

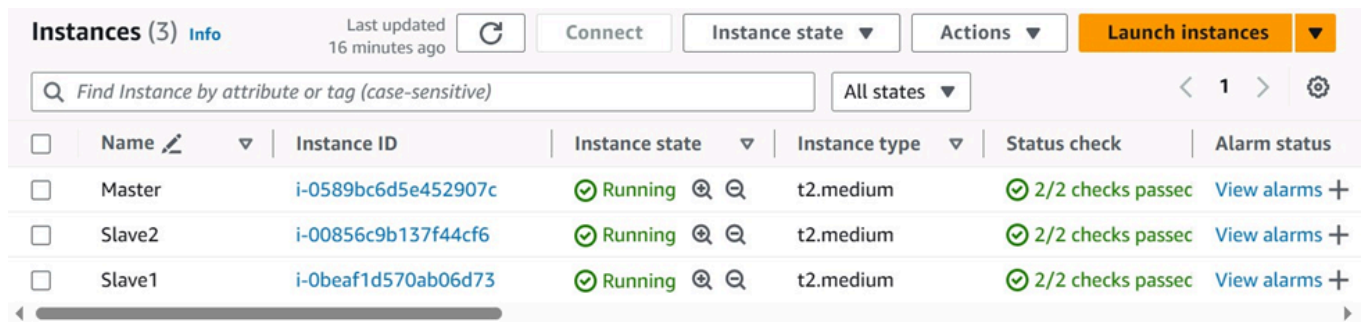
Advanced DevOps Lab

Experiment:3

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

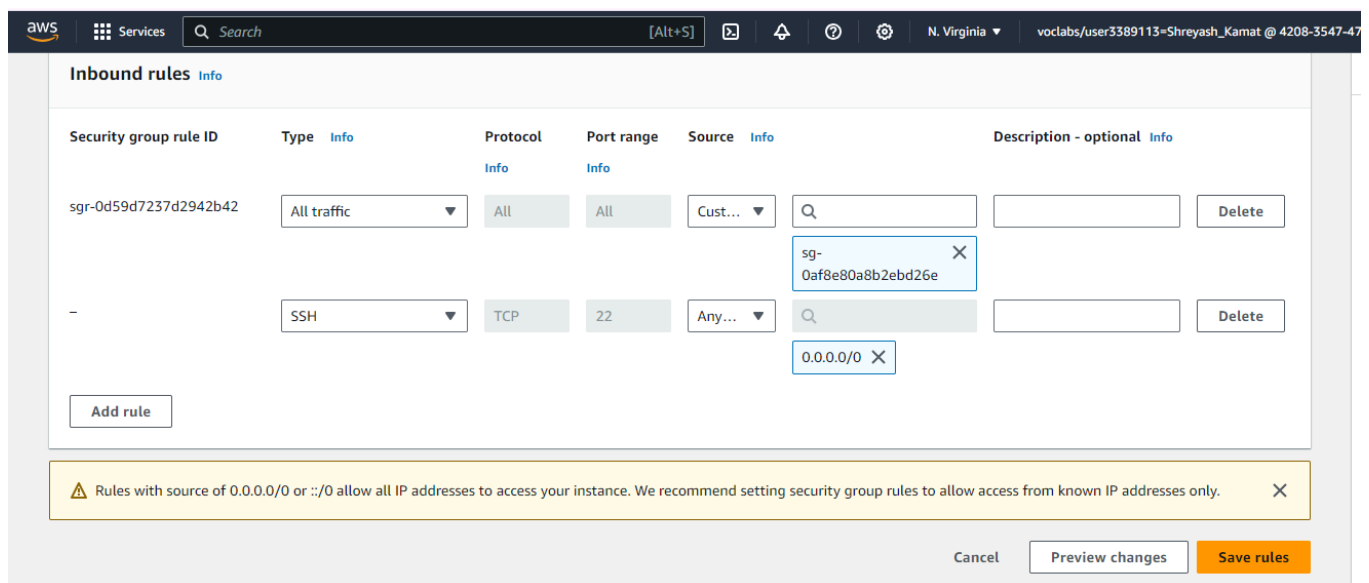
Reference: <https://www.youtube.com/watch?v=Cz7hSJNq2GU>

