## How does Kubernetes ensure high availability and scalability of applications?

Kubernetes ensures high availability and scalability through features like

Replication controllers, which maintain a specified number of identical pods;

horizontal pod autoscaling, which automatically adjusts the number of pods in a deployment based on resource usage;

readiness and liveness probes,

which ensure the health of application instances.

## What is a Kubernetes Service, and how does it work?

A Kubernetes Service is an abstraction that defines a logical set of pods and a policy by which to access them.

A Kubernetes Service is an abstraction that defines a logical set of pods and a policy by which to access them.

It provides a stable endpoint for accessing a set of pods, abstracting the underlying infrastructure.

Services can be exposed internally within the cluster or externally to the internet.

**Explain the difference between a Deployment and a StatefuI Set in Kubernetes.**

Deployments are used to manage stateless applications and provide features like rolling updates and scaling.

StatefulSets, on the other hand, are used to manage stateful applications that require stable, unique network identifiers and persistent storage.

StatefulSets provide ordering and uniqueness guarantees for pod management.

**What are Kubernetes Labels and Selectors, and how are they used?**

Kubernetes Labels are key-value pairs attached to objects like pods, services, and deployments.

metadata:

labels:

app: myapp

env: production

Selectors are used to query objects based on labels.

selector:

matchLabels:

app: myapp

## Explain the difference between a Deployment and a StatefuI Set in Kubernetes.

Deployments are used to manage stateless applications and provide features like rolling updates and scaling.

StatefulSets, on the other hand, are used to manage stateful applications that require stable, unique network identifiers and persistent storage. It provides ordering and uniqueness guarantees for pod management

## What is a Kubernetes Namespace, and why is it useful?

A Kubernetes Namespace is a virtual cluster within a Kubernetes cluster.

It provides a way to logically divide cluster resources between multiple users or teams.

Namespaces are useful for organizing and isolating resources, managing access control, and avoiding naming collisions.

## Explain the concept of Kubernetes Persistent Volumes (PVs) and Persistent Volume Claims (pycs).

Persistent Volumes (PVs) are storage resources provisioned by an administrator in the cluster.

Persistent Volume Claims (PVCs) are requests for storage made by users.

PVCs consume PVs, providing a way to abstract storage requirements from underlying implementation details.

## Explain Kubernetes Rolling Updates and Rollbacks.

Kubernetes Rolling Updates allow changes to be applied to a deployment with minimal downtime by gradually replacing old pods with new ones.

Rollbacks revert to a previous version of a deployment in case of issues or failures during an update.

# ****Basic Kubernetes Concepts****

1. **What is Kubernetes, and what problem does it solve?**

Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications.

It simplifies the management of containerized applications across clusters of hosts, addressing challenges related to deployment, scaling, and maintenance.

Load balancing

Self-healing (auto-restarting failed containers)

Rolling updates & rollbacks

Storage orchestration

Secret & configuration management

1. **What are the main components of Kubernetes architecture?**

Kubernetes has two primary layers:

**A) Control Plane (Master Components)**

**API Server**: Acts as the central access point for all Kubernetes commands and communication.

**Scheduler**: Allocates Pods to suitable Nodes based on resource availability and policies.

**Controller Manager**: Maintains the cluster's desired state by managing controllers.

**etcd**: A distributed key-value store that stores the entire cluster state and configuration.

**B) Worker Nodes**

**Kubelet** – communicates with the Control Plane

**Kube-Proxy** – Manages network rules for Pod communication.

**Container Runtime** (e.g., Docker, containerd) – Runs containers.

1. **What is a Pod in Kubernetes?**

A Pod is the smallest deployable unit in Kubernetes.

It can contain one or more containers (usually one main container + sidecars).

Containers in a Pod share:

The same network namespace (same IP).

Storage volumes (shared filesystem).

1. **What is a Node?**

A Node is a worker machine (physical or VM) that runs Pods.

Each Node has:

Kubelet (communicates with the Control Plane).

Container Runtime (e.g., Docker).

Kube-proxy (handles networking).

1. **What is a Cluster?**

A Cluster is a set of Nodes (workers) managed by the Control Plane.

It allows Kubernetes to distribute workloads across multiple machines for high availability and scaling

1. **What is Kubelet?**

The Kubelet is an agent running on each Node.

