第 1 章

Kubernetes 入門

1.1 Kubernetes 是什麼？

Kubernetes 是什麼？

首先，它是一個全新的基於容器技術的分散式架構解決方案。這個方案雖然還很新，但它是 Google 十幾年來大規模應用容器技術的經驗累積和演進的一個重要成果。確切地說，Kubernetes 是 Google 嚴格保密十幾年的秘密武器——Borg 的開源專案版本。Borg 是 Google 久負盛名的一個內部使用的大規模叢集管理系統，它基於容器技術，目的是實現資源管理的自動化，以及跨多個資料中心的資源利用率最大化。十幾年來，Google 一直透過 Borg 系統管理著數量龐大的叢集式應用系統。由於 Google 員工都簽署了保密協議，即便離職也不能洩露 Borg 的內部設計，所以外界一直無法瞭解它的相關資訊。直到 2015 年 4 月，傳聞許久的 Borg 論文伴隨著 Kubernetes 的發布宣傳被 Google 首度公開，大家才得以瞭解它的更多內幕。正因站在 Borg 這個前輩的肩膀上，吸取了 Borg 過去十年間的經驗與教訓，所以 Kubernetes 一經開源就一鳴驚人，並迅速席捲了容器技術領域。

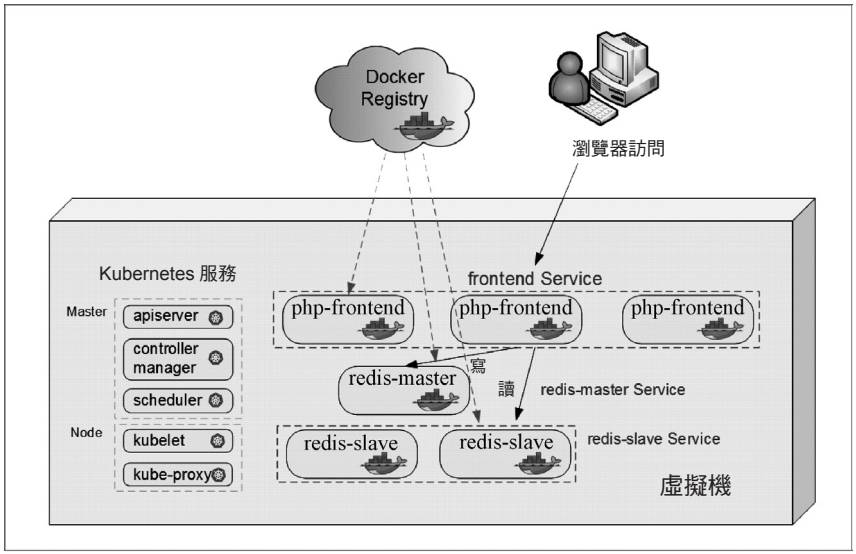

圖 1.3 Kubernetes 部署架構圖

1.3.1 建立 redis-master Pod 及服務

我們可以先定義 Service，然後再定義一個 RC 來建立和控制相對應的 Pod，或者先定義 RC 來建立 Pod，然後定義與其關聯的 Service，這兩種方式最終的結果都一樣，這裡我們採用後面這種方式。

首先為 redis-master 服務建立一個名為 redis-master 的 RC 定義檔：redis-master-controller.yaml。下面提供該檔的完整內容：

