KUBERNETES INTERVIEW QUESTIONS

1. What is Kubernetes?

Kubernetes Is a tool used for managing containers. it manages clusters of container hosts and the containers which are enclosed in a pod that is managed by Kubernetes.

2. What are the benefits of using Kubernetes?

Monitoring-kubernetes monitor the availability of the container in every millisecond and whenever the OS/Container goes down it will launch the identical container and serve the requests coming.
It's portable and 100% open source.
Auto-Scaling-As load of the container increase the k8s launch more containers to handle the client request and as the load decrease the k8s shutdown the containers which causing less resource utilization

3. On which architecture does Kubernetes work?

Kubernetes work on Master-Slaves Architecture. There is a node also called controller node on which k8s is running is called a master node and the container are running on the other node called slaves node.

4. What are the main components of Kubernetes architecture?

- **Master Node**:
  - **API Server**: Acts as a frontend for the Kubernetes cluster, handling RESTful requests.
  - **Scheduler**: Assigns workloads (pods) to worker nodes based on resource availability and policies.
  - **Controller Manager**: Runs controllers to handle tasks like node management, replication, and endpoints.
  - **etcd**: A key-value store that stores cluster state and configuration.
- **Worker Node**:
  - **Kubelet**: Communicates with the master, ensuring containers are running as expected.
  - **Kube-Proxy**: Manages networking for services on the node.
  - **Container Runtime**: Executes containerized applications (e.g., Docker, containerd).

5. What is the role of kubelet (or) Who deploys the PODs?

Kubelet is responsible for deploying the PODs on the worker nodes. It takes the POD definition from the API Server and deploys the PODs on the worker nodes.

6. What is etcd and why is it important in Kubernetes?

etcd is a distributed key-value store used in Kubernetes to store all cluster data. It is critical for maintaining the cluster state, configuration, and metadata. If etcd is compromised, the cluster's state can be lost.

7. What is the role of the Scheduler in Kubernetes?

The Scheduler is responsible for determining on which node a pod should run based on resource requirements, policies, and constraints.

8. What do you mean by a pod in Kubernetes?
pod is the smallest deployable unit of computing you can create and manage in k8s.

9. What is the role of kube-proxy?

Kube-proxy is responsible for the networking in the cluster. It maintains the network rules on the worker nodes and also responsible for the load balancing. This can be done by IPTables. Means if someone tries to access the service, kube-proxy will redirect the request to the appropriate POD.

10. What is the role of CNI in Kubernetes?

By default, K8S doesn't have any networking solution. So, we need to install the CNI plugin to provide the networking solution. CNI is a standard for the container networking. It provides the networking solution to the PODs in the cluster. There are multiple CNI plugins available like Calico, Flannel, Weave, etc. IF we use AWS EKS, it uses AWS VPC CNI

11. Who is providing IP address to the PODs?

CNI plugin is responsible for providing the IP address to the PODs. It assigns the IP address to the PODs from the subnet provided by the CNI plugin.

12. What are Namespaces in Kubernetes?

Namespaces provide a way to divide cluster resources among multiple users or teams. They act as virtual clusters within the same physical cluster.

13. Explain the role of Controllers in Kubernetes.

Controllers maintain the desired state of the cluster. Examples include:

Replication Controller: Ensures a specified number of replicas are running.

Node Controller: Manages node status.

Endpoint Controller: Updates endpoints in services.

14. Does two PODs in different namespaces can communicate with each other (or) How to communicate between the PODs in different namespaces?

By default PODs in different namespaces can communicate with each other with IP address. In general we dont do in that way. PODs are exposed using the services. So, we need to create the service with the type ClusterIP and then we can access the PODs in different namespaces.

15. What will happen if kubelet is down on the worker node?

If kubelet is down on the worker node, then the PODs on that worker node will be in the NotReady state. The kubelet is responsible for deploying the PODs on the worker nodes. If kubelet is down, then the PODs will not be deployed on the worker nodes. The replication controller will try to deploy the PODs on the worker nodes but it will not be successful. To resolve this issue, we need to restart the kubelet service on the worker node or we need to restart the worker node and if it is still not working, then we can drain the node for further investigation.

16. What are the default namespaces in Kubernetes?

Kubernetes has four default namespaces:

1. **default**: Used for resources if no other namespace is specified.
2. **kube-system**: Contains system components like the API Server, Scheduler, and DNS.
3. **kube-public**: Holds publicly readable data, such as cluster info.
4. **kube-node-lease**: Used for tracking node heartbeats to manage cluster health efficiently.

