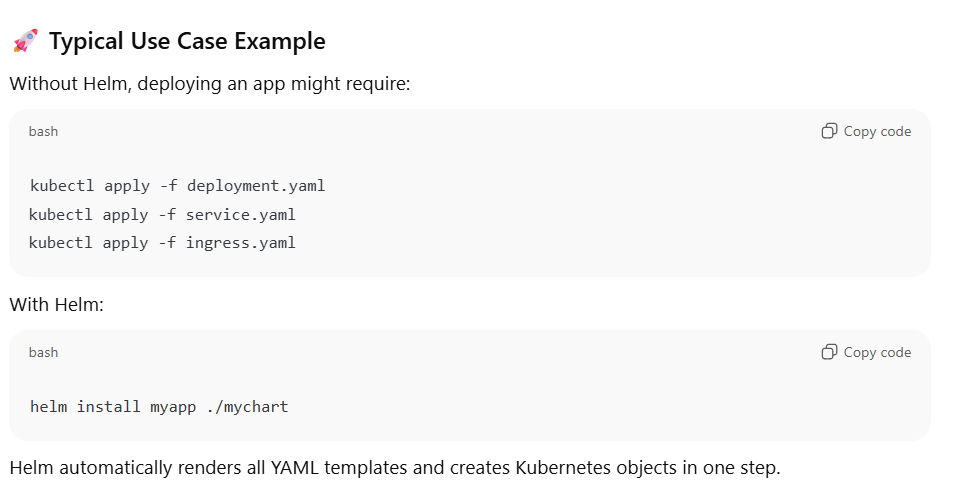
**HELM**

1. What is Helm and why is it used?

Helm is a **package manager for Kubernetes** that helps you define, install, and upgrade complex Kubernetes applications using reusable templates called **charts**.

It’s used for:

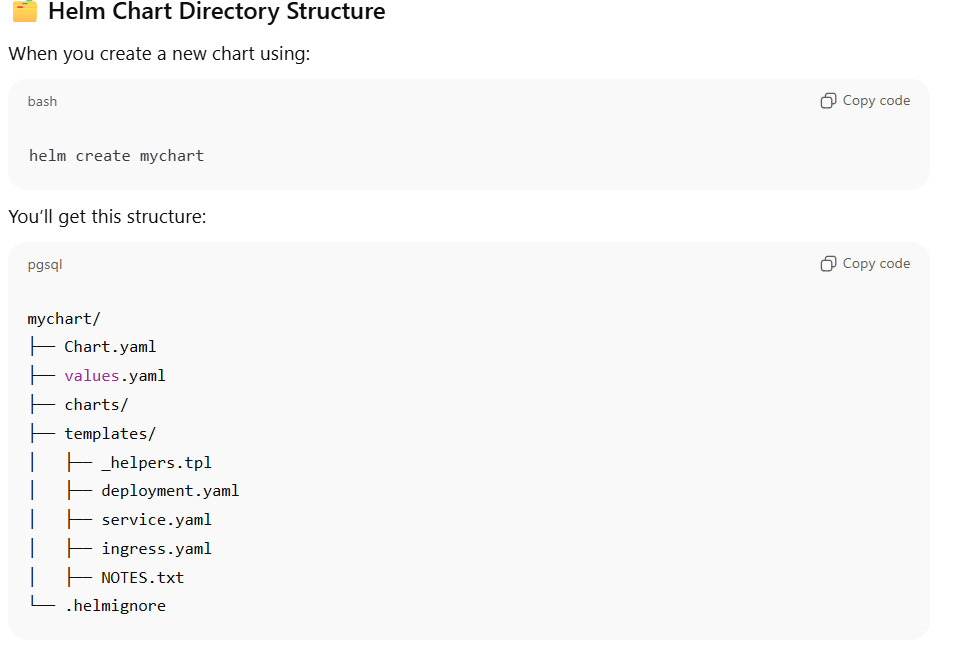
* Simplifying deployment through **parameterized templates**.
* Managing **versioned releases** (rollback support).
* Enforcing consistency across environments (dev, staging, prod).
* Reducing YAML duplication and manual errors.



A screenshot of a computer

AI-generated content may be incorrect.

1. Explain the structure of a Helm Chart.



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.



