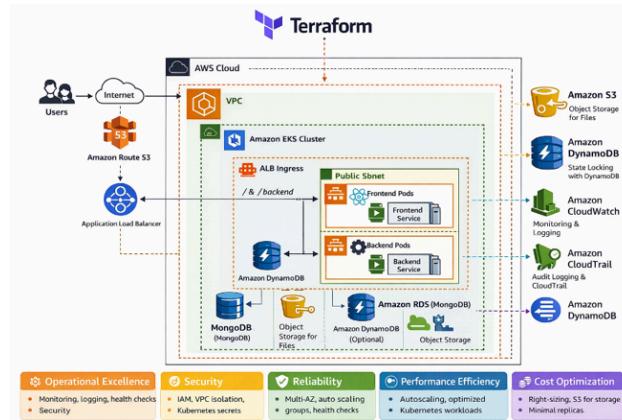


## Infrastructure-as-Code Driven Cloud-Native Three-Tier Application on AWS EKS



**Figure 1:** AWS Well-Architected three-tier cloud-native application deployed on Amazon EKS using Terraform, ALB Ingress Controller, S3 remote state, and DynamoDB locking.

### 📍 Overview

This diagram illustrates a **cloud-native three-tier application** deployed on **Amazon EKS**, fully provisioned using **Terraform** and aligned with the **AWS Well-Architected Framework**.

The solution demonstrates production-grade practices including Infrastructure as Code, Kubernetes ingress management, scalable application design, and secure state management.

### █ Architecture Components

#### Infrastructure & Platform

- Amazon VPC** – Isolated networking with public and private subnets
- Amazon EKS** – Managed Kubernetes control plane
- Managed Node Groups** – Auto-scaling worker nodes
- Terraform** – Infrastructure provisioning and lifecycle management
- S3 (Remote State)** – Centralized Terraform state storage
- DynamoDB** – Terraform state locking to prevent race conditions

#### Application Layer

- Frontend (React / Node.js)** – Served via Kubernetes Service
- Backend (API)** – REST API exposed internally
- MongoDB** – Persistent data storage inside the cluster

#### Traffic & Networking

- AWS Load Balancer Controller**
- Application Load Balancer (ALB)**
- Path-based routing**
  - `/` → Frontend service
  - `/backend` → Backend service (path rewritten at ingress)

### ⌚ AWS Well-Architected Framework Alignment

#### █ Operational Excellence

- Terraform-driven infrastructure

- Declarative Kubernetes manifests
- Repeatable deployments and rollbacks

#### Security

- IAM roles for service accounts (IRSA)
- Private subnets for workloads
- No hard-coded secrets (Kubernetes Secrets)

#### Reliability

- Health checks (liveness & readiness probes)
- ALB target group health monitoring
- Self-healing Kubernetes pods

#### Performance Efficiency

- Horizontal scalability via Kubernetes
- ALB path-based routing
- Managed EKS control plane

#### Cost Optimization

- On-demand resources only
- No over-provisioned services
- Terraform destroy for clean teardown

#### Challenges & Learnings

- Fixed **ImagePullBackOff** issues
- Resolved **ALB target group health check failures**
- Implemented **Ingress path rewriting without changing application code**
- Debugged **Kubernetes rollout and deployment selector immutability**

### Key Highlights

- End-to-end **Terraform + Kubernetes** deployment
- **Ingress path rewriting** without modifying containers
- Real-world troubleshooting (ImagePullBackOff, ALB health checks)
- Production-ready architecture

### Executive Summary – AWS EKS Three-Tier Application

This project showcases a **cloud-native three-tier application** deployed on **Amazon EKS**, designed and implemented following **AWS Well-Architected Framework principles**.

The infrastructure is fully automated using **Terraform**, with state managed securely via **Amazon S3** and **DynamoDB locking**. Application traffic is handled through an **Application Load Balancer (ALB)** using **path-based routing and ingress rewrite rules**, enabling clean separation of frontend and backend services without modifying application code.

Key operational challenges such as **container image pull failures**, **ingress health check mismatches**, and **Kubernetes rollout issues** were identified and resolved using industry best practices. The final system is **scalable, resilient, secure, and cost-efficient**, reflecting real-world cloud engineering scenarios.

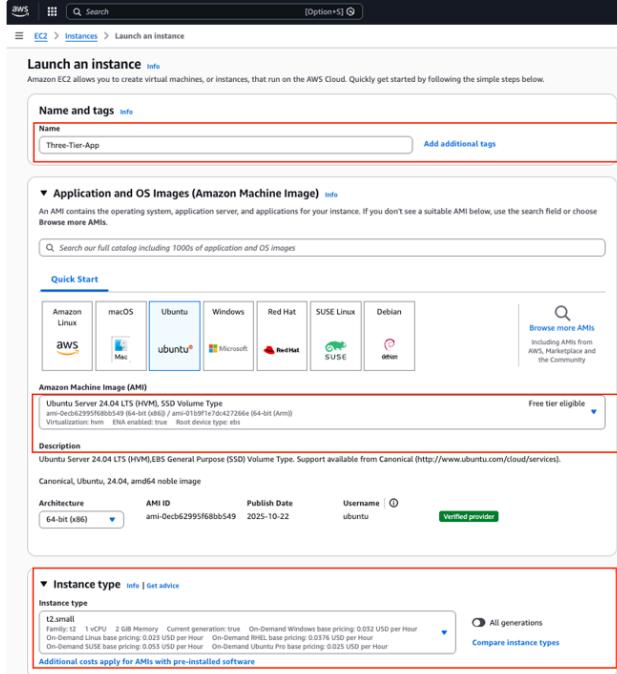
This project demonstrates hands-on expertise in:

- AWS networking and EKS
- Kubernetes operations and ingress design
- Terraform Infrastructure as Code
- Production troubleshooting and observability
- Well-Architected cloud design

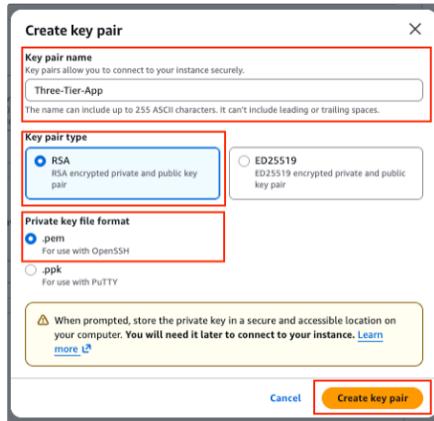
## Step 1: Set up EC2 Instance

This step can be automated using Terraform or CloudFormation in AWS. However, I chose this as we are only setting up one EC2 instance and not multiple EC2 instances.

Note: As a future enhancement – we can update the code in Terraform to launch the EC2 instance automatically.



- The next step is to create the keypair as **Three-Tier-App**



Once we enter this – a **Three-Tier-App.pem** file will be automatically downloaded in your browser.

- Now let's select an existing security group and select from the drop down options
- Configure storage at 16Gib
- Click on **Launch Instance**

The screenshot shows the AWS EC2 instance creation process. It highlights several configuration steps:

- Instance type:** t2.small
- Key pair (login):** Three-Tier-App
- Network settings:** Using default security group (default-tg-01e482c8d9b0e028)
- Configure storage:** 16 GiB gp3
- Advanced:** Shows the free tier information: "Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage."

## Step 2: Connect to the EC2 Instance remotely

Since we downloaded the PEM file we can now connect to the instance.

We need to retrieve the IP address that we need to connect so we will go to the EC2 Instance and click on **Connect**

The screenshot shows the AWS EC2 Instances list. The instance 'Three-Tier-App' is listed as 'Running'. The 'Connect' button is highlighted in red.

So we will now have the instructions needed to connect to the EC2 instance via SSH.

