12 在不同平台上運行容器

CPU架構有兩種主要類型：英特爾和手臂。並且操作系統有兩種主要類型：Linux和Windows。應用程序是為特定平台構建的，並且無法便攜 - 為Linux在ARM上構建了您的應用程序，並且它將在Intel機器上或Windows上不起作用。 容器圖像也是如此，並且可能導致很多複雜性和混亂。 用戶可能會發現圖像在其（Intel）桌面上正常工作，但在其（ARM）MacBook上失敗。幸運的是，Docker已經使用了*multi-platform images:*的外觀和表現像普通的Docker映像已經涵蓋了很長時間，但是它們具有多種變體，每個變體都針對不同的平台，但都使用相同的圖像名稱。 當您從多平台圖像運行容器時，Docker會為您的CPU和OS拉匹配變體。 如果您在另一個平台上使用相同的圖像名稱，則將獲得不同的圖像變體，但它將是同一應用程序，並且它將以相同的方式工作。對於用戶來說，這是一個超級令人放鬆的工作流程，但是對於圖像發布者來說，這需要一些精力。

在本章中，您將學習各種生產多平台構建的方法，但是如果您考慮不使用Windows或Arm，則認為跳過此方法，至少應該閱讀第一部分以了解為什麼這是一種改變遊戲規則的選項。

12.1為什麼您需要支持多個平台

Docker多年來一直支持ARM CPU，但這是一個利基市場，直到蘋果開始建造基於手臂芯片組的處理器，AWS釋放了雲VM，該雲VM的ARM上的ARM比Intel等價物便宜40%。 然後，開發人員意識到，如果他們想在新的Mac上運行自己喜歡的Docker容器，並且管理員想要一種簡單的方法來削減云成本，則必須深入研究多平台圖像。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

Docker Hub上的官方圖像都是多平台的，現在嘗試在本書中練習的所有圖像也是如此。用戶可能不會意識到這一點，因為它們的容器只是在使用的任何平台上工作。但是您需要了解其工作原理，以便您可以支持使用平台混合的團隊，或者是否想部署到雲中更便宜的機器（Azure和Google Cloud也具有ARM選項）。為此，您需要使用ARM和Intel機器構建自己的圖像的變體。這是Docker解決的困難部分，因為生產構建了農場和開發人員的工作流程。 Docker Desktop支持仿真構建Docker圖像並用臂在英特爾機器上運行容器，反之亦然。

現在嘗試

Docker Desktop與開源QEMU模擬器預先配置 - 如果您在Linux上使用Docker Engine，則可以自己安裝QEMU（搜索Tonistiigi/binfmt），如果您在Windows上，請確保使用Linux容器模式。打開終端並使用平台標誌仿真構建圖像：

```
＃切換到練習文件夾：CD CH12/練習


＃為64位臂構建：
Docker Build -T Diamol/ch12-Hello：2e-linux-arm64-平台linux/arm64 ./hello

＃和64位Intel：Docker Build -T Diamol/ch12-Hello：2e-linux-amd64-platform linux/amd64 ./hello


＃檢查引擎的本機體系結構：docker info -f'{{.ostype}}}/{{
。 arthitection}}'

＃和圖像的體系結構：
Docker Image檢查Diamol/ch12-hello：2e-linux-arm64 -f'{{.os}}}}/{{。 {。架構}}'

Docker Image檢查Diamol/ch12-hello：2e-linux-amd64 -f'{{.os}}}/{{。 {。架構}}'
```

您可以構建針對64位ARM平台和64位Intel平台的Linux圖像，即使您自己的機器只是其中之一。 該圖像使用多階段Dockerfile來編譯和包裝GO應用程序。 GO工具是多平台，並且Dockerfile中的基本圖像（對於SDK和運行時）具有ARM和Intel變體。這就是您支持跨平台構建所需的全部。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

您可以將這些圖像推到註冊表上，並在真實的手臂或英特爾機器上從中運行容器，它可以正常工作。您可以在圖12.1中看到我的輸出，在該圖12.1中，我從英特爾機器中構建了一個手臂圖像：

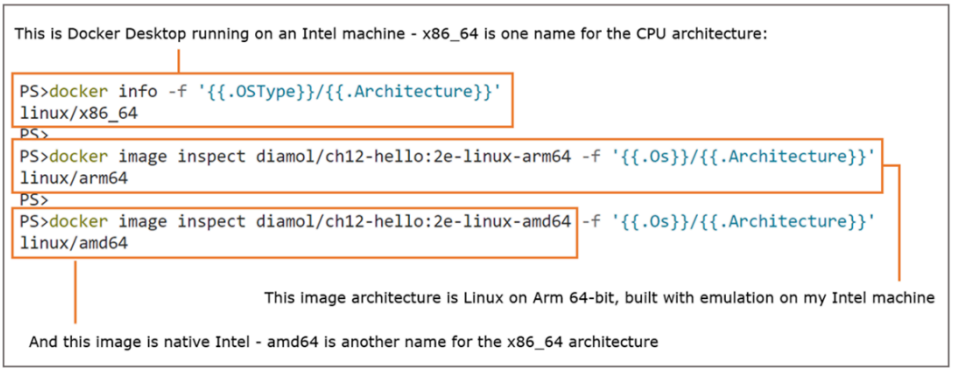


圖12.1在Docker桌面中使用仿真來構建不同平台的圖像

這不是多平台圖像，它的兩個圖像具有不同的標籤。用戶需要選擇正確的標籤才能在其平台上運行容器。Docker了解您的機器，包括操作系統和CPU體系結構，當您嘗試繪製圖像時，它將使用它們作為匹配。 拉動圖像不僅僅是下載圖層，還優化了以擴展壓縮層並準備運行圖像。 僅當要使用的圖像是您正在運行的體系結構的匹配時，該優化僅有效，因此，如果沒有匹配，則會遇到錯誤 - 您甚至無法將圖像拉動以嘗試運行容器。

現在嘗試

您可以使用運行Linux容器的任何Docker引擎來驗證這一點 - 嘗試下載Microsoft Windows Image：

```
#拉動Windows Nano服務器圖像：
Docker Image Pull mcr.microsoft.com/windows/nanoserver：ltsc2022
```

您可以在圖12.2中看到我的輸出 - Docker獲取當前引擎的OS和CPU，並檢查註冊表中是否有匹配的變體。沒有匹配，因此沒有拉動圖像，我會遇到錯誤。

© Manning Publications Co. To comment go to liveBook
許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

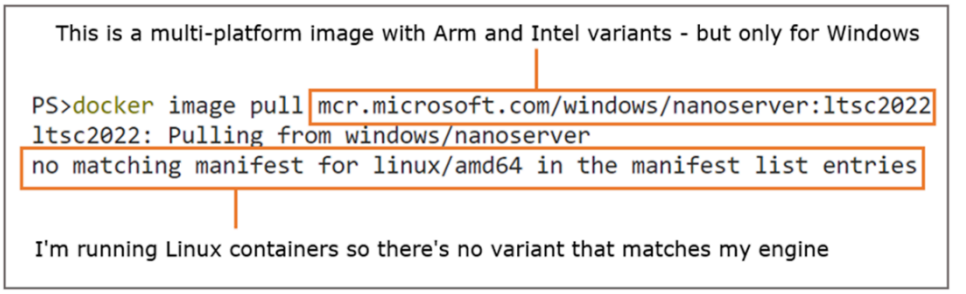


圖12.2如果沒有可匹配的操作系統和CPU，則無法從註冊表中獲取圖像。

*manifest list*是圖像的變體集。 Windows Nano Server映像不是真正的多平台，它將在不同版本的Windows上運行，但是清單列表中沒有Linux變體。 基本原則是，圖像的平台必須匹配引擎的平台，但是有一些細微差別 - 可以將linux映像用於非匹配的CPU架構，但是容器會因晦澀難懂的“用戶流程引起的'execor格式錯誤”消息而失敗。一些較舊的Windows引擎在Windows（LCOW）上具有一個稱為Linux容器的實驗功能，因此它們可以運行Linux容器（但是複雜的應用程序會隨著更晦澀的錯誤而失敗）。 最好堅持使用引擎的匹配體系結構，並且多平台圖像可讓您根據需要將圖像定制為每個OS和CPU。

12.2構建多平台圖像變體

您有幾種選擇用於不同平台的圖像變體的選項。在首先，您按照本章練習中的Hello應用程序的示例：編寫一個多階段Dockerfile，該dockerfile從源中編譯該應用程序並將其包裝在容器中運行。您可以在不同的平台上構建該Dockerfile，如果您用於SDK的圖像和運行時支持您想要支持的所有架構，那麼您就可以了。

這種方法的巨大好處是您擁有一個Dockerfile，並在不同的機器上構建它以獲取平台變體。我使用這種方法為.NET堆棧構建自己的金色圖像；圖12.3顯示了SDK的方法。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

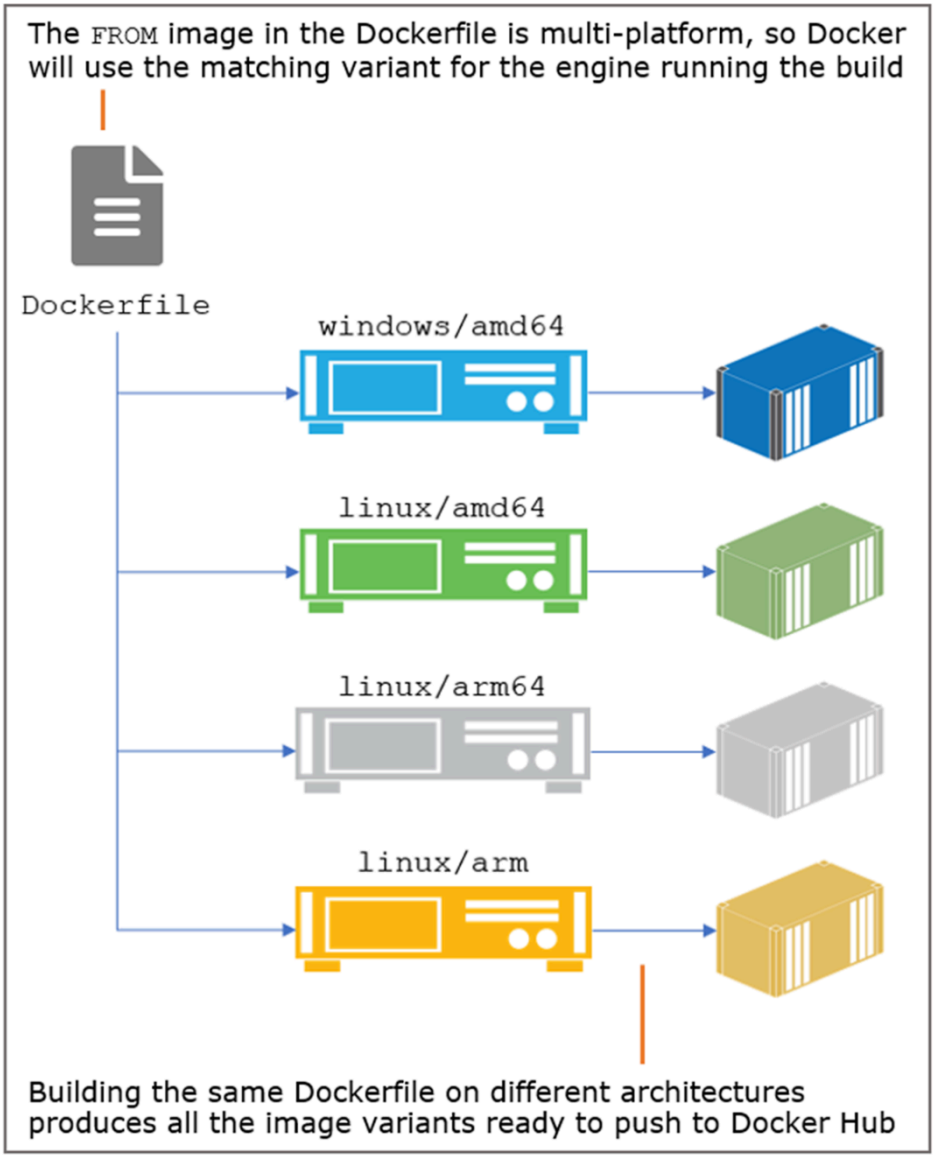


Figure 12.3 Use a multi-stage Dockerfile based on multi-platform images to build multiple image variants

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

如果您的源圖像不是多平台，或者不支持您要支持的所有平台，則無法遵循此方法。 Docker Hub上的官方圖像是多平台，但並不全部支持所有變化。 在這種情況下，您需要不同的Dockerfiles，這些Dockerfiles與圖像不同，也許是Linux，另一個用於Windows，或者是ARM 32位和64位的其他碼頭。 這種方法需要更多的管理，因為您有多個Dockerfiles可以維護，但是它為您提供了更大的自由度適應每個目標體系結構的行為。我將這種方法用於Maven的黃金圖像（一種構建Java應用程序的工具） - 圖12.4顯示了堆棧。

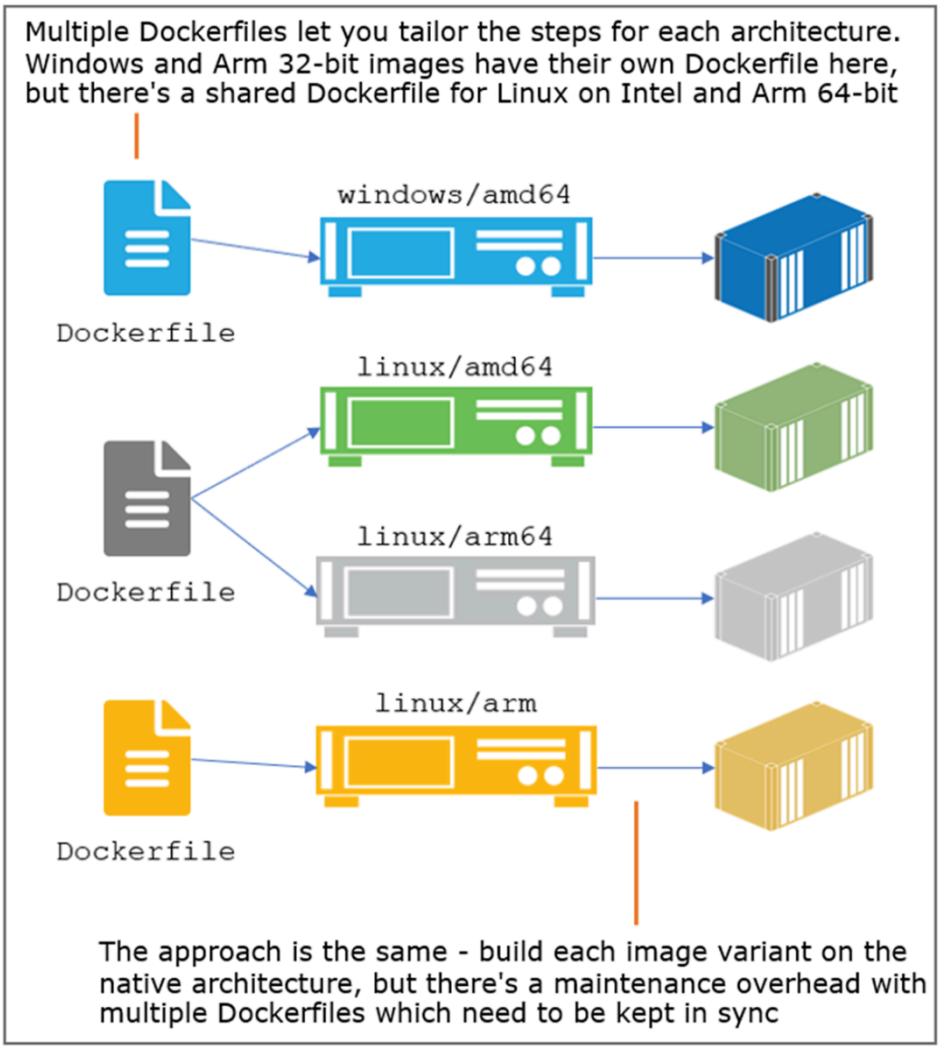


圖12.4您還可以使用為每個平台量身定制的Dockerfiles構建圖像變體。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

本章的練習具有一個非常簡單的文件夾列表應用程序，它只需打印有關運行時的一些基本信息，然後列出文件夾的內容。本書中支持的每個架構都有四個碼頭，其中一個：英特爾的Windows，Intel上的Linux，Linux在32位臂上，Linux在64位臂上。 您可以將Docker Desktop的CPU模擬與Linux容器一起構建和測試其中三個。

現在嘗試

使用每個平台的Dockerfile為不同平台構建圖像。 每個Dockerfile都略有不同，因此我們可以在運行容器時比較結果：

```
cd ./folder-list

＃為本機架構構建 - 英特爾/AMD：Docker Image構建-t diamol/ch12-folder-list：2e
-linux-amd64 -f ./dockerfile.linux-amd64-platform linux/amd64。


＃為64位的手臂建造：
Docker Image Build -T Diamol/ch12-folder-list：2e-linux-arm64 -f ./dockerfile.linux-a
rm64-platform linux/arm64。

＃和手臂32位
Docker Image Build -T Diamol/ch12-folder-list：2e-linux-arm -f ./dockerfile.linux- amm-platform linux/arm
。

＃運行所有容器並驗證輸出：Docker容器運行 -  Platform Linux/amd64 diamol/ch12-folder-list：2e-linux-a
md64

Docker Container Run- Platform Linux/ARM64 DIAMOL/CH12-FOLDER-LIST：2E-Linux-ARM64

Docker容器運行 - 平台Linux/ARM DIAMOL/CH12折線列表：2E-Linux-Arm
```

容器運行時打印一些簡單的文本 - 一個硬編碼的字符串，說明他們應該使用的操作系統和體系結構，然後是操作系統報告的實際OS和CPU，然後是包含一個文件的文件夾列表。您可以在圖12.5中看到我的輸出。Docker在必要時使用仿真，因此在我的英特爾機器上運行ARM-32和ARM-64 Linux變體時，它使用臂仿真器。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

The container prints some static text, which is different in each Dockerfile, and then runs an OS command to print out the CPU architecture

```
PS>docker container run --platform linux/amd64 diamol/ch12-folder-list:2e-linux-amd64
Built as: linux/amd64
Linux abd3d3a2dd75 5.15.167.4-microsoft-standard-WSL2 #1 SMP Tue Nov 5 00:21:55 UTC 2024 x86_64 Linux
file.txt
PS>docker container run --platform linux/arm64 diamol/ch12-folder-list:2e-linux-arm64
Built as: linux/arm64
Linux 857c5a39e6c6 5.15.167.4-microsoft-standard-WSL2 #1 SMP Tue Nov 5 00:21:55 UTC 2024 aarch64 Linux
file.txt
PS>docker container run --platform linux/arm diamol/ch12-folder-list:2e-linux-arm
Built as: linux/arm32
Linux 03cfbd3f6ea3 5.15.167.4-microsoft-standard-WSL2 #1 SMP Tue Nov 5 00:21:55 UTC 2024 armv7l Linux
file.txt
```

Adding the `platform` flag ensures Docker will use emulation where it needs to - this is 32-bit Arm emulation on an Intel machine

圖12.5為特定平台構建的圖像可以使用Docker Desktop中的仿真來運行。

Linux變體的Dockerfile都非常相似，除了預期體系結構的硬編碼字符串。 Windows變體具有相同的行為，但是Windows具有不同的命令來打印輸出。 在這裡，每個體系結構的多個Dockerfile都會有用；我可以擁有完全不同的Dockerfile說明，但仍然可以得到相同的所需輸出。列表12.1比較了64位ARM Linux版本和64位Intel Windows版本的Dockerfile。

列表12.1 Linux和Windows Image變體的Dockerfiles

```
＃diamol /base的Linux：2e-linux-arm64 workdir /app copy file.t
xt。 CMD Echo“構建為：Linux /ARM64” && \ UNAME -A
&& \ ls /App /App＃Windows＃Windows＃從diamol /base：2e
-windows-amd64 workdir /app copy copy file.txt。 CMD ECHO
構建為：Windows/AMD64 && `echo%processor_architecture
%processor_ide
ntifier%&& `
dir /b c：\ app
```

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

每個版本都與圖像不同，該圖像特定於目標體系結構，而不是多平台圖像。 Windows DockerFile使用逃生關鍵字來更改線路斷開字符，將其更改為反向而不是默認的BackSlash，因此我可以在目錄路徑中使用後斜線。 沒有等同於Linux Uname命令的Windows，因此要打印CPU體系結構，我迴盪了Windows設置的一些環境變量。該功能大致相同，但是我可以採取不同的路徑到達那裡，因為這是一個特定於Windows的Dockerfile。

這是一個簡單的示例，但是如果要構建第三方應用程序的多平台版本，通常需要多個Dockerfiles。這本書的普羅米修斯和格拉法納金圖像是一個很好的例子。 項目團隊為我要使用的所有Linux變體發布了多平台圖像，但不適合Windows。因此，我有基於項目圖像和Windows Dockerfiles的Linux Dockerfiles，這些dockerfiles從Web下載中安裝了該應用程序。對於您自己的應用程序，擁有一個Dockerfile並避免額外的維護應該很容易，但是您需要小心，只需使用一個您知道在所有目標體系結構中工作的OS命令的子集。 很容易不小心包括一個在一個體系結構上不起作用的命令（例如CHMOD），最終會出現破裂的變體。

現在嘗試

文件夾列表應用程序還有另一個Dockerfile，這是多個平台DockerFile的嘗試。 它使用多平台圖像作為基礎，但混合了Linux和Windows命令，因此它構建的圖像將在每個體系結構上失敗：

```
＃構建多平台應用程序：Docker Image Build -T Diamol/ch12-fol
der-list：2e。

＃並嘗試運行它：
Docker Container Run Diamol/CH12折線列表：2E
```

您會發現構建成功完成，因此看起來您的圖像良好，但是每次運行時容器都會失敗。 您可以在圖12.6中看到我的輸出。 我同時運行了圖像的Linux和Windows版本，並且兩個容器都失敗了，因為CMD指令包含無效命令。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

This is the Linux image variant - the container fails because `dir` is not a Linux command

```
PS>docker container run diamol/ch12-folder-list:2e
Built as multi-arch
Linux e0231f84efde 5.15.167.4-microsoft-standard-WSL2 #1 SMP Tue Nov 5 00:21:55 UTC 2024 x86_64 Linux
/bin/sh: dir: not found
PS>
PS># switch to Windows containers
PS>
PS>docker container run diamol/ch12-folder-list:2e
Built as multi-arch
'uname' is not recognized as an internal or external command,
operable program or batch file.
```

And the Windows variant also fails, because `uname` is not a Windows command

圖12.6很容易構建在某些（或所有）平台上運行時失敗的多平台圖像。

牢記這一點很重要，特別是如果您使用複雜的啟動腳本。 如果您使用未知的OS命令，則運行說明將在構建時間失敗，但是CMD說明未經驗證，因此在嘗試運行容器之前，您不會知道該圖像已損壞。

在我們繼續推動變體並創建多平台圖像之前，這裡的最後一件事就是了解哪些架構Docker支持的支持以及您在開始使用它們時會遇到的各種奇怪的代號。 表12.1顯示了CPU的主要操作系統和架構組合和別名。

表12.1 Docker支持的體系結構，其代碼名稱

OS	CPU	Word Length	CPU Name	CPU Aliases
Windows	Intel/AMD	64-bit	amd64	x86_64, x64
Linux	Intel/AMD	64-bit	amd64	x86_64, x64
Linux	Arm	64-bit	arm64	aarch64, armv8
Linux	Arm	32-bit	arm	arm32v7, armv7, armhf

Docker支持更多的架構，但這些是您會發現的主要建築。 AMD64 CPU類型是Intel和AMD機器中相同的指令集，該指令為大多數台式機，服務器和筆記本電腦提供動力（Docker還支持32位Intel X86處理器）。 Apple Silicon是64位的臂，並且在手機，IoT設備和單板計算機中也發現了ARM CPU。 最著名的是Raspberry Pi，直到PI4釋放為32位，為64位。大型機用戶也沒有被排除在外 - Docker支持Linux的IBM CPU架構，因此，如果您的地下室中有IBM Z，Power或PowerPC機器，則可以將大型機應用程序遷移到Docker容器。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

12.3分組特定於平台的圖像與清單

您可以使用Docker桌面為不同平台構建圖像，但是它們不會成為多平台圖像，直到您將所有變體推向註冊表並將其分組為清單。 *manifest*是將圖像變體與同一圖像標籤鏈接的元數據。 您使用Docker命令行生成清單，並將其推到註冊表。清單包含所有圖像變體的列表，首先需要在註冊表上存在，因此工作流程是：創建和推動所有圖像，然後創建並推動清單。

現在嘗試

按下您構建的文件夾列表應用程序的圖像變體。 首先，您需要使用Docker Hub ID標記它們，以便您可以將它們推到您的帳戶中 - 您無權推動Diamol組織：

```
＃將Docker ID存儲在變量中 - 使用PowerShell：$ dockerid ='<您的doc
ker-hub-id >'

＃或a *nix shell：
dockerid ='<你的docker-hub-id >'

＃用您自己的帳戶名稱標記圖像：Docker Image Tag Diamol/ch12-folder-list：2e-linux-amd64“ $ doc
kerid/ch12-folder-列表：2e-linux-amd64”


Docker Image TAG DIAMOL/CH12-FOLDER-LIST：2E-LINUX-ARM64“ $ DOCKERID/CH12-FOLD
ER-列表：2E-Linux-ARM64”

Docker Image TAG DIAMOL/CH12折線列表：2E-Linux-Arm“ $ Dockerid/ch12-folder-列表：2e-lin
ux-arm arm”

＃並推到Docker Hub（這推動圖像的所有標籤）：Docker Image推動“ $ dockerid/ch
12-folder-list”
```

您會看到所有圖像都被推到Docker Hub。 Docker登記處是平台 - 不可知論 - 所有體系結構的圖像規範都是相同的，並且註冊表以相同的方式存儲它們。註冊機構確實知道為哪個平台構建了圖像，它們將其提供給Docker Engine作為檢查，然後才能將其拉動。我的輸出如圖12.7所示 - 每個圖像的體系結構都存儲在圖像元數據中。 我也喜歡在圖像標籤中包含平台名稱，但這不是必需的。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

We learned this in chapter 5 - you can only push images to Docker Hub if your account ID is in the image tag, this adds new tags with my account:

```
PS>$dockerId='sixeyed'
PS>
PS>docker image tag diamol/ch12-folder-list:2e-linux-amd64 "$dockerId/ch12-folder-list:2e-linux-amd64"
PS>
PS>docker image tag diamol/ch12-folder-list:2e-linux-arm64 "$dockerId/ch12-folder-list:2e-linux-arm64"
PS>
PS>docker image tag diamol/ch12-folder-list:2e-linux-arm "$dockerId/ch12-folder-list:2e-linux-arm"
PS>
PS>docker image push --all-tags "$dockerId/ch12-folder-list"
The push refers to repository [docker.io/sixeyed/ch12-folder-list]
d7761baad294: Pushed
d8993d35ea55: Pushed
d292b310006c: Mounted from diamol/base
78561cef0761: Mounted from diamol/golang
2e-linux-amd64: digest: sha256:f657f4c490b6db78b138ff57e3792b4b0e0df9abaa4991cb05fa3e82681f2dd4 size: 1152
ae5af6ac1e46: Pushed
8c9623c6b087: Pushed
034a881032ba: Mounted from diamol/base
799aa7236df0: Mounted from diamol/base
2e-linux-arm: digest: sha256:ddaeada5a19b060018a7a77c0dae3ceb4041c2185a0d5c281faae52700883b57 size: 1151
95a4b57c4b47: Pushed
4abd3fc31098: Pushed
2a018114310c: Mounted from diamol/base
9110f7b5208f: Mounted from diamol/base
2e-linux-arm64: digest: sha256:b08ec8beba73a87536279238701e626103ef8a5353fe44882d59432099799b17 size: 1152
```

You can push multiple images at once with the `all-tags` option - Docker pushes all the tags for that image name

圖12.7膿液　　牽引所有圖像變體是發布多平台的第一階段　　m圖像。

在創建清單之前，我們將查看現有的多平台圖像。您使用Docker清單命令在本地創建清單，將其推入註冊表，並檢查註冊表上的現有清單。 檢查清單是查看圖像所支持的架構的一種有用方法，而無需導航Docker Hub UI。您無需在本地刪除任何圖像 - 命令讀取註冊表中的所有元數據。

現在嘗試

使用CLI檢查本書的基本圖像的已發表清單：

Docker清單檢查Diamol/base：2e

清單檢查命令沒有過濾器或格式參數，因此您不能限制輸出。它將向您顯示圖像標籤的所有清單詳細信息，因此它可以與單個圖像以及多平台圖像一起使用。 在輸出中，您將看到每個圖像的唯一摘要以及CPU架構和操作系統。 我的（刪節）輸出如圖12.8所示。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

diamol/base:2e is a multi-platform image I use as the base for most of the other images in the book. It's based on Alpine for the Linux variants and Nano Server for Windows.

```
PS>docker manifest inspect diamol/base:2e
{
   "schemaVersion": 2,
   "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
   "manifests": [
      {
         "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
         "size": 738,
         "digest": "sha256:c716d4a16cf91af5cb2ba95643029a2768f9a0426b1f6549349fadb8159027e0",
         "platform": {
            "architecture": "amd64",
            "os": "linux"
         }
      },
      {
         "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
         "size": 738,
         "digest": "sha256:52b6d4e60ade368f53de11f4266493e5006ce4231fc5bfe5af345ed366e21f47",
         "platform": {
            "architecture": "arm64",
            "os": "linux",
            "variant": "v8"
         }
      },
```

Inspecting a multi-platform image manifest returns a *manifest list* with an array of manifests - one for each image variant

The digest is the unique ID for each variant. All manifests include the OS and CPU architecture; they may also include a CPU variant and an OS version

圖12.8多平台圖像具有一個清單列表，其中包含每個平台變體的清單。

現在，您可以創建清單，就像圖像一樣，它將首先存在於本地計算機上，然後將其推到註冊表上。從技術上講，您要創建的是*manifest list*，它將單個圖像標籤下的一組圖像組合在一起。每個圖像已經有自己的清單，您可以從註冊表中檢查，但是如果返回多個清單，您將擁有一個多構造圖像。 圖12.9顯示了圖像，清單和清單列表之間的關係。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

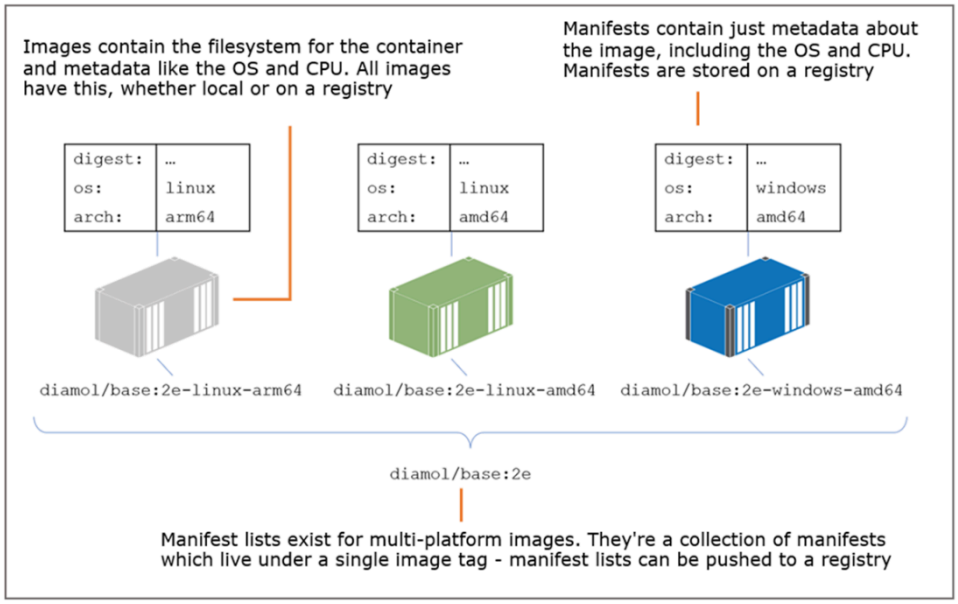


圖12.9 MA　　Nifests和清單清單存在於Docker註冊表上並包含Metada　　關於圖像。

您可以將清單列表視為圖像標籤的列表，而清單列表的名稱為多平台圖像的名稱。 您在本章中構建的圖像變體都有標籤以識別OS和CPU的標籤； 您可以使用沒有標籤的同一圖像名稱創建清單，並且可以使用默認最新標籤作為多平台圖像可用。您還可以使用一個標籤推動圖像，該標籤還包括OS和CPU之外的版本編號，然後僅將版本編號用於多平台圖像標籤。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

創建一個清單以鏈接所有Linux變體，然後將其推到Docker Hub。清單的名稱變為多架圖像的圖像標籤：

```
＃創建一個帶有名稱的清單列表，然後是其中包含的所有標籤：Docker清單創建“ $ dockerid/ch12-fold
er-list：2e”。文件夾列表：2e-linux-arr””



＃將清單推向Docker Hub：Docker清單推送“ $ dockerid/ch12-fold
er-list：2e”

＃現在在Docker Hub上瀏覽您的頁面並檢查IMA                    GE
```

當您瀏覽Docker Hub上的圖像時，您會發現有一個2E標籤具有多個變體 - UI顯示了OS和CPU架構以及唯一標識每個圖像的消化。任何具有Linux Docker發動機的人都可以從該圖像中運行一個容器，並且它將在Intel或AMD機器上運行AMD64變體，Apple Silicon上的ARM64變體，雲臂VM或最近的Raspberry Pi和近期的Raspberry Pi以及較舊PIS的ARM變體。您可以在圖12.10中查看我的Docker Hub存儲庫。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

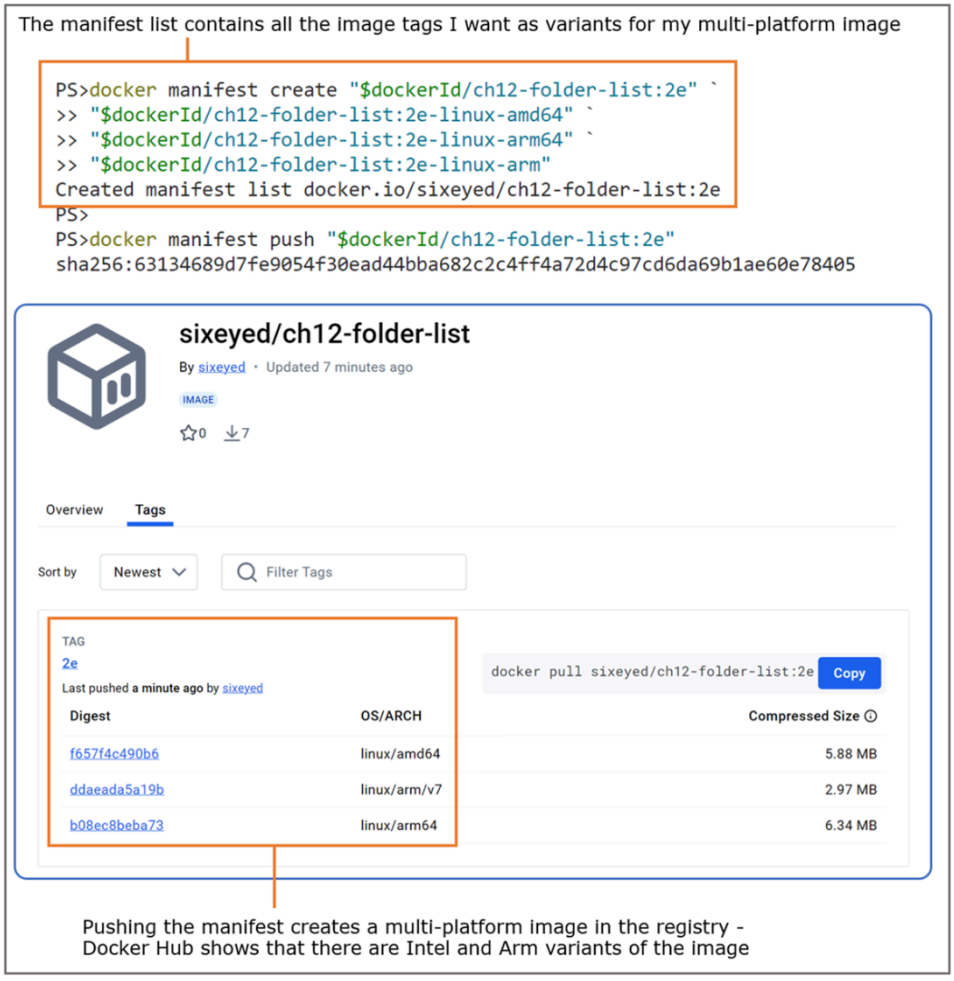


圖12.10多平台圖像具有單個標籤，但許多變體。 Docker Hub顯示所有變體。

這些手臂圖像是在Docker Desktop中模仿的，這僅對偶爾的構建真正可行。仿真很慢，並不是每個指令在仿真下的工作方式與實際CPU中的效果相同。如果您想發布多平台圖像，並且希望構建對於目標平台的快速且100%準確，則需要一個構建農場。這就是我必須在本書中構建圖像的方法，這是一套具有不同CPU體系結構的機器，並設置了我想要支持的所有操作系統。 我的CI/CD管道連接到每台機器上的Docker發動機，以構建圖像變體並將其推到Docker Hub，然後作業創建並推動清單。您可以使用雲中的託管服務做同樣的事情； GitHub Action和Azure DevOps為構建代理提供了不同的操作系統和CPU架構，您可以在多平台管道中使用它們。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

12.4用buildx構建和推動多平台圖像

還有另一種運行Docker Build Farm的方法，該農場更加高效，更容易使用，而Docker的最新功能稱為Build X。 *Buildx*是Docker Build命令的擴展版本，它使用了大量優化以提高構建性能的新構建引擎。它仍然使用Dockerfiles作為輸入並產生圖像作為輸出，因此您可以將其用作Docker Image Build的直接替換。 buildx真正為跨平台構建而發光，因為它可以將任務發送到多個Docker引擎，這意味著您可以使用單個命令在幾個平台上分發構建。

BuildX現在不適用於Windows容器，它僅支持從單個Dockerfile建造的，因此它不會涵蓋您在變體中需要不同設置的方案。 （我無法將其用於為本書構建每個圖像。）但是，如果您只需要支持Linux的CPU變體，則可以很好地工作。您使用buildx創建和管理構建農場以及構建圖像。

我們將瀏覽一個示例，以便您可以嘗試一個真正的分佈式構建農場，但是我們只使用一台機器並假裝有兩個機器。 Docker命令行可以使用*contexts,* 與不同的Docker引擎一起使用，*contexts,* 包含遠程計算機的地址和連接詳細信息。我們將創建默認上下文Docker正在使用的副本，因此我們將有兩個上下文指向我們的單個Docker引擎 - 建立Buildx認為我們有兩台機器。圖12.11顯示了我們的建築農場將如何工作。

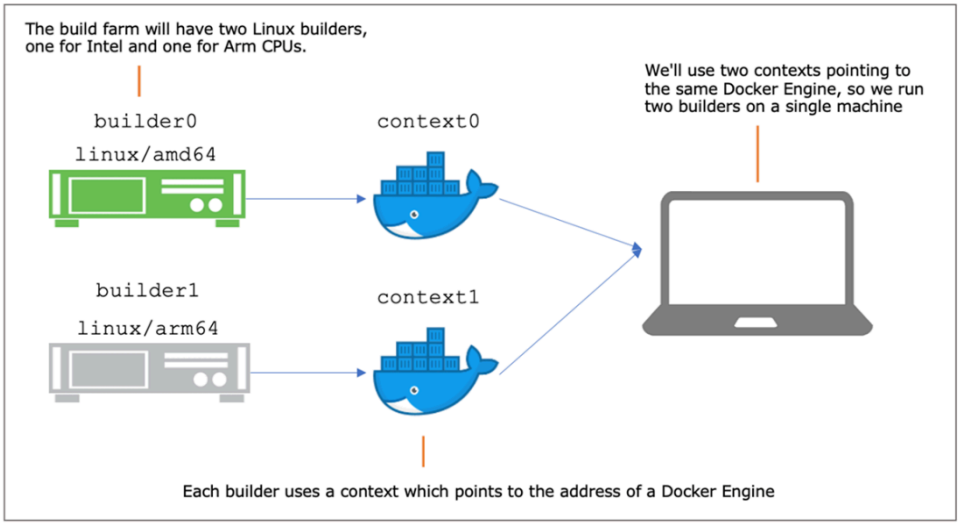


圖12.11您可以為單個Docker引擎使用兩個地址來託管兩個buildx構建器

Real Build Farms使用具有每個Docker引擎網絡地址的上下文。使用多台計算機，您需要設置Docker在網絡上聆聽，並使用SSH或TLS保護它。對於我們的Build Farm，我們不需要這樣做，因為我們將使用本地地址，並且Buildx不會在整個網絡上發送流量。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

創建當前Docker上下文的副本。

```
#打印您當前的conetxt - Docker始終默認創建一個：Docker上下文顯示


#現在創建當前上下文的副本：docker上下文創建-from $（Docker上下文顯示）
CH12-CONTEXT


#列出您的上下文 - 新的上下文將具有相同的地址：Docker上下文LS
```

圖12.12顯示了我的新Docker上下文，並從默認Docker桌面上下文中復制端點地址（實際地址對您來說是不同的）：

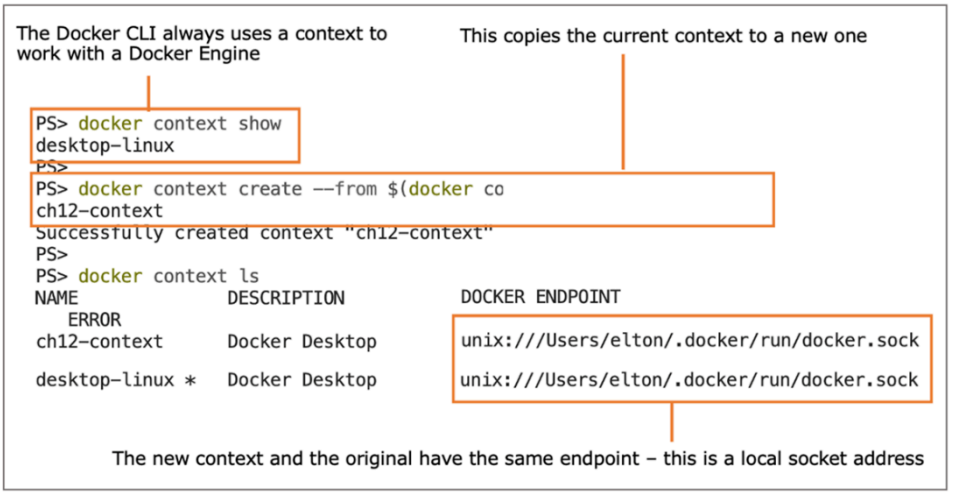


圖12.12 buildx使用上下文與構建者進行交談 - 它不介意不同的上下文是否使用相同的地址。

buildX設置並標識引擎可以構建的平台時，將連接到Docker上下文。 如果發動機有仿真，則可以構建多個平台，但是通常您的構建農場將是每個構建器的一個平台，因此所有圖像變體都是本地構建的。您還可以告訴buildx您要在每個Docker Engine上構建的平台（BuildX稱其為*build nodes*），這就是我們使用模擬的Build Farm進行的。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

創建一個帶有兩個構建節點的構建器 - 一個用於Linux ARM64，一個用於Linux Intel。 buildx將在需要的地方使用仿真，因此您的實際CPU體系結構是什麼：

```
＃創建構建器 - 這添加了一個帶有當前上下文的節點：docker buildx create-name ch
12-builder -platform linux/amd64

＃使用重複上下文附加第二個節點：docker buildx create-append -name ch12-builder -platform linux/ar
m64 ch12-上下文

＃列出您的構建者：Docke
r Buildx LS
```

這似乎是為了構建模仿圖像的努力（我們已經用簡單的圖像構建命令完成了），但這在這裡很重要。您將做一個完全相同的方法來設置具有多個本機節點的構建器：為每個Docker引擎創建上下文，然後將它們分組為Buildx Builder Nodes。圖12.13顯示了我的建築商。

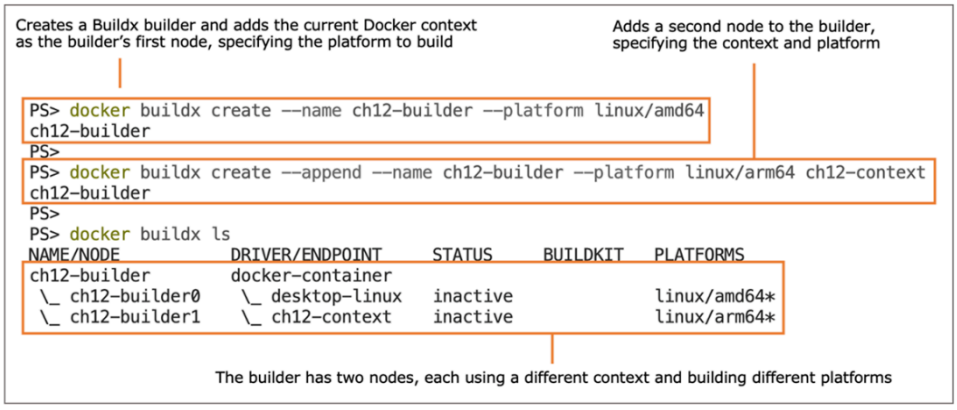


圖12.13我們 e創建了一個只有幾個命令的假建築農場 - 一個真正的建築農場 同樣容易

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在，建築農場已經準備好了。 它可以構建多平台圖像，只要它的建築物使用支持這兩個體系結構的圖像，就可以以64位Intel或ARM Linux容器運行。 buildx旋轉在構建器節點上同時構建，並通過Docker build build上下文將dockerfile和文件夾發送（通常包含您的源代碼）。構建命令還可以指示節點將其變體推向註冊表，而buildx將編譯並推出清單列表。使用一個命令，您可以使用與建築農場支持一樣多的變體來構建和推動多平台圖像。

現在嘗試

使用新的構建器從第1節中構建同一示例應用程序，然後直接推向Docker Hub。

```
＃為所有構建器的平台構建並推動Hello應用程序：Docker Buildx Build-builder ch12-Builder -platform linux/arm64，linux/amd64 -t “ $ dockerid/ch12-hello-x：2e”
```

BuildX構建的輸出令人印象深刻。當其他人看到您的屏幕時，跑步是一件很棒的事情。客戶端顯示每個構建器節點的日誌線，您將獲得很多快速輸出，因此看起來您正在做一些技術上的事情。 實際上，buildx完成了所有工作，您將從輸出中看到它推動圖像，創建清單列表並為您推動清單。圖12.14顯示了我的構建的開始和結尾。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Use the build farm to build two image variants for different platforms from one Dockerfile

```
PS> docker buildx build --builder ch12-builder --platform linux/arm64,linux/amd64
-t "$dockerId/ch12-hello-x:2e" --push ./hello
[+] Building 29.8s (35/35) FINISHED                 docker-container:ch12-builder
 => [internal] load build definition from Dockerfile                          0.0s
 => => transferring dockerfile: 229B                                          0.0s
 => [internal] load build definition from Dockerfile                          0.0s
 => => transferring dockerfile: 229B                                          0.0s
 => [linux/amd64 internal] load metadata for docker.io/diamol/base:2e        18.1s
 => [linux/amd64 internal] load metadata for docker.io/diamol/golang:2e      18.8s
 => [linux/arm64 internal] load metadata for docker.io/diamol/base:2e        18.8s
 => [linux/arm64 internal] load metadata for docker.io/diamol/golang:2e      18.7s
                                   ...
 => exporting to image                                                        8.3s
 => => exporting layers                                                       0.0s
 => => exporting manifest sha256:526f846d0c146588f97c60d597edbd730e4c4c1b22   0.0s
 => => exporting config sha256:7e409c1d54580aef5f9665e4244677a5a83a664ae197   0.0s
 => => exporting attestation manifest sha256:feb55cecb9cbdba718c0316efdb244   0.0s
 => => exporting manifest list sha256:50768139403606c9c9402dc8392be229e1dd4   0.0s
 => => pushing layers                                                         5.6s
 => => pushing manifest for docker.io/sixeyed/ch12-hello-x                    2.7s
 => [auth] sixeyed/ch12-hello-x:pull,push token for registry-1.docker.io      0.0s
 => [auth] sixeyed/ch12-hello-x:pull,push token for registry-1.docker.io      0.0s
 => [auth] sixeyed/ch12-hello:pull sixeyed/ch12-hello-x:pull,push token for   0.0s
 => [auth] sixeyed/ch12-hello:pull sixeyed/ch12-hello-x:pull,push token for   0.0s
 => merging manifest list sixeyed/ch12-hello-x:2e                             2.4s
```

Buildx collects and shows the output from the builder – the Intel and Arm64 builds are running on different nodes

At the end of the build the images are pushed to the registry, along with the multi-platform manifest

圖12.14使用t　　他建造者 - 每個節點構建一個平台，buildx推動了多個平台　　平台清單。

buildx使這些多平台構建非常簡單。 您可以為要支持的每個體系結構提供節點，並且buildx可以全部使用它們，因此您的構建命令都不會更改，無論您是為兩個架構還是十個建築構建。 與Docker Hub上的BuildX Image有一個有趣的區別：變體沒有單獨的圖像標籤，只是單個多平台標籤。將其與上一節進行比較，我們在這里手動推動了變體，然後添加了清單 - 所有變體都在Docker Hub上具有自己的標籤，並且隨著您構建和部署更多圖像版本，用戶很難導航。如果您不需要支持Windows容器，則buildX是構建多平台圖像的最佳方法。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

12.5考慮路線圖中的平台支持

但是，也許您現在不需要多平台圖像。很好 - 無論如何要閱讀本章。絕對值得知道多平台圖像的工作原理以及如何構建自己的構建，即使您不打算這樣做，因為它們很可能會出現在您的路線圖上。 您可以加入一個項目，開發人員使用Apple和Intel筆記本電腦的混合物，或者您可能需要削減雲中的運行成本，或者您的客戶可能會為Windows提供支持。圖12.15顯示瞭如何通過多平台支持來發展的項目，並在需要時增加了更多的變體。

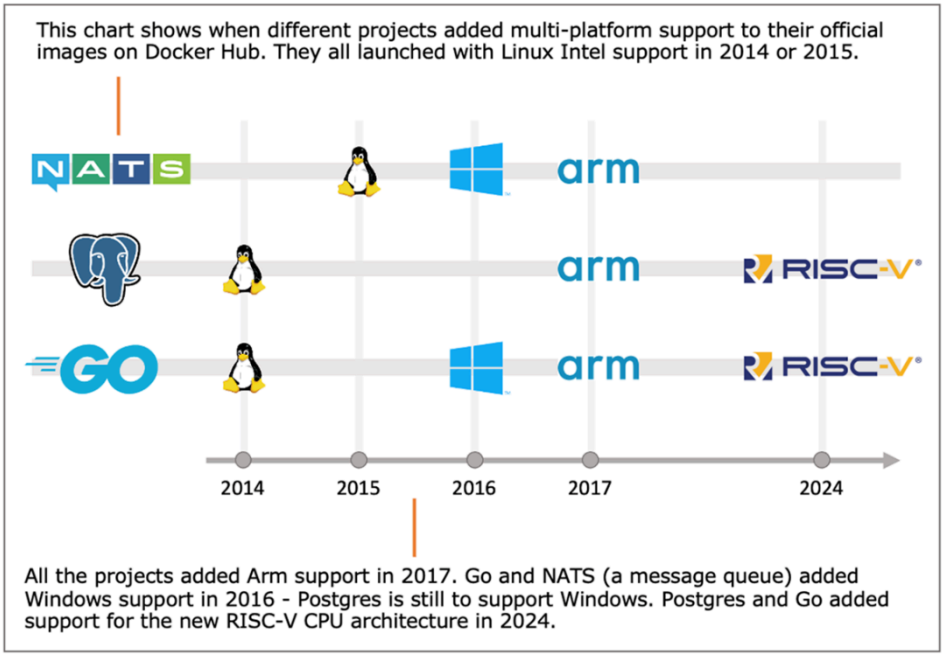


圖12.15項目通常以Linux Intel支持啟動，並在流行時添加變體。

如果您遵守所有DockerFiles的兩個簡單規則，可以輕鬆地將自己防止的未來效果：始終從說明中使用現有的多平台圖像，並且在Run或CMD說明中不包含任何特定於OS的命令。 如果您需要一些複雜的部署或啟動邏輯，則可以使用與應用程序相同的語言將其構建到一個簡單的實用應用程序中，然後在構建的另一個階段進行編譯，或在圖像中安裝跨平台外殼（如PowerShell）。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Docker Hub上的所有官方圖像都是多平台，因此最好將它們用作基本圖像（或使用官方圖像創建自己的金鹼圖像）。這本書的所有黃金圖像也是多平台，如果您正在尋找靈感，則可以在源中查看圖像文件夾中的圖像中的大量示例。作為粗略的指南，所有現代應用程序平台都支持多個架構（GO，Node.js，.net，Java），如果您正在尋找數據庫，那麼Postgres是我找到的最佳多平台選項。

那裡沒有任何託管構建服務可以支持全部架構的範圍 - Docker Build Cloud支持Intel和Arm，並與Buildx集成，但僅用於Linux。 Azure DevOps和GitHub動作提供了Windows代理，以及Intel和ARM64上的Linux。您還可以使用安裝Docker的Linux，Windows和Arm VM在雲中經營一個相當便宜的建築農場。重要的是不要假設您永遠不需要支持其他平台：遵循Dockerfiles中的最佳實踐，以使多平台支持變得容易，並知道如果您需要添加其他平台支持，則需要採取哪些步驟來發展構建管道。

12.6實驗室

本章的實驗室要求您修復Dockerfile，以便可以用於生成多平台圖像。如果您有一個沒有遵循我最佳練習建議的Dockerfile，您可能會遇到這種情況。 此DockerFile基於特定體系結構的圖像，它使用不可移植的OS命令。我希望您在本章的實驗室文件夾中修復Dockerfile，以便它可以在Intel或Arm上構建針對Linux的圖像，以及在Intel上的Windows。有很多方法可以解決這個問題。這些只是給您的幾個提示：

- 一些Dockerfile指令是跨平台，而運行指令中的等效OS命令可能不是。 某些Windows命令與Linux相同，在本書的黃金基礎圖像中，有一些使其他Linux命令在Windows中
- 起作用的別名。

You'll find my approach in the `Dockerfile.solution` file on GitHub for this chapter: https://github.com/sixeyed/diamol/blob/2e/ch12/lab/README.md.

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

13 重新構造遺產：docker中的包裝和運行Windows應用程序

雲是大多數公司想要運行大部分工作量的地方。初創企業可以在Go或Rust中從頭開始構建所有內容，在Linux Docker Images中包裝所有組件，然後在任何云上的容器中運行它們。成熟的公司並不容易 - LARGER和舊組織擁有更大和舊的應用程序，並且很可能包括僅在Windows上運行的產品。 這些傳統應用程序通常是業務的核心，他們希望以相同的規模和每個人想要的成本收益將它們移至雲中。 但是，在最新的技術堆棧中重寫它們不是一個選擇 - 我們正在談論十年前設計的應用程序，現在擁有數百萬行代碼。 如何將它們包裝在Windows Docker圖像中以在現代應用等容器中運行？這是某些人不在乎的一種選擇，因為感覺就像是一個偽裝舊代碼庫的真正問題。但是我不同意 - 這是*replatforming*舊應用程序，因此可以以與現代應用程序相同的方式來構建，部署和管理它們。它為您的工具和實踐提供了一致性，這意味著您可以在雲中同一平台上運行所有應用程序。

在本章中，您將學習如何使用已經學到的許多相同的技術和技能來容忍舊應用程序。所有舊應用程序的方法都是相同的，但是這裡的重點是在Windows容器上，這些容器具有一些怪癖，您需要了解一些怪癖，這可能會從搬到Docker中獲得最大的好處。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

13.1 Windows容器如何不同

微軟很早就看到了Docker中的價值，並以專門將容器帶到Windows和Windows容器（正確稱為*Windows Server Containers*）的團隊來支持該技術。 Windows Server容器帶有Windows Server 2016和Windows 10到達，並在隨後的每個版本中包括並改進。關於Windows Server容器，有兩個基本知識：它們僅在Windows上運行，Windows的版本必須匹配容器映像和運行該容器的機器。 這為您在第12章中所學的多平台圖像添加了另一個變量：Windows中的平台匹配包括CPU架構，操作系統和操作系統版本。

現在嘗試

您可以通過比較存儲在Microsoft的容器註冊表（MCR）和多平台Docker Hub圖像的Windows容器圖像的清單來看到這一點。您無需為此運行Windows；您可以使用任何平台看到清單列表。

```
＃在MCR Docker清單上打印僅Windows圖像的清單列表檢查Mcr.microsoft.com/dotn
et/framework/sdk:4.8


＃和Docker Hub上的多OS圖像：Docker清單檢
查Golang：1.23
```

您會看到每個圖像返回多個變體。 .NET Framework 4.8是Microsoft僅限Windows開發平台的最後一個支持版本，所有圖像變體均用於Windows和64位Intel平台。 GO圖像是多平台，包括Linux和Windows的不同版本。 在圖13.1的輸出中，我將結果格式化為表格；您的輸出將具有相同的數據，但是在JSON中：

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The manifest list contains all the platform details for each variant. For Windows images that includes the Windows OS version (I'm using a JQ filter to format the results):

```
PS> docker manifest inspect mcr.microsoft.com/dotnet/framework/sdk:4.8 |
 jq -r $filter
CPU     OS      OS Version
---     --      ----------
amd64   windows 10.0.14393.7515
amd64   windows 10.0.17763.6532
amd64   windows 10.0.20348.2849
PS>
PS> docker manifest inspect golang:1.23 | jq -r $filter
CPU     OS      OS Version
---     --      ----------
amd64   linux
arm64   linux
amd64   windows 10.0.20348.2849
amd64   windows 10.0.17763.6532
```

The Go image is multi-platform. There are Linux variants for Intel and Arm64, but for Windows the variants are for Intel and different Windows versions.

圖13.1多平台圖像變體包括Windows圖像的OS版本。

這不是開始本章的令人興奮的方法，但重要的是要知道：為Windows Server 2025構建的Docker映像不會在Windows Server 2022機器上運行。這也適用於客戶端操作系統：您可以在Windows 11上運行Windows Server 2022映像，但在Windows 10上不能運行。表13.1顯示了當前的服務器和OS版本，如果您嘗試運行不兼容的映像並獲得錯誤，則需要參考，沒有匹配的清單。

表13.1 Docker圖像和操作系統之間的Windows版本匹配

Image OS Version	Server Version	Client Version
10.0.14393	Windows Server 2016	Windows 10 (version 1609)
10.0.17763	Windows Server 2019	Windows 10 (update 1809)
10.0.20348	Windows Server 2022	Windows 11 (version 23H2)
10.0.26100	Windows Server 2025	Windows 11 (update 24H2)

現在我們已經解決了這個問題，Linux和Windows Docker映像之間還有另一個區別：Windows圖像很大 - 非常大。 Windows Server Core映像實際上是所有Windows Server沒有GUI的。目的是您可以使用它運行幾乎所有Windows應用程序，因此它具有大型功能集以使其向後兼容較舊的應用程序，而該圖像的2025版本為5.1GB。如果您的應用不需要完整的Windows Server功能集，則可以使用Nano Sever Image，它仍然是Windows，但請嚴重修剪。 Nano Server的位於480MB，這是完整服務器核心映像的大量節省，但與Alpine Linux（即8MB）相比效果不佳。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

13.2帶有Windows Server容器的包裝應用程序

本章的其餘部分是使用Windows Server容器的動手操作，因此您需要訪問Windows計算機以遵循（Windows 10或11帶Docker桌面的Windows 10或11是最簡單的選擇，或者您可以在雲中旋轉Windows Server VM並安裝Docker）。好消息是，到目前為止，您所學到的核心Docker概念和工作流程與Windows相同：您在Dockerfile中指定安裝步驟，將其構建到圖像中，將圖像推到註冊表中，然後將圖像拉動以運行容器。

現在嘗試

單擊Docker桌面的鯨魚圖標，然後選擇切換到Windows容器的選項。然後使用Microsoft的Internet信息服務（IIS）在Windows容器中運行Web服務器：

```
＃確保您使用的是Windows容器:) docker info -f'{{.operating
system}}}'

＃此圖像支持幾個Windows版本：Docker清單檢查MCR.MICROSOFT.com/windo
ws/servercore/iis

＃運行一個IIS容器 - Docker選項與Linux相同：Docker Run -D -P 8083：80 mcr.micr
osoft.com/windows/servercore/iis
```

這似乎都很熟悉。 Docker拉動圖像並在後台啟動一個容器，將端口從容器中發佈到主機計算機。您可以瀏覽到http：// localhost：8083，您應該看到標準的IIS歡迎頁面 - Mine在圖13.2中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

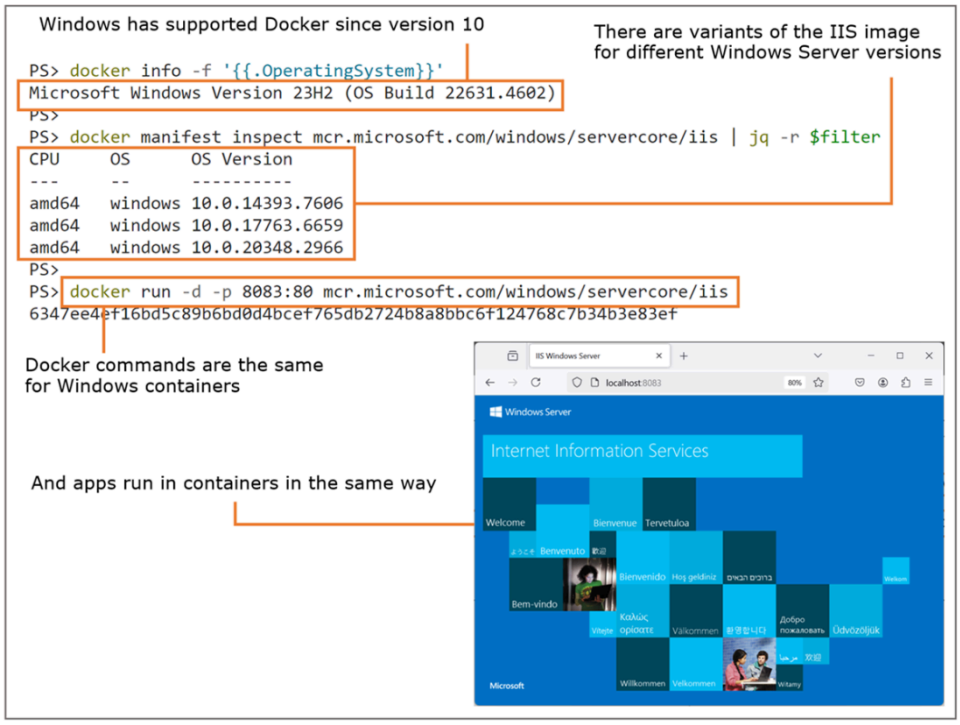


圖13.2在Windows服務器容器中運行Windows Web服務器（IIS）

Microsoft在MCR上提供了許多基本圖像 - 所有構建在核心Windows圖像之上的所有基礎圖像 - 包括IIS，.NET Framework和ASP.NET。包裝Windows應用程序在Docker中運行的第一個任務是選擇正確的基本圖像，而指導與Linux相同：選擇一個為您提供所需的大多數依賴項（因此您不必自己維護）的圖像，而沒有其他依賴關係（以保持圖像大小和攻擊表面積小）。 然後，您可以確定如何自動化應用程序的部署，並在Dockerfile中腳本腳本。

Windows應用程序具有一些典型的部署方法。 簡單的應用程序可以分佈在zip文件中，並僅通過提取存檔來部署。 第三方應用程序或更複雜的企業應用程序可以在MSI（Microsoft Installer）文件中傳遞。 MSI可以做很多事情 - 創建文件，編輯Windows註冊表，設置配置選項。 通常，您通過雙擊文件並遵循安裝窗口來部署MSI，但是在容器中沒有GUI，因此您需要腳本腳本。列表13.1顯示瞭如何將MSI包裝應用程序部署到Windows容器映像中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

清單13.1在Dockerfile中部署Windows MSI

```
arg windows_version = ltsc2025來自mcr.microsoft.com/windows/servercore：$ {windows_version}作為in
staller shell [ “ powerShell” ， “ - command” ] workdir/install astall copy copy copy helly-world.msi。
運行start -process msiexec.exe -argumentList'/i'，'hello -world.msi'，'/letim'，'/norestart' - nonewwindow
-wait-來自mcr.microsoft.com/windows/windows/windows/nanoserver：jullydir} （x86）/diamol-ch13/hel
lodiamol.txt “，” /app/hello-world.txt “] cmd [ “ cmd” ， “/c” ， “ type” ， “ Hello-World.txt”
```

這個簡短的Dockerfile具有許多有趣的功能：

- 這是多階段的構建。安裝階段使用Windows Server Core和構建參數設置Windows版本（默認為Windows Server 2025 -*LTSC*代表長期支持頻道的代表）。
- 安裝程序切換到PowerShell，該PowerShell已在服務器核心圖像中預安裝。
- 從Docker上下文中復制MSI文件，然後使用MSIEXEC工具部署（也已在服務器核心中預裝）。該安裝使用PowerShell的Start-Process，該過程在後台運行安裝程序並等待完成。
- 最終的應用程序映像基於Nano服務器 - 使用與安裝程序階段相同的構建參數來設置Windows版本。
- 將文本文件從安裝程序複製到最終圖像中，並將容器命令設置為使用Windows命令工具來打印文件的內容。

您可以構建此圖像並以通常的方式運行一個容器，但是請注意，並非所有Microsoft的圖像都是多平台的，因此，如果您構建Windows 2025版本的錯誤，則需要在構建參數中設置您的版本。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

從安裝程序FIL0065構建容器圖像並運行容器：

```
CD CH13 \練習

＃使用默認2025 OS版本構建：Docker Build -T Diamol/ch13-hello-world
：2e。 \ Hello-World

＃如果遇到明顯錯誤，請將版本設置為2019或2022：＃docker build-build-arg windows_version =
’ltsc2022’-t diamol/ch13-hello-世界：2e。 \ hello-world

＃運行容器：Docker Run Diamol/CH13-HELLO
-WORLD：2E
```

我的輸出在圖13.3中 - 您應該看到這是標準的Docker工作流程，而Dockerfile語法和CLI命令在Windows和Linux中相同。您可以使用Docker標籤命令與ID創建另一個標籤，然後將此圖像推到Docker Hub。您會在Web UI中看到它是Windows AMD64圖像。

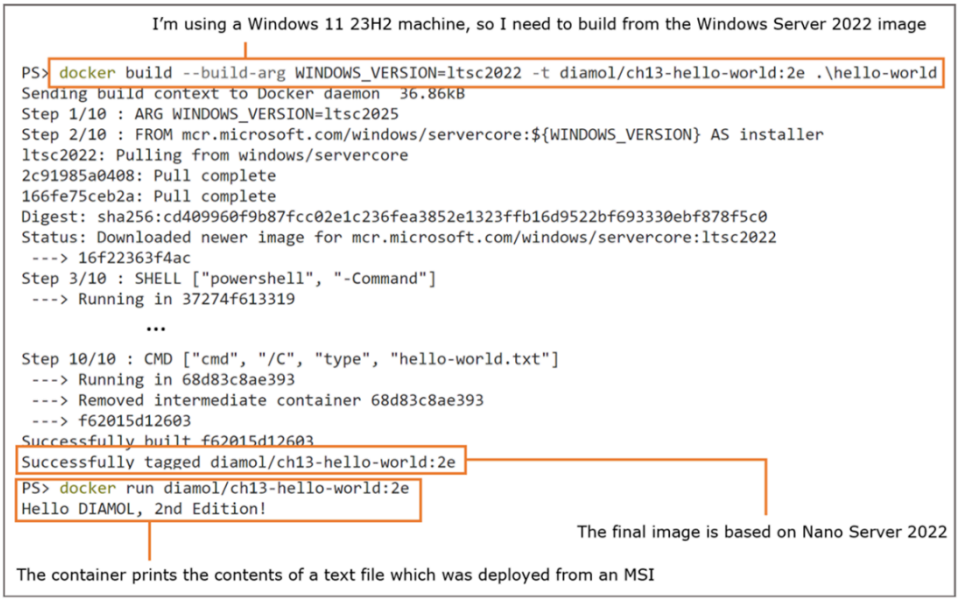


圖13.3通過在Dockerfile中安裝MSI來包裝Windows Docker映像

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

如果您只能使用安裝程序軟件包，則部署MSI是一個有效的選項。您可以使用這樣的多階段構建來在一個階段部署，然後將所需的內容複製到最後階段。您如何知道使用MSI安裝文件的位置？您必須知道……希望有文檔，或者您可以與開發團隊合作。如果沒有，那麼您可以嘗試在一個階段部署MSI，並且您可能會使您的應用程序正常工作，但是您的Dockerfile並不是對部署的明確規範。該邏輯仍然隱藏在MSI中。

如果您的應用程序可以選擇，則文件系統部署更好。 基於Windows Server Core的圖像已安裝了PowerShell，它為您提供了巨大且可擴展的工具集（您可以在DockerFile中運行安裝模塊命令），並且最近的圖像還包括熟悉的Linux工具，例如Curl和Tar。這解鎖了典型的工作流以下載存檔，檢查哈希並提取它。 MCR上的圖像範圍意味著您應該能夠找到合適的基礎，以使您的Dockerfile保持簡單。 清單13.2顯示了使用存檔模式部署ASP.NET應用程序的Dockerfile。

清單13.2從文件存檔部署ASP.NET應用程序

```
arg windows_version = ltsc2025來自mcr.microsoft.com/windows/servercore：$




arg zip_url = https://is.gd/diamol_2e_ch13_app_zip_zip arg hash_ur
l = https://is.gd/diamol_2e_2e_2e_ch13_ch13_ch13_app_hash


WorkDir /安裝


運行curl.exe -ssl -o app.zip $ env：zip_url; \ curl.exe -ssl -o sha256 $ env：hash_url; \ $ endureh
hash = get -content -path sha256; \ $ usealhash =（get -filehash -algorithm sha256 -path app.zip）
.hash; \ if（$ mutherhash -ne $ endure hash）{exit 1}; \ Expand -Archive -path App.zip -destinati
onpath /app -force




來自mcr.microsoft.com/dotnet/framework/aspnet:4.8


env diamol_key_1 =“第一個值” \ diamo
l_key_2 =“第二個值”


workdir/inetpub/wwwroot副本-from = i
nstaller/app/./
```

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

我不會在此處詳細介紹PowerShell（該主題有一個*Month of Lunches*標題），但希望您能看到行動中的模式。 安裝程序階段使用基本Windows Server Core映像進行下載，並且僅將經過驗證的應用內容複製到最後階段 - 基於ASP.NET映像。您構建圖像並以通常的方式運行容器，但是值得一看到達容器中運行的過程。

現在嘗試

構建ASP.NET圖像，運行一個容器並檢查詳細信息。

```
＃使用默認2025 OS版本構建：Docker Build -T diamol/ch13-aspnet：2e。 \ aspnet＃如果遇到明顯錯誤，請將版本設置為2019或2022：＃docker build -build-arg windows_version-windows_version-windows_version =’ltsc2022’ltsc2022’-tiamol/ch13-asp ant。




＃在後台運行容器：
Docker Run -name Aspnet -D -P 8084：80 Diamol/ch13 -Appnet：2e

＃查看容器過程：Docker Top Aspnet
```

運行此操作時，您會注意到有關Windows Server容器的其他幾件事：它們的啟動速度要比Linux容器慢得多，容器中正在運行許多過程。您可以在圖13.4中看到我的輸出。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

This is multi-stage build which downloads an application archive and expands it. The final image is based on Microsoft's ASP.NET 4.8 Windows image.

```
PS> docker build --build-arg WINDOWS_VERSION=ltsc2022 -t diamol/ch13-aspnet:2e .\aspnet
Sending build context to Docker daemon   5.12kB
Step 1/11 : ARG WINDOWS_VERSION=ltsc2025
Step 2/11 : FROM mcr.microsoft.com/windows/servercore:${WINDOWS_VERSION} AS installer
 ---> 16f22363f4ac
                ...
Successfully built 0f49ed4c57f6
Successfully tagged diamol/ch13-aspnet:2e
PS>
PS> docker run --name aspnet -d -p 8084:80 diamol/ch13-aspnet:2e
8bbedebce758e93c2dfdb726183e144b411b0799e30040b16bfbc9d8f7aaf76b
PS>
PS> docker top aspnet
Name                PID                 CPU                 Private Working Set
smss.exe            984                 00:00:06.125        258kB
csrss.exe           68                  00:00:00.140        884.7kB
wininit.exe         304                 00:00:00.281        802.8kB
services.exe        544                 00:00:00.203        2.122MB
lsass.exe           652                 00:00:00.468        2.515MB
svchost.exe         1056                00:00:00.171        2.163MB
svchost.exe         1184                00:00:00.125        1.692MB
svchost.exe         1280                00:00:00.109        2.281MB
```

It takes a lot longer for a Windows container to start – it could be 30+ seconds

One reason is all the Windows Services which start in the background. This is abbreviated output, you'll see even more processes in your container.