17. What is the least primitive object in Kubernetes?

POD => ReplicaSet => Deployment/DaemonSet/StatefulSet => Service
Based on this the lowest primitive object in K8S is POD.

18. How many contaniners I can run in a POD?

We can run n number of containers in a POD

19. How to check the IP Address of the PODs and the node its running?

kubectl get pods -o wide

20. What are Kubernetes resources?

Kubernetes resources are objects in the Kubernetes API that represent the state and configuration of the system, such as Pods, Deployments, Services, ConfigMaps, and more. These resources define what applications run on the cluster, how they run, and how they interact.

21. What are the key Kubernetes resources used to manage applications?

- **Pods**: The smallest deployable unit in Kubernetes.
- **ReplicaSets**: Ensures a specific number of pod replicas are running.
- **Deployments**: Manages ReplicaSets and ensures declarative updates.
- **Services**: Exposes Pods to external/internal traffic.
- **ConfigMaps**: Stores non-confidential configuration data.
- **Secrets**: Stores sensitive information like passwords.

22. What is the difference between ConfigMap and Secret?

- **ConfigMap**: Stores configuration data in plain text.
- **Secret**: Stores sensitive data like passwords and tokens in base64-encoded format.

23. How do you check the logs of the POD?

kubectl logs POD_NAME

24. How to check the logs of the POD which is running in the different namespace?

kubectl logs POD_NAME -n NAMESPACE

25. How can I copy a file to a running container/Pod?

kubectl cp /path/to/local/file <pod-name>:/path/in/container

25. What is the difference between kubectl create and kubectl apply?

kubectl create is used to create resources for the first time, while kubectl apply is used to create or update resources declaratively.

26. How to go inside the POD and execute the commands?
kubectl exec -it POD_NAME -- /bin/bash

27.Imagine there are two containers in a POD. How they communicate with each other?

Containers in the same Pod communicate via **localhost** since they share the same network namespace and IP address. They can also share data using **shared volumes** mounted in the Pod.

28. What is the difference between Deployment, DaemonSet, StatefulSet?

- **Deployment**

- **Purpose**: Manages stateless applications.
- **Use Case**: Web servers or APIs.
- **Key Features**:

- Ensures a specific number of identical pods.
- Supports scaling and rolling updates.
- Pods are replaceable and don't retain identity.

DaemonSet

- **Purpose**: Ensures one pod runs on every node (or selected nodes).
- **Use Case**: System-level services like logging or monitoring agents.
- **Key Features**:
  - Automatically adds/removes pods as nodes join/leave the cluster.
  - Designed for uniform node coverage.

StatefulSet

- **Purpose**: Manages stateful applications with unique pod identities.
- **Use Case**: Databases or applications needing persistent storage.
- **Key Features**:
  - Ensures ordered and unique pod deployment.
  - Provides stable network IDs and persistent storage.

29. **How to deploy a POD on a specific node?**

We can use nodeSelector (Simple Method)

```
spec:
  nodeSelector:
    kubernetes.io/hostname: <instance_id>
```

30. How do you expose a deployment as a service?

kubectl expose deployment <deployment-name> --type=<service-type> --port=<port>

31. What are the types of Kubernetes Services?

Kubernetes supports four types of Services:

ClusterIP: Exposes the Service on an internal IP address within the cluster. It is the default type.

NodePort: Exposes the Service on each Node's IP at a static port.

LoadBalancer: Exposes the Service externally using a cloud provider's load balancer.

ExternalName: Maps a Service to a DNS name, redirecting traffic outside the cluster.

32. How does a Service select Pods?

Services use **label selectors** to match Pods. Only the Pods with labels matching the selector will be targeted by the Service.

```
selector:
```

```
app: myapp
```

33. What is the difference between NodePort and LoadBalancer?

- NodePort: Exposes the Service on a static port on each Node's IP. Access is via <NodeIP>:<NodePort>. It is less flexible and not ideal for production.
- LoadBalancer: Provisions a load balancer to expose the Service to external traffic. It automatically handles traffic distribution across Nodes.

34. What is a Headless Service?

A Headless Service (created by setting clusterIP: None) does not allocate a ClusterIP. Instead, it directly returns the IP addresses of the associated Pods, enabling direct Pod-to-Pod communication.

35. **How do you verify if a Kubernetes Service is connected to its Pods?**

- Run kubectl describe service <service-name> to check the Service's selector and endpoints.
- Use kubectl get endpoints <service-name> to see the Pods' IPs listed as endpoints.
- Ensure the Pods have the correct labels matching the Service's selector using kubectl get pods --show-labels.
- Test connectivity from a test Pod to the Service's ClusterIP and port.

