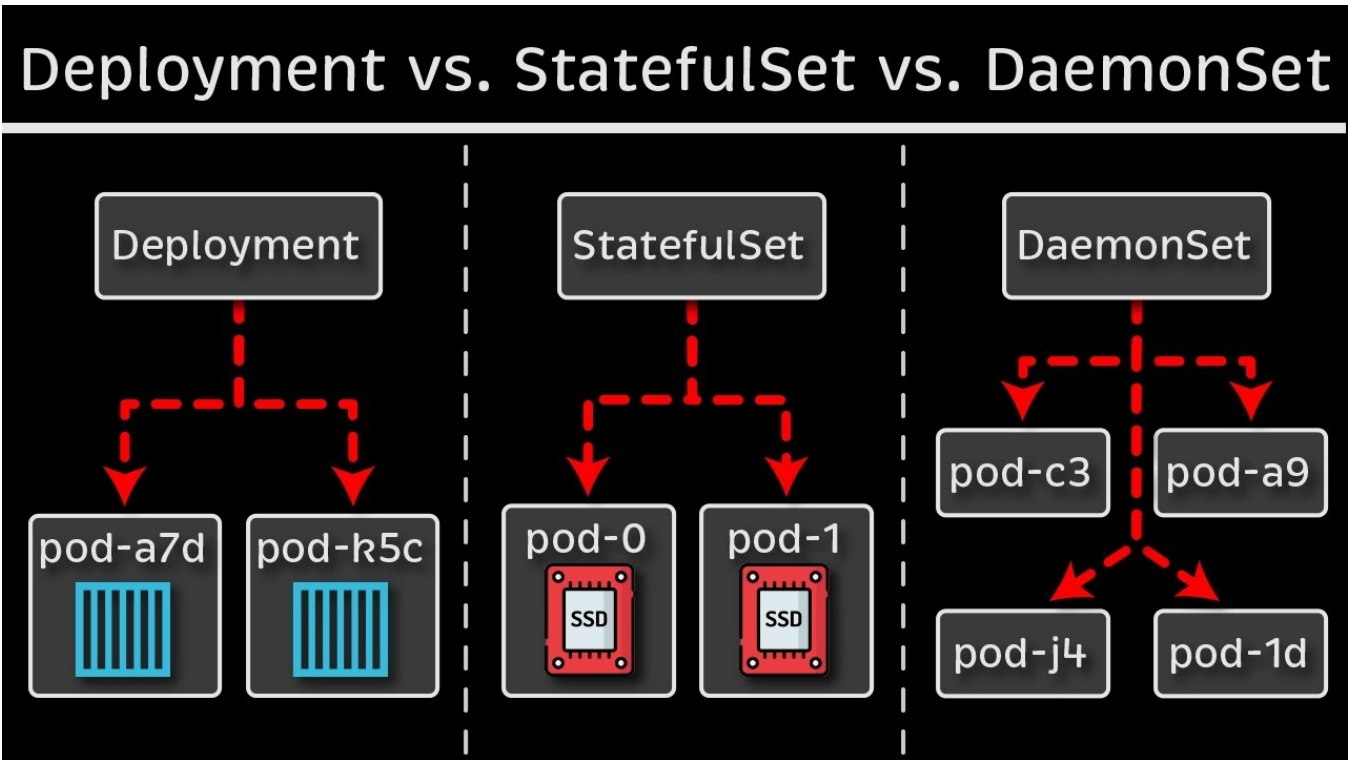
