1(a). How orchestration tools help manage and scale application servers

Tools like Kubernetes automate the deployment, management, and scaling of application servers across clusters of machines.

They:

- Monitor container health: automatically restart failed containers or move them to healthy nodes. - Balance workloads: distribute applications evenly across nodes for efficient resource use. - Scale dynamically: add or remove replicas based on CPU/memory usage or custom metrics. - Automate rollouts and rollbacks: update applications seamlessly without downtime.

In short: orchestration tools provide self-healing, auto-scaling, and load balancing for containerized applications.

1(b). Automated deployment, scaling, and management

Kubernetes uses Deployments or Helm charts to define and automate container rollout across multiple nodes.

Horizontal Pod Autoscaler (HPA) automatically adjusts the number of replicas based on resource usage.

Kubernetes continuously monitors desired vs. actual state and reconciles differences automatically (the “desired state” model).

Result: consistent, reliable, and automated app lifecycle management.

2. Difference between Pod, Deployment, and Service

3. What is a Namespace in Kubernetes?

A Namespace is a logical partition within a Kubernetes cluster that allows resources to be grouped and isolated.

It helps separate environments (e.g., dev, test, prod), avoid naming conflicts, and apply resource quotas or access controls.

Example:

```
kubectl create namespace development
```

4. Role of the Kubelet and how to check nodes

The Kubelet runs on each worker node and ensures that containers described in PodSpecs are running and healthy.

It reports node status and metrics back to the control plane.

Command to check nodes:

```
kubectl get nodes
```

5. Difference between ClusterIP, NodePort, and LoadBalancer services

6. Scale a Deployment to 5 replicas

Command:

```
kubectl scale deployment --replicas=5
```

Example:

```
kubectl scale deployment my-app --replicas=5
```

7. Update the image of a Deployment without downtime

Command:

```
kubectl set image deployment/ =
```

Example:

```
kubectl set image deployment/my-app my-app-container=nginx:1.25
```

Kubernetes performs a rolling update, replacing old Pods gradually to ensure zero downtime.

8. Expose a Deployment to external traffic

Command:

```
kubectl expose deployment --type=LoadBalancer --port=80 --target-port=8080
```

This creates a Service of type LoadBalancer that routes external requests to the Pods.

9. How Kubernetes scheduling decides which node a Pod runs on

The Kubernetes scheduler assigns Pods to nodes based on:

- Resource availability (CPU, memory requests and limits) - Affinity/anti-affinity rules - Taints and tolerations - Custom scheduling policies (if defined)

It picks the best-fitting node that meets all requirements and has available capacity.

10. Role of Ingress and how it differs from a Service

Ingress manages HTTP/HTTPS access to Services from outside the cluster. It supports features like path-based routing, TLS termination, and virtual hosting, and requires an Ingress Controller (like NGINX Ingress).

Difference:

Ingress = intelligent HTTP gateway; Service = network access point for Pods.