2

如何以 Docker 容器封裝並執行應用程式

Docker 可以將基礎服務的邏輯觀點縮減為三種核心元件：主機、容器和映像檔。主機負責運作容器，而後者其實是一個封閉的應用程式執行個體。容器始於映像檔，而映像檔則是封裝好的應用程式。Docker 容器映像檔的概念十分簡單，它其實就是一個單一個體，內含一個完整的、自給自足的應用程式。映像檔格式十分有效率，而映像檔與執行階段（runtime）之間的整合是十分巧妙的，因此要有效地運用 Docker，首要之務便是要掌握映像檔。

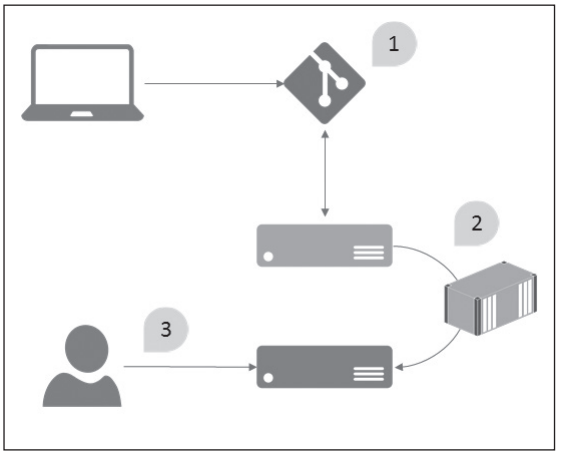
你已在**第 1 章** *Docker on Windows 初探*裡見識到若干映像檔，同時也運行了一些基本的容器來檢查你安裝的 Docker 是否可以正常運作，但是大家都還沒仔細觀察過映像檔，也不知道 Docker 如何運用映像檔。本章可以幫助你瞭解 Docker 的映像檔：包括學習映像檔的架構、以及 Docker 如何運用它們，還有如何將你自己的應用程式封裝成 Docker 映像檔。

首先要搞清楚的是映像檔和容器之間的差異，當你用同一個映像檔運行各種不同類型的容器時，就可以很容易地辨別出來。

本章可以幫助您學到：

- 從映像檔運行容器
- 利用 Dockerfiles 建置映像檔
- 把自己的應用程式封裝成 Docker 映像檔
- 操作映像檔和容器裡的資料
- 將傳統的 ASP.NET 網頁應用程式封裝成 Docker 映像檔

碁峰 www.gotop.com.tw



在這個管線裡，開發者把他們更改過的程式碼上傳至中央原始碼倉庫 **(1)**。然後負責建置的伺服器會據以編譯應用程式，並執行單元測試，如果通過測試，就會再建置容器映像檔，並配置到準正式（staging）環境 **(2)**。接著我們會對這個準正式環境進行整合測試（integration tests）及點對點測試（end-to-end tests），如果都過關了，那麼最後這個版本的容器映像檔就是可以讓測試人員驗證的最佳候選釋出版本 **(3)**。

最後你會在正式環境中以這個映像檔運行容器，完成新版本的配置，而且可以肯定的是，整個應用程式堆疊絕對和通過所有測試的二進位檔案集合完全一致。

這種方式的缺點是，你得把應用程式的 SDK 裝到每一台建置用的代理端（build agents），而代理端彼此之間 SDK 版本及相關內容都要呼應開發者使用的環境。在 Windows 專案裡，通常你都可以看到裝有 Visual Studio 的 CI 伺服器，這就是要確保伺服器和開發者使用相同建置工具之故。也因此重大的建置伺服器尤其需要多花工夫照顧。

