Kubernetes – From the Firehose

Objectives

Key objectives of this chapter

- Masters
- Nodes
- Pods
- Namespaces
- Resource Quota
- Authentication and Authorization
- Routing
- Registry
- Storage Volumes

1.1 What is Kubernetes?

- Kubernetes 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.
- Designed on the same principles that allows Google to run billions of containers a week, Kubernetes can scale without increasing your ops team.
- Whether testing locally or running a global enterprise, Kubernetes flexibility grows with you to deliver your applications consistently and easily no matter how complex your need is
- Kubernetes is open source giving you the freedom to take advantage of on-premises, hybrid, or public cloud infrastructure, letting you effortlessly move workloads to where it matters to you.

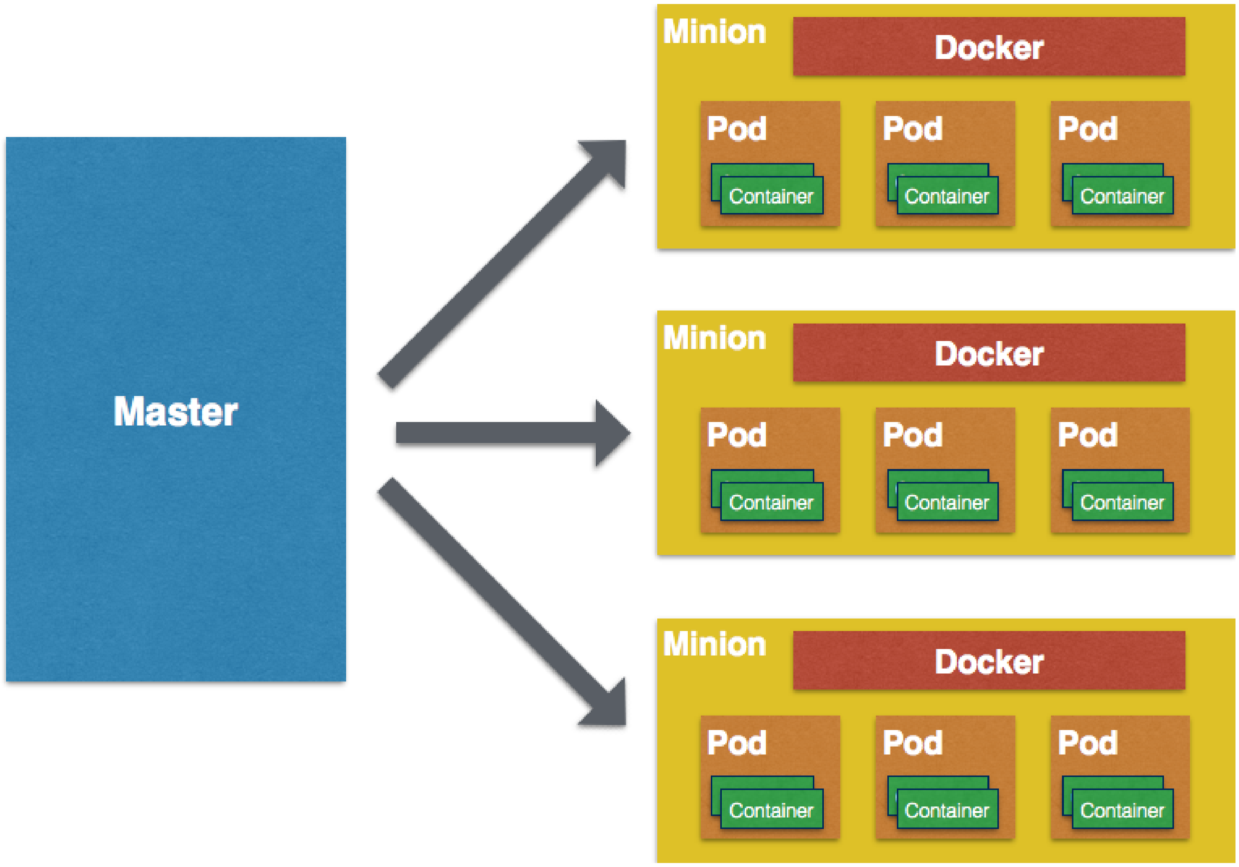
- Kubernetes can be deployed on a bare-metal cluster (real machines) or on a cluster of virtual machines.