36. How can you debug if a Service is not working?

1. Check Service Details: Use kubectl describe service <service-name> to confirm the selector and endpoints.
2. Check Endpoints: Use kubectl get endpoints <service-name> to verify that the Pods are listed.
3. Inspect Pod Labels: Run kubectl get pods --show-labels to ensure the Pods have labels matching the Service selector.
4. Test Connectivity: Create a test Pod and use tools like wget or curl to ping the Service.
5. Logs and Events: Check kube-proxy logs and cluster events for errors.

37. What are common tools used to test Kubernetes Service connectivity?

- curl: To test HTTP/HTTPS endpoints.
- wget: To fetch content from the Service.
- telnet: To test if the Service port is open.
- nslookup or dig: To resolve the Service DNS.

38. What is an Ingress Controller? Why is it required?
An Ingress Controller is a Kubernetes component that implements the rules defined in Ingress resources. Kubernetes itself does not process Ingress resources; an Ingress Controller like NGINX, Traefik, or AWS ALB is required to handle traffic routing.

39. Can an Ingress resource route traffic to multiple namespaces?
No, an Ingress resource is namespace-scoped. To route traffic to Services in other namespaces, you need to either replicate the Ingress in each namespace or use a multi-namespace Ingress Controller with specific annotations.

40. What are the key components of RBAC in Kubernetes?

The key components of RBAC in Kubernetes are:

1. Role: Defines a set of permissions (verbs) to access resources within a namespace.
2. ClusterRole: Similar to Role but applies cluster-wide, not just to a namespace.
3. RoleBinding: Associates a user or group with a Role within a specific namespace.
4. ClusterRoleBinding: Associates a user or group with a ClusterRole, giving them permissions across the entire cluster.

41. What is the difference between a Role and a ClusterRole in RBAC?

- Role: Defines permissions within a specific namespace. It grants access to resources only in that namespace.
- ClusterRole: Defines permissions that are not limited to a namespace and can be applied cluster-wide. It can be used for both namespaced and non-namespaced resources.

42. How do you create an RBAC Role?

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- verbs: ["get", "list"]
  apiGroups: [""]
  resources: ["pods"]
```

43. What are some common verbs in Kubernetes RBAC rules?

- **get**: Read a resource.
- **list**: List resources.
- **create**: Create a new resource.
- **update**: Update an existing resource.
- **delete**: Delete a resource.
- **patch**: Modify a resource partially.
- **watch**: Watch a resource for changes.
- **deletecollection**: Delete multiple resources at once.

44. How do you restrict access to specific resources using RBAC?
You can restrict access to specific resources by defining **Roles** with very specific permissions (verbs and resources) and using **RoleBindings** or **ClusterRoleBindings** to assign the role to specific users or service accounts. For example, you can give read-only access to Pods but not to Deployments by defining appropriate rules in a Role.

45. Can RBAC be used to control access to the Kubernetes API server?
Yes, RBAC is used to control access to the Kubernetes API server. By assigning roles and binding them to users, service accounts, or groups, RBAC controls which users can access and interact with the Kubernetes API server and which actions they can perform (e.g., viewing resources, creating/deleting resources).

46. How can you use RBAC to secure a Kubernetes cluster?
RBAC helps secure a Kubernetes cluster by:

- Enforcing **least privilege** access: Granting users only the permissions they need.
- Using **namespaces** to isolate resources and manage access separately for different teams.
- Ensuring **service accounts** have limited access to resources, reducing the potential for malicious actions.

47. What is the difference between a ConfigMap and a Secret?

- **ConfigMap**: Stores non-sensitive data such as configuration files, environment variables, etc., in plain text.
- **Secret**: Stores sensitive data (e.g., passwords, tokens) and is encoded (typically base64). Secrets are handled more securely by Kubernetes.

48. **How do you use ConfigMaps and Secrets in a pod?**
You can use both ConfigMaps and Secrets as environment variables or mount them as files inside a Pod.

Configmaps:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: nginx
    envFrom:
    - configMapRef:
        name: my-config
```