圖13.4查看Windows容器中的過程比Linux中顯示的要多得多。

與Linux容器相比，Windows Server容器可能需要30秒才能啟動並在其中運行20個進程，該過程可能會從一秒開始並運行一個進程。這兩個點是相關的。 Windows機器啟動時會做很多工作，將核心組件作為Windows服務運行。 那些在Linux Daemons之類的背景中運行 - 您會看到它們作為svchost.exe流程運行，並且它們管理OS級的工作，例如儀器和DNS查找。您用於運行應用程序的平台可能還可以作為Windows服務運行，這為容器化Windows應用程序增加了另一個挑戰。

ASP.NET應用程序託管在容器內的IIS Web服務器中。 IIS作為Windows服務運行，因此它由您可以看到的SVChost.exe進程之一管理。當第一個客戶端向IIS提出Web請求時，它會啟動一個工作流程 - 您將其視為w3wp.exe。 但是請記住Docker的工作原理：它希望在運行容器時啟動單個進程，並在該過程結束時監視該過程，收集日誌輸出並退出容器。 這是實際運行該應用程序的工作過程，但是Docker對此一無所知或IIS Windows服務，因此Microsoft提供了兩個實用程序來彌合差距：

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

- *Service Monitor*是一個控制台應用程序，您可以用作Docker容器的入口點。您告訴它您要運行的Windows服務，然後開始並觀看該服務。如果服務失敗，則服務監視器結束並且您的容器退出。這遵循Docker模式，表示您的容器狀態正確反映了您的應用程序狀態。 *Log Monitor*是您配置的另一個控制台應用程序，以讀取來自不同源的日誌條目（日誌文件，Windows的事件日誌和事件跟踪）。它收集日誌並將其打印到標準輸出中，因此Docker將其視為容器日誌。日誌監視器包裝您的應用程序過程，因此將日誌監視器設置為輸入點，並告訴它要啟動哪個過程（可以是服務監視器）。
-

這兩個工具可以很好地工作。 Windows上的Web應用程序通常使用IIS並將日誌寫入文件，您無需更改它們的工作方式即可將其帶到Docker。 圖13.5顯示了您如何使用監視器工具來製作像普通容器一樣行為的應用。

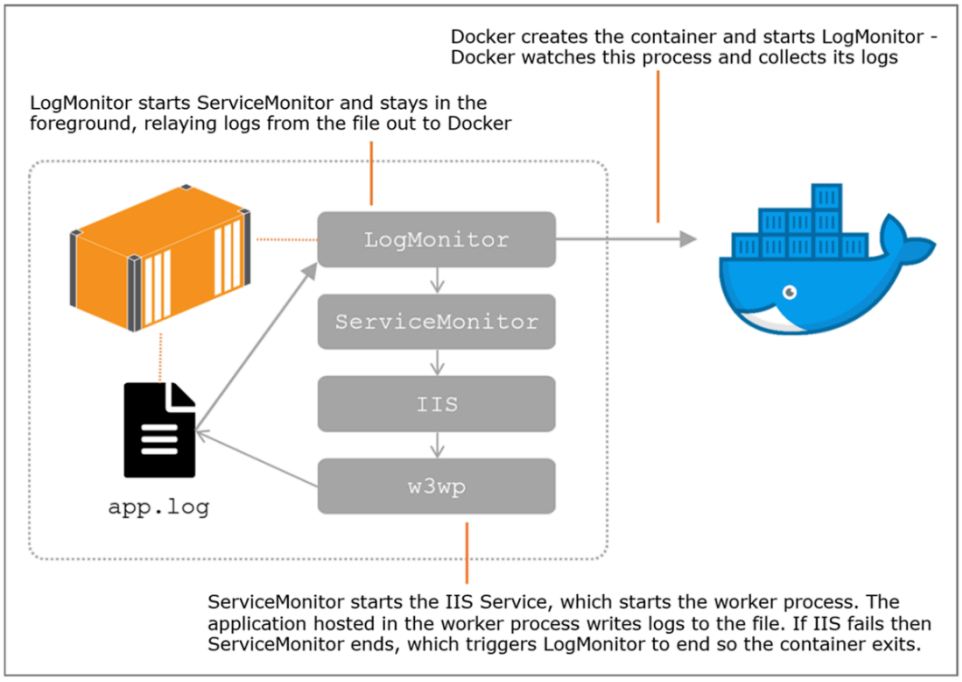


圖13.5監視器工具按照設計方式運行舊應用程序，但行為就像普通的容器。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

這種方法意味著您的舊Windows應用程序的工作方式與所有其他集裝箱應用程序相同。 這就是為什麼用Docker重新構造的舊應用程序如此吸引人的原因 - 您將它們帶入現代運行時而無需重寫它們。 您甚至可能再也沒有該應用程序的源代碼，但是您仍然可以使用現有的部署工件將其包裝到Docker映像中。如果您確實具有源代碼，並且該應用程序仍在積極的開發中，那麼更好 - 您可以將Docker用於構建，並採用我們在第11章中學到的相同的CI過程。

13.3在容器中建立.NET框架應用程序

Microsoft於2002年發布了.NET框架（這是我與C＃開始的旅程的時候），並繼續將平台擴展到2022年版本4.8。現在，它已經演變成跨平台.NET，但是將大型舊應用程序從.NET框架遷移到.NET可能是一個昂貴的項目。用Docker對該舊應用重新構造是將其帶入現代云環境的一種更快的方法。 Microsoft為.NET Framework發布了SDK和運行時圖像，因此您可以使用多階段DockerFiles從源來編譯您的應用程序。

在本章的其餘時間，我們將對一個名為Nerd Dinner的舊.NET Framework應用程序進行停靠。 該名稱將是特定年份的C＃開發人員熟悉的 - 這是Microsoft的某些團隊編寫的開放源應用程序，以展示ASP.NET的新功能。最後一個代碼更改是在2013年進行的，因此是一個理想的候選人，看看我們是否可以將其帶到Docker而不需要更改任何代碼。看看列表13.3中的Dockerfile，從我們在第4章中看到的構建器模式中應該非常熟悉。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Listing 13.3 Building and packaging a .NET Framework app in Docker

```
FROM mcr.microsoft.com/dotnet/framework/sdk:4.8 AS build

WORKDIR /src
COPY src/NerdDinner.sln .
COPY src/NerdDinner/NerdDinner.csproj ./NerdDinner/NerdDinner.csproj
COPY src/NerdDinner/packages.config ./NerdDinner/packages.config
RUN nuget restore

COPY src/ .
RUN msbuild NerdDinner/NerdDinner.csproj /p:DeployOnBuild=true

FROM mcr.microsoft.com/dotnet/framework/aspnet:4.8

WORKDIR /inetpub/wwwroot
COPY --from=build /src/NerdDinner/obj/Debug/Package/PackageTmp/ ./
COPY docker/web/override/ ./
COPY docker/web/startup.ps1 /

ENTRYPOINT ["powershell"]
CMD ["-File", "/startup.ps1"]
```

構建階段使用MCR的.NET Framework 4.8 SDK，並在C＃項目和Nuget軟件包文件中復制（列出了所有依賴項）。 接下來，它運行一個還原命令來加載這些依賴項，然後在源代碼的其餘部分中復制並編譯應用程序（Nuget和Msbuild工具已經安裝在SDK Image中）。 最終圖像從Microsoft的ASP.NET圖像開始，該圖像與IIS一起安裝了.NET框架。 構建階段的輸出複製到默認的IIS內容目錄中。 有一個自定義的啟動PowerShell腳本，以及從Override文件夾中復制的幾個配置文件，因此我們可以具有特定於容器的配置，並將默認配置文件放在開發人員的源文件夾中。

現在是時候嘗試書呆子晚餐了。當您等待下載圖像並運行的構建時，請記住這是源代碼 *which hasn't changed since 2013*。我們正在使用一項甚至沒有夢想進行最後一次更改的技術來構建它。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

從來源構建書呆子晚餐圖像 - 請注意您在構建過程中看到的任何有趣的消息。然後從圖像運行一個容器。

```
#build - dockerfile嵌套在項目文件夾中：docker build -t diamol/ch13-ner
d-dinner-web：2e -f。


#運行 - 發佈網絡端口：Docker Run -D -P 8085：80 Diamol/ch13 -nerd
-dinner -web：2e


#瀏覽到http：// localhost：8085
```

我發現該應用程序構建，更不用說運行了。 可以證明，Docker圖像擁有您所需的所有內容，.NET Framework 4.8工具能夠編譯和包裝代碼為舊版本（在這種情況下為4.5），並且可以簡單地用Docker替換。 我在圖13.6中的輸出表明該網站加載沒有任何錯誤，並且主頁看起來幾乎可以預期。

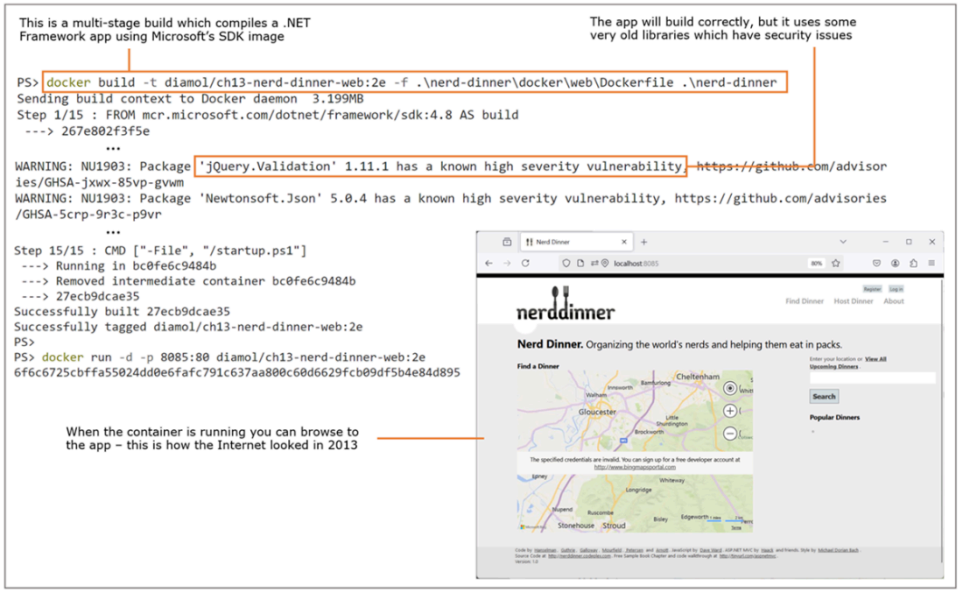


圖13.6歡迎來到2013年！在ASP.NET窗戶容器中運行的書呆子晚餐

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

不過，我們還沒有完成。所有應用程序都對其他組件以及舊版應用程序都有依賴性，其中一些可能會帶來意外的問題。 您可能已經註意到構建輸出中的一些令人恐懼的警告，稱圖書館包具有“已知的高度嚴重性漏洞”。這些是十年來的開源圖書館，因此發現了漏洞。這些已在以後的版本中進行了修補，如果這是一個真實的遷移項目，我們希望更新參考文獻，並嘗試找到一個沒有漏洞但仍與應用程序兼容的版本。

外部依賴性在較舊的應用程序上也存在問題，因為它們可能使用不支持或不再存在的第三方服務。書呆子晚餐在UI中使用Bing Maps和稱為ipinfodb的地理位置服務。 在撰寫本文時，兩者都仍然可用，您可以註冊API密鑰並將其插入書呆子晚餐配置，並且該應用程序可以按預期工作（請參閱Docker文件夾中的ReadMe文檔，如果您想嘗試一下，請參閱Docker文件夾中的讀數文件）。 企業應用程序往往會緊密耦合，因此您可能會發現一個應用程序向雲的遷移很複雜，因為它取決於其他仍在本地運行的應用程序。

書呆子晚餐的最終依賴性是SQL數據庫。這裡的故事可能是一個快樂的故事，因為數據庫傾向於在許多發行版中保持向後兼容，因此舊應用程序將與最新的數據庫版本一起使用。 正如我們從第6章中知道的那樣，數據庫在容器中運行良好。

13.4使用SQL Server數據庫容器

SQL Server是Microsoft的關係數據庫，它的歷史記錄幾乎與Windows本身一樣長。 最新版本是跨平台的，可用於100％兼容的Linux和Windows的構建。 Microsoft在Linux上發布了SQL Server的MCR圖像，但是如果要在Windows容器中運行數據庫，則需要包裝自己的SQL Server Express的安裝（可以免費使用）。 我有一個在Docker Hub上發布的圖像，其中包含在Windows Server Core Base Image上安裝的最新SQL Server版本。

© Manning Publications Co. To comment go to liveBook

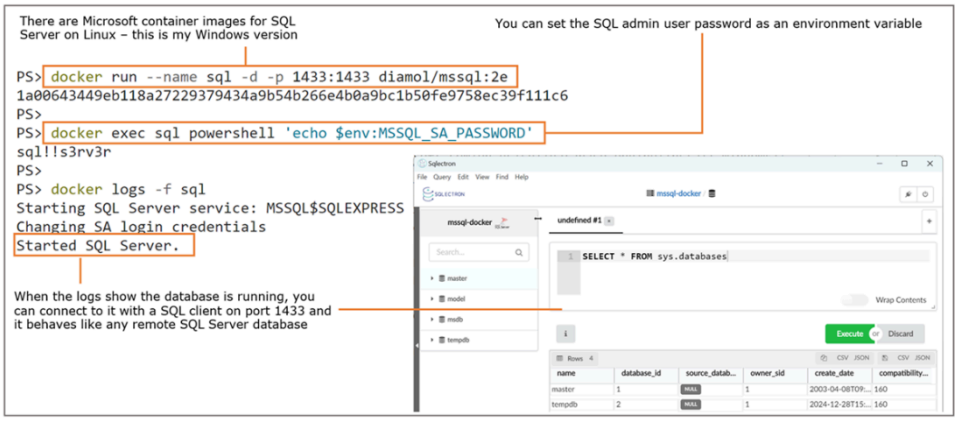
Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

在Windows容器中運行SQL Server - 如果您安裝了SQL客戶端，則可以使用端口1433在Localhost上連接。

```
＃運行SQL Server，公開客戶端端口：Docker Run -name SQL -D -D
-P 1433：1433 DIAMOL/MSSQL：2E

＃打印`sa 數據庫用戶的密碼：docker exec sql powershell'echo $ env：ms
sql_sa_password'

＃觀看日誌以查看數據庫何時準備連接Docker Logs -F SQL

＃當您看到日誌時，請使用CTRL-C退出
```

如果您曾經下載並安裝了SQL Server的全新部署，您會知道這可能需要幾個小時，但是使用我們的容器，我們將在短短幾分鐘內就有一個完全可用的數據庫。您可以使用SQL Server Management Studio或輕量級SQL客戶端使用Localhost地址和SA用戶憑據連接。圖13.7顯示了我的數據庫容器運行，並使用SQLectron使用客戶端連接。



figu RE 13.7歡迎到2013年！書呆子晚餐在ASP.NET窗戶中運行　　容器

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

您的應用程序可能需要一個空的數據庫容器 - 如果代碼使用對象關聯映射（ORM）工具（例如Entity Framework），那麼該應用程序將具有自己的邏輯來創建數據庫結構。書呆子晚餐採用另一種方法，那就是它希望數據庫已經存在。該結構作為一組SQL腳本保存在源代碼中，但是沒有自動部署。如果我們可以對數據庫代碼採用與應用程序代碼相同的方法並在Docker中構建和包裝，那不是很好嗎？ SQL Server實際上可以輕鬆執行此操作。

我們不會因為技術特定於SQL Server而深入研究它，但是該方法對於任何數據庫都是一個很好的模式，如果您想更近的話，代碼就在那裡。目的是擁有一個使用版本的設置為您的數據庫構建的Docker映像。 您可以通過兩種方式使用此圖像：從中運行一個容器來使用本地數據庫進行開發或測試環境； 或運行一個容器以從以前的容器或遠程數據庫中升級架構。 圖13.8顯示了它的外觀，這是將現代CI/CD方法帶入數據庫開發的好方法。

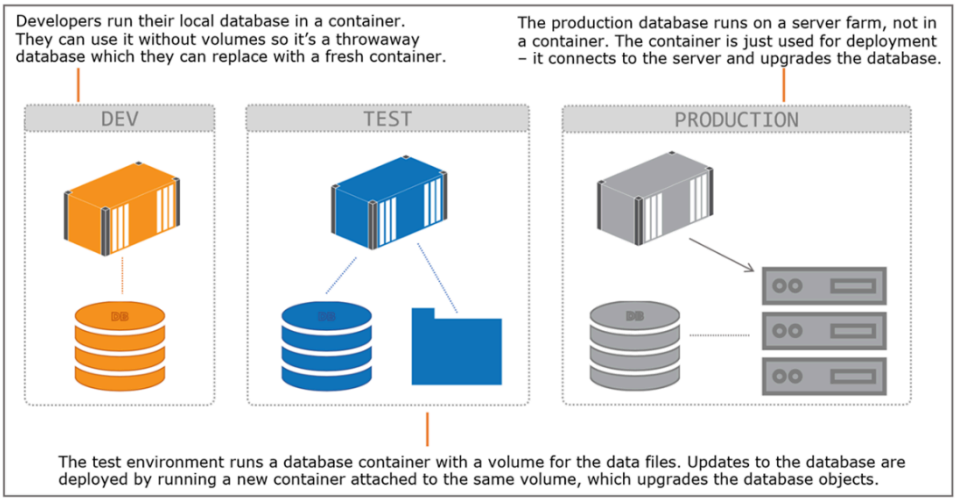


圖13.8 Docker映像中的SQL數據庫包裝可讓您運行容器化數據庫，或使用容器來升級現有數據庫服務器。

您可以使用稱為DACPAC的部署工件在SQL Server中執行此操作。 您以與構建.NET項目相同的方式來構建它，而輸出是包含數據庫架構的二進製文件（DACPAC）。該DACPAC可以部署到空的SQL Server實例中，以創建數據庫的新版本。 或者，您可以將部署工具指向現有的SQL Server數據庫，並將當前架構升級到DACPAC中的定義。這意味著您可以在非生產環境中使用快速，一次性的容器，並且當您有新的版本要部署時，您將使用所測試的相同部署偽像在生產服務器上進行操作。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這聽起來都很神奇，尤其是當您來自企業背景，在該背景下，單獨的團隊應用數據庫更改，並且該模式甚至沒有在源控制中。但是實現非常簡單。 清單13.4顯示了書呆子晚餐數據庫圖像的Dockerfile，該數據庫圖像使用了與應用程序Dockerfile相同的多階段構建方法。

列表13.4 docker中的構建和包裝.SQL服務器數據庫

從diamol/dotnet-sdk：2e 64作為構建

workdir/src複製src/nerddinner.database/./nerddinner.database複製src/nerddinner.database.build/./nerddinner.database.build workdir workdir/src/nerddinner.datinner.database.build build nerddinner.database.build.csproj -o /out diamol /mssql：2e shell [“ powerShell” ，“ - c ommand” ，“ $ erroractionPreference = stop'stop';”]]

Workdir /Init

複製-from = build/out/out/nerddinner.database.build.dacpac ./nerddinner.dacpac複製docker/db/inditialize.ps1 .ps1 ./

CMD。

這裡的構建器圖像是標準.NET SDK。 原始的書呆子晚餐代碼具有用於數據庫定義的Visual Studio SQL項目。 我還沒有觸摸過這一點，但是我添加了一個額外的數據庫構建項目，該項目使用社區工具使用新的.NET SDK構建了舊的SQL項目。 構建階段生成DACPAC文件，最後階段從SQL Server映像開始，並在DACPAC中復制和啟動PowerShell腳本。部署邏輯位於Intialize.ps1文件中，該文件使用SQL Server工具來安裝或更新DACPAC數據庫。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

從Docker Hub上的圖像交互式運行書呆子晚餐數據庫容器，您將看到DACPAC部署的輸出。

```
＃運行數據庫，映射另一個客戶端端口：Docker Run -IT -P 1434：1433
Diamol/ch13-nerd-Dinner-DB：2E

＃當您看到日誌時，請使用CTRL-C退出
```

您會在容器日誌中看到SQL Server啟動，然後從DACPAC生成部署SQL腳本，並應用該腳本，從而創建所有數據庫表，索引和鍵。 圖13.9顯示了我的容器與SQL查詢一起運行，顯示了數據庫中表的列表。

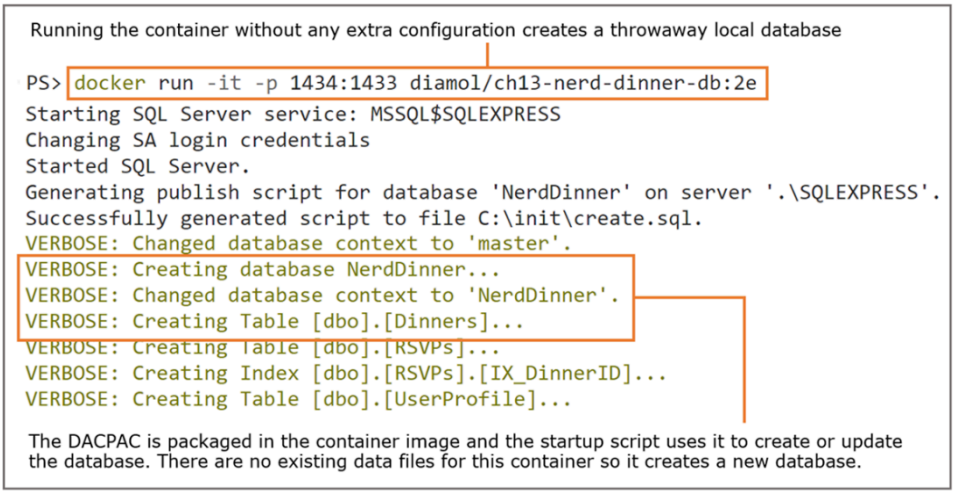


圖13.9運行數據庫容器部署DACPAC，該DACPAC創建所有對象。

現在，我們已經有了應用程序和包裝的數據庫以在Windows容器中運行，這意味著我們可以用Docker Compose建模應用程序。 Windows和Linux容器的組合規範是相同的，我們可以使用第8章中所學的所有功能來描述應用程序的兩個組件，它們的配置設置以及它們之間的關係。清單13.5顯示了書呆子晚餐的撰寫文件：

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

清單13.5與Docker組成的書呆子晚餐建模

```
services:  db:    image: diamol/ch13-nerd-dinner-db:2e    healthcheck:      test: ["C
MD", "powershell", "Test-Path", "/init.done"]      start_period: 30s    networks:
- nd-net  web:    image: diamol/ch13-nerd-dinner-web:2e    environment:      ENV
_DB_CONNECTION_STRING: ... env_ipinfodb_key：... env_bingmaps_key：..
.端口：-8083：80依賴性：db：db：corytion：service_healthy網絡： -  ND -N
ET網絡：ND -NET：ND -NET：
```

我們尚未涵蓋的撰寫規範中只有一個功能，非常適合這樣的舊應用。 數據庫在線之前可能需要一兩分鐘，我們不希望Web容器在此之前嘗試連接。依賴關係是用條件值建模的，將Web容器鏈接到數據庫容器的HealthCheck。 直到數據庫容器啟動並將容器標記為健康後，網絡容器才會啟動。數據庫HealthCheck非常簡單 - 它只是尋找文本文件的存在，當DACPAC部署完成時，初始化腳本會創建該文件：一旦部署了數據庫，就寫了文件，然後將其寫入文件，然後下一個HealthCheck成功並創建Web容器。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

讓我們參加書呆子晚餐吧！您需要註冊第三方服務，並在需要完整的體驗時將密鑰添加到撰寫文件中。其餘的一切都發生在Docker中。

```
＃運行應用程序和數據庫容器：Docker Compose -f。 \ nerd -Dinner \ Docker \ Do
cker -compose.yml -up -d
```

您將在Compose命令的輸出中清楚地看到依賴關係映射 - Web容器將在DB容器啟動時在狀態等待，當DB容器狀態切換到健康時，則Web容器將啟動。瀏覽到http：// localhost：8086，您將能夠使用具有所有功能的舊式Windows應用程序 - 您可以註冊帳戶，創建晚餐並查看已列出。圖13.10顯示了我的Windows台式機上。

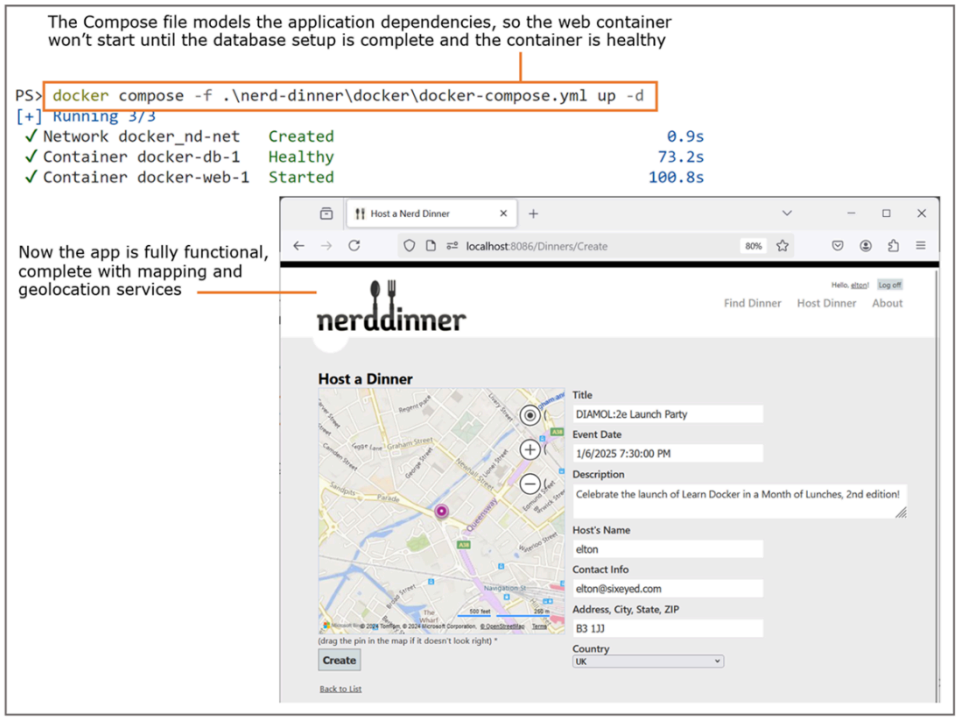


圖13.1　　0完整的書呆子晚餐體驗，在最新的Windows Server中運行　　容器

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

我最後一次說：這是一個用過時的語言版本編寫的Windows應用程序，自2013年以來沒有人訪問過。它是使用docker構建的，沒有任何更改源代碼的更改，並且構建和部署過程與您想在現代容器平台中運行的任何其他應用程序相同。到目前為止，與書籍中的任何其他應用相比，運行此應用程序的唯一區別在於，您需要擁有Windows機器來託管它。

13.5了解窗戶容器適合您的景觀

容器和雲已將許多Windows應用程序，開發人員和管理員吸入Linux世界。您可以將許多舊的.NET框架應用程序遷移到最新的.NET，而無需複雜的重寫項目，如果可以的話，您應該。 可以使用Windows筆記本電腦的開發人員構建Cross Platform .NET應用程序，並在雲中的Linux容器中運行，該容器比Windows Server容器更小，更快且便宜。 但是，並非每個應用程序都很容易遷移，並且很難說服利益相關者是否需要18個月並提供完全相同的功能，可以重寫應用程序的好處。在這種情況下，將應用程序轉移到Windows容器的替代方法是一個有吸引力的選擇。

大型組織可能會擁有他們支持的應用程序目錄，在遷移計劃中，您可以從繪製應用程序格局並根據遷移的目標和業務利益進行分類。 Gartner Research在雲遷移的“ 5 Rs”中提出了一系列類別。 這是人們擴展或重新定義的有用框架，這是我對分類的看法：

- *Retire*。預計將退役的應用程序可以合理地保持原樣，如果可能在不久的將來被關閉或更換的數據中心，則在數據中心的VM上運行。但是，仍然需要確保他們使用，因此，如果他們使用已知漏洞的依賴關係，則可能需要升級到另一個類別。 *Rehost*。這是經典的升降機轉移，在該應用程序中，應用程序不能證明很多投資是合理的，並且可以使用當前基礎架構的緊密近似，在雲中供應VM並以您當前的方式部署應用程序，無論是自動化的還是手動的，可以在雲中運行。 *Replatform*。這是我們在本章中介紹書呆子晚餐的方法。不要更改應用程序，而要構建和建模以在其他平台上運行。 Docker是一個不錯的選擇，因為每個雲都有一個容器服務，您也可以在容器中本地運行該應用程序。這對於非常複雜的應用程序或沒有太多持續開發的應用程序是有益的，因此沒有資格重寫。這是一種低迴報的低投資方法，在具有一致的工具和實踐的現代平台中將您的應用程序移至雲。
-

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

- *Refactor*。進化使用更現代體系結構的應用程序。這是重型形式的不錯的擴展，非常適合更積極地開發並具有更長預期的壽命的應用程序。容器平台是分解單片應用程序的理想方式 - 可以內部分解並移動到在單獨的容器中運行的新組件。您可以逐步執行此操作，並專注於業務價值，切開功能以提高速度，質量或交付節奏。

- *Rewrite*。將所有語言，平台和框架都是一天的風味，將其全部扔掉，然後重新開始，對於開發團隊來說通常很誘人，但很少是最好的方法。應用程序總是比您意識到的要復雜得多，並且重寫中存在巨大的風險：遺漏功能，邏輯的實現方式不同，服務合同被更改。

這些遷移本身以復雜性的上升順序列出，但這並不是整個故事。 重新安置應用程序可能比重新構圖更快地將其升至雲，但是最終結果可能是一個仍然具有手動部署步驟，沒有監視的應用程序，並且無需額外的努力即可將其移至另一個雲。對容器進行替代通常是一個不錯的第一步，它會根據業務價值進行重構。重寫似乎很有吸引力，但是請確保您對他們有良好的理由，並且如果您沒有非常徹底的自動測試套件，請重新考慮。

13.6實驗室

在這里為您提供了不錯的重型實驗室 - 它將進行一些調查，但這始終是Legacy應用程序遷移的途徑。本章實驗室目錄中的MSI安裝程序文件部署了ASP.NET Web應用程序。您的目標是通過Windows Server容器在Docker中正確運行此應用程序。您需要為Web應用程序編寫一個DockerFile，並為運行應用程序和數據庫編寫規格。這是您需要知道的一些事情：

- 該應用連接到SQL Server數據庫。空數據庫很好，但是應用程序配置已固定在MSI中。它希望在DNS地址INBINP-DB-DEV01上找到數據庫，並使用密碼Dockercon使用SA用戶帳戶！！！ 這是一個將使用IIS Windows服務（稱為W3SVC）運行的Web應用程序，但需要安裝.NET Framework 4.8。 該應用程序將日誌寫入C：\ logs \ imbisup.log上的文件。我們希望那些作為容器日誌發布的人 - 您需要在部署的一部分安裝LogMonitor工具。實驗室文件夾中有一個LogMonitorConfig.json，該文件已正確配置，您需要將其複製到Docker映像中並設置正確的入口點。 部署MSI是最簡單的部分 - 它不需要傳遞的任何配置設置；您只需使用MSIEXEC工具安裝它即可。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

- 當您運行所有內容時，您應該能夠通過http：// localhost：8088/Invicup瀏覽該應用程序，並查看新聞通訊註冊頁面。如果單擊“註冊”按鈕並填寫詳細信息，則應在Web容器中看到日誌，數據將存儲在SQL Server容器中。

與往常一樣，實驗室文件夾中有解決方案文件，並且在readme文件中有更多詳細信息：https：// github。 com/sixeyed/diamol/blob/2e/ch13/lab/readme.md。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

14 在Microsoft Azure和Google Cloud的雲中的容器

Docker最早的口號是“在任何地方建造，運送並運行任何應用程序”。 我們已經在本書中使用了許多不同的應用程序介紹了構建，運輸和運行工作流程，但是到目前為止，我們一直在機器上運行集裝箱。是時候查看原始承諾的*anywhere*一部分了，為此，我們將使用雲中的託管容器服務來關注運行應用程序。 這些是簡化的容器平台，可讓您在容器中運行應用程序，而無需提供VMS或安裝Docker或管理網絡，存儲或其他任何內容。每個雲的方法略有不同，並且每個雲都有自己的建模語言和工具 - 因此您不會使用Docker或組成命令行。但是，鑑於您已經知道的一切，學習曲線很淺，您將獲得很少的努力，可以訪問大量功能。

在本章中，我們將使用一些最有吸引力的雲服務：Azure容器實例，Azure容器應用程序和Google Cloud運行。 AWS具有類似的服務，稱為Fargate，該服務在彈性容器服務上運行，但使用要復雜得多，因此，如果您是AWS用戶，我將離開您探索。 您將需要註冊Azure和Google Cloud才能遵循本章中的示例，但是兩種雲都為新帳戶提供免費的積分，這將足以涵蓋練習，並且您將真正從不同雲中的容器中獲得動手實踐經驗而從中受益。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

14.1託管容器服務適合

我在上一章中說：“每個雲都有一個容器服務”，但實際上，主要雲具有*multiple*容器服務。 微軟，Google和亞馬遜作為其旗艦容器提供了圍繞Kubernetes的合併，但是很多用例不需要kubernetes的完整功能集和開銷，因此它們還提供了更簡單的替代方案。這些託管容器服務的主要目標是提供與Docker開發人員工作流非常相似的生產部署工作流程：您運行命令以啟動容器，將其傳遞給使用圖像以及任何配置設置，也就是這樣。這比在雲中的VM上使用普通碼頭要好得多。 您無需提供或維護任何基礎架構，並且可以從開箱即用的可靠性中獲得可靠性 - 該服務將在停機時可用。

那麼，適合這些服務的用例是什麼？關鍵屬性是簡單性：沒有具有許多依賴服務的分佈式體系結構的應用程序；具有適度縮放要求的應用程序；不需要五個可靠性和詳細監視的應用程序。這並不意味著託管容器服務僅對基本的待辦事項應用程序有用，如果產品所有者對其非功能性要求誠實，則一組應用程序將屬於簡單類別。在雲遷移程序中，您將需要確定哪些應用需要完整的容器平台的功能集，並且可以使用更簡單的託管服務。您可能會更快地轉向雲，並且在託管容器服務中運行的應用程序的總擁有成本較低。

現在嘗試

讓我們看看使用Azure容器實例（ACI）有多容易。 您將需要Azure訂閱和安裝Azure CLI。 AZ容器命令管理容器；您可以添加 - 螺旋 - 到任何命令以打印幫助文本。

```
＃登錄到您的Azure訂閱：AZ登錄


＃創建一個資源組 - 一個用於Azure服務的邏輯組：AZ組創建 - 位置Eastus -nam
e Diamol -CH14


＃創建一個容器實例以運行一個簡單的Hello World應用程序：AZ容器創建-G Diamol-Ch14 -name H
ello -name Hello -image ghcr.io/diamol2e/ch14- Hello-cloud -os-type-type linux-cpu 0.1--memory 0.1-me
mory 0.1


＃打印容器日誌：AZ容器日誌-N Hello -G Diamol -
CH14
```

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這就是您需要做的。 AZ容器創建命令需要幾秒鐘才能返回 - 到那時該容器正在運行。 Azure發現並保留了一些計算能力（我們只要求少量），拉動圖像並啟動了容器。您可以使用Azure CLI與容器進行交互並打印日誌。您也可以使用Azure門戶瀏覽ACI實例，在那裡您可以看到運行容器的詳細信息，其生命週期中的事件以及日誌的另一種視圖。我的輸出如圖14.1所示。

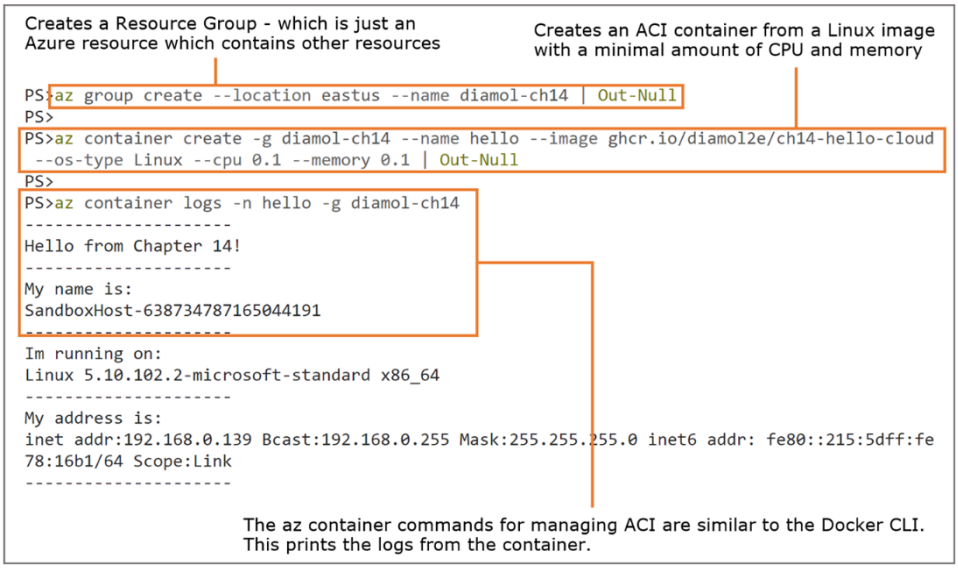


圖14.1運行帶有Azure容器實例的簡單容器 s

與Docker運行相比，容器創建命令需要更多的信息 - 需要使用的計算量以及需要容器映像的操作系統 - 但這仍然非常簡單。這可能非常適合在CI/CD過程中部署測試環境。 運行速度是如此之快，便宜，以至於您可以在源代碼中的每個分支機構中構建和部署，從而為開發人員提供了自己的煙霧測試設置，可以在推動更改的一分鐘內提供。當然，如果他們可以訪問該環境，那麼在Web應用程序或API服務器的情況下，這將有所幫助。 ACI支持發布端口，並可以為您的容器提供DNS路線，從而在公共互聯網上提供。

現在嘗試

您可以使用帶有一些額外參數的同一容器創建命令來發布HTTP端口，並將Azure路由流量從公共URL到容器中。您需要為容器選擇全球唯一的DNS前綴，該前綴將發佈到Azure域。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
#選擇一個好的DNS名稱並放入變量 - 帶有PowerShell：$ dns_prefix ='<


#或linux shell：
dns_prefix ='< unique-name >'

#現在使用您設置的DNS前綴創建容器：AZ容器創建-g diamol-ch14 -name hello-web -image ghcr.io/diamol2e/ch14-ch14--hello-cloud-web-ports 80 -dns-name-label $ dns-label $ dns_prefix -os-os-os-os-os-os-os-os-os -type-cpe-cpu-cpu--cpu--memememe


#查詢ACI打印完整的域名，該名稱將路由到容器：AZ容器顯示-G Diamol -Ch14 -n Hello -Web -Query'ipaddress.fqdn'
```

容器Show命令的輸出將是您實例的完全限定的域名。 您可以瀏覽到URL，然後看到一個基本的靜態Web應用程序（這使用Apache Web服務器）。 此設置中沒有SSL，這不是ACI開箱即用的功能。有可以保護訪問權限的選項，但這只是簡單的HTTP。我的輸出在圖14.2中。

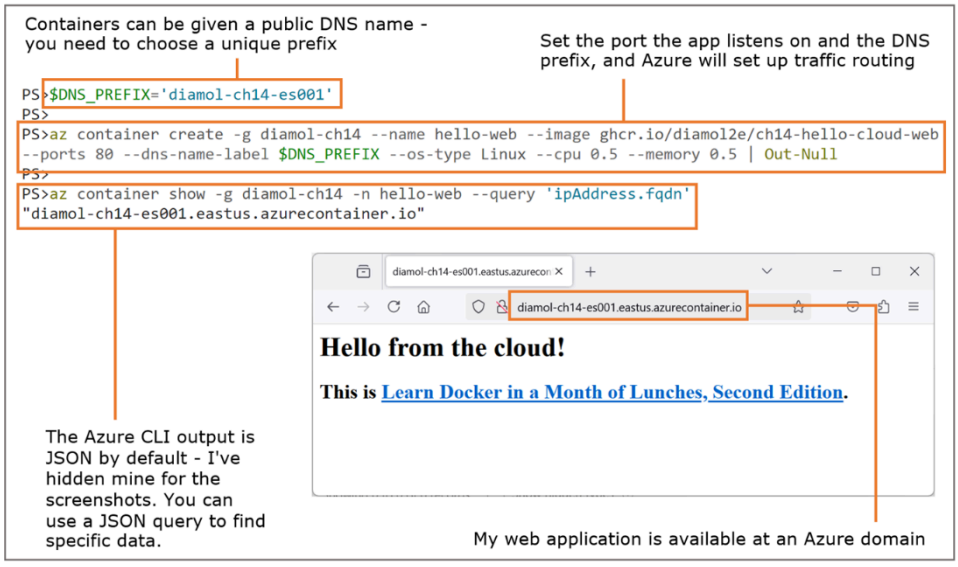


圖14.2發布帶有ACI的公共DNS名稱的網絡容器

© Manning Publications Co. To comment go to liveBook
許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

您與Docker一起使用的大多數容器選項都可以在託管容器服務中使用。 您可以設置用於配置，附加存儲，連接到網絡等的環境變量。 雲提供商通常支持Linux和Windows Server容器，並且可以使用對ARM處理器的支持。但是，雲容器服務的真正力量是它們與平台的其餘部分很容易集成。您可能擁有一個使用SQL數據庫，消息隊列和REST API的Web應用程序。開發人員可以在不同的前端組件上工作，但是他們都可以使用一組共享的後端服務。在這種情況下，您的測試體系結構看起來像圖14.3。

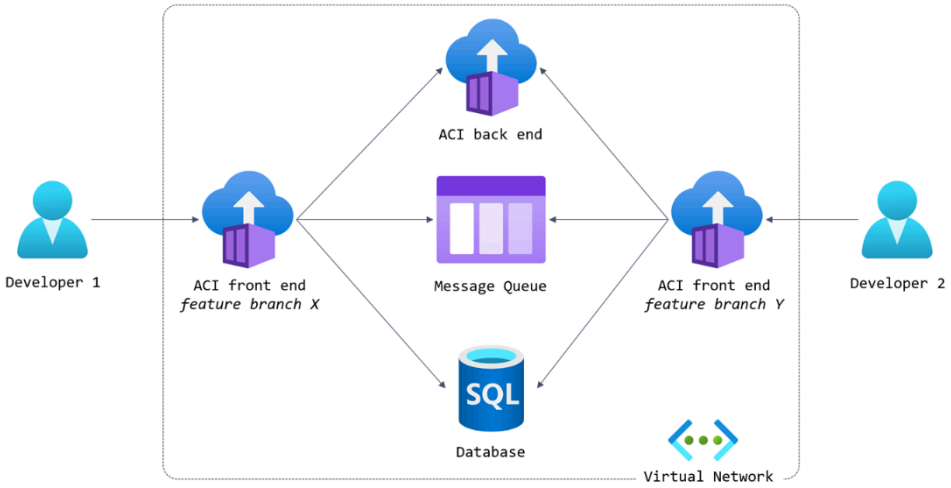


圖14.3使用共享服務和ACI容器託管多個測試環境

這裡的服務是特定於Azure的，但是其他雲的概念是相同的：您的後端是在充當安全邊界的虛擬網絡中隔離的。 數據庫和消息隊列在公共Internet上無法訪問，但是可以通過連接到虛擬網絡的容器實例來達到它們。有一個共享的API容器，每個開發人員都有自己的Web容器從代碼的分支中構建。這是一種非常有效的工作方式，並且非常容易自動化 - 容器的DNS名稱可以使用源代碼中的分支名稱，因此創建新的分支機構時不需要任何手動干預。 您可以有另一個管道來卸下容器，當分支被刪除時觸發。

在我們繼續前進之前，還有一個簡單的示例。成功的雲遷移項目幾乎總是隨後是一個成本優化項目，通常大約一個月後，第一個雲賬單匯集了。單個雲服務的價格往往會吸引人，但是當您運行數十或數百個雲服務時，成本很快就會增加。 在您部署到雲的所有內容中，您需要了解影響成本的部署選項。 對於容器服務，主要成本是為您的容器保留的計算大小。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

我們為少量計算提供了ACI容器。現在，我們將運行同一應用程序的兩個實例（一個計算PI的Web應用程序），並具有不同量的計算。我們將比較它們，以了解它們的表現。

```
＃選擇一個不同的DNS名稱並放入變量 - 使用PowerShell：$ dns_prefix ='< unique -name >
'

＃或linux shell：dns_prefix ='< un
ique-name >'

＃創建一個具有0.1 CPU內核的容器：
az container create -g diamol-ch14 --name pi-1 --image ghcr.io/diamol2e/ch14-pi- web --ports 80 --dns-name
-label "${DNS_PREFIX}-1" --os-type Linux --cpu 0.1 -- memory 0.1 --query 'ipAddress.fqdn'

＃和帶有3.5 CPU內核的同一圖像的圖：AZ容器創建-g diamol-ch14 -name pi-2  -  image ghcr.io/diamol2
e/ch14-pi- web-ports 80 -dns-name-label “ $

＃瀏覽兩個域中的每個域，並迅速刷新幾次
```

我的輸出如圖14.4所示，在該圖14.4中，我將兩個站點都以重複的刷新置於壓力下。在計算PI時，第一個具有最小CPU的容器比第二個容器要慢。這是一個計算密集型工作負載，顯示了您如何在性能和成本之間找到平衡。您僅在運行時支付這些容器的費用，但是團隊無法旋轉一些測試環境並讓它們運行24x7，即使它們每天只使用幾個小時。您應該確保知道哪些參數會影響成本，並確保您的部署尺寸。 然後考慮一下，如果不始終需要，您如何按需提供環境。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

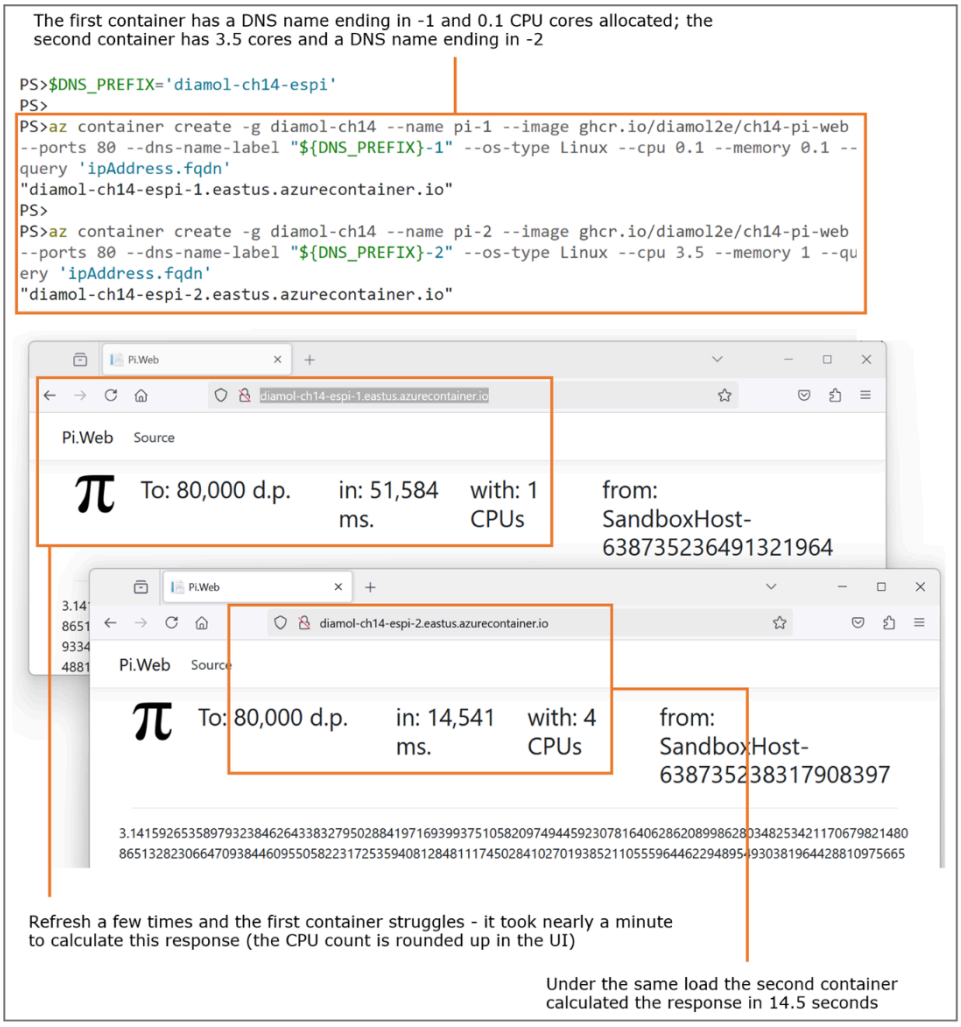


圖14.4 ACI中的計算大小直接影響應用程序性能和您的Azure賬單。

您可以將直接的應用程序部署到ACI，而無需學習比我們在本節中涵蓋的更多內容 - 該服務確實很容易使用。但是您不會做的一件事是將圖像發佈到公共註冊表中，以便您可以在雲中運行容器。那將無法治療您公司的知識產權。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

14.2私人註冊機構與Docker Hub

託管容器服務可讓您使用開發人員的工作流進行生產部署，但是工作流的“船”和“運行”部分還有一個額外的元素：您也需要使用託管在雲中的圖像註冊表。您從第5章中知道，註冊表是標準化的，並且可以在Docker容器中運行。 雲提供商還具有容器註冊表服務，因此您可以將應用程序圖像推到雲中的私人註冊表，該註冊表物理靠近您的容器。這使您可以快速的圖像拉動和保護註冊表的能力，因此僅適用於您的服務。

AWS具有彈性容器註冊表，Google具有工件註冊表，Microsoft具有Azure容器註冊表。從根本上講，它們都像Docker Hub一樣行為 - 您可以使用Docker命令行對圖像進行身份驗證，推動和拉動圖像。 但是它們都具有其他功能，例如與雲的身份提供商集成的自動安全掃描和訪問控制。每個雲對其註冊表服務也有不同的看法 - Google Artifact Registry旨在存儲所有類型的工件，而不僅僅是容器圖像。 ECR與供應商應用的AWS市場集成。 ACR旨在支持應用程序生命週期，並具有自動化的任務來構建圖像，並在觸發事件上重建它們，例如更改基本圖像。在本節中，我們將獲得ACR的味道。

現在嘗試

ACR Instanes可以使用Azure域獲得DNS條目 - 當您創建一個域時，您需要提供一個全球唯一的名稱。在您的資源組中創建註冊表，並使用Azure命令行登錄。

```
＃選擇一個DNS名稱並放入變量 - 使用PowerShell：$ acr_name ='< unique -n
ame >'

＃或linux shell：acr_name ='<
unique-name >'

＃現在使用最便宜的層創建註冊表：Az ACR CREATE -N $ ACR_N
AME -G DIAMOL -CH14 -SKU BASIC

＃登錄到ACR-此配置Docker CLI中的身份驗證：AZ ACR登錄-N $ ACR_NAME
```

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

這是您的自定義子域的私人註冊表。登錄命令在場景後面設置一個限制的令牌，因此它在一段時間後到期，如果以後會遇到身份驗證錯誤，您將需要再次登錄。我的輸出在圖14.5中；這就是您需要做的一切，因此現在您可以用ACR域名標記圖像並將其推到Azure。

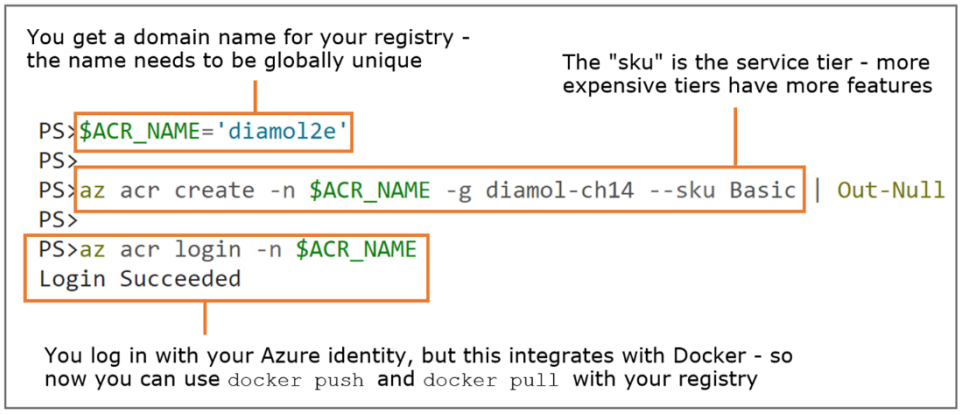


圖14.5使用Azure命令行創建和身份驗證ACR

您可以在訂閱中創建多個ACR的實例，這是您通常會做的 - 您可能有一個註冊表來存儲非生產環境的圖像，而另一個用於生產圖像。 您的CI/CD管道將構建並將圖像推向非生產存儲庫，並從那裡部署到測試環境。當您準備好釋放候選者時，第二個管道將測試的圖像推向生產註冊表，並將其部署到預生產的分期環境中。使用單獨的註冊表意味著您可以獨立保護它們並具有不同的服務層 - 節省了更昂貴的生產選擇。

現在嘗試

我們只有一個ACR，但是我們可以看到如何使用Docker Hub的圖像從另一個註冊表中復製圖像。

＃使用Azure工具從Docker Hub導入圖像：AZ ACR導入-n $ ACR_NAME -source Docker.io/diamol/apache: 2e -t diamol/apache：2e＃檢查圖像清單：Docker Suvinest Inspercest Inspercest Inspercest Inscect Inscect Ins cect Inscept of COMPLEST“ $ {acr_name} .azurecr.io/diamol/diamol/diamol/apache：2e：2e：2e： “ $ {a cr_name} .azurecr.io/diamol/apache：2e”

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

本節中的AZ命令對您來說可能是新的，但是Docker命令將非常熟悉。現在，您可以在自己的私人圖像註冊表中擁有一個存儲庫，您可以在其中使用不同的標籤安全地推動和拉動任意數量的圖像。當您運行此操作時，您會看到Docker Hub上的源存儲庫是一個多平台圖像。 ACR具有多個平台支持，並且在所有圖像變體中的導入命令副本 - ACR中的清單列表在圖14.6中。

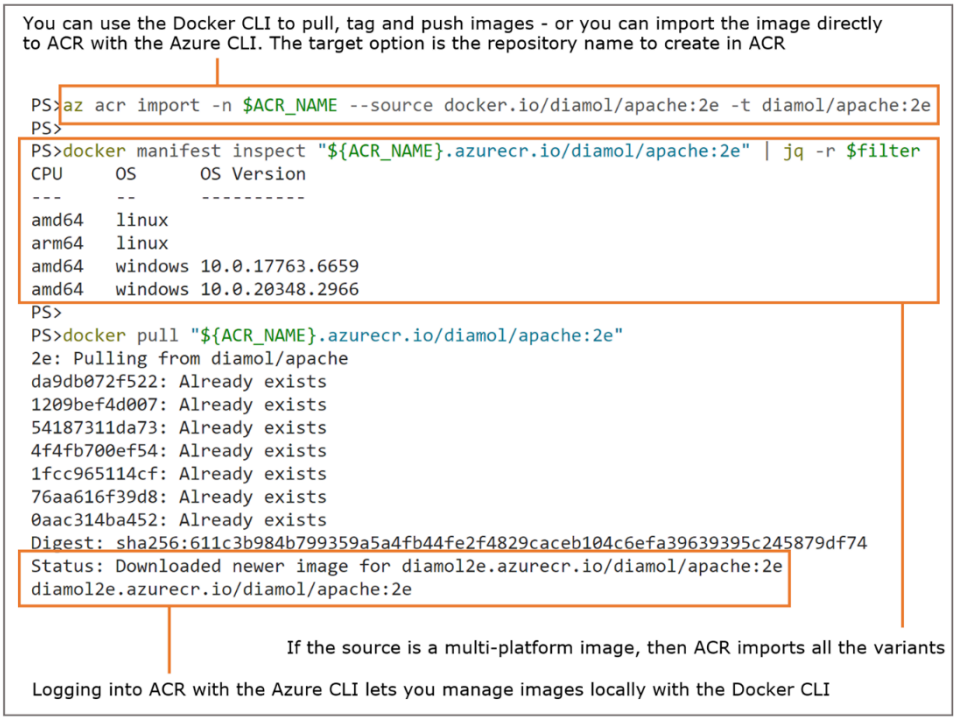


圖14.6使用Azure命令行將圖像從Docker Hub加載到ACR中

記住雲服務花費了金錢，您需要知道ACR的定價基於您選擇的服務層（更多功能成本），還基於您使用的存儲量和傳輸的數據量。 從每個提交中構建和推動圖像很誘人，您絕對可以做到這一點。但是，您需要一個自動清理過程來刪除未使用的圖像，因此您需要考慮使用的標籤以及如何確定哪些構建是用於保留的。不要讓自己依賴不普遍可用的功能也是一個好主意。 ACR沒有Docker Hub（所有者/應用程序）的嚴格兩部分存儲庫計劃，因此您可以決定使用自定義命名系統。 這意味著您可能無法在不更改所有名稱的情況下移至另一個註冊表。 Microsoft使用其.NET框架圖像自己完成此操作 - 您可以將mcr.microsoft.com/dotnet/framework/sdk:4.8拉動，標記它以用您的ACR域替換MCR域並成功推動。 但是，您無法將圖像推向Docker Hub，因為它在存儲庫名稱中具有三個部分。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在，您已經設置了ACR設置，我們可以返回ACI，看看如何從私人註冊表中部署。我們還將研究將圖像進入ACR的另一個選項。雲提供商喜歡使盡可能容易地消費其服務，因為它們越容易，您就越有可能使用並繼續使用它們。 這意味著工具之間通常會重疊，這些工具提供了不同的做同一件事的方式。 您可以使用Docker命令行構建圖像並將其推向ACR，也可以使用ACR的自動化功能在Azure中構建圖像。 ACR Build命令與Docker Build一樣工作非常有效 - 它壓縮包含您的Dockerfile和源代碼的本地文件夾，並將其發送到Docker Engine以構建圖像。 使用ACR，使用Azure Compute進行構建，並將結果圖像存儲在您的註冊表中。

現在嘗試

使用ACR構建本章的簡單網站圖像的版本。這是基於Alpine Linux，因此基本圖像很小。 與當地的Docker機器相比，查看ACR構建所需的時間多長。

```
CD CH14/練習/Hello-Cloud-Web

＃在本地構建：
Docker Build -T Diamol/ch14 -Web -quiet。

＃並在ACR中構建相同的圖像：AZ ACR Build - Image Diamol/ch14-Web -Re
gistry $ {acr_name}。

＃如果您使用＃免費積分，則可能會得到一個taskSoperationsNotallowed錯誤 - 如果您只需導入圖
像：AZ ACR Import -n $ ACR_NAME -source ghcr.io/diamol2e/ch14-hello-cloud-web -t web -t diamol
/ch14-web

＃打印新圖像存儲庫的標籤列表：AZ ACR存儲庫顯示標籤-n $ {acr_name} - Repository D
iamol/ch14-Web
```

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

ACR命令運行的任務將作品卸載到Azure，並且構建並不比本地Docker Build更長。您可以為每個ACR任務獲得一個乾淨的構建代理，因此，如果您有一個大的基本映像，則每次都需要拉動構建代理，因為每個代理都以新的圖像緩存開始。在CI/CD管道中，這可能不是一個問題，尤其是在一個有很多攪拌的項目中，您將受益於並行構建不同分支的圖像。 像其他情況下，雲提供商提供了替代工具和流程一樣，您需要確定簡單性是否值得遠離普通工具集。 您在圖14.7中看到我的輸出； 現在，我準備在我的私人註冊表中使用的圖像，該圖像甚至無需在我的機器上使用docker而建造。

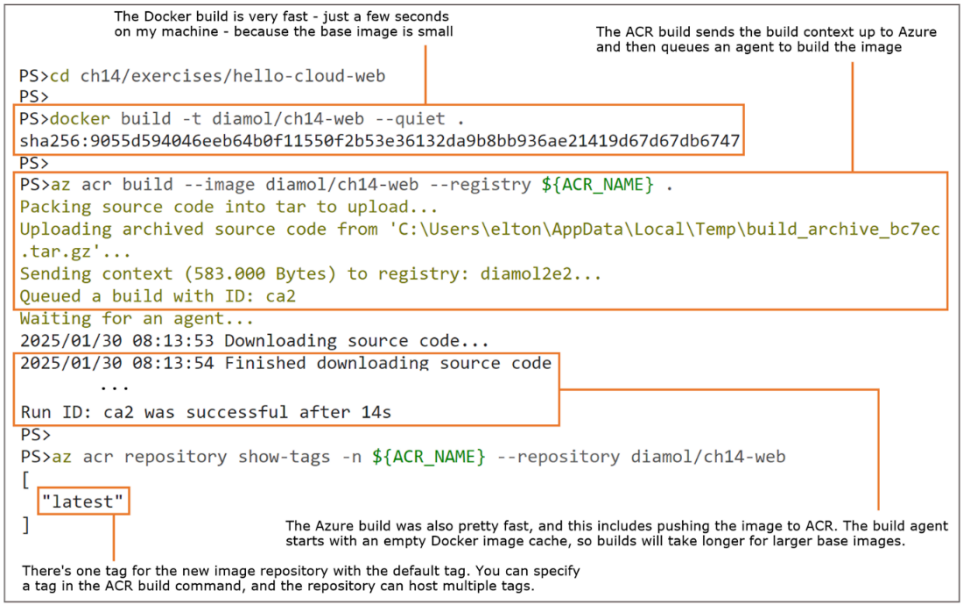


圖14.7　　使用ACR任務從本地SOU中構建和推動Azure中的圖像　　RCE文件夾

您可以使用來自任何註冊表的圖像在ACI中運行容器。 我們已經在GitHub容器註冊表上使用了公共圖像，而該圖像不需要身份驗證，但是要使用我們的私人圖像，我們將需要提供憑據。有很多選擇要這樣做；最安全的與雲的身份服務集成在一起，因此您無需傳遞部署腳本中的任何敏感數據。 這些需要更多的設置，並且是針對雲提供商的特定設置，因此我們將堅持最簡單的選項，這也將向您展示如何使用用戶名和密碼登錄ACR。

現在嘗試

在ACR中創建一個令牌以用於身份驗證，該驗證為您提供用戶名和密碼，並使用它們從ACR中的圖像中運行ACI容器。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
#生成ACR令牌並打印密碼：AZ ACR令牌創建-n aci Reader -r $ acr_name -repository diamol/ch14
-web content/read metadata/read read-query'credentials.passs.passs.passwords [0] .value".value"



#想想一個新的唯一DNS前綴並存儲它 - 在PowerShell中：$ dns_prefix ='< new -n
ame >'


#或linux shell：dns_prefix ='
< new-name >'


#並運行ACI映像 - 用戶名是您的acr token名稱#，您需要插入令牌密碼：AZ容器創建-G diamol-ch14 -na
me web-2 - image“ $ {acr_name} .azurecr.io/diamol/diamol/diamol/diamol/diamol/diamol/ch14-web” -ports
80 -dns-dns-liniund liniund liniunc liniusd yneme-lins-dns_dnem lin -cpu 0.2 - 示例0.2 - registry-login-server
“ $ {acr_name} .azurecr.io” - registry-username aci-reader -registry-password'< token-password'< token-pass
word >'




#打印完整域：
AZ容器顯示-G diamol-ch14 -name web-2 -query'ipaddress.fqdn'

#現在您可以瀏覽到網站
```

您必須在ACI Create命令中指定很多細節，因為Azure CLI不假定默認值很多。 這就是我們設置的 - 如果圖像不可公開訪問，則只需要包含註冊表信息：

- -g（或-group）是資源組-name是ACI實例的名稱-Image是docker image從-ports列出運行容器的docker映像，列出了port（s）以發布-dns-name-label是用於容器公共DNS的前綴
- 入口
- -os-type是操作系統（Linux或Windows）-CPU和-Memory指定計算資源為分配-Registry-login-server是註冊表的完整域名-Registry-username是身份驗證的用戶名 - registry-password是密碼

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

我的輸出在圖14.8中。 這是相同的基本網站，但這裡的工作流程是這裡的重點 - 該容器是從ACR中的安全圖像運行的，我是通過本地源代碼構建的。而且不要期望使用我的令牌或我的ACR域 - 在閱讀本文時，這會很早就消失了。

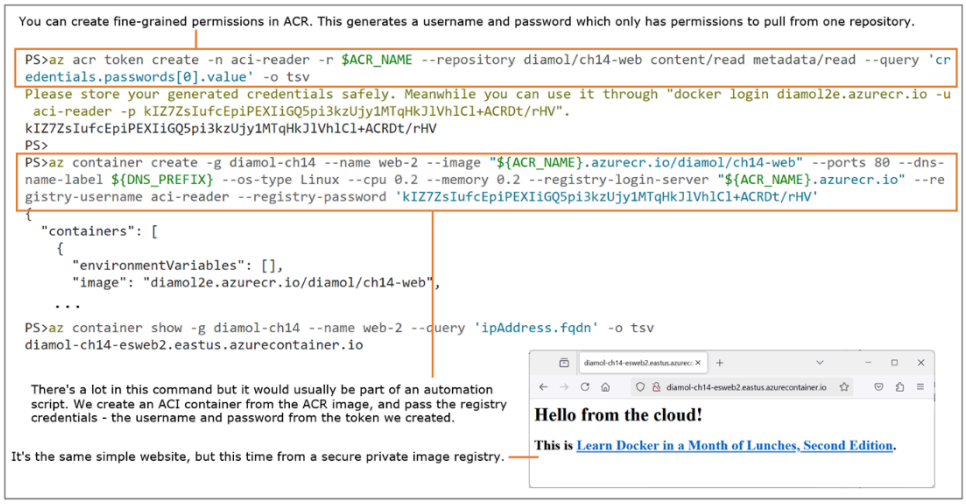


圖14.8從我們內置的ACR中的安全圖像中運行ACI的容器

我們現在已經完成了ACR和ACI。您可以刪除刪除其中所有資源的資源組；他們將永遠消失。

現在嘗試

您將被指控（或用來使用免費的ACR和ACI）。 除非您想繼續進行實驗，否則請刪除資源組。

```
#-Yes Flag停止CLI要求確認：AZ Group Delete-Name Diamol-CH14-
是的
```

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

本節中的許多練習都有樣板設置工作，例如創建Azure資源組和ACR並生成令牌。 如果您使用的是Azure，那麼這些將不會成為日常工作流程的一部分 - 您將在項目的開始以及創建新環境時進行一次，然後每隔幾個月生成代幣。日常工作流程可能就像AZ ACR Build和AZ容器創建一樣簡單，這使您可以在Azure中進行局部剪切，以準備進行測試。這可能是將簡單應用程序在雲中取得的有用方法，但是當您到達更複雜的應用程序時，從託管的容器服務到Kubernetes，學習曲線非常陡峭。當您需要多個容器，但不需要kubernetes的全部重量時，有一些雲服務位於兩者之間，並且通過支持Docker撰寫格式來加速部署。

14.3 Azure容器應用程序

Docker Compose被設計為一種通用規範，以建模在容器上運行的應用程序。您已經學習瞭如何與Docker一起使用Complose來運行容器並在本地構建圖像，並且可以與某些雲容器服務一起使用相同的規範。 AWS支持使用ECS-CLI工具組成文件的彈性容器服務部署，而Azure擁有一項專用服務，稱為Azure Container Apps（ACA），該服務具有本機組成支持（Google Cloud沒有具有組合支持的容器服務）。 ACA是一個完全起步的平台，它支持高級功能，例如自動尺度和比例為零，部署到分期插槽，並與ACR緊密集成。值得探索ACA，因為它為工具包增加了重要組成部分 - 如果您打算使用Azure，您可以使用一套容器服務，您可以將其映射到應用程序景觀：最簡單的應用程序的ACI，ACA的典型應用程序，可以為Compose而建模的ACA，並且可以為Azure Kubernetes Service（AKS）進行建模，以適用於最大的配置應用程序。

清單14.1顯示了我們在書中已經使用的隨機數應用程序的撰寫文件。它有兩個容器 - 一個Web應用程序和一個REST API，並且都在公共Docker Images中可用。 規格中沒有什麼不尋常的：容器連接到同一網絡，每個網絡都發布端口，並且Web應用程序使用環境變量來設置訪問API的URL。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Listing 14.1 A simple Docker Compose specification

```
services:
  numbers-api:
    image: ghcr.io/diamol2e/ch08-numbers-api:2e
    ports:
      - "8081:8080"
    networks:
      - app-net

  numbers-web:
    image: ghcr.io/diamol2e/ch08-numbers-web:2e
    environment:
      - RngApi__Url=http://numbers-api:8080/rng
    depends_on:
      - numbers-api
    ports:
      - "8080:8080"
    networks:
      - app-net

networks:
  app-net: {}
```

您可以使用此組合文件在計算機上或云中的VM上使用Docker運行該應用程序。 但是ACA比VM更好：它將為您提供所有基礎架構，並連接其他功能進行操作和監視，並且可以通過完全相同的組合文件來完成所有功能。

現在嘗試

使用Docker組合文件在ACA中運行隨機數應用程序。我們將使用一個新的資源組來使以後更容易清理，但是首先，您需要確保為ACA設置Azure訂閱和CLI。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
＃ACA是一項相對較新的服務 - 將其設置為：AZ擴展名添加-name Container
App -upgrade AZ Provider寄存器-N Microsoft.App.App- WAIT AZ PROVITER
寄存器-N Microsoft.Microsoft.OperationalInsights  -  wait -wait



＃創建一個新的資源組（任何位置都可以做）：AZ組創建 - 位置Eastus
-Name Diamol-CH14-ACA

＃切換到練習目錄：CD CH14/練習/數字



＃部署到ACA-這將需要一段時間，請務必觀看日誌：AZ ContainerApp撰寫-G diamol -ch14 -aca -environ
ment Numbers -F docker- docker- compose.yml



＃現在列出您的容器應用程序以顯示URL：AZ ContainerApp
List -G Diamol -CH14 -ACA -O表

＃測試API：
＃curl https：// < number-api-domain >/rng

＃瀏覽到https：// < numbers-web-domain >以測試網絡                    ITE
```

當您關注此練習時，會發生很多有趣的事情。您會看到正在創建的Azure資源，並沒有要求這些資源，並警告撰寫的部分規格，這些規範會跳過ACA架構的翻譯。 您可以忽略所有這些。 額外的資源有助於使其成為生產級的部署，並為您設置了監視和集中的日誌集合。 不翻譯的組成規格的部分是在ACA中（例如Docker Networks）中沒有意義的部分（儘管在同一容器應用程序環境中的容器應用程序之間存在網絡設置）。 我的輸出在圖14.9中；您可以看到隨機數API工作，但是當您嘗試使用Web應用程序時，您會遇到錯誤。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

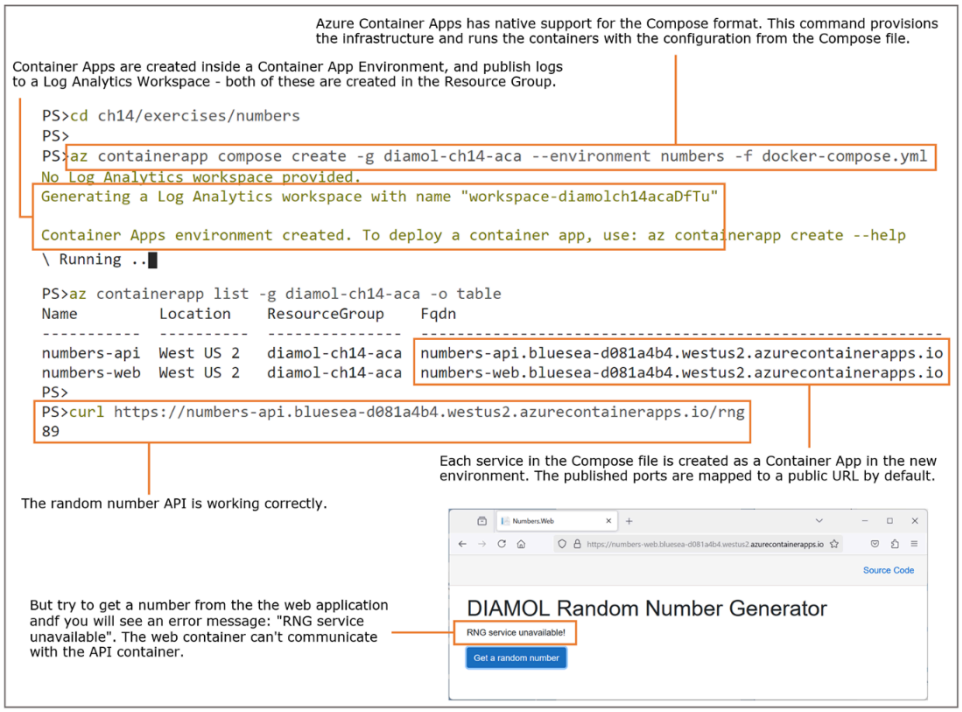


圖14.9　　運行使用Azure Co以組合建模的分佈式應用程序　　ntainer應用程序

損壞的Web應用程序是容器通信的典型問題 - 服務不能神奇地發現對方 - 它們需要使用一個地址。 Web組件從環境變量讀取API的地址。該值設置在撰寫文件中，並與Docker一起使用，可以從Docker Network（Numbers-API）內的名稱到達API容器。這不是Azure中API的地址，因此我們需要更新Web容器應用程序的配置。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

打印Web容器的當前配置設置，然後更新API地址並查看ACA如何實現更改。

```
＃打印當前環境變量：AZ ContainerApp Show -G Diamol-CH14-ACA -N Numbers
-Web -Query'properties.template.Containers [0] .ENV'


＃更新API地址 - 確保使用ACA域：AZ ContainerApp Update -G Diamol-CH14-ACA -N數字-
Web -set-env-vars'rngapi__url =


＃ACA通過創建新的修訂來實現更改：AZ ContainerApp修訂列表-G DIAMOL -CH14 -ACA
-N數字-Web -o表


＃當新修訂健康時，請再次嘗試您的網絡URL
```

那使事情有效。 如果您希望應用程序中的所有容器都有一個公共地址，則可以採用這種方法 - 在這種情況下，客戶可以直接使用該網站或API。 Azure容器應用程序為環境生成DNS子域，並且每個容器在該域中獲取一個地址。 您還將看到URL使用HTTP，我們可以在ACA中免費設置HTTPS，映射組合文件中指定的容器端口到標準端口443。

© Manning Publications Co. To comment go to liveBook

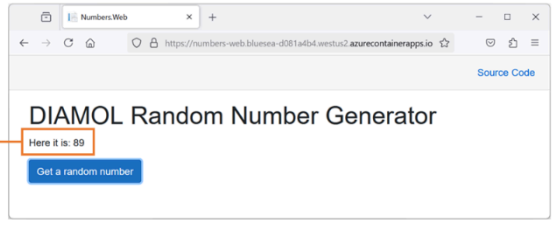
Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Any environment variables set in the Compose file are carried through to the Azure Containers App. This is the address the web app uses to find the API container.

