Configurable and Scalable IITBombayX MOOC platform on commodity servers

Amit Kumar Tiwari, Harshit Mogalapalli, Mridul Mahajan, Ritik Kumar, Soumyakant Mohakul

Last Updated: July 2, 2019

List of Figures

List of Tables

Contents

1 Introduction

The aim of this project is to deploy an instance of Open edX MOOC platform on a multi-node cluster using Kubernetes as a container orchestrator.

Technologies used:

- Docker
- Kubernetes
- Ansible
- Shell scripting
- Git
- Cluster management and configuration

2 Open edX

Open edX is an open source online MOOC provider. It provides both web and mobile platforms to clone the basic template of e-learning platform. In this project, we are working on the web platform.

2.1 Releases

Open edX has multiple releases. Some of the prominent ones are:

- Ironwood
- Hawthorn
- Ginkgo

We are using the Ironwood.1 release in this project.

2.2 Installation

There are two ways to install Open edX:

- Native installation
- Docker-based installation

2.2.1 Native installation

Write text here.

2.2.2 Docker-based installation

i) Installing docker engine, docker machine and docker compose:

The docker based installing allows the resources of your system to be utilized in an optimal way. Docker is based on the principle of containerization. For proper working of the devstack we need the Docker engine, docker machine and docker compose. We shall install each of them one after another.

For installing the docker engine the following commands need to be run:

1. sudo apt-get update
2. sudo apt-get install
   apt-transport-https ca-certificates curl gnupg-agent software-properties-common
3. curl -fsSL https://download.docker.com/linux/ubuntu/gpg — sudo apt-key add -
4. sudo apt-key fingerprint 0EBFCD88
5. sudo add-apt-repository
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu
   $(lsb_release -cs)
   stable"
6. sudo apt-get update
7. sudo apt-get install docker-ce docker-ce-cli containerd.io
8. apt-cache madison docker-ce
9. sudo apt-get install docker-ce=$< VERSION\_STRING >$
   docker-ce-cli=$< VERSION\_STRING >$ containerd.io

In order to test the successfull installation of Docker, run the following command:
sudo docker run hello-world

For installing the docker machine the following command needs to be run:

base=https://github.com/docker/machine/releases/download/v0.16.0 &&
curl -L $base/docker-machine-$(uname -s)-$(uname -m)>/tmp/docker-machine &&
sudo install /tmp/docker-machine /usr/local/bin/docker-machine

For installing docker compose the following commands need to be followed:

1. sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
2. sudo chmod +x /usr/local/bin/docker-compose

For checking the successful installation of all the above you can use the - -version flag with the corresponding command to check the installed version.

ii) Installing Ironwood Devstack:

Devstack comes with different variations as per the different versions. The three popular versions of the devstack are Ginkgo, Hawthron and Ironwood.

The following commands perform the Ironwood-release installation of the devstack:

1. git clone https://github.com/edx/devstack
2. cd devstack
3. git checkout open-release/ironwood.master
4. export OPENEDX_RELEASE=ironwood.master
5. make dev.checkout
6. make dev.clone
7. make dev.provision

On successfully running dev.provision you will have 15 docker-containers running. The list of them can be obtained using command: docker ps

The list of docker-containers are: Docker-containers List

The Provision Log for the tasks executed are: provision.log

3 Docker

3.1 About Docker

Docker is an open-source software tool designed to automate and ease the process of creating, packaging, and deploying applications using an environment called a container. The use of Linux containers to deploy applications is called containerization. A Container allows us to package an application with all of the parts needed to run an application (code, system tools, logs, libraries, configuration settings and other dependencies) and sends it out as a single standalone package deployable via Ubuntu (in this case 16.04 LTS). Docker can be installed on other platforms as well. Currently, the Docker software is maintained by the Docker community and Docker Inc. Check out the official documentation to find more specifics on Docker. Docker Terms and Concepts.

3.2 Terms related to Docker

3.2.1 Containerization

Containerization is a lightweight alternative to full machine virtualization (like VMWare) that involves encapsulating an application within a container with its own operating environment.

3.2.2 Docker Image

A *Docker Image* is the basic unit for deploying a Docker container. A Docker image is essentially a static snapshot of a container, incorporating all of the objects needed to run a container.