The screenshot shows the AWS Connect interface for the EC2 instance. The 'SSH client' tab is selected. It provides instructions for connecting using an SSH client, including the instance ID and the command to run.

Let's keep these commands handy and proceed to VS code

**Tip:** Ensure the \*.pem file is in the same folder while connecting. For this demo- The file was downloaded to Downloads so I've navigated to Downloads folder

**Advanced Tip:** For security reasons – I've edited my Inbound Rules for this security group to only allow access from my IP rather than 0.0.0.0/0.

Since this project is less than a few hours – It's okay to keep it like this.

```

sundeep@Bandeep-MacBook-Pro-2021:~$ chmod 400 "Three-Tier-App.pem"
sundeep@Bandeep-MacBook-Pro-2021:~$ ssh -i "Three-Tier-App.pem" ec2-54-90-209-184.compute-1.amazonaws.com
The authenticity of host 'ec2-54-90-209-184.compute-1.amazonaws.com (54.90.209.184)' can't be established.
ECDSA key fingerprint is SHA256:JzP5zWxRkXyfLqHnZmQd000000000000.
This key is not known by any other name.
Are you sure you want to continue connecting? (yes/no) [yes/no]: yes
Warning: Permanently added 'ec2-54-90-209-184.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1315-aws X86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/for-business
 * Bug reports: https://bugs.launchpad.net/ubuntu

System information as of Sun Dec 21 #71749 04:26:25 UTC 2025

System load: 0.0    Processes:          387
Usage:   1.4G / 16.4GB  Pending:        0
Memory usage: 100%   IPv4 address for eth0: 172.31.29.151
Swap:    0K / 0K      IPv6 address for eth0: fe80::42:1ff:fe31:2915%151

Expanded Security Maintenance for Applications is not enabled.
# updates can be applied immediately.
Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/enes or run sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

This program includes automatically included software;
the specific distribution terms for each individual file are described at
the file's location in /usr/share/doc/<file>/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
international law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

sundeep@ip-172-31-29-151:~$ 

```

We will now install the necessary updates for this Ubuntu machine

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG
ubuntu@ip-172-31-20-151:~$ sudo apt update
```

### Step 3: Install Docker Container

We will now install Docker for this instance

```
1 sudo apt install docker.io -y && sudo usermod -aG docker $USER && newgrp
docker
```

### Step 4: Set up AWS CLI and IAM User, Connect to the host machine

Navigate to AWS Console and click on IAM

The screenshot shows the AWS IAM console with the 'Applications' section open. It displays a list of applications with a 'Create application' button. The left sidebar shows recently visited services like IAM, EC2, and Lambda.

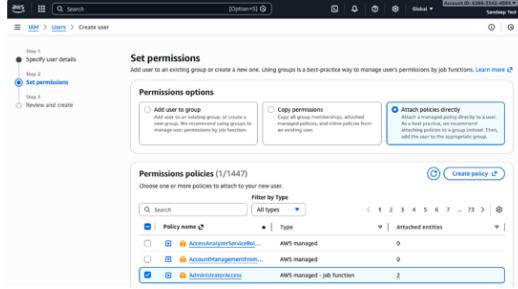
Now click on **Users** and click on **Create User**

The screenshot shows the AWS IAM console with the 'Users' section open. It lists two users: 'sashcell64' and 'sashcell97'. The 'Create user' button is highlighted with a red box. The left sidebar shows the 'Access management' section with 'User groups' and 'Users' selected.

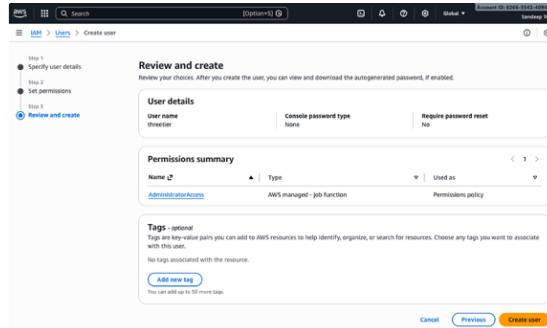
Next Step add a Name for the User and then click on Next

The screenshot shows the 'Specify user details' step of the IAM user creation wizard. It asks for a user name ('sashcell') and勾选s 'AdministratorAccess'. The 'Next Step' button is highlighted with a red box.

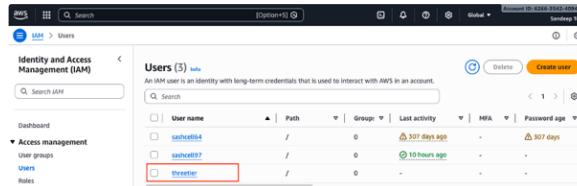
Now select the level of Permissions you need for access. We've selected **AdministratorAccess** however this depends in an enterprise or corporate setup. Then click on Next



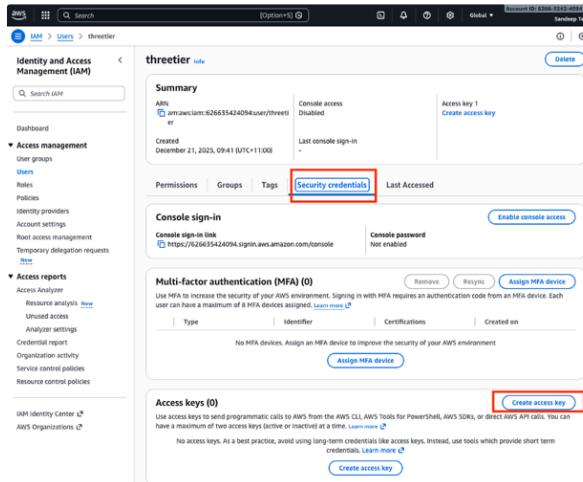
Review the options and then click on **Create User**



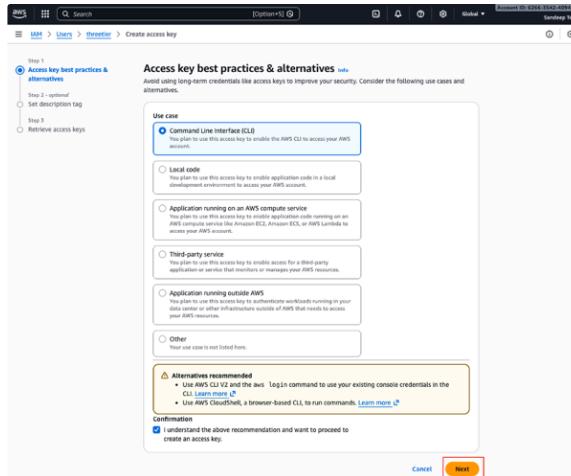
Once this is setup – we need to ensure that we can provide CLI access – Hence for this we now need to click on the newly created Administrator access



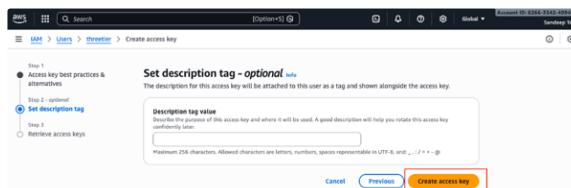
Select **Security Credentials** and then click on **Create access key**



Now select **Command Line Interface**, select the Confirmation box and then click on **Next**



Move on to the next step and click on **Create Access Key**



Now we will have the Access Keys. We need to make sure that we keep these keys secure and safe.

We will now need to download AWS CLI on our machine

```
1 curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
      "awscliv2.zip"
2 sudo apt install -y unzip curl
3 unzip awscliv2.zip
4 sudo ./aws/install
```

Check the AWS version by going to

```
1 aws --version
```

We will now connect to the host machine securely

```
1 aws configure
```

It will prompt you to enter

- **Access Key ID**
- **Secret Access Key**
- **Default region name:** You can use `US-east-1` for this demo
- **Default output format:** Enter `json`

Once this is done we are now ready for the next step.

## Step 5 : Install Terraform and set up remote Backend

Since the AWS CLI has been installed and configured we will now install Terraform on the backend.

