

Name : Nidhi Rijhwani

Subject : Adv Devops Exp No 03

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

Theory:

Kubernetes is a powerful container orchestration platform that automates the deployment, scaling, and management of containerized applications. Its architecture consists of a Control Plane (including the API server, controller manager, scheduler, etc) that manages the cluster state, and worker nodes that run applications in pods via agents like kubelet and kube-proxy. To spin up a Kubernetes cluster on cloud platforms like AWS, Google Cloud, or Azure, you typically use managed services like EKS, GKE, or AKS, where you can create clusters through their respective management consoles or command-line tools, ensuring seamless scalability and management of your containerized applications.

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.

Steps:

1. Create 3 EC2 Ubuntu Instances on AWS.

(Name 1 as Master, 2nd as Worker1, 3rd as Worker2)

The screenshot shows the AWS Management Console interface for launching an EC2 instance. The breadcrumb navigation at the top indicates the path: EC2 > Instances > Launch an instance. The main heading is 'Launch an instance' with an 'Info' link. Below the heading, a brief description states: 'Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.' The 'Name and tags' section is active, showing a text input field with the name 'Master' and an 'Add additional tags' button. On the right, the 'Summary' section is expanded, displaying the following configuration: 'Number of instances' is set to 1; 'Software Image (AMI)' is 'Amazon Linux 2023.5.2' with a 'read more' link; 'Virtual server type (instance type)' is 't2.micro'; and 'Firewall (security group)' is 'New security group'.

ch

[Alt+S]

N. Virginia

voclabs/user3402666=d2022.niddhi.rijhwani@ves.ac.in @ 7369

Instances (3) Info

Last updated less than a minute ago

Connect

Instance state

Actions

Launch instances

Find Instance by attribute or tag (case-sensitive)

All states

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability
<input type="checkbox"/>	Master	i-0983647a4f901d302	Running	t2.micro	Initializing	View alarms +	us-east-1d
<input type="checkbox"/>	Worker2	i-0ff7c56de01abc4be	Running	t2.micro	Initializing	View alarms +	us-east-1d
<input type="checkbox"/>	Worker1	i-0de8cab99c709e023	Running	t2.micro	Initializing	View alarms +	us-east-1d

2. Edit the Security Group Inbound Rules to allow SSH

EC2 > Security Groups > sg-0fee21dba433d32c2 - launch-wizard-1 > Edit inbound rules

Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules Info

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	Description - optional Info
sg-r-02280ef21f8ba9a15	SSH	TCP	22	Cust... 0.0.0.0/0	

Add rule

Delete

EC2 > Instances > i-0ff7c56de01abc4be > Connect to instance

Connect to instance Info

Connect to your instance i-0ff7c56de01abc4be (Worker2) using any of these options

EC2 Instance Connect

Session Manager

SSH client

EC2 serial console

⚠

Port 22 (SSH) is open to all IPv4 addresses
 Port 22 (SSH) is currently open to all IPv4 addresses, indicated by **0.0.0.0/0** in the inbound rule in [your security group](#). For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: 18.206.107.24/29. [Learn more.](#)

Instance ID

i-0ff7c56de01abc4be (Worker2)

Connection Type

☒ **Connect using EC2 Instance Connect**
 Connect using the EC2 Instance Connect browser-based client, with a public IPv4 address.

☐ **Connect using EC2 Instance Connect Endpoint**
 Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IP address

3. Install Docker

Add the missing GPG key:

First, manually add the missing GPG key using the following command:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
7EA0A9C3F273FCD8
```

Alternatively, if apt-key is deprecated and not working, you can use gpg to fetch and add the key directly:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg
--dearmor | sudo tee /usr/share/keyrings/docker-archive-keyring.gpg >
/dev/null
```

Update the Docker repository configuration:

Ensure your Docker repository configuration is using the correct keyring file. The repository configuration should include the signed-by option pointing to the keyring file:

```
echo "deb [arch=amd64
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" |
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Update your package index and install Docker:

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Then, configure cgroup in a daemon.json file.

