Chapter 3. Deploying a Kubernetes Cluster

Now that you have successfully built an application container, you are motivated to learn how to deploy it into a complete reliable, scalable distributed system. Of course, to do that, you need a working Kubernetes cluster. At this point, there are several cloud-based Kubernetes services that make it easy to create a cluster with a few command-line instructions. We highly recommend this approach if you are just getting started with Kubernetes. Even if you are ultimately planning on running Kubernetes on bare metal, it makes sense to quickly get started with Kubernetes, learn about Kubernetes itself, and then learn how to install it on physical machines.

Of course, using a cloud-based solution requires paying for those cloud-based resources as well as having an active network connection to the cloud. For these reasons, local development can be more attractive, and in that case the `minikube` tool provides an easy-to-use way to get a local Kubernetes cluster up running in a VM on your local laptop or desktop. Though this is attractive, `minikube` only creates a single-node cluster, which doesn't quite demonstrate all of the aspects of a complete Kubernetes cluster. For that reason, we recommend people start with a cloud-based solution, unless it really doesn't work for their situation. If you truly insist on starting on bare metal, Appendix A at the end of this book gives instructions for building a cluster from a collection of Raspberry Pi single-board computers. These instructions use the `kubeadm` tool and can be adapted to other machines beyond Raspberry Pis.

Installing Kubernetes on a Public Cloud Provider

This chapter covers installing Kubernetes on the three major cloud providers, Amazon Web Services (AWS), Microsoft Azure, and the Google Cloud Platform.

Google Container Service

The Google Cloud Platform offers a hosted Kubernetes-as-a-Service called Google Container Engine (GKE). To get started with GKE, you need a Google Cloud Platform account with billing enabled and the `gcloud` tool installed.

Once you have `gcloud` installed, first set a default zone:

```
$ gcloud config set compute/zone us-west1-a
```

Then you can create a cluster:

```
$ gcloud container clusters create kuar-cluster
```

This will take a few minutes. When the cluster is ready you can get credentials for the cluster using:

```
$ gcloud auth application-default login
```

At this point, you should have a cluster configured and ready to go. Unless you would prefer to install Kubernetes elsewhere, you can skip to "The Kubernetes Client".

If you run into trouble, the complete instructions for creating a GKE cluster can be found in the Google Cloud Platform documentation.

Installing Kubernetes with Azure Container Service

Microsoft Azure offers a hosted Kubernetes-as-a-Service as part of the Azure Container Service. The easiest way to get started with Azure Container Service is to use the built-in Azure Cloud Shell in the Azure portal. You can activate the shell by clicking the shell icon:

in the upper-right toolbar. The shell has the `az` tool automatically installed and configured to work with your Azure environment.

Alternatively, you can install the `az` command-line interface (CLI) on your local machine.

Once you have the shell up and working, you can run:

```
$ az group create --name=kuar --location=westus
```

Once the resource group is created, you can create a cluster using:

```
$ az acs create --orchestrator-type=kubernetes \
  --resource-group=kuar --name=kuar-cluster
```

This will take a few minutes. Once the cluster is created, you can get credentials for the cluster with:

```
$ az acs kubernetes get-credentials --resource-group=kuar --name=kuar-cluster
```

If you don't already have the `kubectl` tool installed, you can install it using:

```
$ az acs kubernetes install-cli
```

Complete instructions for installing Kubernetes on Azure can be found in the Azure documentation.

Installing Kubernetes on Amazon Web Services

AWS does not currently offer hosted Kubernetes service. The landscape for managing Kubernetes on AWS is a fast-evolving area with new and improved tools being introduced often. Here are a couple of options that make it easy to get started:

- The easiest way to launch a small cluster appropriate for exploring Kubernetes with this book is using the Quick Start for Kubernetes by Heptio. This is a simple CloudFormation template that can launch a cluster using the AWS Console.

- For a more fully featured management solution, consider using a project called `kops`. You can find a complete tutorial for installing Kubernetes on AWS using `kops` on GitHub.

Installing Kubernetes Locally Using minikube

If you need a local development experience, or you don't want to pay for cloud resources, you can install a simple single-node cluster using `minikube`. While `minikube` is a good simulation of a Kubernetes cluster, it is really intended for local development, learning, and experimentation. Because it only runs in a VM on a single node, it doesn't provide the reliability of a distributed Kubernetes cluster.

In addition, certain features described in this book require integration with a cloud provider. These features are either not available or work in a limited way with `minikube`.

NOTE

You need to have a hypervisor installed on your machine to use `minikube`. For Linux and macOS, this is generally `virtualbox`. On Windows, the `Hyper-V` hypervisor is the default option. Make sure you install the hypervisor before using `minikube`.

You can find the `minikube` tool on GitHub. There are binaries for Linux, macOS, and Windows that you can download. Once you have the `minikube` tool installed you can create a local cluster using:

```
$ minikube start
```

This will create a local VM, provision Kubernetes, and create a local `kubectl` configuration that points to that cluster.

