Date

Kubernetes - K8s

- Initially: We had to buy new server for each application; if anybody wants ~~to buy me~~ more applications running, you needed buy more servers.

- SysAdmins were very crucial in handling the servers.

Then containers came into play.
They will run on the host- operating system only, using some container engine, which was "Docker".

One more thing also came around this time, which was:

- Configuration Management : It meant you can maintain all the changes in your infrastructure with help of software.

Note: Configuration management shifted from mutable to immutable

- Infrastructure as code principles:

1. Chef
2. Puppet

Date ____________

The shift from from mutable to immutable configurations occured due to the need for more reliable and scalable infrastructure.

Explaination:

Mutable Configuration:- (Before)

Traditional systems used mutable configurations, where servers were constantly updated in place.

This approach often led to configuration drift, inconsistencies, and difficult rollbacks.

Managing servers was complex and time consuming.

Immutable Configurations: (Now)

Immutable configurations involves creating and deploying identical, unchangeable server instances.

This approach ensures consistency, and predictability.

Immutability simplifies updates and rollbacks, enhancing reliability in the Devops pipeline.

Date

Monolithic Application: It is a software, where everything is connected together like a single, large block, making it harder to change or scale individual parts.

e.g.

Front-end
Back-end
chat messages
Database
Networking

} This whole thing is in single container, so this whole container is deployed at once.

Problem: As this whole thing is packed as one, if you want to scale only frontend, you have to scale backend, along with other components.

e.g: If you want to scale front-end on ten servers, but you don't want scale other components on ten servers, so it's a problem.

Note: We solved this problem by microservices.

Date

Microservices

Now, instead of deploying previously displayed components as one package in a container, they're being deployed differently / seperately.

e.g.

- Front-end
- Back-end
- Database.
- Messages
- Networking.

They are deployed seperately, but are part of single application, So they have to communicate with each other. (Manage?)

Note. You can run your microservices inside containers.

- How to manage ?
- How to communicate?

One app on one VM
This is not cool for scaling, Hence Docker.

Date

But we want to update servers while they're running, 'Orchestration" helps with this .

~~Orchestration.~~

Orchestrator :

It helps us in deploying and managing applications. dynamically.

- Kubernetes helps in doing this .

Functions of Orchestrator:

- Deploy
- Zero Down-time
- Updates
- Scale
- Heal container, when some containers go down.

These are all features of cloud-native applicatin

Note: Applications which follow all the above functions and applications that can run on top of Kubernetes, these are known as cloud-native - applications.

- Anywhere you have kubernetes, you can run cloud-native applications.

Date

- Kubernetes offers more than an orchestrator, so Kubernetes is more than a container orchestrator.

Kubernetes points:

- You can run it on your own cloud, or on your cloud providers, you can migrate from one provider to another provider.

- Kubernetes can replicate services, can scale services, put on dedicated servers, zero down-time deployment, fault tolerance, self healing of clusters, you can use volumes with it, provides load-balancing, you can access logs, service discovery, can save passwords and stuff using secrets.

- Google made kubernetes opensource in 2014 & donated it to CNCF.

Docker vs Kubernetes

Docker: It is a platform for containerization, which packages applications and their dependencies for consistent and portable execution.

. It is a platform that uses OCI (open container Initiative) standard to create and run containers.

Kubernetes: It is a container orchestration platform, used to automate the deployment, scaling, and managing of containerized applications across cluster of machines. It's about managing and orchestrating multiple containers.

. It is an Orchestration platform that can manage containers created using the OCI standard, which includes Docker containers, but it's not limited to Docker and can manage other OCI - compliant contaiers as well.

Date

Docker vs kubernetes continued...

