AWS Marketplace AI/Run Deployment Guide on AWS

1. Overview

This guide provides step-by-step instructions for deploying the AI/Run CodeMie application to Amazon EKS and related AWS services. By following these instructions, you will:

- Get along with AI/Run CodeMie architecture
- Deploy AWS infrastructure using Terraform
- Configure and deploy all AI/Run CodeMie application components
- Integrate and configure AI models

1.1. How to Use This Guide

For successful deployment, please follow these steps in sequence:

1. First, verify all prerequisites and set up your AWS environment accordingly.Next, deploy the required infrastructure using Terraform.
2. Finally, deploy and configure the AI/Run CodeMie components on EKS cluster.
3. Complete post-installation configuration

Each section contains detailed instructions to ensure a smooth deployment process. The guide is structured to walk you through from initial setup to a fully functional AI/Run CodeMie environment on AWS.

2. Prerequisites

Before installing AI/Run CodeMie, carefully review the prerequisites and requirements.

2.1. Prerequisites Checklist

2.1.1.1. AWS Account Access Requirements

✓ Active AWS Account with preferable region for deployment

✓ User credentials with programmatic access to AWS account with permissions to create and manage IAM Roles and Policy Documents

2.1.1.2. Domain Name

✓ Available wildcard DNS hosted zone in Route53

> AI/Run CodeMie terraform modules will automatically create:
>
> - DNS Records
> - TLS certificate through AWS Certificate Manager, which will be used later by the ALB and NLB

2.1.1.3. External connections

✓ Firewall or SG and NACLs of EKS cluster allow outbound access to:

- AI/Run CodeMie container registry – <Need Update>
- 3rd party container registries – quay.io, docker.io, registry.developers.crunchydata.com
- Any service you're planning to use with AI/Run CodeMie (for example, GitLab instance)

✓ Firewall on your integration service allow inbound traffic from the AI/Run CodeMie NAT Gateway public IP address

> NAT Gateway public IP address will be known after EKS installation

2.1.1.4. LLM Models

✅ Activated region in AWS where AWS Bedrock Models are available

✅ Activated desired LLMs and embeddings models in AWS account (for example, Sonnet 3.5v3/3.7, AWS Titan 2.0)

> ⓘ AI/Run CodeMie can be deployed with mock LLM configurations initially. Real configurations can be provided later if client-side approvals require additional time.

2.1.1.5. User Permissions and Admission Control Requirements for EKS

✅ Admin EKS permissions with rights to create `namespaces`

✅ Admission webhook allows creation of Kubernetes resources listed below (applicable when deploying onto an existing EKS cluster with enforced policies):

AI/Run CodeMie Component	Kubernetes APIs	Description
NATS	`Service`	NATS messaging system requires a LoadBalancer service type for client-server communication. When running `codemie-plugins`: – within the same VPC as the EKS cluster – Internal LoadBalancer configured for secure, private network communication – outside the EKS cluster's VPC – Public LoadBalancer required for cross-network communication

keycloak-operator	`ClusterRole`, `ClusterRoleBinding`, `Role`
	`RoleBinding`, `CRDs`, `CRs`
		Cluster-wide permissions required for managing Keycloak configuration, including realms, clients, and user federation settings

Postgres-operator	`ClusterRole`, `ClusterRoleBinding`, `CRDs`, `CRs`	Cluster-wide permissions required for managing PostgreSQL instances and their lifecycle
Elasticsearch	`Pod(securityContext)`	InitContainer must run as root user to set system parameter `vm.max_map_count=262144`
All components	`Pod(securityContext)`	All components require SecurityContext with `readOnlyRootFilesystem: false` for proper operation