When you are done with your cluster, you can stop the VM with:

```
$ minikube stop
```

If you want to remove the cluster, you can run:

```
$ minikube delete
```

Running Kubernetes on Raspberry Pi

If you want to experiment with a realistic Kubernetes cluster but don’t want to pay a lot, a very nice Kubernetes cluster can be built on top of Raspberry Pi computers for a relatively small cost. The details of building such a cluster are out of scope for this chapter, but they are given in Appendix A at the end of this book.

The Kubernetes Client

The official Kubernetes client is `kubectl`: a command-line tool for interacting with the Kubernetes API. `kubectl` can be used to manage most Kubernetes objects such as pods, ReplicaSets, and services. `kubectl` can also be used to explore and verify the overall health of the cluster.

We’ll use the `kubectl` tool to explore the cluster you just created.

Checking Cluster Status

The first thing you can do is check the version of the cluster that you are running:

```
$ kubectl version
```

This will display two different versions: the version of the local `kubectl` tool, as well as the version of the Kubernetes API server.

> **NOTE**
>
> Don’t worry if these versions are different. The Kubernetes tools are backward- and forward-compatible with different versions of the Kubernetes API, so long as you stay within two minor versions of the tools and the cluster and don’t try to use newer features on an older cluster. Kubernetes follows the semantic versioning specification, and this minor version is the middle number (e.g., the 5 in 1.5.2).

Now that we’ve established that you can communicate with your Kubernetes cluster, we’ll explore the cluster in more depth.

First, we can get a simple diagnostic for the cluster. This is a good way to verify that your cluster is generally healthy:

```
$ kubectl get componentstatuses
```

The output should look like this:

```
NAME                 STATUS    MESSAGE              ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-0               Healthy   {"health": "true"}
```

You can see here the components that make up the Kubernetes cluster. The `controller-manager` is responsible for running various controllers that regulate behavior in the cluster: for example, ensuring that all of the replicas of a service are available and healthy. The `scheduler` is responsible for placing different pods onto different nodes in the cluster. Finally, the `etcd` server is the storage for the cluster where all of the API objects are stored.

Listing Kubernetes Worker Nodes

Next, we can list out all of the nodes in our cluster:

```
$ kubectl get nodes
NAME         STATUS         AGE
kubernetes   Ready,master   45d
node-1       Ready          45d
node-2       Ready          45d
node-3       Ready          45d
```

You can see this is a four-node cluster that's been up for 45 days. In Kubernetes nodes are separated into `master` nodes that contain containers like the API server, scheduler, etc., which manage the cluster, and `worker` nodes where your containers will run. Kubernetes won't generally schedule work onto `master` nodes to ensure that user workloads don't harm the overall operation of the cluster.

You can use the `kubectl describe` command to get more information about a specific node such as `node-1`:

```
$ kubectl describe nodes node-1
```

First, you see basic information about the node:

```
Name:                   node-1
Role:
Labels:                 beta.kubernetes.io/arch=arm
                        beta.kubernetes.io/os=linux
                        kubernetes.io/hostname=node-1
```

You can see that this node is running the Linux OS and is running on an ARM processor.

Next, you see information about the operation of `node-1` itself:

```
Conditions:
  Type           Status LastHeartbeatTime  Reason                     Message
  ----           ------ -----------------  ------                     -------
  OutOfDisk      False  Sun, 05 Feb 2017…  KubeletHasSufficientDisk   kubelet…
  MemoryPressure False  Sun, 05 Feb 2017…  KubeletHasSufficientMemory kubelet…
  DiskPressure   False  Sun, 05 Feb 2017…  KubeletHasNoDiskPressure   kubelet…
  Ready          True   Sun, 05 Feb 2017…  KubeletReady               kubelet…
```

These statuses show that the node has sufficient disk and memory space, and it is reporting that it is healthy to the Kubernetes master. Next, there is information about the capacity of the machine:

```
Capacity:
 alpha.kubernetes.io/nvidia-gpu:       0
 cpu:                                  4
 memory:                               882636Ki
 pods:                                 110
Allocatable:
 alpha.kubernetes.io/nvidia-gpu:       0
 cpu:                                  4
 memory:                               882636Ki
 pods:                                 110
```

Then, there is information about the software on the node, including the version of Docker running, the versions of Kubernetes and the Linux kernel, and more:

```
System Info:
 Machine ID:                   9989a26f06984d6dbadc01770f018e3b
 System UUID:                  9989a26f06984d6dbadc01770f018e3b
 Boot ID:                      98339c67-7924-446c-92aa-c1bfe5d213e6
 Kernel Version:               4.4.39-hypriotos-v7+
 OS Image:                     Raspbian GNU/Linux 8 (jessie)
 Operating System:             linux
 Architecture:                 arm
 Container Runtime Version:    docker://1.12.6
 Kubelet Version:              v1.5.2
 Kube-Proxy Version:           v1.5.2
PodCIDR:                       10.244.2.0/24
ExternalID:                    node-1
```

