Scenario based questions : EKS

1. Your team needs to deploy a microservices-based application on AWS using Kubernetes. How would you set up the EKS cluster?

Steps

1. VPC Setup

- Use an existing VPC or create a new one with:
  - **Public and private subnets** across multiple Availability Zones (for high availability).
  - Enable **internet gateway** and **NAT gateways** for outbound access.

2. Create EKS Cluster

- Go to **EKS Console** or use **eksctl** (simpler CLI tool).
- Example eksctl command:

bash

```
eksctl create cluster \
 --name my-eks-cluster \
 --region us-east-1 \
 --nodes 2 \
 --node-type t3.medium \
 --managed
```

3. Configure kubectl

- Update the kubectl config to talk to the new cluster:

bash

```
aws eks --region us-east-1 update-kubeconfig --name my-eks-cluster
```

4. Deploy Core Add-ons

- Install essential Kubernetes add-ons:
  - **VPC CNI plugin** (networking)
  - **CoreDNS** (service discovery)
  - **kube-proxy** (network routing)
  - Optionally: **AWS Load Balancer Controller**, **Cluster Autoscaler**, **Metrics Server**

5. Secure the Cluster

- Use **IAM roles for service accounts (IRSA)** to allow pods access to AWS services securely.

- Apply **RBAC** to restrict user and application permissions.
- Enable **logging** using CloudWatch.

6. Deploy Microservices

- Use kubectl apply -f <deployment.yaml> or **Helm** to deploy your microservices.
- Use **Kubernetes Services** (ClusterIP, NodePort, or LoadBalancer) to expose them.
- Organize using **Namespaces** (e.g., dev, test, prod) for logical separation.

7. Monitor & Auto-scale

- Enable **Horizontal Pod Autoscaler (HPA)** and **Cluster Autoscaler**.
- Use **CloudWatch Container Insights** or **Prometheus + Grafana** for observability.

8. CI/CD Integration (Optional)

- Set up pipelines using **CodePipeline**, **GitHub Actions**, or **Jenkins** to automate deployments.

2. Scenario:

You have a backend app and a frontend React app that need to be deployed on EKS. Walk us through the CI/CD deployment strategy.

CI/CD Deployment Strategy to EKS

1. Prepare Your Code Repositories

- **Frontend** and **Backend** live in separate GitHub repositories (or in separate folders of a mono-repo).
- Each project contains:
  - Dockerfile (for containerizing)
  - deployment.yaml and service.yaml files (for Kubernetes deployment)

---

2. Build & Push Docker Images (CI)

For both apps, I would:

- Set up CI using **GitHub Actions**, **CodeBuild**, or **Jenkins**.

CI Workflow Steps:

1. **Checkout Code**
2. **Build Docker Image**
3. **Tag Image** with commit SHA or version
4. **Push Image to Amazon ECR**

Example CI for backend (GitHub Actions):

yaml

```
-------
- name: Build and Push Docker Image
  run: |
    docker build -t $ECR_URI/backend:$GITHUB_SHA .
    docker push $ECR_URI/backend:$GITHUB_SHA
```

Repeat similarly for frontend app.

---

3. Kubernetes Manifests (with Image Tags)

- Use kustomize or helm or dynamic sed to update the image tags in the manifest.

Example:

yaml

```
containers:
  - name: backend
    image: <ECR_URI>/backend:<GITHUB_SHA>
```

---

4. Deploy to EKS (CD)

- Use kubectl or helm to deploy updated manifests to the EKS cluster.

CD Workflow:

- Authenticate using:

bash

```
aws eks update-kubeconfig --name my-cluster
```

- Apply manifests:

bash

```
kubectl apply -f backend-deployment.yaml
kubectl apply -f frontend-deployment.yaml
```

---

5. Automate the Pipeline

Put everything into CI/CD stages:

Stage	Tool Used	Purpose
CI Build	GitHub Actions	Build and push Docker images to ECR
CD Deploy	GitHub Actions / ArgoCD / CodePipeline	Deploy latest changes to EKS

---

6. Expose Services

- Use LoadBalancer type service for frontend.
- Use ClusterIP for backend (accessed internally), or Ingress for routing.

