

KUBERNETES

# Kubernetes 私有雲實作指南： 企業 IT 現代化的第一步



資深技術顧問  
Andy Chung

August 2025

鉅迪資訊



# 介紹

HTIC

# 鍾迪 Andy Chung

- 現職：
  - 鉅迪資訊股份有限公司-資深技術顧問
- 學歷：
  - 加州長堤大學 CSULB 碩士
- 專案建置經歷 (22年)：
  - 政府、電信、航空、學校、海外：50個專案+
- 專業能力：
  - 身份管理系統規劃與建置
  - 帳號整合同步規劃與建置
  - 單一簽入系統規劃與建置
  - 特權管理系統規劃與建置
  - 多因素認證系統規劃與建置
  - Kubernetes系統規劃與建置
  - Java / SQL/ LDAP/ Scripting 開發

台大網路與資安技術研討會:

2022 - 運用零信任策略落實身份治理與單一簽入整合。

2023 - 透過單一簽入解決方案整合地端應用系統與雲端服務認證。

2024 - 深入探討特權帳號管理系統整合運用。

2025 - Kubernetes 私有雲實作指南：企業 IT 現代化的第一步。



# 課程大綱

1. 為何選擇 Kubernetes 打造私有雲
2. Kubernetes 架構與核心元件解析
3. 監控與日誌管理
4. 應用部署與資源管理
5. 安全性與權限控管
6. 整合 CI/CD 流程
7. 開源軟體介紹
8. 總結

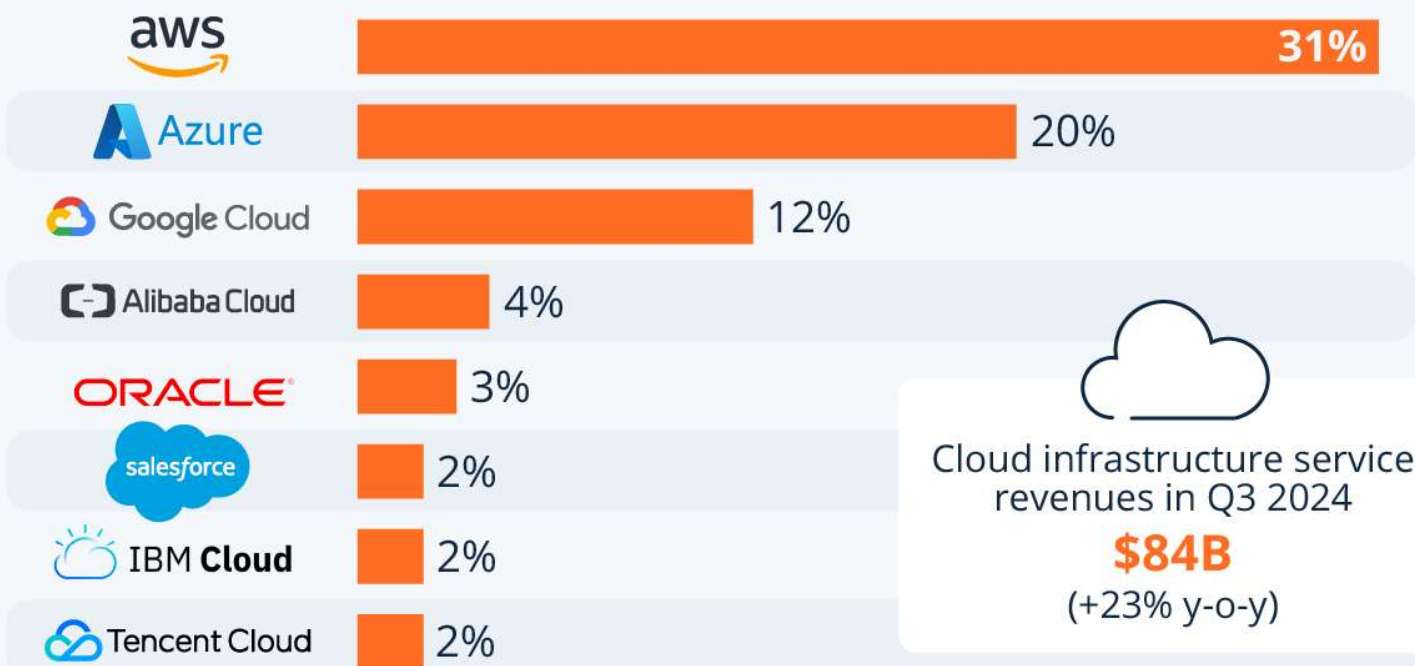


# 為何選擇 Kubernetes 打造私有雲

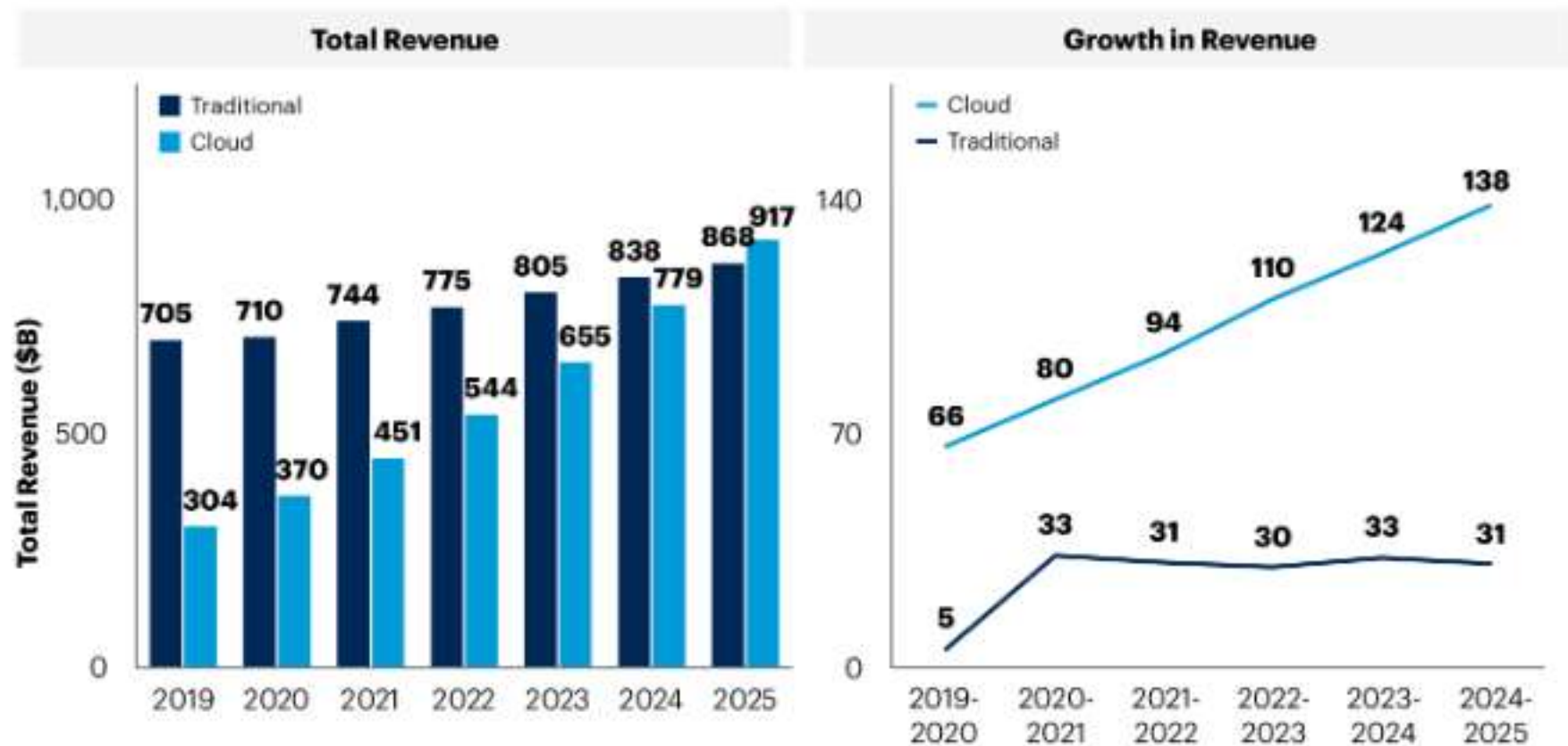
HTIC

# Amazon Maintains Dominant Lead in the Cloud Market

Worldwide market share of leading cloud infrastructure service providers in Q3 2024\*



## 2025 年將超越傳統 IT 支出



Source: Gartner

758067\_C

Gartner

# 公有雲重大事件

## AWS 不當刪除使用者資料事件

發生在 2025 年 3 月左右，屬於雲端服務操作或政策爭議型的資安 / 營運風險案例。一名軟體工程師稱其十年累積的 AWS 雲端資料因驗證失敗遭刪除，AWS 說是依既有政策操作，但疑涉內部誤判，導致無法復原的重要資源消失。此事件凸顯雲端帳號管理與資料保護的風險。

## Microsoft SharePoint 伺服器遭零時差攻擊

2025年7月 Microsoft 發出警報，指出全球數萬台 SharePoint ( 內部部署版 ) 伺服器遭到 “零時差” 漏洞 (CVE-2025-53770) 利用，駭客可執行偽造 ( spoofing ) 攻擊，模仿內部使用者或服務進行未授權操作。影響範圍至少 75 台伺服器已被攻破，涵蓋企業機構、政府、學校、電信公司等。

## Azure Blob 儲存錯誤設定導致履歷洩漏

2025年7月，雲端人力資源平台 TalentHook 將 Azure Blob 容器公開設定，導致近 2,600 萬份美國求職者履歷資料曝光，包括姓名、聯絡方式、教育與工作經歷等 IT Pro+1TechRadar+1。風險說明：此類資訊極易被用於訂製型釣魚、詐騙或社交工程攻擊，特別由具資安能力的駭客集團 ( 如朝鮮 Lazarus ) 濫用 IT ProTechRadar。

## Azure 被用於監控與戰爭行動

2025 年 8 月英國《The Guardian》揭露的重大雲端倫理與資安爭議案例，涉及 雲端平台被政府軍事單位長期用於監控。2022 年起，以色列 Unit 8200 利用 Azure 分隔區儲存每日數百萬通話錄音，並應用於軍事行動決策。雖然 Microsoft 聲稱不知情，但內部資料顯示其與該軍事單位具緊密聯繫，引發雲端平台倫理與使用者保護的重大爭議。