2.2. Deployer instance requirements

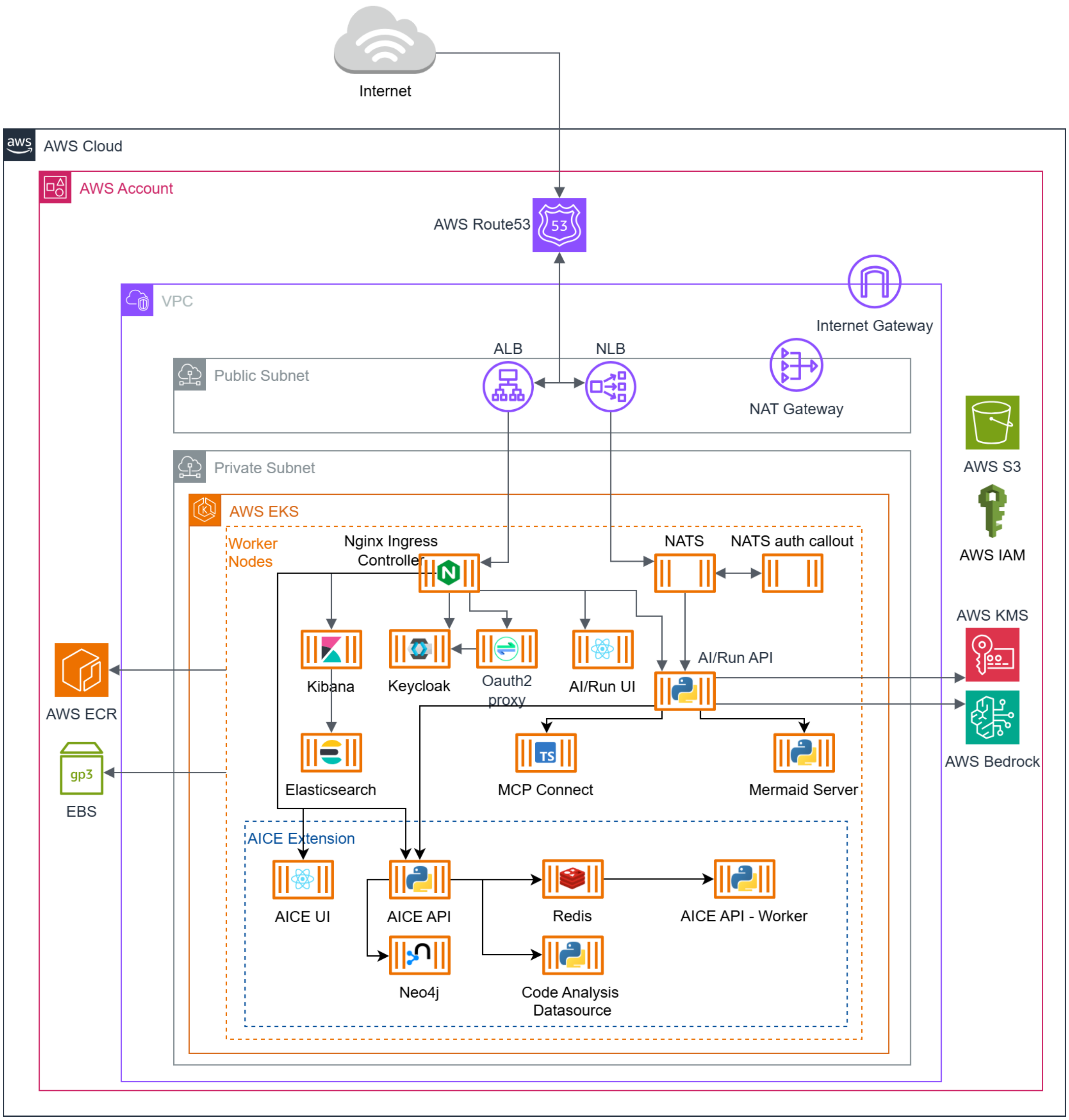
✅ The next software must be pre-installed and configured on the deployer laptop or VDI instance before beginning the deployment process(If you're using Windows, avoid mixing WSL with a native Windows installation):

- terraform `v1.5.7`
- kubectl
- helm `v3.16.0+`
- AWS CLI
- docker
- natscli
- nsc
- htpasswd

> ⓘ If you use Windows,. please, use linux shells such as Git Bash, WSL, etc

3. AI/Run CodeMie deployment architecture

The diagram below depicts the AI/Run CodeMie infrastructure deployment in one region (AZ) of the AWS public cloud environment.