A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

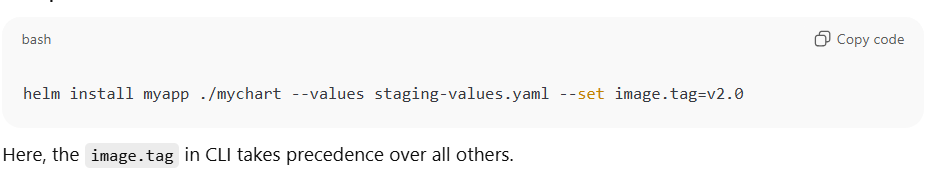
AI-generated content may be incorrect.

1. What are Helm values and how can they be overridden?

Helm values are variables used to parameterize chart templates.  
There are multiple precedence levels when overriding:

**Order of precedence (highest to lowest):**

1. --set key=value (CLI flag)
2. --values myvalues.yaml (custom values file)
3. Chart’s default values.yaml



1. How do Helm dependencies (subcharts) work?

A Helm chart can depend on other charts using the dependencies: section in Chart.yaml.



A white rectangular object with a black border

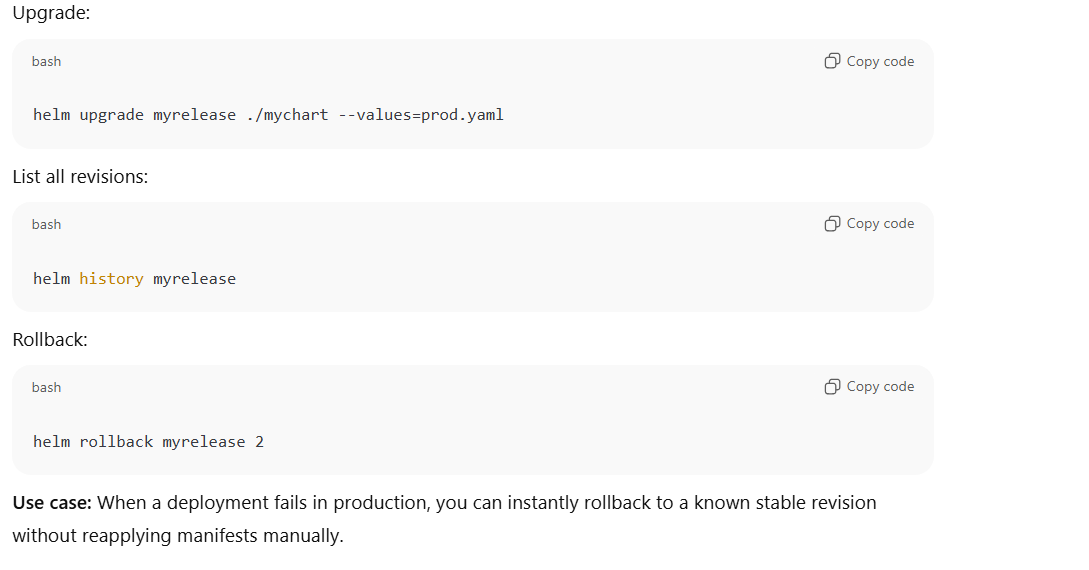
AI-generated content may be incorrect.

1. What are Helm hooks and when are they used?

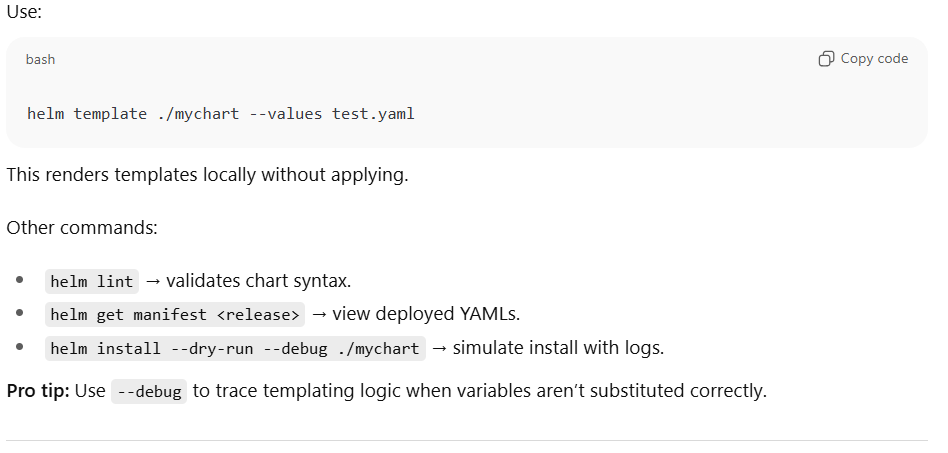
Hooks allow you to **run specific jobs or resources at lifecycle events** (install, upgrade, delete).