```
1 sudo apt-get update && sudo apt-get install -y gnupg software-properties-common
2
3 wget -O- https://apt.releases.hashicorp.com/gpg | \
4 gpg --dearmor | \
5 sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null
6
7 echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
8 https://apt.releases.hashicorp.com $(grep -oP '(?=<=UBUNTU_CODENAME=).*)' /etc/os-release || lsb_release -cs) main" | \
```

```

9 sudo tee /etc/apt/sources.list.d/hashicorp.list
10
11 sudo apt update
12
13 sudo apt-get install terraform

```

Let's validate the install now

```
1 terraform -v
```

```
ubuntu@ip-172-31-20-151:~$ terraform -v
Terraform v1.14.3
on linux_amd64
```

Let's clone the repository and listing the contents we will see the below

```
1 git clone https://github.com/sandeep-ssh/aws-eks-terraform-three-tier.git
```

```
ubuntu@ip-172-31-20-151:~$ git clone https://github.com/sandeep-ssh/aws-eks-terraform-three-tier.git
Cloning into 'aws-eks-terraform-three-tier'...
remote: Enumerating objects: 57, done.
remote: Counting objects: 100% (57/57), done.
remote: Compressing objects: 100% (45/45), done.
remote: Total 57 (delta 7), reused 57 (delta 7), pack-reused 0 (from 0)
Receiving objects: 100% (57/57), 199.91 KiB | 12.49 MiB/s, done.
Resolving deltas: 100% (77/77), done.
ubuntu@ip-172-31-20-151:~$ cd aws-eks-terraform-three-tier
ubuntu@ip-172-31-20-151:~/aws-eks-terraform-three-tier$ ls
README.md backend frontend k8s_manifests terra-config
ubuntu@ip-172-31-20-151:~/aws-eks-terraform-three-tier$ ll
total 49
drwxrwxr-x 7 ubuntu ubuntu 4096 Dec 21 07:54 /
drwxr-x--- 8 ubuntu ubuntu 4096 Dec 21 07:54 ./
-rw-rw-r-- 1 ubuntu ubuntu 6148 Dec 21 07:54 .DS_Store
drwxrwxr-x 8 ubuntu ubuntu 4096 Dec 21 07:54 git/
-rw-rw-r-- 1 ubuntu ubuntu 31 Dec 21 07:54 README.md
drwxrwxr-x 4 ubuntu ubuntu 4096 Dec 21 07:54 backend/
drwxrwxr-x 4 ubuntu ubuntu 4096 Dec 21 07:54 frontend/
drwxrwxr-x 2 ubuntu ubuntu 4096 Dec 21 07:54 k8s_manifests/
drwxrwxr-x 2 ubuntu ubuntu 4096 Dec 21 07:54 terra-config/
ubuntu@ip-172-31-20-151:~/aws-eks-terraform-three-tier$
```

Terraform tracks the state of your infrastructure in a file called `terraform.tfstate`. By default, this state is stored locally, which is neither secure nor scalable. To follow Terraform best practices, we configure a **remote backend using Amazon S3**, ensuring safe, centralized, and collaborative state management.

We will now create a S3 bucket for the Terraform backend.

```
1 aws s3api create-bucket \
2   --bucket sandeep-eks-terraform-three-tier \
3   --region us-east-1
```

Enabling versioning so we can track any changes in S3

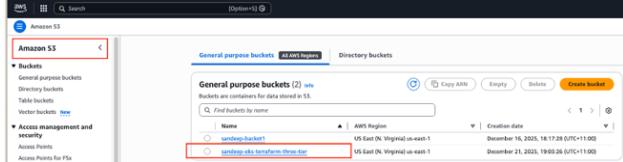
```
1 aws s3api put-bucket-versioning \
2   --bucket sandeep-eks-terraform-three-tier \
3   --versioning-configuration Status=Enabled
```

Enable Default Encryption

```
1 aws s3api put-bucket-encryption \
2   --bucket sandeep-eks-terraform-three-tier \
3   --server-side-encryption-configuration '{ \
4     "Rules": [ \
5       "ApplyServerSideEncryptionByDefault": { \
6         "SSEAlgorithm": "AES256" \
7       } \
8     ] \
9   }'
```

```
● sandeephedge@Sandeeps-MacBook-Pro-2023 Downloads % aws configure
AWS Access Key ID [xxxxxxxxxxxxx-MLUT]: AKIAZDZIBYPDR3E42L7
AWS Secret Access Key [xxxxxxxxxxxxx-0Uw]: Lqb7ulPSaJzIxSNQd/HBimG3o3Ujp0jnxEuDy0L
Default region name [us-east-1]: us-east-1
Default output format [json]: json
● sandeephedge@Sandeeps-MacBook-Pro-2023 Downloads % aws s3api create-bucket \
  --bucket sandeep-eks-terraform-three-tier \
  --region us-east-1
{
  "Location": "/sandeep-eks-terraform-three-tier",
  "BucketArn": "arn:aws:s3:::sandeep-eks-terraform-three-tier"
}
● sandeephedge@Sandeeps-MacBook-Pro-2023 Downloads % aws s3api put-bucket-versioning \
  --bucket sandeep-eks-terraform-three-tier \
  --versioning-configuration Status=Enabled
● sandeephedge@Sandeeps-MacBook-Pro-2023 Downloads % aws s3api put-bucket-encryption \
  --bucket sandeep-eks-terraform-three-tier \
  --server-side-encryption-configuration '{ \
    "Rules": [ \
      "ApplyServerSideEncryptionByDefault": { \
        "SSEAlgorithm": "AES256" \
      } \
    ] \
  }'
```

Lets validate the same in AWS Console by going to AWS and selecting S3



## Step 5 : Provision the files with Terraform

Since the Terraform files are sitting within the backend, we will now move to the directory where the Terraform files are sitting

```
ubuntu@ip-172-31-20-151:~/aws-eks-terraform-three-tier$ ll
total 40
drwxr-xr-x 7 ubuntu ubuntu 4096 Dec 21 07:54 .
drwxr-xr-x 1 ubuntu ubuntu 4096 Dec 21 07:54 .DS_Store
-rw-r--r-- 1 ubuntu ubuntu 1648 Dec 21 07:54 README.md
drwxr-xr-x 4 ubuntu ubuntu 4096 Dec 21 07:54 backend/
drwxr-xr-x 3 ubuntu ubuntu 4096 Dec 21 07:54 eks/
drwxr-xr-x 3 ubuntu ubuntu 4096 Dec 21 07:54 k8s_manifests/
drwxr-xr-x 2 ubuntu ubuntu 4096 Dec 21 07:54 terra-config/
ubuntu@ip-172-31-20-151:~/aws-eks-terraform-three-tier$ cd terra-config
ubuntu@ip-172-31-20-151:~/aws-eks-terraform-three-tier$ cat backend.tf
terraform
  backend "s3" {
    bucket      = "sandeepeksterraformthree-tier" # Change if the name already exists.
    key         = ".tfstate"
    region     = "us-east-1"
    encrypt    = true
  }
```

We will now initiate the Terraform backend

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG
ubuntu@ip-172-31-20-151:~/aws-eks-terraform-three-tier/terra-config$ terraform init
Initializing the backend...
Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Installing hashicorp/aws v5.100.0...
- Installed hashicorp/aws v5.100.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
ubuntu@ip-172-31-20-151:~/aws-eks-terraform-three-tier/terra-config$
```

We will now check to see if the terraform plan to validate the configuration, Now we're ready to provision the infrastructure! Run the following command:

```
1 terraform apply --auto-approve
```

## Understanding the Terraform Code Structure

### Option One : Broken down each file into a module to understand

I've broken the configuration into logical modules for clarity and scalability. Here's a quick overview:

- [provider.tf](#): Specifies the cloud provider. In our case, it's AWS (no surprise there!).
- [backend.tf](#): Configures Terraform to store state remotely in our S3 bucket.
- [ecr.tf](#): Creates two public repositories in ECR: 3-tier-frontend and 3-tier-backend for storing Docker images.
- [vpc.tf](#): Fetches the default VPC and subnet details.
- [role.tf](#): Defines IAM roles:

§ One for the EKS cluster (includes AmazonEKSClusterPolicy)

§ One for the Node Group (includes policies like AmazonEKSWorkerNodePolicy, AmazonEC2ContainerRegistryReadOnly, and AmazonEKS\_CNI\_Policy)

- [eks.tf](#): Provisions the EKS cluster named Three-tier-cloud.
- node\_group.tf: Creates the worker node group for the cluster with one t2.medium EC2 instance.

