第三章

Kubernetes 的監看與日誌紀錄

在這　章裡，我們要來探討　些關於在 Kubernetes 內進行監看與日誌紀錄的最佳實務做法。我們會深入介紹各種監看模式的細節、需要蒐集的重要指數（metrics）、以及如何以原始的指數建立看板。然後我們會用範例說明如何做出 Kubernetes 叢集的監看功能。

指標與日誌

你必須先了解日誌蒐集和指數蒐集的差別。它們彼此相輔相成，但各自卻有截然不同的目的。

指數（*Metrics*）

　在　段時間內測量而得的　連串數字

日誌（*Logs*）

　用來對系統進行探索分析

何處會同時需要用到指數和日誌？最好的例子就是當應用程式表現差勁的時候。問題的第　道警訊也許是含有應用程式的 pod 出現顯著延遲，但指數卻不　定能有效地顯示問題。於是我們便開始研究日誌，著手調查應用程式發出的錯誤訊息。

監看的技術

黑箱監看多半專注在從應用程式外部進行的監看，傳統上在監看 CPU、記憶體、儲存設備等系統元件時都以這種方式為主。對於基礎設施層級的監看而言，黑箱監看也還是很有用，但它缺乏對於應用程式如何運作的背景知識。例如說，若要測試叢集是否健康，我們也許會試著準備　個 pod，如果 pod 成功建立，我們就知道叢集中的調度工具（scheduler）和服務尋找（service discovery）是正常的，進而假設叢集其他元件也都健康無礙。

白箱監看則專注在應用程式狀態的背景詳情上，例如 HTTP 請求的總數、500 號錯誤訊息[譯註 1] 的數量、請求的延滯時間等等。透過白箱監看，我們就可以理解系統狀態「何以至此」。然後我們就能自問「為什麼磁碟空間會爆滿？」而不是只知道「噢，磁碟滿了。」

監看的模式

你也許會覺得「這有何難？我們　直都有在監看系統啊。」沒錯，如今有　些典型的監看模式同樣也適於用來監看 Kubernetes。差別只在於像 Kubernetes 這樣的平台會比傳統環境更為動態易變，而你必須改變對於如何監看這類環境的思維。例如說，在監看　個虛擬機器（VM）時，你預期該部 VM 會全天候運轉、而且所有的狀態都會保存完好。但是在 Kubernetes 裡，pods 會頻繁變動、而且十分短命，因此你的監看方式必須要能處理動態和短暫等特性。

在監看分散式系統時，有幾種不同的監看模式。

首先是 Brendan Gregg 提出的 *USE* 方法，它專注在以下內容：

- U—使用率（Utilization）
- S—飽和程度（Saturation）
- E—錯誤（Errors）

譯註 1　500 代表伺服器內部錯誤（internal server error）。

以 Prometheus 監看 Kubernetes

在這　節裡，我們要特別說明 Prometheus 的監看指數，它與 Kubernetes 的標籤、服務搜尋（service discovery）、以及中介資料的整合都十分良好。我們在本章實作時所憑據的高階觀點，同樣適用於其他的監看產品。

Prometheus 是　項由 CNCF（雲端原生運算基金會）主持的開放原始碼專案。它原本由 SoundCloud 開發，大部分的觀念都源於 Google 內部的監看系統 BorgMon。Prometheus 利用了成對鍵（keypairs）來實作多維度資料模型，其運作方式與 Kubernetes 標籤系統頗為相似。Prometheus 同時以方便人類判讀的格式來公開指數，範例如下：

```
# HELP node_cpu_seconds_total Seconds the CPU is spent in each mode.
# TYPE node_cpu_seconds_total counter
node_cpu_seconds_total{cpu="0",mode="idle"} 5144.64
node_cpu_seconds_total{cpu="0",mode="iowait"} 117.98
```

為了蒐集指數，Prometheus 採用抽取模式，主動取得受監看端點的指數，經過處理後交給 Prometheus 伺服器。像 Kubernetes 這樣的系統早已使用 Prometheus 格式來公開其內部指數，因此以 Prometheus 蒐集指數會更為容易。許多 Kubernetes 的周邊專案（例如 NGINX、Traefik、Istio、LinkerD 等等）同樣也採用 Prometheus 格式來公開指數資料。Prometheus 也提供匯出工具（exporters），讓你可以取得服務所發出的指數、再轉譯成 Prometheus 格式的指數。