1. How do you perform Helm upgrade and rollback?



1. How can you debug Helm charts?

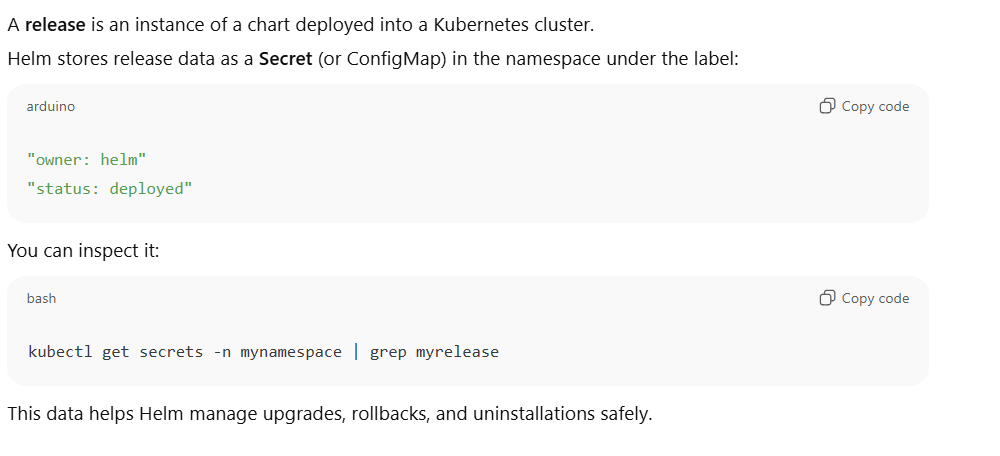


1. How do Helm chart variables work with environment-specific deployments?

A screenshot of a computer

AI-generated content may be incorrect.

1. What is a Helm release and how is it stored in Kubernetes?



1. How do you integrate Helm with CI/CD pipelines (GitLab/GitHub Actions)?



1. How do you share or distribute custom Helm charts?

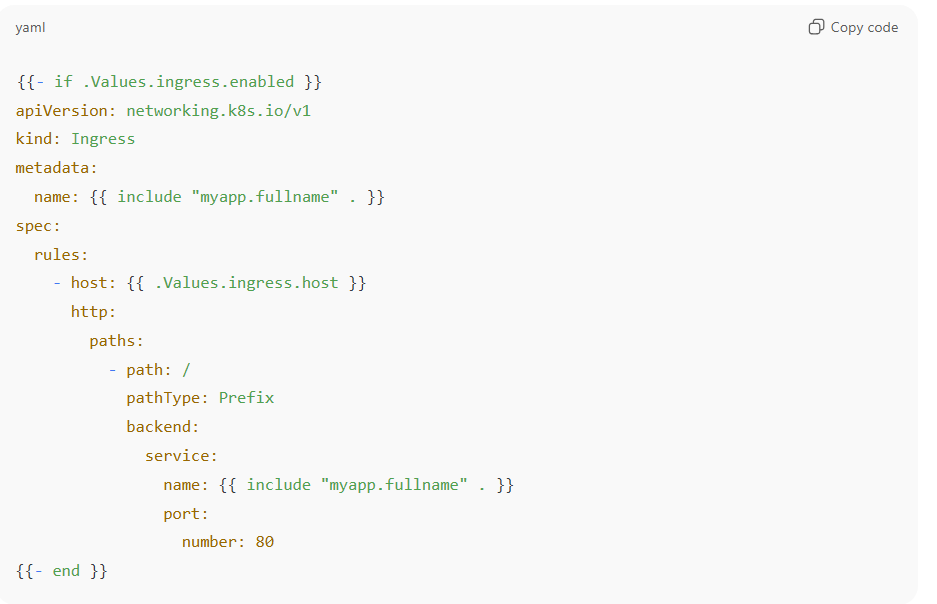
A screenshot of a computer

AI-generated content may be incorrect.

