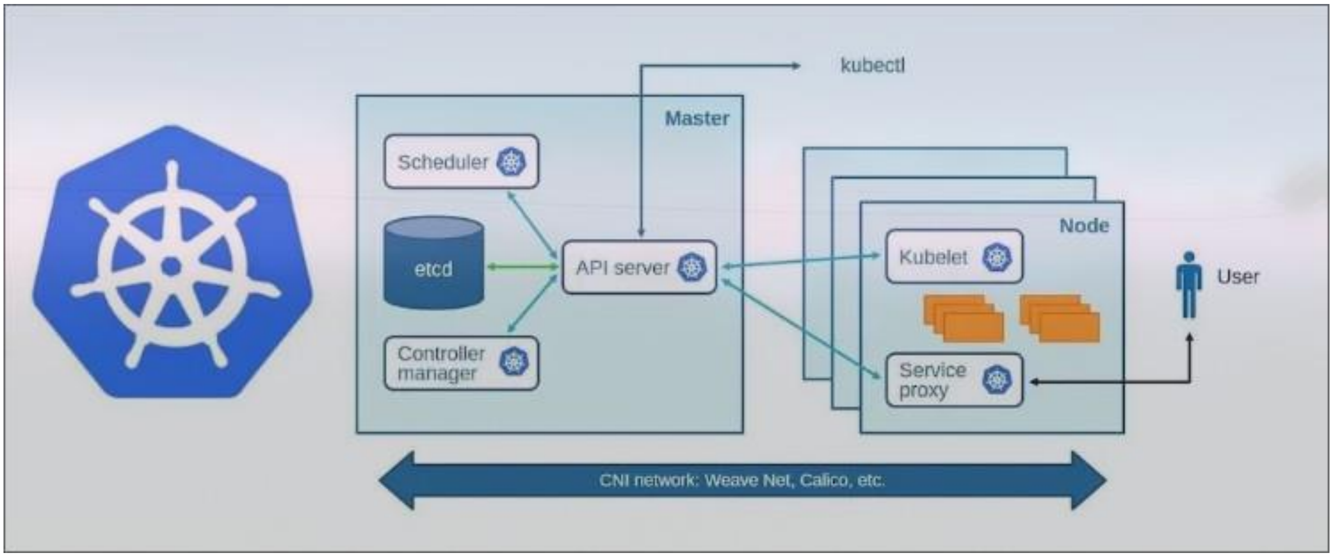
# EXPERIMENT - 3

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

## Theory:

### Kubernetes

Kubernetes is an open source platform for managing container technologies such as Docker. Docker lets you create containers for a pre-configured image and application. Kubernetes provides the next step, allowing you to balance loads between containers and run multiple containers across multiple systems.



### Kubernetes Architecture:

Kubernetes is consists of two nodes master & worker, Nodes are the physical or virtual machines that are used to run pods. There can be only one master and multiple worker nodes.

- Master (Control plane)

Master node often reffered to as control plane and responsible for managing and orchestrating the overall operations of the system, It serves as central control point of cluster. It interact with worker node to deploy pods.

Components:

- Schedhuler: schedule worker node for running pods.
- Controller manager: The main function is to maintain desired state of cluster. Checks what the workers are doing and they are up.
- API server: Directly communicated with worker from master and vice versa.
- etcd: Store state of kubernetes cluster in key-value data store.
- Kubectl: Kubectl is a command-line tool used to communicate with a Kubernetes cluster's control plane using the API server. Allows to run commands to deploy application, inspect and manage resources.
- CNI: Container network interface.
- Pod: This host and manage our containers that run our application.

- Worker:

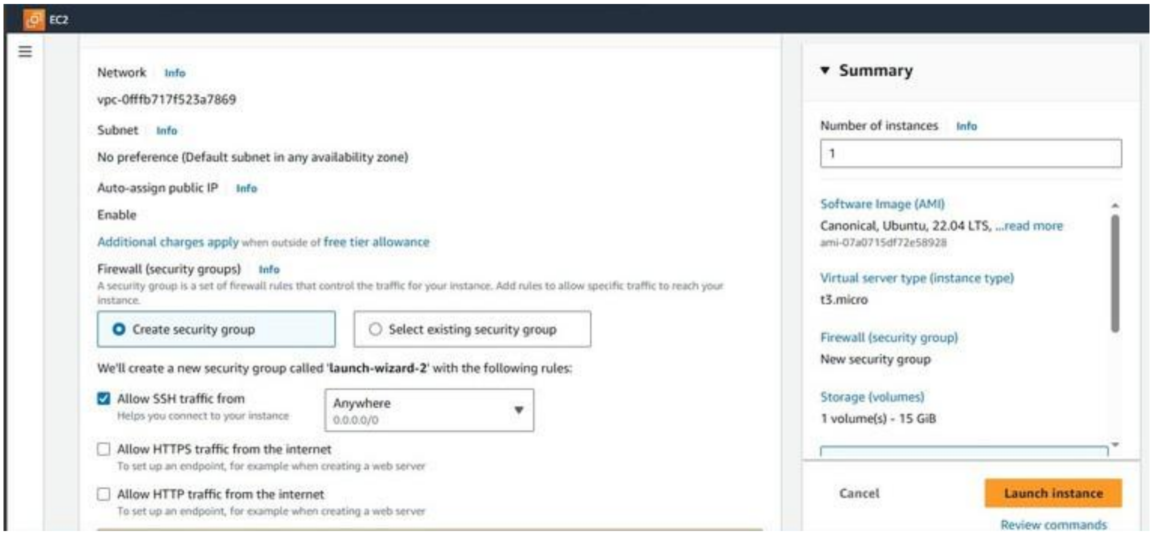
Machine which runs containers and workloads. This is where the actual application is running.

