Microsoft

Craig Zacker

Installation, Speicher-technologien und Computing mit Windows Server 2016

Original Microsoft Prüfungstraining

70-740

dpunkt.verlag

Inhalt

Kapitel 3 Implementieren von Hyper-V

- Prüfungsziel 3.1: Hyper-V installieren und konfigurieren
  - Hardware- und Kompatibilitätsanforderungen für die Installation von Hyper-V bestimmen
  - Hyper-V installieren
  - Verwaltungstools installieren
  - Von vorhandenen Hyper-V-Versionen aktualisieren
  - Verwaltung virtueller Computer delegieren
  - Remoteverwaltung von Hyper-V-Hosts durchführen
  - Virtuelle Computer mit Windows SharePoint Services Direct konfigurieren
  - Geschachtelte Virtualisierung implementieren
- Prüfungsziel 3.2: Einstellungen für virtuelle Computer konfigurieren
  - Einen virtuellen Computer erstellen
  - Beim Ausführen eines virtuellen Computers Arbeitsspeicher hinzufügen oder entfernen
  - Dynamischen Arbeitsspeicher konfigurieren
  - Dynamische Speicherzuordnungen
  - NUMA-Unterstützung konfigurieren
  - Smart Paging konfigurieren
  - Ressourcenmessung konfigurieren
  - Integrationsdienste verwalten
  - Virtuelle Computer der Generation 1 und 2 erstellen und konfigurieren und angemessene Nutzungsszenarien bestimmen
  - Erweiterten Sitzungsmodus implementieren
  - Virtuelle Computer für Linux und FreeBSD erstellen
  - Linux Integration Services (LIS) installieren und konfigurieren
  - FreeBSD Integration Services installieren und konfigurieren
  - Sicheren Start für Windows- und Linux-Umgebungen implementieren

Prüfungsziel 5.1: Hochverfügbarkeits- und Notfallwiederherstellungsoptionen in Hyper-V implementieren

- Hyper-V-Replikat implementieren
- Livemigration implementieren
- Shared Nothing-Livemigration implementieren
- CredSSP- oder Kerberos-Authentifizierungsprotokoll für Livemigration konfigurieren
- Speichermigration implementieren

Prüfungsziel 5.2: Failoverclustering implementieren

- Arbeitsgruppe, Cluster für einfache und mehrere Domänen implementieren
- Quorum konfigurieren
- Clusternetzwerk konfigurieren
- Einzelne Knoten oder Clusterkonfiguration wiederherstellen
- Clusterspeicher konfigurieren
- Clusterfähiges Aktualisieren implementieren
- Paralleles Upgrade für Clusterbetriebssysteme implementieren
- Freigegebene Clustervolumes konfigurieren und optimieren
- Cluster ohne Netzwerknamen konfigurieren
- Dateiserver mit horizontaler Skalierung implementieren
- Verschiedene Szenarien für die Verwendung von SoFS statt eines gruppierten Dateiservers bestimmen
- Nutzungsszenarien für die Implementierung von Gastclustering bestimmen
- Eine Clusterspeicherplätze-Lösung mit freigegebenen SAS-Speicheranlagen implementieren
- Speicherreplikat implementieren
- Cloudzeugen implementieren
- VM-Resilienz implementieren

KAPITEL 4

Implementieren von Windows-Container

Container sind ein Hilfsmittel, um virtualisierte, isolierte Betriebssystemumgebungen für die Bereitstellung und Ausführung von Anwendungen schnell bereitstellen zu können. Windows Server 2016 unterstützt Container in Zusammenarbeit mit einem Open-Source-Container-Modul namens Docker.

Dieses Kapitel befasst sich mit folgenden Prüfungszielen:

- Windows-Container bereitstellen
- Windows-Container verwalten

Prüfungsziel 4.1: Windows-Container bereitstellen

Seit den frühen Tagen von Windows ist Virtualisierung ein wichtiges Schlagwort gewesen. Virtueller Speicher ist schon seit Jahrzehnten gebräuchlich; Windows kann Festplattenplatz so nutzen, dass das System scheinbar mehr Arbeitsspeicher besitzt als tatsächlich vorhanden ist. Hyper-V virtualisiert Hardware, erstellt dabei Computer innerhalb eines Computers, die scheinbar ihre eigenen Prozessoren, Arbeitsspeicher und Festplatten haben, obwohl sie in Wirklichkeit die Ressourcen des Hostservers gemeinsam nutzen. Container sind ein neues Feature in Windows Server 2016, das Betriebssysteme virtualisiert.

In diesem Abschnitt geht es um folgende Themen:

- Installationsanforderungen und angemessene Szenarien für Windows-Container bestimmen
- Windows Server-Containerhost in physischen oder virtualisierten Umgebungen installieren und konfigurieren
- Windows Server-Containerhost auf Windows Server Core oder Nanoserver in einer physischen oder virtualisierten Umgebung installieren und konfigurieren
- Docker auf Windows-Server und Nanoserver installieren

- Docker Daemon-Startoptionen konfigurieren
- Windows PowerShell für die Verwendung mit Containern konfigurieren
- ein Basisbetriebssystem installieren
- ein Image markieren
- ein Betriebssystem-Image deinstallieren
- Windows Server-Container erstellen
- Hyper-V-Container erstellen

Installationsanforderungen und angemessene Szenarien für Windows-Container bestimmen

So wie virtuelle Computer gewissermaßen Abbilder von separaten Computern darstellen, bieten Container das, was wie separate Instanzen des Betriebssystems erscheint, wobei jeder Container mit eigenem Speicher und Dateisystem ausgestattet ist und eine saubere, neue Kopie des Betriebssystems ausführt. Im Unterschied zu virtuellen Computern allerdings, die separate Kopien des Betriebssystems ausführen, nutzen Container das Betriebssystem des Hostsystems gemeinsam. Weder ist es erforderlich, eine separate Instanz des Betriebssystems für jeden Container zu erstellen, noch führt der Container eine Bootsequenz aus, lädt Bibliotheken oder teilt den Betriebssystemdateien Arbeitsspeicher zu. Container starten sekundenschnell und Sie können mehr Container auf einem Hostsystem einrichten als virtuelle Computer.

Benutzer, die mit Containern arbeiten, sehen zunächst eine saubere Betriebssysteminstallation, die für Anwendungen bereitsteht. Die Umgebung ist vollkommen vom Host getrennt und auch von anderen Containern, was mit Namespace-Isolierung und Ressourcenkontrolle realisiert wird.

Namespace-Isolierung heißt, dass jeder Container nur auf die Ressourcen zugreifen darf, die für ihn verfügbar sind. Dateien, Ports und laufende Prozesse scheinen alle dem Container fest zugeordnet zu sein, selbst wenn sie mit dem Host und mit anderen Containern gemeinsam genutzt werden. Die Arbeitsumgebung sieht wie die eines virtuellen Computers aus, doch im Unterschied zu einem virtuellen Computer, der separate Kopien aller Betriebssystemdateien verwaltet, nutzt ein Container diese Dateien mit dem Host gemeinsam und kopiert sie nicht. Nur dann, wenn ein Benutzer oder eine

Anwendung eine Datei in einem Container ändert, wird eine Kopie im Dateisystem des Containers angelegt.

Ressourcenkontrolle bedeutet, dass ein Container nur auf eine bestimmte Menge von Prozessorzyklen, Systemspeicher, Netzwerkbandbreite und anderer Ressourcen zugreifen darf, nicht auf mehr. Eine Anwendung, die in einem Container läuft, findet eine saubere Sandbox-Umgebung vor, ohne Zugriff auf Ressourcen, die anderen Containern oder dem Host zugeordnet sind.

Container-Images

Da sich neue Container in Sekunden erstellen lassen und jeder Container vollkommen isoliert ist, sind Container eine ideale Plattform für die Anwendungsentwicklung und das Testen von Software. Es gibt aber noch weitere Vorteile.

Container basieren auf Images. Um einen neuen Container zu erstellen, laden Sie ein Image aus einem Repository herunter und führen es aus. Wenn Sie ein Image von Windows Server 2016 Server Core ausführen, bekommen Sie einen Container mit einer sauberen Instanz des Betriebssystems, das in ihm läuft. Alternativ können Sie Windows Server-Images mit Rollen oder Anwendungen herunterladen, zum Beispiel mit Internet Information Services (IIS) oder Microsoft SQL Server, die bereits installiert und ausführungsbereit sind.

Das grundlegende Betriebssystemimage ändert sich niemals. Wenn Sie eine Anwendung im Container installieren und dann ein neues Image erstellen, enthält das resultierende Image nur die Dateien und Einstellungen, die für die Ausführung der Anwendung erforderlich sind. Natürlich ist das neue Image, das Sie erstellt haben, relativ klein, weil es nicht das gesamte Betriebssystem enthält. Um die Anwendung mit anderen Benutzern zu teilen, brauchen Sie ihnen nur das neue, kleinere Image zu schicken, sofern sie bereits das Basisbetriebssystemimage besitzen.

Dieser Vorgang kann sich über so viele Iterationen wie nötig fortsetzen, mit Schicht auf Schicht von Images auf dieser ursprünglichen Basis. Dies kann in einer äußerst effizienten Softwareentwicklungsumgebung resultieren. Anstatt riesige VHD-Dateien übertragen oder ständig neue virtuelle Computer erstellen und installieren zu müssen, brauchen Sie nur kleine Container-Images zu übertragen, die ohne Hardwarekompatibilitätsprobleme laufen.

Windows Server-Containerhost in physischen oder virtualisierten Umgebungen installieren und konfigurieren

Windows Server 2016 unterstützt zwei Arten von Containern: Windows Server-Container und Hyper-V-Container. Der Unterschied zwischen den beiden liegt im Grad der Containerisolierung, die sie bieten. Windows Server-Container arbeiten im Benutzermodus und nutzen alles mit dem Hostcomputer gemeinsam, Betriebssystemkernel und Systemspeicher eingeschlossen.

Aus diesem Grund ist es vorstellbar, dass eine Anwendung ob versehentlich oder bewusst in der Lage sein könnte, aus den Grenzen ihres Containers auszubrechen und andere Prozesse zu beeinflussen, die auf dem Host oder in anderen Containern laufen. Diese Option ist deshalb vorzuziehen, wenn die in verschiedenen Containern laufenden Anwendungen grundsätzlich vertrauenswürdig sind.

Hyper-V-Container bieten eine zusätzliche Isolationsstufe, indem sie mit dem Hypervisor eine separate Kopie des Betriebssystemkernels für jeden Container erstellen. Obwohl sie für die manuelle Verwaltung weder sichtbar noch zugänglich sind, erstellt Hyper-V virtuelle Computer, die Windows-Container enthalten, wobei sie die Basiscontainer-Images verwenden, wie Abbildung 4–1 zeigt. Die Containerimplementierung ist praktisch die gleiche, der Unterschied zeigt sich in den Umgebungen, in denen die beiden Arten von Containern existieren.

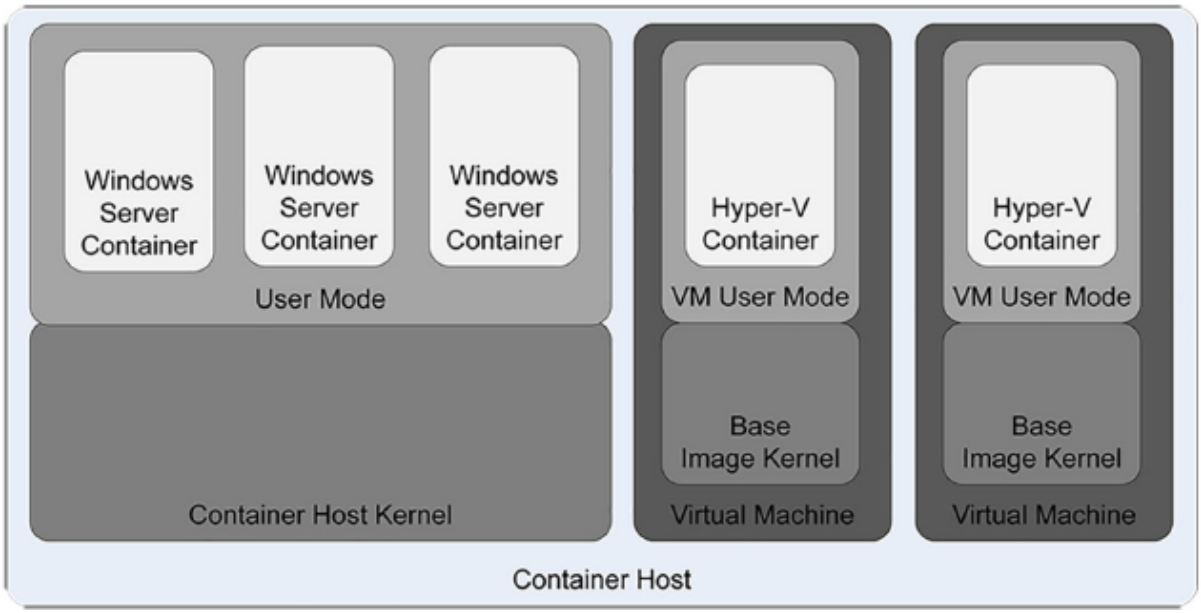

Abb. 4–1 Architektur von Windows-Containern

Weil Hyper-V-Container in einem virtuellen Computer existieren, verfügen sie über einen eigenen zugewiesenen Arbeitsspeicher sowie über isolierten Speicher und Netzwerk-E/A. Dies ergibt eine Container-Umgebung, die sich für das eignet, was Microsoft als »feindliche mehrinstanzfähige« Anwendungen bezeichnet, beispielsweise eine Situation, in der ein Unternehmen Container für Kunden bereitstellt, in denen sie ihren eigenen Code ausführen können, was nicht als vertrauenswürdig gilt. Somit bietet Windows Server 2016 mit den hinzugefügten Hyper-V-Containern drei Isolationsebenen, die von der getrennten Betriebssysteminstallation von virtuellen Hyper-V-Computern über die Hyper-V-Container mit getrenntem Kernel und Arbeitsspeicher bis hin zu Windows Server-Containern mit freigegebenen Kernels und anderen Ressourcen reichen.

Einen Containerhost installieren

Windows Server 2016 bringt ein Feature namens *Container* mit, das Sie installieren müssen, um Container zu unterstützen. Um aber Container zu erstellen und zu verwalten, müssen Sie Docker herunterladen und installieren. Das ist die Anwendung, die das Feature unterstützt.

Das Feature *Container* installieren Sie mit dem Assistenten zum Hinzufügen von Rollen und Features. Auf der Seite *Features auswählen* aktivieren Sie das Kontrollkästchen *Container* (siehe Abbildung 4–2).

HINWEIS **Windows Server-Installation**

Um Windows Server-Container zu erstellen, muss das Hostbetriebssystem auf dem Laufwerk C des Computers installiert sein, was standardmäßig gegeben ist. Die gemeinsame Nutzung des Betriebssystemkernels ist dadurch einfacher. Für das Erstellen von Hyper-V-Containern ist dies nicht erforderlich, weil der Hypervisor dafür zuständig ist, jedem Container eine Kopie des Kernels bereitzustellen.