1. What is \_helpers.tpl in Helm charts?



1. How do you enable optional resources like ingress only for some environments?



A white background with black text

AI-generated content may be incorrect.

**EKS**

1. what is the usage of kubeconfig file ?

The **kubeconfig** file is a **configuration file used by kubectl, helm, and other Kubernetes clients** to know **how to connect** to a Kubernetes cluster — including **authentication, context, and cluster details**.

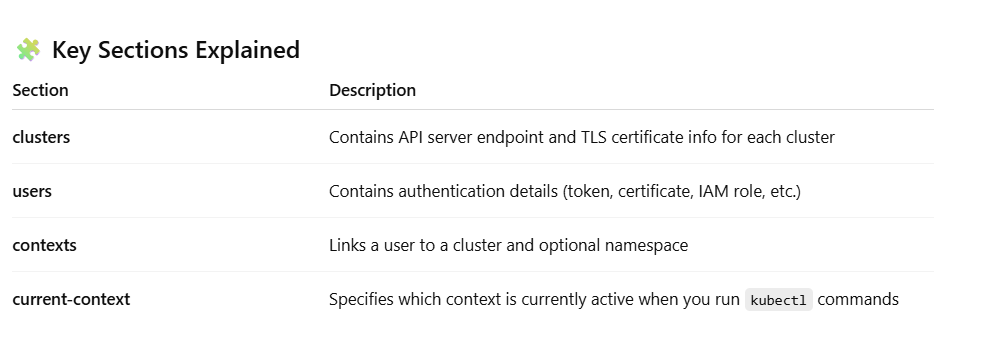
A screenshot of a computer

AI-generated content may be incorrect.

The kubeconfig file tells your kubectl **which cluster to connect to**, **how to authenticate**, and **which namespace/context to use**.

A screenshot of a computer

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Explain how OIDC integration improves security in EKS.

OpenID Connect (OIDC) integration significantly **improves security in Amazon EKS (Elastic Kubernetes Service)** by enabling **secure, fine-grained, and temporary identity-based access** to AWS resources without relying on static credentials.

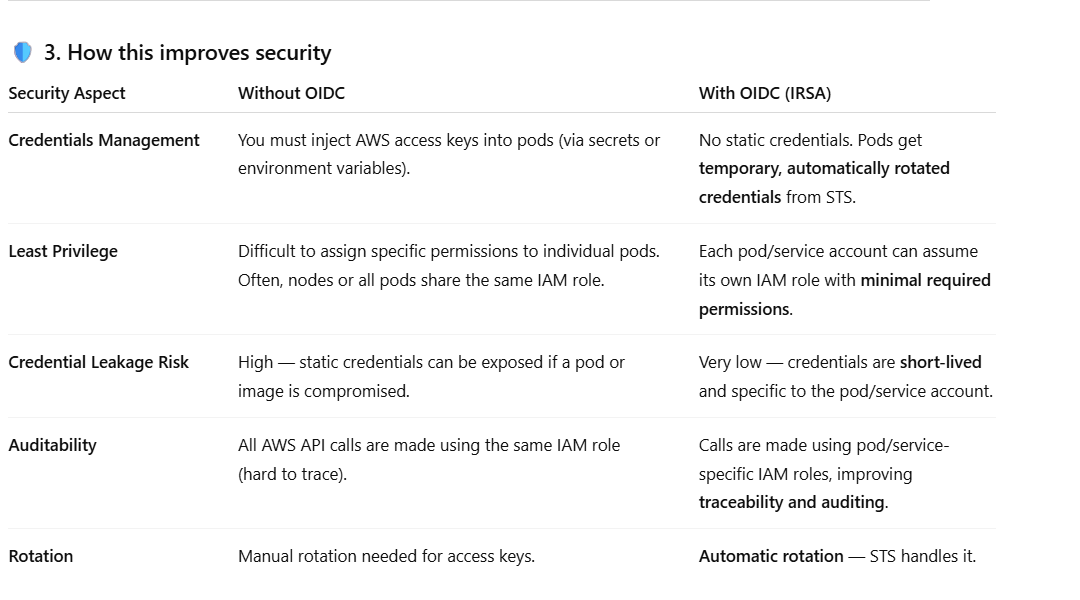
**🔐 1. What is OIDC in EKS?**

EKS integrates with **AWS IAM OIDC identity providers** to let Kubernetes service accounts assume IAM roles using **federated authentication**.  
This means instead of embedding long-lived AWS access keys in pods or ConfigMaps, pods get **short-lived, automatically rotated credentials** from AWS STS.