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: redis-master
  labels:
    name: redis-master
spec:
  replicas: 1
  selector:
    name: redis-master
  template:
    metadata:
      labels:
        name: redis-master
```

1.4.9 小結

上述這些元件是 Kubernetes 系統的核心元件，它們共同構成 Kubernetes 系統的框架和運算模型。透過對它們進行靈活組合，使用者就可快速、方便地對容器叢集進行配置、建置和管理。

除了以上核心元件，在 Kubernetes 系統中還有許多可供配置的資源物件，例如 LimitRange、ResourceQuota。另外，一些系統內部使用的物件 Binding、Event 等請參考 Kubernetes 的 API 說明文件。

1.5 Kubernetes 整體架構

Kubernetes 叢集由兩種節點所組成：Master 和 Node。在 Master 上執行 etcd、API Server、Controller Manager 和 Scheduler 四個元件，其中後三個元件構成了 Kubernetes 的管控中心，負責對叢集中所有資源進行調度和協作。在每個 Node 上執行 Kubelet、Proxy 和 Docker Daemon 三個元件，負責對本節點上的 Pod 的生命週期進行管理，以及實現服務代理的功能。另外在所有節點上都可以執行 Kubectl 命令列工具，它提供了 Kubernetes 的叢集管理工具集。圖 1.14 描述了 Kubernetes 的系統架構。