3.1. Container Resources Requirements

Component	Pods	RAM	vCPU
CodeMie API	2	8Gi	4.0
CodeMie UI	1	128Mi	0.1
Elasticsearch	2	16Gi	4.0
Kibana	1	1Gi	1.0
Mermaid-server	1	512Mi	1.0
PostgreSQL	1	1Gi	0.2
Keycloak + DB	1 + 1	4Gi	2.0

Oauth2-proxy	1	128Mi	0.1
NATS + Auth Callout	1 + 1	512Mi	1.0
MCP Connect	1	1Gi	0.5
Fluentbit	daemonset	128Mi	0.1
LLM Proxy*	1	1Gi	1.0

*Depends on the exact LLM proxy type

4. AWS Infrastructure Deployment

4.1. Overview

Skip if you have ready EKS cluster with all required services (check the diagram above).

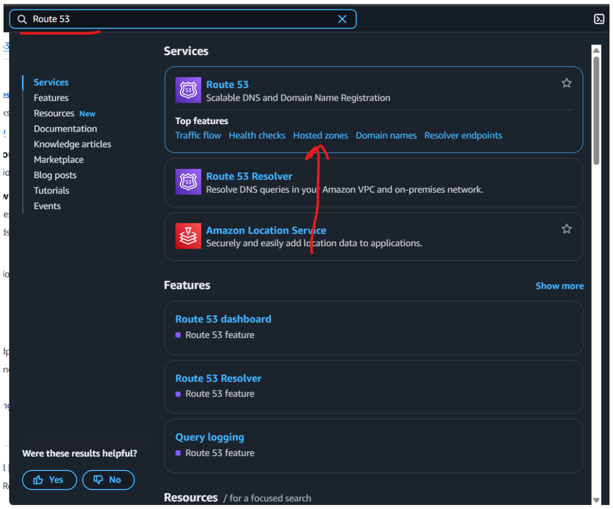
This section describes the process of deploying the AI/Run CodeMie infrastructure within an AWS environment. Terraform is used to manage resources and configure services.

⚠ A crucial step involves using a registered domain name added to AWS Route 53, which allows Terraform to automatically create SSL/TLS certificates via AWS Certificate Manager. These certificates are essential for securing traffic handled by the Application Load Balancer (ALB) and Network Load Balancer (NLB).

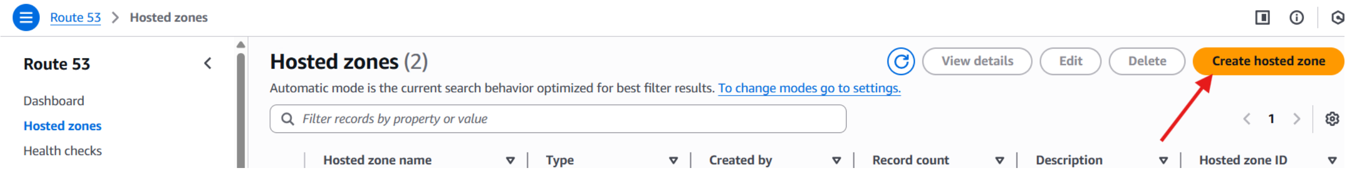
There are two deployment options available. Use the script if you want an easier deployment flow. Use the manual option if you want to control Terraform resources and provide customization.