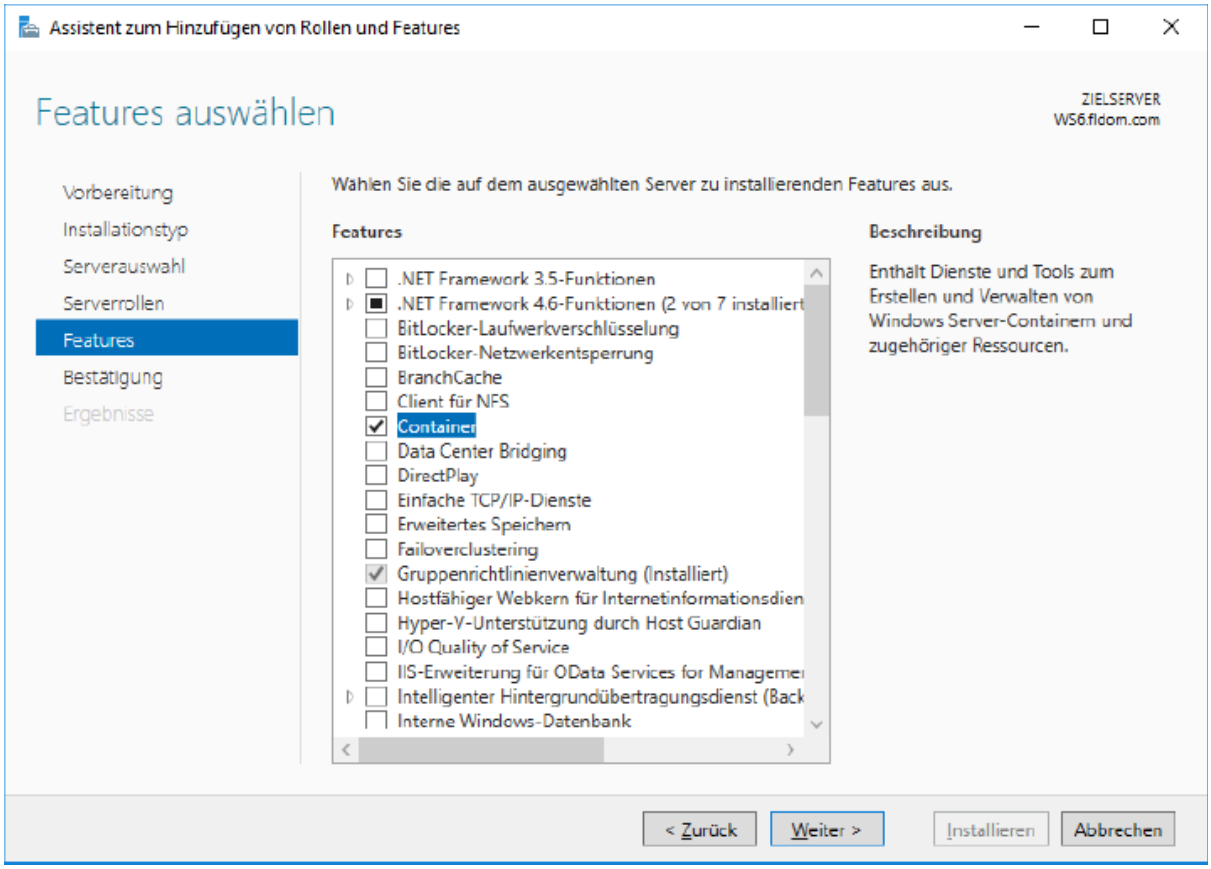

Abb. 4–2 Das Feature *Container* im Assistenten zum Hinzufügen von Rollen und Features installieren

Um Hyper-V-Container zu erstellen, müssen Sie sowohl das Feature *Container* als auch die Rolle *Hyper-V* installieren. Selbst wenn Sie für die Container keine virtuellen Computer erstellen, installiert die Rolle *Hyper-V* den Hypervisor. Dieser ist erforderlich, um die separate Kopie des Windows-Kernels für jeden Hyper-V-Container zu erstellen.

Die allgemeinen Hardwareanforderungen der Hyper-V-Rolle gehen über die Anforderungen des Betriebssystems Windows Server 2016 selbst hinaus. Bevor Sie die Hyper-V-Rolle auf einem Server unter Windows Server 2016 installieren können, muss folgende Hardware vorhanden sein:

- Ein 64-Bit-Prozessor mit hardwareseitiger Unterstützung der Virtualisierung und Second Level Address Translation (SLAT). Eine derartige Virtualisierung ist in Prozessoren vorhanden, die eine Virtualisierungsoption beinhalten, beispielsweise Intel-Virtualisierungstechnologie (Intel VT-x) oder AMD-Virtualisierung (AMD-V).
- Hardwaregestützte Datenausführungsverhinderung (Data Execution Prevention, DEP), bei Intel als eXecute Disable (XD) und bei AMD als No

eXecute (NX) bezeichnet. CPUs verwenden diese Technik, um Speicherbereiche für Prozessoranweisungen von Speicherbereichen für Daten zu trennen. Bei Intel-Prozessoren muss das XD-Bit (eXecute Disable) aktiviert werden, bei AMD-Prozessoren das NX-Bit (No eXecute).

- Erweiterungen für den VM-Überwachungsmodus, bei Intel-Prozessoren als VT-c bezeichnet.
- Ein System-BIOS oder UEFI, das die Virtualisierungshardware unterstützt und auf dem die Virtualisierungsfunktion aktiviert wurde.

Wenn Sie die Hyper-V-Rolle mit dem Hyper-V-Manager installieren, fordert der Assistent zum Hinzufügen von Rollen und Features Sie auf, auch die Hyper-V-Verwaltungstools zu installieren. Erstellen Sie Hyper-V-Container, aber keine virtuellen Hyper-V-Computer, ist es nicht erforderlich, die Verwaltungstools zu installieren.

Container virtualisieren

Windows Server 2016 unterstützt die Verwendung von Containern in virtuellen Hyper-V-Computern. Das Feature *Container* und die Docker-Dateien können Sie in jedem virtuellen Computer installieren. Allerdings muss das System die Anforderungen für geschachtelte Virtualisierung erfüllen, damit sich auf einem virtuellen Computer Hyper-V-Container erstellen lassen.

Um einen geschachtelten Hyper-V-Hostserver zu erstellen, müssen sowohl der physische Host als auch der virtuelle Computer, auf dem Sie die Hyper-V-Container erstellen, Windows Server 2016 ausführen. Der virtuelle Computer kann mit den Installationsoptionen Desktopdarstellung, Server Core oder Nanoserver ausgeführt werden. Des Weiteren ist im physischen Host ein Intel-Prozessor mit VT-x und Unterstützung für Extended Page Tables (EPT) erforderlich.

Bevor Sie Hyper-V auf dem virtuellen Computer installieren, müssen Sie seinem virtuellen Prozessor die Virtualisierungstechnologie auf dem physischen Computer zugänglich machen. Dazu ist es erforderlich, den virtuellen Computer herunterzufahren und auf dem physischen Host in einer PowerShell-Sitzung einen Befehl wie im folgenden Beispiel auszuführen:

```
set-vmprocessor -vmname server1
```

```
    -exposevirtualizationextensions $true
```

Darüber hinaus sind die folgenden Konfigurationsänderungen auf dem virtuellen Computer, der als Hyper-V-Host fungiert, vorzunehmen (angegeben zuerst als Ort im Dialogfeld *Einstellungen* des virtuellen Computers im Hyper-V-Manager und anschließend als PowerShell-Befehl):

- Auf der Seite *Arbeitsspeicher* geben Sie dem virtuellen Computer mindestens 4 GB RAM und deaktivieren das Kontrollkästchen *Dynamischen Arbeitsspeicher aktivieren*.

```
set-vmmemory -vmname server1

    -startupbytes 4gb

    -dynamicmemoryenabled $false
```

- Auf der Seite *Prozessor* setzen Sie *Anzahl virtueller Prozessoren* auf 2.

```
set-vmprocessor -vmname server1

    -count 2
```

- Auf der Seite *Netzwerkkarte/Erweiterte Features* aktivieren Sie das Kontrollkästchen *Spoofing von MAC-Adressen aktivieren*.

```
set-vmnetworkadapter -vmname server1

    -name "network adapter"

    -macaddressspoofing on
```

Nach diesen Änderungen können Sie den virtuellen Computer starten, die Rolle *Hyper-V* installieren und mit Docker fortfahren, um Hyper-V-Container zu

erstellen.

Windows Server-Containerhost auf Windows Server Core oder Nanoserver in einer physischen oder virtualisierten Umgebung installieren und konfigurieren

Ein Computer, der mit der Option *Server Core* installiert wurde, kann als Containerhost fungieren. Die Anforderungen sind die gleichen wie für einen Server, der mit der Option *Desktopdarstellung* installiert wurde, außer dass Sie die erforderlichen Features entweder per Befehlszeile installieren oder das System remote verwalten müssen.

Nachdem Sie in eine PowerShell-Sitzung gewechselt sind, können Sie das Feature *Container* und die Rolle *Hyper-V* mit dem folgenden Befehl installieren:

```
install-windowsfeature -name containers, hyper-v
```

Nanoserver als Containerhost konfigurieren

Die Windows Server 2016-Installationsoption *Nanoserver* unterstützt sowohl Windows Server-Container als auch Hyper-V-Container. Die Nanoserver-Implementierung enthält Pakete, die das Feature *Container* und die Rolle *Hyper-V* unterstützen. Beides können Sie hinzufügen, wenn Sie ein Nanoserver-Image mit dem Windows PowerShell-Cmdlet *New-NanoServerImage* erzeugen, wie das folgende Beispiel zeigt:

```
new-nanoserverimage -deploymenttype guest

   -edition datacenter

   -mediapath d:\

   -targetpath c:\nano\nano1.vhdx

   -computername nano1
```

```
    -domainname contoso

    -containers
```

Dieser Befehl erzeugt ein Nanoserver-Image mit folgenden Eigenschaften:

- **deploymenttype guest** Erzeugt ein Image für die Verwendung auf einem virtuellen Hyper-V-Computer.
- **edition datacenter** Erzeugt ein Image mit der Datacenter-Edition von Windows Server.
- **mediapath d:** Greift auf die Nanoserver-Quelldateien von Laufwerk *D:* zu.
- **targetpath c:\nano\nano1.vhdx** Erzeugt eine VHDX-Imagedatei im Ordner *C:\nano* mit dem Namen *Nano1.vhdx*.
- **computername nano1** Weist dem Nanoserver den Computernamen *Nano1* zu.
- **domainname contoso** Veranlasst den Beitritt des Computers in die Domäne *Contoso*.
- **containers** Installiert das Feature *Container* als Teil des Images.
- **compute** Installiert die Rolle *Hyper-V* als Teil des Images.

Wenn Sie vorhaben, Hyper-V-Container auf dem Gast-Nanoserver zu erstellen, müssen Sie ihm Zugriff auf die Virtualisierungsfunktionen des Hyper-V-Servers bieten. Das bewerkstelligen Sie in folgenden Schritten:

1. Erstellen Sie einen neuen virtuellen Computer mit der von Ihnen erstellten Nanoserver-Imagedatei. Starten Sie ihn aber noch nicht.
2. Gewähren Sie dem virtuellen Computer auf dem Hyper-V-Hostserver Zugriff auf die Virtualisierungsfunktionen des physischen Prozessors im Hyper-V-Server mit einem Befehl wie dem folgenden:

```
set-vmprocessor -vmname nano1

    -exposevirtualizationextensions $true
```

3. Starten Sie den virtuellen Nanoserver-Computer.

Wenn der virtuelle Nanoserver-Computer läuft, müssen Sie eine Remote-PowerShell-Sitzung von einem anderen Computer aus einrichten, damit Sie den Nanoserver-Computer verwalten können. Führen Sie dazu einen Befehl wie den folgenden auf dem Computer aus, mit dem Sie Nanoserver verwalten:

```
enter-pssession -computername nano1

   -credential
```

> ***HINWEIS*** **Nanoserver-Remoteverwaltung**
>
> **Dieser Abschnitt geht davon aus, dass sich der Nanoserver in einem Netzwerk befindet, in dem ein DHCP-Server die TCP/IP-Einstellungen zuweist und er erfolgreich einer AD DS-Domäne beigetreten ist. Wenn dies nicht der Fall ist, müssen Sie die TCP/IP-Einstellungen für den Nanoserver von seiner Konsole aus manuell konfigurieren und dann den Nanoserver zur Liste der vertrauenswürdigen Hosts auf dem Computer hinzufügen, mit dem Sie ihn verwalten.**

Docker auf Windows Server und Nanoserver installieren

Docker ist ein Open-Source-Tool, das seit Jahren die Containerfunktionalität für die Linux-Community realisiert. Da es nun portiert wurde, können Sie die gleiche Funktionalität in Windows implementieren. Docker besteht aus zwei Dateien:

- **Dockerd.exe** Das Docker-Modul, auch als Dienst oder Dämon bezeichnet, läuft im Hintergrund auf dem Windows-Computer.
- **Docker.exe** Der Docker-Client, eine Befehlsshell, in der Sie Container erstellen und verwalten.

Zusätzlich zu diesen beiden Dateien, die Sie herunterladen und installieren müssen, um Container zu erstellen, enthält Docker auch die folgenden Ressourcen:

- **Dockerfiles** Skriptdateien mit Anweisungen für das Erstellen von Container-Images.
- **Docker Hub** Eine cloudbasierte Registrierung, die es Docker-Benutzern ermöglicht, sowohl auf Image- und Code-Repositories zu verweisen als auch eigene Images zu erstellen und zu speichern.
- **Docker Cloud** Ein cloudbasierter Dienst, mit dem Sie Ihre Containeranwendungen bereitstellen können.

Docker auf Windows-Server installieren

Da Docker ein Open-Source-Produkt ist, gehört es nicht zum Lieferumfang von Windows Server 2016. Auf einem Computer, auf dem Windows Server 2016 mit der Option Desktopdarstellung oder Server Core installiert ist, müssen Sie Docker herunterladen und installieren, bevor Sie Container erstellen können. Docker lässt sich mithilfe von *OneGet* herunterladen, einem cloud-basierten Paketmanager für Windows.

Damit Sie auf OneGet zugreifen können, müssen Sie das Modul *DockerMsftProvider* mit dem folgenden Befehl installieren. Wenn die Aufforderung erscheint, einen NuGet-Anbieter zu installieren, antworten Sie mit J (oder drücken ↵).

```
install-module -name dockermsftprovider

   -repository psgallery

   -force
```

Das Cmdlet *Install-Module* lädt das angeforderte Modul herunter und installiert es im Ordner *C:\Program Files\Windows PowerShell\Modules*, wo es von jeder PowerShell-Eingabeaufforderung aus zugänglich ist. Als Nächstes führen Sie den folgenden *Install-Package*-Befehl aus, um Docker herunterzuladen und zu installieren. Wenn die Abfrage erscheint, ob Sie das als nicht vertrauenswürdig gekennzeichnete Paket installieren möchten, antworten Sie mit J.

```
install-package -name docker
```

```
-providername dockermsftprovider
```

Nachdem die Docker-Dateien heruntergeladen sind, wird *Dockerd.exe* als Windows-Dienst registriert und der Client *Docker.exe* in den Pfad aufgenommen, sodass sich das Programm von jedem Ort im Dateisystem ausführen lässt.

Nachdem die Installation abgeschlossen ist, starten Sie den Computer mit dem folgenden Befehl neu:

```
restart-computer -force
```

Docker auf Nanoserver installieren

Wenn Sie einer PowerShell-Remotesitzung mit einem Nanoserver-Computer beigetreten sind, können Sie Docker mit den gleichen Befehlen wie auf einem Computer mit den Installationsoptionen Desktopdarstellung oder Server Core installieren. Microsoft empfiehlt allerdings, dass Sie den Docker-Client vom Remotesystem aus ausführen, nachdem Sie den *Dockerd*-Dienst auf dem Nanoserver installiert haben.

Hierfür müssen Sie die folgenden Aufgaben fertigstellen:

1. **Eine Firewallregel erstellen.** Damit Nanoserver den Docker-Clientverkehr in das System lässt, müssen Sie eine neue Firewallregel erstellen, die Port 2375 für TCP-Verkehr öffnet. Führen Sie dazu den folgenden Befehl in der Nanoserver-Sitzung aus:

```
netsh advfirewall firewall

   add rule name="docker daemon"

   dir=in

   action=allow
```

```
protocol=tcp localport=2375
```

2. **Das Dockerd-Modul konfigurieren, um Netzwerkverkehr zu akzeptieren.** Docker hat seine Wurzeln in Linux und, wie die meisten Linux-Anwendungen, verwendet auch Docker Textdateien für die Konfiguration. Um das *Dockerd*-Modul in die Lage zu versetzen, Clientverkehr über das Netzwerk zu akzeptieren, erstellen Sie auf dem Nanoserver im Verzeichnis *C:\ProgramData\Docker* eine Textdatei namens *daemon.json*, die die folgende Zeile enthält:

```
{ "hosts": ["tcp://0.0.0.0:2375", "npipe://"] }
```

Die beiden folgenden PowerShell-Befehle legen die neue Datei an und fügen den erforderlichen Text ein:

```
new-item -type file
c:\programdata\docker\config\daemon.json
```

```
add-content 'c:\programdata\docker\config\daemon.json'
```

```
'{ "hosts":["tcp://0.0.0.0:2375", "npipe://"] }'
```

3. **Das Dockerd-Modul neu starten.** Nachdem Sie die Datei *daemon.json* erstellt haben, müssen Sie das *Dockerd*-Modul mit dem folgenden Befehl neu starten:

```
restart-service docker
```

4. **Den Docker-Client herunterladen.** Um das *Dockerd*-Modul remote zu verwalten, müssen Sie den *Docker.exe*-Client auf dem Remotesystem (nicht innerhalb der Nanoserver-Sitzung) herunterladen und installieren.