```
PS>az containerapp show -g diamol-ch14-aca -n numbers-web --query 'properties.template.c
ontainers[0].env'
[
  {
    "name": "RngApi__Url",
    "value": "http://numbers-api:8080/rng"
  }
]
PS>
PS>az containerapp update -g diamol-ch14-aca -n numbers-web --set-env-vars 'RngApi__Url=
https://numbers-api.bluesea-d081a4b4.westus2.azurecontainerapps.io/rng'
- Running ..
PS>
PS>az containerapp revision list -g diamol-ch14-aca -n numbers-web  -o table
CreatedTime                 Active    Replicas    TrafficWeight    HealthState    Provisi
oningState    Name
--------------------------  --------  ----------  ---------------  -------------  -------
------------  --------------------
2025-01-31T12:54:20+00:00   True      1           0                Healthy        Provisi
oned          numbers-web--99c0c8o
2025-01-31T12:55:14+00:00   True      1           100              Healthy        Provisi
oned          numbers-web--4vnfawp
```

We could update the Compose file and redeploy, but it's better to manage that in ACA - this updates the environment variable value to use the correct API address.

You know you can't change an environment variable in a running Docker container, and the same is true in ACA. The change is effected by creating a new container - called a *revision* - and shifting all the traffic to that container when it is healthy.



And the app works correctly now

圖14.10更新ACA中的應用程序配置設置以啟用容器之間的通信

ACA還有更多的內容，但是我們將使用另外兩個功能來完成對它的研究：從源代碼構建作為部署的一部分，並支持並非所有組件都應公開訪問的分佈式體系結構。為此，我們將返回到良好的舊待辦事項列表應用程序，我們已經為此撰寫了該練習。有兩個組件：Web應用應該是公共的，而後端數據庫應該是私人的。清單14.2顯示了Web組件的規範。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Listing 14.2 The to-do web application, configured in Compose

```
todo-web:
  image: ch14-todo-web
  ports:
    - "8082:8080"
  environment:
    - Database__Provider=Postgres
  build:
    context: .
  depends_on:
    - todo-db
  networks:
    - app-net
  volumes:
    - ./config/secrets.json:/app/config/secrets.json
```

您從未見過。要注意的關鍵是“構建”部分，該部分告訴撰寫在哪裡可以找到構建上下文（包含Dockerfile的文件夾和應用程序的源代碼），以及將JSON文件加載到應用程序配置文件夾中的捲安裝。 您可以運行Docker組成的構建，然後從練習文件夾中構成Docker，以查看本地運行的應用程序。我們可以使用相同的組合文件使用Azure容器應用程序構建和運行。

現在嘗試

刪除數字應用程序，然後從源代碼部署待辦事項應用程序。這次，ACA還將為其構建的圖像提供一個容器註冊表。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
# start by deleting the previous apps and environment: az containerapp delete -g diamol-ch14-aca -
n numbers-web --yes --no-wait az containerapp delete -g diamol-ch14-aca -n numbers-api --yes --n
o-wait az containerapp env delete -g diamol-ch14-aca -n numbers --yes --no-wait



＃切換到源文件夾：CD CH14/練習/待
辦事項列表


＃部署應用程序 -  CLI在撰寫文件中看到構建規範：AZ ContainerApp costables coles cr
eate -g diamol -ch14 -aca -eenvironment todo


＃如果您的應用程序無法建立在免費積分上，請改用我的圖像：＃AZ ContainerApp Cosece create -G di
amol-ch14-aca -environment todo -f docker- docker- compose-public.yml



＃列出容器應用程序URL AZ ContainerApp列表-G diamol -ch1
4 -aca -o表＃瀏覽到Web域 - 該應用程序不起作用
```

當您創建此容器應用程序時，幕後發生了更多事情。 組合文件中的構建部分觸發了一個新的工作流程，該工作流程已提供了Azure容器註冊表，並使用ACR任務來構建容器映像。創建了另一個日誌分析工作空間，用於監視，以及部署兩個容器應用程序的新容器應用程序環境。數據庫和Web應用程序都有公共DNS條目，但是當您瀏覽網站時，您只會看到一個空頁面，如圖14.11所示。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

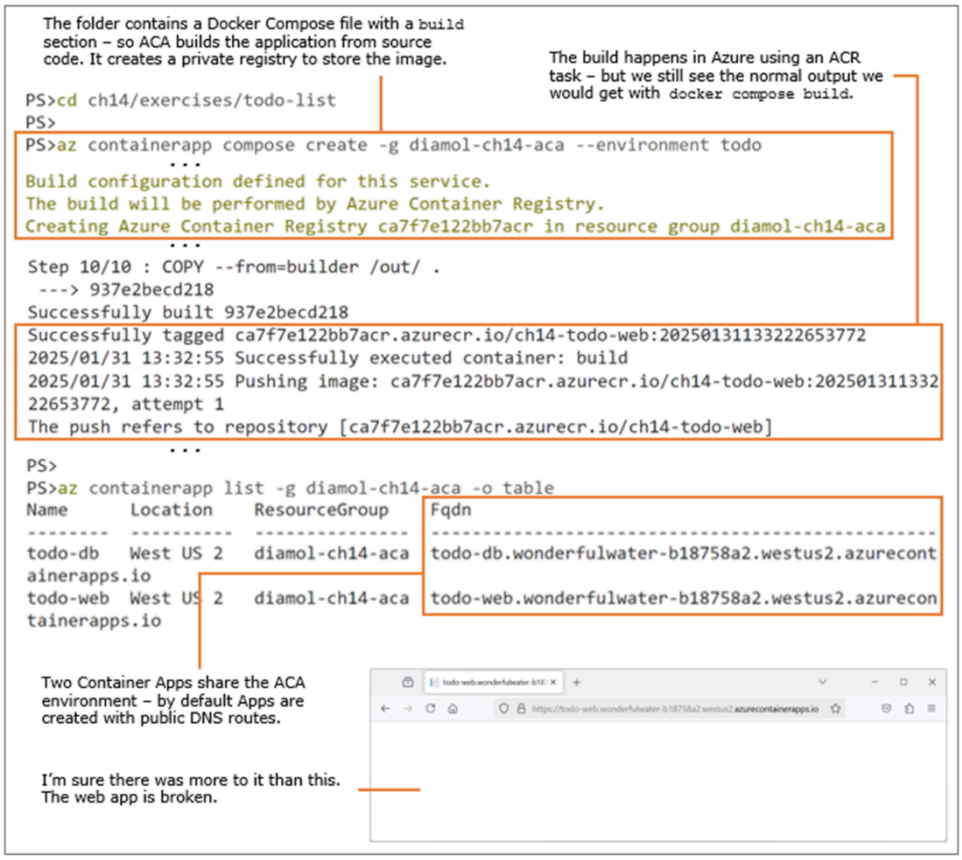


圖14.11從源代碼構建並部署到Azure容器應用

應用程序的兩個部分都在運行，但這是相同的通信問題。在這種情況下，有兩個問題要解決。 該數據庫是可以公開訪問的，但是默認情況下為HTTPS流量配置了該路由，該路由對數據庫協議不起作用，該數據庫協議期望特定端口上的TCP流量。我們需要更新這些網絡設置，還需要更新Web容器來查找數據庫的連接詳細信息。 兩者都可以使用ACA功能配置：

- *Ingress*是網絡路由組件。可以將容器應用程序配置為內部訪問，這意味著它僅適用於同一容器應用程序環境中的其他容器。內部路由可以在任何端口上使用TCP。
- 可以將應用程序配置設置為值，也可以在容器應用程序中將*secrets,*設置為*secrets,*，並以環境變量或文件安裝為單位呈現給容器。秘密非常適合機密詳細信息，例如數據庫連接的用戶名和密碼。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

我們需要進行的所有更改都可以使用Azure CLI進行，而無需重新部署應用程序。更新Web應用程序的最後一步會創建一個新的修訂版，並且當推出該應用程序時，該應用程序將正常工作。

```
＃打印數據庫的當前網絡路由：AZ ContainerApp Ingress Show -G Diamol -CH14 -AC
A -N TODO -DB -O表

＃使數據庫路由私有：
AZ ContainerApp Ingress Update -G DIAMOL-CH14-ACA -N TODO-DB  -  TYPE內部 - 運輸TCP -Expe
d-port 5432 -target-port 5432 -o表

＃使用數據庫詳細信息在Web應用程序中創建一個秘密：AZ ContainerApp Secret Set -G Diamol-Ch14-A
ca -n Todo-Web  -  -secrets的秘密-JSON = “ {\”  ConnectionStrings \“：{\”  connectionStrings \“：{\”
id = postgres; password = postgres; \“}}”"

＃並將秘密固定為卷中的捲：AZ ContainerApp Update -G Diamol-Ch14-Aca -n todo-web -secret-vo
lume-mount“/app/config”  -query'properties.configuration.configuration.fqdn'

＃該更新創建新的修訂：AZ ContainerApp修訂列表-G Diamol -Ch14 -Aca -n todo -web -o
表

＃當修訂版推出時，再次瀏覽到應用程序
```

我敢肯定，這似乎很貼心，但是所有這些步驟都可以在您的CI/CD管道中自動化。您還可以選擇進行更多的控制權，您將在生產部署中進行此操作 - 使用現有的容器註冊表或日誌分析工作空間並將其插入容器應用程序，並將入口規則和秘密定義為部署的一部分。 但是，此示例向您表明，您可以使用僅使用撰寫文件和源代碼的分佈式應用程序來啟動並運行，並且只需幾個命令在雲中運行。圖14.12顯示了我的待辦事項列表應用程序再次工作。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

圖14.12通過入學和秘密更新修復應用程序通信

Azure是在雲中運行容器化應用程序的絕佳選擇。 一組服務為您提供了全面應用程序景觀所需的所有功能，並且您可以為每個應用程序選擇正確的選項。但是，部署的核心仍然是Docker - 您有Dockerfiles可以包裝所有組件，最終容器服務只是提供了配置和運行容器的不同方法。 Docker的可移植性是其最大的優勢之一，尤其是當您在雲中運行時。 業務需求和關係發生了變化，我參與了許多諮詢活動，以將應用程序從一個雲遷移到另一個雲。 當這些應用程序已經包裝到容器中運行時，遷移變得更加簡單。我們將通過查看Google Cloud中的託管容器服務來完成本章，並查看與ACI和ACA的比較。 我們現在已經完成了Azure，因此您可以刪除ACA資源並整理所有內容。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

這刪除了所有Azure容器應用程序及其創建的其他服務。

```
AZ Group Delete-名稱DIAMOL-CH14-ACA-是的
```

14.4 Google Cloud Run

擁有多個雲的經驗很有用。 從根本上講，它們都提供了類似的服務來解決類似的問題，但是他們採用了截然不同的方法。 Google Cloud Run與ACI的易用性相似，但與ACA相似，在其為您提供的其他功能中類似。我們將運行與Google Cloud上本章開始的相同應用程序，因此您可以了解雲之間的體系結構和工具的差異。 Google Cloud使用*projects*將服務分組在一起（例如Azure資源組）。 創建一個新項目時，您需要將其鏈接到您的計費帳戶，並明確啟用要使用的服務。

現在嘗試

您需要安裝的Google Cloud帳戶設置和GCLOUD CLI。 與Docker和Azure Clis一樣，您可以將-HELP添加到任何命令中以打印幫助文本。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
# initialise your CLI (if you haven't already):
gcloud init

# store variables - you will need a unique project name,
# and any region will dol; with PowerShell:
$REGION='us-east1'
$PROJECT='<your-project-name>'

# OR Linux shell:
REGION='us-east1'
PROJECT='<your-project-name>'

# create the project:
gcloud projects create $PROJECT

# list your billing accounts:
gcloud billing accounts list

# link your new project to the billing account:
gcloud billing projects link $PROJECT --billing-account=<your-billing-account-id>

# and now enable the Artifact Registry service:
gcloud services enable artifactregistry.googleapis.com --project=$PROJECT
```

This is all one-time setup work for a new project—which could all be automated when you onboard a new application to Google Cloud. My output is in figure 14.13; nothing exciting to see there, but just make sure you follow the steps so your own project is ready to use.

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
Projects are the grouping mechanism in Google Cloud - unique names are required

PS>$REGION='us-east1'
PS>$PROJECT='diamol2ees'
PS>
PS>gcloud projects create $PROJECT
Create in progress for [https://cloudresourcemanager.googleapis.com/v1/projects/diamol2ees].
Waiting for [operations/cp.5734083539280162292] to finish...done.
Enabling service [cloudapis.googleapis.com] on project [diamol2ees]...
Operation "operations/acat.p2-403316727668-06cfcd81-56a8-43c7-b406-7168a3b7b9c2" finished successfully.
PS>
PS>gcloud billing accounts list
ACCOUNT_ID            NAME                OPEN  MASTER_ACCOUNT_ID
[redacted]            My Billing Account  True
PS>
PS>gcloud billing projects link $PROJECT --billing-account=[redacted]
billingAccountName: billingAccounts/[redacted]
billingEnabled: true
name: projects/diamol2ees/billingInfo
projectId: diamol2ees
PS>
PS>gcloud services enable artifactregistry.googleapis.com --project=$PROJECT
Operation "operations/acat.p2-403316727668-c433139d-2114-4229-b219-000e837d39c9" finished successfully.

And you need to enable the services you want
to use; this is for Google Artifact Registry

New projects need to be linked to a billing
account - a free account with credits is fine
```

圖14.13在Google Cloud中創建和配置新項目

與Azure的主要區別在於，Google Cloud Run不會從公共圖像註冊表中部署容器 - Google的docker Hub或Github容器註冊表，因此該圖像必須從Google Artifact Registry中獲取。在運行使用Cloud Run的所有容器之前，您需要創建註冊表，並使用Docker進行身份驗證，並推動要使用的圖像。和工件註冊中心有其自己的命名計劃。註冊表域是固定的，並且基於您創建服務的位置，就像在ACR中選擇自己的子域。 域名出現後，您的雲項目名稱，然後是圖像的存儲庫，然後是圖像名稱。我使用的是美國東部地區，我的項目名稱是Diamol2ees，我將為這些圖像創建一個名為CH14的存儲庫。 我可以拉動圖像ghcr.io/diamol2e/ch14- hello-cloud; 為了將其推入工件註冊表，我需要給它一個新的參考：us----------. dev/diamol2ees/ch14/hello-cloud。

現在嘗試

在您的項目中創建工件註冊表，然後從Github容器註冊表中推出Hello-Cloud圖像。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
# create the registry in your project:
gcloud artifacts repositories create ch14 --repository-format=docker --
project=$PROJECT --location=$REGION

# print the information - this includes the registry domain:
gcloud artifacts repositories describe ch14 --project=$PROJECT --location=$REGION

# store the domain in a variable - PowerShell:
$REGISTRY="${REGION}-docker.pkg.dev"

# OR Linux shell:
REGISTRY="${REGION}-docker.pkg.dev"

# authenticate the registry with the Docker CLI:
gcloud auth configure-docker --quiet $REGISTRY

# pull the sourec image:
docker pull ghcr.io/diamol2e/ch14-hello-cloud

# tag it with your Artifact Registry reference:
docker tag ghcr.io/diamol2e/ch14-hello-cloud "$REGISTRY/$PROJECT/ch14/hello-cloud"

# and push:
docker push "$REGISTRY/$PROJECT/ch14/hello-cloud"
```

Artifact註冊表沒有與ACR相同的導入功能，因此您不能直接從其他註冊表中導入圖像。 您可以使用gcloud構建從本地源代碼中構建雲中的圖像，並將其直接推向偽影註冊表。 在此練習中，我們只是拉出並推動了目標圖像，因此現在可以在雲運行中使用 - 我的輸出在圖14.14中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Artifact Registry repositories are created explicitly - the project name and repository name will be used in the image reference.

Describing the repository shows the URI - the image name will be added as the final part of the reference.

```
PS>gcloud artifacts repositories create ch14 --repository-format=docker --project=$PROJECT
--location=us-east1
Create request issued for: [ch14]
Waiting for operation [projects/diamol2ees/locations/us-east1/operations/b9199205-a2bf-46f
b-8268-b052887c2281] to complete...done.
Created repository [ch14].
PS>
PS>gcloud artifacts repositories describe ch14 --project=$PROJECT --location=us-east1
format: DOCKER
mode: STANDARD_REPOSITORY
name: projects/diamol2ees/locations/us-east1/repositories/ch14
registryUri: us-east1-docker.pkg.dev/diamol2ees/ch14
PS>
PS>$REGISTRY="${REGION}-docker.pkg.dev"
PS>
PS>gcloud auth configure-docker --quiet $REGISTRY
Adding credentials for: us-east1-docker.pkg.dev
gcloud credential helpers already registered correctly.
PS>
PS>docker tag ghcr.io/diamol2e/ch14-hello-cloud "$REGISTRY/$PROJECT/ch14/hello-cloud"
PS>
PS>docker push "$REGISTRY/$PROJECT/ch14/hello-cloud"
Using default tag: latest
The push refers to repository [us-east1-docker.pkg.dev/diamol2ees/ch14/hello-cloud]
e3bd2dd57b77: Pushed
```

Docker CLI authentication is set up at the domain level

Then it's the normal tag and push workflow

圖14.14創建文物註冊表中的存儲庫並推動本地圖像

現在我們可以運行一個容器！似乎沒有必要創建自己的註冊表並導入要使用的圖像，但這確實是一種安全措施。公共註冊機構可以並且確實將各種惡意軟件偽裝成有用的應用程序。即使有真正的圖像，也有供應鏈注入的風險 - 攻擊者可以損害合法的圖像並推動惡意版本。 許多較大的組織都禁止使用Docker Hub避免這種風險。 Google Cloud對該規則進行了形式化，並迫使您導入要使用的圖像，該圖像應成為工作流程的一部分，其中包括驗證發布者和源圖像的正確性。

我們將運行的第一個容器是Hello World應用程序，它只是打印一些信息，然後坐在等待中。 這將有助於我們了解Google雲中的下一個區別 - 雲運行容器可以是服務或作業。 *Services*是接收網絡請求的組件：網站或API。 他們以公共域名連接，並通過健康檢查創建，以確保僅將流量發送到健康的容器。我們不希望此應用程序只是背景組件。這將需要是雲運行作業，您可以通過部署作業規範然後執行作業來運行容器來創建它。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

首先啟用項目中的雲運行服務，然後使用上傳到工件註冊表的圖像創建和執行作業。

```
＃啟用服務：gcloud Services enable run.googleapis.com-project = $ project

＃使用註冊表映像部署作業：gcloud運行作業部署hello -cloud -image =“ $ registry/$ project/ch14/hello -cloud” - -project = $ jegion = $ region

＃現在執行作業 - 這啟動容器：gcloud run作業執行Hello -cloud -project = $ project -region = $ region

＃並打印日誌：
gcloud run作業日誌閱讀hello-cloud -project = $ project -region = $ region
```

您無需像ACI那樣指定雲運行的參數。 在撰寫本文時，僅支持英特爾芯片上的Linux容器，因此沒有操作系統或架構選項。如果您不提供默認值，則計算資源將使用默認值； 這些將是最小的量，例如一個CPU核心和512MB內存，但要當心默認值 - 在容器運行時分配的計算量會收取您的費用。我在圖14.15中的輸出顯示了一個作業執行中的日誌輸出，並註意該容器仍在運行，因為腳本坐著並無限期等待。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

This enables the Cloud Run service for the project

Deploying a Cloud Run job just creates the specification. You need to explicitly execute the job to run a container

```
PS>gcloud services enable run.googleapis.com --project=$PROJECT
Operation "operations/acf.p2-403316727668-7c92bacc-7738-465b-bae9-a78b68ac7f8e" finished suc
cessfully.
PS>
PS>gcloud run jobs deploy hello-cloud --image="$REGISTRY/$PROJECT/ch14/hello-cloud" --projec
t=$PROJECT --region=$REGION
Deploying container to Cloud Run job [hello-cloud] in project [diamol2ees] region [us-east1]
OK Creating job... Done.
Done.
Job [hello-cloud] has successfully been deployed.
PS>
PS>gcloud run jobs execute hello-cloud --project=$PROJECT --region=$REGION
OK Creating execution... Done.
  OK Provisioning resources...
Done.
Execution [hello-cloud-8z8f8] has successfully started running.
PS>
PS>gcloud run jobs logs read hello-cloud --project=$PROJECT --region=$REGION
2025-02-06 10:41:38 ---------------------
2025-02-06 10:41:38 Hello from Chapter 14!
2025-02-06 10:41:38 ---------------------
2025-02-06 10:41:38 My name is:
2025-02-06 10:41:38 localhost
2025-02-06 10:41:38 ---------------------
```

Executing the job allocates some compute resources and runs a container

This prints the job logs - which is just the output from the Docker container - in this case some lines printed from a script

圖14.15運行容器作為Google雲運行作業

作業非常適合背景服務，例如消息處理人員從隊列中提取工作，或者可以將其配置為按計劃進行重複流程。雲運行作業預計將在後台工作並在某個時候完成，因此，當容器停止時，它將不會重新啟動 - 除非它從啟動過程中使用錯誤代碼退出，在這種情況下，雲運行將使用新容器重試執行，最多三次嘗試。 這顯然不適合您一直都可以使用的前端組件，因此，您將使用雲運行服務。部署服務的命令類似於作業，但具有更大範圍的選項，包括端口映射和身份驗證。

現在嘗試

將PI Web應用程序部署為雲運行服務 - 首先將圖像推向您的人工製品註冊表。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
# pull the source image: docker pull ghcr.io/diamol2e/ch14-pi-web # tag it with your Artifact Registry refe
rence: docker tag ghcr.io/diamol2e/ch14-pi-web "$REGISTRY/$PROJECT/ch14/pi-web" # and push: doc
ker push "$REGISTRY/$PROJECT/ch14/pi-web" # now deploy the service: gcloud運行部署pi-web - Im
age =“ $註冊/$ project/ch14/pi-web” - port = 80 - 允許 - unauthenticatiencatient-project -project = $
project-project-region = $ gcloud unign＃打印詳細信息：gcloud run services：gcloud run服務pi web-pr
oper-jection} $ project} $ v4} $ v4} $ v4} $ v4} $ v4}瀏覽到服務網址測試應用程序
```

當您部署服務時，請密切關注輸出，您會看到Azure容器應用程序中熟悉的術語 - 重新訪問，入口和流量路由。 這些是相同的概念：對規格的更新創建不同的修訂； 多次修訂可以同時處於活動狀態，從而獲得了傳入流量的指定百分比；全部在一條DNS路線下設置了入口。默認的自動縮放設置也與ACA相同，當沒有HTTP請求時，服務縮放為零，並且最多可以擴展三個容器。您可以在圖14.16中看到我的輸出，其中容器配有一個CPU核心（儘管應用程序代碼認為它具有更多）。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

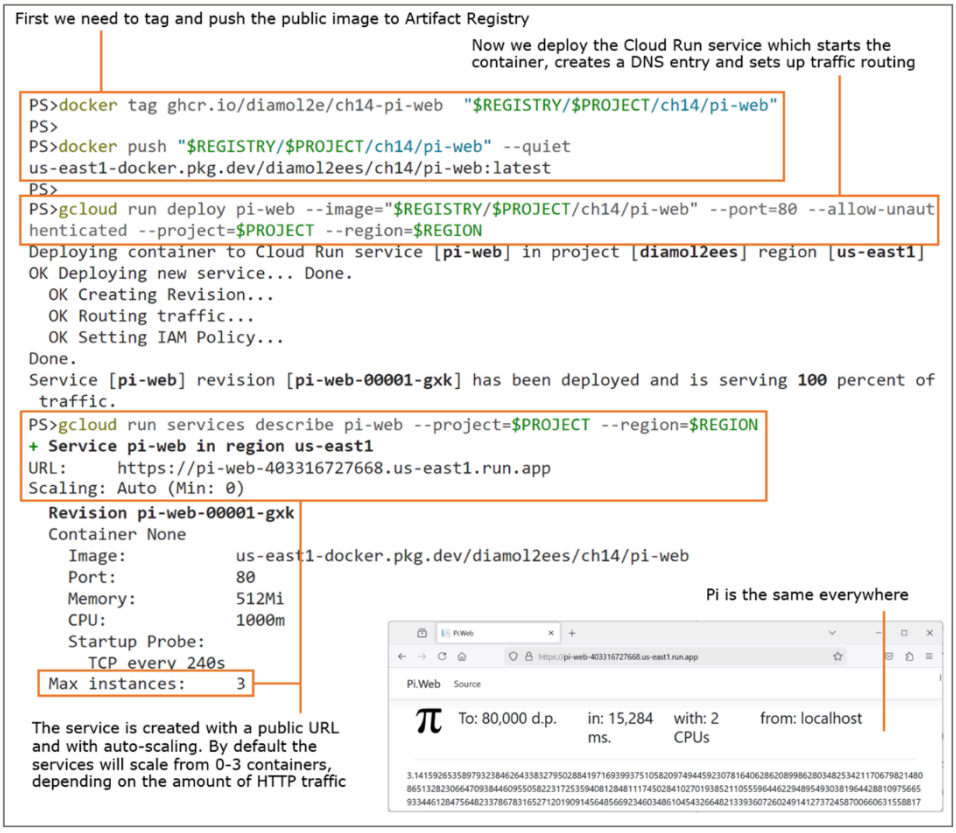


圖14.16在雲運行服務中部署Web應用程序容器

就我們而言，這是我們的Google Cloud Run，但是您應該看到，這是在生產級環境中運行應用程序的強大而快速的方法。 雲運行作業類似於ACI，雲運行服務類似於ACA，但最大的差異是使用私人註冊表，缺乏撰寫支持以及對英特爾上Linux容器的限制的要求，隨著平台的發展，所有這些都可能會改變。 這裡的關鍵要點是，當您使用Docker時，將應用程序從一個雲轉移到另一個雲所需的精力。 PI Web應用程序是一個簡單的示例，但是您知道可以在容器圖像中包裝任何大小的應用程序，而且更複雜的應用程序不需要比這些基本示例更多的努力。我們將通過查看如何選擇雲容器平台來解決本章，除非您要繼續進行實驗，否則您可以刪除Google Cloud Project。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

刪除項目，所有服務也將被刪除。

```
gcloud項目刪除$項目
```

14.5在容器平台之間選擇

您選擇的雲可能會為您決定 - 組織傾向於有一個首選的提供商，並且可能要求將所有應用都部署到同一云中。但是，該雲將具有多個容器平台，因此總會有一個選擇。您已經了解了本章中簡單，更複雜的服務，嘗試了部署經驗，並看到了一些典型的功能。 如果運行或遷移到雲到雲中，則首先搬到Docker絕對是一個明智的舉動。那麼，您應該期望花更多的時間來了解您正在考慮的每種服務的選項，並繪製哪個服務最適合每個應用程序。選擇通常將由應用程序的要求決定，但這是您應該考慮的主要因素：

- 平台支持：您需要Windows和Linux嗎？您可以伸出手臂來節省成本嗎？
- 建築商：您可以在雲中構建圖像嗎？是否與GitHub（或您存儲代碼的任何地方）和CI/CD平台有集成？
- Docker組成：CANCAN您將應用程序建模，以構成本地運行*and*雲？這將減少學習曲線以及開發人員和OPS操作之間的差距
- 平台可觀察性嗎？是否有豐富的配置模型支持敏感數據？
- 基於身份的集成與其他雲服務：您可以連接到存儲或隊列而無需存儲密碼嗎？
- 易用性：服務容易開始嗎？這是以完整功能集為代價的嗎？
- 路線圖：隨著應用程序的發展，平台可以支持您嗎？您是否會超越簡單的容器服務，如果是這樣，下一個託管選項會有很大的跳躍嗎？

雲中的容器服務實際上沒有錯誤的選擇。有些比其他應用更適合某些應用程序，但是它們都以可擴展可靠的方式運行容器的基本目的，而沒有管理基礎架構的開銷。而且，如果您確實在服務功能方面遇到了天花板，那麼您就會知道將容器化應用程序從一個服務轉移到另一個服務，甚至轉移到其他雲到其他雲是多麼容易。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

14.6實驗室

您可能在Google Cloud帳戶中剩下一些積分，該實驗室將有助於使用它們。目的是涵蓋我們沒有獲得的功能之一，並使用Google Cloud中的構建器從源構建PI Web應用程序。以下是一些讓您入門的技巧：

- 您需要創建一個新項目，以便可以隔離這項工作。 該項目將需要啟用計費和服務。
- 在新項目中，您需要一個文物註冊表。 GCLOUD構建的命令幫助將很有用。 源
- 代碼和Dockerfile在CH14/練習/pi-web中。 嘗試使用更多的CPU部署該應用程序，而
- 不是比以前的部署更快。
-
-

As always, the solution is in the lab folder, documented in the README file: https://github.com/sixeyed/diamol/blob/2e/ch14/lab/README.md.

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

15 kubernetes：primer

一開始，有容器。但是，對於Docker帶來的所有好處，生產時仍然存在一些差距。 您必須提供機器，然後安裝OS和Docker引擎。您必須在這些機器上設置對Docker的安全訪問。 而且最重要的是，沒有高可用性或規模 - 如果您想要多個服務您應用程序的容器，分佈在機器機器上，那麼（並且）無法使用Docker或組成。出現了一項新技術來填補這些空白 - *container orchestrator* - 一個為您管理容器的平台。 借助編排，您可以像作曲一樣對應用程序進行建模，然後將模型發送給編排，並承擔責任。 它將容器分配給機器，組織網絡和存儲，並使您的應用程序運行。如果容器退出或託管機器脫機，則編排器將其替換為新容器。

在本章中，您將迅速介紹世界上最喜歡的集裝箱編排的Kubernetes。 您將使用本地Kubernetes環境來學習如何使用其他章節中熟悉的示例應用程序來建模，部署和管理應用程序。 Kubernetes可以學習很多東西，但是本章將為您提供進一步探索的基礎知識和信心。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

15.1為什麼要Kubernetes？

曾經有一些編排平台的選擇：Docker Swarm，Apache DC/OS和Hashicorp Nomad都提供了相同的核心功能，其方法略有不同。這些技術仍然存在，但實際上別無選擇。 在生產中運行容器的公司可能是在數字的編排者Kubernetes上進行的。 有兩個充分的理由。 雲提供商提供託管的Kubernetes服務，為您提供一個可擴展的平台，沒有基礎架構可以管理。 Kubernetes幾乎可以運行任何應用程序，從現代微服務體系結構，傳統的N-Tier應用程序到復制的數據庫。 能夠為單個平台建模所有應用程序並能夠從一個雲到另一個雲的所有應用程序而不必重塑任何內容的功能，這使得很難證明選擇Kubernetes以外的其他內容。

Kubernetes的唯一真正缺點是它的複雜性。 與Docker組合相比，建模語言更豐富，更詳細，而邏輯概念沒有直接映射，因此學習曲線很陡。在數據中心中設置自己的Kubernetes群集也不是微不足道的，因此大多數管理Kubernetes本身的公司都使用商業平台（可能很昂貴）和/或決定加速其移動到雲（這可能是比他們預期的更大的程序）。您的投資的巨額回報是可移植性。企業的供應商可移植性 - 所有Kubernetes服務都在運行相同的核心平台，因此您可以將應用程序移至幾乎沒有或根本沒有返工的其他提供商。以及產品的技能可移植性 - 一開始開發和OPS團隊已經學習了Kubernetes，他們可以將這些技能帶入任何其他應用程序並立即富有成效。

因此，讓我們從那段旅程開始。第一個理解的想法是集群。 *cluster*是一組機器，每個機器都安裝了一個容器運行時，並將其分組為Kubernetes。運行時可以是Docker或Docker Engine的輕量級核心（稱為*containerd*），並且機器可以在Intel或Arm上運行Linux或Windows。 圖15.1顯示了樣本kubernetes群集的體系結構。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

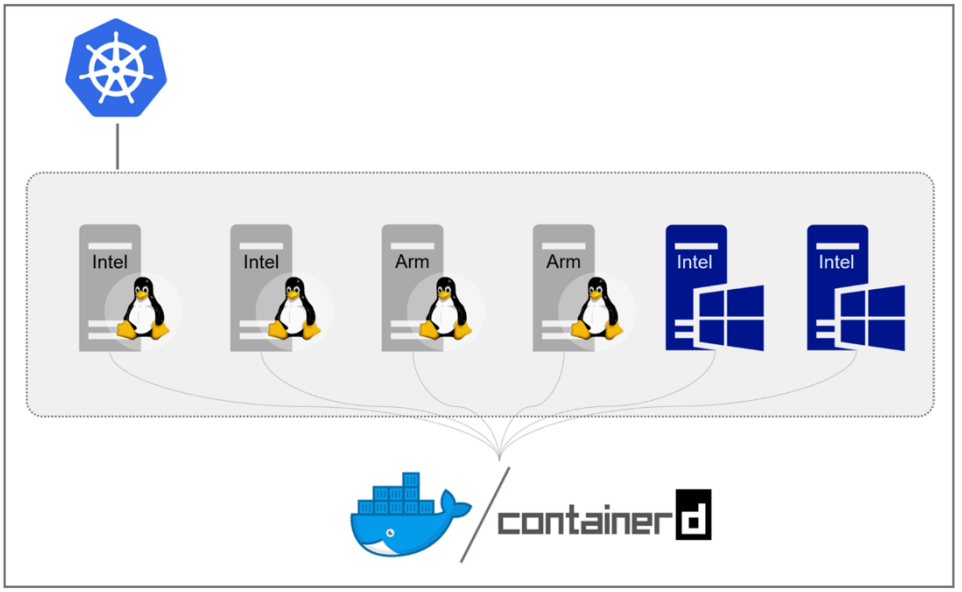


圖15.1多平台kubernetes群集可以運行任何類型的容器

配置群集很容易做到自己，但是大多數情況下它將通過平台為您完成 - 在Azure中，它可能與運行AZ AKS CREATE CREATE命令一樣簡單。 然後，您幾乎可以忘記各個機器，因為現在您將與整個集群合作。 您使用Kubernetes命令行Kubectl（發音為“ Cube-Cuttle” - 並且不要讓任何人告訴您與眾不同）與群集交談，發送查詢或命令。您可以通過將應用程序建模在YAML文件中並使用CLI將其發送到集群。然後，群集中的管理組件（稱為*control plane*）委員會容器可以在計算機上運行，但通常您現在對基礎架構層不太感興趣 - 如圖15.2所示。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

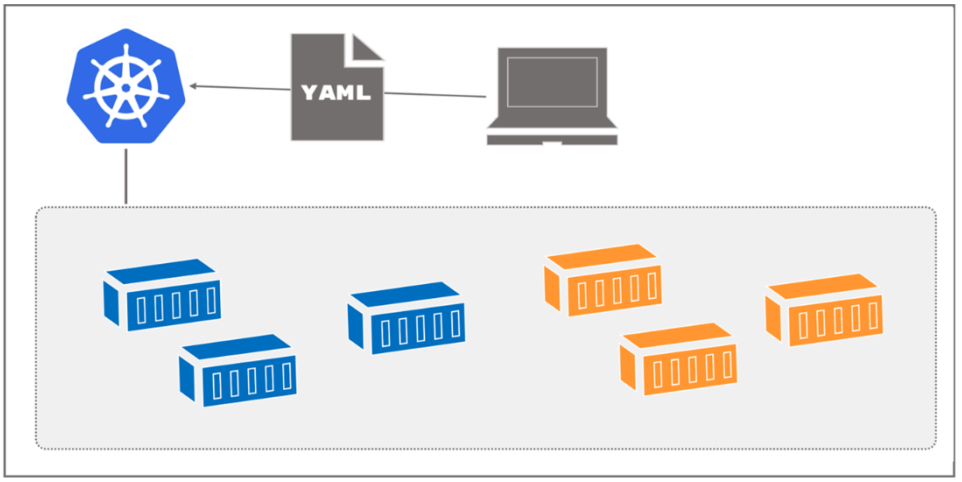


圖15.2通過將YAML模型發送到群集來部署應用程序

在我們陷入困境之前，您可以使用Kubernetes API來描述您的應用模型，這是Kubernetes的真正力量。 這是聲明性的YAML規範，您列出了應用程序的所有組件以及它們之間的連接。您使用它來建模應用程序的完整實施：

- *compute layer*將部署為容器的*communication layer*，該
- *communication layer*通過群集內的虛擬網絡實現，並在群集中和流量中進行了
- *storage layer*的路由*storage layer*，您可以描述*storage layer*，kubernetes將
- 分配抽像或提供*configuration layer*的*configuration layer*，其中應用程序設置和機密數據存儲在您的compents中，並提供了您的組合中的組合，
-

API是標準化的，這就是使您的應用程序在Kubernetes服務之間便攜式的原因。即使您需要對自定義設置進行建模以與特定的雲服務集成，也可以抽像地完成，以使雲之間的遷移更加容易。圖15.3顯示了Kubernetes API的高級層。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

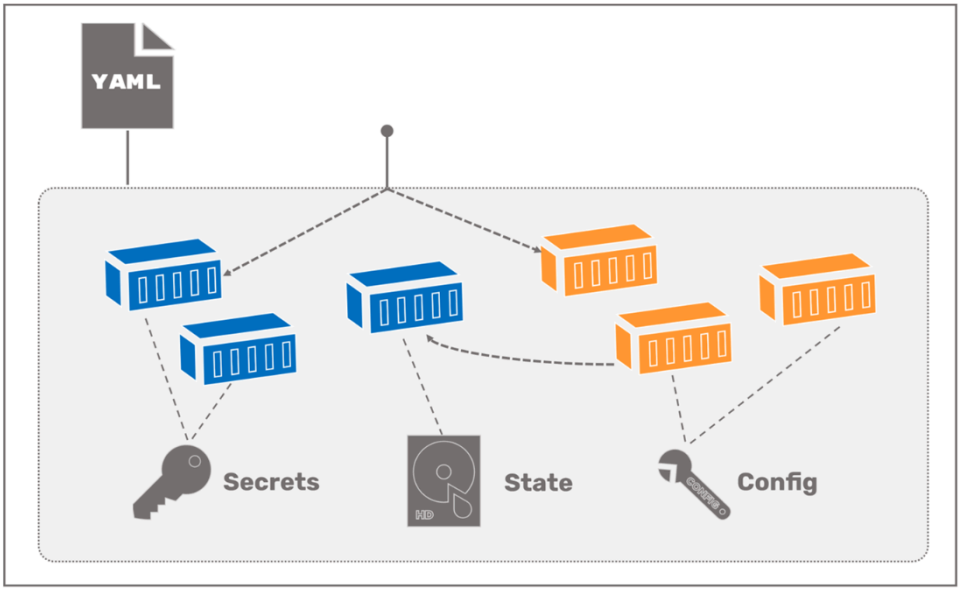


圖15.3 Kubernetes API中的建模計算，網絡，存儲和配置

渴望嘗試嗎？ 當然是。 而且您可以很容易地開始，因為有Kubernetes部署旨在作為單台計算機上的開發環境運行。 運行Kubernetes的最簡單方法是在Docker桌面中使用內置功能。打開配置設置，選擇*Kubernetes*選項，然後在*Enable*按鈕上切換 - 如圖15.4所示。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

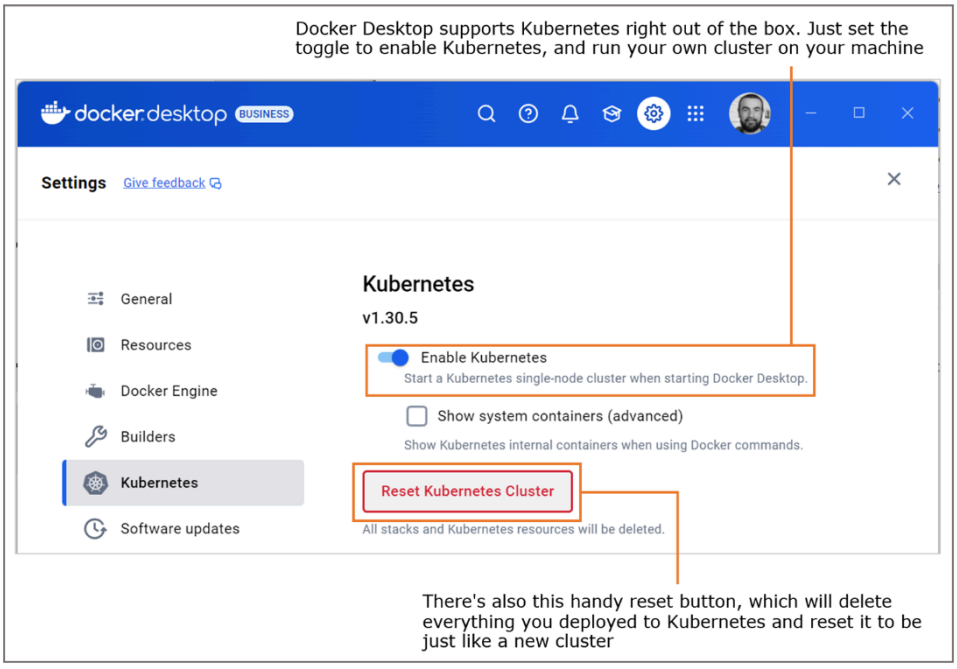


圖15.4在Docker桌面中啟用Kubernetes

Kubernetes控制平面組件都在容器中運行，Docker Desktop下載圖像並為您啟動所有內容。如果您不使用Docker桌面，那麼最佳本地選項稱為K3D（https：//k3d。IO），或者如果您可以訪問云環境，則Azure中的AKS，Amazon或Google中的EKS中的EKS都很好。 任何Kubernetes群集都將在本章中運行樣本。

現在嘗試

確保通過運行Kubernetes CLI來查詢群集來確保設置所有設置：

```
#確保您已連接到Docker DekStop群集：Kubectl配置二種docker-desktop


#群集中有關機器的打印信息 - #kubernetes稱其為“節點”：kubectl
獲取節點
```

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

如果您的所有設置都正確設置，則會看到圖15.5中的輸出。 如果您連接到雲中的較大群集，或者，如果您使用K3D，則可能會有更多的節點，那麼您的節點將具有不同的名稱，但是只要您看到*Ready*狀態中列出的控制平面節點，那麼您就可以了。

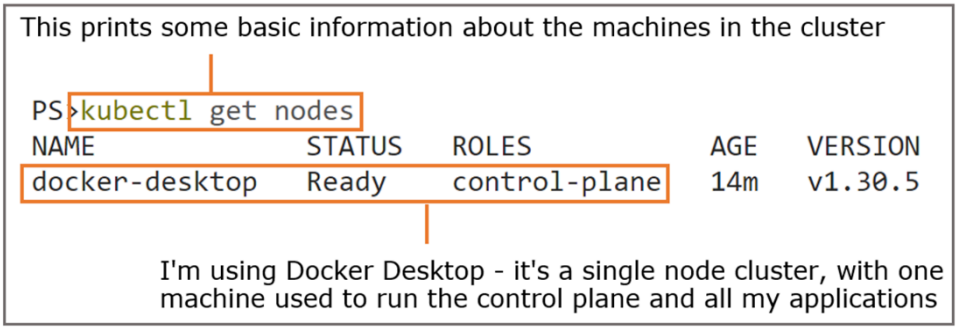


圖15.5列出群集中的節點是確保一切正常運行的好方法。

現在，我們可以開始探索API，對一些應用進行建模，並查看如何在Kubernetes中部署和管理它們。

15.2用豆莢和服務建模應用

圍繞您的頭腦的第一件事是，您不會直接在API中建模容器：容器總是包裹在稱為*Pod*的Kubernetes對像中。 POD是一個或多個容器的包裝器，其工作是管理容器。 在Docker中，當一個容器由於應用程序崩潰而退出時，這就是結束。在Kubernetes中，Pod正在觀看容器 - 如果它退出，則POD會立即創建一個替換容器。這是Kubernetes添加的額外可用性的第一層。

POD規格非常簡單：至少，您給Pod一個名稱，並列出POD需要運行的容器。對於每個容器（通常只有一個），您可以給它一個名稱並指定要運行的Docker映像。 清單15.1顯示了一個簡單的POD規格，用於打印PI值的控制台應用程序。

列表15.1基本的POD規格

apiversion：V1類型：POD元數據
：名稱：PI規格：容器： - 名稱：
PI圖像：Kiamol/CH05 -PI

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

其中還有比等效構成規格或Docker Run Kiamol/CH05-PI的更多，但其中大部分是樣板。 Kubernetes具有版本的API，分為部分。每個kubernetes yaml文件都將具有以下字段：

- *apiVersion* - 該模型使用的API版本。 Kubernetes的不同版本支持不同的API版本，這就是您告訴Kubernetes您需要哪個版本的方式。
- *kind* - 對象建模的類型。
- *metadata* - 至少以其名稱包含有關對象的信息，但可以包含其他字段。
- *spec* - 資源的實際規範。規格中可用的字段取決於API版本和類型。 POD在規格中包括容器列表；其他類型的資源具有不同的數據。

當您部署此應用程序時，Kubernetes將把POD分配給節點，並且該節點將告訴容器運行時，以拉動圖像並啟動容器。

現在嘗試

您使用kubectl應用將YAML型號發送到集群。 Kubernetes CLI採用標準方法來管理使用通用動詞的資源 - 您使用kubectl可以列出特定類型的所有資源：

```
＃部署pod yaml：kubectl apply -f ch15/練習/pi/pod.yaml
＃顯示吊艙表：kubectl get pods＃從pi pod中打印日誌
：kubectl logs pi pi pi





＃再次顯示帶有任何更改的手錶標誌的吊艙：kubectl獲取豆莢 - 觀看



＃CTRL-C或CMD-C退出手錶
```

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

運行此過程時的結果可能會感到驚訝。 POD旨在建模用於保持運行的應用程序組件（例如Web應用程序或隊列處理器），但是我們的規格是使用一個圖像，該圖像在該圖像中，容器可以在其中完成一件事然後退出。 Kubernetes使用*desired state*方法，該方法描述了群集中應運行的內容。我們的容器出乎意料地退出，因此當前狀態與所需的狀態不匹配，並且POD創建了一個替換容器（Kubernetes稱此pod retart是重新啟動）。當該容器退出時，POD會創建另一個替代品，依此類推。我的輸出在圖15.6中。

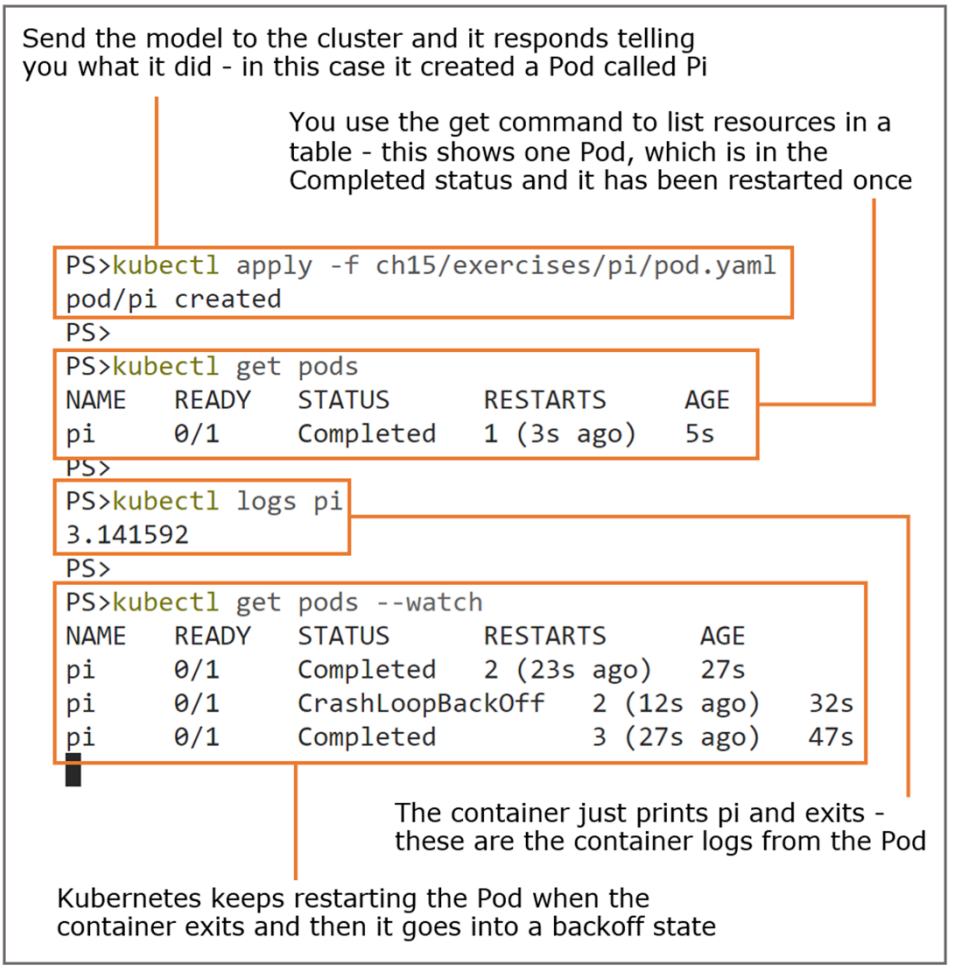


圖15.6如果容器退出，Kubernetes將繼續重新啟動POD 。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

Kubernetes足夠聰明，可以知道何時在同一POD中崩潰的容器 - 如果進行進行操作，立即替換了一個失敗的應用程序，那麼它可能會超載運行POD的節點。 如果POD在短時間內繼續重新啟動，那麼Kubernetes添加了退縮政策，在重新啟動之間等待越來越長的時間來嘗試保持群集健康。這是*CrashLoopBackOff*狀態 - 當您在真實應用中看到它時，這意味著有些錯誤。 在這種情況下，我們的模型處於錯誤狀態 - 我們不會正確配置容器以確保其保持運行，或者如果我們只想運行預期結束的任務，則使用API的錯誤部分。

讓我們切換到正確建模的示例。列表15.2中的POD規格使用隨機數生成器的變體，這是REST API組件。該容器應保持運行，因此，如果它確實崩潰並退出，那麼我們確實希望該吊艙重新啟動並保持應用程序。

Listing 15.2 A Pod spec for the random number REST API

```
apiVersion: v1
kind: Pod
metadata:
  name: numbers-api
  labels:
    app: numbers
    component: api
spec:
  containers:
    - name: api
      image: kiamol/ch03-numbers-api
      ports:
        - containerPort: 80
          name: http
```

此規範中有兩個新字段：

- *labels*是鑰匙值對，您可以將其應用於Kubernetes中的任何資源。它們用於管理對象，也用於以鬆散的耦合方式將模型的一部分鏈接在一起。
- *ports*列出了容器正在偵聽的端口。這也有助於將模型的一部分鏈接在一起，但沒有由Kubernetes強制執行 - 該容器實際上可能是在不同的端口上聆聽，或者根本沒有聆聽。

您可能會猜到，我們仍然只使用完整POD規範的一小部分。您可以包括更多可選字段，以利用更多的Kubernetes功能，但這是學習曲線的一部分。在此示例中有額外的字段，但是部署此模型將創建一個單個吊艙，就像在上一個練習中一樣。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

部署隨機數api pod。 它是同一kubectl apply命令，但是此應用程序正在偵聽網絡流量，因此我們可以對其進行HTTP調用：

```
# deploy the application: kubectl apply -f ch15/exercises/numbers-api/
pod.yaml # list Pods, showing the labels: kubectl get pods --show-label
s # print human-friendly details about the Pod: kubectl describe pod nu
mbers-api # set kubectl to listen on port 8015 # and relay traffic to the
Pod on port 80: kubectl port-forward POD/數字-API 8015：80 ＃現在瀏覽到http：// localhost：8015/rng

＃CTRL-C或CMD-C退出港口
```

Kubernetes在集群中提供網絡堆棧。每個POD都有自己的虛擬IP地址，並且POD可以使用該地址互相到達。您無法將群集以外的流量直接發送到POD中，但是Kubectl CLI具有方便的Port-Forward命令。它聽取本地網絡流量並將其發送到吊艙中 - 很棒的是，它也適用於遠程群集，因此您可以通過通過本地計算機將流量發送到雲中的Kubernetes上的Pod來調試。您可以在圖15.7中看到我的輸出。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

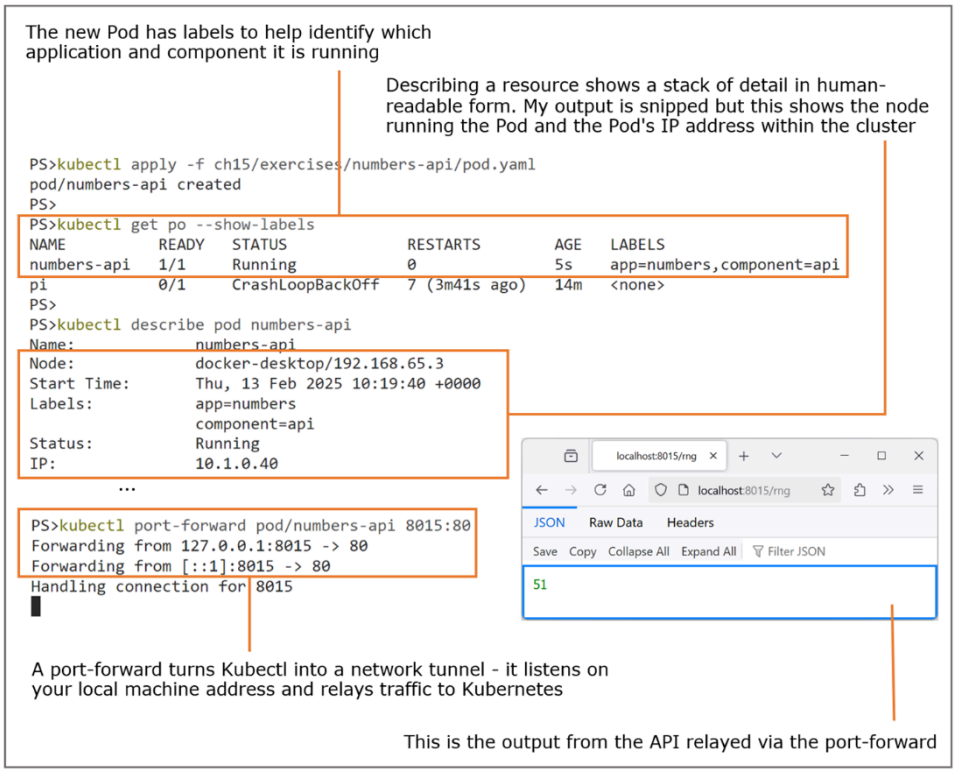


圖15.7在POD中運行HTTP服務器並通過Kubernetes CLI路由流量

端口轉發是一種有用的測試工具，但顯然您無法將其用於應公開暴露的應用程序。 為此，您需要使用另一種類型的Kubernetes對象來對網絡進行建模：*Service*。 不要將其與Docker組成的服務混淆 - 在Kubernetes中，它們是特定的網絡路由組件。可以將服務配置為偵聽進入群集（在任何節點上）的流量並將其路由到吊艙中（無論吊艙正在運行的任何節點）。 服務和POD之間的鏈接使用標籤用於鬆動耦合 - 單位15.3顯示了一項在端口8015上聽，並將流量引導到具有匹配*app*和*component*標籤的POD。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

清單15.3帶有選擇器的服務規格，以通過其標籤識別POD

```
apiversion：V1 KIND：服務
元數據：名稱：Numbers -A
PI規格：端口： - 端口：80
15 TargetPort：http selector
：app：app：app：apponent
：apponent：api類型：load
Balancer
```

用標籤和選擇器對象之間的關係建模是一個強大的概念，Kubernetes使用了很多。 它允許以相同的方式建模一對一，一對一和一對一的關係。服務尋找與選擇器相匹配的標籤的豆莢 - 如果沒有，那麼他們將無法轉發任何流量；如果有一場匹配，那麼Pod就會獲得所有流量。如果有多個匹配項，則該服務充當負載平衡器，在POD中均勻分發傳入的請求。

現在嘗試

驗證您的POD帶有與服務選擇器相匹配的標籤-Kubectl支持相同的機制。然後部署服務並查看其工作原理：

```
＃您可以使用標籤選擇器獲得資源：kubectl get po -selector應用=
數字，component = api

＃部署使用相同選擇器的服務：kubectl apply -f ch15/ercoreises/numbers
-api/service.yaml

＃列表服務：kubect
l獲取SVC

＃並打印新服務的詳細信息：kubectl描述SVC Number
s-api＃使用服務端口嘗試該應用程序：curl http：// loc
alhost：8015/rng
```

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這項練習中只有幾個命令，但是有很多事情要做。 當您完成它時，您會看到有一個與服務中的標籤選擇器匹配的POD - 它是我們在上一個練習中部署的POD。當創建服務時，它發現POD並將POD的IP地址添加到其*endpoints*中，這是它將發送流量的目的地列表。當您向群集提出HTTP請求時，它會被服務拾取並將其路由到POD，我的輸出在圖15.8中。

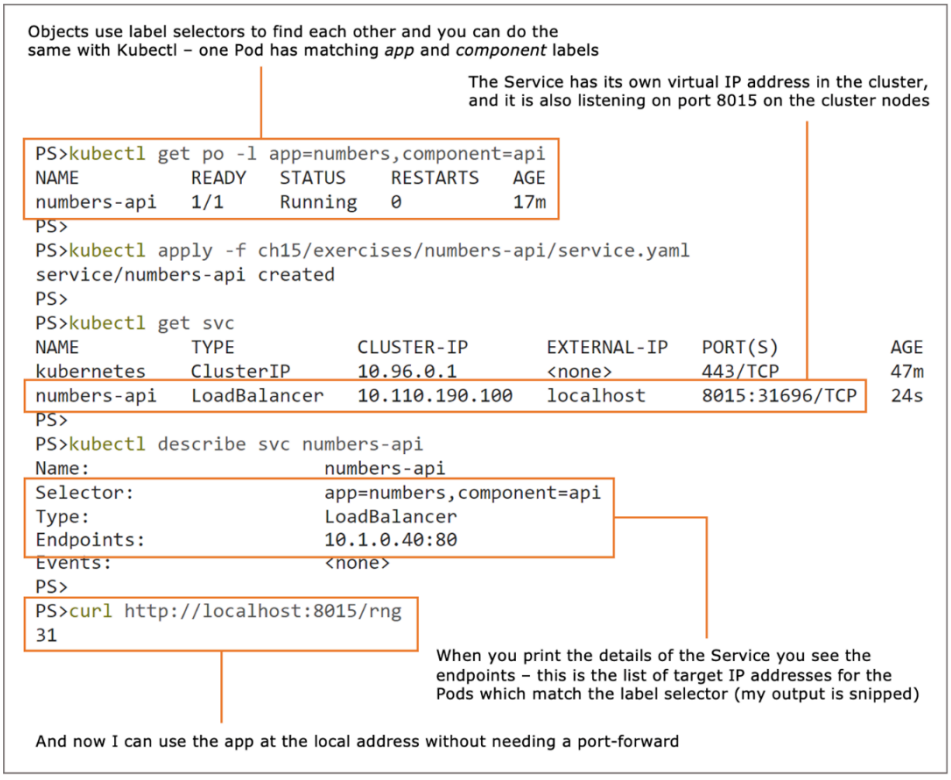


圖15.8部署服務以將傳入的集群流量路由到P 的

該服務如何知道將其綁定到Localhost地址？這是Kubernetes真正聰明的部分 - 關鍵器可能具有不同的功能，您可以部署相同的模型並獲得不同的結果。 LoadBalancer的服務類型與集群的網絡堆棧集成在一起。 Docker Desktop在您的計算機上運行一個單節點群集，因此它可以綁定到Localhost。如果您將完全相同的YAML部署到Cloud Kubernetes服務中，它將為服務分配公共IP地址，並將所有基礎架構連接到將地址綁定到集群的所有基礎架構。然後，您的集裝箱應用程序將在全球範圍內提供。

您將始終使用服務對應用程序的通信層進行建模，但是POD實際上只是Kubernetes API中的一個構建塊。您不能用一個吊艙做太多的事情，在現實世界中，您將使用建立在豆莢上並為您管理的對象。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

15.3使用部署的擴展和升級應用程序

似乎我們在這裡的深處跳來跳去，但這都是庫伯尼特的深處。您必須熟悉最少的資源來執行任何有用的事情並了解平台的某些功能。我們將在本章的其餘部分介紹這些資源，從您將用於管理POD的主要對像開始：*Deployment*。部署是POD控制器 - 部署對象的規格包括模板的POD規格，而部署對象則從模板中創建POD。 部署的工作是確保部署並運行正確數量的POD，就像Pod的工作一樣，是為了確保其容器正在運行。

清單15.4顯示了縮寫的部署yaml - 您將在部署規格內看到一個POD模板。嵌套對像是Kubernetes API經常遵循的模式，並且非常強大。 部署內部的POD資源與您可以直接在YAML中建模的POD資源相同，因此，一旦您了解了POD的所有功能，就可以在部署中使用它們。

Listing 15.4 A Deployment spec that includes the templated Pod spec

```
apiVersion: v1
kind: Deployment
metadata:
  name: numbers-api
spec:                   # this is the Deployment spec
  replicas: 1              # the number of Pods to create
  selector:             # selector to find Pods
    matchLabels:
      app: numbers
      component: api
  template:             # this is the template for creating Pods
    metadata:
      labels:           # labels the Deployment selector uses
        app: numbers
        component: api
    spec:               # and this is the Pod spec
      containers:
        - name: api     # full container spec snipped
```

部署使用標籤選擇器來查找他們管理的豆莢，就像服務使用它們來查找將流量發送到的吊艙一樣。您可以使用自己喜歡的任何標籤，但是應用程序的名稱和組件是常見的標籤。 您可以在POD模板中定義它們，並且部署中的選擇器必須包括一些或全部，以便能夠找到POD。當您創建部署時，它會創建POD（S），然後每個POD創建容器。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

刪除所有現有的POD，然後將新模型發送到群集 - 文件夾路徑包括用於部署和服務的YAML文件。列表部署和POD，您將看到一個帶有隨機（ISH）名稱的POD：

```
＃刪除所有吊艙：kubectl刪除
pod - all

＃部署新型號：kubectl應用-f ch15/練習/數字-API-V2/

＃列出所有部署：kubectl獲
取部署

＃帶有匹配應用標籤的吊艙：kubectl獲取po -l應用=數
字 - 顯示標籤

＃刪除吊艙：kubectl刪除pod -l app =數字

＃再次列出它們：kubectl獲取po -l app =數字-Show -La
bels
```

此部署會產生一個單個豆莢，到目前為止，結果與在YAML中指定POD並沒有太大不同。 豆莢的管理與眾不同。 如果運行POD的節點脫機，則部署將在另一個節點上創建一個替換POD。 這再次是採取行動的所需狀態方法 - 如果POD的數量與模型中所述的數字不匹配，則Kubernetes採取了行動。您可以在圖15.9中看到我的輸出。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The delete command removes all resources – note there is no confirmation prompt so be careful when you use it

You can deploy individual YAML files or a folder full of files with the same apply command

```
PS>kubectl delete pod --all
pod "numbers-api" deleted
pod "pi" deleted
PS>
PS>kubectl apply -f ch15/exercises/numbers-api-v2/
deployment.apps/numbers-api created
service/numbers-api unchanged
PS>
PS>kubectl get deploy
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
numbers-api   1/1     1            1           6s
PS>
PS>kubectl get po -l app=numbers --show-labels
NAME                           READY   STATUS    RESTARTS   AGE   LABELS
numbers-api-5d6bb8c468-flm6t   1/1     Running   0          12s   app=numbers,component=api,,
pod-template-hash=5d6bb8c468
PS>
PS>kubectl delete pod -l app=numbers
pod "numbers-api-5d6bb8c468-flm6t" deleted
PS>
PS>kubectl get po -l app=numbers --show-labels
NAME                           READY   STATUS    RESTARTS   AGE   LABELS
numbers-api-5d6bb8c468-sc6ch   1/1     Running   0          5s    app=numbers,component=api,
pod-template-hash=5d6bb8c468
```

The Deployment object created this Pod. The name starts with the Deployment name then adds a hash to represent the spec and a random suffix

When you delete a Pod managed by a Deployment, it immediately creates a replacement

圖15.9部署管理POD；如果缺少POD，部署將創建另一個。

檢查當前狀態針對所需狀態的過程稱為對帳循環，而Kubernetes一直都這樣做。 如果您在上一項練習中檢查Web應用程序，則在刪除POD後會看到它會繼續工作。到那時，部署發現沒有與其選擇器相匹配的豆莢並創建了一個新的豆莢。同時，該服務看到與其選擇器相匹配的POD（這與用於部署的標籤相同）有一個更改，並且更新了端點列表以將流量發送到新POD。彈性是部署的特徵之一。

比例是另一個功能。 部署具有創建POD的模板，它可以根據需要創建盡可能多的模板。 實際上，這與彈性在一起，因為如果您有一個吊艙並且倒塌，則在創建替換吊艙並將其啟動時會有一些停機時間。如果您運行了多個吊艙，並且一個吊架會下降，則在啟動替換吊艙時，其他豆莢將繼續服務流量。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

您按部署應具有的複製品數量定義比例。我們將其設置為YAML文件中的一個，但您可以用CLI手動強制更改：

```
＃將比例設置為三個 - 這將創建兩個新的豆莢：kubectl量表部署/number
s -api -replicas 3


＃所有豆莢都有相同的規格：kubectl獲取po -l
應用=數字-O寬


＃它們都將添加到服務終點：kubectl獲取端點數字-API



＃重複幾次 - 當一個POD失敗時，另一個POD仍然有效：curl http：// localhost：8
015/rng
```

該應用程序故意破壞，以說明服務如何分配流量。 Kubernetes採用圓形旋轉方法將請求均勻地傳播到Pods。當一個應用程序變得不健康時，下一個請求將起作用，因為其他POD正在運行健康的應用程序。但是，該應用程序規格不能讓服務知道POD是否不健康，因此即使所有應用程序都被損壞，流量仍然被路由 - 如圖15.10所示。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

You can manage scale *declaratively* in the YAML or *imperatively* using the command line

Two new Pods are created - they have random suffixes in the name and their own IP addresses

```
PS>kubectl scale deploy/numbers-api --replicas 3
deployment.apps/numbers-api scaled
PS>
PS>kubectl get po -l app=numbers -o wide
NAME                            READY   STATUS    RESTARTS   AGE     IP          NODE
   NOMINATED NODE   READINESS GATES
numbers-api-5d6bb8c468-8qxqt    1/1     Running   0          6s      10.1.0.45   docker-desktop
   <none>           <none>
numbers-api-5d6bb8c468-ghgr5    1/1     Running   0          3m45s   10.1.0.43   docker-desktop
   <none>           <none>
numbers-api-5d6bb8c468-jv6wf    1/1     Running   0          6s      10.1.0.44   docker-desktop
   <none>           <none>
PS>
PS>kubectl get endpoints numbers-api
NAME          ENDPOINTS                                       AGE
numbers-api   10.1.0.43:80,10.1.0.44:80,10.1.0.45:80   4m50s
PS>
PS>curl http://localhost:8015/rng
17
PS>curl http://localhost:8015/rng
{"message":"Unhealthy!"}
PS>curl http://localhost:8015/rng
20
PS>curl http://localhost:8015/rng
{"message":"Unhealthy!"}
```

The new Pods have the same labels so they get picked up by the Service

Traffic is shared among the Pods. When one of them has become unhealthy, the others still work (to begin with).

