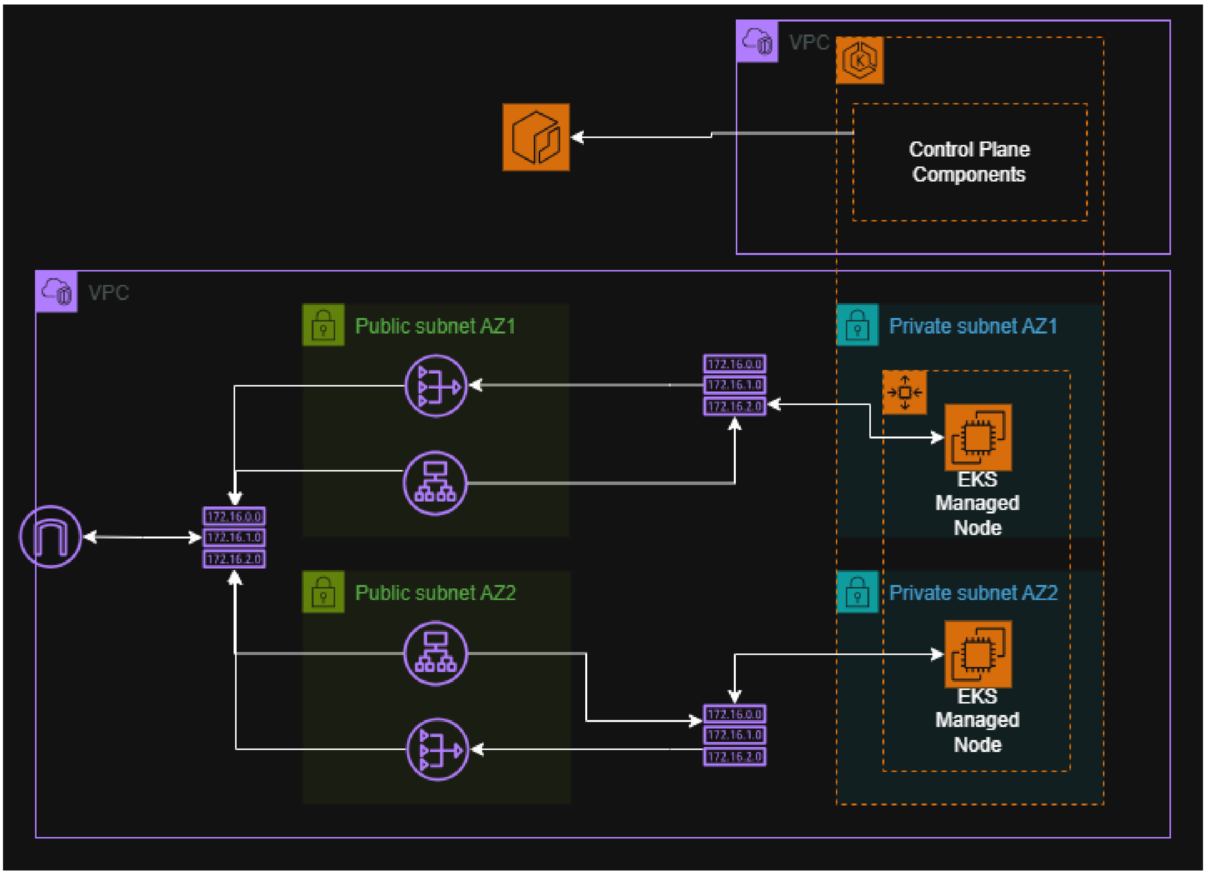
Deploying 3-Tier NodeJS Application on AWS EKS with Application Load Balancer (Task 17)

Zaeem Attique Ashar
Cloud Intern

Task Description:

This project involves deploying a 3-tier NodeJS application on AWS Elastic Kubernetes Service (EKS). The infrastructure will be provisioned using Terraform and will include a VPC with public and private subnets across two availability zones, an EKS cluster with managed node groups, and an Application Load Balancer (ALB) for ingress traffic management. The application consists of three tiers: a web frontend, an application backend, and a MySQL database with persistent storage. The AWS Load Balancer Controller will be installed using Helm to automatically provision and manage the ALB based on Kubernetes Ingress resources.

Architecture Diagram:



Task 17.1: Create Networking Infrastructure with Terraform

Terraform Module Structure:

The networking infrastructure is created using the `modules/networking` Terraform module.

