第一章

前言

Kubernetes 是部署容器化應用程式的開源編排器（orchestrator）。最初由 Google 開發，靈感來自於十年間透過應用導向 API 部署可擴展且可靠的容器系統經驗[1]。

Kubernetes 不僅是 Google 開發的技術，還擁有豐富且持續成長的開源社群。Kubernetes 不但適合一般網路服務公司，也適合各種規模的雲端原生開發者，小至樹莓派（Raspberry Pi）叢集，大至充滿最新設備的資料中心。都可以透過 Kubernetes 建置和部署可靠又可擴展的分散式系統。

你可能好奇「可靠可擴展的分散式系統」是什麼。越來越多的服務透過 API 在網路上被實現著。這些 API 通常被**分散式系統**實現，運行在不同的機器上，和透過網路連接及溝通協調。因為我們在日常生活中越來越倚賴這些 API（例如：尋找最近醫院的路徑），所以這些系統必須有很高的**穩定性**。這可不能出錯，即使只是小部分的當機或是其他的問題。軟體更新或是其他的系統維護也是如此，都要保持**可用性**。最後，由於越來越多的人使用這些服務，因此必須具有高**可擴展性**，以便能跟上不斷增加的使用量，而不用徹底的重新設計分散式系統來改善服務。

[1] 由 Brendan Burns 以及其他人所著作的「Google 十幾年來從 Borg、Omega 和 Kubernetes 中得到的經驗教訓」，*ACM Queue* 第 14 卷（2016）：第 70-93 頁，網址：*http://bit.ly/2vIrL4S*。

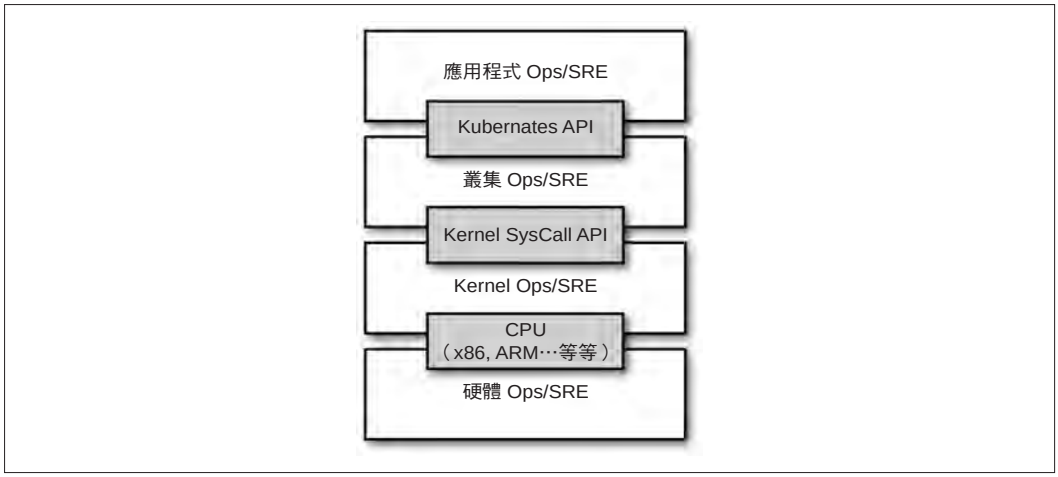

圖 1-1　說明不同團隊如何利用 API 進行去耦化

當然，投入一個團隊去管理 OS，是超出許多組織能力所及的事了。這個時刻，公有雲供應商提供了 Kubernetes-as-a-Service（KaaS）也是不錯的選擇。

撰寫本文時，你可以在 Microsoft Azure、Google Cloud Platform 上使用 KaaS，分別為 Azure Container Service 和 Google Kubernetes Engine（GKE）。而 Amazon Web Services（AWS）沒有類似的服務，可以透過 kops 的工具簡單安裝和管理 Kubernetes（請參閱第 27 頁：安裝 Kubernetes 在 Amazon Web Services 上）。

要不要使用 KaaS 或自行管理 Kubernetes，按照個人的技能和需求。通常對小規模的組織來說，KaaS 提供易用的解決方案，讓組織將時間精力專注於建構軟體，而不是管理叢集上。對於一個能夠負擔 Kubernetes 叢集專業管理團隊的大型組織來說，自行管理是有意義的，因為在叢集的功能和操作方面提供了更大的靈活性。

