Kubernetes

What is Kubernetes, and what are its main components?

Answer:
Kubernetes is an open-source container orchestration platform designed to automate deploying, scaling, and managing containerized applications.

Its main components include:

- **Master Node:** Manages the Kubernetes cluster and includes components like the API server, scheduler, and controller manager.
- **Worker Nodes:** Run containerized applications and include components like kubelet, kube-proxy, and container runtime.
- **Pods:** The smallest deployable units in Kubernetes, which can contain one or more containers.
- **Services:** Define policies for accessing and managing pods.
- **Deployments:** Manage and scale pods and replicas.
- **ConfigMaps and Secrets:** Manage configuration and sensitive data.

How does Kubernetes ensure high availability and fault tolerance?

Answer:
Kubernetes ensures high availability and fault tolerance through:

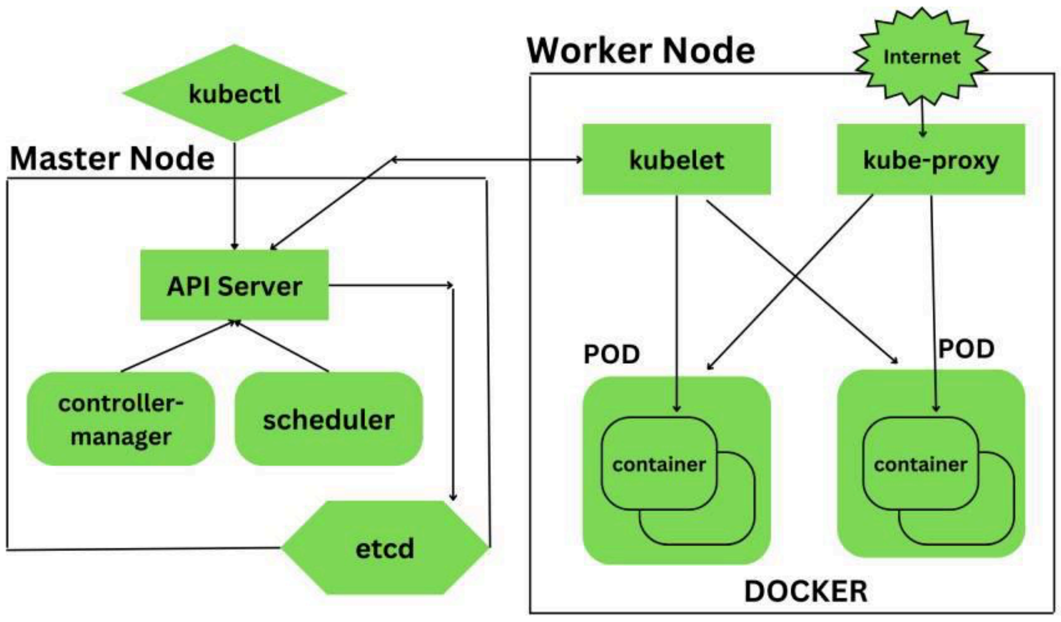
- **Replication:** Using Deployments and ReplicaSets to ensure multiple replicas of Pods.
- **Pod Distribution:** Spreading Pods across nodes to prevent single points of failure.
- **Health Checks:** Implementing liveness and readiness probes to detect and replace unhealthy Pods.
- **Automatic Rescheduling:** Restarting Pods on different nodes if a node fails.

What is a Kubelet?

Answer:
Kubelet is an agent on each worker node that:

- runs containers
- reports node health
- communicates with control plane

1. Explain Kubernetes Architecture.



“Kubernetes has a clean separation between the control plane and the data plane.

The **control plane** is responsible for making all cluster-level decisions.
It includes:

- **API Server**, which acts as the front door of the cluster. Every `kubectl` command hits the API server first.
- **etcd**, **which is the distributed key-value store where the entire cluster state is stored.**
- **Scheduler**, which decides *where* pods should run based on resource requirements and constraints.
- **Controller Manager**, which constantly checks the cluster’s actual state versus desired state and tries to reconcile differences.

The **data plane** runs on worker nodes and it hosts actual applications.
It includes:

- **Kubelet**, which ensures that the containers running on that node match the specs sent from the API server.
- **Kube-proxy**, which maintains networking rules so services can talk to pods.
- And the **container runtime**, like containerd or CRI-O, which actually runs the containers.

In simple terms:
Control plane = brain
Data plane = muscle.”

---

2. What is container orchestration?

Answer:

“Container orchestration means automating everything needed to run containers at scale — deployment, scaling, networking, health checks, and failover.

Instead of manually starting containers or monitoring them, a platform like Kubernetes keeps everything in the desired state.

For example, if a node dies, Kubernetes automatically reschedules pods elsewhere.
If load increases, the cluster auto-scales.

So orchestration basically ensures that the application is always running reliably without manual intervention.”

---

3. Difference between a Pod and a Container.

Answer:

A Pod is the smallest and simplest Kubernetes object. It represents a single instance of a running process in the cluster and can encapsulate one or more containers.

Pods share the same network namespace and can communicate with each other using localhost.

“A container is just a single isolated process.
A Pod is a higher-level abstraction in Kubernetes.

