IN THIS CHAPTER

- Understanding containers in Windows Server 2022
- Examining common use cases for containers
- Deciding what type of containers you want to use
- Implementing containers at scale

Chapter 1

Introduction to Containers in Windows Server 2022

Virtualization drastically changed the way that IT operated in organizations of all sizes, but containers have had a large impact as well. You may be wondering why someone would want to use containers. They're just virtual machines (VMs), right? Well, not exactly. The technologies may seem similar, but containers and VMs are not the same. VMs are presenting virtual hardware to the user. Containers don't expose the hardware or the operating system; they're meant to run applications in isolation.

In this chapter, I fill you in on containers — what they are, why you would use them, and how the Windows Server implementation will work for you.

Understanding Containers

VMs can be thought of as Infrastructure as a Service (IaaS). Although VMs do present virtual hardware to system administrators, the administrators of virtual servers don't have to be concerned about the underlying hardware. They can focus on the operating system and applications that they're responsible for.

Containers take this idea and refine it to where each container is responsible for running an application. The application is baked into the image so the containers can be stood up and torn down constantly. This is great for Platform as a Service (PaaS) scenarios where developers just want to test their code and not worry about getting servers provisioned to test against. Developers don't generally care about hardware or operating systems; they just want to know that their code works in the way they expect it to.

The main idea behind containers is that the application inside of each container has all the resources that it requires to function within the same container. This means that you can drop the container on any container host, and all the application's requirements will still be met because those requirements (.NET, for example) move with the application inside the container.

Knowing what a container looks like

You may be wondering what containers look like. Let me use the example of containers in Windows specifically. At a high level, the architecture looks something like Figure 1-1.

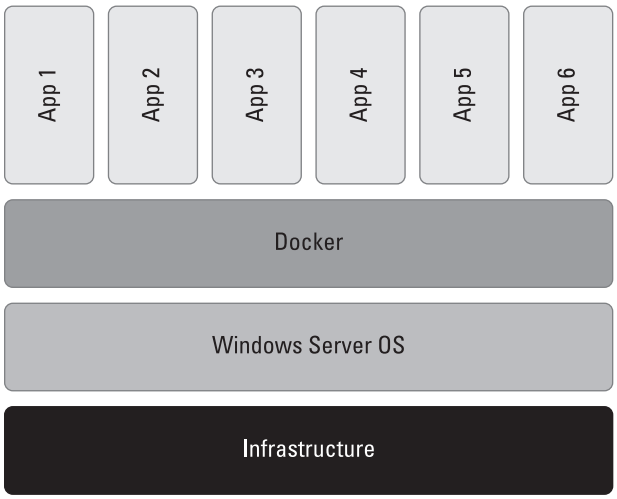

FIGURE 1-1: Container architecture on Windows Server involves several layers and utilizes the Docker Engine to work with containers.

In a Windows Server operating system, after you enable the containers feature, you install the Docker Engine. The Docker Engine is responsible for packaging and deploying the containers. Microsoft partnered with Docker for the first time with Windows Server 2016 to support running containers on a Windows operating system.

Defining important container terms

As with most newer technologies, there are new terms that you need to understand to be on the same page as other system administrators who work with containers. Here are the most important terms:

- **Container host:** The container host is the system that is configured with the Windows Container feature. It can be a physical host or, through the joys of nested virtualization, a virtual host. All the containers on the container host share the host's resources.
- **Container image:** When you create a container image, you create a deployable image that contains the changes you made to the original image, which were stored in the sandbox. The container image does not contain the operating system (OS); instead, when you deploy custom container images, they're a layer of customization that is added on top of the container OS image.
- **Sandbox:** The sandbox saves changes as they're made to the container image. This can include modifications to the file system and Registry, and any new applications you might install. Changes saved in the sandbox can be saved as container images so they can be reused.
- **Container OS image:** Not to be confused with the container image, the container OS image can't be modified. It is the first layer in the container sandwich and provides the operating system that the container will use.
- **Container repository:** Container images along with any dependencies they may have are stored in a container repository so that they can be reused. They can be stored in a local repository, or if you plan on using the image across multiple container hosts, you can create private or public repositories on Docker Hub. Repositories may also be referred to as registries; Docker Hub, for instance, is often referred to as a container registry.

Seeing how containers run on Windows

You may have noticed that I've referenced Docker multiple times. That's because containers use the Docker Engine to run on Windows Server! Containers were first

introduced in Windows Server 2016, but the technology and, of course, Docker itself have been around a lot longer than that.

Docker is the engine that is responsible for packaging and delivering container images. Those container images can be based on Windows or Linux operating systems and can run in your datacenter an Windows Server 2022.

You can find more information on Docker in Chapter 2 of this minibook.

Considering Use Cases for Containers

There are several different use cases for containers. The use cases generally depend on what interests you most. System administrators, for example, will have very different use cases from developers. Thankfully, containers can support the majority of the use cases.