圖15.10您可以擴展部署以添加POD；新豆莢共享服務的流量分佈。

請記住，從第8章中，您可以將HealthChecks添加到Docker圖像中，以使容器運行時知道容器內的應用程序是否健康。 Kubernetes具有相同的概念，但是您需要指定模型中的檢查 - Docker Image中的Healthcheck指令被忽略。您可以配置不同類型的HealthCheck，這些檢查失敗時會導致不同的動作。清單15.5顯示了部署規範的更新，其中包括兩個定義為*container probes*的HealthChecks。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Listing 15.5 Adding healthchecks to Pod containers in a Deployment

```
apiVersion: v1
kind: Deployment
metadata:
  name: numbers-api
spec:
  replicas: 2
  # ...
    spec:
      containers:
        - name: api
          image: kiamol/ch03-numbers-api
          readinessProbe:
            httpGet:
              path: /healthz
              port: http
            periodSeconds: 5
          livenessProbe:
            httpGet:
              path: /healthz
              port: http
            periodSeconds: 20
          # the rest of the container spec is the same
```

Kubernetes支持多種類型的HealthCheck。 您可以測試以查看容器是否具有打開的TCP端口，檢查該應用程序是否正在偵聽流量；您可以在容器內運行自定義命令，該命令可能是運行大量測試的腳本。您可以提出HTTP請求並檢查響應代碼。我們已經配置了HTTP獲取請求到URL中的健康終點的兩個探針：

- 一件準備就緒的通知者每五秒鐘運行一次。如果重複失敗，則POD將離開*Ready*狀態，而服務將其IP地址從端點列表中刪除，將其設置為每20秒運行一次。重複故障會導致POD重新啟動，從而創建一個替換容器。
-

共同為應用程序配置了高可用性。當一個應用程序實例失敗時，該POD將很快從服務中刪除，並且不會獲得任何流量。 這應該意味著客戶不會看到任何錯誤響應，在某些情況下，暫停可能會給申請時間治愈。該應用程序無法自愈，因此一段時間後重新啟動，一個新的容器以在線狀態在線。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

應用更新的部署規範，看看我們的HealthChecks是否將應用程序保持在線：

```
＃更新規格：kubectl應用-f ch15/練習/numbess-api-v3/


＃觀看吊艙以查看更新的工作方式：kubectl獲取po -l應用
=數字-o wide -w

＃在http：// localhost：8015/rng瀏覽該應用程序幾次

＃CTRL-C或CMD-C退出手錶
```

CLI中的-WATCH（或-W）標誌在觀察資源的狀態更改時，都會打印一條新線路。這對於查看集群中發生的事情非常有用，但確實會產生很多文本。如果您敏銳地關注*Ready*，*Status*和*Restarts*列進行此練習時，您將看到Kubernetes如何努力工作以使您的應用程序與所需的狀態進行調和。 我在圖15.11中凝結了我的輸出，但是您仍然需要仔細看。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Deployments make changes with a staged rollout - one new Pod is created and when it enters the Ready state one of the old Pods is terminated

```
PS>kubectl apply -f ch15/exercises/numbers-api-v3/
deployment.apps/numbers-api configured
service/numbers-api unchanged
PS>
PS>kubectl get po -l app=numbers -o wide -w
NAME                              READY   STATUS              RESTARTS   AGE   IP
         NODE             NOMINATED NODE   READINESS GATES
numbers-api-959ddb77c-n6lpk    0/1     ContainerCreating   0          1s    <no
ne>      docker-desktop   <none>           <none>
numbers-api-959ddb77c-n6lpk    0/1     Running             0          2s    10.
1.0.80   docker-desktop   <none>           <none>
numbers-api-959ddb77c-n6lpk    1/1     Running             0          2s    10.
1.0.80   docker-desktop   <none>           <none>
numbers-api-5d6bb8c468-clxj8   1/1     Terminating         0          37s   10.
1.0.79   docker-desktop   <none>           <none>
numbers-api-959ddb77c-b6chk    0/1     Pending             0          0s    <no
ne>      <none>           <none>           <none>

PS>kubectl get po -l app=numbers -o wide -w
NAME                            READY   STATUS    RESTARTS   AGE   IP
NODE             NOMINATED NODE   READINESS GATES
numbers-api-959ddb77c-b6chk   1/1     Running   0          35s   10.1.0.81
docker-desktop   <none>           <none>
numbers-api-959ddb77c-b6chk   0/1     Running   0          51s   10.1.0.81
docker-desktop   <none>           <none>
numbers-api-959ddb77c-b6chk   0/1     Running   1 (2s ago)   83s   10.1.0.81
  docker-desktop   <none>           <none>
numbers-api-959ddb77c-b6chk   1/1     Running   1 (2s ago)   83s   10.1.0.81
  docker-desktop   <none>           <none>
```

I've cropped the output to focus on Pod `b6chk` That Pod responded to a request so the app in the container then failed. It fails the healthchecks so first it leaves the Ready state and then the Pod is restarted

圖15.11模型中的容器探針增加了運行應用程序的高可用性。

部署中POD規範的更改逐漸推出。 從更新的規格創建了一個新的POD，Kubernetes等待新的POD容器啟動 - 如果容器中有準備就緒探測器，它也會等到支票通過。當準備好新豆莢時，舊的豆莢之一被終止，並且該過程一直持續到所有舊豆莢都被更換為止。這可以保護您免受不良版本的侵害；如果更新有問題，而新的豆莢無法正確啟動，則舊吊艙未更換，並且您的應用程序保持運行。這全都與高可用性有關，並使您的應用程序大規模運行。您的應用程序中偶爾會發生故障，可以通過配置良好的容器探針來處理，這些午夜的支持電話將僅限於需要人類干預的更嚴重的問題。

Kubernetes還有更多，但我們將以您需要理解有效的模型的最後一個大型組件來完成本章：應用程序配置。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

15.4注射配置設置中使用ConfigMaps和秘密

應用程序配置是一個主要主題。在本書的第4部分中，我們有一個專門針對它的章節，我們將介紹如何在Docker中設計和包裝應用程序，以支持靈活的配置方法。 在這裡，我們將查看這種方法在Kubernetes中的含義。 您知道Docker創建了應用程序運行的虛擬環境，其中可以包括設置環境變量並將文件加載到容器文件系統中。這是Kubernetes中配置模型的基礎，您可以在其中定義應用程序設置作為模型的一部分，該模型被饋入容器運行時。

清單15.6顯示了*ConfigMap*資源的Kubernetes規格 - 該配置映射用於將小數據存儲為文本文件或鍵值配對。

列表15.6 configmap對象存儲單個鍵值對

```
apiversion：v1類：
configmap
元數據：
  姓名：Sleep-Config-env數據
：Diamol_edition：2E
```

這是一個獨立的對象。 您可以將yaml發送到kubernetes，它將創建一個configmap對象，並且它本身不會做任何事情。 但是，您可以在POD的容器規範中引用ConfigMap，並將其作為環境變量加載的configmap的內容到容器中。

現在嘗試

運行一個帶有POD的簡單應用程序，該應用程序可以從ConfigMap讀取環境變量。通過在POD容器內運行命令來檢查變量是否設置。

```
＃這將創建一個部署，引用configmap：kubectl應用-f ch15/練習/睡眠配
置

＃打印configmap的詳細信息：kubectl descripmapl confi
gmap sleep-config-env

＃並在容器中打印環境變量：kubectl exec deption/sleep -printenv
```

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

Kubernetes CLI的EXEC命令與Docker Exec的工作方式相同 - 它在容器內運行一個命令，這可能是將您連接到容器中的交互式外殼。 您可以將其用於特定的POD或部署，然後Kubernetes會選擇一個吊銷以持續使用的吊艙。 您會在圖15.12中看到ConfigMap的詳細信息顯示數據的完整內容，並且容器中的環境變量具有相同的數據。

This Pod starts a container which doesn't do anything - the image has a script which runs a sleep command

Printing the details of a ConfigMap shows the raw data - not suitable for anything sensitive like passwords

```
PS>kubectl apply -f ch15/exercises/sleep-configured
configmap/sleep-config-env created
deployment.apps/sleep created
PS>
PS>kubectl describe configmap sleep-config-env
Name:         sleep-config-env
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
DIAMOL_EDITION:
----
2e
PS>
PS>kubectl exec deploy/sleep -- printenv
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=sleep-d759c77dc-8cg9w
DIAMOL_EDITION=2e
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
NUMBERS_API_SERVICE_PORT=8015
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
```

Data from the ConfigMap is loaded as environment variables in the container - there can be one or many

Kubernetes also injects its own environment variables to help app communication, but this is a legacy approach which is rarely used now

圖15.12將ConfigMap數據注入到POD容器中，作為環境變量

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現代應用程序堆棧通常從多個來源合併配置設置 - 來自不同位置的環境變量和文本文件。 ConfigMap中的數據部分可以包含多個鍵值對，也可以包含XML，YAML，TOML或應用程序所需的任何格式的整個文本塊。 列表15.7顯示了一個存儲JSON文本塊的ConfigMap對象。 規格與存儲環境變量相同，但是在這種情況下，文本用作文件的內容，並且鍵用作文件名。

列表15.7將JSON配置文件存儲為configmap數據

```
apiversion：v1類：configmap metadata
：name：sleep-config-fig數據：config.js
on：| - {“ database”：{“ provider
”：“ postgres”}}
```

那個奇怪的管道鍵合組合（| - ）是yaml，用於表示多行跟隨，但它們是同一字段的一部分，因此您可以保持格式化並使規格易於閱讀。 Pods可以將ConfigMap安裝為*volume* - 與Docker卷相似的想法，但更靈活，因為您可以擁有將不同類型的數據源安裝到容器文件系統中。設置的POD將其添加到文件夾路徑中的POD將看到一個名為Config.json在該路徑上的文件，並使用數據字段的內容。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

使用一個新規格更新睡眠部署，該規範引用了兩個配置圖 - 一個加載到環境變量中，另一個將安裝到文件系統中：

```
＃更新部署規範 - 這將替換pod：kubectl應用-f ch15/ercory/sleep -configur
ed -v2


＃打印新configmap的詳細信息：kubectl描述cm slee
p-config-file


＃列出在容器中安裝configmap的路徑：kubectl exec deploy/sleep -ls -l/app/config



＃打印固定文件的內容：kubectl exec deploy/sleep -cat/app/config/config.
json
```

您知道，在Docker中，您無法更改運行容器中的環境變量或音量安裝，只有在創建容器時才能設置它們。在kubernetes中也是如此，因此，當您在改變容器設置的部署對像中更改POD SPEC模板時，通過推出新POD來實現。 此練習表明，新容器將配置的內容視為文件，它看起來像是文件系統的普通部分。我的輸出在圖15.13中；如果您還將環境變量打印在新POD中，則會發現上一個練習中的配置也已加載。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The Pod spec in the Deployment has changed. The UI doesn't show it but this will be rolled out by replacing the Pod.

This ConfigMap contains a text file - Kubernetes doesn't care about the format, it can be JSON or anything else

```
PS>kubectl apply -f ch15/exercises/sleep-configured-v2
configmap/sleep-config-env unchanged
configmap/sleep-config-file created
deployment.apps/sleep configured
PS>
PS>kubectl describe cm sleep-config-file
Name:         sleep-config-file
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
config.json:
----
{
  "Database" : {
    "Provider" : "Postgres"
  }
}
PS>
PS>kubectl exec deploy/sleep -- ls -l /app/config
total 0
lrwxrwxrwx    1 root     root   18 Feb 13 11:00 config.json -> ..data/config.json
PS>
PS>kubectl exec deploy/sleep -- cat /app/config/config.json
{
  "Database" : {
    "Provider" : "Postgres"
  }
}
```

The ConfigMap is mounted as a volume to the path `/app/data` - the `config.json` file is actually a symbolic link managed by Kubernetes

The contents of the file are materialized from the ConfigMap data value

圖15.13將ConfigMap安裝到文件夾路徑將數據內容顯示為文件。

ConfigMaps是保持應用程序模型靈活的有用方法。您可以將規範的核心部分保持相同，但是在ConfigMap級別的不同環境中有所不同。您已經看到，任何可以訪問群集的人都可以用純文本打印配置map的內容，因此它們不適合存儲諸如憑據之類的敏感數據。為此，Kubernetes具有*Secrets*的工作方式，其工作方式與ConfigMaps相同，但可以鏈接到外部特勤服務（例如數據中心中的Azure或Hashicorp Vault中的KeyVault），因此數據源可確保集群外。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

就是Kubernetes API的主要部分。還有更多內容可以挖掘，但我們涵蓋了模型的計算，網絡和配置部分的關鍵部分。最後一個練習將使用Postgres數據庫運行我們的舊待辦事項列表應用程序。該模型包括部署，服務，配置示意圖和秘密，這是您開始使用Kubernetes提高生產力所需的一切。圖15.14顯示了體系結構，如果您想查看它們如何融合在一起，我將讓您探索YAML。

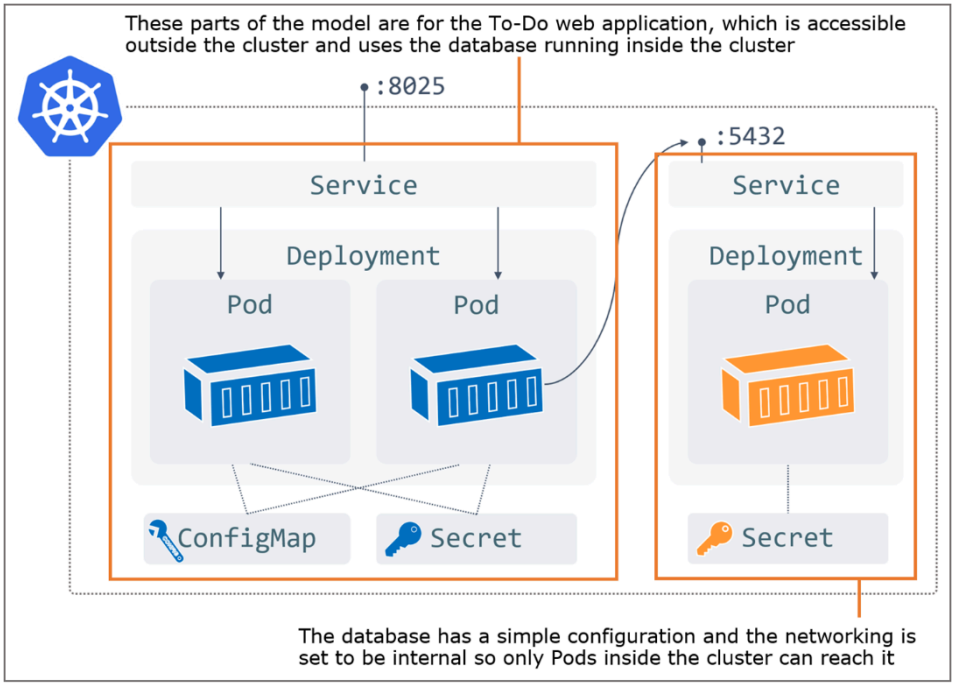


圖15.14使用所有核心資源，具有多個組件的Kubernetes中的完整應用模型

無論您的模型多麼複雜，您都以Kubernetes CLI的方式部署它。集群為您創建所有資源，並為您匯合。 該規格為Web應用程序創建了兩個複製品，一個為數據庫創建了兩個複製品，因此您將看到三個吊艙。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

部署待辦事項列表應用程序並驗證所有這些都按預期工作。 如果您有時間，可以探索很多資源：:)

```
＃部署應用程序：kubectl apply -f ch15/練習/todo -lis
t


＃打印豆莢和服務創建：kubectl get po，svc -l app = t
odo -o -o wide


＃當豆莢準備就緒時，請從數據庫中打印最後的日誌行：kubectl日誌部署/todo-db  -  -tai
l 1


＃瀏覽到http：// localhost：8025-添加一些數據並刷新
```

運行此操作時，您會發現在Kubernetes中建模分佈式應用程序時，這是一個很好的開始。更好的細節不存在 - 用於容器的醫療檢查並在其自身的捲中隔離數據庫數據 - 但首次剪切很好。這就是我將如何處理建模應用程序的方式：讓每個組件首先工作，然後將它們連接並使整個應用程序工作，然後迭代其以提高性能，可用性和可維護性。我在圖15.15中的輸出以所有榮耀顯示了該應用程序。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

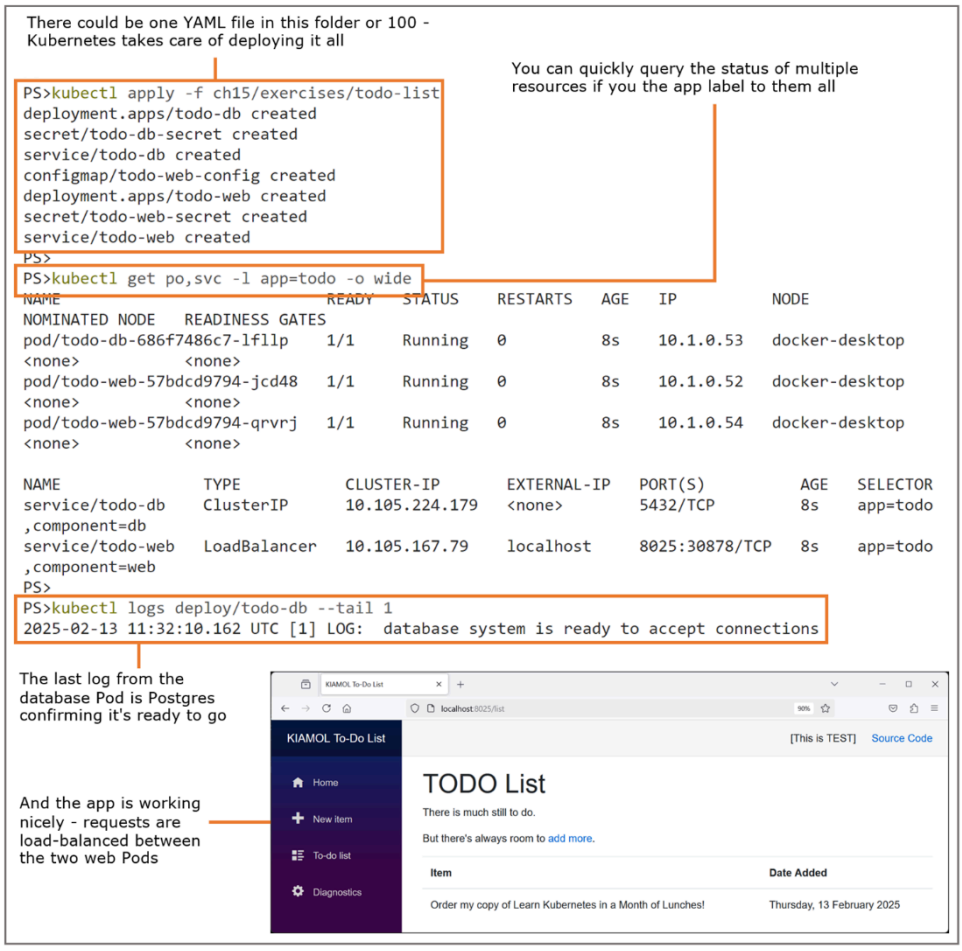


圖15.15在Kubernetes中部署和管理分佈式應用程序；無論應用多麼複雜，工具和過程都是相同的。

這就是我們要涵蓋的全部。當我第一次開始使用Kubernetes時，我從撰寫格式中獲得了新鮮的，我想：“哇。這太複雜了。肯定不會起飛嗎？”因此，不要來找我進行熱門技術預測。 Kubernetes的重點是，所有復雜性都可以確保您可以建模任何類型的應用程序。在我的諮詢工作中，我幫助公司將各種應用程序轉移到雲中的Kubernetes中 - 從十年曆史的.NET .NET框架Windows應用程序到全新的微服務應用程序到大數據系統。他們都使用相同的kubernetes API和工具，並且它們都靈活地設計了，因此相同的模型可以使用最小的測試環境和完全縮放的生產環境，並且具有一些不同的配置設置。 這確實是有價值的，因為部署是可移植的和防止未來的，而且要做的技能正在努力學習。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

15.5計劃Kubernetes學習路徑

有效地使用Kubernetes的第一步是掌握Docker，因此您已經走了。本書的其余章節將使您考慮使用模式和概念也適用於Kubernetes的生產準備。完成*Learn Docker in a Month of Lunches*並想繼續使用容器的旅程時，這就是我建議的：

- *Read Learn Kubernetes in a Month of Lunches*。哎喲。宣傳自己的書有點多，但是我認為*Month of Lunches*格式非常適合Kubernetes。您需要花費大量時間與Kubernetes一起工作，以使其感到滿意，並且您需要體驗很多API，才能真正了解您可以做什麼。 *Learn Kubernetes in a Month of Lunches*通過您在本書中採用的相同目標現實世界的方法來指導您。不過，一個小警告 - 在Kubernetes中有很多可怕的信息，我不會跳過重要的事情來使章節保持短距離，因此您可能需要更長的午餐休息時間。

- 當您對Kubernetes API感到滿意並且可以建模並運行自己的應用程序時，您很快就會意識到該方法有一些限制。 YAML模型是靜態的 - 如果將復製品計數設置為部署中的1個，則不能在不更改YAML的情況下將其更改為10。您不需要與略有差異的同一模型的多個副本，因此Kubernetes生態系統具有幾個模板選項。這些語言是採用Core Kubernetes API的語言，並為您提供可擴展的模型，以便您可以輕鬆配置同一應用程序的不同變化。 *Learn Kubernetes in a Month of Lunches*向您介紹*Helm* - 也有*Kustomize*。您需要對兩者都有一種感覺，然後選擇一種（可能是舵手）以詳細學習。

- 如果您想證明自己對Kubernetes的了解程度，則有三個認證選項來自Cloud Native Computing Foundation（管理Kubernetes Project）：經過認證的Kubernetes應用程序開發人員（CKAD），認證的Kubernetes管理員（CKA）和認證的Kubernetes Security Security Specialist（CKS）。 CKAD可能是您想要的那個，它專注於建模，運行和管理應用程序。我在網上有大量的培訓材料，可以在https：// eltons為您提供幫助。學院。 恐怕這還不夠。 Kubernetes與它正在運行的平台深入集成，因此您需要了解這些集成點和提供商的功能集。我提到，當您部署到雲中的Kubernetes時，LoadBalancer服務將分配公共IP地址並將路由連接。雲平台還具有自己的存儲選項，秘密管理，訪問控制和監視。所有這些都建立在核心Kubernetes API之上，但是您需要了解AK，EKS或GKE中的哪些功能，以便您可以在應用程序中充分利用它們。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這聽起來像是一個大問題，但是容器確實是一種改變職業的技術。我從2014年開始與Docker和2016年的Kubernetes合作，並且（與雲層一起）仍然是我今天在幫助公司的主要工作。 Kubernetes已成為一種主要的技術，即使它確實被取代了，無論它取代了支持Kubernetes API所需的一切，因此您的學習投資應該在未來幾年內安全。

15.6實驗室

您將花費大量的Kubernetes建模時間來進行應用程序配置。這是一個OPS-DEV交點，OPS團隊需要知道應用程序如何加載配置設置以及可以改變什麼，而開發團隊需要知道在不同環境中運行的選項。這個實驗室為您提供了一些經驗。

- 我們從您在第15.4節中部署的待辦事項應用程序開始 - 如果您沒有它，請從文件夾CH15/ercory/todo-list再次部署它。 使用該應用程序並檢查Web組件的日誌 - 您不會看到任何內容，因為默認配置設置僅在存在錯誤時打印日誌。 CH15/實驗室文件夾中有一個名為Logging.json的文件，該文件增加了記錄級別。您的目標是將配置文件注入Web容器文件系統。 該應用程序在與config.json文件的同一文件夾中尋找該日誌記錄配置文件。 進行更改時，現有豆莢是否拾取並開始編寫日誌，還是需要強迫部署來創建新的POD來拾取新的配置文件？
-
-
-

我的解決方案是在實驗室文件夾中，如果需要的話，並且在RealMe文件中有一些文檔。 並記住Docker桌面中的重置Kubernetes按鈕：https：// github。 com/sixeyed/diamol/blob/2e/ch15/lab/readme.md。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

16 CI/CD在雲中
d Ocker和GitHub Action

連續集成/連續部署（CI/CD）管道是現代軟件項目的核心。開發人員將更改推向源代碼，並觸發管道運行，該管道運行構建，測試，軟件包和部署整個應用程序。 對於某些組織而言，自動部署到生產可能會有點多，但是您可以不斷地部署到預生產環境中，並確信您始終擁有最新功能。 該管道對項目的成功至關重要，因此它需要快速運行並可以重複且便攜。這為開發團隊提供了快速的反饋，並讓您輕鬆地通過管道調試問題，這意味著您可以在CI/CD服務之間以最少的返工數量移動。 Docker非常適合為管道供電 - 您可以在多級版本中編譯代碼並運行單元測試；使用Docker構成對其進行測試的應用程序的本地部署； 並將測試的圖像發佈到發布存儲庫進行生產部署。

在本章中，我們將使用Docker在雲中瀏覽完整的CI/CD管道，以完成所有艱苦的工作。 主要的CI/CD提供商支持Docker（Azure DevOps，Circle CI，Gitlab更多），因此您可以使用熟悉的命令將構建步驟放在一起。我們將使用GitHub Action，這是一種成熟且功能良好的產品，並支持Windows和Linux構建代理，因此我們還可以通過多平台構建來工作。 GitHub動作可以在公共存儲庫中自由運行。如果您還沒有一個，則需要註冊一個GitHub帳戶，但是練習之後不會花費任何費用。

16.1雲中帶有github動作的DevOps

GitHub Action是一項相對較新的服務，非常受歡迎，因為它快速，功能豐富且易於使用。 它遵循與所有云CI/CD服務相同的基本模式：

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

- 您可以在將代碼更改時定義源代碼存儲庫中的文件中的構建步驟，從而觸發新的構建
- github操作規定，某些計算（a *runner*}）執行構建式跑步者拉動了源代碼repo並在構
- 建末端執行構建步驟，跑步者已清理和排行。
-
-
-

您使用yaml來定義構建過程，每個YAML文件都是github *workflow*。您可以使用大量的步驟來構建工作流程 - 檢查存儲庫，構建代碼，並使用AI工具查看代碼。 這些是對您的工作流進行建模的個人*actions*，儘管其中數千個可用，但我更喜歡使用盡可能少的，並將大部分邏輯放入腳本文件中。這是我們在第11章中與詹金斯（Jenkins）遵循的方法，理由是相同的。 GitHub的Workflow YAML是一種自定義格式，這些操作僅在GitHub工作流中起作用。 如果您想遷移您的構建提供商（最終您將最終），那麼很難取消所有自定義步驟並在新服務中重新創建它們。相反，使用腳本可以使遷移更加簡單，這也意味著您可以在自己的計算機上運行步驟來調試問題或添加功能。

現在嘗試

您將需要安裝更多工具來遵循本章中的所有練習。您可能已經有一個（免費的）GitHub帳戶和GIT客戶端。構建邏輯全都在腳本中，因此您還需要PowerShell 7 +（https：//aka。MS/PWSH）。可選地，如果您不想繼續刷新Web UI，請安裝GitHub命令行（https：//cli。github.com）。

```
＃檢查您的依賴項可用：
git版本
pwsh- version

＃啟動PowerShell：
PWSH

＃並確保您的Docker設置使用Linux容器：$ info = docker版本-f json | conver
tfrom -json


echo“平台：$（$ info.server.os.tolower（））/$（$ info.server.ark.tolower（））”
```

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

如果您不喜歡安裝PowerShell的想法，則應該相信我，並將本章作為學習愛它的機會。圍繞多平台構建的邏輯總是很貼心，PowerShell為您提供了一組可讀且易於維護的腳本集（我不想看看Bash中的某些步驟會是什麼樣子）。 這也是一個跨平台工具，因此團隊成員可以在其擁有的任何筆記本電腦操作系統上運行這些腳本，而GitHub跑步者則配有PowerShell預裝，因此您可以在Linux和Windows上的雲中獲得相同的體驗。 圖16.1顯示了我的輸出 - 只要您有類似的東西，就可以開始使用。

Git and PowerShell 7+ are pre-requisites for this chapter. Your versions don't need to match mine, as long as you don't get errors from these commands.

```
PS>git version
git version 2.31.1.windows.1
PS>
PS>pwsh -version
PowerShell 7.4.6
PS>
PS>$info = docker version -f json | ConvertFrom-Json
PS>
PS>Write-Output "Platform: $($info.Server.Os.ToLower())/$($info.Server.Arch.ToLower())"
Platform: linux/amd64
```

This script prints out your Docker platform - it confirms you're using PowerShell and are in Linux container mode. Intel or Arm CPUs are fine.

圖16.1驗證您擁有本章所需的所有工具

本章中的練習是他們自己的專用GitHub存儲庫。您將自己製作該倉庫的副本 - 在Github中添加了一個*fork*，因此您可以在您自己的版本的存儲庫中推動更改並觸發工作流程。在瀏覽器中打開github並登錄，然後轉到https：// github。 com/diamol2e/ch16/fork。您將在圖16.2中看到屏幕，您可以在其中離開默認值並單擊創建叉。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

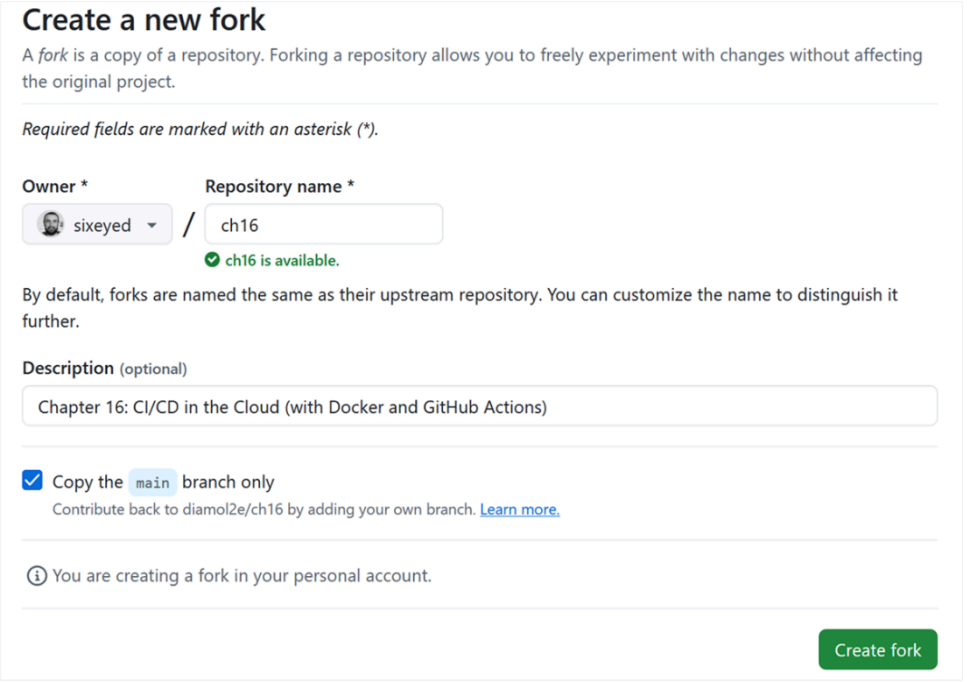


圖16.2本章的回購在您自己的github帳戶中創建副本。

要花一點時間來分配存儲庫，然後您將擁有自己的副本 - 請記下url的註釋（我的github用戶名是*sixeyed*，所以我的叉子在https：//github。com/sixeyed/）。存儲庫包含我們在第4章中介紹的日期應用程序的天文學圖片的源代碼的版本，以及Dockerfiles，撰寫文件，CI/CD腳本和GitHub Workfrows。這些Dockerfiles已被擴展到包括在構建單獨階段運行的單位測試。清單16.1顯示了Go組件的dockerfile剪輯。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

清單16.1在多階段Dockerfile中運行單元測試

```
來自Diamol/Golang：2e作為構建

複製go.mod main.go /src /workdi
r /src run go mod tidy run go build
-o /server



＃從構建中的單元測試作為測試副
本sample_test.go / src / run go test &
& \ echo'> tests.passsed＃延續了最
後階段
```

這裡有兩件事要注意。 首先，單位測試在自己的階段進行，取決於構建階段。如果要編譯應用程序并快速運行測試，則您對包裝最終圖像不感興趣，然後您只能針對測試階段。其次，如果測試通過，則測試階段會生成文本文件。該文本文件被複製到最終圖像中以創建依賴關係，並確保在構建完整圖像時不會跳過測試階段。 該應用程序有三個組件，它們都具有相同的Dockerfile結構。清單16.2顯示了構建腳本的一部分，該腳本運行了每個腳本的編譯和單元測試階段。

清單16.2構建針對測試階段的Docker圖像

```
＃$組件是一個變量，可存儲foreach的組件名稱（$ components in $ compone
nts）{pushd $ component docker build -t“ $ component -test” - -target Test。
如果（$ lastExitCode -ne 0）{exit 1} popd}
```

這裡的重點是快速反饋。這將作為CI/CD管道的第一部分運行，如果代碼或失敗的測試發生了破壞的變化，那麼我們想盡快知道這一點。腳本一旦發生故障就會退出。這也是一個不錯的快速腳本，開發人員可以在推動更改之前運行，並確信如果它在其機器上成功，它也將在Github中起作用。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

我們可以做同樣的事情 - 這是我們在構建完整管道時將要遵循的模式：首先為本地的每個部分運行腳本，然後在Github中觸發該部分的工作流程。這樣，您可以看到機器上的構建步驟，然後在GitHub中查看其外觀。

現在嘗試

克隆存儲庫的叉子，並運行第一個構建腳本以編譯源代碼並使用Docker運行單元測試。

```
＃將您的github ID存儲在變量中：$ ghid ='< yo
ur-github-username >'

＃並克隆自己的倉庫副本：cd < location-for-the-code
> git clone“ https://github.com/ $


＃切換到構建腳本文件夾：CD CH16/CICD


＃運行構建腳本：./build.ps1
```

首次運行該構建腳本時，可能需要一兩分鐘才能下載所有基本圖像，但是如果重複它，則應在幾秒鐘內完成。這是快速的反饋，開發人員沒有任何藉口跳過這一步驟並冒著破壞構建的風險。我在圖16.3中的輸出已刪節，但是您知道到目前為止，Docker構建的樣子。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
You're cloning your own fork of the code -
be sure to use your own GitHub username.

                                    The pipeline scripts are all in the cicd folder. This script
                                    builds each Docker image - targeting the unit test stage.

PS>$ghId='sixeyed'
PS>
PS>git clone "https://github.com/$ghId/ch16" --quiet
PS>
PS>cd ch16/cicd
PS>
PS>./build.ps1
*** Building: access-log with unit tests
[+] Building 4.9s (5/12)                                                          docker:wsl
 => [internal] load build definition from Dockerfile                                    0.2s
 => => transferring dockerfile: 562B                                                    0.1s
 => [internal] load metadata for docker.io/diamol/node:2e                               1.7s
 => [auth] diamol/node:pull token for registry-1.docker.io                              0.0s
 => [internal] load .dockerignore                                                       0.1s
 => => transferring context: 2B                                                         0.0s
 => [build 1/4] FROM docker.io/diamol/node:2e@sha256:e78e597b35b220832c8c9933ddb5773    2.8s

  I've cut the full output but you should find all the unit tests pass and all three images build correctly.
```

圖16.3　　scr IPTing Docker映像構建，其中包括編譯代碼和運行U　　NIT測試

這一切都很好，但是僅僅測試快樂的道路還不夠好。 如果代碼更改會導致測試失敗怎麼辦？ 我們需要確保故障通過Docker構建起來並導致構建腳本失敗，因此我們保持快速的反饋並知道存在問題。

現在嘗試

Node.js應用程序源代碼中有一個不良測試。目前它被忽略了，但是當您重命名文件時，它將包含在內並導致構建腳本失敗。

```
#重命名不良的測試文件，以便將其包含在測試運行中：MV ../ src/access-log/sr
c/failing.test.js.ignore ../src/access-log/src/failing.test.test.js



#現在再次運行構建：./build.ps1
```

此運行應該很快。失敗的組件是第一個構建的組件，並且腳本在失敗後立即退出。 這正是我們想要的 - 開發人員經常在綜合開發環境（IDE）中工作，這與構建環境完全不同。 單元測試可能全部通過IDE，但在容器中失敗，該容器從不同位置加載庫和配置。 您可以在圖16.4中看到我的輸出，這告訴我不要推動更改，因為我會破壞構建。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
This file has a unit test which deliberately fails. Changing the
file extension to .js means it gets included in the test run.

PS>mv ../src/access-log/src/failing.test.js.ignore ../src/access-log/src/failing.test.js
PS>
PS>./build.ps1
*** Building: access-log with unit tests
[+] Building 3.7s (11/11) FINISHED                                          docker:wsl
 => [internal] load build definition from Dockerfile                              0.2s
 => => transferring dockerfile: 562B                                              0.1s
 => [build 1/4] FROM docker.io/diamol/node:2e@sha256:e78e597b35b220832c8c9933ddb57730f9bcd3  0.0s
 => [internal] load build context                                                 0.2s
 => => transferring context: 361B                                                 0.1s
 => CACHED [build 2/4] WORKDIR /src                                               0.0s
 => CACHED [build 3/4] COPY src/package.json .                                    0.0s
 => CACHED [build 4/4] RUN npm install                                            0.0s
 => CACHED [jest 1/1] RUN npm install jest --save-dev                             0.0s
 => CACHED [test 1/2] COPY src/ .                                                 0.0s
 => ERROR [test 2/2] RUN npm test &&     echo '' > tests.passed                   2.8s
                ...
2.680 Tests:       1 failed, 1 passed, 2 total

The test fails and that causes the Docker build to fail. I've snipped the output,
but yours will show the full details of the test suite and the failures.
```

數字　16.4證明構建腳本如果在酸味中進行單位測試失敗，則會失敗　　此代碼

清單16.3顯示了我們將在Github運行的第一個工作流程。如果您完全是語法的新手，那麼看到正在發生的事情仍然相當容易。我們將在最新的Ubuntu Linux Runner上運行作業，該作業有兩個步驟：查看代碼並在CICD目錄中運行build.ps1 powershell腳本。

列表16.3一個簡單的github工作流以運行PowerShell腳本

```
工作：
 構建：runs-on：ubuntu-latest步驟： - 用法：a
 ctions/neckout@master-名稱：build working-dir
 ectory：cicd run：./build.ps1 shell：pwsh
```

這個故意簡單的工作流程採用了我喜歡的方法 - 有趣的邏輯都在PowerShell中，正如我們所看到的，您可以將其作為唯一的依賴性在本地運行。如果我們需要將此構建移至Azure DevOps，則只有YAML才需要更改。但是，讓我們在github中運行它 - 我們也會推動我們的破壞單元測試，並確認構建失敗。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

GitHub在文件夾.github/Workflows中查找工作流yaml文件 - 因此，要觸發構建，您只需要將工作流文件複製到正確的位置並推動回購即可。

```
＃移動構建工作流文件：mv ../.github/workflows.ignore/build.yaml ../.github/workf
lows/

＃提交並推動更改：git add -all; git commit -m'添加構建工作流'; git推

＃瀏覽此URL以查看您在雲中的構建：echo“ https：//github.com/
qunghid/ch16/actions”
```

我希望您對它的簡單程度以及無縫的github行動對正常開發實踐的無縫性感到驚訝。 只有幾頁，我們有一個用不同語言編寫的分佈式應用程序，準備在雲中構建。 是的，正如您在圖16.5中所看到的那樣，構建失敗了，但是我們故意這樣做，很快我們將其正常工作。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

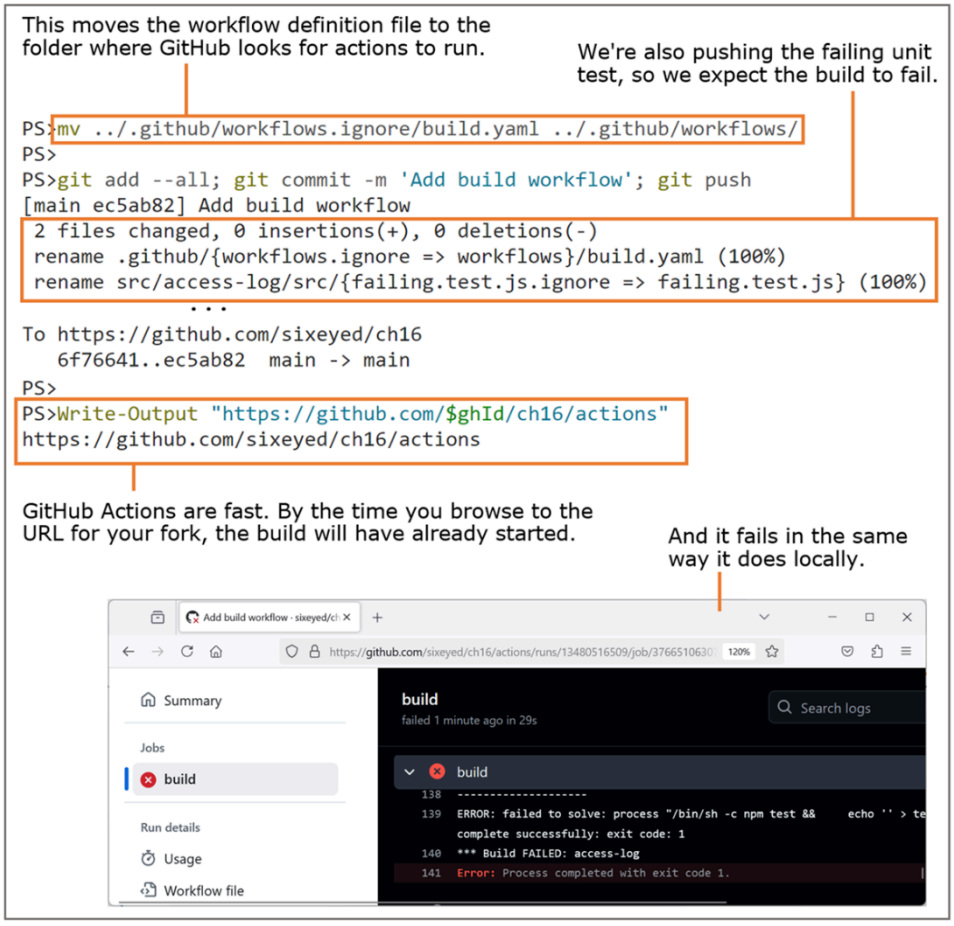


圖16.5在GitHub中觸發構建，該版本使用PowerShell腳本中的邏輯

當我們刪除失敗的測試並再次推動時（我們等於修復編碼錯誤或不良測試設置）時，它將觸發成功的新工作流程。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

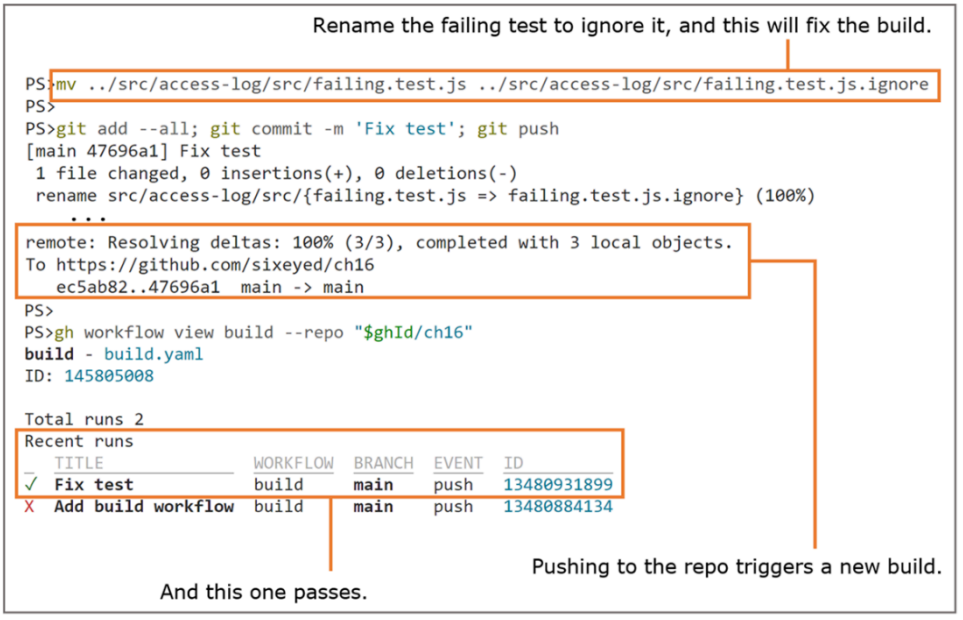
重命名失敗測試，使其被忽略，然後推動更改。 我正在使用GitHub CLI打印構建狀態，但是您可以使用Web UI，並且您會看到相同的內容。

```
＃重命名失敗測試：MV ../src/access-log/src/failing.test.js ../src/access-lo
g/src/failing.test.js.ignore



＃提交並推動您的更改：git add -all; git commit -m'fix test'; git
推


＃在Github中檢查新版本
```

到目前為止，您可能會覺得自己在本章中沒有做太多事情，但這確實是重點。 Dockerfiles正在做繁重的工作，您沒有什麼都不知道。 PowerShell腳本和YAML工作流確實只是在腳手架來構建Docker流程，如果您不想，您不必深入研究。在典型的DevOps交付中，團隊成員將擁有自己的重點，如果您可以在容器中進行複雜的工作並使其他所有人都很容易，那麼您將成為演出的明星。 圖16.6顯示了構建工作現在已刪除了不良測試。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>



f igure 16.6通過推動更改並修復github動作　　建造

這是構建的連續集成部分。 *Integration*在這裡意味著要為產品製作完整的代碼，這可能會由許多不同的團隊提供盡可能多的不同組件，並將其一起構建。 您要這樣做，以便這些團隊可以對建築物有一份視圖，他們可以迅速看到其區域的變化是否在其他地方引起問題。 接下來是連續部署，這意味著包裝進行分發，並確信您包裝的應用程序在部署時將起作用。

16.2將圖像推向GitHub容器註冊表

GitHub作為CI/CD提供商的主要景點之一是，它為您提供了一個地方：源代碼管理，構建服務和圖像註冊表（更不用說項目管理，問題跟踪，身份，權限，權限以及其他所有內容）。 GitHub容器註冊表（GHCR）遵循與GitHub Action相同的計費方法，它可以自由推動和拉動公共軟件包（對數據存儲的限制）。 您可以支付訂閱費用，這些訂閱為您提供更多的構建跑步者分鐘和更多圖像存儲千兆字節的私人存儲空間，但是免費配額對於許多項目都足夠慷慨。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

我們的CI/CD管道中的下一個階段是為每個組件構建Docker圖像，並將它們存儲在*staging*存儲庫中。這只是一個普通的圖像存儲庫，但我們將其用作未經證實的圖像的臨時存儲位置，因為它們僅通過了單元測試。一旦我們進行了分期的圖像，我們就可以使用它們來運行端到端測試，如果這些測試通過，我們知道這些圖像很好，並且可以將它們推到釋放存儲庫中。 當我們部署到生產中時，它將來自發行庫 - 圖16.7顯示了完整的管道，每個部分都將作為單獨的GitHub工作流程實現。

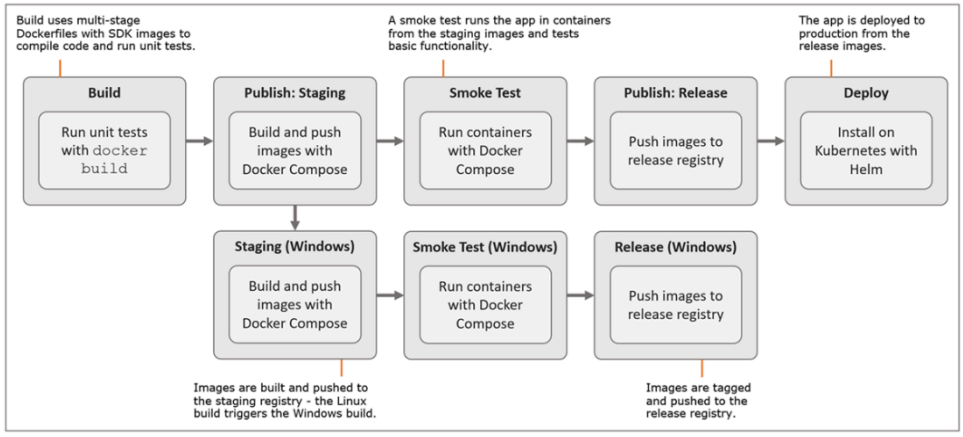


圖16.7我們將在本章中構建的完整多平台CI/CD管道

在我們的構建中，登台存儲庫將是公開的，但是在一個真正的項目中，它可能僅限於構建過程和開發團隊。每個圖像都將用一個構建編號標記，並且圖像中的標籤將存儲構建的審核跟踪。要構建和推動圖像的腳本非常簡單 - 它使用撰寫文件，並且幾乎只是Docker組成的構建，並且Docker構成了推動。組合文件中有一些不錯的觸摸，在保持文件易於讀取的同時，在需要的功能上進行了一些層次。清單16.4顯示了訪問日誌組件的不同文件中撰寫規格的一部分。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

清單16.4分層碼頭中的環境變量構成文件

```
＃docker-compose.yml-基本規格：訪問-LOG：圖像：CH16-ACCESS-LOG：
2E端口： - “ 8012：80”網絡： - app-net＃docker-compose-build.yml.yml-
構建部分：access-log：access-log：access-log：build：build：build：build：
build：：./ access-log grads： docker-compose-build-tags.yml-圖像參考變量：
訪問-LOG：

圖片：$ {註冊表}/$ {repository}/staging/
CH16-ACCESS-LOG：
2E- $ {docker_build_os} $ {os_version_tag} -
$ {docker_build_cpu} - $ {build_number}
```

每當您在構建管道上工作時，您都需要考慮兩個受眾：開發人員方和操作方面。開發人員可能希望使用Docker組合在本地運行該應用程序，如果需要調整配置，他們不希望100線YAML文件拖動。 因此，您可以在不同文件中隔離組合模型的不同部分 - 基本文件非常適合運行Docker組成；構建文件在構建上下文和參數中添加； 標籤文件添加了由環境變量填充的完整圖像參考。 OPS人們可以處理他們關心的部分，開發人員有一個不錯的干淨撰寫文件，用於本地運行。

構建腳本也是如此。構造它們是有禮貌的，以便它們可以在本地環境中運行，而這些環境可能沒有云構建的所有功能。下一個腳本我們將運行構建並將圖像推向登台存儲庫，但是我們沒有運行本地註冊表，因此我們可以傳遞一個標誌，告訴它在不推動的情況下進行構建部分。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

在本地運行下一個工作流的腳本 - 它使用相同的Dockerfiles，但完成了所有階段，而不是在設備測試下完成。構建後，檢查您的新圖像之一的標籤。

```
＃在不推動圖像的情況下運行構建腳本：./build-push.ps1 -
nopush


＃列出帶有舞台回購的圖像：Docker Image LS'註冊表：5010/diamol/ste
raging/*：*"


＃啟用其中一個圖像的標籤：Docker Inspect Rectry.Local：5010/diamol/staging/ch16-Image-Gallery
：2e-linux-Amd64-0 -f'{{json .config.labels}}
```

當您瀏覽此練習時，您會看到圖像命名為較長，笨重的參考，圖像中的標籤設置為默認值。 這是設計構建管道的另一件事 - 許多選擇取決於個人喜好或團隊的期望。我喜歡在不中斷開發的情況下盡可能多地使用真實的CI/CD過程的邏輯。因此，長圖像名稱 - 符合參考文獻有一些功能，即使在本地測試，即使這意味著名稱並不那麼容易使用。 圖16.8顯示了我的輸出 - 在您自己的管道中，您可能更喜歡為本地構建使用一個簡單的標籤。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

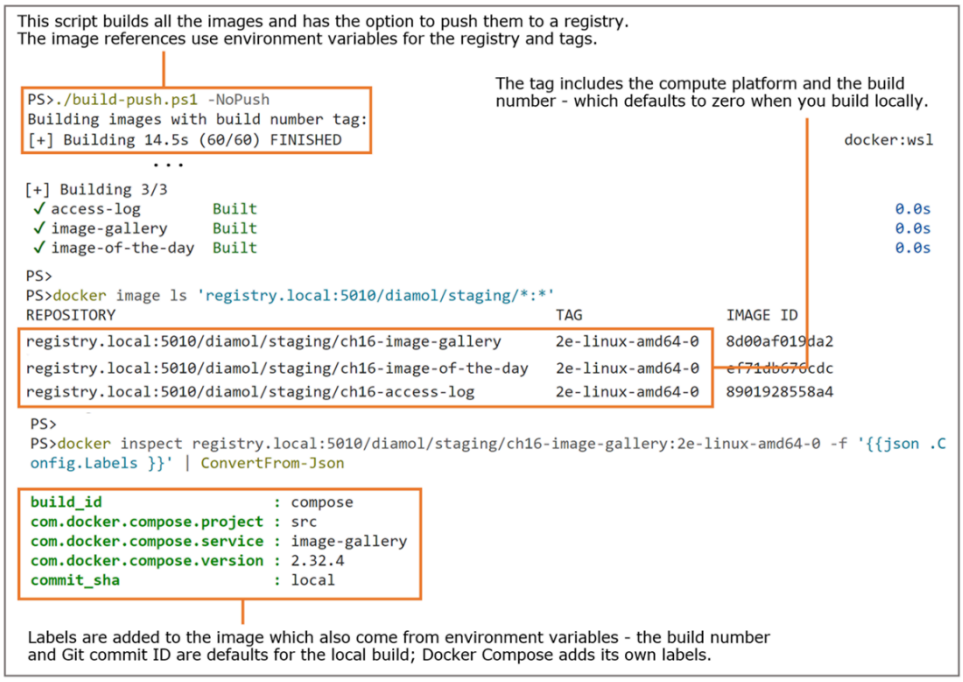


圖16.8下一個構建階段是一個腳本，該腳本構建和標記登台存儲庫的圖像。

是時候在Github運行此操作了。該應用將在支持Windows和Linux用戶的支持下發布。 構建腳本和DockerFiles都是跨平台，因此，如果您在Linux計算機上運行build-push.ps1，您將獲得Linux圖像，並且在Windows上，您將獲得Windows Images。 GitHub Action為這兩個操作系統提供了跑步者（僅在撰寫本文時在Intel CPU上 - ARM64支持即將進行）。 我們有兩個工作流程來構建和推動登台圖像。 他們使用相同的powershell腳本，並將環境變量從github設置為設置圖像引用。 您可以探索package staging.yaml和package starging-windows.yaml中的定義。 Linux和Windows版本之間的唯一區別是Runner的類型。

現在嘗試

有兩個階段工作流程 - 如果Linux運行成功，那麼它會觸發Windows運行。將它們都移到正確的文件夾中，然後推動更改以啟動構建。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
＃移動登台工作流程定義：mv ../.github/workflows.ignore/package-staging* .yaml ../.github/workflows/＃按下叉子：git add add -all; git commit -m“添加登台工作流”； git push＃列表所有工作流程：GH工作流列表 - repo“ $ ghid/ch16”＃打印Linux登台工作流的狀態：GH Workflow視圖套件套件 - REPO“ $ GHID/CH16”＃打開此鏈接以查看您存儲在GHCR中的圖像“ https://github.com/qunghid/ch16/pkgs/container/staging%2fch16-image-gallery”
```

這些工作流程被推動觸發，也從成功完成構建工作流程中觸發。 運行登台工作流程需要更長的時間，但是如果構建工作流量失敗，則不會執行，因此我們將保持快速失敗的理念，每個階段的做法盡可能少。圖16.9顯示了我在Linux上運行的分階段工作流程。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

There are two staging workflows which both run the same build-push.ps1 script. One runs on Linux and one on Windows.

```
PS>mv ../.github/workflows.ignore/package-staging*.yaml ../.github/workflows/
PS>
PS>git add --all; git commit -m 'Add staging workflows'; git push
[main 72014e0] Add staging workflows
 2 files changed, 0 insertions(+), 0 deletions(-)
 rename .github/{workflows.ignore => workflows}/package-staging-windows.yaml (100%)
 rename .github/{workflows.ignore => workflows}/package-staging.yaml (100%)
To https://github.com/sixeyed/ch16
   938f2ec..72014e0  main -> main
PS>
PS>gh workflow list --repo "$ghId/ch16"
NAME                    STATE   ID
build                   active  145808031
package-staging-windows active  145808355
package-staging         active  145808356
PS>
PS>gh workflow view package-staging --repo "$ghId/ch16"
package-staging - package-staging.yaml
ID: 145808356

Total runs 1
Recent runs
  TITLE                 WORKFLOW        BRANCH  EVENT  ID
* Add staging workflows package-staging main    push   13481106835
```

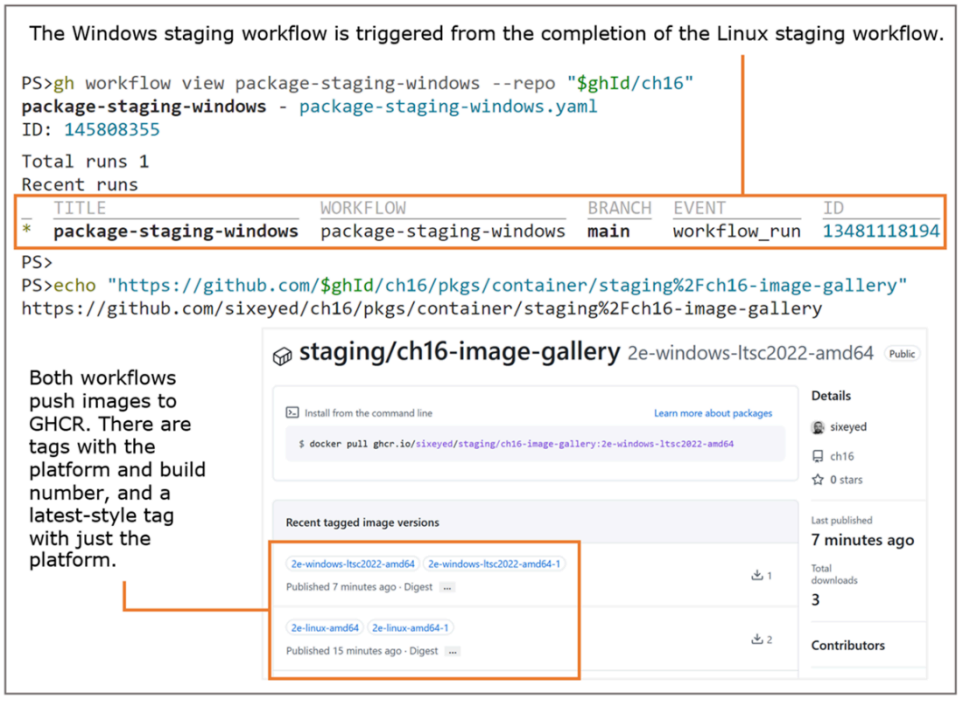
The Linux workflow is running - the event which triggered it is the push to the repo.

圖16.9推動工作流文件觸發Linux操作運行。

Windows Workflow需要更長的時間才能運行 - 部分是因為Windows Runner準備就緒需要更多時間，部分原因是Windows Base圖像更大，因此它們需要更多的時間來拉動。 同樣，僅當Linux工作流程成功時，才會觸發Windows Workflow。他們在不同的平台上做同樣的事情，因此，如果它在Linux上失敗，我們不想浪費時間等待它在Windows上也失敗。 圖16.10顯示了我的窗口正在進行中的構建以及兩個工作流程推動時的登台圖像。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>



數字　16.10 GitHub動作為您提供了一個多平台的建築農場　　- 免費。

您的github存儲庫是公開的，因此您在GHCR中的圖像默認為公開。您可能不希望為您的登台圖像 - 儘管很多開源項目都可以進行，因此社區可以測試新的版本並提供其反饋。您可以使用通常的Docker登錄過程對GHCR的私人註冊表進行身份驗證，但是您需要從Github生成令牌。 由於這些圖像是公開的，任何人都可以在不進行身份驗證的情況下將其拉動。

現在嘗試

從GHCR中拉出您的分期圖像之一，以確認它是可以公開訪問的，然後檢查圖像中的標籤。

```
＃無需為公共圖像進行身份驗證：docker拉“ ghcr.io/qulghid/staging/ch16-image-gallery:2e-linux-amd64”

＃檢查標籤以查看工作流中設置的值：docker Inspect“ ghcr.io/jerghid/staging/ch16-image-gallery:2e-linux-amd64” -f'{{json .config.labels}}
```

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這是包裝在最小基礎圖像上的GO組件，因此拉力將很快。將圖像參考和標籤與本地圖像進行比較，您會發現GitHub操作提供了我們流入DockerFiles並組成文件的其他上下文。 圖像中的標籤識別構建其構建的工作流以及源代碼的提交ID，您在圖16.11中看到了我。

GCHR images are rooted to the user account, not the repository. You can have multi-part names - these references include `staging` in the image name.

```
PS>docker pull "ghcr.io/$ghId/staging/ch16-image-gallery:2e-linux-amd64"
2e-linux-amd64: Pulling from sixeyed/staging/ch16-image-gallery
Digest: sha256:0e23a9fc1bef0a726906d8352f9386d0fe7b3d2a61f8306ef6a4e7e24154df84
Status: Image is up to date for ghcr.io/sixeyed/staging/ch16-image-gallery:2e-linux-amd64
ghcr.io/sixeyed/staging/ch16-image-gallery:2e-linux-amd64
PS>
PS>docker inspect "ghcr.io/$ghId/staging/ch16-image-gallery:2e-linux-amd64" -f '{{json .Config.Labels
}}' | ConvertFrom-Json

build_id                  : sixeyed/ch16/.github/workflows/package-staging.yaml@refs/heads/main
com.docker.compose.project : src
com.docker.compose.service : image-gallery
com.docker.compose.version : 2.27.1
commit_sha                : 2d5b36a9c18a1901bed0abf2091e91fc5d9e2486
```

The Dockerfile and Compose files are the same we used for the local build, but we set environment variables in the GitHub workflow to populate the build ID and Git commit.

圖16.11從GitHub容器註冊表中摘取工作流構建的圖像

現在，我們在所有組件的登台存儲庫中存儲了圖像。為了實現這一目標，我們已經通過了一些高質量的門：源代碼編譯，單元測試成功運行，並且docker映像構建。 但是，這並不意味著應用程序並不完整。 可能有一個缺少的配置文件或應用程序二進製文件的名稱可能是錯誤的，因此容器將無法啟動。下一個質量門是抽煙測試整個應用程序，並證明其啟動，運行並響應請求。

16.3 Docker撰寫的端到端測試

端到端測試意味著啟動整個應用程序並運行自動測試套件以驗證關鍵用戶旅程。 它曾經是一項耗時且昂貴的業務。 您可能需要將幾個應用程序組件與數據庫一起旋轉幾個應用程序組件和通信的消息隊列。 您需要基礎架構來運行所有這些，如果您擁有基礎架構，則不妨將測試環境留在整個時間內，這會鼓勵手動部署和配置變化，以及一切都過時和超出同步的風險。 以前，許多組織將完全跳過端到端的自動測試，因為它太複雜而無法設置和維護 - 他們要等到他們擁有釋放候選者並將其部署到環境中供用戶進行測試。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

不再！ 借助Cloud CI/CD服務以及在容器中建模和運行整個應用程序的能力，沒有任何藉口不在每個構建中運行自動化的端到端測試套件。該階段的運行時間比其他階段要長，但是通過像我們這樣的結構良好的管道，只有在我們已經對構建具有良好的信心時，它才會運行。 我們的端到端測試將是最小的，它們實際上只是煙霧測試，以確保應用程序合併在一起。 煙霧測試腳本使用Docker組成來啟動該應用程序，然後提出一些Web請求來驗證該應用程序至少可以做某事，然後將其全部關閉。

現在嘗試

本地運行煙霧測試腳本 - 它將在雲中運行時使用登台圖像，但是本地標誌使用自己的圖像構建支持開發人員。

```
＃標誌意味著我們使用本地圖像：./smoke -test.
ps1-局部
```

使用本地標誌運行測試腳本使用您在本章中已經構建的圖像 - 您可以在沒有該標誌的情況下再次運行它，它將從登台存儲庫中汲取最新圖像。和以前一樣，這是關於支持開發人員工作流程的。如果您進行了要測試的本地更改，則將使用自己的圖像，但是如果您要重現測試人員報告的問題，則可以使用登台圖像。圖16.12顯示了我的輸出，在該輸出中，要運行PowerShell腳本的單個命令在Docker上進行了很多工作。

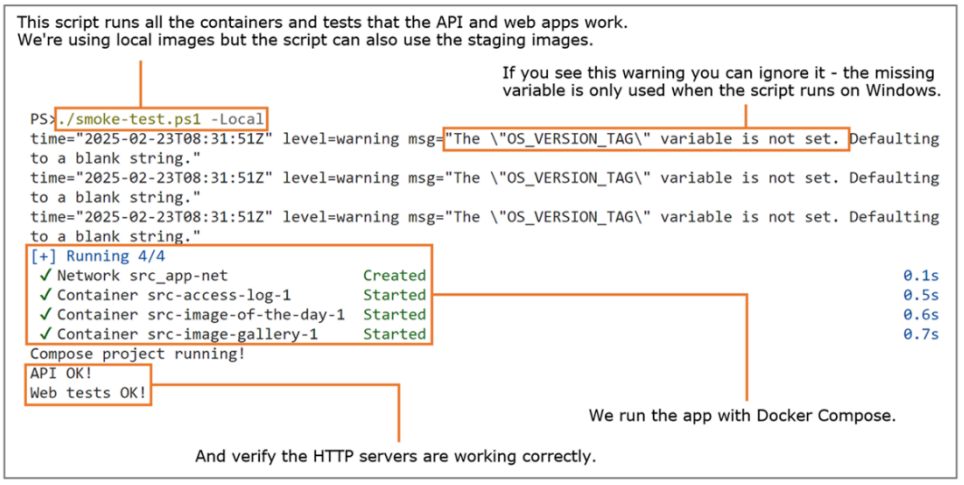


圖16.12運行煙霧測試以證明應用程序啟動和R 我們

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這些是非常基本的測試，但請記住，這是我們在這里關注的管道。一組更完整的測試可以使用行為驅動的發展（BDD）來定義人類語言的期望，使用測試框架和模擬瀏覽器執行。我們的構建可以輕鬆支持這些測試； 這只是意味著在煙霧測試腳本中添加命令。我們已經構建了構建，以使其易於擴展。一個更複雜的應用程序可能還需要大量的CPU或內存才能在容器中運行所有內容。 GitHub Actions跑步者即使在自由層（4個CPU內核和16GB RAM）中也具有合理的規格；在付費計劃中，您可以使用較大的跑步者來滿足您的應用程序的要求，但是標準跑步者適合我們的目的。

現在嘗試

我們有針對Linux和Windows容器的分期圖像，煙霧測試腳本是跨平台。將工作流YAML文件移至正確的位置，然後推動回購以觸發Linux和Windows Workflow運行。

```
＃移動兩個煙霧測試工作流程：mv ../.github/workflows.ignore/smoke-testthe.yaml ../.github/workflows/

＃提交和推動更改：
git add -all; git commit -m“添加煙霧測試工作流”； git推

＃如果您有CLI：GH工作流列表-Repo“ $ GHID/CH16”，請列出工作流程

＃檢查進度 - 或使用網絡：GH Workflow查看煙霧測試-Repo“ $ GHID/CH16” GH WorkFlow視圖煙霧測試 - 雷波“ $ GHID/CH16”
```

與往常一樣，Windows運行的時間比Linux上的時間更長 - 您可以在圖16.13中看到我的Windows Workflow仍在運行 - 但是幾分鐘後兩者都將是綠色的。 那是下一個質量的大門； 這意味著該應用程序在我們支持的兩個平台上都按預期工作。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The two workflows run the same smoke-test.ps1 script - one uses Linux and the other Windows. They will be triggered when you push changes.