### Option Two: A Main Terraform File for us to ensure we can update everything easily

- [main.tf](#)
  - specifies all of the above in a single file which is just an amalgamation of all the TF files.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG
# aws_iam_role_policy_attachment.example-AWSWorkerNodePolicy will be created
+ resource "aws_iam_role_policy_attachment" "example-AWSWorkerNodePolicy" {
  + id          = (known after apply)
  + policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
  + role        = "eks-node-group-example"
}

# aws_iam_role_policy_attachment.example-AmazonEKS_CNI_Policy will be created
+ resource "aws_iam_role_policy_attachment" "example-AmazonEKS_CNI_Policy" {
  + id          = (known after apply)
  + policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
  + role        = "eks-node-group-example"
}

# aws_launch_template.eks_node_launch_template will be created
+ resource "aws_launch_template" "eks_node_launch_template" {
  + arn           = (known after apply)
  + default_version = (known after apply)
  + id            = (known after apply)
  + instance_type = "t2.medium"
  + latest_version = (known after apply)
  + name          = "Three-tier-cloud-node-template"
  + name_prefix   = (known after apply)
  + tags          = [
    + "Name" = "Three-tier-cloud-node-template"
  ]
  + tags_all     = [
    + "Name" = "Three-tier-cloud-node-template"
  ]

  + metadata_options {
    + http_endpoint      = "enabled"
    + http_use_private_ipv6 = (known after apply)
    + http_put_response_hop_limit = 2
    + http_tokens       = "required"
    + instance_metadata_tags = (known after apply)
  }
}

```

Plan: 11 to add, 0 to change, 0 to destroy.

We will now apply the Terraform Configuration

```
1 terraform apply --auto-approve
```

Once done it will show like this

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG
aws_eks_cluster.eks_cluster: Still creating... [06m30s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [06m40s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [06m50s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [07m00s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [07m10s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [07m20s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [07m30s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [07m40s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [07m50s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [08m00s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [08m10s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [08m20s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [08m30s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [08m40s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [08m50s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [09m00s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [09m10s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [09m20s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [09m30s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [09m40s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [09m50s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [10m00s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [10m10s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [10m20s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [10m30s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [10m40s elapsed]
aws_eks_cluster.eks_cluster: Still creating... [10m50s elapsed]
aws_eks_cluster.eks_cluster: Creation complete after 10m54s [id=three-tier-cloud-eks]
aws_launch_template.eks_node_launch_template: Creating...
aws_launch_template.eks_node_launch_template: Creation complete after 6s [id=lt-091be702f7998b1f0]
aws_eks_node_group.example: Creating...
aws_eks_node_group.example: Still creating... [00m10s elapsed]
aws_eks_node_group.example: Still creating... [00m20s elapsed]
aws_eks_node_group.example: Still creating... [00m30s elapsed]
aws_eks_node_group.example: Still creating... [00m40s elapsed]
aws_eks_node_group.example: Still creating... [00m50s elapsed]
aws_eks_node_group.example: Still creating... [01m00s elapsed]
aws_eks_node_group.example: Still creating... [01m10s elapsed]
aws_eks_node_group.example: Still creating... [01m20s elapsed]
aws_eks_node_group.example: Still creating... [01m30s elapsed]
aws_eks_node_group.example: Still creating... [01m40s elapsed]
aws_eks_node_group.example: Creation complete after 1m46s [id=three-tier-cloud-eks:eks-node-group]

Apply complete! Resources: 11 added, 0 changed, 0 destroyed.
ubuntu@ip-172-31-18-49:~/aws-eks-terraform-three-tier/Main-TF$
```

## Step 6: Pushing the images to ECR

Lets fo to ECR on AWS Console



We will need to get the push command for Docker – for this we need to click on **three-tier-backend** then click on **View push commands**

```
1 aws ecr-public get-login-password --region us-east-1 | docker login --
username AWS --password-stdin public.ecr.aws/p2s5y711
```

We will now need to get back to the terminal and navigate to the directory for backend and use this push command to push the Docker Image to ECR repository.

**Note 2:** Enable buildkit issue : If you get an error message which wont allow you to connect.

You can use the following commands

Add current user to docker group

```
1 sudo usermod -aG docker $USER
```

Apply group changes

```
1 newgrp docker
```

Verify

```
1 docker ps
```

Temporary workaround(not recommended long term)

```
1 sudo docker build -t three-tier-backend .
```

