

# K8s EKS Troubleshooting Guide

Explain scenario based interview question answers for k8s and eks

Scenario-based Kubernetes (K8s) and Amazon Elastic Kubernetes Service (EKS) interview questions often test your ability to troubleshoot, optimize, and manage real-world deployments. Below are some common scenarios along with structured answers.

---

## Scenario 1: Pod is in CrashLoopBackOff

### Question:

Your application pod is stuck in a `CrashLoopBackOff` state. How do you debug and fix it?

### Answer:

#### 1. Check pod status:

```
sh

kubectl get pods -n <namespace>
```

- If you see `CrashLoopBackOff`, it means the pod is repeatedly crashing.

#### 2. Inspect logs:

```
sh

kubectl logs <pod-name> -n <namespace> --previous
```

- Look for errors like misconfiguration, missing files, or database connection failures.

#### 3. Describe pod for events and failures:

```
sh

kubectl describe pod <pod-name> -n <namespace>
```

- Check for failed mounts, resource limits, or readiness probe failures.

#### 4. Common fixes:

- If the issue is due to a missing configuration, update the ConfigMap/Secret.
- If an application is crashing due to memory, increase resource requests and limits in the deployment YAML.
- If an init container is failing, check its logs using:

```
sh
```

```
kubectl logs <pod-name> -c <init-container-name> -n <namespace>
```

---

## Scenario 2: High CPU and Memory Usage in an EKS Cluster

### Question:

Your EKS cluster is experiencing high CPU/memory utilization, leading to pod evictions and slow performance. How do you investigate and resolve it?

### Answer:

#### 1. Check node utilization:

```
sh
```

```
kubectl top nodes
```

#### 2. Check pod resource consumption:

```
sh
```

```
kubectl top pods -n <namespace>
```

#### 3. Identify the most resource-heavy workloads:

- Use Prometheus, CloudWatch, or `kubectl describe pod` to check resource requests and limits.

#### 4. Possible solutions:

- **Scale up:** If nodes are under-provisioned, increase the number of nodes.

```
sh
```

```
eksctl scale nodegroup --name=<nodegroup-name> --cluster=<cluster-name> --  
nodes=<count>
```

- **Optimize pod resource requests/limits:**

```
yaml
```

```
resources:  
  requests:  
    cpu: "500m"  
    memory: "256Mi"  
  limits:  
    cpu: "1000m"  
    memory: "512Mi"
```

- **Enable Cluster Autoscaler:**

```
sh
```

```
eksctl enable cluster-autoscaler --cluster=<cluster-name>
```

- **Evict unnecessary pods:**

```
sh
```

```
kubectl delete pod <pod-name> -n <namespace>
```

---

## Scenario 3: EKS Cluster Failing to Schedule Pods

### Question:

Your pods are stuck in a **Pending** state in EKS. How do you debug the issue?

### Answer:

1. Check pod status:

```
sh
```

```
kubectl get pods -n <namespace> -o wide
```

## 2. Describe the pod:

```
sh
```

```
kubectl describe pod <pod-name> -n <namespace>
```

- Look for errors like `Insufficient CPU`, `Insufficient Memory`, or `FailedScheduling`.

## 3. Investigate cluster capacity:

- Check available nodes:

```
sh
```

```
kubectl get nodes
```

- If no nodes are available, scale up the node group.

## 4. Potential solutions:

- **If insufficient resources**, add more nodes or optimize pod resource requests.
- **If a node selector/taint is blocking scheduling**, check node affinity:

```
yaml
```

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: "cloud.google.com/gke-nodepool"
              operator: In
              values:
                - "default-pool"
```

- **If PersistentVolume (PV) is not bound**, check with:

```
sh
```

```
kubectl get pvc -n <namespace>
```

## Scenario 4: Service is Not Accessible in EKS

### Question:

You have deployed a service in EKS, but it is not accessible. How do you debug it?

### Answer:

#### 1. Check service details:

```
sh

kubectl get svc -n <namespace>
```

#### 2. Describe the service:

```
sh

kubectl describe svc <service-name> -n <namespace>
```

#### 3. Verify endpoints:

