Terraform), so you either end up writing lots of complicated deployment scripts or you turn to orchestration tools, as described next.

Provisioning plus server templating plus orchestration

Example: Terraform, Packer, Docker, and Kubernetes. You use Packer to create a VM image that has Docker and Kubernetes agents installed. You then use Terraform to deploy a cluster of servers, each of which runs this VM image, and the rest of your infrastructure, including the network topology (i.e., VPCs, subnets, route tables), data stores (e.g., MySQL, Redis), and load balancers. Finally, when the cluster of servers boots up, it forms a Kubernetes cluster that you use to run and manage your Dockerized applications, as shown in Figure 1-11.

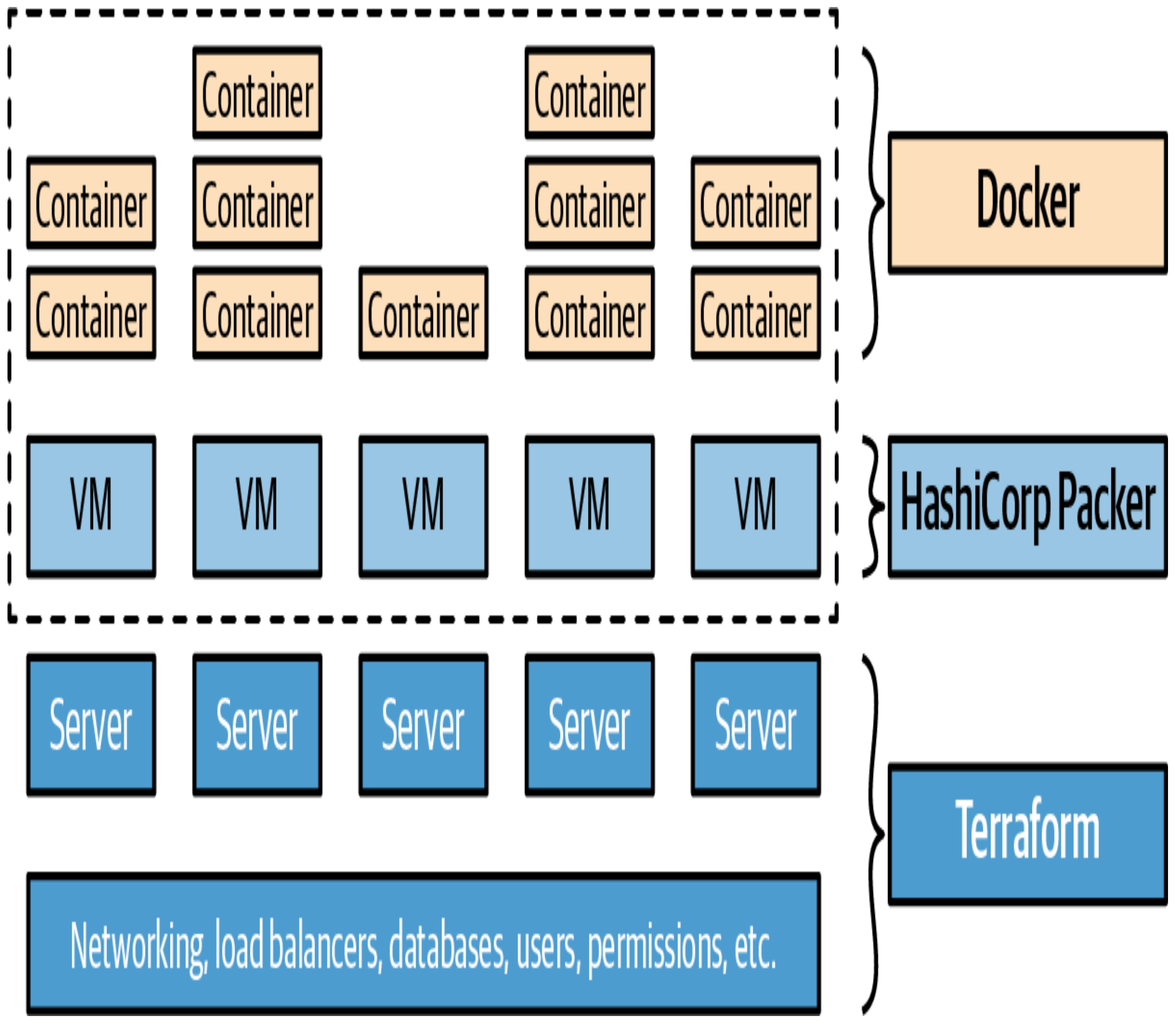

Figure 1-11. Terraform deploys the infrastructure, including servers; Packer creates the VMs that run on those servers; and Kubernetes manages those VMs as a cluster for running Docker containers.

The advantage of this approach is that Docker images build fairly quickly, you can run and test them on your local computer, and you can take advantage of all of the built-in functionality of Kubernetes, including various deployment strategies, auto healing, auto scaling, and so on. The drawback is the added complexity, both in terms of extra infrastructure to run (Kubernetes clusters are difficult and expensive to deploy and operate, though most major cloud providers now provide managed Kubernetes services, which can offload some of this work) and in terms of several extra layers of abstraction (Kubernetes, Docker, Packer) to learn, manage, and debug.