Öffnen Sie hierzu einen Browser und geben Sie die folgende URL ein, um das Docker-Paket herunterzuladen:

```
https://download.docker.com/components/engine/windows-
server/cs-1.12/docker.zip
```

5. In PowerShell lässt sich dies mit dem folgenden Befehl bewerkstelligen:

```
invoke-webrequest "https://download.docker.com/

   components/engine/windows-server/cs-1.12/docker.zip"

   -outfile "$env:temp\docker.zip"

   -usebasicparsing
```

6. **Docker.exe installieren.** Wenn Sie die *Docker.zip*-Datei über einen Browser heruntergeladen haben, legen Sie einen Ordner *C:\ProgramData\Docker* an, extrahieren die Datei *Docker.exe* aus dem ZIP-Archiv und kopieren Sie in diesen Ordner, um die Anwendung zu installieren. Diese Schritte führen Sie in PowerShell mit dem folgenden Befehl aus:

```
expand-archive -path "$env:temp\docker.zip"

   -destinationpath $env:programfiles
```

7. **Die Umgebungsvariable PATH festlegen.** Um den Docker-Client von jedem Ort im Verwaltungssystem ausführen zu können, müssen Sie den Ordner *C:\ProgramData\Docker* zur Umgebungsvariablen *PATH* des Systems hinzufügen. In der grafischen Benutzeroberfläche öffnen Sie über die Systemsteuerung die App *System*, klicken dort auf *Erweiterte*

Systemeinstellungen und im Dialogfeld *Systemeigenschaften* auf die Schaltfläche *Umgebungsvariablen*, um das in Abbildung 4–3 gezeigte Dialogfeld anzuzeigen.

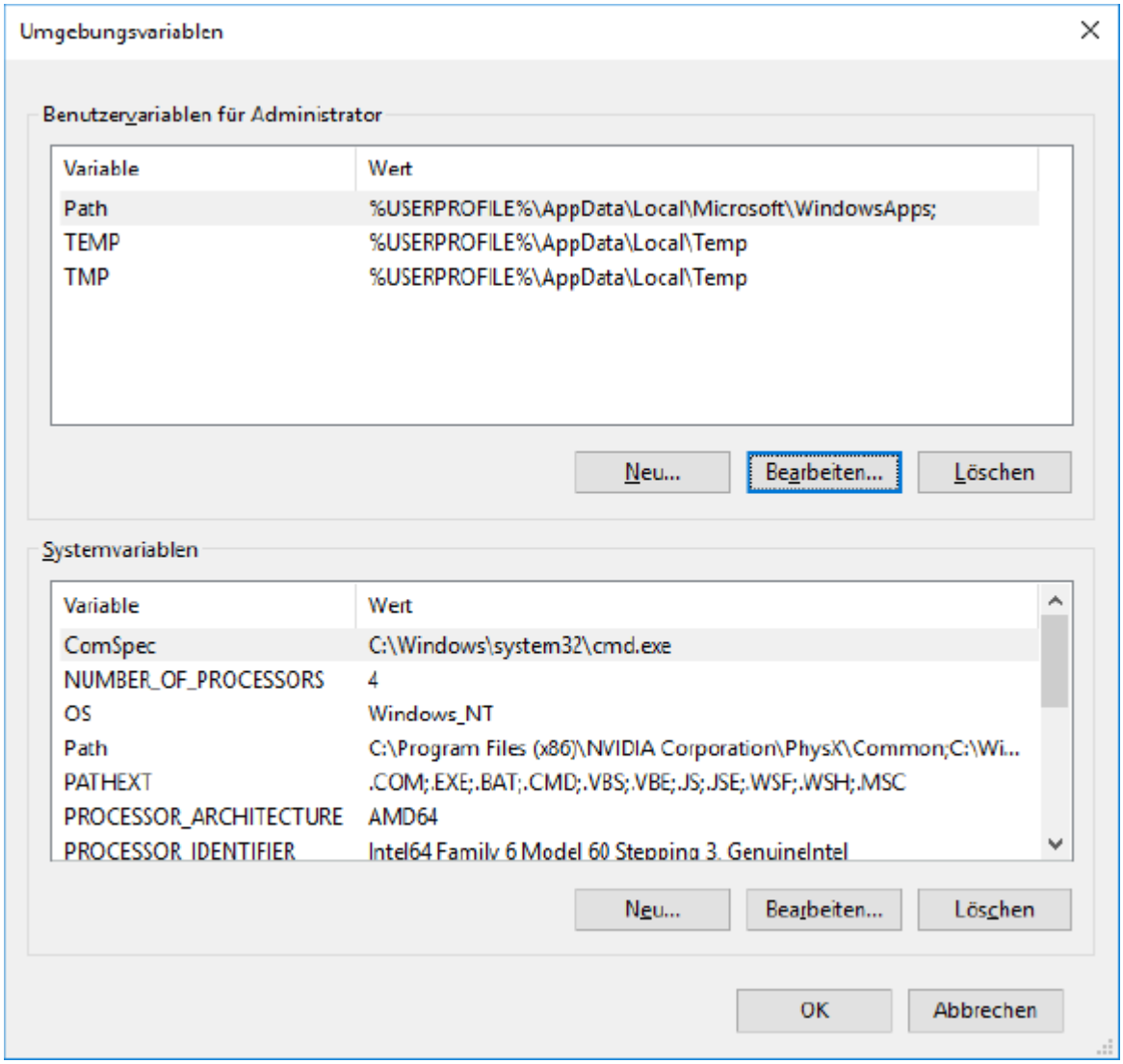

Abb. 4–3 Das Dialogfeld *Umgebungsvariablen*

8. In PowerShell erledigen Sie dies mit dem folgenden Befehl:

```
[environment]::setenvironmentvariable("path",
   $env:path + ";c:\program files\docker",
   [environmentvariabletarget]::machine)
```

Nachdem Sie diese Schritte abgeschlossen haben, können Sie den *Docker.exe*-Client außerhalb der Nanoserver-Sitzung starten, müssen aber in jedem Befehl den folgenden Parameter angeben, wobei Sie die Variable *<ipaddress>* durch die Adresse des Nanoservers ersetzen, den Sie verwalten möchten:

```
-h tcp://<ipaddress>:2375
```

So erstellen Sie beispielsweise mit dem folgenden Befehl einen neuen Container mit dem Image *microsoft/nanoserver*:

```
docker -h tcp://172.21.96.1:2375

   run -it microsoft/nanoserver

   cmd
```

Damit Sie den Parameter *-h* nicht an jeden Befehl anfügen müssen, erstellen Sie wie folgt eine neue Umgebungsvariable:

```
docker_host = "tcp://ipaddress:2375"
```

In PowerShell verwenden Sie hierfür einen Befehl wie den folgenden:

```
$env:docker_host = "tcp://172.21.96.1:2375"
```

Docker Daemon-Startoptionen konfigurieren

Wie bereits im vorherigen Abschnitt erwähnt, ist die Konfigurationsdatei für das *Dockerd*-Modul eine einfache Textdatei namens *daemon.json*, die Sie im selben Ordner wie die Datei *Dockerd.exe* speichern. Neben den Einstellungen, mit denen Sie weiter oben den Clientverkehr über das Netzwerk zugelassen haben, gibt es noch viele andere Konfigurationseinstellungen, die Sie in die Datei aufnehmen können. Sämtliche Einstellungen, die Sie in eine einzelne

daemon.json-Datei schreiben, sollten in einem einzigen Satz von geschweiften Klammern eingeschlossen sein, wie das folgende Beispiel zeigt:

```
{

"graph": "d:\docker"

 "bridge" : "none"

 "group" : "docker"

{"dns": 192.168.9.2, 192.168.9.6 }

}
```

PRÜFUNGSTIPP

Seien Sie sich bewusst, dass die Windows-Portierung von Docker zwar viele Konfigurationseinstellungen der Linux-Version von *Dockerd* unterstützt, aber nicht den kompletten Funktionsumfang bietet. Wenn Sie sich die Docker-Dokumentation ansehen, achten Sie darauf, dass es sich um die Windows-Version des Dokuments handelt.

Images und Container umleiten

Um das *Dockerd*-Modul so zu konfigurieren, dass es Imagedateien und Container in einem alternativen Speicherort ablegt, schreiben Sie in die Datei *daemon.json* den folgenden Befehl, wobei Sie *d:\docker* durch den gewünschten Speicherort ersetzen:

```
{ "graph": "d:\docker" }
```

NAT unterdrücken

Standardmäßig erzeugt das *Dockerd*-Modul eine NAT-Umgebung (Network Address Translation) für Container, damit diese untereinander und mit dem externen Netzwerk kommunizieren können. Sie können dieses Standardverhalten ändern und verhindern, dass das Modul NAT verwendet, wenn Sie den folgenden Befehl in die Datei *daemon.json* einfügen:

```
{ "bridge" : "none" }
```

Eine administrative Gruppe erstellen

Standardmäßig können nur Mitglieder der lokalen Gruppe *Administratoren* mit dem Docker-Client das *Dockerd*-Modul steuern, wenn sie im lokalen System arbeiten. In manchen Fällen können Sie aber Benutzern diese Fähigkeit gewähren, ohne ihnen die Mitgliedschaft *Administratoren* zu geben. *Dockerd* lässt sich mit der folgenden Einstellung in der Datei *daemon.json* so konfigurieren, dass eine andere Gruppe erkannt wird – in diesem Fall die Gruppe namens »docker«:

```
{ "group" : "docker" }
```

DNS-Serveradressen festlegen

Um alternative DNS-Serveradressen für die Betriebssysteme in Containern festzulegen, können Sie die folgende Einstellung in die Datei *daemon.json* einfügen, wobei *address1* und *address2* die IP-Adressen von DNS-Servern sind:

```
{"dns": "address1", "address2" }
```

Windows PowerShell für die Verwendung mit Containern konfigurieren

Das *Dockerd*-Modul wird zwar mit einer *Docker.exe*-Clientshell bereitgestellt, ist aber nicht davon abhängig. Die gleichen Funktionen können Sie auch mit Windows PowerShell-Cmdlets ausführen. Das PowerShell-Modul *Docker* befindet sich wie Docker selbst in einer ständigen Phase gemeinsamer Entwicklung und ist demzufolge kein Bestandteil von Windows Server 2016.

Die aktuelle Version des PowerShell-Moduls können Sie aus einem Repository namens *DockerPS-Dev* mit den folgenden Befehlen herunterladen:

```
register-psrepository -name dockerps-dev

   -sourcelocation https://ci.appveyor.com/nuget/docker-
   powershell-dev

install-module docker

   -repository dockerps-dev

   -scope currentuser
```

Wenn der Download abgeschlossen ist, können Sie sich eine Liste der Docker-Cmdlets mit dem folgenden Befehl anzeigen lassen:

```
get-command -module docker
```

Abbildung 4–4 zeigt die aktuelle Ausgabe des Befehls.

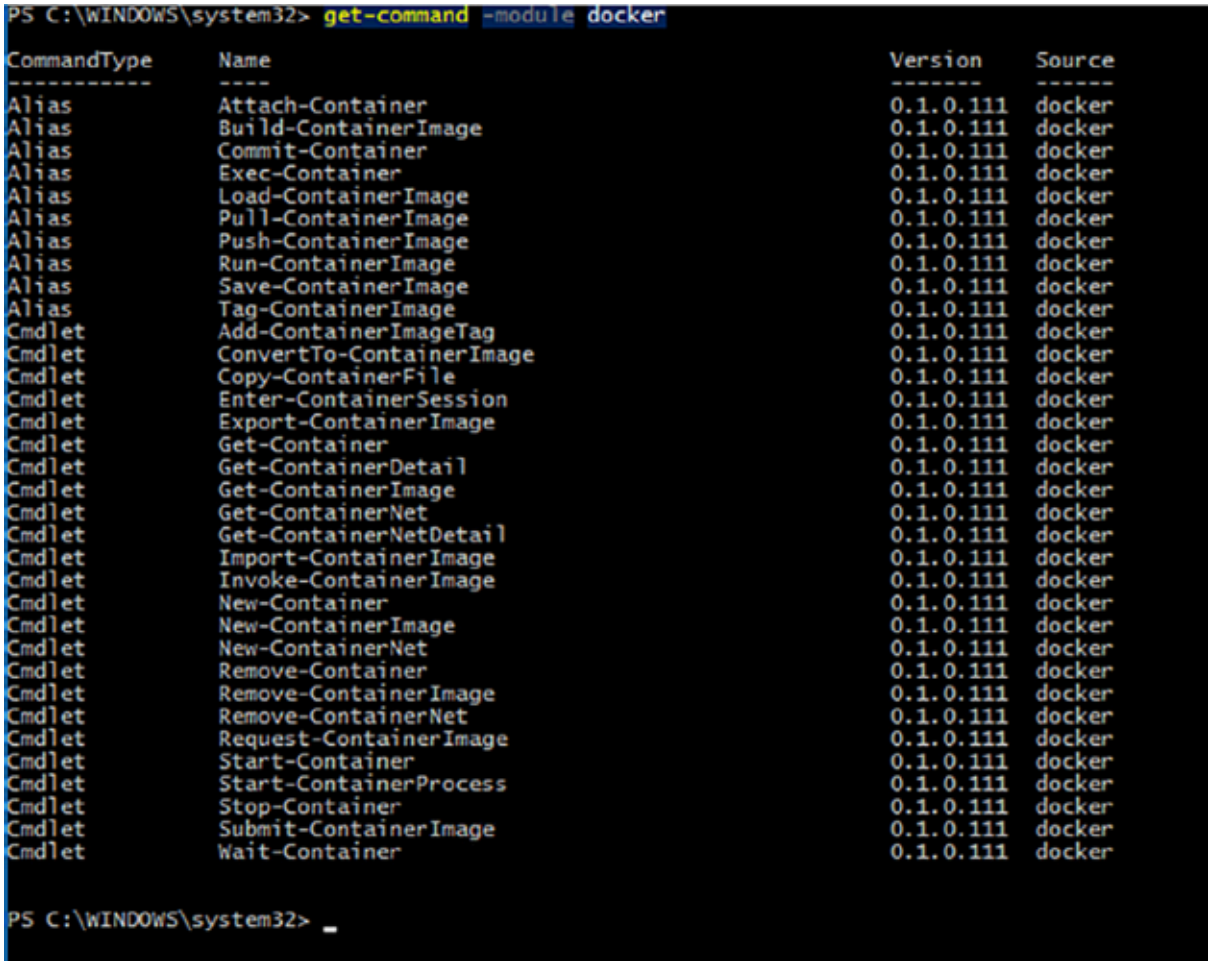

Abb. 4–4 Cmdlets im Modul *Docker* für Windows PowerShell

Wenn Sie einmal das Repository registriert und das Docker-Modul importiert haben, brauchen Sie diese Befehle nicht noch einmal auszuführen. Die neueste Version des Moduls rufen Sie mit dem folgenden Befehl ab:

```
update-module docker
```

Ein Basisbetriebssystem installieren

Sind das *Dockerd*-Modul und der Docker-Client installiert und betriebsbereit, können Sie den ersten Schritt zum Erstellen von Containern gehen: ein Basisbetriebssystemimage aus dem Docker Hub-Repository herunterladen. Microsoft stellt im Repository die Images für Windows Server 2016 Server Core und Nanoserver bereit. Diese können Sie herunterladen, Container damit erstellen und dann Ihre eigenen Container-Images erzeugen.

Um den Docker-Client zu verwenden, führen Sie die Datei *Docker.exe* mit einem Befehl und gegebenenfalls zusätzlichen Optionen und Parametern aus. Wollen Sie ein Image herunterladen, rufen Sie Docker mit dem Befehl *Pull* und dem Namen des Images auf. Zum Beispiel lädt der folgende Befehl das Server Core-Image aus dem Repository herunter:

```
docker pull microsoft/windowsservercore
```

Der äquivalente PowerShell-Befehl sieht folgendermaßen aus:

```
request-containerimage

   -repository microsoft/windowsservercore
```

Abbildung 4–5 zeigt die Ausgabe des Befehls (die je nach Geschwindigkeit Ihrer Internetverbindung etwas dauern kann).

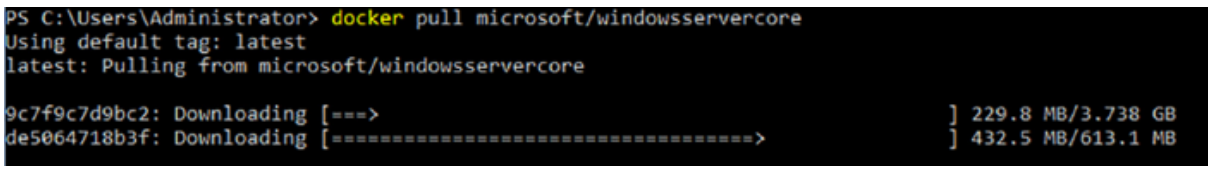

Abb. 4–5 Ausgabe des *Docker Pull*-Befehls

Der *Docker Pull*-Befehl lädt standardmäßig die neueste Version des angegebenen Images herunter, was an der Markierung »latest« zu erkennen ist. Stehen mehrere Versionen des gleichen Images zur Verfügung, wie es in einem Projekt der Anwendungsentwicklung der Fall ist, können Sie das jeweilige Tag spezifizieren und somit eines der vorherigen Images zum Download auswählen. Wenn Sie den *Docker Pull*-Befehl mit dem Parameter *-a* ausführen, erhalten Sie alle Versionen des Images. Besteht das Image, das Sie abrufen, aus mehreren Schichten, lädt der Befehl automatisch alle benötigten Schichten herunter, um das Image in einem Container bereitzustellen.

