**Technical Implementation Design**

# 1. Terraform Environment Setup

Terraform Bootstrap EC2 Instance

Purpose: To run and manage Terraform code

* Instance Type: t3.medium (2 vCPU, 4GB RAM)
* AMI: ubuntu
* Storage: 50 GB gp3 EBS
* Security Group: SSH (restricted IP), HTTPS outbound

Terraform Project Structure

terraform/  
├── modules/  
│ ├── eks.tf/  
│ ├── vpc.tf/  
│ ├── rds.tf/  
│ ├── alb.tf/

| |\_\_\_\_ecr.tf

| |\_\_\_\_securitygroup.tf  
│ └── s3.tf/

├── locals.tf  
├── output.tf  
├── provider.tf  
├── terraform.tfvars  
└── variables.tf  
├── envs/  
│ ├── dev/  
│ └── prod/  
└── backend/  
 └── main.tf (S3 backend config)

State Management

S3 Bucket: cmm-terraform-backend

**Install and Configure terraform on Bastion-Host-Server:**

Create ubuntu t3.medium EC2 machine for Install & configure terraform

**Install Terraform**:

sudo apt-get update && sudo apt-get install -y gnupg software-properties-common

**Install the HashiCorp**[**GPG key**](https://apt.releases.hashicorp.com/gpg)**:**

wget -O- https://apt.releases.hashicorp.com/gpg | \

gpg --dearmor | \

sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null

**Verify the key's fingerprint:**

gpg --no-default-keyring \

--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \

--fingerprint

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \

https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | \

sudo tee /etc/apt/sources.list.d/hashicorp.list

**Install Terraform from the new repositor:**

sudo apt update

sudo apt-get install terraform

**Verify the installation**

terraform -help

**Install and Configure AWS CLI on Bastion-Host-Server**

1. Create IAM User
2. Create Access Key and Secret Key for IAM User
3. Install AWS CLI

# Install eksctl, kubectl, and Helm on Bastion-Host-Server

# 2. VPC & Networking Setup

**VPC**

CIDR Block: 10.242.0.0/16

**Subnets**

2x Public Subnets: 10.242.1.0/24, 10.242.2.0/24

2x Private Subnets: 10.242.3.0/24, 10.242.4.0/24

**Gateways**

Internet Gateway for public access

NAT Gateway for private subnet outbound access

**Route Tables**

Public and Private routing logic associated with subnets

# 3. EKS Cluster Setup

**EKS Cluster**

- Version: 1.32

- Endpoint Access: Private

- Logging: API, Scheduler, Controller Manager

**Node Groups (3)**

* **Frontend App Nodes – (public)**  
   Instance Type: t3.large (2 vCPU, 8GB RAM)  
   Desired: 1 | Min: 1 | Max: 2  
   Scaling: Enabled (Cluster Autoscaler)  
   AMI: Amazon EKS Optimized Linux
* **Backend App Nodes (private)**  
   Instance Type: m4.xlarge (4 vCPU, 16GB RAM)  
   Desired: 1 | Min: 1 | Max: 5  
   Scaling: Enabled (Cluster Autoscaler)  
   AMI: Amazon EKS Optimized Linux
* **Inference Node Group (GPU) (private)**  
   Instance Type: g4dn.xlarge (4 vCPU,16GB RAM)  
   Desired: 0 | Min: 0 | Max: 100

**IAM Role for EKS: : (AWS managed Roles)**

# AmazonEKSClusterPolicy

**IAM Role for Node Group: (AWS managed Roles)**

* AmazonEC2ContainerRegistryReadOnly
* AmazonEKS\_CNI\_Policy
* AmazonEKSWorkerNodePolicy

**EKS Add-ons**

* Amazon VPC CNI
* coreDNS
* kube-proxy
* metrics-server
* cluster-autoscaler
* Amazon pod identity EKS agent
* Node Monitoring Agent

**EKS Security Group**

| **Rule type** | **Protocol** | **Port** | **Destination** |
| --- | --- | --- | --- |
| **Inbound** | **All** | **All** | **self** |
| **Outbound** | **TCP** | **443** | **Cluster security group** |
| **Outbound** | **TCP** | **10250** | **Cluster security group** |
| **Outbound (DNS)** | **TCP and UDP** | **53** | **Cluster security group** |

# 4. ALB Setup

**Load Balancer**

ALB with Ingress Controller (via Helm)

Ingress annotations per service

# 5. Aurora Serverless PostgreSQL

**Aurora Setup**

* Engine: Aurora Serverless v2
* DB: PostgreSQL
* ACU: Min 0.5, Max 4
* Subnet Group: Private subnets only
* IAM Authentication: Enabled
* Security
* DB Credentials: Stored in AWS Secrets Manager
* SG allows traffic only from EKS nodes
* Create RDS subnet group
* Create Aurora Serverless cluster
* Create security groups and access roles
* Store DB credentials in Secrets Manager

h

# 6. S3 Bucket Design (3)

**S3 Buckets name**

1. cmm-upload-bucket – put – 60 days
2. cmm-download-bucket – get - 60 days
3. cmm-app-data-bucket – no presen url

* Create private buckets: upload, download, app data
* Define IAM policies for scoped access
* Enable encryption and lifecycle policies

**Security**

Encryption: SSE-S3 or SSE-KMS

IAM Role-Based Access (IRSA)

Lifecycle Policies: archive after 60 days & delete after 90

# 7. AWS Secrets Manager

* Stores: IDM Auth credentials, DB credentials
* Access Policy: Attached to EKS service roles via IRSA
* Automatic rotation support (optional)

# 8. IAM & Security

OIDC Provider

Linked with EKS for IRSA

Roles

* EKS Node IAM Role
* App Pod Role (with IRSA)
* Terraform Role (Admin)
* DB Access Role
* S3 Read/Write Role

**Policies**

Fine-grained access to S3, Secrets Manager, Aurora

# 9. Post-Provisioning Actions

**Output**

VPC ID, Subnet IDs, Cluster ARN, S3 Bucket Names, DB Endpoint

**Validation**

* Pod connectivity to DB/S3
* Secret fetch from Secrets Manager
* ALB routing for all services