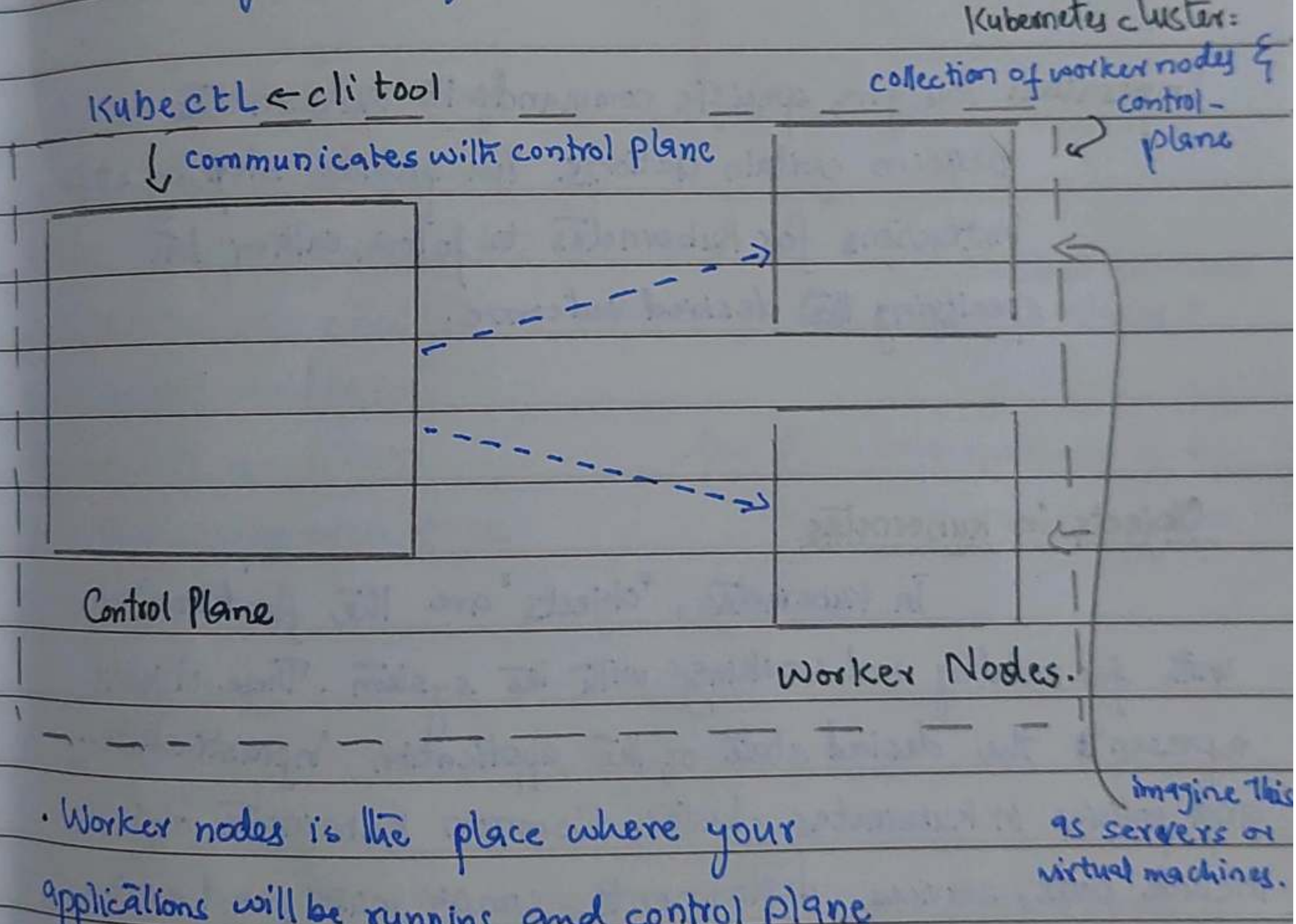
Docker: It uses the OCI (open container initiative) standard for container runtime and image format. It uses its own CRI (Container Run time Interface) called containered.

kubernetes: It also uses OCI for container runtime and image format, and its supports multiple CRI implementations, including containered, CRI-O, and others. Kubernetes is container runtime-agnostic.

Architecture Of Kubernetes

Cluster: Control plane + Node.

Note: Node can be treated a virtual machines. /servers.
Control Plane was previously knooon as "Master Node", but it was not a good naming convention.


