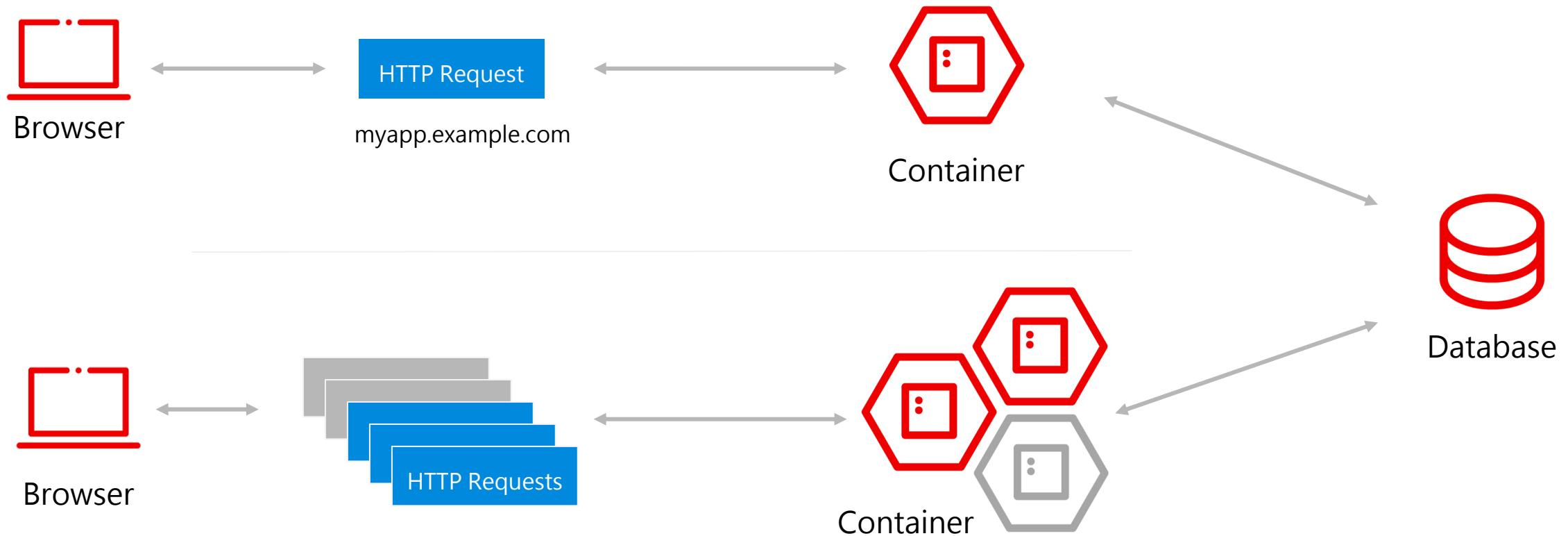
Move functions from Microservices chassis to Service Mesh and Container Platform

Serverless

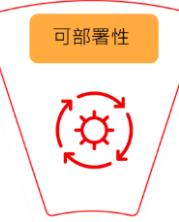
Serverless Pattern - Sync

可部署性

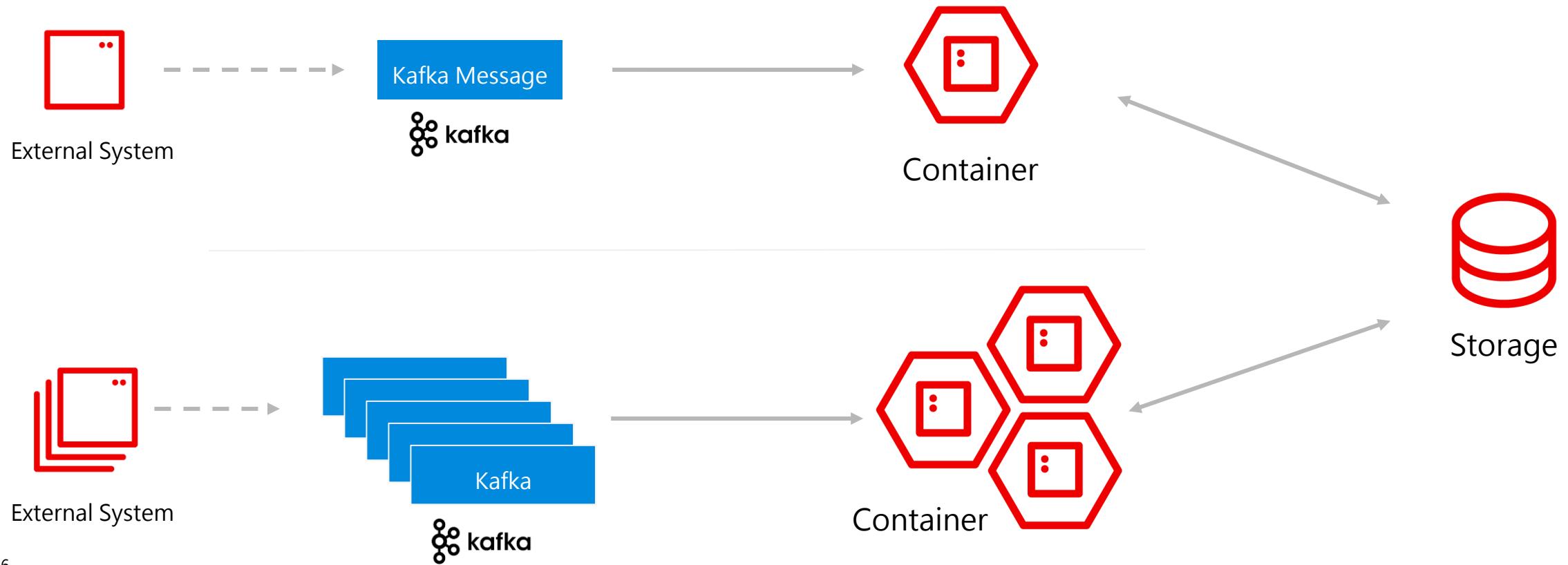
Serverless 的 Web 應用



Serverless Pattern - Event



Serverless 的事件驅動應用



Knative 物件說明

可部署性



SERVING

Service

Manage the lifecycle of your workload and group necessary objects for your application.

Revision

Immutable point-in-time snapshot of code & configuration.

Configuration

Maintains deployment state, environment variables and other runtime values.

Route

Maps a network endpoint, including an URL to a given revision or service.

EVENTING

Filter

Applied to a Broker in order to allow types of events to be selected.

Trigger

A desire to subscribe to events of a given Broker or event consumers using a Filter.

Subscription

Connect all events from a given channel to a service

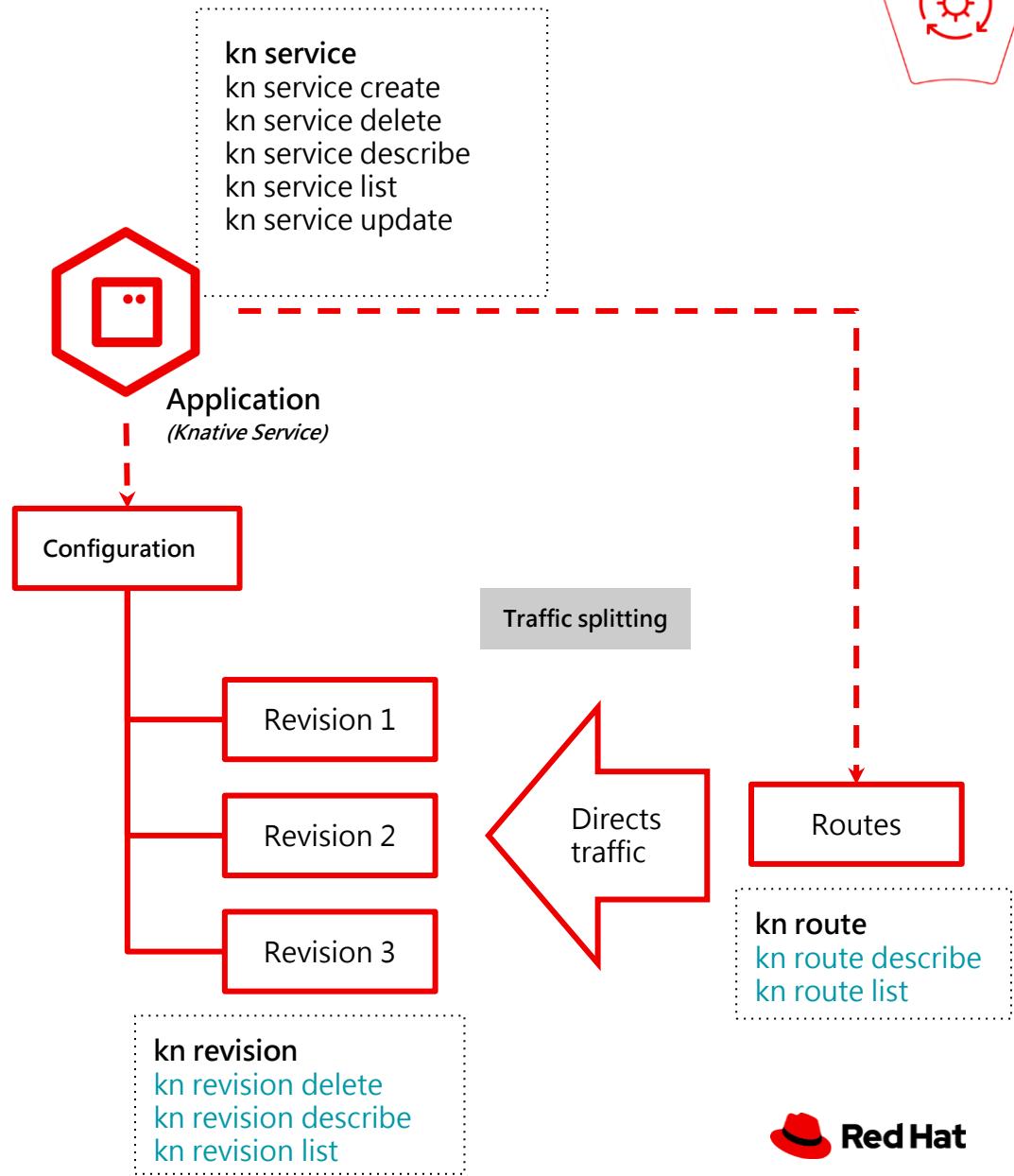
Choice

List of Filter options that will receive the same Event for processing.

