Learning Objectives

By the end of this chapter, you should be able to:

- Download installation and configuration tools.
- Install a Kubernetes master and grow a cluster.
- Configure a network solution for secure communications.
- Discuss highly-available deployment considerations.

Installation and Configuration

Installation Tools

This chapter is about Kubernetes installation and configuration. We are going to review a few installation mechanisms that you can use to create your own Kubernetes cluster.

To get started without having to dive right away into installing and configuring a cluster, there are two main choices.

One way is to use Google Container Engine (GKE), a cloud service from the Google Cloud Platform, that lets you request a Kubernetes cluster with the latest stable version.

Another easy way to get started is to use Minikube. It is a single binary which deploys into the Oracle VirtualBox software, which can run in several operating systems. While Minikube is local and single node, it will give you a learning, testing, and development platform. MicroK8s is a newer tool tool developed by Canonical, and aimed at easy installation. Aimed at appliance-like installations, it currently runs on Ubuntu 16.04 and 18.04.

To be able to use the Kubernetes cluster, you will need to have installed the Kubernetes command line, called **kubectl**. This runs locally on your machine and targets the API server endpoint. It allows you to create, manage, and delete all Kubernetes resources (e.g. Pods, Deployments, Services). It is a powerful CLI that we will use throughout the rest of this course. So, you should become familiar with it.

In this course, we will use **kubeadm**, the community-suggested tool from the Kubernetes project, that makes installing Kubernetes easy and avoids vendor-specific installers. Getting a cluster running involves two commands: **kubeadm init**, that you run on one Master node, and then, **kubeadm join**, that you run on your worker or redundant master nodes, and your cluster bootstraps itself. The flexibility of these tools allows Kubernetes to be deployed in a number of places. Lab exercises use this method.

We will also talk about other installation mechanisms, such as kubespray or kops, another way to create a Kubernetes cluster on AWS. We will note you can create your systemd unit file in a very traditional way. Additionally, you can use a container image called **hyperkube**, which contains all the key Kubernetes binaries, so that you can run a Kubernetes cluster by just starting a few containers on your nodes.

Installing kubectl

To configure and manage your cluster, you will probably use the **kubectl** command. You can use RESTful calls or the Go language, as well.

Enterprise Linux distributions have the various Kubernetes utilities and other files available in their repositories. For example, on RHEL 7/CentOS 7, you would find **kubectl** in the **kubernetes-client** package.

You can (if needed) download the code from GitHub, and go through the usual steps to compile and install **kubectl**.

This command line will use **~/.kube/config** as a configuration file. This contains all the Kubernetes endpoints that you might use. If you examine it, you will see cluster definitions (i.e. IP endpoints), credentials, and contexts.

A *context* is a combination of a cluster and user credentials. You can pass these parameters on the command line, or switch the shell between contexts with a command, as in:

$ kubectl config use-context foobar

This is handy when going from a local environment to a cluster in the cloud, or from one cluster to another, such as from development to production.

Installing with kubeadm

Once you become familiar with Kubernetes using Minikube, you may want to start building a real cluster. Currently, the most straightforward method is to use **kubeadm**, which appeared in Kubernetes v1.4.0, and can be used to bootstrap a cluster quickly. As the community has focused on **kubeadm**, it has moved from beta to stable and added high availability with v1.15.0.

The Kubernetes website provides documentation on how to use kubeadm to create a cluster.

Package repositories are available for Ubuntu 18.04 and CentOS 7.1. Packages have not yet been made available for Ubuntu 18.04, but will work as you will see in the lab exercises.

To join other nodes to the cluster, you will need at least one token and an SHA256 hash. This information is returned by the command **kubeadm init**. Once the master has initialized, you would apply a network plugin. Main steps:

- Run **kubeadm init** on the head node.
- Create a network for IP-per-Pod criteria.
- Run **kubeadm join --token *token head-node-IP*** on worker nodes.

You can also create the network with **kubectl** by using a resource manifest of the network.

For example, to use the Weave network, you would do the following:

$ kubectl create -f https://git.io/weave-kube

Once all the steps are completed, workers and other master nodes joined, you will have a functional multi-node Kubernetes cluster, and you will be able to use **kubectl** to interact with it.

Installing a Pod Network

Prior to initializing the Kubernetes cluster, the network must be considered and IP conflicts avoided. There are several Pod networking choices, in varying levels of development and feature set.

Many of the projects will mention the Container Network Interface (CNI), which is a CNCF project. Several container runtimes currently use CNI. As a standard to handle deployment management and cleanup of network resources, CNI will become more popular.

