KUBERNETES

# **Kubernetes creation**

**What is Kubernetes?**

Kubernetes is an open-source container management (orchestration) tool. Its container management responsibilities include container deployment, scaling & descaling of containers & container load balancing.

Kubernetes is a platform that eliminates the manual processes involved in deploying containerized applications.

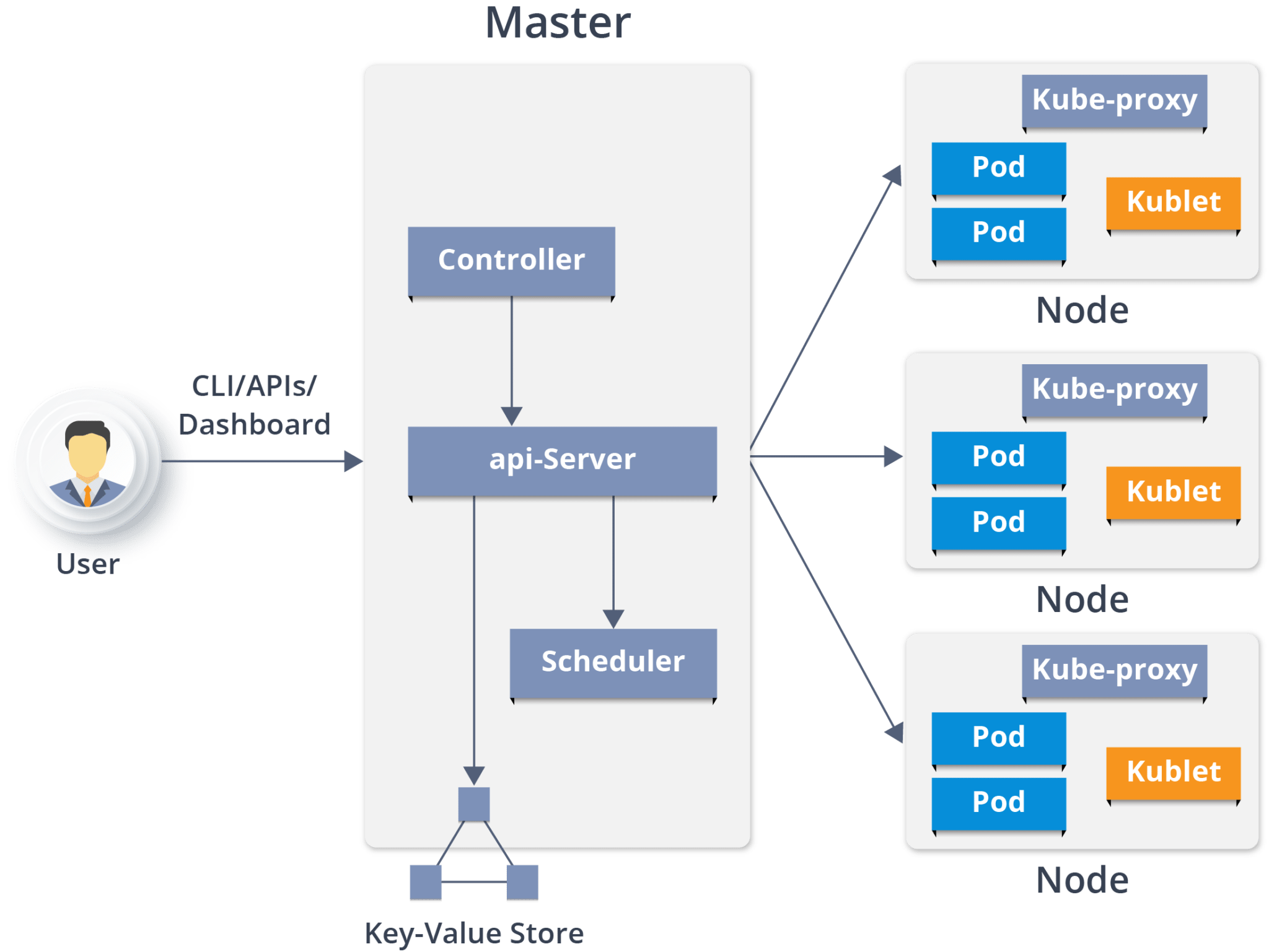


**Why Kubernetes?**

Containers are a good way to bundle and run our applications. In a production environment, we need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to restart. It would be easier if this behavior was handled by a system.

That’s how Kubernetes comes into picture! Kubernetes provides us with a framework to run distributed systems resiliently. It takes care of scaling requirements, failover, deployment patterns, and more.