```
sh

kubectl get endpoints -n <namespace>
```

- If no endpoints are listed, check if pods are running.

#### 4. Debug based on service type:

- **For ClusterIP:** Ensure the pod is in `Running` state and check logs.
- **For LoadBalancer:** Ensure the AWS ELB is created properly:

```
sh

aws elb describe-load-balancers --query
"LoadBalancerDescriptions[*].DNSName"
```

- **For NodePort:** Ensure security groups allow traffic to the node port.

#### 5. Check Network Policies:

```
sh
```

```
kubectl get networkpolicy -n <namespace>
```

- Modify network policies to allow traffic if needed.

## Scenario 5: Upgrading an EKS Cluster Without Downtime

### Question:

Your team needs to upgrade an EKS cluster from version 1.23 to 1.24 with minimal downtime. What steps do you take?

### Answer:

#### 1. Check current version:

```
sh

aws eks describe-cluster --name <cluster-name> --query "cluster.version"
```

#### 2. Upgrade EKS control plane:

```
sh

eksctl upgrade cluster --name=<cluster-name> --version=1.24
```

#### 3. Upgrade worker nodes:

- Create a new node group with version 1.24.
- Drain and delete old nodes:

```
sh

kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data
eksctl delete nodegroup --cluster=<cluster-name> --name=<old-nodegroup>
```

#### 4. Upgrade `kubectl` and `eksctl`:

```
sh
```

```
kubectl version --short  
eksctl version
```

## 5. Verify workloads and rollback if needed:

```
sh  
  
kubectl get pods -A
```

---

## Final Tips for EKS & Kubernetes Interviews

- **Use real-world scenarios:** Mention past experiences where you resolved similar issues.
- **Demonstrate troubleshooting skills:** Follow a structured debugging approach (logs, `kubectl describe`, metrics).
- **Explain AWS integrations:** Show knowledge of IAM roles, VPC networking, and AWS services like ALB, CloudWatch, and EFS.
- **Discuss best practices:** Talk about autoscaling, cost optimization, and security.

Would you like help with any specific scenario? 🚀

Explain node group and cluster autoscaler in aws eks

## Node Group and Cluster Autoscaler in AWS EKS

### 1. Node Group in AWS EKS

A **Node Group** in Amazon Elastic Kubernetes Service (EKS) is a set of Amazon EC2 instances that act as worker nodes in a Kubernetes cluster. These nodes run the application workloads and are managed as a group.

### Types of Node Groups in EKS

EKS supports three types of node groups:

#### 1. Managed Node Groups

- AWS automatically provisions and manages EC2 instances.
- Supports automated updates, scaling, and health monitoring.

- Uses Amazon Machine Images (AMIs) optimized for EKS.
- Example:

```
sh
```

```
eksctl create nodegroup --cluster <cluster-name> --name <nodegroup-name> --  
managed --instance-types t3.medium
```

## 2. Self-Managed Node Groups

- You manually create and manage the EC2 instances.
- Greater flexibility for custom configurations.
- Requires manual scaling, patching, and updates.

## 3. Fargate (Serverless Nodes)

- No need to manage EC2 instances.
- AWS automatically provisions resources based on pod requests.
- Used for running lightweight applications or on-demand workloads.

---

## 2. Cluster Autoscaler in AWS EKS

The **Cluster Autoscaler (CA)** automatically adjusts the number of nodes in an EKS cluster based on pod demand. It ensures that:

- New nodes are added when pods cannot be scheduled due to insufficient resources.
- Unused nodes are removed to optimize costs.

### How Cluster Autoscaler Works

1. A pod cannot be scheduled due to insufficient CPU/memory.
2. Cluster Autoscaler checks if adding a new node can resolve the issue.
3. If a suitable instance type is available, it provisions a new node.
4. If an existing node becomes underutilized, it is removed.



### 3. Setting Up Cluster Autoscaler in EKS

#### Step 1: Deploy Cluster Autoscaler

Use Helm to install Cluster Autoscaler:

```
sh

helm repo add autoscaler https://kubernetes.github.io/autoscaler
helm upgrade --install cluster-autoscaler autoscaler/cluster-autoscaler \
  --namespace kube-system \
  --set autoDiscovery.clusterName=<cluster-name> \
  --set awsRegion=<region> \
  --set extraArgs.balance-similar-node-groups=true
```

#### Step 2: Configure IAM Permissions

Attach the following IAM policy to the EKS node group role:

```
json

{
  "Effect": "Allow",
  "Action": [
    "autoscaling:DescribeAutoScalingGroups",
    "autoscaling:DescribeAutoScalingInstances",
    "autoscaling:SetDesiredCapacity",
    "autoscaling:TerminateInstanceInAutoScalingGroup"
  ],
  "Resource": "*"
}
```

#### Step 3: Enable Auto-Discovery

Edit the Cluster Autoscaler deployment:

```
sh

kubectl edit deployment cluster-autoscaler -n kube-system
```

Add the following flag:

```
yaml
```

command:

- `./cluster-autoscaler`
- `--cloud-provider=aws`
- `--namespace=kube-system`
- `--auto-discovery-cluster-name=<cluster-name>`

## 4. Scaling Behavior

Scenario	Cluster Autoscaler Action
A new pod cannot be scheduled due to lack of resources.	A new node is added.
A node is underutilized for a long time.	The node is removed.
Workloads increase suddenly.	Multiple nodes are added dynamically.

## 5. Differences Between Cluster Autoscaler and HPA/VPA

Feature	Cluster Autoscaler	HPA (Horizontal Pod Autoscaler)	VPA (Vertical Pod Autoscaler)
Scaling Type	Adds/removes worker nodes	Adds/removes pod replicas	Adjusts pod CPU/memory requests
Scaling Target	Node Groups	Deployments/ReplicaSets	Individual Pods
Use Case	Scaling infrastructure	Scaling applications	Optimizing pod resources

## 6. Best Practices

- ✓ Use **Managed Node Groups** for easy maintenance.
- ✓ Set appropriate **min/max node limits** in the Auto Scaling Group.
- ✓ Use **Cluster Autoscaler with HPA** for optimal scaling.

- ✓ Monitor scaling activities in **CloudWatch** for cost control.
- ✓ Optimize workloads to prevent unnecessary node scaling.

Would you like a hands-on demo or Terraform script for setting up an EKS Node Group and Cluster Autoscaler? 🚀

What are the new concepts we need to learn when it comes to EKS from traditional on-prem k8s setup.

## New Concepts to Learn When Moving from On-Prem Kubernetes to AWS EKS

When migrating from a **traditional on-prem Kubernetes** setup to **Amazon Elastic Kubernetes Service (EKS)**, there are several new concepts, tools, and best practices that you need to understand. Here's a breakdown of the key differences and what you need to learn:

---

### 1. EKS Cluster Management

#### Traditional On-Prem K8s

- You set up and manage the **control plane** (API server, etcd, scheduler, controllers) manually.
- You are responsible for **upgrades, backups, and high availability**.

#### AWS EKS

- **EKS provides a fully managed control plane**, so you don't need to manage API servers or etcd.
  - Upgrades are easier but require coordination for worker nodes.
  - **New Concept: EKS Versioning & Upgrades**
    - Control plane upgrades are managed via `eksctl` or AWS Console.
    - Worker nodes must be updated separately.
-

## 2. Node Management & Auto Scaling

### Traditional On-Prem K8s

- You provision and manage worker nodes manually (bare metal or VMs).
- Scaling is done using scripts or third-party tools.

### AWS EKS

- **Nodes are EC2 instances managed in Node Groups** (Managed, Self-Managed, or Fargate).
  - **Cluster Autoscaler (CA)** automatically adds/removes nodes.
  - **AWS Auto Scaling Groups (ASG)** are used for node provisioning.
  - **New Concepts:**
    - **Managed Node Groups vs. Self-Managed Node Groups**
    - **EKS Cluster Autoscaler (CA)**
    - **AWS Auto Scaling Group (ASG)**
- 

## 3. Networking & Security in AWS

### Traditional On-Prem K8s

- Uses physical network setup with internal DNS, firewalls, and VLANs.
- Network policies are often simpler.

### AWS EKS

- **VPC & Subnets:** EKS runs inside an AWS VPC with private and public subnets.
- **VPC CNI Plugin:** Allows pods to get private IPs directly from the VPC CIDR range.
- **Security Groups & IAM Roles:**
  - Nodes & workloads use **IAM roles** instead of local Linux users.
  - Security is managed via **AWS Security Groups** instead of firewall rules.
- **New Concepts:**

- AWS VPC CNI (for pod networking)
  - Private & Public Subnets in EKS
  - IAM Roles for Service Accounts (IRSA) for granting AWS permissions to pods
  - EKS Security Groups & Network Policies
- 

## 4. Storage & Persistent Volumes

### Traditional On-Prem K8s

- Uses NFS, Ceph, GlusterFS, or local persistent volumes.

### AWS EKS

- Uses **AWS Elastic Block Store (EBS)** for stateful applications.
  - Can integrate with **Amazon Elastic File System (EFS)** and **Amazon S3**.
  - **New Concepts:**
    - EBS-backed Persistent Volumes (PVs)
    - EFS for multi-node shared storage
    - S3 Integration for backups & object storage
- 