Developers

Because containers have everything your application depends on to run, you can move those containers anywhere (so long as it's running Windows Server 2016 or newer). This means that you can start your development work from just about anywhere and finish by moving your container image into production. Plus, Docker Hub has more than 180,000 applications already packaged and available, which can greatly speed up your development work!

System administrators

I don't know many system administrators who enjoy building environments over and over again — pulling out the dreaded checklist to ensure that you don't forget to install this one thing or configure that one feature.

Rejoice, system administrators! With containers, you have an instant packaged environment for all your teams from the developers to your production support teams. No more dreaded checklist! When you need to make a change or update the containers, you simply update the container image that the containers are built from, and — *voilà!* — the next time they build their container from the container image, they're all up to date. Something breaks because of an update or a change? Roll them back to the previous container image.

Deciding What Type of Containers You Want to Use

Windows Server allows you to use two different methodologies to manage your containers:

- **Windows Server containers:** Windows Server containers are simple and are good for workloads where security and isolation are not as important as simply getting them up and going.
- **Hyper-V containers:** Hyper-V containers provide a higher degree of security and isolation and are the perfect choice for when you need to more strongly enforce these on your containers.

You aren't stuck with your decision. You can create a container as a Hyper-V container, for instance, and change it to a Windows container later on.

I cover the installation of containers in Chapter 3 of this minibook.

Windows Server containers

Windows Server containers provide a simple method of application isolation that leverages process and namespace isolation. Because this type of container shares the kernel of the container host that it's on, and by extension all the other containers on that host, you don't get true isolation. If you write code that you think could be destructive, this is not the ideal environment for that. Windows Server containers should be used for trusted or non-impactful code that does not require a strong security component.

REMEMBER

Because the kernel is shared, it's important to remember that the host and containers must all be on the same kernel version. This may cause issues for developers — you do lose some flexibility for testing because of this requirement.

Hyper-V containers

Hyper-V containers provide a true degree of isolation. Each of the containers is run in its own VM and does not share the kernel with either the container host or the other containers on that same host. These containers are perfect for workloads that require a higher degree of security, or when running untrusted code, because damage is contained in the individual container. If the untrusted code causes damage, you can simply spin up another Hyper-V container and try again after the issue is fixed.

Because the kernel isn't shared between the container host and the containers that reside on it, you can have different kernel versions throughout all the containers.

Managing Containers at Scale

Even if you're convinced of the benefit of containers, you may be wondering how difficult it will be if containers catch on at your workplace. How are you going to manage all those containers?!

Dockerfile allows you to automate the creation of your container images. Think of Dockerfile like a script for your containers; Dockerfile specifies which features to install and configures the container.

Microsoft maintains a GitHub repository with tons of examples of Dockerfiles to get you started. You can check it out at `https://github.com/MicrosoftDocs/Virtualization-Documentation/tree/master/windows-container-samples`. What I love most about this repository is that there are so many use cases represented, and not all are with Microsoft products. There are Dockerfiles for Apache, MongoDB, MySQL, and nginx, as shown in Figure 1-2.

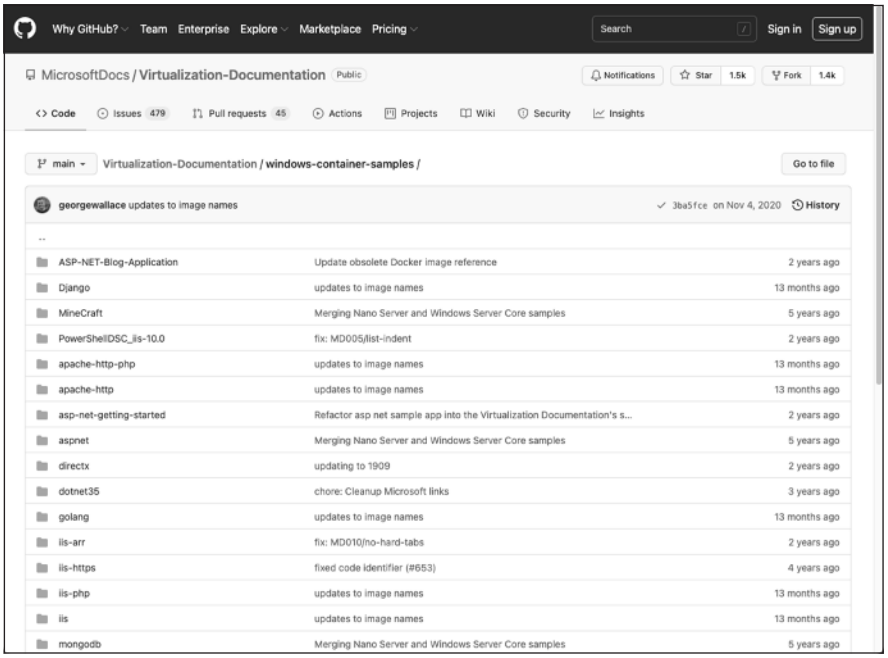

FIGURE 1-2: The GitHub repository where Microsoft stores examples of Dockerfiles you can use with Windows Server container implementations.