Kubernetes

Firstly, Kubernetes in easy, it rules the Futures

K8s= KUBERNETES

1. Differences between Docker and Kubernetes?

Feature	Docker	Kubernetes
Core Functionality	Builds, ships, and runs containers.	Orchestrates and manages containers across multiple nodes.
Role in Ecosystem	Focuses on individual container lifecycle management.	Manages clusters of containers for scalability and high availability.
Scope	Operates at the container level.	Operates at the cluster level.
Networking	Provides basic networking (bridge, host, overlay).	Offers advanced networking with service discovery, load balancing, and network policies.
Scaling	Manual scaling by starting additional containers.	Automatic scaling using Horizontal Pod Autoscaler based on resource usage.
Fault Tolerance	Limited to restarting failed containers manually or using Docker Compose.	Provides self-healing, reschedules failed containers, and maintains application availability.
Monitoring	Basic container logs and resource usage stats.	Integrates with tools like Prometheus and Grafana for advanced monitoring and alerting.
Deployment	Simple container startup using commands	Complex deployment with YAML manifests for pods, deployments, and services.

1.1 What is Kubernetes ?

- **It all about YAML Files**
- **Kubernetes is an open-source container orchestration platform used to automate deployment, scaling, and management of containerized application**
- **Pods are the smallest deployable units in Kubernetes that encapsulate one or more containers, storage, and a network.**

1.2 K8s Production System

PODS

Defination of how to run containers in pod.yaml file (single or multiple containers called pod)

Kubectl is command line for Kubernetes

Replicaset Actually doing auto healing **kubectl get all Kubectl get pod -o wide (to get ip address)**

2 Architecture of Kubernetes

MASTER NODE {CONTROL PLANE}

API SERVER Expose k8s to external world (it takes all incoming request)

SCHEDUIER it decides which worker nodes that pod should run

ETCD key value store cluster related information

Controller manager it will look into every application or pods running as a planned

Cloud Controller manager it is helper to connect to multiple clouds

WORKER NODE (DATA PLANE}

Kublet is "It will take care of every pod" → "It manages Pods on a worker node and ensures their desired state."

Container RUN Time It will run the container

Kubeproxy it will responsible for networking that allocates ip address (AUTO SCALING CONCEPT COMES HERE)

Component	Role in Pod Creation
API Server	Receives, validates, and processes the request.
etcd	Stores all cluster state information.
Scheduler	Decides which Worker Node will run the Pod.
Kubelet	Runs and manages the Pod on the assigned Worker Node.
Container Runtime	Pulls the image and starts the container.
Kube-Proxy	Handles networking, IP allocation, and service routing.
Controller Manager	Ensures the Pod runs as expected.

Example : Story: A Developer Creates a Pod in Kubernetes

- A **developer** wants to deploy a new application, so they create a **Pod definition** in YAML and submit it to Kubernetes using the kubectl apply -f pod.yaml command.
- The request is sent to the **API Server** because the API Server is the main gateway that exposes Kubernetes to the external world.
- It checks if the developer has the right permissions using **RBAC (Role-Based Access Control).** If authorized, it proceeds; otherwise, it rejects the request
- If the request is valid, it stores the information in the **etcd database**.
- The **etcd** database is the **brain of Kubernetes**. It stores all cluster-related data, including: What Pods should exist. What Nodes are available. The current state of the cluster
- After storing the new Pod's details, etcd informs the **Controller Manager** and **Scheduler** that a new Pod needs to be placed.
- **Scheduler checks the available Worker Nodes** and evaluates their resources (CPU, memory, etc.).
- Once a Node is selected, the Scheduler updates the **etcd** database with the decision.
- Now, the **Worker Node** assigned by the Scheduler receives an update.
- The **Kubelet (Node Agent)** running on that Worker Node detects that a new Pod is scheduled to run on it.
- The **Kubelet pulls the Pod's container image** from the configured container registry (e.g., Docker Hub, AWS ECR, or GCR).
- It then asks the **Container Runtime** to start the container.
- The Kubelet continuously monitors the Pod, ensuring it remains in the desired state.
- The **Kube-Proxy**, running on the Worker Node, sets up networking so the new Pod can: **Communicate with other Pods** in the cluster.
- If **Autoscaling** is enabled, Kubernetes monitors resource usage and can **add or remove Pods automatically** based on traffic.