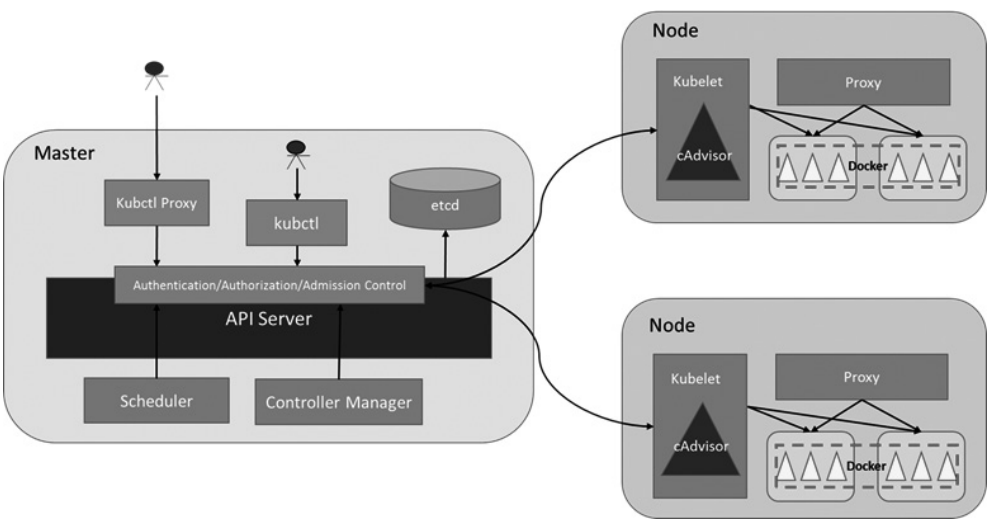

圖 1.14 Kubernetes 的系統架構圖

2.1.2 透過 API Server 訪問 Node、Pod 和 Service

圖 2.1 列出了存取叢集 API Server 及叢集內資源物件互動的情況。

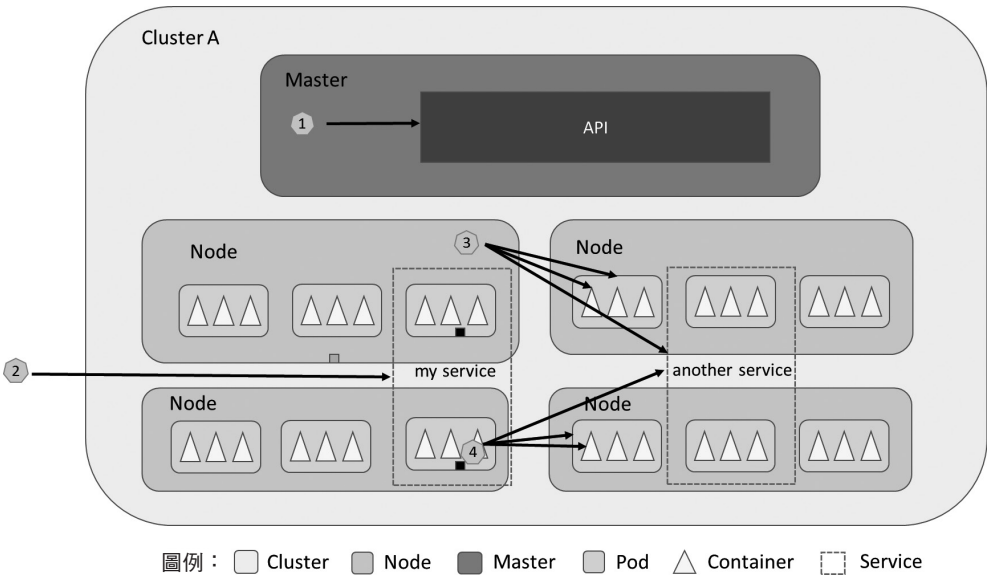

圖 2.1 存取 Kubernetes 叢集

以下是對圖 2.1 中各種存取途徑的詳細說明（每個數字表示一種訪問的途徑）。

(1) 叢集內部各元件、應用或叢集外部應用程式存取 API Server。

(2) 叢集外部系統存取 Service。

(3) 此種情況包含：

- 叢集內跨節點存取 Pod；
- 叢集內跨節點存取容器；
- 叢集內跨節點存取 Service。

(4) 這類情況包含：

- 叢集內的容器存取 Pod；
- 叢集內的容器存取其他叢集內的容器；
- 叢集內的容器存取 Service。

(1) HTTPS 通訊雙方的伺服器端向 CA 機構申請憑證，CA 機構是可信任的協力廠商機構，它可以是一個公認的權威企業，也可以是企業本身。企業內部系統一般都用企業本身的認證系統。CA 機構下發根憑證、服務端憑證及私密金鑰給申請者。

(2) HTTPS 通訊雙方的客戶器端向 CA 機構申請憑證，CA 機構下發根憑證、用戶端憑證及私密金鑰給申請者。

(3) 用戶端向伺服器端發起請求，服務端下發服務端憑證給用戶端。用戶端接收到憑證後，透過私密金鑰解密憑證，並利用伺服器端憑證中的公開金鑰認證憑證資訊比較憑證裡的消息，例如功能變數名稱和公開金鑰與伺服器剛剛發送的相關消息是否一致，如果一致，則用戶端認可這個伺服器的合法身份。

(4) 用戶端發送用戶端憑證給伺服器端，服務端接收到憑證後，透過私密金鑰解密憑證，獲得用戶端憑證公開金鑰，並用該公開金鑰認證憑證資訊，並確認用戶端是否合法。

