1. Orchestration tools, such as Kubernetes, play a key role in the server infrastructure for the modern applications.

(a) Explain how these tools help manage and scale application servers.

Orchestration tools such as Kubernetes manage and scale application servers by treating a cluster of machines as a single unified platform. Instead of manually deploying and maintaining servers, administrators define the desired state of the application—such as how many instances should run or what resources they require—and the orchestrator ensures that this state is maintained. It automatically schedules containers onto available servers based on CPU and memory availability, distributes traffic to balance the load, and continuously monitors the health of applications. When demand increases, the orchestrator can add more instances (scaling out), and when demand decreases, it can reduce them to save resources, ensuring efficient use of infrastructure while maintaining high availability and performance.

(b) Describe how orchestration tools facilitate automated deployment, scaling, and management of application servers.

Orchestration tools facilitate automated deployment, scaling, and management by using declarative configuration files (like YAML in Kubernetes) that describe how applications should run. Once defined, the tool automatically pulls container images, deploys them, configures networking, and keeps the system in the desired state. It supports automated scaling by monitoring metrics such as CPU usage and adding or removing instances as needed. These tools also handle rolling updates to release new versions without downtime and can roll back automatically if errors occur. Additionally, they provide self-healing by restarting failed containers and rescheduling workloads on healthy nodes, while integrating monitoring and logging to continuously manage and optimize application servers with minimal manual intervention.

2. Explain the difference between a Pod, Deployment, and Service.

A **Pod** is the smallest deployable unit in Kubernetes and represents one or more tightly coupled containers that share the same network namespace and storage. A

Deployment is a higher-level object that manages Pods by defining how many replicas should run, handling rolling updates, and ensuring the desired state is maintained. A **Service** provides a stable network endpoint to access Pods, because Pods are temporary and their IPs change; Services ensure consistent communication by load balancing traffic across the matching Pods.

3. What is a Namespace in Kubernetes? Please list one example

A **Namespace** in Kubernetes is a logical partition within a cluster that isolates resources to organize workloads and prevent naming conflicts. It is often used to separate environments (such as development, testing, and production) or to allocate resources to different teams or projects. For example, a cluster may have a namespace called **"dev"** where developers deploy and test applications separately from the **"prod"** namespace used for live production workloads.

4. Explain the role of the Kubelet. How do you check the nodes in a Kubernetes cluster? (kubectl command expected)

The **Kubelet** is an agent that runs on each worker node and ensures that the containers in Pods are running as expected. It communicates with the Kubernetes control plane, receives Pod specifications, starts containers using the container runtime (e.g., Docker or containerd), and continuously monitors their health. If a container fails, the Kubelet attempts to restart it to maintain the desired state. To check the nodes in a Kubernetes cluster, you can use the command:
kubectl get nodes.

5.What is the difference between ClusterIP, NodePort, and LoadBalancer services?

ClusterIP is the default Service type and exposes the service only inside the cluster using an internal virtual IP, making it suitable for internal communication between Pods. **NodePort** exposes the service on a static port on every node in the cluster, allowing external access by sending traffic to any node's IP and the specified port, but it offers limited flexibility. **LoadBalancer** creates an external load balancer (usually from a cloud provider) that distributes traffic to the NodePort behind the

scenes, providing a single external IP and handling traffic distribution more efficiently for production-level external access.

6. How do you scale a Deployment to 5 replicas using kubectl?

To scale a Deployment to 5 replicas using kubectl, you instruct Kubernetes to change the desired number of running Pod instances in the Deployment. Kubernetes will then automatically create or terminate Pods to match the new count. The command is:

kubectl scale deployment <deployment-name> --replicas=5.

7. How would you update the image of a Deployment without downtime?

To update the image of a Deployment without downtime, Kubernetes performs a rolling update, gradually replacing old Pods with new ones while keeping the service available. This can be done with the command:

kubectl set image deployment/<deployment-name> <container-name>=<new-image>.

Kubernetes ensures that at least some Pods remain running during the transition, preventing service interruption.

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

To expose a Deployment to external traffic, you create a Service of type **NodePort** or **LoadBalancer**, depending on the environment. This Service maps an external port to the Pods managed by the Deployment, allowing users outside the cluster to access the application. For example:

kubectl expose deployment <deployment-name> --type=LoadBalancer --port=80 --target-port=8080.

9. How does Kubernetes scheduling decide which node a Pod runs on?

Kubernetes scheduling decides which node a Pod runs on by evaluating resource availability and constraints. The scheduler checks each node's free CPU and memory, and then applies rules such as node selectors, taints and tolerations, and affinity/anti-affinity. It picks the node that satisfies all requirements and optimizes resource usage. If no node can meet the Pod's needs, the Pod remains pending until resources are available.

10. What is the role of Ingress and how does it differ from a Service?

Ingress manages external HTTP/HTTPS traffic and routes it to Services based on hostnames or URL paths, acting like an application-layer load balancer. Unlike a Service, which only exposes a single application or port, an Ingress can combine multiple Services behind one external IP and apply routing rules, SSL termination, and virtual hosting. In short, a Service exposes Pods, while an Ingress controls and routes external traffic to those Services more intelligently.