Step 1: : Create 3 EC2 instances (1 master and 2 slaves). Select SSH option in the inbound rules. Created a key pair to be used commonly between all 3 instances created. I selected AWS Linux as my operating system and enabled t2 medium option for kubernetes cluster to run smoothly



Instances (3) Info		Last updated 16 minutes ago	Connect	Instance state ▼	Actions ▼	Launch instances ▼
Find Instance by attribute or tag (case-sensitive)		All states ▼	< 1 > ⚙			
<input type="checkbox"/>	Name ↗	Instance ID	Instance state ▼	Instance type ▼	Status check	Alarm status
<input type="checkbox"/>	Master	i-0589bc6d5e452907c	Running	t2.medium	2/2 checks passed	View alarms +
<input type="checkbox"/>	Slave2	i-00856c9b137f44cf6	Running	t2.medium	2/2 checks passed	View alarms +
<input type="checkbox"/>	Slave1	i-0beaf1d570ab06d73	Running	t2.medium	2/2 checks passed	View alarms +

2. Edit the Security Group Inbound Rules to allow SSH



Security group rule ID	Type	Protocol	Port range	Source	Description - optional	
sgr-0d59d7237d2942b42	All traffic	All	All	Cust...		Delete
-	SSH	TCP	22	Any...		Delete

Add rule

Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Cancel Preview changes Save rules

3. SSH into all 3 machines

```
ssh -i <keyname>.pem ubuntu@<public_ip_address>
```

```
quantum@machine ~/Downloads ssh -i "ec-2-ubuntu.pem" ec2-user@ec2-3-88-111-183.compute-1.amazonaws.com
The authenticity of host 'ec2-3-88-111-183.compute-1.amazonaws.com (3.88.111.183)' can't be established.
ED25519 key fingerprint is SHA256:pQu+xs9foYbY3de1twjZcVVA0zmGwGv6PHmVruF/Q1s.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-88-111-183.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

#_
~\_ #####_ Amazon Linux 2023
~~ \_#####\
~~ \###|
~~ \#/ --- https://aws.amazon.com/linux/amazon-linux-2023
~~ V~' '->
~~~
~~ ._. /
  _/_/_/
    _/m/'
```

4. 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
```

```
[ec2-user@ip-172-31-22-31 ~]$ yum install docker -y
Error: This command has to be run with superuser privileges (under the root user on most systems).
[ec2-user@ip-172-31-22-31 ~]$ sudo su
[root@ip-172-31-22-31 ec2-user]# yum install docker -y
Last metadata expiration check: 0:03:24 ago on Sat Sep 14 14:54:57 2024.
Dependencies resolved.

=====
Package                                Architecture      Version            Repository          Size
=====
Installing:
docker                                x86_64            25.0.6-1.amzn2023.0.2  amazonlinux        44 M
Installing dependencies:
containerd                            x86_64            1.7.20-1.amzn2023.0.1  amazonlinux        35 M
iptables-libs                         x86_64            1.8.8-3.amzn2023.0.2  amazonlinux        401 k
iptables-nft                         x86_64            1.8.8-3.amzn2023.0.2  amazonlinux        183 k
libcgroup                             x86_64            3.0-1.amzn2023.0.1    amazonlinux        75 k
libnetfilter_conntrack               x86_64            1.0.8-2.amzn2023.0.2  amazonlinux        58 k
libnftlink                           x86_64            1.0.1-19.amzn2023.0.2 amazonlinux        30 k
libnftnl                             x86_64            1.2.2-2.amzn2023.0.2  amazonlinux        84 k
pigz                                 x86_64            2.5-1.amzn2023.0.3    amazonlinux        83 k
runc                                 x86_64            1.1.13-1.amzn2023.0.1  amazonlinux        3.2 M
=====

Transaction Summary

Install 10 Packages

Total download size: 84 M
Installed size: 317 M
```

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

```

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

```

```

[root@ip-172-31-22-31 ec2-user]# # Set SELinux in permissive mode (effectively disabling it)
sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
[root@ip-172-31-22-31 ec2-user]# # This overwrites any existing configuration in /etc/yum.repos.d/kubernetes.repo
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
[root@ip-172-31-22-31 ec2-user]# sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
Kubernetes
Dependencies resolved.
48 kB/s | 9.4 kB    00:00
=====
Package                Architecture      Version           Repository        Size
=====
Installing:
kubeadm                x86_64            1.31.1-150500.1.1 kubernetes        11 M