VPC Configuration:

- **VPC CIDR Block:** 10.0.0.0/16
- **DNS Support:** Enabled
- **DNS Hostnames:** Enabled

Subnet Configuration:

Public Subnets:

- Public Subnet A (us-east-1a): 10.0.101.0/24
- Public Subnet B (us-east-1b): 10.0.102.0/24
- Auto-assign Public IP: Enabled
- Tagged for ELB: `kubernetes.io/role/elb = 1`

Private Subnets:

- Private Subnet A (us-east-1a): 10.0.1.0/24
- Private Subnet B (us-east-1b): 10.0.2.0/24
- Tagged for Internal ELB: `kubernetes.io/role/internal-elb = 1`

All subnets are tagged with: `kubernetes.io/cluster/Task17-EKS-Cluster-Zaeem = shared`

Internet Gateway:

- Attached to VPC for public subnet internet access

NAT Gateway:

- Deployed in Public Subnet A
- Elastic IP allocated for NAT Gateway

- Provides internet access for private subnets

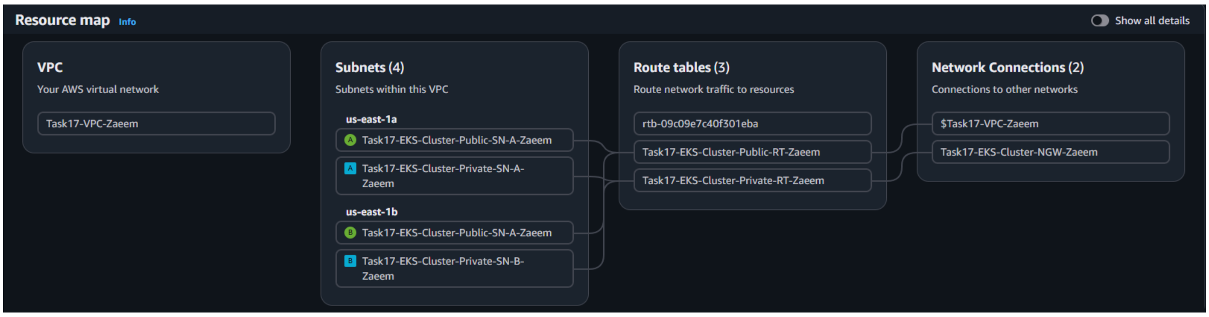
Route Tables:

Public Route Table:

- Route: 0.0.0.0/0 → Internet Gateway
- Associated with both public subnets

Private Route Table:

- Route: 0.0.0.0/0 → NAT Gateway
- Associated with both private subnets


Resource map Info
Show all details
VPC
Your AWS virtual network
Task17-VPC-Zaeem
Subnets (4)
Subnets within this VPC
us-east-1a
A Task17-EKS-Cluster-Public-SN-A-Zaeem
A Task17-EKS-Cluster-Private-SN-A-Zaeem
us-east-1b
B Task17-EKS-Cluster-Public-SN-A-Zaeem
B Task17-EKS-Cluster-Private-SN-B-Zaeem
Route tables (3)
Route network traffic to resources
rtb-09c09e7c40f301eba
Task17-EKS-Cluster-Public-RT-Zaeem
Task17-EKS-Cluster-Private-RT-Zaeem
Network Connections (2)
Connections to other networks
$Task17-VPC-Zaeem
Task17-EKS-Cluster-NGW-Zaeem

Task 17.2: Create IAM Roles for EKS

EKS Cluster Role:

Created using the `modules/iam` Terraform module.

Role Name: Task17-EKS-Cluster-Role-Zaeem

Trust Policy:

```
"Version": "2012-10-17",
"Statement": [{
  "Effect": "Allow",
  "Principal": { "Service": "eks.amazonaws.com" },
  "Action": "sts:AssumeRole"
```

Attached Policy:

- `arn:aws:iam::aws:policy/AmazonEKSClusterPolicy`

EKS Node Role:

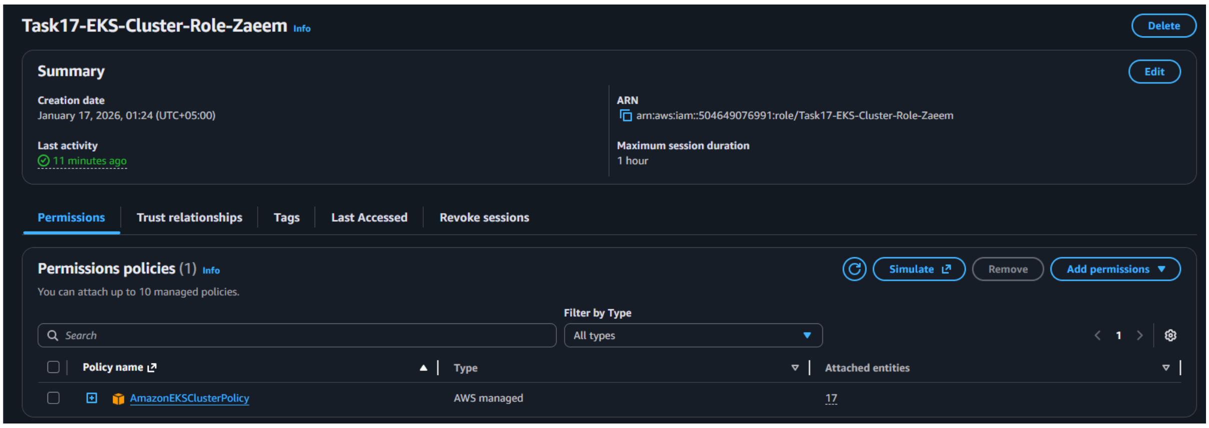
Role Name: Task17-EKS-Node-Role-Zaeem

Trust Policy:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": { "Service": "ec2.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }]
}
```

Attached Policies:

- `arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy`
- `arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy`
- `arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly`



Task17-EKS-Cluster-Role-Zaeem

Summary

Creation date
January 17, 2026, 01:24 (UTC+05:00)

Last activity
11 minutes ago

ARN
arn:aws:iam::504649076991:role/Task17-EKS-Cluster-Role-Zaeem

Maximum session duration
1 hour

Permissions policies (1)

You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type	Attached entities
AmazonEKSClusterPolicy	AWS managed	17

Task 17.3: Create EKS Cluster and Node Group

EKS Cluster Configuration:

Created using the `modules/eks` Terraform module.

Cluster Name: Task17-EKS-Cluster-Zaeem
Kubernetes Version: Latest (managed by AWS)
Role ARN: Task17-EKS-Cluster-Role-Zaeem

VPC Configuration:

- All four subnets (2 public + 2 private) attached to cluster

EKS Node Group Configuration:

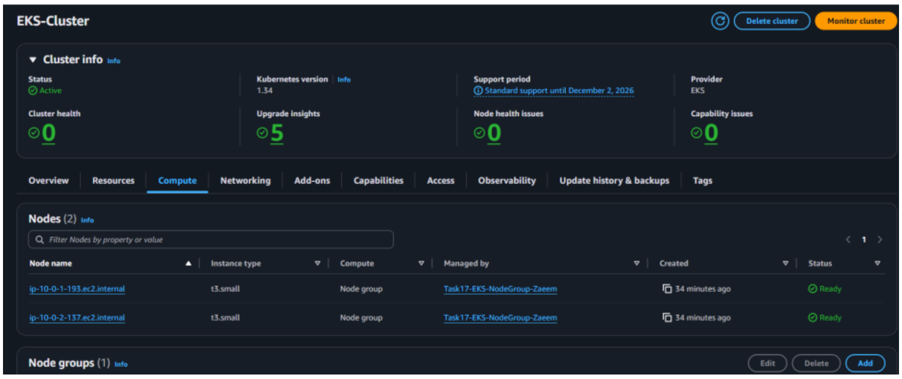
Node Group Name: Task17-EKS-NodeGroup-Zaeem
Node Role ARN: Task17-EKS-Node-Role-Zaeem
Subnets: Both private subnets (us-east-1a, us-east-1b)

Scaling Configuration:

- Desired Size: 2
- Maximum Size: 3
- Minimum Size: 1

Instance Configuration:

- Instance Type: t3.small
- Disk Size: 20 GB
- AMI Type: AL2_x86_64 (Amazon Linux 2)



Status: Active
Kubernetes version: 1.34
Support period: Standard support until December 2, 2026
Provider: EKS
Cluster health: 0
Upgrade insights: 5
Node health issues: 0
