Kubernetes

Docker Swarm.	Kubernetes.
(1) Docker Swarm does not manage any Containers but instead is a Cluster manager for Docker Containers.	Kubernetes also has built-in support for Stateful applications. where as Docker Swarm does not :-
a) Docker Swarm only works with Docker Engine API	a) Kubernetes work with any programming language & framework.
b) Swarm runs inside Docker Containers.	b) Kubernetes runs on Top of Linux Containers.
c) Swarm use manager-node & worker node. Manager-node Controls, Scheduled Task, Execution on worker node.	c) Kubernetes uses a master-slave architecture where one master instance Controls multiple worker instance;
	d) Kubernetes uses an active/passive model where each worker instance is Controlled by it own master instance.
(2) Docker Swarm in Docker property. (Orchestrator)	(2) Kubernetes is Google property (Orchestrator)
(3) Docker Swarm is an Open-Source project the aims to provide a Simple, Yet Powerful Cluster manager to run your Containers at Scale. it offers lots of feature like load balancing, service discovery, high availability.	(3) Kubernetes is an Open-Source Container orchestrator & Scheduler to mange Containerized applications & Services.

Page No.
Date

What is Kubernetes

Kubernetes (K8s) is an Open Source System to deploy, Scale & manage Containerized application anywhere.

Kubernetes automates Operation tasks of Container management & include built in Commands for deploying applications, Rolling Out Changes to your applications, Scaling your applications up & down to fit Changing needs, monitoring your applications & more - making it easier to manage applications.

Udemy Explanation :-

(1) Kubernetes is Open Source Orchestration System for Docker Containers.

(2) Kubernetes Implemented by Google

(3) Kubernetes Manage & Control the Containerized application.

(4) Kubernetes is a platform that eliminates the Manual processes involved in deploying Containerised application Containers.

(5) Kubernetes used to manage the State of Containers. (State means)
   (a) Start Containers On Specific Nodes
   (b) Restart Containers when gets killed (Healthy Container)
   (c) move Containers One Node to Another.

Let's discusses few problems which Orchestrator tools like Kubernets or Docker Swarm will reduce.

I) Increase the Human Cost of Running Service.

a) we need to manage all Container manually, wee need to monitor upgrade, health check & lots of thing which can impact your Environment.

II) Increase the Bills from Cloud Service provider

a) Somehow it is hard to predict that how many hardware resources required to execute a specific number of Containers.

III) Increase Complexity of infrastructure.

a) increase Containers up on Years down the line, multiple Environment, the number of application also increasing Can Create Complexity of infrastrue.

IV) Setting up Services Manually.

a) Setting up Services Manually via Command line.

IV) Scaling was very Difficult.

a) Need to Scale manually. identify & Scaling is difficult.

VI) Manual fix if any Node is Crashed.

a) fix manually, or rebuild the Node, Restart Manually.

multiple Services Running inside Containers.

might around 800 to 1000 Containized Services.

Page No.
Date

Features of Kubernetes :-

(1) Automated Scheduling :- Kubernetes provide advanced Scheduler to launch Container On Cluster nodes based

(2) Healing Capabilites :- Kubernetes identify if Something go wrong with Containers & take appriate action like Ressource increases, Restart ... etc

(3) Auto upgrade & Roll Back :- Automatic roll Back.

(4) Horizontal Scaling :- Automatical Scale

(5) Storage Orchestration :- with Kubernetes you can mount the Storage System of your choice you can either opt for local Storage, or choose a public cloud provider.

(6) Secret & Configuration management : Kubernetes Keep Secret & won't Exposes & also maintain the Configuration file. So that any time if Container Crashed it can Re-Spin the Containers.

(7) You can Run Kubernetes Anywhere :
On-premise (Own Data Center)
public cloud (Google, AWS, Azure, Digital Ocean...)
Hybrid Cloud.

KUBERNETES ARCHITECTURE

(1) Kubernetes follow the master-Slave (worker) Node Architecture.