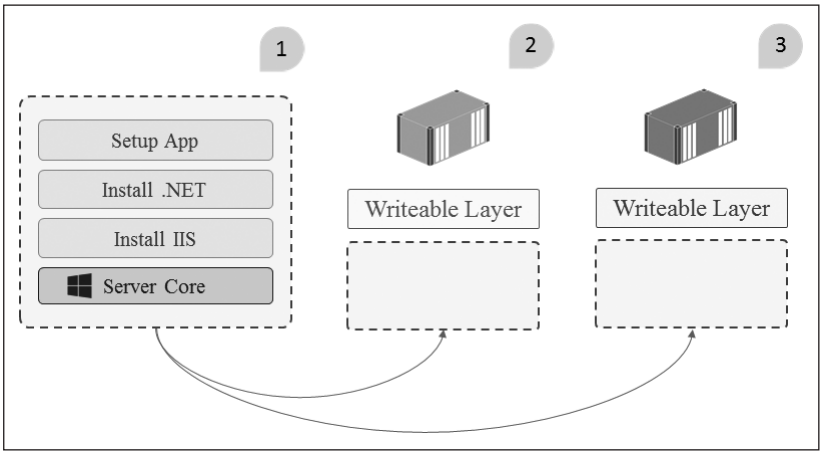
這也意味著，除非你自己的機器上裝有 .NET Core 1.1 SDK，不然就沒法自己建置這個瘦身過的 Docker 映像檔。

碁峰 www.gotop.com.tw

各層之中的資料和虛擬的 C 磁碟機

虛擬檔案系統是 Docker 將一組實體映像層結合，並將其視為單一邏輯容器映像檔的結果。映像層會以唯讀的方式掛載（mounted）成容器檔案系統的一部分，因此其內容是不能變更的，這也是它們可以安全地為許多容器所共用的方式。

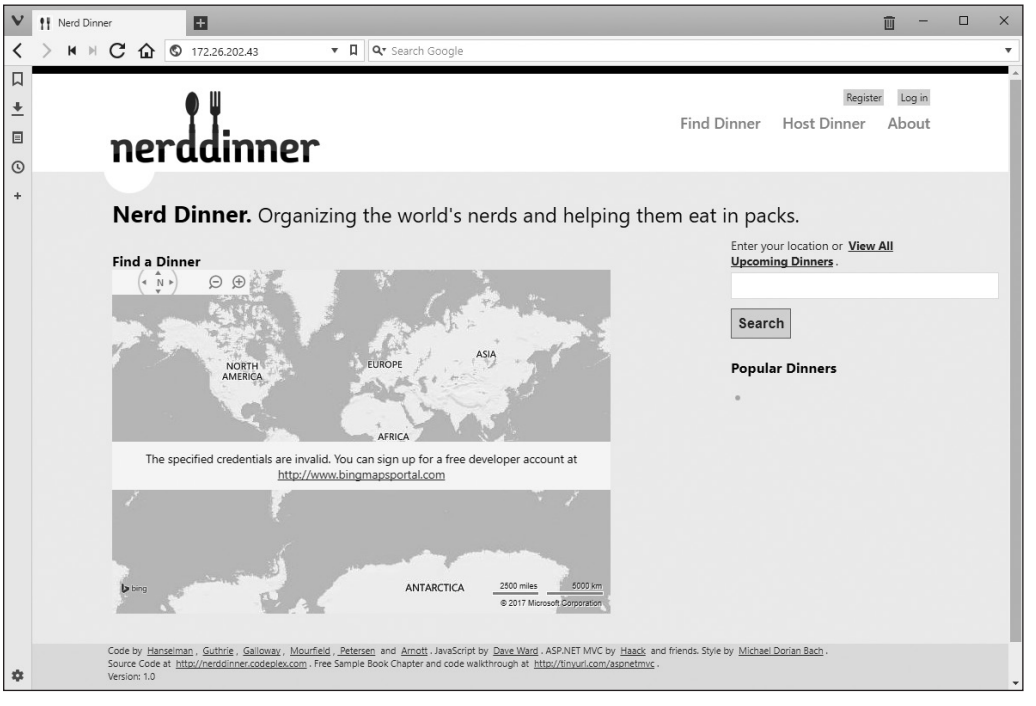
在所有的唯讀層上，每個容器都擁有自己的可寫入層，因此每個容器都能夠修改自己的資料，但不會影響到其他的容器：



上圖顯示了兩個來自同一個映像檔的容器。映像檔 (**1**) 實際上由許多層堆疊而成——每一層都來自 Dockerfile 中的一道指示語句。兩個容器 (**2** 和 **3**) 在運行時都使用了來自映像檔的相同底層，但各自擁有自己的可寫入層。