4.2. Set up Hosted zone

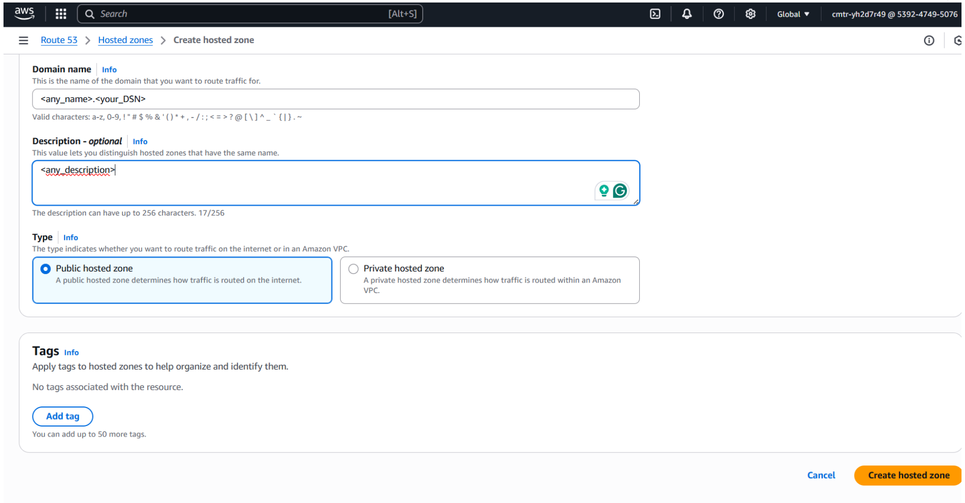
4.2.1. Open Hosted zone page



4.2.2. Click on Create hosted zone button



4.2.3. Create new hosted zone. Domaine name should has next pattern <any_name>.<your_DNS>



Route 53 > Hosted zones > Create hosted zone

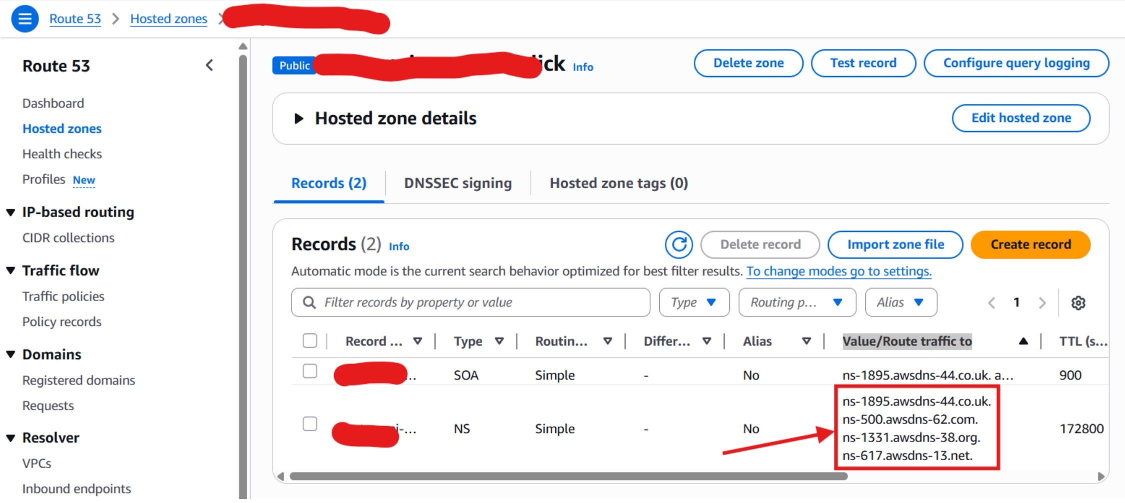
Domain name | Info
This is the name of the domain that you want to route traffic for.
<any_name>.<your_DSN>
Valid characters: a-z, 0-9, ! " # $ % & ' ( ) * + , - / : ; < = > ? @ [\ ] ^ _ ` { | } . ~

Description - *optional* | Info
This value lets you distinguish hosted zones that have the same name.
<any_description>
The description can have up to 256 characters. 17/256

Type | Info
The type indicates whether you want to route traffic on the internet or in an Amazon VPC.
● Public hosted zone
A public hosted zone determines how traffic is routed on the internet.
○ Private hosted zone
A private hosted zone determines how traffic is routed within an Amazon VPC.

Tags Info
Apply tags to hosted zones to help organize and identify them.
No tags associated with the resource.
You can add up to 50 more tags.

4.2.4. Copy "Value/Route traffic to" value from NS record



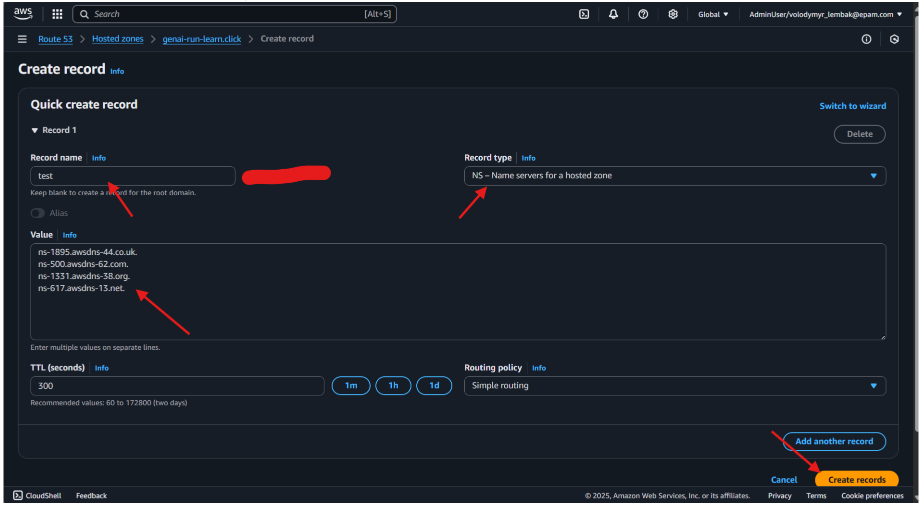
Route 53 > Hosted zones > [redacted]