Components:

- kubelet: Stay in worker node, Manage containers and ensures they're running as expected. It communicates with the API server to receive information about the pods that are assigned to the node.
- service proxy : It maintains some network rules which determines how traffic is allowed to and from the Pods. It also Allow users/clients to access application.

Steps:

1. Start by creating two Aws EC2 instance for worker & master node



Network
vpc-0fffb717f523a7869

Subnet
No preference (Default subnet in any availability zone)

Auto-assign public IP
Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups)
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

● Create security group ○ Select existing security group

We'll create a new security group called 'launch-wizard-2' with the following rules:

☑ Allow SSH traffic from — Anywhere 0.0.0.0/0
Helps you connect to your instance

□ Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server

□ Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server

▾ Summary

Number of instances: 1

Software Image (AMI)
Canonical, Ubuntu, 22.04 LTS, ...read more
ami-07a0715df72e58928

Virtual server type (instance type)
t3.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 15 GiB

2. After creating servers execute commands according on master and worker:

- Install Dependencies *# Execute On both the nodes (master and worker)*

```
~$ sudo apt update

~$ sudo apt-get install -y apt-transport-https ca-certificates curl

~$ sudo apt install docker.io -y

~$ sudo systemctl enable --now docker
```

- Install kubeadm # Execute On both the nodes (master and worker)

```
~$ echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https:-/pkgs.k8s.io/core:/stable:/v1.28/deb/ /" | sudo tee
/etc/apt/sources.list.d/kubernetes.list

~$ curl -fsSL https:-/pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -
o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
~$ sudo apt update

~$ sudo apt install kubeadm kubectl kubelet -y
```

- Now, initialize kubeadm (kubernetes) in Master node.

```
~$ sudo kudeadm init
```

- Setup local kubeconfig # Execute on master node.

```
~$ mkdir -p $HOME/.kube

~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

- Apply weave network # On both

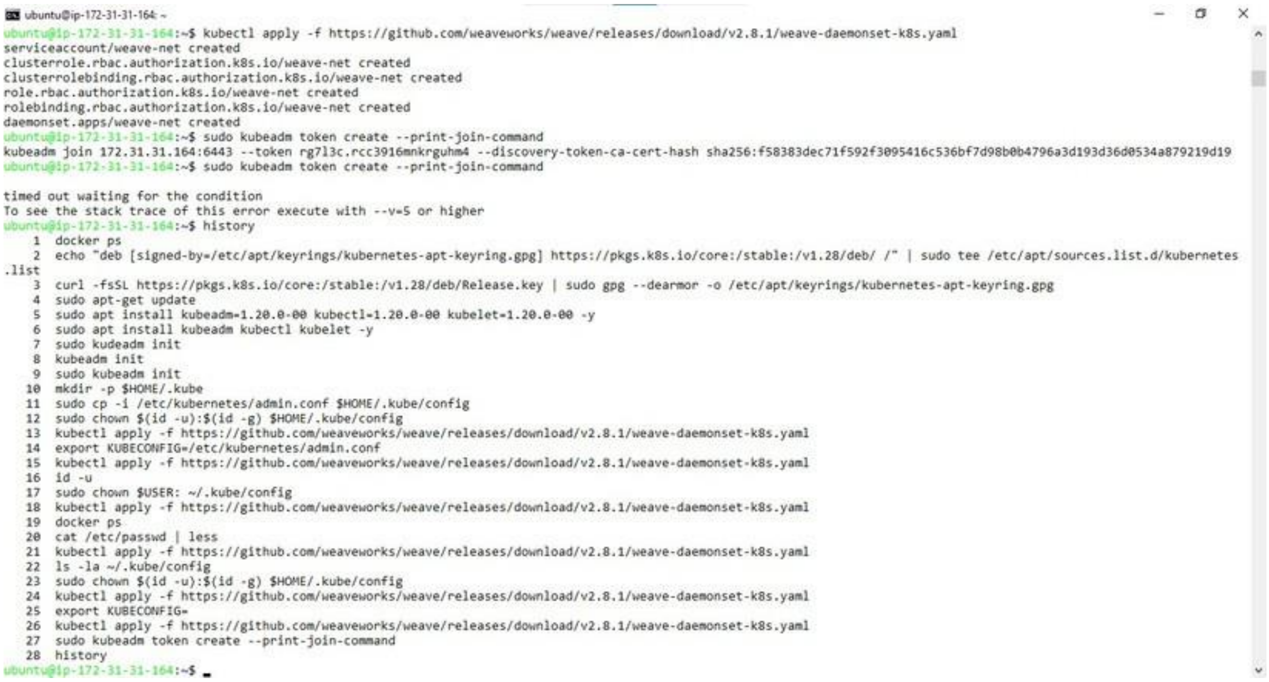
```
~$ kubectl apply -f https:-/github.com/weaveworks/weave/releases/download/v2.8.1/weave-
daemonset-k8s.yaml
```

- Generate token for worker node to join # On master