Docker 呈現給容器的是一個單一的檔案系統。對於容器來說，分層的概念和唯讀的基本層都是隱諱的，而容器讀寫資料時卻像是面對完整的原生檔案系統一般，而且只有一個磁碟機。如果你在建置 Docker 映像檔時也新建了一個檔案（位於映像層），隨後又在容器中編輯了這個檔案，Docker 實際上是在容器的可寫入層裡先複製一份這個變更過的檔案，然後把原本的唯讀檔案「藏」起來。這樣一來，容器編輯的其實是檔案的副本，但原本映像檔中的檔案其實是保持不變的。

碁峰 www.gotop.com.tw



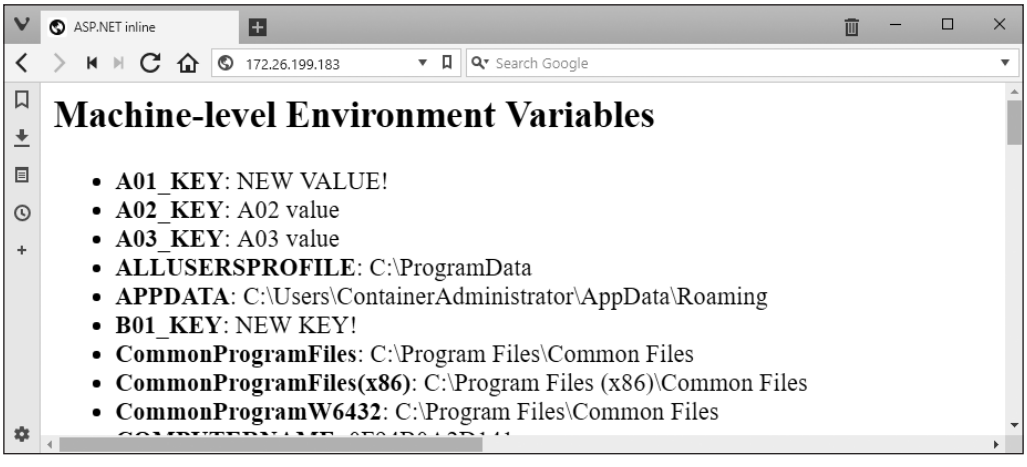
這時應用程式還不算是功能齊備（我不過是讓最基本的部分跑起來而已）。網頁中的 Bing Maps 物件尚未顯示出真正的地圖，因為我還沒把 API 金鑰放進去。API 金鑰在每個環境中都會變動（每個開發用環境、測試用環境和正式環境的金鑰都會不一樣）。你可以在 Docker 裡透過環境變數來管理環境組態，而在**第 3 章開發 *Docker* 化的 *.NET* 和 *.NET Core* 應用程式**的 Dockerfile 裡，我也會運用這個技巧。

如果你四處瀏覽這個特製的 NerdDinner 版本，並試著登錄一名用戶、或是搜尋一份餐點，就會看到一個黃色的受損頁面，通知你資料庫不存在無法使用。在它原本的格式中，NerdDinner 是以 SQL Server 的 LocalDB 做為輕量型資料庫，並將資料庫檔案存放在應用程式目錄裡。當然我也可以把 LocalDB 的執行平台安裝到容器映像檔裡，但那樣一來就不符合 Docker 的「一個容器對應一個功能」的宗旨了。相反地，我要為資料庫另建一個分離的映像檔，以便讓它自己在自己的容器中獨立運行。

下一章還是會以 NerdDinner 為例反覆說明，加上環境變數，並以分離元件的方式讓 SQL Server 以自己的容器運行，同時向各位展示，如何透過 Docker 平台，著手將傳統的 ASP.NET 應用程式進行現代化改裝。

碁峰 www.gotop.com.tw

再次瀏覽容器的 IP 位址，各位應該注意到 ASP.NET 已經把新的變數值寫出來了：



現在我已把對於 Docker 環境變數管理的支援加到 IIS 映像檔裡了，因此 ASP.NET 應用程式就可以利用 `System.Environment` 這個類別（class）來讀取組態設定值。此外我在同一個映像檔裡也保留了 IIS 的日誌複寫功能，所以現在這是一個奉公守法的 Docker 成員——你可以透過 Docker 來設定應用程式，同時檢查其日誌了。