Prometheus 的架構簡單明瞭，如圖 3-1 所示。

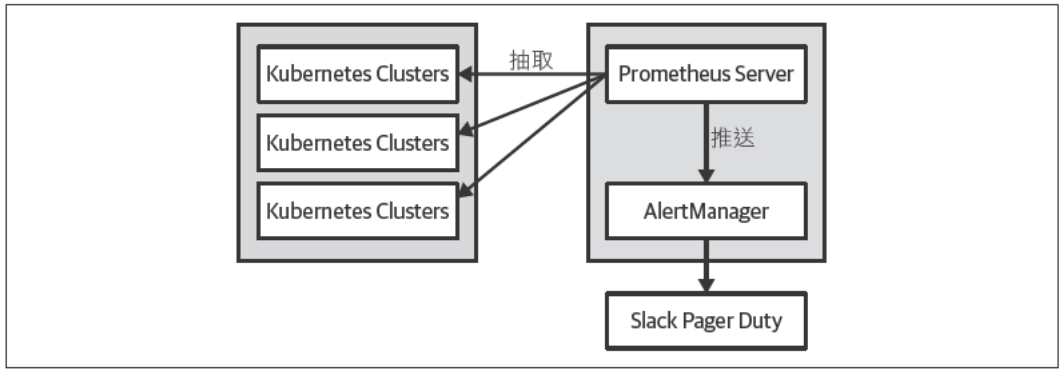

圖 3-1　Prometheus 架構

監看堆疊會部署以下的內容：

- Elasticsearch Operator
- Fluentd（負責從 Kubernetes 環境將日誌轉送至 Elasticsearch）
- Kibana（負責將搜尋、檢視，以及與儲存在 Elasticsearch 中的日誌互動視覺化的工具）

把以下的項目清單（manifest）部署到 Kubernetes 叢集當中：

```
kubectl create namespace logging

kubectl apply -f https://raw.githubusercontent.com/dstrebel/kbp/master/
elasticsearch-operator.yaml -n logging
```

部署 Elasticsearch operator 以便集中所有轉送的日誌：

```
kubectl apply -f https://raw.githubusercontent.com/dstrebel/kbp/master/efk.yaml
-n logging
```

這會分別部署 Fluentd 和 Kibana，讓我們可以把日誌轉送到 Elasticsearch、並用 Kibana 將日誌視覺化。

你應該會看到叢集中出現以下的 pods：

```
kubectl get pods -n logging

efk-kibana-854786485-knhl5                1/1     Running   0          4m
elasticsearch-operator-5647dc6cb-tc2st    1/1     Running   0          5m
elasticsearch-operator-sysctl-ktvk9       1/1     Running   0          5m
elasticsearch-operator-sysctl-lf2zs       1/1     Running   0          5m
elasticsearch-operator-sysctl-r8qhb       1/1     Running   0          5m
es-client-efk-cluster-9f4cc859-sdrsl      1/1     Running   0          4m
es-data-efk-cluster-default-0             1/1     Running   0          4m
es-master-efk-cluster-default-0           1/1     Running   0          4m
fluent-bit-4kxdl                          1/1     Running   0          4m
fluent-bit-tmqjb                          1/1     Running   0          4m
fluent-bit-w6fs5                          1/1     Running   0          4m
```

旦所有的 pods 都「跑起來」（Running），就可以用本機通訊埠轉向連線到 Kibana 了：

```
export POD_NAME=$(kubectl get pods --namespace logging -l
"app=kibana,release=efk" -o jsonpath="{.items[0].metadata.name}")

kubectl port-forward $POD_NAME 5601:5601
```

用瀏覽器開啟 *http://localhost:5601* 以開啟 Kibana 看板。

要操作來自 Kubernetes 叢集的日誌，必須先建立索引。

當你初次啟動 Kibana 時，請先瀏覽管理（Management）頁籤，並建立 Kubernetes 日誌的索引模式（index pattern）。系統會引導你完成必要的步驟。

