4.1 Virtual Machines and Containers

Public cloud data centers comprise many thousands of individual servers. Some servers are used exclusively for data services and supporting infrastructure and others for hosting your computations. When you compute in the cloud, you do not run directly on one of these servers in the way that you would in a conventional computational cluster. Instead, you are provided with a virtual machine running your favorite operating system. A **virtual machine** is just the software image of a complete machine that can be loaded onto the server and run like any other program. The server in the data center runs a piece of software called a **hypervisor** that allocates and manages the server's resources that are granted to its "guest" virtual machines. In the next chapter, we delve into how virtualization works, but the key idea is that when you run in a VM, it looks exactly like a server running whatever operating system the VM is configured to run.

For the cloud operator, virtualization has huge advantages. First, the cloud provider can provide dozens of different operating systems packaged as VMs for the user to choose from. To the hypervisor, all VMs look the same and can be managed in a uniform way. The cloud management system (sometimes called the **fabric controller**) can select which server to use to run the requested VM instances, and it can monitor the health of each VM. If needed, the cloud monitor can run many VMs simultaneously on a single server. If a VM instance crashes, it does not crash the server. The cloud monitor can record the event and restart the VM. User applications running in different VMs on the same server are largely unaware of each other. (A user may notice another VM when they impact the performance or response of their VM.)

We provide in chapter 5 detailed instructions on how to deploy VMs on the Amazon and Azure public clouds, and on OpenStack private clouds.

Containers are similar to VMs but are based on a different technology and serve a slightly different purpose. Rather than run a full OS, a container is layered on top of the host OS and uses that OS's resources in a clever way. Containers allow you to package up an application and all of its library dependencies and data into a single, easy-to-manage unit. When you launch the container, the application can be configured to start up, go through its initialization, and be running in seconds. For example, you can run a web server in one container and a database server in another; these two containers can discover each other and communicate as needed. Or, if you have a special simulation program in a container, you can start multiple instances of the container on the same host.

Containers have the advantage of being extremely lightweight. Once you have downloaded a container to a host, you can start it and the application(s) that it contains quasi-instantly. Part of the reason for this speed is that a container instance can share libraries with other container instances. VMs, because they are complete OS instances, can take a few minutes to start up. You can run many more containers on a single host machine than you can effectively run the same number of VMs. Figure 4.1 illustrates the difference between the software stack of a server running multiple VMs versus a server running a single OS and multiple containers on a typical server in a cloud.

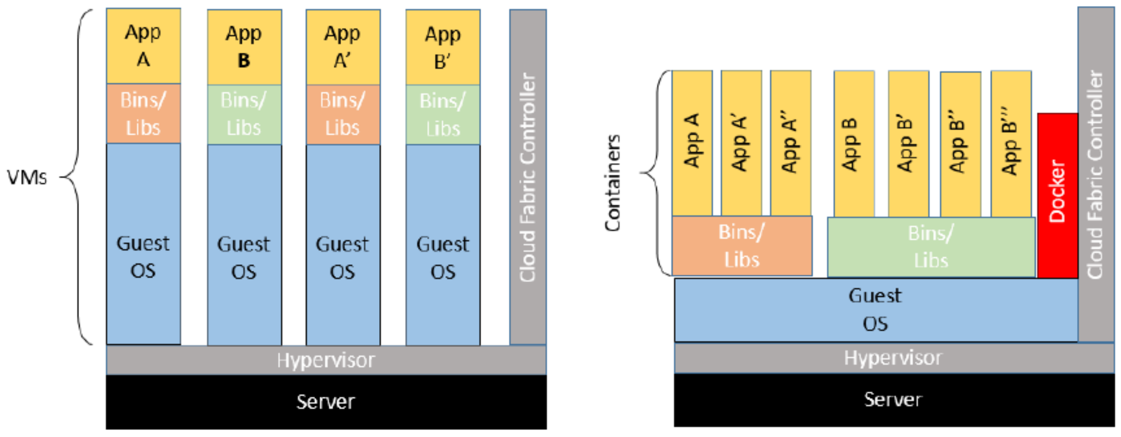

Figure 4.1: Virtual machines vs. containers on a typical cloud server.

Building a container to run a single application is simple compared with the task of customizing a VM to run a single application. All you need to do is create a script that identifies the needed libraries, source files, and data. You can then run the script on your laptop to test the container, before uploading the container to a repository, from where it can be downloaded to any cloud. Importantly, containers are completely portable across different clouds. In general, VM images cannot be ported from one cloud framework to another.

Containers also have downsides. The most serious issue is security. Because containers share the same host OS instance, two containers running on the same

Table 4.1: Virtual machines and containers, compared.

Virtual machines	**Containers**
Heavyweight	Lightweight
Fully isolated; hence more secure	Process-level isolation; hence less secure
No automation for configuration	Script-driven configuration
Slow deployment	Rapid deployment
Easy port and IP address mapping	More abstract port and IP mappings
Custom images not portable across clouds	Completely portable

host are less isolated than two VMs running on that host. Managing the network ports and IP addresses used by containers can be slightly more confusing than when working with VMs. Furthermore, containers are often run on top of VMs, which can exacerbate the confusion.

In chapter 6, we work though examples of using containers in some detail, describing in particular the Docker container system `docker.com`, how to run containers, and how to create your own containers.

Table 4.1 lists some of the the pros and cons of the virtual machine and container approaches. Needless to say, both technologies are evolving rapidly.