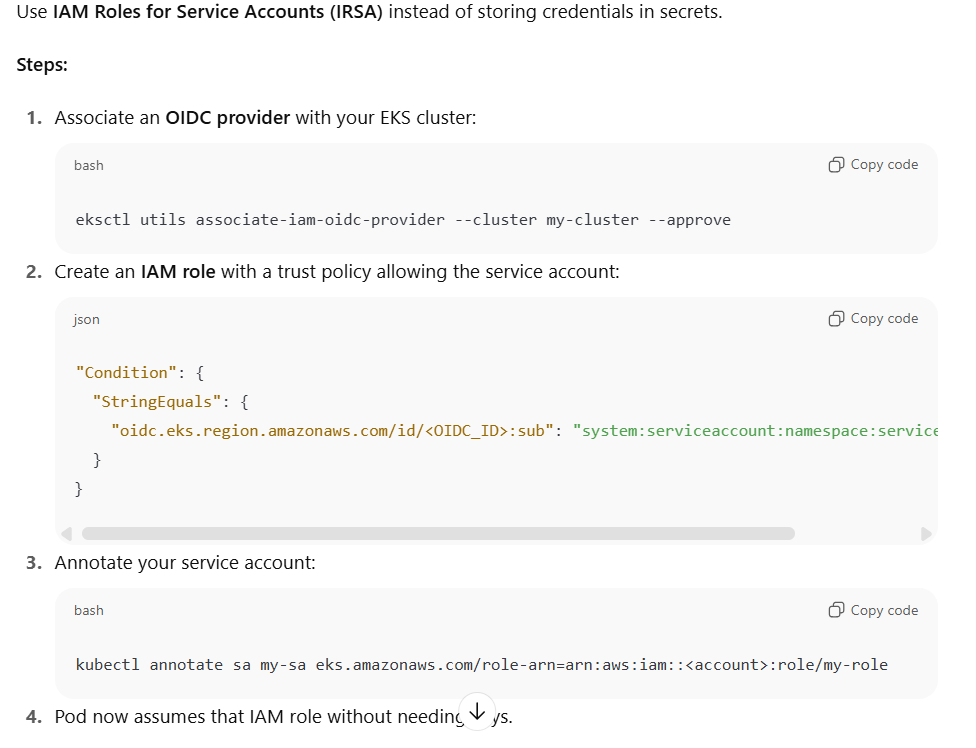
**⚙️ 2. How OIDC integration works**

When you enable OIDC in your EKS cluster:

1. EKS creates or links to an **OIDC identity provider** in IAM (using the cluster’s issuer URL).
2. You create an **IAM role** with a **trust policy** that allows tokens from this OIDC provider.
3. You associate the IAM role with a **Kubernetes service account** using annotations.
4. When a pod runs using that service account:
   * The pod gets a **signed OIDC token** from the Kubernetes API server.
   * AWS STS verifies the token against the OIDC provider.
   * If valid, it issues **temporary credentials** for the IAM role.



1. How do you securely provide AWS credentials to pods in EKS?



1. How does networking work in EKS?

**🧩 1. EKS Networking Overview**

EKS networking is a combination of:

* **AWS networking** (VPC, subnets, security groups, route tables)
* **Kubernetes networking** (CNI plugins, Services, Pods, Ingress, etc.)

EKS leverages both layers:

* AWS provides **infrastructure-level networking**
* Kubernetes provides **application-level networking**

**🌐 2. AWS VPC CNI Plugin — the Core of EKS Networking**

By default, EKS uses the **Amazon VPC CNI plugin** to manage pod networking.

