Advance DevOps

Experiment 3

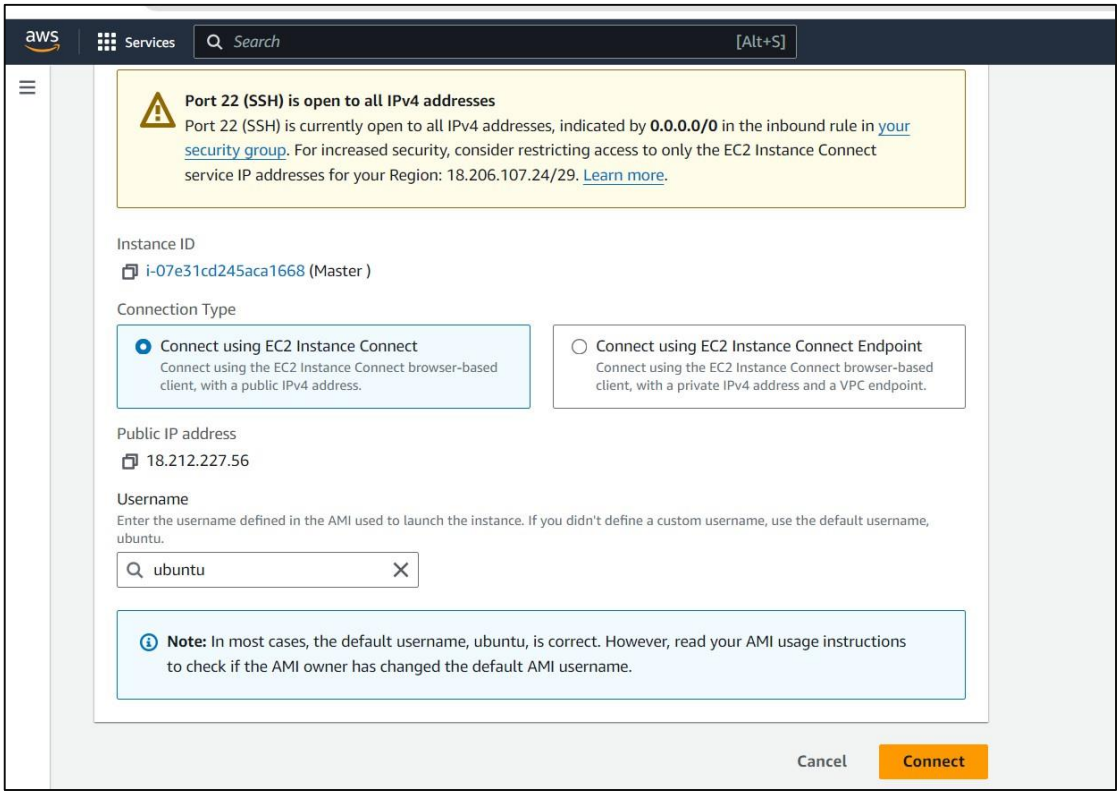
Aim: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.

Steps:

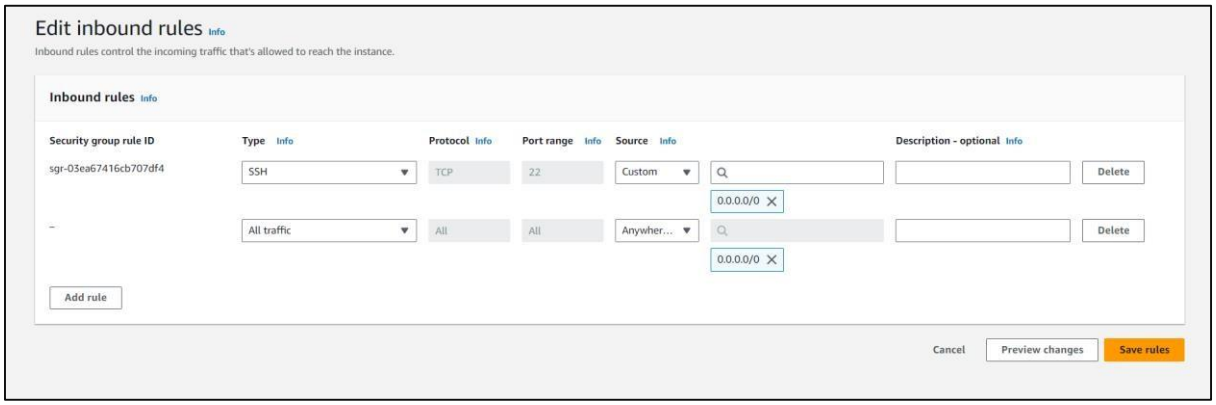
1. We will create 3 EC2 instances. One will be the master node and the other 2 will be slave/worker nodes.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
master	i-057a73de5d3df0746	Running	t2.micro	2/2 checks passec	View alarms
slave node 1	i-01ea28da4403edd84	Running	t2.micro	2/2 checks passec	View alarms
slave node 2	i-0a47253bfd474144d	Running	t2.micro	2/2 checks passec	View alarms