3.2.3 Docker Container

A *Docker Container* encapsulates a Docker image and when live and running, is considered a container. Each container runs isolated in the host machine.

3.2.4 Docker Registry

The *Docker Registry* is a stateless, highly scalable server-side application that stores and distributes Docker images. This registry holds Docker images, along with their versions and, it can provide both public and private storage location. There is a public Docker registry called Docker Hub which provides a free-to-use, hosted Registry, plus additional features like organization accounts, automated builds, and more. Users interact with a registry by using Docker push or pull commands.

3.2.5 Docker Engine

The *Docker Engine* is a layer which exists between containers and the Linux kernel and runs the containers. It is also known as the Docker daemon. Any Docker container can run on any server that has the Docker-daemon enabled, regardless of the underlying operating system.

3.2.6 Docker Compose

Docker Compose is a tool that defines, manages and controls multi-container Docker applications. With Compose, a single configuration file is used to set up all of your applications services. Then, using a single command, you can create and start all the services from that file

3.2.7 Dockerfiles

Dockerfiles are merely text documents (.yaml files) that contains all of the configuration information and commands needed to assemble a container image. With a Dockerfile, the Docker daemon can automatically build the container image.

3.3 Docker-Machine

Write text here

3.4 Docker-Compose

Docker compose is a tool to define and run docker applications using multiple containers. The services of the applications can be configured using a YAML file which is by default named as docker-compose.yml. Docker compose enables the users to create and start all the services from the configuration file with a single command.

3.5 Docker Swarm

Write text here

4 Ansible

4.1 Installing openssh

$ sudo apt-get install openssh-server
$ sudo apt-get install openssh-client

4.2 Generating key at server and copying in client

$ ssh-keygen (generate ssh-key)
Copy this key to all nodes so that you dont have to enter a password each time while connecting to the node.
To copy ssh-key type command:
$ ssh-copy-id $username@IP\_address$

4.3 Ansible setup

4.3.1 Adding repository

$ sudo apt-add-repository ppa:ansible/ansible
$ sudo apt-get update

4.3.2 Installing ansible

$ sudo apt-get install ansible

4.3.3 Checking version

$ ansible –version

4.3.4 Ansible directories

$ ls -lha /etc/ansible/

4.3.5 Making a copy a directory for our use

$ cp -R /etc/ansible/ myplatform
vi ansible.cfg (uncomment the inventory = hosts)
vi hosts (remove all and write all nodes in new line)

You can check if all the nodes working by running:
$ ansible -m ping all

4.4 Some useful commands for ansible

4.4.1 To see the hostnames of all nodes

$ ansible -m shell -a 'hostname' all

4.4.2 To see the volumes used by all nodes

$ ansible -m shell -a 'df -h' all

4.4.3 To run an ansible-playbook

$ ansible-playbook -K playbook-docker.yml

5 Kubernetes

5.1 What is Kubernetes

Kubernetes is a container-orchestration tool developed by Google in 2015 and later donated to CNCF (Cloud Native Computing Foundation). Kubernetes provides a smart way for orchestration in setting a multi-node cluster. Kubernetes comes with a lot of functionalities such as load balancing, pod scaling, process scheduling and process management.

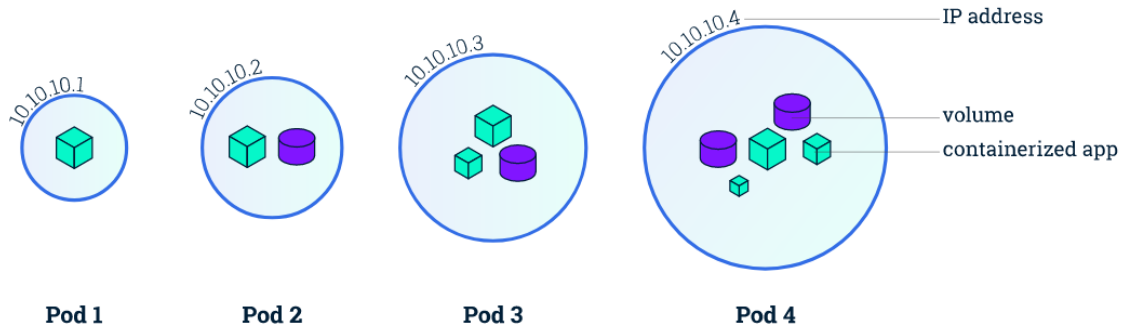
5.2 Main Features Of Kubernetes

- Service discovery and load balancing
- Storage orchestration
- Automatic bin packing
- Self-healing
- Automated rollouts and rollbacks
- Secret and configuration management
- Batch execution
- Horizontal scaling