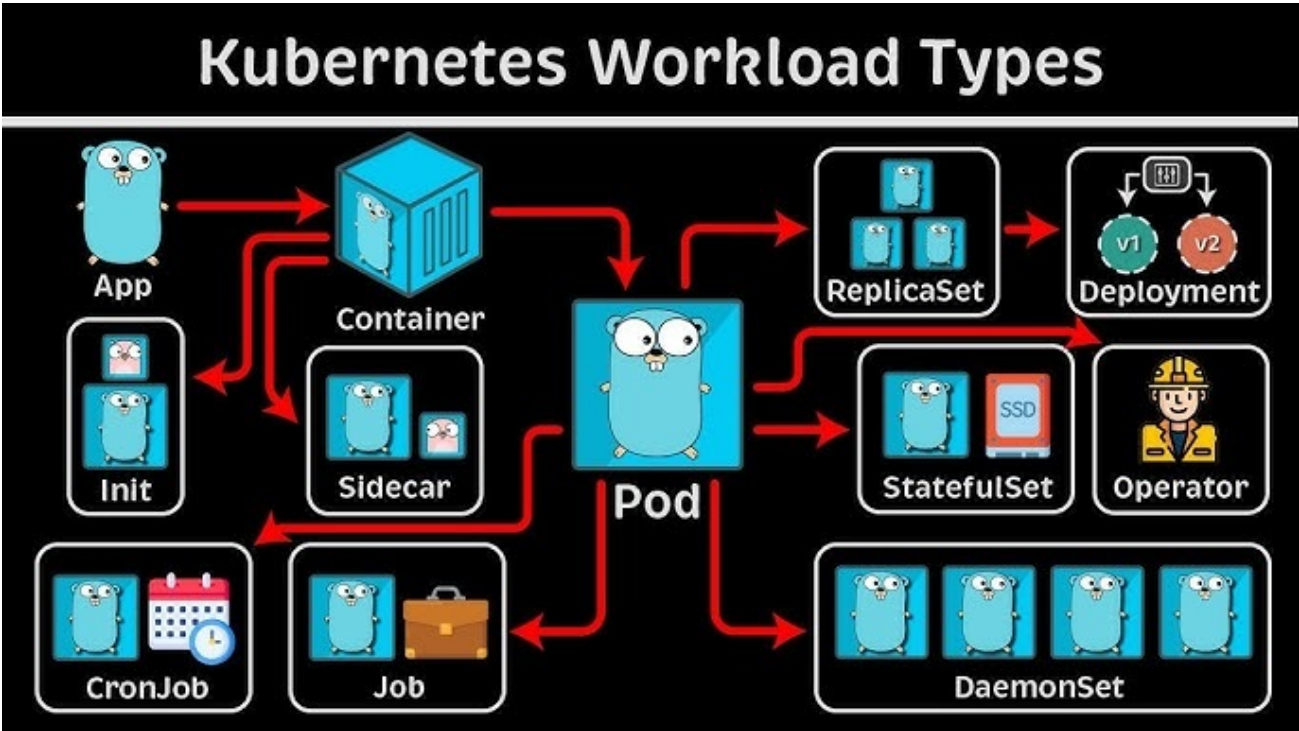