(5) 用戶端透過隨機金鑰加密資訊，並發送加密後的資訊給服務端。伺服器端和用戶端協商好加密方案後，用戶端會產生一個隨機的金鑰，用戶端透過協商好的加密方案，加密這隨機金鑰，並發送隨機金鑰到伺服器端。 伺服器端接收這個金鑰後，雙方通訊的所有內容都透過隨機金鑰加密。

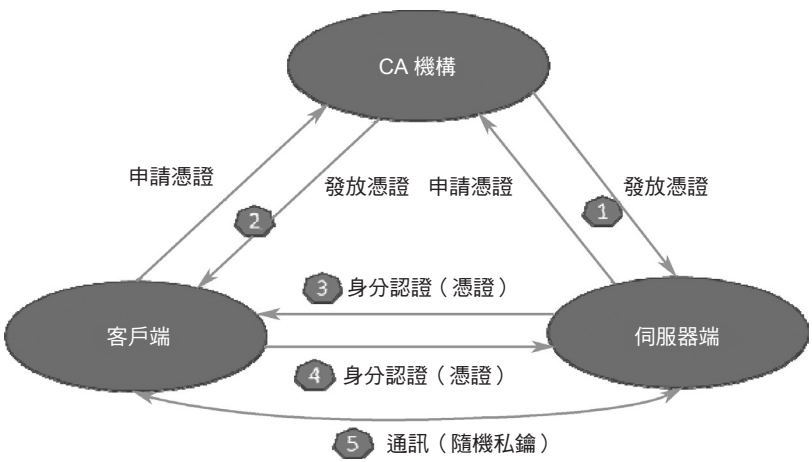

圖 2.12 CA 認證流程

(4) 添加一個“volume”給 Pod，在該“volume”中設定一個能存取 API Server 的 Token（該 Token 來自 Service Account Secret）；

(5) 透過添加“volumeSource”的方式，將上面提到的“volume”掛載到 Pod 中所有容器的 /var/run/secrets/kubernetes.io/serviceaccount 目錄中。

Token Controller 和 Service AccountController 在其自動化過程中所起到的作用請參考 2.2.5 節。

2.5 網路原理

關於 Kubernetes 網路，通常有下列問題需要回答，如圖 2.17 所示。

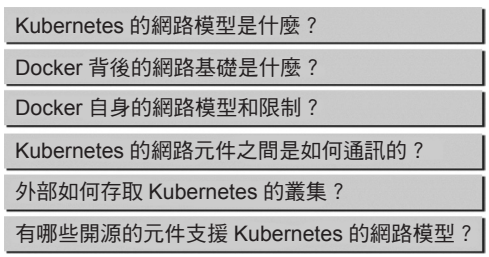

圖 2.17 Kubernetes 常見問題

在本節將分別回答這些問題，然後透過一個具體的試驗，將這些相關的知識串聯在一起。

2.5.1 Kubernetes 網路模型

Kubernetes 網路模型設計的一個基礎原則是：每個 Pod 都擁有一個獨立的 IP 位址，而且假設所有 Pod 都在一個可以直接連線的、扁平的網路空間中。所以不管它們是否運行在同一個 Node（Host 主機）中，都要求它們可以直接透過對方的 IP 進行存取。設計這個原則的主要原因是，使用者不需要額外考慮如何建立 Pod 之間的連線，也不需要考慮將容器連接埠對應到主機連接埠等問題。

