

Configurable and Scalable IITBombayX MOOC platform on commodity servers

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

Guided by Mr.Nagesh Karmali

Last Updated: July 4, 2019

List of Figures

1	TLS handshake timeout error	6
2	Kubernetes pods [1]	12
3	Kubernetes deployment [2]	12
4	Kubernetes service [3]	13
5	Ingress network [4]	13
6	Flannel network [5]	14
7	Calico network [6]	15
8	Weavenet [7]	16
9	Kubernetes architecture [8]	20
10	Request management [9]	21
11	Kompose [10]	22
12	Persistent volumes [11]	23
13	Nodes connected in a Kuberentes cluster	24
14	Persistent Volumes	25
15	Persistent Volume Claims	26
16	Kubernetes deployments	26
17	Kubernetes deployments after removing gradebook and forum	27
18	Pods running before deleting a node	27
19	Pods running after deleting the node	28
20	SAR CPU Metrics	32
21	SAR Memory Metrics	32
22	A look at the top command, 15 minutes into the test. Guicorn, which is a Python Web Server Gateway Interface HTTP Server, takes up most of the CPU resources.	33
23	A look at the top command, 30 minutes into the test. Guicorn, which is a Python Web Server Gateway Interface HTTP Server, takes up most of the CPU resources.	34
24	A look at the top command, 45 minutes into the test. Guicorn, which is a Python Web Server Gateway Interface HTTP Server, takes up most of the CPU resources.	35
25	A look at the top command, 60 minutes into the test. Guicorn, which is a Python Web Server Gateway Interface HTTP Server, takes up most of the CPU resources.	36
26	SAR CPU Metrics	38
27	SAR Memory Metrics	38
28	A look at the top command, 15 minutes into the test. Python takes up most of the CPU Resources.	39
29	A look at the top command, 30 minutes into the test. Python takes up most of the CPU Resources.	40
30	A look at the top command, 45 minutes into the test. Python takes up most of the CPU Resources.	41
31	A look at the top command, 60 minutes into the test. Python takes up most of the CPU Resources.	42
32	JMeter Summary Report for Open edX Native.	43
33	JMeter Summary Report for Open edX Devstack.	43

Contents

1	Introduction	1
2	Open edX	1
2.1	Releases	1
2.2	Installation	1
2.2.1	Native installation	2
2.2.2	Docker-based installation	2
3	Docker [12]	5
3.1	About Docker	5
3.2	Terms related to Docker	5
3.2.1	Containerization	5
3.2.2	Docker Image	5
3.2.3	Docker Container	6
3.2.4	Docker Registry	6
3.2.5	Docker Engine	6
3.2.6	Docker Compose	7
3.2.7	Dockerfiles	7
3.3	Docker-Machine	7
3.3.1	Install Docker Machine:	7
3.3.2	Use Docker Machine to run Docker Containers:	7
3.3.3	Create a Machine	7
3.3.4	Connect your shell to the new machine	7
3.3.5	Run containers and experiment with machine commands	8
3.4	Docker Swarm	8
3.4.1	Types of networks in docker	8
3.4.2	Initialising a Swarm Node as Manager	9
3.4.3	Docker Stack	9
4	Ansible	10
4.1	Installing openssh	10
4.2	Generating key at server and copying in client	10
4.3	Ansible setup	10
4.3.1	Adding repository	10
4.3.2	Installing ansible	10
4.3.3	Checking version	10
4.3.4	Ansible directories	10
4.3.5	Making a copy a directory for our use	10
4.4	Some useful commands for ansible	11
4.4.1	To see the hostnames of all nodes	11
4.4.2	To see the volumes used by all nodes	11
4.4.3	To run an ansible-playbook	11
5	Kubernetes[13]	11
5.1	What is Kubernetes	11
5.2	Main Features Of Kubernetes	11
5.3	Terminologies in Kubernetes	11

5.3.1	Pod	11
5.3.2	Deployment	12
5.3.3	Service	13
5.3.4	Ingress network	13
5.4	Networking in Kubernetes	14
5.4.1	Flannel	14
5.4.2	Calico	15
5.4.3	Weavenet	15
5.5	Prerequisites for running kubernetes [14]	16
5.5.1	Disable Swap Space	16
5.5.2	Make IP static	16
5.6	Kubernetes setup	16
5.7	Setting up your Kubernetes cluster	17
5.7.1	Setting up a pod network	18
5.7.2	Setting up and hosting the dashboard [15]	18
5.7.3	Kubernetes commands for getting resource info	19
5.8	Kubernetes API Server	19
5.8.1	Pieces of API Server	19
5.8.2	API Management [16]	20
5.9	Request Management	20
5.10	ETCD	20
5.11	Kube-Scheduler	21
5.12	Kube-Control-Manager	21
5.13	Converting docker-compose file to its Kubernetes equivalent	21
5.13.1	Kompose	21
5.13.2	Compose on Kubernetes	23
6	Our progress	23
6.1	Kubernetes cluster	23
6.2	Kompose	24
6.3	Persistent volumes and Persistent volume claims	24
6.4	Kubernetes deployment	25
6.5	Fault handling	25
6.6	Provisioning	26
6.7	System Configurations	31
6.8	System Configurations	37

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 the container orchestrator and Docker as the container runtime environment.

Technologies used:

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

2 Open edX

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

2.1 Releases

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

- Ironwood
- Hawthorn
- Ginkgo

We have used the Ironwood.1 release in this project.

2.2 Installation

Open edX can be installed in two ways:

- Native installation
- Docker-based installation

2.2.1 Native installation

i) Prerequisites

- Ubuntu 16.04 amd64

ii) Set-up configurations

Ubuntu package sources need to be updated:

```
$ sudo apt-get update -y
$ sudo apt-get upgrade -y
$ sudo reboot
```

iii) Installation procedure

1. Set the OPENEDX_RELEASE variable to the Git Tag corresponding to the Ironwood release of Open edX:

```
export OPENEDX_RELEASE=open-release/ironwood.1
```

2. Next, create a config.yml file, which states the hostname of the Learning Management System (LMS) and the Content Management System (CMS):

```
EDXAPP_LMS_BASE: 127.0.0.1
EDXAPP_CMS_BASE: 127.0.0.1:8080
```

(works only when the LMS hostname and the CMS hostnames are the same, or the CMS hostname is a subdomain of the LMS hostname).

3. Now bootstrap the Ansible installation.
4. Retrieve the ansible-bootstrap.sh bash script from the Github server and execute it:

```
wget https://raw.githubusercontent.com/edx/configuration/
$OPENEDX_RELEASE/util/install/ansible-bootstrap.sh -O - | sudo bash
```

5. Use the generate-password.sh file from the Github server to randomize password:

```
wget https://raw.githubusercontent.com/edx/configuration/
$OPENEDX_RELEASE/util/install/generate-passwords.sh -O - | bash
```

6. Finally, use the native.sh script from the Github server to install Open edX:

```
wget https://raw.githubusercontent.com/edx/configuration/
$OPENEDX_RELEASE/util/install/native.sh -O - | bash
```

2.2.2 Docker-based installation

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

The docker based installation allows the resources of our system to be utilized in an

optimal way. Docker is based on the principle of containerization. For Open edX Devstack to work properly, we would need the Docker engine, Docker Machine and Docker Compose tools.

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 if Docker has been installed properly, one can run the following

command:

```
$ sudo docker run hello-world
```

For installing docker machine, run the following commands:

```
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, run the following commands:

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 if the tools mentioned above have been installed properly, you can use the `-version` flag with the corresponding command to check the installed version.

ii) Installing Ironwood Devstack:

The three most recent versions of the Open edX Devstack are Ginkgo, Hawthorn and Ironwood.

The following commands perform the Ironwood-release installation of 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 the command: `docker ps`

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

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

iii) Errors:

1. `./provision.sh: line 21: /usr/local/bin/docker-compose: Permission denied Makefile:59: recipe for target 'dev.provision.run' failed make: *** [dev.provision.run] Error 126`

Solution :

```
$ sudo -i
$ curl -L https://github.com/docker/compose/releases/download/1.18.0/docker-compose-
'uname -s'-uname -m' -o /usr/local/bin/docker-compose
$ chmod 755 /usr/local/bin/docker-compose
$ exit
```

2. `TASK [common : Update expired apt keys] ***** fatal: [127.0.0.1]: FAILED! =_ "changed": true, "cmd": "apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 69464050", "delta": "0:02:00.744139", "end": "2019-05-24 07:49:07.605347", "failed": true, "rc": 2, "start": "2019-05-24 07:47:06.861208", "stderr": "gpg: requesting key 69464050 from hkp server keyserver.ubuntu.com\nngpg: keyserver timed out\nngpg: keyserver receive failed: keyserver error", "stdout": "Executing:`

```
/tmp/tmp.NmWlivV5Jo/gpg.1.sh --recv-keys\n--keyserver\nkeyserver.ubuntu.com\n69464050",  
"stdout_lines": ["Executing: /tmp/tmp.NmWlivV5Jo/gpg.1.sh --recv-keys", "--  
keyserver", "keyserver.ubuntu.com", "69464050"], "warnings": [] to retry, use: -  
-limit @/edx/app/edx_angular/edx_angular/playbooks/demo.retry
```

Solution :

This error occurs due to the fetching to apt-keys from an expired link.
To resolve follow the following steps :

- (a) `sudo exec -it edx.devstack.lms bash`
- (b) `cd app/edx_angular/edx_angular/playbooks/roles/common_vars/defaults/`
- (c) `vi main.yml`
- (d) On entering the main.yml file in vim editor change the link for the keyword `COMMON_EDX_PPA_KEY` from `keyserver.ubuntu.com` to `hkp://keyserver.ubuntu.com:80`
- (e) `exit`

3 Docker [12]

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. Using application running as a docker container removes the system dependency of the application. No changes has to be made in applications to run them in different environments or machines. Docker based application provides liberty to the developer, tester and deployer to run the same application which any changes specific to their environment.

3.2 Terms related to Docker

3.2.1 Containerization

Containerization is a lightweight alternative to full machine virtualization that involves encapsulating an application within a container with its own operating environment. Essentially, containers share the kernel space with the host OS, but have a separate user space. This makes them lightweight.

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.

TLS handshake timeout error in pulling Docker image :

In case there is a TLS handshake timeout error while pulling a Docker image from an online repository, try running the command again and again and eventually the image will be pulled:

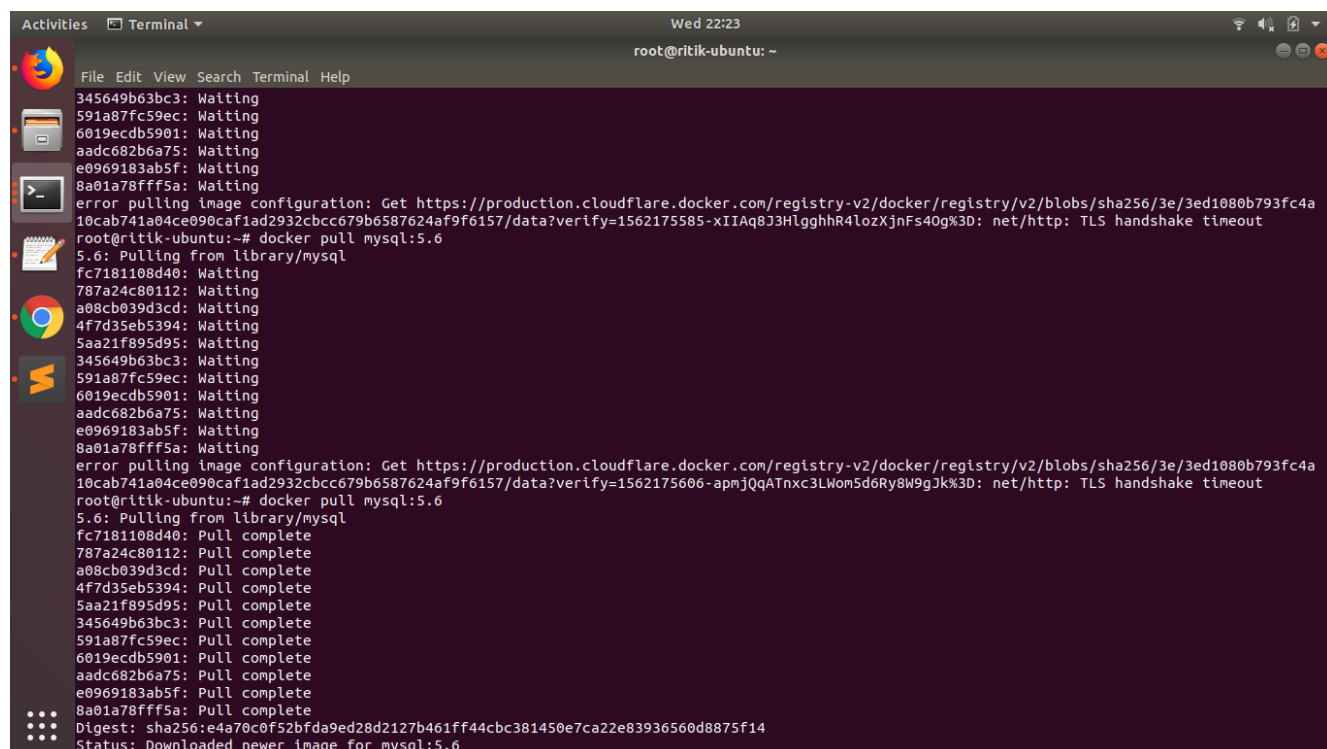


Figure 1: TLS handshake timeout error

3.2.3 Docker Container

A *Docker Container* encapsulates a Docker image and when live and running, is considered a container. Each container runs in an isolated environment on 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 YAML configuration 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

Docker Machine can be used to :

- Manage and provision multiple remote Docker hosts.
- Provision Swarm clusters.

Docker Machine is a tool that lets you install Docker Engine on virtual hosts, and manage hosts with docker-machine commands. Using docker-machine command you can start, inspect, stop and restart a managed host, upgrade the Docker client and daemon, and configure a Docker client to talk to your host.

3.3.1 Install Docker Machine:

- For Linux systems, execute the following command:

```
$ base=https://github.com/docker/machine/releases/download/v0.16 &&  
curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/usr/local/bin/docker-  
machine && chmod +x /usr/local/bin/docker-machine
```

- Check the installation by displaying the Machine version:

```
$ docker-machine version
```

3.3.2 Use Docker Machine to run Docker Containers:

- Create a new docker virtual machine.
- Switch your environment to your new VM.
- Use the docker client to create, load and manage containers.

3.3.3 Create a Machine

```
$ docker-machine create --driver virtualbox default
```

3.3.4 Connect your shell to the new machine

```
$eval $(docker-machine env default)
```

3.3.5 Run containers and experiment with machine commands

- Use **docker-run** to download and run images
- Run **docker-machine ip default** to get the host IP address.
- Run **docker-machine stop default** to stop the host default.
- Run **docker-machine start default** to start the host default.
- Run **docker-machine env -u** to see which Docker variables we need to unset to switch to the current shell.

3.4 Docker Swarm

Docker swarm is used to create a cluster of nodes. Multi-container, multi-machine applications are made possible by joining multiple machines into a Dockerized cluster called a swarm. Docker swarm is a group of machines running docker and are connected to each other by means of an overlay network. There can be one or multiple nodes that can manage the swarm, these nodes are called manager nodes. The rest of the nodes only provide capacity to run the containers and are called worker nodes.

3.4.1 Types of networks in docker

- **Bridge Network** : This is the type of network that allows the docker containers to communicate within a single host system. This type of network is not feasible to communicate between multiple host systems.

A bridge network can be created by :

```
$ docker network create --driver bridge mynetwork
```

User-defined bridge networks are best when you need multiple containers to communicate on the same Docker host.

- **Overlay Network** : An overlay network is used for multi host network communication. It connects multiple docker daemons together and enables swarm services to communicate with each other. A docker swarm must always be connected via an overlay network.

An overlay network can be created by :

```
$ docker network create --driver overlay mynetwork
```

Overlay networks are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.

- **Macvlan Network** : Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses.

A Macvlan Network can be created by :

```
$ docker network create -d macvlan \  
--subnet=192.168.40.0/24 \  
--gateway=192.168.40.1 \  
-o parent=eth0 my-macvlan-net
```

Macvlan networks are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.

- **Host Network :** For standalone containers, remove network isolation between the container and the Docker host, and use the hosts networking directly.

We can use the hosts system network to at a particular port to allow the container to share data. Host networks are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.

3.4.2 Initialising a Swarm Node as Manager

In order to initialize a swarm node a manager we need to find the IP address of the node using the command **ifconfig**. Then run the command:

```
$ docker swarm init --advertise-addr IP_OF_THE_MANAGER
```

The join token for another node as a manager can be found by :

```
$ docker swarm join-token manager
```

The join token of another node as a worker can be found by :

```
$ docker swarm join-token worker
```

3.4.3 Docker Stack

For multiple services that are dependent on each other to communicate with each other on a swarm network, we may use the Docker Stack command. Docker stack deployment is used to deploy a list of interdependent services on a swarm network.

To deploy a stack we create a docker-compose file with an additional feature in each service as deploy.

```
$ docker stack deploy -c docker-compose.yml myapp
```

The -c flag is used to execute the docker compose file and create multiple interdependent services which can communicate with each other via the swarm network.

An example of docker-compose file for stack deployment: example-voting-app.yml

4 Ansible

Installing docker, kubernetes, and all the other components in each node in a multi-node cluster, which contains many nodes, can be a tedious task. To overcome this issue, Ansible can be used for configuration management. Furthermore, Ansible can be used to configure the nodes too. By default Ansible uses a native openSSH connection and runs commands simultaneously on multiple nodes. We have created Ansible Playbooks for all the needed configurations.

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 to avoid the need 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[13]

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 is assigned an IP address.

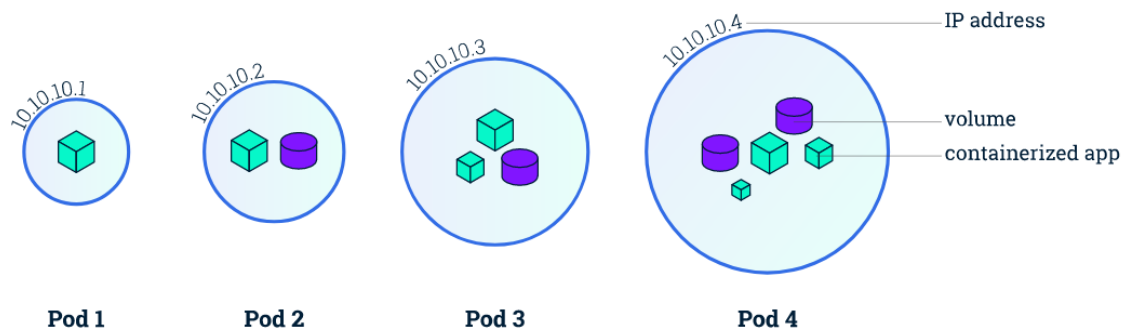


Figure 2: Kubernetes pods [1]

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.

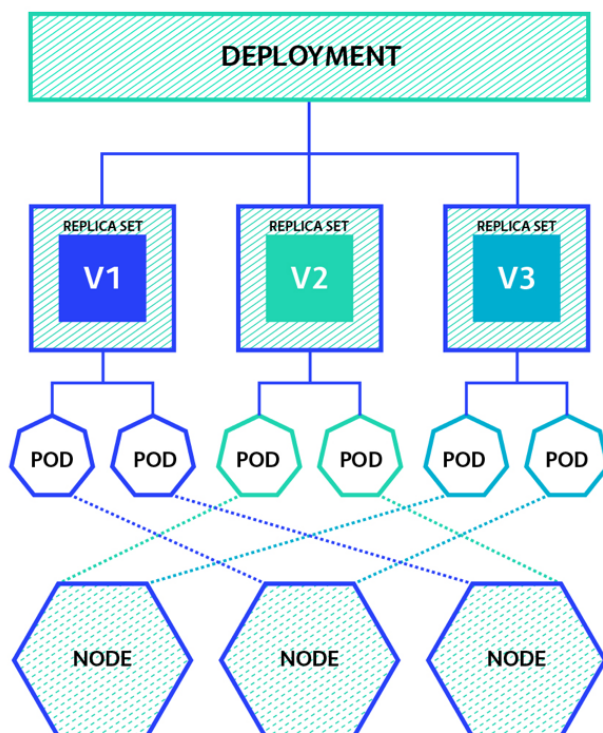


Figure 3: Kubernetes deployment [2]

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.

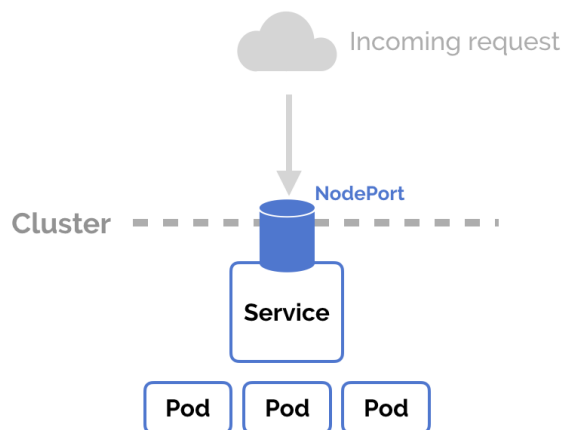


Figure 4: Kubernetes service [3]

5.3.4 Ingress network

An ingress network exposes the services in the cluster to the outside network i.e. to the clients. An ingress helps us in handling the outside traffic and routing it based on the context.

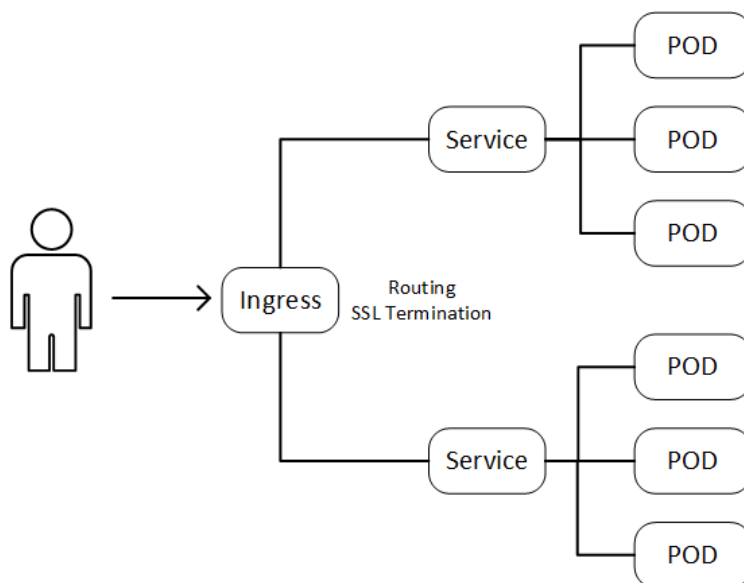


Figure 5: Ingress network [4]

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 primarily broken down into 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.

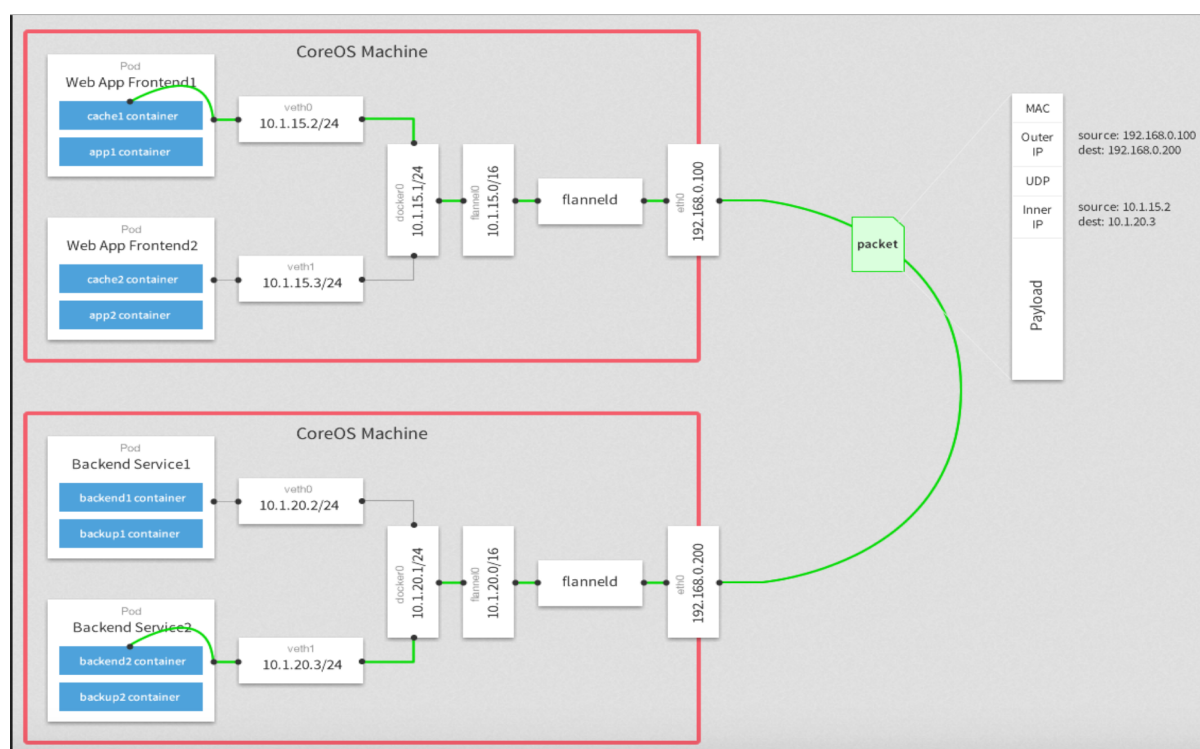


Figure 6: Flannel network [5]

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.

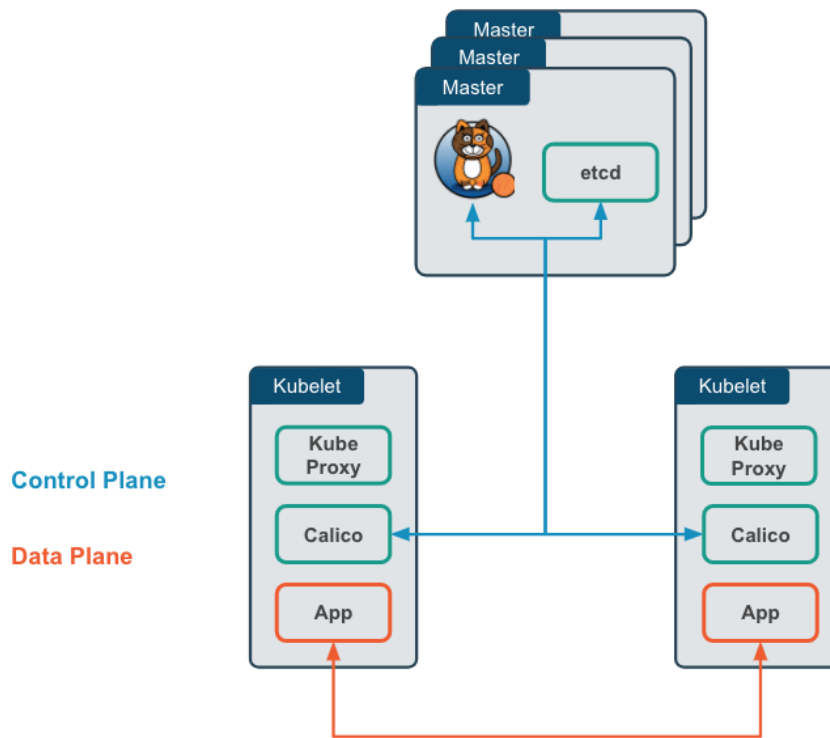


Figure 7: Calico network [6]

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.

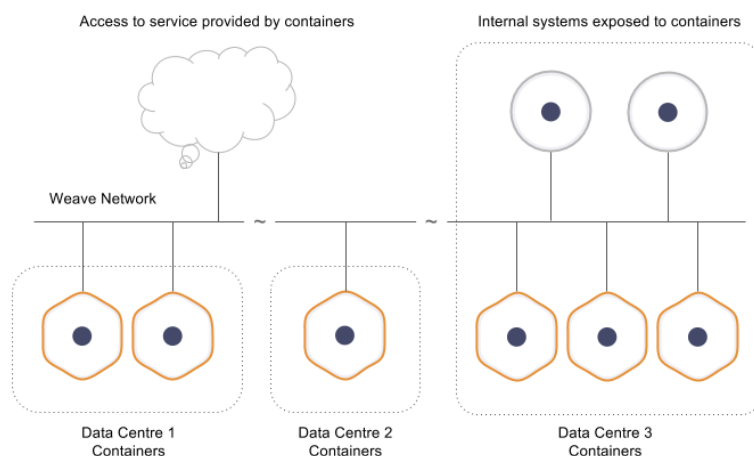


Figure 8: Weavenet [7]

In addition to these, kubernetes supports many other plugins too. A list of them can be found in the official documentation page of kubernetes (Networking in Kubernetes).

5.5 Prerequisites for running kubernetes [14]

5.5.1 Disable Swap Space

In order to set up a kubernetes cluster, all nodes including the 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 run automatically on startup:
\$ 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
```

Once the cluster has been successfully initialised, 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

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. Flannel is a overlay network whereas calico is an L3 network. The advantages in detail can be read from this medium Blog.