# 依賴公有雲的壞處

## 1. 長期成本不透明、可能更高

- 容易出現「雲費用爆炸」( cloud bill shock )。
- 特定服務 ( 如資料傳輸、儲存 ) 隱含高額費用。

## 2. 敏感資料與合規風險

- 無法完全掌控資料的儲存位置與存取權限。
- 某些產業 ( 如金融、政府、醫療 ) 對資料主權要求嚴格。

## 3. 過度依賴單一供應商 ( Vendor Lock-in )

- 選擇某家公有雲平台後，系統架構、API、工具等可能深度綁定。
- 未來若想遷移至其他雲平台或回歸地端，成本與技術挑戰巨大。

## 4. 網路依賴度高

- 所有存取必須透過網路，若網路品質不穩將直接影響系統可用性與效能。
- 資料傳輸延遲無法避免，對低延遲應用 ( 如工控、交易系統 ) 不利。

## 5. 控制權有限

- 雲平台維運、升級、故障修復由供應商主導，用戶難以介入或優先處理。
- 若供應商發生服務中斷 ( 如 AZURE大當機 )，無法自力處理或快速恢復。

## 6. 安全事件與責任不完全掌控

- 雲端平台的安全模型多為「共享責任制」：雲商負責平台安全，使用者負責資料與設定。
- 若設定錯誤，仍可能造成重大資安事件。

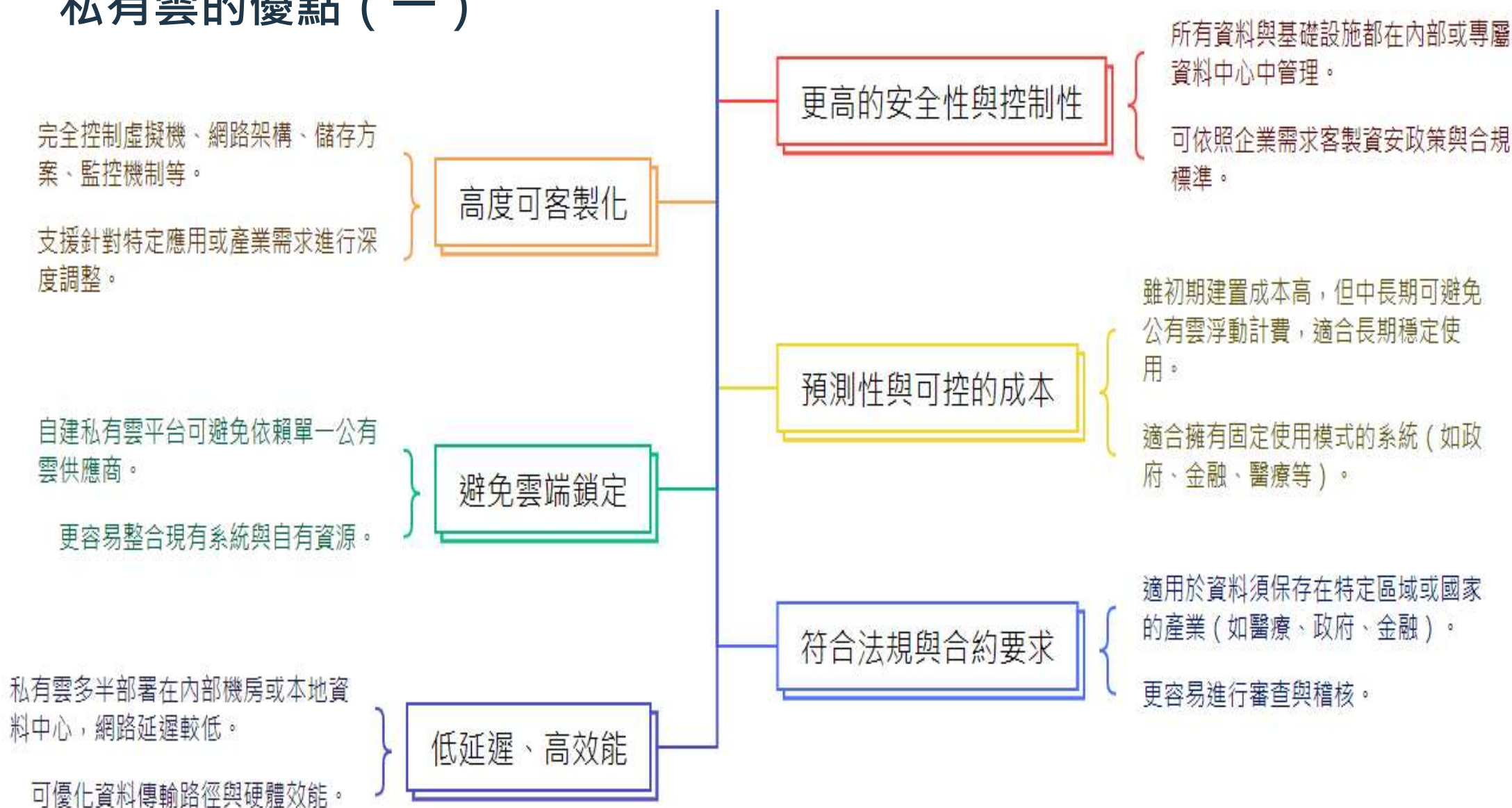
## 7. 某些舊系統或應用不適合搬上雲

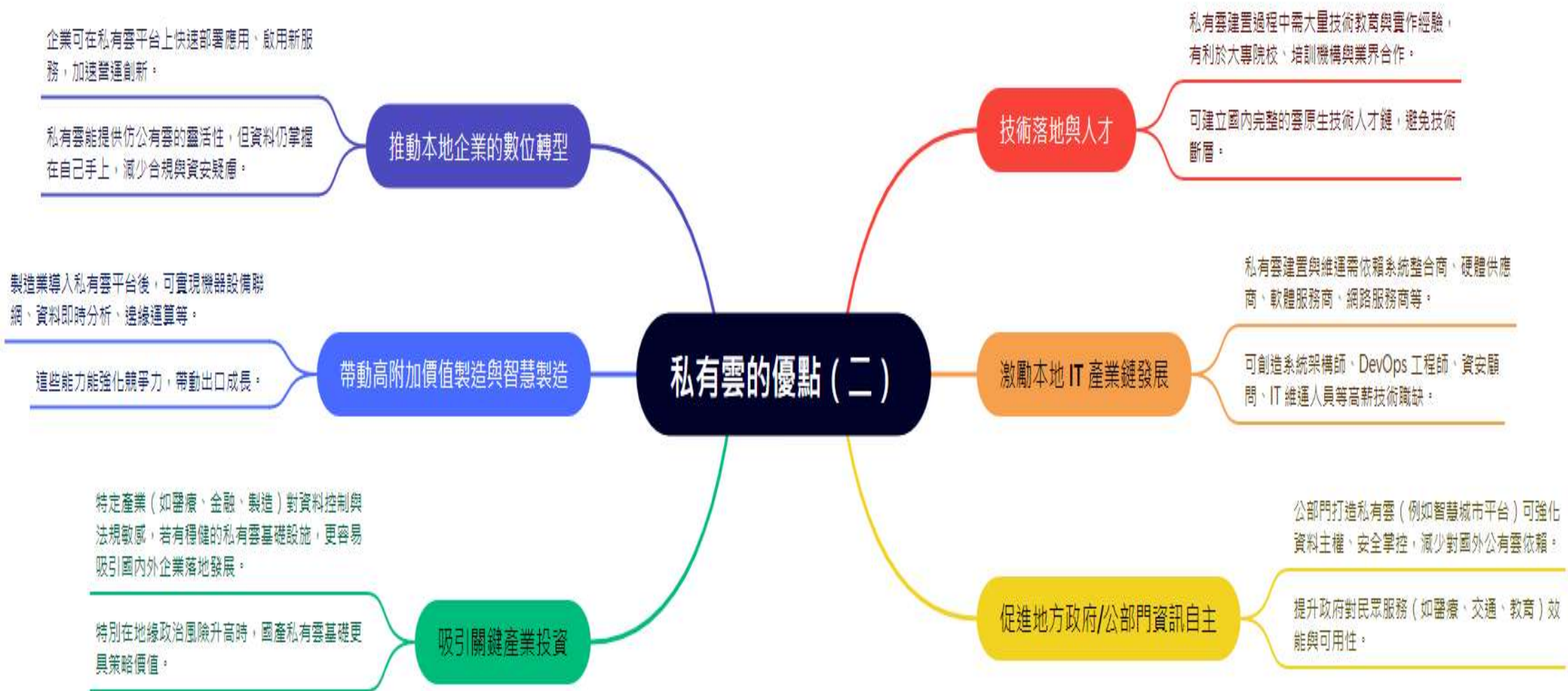
- 傳統 monolithic 系統或需低層硬體控制的應用難以部署在公有雲。
- 若強行遷移，可能需大幅重構，成本與風險高。

## 8. 難以實現客製化或特殊需求

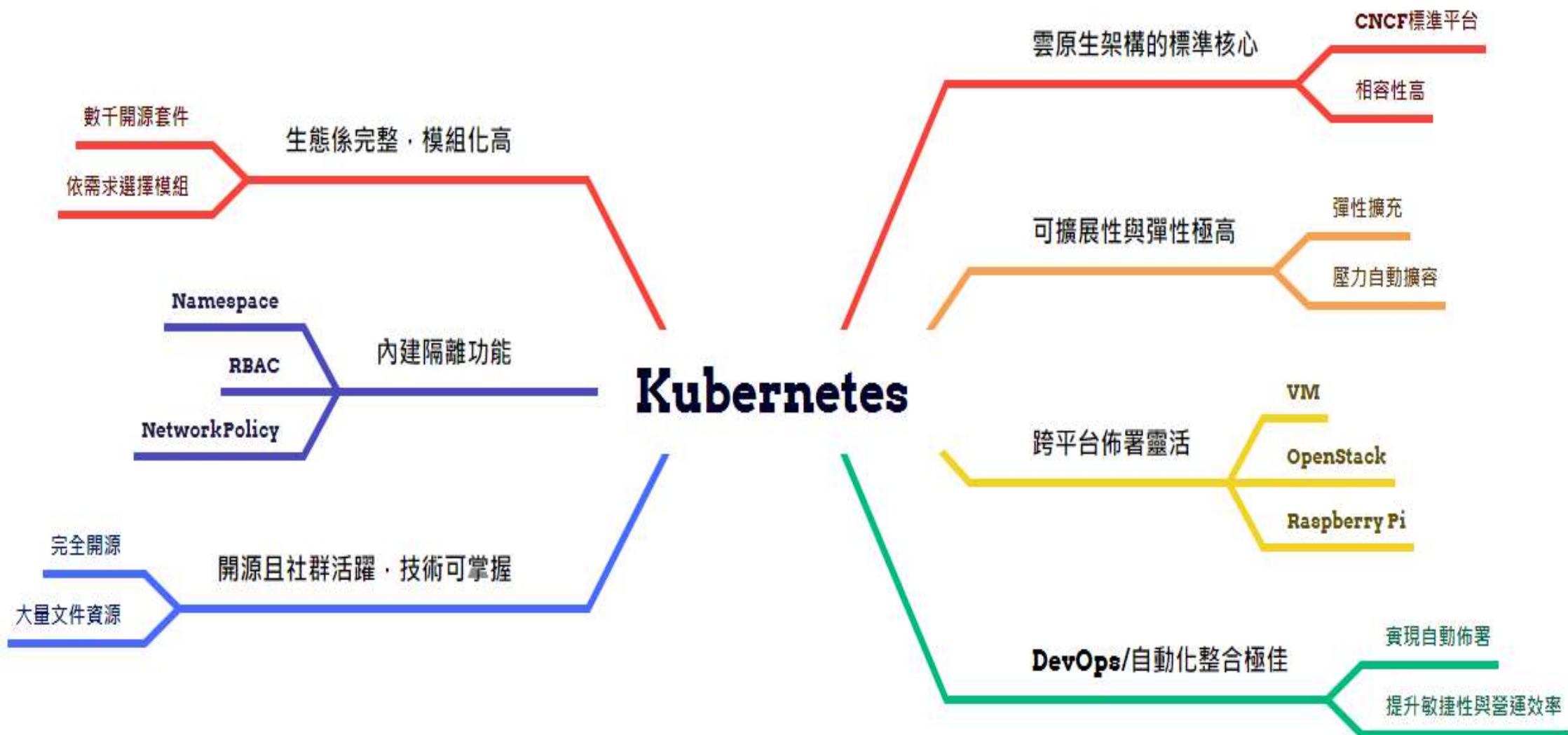
- 公有雲提供標準化服務，若需特殊軟硬體需求，實現困難。
- 遇到特殊法規、審計或特殊內部流程，也可能無法完全配合。

## 私有雲的優點（一）





# Kubernetes 是打造私有雲的最佳選擇





# Kubernetes發展重大事件時間軸表格



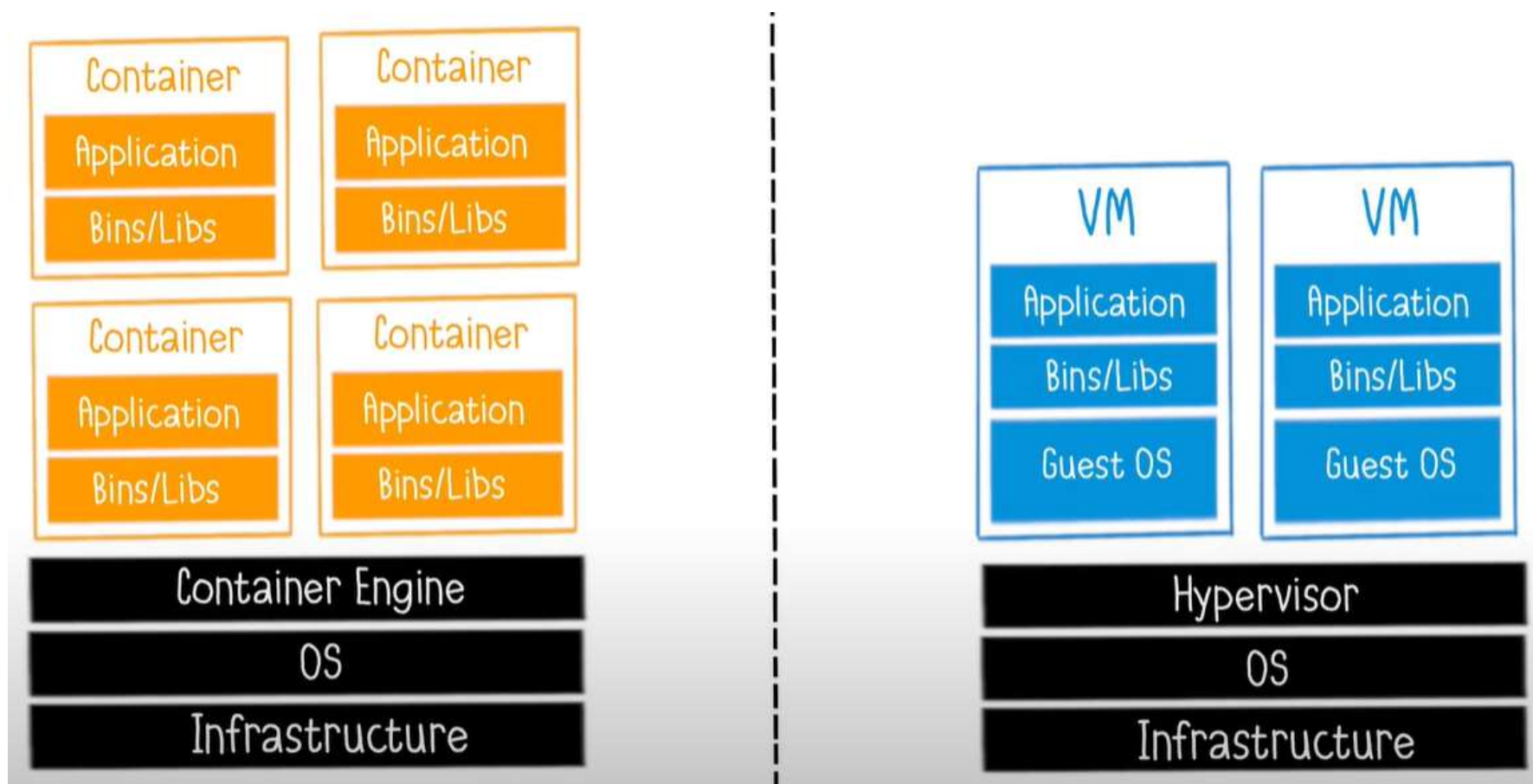


# Kubernetes 架構與核心元件解析

HTIC

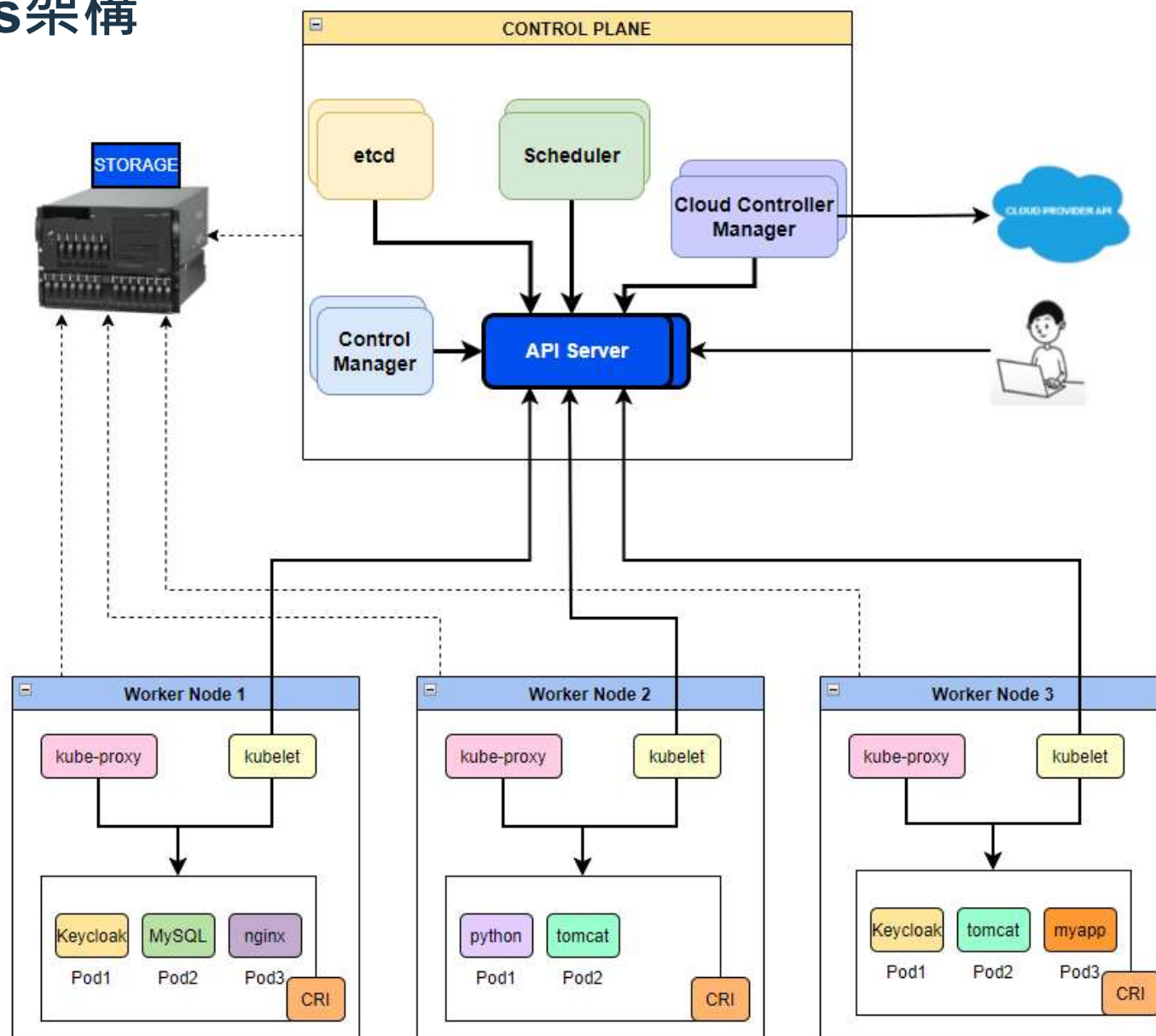
# Kubernetes是什麼？

Kubernetes ( 簡稱 K8S ) 是一個開源的容器編排平台。它最初由 Google 開發，現在由 CNCF ( Cloud Native Computing Foundation ) 維護。它旨在提供「跨主機叢集的自動部署、擴充以及執行應用程式容器的平台」。