Capability issues: 0

Nodes (2)

Node name	Instance type	Compute	Managed by	Created	Status
ip-10-0-1-193.ec2.internal	t3.small	Node group	Task17-EKS-NodeGroup-Zaeem	34 minutes ago	Ready
ip-10-0-2-137.ec2.internal	t3.small	Node group	Task17-EKS-NodeGroup-Zaeem	34 minutes ago	Ready

Node groups (1)

Task 17.4: Create OIDC Provider for EKS

The OIDC (OpenID Connect) provider enables IAM roles for service accounts (IRSA), allowing Kubernetes service accounts to assume IAM roles.

OIDC Provider Configuration:

Created using the `modules/eks` Terraform module.

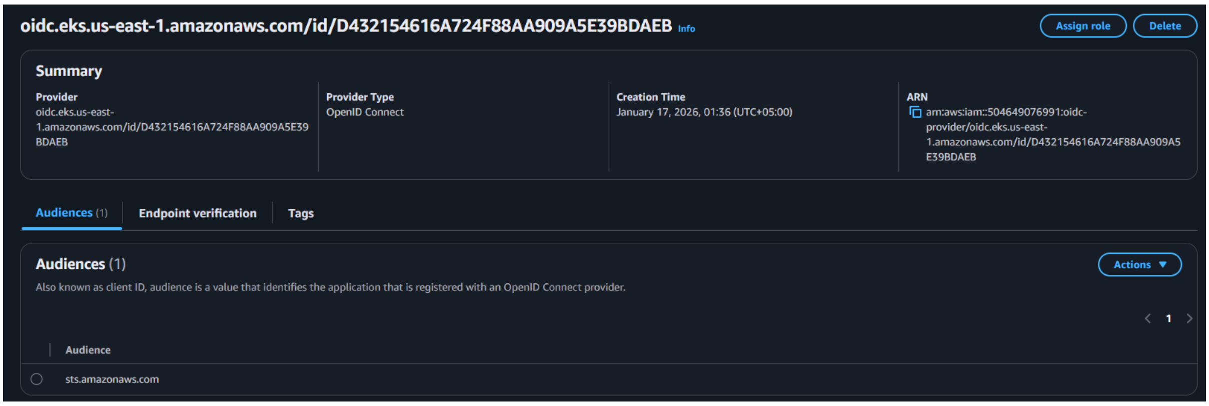
Data Source: TLS Certificate from EKS cluster OIDC issuer URL

Provider Configuration:

- URL: EKS Cluster OIDC Issuer URL
- Client ID List: `["sts.amazonaws.com"]`
- Thumbprint List: SHA1 fingerprint from TLS certificate

Dependencies:

- Waits for cluster to be active using `null_resource`
- TLS certificate data source depends on cluster creation



oidc.eks.us-east-1.amazonaws.com/id/D432154616A724F88AA909A5E39BDAEB

Summary

Provider
oidc.eks.us-east-1.amazonaws.com/id/D432154616A724F88AA909A5E39BDAEB

Provider Type
OpenID Connect

Creation Time
January 17, 2026, 01:36 (UTC+05:00)

ARN
arn:aws:iam::504649076991:oidc-provider/oidc.eks.us-east-1.amazonaws.com/id/D432154616A724F88AA909A5E39BDAEB

Audiences (1)

Also known as client ID, audience is a value that identifies the application that is registered with an OpenID Connect provider.

	Audience
○	sts.amazonaws.com

Task 17.5: Create IAM Role for ALB Controller

ALB Controller IAM Role:

Created using the `modules/iam_alb` Terraform module.

Role Name: Task17-ALB-Controller-Role-Zaeem

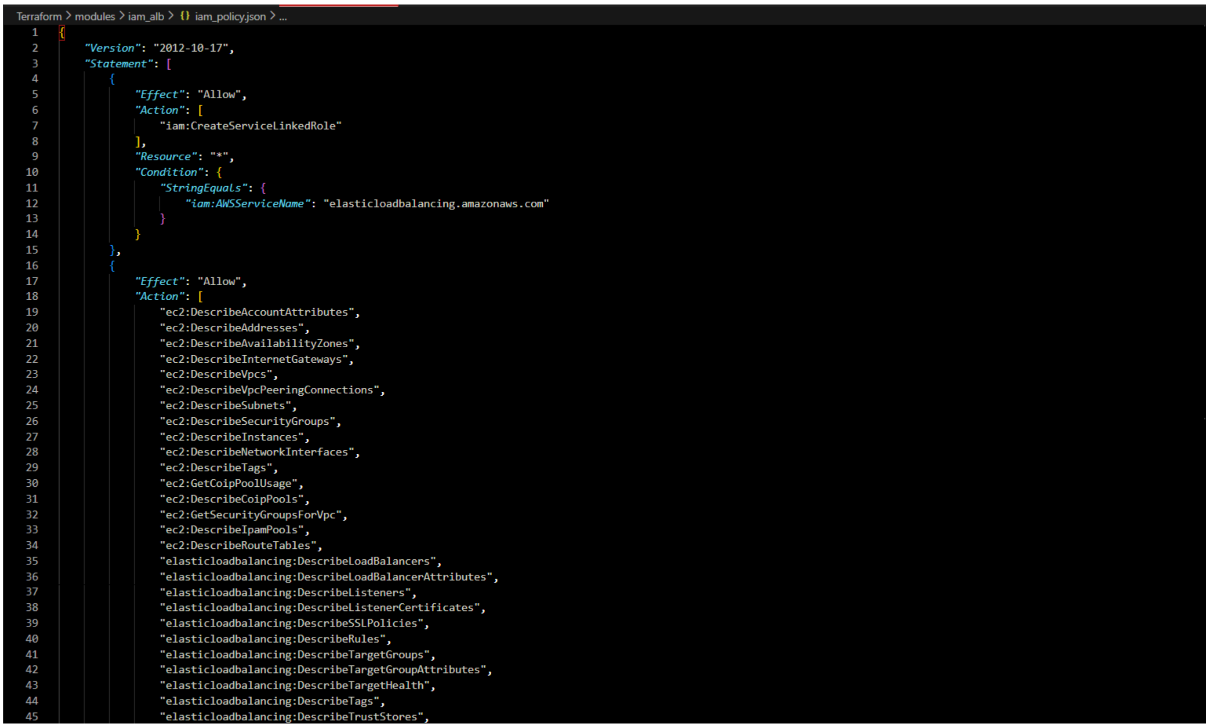
Trust Policy:

```
    {
      "Version": "2012-10-17",
      "Statement": [{
        "Effect": "Allow",
        "Principal": {
          "Federated": "<OIDC_PROVIDER_ARN>"
        },
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {
          "StringEquals": {
            "<OIDC_PROVIDER>:sub": "system:serviceaccount:kube-
system:aws-load-balancer-controller",
            "<OIDC_PROVIDER>:aud": "sts.amazonaws.com"
          }
        }
      }]
    }
```

IAM Policy:
The ALB Controller requires permissions for:

- Creating/managing Application Load Balancers
- Managing target groups
- Creating/managing security groups
- Managing EC2 resources (subnets, VPCs, ENIs)
- Managing WAF and Shield resources

Policy Name: Task17-ALB-Controller-Policy-Zaeem
Policy Document: `iam_policy.json` (AWS Load Balancer Controller IAM policy)



```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "elasticloadbalancing.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeVpcs",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeInstances",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeTags",
                "ec2:GetCoipPoolUsage",
                "ec2:DescribeCoipPools",
                "ec2:GetSecurityGroupsForVpc",
                "ec2:DescribeIpamPools",
                "ec2:DescribeRouteTables",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeLoadBalancerAttributes",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeListenerCertificates",
                "elasticloadbalancing:DescribeSSLPolicies",
                "elasticloadbalancing:DescribeRules",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeTargetGroupAttributes",
                "elasticloadbalancing:DescribeTargetHealth",
                "elasticloadbalancing:DescribeTags",
                "elasticloadbalancing:DescribeTrustStores",
```

Task 17.6: Configure kubectl

Configure kubectl:

```
$CLUSTER_NAME = terraform output -raw cluster_name
aws eks update-kubeconfig --name $CLUSTER_NAME --region us-east-1
kubectl get nodes
```

Task 17.7: Install AWS Load Balancer Controller using Helm

Helm Installation:

Step 1: Add EKS Chart Repository

```
    helm repo add eks https://aws.github.io/eks-charts
    helm repo update
```

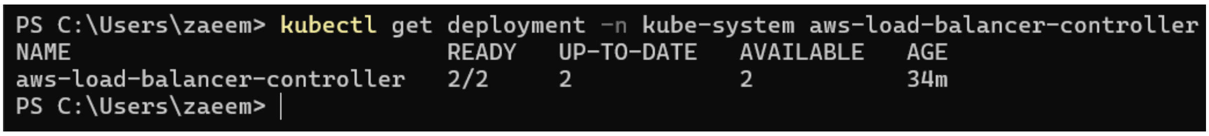
Step 2: Install AWS Load Balancer Controller

```
    $CLUSTER_NAME = terraform output -raw cluster_name
    $ALB_ROLE_ARN = terraform output -raw alb_controller_iam_role_arn
    $VPC_ID = terraform output -raw vpc_id

    helm install aws-load-balancer-controller eks/aws-load-balancer-
controller `
      -n kube-system `
      --set clusterName=$CLUSTER_NAME `
      --set serviceAccount.create=true `
      --set serviceAccount.name=aws-load-balancer-controller `
      --set "serviceAccount.annotations.eks\.amazonaws\.com/role-
arn=$ALB_ROLE_ARN" `
      --set region=us-east-1 `
      --set vpcId=$VPC_ID
```

Verification:

```
    kubectl wait --for=condition=Ready pod -l
app.kubernetes.io/name=aws-load-balancer-controller -n kube-system --
timeout=120s
    kubectl get deployment -n kube-system aws-load-balancer-controller
```



```
PS C:\Users\zaeem> 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           34m
PS C:\Users\zaeem> |
```

Task 17.8: Deploy Application Components

Storage Class Configuration (storageclass.yaml):

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gp3
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.aws.com
parameters:
  type: gp3
  encrypted: "true"
  fsType: ext4
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
```

Key Features:

- Uses EBS CSI driver (`ebs.csi.aws.com`)
- GP3 volume type (better performance than GP2)
- Encryption enabled
- WaitForFirstConsumer binding mode
- Volume expansion allowed