你可以看到這是一個擁有 4 個 node 的叢集，它們已經運行了 45 天。在 Kubernetes 中，node 分為主節點和工作節點，主節點負責管理叢集，它包含 API 伺服器、排程器的容器…等，而工作節點則是負責運行你的容器。為了確保使用者的工作負載不會對叢集的整體操作造成損害，因此 Kubernetes 通常不會將工作安排到主節點上。

你可以使用 `kubectl describe` 指令取得更多有關特定 node（如 `node-1`）的資訊：

```
$ kubectl describe nodes node-1
```

首先，你會看到有關該 node 的基本資訊：

```
Name:                   node-1
Role:
Labels:                 beta.kubernetes.io/arch=arm
                        beta.kubernetes.io/os=linux
                        kubernetes.io/hostname=node-1
```

你可以看到此 node 使用 ARM 處理器並且運行 Linux 作業系統上。

接下來，你會看到有關 `node-1` 本身操作的資訊：

```
Conditions:
  Type           Status LastHeartbeatTime  Reason                     Message
  ----           ------ -----------------  ------                     -------
  OutOfDisk      False  Sun, 05 Feb 2017…  KubeletHasSufficientDisk   kubelet…
  MemoryPressure False  Sun, 05 Feb 2017…  KubeletHasSufficientMemory kubelet…
  DiskPressure   False  Sun, 05 Feb 2017…  KubeletHasNoDiskPressure   kubelet…
  Ready          True   Sun, 05 Feb 2017…  KubeletReady               kubelet…
```

這些狀態表明該 node 具有足夠的硬碟和記憶體空間，同時回報至 Kubernetes 主節點，這個 node 是健康的。接下來的資訊有關於機器的容量：

```
Capacity:
 alpha.kubernetes.io/nvidia-gpu:        0
 cpu:                                   4
 memory:                                882636Ki
 pods:                                  110
```

```
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-cg5f5 (ro)
```

最後是有關 Pod 的事件（例如，被調度的時間、被提取映像檔的時間，以及是否因為未通過健康檢查而必須重啟）。

```
Events:
  Seen From              SubObjectPath           Type      Reason    Message
  ---- ----              -------------           --------  ------    -------
  50s  default-scheduler                         Normal    Scheduled Success…
  49s  kubelet, node1    spec.containers{kuard}  Normal    Pulling   pulling…
  47s  kubelet, node1    spec.containers{kuard}  Normal    Pulled    Success…
  47s  kubelet, node1    spec.containers{kuard}  Normal    Created   Created…
  47s  kubelet, node1    spec.containers{kuard}  Normal    Started   Started…
```

移除 Pod

當要刪除 Pod，可以透過名稱刪除：

```
$ kubectl delete pods/kuard
```

或透過當初建立的 manifest 檔，刪除它：

```
$ kubectl delete -f kuard-pod.yaml
```

當 Pod 被刪除，不會馬上被移除：執行 `kubectl get pods`，可以看見這個 Pod 目前處於 `Terminating` 的狀態。所有 Pod 都有終止限期（*grace period*）。預設是 30 秒。當 Pod 轉換成 `Terminating`，不再接受新的請求。在正常的場景，限期（grace period）對於可靠性很重要，因為這樣能夠讓 Pod 在終止之前，將正在處理中的請求處理完畢。

必須特別注意，當刪除 Pod 時，所有存在於相關容器內的資料都會被刪除。如果想跨多機器永久儲存資料，必須使用 `PersistentVolume`，在本章後面會詳細介紹。

你有可能會看到，上面範例中沒看到的 label（`pod-template-hash`）。這個 label 由 Deployment 附加的，以便追蹤不同 Pod 模板，產生了哪些 Pod。這能夠讓 Deployment 以簡潔的方式管理任何更新，將會在第 12 章有更深入的內容