實際上在 Kubernetes 的世界裡，IP 是以 Pod 為單位來進行分配的。一個 Pod 內部的所有容器共用一個網路底層堆疊（實際上就是一個網路命名空間，包括它們的 IP

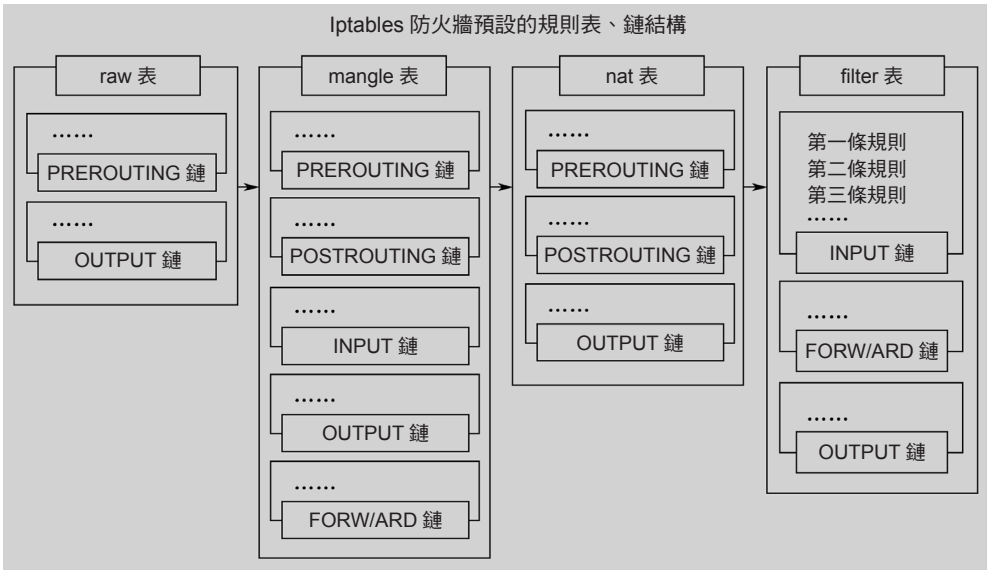

圖 2.23 四種掛接點的規則表

當 Linux 協定堆疊的資料處理執行到掛載節點時，它會依序呼叫掛接點上所有的 hook 函數，直到資料封包的處理結果是確定地接收或拒絕。

2 處理規則

每個規則的特性皆分為以下幾部分：

- 表類型（準備做什麼事情？）；
- 何種掛接點（何時發揮作用？）；
- 比對的參數是什麼（針對何種類型的資料封包？）；
- 比對後有什麼動作（比對後具體的處理是什麼？）。

表類型和何種掛接點在前面已經介紹過，現在我們來看看比對的參數和比對後的動作。

綜合上述，由於 kube-proxy 的作用，在 Service 的呼叫過程中用戶端無須關心後端有幾個 Pod，中間過程的通訊、負載平衡及故障恢復都是透明的，如圖 2.29 所示。

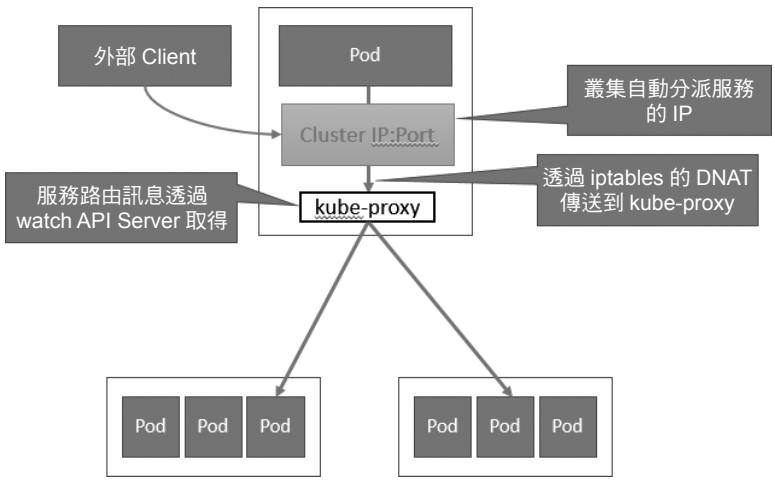

圖 2.29 Service 的負載平衡轉發規則

存取 Service 的請求，不論是用 Cluster IP + TargetPort 的方式，還是用節點主機 IP+ NodePort 的方式，都會被節點主機的 Iptables 規則重導向到 kube-proxy 監聽 Service 服務代理連接埠。kube-proxy 接收到 Service 的存取請求後，會如何選擇後端的 Pod 呢？