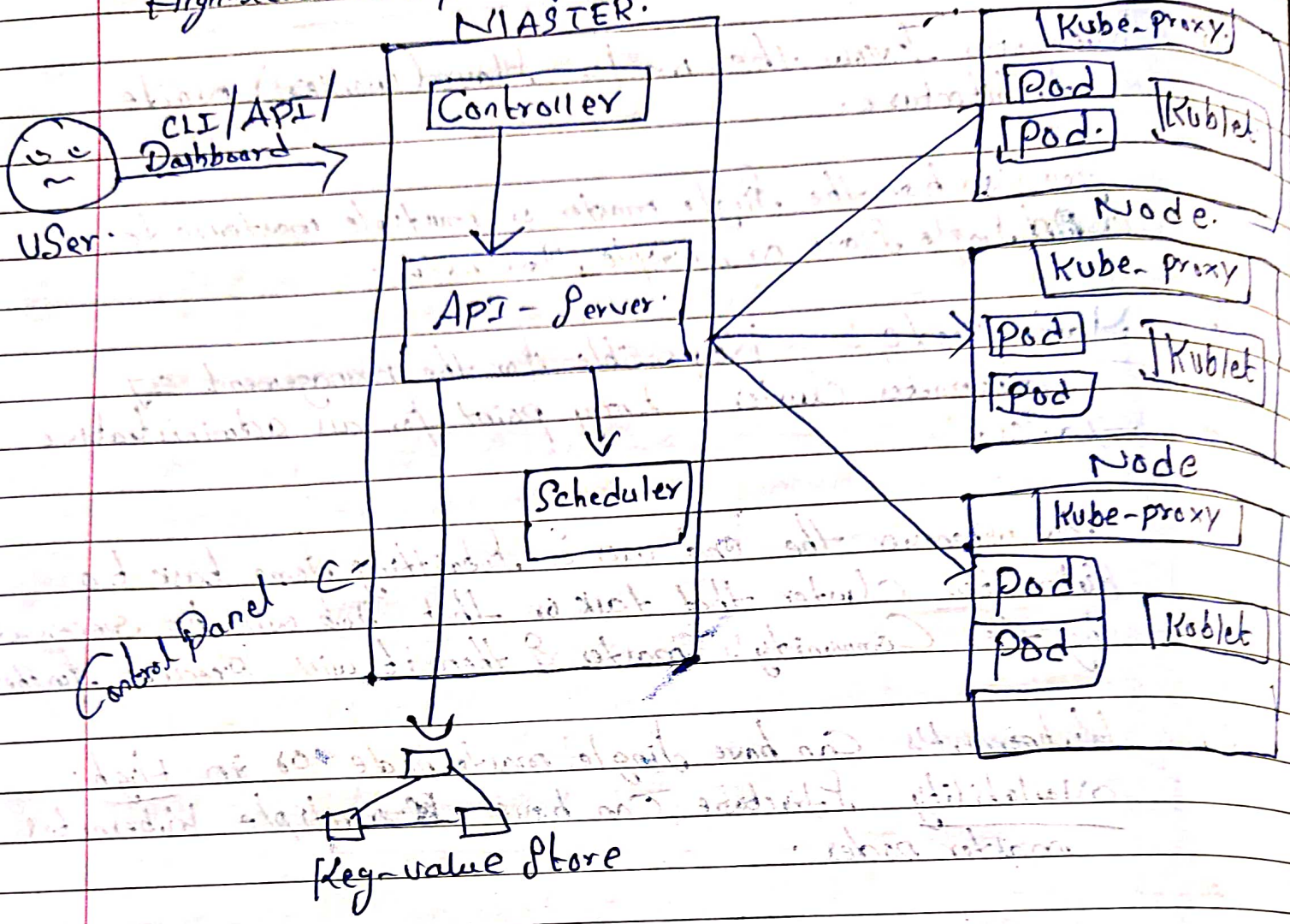
You can have the Single master or multiple masternode and Single Slave or multiple Slave nodes.

(2) Master node :- Responsible for the management of Kubernetes cluster. Entry point for all administrative tasks.

So whenever the end user is submitting some task to Kubernetes cluster, that task or that Job will be received by the Community's master & then it will execute further.

(3) Kubernetes can have single masternode or in High availability structure can have multiple Kubernetes master nodes.

High-level workflow of Kubernetes :



(a) End user can submit the request to Kubernetes cluster, either by CLI / rest API / dashboard.

(b) that event will be accepted by the Kubernetes master node, which have the multiple components and the combination of these components called the "Control Plane".