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
```

5.7.2 Setting up and hosting the dashboard [15]

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
```

Once the cluster is set up, 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

The Gateway to the Kubernetes cluster is the Kubernetes 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.

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 internal responsibilities for background operations necessary to the successful operation of the API server.

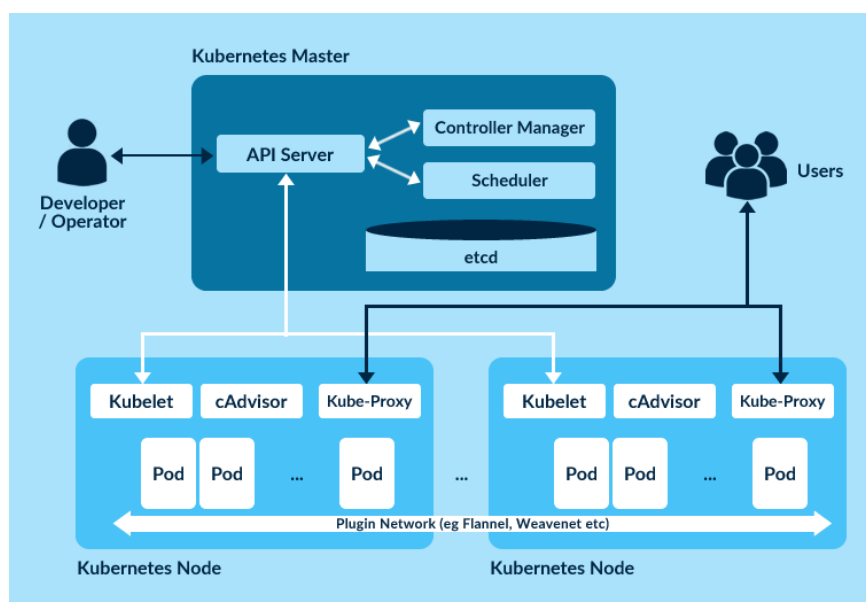


Figure 9: Kubernetes architecture [8]

5.8.2 API Management [16]

The API server is an HTTP server thus, 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, it's 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 the API server is to receive and process API calls in the form of HTTP requests.

Type of request performed by the API server are as follows:

- GET
- LIST
- POST
- DELETE

5.10 ETCD

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

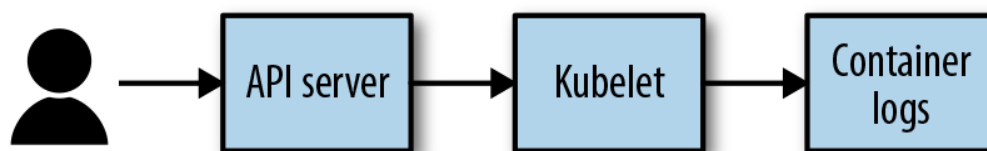


Figure 10: Request management [9]

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

As we have docker-compose in docker we similarly have a kompose file with kubernetes. We can deploy a docker-compose created stack on a docker-swarm cluster.

5.13.1 Kompose

Kompose is tool provided by kubernetes which converts a docker-compose file into kompose files for providers like kubernetes and openshift. For each service in a compose file we have a deployment, service and persistent volume claim file in kompose. We can also create our persistent volume file if we require a persistent volume for our services.

Installing Kompose on Linux :

1. Download the latest release and install in the system:

```
$ curl -L https://github.com/kubernetes/kompose/releases/download/v1.17.0/kompose-linux-amd64 -o kompose
```

2. Give executable permission and move the directory:

```
$ chmod +x kompose
$ sudo mv ./kompose /usr/local/bin/kompose
```

In order to convert the compose file into kompose run the command :

```
$ kompose convert -f docker-compose.yml
```

In order to run yaml file in kubernetes we use the command :

```
$ kubectl apply -f FILE_NAME.yaml
```

Kompose provides an easy solution for converting a docker-compose file into kompose. Docker-compose is an easy to write file whereas kompose files are a bit lengthy and it gets a bit tedious to code. So kompose provides an easy alternative. The files provided by kompose does not have a 100% conversion rate but we can change files as per our requirements.

We have converted the docker-compose.yml and docker-compose-host.yml file in devstack to kompose file. We have created a PersistentVolume.yml for each service which creates a pool of space from where a persistentvolumeclaim can claim a volume for the pod. The kompose converts the file in kompose objects and then into kubernetes objects and then into yaml or json output as per our requirements.

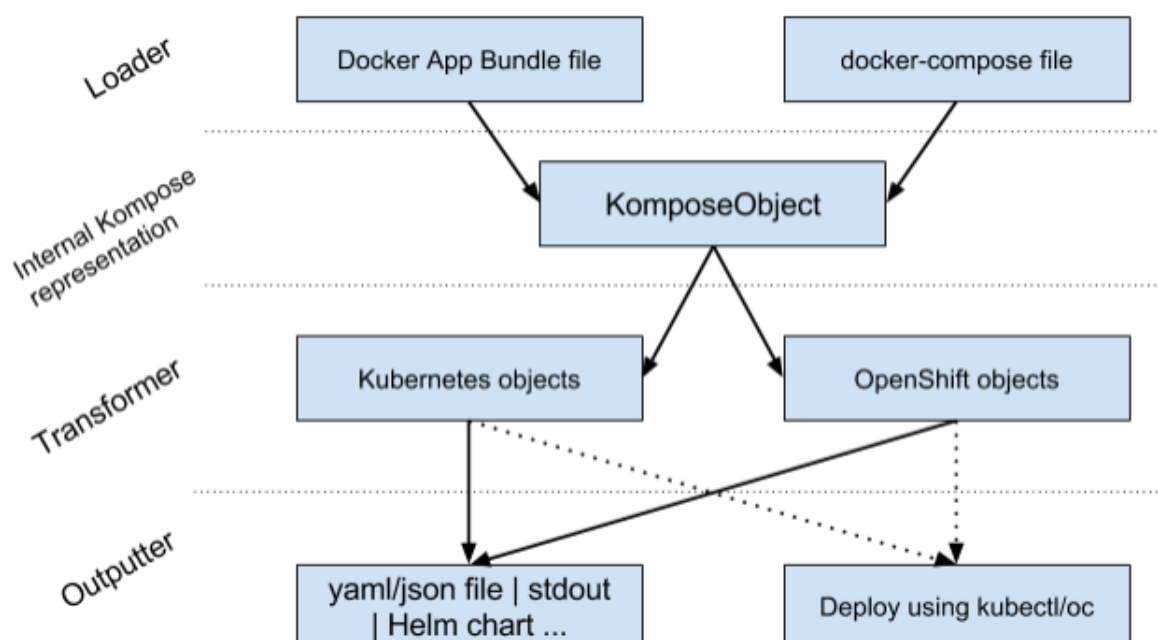


Figure 11: Kompose [10]

The persistent volumes present are created by admin so that a developer can attach a claim and use the volume for the pod.

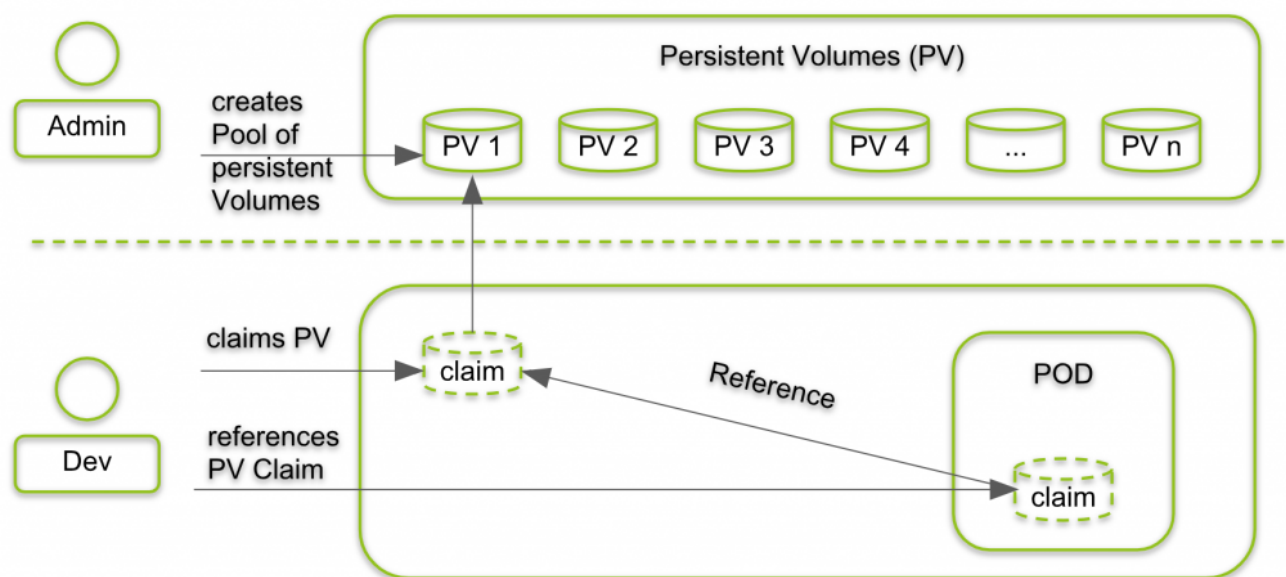


Figure 12: Persistent volumes [11]

5.13.2 Compose on Kubernetes

We also have a tool `compose-on-kubernetes` provided by docker which converts a docker-compose file into compose file. It is an open source tool and can be explored to check its accuracy and efficiency. (<https://github.com/docker/compose-on-kubernetes.git>)

6 Our progress

In this project, our aim was to deploy an instance of Open edX on a Kubernetes cluster.

6.1 Kubernetes cluster

At first, we set up a Kubernetes cluster as mentioned above with one master and three nodes.

Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age
node1000	beta.kubernetes.io/arch: armv7l beta.kubernetes.io/os: linux kubernetes.io/arch: armv7l kubernetes.io/hostname: node1000 kubernetes.io/os: linux	True	0.25 (8.33%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	11 minutes
kmaster	beta.kubernetes.io/arch: armv7l beta.kubernetes.io/os: linux kubernetes.io/arch: armv7l kubernetes.io/hostname: kmaster kubernetes.io/os: linux	True	0.25 (12.50%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	14 minutes
epsy	beta.kubernetes.io/arch: armv7l beta.kubernetes.io/os: linux kubernetes.io/arch: armv7l kubernetes.io/hostname: epsy kubernetes.io/os: linux	True	0.25 (6.25%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	30 minutes
ritik-ubuntu	beta.kubernetes.io/arch: armv7l beta.kubernetes.io/os: linux kubernetes.io/arch: armv7l kubernetes.io/hostname: ritik-ubuntu kubernetes.io/os: linux	True	1 (25.00%)	0 (0.00%)	140 Mi (1.77%)	340 Mi (4.31%)	50 minutes

Figure 13: Nodes connected in a Kuberentes cluster

6.2 Kompose

We converted the docker-compose.yaml file for Open edX containers to multiple Kubernetes deployment files (YAML). But, since the conversion rate was not 100%, we had to create persistent volumes and claims for those volumes on our own.

The converted files can be found in the github repository :

https://github.com/fresearchgroup/Configurable-and-Scalable-IITBombayX-MOOC-platform-on-Commodity-Servers/tree/master/kompose_files [17]

6.3 Persistent volumes and Persistent volume claims

We made YAML files to create persistent volumes and then attached each volume to a claim using its corresponding persistent-volume-claim.yaml file.

The persistent volume and claim files can be found in the github repository :

https://github.com/fresearchgroup/Configurable-and-Scalable-IITBombayX-MOOC-platform-on-Commodity-Servers/tree/master/host_volumes [18].

You need not create all the persistent volume and claim files manually, we have created a bash script **volumes.sh** in the same directory that does the job.