Secrets:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: nginx
    envFrom:
    - secretRef:
        name: my-secret
```

49. How is sensitive data handled in Secrets?

Kubernetes stores Secret data in base64 encoding to protect it from easy readability. However, base64 encoding is not encryption. Kubernetes also ensures Secrets are only accessible to users or pods with the appropriate permissions.

50. How do you view and delete a ConfigMap or Secret in Kubernetes?

- To view a ConfigMap:

  kubectl get configmap <configmap-name> -o yaml

- To view a Secret (encoded):

  kubectl get secret <secret-name> -o yaml

- To decode and view the data of a Secret:

  kubectl get secret <secret-name> -o jsonpath='{.data.<key>}' | base64 --decode

- To delete a ConfigMap:

  kubectl delete configmap <configmap-name>

- To delete a Secret:

kubectl delete secret <secret-name>

51. What is the difference between a PersistentVolume (PV) and a PersistentVolumeClaim (PVC)?

- PersistentVolume (PV): A piece of storage in the cluster provisioned by an administrator. It is a resource in Kubernetes that abstracts storage details.
- PersistentVolumeClaim (PVC): A request by a user for storage. PVCs request storage resources from PVs and bind to a suitable PV when available.

52. What are Liveness and Readiness probes in Kubernetes?

- **Liveness Probe**: Checks if the container is still running. If the probe fails, Kubernetes will restart the container.
- **Readiness Probe**: Checks if the container is ready to handle traffic. If it fails, the Pod will be excluded from the Service load balancer until it passes.

53. What is a StorageClass in Kubernetes?
A **StorageClass** in Kubernetes defines the types of storage available in the cluster. It allows users to request dynamic provisioning of PersistentVolumes (PVs). Different classes can specify different types of underlying storage, like SSDs, HDDs, or network-attached storage (NAS).

54. How does the Kubernetes Horizontal Pod Autoscaler (HPA) work?
The Horizontal Pod Autoscaler (HPA) scales the number of Pods based on observed metrics like CPU utilization or custom metrics. It continuously monitors the resource usage of Pods, and when the resource usage crosses a defined threshold (e.g., 50% CPU utilization), it increases or decreases the number of Pods to maintain performance

55. What is the difference between ReadWriteOnce, ReadWriteMany, and ReadOnlyMany?

1. **ReadWriteOnce (RWO)**:
   - The volume can be mounted as **read-write** by **one node** only.
   - Use case: Single-instance applications like databases.
2. **ReadWriteMany (RWX)**:
   - The volume can be mounted as **read-write** by **multiple nodes**.
   - Use case: Shared storage for applications like CMS or collaborative tools.
3. **ReadOnlyMany (ROX)**:
   - The volume can be mounted as **read-only** by **multiple nodes**.
   - Use case: Serving static files or shared read-only data.

56. What are the 3 types of checks used by Probes in Kubernetes?

1. **httpGet**:
   Sends an HTTP GET request to a specific path on the container to check if it responds correctly.
2. **tcpSocket**:
   Checks if a specific port on the container is reachable.
3. **exec**:
   Runs a command inside the container and checks if it executes successfully.

57. What are the different metrics used in Horizontal Pod Autoscaler (HPA)?

Horizontal Pod Autoscaler (HPA) can use the following metrics:

1. **CPU**: Monitors the CPU usage of Pods.
2. **Memory**: Monitors the memory usage of Pods.
3. **Custom Metrics**: Monitors application-specific or external metrics (e.g., request rate, queue length) using tools like Prometheus.

58. What is Helm, and why is it used in Kubernetes?
Helm is a package manager for Kubernetes that simplifies the deployment and management of applications. It uses **charts**, which are reusable templates, to deploy applications consistently across environments.

Benefits:

- Simplifies complex Kubernetes configurations.
- Supports versioning and rollbacks of releases.
- Enables parameterization for dynamic deployments.

59. What is a Helm repository?
A Helm repository is a collection of packaged Helm charts.

- Popular repositories:
  - stable (now deprecated).
  - bitnami (widely used for pre-configured charts).
- Commands to work with repositories:

```
helm repo add <repo-name> <repo-url>

helm repo update

helm search repo <chart-name>
```

60. Can you explain the Helm Chart structure?

A Helm chart is a collection of files and directories that define a Kubernetes application.

my-chart/

├── Chart.yaml # Metadata about the chart

├── values.yaml # Default configurations

├── templates/ # Kubernetes resource templates

└── README.md # Documentation

---

my-chart/

├── Chart.yaml

├── values.yaml

├── templates/

│ ├── deployment.yaml

│ ├── service.yaml

│ ├── _helpers.tpl

│ └── other-resource.yaml

├── charts/

└── README.md

61. What are the different deployment strategies in Kubernetes?
Kubernetes supports the following deployment strategies:

1. **Rolling Update**:
   - Gradually replaces old Pods with new ones.
   - Ensures zero downtime by updating Pods incrementally.
   - Default deployment strategy.
   - Command:

```
kubectl rollout restart deployment <deployment-name>
```

2. **Recreate**:
   - Stops all old Pods before starting new ones.
   - Suitable when downtime is acceptable or necessary (e.g., incompatible updates).
   - Command:

```
strategy:
  type: Recreate
```

3. **Blue-Green Deployment**:
   - Maintains two environments: one live (blue) and one staging (green).
   - Traffic switches to the green environment after validation.
   - Often managed externally (e.g., using Ingress or Service updates).
4. **Canary Deployment**:
   - Gradually rolls out the update to a subset of users before a full rollout.
   - Allows testing in production with minimal impact.

62. How does the Rolling Update strategy work in Kubernetes?

In a **Rolling Update**, Pods are replaced incrementally to ensure the application remains available.

- Controlled by the maxUnavailable and maxSurge fields:
  - maxUnavailable: Maximum number of Pods that can be unavailable during the update.
  - maxSurge: Maximum number of extra Pods that can be created during the update. Example configuration:

```
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 1
    maxSurge: 1
```

63. How do you roll back a failed deployment in Kubernetes?

```
kubectl rollout undo deployment <deployment-name>
```

Check the rollout history:

```
kubectl rollout history deployment <deployment-name>
```

64. What is ArgoCD?

ArgoCD is a declarative, GitOps-based continuous delivery tool for Kubernetes. It continuously synchronizes the desired application state defined in a Git repository with the actual state of the Kubernetes cluster.

- Key features:
  - Automates deployment of Kubernetes resources.
  - Ensures the cluster state matches the Git repository state.
  - Supports multi-cluster environments & Provides a web UI for monitoring and managing deployments.

65. What is GitOps, and how does ArgoCD implement it?

- **GitOps**: A DevOps practice where Git is the single source of truth for application and infrastructure configurations.
- **ArgoCD Implementation**:
  - Monitors Git repositories for changes.
  - Applies Kubernetes manifests (e.g., YAML/Helm charts) to clusters.
  - Provides real-time feedback on synchronization status.

66. How does ArgoCD manage application synchronization?

ArgoCD continuously compares the desired state in the Git repository with the actual state in the Kubernetes cluster.

- **Sync Status**:
  - **Synced**: Desired state matches the cluster state.
  - **OutOfSync**: Cluster state differs from the desired state (e.g., changes in Git).
- Manual or automated sync policies can be applied.

67. What are the key components of ArgoCD?

- **API Server**: Provides the REST API for managing applications.
- **Controller**: Monitors Git repositories and ensures cluster state matches the desired state.
- **Repository Server**: Handles Git operations (e.g., cloning, fetching manifests).
- **Web UI/CLI**: Provides tools for managing and visualizing application states.

68. What is the difference between "Synced" and "Healthy" states in ArgoCD?

- **Synced**: The cluster state matches the desired state in Git.
- **Healthy**: The application is functioning as expected (e.g., all Pods are running, and services are operational).
- An application can be synced but not healthy if resources fail to deploy properly.

69. How can you integrate ArgoCD with CI/CD pipelines?
ArgoCD integrates seamlessly with CI/CD pipelines by:

- Triggering Git commits that ArgoCD automatically syncs.
- Using ArgoCD CLI commands in pipeline scripts for manual control.

Scenario-Based Questions

1. **How does Kubernetes handle node failure?**
   - When a node fails:
     - The Node Controller detects the failure.
     - The Pods on the failed node are marked as "NotReady."
     - The Scheduler reschedules the Pods on available nodes.
2. **How do you scale applications in Kubernetes?**
   - Applications can be scaled using:
     - **kubectl scale command**: kubectl scale deployment <deployment-name> --replicas=<number>
     - **Horizontal Pod Autoscaler**: Adjusts the number of replicas based on CPU/memory usage.
     - **Vertical Pod Autoscaler**: Adjusts resource limits for Pods.

Debugging Questions

1. **How do you troubleshoot a Pod stuck in the Pending state?**
   - Check events: kubectl describe pod <pod-name>

- Verify node capacity: Ensure the cluster has enough resources.
- Inspect network policies or quotas.

2. **What would you do if the API Server becomes unavailable?**
    - Check the status of the master node components.
    - Verify the health of etcd, as the API Server depends on it.
    - Restart the API Server or troubleshoot logs for issues.