最後一個要改善的，就是要讓 Docker 知道如何監視容器中運行的應用程式，這樣 Docker 才能判斷應用程式是否健康，也知道在它不健康時應採取何種因應措施。

建置會監視應用程式的 Docker 映像檔

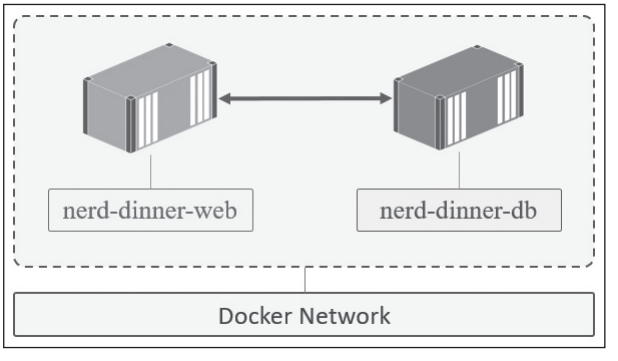
一旦我替 NerdDinner 的 Dockerfile 加入了上述的新功能（即日誌管理與環境變數），並從加強版的 Dockerfile 製做出的映像檔運行容器，我就能透過 `docker container logs` 指令看到網頁請求和回覆的日誌紀錄，因為這個容器會把所有 Docker 攔截到 IIS 日誌紀錄都轉發出來，而且我也能利用環境變數來指定資料庫的使用者身份。這樣一來，就像面對 Docker 上運行的任何容器化應用程式般，我可以用相同的方式運行與管理老舊的 ASP.NET 應用程式。但我還可以再加上一個 Docker 設定，以便讓它為我監視容器狀態，進而掌控任何意料外的故障狀況。

Docker 有能力監視應用程式的健康狀態，而不僅僅是檢查其程序是否仍在執行而已，做法就是在 Dockerfile 裡加上 `HEALTHCHECK` 指示語句。你只需藉由 `HEALTHCHECK`，就可以指揮 Docker 如何測試出應用程式健康與否。其語法和 `RUN` 以及 `CMD` 等指示語句相仿

碁峰 www.gotop.com.tw

從應用程式容器連接資料庫容器

Docker 平台內建自己的 DNS server，容器彼此間就利用它來尋找服務來源。我在啟動 NerdDinner 資料庫容器時順便指定了一個名稱，這樣位在同一個 Docker 網路上的任何其他容器就可以藉由名稱取用該容器——就像網頁伺服器透過 DNS 主機名稱取用遠端資料庫伺服器一樣：



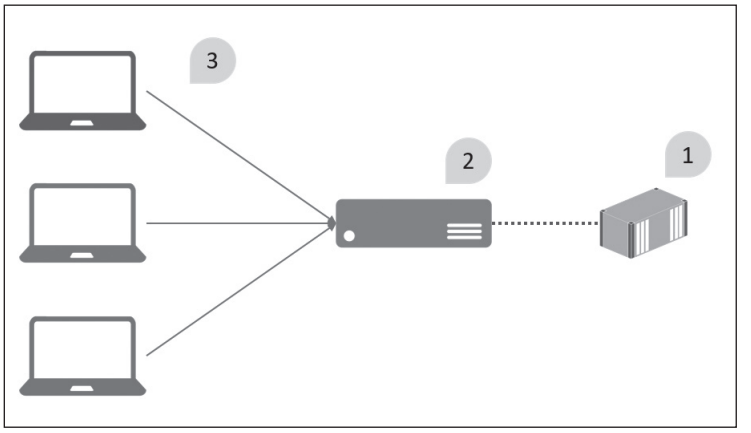
這種做法使得應用程式組態比傳統的分散式解決方案更為簡化。因為每個環境的配置看起來都一樣——不論是在開發環境、品管驗證還是正式環境，網頁容器和資料庫連線時使用的主機名稱永遠都會是 `nerd-dinner-db`，因為它其實存在於容器中。但是它可以和網頁容器位於同一個 Docker 主機，也可以放在 swarm 叢集的另一台機器上，應用程式是不知道其中區別的。

Docker 裡的服務尋找功能並不限於容器本身。容器也可以透過主機名稱取用同一網路上其他主機的服務。你可以用容器運行網頁應用程式，但連接的 SQL Server 則是位於一台實體機器上，資料庫並不一定也要是容器的形式。