A Pod can contain one or more containers that are tightly coupled and must share:

- the same IP address
- the same network namespace
- and storage volumes.

Kubernetes doesn’t schedule individual containers — it schedules Pods.
So, Pods are the smallest deployable unit in K8s, not containers.”

---

4. How does Kubernetes handle scaling?

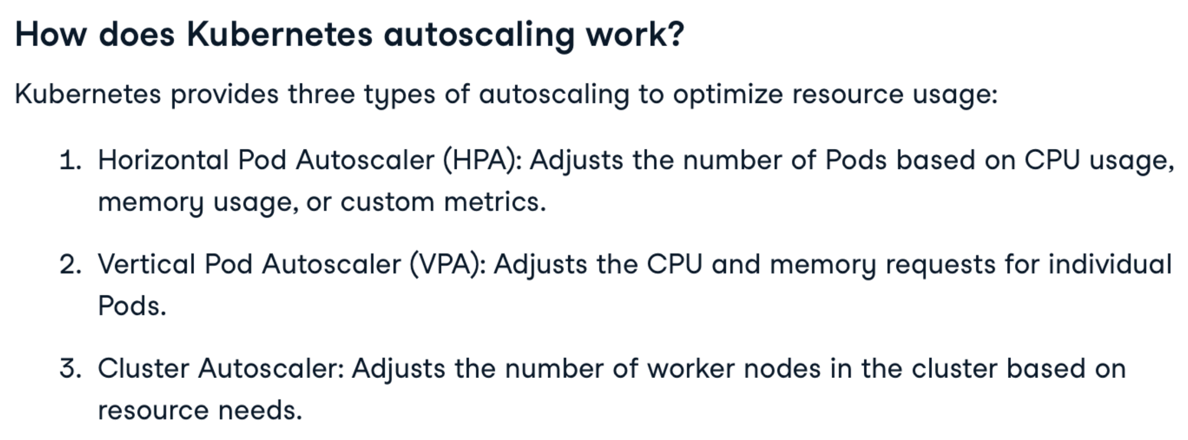
Answer:

“Kubernetes handles scaling using the Horizontal Pod Autoscaler.

The HPA constantly monitors metrics like CPU, memory, or custom business metrics.
Based on the load, it automatically increases or decreases the number of pod replicas.

For example, if CPU usage goes above the defined threshold, Kubernetes will spin up additional pods.
If the load drops, it scales the pods down to save resources.

This makes applications elastic and capable of handling variable traffic automatically.”



How does Kubernetes autoscaling work?

Kubernetes provides three types of autoscaling to optimize resource usage:

1. Horizontal Pod Autoscaler (HPA): Adjusts the number of Pods based on CPU usage, memory usage, or custom metrics.
2. Vertical Pod Autoscaler (VPA): Adjusts the CPU and memory requests for individual Pods.
3. Cluster Autoscaler: Adjusts the number of worker nodes in the cluster based on resource needs.

---

5. What is a Sidecar container?

“A sidecar container is an additional container that runs inside the same Pod as the main application container.
Its purpose is to enhance or support the main app without being part of the core logic.

Typical examples include:

- log shippers
- metrics collectors
- service mesh proxies
- TLS helpers
- config sync agents

Since both containers share the same network namespace, the sidecar can communicate with the main app very easily.”

---

6. Difference between StatefulSet and Deployment.

A Deployment is used for stateless applications where all pods are identical and interchangeable. It doesn’t matter which pod handles a request because none of them maintain their own identity or data. If a pod dies, Kubernetes can recreate another pod anywhere.

A StatefulSet is used for stateful applications that need a stable identity and persistent storage. Each pod gets a unique name, ordered startup and shutdown, and its own persistent volume. If a pod dies, Kubernetes brings it back with the same identity and data.

In short:

- Deployment = stateless, interchangeable pods
- StatefulSet = stateful, uniquely identifiable pods with persistent data

Explain the role of a Deployment in Kubernetes.

Answer:
A Deployment is a Kubernetes resource that provides declarative updates to applications. It manages the creation, scaling, and rolling updates of Pods.
With a Deployment, you can ensure that a specified number of pod replicas are running at all times and manage updates to those pods without downtime.

What is a StatefulSet and how does it differ from a Deployment?

Answer:
A StatefulSet is a Kubernetes resource designed for managing stateful applications.
Unlike Deployments, StatefulSets provide stable, unique network identities and persistent storage for Pods.
They ensure the ordering and uniqueness of Pods, which is crucial for stateful applications like databases.

---

7. Relationship between Deployment, ReplicaSet, and Pod.

“A Deployment is the highest-level object — it defines what the application should look like: image, number of replicas, update strategy.

A ReplicaSet works underneath the Deployment. Its job is simply to ensure the specified number of pods are running.

Pods are the actual running instances of the application.

When I update a Deployment — for example, change the image — Kubernetes creates a new ReplicaSet and gradually scales pods up and down to perform a rolling update.
The old ReplicaSet is retained with zero replicas, so rollback becomes easy.”

---

8. What are ConfigMaps and Secrets?

“ConfigMaps and Secrets are used to externalize configuration — but for different types of data.

