10

Exploring Windows Containers

This chapter covers the following recipes:

- Configuring a container host
- Deploying sample containers
- Deploying IIS in a container
- Using a Dockerfile to create a custom container

Introduction

Containers and container technology have been around on the Linux platform for some time and can also be run on Windows (both Windows 10/11 and Windows Server 2022). Containers can assist in deploying applications. The opensource Docker initiative popularized containers. Windows Server 2022 supports both Docker and Docker containerization integrated with Hyper-V.

With containers in Windows Server 2022, you perform most administration tasks not by using PowerShell cmdlets but by using a command-line tool called `docker.exe`. For those used to PowerShell's object-oriented and task-focused approach, you may find this application hard to use. I daresay you are not alone. The `docker.exe` application works in PowerShell, and you can use PowerShell to wrap the command. As ever with command-line tools, you can wrap the command-line application with a PowerShell function to get the benefits of object orientation.

To use containers with Windows Server 2019, you must download and install several components, including container base images. These downloads require an internet connection and are not particularly small. So be sure to have lots of disk space.

Containers provide scalability by enabling you to run multiple containers directly on top of Windows Server 2022. Containers can take up considerably fewer resources than if you run each container in separate **virtual machines (VMs)**. In theory, running multiple containers on a single host could be a security risk because malware could enable bad actors to access the contents of one container from another unfriendly container. To reduce those risks, you can run containers virtualized inside Hyper-V. With Hyper-V containers, Windows runs the container inside a virtualized environment that provides additional hardware-level security, albeit at the price of performance. Hyper-V containers are also useful in a shared tenant environment, where one container host can run containers belonging to different organizations.

By default, you cannot run a container that uses one version of Windows Server within another version of Windows Server. So if you base a container on Windows Server 2019 Server Core, you cannot, by default, run it on top of Windows Server 2022. This does incur a small performance overhead when you start and stop the container. To get around this issue, you can run the container virtualized.

The first step in deploying containers in Windows Server 2022 is configuring your container host, including adding several downloaded components. In the *Configuring a container host* recipe, you configure a host, `CH1`, to run containers.

Once you have configured a container host, it's a great idea to test that you can run containers successfully. You can download many sample containers to test out the basic container functionality (and the use of `docker.exe`). You explore and download key base container images and sample containers in the *Deploying sample containers* recipe.

A common application that you can deploy using containers is **Internet Information Server (IIS)**. In the *Deploying IIS in a container* recipe, you download a container base image containing IIS and then run this image inside a container.

You can also build customized container images containing your applications, as seen in the *Using a Dockerfile to create a custom container* recipe.

The recipes in this chapter use the command-line tool `docker.exe`. For those familiar and comfortable with all of PowerShell's awesomeness, this may come as a bit of a shock. As you can observe, `docker.exe` has no tab completion, all output is minimal text blobs (no objects), parameter names seem random and curious, the online help is not very helpful, and the error reporting is downright atrocious. All in all, `docker.exe` takes time to get to grips with, is less easy to automate than other Windows features, and feels very, very slow even on a well-equipped workstation. But if you plan to use the awesome container features in Windows Server, consider spending some time building a good framework and framework tools for your environment. One example is the `Get-ContainerImage` function you create in the *Deploying IIS in a container* recipe.

It's also worth noting that if you use most major search engines to discover aspects of containers, searches tend to yield many useful pages. However, many pages focus on Linux as a container host and using tools and features not available under Windows. It can be a struggle to find good Windows-based documentation.

There is much more to explore with containers. This chapter only introduces containers, container images, `docker.exe`, and Dockerfile files. Topics including Docker networking, Docker Swarm, and more are outside the scope of this book. To discover more about containers than we can fit here, look at Packt's book: *Learning Windows Server Containers* by Srikanth Machiraju. And, for more on the endearingly awful `docker.exe` application, take a look at *Docker on Windows* by Elton Stoneman.

For more information on Windows containers, see this link: `https://docs.microsoft.com/virtualization/windowscontainers/about/`

The systems used in the chapter

In this chapter, you configure and manage containers on the Windows Server 2022 host `CH1`. You have installed PowerShell 7 (and VS Code) on this host. You also need the domain controller and DNS server for the Reskit.Org domain, `DC1`, online.

The hosts are as follows:

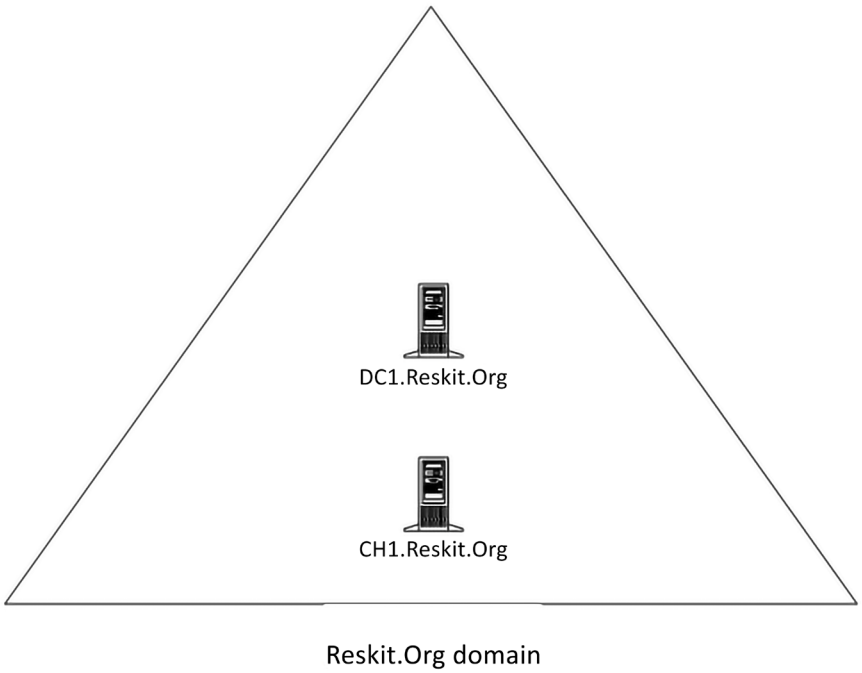

Figure 10.1: Host in use for this chapter

Configuring a container host

The first step in containerization is to configure a container host. The container host is a Windows system (virtual or physical). You have installed the Windows Server 2022 host with just PowerShell loaded.