Its job is to:

Ensure Pods are running as expected.

Report Node & Pod status to the Control Plane.

Execute commands from the API Server (e.g., start/stop containers).

1. **What is a Kubernetes Namespace, and why is it useful?**

A Kubernetes Namespace is a virtual cluster within a Kubernetes cluster.

It provides a way to logically divide cluster resources between multiple users or teams.

Namespaces are useful for organizing and isolating resources, managing access control, and avoiding naming collisions.

1. **What is etcd and its role in Kubernetes?**

etcd is a distributed key-value store used as Kubernetes’ primary database.

It stores:

Cluster state (e.g., Pods, Nodes, Configs).

Secrets, ConfigMaps, and other metadata.

Runs on the Control Plane and is critical for cluster operations.

1. **What is the control plane?**

The Control Plane is the "brain" of Kubernetes, managing the cluster’s state.

Includes:

API Server (gateway for all operations).

Scheduler (assigns Pods to Nodes).

Controller Manager (handles scaling, healing).

etcd (cluster state storage).

1. **What is the role of the API Server?**

The API Server (kube-apiserver) is the central management point in Kubernetes.

Its jobs:

Validates & processes REST requests (kubectl commands).

Updates etcd with cluster state changes.

Communicates with kubelet on Nodes to enforce changes.

Uses RBAC (Role-Based Access Control) for security.

### 🔁 Kubernetes Objects & Workloads

1. **What is a Deployment in Kubernetes?**

A Deployment is a Kubernetes object that manages stateless applications by:

Declaring the desired state of Pods (replicas, container images).

Ensuring a specified number of Pods are running (ReplicaSet under the hood).

Supporting rolling updates and rollbacks.

1. **What is the difference between ReplicaSet and Deployment?**

| **Feature** | **ReplicaSet** | **Deployment** |
| --- | --- | --- |
| **Purpose** | Ensures a fixed number of Pod replicas are running. | Manages ReplicaSets + enables **updates/rollbacks**. |
| **Rolling Updates** | ❌ No native support | ✅ Yes (via strategy: RollingUpdate) |
| **Rollback** | ❌ Manual Pod adjustments | ✅ kubectl rollout undo |
| **Use Case** | Rarely used directly (Deployments are preferred). | Standard for stateless apps. |

1. **What is a StatefulSet?**

A StatefulSet manages stateful applications (e.g., databases) with:

Stable, unique Pod names (e.g., mysql-0, mysql-1).

Persistent storage (via PersistentVolumes).

Ordered deployment/scaling (Pod n-1 must be ready before n).

Example Use Cases:

MySQL, MongoDB, Kafka (where Pod identity/data matters).

1. **What is a DaemonSet?**

A DaemonSet ensures every Node (or a subset) runs a copy of a Pod, used for:

Node-level services (e.g., logging, monitoring).

Network plugins (e.g., Calico, Fluentd).

1. **What is a Job and CronJob?**

Job: Runs a Pod to completion (batch tasks).

CronJob: Runs Jobs on a schedule (like Linux cron).

1. **How do you scale a deployment?**

kubectl scale deployment/nginx --replicas=5

1. **How do you update a deployment without downtime?**

Use a Rolling Update (default strategy in Deployments):

1. **What is the difference between kubectl apply and kubectl create?**

| **Command** | **kubectl create** | **kubectl apply** |
| --- | --- | --- |
| **Behavior** | **Imperative** – Directly creates resources. Fails if resource exists. | **Declarative** – Creates or updates resources based on YAML. |
| **Idempotency** | ❌ No (errors on re-run). | ✅ Yes (reapplies config). |
| **Use Case** | One-time creation (e.g., kubectl create namespace test). | GitOps, CI/CD (reapplying YAML). |

### 🌐 Networking in Kubernetes

1. **What is a Service in Kubernetes?**

A Service is an abstraction that provides a stable IP address, DNS name, and load balancing for Pods, even as they scale up/down or restart.

Key Features:

Exposes Pods internally or externally.

Routes traffic using selectors (e.g., app: nginx).

Maintains network identity (Pods can die, but the Service persists).