- **ConfigMaps** store non-sensitive data like config files, environment variables, flags.
- **Secrets** store sensitive information like passwords, tokens, or certificates.

Secrets are base64-encoded and have stricter RBAC permissions, while ConfigMaps are plain text.

The idea is separating configuration from code, and most importantly, keeping sensitive data under tighter security.”

---

9. Difference between Deployment, StatefulSet, and DaemonSet.

“Deployments are for stateless apps — web servers, APIs — where pods are identical and can run anywhere.

StatefulSets are for stateful apps where each pod needs a fixed identity and persistent storage — like databases or Kafka brokers.

DaemonSets ensure that one pod runs on *every* node in the cluster.
They’re used for things like log collectors or monitoring agents that must run cluster-wide.

So:

- Deployment → stateless
- StatefulSet → stateful
- DaemonSet → one pod per node”

---

10. Explain Liveness, Readiness, and Startup Probes.

“Kubernetes uses three probes to manage container health.

Liveness probe checks whether the container is alive.
If liveness fails, Kubernetes restarts the container. This prevents deadlocks.

Readiness probe checks if the app is *ready to receive traffic*.
If it fails, Kubernetes removes the pod from the service’s endpoint list.

Startup probe is used for applications that take a long time to initialize.
Until this probe passes, Kubernetes won’t run liveness or readiness checks, preventing premature restarts.

Together, these probes ensure apps start correctly, stay alive, and only receive traffic when they're actually ready.

11. Explain the concept of Ingress in Kubernetes.

"Ingress is how Kubernetes exposes HTTP and HTTPS traffic from outside the cluster to services inside the cluster.

The Ingress object itself only defines **Layer 7 routing rules**, like:

- send `api.example.com` to the API service
- send `/users` path to the user service
- send `/orders` path to the orders service

But Ingress alone doesn't actually route anything.
You also need an **Ingress Controller**, like NGINX, Traefik, or HAProxy.
The controller acts as the actual reverse proxy or load balancer and enforces the routing rules defined in the Ingress object.

So in simple terms:

- **Ingress = rules**
- **Ingress Controller = engine that applies those rules**

Example:
If I define an Ingress saying `/payments` should go to the payments service, the NGINX Ingress Controller will update its internal routing to make that happen."

---

12. How does Kubernetes enforce communication boundaries between Pods?

"Kubernetes uses **Network Policies** to enforce communication boundaries.

By default, all Pods can talk to each other freely.
But when you apply a Network Policy, you can restrict traffic just like a firewall — based on IP, port, or Pod labels.

This helps implement a zero-trust model inside the cluster.

In the broader networking stack:

- **Services** handle internal load balancing.
- **Ingress** handles external HTTP/HTTPS routing.
- **Network Policies** handle security — who can talk to whom.

This layered architecture allows for separation of concerns:

- Developers create Services.
- Platform teams manage Ingress.
- Security teams define Network Policies.”

---

13. Describe the role of etcd in Kubernetes.

etcd is basically the **database of Kubernetes**.
It stores every piece of cluster data — both the desired state and the real-time state.

It stores things like:

- what Deployments exist
- how many replicas you want
- state of pods and nodes
- ConfigMaps and Secrets
- cluster metadata

Kubernetes components constantly read and write to etcd through the API server.

If etcd goes down, the control plane can’t make decisions — because it has no state to work with.
That’s why etcd is always kept highly available and backed up.”