Kubernetes 是一個幫你自動化管理成千上萬個容器的系統。

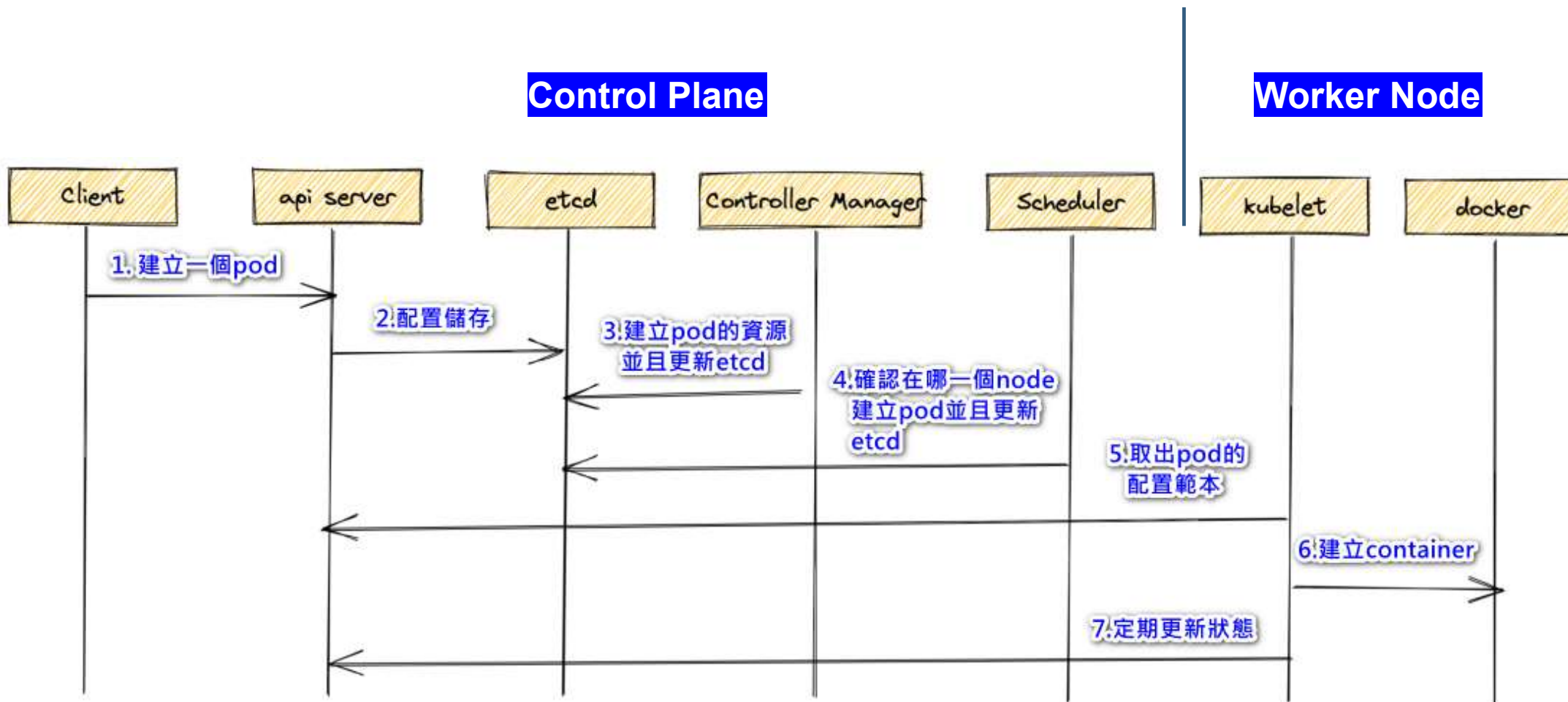
# Kubernetes架構



單一集群支援到 5,000 nodes、15 萬 Pods、30 萬 Container。



## 建立pod的流程 (圖示)



# Kubernetes Cluster Hardware

<input type="checkbox"/>	Virtual machine	Status	Used space	Guest OS	Host name	Host CPU	Host mem
<input type="checkbox"/>	HTIC-K8S-MASTER-1 (1...	✓	36.1 GB	Ubuntu Linux (...)	k8s-master-01	784 MHz	7.96 GB
<input type="checkbox"/>	HTIC-K8S-WORK-1 (192...	✓	61.2 GB	Ubuntu Linux (...)	k8s-worker-01	691 MHz	15.96 GB
<input type="checkbox"/>	HTIC-K8S-WORK-2 (19...	✓	71.03 GB	Ubuntu Linux (...)	k8s-worker-02	754 MHz	15.94 GB



最低需求:

- CPU: 1
- RAM: 2G
- HD: 20G

# Kubernetes各種知名開源發行版本

## 輕量化 / 邊緣運算 / 開發者取向

發行版名稱	備註
k3s	Rancher 推出的輕量級 K8s，適合 IoT / 邊緣應用
MicroK8s	Canonical 的單一安裝包，適合開發/測試
Minikube	本地開發與測試用途，可跑在 VM 或裸機上
Kind	Kubernetes in Docker，適用於 CI/CD

重點	MicroK8s	Minikube	Kind 
⚙️ 安裝複雜度	★★	★★★★★	★★★★★★
🎯 學習門檻	★★★★	★★	★
🚀 測試效能	★★★★	★★★★	★★★★★★
📦 資源需求	中	中	最低
🏗️ CI/CD 適合	一般	一般	最佳

- ▶ Documentation
- ▶ Getting started
- ▶ Concepts
- ▶ Tasks
- ▼ Tutorials
  - Hello Minikube
    - ▶ Learn Kubernetes Basics
    - ▶ Configuration
    - ▶ Security
    - ▶ Stateless Applications
    - ▶ Stateful Applications
    - ▶ Cluster Management
    - ▶ Services
  - ▶ Reference
  - ▶ Contribute

<https://kubernetes.io/docs/tutorials/>

# Kubernetes衍生的各種知名發行版本

## 私有雲

### 企業級 Kubernetes 發行版

發行版名稱	提供廠商	特點
Red Hat OpenShift	Red Hat	整合完整 CI/CD、RBAC、UI 介面、企業級支援
VMware Tanzu Kubernetes Grid (TKG)	VMware	整合 vSphere、NSX 與 Tanzu 生態系統
Rancher Kubernetes	SUSE	多叢集集中管理、圖形化介面、輕量化
Mirantis Kubernetes Engine (MKE)	Mirantis	前身為 Docker EE，提供企業支援
Canonical Kubernetes	Canonical ( Ubuntu )	原生 Kubernetes，加值支援與整合

### 雲端原生託管型 Kubernetes

發行版名稱	雲端供應商
Amazon EKS	AWS
Azure AKS	Microsoft Azure
Google GKE	Google Cloud
Oracle OKE	Oracle Cloud
IBM IKS / ROSA	IBM / Red Hat

## 公有雲

# 系統期待



## 系統維護人員：

1. 服務掉下來自己重起
2. 流量飆高時可輕易橫向擴充應用系統資源
3. 伺服器掛掉時，應用系統可持續運作
4. 沒人維護的系統，持續運作
5. 系統有安全與權限管控設定
6. 集中管理