Name	Capacity	Access Modes	Reclaim Policy	Status	Claim	Storage Class	Reason	Age
studio-volume2	300Mi	ReadWriteMany	Delete	Bound	default/studio-claim	manual	-	16 minutes
studio-volume0	300Mi	ReadWriteMany	Delete	Bound	default/studio-claim	manual	-	16 minutes
edxapp-studio-asset	300Mi	ReadWriteMany	Delete	Bound	default/edxapp-stud	manual	-	16 minutes
lms-volume2	300Mi	ReadWriteMany	Delete	Bound	default/lms-claim2	manual	-	16 minutes
lms-volume0	300Mi	ReadWriteMany	Delete	Bound	default/lms-claim0	manual	-	16 minutes
edxapp-lms-assets-v	300Mi	ReadWriteMany	Delete	Bound	default/edxapp-lms-	manual	-	16 minutes
gradebook-node-volu	300Mi	ReadWriteMany	Delete	Bound	default/gradebook-n	manual	-	16 minutes
gradebook-volume0	300Mi	ReadWriteMany	Delete	Bound	default/gradebook-c	manual	-	16 minutes
forum-volume0	300Mi	ReadWriteMany	Delete	Bound	default/forum-claim	manual	-	16 minutes
edx-notes-api-volume	300Mi	ReadWriteMany	Delete	Bound	default/edx-notes-aj	manual	-	16 minutes
edx-notes-api-volume	300Mi	ReadWriteMany	Delete	Bound	default/edx-notes-aj	manual	-	16 minutes
edxapp-node-volume	300Mi	ReadWriteMany	Delete	Bound	default/edxapp-node	manual	-	16 minutes
devpi-volume	300Mi	ReadWriteMany	Delete	Bound	default/devpi-data	manual	-	16 minutes
ecommerce-node-vo	300Mi	ReadWriteMany	Delete	Bound	default/ecommerce	manual	-	16 minutes
ecommerce-volume2	300Mi	ReadWriteMany	Delete	Bound	default/ecommerce	manual	-	16 minutes

Figure 14: Persistent Volumes

6.4 Kubernetes deployment

We then deployed all the 15 containers on the cluster. Kubernetes assigned the appropriate node (one among the two) for each container to run. The gradebook and forum deployments were unsuccessful. But, those two are not necessary for running the bare minimum Open edX instance and hence can be removed:

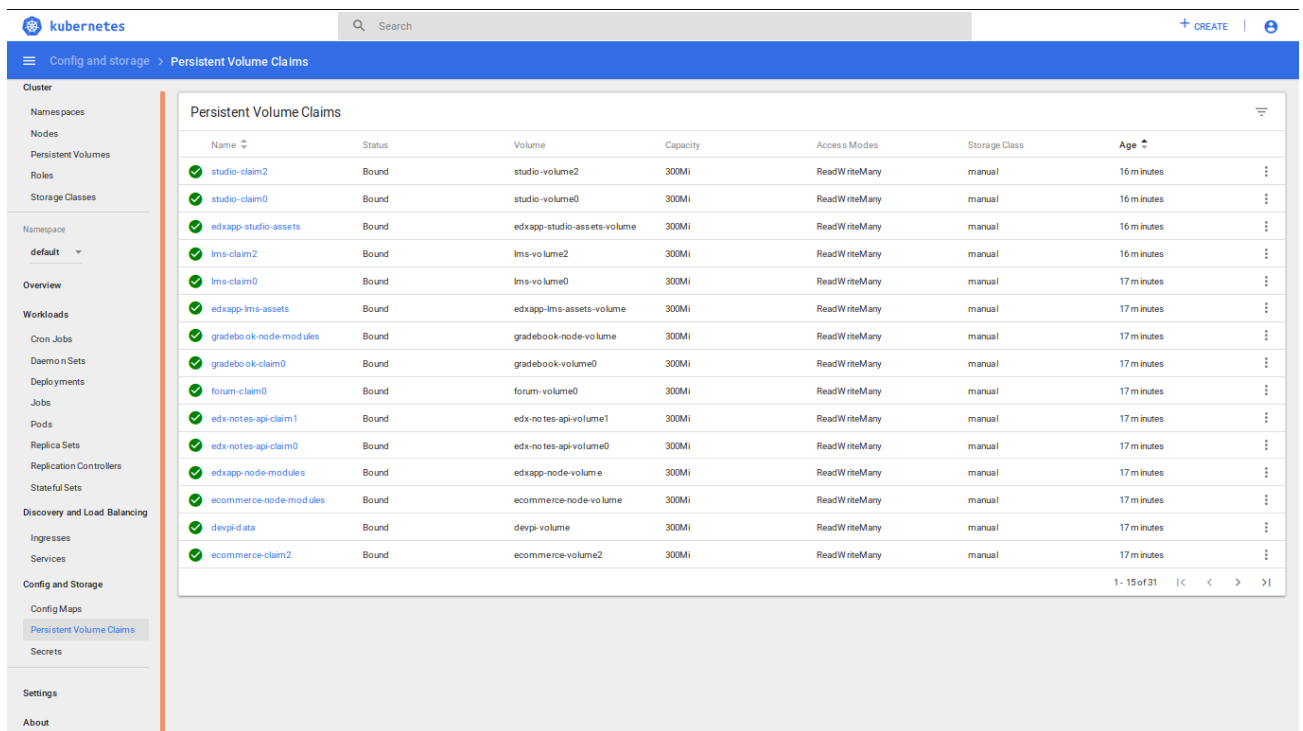
6.5 Fault handling

We removed one node from the cluster and found out that Kubernetes deployed all the containers running on that node to the node that was still a part of the cluster.

The pods running in the cluster:

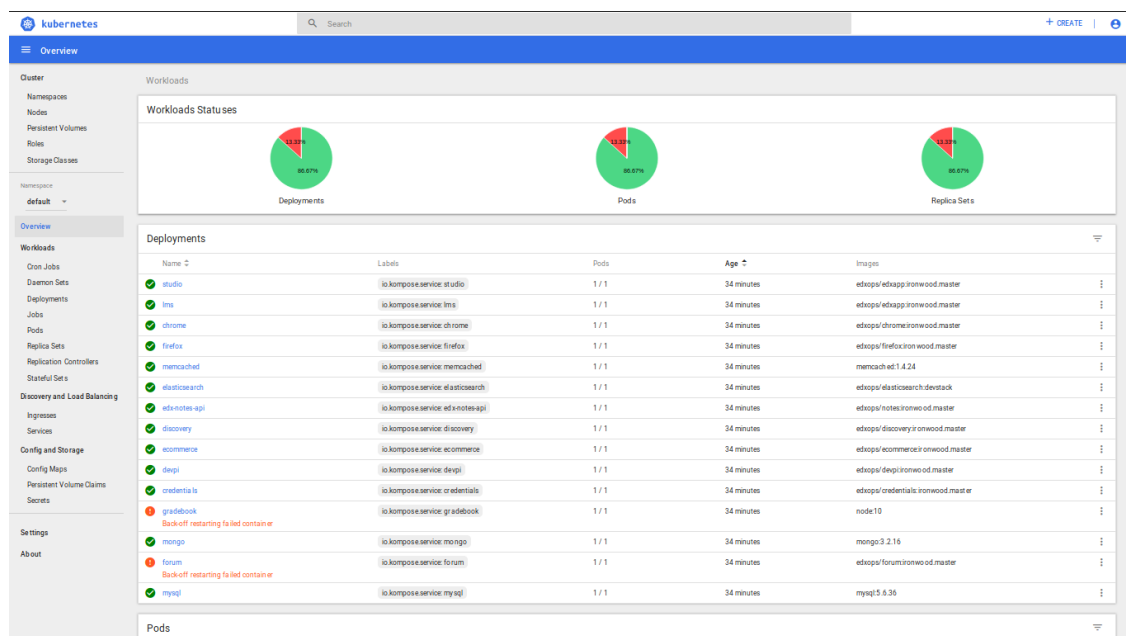
- Before deleting the node epsy:
- After deleting the node epsy:

As we can see, the pods which were earlier running on the node epsy, now run on the remaining nodes after, their host node is deleted. This rescheduling of the pods is done automatically by Kubernetes once a node goes down. The reshuffling takes very less time as is clear from the image (15-16 seconds in this case).



Name	Status	Volume	Capacity	Access Modes	Storage Class	Age
studio-claim2	Bound	studio-volume2	300Mi	ReadWriteMany	manual	16 minutes
studio-claim0	Bound	studio-volume0	300Mi	ReadWriteMany	manual	16 minutes
edkapp-studio-assets	Bound	edkapp-studio-assets-volume	300Mi	ReadWriteMany	manual	16 minutes
lms-claim2	Bound	lms-volume2	300Mi	ReadWriteMany	manual	16 minutes
lms-claim0	Bound	lms-volume0	300Mi	ReadWriteMany	manual	17 minutes
edkapp-lms-assets	Bound	edkapp-lms-assets-volume	300Mi	ReadWriteMany	manual	17 minutes
gradebook-node-modules	Bound	gradebook-node-volume	300Mi	ReadWriteMany	manual	17 minutes
gradebook-claim0	Bound	gradebook-volume0	300Mi	ReadWriteMany	manual	17 minutes
forum-claim0	Bound	forum-volume0	300Mi	ReadWriteMany	manual	17 minutes
edx-notes-api-claim1	Bound	edx-notes-api-volume1	300Mi	ReadWriteMany	manual	17 minutes
edx-notes-api-claim0	Bound	edx-notes-api-volume0	300Mi	ReadWriteMany	manual	17 minutes
edkapp-node-modules	Bound	edkapp-node-volume	300Mi	ReadWriteMany	manual	17 minutes
ecommerce-node-modules	Bound	ecommerce-node-volume	300Mi	ReadWriteMany	manual	17 minutes
devpi-data	Bound	devpi-volume	300Mi	ReadWriteMany	manual	17 minutes
ecommerce-claim2	Bound	ecommerce-volume2	300Mi	ReadWriteMany	manual	17 minutes

Figure 15: Persistent Volume Claims



Name	Labels	Pods	Age	Images
studio	io.kompose.service:studio	1/1	34 minutes	edkops/edkapp:ironwood.master
lms	io.kompose.service:lms	1/1	34 minutes	edkops/edkapp:ironwood.master
chrome	io.kompose.service:chrome	1/1	34 minutes	edkops/chrome:ironwood.master
firefox	io.kompose.service:firefox	1/1	34 minutes	edkops/firefox:ironwood.master
memcached	io.kompose.service:memcached	1/1	34 minutes	memcached:1.4.24
elasticsearch	io.kompose.service:elasticsearch	1/1	34 minutes	edkops/elasticsearch:devstack
edxnotes-api	io.kompose.service:edxnotes-api	1/1	34 minutes	edkops/notes:ironwood.master
discovery	io.kompose.service:discovery	1/1	34 minutes	edkops/discovery:ironwood.master
ecommerce	io.kompose.service:ecommerce	1/1	34 minutes	edkops/ecommerce:ironwood.master
devpi	io.kompose.service:devpi	1/1	34 minutes	edkops/devpi:ironwood.master
credentials	io.kompose.service:credentials	1/1	34 minutes	edkops/credentials:ironwood.master
gradebook	io.kompose.service:gradebook	1/1	34 minutes	node:10
mongo	io.kompose.service:mongo	1/1	34 minutes	mongo:3.2.16
forum	io.kompose.service:forum	1/1	34 minutes	edkops/forum:ironwood.master
mysql	io.kompose.service:mysql	1/1	34 minutes	mysql:5.6.36

Figure 16: Kubernetes deployments

6.6 Provisioning

The file provision.sh in the devstack directory has the commands and links to the files that are to be executed for provisioning and migration of databases. Those are simply