Serving

可部署性

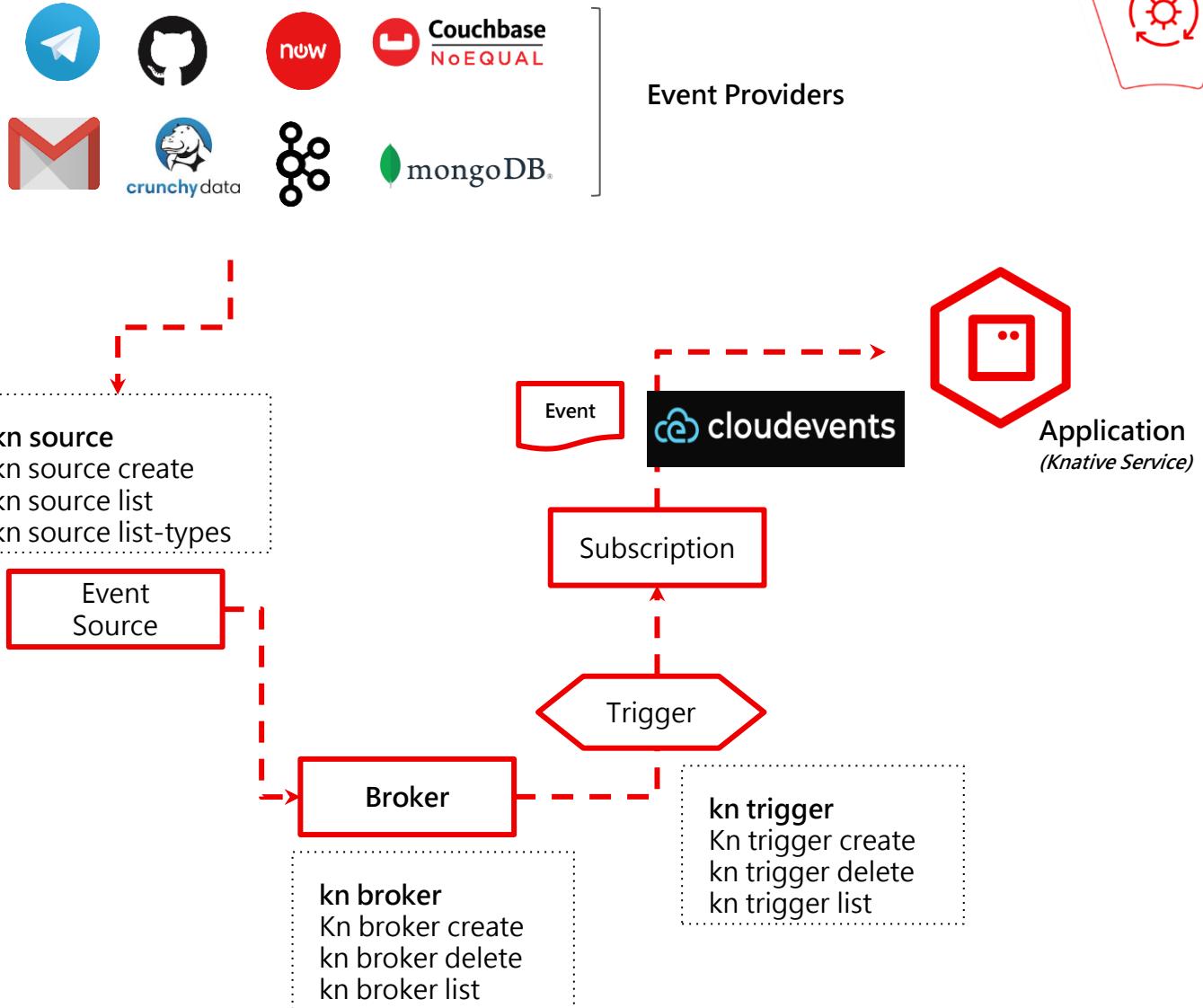
```
kind: Service
apiVersion: serving.knative.dev/v1
metadata:
  name: inventory-svc
spec:
  template:
    spec:
      containers:
        - image: docker.io/modernizingjavaappsbook/inventory-quarkus:latest
          ports:
            - containerPort: 8080
```



Eventing

```
kind: Broker
apiVersion: eventing.knative.dev/v1
metadata:
  name: default
  namespace: coolstore
```

```
kind: Trigger
apiVersion: eventing.knative.dev/v1
metadata:
  name: inventory-trigger
spec:
  broker: default
  filter:
    attributes:
      type: web-wakeup
  subscriber:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: inventory
```



容器 CI/CD 實踐

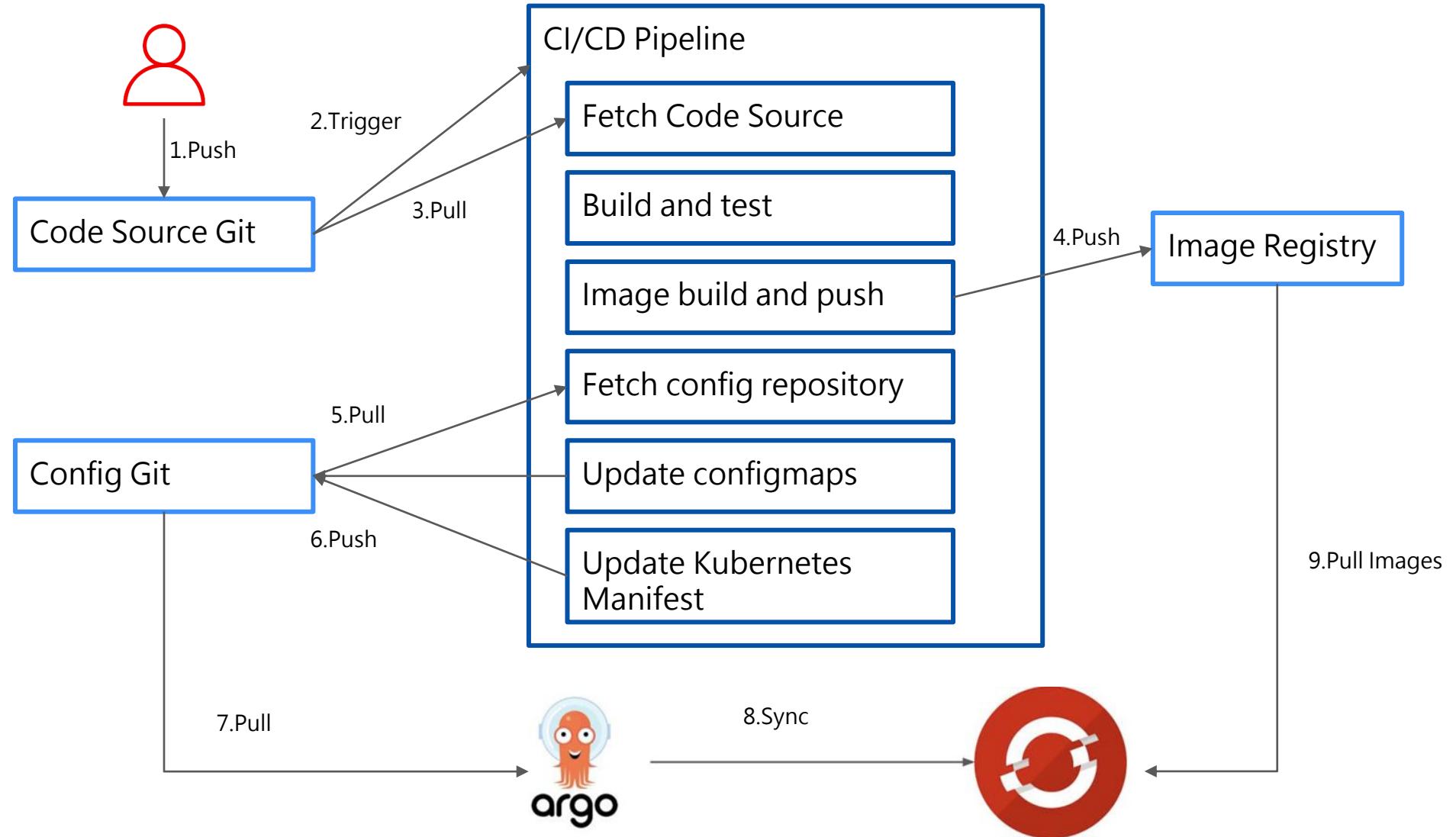
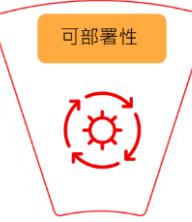
容器基本 CI/CD

可部署性

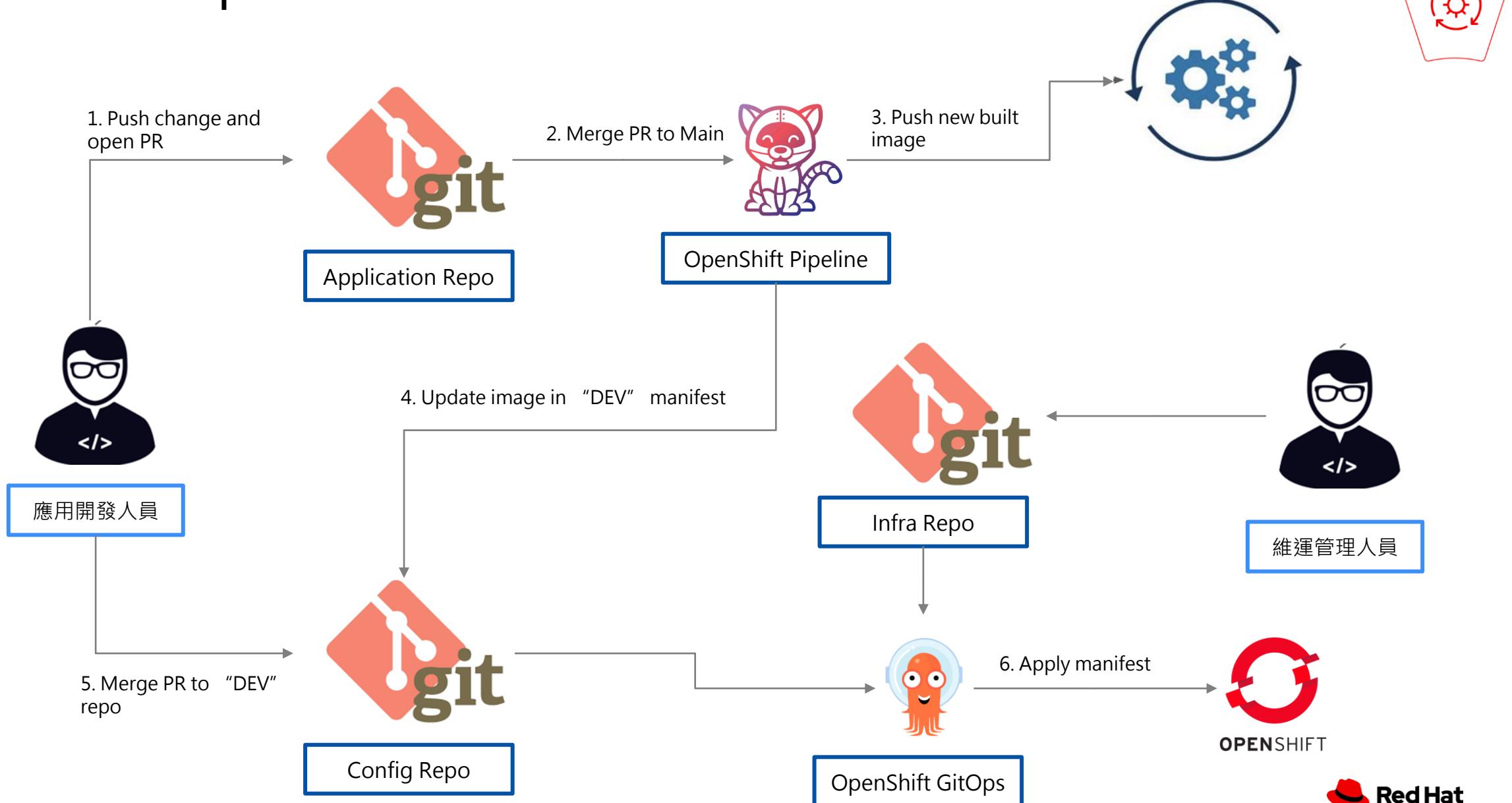
- 對於容器化系統，系統進行更改需要建立新的容器鏡像並進行宣告式物件部署。因此需要建立 CI/CD 自動執行這些任務。



基於 GitOps 的 CI 整合



基於 GitOps 的流程整合



GitOps 及 Kustomize

ArgoCD

可部署性



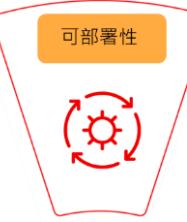
ArgoCD 是一種宣告式持續交付工具，透過 GitOps 原則來維護集群資源。ArgoCD 做為一個控制器實踐，持續監控 Git repo 中定義的應用程式定義和配置，並將這些配置的所需狀態與其在集群上的即時狀態進行比較。

如果同步失敗，ArgoCD 可以讓配置回復到上一版的配置

The screenshot shows the ArgoCD interface for the 'pipeline-test-app'. At the top, there are tabs for 'APP DETAILS', 'APP DIFF', 'SYNC', 'SYNC STATUS', 'HISTORY AND ROLLBACK' (which is highlighted with a red box), 'DELETE', and 'REFRESH'. Below this, the 'APP HEALTH' section shows 'Healthy'. The 'CURRENT SYNC STATUS' section shows a green checkmark and the word 'Synced'. It includes a 'MORE' link, a 'To HEAD (ec6a827)' link, and details about the author and comment. The 'LAST SYNC RESULT' section shows a green checkmark and the word 'Sync OK'. It includes a 'MORE' link, a 'To ec6a827' link, and a note that it succeeded in a few seconds. A large red arrow points from the text above to the 'HISTORY AND ROLLBACK' button. Another red arrow points from the text below to the 'Synced' status message.