Configuring the host requires you to add Windows features and download/install the Docker components you need to manage containers.

You can also run containers on Windows 11, and this recipe would mostly work on that platform but would need some modifications. These changes and containers on Windows 11 are outside the scope of this chapter.

Getting ready

You run this recipe on `CH1` after you have installed PowerShell 7 (and VS Code).

How to do it...

1. Installing the Docker provider module

```
$InstallHT1 = @{
  Name       = 'DockerMSFTProvider'
  Repository = 'PSGallery'
  Force      = $True
}
Install-Module @InstallHT1
```

2. Installing the latest version of the Docker package

```
$InstallHT2 = @{
  Name         = 'Docker'
  ProviderName = 'DockerMSFTProvider'
  Force        = $True
}
Install-Package @InstallHT2
```

3. Ensuring that the Hyper-V and related tools are added

```
Add-WindowsFeature -Name Hyper-V -IncludeManagementTools |
  Out-Null
```

4. Removing Defender, as it interferes with Docker

```
Remove-WindowsFeature -Name Windows-Defender |
  Out-Null
```

5. Restarting the computer to enable Docker and containers

```
Restart-Computer
```

6. Checking Windows containers and Hyper-V features are installed on `CH1`

```
Get-WindowsFeature -Name Containers, Hyper-v
```

7. Checking Docker service

```
Get-Service -Name Docker
```

8. Checking Docker version information

```
docker version
```

9. Displaying Docker configuration information

```
docker info
```

How it works...

In *step 1*, you download and install the Docker provider module for Windows. This step produces no console output. In *step 2*, you install the latest version of the Docker provider, which produces the following output:

```
PS C:\Foo> # 2. Installing the latest version of the docker package
PS C:\Foo> $InstallHT2 = @{
            Name         = 'Docker'
            ProviderName = 'DockerMSFTProvider'
            Force        = $True
           }
PS C:\Foo> Install-Package @InstallHT2

WARNING: A restart is required to enable the containers feature. Please restart your machine.

Name     Version   Source         Summary
----     -------   ------         -------
Docker   20.10.9   DockerDefault  Contains Docker EE for use with Windows Server.
```

Figure 10.2: Installing the Docker package

In *step 3*, you add the Hyper-V feature to `CH1` and add the Hyper-V management tools, producing output like this:

```
PS C:\Foo> # 2. Installing the latest version of the docker package
PS C:\Foo> $InstallHT2 = @{
            Name         = 'Docker'
            ProviderName = 'DockerMSFTProvider'
            Force        = $True
           }
PS C:\Foo> Install-Package @InstallHT2

WARNING: A restart is required to enable the containers feature. Please restart your machine.

Name     Version   Source         Summary
----     -------   ------         -------
Docker   20.10.9   DockerDefault  Contains Docker EE for use with Windows Server.
```

Figure 10.3: Adding Hyper-V Windows Server feature to CH1

The Windows Defender package can interfere with the operation of containers. For that reason, in *step 4*, you remove this feature. To complete the setup of the components added and removed so far, you need to reboot the machine, which you do in *step 5*. These two steps produce no console output.

After rebooting `CH1`, in *step 6*, you confirm that you have added the containers and Hyper-V features successfully, producing output like this:

```
PS C:\Foo> # 6. Checking Windows Containers and Hyper-V features are installed on CH1
PS C:\Foo> Get-WindowsFeature -Name Containers, Hyper-V

Display Name Name       Install State
------------ ----       -------------
             Hyper-V        Installed
             Containers     Installed
```

Figure 10.4: Checking the Hyper-V and Containers features on CH1

Within Windows Server, the Docker service does much of the work of running containers. In *step 7*, you verify that the Docker service is running. This step produces output like this:

```
PS C:\Foo> # 7. Checking Docker service
PS C:\Foo> Get-Service -Name Docker

Status   Name               DisplayName
------   ----               -----------
Running  Docker             Docker Engine
```

Figure 10.5: Checking the Hyper-V and Containers features on `CH1`

In *step 8*, you use the `docker.exe` command to display Docker's version information, with output like this:

```
PS C:\Foo> # 8. Checking Docker Version information
PS C:\Foo> docker version
Client: Mirantis Container Runtime
 Version:           20.10.9
 API version:       1.41
 Go version:        go1.16.12m2
 Git commit:        591094d
 Built:             12/21/2021 21:34:30
 OS/Arch:           windows/amd64
 Context:           default
 Experimental:      true

Server: Mirantis Container Runtime
 Engine:
  Version:          20.10.9
  API version:      1.41 (minimum version 1.24)
  Go version:       go1.16.12m2
  Git commit:       9b96ce992b
  Built:            12/21/2021 21:33:06
  OS/Arch:          windows/amd64
  Experimental:     false
```

Figure 10.6: Checking the Docker version information

In *step 9*, you display the Docker configuration details with output like this:

```
PS C:\Foo> # 9. Displaying Docker configuration information
PS C:\Foo> docker info
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  cluster: Manage Mirantis Container Cloud clusters (Mirantis Inc., v1.9.0)
  registry: Manage Docker registries (Docker Inc., 0.1.0)

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 7
 Server Version: 20.10.9
 Storage Driver: windowsfilter
  Windows:
 Logging Driver: json-file
 Plugins:
  Volume: local
  Network: ics internal l2bridge l2tunnel nat null overlay private transparent
  Log: awslogs etwlogs fluentd gcplogs gelf json-file local logentries splunk syslog
 Swarm: inactive
 Default Isolation: process
 Kernel Version: 10.0 20348 (20348.1.amd64fre.fe_release.210507-1500)
 Operating System: Windows Server 2022 Datacenter Version 2009 (OS Build 20348.169)
 OSType: windows
 Architecture: x86_64
 CPUs: 6
 Total Memory: 4.124GiB
 Name: CH1
 ID: VUDQ:UPUI:ZMI4:G53V:WTFT:VTZK:6YBB:SQI2:K34F:F7OZ:DXW7:NEJ2
 Docker Root Dir: C:\ProgramData\docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Live Restore Enabled: false
```

Figure 10.7: Displaying the Docker configuration

There's more...