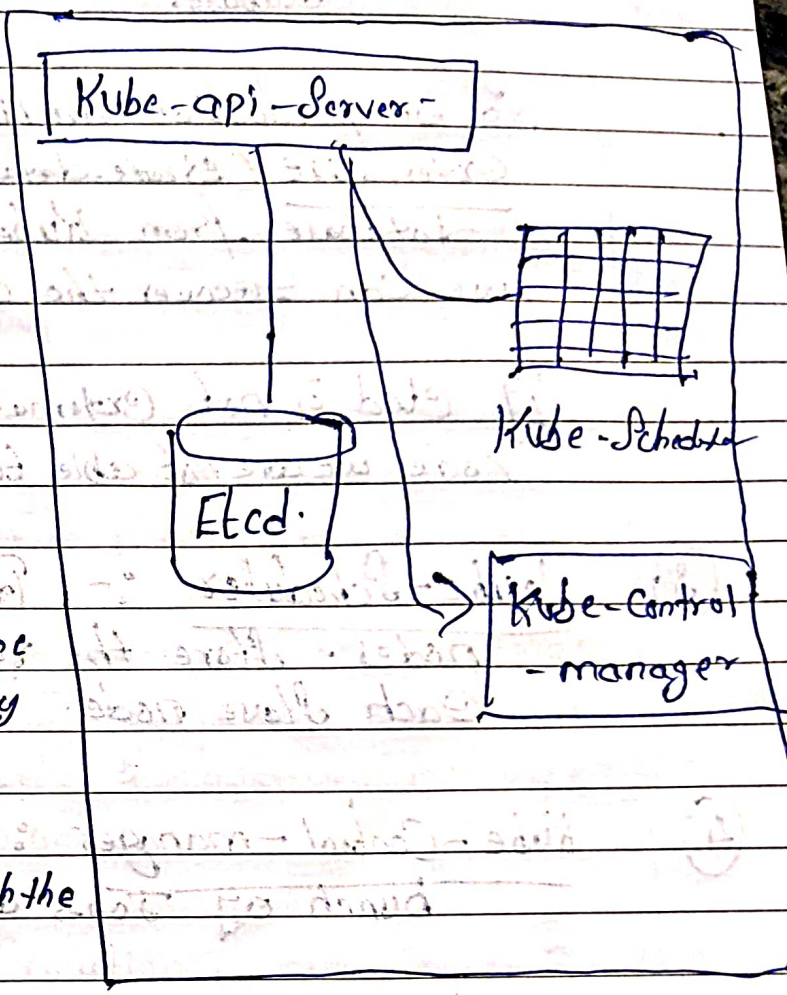
this is Control plane :- we have Kube Controller & Kube-API-Server, KUBE Scheduler & KUBE-Key-Value which is called "ETCd"

(3) On worker node we have Kube-proxy & Kubelet

MASTER NODE COMPONENTS DISCUSSION.

(1) Kube-API-SEVER:-

- API Server is an entry point with in the cluster & that is a single destination for all the rest API's



(2) whatever you are submitting to your Kubernetes Cluster with in Kubernetes cluster all the requests accepted by the Kubernetes that is being done by the rest API's.

(3) the only interaction point with the Kubernetes.

(4) Etcd:- is a key-value or you can say that is a distributed database for the Kubernetes.

a) Distributed key-value store which stores the cluster state

it will contain the state of your Kubernetes cluster & that is used as a backend service for your Kubernetes cluster.

b) provides high availability of Data related to cluster state.

The Complete State of your Kubernetes Cluster is saved in the "etcd" database. due to any case if you lost your cluster, then you can recover your cluster with the "etcd" database.

So on production machines, it is very necessary to externalise (Create datastore in local pc) the "etcd" database from Kubernetes Cluster, in case of down we can recover the data.

If etcd is not externalise, in case of failure of master node we are not able to recover the data.

(3) Kube-Scheduler :- Regulates the tasks on slave nodes. Store the resource usage information of each slave node.

(4) Kube-Control-manager :- Controller manage the bunch of jobs as a single process.

In Kubernetes cluster, there's a lot of automated task, there is a lot of automated process which cluster can execute & all that particular thing is done by the Kube Control manager or Kube Controller.

Page No.
Date

WORKER Components :-

1) Worker Node :- It's a physical Server or you can say a VM where the Container managed by the cluster Run. (managed by master node).

worker nodes contain all the necessary Services to manage the n/w b/w the Containers, Communicate with the master node, & assign resources to the Scheduled Containers.

2) Kubelet :- On each Kubernet worker nodes will get a utility called Kubelet.

It is agent that execute on the worker nodes. that agent will directly Communicate with the Kube API Server on the master node & that will manage the State of your worker nodes.