集群物件透過宣告式的方式與 Git repo
中的配置檔原始碼同步

OpenShift GitOps 實體



OpenShift GitOps 定義了一組 Kubernetes Custom Resources 讓部署流程可以自己組裝這些定義檔

Project

應用的邏輯群組

Application

部署至 OpenShift 的應用定義

Application Set

部署至多個環境的應用建造框架

Resource Hook

管理部署生命週期的 Hooks

Sync Wave

部署的同步順序

ArgoCD application 概念

可部署性

Application

一組由 manifest 定義的 Kubernetes。是一個 Custom Resource Definition (CRD)

Target state

應用的目標狀態，存放在 Git 中

Sync

將應用執行到其目標狀態的流程。如，將 Yaml 變更應用到 Kubernetes cluster.

Application source type

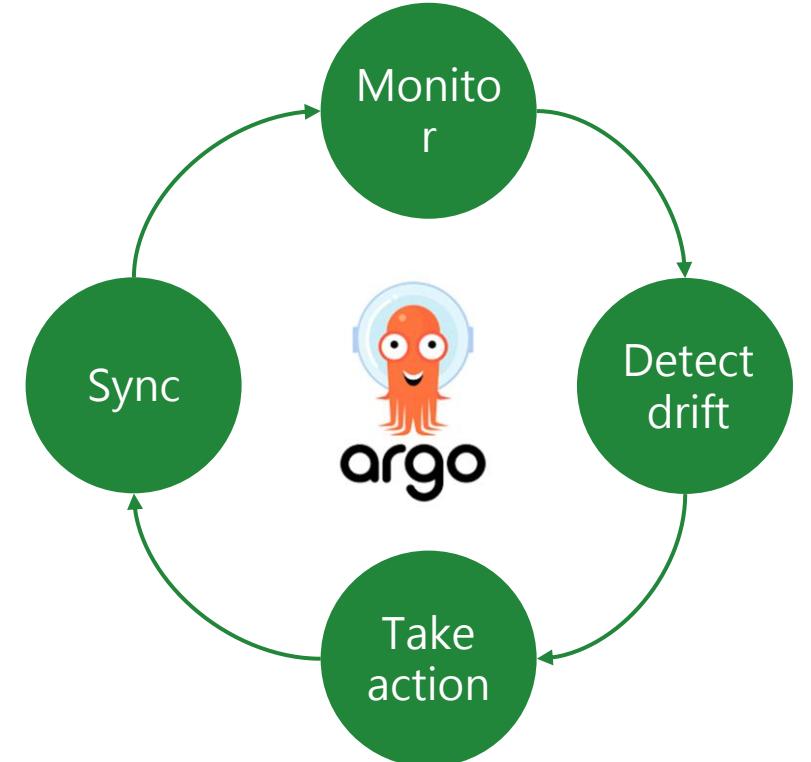
用來構建應用的工具 (Helm, Kustomize, etc)

Live state

應用的真實狀態，pods 等應用服務部署的情形

Health

應用的健康狀態。如，是否正確運行、是否可以對外調用



Application 物件

可部署性



Application CRD 是代表部署
應用的 Kubernetes 物件

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: jumpapp-development
spec:
  project: development
  source:
    repoURL: https://github.com/acidonper/jump-app-gitops
    targetRevision: feature/jump-app-dev
    path: .
  destination:
    server: https://kubernetes.default.svc
    namespace: jumpapp
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
      allowEmpty: false
```

ArgoProject 連結

參考之 Git repo

應用部署的集群和 namespace

同步流程配置

Application 限制



- 每個 Application 定義僅有一個集群
- 只能定義一個種類的資源
 - Git or Helm
- 只能定義一個來源，Git repo 或 Helm chart
- 非 Applications 上的其他的定義需要多個 Applications 實踐



ApplicationSet 物件

可部署性

ApplicationSet 物件使用模板自動化來同時建立、修改和管理多個 ArgoCD 應用程式。

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: guestbook
spec:
  generators:
    - list:
      elements:
        - cluster: development
          url: https://ocp-development.acidonpe.com
          values:
            project: development
            branch: feature/jump-app-dev
        - cluster: production
          url: https://ocp-production.acidonpe.com
          values:
            project: production
            branch: feature/jump-app-pro
  template:
    metadata:
      name: 'jumpapp-{{cluster}}'
    spec:
      project: '{{values.project}}'
      source:
        repoURL: https://github.com/acidonper/jump-app-gitops
        targetRevision: '{{values.branch}}'
        path: guestbook/{{cluster}}
      destination:
        server: '{{url}}'
        namespace: jumpapp
```

生成 Applications
可以替換的元素

實際產生
Application 可填入
變數部分

ApplicationSet 物件

可部署性

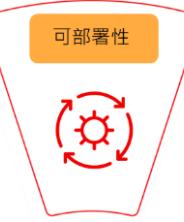


ApplicationSet 物件改進 ArgoCD 內的多集群支援和集群多租用戶支援。ArgoCD 應用程式可以從多個不同的來源進行模板化，包括來自 Git 或 ArgoCD 自己定義的集群清單。

- ApplicationSet controller 和 ArgoCD 安裝在同一個 namespace 中
- ApplicationSet controller 基於 ApplicationSet Custom Resource (CR) 自動化產生 ArgoCD Applications
- ApplicationSet controller 允許開發人員在不需要集群管理者權限的前提下自主的建立 Applications

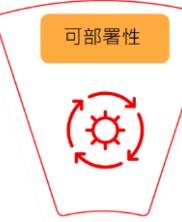
同步機制

可部署性

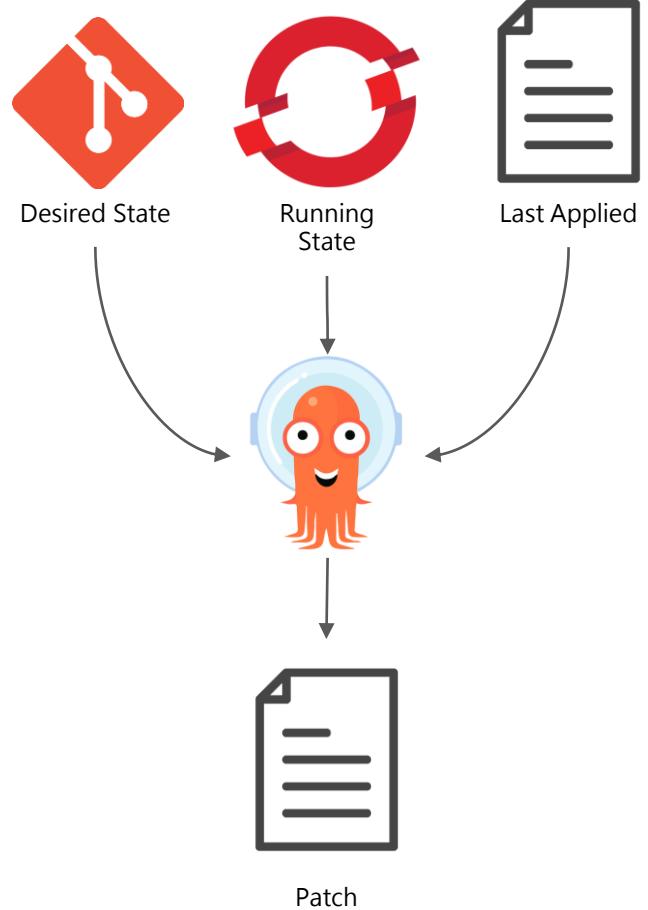


- ArgoCD 有一個「本地快取」Git Repo
- 在同步時「git ls-remote」會被執行
 - Fetches 最近一次 commit
 - 如果 manifests 沒有被快取儲存，就會執「git pull」
 - 僅有目標 branch 被使用比對
- 目標配置和真實配置進行比較
 - 實踐 “three way diff”
 - 目標配置、最後配置、當前配置
- 無須備份快取
 - 當本地快取不存在時，便會執行「git clone」

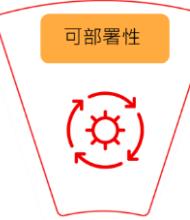
Three Way Diff



- 從 Repo 中讀取所有的目標狀態 manifests，並將其放置於快取
- 讀取集群內相關的當前配置，並將其放置於快取
- 計算物件的 Diff，並將當前配置的 *last-applied-configuration* 納入考量
- 將每個一需要做 diff 的物件進行 patch 動作
 - 每個物件當單獨的被調整



使用 ArgoCD 同步策略



Automated

預設自動同步，錯誤後重視5次，間隔實間為(5s, 10s, 20s, 40s, 80s)

其他配置策略:

- Prune
- Self Heal
- Allow Empty

Sync Options

透過以下同步選項來調整同步行為:

- Validate
- Create Namespace
- Prune Prolongation Policy
- Prune Last

Retries

重試策略的配置選項:

- Limit
- Backoff
 - Duration
 - Factor
 - Max Duration

同步參數 (1)

可部署性



- 防止物件被清除
- 略過物件 Dry Run (通常在 CRD 同步 Apply 時使用)
- 防止物件在應用被刪除時被同步刪除(如 PVC)
- 不進行 kubectl apply 的檢查動作 (針對一些 extension API)

```
metadata:  
annotation:  
    argocd.argoproj.io/sync-options: Prune=false
```

```
metadata:  
annotation:  
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
```

```
metadata:  
annotation:  
    argocd.argoproj.io/sync-options: Delete=false
```

```
metadata:  
annotation:  
    argocd.argoproj.io/sync-options: Validate=false
```

同步參數 (2)

- 清除物件在最後階段執行
- 自動建立 Namespace
- 僅同步 Out of Sync 物件

```
spec:
  syncPolicy:
    syncOptions:
      - PruneLast=true
```

```
spec:
  syncPolicy:
    syncOptions:
      - CreateNamespace=true
```

```
spec:
  syncPolicy:
    syncOptions:
      - ApplyOutOfSyncOnly=true
```