## 使用者：

- 系統不要掛掉



## 軟體開發人員：

1. 方便程式上版
2. 上版時盡量不影響現行使用者
3. 發生問題可快速回復前一版

## 主管：

1. 親民價格的穩定系統
2. 可符合資安的系統





## 情境一

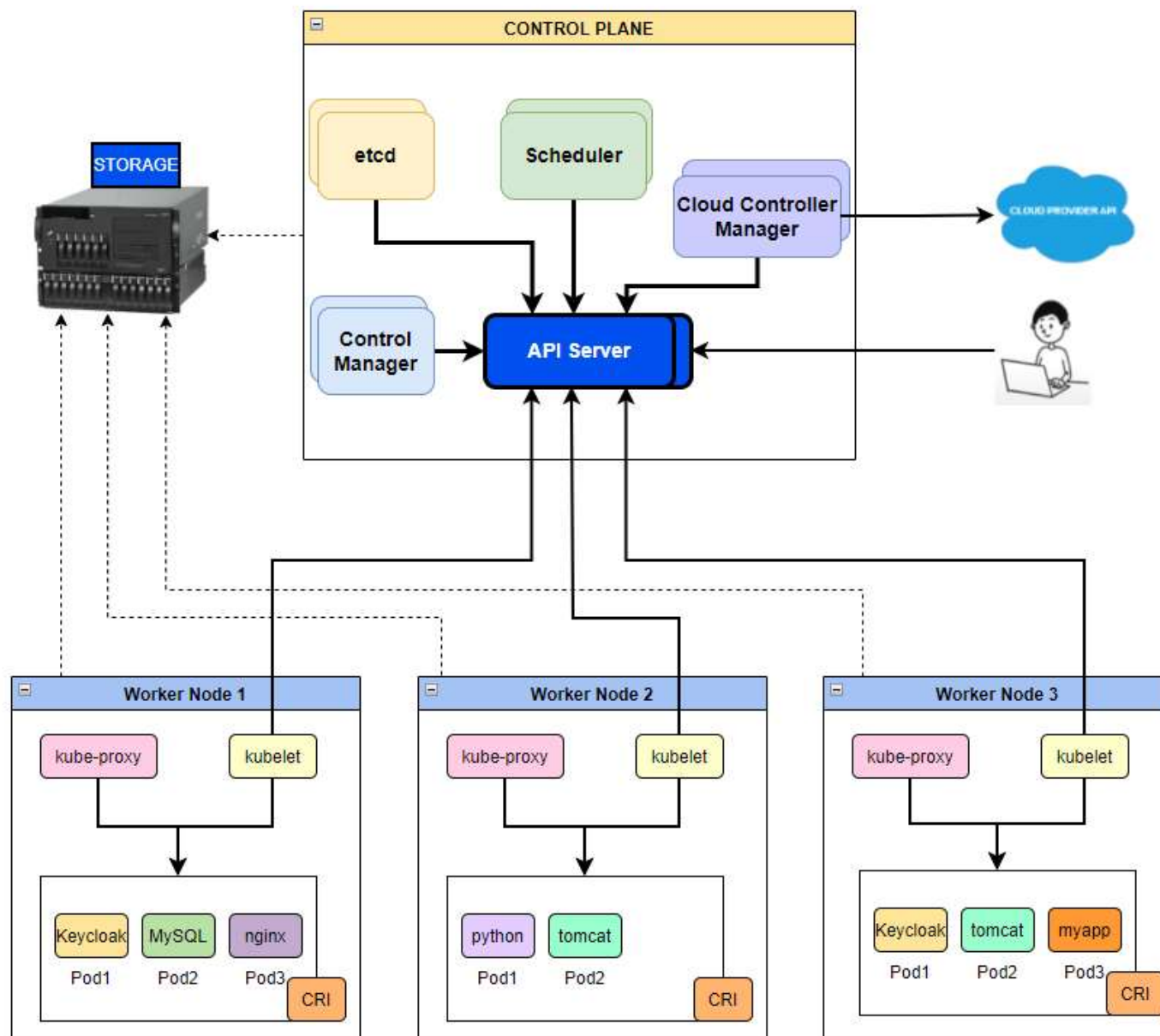
因為大量requests

- postgres x 1 -> 2
- keycloak x 2 -> 3
- myapp x 1 -> 6

自動成長不需人力介入

使用量低

自動回收資源



## 情境二

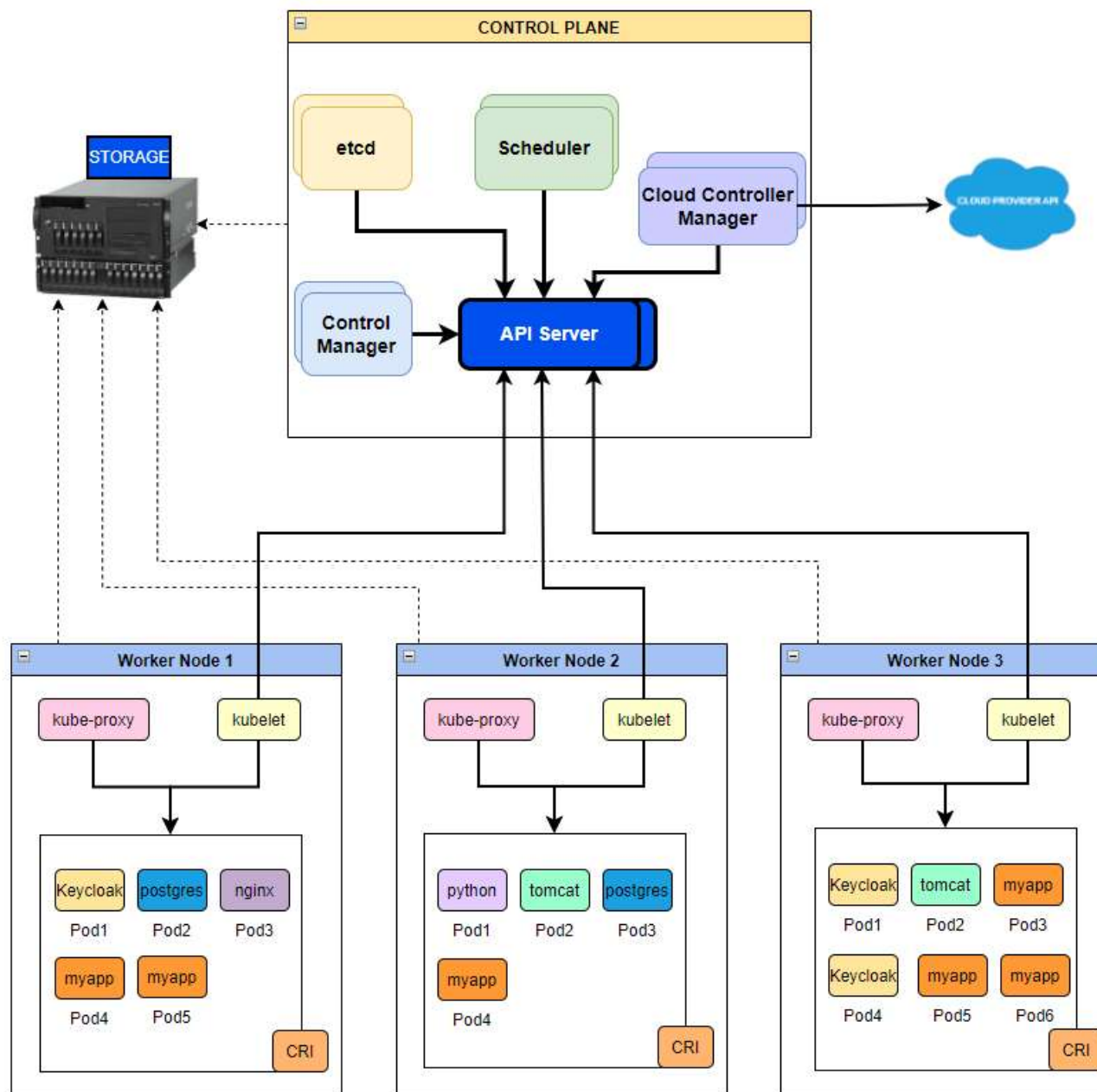
因為大量requests  
Node 2 掛掉

- 維持所有pods

自動轉移不需人力介入

Nodes恢復

Pods自動分散轉移



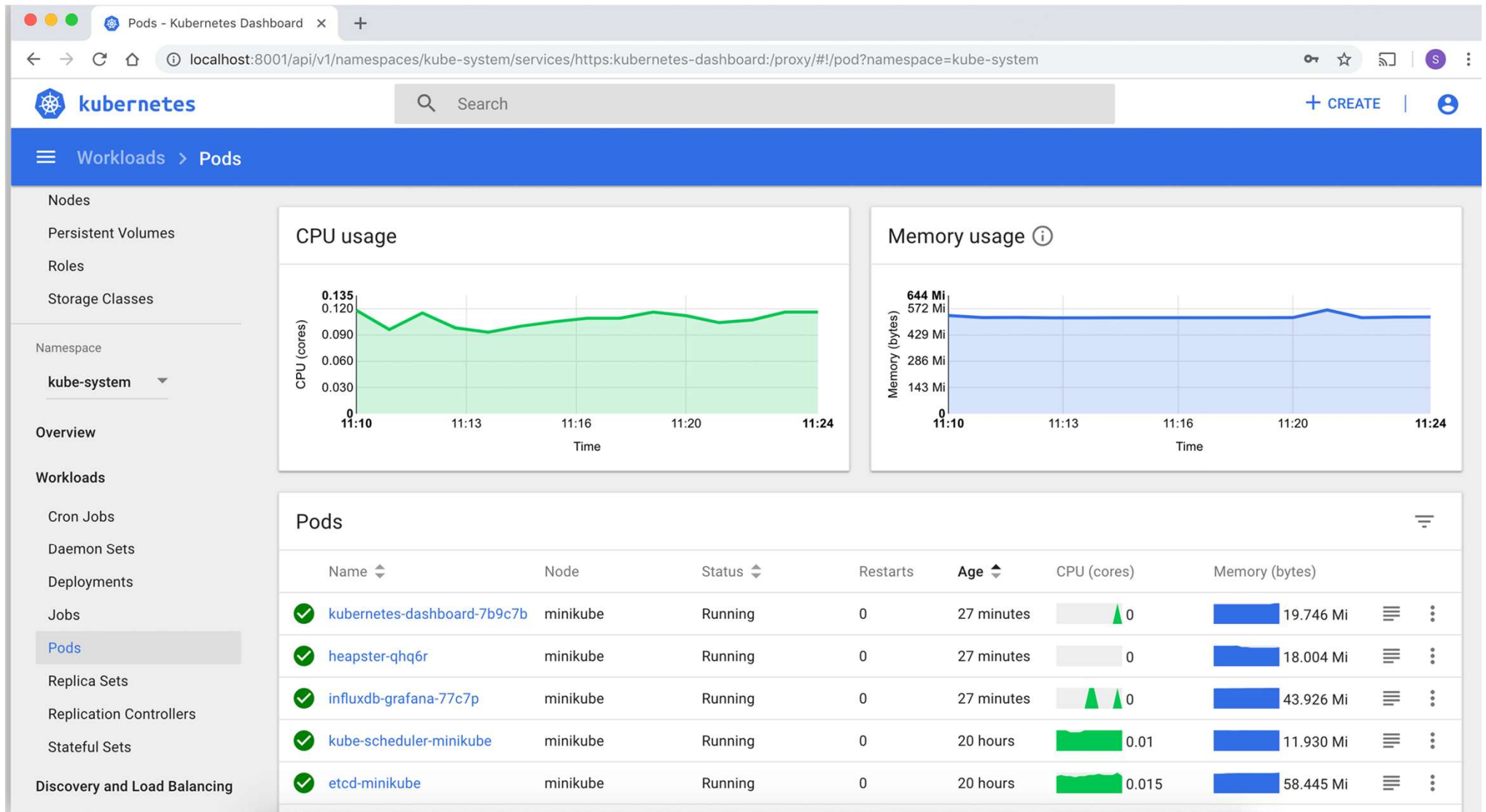


# 監控與日誌管理

HTIC



# Kubernetes Dashboard



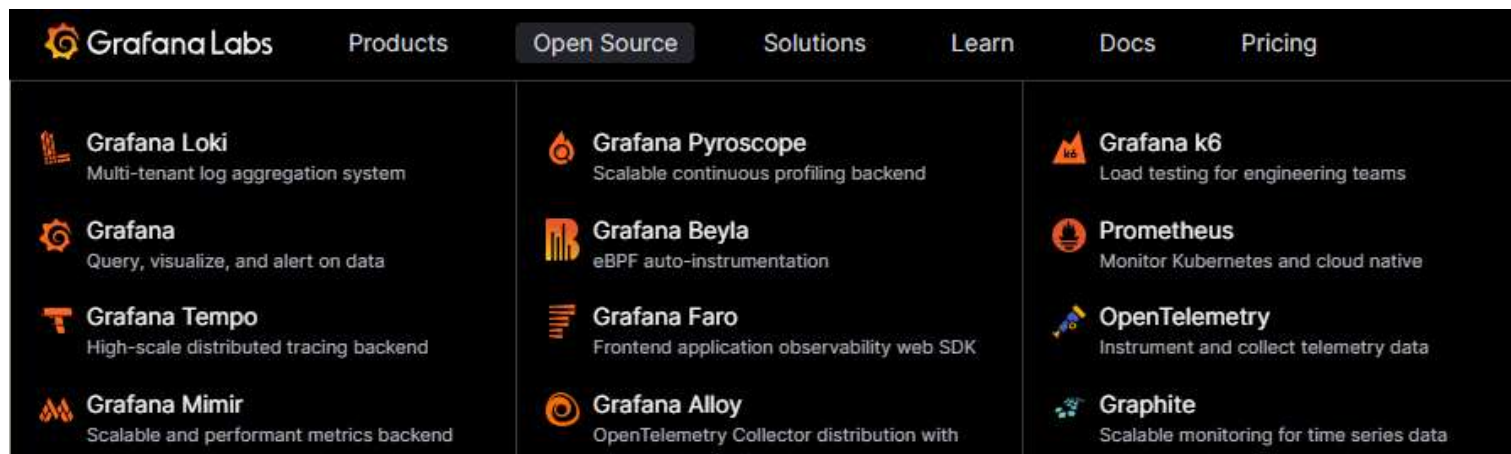


Grafana 是一個開源的數據視覺化與儀表板工具，可以將來自各種資料來源（如 Prometheus、InfluxDB、Elasticsearch、MySQL、Loki 等）的數據做成圖表、表格、警報等視覺化呈現。它最常用於系統監控、效能分析、商業指標追蹤等場景，並與 Prometheus 一起搭配成為 DevOps 與 SRE 常用的監控解決方案。

Grafana 常見使用場景：

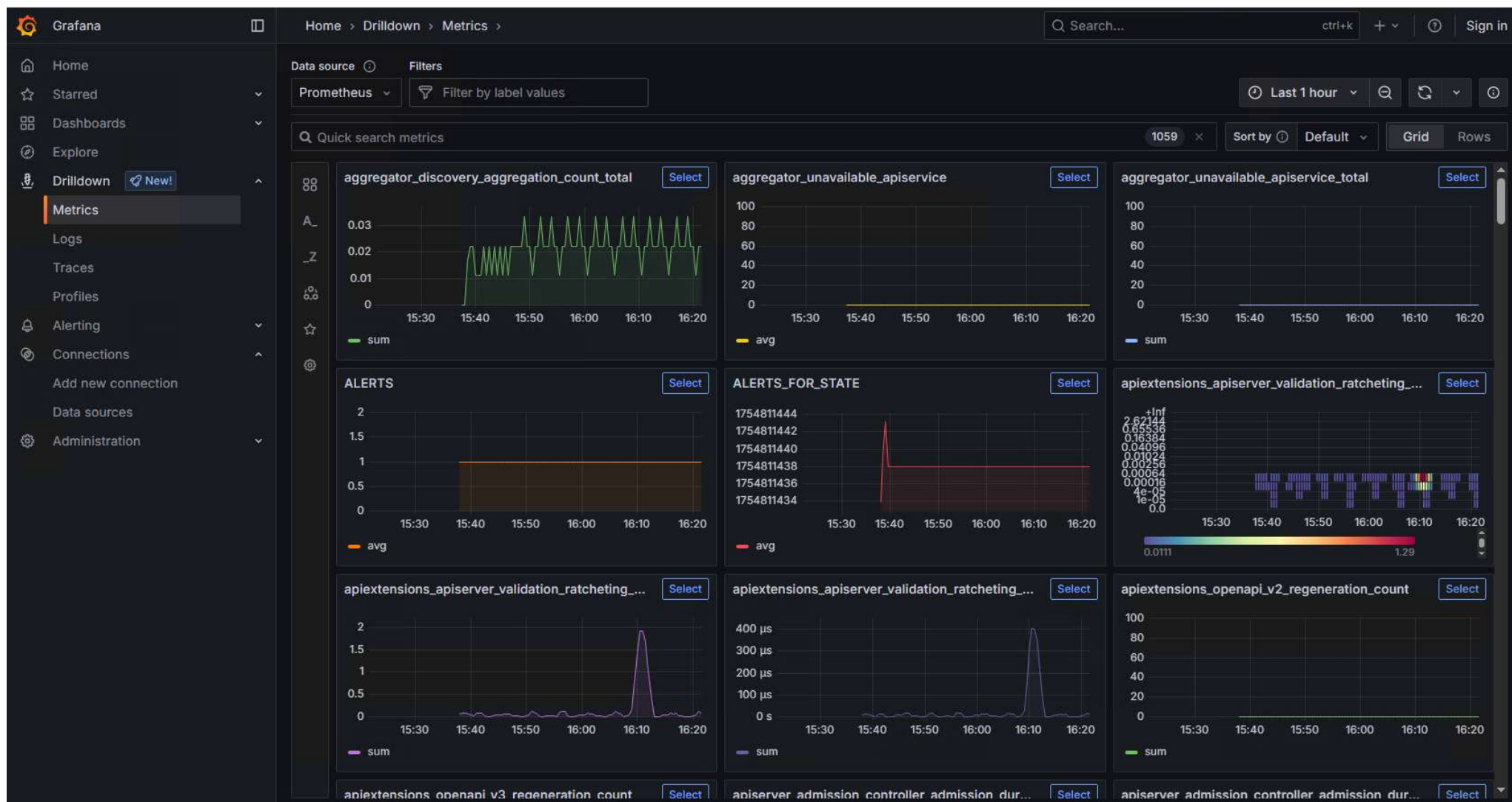
- 系統監控 (CPU、記憶體、網路流量、磁碟 I/O)
- Kubernetes 監控 (Pod 效能、Node 狀態、Deployment 變化)
- APM(應用效能監控)(HTTP 請求數、錯誤率、回應時間)
- 商業指標視覺化 (用戶註冊數、訂單量、營收趨勢)
- 日誌分析 (搭配 Loki 或 Elasticsearch 搜索與分析日誌)

# Grafana

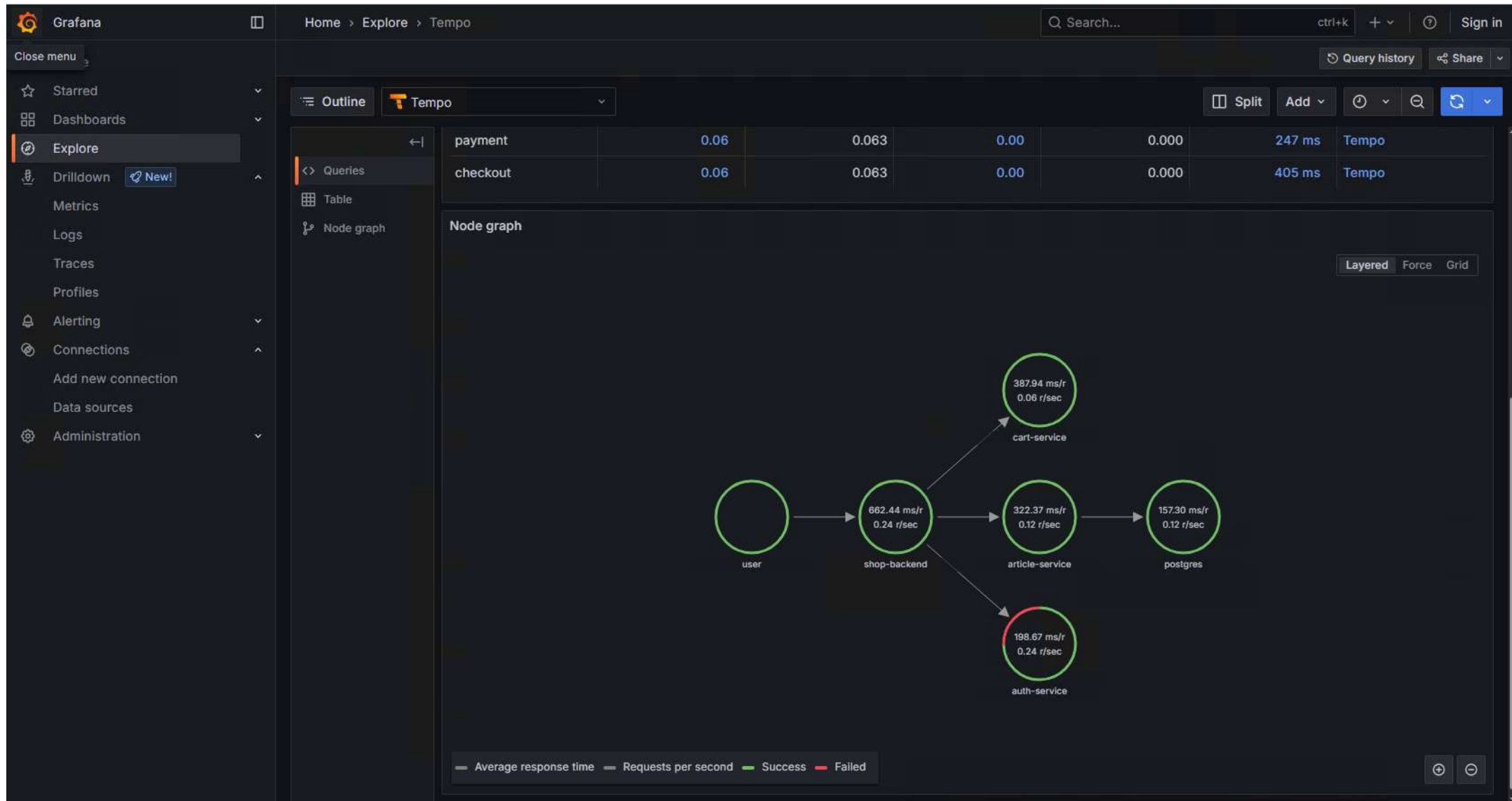


類別	元件名稱	功能簡介
 前端可視化	<b>Grafana</b> (主體)	顯示 Dashboard、設定告警、整合資料來源
 指標收集	<b>Prometheus</b>	收集與查詢 Metrics (例如 CPU 使用率)
 日誌系統	<b>Loki</b>	類似 ELK 的日誌平台，專為 Kubernetes 打造
 追蹤系統	<b>Tempo</b>	收集分散式 Trace (例如每次 HTTP 請求的延遲)
 整合代理	<b>Alloy</b> (前身為 Agent)	收集 Metrics/Logs/Traces，送給上方元件