1.2 Container Orchestration

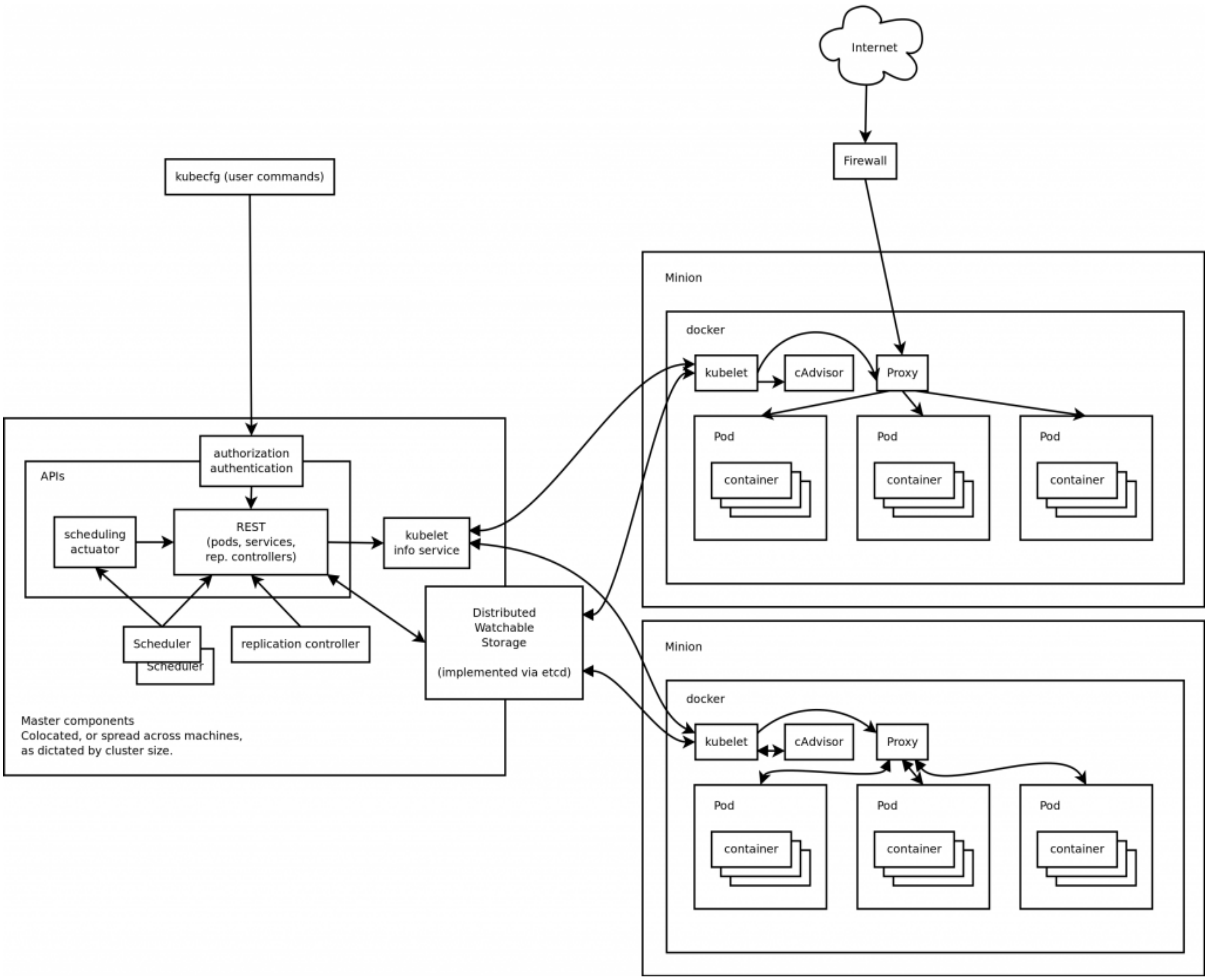
- The primary responsibility of Kubernetes is container orchestration.
- Kubernetes ensures that all the containers that execute various workloads are scheduled to run physical or virtual machines
- The containers must be packed efficiently following the constraints of the deployment environment and the cluster configuration
- Kubernetes keeps an eye on all running containers and replaces dead, unresponsive, or otherwise healthy containers.
- Kubernetes can orchestrate the containers it manages directly on bare-metal or on virtual machines
- A Kubernetes cluster can also be composed of a mix of bare-metal and virtual machines, but this is not very common.
- Containers are ideal to package microservices because, while providing isolation to the microservice, they are very lightweight compared virtual machines. This makes containers ideal for cloud deployment

1.3 Kubernetes Basic Architecture

- At a very high level, there are three key concepts
  - Pods
  - Master
  - Minions (old term) / Nodes (new term)



1.4 Kubernetes Detailed Architecture



1.5 Kubernetes Concepts

- Cluster and Namespace
- Node
- Master
- Pod
- Label
- Annotation
- Label Selector
- Replication Controller and replica set
- Services
- Volume
- Secret

1.6 Cluster and Namespace

- Cluster
  - A collection of physical resources, such as hosts storage and networking resources
  - The entire system may consist of multiple clusters
- Namespace
  - It is a virtual cluster
  - A single physical cluster can contain multiple virtual clusters segregated by namespaces
  - Virtual clusters can communicate through public interfaces

- Pods can live in a namespace, but nodes can not.
- Kubernetes can schedule pods from different namespaces to run on the same node

1.7 Node

- A single host
- It may be a physical or virtual machine
- It's job is to run pods
- Each node runs several Kubernetes components, such as a kubelet and a kube proxy
- kubelet is a service which reads container manifests as YAML files that describes a pod.
- Nodes are managed by a Kubernetes master
- The nodes are worker bees of Kubernetes and shoulder all the heavy lifting
- In the past they were called minions.