**🔸 How it works**

* Each worker node (EC2 or Fargate) is attached to your **VPC** and gets an **ENI (Elastic Network Interface)**.
* Each **Pod** gets its own IP address **from the VPC subnet** — **same network as the node**.
* This means:
  + Pods are **first-class citizens** in the VPC.
  + You can reach a pod directly using its IP from any VPC resource (no NAT or overlay needed).

**🔸 Key Benefits**

✅ Native VPC networking  
✅ Simplified integration with AWS services  
✅ High performance (no encapsulation/overlay)  
✅ Direct security control using VPC security groups and NACLs

**⚙️ 3. Pod-to-Pod, Pod-to-Service, and External Communication**

Let’s see how packets actually flow:

**A. Pod-to-Pod (Same Node)**

* Communication happens via Linux virtual Ethernet pairs (veth) on the same node.
* Handled locally — doesn’t go out to the VPC.

**B. Pod-to-Pod (Different Nodes)**

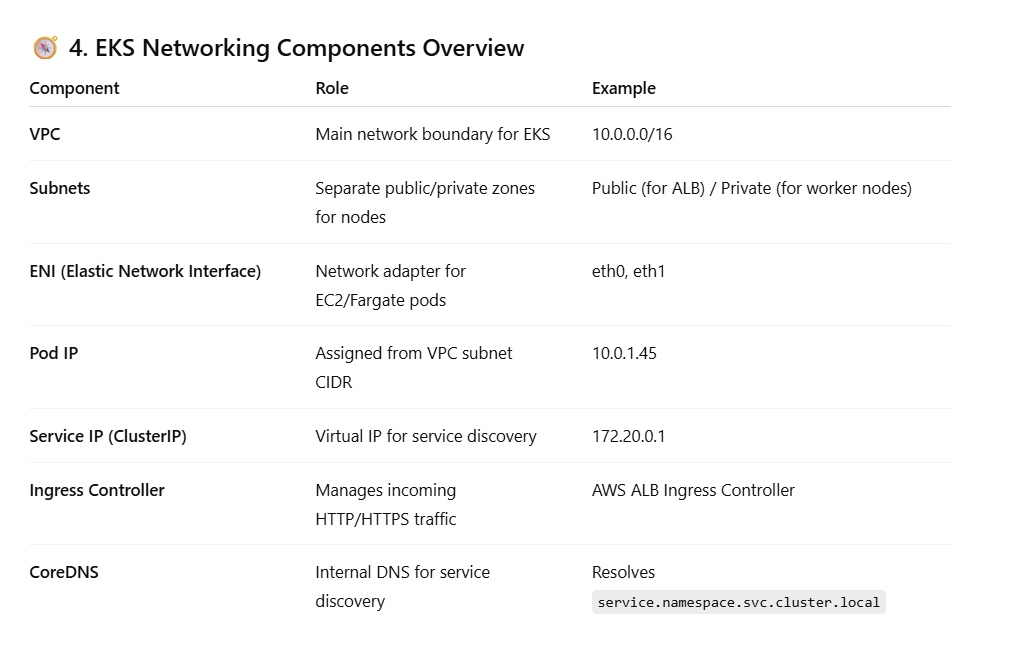
* The Amazon VPC CNI attaches secondary ENIs to each node.
* Pods get IPs from those ENIs’ subnets.
* Packets are routed through the **VPC routing layer** just like any EC2-to-EC2 communication.

**C. Pod-to-Service (ClusterIP)**

* Services use **kube-proxy** (usually with iptables or IPVS).
* kube-proxy load-balances traffic to the backend pods.