In *step 2*, you install the `DockerMSFTProvider` package. This package also has the necessary Windows components to run containers. On Windows Server, this includes the containers Windows feature (as you check in *step 6*).

In *step 5*, you remove Defender. This is a simple fix to avoid issues that can arise when you test this recipe. For production, you should consult the Docker documentation at `https://docs.docker.com/engine/security/antivirus/`.

Deploying sample containers

Once you have a container host configured, you should check that you have set up your environment successfully and your container host can utilize containers. A really simple way to check that your container host is fully able to run containers is by downloading and running a simple container image. Docker publishes a useful hello-world container image you can use. Or you can download and run one of the standard OS images.

Before you can run a container, you must acquire a container image. There are several ways to obtain images, as you see in this chapter. Using the `docker` command, you can search and download images either to use directly or to use as the basis of a custom-built container. Docker maintains an online registry that contains a variety of container images for you to leverage.

This recipe demonstrates using the Docker registry to obtain images and then using those images locally. This recipe looks at some basic container management tasks and shows some methods to automate the `docker.exe` command.

Getting ready

You run this recipe in the container host, `CH1`. You set up this host in the *Configuring a container host* recipe.

How to do it...

1. Finding hello-world container images at the Docker hub

```
docker search hello-world
```

2. Pulling the Docker official hello-world image

```
docker pull hello-world
```

3. Checking the image just downloaded

```
docker image ls
```

4. Running the hello-world container image

```
docker run hello-world
```

5. Getting Server Core base image

```
$ServerCore = 'mcr.microsoft.com/windows/servercore:ltsc2022'
docker pull $ServerCore
```

6. Checking the images available now on `CH1`.

```
docker image ls
```

7. Running the `ServerCore` container image

```
docker run $ServerCore
```

8. Creating a function to get the Docker image details as objects

```
Function Get-DockerImage {
  # Getting the images
  $Images = docker image ls | Select-Object -Skip 1
  # Regex for getting the fields
  $Regex = '^(\S+)\s+(\S+)\s+(\S+)\s+([ \w]+)\s+(\S+)$'
  # Creating an object for each image and emit
  foreach ($Image in $Images) {
  $image -match $Regex  | Out-Null
  $ContainerHT = [ordered] @{
      Name    = $Matches.1
      Tag     = $Matches.2
      ImageId = $Matches.3
      Created = $Matches.4
      Size    = $Matches.5
    } # end hash table
    New-Object -TypeName pscustomobject -Property $ContainerHT
  } # end foreach
  } # end function
```

9. Inspecting `ServerCore` image

```
$ServerCoreImage = Get-DockerImage | Where-Object name -match
servercore
docker inspect $ServerCoreImage.ImageId | ConvertFrom-Json
```

10. Pulling a Server 2019 container image

```
$Server2019Image = 'mcr.microsoft.com/windows:1809'
docker pull $Server2019Image
```

11. Running older server image

```
docker run $Server2019Image
```

12. Running the image with `isolation`

```
docker run --isolation=hyperv $Server2019Image
```

13. Checking difference in run times with Hyper-V

```
# Running with no isolation
$Start1 = Get-Date
docker run hello-world |
    Out-Null
$End1 = Get-Date
$Time1 = ($End1-$Start1).TotalMilliseconds
# Running with isolation
$Start2 = Get-Date
docker run --isolation=hyperv hello-world | Out-Null
$End2 = get-date
$Time2 = ($End2-$Start2).TotalMilliseconds
# Displaying the time differences
"Without isolation, took : $Time1 milliseconds"
"With isolation, took    : $Time2 milliseconds"
```

14. Viewing system disk usage

```
docker system df
```

15. Viewing active containers

```
docker container ls -a
```

16. Removing active containers

```
$Actives = docker container ls -q -a
foreach ($Active in $actives) {
  docker container rm $Active -f
}
```

17. Removing all Docker images

```
docker rmi $(docker images -q) -f  | Out-Null
```

18. Removing other Docker detritus.

```
docker prune -f
```

19. Checking images and containers

```
docker image ls
docker container ls
```

How it works...