## 5. Load Balancing & Ingress

### Traditional On-Prem K8s

- Uses Nginx, HAProxy, or MetalLB for ingress/load balancing.
- Load balancers are manually configured.

### AWS EKS

- Uses **AWS Load Balancer Controller** to create ALBs/ELBs automatically.
- Supports **Application Load Balancer (ALB)** for Ingress and **Network Load Balancer (NLB)** for Layer 4 traffic.

- New Concepts:
    - AWS Load Balancer Controller
    - ALB Ingress Controller
    - NLB for high-performance load balancing
- 

## **6. Identity & Access Management (IAM)**

### **Traditional On-Prem K8s**

- Uses Kubernetes RBAC for user and service permissions.

### **AWS EKS**

- Uses IAM roles mapped to Kubernetes users and service accounts via IRSA (IAM Roles for Service Accounts).
  - New Concepts:
    - AWS IAM Integration with Kubernetes RBAC
    - IRSA (IAM Roles for Service Accounts) to grant AWS permissions to pods
    - AWS Secrets Manager for storing sensitive credentials
- 

## **7. Observability, Logging, & Monitoring**

### **Traditional On-Prem K8s**

- Uses Prometheus + Grafana for monitoring.
- Uses Fluentd/ELK Stack for logging.

### **AWS EKS**

- CloudWatch Logs & Metrics are natively integrated.
- AWS Managed Prometheus & AWS Managed Grafana are available.
- AWS X-Ray can be used for distributed tracing.

- **New Concepts:**
    - AWS CloudWatch for EKS Monitoring & Logs
    - AWS Managed Prometheus & Grafana
    - AWS X-Ray for tracing microservices
- 

## **8. CI/CD & GitOps in AWS**

### **Traditional On-Prem K8s**

- Uses Jenkins, ArgoCD, or GitLab for CI/CD.

### **AWS EKS**

- Supports AWS CodePipeline, AWS CodeBuild, and GitHub Actions.
  - ArgoCD & FluxCD can be used for GitOps.
  - **New Concepts:**
    - EKS Blue-Green & Canary Deployments with ALB
    - AWS CodePipeline for CI/CD
    - GitOps with ArgoCD in AWS
- 

## **9. Security & Compliance**

### **Traditional On-Prem K8s**

- Security is managed via internal policies and access controls.

### **AWS EKS**

- Security is AWS shared responsibility (AWS manages the control plane security).
- AWS Security Hub & GuardDuty help monitor security threats.
- **New Concepts:**

- AWS GuardDuty for anomaly detection
- AWS Security Hub for compliance monitoring
- AWS KMS for encrypting secrets

---

## Summary of New Concepts to Learn in AWS EKS

Category	Traditional On-Prem	AWS EKS
Cluster Management	Manual setup of control plane	AWS manages control plane
Node Management	Manually provisioned nodes	Managed Node Groups & Fargate
Networking	VLANs, firewalls	VPC CNI, Security Groups, Private/Public Subnets
Storage	NFS, Ceph	EBS, EFS, S3
Load Balancing	HAProxy, Nginx	AWS ALB, NLB
Access Management	RBAC	IAM Roles, IRSA
Monitoring	Prometheus	AWS CloudWatch, AWS Managed Prometheus
CI/CD	Jenkins, ArgoCD	AWS CodePipeline, GitOps
Security	Custom policies	AWS GuardDuty, Security Hub

---

## Next Steps

- If you're transitioning to AWS EKS, focus on **VPC networking, IAM, and AWS-specific storage solutions first**.
- Experiment with **Managed Node Groups, IRSA, and AWS Load Balancer Controller**.
- Set up **monitoring with CloudWatch, AWS Managed Prometheus, and Grafana**.
- Learn about **cost optimization** (right-sizing instances, using Spot Instances, scaling efficiently).



Would you like help with Terraform or eksctl scripts for setting up an EKS cluster? 😊

when to use terraform and when to use eksctl? also write scripts to create eks cluster using both approaches

## When to Use Terraform vs. eksctl for AWS EKS Deployment

### 1. Use `eksctl` When:

- ✓ You need a **quick and easy** way to create an EKS cluster.
- ✓ You prefer **declarative YAML-based** configuration.
- ✓ You don't require complex infrastructure as code (IaC) management.
- ✓ You are managing a **single EKS cluster manually** without advanced dependencies.

#### 🔴 Limitations of `eksctl` :

- Less modular and not suitable for large-scale infrastructure automation.
- Limited state management and rollback features compared to Terraform.

---

### 2. Use `Terraform` When:

- ✓ You need **infrastructure as code (IaC)** with version control.
- ✓ You are managing **multiple AWS services** beyond EKS (VPC, IAM, RDS, S3, etc.).
- ✓ You require **state management** and better collaboration.
- ✓ You need **multi-region/multi-account deployments**.

#### 🔴 Limitations of Terraform:

- Requires **more effort** for setup and maintenance.
- Debugging can be **complex** if state files are mismanaged.

---

## Creating an EKS Cluster using `eksctl`

### 1 Install eksctl:

```
sh
```

```
curl --silent --location
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -
s)_amd64.tar.gz" | tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin
```

## 2 Create an EKS Cluster with eksctl :

```
sh

eksctl create cluster \
  --name my-cluster \
  --region us-east-1 \
  --nodegroup-name my-node-group \
  --node-type t3.medium \
  --nodes 2 \
  --nodes-min 1 \
  --nodes-max 3 \
  --managed
```

✓ This command:

- Creates a new EKS cluster ( `my-cluster` ) in `us-east-1` .
- Sets up a managed node group ( `my-node-group` ) with 2 initial nodes.
- Enables **Auto Scaling** (min: 1, max: 3).

# Creating an EKS Cluster using Terraform

## 1 Install Terraform

```
sh

curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com
$(lsb_release -cs) main"
sudo apt-get update && sudo apt-get install terraform
```

## 2 Create Terraform Configuration ( `eks.tf` )

hcl

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_vpc" "eks_vpc" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "eks_subnet_1" {
  vpc_id          = aws_vpc.eks_vpc.id
  cidr_block      = "10.0.1.0/24"
  availability_zone = "us-east-1a"
}

resource "aws_subnet" "eks_subnet_2" {
  vpc_id          = aws_vpc.eks_vpc.id
  cidr_block      = "10.0.2.0/24"
  availability_zone = "us-east-1b"
}

resource "aws_eks_cluster" "eks_cluster" {
  name      = "my-cluster"
  role_arn = aws_iam_role.eks_role.arn

  vpc_config {
    subnet_ids = [aws_subnet.eks_subnet_1.id, aws_subnet.eks_subnet_2.id]
  }
}

resource "aws_iam_role" "eks_role" {
  name = "eks-cluster-role"

  assume_role_policy = jsonencode({
    Statement = [{
      Action      = "sts:AssumeRole"
      Effect      = "Allow"
      Principal = { Service = "eks.amazonaws.com" }
    }]
  })
}
```

### 3 Initialize Terraform

```
sh

terraform init
```

### 4 Apply the Terraform Configuration

```
sh

terraform apply -auto-approve
```

✅ This configuration:


- Creates an EKS cluster ( `my-cluster` ).
- Sets up a VPC and subnets.
- Assigns an IAM role for EKS.

## Which One Should You Choose?

Feature	eksctl	Terraform
Ease of Use	Simple, one-liner commands	More complex but powerful
Infrastructure as Code	YAML-based	Full IaC with version control
State Management	No state management	Maintains state files
Multi-Cloud Support	AWS only	Supports multi-cloud
Complex Deployments	Limited	Modular and scalable

## Final Recommendation

- ✅ For Quick, Simple EKS Deployment → Use `eksctl`
- ✅ For Large-Scale, Multi-Service AWS Deployments → Use `Terraform`

Would you like a full **CI/CD pipeline** example for deploying EKS clusters? 