2. After the instances have been created, we will connect them one by one.

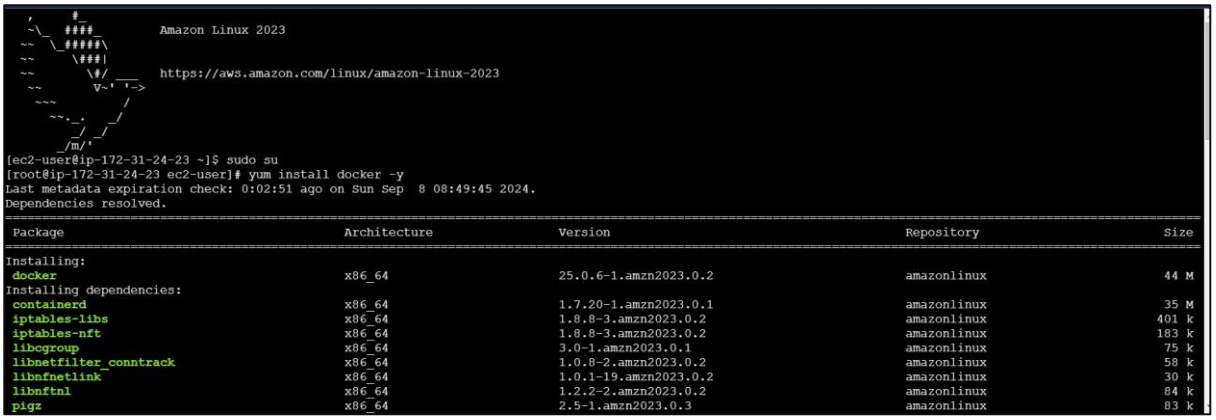


3. Edit the inbound rule to allow all traffic

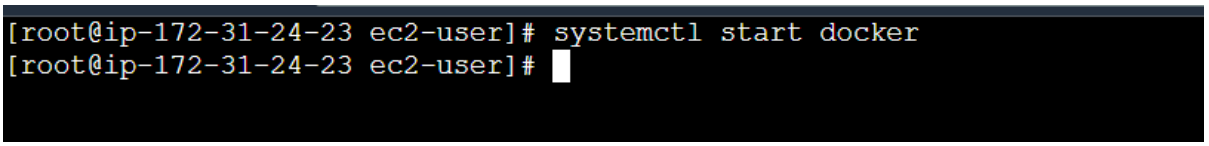


4. Docker installation:

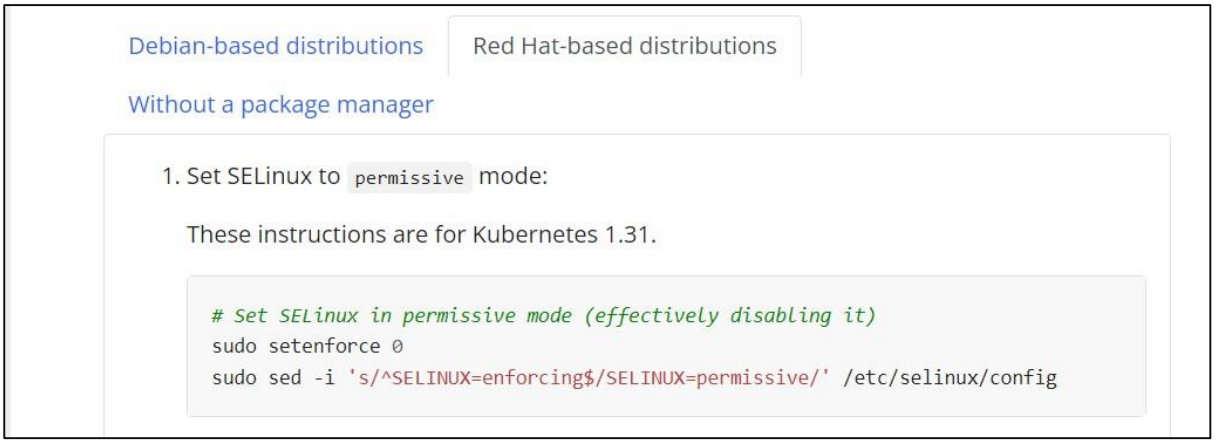
   This step has to be performed on all the 3 instances.
   The following command has to be run:
   yum install docker -y



5. After successfully docker has been installed it has to be started on all machines by using the command “systemctl start docker”



6. Kubernetes installation:
   Search kubeadm installation on your browser and scroll down and select red hat-based distributions.



```
# 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
```

3. Install kubelet, kubeadm and kubectl:

```
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
```

4. (Optional) Enable the kubelet service before running kubeadm:

```
sudo systemctl enable --now kubelet
```

Copy the above given steps and paste in the terminal. This will create a Kubernetes repository, install kubelet, kubeadm and kubectl and also enable the services.

```
  Verifying        : libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64
  Verifying        : cri-tools-1.31.1-150500.1.1.x86_64
  Verifying        : kubeadm-1.31.1-150500.1.1.x86_64
  Verifying        : kubectl-1.31.1-150500.1.1.x86_64
  Verifying        : kubelet-1.31.1-150500.1.1.x86_64
  Verifying        : kubernetes-cni-1.5.1-150500.1.1.x86_64

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                             kubectl-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-172-31-18-228 ec2-user]# yum repolist
repo id                                          repo name
amazonlinux                                      Amazon Linux 2023 repository
kernel-livepatch                                 Amazon Linux 2023 Kernel Livepatch repository
kubernetes                                       Kubernetes
[root@ip-172-31-18-228 ec2-user]#
```

7. We can check if repository has been created by using yum repolist command

```
[root@ip-172-31-24-23 ec2-user]# yum repolist
repo id                                                    repo name
amazonlinux                                                Amazon Linux 2023 repository
kernel-livepatch                                           Amazon Linux 2023 Kernel Livepatch repository
kubernetes                                                 Kubernetes
[root@ip-172-31-24-23 ec2-user]#
```