---

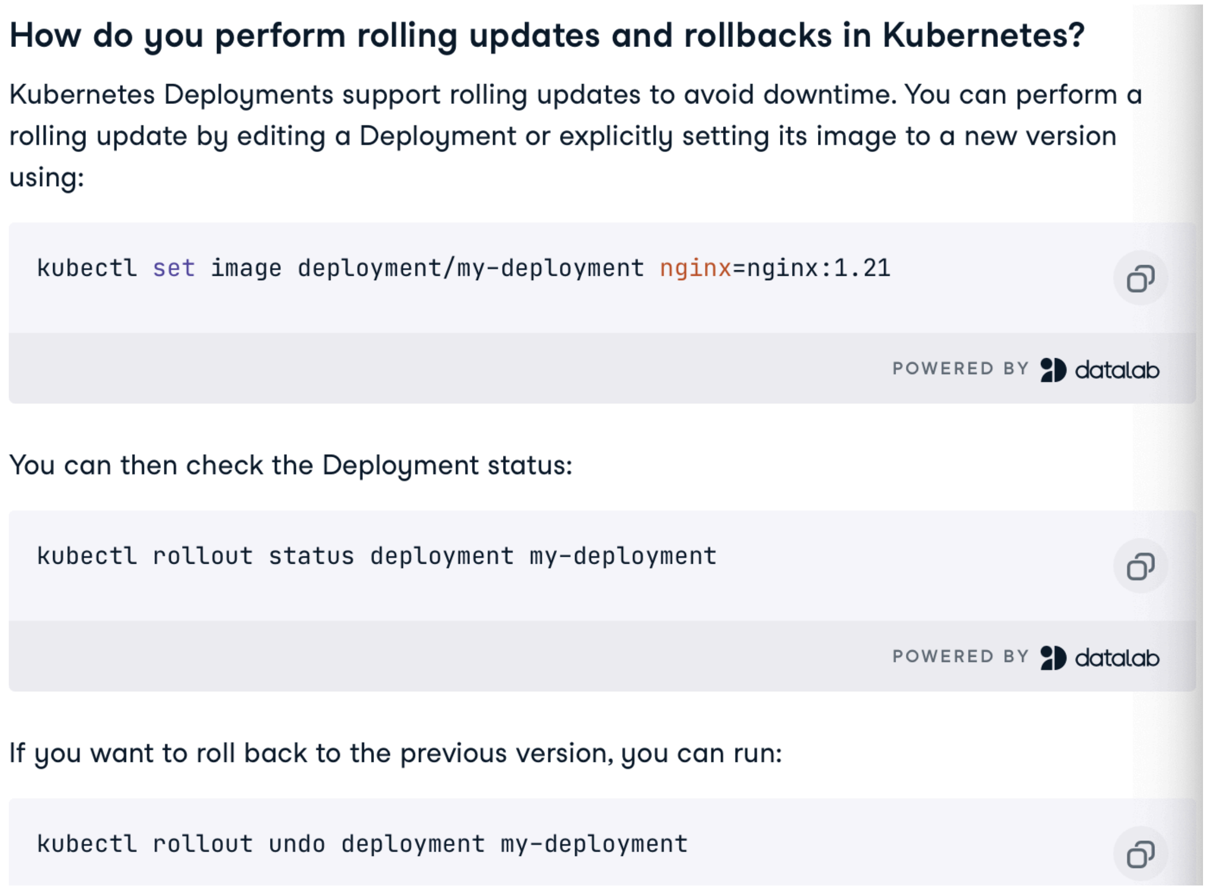
14. How do rolling updates work in a Deployment?

“In a rolling update, Kubernetes updates your application gradually, without downtime.

Here’s what happens:

1. The Deployment creates a **new ReplicaSet** with the new configuration — usually a new container image.
2. It starts scaling **up** the new ReplicaSet and scaling **down** the old one.
3. This continues until all Pods are updated.
4. The old ReplicaSet remains with zero replicas, so rollback is easy.

This ensures continuous availability during the update.”



How do you perform rolling updates and rollbacks in Kubernetes?

Kubernetes Deployments support rolling updates to avoid downtime. You can perform a rolling update by editing a Deployment or explicitly setting its image to a new version using:

```
kubectl set image deployment/my-deployment nginx=nginx:1.21
```

You can then check the Deployment status:

```
kubectl rollout status deployment my-deployment
```

If you want to roll back to the previous version, you can run:

```
kubectl rollout undo deployment my-deployment
```

15. You’re managing a Kubernetes cluster shared by multiple teams working on different projects. How would you isolate their resources and avoid naming conflicts?

“I would isolate teams using **Namespaces**.

Namespaces act as virtual clusters inside a real cluster. They allow to divide a single cluster into virtual sub-clusters, each with their own scope for resources like Pods, Services, and ConfigMaps.
They prevent naming conflicts and allow fine-grained access control.

Benefits:

- **Isolation** — each team gets its own workspace.
- **No naming conflicts** — `backend-service` can exist in many namespaces.
- **RBAC** — I can control who can access what.
- **Resource quotas** — I can limit CPU, memory, and number of objects per team.

This is the recommended way to divide a cluster across teams or environments.”

Explain the concept of Namespaces in Kubernetes.

Answer:
Namespaces are a way to partition resources in a Kubernetes cluster.
They provide a mechanism for isolating resources such as Pods, Services, and Deployments within a single cluster.
This helps in organizing and managing resources for different teams or projects.

---

16. Explain Labels and Selectors in Kubernetes.

Labels and Selectors are how Kubernetes connects components together.

- **Labels** are key-value pairs added to resources like Pods.
- **Selectors** are how other resources — like Services or Deployments — find those Pods.

For example, a Service uses a selector like `app: frontend` to know which Pods to send traffic to.

So labels identify Pods, and selectors match them.
This is how Deployments track their Pods and Services route traffic to the right workload."

---

17. Describe the role of Kube-Proxy in Kubernetes.

Kube-proxy runs on every node and handles cluster networking for Services.

Its job is to:

- watch for changes to Services and Endpoints
- convert those into real network rules
- make sure traffic gets load-balanced correctly to Pods

Depending on the cluster setup, it may configure iptables, IPVS, or userspace proxies.

Basically, kube-proxy ensures that when I hit a Service, traffic is correctly routed to an available Pod.

---

18. What are Persistent Volumes and PersistentVolumeClaims?

“Kubernetes separates storage into two parts: PVs and PVCs.

- **Persistent Volume (PV)**
  This is the actual storage resource in the cluster. It’s like a disk.
  It can be created by an admin or dynamically provisioned.
- **Persistent Volume Claim (PVC)**
  This is a request for storage by a user or application. It is similar to a Pod requesting CPU or memory.
  A PVC asks for size and access mode, and Kubernetes binds it to a matching PV.

This abstraction makes storage independent of the application — just like how Pods request CPU and memory.”

