1. Orchestration Tools (e.g., Kubernetes)

(a) These tools help manage servers by grouping them into a cluster, so you don’t have to handle each server alone. For scaling, they add more copies of apps (replicas) when traffic is high and remove them when it’s low—this keeps apps fast without wasting resources.

(b) They automate deployment by using config files to set up apps the same way every time, no manual steps needed. For scaling, they monitor app load and adjust replicas automatically. For management, they check if apps are running; if one crashes, they restart it right away.

2. Difference Between Pod, Deployment, and Service

Pod: The smallest unit in Kubernetes, like a tiny "box" that holds one or more app containers. It’s temporary—if it dies, it doesn’t come back on its own.

Deployment: Manages Pods to keep them running. It makes sure the right number of Pod replicas exist, and it can update Pods without downtime.

Service: Acts like a "stable address" for Pods. Since Pods can die and be replaced (with new IPs), a Service lets apps connect to Pods using a fixed IP, so connections don’t break.

3. Namespace in Kubernetes

A Namespace is like a "folder" in Kubernetes that divides cluster resources (like Pods or Services) into separate groups. It helps teams or projects share a cluster without mixing up their stuff.

Example: The `default` Namespace—if you don’t specify a Namespace when creating resources, they go here automatically.

4. Role of Kubelet & Checking Nodes

Kubelet’s role: It’s an agent that runs on every node in the cluster. Its job is to make sure containers in Pods are running properly on the node, following the instructions from the Kubernetes control plane.

Command to check nodes: `kubectl get nodes` — this shows all nodes in the cluster, their status (e.g., "Ready"), and other details.

5. Difference Between ClusterIP, NodePort, and LoadBalancer

ClusterIP: Only lets apps \*inside the cluster\* access the Service. It uses an internal IP, so external traffic (from outside the cluster) can’t reach it.

NodePort: Opens a fixed port on every node in the cluster. External traffic can access the Service using `NodeIP:NodePort` (e.g., `192.168.1.100:30080`).

LoadBalancer: Works with cloud providers (like AWS or GCP) to create a public load balancer. It routes external traffic to the Service automatically, so you don’t need to use node IPs.

6. Scaling a Deployment to 5 Replicas

Use the command: `kubectl scale deployment <deployment-name> --replicas=5`

(Replace `<deployment-name>` with your Deployment’s actual name, like `my-app-deployment`.)

7. Updating a Deployment’s Image Without Downtime

Kubernetes uses "rolling updates" by default, which replace old Pods with new ones one by one (so some Pods always run).

The command is: `kubectl set image deployment/<deployment-name> <container-name>=<new-image:tag>`

Example: `kubectl set image deployment/my-app-deployment my-app=my-app:v2`

8. Exposing a Deployment to External Traffic

You need to create a Service of type `NodePort` or `LoadBalancer` (since `ClusterIP` is internal only).

Use the `expose` command:

For NodePort: `kubectl expose deployment <deployment-name> --type=NodePort --port=<app-port>`

For LoadBalancer (cloud-only): `kubectl expose deployment <deployment-name> --type=LoadBalancer --port=<app-port>`

9. Kubernetes Scheduling Logic

The Kubernetes Scheduler (a control plane component) decides which node a Pod runs on. It checks two main things:

1. Resource requirements: Does the node have enough CPU, memory, or storage to run the Pod?

2. Node selectors/affinity rules: If the Pod has rules (e.g., "run only on nodes with a ‘production’ label"), the scheduler picks nodes that match.

It then chooses the best node (with the least used resources, usually) to place the Pod.

10. Role of Ingress & Difference from Service

Ingress’s role: It’s like a "traffic controller" for external traffic. It routes HTTP/HTTPS traffic from outside the cluster to different Services inside, based on rules (e.g., send traffic to `api.myapp.com` to the `api-service`, and `web.myapp.com` to the `web-service`).

Difference from Service: A Service routes traffic to one set of Pods (one Service = one app endpoint). Ingress works \*above\* Services—it can route traffic to multiple Services using a single external IP, and it supports things like SSL/TLS and path-based routing (which Services don’t do).