```
PS>mv ../.github/workflows.ignore/smoke-test*.yaml ../.github/workflows/
PS>
PS>git add --all; git commit -m 'Add smoke test workflows'; git push
[main d9be5a2] Add smoke test workflows
 3 files changed, 9 insertions(+), 6 deletions(-)
 rename .github/{workflows.ignore => workflows}/smoke-test-windows.yaml (100%)
 rename .github/{workflows.ignore => workflows}/smoke-test.yaml (100%)
To https://github.com/sixeyed/ch16
   2d5b36a..d9be5a2  main -> main
PS>
PS>gh workflow list --repo "$ghId/ch16"
NAME                     STATE   ID
build                    active  145808031
package-staging-windows  active  145808355
package-staging          active  145808356
smoke-test-windows       active  145875677
smoke-test               active  145875678
PS>
PS>gh workflow view smoke-test --repo "$ghId/ch16"
smoke-test - smoke-test.yaml
ID: 145875678

Total runs 1
Recent runs
  TITLE                    WORKFLOW    BRANCH  EVENT  ID
✓ Add smoke test workflows smoke-test  main    push   13486198833
PS>
PS>gh workflow view smoke-test-windows --repo "$ghId/ch16"
smoke-test-windows - smoke-test-windows.yaml
ID: 145875677

Total runs 1
Recent runs
  TITLE                    WORKFLOW            BRANCH  EVENT  ID
* Add smoke test workflows smoke-test-windows  main    push   13486198839
```

We're building up the full CI/CD pipeline, using separate workflows for each stage.

The Windows workflow takes longer but will complete successfully like the Linux run.

圖16.13在Linux和windows上運行煙霧測試

現在，我們對構建有信心，我們可以將這些圖像推向發行庫。這就是生產部署將獲得圖像的地方，也可能是我們應用安全掃描或圖像簽名等政策的地方。 重要的是，我們不要從源頭重建圖像，以防萬一有任何更改，因為登台圖像被推了。通過對基本圖像或庫參考的更新可能是可能的 - 同一提交可以在下一個構建上產生不同的Docker圖像。我們已經測試了這些圖像，所以這是我們要推廣到發行庫的這些精確圖像。 如您所知，我們可以通過拉動登台圖像，給它們一個新標籤並再次推動它們來做到這一點。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

又有兩個工作流程要在此處推出 - 這些使用腳本來拉出最新的登台圖像，用發行庫名稱標記它們並將其推入。

```
＃移動發行版工作流程：MV ../.github/workflows.ignore/package-release* .yaml ../.github/workflo
ws/

＃推動更改以觸發構建：git add -all; git commit -m“添加釋放工作流”； git推

＃列出工作流：GH工作流列表-Repo“ $ GHI
D/CH16”

＃監視進度並在GitHub中檢查您的軟件包
```

如果您渴望在場景後面看起來更近一點，則會看到發行版工作流來調用一個腳本，該腳本使用登台撰寫文件來拉圖像並使用Release Compose compose文件來推動圖像。標記使用一些簡單的PowerShell將撰寫規格呈現為JSON，然後匹配源和目標圖像參考。 也許您仍然對在Mac或Linux框中安裝PowerShell感到不滿意 - 請看一下Tag-push.ps1，我很高興聽到您如何在Linux Shell腳本中執行相同的操作。 :)您不需要深入研究，只是知道撰寫文件集中了圖像命名方案並等待您的github構建以查看您獲得的內容 - 它看起來像圖16.14。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

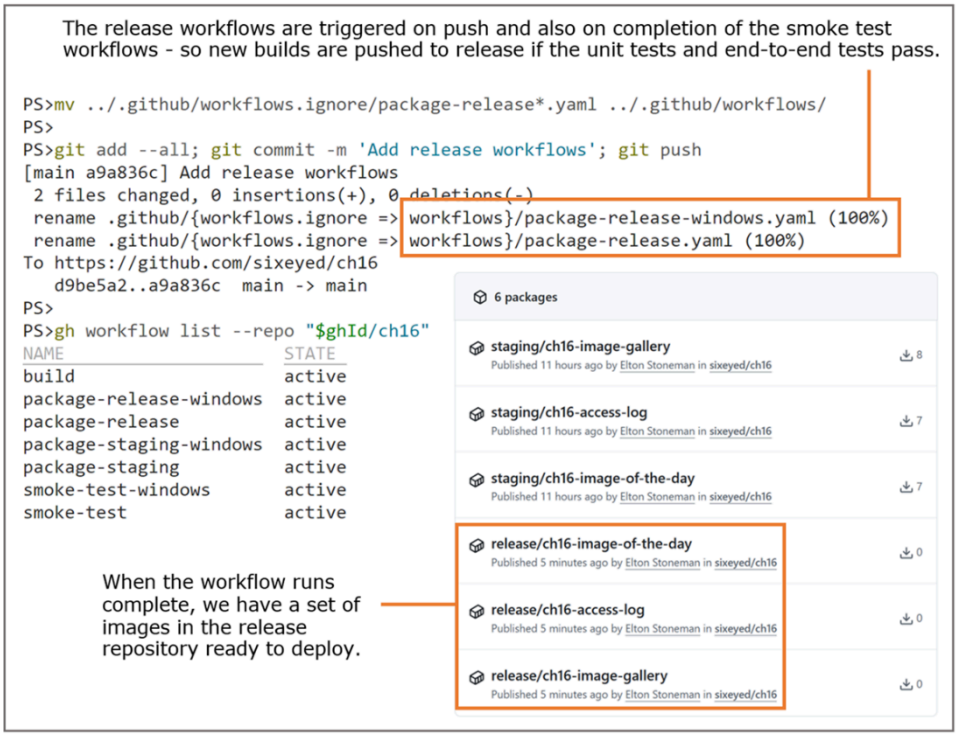


圖16.14標記登台圖像並將其推入版本存儲庫

幾乎是它。 我們已經從不同平台的來源構建了應用程序組件，在不同級別進行了測試，並將其打包以進行分發。 最後階段正在部署到生產（或類似生產）的環境中。

16.4部署到Kubernetes

將Docker用於CI/CD流程的最大優勢之一是使您的測試環境與生產盡可能相似。我們所有的組件都包裝在Docker圖像中，這意味著我們部署時不會遇到任何缺失的依賴關係或不匹配的運行時間。 我們已經測試了Docker組成的所有內容，因此我們證明了應用程序組件可以正確工作。但是，我們的生產部署不太可能使用Docker組成，因為這只是為我們提供了一台具有零可靠性或可伸縮性的機器。 因此，我們已經證明了使用錯誤平台的應用程序工作，這是一個妥協：撰寫應用程序以最少的計算要求啟動和運行，並且我們已經擁有開發人員在本地使用的模型。 不過，生產可能是Kubernetes，因此我們確實需要使用Kubernetes API對該應用進行建模並進行測試。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

這是另一個決定，這對個人喜好和項目的資產造成了很大的影響。 如果您的AutoScale始終配置了Kubernetes群集，則可能希望將其部署到管道的煙霧測試階段，以便您儘早在工作流程中儘早獲得生產。 在我們的情況下，我們將Kubernetes留在了最後，我們的最後階段將在GitHub Runner上創建一個臨時群集，只是為了測試該模型從發布圖像中部署而沒有任何錯誤。清單16.5顯示了使用社區支持的GitHub動作的容易。

清單16.5在github操作中使用K3D運行測試kubernetes群集

```
＃這將在容器中創建一個單節的kubernetes群集： - 名稱：create-cluster用途：ab
saoss/k3d-action@v2 with：cluster-name：ch16 args：> -  agents 1 -no-lb-no-lb -k3
s-k3s-arg-k3s-arg “ -no-deploy = v1} v1} v1}

＃這將設置kubectl cli使用該群集： - 名稱：set-context run：| kub
ectl config用途二下K3D-CH16
```

K3D（https：//k3d。io）是運行Kubernetes測試環境的有用工具。 它使用Docker在容器中運行Kubernetes節點，這些節點可以訪問他們正在運行的Docker引擎以安排吊艙。您也可以創建多節點簇，並使用它們來測試您的POD在節點上分佈，並且應用程序可以正確通信。 社區GitHub操作總結了K3D工具，因此您無需將其作為工作流程的一部分進行安裝和配置。這些參數絕對是深奧的，但他們只是設置一個沒有負載平衡器或K3D安裝的任何其他默認值的簡單的單節點群集。

您無法在本地運行工作流的這一部分，因為K3D設置都在自定義GitHub操作中。 我本可以腳本腳本的K3D用法，但這意味著要安裝另一個工具，此階段僅表示真正的部署。 Kubernetes模型使用Helm，而Deploy.PS1腳本設置了圖像引用的值以使用發布存儲庫。 如果您從第15章中獲得了Kubernetes的品味，那麼Helm/iotd文件夾中的圖表是掌舵的非常簡單的介紹。 GitHub Actions跑步者已安裝Kubectl和Helm命令行，因此我們只需要推動最終的工作流定義以完成管道即可。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

將部署工作流程添加到您的存儲庫中，並在Github UI中觀看構建，以查看其運行方式。

```
＃移動部署工作流程：mv ../.github/workflows.ignore/deploy.yaml ../.github/workflo
ws/

＃推動更改：
git add -all; git commit -m'添加部署工作流'; git推
```

我們生活在一個世界中 - 此工作流使用開源GitHub操作來部署開源工具來創建Kubernetes群集，在Docker中運行在雲中的託管服務上，我們沒有支付一分錢。然後，它為我們的應用程序部署了Kubernetes模型，該模型插入了Pods中，這些吊艙將我們剛剛推到GitHub容器註冊表的圖像。 我在圖16.15中的輸出顯示了連續的部署階段（儘管到了一個立即被破壞的環境）。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

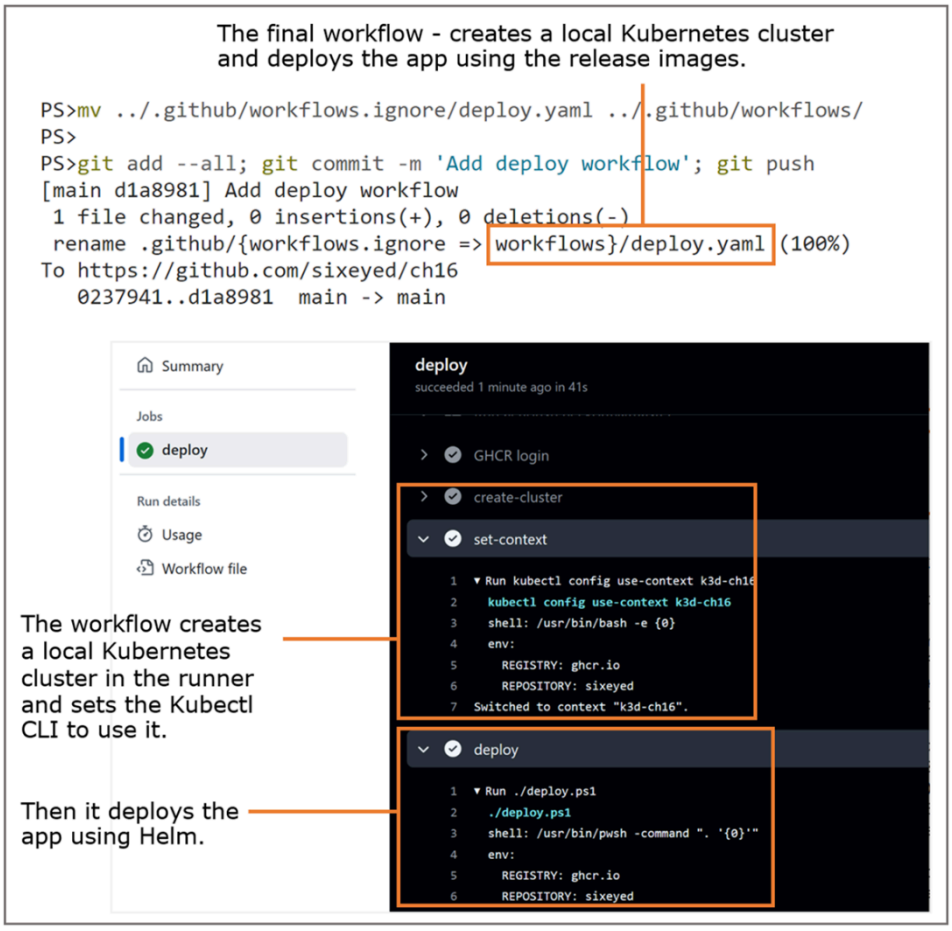


圖16.15通過部署到本地Kubernetes群集模擬生產部署

我們現在已經構建了整個管道，並分別測試了每個階段，因此最終*final*步驟是確保當我們推動源代碼更改時整個過程運行。 GitHub工作流可以具有多個觸發事件 - 如果工作流定義本身更改，然後使用其他觸發器來建模過程，則需要觸發運行。 請記住該過程從圖16.7中。 我們的第一個工作流程將從推送到存儲庫中觸發，然後從成功完成其前身的每個後續工作流程觸發器。 Windows Workflows可以與Linux的工作流並行運行，最終部署僅取決於要按下的Linux Release圖像。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

更改GO應用程序中的HTML頁面，然後將其推向您的存儲庫。觀看Web UI中的操作頁面，您將看到所有工作流程以正確的順序觸發，並以虛擬部署到Kubernetes結束。

```
＃覆蓋一個源文件：mv -fo ../ src/image-gallery/index.html.ignore ../src/image-gallery/index.html


＃推動更改：
git add -all; git commit -m'確認apod'; git推

＃觀看管道展開：
Echo“ https://github.com/qunghid/ch16/actions”
```

整個管道運行需要幾分鐘，大部分時間都花在Windows工作流程上。 就像圖16.16所示一樣，在Windows零件仍在運行時，您的Linux構建和部署很可能已經完成。 許多應用程序不需要多平台支持，因此您可以在構建中沒有這些階段的情況下做。但是，如果您編寫dockerfiles並構建腳本以跨平台，那麼在管道中添加Windows，ARM64或RISC-V Image構建將非常容易。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

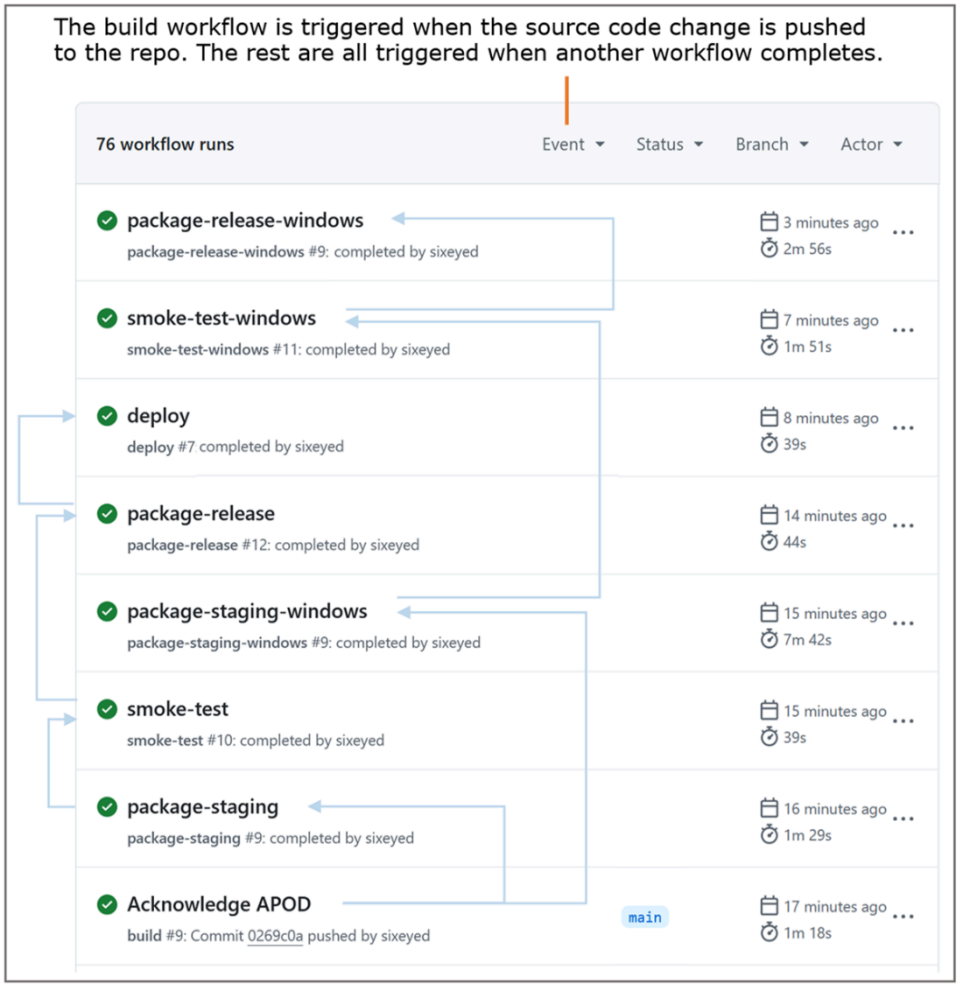


圖16.16從更改到源代碼觸發的完整CI/CD管道。

僅此而已：雲中的docker和kubernetes的完全剛起步的CI/CD工藝。 我故意關注碼頭零件和整體方法，而不是挖掘github工作流程語法和powershell，但是您在存儲庫中擁有所有這些資產，並且可以將其用作您自己的管道的基礎。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

16.5構建便攜式構建過程

我們在本章中構建的管道與真實生產CI/CD設置相去甚遠。 商業項目的最大區別是GitHub存儲庫和包裝將是私人的。 這需要在工作流程中進行更多的配置才能進行身份驗證，但這都是非常標準的內容，而GitHub的操作已得到充分記錄。然後是一些設計選擇 - 您可能更喜歡擁有一個工作流，其中所有階段在一個地方定義。 這使建模依賴項並在UI中看到它們變得更加容易，因為它顯示了工作流程中不同作業的圖表，但是如果需要，它使運行單獨的階段更加困難。除此之外，這是一個相當不錯的入門項目，下次您需要在GitHub操作中進行構建，並且對於任何其他CI/CD服務，這只是您需要更改的YAML。 這是我構建利用平台的管道的建議，而無需犧牲可移植性：

- 在Docker中盡可能多地做。您可能會在本書中期待這個建議，但這是一種普遍的好方法。在您的Dockerfile中編譯代碼和運行單元測試，您沒有安裝的依賴項來構建應用程序，並獲得了一定的信心。用曲目建模您的應用程序，您可以輕鬆地旋轉煙霧測試環境並集中圖像命名方案。

- 在腳本中做其他所有操作。隔離每個構建階段，您將擁有一組簡單，易於管理的腳本，這些腳本主要是運行熟悉的Docker命令。腳本的作用是協調工作並充當Docker和構建服務之間的中間位置 - 將諸如構建號（構建號）等管道上下文轉換為Docker使用並組成的環境變量。

- 支持開發人員工作流程和OPS工作流程。管道將主要在CI/CD服務中運行，但是如果您保持腳本靈活，那麼它們也適合開發人員使用。能夠使用與構建服務完全相同的基礎架構來構建代碼和運行測試，應消除開發人員檢查失敗測試並打破構建的情況，或者至少給您良好的理由來大喊大叫（或鼓勵他們，取決於您的團隊文化）。 不要因為構建服務提供的所有閃亮的東西而陷入困境。您使用的任何專有功能都將使移動另一個提供商更加困難。我想不出我從事的一個長期項目最終並沒有在兩次服務之間移動，而那些腳本構造的項目最容易遷移。您需要務實，並且不要花時間構建服務免費服務的時間，但是您應該嘗試將特定於提供商的功能保持在最低限度。

記住：這只是我的看法。您可能不會選擇（或者您都沒有選擇）遵循我的建議，這很好，但確實要牢記指導原則。他們將幫助您，無論您走哪一條路。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

16.6實驗室

這款超級duper CI/CD管道中缺少的一件事是，我們為不同平台構建了圖像，但是我們不將其作為一組多平台圖像發布。 您可以將清單列表推向GHCR，因此該實驗室讓您執行最後一步：

- 編寫一個腳本，該腳本為本章應用程序中的三個組件中的每個組件創建並推出清單列表。 每個清單列表應包含Intel（AMD64）CPU體系結構的Linux和Windows變體。
- 您的腳本應遵循與其他腳本相同的方法，使用撰寫文件作為圖像參考的來源。 當您在本地測試您的腳本時，請編寫GitHub操作工作流以運行腳本。
-
-

My solution is in the lab folder of the ch16 repo. The documentation is in the README file, which even has pictures to prove that the solution works: https://github.com/diamol2e/ch16/blob/main/lab/README.md.

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

17 優化您的Docker圖像以進行尺寸，速度和安全性

一旦將應用程序包裝到CI/CD流程以部署到容器平台上，您可能會認為您可以擅長進行生產，但是仍然需要一些最佳實踐來投入時間。優化Docker圖像是最重要的，因為您需要構建和部署的播放器，並且您的申請書可以自由訪問，並且需要自行培訓，並且需要自行培訓，並且需要將自己的檔次置於範圍內。磁盤空間。 Dockerfile語法小巧而直觀，但它隱藏了一些複雜性，您需要了解以充分利用圖像的構建。

本章將引導您瀏覽圖像格式的更細節，以便您知道如何以及為什麼優化它。我們將在第3章中構建，您了解到Docker圖像實際上是從多個圖像層合併的。

17.1如何優化Docker圖像

Docker圖像格式已經進行了大量優化。在可能的情況下在圖像之間共享層，這會減少構建時間，網絡流量和磁盤使用情況。但是Docker對數據有保守的方法，並且它不會自動刪除您拔出的圖像，這是您需要明確做的事情。因此，當您更換容器以更新應用程序時，Docker將下載新的圖像層，但不會刪除任何舊的圖像層。 磁盤很容易被許多舊圖像層吞噬，尤其是在定期更新的開發或測試機上。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

您可以看到圖像與系統DF命令進行物理使用的磁盤空間，該命令還顯示了容器，音量和構建高速緩存磁盤的使用情況。

```
Docker系統DF
```

如果您從未清除Docker引擎中的舊圖像，則可能會對結果感到驚訝。我的輸出來自圖17.1，來自運行兩個Docker發動機的Windows計算機。即使我沒有運行任何容器，我總共使用90GB的存儲空間有100多個圖像。

Prints disk usage including the image cache, container writeable layers, volumes and the build cache.

This is a Linux Docker Engine, using 1.7GB of image storage with no containers running.

```
PS>docker system df
TYPE            TOTAL     ACTIVE    SIZE       RECLAIMABLE
Images          12        0         1.774GB    1.774GB (100%)
Containers      0         0         0B         0B
Local Volumes   0         0         0B         0B
Build Cache     73        0         22.12MB    22.12MB
PS>
PS>docker context use windows
windows
Current context is now "windows"
PS>
PS>docker system df
TYPE            TOTAL     ACTIVE    SIZE       RECLAIMABLE
Images          99        0         90.89GB    90.89GB (100%)
Containers      0         0         0B         0B
Local Volumes   6         0         83.89MB    83.89MB (100%)
Build Cache     0         0         0B         0B
```

For comparison here's a Windows Docker Engine - with 99 images in the cache, taking up 90GB of storage.

圖1　7.1很容易看到您的磁盤被Docker Images吞沒了，您不是E　使用。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這個示例是一個溫和的示例 - 我已經看到了多年運行Docker的非公開服務器，在未使用的圖像上浪費了數百千兆字節。 經常運行Docker System的習慣是一個很好的習慣 - 它可以清除圖像層和構建緩存而無需刪除完整的圖像。 您可以通過計劃的作業運行它來刪除未使用的層，但是如果您的圖像進行了優化，則會不再是問題。優化技術堆棧的一部分通常是一個週期性的過程，並進行了許多小改進，但是藉助Docker，通過遵循一些簡單的最佳實踐，可以很容易地進行大量改進。

除非您需要，否則第一個不是將文件包含在圖像中。聽起來很明顯，但是您通常會編寫一個dockerfile，該文件夾在整個文件夾結構中復制，而不會意識到該文件夾包含文檔，圖像或其他在運行時不需要的二進製文件。 明確您複製的文件可能是您製作的第一個大保存。比較列表17.1中的Dockerfiles（在整個文件夾中的第一個示例副本，而第二個示例實現了副本添加了一些額外的文件，並包含一個新的步驟來刪除它們。

清單17.1試圖通過刪除文件來優化dockerfile

```
＃Dockerfile V1  - 整個目錄結構中的副本：摘自Diamol/base：2e CMD Ec
ho App- && && LS App && echo docs- && && ls docs copy。 。



＃dockerfile v2  - 添加了一個新步驟，從diamol/base刪除未使用的文件
：2e cmd echo app- && && ls app && echo docs- && && ls docs copy。
。 運行RM -RF文檔
```

在V2 Dockerfile中，您會認為圖像大小會較小，因為它會刪除額外的文檔文件夾，但這不是圖像層工作的方式。 圖像是所有層的合併，因此文件仍然存在於復制層中；它們只是隱藏在刪除層中，因此總圖像大小不會縮小。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

構建示例並比較大小。

＃切換到源文件夾：CD CH17/練習/build-context＃構建第一個圖像：
Docker Image Build -T Diamol/ch17-Build-context：2e-v1。

＃構建所謂的優化圖像：Docker Image Build -T Diamol/CH17-Build-Context：2e-v2 -f ./dockerfi
le.v2。

＃比較尺寸：Docker Image LS“ Diamol/ch17-Build-cont
ext*”

您會發現V2映像與V1圖像完全相同，就好像刪除文件夾根本沒有運行的RM命令一樣。您可以在圖17.2中看到我的輸出 - 我正在使用Linux容器，因此尺寸很小，但幾乎一半的大小來自DOCS文件夾中不必要的文件。

Building images from the different Dockerfiles. The `--quiet` (or `-q`) flag silences the usual build output.

```
PS>cd ch17/exercises/build-context
PS>
PS>docker image build -t diamol/ch17-build-context:2e-v1 . --quiet
sha256:612ca20cd70394778fab7f28fcc8848d7eb3cb4dcd2fd49a98818e37fc8741f3
PS>
PS>docker image build -t diamol/ch17-build-context:2e-v2 -f ./Dockerfile.v2 . -q
sha256:b3c61df4a4a0d639dc0dbc34f810da0ee863c83bdaf2efa19548498243227504
PS>
PS>docker image ls "diamol/ch17*"
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
diamol/ch17-build-context   2e-v2     b3c61df4a4a0   37 seconds ago   15.3MB
diamol/ch17-build-context   2e-v1     612ca20cd703   55 seconds ago   15.3MB
```

The v2 image deletes files, but the image size is the same. The image is a merge of all layers, so the delete layer just hides file from the previous layer.

圖17.2驚喜！如果刪除在自己的層中，則刪除文件不會降低圖像大小。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

Dockerfile中的每個指令都會產生一個圖像層，並將層合併在一起以形成整個圖像。如果將文件寫入一層，這些文件將永久存在；如果將它們刪除在後續層中，那麼所有Docker都會將它們隱藏在文件系統中。當您進行圖像優化時，要理解這是一件基本的事情 - 嘗試在以後的層中刪除膨脹是不好的 - 您需要優化每一層。 您可以通過打印圖像歷史記錄來看到層及其尺寸。

現在嘗試

Docker中的合併文件系統構建了效率低下的圖像。 打印圖像歷史記錄以顯示圖層的物理結構。

```
＃從V1圖像運行一個容器：Docker Run-IT Diamol/ch17-Bu
ild-context：2e-v1

＃Docs文件夾不在V2映像中：Docker Run-IT Diamol/ch17-
Build-context：2e-v2

＃檢查V2圖像的歷史：Docker歷史記錄Diamol/ch17-Build-
context：2e-v2
```

圖層歷史記錄是查看圖像大小崩潰的有用方法。當您從這兩個圖像中運行容器時，您會看到在V2構建中無法訪問文件 - 但歷史記錄顯示了添加大文件的複制層。一旦數據進入一層，就無法將其從圖像中刪除。圖17.3顯示了我的輸出 - 如果您查看Docker Hub上的2E-V2標籤的圖像詳細信息（https：//hub。docker.docker.com/ r/diamol/diamol/ch17-build-上下文/標籤），則可以看到相同的圖層信息。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The container just prints out directory contents.

In the v2 image there are no contents for the docs directory, because they were deleted in the Dockerfile.

```
PS>docker run -it diamol/ch17-build-context:2e-v1
app-
init.txt
docs-
README.md  whale.jpg
PS>
PS>docker run -it diamol/ch17-build-context:2e-v2
app-
init.txt
docs-
ls: docs: No such file or directory
exit status 1
PS>
PS>docker history diamol/ch17-build-context:2e-v2
IMAGE          CREATED       CREATED BY                                      SIZE
4d27fcd4fe87   2 days ago    RUN /bin/sh -c rm -rf docs # buildkit           0B
<missing>      2 days ago    COPY . . # buildkit                             2.1MB
<missing>      2 days ago    CMD ["/bin/sh" "-c" "echo app- && ls app && …   0B
<missing>      2 weeks ago   RUN /bin/sh -c apk add --no-cache curl # bui…   5.38MB
<missing>      2 weeks ago   CMD ["/bin/sh"]                                 0B
<missing>      2 weeks ago   ADD alpine-minirootfs-3.20.6-x86_64.tar.gz /…   7.8MB
PS>
```

Printing the image history shows the layers - the 2MB COPY layer is still part of the image, even though the files are not available in the final image.

圖17.3合併的文件系統隱藏已刪除的文件，但這些圖層仍然是圖像的一部分。

這是有關優化的第一點 - 不要將任何內容複製到您不需要運行應用程序的圖像中。 即使您嘗試在以後的說明中刪除它，圖像堆棧中仍然存在一些磁盤空間。精確地在復制說明中僅將您想要的文件帶入圖像中要好得多。 這使圖像尺寸較小，並且在您的Dockerfile中進行了更清晰的安裝。列表17.2顯示了此簡單應用的優化V3 Dockerfile，唯一與V1更改是它複製了應用程序子文件夾，而不是整個目錄。

清單17.2僅複製必要文件的優化碼頭

來自Diamol/Base

CMD Echo App- && LS App && echo docs- && && ls docs co
py ./app ./app

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

如果構建此功能，您會發現圖像大小較小，並且構建上下文也小。 Docker壓縮構建上下文（您運行構建的文件夾），並在運行構建時將其與Dockerfile一起發送到引擎。這就是您可以通過本地計算機上的文件在遠程引擎上構建圖像的方式。構建上下文通常具有您不需要的文件 - docker用於發送整個文件夾，但是在buildKit（圖像構建器組件）中進行了優化檢查dockerfile，以查看需要包含哪些內容。

在簡單的Dockerfile中，您可以使用精確的複制命令，並在一個較小的上下文中獲得兩個優化，並加快構建的速度，而較小的圖像加快了推動和拉動。在更複雜的應用程序中，很難做到這一點，在更複雜的應用程序中，Dockerfile是嵌套文件夾結構的一部分。您可以最終獲得許多複雜的複制說明，並帶有特殊的文件匹配模式污染您的Dockerfile。 還有一個替代優化，可以實現相同的結果 - 您可以列出文件和文件夾，以在稱為.dockerignore的文件中排除在構建上下文中。忽略的內容不會在構建上下文中發送，因此您無需在復制命令中具體特定，因為源文件夾已經排除了您不需要的所有內容。

現在嘗試

由原始的Dockerfile構建，該Dockerfile在整個上下文文件夾中復制，然後使用.dockerignore文件再次構建，以減少上下文和圖像的大小。

```
＃構建V3圖像 - 不要使用緩存：Docker Build -T Diamol/CH17-Build-Context
：2e-V3 -no-cache。

＃現在重命名已準備好的忽略文件並檢查內容：MV Dockerignore.txt .dockerignore

貓.dockerignore

＃再次使用一個新標籤運行相同的構建：docker build -t diamol/ch17-build-co
ntext：2e-v4 -no-cache。
```

您會看到，在第一個構建命令中，Docker向引擎發送了2 MB的構建上下文。 這就是該文件夾中文件的未壓縮大小，其中大部分是鯨魚的2 MB圖片。在第二個構建中，當前目錄中有一個.dockerignore文件，該文件告訴Docker排除Docs文件夾，因此構建上下文僅為190字節。如果您列出圖像，則會看到V4標籤比其他標籤小2MB，因此我們終於實現了正確的優化。您可以在圖17.4中看到我的輸出。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

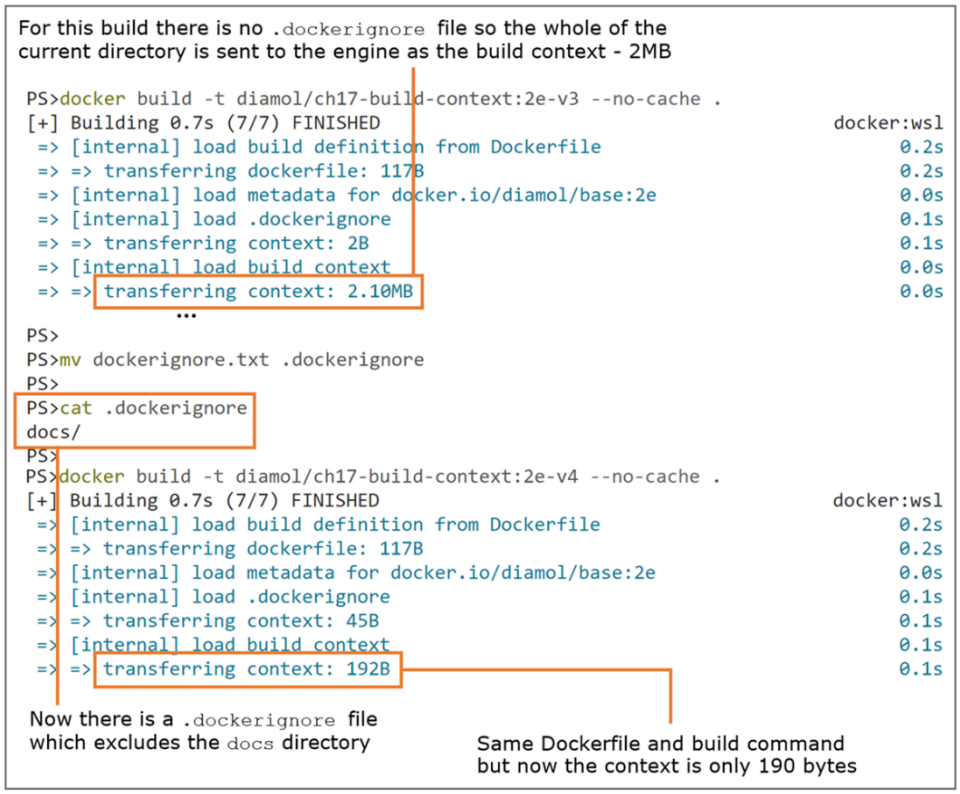


圖17.4 USIN g a .dockerignore文件減少了構建上下文的大小和tim e發送。

當您計算在構建上下文中發送未使用的數據的成本時，一個.dockerignore文件也可以節省大量時間，即使您在dockerfile中使用顯式路徑也可以節省空間。 可能是您在本地構建代碼，還可以使用多階段的構建來編譯Docker中 - 您可以在.dockerignore文件中指定構建二進製文件，並確保本地構建輸出不會復製到圖像中。文件格式與Git的.gitignore文件相同，您可以從GitHub使用App Platform的模板作為一個很好的起點（如果您的DockerFile也是Repo的根源，則應包括Git History Folder .git。

現在，您已經看到了管理哪些文件進入您的Docker映像的重要性，我們將退後一步，看看您用作基礎的圖像。

17.2選擇正確的基本圖像

基本圖像選擇與安全性一樣多，與磁盤空間和網絡傳輸時間有關。如果您的基本操作系統圖像很大，則可能具有各種工具，這些工具可能在真實的機器上有用，但可能是容器中的安全孔。如果您的操作系統基本映像已安裝*curl*，則攻擊者可以使用它下載惡意軟件或將數據上傳到其服務器，如果他們設法將您的應用程序插入到容器中。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

應用程序平台基礎圖像也是如此。 如果您正在運行Java應用程序，則OpenJDK官方圖像是一個很好的基礎（在Docker Hub上以Eclipse-Temurin的形式出版），但是有許多標籤具有不同的Java Runtime（JRE）和開發人員SDK（JDK）的配置不同的標籤。表17.1顯示了SDK的多平台圖像與運行時與最小版本的多平台圖像之間的尺寸差異：

選項卡17.1兼容Java 22碼頭上的尺寸差異 UB

eclipse-temurin	:22-jdk	:22-jre	:22-jre-alpine	:22-jre-nanoserver
Linux	440 MB	281 MB	189 MB	
Windows	3.78 GB	3.45 GB		444 MB

Linux用戶可以使用189 MB的基本圖像而不是440 MB，並且Windows用戶只需檢查Docker Hub上的變體並選擇具有最小的OS映像和最小的Java安裝的變體，而不是使用444 MB而不是3.8 GB。 OpenJDK團隊對他們的多平台圖像謹慎。他們選擇兼容性最寬的圖像，但是使用較小的變體測試您的應用程序很容易。嘗試使用Alpine或Debian Slim圖像作為Linux容器的基本OS，以及用於Windows容器的Nano Server（替代方案是Windows Server Core，它幾乎是完整的Windows Server OS，這就是為什麼這些圖像攝入磁盤的Gigabytes的原因）。並非每個應用程序都可以與較小的變體一起使用，但是很容易從行中切換圖像並進行測試。

尺寸不僅與磁盤空間有關，還與使用該空間有關的內容。 最大的OpenJDK圖像包括整個Java SDK，因此，如果有人設法妥協了您的容器，那裡有一個不錯的攻擊向量。他們可以將一些Java源代碼文件寫入容器的磁盤中，使用SDK編譯它們，然後運行一個應用程序，該應用程序在您的應用程序容器的安全性上下文中執行他們想要的任何操作。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

在本章的練習中是使用默認JDK映像的Java應用程序。 它運行一個非常簡單的REST API，該API總是返回值true。

```
＃來自存儲庫根文件夾：CD CH17/練習/Tr
uth-App

＃構建圖像 - 基本圖像使用：22-JDK標籤：Docker Image Build -T Diam
ol/ch17-Truth-app：2e。

＃運行應用程序：
Docker Run -D -P 8010：80-名稱真相Diamol/ch17 -Truth -app：2e

＃並嘗試一下：
curl http：// localhost：8010/Truth
```

您正在運行的容器配備了圖像中編譯的Java REST API，但它還具有編譯其他Java應用程序的所有工具。如果攻擊者設法突破應用程序並在容器上運行任意命令，則可以運行自己的代碼來執行自己喜歡的任何事情。 在此圖像中，我“不小心”包含一個測試代碼文件，惡意用戶可以找到並運行該文件以更改應用程序的行為。

現在嘗試

通過連接到API容器中的外殼來模擬容器突破。 然後使用JDK編譯並運行測試代碼，然後再次檢查應用程序。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
＃連接到API容器 - 對於Linux容器：Docker Container exec- It Truth S
H

＃或Windows容器：Docker Container exec -it
真相CMD

＃在容器中編譯並運行測試Java文件：Javac FileUpdateTest.java Java Java
FileUpDateTest Exit



＃返回機器，再次嘗試API：curl http：// localhost：8
010/truth
```

您會看到該應用程序的行為發生了變化。在包裝的圖像中，API調用始終返回真實 - 但是測試固定設置將響應設置為false。 我在圖17.5中的輸出顯示了“黑客”之後的原始響應和變化的響應。

圖17.5在您的應用程序圖像中具有SDK，使您可以接受任意代碼執行攻擊。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這是一個稍微人為的例子，圖像中的方便測試文件使事情變得容易，但是容器突破是可能的，這說明了一個有趣的攻擊選項。 該容器可以被平台鎖定以防止網絡訪問，此攻擊仍然可以工作。教訓是，您的基本圖像應該擁有運行應用程序所需的一切，但是沒有用於構建應用程序的額外工具（解釋的語言（例如Node.js和Python）是一個例外，因為該應用程序對於應用程序運行是必需的）。

黃金圖像是解決這個問題的一種方式。您有一個團隊，可以選擇正確的基本圖像並為您的組織構建自己的版本。 我在這本書中使用了這種方法 - 我的Java應用程序是由Diamol/OpenJDK：2E構建的，這是一個多平台圖像，使用每個OS的最小變體。我可以控制黃金圖像更新的頻率，並且可以在構建黃金圖像後觸發應用程序圖像構建。構建自己的黃金圖像的另一個優點是，您可以使用Trivy等第三方工具在構建過程中的基礎層上集成其他安全檢查。

現在嘗試

Trivy是用於安全掃描Docker圖像和其他內容的開源項目。您可以在本地安裝Trivy命令行或將其運行在容器中。

```
＃創建一個用作數據緩存的捲：Docker卷創建Trivy
Cache

＃在本書的OpenJDK映像上運行漏洞掃描：Docker Run -v Trivy -Cache：/root/.cache-cache -it diamol/triv
y：2e image -scanners vuln vuln diamol/openjdk：2e

＃運行相同的掃描但忽略低嚴重性問題：docker run -v trivy -cache：/root/.cache-cache -it diamol/trivy：2e
image -scanners vuln -severity -severity -severity high，critical diamol/openjdk：2e

＃最後忽略了我們已經接受的特定問題：docker run -v trivy -cache：/root/.cache -it diamol/trivy：2e image
-scanners vuln-- severity -severity high，critical -ignoreFile/sample/sample.trivyignore diamol/openjdk：2e
```

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Trivy需要一段時間開始進行第一次掃描，因為它下載了已知安全問題的數據庫。 這就是為什麼我們創建卷的原因 - 使數據被緩存，隨後的運行不需要再次下載。通常，您會將Trivy集成到CI/CD進程中 - 您可以將容器作為集中式服務器運行以供其他構建使用 - 因此，只有在您首次部署它以及數據庫需要更新時，此命中才會發生。圖17.6顯示了我的輸出；當我進行掃描時，最終的基本圖像是Alpine版本3.20.3，它沒有關鍵的漏洞，但有三個高度問題。

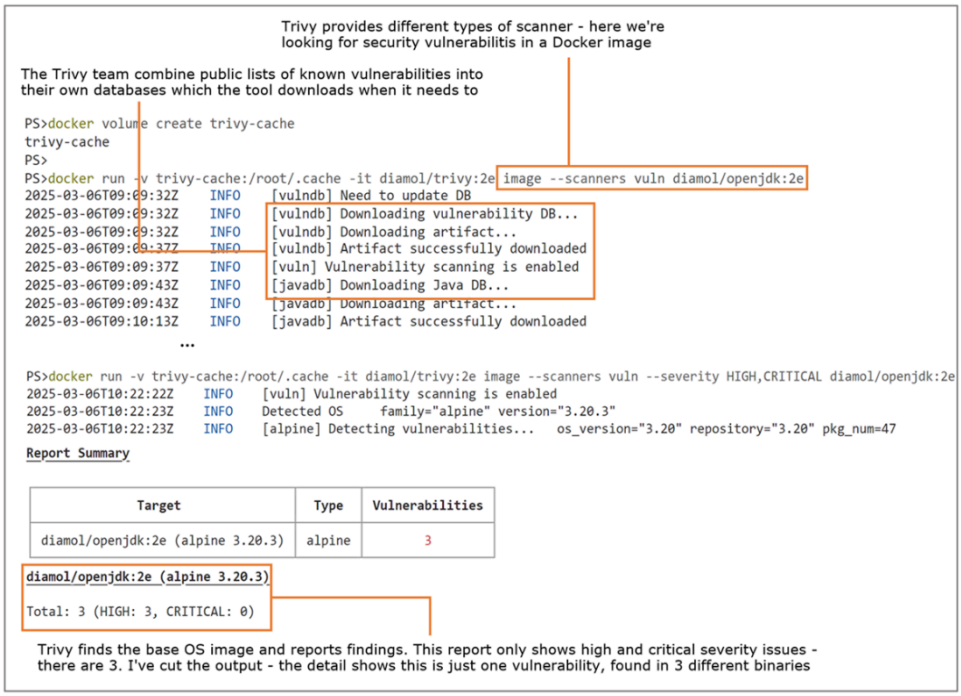


圖17.6使用Trivy分析Docker圖像是否已知安全漏洞。

漏洞數據庫不斷更新，因此您的輸出可能與我的輸出不同。 我發現一個高度嚴重性問題-CVE -2024-12797-存在於三個圖書館中。 Trivy的輸出告訴您問題是否已固定在較新版本的組件中，其中包括描述漏洞的鏈接。

當您發現這樣的問題時，您需要就如何進行自己的決定做出決定。對於您來說，這可能至關重要，在這種情況下，如果有修復程序，則更新脆弱的組件，或者阻止部署，直到釋放修復程序為止。或者，您可能認為這不是問題 - 在這種情況下，這是HTTPS的晦澀使用，可能會受到中間攻擊的人。我相信我的一個應用都不使用RAW公共密鑰（RPK），這是利用的工作方式，因此我將CVE編號添加到.trivyignore文件中。您將其保留在源文件夾中，以抑制Trivy的警告，該警告的優勢是跟踪您隨著時間的推移所抑制的內容。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

Trivy的另外兩個功能值得使用，並將其納入您的構建過程中，以確保您的構建和部署是安全的。它可以掃描源代碼文件夾和Git存儲庫以及Docker Images，並且可以掃描不同類型的問題。

現在嘗試

使用Trivy掃描本書第16章的GitHub存儲庫，以尋找應用程序庫中的漏洞。

```
＃運行漏洞掃描：
docker run -v trivy -cache：/root/.cache -it diamol/trivy：2e存儲庫-Scanners vuln https：//github.com/diamol2e/ch16.git

＃運行相同的掃描但忽略低嚴重性問題：Docker Run -v Trivy -Cache：/root/.cache -it diamol/trivy：2e reposority -scanners vuln -severity -severity -high，critical https：//github.com/github.com/diamol2e/diamol2e/ch16.git2e/ch16.git
```

您的結果可能會比我的更糟，因為自從我進行掃描以來，就會發現新的漏洞。 Trivy不會在您自己的代碼中尋找安全問題（您需要像Sonarqube這樣的靜態分析工具），但它了解諸如Node.js中的package.json之類的依賴項列表的格式或C＃.csproj文件中的軟件包列表。這些庫將在您的應用程序構建中添加為二進製文件，而這是Trivy檢查漏洞的二進製文件。 該應用程序的輸出表明我們在上線之前有一些硬工作要做。您可以在圖17.7中看到我有一個問題可以查看node.js組件，而我的Java組件中有84個高或關鍵的漏洞。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Trivy scans source code for dependency files - lists of libraries which it then checks for vulnerabilities.

This repo has Go, Node.js and Java components. Trivy scans the dependencies for each of them.

```
PS>docker run -v trivy-cache:/root/.cache -it diamol/trivy:2e repository `
>> --scanners vuln https://github.com/diamol2e/ch16.git
2025-03-06T10:59:50Z    INFO    [vuln] Vulnerability scanning is enabled
2025-03-06T10:59:51Z    INFO    Number of language-specific files       num=3
2025-03-06T10:59:51Z    INFO    [gomod] Detecting vulnerabilities...
2025-03-06T10:59:51Z    INFO    [npm] Detecting vulnerabilities...
2025-03-06T10:59:51Z    INFO    [pom] Detecting vulnerabilities...
Report Summary
```

Target	Type	Vulnerabilities
src/access-log/src/package-lock.json	npm	2
src/image-gallery/go.mod	gomod	0
src/image-of-the-day/pom.xml	pom	111

...

```
PS>docker run -v trivy-cache:/root/.cache -it diamol/trivy:2e repository `
>> --scanners vuln --severity HIGH,CRITICAL https://github.com/diamol2e/ch16.git
2025-03-06T11:00:47Z    INFO    [vuln] Vulnerability scanning is enabled
```

...

```
src/access-log/src/package-lock.json (npm)

Total: 1 (HIGH: 1, CRITICAL: 0)
```

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version
find-my-way	CVE-2024-45813	HIGH	fixed	7.7.0	8.2.2, 9.0.1

```
src/image-of-the-day/pom.xml (pom)

Total: 84 (HIGH: 66, CRITICAL: 18)
```

Filtering out the lower severity issues, there is one vulnerability to look into for the Node.js component - but we need to block some time to investigate the Java issues.

圖17.7

這是添加到CI/CD管道中的好東西。您可以向Trivy提供出口代碼，以便如果發現問題，它將失敗。 在第16章的示例管道中，我們可以添加安全掃描作為構建階段的一部分，因此，如果它們具有關鍵的漏洞，我們甚至不會將圖像推向登台存儲庫。 或者，如果您有安全專家，則可以運行一個單獨的掃描階段，因此，只要代碼有效，管道就不會失敗，但是安全漏洞會給另一個團隊提供通知。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

目標是定期運行圖像和存儲庫測試，並構建嚴重性並忽略過濾器，以便Trivy報告零故障。 這意味著您的基本圖像中的所有二進製文件以及應用程序參考中的所有庫都是乾淨的 - 或者它們具有無法通過應用程序運行方式利用的已知問題 - 並且在發現新的利用時，您的構建只會失敗。 從Trivy獲得的最後一件事是掃描應用程序的部署模型。 您可以在Dockerfiles中尋找不良的配置，Kubernetes表現出來，頭盔圖表和雲基礎架構AS-CODE（IAC）文件。

現在嘗試

使用Trivy再次掃描第16章GitHub存儲庫，這次尋找了配置錯誤的部署工件。

```
＃運行錯誤的配置掃描，忽略低嚴重性問題：docker run -v trivy -cache：/root/.cache -it diamol/trivy：2e
存儲庫-Scanners misconfig-severity -severity -severity高，關鍵
```

此掃描的結果應該更容易接受，它們突出了有關應用程序安全性硬化的重要一點。可以輕鬆地檢查Docker映像以創建軟件材料清單（SBOM），該材料清單列出了文件系統中的所有二進製文件，然後很容易地搜索這些二進製文件的利用。 Dockerfiles和Kubernetes表現出是公共規格，同樣，有一組評估它們的規則很容易。但是，當工具找到安全問題時，您需要在處理方式方面務實。 錯覺掃描將發現您需要了解的一些問題 - 我的圖17.8-但您不一定要修復它們。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

This is a new type of scan so Trivy starts by downloading another database of checks. Then it looks for IaC files - it finds six in the chapter 16 repo.

```
PS>docker run -v trivy-cache:/root/.cache -it diamol/trivy:2e repository `
>> --scanners misconfig --severity HIGH,CRITICAL `
>> https://github.com/diamol2e/ch16.git
2025-03-06T11:33:28Z    INFO    [misconfig] Misconfiguration scanning is enabled
2025-03-06T11:33:28Z    INFO    [misconfig] Need to update the built-in checks
2025-03-06T11:33:28Z    INFO    [misconfig] Downloading the built-in checks...
2025-03-06T11:33:33Z    INFO    Detected config files   num=6
```

Report Summary

Target	Type	Misconfigurations
helm/iotd/templates/api.yaml	kubernetes	2
helm/iotd/templates/log.yaml	kubernetes	2
helm/iotd/templates/web.yaml	kubernetes	2
src/access-log/Dockerfile	dockerfile	1
src/image-gallery/Dockerfile	dockerfile	1
src/image-of-the-day/Dockerfile	dockerfile	1

Not too bad. One issue to look at in each Dockerfile, and two in each of the Kubernetes manifests in the Helm chart.

圖17.8掃描錯誤配置的Dockerfiles和Kubernetes與Trivy表現出來。

如果您介紹了這些問題的細節，那麼您的應用程序也會看到幾個點。 DockerFiles使用*root*管理員帳戶，這意味著您不會在應用程序中遇到任何權限問題，但也意味著，如果攻擊者突破了您的容器，它們可能會在服務器上作為管理員帳戶運行。 這是一個有記錄的問題，可以在Dockerfile中使用用戶說明易於解決 - 但是有些應用期望將其運行為root，如果他們不這樣做，將會失敗。 Kubernetes也是如此 - 一個常見的建議是使容器文件系統僅閱讀，這使攻擊者很難下載工具或編寫腳本。但是有些應用需要一個可寫的文件系統，甚至沒有一個應用程序，甚至不會啟動。對於所有這些問題，您需要應用修復程序，然後運行完整的端到端測試 - 您的目標是盡可能確保部署，同時確保該應用程序實際上仍然有效。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

17.3最小化圖像層計數和層大小

最小且安全的基礎圖像，安全的應用程序庫和部署模型是使應用程序圖像優化並準備生產的先決條件。下一步實際上是要在應用程序所需的所有內容中設置圖像，而沒有其他內容，這比聽起來要深得多。許多用於安裝軟件的流程將殘留物留在後面，因為它們緩存軟件包列表或部署額外的推薦軟件包。您可以控制這些方法 - 不同操作系統的詳細信息是不同的，但是一般方法是相同的。

現在嘗試

Debian Linux使用APT（高級軟件包工具）安裝軟件。 本練習使用一個簡單的示例來說明如何刪除不必要的軟件包和清理包裝列表可提供大量節省（此練習與Windows容器無法使用）：

```
＃切換到練習文件夾：
CD CH17/練習/震驚

＃V1圖像使用標準命令安裝軟件包：Docker Image Build -T Diamol/CH1
7-SOCAT：2E-V1。

＃V2安裝相同的軟件包，但使用優化調整：Docker Image Build -T Diamol/ch17 -S
ocat：2e -V2 -f Dockerfile.v2。

＃檢查圖像大小：Docker Image LS“ Diamol/
ch17-Socat*”
```

dockerfile的兩個版本都安裝了相同的兩個工具 - *curl*和*socat* - 在同一debian slim映像上，它們都具有完全相同的功能。但是，您會發現V2圖像大約為25 MB，如我的輸出所示，圖17.9中的輸出。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The v2 image optimizes the commands used to install packages.

```
PS>cd ch17/exercises/socat
PS>
PS>docker image build -t diamol/ch17-socat:2e-v1 . --quiet
sha256:58defd58faffcbaf42d038de74d764d786adbbe365f9861cf36178952b8dae05
PS>
PS>docker image build -t diamol/ch17-socat:2e-v2 -f Dockerfile.v2 . -q
sha256:b3f2958fe9ad0e4961f387e5e82c50a481e107d018428fd1dd6a4d1db87aa450
PS>
PS>docker image ls "diamol/ch17-socat*"
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
diamol/ch17-socat   2e-v2     b3f2958fe9ad   5 seconds ago    87.8MB
diamol/ch17-socat   2e-v1     58defd58faff   20 seconds ago   112MB
```

The optimized image is 25MB smaller but functionally exactly the same.

圖17.9優化軟件安裝可將本練習中的圖像大小降低20%。

我只使用了幾個調整來安裝命令來保存。首先利用APT功能僅安裝列出的軟件包而不是任何建議。第二個是將安裝步驟組合到單個運行指令中，該指令以命令結尾，以刪除軟件包列表緩存並釋放該磁盤空間。清單17.3顯示了Dockerfiles之間的區別。

清單17.3安裝軟件包 - 錯誤的方式和優化方式

```
# Dockerfile - the naive install with APT: FROM debian:12-slim
RUN apt-get update RUN apt-get install -y curl=7.88.1-10+deb12
u8 RUN apt-get install -y socat=1.7.4.4-2 # Dockerfile.v2 - optim
izing the install steps: FROM debian:12-slim RUN apt-get update
\ && apt-get install -y -no-install-recommends \ curl = 7.88.1-10
+ deb12u8 \ socat = 1.7.4.4-2
```

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

單個運行指令中多個步驟組合的另一個優點是它會產生單個圖像層。 減少圖像層的數量並不是真正的優化。有一個最大的層計數，但是它應該足夠大 - 根據操作系統的不同，尤其是127。但是，更少的層確實會使跟踪您的文件系統更容易。將最終的RM命令放在自己的運行指令中很容易，並且可以說這使Dockerfile易於閱讀。但是您從本章中知道，從上一層刪除文件只會將文件隱藏在文件系統中，因此，如果這樣做，則不會在磁盤空間上保存。

讓我們看一下此模式的一個示例，該模式適用於所有平台。通常，您需要從Internet下載被壓縮的軟件包，然後將其擴展。在您在Dockerfile上工作時，將下載步驟放在單獨的指令中很容易，以便您可以使用緩存的下載層並加快開發迭代速度。 很好，但是一旦您的Dockerfile工作起作用，您就需要經過並整理一下，將下載擴展的步驟組合到單個指令中。

現在嘗試

機器學習數據集在這裡是一個很好的例子，因為它們是擴展到更大文件夾結構的大量下載。 在本章的練習中，有一個示例從加利福尼亞大學的Irvine（UCI）檔案中下載數據集，並從數據集中提取一個文件。

```
＃切換到練習文件夾：CD CH17/練習/ML-DATASET＃V1下載並擴展存檔，然後刪除不
必要的文件：Docker Image Build -T Diamol/ch17-ml-dataset：2e-v1。

＃V2下載存檔，但僅擴展必要的文件：Docker Image Build -T Diamol/CH17-ML數據庫
：2E-V2 -F Dockerfile.v2。

＃比較尺寸：Docker Image LS“ Diamol/ch17-ml-dat
aset*”
```

您會看到巨大的尺寸差異，這純粹是由於相同的優化技術 - 確保這些圖層沒有比它們需要更多的文件。我的結果在圖17.10中 - 圖像的數據下載中都有相同的單個文件，但一個文件接近2.5 GB，另一個僅為32 MB。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
Builds the standard Dockerfile and the v2 version which optimizes file extraction
PS>cd ch17/exercises/ml-dataset
PS>
PS>docker image build -t diamol/ch17-ml-dataset:2e-v1 . -q
sha256:342047468c41737a2cc4fedf233154748b3fb25436111d8ba90c4d537691b343
PS>
PS>docker image build -t diamol/ch17-ml-dataset:2e-v2 -f Dockerfile.v2 . -q
sha256:4e463e056bac51aa9cd854827f99c97a2e44b579c10457a052900a16341492d1
PS>
PS>docker image ls "diamol/ch17-ml-dataset*"
REPOSITORY                TAG       IMAGE ID       CREATED      SIZE
diamol/ch17-ml-dataset    2e-v2     4e463e056bac   2 days ago   31.6MB
diamol/ch17-ml-dataset    2e-v1     342047468c41   3 days ago   2.49GB
The optimized version is a whopping 2.4GB smaller!
```

圖17.10密切關注您如何使用文件可以節省大量磁盤空間。

這不是一個人為的例子。 當您在Dockerfile上迭代以將說明保持分開時，這很常見，因為這使調試更容易 - 您可以在以後的說明上進行工作，但請保留緩存的下載。 當您將多個命令壓入一個運行指令時，您將無法做到這一點，但是一旦您對構建感到滿意，就必須進行優化。 列表17.4顯示了優化的Dockerfile，該Dockerfile為數據文件生成一個單層（此處下載URL在此處縮寫，但您會在本章的源代碼中看到它）。

清單17.4一種用於下載和提取文件的優化方法

來自Diamol/base：2e

arg dataset_url = https://archive.ics.uci.edu/.../url_svmlight.tar.gz

WorkDir /數據集

運行wget -o dataset.tar.gz $ {dataset_url} && \ tar -xf dataset.tar.gz url _svmlight/day1.svm && \ rm -f dataset.tar.gz

這裡最大的節省實際上不是刪除檔案。它僅提取單個文件。 V1方法擴展了整個存檔（這是磁盤空間2 GB的位置），然後刪除除所需的文件以外的所有文件。正如您在本示例中看到的*tar*所見，並且在上一個示例中看到*tar*，並且在上一張中看到*tar*，知道您的工具的行為方式以及最小化磁盤使用情況有助於您控制層的大小。

對於這種情況，還有一種替代方法，它為您提供了最佳的開發人員工作流程和優化的最終圖像，並且使用多階段的Dockerfiles使用單獨的階段來完成所有渴望磁盤的步驟。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

17.4將您的多階段構建提升到一個新的水平

您首先在第4章中看到了多階段的構建，我們使用一個階段從源代碼中編譯了一個應用程序，以及一個以後的階段來打包編譯的二進製文件以進行運行時。 除最簡單的圖像外，多階段Dockerfiles應該是所有人的最佳實踐，因為它們使優化最終圖像變得更加容易。

我們可以重新訪問數據集下載器，並為每個步驟使用單獨的階段。清單17.5表明我們以這種方式獲得了更可讀的Dockerfile（下載URL再次被縮寫）。

列表17.5多階段Dockerfiles援助可讀性並簡化優化

```
來自Diamol/base：2e作為基礎

從基礎下載arg dataset_url = https://archive.ics.uci.uci.edu/ml/.../url/url/url_svmlight.tar.gz

運行wget -o dataset.tar.gz $ {dataset_url}

從基礎擴展
複製-from =下載dataset.tar.gz。
運行tar xvzf dataset.tar.gz

從基地
workdir /dataset /url_svmlight
複製-from =展開url_svmlight/day1.svm。
```

在每個階段，您的工作都很明顯，您無需深入研究磁盤空間的異常命令優化，因為最終圖像只會從早期階段明確複製文件。當您構建V3時，您會發現它的大小與優化的V2版本相同，但是它具有易於調試的優勢。多階段的Dockerfiles可以建立到特定階段，因此，如果您需要檢查構建的文件系統零件，則可以輕鬆地進行操作。

現在嘗試

目標參數可讓您在特定階段停止多階段構建。 嘗試使用不同目標構建該V3圖像。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

＃切換到練習文件夾：
CD CH17/練習/ml-dataset

＃構建完整的V3圖像：
Docker Image Build -T Diamol/CH17-ML數據庫：2E-V3 -F Dockerfile.V3。

＃構建到“下載”目標 - 同一dockerfile，不同的標籤：docker映像構建-t diamol/ch17-ml-dataset：2e-v3-download -f dockerfile.v3-目標下載。

＃並構建到“展開”目標：Docker Image Build -T Diamol/ch17-ml數據庫：2e-v3-Expand -f dockerfile.v3-目標擴展。

＃檢查圖像尺寸：Docker Image LS“ Diamol/ch17-ml-datase
t：2e-v3*”

現在，您將有三種V3圖像的變體。完整的構建與優化的構建相同，因此我們沒有通過移動到多階段的Dockerfile而失去任何優化。其他變體在特定階段停止構建，並且您可以從其中一個圖像中運行一個容器，以導航文件系統，如果需要調試。舞台構建還顯示了磁盤空間的發展位置 - 您可以在圖17.11中看到下載約為260 MB，並且擴展到近2.5 GB。

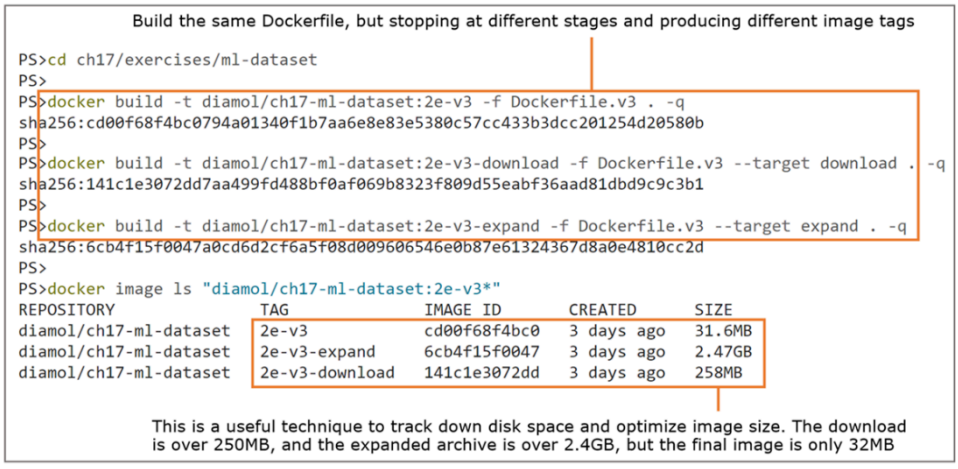


圖17.11在特定階段構建多階段Dockerfiles，可讓您調試內容和檢查尺寸。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這確實是最好的方法 - 您獲得了優化的圖像，但是您可以將Dockerfile指令保持簡單，因為您不需要在中間階段清理磁盤。

多階段構建的最終優勢確實使它帶回家：每個階段都有自己的構建緩存。 如果您需要調整構建並為擴展階段編輯DockerFile，則在運行構建時，下載階段仍將來自緩存。最大化構建緩存是優化的最後一部分，這全都與構建圖像的速度有關。

充分利用構建緩存的基本方法是訂購Dockerfile中的指示，以便最少變化的事物是開始時的，而最常變化的事物朝著最後。這可能需要一些迭代才能正確，因為您需要了解步驟更改的頻率，但是通常可以在文件開頭放置靜態設置，例如曝光端口，環境變量和應用程序入口點。改變最大的事情是您的應用程序二進製文件和配置文件，它們可以走到最後。正確理解這一點，您可以大大減少構建時間。

現在嘗試

此練習建立了最小的詹金斯安裝。它是不完整的，所以不要試圖運行它 - 我們只是在使用它，因為它可以進行良好的演示。 Dockerfile下載Jenkins Java文件並設置初始配置。 V2 Dockerfile充分利用了緩存，當您更改內容時，您會看到它。

```
＃切換到Jenkins文件夾：CD CH17/練
習/Jenkins

＃構建V1圖像和優化的V2圖像：Docker Build -T Diamol/ch17-Jenkins：2E-V
1。 Docker Build -T Diamol/ch17 -Jenkins：2e -v2 -f dockerfile.v2。

＃現在更改配置文件兩個dockerfiles都使用：echo 2.0 > jenkins.i
nstall.upgradewizard.state

＃重複構建，看看它們運行了多長時間：docker build -t diamol/ch17-jenkins
：2e-v1。 Docker Build -T Diamol/ch17 -Jenkins：2e -v2 -f dockerfile.v2。
```

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

第二輪構建是緩存的位置。 V1Dockerfile在下載Jenkins文件之前將配置文件複製到圖像中（90 MB下載），因此，當更改配置文件時，將損失緩存並再次發生下載。 V2 Dockerfile使用多階段構建，並訂購指令將配置文件副本最後放置。我使用powershell中的Mouse-Command函數進行了練習，以檢查每個構建的持續時間（在Linux中有一個等效的時間）。您可以在圖17.12中看到正確訂購指令並使用多階段的Dockerfile將構建時間從近10秒縮短到僅一秒鐘。

The first builds with no cached layers 8-9 seconds. The difference between v1 and v2 is just network variation; `Measure-Command` is a PowerShell timing mechanism like `time` in Linux shells

```
PS>cd ch17/exercises/jenkins
PS>
PS>Measure-Command {docker build -t diamol/ch17-jenkins:2e-v1 .} | Select TotalSeconds

TotalSeconds
------------
        8.41

PS>Measure-Command {docker build -t diamol/ch17-jenkins:2e-v2 -f Dockerfile.v2 .} | Select Tota
lSeconds

TotalSeconds
------------
        8.74

PS>echo 2.0 > jenkins.install.UpgradeWizard.state
PS>
PS>Measure-Command {docker build -t diamol/ch17-jenkins:2e-v1 .} | Select TotalSeconds

TotalSeconds
------------
        9.20

PS>Measure-Command {docker build -t diamol/ch17-jenkins:2e-v2 -f Dockerfile.v2 .} | Select Tota
lSeconds

TotalSeconds
------------
        1.06
```

Changing the input file busts the cache in the v1 Dockerfile and the download happens all over again. v2 is correctly structured so the cache isn't broken and only the changed layer gets executed

圖17.12正確訂購Dockerfile指令可能意味著在構建時間中節省大量。

充分利用緩存可以使您可以從每個更改中構建和推動Docker映像，從而無需在CI/CD管道中吸收時間。 不過，您確實需要確保不會過度調查的內容，因為如果您使用運行說明安裝或下載軟件，則它們將被緩存，直到dockerfile的說明更改（假設在該說明之前沒有停滯）。 將軟件包添加到圖像中時，應始終使用顯式版本，以便您確切地知道自己正在運行的內容，並且可以選擇何時更新。 *apt*命令中使用的*socat*示例17.3使用了明確的版本號，jenkins示例使用了用於下載的版本的ARG指令 - 兩種方法都可以使用緩存，直到更改要安裝的版本為止。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

層優化有一個最終方面，這與我們到目前為止所涵蓋的相反。通常，您想將圖像優化為少數層，但有時候這是一種過度優化的，會損害性能。在某些情況下，您可以在其中構建一個存儲大量數據的圖像 - 我的客戶端構建了他們在具有千兆字節的端到端測試中使用的數據庫圖像。如果您有一個非常大的圖像，那麼當Docker推動並拉動圖像時，您將不會受益於並行性。 當您提取圖像時，Docker Engine同時下載多層（默認值為三層，但您可以在引擎配置中更改此圖像）。當您將所有數據放在一個6 GB層中時，就不會得到並行性，如果您將數據文件劃分為許多層，它的性能會更好。

清單17.6顯示了ML數據集下載的變體。在這種情況下，我們希望在最終圖像中的第70-79天的所有數據文件。為了進行比較，我們有一個稱為“一層”的階段，我們將文件複製到單層中，而替代階段稱為多層，我們將每個文件複製到其自己的層中。

清單17.6複製一層或多層的大文件

```
＃其他階段 - 基礎，下載和展開 - 如前所述

從基礎作為單層工作dir/dataset/url_svmlight copy-from
=展開url_svmlight/day7？ .svm。



從基礎作為多層workdir/dataset/url_svmlight copy-from
=展開url_svmlight/day70.svm。複製-from =展開url_svm
light/day71.svm。 ...複製-from =展開url_svmlight/day78.
svm。複製-from =展開URL_SVMLIGHT/Day79.SVM
。
```

多層方法更脆弱，因為您必須明確列出要復制的每個文件，而不是使用更靈活的文件模式。 但這是一個不太常見的情況，當您試圖加快傳輸時間以獲得大圖像時，這絕對值得嘗試。

現在嘗試

建造 ML數據集圖像的兩個目標並比較圖像 層。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
＃切換到ML文件夾：
CD CH17/練習/ml-dataset

＃構建單層和多層圖像：Docker Build -T Diamol/ch17-ml-dataset：v4-one-layer -f dockerfile.v4-鍵入一層
。



Docker Build -T Diamol/CH17-ML數據庫：V4-Multi-Layer -f Dockerfile.v4  - 靶標多層。



＃比較圖像尺寸：Docker Image LS“ Diamol/ch17-ml-da
taset：v4*”

＃並檢查圖層：Docker歷史Diamol/CH17-ML數據庫：V4-One層D
ocker歷史記錄Diamol/CH17-ML-DATASET：V4-MULTI-LAYER
```

您將看到如何在運行此操作時如何更好地優化多層圖像。圖17.13顯示，它具有許多小層，當將圖像拉到新機器上時（就像在自動縮放容器平台中一樣）時，特別有益。在我客戶的用例中，這種方法將下載時間從幾分鐘減少到秒。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

We build from the same Dockerfile but target different stages.

The images are the same size.

```
PS>cd ch17/exercises/ml-dataset
PS>
PS>docker build -t diamol/ch17-ml-dataset:v4-one-layer -f Dockerfile.v4 --target one-la
yer . -q
sha256:b158f48b90d06eadfbf2551033c0f5fffa6a29af7df1a426c3376c7d46cf4364
PS>
PS>docker build -t diamol/ch17-ml-dataset:v4-multi-layer -f Dockerfile.v4 --target mult
i-layer . -q
sha256:a0013c0424e935c10b4dff2eb94a5579902b182c6cf0157455b5fd3a7f1e4ed5
PS>
PS>docker image ls "diamol/ch17-ml-dataset:v4*"
REPOSITORY                TAG               IMAGE ID       CREATED          SIZE
diamol/ch17-ml-dataset    v4-multi-layer    a0013c0424e9   8 seconds ago    198MB
diamol/ch17-ml-dataset    v4-one-layer      b158f48b90d0   22 seconds ago   198MB
PS>
PS>docker history diamol/ch17-ml-dataset:v4-one-layer
IMAGE          CREATED          CREATED BY                                      SIZE
   COMMENT
b158f48b90d0   30 seconds ago   COPY url_svmlight/Day7?.svm . # buildkit        184MB
                  ...
PS>
PS>docker history diamol/ch17-ml-dataset:v4-multi-layer
IMAGE          CREATED              CREATED BY                                  SIZE
COMMENT
a0013c0424e9   About a minute ago   COPY url_svmlight/Day79.svm . # buildkit    18.2MB
<missing>      About a minute ago   COPY url_svmlight/Day78.svm . # buildkit    18.2MB
<missing>      About a minute ago   COPY url_svmlight/Day78.svm . # buildkit    18.2MB
<missing>      About a minute ago   COPY url_svmlight/Day77.svm . # buildkit    18.1MB
                  ...
```

But one image has a single 180MB layer with all the data files in, and the other has ten 18MB layers - which is better for optimizing push and pull performance.

圖17.13在許多層上拆分數據文件可以使性能提高。

最後一個示例證明了優化是至關重要的，但不受固定規則的約束。 您需要知道在更改圖像構建方式時會影響哪些參數，以及如何改變時間，轉移時間和安全性，以便您可以根據自己的要求量身定制方法。

17.5了解為什麼優化計數

您在本章中看到，您可以遵循一些簡單的最佳實踐，並使您的Dockerfiles和Container Images可以與您一起工作。這些做法歸結為

- 選擇正確的基本圖像 - 理想地策劃自己的一組黃金圖像。 除最簡單的應用程序以外，使用多階段Dockerfiles。 不要添加任何不必要的軟件包或文件 - 層尺寸的焦點。 通過更改頻率對Dockerfile指令進行排序 - 最大程度地將緩存化。
- 監視您的構建，推動和拉動時間並迭代以保持優化。
-
-

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

隨著您將更多的應用程序移至容器，構建，推動和拉圖像成為組織工作流程的核心部分。優化這些圖像可以消除大量疼痛點，加快工作流程並防止更嚴重的問題。 圖17.14顯示了圖像的典型生命週期以及優化計數的區域。

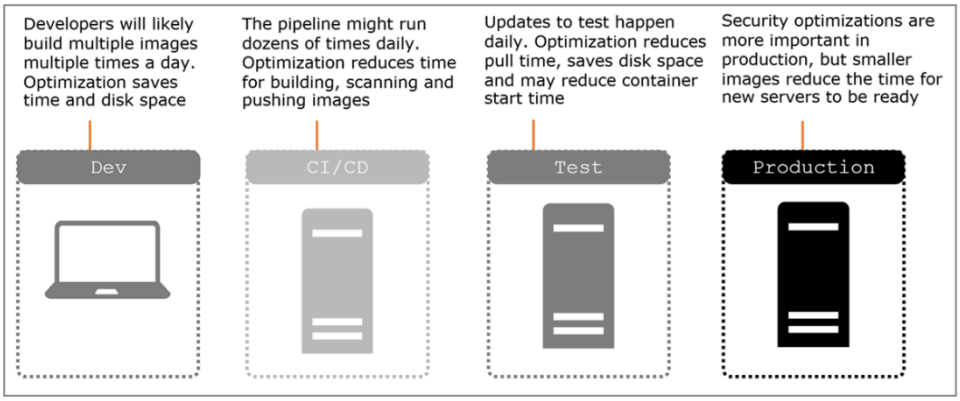


圖17.14優化您的Docker圖像在項目的整個生命週期中都會產生有益的影響。

17.6實驗室

現在是時候將您的優化技巧進行測試了。您的目標是優化安裝Docker命令行的圖像。本章實驗室文件夾中有Linux和Windows示例。 Dockerfiles現在正在工作，但它們產生了不必要的大圖像。您的目標是

- 優化文件系統，以使Linux容器的圖像低於50 MB，或者用於Windows容器的330 MB以下。 利用圖像層緩存，以使圖像的重複構建少於一秒鐘。 製作一個圖像，可以
- 從Docker Run -IT < Image > Docker版本正確地寫入Docker CLI版本（該命令將為您提供服務器的錯誤，因為它未連接到Docker Engine，但是CLI版本應該正確打印）。
-

您不需要任何提示，但是當您查看原始Dockerfiles時，需要創造性地思考。 您可能不會通過優化現有說明來到達那裡； 從圖像的目標中向後工作可能會更好。

我的優化文件位於同一實驗室文件夾中 - 您還可以在github：https：// github上檢查它們。 com/sixeyed/diamol/blob/2e/ch17/lab/readme.md。

您有知識 - 現在進行優化！

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

18 在容器中的應用程序配置管理

應用程序需要從正在運行的環境中加載其配置，這通常是環境變量和磁盤讀取的文件的組合。 Docker為在容器中運行的應用程序創建了該環境，它可以設置環境變量並從許多不同的來源構建文件系統。 這些作品都可以幫助您為應用程序構建靈活的配置方法，因此，當您部署到生產時，您使用的是通過所有測試階段的相同圖像，但您會為實時環境注入所需的設置。您只需要做一些工作即可將這些部件整合在一起，將應用程序設置為從多個位置合併配置值。

本章將使用.NET，Java，Go和Node.js中的示例來介紹推薦的方法（以及一些替代方法）。 這裡的某些工作生活在開發人員領域，引入了庫來提供配置管理，其餘的生活在依靠通信的開發區和OP之間的灰色區域中，以便雙方都知道配置模型的工作原理。

18.1 應用程序配置的多層方法 離子

您的配置模型應反映您要存儲的數據的結構，這通常是三種類型之一：

- 發布級別的設置，對於給定發行版的環境級設置，每個環境都相同，對於每個環境（測試，產品等）功能級設置都不同，該設置可用於更改版本之間的行為
-
-

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

其中一些是相當靜態的，有些是動態的，具有一組已知的變量，而另一些則具有動態性，並具有未知的變量集。 圖18.1顯示了一些示例配置設置以及可以從環境中讀取的何處。

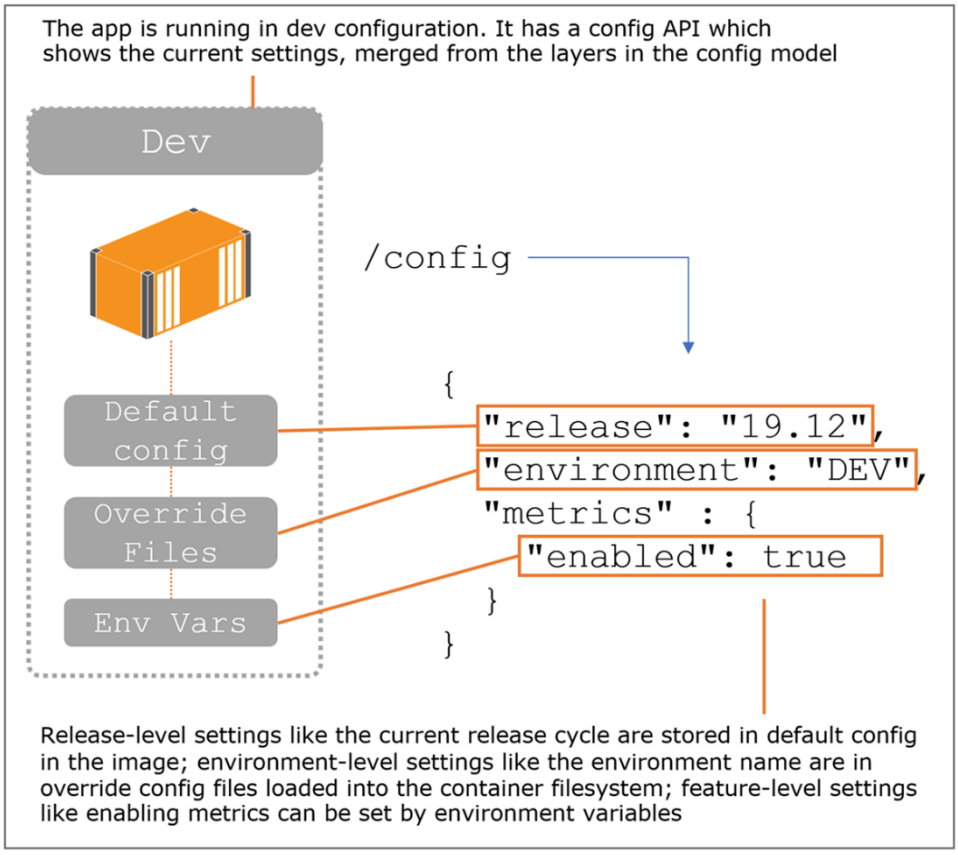


圖18.1具有來自圖像，文件系統和環境變量的設置的配置層次結構

我們將使用的第一個示例是Node.js，帶有流行的配置管理庫，稱為*node-config*。該庫可讓您從層次結構中的多個文件位置讀取配置設置，並使用環境變量覆蓋它們。本章練習中的Access-Log示例應用程序使用Node-Config庫，並設置了兩個目錄以讀取配置文件：

- 配置 - 將在Docker Image中使用默認設置包裝。 config-override-圖像中不存在，但
- 可以通過卷，配置對像或秘密從容器文件系統中進行配置。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

使用圖像的默認配置運行示例應用程序，然後使用用於開發環境的覆蓋文件的同一圖像。

```
＃切換到練習文件夾：CD CH18/練習/訪
問儀

＃在圖像中運行一個帶有默認配置的容器：Docker Run -D -P 8080：8
0 Diamol/ch18 -Access -Log：2e

＃運行一個容器加載本地配置文件覆蓋：Docker Run -D -P 8081：80 -V“ $（PWD）/config/config/dev
：/app/app/config -override” diamol/ch18- access -log：2e


＃檢查每個容器中的配置API：curl http：// localhost
：8080/config curl http：// localhost：8081/config
```

第一個容器僅使用圖像中包裝的默認配置文件，該文件指定了發行週期的名稱（19.12），並設置了啟用Prometheus指標。環境名稱有一個未知的設置 - 如果您看到的話，您知道環境級配置設置尚未正確應用。 第二個容器將本地配置目錄加載為預期位置中的捲，以供應用程序找到覆蓋物，以設置環境名稱並翻轉“指標”功能。您會看到何時調用來自同一圖像的容器已應用不同的設置的配置API時，Mine在圖18.2中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The first container runs using the default config file built into the image

The second container loads a config override file from a local directory as a bind mount

```
PS> cd ch18/exercises/access-log
PS>
PS> docker run -d -p 8080:80 diamol/ch18-access-log:2e
e051f2110e15b7528fda83d9b6e1cf58ca44b4a8a70eddf9c0a66cfcae8e971a
PS> docker run -d -p 8081:80 -v "$(pwd)/config/dev:/app/config-override" `
>> diamol/ch18-access-log:2e
1d0b06c1f8c441222727fcea8dc716243fab5f3143c1fd7dcb4516297927274b
PS>
PS> curl http://localhost:8080/config
{"release":"19.12","environment":"UNKNOWN","metricsEnabled":true}
PS>
PS> curl http://localhost:8081/config
{"release":"19.12","environment":"DEV","metricsEnabled":false}
```

The app has a config API. In the second container the settings for environment and metrics have come from the override file - the release name is from the default config

圖18.2使用卷，配置對像或秘密合併配置文件很簡單。

加載配置從應用程序代碼中的已知路徑覆蓋，使您可以將它們從任何源提供到容器中。 我使用的是本地綁定安裝座，但是源可以是在容器平台中存儲的配置對像或秘密（Kubernetes具有ConfigMap和Secret Resources，如我們在第15章中所見），並且行為是相同的。這種模式有一個細微差別 - 您的配置目標可以是特定的文件路徑或目錄。 目錄目標更靈活（Windows容器不支持單個文件中的加載量），但是源文件名需要匹配應用程序期望的配置文件名。在此示例中，綁定源是目錄config/dev，它具有一個文件 - 容器See/app/config-override/local.json，它是它尋求覆蓋的地方。

Node-Config軟件包還可以從環境變量加載設置，它們覆蓋了從文件層次結構加載的任何設置。 這是“十二個因子應用程序”（https：//12factor。net）中推薦的配置方法 - 一種流行的應用程序體系結構風格，環境變量總是比其他配置源優先。這是一種有用的方法，可以幫助您進入心態，即容器是短暫的，因為更改環境變量以設置應用程序配置意味著更換容器。 Node-Config具有略有不尋常的實現：如果要將單個設置指定為環境變量，則需要將JSON文件映射環境變量名稱變量到設置（訪問-LOG應用程序中的圖像中內置此內置）。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