# Grafana

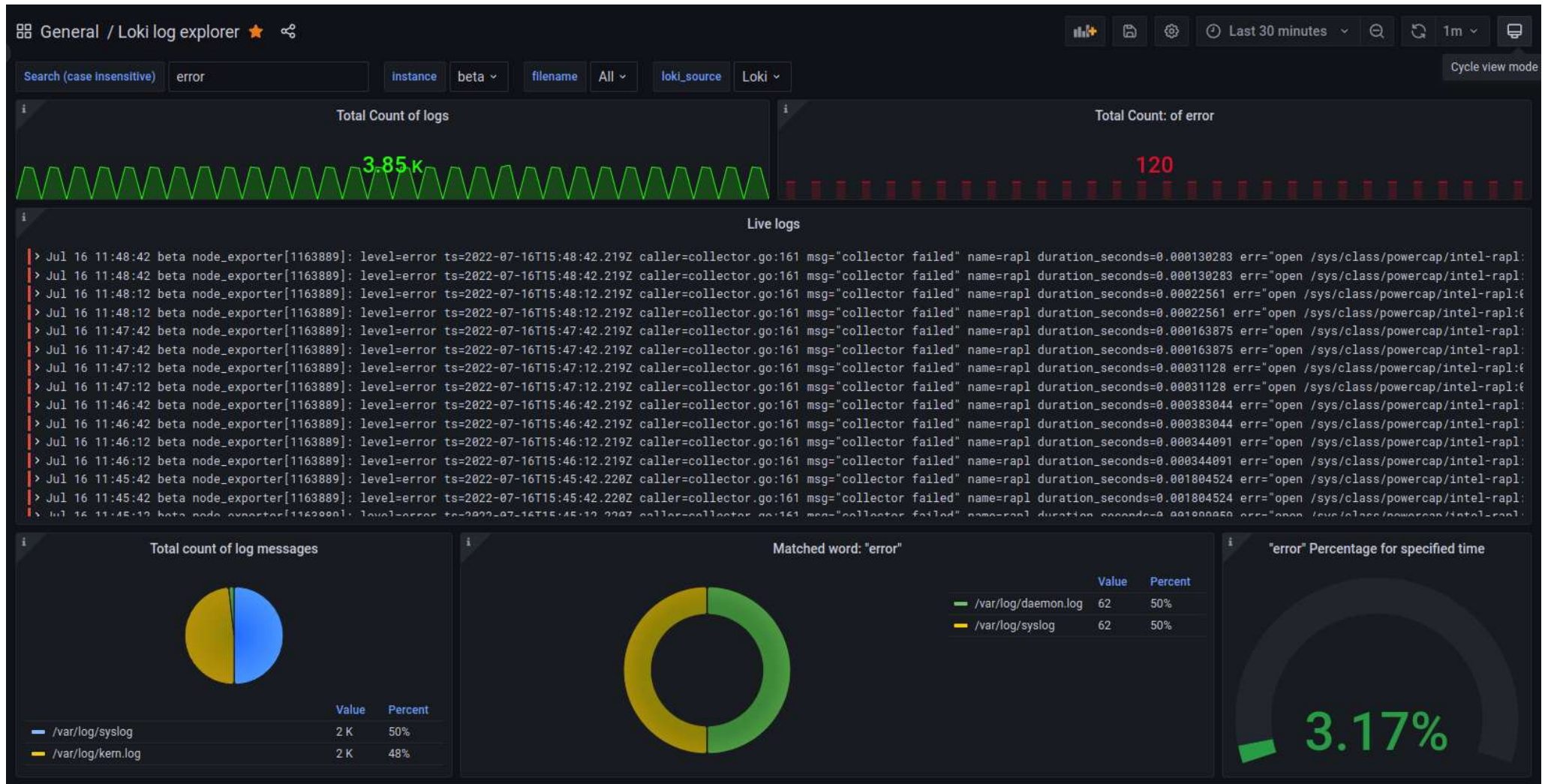


# Tempo





# Loki





# 應用部署與資源管理

HTIC



# 資源管理

HTIC



# Workload Management

在 Kubernetes 中，**工作負載 ( Workload ) 資源**是用來定義應用程式如何在叢集中執行與擴展的物件。以下是 **6 種主要的工作負載類型**，每種都有其特定用途：

## 1. Pod ( 容器組 )

- Kubernetes 中最小的可部署單位。
- 包含一個或多個共享儲存與網路的容器。
- 通常由更高階的控制器 ( 例如 Deployment 或 Job ) 來管理。

## 2. Deployment ( 部署 )

- 用來管理**無狀態應用程式**。
- 可確保指定數量的 Pod 副本持續運行。
- 支援**滾動更新與回滾**。

## 3. StatefulSet ( 有狀態組 )

- 用來管理**有狀態應用程式**。
- 每個 Pod 有**穩定的身份** ( DNS、儲存 )。
- 適用於需要**持久化儲存與啟動/停止順序**的應用。

## 4. DaemonSet ( 守護程序組 )

- 確保在**\*\*每個節點 ( 或選定節點 ) \*\***上都執行一個 Pod。
- 常用於執行像是日誌收集器或節點監控程式等背景守護程式。

## 5. Job ( 工作 )

- 執行 Pod 並在完成後結束 ( **一次性任務** )。
- 確保指定次數的成功執行。
- 適合執行批次任務或一次性作業。

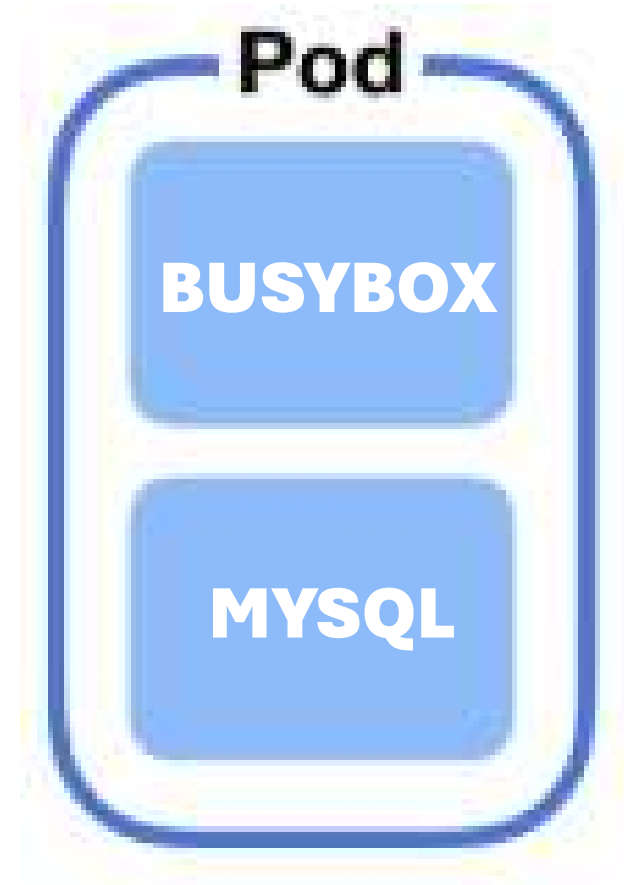
## 6. CronJob ( 排程工作 )

- 根據時間排程來執行**定期的 Job** ( 類似 Linux 的 cron )。
- 適合用於定期備份、報表產生等情境。

# Pod

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: nfs-test-pod
5  spec:
6    containers:
7    - name: nfs-test-container
8      image: busybox
9      command: [ "sh", "-c", "sleep 3600" ]
10     resources:
11       requests:
12         memory: "64Mi"
13         cpu: "250m"
14       limits:
15         memory: "128Mi"
16         cpu: "500m"
17     volumeMounts:
18     - name: nfs-volume
19       mountPath: /mnt/nfs
20   volumes:
21   - name: nfs-volume
22     persistentVolumeClaim:
23       claimName: keycloak-postgres-pvc
```

Pod 是最小的可部署單位，可以有一個或多個容器的封裝包



# Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

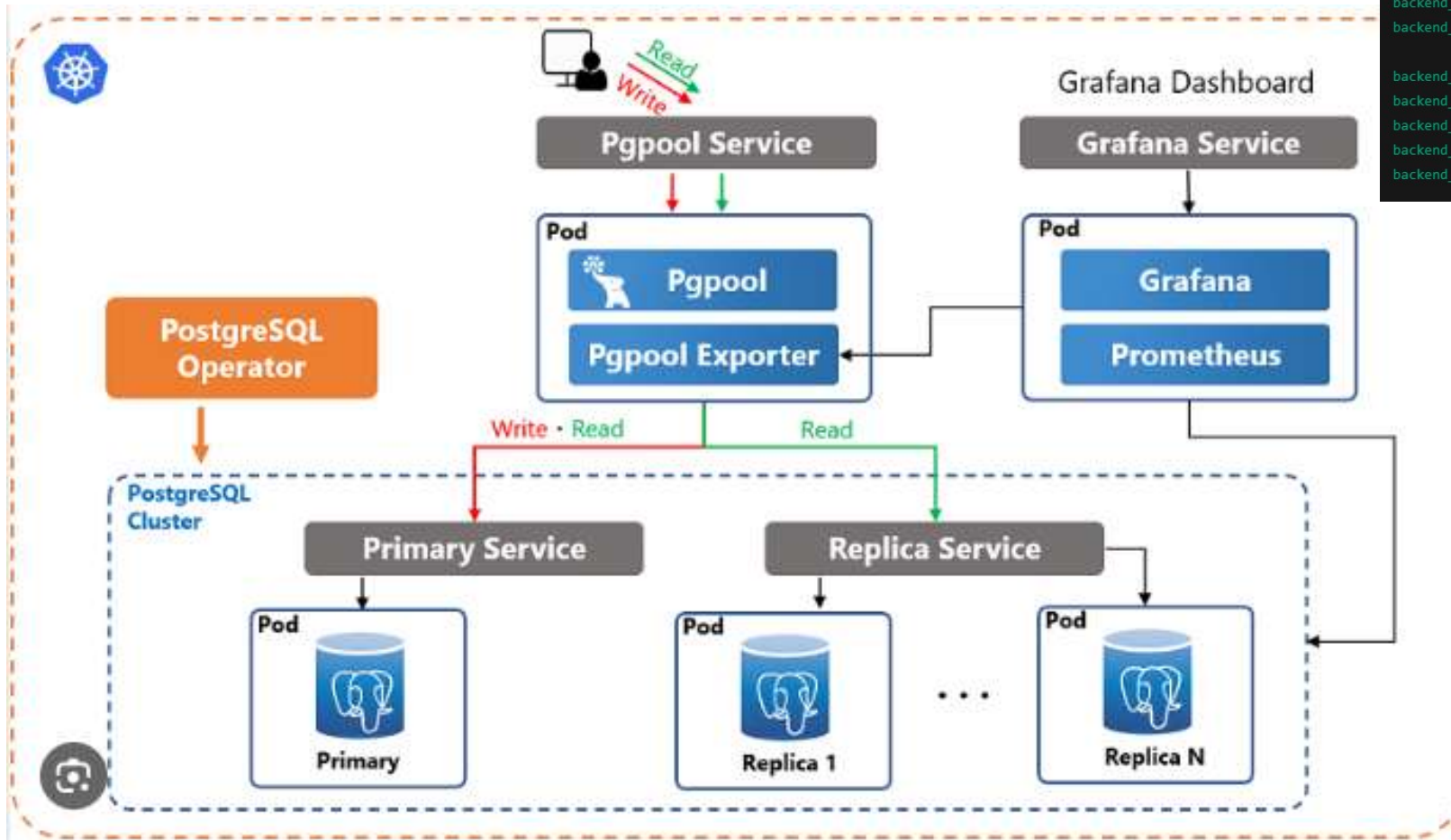
是一種控制器 ( Controller )，專門用來管理 無狀態應用 的部署、確保 Pod 的數量與版本狀態符合你定義的期望。



Pod隨時會被刪掉與產生

# StatefulSet

適用於需要「穩定網路識別、穩定存儲」以及「有順序的部署/更新」的應用



```
# Backend Nodes
backend_hostname0 = 'postgres-0.postgres.default.svc.cluster.local'
backend_port0 = 5432
backend_weight0 = 1
backend_data_directory0 = '/var/lib/postgresql/data'
backend_flag0 = 'PRIMARY'

backend_hostname1 = 'postgres-1.postgres.default.svc.cluster.local'
backend_port1 = 5432
backend_weight1 = 1
backend_data_directory1 = '/var/lib/postgresql/data'
backend_flag1 = 'STANDBY'

backend_hostname2 = 'postgres-2.postgres.default.svc.cluster.local'
backend_port2 = 5432
backend_weight2 = 1
backend_data_directory2 = '/var/lib/postgresql/data'
backend_flag2 = 'STANDBY'
```

Pgpool-II + postgresQL

# Autoscaling

```
1  apiVersion: autoscaling/v2
2  kind: HorizontalPodAutoscaler
3  metadata:
4    name: keycloak-hpa
5    namespace: keycloak
6  spec:
7    scaleTargetRef:
8      apiVersion: apps/v1
9      kind: Deployment
10     name: keycloak
11    minReplicas: 2
12    maxReplicas: 6
13    metrics:
14      - type: Resource
15        resource:
16          name: cpu
17          target:
18            type: Utilization
19            averageUtilization: 70
20      - type: Resource
21        resource:
22          name: memory
23          target:
24            type: Utilization
25            averageUtilization: 80
```



## Kubernetes Autoscaling 類型

類型	說明	使用資源
HPA (Horizontal Pod Autoscaler)	根據 CPU、記憶體或自訂指標，自動調整 Pod 數量	Pod
VPA (Vertical Pod Autoscaler)	根據實際使用情況，自動調整 Pod 的 CPU / Memory 限制與請求	Pod
Cluster Autoscaler	根據需要自動調整 Node 數量 (需配合雲端環境)	Node

### 水平擴展 Pods 數量 (HPA)

- 根據 CPU、記憶體使用率，或自訂指標 (如 QPS) 動態調整 replicas 數量。
- 常用於 Deployment、StatefulSet 等資源。



# 資源管理

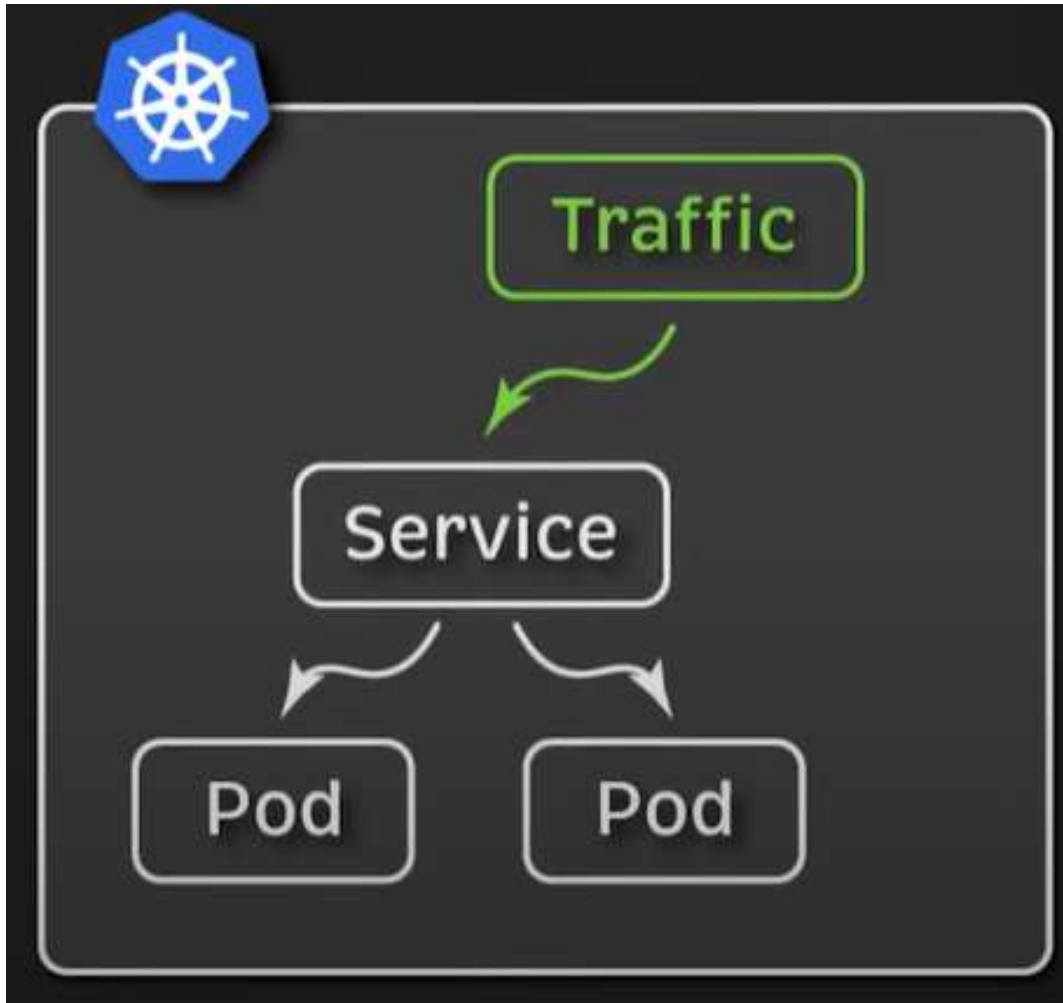
```
1  apiVersion: autoscaling/v2
2  kind: HorizontalPodAutoscaler
3  metadata:
4    name: keycloak-hpa
5    namespace: keycloak # 限制在keycloak這一個namespace
6  spec:
7    scaleTargetRef:
8      apiVersion: apps/v1
9      kind: Deployment
10     name: keycloak
11    minReplicas: 2 # 最少保留 2 個副本 (Pod)
12    maxReplicas: 6 # 最多擴展到 6 個副本 (Pod)
13    metrics:
14      - type: Resource
15        resource:
16          name: cpu
17          target:
18            type: Utilization # 當Pod的CPU平均使用率超過 70% 時，HPA 將考慮擴展
19            averageUtilization: 70 # 當低於該閾值時，可能會縮減副本數（不低於 minReplicas）
20      - type: Resource
21        resource:
22          name: memory
23          target:
24            type: Utilization
25            averageUtilization: 80 # 監控記憶體使用率，當平均超過 80% 就會觸發擴展行為
```

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: keycloak
5    namespace: keycloak
6  spec:
7    replicas: 2
8    selector:
9      matchLabels:
10       app.kubernetes.io/instance: keycloak
11    template:
12      metadata:
13        labels:
14         app.kubernetes.io/name: keycloak
15      spec:
16        containers:
17          - name: keycloak
18            image: quay.io/keycloak/keycloak:26.2.0
19            imagePullPolicy: IfNotPresent
20            args: ["start", "--health-enabled=true"]
21            resources:
22              requests:
23                cpu: 300m
24                memory: 1Gi
25              limits:
26                cpu: 1000m
27                memory: 2Gi
```