Pod Networking Choices

Close

Calico

A flat Layer 3 network which communicates without IP encapsulation, used in production with software such as Kubernetes, OpenShift, Docker, Mesos and OpenStack. Viewed as a simple and flexible networking model, it scales well for large environments. Another network option, Canal, also part of this project, allows for integration with Flannel. Allows for implementation of network policies.

For more details, check out the Project Calico web page.

Close

Flannel

A Layer 3 IPv4 network between the nodes of a cluster. Developed by CoreOS, it has a long history with Kubernetes. Focused on traffic between hosts, not how containers configure local networking, it can use one of several backend mechanisms, such as VXLAN. A flanneld agent on each node allocates subnet leases for the host. While it can be configured after deployment, it is much easier prior to any Pods being added.

You can learn more about Flannel from their GitHub pages.

Close

Kube-Router

Feature-filled single binary which claims to "*do it all*". The project is in the alpha stage, but promises to offer a distributed load balancer, firewall, and router purposely built for Kubernetes.

For more details, check out the Kube-Router web page.

Close

Romana

This is another project aimed at network and security automation for cloud native applications. Aimed at large clusters, IPAM-aware topology and integration with kops clusters.

To learn more, check out the Romana GitHub web page.

Close

Weave Net

It is typically used as an add-on for a CNI-enabled Kubernetes cluster.

To learn more, check out the Weave Net web page.

More Installation Tools

Since Kubernetes is, after all, like any other applications that you install on a server (whether physical or virtual), all of the configuration management systems (e.g., Chef, Puppet, Ansible, Terraform) can be used. Various recipes are available on the Internet.

The best way to learn how to install Kubernetes using step-by-step manual commands is to examine the Kelsey Hightower walkthrough.

Examples of Installation Tools

Close

kubespray

kubespray is now in the Kubernetes incubator. It is an advanced Ansible playbook which allows you to set up a Kubernetes cluster on various operating systems and use different network providers. It was once known as kargo.

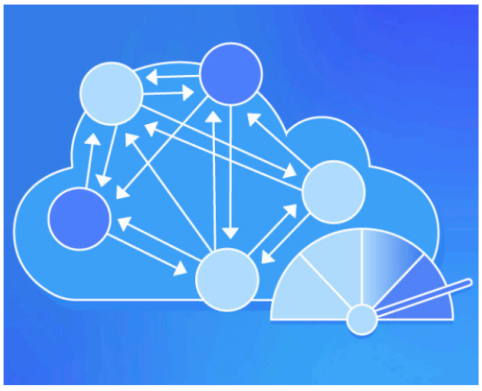
To learn more about kubespray, check out their GitHub page.

Close

kops

kops (Kubernetes Operations) lets you create a Kubernetes cluster on AWS via a single command line. Also in beta for GKE and alpha for VMware.

Learn more about kops from their GitHub page.



Close

kube-aws

kube-aws is a command line tool that makes use of the AWS Cloud Formation to provision a Kubernetes cluster on AWS.

For more details about kube-aws, check out their web page.

kubicorn

Close

kubicorn is a tool which leverages the use of kubeadm to build a cluster. It claims to have no dependency on DNS, runs on several operating systems, and uses snapshots to capture a cluster and move it.

For more information, check out the kubicorn web page.

Installation Considerations

To begin the installation process, you should start experimenting with a single-node deployment. This single-node will run all the Kubernetes components (e.g. API server, controller, scheduler, kubelet, and kube-proxy). You can do this with Minikube for example.

Once you want to deploy on a cluster of servers (physical or virtual), you will have many choices to make, just like with any other distributed system:

- Which provider should I use? A public or private cloud? Physical or virtual?
- Which operating system should I use? Kubernetes runs on most operating systems (e.g. Debian, Ubuntu, CentOS, etc.), plus on container-optimized OSes (e.g. CoreOS, Atomic).
- Which networking solution should I use? Do I need an overlay?
- Where should I run my etcd cluster?
- Can I configure Highly Available (HA) head nodes?

To learn more about how to choose the best options, you can read the *Picking the Right Solution* article.

With systemd becoming the dominant init system on Linux, your Kubernetes components will end up being run as *systemd unit files* in most cases. Or, they will be run via a kubelet running on the head node (i.e. **kubadm**).