在開發模式下運行第三個版本的訪問日誌容器，但啟用了指標。使用音量安裝座加載開發配置和環境變量以覆蓋指標設置。

```
＃運行一個帶有覆蓋文件和環境變量的容器：docker容器運行-d -p 8082：80 -v“ $（pwd）/config/dev：/app/app/config -override” -e app_metrics_enabled = true diamol/diamol/ch18 -access -log：2e

＃檢查配置：curl http：// localhost：8082/
config
```

第三個容器從圖像中的默認文件，卷中的本地配置覆蓋文件和特定的環境變量設置合併。 這是構建配置的一個很好的例子，可以使開發人員的工作流程順利運行。開發人員可以在不啟用指標的情況下運行默認設置（這將節省CPU週期和內存），但是當它們需要打開指標以進行某些調試時，他們可以使用相同的圖像和環境變量開關來完成。圖18.3顯示了我的輸出。

This container loads config from the image, from the override file in the bind mount directory, and from the environment variable

```
PS> docker container run -d -p 8082:80 `
>>  -v "$(pwd)/config/dev:/app/config-override" `
>>  -e APP_METRICS_ENABLED=true `
>>  diamol/ch18-access-log:2e
09c323ed860330a9910f227fe17ea98d0f5340c50908872cf8e0896e96347771
PS>
PS> curl http://localhost:8082/config
{"release":"19.12","environment":"DEV","metricsEnabled":"true"}
```

The release cycle comes from the default file, the environment name from the override file and the metrics setting from the environment variable

圖18.3與環境變量合併配置使更改特定功能變得容易。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這是配置的核心模式，您應該考慮將其應用於所有應用程序。從此示例中，您可以看到該模式很清楚，但是細節很重要，這是灰色區域可以在交付和部署之間分解知識的灰色區域。 Access-Log應用程序可讓您使用新的默認配置文件覆蓋默認配置文件，但是該目標文件必須位於特定位置。 您還可以覆蓋所有具有環境變量的文件設置，但是只有帶有正確名稱的環境變量才能拾取。 最終，這將在您用於部署的應用程序模型中進行記錄，但是您需要意識到該模式具有錯誤的潛力。 一種替代方法消除了這種風險，以使配置管理降低靈活性。

18.2每個環境的包裝配置

許多應用程序框架都支持配置管理系統，您可以將部署中每個環境的所有配置文件捆綁在一起，並且在運行時您設置一個值以指定您正在運行的環境的名稱。該應用程序平台將配置文件加上匹配的環境名稱，並且應用程序已完全配置。 .NET使用其默認配置提供商設置來執行此操作，其中配置設置是從這些來源合併的：

- AppSettings.json - 所有環境的默認值AppSettings。 {Environment} .json-命名環境
- *Environment variables*的覆蓋 - 用於指定環境名稱，並設置覆蓋
-

本章有一個新版本的待辦事項列表應用程序，它使用了包裝Docker Image中所有配置文件的方法。您使用特定的環境變量來提供當前的環境名稱，並在其餘配置文件之前加載。

現在嘗試

使用默認配置運行待辦事項列表應用程序，該應用程序由環境名稱開發設置，然後使用測試環境設置：

```
＃使用默認配置運行待辦事項列表應用程序：docker run -d-p 808
3：8080 diamol/ch18-todo-list：2e


＃使用用於測試環境的配置運行應用程序：Docker Run -d -P 8084：8080 -E dotnet_environment =測
試diamol/ch18 -todo -list：2e
```

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這兩個容器從同一圖像運行，但加載不同的配置文件。 圖像內部是用於開發，測試和生產環境的環境文件。 第一個容器將Core AppSettings.json與AppSettings.development.json合併，它在開發模式下運行，因為開發被設置為DockerFile中的默認環境。 第二個將AppSettings.json與AppSettings.test.json合併。兩個環境配置文件已經存在於Docker映像中，因此無需安裝新配置的外部源。瀏覽到http：// localhost：8083/Diagnostics查看Dev Config和http：// localhost：8084/Diagnostics查看測試版本。我的輸出在圖18.4中。

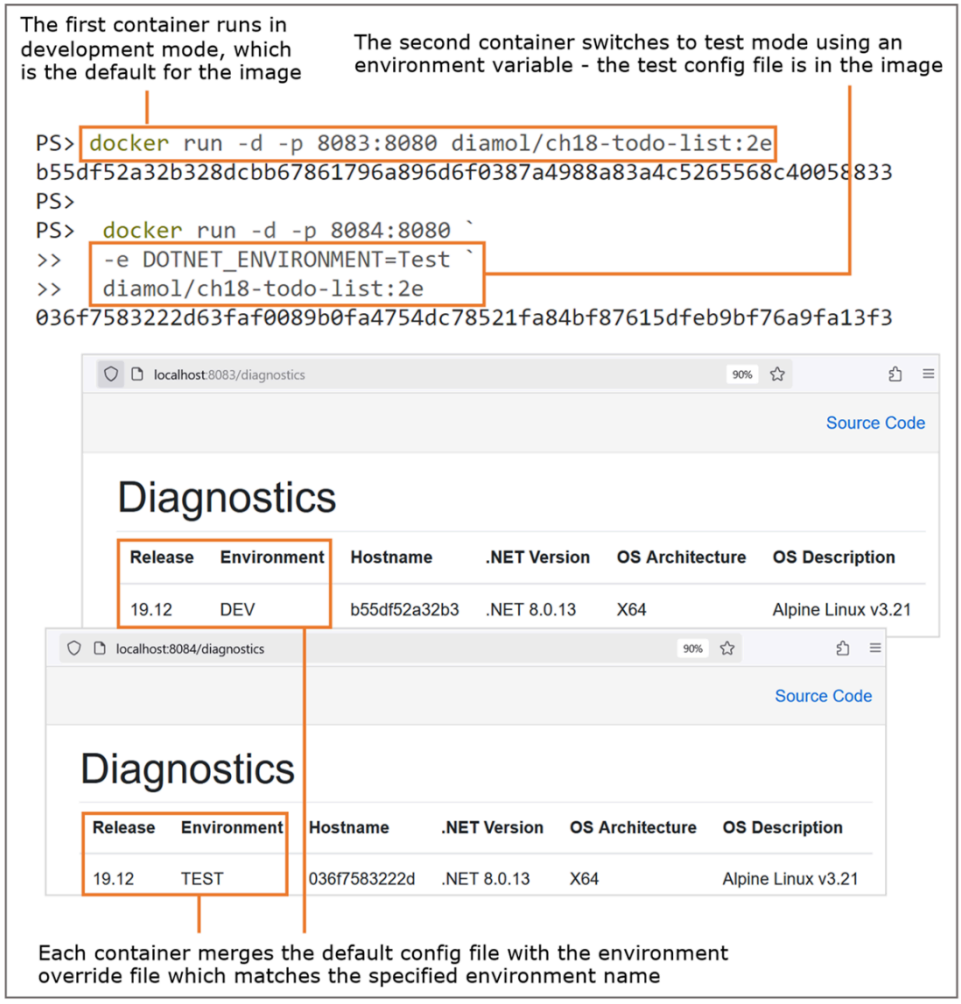


圖18.4圖像中的每個環境的配置文件都可以輕鬆切換環境。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

如果您有單獨的系統來管理配置文件和源代碼，則此方法可以很好地工作。 CI/CD管道可以作為構建的一部分將配置文件帶入Docker映像中，因此您將配置管理與開發保持分開。缺點是您仍然無法打包每個設置，因為您需要將機密信息放在Docker映像之外。 您需要採用深入的安全方法，並假設您的註冊表可能會受到損害 - 在這種情況下，您不希望某人在圖像中不錯的普通文本文件中找到所有密碼和API鍵。

如果您喜歡這種方法，則仍然需要允許覆蓋文件，並且要使用環境變量進行最終覆蓋。待辦事項列表應用程序執行此操作，從一個名為“ config-overrides”的文件夾中加載文件，並使用標準.NET使用加載環境變量的標準方法。 如果您想複製問題，則可以在本地運行生產環境，但可以覆蓋環境設置以使用數據庫文件而不是遠程數據庫服務器。

現在嘗試

即使所有環境配置都捆綁在應用程序中，待辦事項列表應用程序也支持配置覆蓋層。 如果您以生產模式運行，則該應用程序會失敗，因為它希望找到數據庫服務器，但是您可以使用覆蓋文件在生產中運行以使用數據庫文件。

```
CD CH18/練習/待辦事項

Docker Run -D -P 8085：80 -e dotnet_environment =生產-v“ $（pwd）/config/config/prod-本地：/app/app
/config -override” diamol/ch18 -todo -list：2e
```

您可以瀏覽到http：// localhost：8085/Diagnostics，並查看該應用程序在生產模式下運行，但是配置文件覆蓋更改數據庫設置，以便該應用程序仍然可以在不運行Postgres容器的情況下工作。我的輸出在圖18.5中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

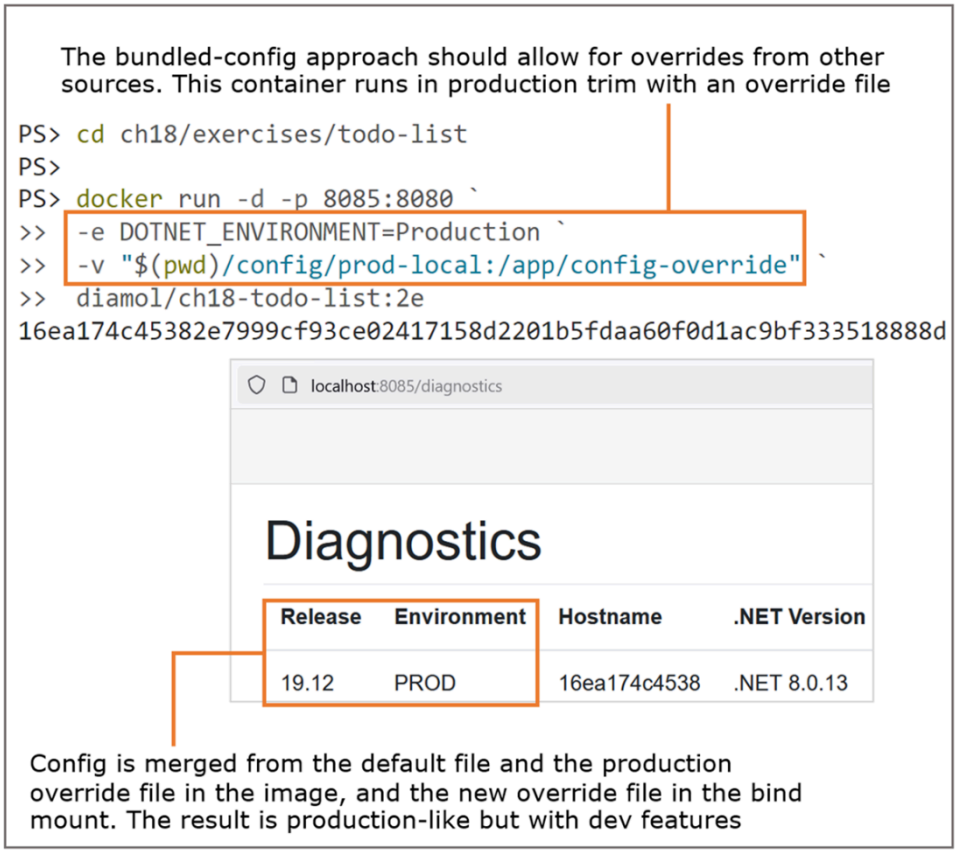


圖18.5選擇要運行的環境仍應支持其他文件中的配置覆蓋。

該容器將默認的appSettings.json文件與圖像中的環境文件appsettings.json合併在一起，以及prod-loc-local operocal文件夾中的override file.json。該設置類似於node.js示例，因此文件夾和文件名稱具有一定的一致性，但是.NET採用不同的方法來通過環境變量設置覆蓋。在Node-Config中，您可以使用JSON文件映射環境變量進行配置設置，因此您可以使用任何喜歡的環境變量名稱。您可以通過將JSON文件中的屬性名稱作為環境變量名稱指定為.NET中的環境變量。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

運行相同的本地版本生產版本，但使用具有環境變量的設置覆蓋該設置，並使用自定義發行名稱。

```
#運行一個帶有綁定安裝和自定義環境變量的容器：docker Run -d -p 8086：8080 -e dotnet_environment
=生產-e版本-e版本=自定義-V“ $（pwd）/config/prod/prod/prod -local：/app/app/propporch -over -overrid
e” diamol/ch18 -todo -list：
```

瀏覽到http：// localhost：8086/Diagnostics，您會從環境變量中看到自定義發行名稱。我的輸出在圖18.6中。

圖18.6配置層次結構覆蓋了帶有環境變量的任何配置文件的值。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

我不得不說我不喜歡這種包裝多個配置文件的方式，儘管這是許多應用程序平台上的常見方法。 您的圖像中會有一些配置設置的危險，您認為這不敏感，但您的安全團隊可能會不同意。服務器名稱，URL，文件路徑，日誌記錄級別甚至緩存尺寸都可能對試圖破解系統的任何人都是有用的信息。 當您將所有機密設置移動到從容器平台上應用的文件時，無論如何，這些包裝環境文件中可能幾乎沒有什麼。 我也不喜歡拆分，在源控制中管理某些設置，而另一些設置則在配置管理系統中。

容器的美麗是，您可以遵循自己喜歡的模式，所以不要讓我為您決定。 根據您的組織和技術堆棧，有些方法可以更好地工作。如果您有多個堆棧可以處理，事情也會變得更加複雜 - 您會在下一個使用GO應用程序的示例中看到這一點。

18.3從運行時加載配置

Go具有一個流行的配置模塊，稱為Viper，它提供了與.NET庫或Node-Config相同的功能。您將模塊添加到軟件包列表中，在應用程序代碼中，您指定了配置目錄的路徑，以及是否希望帶來環境變量來覆蓋配置文件。我使用與其他示例類似的層次結構將其添加到本章的Image Gallery應用程序中：

- 文件是從config目錄中加載的，該目錄首先是在Docker映像中填充的。 特定於環境的文件是從config-override目錄中加載的，該目錄在圖像中是空的，可以是容器文件系統安裝座的目標。 環境變量覆蓋文件設置。
-
-

與其他示例相比，Viper支持配置文件的一組更寬的語言。您可以使用JSON或YAML，但是Go World中流行的格式是Toml（以其創作者Tom Preston-Werner的名字命名）。 TOML對配置文件很有幫助，因為它可以輕鬆地映射到代碼中的字典，並且比JSON或YAML更容易閱讀。清單18.1顯示了圖庫應用程序的TOML配置。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

清單18.1 TOML格式使易於管理的配置文件

```
釋放=“ 19.12”環境=“未知” [UNKENICS]
啟用= true [apis] [apis.image] url =“ http：/// i
otd/image” http：//iotd/image“ [apis.access] u
rl {apis.access] url = url =”
```

您會看到TOML在許多雲原生項目中都使用，因為它比替代方案要容易得多。如果您選擇格式，TOML值得考慮，因為“易於閱讀”也意味著易於調試，並且易於查看合併工具中版本之間的差異。除文件格式外，此示例以與node.js應用程序相同的方式工作，默認的config.toml文件包裝到docker映像中。

現在嘗試

在沒有任何其他配置設置的情況下運行GO應用程序來檢查默認設置。

```
＃運行容器：
Docker Run -D -P 8087：80 Diamol/ch18 -Image -Gallery：2E

＃檢查配置API：
curl http：// localhost：8087/config
```

運行此練習時，您會看到當前的應用程序配置，這些配置全部來自默認的TOML文件。我的輸出是在圖18.7中（我已經通過*jq*工具對響應進行了管道，以使其更易於閱讀），並顯示了該應用程序消耗的API的發行週期和默認URL。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

The container image is packaged with a single config file, which has default values but doesn't represent an actual environment

```
PS> docker run -d -p 8087:80 diamol/ch18-image-gallery:2e
d40e9da4bd92330b709c1ecb77fff54e5e4e66f25e87d0acd69ecc39fe045ad7
PS>
PS> curl -s http://localhost:8087/config | jq
{
  "Release": "19.12",
  "Environment": "UNKNOWN",
  "Metrics": {
    "Enabled": true
  },
  "Apis": {
    "access": {
      "Url": "http://accesslog/access-log"
    },
    "image": {
      "Url": "http://iotd/image"
    }
  }
}
```

These are release-level settings which apply across all environments, but the "unknown" environment value tells you this app isn't fully configured

圖18.7你　　可以用可行的默認設置包裝您的應用程序，但不是comp　　萊特環境。

輸出來自為當前配置設置返回JSON的配置API。當您擁有多個配置源時，配置API是應用程序中非常有用的功能。它使調試配置問題變得更加容易，但是您需要保護該數據。如果任何嘗試瀏覽 /配置的人都可以公開閱讀機密設置，則沒有意義，因此，如果您要添加配置API，則需要執行三件事：

- 不要只是發布整個配置；選擇性，永遠不要包括秘密。　保護端點，以便只有授權用戶才能訪問它。　使配置API成為可以通過Config啟用的功能。
-
-

圖片庫GO組件與層次配置模型採用略有不同的方法 - 默認設置保存在圖像中，但沒有用於任何特定的環境。期望每個環境都會指定其自己的附加配置文件，該文件擴展或覆蓋默認文件中的設置以設置完整的環境。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

再次使用覆蓋文件運行同一應用程序以構建一個完整的環境。

```
＃切換到應用程序目錄：CD CH18/練
習/圖像 - 套裝

＃在配置目錄中運行帶有綁定的容器：docker容器運行-D -P 8088：80 -V“ $（pwd）/config/config/d
ev：/app/app/config -override” diamol/ch18 -image -gallery：2e

＃再次檢查配置：curl http：// localhost：
8088/config
```

我在圖18.8中的輸出顯示了現在為開發環境完全配置的應用程序，將圖像中的發行級配置文件與來自容器平台（我的本地docker機器）的環境覆蓋文件合併。

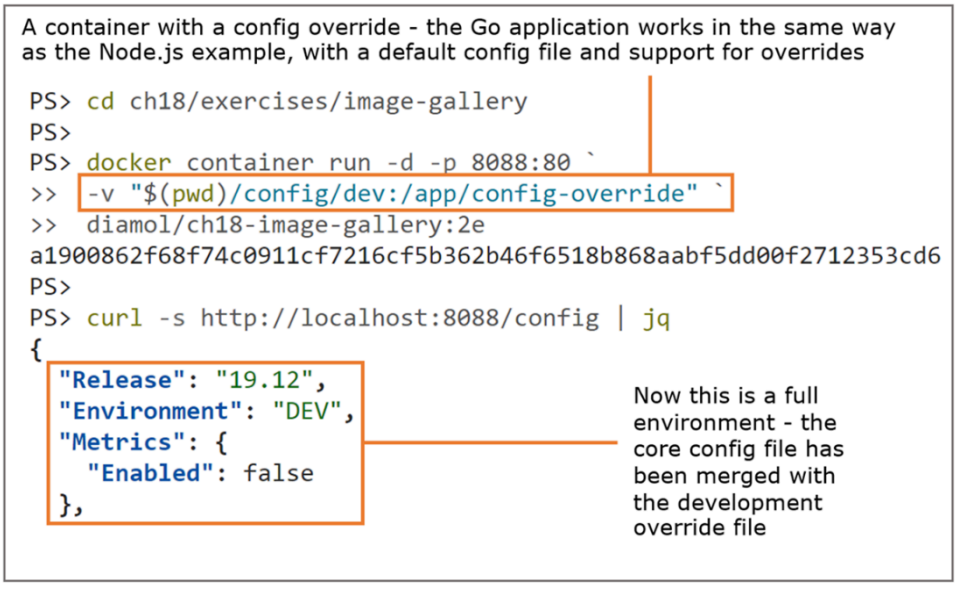


圖18.8 GO Viper模塊以與Node-Config軟件包相同的方式合併了配置文件。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

向您展示配置主題的所有這些輕微變化不僅是填寫本章的便宜方式。當組織採用Docker時，他們傾向於發現使用量會很快加速，並且很快就會在容器中運行許多應用程序，每個應用程序都對配置有自己的意見。 由於應用程序平台提供的功能和期望的約定不同，因此類似這樣的小型變體一定會發生。您可以在高級別上應用標準 - 映像必須用默認配置包裝，並且必須支持文件和環境變量覆蓋，但是配置文件和環境變量格式的詳細信息將很難標準化。

我們將在GO應用程序的最後一個示例中看到這一點。 Viper模塊支持環境變量以覆蓋配置文件中的設置，但與node-config和.net又不同。

現在嘗試

運行具有環境變量覆蓋的容器。 此應用中的配置模型僅使用帶有IG前綴的環境變量。

```
＃確保您在應用程序目錄中：CD CH18/練習/圖像
- 套裝

＃使用配置替代和環境變量運行容器：docker run -d -p 8089：80 -v“ $（pwd）/config/dev
：/app/app/config -override” -e ig_metrics.enabled = true diamol/ch18 -image/ch18 -image -gall
ery：2e：2e：2e：2e

＃檢查配置：
curl http：// localhost：8089/config
```

Viper的慣例是您應該前綴環境變量名稱，以免它們與其他環境變量衝突。在此應用程序中，前綴為IG，其次是下劃線，然後是DOT符號中的配置設置名稱（因此IG_Metrics.Enabled匹配TOML文件中的指標組中的啟用值）。您可以從圖18.9中的輸出中看到此設置在默認設置之上添加了開發環境，但隨後覆蓋了指標設置以啟用Prometheus指標。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

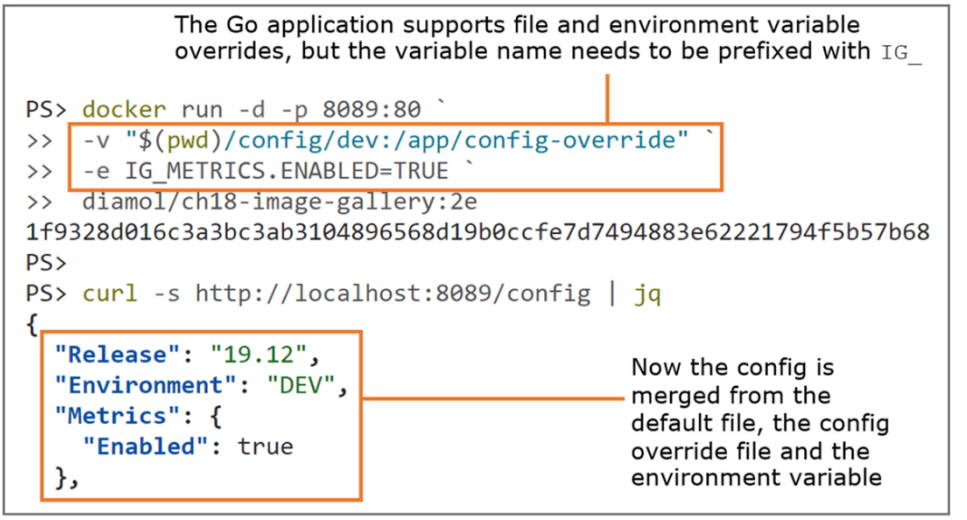


圖18.9所有示例應用程序都支持配置的環境變量，但變化很小。

我們已經使用三個不同的應用程序瀏覽了配置建模，並且我們採用了三種略有不同的方法。 這些差異在應用程序清單文件中易於管理且易於記錄，並且實際上不會影響您構建圖像或運行容器的方式。 我們將查看本章中的最後一個示例，該示例採用相同的配置模型，並將其應用於沒有一個不錯的新配置庫的應用程序，因此它需要一些額外的工作才能使其像現代應用一樣行為。

18.4以與新應用相同的方式配置舊應用程序

傳統應用程序對配置有自己的想法，這些想法通常不涉及環境變量或文件合併。 .NET Framework應用程序是一個很好的例子 - 他們期望在特定位置進行XML配置文件。 他們不喜歡在應用程序根文件夾之外尋找文件，也不喜歡查看環境變量。重新啟動這些應用程序時，您仍然可以採用相同的配置方法，但是您需要在Dockerfile中做一些額外的工作。

這裡的方法是包裝一個實用程序應用程序或一組腳本，該應用程序將容器環境中的配置設置轉換為應用程序期望的配置模型。確切的實現將取決於您的應用程序框架以及如何使用配置文件，但是邏輯將是這樣的：

1。從容器中指定的源文件中讀取配置覆蓋設置。 2。在環境變量中讀取覆蓋物。 3。合併兩組覆蓋，以使環境變量優先。 4。將合併的覆蓋物寫入容器中指定的目標文件。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

在本章的練習中，使用此方法的日常Java API的圖像的更新版本。它實際上不是一個舊應用程序，但是我已經構建了舊版模式的圖像，好像該應用程序無法使用普通的容器配置選項。有一個實用程序應用程序在啟動時運行並設置配置，因此，儘管內部配置機制不同，但用戶可以以與其他示例相同的方式配置容器。

現在嘗試

使用默認配置設置和文件覆蓋的“傳統”應用程序。

```
＃切換到應用程序目錄：CD CH18/練習/當
天圖像


＃運行帶有默認配置的容器：Docker容器運行-D -P 8090：80 Diamol/ch18-Image-tay
-tay：2E


＃在bind Mount中使用Config Override文件運行：Docker容器運行-D -P 8091：80 -V“ $（PWD）/config/
dev：/config-override” -e config_source_source_path =




＃檢查配置設置：curl http：// localhost：
8090/config curl http：// localhost：8091/co
nfig
```

用戶體驗與其他應用程序非常相似 - 用環境覆蓋文件（並且源可以是配置對像或秘密）的捲 - 但是您必須在環境變量中另外指定覆蓋文件位置，以便啟動實用程序知道在哪裡查看。您會在運行練習中看到圖像中默認配置指定發行週期的練習，而不是環境 - 與第二個容器中的覆蓋文件合併。我的輸出在圖18.10中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The image has a settings file, so this container runs with the default config

This container loads an override file in a bind mount, and the environment variable specifies the location of the override file

```
PS> cd ch18/exercises/image-of-the-day
PS>
PS> docker run -d -p 8090:80 diamol/ch18-image-of-the-day:2e
d119309091b12443361b6ed621eed364df0b49c1ced7b7c44156b6a126771b6b
PS>
PS> docker  run -d -p 8091:80 `
>>  -v "$(pwd)/config/dev:/config-override" `
>>  -e CONFIG_SOURCE_PATH="/config-override/application.properties" `
>>  diamol/ch18-image-of-the-day:2e
a7a1ec174462f737ee3cb3765245231b2eb5a00cc5d0bdba3c7e11181e6b91fe
PS>
PS> curl -s http://localhost:8090/config | jq
{
  "release": "19.12",
  "environment": "UNKNOWN",
  "managementEndpoints": "health,info,prometheus",
  "apodUrl": "https://api.nasa.gov/planetary/apod?api_key="
}
PS>
PS> curl -s http://localhost:8091/config | jq
{
  "release": "19.12",
  "environment": "DEV",
  "managementEndpoints": "health",
  "apodUrl": "https://api.nasa.gov/planetary/apod?api_key="
}
```

The default config has a release cycle but no environment name

The container with the config override file has the environment name merged in, and doesn't expose Prometheus metrics

圖18.10　　應用程序具有啟動配置模型的實用程序，但是用戶體驗　　E是一樣的。

魔術在一個簡單的Java實用程序中發生，該程序將與應用程序的其餘部分相同的多級版本進行編譯和包裝。清單18.2顯示了構建實用程序並將其設置為在啟動時運行的Dockerfile的關鍵部分。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Listing 18.2 Building and using a config load utility in the Dockerfile

```
FROM diamol/maven:2e AS builder
# ...
RUN mvn package

# config util
FROM diamol/maven:2e as utility-builder
WORKDIR /usr/src/utilities
COPY ./src/utilities/ConfigLoader.java .
RUN javac ConfigLoader.java

# app
FROM diamol/openjdk:2e
ENV CONFIG_SOURCE_PATH="" \
    CONFIG_TARGET_PATH="/app/config/application.properties"

CMD java ConfigLoader && \
    java -jar /app/iotd-service-0.1.0.jar

WORKDIR /app
COPY --from=utility-builder /usr/src/utilities/ConfigLoader.class .
COPY --from=builder /usr/src/iotd/target/iotd-service-0.1.0.jar .
```

這裡重要的要點是，您可以擴展Docker映像，以使舊應用的行為與新應用程序相同。您可以控制啟動邏輯，因此您可以在運行實際應用程序之前執行所需的任何設置步驟。 當您執行此操作時，您將增加容器啟動和應用程序準備就緒之間的時間，並且您還會增加容器可能失敗的風險（如果啟動邏輯有錯誤）。您應該始終在圖像或應用程序模型中進行健康檢查，以減輕這種狀態。

我的配置加載程序實用程序應用程序支持具有環境變量覆蓋其他設置的12個因子方法。它將環境變量與Override Config File合併，並將輸出作為配置文件寫入該應用程序期望的位置中的配置文件。該實用程序採用與Viper相同的方法，尋找具有特定前綴的環境變量，該變量有助於使應用程序設置與容器中的其他設置分開。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

傳統應用程序不使用環境變量，但是配置實用程序將它們設置為將它們設置為它們，從而使用戶體驗與現代應用程序相同。

```
＃運行一個帶有替代文件和環境變量的容器：Docker運行-D -P 8092：80 -V“ $（pwd
）/config/dev：/config-override” -e config_source_path



＃檢查配置設置：curl http：// localhost：
8092/config
```

該實用程序使我以與其他應用程序相同的方式使用舊應用程序。 它主要對用戶透明 - 它們只是設置環境變量並將覆蓋文件加載到卷中。 它與剛剛讀取其期望看到的配置文件的應用程序是透明的 - 此處的原始應用程序沒有更改。圖18.11顯示，此“遺產”應用程序使用現代的多層配置方法。

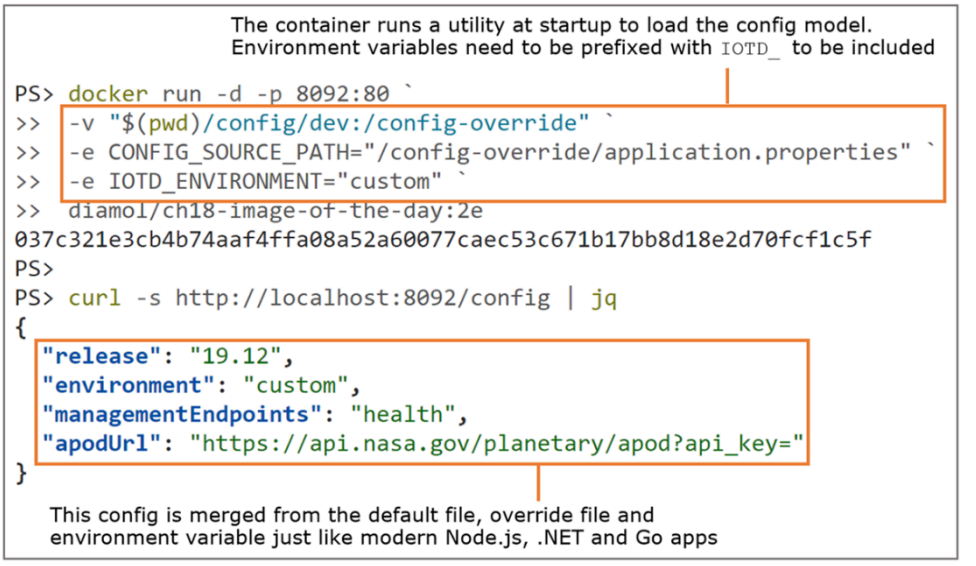


圖18.11環境變量使此舊應用程序的配置模型像新應用一樣行為。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在，圖庫應用程序中的每個組件都使用相同的配置模式。所有組件的標準化都有一定程度的標準化，但是實施差異也很小。每個組件都可以使用文件覆蓋以在開發模式下運行，並且每個組件都可以使用環境變量配置以啟用Prometheus指標。每個應用程序的實際做法都不同，這是我一開始就提到的灰色區域 - 如果環境變量enable enable_metrics = true true true true true noveruce enable enable_metrics =，則很難說出每個組件都會運行Prometheus端點，因為應用程序平台以不同的方式工作。

文檔是您消除這種混亂的方式，在Docker World中，最好在應用程序模型中完成部署文檔。本章的練習中有一個Docker撰寫文件，可以完全執行我在上一段中所規定的內容 - 將每個組件設置為開發模式並啟用Prometheus指標。 清單18.3顯示了撰寫文件的配置部分。

列表18.3記錄docker中的配置設置

```
services:
  accesslog:
    image: diamol/ch18-access-log:2e
    environment:
      APP_METRICS_ENABLED: "true"
    secrets:
      - source: access-log-config
        target: /app/config-override/local.json

  iotd:
    image: diamol/ch18-image-of-the-day:2e
    environment:
      CONFIG_SOURCE_PATH: "/config-override/application.properties"
      IOTD_MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE: "health,prometheus"
    secrets:
      - source: iotd-config
        target: /config-override/application.properties

  image-gallery:
    image: diamol/ch18-image-gallery:2e
    environment:
      IG_METRICS.ENABLED: "TRUE"
    secrets:
      - source: image-gallery-config
        target: /app/config-override/config.toml

secrets:
  access-log-config:
```

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

文件：access-log/config/dev/local.json
iotd-config：
文件：日期/config/dev/application.properties
圖像 - 加利爾 - 構菲格：
文件：image-gallery/config/dev/config.toml

這是一個冗長的代碼列表，但我想將其全部保存在一個地方，以便您可以看到模式是相同的，儘管細節是不同的。 Node.js應用程序設置了一個環境變量以啟用指標，並且將JSON文件加載為配置替代。 Java應用程序使用一個環境變量，該變量列出了要包含的管理端點； 在其中添加Prometheus啟用指標集合。 然後，它從屬性文件中加載config替代，該文件是一系列鍵/值對。 GO應用程序使用環境變量啟用指標並將配置替代加載為TOML文件。我正在使用Docker中的秘密支持用於文件源，但是對於Kubernetes中的ConfigMaps或Secret的模式是相同的。

這裡的用戶體驗既好又壞。這很好，因為您可以通過更改配置替代的源路徑來輕鬆加載不同的環境，並且可以使用環境變量更改單個設置。這很糟糕，因為您需要知道應用程序的怪癖。 項目團隊可能會進化各種Docker構成替代以涵蓋不同的配置，因此編輯配置設置將不是一個常見的活動。運行該應用程序將更加普遍，這就像使用Compose啟動任何應用程序一樣容易。

現在嘗試

讓我們整體運行該應用程序，並為所有組件進行固定的配置集。首先清除容器，然後使用Docker Compose運行應用程序。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
＃刪除所有容器：Docker RM -F $（Docker容器L
S -AQ）＃切換到練習文件夾：CD CH18/練習




＃使用配置設置運行應用程序：Docker組成-D


＃檢查所有配置API：curl http：// localhos
t：8030/config curl http：// localhost：8020
/config curl http：// localhost：8010/config
```

您可以瀏覽到http：// localhost：8010，並以正常方式使用該應用，然後瀏覽到Prometheus端點，以查看組件指標（在http：// localHost：8010/Metrics，http：// http：// http：// localhost：8030/localhost：8030/metrics and httpp：// localhtp：// localhhhometator：8020：// proserator：//020：2020年/200.20：//////////020。但是實際上，正確配置該應用程序的所有確認都來自這些配置API。

您可以在圖18.12中看到我的輸出。每個組件從圖像中的默認配置文件，Config Override文件中的環境名稱以及來自環境變量的指標設置加載了釋放週期名稱。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The compose file specifies the dev environment for every component, with a feature override in an environment variable to enable metrics

```
PS> docker rm -f $(docker container ls -aq) | Out-Null
PS>
PS> cd ch18/exercises
PS>
PS> docker compose up -d
[+] Running 4/4
 ✓ Network exercises_iotd-net           Created                 0.0s
 ✓ Container exercises-iotd-1           Started                 0.5s
 ✓ Container exercises-accesslog-1      Started                 0.5s
 ✓ Container exercises-image-gallery-1  Sta...                  0.8s
PS>
PS> curl -s http://localhost:8030/config | jq '.environment, .metricsEnabled'
"DEV"
"true"
PS>
PS> curl -s http://localhost:8020/config | jq '.environment, .managementEndpoints'
"DEV"
"health,prometheus"
PS>
PS> curl -s http://localhost:8010/config | jq '.Environment, .Metrics.Enabled'
"DEV"
true
```

Each config API shows the settings for the component. I've filtered the output to show just the environment name and the metrics feature settings for each.

圖18.12 Docker Compose可以記錄應用程序配置設置並使用該配置啟動應用程序。

這就是我們真正需要介紹的有關構建應用程序以從容器環境中獲取配置的模式的圖案。我們將對本章進行一些思考，以了解多層配置模型可以將您帶到哪裡。

18.5了解為什麼靈活的配置模型可以回報

我在第11章和第16章中使用Docker介紹了CI/CD管道，該管道的核心設計是您構建一個圖像，而您的部署過程是關於通過生產之前的環境來促進該圖像。 您的應用程序需要在每個環境中的工作方式略有不同，並且在保持單圖像方法的同時支持該應用程序是使用多層配置模型。

在實踐中，您將使用容器映像中內置的釋放級設置，幾乎在所有情況下，容器平台提供了環境級覆蓋文件，但是可以使用環境變量設置特徵級配置的功能是有用的添加。 這意味著您可以快速對生產問題做出反應 - 如果這是性能問題，則可以降低日誌記錄的水平，或者關閉具有安全孔的功能。這也意味著您可以在開發機上創建類似於生產的環境，以使用刪除秘密並使用環境變量的“生產配置覆蓋”複製錯誤。

在任何環境中運行完全相同圖像的能力是在配置模型中投資時間的回報。 圖18.13顯示了從CI/CD管道開始的圖像的生命週期。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

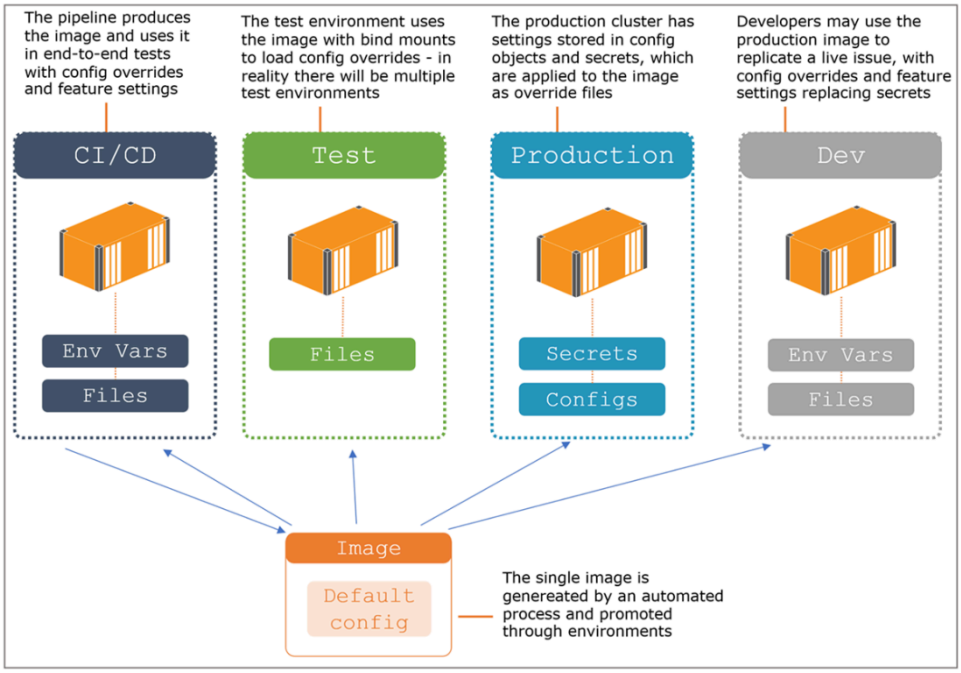


圖18.13 CI/CD管道會產生一個圖像，然後使用配置模型來改變行為。

您在生產這種靈活配置模型方面所做的工作將對未來的應用程序有很大幫助。 所有容器平台都支持將文件從配置對像或秘密以及設置環境變量中加載到容器中。 本章圖像庫應用程序的Docker映像將與Docker Compose，Azure容器實例或Kubernetes以相同的方式工作。 而且不僅是容器平台 - 標準配置文件和環境變量是平台即服務（PAAS）產品和無服務器功能中使用的模型。

18.6實驗室

挖掘新應用程序的配置模型並弄清楚如何設置覆蓋文件和配置功能替代可能會很棘手，因此您將在本實驗室中獲得一些練習。您將使用同一圖像庫應用程序 - 在本章的實驗室文件夾中，有一個Docker組成的文件，其中指定了應用程序組件，但沒有配置。您的工作是將每個組件設置為

- 使用卷加載配置覆蓋文件。 為測試環境加載配置替代。 覆蓋發
- 行週期為“ 25.03”，而不是“ 19.12”。
-

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

這應該很簡單，但是花一些時間對應用程序進行調整而不對應用程序進行任何更改將很有用。 當您使用Docker組成的應用程序運行時，您應該能夠瀏覽到http：// localhost：8010，並且該應用程序應起作用。而且您應該能夠瀏覽所有三個配置API，並看到發行名稱為25.01，並且環境是測試。

我的解決方案位於docker-compose-solution.yml文件中的同一文件夾中，或者您可以在此處在github上進行檢查：https：// github。 com/sixeyed/diamol/blob/2e/ch18/lab/readme.md。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

19 用docker 編寫和管理應用程序日誌

伐木通常是學習新技術的最無聊的部分，但對Docker並非如此。基本原理很簡單：您需要確保將應用程序日誌寫入標準輸出流，因為那是Docker尋找它們的地方。有幾種方法可以實現我們將在本章中涵蓋的目標，然後開始樂趣。 Docker有一個可插入的記錄框架 - 您需要確保您的應用程序日誌從容器中出現，然後Docker可以將它們發送到不同的位置。這使您可以構建一個功能強大的記錄模型，其中所有容器中的應用程序日誌都將發送到中央日誌存儲，並在其頂部使用可搜索的UI（所有人都使用開源組件，都在容器中運行。

19.1歡迎來到Stderr和Stdout！

Docker映像是使用所有應用程序二進製文件和依賴項的文件系統的快照，也是一些元數據告訴Docker，當您從圖像運行容器時，可以從哪個過程開始。該過程在前景中運行，因此就像啟動外殼會話然後運行命令一樣。只要命令處於活動狀態，它就可以控制終端輸入和輸出。 命令寫入標準輸出和標準錯誤流的日誌條目（稱為*stdout*和*stderr*），在終端會話中，這意味著您在窗口中看到了輸出。在容器中，Docker Engine觀察Stdout和Stderr，並從流中收集輸出，這是容器日誌的來源。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

如果運行一個簡單的應用程序，可以在循環中打印時間，可以很容易地看到這一點。 該應用程序本身在前景中運行，並將日誌條目寫入Stdout。

```
#在前景中運行容器：Docker Run -IT Diamol/ch1
9 -Timecheck：2e


#完成後，用CTRL-C/CMD-C退出容器
```

您會在終端中看到一些日誌線，並且會發現您無法輸入更多命令 - 容器在前景中運行，因此就像在終端中運行應用程序本身一樣。該應用程序每隔幾秒鐘就會為STDOUT編寫另一個時間戳，因此您會在會話窗口中看到另一行。我的輸出在圖19.1中。

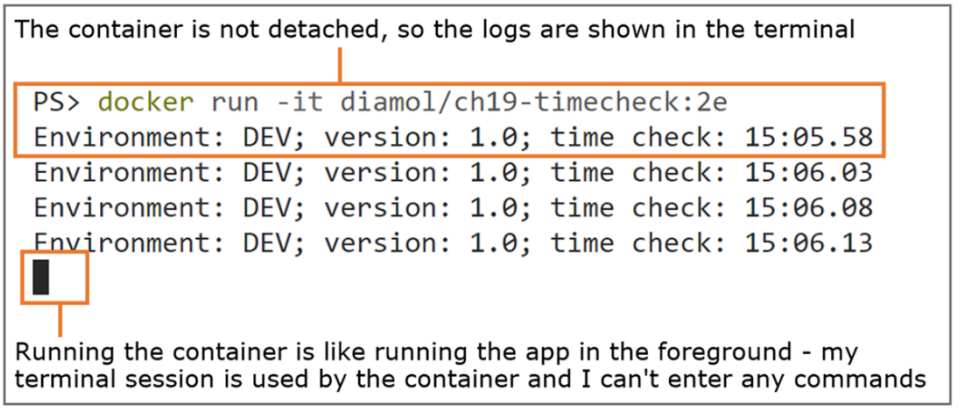


圖19.1前景中的一個容器接管了終端會話，直到它退出為止。

這是容器的標準操作模型 - Docker在容器內部啟動一個過程，並從該過程中收集輸出流。我們在本書中使用的幾乎所有應用程序都遵循相同的模式：應用程序過程在前景中運行（可能是二進製或Java運行時），並且該應用程序本身被配置為編寫日誌以以相同的方式來stdout（或docker; docker; docker; docker tore conter s toter conters corters corters toters corters toters tots thements"。這些日誌由應用程序運行時寫入輸出流，並收集它們。 圖19.2顯示了應用程序，輸出流和Docker之間的相互作用。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

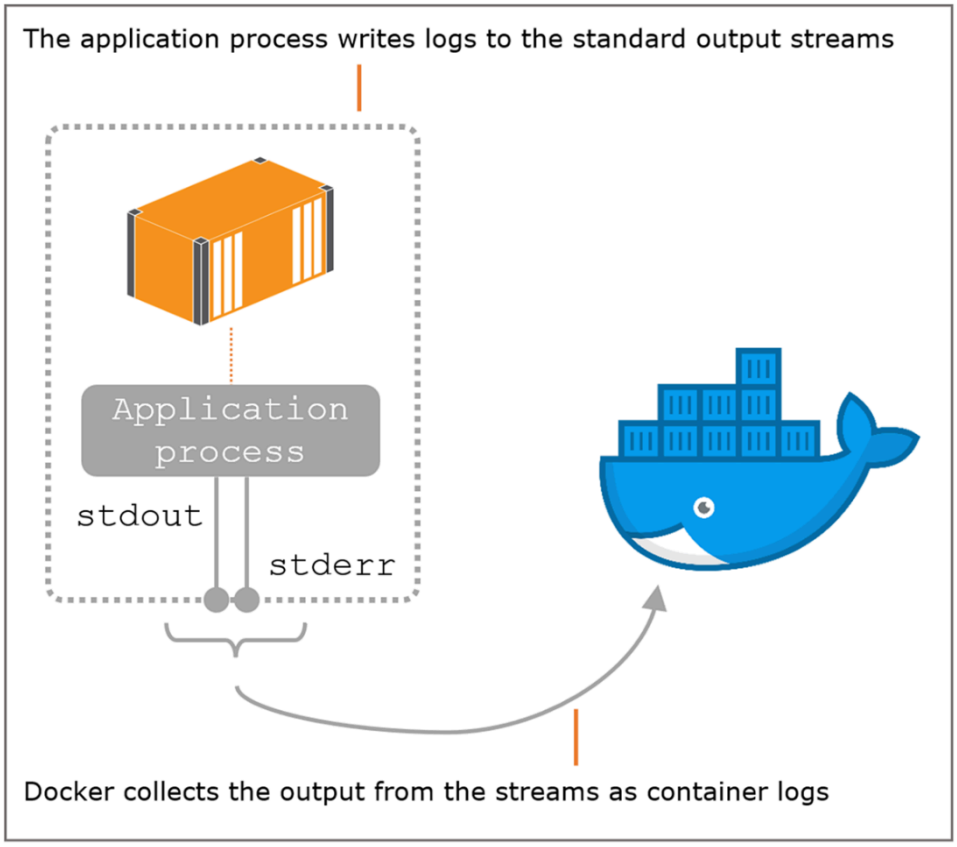


圖19.2 Docker在容器中觀看申請過程並收集其輸出流。

容器日誌由Docker Engine存儲為JSON文件，因此日誌條目仍然可用於沒有終端會話的獨立容器，以及已退出的容器，因此沒有申請流程。 Docker為您管理JSON文件，並且它們具有與容器相同的生命週期 - 當卸下容器時，也將刪除日誌文件。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

從背景中的同一圖像與獨立容器一起運行一個容器，然後檢查日誌，然後檢查日誌文件的路徑。

```
＃運行一個獨立的容器：Docker Run -D -Name Timecheck Diamol/ch19
-Timecheck：2e

＃檢查最新的日誌條目：Docker Logs -Tail
1 TimeCheck

＃停止容器並再次檢查日誌：Docker停止Timecheck Docke
r Logs -Tail 1 TimeCheck

＃檢查Docker在哪裡存儲容器日誌文件：Docker Container Inspert -Format
='{{。 logPath}}'TimeCheck
```

如果您使用帶有Linux容器的Docker桌面，請記住，Docker引擎在Docker為您管理的VM內運行 - 您可以看到容器的日誌文件的路徑，但是您無法訪問VM，因此您無法直接讀取文件。 如果您在Linux上運行Docker CE或使用Windows容器，則將在本地計算機上進行日誌文件的路徑，您可以打開文件以查看原始內容。您可以在圖19.3中看到我的輸出（使用Windows子系統在Windows子系統中運行的Docker引擎）。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

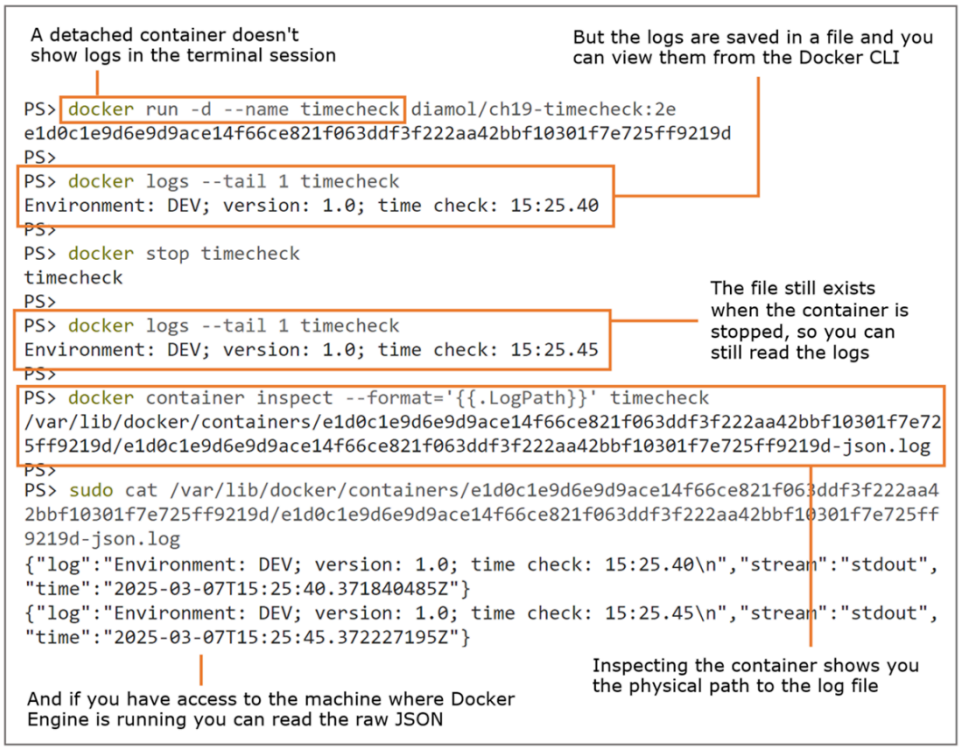


圖19.3 Docker將容器日誌存儲在JSON文件中，並管理該文件的壽命。

Docker只要存在容器即將停止，就可以保留日誌文件，但在卸下容器時會刪除文件。實際文件實際上是您通常不必擔心的實現細節。格式非常簡單；它包含每個日誌條目的JSON對象，其中包含日誌的字符串，日誌來自（stdout或stderr）的流的名稱以及一個時間戳。清單19.1顯示了我的TimeCheck容器的日誌示例。

清單19.1容器日誌的原始格式是一個簡單的JSON對象

```
{“log”：“環境：dev;版本：1.0;時間檢查：15：25.40\n”，“stream”：“stdout”
，“time”：“2025-03-07T15：25：40.371840485Z”}


{“log”：“環境：dev;版本：1.0;時間檢查：15：25.45\n”，“stream”：“stdout”
，“time”：“2025-03-07T15：25：45.3722271955
```

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

您唯一需要考慮JSON的時間是，如果您有一個產生大量日誌的容器，並且希望將所有日誌條目保留一段時間，但是將它們保持在可管理的文件結構中。 Docker默認情況下為每個容器創建一個JSON日誌文件，並將其生長到任何尺寸（直到填充磁盤）。您可以將Docker配置為使用最大尺寸限制的滾動文件，以便在日誌文件填充時，Docker開始寫入新文件。您還配置了要使用多少日誌文件，並且當它們全部完整時，Docker開始覆蓋第一個文件。 您可以在Docker發動機級別設置這些選項，以便更改適用於每個容器，也可以為單個容器設置它們。為特定容器配置日誌記錄選項是為一個應用程序獲取小的，旋轉的日誌文件但保留其他容器的所有日誌的好方法。

現在嘗試

再次運行同一應用程序，但是這次指定日誌選項，使用三個滾動日誌文件，每個文件最多為5 kb。

```
＃運行日誌選項和應用程序設置來編寫大量日誌：docker run -d-name timeCheck2  -  log-opt max-siz
e = 5k -log-opt max-file = 3 -e Timer_Eintervalseconds =


＃在容器中驗證日誌設置：docker容器Inspert -format ='{{json .hostconfig.logconfig}}}'timecheck2


＃等待幾分鐘

＃檢查日誌：
Docker容器檢查-Format ='{{。 logPath}}'timechec                                K2
```

您會看到容器的日誌路徑仍然只是一個JSON文件，但是Docker實際上是使用該名稱作為基礎旋轉日誌文件，但具有存檔日誌文件編號的後綴。如果您在Linux上運行Windows Server容器或Docker CE，則可以列出保留日誌的目錄的內容，並看到這些文件後綴。地雷如圖19.4所示。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The log options configure Docker to use at most three log files, rolling onto the next one when the current file hits 5KB in size

```
PS> docker run -d --name timecheck2 --log-opt max-size=5k --log-opt max-file=3 -e Timer__IntervalSeconds=1 diamol/ch1
9-timecheck:2e
c3f2667d9fbdb091f2984f739685556b7d4b5ff9255eb95163b772f9072862e3
PS>
PS> docker container inspect --format='{{json .HostConfig.LogConfig}}' timecheck2
{"Type":"json-file","Config":{"max-file":"3","max-size":"5k"}}
PS>
PS> docker container inspect --format='{{.LogPath}}' timecheck2
/var/lib/docker/containers/c3f2667d9fbdb091f2984f739685556b7d4b5ff9255eb95163b772f9072862e3/c3f2667d9fbdb091f2984f739
685556b7d4b5ff9255eb95163b772f9072862e3-json.log
PS>
PS> sudo ls /var/lib/docker/containers/c3f2667d9fbdb091f2984f739685556b7d4b5ff9255eb95163b772f9072862e3/ -l
total 52
-rw-r----- 1 root root  982 Mar  8 06:31 c3f2667d9fbdb091f2984f739685556b7d4b5ff9255eb95163b772f9072862e3-json.log
-rw-r----- 1 root root 5041 Mar  8 06:31 c3f2667d9fbdb091f2984f739685556b7d4b5ff9255eb95163b772f9072862e3-json.log.1
-rw-r----- 1 root root 5037 Mar  8 06:30 c3f2667d9fbdb091f2984f739685556b7d4b5ff9255eb95163b772f9072862e3-json.log.2
```

Inspecting the container only shows one file, but this is the current log file and Docker is archiving old ones

On the Docker Engine there is a folder for each container's data - here you see the current log file and the two previous ones, archived at 5KB

圖19.4滾動日誌文件可讓您保留每個容器的已知日誌數據。

來自STDOUT的應用程序日誌有一個集合和處理階段，您可以在這裡配置Docker對日誌的操作。 在上次練習中，我們配置了日誌處理以控制JSON文件結構，並且可以使用容器日誌進行更多操作。為了充分利用這一點，您需要確保每個應用程序都從容器中推出日誌，在某些情況下需要更多的工作。

19.2從其他水槽到Stdout的接力日誌

並非每個應用程序都非常適合標準記錄模型； 當您將某些應用程序容器化時，Docker將在輸出流中看到任何日誌。 某些應用程序以Windows服務或Linux Daemons的身份運行，因此容器啟動過程實際上不是應用程序過程。其他應用程序可能會使用將記錄框架寫入日誌文件或其他位置的現有記錄框架（在Linux或Windows Event Log中的Syslog中稱為*sinks*）。 無論哪種方式，都沒有來自容器啟動過程的應用程序日誌，因此Docker看不到任何日誌。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

有一個新版本的TimeCheck應用程序將日誌寫入文件而不是STDOUT。運行此版本時，儘管應用程序日誌存儲在容器文件系統中，但沒有容器日誌。

```
#從新圖像運行一個容器：docker run -d-name timecheck3 diamol/ch19-time
check：2e-v2

#檢查 - 沒有來自stdout的日誌：docker容器日誌timecheck3

#現在在容器中運行一個命令以打印日誌文件：docker容器exec -it timecheck
3 cat /logs/timecheck.log
```

即使應用程序本身正在編寫許多日誌條目，您也會看到沒有容器日誌。我的輸出在圖19.5中 - 我需要連接到容器並從容器文件系統中讀取日誌文件以查看日誌條目。

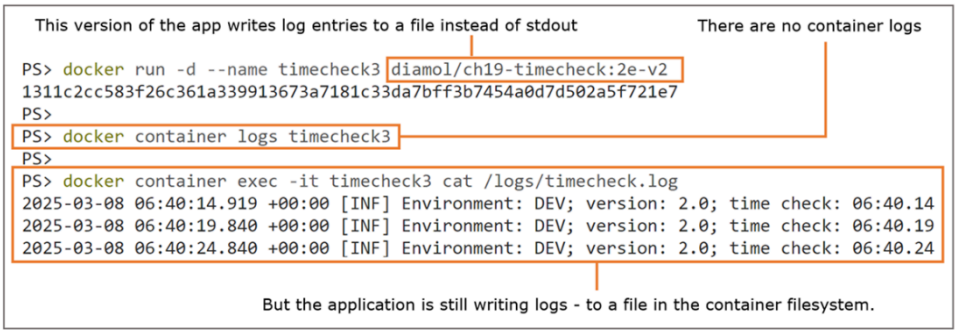


圖19.5如果該應用不在輸出流寫任何內容，則不會看到任何容器日誌。

之所以發生這種情況，是因為該應用程序使用了自己的日誌匯（本練習中的一個文件），而Docker對此水槽一無所知。 Docker只會從Stdout讀取日誌；無法將其配置為從容器內部的其他日誌接收器中讀取。當您重寫在容器中運行的應用程序時，這是一個非常普遍的問題，目標是調整容器設置以生成普通的容器日誌而無需更改代碼。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

處理這樣的應用程序的模式是在容器啟動命令中運行第二個進程，該過程讀取應用程序使用的匯處日誌條目，並將其寫入stdout。該過程可能是Shell腳本或簡單的實用程序應用程序，它是開始序列中的最終過程，因此Docker讀取其輸出流，並且應用程序日誌被以容器日誌為單位。圖19.6顯示了它的工作原理。

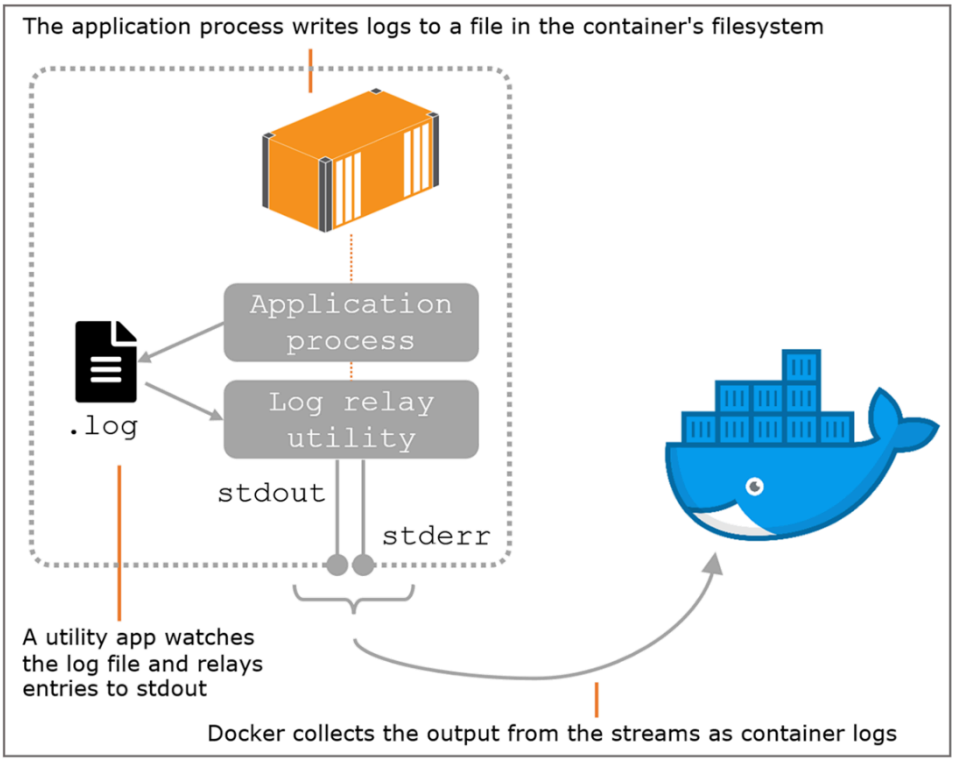


圖1　9.6您需要在容器映像中包裝一個實用程序以通過　m文件。

這不是一個完美的解決方案。 您的日誌實用程序流程在前景中運行，因此它需要堅固，因為即使實際應用程序仍在後台工作，您的容器會退出，即使您的容器退出。相反的是：如果應用程序失敗，但日誌中繼持續運行，即使應用程序不再工作，您的容器仍會保留。您需要在圖像或部署模型中進行健康檢查，以防止這種情況發生。這不是對磁盤的有效用途，尤其是如果您的應用程序寫了很多日誌 - 它們將在容器文件系統*and*中填充文件，填充Docker Host Machine上的JSON文件。

即便如此，這還是一個有用的模式。如果您的應用程序在前景中運行，並且您可以調整配置以將日誌寫入Stdout，那麼這是一種更好的方法。但是，如果您的應用程序在後台運行，則沒有其他選擇，最好接受效率低下，並像其他所有容器一樣行事。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

本章中有一個TimeCheck應用程序的更新，添加了此模式，構建一個小型實用應用程序以觀看日誌文件並將行傳遞到STDOUT。清單19.2顯示了多階段Dockerfile的最後階段。對於Linux和Windows，有不同的啟動命令，因為Shells使用不同的語法在後台啟動過程。

清單19.2與您的應用一起構建和包裝log-relay實用程序

```
＃DIAMOL/DOTNET-RUNTIME的App Image
：2e作為基礎…


WorkDir /App
複製 - 從= builder / out /。複製-fro
m =實用程序 / OUT /。 ＃Window
s


從base作為Windows CMD start /b dotnet timecheck.dll && dotnet tail.dll /logs timecheck.log



＃Linux從linux as Linux cmd dotnet timecheck.dll＆dotnet tail.dll /logs timecheck.l
og
```

這兩個CMD指令使用兩個操作系統的不同方法實現了同一件事。 首先，.NET應用程序進程是在後台開始的，在Windows中使用start命令，並在Linux中使用單個ampersand（＆）在命令中加上命令。 然後啟動.NET尾巴實用程序，並配置為讀取應用程序編寫的日誌文件。尾巴實用程序只能觀看該文件，並在編寫每條新線路時傳達每條新行，因此日誌浮出水面並成為容器日誌。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

從新圖像運行一個容器，並驗證日誌是否來自容器，並且仍然寫在文件系統中。

```
#從新圖像運行一個容器：docker run -d-name timecheck4 diamol/ch19-time
check：2e-v3

#檢查日誌：docker容器日誌timecheck4

#並在容器中打印日誌文件：docker容器exec -it timecheck4 cat /logs/timechec
k.log
```

現在，日誌來自容器。到達那裡的方法是一種令人費解的方法，並運行了一個額外的過程，可以將日誌文件內容傳遞給Stdout，但是一旦容器運行，這都是透明的。這種方法的缺點是日誌繼電器使用的額外處理能力以及兩次存儲日誌的額外磁盤空間。您可以在圖19.7中看到我的輸出，該輸出顯示日誌文件仍在容器文件系統中。

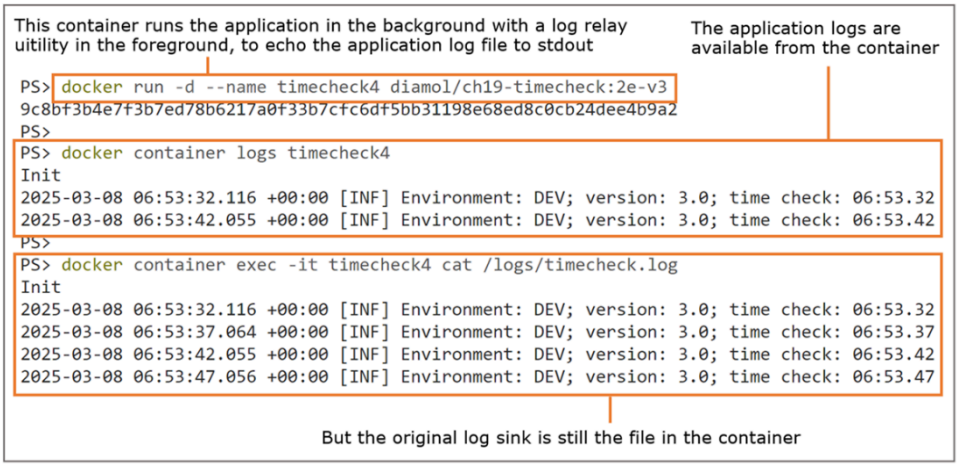


圖19.7日誌繼電器實用程序將應用程序記錄到Docker，但使用了兩倍的磁盤空間。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

我使用自定義實用程序將日誌條目中的日誌條目中繼，因為我希望該應用程序可以跨平台工作。 我可以使用標準的Linux尾部命令，但是您可以使用沒有該工具的最小基礎圖像，在Windows上，沒有等效的。自定義實用程序方法也更加靈活，因為它可以從任何水槽中讀取並繼電器到STDOUT。這應該涵蓋您的應用程序日誌被鎖定在Docker看不到的容器中的任何情況。

運行此練習時，您可能會注意到還有一個問題：應用程序編寫的日誌與作為容器日誌的表面上有一個滯後。 繼電器實用程序在編寫日誌文件時觀看，然後讀取內容並打印新行。這引入了延遲，但是除實時應用程序外，所有這些都應接受。 當您設置所有容器圖像以將應用程序日誌作為容器日誌寫入時，您可以開始使用Docker的可插入日誌記錄系統，並合併所有來自所有容器的日誌。

19.3收集和轉發容器日誌

回到第2章中，我談到了Docker如何在所有應用程序上添加一個一致的管理層，這在容器內部發生了什麼，這並不重要。您以相同的方式開始，停止並檢查所有內容。 當您將合併的日誌記錄系統帶入架構時，這對日誌特別有用，我們將瀏覽其中最受歡迎的開源示例之一：Fluentd。

Fluentd是一個統一的記錄層。它可以從許多不同來源攝入日誌，過濾或豐富日誌條目，然後將其轉發到許多不同的目標。這是一個由Cloud Native Computing Foundation管理的項目（還管理Kubernetes，Prometheus和Docker的*containerd*運行時，以及其他項目），並且是一個成熟且非常靈活的系統。您可以在容器中運行Fluentd，它將收聽日誌條目。 然後，您可以運行其他使用Docker的Fluentd Logging驅動程序而不是標準JSON文件的容器，這些容器日誌將發送到Fluentd。

現在嘗試

Fluentd使用配置文件來處理日誌。運行一個具有簡單配置的容器，該容器可以設置Fluentd來收集日誌並迴聲以在容器中刻上。 然後使用該容器將日誌發送到Fluentd。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
CD CH19/練習/Fluentd

＃運行Fluentd發布標準端口並使用配置文件：Docker Run -D -P 24224：24224 -name Fluentd -v “
$（pwd）/conf：/conf：/conf：/fluentd/etc” -e fluentd_conf =



＃現在運行一個TimeCheck容器集用於使用Docker的Fluentd日誌驅動程序：Docker Run -d-Log-Driver
= fluentd -name TimeCheck5 Diamol/ch19-Timecheck：2e-v3

＃檢查TimeCheck容器日誌：Docker Logs Time
Check5

＃並檢查Fluentd容器日誌：Docker Logs -Tail 1 Fl
uentd
```

Fluentd Logging驅動器內置在Docker發動機中，並且可以透明地工作 - 將容器日誌發送到Fluentd容器，但在打印容器日誌時也會顯示它們。 Fluentd從Docker收集日誌輸出並進行處理。 在這種配置中，它只會將輸出寫入Stdout，因此您還可以通過查看Fluentd容器的日誌來查看Timecheck容器的日誌。 我的輸出在圖19.8中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Run Fluentd in a container. This listens on port 24224 for log entries, and the configuration file sets it to echo all the logs it receives to stdout

Run an application container using the Fluentd logging driver - Docker will send all container logs to port 24224

```
PS> cd ch19/exercises/fluentd
PS>
PS> docker run -d -p 24224:24224 --name fluentd `
>>  -v "$(pwd)/conf:/fluentd/etc" `
>>  -e FLUENTD_CONF=stdout.conf `
>>  diamol/fluentd:2e
ca9a996162b2bf32cc995199a510703e1a667f7f22a3d448fc3f083a7b74f079
PS>
PS> docker run -d --log-driver=fluentd --name timecheck5 `
>>  diamol/ch19-timecheck:2e-v3
528e726f03191aeab3977c1c9943a2d7dbd79fbad7b1542038e0d66355701ad9
PS>
PS> docker logs timecheck5
Init
2025-03-08 07:18:32.376 +00:00 [INF] Environment: DEV; version: 3.0; time check: 07:18.32
PS>
PS> docker logs --tail 1 fluentd
2025-03-08 07:18:43.000000000 +0000 528e726f0319: {"container_id":"528e726f03191aeab3977c1c
9943a2d7dbd79fbad7b1542038e0d66355701ad9","container_name":"/timecheck5","source":"stdout",
"log":"2025-03-08 07:18:42.290 +00:00 [INF] Environment: DEV; version: 3.0; time check: 07:
18.42"}
```

The Fluentd logging driver still exposes container logs

This Fluentd setup is echoing logs to stdout, so I can see the same logs from my timecheck container in the Fluentd container

圖19.8 Fluentd從其他容器中收集日誌，它可以存儲或寫入Stdout。

Fluentd在存儲日誌時（包括容器ID和名稱）時，將其自己的元數據添加到每個記錄中。您之所以需要這個，是因為Fluentd成為所有容器的中央日誌收集器，並且必須能夠確定哪些日誌條目來自哪個應用程序。將Stdout作為Fluentd的目標只是查看所有內容如何工作的一種簡單方法。通常，您會將日誌轉發到中央數據存儲。 Elasticsearch是一個非常受歡迎的選項 - 它是一個適用於日誌的NO-SQL文檔數據庫。您可以在一個容器中運行lasticsearch，以進行日誌存儲和另一個容器中的搜索UI的伴隨應用Kibana。圖19.9顯示了記錄模型的外觀。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

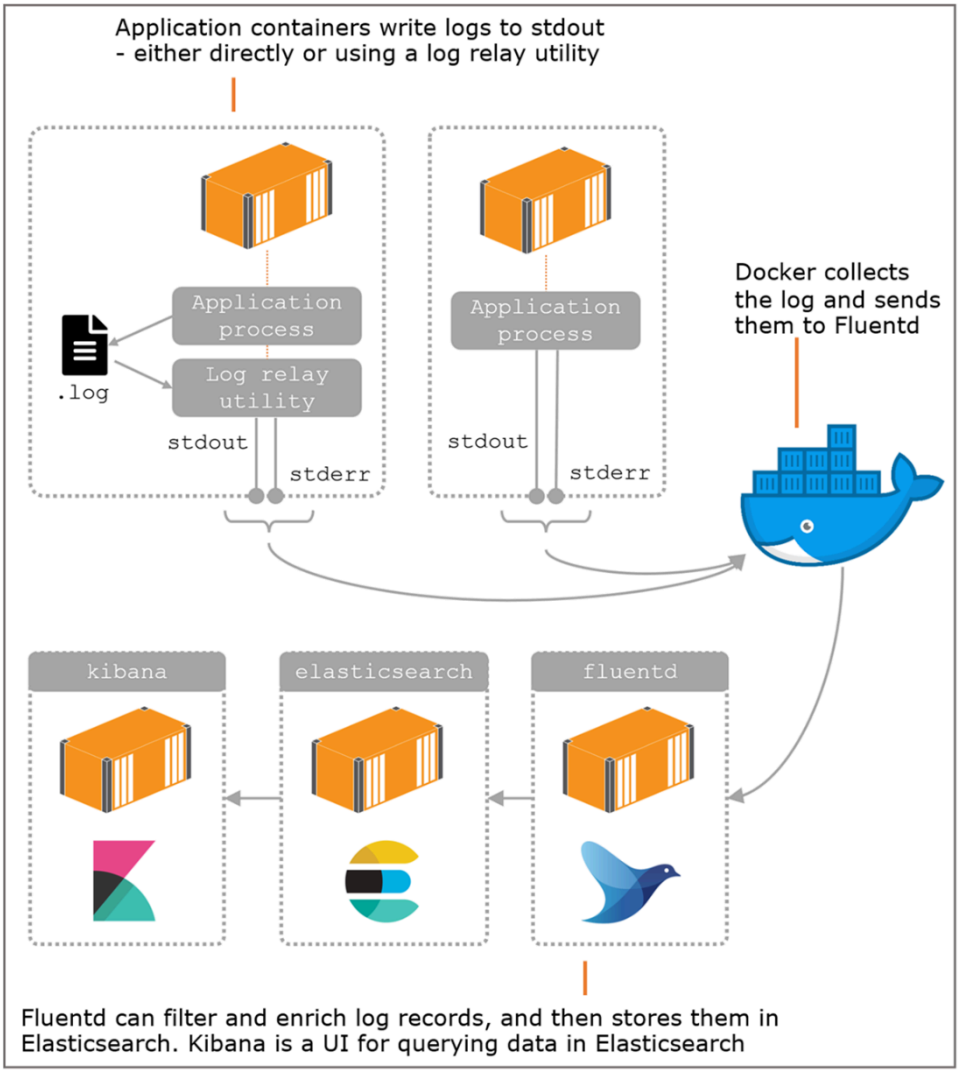


圖19.9集中式記錄模型將所有容器日誌發送到Fluentd進行處理和存儲。

它看起來像一個複雜的體系結構，但與Docker一樣，很容易在Docker組合文件中對記錄設置的所有部分進行建模，然後用一個命令將整個堆棧旋轉。當您在容器中運行的日誌記錄基礎架構時，只需將Fluentd Logging驅動程序用於任何要選擇進入集中日誌記錄的容器即可。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