每個pod



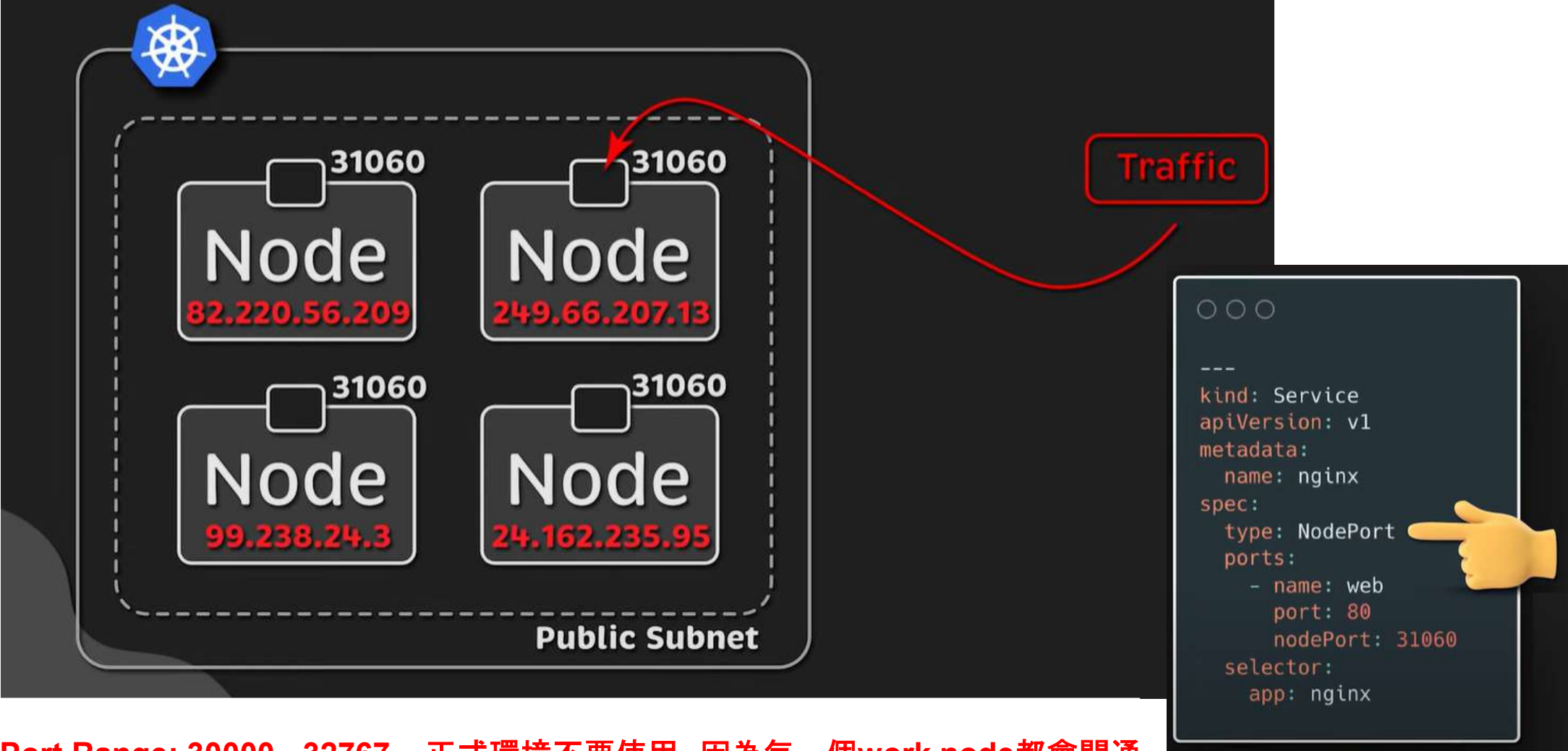
# ClusterIP



```
---
kind: Service
apiVersion: v1
metadata:
  name: nginx
spec:
  type: ClusterIP
  ports:
    - name: web
      port: 80
  selector:
    app: nginx
```

# NodePort

## NodePort



**Port Range: 30000 - 32767** 正式環境不要使用, 因為每一個work node都會開通

# Load Balancer

## Ingress



Ingress Controller

api.example.com

example.com/foo

other

Service

Service

Service

Pod

Pod

Pod

Pod

Pod

Pod

---

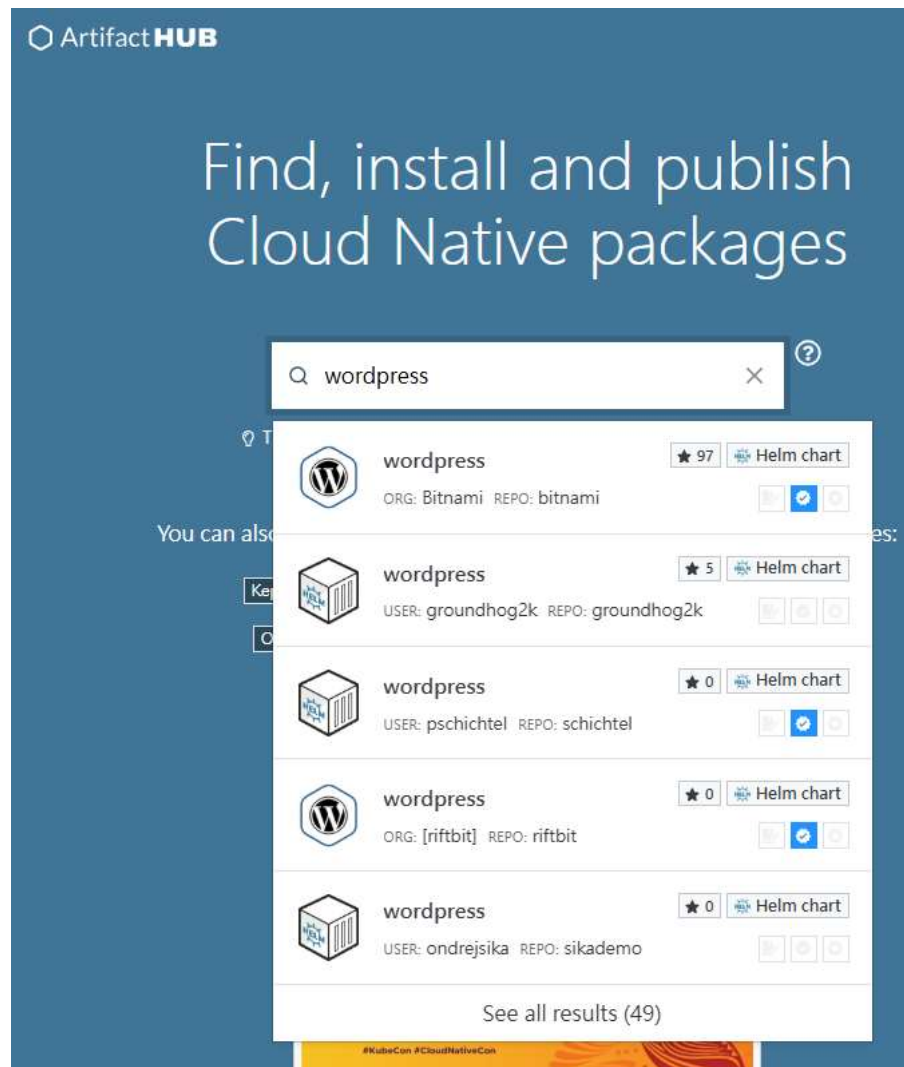
```
kind: Service
apiVersion: v1
metadata:
  name: api
spec:
  type: LoadBalancer
  ports:
    - name: http
      port: 8080
  selector:
    app: api
```



# 應用佈署

HTIC

# Helm



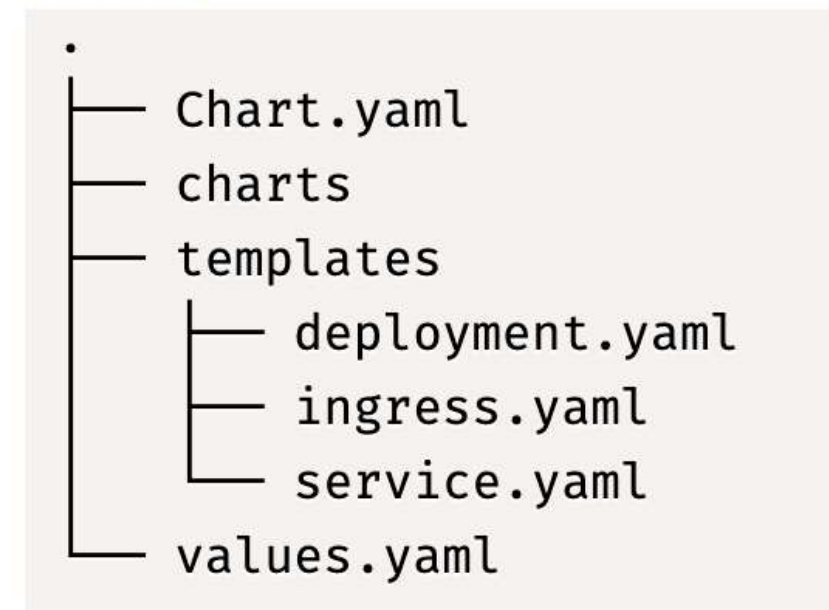
## 安裝套件

`helm install wordpress oci://xxx/wordpress`

## 反安裝套件

`helm uninstall wordpress`

## Helm打包功能



# 安裝與升級Kubernetes-dashboard

# 加 kubernetes-dashboard repository

helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/

# 佈署 a Helm Release named "kubernetes-dashboard" using the kubernetes-dashboard chart

helm upgrade --install kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard --create-namespace -  
-namespace kubernetes-dashboard

```
• sysadm1@k8s-master-01:~/K8SConfig/Dashboard$ helm list -A
```

NAME	NAMESPACE	REVISION	UPDATED	STATUS	CHART	APP VERSION
csi-driver-nfs	kube-system	1	2025-05-25 06:42:04.814329661 +0800 CST	deployed	csi-driver-nfs-4.11.0	4.11.0
ingress-nginx	ingress-nginx	1	2025-05-18 09:57:57.673223141 +0800 CST	deployed	ingress-nginx-4.12.2	1.12.2
istio-base	istio-system	1	2025-07-18 21:39:27.819750168 +0800 CST	deployed	base-1.26.2	1.26.2
istio-ingress	istio-system	1	2025-07-18 21:52:22.781644975 +0800 CST	deployed	gateway-1.26.2	1.26.2
istiod	istio-system	1	2025-07-18 21:42:48.204098513 +0800 CST	deployed	istiod-1.26.2	1.26.2
kubernetes-dashboard	kubernetes-dashboard	2	2025-05-26 14:55:24.951650507 +0800 CST	deployed	kubernetes-dashboard-7.12.0	

```
• sysadm1@k8s-master-01:~/K8SConfig/Dashboard$ helm search repo kubernetes-dashboard
```

NAME	CHART VERSION	APP VERSION	DESCRIPTION
kubernetes-dashboard/kubernetes-dashboard	7.13.0		General-purpose web UI for Kubernetes clusters

#升級

helm upgrade --install kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard --namespace kubernetes-  
dashboard

```
• sysadm1@k8s-master-01:~/K8SConfig/Dashboard$ helm list -A
```