索引建好後，就可以用 Lucene 查詢語法搜尋日誌內容，就像這樣：

```
log:(WARN|INFO|ERROR|FATAL)
```

這會取出含有 warn（警告）、info（資訊）、error（錯誤）或 fatal（致命）等欄位的所有日誌。圖 3-4 顯示的就是示範輸出。

圖 3-4　Kibana 的看板

在 Kibana 裡，你可以對日誌執行特殊查詢，也可以建立包含環境　覽的看板。

請自己四下探索　番，看看 Kibana 裡有那些日誌可以加上視覺化效果。

部署策略

既然我們已經對 CD 的原理略有所知了，現在就來看看有哪幾種不同的推行策略可以使用。Kubernetes 為新版應用程式的推行提供了數種策略。即使 Kubernetes 自己內建了一套滾動式更新機制，你還是應該參酌其他更高段的策略。以下我們會　　檢視幾種為應用程式交付更新內容的策略：

- 滾動式升級（rolling updates）
- 藍標 / 綠標部署法
- 金絲雀（Canary）部署法

滾動式升級是 Kubernetes 內建的作法，讓你可以對正在運行的應用程式觸發　次更新，但不會引起服務中斷（downtime）。舉例來說，如果你的前端應用程式正在執行 frontend:v1，而 Deployment 已更新為 frontend:v2，Kubernetes 就會以滾動的方式將抄本更新為 frontend:v2。圖 5-1 展示的就是滾動式更新。

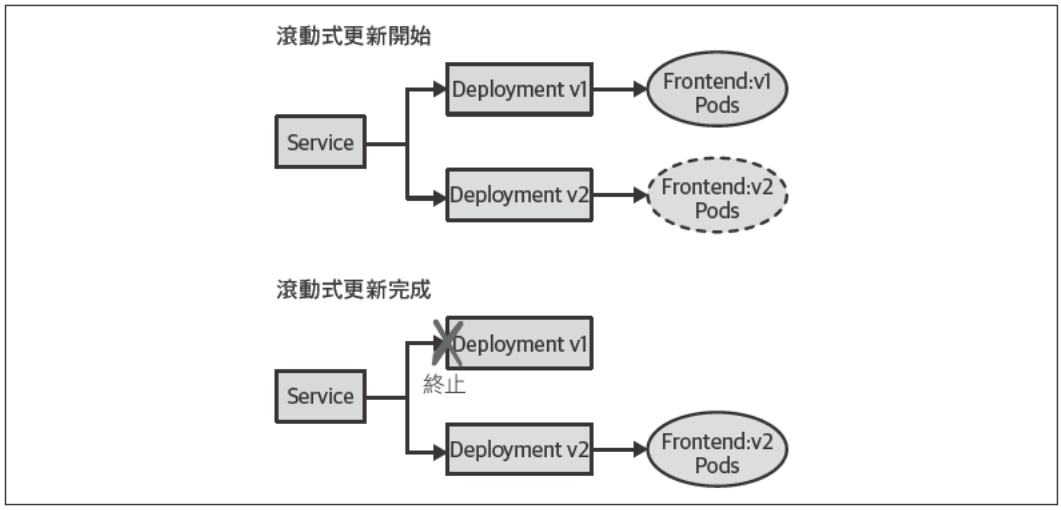

圖 5-1　Kubernetes 的一次滾動式更新

你可以在 Deployment 物件中定義　次最多可以更新幾份抄本，以及發行更新時最多可以有多少個 pods 離線。下述項目清單（manifest）就是如何指定滾動式更新策略的示範：

