|  |  |
| --- | --- |
| What is namespace and how you create namespace ? | A namespace in Kubernetes is a logical way to partition resources within a cluster. You can create one using `kubectl create namespace <namespace\_name>`. |
| How do you troubleshoot if pod get image pullback error? | 1. \*\*Check Image Name\*\*: Confirm the correctness of the image name and repository.  2. \*\*Network Connectivity\*\*: Ensure network access to the image registry from the cluster.  3. \*\*Authentication\*\*: Verify the correct configuration of authentication credentials.  4. \*\*Inspect Pod Events\*\*: Use `kubectl describe pod <pod\_name>` to review specific error messages. |
| what is the Command to check pod log? | kubectl logs <pod\_name> |
| If pod failed with error crash-back loop what will you do? | 1. \*\*Inspect Logs\*\*: Check pod logs for error messages using `kubectl logs <pod\_name>`.  2. \*\*Describe Pod\*\*: Use `kubectl describe pod <pod\_name>` to identify the cause of the crash.  3. \*\*Update Configuration\*\*: Modify pod configuration to address the issue, such as correcting misconfigurations or updating resource limits.  4. \*\*Restart Pod\*\*: Manually restart the pod using `kubectl delete pod <pod\_name>` followed by `kubectl apply -f <pod\_manifest.yaml>`. |
| what are the Deployment types in Kubernetes. | ypes of Deployment: Rolling Update, Recreate, Blue-Green, Canary for managing updates and minimizing downtime. |
| What is Bluegreen and canary deployments? | 1. \*\*Blue-Green Deployment\*\*: - Blue environment serves production traffic. - Green environment is updated, and traffic is switched after testing.  2. \*\*Canary Deployment\*\*: - New version gradually rolls out to a small subset. - Performance is monitored before full rollout to all users. |
| Explain the deployment method you follow for code deployment | In AKS (Azure Kubernetes Service), we typically use rolling deployments for code deployment:   1. \*\*Rolling Deployment\*\*:  - New version is gradually rolled out while old version remains active.  - Pods are replaced incrementally, ensuring smooth transition without downtime.  - Health checks are performed to validate each new pod before replacing the next one. |
| What are the Steps involved in release management of application | In a technical pipeline, the release management steps are:   1. \*\*Source Control\*\*: Retrieve code from Git.  2. \*\*Build\*\*: Compile and package code.  3. \*\*Automated Testing\*\*: Conduct tests (unit, integration).  4. \*\*Artifact Management\*\*: Store artifacts (Docker images).  5. \*\*Deployment\*\*: Deploy to environments (dev, prod).  6. \*\*Configuration\*\*: Apply settings (env variables).  7. \*\*Monitoring\*\*: Track performance.  8. \*\*Rollback Plan\*\*: Plan for failures.  9. \*\*Approval\*\*: Manual validation gates.  10. \*\*Notification\*\*: Inform stakeholders. |
| What are the volume in Kubernetes. | In Kubernetes:   - \*\*PersistentVolume (PV)\*\*: Think of it as a storage unit in a shared storage facility. It's pre-allocated storage space available for use by applications.   - \*\*PersistentVolumeClaim (PVC)\*\*: It's like renting a storage unit. Applications (pods) request storage by claiming a portion of the available storage unit (PV). PVCs abstract away the details of the storage technology, allowing applications to request storage without needing to know where or how it's stored. |
| Wrute down deployment.yaml file and explain it. | ```yaml  apiVersion: apps/v1  kind: Deployment  metadata:  name: my-deployment  spec:  replicas: 3  selector:  matchLabels:  app: my-app  template:  metadata:  labels:  app: my-app  spec:  containers:  - name: my-container  image: nginx:latest  ports:  - containerPort: 80  ``` |