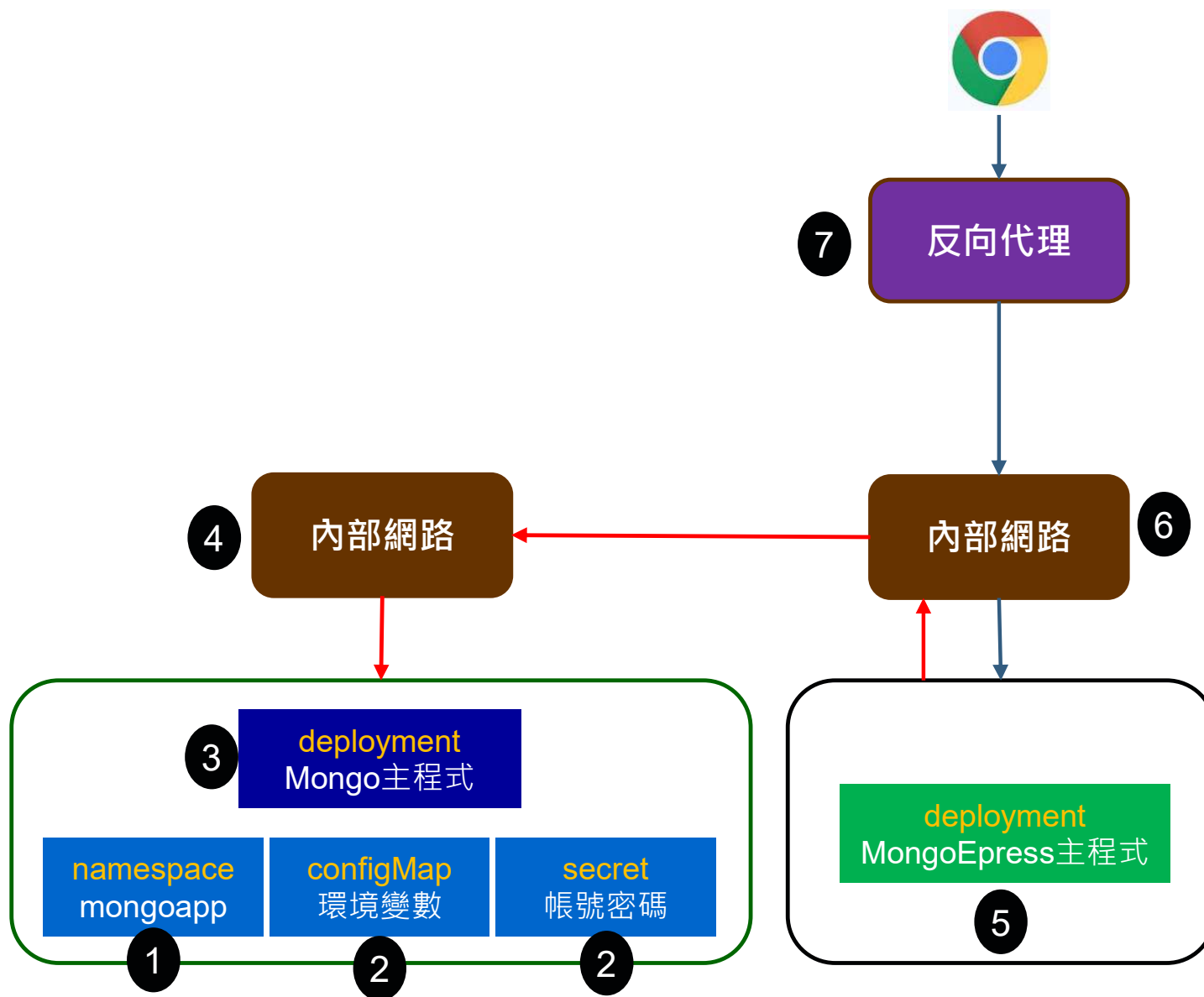
NAME	NAMESPACE	REVISION	UPDATED	STATUS	CHART	APP VERSION
csi-driver-nfs	kube-system	1	2025-05-25 06:42:04.814329661 +0800 CST	deployed	csi-driver-nfs-4.11.0	4.11.0
ingress-nginx	ingress-nginx	1	2025-05-18 09:57:57.673223141 +0800 CST	deployed	ingress-nginx-4.12.2	1.12.2
istio-base	istio-system	1	2025-07-18 21:39:27.819750168 +0800 CST	deployed	base-1.26.2	1.26.2
istio-ingress	istio-system	1	2025-07-18 21:52:22.781644975 +0800 CST	deployed	gateway-1.26.2	1.26.2
istiod	istio-system	1	2025-07-18 21:42:48.204098513 +0800 CST	deployed	istiod-1.26.2	1.26.2
kubernetes-dashboard	kubernetes-dashboard	3	2025-07-20 08:12:52.470565177 +0800 CST	deployed	kubernetes-dashboard-7.13.0	



# Kubectl

```
1  kubectl get nodes           # 查詢節點
2  kubectl get pods           # 查詢所有 Pod (預設 namespace)
3  kubectl get pods -n <namespace> # 查詢指定 namespace 的 Pod
4  kubectl get svc            # 查詢所有 Service
5  kubectl get deployments    # 查詢 Deployments
6  kubectl get all            # 查詢所有資源 (Pod, Service, Deployment 等)
7  kubectl get all -A         # 查詢所有資源所有的namespace
8  kubectl describe pod <pod-name> # 查看 Pod 詳細資訊
9  kubectl logs <pod-name>     # 查看 Pod log (僅限單容器)
10 kubectl apply -f <file>.yaml # 使用 YAML 建立或更新資源
11 kubectl apply -f .         # 套用這目錄夾底下所有YAML定義
12 kubectl delete -f <file>.yaml # 刪除 YAML 中定義的資源
13 kubectl delete -f .       # 刪除這目錄夾底下所有YAML定義
14 kubectl delete pod <pod-name> # 刪除單一 Pod
15 kubectl delete svc <svc-name> # 刪除 Service
16 kubectl exec -it <pod-name> -- /bin/bash # 進入 Pod 中 (需 bash/sh)
17 kubectl port-forward <pod-name> 8080:80 # 本地端口轉發到 Pod
18 kubectl top pod           # 顯示 Pod 資源用量 (需 metrics-server)
19 kubectl get namespaces    # 查看所有命名空間
20 kubectl create namespace <name> # 建立命名空間
21 kubectl delete namespace <name> # 刪除命名空間
22 kubectl config set-context --current --namespace=<name> # 切換預設 namespace
23 kubectl get pods -o wide  # 顯示更多欄位資訊
24 kubectl get pods -o yaml  # 顯示這一個pod的YAML定義
25 kubectl get pods -l app=nginx # 根據 label 過濾
```

## 展示佈署MONGO + MONGOEXPRESS服務

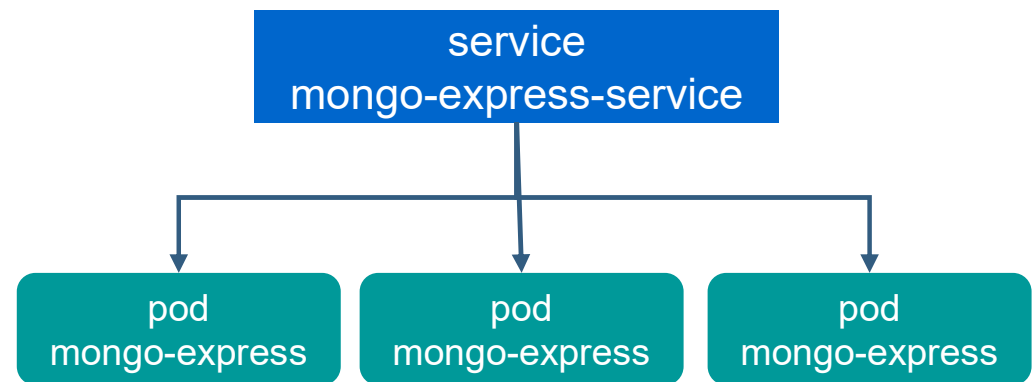


```
! 01_mongodb_namesapce.yaml
! 02_mongodb_configmap.yaml
! 02_mongodb_secret.yaml
! 03_mongodb_deployment.yaml
! 04_mongodb_service.yaml
! 05_mongoexpress_deployment.yaml
! 06_mongoexpress_service.yaml
! 07_mongoexpress_ingress.yaml
```

# Deployment + Service

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: mongo-express
5    namespace: mongoapp
6  spec:
7    replicas: 3
8    selector:
9      matchLabels:
10       app: mongo-express
11  template:
12    metadata:
13      labels:
14       app: mongo-express
15    spec:
16      containers:
17      - name: mongo-express
18        image: mongo-express
19        ports:
20        - containerPort: 8081
21      resources:
22        requests:
23          memory: "128Mi"
24          cpu: "100m"
25        limits:
26          memory: "256Mi"
27          cpu: "250m"
```

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: mongo-express-service
5    namespace: mongoapp
6  spec:
7    type: LoadBalancer
8    selector:
9      app: mongo-express
10   ports:
11   - protocol: TCP
12     port: 8080
13     targetPort: 8081
```





# 安全性與權限控管

HTIC

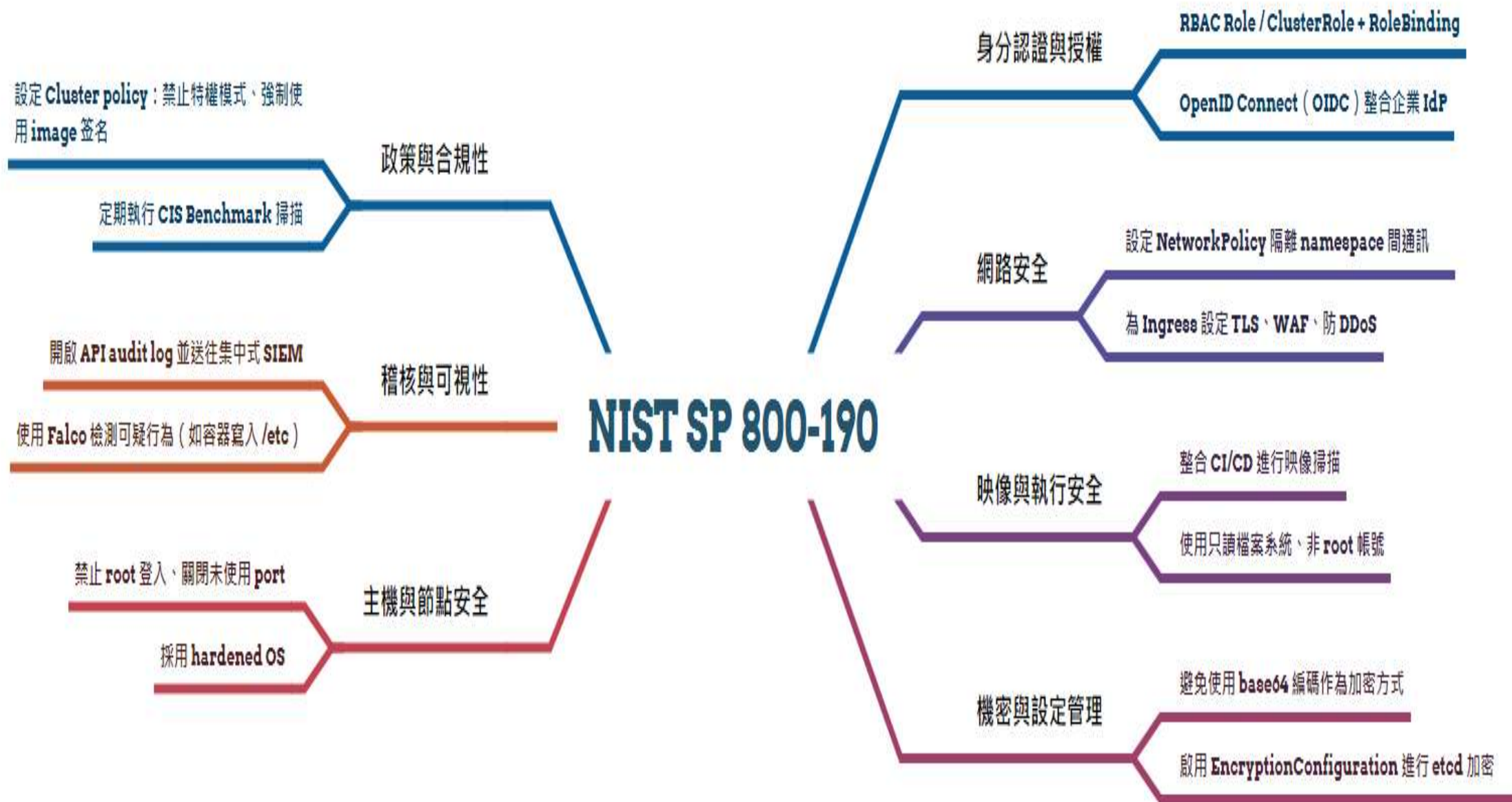
# NIST SP 800-190

這份文件的主要目標是提供政府機關、企業與開發人員在採用應用程式容器技術時，能夠建立一套有效的 安全實務建議與風險控管方法。

**NIST 將容器安全分成以下七個層面，並針對每一層提供威脅分析與防禦建議：**


元件	說明
<b>Container Image</b>	映像檔可能含有漏洞或惡意程式碼
<b>Registry</b>	儲存映像的登錄中心，可能遭竄改或未授權訪問
<b>Orchestrator</b>	如 Kubernetes，可能遭濫用進行未授權操作
<b>Container Runtime</b>	執行容器的環境，需防止逃逸或升權
<b>Host OS</b>	容器共用的主機系統，為高風險目標
<b>Hardware</b>	底層硬體或虛擬化平台，需確保信任基礎
<b>Networking</b>	容器間通訊可能造成資訊洩漏







# 企業版的K8S for NIST SP 800-190

 **Red Hat**

AI ▾ Hybrid cloud ▾ Products ▾ Training ▾ Learn ▾ Partners ▾

**Table of contents**

[Understanding the threat landscape for container usage](#)

[Red Hat OpenShift Platform Plus](#)

[Kubernetes security at a glance](#)

[How OpenShift Platform Plus supports NIST SP 800-190](#)

[Summary](#)

[Further reading: Implementing Kubernetes-native security with Red Hat](#)

[Home](#) > [Resources](#) > [Guide to NIST SP 800-190 compliance in co...](#)

## Guide to NIST SP 800-190 compliance in container environments

May 22, 2024 • Resource type: Detail

### Understanding the threat landscape for container usage

Organizations are eagerly adopting containers, Kubernetes, and microservices, but security concerns can be an impediment, as our latest [State of Kubernetes Security Report](#)<sup>1</sup> reveals. While many are embracing containers in their organizations, some are still learning about containers and Kubernetes themselves—understanding that the security implications of this infrastructure add to the learning curve. It is critical to understand the threat landscape in these environments, assess your risk posture when using containers, and mitigate the security risks associated with container adoption.

 **BROADCOM**

Products Solutions Support and Services Company How To Buy

[Home](#) / [VMware Tanzu Software](#) / [Standalone Components](#) / [Tanzu Kubernetes Grid](#) / [VMware Tanzu Kubernetes Grid v2.5.x](#)

## Tanzu Kubernetes Grid 2.5

 Version 2.5 ▾

 **Topics**

**Tanzu Kubernetes Grid 2.5**

TKG 2.5.x Release Notes

About Tanzu Kubernetes Grid

Deploying and Managing Tanzu Kubernetes Grid

—

### VMware Tanzu Kubernetes Grid v2.5.x

*Last Updated June 26, 2025*

VMware Tanzu Kubernetes Grid provides a consistent, upstream-compatible Kubernetes end-user workloads and ecosystem integrations. Tanzu Kubernetes Grid (TKG) provides vSphere.