同步步驟

可部署性



當 ArgoCD 啟動同步，它以下列的順序排序資源：

- The phase
- The wave (數字越低越優先)
- By kind (e.g. namespaces 優先)
- By name

其重複這個流程直到所有的 phases 和 waves 呈現同步狀態(in-sync) 及健康狀態(healthy)

ArgoCD 以數個步驟(steps)執行一個同步操作，總共有三個 phases，分別是最上層的 pre-sync, sync 和 post-sync；而在每個 phase 可以有一個或多個 waves，透過這個步驟讓使用者確保某些特定 resources 在下一個 resources 建立前是同步狀態

- 使用 “Syncwaves” 來定義部署順序
- Manifests 基於數字來建立
 - 數字可以為負
 - 越低越優先
- 預設是 0

```
metadata:  
annotations:  
argocd.argoproj.io/sync-wave: "5"
```

同步 Hooks

可部署性



Hooks 是在同步前、中、後三個階段運行腳本的手段。Hooks 也可以在同步執行的任何時間點發生錯誤時值行。Hook 的使用情境如下：

- **PreSync** hook 可以在部署一個新版本的應用前執行資料庫 schema 轉移動作
- **Sync** hook 可以調度比起單純的 rolling update 更為複雜的部署策略.
- **PostSync** hook 可以在部署完成後值行一些整合和健康檢查作業
- **SyncFail** hook 可以在同步執行錯誤時運行清理邏輯

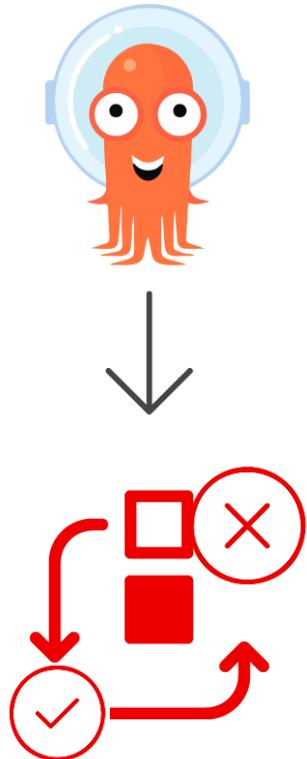
Hooks 可以任何形式的 Kubernetes resource 執行，通常是 Jobs or Pods

```
apiVersion: batch/v1
kind: Job
metadata:
  generateName: schema-migrate-
  annotations:
    argocd.argoproj.io/hook: PreSync
```

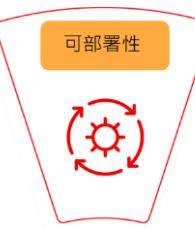
Hook 刪除機制

可部署性

- Hooks 可以被刪除
 - 缺乏刪除策略代表 hook 一直存在
- 透過 annotation 配置
 - e.g. ~> *argocd.argoproj.io/hook-delete-policy: HookSucceeded*
- 刪除策略
 - HookSucceeded
 - HookFailed
 - BeforeHookCreation

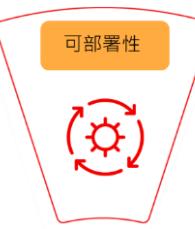


Resource Hooks 範例



```
apiVersion: batch/v1
kind: Job
metadata:
  generateName: app-slack-notification-
  Annotations:
    argocd.argoproj.io/hook: PostSync | --> Hook 類型
    argocd.argoproj.io/hook-delete-policy: HookSucceeded | --> Hook 自動刪除策略
spec:
  template:
    spec:
      containers:
        - name: slack-notification
          image: curlimages/curl
          command:
            - "curl"
            - "-X"
            - "POST"
            - "--data-urlencode"
            - "payload={"channel": "#somechannel",
              "username": "hello",
              "text": "App Sync succeeded",
              "icon_emoji": ":ghost:"}
            - "https://hooks.slack.com/services/..."
          restartPolicy: Never
          backoffLimit: 2
  
```

SyncWaves 範例



```
apiVersion: batch/v1
kind: Job
metadata:
  name: init-job
  namespace: test
  annotations:
    argocd.argoproj.io/sync-wave: "-1"
spec:
  template:
    spec:
      containers:
        - name: postgresql-client
          image: postgres:12
          env:
            - name: PGPASSWORD
              value: admin
          command: ["psql"]
          args:
            [...]
  restartPolicy: Never
```

OpenShift Job 可為 Application 一部分在 Git Repo 中提供

Sync-Wave Annotation : 設置 -1 可以在任何沒有提供 annotation 的物件或 sync-wave 為正的物件前執行此任務 (因預設是 0)

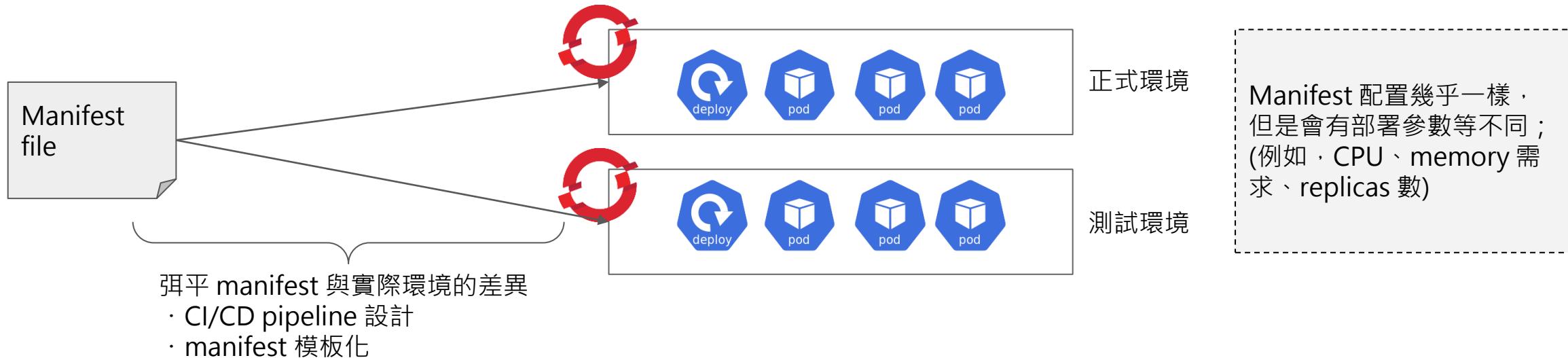
GitOps 需要模板化配置

可部署性



GitOps 使用 Git 管理的配置進行部署。但是實務上有多个部署標的時（例如，測試環境和生產環境），則會產生下述問題：

- 有些參數資源在每個環境中通常是不同的
- 除了更改應用程式之外，還需要將所使用的容器鏡像的標籤重寫為最新版本等操作



ArgoCD Kubernetes Objects Generator



Helm

Helm uses a packaging format called charts. A chart is a collection of files that describe a related set of Kubernetes resources



Kustomize

Template-free way to customize application configuration that simplifies the use of off-the-shelf applications



Kubernetes Manifests

Plain text kubernetes object located in YAML or JSON format

什麼是 Kustomize

可部署性



Kustomize 讓 CD 部署的 yaml manifest 可以進行無模板的客製化，進行多種配置修改用途，並且讓初始 yaml manifest 保留原樣不受影響

- Kustomize 是上 patch 的框架
- 讓環境特定的配置改動在不重複製作 yaml 的前提下提供
- 不同於模板框架，所有的 yaml 都是可以直接使用於 kubernetes 環境
- Kustomize 可以直接用 oc 指令操作

```
oc apply -k apps/myapp/overlays/dev
```

Kustomize 目錄架構

可部署性



Kustomize 透過階層式的目錄架構進行管理，包含 **bases** 和 **overlays**

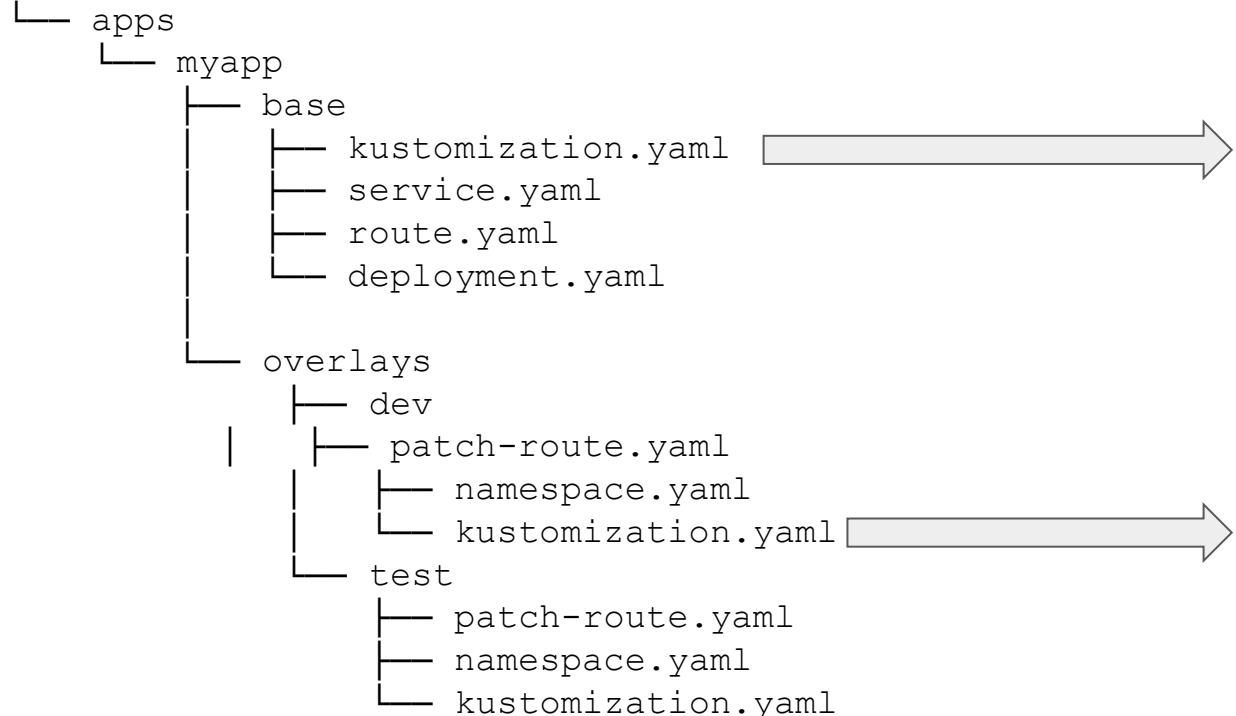
- **base**: 存放的 kustomization.yaml 文件的目錄包含一組資源和相關的客製內容
 - base 不包含任何 overlay 相關的資訊
 - base 可以被任一 overlays 參考
- **overlays**: 存放的 kustomization.yaml 參考其他的 kustomization 目錄
 - overlay 可能整合多個 bases/overlays 且可能有多個客製內容

```
# tree myapp
myapp
└── base
    └── overlays
        ├── development
        ├── production
        └── staging

5 directories, 0 files
```

Kustomize 目錄架構

可部署性



```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- service.yaml
- route.yaml
- deployment.yaml
```

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

Namespace: dev
bases:
- ../../base
resources:
- namespace.yaml
patchesStrategicMerge:
- patch-route.yaml
```

Kustomize 目錄架構

Kustomize 的每個配置集透過 kustomization.yaml 檔案作為起點



```
# tree myapp
myapp
└── base
    ├── deployment.yaml
    ├── namespace.yaml
    ├── route.yaml
    └── service.yaml
overlays
└── development
└── production
└── staging