5.3 Terminologies in Kubernetes

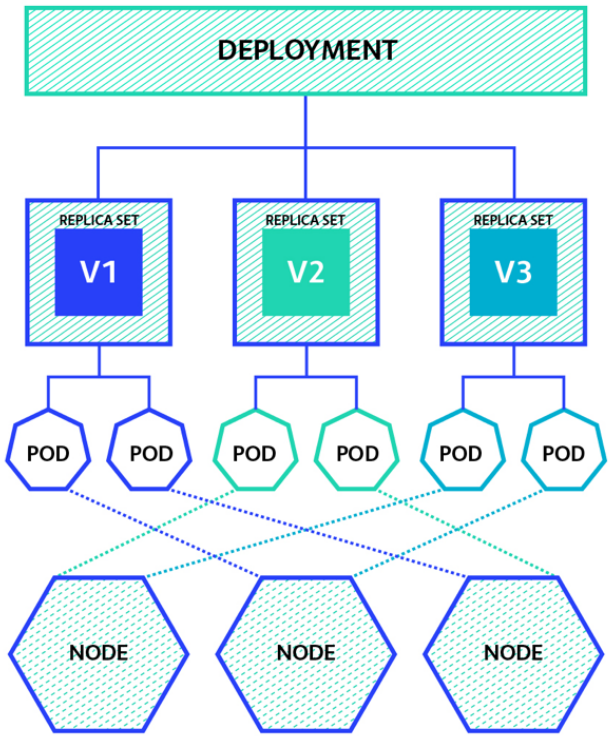
5.3.1 Pod

The most fundamental unit in a kubernetes cluster is a pod. A pod is a collection of container and volumes. Each pod has an IP address.



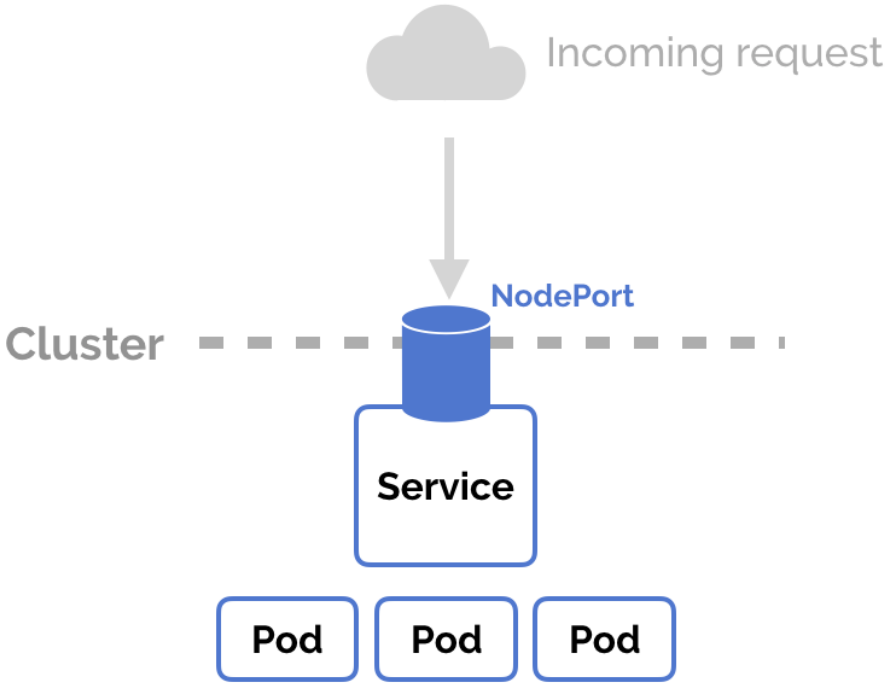
5.3.2 Deployment

A deployment can be seen as a stateless state of the pod. A deployment is used to provide pod definition and rolling updates to the pod. A deployment contains information such as number of replicas, image, volume mount, hostname and restart policy.