```
cd /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts":
  ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
```

Install Kubernetes on all 3 machines

Download the GPG key and save it to a file:

```
curl -fsSL
https://packages.cloud.google.com/apt/doc/apt-key.gpg | gpg
--dearmor | sudo tee /usr/share/keyrings/cloud-google.gpg >
/dev/null
```

Add the Google Cloud repository to your sources list using the signed-by option:

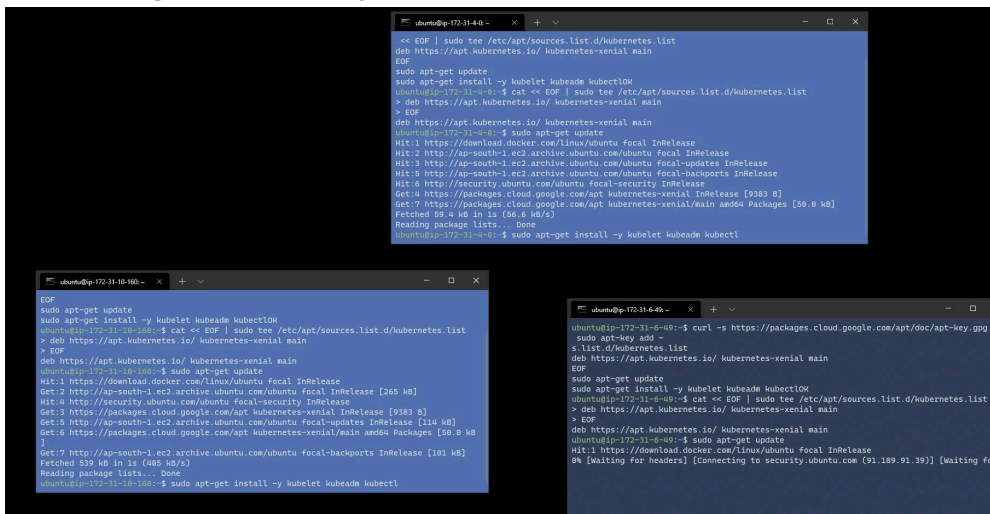
Create a new file for the Google Cloud repository or update an existing one. For example, you can create a file called `/etc/apt/sources.list.d/google-cloud.list`:

```
echo "deb [arch=amd64
signed-by=/usr/share/keyrings/cloud-google.gpg]
https://packages.cloud.google.com/apt cloud-sdk main" | sudo
tee /etc/apt/sources.list.d/google-cloud.list > /dev/null
```

Update your package index and install the desired packages:

```
sudo apt-get update
sudo apt-get install google-cloud-sdk
```

```
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-~$ cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
ubuntu@ip-172-31-40-~$ sudo apt-get update
Hit:1 https://download.docker.com/linux/ubuntu focal InRelease
Hit:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Hit:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:4 http://security.ubuntu.com/ubuntu focal-security InRelease
Get:5 https://packages.cloud.google.com/apt kubernetes-xenial InRelease [1983 B]
Get:6 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 Packages [50.0 kB]
Fetched 50.0 kB in 1s (50.0 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-40-~$ sudo apt-get install -y kubelet kubeadm kubectl
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are now recommended:
  kubeadm kubelet
The following NEW packages will be installed:
  kubelet kubeadm kubectl
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 50.0 kB of archives.
After this operation, 198 kB of additional disk space will be used.
Get:1 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubeadm amd64 1.10.0-0ubuntu1 [50.0 kB]
Get:2 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubelet amd64 1.10.0-0ubuntu1 [50.0 kB]
Get:3 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubectl amd64 1.10.0-0ubuntu1 [50.0 kB]
Fetched 50.0 kB in 1s (50.0 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Setting up kubeadm (1.10.0-0ubuntu1) ...
Setting up kubelet (1.10.0-0ubuntu1) ...
Setting up kubectl (1.10.0-0ubuntu1) ...
Processing triggers for libc-bin (2.27-0ubuntu1) ...
ubuntu@ip-172-31-40-~$
```