5 directories, 4 files
# cd myapp/base
# oc kustomize create --autodetect
# ls -l
合計 20
-rw-r--r--. 1 root root 503 7月 23 09:09 deployment.yaml
-rw-r--r--. 1 root root 138 7月 23 09:16 kustomization.yaml
-rw-r--r--. 1 root root 125 7月 23 09:10 namespace.yaml
-rw-r--r--. 1 root root 214 7月 23 09:10 route.yaml
-rw-r--r--. 1 root root 235 7月 23 09:09 service.yaml
```

```
# cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
- namespace.yaml
- route.yaml
- service.yaml
```

Kustomization File

可部署性



- Kustomization.yaml 檔案中清單中的順序相關且重要
- 除了這四個清單之外，還可以添加其他欄位
(commonLabels、namePrefixes...等)
- Build 階段的處理程序：
 - 處理 resources
 - 產生 generator 並將其新增至 resources 清單
 - 透過 transformers 進一步修改 resources 清單
 - 透過 validators 檢查 resources 清單是否有錯
- resources：
 - 設定要處理的YAML清單檔案的路徑以及 kustomization.yaml 檔案所在目錄的路徑

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- {pathOrUrl}
- ...

generators:
- {pathOrUrl}
- ...

transformers:
- {pathOrUrl}
- ...

validators:
- {pathOrUrl}
- ...
```

多數的 kustomize 不
需要全部四種清單

Kustomize Overlays 環境建構

可部署性



Kustomize 將特定於環境的 patch 套用至一組 base 配置。每個覆蓋環境都需要一個 kustomization.yaml 檔案

```
# tree myapp
myapp
├── base
│   ├── deployment.yaml
│   ├── namespace.yaml
│   ├── route.yaml
│   └── service.yaml
└── overlays
    ├── development
    │   └── kustomization.yaml
    ├── production
    │   └── kustomization.yaml
    └── staging
        └── kustomization.yaml
5 directories, 8 files
```

```
# cat myapp/overlays/development/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- ../../base

namespace: development

images:
- digest: sha256:50672709e10ecde5fadb1c6b29b5140b350031fc2c38b0314fb848ac4b0e3814
  name: __IMAGE_URL__@__IMAGE_DIGEST__
  newName: image-registry.openshift-image-registry.svc:5000/development/health-record
```

Kustomize Build

可部署性



當 Kustomize build 時將每個環境建議一個 manifest，並套用 patch

```
# cat myapp/base/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: health-record
    name: health-record
spec:
  replicas: 1
  annotations:
    sidecar.istio.io/inject: "true"
  spec:
    containers:
      - image: __IMAGE_URL__@__IMAGE_DIGEST__
        name: health-record
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: health-record
    name: health-record
  namespace: development
spec:
  replicas: 1
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: health-record
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true"
      labels:
        app: health-record
    spec:
      containers:
        - image: image-registry.openshift-image-
          registry.svc:5000/development/health-
          record@sha256:50672709e10ecde5fadb1c6b29b5140b350031fc2c38b0314
          fb848ac4b0e3814
          name: health-record
          ports:
            - containerPort: 8080
              protocol: TCP
```

Kustomize 內建功能

可部署性



有兩種方法可以使用 Kustomize 的內建功能：

觸發 kustomization 中的欄位參數

```
# kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

commonAnnotations:
  app: myApp

resources:
  - deployment.yaml
```

指定 kustomization 中的 generators 或 transformers 檔案作為觸發

```
# kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

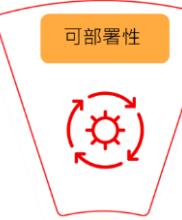
transformers:
  - annotationtransformer.yaml

resources:
  - deploy.yaml
```

```
# annotationtransformer.yaml
apiVersion: builtin
kind: AnnotationsTransformer
metadata:
  name: not-important-to-example
annotations:
  app: myApp
  - path: metadata/annotations
    create: true
```

第一種方法比較容易使用，但可配置的選項有限。第二種方法允許完整的規範。

Kustomize 內建功能



- transformers：
 - 可用於 patch 特定欄位，如 image、namespace 和 commonLabels
 - 可用於定義任意 patch 的插件
- Generator：
 - 有一些內建插件（例如 configMapGenerator 和 SecretGenerator），可以透過各種方式為給定配置產生資源。

Kustomize Transformer (1)

namespace / namePrefix / nameSuffix

```
# kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

namespace: kustomize-namespace
```

resources:

- deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: the-deployment
  • namespace: kustomize-namespace
spec:
  replicas: 5
  template:
    containers:
      - image: registry/conatiner:latest
        name: the-container
```

namespace

<https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/namespace/>

namePrefix

<https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/nameprefix/>

nameSuffix

<https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/namesuffix/>

Kustomize Transformer (2)

images

```
# kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

images:
- name: postgres
  newName: my-registry/my-postgres
  newTag: v1
- name: nginx
  newTag: 1.8.0
- name: my-demo-app
  newName: my-app
- name: alpine
  digest:
    sha256:24a0c4b4a4c0eb97a1aabb8e29f18e917d05abfe1b7a7c07857230879
    ce7d3d3
resources:
- deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: the-deployment
spec:
  template:
    spec:
      containers:
        - image: my-registry/my-postgres:v1
          name: mypostgresdb
        - image: nginx:1.8.0
          name: nginxapp
        - image: my-app:latest
          name: myapp
        - image:
            alpine@sha256:24a0c4b4a4c0eb97a1aabb8e29f18e917d05abfe1b7a7c07857230879ce7d
            3d3
          name: alpine-app
```

Kustomize Transformer (3)



commonLabels / commonAnnotations

```
# kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

commonLabels:
  someName: someValue
  owner: alice
  app: bingo

resources:
- deployment.yaml
- service.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: bingo
    owner: alice
    someName: someValue
  name: example
spec:
  selector:
    app: bingo
    owner: alice
    someName: someValue
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: bingo
    owner: alice
    someName: someValue
  name: example
spec:
  selector:
    matchLabels:
      app: bingo
      owner: alice
      someName: someValue
  template:
    metadata:
      labels:
        app: bingo
        owner: alice
        someName: someValue
```

commonLabels

<https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/commonlabels/>

commonAnnotations

<https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/commonannotations/>

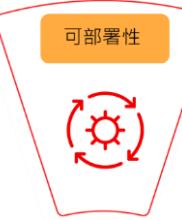
Kustomize Transformer (4)

patches / patchesJson6902 「JSON Patch」

```
# kustomization.yaml
resources:
- deployment.yaml
patches:
- patch: |-
  - op: add
    path: /metadata/labels/app.kubernetes.io~1version
    value: 1.21.0
  target:
    group: apps
    version: v1
    kind: Deployment
- patch: |-
  - op: replace
    path: /spec/template/spec/containers/0/image
    value: nginx:1.21.0
  target:
    labelSelector: "app.kubernetes.io/name=nginx"
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/name: nginx
    ● app.kubernetes.io/version: 1.21.0
    name: dummy-app
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: nginx
  template:
    metadata:
      labels:
        app.kubernetes.io/name: nginx
    spec:
      containers:
        ● - image: nginx:1.21.0
          name: nginx
          ports:
            - containerPort: 80
              name: http
```

Kustomize Transformer (5)



patches / patchesStrategicMerge 「 strategic-merge-style patch (SMP) 」

```
# kustomization.yaml
resources:
- deployment.yaml
patches:
- patch: |-  
    apiVersion: apps/v1  
    kind: Deployment  
    metadata:  
      name: dummy-app  
      labels:  
        app.kubernetes.io/version: 1.21.0
- patch: |-  
    apiVersion: apps/v1  
    kind: Deployment  
    metadata:  
      name: not-used
    spec:  
      template:  
        spec:  
          containers:  
            - name: nginx  
              image: nginx:1.21.0
target:  
  labelSelector: "app.kubernetes.io/name=nginx"
```

The diagram illustrates the merging process of Kustomize patches into a target Kubernetes manifest. A large grey box on the left contains the 'kustomization.yaml' file with two patches. Red arrows point from the 'patch' blocks in the file to specific sections of a target Deployment manifest on the right. One arrow points to the 'labels' section of the first patch, and another points to the 'containers' section of the second patch. The target manifest itself is a standard Deployment configuration with annotations and a template.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/name: nginx
    app.kubernetes.io/version: 1.21.0
  name: dummy-app
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: nginx
  template:
    metadata:
      labels:
        app.kubernetes.io/name: nginx
    spec:
      containers:
        - image: nginx:1.21.0
          name: nginx
          ports:
            - containerPort: 80
              name: http
```

Kustomize Generators

configMapGenerator / secretGenerator



```
# kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
configMapGenerator:
- name: my-java-server-env-vars
  literals:
    - JAVA_HOME=/opt/java/jdk
    - JAVA_TOOL_OPTIONS=-agentlib:hprof
```



```
apiVersion: v1
data:
  JAVA_HOME: /opt/java/jdk
  JAVA_TOOL_OPTIONS: -agentlib:hprof
kind: ConfigMap
metadata:
  name: my-java-server-env-vars-44k658k8gk
```

configMapGenerator

<https://kubectldocs.kubernetes.io/references/kustomize/kustomization/configmapgenerator/>

158

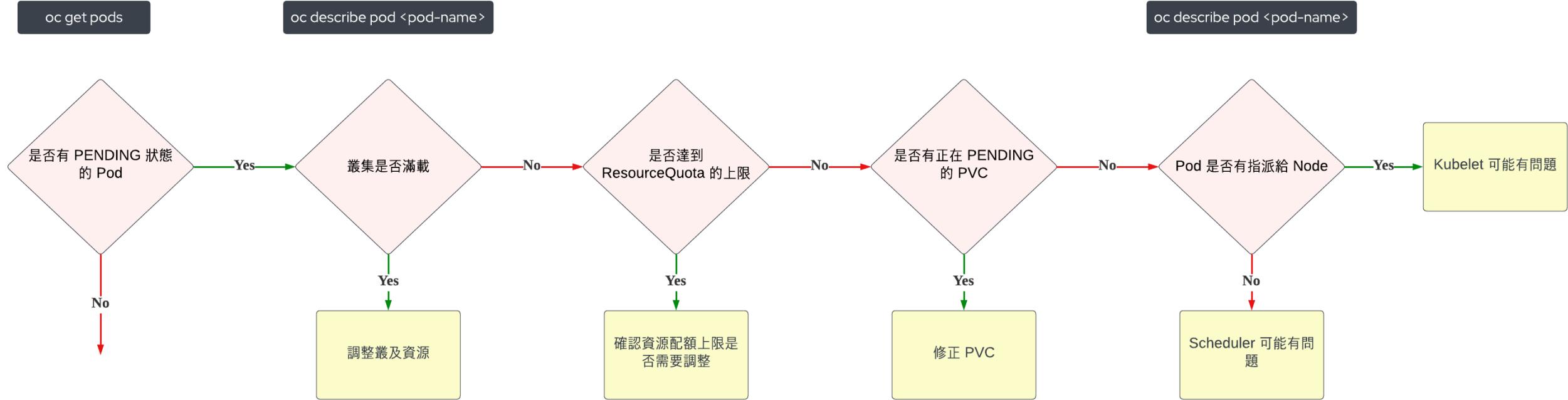
secretGenerator

<https://kubectldocs.kubernetes.io/references/kustomize/kustomization/secretgenerator/>