| A pod is not comming up what you will check? | 1. \*\*Pod Status\*\*: Check pod status (`kubectl get pods`) for errors or pending state. 2. \*\*Events\*\*: Inspect pod events (`kubectl describe pod <pod\_name>`) for creation or scheduling issues. 3. \*\*Container Logs\*\*: Review container logs (`kubectl logs <pod\_name>`) for startup errors. 4. \*\*Resource Availability\*\*: Verify node resources and namespace quotas for scheduling constraints. 5. \*\*Networking\*\*: Ensure connectivity to pod endpoints and external resources. 6. \*\*Pod Configuration\*\*: Confirm correctness of pod configuration. 7. \*\*Node Conditions\*\*: Check node health and readiness for accepting new pods. 8. \*\*Affinity Rules\*\*: Review pod affinity/anti-affinity rules impacting scheduling. 9. \*\*Image Availability\*\*: Validate accessibility of container image in the runtime environment. |
| Google is hosted in aks cluster pod is running but url is not working how will you troubleshoot | 1. \*\*Pod Status\*\*: Check if the pod is running (`kubectl get pods`).  2. \*\*Container Logs\*\*: Review container logs for errors (`kubectl logs <pod\_name>`).  3. \*\*Service Configuration\*\*: Verify the Kubernetes Service setup.  4. \*\*Networking\*\*: Ensure network policies and DNS are configured.  5. \*\*Ingress Controller\*\*: Check Ingress setup for routing.  6. \*\*External Connectivity\*\*: Test connectivity using `curl` or `nslookup`. |
| How to move/copy a file inside pod? | kubectl cp /path/to/local/file <namespace>/<pod\_name>:/path/inside/pod |
| How will you do Aks to acr connection where you will fill the details in aks cluster? | You'll create a Kubernetes secret containing the ACR credentials and then reference it in the AKS cluster's Kubernetes Service Principal.   1. Create a Kubernetes secret with ACR credentials.  2. Reference the secret in the AKS cluster's Kubernetes Service Principal. |
| How image will get pulled to aks cluster from acr where you will mention details of acr on aks? | You'll create a Kubernetes secret containing the ACR credentials and reference it in the AKS cluster's Kubernetes Service Principal. This allows AKS to authenticate with ACR during image pulls. |
| Difference between Statefull vs stateless | - \*\*Stateful\*\*: Applications that maintain persistent state across instances, like databases. They require dedicated storage and ensure data consistency.  - \*\*Stateless\*\*: Applications that don't store data locally and process each request independently, like web servers. They rely on external storage and don't maintain state across instances.  - \*\*DaemonSet\*\*: Ensures a copy of a pod runs on each node in the cluster, typically used for system-level services like logging or monitoring agents. Each node has its instance, ensuring availability across the cluster. |
| what is the Command to go inside pod. | kubectl exec -it <pod\_name> -- /bin/bash |
| Have you observed any state of pod other than create and running | Yes, besides "create" and "running", pods can be in states like "pending", "succeeded", "failed", or "unknown". |
| what are the Services you have used in kubernetes. | In Kubernetes, common services include:   1. \*\*ClusterIP\*\*: Exposes a service on an internal IP in the cluster.  2. \*\*NodePort\*\*: Exposes a service on each node's IP at a static port.  3. \*\*LoadBalancer\*\*: Exposes a service externally using a cloud provider's load balancer.  4. \*\*ExternalName\*\*: Maps a service to a DNS name.  5. \*\*Headless\*\*: Exposes a service without a cluster IP, useful for StatefulSets. |
| whai is configmap.yaml, secret.yaml | - \*\*ConfigMap.yaml\*\*: A Kubernetes resource used to store non-sensitive configuration data in key-value pairs, which can be consumed by pods as environment variables, command-line arguments, or as files in a volume.   - \*\*Secret.yaml\*\*: A Kubernetes resource similar to ConfigMap, but designed to store sensitive information such as passwords, tokens, or keys, encrypted at rest. They're used to securely provide this information to pods without exposing it in plaintext. |