每個環境中只有一部分組態會略有差異，就是 SQL Server 的登入身份。在 NerdDinner 的資料庫映像檔裡，我使用了一個環境變數，並指定一個預設值給它，藉以設置管理員密碼，而我在網頁應用程式的容器中也運用了類似的方式。資料庫的連線字串（connection string）位在 `Web.config` 檔案中，其中包括了預期連線的主機名稱和使用者 ID，但在密碼的部分則是以借位字符（placeholder）來代替：

```
Data Source=nerd-dinner-db,1433;Initial Catalog=NerdDinner;User
Id=sa;Password={SA_PASSWORD}
```


碁峰 www.gotop.com.tw



登錄所運行在伺服器 (**2**) 的容器上 (**1**)。用戶端的機器 (**3**) 連到伺服器，以便使用本地端登錄所來上傳或下載映像檔。

為了能連到登錄所容器，你必須開放容器的 `5000` 號通訊埠，並對映到 Docker 主機 `5000` 號通訊埠。登錄所用戶只需透過主機的 IP 位址或名稱就可以連進容器，而這部分就會是你用在倉庫名稱中的登錄所網域名稱。你也許還會想要從主機掛載一個卷冊，以便將映像檔資料存放在某個已知位置。就算你替登錄所更新了新版容器，它還是會繼續以主機的網域名稱形式存在，而且先前的登錄所容器所儲存映像檔的每一層它都還可以繼續沿用。

在我的主機上有一個 RAID 陣列設定成磁碟機 `E:`，我就用它來存放登錄所的資料，所以當我運行登錄所容器時，我會這樣對應卷冊：

```
mkdir E:\registry-data
docker container run -d -p 5000:5000 -v E:\registry-data:C:\data
dockeronwindows/ch04-registry
```

TIP

卷冊會對應到 `C:\data` ——也就是已經存在容器中的、別名 `G:` 的磁碟機。

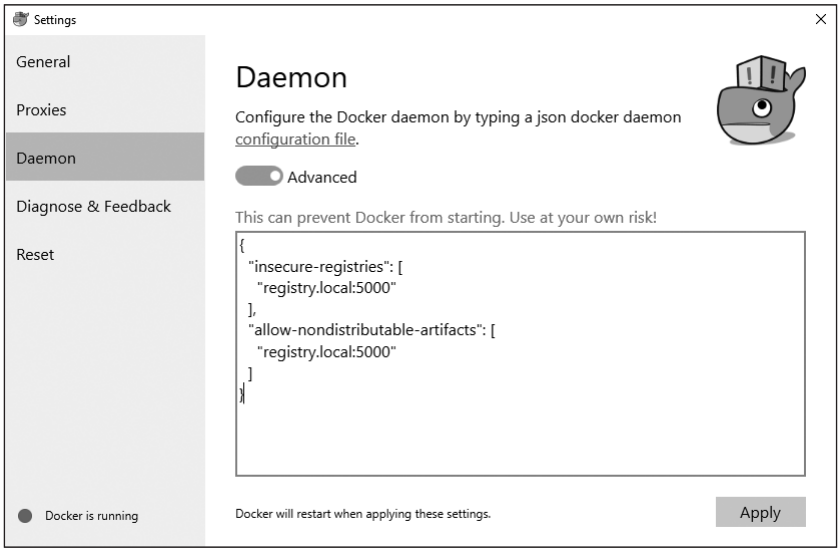
碁峰 www.gotop.com.tw

地端登錄所下載映像檔時，就只需經過快速的區域網路，不必老遠地跑到網際網路上去找。

要達成這一點，必須在 Docker 組態檔裡針對特定登錄所啟用這個功能，亦即修改 `allow-nondistributable-artifacts` 區塊：

```
{
  "insecure-registries": [
    "registry.local:5000"
  ],
  "allow-nondistributable-artifacts": [
     "registry.local:5000"
  ]
}
```