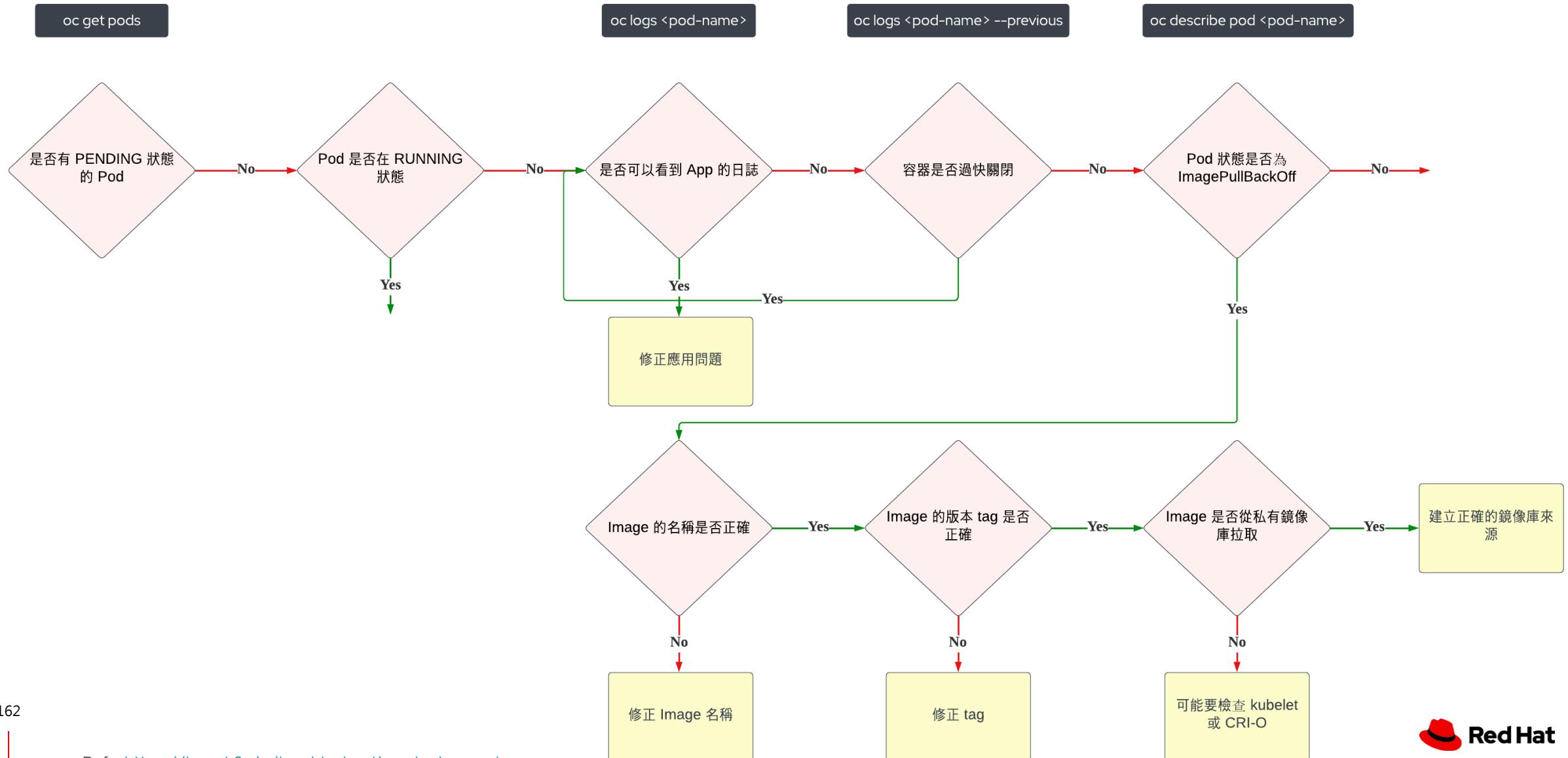
Lab 02 - GitOps & Kustomize

應用除錯流程及操作介紹

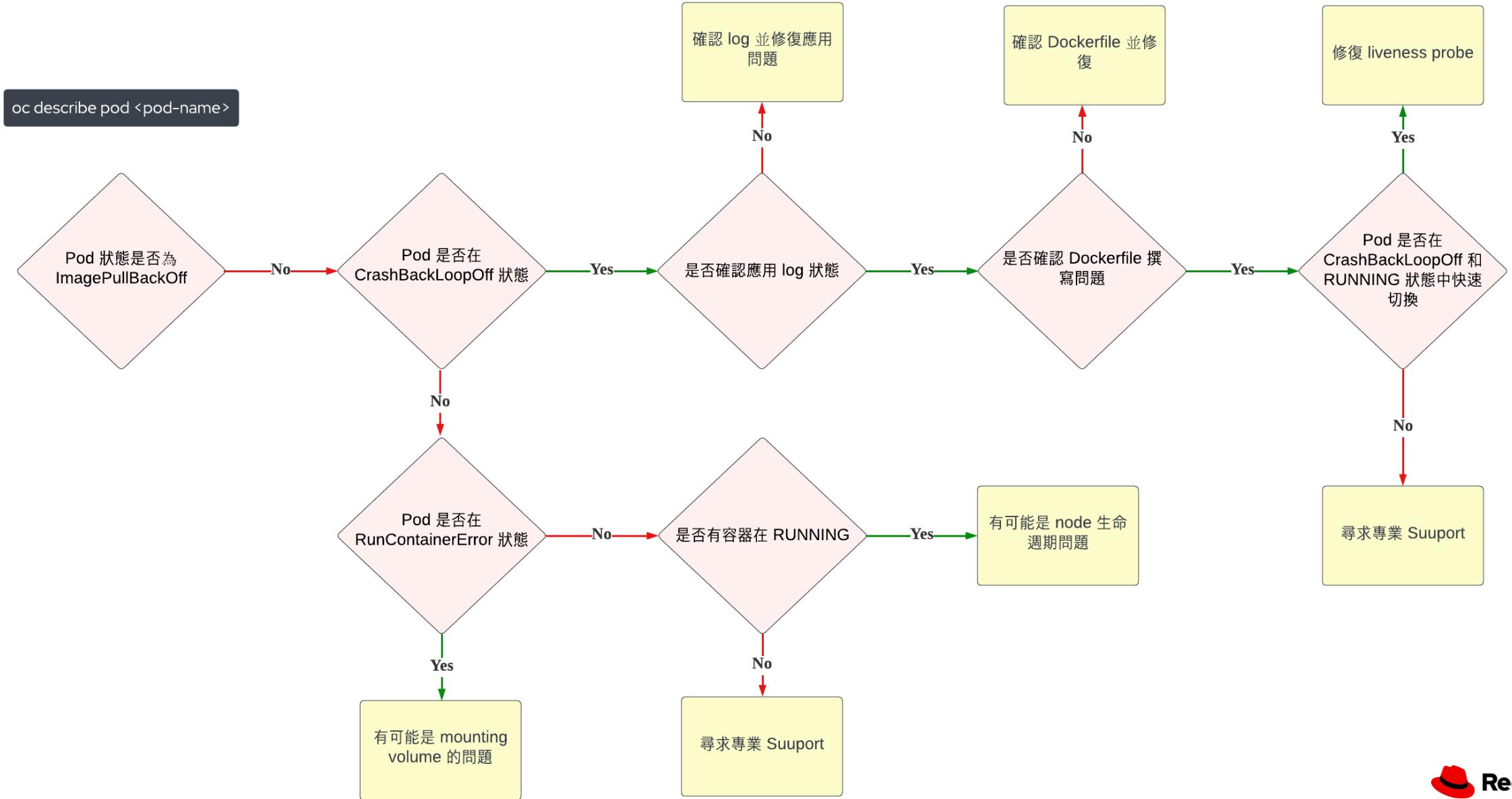
OpenShift 應用部署排查問題路線圖



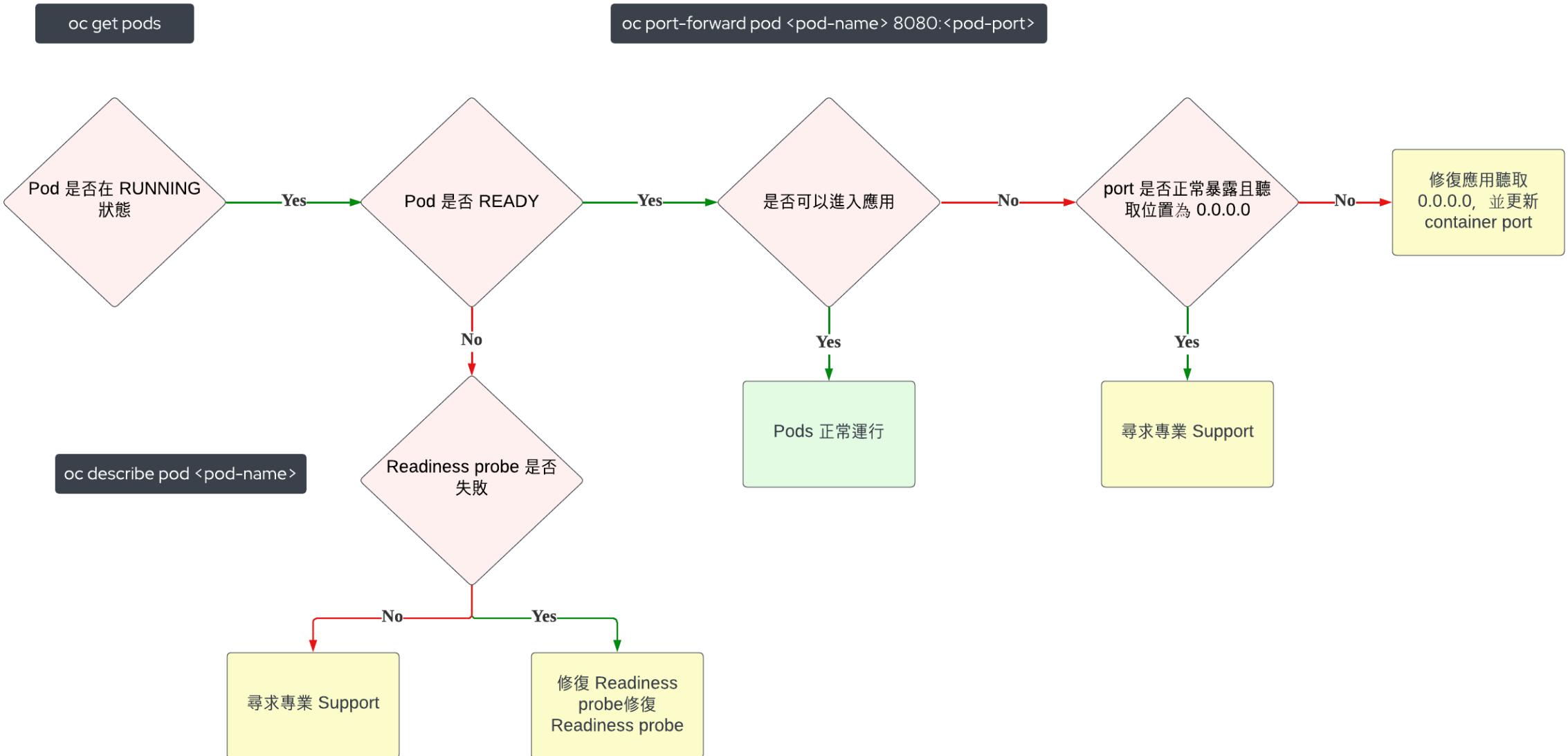
OpenShift 應用部署排查問題路線圖



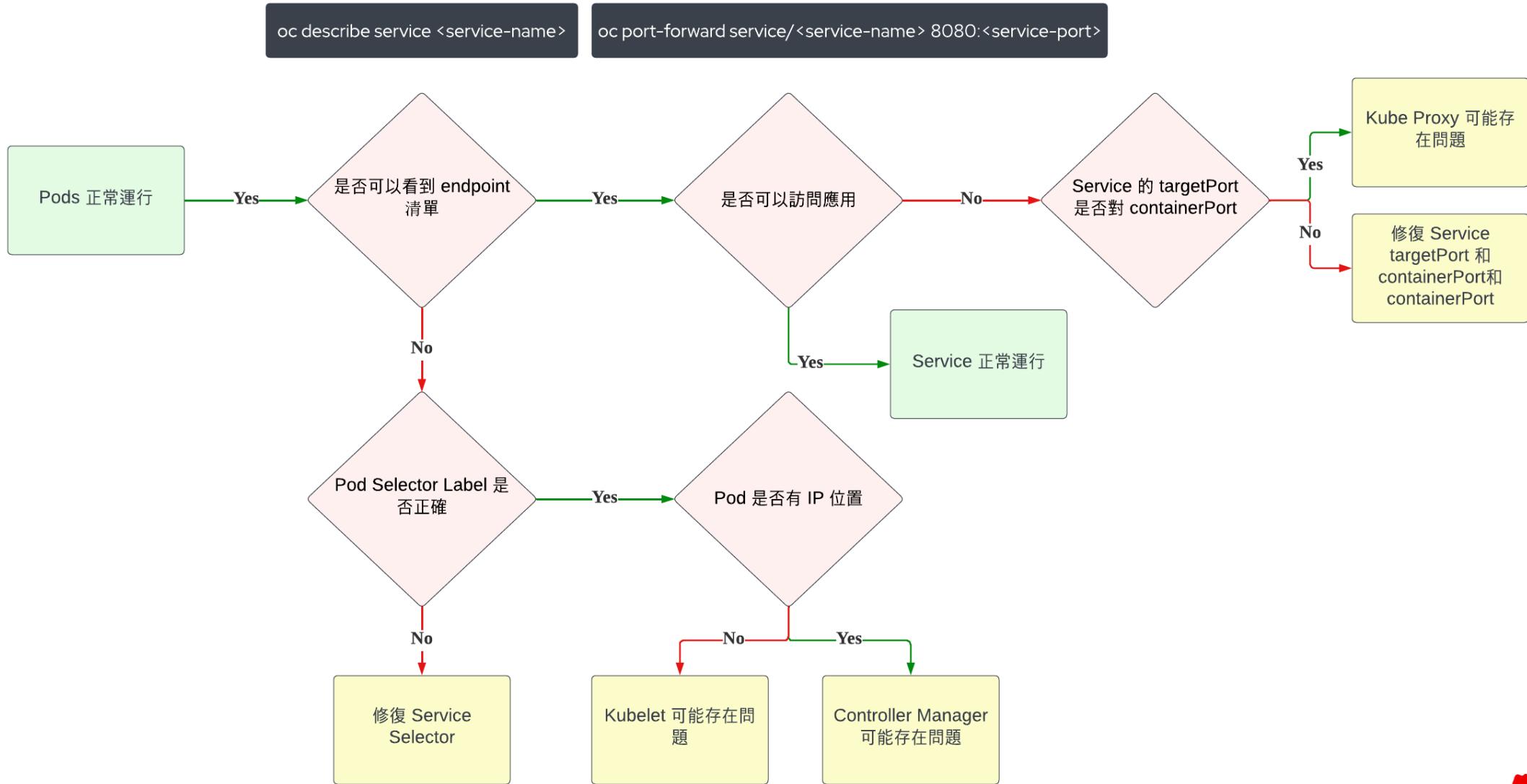
OpenShift 應用部署排查問題路線圖



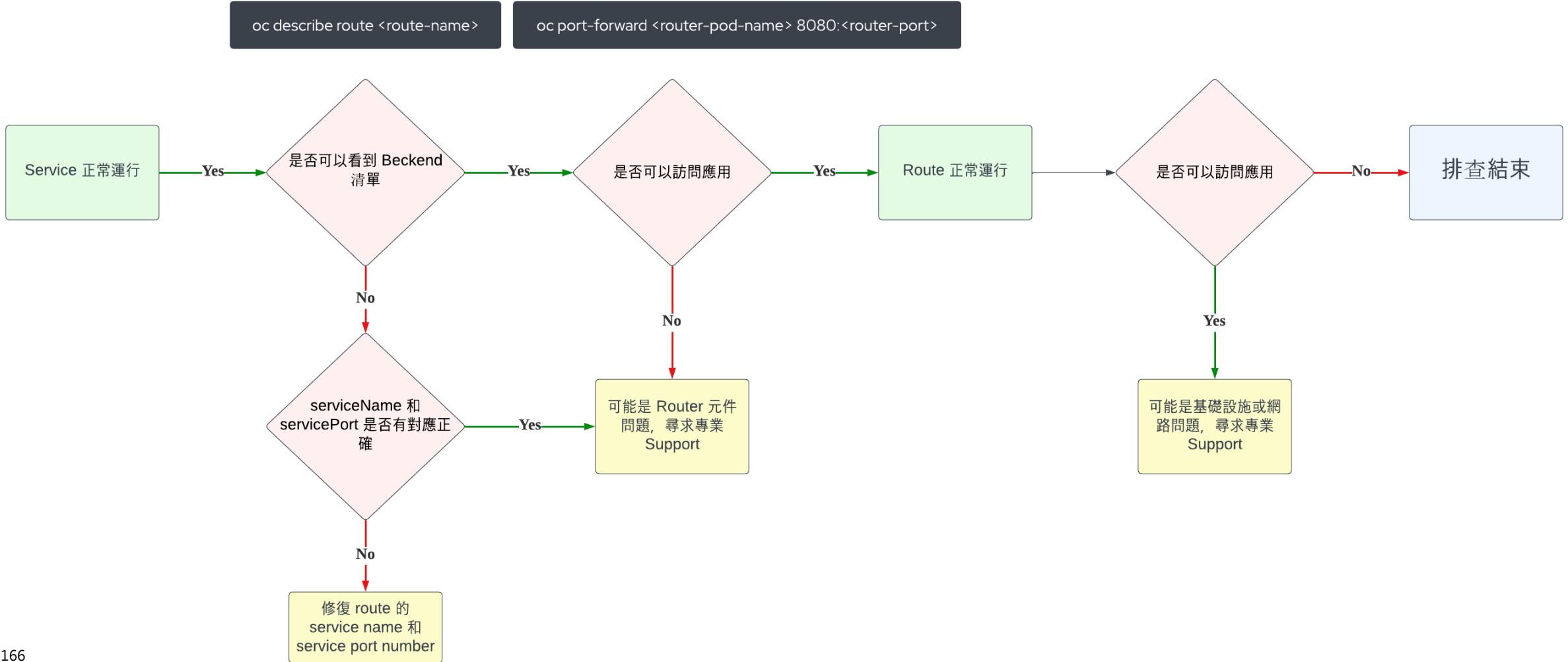
OpenShift 應用部署排查問題路線圖



OpenShift 應用部署排查問題路線圖



OpenShift 應用部署排查問題路線圖



除錯實操介紹 – 容器

多數的 OpenShift 基礎設施元件是以容器的形式實踐

```
[maemmanu-redhat.com@clientvm 1 ~]$ oc logs prometheus-k8s-0 -c prometheus -n openshift-monitoring
```

Opens a shell inside a pod to run shell commands interactively and non-interactively.

```
[maemmanu-redhat.com@clientvm 1 ~]$ oc project openshift-sdn
Now using project "openshift-sdn" on server "https://api.cluster-586a.sandbox452.opentlc.com:6443".
[maemmanu-redhat.com@clientvm 0 ~]$ oc rsh pod/sdn-4xjbx
sh-4.2#
```

Copies local files to a location inside a pod or copy pod files to local filesystem

```
$ oc cp /local/path my-pod-name:/container/path
```

Creates a TCP tunnel from local-port on your workstation to local-port on the pod

```
$ oc port-forward my-pod-name local-port:remote-port
```

除錯實操介紹 – 容器

批量蒐集叢集現狀資料以進行 debug。等同於 openshift-install gather 指令。

```
$ oc adm must-gather
```

```
[maemmanu-redhat.com@clientvm 0 ~]$ oc adm must-gather
[must-gather      ] OUT Using must-gather plugin-in image: quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:6326b78d6c05d9925576ae8c6768f7ee846b8e99618da
670b5e7b698e1dd433a
[must-gather      ] OUT namespace/openshift-must-gather-6ldhl created
[must-gather      ] OUT clusterrolebinding.rbac.authorization.k8s.io/must-gather-9xb7w created
[must-gather      ] OUT pod for plug-in image quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:6326b78d6c05d9925576ae8c6768f7ee846b8e99618da670b5e7b698e1
dd433a created
[must-gather-54z7w] POD Wrote inspect data to must-gather.
[must-gather-54z7w] POD Gathering data for ns/openshift-cluster-version...
[must-gather-54z7w] POD Wrote inspect data to must-gather.
[must-gather-54z7w] POD Gathering data for ns/openshift-config...