首先，目前 kube-proxy 的負載平衡器只支援 ROUND ROBIN 演算法。ROUND ROBIN 演算法按照成員清單逐一選取成員，如果一個回合完後，便從頭開始下一輪，如此循環反覆。kube-proxy 的負載平衡器在 ROUND ROBIN 演算法的基礎上還支援 Session 持續。如果 Service 在定義中指定 Sesssion 持續，則 kube-proxy 接收請求時會從本地端記憶體中尋找是否存在來自該請求 IP 的 affinityState 物件，如果存在該物件，且 Session 沒有超時，則 kube-proxy 將請求轉向該 affinityState 所指向的後端 Pod。如果本地端記憶體沒有來自該請求 IP 的 affinityState 物件，則按照 ROUND ROBIN 演算法為該請求挑選一個 Endpoint，並建立一個 affinityState 物件，記錄請求的 IP 和指向的 Endpoint。後面的請求就會綁定到這個建立好的 affinityState 物件上，這就實現了用戶端 IP session 持續的功能。

接下來將深入分析 kube-proxy 的實作細節。kube-proxy 程序為每個 Service 都建立了一個“服務代理物件”，服務代理物件是 kube-proxy 程式內部的一種資料結構，

目前已經有多個開源元件支援這個網路模型。這裡將介紹幾個常見的模型，分別是 Flannel、Open vSwitch 及直接路由的方式。

1. Flannel

Flannel 之所以可以建置 Kubernetes 依賴的底層網路，是因為它能實現以下兩點：

(1) 它能協助 Kubernetes，給每一個 Node 上的 Docker 容器分配互不衝突的 IP 地址。

(2) 它能在這些 IP 位址之間建立一個層疊網路（Overlay Network），透過這個層疊網路，將資料封包原封不動地傳遞到目標容器內。

