few examples:

Terraform

As you've seen in this book, you can use Terraform to deploy certain types of applications. For example, in earlier chapters, you created a module called `asg-rolling-deploy` that could do a zero-downtime rolling deployment across an ASG. If you package your application as an AMI (e.g., using Packer), you could deploy new AMI versions with the `asg-rolling-deploy` module by updating the `ami` parameter in your Terraform code and running `terraform apply`.

Orchestration tools

There are a number of orchestration tools designed to deploy and manage applications, such as Kubernetes (arguably the most popular Docker orchestration tool), Amazon ECS, HashiCorp Nomad, and Apache Mesos. In Chapter 7, you saw an example of how to use Kubernetes to deploy Docker containers.

Scripts

Terraform and most orchestration tools support only a limited set of deployment strategies (discussed in the next section). If you have more complicated requirements, you may have to write custom scripts to implement these requirements.

Deployment strategies

There are a number of different strategies that you can use for application deployment, depending on your requirements. Suppose that you have five copies of the old version of your app running, and you want to roll out a new version. Here are a few of the most common strategies you can use:

Rolling deployment with replacement

Take down one of the old copies of the app, deploy a new copy to replace it, wait for the new copy to come up and pass health checks, start sending the new copy live traffic, and then repeat the process until all of the old copies have been replaced. Rolling deployment with replacement ensures that you never have more than five copies of the app running, which can be useful if you have limited capacity (e.g., if each copy of the app runs on a physical server) or if you're dealing with a stateful system where each app has a unique identity (e.g., this is often the case with consensus systems, such as Apache ZooKeeper). Note that this deployment strategy can work with larger batch sizes (you can replace more than one copy of the app at a time if you can handle the load and won't lose data with fewer apps running) and that during deployment, you will have both the old and new versions of the app running at the same time.

Rolling deployment without replacement

Deploy one new copy of the app, wait for the new copy to come up and pass health checks, start sending the new copy live traffic, undeploy an old copy of the app, and then repeat the process until all the old copies have been replaced. Rolling deployment without replacement works only if you have flexible capacity (e.g., your apps run in the cloud, where you can spin up new virtual servers any time you want) and if your application can tolerate more than five copies of it running at the same time. The advantage is that you never have less than five copies of the app running, so you're not running at a reduced capacity during deployment. Note that this deployment strategy can also work with larger batch sizes (if you have the capacity for it, you can deploy five new copies all at once) and that during deployment, you will have both the old and new versions of the app running at the same time.

Blue-green deployment

Deploy five new copies of the app, wait for all of them to come up and pass health checks, shift all live traffic to the new copies at the same time, and then undeploy the old copies. Blue-green deployment works

only if you have flexible capacity (e.g., your apps run in the cloud, where you can spin up new virtual servers any time you want) and if your application can tolerate more than five copies of it running at the same time. The advantage is that only one version of your app is visible to users at any given time and that you never have less than five copies of the app running, so you're not running at a reduced capacity during deployment.

Canary deployment

Deploy one new copy of the app, wait for it to come up and pass health checks, start sending live traffic to it, and then pause the deployment. During the pause, compare the new copy of the app, called the "canary," to one of the old copies, called the "control." You can compare the canary and control across a variety of dimensions: CPU usage, memory usage, latency, throughput, error rates in the logs, HTTP response codes, and so on. Ideally, there's no way to tell the two servers apart, which should give you confidence that the new code works just fine. In that case, you unpause the deployment and use one of the rolling deployment strategies to complete it. On the other hand, if you spot any differences, then that may be a sign of problems in the new code, and you can cancel the deployment and undeploy the canary before the problem becomes worse.

The name comes from the "canary in a coal mine" concept, where miners would take canary birds with them down into the tunnels, and if the tunnels filled with dangerous gases (e.g., carbon monoxide), those gases would affect the canary before the miners, thus providing an early warning to the miners that something was wrong and that they needed to exit immediately, before more damage was done. The canary deployment offers similar benefits, giving you a systematic way to test new code in production in a way that, if something goes wrong, you get a warning early on, when it has affected only a small portion of your users and you still have enough time to react and prevent further damage.