### **1. What is a Kubernetes Probe?**

A **Kubernetes Probe** is a mechanism used to check the health of a container inside a pod. There are three types of probes:

1. **Liveness Probe**: Checks if the container is still running. If it fails, Kubernetes restarts the container.
2. **Readiness Probe**: Checks if the container is ready to accept traffic. If it fails, the pod is removed from the service endpoint.
3. **Startup Probe**: Checks if the container has started successfully before running other probes.

🔹 **Example of a Liveness and Readiness Probe in a Deployment:**

livenessProbe:

httpGet:

path: /health

port: 8080

initialDelaySeconds: 5

periodSeconds: 10

readinessProbe:

httpGet:

path: /readiness

port: 8080

initialDelaySeconds: 3

periodSeconds: 5

Probes ensure better availability and reliability of applications.

**2. What is the difference between a Kubernetes Deployment and a DaemonSet?**

| Purpose | Manages replicas of an application across nodes | Ensures that a pod runs on every node |
| --- | --- | --- |
| Scaling | Can be scaled up/down dynamically | Runs exactly one pod per node (or on specific nodes) |
| Use Cases | Web applications, APIs, microservices | Logging agents, monitoring, node-level services |
| Example | Nginx, Java applications | Fluentd, Prometheus Node Exporter |

Example Deployment:

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app

spec:

replicas: 3

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app

image: my-app:v1

Example DaemonSet (Runs on all nodes):

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: logging-agent

spec:

selector:

matchLabels:

name: logging-agent

template:

metadata:

labels:

name: logging-agent

spec:

containers:

- name: fluentd

image: fluentd:v1

Use Deployments for scalable apps and DaemonSets for node-level services.

### **3. How can deployments be managed in Kubernetes?**

Deployments in Kubernetes can be managed using kubectl commands or YAML manifests.

🔹 Managing Deployments:

1. Create Deployment:  
   kubectl apply -f deployment.yaml
2. Check Deployment Status:

kubectl get deployments

1. Scale a Deployment:

kubectl scale deployment my-app --replicas=5

1. Update an Image Version

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

1. Roll Back a Deployment

kubectl rollout undo deployment my-app

Use Rolling Updates for zero-downtime deployments.

### **4. How do you update the version in EKS, and how will it affect the nodes and pods?**

**To update the Kubernetes version in EKS, follow these steps:**

**🔹** Steps to Upgrade EKS Cluster:

1. Check current EKS version:

aws eks describe-cluster --name my-cluster --query cluster.version

2. Upgrade EKS using AWS CLI:

aws eks update-cluster-version --name my-cluster --kubernetes-version 1.26

3. Upgrade Node Groups:

aws eks update-nodegroup-version --cluster-name my-cluster --nodegroup-name my-nodegroup

4. Verify Nodes and Pods

kubectl get nodes

kubectl get pods -A

**Effect on Nodes and Pods:**

* Control plane is upgraded first.
* Older worker nodes may become incompatible.
* Restarting pods may be required.
* **Use Managed Node Groups** to avoid downtime.

**Always upgrade nodes and test in a staging environment before production.**

### **5. If an application is deployed in Kubernetes and needs to connect to an RDS instance, how can it establish a connection from Pods/Nodes?**

Steps to connect Kubernetes Pods to an RDS instance:

1. Create a Kubernetes Secret for Database Credentials:

kubectl create secret generic db-credentials \

--from-literal=username=admin \

--from-literal=password=securepass

2. Use the Secret in a Deployment:

env:

- name: DB\_HOST

value: mydb.abc123.us-east-1.rds.amazonaws.com

- name: DB\_USER

valueFrom:

secretKeyRef:

name: db-credentials

key: username

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: db-credentials

key: password

3. Ensure Pods Can Reach RDS:

* + Allow EKS worker nodes in the **RDS Security Group**.
  + Use **VPC Peering** or **PrivateLink** if RDS is in another VPC.

**Use IAM authentication for secure database access instead of credentials.**

### **6. How can AWS Lambda logs be stored in an EKS cluster?**

Steps to store AWS Lambda logs in an EKS cluster:

1. Enable Lambda logging to CloudWatch:

aws lambda update-function-configuration \

--function-name my-lambda \

--log-group-name /aws/lambda/my-lambda

2. Deploy Fluent Bit in EKS to collect logs:

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluent-bit

spec:

template:

spec:

containers:

- name: fluent-bit

image: fluent/fluent-bit

args:

- "--input=cloudwatch\_logs"

- "--output=elasticsearch"

**3 Use Elasticsearch/OpenSearch for log visualization.**

Fluent Bit and OpenSearch provide centralized logging for Lambda and EKS.

### **7. How can we upgrade an EKS cluster without affecting application deployments?**

**Bes**t Practices for Zero-Downtime Upgrades:

1. Use Blue-Green Upgrades – Deploy a new EKS cluster and migrate workloads.
2. Upgrade the Control Plane First – AWS handles this without downtime.
3. Use Managed Node Groups – Upgrade nodes gradually.
4. Drain Old Nodes Before Termination:

kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data

5. Monitor with Kubernetes Health Checks

kubectl get pods -o wide

Perform rolling updates and use CI/CD to automate the process.

### **8. If a UI-based application is deployed in Kubernetes and accessible via a URL, how can we deploy it, and how will the backend API respond?**

1. **Deploy Frontend UI using a Kubernetes Service and Ingress:**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: frontend**

**spec:**

**replicas: 2**

**template:**

**spec:**

**containers:**

**- name: frontend**

**image: frontend:v1**

**---**

**apiVersion: networking.k8s.io/v1**

**kind: Ingress**

**metadata:**

**name: frontend-ingress**

**spec:**

**rules:**

**- host: ui.example.com**

**http:**

**paths:**

**- path: /**

**backend:**

**service:**

**name: frontend**

**port:**

**number: 80**

**2. Backend API Service (Internal)**

**- name: BACKEND\_API\_URL**

**value: "http://backend-service"**

This makes the UI accessible via ui.example.com, and the backend API will be reachable internally.