# Kubernetes Resources: Cluster Scoped and Namespace Scoped

### 🔹 What is a Kubernetes Cluster?

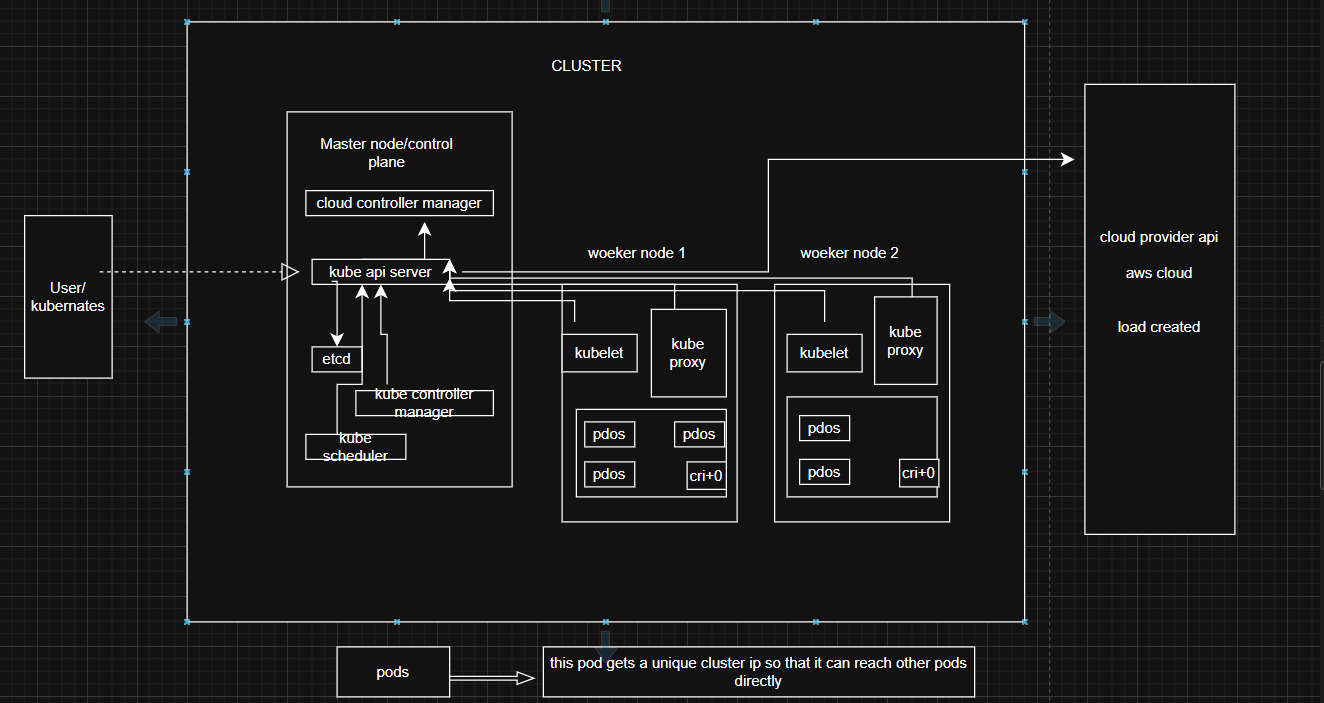
A **Kubernetes cluster** is a group of machines (nodes) that run containerized applications managed by Kubernetes. It has two main parts:

* **Master/Control Plane** – manages the cluster (API server, scheduler, controller manager, etcd).
* **Worker Nodes** – run your applications inside pods (containers).

## 🔹 Why ****Cluster****?

A **Kubernetes cluster** is needed because:

* It gives you a group of machines (nodes) managed together.
* It runs your **containerized apps** in a reliable, scalable way.
* It handles:
  + **High availability** (if one node fails, pods can move to another).
  + **Scaling** (add/remove replicas of apps easily).
  + **Load balancing** between pods.
  + **Self-healing** (restarts failed pods).