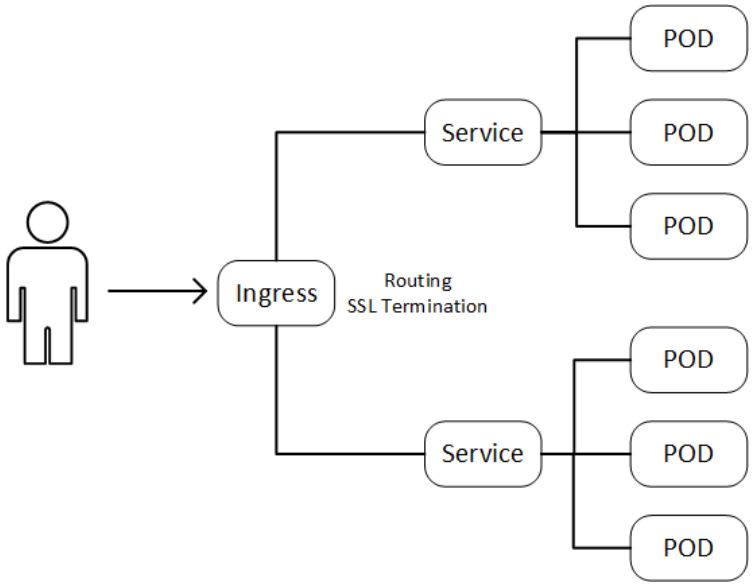
5.3.3 Service

A service exposes a deployment as a network service. A deployment is stateless whereas a service can be considered as a stateful definition. The three types of services in kubernetes are: ClusterIP, NodePort and LoadBalancer.



5.3.4 Ingress network

An ingress network exposes the services in the cluster to the outside network i.e. the clients. An ingress helps us in handling the outside traffic and directing them to the various services.



5.4 Networking in Kubernetes

Kubernetes supports a large number of network plugins using different protocols. The type of networking to be used completely depends on the type of cluster you want to set up and your cluster requirements.

In a kubernetes network each pod is assigned its own IP address. Pods can communicate with pods, nodes and services in a node without NAT.