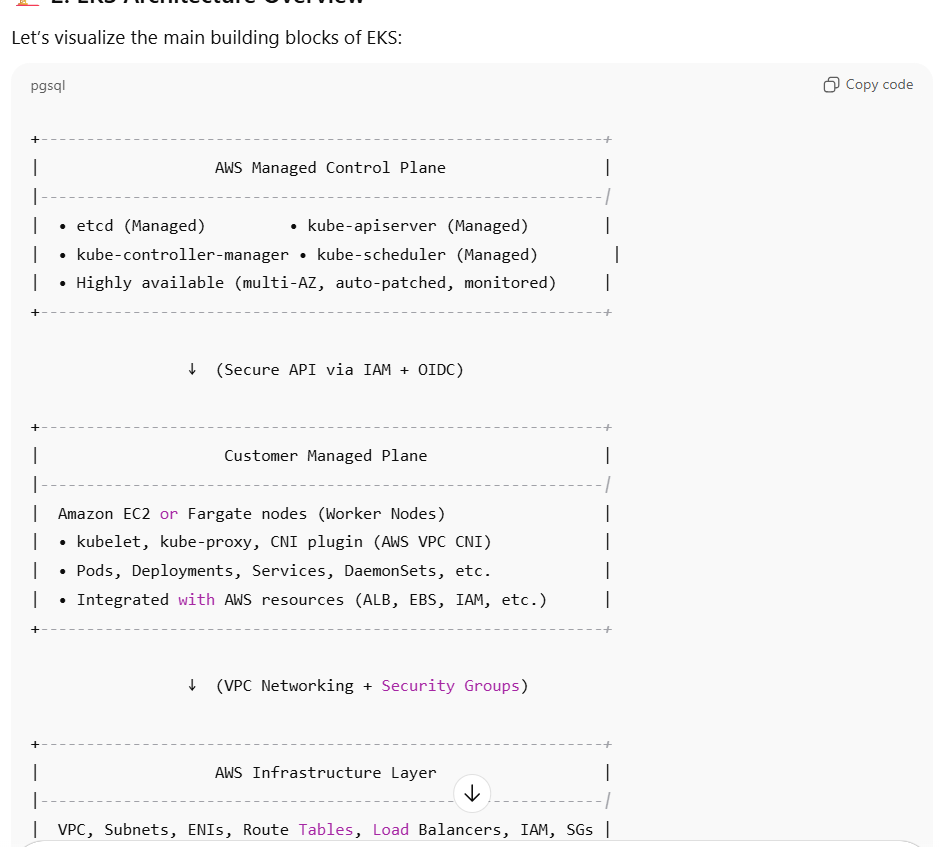
**D. External Communication (Internet/AWS Services)**

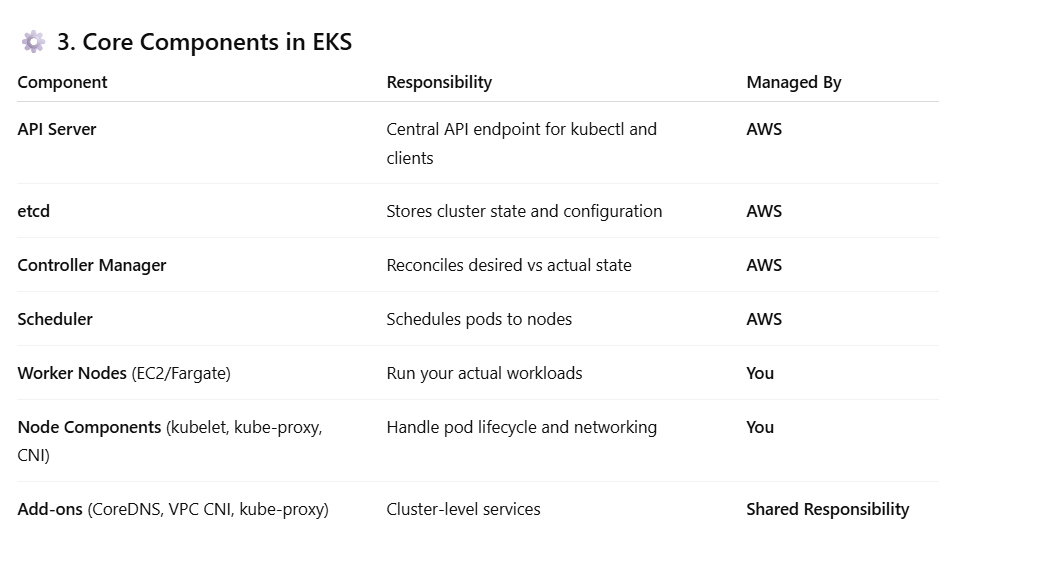
* Outbound traffic goes via the node’s ENI to the **Internet Gateway (IGW)** or **NAT Gateway**, depending on subnet type.
* For AWS APIs (like S3, DynamoDB), use **VPC Endpoints** for private communication.