Kubernetes cluster:
collection of worker nodes & control-plane
Kubectl ← cli tool
communicates with control plane
Control Plane
Worker Nodes.
imagine this as servers or virtual machines.

- Worker nodes is the place where your applications will be running and control plane is actually going to manage the worker nodes.

Note: Micro-services are on the worker nodes.

Date

Declarative & Imperative way in kubernetes :-

Declarative: You tell kubernetes what you want, it figures out how to do it. You describe you desired state, and kubernetes ~~to perform~~ continously works to maintain that state. (you tell by using "YAML")

Imperative: You give specific commands to kubernetes to perform certain actions. You provide step by step instructions for kubernetes to follow, rather than specifying the desired outcome.

Objects in kubernetes

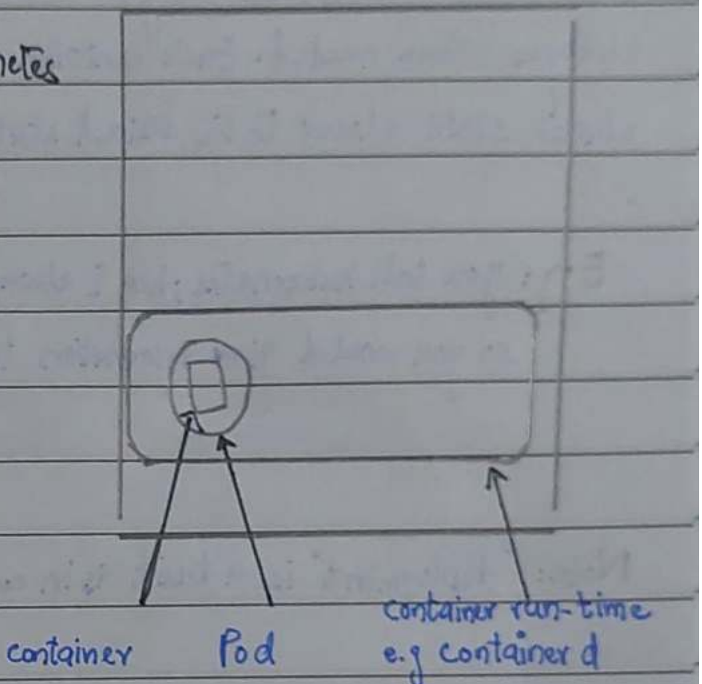
In kubernetes, "objects" are the fundamental units for modeling and working with the system. These objects represents the desired state of the application, infrastructure, and policies in kubernetes cluster. Common kubernetes object include pods, services, deployments, config maps, and more. Kubernetes uses these objects to manage, scale and maintain applications and infrastructure in a declarative way.

Pod : smallest scheduling unit

Steps for running app in kubernetes

1. Create your web-applications in micro-services.

2. Containerize it. (Add every micro-services in its container)
Note: Every stack would be in different container and every container would be in different pods, so one can scale and manage each stack individually.



One Pod - One container is a good convention.

3. Put every container in respective pods.

4. Deploy these pods to controllers, such as 'deployments'.

To understand pod and deployment assume:
pod can be like (int) primitive data-type.
deployment can be like (arrays) complex data-typ.

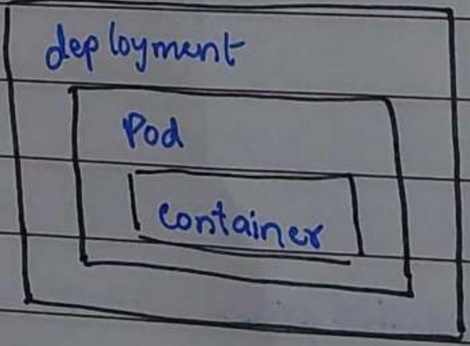
e.g: when you want to scale youre integer, you resize the array.

Controllers:

In kubernetes, controllers are control loops that watch the state of your cluster, then make or request changes where needed. Each controller tries to move the current cluster state closer to the desired state.