[must-gather-54z7w] POD Gathering data for ns/openshift-config-managed...
[must-gather-54z7w] POD Gathering data for ns/openshift-authentication...
```

展現運行資源的數據。

```
$ oc adm top pods
```

```
$ oc adm top images
```

除錯實操介紹 – 容器

Red Hat OpenShift Container Platform admin

Administrator

Home >

Operators >

Workloads >

Pods

Deployments

Deployment Configs

Stateful Sets

Secrets

Config Maps

Cron Jobs

Jobs

Daemon Sets

Replica Sets

Replication Controllers

Project: openshift-machine-api

Details NAME ENVIRONMENT LOGS EVENTS TERMINAL

Pod Details

Memory Usage

CPU Usage

Filesystem

Network In

Network Out

169

Red Hat

除錯實操介紹 – 配置

叢集配置是由 config.openshift.io 的 API 群組提供。這些資源由於是屬於叢集等級的配置，因此沒有分 namespace。

可以看到所有全域配置

```
[maemmanu-redhat.com@clientvm 0 ~]$ oc api-resources | grep config.openshift.io
apiservers
authentications
builds
clusteroperators
clusterversions
consoles
dnses
featuregates
images
infrastructures
ingresses
networks
oauths
operatorhubs
projects
proxies
schedulers
```

```
$ oc get <config-api-resource-name> -o yaml
```

除錯實操介紹 – 配置資源文件

可以用 `oc explain` 顯示資源的文件。這個指令可以讓操作人員了解資源定義的每個欄位資訊。

```
$ oc explain <resource>
```

```
$ oc explain pods
```

```
$ oc explain pods.spec.containers
```

```
$ oc explain pods --recursive=true
```

除錯實操介紹 – 配置資源文件

在 Web UI 介面上，你可以透過 sidebar 展現某個資源文件。讓使用者了解資源可以填入的資訊。

Custom Resource Definitions > Custom Resource Definition Details

CRD configs.imageregistry.operator.openshift.io

Actions ▾

Details YAML Instances

1 kind: CustomResourceDefinition
2 apiVersion: apiextensions.k8s.io/v1beta1
3 metadata:
4 name: configs.imageregistry.operator.openshift.io
5 selfLink: >-
6 /apis/apiextensions.k8s.io/v1beta1/customresourcedefinitions/configs.imageregistry.
7 uid: 42e71785-18df-42d2-8cc9-88479f8d0e2f
8 resourceVersion: '519'
9 generation: 1
10 creationTimestamp: '2020-06-29T08:38:51Z'
11 spec:
12 subresources:
13 status: {}
14 names:
15 plural: configs
16 singular: config
17 kind: Config
18 listKind: ConfigList
19 scope: Cluster
20 conversion:
21 strategy: None