## 1. Cluster Scoped Resources

### Node

What: A worker machine in Kubernetes where pods are scheduled.

Why: Nodes provide the compute power (CPU, RAM, storage) to run workloads.

**Command: kubectl get nodes**

### PersistentVolume (PV)

What: A piece of storage in the cluster provisioned by an administrator or dynamically.

Why: Provides storage resources independent of any namespace.

**Command: kubectl get pv**

### ClusterRole

What: Defines permissions (rules) across the entire cluster.

Why: Ensures cluster-wide access control for users, groups, or service accounts.

**Command: kubectl get clusterroles**

### ClusterRoleBinding

What: Grants ClusterRole permissions to users/groups/service accounts cluster-wide.

Why: Ensures security policies are applied across namespaces.

Command: kubectl get clusterrolebindings

### CustomResourceDefinition (CRD)

What: Allows users to create custom resources.

Why: Extends Kubernetes functionality beyond built-in resources.

Command: kubectl get crd

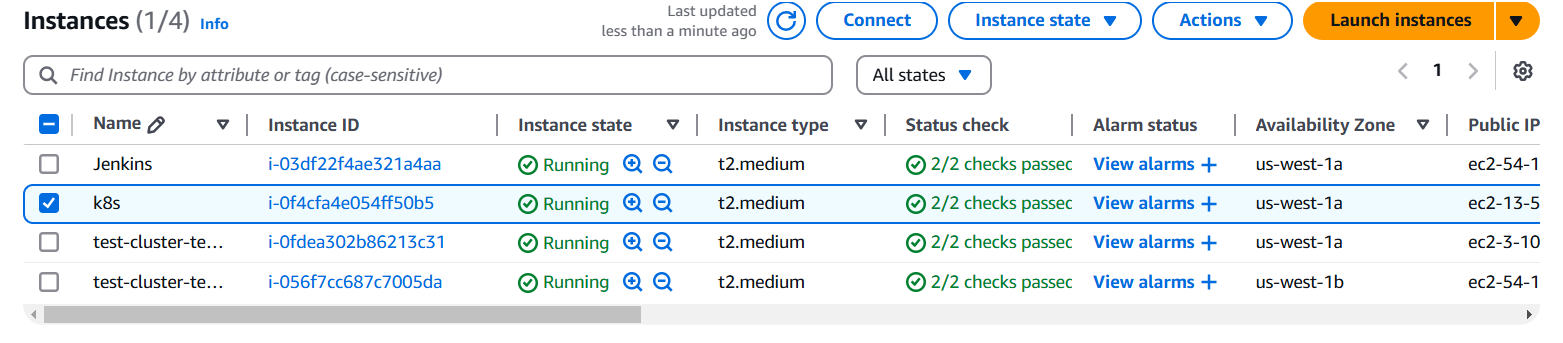
### Namespace

What: A logical partition of the cluster.

Why: Used for multi-tenancy, isolation, and resource organization.

**Eks cluster**

* Create a Linux server with ubuntu os for setting up eks cluster



* Connect your ec2 instance with putty or mobaxterm using shh client
* After that you have to install following cli tools

1. **Eksctl**

**Description: -** eksctl is a simple CLI tool for creating and managing clusters on EKS - Amazon's managed Kubernetes service for EC2. It is written in Go, uses CloudFormation

**Command:**  # for ARM systems, set ARCH to: `arm64`, `armv6` or `armv7`

ARCH=amd64

PLATFORM=$(uname -s)\_$ARCH

curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_$PLATFORM.tar.gz"

# (Optional) Verify checksum

curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_checksums.txt" | grep $PLATFORM | sha256sum --check

tar -xzf eksctl\_$PLATFORM.tar.gz -C /tmp && rm eksctl\_$PLATFORM.tar.gz

sudo mv /tmp/eksctl /usr/local/bin

1. **Kubectl**

**Description: -** kubectl is the command-line tool for interacting with Kubernetes clusters. It allows users to manage Kubernetes resources and perform various tasks such as deploying applications, inspecting and modifying cluster resources, and troubleshooting cluster issues.