可以利用 `--selector` 的旗標，列出 `ver` 的 label 等於 `2` 的 Pod：

```
$ kubectl get pods --selector="ver=2"

NAME                                READY     STATUS    RESTARTS   AGE
alpaca-test-1004512375-3r1m5        1/1       Running   0          3m
bandicoot-prod-373860099-0t1gp      1/1       Running   0          3m
bandicoot-prod-373860099-k2wcf      1/1       Running   0          3m
bandicoot-staging-1839769971-3ndv5  1/1       Running   0          3m
```

如果指定兩個選擇器用逗號（,）區隔，此時只有兩個條件同時符合的物件，才會被輸出。這就是 AND 的邏輯操作：

```
$ kubectl get pods --selector="app=bandicoot,ver=2"

NAME                                READY     STATUS    RESTARTS   AGE
bandicoot-prod-373860099-0t1gp      1/1       Running   0          4m
bandicoot-prod-373860099-k2wcf      1/1       Running   0          4m
bandicoot-staging-1839769971-3ndv5  1/1       Running   0          4m
```

也可以篩選 label 是否有包含某些值。以下範例介紹，如何在 `app` label 中，包含 `alpaca` 或 `bandicoot` 的 Pod（應該會輸出剛剛建立的六個 Pod）。

```
$ kubectl get pods --selector="app in (alpaca,bandicoot)"

NAME                                READY     STATUS    RESTARTS   AGE
alpaca-prod-3408831585-4nzfb        1/1       Running   0          6m
alpaca-prod-3408831585-kga0a        1/1       Running   0          6m
alpaca-test-1004512375-3r1m5        1/1       Running   0          6m
bandicoot-prod-373860099-0t1gp      1/1       Running   0          6m
bandicoot-prod-373860099-k2wcf      1/1       Running   0          6m
bandicoot-staging-1839769971-3ndv5  1/1       Running   0          6m
```

```
bandicoot-prod-5678-sbxzl  ... 10.112.1.55 ... app=bandicoot,env=prod,ver=2
bandicoot-prod-5678-x0dh8  ... 10.112.2.86 ... app=bandicoot,env=prod,ver=2
```

這很方便，但如果有一堆 Pod 要怎麼辦？你應該會想要基於 label 篩選一部分的 Deployment。以下方法可以只篩選 `alpaca` 的應用程式：

```
$ kubectl get pods -o wide --selector=app=alpaca,env=prod

NAME                           ... IP          ...
alpaca-prod-3408831585-bpzdz ... 10.112.1.54 ...
alpaca-prod-3408831585-kncwt ... 10.112.2.84 ...
alpaca-prod-3408831585-l9fsq ... 10.112.2.85 ...
```

此時，我們有基礎服務探索的功力了！可以利用 label 來識別有需要的 Pod，透過這些 label 取得 Pod 和取得這些 IP 位址。但對於持續配置一組正確的 label 是棘手的。這就是為什麼要有 `Service` 物件的原因。

kube-proxy 和 Cluster IP

cluster IP 是靜態的虛擬 IP，它負載平衡流量到 service 中的所有 endpoint。有一個很厲害的組件叫做 `kube-proxy`，它運行在每個 node 中。（圖 7-1）

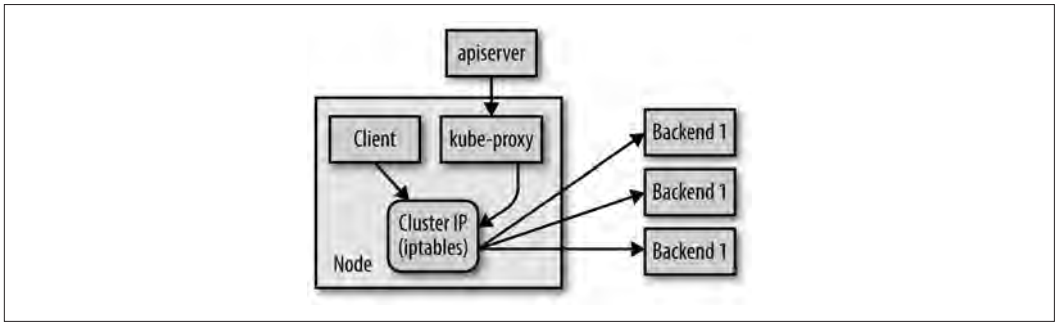

圖 7-1　配置和使用 cluster IP

運行這個 Pod，讓我們來看看應用程式如何看待這個世界：

```
$ kubectl apply -f kuard-config.yaml
$ kubectl port-forward kuard-config 8080
```