Finally, there is information about the pods that are currently running on this node:

```
Non-terminated Pods:           (3 in total)
  Namespace   Name      CPU Requests CPU Limits Memory Requests Memory Limits
  ---------   ----      ------------ ---------- --------------- -------------
  kube-system kube-dns…  260m (6%)    0 (0%)     140Mi (16%)     220Mi (25%)
  kube-system kube-fla…  0 (0%)       0 (0%)     0 (0%)          0 (0%)
  kube-system kube-pro…  0 (0%)       0 (0%)     0 (0%)          0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.
  CPU Requests  CPU Limits      Memory Requests Memory Limits
  ------------  ----------      --------------- -------------
  260m (6%)     0 (0%)          140Mi (16%)     220Mi (25%)
No events.
```

From this output you can see the pods on the node (e.g., the `kube-dns` pod that supplies DNS services for the cluster), the CPU and memory that each pod is requesting from the node, as well as the total resources requested. It's worth noting here that Kubernetes tracks both the *request* and upper *limit* for resources for each pod that runs on a machine. The difference between requests and limits is described in detail in Chapter 5, but in a nutshell, resources *requested* by a pod are guaranteed to be present on the node, while a pod's limit is the maximum amount of a given resource that a pod can consume. A pod's limit can be higher than its request, in which case the extra resources are supplied on a best-effort basis. They are not guaranteed to be present on the node.

Cluster Components

One of the interesting aspects of Kubernetes is that many of the components that make up the Kubernetes cluster are actually deployed using Kubernetes itself. We’ll take a look at a few of these. These components use a number of the concepts that we’ll introduce in later chapters. All of these components run in the `kube-system` namespace.[1]

Kubernetes Proxy

The Kubernetes proxy is responsible for routing network traffic to load-balanced services in the Kubernetes cluster. To do its job, the proxy must be present on every node in the cluster. Kubernetes has an API object named `DaemonSet`, which you will learn about later in the book, that is used in many clusters to accomplish this. If your cluster runs the Kubernetes proxy with a `DaemonSet`, you can see the proxies by running:

```
$ kubectl get daemonSets --namespace=kube-system kube-proxy
NAME         DESIRED   CURRENT   READY     NODE-SELECTOR   AGE
kube-proxy   4         4         4         <none>          45d
```

Kubernetes DNS

Kubernetes also runs a DNS server, which provides naming and discovery for the services that are defined in the cluster. This DNS server also runs as a replicated service on the cluster. Depending on the size of your cluster, you may see one or more DNS servers running in your cluster. The DNS service is run as a Kubernetes deployment, which manages these replicas:

```
$ kubectl get deployments --namespace=kube-system kube-dns
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
kube-dns   1         1         1            1           45d
```

There is also a Kubernetes service that performs load-balancing for the DNS server:

```
$ kubectl get services --namespace=kube-system kube-dns
NAME       CLUSTER-IP   EXTERNAL-IP   PORT(S)         AGE
kube-dns   10.96.0.10   <none>        53/UDP,53/TCP   45d
```

This shows that the DNS service for the cluster has the address 10.96.0.10. If you log into a container in the cluster, you’ll see that this has been populated into the */etc/resolv.conf* file for the container.

Kubernetes UI

The final Kubernetes component is a GUI. The UI is run as a single replica, but it is still managed by a Kubernetes deployment for reliability and upgrades. You can see this UI server using:

```
$ kubectl get deployments --namespace=kube-system kubernetes-dashboard
NAME                   DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
kubernetes-dashboard   1         1         1            1           45d
```

The dashboard also has a service that performs load balancing for the dashboard:

```
$ kubectl get services --namespace=kube-system kubernetes-dashboard
NAME                   CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes-dashboard   10.99.104.174   <nodes>       80:32551/TCP   45d
```

We can use the `kubectl` proxy to access this UI. Launch the Kubernetes proxy using:

```
$ kubectl proxy
```

This starts up a server running on *localhost:8001*. If you visit *http://localhost:8001/ui* in your web browser, you should see the Kubernetes web UI. You can use this interface to explore your cluster, as well as create new containers. The full details of this interface are outside of the scope of this book, and it is changing rapidly as the dashboard is improved.

Summary

Hopefully at this point you have a Kubernetes cluster (or three) up and running and you've used a few commands to explore the cluster you have created. Next, we'll spend some more time exploring the command-line interface to that Kubernetes cluster and teach you how to master the `kubectl` tool. Throughout the rest of the book, you'll be using `kubectl` and your test cluster to explore the various objects in the Kubernetes API.

---

1 As you'll learn in the next chapter, a namespace in Kubernetes is an entity for organizing Kubernetes resources. You can think of it like a folder in a filesystem.