# Kind 簡易說明

## K8s Workload



以下是 **DaemonSet**、**StatefulSet**、**ReplicaSet** 和 **Deployment** 的詳細比較：

特性	DaemonSet	StatefulSet	ReplicaSet	Deployment
用途	確保每個 Node 上運行一個 Pod 副本。	部署有狀態的應用，每個 Pod 都有固定標識。	確保指定數量的 Pod 副本運行。	管理 Pod 和 ReplicaSet，提供滾動更新功能。
典型場景	日誌收集器、監控代理等需要全節點覆蓋。	資料庫、分佈式系統等需要穩定標識或存儲的應用。	確保無狀態應用的高可用性。	部署和管理無狀態應用，支持版本控制和更新。

特性	DaemonSet	StatefulSet	ReplicaSet	Deployment
Pod 名稱	名稱可變，與 Node 關聯，無固定格式。	有固定名稱（如 db-0, db-1）。	無固定格式，與 Deployment 類似。	通常由 Deployment 自動生成，不直接管理 Pod。
存儲	不涉及持久化存儲，通常為臨時存儲。	每個 Pod 可關聯專屬持久存儲。	通常無需存儲，支持臨時存儲。	通常無需存儲，支持無狀態應用。
縮放（副本數）	無法指定副本數，與 Node 數量相關。	支持指定副本數，且保證順序和標識穩定。	支持指定副本數，所有副本平等。	支持指定副本數，自動調整和滾動更新。
更新方式	滾動更新，逐一更新節點上的 Pod。	滾動更新，有序處理 Pod，確保依賴順序。	滾動更新，副本數量保持穩定。	滾動更新，支持回滾和藍綠部署策略。
依賴的資源	與 Node 關聯，不依賴其他資源。	通常依賴持久化存儲（如 PVC）。	通常不依賴其他資源。	自動創建和管理 ReplicaSet。
伸縮性	固定與 Node 數量匹配，無法手動調整。	可靈活擴展，並保證 Pod 的穩定標識。	手動指定副本數，無狀態，副本之間無區別。	自動管理副本數，支持水平和垂直擴展。
依賴順序	Pod 啟動順序無依賴。	Pod 啟動和刪除有順序，確保依賴關係。	無順序依賴，隨機啟動和刪除。	無順序依賴，通常關注應用級別的管理。
高可用性	和 Node 數量相關，通常與節點數一致。	通過固定標識和穩定的存儲保證高可用性。	通過多副本保證高可用性，但不管理應用版本。	提供高可用性並支持應用升級與版本回滾。

## 適用場景總結

- DaemonSet：**
  - 用於需要在所有 Node 上部署的服務，例如：
    - 日誌收集器（如 Fluentd）。
    - 監控代理（如 Prometheus Node Exporter）。
    - 網絡插件（如 Calico, Weave）。
- StatefulSet：**
  - 適用於需要穩定標識和存儲的有狀態應用，例如：
    - 資料庫（如 MySQL, PostgreSQL）。
    - 分布式系統（如 Kafka, ZooKeeper）。
    - 需要保持 Pod 順序的應用。
- ReplicaSet：**
  - 用於確保固定數量的無狀態 Pod 副本運行，例如：
    - 部署應用的核心部分，但一般由 Deployment 自動管理。
- Deployment：**
  - 最常用於管理無狀態應用，支持滾動更新、版本控制和自動擴展，例如：
    - Web 應用（如 Nginx, Node.js）。
    - 微服務應用的部署。

這樣的比較有助於釐清這些資源的功能和適用情境，幫助在實際項目中選擇合適的解決方案。

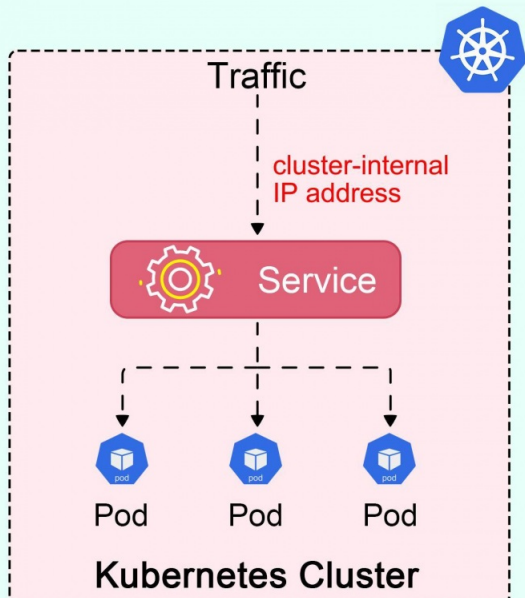
在 Kubernetes 中，`Service` 的 `type` 定義了服務的訪問方式和範圍。以下是 Kubernetes 支持的四種類型及其用途說明：

