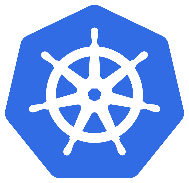
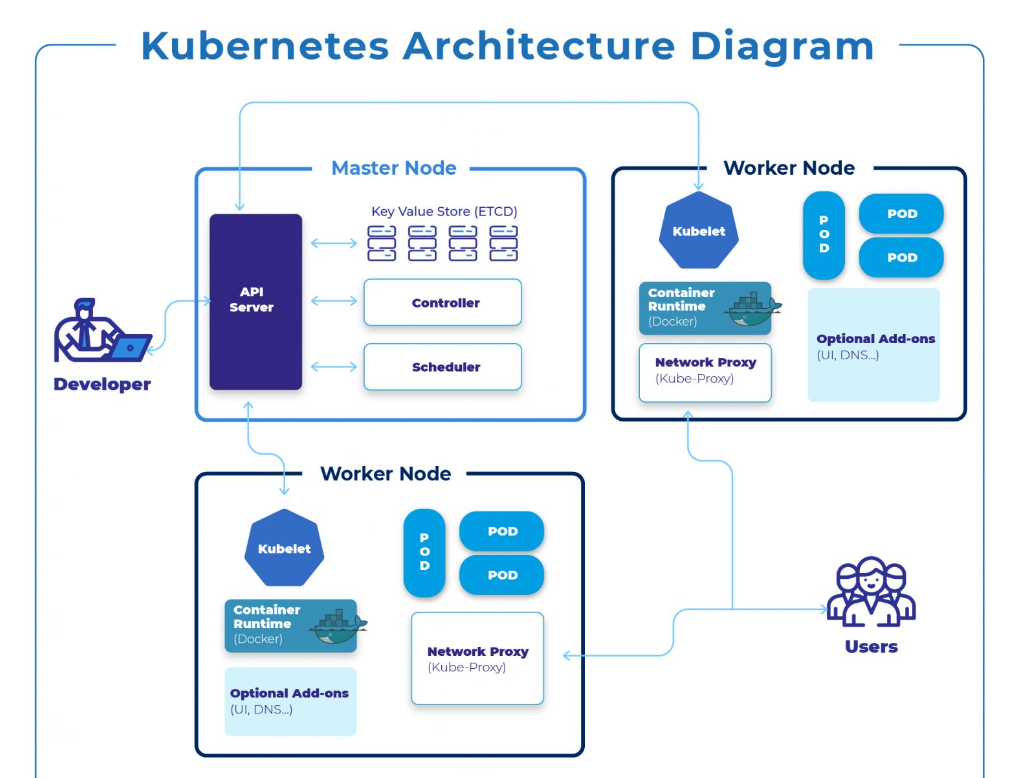
* Explain Kubernetes architecture and components in detail



|]020

.lm\612

Client server mode 🡪 Master node and Worker node consists of 8 components

* **Master** : Master Node is the starting point for all administrative tasks, and its responsibility is managing the Kubernetes cluster architecture

4 components below listed :

**API server** : A user sends the rest commands as YAML/JSON format to the API server, then it processes and executes them. The Kube API-server is the front end of the Kubernetes control plane

**etcd** : distributed key-value store Kubernetes file in a database called the etcd Meta data

**scheduler** : used to schedule the work to different worker nodes also manages the new requests coming from the API Server and assigns them to healthy nodes

**Control manager** : runs continuously and watches/monitor the actual and desired state of objects, create, delete, recreate the objects and nodes

* **Worker :** Worker Node runs apps via Pods, and Master Node controls the Pods. Pods are scheduled on a physical server (slave node). So, when you want to access the apps from an external environment, you’ll have to connect to these nodes.

4 components below listed:

**Kubelet** : agent that runs on each worker node and communicates with the master node. It also makes sure that the containers which are part of the pods are always healthy. It watches for tasks sent from the API Server, executes the task like deploy or destroy the container, and then reports back to the Master.

**Kube-proxy :** used to communicate between the multiple worker node, maintains network rules on nodes, so the container can communicate to each in different nodes.

**Kubernetes pod** is a group of one or more containers that are deployed together on the same host. Pod is deployed with a shared storage/network

**Container Runtime** is the software that is responsible for running containers. Kubernetes supports several container runtimes: Docker, containers.

* **Cluster creation in Kubernetes**

Create a EC2 and launch

Copy and drop the below commands in EC2 to install kubectl and kops

**KOPS BINARY SETUP**

# curl -Lo kops 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

# sudo mv ./kops /usr/local/bin/

**KUBECTL BINARY SETUP**

# curl -Lo kubectl 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

Install awscli -y \*to install awscli\*

Aws configure -> access key,secret key,location, format

Kubectl version –short

Input variables in EC2 root edit copy and paste

export AWS\_ACCESS\_KEY\_ID=AKIATLJI2ELYZPKW3LD5

export AWS\_SECRET\_ACCESS\_KEY=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