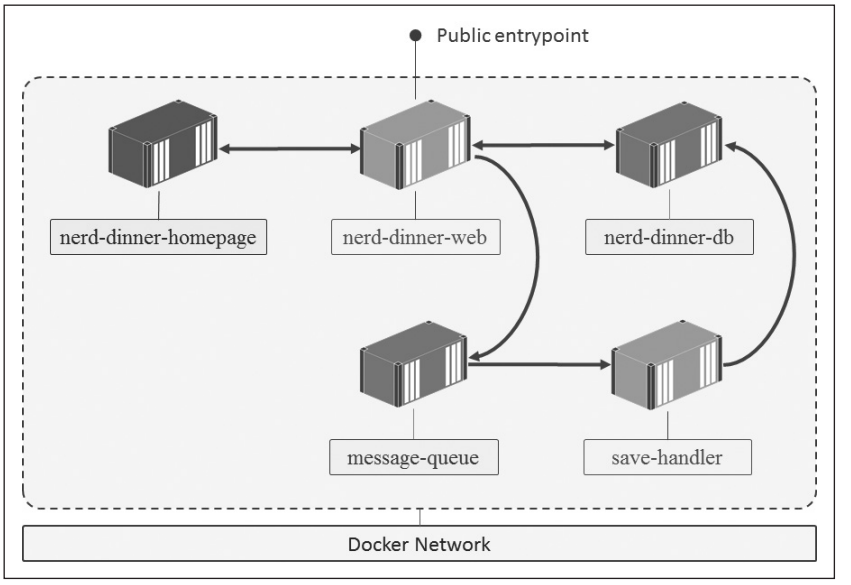
在 Docker for Windows 的 UI 裡，這個設定並未直接顯示，但你可以在設定畫面的 **Advanced** 模式裡設定它：



現在我可以把 Windows 的 *foreign layers* 上傳到本地端登錄所了（先前會有 *Skipped foreign layer* 訊息的，記得嗎？）。我可以用我自己的登錄所網域名稱來標記最新版的 Nano Server 映像檔，並上傳到自家登錄所：

```
PS> docker image tag microsoft/nanoserver:10.0.14393.1358
registry.sixeyed:5000/microsoft/nanoserver:10.0.14393.1358
```


碁峰 www.gotop.com.tw



如果我們可以一次只專注在一個功能上，就能迅速地實施變動。這種設計方式全無以上其他方式的缺點：

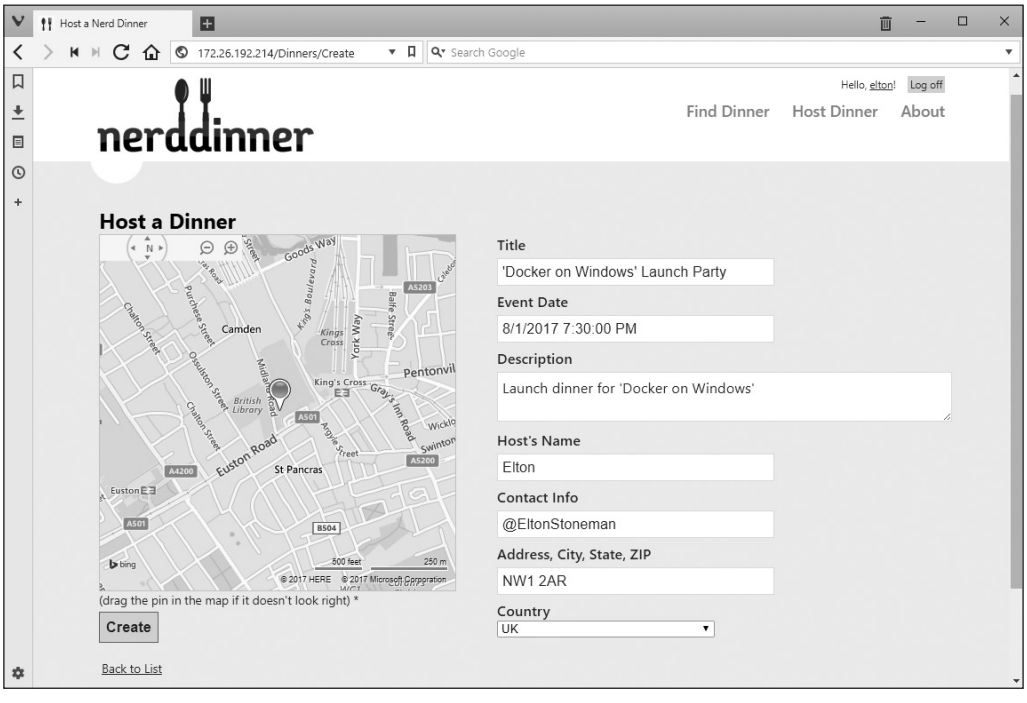
- 這是有目標的改寫，而且主應用程式裡一次只更改一個控制器動作
- 新的訊息處理器（message handler）元件既小巧又緊密，因此很容易測試
- 網頁層和資料層的耦合關係被拆開，因此便於各自獨立延伸
- 我把工作從網頁應用程式中分離出來，這樣當然有助於改善效能