1. **What are the types of Kubernetes Services? (ClusterIP, NodePort, LoadBalancer, ExternalName)**

| **Type** | **Description** | **Use Case** | **Example Command** |
| --- | --- | --- | --- |
| **ClusterIP** (Default) | Exposes the Service on an **internal cluster IP**. | Internal communication between Pods. | kubectl expose deploy/nginx --port=80 |
| **NodePort** | Exposes the Service on a **static port** on each Node’s IP. | Access from outside the cluster (dev/testing). | kubectl expose deploy/nginx --type=NodePort --port=80 |
| **LoadBalancer** | Provisions an **external cloud load balancer** (e.g., AWS ELB, GCP LB). | Production traffic from the internet. | kubectl expose deploy/nginx --type=LoadBalancer --port=80 |
| **ExternalName** | Maps a Service to a **DNS name** (e.g., external DB). | Integrating external services. | kubectl create svc externalname my-db --external-name=db.example.com |

1. **What is kube-proxy?**

kube-proxy is a network agent running on every Node.

Responsibilities:

Maintains network rules to forward traffic to Pods.

Implements ClusterIP, NodePort, and LoadBalancer Services.

Uses iptables/IPVS (Linux kernel features) for routing.

1. **How does DNS work in Kubernetes?**

CoreDNS (default) or kube-dns provides DNS resolution.

Naming Conventions:

Service DNS: <service-name>.<namespace>.svc.cluster.local

Pod DNS (if enabled): <pod-ip>.<namespace>.pod.cluster.local

Example:

A Pod in default namespace can reach my-service.default.svc.cluster.local

1. **What is Ingress and Ingress Controller?**

Ingress = A Kubernetes object that manages external HTTP(S) traffic (Layer 7 routing).

Defines rules for paths, hosts, and TLS.

Ingress Controller = The actual software (e.g., Nginx, Traefik, AWS ALB) that implements Ingress rules.

Must be deployed separately (not part of Kubernetes core).

1. **What’s the difference between Ingress and LoadBalancer?**

| **Feature** | **Ingress** | **LoadBalancer** |
| --- | --- | --- |
| **Layer** | Layer 7 (HTTP/HTTPS) | Layer 4 (TCP/UDP) |
| **Routing** | Path/host-based rules | Simple port forwarding |
| **Cost** | 1 LoadBalancer per Ingress Controller | 1 LoadBalancer per Service |
| **Use Case** | Web apps (multiple domains/paths) | Raw TCP services (e.g., databases) |

📦 **Kubernetes Storage & Configurations**

1. **What are ConfigMaps?**

A ConfigMap is a Kubernetes object used to store non-sensitive configuration data (e.g., environment variables, config files) separately from application code.

1. **What are Secrets in Kubernetes?**

A Secret securely stores sensitive data (e.g., passwords, API keys, TLS certificates) in an encoded (base64) or encrypted format.

Key Features:

Stored in etcd (optionally encrypted at rest).

Accessed by Pods via:

Environment variables

1. **What is the difference between ConfigMap and Secret?**

| **Feature** | **ConfigMap** | **Secret** |
| --- | --- | --- |
| **Data Type** | Non-sensitive (e.g., configs, URLs) | Sensitive (e.g., passwords, tokens) |
| **Encoding** | Plaintext | Base64-encoded (not encrypted by default) |
| **etcd Storage** | Unencrypted | Encrypted (if configured) |
| **Use Case** | Environment variables, config files | Credentials, TLS certs |

1. **What are Persistent Volumes (PV) and Persistent Volume Claims (PVC)?**

Persistent Volume (PV)

A cluster-wide storage resource (e.g., NFS, AWS EBS, GCP PD).

Manually or dynamically provisioned.

Persistent Volume Claim (PVC)

A request for storage by a user/Pod. Binds to a matching PV.

1. **What is a StorageClass?**

A StorageClass defines a "class" of storage (performance tiers, provisioning methods) for dynamic PV provisioning.

Key Features:

Enables on-demand storage (no manual PV creation).

Integrates with cloud providers (e.g., aws-ebs, gce-pd).

Defines reclaim policies (Delete, Retain).

⚙️ Helm & Package Management

1. **What is Helm in Kubernetes?**

Package manager for Kubernetes.

Manages Kubernetes applications.

1. **What are Helm charts?**

Pre-configured Kubernetes resource packages.

Contains YAML files and templates for deployment.