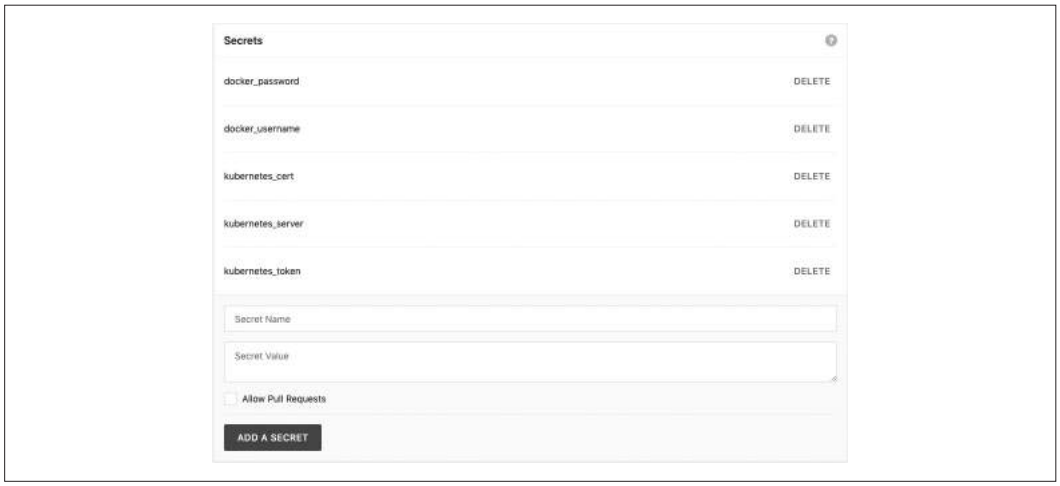

圖 5-4　Drone 的密語設定畫面

你需要在 Kubernetes 叢集中擁有 cluster-admin 的權限，才能執行本小節所述的步驟。

以下指令可以取得 API 端點的資料：

```
kubectl cluster-info
```

這時你應該會看到以下的訊息：Kubernetes master is running at *https://kbp.centralus.azmk8s.io:443*（意為 Kubernetes 主節點運行在這串網址上）。請把取得的資料放在 `kubernetes_server` 這個密語裡。

接著要建立一個服務帳號，讓 Drone 可以連接到叢集。請以下列指令建立一個 `serviceaccount`：

```
kubectl create serviceaccount drone
```

再以下列指令替 `serviceaccount` 建立一個 `clusterrolebinding`：

```
kubectl create clusterrolebinding drone-admin \
  --clusterrole=cluster-admin \
  --serviceaccount=default:drone
```

圖 8-1 展示的就是一個被 gpu=true:NoSchedule 給 taint 過的節點。Pod Spec 1 裡有一個 toleration 鍵，其值為 gpu，因此它會接受 tainted 節點調度進入。反觀 Pod Spec 2，其 toleration 鍵值為 no-gpu，因此它不會被調度進入此一節點。

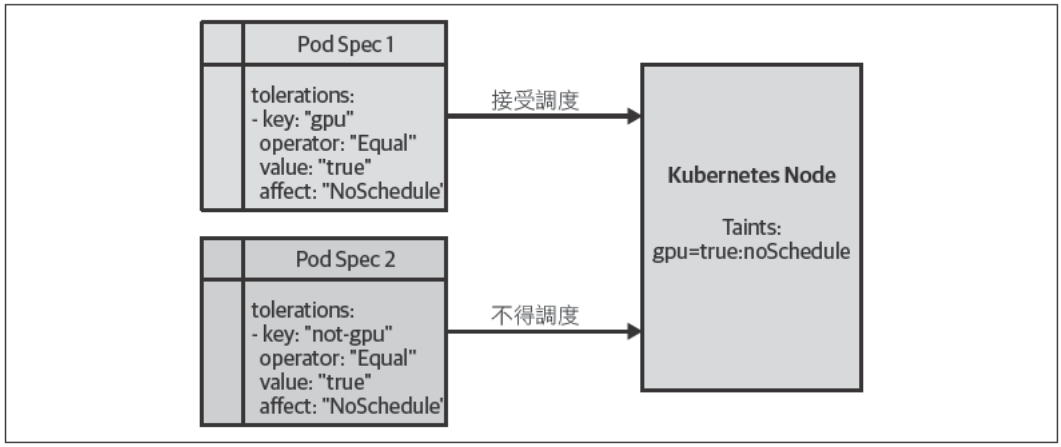

圖 8-1　Kubernetes 的 taints 和 tolerations

當 pod 因節點為 tainted 而無法調度進入時，你會看到以下的錯誤訊息：

```
Warning: FailedScheduling 10s (x10 over 2m) default-scheduler 0/2 nodes are
available: 2 node(s) had taints that the pod did not tolerate.
```

現在，你已經知道如何手動為節點加上 taints 以影響調度。此外，尚有一種名為 *taint-based eviction*（按照 *taint* 的狀態驅逐）的有力概念，可以藉其驅逐運行中的 pods。舉例來說，如果一個節點因為磁碟損壞而變得不健康，按照 taint 狀態驅逐的方式就會把主機上的 pods 重新調度至叢集中其他健康的節點。