Public [redacted]ick Info

☐	Record ...	Type	Routin...	Differ...	Alias	Value/Route traffic to ▲	TTL (s...
☐	[redacted]..	SOA	Simple	-	No	ns-1895.awsdns-44.co.uk. a...	900
☐	[redacted]i-...	NS	Simple	-	No	ns-1895.awsdns-44.co.uk. ns-500.awsdns-62.com. ns-1331.awsdns-38.org. ns-617.awsdns-13.net.	172800

4.2.5. Open parent Hosted zone with name which equal to DNS name

Create new record in hosted zone from previous step
Record name - should be the same value as <any_name> from step 4.2.4
Record type - select "NS" option
Value - Past value from step 4



Route 53 > Hosted zones > genai-run-learn.click > Create record

Create record Info

Quick create record

Record 1

Record name | Info
test [redacted]
Keep blank to create a record for the root domain.

Record type | Info
NS – Name servers for a hosted zone

Value | Info
ns-1895.awsdns-44.co.uk.
ns-500.awsdns-62.com.
ns-1331.awsdns-38.org.
ns-617.awsdns-13.net.
Enter multiple values on separate lines.

TTL (seconds) | Info
300
Recommended values: 60 to 172800 (two days)

Routing policy | Info
Simple routing

© 2025, Amazon Web Services, Inc. or its affiliates.

4.3. Set up credential to AWS

1. Find or create ".credential" file. By default, the file is located in the following directory:
   "/Users/<user_name>/.aws" - Linux/Mac
   "C:\Users\<user_name>\.aws" - Windows
2. Open the file and update next property: aws_region, aws_access_key_id, aws_secret_access_key, aws_session_token (if you use temporary credential)

4.4. Clone repository

```
git clone https://github.com/epam/EPAM-AI-RUN-Marketplace/tree/main
cd EPAM-AI-RUN-Marketplace\deployment\terraform-scripts
```

4.5. Scripted Deployment

4.5.1. Run Installation Script

The `terraform.sh` script automates the deployment of infrastructure.

To deploy AI/Run CodeMie infrastructure to AWS use the folowing steps:

1. Fill configuration details that specific for your AWS account in `deployment.conf`:

```
# AI/Run CodeMie deployment variables configuration
# Fill required values and save this file as deployment.conf

TF_VAR_region="<REGION>" # Example: us-east-1
TF_VAR_subnet_azs='[<SUBNET AZS>]' # Example: '["us-east-1a", "us-east-1b", "us-east-1c"]'

TF_VAR_platform_name="<PLATFORM NAME>" # Example: codemie
TF_VAR_deployer_role_name="<ROLE>" # Example: AIRunDeployerRole

TF_VAR_s3_states_bucket_name="<BUCKET NAME>" # Example: codemie-terraform-states
TF_VAR_table_name="<TABLE NAME>" # Example: codemie_terraform_locks

TF_VAR_platform_domain_name="<DOMAIN NAME>" # Example: example.com

TF_VAR_spot_instance_types='[{"instance_type":"c5.2xlarge"}]'
TF_VAR_spot_max_nodes_count=0
TF_VAR_spot_desired_nodes_count=0
TF_VAR_spot_min_nodes_count=0
TF_VAR_demand_instance_types='[{"instance_type":"c5.2xlarge"}]'
TF_VAR_demand_max_nodes_count=2
TF_VAR_demand_desired_nodes_count=2
TF_VAR_demand_min_nodes_count=1
```