1. **How do you deploy an application using Helm?**

Install Helm CLI.

Add a Helm repo: helm repo add [repo-name] [repo-url].

Install chart: helm install [release-name] [chart-name].

1. **What is the difference between Helm 2 and Helm 3?**

Helm 2 server side component used triller

Helm 3 no triller improved security and simplicity

☸️ Kubernetes CLI & YAML

1. **How do you get the list of pods running in all namespaces?**kubectl get pods --all-namespaces
2. **How do you describe a pod?**kubectl describe pod pod-name
3. **How do you delete a pod?**kubectl delete pod pod-name
4. **How do you exec into a running container?**kubectl exec -it pod-name -- /bin/bash
5. **What are common sections in a Pod YAML file?**

apiVersion – Kubernetes API version (e.g., v1 for Pods).

kind – Resource type (Pod).

metadata – Pod details (name, labels, annotations).

spec – Pod specifications (containers, volumes, etc.).

containers (under spec) – List of containers in the Pod.

🛡️ Monitoring, Logging, and Health

1. **How do liveness and readiness probes work?**

Liveness Probe: Checks if the container is running. If it fails, Kubernetes restarts the container.

Readiness Probe: Checks if the container is ready to serve traffic. If it fails, the Pod is removed from service endpoints.

1. **What tools are used for Kubernetes monitoring?**

(Prometheus, Grafana)

1. **How do you view logs from a container?**kubectl logs pod-name
2. **How can you test application resilience in Kubernetes?**

Manual Tests: Delete Pods, drain nodes, throttle network.

Automated Chaos: Litmus, kube-monkey.

Validation: Verify self-healing via logs, metrics, and endpoints.

🔐 Security & RBAC

1. **What is Role-Based Access Control (RBAC)?**

RBAC is a method to regulate access to Kubernetes resources based on roles assigned to users or service accounts.

1. **What are Roles and ClusterRoles?**

Roles: Define permissions within a specific namespace.

ClusterRoles: Define permissions cluster-wide (across all namespaces).

1. **What are ServiceAccounts?**

ServiceAccounts are identities used by pods and services to authenticate with the Kubernetes API server.

1. **How do you secure secrets in Kubernetes?**

Use Kubernetes Secrets (base64-encoded).

Enable encryption at rest for Secrets.

Restrict access using RBAC.

Avoid logging or exposing Secrets in manifests.

Use external secret managers (e.g., AWS Secrets Manager, HashiCorp Vault).

1. **What is NetworkPolicy in Kubernetes?**

NetworkPolicy defines rules for controlling traffic between pods (ingress/egress) based on labels and namespaces.

🚀 CI/CD & Real-World Usage

1. **How can you automate Kubernetes deployments with GitHub Actions?**

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Deploy to Kubernetes

run: kubectl apply -f k8s-manifests/

env:

KUBE\_CONFIG: ${{ secrets.KUBE\_CONFIG }}

1. **Describe a real-world Kubernetes issue you troubleshooted and how you resolved it.**

Issue: A pod was stuck in Pending state due to insufficient CPU resources.  
Troubleshooting Steps:

Checked pod events: kubectl describe pod POD\_NAME → Found "Insufficient CPU" error.

Verified node resources: kubectl top nodes → Nodes were at full capacity.

Solution: Scaled the cluster by adding more nodes (kubectl scale or cloud provider CLI).

1. **What is a Microservice?**

A microservice is a software development approach where applications are built as small, independent services that each handle a single business function. Each service runs in its own process, can be built with different technologies, and communicates with other services via well-defined APIs (typically REST or messaging). Microservices can be deployed independently, allowing teams to update specific functionality without rebuilding the entire application. This architecture promotes scalability and resilience but requires managing distributed systems complexity.

# Projects That Benefit from Kubernetes on Azure

### Complex Microservices Architecture

* Applications with 10+ separate microservices that need orchestration
* Services with different scaling needs and resource requirements
* Systems requiring sophisticated service discovery and internal routing

### High Scalability Requirements

* Applications expecting unpredictable traffic spikes
* Systems needing horizontal scaling across many nodes
* Global applications requiring multi-region deployment

### Stateful Applications with Special Requirements

* Applications with complex stateful components beyond what App Service can handle
* Workloads requiring specialized storage configurations
* Systems with specific node affinity requirements