1. Explain the EKS architecture and how it differs from a self-managed Kubernetes cluster.

**Amazon Elastic Kubernetes Service (EKS)** is a **managed Kubernetes control plane** provided by AWS.  
It runs upstream Kubernetes (no AWS-specific fork) — but **AWS manages the control plane**, while you manage the worker nodes and workloads.





A screenshot of a computer

AI-generated content may be incorrect.

1. How do you troubleshoot pod networking issues in EKS?

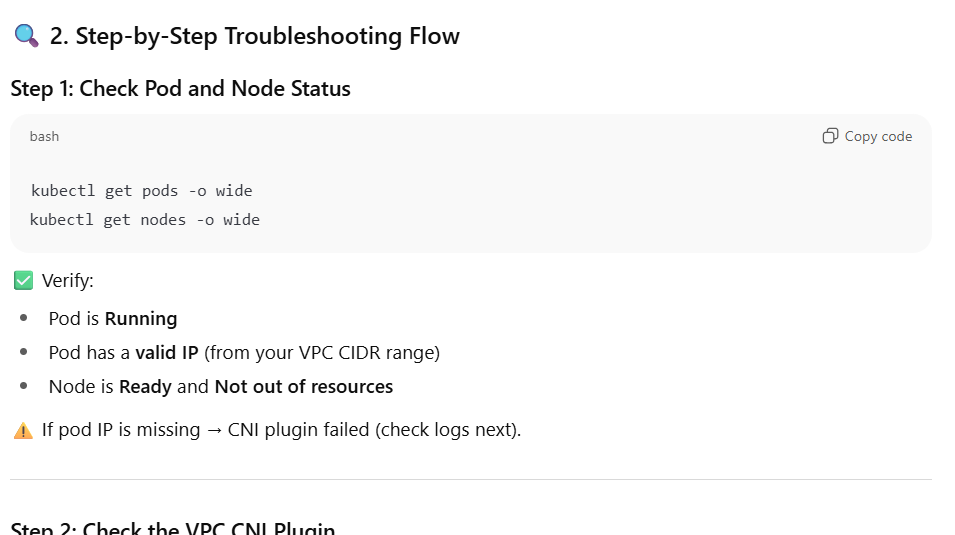
**🧩 1. First — Understand the Context**

EKS networking = **Kubernetes networking** (pods, services, CNI) + **AWS networking** (VPC, subnets, ENIs, security groups, route tables).

So when pods can’t talk, the issue could be:

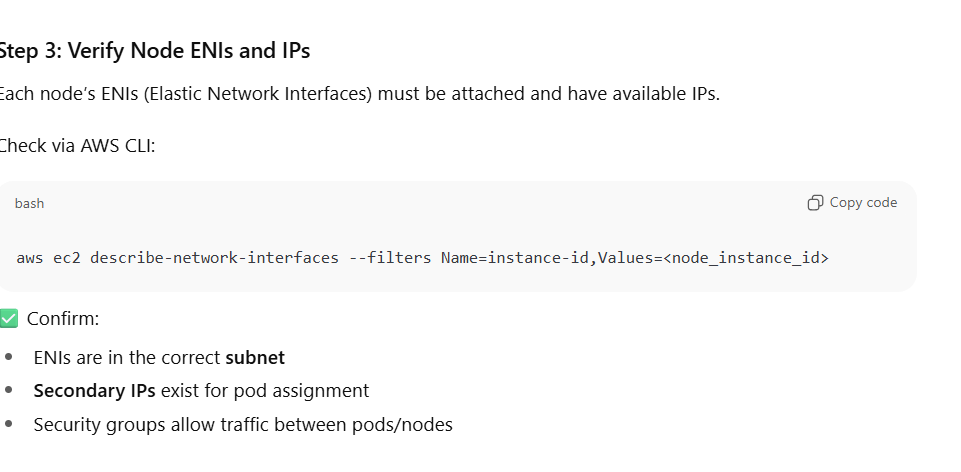
* At the **Kubernetes layer** (CNI, kube-proxy, service, DNS)
* At the **AWS layer** (VPC config, ENIs, routes, SGs)

We'll troubleshoot from **pod → node → VPC → AWS service**.



A screenshot of a computer

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A white background with black text

AI-generated content may be incorrect.