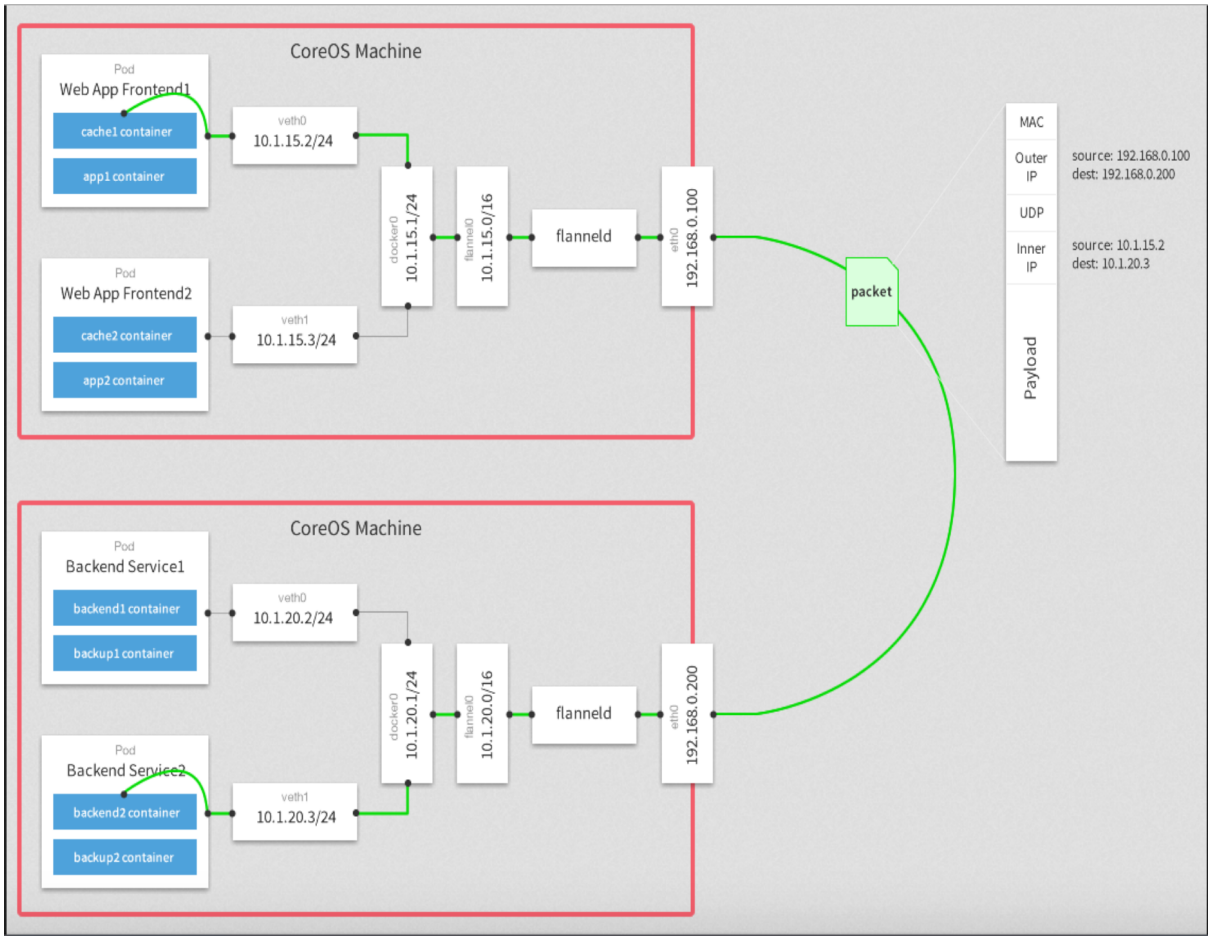
The networking in kubernetes can be broken down into majorly 4 parts :

1. Container to container communication
2. Pod to pod communication
3. Pod to service communication
4. Service to external Communication

The major plugins used for achieving these communications are :

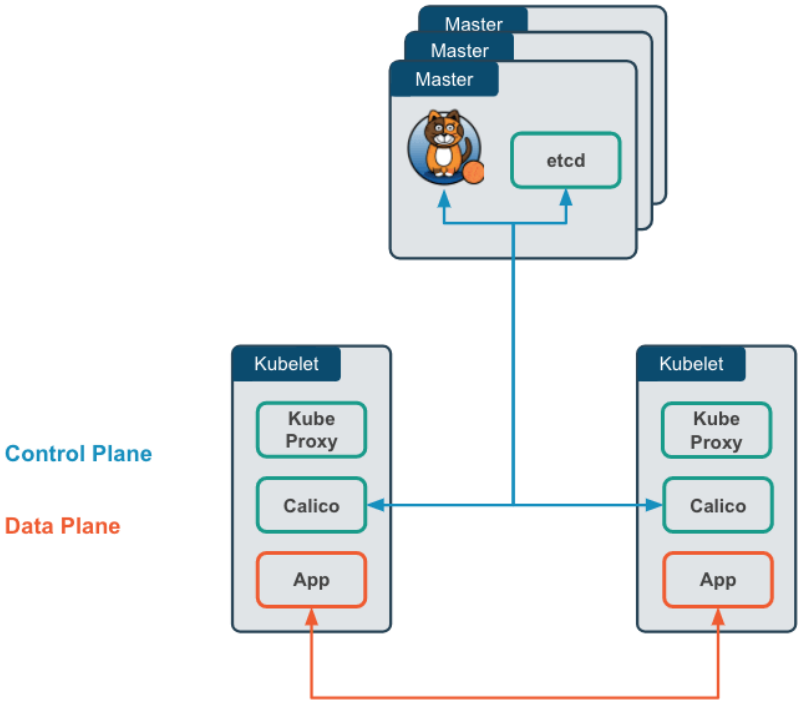
5.4.1 Flannel

Flannel is the simplest kubernetes network which satisfies all kubernetes requirements. Flannel is basically an overlay network.