3. What is a Kubernetes Cluster?

- **A Kubernetes cluster consists of a master node (control plane) and worker nodes to manage and run containers.**

3.1 Services

- Loadbalancing
- Service discovery : labels and selectors

Labels: it will create a label for each pod (In deployment. Yaml we have metadata there we create lable)

Metadata Label

app:paymen

t

- Expose to outside world

3.2 Types of Services

- ClusterIP → Internal use only.
- NodePort → Limited external access.
- LoadBalancer → Fully exposed to the internet.

Example Kubernetes Service Story: Step by Step

1 Cluster IP (Internal Communication)

- Developer deploys the application → Services communicate inside the cluster.
- Example: Backend connects to Database.
- Command: kubectl expose deployment backend --type=ClusterIP --port=80
- Access: backend-service:80 (No external access).

2 Node Port (Limited External Access)

- QA Testers need access from the organization's network.
- Command: kubectl expose deployment frontend --type=NodePort --port=80.
- Access: http://<node-ip>:<node-port> (e.g., http://192.168.1.100:30007).

3 Load Balancer (Public Access)

- Application goes live for users on the internet.
- Command: kubectl expose deployment frontend --type=LoadBalancer --port=80
- Access: http://<public-ip> (Cloud provider assigns a public IP).

4 RBAC (Role-Based Access Control)

You are managing a Kubernetes cluster for a development team. The team includes Developers and QA Testers, and you want to enforce access control based on their roles.

Example

You are managing a Kubernetes cluster for a development team that includes **Developers** and **QA Testers**. You need to enforce access control based on their roles.

- **Developers** should have permissions to create, modify, and delete resources such as Pods, Deployments, and Services within the development namespace.
- **QA Testers** should only have permissions to view logs and check the status of resources but not modify them.

5. What is a Deployment in Kubernetes?

A Deployment manages a set of identical Pods, ensuring the desired number of Pods are running.

6. What is the difference between a Deployment and a Stateful Set?

a. **Deployment: Deployment is used in Kubernetes to manage stateless applications.**

Example

If a pod crashes, Kubernetes replaces it with a new one, but the new pod **does not** retain any previous data.

b. **StatefulSet: A StatefulSet is used for applications that require persistent storage and stable network identities.**

Example

A **MySQL database** where each instance must remember its stored data.

7. What are Kubernetes Namespaces?

Namespaces are logical (Separation) partitions within a Kubernetes cluster to segregate resources and users.

Example

- **dev-team** → Works on development applications.
- **qa-team** → Runs tests in a separate environment.
- **prod-team** → Manages live production applications.

8. What is etcd in Kubernetes?

Etcd is a distributed key-value store that stores Kubernetes cluster data persistently.

9. **What is the role of the API Server in Kubernetes?**

   The API Server is the front-end for the Kubernetes control plane, exposing REST APIs to interact with the cluster.

10. **What is a Service in Kubernetes?**

    A Service defines a logical set of Pods and a policy to access them, enabling communication within or outside the cluster.

11. **What is a ReplicaSet?**

    ReplicaSet ensures a specified number of pod replicas are running at any given time.

12. **What is the difference between a ReplicaSet and a ReplicationController?**

    ReplicaSet Supports both equality-based and set-based selectors

    ReplicationController Ensures a fixed number of pod replicas are running.

13. **What is a ConfigMap in Kubernetes?**

    ConfigMaps are used to store non-confidential data in key-value pairs, such as configuration files or environment variables.

14. **What is a Secret in Kubernetes?**

    Secrets store sensitive information, like passwords or tokens, securely.

15. **What is a DaemonSet?**

    A DaemonSet ensures a copy of a Pod runs on all or some nodes in the cluster.

16. **What is Ingress in Kubernetes?**

    Ingress is an API object that manages external access to services in the cluster, typically HTTP/HTTPS.

16.0 Why Ingress class name

This reduces the risk of traffic being routed incorrectly, especially in complex environments with multiple controllers.

- public-ingress: Handles public-facing services.
- private-ingress: Handles internal services.

