**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.**

Automated Management:They automate the processes of deploying, starting, stopping, restarting, and monitoring application servers (containers). There's no need to manually log into each server to perform operations.

Declarative Configuration:You declare the desired state of your application (e.g., "need to run 3 instances") through configuration files (YAML/JSON). The orchestration tool continuously works to ensure the actual state matches the desired state.

Self-healing:When an application server instance (container) fails, the orchestration tool automatically detects the failure and restarts a new instance on a healthy node, ensuring high application availability.

Elastic Scaling:They can automatically increase or decrease the number of application instances based on predefined rules (such as CPU usage). This allows you to easily handle traffic spikes and optimize resource usage costs.

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

Orchestration tools facilitate automation by providing a control plane. We define your application and its requirements (like container images, resources, network, storage) in a set of configuration files. The tool, upon receiving these definitions, automates the entire lifecycle:

Automated Deployment: It selects appropriate hosts, pulls container images, and starts the containers. It also supports complex deployment strategies like rolling updates (for zero-downtime deployments) and blue-green deployments.

Automated Scaling:Through mechanisms like the Horizontal Pod Autoscaler, it can automatically adjust the number of application instances based on real-time metrics, achieving true elasticity.

Automated Management:It continuously performs health checks, manages service discovery and load balancing to ensure requests are routed to healthy instances, and automatically reschedules failed instances.

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

Pod:The smallest and simplest unit in Kubernetes that you can create or manage. It is a group of one or more containers that share storage, network, and a specification for how to run. The Pod is the instance that actually runs your application code. However, Pods themselves are ephemeral (mortal).

Deployment: A higher-level concept that manages the deployment and lifecycle of Pods. You define a Deployment (e.g., need to run 3 replicas of a Pod using the nginx:1.20 image), and the Deployment controller ensures that exactly 3 Pods are always running. If a Pod fails, it replaces it. It also provides rolling updates and rollbacks for Pods. Deployment is the recommended way to manage a set of Pod replicas.

Service: An abstraction layer that exposes a set of functionally equivalent Pods as a network service. Because Pods are created and destroyed dynamically, their IP addresses change. A Service provides a stable IP address, DNS name, and port. Other applications can access the group of Pods through this stable endpoint without needing to know which specific Pod is handling the request. A Service also provides load balancing.

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

What it is:A Namespace provides a virtual isolation mechanism within a Kubernetes cluster. It allows you to divide cluster resources into multiple independent, logical groups. This helps in achieving resource isolation and organization between different teams, projects, or environments (like development, testing, production), avoiding naming conflicts.

Example:A common example is the kube-systemnamespace. This namespace is used for running Kubernetes system components, such as Kubernetes DNS, the dashboard, networking plugins, etc. User applications typically should not be deployed in this namespace, thus isolating system components from user applications.

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

Role of the Kubelet:The Kubelet is an agent that runs on each node in the cluster. Its primary responsibility is to ensure that containers are running in a Pod on its node. Specifically, it receives PodSpecs (Pod definitions) from the API server and ensures that the containers described in those Specs are started, stopped, and managed. It is also responsible for executing liveness and readiness probes for the containers and reporting the node and Pod status back to the control plane.

Command to check nodes:Use the *kubectl get node*s command.

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

These three are different types of Kubernetes Services, and the main difference lies in the scope of exposure:

ClusterIP:This is the default Service type. It assigns the Service a virtual IP address that is only accessible within the cluster. This service cannot be accessed from outside the cluster. It is suitable for backend services like databases that are only accessed by other services within the cluster.

NodePort:On top of the ClusterIP, it exposes the Service on the same static port (called the NodePort, range 30000-32767) on every node in the cluster. This means you can access the service from outside the cluster by accessing any node's IP address combined with the NodePort. It is often used for custom solutions or temporary services that need to be directly accessible externally.

LoadBalancer: This is the most common way to expose a service externally, typically used on cloud providers (like AWS, GCP, Azure). Building on NodePort, it automatically provisions and configures an external load balancer from the cloud platform. This load balancer directs external traffic to the NodePorts on your cluster nodes, ultimately reaching your service. It provides a stable, professional external entry point for production environments.

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

Use the following command:

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

Replace*<deployment-name>*with your actual Deployment name.

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

Use the*kubectl set image* command. This triggers the "rolling update" strategy of the Deployment, which is the default behavior. The rolling update gradually replaces old Pod versions with new ones, ensuring that a sufficient number of old Pods are still handling requests during the process, achieving zero-downtime updates.

1. **How do you expose a Deployment to external traffic?**

The most common method is to create a Service of type LoadBalanceror NodePortthat targets the Deployment.

Using the*kubectl expose* command: *kubectl expose deployment <deployment-name> --type=LoadBalancer --port=80 --target-port=8080*

Alternatively, a more modern and feature-rich way is to use an Ingress resource along with an Ingress Controller. This is often preferred as it provides advanced features like domain and path-based routing.

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

Scheduling is handled by a component called the kube-scheduler. The process mainly involves two steps:

1.Filtering:First, the scheduler filters out all nodes that do not meet the Pod's requirements. These requirements include:

Whether the node has enough available resources (CPU and memory) to meet the Pod's requests.

Whether the node has specific labels required by the Pod (using nodeSelectoror nodeAffinity).

Whether the node's taints are tolerated by the Pod's tolerations.

Other constraints like storage, ports, etc.

The nodes that pass this filter are called "feasible" nodes.

2.Scoring:Next, the scheduler scores each feasible node. Scoring is based on various strategies, such as:

Selecting the node whose resource profile best fits the Pod's requests (for resource balancing).

Selecting the node with the fewest existing Pods.

Selecting a node based on affinity/anti-affinity rules.

The node with the highest score is selected as the target node to run the Pod.

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

Role of Ingress:An Ingress is an API object whose main role is to manage external access to the services within a cluster, typically HTTP and HTTPS traffic. It acts as an entry point for the cluster, providing features like domain name (host) and path-based routing, SSL/TLS termination, and load balancing. Important: The Ingress itself does not handle traffic; it must be used with an actual Ingress Controller (like Nginx Ingress Controller, Traefik) which is responsible for implementing the Ingress rules.​​

Difference from Service:​​

Layer and Functionality:A Service primarily operates at the TCP/IP layer (Layer 4), providing internal load balancing and a stable network endpoint. An Ingress operates at the application layer (Layer 7, HTTP/HTTPS), providing smarter routing rules (e.g., routing api.example.comto the API Service and www.example.comto the front-end Service).

Exposure Method:A LoadBalancerService typically creates a new cloud load balancer for each service that needs to be exposed, which can be costly. An Ingress can act as a single entry point,managing access to many different services inside the cluster, requiring only one load balancer, which is more efficient and economical.