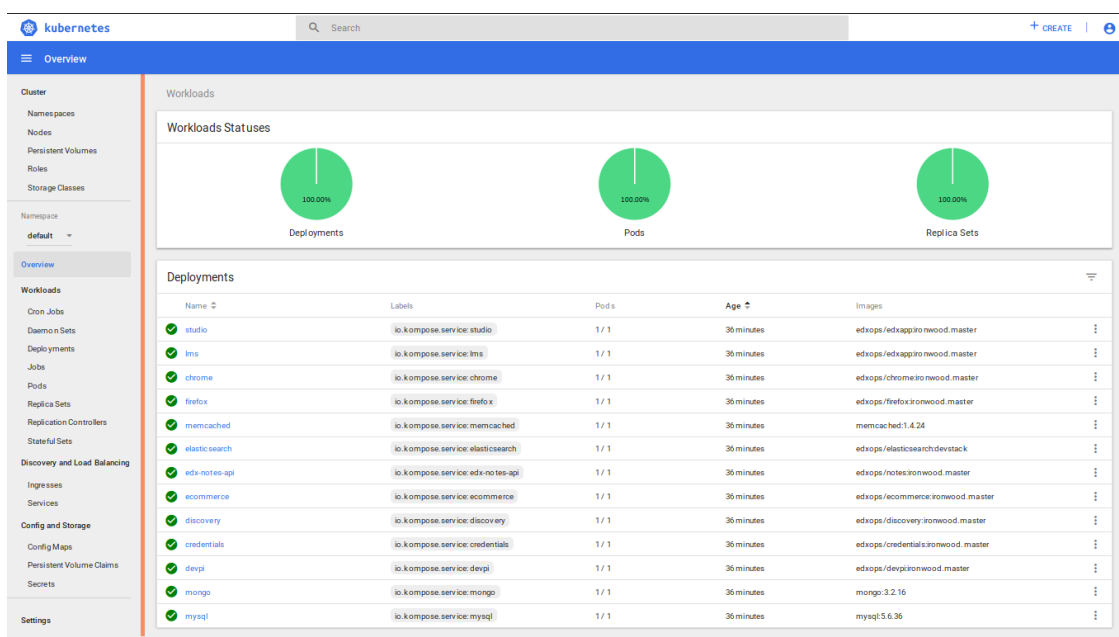


Figure 17: Kubernetes deployments after removing gradebook and forum

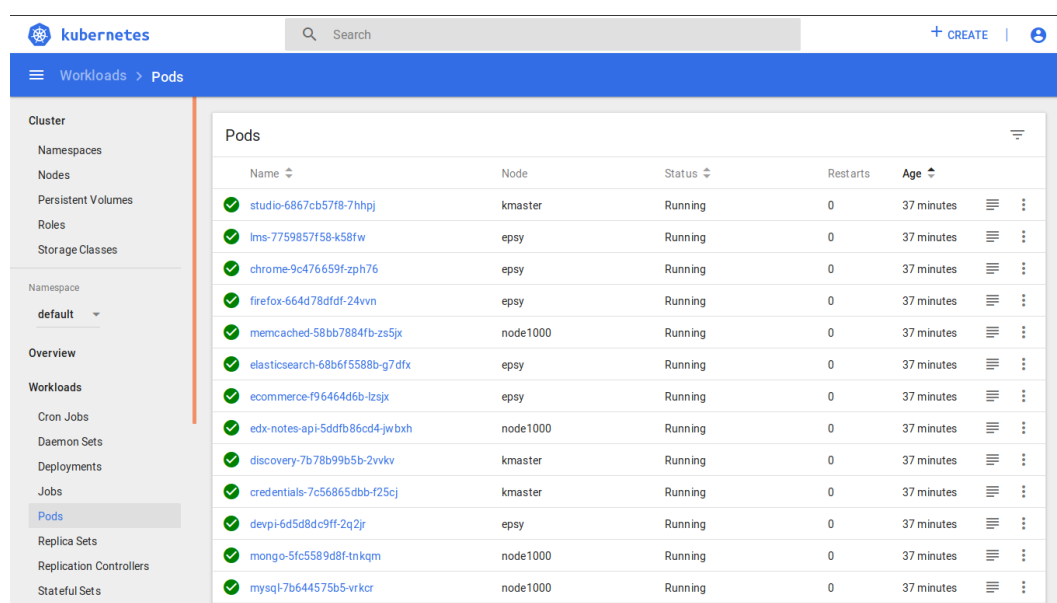
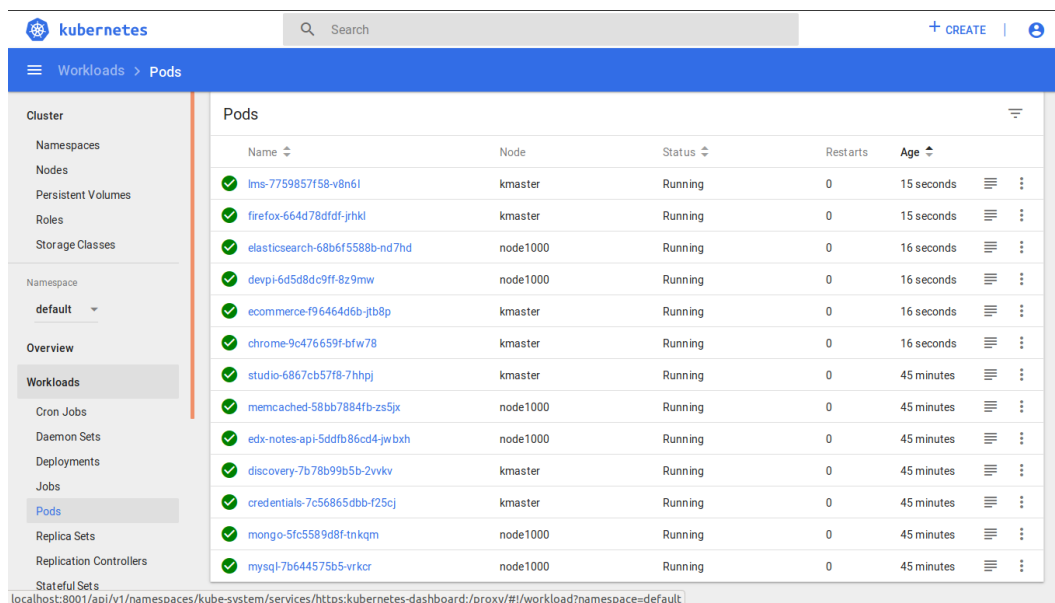


Figure 18: Pods running before deleting a node

docker commands. Since, our containers are running on a Kubernetes cluster, we used the **kubectl exec** in place of **docker exec** to run commands inside the containers. But we faced an internet connectivity issue for the pods:

The containers running inside the Kubernetes cluster were not able to connect to the internet

Commands those were to be executed inside a container and required internet access



Name	Node	Status	Restarts	Age
lms-7759857f58-v8n6l	kmaster	Running	0	15 seconds
firefox-664d78dfdf-jrhkl	kmaster	Running	0	15 seconds
elasticsearch-68b6f5588b-nd7hd	node1000	Running	0	16 seconds
devpi-6d5d8dc9ff-8z9mw	node1000	Running	0	16 seconds
ecommerce-f96464d6b-jb8p	kmaster	Running	0	16 seconds
chrome-9c476659f-bfw78	kmaster	Running	0	16 seconds
studio-6867cb57f8-7hhpj	kmaster	Running	0	45 minutes
memcached-58bb7884fb-zs5jx	node1000	Running	0	45 minutes
edx-notes-api-5ddfb86cd4-jwbxh	node1000	Running	0	45 minutes
discovery-7b78b99b5b-2vkv	kmaster	Running	0	45 minutes
credentials-7c56865dbb-f25cj	kmaster	Running	0	45 minutes
mongo-5fc5589d8f-tnkqm	node1000	Running	0	45 minutes
mysql-7b644575b5-vrkcr	node1000	Running	0	45 minutes

Figure 19: Pods running after deleting the node

failed, as containers were not able to connect to the internet.

Suggested resolutions:

These are some methods that may solve this issue (due to lack of time, we were not able to try them properly and anyone taking up this project from here on can try these methods):

- On further research, we found out that the internet connectivity problem of the containers was probably due to some issue in the DNS resolution. This can be further looked up using this link: <https://blog.yaakov.online/kubernetes-getting-pods-to-talk-to-the-internet/>[19]
- We can mention the commands that are to be executed inside a container inside its deployment file itself. This is probably a better way than the above one as here the commands will be executed inside any new container that is created through that deployment file. This can be done by using **command** and **args** fields in the deployment file.
- If we have an already running instance of Open edX devstack version on a machine, then we can use the command **docker commit** to create a new image that reflects the changes made on the container while provisioning and then use the new image for deployment purpose in our Kubernetes cluster.

Performance Testing Open edX Ironwood Release

Given a certain load, Performance Testing allows us to determine and examine non-functional parameters like speed and stability of an application.

Tools used for Performance Testing

JMeter is a popular tool used for performance testing. Apart from being an open-source software, JMeter is purely written in Java and hence, is platform independent. Also, it provides a GUI to easily set up a test plan. This, combined with the support for a full multithreading framework and the ability to test a wide range of protocols and applications, makes it a very good tool for performance testing.

One can install JMeter by downloading the required binaries from https://jmeter.apache.org/download_jmeter.cgi.

JMeter Test Scripts

A JMeter test script is a collection of user activities that have been pre-recorded using a proxy. These scripts are stored in the XML format with *.jmx* extension.

The test script can be recorded as follows:[20]

1. Firstly, create a new test plan. A test plan describes what JMeter should do when we run a test script.
2. Next, right-click on the newly created test plan and add a thread group. (Add->Threads (Users)->Thread Group) A thread group is used to simulate users for testing an application.
3. Add an instance of the HTTP(S) Test Script Recorder via the add sub-menu accessed by right-clicking on the test plan. (Add->Non-Test Elements->HTTP(S) Test Script Recorder)
4. In the recorder instance created in the previous step, set the Target Controller field to the newly created Thread Group. This field can be accessed in the Test Plan Creation tab.
5. Thereafter, configure the port field to an unused port on your system.
6. We must instruct the recorder to bypass recording the loading of static elements like images. To do this, go to the request tab in the recorder instance created earlier, and click on the Add Suggested Excludes button to add default static files' extensions. This can be modified as per convenience.
7. Since a user may pause for a while between successive requests, the need to add think time to our test script is evident. JMeter provides various timers. We will use a Uniform Random Timer here, which can be added to the HTTP(S) Test Script Recorder instance. With each timer, we associate some parameters. For a Uniform Random Timer, these are Random Delay Maximum and Constant Delay Offset. This timer samples a point from a Uniform Probability Distribution such that it

lies between 0 and the Random Delay Maximum value. Thereafter, it adds the Constant Delay Offset to it. To modify the Constant Delay Offset value, use $\{T\}$ in the corresponding field.

8. JMeter would now act as an HTTP proxy and would listen to all the incoming and outgoing requests on the configured port.
9. Now, we configure the proxy settings in the browser to send requests to the JMeter proxy.
10. Click on the Start button available in the State tab in the recorder instance to start the recording.
11. Note that the JMeter Root CA certificate must be installed in the web browser to record encrypted web requests.
12. To do this, go to the bin directory in the JMeter directory to locate the file named ApacheJMeterTemporaryRootCA, and install it in the system as well as the browser being used for recording the test script. In Firefox, for example, one needs to go to the options page, move to the Privacy and Security tab, click on the View Certificates button, and then click on the Import button to add the JMeter's self-signed certificate.
13. Once the recording starts, execute the test scenario.
14. Thereafter, click on the Stop button in the Transaction Controller to stop the recording.

Viewing JMeter Test Results using Listeners

JMeter listeners enable us to view and analyze the test results in graphical or tabular form. Some of these are:

- View Result Tree: This listener allows us to view a tree of samples' responses with their response codes. Furthermore, the response of each sample is shown separately.
- Summary and Aggregate Report: These listeners give an abridged view of information like Request Count, Average, Median, Min, Max, 90% Line, Error Rate, Throughput, Requests/second, and KB/sec for each uniquely named sample in the test plan. For low memory consumption, the Summary Report listener can be a good alternative.

Due to insufficient Java Heap Space, a test plan execution may result in JMeter giving an Out of Memory error. To resolve this, edit the `jmeter.bat` (for Windows) or `jmeter.sh` (for Linux) files and update the `JVM_ARGS` parameter accordingly.

Test Scenario

Our test scenario is as follows:

1. The user registers on the website.
2. The user views a list of available courses.
3. The user chooses a test course.
4. The user enrolls in the chosen course.
5. The user views the chosen course's dashboard.
6. The user chooses the quiz modules.
7. The user attempts the quiz.
8. The user submits the quiz responses for evaluation.

We had used this scenario since it involves all the activities that can be performed by a user on a course page.

Testing Open edX Native Installation

6.7 System Configurations

Here are the details of the hardware and software configuration of the application under test and the client machine that has been used to generate load.

Load Generator System Details

- Operating System: Windows 10 Home 64-bit
- RAM: 8GB
- Processor: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz x 2 Cores

Server Configuration

- Operating System: Ubuntu 16.04.6 LTS
- RAM: 7.61 GB
- Processor: Intel Common KVM Processor @ 2.095 MHz x 2

Test Details

- Duration: 1 Hour
- Virtual Connections: 100

Test Results

The application under test could handle 100 users for the specified server configuration under the current test setup. Furthermore, the test fails for around 120-130 users.