2. Run the following command if using a Unix-like operating system:

```
chmod +x terraform.sh
```

3. Run installation script, possible flags:
   --access-key ACCESS_KEY; Use the flag if the `.aws\credentials` file has not been updated.
   --secret-key SECRET_KEY;   Use the flag if the `.aws\credentials` file has not been updated.
   --region REGION;   Use the flag if the `.aws\credentials` file has not been updated.
   --rds-enable;
   --config-file FILE ;   Load configuration from file (default: deployment.conf)
   --help;

```
bash terraform.sh
or
./terraform.sh
```

After execution, the script will:

1. Validate your deployment environment:
   a. Check for required tools (`kubectl`, `AWS CLI`, `terraform`)
   b. Verify AWS authentication status
   c. Validate configuration parameters
2. Create IAM Deployer role and policy :
3. Deploy Infrastructure:
   a. Create Terraform backend storage (S3 bucket and DynamoDB table)
   b. Deploy core AI/Run CodeMie Platform infrastructure
   c. Set up necessary AWS resources
4. Generate Outputs:
   a. The script will create a `deployment_outputs.env` file containing essential infrastructure details:

      ```
      AWS_DEFAULT_REGION=eu-west-2
      EKS_AWS_ROLE_ARN=arn:aws:iam::123456789012:role/...
      AWS_KMS_KEY_ID=12345678-90ab-cdef-1234-567890abcdef
      AWS_S3_BUCKET_NAME=codemie-platform-bucket
      ```

   b. If the user includes the --rds-enable flag, the `deployment_outputs.env` file will be generated with the relevant infrastructure details.:

      ```
      AWS_DEFAULT_REGION=eu-west-2
      EKS_AWS_ROLE_ARN=arn:aws:iam::123456789012:role/...
      AWS_KMS_KEY_ID=12345678-90ab-cdef-1234-567890abcdef
      AWS_S3_BUCKET_NAME=codemie-platform-bucket


      AWS_RDS_ADDRESS=codemie.aaaaaaaaaaaa.us-east-1.rds.amazonaws.com
      AWS_RDS_DATABASE_NAME=codemie
      AWS_RDS_DATABASE_USER=dbadmin
      AWS_RDS_DATABASE_PASSWORD=SomePassword
      ```

5. Deployment Completion:
   a. A success message will confirm the deployment
   b. Logs will be available in `codemie_aws_deployment_YYYY-MM-DD-HHMMSS.log`
   c. The script will display a summary of deployed resources

> Keep the `deployment_outputs.env` file secure as it contains sensitive information. Do not commit it to version control.

After successful deployment, you can proceed with the AI/Run CodeMie components installation and start using AI/Run CodeMie services.

4.6. Manual Deployment (If the previous step has already been completed, please proceed to skip this step.)

4.6.1. Deployment Order

#	Resource name
1	IAM deployer role
2	Terraform Backend
3	Terraform Platform

4.6.2. IAM Deployer Role creation

This step covers the `DeployerRole` AWS IAM role creation.

> The created IAM role will be used for all subsequent infrastructure deployments and contains required permissions to manage AWS resources

To create the role, take the following steps:

1. Navigate to codemie-aws-iam folder:

   ```
   cd codemie-aws-iam
   ```

2. Review the input variables for Terraform in the `deployment/terraform-scripts/codemie-aws-iam/variables.tf` file and create a `<fileName>.tfvars` in the repo to change default variables values there in a format of key-value. For example:

   ```
   region               = "your-region"
   role_arn             = "arn:aws:iam::xxx:role/yourRole"
   platform_domain_name = "your.domain"
   ...
   ```

   > Ensure you have carefully reviewed all variables and replaced mock values with yours.

3. Initialize the backend and apply the changes:

   ```
   terraform init --var-file <fileName>.tfvars
   terraform plan --var-file <fileName>.tfvars
   terraform apply --var-file <fileName>.tfvars
   ```

4.6.3. Terraform backend resources deployment

This step covers the creation of:

- S3 bucket with policy to store terraform states
- DynamoDB to support state locking and consistency checking

To create an S3 bucket for storing Terraform state files, follow the steps below:

1. Navigate to codemie-aws-remote-backend folder:

   ```
   cd ../codemie-aws-remote-backend
   ```

2. Review the input variables for Terraform in the `deployment/terraform-scripts/codemie-aws-remote-backend/variables.tf` file anc create a `<fileName>.tfvars` in the repo to change default variables values there in a format of key-value. For example:

   ```
   region               = "your-region"
   role_arn             = "arn:aws:iam::xxx:role/yourRole"
   platform_domain_name = "your.domain"
   ...
   ```

   > Ensure you have carefully reviewed all variables and replaced mock values with yours.