好處還不僅於此。新元件與原本的應用程式是完全分開的，它只需傾聽事件訊息，然後對此做出回應就好。你可以採用 .NET、.NET Core 或任何其他技術堆疊來擔任訊息處理器；不必受制於單一堆疊。而且應用程式還可以繼續發出各種事件，這樣就有機會稍後再加上傾聽這些事件的新處理器，藉此添加新功能。

把 NerdDinner 的組態 Docker 化

NerdDinner 使用 `Web.config` 檔來管理組態——它不但控制各版本之間保持一致不變的應用程式組態值、也控制那些會隨著不同環境而變化的環境組態值。由於組態檔深植在釋出的套件之中，導致它十分難以更改。但在**第 3 章開發 *Docker* 化的 *.NET* 和 *.NET Core* 應用程式**裡，我設法另闢蹊徑，利用 Dockerfile 裡的起始指令稿和

碁峰 www.gotop.com.tw



一旦我送出表單，網頁應用程式就會對佇列發出事件訊息。這個動作的負擔極小，因此網頁應用程式幾乎立刻就可以完成動作並回應使用者。傾聽訊息的則是運行在另一個容器中、新撰寫的控制台應用程式（就是訊息處理器）——也許執行在另一台主機上。它會撿起訊息並加以處理。處理器會把動作紀錄下來並傳給 Docker 控制台，因此管理員就能用 `docker container logs` 指令監視訊息處理器容器裡的動作：

```
> docker container logs nerd-dinner-save-handler

Connecting to message queue url: nats://message-queue:4222
Listening on subject: events.dinner.created, queue: save-dinner-handler
Received message, subject: events.dinner.created
Saving new dinner, created at: 6/24/2017 8:44:21 PM; event ID: b7ecb300-
af6f-4f2e-ab18-19bea90d4684
Dinner saved. Dinner ID: 1; event ID: b7ecb300-af6f-4f2e-ab18-19bea90d4684
```

建立用餐資訊的外觀功能還是一樣的——使用者輸入的資料仍會寫到 SQL Server 裡，而且使用者的感受是不變的，但功能背後的延展性卻已大為不同。容器化設計讓我得以把抽出的固有程式碼放到新元件裡，而且心裡很清楚該元件可以像既有解決方案一