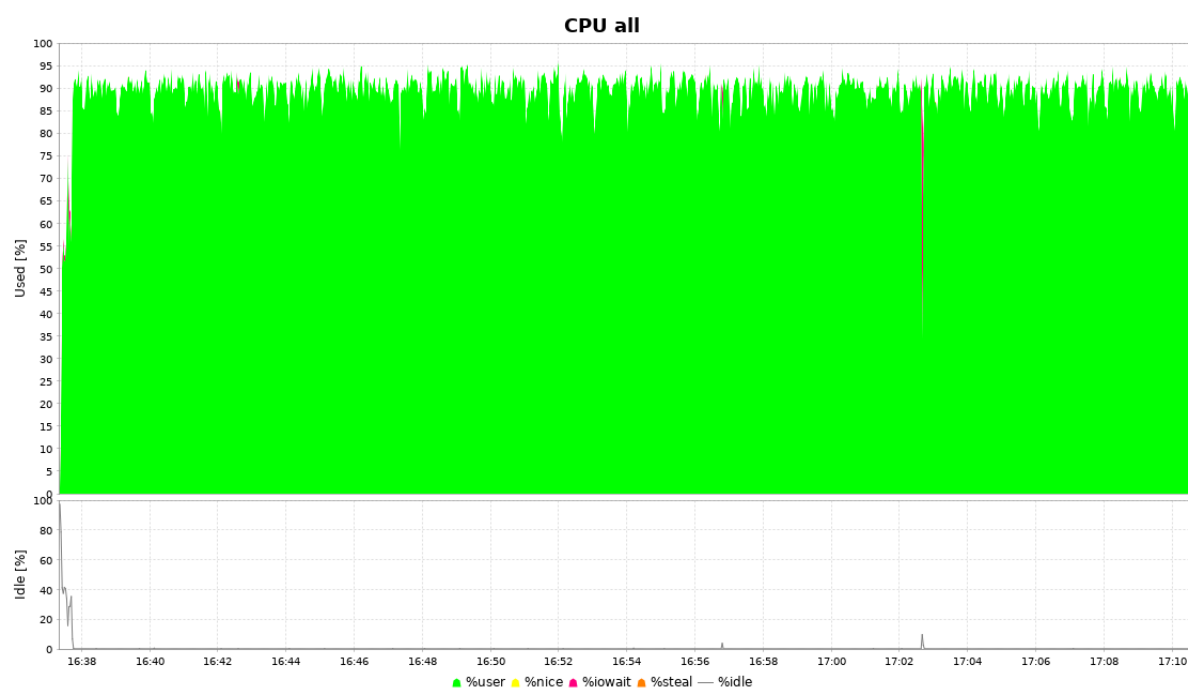


Figure 20: SAR CPU Metrics

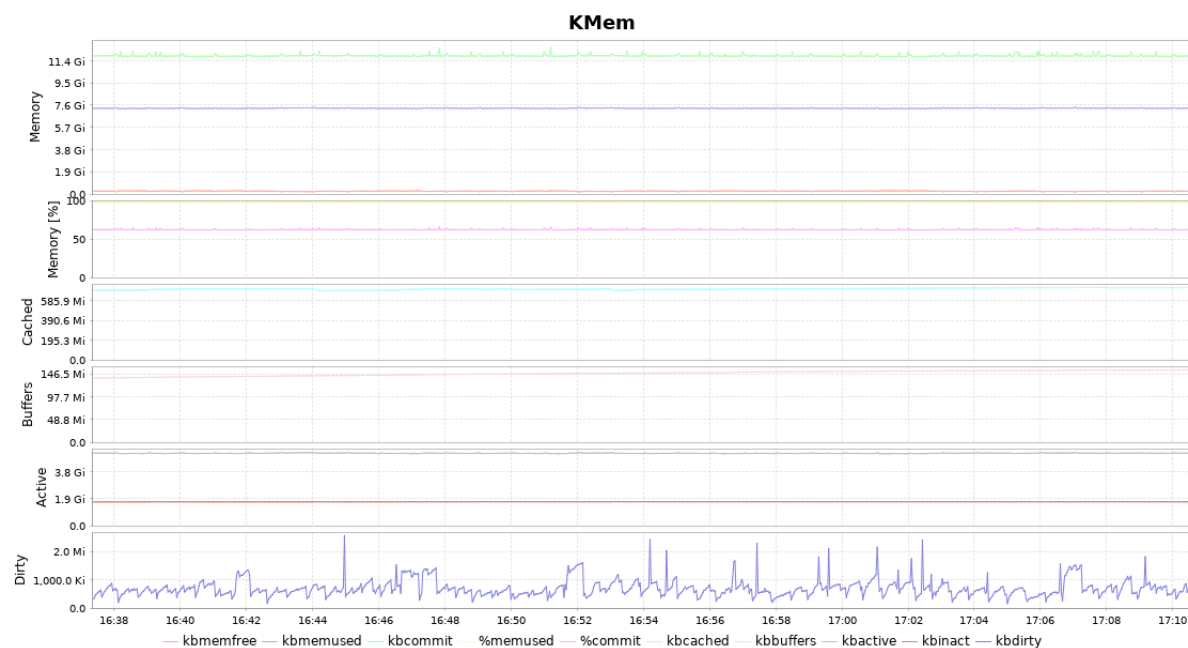


Figure 21: SAR Memory Metrics

```

user@vm5: ~
top - 16:51:42 up 21 days, 4:31, 3 users, load average: 9.54, 9.07, 7.88
Tasks: 271 total, 10 running, 247 sleeping, 4 stopped, 10 zombie
%Cpu(s): 82.8 us, 16.7 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.5 si, 0.0 st
KiB Mem : 7980484 total, 261864 free, 6412868 used, 1305752 buff/cache
KiB Swap: 12095168 total, 10978504 free, 1116664 used. 1136932 avail Mem

26792 www-data 20 0 896384 345860 28852 R 19.2 4.3 15:16.88 gunicorn
26782 www-data 20 0 893764 343372 28852 R 20.2 4.3 15:42.80 gunicorn
26787 www-data 20 0 889652 338624 28852 R 18.5 4.2 15:51.95 gunicorn
26780 www-data 20 0 895392 344844 28852 R 18.2 4.3 15:49.69 gunicorn
26790 www-data 20 0 902700 352196 28852 R 17.5 4.4 15:42.94 gunicorn
26792 www-data 20 0 892656 342260 28852 R 17.2 4.3 15:43.72 gunicorn
26784 www-data 20 0 896768 346268 28852 S 16.6 4.3 15:51.71 gunicorn
26785 www-data 20 0 902400 352040 28852 S 15.9 4.4 15:45.63 gunicorn
26788 www-data 20 0 892696 342324 28852 R 15.6 4.3 15:49.36 gunicorn
26702 www-data 20 0 764460 261432 7444 R 9.6 3.3 3:31.59 python
588 root 20 0 44620 10880 7656 S 6.0 0.1 15:32.39 systemd-j+
4287 mysql 20 0 1454928 89572 7152 S 5.6 1.1 50:55.92 mysqld
32214 postfix 20 0 68368 3476 2464 S 5.3 0.0 0:19.58 qmgr
9837 rabbitmq 20 0 1196576 50380 4260 S 4.3 0.6 634:59.51 beam.smp
22415 syslog 20 0 258448 3000 1636 S 1.7 0.0 3:58.58 rsyslogd
32212 root 20 0 65408 2132 2004 S 1.7 0.0 0:28.91 master
3811 mongod 20 0 844480 65388 8552 S 1.3 0.8 245:08.40 mongod
528 root 20 0 0 0 0 S 1.0 0.0 2:28.88 jbd2/sda1-8
1158 memcache 20 0 357064 16316 1772 S 1.0 0.2 3:55.02 memcached
17560 user 20 0 40656 3808 3040 R 0.7 0.0 0:04.04 top
21788 postfix 20 0 67884 4404 3944 S 0.7 0.1 0:00.05 trivial-re+
22863 postfix 20 0 67904 4420 3968 S 0.7 0.1 0:00.02 bounce
23279 www-data 20 0 754596 235548 6460 S 0.7 3.0 15:09.45 python
7 root 20 0 0 0 0 S 0.3 0.0 31:11.21 rcu sched

```

Figure 22: A look at the top command, 15 minutes into the test. Guicorn, which is a Python Web Server Gateway Interface HTTP Server, takes up most of the CPU resources.

user@vm5: ~

```

- 17:07:25 up 21 days,  4:46,  3 users,  load average: 9.82, 9.39, 8.78
cs: 265 total,  11 running, 240 sleeping,  4 stopped,  10 zombie
(s): 91.6 us,  7.6 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.8 si,  0.0 st
Mem : 7980484 total,  248276 free,  6400176 used, 1332032 buff/cache
Swap: 12095168 total, 10973708 free,  1121460 used. 1145236 avail Mem

```

ID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
87	www-data	20	0	888308	337612	28852	R	20.8	4.2	18:58.67	gunicorn
82	www-data	20	0	897240	346720	28852	R	20.1	4.3	18:48.76	gunicorn
92	www-data	20	0	900888	350140	28852	R	20.1	4.4	18:50.32	gunicorn
85	www-data	20	0	892708	342348	28852	R	19.8	4.3	18:51.86	gunicorn
88	www-data	20	0	890700	340120	28852	R	19.8	4.3	18:54.76	gunicorn
84	www-data	20	0	899836	349336	28852	R	19.5	4.4	18:57.57	gunicorn
90	www-data	20	0	897292	346788	28852	R	19.5	4.3	18:49.71	gunicorn
80	www-data	20	0	893088	342588	28852	R	19.1	4.3	18:56.01	gunicorn
92	www-data	20	0	764460	261432	7444	S	15.5	3.3	4:27.98	python
87	mysql	20	0	1454928	92788	7152	S	7.3	1.2	52:18.81	mysqld
87	rabbitmq	20	0	1201712	50420	4260	S	5.0	0.6	635:36.14	beam.smp
88	memcache	20	0	357064	16316	1772	S	1.7	0.2	4:12.86	memcached
81	mongodb	20	0	844480	65396	8552	R	1.3	0.8	245:23.31	mongod
88	www-data	20	0	35244	6140	2924	S	1.3	0.1	0:25.32	nginx
88	root	20	0	44620	12200	8976	S	0.7	0.2	15:42.19	systemd-journal
87	www-data	20	0	35144	6172	2944	S	0.7	0.1	0:19.52	nginx
80	user	20	0	40792	3808	3040	R	0.7	0.0	0:08.93	top
88	www-data	20	0	754504	233680	5932	S	0.7	2.9	14:49.98	python
89	www-data	20	0	754596	235532	6460	S	0.7	3.0	15:14.53	python
85	www-data	20	0	391748	67632	5312	S	0.7	0.8	0:29.19	ruby
83	root	20	0	0	0	0	S	0.3	0.0	2:12.82	ksoftirqd/1
86	www-data	20	0	34840	5496	2912	S	0.3	0.1	0:29.50	nginx
89	www-data	20	0	60888	10224	4084	S	0.3	0.1	21:41.01	supervisord
81	elastic+	20	0	3054468	267640	8800	S	0.3	3.4	133:13.75	java
88	user	20	0	9712	1120	1028	S	0.3	0.0	0:01.16	sadc
89	user	20	0	9712	1076	984	S	0.3	0.0	0:01.22	sadc
83	www-data	20	0	332616	46388	8700	S	0.3	0.6	0:23.85	python

Figure 23: A look at the top command, 30 minutes into the test. Guicorn, which is a Python Web Server Gateway Interface HTTP Server, takes up most of the CPU resources.

```

user@vm5: ~
top - 17:22:23 up 21 days, 5:01, 3 users, load average: 9.78, 9.25, 9.02
Tasks: 248 total, 8 running, 226 sleeping, 4 stopped, 10 zombie
%Cpu(s): 84.6 us, 15.1 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.3 si, 0.0 st
KiB Mem : 7980484 total, 276760 free, 6393076 used, 1310648 buff/cache
KiB Swap: 12095168 total, 10966484 free, 1128684 used. 1152376 avail Mem

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
26788	www-data	20	0	890904	340532	28852	S	21.9	4.3	21:53.62	unicorn
26787	www-data	20	0	901084	350580	28852	R	21.2	4.4	21:57.39	unicorn
26780	www-data	20	0	894624	343908	28852	R	19.9	4.3	21:54.26	unicorn
26782	www-data	20	0	901024	350504	28852	R	19.9	4.4	21:47.64	unicorn
26784	www-data	20	0	892348	341976	28852	S	19.9	4.3	21:54.85	unicorn
26790	www-data	20	0	895296	344792	28852	R	19.9	4.3	21:48.39	unicorn
26785	www-data	20	0	898688	348328	28852	S	19.2	4.4	21:49.21	unicorn
26792	www-data	20	0	896384	345860	28852	R	17.5	4.3	21:48.21	unicorn
26702	www-data	20	0	764460	261432	7444	S	8.9	3.3	5:19.16	python
4287	mysql	20	0	1454928	96752	7148	S	5.3	1.2	53:38.56	mysqld
9837	rabbitmq	20	0	1194008	50508	4260	S	2.3	0.6	636:10.45	beam.smp
1158	memcache	20	0	357064	16316	1772	S	1.3	0.2	4:30.18	memcached
588	root	20	0	44620	5868	2644	S	1.0	0.1	15:51.57	systemd-journal
3811	mongodb	20	0	844480	65656	8544	S	1.0	0.8	245:37.50	mongod
3	root	20	0	0	0	0	S	0.3	0.0	1:48.67	ksoftirqd/0
7	root	20	0	0	0	0	S	0.3	0.0	31:16.28	rcu_sched
10517	www-data	20	0	35144	6172	2944	S	0.3	0.1	0:22.28	nginx
10518	www-data	20	0	34508	5256	2924	S	0.3	0.1	0:28.58	nginx
12139	www-data	20	0	60888	10224	4084	R	0.3	0.1	21:45.01	supervisord
16781	elastic+	20	0	3054468	267976	8892	S	0.3	3.4	133:19.38	java
17560	user	20	0	40792	3808	3040	R	0.3	0.0	0:13.29	top
23267	www-data	20	0	744380	221024	6004	S	0.3	2.8	14:56.83	python
23279	www-data	20	0	754596	235440	6460	S	0.3	3.0	15:19.25	python
23359	www-data	20	0	204836	28576	3244	S	0.3	0.4	1:37.23	unicorn
1	root	20	0	37984	4988	3176	S	0.0	0.1	16:03.67	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:01.20	kthreadd
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H

Figure 24: A look at the top command, 45 minutes into the test. Guicorn, which is a Python Web Server Gateway Interface HTTP Server, takes up most of the CPU resources.