透過圖 2.31 來看看 Flannel 是如何實現這兩點，如圖 2.31 所示。

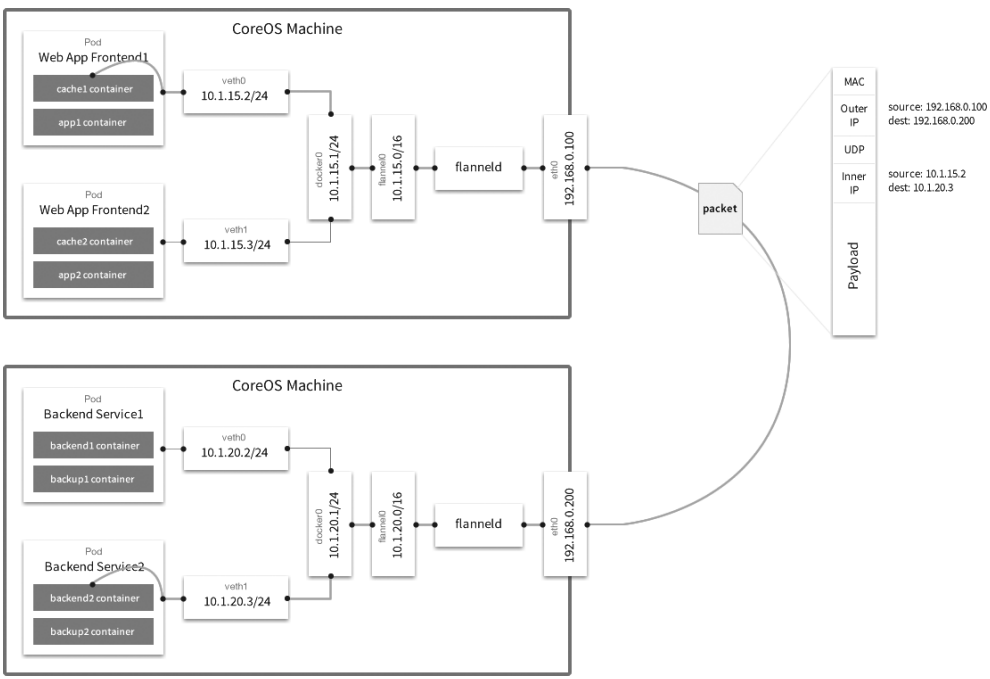

圖 2.31 Flannel 架構圖

透過圖 2.31 可以看到，首先 Flannel 建立了一個 flannel0 的橋接器，而且這個橋接器的一端連接 docker0 橋接器，另一端連接著一個叫作 flanneld 的 daemon 服務。

在 Kubernetes 叢集中，對於一個新 Node 的加入是非常簡單的。可以在 Node 節點上安裝 Docker、Kubelet 和 kube-proxy 服務，然後將 Kubelet 和 kube-proxy 的啟動參數中的 Master URL 指定為目前 Kubernetes 叢集 Master 的位址，最後啟動這些服務。基於 Kubelet 的自動註冊機制，新的 Node 將會自動加入現有的 Kubernetes 叢集中，如圖 4.1 所示。

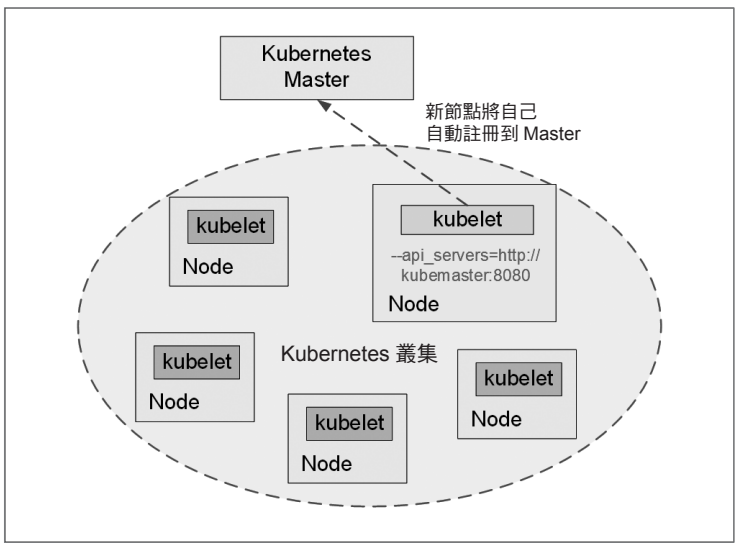

圖 4.1 新節點自動註冊完成擴充

Kubernetes Master 在接受新 Node 的註冊之後，會自動將其納入目前叢集的調度範圍內，在之後建立容器時，就可以對新的 Node 進行調度了。

透過這種機制，Kubernetes 實現了叢集的擴展。

4.3.3 Pod 動態擴展和縮放

在實際營運系統中，我們經常會遇到某個服務需要擴展的情境，也可能會遇到由於資源緊張或工作負載降低而需要減少服務實例數的情形。此時可以利用命令 kubectl scale rc 來完成這些任務。以 redis-slave RC 為例，已定義的最初抄本數量為 2，透過執行下面的命令將 redis-slave RC 控制的 Pod 抄本數量從初始的 2 更新為 3：