---

19. Difference between a DaemonSet and a ReplicaSet.

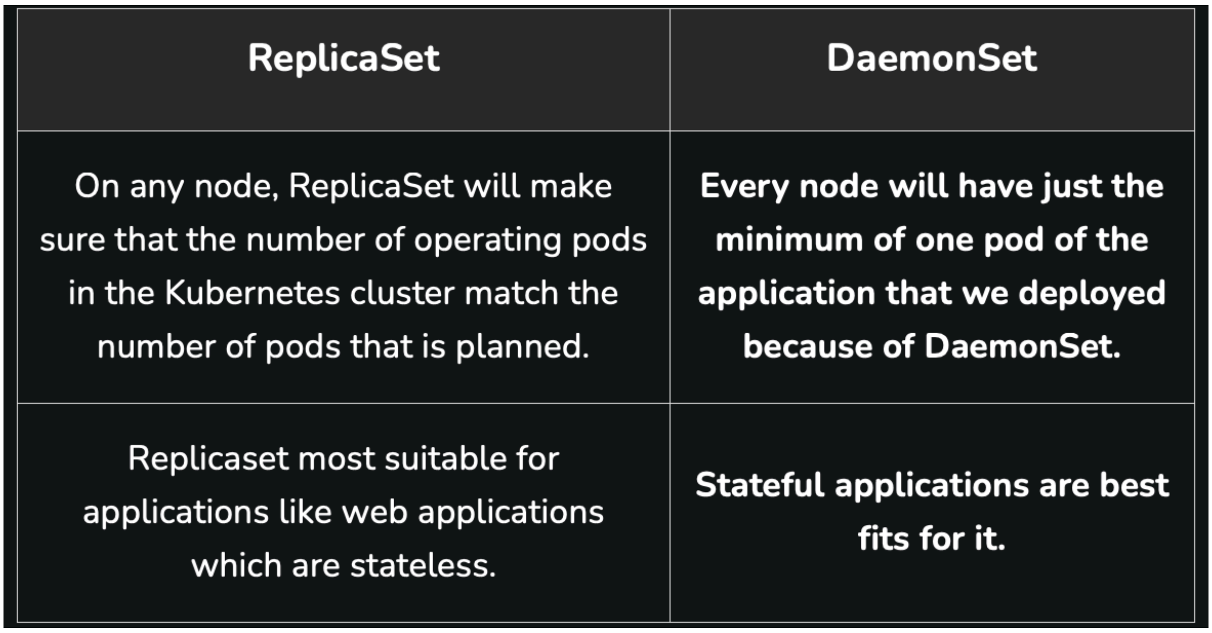
A ReplicaSet ensures a specific number of Pods are running in the cluster, regardless of nodes.
For example, you might want 5 replicas of your web service, and they can run on any nodes.

A DaemonSet ensures **one Pod per node** (or per selected nodes).
This is used for cluster-wide agents like:

- log shippers (Fluentd)
- monitoring agents (Prometheus node exporter)
- storage daemons

So:

- ReplicaSet = scale across cluster
- DaemonSet = one pod per node
- ReplicaSet is for apps
- DaemonSet is for cluster-level agents



ReplicaSet	DaemonSet
On any node, ReplicaSet will make sure that the number of operating pods in the Kubernetes cluster match the number of pods that is planned.	**Every node will have just the minimum of one pod of the application that we deployed because of DaemonSet.**
Replicaset most suitable for applications like web applications which are stateless.	**Stateful applications are best fits for it.**

---

20. How do Pods on different nodes communicate?

Pods can communicate across nodes using their **cluster-internal IPs** addresses. Kubernetes networking ensures that:

- every Pod gets a unique IP
- Pods can reach each other without NAT

21. How does the Kubernetes Scheduler assign Pods to Nodes?

19. What is the Kubernetes scheduler?

Answer:
The scheduler decides which Pod runs on which Node based on resource availability, affinity rules, and constraints.

The Kubernetes scheduler decides which node a pod should run on. It follows a three-step process:

1. Filtering (Predicates)
The scheduler first removes nodes that cannot run the pod — for example:

- not enough CPU or memory
- node taints without matching tolerations
- nodeSelector or affinity rules not matching

- storage or topology constraints

2. Scoring (Priorities)
From the remaining nodes, it scores each one — for example:

- nodes with more free resources score higher
- nodes with better topology fit may score higher

3. Binding
Finally, it picks the highest-scoring node and binds the pod to it.
The API Server records this decision, and the kubelet on that node starts the pod.

So in simple terms:
Filter → Score → Assign."

---

23. Describe how the Horizontal Pod Autoscaler (HPA) works.

The HPA automatically adjusts the number of pod replicas in a Deployment, ReplicaSet, or StatefulSet based on metrics.

Here's how it works:

1. HPA monitors metrics like CPU, memory, or custom application metrics.
2. If the metric crosses the threshold — for example, CPU > 70% — it scales **up** pod replicas.
3. When the load decreases, it scales **down** to save resources.
4. The controller continuously checks metrics from the metrics-server and updates replicas.

It basically ensures elasticity — more pods during high traffic, fewer pods during low traffic.