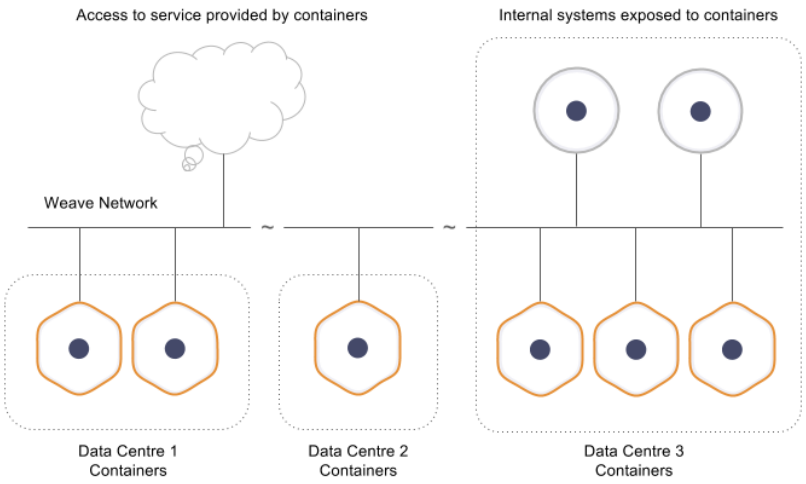
5.4.2 Calico

Calico provides a highly scalable networking and network policy solution for connecting Kubernetes pods based on the same IP networking principles as the internet, for both Linux and Windows. Calico can be deployed without encapsulation or overlays to provide high-performance, high-scale data center networking.



5.4.3 Weavenet

Weavenet is a simple network for kubernetes and its hosted applications. Weave Net runs as a CNI plug in or stand-alone.



In addition to these we have a lot of network plugins supported by kubernetes. A list of them can be found in the official documentation page of kubernetes (Networking in Kubernetes).

5.5 Prerequisites for running kubernetes

5.5.1 Disable Swap Space

In order to set up a kubernetes cluster all nodes including master must have swap space disabled.

- To disable swap space for current session run:
  $ sudo swapoff -a
- To permanently disable swap space go to /etc/fstab and comment the swap space line.

5.5.2 Make IP static

All nodes in cluster including master must have a static IP address.
In order to manually make the IP static add the following lines to the file /etc/network/interfaces:

auto wlo1
iface wlo1 inet static
address YOUR_IP_ADDRESS

5.6 Kubernetes setup

1. Install docker (follow the steps given here to successfully install docker).
2. Enable docker to autostart during restart:
   $ sudo systemctl enable docker
3. Install curl:
   $ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add
4. Add Googles kubernetes repository:
   $ sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
5. Install kubeadm, kubectl and kubelet:
   $ sudo apt-get install kubeadm
   $ sudo apt-get install kubectl
   $ sudo apt-get install kubelet

5.7 Setting up your Kubernetes cluster

In order to set up your cluster first initialise your master with kubeadm:

$ kubeadm init --apiserver-advertise-address=YOUR_IP_ADDRESS --pod-network-cidr=40.196.0.0/16

On successful execution of the command we get a join command for the worker nodes to join the cluster. Preserve this command to bring nodes into the cluster once the dashboard is created.

You can also generate the join token by running the command :

$ kubeadm token create

And the SHA256 key can be generated by running the command:

$ openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex — sed 's/^.* //'

Now to bring any node into the cluster run the following command into the cluster after changing the IP with your master systems IP address, token and SHA256 key with that generated by running the above command:

$ kubeadm join IP_OF_MASTER:6443 --token GENERATED_TOKEN --discovery-token-ca-cert-hash sha256:GENERATED_HASH

Now once you have successfully initialised the cluster run the following commands :

$ mkdir -p $HOME/.kube
$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

5.7.1 Setting up a pod network

i) To install and set up a calico pod network run the command :

$ kubectl apply -f https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installation/hosted/kubeadm/1.7/calico.yaml

ii) To install and set up flannel pod network run the command :

$ sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

There are a lot of plugins available that can be used as per the requirements. In this project we have used a calico pod network as CNI.
Once the network is set up we need to install and host the dashboard on localhost.