In *step 1*, you use the `docker.exe` search command to find hello-world containers at Docker hub. The output looks like this:

```
PS C:\Foo> # 1. Finding hello-world container images at the Docker Hub
PS C:\Foo> docker search hello-world
NAME                                      DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
hello-world                               Hello World! (an example of minimal Dockeriz…   1835      [OK]
kitematic/hello-world-nginx               A light-weight nginx container that demonstr…   152
tutum/hello-world                         Image to test docker deployments. Has Apache…   89                   [OK]
dockercloud/hello-world                   Hello World!                                    19                   [OK]
crccheck/hello-world                      Hello World web server in under 2.5 MB          15                   [OK]
vad1mo/hello-world-rest                   A simple REST Service that echoes back all t…   5                    [OK]
ppc64le/hello-world                       Hello World! (an example of minimal Dockeriz…   2
rancher/hello-world                                                                       2
ansibleplaybookbundle/hello-world-db-apb  An APB which deploys a sample Hello World! a…   2                    [OK]
thomaspoignant/hello-world-rest-json      This project is a REST hello-world API to bu…   1
ansibleplaybookbundle/hello-world-apb     An APB which deploys a sample Hello World! a…   1                    [OK]
strimzi/hello-world-consumer                                                              0
armswdev/c-hello-world                    Simple hello-world C program on Alpine Linux…   0
strimzi/hello-world-producer                                                              0
koudaiii/hello-world                                                                      0
businessgeeks00/hello-world-nodejs                                                        0
strimzi/hello-world-streams                                                               0
tacc/hello-world                                                                          0
garystafford/hello-world                  Simple hello-world Spring Boot service for t…   0                    [OK]
freddiedevops/hello-world-spring-boot                                                     0
tsepotesting123/hello-world                                                               0
okteto/hello-world                                                                        0
rsperling/hello-world3                                                                    0
dandando/hello-world-dotnet                                                               0
kevindockercompany/hello-world
```

Figure 10.8: Discovering hello-world container images

In *step 2*, you use the `docker.exe` command to pull the Docker official hello-world image with output like this:

```
PS C:\Foo> # 2. Pulling the Docker official hello-world image
PS C:\Foo> docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
2ebf439f800c: Pull complete
59d9f62c09b7: Pull complete
d6884bc3f6c7: Pull complete
Digest: sha256:7d246653d0511db2a6b2e0436cfd0e52ac8c066000264b3ce63331ac66dca625
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
```

Figure 10.9: Downloading the hello-world official Docker container from the Docker hub

In *step 3*, you use the `docker image` command to view the images that reside on `CH1`, with output like this:

```
PS C:\Foo> # 3. Checking the Image just downloaded
PS C:\Foo> docker image is
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
hello-world   latest    d4d88879abb0   3 weeks ago   297MB
```

Figure 10.10: Viewing Docker image on `CH1`

With the container image downloaded, in *step 4*, you run this image in the container to validate your container installation and configuration. The output looks like this:

```
PS C:\Foo> # 4. Running the hello-world container image
PS C:\Foo> docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (windows-amd64, nanoserver-ltsc2022)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run a Windows Server container with:
 PS C:\> docker run -it mcr.microsoft.com/windows/servercore:ltsc2022 powershell

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figure 10.11: Running the hello-world container image

In *step 5*, you download another container image, a Windows `ServerCore` image. The output from this step is as follows:

```
PS C:\Foo> # 5. Getting Server Core image base image
PS C:\Foo> docker pull mcr.microsoft.com/windows/servercore:ltsc2022
ltsc2022: Pulling from windows/servercore
97f65a0ec59e: Pull complete
97b25a378238: Pull complete
Digest: sha256:35c3cb29ef2c9f05e36070de4c79d7fc861c035fa5df2df64ae607a276db42c6
Status: Downloaded newer image for mcr.microsoft.com/windows/servercore:ltsc2022
mcr.microsoft.com/windows/servercore:ltsc2022
```

Figure 10.12: Pulling a Windows Server Core container image

In *step 6*, you check the container images on CH1, where you can see the newly downloaded server core container image. The output looks like this:

```
PS C:\Foo> # 6. Checking the images available now on CH1
PS C:\Foo> docker image is
REPOSITORY                              TAG         IMAGE ID       CREATED       SIZE
hello-world                             latest      d4d88879abb0   3 weeks ago   297MB
mcr.microsoft.com/windows/servercore    ltsc2022    5798b78d003a   4 weeks ago   5.08GB
```

Figure 10.13: Listing container images on CH1

In *step 7*, you run the container, returning output like this:

```
PS C:\Foo> # 7. Running the servercore container image
PS C:\Foo> docker run $ServerCore
Microsoft Windows [Version 10.0.20348.887]
(c) Microsoft Corporation. All rights reserved.
```

Figure 10.14: Executing the servercore container image

The `docker.exe` command produces simple string output, unlike PowerShell cmdlets, which produce rich objects. One way to overcome that limitation is to create objects on the fly using a `wrapper` function.

In *step 8*, you create a new PowerShell function (`Get-DockerImage`), which gets the images but returns the information as custom objects. There is no output directly from this step. Within this step, you create a hash table, which you then convert to a custom object. You could have created the object directly – as always you have choices.

In *step 9*, you use the `Get-DockerImage` function to get a specific container image, then you do a detailed examination of this container image, with output like this:

```
PS C:\Foo> # 9. Inspecting Server Core Image
PS C:\Foo> $ServerCoreImage = Get-DockerImage | Where-Object name -match servercore
PS C:\Foo> docker inspect $ServerCoreImage.ImageId | ConvertFrom-Json

Id              : sha256:5798b78d003a0eb4c52ddc590a333254e974bdc400f262bd7b4442bb2c6e49a2
RepoTags        : {mcr.microsoft.com/windows/servercore:ltsc2022}
RepoDigests     : {mcr.microsoft.com/windows/servercore@sha256:
                   35c3cb29ef2c9f05e36070d04c79d7fc861c035fa5df2df64ae607a276db42c6}
Parent          :
Comment         :
Created         : 06/08/2022 02:59:35
Container       :
ContainerConfig : @{Hostname=; Domainname=; User=; AttachStdin=False; AttachStdout=False;
                    AttachStderr=False; Tty=False; OpenStdin=False; StdinOnce=False; Env=;
                    Cmd=; Image=; Volumes=; WorkingDir=; Entrypoint=; OnBuild=; Labels=}
DockerVersion   :
Author          :
Config          : @{Hostname=; Domainname=; User=; AttachStdin=False; AttachStdout=False;
                   AttachStderr=False; Tty=False; OpenStdin=False; StdinOnce=False; Env=;
                   Cmd=System.Object[]; Image=; Volumes=;
                  WorkingDir=; Entrypoint=; OnBuild=; Labels=}
Architecture    : amd64
Os              : windows
OsVersion       : 10.0.20348.887
Size            : 5083872027
VirtualSize     : 5083872027
GraphDriver     : @{Data=; Name=windowsfilter}
RootFS          : @{Type=layers; Layers=System.Object[]}
Metadata        : @{LastTagTime=01/01/0001 00:00:00}
```

Figure 10.15: Inspecting the server core image

In *step 10*, you pull a Windows Server image for an older version (Windows Server 2019):