卸下任何運行的容器，然後啟動Fluentd-elasticsearch-kibana記錄容器。然後使用Fluentd Logging驅動程序運行Timecheck容器。

```
＃在前景中運行容器：docker rm -f $（docker ps -
aq）

＃切換到練習文件夾：CD CH19/練習

＃啟動伐木堆棧：docker compose -f fluentd/docker -compose.yml
up -d

＃並運行一個新的TimeCheck容器來生成日誌：docker run -d-log-driver = fluent
d diamol/ch19-timecheck：2e-v3
```

給Elasticsearch幾分鐘準備就緒，然後通過http：// localhost瀏覽到基巴納：5601/app/discover。 您需要設置Kibana才能在Elasticsearch中找到日誌 - 單擊創建數據視圖，表單將要求使用文檔集合的名稱（在Elasticsearch中稱為*index*）以搜索。如圖19.10所示，將日誌用作數據視圖名稱，FluentD*作為索引模式，將@timestamp用作時間戳字段。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

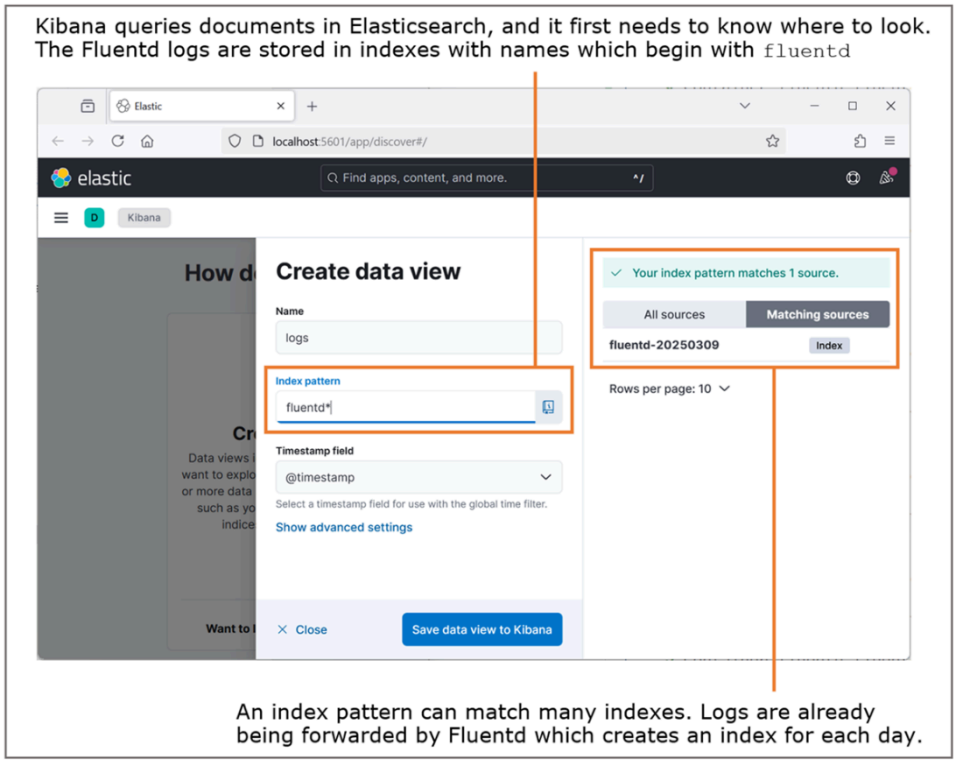


圖19.10 Elasticsearch將文檔存儲在稱為索引的集合中 - Fluentd每天創建一個。

單擊將數據視圖保存到Kibana，發現頁面將打開，顯示迄今為止收集的所有日誌。您會看到為什麼Elasticsearch非常適合日誌集合，因為日誌被存儲為帶有可以搜索和過濾的字段的結構化文檔，但是沒有數據庫架構，這意味著日誌條目可以具有不同的字段。 圖19.11顯示了Timecheck應用程序中的一些日誌 - 以及記錄日誌時的日誌條目，您可以看到容器名稱，容器ID和時間戳。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

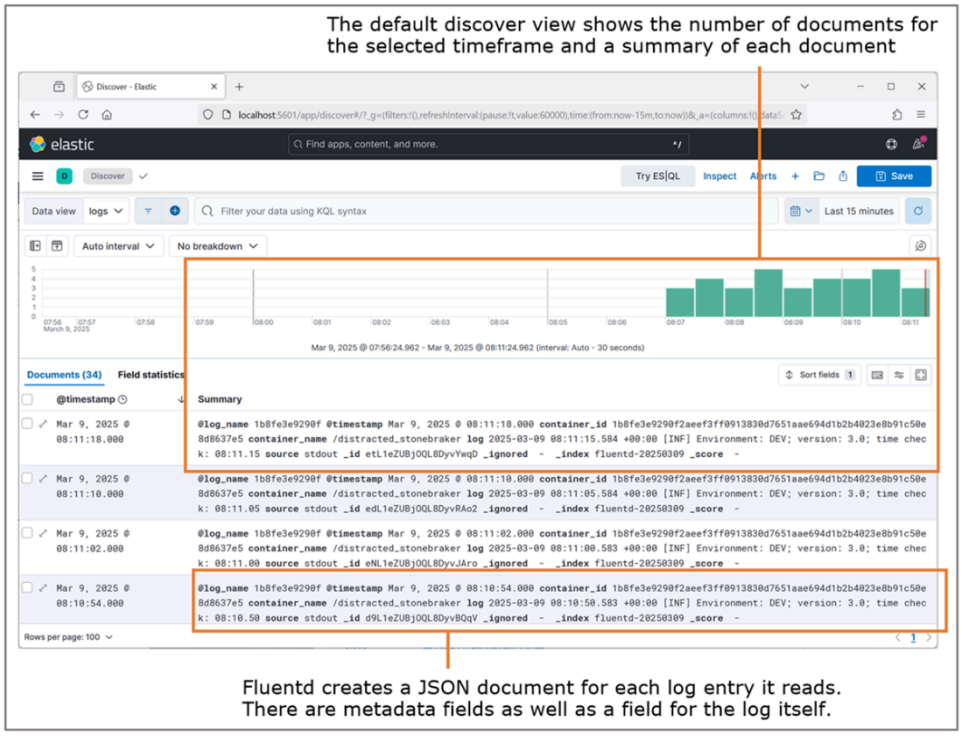


圖19.11 Fluentd已經在Elasticsearch中保存了數據，因此Kibana可以看到字段名稱。

您可以自動化Kibana設置，但我沒有，因為如果您是Elasticsearch堆棧的新手，那麼值得一經踏上它，看看這些件如何結合在一起。每個日誌輸入Fluentd收集的收集都在Elasticsearch中保存為文檔，該文檔集合在名為Fluentd- {date}的文檔集合中保存。 Kibana為您提供了所有這些文檔的視圖 - 在“ Discover”選項卡中，您會看到一個條形圖，顯示隨著時間的推移創建了多少個文檔，並且您可以鑽入各個文檔的詳細信息。 在此練習中，每個文檔都是Timecheck應用程序的日誌條目。您可以在圖19.12中的基巴納（Kibana）中看到數據。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

圖19.12 EFK堆棧的所有榮耀 - 收集和存儲以進行簡單搜索

Kibana可讓您在所有文檔中搜索特定文本，或按日期或其他字段進行過濾文檔。它還具有類似於Grafana的儀表板功能，您在第9章中看到了該儀表板功能，因此您可以構建圖表，顯示每個應用程序的日誌計數或錯誤日誌計數。 Elasticsearch非常可擴展，因此它適用於生產中的大量數據，當您開始通過Fluentd發送所有容器日誌時，您很快就會發現，它比瀏覽控制台中的日誌線更容易管理。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

運行圖像庫應用程序，每個組件都配置為使用Fluentd Loggging驅動程序。

```
#來自CD CH19/練習文件夾

#啟動apod應用程序：
Docker Compose -f圖像 - 套裝/docker -compose.yml up -d
```

瀏覽到http：// localhost：8010生成一些流量，這些容器將開始編寫日誌。圖像庫應用程序的Fluentd設置在每個日誌中添加標籤，識別生成的組件，因此可以輕鬆地識別日誌線，而不是使用容器名稱或容器ID。 您可以在圖19.13中看到我的輸出。 我正在運行完整的Image Gallery應用程序，但是我正在過濾Kibana中的日誌，以顯示訪問-LOG組件 - 訪問應用程序時記錄的API。

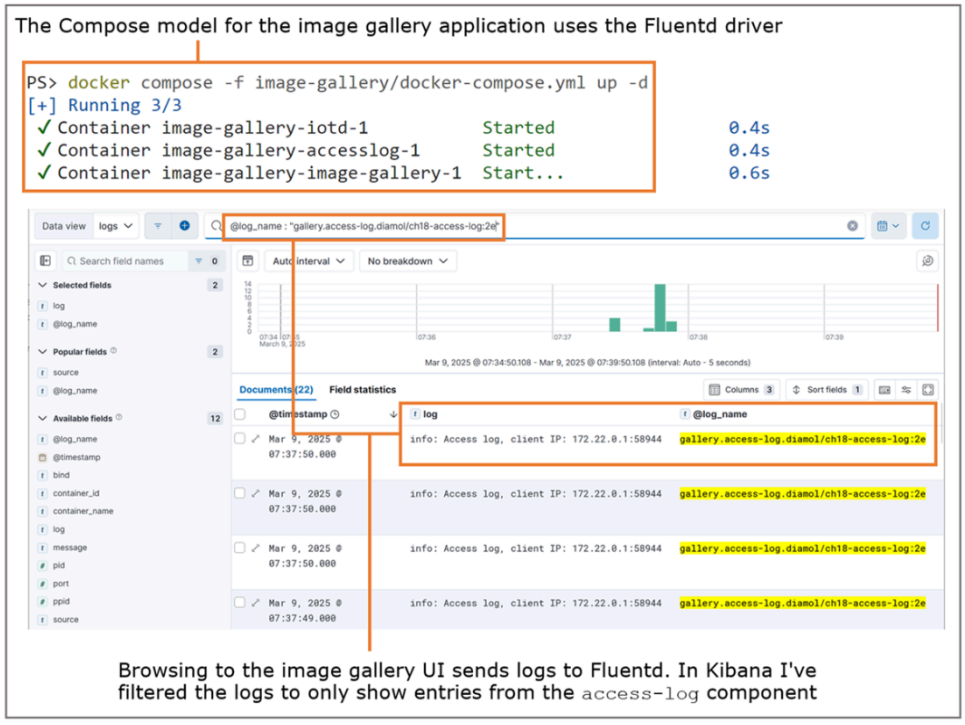


圖19.13在Elasticsearch中收集了圖庫和Timecheck容器的日誌。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

您可以在容器的日誌記錄選項中指定日誌條目的標籤，並且數據存儲在Elasticsearch中的log_name字段中，因此您可以在Kibana中過濾。您可以使用固定的名稱或註入一些有用的標識符 - 在本練習中，我將畫廊用作應用程序前綴，然後為生成日誌的容器添加組件名稱和圖像名稱。 這是確定每個日誌線運行的應用程序，組件和精確版本的好方法。 清單19.3顯示了Docker撰寫圖庫應用程序的文件記錄選項。

清單19.3使用標籤來識別Fluentd的日誌條目的來源

```
services:
  accesslog:
    image: diamol/ch18-access-log:2e
         logging:
      driver: "fluentd"
      options:
        tag: " gallery.access-log.{{.ImageName}}"

  iotd:
    image: diamol/ch18-image-of-the-day:2e
    logging:
      driver: "fluentd"
      options:
        tag: "gallery.iotd.{{.ImageName}}"

  image-gallery:
    image: diamol/ch18-image-gallery:2e
    logging:
      driver: "fluentd"
      options:
        tag: "gallery.image-gallery.{{.ImageName}}"
…
```

使用可搜索的數據存儲和用戶友好的UI的集中記錄的模型是當您準備好生產的容器時，應該考慮的模型。您不僅限於使用Fluentd，還有許多其他用於Docker的伐木驅動程序，因此您可以使用其他流行的工具，例如Graylog或Splunk等商業工具。 請記住，您可以在Docker配置中的引擎級別設置默認的記錄驅動程序和選項，但我認為在應用程序模型中進行此操作是有價值的，它可以清楚您在每個環境中使用的是哪種記錄系統。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

如果您還沒有建立的記錄系統，則Fluentd是一個不錯的選擇。 它易於使用，並且可以從一台開發機器到完整的生產容器平台擴展，並且您在每個環境中都以相同的方式使用它。 您還可以配置Fluentd來豐富日誌數據，以使其更易於使用，並過濾日誌並將其發送到不同的目標。

19.4管理您的日誌輸出和收集

記錄是捕獲足夠的信息以在診斷問題和不浪費空間來存儲大量數據的空間之間的微妙平衡。 Docker的日誌記錄型號為您提供了一些額外的靈活性來幫助達到平衡，因為您可以在詳細的水平上生產比您預期的更詳細的容器日誌，但在存儲之前將其過濾掉。然後，如果您需要查看更多的詳細日誌，則可以更改過濾器配置而不是應用程序配置，以便更換Fluentd容器而不是您的應用程序容器 - 因此，當您調整日誌記錄級別時沒有停機時間。

您可以在Fluentd配置文件中配置此級別的過濾。最後一個練習中的配置將所有日誌發送給Elasticsearch，但是在列表19.4中更新的配置中，從更詳細的訪問-LOG組件中濾出了日誌。這些日誌轉到Stdout，其餘的應用程序日誌轉到Elasticsearch。

列表19.4根據記錄的標籤將日誌條目發送到不同目標

```
<匹配庫.Access-log。
```

匹配塊告訴Fluentd如何處理日誌記錄，並且過濾器參數使用在記錄驅動程序選項中設置的標籤。當您運行此更新的配置時，訪問-Log條目將匹配第一個匹配塊，因為標籤前綴是Gallery.Access-log。 這些記錄將停止在Elasticsearch中浮出水面，並且只能通過閱讀Fluentd容器的日誌來獲得。更新的配置文件還豐富了所有日誌條目，將標籤分為單獨的字段，以獲取應用程序名稱，服務名稱和圖像名稱，這使得在Kibana中進行過濾得很容易。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

通過部署Docker組合覆蓋文件來更新FluentD配置，該文件指定新的配置文件，然後更新圖像庫應用程序以生成更多的詳細日誌。

```
＃更新fluentd配置：docker compose -f fluentd/docker -compose.yml -f fluentd/override -gallery -efflyed.yml up -d


＃更新應用程序記錄配置：Docker Compose -F Image-Gallery/Docker-compose.yml -f Image-Gallery/Override- logging.yml up -d
```

您可以檢查這些覆蓋文件的內容，並且您會看到它們只是為應用程序指定配置設置。 所有圖像都是相同的。 現在，當您在http：// localhost：8010上使用該應用程序時，Access-Log條目仍會生成，但是它們會被Fluentd濾除，因此您不會在Kibana中看到任何新日誌。您會看到來自其他組件的日誌，這些日誌充滿了新的元數據字段。您可以在圖19.14中的輸出中看到這一點。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

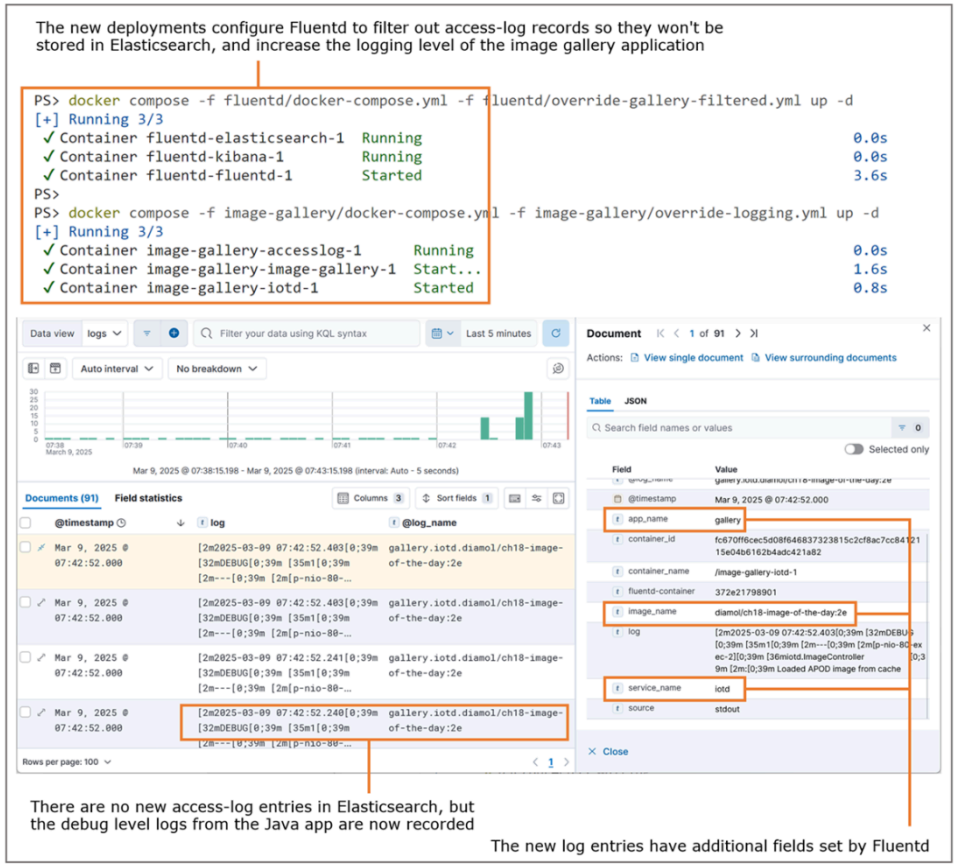


圖19.14 Fluentd使用日誌中的標籤來過濾記錄並生成新字段。

仍然可以使用Access-Log條目，因為它們正在寫入Fluentd容器中的stdout。 您可以將它們視為容器日誌，但可以從Fluentd容器而不是訪問儀容器中看到它們。

現在嘗試

檢查Fluentd容器日誌，以確保記錄仍然可用。

```
Docker Logs-Tail 1 Fluentd-Fluentd-1
```

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

您可以在圖19.15中看到我的輸出。 Access-Log條目已發送到另一個目標，但是它仍然經過相同的處理，可以通過應用程序，服務和圖像名稱豐富記錄：

Fluentd is configured to send access-log records to stdout,
so they're available as logs from the Fluentd container

```
PS> docker logs --tail 1 fluentd-fluentd-1
2025-03-09 07:42:52.000000000 +0000 gallery.access-log.diamol/ch18-access-log:2e:
{"container_id":"02c540a902cfe4d069357f78635b2c234713964eadbd46b0f89e55ea92c75443"
,"container_name":"/image-gallery-accesslog-1","source":"stdout","log":"info: Acce
ss log, client IP: 172.22.0.1:42034","fluentd-container":"372e21798901","app_name"
:"gallery","service_name":"access-log","image_name":"diamol/ch18-access-log:2e"}
```

The records have been updated to split the
tag into app, service and image name fields

圖19.15這些日誌被過濾，因此它們沒有存儲在Elasticsearch中，而是迴盪以進行Stdout。

這是將核心應用程序日誌與良好到達日誌分開的好方法。您不會在生產中使用Stdout，但是對於不同類別的日誌，您可能會有不同的輸出 - 至關重要的日誌條目可以將日誌條目發送到Kafka，面向用戶的日誌可能會用於Elasticsearch，其餘的可以在Amazon S3 Cloud Storage中提交。 這些都是Fluentd中受支持的日誌存儲。

本章有一個最後的練習來重置日誌記錄，並將訪問錄入條目重新放回Elasticsearch中。 這近似於您發現系統問題的生產情況，並且想增加日誌以查看發生的事情。 使用日誌記錄設置，該日誌已經由應用程序編寫。我們只能通過更改Fluentd配置文件來浮出水面。

現在嘗試

部署一種新的Fluentd配置，該配置將Access-Log記錄發送到Elasticsearch。

```
Docker Compose -f Fluentd/docker -compose.yml -f fluentd/override -gallery.yml up -d
```

此部署使用一個配置文件，該文件將刪除訪問記錄的匹配塊，因此所有圖庫組件日誌都存儲在Elasticsearch中。當您刷新瀏覽器中的圖像庫頁面時，將收集並存儲日誌。您可以在圖19.16中看到我的輸出，其中最新日誌從API和Access-Log組件中顯示。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

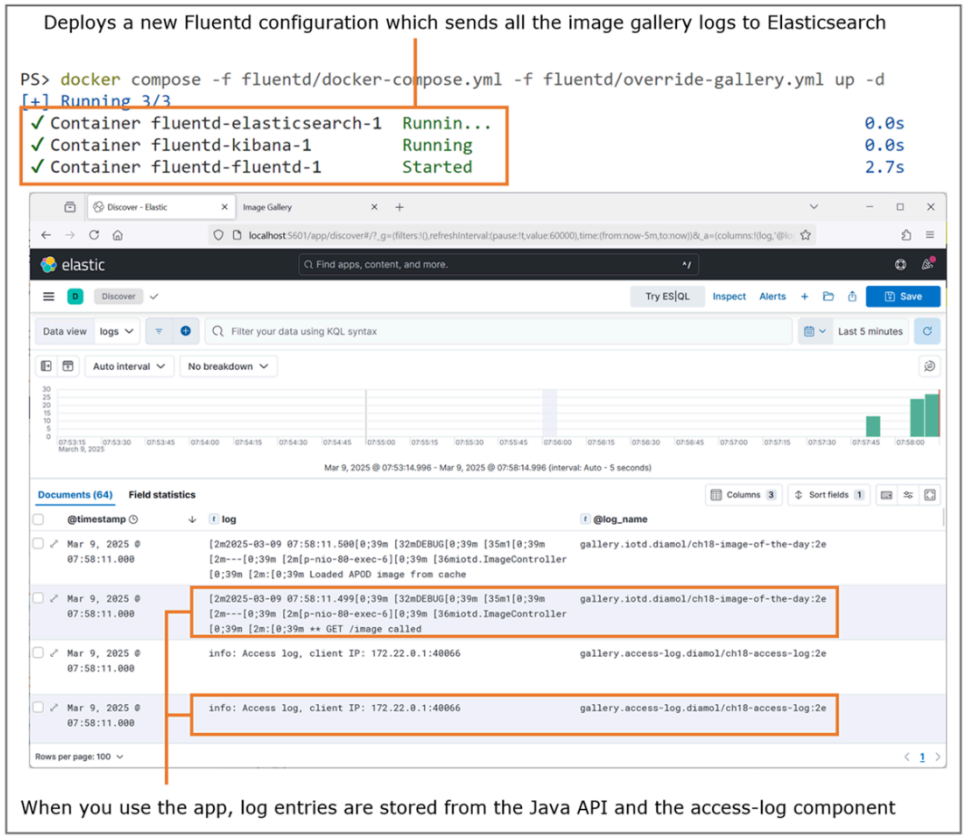


圖19.16對Fluentd配置的更改將日誌添加回Elasticsearch，而無需任何應用程序更改。

您確實需要意識到使用這種方法有可能丟失日誌條目。在部署期間，當沒有運行Fluentd的容器來收集它們時，容器可能會發送日誌。 在這種情況下，Docker可以優雅地繼續進行，您的應用程序容器不斷運行，但是日誌條目不會被緩衝，因此會丟失。 在聚類的生產環境中，這不太可能是一個問題，但是即使確實發生了，也最好重新啟動使用日誌記錄配置增加的應用程序容器 - 至少是因為新容器可能與舊容器沒有相同的問題，因此您的新日誌不會告訴您任何有趣的事情。

19.5了解容器記錄模型

Docker中的記錄方法是超級誘的，但僅當您使應用程序日誌可見為容器日誌時。 您可以直接通過將應用程序寫入日誌來直接執行此操作，或者通過在容器中使用繼電器實用程序將log條目複製到stdout中間接做到這一點。 您需要花一些時間確保所有應用程序組件都寫入容器日誌，因為一旦您可以使用該日誌，就可以根據自己喜歡的方式處理日誌。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

我們在本章中使用了EFK堆棧（Olasticsearch，Fluentd和Kibana），並且您已經看到使用用戶友好的搜索UI將所有容器日誌拉入集中式數據庫是多麼容易。 所有這些技術都是可交換的，但是Fluentd是最常用的技術之一，因為它是如此簡單和強大。該堆棧在單機器環境中很好地運行，並且它也可以為生產環境而定，並在內置支持Kubernetes。圖19.17顯示了聚類環境如何在每個節點上運行一個流利的容器，其中Fluentd容器從該節點上的其他容器中收集日誌，並將其發送到彈性搜索群集，並在容器中運行。

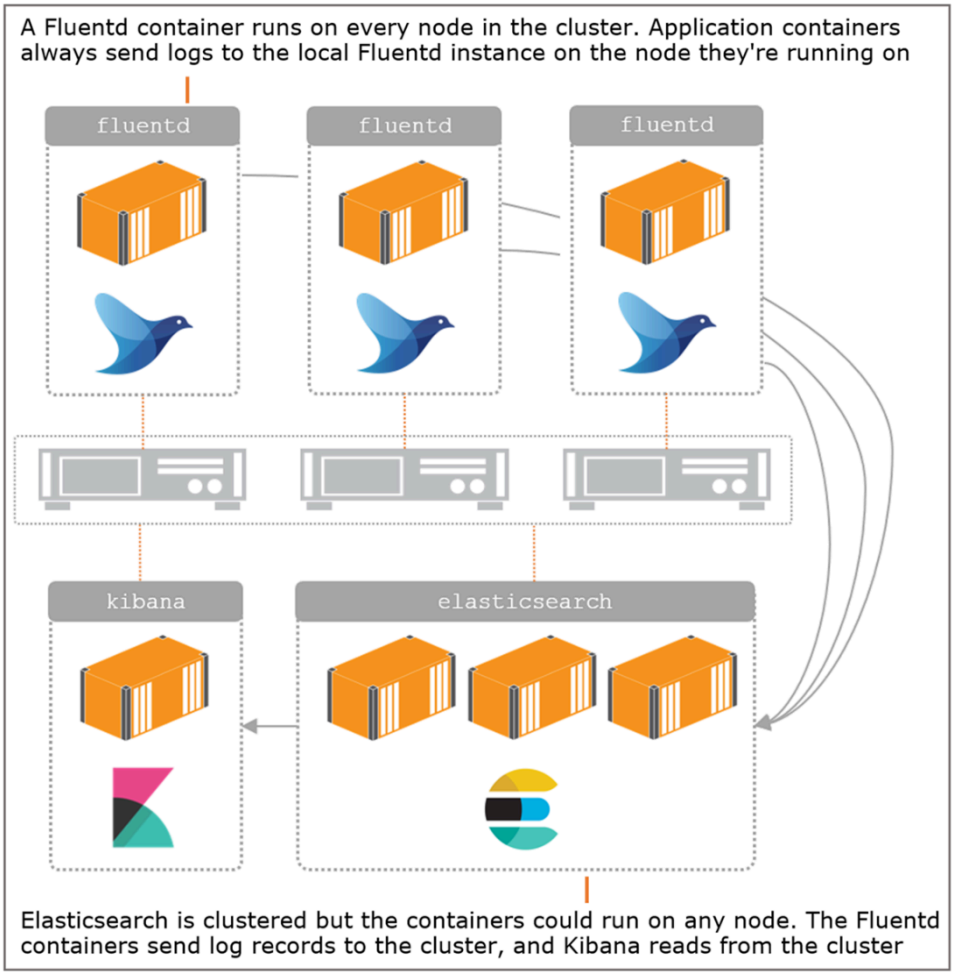


圖19.17 EFK堆棧在生產中起作用，並具有群集存儲和多個FluentD實例。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

在進入實驗室之前，我會謹慎謹慎。有些團隊不喜歡此容器記錄模型中的所有處理層； 他們更喜歡將應用程序日誌直接寫入最終商店。因此，該應用程序直接將數據直接寫給Elasticsearch，而不是讓Fluentd將數據發送到Elasticsearch，而不是將數據發送給Elasticsearch。 我真的不喜歡這種方法。您可以節省一些處理時間和網絡流量，以換取完全缺乏靈活性。您已經將日誌記錄堆棧進行了硬編碼，並且如果您想切換到Graylog或Splunk，則需要去重新工作。我總是更喜歡保持簡單和靈活的功能 - 編寫您的應用程序日誌以進行stdout並利用平台收集，豐富，過濾和存儲數據。

19.6實驗室

在本章中，我並沒有過多地專注於配置Fluentd，但是值得獲得一些經驗，因此我要請您在實驗室中進行。在本章的實驗室文件夾中，是一個Docker撰寫的隨機號碼應用程序，而Docker構成了EFK堆棧的文件。該應用程序容器沒有配置為使用Fluentd，並且Fluentd設置沒有任何功能，因此您有三個任務：

- 擴展數字應用程序的撰寫文件，以便所有組件都使用Fluentd Logging驅動程序，並設置一個帶有應用程序名稱，服務名稱和圖像的標籤。
- 將FluentD配置文件（Elasticsearch.conf）擴展，以將標籤拆分為來自數字應用程序中所有記錄的應用程序名稱，服務名稱和圖像名稱字段。
- 在FluentD配置中添加FailSafe匹配塊，以便任何不是來自數字應用程序的記錄都將轉發到Stdout。

沒有提示此內容，因為這是通過圖庫應用程序的配置設置工作並查看需要為數字應用程序添加的零件的情況。 與往常一樣，我的解決方案在github上供您檢查：https：// github。 com/sixeyed/diamol/blob/2e/ch19/lab/readme.md。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

20 將HTTP流量控製到具有反向代理的容器

Docker負責將外部流量路由到您的容器中，但是您只能在網絡端口上聆聽一個容器。在非生產環境中使用任何舊端口是可以的 - 在本書的某些章節中，我們已經使用了十個不同的端口將應用程序分開，但是當您上線時，您無法做到這一點。 您需要在一個群集上運行大量應用程序，但是您需要在標準HTTP和HTTPS端口（80和443）上訪問它們。

那就是*reverse proxy*進來的地方。它是容器化環境的架構中的關鍵部分，在本章中，您將了解其提供的功能及其啟用的模式。當然，我們將使用該領域中最受歡迎的兩種技術 - Nginx（發音為“引擎X”）和Traefik（大約明顯的“流量”） - 當然是在容器中運行的。

20.1什麼是反向代理？

*proxy*是一個網絡組件，代表其他組件處理網絡流量。您可能會在公司網絡中有一個代理，該代理截止您的瀏覽器請求，並決定是否允許您訪問某些站點，記錄所有活動並緩存響應，以便同一網站的其他用戶獲得更快的體驗。反向代理會做類似的事情，但相反。您運行反向代理作為通往多個Web應用程序的網關；所有流量都歸頻代理，它決定從哪個應用程序中獲取內容。它可以緩存響應並將其突變，然後再將它們發送回客戶。 圖20.1顯示了容器中反向代理的外觀。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

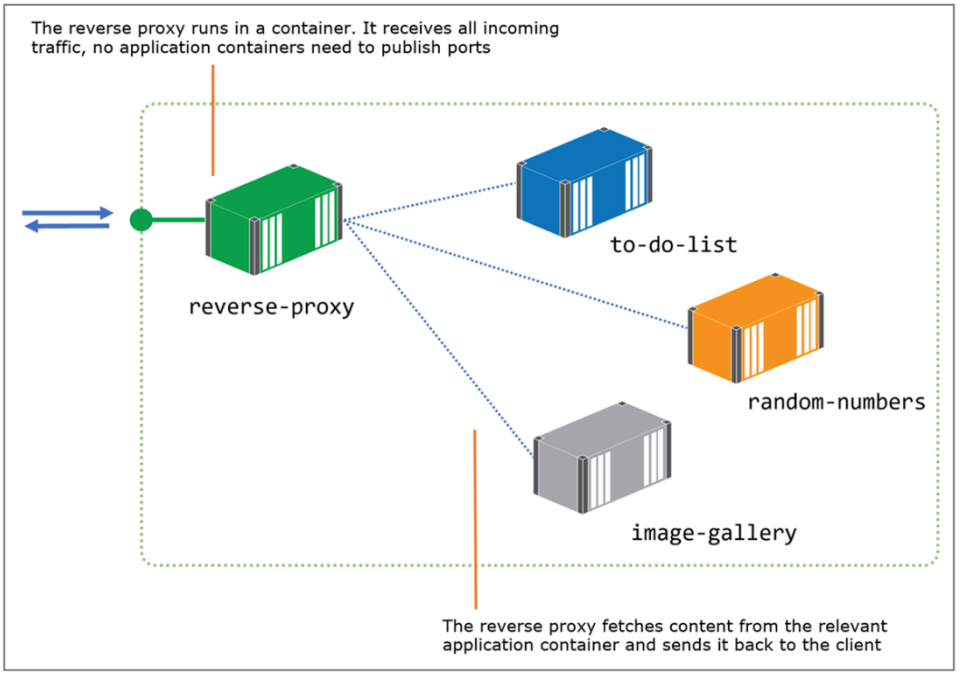


圖20.1反向代理是通往應用程序的門戶。申請容器無法公開可用。

反向代理是唯一具有已發布端口的容器，它會收到所有傳入的請求，並從其他容器中獲取響應。這意味著您的所有應用程序容器都成為內部組件，這可以使其更容易擴展，更新和保護它們。 反向代理不是一項新技術，但是它們隨著集裝箱革命而向左移動。 他們曾經坐在生產中，並由OPS團隊管理，甚至沒有開發人員知道有代理人。現在它們以輕質容器運行，您可以在每個環境中具有相同的代理配置。

現在嘗試

多年來，NGINX一直是流行的反向代理選擇，它的功率超過了30%的互聯網。 這是一款非常輕巧，快速且功能強大的HTTP服務器，可以提供自己的內容以及代理其他服務器。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
# create a network for this chapter's apps - for Linux containers: docker network create ch
20 # OR for Windows Server containers: docker network create --driver=nat ch20 # switc
h to the exercise folder: cd ch20/exercises # run Nginx with a bind mount to a local config
uration folder: docker compose -f nginx/docker-compose.yml up -d




＃瀏覽到http：// localhost
```

Nginx uses a configuration file for each of the websites it serves. This container has a bind mount to the local folder called `sites-enabled`, but there are no config files in there yet. Nginx has a default site that is a simple HTML page—you can see my output in figure 20.2.

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

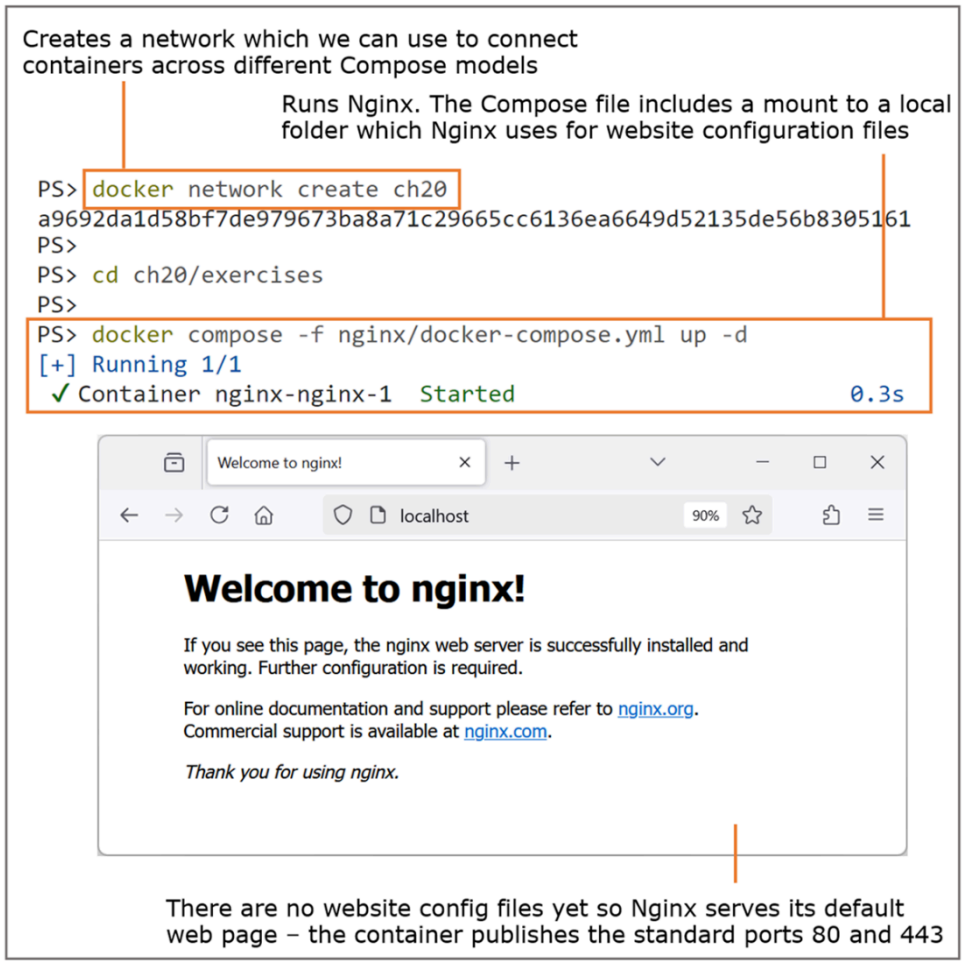


圖2　0.2 nginx是HTTP服務器 - 它可以提供靜態內容並作為REVE運行　RSE代理。

我們還沒有使用Nginx作為反向代理，但是我們可以通過為另一個網站添加配置文件來設置它。當您在同一端口上託管多個應用程序時，您需要一種方法來區分它們，通常使用網站的域名。當您瀏覽https：//博客之類的網站時。 sixeyed.com，瀏覽器在客戶端請求中包含一個HTTP標頭：主機= blog.sixeyed.com。 NGINX使用該主機標頭找到用於服務站點的配置文件。在本地計算機上，您可以在主機文件中添加域，這是一個簡單的DNS查找，可以從您的NGINX容器中提供不同的應用程序。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

我們將在容器中運行一個簡單的WHO-AM-I Web應用程序，而無需發布任何端口，並通過主機域Whoami.local上的Nginx提供。

```
＃將域添加到Mac或Linux上的本地主機文件：../../scripts/add-to-host
s.sh whoami.local

＃或使用PowerShell（使用高架外殼）：../../../scripts/add-to-hosts.
ps1-domain whoami.local＃啟動WHO-AM-I容器：Docker compose
-f whoami/docker-compose.yml




＃將應用程序配置複製到nginx配置文件夾：cp ./nginx/sites-available/whoami.lo
cal./nginx/sites-enets-enabled/

＃並運行nginx命令以重新加載config：docker exec nginx-nginx-1 ngi
nx -s reload

＃測試應用程序：
  curl http：//whoami.local
```

當您將請求發送到http：//whoami.local時，主機文件中的條目將您引導到本地計算機，而Nginx容器會收到請求。 它使用HTTP標頭主機= whoami.local查找正確的網站配置，然後從WHO-AM-I容器中加載內容並將其發送回。您會在圖20.3中看到響應與響應直接來自WHO-AM-I應用程序容器相同。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

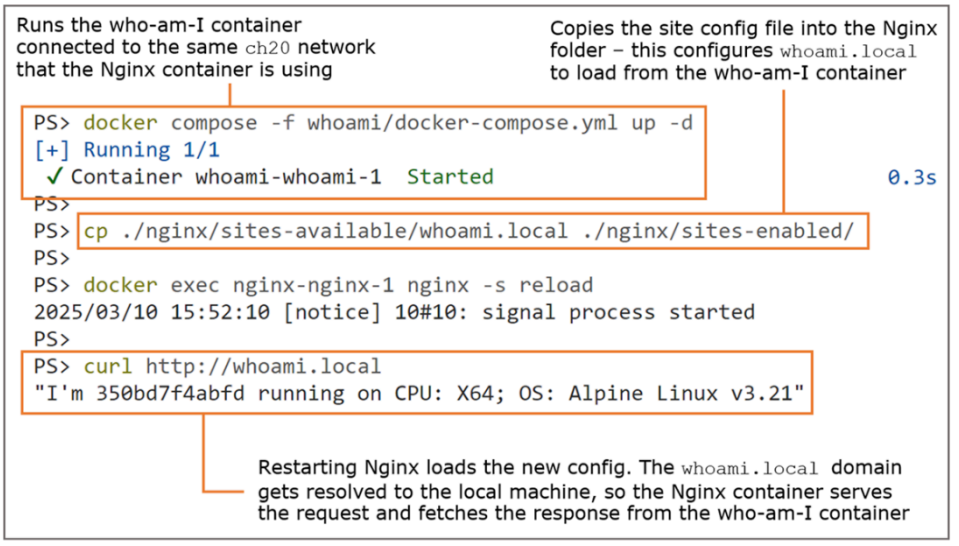


圖20.3行動中的反向代理，從應用程序容器的幕後加載內容

NGINX是一個功能強大的服務器，具有巨大的功能集，但是代理Web應用程序的基本配置文件非常簡單。您需要指定服務器的域名和內容的位置，這可以是內部DNS名稱。 NGINX容器將使用DNS的容器名稱從Docker網絡上從應用程序容器中獲取內容。清單20.1顯示了WHO-AM-I網站的完整配置文件。

列表20.1 NGINX代理配置WHO-AM-I網站

```
server {
  server_name whoami.local;               # the domain host name

  location / {
    proxy_pass        http://whoami:8080; # source address for content
    proxy_set_header  Host $host;         # copy the host header
    add_header        X-Host $hostname;   # add proxy name in response
  }
}
```

反向代理不僅適用於網站，它們適用於任何HTTP內容，因此REST API是好的目標，並且也可能支持其他類型的流量（Pline TCP/IP或GRPC）。 這種簡單的配置使NGINX像傳遞一樣工作，因此，對於它收到的每個請求，它將調用源容器（稱為*upstream*）並將響應發送回客戶端（稱為*downstream*）。如果上游應用程序失敗，則NGINX將失敗響應發送回下游客戶端。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

將另一個域添加到主機文件中，然後運行隨機數應用程序的API，並使用Nginx代理。這是幾次通話後失敗的API，刷新後您會看到Nginx的500響應。

```
＃在Mac或Linux上添加域：../../scripts/add-to-hosts.sh api.nu
mbers.local

＃或使用PowerShell（高架）：../../scripts/add-to-hosts.ps1- domain api.n
umbers.local

＃運行API：
docker compose -f號/docker -compose.yml up -d

＃複製網站配置文件：cp ./nginx/sites-api.numbers.local./nginx/nginx/sites-enabled/＃rel
oad nginx config：docker exec exec nginx-nginx-nginx-nginx -s s s s s s reload

＃測試API並重複直到破裂捲曲http：//api.numbers.loc
al/rng
```

從本練習中，您可以看到應用程序的用戶體驗是相同的，無論是直接訪問還是通過NGINX訪問。您有兩個從NGINX託管的應用程序，因此它正在管理到上游容器的路由。但是，它不會操縱流量，因此響應機構與應用程序容器發送的完全一樣。圖20.4顯示了API通過反向代理返回的故障響應。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
Runs the random-number API container
in the same Docker network as Nginx

                                        Adds the site file to the enabled sites
                                        folder and reloads Nginx config

PS> docker compose -f numbers/docker-compose.yml up -d
[+] Running 1/1
 ✓ Container numbers-numbers-api-1  Started                                  0.3s
PS>
PS> cp ./nginx/sites-available/api.numbers.local ./nginx/sites-enabled/
PS>
PS> docker exec nginx-nginx-1 nginx -s reload
2025/03/10 15:55:08 [notice] 18#18: signal process started
PS>
PS> curl http://api.numbers.local/rng
54
PS> curl http://api.numbers.local/rng
88
PS> curl http://api.numbers.local/rng
81
PS> curl http://api.numbers.local/rng
{"type":"https://tools.ietf.org/html/rfc9110#section-15.6.1","title":"An error occurred while
 processing your request.","status":500,"traceId":"00-06f4bf54f26976238d537444ce0df50f-2311d2
d62af2ee4d-00"}

This API fails if you call it more than three times, and when that happens it sends
a 500 error response. Proxying the app with Nginx gets the same response
```

圖20.4在簡單的代理配置中，nginx發送了該應用程序的響應，即使是故障。

反向代理可以做的遠遠超過這一點。 您的所有應用程序流量都進入代理，因此它可以是配置的中心地點，您可以將很多基礎架構級別的問題放在應用程序容器中。

20.2在反向代理中處理路由和SSL

我們一直在遵循將新應用添加到NGINX的過程是啟動應用程序容器，在配置文件中復制並重新加載Nginx配置。該順序很重要，因為，當Nginx讀取服務器配置時，它會檢查它可以訪問所有上游。 如果沒有任何東西，它將退出； 如果它們全部可用，它將構建一個內部路由列表，將主機名鏈接到IP地址。這是代理人可以處理的第一個基礎架構問題 - 如果有多個上游容器，它將加載平衡請求。

現在嘗試

我們現在將運行Image Gallery應用程序，通過NGINX代表主要的Web應用程序。我們可以擴展Web組件，NGINX將在容器之間加載平衡請求。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
＃將域添加到Mac或linux上的主機：../../scripts/add-to-hosts.sh i
mage-gallery.local


＃或使用PowerShell（高架）：../../scripts/add-to-hosts.ps1- domain image-g
allery.local


# run the app with 3 web containers: docker compose -f ./image-gallery/docker-compose.yml up -d --scale imag
e-gallery=3 # add the config file and restart Nginx: cp ./nginx/sites-available/image-gallery.local ./nginx/sites-en
abled/ docker exec nginx-nginx-1 nginx -s重新加載＃幾次致電網站：curl -i--head http：//image-gallery.local
```

圖片庫網站的NGINX配置與列表20.1中的代理設置相同，使用其他主機名和上游DNS名稱。它還添加了一個額外的響應標頭X-Upstream，該標頭顯示了NGINX從中獲取響應的容器的IP地址。您在圖20.5中看到，上游IP地址在我的172.20範圍內，這是應用程序集裝箱網絡上的應用程序集裝箱的IP地址。如果重複幾次捲曲調用，則會將不同的IP地址視為Web容器之間的NGINX負載量。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Run the image gallery app with three web containers, connected to the same Docker network as the Nginx container

```
PS> docker compose -f ./image-gallery/docker-compose.yml up -d --scale image-gallery=3
[+] Running 5/5
 ✓ Container image-gallery-iotd-1            Started                                0.4s
 ✓ Container image-gallery-accesslog-1       St...                                  0.4s
 ✓ Container image-gallery-image-gallery-3   Started                                1.0s
 ✓ Container image-gallery-image-gallery-1   Started                                0.6s
 ✓ Container image-gallery-image-gallery-2   Started                                0.8s
PS>
PS> cp ./nginx/sites-available/image-gallery.local ./nginx/sites-enabled/
PS>
PS> docker exec nginx-nginx-1 nginx -s reload
2025/03/10 15:57:45 [notice] 26#26: signal process started
PS>
PS> curl -i --head http://image-gallery.local
HTTP/1.1 200 OK
Server: nginx/1.27.4
Date: Mon, 10 Mar 2025 15:57:52 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 790
Connection: keep-alive
X-Proxy: 7a35f8ccd60d
X-Upstream: 172.18.0.8:80

PS> curl -i --head http://image-gallery.local
HTTP/1.1 200 OK
     ...
X-Upstream: 172.18.0.7:80

PS> curl -i --head http://image-gallery.local
HTTP/1.1 200 OK
     ...
X-Upstream: 172.18.0.9:80
```

Add the image gallery config to the enabled sites and reload Nginx

Nginx load-balances requests across the three web containers. The response includes a header with the IP address of the upstream container – repeat the calls and you'll see it change

圖20.5 NGINX負責負載平衡，因此您可以大規模運行應用程序容器。

現在，您可以在單個Docker機器上使用負載平衡運行您的應用程序 - 您無需切換到雲平台或旋轉Kubernetes群集即可以類似生產的配置來測試您的應用程序。 這是驗證功能的簡便方法，否則很難測試，例如在不同實例上訪問共享狀態。 該應用程序也沒有代碼更改或配置更改，這都是由代理人處理的。

到目前為止，我們已經使用NGINX使用不同主機名之間的容器來路由，這就是您在一個環境中運行多個應用程序的方式。您還可以為NGINX路由配置細顆粒路徑，因此，如果您想選擇性地揭示應用程序的一部分，則可以在同一域名中執行此操作。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

Image Gallery應用程序使用REST API，您可以使用HTTP請求路徑配置NGINX來代理API。儘管它實際上來自一個單獨的容器，但API似乎與Web UI是同一應用程序的一部分。

```
＃刪除原始圖像 - 件配置：rm ./nginx/sites-enabled/image-gallery.
local

＃在新的添加API的新版本中復制，然後重新啟動nginx：cp ./nginx/sites-available/image-gallery-2.l
ocal ./nginx/nginx/sites-enets-enabled/image-mimage-allery.local



docker exec nginx-nginx-1 nginx -s重新加載

curl -i http：//image-gallery.local/api/image
```

這是一個非常不錯的模式，用於選擇性地暴露應用程序堆棧的一部分，從同一域名下的許多組件中組裝一個應用程序。 圖20.6顯示了我的輸出 - 響應來自API容器，但客戶端正在使用用於Web UI的同一映像 - 添加劑域上提出請求。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The new site configuration also proxies the API container, at the path `/api/image`. Different paths for the same domain are proxied from different containers

```
PS> rm ./nginx/sites-enabled/image-gallery.local
PS>
PS> cp ./nginx/sites-available/image-gallery-2.local `
>>  ./nginx/sites-enabled/image-gallery.local
PS>
PS> docker exec nginx-nginx-1 nginx -s reload
2025/03/10 16:00:07 [notice] 34#34: signal process started
PS>
PS> curl -i http://image-gallery.local/api/image
HTTP/1.1 200
Server: nginx/1.27.4
Date: Mon, 10 Mar 2025 16:00:13 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Proxy: 7a35f8ccd60d
X-Upstream: 172.18.0.6:80

{"url":"https://apod.nasa.gov/apod/image/2503/California_Mendez_
960.jpg","caption":"NGC 1499: The California Nebula","copyright"
:"\nToni Fabiani Mendez\n"}
```

Nginx makes the API container available on the `image-gallery.local` domain which is also used for the web container

圖20.6 NGINX可以根據域名或請求路徑將請求路由到不同的容器。

負載平衡和路由可讓您接近單個開發人員或測試機上的生產環境，而反向代理負責的另一種基礎架構組件是SSL終止。 如果您的應用程序以HTTPS網站（應為）發布，則需要居住在某個地方的配置和證書，並且最好將其放在您的中央代理中而不是在每個應用程序組件中。 可以使用您從域提供商獲得的真實證書或像Let's Encrypt這樣的服務配置，但是對於非生產環境，您可以創建自己的自我簽名證書並使用它們。

現在嘗試

為Image Gallery應用程序生成SSL證書，並使用證書將其用作HTTPS站點，並通過NGINX代理。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
＃生成自簽名證書 -  Linux容器：Docker容器運行-V“ $（PWD）/nginx/certs：/certs” -e h
ost_name = image -gallery.local diamol/cert -enererator：2e



＃或Windows Server容器：Docker容器運行-V“ $（pwd）/nginx/certs：c：\ certs” -e host_na
me = image -gallery.local diamol/cert -enerator：2e



＃刪除現有的圖像 - 件配置：rm ./nginx/sites-enabled/image-galle
ry.local

＃在新網站配置中使用SSL複製：



＃和RESTART NGINX：DOCKER EXEC NGINX-N
GINX-1 NGINX -S重新加載

＃瀏覽到http：//image-gallery.local
```

這項練習有很多事情。您運行的第一個容器使用OPENSL工具來生成自簽名證書，並將其複製到您的本地證書目錄，該目錄也已綁定到NGINX容器中。 然後，您將圖像庫配置文件替換為使用這些證書和重新加載nginx的配置文件。當您使用HTTP瀏覽網站時，您將被重定向到HTTPS，並且由於不信任自我簽名的證書，因此您會收到瀏覽器警告。在圖20.7中，您可以看到Firefox的警告 - 我可以單擊“高級”按鈕以忽略警告並繼續查看該站點。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

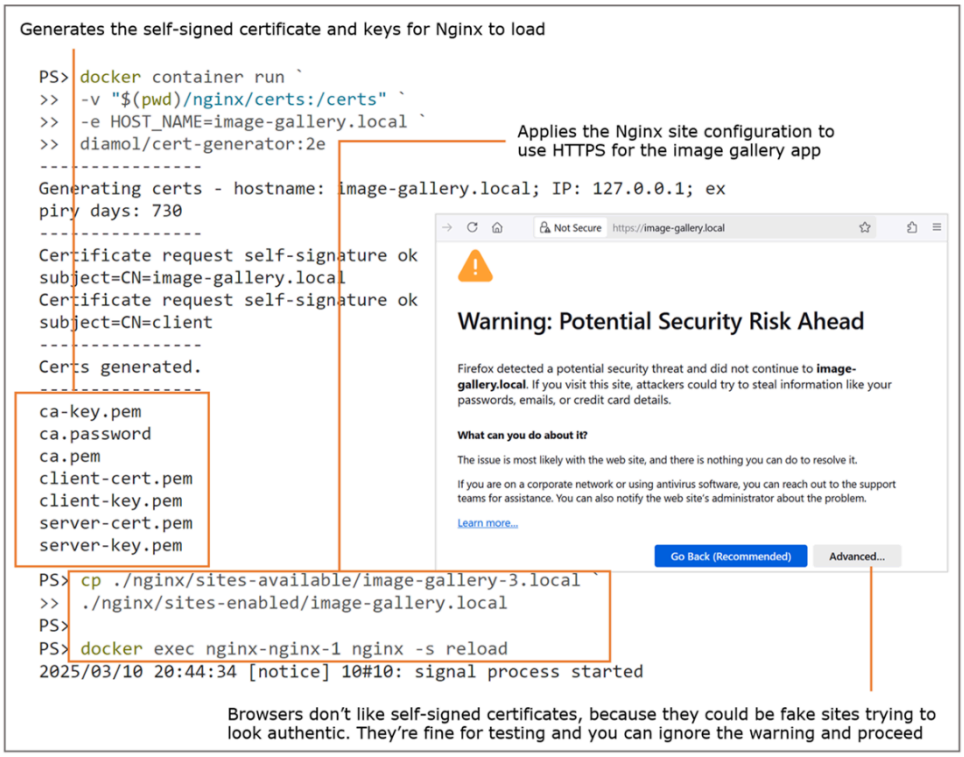


圖20.7 NGINX重定向HTTP請求將HTTPS請求並使用SSL證書提供服務。

NGINX可讓您為您支持的協議和密碼配置各種詳細信息（您可以檢查您的網站並獲取從www.ssllabs。com獲取最佳實踐列表）。我不會詳細介紹所有細節，但是HTTPS設置的核心部分是在列出20.2中 - 您可以在端口80上聆聽HTTP網站，並返回HTTP 301響應，該響應將客戶端重定向到HTTPS網站上的HTTPS網站在端口443上聆聽並使用TLS Spec的最新版本。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Listing 20.2 Serving a site on HTTPS with an HTTP redirect

```
server {
    server_name image-gallery.local;
    listen 80;
    return 301 https://$server_name$request_uri;
}

server {
    server_name  image-gallery.local;
    listen 443 ssl;

    ssl_certificate        /etc/nginx/certs/server-cert.pem;
    ssl_certificate_key    /etc/nginx/certs/server-key.pem;
    ssl_protocols          TLSv1.2 TLSv1.3;
...
```

配置從容器的文件系統加載證書和密鑰文件。 每個證書和密鑰對僅適用於一個域名，因此您將為您使用的每個應用程序提供一組文件（儘管您可以生成一個涵蓋多個子域的證書）。 這些是機密文件，因此在生產中，您將在容器平台中使用秘密來存儲它們。 將HTTPS放在應用程序容器之外意味著更少的配置和證書管理，這是一個基礎架構問題，現在一直存在於代理中 - 開發人員可以旋轉簡單的HTTP版本進行測試。

我們將在此處介紹的Nginx的最後一個功能可以提高性能：上游組件的緩存響應，這是您自己的Web應用程序。

20.3 通過提高性能和可靠性 代理人

NGINX是高性能的HTTP服務器。 您可以使用它為簡單站點或單頁應用程序提供靜態HTML，並且一個容器每秒可以舒適地處理數千個請求。您也可以使用該性能來改進自己的應用程序 - Nginx可以用作緩存代理，因此，當它從應用程序中獲取內容時，它將在其本地磁盤或內存商店中存儲一個副本。隨後的相同內容請求直接從代理中獲得，並且不使用上游。圖20.8顯示了緩存的工作原理。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

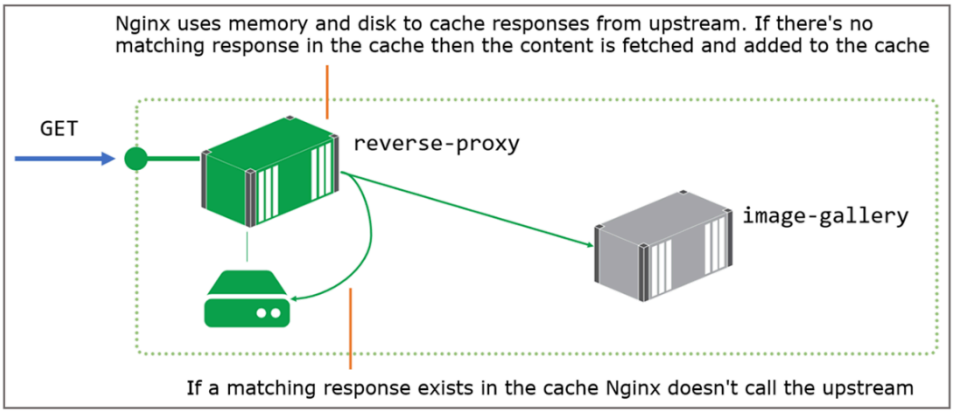


圖20.8使用NGINX作為緩存代理可減少應用程序容器的工作量。

這有兩個好處。 首先，您減少了服務請求所需的時間，因為應用程序平台的生成響應所做的一切必然比NGINX所需的時間更長，而從內存中讀取緩存的響應。其次，您減少了應用程序的總流量，因此您應該能夠處理來自同一基礎架構的更多用戶。您只能緩存不是特定於用戶的內容，但是如果存在身份驗證cookie，則可以像繞過緩存一樣簡單。 諸如Image Gallery應用程序之類的通用站點可以由緩存完全提供。

現在嘗試

將NGINX用作圖像庫應用程序的緩存代理。此配置同時將Web應用程序和API設置為使用NGINX緩存。

```
＃刪除當前網站配置：rm ./nginx/sites-enabled/image-gall
ery.local


＃在緩存配置中復制並重新啟動nginx：cp ./nginx/sites-available/image-gallery-4.local ./nginx/nginx/s
ites-enabled/image-image-aglery.local



docker exec nginx-nginx-1 nginx -s重新加載


＃向網站提出一些請求：curl -i -head  -  Insecure https：//image-galle
ry.local curl -i -head -head -head -insecure https：//image-gallery.local
```

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

新的代理配置設置了一個自定義響應標頭X-CACHE，該標頭X-CACHE在Cache查找的結果中填充了NGINX。如果緩存中沒有匹配（這是您對網站進行的第一個呼叫的情況），則響應標題是X-CACHE：MISS，這意味著緩存中沒有匹配的響應，並且有一個X-Upstream標頭，其中包含nginx的IP地址NGINX的IP地址，其中Nginx獲取了內容。重複呼叫時，響應確實來自緩存，因此您會看到X-CACHE：hit and the X-Upstream標頭，因為Nginx不使用上游。我的輸出在圖20.9中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

This site configuration uses Nginx as a caching proxy for the image gallery app

The cache starts empty, so Nginx fetches content from upstream and adds it to the cache

```
PS> rm ./nginx/sites-enabled/image-gallery.local
PS>
PS> cp ./nginx/sites-available/image-gallery-4.local `
>>  ./nginx/sites-enabled/image-gallery.local
PS>
PS> docker exec nginx-nginx-1 nginx -s reload
2025/03/10 16:05:57 [notice] 50#50: signal process started
PS>
PS> curl -i --head --insecure https://image-gallery.local
HTTP/1.1 200 OK
Server: nginx/1.27.4
Date: Mon, 10 Mar 2025 16:06:03 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 790
Connection: keep-alive
X-Cache: MISS
X-Proxy: 7a35f8ccd60d
X-Upstream: 172.18.0.8:80

PS> curl -i --head --insecure https://image-gallery.local
HTTP/1.1 200 OK
Server: nginx/1.27.4
Date: Mon, 10 Mar 2025 16:06:05 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 790
Connection: keep-alive
X-Cache: HIT
X-Proxy: 7a35f8ccd60d
```

On the second call there's a cache hit for the request. Nginx serves the response directly without using the upstream at all

圖20.9如果代理在其高速緩存中具有響應，則它將其發送而不使用上游。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

nginx讓您可以微調使用緩存。在最新的配置中，我將API設置為使用短壽命的緩存，因此響應在一分鐘後是陳舊的，然後Nginx從API容器中獲取最新內容。對於需要新鮮的內容但您的負載很高的內容，這是一個很好的設置，如果您的API每秒收到5,000個請求，即使是一分鐘的緩存也可以節省300,000個請求，從達到您的API。 Web應用程序設置為使用更長的緩存，因此響應保持新鮮六個小時。 清單20.3顯示了緩存配置。

Listing 20.3 Nginx as a caching reverse proxy for API and web content

```
...
location = /api/image {
    proxy_pass              http://iotd/image;
    proxy_set_header        Host $host;
    proxy_cache             SHORT;
    proxy_cache_valid       200  1m;
    ...
}
location / {
    proxy_pass              http://image-gallery;
    proxy_set_header        Host $host;
    proxy_cache             LONG;
    proxy_cache_valid       200  6h;
    proxy_cache_use_stale   error timeout invalid_header updating
                            http_500 http_502 http_503 http_504;
    ...
}
```

在Diamol/nginx中的核心NGINX配置中定義了名為Long和Short的caches：2e圖像。 緩存規格設置了用於響應的內存和磁盤，以及陳舊項目的驅逐時間。

我不想深入研究NGINX配置，但是您可以使用一個有用的功能來提高應用程序可靠性，該功能是為Proxy_cache_use_stale設置中的Web應用程序定義的。這告訴NGINX，即使上游不可用，它也可以使用緩存的響應。在緩存中的陳舊項目中提供內容意味著您的應用程序在線停留（儘管該應用程序可能無法正常功能），即使應用程序容器已關閉。這是解決應用程序中的瞬態故障或需要向後退回的應用程序推出的非常有用的備份。 您需要仔細考慮可以從緩存成功提供的路徑，但是在一個不錯的簡單演示應用程序中，您可以為整個過程提供服務。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

對圖庫應用程序和API進行幾個電話，以便Nginx將這些響應保存在其緩存中。然後殺死容器，然後嘗試再次請求內容。

```
＃致電網站和API：curl -s -Insecure https：//image-gallery.local curl -s -s
-insecure https https：//image-gallery.local/api/api/image



＃刪除所有Web容器：Docker RM -F $（Docker PS -F名稱= Image -Gallery -image -G
allery_* -Q）


＃再次嘗試Web應用程序：curl -i -head -Insecure https：//image-galle
ry.local


＃卸下API容器：Docker RM -F Image-Gal
lery-iotd-1


＃等待一分鐘左右，然後嘗試API：curl -i- -head -Insecure https https：//image-gall
ery.local/api/image
```

您將在此處看到不同的緩存配置。 Web緩存設置為六個小時後到期，因此，即使沒有Web容器可用，內容也可以從NGINX的緩存中提供。 API響應緩存在一分鐘後到期，並且不設置陳舊的緩存，因此您將從NGINX中獲得HTTP 502錯誤，這意味著它無法到達上游組件。我的輸出在圖20.10中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Fetching content ensures the Nginx cache is populated - the insecure flag is needed because the site is HTTPS with a self-signed certificate

Remove the web containers and Nginx serves the content from its cache

```
PS> curl -s --insecure https://image-gallery.local | Out-Null
PS>
PS> curl -s --insecure https://image-gallery.local/api/image | Out-Null
PS>
PS> docker rm -f $(docker ps -f name=image-gallery-image-gallery_* -q)
cc012464f8f6
5fd47731c368
a07966354106
PS>
PS> curl -i --head --insecure https://image-gallery.local
HTTP/1.1 200 OK
Server: nginx/1.27.4
Date: Mon, 10 Mar 2025 16:08:14 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 790
Connection: keep-alive
X-Cache: HIT
X-Proxy: 7a35f8ccd60d

PS>
PS> docker rm -f image-gallery-iotd-1
image-gallery-iotd-1
PS>
PS> curl -i --head --insecure  https://image-gallery.local/api/image
HTTP/1.1 502 Bad Gateway
Server: nginx/1.27.4
Date: Mon, 10 Mar 2025 16:09:20 GMT
Content-Type: text/html
Content-Length: 157
Connection: keep-alive
```

But the API cache is short-lived and has expired by now – so when I remove the container I get an error instead of the cached response

圖20.10可以對Nginx緩存進行微調，以使內容保持新鮮或為您的應用增加可靠性。

就我們將進行NGINX的練習而言。 這是一個非常有能力的反向代理，您可以做更多的操作 - 例如，啟用HTTP響應的GZIP壓縮並添加客戶端緩存標頭 - 可以改善最終用戶性能並減少應用程序容器的負載。這是一種在容器之前存在的技術，因此實際上並未與容器平台集成；它只是在網絡級別上工作，查找DNS名稱的IP地址，這是Docker提供容器IP地址的地方。它運行良好，但是您需要在每個應用程序上維護每個應用程序的配置文件，並在配置更改時重新加載nginx。

我們將通過查看一種現代替代方案來結束本章，並與Docker完美地集成在一起。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

20.4使用雲原狀反向代理

回到第11章中，我們使用在容器中運行的Jenkins構建了CI管道。 連接到正在運行的Docker發動機的容器，因此可以構建和推動圖像。 將容器連接到Docker Engine還可以使應用程序查詢Docker API以了解其他容器，這正是雲代表反向代理Traefik的功能。 對於要在代理中提供的每個應用程序，沒有靜態配置文件； 相反，您將標籤添加到容器中，而Traefik使用這些標籤來構建自己的配置和路由圖。

動態配置是像Traefik這樣的容器感知代理的主要好處之一。 在運行Traefik之前，您無需啟動上游應用程序，因為它在運行時要注意新容器。 您不必重新啟動Traefik或重新加載配置即可更改應用程序設置，這是應用程序部署的所有部分。 Traefik擁有自己的API和Web UI，可以顯示規則，因此您可以在沒有任何其他容器的情況下運行Traefik，然後部署應用程序，然後查看如何構建配置。

現在嘗試

首先卸下所有現有的容器，然後運行Traefik並檢查UI，以感覺到Traefik如何管理組件。

```
＃刪除NGINX和應用程序容器：Docker容器RM -F $（Docker
Container LS -AQ）

＃啟動Traefik-連接到Linux Docker發動機：Docker Compose -F Traefik/Docker -Compose.yml -F Traefik
/override -linux.yml up -d

＃或使用Windows容器：docker -compose -f traefik/docker -compose.yml -f traefik/override -windows.yml up
-d

＃瀏覽到http：// localhost：8080
```

Linux和Windows有不同的覆蓋文件，因為它們使用不同的私人頻道來連接到Docker引擎。除此之外，在所有平台上，Traefik的行為都是完全相同的。 儀表板是您對Traefik所在的應用程序以及如何配置的應用程序的視圖。您可以在圖20.11中看到Traefik用於配置代理的資源。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

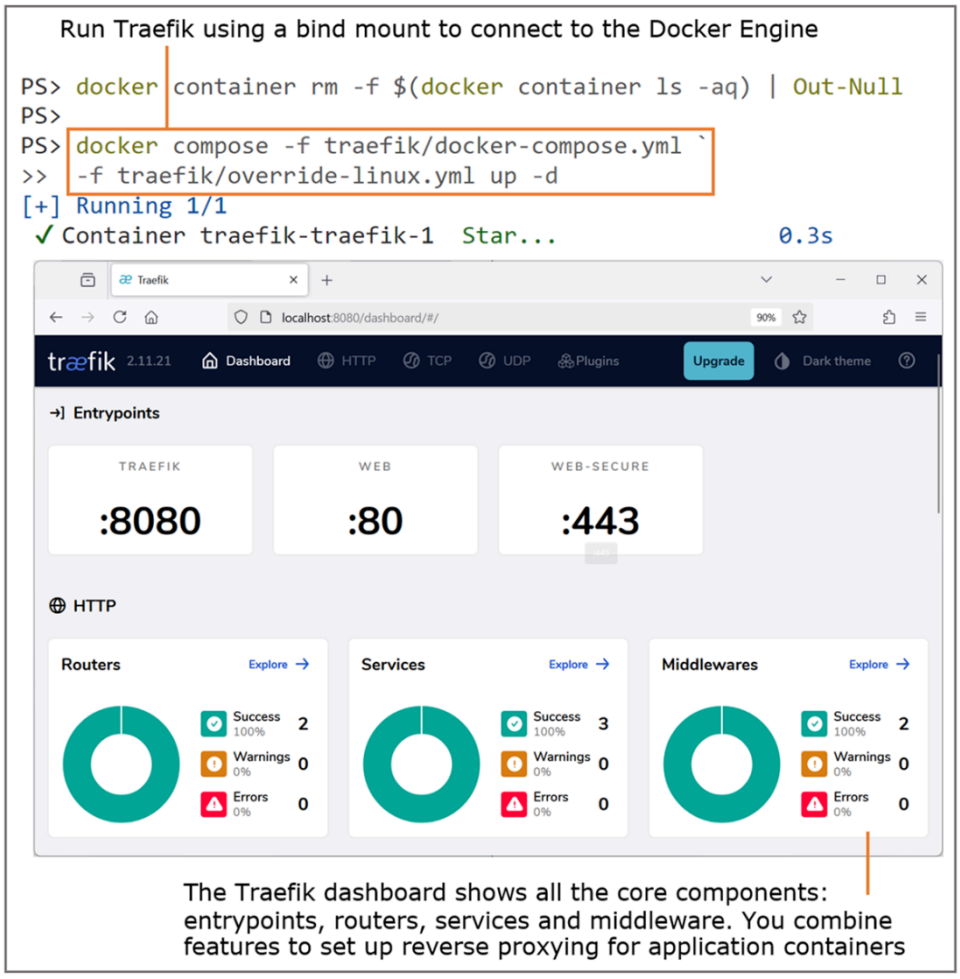


圖20.1 1 Traefik儀表板向您展示了所有應用程序的配置 代理。

Traefik非常廣泛使用，並且具有與Nginx相似的操作模型 - 有一種免費的開源產品，該產品作為Docker Hub中的官方圖像發布，並且如果您想在支持下運行，則有一個商業版本。如果您是反向代理人的新手，我推薦的兩個選擇是Nginx和Traefik； 它將成為基礎架構的主要部分，因此您應該花一些時間比較兩者。讓我們深入研究Traefik的工作方式：

- *Entrypoints* - 這些端口是traefik聆聽外部流量，因此這些映射到已發布的容器的端口。我在HTTP和HTTP中使用了80和443，而Traefik儀表板使用了8080。
- *Routers* - 這些是將傳入請求與目標容器匹配的規則。 HTTP路由器具有諸如主機名和識別客戶端請求的路徑之類的規則。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

- *Services* - 這些是上游組件 - 實際上為Traefik提供內容的應用程序容器，以便它可以將響應傳遞給客戶端。
- *Middlewares* - 這些組件可以在將路由器發送到服務之前從路由器中修改請求。您可以使用中間件組件更改請求路徑或標題，甚至可以執行身份驗證。

最簡單的配置只需要設置一個帶有規則的路由器即可將客戶端請求與路由器所附的服務匹配。

現在嘗試

部署具有更新的撰寫定義的WHO-AM-I應用程序，其中包含標籤，以通過Traefik進行路由。

```
＃在覆蓋文件中使用Traefik標籤部署該應用


＃瀏覽路由器的traefik配置：＃http：// localhost：8080/dashboard/＃/http/routers
/whoami@docker


＃並檢查路由：curl -i http：//whoa
mi.local
```