```
$ kubectl scale rc redis-slave --replicas=3
scaled
```

```
    kubernetes.io/name: "KubeUI"
spec:
  selector:
    k8s-app: kube-ui
  ports:
  - port: 80
    targetPort: 8080
```

透過 kubectl create 命令完成建置：

```
$ kubectl create -f kube-ui-rc.yaml
$ kubectl create -f kube-ui-svc.yaml
```

啟動成功後，透過 Kubernetes Master 的 IP 位址來存取 kube-ui：

```
https://<kubernetes-master>:<port>/ui
```

該 URL 將會被重導向到：

```
https://<kubernetes-master>:<port>/api/v1/proxy/namespaces/kube-system/services/
kube-ui/#/dashboard/
```

圖 4.6 顯示了 kube-ui 的主頁，展示所有 Node 的資訊，並且每秒更新顯示每個 Node 的 CPU 使用率、記憶體使用情況和檔案系統的使用情況。

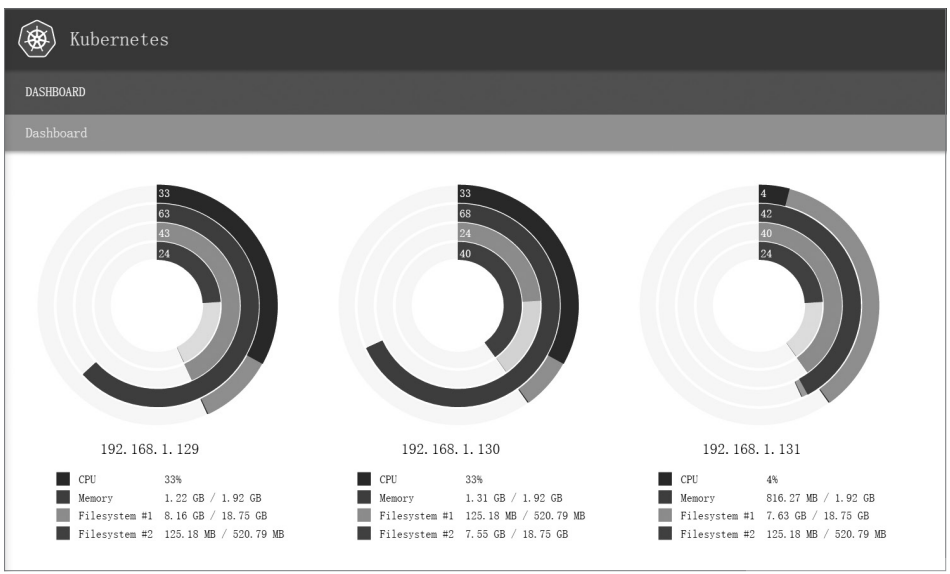

圖 4.6 kube-ui 主頁

- Grafana：透過 Dashboard 將 InfluxDB 中的時序資料展現成圖表或曲線等形式，便於維運人員查看叢集的運行狀態。Grafana 的網頁為 http://grafana.org。

整體架構如圖 5.2 所示。

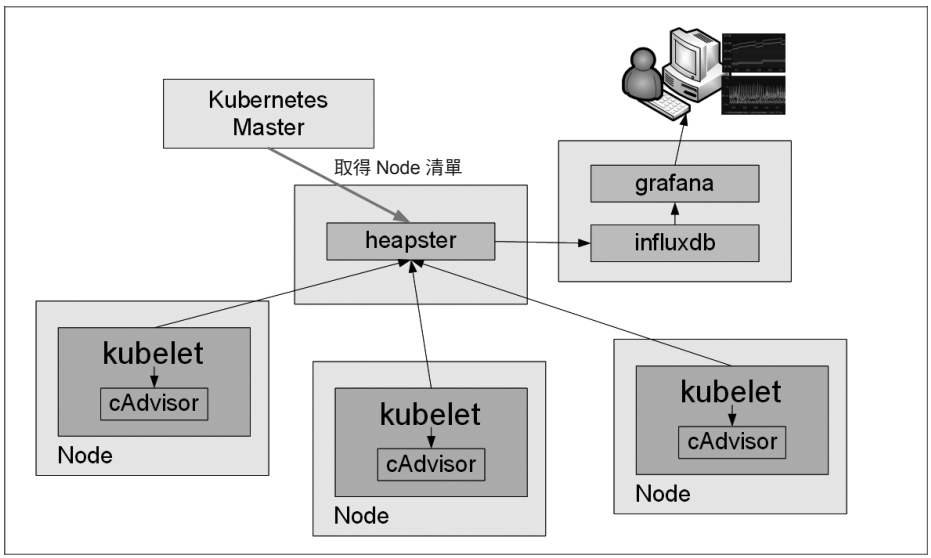

圖 5.2 Heapster 叢集監控系統架構圖

在 Kubernetes 的目前版本中，Heapster、InfluxDB 和 Grafana 均以 Pod 的形式啟動和運行。

在啟動這些 Pod 之前，首先需要為 Heapster 配置與 Master 的安全連線。

5.2.1 配置 Kubernetes 叢集的 ServiceAccount 和 Secret

Heapster 目前版本需要使用 HTTPS 的安全方式與 Kubernetes Master 進行連線，所以需要先進行 ServiceAccount 和 Secret 的建立。如果不使用 Secret，則 Heapster 啟動時將會出現錯誤：

```
/var/run/secret/kubernetes.io/serviceaccount/token no such file or directory
```

然後 Heapster 容器會被 ReplicationController 反覆銷毀、建立，無法正常運作。