## ****Kubernetes Architecture:****



## ****Master Node:****

It is the entry point for all administrative tasks which is responsible for managing the Kubernetes cluster.

**API Server:**

* Performs all the administrative tasks through the API server within the master node.
* In this REST commands are sent to the API server which validates and processes the requests.
* After requesting, the resulting state of the cluster is stored in the distributed key-value store.

**ETCD:**

It is a distributed reliable key-value store. Kubernetes store the data in etcd to manage the cluster. In Kubernetes we have master and worker nodes, how many master and worker nodes we have all that information is stored in etcd**.**

**Scheduler:**

The scheduler schedules the tasks to slave nodes. It stores the resource usage information for each slave node. It schedules the work in the form of Pods and Services.

**Controller manager:**

Basically called as Controller. A controller watches the desired state of the objects, it manages and watches their current state through the API server. If the current state of the objects it manages does not meet the desired state, then the control loop takes corrective steps to make sure that the current state is the same as the desired state.

## ****Worker Node:****

It is a physical server which runs the applications using Pods (**a pod scheduling unit**) which is controlled by the master node.

**Kubelet**:

It is an agent which communicates with the Master node and executes on nodes or the worker nodes. It gets the Pod specifications through the API server and executes the containers associated with the Pod and ensures that the containers described in those Pod are running and healthy.

**Kube-proxy:**

Kube-proxy runs on each node to deal with individual host sub-netting and ensure that the services are available to external parties.

### **Pods:**

A pod is one or more containers that logically go together. Pods run on nodes. Pods run together as a logical unit. One node can run multiple pods.

**Kubernetes features:**

### **1. Automatic Binpacking**

Kubernetes automatically packages application and schedules the containers based on their requirements and available resources while not sacrificing availability. To ensure complete utilization and save unused resources, Kubernetes balances between critical and best-effort workloads.

### **2. Service Discovery & Load balancing**

Kubernetes will automatically assign IP addresses to containers and a single DNS name for a set of containers that can load-balance traffic inside the cluster.

### **3. Storage Orchestration**

With Kubernetes, we can mount the storage system of our choice. We can either opt for local storage, or choose a public cloud provider such as AWS, or perhaps use a shared network storage system

### **4. Self-Healing**

Kubernetes can automatically restart containers that fail during execution and kills those containers that don’t respond to user-defined health checks. But if nodes itself die, then it replaces and reschedules those failed containers on other available nodes.

### **5. Secret & Configuration Management**

Kubernetes can help us deploy and update secrets and application configuration without rebuilding your image and without exposing secrets in the stack configuration.

### **6. Batch Execution**

Kubernetes manage batch and CI workloads, thus replacing containers that fail, if desired.

### **7. Horizontal Scaling**

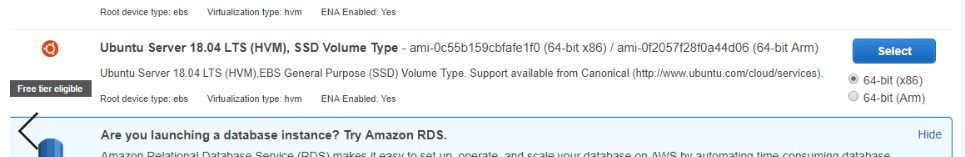
Kubernetes needs only 1 command to scale up the containers, or to scale them down when using the CLI.

### **8. Automatic Rollbacks & Rollouts**

Kubernetes progressively rolls out changes and updates application or its configuration, by ensuring that not all instances are worked at the same instance. Even if something goes wrong, Kubernetes will roll back the change.