Database Tier - MySQL (mysql.yaml):

Secret Configuration:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
type: Opaque
stringData:
  MYSQL_ROOT_PASSWORD: rootpass
  MYSQL_DATABASE: appdb
  MYSQL_USER: appuser
  MYSQL_PASSWORD: apppass
```

StatefulSet Configuration:

- **Name:** mysql
- **Replicas:** 1
- **Service Name:** mysql (headless service)
- **Container Image:** mysql:8.0

- **Port:** 3306
- **Volume Mount:** /var/lib/mysql
- **Storage:** 5Gi EBS volume (dynamically provisioned)

Resource Limits:

- Memory: 512Mi (request), 1Gi (limit)
- CPU: 250m (request), 500m (limit)

Health Checks:

- Liveness Probe: mysqladmin ping
- Readiness Probe: MySQL connection test

Service Configuration:

- Type: ClusterIP (Headless)
- Port: 3306
- Selector: app=mysql

Application Tier - Backend (backend.yaml):

Deployment Configuration:

- **Name:** app-backend
- **Replicas:** 2
- **Container Image:** 504649076991.dkr.ecr.us-east-1.amazonaws.com/task17/zaeem:app-tier
- **Port:** 4000

Environment Variables:

- DB_HOST: mysql.default.svc.cluster.local
- DB_USER: (from mysql-secret)
- DB_PWD: (from mysql-secret)
- DB_DATABASE: (from mysql-secret)

Resource Limits:

- Memory: 256Mi (request), 512Mi (limit)
- CPU: 200m (request), 500m (limit)

Health Checks:

- Liveness Probe: HTTP GET /health on port 4000
- Readiness Probe: HTTP GET /health on port 4000

Service Configuration:

- **Name:** backend-service
- Type: ClusterIP
- Port: 4000
- Selector: app=backend

Web Tier - Frontend (frontend.yaml):

Deployment Configuration:

- **Name:** web-frontend
- **Replicas:** 2
- **Container Image:** 504649076991.dkr.ecr.us-east-1.amazonaws.com/task17/zaeem:web-tier
- **Port:** 80

Resource Limits:

- Memory: 128Mi (request), 256Mi (limit)
- CPU: 100m (request), 300m (limit)

Health Checks:

- Liveness Probe: HTTP GET / on port 80
- Readiness Probe: HTTP GET / on port 80

Service Configuration:

- **Name:** frontend-service
- Type: NodePort
- Port: 80
- NodePort: 30000
- Selector: app=frontend

Ingress Configuration:

- **Name:** frontend-ingress
- **IngressClassName:** alb

ALB Annotations:

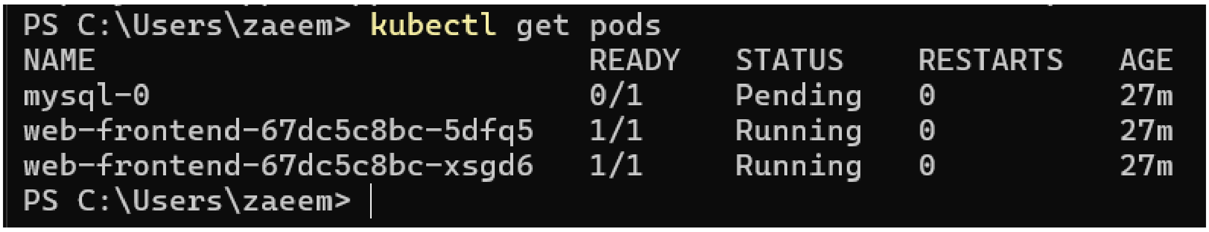
- `alb.ingress.kubernetes.io/scheme: internet-facing`
- `alb.ingress.kubernetes.io/target-type: instance`
- `alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}]'`
- `alb.ingress.kubernetes.io/healthcheck-path: /`
- `alb.ingress.kubernetes.io/healthcheck-protocol: HTTP`
- `alb.ingress.kubernetes.io/success-codes: "200"`
- `alb.ingress.kubernetes.io/load-balancer-name: task17-frontend-alb`

Ingress Rules:

- Path: /
- Path Type: Prefix
- Backend Service: frontend-service (port 80)

Deployment Commands:

```
kubectl apply -f storageclass.yaml
kubectl apply -f mysql.yaml
kubectl apply -f backend.yaml
kubectl apply -f frontend.yaml
```



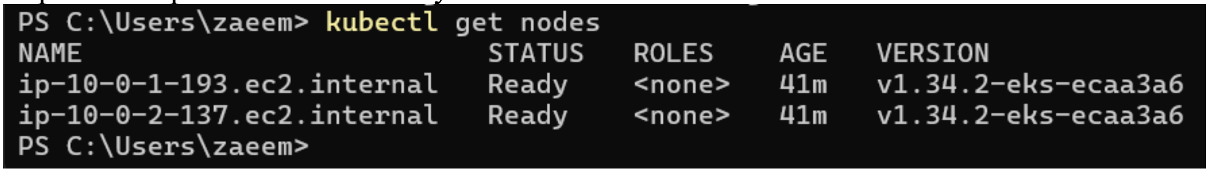
```
PS C:\Users\zaeem> kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
mysql-0                           0/1     Pending   0          27m
web-frontend-67dc5c8bc-5dfq5      1/1     Running   0          27m
web-frontend-67dc5c8bc-xsgd6      1/1     Running   0          27m
PS C:\Users\zaeem> |
```

Task 17.9: Testing and Verification

Verify Node Status:

```
kubectl get nodes
```

Expected output: 2 nodes in Ready state



```
PS C:\Users\zaeem> kubectl get nodes
NAME                          STATUS   ROLES    AGE   VERSION
ip-10-0-1-193.ec2.internal    Ready    <none>   41m   v1.34.2-eks-ecaa3a6
ip-10-0-2-137.ec2.internal    Ready    <none>   41m   v1.34.2-eks-ecaa3a6
PS C:\Users\zaeem>
```

Verify Pod Status:

```
kubectl get pods -o wide
```

Expected output:

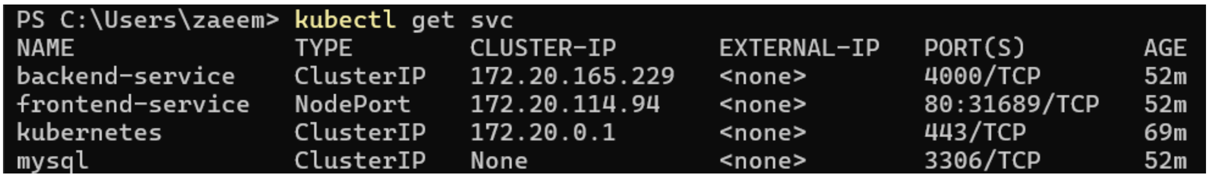
- 1 MySQL pod running
- 2 Backend pods running
- 2 Frontend pods running

Verify Services:

```
kubectl get svc
```

Expected output:

- mysql (ClusterIP, port 3306)
- backend-service (ClusterIP, port 4000)
- frontend-service (NodePort, port 80:30000)

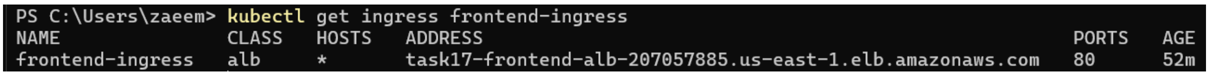


```
PS C:\Users\zaeem> kubectl get svc
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
backend-service    ClusterIP   172.20.165.229   <none>        4000/TCP       52m
frontend-service   NodePort    172.20.114.94    <none>        80:31689/TCP   52m
kubernetes         ClusterIP   172.20.0.1       <none>        443/TCP        69m
mysql              ClusterIP   None             <none>        3306/TCP       52m
```

Verify Ingress and ALB:

```
kubectl get ingress frontend-ingress
```

Expected output: Ingress with ALB DNS name in ADDRESS column

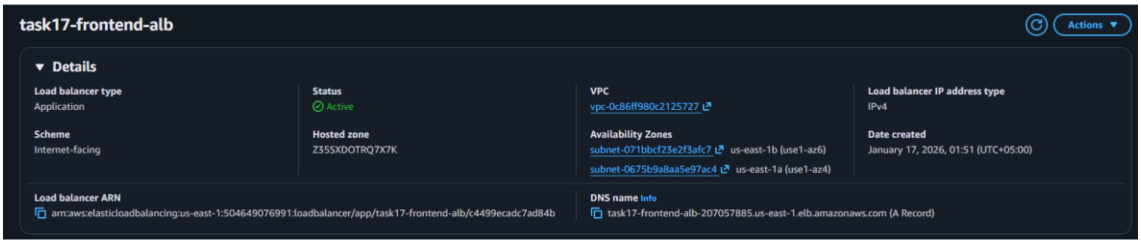


