**KUBERNETES**

* When master node wants to talk to worker node, it directly talk. Because it is aware of all the loads of all nodes.
* When worker node wants to talk to master, talk through load balancer.
* Architecture

Master node 🡪 Manage ,Plan, Schedule, Monitor nodes

Kube-Scheduler

Kube-Controller

Kube-apiserver

ETCD cluster

* Whenever u want to execute a command done through Kubectl(client).Generally talk to API server using kubectl. It will convert your request to REST API request. API server will authenticaticate (Are u a real user or not), authorize(specific people have access), validate (whether meets the expectation ie, well formed, mandatory fields are there or not) and talk to ETCD.
* API server will only do CRUD operation (r- read (list and get). If modify a pod, API server will update the SQL table, if remove a pod, API server will delete in the table.
* ETCD – NoSQL database. It is the only solution which stores data. API server will not store data. Every master has different ETCD. It uses the **Raft Consensus Algorithm**
* If u have a entry (pod)in the table, scheduler will come up and find out which machine best for this. Ask the ETCD and find out this and update back to ETCD.
* Other than API server, nobody will directly talk ETCD. Everyone will talk through API server.
* Controller will control all these operation. It will ensures all the operation are done correctly. Each controller have its own business logic to solve the problem.

Node controller – check node is up or not

Replication controller

Deployment controller

Token controller

Job controller

Pod controller

* Control plane – Master node
* Data plane – where your application runs.(worker nodes)

Worker node – Host application as containers

Kubelet

Kube-proxy

Containerd/ CRIO(CRI) / pods (container runtime)

* POD is a group of 1 or more containers. Recommendation: run only one container in a pod. All the containers in the pod are collocated, ie all of them run on same node. Also all container in a pod are co scheduled (they come up together) . And multiple containers in pods share same name spaces.(but uts name space will not share, also every container have separate user space)
* In a pod, all containers have same ip address which is the ip of pod itself. But they communicate through different ports.
* Within containerd, have container
* Kubelet is a small program which is like a local manager. Most of the purpose directly talking to Kubelet. API server will talk with kubelet and it will talk with containerd. kubelet interact/work with container runtime and create pod.
* Most using container runtime is
* Proxy run on every worker node and understand the networking of every node. ie, proxy run on node1 have understanding of node2. It uses ip tables for network related configurations.
* Except kubelet and container run time, all other components of worker node will run on master node also.
* When u want create a container, u will talk with API server in master node and then ETCD, scheduler and controller get involved and then it will talk with kubelet in worker node and it will talk with Containerd. If a client want to access, will directly access from worker node through the load balancer.
* When a pod created 🡪kubectl request a REST API request to API server includes certification(keys), request (GET,PUT) 🡪 API server authenticate, auth … 🡪 scheduler run its algorithm an find out which machine s best and update 🡪 pod controller see this and update the ETCD that there is a new entry 🡪 API talk to kubelet 🡪 kubelet will take care what to do in that worker node based on this and keep updated
* Kubeadm init

Used to initialize a **Kubernetes control plane** node in a Kubernetes cluster. This sets up the first master node (control plane) and prepares the system to run a Kubernetes cluster.

**What Does kubeadm init Do?**

When you run kubeadm init, it performs the following steps to initialize your Kubernetes master node:

* **Set Up the Control Plane**: It installs and configures key components like the Kubernetes API server, controller manager, and scheduler.
* **Configure Kubeconfig**: It sets up a kubeconfig file (usually located at /etc/kubernetes/admin.conf), which is used by kubectl to interact with the Kubernetes API server. This file contains authentication details for your Kubernetes cluster.
* **Set Up Networking**: It prepares the system for networking (typically by applying a network plugin).
* **Set Up the Cluster Token**: It generates a token, which is used by other nodes (worker nodes) to join the cluster.
* **Display Join Command**: After initializing the master node, it will print a command that you can run on worker nodes to join the cluster.
* **Optional Add-ons**: It will install certain add-ons like CoreDNS (for DNS resolution) and a default network plugin (such as kube-proxy).

### [docker-k8s/notes/day4 at master · vilasvarghese/docker-k8s](https://github.com/vilasvarghese/docker-k8s/tree/master/notes/day4)

Commands

* Systemctl status kubelet 🡪 display the current status of kubelet service
* Kubectl get nodes 🡪 list all the nodes (kubectl is for communicate with API server)
* Kubectl get pods 🡪 list all the nodes
* Kubectl get pod <pod\_name> 🡪 retrieve information about a specific Pod
* Kubectl get pods -n kube-system 🡪 lists all the Pods running in the kube-system namespace.
* Kubectl get pods -n kube-system -o wide 🡪 display the output in a more detailed and expanded format
* Kubectl run pod\_name> --image=nginx 🡪 To create a pod

Eventual consistency – Things are reported are done. But actually it is not done.