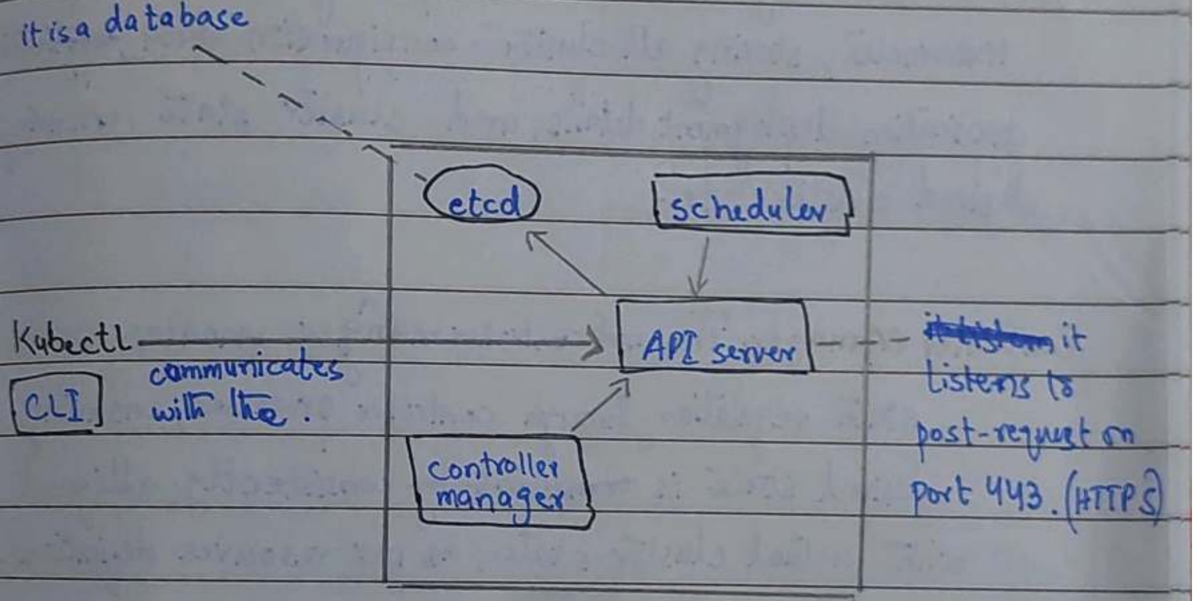
E.g: you tell kubernetes; hey I always want you to run 5 pods, so you would giver information to ~~the~~ controller ← e.g. deployment

Note: "deployment" is a built ~~is~~ in controller in kubernetes.



Control plane:

It is like the "brain" of the cluster. It manages and coordinates all the activities, making sure applications run as intended and maintaining their state. It includes components such as the API server, etcd, controller manager, ~~scheduler~~ scheduler and more to ensure the cluster operates smoothly.



Architecture of contol plane.

Note: All the communication will happen via API server

- YAML files are known as kubernetes manifest files.

small note: Controller manager:

- It ensures the desired state.
- It manages the current state.
- Checkout the differences b/w d.s & c.s
- and makes changes

API server: The central control plane enables communication with the cluster through the kubernetes API server, facilitating administrative tasks like application deployment, scaling, and configuration updates.

etcd: The etcd key-value store is the primary data store for kubernetes, storing all cluster configuration data, ~~application~~ application deployment details, and cluster state, ensuring desired cluster state.