---

7. Post-deploy Checks

- Run health checks and notify if deployment failed.
- Optionally: Rollback if required using Helm or ArgoCD.

Why this strategy?

- Ensures **automation**, **repeatability**, and **version control**.
- Can easily extend to multiple environments (dev, staging, prod).
- Uses AWS-native services like **ECR**, **EKS**, and optionally **CodePipeline** or **ArgoCD**.

---

3. A service running in EKS can't connect to an RDS database in a private subnet. How would you debug this?

To debug a service in **EKS** that cannot connect to an **RDS** database in a **private subnet**, I would follow this **layered approach**:

Debugging Checklist

🧠 **1. Confirm RDS Endpoint & Port**

- Double-check:
  - Correct **RDS hostname** (endpoint)
  - Correct **port** (usually 5432 for PostgreSQL, 3306 for MySQL)

🌐 **2. VPC Networking Check**

- ✅ Ensure **EKS worker nodes** (EC2) are in the **same VPC** as RDS, or **VPC peering** is in place.
- ✅ Confirm RDS is in a **private subnet**, and the **Kubernetes pods** can access private subnets.
- Use kubectl get nodes -o wide to see EC2 node IPs and subnet info.

🔐 **3. Security Groups**

- Check RDS **security group inbound rules**:
  - Allow traffic from EKS **worker node security groups** or specific CIDR.
  - For example:

yaml

Type: TCP
Port: 5432
Source: sg-xxxx (EKS Node SG) or 10.0.0.0/16

📜 **4. NACL (Network ACL) Rules**

- Ensure **subnet-level NACLs** are not blocking traffic:
  - Both **inbound** and **outbound** rules should allow traffic on the correct port.

🛠️ **5. DNS Resolution**

- Run a test pod to check if RDS hostname resolves:

bash

---

kubectl run curlpod --image=amazonlinux --restart=Never -it -- bash
yum install -y bind-utils
nslookup mydb.xxxxxxxxx.rds.amazonaws.com

📶 **6. Connectivity Test**

- From the pod, test DB connectivity:

bash

---

curlpod$ telnet <rds-endpoint> 5432
Or use psql or mysql client to connect and confirm error messages.

🔐 **7. IAM or DB Auth (Optional)**

- If you're using **IAM authentication**:
  - Ensure proper IAM roles and token generation.
- Also check **DB-level user credentials** are correct.

---

8. Review Logs

- Check:
  - Pod logs: kubectl logs <pod>
  - App logs for connection timeouts or errors
  - RDS logs (from AWS Console) for rejected connections

🔁 **9. Restart or Rollout Pod**

- If you made config changes (like adding new env vars for DB endpoint), restart the pod:

bash
---
kubectl rollout restart deployment my-backend

🧱 **10. Add NAT Gateway (If Needed)**

- If pods are in private subnets and need **outbound internet** (e.g., for DNS resolution), ensure there's a **NAT Gateway**.

Bonus Tip:

- Use a **debug container** (curl, netcat, busybox) to test connectivity from within the cluster.

Summary
This issue often boils down to **VPC misconfiguration**, **security groups**, or **missing DNS/NAT settings**. By validating **network path**, **security layers**, and **service discovery**, we can quickly isolate and fix the problem.

Which add-on is **mandatory** for Kubernetes networking in EKS?
A. CoreDNS
B. VPC CNI plugin
C. Metrics Server
D. Cluster Autoscaler

✅ **Answer: B**
💡 **Reason:** The VPC CNI plugin is required for pod networking in AWS VPC.

For the frontend React app exposed to the internet, which Kubernetes Service type is best?

A. ClusterIP
B. NodePort
C. LoadBalancer
D. Headless Service

✅ **Answer: C**

💡 **Reason:** LoadBalancer type automatically provisions an AWS ELB to expose services externally.

If DNS resolution fails inside the pod when connecting to RDS, which command helps check the hostname?

A. ping
B. nslookup
C. netstat
D. curl

✅ **Answer: B**

💡 **Reason:** nslookup verifies DNS resolution inside the cluster network.