3. Initialize the backend and apply the changes:

   ```
   terraform init --var-file <fileName>.tfvars
   terraform plan --var-file <fileName>.tfvars
   terraform apply --var-file terraform.tfvars
   ```

The created S3 bucket will be used for all subsequent infrastructure deployments.

4.6.4. Terraform Platform

This step will cover the following topics:

- Create the EKS Cluster
- Create the AWS ASGs for the EKS Cluster
- Create the AWS ALB
- Create the AWS NLB
- Create the AWS KMS key to encrypt and decrypt sensitive data in the AI/Run CodeMie application.
- Create the AWS IAM Role to access the AWS KMS and Bedrock services
- Create the AWS IAM role ExternalSecretOperator to use AWS Systems Manager

To accomplish the tasks outlined above, follow these steps:

1. Navigate to codemie-aws-platform folder:

```
cd ../codemie-aws-platform
```

2. Review the input variables for Terraform in the `deployment/terraform-scripts/codemie-aws-platform/variables.tf` file and create a `<fileName>.tfvars` in the repo to manage custom variables there in a format of key-value. For example:

```
region                        = "us-east-1"
s3_states_bucket_name = "codemie-qa-terraform-states"
table_name                  = "codemie_qa_terraform_locks"
role_arn                      = "arn:aws:iam::111111111111:role/<RoleName>"
platform_domain_name          = "qat.genai-run-learn.click"
platform_name                 = "codemie-qat"
platform_cidr                 = "10.0.0.0/16"
subnet_azs = ["us-east-1a", "us-east-1b", "us-east-1c"]
private_cidrs = ["10.0.0.0/22", "10.0.4.0/22", "10.0.8.0/22"]
public_cidrs = ["10.0.12.0/24", "10.0.13.0/24", "10.0.14.0/24"]
ssl_policy                    = "ELBSecurityPolicy-TLS-1-2-2017-01"
eks_admin_role_arn            =  "arn:aws:iam::111111111111:user/<userName>"
add_userdata                  = ""
spot_instance_types = [{ instance_type = "c5.2xlarge" }]
spot_max_nodes_count          = 0
spot_desired_nodes_count      = 0
spot_min_nodes_count          = 0
demand_instance_types = [{ instance_type = "c5.2xlarge" }]
demand_max_nodes_count        = 2
demand_desired_nodes_count    = 2
demand_min_nodes_count        = 1
cluster_identity_providers = {}
aws_auth_users = []
aws_auth_roles = []
tags = {
  "SysName"     = "CodeMie"
  "Environment" = "qat"
  "Project"     = "CodeMie"
}
node_iam_role_additional_policies = [
  {
    sid    = "CloudWatchServerPermissions",
    effect = "Allow",
    actions = [
      "logs:PutLogEvents",
      "logs:DescribeLogStreams",
      "logs:DescribeLogGroups",
      "logs:CreateLogStream",
      "logs:CreateLogGroup"
    ],
    resources = ["*"]
  }
]
...
```

> Ensure you have carefully reviewed all variables and replaced mock values with yours

3. Initialize the platform and apply the changes:

```
terraform init --var-file <fileName>.tfvars
terraform plan --var-file <fileName>.tfvars
terraform apply --var-file <fileName>.tfvars
```

4.6.5. Terraform RDS

1. Navigate to codemie-aws-platform folder:

```
cd ../codemie-aws-rds
```

2. Review the input variables for Terraform in the `deployment/terraform-scripts/codemie-aws-rds/variables.tf` file and create a `<fileName>.tfvars` in the repo to change default variables values there in a format of key-value. For example:

```
region               = "your-region"
role_arn             = "arn:aws:iam::xxx:role/yourRole"
platform_domain_name = "your.domain"
vpc_state_bucket     = "your.vpc_state_bucket"
vpc_state_key             = "your.vpc_state_key"
...
```

3. Initialize the RDS and apply the changes:

```
terraform init --var-file <fileName>.tfvars
terraform plan --var-file <fileName>.tfvars
terraform apply --var-file <fileName>.tfvars
```

5. AI Models Integration and Configuration

5.1. Managing LLM and embedding models

AI/Run CodeMie provides a way to configure LLM and embedding models from different cloud providers. Configuration file can be found by path: `config/llms`.

