Chapter 2
Kubernetes basics

This chapter introduces kubernetes, what is it, the basic terminologies and key concepts of it. You will learn most of the commonly used and mentioned components in kubernetes architecture. This chapter also provides an example to launch a "minimal" kubernetes environment to demonstrate how to interact with kubernetes in practice and how the virtual environment orchestrated by kubernetes looks like.

What is Kubernetes

this is the description from the main page of official kubernetes website (https://kubernetes.io/) :

> Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.

this brief description tells a few important facts about kubernetes:

- initiated by google but open-sourced now
- mature and stable product after many years effort
- orchestration tool
- dealing with containers

kubernetes was created by a group of engineers in google in 2014, with a design and development model heavily influenced by Google's internal system named "Borg" (still being used in google internally today). Kubernetes defines a set of "building objects" (e.g. "pod", "service") which collectively provides mechanisms that orchestrates containerized applications across a distributed cluster of nodes, based on system resources (CPU, memory or other custom metrics). Kubernetes hides the complexity of managing a group of containers by providing REST APIs for the required functionalities.

In simple words, container technologies like docker provides you the capability of packaging and distributing containerized applications, while an orchestration system like kubernetes allows you to deploy and manage the dockers in a relatively higher level and a much easier way.

TIP

- in many document kubernetes is frequently abbreviated as "k8s" (K - eight characters - S),
- the "current" (as of the writing of this book) major release is v1.14.

Kubernetes Architecture and components

in a Kubernetes cluster there are two type of nodes, each running a very well-defined set of processes:

- head node: called "master", or "master node", brain of kubernetes cluster, all of intelligence are located here.
- worker node: called "node", or "union" in old document, the workforce of cluster.

Kubernetes master

A master node provides the cluster's control plane. Master node makes "global decisions" about the cluster. for example, master does scheduling and decide which node will spawn a container. master is also responsible for maintaining the desired state for the cluster. One of the most common interface between you and the cluster is a command-line tool "kubectl". When you are interacting with Kubernetes by "kubectl" command, you're essentially communicating with the cluster's "master".

The main functions of cluster is implemented by a collection of processes running in the master node. Typically you only need a single master node in the cluster, however, the master can also be replicated for higher availability (HA) and redundancy.

The processes in a master node providing the primary features are:

- **kube-apiserver**: front-end of the control plane, providing REST APIs
- **kube-scheduler**: decide where to place the containers depending on system requirement (CPU, memory, harddisk, etc) and other custom parameters (e.g. affinity specification)
- **kube-controller-manager**: the single process implementing most of the "controllers", which makes sure that the state of the system is what it should be. TODO
    - RC(Replication Controller)
    - RS(ReplicaSet)
    - Deployment
    - StatefulSet
    - DS(DaemonSet)
    - Job
    - Node Controller
    - Service Controller
- **cloud-controller-manager**: TODO
- **etcd**: data store to store the state of the system.
- **DNS server** for Kubernetes services.
- **kubelet**: TODO

Kubernetes node

nodes in a cluster are the machines that run the applications. in production there can be dozens or hundreds of nodes in one cluster depending on the designed scales. nodes are the real workforce under the hood provided by a cluter. The Kubernetes master controls each node and you'll rarely "bypass" the master and interact with nodes directly.

A "node" runs following processes:

- **Kubelet**: the Kubernetes agent process that runs on all the nodes. it interacts with kube-apiserver and manage the containers in local host.
- **kube-proxy**: process that implements "kubernetes service" (wil introduce later) using linux iptable in the node
- **container-runtime** - local container - mostly docker in today's market, holding all of the "dockerized" applications.

TIP: the name "proxy" may sound confusing for kubernetes beginners. it's not really a "proxy" in current kubernetes architecture. kube-proxy is a system that manipulates linux IP tables in that node so that the the traffic between the pods and the nodes will flows correctly.

kubernetes objects

docker vs kubernetes

Now you understand the role of master and nodes in a kubernetes cluster, it is the time to introduce more concepts in the architecture.

as mentioned earlier, technically speaking, kubernetes works in a relatively higher level than dockers. what does that really mean? Assuming you want to run multiple containers across multiple machines, you will have a lot of work to do if you interact with docker directly. You need to:

- Running containers across many different machines
- Scaling up or down by adding or removing containers when demand changes
- Keeping storage consistent with multiple instances of an application
- Distributing load between the containers
- Launching new containers on different machines if something fails

Doing all of this manually with docker will be overwhelming. with kubernetes all of these tasks become much easiler.

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: myweb
spec:
  replicas: 2       #<-----
  selector:
    app: myweb
  template:
    metadata:
      labels:
        app: myweb
    spec:
      containers:
      - name: myweb
        image: kubeguide/tomcat-app:v1
        ports:
        - containerPort: 8080
```

create the RCs

```
    ```bash
    kubectl create -f mysql_rc.yaml
    kubectl create -f myweb_rc.yaml
    ```
list the created RCs
```

```
$ kubectl get rc
NAME      DESIRED   CURRENT   READY     AGE
mysql     1         1         0         10s         #<------
myweb     2         2         2         10s

$ kubectl get pod
NAME          READY     STATUS              RESTARTS   AGE
mysql-d96z2   0/1       ContainerCreating   0          1m   #<------
myweb-nv4h8   1/1       Running             1          1m
myweb-vzvk4   1/1       Running             1          1m

$ kubectl get pod
NAME          READY     STATUS    RESTARTS   AGE
mysql-d96z2   1/1       Running   0          2m     #<------
myweb-nv4h8   1/1       Running   1          2m
myweb-vzvk4   1/1       Running   1          2m
```

in the frontend, kubernetes get all these things done via a group of abstractions, each represented in the form of an "object". with kubernetes you only needs to think of how to describe your task in the config file, without the need to worry about how it will be implemented.

"under the hood", kubernetes interact with the Docker engine to coordinate the scheduling and execution of Docker containers on Kubelets. The Docker engine itself is responsible for running the actual container image (e.g. by 'docker build').

Higher level concepts such as service-discovery, loadbalancing and network policies are handled by Kubernetes as well.

TODO: use ftp/tcp/ip as example.

features and abstractions

features, objects, abstractions, processes, controllers

Kubernetes contains a number of abstractions that represent the state of your system: deployed containerized applications and workloads, their associated network and disk resources, and other information about what your cluster is doing. These abstractions are represented by objects in the Kubernetes API; see the Kubernetes Objects overview for more details.

The basic Kubernetes objects include:

- Pod
- Service
- Volume
- Namespace

In addition, Kubernetes contains a number of higher-level abstractions called Controllers. Controllers build upon the basic objects, and provide additional functionality and convenience features. They include:

- ReplicaSet
- Deployment
- StatefulSet
- DaemonSet
- Job

kubectl

now let’s talk about kubectl - the tool you will need to interact with all these abstractions/objects.

autocompletion

Kubernetes networking

ip-per-pod model

a "full picture" - put everything together

Building Kubernetes POD

YAML file for Kubernetes

POD example using YAML files

Kubectl tool

Login to container

Intra-POD communications

Inter-POD communications