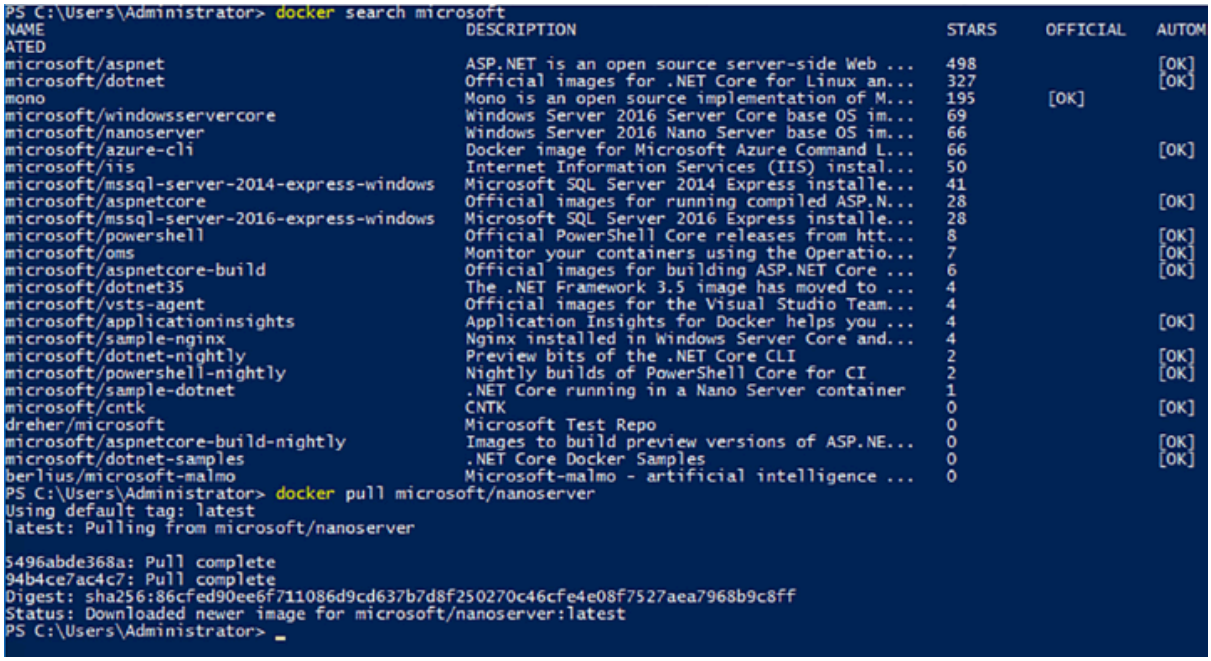

Abb. 4–6 Ausgabe des Befehls *Docker Search*

Wenn Sie wissen, dass im Repository ein Nanoserver-Image bereitsteht, Sie aber dessen genauen Namen nicht kennen, können Sie mit dem Befehl *Docker Search* das Image suchen und es dann mit *Docker Pull* herunterladen, wie Abbildung 4–6 zeigt.

Ein Image markieren

Das Markieren (Tagging) in einem Container-Repository ist ein Mechanismus der Versionskontrolle. Wenn Sie mehrere Versionen des gleichen Images erstellen, beispielsweise aufeinanderfolgende Builds einer Anwendung, ermöglicht Docker es, diesen Images Tags zuzuweisen, die die Versionen identifizieren. Tags sind normalerweise Nummern, die das relative Alter der Image-Iterationen anzeigen, etwa 1.1, 1.2, 2.0 usw.

Es gibt zwei Methoden, um einem Image ein Tag zuzuweisen: *Docker* mit dem Befehl *Tag* ausführen und *Docker Build* mit dem Parameter *-t* aufrufen. In beiden Fällen ist das Format des Image-Kennzeichners das gleiche.

Um ein Image auf Ihrem lokalen Containerhost zu markieren, verwenden Sie die folgende Syntax:

```
docker tag <imagename>:<tag>
```

Wenn Sie das Image auf den Docker Hub uploaden, müssen Sie vor den Imagenamen Ihren Docker Hub-Benutzernamen und einen Schrägstrich setzen:

```
docker tag <username>/<imagename>:<tag>
```

Zum Beispiel könnte ein Benutzer Holly Holt den letzten Build ihrer neuen Anwendung wie folgt markieren:

```
docker tag hholt/killerapp:1.5
```

In Windows PowerShell führen Sie den entsprechenden Befehl mit dem Cmdlet *Add-ContainerImageTag* wie folgt aus:

```
add-containerimagetag -imageidorname c452b8c6ee1a

   -repository hholt/killerapp

   -tag 1.5
```

Fehlt der Tag-Wert im Befehl, weist Docker dem Image automatisch den Tag-Wert »latest« zu, was zu Verwirrungen führen kann. Wenn Sie ein Image aus einem Repository abrufen, ohne ein Tag anzugeben, liefert das Repository das Image mit dem Tag »latest« zurück. Allerdings bedeutet das nicht unbedingt, dass Sie wirklich das neueste Image bekommen.

Das Tag »latest« soll anzeigen, dass es sich beim Image mit dieser Markierung um die neueste Version handelt. Ob das jedoch tatsächlich stimmt oder nicht, hängt von den Leuten ab, die die Tags für dieses Repository verwalten. Man könnte annehmen, dass das Tag »latest« automatisch der jeweils allerneuesten Version eines Images neu zugewiesen wird. Das ist aber nicht der Fall. Das Tag »latest« können Sie jeder Version eines Images zuweisen, auch der ältesten oder der neuesten. Es liegt allein bei den Managern des Repositorys, die Tag-Werte geeignet zu verwalten. Wenn Sie jemand um den letzten Build eines Images bittet, meint diese Person dann den neuesten Build oder den Build mit dem Tag »latest«? Das ist nicht immer dasselbe.

Ein Betriebssystem-Image deinstallieren

Wenn Sie *Docker* mit dem Befehl *Images* ausführen, werden alle Images auf dem Containerhost angezeigt, wie in Abbildung 4–7 zu sehen ist.

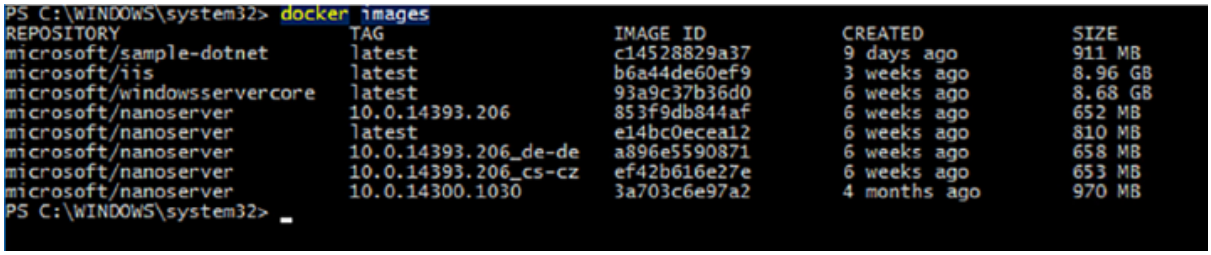

Abb. 4–7 Ausgabe des Befehls *Docker Images*

In manchen Fällen wird Ihnen beim Durchsehen der Liste von Images auffallen, dass Sie manche Images nicht mehr brauchen. In diesem Beispiel sind dies zwei nicht-englische Versionen von Nanoserver, die versehentlich heruntergeladen wurden.

Möchten Sie nicht mehr benötigte Images entfernen und von ihnen belegten Speicherplatz freigeben, führen Sie Docker mit dem Befehl *Rmi* aus und geben entweder das Repository und das Tag des zu löschenden Images oder den Image-ID-Wert wie in den folgenden Beispielen an:

```
docker rmi -f microsoft/nanoserver:10.0.14393.206_de-de
```

```
docker rmi -f a896e5590871
```

Das PowerShell-Äquivalent ist das Cmdlet *Remove-ContainerImage*, wie in den folgenden Beispielen:

```
remove-containerimage
microsoft/nanoserver:10.0.14393.206_de-de
```

```
remove-containerimage a896e5590871
```

Es ist durchaus möglich, dass dasselbe Image mit mehreren Tags aufgelistet ist. Das erkennen Sie an den übereinstimmenden Image-ID-Werten. Wenn Sie versuchen, eines dieser Images mithilfe des Tags zu entfernen, erscheint ein Fehler, weil das Image mit anderen Tags verwendet wird. Mit dem zusätzlichen

Parameter *-f* zwingen Sie den Befehl, alle markierten Verweise auf dasselbe Image zu löschen.

Windows Server-Container erstellen

Mit dem eingerichteten Container-Feature und dem installierten Docker können Sie nun einen Windows Server-Container erstellen. Dazu führen Sie den Befehl *Docker Run* aus und geben das Image an, das Sie im Container ausführen möchten. Zum Beispiel erzeugt der folgende Befehl einen neuen Container mit dem Server Core-Image, das von Docker Hub heruntergeladen wird:

```
docker run -it microsoft/windowsservercore powershell
```

Außer dass das Image in den Container geladen wird, bewirken die Parameter in diesem Befehl Folgendes:

- **i** Erstellt eine interaktive Sitzung mit dem Container.
- **t** Öffnet ein Terminalfenster in den Container.
- **powershell** Führt den PowerShell-Befehl in der Containersitzung aus.

Im Ergebnis erscheint nach dem Laden des Containers eine PowerShell-Sitzung, sodass Sie innerhalb des Containers arbeiten können. Wenn Sie in dieser Sitzung das Cmdlet *Get-ComputerInfo* ausführen, sehen Sie im oberen Teil der Ausgabe (siehe Abbildung 4–8), dass Server Core im Container ausgeführt wird, während die Version Desktopdarstellung im Containerhost läuft.

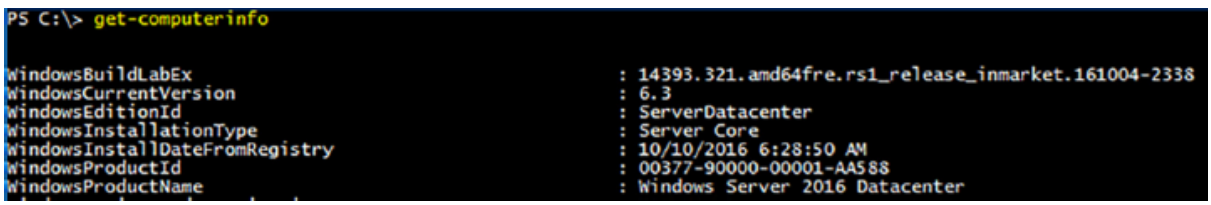

Abb. 4–8 Ausgabe des Cmdlets *Get-ComputerInfo*

Die Schalter des Befehls *Docker Run* können Sie zusammenfassen, sodass die Schalter *-i* und *-t* als *-it* erscheinen. Nach dem Namen des Images kann jeder Befehl stehen, der im Container auszuführen ist. Wenn Sie zum Beispiel *cmd* angeben, öffnen Sie damit die normale Befehlsshell von Windows anstelle von PowerShell.

HINWEIS **Images abrufen**

Ein Image müssen Sie nicht unbedingt als Erstes vom Docker Hub abrufen, bevor Sie es ausführen können. Wenn Sie einen *Docker Run*-Befehl ausführen und das erforderliche Image in Ihrem Containerhost nicht vorhanden ist, leitet Docker automatisch ein Abrufen ein und erstellt dann den Container. Bei großen Images sparen Sie aber Zeit, wenn Sie neue Container erstellen und die jeweiligen Images vorab heruntergeladen haben.

Der Befehl *Docker Run* unterstützt viele Befehlszeilenparameter und Schalter, mit denen Sie die Umgebung des zu erzeugenden Containers optimieren können. Mit dem folgenden Befehl lassen sich diese Parameter und Schalter anzeigen:

```
docker run–help
```

HINWEIS **Docker-Befehle ausführen**

Bei diesem und vielen anderen Docker-Befehlen sind die Befehlszeilenparameter mit doppelten Bindestrichen zu schreiben.

Abbildung 4–9 zeigt ungefähr die Hälfte der verfügbaren Parameter. Zum Beispiel können Sie mit dem Parameter *-h* einen Hostnamen für den Container angeben anstelle der hexadezimalen Zeichenfolge, die der Befehl standardmäßig zuweist.

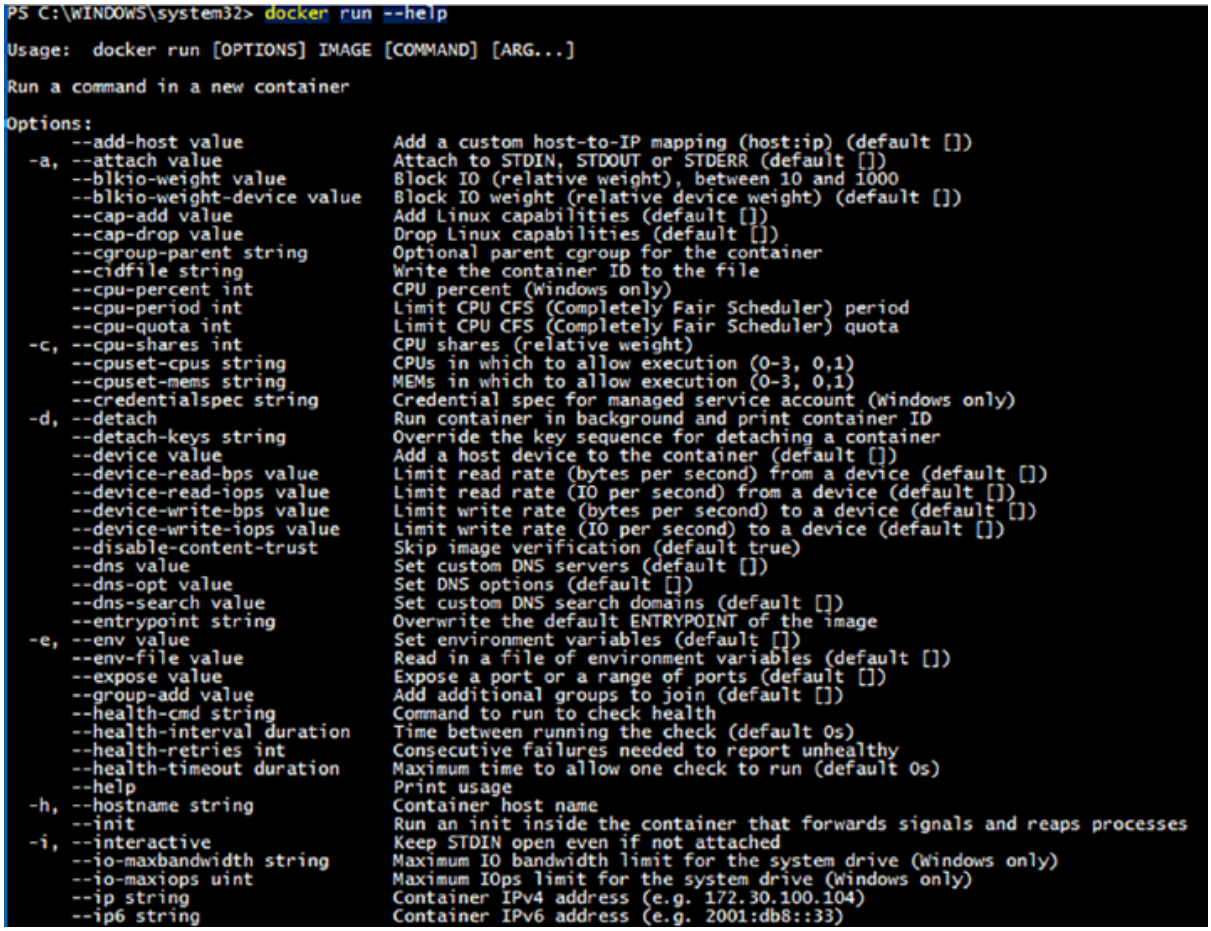

Abb. 4–9 Ausgabe des Befehls *Docker Run —help*

Das PowerShell-Äquivalent des Befehls *Docker Run* verwendet das Cmdlet *New-Container*, wie das folgende Beispiel zeigt:

```
new-container -imageidorname microsoft/windowsservercore

    -input

    -terminal

    -command powershell
```

Hyper-V-Container erstellen

Einen Hyper-V-Container erstellen Sie in fast den gleichen Schritten wie einen Windows Server-Container. Hierfür verwenden Sie denselben *Docker Run*-Befehl,

außer dass Sie den Parameter *—isolation=hyperv* anfügen, wie es im folgenden Beispiel zu sehen ist:

```
docker run -it

   —isolation=hyperv microsoft/windowsservercore

   powershell
```

Nachdem Sie einen Hyper-V-Container erstellt haben, lässt er sich kaum von einem Windows Server-Container unterscheiden. Es gibt nur wenige Methoden, die Arten der Container auseinanderzuhalten. So kann man untersuchen, wie sie mit Prozessen umgehen. Zum Beispiel können Sie zwei Container erstellen und in jedem einen Befehl ausführen, der an den jeweils anderen Computer fortlaufend Ping-Befehle sendet:

```
docker run -it microsoft/windowsservercore

   ping -t localhost

docker run -it

   —isolation=hyperv microsoft/windowsservercore

   ping -t localhost
```

Der vom ersten Befehl erstellte Windows Server-Container lässt im Container einen PING-Prozess laufen, wie die Ausgabe des Befehls *Docker Top* in Abbildung 4–10 zeigt. Die Prozess-ID (PID) lautet hier 404. Wenn Sie dann das Cmdlet *Get-Process* ausführen, um die Prozesse (die mit P beginnen) auf dem Containerhost anzuzeigen, sehen Sie denselben PING-Prozess mit der ID 404. Das hängt damit zusammen, dass der Container den Kernel mit dem Containerhost gemeinsam nutzt.

