# In-Class Assessment 2

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

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

Orchestration tools manage and scale application servers by abstracting the underlying infrastructure and providing a declarative, automated control plane. Their role in **management** involves:

**1) Declarative Model and Desired State**

Users define the "desired state" of their application (e.g., which container image to run, the number of replicas, network ports) in a configuration file. The orchestration tool's control loop continuously works to ensure the actual state of the cluster matches this declared desired state. If an application instance fails, the tool automatically detects the discrepancy and recreates it, ensuring high availability without manual intervention.

**2) Efficient Resource Utilization**

These tools include a scheduler that intelligently places application instances (Pods) onto available worker nodes based on their resource requirements (CPU, memory), thus optimizing the use of cluster resources and preventing any single node from being overloaded.

For **scaling**, they provide:

**1)Horizontal Scaling**

Orchestration tools can automatically increase or decrease the number of running application instances based on observed metrics like CPU utilization or custom application metrics. This allows the application to handle increased traffic seamlessly and scale down to reduce costs during periods of low demand.

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

**1) Automated Deployment**

They enable sophisticated deployment strategies like rolling updates. When a new version of an application is deployed, the tool incrementally replaces old instances with new ones, ensuring zero-downtime and allowing for easy rollback if a failure is detected.

**2) Automated Scaling**

As mentioned, they support Horizontal Pod Autoscaling (HPA), which automatically adjusts the number of application replicas in response to real-time load, ensuring performance and cost-efficiency.

**3) Automated Management**

This is the continuous reconciliation process. The tool constantly monitors the health of application servers, restarts failed containers, reschedules them if a node fails, and manages service discovery and load balancing to ensure network traffic is correctly routed to healthy instances.

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

These are three core concepts in Kubernetes that work together at different layers of abstraction.

**1) Pod**

The smallest and simplest deployable unit in Kubernetes. A Pod represents a single instance of a running process and can contain one or more tightly coupled containers that share the same network namespace, IP address, and storage volumes. However, a Pod by itself is ephemeral; if it dies, it is not recreated unless managed by a higher-level controller.

**2) Deployment**

A higher-level abstraction that **manages the lifecycle of Pods**. A Deployment is a declarative way to manage a set of identical Pods.We define a desired state (e.g., "run three replicas of this Pod template"), and the Deployment controller works to maintain that state. It enables crucial operations like scaling the number of Pods up or down and performing rolling updates or rollbacks. It provides the mechanism for reliable, scalable application deployment.

**3) Service**

An abstraction that defines a **logical set of Pods and a policy to access them**. Because Pods are ephemeral and can get new IP addresses when they are recreated, we cannot rely on their IPs for stable network access. A Service provides a stable IP address and DNS name that acts as a permanent endpoint. It also load-balances traffic across all the healthy Pods that match its selector, ensuring reliable network connectivity for our application.

**In conclusion, The Pod** is the actual running instance. **The Deployment** creates and manages the lifecycle of multiple Pods. **The Service** provides a stable network pathway to access those Pods.

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

A Namespace in Kubernetes is a virtual cluster within a physical Kubernetes cluster. It provides a mechanism for isolating groups of resources within a single cluster. This is useful for dividing a cluster between multiple users, teams, or projects (e.g., development, staging, production) without needing multiple physical clusters.

* **Example: kube-system**

This is a default namespace where Kubernetes system components (like the DNS service, metrics server, and networking plugins) are deployed. Users typically should not deploy their application Pods into this namespace to avoid interfering with critical system functions.

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

The **Kubelet** is an agent that runs on each worker node in a Kubernetes cluster. Its primary role is to ensure that containers are running in a Pod. It works by taking a set of PodSpecs (Pod definitions) provided to it (primarily from the API server) and ensuring that the containers described in those PodSpecs are running and healthy. It is the central "node manager" that directly interacts with the container runtime (like Docker or containerd) to start, stop, and maintain containers.

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

**kubectl get nodes**

This command will list all the worker and control plane nodes in the cluster, along with their status, roles, age, and Kubernetes version. To get more detailed information, we can use the **-o wide** flag.

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

These are different Service types in Kubernetes that determine how network traffic is routed to our Pods, primarily distinguished by their scope and exposure level.

**a. ClusterIP**

This is the **default** Service type. It assigns the Service a stable, internal IP address that is only accessible from within the Kubernetes cluster. It is used for communication between different components of our application (e.g., a backend API talking to a frontendWeb server) where no external access is required.