Pod 的資源管理

在 Kubernetes 中管理應用程式，最要緊的面向之一就是適度地管理 pod 的資源。Pod 的資源管理涵蓋 CPU 與記憶體，你必須將 Kubernetes 叢集的整體使用率最佳化。這些資源可以從容器的層面管理、也可以從命名空間的層面管理。當然還有其他像是網路和儲存之類的資源，但 Kubernetes 尚未發展出對這類資源設置請求和限制的方式。

第九章

網路、網路安全和服務網格

在以相互連結的系統所構成的叢集裡，Kubernetes 可以有效地管理分佈其中的分散式系統。因此這些互連的系統彼此間如何溝通的重要性顯然會大為增加，這時網路就成了關鍵因素。要能有效地運用服務間的通訊，就必須先了解 Kubernetes 是如何簡化它所管理的分散式服務之間的通訊。

本章會專門介紹 Kubernetes 的網路原則，以及如何在各種情況下運用這些觀念的最佳實務做法。凡是討論到網路，就　定離不開安全性。傳統上，由網路層控制的網路安全邊界模型，在 Kubernetes 的嶄新分散式系統世界中也仍然存在，只不過其實作方式和功能略有變動。Kubernetes 引進了原生的網路安全策略 API，與傳統的防火牆規範有異曲同工之妙。

本章的最後　節特別鑽研了新穎又令人畏懼的服務網格。雖說我們用了「畏懼」　詞，但服務網格技術對於 Kubernetes 而言，其實仍是　個新興的領域。

Kubernetes 的網路原則

了解 Kubernetes 如何利用底層網路來簡化服務間通訊，是有效規劃應用程式架構的關鍵。通常網路題材都會讓大多數人感到頭疼。我們在說明時會儘量保持簡化，因為到頭來本書不過只是最佳實務指南，而非容器網路課程。幸好 Kubernetes 已經立下了若干網路規範，有助於我們起步。這些規範詳列了不同元件間的通訊方式。我們這就來詳細地檢視每個規範：

同一 *pod* 中的容器間通訊

所有位於相同 pod 中的容器都共用一樣的網路空間。因此容器間只需使用本機（localhost）就能有效地通訊。它也意味著同一個 pod 中的容器必須以各自不同的通訊埠對外公開。這需要用到 Linux 的命名空間（namespaces）和 Docker 的網路功能，透過每個 pod 中都有的一個停滯（paused）容器來專門運作 pod 網路功能，才能讓這些容器處在同一個局部網路上。圖 9-1 展示的就是容器 A 如何只靠 localhost 和容器 B 聆聽的通訊埠號，就能與容器 B 直接通訊。

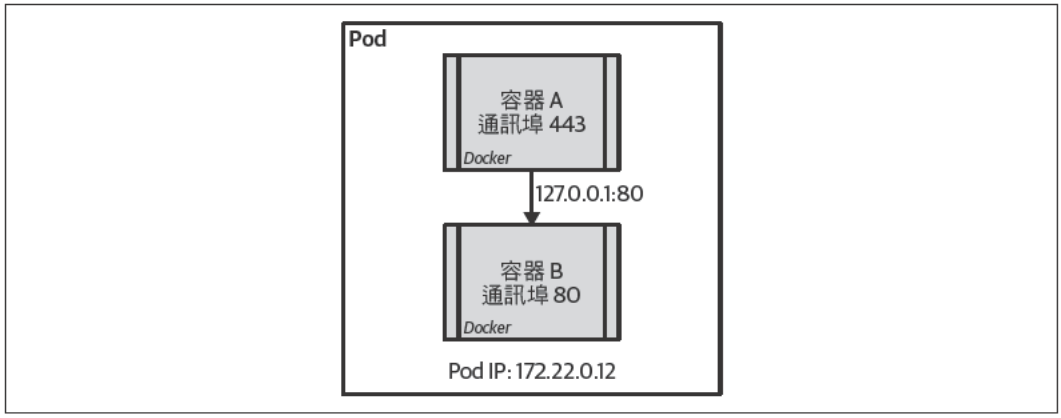

圖 9-1　在 pod 內的容器間通訊

Pod 對 *pod* 之間的通訊

所有的 pods 彼此通訊時都不需用到網路位址轉換（network address translation, NAT）。亦即發送端 pod 為接收端 pod 所知的 IP 位址，就是發送端實際的 IP 位址。這有幾種不同的處理方式，端看用的是哪一種網路植入元件（plug-in）而定，這在本章稍後會詳細解說。此一規範對於位在同一節點內的 pod 之間、以及位在同叢集但不同節點的 pod 之間，都一樣有效。這同時也延伸到可以不透過 NAT、直接與 pod 溝通的節點。必要時這還可以讓主機代理程式（host-based agents）或系統服務（system daemons）直接與 pod 通訊。圖 9-2 呈現的便是在相同節點的 pods 之間、以及位於叢集中不同節點的 pods 之間的通訊過程。