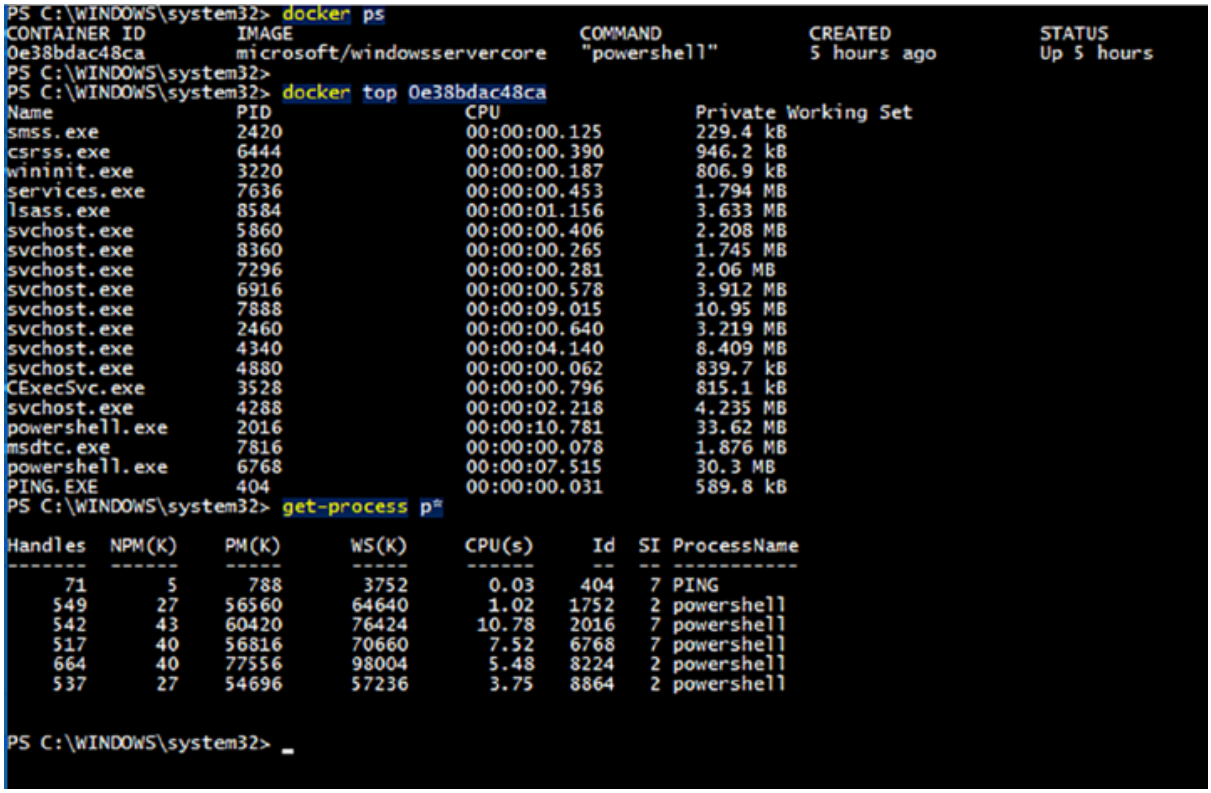

Abb. 4–10 Ausgabe der Befehle *Docker Top* und *Get-Process* für einen Windows Server-Container

Wenn Sie dagegen den Befehl *Docker Top* auf dem Hyper-V-Container ausführen, sehen Sie wieder den PING-Prozess, dieses Mal mit der PID 1852 (siehe Abbildung 4–11). In der Ausgabe des Cmdlets *Get-Process* erscheint aber kein PING-Prozess, weil dieser Container seinen eigenen Kernel besitzt, der vom Hypervisor bereitgestellt wird.

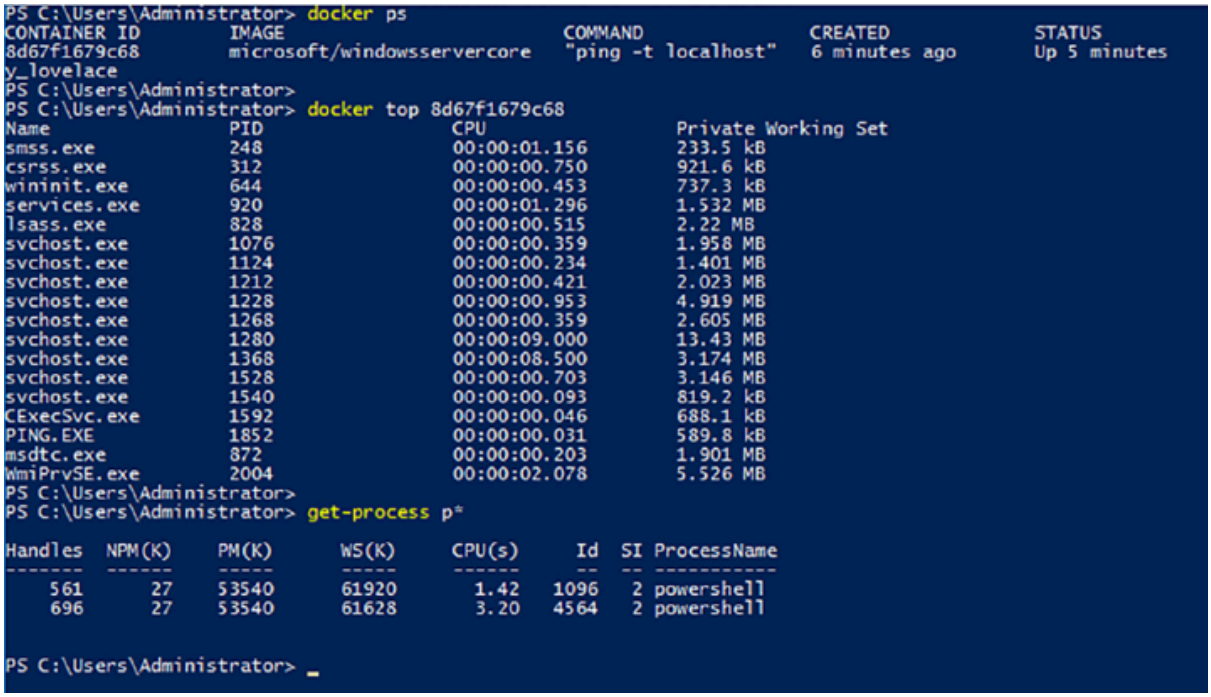

Abb. 4–11 Ausgabe der Befehle *Docker Top* und *Get-Process* für einen Hyper-V-Container

Prüfungsziel 4.2: Windows-Container verwalten

- Windows- oder Linux-Container mit dem Docker-Daemon verwalten
- Windows- oder Linux-Container mithilfe von Windows PowerShell verwalten
- Containernetzwerke verwalten
- Container-Datenvolumen verwalten
- Ressourcensteuerung verwalten
- Neue Container-Images mit Dockerfile erstellen
- Container-Images mit DockerHub-Repository für öffentliche und private Szenarien verwalten
- Container-Images mit Microsoft Azure verwalten

Windows- oder Linux-Container mit dem Docker-Daemon verwalten

Wenn Sie mit dem Befehl *Docker Run* einen neuen Container erstellen, können Sie die Schalter *-it* angeben, um mit ihm interaktiv zu arbeiten, oder sie

weglassen und den Container im Hintergrund laufen lassen. In jedem Fall können Sie mit dem Docker-Client den Container verwalten – ob Windows oder Linux.

Container auflisten

Möchten Sie eine PowerShell- oder CMD-Sitzung verlassen, die Sie in einem Container begonnen haben, können Sie einfach den folgenden Befehl eingeben:

```
exit
```

Dieser Befehl schließt allerdings nicht nur die Sitzung, sondern hält auch den Container an. Ein angehaltener Container existiert weiterhin auf dem Host; er ist lediglich funktionell abgeschaltet. Um eine Sitzung zu verlassen, ohne den Container anzuhalten, drücken Sie Strg + Q und dann Strg + P.

Eine Liste aller auf dem Host ausgeführten Container zeigen Sie mit dem Befehl *Docker ps* an. Wenn Sie den Schalter *-a* (für alle) wie im folgenden Beispiel anfügen, zeigt der Befehl alle Container auf dem Host an, ob sie ausgeführt werden oder nicht (siehe Abbildung 4–12):

```
docker ps -a
```

Abb. 4–12 Ausgabe eines Befehls *Docker ps*

Container starten und anhalten

Einen angehaltenen Container starten Sie mit dem Befehl *Docker Start*, zum Beispiel:

```
docker start dbf9674d13b9
```

Mit dem Befehl *Docker Stop* können Sie einen Container auch zwangsweise anhalten, zum Beispiel:

```
docker stop dbf9674d13b9
```

Die sechs Bytes umfassende hexadezimale Zeichenfolge in diesen Befehlen ist die Container-ID, die Docker dem Container beim Erstellen zuweist. Mit diesem Wert identifizieren Sie in Docker-Befehlen den Container, den Sie verwalten möchten. Außerdem wird dieser Wert zum Computernamen des Containers, wovon Sie sich mit dem Cmdlet *Get-ComputerInfo* überzeugen können, das Sie in einer Containersitzung ausführen. Wenn Sie *Docker PS* mit dem Parameter —*notrunc* (für no truncation, ungekürzt) ausführen, zeigt sich, dass die Container-ID tatsächlich eine Zeichenfolge aus 32 Bytes in hexadezimaler Schreibweise ist, auch wenn es wesentlich bequemer ist, nur die ersten sechs Bytes auf der Befehlszeile zu verwenden (siehe Abbildung 4–13).

Abb. 4–13 Ausgabe des Befehls *Docker ps -a —no-trunc*

An Container anfügen

Mit dem Befehl *Docker Attach* können Sie sich mit einer Sitzung auf einem ausgeführten Container verbinden, wie das folgende Beispiel zeigt:

```
docker attach dbf9674d13b9
```

Wenn Sie den Befehl in mehreren Fenstern ausführen, werden zusätzliche Sitzungen geöffnet, sodass Sie in mehreren Fenstern gleichzeitig arbeiten können.

Images erstellen

Haben Sie einen Container in irgendeiner Weise geändert, können Sie die Änderungen in einem neuen Image speichern. Verwenden Sie hierfür den Befehl *Docker Commit* wie im folgenden Beispiel:

```
docker commit dbf9674d13b9 hholt/killerapp:1.5
```

Dieser Befehl erstellt ein neues Image namens *hholt/killerapp* mit einem Tag-Wert von 1.5. Der Befehl *Docker Commit* erzeugt kein Duplikat des Basisimages mit den von Ihnen vorgenommenen Änderungen, sondern speichert nur die Änderungen. Wenn Sie zum Beispiel mit dem Basisimage *Microsoft/windowsservercore* den Container erstellen und dann Ihre Anwendung installieren, speichert der Befehl *Docker Commit* nur die Anwendung. Und wenn Sie das neue Image einem Kollegen geben, muss er über das Basisimage verfügen (oder es abrufen), um den Container ausführen zu können.

Container entfernen

Möchten Sie einen Container vollständig entfernen, führen Sie den Befehl *Docker rm* wie im folgenden Beispiel aus:

```
docker rm dbf9674d13b9
```

Bevor Sie Container auf diese Weise entfernen können, müssen sie sich im angehaltenen Zustand befinden. Allerdings können Sie auch den Schalter *-f* (für force, erzwingen) anfügen, sodass der Befehl *Docker rm* jeden beliebigen Container entfernt, selbst einen, der gerade ausgeführt wird.

Windows- oder Linux-Container mithilfe von Windows PowerShell verwalten

Wie bereits weiter vorn erwähnt, ist es für das *Dockerd*-Modul nicht erforderlich, das Clientprogramm *Docker.exe* zu verwenden. Da es sich bei Docker um ein Open-Source-Projekt handelt, lässt sich auch eine alternative Clientimplementierung mit *Dockerd* verwenden, und in Zusammenarbeit mit der Docker-Gemeinde realisiert Microsoft genau dies mit einem PowerShell-Modul, mit dem Sie Docker-Container erstellen und verwalten können.

Weil sich das Docker-Modul für PowerShell noch im Entwicklungsstadium befindet, unterstützt es nicht unbedingt sämtliche Funktionen, die mit dem *Docker.exe*-Client möglich sind. Die wichtigsten Funktionen sind aber bereits realisiert, wie die folgenden Abschnitte zeigen.

Container auflisten

Eine Liste aller Container auf dem Host können Sie per Windows PowerShell mit dem Cmdlet *Get-Container* anzeigen, wie Abbildung 4–14 zeigt. Im Unterschied zum Befehl *Docker ps* zeigt das Cmdlet *Get-Container* sämtliche Container auf dem Host an, ob sie ausgeführt werden oder nicht.

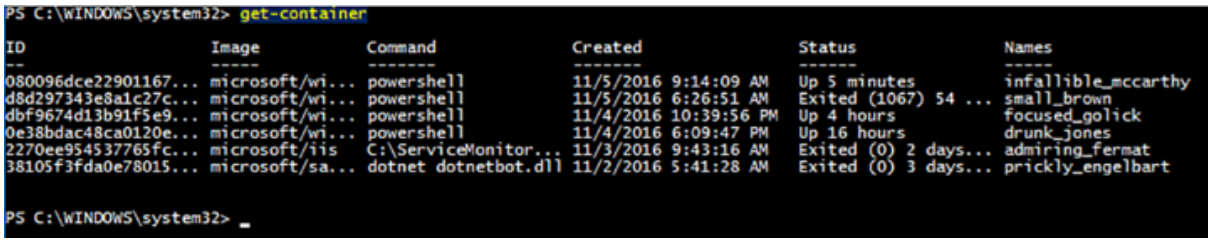

Abb. 4–14 Ausgabe des Cmdlets *Get-Container*

Container starten und anhalten

Wenn Sie einen Container mit dem Cmdlet *New-Container* erstellen, startet es den Container standardmäßig nicht. Vielmehr müssen Sie ihn explizit starten. Um einen angehaltenen Container zu starten, führen Sie das Cmdlet *Start-Container* wie im folgenden Beispiel aus:

```
start-container dbf9674d13b9
```

Einen Container können Sie auch anhalten, indem Sie einfach das Verb im Cmdlet zu *Stop-Container* wechseln:

```
stop-container dbf9674d13b9
```

Container anfügen

Um sich mit einer Sitzung auf einem ausgeführten Container zu verbinden, rufen Sie das Cmdlet *Enter-ContainerSession* wie im folgenden Beispiel auf:

```
Enter-containersession dbf9674d13b9
```

Dieses Cmdlet trägt auch den Aliasnamen *Attach-Container*, sodass Sie einen anderen Befehl wiederverwenden können, indem Sie lediglich ein Verb austauschen.

Images erstellen

Haben Sie einen Container in irgendeiner Weise geändert, können Sie die Änderungen in einem neuen Image speichern, indem Sie das Cmdlet *ConvertTo-ContainerImage* wie im folgenden Beispiel ausführen:

```
convertto-containerimage -containeridorname dbf9674d13b9

   -repository hholt/killerapp

   -tag 1.5
```

Dieses Cmdlet hat auch den Alias *Commit-Container*.

Container entfernen

Einen Container entfernen Sie mit dem Cmdlet *Remove-Container*, zum Beispiel:

```
remove-container dbf9674d13b9
```

Wie beim Befehl *Docker RM* müssen sich Container im angehaltenen Zustand befinden, bevor man sie entfernen kann. Mit dem Schalter *Force* entfernt der Cmdlet-Befehl jedoch jeden Container, selbst wenn er ausgeführt wird.