16.1 ingress controller

An Ingress Controller is a specialized Kubernetes controller that manages HTTP and HTTPS traffic routing between external clients and services within a Kubernetes cluster

apiVersion: Networking.k8s.io/v1

Kind: Ingress

Metadata:

Name: test-

ingress Spec:

16.2 host-based routing and path-based routing

Here's a comparison between host-based routing and path-based routing in Kubernetes, summarized in a table:

Feature	Host-Based Routing	Path-Based Routing
Definition	Routes traffic based on the host/ HTTP request (e.g., app.example api.example.com).	Routes traffic based on the URL path in the HTTP request (e.g., /app, /api).
Use Case	Useful when hosting multiple applications/services on the same domain or in subdomains.	Useful when a single domain serves multiple paths or services (e.g., API, UI, etc.).
Configuration	Specified using the host field in the Ingress resource.	Specified using the path field in the Ingress resource.
Traffic Separation	Routes traffic to different services based on the hostname.	Routes traffic to different services based on URL paths.
Example	Route app.example.com to app-service and api.example.com to api-service.	Route /app to app-service and /api to apiservice on the same domain (example.com).
Key Advantage	Easy to segregate traffic for different applications using domain names.	Cost-effective, as multiple services can run under the same domain using different URL paths.
Common Scenarios	Multi-tenant applications, microservices with unique subdomains.	API gateways, apps with shared domains, and single-page applications serving different components.

18.3 Load balancer and Ingress when to Use Which?

Scenario	Recommended Approach
You want to expose a single service to the internet.	Use LoadBalancer for direct and simple access.
You have multiple services and need centralized routing.	Use Ingress to manage traffic effectively and reduce costs.

19. **What is Horizontal Pod Autoscaler (HPA)?**

    HPA automatically scales the number of Pods based on resource usage like CPU or memory.

20. **What is the difference between Horizontal Pod Autoscaler and Vertical Pod Autoscaler?**
    - **HPA scales the number of Pods.**
    - **VPA adjusts the resource requests/limits of Pods.**

21. **What is a Persistent Volume (PV) in Kubernetes?**

    A PV is a piece of storage provisioned in the cluster, independent of Pods.

22. **What is a Persistent Volume Claim (PVC)?**

    A PVC is a request for storage by a Pod.

23. **What are Kubernetes Operators?**

    Operators are custom controllers that extend Kubernetes functionality for specific applications.

24. **What is the role of Kube-Proxy?**

    Kube-Proxy manages networking rules to allow communication to and from Pods.

25. **What are taints and tolerations in Kubernetes?**
    - **Taints are applied to nodes to restrict Pod scheduling.**
    - **Tolerations allow Pods to be scheduled on tainted nodes.**

26. **What is a Node in Kubernetes?**

    A Node is a physical or virtual machine where Pods run.

27. **How does Kubernetes achieve high availability?**

    By running multiple master nodes, using etcd replication, and distributing workloads across nodes.

28. **What is a Kubernetes Job?**

    A Job creates one or more Pods to complete a task and ensures the task completes successfully.

29. **What is a CronJob in Kubernetes?**

    A CronJob runs Jobs on a schedule.

    Automates repetitive tasks.
    - ☑ Runs jobs at a specific time or interval.
    - ☑ Ensures jobs are executed even if the system restarts.
    - ☑ Supports retries in case of failures.

30. **What is the difference between a Service and an Ingress?**
    - **Service provides internal/external Pod access.**
    - **Ingress manages HTTP/HTTPS routing for external traffic.**

31. How would you troubleshoot a Pod stuck in Pending state?

- **Check events, node capacity, and resource requests/limits using:**
- **kubectl describe pod <pod\name>**

- **32. How do you debug a crashingPod?**
- **View logs:**
- **kubectl logs <pod-name>**
- **Check events and configurations.**

33. How would you scale a Deployment in Kubernetes?

- **Scale using kubectl:**
- **kubectl scale deployment <deployment-name> --replicas=5**

34. How do you expose a Deployment to external traffic?

- **Use a Service of type LoadBalancer or NodePort.**

35. How do you implement rolling updates in Kubernetes?

- **Update the Deployment:**
- **kubectl set image deployment/<deployment-name> <container-name>=<new-image>**

36. How do you rollback a Deployment in Kubernetes?

Use: ○ **kubectl rollout undo deployment <deployment-name>**

Best Practices for Rollbacks

☑ Always **test new versions** in a **staging environment** before deploying.
☑ Use **readiness probes** to ensure the new version is healthy before full rollout.
☑ Keep an eye on **logs and metrics** after deploying new versions.