How does Kubernetes handle scaling of applications?

Answer:
Kubernetes handles scaling through Deployments and Horizontal Pod Autoscalers (HPA).
You can manually scale applications by adjusting the replica count in a Deployment.
The HPA automatically adjusts the number of pod replicas based on CPU utilization or other metrics.

---

24. Explain Custom Resources in Kubernetes.

“Custom Resources extend the Kubernetes API.
They let you add your own object types — just like Pods or Services — without modifying Kubernetes itself.

Once a Custom Resource is installed, users can create and manage those objects using `kubectl` as if they were native.

Kubernetes has become more modular because many major features (like Ingress, Operators, CSI drivers) are now built using Custom Resources.

They can also be added or removed dynamically without restarting the cluster, which makes Kubernetes very extensible.”

---

25. What are Affinity and Anti-Affinity?

“Affinity and Anti-affinity control *where* pods get scheduled.

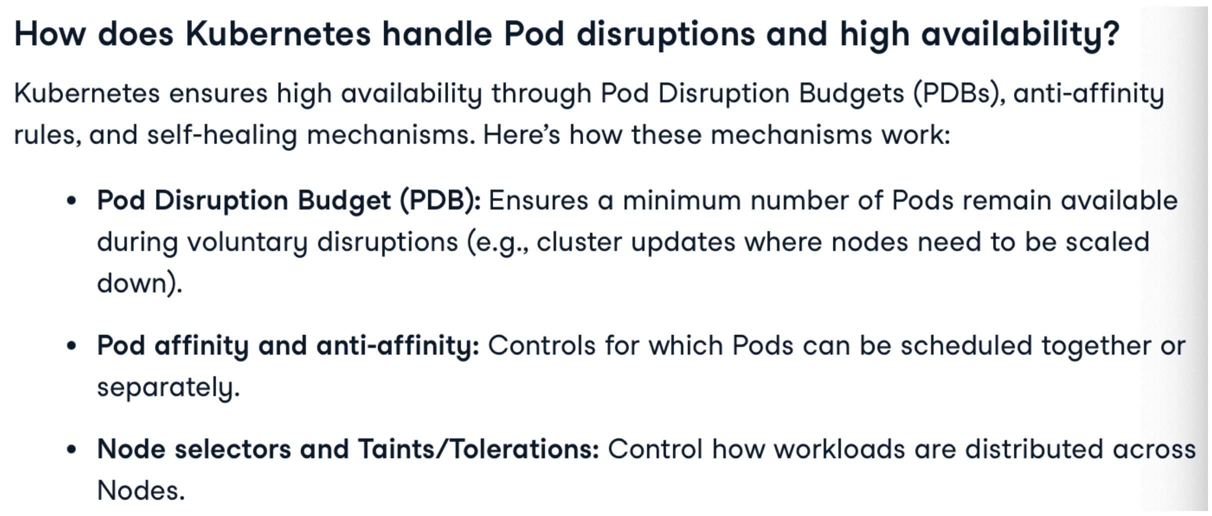
Node Affinity
Attracts pods to specific nodes based on node labels.
It’s like a more flexible version of `nodeSelector`.

Example:
Run all frontend pods on nodes labeled `env=prod`.

Pod Affinity
Makes pods run *near* other pods.
Example:
Place app and cache pods in the same availability zone for low latency.

Pod Anti-Affinity
Makes pods avoid each other.
Example:
Spread database pods across different zones so one failure doesn’t kill the entire database cluster.

Together, they give fine-grained control over pod placement.”



How does Kubernetes handle Pod disruptions and high availability?

Kubernetes ensures high availability through Pod Disruption Budgets (PDBs), anti-affinity rules, and self-healing mechanisms. Here's how these mechanisms work:

- **Pod Disruption Budget (PDB):** Ensures a minimum number of Pods remain available during voluntary disruptions (e.g., cluster updates where nodes need to be scaled down).
- **Pod affinity and anti-affinity:** Controls for which Pods can be scheduled together or separately.
- **Node selectors and Taints/Tolerations:** Control how workloads are distributed across Nodes.

---

26. What is a Network Policy in Kubernetes?

"A Network Policy is like a firewall rule for Pods.

It defines which Pods can talk to which other Pods, IPs, or namespaces — at Layer 3 and Layer 4.

By default, all Pods can communicate freely.
When a Network Policy is applied, everything becomes blocked except whatever is explicitly allowed.

So Network Policies enforce:

- ingress rules (who can send traffic to me)
- egress rules (where I can send traffic)

They are essential for implementing zero-trust networking inside the cluster."

---

27. Describe the role of kube-proxy.

"Kube-proxy is a critical network component that runs on every mode in a Kubernetes cluster. Its primary job is to enable connection between services and Pods, ensuring that traffic is routed correctly across the cluster.

Its responsibilities include:

1. Service routing
It watches for new Services and updates and configures routing rules.

2. Load balancing
It load-balances traffic across the backend Pods.

3. Protocol handling
Depending on the mode (iptables, IPVS), it sets up networking rules at the OS level.

Without kube-proxy, Services wouldn't know how to reach pods.”

---

28. What is a Helm chart and how is it used?

“A Helm chart is basically a package for Kubernetes applications.