8. Now we will be initializing the kubeadm. For that “kubeadm init” command has to be used. It may show errors but those can be ignored by using --ignore-preflight-errors=all

```
[root@ip-172-31-24-23 ec2-user]# kubeadm init
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
        [WARNING FileExisting-tc]: tc not found in system path
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR NumCPU]: the number of available CPUs 1 is less than the required 2
        [ERROR Mem]: the system RAM (949 MB) is less than the minimum 1700 MB
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-preflight-errors=...`
To see the stack trace of this error execute with --v=5 or higher
[root@ip-172-31-24-23 ec2-user]# ^C
[root@ip-172-31-24-23 ec2-user]# kubeadm init --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
        [WARNING NumCPU]: the number of available CPUs 1 is less than the required 2
        [WARNING Mem]: the system RAM (949 MB) is less than the minimum 1700 MB
        [WARNING FileExisting-tc]: tc not found in system path
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W0908 09:11:46.376173   28683 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runtime is inconsistent with that used b
y kubeadm.It is recommended to use "registry.k8s.io/pause:3.10" as the CRI sandbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-24-23.ec2.internal kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.c
luster.local] and IPs [10.96.0.1 172.31.24.23]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
```

```
[root@ip-172-31-82-191 ec2-user]# kubeadm init --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
W0913 15:43:09.666583    2255 checks.go:1080] [preflight] WARNING: Couldn't create the interface used for talking to the container runtime: failed to create new C
RI runtime service: validate service connection: validate CRI v1 runtime API for endpoint "unix:///var/run/containerd/containerd.sock": rpc error: code = Unavaila
ble desc = connection error: desc = "transport: Error while dialing: dial unix /var/run/containerd/containerd.sock: connect: no such file or directory"
        [WARNING FileExisting-socat]: socat not found in system path
        [WARNING FileExisting-tc]: tc not found in system path
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR FileContent--proc-sys-net-ipv4-ip_forward]: /proc/sys/net/ipv4/ip_forward contents are not set to 1
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-preflight-errors=...`
To see the stack trace of this error execute with --v=5 or higher
[root@ip-172-31-82-191 ec2-user]# ^C
[root@ip-172-31-82-191 ec2-user]# systemctl start docker
[root@ip-172-31-82-191 ec2-user]# kubeadm init --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
        [WARNING FileExisting-socat]: socat not found in system path
        [WARNING FileExisting-tc]: tc not found in system path
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W0913 15:44:18.815161    2565 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runtime is inconsistent with that used b
y kubeadm.It is recommended to use "registry.k8s.io/pause:3.10" as the CRI sandbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-82-191.ec2.internal kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.
```

9. On successful initialization we need to copy and paste the following commands on the master machine itself:

```
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
```

10. Next copy and paste the join link in the worker nodes so that the worker nodes can join the cluster.

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

kubeadm join 172.31.82.191:6443 --token 8450pt.tdprcovwa61rqyo1 \
        --discovery-token-ca-cert-hash sha256:b11f191f3df19a2e9112a5c19b4461bffeaddd8b5be8625ad8451019aecc043c
```

11. We can check the nodes that have joined the cluster using kubectl get nodes. Right now, there is only one node which is the master node.

```
[root@ip-172-31-85-89 ec2-user]# kubectl get nodes
NAME                          STATUS     ROLES           AGE   VERSION
ip-172-31-85-89.ec2.internal   NotReady   control-plane   72s   v1.26.0
```

12. After performing join commands on the worker nodes, we will get following output:

```
This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

Once again when you run kubectl get nodes you will now see all 3 nodes have joined the cluster:

```
[root@ip-172-31-34-212 ec2-user]# kubectl get nodes
NAME                           STATUS   ROLES           AGE   VERSION
ip-172-31-34-212.ec2.internal   Ready    control-plane   18m   v1.31.1
ip-172-31-37-229.ec2.internal   Ready    <none>          13m   v1.31.1
ip-172-31-45-98.ec2.internal    Ready    <none>          14m   v1.31.1
[root@ip-172-31-34-212 ec2-user]#
```

Conclusion:

This experiment enabled the creation of a Kubernetes cluster and the successful joining of all 3 nodes using various commands. Errors during initialization can be handled in two ways: 1. By ignoring the errors, or 2. By changing the instance type to t3.medium or t3.large if the issue is related to insufficient memory space or CPU resources. Also, it is to be ensured that the inbound rules and outbound rules allow all traffic or else it leads to connectivity issues between master node and worker nodes.