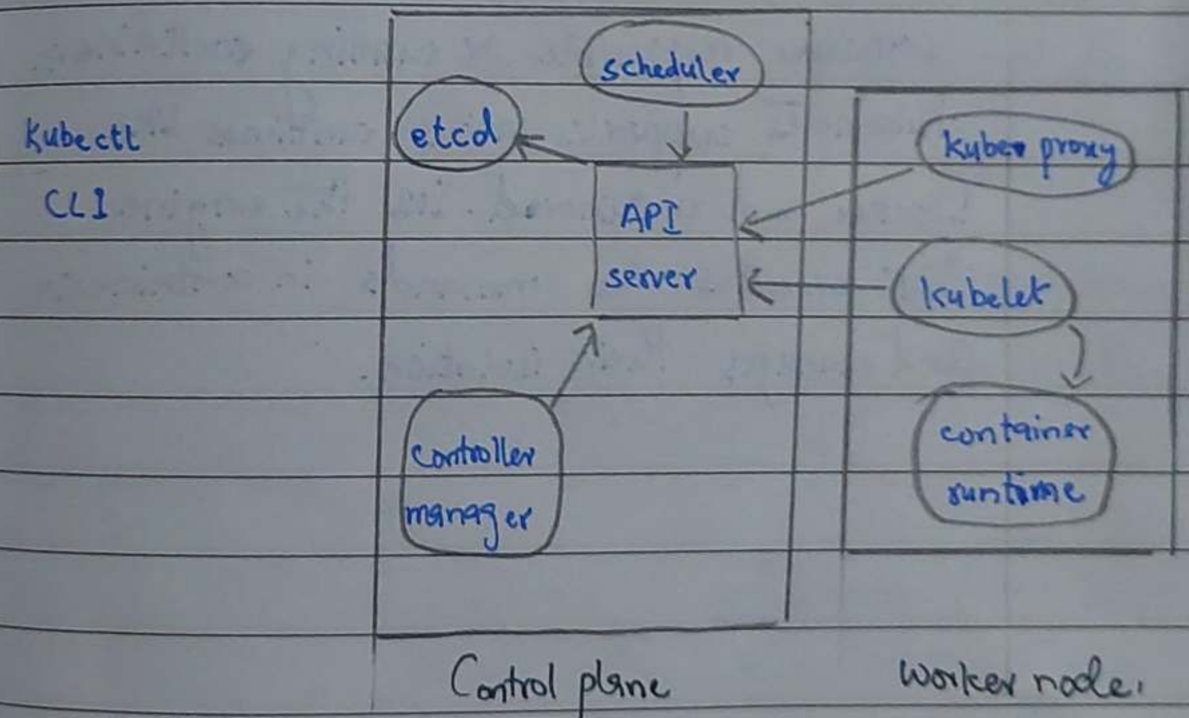
Controller Manager: The ~~cot~~ controller manager manages system state regulation through controller processes, ensuring desired state is ~~consistent ly~~ consistently alligned with actual cluster state, as per resource definitions.

Scheduler: The scheduler manages clusters efficiently by placing pods onto nodes, considering resource requirements, node health, and user-defined constraints.

Cloud Controller manager (Optional): This kubernetes component manages cloud infrastructure interations, creating load balancers and volumes, and abstracting cloud provider-specific details from core kubernetes-componets.

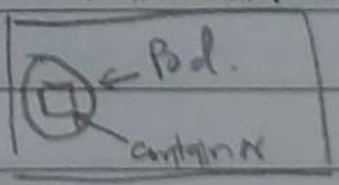
DNS server (Optional): Kubernetes typically includes a DNS server for service discovery, within the cluster, enabling pods to locate services by their DNS names.

Architecture Of kubernetes :



Note: Control plane is running on top of Linux; Linux is on top of hardware

Container run-time =



~~Kublet:~~

Kublet: kublet is an agent running on each node in the cluster. It ensures that containers are running in a pod as expected. It communicates with the control plane and manages the containers' lifecycle.

Kube-proxy: It is a network proxy that helps manage network connectivity for pods. It maintain networks rules on each node to enable communication between different pods and services in the cluster.

Container-Runtime: The container runtime is the software responsible for running containers. Kubernetes supports various runtimes like Docker and containerd. It's the engine that executes the commands in containers and manages their isolation.

Worker node components in an easy way:

Kublet:. Its on every worker node. Every time a worker node is produced and attached to control plane, kublet is installed on it.

Its JOB: It will listen to API server api; so via kubectL, ~~it will be told to create Pods~~ and via API server, it will be told to create Pods.

Kubelet will say; "okay you are requesting something, I'll allocate those on my own. (

If its not able to do the task, it will report back to control plane, then the API server will tell this to scheduler and control manager, and will do something about it.