Lab exercises in this course were written using Google Compute Engine (GCE) nodes. Each node has 2vCPUs and 7.5GB of memory, running Ubuntu 16.04. Smaller nodes should work, but you should expect slow response. Other operating system images are also possible, but there may be slight differences in some command outputs. Use of GCE requires setting up an account and will incur expenses if using nodes of the size suggested. You can view the *Getting Started* pages for more details.

Amazon Web Services (AWS) is another provider of cloud-based nodes, and requires an account; you will incur expenses for nodes of the suggested size. You can find videos and information of how to get started online.

Virtual machines such as KVM, VirtualBox, or VMware can also be used for lab systems. Putting the VMs on a private network can make troubleshooting easier.

Finally, using bare metal nodes with access to the Internet will also work for lab exercises.

Main Deployment Configurations

At a high level, you have four main deployment configurations:

- **Single-node**
  With a single-node deployment, all the components run on the same server. This is great for testing, learning, and developing around Kubernetes.
- **Single head node, multiple workers**
  Adding more workers, a single head node and multiple workers typically will consist of a single node etcd instance running on the head node with the API, the scheduler, and the controller-manager.
- **Multiple head nodes with HA, multiple workers**
  Multiple head nodes in an HA configuration and multiple workers add more durability to the cluster. The API server will be fronted by a load balancer, the scheduler and the controller-manager will elect a leader (which is configured via flags). The etcd setup can still be single node.
- **HA etcd, HA head nodes, multiple workers**
  The most advanced and resilient setup would be an HA etcd cluster, with HA head nodes and multiple workers. Also, etcd would run as a true cluster, which would provide HA and would run on nodes separate from the Kubernetes head nodes.

Which of the four you will use will depend on how advanced you are in your Kubernetes journey, but also on what your goals are.

The use of Kubernetes Federation also offers high availability. Multiple clusters are joined together with a common control plane allowing movement of resources from one cluster to another administratively or after failure. While Federation has has some issues, there is hope v2 will be a stronger product

systemd Unit File for Kubernetes

In any of these configurations, you will run some of the components as a standard system daemon. As an example, you can see here a sample systemd unit file to run the controller-manager. Using **kubeadm** will create a system daemon for kubelet, while the rest will deploy as containers:

```
- name: kube-controller-manager.service
    command: start
    content: |
      [Unit]
      Description=Kubernetes Controller Manager
      Documentation=https://github.com/kubernetes/kubernetes
      Requires=kube-apiserver.service
      After=kube-apiserver.service
      [Service]
      ExecStartPre=/usr/bin/curl -L -o /opt/bin/kube-controller-manager -z
/opt/bin/kube-controller-manager https://storage.googleapis.com/kubernetes-
release/release/v1.7.6/bin/linux/amd64/kube-controller-manager
      ExecStartPre=/usr/bin/chmod +x /opt/bin/kube-controller-manager
```

```
      ExecStart=/opt/bin/kube-controller-manager \
        --service-account-private-key-file=/opt/bin/kube-serviceaccount.key \
        --root-ca-file=/var/run/kubernetes/apiserver.crt \
        --master=127.0.0.1:8080 \
...
```

This is by no means a perfect unit file. It downloads the controller binary from the published release of Kubernetes and sets a few flags to run.

As you dive deeper in the configuration of each component, you will become more familiar not only with its configuration, but also with the various existing options, including those for authentication, authorization, HA, container runtime, etc. Expect them to change.

For example, the API server is highly configurable. The Kubernetes documentation provides more details about the kube-apiserver.

Compiling from Source

The list of binary releases is available on GitHub. Together with **gcloud**, **minikube**, and **kubeadmin**, these cover several scenarios to get started with Kubernetes.

Kubernetes can also be compiled from source relatively quickly. You can clone the repository from GitHub, and then use the **Makefile** to build the binaries. You can build them natively on your platform if you have a Golang environment properly set up, or via Docker containers if you are on a Docker host.

To build natively with Golang, first install Golang. Download files and directions can be found online.

Once Golang is working, you can clone the **kubernetes** repository, around 500MB in size. Change into the directory and use **make**:

$ cd $GOPATH

$ git clone https://github.com/kubernetes/kubernetes

$ cd kubernetes

$ make

On a Docker host, clone the repository anywhere you want and use the **make quick-release** command. The build will be done in Docker containers.

The **_output/bin** directory will contain the newly built binaries.

Lab Exercises

Lab 3.1 - Install Kubernetes

Lab 3.2 - Grow the Cluster

Lab 3.3 - Finish Cluster Setup

Lab 3.4 - Deploy a Simple Application

Lab 3.5 - Access from Outside the Cluster