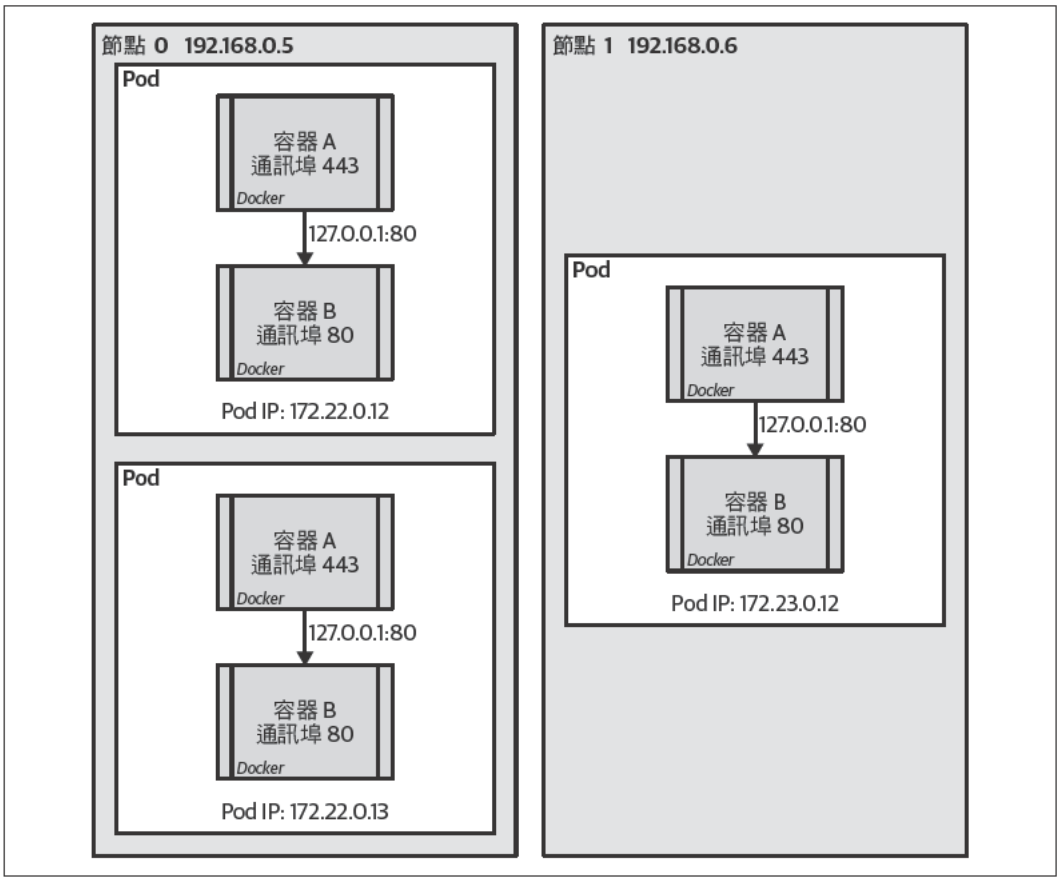

圖 9-2　在節點內外的 Pod 對 pod 的通訊

服務對 *pod* 的通訊

Kubernetes 裡的服務，係以　個持續存在的 IP 位址和各節點的通訊埠來呈現，該通訊埠會把所有的流量轉給服務所對映（mapped）的端點。隨著 Kubernetes 的演變，其偏好的實作方式也不斷在變動，但主要的兩種方式，就是透過 iptables、或是較新穎的 IP 虛擬伺服器（IP Virtual Server, IPVS）。今日大部分的實作皆以 iptables 為大宗，它會在每個節點啟用　個虛擬的第 4 層負載平衡器。圖 9-3 就是如何以標籤選擇器（label selectors）將服務和 pods 結合的示意圖。