Containernetzwerke verwalten

Container können auf das externe Netzwerk zugreifen. Das lässt sich leicht nachweisen, indem man einen Server im lokalen Netzwerk oder im Internet anpingt. Wenn Sie allerdings den Befehl *Ipconfig /all* in einer Containersitzung ausführen (siehe Abbildung 4–15), sind Sie vielleicht vom Ergebnis überrascht.

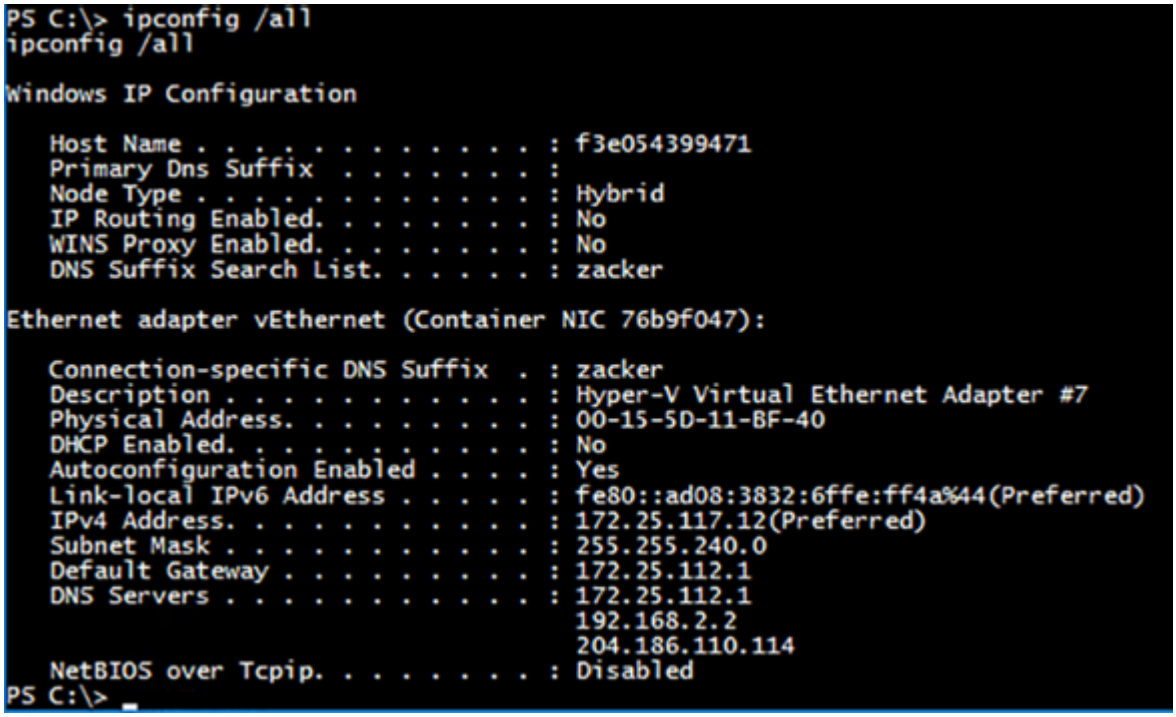

Abb. 4–15 Ausgabe des Befehls *Ipconfig /all* auf einem Container

In diesem Beispiel lautet die IP-Adresse des Netzwerkadapters im Container 172.25.117.12/12, was nichts weiter ist als die Adresse des Netzwerks, in dem sich der Containerhost befindet. Wenn Sie aber den Befehl *Ipconfig /all* auf dem Containerhost ausführen (siehe Abbildung 4–16), wird die Situation verständlicher.

Es gibt zwei Ethernet-Adapter, die im Containerhostsystem zu sehen sind. Der eine hat eine IP-Adresse im Netzwerk 192.168.2.0/24, die für das physische Netzwerk verwendet wird, mit dem der Containerhost verbunden ist. Der andere Adapter hat die Adresse 172.25.112.1/12, die im selben Netzwerk wie die Adresse des Containers liegt. Sehen Sie sich noch einmal die Konfiguration des Containers an: Die Adresse des Containerhosts ist als Standardgateway und DNS-Serveradresse für den Container aufgeführt. Der Containerhost fungiert letztlich als Router zwischen dem Netzwerk 172.16.0.0/12, in dem sich der Container befindet, und dem physischen Netzwerk 192.168.2.0/24, mit dem der Host verbunden ist. Darüber hinaus fungiert der Host als DNS-Server für den Container.

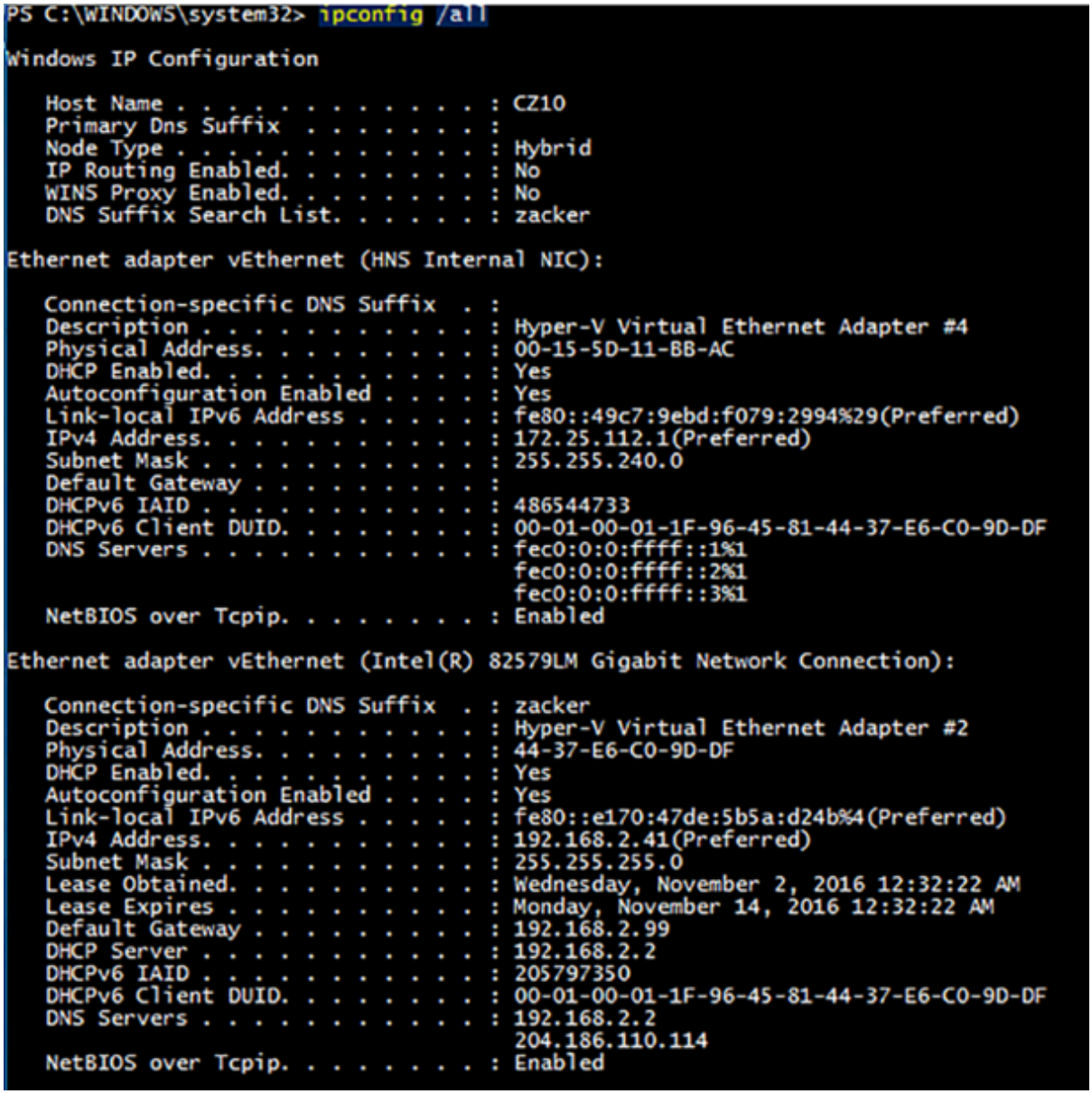

Abb. 4–16 Ausgabe des Befehls *Ipconfig /all* auf einem Containerhost

Sehen Sie sich einen anderen Container auf demselben Host an: Er hat eine IP-Adresse im selben Netzwerk wie der erste Container. Die beiden Container können die Adressen des jeweils anderen anpingen, aber auch solche von Systemen außerhalb des Netzwerks 172.16.0.0/12.

Das ist möglich, weil das Feature *Container* und das Paket Docker standardmäßig mit Netzwerkadressübersetzung (Network Address Translation, NAT) arbeiten, um eine Netzwerkumgebung für die Container auf dem Host zu schaffen. NAT ist ein Routingverfahren. Es ersetzt die IP-Adressen in den Netzwerkpaketen, die von einem System generiert und an ein System gerichtet werden, sodass die Adressen so aussehen, als würden sie sich in einem anderen Netzwerk befinden.

Wenn Sie einen Computer im Hostnetzwerk aus einer Containersitzung heraus anpingen, modifiziert der Containerhost die Ping-Pakete und ersetzt die Adresse des Containers 172.25.117.12 durch seine eigene Adresse 192.168.2.43 in jedem Paket. Treffen die Antworten vom angepingten System ein, läuft der Vorgang in umgekehrter Richtung ab.

Das *Dockerd*-Modul erstellt standardmäßig ein NAT-Netzwerk, wenn es zum ersten Mal ausgeführt wird, und weist jedem Container eine Adresse in diesem NAT-Netzwerk zu. Die Verwendung der Netzwerkadresse 172.16.0.0/12 ist zudem eine Standardeinstellung, die in Docker codiert ist. Diese Standardeinstellungen können Sie aber ändern, indem Sie eine andere NAT-Adresse festlegen oder NAT überhaupt nicht verwenden.

Die Netzwerkadapter in den Containern sind selbstverständlich virtuell. In der weiter vorn gezeigten Konfiguration ist zu erkennen, dass der Adapter für diesen Container als *vEthernet (Container NIC 76b9f047)* gekennzeichnet ist. Auf dem Containerhost gibt es ebenfalls einen virtuellen Adapter, hier *vEthernet (HNS Internal NIC)* genannt. HNS ist der Hostnetzwerkdienst (Host Network Service), der die von Docker verwendete NAT-Implementierung darstellt. Wie die Ausgabe des Cmdlets *Get-VMSwitch* auf dem Containerhost zeigt oder wie Sie im Hyper-V-Manager im Manager für virtuelle Switches (siehe Abbildung 4–17) sehen können, hat Docker ebenfalls einen virtuellen Switch namens *nat* erstellt. Dabei handelt es sich um den Switch, mit dem alle Adapter in den Containern verbunden sind. Was das Netzwerk angeht, ist also festzustellen, dass Container fast so wie virtuelle Computer arbeiten.

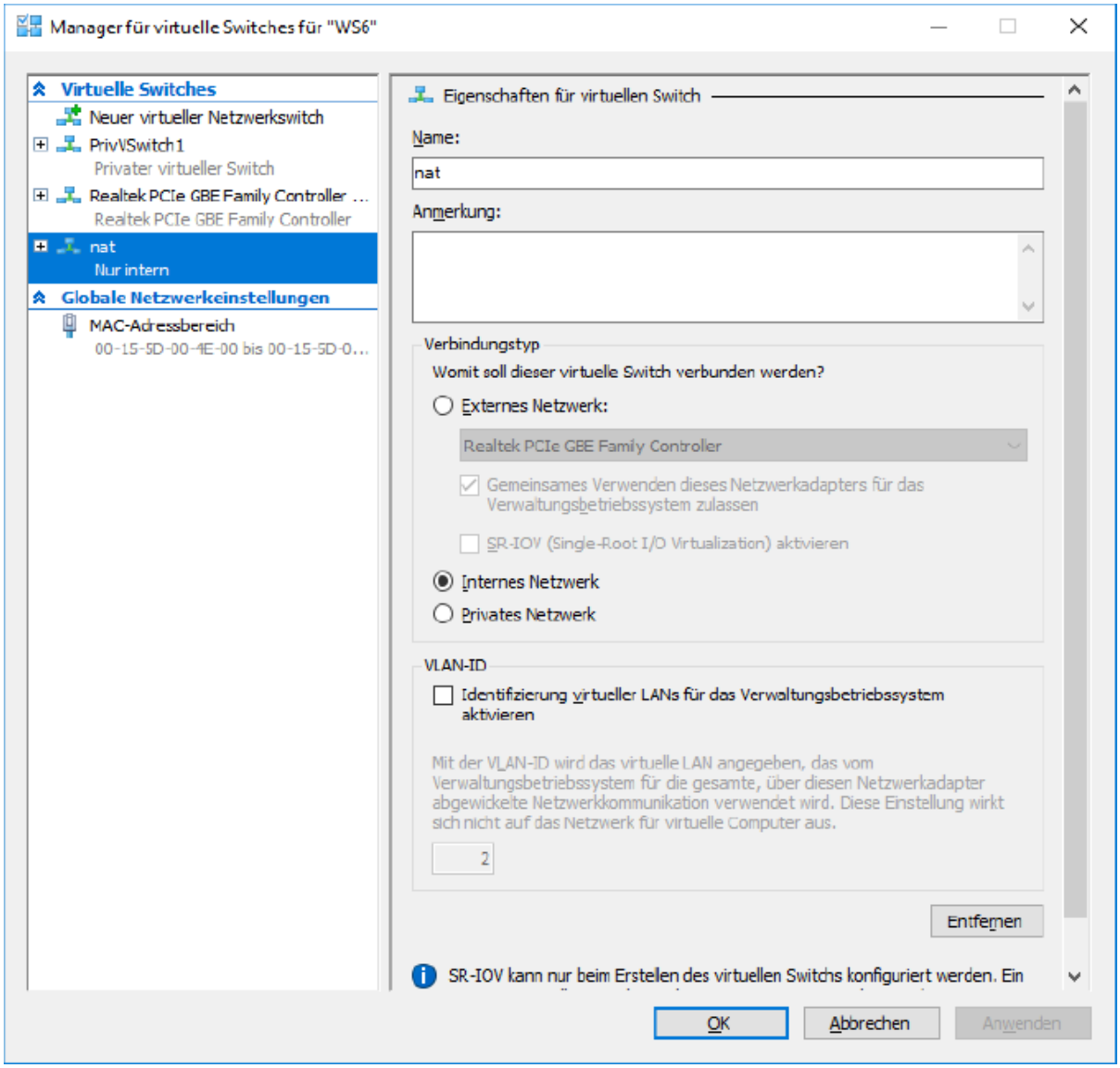

Abb. 4–17 Der Switch *nat* im Manager für virtuelle Switches

NAT-Standardeinstellungen anpassen

Wenn Sie für die NAT-Konfiguration von Docker eine andere Netzwerkadresse verwenden wollen, weil vielleicht ein Netzwerk mit derselben Adresse vorhanden ist, können Sie diese Adresse selbstverständlich ändern. Eine alternative Adresse legen Sie in der Konfigurationsdatei *daemon.json* fest, wie es bereits weiter vorn bei der Docker-Clientkonfiguration erläutert wurde.

Die Datei *daemon.json* ist eine einfache Textdatei, die Sie in dem Verzeichnis erstellen, in dem das Programm *Dockerd.exe* untergebracht ist. Eine alternative NAT-Netzwerkadresse legen Sie fest, indem Sie den folgenden Text in die Datei einfügen:

```
{ "fixed-cidr":"192.168.10.0/24" }
```

Für die NAT-Implementierung können Sie zwar beliebige Netzwerkadressen verwenden, doch um Adresskonflikte mit dem Internet zu verhindern, sollten Sie ein Netzwerk in einer der folgenden reservierten privaten Netzwerkadressen verwenden:

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16

Soll das *Dockerd*-Modul erst gar keine Netzwerkimplementierung erstellen, schreiben Sie den folgenden Text in die Datei *daemon.json*:

```
{ "bridge":"none" }
```

Wenn Ihre Container über Netzwerkkonnektivität verfügen sollen, müssen Sie in diesem Fall ein Containernetzwerk manuell einrichten.