It contains all the YAML files — Deployments, Services, ConfigMaps, Secrets, and so on — needed to deploy an application.

Using Helm, you can:

- deploy apps with one command
- manage versions
- update apps easily
- template configurations
- share or reuse deployments

It’s like apt or yum, but for Kubernetes.”

---

29. Explain Taints and Tolerations.

“Taints and Tolerations work together to control which pods can run on which nodes.

Taints (on nodes)
A taint says:
‘Don’t schedule pods here unless they tolerate this condition.’

For example:
A GPU node might have a taint saying only GPU workloads are allowed.

Tolerations (on pods)
A toleration on a pod lets it tolerate a node’s taint.
It doesn’t guarantee placement — it only allows it.

This mechanism is great for:

- isolating workloads
- dedicating nodes for special tasks

- preventing noisy-neighbor issues”

---

30. How does Kubernetes manage storage orchestration?

Kubernetes uses the Container Storage Interface (CSI) for storage orchestration.
CSI allows storage vendors to integrate their own drivers into Kubernetes.

The main components are:

PersistentVolume (PV)
Cluster-wide storage resource — like a disk.

PersistentVolumeClaim (PVC)
A request for storage from a pod.
PVC gets bound to a matching PV.

StorageClass
Defines how storage should be provisioned — for example:

- SSD or HDD
- encrypted or not
- which CSI driver to use

With CSI + StorageClass + PV + PVC, Kubernetes makes storage portable and dynamically provisioned.

---

31. Describe the use of init containers in Kubernetes.

Init containers are special containers that run **before** the main application containers in a Pod.
They are used for setup or initialization tasks that must complete successfully before the app starts.

Examples of what init containers are used for:

- running setup scripts
- waiting for a database to become ready
- downloading configuration files
- performing environment checks

They run **sequentially**, not in parallel. Each init container must finish before the next one starts, and the main containers don’t start until all init containers finish.

This allows you to keep your main application container clean and lightweight, while offloading one-time logic into separate init containers.

---

32. What are the various services available in Kubernetes?

A Service exposes one or more Pods over the network with a stable IP.

Kubernetes offers four main service types to expose Pods:

1. ClusterIP
This is the default service type. It exposes a group of pods internally within the cluster using a virtual IP.
Used for internal communication only.

2. NodePort
Exposes the service on a static port on every node.
External clients can reach the service using `<NodeIP>:<NodePort>`.

3. LoadBalancer
Used in cloud environments. Automatically provisions a cloud load balancer that forwards traffic to NodePort and then to pods.
Used when you need external public access.

4. ExternalName
Maps a service to an external DNS name. It acts like an alias.
Used when pods inside the cluster need to access an external service via a user-defined DNS name.

---

33. Explain the concept of a Custom Operator in Kubernetes.

A Kubernetes Operator automates complex application operations using native Kubernetes APIs.

It is built using two components:

- A **Custom Resource Definition (CRD)** that defined a new type of object (e.g. MyDatabase)
- A **Controller** that continuously watches for changes and acts to maintain the desired state

Operators encode domain-specific logic.
They automate tasks like provisioning, scaling, backups, failover, and upgrades.

They are especially useful for running stateful systems such as databases or distributed systems that require operational intelligence.

---

34. How do you troubleshoot a Pod in a CrashLoopBackOff state?

A CrashLoopBackOff means the container keeps crashing and Kubernetes keeps restarting it.

To troubleshoot:

1. **Describe the Pod**
   Check exit codes and the container's last state.
   Exit code 1 usually indicates an app error; OOMKilled indicates memory limits were exceeded.
2. **Check logs**
   Use `kubectl logs --previous` if the pod restarts quickly.
   Logs often reveal missing configs, failing dependencies, or runtime errors.
3. **Verify configuration**
   Inspect environment variables, ConfigMaps, Secrets, volumes, and command-line args.
4. **Check health probes**
   Incorrect liveness or readiness probes can cause Kubernetes to restart a healthy container, creating a crash loop.

This structured approach usually identifies the root cause quickly.

---

35. What is the purpose of the Kubernetes API server?

The API server is the **central communication point** of the Kubernetes control plane.

Its responsibilities are:

- handling all REST API calls from kubectl, controllers, and other components
- validating and storing resource definitions into etcd
- exposing the Kubernetes API to users and internal components
- acting as the front end for the cluster's shared state

Every interaction with the cluster goes through the API server, making it a core component of the control plane.

---

36. How do you handle an ImagePullBackOff error?

An ImagePullBackOff means Kubernetes could not pull the container image.

Troubleshooting steps:

1. **Describe the Pod**
   Look at the Events section for detailed error messages (such as “image not found”).
2. **Verify the image name and tag**
   Check for typos or incorrect tags in your manifest.
3. **Check registry connectivity**
   Ensure the node can reach the registry (DNS or network issues may block pulls).
4. **Check imagePullSecrets**
   If the image is in a private registry, confirm your pull secrets are correctly set and referenced in the Pod spec.

Fixing the image name or authentication usually resolves the issue.

---

37. How would you perform a Kubernetes cluster upgrade?

Upgrading a cluster involves upgrading both control plane and worker nodes.