Explain these concepts in brief wrt AWS EKS. Use code snippets as and when required -> Control plane upgrades are managed via `eksctl` or AWS Console.  
Worker nodes must be updated separately.  
Managed Node Groups vs. Self-Managed Node Groups  
EKS Cluster Autoscaler (CA)  
AWS Auto Scaling Group (ASG)  
AWS VPC CNI (for pod networking)  
Private & Public Subnets in EKS  
IAM Roles for Service Accounts (IRSA) for granting AWS permissions to pods  
EKS Security Groups & Network Policies  
EBS-backed Persistent Volumes (PVs)  
EFS for multi-node shared storage  
S3 Integration for backups & object storage  
AWS Load Balancer Controller  
ALB Ingress Controller  
NLB for high-performance load balancing  
AWS IAM Integration with Kubernetes RBAC  
IRSA (IAM Roles for Service Accounts) to grant AWS permissions to pods  
AWS Secrets Manager for storing sensitive credentials  
AWS CloudWatch for EKS Monitoring & Logs  
AWS Managed Prometheus & Grafana  
AWS X-Ray for tracing microservices  
EKS Blue-Green & Canary Deployments with ALB  
AWS CodePipeline for CI/CD  
GitOps with ArgoCD in AWS  
AWS GuardDuty for anomaly detection  
AWS Security Hub for compliance monitoring  
AWS KMS for encrypting secrets

Here's a **brief explanation** of each AWS EKS concept with **code snippets** where relevant:

---

## 1 Control Plane Upgrades (Managed via `eksctl` or AWS Console)

- The **EKS control plane** (API server, etcd, controller manager) is **managed by AWS**.
- You must **manually upgrade** it using `eksctl` or AWS Console.

```
sh
```

```
eksctl upgrade cluster --name my-cluster --region us-east-1
```

- Worker nodes must be updated **separately**.

## 2 Worker Node Updates (Must Be Updated Separately)

- Worker nodes **do not upgrade automatically**.
- You need to **replace existing nodes** with updated AMIs.

```
sh
```

```
eksctl upgrade nodegroup --cluster my-cluster --name my-nodegroup
```

## 3 Managed Node Groups vs. Self-Managed Node Groups

Feature	Managed Node Groups	Self-Managed Node Groups
Management	AWS handles lifecycle	Full control by user
Auto Scaling	Integrated	Needs manual setup
OS & AMI Updates	Auto-upgradable	Must be managed manually
Use Case	Standard workloads	Custom configurations

## 4 EKS Cluster Autoscaler (CA)

- Automatically **adds/removes nodes** based on pod scheduling needs.
- Requires IAM permissions to modify Auto Scaling Groups.

**Helm Installation:**

```
sh
```

```
helm install cluster-autoscaler autoscaler/cluster-autoscaler \
  --namespace kube-system \
  --set autoDiscovery.clusterName=my-cluster \
  --set awsRegion=us-east-1
```

---

## 5 AWS Auto Scaling Group (ASG)

- Manages EC2 instances in EKS clusters.
- Used with **Cluster Autoscaler** to scale nodes.

### Terraform Example:

```
hcl
```

```
resource "aws_autoscaling_group" "eks_nodes" {
  desired_capacity = 2
  max_size        = 5
  min_size        = 1
  launch_configuration = aws_launch_configuration.eks_nodes.id
}
```

---

## 6 AWS VPC CNI (for Pod Networking)

- Enables pods to get VPC-native IP addresses.