* Kubectl describe pod vinaypod 🡪 like inspect
* Watch -n 1 kubectl get pod -n kube-system -o wide 🡪 **continuously monitor the state of pods** and refresh the output every 1 second.
* Kubectl api-resources 🡪 show all resources
* Kubectl delete pod --all 🡪 delete all pods. But Kubernetes have the property of recreate the pods which are deleted.
* Kubectl delete -f <name> 🡪 **delete resources** in your Kubernetes cluster that are defined in a YAML or JSON file specified by <name>. This file contains the configuration of the resources, such as Pods, Deployments, Services, ConfigMaps, etc., and when you run this command, Kubernetes will remove the resources defined in that file.
* Kubectl delete pod <pod\_name> 🡪 delete a specific pod
* Kubectl apply -f <file\_name>
* Kubectl get pod -o yaml 🡪 retrieve detailed information about **Pods** in your Kubernetes cluster and display the output in **YAML** format.
* Kubectl get pod -o yaml > newpod.yaml 🡪 retrieve the YAML definition of **all Pods** in the current namespace and redirect the output into a file called newpod.yaml. Now we can edit in this.
* Kubectl run <pod\_name> --image=nginx –dry-run=client -o yaml > filename.yaml 🡪 create a pod and create yaml file for that and we can access that and edit. (--dry-run=client flag simulates the creation of the Pod locally and outputs the generated YAML to a file named filename.yaml.)
* Kubectl api-resources |grep -i replicaset 🡪 to see replicaset api version
* Kubectl get ns 🡪 show the namespaces

Whatever pod u running in current is execute in default name space

* Kubectl edit rs frontend 🡪 edit a ReplicaSet **(rs)** named frontend directly in your Kubernetes cluster
* Kubectl scale rs frontend –replicas=3
* Kubectl delete rs –all 🡪 delete all replica set
* when a container crashes or u killed it, Kubernetes will replace the container. In some case Kubernetes will inject a pause container. if the pause container crashes, the entire pod will replace.
* When you create a pod, Kubernetes automatically creates the pause container as the first container in the pod. This pause container is a docker container of Kubernetes.
* when a container stopped 🡪 assign to node🡪pull the image 🡪 error occurred 🡪 failure happen 🡪 again try pull 🡪 create the container again (this all done by kubelet)
* kubelet and container run time is not run as a pod in master. All other will run as a pod.
* container runtime in Docker – Docker shim
* container runtime in Kubernetes – Containerd, Cri-o, Docker shim
* yaml file contains
* api version
* kind 🡪 what kind of object (Eg :- pod)
* metadata 🡪 if give name in this section , it will create a object with that name.

Eg : name : nginx (create pod with name nginx)

* spec 🡪 for pod, u have to give containers and inside that image
* replica-set.yaml file contains
  + api version
  + kind
  + metadata : name, labels, name space
  + spec 🡪 template : in spec template section talk about pod. And mention label in that is imp.

Selector: select that label

Replicas : how many replicas u want

* pods running in kube-system 🡪 api server, controller, scheduler, etcd, kube-proxy, core dns
* For pod creation, scheduler comes first later controller come. But in replicas, replication controller first come and then the scheduler.

**Deployement (**another object**)**

* Kubectl get deploy 🡪 retrieve information about Deployments
* Kubectl describe deploy <deploy\_name>
* Kubectl rollout status deploy <deploy\_name> 🡪 checks the status of the rollout for the deployment.

 **In Progress**: If the Deployment is being updated (e.g., a new version of the container is being applied).

 **Successful**: If the update is complete and the new Pods are running as expected.

 **Failed**: If there are issues during the rollout (e.g., Pods are not being updated or are failing)

* Kubectl rollout history deploy <deploy\_name> 🡪view the rollout history of a Deployment. It shows you the history of revisions for a Deployment.
* kubectl scale deploy nginx-deployment --replicas=5
* kubectl set image deploy nginx-deployment nginx=nginx:1.16.1 🡪 update the container image for a Deployment
* kubectl annotate deploy nginx-deployment kubernetes.io/change-cause="upgraded from failing version 1.9.1 to 1.16.1"
* Kubectl rollout undo deploy <deploy-name> 🡪 roll back to previous version
* Kubectl rollout undo deploy <deploy-name> --to-revision=2 🡪 roll back to 2nd version
* Kubectl get pod -l env=deployment 🡪 selects only the Pods that have a label env=deployment.
* Kubectl get pod --show-labels 🡪 show all pods with labels
* Kubectl create ns newns 🡪 create a namespace
* Kubectl run nspod -n newns --image=nginx 🡪 create a pod in newns namespace
* Kubectl get pod -n newns 🡪 show all Pods in the newns namespace
* Kubectl create deploy <name> --image=nginx –replicas =3
* When deployment is created , internally it is creating replicas and internally replicas creating pod
* Rollout – checking it is stable not or not