**Command: -**

* Install kubectl binary with curl on Linux

curl -LO [https://dl.k8s.io/release/**$(**curl -L -s https://dl.k8s.io/release/stable.txt**)**/bin/linux/amd64/kubectl](https://dl.k8s.io/release/$(curl%20-L%20-s%20https:/dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl)

* Validate the binary (optional)

Download the kubectl checksum file:

curl -LO [https://dl.k8s.io/release/**$(**curl -L -s https://dl.k8s.io/release/stable.txt**)**/bin/linux/amd64/kubectl.sha256](https://dl.k8s.io/release/$(curl%20-L%20-s%20https:/dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256)

* Validate the kubectl binary against the checksum file:

echo "**$(**cat kubectl.sha256**)** kubectl" | sha256sum –check

* Install kubectl

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

* Test to ensure the version you installed is up-to-date:

kubectl version --client

* If kubectl version is not visible execute following commands (its optional)

chmod +x kubectl

mkdir -p ~/.local/bin

mv ./kubectl ~/.local/bin/kubectl

kubectl version --client

1. **AWS CLI**

**Description: -** The **AWS Command Line Interface (CLI)** is a tool provided by Amazon Web Services that enables users to interact with AWS services from a terminal or command prompt. It allows you to perform various AWS tasks directly from the command line, including managing resources, configuring settings, and automating workflows.

**Command: -**

apt install unzip -y

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

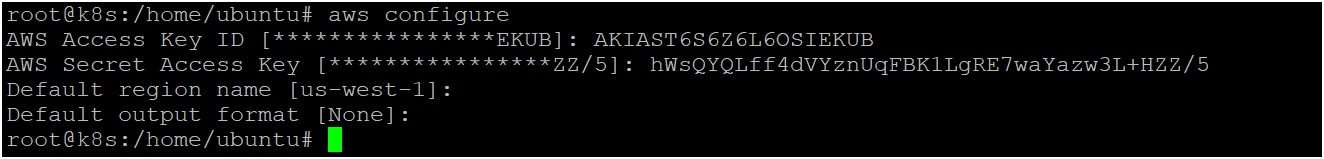
sudo ./aws/install

**Confirm the installation with the following command: -**

**aws --version**

1. **AWS access key and secret key** you have to provide before that you have to create one IAM user with administrator access permission then generate access key and secret key and you configure these keys on your terminal