5.7.2 Setting up and hosting the dashboard

i) For installation and hosting in localhost :

$ kubectl create -f https://raw.githubusercontent.com/kubernetes/dashboard/v1.8.3/src/deploy/recommended/kubernetes-dashboard.yaml

ii) For Creating service account:

$ kubectl create serviceaccount dashboard -n default

iii) To add cluster binding rules to the dashboard:

$ kubectl create clusterrolebinding dashboard-admin -n default
--clusterrole=cluster-admin
--serviceaccount=default:dashboard

iv) To generate token for login

$ kubectl get secret $(kubectl get serviceaccount dashboard
-o jsonpath=".secrets[0].name") -o jsonpath=".data.token" | base64 --decode

After the token is generated you can run the dashboard on localhost on port 8001(default). To bring up the dashboard run the command :

$ kubectl proxy

Now you can visit the dashboard on the localhost:8001 at url:

http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/

The dashboard can be accessed by entering the token generated. We can see the pods running, deployments, services and volumes created. We can also scale up and scale down the number of replicas.

5.7.3 Kubernetes commands for getting resource info

In order to get the complete details of the pods via command line run the command :

$ kubectl get pods -o wide --all-namespaces

For viewing pods in default namespace run the command:

$ kubectl get pods

Similarly for viewing deployments and services run the corresponding commands:

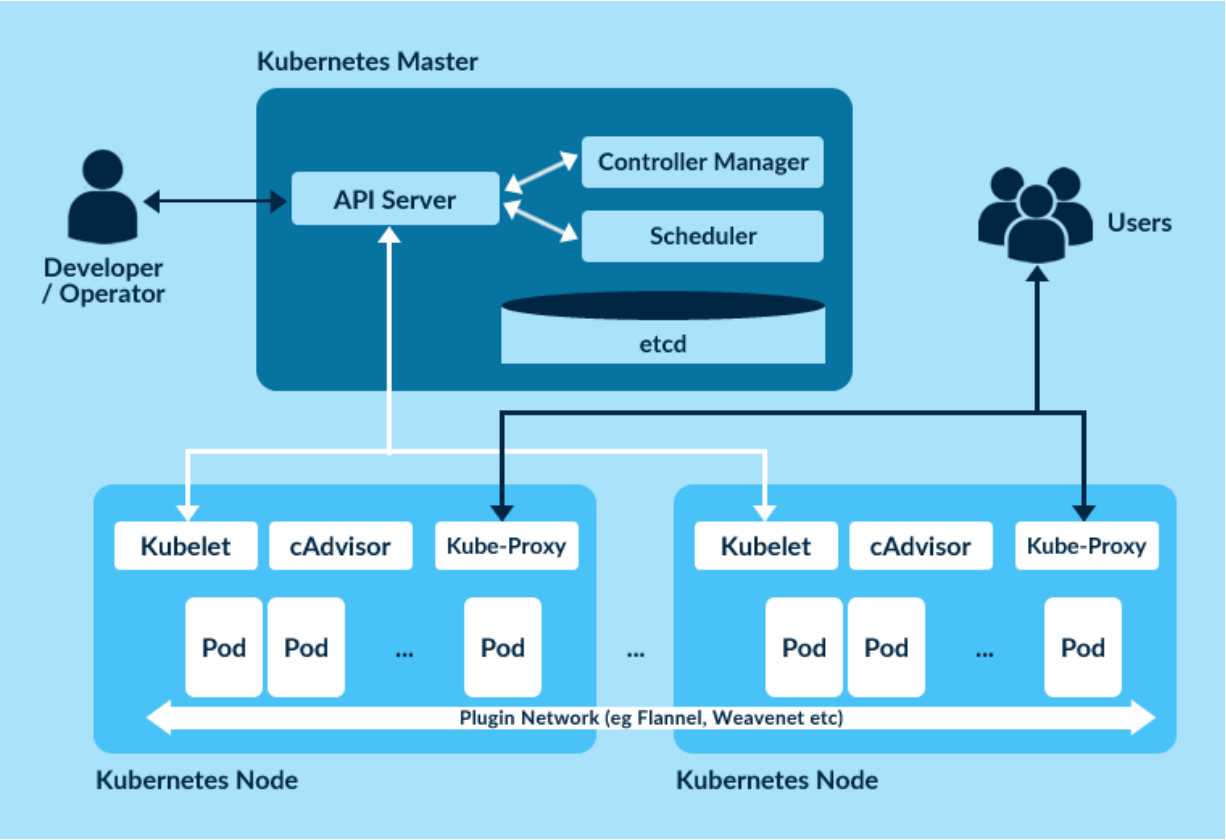
$ kubectl get deployments
$ kubectl get services

So now the cluster is set up and we can create deployments for the running in the cluster.