Best Option

```
1 DOCKER_BUILDKIT=1 docker build -t three-tier-backend .
```

We will now perform the same for the Frontend Functions as well

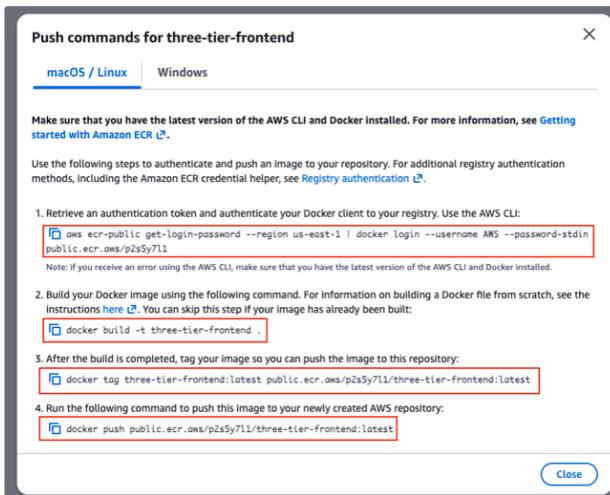
```
aws s3 sync ebs-terraform-three-tier ./awscli2.zip
ubuntu@ip-172-31-18-49:~$ cd aws-eks-terraform-three-tier
ubuntu@ip-172-31-18-49:~/aws-eks-terraform-three-tier$ cd backend
ubuntu@ip-172-31-18-49:~/aws-eks-terraform-three-tier/backend$ aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-stdin public.ecr.aws/p2s5y711
WARNING: Your credentials are stored unencrypted in '/home/ubuntu/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/
Login Succeeded
ubuntu@ip-172-31-18-49:~/aws-eks-terraform-three-tier/backend$ docker build -t three-tier-backend .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
Install the buildx component to build images with BuildKit:
https://docs.docker.com/go/buildx/
Sending build context to Docker daemon 40.96kB
Step 1/7 : FROM node:14-alpine
14-alpine: Pulling from library/node
f56be85fc22e: Pulling fs layer
bf665685b215: Pulling fs layer
w5fcfaefc395a6: Pulling fs layer
561cb6b9653d5: Pulling fs layer
bf665685b215: Waiting
w5fcfaefc395a6: Verifying Checksum
w5fcfaefc395a6: Download complete
f56be85fc22e: Verifying Checksum
f56be85fc22e: Download complete
561cb6b9653d5: Verifying Checksum
561cb6b9653d5: Download complete
bf665685b215: Verifying Checksum
bf665685b215: Download complete
f56be85fc22e: Pull complete
bf665685b215: Pull complete
w5fcfaefc395a6: Pull complete
561cb6b9653d5: Pull complete
Digest: sha256:+34215b487a329c9e867202ff89e704d3a75e554822e07f3e0c0f9e606121b33
Status: Downloaded newer image for node:14-alpine
--> 0dac3d27b1a
Step 2/7 : WORKDIR /usr/src/app
--> Running in 0255552b21c9
--> Removed intermediate container 0255552b21c9
--> c806a5eff51e3
Step 3/7 : COPY package*.json .
--> 46a8b6546a5c
Step 4/7 : RUN npm install
--> Running in 09bb94860db
npm WARN server@1.0.0 No description
npm WARN server@1.0.0 No repository field.

added 81 packages from 126 contributors and audited 81 packages in 2.772s

2 packages are looking for funding
  run 'npm fund' for details

found 18 vulnerabilities (5 low, 3 moderate, 7 high, 3 critical)
  run "npm audit fix" to fix them, or "npm audit" for details
--> Removed intermediate container 809bb94860db
--> 767e72291045
Step 5/7 : COPY .
--> 328b6cf7d7b3
Step 6/7 : EXPOSE 8080
--> Running in 40f351b9balc
--> Removed intermediate container 40f351b9balc
--> eea6115ebbd1
Step 7/7 : CMD [ "node", "index.js" ]
--> Running in 8ca0bc20b225
--> Removed intermediate container 8ca0bc20b225
--> f361adb219d1
Successfully built f361adb219d1
Successfully tagged three-tier-backend:latest
ubuntu@ip-172-31-18-49:~/aws-eks-terraform-three-tier/backend$ docker tag three-tier-backend:latest public.ecr.aws/p2s5y711/three-tier-backend:latest
ubuntu@ip-172-31-18-49:~/aws-eks-terraform-three-tier/backend$ docker push public.ecr.aws/p2s5y711/three-tier-backend:latest
The push refers to repository [public.ecr.aws/p2s5y711/three-tier-backend]
26954317a032: Pushed
35df64dd107e: Pushed
4700fe9bde0c: Pushed
d040441d5bb5: Pushed
31f710dc1782: Pushed
a599bf3e59b8: Pushed
e67e0085abae: Pushed
f1417c283b31: Pushed
latest: digest: sha256:affa7dde0400261081eba53221ace98850a658cdc7020401dda764ed0a619265 size: 1992
ubuntu@ip-172-31-18-49:~/aws-eks-terraform-three-tier/backend$ |
```

Navigating to ECR and selecting **three-tier-frontend** and then selecting **View push commands**



Once done you can validate the images by going to the ECR folders and viewing the images there

### three-tier-backend

Images (1)			Delete	Details	<a href="#">View in gallery</a>	<a href="#">View push commands</a>
Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	
latest	Image	22 December 2025, 21:13:58 (UTC+11)	47.28	Copy URI	sha256:affa7dde0400261...	

### three-tier-frontend

Images (1)			Delete	Details	<a href="#">View in gallery</a>	<a href="#">View push commands</a>
Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	
latest	Image	22 December 2025, 21:24:55 (UTC+11)	118.24	Copy URI	sha256:578e417dbefac...	

## Step 7 : Deploy to EKS with kubectl and Set Up Ingress via ALB

Now that your EKS cluster and ECR repositories are ready, it's time to interact with the cluster, deploy your workloads, and expose your application to the internet. We'll use **kubectl** for that — the command-line tool to manage Kubernetes clusters.

```
1 curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl
2 chmod +x ./kubectl
3 sudo mv ./kubectl /usr/local/bin
4 kubectl version --short --client
```

```
ubuntu@ip-172-31-18-49:~/aws-eks-terraform-three-tier$ cd
ubuntu@ip-172-31-18-49:~ curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin
kubectl version --short --client
* Total * Received * Xferd Average Speed Time Time Current
* Total * Received * Xferd Upload Total Spent Left Speed
100 57.4M 100 57.4M 0 0 7459K 0 0:00:07 0:00:07 ==:-=-- 8239K
Client Version: v1.19.6-eks-49a6c0
ubuntu@ip-172-31-18-49:~
```

We will now need to install kubectl on our machine and deploy the Kubernetes manifests

Before we can do this – we must edit the Deployment.YAML file for both backend and frontend.

To do this – we must go to the folder containing the YAML file

```
1 vim frontend-deployment.yaml
```

Image	Details	General information
	Image tag: latest <a href="#">URI</a> public.ecr.aws/p2s5y71/three-tier-frontend:latest Digest:  sha256:978e417dbefac761dfe12d80a0a6e7e76af0f79c36f341329e41d556a88	
	Artifact type: Image	Repository: three-tier-frontend
Size (MB) 118.24	image status	Pushed at: 22 December 2025, 21:24:55 (UTC+11)

You will see the image like this and we need to edit it with the unique characters highlighted above.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: workshop
  labels:
    role: frontend
    env: demo
spec:
  replicas: 1
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 25%
  selector:
    matchLabels:
      role: frontend
  template:
    metadata:
      labels:
        role: frontend
    spec:
      containers:
        - name: frontend
          image: public.ecr.aws/p2s5y7l1/three-tier-frontend:latest
          imagePullPolicy: Always
          env:
            - name: REACT_APP_BACKEND_URL
              value: "backend:8080" # assuming backend service is on port 8080
          ports:
            - containerPort: 3000
```

We will now do the same for backend as well

```
1 vim backend-deployment.yaml
```



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  namespace: workshop
  labels:
    role: backend
    env: demo
spec:
  replicas: 1
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 25%
  selector:
    matchLabels:
      role: backend
  template:
    metadata:
      labels:
        role: backend
    spec:
      containers:
        - name: backend
          image: public.ecr.aws/p2s5y7l1/three-tier-backend:latest
          imagePullPolicy: Always
          env:
            - name: MONGO_CONN_STR
              value: mongodb://mongodb-svc:27017/todo?directConnection=true
            - name: MONGO_USERNAME
              valueFrom:
                secretKeyRef:
                  name: mongo-sec
                  key: username
            - name: MONGO_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mongo-sec
                  key: password
          ports:
            - containerPort: 8080
          livenessProbe:
            httpGet:
              path: /ok
              port: 8080
              initialDelaySeconds: 2
              periodSeconds: 5
          readinessProbe:
            httpGet:
              path: /ok
              port: 8080
              initialDelaySeconds: 5
              periodSeconds: 5
              successThreshold: 1
```

## Connect kubectl to Your EKS Cluster

Now configure kubectl to use your EKS cluster:

Steps followed are :

*List clusters*

```
1 aws eks list-clusters
```

*Describe clusters*

```
1 aws eks describe-cluster --name three-tier-cloud-eks --region us-east-1
```

We will now install eksctl

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

```
ubuntu@ip-172-31-24-16:~/aws-eks-terraform-three-tier/k8s_manifests$ 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
eksctl version
0.221.0
```

We will now connect our ECS Clusters and connect kubeconfig

```
1 aws eks update-kubeconfig --region us-east-1 --name three-tier-cloud-eks
```

```
ubuntu@ip-172-31-24-16:~/aws-eks-terraform-three-tier/k8s_manifests$ aws eks update-kubeconfig --region us-east-1 --name three-tier-cloud-eks
Updated context arn:aws:eks:us-east-1:626635424094:cluster/three-tier-cloud-eks in /home/ubuntu/.kube/config
ubuntu@ip-172-31-24-16:~/aws-eks-terraform-three-tier/k8s_manifests$
```

Check the current context

```
1 kubectl config current-context
```

You should see

```
ubuntu@ip-172-31-24-16:~/aws-eks-terraform-three-tier/k8s_manifests$ kubectl config current-context
arn:aws:eks:us-east-1:626635424094:cluster/three-tier-cloud-eks
```

Now we will verify cluster nodes

```
1 kubectl get nodes
```

Tip : If kubectl get nodes returns an error message like the below

```
ubuntu@ip-172-31-24-16:~/aws-eks-terraform-three-tier/k8s_manifests$ kubectl get nodes
ubuntu@ip-172-31-24-16:~/aws-eks-terraform-three-tier/k8s_manifests$ aws eks describe-cluster --name three-tier-cloud-eks --region us-east-1
--query "cluster.endpoint"
"https://b726b0b0ba4c47144334b6e08f7f9ad4.gr7.us-east-1.eks.amazonaws.com"
```

And after navigating to the page you see this error message



If it returns a **public DNS**, check that your **security group allows port 443** from your IP

To do this – go to EKS→Networking→Add your IP or set to 0.0.0.0/0 to allow access from anywhere.

