Kubernetes project:-

- You're doing **two things** here:

**Infrastructure Setup (Terraform)** → Creating a virtual machine (EC2) on AWS.

**Application Deployment (CI/CD + Kubernetes)** → Deploying your app automatically using GitHub Actions to a Kubernetes cluster.

Terraform:-

Note: Terraform is a **tool to automate cloud infrastructure** (like creating servers, networks, databases) using **code**.

You write code in .tf files and Terraform **builds** or **destroys** the infrastructure for you on AWS, GCP, Azure, etc.

| **Command** | **Purpose** |
| --- | --- |
| terraform init | Set up the Terraform project |
| terraform plan | Show what will be created |
| terraform apply | Actually create resources |
| terraform destroy | Delete everything |
| terraform fmt | Format your .tf files nicely |
| terraform validate | Check if your config is valid |

Main.tf

provider "aws" {

region = "us-east-1"

} # Use **AWS** as the cloud provider.

resource "aws\_instance" "my\_ec2" {

ami = "ami-002cb6127676a5723" # Amazon Linux AMI

instance\_type = "t4g.micro" # Small size instance

tags = {

Name = "MyTerraformInstance"

}

}

output "instance\_public\_ip" {

value = aws\_instance.my\_ec2.public\_ip

}

- An **AMI (Amazon Machine Image)** is like a blueprint for launching a virtual machine (EC2 instance) in AWS. It includes:

**Operating System** (Linux, Ubuntu, Windows, etc.)

**Pre-installed software** (e.g., Python, Docker, Apache)

**File system snapshot**

**Launch permissions** and configurations

### 1.🔍 How to Find AMI IDs?

Go to [EC2 AMI page](https://console.aws.amazon.com/ec2/home?" \l "Images:" \t "_new) (you need to log in).

Choose **Public Images** or **Quick Start**.

Search for OS like "Ubuntu", "Amazon Linux", etc.

You’ll get the **AMI ID**, e.g., ami-002cb6127676a5723.

You can also use the AWS CLI:

aws ec2 describe-images --owners amazon --filters "Name=name,Values=amzn2-ami-hvm-\*-x86\_64-gp2" --query 'Images[\*].[ImageId,Name]' --output table

2. Region is like the **city** where your data lives.  
Choosing the right city (region) gives you:

📍 Closer users → Faster

💸 Right region → Cheaper

⚙️ Right place → More features

⚖️ Legal match → No trouble

## Why AWS Region Matters?

### 1. ****Geographical Location****

AWS has **data centers around the world** (called "regions") like:

us-east-1 → North Virginia

ap-south-1 → Mumbai

eu-west-1 → Ireland

**Whatever you create (like EC2 instance), it will be physically located in that region.**

### 2. ****Latency & Speed****

The closer the region is to your users, the **faster the response time**.  
Example:

App for India? Use ap-south-1 (Mumbai)

App for US? Use us-east-1 (Virginia)

### 3. ****Pricing****

✅ Some regions are **cheaper**, and some are **costlier** due to demand and infrastructure.

Example:

us-east-1 is usually **cheaper**

ap-northeast-1 (Tokyo) is a bit **costlier**

### 4. ****Availability of Services****

Not all AWS features are available in every region.

✅ us-east-1 has **almost all services**  
❌ Some newer services may **not be in Mumbai** yet.

### 5. ****Legal & Compliance Reasons****

Some companies need data to stay **within country boundaries** (like EU or India) due to laws like GDPR.

- output block:

After creating the EC2 instance, Terraform will print the **public IP** of the instance in the terminal. Useful to SSH into the instance or visit it via browser.

AWS CLI:

### ****Configure AWS CLI (one-time setup)****

You need to give Terraform access to your AWS account.

aws configure

It will ask for:

AWS Access Key ID

AWS Secret Access Key

Region (e.g., us-east-1)

Output format (default: json)

## - How to Use This

### Step 1: Make sure you have these:

AWS CLI configured with your credentials

Terraform installed

### Step 2: Run in terminal

Go to the folder with this .tf file and run:

terraform init # Set up Terraform initially

terraform apply # Create the EC2 instance (type 'yes' when prompted)

You’ll see something like:

Apply complete! Resources: 1 added.Outputs:

instance\_public\_ip = "18.222.12.34"

## 💠 1. Terraform Files (in terraform/ folder)

### 🔸 main.tf

This file tells Terraform:

Use **AWS** as the cloud provider.

Create a **virtual machine (EC2 instance)** using a specific Linux image (AMI).

Print the machine's public IP when created.

### 🔸 variables.tf

Defines variables you can customize later instead of hardcoding them

### 🛠 How to Run Terraform (to create the EC2 instance)

Open terminal in the terraform/ directory and run:

terraform init # Set up Terraform

terraform apply # Create resources (type "yes" when prompted)

It will output your EC2 public IP at the end.

## 2. Kubernetes Files (in k8s/ folder)

These files tell Kubernetes how to deploy your app:

### 🔹 configmap.yaml

Stores environment variables (non-sensitive data) for your app.

### secrets.yaml

Stores sensitive data (like API keys and passwords) securely.

### secrets.yaml

Stores sensitive data (like API keys and passwords) securely.

### service.yaml

Makes your app reachable from outside using a **LoadBalancer**.

### 🔹 ingress.yaml

Allows access to your app via a custom domain (e.g., my-app.example.com). You’ll need an **Ingress Controller** (like Nginx) in your Kubernetes cluster to use this.

3. CI/CD Pipeline (in ci-cd/pipelines.yaml

This is a **GitHub Actions** workflow that runs on every git push to the main branch.

### Steps:

**Checkout code**

**Install kubectl**

**Deploy your Kubernetes YAML files**

- name: Deploy to Kubernetes

run: |

kubectl apply -f k8s/

Note:

- The **full form of YAML** is **YAML Ain’t Markup Language.**

**Ain’t** = **Is not** / **Are not** / **Has not** / **Have not** (depending on context)

- It’s a **human-readable** format for writing and storing structured data — used heavily in configuration files.

✅ YAML is like a simplified version of JSON or XML, but much easier to read and write.

- It was first called **"Yet Another Markup Language"**, but that name was later changed to emphasize that **YAML is not a markup language** (like HTML or XML), but rather a **data serialization format**.

- There’s **no difference** between .yaml and .yml — they both work the same

## JSON: JavaScript Object Notation

### 🔤

A **lightweight data format** used to store and exchange data.

Easy for **humans to read** and **machines to parse**.

Looks just like JavaScript objects.

### 🔧 Used in:

Web APIs

Config files

Mobile/web app data exchange

## 🧾 3. ****XML**** – eXtensible Markup Language

### 🔤 Full Form:

**e**Xtensible **M**arkup **L**anguage

### ✅ What is XML?

A **markup language** like HTML but for **storing structured data**.

Designed to be **both human- and machine-readable**.

More **verbose** (lengthy) than JSON.

### 🔧 Used in:

Old-school APIs (like SOAP)

Configs (e.g., Android, Maven pom.xml)

Documents (e.g., MS Word uses XML internally)

## 🔍 JSON vs XML (Quick Comparison)

| **Feature** | **JSON** | **XML** |
| --- | --- | --- |
| Syntax | Lightweight, minimal | Verbose, uses tags |
| Readability | Very readable | Slightly harder to read |
| Data Type | Supports arrays, numbers, etc | Everything is string |
| Comments | ❌ No native support | ✅ Supports comments |
| Use Case | APIs (REST), Config files | Legacy systems, SOAP APIs |