```

user@vm5: ~
top - 17:37:10 up 21 days, 5:16, 3 users, load average: 9.21, 9.08, 9.08
Tasks: 268 total, 9 running, 245 sleeping, 4 stopped, 10 zombie
%Cpu(s): 66.7 us, 33.3 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 7980484 total, 266332 free, 6406588 used, 1307564 buff/cache
KiB Swap: 12095168 total, 10946316 free, 1148852 used. 1142048 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
17560 user       20   0   40792    3808    3040 R   66.7   0.0   0:17.63 top
14187 root        20   0         0         0         0 S   33.3   0.0   0:00.06 kworker/u4:1
24542 www-data   20   0  391848   67788   5312 R   33.3   0.8   0:37.59 ruby
26785 www-data   20   0  897916  347428  28852 R   33.3   4.4  24:45.57 gunicorn
26792 www-data   20   0  900376  349852  28852 R   33.3   4.4  24:42.57 gunicorn
   1 root        20   0   37984    4988    3176 S    0.0   0.1  16:04.20 systemd
   2 root        20   0         0         0         0 S    0.0   0.0   0:01.20 kthreadd
   3 root        20   0         0         0         0 S    0.0   0.0   1:49.15 ksoftirqd/0
   5 root         0 -20         0         0         0 S    0.0   0.0   0:00.00 kworker/0:0H
   7 root        20   0         0         0         0 S    0.0   0.0  31:18.64 rcu_sched
   8 root        20   0         0         0         0 S    0.0   0.0   0:00.00 rcu_bh
   9 root        rt    0         0         0         0 S    0.0   0.0   0:27.82 migration/0
  10 root        rt    0         0         0         0 S    0.0   0.0   0:09.01 watchdog/0
  11 root        rt    0         0         0         0 S    0.0   0.0   0:06.91 watchdog/1
  12 root        rt    0         0         0         0 S    0.0   0.0   0:28.15 migration/1
  13 root        20   0         0         0         0 S    0.0   0.0   2:13.84 ksoftirqd/1
  15 root         0 -20         0         0         0 S    0.0   0.0   0:00.00 kworker/1:0H
  16 root        20   0         0         0         0 S    0.0   0.0   0:00.00 kdevtmpfs
  17 root         0 -20         0         0         0 S    0.0   0.0   0:00.00 netns
  18 root         0 -20         0         0         0 S    0.0   0.0   0:00.00 perf
  19 root        20   0         0         0         0 S    0.0   0.0   0:04.79 khungtaskd
  20 root         0 -20         0         0         0 S    0.0   0.0   0:00.00 writeback
  21 root        25   5         0         0         0 S    0.0   0.0   0:00.00 ksm
  23 root         0 -20         0         0         0 S    0.0   0.0   0:00.00 crypto
  24 root         0 -20         0         0         0 S    0.0   0.0   0:00.00 kintegrityd
  25 root         0 -20         0         0         0 S    0.0   0.0   0:00.00 bioset
  26 root         0 -20         0         0         0 S    0.0   0.0   0:00.00 kblockd

```

Figure 25: A look at the top command, 60 minutes into the test. Guicorn, which is a Python Web Server Gateway Interface HTTP Server, takes up most of the CPU resources.

Testing Open edX Developer Stack (Devstack)

6.8 System Configurations

Here are the details of the hardware and software configuration of the application under test and the client machine that has been used to generate load.

Load Generator System Details

- Operating System: Windows 10 Home 64-bit
- RAM: 8GB
- Processor: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz x 2 Cores

Server Configuration

- Operating System: Ubuntu 16.04.4 LTS
- RAM: 7.61 GB
- Processor: Intel Common KVM Processor @ 2.095 MHz x 2

Test Details

- Duration: 1 Hour
- Virtual Connections: 10

Test Results

The application under test could handle 10 users for the specified server configuration under the current test setup. Furthermore, the test fails for around 15-20 users.

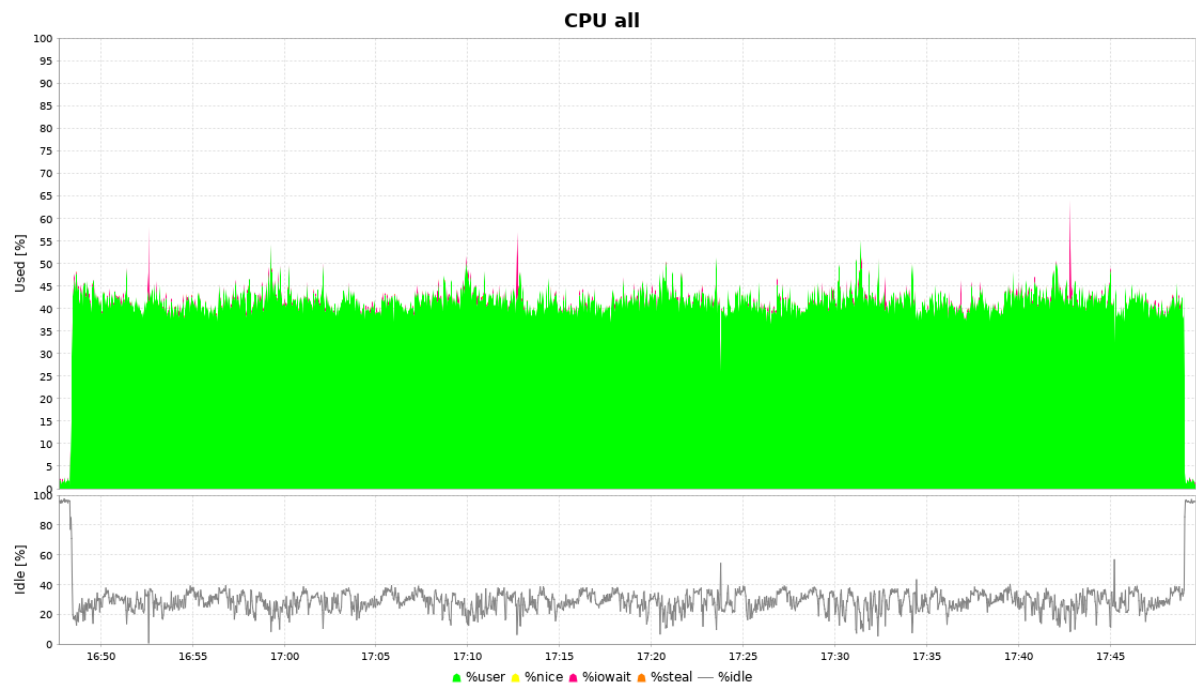


Figure 26: SAR CPU Metrics

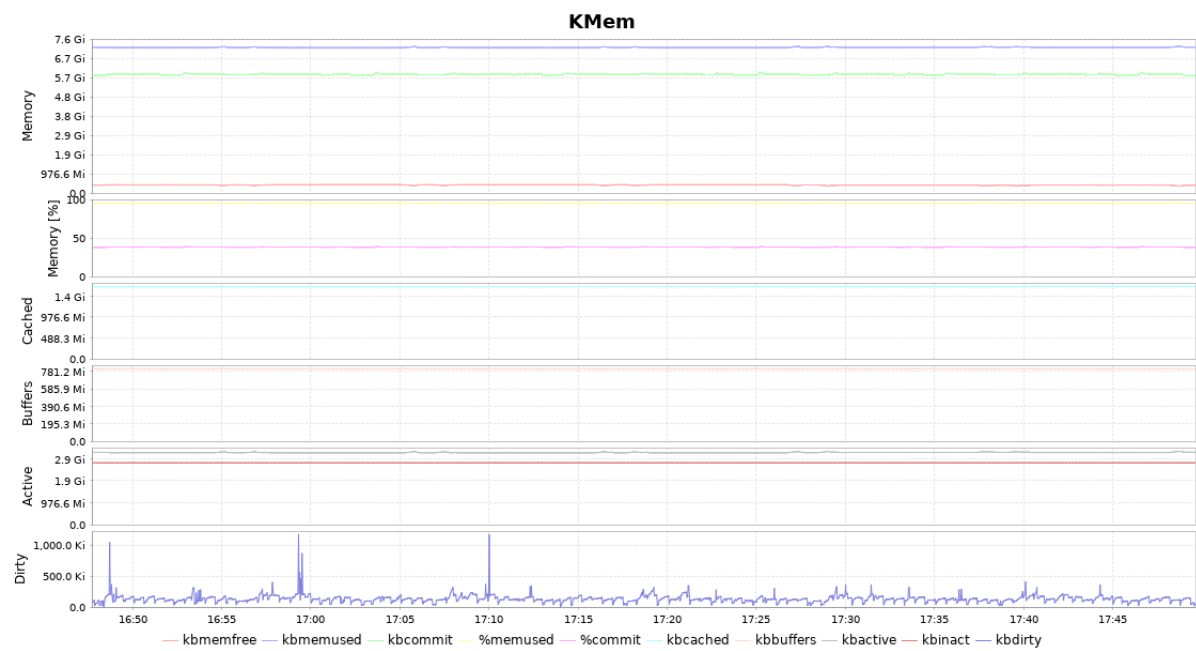


Figure 27: SAR Memory Metrics

```

top - 17:39:10 up 24 days, 5:22, 3 users, load average: 2.46, 3.31, 3.70
Tasks: 193 total, 1 running, 192 sleeping, 0 stopped, 0 zombie
%Cpu(s): 38.9 us, 28.6 sy, 0.0 ni, 31.9 id, 0.0 wa, 0.0 hi, 0.5 si, 0.0 st
KiB Mem : 7980484 total, 405856 free, 3641224 used, 3933404 buff/cache
KiB Swap: 8188924 total, 8111528 free, 77396 used. 3908028 avail Mem

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
24635	root	20	0	2002208	792816	27348	S	144.2	9.9	903:21.15	python
10424	root	20	0	246688	47964	3880	S	5.6	0.6	994:33.08	python
8471	999	20	0	1821340	618596	14260	S	0.7	7.8	24:14.04	mysqld
13545	user	20	0	40544	3688	3000	R	0.7	0.0	0:07.95	top
8194	999	20	0	235136	55756	22364	S	0.3	0.7	111:37.82	mongod
9297	user	20	0	113380	8156	1016	S	0.3	0.1	17:06.17	x11vnc
12907	user	20	0	9712	1072	984	S	0.3	0.0	0:03.64	sadc
12908	user	20	0	9712	1040	948	S	0.3	0.0	0:03.55	sadc
14291	root	20	0	0	0	0	S	0.3	0.0	0:00.18	kworker/u4:2
23800	root	20	0	582060	80820	8736	S	0.3	1.0	5:21.01	ruby
1	root	20	0	37996	5508	3484	S	0.0	0.1	0:33.67	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:01.61	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:09.30	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
7	root	20	0	0	0	0	S	0.0	0.0	9:55.34	rcu_sched
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	rt	0	0	0	0	S	0.0	0.0	0:00.40	migration/0
10	root	rt	0	0	0	0	S	0.0	0.0	0:10.47	watchdog/0
11	root	rt	0	0	0	0	S	0.0	0.0	0:08.50	watchdog/1
12	root	rt	0	0	0	0	S	0.0	0.0	0:00.43	migration/1
13	root	20	0	0	0	0	S	0.0	0.0	0:27.07	ksoftirqd/1
15	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/1:0H
16	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
17	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
18	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	perf
19	root	20	0	0	0	0	S	0.0	0.0	0:12.68	khungtaskd
20	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	writeback
21	root	25	5	0	0	0	S	0.0	0.0	0:00.00	ksmd
22	root	39	19	0	0	0	S	0.0	0.0	0:14.46	khugepaged
23	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	crypto
24	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
25	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	bioaset
26	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
27	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	ata_sff
28	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	md
29	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	devfreq_wq
33	root	20	0	0	0	0	S	0.0	0.0	0:16.93	kswapd0
34	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	vmstat
35	root	20	0	0	0	0	S	0.0	0.0	0:00.06	fsnotify_mark
36	root	20	0	0	0	0	S	0.0	0.0	0:00.00	ecryptfs-kthr
52	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kthrotld
53	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	acpi_thermal
54	root	20	0	0	0	0	S	0.0	0.0	9:12.81	vballoon

Figure 28: A look at the top command, 15 minutes into the test. Python takes up most of the CPU Resources.

```
user@vm14: ~
```

```
top - 17:09:54 up 24 days, 4:52, 3 users, load average: 2.94, 2.96, 3.17
Tasks: 194 total, 2 running, 192 sleeping, 0 stopped, 0 zombie
%Cpu(s): 46.2 us, 30.5 sy, 0.0 ni, 22.3 id, 0.3 wa, 0.0 hi, 0.5 si, 0.2 st
KiB Mem : 7980484 total, 426784 free, 3622320 used, 3931380 buff/cache
KiB Swap: 8188924 total, 8111528 free, 77396 used. 3926660 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
24635	root	20	0	2027308	774632	27348	S	154.5	9.7	860:16.23	python
10424	root	20	0	246688	47964	3880	S	5.3	0.6	992:58.04	python
8471	999	20	0	1821340	618596	14260	S	1.3	7.8	23:54.23	mysqld
8194	999	20	0	235136	55756	22364	S	1.0	0.7	111:27.16	mongod
23800	root	20	0	582060	80780	8736	S	1.0	1.0	5:19.02	ruby
12609	root	20	0	1250640	74704	28700	S	0.7	0.9	38:02.80	dockerd
8120	_apt	20	0	3586204	300052	6576	S	0.3	3.8	70:34.91	java
8499	root	20	0	1516272	52280	3940	S	0.3	0.7	39:36.97	devpi-server
9297	user	20	0	113380	8156	1016	S	0.3	0.1	17:04.56	x11vnc
9300	user	20	0	4507544	51996	8396	S	0.3	0.7	20:38.49	java
9841	root	20	0	10732	4548	3708	S	0.3	0.1	1:04.46	containerd-shim
9859	root	20	0	9324	4088	3328	S	0.3	0.1	1:00.17	containerd-shim
13006	user	20	0	92828	4344	3384	S	0.3	0.1	0:00.66	sshd
13466	root	20	0	0	0	0	S	0.3	0.0	0:00.06	kworker/u4:0
13545	user	20	0	40544	3688	3000	R	0.3	0.0	0:00.44	top
24272	root	20	0	11788	5284	4128	S	0.3	0.1	2:26.39	containerd-shim
1	root	20	0	37996	5508	3484	S	0.0	0.1	0:33.65	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:01.61	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:09.18	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
7	root	20	0	0	0	0	R	0.0	0.0	9:54.21	rcu_sched
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	rt	0	0	0	0	S	0.0	0.0	0:00.40	migration/0
10	root	rt	0	0	0	0	S	0.0	0.0	0:10.46	watchdog/0
11	root	rt	0	0	0	0	S	0.0	0.0	0:08.50	watchdog/1
12	root	rt	0	0	0	0	S	0.0	0.0	0:00.43	migration/1
13	root	20	0	0	0	0	S	0.0	0.0	0:26.74	ksoftirqd/1
15	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/1:0H
16	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
17	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
18	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	perf
19	root	20	0	0	0	0	S	0.0	0.0	0:12.68	khungtaskd
20	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	writeback
21	root	25	5	0	0	0	S	0.0	0.0	0:00.00	ksmd
22	root	39	19	0	0	0	S	0.0	0.0	0:14.43	khugepaged
23	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	crypto
24	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	integrityd
25	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	bioaset
26	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
27	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	ata_sff
28	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	md
29	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	devfreq_wq
33	root	20	0	0	0	0	S	0.0	0.0	0:16.93	kswapd0

Figure 29: A look at the top command, 30 minutes into the test. Python takes up most of the CPU Resources.