After fixing this – we should be able to see

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG

ubuntu@sandeep-ssh:~$ kubectl get nodes
NAME           STATUS   ROLES      AGE    VERSION
ip-172-31-43-190.ec2.internal   Ready    <none>   33m   v1.34.2-eks-ecaa3a6
```

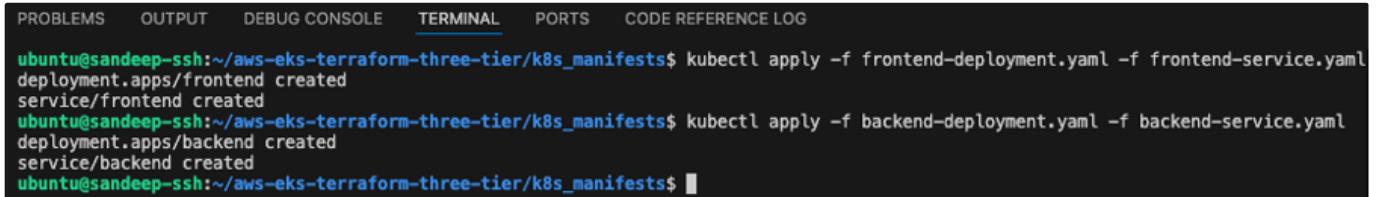
We will now create a namespace using the command

```
1 kubectl create namespace workshop
2 kubectl config set-context --current --namespace workshop
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$ kubectl create namespace workshop
ubuntu@ip-172-31-24-16:~/aws-eks-terraform-three-tier/k8s_manifests$ kubectl config set-context --current --namespace workshop
namespace/workshop created
Context "arn:aws:eks:us-east-1:626635424094:cluster/three-tier-cloud-eks" modified.
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$
```

We will now deploy the updated manifests we updated earlier

```
1 kubectl apply -f frontend-deployment.yaml -f frontend-service.yaml
2 kubectl apply -f backend-deployment.yaml -f backend-service.yaml
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$ kubectl apply -f frontend-deployment.yaml -f frontend-service.yaml
deployment.apps/frontend created
service/frontend created
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$ kubectl apply -f backend-deployment.yaml -f backend-service.yaml
deployment.apps/backend created
service/backend created
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$
```

We will now deploy MongoDB

```
1 cd mongo/
2 kubectl apply -f .
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$ cd mongo/
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$ kubectl apply -f .
deployment.apps/mongodb created
secret/mongo-sec created
service/mongodb-svc created
service/mongodb created
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$
```

We will now check pods

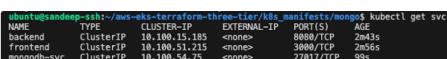
```
1 kubectl get pods
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$ cd mongo/
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$ kubectl get pods
deployment.apps/backend created
deployment.apps/frontend created
secret/mongo-sec created
secret/mongodb created
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$
```

Check services

```
1 kubectl get svc
```



NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
backend	ClusterIP	10.100.15.185	<none>	8080/TCP	2m43s
frontend	ClusterIP	10.100.51.215	<none>	30000/TCP	2m56s
mongodb-svc	ClusterIP	10.100.54.75	<none>	27017/TCP	99s

#### Set Up Application Load Balancer (ALB) and Ingress

To route external traffic into your Kubernetes services, we'll use an **AWS Application Load Balancer** along with an **Ingress Controller**.

The IAM policy json is present inside the kubernetes manifests dir:

#### *k8s\_manifests/*

Create the IAM policy in AWS:

```
1 aws iam create-policy \
2   --policy-name AWSLoadBalancerControllerIAMPolicy \
3   --policy-document file://iam_policy.json
```

We've already set up the OIDC provider earlier – we will now associate it

```
1 eksctl utils associate-iam-oidc-provider \
2   --region=us-east-1 \
3   --cluster=three-tier-cloud-eks \
4   --approve
```

We will now create a service account for the load balancer – I've added my account number so if you wish to run for your project you can change it

```
1 eksctl create iamserviceaccount \
2   --cluster=three-tier-cloud-eks \
3   --namespace=kube-system \
4   --name=aws-load-balancer-controller \
5   --role-name AmazonEKSLoadBalancerControllerRole \
6   --attach-policy-
    arn:arn:aws:iam:::policy/AWSLoadBalancerControllerIAMPolicy \
7   --approve \
8   --region=us-east-1
```

We will now install Helm and deploy the Load Balancer Controller

```
1 sudo snap install helm --classic
2
```

```

3 helm repo add eks https://aws.github.io/eks-charts
4 helm repo update eks
5
6 helm install aws-load-balancer-controller eks/aws-load-balancer-
controller \
7   -n kube-system \
8   --set clusterName=three-tier-cloud-eks \
9   --set serviceAccount.create=false \
10  --set serviceAccount.name=aws-load-balancer-controller

```

Lets check if its running

```

helm repo add eks https://aws.github.io/eks-charts
helm repo update eks
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
-n kube-system \
--set clusterName=three-tier-cloud-eks \
--set serviceAccount.create=false \
--set serviceAccount.name=aws-load-balancer-controller
helm 4.0.4 from Snapcrafters installed
eks/aws-load-balancer-controller
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "eks" chart repository
Update Complete! Happy Helm-ing!
NAME: aws-load-balancer-controller
LAST DEPLOYED: Thu Dec 25 22:20:37 2025
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
DESCRIPTION: Install complete
TEST SUITE: None
NOTES:
AWS Load Balancer controller installed!
ubuntu@sandeep-ssh:~$ 

```

```
1 kubectl get deployment -n kube-system aws-load-balancer-controller
```

**Now lets navigate to the k8s\_manifests folder and apply the ingress**

```
1 kubectl apply -f full_stack_lb.yaml
```

Wait for 5-7 minutes to allow the ingress and ALB to be fully provisioned

If not provisioned – it will show like this where there is no address showing up

```
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$ kubectl get ing -n workshop
NAME      CLASS   HOSTS   ADDRESS      PORTS   AGE
mainlb    alb     *        80          85s
```

**Access your application**

```
1 kubectl get ing -n workshop
```

```
ubuntu@sandeep-ssh:~$ kubectl get deployment -n kube-system aws-load-balancer-controller
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2           2           62s
```

**Tip:** ALB Address may take over 7-8 mins to show up

It was over 10 mins and the address didn't pop up so ran command to get the list of all subnets and it showed the below

```

1 aws ec2 describe-subnets \
2   --filters "Name=vpc-id,Values=$(aws ec2 describe-vpcs --filters \
3   Name=isDefault,Values=true --query 'Vpcs[0].VpcId' --output text)" \
4   --query 'Subnets[*].[SubnetId,AvailabilityZone]' \
5   --output table

```