37. What happens if a Pod crashes in Kubernetes? Repetedly?

○ **Kubernetes recreates the Pod based on the ReplicaSet/Deployment specification.**

Why

- **The container inside the Pod crashes repeatedly due to an error in the application.**
  Fix the application issue (wrong config, missing dependencies, etc.
- **The Pod is consuming more resources than allowed, causing the OOMKilled (Out of Memory) error or CPU throttling**.
  increase resource limits in the Pod YAML (resources.requests.limits).

- **The Pod fails health checks, causing Kubernetes to kill and restart it**
  Ensure the application **responds properly** to health checks.

38. How do you secure Secrets in Kubernetes?

Use encryption at rest and RBAC to restrict access

39. How do you manage cluster-wide configurations?

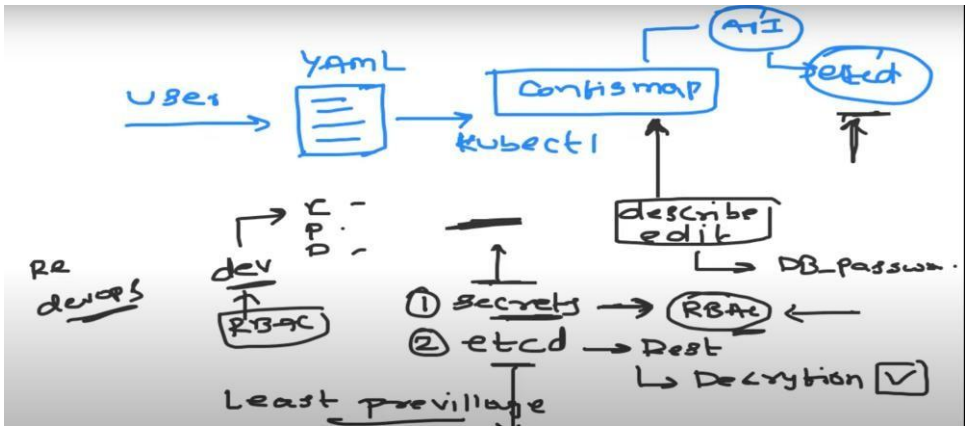
- **Use ConfigMaps, Secrets, and Helm charts.**

40. How do you monitor a Kubernetes cluster?

- **Use tools like Prometheus, Grafana, or Kubernetes Dashboard.**

41. Custom resources

CRD Custom resource Definition
CR
Custom Controller



To see environmental variables for Yamal file (Kubectl exec -it pod name -- /bin/bash)

```
spec:
  containers:
  - name: app-container
    image: nginx
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: app-config
```

CRD (Custom Resource Definition)

A **Custom Resource Definition** (CRD) is a way to extend Kubernetes' API to define your own resource types. It allows you to manage application-specific configurations or abstractions as Kubernetes-native objects, just like built-in resources like Pods, Deployments, or Services.

CR (Custom Resource)

A **Custom Resource** is an instance of a CRD. Once the CRD is defined and registered, you can create resources based on it.

Custom Controller

A **Custom Controller** is a program that watches for changes to your Custom Resources (CRs) and takes action based on the desired state defined in the resource's spec. It ensures the cluster's actual state matches the desired state specified in your CR.

42. Config maps and secretes

Its solving the problem of storing application, data later of time used by application

ConfigMaps vs Secrets in Kubernetes

Feature	ConfigMap	Secret
Purpose	Stores **configuration data** (non-sensitive)	Stores **sensitive data** (passwords, tokens, etc.)
Data Type	Plain text (key-value pairs, files)	Base64 encoded (for security)
Storage	Stored in **etcd** (not encrypted)	Stored in **etcd** (can be encrypted)
Usage Example	Database URL, log level settings	Database password, API keys
Encoding	Stored as plain text	Stored as **base64-encoded** text
Security	Less secure (any user with access can see values)	More secure (requires specific RBAC permissions)