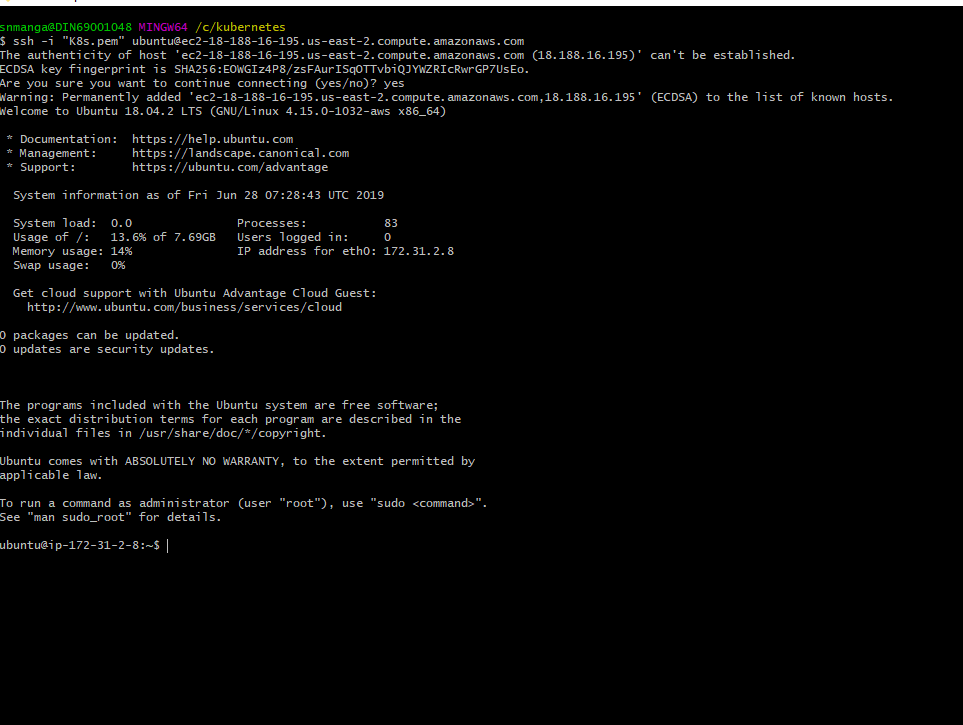
**Step-Up Kubernetes (k8s) Cluster on AWS:**

1. Create Ubuntu EC2 Instance (Because to setup kubernetes we need a Linux system)



Select EC2 🡪 Instance Launch 🡪 Ubuntu Server 🡪 Choose an Instance Type of t2.micro 🡪 Click Next 🡪 Add Storage 🡪 Add Tags Give Key & Value names 🡪 Configure Security Group 🡪 Review

After instance launched we will connect to the Ubuntu instance by using gitbash.

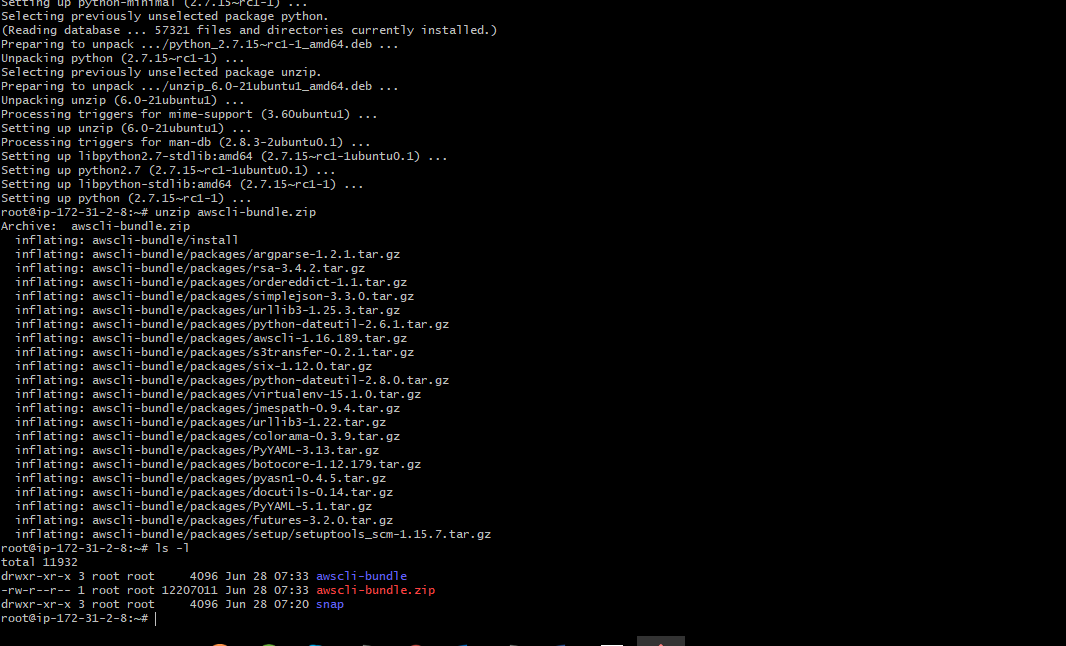


After connected to the Ubuntu server connect as root user using following command

sudo su –

1. Install AWSCLI

* curl https://s3.amazonaws.com/aws-cli/awscli-bundle.zip -o awscli-bundle.zip
* apt install unzip python
* unzip awscli-bundle.zip
* #sudo apt-get install unzip - if you dont have unzip in your system
* ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws

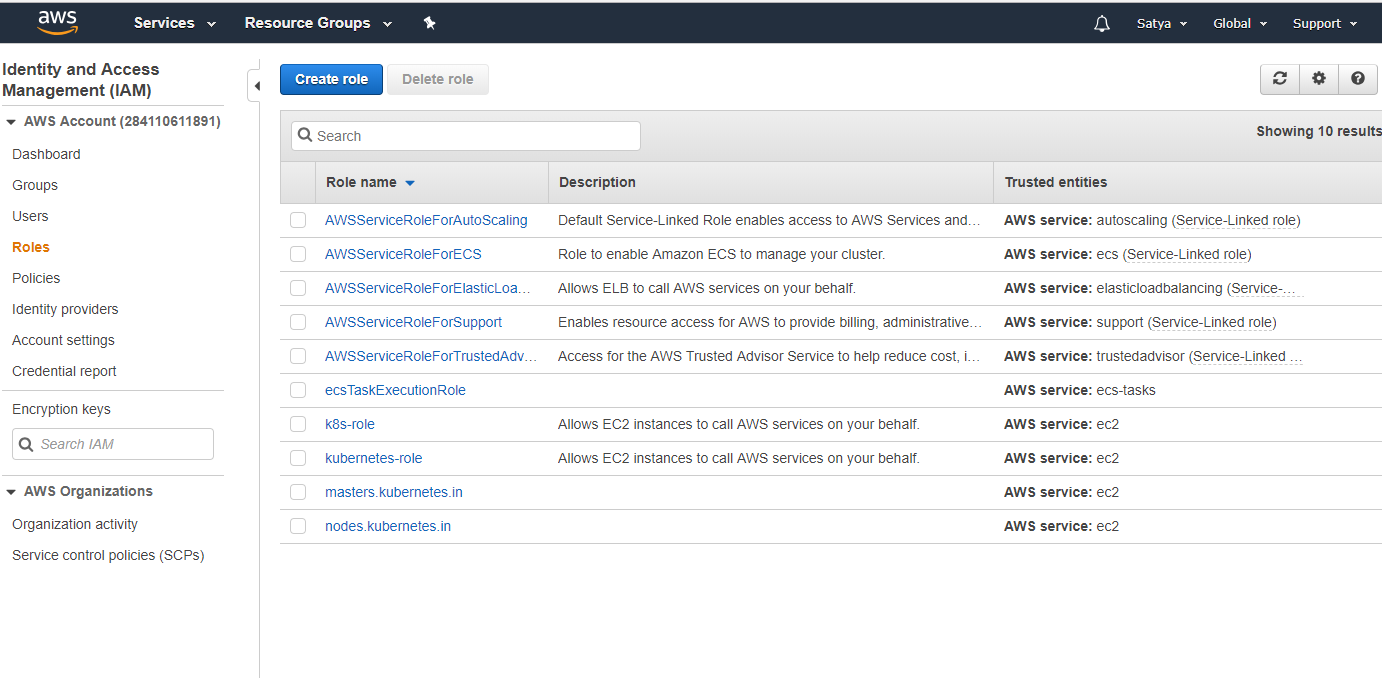


1. Install kubectl

* curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
* chmod +x ./kubectl
* sudo mv ./kubectl /usr/local/bin/kubectl

1. Create an IAM user/role with Route53, EC2, IAM and S3 full access

Services 🡪 IAM 🡪 Roles->create a Role 🡪 Ec2 🡪 click on Next: permissions 🡪 search for s3full access🡪 ec2 full access🡪 route53fullacess and IAM full access🡪 click on NextTag 🡪 Give key as name and value as kubernetes-role 🡪 Click on review 🡪 confirm name and click on create role.



1. Attach IAM role to ubuntu server

Go to instance and select🡪 Action🡪 Instance settings🡪 add or replace instance🡪 select our IAM role🡪 click on apply