```
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$ kubectl get ing -n workshop
NAME      CLASS   HOSTS   ADDRESS      PORTS   AGE
mainlb    alb     *        80          13m
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$ kubectl get pods -n kube-system | grep load-balancer
aws-load-balancer-controller-794f9ecdf-7fc5c   1/1   Running   0   17m
aws-load-balancer-controller-794f9ecdf-7fc5d   1/1   Running   0   17m
ubuntu@sandeep-ssh:~/aws-eks-terraform-three-tier/k8s_manifests$ aws ec2 describe-subnets \
`--filters "Name=vpc-id,Values=$(aws ec2 describe-vpcs --filters Name=isDefault,Values=true --query 'Vpcs[0].VpcId' --output text)" \
`--query 'Subnets[*].[SubnetId,AvailabilityZone]' \
`--output table
|  DescribeSubnets
+-----+
| subnet-91b2ca0b4b4331e0f   us-east-1a |
| subnet-904d4882b30571091   us-east-1a |
| subnet-99d7aa81986be9c4   us-east-1a |
| subnet-80e61117ae9d44d   us-east-1a |
| subnet-0726ac3f060593b6d  us-east-1a |
| subnet-24ccedf7fe029ad89  us-east-1c |

```

Ran the command to replace the public subnets to route the application

**Tip:** AWS Load Balancer Controller discovers subnets using Kubernetes tags — without them, ALBs cannot be provisioned.

```

1 aws ec2 create-tags \
2   --resources subnet-0b04d88b285b71091 subnet-01b42aa4b4331e02f \
3   --tags \
4     Key:kubernetes.io/cluster/three-tier-cloud-eks,Value=shared \
5     Key:kubernetes.io/role/elb,Value=1

```

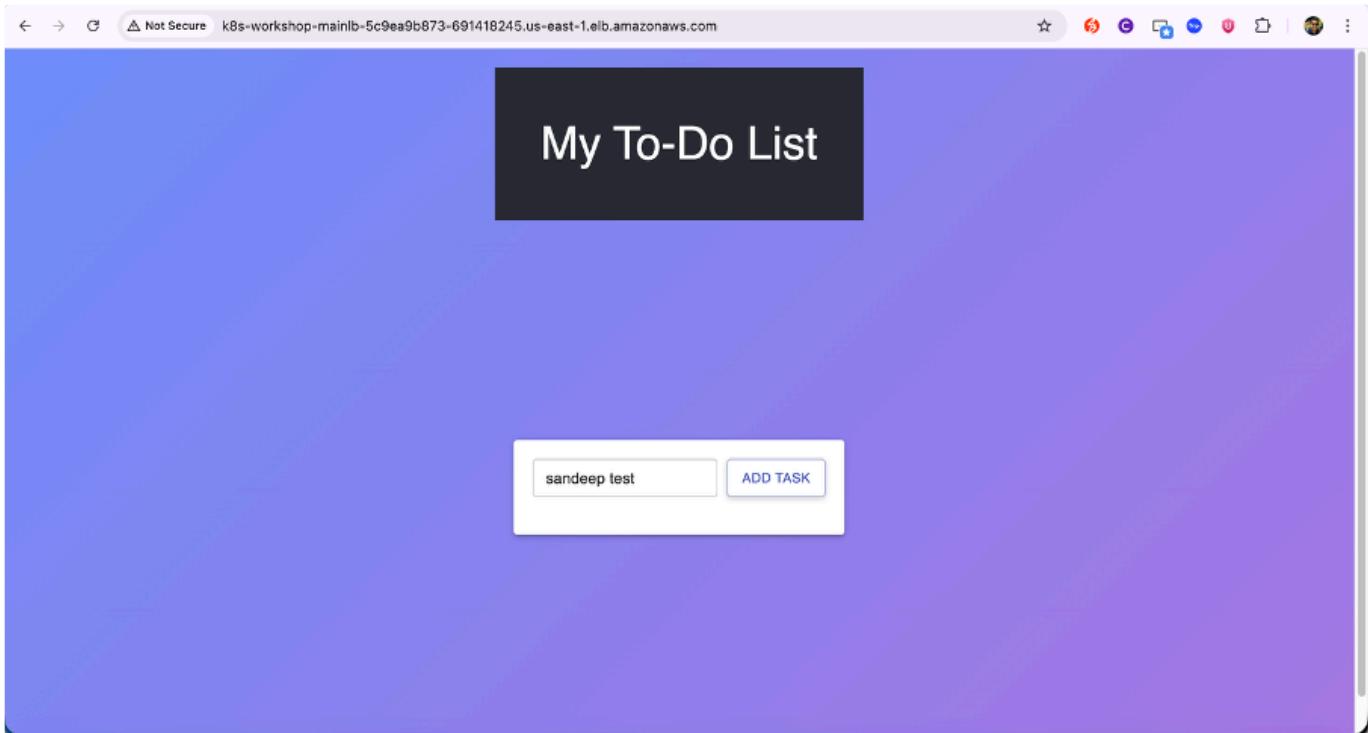
Re ran code