Portzuordnungen

Wollen Sie in einem Container eine Serveranwendung ausführen, die Ports für eingehenden Clientverkehr offenlegt, sind sogenannte *Portzuordnungen* erforderlich. Damit kann der Containerhost, der den Clientverkehr empfängt, die Pakete an den passenden Port im Container, der die Anwendung ausführt, weiterleiten. Um Portzuordnungen zu verwenden, fügen Sie den Schalter *-p* an den Befehl *Docker Run* zusammen mit den Portnummern auf dem Containerhost bzw. dem Container wie im folgenden Beispiel an:

```
docker run -it

   -p 8080:80 microsoft\windowsservercore powershell
```

In diesem Beispiel wird jeder Datenverkehr, der über den Port 8080 des Containerhosts eintrifft, an den Port 80 des Containers weitergeleitet. Port 80 ist

der bekannte Port für Webserververkehr und diese Anordnung versetzt den Container in die Lage, diesen Standardport zu nutzen, ohne ihn auf dem Containerhost vollkommen zu beanspruchen, der Port 80 möglicherweise für seinen eigenen Webserver benötigt.

Ein transparentes Netzwerk erstellen

Anstatt NAT zu verwenden, können Sie auch ein transparentes Netzwerk erstellen, in dem die Container mit demselben Netzwerk wie der Containerhost verbunden sind. Wenn der Containerhost ein physischer Computer ist, sind die Container mit dem physischen Netzwerk verbunden. Ist der Containerhost ein virtueller Computer, sind die Container mit dem virtuellen Switch verbunden, den der virtuelle Computer verwendet.

Docker erstellt standardmäßig kein transparentes Netzwerk. Folglich müssen Sie es mit dem Befehl *Docker Network Create* erstellen, wie das folgende Beispiel zeigt:

```
docker network create -d transparent trans
```

Der Befehl in diesem Beispiel erstellt ein Netzwerk mithilfe des transparenten Treibers, der durch den Schalter *-d* gekennzeichnet wird, und weist ihm den Namen *trans* zu. Der folgende Befehl zeigt eine Liste aller Containernetzwerke an, die jetzt auch das eben erstellte Netzwerk *trans* enthält (siehe Abbildung 4–18).

```
docker network ls
```

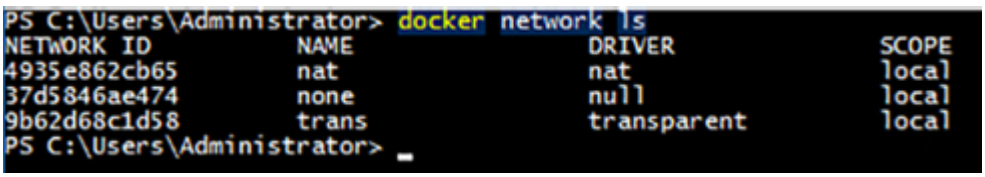

Abb. 4–18 Ausgabe des Befehls *Docker Network LS*

Ist das transparente Netzwerk erstellt, können Sie Container anlegen, die das Netzwerk verwenden. Dazu fügen Sie den Parameter *network* in den Befehl *Docker Run* wie im folgenden Beispiel hinzu:

```
docker run -it

    –network=trans microsoft/windowsservercore powershell
```

In diesem Container ausgeführt, zeigt der Befehl *Ipconfig /all*, dass der Container eine IP-Adresse im Netzwerk 10.0.0.0/24 besitzt, d. h. im selben Netzwerk, das der virtuelle Computer verwendet, der als Containerhost fungiert.

Beim Erstellen eines transparenten Netzwerks und der Container, die es verwenden, bekommen alle Teilnehmer die IP-Adressen von einem DHCP-Server im Containerhostnetzwerk zugeteilt. Ist jedoch kein DHCP-Server verfügbar, müssen Sie die Netzwerkadresseinstellungen festlegen, wenn Sie das Netzwerk erstellen, und die IP-Adressen aller Container manuell konfigurieren, indem Sie sie auf der *Docker Run*-Befehlszeile angeben.

Ein transparentes Netzwerk mit statischen IP-Adressen bauen Sie mit einem Befehl wie dem folgenden auf:

```
docker network create -d transparent

    –subnet=10.0.0.0/24

    –gateway=10.0.0.1

    trans
```

Um dann einen Container mit einer statischen IP-Adresse in diesem Netzwerk einzurichten, rufen Sie einen *Docker Run*-Befehl wie den folgenden auf:

```
docker run -it

    –network=trans

    –ip=10.0.0.16

    –dns=10.0.0.10 microsoft/windowsservercore powershell
```

Container-Datenvolumes verwalten

In manchen Fällen ist es erforderlich, Datendateien über Container hinweg zu bewahren. In Docker ist dies möglich, indem Sie Datenvolumes auf einem Container erzeugen, die einem Ordner auf dem Containerhost entsprechen. Nachdem Sie das Datenvolume auf dem Container erstellt haben, sind die Daten, die Sie dort speichern, auch im korrespondierenden Ordner auf dem Containerhost zu finden. Umgekehrt gilt das ebenso: Sie können Dateien in den Ordner auf dem Host kopieren und im Container darauf zugreifen.

Datenvolumes bestehen unabhängig vom Container. Wenn Sie den Container löschen, verbleibt das Datenvolume auf dem Containerhost. Dann können Sie den Containerhostordner in einem anderen Container bereitstellen, sodass Sie Ihre Daten über mehrere Iterationen einer Anwendung, die in Ihren Containern läuft, beibehalten können.

Um ein Datenvolume zu erstellen, fügen Sie den Schalter *-v* an einen *Docker Run*-Befehl an, wie zum Beispiel:

```
docker run -it

   -v c:\appdata microsoft/windowsservercore

   powershell
```

Dieser Befehl erstellt einen Ordner *C:\appdata* im neuen Container und verknüpft ihn mit einem Unterordner in *C:\ProgramData\docker\volumes* auf dem Containerhost. Um den genauen Speicherort zu erfahren, führen Sie den folgenden Befehl aus und sehen sich den Abschnitt *Mounts* an (siehe Abbildung 4–19):

```
docker inspect dbf9674d13b9
```

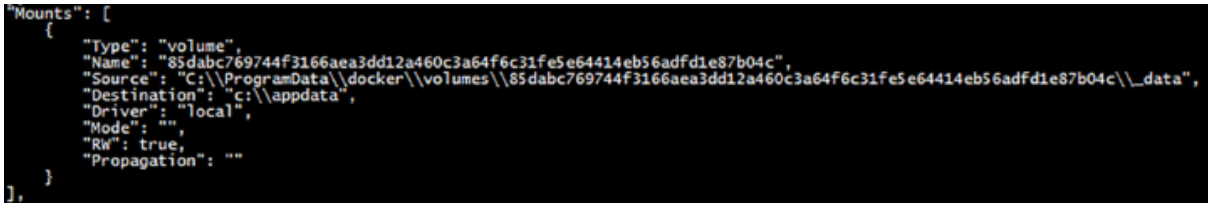

Abb. 4–19 Ein Teil der Ausgabe des Befehls *Docker Inspect*

Der Abschnitt *Mounts* (Bereitstellungen), der nur ein kleiner Teil einer langen und umfangreichen Auflistung der Spezifikationen des Containers ist, enthält die Eigenschaften *Source* (Quelle) und *Destination* (Ziel). Die Eigenschaft *Destination* gibt den Ordnernamen im Container an und *Source* ist der Ordner auf dem Containerhost. Um ein Datenvolume wiederzuverwenden, können Sie die Ordner sowohl für die Quelle als auch für das Ziel im *Docker Run*-Befehl wie im folgenden Beispiel angeben:

```
docker run -it

   -v c:\sourcedata:c:\appdata microsoft/windowsservercore
   powershell
```

Wenn Sie ein Datenvolume erstellen und dabei auf dem Container einen Ordner angeben, der bereits Dateien enthält, wird der vorhandene Inhalt durch das Datenvolume überlagert, aber nicht gelöscht. Die betreffenden Dateien sind wieder zugänglich, wenn die Bereitstellung des Datenvolumes aufgehoben wird.

Docker erstellt Datenvolumes standardmäßig im Lese-/Schreibmodus. Um ein schreibgeschütztes (»read only«) Datenvolume anzulegen, fügen Sie *:ro* an den Ordnernamen des Containers an, wie es das folgende Beispiel zeigt:

```
docker run -it

   -v c:\appdata:ro microsoft/windowsservercore powershell
```

HINWEIS **Ein Datenvolume hinzufügen**

Um einem vorhandenen Container ein Datenvolume hinzuzufügen, haben Sie praktisch nur eine Möglichkeit: Sie speichern mit *Docker Commit* alle Änderungen, die Sie am vorhandenen Container

vorgenommen haben, in einem neuen Image und erstellen dann mit *Docker Run* einen neuen Container aus dem neuen Image, wobei Sie den Schalter *-v* angeben, um das Datenvolume hinzuzufügen.

Ressourcensteuerung verwalten

Wie bereits weiter vorn erwähnt, unterstützt der Befehl *Docker Run* viele Parameter und Schalter, von denen Sie in diesem Kapitel bereits einige kennengelernt haben. Zum Beispiel haben Sie gesehen, wie der Schalter *-it* einen interaktiven Container erstellt, der eine bestimmte Shell oder einen anderen Befehl ausführt. Um einen Container zu erstellen, der im Hintergrund – im sogenannten getrennten Modus – läuft, fügen Sie den Schalter *-d* an, wie es folgendes Beispiel zeigt:

```
docker run -d -p 80:80 microsoft/iis
```

Mit einem getrennten Container können Sie über Netzwerkverbindungen oder Dateisystemfreigaben interagieren. Es ist auch möglich, die Verbindung zum Container mit dem Befehl *Docker Attach* herzustellen.

Mit Containernamen arbeiten

Wenn Sie mit dem Befehl *Docker Run* einen Container erstellen, weist das *Dockerd*-Modul dem Container standardmäßig drei Bezeichner zu, wie Abbildung 4–20 zeigt:

- **Lange UUID** Eine 32-Bytes-Zeichenfolge, die mit 64 Hexadezimalziffern dargestellt wird, zum Beispiel: 0e38bdac48ca0120eff6491a7b9d1908e65180213b2c1707b924991ae8d1504f.
- **Kurze UUID** Die ersten sechs Bytes der langen UUID, dargestellt mit 12 Ziffern, zum Beispiel: 0e38bdac48ca.
- **Name** Ein zufällig gewählter Name, bestehend aus zwei Wörtern, die durch einen Unterstrich getrennt sind, zum Beispiel: *drunk_jones*.

Abb. 4–20 Ausgabe des Befehls *Docker ps —no-trunc command*

Mit jedem dieser drei Bezeichner können Sie den Container auf der Befehlszeile referenzieren. Aber Sie haben auch die Möglichkeit, dem Container einen selbst gewählten Namen zuzuweisen, wenn Sie den Container erstellen. Dazu fügen Sie auf der *Docker Run*-Befehlszeile den Parameter *name* wie im folgenden Beispiel hinzu:

```
docker run -it microsoft/windowsservercore

   powershell

   –name core1
```

Arbeitsspeicher begrenzen

Über die Parameter des *Docker Run*-Befehls können Sie festlegen, wie viel Arbeitsspeicher ein Container verwenden darf. Standardmäßig dürfen Containerprozesse so viel Hostarbeitsspeicher verwenden und auslagern, wie sie benötigen. Wenn Sie mehrere Container auf demselben Host oder eine speicherhungrige Anwendung auf dem Host selbst ausführen, müssen Sie gegebenenfalls Beschränkungen hinsichtlich des Arbeitsspeichers, den bestimmte Container verwenden können, festlegen.

Die folgenden *Docker Run*-Parameter beziehen sich auf den Arbeitsspeicher:

- **-m** (oder **—memory**) Gibt die Menge des Arbeitsspeichers an, den der Container verwenden darf. Die Werte bestehen aus einer Ganzzahl und dem Einheitenzeichen b, k, m oder g (für Bytes, Kilobytes, Megabytes und Gigabytes).
- **-memory-swap** Gibt die Gesamtmenge des Arbeitsspeichers und des virtuellen Speichers an, den der Container verwenden kann. Die Werte bestehen aus einer Ganzzahl und dem Einheitenzeichen b, k, m oder g.
- **-memory-reservation** Spezifiziert eine weiche Grenze für den Arbeitsspeicher, den der Host für den Container reserviert, selbst wenn um

Systemspeicher konkurriert wird. Zum Beispiel können Sie mit dem Schalter *-m* eine harte Grenze von 1 GB und eine Speicherreservierung von 750 MB festlegen. Wenn andere Container oder Prozesse zusätzlichen Arbeitsspeicher benötigen, kann der Host bis zu 250 MB vom Arbeitsspeicher des Containers anfordern, lässt aber mindestens 750 MB unberührt. Die Werte bestehen aus einer Ganzzahl, die kleiner als die im Schalter *-m* ist, oder dem *—memory-swap*-Wert und dem Einheitenzeichen b, k, m oder g.

- **-kernel-memory** Gibt an, dass die mit dem Schalter *-m* festgelegte Speichergrenze für Kernelspeicher gilt. Die Werte bestehen aus einer Ganzzahl und dem Einheitenzeichen b, k, m oder g.
- **-oom-kill-disable** Hindert den Kernel daran, Containerprozesse abzubrechen, wenn ein Fehler aufgrund von Speichermangel (Out Of Memory, OOM) auftritt. Verwenden Sie diese Option niemals ohne den Schalter *-m*, um ein Speicherlimit für den Container festzulegen. Andernfalls könnte der Kernel Prozesse auf dem Host beenden, wenn ein OOM-Fehler auftritt.

CPU-Zyklen begrenzen

Es sind auch Parameter vorgesehen, über die Sie die Anzahl der CPU-Zyklen begrenzen können, die einem Container zugeteilt werden. Standardmäßig teilen sich alle Container auf einem Host die verfügbaren Zyklen gleichmäßig untereinander. Mit den Zyklusparametern lassen sich den Containern Prioritäten zuweisen, die bei CPU-Auslastung wirksam werden.

Den Zugriff auf CPUs können Sie mit folgenden *Docker Run*-Parametern steuern:

- **-c** (oder **—cpu-shares**) Spezifiziert mit einem Wert von 0 bis 1024 das Gewicht des Containers im Rennen um CPU-Zyklen. Die tatsächliche Anzahl der Prozessorzyklen, die ein Container erhält, hängt von der Anzahl der Container, die auf dem Host laufen, und ihrer Gewichte ab.
- **-cpuset-cpus** Spezifiziert in einem Multiprozessor-Hostsystem, welche CPUs der Container verwenden kann. Die Werte bestehen aus Ganzzahlen, die durch Kommas voneinander getrennt sind und die CPUs im Hostcomputer darstellen.
- **-cpuset-mems** Gibt an, welche Knoten auf einem NUMA-Host der Container verwenden kann. Die Werte bestehen aus Ganzzahlen, die durch

Kommas voneinander getrennt sind und die CPUs im Hostcomputer darstellen.

Neue Container-Images mit Dockerfile erstellen

Wurde ein Container geändert, seit Sie ihn mit dem Befehl *Docker Run* erstellt haben, lassen sich diese Änderungen auch speichern. Das ist beispielsweise möglich, wenn Sie ein neues Container-Image mit dem Befehl *Docker Commit* erstellen. Empfohlen wird jedoch, Container-Images von Grund auf neu zu erstellen, und zwar mit einem als *Dockerfile* bezeichneten Skript.

Eine *Dockerfile-Datei* ist eine einfache Textdatei mit dem Namen *dockerfile*. Sie enthält die erforderlichen Befehle, um ein neues Image zu erstellen. Die fertiggestellte Dockerfile-Datei führen Sie dann mit dem Befehl *Docker Build* aus, um die neue Datei zu erzeugen. Bei Dockerfile handelt es sich einfach um eine Methode, die die Schritte automatisiert, mit denen Sie einen Container sonst manuell ändern würden. Wenn Sie den Befehl *Docker Build* mit der Dockerfile-Datei aufrufen, führt das *Dockerd*-Modul jeden Befehl im Skript aus, indem es einen Container erstellt, die angegebenen Änderungen vornimmt und mit einem *Docker Commit*-Befehl die Änderungen als neues Image speichert.

Eine Dockerfile-Datei besteht aus Befehlen wie FROM oder RUN und einer Anweisung für jeden Befehl. Es hat sich eingebürgert, die Befehle großzuschreiben. In das Skript können Sie Kommentare einfügen, die mit einem Nummernzeichen (#) eingeleitet werden.

Das folgende Beispiel zeigt eine einfache Dockerfile-Datei:

```
#install DHCP server

FROM microsoft/windowsservercore

RUN powershell

   -command install-windowsfeature dhcp

   -includemanagementtools
```

```
RUN powershell

   -configurationname microsoft.powershell

   -command add-dhcpserverv4scope

   -state active

   -activatepolicies $true

   -name scopetest

   -startrange 10.0.0.100

   -endrange 10.0.0.200

   -subnetmask 255.255.255.0

RUN md boot

COPY ./bootfile.wim c:/boot/ CMD powershell
```

Die Elemente in diesem Beispiel bedeuten:

- Der FROM-Befehl spezifiziert das Basisimage, von dem das neue Image erstellt wird. In diesem Fall beginnt das neue Image mit dem Image *microsoft/windowsservercore*.
- Der erste RUN-Befehl öffnet eine PowerShell-Sitzung und ruft das Cmdlet *Install-WindowsFeature* auf, um die Rolle DHCP zu installieren.
- Der zweite RUN-Befehl richtet mit dem Cmdlet *Add-DhcpServerv4Scope* einen neuen Bereich auf dem DHCP-Server ein.
- Der dritte RUN-Befehl legt ein neues Verzeichnis namens *boot* an.
- Der COPY-Befehl kopiert eine Datei *bootfile.wim* aus dem aktuellen Ordner auf dem Containerhost in den Ordner *C:\boot* auf dem Container.