The steps are:

1. Review the release notes for breaking changes.
2. Back up etcd to preserve the cluster state.
3. Upgrade control plane components one master node at a time.
4. Upgrade worker nodes:
   - `kubectl drain` to move workloads
   - update packages
   - restart services
   - `kubectl uncordon` to return nodes to service
5. Upgrade add-ons like CNI plugins, CoreDNS, and Ingress controllers.
6. Always test the upgrade in staging first.

This ensures a smooth and safe upgrade.

---

38. How would you back up and restore a cluster?

A complete backup includes both cluster state and application data.

1. **Cluster State (etcd backup)**
   etcd holds the entire Kubernetes configuration.
   Regular snapshots should be taken.
   Restore by stopping the API server, restoring the snapshot, and restarting control plane components.
2. **Application Data (Persistent Volumes)**
   Tools like Velero are commonly used to take snapshots of both PVCs and Kubernetes resources.
   Velero allows full restoration of workloads, including PV data and object manifests.

This ensures both the control plane and stateful applications can be fully recovered.

---

39. How does Kubernetes handle node failures and resiliency?

Kubernetes ensures resiliency through several mechanisms:

- **Node health monitoring:** The control plane marks a node as NotReady if heartbeats stop.
- **Pod restart policies:** Failed containers are restarted automatically by kubelet.
- **Replication and desired state:** Deployments ensure the correct number of replicas are always running.
- **Pod Disruption Budgets:** Prevent too many pods of the same application from going down during maintenance.
- **Node pools and multi-zone deployments:** Distribute workloads across zones or node groups for high availability.

If a node fails, Kubernetes reschedules pods on healthy nodes to maintain availability.

40. Explain how to set up and use Role-Based Access Control (RBAC) in Kubernetes.

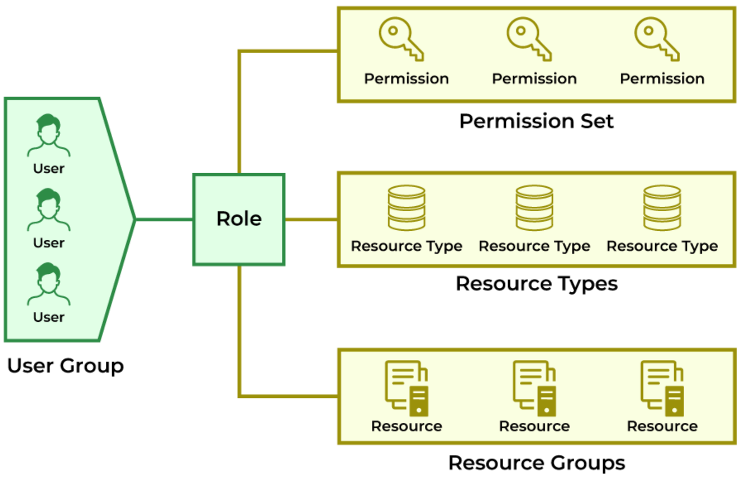
RBAC controls **who** can do **what** in the cluster.

The process:

1. **Define users or groups**
   These can come from certificate-based authentication, cloud IAM, or OIDC.
2. **Create Roles or ClusterRoles**

    - A **Role** defines permissions within a namespace.
    - A **ClusterRole** defines cluster-wide permissions.
3. **RoleBinding or ClusterRoleBinding**
   Bind the role to specific users or groups.
4. **Apply policies**
   The API server enforces RBAC whenever someone makes a request.

RBAC is essential for multi-team clusters, least-privilege access, and secure operations.



Senior Software Engineer

What are Kubernetes Services, and what types are there?

Answer:
Kubernetes Services provide stable networking and load balancing for Pods.
Types of Services include:

- **ClusterIP:** Exposes the Service on a cluster-internal IP. This is the default type.
- **NodePort:** Exposes the Service on each node's IP at a static port, making the Service externally accessible.
- **LoadBalancer:** Exposes the Service externally using a cloud provider's load balancer.
- **Headless Service:** No load balancing or proxying; useful for StatefulSets.

—--

What are Jobs and CronJobs?

Answer:

- **Job** → runs a task once and exits.
- **CronJob** → runs on a schedule (e.g., nightly backups).

18. What are Requests and Limits in Kubernetes?

Answer:

- **Requests:** minimum CPU/memory the Pod needs.
- **Limits:** maximum CPU/memory the Pod is allowed.
  Used by scheduler to plan resources.

20. What is kubectl?

Answer:
kubectl is the command-line tool used to interact with the Kubernetes cluster.

21. Common kubectl commands?

Answer:

```
kubectl get pods
kubectl describe pod <name>
kubectl logs <pod>
kubectl exec -it <pod> -- bash
kubectl apply -f file.yaml
kubectl delete pod <name>
```

22. How do you deploy an application in Kubernetes?

Answer:

1. Write a **Deployment YAML**
2. (Optional) Write a **Service YAML**
3. Apply using:

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

23. What is a Kubernetes YAML file?

Answer:
A YAML file describes Kubernetes resources such as Pods, Deployments, and Services in a declarative format.