這是一個簡單的配置 - 該路由僅將入口處端口鏈接到上游服務，即WHO-AM-I容器。您可以在圖20.12中看到Traefik為路由器構建了配置，將主機域Whoami.local鏈接到Whoami服務。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

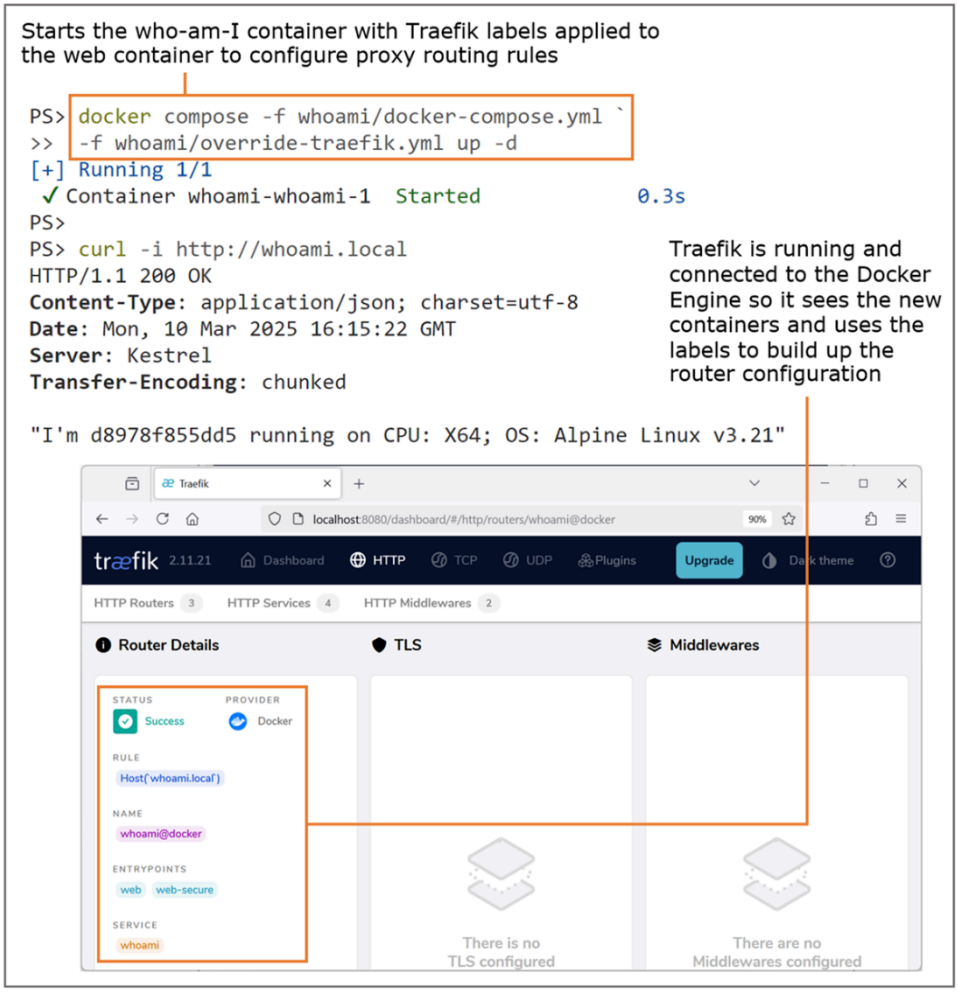


圖20.12 Traefik使用Docker API查找容器和標籤，並使用它們來構建配置。

這是通過在容器上應用兩個標籤來完成的：一個用於啟用該應用程序的traefik，另一個用於指定要匹配的主機名。列表20.4在覆蓋文件中顯示了這些標籤。

清單20.4通過將標籤添加到應用程序容器中配置Traefik

```
服務：
  Jeami：
    標籤：
      - “ traefik.enable = true”
      - “ traefik.http.Routers.whoami.rule = host（`whoami.local.local ）”
      - “ traefik.http.services.whoami.loadbalancer.server.port = 8080”
```

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

Traefik支持一些複雜的路由選項。您可以按主機名和路徑或路徑前綴進行匹配，然後使用中間件組件剝離前綴。 這聽起來很複雜，但這只是我們對圖像庫API所需的需求，因此我們可以將其視為主圖像庫域中的一條路徑。 我們可以配置Traefik以使用“ API”路徑前綴收聽傳入的請求，然後在調用服務之前從請求URL中剝離前綴，因為服務本身不使用該前綴。

現在嘗試

Image Gallery應用程序只需要一個帶有標籤的覆蓋文件，以啟用Traefik支持。部署該應用程序，Traefik將將配置添加到其路由規則中。

```
＃使用新的Traefik標籤啟動應用程序：Docker Compose -F Image-Gallery/Docker-compose.yml -f Image-Gallery/override- traefik.yml up -d


＃檢查Web應用程序：
捲髮 - 頭http：//image-gallery.local

＃和API：
curl -i http：//image-gallery.local/api/image
```

您會在輸出中看到從API調用獲得正確的響應 - Traefik在http：//image-gallery.local/api/image上接收外部請求，並使用路由器和中間件配置在http：// iotd/image上對容器進行內部調用。該配置的配置略有神秘。您可以定義路由器，然後定義中間件組件，然後將中間件連接到路由器上 - 如果要檢查出來，它在文件圖像庫/Override-traefik.yml中。

這種複雜性對消費者都是透明的。您可以在圖20.13中看到響應看起來直接來自API。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

This setup exposes the web container and the API container through Traefik, using the path prefix "api" for API requests

```
PS> docker compose -f image-gallery/docker-compose.yml `
>>  -f image-gallery/override-traefik.yml up -d
[+] Running 3/3
 ✓ Container image-gallery-accesslog-1      Started                    0.5s
 ✓ Container image-gallery-iotd-1           Started                    0.4s
 ✓ Container image-gallery-image-gallery-1  Started                    0.8s
PS>
PS> curl --head http://image-gallery.local
HTTP/1.1 200 OK
Content-Length: 790
Content-Type: text/html; charset=utf-8
Date: Mon, 10 Mar 2025 16:17:26 GMT

PS>
PS> curl -i http://image-gallery.local/api/image
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Mon, 10 Mar 2025 16:17:32 GMT
Transfer-Encoding: chunked

{"url":"https://apod.nasa.gov/apod/image/2503/California_Mendez_960.jpg","c
aption":"NGC 1499: The California Nebula","copyright":"\nToni Fabiani Mende
z\n"}
```

The REST API is available from the same domain as the web app, although the content is fetched from a different container

圖20.13路由規則可讓您在一個域中介紹一個多容器應用程序。

反向代理都有不同的功能集。 Traefik沒有緩存（截至版本2.11），因此，如果您需要緩存代理，則NGINX是必經之路。但是，當涉及到SSL時，Traefik提供了更好的支持 - 它可以與證書提供商開箱即用，因此您可以自動連接以使我們對您進行加密和更新證書。或者，您可以使用默認的自簽名證書提供商，並在沒有任何證書管理的情況下，在非生產環境中添加SSL。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

在圖像庫應用程序和API中添加SSL支持需要更複雜的Traefik設置。它需要在HTTPS入口點以及HTTP上聆聽，但是將HTTP重定向到HTTPS。這一切仍然使用標籤完成，因此部署只是應用程序更新。

```
＃使用https的traefik標籤運行該應用：docker compose -f image-gallery/docker-compose.yml -f image
-gallery/override- traefik-ssl.yml


＃使用https檢查網站：curl--head-Insecure https：//image-gallery.
local


＃和API：
curl  -  Insecure https：//image-gallery.local/api/image
```

如果您瀏覽到網站或API，您會在瀏覽器中看到與使用NGINX使用SSL相同的警告消息 - 證書不受已知證書授權的信任。但是這次我們不需要創建自己的證書並仔細管理證書和密鑰文件，而Traefik則做到了這一切。您可以在圖20.14中看到我的輸出。將捲發與不安全的旗幟一起告訴它，即使證書是不信任的，也要繼續進行。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

This configuration sets up HTTPS with Traefik's default certificate provider - it will generate self-signed certificates for the site

```
PS> docker compose -f image-gallery/docker-compose.yml `
>>  -f image-gallery/override-traefik-ssl.yml up -d
[+] Running 3/3
 ✓ Container image-gallery-accesslog-1      Running                    0.0s
 ✓ Container image-gallery-image-gallery-1  Started                    1.6s
 ✓ Container image-gallery-iotd-1           Started                    0.9s
PS>
PS> curl --head --insecure https://image-gallery.local
HTTP/2 502
content-type: text/plain; charset=utf-8
content-length: 11
date: Mon, 10 Mar 2025 16:19:13 GMT

PS> curl --head --insecure https://image-gallery.local
HTTP/2 200
content-type: text/html; charset=utf-8
date: Mon, 10 Mar 2025 16:19:20 GMT
content-length: 790

PS>
PS> curl --insecure https://image-gallery.local/api/image
{"url":"https://apod.nasa.gov/apod/image/2503/California_Mendez_960.jpg","c
aption":"NGC 1499: The California Nebula","copyright":"\nToni Fabiani Mende
z\n"}
```

The certificate isn't trusted so you need to use curl's `insecure` flag, then you see the web and API content over HTTPS

圖20.14將Traefik用於HTTPS - 它可以生成證書或從第三方提供商那裡獲取證書。

路由，負載平衡和SSL終止是反向代理的主要功能，而Traefik通過通過容器標籤的動態配置來支持它們。如果您對NGINX進行評估，則需要記住，Traefik不會為您提供緩存 - 這是一項備受要求的功能，在以後的版本中可能會帶入Traefik。

我們將嘗試的最後一個功能，在Traefik中很容易，而在Nginx：粘性會話中更加困難。現代應用程序的構建是具有盡可能多的無狀態組件的 - 重要的是，當您按大規模運行時，可以將客戶請求路由到任何容器中，以便您從負載平衡中受益，並在擴展時看到立即的結果。舊的應用程序往往不是由無狀態組件構建的，並且當您將這些應用遷移到希望每次將用戶路由到同一容器的容器中運行時，您可能會發現。這就是所謂的*sticky session*，您可以在Traefik中啟用它，並設置服務。

現在嘗試

Whoami應用程序是粘性會話的一個簡單示例。您可以擴展當前部署並重複呼叫 - Traefik在容器之間將負載平衡。 部署帶有粘性會話的新版本，您的所有請求將由同一容器處理。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
＃運行帶有多個容器的WHO -AM -I應用程序：Docker compose -f whoami/docker -compose.yml -f whoami
/override -traefik.yml up -d  -  scale whoami = 3


＃在容器之間檢查請求是負載平衡的：curl -c c.txt -b c.txt http http：//whoa
mi.local curl -c c.txt -b c.txt http http http：//whoami.local


＃現在用粘性會話支持：Docker compose -f whoami/docker -compose.yml -f whoami/override -traefik -stick
y.yml up -d -cale whoami = 3


＃並檢查請求是否由相同的容器提供：curl -c c.txt -b c.txt http http：//whoa
mi.local curl -c c.txt -b c.txt http http http：//whoami.local
```

啟用粘性會話後，您的請求每次都會由同一容器提供，因為Traefik設置了cookie識別該客戶端應使用的容器（您也會看到瀏覽器的行為相同）。 如果您有興趣，可以檢查瀏覽器會話中的cookie或C.TXT文件中的cookie，並且您會發現Traefik在該cookie中為特定容器提供了標識符。下次打電話時，它將從cookie提取標識符，並使用它將請求路由到同一容器。 我的輸出在圖20.15中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Running without sticky sessions, requests are load-balanced by Traefik between the application containers:

```
PS> docker compose -f whoami/docker-compose.yml `
>>  -f whoami/override-traefik.yml up -d --scale whoami=3
[+] Running 3/3
 ✓ Container whoami-whoami-1  Running                    0.0s
 ✓ Container whoami-whoami-3  Started                    0.5s
 ✓ Container whoami-whoami-2  Started                    0.3s
PS>
PS> curl -c c.txt -b c.txt http://whoami.local
"I'm d8978f855dd5 running on CPU: X64; OS: Alpine Linux v3.21"
PS>
PS> curl -c c.txt -b c.txt http://whoami.local
"I'm 6274fe491c31 running on CPU: X64; OS: Alpine Linux v3.21"
PS>
PS> docker compose -f whoami/docker-compose.yml `
>>  -f whoami/override-traefik-sticky.yml up -d --scale whoami=3
[+] Running 3/3
 ✓ Container whoami-whoami-1  Started                    1.6s
 ✓ Container whoami-whoami-2  Started                    1.2s
 ✓ Container whoami-whoami-3  Started                    1.4s
PS>
PS> curl -c c.txt -b c.txt http://whoami.local
"I'm 0725aa2d2252 running on CPU: X64; OS: Alpine Linux v3.21"
PS>
PS> curl -c c.txt -b c.txt http://whoami.local
"I'm 0725aa2d2252 running on CPU: X64; OS: Alpine Linux v3.21"
```

Traefik uses a cookie to tie the client to one container, so the next client request is served by the same container. curl uses text files to store and present cookies, so this simulates browser behavior

圖20.15在Traefik中啟用粘性會話 - 它使用cookie將客戶端發送到同一容器。

粘性會話是將舊應用程序轉移到容器中的團隊的主要要求之一，而Traefik使其變得非常容易。它與物理服務器或VM的粘性會話不完全相同，因為通常更頻繁地更換容器，因此客戶端可能會粘在不再存在的容器上。如果Cookie將Traefik引導到一個不可用的容器，它將選擇另一個容器，以便用戶會看到響應，但他們的會話將結束。

20.5了解模式反向代理啟用

當您開始在生產中運行許多容器化應用時，反向代理幾乎至關重要。 我們已經介紹了本章中一些更高級的功能 - SSL，緩存和粘性會話 - 但即使沒有這些，您也會發現您遲早需要反向代理。 Kubernetes和雲平台有自己的方式集成反向代理的方式，但是原理是相同的。 反向代理有三種主要模式，我們將通過瀏覽它們來結束。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

第一個是在標準HTTP和HTTPS端口上託管多個Web應用程序，使用客戶端請求中的主機名來獲取正確的內容，如圖20.16所示。

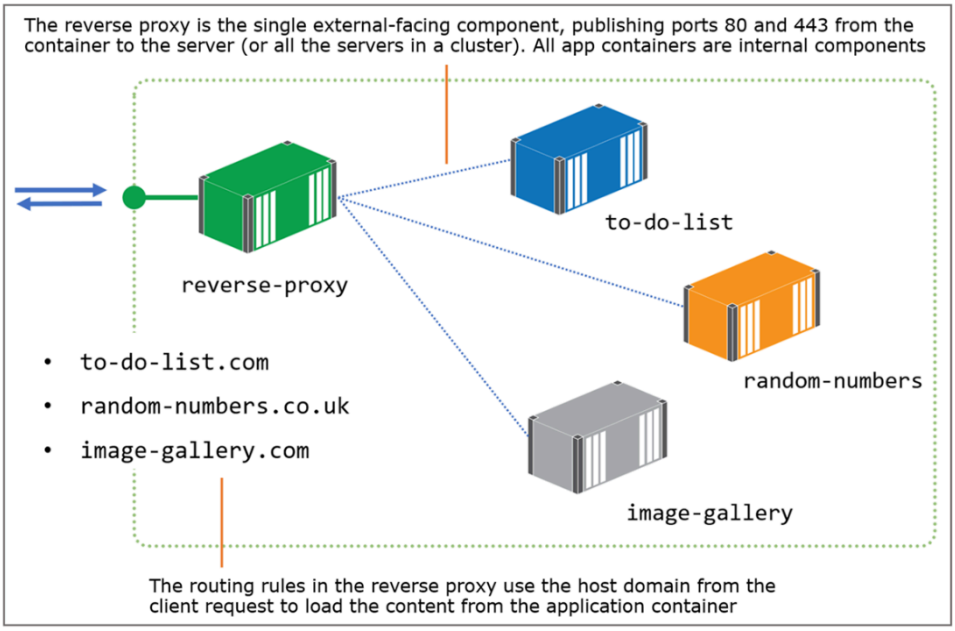


圖20.16使用反向代理在一個集群中託管許多具有不同域名的應用程序。

第二個是用於微服務體系結構，其中一個應用程序跨多個容器運行。 您可以使用反向代理來選擇性地公開由HTTP請求路徑路由的單個微服務。在外部，您的應用程序具有一個域，但是不同的容器為不同的路徑提供了不同的路徑。圖20.17顯示了此模式。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

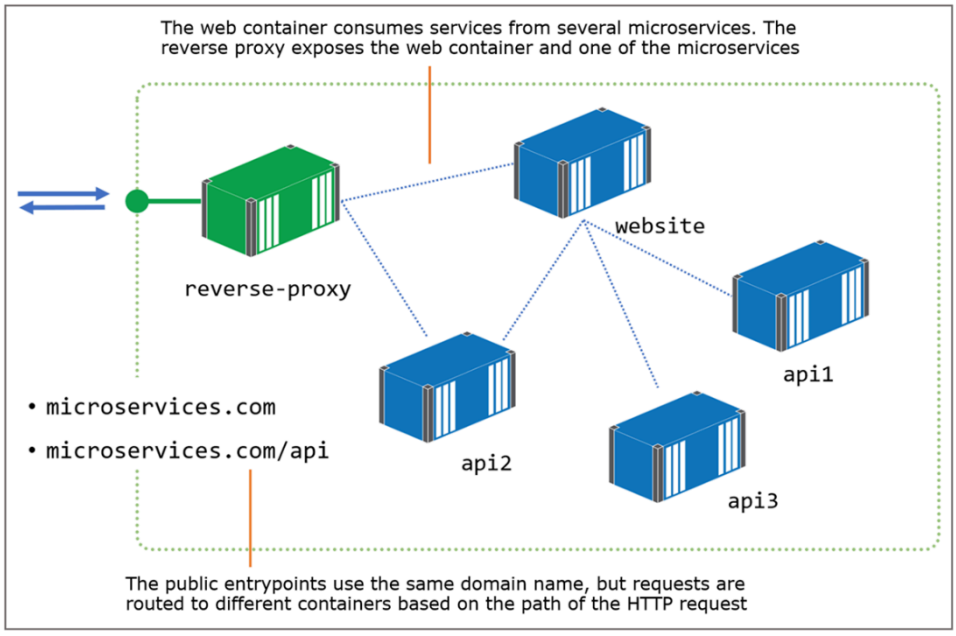


圖20.17反向代理暴露的微服務是同一應用程序域的一部分。

如果您要遷移到容器中，則最終模式非常強大。 您可以使用反向代理開始分解舊應用程序的單片前端，將功能分為新容器。這些新功能由反向代理路由，並且由於它們在不同的容器中，因此可以使用其他現代技術堆棧。圖20.18顯示了這個概念。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

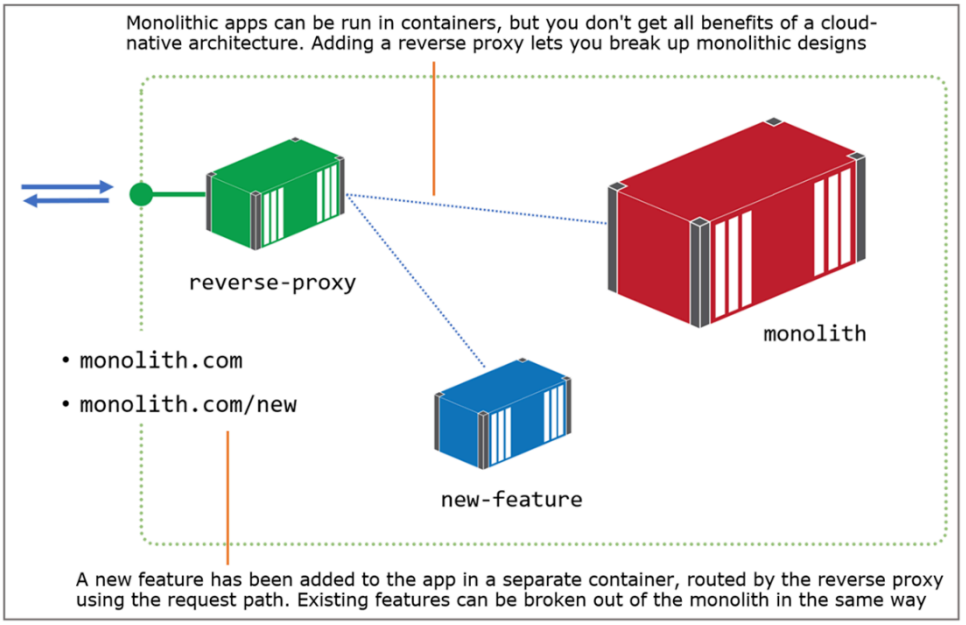


圖20.18 反向代理隱藏了整體建築，因此可以將其分解為S 購物中心服務。

這些模式不是相互排斥的 - 在單個群集中，您可能會有一個反向代理，為所有三種模式提供動力，託管了多種域，其中包括微服務的混合物和在容器中運行的單片應用。

20.6實驗室

我們為該實驗室提供了一個全新的應用程序，該應用程序將清楚地表明緩存反向代理的功能。 這是一個簡單的網站，可將PI計算為指定數量的小數位數。 在本章的實驗室文件夾中，您可以使用Docker組成並瀏覽到http：// localhost：8031/？ dp = 50000，以查看PI的外觀50,000小數點。 刷新瀏覽器，您會發現要計算相同的響應需要盡可能長的時間。您的工作是在反向代理後面運行該應用程序：

- 該應用程序應在標準HTTP端口的域中可用。 代理應緩存響應，以便用戶重複相同的請求時，從緩存提供響應，並且比應用程序要快得多。 代理應增加彈性，因此，如果殺死應用程序容器，則可以從代理中獲得任何緩存的響應。
-

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

我的解決方案是在GitHub上提出的，您會發現，從類似的計算密集型工作中，可以節省大量時間：https：//github.com/sixeyed/diamol/ blob/blob/2e/2e/ch20/ab20/lab/readme.md。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

21 帶有消息隊列的異步通信

這是本書的最後一章，它為系統的組件提供了一種新的方式：使用隊列發送和接收消息。消息隊列已經存在很長時間了 - 它們是一種解耦組件的一種方式，因此他們不會直接連接以相互通信，而是將消息發送到隊列。隊列可以向一個或多個收件人傳遞消息，這為您的架構增添了很多靈活性。

在本章中，我們將重點介紹在您的應用程序中添加消息隊列時啟用的兩個方案：改善系統性能和可擴展性，並添加以零停機時間的添加新功能。我們將使用兩個現代消息隊列在Docker，Redis和Nats中運行得很好。

21.1什麼是異步消息傳遞？

軟件組件通常同步通信 - 客戶端與服務器建立連接，發送請求，等待服務器發送響應，然後關閉連接。 所有使用HTTP連接的REST API，SOAP Web服務和GRPC都是如此。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

*Synchronous*溝通就像撥打電話一樣：它需要雙方同時可用，因此需要仔細管理。 服務器可能會脫機或滿負荷運行，因此無法接受連接。服務可能需要很長時間來處理，客戶連接可能會超時等待響應。 連接可能會在網絡級別上失敗，並且客戶端需要知道他們是否可以安全地重複該請求。 您需要在應用程序代碼或庫中進行大量邏輯來處理所有故障模式。

*Asynchronous*通信在客戶端和服務器之間添加了一層。 如果客戶端需要服務器來做某事，它將向隊列發送消息。 服務器在隊列上偵聽，拿起消息，然後處理消息。服務器可以將響應消息發送到隊列，如果客戶端想要響應，它將在隊列上偵聽並將其撿起。異步消息就像通過電子郵件交流：當事人可以在空閒時間時加入。 如果服務器脫機或失效，則該消息位於隊列中，直到可以使用服務器將其撿起為止。 如果消息需要很長時間來處理，則不會影響客戶或隊列。如果客戶發送消息時發生故障，則該消息不在隊列中，客戶可以安全地發送。圖21.1顯示了與異步消息傳遞的通信。

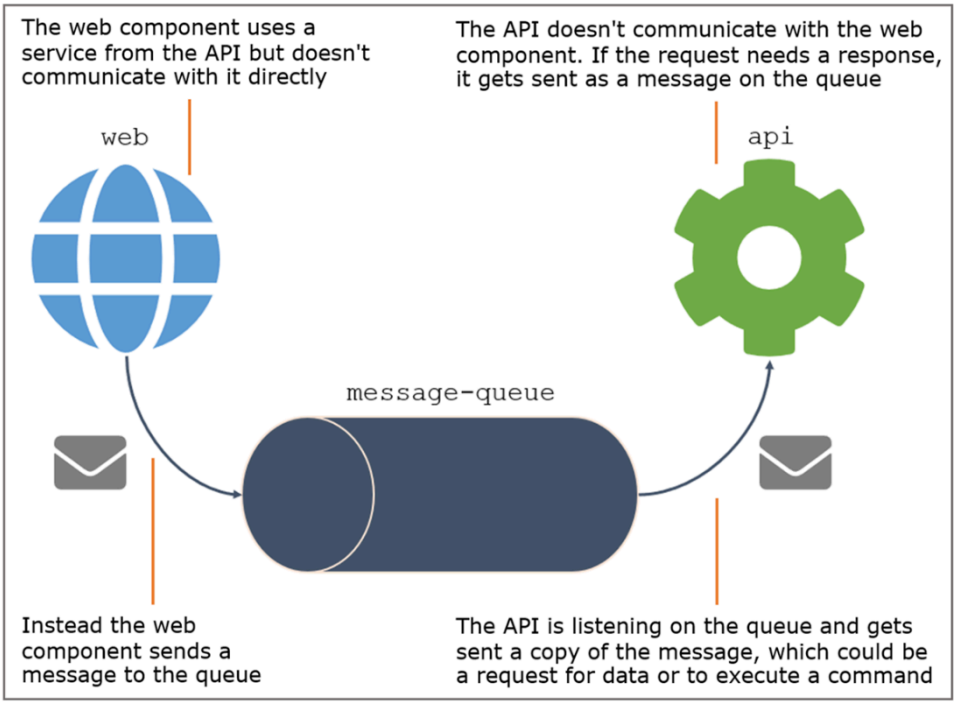


圖21.1消息隊列將組件分解，以使它們不會直接互動。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

消息傳遞一直是集成架構師的最愛選擇，但是它用來提出一些困難的問題 - 隊列技術需要超級可靠，但是企業隊列在測試環境中運行太貴了，所以我們可以在不同的環境中使用不同的隊列或在開發人員中完全跳過隊列嗎？ Docker對此進行了修復，從而可以輕鬆地將企業級開源隊列添加到您的應用程序中。 輕巧的容器中運行隊列意味著您可以為每個應用程序運行專用隊列，免費的開源軟件意味著您可以在每個環境中使用相同的技術。 REDIS是一個流行的消息隊列選項（您也可以用作數據存儲），您可以輕鬆嘗試以感覺到異步消息傳遞。

現在嘗試

將REDIS服務器運行到一個容器中，連接到網絡，您可以在其中運行其他容器以發送和接收消息：

```
＃創建網絡 - 在Linux容器上：Docker網絡創建CH21


＃或Windows Server容器：
Docker網絡創建-D NAT CH21

＃運行Redis服務器：
Docker容器運行-D -NAME REDIS -NETWORK CH21 DIAMOL/REDIS：2E

＃檢查服務器是否正在偵聽：Docker容器日誌
Redis -Tail 1
```

消息隊列是服務器組件，直到您停止它們為止。 REDIS在端口6379上聆聽連接，客戶使用相同的地址來發送消息和服務器來偵聽消息。從容器日誌中，您可以看到Redis啟動容器後僅幾秒鐘即可運行，我的輸出在圖21.2中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
PS> docker network create ch21
e94a95cd8bf5bc22bcb881125a7d8177c5b6476487c02bb60bea717edbba0bd3
PS>
PS> docker container run -d --name redis --network ch21 diamol/redis:2e
858b0000a36d00bef3a915eefe54b89dee89ac9fe91a6edf2b7b9dab81b1ed99
PS>
PS> docker container logs redis --tail 1
1:M 11 Mar 2025 10:28:05.724 * Ready to accept connections tcp
```

Run a Redis container - no ports are published because the queue will only be used by other containers

Redis writes out logs when it starts, ending with a confirmation that it's ready for clients

圖21.2消息隊列就像其他任何背景容器一樣，等待連接。

客戶需要打開與隊列的連接以發送他們的消息，並且如果您想知道這比直接致電REST API更好，那麼這一切都會降低。隊列通常具有高度優化的自定義通信協議，因此，當客戶端發送消息時，它只是傳輸請求的字節，並等待已收到已收到的確認。隊列在消息上沒有進行任何復雜的處理，因此他們每秒應輕鬆處理數千條消息。

現在嘗試

我們不會發送數千個請求，但我們將使用Redis CLI發送一些消息。 命令語法有點涉及，但這將在名為Channel21的通道上發布消息“ ping”，它將以5秒的間隔重複該消息50次。

```
＃在後台運行Redis客戶端以發布消息：Docker Run -D -NAME Publisher -network CH21 Diamol/red
is -cli：2e -r 50 -i 5

＃檢查日誌要查看消息正在發送：Docker Logs Publisher
```

此Redis客戶端容器將坐在後台，並每五秒鐘發送消息。 日誌輸出僅顯示每個消息發送的響應代碼，因此，如果一切正常，您會看到很多零，這是“確定”響應。 您可以在圖21.3中看到我的。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

The Redis CLI uses the Redis messaging protocol. This container runs a command which sends 50 messages with a 5 second wait between each one

```
PS> docker run -d --name publisher --network ch21 `
>>  diamol/redis-cli:2e `
>>  -r 50 -i 5 PUBLISH channel21 ping
866a3613ed7fa95ec79b382bcbb3c0be3068003c9a38a76b662ce49aea455c3e
PS>
PS> docker logs publisher
0
0
```

There's not much to see - the output is the acknowledgement response code from the CLI for each message send. Zero means OK

圖21.3 Redis CLI是將消息發送到在運行的隊列中的一種簡單方法 REDIS容器。

這裡有一些新的術語，因為*client*和*server*在消息方面沒有任何意義。 每個組件都是消息隊列的客戶端；他們只是以不同的方式使用它。 發送消息的組件是*publisher*，並且接收消息的組件是*subscriber*。 使用隊列可能會有許多不同的系統，因此Redis使用*channels*將消息分開。 在這種情況下，發布者正在在名為Channel21的通道上發送消息，因此要使組件讀取這些消息，它需要訂閱同一頻道。

現在嘗試

使用Redis CLI運行另一個容器，這次訂閱了其他容器正在發布消息的頻道：

```
＃運行一個交互式訂戶，您將看到每五秒鐘收到的＃消息：docker run -it-network ch21  - 名稱訂戶d
iamol/redis -cli：2e subscribe channel21
```

我們正在使用Redis CLI，這是一個使用REDIS消息傳遞協議進行對話的簡單客戶端 - 所有主要應用程序平台都有REDIS SDK，因此您也可以將其與您自己的應用程序集成在一起。 CLI在多行上打印輸出，因此您將首先看到從訂閱到隊列的輸出。 發布容器仍在背景中運行，每次發布消息時，Redis都會將副本發送到訂戶容器，然後您會在日誌中看到消息詳細信息。我的位於圖21.4中。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

This is an interactive container running a subscribe command in the Redis CLI. The initial output confirms the subscription to the channel

```
PS> docker run -it --network ch21 --name subscriber `
>>  diamol/redis-cli:2e `
>>  SUBSCRIBE channel21
1) "subscribe"
2) "channel21"
3) (integer) 1
1) "message"
2) "channel21"
3) "ping"
```

Every time the publishing container sends a message to Redis, it gets delivered to this container because it's subscribing to messages on the same channel. The output shows a message has been received on channel 21 with the content "ping"

圖21.4隊列的訂閱者接收到頻道上發布的每個消息的副本。

您可以使用CTRL -C從容器中退出，也可以使用Docker RM -F用戶殺死容器。 在此之前，它將不斷聆聽消息。 您可以看到這是異步通信：發布者在任何訂閱者都在收聽之前發送消息，即使沒有發布者，訂戶也會繼續聆聽消息。 每個組件與消息隊列一起使用，並且不知道發送或接收消息的其他組件。

將發件人和接收器與隊列解耦的簡單原則有助於您使應用程序的性能和可擴展性，您將在接下來看到一個新版本的待辦事項列表應用程序。

21.2使用雲本地消息隊列

待辦事項應用程序具有Web Frontend和一個用於存儲的SQL數據庫。在原始實現中，組件之間的所有通信都是同步的 - 當Web應用程序發送查詢或插入數據時，它將打開與數據庫的連接，並保持打開狀態，直到請求完成。 該體系結構的擴展不是很好。 我們可以運行數百個Web容器來支持高用戶負載，但是最終我們在使用所有可用數據庫連接的地方達到了一個限制，並且該應用程序將開始失敗。

這是消息隊列幫助性能和規模的地方。 待辦事項應用的新版本使用異步消息傳遞用於保存工作流程 - 用戶添加新的待辦事項時，Web應用程序將在隊列上發布消息。 隊列可以處理比數據庫更多的連接，並且連接的壽命要短得多，因此即使在很高的用戶負載下，隊列也不會最大化。 我們將使用不同的隊列技術進行此練習：NAT，這是一個雲本機計算基礎（CNCF）項目，已成熟且廣泛使用。它將消息存儲在內存中，因此非常快速，非常適合容器之間的通信。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

在容器中運行NAT。 它具有一個簡單的管理API，您可以用來查看有多少客戶連接到隊列。

```
# switch to the exercise folder: cd ch21/exercises/todo-list # star
t the message queue: docker compose up -d message-queue # ch
eck the logs: docker container logs todo-list-message-queue-1 #
and check active connections: curl http://localhost:8222/connz
```

The connections API call returns JSON details about the number of active connections. There could be many thousands so the response is paged, but in this case there’s only one page of data because there are zero connections. You can see my output in figure 21.5.

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

The Compose file defines the whole to-do list app. This command just runs the NATS message queue

NATS listens on different ports for clients, other NATS servers and for the admin API. The Compose file publishes API port 8222

```
PS> cd ch21/exercises/todo-list
PS>
PS> docker compose up -d message-queue
[+] Running 2/2
 ✓ Network todo-list_app-net            Created                                  0.0s
 ✓ Container todo-list-message-queue-1  Started                                  0.3s
PS>
PS> docker container logs todo-list-message-queue-1
[1] 2025/03/11 10:32:33.609791 [INF] Starting nats-server
      ...
[1] 2025/03/11 10:32:33.610773 [INF] Listening for client connections on 0.0.0.0:4222
[1] 2025/03/11 10:32:33.610935 [INF] Server is ready
[1] 2025/03/11 10:32:33.611084 [INF] Cluster name is my_cluster
[1] 2025/03/11 10:32:33.611184 [INF] Listening for route connections on 0.0.0.0:6222
PS>
PS> curl http://localhost:8222/connz
{
  "server_id": "NCHXQSR2ZZ3EN4Z57MMPZTAME7NQMOH3MJ72ZF2N7HNV5WRCPPJSCLCX",
  "now": "2025-03-11T10:32:48.025223202Z",
  "num_connections": 0,
  "total": 0,
  "offset": 0,
  "limit": 1024,
  "connections": []
}
```

The admin API includes a connection count, showing how many clients are connected to the queue - right now there are none

數字　21.5 Nats是一個替代消息隊列，很輕巧，有一個廣告　　最小。

當您轉到異步消息傳遞以及待辦事項應用程序時，涉及開發工作，這意味著對Web應用程序進行了一些更改。現在，當用戶添加待辦事項時，Web應用程序會向NAT發布消息，而不是在數據庫中插入數據。 更改實際上很小。即使您不熟悉.NET Core，也可以在列表21.1中看到發布消息涉及的工作並不多。

列出21.1發布消息而不是將數據寫入數據庫

```
公共無效ADDITE（一切）
{MessageQueue.publish（new newitemevent（all））；　_newTa
skScounter.inc（）；}
```

NAT不使用與Redis相同的頻道概念。 相反，每個消息都有一個*subject*，它是用於標識消息類型的字符串。 您可以為消息主題選擇自己的命名方案。 這個主題是events.todo.newitem，它說這是待辦事項應用程序中的新項目。如果訂戶對新項目事件感興趣，他們將能夠聽取該主題的消息，但是即使沒有訂戶，該應用程序仍然會發布消息。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

運行待辦事項Web應用程序和數據庫的新版本。您會看到該應用程序加載，並且可以使用它而沒有任何錯誤，但是它不能完全正確。

```
＃啟動網絡和數據庫容器：Docker組成-d todo-web
todo-db

＃瀏覽到http：// localhost：8080並添加一些項目
```

您會發現該應用程序可以讓您添加新項目，但是當您瀏覽列表時，沒有。這是因為列表頁面從數據庫中獲取數據，但是新項目頁面不再將數據插入數據庫。 新項目事件消息已發布給NATS消息隊列，但什麼都沒有聽。 您可以在圖21.6中看到我的空待辦事項清單（根本不代表現實生活中的代表）。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

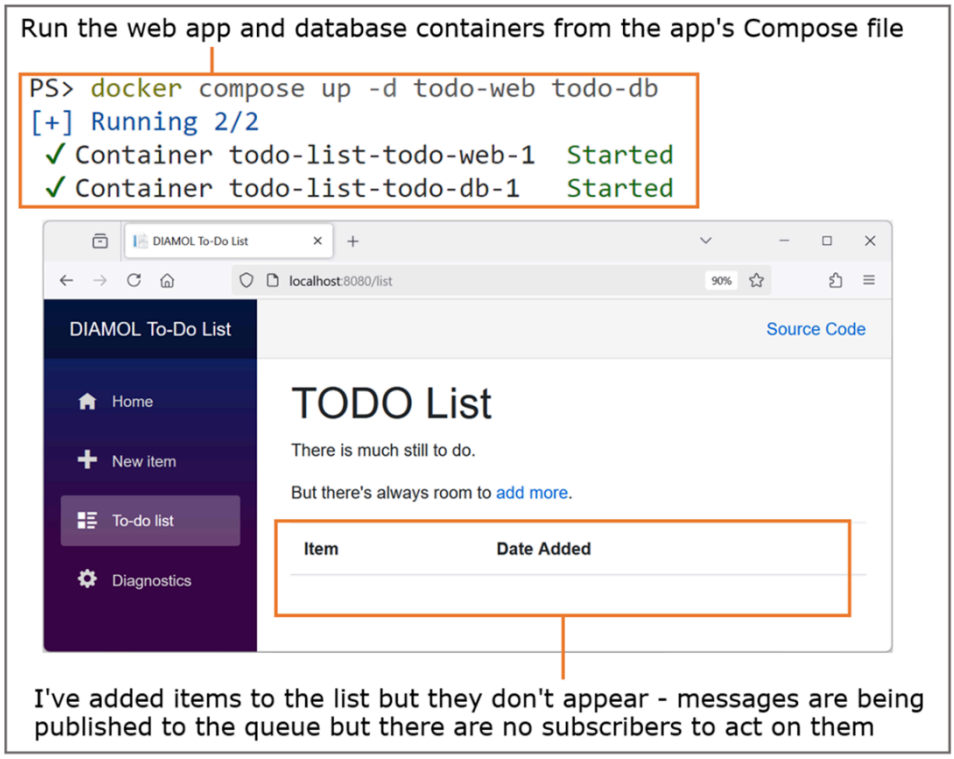


圖21.6帶有消息發布的待辦事項應用；沒有任何訂戶，就有缺少的功能。

有很多消息隊列技術，採用不同的方法來處理這種情況 - 在其中發布了消息，但沒有訂戶。 有些隊列將它們移至*dead-letter queues*供管理員進行管理，而另一些隊列將消息存儲，以便在客戶端連接和訂閱時可以交付它們。 Redis和Nats有效地吞下了這些消息 - 他們承認收據給客戶，但是無處可發送它們，因此它們被丟棄了。 redis或Nats排隊的新訂閱者僅接收*are*發布*after*的消息，他們開始偵聽。

現在嘗試

該項目在GitHub上的示例中有一個簡單的NATS訂戶工具。 您可以使用它來收聽具有特定主題的消息，因此我們可以檢查以查看實際上正在發布待辦事項事件：

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

```
#運行訂戶收聽“ events.todo.newitem”消息docker run -d-name todo-sub -network todo-list_app-net
diamol/nats-sub：2e Events.todo.newitem



#檢查訂戶日誌：docker登錄todo-s
ub


#瀏覽到http：// localhost：8080並添加一些新項目


#檢查新項目事件是否已發布：docker登錄todo-sub
```

用戶體驗完全相同 - Web應用程序仍然不起作用。 它發布消息，不知道他們會發生什麼，但是現在有一個訂戶收到每個消息的副本。如果您在網站中輸入一些待辦事項，則會看到它們在訂戶的日誌中列出，如圖21.7所示。

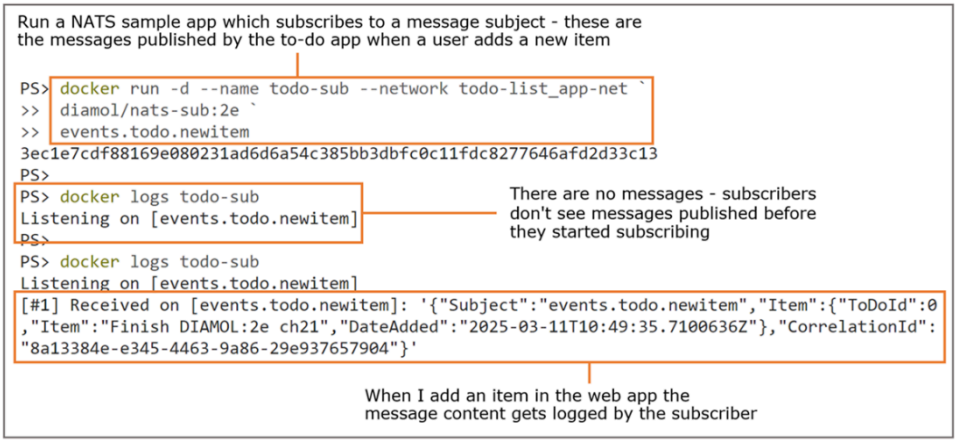


圖21.7記錄消息的簡單訂戶是驗證它們正在發布的好方法。

您現在已經意識到，待辦事項應用程序缺乏對正在發布的消息的組件。 您需要做三項工作來移動異步消息傳遞：運行消息隊列，在有趣的事件發生時發布消息並訂閱這些消息，以便在事件發生時可以做一些工作。待辦事項應用程序缺少最後一部分，我們將添加下一個。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

21.3消費和處理消息

訂閱隊列的組件稱為*message handler,*，通常您會為每種消息（Redis中的每個通道或NATS中的主題）提供一個處理程序。待辦事項應用需要一個消息處理程序，該消息處理程序會聆聽新項目事件並將數據插入數據庫中。圖21.8顯示了完整的體系結構。

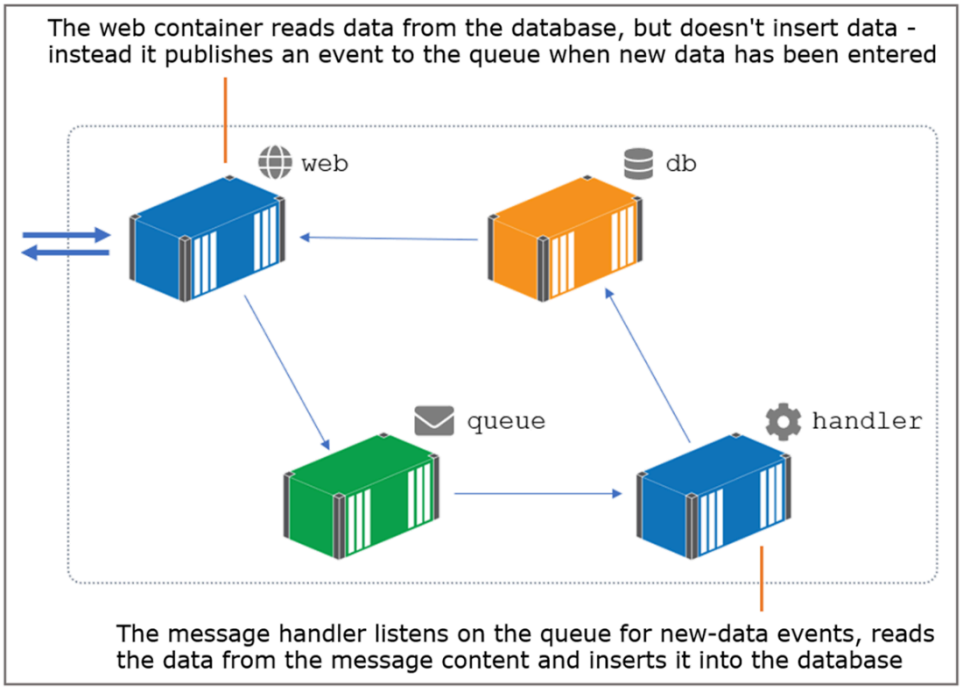


圖21.8異步處理使用消息處理程序以及事件發布者。

該設計確實會縮放，因為隊列的作用就像一個緩衝區，從傳入的用戶負載中弄平了任何峰值。您可以擁有數百個Web容器，但只有10個消息處理程序容器 - 處理程序在一個組中，因此隊列共享郵件周圍的消息，每個消息都由一個容器處理。容器一次處理消息，因此，這將限制用於將數據插入數據的最大SQL連接數量到10，無論有數千個用戶都瘋狂單擊按鈕。如果收入的負載超過那10個處理程序的負擔，則消息將保存在隊列中，直到處理人員準備好進行更多處理。該應用程序繼續工作，並且數據最終被保存（這聽起來很可怕，但通常意味著幾秒鐘內）。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

待辦事項應用程序的消息處理程序已經構建並發布給Docker Hub，因此可以使用。現在運行它，看看該應用如何與ASYNC消息一起工作：

```
#啟動消息處理程序：Docker組成-D Save
Handler

#從容器日誌中檢查連接：docker logs todo-list-save handler-
1

#瀏覽到http：// localhost：8080並添加一些新項目

#檢查事件是否已處理：Docker登錄todo-list-save han
dler-1
```

該應用程序再次工作！幾乎。您會發現可以添加新項目，並且它們出現在列表頁面中，但不能立即出現。保存項目時，Web應用程序將重定向到列表頁面，該頁面在消息仍在通過隊列和處理程序的過程中加載。當查詢在數據庫上運行時，新項目尚未保存，因此未顯示新數據。您可以在圖21.9中看到我的輸出 - 在這一點上，即使保存了一個新項目，我的網頁也沒有顯示任何項目。

圖21.9我 SSAGE處理者訂閱隊列，接收每個消息的副本， 並採取行動。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

這是稱為*eventual consistency*的異步消息傳遞的副作用 - 當處理所有消息時，應用程序數據的狀態將是正確的，但是在此之前，您可能會獲得不一致的結果。有一些方法可以解決此操作，可以使整個UI異步進行，因此待辦事項網絡應用程序會聆聽一個事件，說明列表已經更改，然後刷新本身。這種推動模型比投票查詢要高得多，但這對本書來說太多了。我們現在可以刷新刷新。

轉向異步消息傳遞是一個相當大的建築變化，但它打開了很多機會，因此絕對值得知道這是如何工作的。消息處理程序是小的，集中的組件，可以獨立於主應用程序或彼此獨立進行更新或縮放。 在此練習中，我們已經使用隊列解決了一個擴展問題，現在我們可以運行保存消息處理程序的多個實例來通過輸入負載來工作，同時有效地限制了我們使用的SQL連接數量。

現在嘗試

消息處理程序是內部組件；他們不在任何端口上收聽，因此您可以在單台計算機上使用多個容器按比例地運行它們。如果有幾個實例，則NAT支持負載平衡以共享消息。

```
＃擴展處理程序：Docker組成-d -scale Save Handler = 3


＃檢查一個新處理程序之一已連接：Docker登錄todo-list-save han
dler-2


＃瀏覽到http：// localhost：8080並添加一些新項目


＃查看哪些處理程序已經處理了消息：docker撰寫日誌 -  -tail
= 1 Save Handler
```

您會看到消息發送到不同的容器。 NAT使用圓形旋轉負載平衡來共享連接的訂戶之間的負載，您會發現所承受的負載越多，分佈越均勻。 我在圖21.10中的輸出表明，容器1和2具有處理的消息，但沒有容器3。

© Manning Publications Co. To comment go to liveBook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

Async processing can be scaled just by adding more containers- the NATS queue distributes the messages across all connected message handlers

```
PS> docker compose up -d --scale save-handler=3
[+] Running 6/6
 ✓ Container todo-list-todo-db-1          Running                0.0s
 ✓ Container todo-list-save-handler-1     Runnin...              0.0s
 ✓ Container todo-list-save-handler-3     Starte...              0.3s
 ✓ Container todo-list-message-queue-1    Runni...               0.0s
 ✓ Container todo-list-save-handler-2     Starte...              0.5s
 ✓ Container todo-list-todo-web-1         Running                0.0s
PS>
PS> docker logs todo-list-save-handler-2
Connecting to message queue url: nats://message-queue:4222
Listening on subject: events.todo.newitem, queue: save-handler
PS>
PS> docker compose logs --tail=1 save-handler
save-handler-1  | Item saved; ID: 3; event ID: df941852-16c5-4fdb-a3e4-647e997a9f15
save-handler-3  | Listening on subject: events.todo.newitem, queue: save-handler
save-handler-2  | Item saved; ID: 2; event ID: 8da694c0-fc54-4ab3-af59-bbe79e862fe5
```

After adding some more items in the web UI - they all get published as events to the queue, but different handlers pick them up and process them

圖21.10多個消息處理程序共享工作負載，以便您可以擴展以滿足需求。

重要的是要意識到，我沒有更改任何東西來獲得我的新項目功能的三倍 - 網站和消息處理程序代碼完全相同。我只是運行了更多的Message Handler容器的實例。如果您具有另一個由同一事件觸發的功能，則可以運行一個訂閱同一消息主題的其他消息處理程序。 這打開了能夠在不更改現有代碼的情況下將新功能部署到應用程序的有趣選擇。

21.4添加帶有消息句柄的新功能　　盧比

我們已經將待辦事項應用程序移至*event-driven architecture*，該應用程序是應用程序發布事件的設計，以說事情發生了，而不是在發生所有事件時就處理所有內容。這是構建鬆散耦合應用程序的好方法，因為您可以在不更改發布事件的邏輯的情況下更改對事件的響應發生的事情。我們只是在此應用程序中將其用於單一類型的事件，但這仍然具有添加新功能而無需更改現有應用程序的靈活性。

最簡單的方法是在一個新的組中添加一個新的消息處理程序，該組件獲得每個事件的副本，但在響應中會做一些不同的事情。 現有消息處理程序將數據保存在SQL數據庫中；一個新的消息處理程序可以保存Elasticsearch中的數據，以使用戶可以輕鬆在Kibana查詢，或者可以將項目添加為Google日曆中的提醒。 我們為下一個練習提供了一個更簡單的示例 - 一個像審核步道一樣工作的處理程序，為每個新項目編寫日誌條目。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在嘗試

新消息處理程序位於撰寫的覆蓋文件中。 部署它時，您會發現這是一個加性部署。 撰寫創建一個新容器，但其他容器都沒有更改。

```
＃運行審核消息處理程序，保留相同的比例＃為保存處理程序：Docker Compose -f Docker-compose.
yml -f docker-compose-audit.yml up -d-cale save handler = 3



＃檢查審計處理程序是否正在聽：Docker登錄todo-list-a
udit handler-1


＃瀏覽到http：// localhost：8080並添加一些新項目


＃檢查審核步道：Docker登錄todo-list-audit han
dler-1
```

這是一個零下的部署； 原始的應用程序容器沒有變化，新功能將在新容器中實現。審計處理程序訂閱了與保存處理程序相同的消息主題，因此它獲取了每個消息的副本，而該消息的另一副本則將發送到一個保存處理程序容器之一。您可以在圖21.11中看到我的輸出，其中審計處理程序寫下了待辦事項日期和文本。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Run the new auditing message handler - all the other containers are up-to-date so there are no changes to those, just a new container for the new feature

```
PS> docker compose -f docker-compose.yml `
>>  -f docker-compose-audit.yml `
>>  up -d --scale save-handler=3
[+] Running 7/7
 ✓ Container todo-list-audit-handler-1  Start...        0.3s
 ✓ Container todo-list-todo-web-1       Running         0.0s
 ✓ Container todo-list-save-handler-1   Runnin...       0.0s
 ✓ Container todo-list-save-handler-2   Runnin...       0.0s
 ✓ Container todo-list-todo-db-1        Running         0.0s
 ✓ Container todo-list-save-handler-3   Runnin...       0.0s
 ✓ Container todo-list-message-queue-1  Runni...        0.0s
PS>
PS> docker logs todo-list-audit-handler-1
Connecting to message queue url: nats://message-queue:4222
Listening on subject: events.todo.newitem, queue: audit-handler
PS>
PS># add new item in the web UI
PS>
PS> docker logs todo-list-audit-handler-1 --tail 1
AUDIT @ 03/11/2025 10:55:49: Promote DIAMOL:2e on social
```

The new feature just lists all the new items - this container gets a copy of each event message, and the original save handlers still get a copy between them

圖21.11發布事件將應用程序的組件分發，並讓您添加新功能。

現在，兩個過程是針對創建待辦事項項目的用戶觸發的，並且它們都在在單獨的容器中運行的單獨組件中進行操作。這些過程可能會花費任何時間，並且不會影響用戶體驗，因為Web UI不會等待它們（甚至了解它們） - 它只會將事件發佈到隊列中，無論有多少訂戶正在聆聽，該行為都具有相同的延遲。

從這個簡單的示例中，您應該了解這種體系結構的強大程度。一旦您的應用將關鍵事件作為給隊列的消息發布，則可以構建全新功能而無需觸摸現有組件。 新功能可以獨立構建和測試，並且可以對運行應用程序沒有影響而進行部署。如果該功能有問題，則可以通過停止消息處理程序來掩蓋它。

我們將研究本章的最後一項練習，以幫助您說服您，ASYNC消息傳遞是您應該考慮的應用程序的模式。 我們可以為某種類型的事件提供多個訂戶，但是我們也可以擁有多個發布者。 新項目事件是代碼中的固定結構，因此任何組件都可以發布該事件，這為我們提供了創建待辦事項項目的新選項。 我們將使用它為應用程序部署REST API而不更改任何現有零件。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

現在嘗試

待辦事項列表API已編寫並準備部署。它會在端口8081上聽，並在用戶製作HTTP Post Requeues時發布新項目事件。

```
＃啟動API容器，在覆蓋文件中定義：docker compose -f docker-compose.yml -f docker-compose-a
udit.yml -f docker- docker- compose-api.yml -d todo-api



＃通過API添加新項目：curl http：// localhost：8081/todo -d'{“ item”：“ record promo video”}'}'-
H'content- type-類型：application/json'



＃檢查審核日誌：Docker登錄todo-list-audit han
dler-1
```

新的API是一台簡單的HTTP服務器，其中唯一的真正邏輯是使用列表21.1中的同一Message-Quale方法向隊列發布事件。您會看到通過API輸入的新項目由審核處理程序和保存處理程序處理，因此有審核條目，當您刷新Web應用程序時，您會看到新項目在數據庫中。我的輸出在圖21.12中。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

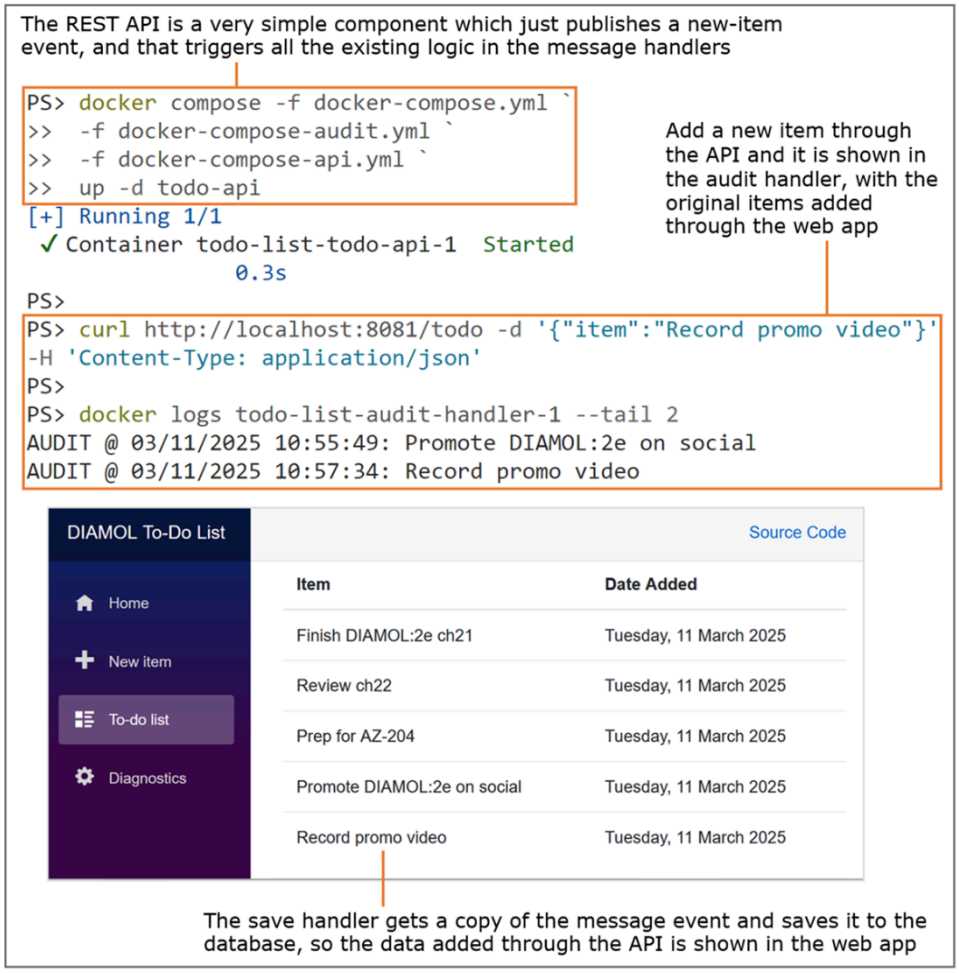


圖21.12 ev Ent可以有許多訂戶和許多發行商，這使得 OOSE耦合。

這是強大的東西，這全都來自應用程序中發布的單個事件。 異步消息傳遞使您能夠構建更易於擴展和更新的更靈活的應用程序，並且您也可以將所有這些好處添加到現有應用程序中，首先要從幾個關鍵事件開始，然後從那裡構建。

在您前往白板之前，我們將仔細查看消息傳遞模式，以便您知道自己可能會進入的內容。

21.5理解異步消息傳遞模式

異步消息傳遞是一個更高級的主題，但是Docker確實降低了入口欄，因為它很容易在容器中運行隊列，並且您可以快速原型應用程序如何與Event Publishing一起使用。 有不同的方式可以在隊列上發送和接收消息，值得了解一些替代方案。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

我們在本章中使用的模式稱為*publish-subscribe*（或“ pub-sub”），它允許零或更多訂戶接收已發布的消息，如圖21.13所示。

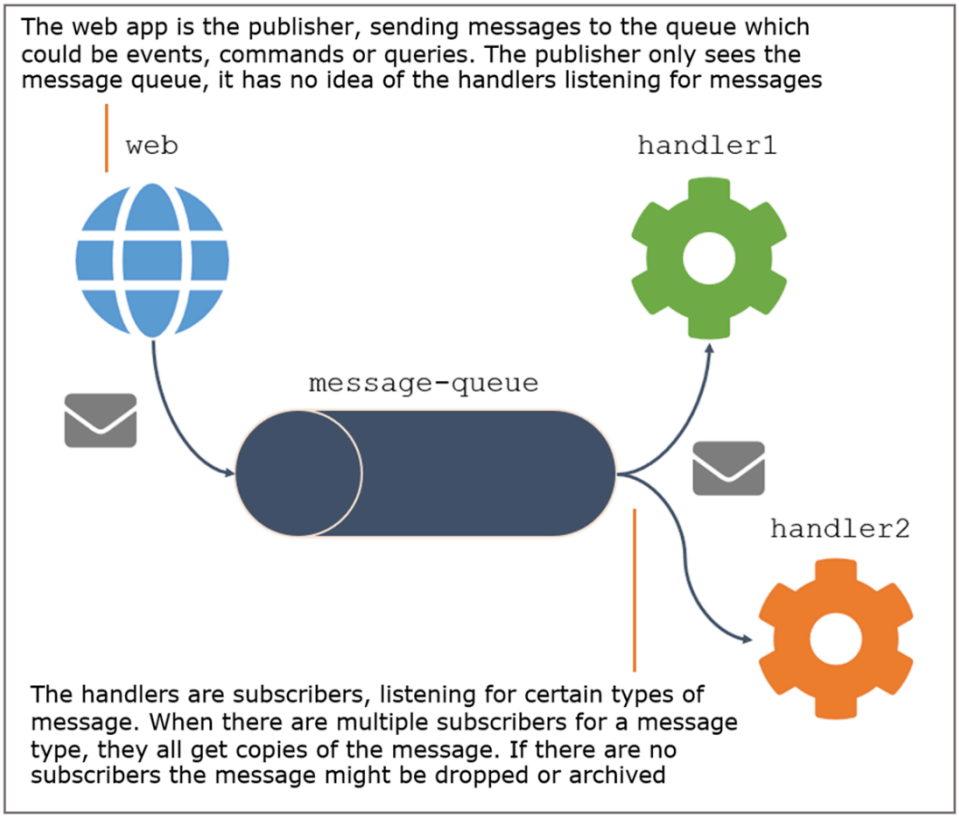


圖21.13 Pub-sub消息傳遞使許多過程都在發布的相同消息上起作用。

這種模式並不適合各種情況，因為消息發布者不知道誰消耗了消息，他們對它的處理方式或何時完成。替代方案是*request-response*消息傳遞，客戶端將消息發送給隊列並等待響應。 處理程序處理請求消息，然後發送一條響應消息，該消息將隊列路由回到客戶端。 這可以用來替換標準同步服務呼叫，而將處理者無法超負荷，並且客戶在等待響應時可以做其他工作。 圖21.14顯示了此模式。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

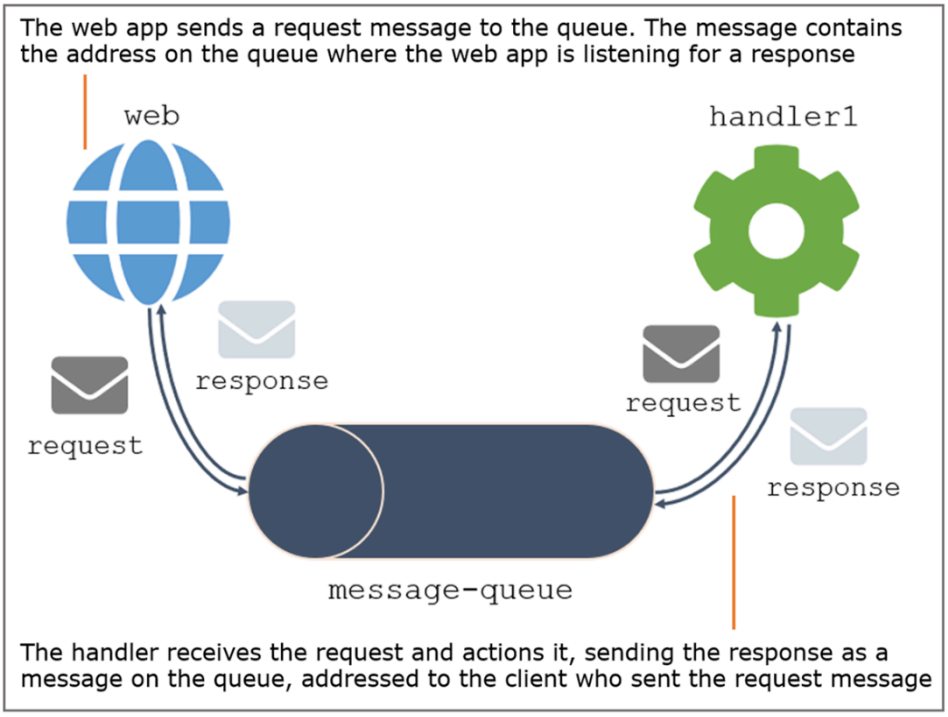


圖21.14請求響應消息是客戶服務通信，而無需直接連接。

幾乎所有隊列技術都支持這些模式以及*fire- and-forget*等變體（其中客戶端在消息中發送命令請求而不是發布事件，但不關心響應）和*scatter-gather*（其中客戶發布了幾個訂閱者的消息，然後將幾個訂戶發揮作用，然後將所有響應歸結為所有響應）。我們在本章中研究了Redis和Nats，但是您還應該考慮另外一項技術：RabbitMQ。 RabbitMQ是一個更高級的隊列，支持複雜的路由和持續的消息傳遞，因此消息將保存到磁盤上，並且隊列內容在容器中重新啟動。 所有這些隊列技術都可以作為Docker Hub上的官方圖像獲得。

消息隊列技術解放了您的應用程序設計。您可以在事件驅動的體系結構中構建或逐漸發展為一個，或者只是將消息用於關鍵事件。 當您開始部署沒有停機時間的新功能，或者縮放處理程序以保護飢餓的數據庫而不會崩潰時，您將意識到模式的力量，您將很高興能夠在本章結尾處完成。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

21.6實驗室

這是本書的最後一個實驗室，這有點偷偷摸摸。目的是為待辦事項應用程序添加另一個消息處理程序 - 一個在保存後更改項目的文本。該處理程序已經存在，因此這主要是將新服務接線到撰寫文件中，但是您也需要進行一些配置設置。

您的解決方案需要使用Image Diamol/CH21-Mutating-Handler：2e從Docker Hub運行新處理程序，並且當您有工作時，有幾件事要調查：

- 新的組件會聆聽稱為event.todo.itemsaved的事件的事件，但尚未發布這些事件。您需要搜索一個配置設置，可以應用於現有組件之一，以使其發布這些事件。 新組件具有不良的默認配置集，因此它不會使用消息隊列的正確地址。您需要搜索設置並將其修復。
-

這不像看起來那樣討厭。您需要的答案全部都在Dockerfiles中，而您只需要在撰寫文件中設置值，而無需更改源代碼或重建圖像。這是一個有用的練習，因為當您將Docker用於REAL時，您肯定會花一些時間嘗試找出配置設置，而最終消息處理程序為待辦事項應用程序添加了一個有用的功能。

與往常一樣，我的解決方案是在Github上的，並通過屏幕截圖證明它有效：https：// github。 com/sixeyed/diamol/blob/2e/ch21/lab/readme.md。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

22 永遠不要結束

Docker是一項真正令人興奮的技術，因為它具有許多用途 - 從運行自己的Git服務器到遷移傳統應用程序到雲到構建和運行全新的雲本地應用程序的所有用途。我希望我們在這本書中的旅程能夠幫助您對容器獲得信心，現在您知道可以將它們放在當前或下一個項目中。最後一章為您提供了一些有關如何成功實現這一目標的暗示，並以對Docker社區的介紹結束。

22.1運行自己的概念證明

您使用Docker越多，對容器的舒適性就越舒適，您將越是從技術中獲得。 幾乎所有應用程序都可以容器化，因此運行概念證明（POC）將您自己的應用程序之一遷移到Docker是一個很好的開始。 這將使您有機會將本書的實用技能從您自己的工作中帶入您的作品，最終結果將是您可以向團隊其他成員展示的東西。

成功的POC不僅僅是Docker Image Build and Docker Container Run。如果您真的想向人們展示容器的力量，那麼您的POC應該具有更多的範圍：

- 旨在擴展多個組件，以便您可以展示Docker組成的功能以運行應用程序的不同配置（請參閱第10章）。
- 從一開始就從最佳實踐開始，並展示向Docker的移動如何改善整個交付生命週期 - 使用多階段的Dockerfiles並優化它們，包括您自己的黃金圖像（請參閱第17章）。
- 包括與集中記錄（第19章）和指標（第9章）的可觀察性。有意義的Grafana儀表板和使用Kibana搜索日誌的能力使您的POC超出了基礎知識。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

- 構建CI/CD管道，即使它是一個非常簡單的使用jenkins（第11章）或GitHub操作（第16章），以顯示如何與Docker自動化所有內容。
- 用Kubernetes（第15章）對您的應用進行建模，並在Docker Desktop上本地運行它，以說明您可以輕鬆地將容器化的應用程序移至雲中。

POC不需要付出巨大的努力。 即使有這樣的擴展範圍，我認為，如果您從相當簡單的應用程序開始，您可以舒適地將練習限制在五天。您不需要整個團隊參與；在此階段，它可以是一個附帶項目。

但是，如果您不能讓批准在工作中使用Docker，那並不意味著您需要停止 - 少數Docker Power用戶在家開始。您可以在Raspberry Pi上的容器中運行一些令人印象深刻的軟件，這將使您定期使用Docker（查看Home Assistant，將所有智能設備集中到Home Assistant，而Pi-Hole則是一個網絡 - 廣告廣告塊和內容緩存）。

22.2為您的組織中的Docker辯護

對於大多數組織而言，Docker是一個巨大的變化，因為它幾乎會影響其各個方面，而且並非每個團隊都準備好接受一種新的工作方式。本書中應該有足夠的時間來幫助您向其他技術團體展示搬到Docker的優勢，但是這是我發現對不同利益相關者吸引的關鍵主題：

- 開發人員可以在其機器上運行整個應用程序堆棧，並使用生產中使用的技術完全相同。沒有更多的時間來追踪丟失的依賴關係或雜耍多個版本的軟件。開發團隊使用與運營團隊相同的工具，因此組件的共同所有權。
- 操作員和管理員可以為每個應用程序使用標準工具和流程，並且每個容器的組件都有用於記錄，指標和配置的標準API。部署和回滾變得完全自動化，失敗應該變得罕見，並且發布可能會更頻繁地發生。
- 數據庫管理員不想在生產中運行數據庫容器，但是容器是為開發人員和測試團隊提供自助服務的有用方法，因此他們不需要DBA來為其創建數據庫。數據庫模式可以移動到源控制並包裝在Docker圖像中，也將CI/CD帶到數據庫開發中。
- 安全團隊將擔心運行時的容器妥協，但是Docker讓您在整個生命週期中深入採用安全性。黃金圖像，安全掃描和圖像簽名都提供了一個安全的軟件供應鏈，使您對部署的軟件有更大的信心。諸如Aqua和Twistlock之類的運行時工具可以自動監視容器的行為並關閉攻擊。

©Manning Publications Co.要發表評論，請轉到Livebook

許可獲得Huan-lin Tsai < huanlin.tsai@gmail.com >

- 商業利益相關者和產品所有者了解發行的語言 - 他們知道發行的歷史問題導致越來越多的大門，這導致發行越來越少。自我修復的應用程序，健康儀表板和連續部署都應鼓勵用戶搬到容器意味著更好的質量軟件，而更少的時間等待新功能。 高級管理層的利益將與該業務保持一致（希望），但他們也將密切關注IT預算。將應用程序從VMS移動到容器可以節省很大的總和，因為當您在容器中更少的服務器上運行更多應用程序時，您可以合併硬件。這也減少了操作系統許可。 IT管理人員應該意識到，容器趨勢不會消失。自2014年以來，Docker一直是成功的產品，所有主要雲都提供託管容器平台。將Docker帶入您的路線圖將使您的技術堆棧最新電流並保持團隊的滿意。
-
-

22.3計劃生產路徑

如果您希望您的組織與您一起去，請了解與Docker一起去的地方很重要。在本書的開頭，我引導您瀏覽五種由Docker授權的項目，從現代化的舊應用程序到運行無服務器功能。無論您的POC是否適合其中一個定義，還是您正在做更異國情調的事情，您都需要了解最終目標，以便可以計劃路線圖和跟踪進度。

您需要做出的主要決定是在雲和Kubernetes中的託管容器服務之間。您已經在本書中使用了，託管服務可能是入門的好方法，但是對於大多數組織來說，Kubernetes是一個更好的選擇。您將在Kubernetes中使用所有Docker Skills運行應用程序，但是還有很多要學習的東西，並且您需要考慮一個非常陡峭的學習曲線；*Learn Kubernetes in a Month of Lunches*將幫助您完成它。如果您打算在數據中心運行容器平台，請注意，Kubernetes是一個複雜的系統，需要專用的管理團隊，並且商業產品（如牧場主）可能是一個更好的選擇。

22.4與Docker社區會面

我將通過確保您知道自己並不孤單來結束這本書。 Docker社區很大，在線空間非常活躍，面對面的聚會在全球舉行。您一定會找到樂於分享自己的知識和經驗的人，而Docker社區幾乎是最友好的人。這是加入的地方：

- Docker Community Slack Group -HTTPS：// Dockr。 ly/懈怠。 找到面對面
- 或虛擬的聚會-HTTPS：//事件。 docker.com。

© Manning Publications Co. To comment go to liveBook

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

- 跟隨碼頭機隊長；這些是Docker認可的社區成員的專業知識和分享-WWW.Docker。COM/社區/隊長。 Dockercon，集裝箱會議 - https：// dockercon。 com。
-

我也是那個社區的一部分 - 您會在社區Slack @Elton-Stoneman和X（以前為Twitter）@EltonStoneman上找到我；隨時隨時與我聯繫。您會發現我的其他地方是github（@sixeyed）和我的博客https：// blog。 sixeyed.com。感謝您的閱讀。希望您發現這本書有用，希望Docker帶您去處。

© Manning Publications Co. To comment go to liveBook