The Execution of Pods or Execution of the Services on the workNode is responsibility of the Kubelet.

Kubelet gets the Configuration of a pod from the API Server and Ensures that the described Containers are UP & running.

3) Pod :- basically a group of One & more Containers with in pod, with in a Single ~~pod~~ pod. which is a logical unit in Kubernetes.

you can have the Single Container or the multiple Containers but all the Containers which are running with in a Single pod they must have the Similar kind of Configuration.

You cannot execute Cross Configuration Containers within a Single pod.

If you have the Cross Configuration Containers, then you need to define the Separate pods for there Cross Configuration Containers.

(4) But in layman's term, the pod is a logical unit where you can have the Single ~~Content~~ Content & ~~the Same IP~~.

Container or multiple Container with the Similar Configuration, which will share the Storage, which will share the N/w another thing is that the Containers within the pods can share the Shared Content & the "Same IP"

★ It means IP is not associated with the Containers In Docker Sworm (Another Orchestrator) means IP is associated with the Containers.

But Kubernetes IP is associated with the port, now with Containers which are a part of Single pod.

★ they can Communicate to each Other via local host.

5) Single pod can run on the multiple machines So you can execute a Single pod on a multiple worker nodes.

Single machine can also execute the multiple pods.

(6) Kube-Proxy :- Kube proxy runs on each node to deal with individual host Sub-netting & ensure that the services are available to External parties.

Kube proxy is a Component which will provide the Connectivity b/w the Containers & the external world.

Installation of Kubernetes :-

(1) Kubernets (K8s) Can be installed in Two ways :-

a) Kubernetes HA Deployment (1 - master / 2 - worke Suitable for production like SetUp.

b) Single Node Deployment (miniKube K8s cluster) Suitable for Development / practice.

(2) First install Docker on the machine. (follow procedu by See Doc. Docker.com)

(3) install Kubectl. (Can be used in HA Deployment & in Single node Deployment).

Kubectl is CLI Commands.

(4) for installation follow the udemy Course video no - 184. (Course :- devops-training) and also Refer Online installation documents.

Date

First interact with K8s Cluster :- Video :- 186.

(1) To deploy Hello-node Sample pod/container using kube-ctl.

```
Kubectl Create deployment hello-node --image =
                    k8s.gcr.io/echoserver:1.4
```

(2) To check if deployment is successful. (created or not.

```
Kubectl get deployment.
```

(3) To check if how many pods are there :-

```
Kubectl get pod.
```

(4) To Exposes the pod deployment/pod to the External world. (means we can access the pod for internal, so we need to Expose port).

You need to Create the Service within the Kubernetes.

Service is something which will Create the access point, the proxy point for your application & using that access point, External user can access your Service within the Kubernetes cluster to Create the Service.

```
Kubectl Expose deployment hello-node --type=LoadBalancer
                    --port 8080
```

Page No.
Date

(5) To Verify Services.

```
kubectl get services
```

(6) Want to access Create Service.

```
minikube service hello-node
```

(7) To delete the Service.

```
kubectl delete service hello-node.
```

(8) To delete the Deployment.

```
kubectl delete deployment <hello-node>
```

→ Deployment name.

NameSpace in Kubernets (K8S)

NameSpaces are Virtual cluster backed by the same physical cluster.
K8S objects, such as pods and Containers live in NameSpace. & they will be limited to the naming.

you can understand the name space like workspace & you can Create a multiple namespaces or workspace within a Kubernetes cluster.

ideally the basic function of namespace is to separate & organise the object in K8S cluster.

For Example:-

In production. Environment for ten clients on a single cluster we can Create different Service structure. So, for each customer you can define the different namespace. On a single Kubernetes.

cluster & Services which are executing for that customer. or the Containers ports, which are executing for that specific Container, they will execute within that Customer namespace.

in that case, the resources is specific to a customer will be bind within a logical unit.

Page No.
Date

List Existing Namespace :-

(a) Namespaces can be list using Kubectl :-

```
Kubectl get namespaces
```

(b) if user is not defining the namespace, k8s will assign default namespace

(c) To Create a Specify Namespace.

Kubectl get pods --namespace <namespace>
(will return the pods with namespace)

(d)
```
Kubectl Create namespace <namespace>
```

(e) To get pods in Namespaces

```
Kubectl get pods --namespace <namespace>
```

```
Kubectl get pods --all <namespaces>
```