```

```

Installing      : conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64      6/9
Running scriptlet: conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64      6/9
Installing      : kubelet-1.31.1-150500.1.1.x86_64              7/9
Running scriptlet: kubelet-1.31.1-150500.1.1.x86_64              7/9
Installing      : kubeadm-1.31.1-150500.1.1.x86_64              8/9
Installing      : kubect1-1.31.1-150500.1.1.x86_64              9/9
Running scriptlet: kubect1-1.31.1-150500.1.1.x86_64              9/9
Verifying       : conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64      1/9
Verifying       : libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64  2/9
Verifying       : libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64  3/9
Verifying       : libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64    4/9
Verifying       : cri-tools-1.31.1-150500.1.1.x86_64             5/9
Verifying       : kubeadm-1.31.1-150500.1.1.x86_64             6/9
Verifying       : kubect1-1.31.1-150500.1.1.x86_64             7/9
Verifying       : kubelet-1.31.1-150500.1.1.x86_64             8/9
Verifying       : kubernetes-cni-1.5.1-150500.1.1.x86_64        9/9

Installed:
conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64      cri-tools-1.31.1-150500.1.1.x86_64
kubeadm-1.31.1-150500.1.1.x86_64                kubect1-1.31.1-150500.1.1.x86_64
kubelet-1.31.1-150500.1.1.x86_64                kubernetes-cni-1.5.1-150500.1.1.x86_64
libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64  libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64
libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64

Complete!
[root@ip-172-31-22-31 ec2-user]# sudo systemctl enable --now kubelet
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service -> /usr/lib/systemd/system/kubelet.service.
[root@ip-172-31-22-31 ec2-user]#

```

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

5. Perform this **ONLY** on the Master machine

Run command ...kubeadm init with the proper network pod, here it is --pod-network-cidr=10.244.0.0/16 to initialize kubernetes

```

[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

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 "kubect1 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 172.31.22.31:6443 --token plmkzy.2ocxer4410uwlqk4 \
--discovery-token-ca-cert-hash sha256:2590fd7ba571e7e92b4f18f77c2149583f19f6049e3dfb4d306ac22cf2f465d6
[root@ip-172-31-22-31 ec2-user]#

```

We are supposed to add a networking plugin named flannel with the help of the command mentioned in the console output i.e kubect1 apply -f

```
[ec2-user@ip-172-31-81-63 docker]$ 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
```

By running kubectl get nodes, we get to see the nodes that are currently connected to the master node

6. Perform this **ONLY on the worker machines**

Run the following commands to ensure a smooth and secure joining to the master node

Paste the below command on all 2 worker machines

- sudo yum install iproute-tc-y
- sudo systemctl enable kubelet
- sudo systemctl restart kubelet

Then we are supposed to run the join command that was generated in the console output of our master machine

kubeadm join 172.31.22.31:6443 --token gyakv9.hktjpt5usstl5u3y \ --discovery-token-ca-cert-hash sha256:2590fd7ba571e7e92b4f18f77c2149583f19f6049e3dfb4d306ac22cf2f465d6

```
[root@ip-172-31-23-217 ec2-user]# kubeadm join 172.31.22.31:6443 --token gyakv9.hktjpt5usstl5u3y \
--discovery-token-ca-cert-hash sha256:2590fd7ba571e7e92b4f18f77c2149583f19f6049e3dfb4d306ac22cf2f465d6
[preflight] Running pre-flight checks
```

Post which we are supposed to get the output that our worker nodes have been successfully connected to master node.

Unfortunately, on running the join command i was not able to produce anything beyond 'Running pre-flight checks' which can be seen in the above image

And thus, could not execute the last step of this experiment

Conclusion:In this experiment, we set up a connection between a local machine and an EC2 instance using SSH. After facing issues like timeouts and permission problems, we learned how to check for common causes such as incorrect security group settings, improper key permissions, and network issues. By resolving these, we successfully connected to the EC2 instance. This experiment helped us understand the steps required for remote server access and troubleshooting.