```

1 kubectl annotate serviceaccount aws-load-balancer-controller \
2   -n kube-system \
3   eks.amazonaws.com/role-
4   arn=arn:aws:iam::626635424094:role/AmazonEKSLoadBalancerControllerRole
5
6 helm repo add eks https://aws.github.io/eks-charts
7 helm repo update
8
9 helm install aws-load-balancer-controller eks/aws-load-balancer-
10 controller \
11   -n kube-system \
12   --set clusterName=three-tier-cloud-eks \
13   --set serviceAccount.create=false \
14   --set serviceAccount.name=aws-load-balancer-controller \
15   --set region=us-east-1

```

This resolved the issue.



## Screenshots from EKS

## Elastic Container Registry - AWS ECR

The screenshot shows the AWS ECR Public registry Repositories page. It lists two public repositories: "three-tier-backend" and "three-tier-frontend". Both were created on December 29, 2025, at 21:29:02 UTC+11. The "three-tier-backend" repository has a size of 116.24 MB and the "three-tier-frontend" repository has a size of 47.28 MB.

Repository name	URI	Created at
three-tier-backend	public.ecr.aws/p2s5y71/three-tier-backend	29 December 2025, 21:29:02 (UTC+11)
three-tier-frontend	public.ecr.aws/p2s5y71/three-tier-frontend	29 December 2025, 21:29:02 (UTC+11)

## ECR Frontend

The screenshot shows the AWS ECR Public registry Repository details page for the "three-tier-backend" repository. It displays one image named "latest" with a size of 116.24 MB and a push date of December 29, 2025, at 21:29:02 UTC+11. The image URL is listed as "shard0001.00000000000000000000...".

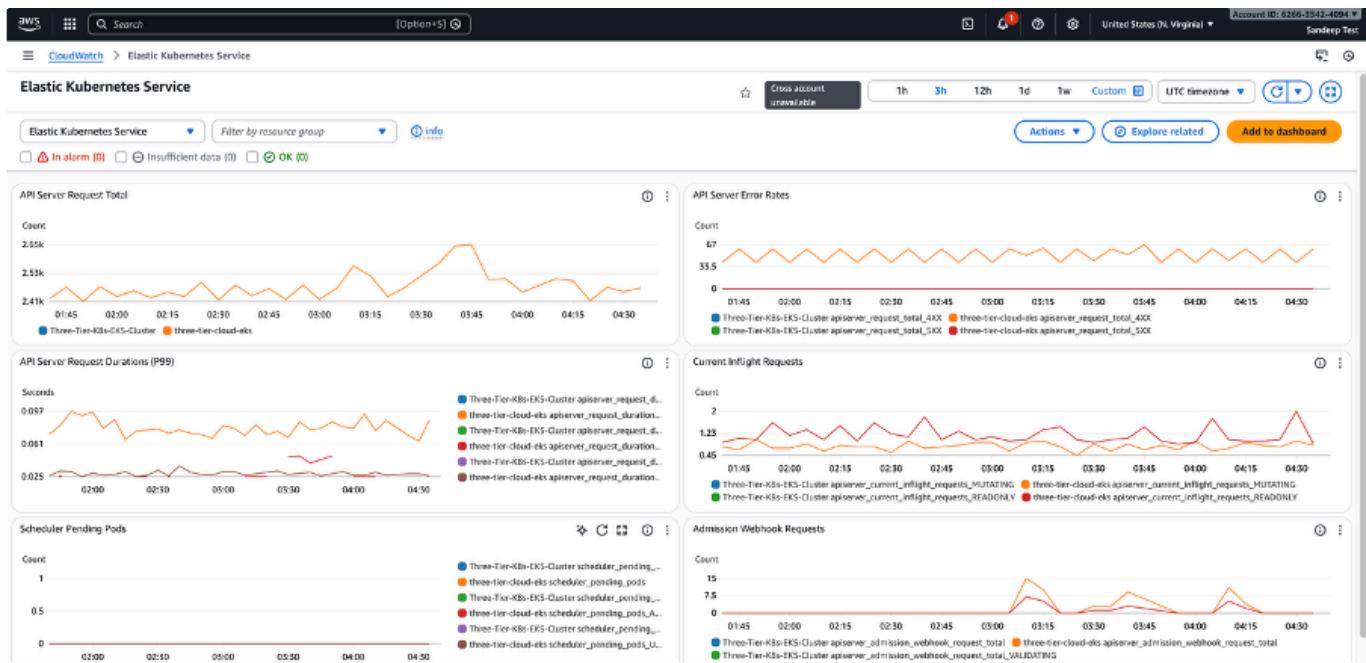
## ECR Backend

The screenshot shows the AWS ECR Public registry Repository details page for the "three-tier-frontend" repository. It displays one image named "latest" with a size of 47.28 MB and a push date of December 29, 2025, at 21:29:49 UTC+11. The image URL is listed as "shard0001.00000000000000000000...".

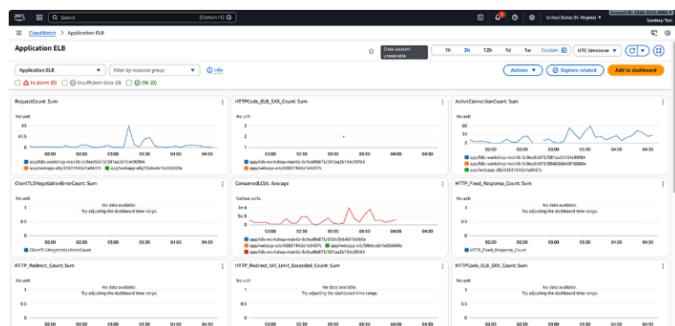
## Dynamo DB

The screenshot shows the AWS DynamoDB Explore Items page for the "terraform-locks" table. The table contains 3 items. A scan operation is being run on the "terraform-locks" table, which has completed with 1 item returned. The table has attributes: LockID (String) and Digest. One item is visible with LockID: "sanderp-eks-terrafor..." and Digest: "0f0fnf2c579c8005047524bcff225fc".

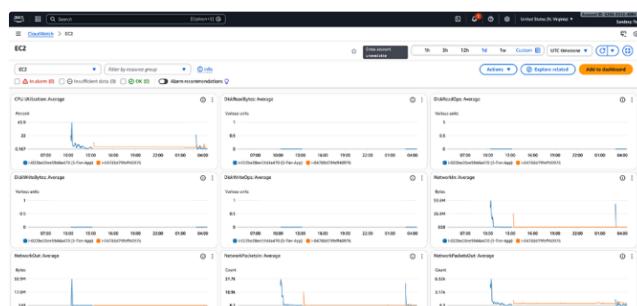
## Cloudwatch-EKS



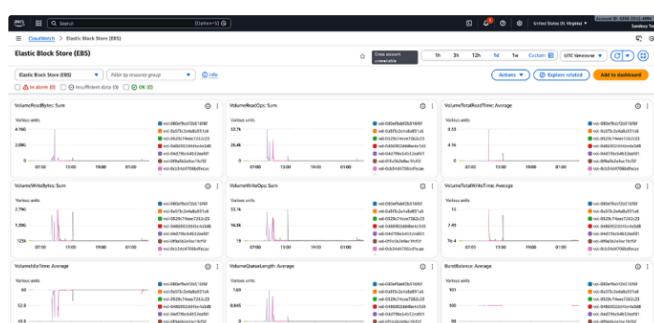
## Cloudwatch -ELB



## Cloudwatch-EC2



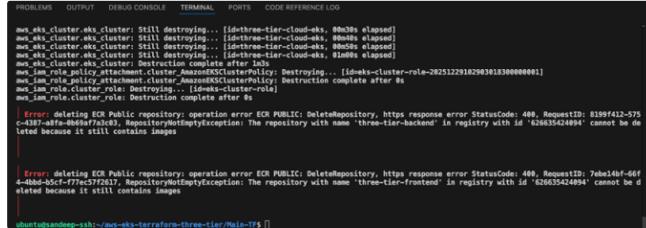
## Cloudwatch - EBS



## Step 10: Cleaning up AWS Resources

To avoid any un-necessary costs – we now need to go ahead and delete all our resources. Since we used Terraform – we can destroy/delete all resources with a touch of a single command.

```
1 terraform destroy -auto-approve
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG
aws_eks_cluster.eks_cluster: Still destroying... [id=three-tier-cloud-eks, 0m03s elapsed]
aws_eks_cluster.eks_cluster: Still destroying... [id=three-tier-cloud-eks, 0m05s elapsed]
aws_eks_cluster.eks_cluster: Still destroying... [id=three-tier-cloud-eks, 0m06s elapsed]
aws_eks_cluster.eks_cluster: Still destroying... [id=three-tier-cloud-eks, 0m08s elapsed]
aws_iam_role_policy_attachment.cluster_AmazonEKSClusterPolicy: Destroying... [id=eks-cluster-role-28251229182963818308000001]
aws_iam_role_policy_attachment.cluster_AmazonEKSClusterPolicy: Destruction complete after 6s
aws_iam_role.cluster_role: Destroying... [id=eks-cluster-role-28251229182963818308000001]
aws_iam_role.cluster_role: Destruction complete after 6s

[Error: deleting ECR Public repository: operation error ECR PUBLIC: DeleteRepository, https response error StatusCode: 400, RequestID: 8199f412-575c-4307-8ff8-0b09a7a2c8] RepositoryNotEmptyException: The repository with name 'three-tier-backend' in registry with id '626635424894' cannot be deleted because it still contains images

[Error: deleting ECR Public repository: operation error ECR PUBLIC: DeleteRepository, https response error StatusCode: 400, RequestID: 7eb14bf-66f4-4bbd-b5cf-f77fec57f2e37] RepositoryNotEmptyException: The repository with name 'three-tier-frontend' in registry with id '626635424894' cannot be deleted because it still contains images

ubuntu@andreas-OptiPlex-5090:~$ terraform destroy -auto-approve
```

We will need to go to ECR and delete the images from the AWS Console manually

Problem #	Issue Faced	Steps Taken to Resolve	Final Resolution
1	Backend targets unhealthy	Fixed health check path to /backend	Backend target became healthy
2	Frontend ImagePullBackOff	Pulled latest image, updated manifest	Frontend pod running successfully
3	Deployment selector immutable	Deleted old pods to allow new rollout	Deployment updated successfully
4	Ingress path routing	Configured ALB annotation actions.backend-rewrite	Path-based routing works
5	Port conflicts on local dev	Stopped local service running on 8080	Backend accessible via cluster service

## Future Improvements

- Enable **HTTPS** with ACM certificates on ALB.
- Implement **CI/CD pipeline** for automated EKS deployments.
- Add **Terraform modules** for reusability across projects.