## Service 的四種Type

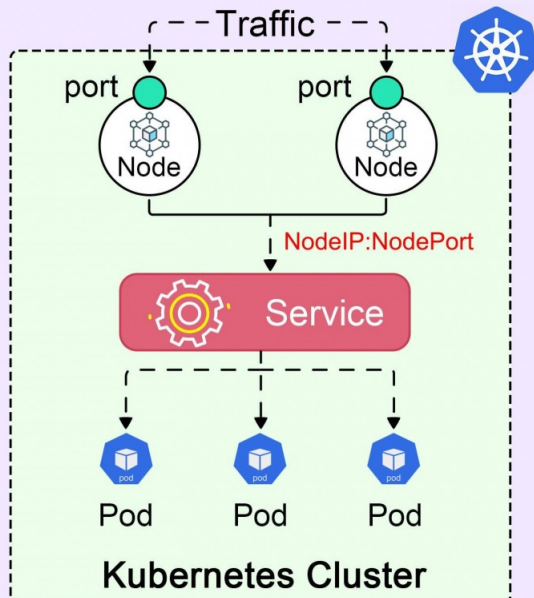
# 4 K8S Service Types



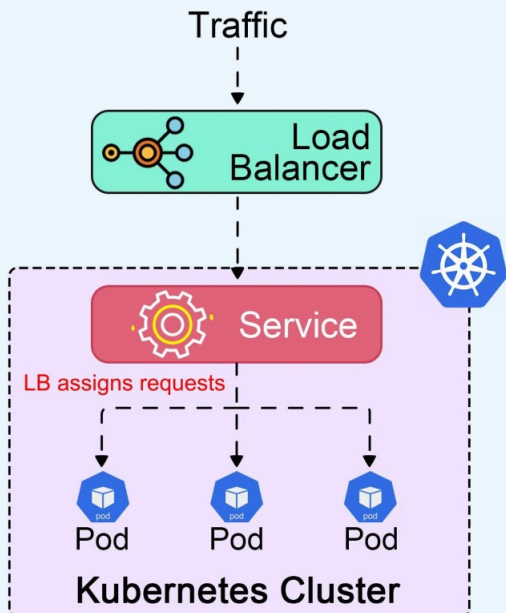
## ClusterIP



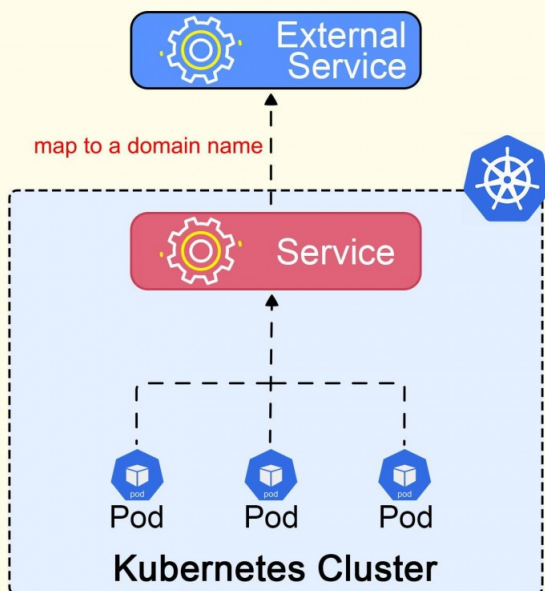
## NodePort



## LoadBalancer



## ExternalName



## 1. ClusterIP (預設類型)

- 說明：
  - 僅限於集群內部的虛擬 IP，用於服務發現和內部通信。
  - 外部無法直接訪問。
- 用途：
  - 用於內部微服務之間的通信。
  - 通常搭配 DNS (如 `kube-dns`) 進行名稱解析。
- 範例：

```
apiVersion: v1
kind: Service
metadata:
  name: clusterip-service
spec:
  type: ClusterIP
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

## 2. NodePort

- **說明：**
  - 將服務暴露在每個節點的固定端口上（範圍：30000-32767）。
  - 用戶可以通過 `<NodeIP>:<NodePort>` 訪問服務。
- **用途：**
  - 測試環境中，用於將服務暴露給集群外部的用戶。
  - 生產環境中通常搭配 **Ingress** 或 **LoadBalancer**。
- **範例：**