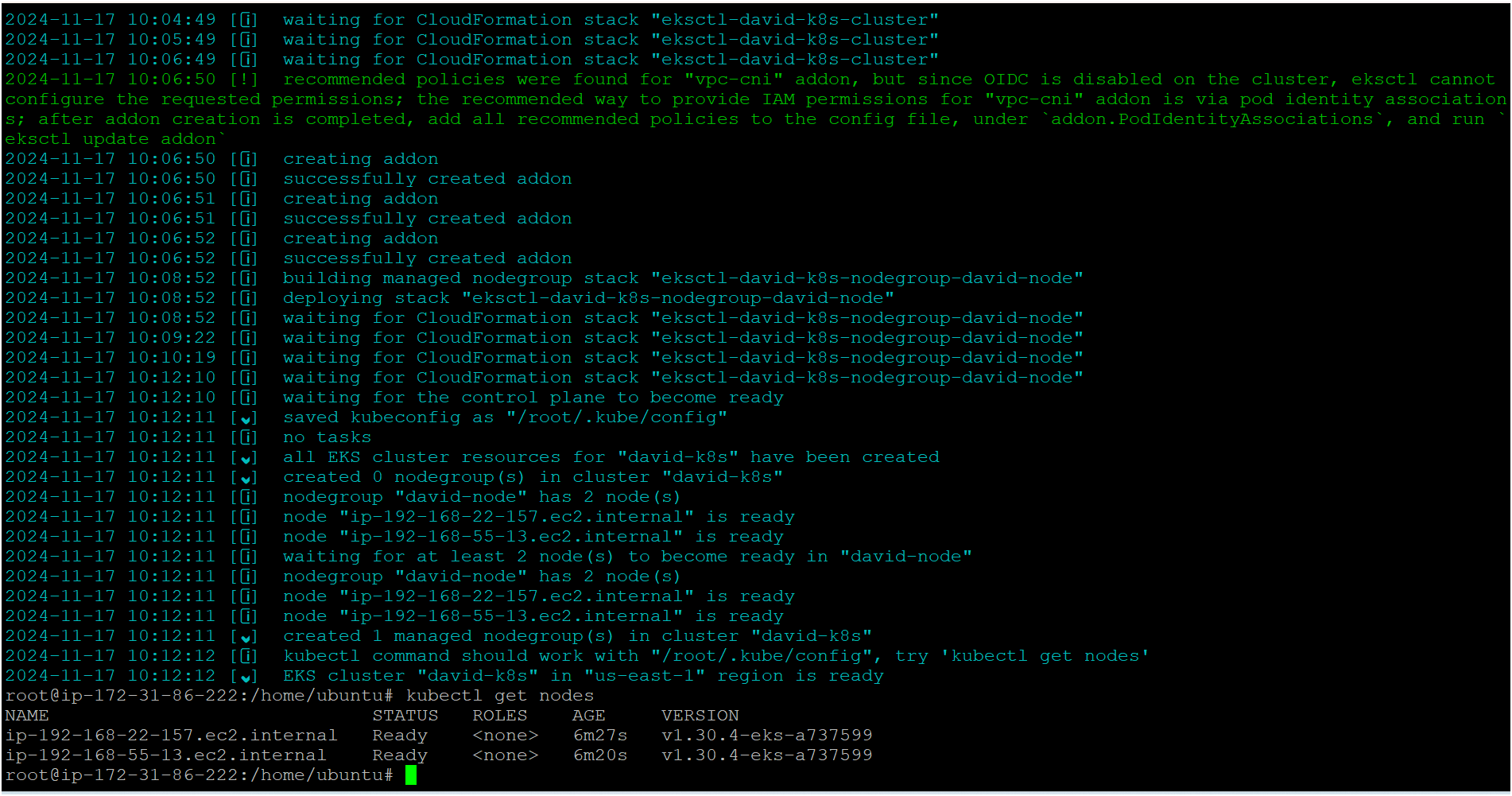
aws configure (execute this command on your terminal)



* Now you can create eks cluster by executing following command

eksctl create cluster --name <cluster\_name> --region <your\_region> --zones <availability\_zones> --nodegroup-name <nodegroip\_name> --node-type <instance\_type> --nodes <no.of.nodes>