```
ubuntu@ip-172-31-40-~$ cat << EOF | sudo tee /etc/apt/sources.list.d/google-cloud.list
deb [arch=amd64
signed-by=/usr/share/keyrings/cloud-google.gpg]
https://packages.cloud.google.com/apt cloud-sdk main
EOF
ubuntu@ip-172-31-40-~$ sudo apt-get update
Hit:1 https://download.docker.com/linux/ubuntu focal InRelease
Hit:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Hit:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:4 http://security.ubuntu.com/ubuntu focal-security InRelease
Get:5 https://packages.cloud.google.com/apt kubernetes-xenial InRelease [1983 B]
Get:6 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 Packages [50.0 kB]
Fetched 50.0 kB in 1s (50.0 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-40-~$ sudo apt-get install -y kubelet kubeadm kubectl
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are now recommended:
  kubeadm kubelet
The following NEW packages will be installed:
  kubelet kubeadm kubectl
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 50.0 kB of archives.
After this operation, 198 kB of additional disk space will be used.
Get:1 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubeadm amd64 1.10.0-0ubuntu1 [50.0 kB]
Get:2 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubelet amd64 1.10.0-0ubuntu1 [50.0 kB]
Get:3 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubectl amd64 1.10.0-0ubuntu1 [50.0 kB]
Fetched 50.0 kB in 1s (50.0 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Setting up kubeadm (1.10.0-0ubuntu1) ...
Setting up kubelet (1.10.0-0ubuntu1) ...
Setting up kubectl (1.10.0-0ubuntu1) ...
Processing triggers for libc-bin (2.27-0ubuntu1) ...
ubuntu@ip-172-31-40-~$
```

```
ubuntu@ip-172-31-40-~$ curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | gpg
--dearmor | sudo tee /usr/share/keyrings/cloud-google.gpg > /dev/null
ubuntu@ip-172-31-40-~$
```

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
```

4. Perform this ONLY on the Master Machine

Initialize the Kubecluster

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all
```

```
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 --token <token> --discovery-token-ca-cert-hash sha256:ab3fb9f05059f2f4829be48caa1830254e6c49218c9aeffc4
```

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.

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

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>

```
ubuntu@ip-172-31-4-0:/etc/docker$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

Check the created pod using this command.

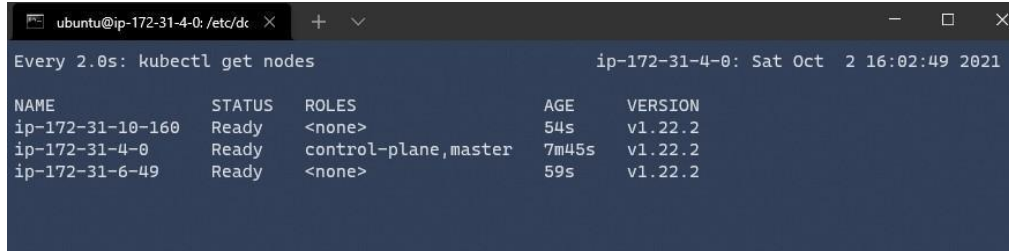
Now, keep a watch on all nodes using the following command

- watch kubectl get nodes

5. Perform this ONLY on the worker machines

`sudo kubeadm join <ip> --token <token> \ --discovery-token-ca-cert-hash <hash>`

Now, notice the changes on the master terminal

A terminal window titled 'ubuntu@ip-172-31-4-0: /etc/d...' shows the command 'Every 2.0s: kubectl get nodes' being executed. The output is a table with 5 columns: NAME, STATUS, ROLES, AGE, and VERSION. It lists three nodes: 'ip-172-31-10-160' (Ready, <none>, 54s, v1.22.2), 'ip-172-31-4-0' (Ready, control-plane, master, 7m45s, v1.22.2), and 'ip-172-31-6-49' (Ready, <none>, 59s, v1.22.2). The terminal window also shows the date and time 'ip-172-31-4-0: Sat Oct 2 16:02:49 2021' in the top right corner.

NAME	STATUS	ROLES	AGE	VERSION
ip-172-31-10-160	Ready	<none>	54s	v1.22.2
ip-172-31-4-0	Ready	control-plane, master	7m45s	v1.22.2
ip-172-31-6-49	Ready	<none>	59s	v1.22.2

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.