1.

(a) Orchestration tools like Kubernetes simplify the management and scaling of application servers by abstracting away the underlying infrastructure. They treat a cluster of machines as a single, unified resource pool. Key ways they help include:

- **Resource Management:** They efficiently allocate resources (CPU, memory) to applications, ensuring optimal utilization and preventing resource starvation.
- **High Availability:** They monitor the health of applications and servers, automatically restarting failed containers or rescheduling them to healthy nodes, thus ensuring continuous availability.
- **Load Balancing:** They distribute incoming traffic across multiple instances of an application, preventing any single instance from becoming a bottleneck and improving responsiveness.
- **Declarative Configuration:** You define the desired state of your applications (e.g., "run 3 instances of this application"), and the orchestrator continuously works to achieve and maintain that state.
- **Service Discovery:** Applications can easily find and communicate with each other within the cluster without needing to know their specific network addresses.

(b) Orchestration tools automate these processes through several mechanisms:

- **Automated Deployment:** You define your application as a set of container images and configuration files. The orchestrator pulls these images, creates containers, and deploys them onto available nodes according to your specifications. It handles the rolling out of new versions and rolling back to previous versions if issues arise.
- **Automated Scaling:**
  - **Horizontal Scaling:** Based on metrics like CPU utilization or custom application metrics, the orchestrator can automatically add or remove instances (replicas) of your

application to handle fluctuating loads.
    - **Vertical Scaling:** While less common for automatic scaling, orchestrators allow you to easily adjust the resource requests and limits for your containers.

- **Automated Management:**
    - **Self-healing:** If a container or even an entire node fails, the orchestrator automatically detects the failure and reschedules the affected workloads to healthy resources.
    - **Configuration Management:** Secrets, configuration files, and environment variables can be managed centrally and injected into applications automatically.
    - **Storage Orchestration:** It can dynamically provision and attach storage to containers as needed.

2.

- **Pod:** The smallest deployable unit in Kubernetes. A Pod represents a single instance of a running process in your cluster. It encapsulates one or more containers (which are tightly coupled and share resources like networking and storage), storage resources, a unique network IP, and options that govern how the container(s) should run. Pods are ephemeral; if a Pod dies, it's not automatically replaced by Kubernetes.
- **Deployment:** A higher-level abstraction that manages the deployment and scaling of a set of identical Pods. Deployments provide declarative updates to Pods and ReplicaSets. They ensure that a specified number of Pod replicas are running at all times. When you update a Deployment, it automatically rolls out the changes to your Pods in a controlled manner (e.g., rolling updates).
- **Service:** An abstract way to expose an application running on a set of Pods as a network service. Services define a logical set of Pods and a policy by which to access them. They provide a stable IP address and DNS name for your application, even if the underlying Pods change (e.g., due to scaling or rescheduling). This decouples clients from the specific Pod instances.

3. A Namespace in Kubernetes provides a mechanism for isolating groups of resources within a single cluster. It's like a virtual cluster inside your Kubernetes cluster. Namespaces are intended for use in environments with many users or teams spread across multiple projects, helping to organize resources and prevent naming collisions. Resources within a namespace are invisible to resources in other namespaces by default.

Example:

kube-system is a common Namespace used for Kubernetes system objects like the API server, scheduler, and controller manager. You'd typically deploy your own applications into namespaces like default or create custom ones like production or development.

4. The Kubelet is an agent that runs on each node in a Kubernetes cluster. Its primary role is to ensure that containers described in PodSpecs are running and healthy on its respective node. It receives PodSpecs from the Kubernetes API server and manages the containers running on that node, including:

- Starting and stopping containers.
- Monitoring the health of containers and reporting their status to the API server.
- Mounting volumes.
- Reporting node status and resources.

To check the nodes in a Kubernetes cluster, you use the kubectl get nodes command.

5.

- **ClusterIP:**
  - **Purpose:** Exposes the Service on an internal IP address within the cluster.
  - **Access:** Only accessible from *inside* the Kubernetes cluster.
  - **Use Case:** Ideal for internal services that other applications within the cluster need to access, but not directly from

outside. It provides a stable internal IP for a set of Pods.

- **NodePort:**
    - **Purpose:** Exposes the Service on a static port on each Node's IP address.
    - **Access:** Accessible from *outside* the cluster by requesting <NodeIP>:<NodePort>. Kubernetes allocates a port from a pre-configured range (default: 30000-32767) on all worker nodes.
    - **Use Case:** A simple way to allow external traffic to your cluster for development or testing. However, it's generally not recommended for production due to the arbitrary port range and the need to manage external access to individual node IPs.
- **LoadBalancer:**
    - **Purpose:** Exposes the Service externally using a cloud provider's load balancer.
    - **Access:** Accessible from *outside* the cluster via a dedicated, externally accessible IP address provisioned by the cloud provider.
    - **Use Case:** The standard way to expose public-facing applications in a cloud environment. The cloud provider automatically creates an external load balancer that directs traffic to your Kubernetes Nodes and then to your Pods. This abstracts away the underlying Node IPs and provides a stable external endpoint.

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

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

8. kubectl expose deployment <deployment-name> --type=LoadBalancer

--port=80 --target-port=8080

9. The scheduler decides based on:

- **Resource requirements** (CPU, memory)
- **Node affinity/anti-affinity** rules
- **Taints and tolerations**
- **Pod priority and preemption**
- **Node selector and node name**
- **Resource availability and constraints**

10.

Service Role: Provides internal load balancing and service discovery within the cluster.

Ingress Role: Manages external access to services, typically HTTP/HTTPS, providing:

- SSL/TLS termination
- Name-based virtual hosting
- Path-based routing
- Load balancing

Key Difference: Services provide L4 (TCP/UDP) load balancing, while Ingress provides L7 (HTTP/HTTPS) routing capabilities and acts as an API gateway for external traffic.