```
apiVersion: v1
kind: Service
metadata:
  name: nodeport-service
spec:
  type: NodePort
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
      nodePort: 30007
```

## 3. LoadBalancer

- **說明：**
  - 在雲提供商（如 GCP、AWS 或 Azure）中創建外部負載均衡器，並將流量轉發到 Service。
  - 每個 LoadBalancer 類型的 Service 都會自動獲得一個外部 IP。
- **用途：**
  - 用於公開服務給互聯網，用於生產環境。
  - 支持流量負載均衡。
- **範例：**

```
apiVersion: v1
kind: Service
metadata:
  name: loadbalancer-service
spec:
  type: LoadBalancer
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

## 4. ExternalName

- **說明：**

- 將 Service 名稱映射到外部 DNS 名稱。
- 不會創建 ClusterIP，也無法進行內部流量轉發。
- **用途：**
  - 將 Kubernetes 內部服務請求代理到集群外部的服務。
- **範例：**

```
apiVersion: v1
kind: Service
metadata:
  name: externalname-service
spec:
  type: ExternalName
  externalName: external.example.com
```

## 服務類型對比

類型	是否對外暴露	訪問方式	典型使用場景
ClusterIP	否	集群內部通信，使用虛擬 IP	微服務之間的內部通信。
NodePort	是	<NodeIP>:<NodePort>	測試環境或簡單的外部訪問。
LoadBalancer	是	自動分配外部 IP	公開服務到互聯網，支持負載均衡。
ExternalName	是（間接）	使用外部 DNS 名稱解析	將內部服務請求代理到集群外部服務。

## 適用場景建議

1. **ClusterIP**：適合內部微服務的互相調用，默認類型。
2. **NodePort**：適合小型測試環境，簡單公開服務。
3. **LoadBalancer**：適合生產環境，需要高可用的負載均衡服務。
4. **ExternalName**：適合需要訪問外部資源但想使用 Kubernetes 內部 DNS 管理的情況。

這些 `type` 結合不同的網絡需求，靈活應對各種應用場景。

## YML 的各種 Kind

Kind	說明	用途
Pod	Kubernetes 中最小的執行單位，包含容器。	部署單個應用程式容器。
Job	執行一次性任務，確保成功完成。	運行批處理任務。
CronJob	定期執行的任務。	排程任務（如每天備份數據）。
StatefulSet	管理有狀態的應用。	部署需固定標識或穩定存儲的 Pod。
ReplicaSet	確保指定數量的 Pod 始終運行。	通常由 Deployment 管理，用於維持 Pod 副本數。
DaemonSet	確保每個 Node 上運行一個 Pod 副本。	部署監控代理、日誌收集器等。
Deployment	管理 Pod 和 ReplicaSet，支持滾動更新。	部署無狀態應用，保持指定數量的 Pod 運行。
Service	定義 Pod 的網絡訪問規則。	將外部流量路由到 Pod，提供負載均衡。
Ingress	提供 HTTP/HTTPS 路由規則。	公開訪問服務，支持負載均衡和基於域名的路由。
NodePort	將外部流量映射到指定節點端口。	將集群外部訪問直接映射到內部 Service，通常為開發測試用途。