現在來看瀏覽器，進入 *http://localhost:8080*。可以看見如何使用三種不同方式，將組態值插入到程式中。

點擊左邊的「Server Env」分頁。這會顯示應用程式，隨著環境一起啟動的命令列，如圖 11-1 所示。

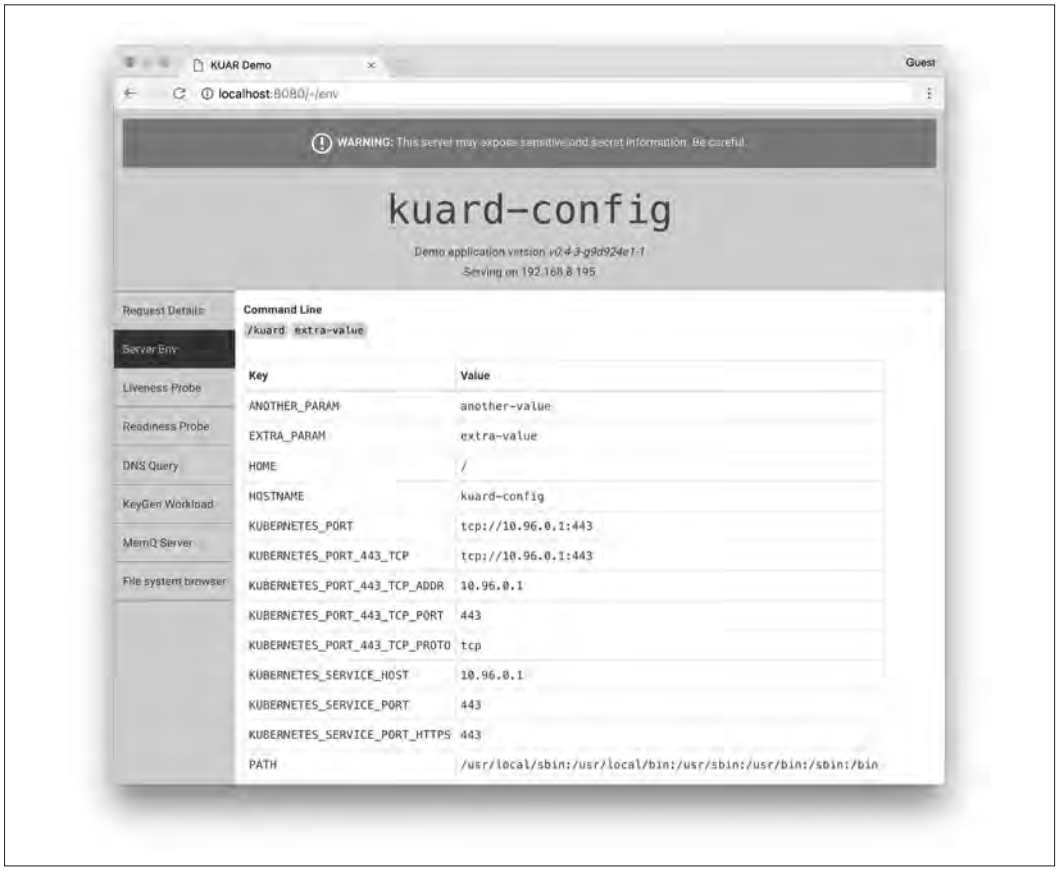

圖 11-1　kuard 的環境變數

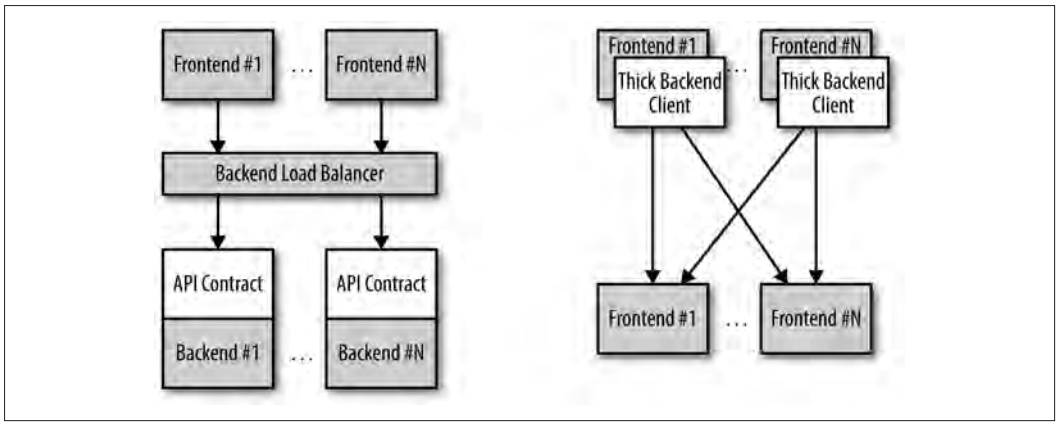

圖 12-1　去耦合（左）和耦合（右）應用架構

配置滾動更新

`RollingUpdate` 是通用的策略；它可以用多樣化的設置來更新各種應用程式。必然地，滾動更新本身是可配置的；可以調整它的行為來因應特殊需求。使用 `maxUnavailable` 和 `maxSurge` 這兩個參數來調整滾動更新的行為。

`maxUnavailable` 參數配置滾動更新期間所允許最大不可用的 Pod 數目。可以設置為絕對值（例如：設定為 `3`，表示最多 3 個 Pod 不可用），或百分比（例如：設定為 `20%`，表示期望 replica 數目中的 20%允許不可用）。

使用百分比表示以大多數服務來說是很好的方法，因為無論 Deployment 中期望的 replica 數是多少，該值都可以正確的被應用。但有時會使用絕對值（例如：將允許不可用的 Pod 限制為一個）。