* Now your cluster is ready as shown below image 👇



Command: kubectl get namespaces

**Namespace-**

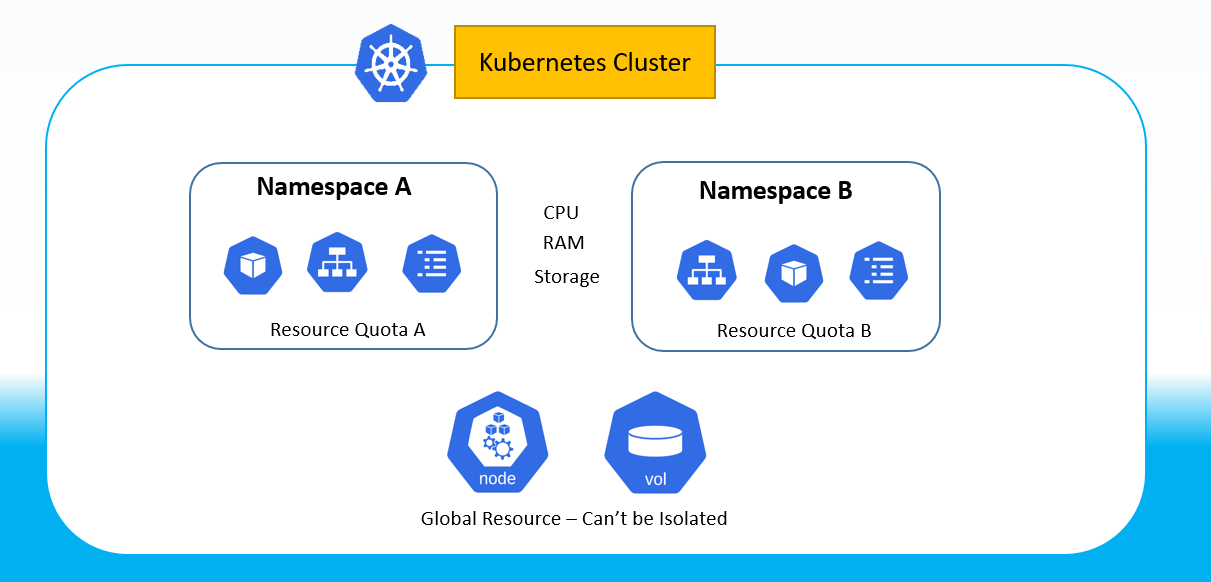
### 🔹 What is a Namespace?

* A **namespace** groups resources (Pods, Services, Deployments, ConfigMaps, etc.) logically.
* It’s useful when multiple teams, projects, or environments (like dev, test, prod) share the same cluster.
* Each namespace has its own:
  + **Name scope** (two resources with the same name can exist in different namespaces).
  + **Resource quotas** (limit CPU, memory, etc.).
  + **Access control** (via RBAC).

## 🔹 Why ****Namespace****?

A **namespace** is needed **inside a cluster** because:

* It **organizes** resources for different teams/projects/environments.
* It prevents **name conflicts** (you can have a backend service in dev and another backend service in prod).
* It allows **resource quotas** (limit CPU/memory for each team).
* It supports **security boundaries** with RBAC (who can access what).



## 2. Namespace Scoped Resources

**Kubectl create ns Shivangi(namespace -name)**

### Pod

What: The smallest deployable unit in Kubernetes containing one or more containers.

Why: Pods are the basic building blocks of applications.

**Command: kubectl get pods -n <namespace>**

### ReplicaSet

What: Ensures a specified number of pod replicas are running.

Why: Provides high availability and scaling of workloads.

**Command: kubectl get rs -n <namespace>**

### Deployment

What: Manages ReplicaSets and provides declarative updates to pods.

Why: Enables rolling updates and rollback of applications.

**Command: kubectl get deployments -n <namespace>**

### Service

What: An abstraction that defines a logical set of pods and a policy to access them.

Why: Provides stable networking and load balancing for applications.

**Command: kubectl get svc -n <namespace>**

### ConfigMap

What: Stores configuration data in key-value pairs.

Why: Decouples configuration from container images.

**Command: kubectl get configmaps -n <namespace**>

### Secret

What: Stores sensitive information such as passwords, tokens, or keys.

Why: Ensures sensitive data is managed securely.

Command: kubectl get secrets -n <namespace>

### PersistentVolumeClaim (PVC)

What: A request for storage by a user.

Why: Allows namespace workloads to consume cluster-wide persistent storage.

Command: kubectl get pvc -n <namespace>

### Role

What: Defines a set of permissions within a namespace.

Why: Restricts access control within a namespace.

Command: kubectl get roles -n <namespace>

### RoleBinding

What: Grants Role permissions to users/groups/service accounts within a namespace.

Why: Ensures fine-grained security policies.

Command: kubectl get rolebindings -n <namespace>