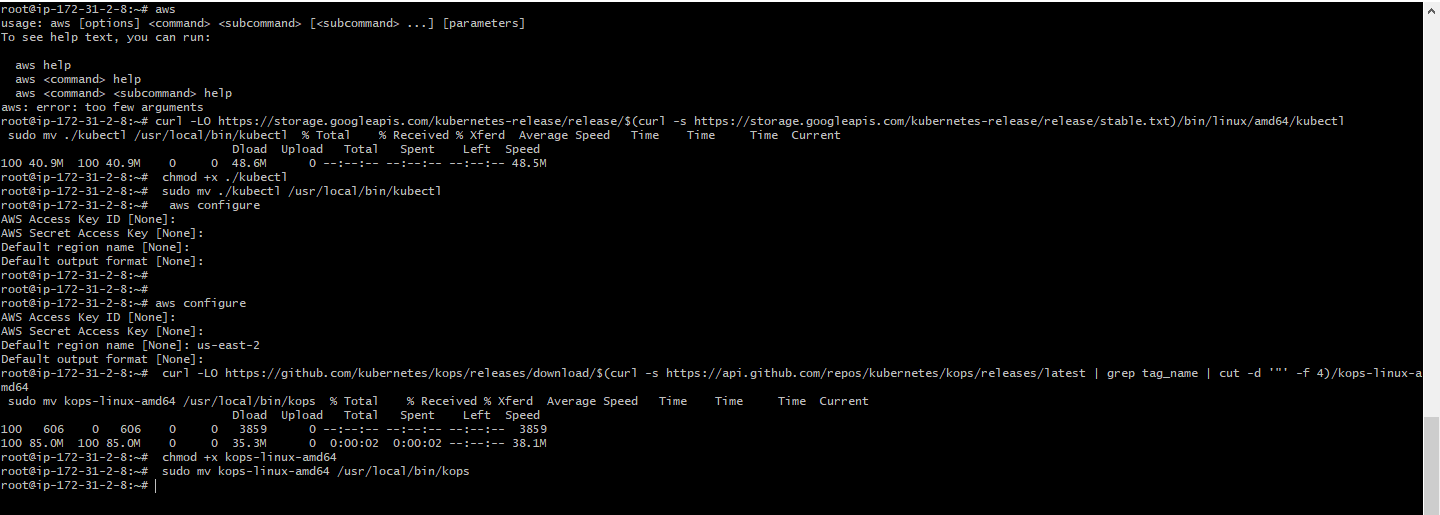
aws configure

1. Install kops on ubuntu instance:

curl -LO https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag\_name | cut -d '"' -f 4)/kops-linux-amd64

chmod +x kops-linux-amd64

sudo mv kops-linux-amd64 /usr/local/bin/kops



1. Create a Route53 private hosted zone (you can create Public hosted zone if you have a domain)

Services 🡪 Route53 🡪 Hosted Zones 🡪 Domain Name 🡪 Type: Private 🡪 Select VPC ID according to region 🡪 click on create

1. Create an S3 bucket

aws s3 mb s3://kube.in

1. Expose environment variable:

export KOPS\_STATE\_STORE=s3://kube.in

1. Create sshkeys before creating cluster

ssh-keygen

1. Create kubernetes cluster definitions on S3 bucket

kops create cluster --cloud=aws --zones=us-east-2a --name=kube.in --dns-zone=kube.in --dns private

1. Create kubernetes cluster

kops update cluster kube.in –yes

Go to AWS EC2 and check the instances are created.

1. Validate your cluster

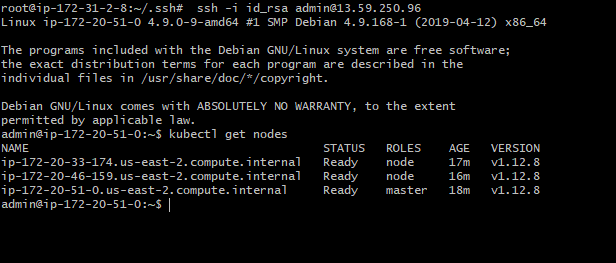
kops validate cluster

To connect to the Master Node use the private key by following commands

ssh -i id\_rsa 13.59.250.96

1. To list nodes

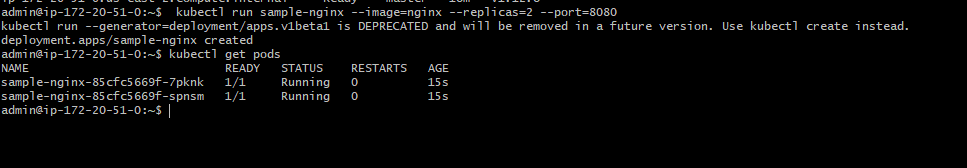
kubectl get nodes



#### Deploying Nginx container on Kubernetes:

1. Deploying Nginx Container
2. kubectl run sample-nginx --image=nginx --replicas=2 --port=8080
3. kubectl get pods

kubectl get deployments



Expose the deployment as service. This will create an ELB in front of those 2 containers and allow us to publicly access them:

kubectl expose deployment sample-nginx --port=8080 --type=LoadBalancer

kubectl get services -o wide

