**answers to basic K8S:**

1. k8s is a tool that helps managing lots of containers across many different machines - called nodes. it makes sure the application is always running, even if some of the containers fail. it can also automatically scale the application up to down on demand.

2. k8s is used to manage containerized applications across a cluster of machines. It automates many tasks involved in deploying, managing, and scaling applications, making it easier to handle complex workloads.

there are some reasons to use k8s:

- scalability: it can automatically scale applications up or down on demand.

- high availability: it ensures that the application is always running by automatically restarting failed pods or moving them to healthy nodes if something goes wrong.

- efficient resource utilization: it can distribute the application across multiple nodes, optimizing the use of computing resources.

- simplified deployment: k8s manages the rollout of new versions of the

application, ensuring smooth updates without downtime. It can roll back to a

previous version if something goes wrong.

- self healing: If a container fails or a node goes down, k8s automatically replaces or reschedules the affected Pods to maintain the desired state of the application.

examples of k8s use cases:

- microservices architecture:

if you have an application built with many microservices, K8s can manage and deploy each microservice independently, scaling them as needed and handling communication between them.

- CI/CD pipelines: k8s can automate the deployment of new application versions, running tests, and rolling out updates seamlessly.

3. the k8s architecture:

- master node (control panel) - manages and controls the cluster, maintaining the desired state of the application. the key components of this node includes:

API server - exposes the k8s API, used by administrators and other components to interact with the cluster

etcd: a key- value store that keeps all cluster data and configuration.

controller manager: monitors and maintains the desired state of the cluster.

scheduler: assigns pods to the most appropriate nodes based on resources.

- worker nodes: these are the machines where the actual application workloads run. each worker nodes communicates with the master node and carries out the tasks assigned to it. it includes the following components:

kubelet: an agent on each node that ensures containers are running as instructed in the master.

container runtime: runs the actual containers (docker for example)

kube-proxy: manages networking for the pods on the node.

- cluster-wide components: these components provide additional services that enhance the functionality and usability of the k8s cluster. it includes pods services configMap secret and pv, pvc.

4. a node is a machine in a k8s cluster that runs your applications. there are

master node and worker node as I explained in question 3.

5. pod - smallest unit in Kubernetes, running one or more containers.

manages pods, handles scaling and updates - Deployment

stable endpoint for accessing pods, enables load balancing. - Service

manages external access to services, usually HTTP/HTTPS.- Ingress

stores non sensitive configuration data for the pods. - ConfigMap

Secret- Stores sensitive data (e.g., passwords) securely.

Isolates resources within a cluster, like virtual clusters. - NameSpace

volume - persistent storage for data used by Pods, beyond container lifespan

6. k8s manages configuration through several resources that allow storing, managing and using configuration data.

1 configmap: stores non sensitive configuration data such as env and configuration files. its mounted as files or env into pods, making configuration easily configurable.

2. secret: similar to configmap but stores sensitive data such as passwords and API keys.

3. Environment variables: pass configuration data directly into containers, set in the pod spec or through configmap/ secrets. Containers read these variables at runtime.

4. command line arguments: configure applications via command line options, set directly in the pod specification or deployment manifest.

5. volumes: provides persistence storage for pods. Used to store configuration files that need to persist beyond the lifecycle of individual containers.

7. kubectl is a command-line tool for managing Kubernetes clusters. It lets you deploy apps, manage resources, view logs, and inspect the cluster's state.

8. The kubeconfig.yml file is a configuration file used by kubectl to connect to and manage Kubernetes clusters. It contains details like cluster API server addresses, authentication information, and contexts, allowing kubectl to access different clusters and namespaces.

9. The best practice is to run one container per Pod whenever possible for the simplicity and scalability it provides. That way, once each container is isolated, it is much easier to manage, understand, solve problems and scale. In addition, scaling is more efficient like that because k8s scales pods, not individual containers.

10. secret stores sensitive data such as passwords and API keys securely, using encryption. It is used when handling confidential information. On the other side, configmap stores non-sensitive configuration data such as envs and config files, and it is used when managing general application settings that don’t require encryption.

11. service provides a stable internal IP and DNS name for accessing a set of pods, enabling load balancing within the cluster. It is used when there is a need for internal communication or simple external access to a service via a single IP. On the other side, an ingress manages external access to multiple services, typically using http/https, with routing based on URL paths or domains. It is used when there is a need in an advanced routing, such as directing traffic to different services based on URLs or hostnames.

12. Kubernetes supports several types of Services, each serving different use cases:

**ClusterIP (default)**: Internal-only service, accessible only within the cluster. It is used when you want to expose a service internally within the cluster.

**NodePort**: Exposes the service on a specific port on each node's IP. Used when you need to access a service externally, directly via the node's IP and port.

**LoadBalancer**: Automatically provisions an external load balancer (like in cloud environments) to route traffic to the service. It is used when you want to expose a service externally with a single IP, and your infrastructure supports load balancers.

**ExternalName**: Maps a service to an external DNS name, without creating a proxy or load balancer. It is used when you want to route traffic to an external service outside the cluster.

**Headless Service**: Does not assign a ClusterIP, allowing direct access to the individual Pods. It is used when you need service discovery or want clients to connect directly to the specific Pod IPs.