1.8 Master

- The master is the control plane of Kubernetes
- It consists of components, such as
  - API server
  - a scheduler
  - a controller manager
- The master is responsible for the global, cluster-level scheduling of pods and handling events.

- Often, all the master components are set up on a single host
- For implementing high-availability scenarios or very large clusters, you will want to have master redundancy.

1.9 Pod

- A pod is the unit of work on Kubernetes
- Each pod contains one or more containers
- Pods provide a solution for managing groups of closely related containers that depend on each other and need to cooperate on the same host
- Pods are considered throwaway entities that can be discarded and replaced at will (i.e. they are cattle, not pets)
- Each pod gets a unique ID (UID)
- Pods are always scheduled together and always run on the same machine
- All the containers in a pod have the same IP address and port space
- The containers within a pod can communicate using localhost or standard inter-process communication
- The containers within a pod have access to shared local storage on the node hosting the pod and is mounted on each container
- The benefit of grouping related containers within a pod, as opposed to having one container with multiple applications, are:
  - Transparency – making the containers within the pod visible to the infrastructure enables the infrastructure to provide services to those containers, such as process management and resource monitoring
  - Decoupling software dependencies – the individual containers maybe be versioned, rebuilt, and redeployed independently
  - Ease of use – users don't need to run their own process managers

- Efficiency – because the infrastructure takes on more responsibility, containers can be more lightweight

1.10 Label

- Labels are key-value pairs that are used to group together sets of objects, often pods.
- Labels are important for several other concepts, such as replication controller, replica sets, and services that need to identify the members of the group
- Each pod can have multiple labels, and each label may be assigned to different pods.
- Each label on a pod must have a unique key
- The label key must adhere to a strict syntax
  - Label has two parts: prefix and name
  - Prefix is optional. If it exists then it is separated from the name by a forward slash (/) and it must be a valid DNS sub-domain. The prefix must be 253 characters long at most
  - Name is mandatory and must be 63 characters long at most. Name must begin with an alphanumeric character and contain only alphanumeric characters, dots, dashes, and underscores. You can create another object with the same name as the deleted object, but the UIDs must be unique across the lifetime of the cluster. UIDs are generated by Kubernetes
  - Values follow the same restrictions as names

1.11 Annotation

- Unlike labels, annotation can be used to associate arbitrary metadata with Kubernetes objects.
- Kubernetes stores the annotations and makes their metadata available
- Unlike labels, annotations don't have strict restrictions about allowed characters and size limits

1.12 Label Selector

- They are used to select objects based on their labels
- A value can be assigned to a key name using equality-based selectors, (=, ==, !=).
  - e.g.
    - role = webserver
    - role = dbserver, application != sales
- **in** operator can be used as a set-based selector
  - .e.g
    - role in (dbserver, backend, webserver)

1.13 Replication Controller and Replica Set

- They both manage a group of pods identified by a label selector
- They ensure that a certain number of pods are always up and running
- Whenever the number drops due to a problem with the hosting node or the pod itself, Kubernetes fires up new instances

- If you manually start pods and exceed the specified number, the replication replication controller kills some extra pods
- Replication controllers test for membership by name equality, whereas replica sets can use set-based selection
- Replica sets are newer and considered as next-gen replication controllers

1.14 Service

- Services are used to expose some functionality to users or other services
- They usually involve a group of pods, usually identified by a label
- Kubernetes services are exposed through endpoints (TCP/UDP)
- Services are published or discovered via DNS, or environment variables
- Services can be load-balanced by Kubernetes

1.15 Storage Volume

- When a pod is destroyed, the data used by the pod is also destroyed.
- If you want the data to outlive the pod or share data between pods, volume concept can be utilized.

1.16 Secret

- Secrets are small objects that contain sensitive info, such as credentials
- They are stored as plain-text in etcd
- They can be mounted as files into pods
- The same secret can be mounted into multiple pods

- Internally, Kubernetes creates secrets for its components, and you can create your own secrets

1.17 Resource Quota

- Kubernetes allows management of different types of quota
- Compute resource quota
  - Compute resources are CPU and memory
  - You can specify a limit or request a certain amount
  - Uses fields, such as requests.cpu, requests. memory
- Storage resource quota
  - You can specify the amount of storage and the number of persistent volumes
  - Uses fields, such as requests.storage, persistentvolumeclaims
- Object count quota
  - You can control API objects, such as replication controllers, pods, services, and secrets
  - You can not limit API objects, such as replica sets and namespaces.