案當中，這些情況大多沒有自動平衡負載的設定。若要從叢集外部直接取用服務，只需使用 NodeIP:NodePort 就可以做到，如同圖 9-5 所示。

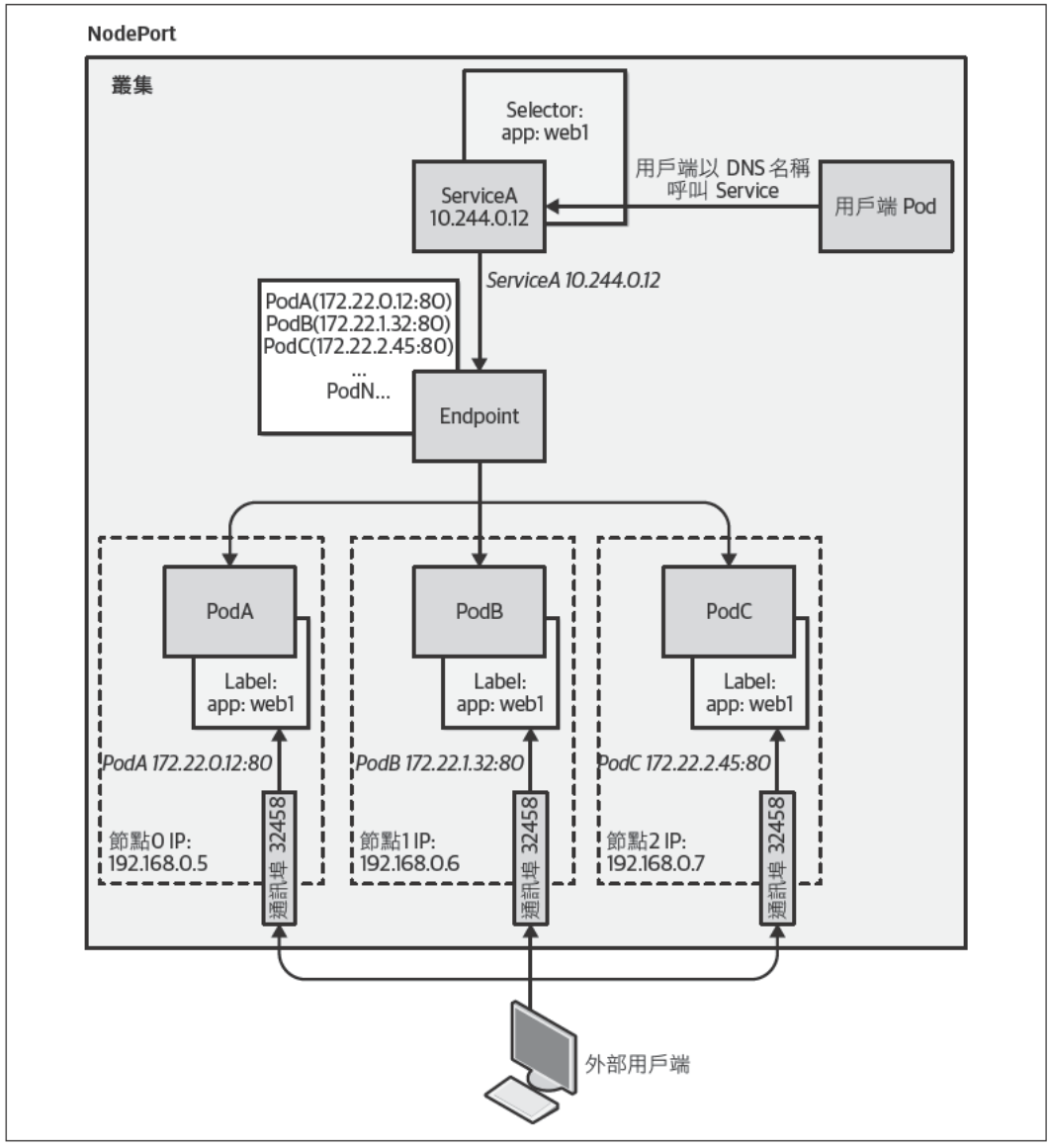

圖 9-5　NodePort　　Pod、服務及主機網路的視覺化示意圖