**Service**

* Kubectl det service
* Kubernetes have 2 types of ports

1. Cluster IP : internal communication (cluster ip have an ip address. So something out can talk to cluster ip and it will talk to the pod)(machines together workings for the same goal is cluster)when u create a cluster ip, it have a port in machine. load balancing and service discovery done by cluster ip.
2. Node port : External communication. It have 3 ports. It will reserve a port on host machine in the range of 30000-32763.if a request come it will pass through one the port in given range by node port and given to cluster ip

* In service deployment if u change something, u should change label ,name and target port.
* When we delete the pods. It will restart pods again in kubernetes but it will assign new ip to pods. So for this problem , we created a service and create a cluster ip through this we can access services running in the pods irrespective of change in ips of pods.
* 3 types of services in kubernetes

Cluster IP , Node port and Load balancer

* Node port 🡪 mention type: NodePort below spec
* Load balancer 🡪 mention type: LoadBalancer
* Whenever u created a load balancer it will internally create a node port and that node port twill communicate with cluster ip and the cluster ip will do the real load balancing
* Linux has capability to create name space. Using that it will create process.
* Network plugin takes to right machine depends on which plugin is using.
* IP in IP (pods ip inside node’s ip). Ones this reaches the right machine, it will unwrap it.
* Default root will go to network plugin. Proxy is act like a network switch
* Cidr range is decided by kubelet. From tunnel ip range , pod will get the ip
* DNS lookup will not working in host. It will works inside pods (curl name )
* In linux there is a file /etc/resolve.config , this will have name server field and shows who is the server
* Service mesh -ingress, to
* Deamon set 🡪 run on all worker nodes. Usecase: monitoring, agents for logging

In these usecases, we want a single pod to run on all the machines.

* Stateful set 🡪which node comes up first, it will take the role of primary.it runs sequentially.
* Replication controller will only work based on equatilty check. But replica-set will work based on in-check. It supports both **equality-based** selectors (like the Replication Controller) as well as **set-based** selectors. Set-based selectors allow you to use more flexible matching options, such as selecting pods by a label with in or notin operators.
* Kubernetes supports declarative syntax (yaml based) and imperative syntax (kubectl based)
* Ingress 🡪 who can talk to me. Once u define ingress, the one with label mentioned can talk.
* Egress 🡪To whom I can talk
* Servicename.namespace.svc.cluster.local 🡪 communicate between pods in different namespace
* Static pod 🡪 automatically create the pod even if it is deleted. And it will create the pod in particular machine. But service will also do the same, but not in particular machine ,may be in any node.
* Upload cert 🡪 not needed when u use one master. Need when multiple master is there.
* Port of api server 🡪 6443
* Port of etcd server 🡪2379,2380,2381
* Port of kubelet 🡪10250

**Scaling in kubernetes**

* 3 types of scaling
* **HPA** (Horizontal Pod Autoscaling)
* Here we define minimum and maximum num of replicas
* Horizontal Pod Autoscaling automatically adjusts the number of pods in a deployment, replica set, or stateful set based on observed CPU utilization.
* HPA constantly monitors the resource usage of pods and adjusts the number of replicas up or down based on a specified target utilization.
* For this u need to install metric server
* resources:

limits:

cpu: 500m 🡪(50 %) above this kill it

requests:

cpu: 200m 🡪(20 %)

* **VPA** (Vertical Pod Autoscaling)
* Here define add more cpu and memory. Not recommended
* Vertical Pod Autoscaling automatically adjusts the CPU and memory requests and limits for a pod based on its usage.
* **CA** (Cluster Autoscaling)
* Add more machines
* Cluster Autoscaling adjusts the number of nodes in your Kubernetes cluster based on the demand for resources. It automatically adds nodes when there are insufficient resources to schedule new pods or removes nodes when they are underutilized.
* Manage kubernetes cluster 🡪 managed by cloud provider . Eg:- eks , aks
* Containers crash because of 2 reason
  + Exceeds the limit
  + No process running inside