export NAME=<cluster name>.k8s.local

create a bucket

export KOPS\_STATE\_STORE=s3://<bucket name>

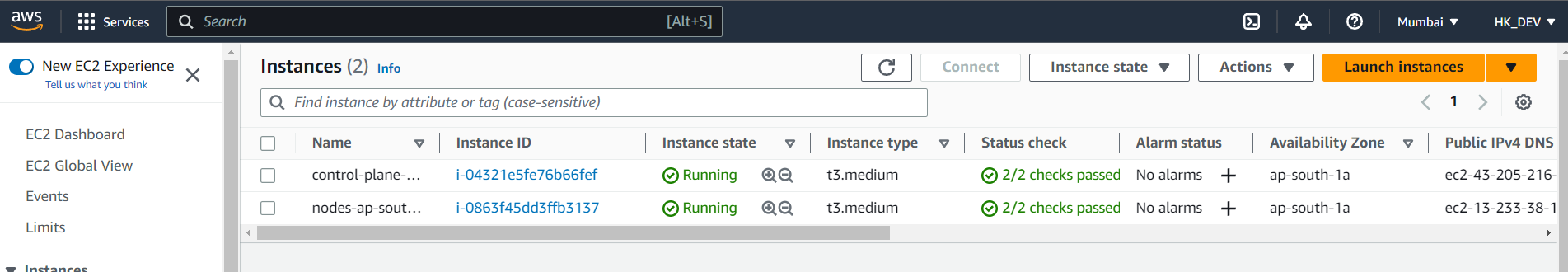
Ssh-keygen 🡪 enter 🡪enter🡪 enter

Create a cluster by cmd below

kops create cluster --zones ap-south-1 ${NAME}

kops update cluster --<cluster name>. k8s.local --yes --admin

Output :



* Launching POD

Write a yaml script and save it <filename>.yml

apiVersion: v1

kind: Pod

metadata:

name: kingfisher

labels:

name: kingfisher

spec:

containers:

- name: kingfisher

image: httpd

resources:

limits:

memory: "128Mi"

cpu: "500m"

ports:

- containerPort: 80

* kubectl apply -f <yaml file name>
* Tools in k8s for cluster creation

**KUBEADM** : cli tool and typically used for setting up clusters on-premises or in cloud environments where you have direct access

**KOPS** : specifically designed for creating, upgrading, and managing Kubernetes clusters on AWS It simplifies the process of provisioning and managing the infrastructure

**Minikube** : sets up a lightweight cluster using a virtual machine (VM) on your local system, allowing you to quickly deploy and test applications without the need for a full-fledged production cluster.

**Kubespray/ Kargo**: highly customizable and allows you to define the desired cluster configuration and infrastructure parameters, it provides a set of Ansible playbooks to provision and configure clusters

* **What are the objects used so far in kubernetes**

**Pod**: The smallest unit of deployment in Kubernetes. A pod represents a one or more instance of a running process or container.

**Deployment**: A higher-level object that manages the creation and scaling of pods. It provides declarative updates and rollbacks, making it easy to manage application deployments.

**Service**: An abstract way to expose an application running on a set of pods. Services enable network access and load balancing to the pods.

**Ingress**: An API object that manages external access to services within a cluster. It acts as a gateway or entry point for HTTP and HTTPS traffic, allowing you to define routing rules, SSL certificates, and other traffic-related configurations.

**ConfigMap**: A configuration object that provides a way to inject configuration data into containers. It can store key-value pairs, files, or even entire directories.

**Secret**: Similar to ConfigMaps, Secrets store sensitive data like passwords, API keys, and TLS certificates. Secrets are base64 encoded and can be mounted into pods as files or exposed as environment variables.

**PersistentVolume** (PV) and **PersistentVolumeClaim** (PVC): PV and PVC are used for persistent storage in Kubernetes. A PV represents a piece of storage provisioned in the cluster, while a PVC is a request for a specific amount and type of storage. PVCs bind to PVs, allowing pods to access persistent storage.

Namespace: A logical separation mechanism that provides scope and isolation within a cluster. Namespaces help organize and manage resources by creating virtual clusters within a physical cluster.

* **What is namespace and namespace syntax**

Namespace is a virtual cluster that provides a way to organize and isolate resources within a Kubernetes cluster

help in partitioning the cluster's resources, allowing multiple teams or projects to share the same cluster while maintaining separation and avoiding naming conflicts.

kubectl create namespace <namespace-name>

kubectl apply -f <config-file> -n <namespace-name>

kubectl delete namespace <namespace-name>

* **What is deployment**

Deployment is an object that defines how to create and manage a group of identical Pods. It ensures that the desired number of Pods are running and handles scaling, rolling updates, and rollback functionality.

* **what is replicaset**

It is responsible for maintaining the desired level of availability and scalability for your application, ensure specified number of identical pods are running at all times

* **What is daemonset**

DaemonSet is a Kubernetes object that ensures that a copy of a Pod runs on every node within a cluster. It is used for deploying system-level daemons or agents on every node, such as logging agents, monitoring agents, or network proxies.

* **Secure secrets**

Use Kubernetes Secrets

Avoid Committing Secrets to Version Control

Use Secrets as Environment Variables or Volumes

Rotate Secrets Regularly

* **What are the services in k8s and types**

ClusterIP

NodePort

Load balancer

External name

Headless services

Ingress

* What is nodeport and range (30000-32767)

nodePort is a service type that exposes a service on a static port on each worker node. It allows external access to the service by mapping a port on the worker nodes to the corresponding port of the service.

* **what is cluster ip**

ClusterIP is a service type that provides internal access to other services within the cluster. It assigns a stable virtual IP (ClusterIP) to the service, allowing other components within the cluster to communicate with the service.

* **What is node balancing**

Distribution of incoming traffic or workload across multiple nodes in a Kubernetes clusterit ensures that the workload is evenly distributed among the available nodes to prevent any single node from becoming overwhelmed with traffic or resource utilization.