```
PS C:\Users\zaeem> kubectl get ingress frontend-ingress
NAME               CLASS   HOSTS   ADDRESS                                                  PORTS   AGE
frontend-ingress   alb     *       task17-frontend-alb-207057885.us-east-1.elb.amazonaws.com   80      52m
```

Verify ALB in AWS Console:

Navigate to EC2 → Load Balancers

- Load Balancer Name: task17-frontend-alb
- State: Active
- Scheme: internet-facing
- Availability Zones: us-east-1a, us-east-1b



task17-frontend-alb

▼ Details

Load balancer type
Application

Scheme
Internet-facing

Status
Active

Hosted zone
Z35SXDOTRQ7X7K

VPC
vpc-0c86ff980c2125727

Availability Zones
subnet-071bbcf23e2f3afc7 us-east-1b (use1-az6)
subnet-0675b9a8aa5e97ac4 us-east-1a (use1-az4)

Load balancer IP address type
IPv4

Date created
January 17, 2026, 01:51 (UTC+05:00)

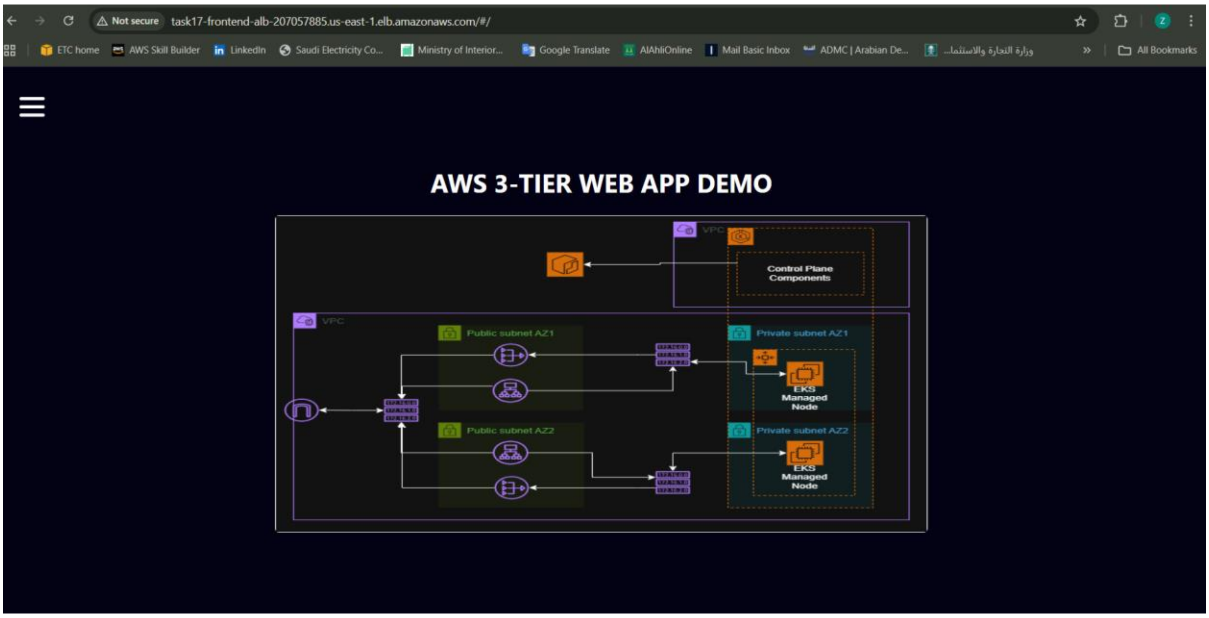
Load balancer ARN
arn:aws:elasticloadbalancing:us-east-1:504649076991:loadbalancer/app/task17-frontend-alb/c4499ecadc7ad84b

DNS name Info
task17-frontend-alb-207057885.us-east-1.elb.amazonaws.com (A Record)

Access Application:

```
    $ALB_URL = kubectl get ingress frontend-ingress -o
jsonpath='{.status.loadBalancer.ingress[0].hostname}'
    Write-Host "Application URL: http://$ALB_URL"
```

Open the URL in a web browser to access the application on port 80.



task17-frontend-alb-207057885.us-east-1.elb.amazonaws.com/#/

AWS 3-TIER WEB APP DEMO

Task 17.10: Lessons Learned

EKS and Kubernetes:

- EKS simplifies Kubernetes cluster management compared to self-managed clusters
- Managed node groups handle node lifecycle automatically
- OIDC provider is essential for IAM roles for service accounts (IRSA)
- Proper subnet tagging is critical for AWS Load Balancer Controller to function

Storage:

- EBS CSI Driver is mandatory for dynamic volume provisioning in EKS
- GP3 volumes offer better performance and cost efficiency than GP2
- `WaitForFirstConsumer` binding mode ensures volumes are created in the correct AZ
- StatefulSets with volume claim templates provide stable storage for databases

AWS Load Balancer Controller:

- Helm is the recommended installation method
- Service account must be annotated with IAM role ARN
- ALB provisioning takes 2-3 minutes after Ingress creation
- Target type `instance` works with NodePort services
- Health check configuration in Ingress annotations is important