```
~$ sudo kubeadm token create –print-join-command
```

- Now, paste the token (output) of 6th step, also append --v=5 at end.

*~$ sudo kubeadm join 172.31.61.228:6443 --token f4mesu.7rzk86ga48n3uydh --discovery-token-ca-cert-hash  sha256:66c9863913ffdb50316e82b74f3703f73e42c4210d3a01ec7afcdbc01f677eec  –v=5*

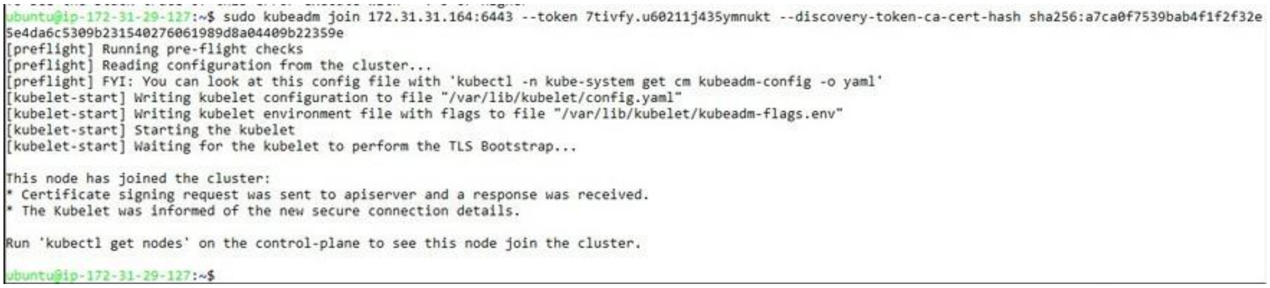


```
ubuntu@ip-172-31-31-164:~$ kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml
serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.apps/weave-net created
ubuntu@ip-172-31-31-164:~$ sudo kubeadm token create --print-join-command
kubeadm join 172.31.31.164:6443 --token rg7l3c.rcc3916mnkrguhm4 --discovery-token-ca-cert-hash sha256:f58383dec71f592f3095416c536bf7d98b0b4796a3d193d36d0534a879219d19
ubuntu@ip-172-31-31-164:~$ sudo kubeadm token create --print-join-command

timed out waiting for the condition
To see the stack trace of this error execute with --v=5 or higher
ubuntu@ip-172-31-31-164:~$ history
    1  docker ps
    2  echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes
.list
    3  curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
    4  sudo apt-get update
    5  sudo apt install kubeadm=1.20.0-00 kubectl=1.20.0-00 kubelet=1.20.0-00 -y
    6  sudo apt install kubeadm kubectl kubelet -y
    7  sudo kudeadm init
    8  kubeadm init
    9  sudo kubeadm init
   10  mkdir -p $HOME/.kube
   11  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
   12  sudo chown $(id -u):$(id -g) $HOME/.kube/config
   13  kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml
   14  export KUBECONFIG=/etc/kubernetes/admin.conf
   15  kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml
   16  id -u
   17  sudo chown $USER: ~/.kube/config
   18  kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml
   19  docker ps
   20  cat /etc/passwd | less
   21  kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml
   22  ls -la ~/.kube/config
   23  sudo chown $(id -u):$(id -g) $HOME/.kube/config
   24  kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml
   25  export KUBECONFIG=
   26  kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml
   27  sudo kubeadm token create --print-join-command
   28  history
ubuntu@ip-172-31-31-164:~$ _
```

- Verify you worker node connection by running this command on Master node.

*~$ kubectl get pods*



```
ubuntu@ip-172-31-29-127:~$ sudo kubeadm join 172.31.31.164:6443 --token 7tivfy.u60211j435ymnukt --discovery-token-ca-cert-hash sha256:a7ca0f7539bab4f1f2f32e
5e4da6c5309b231540276061989d8a04409b22359e
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

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.

ubuntu@ip-172-31-29-127:~$
```

Conclusion: Thus we successfully understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.