重要的是 `maxUnavailable` 可幫助調整滾動更新的執行速度。舉例來說，如果將 `maxUnavailable` 設為 `50%`，則滾動更新時會立即將舊的 ReplicaSet 縮小到原來大小的 50%。如果您有四個 replica，則會縮小為兩個 replica。滾動更新將新的 ReplicaSet 擴展到兩個 replica，來替換被移除的 Pod，現在總共有四個 replica（兩個舊的，兩個新的）。接下來，它會將舊的 ReplicaSet 縮小到 0 個 replica，總大小

限制 rollout 的時間，乍看之下似乎是多餘的。然而，越來越多的事情，像是 rollout 行為它是由系統自動觸發的，但這幾乎是無人工介入。在這樣情況下，超時（time out）成為重要的例外事件，它可以自動觸發回到上一版本，也可以建立 ticket 或事件，以人工來介入處置。

要設定超時，使用 Deployment 中的 `progressDeadlineSeconds`：

```
...
spec:
  progressDeadlineSeconds: 600
...
```

這個範例，將整個過程的最終時間設為 10 分鐘。如果 rollout 中任何特定階段在 10 分鐘內都沒有進展，則 Deployment 會被標記為失敗，並且接下來的 Deployment 都會中斷。

要注意的是，這個超時是根據 Deployment 的**進度**，而不是 Deployment 的總長度。在這種情況下，任何時候 Deployment 建立或刪除 Pod 都會定義進度。當發生這種情況時，超時計時被重置為 0。

