Selfie Shots - Tools 304 Kubernetes

Persistent University

Kubernetes Installation

Key learning points :

- Introduction
- Installing kubectl
- Exploring Kubernetes installation options
  - Google Kubernetes Engine
  - Minikube
  - kubeadm
  - hyperkube
  - Compiling from Source
- Installation of Pod Networks
- Deployment configurations

Introduction

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 **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.
- In both cases, 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.
- We will have a look at **kubeadm**, which is a newer tool coming up in 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 a Master, and then, **kubeadm join**, that you run on your Worker Nodes, and your cluster bootstraps itself. The flexibility of these tools allows Kubernetes to be deployed in a number of places. We will be using this method on AWS nodes.
- We will also talk about other installation mechanisms that can be found in the community, such as **kubespray** or **kops**, another way to create a Kubernetes cluster on AWS. 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.

Using Google Kubernetes Engine (GKE)

- Google takes every Kubernetes release through rigorous testing and makes it available via its GKE service. To be able to use GKE, you will need the following:
- An account on Google Cloud.
- A method of payment for the services you will use.
- The gcloud command line client.
- There is an extensive documentation to get it installed. Pick your favorite method of installation and set it up. For more details, you can visit the Installing Cloud SDK web page.
- You will then be able to follow the GKE quickstart guide and you will be ready to create your first Kubernetes cluster:

```
$ gcloud container clusters create test-cluster

$ gcloud container clusters list

$ kubectl get nodes
```

- By installing gcloud, you will have automatically installed kubectl. In the commands above, we created the cluster, listed it, and then, listed the nodes of the cluster with kubectl.
- Once you are done, do not forget to delete your cluster, otherwise you will keep on getting charged for it:

```
$ gcloud container clusters delete test-cluster
```

Using Minikube

- You can also use **Minikube**, an open source project within the **GitHub** Kubernetes organization. While you can download a release from **GitHub**, following listed directions, it may be easier to download a pre-compiled binary. Make sure to verify and get the latest version.
- For example, to get the v.0.22.2 version, do:

```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.22.2/minikube-linux-amd64

$ chmod +x minikube

$ sudo mv minikube /usr/local/bin
```

- With **Minikube** now installed, starting **Kubernetes** on your local machine is very easy:

```
$ minikube start

$ kubectl get nodes
```

- This will start a **VirtualBox** virtual machine that will contain a single node **Kubernetes** deployment and the **Docker** engine. Internally, **minikube** runs a single Go binary called **localkube**. This binary runs all the components of Kubernetes together. This makes Minikube simpler than a full Kubernetes deployment. In addition, the Minikube VM also runs Docker, in order to be able to run containers.

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.
- The Kubernetes website provides documentation on how to use kubeadm to create a cluster.
- Package repositories are available for **Ubuntu 16.04** and **CentOS 7.1**.
- Run **kubeadm init** on the head node. The token is returned by the command, **kubeadm init**. Create a network for IP-per-Pod criteria. Run **kubeadm join --token token head-node-IP** on the worker nodes. You can also create the network with **kubectl**, by using a resource manifest of the network.
- Once all the steps are completed, 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:

- 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.

- 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.

- 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**.

- Romana
  Another project aimed at network and security automation for cloud native applications. Aimed at large clusters, IPAM-aware topology and integration with **kops** clusters.

- Weave Net
  Typically used as an add-on for a CNI-enabled **Kubernetes** cluster.

- 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, it will become more popular

More Installation Tools

- Since **Kubernetes** is, after all, like any other applications that you install on a server (whether physical or virtual), all the configuration management systems (e.g. **Chef**, **Puppet**, **Ansible**, **Terraform**) can be used. Various recipes are available on the Internet.
- Here are just a few examples of installation tools that you can use:
  - **kubespray** is now in the **Kubernetes** incubator. It is an advanced **Ansible** playbook which allows you to setup a **Kubernetes** cluster on various operating systems and use different network providers. It was once known as **kargo**.
  - **kops** lets you create a **Kubernetes** cluster on **AWS** via a single command line. Also in beta for **GKE** and alpha for **VMware**.
  - **kube-aws** is a command line tool that makes use of the **AWS Cloud Formation** to provision a **Kubernetes** cluster on **AWS**.
  - **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.

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.

Deployment Configurations

- At a high level, you have four main deployment configurations:
  - Single-node
  - Single head node, multiple workers
  - Multiple head nodes with HA, multiple workers
  - HA **etcd**, HA head nodes, multiple workers.
- 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.
- With a single-node deployment, all the components run on the same server. This is great for testing, learning, and developing around Kubernetes.
- 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 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.
- 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.
- The use of Kubernetes Federations 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.

Using Hyperkube

- While you can run all the components as regular system daemons in unit files, you can also run the API server, the scheduler, and the controller-manager as containers. This is what **kubeadm** does.

- Indeed, there is a very handy all-in-one binary named **hyperkube**, which is available as a container image (e.g. **gcr.io/google_containers/hyperkube:v1.9.2**).

- This method of installation consists in running a **kubelet** as a system daemon and configuring it to read in manifests that specify how to run the other components (i.e. the API server, the scheduler, **etcd**, the controller). In these manifests, the **hyperkube** image is used. The **kubelet** will watch over them and make sure they get restarted if they die.

- To get a feel for this, you can simply download the **hyperkube** image and run a container to get help usage:

```
$ docker run --rm gcr.io/google_containers/hyperkube:v1.9.2 /hyperkube apiserver --help

$ docker run --rm gcr.io/google_containers/hyperkube:v1.9.2 /hyperkube scheduler --help

$ docker run --rm gcr.io/google_containers/hyperkube:v1.9.2 /hyperkube controller-manager --help
```

- This is also a very good way to start learning the various configuration flags.

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 setup, 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. https://golang.org/doc/install.
- 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.

Summary

At the end of this session, we see that you are now 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.

Lab Exercise

- Install Kubernetes
- Deploy A Simple Application (e.g. nginx)