Container runtime: It uses containerd (Pulling & Pushing the container, creating a container out of it, stopping the container, destroying the container.)

Previously; there was support for docker and stuff but then CRI (container runtime interface) was introduced, so now docker thing is replaced with containerd. (which is also a cncf project)

Note: Docker doesn't support CRI, (but containerd supports it).

Date

Kube-proxy : It is something responsible for networking. So if you want your workernode or your cluster to communicate outside network, kube-proxy will help in that.
It ensures that every worker node, gets its own unique IP address.
This will provide IP address to nodes.
It also handles load-balancing.

K8s DNS

Example there are two nodes, It basically has IP addresses for every pod and all containers and Pods know how they communicate with each other.

Installation:

Kubctl
Minikube
Kube adm

x——x——x—

Kubeconfig file has some secret information.
If somebody gives you kubeconfig file; it means, ~~it has som~~ you can access their kubernetes cluster.

Date

Comands : # After running command: minikube start

To see the running pods:

- Kubectl get pods

To checkout nodes:

- kubectl get nodes // on it will show only "minikube".
  // role: control plane, master.

To see dashboard of minikube:

- minikube dashboard // it will open dashboard on
  // Local host.

Comand: minikube docker-env

It configures your ~~tocke~~ local Docker client to work with the docker daemon in a MiniKube cluster, enabling seamless container image building and running using your ~~Loc~~ Local Docker installation.

To see containers That are required by Kubernetes:

- docker container ls

Everything that is running, is runing inside a container, but we can see, how many containers are running inside a ~~conta~~ ~~node~~ minikube (node)

To see how many containers are running inside a node:

command: minikube ssh //By this, we'll go inside //the minikube

Note: Inside minikube, docker runtime is installed.

command: docker ps //It will List running //containers.

To view kubeconfig file:

- Kubectl config view

Commgnd: It determines which cluster and namespace you're currently working on and which user's credentials to use.

↳ · kubectl config current-context.

Command: kubectl get all // it will shall all pods, services, deployments & replicas.

To delete pod:

- Kubectl delete pod 'name'

To get deployments:

- Kubectl get deployments

To delete deployment:

kubectl delete deployments "name"

Note: If deployment is deleted pods will be automatically deleted.

Tools: Lens 5
Monokle
kubescape
datree
Teleport

Date

~~Not on GitHub~~

api version:
It defines the version of the kubernetes API you're using to create this project.

v1: It means that the kubernetes object is part of first stable release of the kubernetes API. so it consists of core objects such as pods, Replication Controller and Service.

apps/v1: Includes functionality related to running apps in kubernetes.

batch/v1: Consists of objects related to bash processes and jobs like tasks.

kind: It defines the type of object being created.

Metadata: Data that helps uniquely identify the object, including a name strings, UID, and optional namespace.

Spec: The precise format of the object spec is different for every kelbernetes object, and contains nested fields specific to that objet. ~~For more infor~~
e.g: What type of containers you're running in.

Pod Manifest file (YAML)

```
apiVersion :  v1
 Kind      :  Pod
metadata :
  name :
     nginx-pod
    Labels:
       app: nginx
      tier : dev
spec:
    containers:
     - name : nginx-container
      image: nginx
```

Whenever you want to create object out of this type of file:
Deploy the Pod from nginx-pod.yaml

```
Kubectl create -f nginx-pod.yaml //Pod will be
                                 //created for this file.
```

```
kubectl create -f "name of file".
```

To get info:

```
kubectL get pod "filename"
```

If you want more info:

```
Kubect get pod "filename" -o wide
```

This will display details of the pods which includes list of all events from the time the pod is sent to the node till the current status of the pod

```
Kubectl describe pod "nginx-pod"
```

Check if the pods are accessible; verify if the connectivity from the master node to the pod is working by using the pod's IP address

```
ping "172.17.0.7"
```

You have to ping, being inside of minikube. (outside the worker node)

To access the nginx pod on your local machine:

```
Port-forward nginx-pod 8080:80
```