**Scheduling**

* Based on
* node name : schedule on particular node
* node label : schedule on group of nodes. Because we can give same label to different nodes
* Once a node got tainted no pod will go there.
* We can create a pod which can tolerate the taint.
* Taint can produce 3 possible effects:
* **NoSchedule** 🡪 Don’t touch the pod which is already running. If replacement happens, it will not come to this to node

Eg :

tolerations:

- key: "app"

operator: "Exists"

# value: "DBNode"

effect: "NoSchedule"

>> When you use operator: "Exists", it means that the **toleration will match any taint** with the given key, regardless of its value.

* **PreferNoSchedule** 🡪 Don’t prefer to come to a particular node. But if others are not free then can come to this node.
* **NoExecute** 🡪 This effect prevents new pods from being scheduled onto a node and will also **evict pods that are already running** on the node unless they have a matching toleration.

**For new pods**: It will prevent them from being scheduled onto the tainted node.

**For running pods**: It will **evict the pods** immediately if they do not have a matching toleration.

* Affinity

Affinity is used to specify that a pod should only be scheduled on nodes that have specific labels

2 types

1. requiredDuringSchedulingIgnoredDuringExecution
2. preferredDuringSchedulingIgnoredDuringExecution

* Anti-affinity : Anti-affinity is used to specify that a pod **should not** be scheduled on certain nodes based on the node's label or the presence of other pods.
* Pod affinity : Pod affinity allows you to specify that a pod should be scheduled **on the same node** or in the **same topology domain (e.g., same availability zone)** as another pod or group of pods.
* Anti-pod affinity : Pod anti-affinity specifies that a pod **should not** be scheduled on the same node or in the same topology domain as other pods.

**Jobs**

* Why we use job of kubernetes instead of cron jobs??
  + U can control access
  + DNS resolve ,like curl using name inside the pod is possible in kubernetes not in linux
  + Job will internally create pods. So if pod crashes , kubernetes will take care of replacing it .But in linux, in pod crashes it cannot be replaced
* Kubernetes support 2 type of job

1. One time job
2. Cron job

* When u create pod using job, it will not terminated. Logs are accessible

Commands

* kubectl label nodes <your-node-name> disktype=ssd 🡪 apply to a node using label
* kubectl taint nodes <node> <key1>=<value1>:NoSchedule 🡪 give taint to node
* kubectl taint nodes <node> <key1>=<value1>:NoSchedule- 🡪remove taint to node
* kubernetes can give permission to subject like: -
* user (Eg : kubernetes-admin)
* group (group of users. Usecase : managing)
* service account (when u want access to services , create service account)
* role – action (list,create,delete,get,describe) ,object (pod, deploy,replica-set,service ) (here create a role)
* role binding - bind a subject to a role . contains subject, role, namespace (here assign role to user)
* cluster role 🡪 giving permission to all nodes (entire cluster)
* cluster role and cluster role binding are not related to any namespace. In this perssion will given to entire namespace

commands

* export MAGIC\_USER=dicktracy
* openssl genrsa -out dicktracy.key 2048 🡪 create rsa key
* openssl req -new -key dicktracy.key -out dicktracy.csr -subj "/CN=${MAGIC\_USER}/O=devs/O=tech-leads" 🡪

>> csr - Certificate sign request (used to sign on a key by ca (certificate authority))

>>CN – common name (dicktracy) which belongs to 2 organizations (devs/o and tech-leads)

>> contains begin certificate req and end certificate req

* ls -ltr /etc/kubernetes/pki
* sudo openssl x509 -req -in dicktracy.csr -CA /etc/kubernetes/pki/ca.crt -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial -out dicktracy.crt -days 500 🡪

>>create a certificate valid for 500 days

>> contain begin certificate and end certificate

* kubectl config view🡪 show configuration of cluster
* kubectl config get-contexts
* kubectl config set-credentials ${MAGIC\_USER}@kubernetes --client-certificate=$HOME/.certs/${MAGIC\_USER}.crt --client-key=$HOME/.certs/${MAGIC\_USER}.key --embed-certs=true 🡪 adding a user to configuration (in command @kubernetes is the cluster)
* kubectl config set-context ${MAGIC\_USER}@kubernetes --cluster=kubernetes --user=${MAGIC\_USER}@kubernetes 🡪 adding a user to context
* kubectl create role pod-reader --verb=get --verb=list --verb=watch --resource=pods 🡪 create role

**Service account**

* Service account is for access kubernetes api call
* I) Set the API server URL (APISERVER).
* II) Set the path to the service account credentials (SERVICEACCOUNT).
* III) Read the service account’s token (TOKEN).
* IV) Reference the certificate authority (CACERT).
* V) Make a GET request to the Kubernetes API to list pods in the default namespace, using the provided token and certificate for secure authentication and validation.