**b. NodePort**

This type builds on top of ClusterIP. It exposes the Service on a **static port** on each Node's IP address. In addition to the internal ClusterIP, a port in the range 30000-32767 is opened on every node. Traffic sent to **<any-node-ip>:<nodeport>** is routed to the underlying Service and then to a Pod. It provides a way for external access but is generally not used for production directly due to manual port management and security.

**c. LoadBalancer**

This type builds on top of NodePort. It provisions an **external cloud load balancer** (e.g., in AWS, GCP, or Azure) automatically. The cloud provider assigns the Service a stable external IP address. Traffic to this external IP is load-balanced by the cloud provider's infrastructure to all the nodes in our cluster on the NodePort, which is then routed to the Pods. This is the standard way to expose a Service directly to the internet in a cloud environment.

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

We use the kubectl scale command.

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

For example, to scale a Deployment named my-web-app to 5 replicas:

**kubectl scale deployment/my-web-app --replicas=5**

We can also do this imperatively by editing the Deployment:

**kubectl edit deployment/<deployment-name>**

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

The recommended method is to perform a **rolling update**. This is the default behavior when we update the container image of a Deployment.

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

For example, to update the nginx container in a Deployment named my-app to version 1.21:

**kubectl set image deployment/my-app nginx=nginx:1.21**

* How it works (no downtime):

The Deployment controller will gradually start new Pods with the new image and terminate the old ones. It ensures that a certain number of Pods are always available (by default, at least 75% of the desired replicas) and that the total number does not exceed a specified limit (by default, 125%). This process ensures the application remains available to users throughout the update. If something goes wrong,We can roll back with **kubectl rollout undo deployment/<deployment-name>**.

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

We expose a Deployment to external traffic by creating a Service of type **NodePort** or **LoadBalancer**.

The most common and direct way in a cloud environment is to create a LoadBalancer Service.

Command (using kubectl expose):

**kubectl expose deployment <deployment-name> --type=LoadBalancer --name=<service-name> --port=80**

This command creates a new Service that targets the Pods of the specified Deployment. The cloud provider will provision an external load balancer, and the **kubectl get services** command will eventually show the external IP address.

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

The decision of where to place a Pod is made by the kube-scheduler. It is a two-step process:

**a. Filtering (Predicates)**

The scheduler evaluates all nodes against a set of rules to find feasible nodes. Examples of these rules include:

* **NodeSelector / nodeName:** Does the node have a specific label?
* **Resource Requests:** Does the node have enough free CPU and memory to fit the Pod?
* **Taints and Tolerations:** Is the Pod "tolerating" a node's "taint"?
* **Affinity/Anti-Affinity:** Does the Pod have rules demanding it be placed with or away from other Pods? (e.g., "spread me across availability zones").

**b. Scoring (Priorities)**

The scheduler then ranks the feasible nodes from the filtering step to find the most optimal one. It uses scoring functions like:

* Spreading Pods across nodes and zones for high availability.
* Preferring nodes that already have the required container image cached.
* Preferring nodes with the most free resources (balanced resource allocation).

The node with the highest score is selected, and the kube-scheduler binds the Pod to that node. The kubelet on that node is then informed and runs the Pod.

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

**a. Service Role**

A Service provides internal load balancing and a stable endpoint for a set of Pods. Types like LoadBalancer expose a single Service directly to the outside world, which can be inefficient (one external IP per service) and lack advanced routing features.

**b. Ingress Role**

An **Ingress** is **not** a Service type. It is a separate API object that acts as a smart **API Gateway or L7 (HTTP/HTTPS) load balancer** for our cluster. Its role is to manage external HTTP and HTTPS access to multiple services inside our cluster.

**Key Differences between Service and Ingress**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Service (LoadBalancer)** | **Ingress** |
| **Primary Role** | Internal L4 (TCP/UDP) load balancing & discovery. | External L7 (HTTP/HTTPS) routing and SSL termination. |
| **Exposure** | Exposes a single Service. | Exposes **multiple services** under a single IP address. |
| **Routing** | Basic round-robin load balancing. | **Advanced routing** based on host, path, headers, etc. (e.g., /api to backend, / to frontend). |
| **SSL/TLS** | Does not handle TLS termination. | Can centrally manage **SSL/TLS termination** for all services. |
| **Requirement** | Requires a cloud load balancer for external access. | Requires an **Ingress Controller** (e.g., Nginx, Traefik) to be running in the cluster to fulfill the Ingress rules. |

In conclusion, we use a Service (typically ClusterIP) to make our Pods reachable inside the cluster.We then use an Ingress resource, backed by an Ingress Controller, to define sophisticated rules for how external HTTP/S traffic should be routed to those internal Services.