### Enable Prefix Delegation for Scaling:

```
sh
```

```
kubectl set env daemonset/aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

## 7 Private & Public Subnets in EKS

- **Public subnets** → for external access (ALBs, Internet Gateway).
- **Private subnets** → for internal workloads (Node Groups, RDS).

### Terraform Example:

```
hcl

resource "aws_subnet" "private" {
  vpc_id = aws_vpc.main.id
  cidr_block = "10.0.1.0/24"
  map_public_ip_on_launch = false
}
```

## 8 IAM Roles for Service Accounts (IRSA)

- **Grants AWS permissions** (S3, DynamoDB, etc.) to Kubernetes pods.

### Create an IAM Role for an EKS Pod:

```
sh

eksctl create iamserviceaccount --name my-app --namespace default \
  --cluster my-cluster --attach-policy-arn
arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
  --approve
```

## 9 EKS Security Groups & Network Policies

- **Security Groups** → control inbound/outbound traffic at EC2 level.
- **Network Policies** → control pod-to-pod communication.

### Example Network Policy:

```
yaml
```



```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend
spec:
  podSelector:
    matchLabels:
      app: frontend
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: backend
```

---

## 10 EBS-backed Persistent Volumes (PVs)

- Used for **stateful workloads**.

yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gp2
  resources:
    requests:
      storage: 10Gi
```

---

## 1 1 EFS for Multi-Node Shared Storage

- Allows multiple pods to **share storage**.

yaml

```
storageClassName: efs-sc
provisioner: efs.csi.aws.com
```

---

## 1 2 S3 Integration for Backups & Object Storage

- Pods can **directly store data** in S3 using IRSA.

yaml

```
env:
  - name: AWS_REGION
    value: "us-east-1"
  - name: S3_BUCKET
    value: "my-bucket"
```

---

## 1 3 AWS Load Balancer Controller

- Creates **ALBs & NLBs** automatically.

sh

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  --namespace kube-system
```

---

## 1 4 ALB Ingress Controller

- Automatically provisions **ALB for Ingress**.

yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: alb
```

---

## 1 5 NLB for High-Performance Load Balancing

- Layer 4 (TCP/UDP) load balancing.

yaml

```
metadata:
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
```

---

## 1 6 AWS IAM Integration with Kubernetes RBAC

- Maps IAM users to Kubernetes roles.

yaml

```
mapRoles:
  - rolearn: arn:aws:iam::123456789012:role/KubernetesAdmin
    username: admin
    groups:
      - system:masters
```

---

## 1 7 AWS Secrets Manager for Storing Credentials

- Store sensitive **database credentials & API keys**.

yaml

```
env:  
  - name: DB_PASSWORD  
    valueFrom:  
      secretKeyRef:  
        name: my-secret  
        key: password
```

---

## 1 8 AWS CloudWatch for EKS Monitoring & Logs

- Captures logs and metrics from EKS.

sh

```
aws logs describe-log-groups --log-group-name /aws/eks/my-cluster
```

---

## 1 9 AWS Managed Prometheus & Grafana

- Preconfigured Prometheus & Grafana for Kubernetes.

---

## 2 0 AWS X-Ray for Microservice Tracing

- Tracks distributed requests.

yaml

```
annotations:  
  xray.k8s.aws/enabled: "true"
```

---

## 2.1 EKS Blue-Green & Canary Deployments with ALB

- Gradual rollout strategy.

yaml

```
metadata:
  annotations:
    alb.ingress.kubernetes.io/target-group-weight: "80"
```

---

## 2.2 AWS CodePipeline for CI/CD

- Automates EKS application deployment.

yaml

```
stages:
  - name: Deploy
    actions:
      - name: DeployToEKS
        actionTypeId:
          category: Deploy
          provider: CodeDeploy
```

---

## 2.3 GitOps with ArgoCD in AWS

- Manages EKS configurations via Git.

sh

```
kubectl create namespace argocd
helm install argocd argo/argo-cd -n argocd
```

---

## 2 4 AWS GuardDuty for Anomaly Detection

- Detects **malicious activity** in EKS.

```
sh

aws guardduty list-detectors
```

---

## 2 5 AWS Security Hub for Compliance Monitoring

- Monitors **security threats & misconfigurations**.

```
sh

aws securityhub get-findings
```

---

## 2 6 AWS KMS for Encrypting Secrets

- Encrypts **EKS secrets at rest**.

```
yaml

encryptionConfig:
  - resources: secrets
    provider:
      keyArn: arn:aws:kms:us-east-1:123456789012:key/abcd1234
```

---

🔥 That's a **high-level overview** with relevant **code snippets**. Let me know if you need **detailed examples** or Terraform scripts! 🚀