| Explain AKS architecture, ingress and types of deployment | \*\*AKS Architecture\*\*: Consists of control plane, node pools, networking, storage, and security components.  \*\*Ingress\*\*: Manages external access to services within the cluster, often implemented using Azure services.  \*\*Types of Deployment\*\*: Rolling Update, Recreate, Blue-Green, Canary for managing updates and minimizing downtime. |
| How to check logs in kubernetes | kubectl logs <pod\_name> |
| What type of report you get from Sonar Qube and how you work on it? | SonarQube provides reports on code quality, security vulnerabilities, code smells, and test coverage. To work with it:  1. Integrate SonarQube into your CI/CD pipeline.  2. Analyze code with SonarQube scanner during build.  3. Review generated reports for issues and recommendations.  4. Address identified issues and refactor code as needed.  5. Re-run analysis regularly to maintain code quality. |
| What is pod affinity and node affinity in kubernetes | - \*\*Pod Affinity\*\*: Groups related pods together on the same node or spreads them across nodes.  - \*\*Node Affinity\*\*: Chooses nodes for pod scheduling based on specific node characteristics or labels. |
| What is rolling update? Why we use rolling update kubernetes | Rolling update is a Kubernetes deployment strategy that updates pods gradually to maintain application availability, control risk, and ensure consistency. |
| how to connect kubernetes pod to database | To connect a Kubernetes pod to a database:   1. \*\*Database Configuration\*\*: Ensure database credentials and connection details are securely stored.  2. \*\*Environment Variables\*\*: Pass database connection details to the pod as environment variables.  3. \*\*Volume Mounts\*\*: Mount any necessary database configuration files or certificates into the pod.  4. \*\*Service Discovery\*\*: Use Kubernetes services to abstract database endpoint and provide reliable access.  5. \*\*Networking Policies\*\*: Configure network policies to allow pod communication with the database. |
|  | Certainly!   - \*\*Liveness Probe\*\*: This probe checks if the container is alive and running as expected. If the probe fails (e.g., the container becomes unresponsive), Kubernetes restarts the container to restore its health.   - \*\*Readiness Probe\*\*: This probe checks if the container is ready to serve traffic. If the probe fails (e.g., the container is still initializing or experiencing issues), Kubernetes removes the container from service until it passes the readiness check, ensuring traffic is not sent to unhealthy containers. |
| Liveness & readiness |  |
| Aks Network policies | Network policies in AKS are rules that define how pods can communicate with each other and with other network endpoints. They control traffic flow by allowing or denying communication based on IP addresses, ports, and protocols. This helps to enhance security and isolate workloads within the cluster. |
|  | To implement network policies in AKS:   1. \*\*Choose Network Plugin\*\*: Select a network plugin that supports network policies, such as Azure CNI.  2. \*\*Define Policies\*\*: Create Kubernetes NetworkPolicy objects specifying allowed traffic between pods based on labels, IP ranges, ports, and protocols.  3. \*\*Apply Policies\*\*: Apply the network policies to the AKS cluster using kubectl or YAML manifests.  4. \*\*Enforcement\*\*: The chosen network plugin enforces the defined policies by allowing or denying traffic between pods based on the specified rules. |