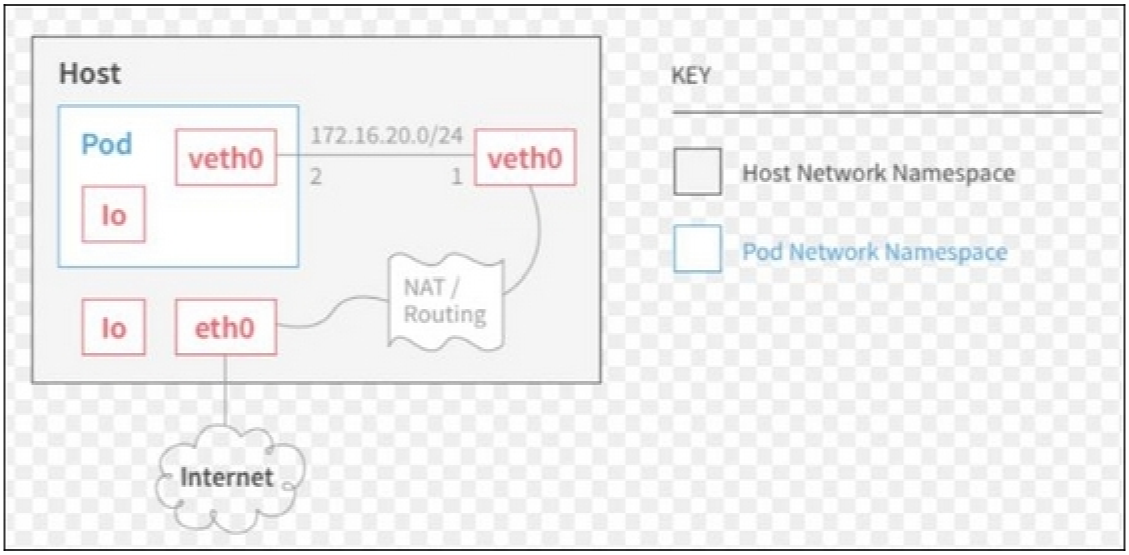
1.18 Authentication and Authorization

- Permission rules can be added to the Kubernetes system for more advanced management
- Applying authentication and authorization is a secure solution to prevent your data being accessed by others.
- Authentication is currently supported in the form of tokens, passwords, and certificates.

- Authorization supports three modes:
  - RBAC (Role-Based Access Control)
  - ABAC (Attribute-Based Access Control) – lets a user define privileges via attributes in a file
  - Webhook – allows for integration with third-party authorization via REST web service calls.

1.19 Routing

- Routing connects separate networks
- Routing is based on routing tables
- Routing table instructs network devices how to forward packets to their destination
- Routing is done through various network devices, such as routers, bridges, gateways, switches, and firewalls



1.20 Registry

- Container images aren't very useful if it's only available on a single machine
- Kubernetes relies on the fact that images described in a Pod manifest are available across every machine in the cluster
- Container images can be stored in a remote registry so every machine in the cluster can utilize the images.
- Registry can be public or private.
- Public registries allow anyone to download images (e.g. Docker Hub), while private registries require authentication to download images (e.g Docker Registry)
- Docker Registry is a stateless, highly scalable-server side application that stores and lets you distribute Docker images.
- Docker Registry is open-source
- Docker Registry gives you following benefits
  - tight control where your images are being stored
  - fully own your images distribution pipeline
  - integrate image storage and distribution tightly into your in-house development workflow

1.21 Using Docker Registry

- Default storage location is

```
/var/lib/registry
```

- Change storage location by creating and environment variable like this

```
REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY=/somewhere
```

- Start your registry (:2 is the version. Check Docker Hub for latest version)

```
docker run -d -p 5000:5000 -name registry registry:2.6
```

- Pull some image from the hub

```
docker pull ubuntu
```

- Tag the image so that it points to your registry

```
docker tag ubuntu localhost:5000/myfirstimage
```

- Push it

```
docker push localhost:5000/myfirstimage
```

- Pull it back

```
docker pull localhost:5000/myfirstimage
```

- Stop your registry and remove all data

```
docker stop registry && docker rm -v registry
```

1.22 Summary

- Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications
- The primary responsibility of Kubernetes is container orchestration.
- At a high level, Kubernetes involves Pods, Master, and Nodes
- Kubernetes also involves labels, annotations, replication controllers, replica sets, and secrets
- Container images can be deployed to a public or private registry