We can expose these deployments to endpoints via services and to external traffic by creating an ingress network.

5.8 Kubernetes API Server

Gateway to the Kubernetes cluster is Api server. It is a centralized system that is accessed by all users, automation, and components in the Kubernetes cluster. The API server implements a RESTful API over HTTP, performs all API operations, and is responsible for storing API objects into a persistent storage backend. This chapter covers the details of this operation.



5.8.1 Pieces of API Server

Kubernetes API server has three core functions:

- **API management:**

  In API management process, APIs are exposed and managed by the server.

- **Request processing:**

  Request processing processes individual API requests from a client.

- **Internal control loops:**

  Internal control loops has internals responsibilities for background operations necessary to the successful operation of the API server.

5.8.2 API Management

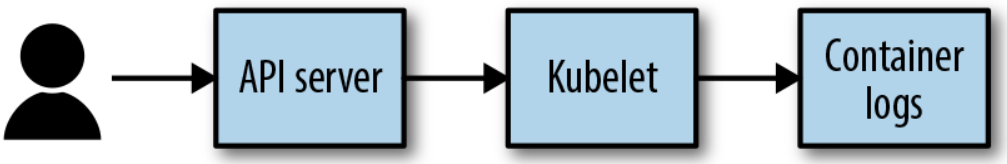
The API server is an HTTP serverthus, every API request is an HTTP request. But the characteristics of those HTTP requests must be described so that the client and server know how to communicate. For the purposes of exploration, its great to have an API server actually up and running so that you can poke at it. You can either use an existing Kubernetes cluster that you have access to, or you can use the minikube tool for a local Kubernetes cluster. To make it easy to use the curl tool to explore the API server, run the kubectl tool in proxy mode to expose an unauthenticated API server on localhost:8001 using the following command:

$ kubectl proxy

5.9 Request Management

The main aim of Api server is to receive and process API calls in form of HTTP request. Type of request performed by Api server are following:

- Get
- List
- Post
- Delete



5.10 ETCD

A key-value storage for kubernetes backing store for all cluster data.

5.11 Kube-Scheduler

If there is a newly created pod which is not allocated any node then Kube-Scheduler select a node for them to run.

5.12 Kube-Control-Manager

Kube-Control-Manager is a component on master that runs controllers. Each controller is a separate process but they are all merged.

These controllers include:

- **Node controller:** Responsible for noticing and responding when nodes go down.
- **Replication controller:** Responsible for maintaining the correct number of pods for every replication controller object in the system.

- **Endpoints controller:** Populates the Endpoints object (that is, joins Services & Pods).
- **Service Account & Token Controllers:** Create default accounts and API access tokens for new namespaces.

5.13 Converting docker-compose file to its Kubernetes equivalent

Write text here

5.13.1 Kompose

Write text here

5.13.2 Compose on Kubernetes

Example of bulleted list

- write point 1 here [1][**?**]
- write point 2 here
- write point 3 here

Example of Numbered list

1. Write point 1 here
2. Write point 2 here
3. write point 3 here

6 Tutor: An alternative

Write text here

7 Deployment

8 Another Section name

Write text here. Example of table.

No.	Company	Operating System
1	Nokia	Symbian (S60, S40)
2	Microsoft	Windows
3	Apple Inc.	IOS
4	Blackberry	Blackberry

Table 1: Mobile Operating Systems

8.1 Subsection

Write text here

9 Fifth Section name

Write text here

9.1 Subsection

Write text here

9.1.1 Sub-Sub-section

Write text here

References

[1] "IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830-1998," *IEEE Computer Society*, October 1998.