View shortcuts

Custom Resource Definition

Schema

CustomResourceDefinition > spec

spec describes how the user wants the resources to appear

additionalPrinterColumns array

additionalPrinterColumns specifies additional columns returned in Table output. See <https://kubernetes.io/docs/reference/using-api/api-concepts/#receiving-resources-as-tables> for details. If present, this field configures columns for all versions. Top-level and per-version

• metadata object

View details

• spec object – required

spec describes how the user wants the resources to appear

View details

• status object

status indicates the actual state of the CustomResourceDefinition

View details

Save Reload Cancel Download

除錯實操介紹 – 應用構建

如果構建失敗的錯誤訊息是 **requested access to the resource is denied** 則需確認超過 image 在 Project 的配額是否超過。

```
$ oc describe quota -n myproject
```

```
[maemmanu-redhat.com@clientvm 0 ~]$ oc describe quota -n myproject
Name:           example
Namespace:      myproject
Resource        Used   Hard
-----
limits.cpu      0      2
limits.memory   0      2Gi
pods            1      1
requests.cpu    0      1
requests.memory 0      1Gi
```

除錯實操介紹 – 應用構建

The screenshot shows the Red Hat OpenShift Container Platform interface. On the left, a sidebar menu lists various resources: Replica Sets, Replication Controllers, Horizontal Pod Autoscalers, Networking, Storage, Builds, Monitoring, Compute, User Management, Administration, Cluster Settings, Namespaces, Resource Quotas, Limit Ranges, and Custom Resource Definitions. The 'Administration' section is currently selected. In the main content area, a project named 'myproject' is selected. The build status indicates a failure: 'Annotations' (0 Annotations), 'Created At' (8 minutes ago), and 'Owner' (No owner). A red arrow points from the text 'Pod 配額已滿，構建失敗' (Pod quota exceeded, build failed) to the 'Used' column of the 'Resource Quota Details' table. The table shows the following data:

Resource Type	Capacity	Used	Max
limits.cpu	○	0	2
limits.memory	○	0	2Gi
pods	●	1	1
requests.cpu	○	0	1
requests.memory	○	0	1Gi

A red box highlights the row for 'pods'. A red arrow points from the text 'Pod 配額已滿，構建失敗' to the 'Used' value of 1 in the 'pods' row.

除錯實操介紹 – 應用部署

你可以使用 **oc api-resources** 指令及 **oc get** 指令可以列出 namespace 中所有實例及其資源：

```
$ oc api-resources --verbs=list --namespaced -o name | xargs -n 1 oc get --show-kind --ignore-not-found -n <namespace>
```

使用 **oc get** 和 **oc describe** 指令去檢查 namespace 中存在的資源

使用 **oc status** 指令去確認現在 project 內目前的資源狀態

```
[maemmanu-redhat.com@clientvm 0 ~]$ oc status -n myproject
In project myproject on server https://api.myocp.sandbox879.opentlc.com:6443

http://http-myproject.apps.myocp.sandbox879.opentlc.com to pod port 8080-tcp (svc/http)
  deployment/http deploys istag/http:latest <
    bc/http source builds https://github.com/sclorg/httpd-ex.git on openshift/httpd:2.4
    deployment #2 running for 32 seconds - 1 pod
    deployment #1 deployed about a minute ago

1 info identified, use 'oc status --suggest' to see details.
```

除錯實操介紹 – 應用部署

確認 Deployment 事件來確認問題

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar is titled 'Workloads' and includes options like 'Pods', 'Deployments' (which is highlighted with a red box), 'Deployment Configs', 'Stateful Sets', 'Secrets', 'Config Maps', 'Cron Jobs', 'Jobs', 'Daemon Sets', 'Replica Sets', 'Replication Controllers', and 'Horizontal Pod Autoscalers'. The main content area shows a 'Project: myproject' for a 'D http' deployment. The top navigation bar has tabs for 'Details', 'YAML', 'Replica Sets', 'Pods', 'Environment', and 'Events' (which is also highlighted with a red box). Below this, there's a 'Streaming events...' indicator and a list of three events:

- D http** Generated from deployment-controller NS myproject 5 minutes ago
Scaled down replica set http-77ccbbfb4d to 0
- D http** Generated from deployment-controller NS myproject 5 minutes ago
Scaled up replica set http-6c9d4757c9 to 1
- D http** Generated from deployment-controller NS myproject 6 minutes ago
Scaled up replica set http-77ccbbfb4d to 1

At the bottom, it says 'There are no events before a few seconds ago'.

除錯實操介紹 – 應用部署

確認應用事件和日誌

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar is titled "Workloads" and includes options for "Pods" (which is selected and highlighted with a red box), "Deployments", "Deployment Configs", "Stateful Sets", "Secrets", "Config Maps", "Cron Jobs", "Jobs", "Daemon Sets", "Replica Sets", "Replication Controllers", and "Horizontal Pod Autoscalers". The main content area shows a project named "myproject". Under "Pods", a specific pod named "http-6c9d4757c9-tsqv4" is listed as "Running". Below the pod name are tabs for "Details", "YAML", "Environment", "Logs" (which is selected and highlighted with a red box), "Events" (which is also highlighted with a red box), and "Terminal". On the right side of the pod details, there are "Actions" and "Actions" dropdown menus. Below the tabs, there is a "Log streaming..." button and a dropdown menu set to "http". To the right of the logs, there are links to "Show in Kibana", "Download", and "Expand". The log output itself shows 18 lines of text, including system and application logs related to Apache and ModSecurity.

```
[Mon Jun 29 13:28:41.841320 2020] [:notice] [pid 1] ModSecurity: PCRE compiled version="8.32 "; loaded version="8.32 2012-11-30"
[Mon Jun 29 13:28:41.841323 2020] [:notice] [pid 1] ModSecurity: LUA compiled version="Lua 5.1"
[Mon Jun 29 13:28:41.841325 2020] [:notice] [pid 1] ModSecurity: YAJS compiled version="2.0.4"
[Mon Jun 29 13:28:41.841327 2020] [:notice] [pid 1] ModSecurity: LIBXML compiled version="2.9.1"
[Mon Jun 29 13:28:41.841329 2020] [:notice] [pid 1] ModSecurity: Status engine is currently disabled, enable it by set SecStatus
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.128.2.16. Set the 'ServerName' d
[Mon Jun 29 13:28:41.933053 2020] [ssl:warn] [pid 1] AH01909: 10.128.2.16:8443:0 server certificate does NOT include an ID which
[Mon Jun 29 13:28:41.933235 2020] [http2:warn] [pid 1] AH10034: The mpm module (prefork.c) is not supported by mod_http2. The mp
[Mon Jun 29 13:28:41.934222 2020] [lbmethod_heartbeat:notice] [pid 1] AH02282: No slotmem from mod_heartbeat
[Mon Jun 29 13:28:41.938899 2020] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.34 (Red Hat) OpenSSL/1.0.2k-fips configured -
[Mon Jun 29 13:28:41.938943 2020] [core:notice] [pid 1] AH00094: Command line: 'httpd -D FOREGROUND'
```

除錯實操介紹 – 應用部署

當應用無法訪問，但日誌無錯誤時，確認 service 物件是否正確定義

Application is not available

The application is currently not serving requests at this endpoint. It may not have been started or is still starting.

i Possible reasons you are seeing this page:

- **The host doesn't exist.** Make sure the hostname was typed correctly and that a route matching this hostname exists.
- **The host exists, but doesn't have a matching path.** Check if the URL path was typed correctly and that the route was created using the desired path.
- **Route and path matches, but all pods are down.** Make sure that the resources exposed by this route (pods, services, deployment configs, etc) have at least one pod running.

```
$ oc logs <pod-name>  
App is ready at : 8080
```

除錯實操介紹 – 應用部署

```
$ oc describe svc/http -n myproject
```

```
[maemmanu-redhat.com@clientvm 127 ~]$ oc describe svc/http -n myproject
Name: http
Namespace: myproject
Labels: app=http
         app.kubernetes.io/component=http
         app.kubernetes.io/instance=http
         app.kubernetes.io/name=httpd
         app.kubernetes.io/part-of=http-app
         app.openshift.io/runtime=httpd
         app.openshift.io/runtime-version=2.4
Annotations: app.openshift.io/vcs-ref: master
             app.openshift.io/vcs-uri: https://github.com/sclorg/httpd-ex.git
             openshift.io/generated-by: OpenShiftWebConsole
Selector: app=http,deploymentconfig=http
Type: ClusterIP
IP: 172.30.229.55
Port: 8080-tcp 8080/TCP
TargetPort: 8080/TCP
Endpoints: 10.128.2.16:8080
Port: 8443-tcp 8443/TCP
TargetPort: 8443/TCP
Endpoints: 10.128.2.16:8443
Session Affinity: None
Events: <none>
```

如果 service 端口沒開，則須確認
selectors 是否正確對應到
Deployment

確認 service 有對應到端口(Endpoint)

除錯實操介紹 – 應用部署

確認 pod 事件來確認可能的錯誤問題

```
$ oc describe pod <pod-name>
Events:
Type Reason ... Message
---- -----
Warning FailedScheduling ... 0/5 nodes are available:5 node(s) didn't match node selector.
```

Watch out for Wrong Node Selectors or taints when running into **FailedScheduling** issues

The screenshot shows the OpenShift web interface for a deployment named 'myproject'. The left sidebar lists various workloads: Operators, Workloads (Pods selected), Deployments, Deployment Configs, Stateful Sets, Secrets, Config Maps, Cron Jobs, and Jobs. The main panel displays deployment configuration details for 'myproject'. It includes sections for Namespace (NS myproject), Labels (app=http, app.kubernetes.io/component=http, app.kubernetes.io/instance=http, app.kubernetes.io/name=httpd, app.kubernetes.io/part-of=http-app, app.openshift.io/runtime=httpd, app.openshift.io/runtime-version=2.4), Pod Selector (app=http), Node Selector (example=wrong-selector, highlighted with a red box), and Tolerations (0 Tolerations). A red box highlights the 'Node Selector' field.

Max Unavailable
25% of 1 pod

Max Surge
25% greater than 1 pod

Progress Deadline Seconds
600 seconds

Min Ready Seconds
Not Configured

除錯實操介紹 – 應用部署

如果 **FailedScheduling** 問題不是印錯誤的 node selector 或 taints，可能是因為缺少資源以至於無法部署至對應 node。

```
$ oc describe pod <pod-name>
Events:
Type    Reason        ...        Message
----    -----        ...        -----
Warning FailedScheduling ... Failed for reason PodExceedsFreeCPU and possibly others.
```

1. 針對叢集增加節點。
2. 透過內部資源調整，將過度使用 pod 資源釋放出來。
3. 確認 Pod 的資源請求合理，未超過 Node 本身。舉例來說，如果 Node 的總空間是 1 cpu，則當 Pod 請求 1+ cpu 則無法被安排部署。
4. 確認 pod 部署位置策略。

Lab 03 - APP trouble shooting

Q&A 和問卷



Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

 [linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)

 [youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)

 [facebook.com/redhatinc](https://www.facebook.com/redhatinc)

 twitter.com/RedHat