Kind	說明	用途
ConfigMap	存儲非機密配置數據。	將配置注入 Pod（環境變量或掛載文件）。
Secret	存儲機密數據（如密碼、API 密鑰）。	安全地將敏感信息注入 Pod。
PersistentVolume (PV)	定義持久化存儲。	提供存儲資源，獨立於 Pod 的生命週期。
PersistentVolumeClaim (PVC)	申請 PersistentVolume。	Pod 使用 PVC 來請求持久存儲資源。
Namespace	將資源分隔成不同的邏輯組。	支持多租戶環境。
Role 和 ClusterRole	定義訪問權限。	管控資源的讀寫權限， <code>Role</code> 限於 Namespace， <code>ClusterRole</code> 全集群。
RoleBinding 和 ClusterRoleBinding	綁定訪問權限到用戶。	授權用戶或服務帳號訪問權限。
HorizontalPodAutoscaler (HPA)	根據指標自動調整 Pod 副本數量。	提高應用的彈性擴展能力。

# 範例 YAML 說明

## 1. CronJob

執行每天凌晨 12 點的數據備份：

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: daily-backup
spec:
  schedule: "0 0 * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: backup
              image: backup-tool:latest
              args:
                - /bin/sh
                - -c
                - "backup-script.sh"
          restartPolicy: OnFailure
```

## 2. DaemonSet

在每個 Node 上部署日誌收集器：

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: log-collector
spec:
  selector:
    matchLabels:
      app: log-collector
  template:
    metadata:
      labels:
```

```
  app: log-collector
spec:
  containers:
  - name: log-collector
    image: log-collector:latest
```

### 3. Deployment

部署一個無狀態 Web 應用，副本數為 3：

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

### 4. Job

執行一次性任務：

```
apiVersion: batch/v1
kind: Job
metadata:
  name: one-time-task
spec:
  template:
    spec:
      containers:
      - name: task-runner
        image: busybox
        args:
        - /bin/sh
        - -c
        - "echo Hello, Kubernetes!"
      restartPolicy: OnFailure
```

### 5. StatefulSet

部署有狀態的數據庫應用：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: database
spec:
  serviceName: "database-service"
  replicas: 3
  selector:
    matchLabels:
      app: database
  template:
```

```
metadata:
  labels:
    app: database
spec:
  containers:
  - name: db
    image: mysql:5.7
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    ports:
    - containerPort: 3306
```

## 6. Pod

部署一個單一容器的應用：

```
apiVersion: v1
kind: Pod
metadata:
  name: single-app
spec:
  containers:
  - name: app
    image: busybox
    args:
    - /bin/sh
    - -c
    - "echo Running a single pod!"
```

## 1. CronJob

執行每天凌晨 12 點的數據備份：

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: daily-backup
spec:
  schedule: "0 0 * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: backup
            image: backup-tool:latest
            args:
            - /bin/sh
            - -c
            - "backup-script.sh"
          restartPolicy: OnFailure
```

## 2. Service

定義一個 Service，將流量路由到 Pod：

```
apiVersion: v1
kind: Service
metadata:
  name: web-app-service
spec:
  selector:
    app: web-app
  ports:
  - protocol: TCP
```



```
port: 80
targetPort: 80
type: ClusterIP
```

- **說明：** 這個 Service 將把進入集群內部的流量路由到標籤 `app: web-app` 的 Pod。

## NodePort 範例

以下是一個簡單的 **NodePort** 配置範例，用於將集群外部的請求轉發到內部的 Pod：

```
apiVersion: v1
kind: Service
metadata:
  name: nodeport-service
spec:
  type: NodePort
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80      # ClusterIP 的服務端口
      targetPort: 8080 # Pod 內部容器的端口
      nodePort: 30007 # 節點上的固定外部端口（30000~32767 之間）
```

- **說明：**
  - **type: NodePort**：指定此服務為 NodePort 類型。
  - **nodePort**：設置固定的外部端口，供外部用戶通過 `<NodeIP>:30007` 訪問服務。
  - **適用場景**：開發測試環境或簡單的外部訪問，不建議用於生產環境，因為其功能有限且缺乏靈活性。

---

🕒 修訂版本 #4

★ 由 treeman 建立於 24 🕒👤👤👤 2024 11:10:01

✎ 由 treeman 更新於 24 🕒👤👤👤 2024 11:53:47