```
PS C:\Foo> # 10. Pulling a Server 2019 container image
PS C:\Foo> $Server2019Image = imcr.microsoft.com/windows:1809'
PS C:\Foo> docker pull $Server2019Image
1809: Pulling from windows
b079fa252589: Pull complete
3100d4854554: Pull complete
Digest: sha256:14241ad3587eb63e81c07e227adfc5b1ee4702d5b047599886fd82144210c479
Status: Downloaded newer image for mcr.microsoft.com/windows:1809
mcr.microsoft.com/windows:1809
```

Figure 10.16: Inspecting the server core image

In *step 11*, you attempt to run this downloaded image with the following output:

```
PS C:\Foo> # 11. Running older server image
PS C:\Foo> docker run $Server2019Image
docker: Error response from daemon: hcsshim::CreateComputeSystem
        3c8495f8debb5bf0cb39d141dbf2ba85c20e4c94b1c465e7228331dacf5de2b3:
        The container operating system does not match the host operating system.
```

Figure 10.17: Running a Windows Server 2019 image natively

Since the Windows kernel in the container host is different from the kernel in the container image, Windows cannot natively run the Windows Server 2019 images. You can work around this by using Hyper-V isolation, like this:

```
PS C:\Foo> # 12. run it with isolation
PS C:\Foo> PS C:\Foo> docker run --isolation=hypery $Server2019Image
Microsoft Windows [Version 10.0.17763.3287]
(c) 2018 Microsoft Corporation. All rights reserved.
```

Figure 10.18: Running a Windows Server 2019 image inside Hyper-V

While Hyper-V isolation can help with incompatible kernel version issues, there is a performance hit with using virtualization. In *step 13*, you run the hello-world container image natively and in Hyper-V. Then the step displays how to Windows run the container with and without virtualization. The output looks like this:

```
PS C:\Foo> # 13. Checking difference in run times with Hyper-V
PS C:\Foo> # Running with no isolation
PS C:\Foo> $Start1 = Get-Date
PS C:\Foo> docker run hello-world | Out-Null
PS C:\Foo> $End1 = Get-Date
PS C:\Foo> $Time1 = ($End1-$Start1).TotalMilliseconds
PS C:\Foo> # Running with isolation
PS C:\Foo> $Start2 = Get-Date
PS C:\Foo> docker run --isolation=hyperv hello-world | Out-Null
PS C:\Foo> $End2 = get-date
PS C:\Foo> $Time2 = ($End2-$Start2).TotalMilliseconds
PS C:\Foo> # Displaying the time differences
PS C:\Foo> "Without isolation, took : $Time1 milliseconds"
PS C:\Foo> "With isolation, took    : $Time2 milliseconds"
Without isolation, took : 2989.7237 milliseconds
With isolation, took    : 5881.3702 milliseconds
```

Figure 10.19: Measuring the performance impact of running a container inside Hyper-V

In *step 14*, you use the `docker system` command to view the disk usage for containers on `CH1` with output like this:

```
PS C:\Foo> # 14. Viewing system disk usage
PS C:\Foo> docker system dF
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          3         3         21.14GB   0B (0%)
Containers      5         0         0B        0B
Local Volumes   0         0         0B        0B
Build Cache     0         0         0B        0B
```

Figure 10.20: Viewing container image disk space usage

In *step 15*, you view the containers currently active on `CH1`, with output like this:

```
PS C:\Foo> # 15. Viewing active containers
PS C:\Foo> docker container ls -a
CONTAINER ID   IMAGE                                          COMMAND                  CREATED         STATUS                     PORTS  NAMES
13e6b90c2a2a   hello-world                                    "cmd /C 'type C:\\hel…"  4 minutes ago   Exited (0) 4 minutes ago          dreamy_cannon
8287cd70030b   hello-world                                    "cmd /C 'type C:\\hel…"  4 minutes ago   Exited (0) 4 minutes ago          objective_ardinghelli
6c0cb5e6505f   mcr.microsoft.com/windows:1809                 "c:\\windows\\system32…" 8 minutes ago   Exited (0) 7 minutes ago          condescending_hugle
3c8495f8debb   mcr.microsoft.com/windows:1809                 "c:\\windows\\system32…" 9 minutes ago   Created                           kind_bassi
c42a3e2b23aa   mcr.microsoft.com/windows/servercore:ltsc2022  "c:\\windows\\system32…" 47 minutes ago  Exited (0) 46 minutes ago         vigilant_swirles
```

Figure 10.21: Viewing active containers on `CH1`

In *step 16*, you remove all active containers on `CH1`. In *step 17*, you remove all Docker images from `CH1`. These two steps produce no console output. In *step 18*, you use the `prune` command to remove all other Docker-related resources with output like this:

```
PS C:\Foo> # 18. Removing other docker detritus
PS C:\Foo> docker system prune -f
Total reclaimed space: OB
```

Figure 10.22: Using the Docker prune command

In *step 19*, you verify that you have removed all containers and container images from `CH1`, with output like this:

```
PS C:\Foo> # 19. Checking images and containers
PS C:\Foo> docker image is

REPOSITORY    TAG       IMAGE ID    CREATED    SIZE

PS C:\Foo> docker container is

CONTAINER ID    IMAGE      COMMAND    CREATED    STATUS     PORTS      NAMES
```

Figure 10.23: Checking on remaining containers and container images

There's more...

In *step 4,* you run the hello-world container you downloaded in *step 2*. When Windows runs this container, the container prints out some text, then exits. These sample container images are a great demonstration that your container host is up, running, and able to host containers.

`Docker.exe` returns the images on the system as an array of strings (that is, not objects!). The first (`$Images[0]`) entry is a string of the line of column headers. The next two entries in the `$Images` array relate to the `ServerCore` and `hello-world` images, respectively.

In *step 8*, you create a simple PowerShell function that uses `docker.exe` to get and return the container images. This function returns the container image details as rich objects, not simple strings. If you are doing a lot of work with `docker.exe`, consider writing wrapper functions like this to simplify scripting.

In step 10, you download a container whose built-in base OS is a different version from the OS running on the container host (CH1). In step 11, you run it and see an error. You should expect this error. You have two options. The first is to use a more up-to-date container image that matches yours. The other alternative is to use Hyper-V isolation, which works fine, as you see in *step 12*.

In *step 13*, you could have used the `measure` command to measure how long it took Windows to load and execute the container.

Using Hyper-V isolation has a performance implication that involves the overhead of Hyper-V. But when you use Hyper-V for containers, you can experience a startup performance hit. But once the container is up and running, the hit is not usually significant. Also, using virtualization can provide added security, which may be appropriate, particularly in a shared-hosting environment.

Deploying IIS in a container

A popular containerized application is IIS. Microsoft publishes a container image that contains everything you need to run IIS containerized.

In this recipe, you download and run a container with IIS running and serving web pages.

Getting ready

This recipe uses the `CH1` host, which you configured in the *Configuring a container host* recipe.

How to do it...

1. Creating the `reskitapp` folder

```
$EA = @{ErrorAction='SilentlyContinue'}
New-Item -Path C:\ReskitApp -ItemType Directory @EA
```

2. Creating a web page

```
$FileName = 'C:\Reskitapp\Index.htm'
$Index = @"
<!DOCTYPE html>
<html><head><title>
ReskitApp Container Application</title></head>
<body><p><center><b>
HOME PAGE FOR RESKITAPP APPLICATION</b></p>
Running in a container in Windows Server 2022<p>
</center><br><hr></body></html>
"@
$Index | Out-File -FilePath $FileName
```

3. Getting a server core with an IIS image from the Docker registry

```
$Image = 'mcr.microsoft.com/windows/servercore/iis'
docker pull $Image
```

4. Running the image as a container named `rkwebc`

```
docker run -d -p80:80 --name rkwebc "$Image"
```

5. Copying the page into the container

```
Set-Location -Path C:\Reskitapp
docker cp .\index.htm rkwebc:c:\inetpub\wwwroot\index.htm
```

6. Viewing the page

```
Start-Process "Http://CH1.Reskit.Org/Index.htm"
```

7. Cleaning up

```
docker rm rkwebc -f | Out-Null
docker image rm  mcr.microsoft.com/windows/servercore/iis |
  Out-Null
```

How it works...

In *step 1*, you create a new folder to hold the new application you are going to create, with output like this:

```
PS C:\Foo> # 1.  Creating the reskitapp folder
PS C:\Foo> $EA = @{ErrorAction='SilentlyContinue'}
PS C:\Foo> New-Item -Path C:\ReskitApp -ItemType Directory @EA

    Directory: C:\

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          05/09/2022    16:13                ReskitApp
```

Figure 10.24: Creating the c:\ReskitApp folder

In *step 2*, you create a simple web page and store it in the `C:\ReskitApp` folder, producing no console output.

In *step 3*, you download a server core image that contains IIS, with output like this:

```
PS C:\Foo> # 3. Getting a server core with IIS image from the Docker registry:
PS C:\Foo> $Image = 'mcr.microsoft.com/windows/servercore/iis'
PS C:\Foo> docker pull $Image
Using default tag: latest
latest: Pulling from windows/servercore/iis
97f65a0ec59e: Pull complete
97b25a378238: Pull complete
7ebd66ebabd1: Pull complete
fa560e2e7835: Pull complete
39278cebafe6: Pull complete
Digest: sha256:d1821f5d785e5e17f4cb4194525dbcb57b7ec2e819d4db4738c14b6f2f2c2ad0
Status: Downloaded newer image for mcr.microsoft.com/windows/servercore/iis:latest
mcr.microsoft.com/windows/servercore/iis:latest
```

Figure 10.25: Pulling a container image with IIS

In *step 4*, you run the container image as a detached (i.e., permanently running) container. This step creates a small bit of console output as follows:

```
PS C:\Foo> # 4. Running the image as a container named rkwebc
PS C:\Foo> docker run -d -p80:80 --name rkwebc "$Image"
244189ade083393e734bf9aff4fb3339e4a6f340922ba812504327255fcaab20
```

Figure 10.26: Running a detached container

Now you have the container running (and running IIS), you can use your browser to connect to the website published by IIS. In *step 5*, you start the browser to view the default home page for the default website, with output like this:

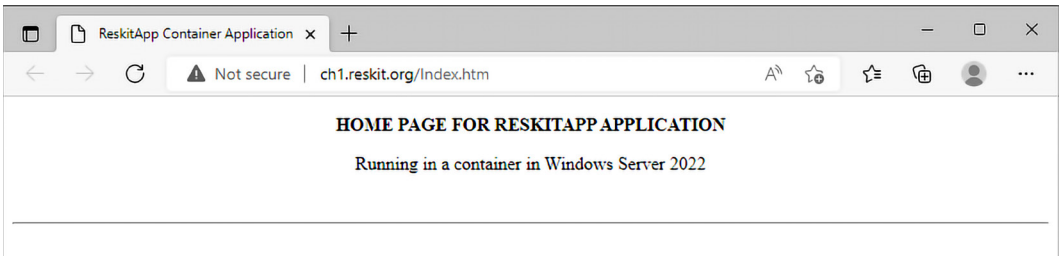

Figure 10.27: Running a detached container

In *step 7*, you clean up by first stopping the container and then removing the IIS container image. This step creates no output.

There's more...

This recipe creates a new web page (in *step 1* and *step 2*) on the `CH1` host, then copies that file into the running container (*step 5*). When you run the container, you use port forwarding to instruct Docker to forward port 80 on the container host to port 80 in the container.

With containers, although you do not have IIS loaded on `CH1`, you can run a container in which IIS is active and provides a website (although, in this case, a default website with no site content).

In this recipe, you use the existing network address/name (that is, of the `CH1` host) to access the container's website. You can see another method to push data into a container in the *Using a Dockerfile to create a custom container* recipe.

In *step 5*, you use the `docker cp` command to copy files from the container host into the container. In this recipe, you only add (and in *step 6*, view) a single page to the existing default website loaded by installing IIS. You can always use the `docker exec` command to create a new website inside the container and run that, much like you did in the recipes in the IIS chapter. You could also copy all the files and other resources necessary for a rich website, set up SSL, and use host headers to support multiple containers. There are other ways to transfer information between your container and other hosts in your environment. See `https://markheath.net/post/transfer-files-docker-windows-containers` and take a look at some methods you can use to transfer data into and out of your containers.

In this recipe, you forwarded traffic inbound to port 80 on the container host to port 80 in the container. This is a very simple way to use containers and container networking. You could also create a Docker network and give your container unique IP settings. For more on Docker networking, see the following: `http://rafalgolarz.com/blog/2017/04/10/networking_golang_app_with_docker_containers/` and `https://docs.docker.com/v17.09/engine/userguide/networking/`. You can, as ever, use your search engine to discover more about containers and networking. One thing to keep in mind as you search is that much of the search results relate to running containers on Linux, where the networking stack is quite different and differently managed.

Using a Dockerfile to create a custom container

You can use containers in various ways, depending on your needs. In most cases, you may find it useful to build custom container images, complete with an OS, OS features, and applications. A great way to make your image is to use a Dockerfile containing the instructions for building a new image and then use the `docker build` command to create a customized container you can then run.

In this recipe, you create a custom container image that provides an IIS website.

Getting ready

In this recipe, you use the container host, `CH1`, that you set up in the *Configuring a container host* recipe.

How to do it...

1. Creating folder and setting location to the folder on `CH1`

```
$SitePath = 'C:\RKWebContainer'
$NewItemHT = @{
  Path         = $SitePath
  ItemType     = 'Directory'
  ErrorAction  = 'SilentlyContinue'
}
New-Item @NewItemHT | Out-Null
Set-Location -Path $SitePath
```

2. Creating a script to run in the container to create a new site in the container

```
$ScriptBlock = {
# 2.1 Creating folder in the container
$SitePath = 'C:\RKWebContainer'
```

```
$NewItemHT2 = @{
  Path         = $SitePath
  ItemType     = 'Directory'
  ErrorAction  = 'SilentlyContinue'
}
New-Item @NewItemHT2 | Out-Null
Set-Location -Path $NewItemHT2.Path
# 2.1 Creating a page for the site
$PAGE = @'
<!DOCTYPE html>
<html>
<head><title>Main Page for RKWeb.Reskit.Org</title></head>
<body><p><center><b>
Home Page For RKWEB.RESKIT.ORG
</b></p>
Windows Server 2002, Containers, and PowerShell Rock!
</center/</body></html>
'@
$PAGE | OUT-FILE $SitePath\Index.html | Out-Null
#2.2 Creating a new web site in the container that uses Host headers
$WebSiteHT = @{
  PhysicalPath = $SitePath
  Name         = 'RKWeb'
  HostHeader   = 'RKWeb.Reskit.Org'
}
New-Website @WebSiteHT
} # End of script block
# 2.5 Save script block to file
$ScriptBlock | Out-File $SitePath\Config.ps1
```

3. Creating a new a record for our soon to be containerized site

```
Invoke-Command -Computer DC1.Reskit.Org -ScriptBlock {
  $DNSHT = @{
    ZoneName  = 'Reskit.Org'
    Name      = 'RKWeb'
    IpAddress = '10.10.10.221'
  }
```

```
  Add-DnsServerResourceRecordA @DNSHT
}
```

4. Creating Dockerfile

```
$DockerFile = @"
FROM mcr.microsoft.com/windows/servercore/iis
LABEL Description="RKWEB Container" Vendor="PS Partnership"
Version="1.0.0.42"
RUN powershell -Command Add-WindowsFeature Web-Server
RUN powershell -Command GIP
WORKDIR C:\\RKWebContainer
COPY Config.ps1 \Config.ps1
RUN powershell -command ".\Config.ps1"
"@
$DockerFile  | Out-File -FilePath .\Dockerfile -Encoding ascii
```

5. Building the image

```
docker build -t rkwebc .
```

6. Running the image

```
docker run -d --name rkwebc -p 80:80 rkwebc
```

7. Navigating to the container

```
Invoke-WebRequest -UseBasicParsing HTTP://RKweb.Reskit.Org
```

8. Viewing the web page in the browser

```
Start-Process "http://RKWeb.Reskit.Org"
```

9. Testing network connection to the local host port 80

```
Test-NetConnection -ComputerName localhost -Port 80
```

10. Cleaning up the container

```
docker container rm rkwebc -f
```

How it works...

In *step 1,* you create a new folder to hold a new IIS website and navigate to the folder. In *step 2*, you create a script you later run inside a container to create a new website (within the container). In *step 3*, you run a script block on `DC1` that creates a new `A` record for the containerized website. In *step 4*, you create a Dockerfile that contains the Docker build instructions that enable `docker build` to create your container image. These four setup steps generate no output.

In *step 5*, you use the Docker command to build your new customized container. The (voluminous) output from this step looks like this:

```
PS C:\RKWebContainer> # 5. Build the Images
PS C:\RKWebContainer> docker build -t rkwebc .
Sending build context to Docker daemon  3.584kB
Step 1/7 : FROM mcr.microsoft.com/windows/servercore/iis
latest: Pulling from windows/servercore/iis
97f65a0ec59e: Pull complete
97b25a378238: Pull complete
7ebd66ebabd1: Pull complete
fa560e2e7835: Pull complete
39278cebafe6: Pull complete
Digest: sha256:d1821f5d785e5e17f4cb4194525dbcb57b7ec2e819d4db4738c14b6f2f2c2ad0
Status: Downloaded newer image for mcr.microsoft.com/windows/servercore/iis:lates
 ---> 8397e926fa67
Step 2/7 : LABEL Description="RKWEB Container" Vendor="PS Partnership" Version="1
 ---> Running in 451b28258dc5
Removing intermediate container 451b28258dc5
 ---> b18bdcdfe1df
Step 3/7 : RUN powershell -Command Add-WindowsFeature Web-Server
 ---> Running in bc01c1831a2e

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    No             NoChangeNeeded {}

Removing intermediate container bc01c1831a2e
 ---> fbe8fcd9327a
Step 4/7 : RUN powershell -Command GIP
 ---> Running in ae9fa7a4d55c


InterfaceAlias       : vEthernet (Ethernet)
InterfaceIndex       : 21
InterfaceDescription : Hyper-V Virtual Ethernet Container Adapter
IPv4Address          : 172.26.216.230
IPv6DefaultGateway   :
IPv4DefaultGateway   : 172.26.208.1
DNSServer            : 172.26.208.1
                       10.10.10.10


Removing intermediate container ae9fa7a4d55c
 ---> f87f6d9a3836
Step 5/7 : WORKDIR C:\\RKWebContainer
 ---> Running in baee270860dd
Removing intermediate container baee270860dd
 ---> 15c07d026d4a
Step 6/7 : COPY Config.ps1 \Config.ps1
 ---> a2c8e08456d6
Step 7/7 : RUN powershell -command ".\Config.ps1"
 ---> Running in 7382f69c7cda

Name             ID   State      Physical Path                Bindings
----             --   -----      -------------                --------
RKWeb            1299 Started    C:\RKWebContainer            http *:80:RKWeb
                 8361                                         .Reskit.Org
                 53


Removing intermediate container 7382f69c7cda
 ---> 8e710ffdf906
Successfully built 8e710ffdf906
Successfully tagged rkwebc:latest
```

Figure 10.28: Running a detached container

In *step 6*, you run a detached container with your new customized container image, producing output like this:

```
PS C:\RKWebContainer> # 6. Running the image
PS C:\RKWebContainer> docker run -d --name rkwebc -p 80:80 rkwebc
e48e53b8bd46e7ca86991ca7c161b57037df3e2dc2989f44946a8d03942865ba
```

Figure 10.29: Running a detached container

In *step 7*, you use the `Invoke-WebRequest` command to get the web page from the container. This step produces output like this:

```
PS C:\RKWebContainer> # 7. Navigating to the container
PS C:\RKWebContainer> Invoke-WebRequest -UseBasicParsing HTTP://RKweb.Reskit.Org

StatusCode        : 200
StatusDescription : OK
Content           : yp<!DOCTYPE html>
                    <html>
                    <head><title>Main Page for RKWeb.Reskit.Org</title></head>
                    <body><p><cen…
RawContent        : HTTP/1.1 200 OK
                    Accept-Ranges: bytes
                    ETag: "bae994ef43c1d81:0"
                    Server: Microsoft-IIS/10.0
                    Date: Mon, 05 Sep 2022 16:44:23 GMT
InputFields       : {}
Links             : {}
RawContentLength  : 416
RelationLink      : {}
```

Figure 10.30: Getting the web page from the container

In *step 8*, you view the web page from the containerized website within the browser, with output like this:

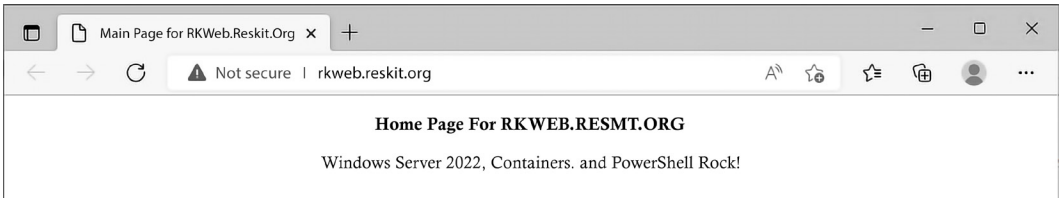

Figure 10.31: Getting the web page from the container

In *step 9*, you view the network connection to port `80` on the local host. Docker binds this local port to port 80 in the container. The output from this step looks like this:

```
PS C:\RKWebContainer> # 9. Testing network connection
PS C:\RKWebContainer> Test-NetConnection -ComputerName localhost -Port 80

ComputerName     : localhost
RemoteAddress    : 127.0.0.1
RemotePort       : 80
InterfaceAlias   : Loopback Pseudo-Interface 1
SourceAddress    : 127.0.0.1
TcpTestSucceeded : True
```

Figure 10.32: Viewing port 80 on the local host

In the final step in this recipe, *step 10*, you stop and close the container producing no output.

There's more...

In this recipe, you use a base container image that you have to download from the Docker registry (`mcr.microsoft.com/windows/servercore/iis`). Then, you build a container that has the web server feature added and in which you can run the `Config.ps1` file to configure the container to run your website. For more information on Dockerfiles, see this link: `https://docs.docker.com/engine/reference/builder/`

Join our community on Discord

Join our community's Discord space for discussions with the author and other readers:

`https://packt.link/SecNet`