

Adv devops 3

Aim: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.

Theory:

Container-based microservices architectures have profoundly changed the way development and operations teams test and deploy modern software. Containers help companies modernize by making it easier to scale and deploy applications, but containers have also introduced new challenges and more complexity by creating an entirely new infrastructure ecosystem.

Large and small software companies alike are now deploying thousands of container instances daily, and that's a complexity of scale they have to manage. So how do they do it?

Enter the age of Kubernetes.

Originally developed by Google, Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. In fact, Kubernetes has established itself as the defacto standard for container orchestration and is the flagship project of the Cloud Native Computing Foundation (CNCF), backed by key players like Google, AWS, Microsoft, IBM, Intel, Cisco, and Red Hat.

Kubernetes makes it easy to deploy and operate applications in a microservice architecture. It does so by creating an abstraction layer on top of a group of hosts so that development teams can deploy their applications and let Kubernetes manage the following activities:

- Controlling resource consumption by application or team
- Evenly spreading application load across a hosting infrastructure
- Automatically load balancing requests across the different instances of an application
- Monitoring resource consumption and resource limits to automatically stop applications from consuming too many resources and restarting the applications again
- Moving an application instance from one host to another if there is a shortage of resources in a host, or if the host dies
- Automatically leveraging additional resources made available when a new host is added to the cluster
- Easily performing canary deployments and rollbacks

Step 1:

Create 3 EC2 Ubuntu Instances on AWS.

(Name 1 as Master, the other 2 as worker-1 and worker-2)

The screenshot shows the AWS Management Console interface for selecting an Amazon Machine Image (AMI). The 'Ubuntu' AMI is selected, showing details for 'Ubuntu Server 20.04 LTS (HVM), SSD Volume Type'. Below this, a table lists the created EC2 instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Publ
master	i-0ff94ac6a4dc32299	Running	t2.micro	Initializing	No alarms	ap-south-1a	ec2-
worker-1	i-040936b2c64e0fbb9	Running	t2.micro	Initializing	No alarms	ap-south-1a	ec2-
worker-2	i-09ca1aa1e4d9366e2	Pending	t2.micro	-	No alarms	ap-south-1a	ec2-

Below the table, a terminal window shows the SSH command and the Ubuntu login prompt:

```
> ssh -i devops-505.cer ubuntu@43.205.138.156
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-1017-aws x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
```

Step 2:

SSH into all 3 machines

The screenshot shows a terminal window with the SSH command and the Ubuntu login prompt:

```
> ssh -i devops-505.cer ubuntu@13.233.212.95
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-1017-aws x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage
```

Step 3:

From now on, until mentioned, perform these steps on all 3 machines.

Install Docker `curl -fsSL https://download.docker.com/linux/ubuntu/gpg |`

`sudo apt-key add sudo add-apt-repository "deb [arch=amd64]`

`https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"`

`sudo apt-get update sudo apt-get install -y docker-ce`

```
Last login: Sat Aug 27 09:36:23 2022 from 103.206.186.99
ubuntu@ip-172-31-39-102:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
OK
ubuntu@ip-172-31-39-102:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
Get:1 https://download.docker.com/linux/ubuntu focal InRelease [57.7 kB]
Get:2 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages [17.9 kB]
Get:3 https://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Hit:4 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Hit:5 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:6 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease
Fetched 189 kB in 1s (250 kB/s)
Reading package lists... Done
```

```
ubuntu@ip-172-31-39-102:~$ cd /etc/docker/
ubuntu@ip-172-31-39-102:/etc/docker$ cat <<EOF | sudo tee /etc/docker/daemon.json
> {
>   "exec-opts": ["native.cgroupdriver=systemd"]
> }
> EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"]
}
ubuntu@ip-172-31-39-102:/etc/docker$ sudo systemctl enable docker
Synchronizing state of docker.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker
ubuntu@ip-172-31-39-102:/etc/docker$ sudo systemctl daemon-reload
ubuntu@ip-172-31-39-102:/etc/docker$ sudo systemctl restart docker
ubuntu@ip-172-31-39-102:/etc/docker$
```

Then, configure cgroup in a daemon.json file.

`cd /etc/docker cat <<EOF | sudo tee /etc/docker/daemon.json`

`{`

`"exec-opts": ["native.cgroupdriver=systemd"]`

`}`

`EOF`

`sudo systemctl enable docker`

`sudo systemctl daemon-reload`

`sudo systemctl restart docker`

Step 4:

Install Kubernetes on all 3 machines `curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg`

`sudo apt-key add cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list deb https://apt.kubernetes.io/ kubernetes-xenial main EOF`

`sudo apt-get update`

`sudo apt-get install -y kubelet kubeadm kubectl`

```
ubuntu@ip-172-31-40-210:/etc/docker$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add
OK
ubuntu@ip-172-31-40-210:/etc/docker$ sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
Hit:1 https://download.docker.com/linux/ubuntu focal InRelease
Hit:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Get:4 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease
Get:6 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:3 https://packages.cloud.google.com/apt kubernetes-xenial InRelease [9383 B]
Get:7 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 Packages [58.4 kB]
Fetched 290 kB in 1s (241 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-40-210:/etc/docker$
```

After installing Kubernetes, we need to configure internet options to allow bridging. `sudo swapoff -a echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf sudo sysctl -p`

Step 5:

Perform this ONLY on the Master machine

Initialize the Kubecluster `sudo kubeadm init --podnetwork-cidr=10.244.0.0/16`

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

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

Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.105.48.149:6443 --token 8hbt0h.h47eeuh4jdnbb0yz \
--discovery-token-ca-cert-hash sha256:17c178ba997d99c7c0d440df5bd7722bb743f3b5a3c8e5a234b2ac8d48d5c097
root@localhost:/etc/docker#
```

`kubeadm join 172.105.48.149:6443 --token mwjrs9.n9m68s5x1a3r0dka --discovery-token-ca-cert-hash`

`sha256:94174c906096688053622fcc60e48b71b934a4074ce79c5f2334f69016ae9a19`

Copy the join command and keep it in a notepad, we'll need it later.

Copy the mkdir and chown commands from the top and execute them

Then, add a common networking plugin called flannel file as mentioned in the code kubectl apply -f

<https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>

```
root@localhost:~# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
```

Check the created pod using this command

Now, keep a watch on all nodes using the following command watch kubectl get nodes **Step 6:**

Perform this ONLY on the worker machines

sudo kubeadm join --token \ --discovery-token-ca-cert-hash

```
root@master:~# kubectl get nodes
NAME          STATUS    ROLES          AGE      VERSION
master        Ready     control-plane   3m58s    v1.25.0
root@master:~#
```

Step 7:

Now, notice the changes on the master terminal

```
root@master:~# kubectl get nodes
NAME          STATUS    ROLES          AGE      VERSION
localhost     Ready     <none>          8m27s    v1.25.0
master        Ready     control-plane   122m     v1.25.0
```

That's it, we now have a Kubernetes cluster running across 3 AWS EC2 Instances. This cluster can be used to further deploy applications and their loads being distributed across these machines

Conclusion:

In this experiment, we learned how to install Kubernetes create a Kubernetes Cluster in AWS EC2 instances and get them up and running.