```
user@vm14: ~
```

```
top - 17:17:12 up 24 days, 5:00, 3 users, load average: 3.51, 3.73, 3.49
Tasks: 194 total, 1 running, 193 sleeping, 0 stopped, 0 zombie
%Cpu(s): 39.1 us, 34.0 sy, 0.0 ni, 26.4 id, 0.0 wa, 0.0 hi, 0.3 si, 0.2 st
KiB Mem : 7980484 total, 426284 free, 3622608 used, 3931592 buff/cache
KiB Swap: 8188924 total, 8111528 free, 77396 used. 3926584 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
24635	root	20	0	2002208	774064	27348	S	155.6	9.7	871:04.43	python
10424	root	20	0	246688	47964	3880	S	3.6	0.6	993:21.69	python
8471	999	20	0	1821340	618596	14260	S	1.0	7.8	23:59.09	mysqld
13545	user	20	0	40544	3688	3000	R	1.0	0.0	0:02.31	top
8194	999	20	0	235136	55756	22364	S	0.7	0.7	111:29.74	mongod
8120	_apt	20	0	3586204	300052	6576	S	0.3	3.8	70:36.58	java
8499	root	20	0	1516272	52280	3940	S	0.3	0.7	39:37.93	devpi-server
9204	user	20	0	235164	32884	6776	S	0.3	0.4	5:25.95	Xvfb
10772	root	20	0	363008	124224	10876	S	0.3	1.6	4:51.07	python
11215	root	20	0	1388940	303904	24908	S	0.3	3.8	5:09.84	python
12609	root	20	0	1250640	74904	28700	S	0.3	0.9	38:03.85	dockerd
12904	user	20	0	4468	756	684	S	0.3	0.0	0:00.22	sar
12907	user	20	0	9712	1072	984	S	0.3	0.0	0:02.14	sadc
12909	user	20	0	9712	1024	936	S	0.3	0.0	0:02.26	sadc
1	root	20	0	37996	5508	3484	S	0.0	0.1	0:33.66	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:01.61	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:09.21	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
7	root	20	0	0	0	0	S	0.0	0.0	9:54.48	rcu_sched
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	rt	0	0	0	0	S	0.0	0.0	0:00.40	migration/0
10	root	rt	0	0	0	0	S	0.0	0.0	0:10.47	watchdog/0
11	root	rt	0	0	0	0	S	0.0	0.0	0:08.50	watchdog/1
12	root	rt	0	0	0	0	S	0.0	0.0	0:00.43	migration/1
13	root	20	0	0	0	0	S	0.0	0.0	0:26.82	ksoftirqd/1
15	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/1:0H
16	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
17	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
18	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	perf
19	root	20	0	0	0	0	S	0.0	0.0	0:12.68	khungtaskd
20	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	writeback
21	root	25	5	0	0	0	S	0.0	0.0	0:00.00	ksmd
22	root	39	19	0	0	0	S	0.0	0.0	0:14.44	khugepaged
23	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	crypto
24	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
25	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	bioset
26	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
27	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	ata_sff
28	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	md
29	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	devfreq_wq
33	root	20	0	0	0	0	S	0.0	0.0	0:16.93	kswapd0
34	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	vmstat
35	root	20	0	0	0	0	S	0.0	0.0	0:00.06	fsnotify_mar

Figure 30: A look at the top command, 45 minutes into the test. Python takes up most of the CPU Resources.


```

user@vm4: ~
top - 17:29:04 up 24 days, 5:12, 3 users, load average: 2.80, 3.56, 3.67
Tasks: 194 total, 1 running, 193 sleeping, 0 stopped, 0 zombie
%Cpu(s): 41.3 us, 26.8 sy, 0.0 ni, 31.4 id, 0.0 wa, 0.0 hi, 0.3 si, 0.2 st
KiB Mem : 7980484 total, 361532 free, 3686512 used, 3932440 buff/cache
KiB Swap: 8188924 total, 8111528 free, 77396 used. 3862784 avail Mem

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
24635	root	20	0	2040860	837864	27348	S	140.6	10.5	888:27.41	python
10424	root	20	0	246688	47964	3880	S	5.3	0.6	993:59.95	python
23800	root	20	0	582060	80808	8736	S	2.3	1.0	5:20.24	ruby
8194	999	20	0	235136	55756	22364	S	1.0	0.7	111:34.03	mongod
8471	999	20	0	1821340	618596	14260	S	1.0	7.8	24:06.86	mysqld
13765	root	20	0	0	0	0	S	0.7	0.0	0:00.32	kworker/u4:0
7	root	20	0	0	0	0	S	0.3	0.0	9:54.95	rcu_sched
8120	_apt	20	0	3586204	300052	6576	S	0.3	3.8	70:39.23	java
8499	root	20	0	1516272	52280	3940	S	0.3	0.7	39:39.43	devpi-server
9297	user	20	0	113380	8156	1016	S	0.3	0.1	17:05.60	x11vnc
12495	root	20	0	1094968	20904	3108	S	0.3	0.3	8:23.09	containerd
12907	user	20	0	9712	1072	984	S	0.3	0.0	0:02.95	sadc
12908	user	20	0	9712	1040	948	S	0.3	0.0	0:02.84	sadc
12909	user	20	0	9712	1024	936	S	0.3	0.0	0:03.17	sadc
13006	user	20	0	92828	4344	3384	S	0.3	0.1	0:01.73	sshd
13545	user	20	0	40544	3688	3000	R	0.3	0.0	0:05.42	top
24272	root	20	0	11788	5252	4128	S	0.3	0.1	2:27.56	containerd-shim
1	root	20	0	37996	5508	3484	S	0.0	0.1	0:33.66	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:01.61	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:09.26	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	rt	0	0	0	0	S	0.0	0.0	0:00.40	migration/0
10	root	rt	0	0	0	0	S	0.0	0.0	0:10.47	watchdog/0
11	root	rt	0	0	0	0	S	0.0	0.0	0:08.50	watchdog/1
12	root	rt	0	0	0	0	S	0.0	0.0	0:00.43	migration/1
13	root	20	0	0	0	0	S	0.0	0.0	0:26.96	ksoftirqd/1
15	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/1:0H
16	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
17	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
18	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	perf
19	root	20	0	0	0	0	S	0.0	0.0	0:12.68	khungtaskd
20	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	writeback
21	root	25	5	0	0	0	S	0.0	0.0	0:00.00	ksmd
22	root	39	19	0	0	0	S	0.0	0.0	0:14.45	khugepaged
23	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	crypto
24	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
25	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	bioaset
26	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
27	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	ata_sff
28	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	md
29	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	devfreq_wq
33	root	20	0	0	0	0	S	0.0	0.0	0:16.93	kswapd0

Figure 31: A look at the top command, 60 minutes into the test. Python takes up most of the CPU Resources.

CONTENTS

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received	Sent	Avg. Bytes
309 /	467	31471	26	74382	17594.29	0.00%	0.23709	1.61	0.43	6940.9
313 /register	462	17669	105	44435	8535.64	0.00%	0.23664	2.72	0.24	11763.9
324 /user_api/v1/account/registrati	458	17503	264	44034	7988.56	0.00%	0.24067	0.33	0.32	1388.8
325 /	458	33225	204	78036	14456.6	0.00%	0.23193	1.49	0.56	6578.9
326 /dashboard	453	17476	164	44284	7792.17	0.00%	0.23172	1.35	0.27	5981.9
336 /courses	449	17407	120	43568	7644.5	0.00%	0.23766	1.17	0.28	5020.4
340 /search/course_discovery/	449	16704	56	41720	7587.38	0.00%	0.23471	0.51	0.32	2238.8
344 /courses/course-v1:edX+Deme	443	19313	402	45575	7937.03	0.00%	0.22953	1.42	0.28	6354.3
350 /change_enrollment	440	19322	142	44096	7636.84	0.00%	0.23375	0.1	0.32	419
351 /dashboard	440	19306	461	42813	7222.98	0.00%	0.23015	1.57	0.28	6966
353 /courses/course-v1:edX+Deme	440	22947	4222	49514	7467.51	0.00%	0.22494	2.35	0.27	10684.9
368 /event	440	21945	927	44781	8474.88	0.00%	0.22253	0.09	0.34	426
458 /event	440	23692	1978	44132	8519.23	0.00%	0.22127	0.09	0.39	426
459 /courses/course-v1:edX+Deme	435	58331	17509	92077	16739.07	0.00%	0.21711	17.52	0.6	82614.2
460 /courses/course-v1:edX+Deme	416	33240	12588	53849	8958.8	0.00%	0.21116	16.87	0.3	81815.5
498 /courses/course-v1:edX+Deme	387	24169	8809	44520	8902.71	0.00%	0.20016	0.16	0.3	811.3
501 /event	374	22862	8273	43750	8182.27	0.00%	0.19608	0.08	0.4	426
502 /courses/course-v1:edX+Deme	364	24479	11310	44928	9034.08	0.00%	0.19425	0.48	0.36	2523
503 /event	359	21414	10118	41553	7664.61	0.00%	0.19473	0.08	3.03	426
TOTAL	8174	24316	26	92077	13799.58	0.00%	4.01031	48.37	8.7	12350.4

Figure 32: JMeter Summary Report for Open edX Native.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received	Sent	Avg. Bytes
505 /	60	45720	65	67966	21874.03	0.00%	0.01887	0.58	0.07	31202.7
572 /register	60	41969	315	54448	17739.53	0.00%	0.01859	0.7	0.11	38549.2
606 /user_api/v1/account/regist	60	12049	3183	16794	2432.88	0.00%	0.01849	0.05	0.04	2516.6
607 /	60	30228	27788	34045	1268.76	0.00%	0.01833	0.5	0.08	27842
608 /dashboard	60	30258	28003	33901	1294.17	0.00%	0.01834	0.49	0.04	27328.9
620 /courses	60	11264	3047	17583	3377.43	0.00%	0.01848	0.45	0.04	24769
623 /courses/course-v1:edX+D	60	39603	30718	49094	3607.05	0.00%	0.01828	0.52	0.04	29282.2
630 /change_enrollment	60	1884	750	3192	546.91	0.00%	0.01849	0.01	0.05	443
631 /dashboard	60	50971	35640	61853	6081.16	0.00%	0.01827	0.57	0.04	32168.1
634 /courses/course-v1:edX+D	60	80800	63578	95378	8477.24	0.00%	0.01809	1.54	0.04	86914.9
650 /event	60	647	247	1453	301.12	0.00%	0.01848	0.01	0.05	450
701 /event	60	676	222	1745	306.8	0.00%	0.01848	0.01	0.05	450
702 /courses/course-v1:edX+D	60	131800	126434	138025	2369.1	0.00%	0.01779	9.31	0.09	535781.9
739 /courses/course-v1:edX+D	50	100446	90343	115660	7293.53	0.00%	0.01845	9.64	0.05	535058.8
814 /courses/course-v1:edX+D	50	3417	1213	6092	1063.3	0.00%	0.01913	0.05	0.05	2721.9
817 /event	50	563	180	1147	230.95	0.00%	0.01916	0.01	0.06	450
818 /courses/course-v1:edX+D	50	7637	3211	12106	1912.21	0.00%	0.0191	0.21	0.06	11332
819 /event	50	776	277	1475	291	0.00%	0.01914	0.01	0.32	450
TOTAL	1030	33315	65	138025	37722.73	0.00%	0.28285	20.85	1.05	75497.6

Figure 33: JMeter Summary Report for Open edX Devstack.

Test Results

- Under the current test setup and server configurations, it is evident that Open edX Native can handle around 100 virtual connections, which is much better than 10, the number of virtual connections that could be handled by Open edX Devstack.
- In 1 hour, the total number of transaction completed by Open edX Native was 8174, much more than that of Open edX Devstack, which amounted to 1030.
- Furthermore, the average response time of Open edX Native is 24.316 seconds, which outperforms Open edX Devstack, whose average response time is 33.313 seconds.
- Along with that, the throughput, which has been measured in requests/sec, for Open edX Native is 4.01, whereas it is 0.28 for Open edX Devstack.
- As for the CPU Utilization, Open edX Native consumes around 85-95% of CPU Resources, which is much more than the CPU Utilization of Open edX Devstack (40-50%).
- The memory consumption was close to 100% in both the setups.
- A closer look at the Summary Reports shows that the response time recorded corresponding to the case where the enrolled course's dashboard is accessed is very high, relative to other requests in a given test plan. Moreover, this response time is exceedingly high for Open edX Devstack.

The official Open edX documentation states that nginx and gunicorn have been disabled in Devstack. Hence, it uses Django's runserver as an alternative. This can be verified by the top command's result as shown in figures 14-17, 20-23.

References

- [1] "Kubernetes Pods images "https://2.bp.blogspot.com/-a7tbQ-93uCA/W9t14zgHlVI/AAAAAAAAACGE/762QwS7_iFM0odmtN-B0tdeHy_I2TQjpgCLcBGAs/s1600/Screen%2BShot%2B2018-11-01%2Bat%2B1.45.06%2BPM.png". Accessed on July 2019."
- [2] "Kubernetes Deployment image "https://storage.googleapis.com/cdn.thenewstack.io/media/2017/11/07751442-deployment.png". Accessed on July 2019."
- [3] "Kubernetes Services image "https://matthewpalmer.net/kubernetes-app-developer/articles/nodeport.png". Accessed on July 2019."
- [4] "Ingress Network image "https://user-images.githubusercontent.com/899878/45129885-91f38a80-b1c9-11e8-9833-0a04455c8304.png". Accessed on July 2019."
- [5] "Flannel Network image "http://blog.shippable.com/hubfs/Routing_using_kube-proxy_and_overlay_network.png". Accessed on July 2019."

- [6] “Calico Network image ”<https://i2.wp.com/blog.docker.com/wp-content/uploads/f5755314-2f6c-46b3-a2e3-c3a6c4e17108.jpg?fit=668%2C598&ssl=1>”. Accessed on July 2019.”
- [7] “Weavenet Network image ”<https://www.weave.works/docs/net/latest/weave-net-overview.png>”. Accessed on July 2019.”
- [8] “Kubernetes architecture image ”https://www.whizlabs.com/blog/wp-content/uploads/2019/05/Kubernetes_Architecture.png”. Accessed on July 2019.”
- [9] “Request management in Kubernetes diagram ”https://www.oreilly.com/library/view/managing-kubernetes/9781492033905/assets/mgk8_0401.png”. Accessed on July 2019.”
- [10] “Kompose architecture image ”https://jaxenter.de/wp-content/uploads/2017/08/kompose_architecture.png”. Accessed on July 2019.”
- [11] “Persistent volume architecture image ”https://vocon-it.com/wp-content/uploads/2018/12/2018-12-02-23_05_06-Minikube-and-Kubeadm-Google-Pr%C3%A4sentationen-1024x478.png”. Accessed on July 2019.”
- [12] “Official website of Docker ”<https://docs.docker.com/>”. Accessed on July 2019.”
- [13] “Official website of Kubernetes ”<https://kubernetes.io/docs/home/>”. Accessed on July 2019.”
- [14] “Prerequisites for installing Kubernetes ”<https://www.edureka.co/blog/install-kubernetes-on-ubuntu>”. Accessed on July 2019.”
- [15] “Setting up and hosting the dashboard ”<https://www.vikki.in/kubernetes-on-ubuntu-18-04-with-dashbaord/>”. Accessed on July 2019.”
- [16] “Kubernetes management ”<https://www.oreilly.com/library/view/managing-kubernetes/9781492033905/ch04.html>”. Accessed on July 2019.”
- [17] “github link to kompose files ”https://github.com/fresearchgroup/Configurable-and-Scalable-IITBombayX-MOOC-platform-on-Commodity-Servers/tree/master/kompose_files”. Accessed on July 2019.”
- [18] “github link to host volumes ”https://github.com/fresearchgroup/Configurable-and-Scalable-IITBombayX-MOOC-platform-on-Commodity-Servers/tree/master/host_volumes”. Accessed on July 2019.”
- [19] “Pods are unable to access internet ”<https://blog.yaakov.online/kubernetes-getting-pods-to-talk-to-the-internet/>”. Accessed on July 2019.”
- [20] B. Erinle, *JMeter cookbook*. Packt Publishing., 2014, 1st Edition.