碁峰 www.gotop.com.tw

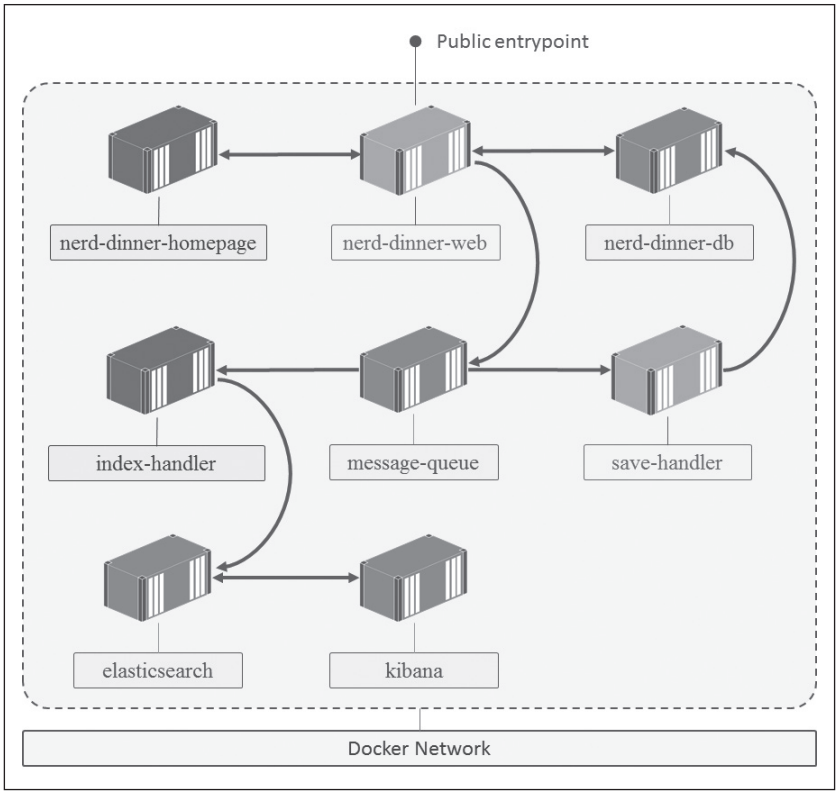
TIP

Elasticsearch 和 Kibana 都是十分強悍的軟體系統。筆者在此不會對它們多所著墨，但它們確實十分受歡迎，線上資源也極為豐富，各位不妨自行鑽研。

從單一整體到分散式解決方案

現在，NerdDinner 已經從老舊的單一整體形式搖身一變，成為能夠輕易延伸和擴展的解決方案，而且採用現代化的設計風格，以現代化的應用程式平台運行。這是一個既快速又低風險的演變，背後則有 Docker 平台和容器優先式設計的支援。

此專案一開頭只試著把整個 NerdDinner 紋風不動地搬到 Docker 上，並且用容器運行網頁應用程式和 SQL Server 資料庫。現在則已分解成八個元件，每個都以自己的輕型 Docker 容器運行，而且都可以個別地部署，因此就可以擁有自己的釋出步調：



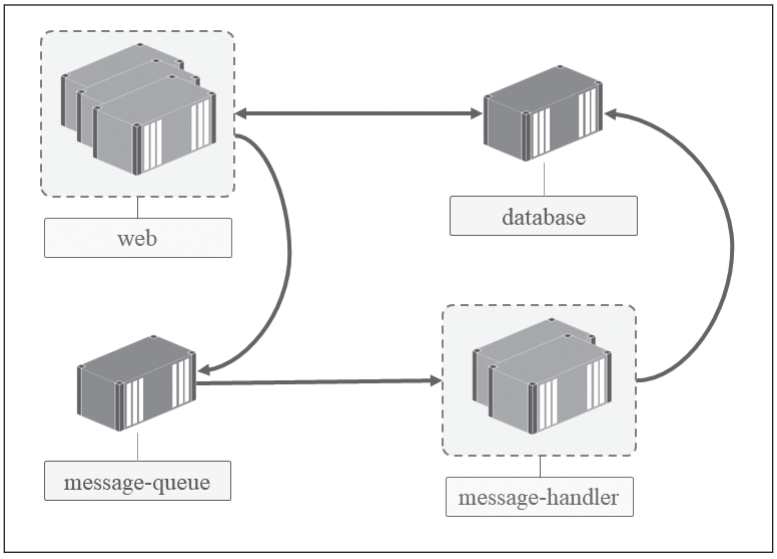
碁峰 www.gotop.com.tw

Docker Compose 預設會做為 Docker for Windows CE 版的一部分安裝。但如果你使用 PowerShell 安裝工具來安裝 Docker，就不會有 compose 可用。你得自行下載 GitHub 在 `docker/compose` 釋出的內容[1]。

以 Docker Compose 定義應用程式

Docker Compose 的檔案格式簡單之至。YAML 其實是一種具備可讀性的 JSON 文件集合（superset），而 Compose 的檔案規格則採用描述式的屬性名稱。在 Compose 檔案裡，你會定義構成應用程式的服務、網路和卷冊。網路和卷冊的概念都和原本 Docker 引擎所使用的相同。服務則是位在容器之上的抽象層。

容器其實是某個元件的單一執行個體，但是服務卻可以由多個執行個體組成，而且這些執行個體都是從同一元件運行起來的一個個容器。你可以在網頁應用程式的服務裡運行三個容器，然後再在訊息處理器的服務裡運行兩個容器：



譯註 1　亦可參照 https://docs.docker.com/compose/install/

碁峰 www.gotop.com.tw