- Der CMD-Befehl öffnet eine PowerShell-Sitzung, wenn das Image ausgeführt wird.

Nachdem Sie das *dockerfile*-Skript erstellt haben, erzeugen Sie mit dem Befehl *Docker Build* das neue Image wie im folgenden Beispiel:

```
docker build -t dhcp .
```

Dieser Befehl liest die Dockerfile-Datei aus dem aktuellen Verzeichnis und erstellt ein Image namens *dhcp*. Wenn das *Dockerd*-Modul das Image erstellt, zeigt es die Ergebnisse jedes Befehls und die IDs der temporär angelegten Container (siehe Abbildung 4–21). Nachdem Sie das Image erstellt haben, können Sie davon mit dem Befehl *Docker Run* in der üblichen Art und Weise einen Container einrichten.

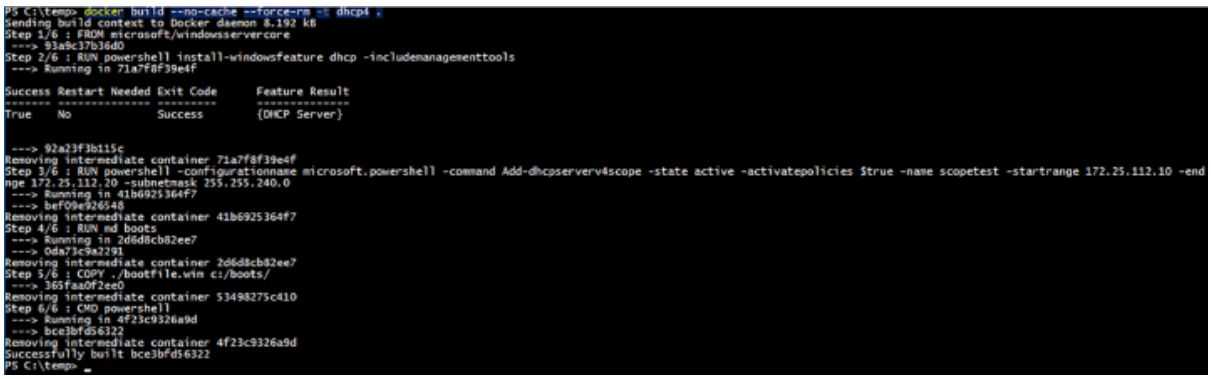

Abb. 4–21 Ausgabe des Befehls *Docker Build*

Die Dockerfile-Datei im Beispiel ist sehr einfach, doch können solche Dateien auch wesentlich länger und komplexer sein.

Schnelltest

Mit welchem der folgenden Docker-Befehle können Sie neue Container-Imagedateien erstellen?

1. Docker Run
2. Docker Commit
3. Docker Build
4. Docker Images

Antwort für den Schnelltest

Die Antworten 2 und 3 sind richtig. Mit dem Befehl *Docker Commit* erzeugt man ein neues Image aus einem vorhandenen Container. Der Befehl *Docker Build* erzeugt ein neues Container-Image entsprechend den Befehlen in einer Dockerfile-Datei.

Container-Images mit DockerHub-Repository für öffentliche und private Szenarien verwalten

DockerHub ist ein öffentliches Repository, mit dem Sie Container-Images speichern und verteilen können. Wenn Sie Container-Images mit dem Befehl *Docker Pull* herunterladen, kommen sie standardmäßig von DockerHub, sofern Sie im Befehl kein anderes Repository angeben. Mit dem Befehl *Docker Push* können Sie Images auch hochladen.

Wenn Sie Images zu DockerHub hochladen, können Sie sie mit Ihren Kollegen und sogar mit sich selbst teilen, sodass Sie Dateien nicht manuell übertragen müssen, um ein Container-Image auf einem anderen Host bereitzustellen.

Bevor Sie Images auf DockerHub hochladen können, müssen Sie sich auf der Site *http://hub.docker.com* registrieren. Nachdem das erledigt ist, wird Ihr Benutzername zum Namen Ihres Repositorys. Zum Beispiel ist das Image *microsoft/windowsservercore*, das Sie vorher heruntergeladen haben, ein Image namens *windowsservercore* im Microsoft-Repository. Lautet Ihr DockerHub-Benutzername *hholt*, beginnen alle Ihre Images mit diesem Repository-Namen, gefolgt vom Imagenamen, zum Beispiel:

```
hholt/nano1
```

Sobald Sie über ein Konto verfügen, müssen Sie sich beim DockerHub-Dienst von der Befehlszeile aus anmelden, bevor Sie Images hochladen können. Hierfür verwenden Sie den folgenden Befehl:

```
docker login
```

Docker fragt Ihren Benutzernamen und das Kennwort ab und bietet dann Upload-Zugriff auf Ihr Repository.

Nach Images suchen

DockerHub können Sie nach Images durchsuchen. Dazu gehen Sie auf die Website, wie sie Abbildung 4–22 zeigt. Diese Oberfläche bietet die neuesten Informationen über das Image sowie Kommentare von anderen Benutzern in der Docker-Community.

Abb. 4–22 Screenshot einer DockerHub-Websuche

Den DockerHub können Sie auch von der Befehlszeile aus durchsuchen, und zwar mit dem Befehl *Docker Search*, zum Beispiel:

```
docker search microsoft –no-trunc
```

Der Parameter *no-trunc* verhindert, dass der Befehl die Imagebeschreibungen abschneidet (siehe Abbildung 4–23).

Abb. 4–23 Ausgabe des Befehls *Docker Search*

Images hochladen

Möchten Sie Ihre Images in das Repository hochladen, rufen Sie den Befehl *Docker Push* wie im folgenden Beispiel auf:

```
docker push hholt/nano1
```

Standardmäßig lädt der Befehl *Docker Push* das angegebene Image in Ihr öffentliches Repository auf dem DockerHub hoch, wie Abbildung 4–24 zeigt. Somit kann jeder auf die hochgeladenen Images zugreifen.

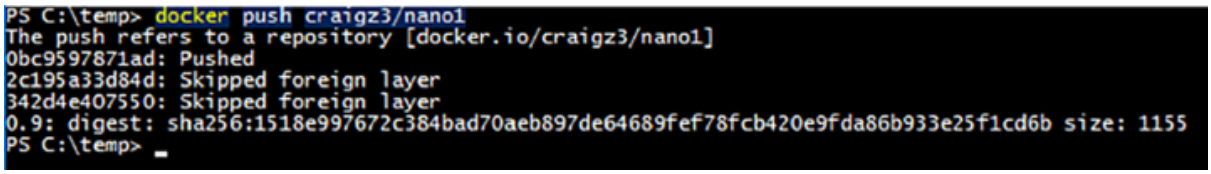

Abb. 4–24 Ausgabe des Befehls *Docker Push*

Weil Docker Open-Source-Software ist, macht das Freigeben von Images und Code für die Community einen großen Teil der Firmenphilosophie aus. Es ist aber auch möglich, private Repositories zu erstellen, die Sie für eine unbegrenzte Anzahl von ausgewählten Mitarbeitern freigeben können. Dadurch lässt sich DockerHub auch für sichere Projekte der Anwendungsentwicklung einsetzen oder allgemein für jede Situation, in der Sie ein Image nicht für die Öffentlichkeit bereitstellen wollen. DockerHub bietet ein einzelnes privates Repository im Rahmen des kostenlosen Dienstes, für zusätzliche Repositories ist ein kostenpflichtiges Abonnement erforderlich.

Außer dem Speichern und Bereitstellen von Images bietet DockerHub noch weitere Dienste, beispielsweise automatisierte Builds. Wenn Sie eine Dockerfile-Datei und alle anderen erforderlichen Dateien in ein Repository hochladen,

können Sie DockerHub so konfigurieren, dass Builds nach Ihren konkreten Spezifikationen automatisch ausgeführt werden. Die Codedateien sind für Ihre Mitarbeiter zugänglich und neue Builds können stattfinden, sobald sich der Code geändert hat.

Container-Images mit Microsoft Azure verwalten

Container können Sie nicht nur lokal erstellen, sondern sie auch auf Microsoft Azure verwenden. Wenn Sie einen virtuellen Computer mit Windows Server 2016 auf Azure einrichten, können Sie Container genauso wie auf einem lokalen Server erstellen und verwalten. Darüber hinaus bietet Azure den Dienst Azure Container Service (ACS), mit dem Sie einen Cluster von virtuellen Computern erstellen, konfigurieren und verwalten können, um Container-basierte Anwendungen mithilfe verschiedener Open-Source-Technologien auszuführen.

Microsoft Azure ist ein abonnementbasierter Clouddienst, der Sie in die Lage versetzt, virtuelle Computer und Anwendungen bereitzustellen und sie in Ihr vorhandenes Unternehmen zu integrieren. Für eine monatliche Gebühr können Sie einen virtuellen Windows Server 2016-Computer erstellen (siehe Abbildung 4–25). Nachdem Sie den virtuellen Computer eingerichtet haben, können Sie das Feature *Container* und das Modul *Docker* installieren. Container und Images, die Sie auf einem virtuellen Azure-Computer installieren, sind vollkommen kompatibel mit den Docker-Implementierungen auf Ihren lokalen Computern.

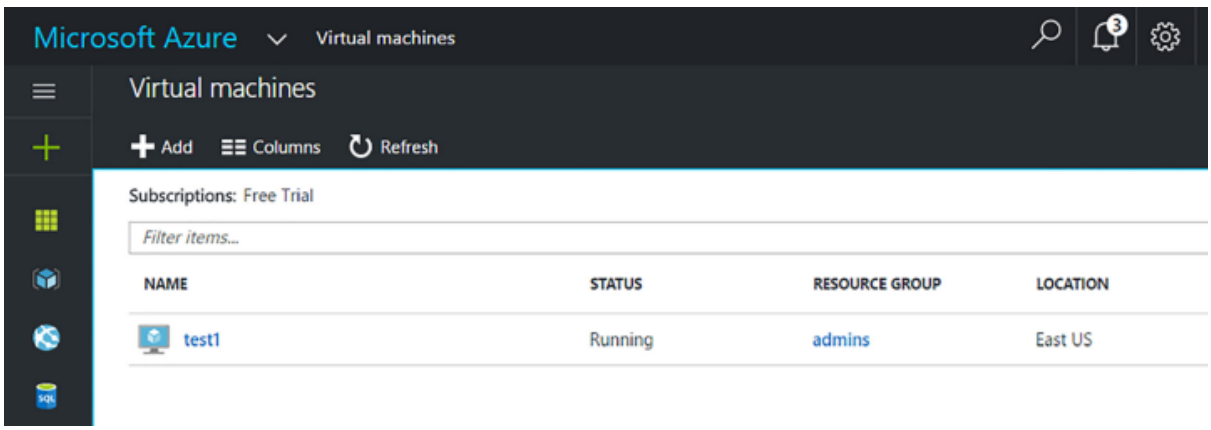

Abb. 4–25 Microsoft Azure Resource Center

Kapitelzusammenfassung

- Container basieren auf Images. Um einen Container zu erstellen, führen Sie ein Image aus, und Sie erstellen ein Image, indem Sie den Inhalt eines Containers speichern.
- Windows Server 2016 enthält das Feature *Container*, das die Unterstützungsumgebung für die Docker-Plattform liefert.
- Sowohl die Server Core- als auch die Nanoserver-Installationsoptionen unterstützen das Erstellen von Windows Server- und Hyper-V-Containern. In Nanoserver können Sie den *Docker.exe*-Client auf einem Remotesystem ausführen.
- Docker ist eine Open-Source-Container-Lösung, die aus zwei Dateien besteht: *Dockerd.exe* ist das Modul, das als Dienst in Windows läuft, und *Docker.exe* ist der Befehlszeilenclient, der das *Dockerd*-Modul steuert.
- Mit einer Textdatei namens *daemon.json* können Sie die Startoptionen für das *Dockerd*-Modul konfigurieren.
- Der Docker-Client ist nicht die einzige Möglichkeit, das Docker-Modul zu steuern. Die gleichen Aufgaben können Sie auch mit dem Docker-Modul für Windows PowerShell durchführen.
- Mit dem Befehl *Docker Pull* lassen sich Images vom DockerHub herunterladen.
- Tags (Markierungen) sind Versionsindikatoren, mit denen Entwickler die Builds oder Versionen eines Container-Images nachverfolgen können. Tag-Werte weisen Sie mit dem Befehl *Docker Tag* zu.
- Ein Container-Image lässt sich mit dem Befehl *Docker RMI* deinstallieren.
- Um einen Windows Server-Container zu erstellen, führen Sie den Befehl *Docker Run* mit dem Namen eines Container-Images aus.
- Der Ablauf für das Erstellen eines Hyper-V-Containers mithilfe von Docker unterscheidet sich vom Ablauf für Windows Server-Container nur darin, dass der Parameter *—isolation* anzugeben ist.
- Der *Docker.exe*-Client erlaubt es Ihnen, Container zu steuern, indem Sie sie starten, anhalten, sichern und entfernen.

- Das Docker-Modul für Windows PowerShell bietet eine Alternative zum *Docker.exe*-Client und kann die meisten (wenn nicht sogar alle) Funktionen ausführen.
- Standardmäßig verwendet Docker Netzwerkadressübersetzung (NAT), um den Netzwerkzugriff für Container bereitzustellen. Allerdings können Sie die Standardeinstellungen überschreiben und Container als Teil Ihres größeren Netzwerks konfigurieren.
- Mit Docker haben Sie die Möglichkeit, Datenvolumes zu erstellen, die auf dem Containerhost existieren, und sie einem Container hinzuzufügen. Datenvolumes verbleiben auf dem Containerhost, selbst wenn Sie den Container löschen.
- Mit Parametern auf der Befehlszeile von *Docker Run* können Sie die Arbeitsspeicher- und CPU-Ressourcen begrenzen, die ein Container nutzen darf.
- Eine Dockerfile-Datei ist ein Skript, das Befehle für das Erstellen eines neuen Container-Images enthält. Mit dem Befehl *Docker Build* führen Sie das Skript aus und erzeugen das Image.
- DockerHub ist ein kostenloses, cloudbasiertes Repository, in das Sie Ihre Container-Images hochladen können.
- Microsoft Azure ermöglicht es, virtuelle Computer zu erstellen, die Sie als Containerhosts verwenden können.

Gedankenexperiment

In diesem Gedankenexperiment wenden Sie Ihre Fähigkeiten und Kenntnisse an, die Sie sich im Rahmen dieses Kapitels angeeignet haben. Die Antwort zu diesem Gedankenexperiment finden Sie im nächsten Abschnitt.

Ralph möchte einen virtuellen Computer namens Core1 erstellen, der als Containerhost sowohl für Windows Server- als auch für Hyper-V-Container fungiert. Um den Containerhost zu erstellen, plant er, die folgenden Aufgaben auszuführen:

- Einen virtuellen Computer erstellen
- Den virtuellen Computer mit 4 GB Arbeitsspeicher, zwei virtuellen Prozessoren und aktiviertem Spoofing von MAC-Adressen

konfigurieren

- Windows Server 2016 auf dem virtuellen Computer installieren
- Das Feature *Container* installieren
- Die Rolle *Hyper-V* installieren
- Das Modul *dockermsftprovider* installieren
- Das Paket *Docker* installieren
- Das Server Core-Image von DockerHub abrufen
- Container mit dem Befehl *Docker Run* erstellen

Welchen Schritt hat Ralph vergessen, was ihn daran hindert, die benötigten Container zu erstellen? Welche Aufgabe muss er ausführen, um seinen Plan fertigzustellen, und wann sollte er ihn fertigstellen?

Antwort für das Gedankenexperiment

Dieser Abschnitt enthält die Lösung für das Gedankenexperiment.

Ralph hat vergessen, die Virtualisierungserweiterungen des Prozessors im physischen Computer für den virtuellen Computer offenzulegen, damit er die Hyper-V-Rolle ausführen kann. Hierzu muss er den folgenden Befehl in einer PowerShell-Sitzung ausführen, nachdem er den virtuellen Computer erstellt hat und bevor er ihn startet:

```
set-vmprocessor -vmname server1

   -exposevirtualizationextensions $true
```

Index

B

C

D

E

F

G

H