**Inject properties**

* Config map and secrets are inject into pod using volume or environment variable.
* Volume – directory, attach, dynamic variable
* Env variable – static
* Secrets are encoded. You can limit the access to config map using rbac, that’s how u make it confidential
* Config map are plain text. You can expose config map to anyone
* You can limit the access to secrets using rbac. But this is not enough because etcd is plane. So

🡪 Don’t give access to etcd

🡪Encrypt data in etcd

🡪But in pod data is plain text. So restrict access of pods

🡪create a encryption configuration file and add it to api server.yaml

* To create secret name test of type generic

kubectl create secret generic test --from-file=./username.txt --from-file=./password.txt

* echo -n 'password'|base64 🡪 to encode
* echo bXlwYXNzd29yZA== |base64 –decode 🡪 to decode
* kubectl get secret test -o yaml > examplesecret.yaml 🡪 create a yaml file for secret

api url in config file.

Rbac contains config and contexts

**Volume**

* Empty directry 🡪 temporary storage. The data inside an EmptyDir is deleted when the Pod is terminated. It can be attached to some other pod and bring the data.
* Host path 🡪 create a volume in host machine and store (/home/vol). but if pod crashes, the data will gone. Allows Pods to access data on the host machine, making it useful for sharing data between Pods and the host but with a limitation that it is node-specific.
* So stored in NFS(network file storage) which can be connected to all node.
* **Persistent Volume claim** 🡪volume inhost machine
* **Persistent volume 🡪** written by storage administrator which contain all the credential
* PVC will request for storage for a pod and if any pv will found , it will match and Pv and pvc have one to one relation.
* If storage size in pv less than size in pvc , they will not get bound. If it is more, its ok it will bound.ie, size of pvc should be less than or equal to size of pv. And class and access should be exactly matching. Then they will bound and create one to one relation.
* default/my-persistent-volumeclaim 🡪 in default name space claim to my-persistent-volumeclaim
* pv bound to pvc which mount to pod
* Pvc is a name spaced object but not pv
* Storage in protection.
* Pv cannot deleted if a pvc associated with that and pvc cannot delete if a pod associate with that.
* Reclaim policy (it is an attribute of pv)
* retain :  When the PersistentVolumeClaim is deleted, the PersistentVolume still exists
* delete : deletion removes both the PersistentVolume object from Kubernetes, as well as the associated storage asset in the external infrastructure.
* Recycle : volume available again for a new claim.
* Access modes
* Read only many :any node attached to volume can read
* Read write many : any node attached to volume can read and write
* Read write once pod : Only the first pod connect can read
* Read write once : Pods in the node where the first pod connect can read

**Probes**

3 types

1. Liveness probe
2. Readiness probe
3. Startup probe

* When pod is not live, kill it then again it will starts
* When pod is not ready, wait for it to ready

**-------------**

**Backup**

**-------------**

* **Json path :**  is inherent property of kubectl which will help u to directly create json and yaml file
* **Cordon nodes** : make specific nodes in your cluster as unschedulable. Don’t make any new nodes. Use case : when some performance issue
* **Drain** : draining a node means safely evicting all workloads (Pods) from it while marking it as unschedulable. This is useful when performing maintenance, upgrades, or decommissioning a node.

**Helm**

* packet manager for kubernetes
* **Charts**: releases u do
* In helm use 2 types of variables
* Values.yaml : It is like a property file for an application
* Logs – shows logs of only pods
* Events – shows logs for all activities
* kubectl api-resources --namespaced=false // for non namespaced resources
* kubectl api-resources --namespaced=true // for namspaced resources
* kubectl get roles -A or kubectl get roles --all-namespaces 🡪 to get all namespace and its resources

~ home

/ root

Killer question 11

**Deploy.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: deploy-important

namespace: project-tiger

labels:

id: very-important

spec:

replicas: 3

selector:

matchLabels:

id: very-important

template:

metadata:

labels:

id: very-important

spec:

affinity:

podAntiAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: id

operator: In

values:

- very-important

topologyKey: "kubernetes.io/hostname"

containers:

- name: container1

image: nginx:latest

- name: container2

image: tomcat:9.0