1. \*\*What is a container?\*\*

- \*\*Question:\*\* Can you explain what a container is and how it differs from traditional virtualization?

- \*\*Answer:\*\* A container is a lightweight, portable, and self-contained environment that packages applications and their dependencies, enabling consistent and isolated execution across different environments. Unlike traditional virtualization, which virtualizes hardware, containers virtualize the operating system, allowing for efficient resource utilization and faster deployment.

2. \*\*How do you create a Docker image?\*\*

- \*\*Question:\*\* What are the steps involved in creating a Docker image from a Dockerfile?

- \*\*Answer:\*\* To create a Docker image, you write a Dockerfile with instructions for building the image, including base image, dependencies, environment setup, and application configuration. Then, you run the `docker build` command with the path to the Dockerfile, which builds the image based on the instructions and produces a reusable artifact for container deployment.

3. \*\*What is Docker Swarm?\*\*

- \*\*Question:\*\* Can you explain what Docker Swarm is and its role in container orchestration?

- \*\*Answer:\*\* Docker Swarm is a container orchestration tool that enables the management and deployment of containerized applications across a cluster of Docker hosts. It provides features such as service discovery, load balancing, scaling, and rolling updates, making it easier to deploy and manage containerized workloads in a distributed environment.

4. \*\*What is Docker prune?\*\*

- \*\*Question:\*\* What does the `docker prune` command do, and when would you use it?

- \*\*Answer:\*\* The `docker prune` command is used to remove unused Docker objects such as containers, images, volumes, and networks, freeing up disk space and cleaning up the Docker environment. It is typically used to reclaim resources consumed by inactive or dangling Docker artifacts, improving system performance and efficiency.

5. \*\*What is Docker commit?\*\*

- \*\*Question:\*\* Explain the purpose of the `docker commit` command in Docker.

- \*\*Answer:\*\* The `docker commit` command creates a new Docker image from changes made to an existing container, allowing you to capture and save container state as a new image. It is useful for creating custom images, debugging applications, or persisting changes made during container runtime.

6. \*\*What is Docker Compose?\*\*

- \*\*Question:\*\* What is Docker Compose, and how does it simplify multi-container application management?

- \*\*Answer:\*\* Docker Compose is a tool for defining and running multi-container Docker applications using a YAML configuration file. It allows you to define services, networks, and volumes for complex application stacks and manage them as a single unit, simplifying development, testing, and deployment workflows.

7. \*\*What is Docker expose?\*\*

- \*\*Question:\*\* Explain the purpose of the `EXPOSE` instruction in a Dockerfile.

- \*\*Answer:\*\* The `EXPOSE` instruction in a Dockerfile specifies the port on which a containerized application listens for incoming connections. It does not actually publish the port, but it serves as metadata for container runtime environments, indicating the intended network communication ports for the application.

8. \*\*How to see available networks in Docker?\*\*

- \*\*Question:\*\* What command would you use to view the available networks in Docker?

- \*\*Answer:\*\* To see available networks in Docker, you can use the `docker network ls` command, which lists all networks created in the Docker environment along with their attributes such as name, ID, and driver.

9. \*\*Difference between Copy vs Add:\*\*

- \*\*Question:\*\* What is the difference between the `COPY` and `ADD` instructions in a Dockerfile?

- \*\*Answer:\*\* Both `COPY` and `ADD` instructions in a Dockerfile are used to copy files from the host machine into the container. However, `COPY` is preferred for straightforward file copying, while `ADD` has additional features such as unpacking compressed files and downloading files from URLs, making it more versatile but potentially less predictable.

10. \*\*What is Entrypoint vs CMD in Docker?\*\*

- \*\*Question:\*\* Explain the difference between the `ENTRYPOINT` and `CMD` instructions in a Dockerfile.

- \*\*Answer:\*\* The `ENTRYPOINT` instruction in a Dockerfile specifies the command to be executed when the container starts, whereas the `CMD` instruction provides default arguments for the entrypoint command. `ENTRYPOINT` is typically used for defining the primary executable for the container, while `CMD` is used for specifying default behavior or parameters for the entrypoint.

11. \*\*What is a Dockerfile?\*\*

- \*\*Question:\*\* What is a Dockerfile, and how is it used in Docker image creation?

- \*\*Answer:\*\* A Dockerfile is a text file that contains instructions for building a Docker image. It specifies the base image, dependencies, environment configuration, and commands needed to create a reproducible and portable image for container deployment.

12. \*\*What is init container and runtime container?\*\*

- \*\*Question:\*\* Explain the concepts of init container and runtime container in Kubernetes.

- \*\*Answer:\*\* In Kubernetes, an init container is a specialized container that runs before the main application containers start. It is used for pre-initialization tasks such as database schema migrations or environment setup. Runtime containers, on the other hand, are the primary application containers that perform the core functionality of the application during normal operation. They run concurrently with init containers within the same pod.