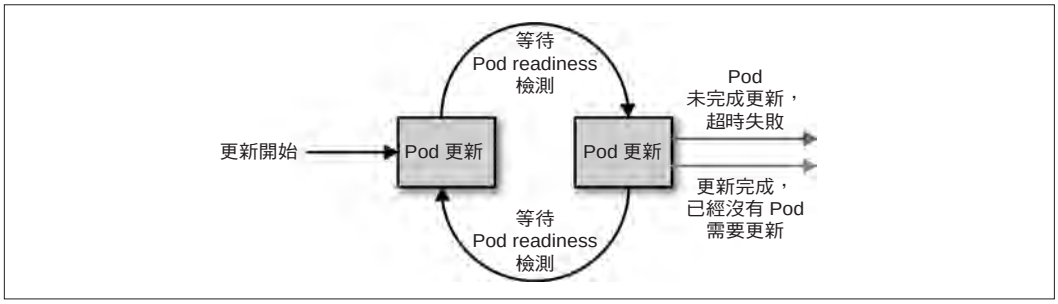

圖 12-2　Kubernetes Deployment 的生命週期

建立樹莓派（Raspberry Pi）的 Kubernetes 叢集

雖然人們入門 Kubernetes 經常都從公有雲開始，藉由瀏覽器或終端機來觸及雲端上的叢集，但是在裸機上構建 Kubernetes 叢集對於實體機上是非常有幫助的經驗。同樣地，沒什麼能比真的拔掉節點上的電源或網路，看見 Kubernetes 怎麼恢復應用程式來說服你它的實用性。

建立自己的叢集看似既是項艱鉅又是個昂貴工作，但其實不是。透過單晶片（system-on-chip）這種低成本系統，以及為了讓 Kubernetes 更容易安裝，社群做了很多完善，這表示著你可以在幾個小時內，建立小型 Kubernetes 叢集。

在接下來的介紹中會專注於構建樹莓派的叢集，但稍加修改後就能以相同的方法來處理其他的單板機。

零件清單

要構建樹莓派的叢集，第一件事就是組裝這些零件。本章會以一組四個節點的叢集做為範例。你也可以建立一組三個節點的叢集，或甚至一百個節點的叢集，但是四個是非常好的數字。

設定網路配置

接下來是設定主節點的網路。

先設定 WiFi 讓叢集與外面世界之間建立連結。編輯 */boot/device-init.yaml* 檔。修改與你現在環境相同的 WiFi SSID 和密碼。如果你要切換網路，那就是要使用這個檔案。編輯完成後，利用 `sudo reboot` 重啟樹莓派，並驗證網路正常運作。

下一步要樹莓派在內部網路中配置靜態 IP 位址。編輯 */etc/network/interfaces.d/eth0*：

```
allow-hotplug eth0
iface eth0 inet static
    address 10.0.0.1
    netmask 255.255.255.0
    broadcast 10.0.0.255
    gateway 10.0.0.1
```

這樣的配置會讓主要乙太網路介面，具有固定 IP 位址（10.0.0.1）。

重啟設備，以宣告 10.0.0.1 這個 IP。

接下來，在該主節點上安裝 DHCP，以便將 IP 分配給其他節點。執行以下指令：

```
$ apt-get install isc-dhcp-server
```

然後按以下方法，配置 DHCP 伺服器：

```
# 設定域名，基本上可以任意設定
option domain-name "cluster.home";

# 預設使用 Google DNS，你可以用 ISP 提供的 DNS 取代
option domain-name-servers 8.8.8.8, 8.8.4.4;

# 我們使用 10.0.0.X 作為子網路
subnet 10.0.0.0 netmask 255.255.255.0 {
    range 10.0.0.1 10.0.0.10;
    option subnet-mask 255.255.255.0;
```

額外加分

對於網路還有一些技巧，能夠更輕鬆地管理叢集。

首先是編輯每台機器上的 */etc/hosts*，以將名稱映射到正確的地址。在每台機器上新增：

```
...
10.0.0.1 kubernetes
10.0.0.2 node-1
10.0.0.3 node-2
10.0.0.4 node-3
...
```

現在，可以在連進這些機器中時使用這些名稱。

第二是設置無密碼 SSH 登入。執行 `ssh-keygen`，然後將 *$HOME/.ssh/id_rsa.pub* 檔，附加到 `node-1`、`node-2` 和 `node-3` 中的 `/home/pirate/.ssh/authorized_keys`。

安裝 Kubernetes

這時候，所有節點應該已經全數上線、有 IP 位址，並且能夠連到網際網路。是時候將所有節點安裝 Kubernetes 了。

利用 SSH，在所有節點中執行以下指令安裝 `kubelet` 和 `kubeadm` 的工具。必須是要 root 才能夠要執行指令。可以利用 `sudo su`，轉換成 root 使用者。

首先，為套件加入加密金鑰：

```
# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
```

然後將套件來源加入到儲存庫列表中：

```
# echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" \
  >> /etc/apt/sources.list.d/kubernetes.list
```