CI/CD

HTIC

# GitOps

GitOps 是一種 基於 Git 儲存庫作為唯一事實來源 ( Single Source of Truth ) 的軟體交付與基礎設施管理方法，核心理念是 用 Git 版本控制基礎設施與應用程式設定，並透過自動化流程讓實際環境自動與 Git 內容保持一致。

## 為何使用GitOps

- ✓ 可追溯性：所有改動有清楚的版本紀錄。
- ✓ 一致性：避免「手動改配置」錯誤。
- ✓ 自動化：減少人工部署錯誤。
- ✓ 回復快速：出錯時直接回到上一個 commit。



# Continuous Integration (持續整合)

## CI：持續整合

### 概念

- 每當開發人員將程式碼提交 ( commit ) 到版本控制系統 ( 如 Git ) 時，系統會自動進行**建置** ( Build )、**測試** ( Test )，確保新程式碼能與現有程式碼整合而不破壞功能。

### 重點

- 提早發現錯誤
- 自動化測試與驗證
- 保持主分支穩定
- 減少「整合地獄」 ( Integration Hell )

### 常用工具

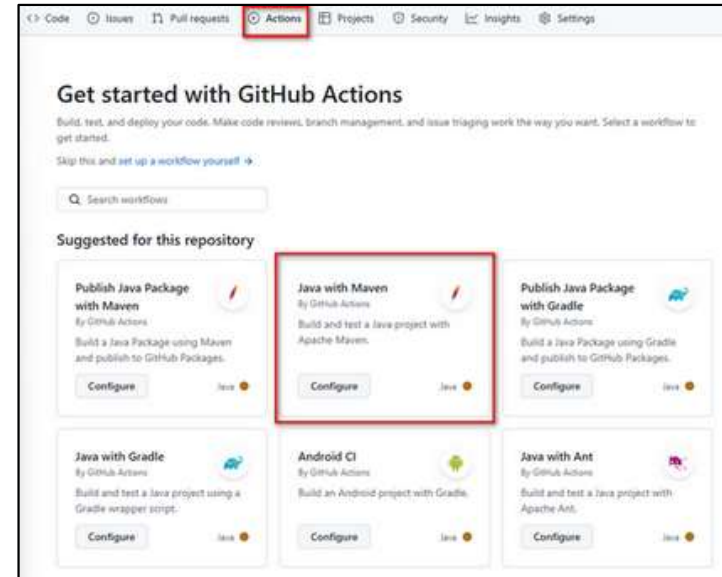
- Jenkins、GitHub Actions、GitLab CI、CircleCI、Azure DevOps Pipelines



```
• Merge branch 'feature/chan... main
• update data value tichung75
• modify cd-cd.yaml add gitops tichung75
• update cd-cd.yaml tichung75
• update ci-cd.yaml tichung75
• modify script for updating gitops tichung75
• update dockerfile tichung75
• update tichung75
• Merge branch 'main' of https://github.co...
• Delete .github/workflows/maven-publish....
• Create maven-publish.yml meatfootman
• update maven action tichung75
• secret name change tichung75
• update ci-cd.yml tichung75
• Merge branch 'feature/dockerfile' tichung75
• add Dockerfile tichung75
• Create ci-cd.yml meatfootman
• http get ready tichung75
• initial commit tichung75
```

Git/GitHub

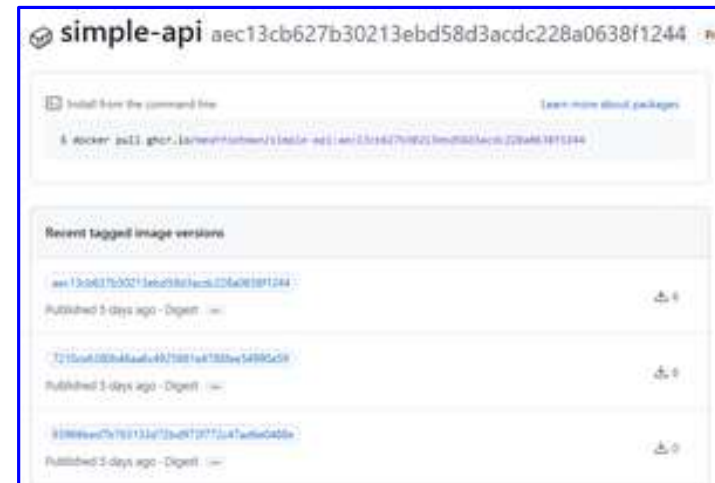
pipeline workflow



docker container



```
12 @RestController
13 public class InfoController {
14     @GetMapping("/")
15     public Map<String, String> getInfo() {
16         Map<String, String> info = new HashMap<>();
17         try {
18             InetAddress inetAddress = InetAddress.getLocalHost();
19             info.put(key:"hostname", inetAddress.getHostName());
20             info.put(key:"ip", inetAddress.getHostAddress());
21             info.put(key:"data", value:"固定碼2");
22         } catch (UnknownHostException e) {
23             info.put(key:"hostname", value:"Unknown");
24             info.put(key:"ip", value:"Unknown");
25         }
26         //DateTimeFormatter formatter = DateTimeFormatter.ofPattern
27         //info.put("time", LocalDateTime.now().format(formatter));
28         info.put(key:"time", LocalDateTime.now().toString());
29         return info;
30     }
31 }
```



# Continuous Delivery / Deployment (持續交付 / 部署)

CD：持續交付 / 持續部署

CD 有兩種常見解釋，差異在於最後一步是否自動部署到生產環境。

## 2.1 持續交付 (Continuous Delivery)

- 程式碼經過 CI 驗證後，自動建置到類生產環境 (Staging) 並準備好部署，但需要人工批准才會推到生產環境。
- 好處：可控、降低風險，適合對營運穩定性要求高的系統。

## 2.2 持續部署 (Continuous Deployment)

- 程式碼經過 CI 驗證後，自動部署到生產環境，無需人工批准。
- 好處：最快速度釋出功能，適合頻繁更新、快速迭代的系統。

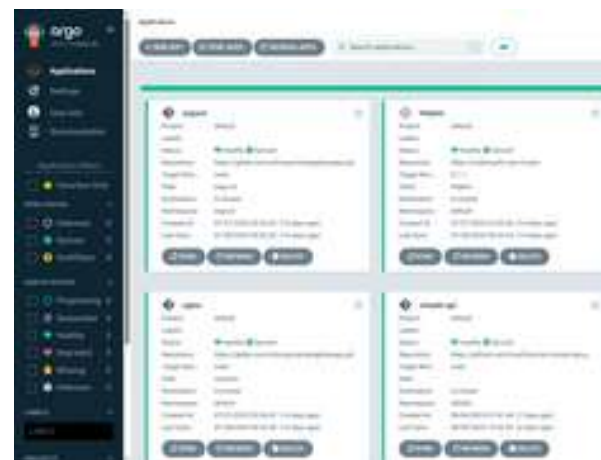
### 常用工具

- ArgoCD、Spinnaker、FluxCD、Jenkins、GitLab CI/CD

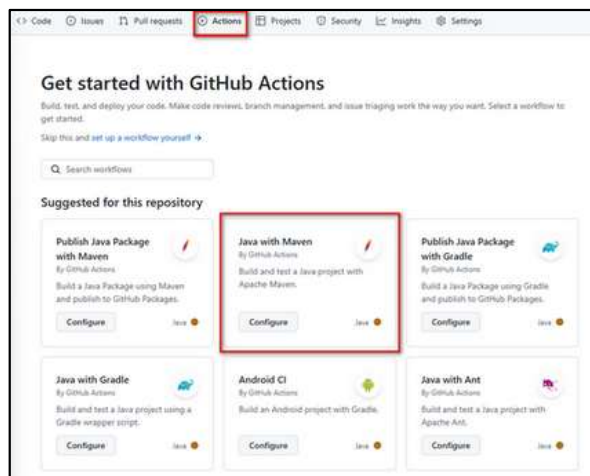




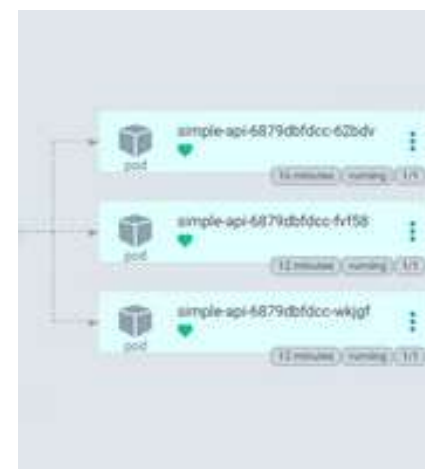
定時確認  
deployment



GitHub Action  
修改deployment



產生Pods





# 開源軟件介紹

HTIC

## 知名開源工具 for Kubernetes



**SSO**  
**OAuth/OIDC Token**



**CI/CD**  
持續交付



**Secret**  
機密資料集中



**Reload** 程式  
異動設定檔重起程式



**API Gateway**  
**API**管控



合規檢測  
安全掃描、漏洞檢測與  
合規性驗證



**Monitoring**  
監控管控

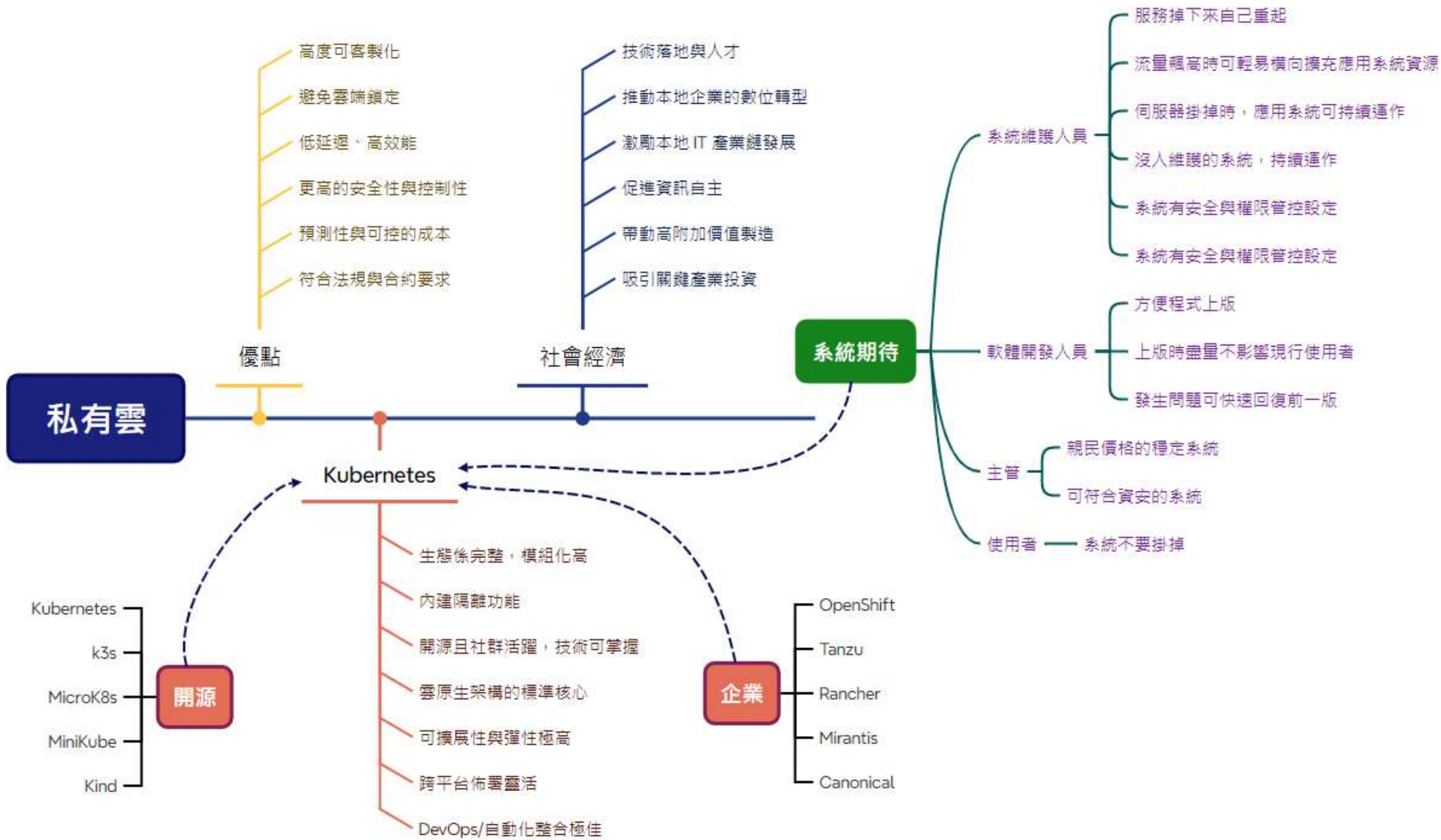


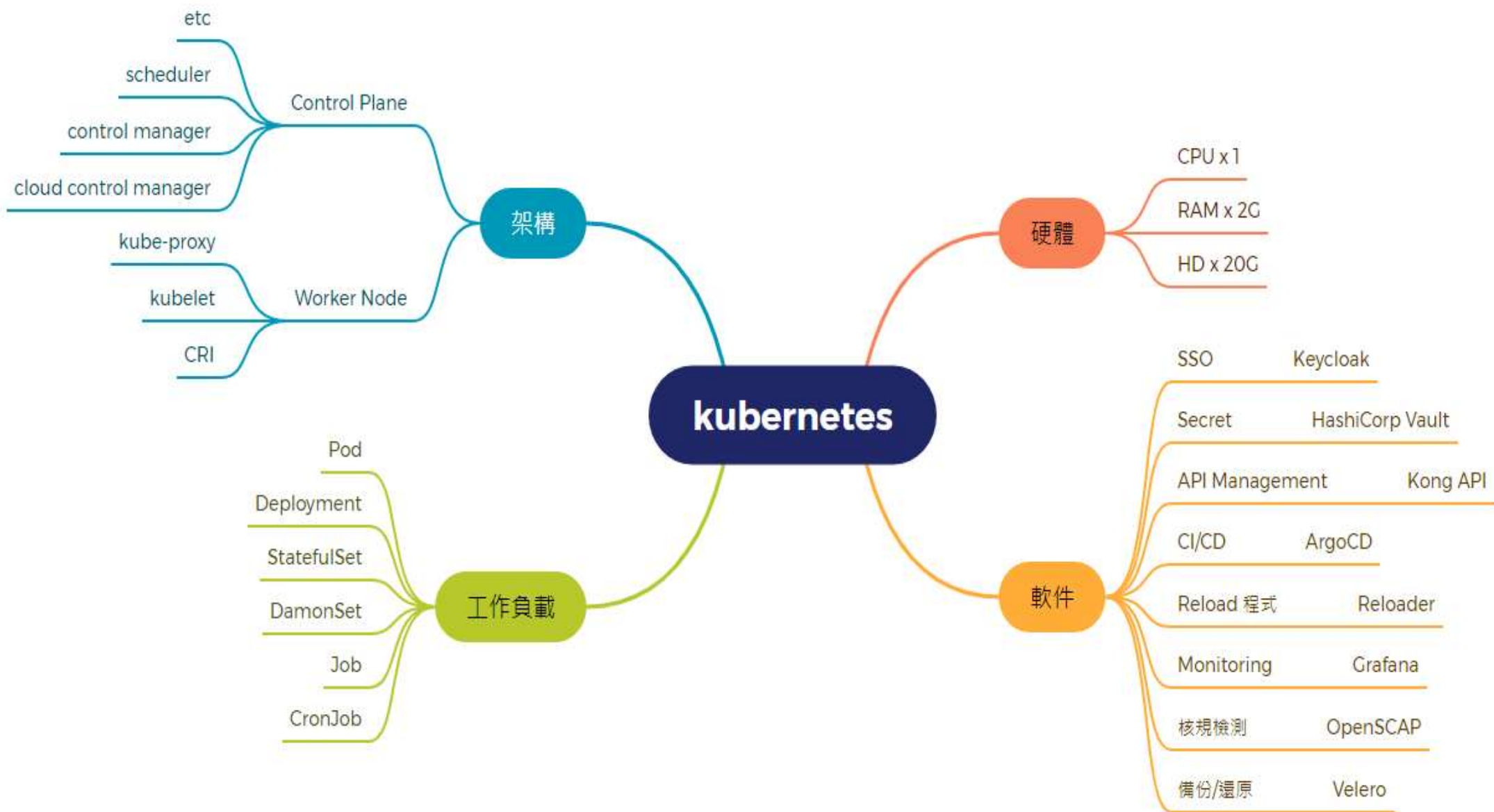
備份與還原  
開源備份還原工具



# 總結

HTIC







## 私有雲企業平台推薦



- 以 K8s 為核心架構所建構
- 強大 GUI 管理界面
- 安裝後即具備完整功能
- Quota 控管機制內建
- 預設啟用強安全措施
- 內建 OpenShift Pipelines
- 可管控傳統VM
- Red Hat 支援



**IBM Flash System**

KUBERNETES



# THANK YOU

如果給我6個小時砍下一顆樹，我會用前面  
4個小時把斧頭磨利。

Give me six hours to chop down a tree and I will  
spend the first four sharpening the axe.

Abraham Lincoln

鉅迪資訊