The `MODELS_ENV` is used to specify the environment for the models. For example, `MODELS_ENV=dial` will use the models from the `config/llms/llm-dial-config.yaml` file (Pattern: `llm-<MODELS_ENV>-config.yaml`).

Example of providing LLM and embedding models for the custom environment:

1. Go to the `codemie-helm-charts/codemie-api/values-awss.yaml` file
2. Fill the following values to create and mount custom configmap to AI/Run pod:

```
extraEnv:
  - name: MODELS_ENV
    value: <project-name>

extraVolumeMounts: |
  ...
  - name: codemie-llm-customer-config
    mountPath: /app/config/llms/llm-<project-name>-config.yaml
    subPath: llm-<project-name>-config.yaml
  ...

extraVolumes: |
  ...
  - name: codemie-llm-customer-config
    configMap:
      name: codemie-llm-customer-config
  ...

extraObjects:
  - apiVersion: v1
    kind: ConfigMap
    metadata:
      name: codemie-llm-customer-config
    data:
      llm-amnaairn-config.yaml: |
        llm_models:
          - base_name: "mistral"
            deployment_name: "mistral.mistral-7b-instruct-v0:2"
            label: "Mistral 7b - Instruct"
            multimodal: false
            enabled: true
            default: true
            provider: "aws_bedrock"
            features:
              system_prompt: false
              max_tokens: false
            cost:
              input: 0.0000025
              output: 0.000011

        embeddings_models:
          - base_name: "titan"
            deployment_name: "amazon.titan-embed-text-v1"
            label: "Titan Embeddings G1 - Text"
            enabled: true
            default: true
            provider: "aws_bedrock"
            cost:
              input: 0.0000001
              output: 0
```

5.2. AWS Bedrock Models

5.2.1. Overview

This section describes the process of enabling AWS Bedrock models in AWS account.

5.2.2. Steps to Enable Bedrock Models

1. Access AWS Bedrock Console

1. Sign in to the AWS Management Console
2. Navigate to the AWS Bedrock service
3. Select "Model access" from the left navigation panel

2. Request Model Access

1. In the Model access page, you'll see available foundation models grouped by providers
2. Common providers include:
   - Anthropic (Claude models)
   - Amazon
3. For each model you want to enable:
   - Locate the model in the list
   - Check the checkbox next to the model name
   - Click "Request model access"

3. Verify Model Access

1. After requesting access, the status will initially show as "Pending"
2. Wait for the status to change to "Access granted"
3. This typically takes only a few minutes
4. Refresh the page to see updated status

4. Region-Specific Configuration

- Note that model access needs to be enabled separately for each AWS region
- Repeat the process for additional regions if needed

6. AI/Run CodeMie Components Deployment

6.1. Overview

This section describes the process of the main AI/Run CodeMie components deployment to the AWS EKS cluster.

6.1.1. Core AI/Run CodeMie Components:

> ⓘ AI/Run CodeMie current versions of codemie: 1.0.0

Component name	Images	Description
AI/Run CodeMie API	<Need Update>	The backend service of the AI/Run CodeMie application responsible for business logic, data processing, and API operations
AI/Run CodeMie UI	<Need Update>	The frontend service of the AI/Run CodeMie application that provides the user interface for interacting with the system
AI/Run CodeMie Nats Auth Callout	<Need Update>	Authorization component of AI/Run CodeMie Plugin Engine that handles authentication and authorization for the NATS messaging system
AI/Run CodeMie MCP Connect	<Need Update>	A lightweight bridge tool that enables cloud-based AI services to communicate with local Model Context Protocol (MCP) servers via protocol translation while maintaining security and flexibility
AI/Run Mermaid Server	<Need Update>	Implementation of open-source service that generates image URLs for diagrams based on the provided Mermaid code for workflow visualization

6.1.2. Required Third-Party Components:

Component name	Images	Description
Ingress Nginx Controller	`registry.k8s.io/ingress-nginx/controller:x.y.z`	Handles external traffic routing to services within the Kubernetes cluster. The AI/Run CodeMie application uses oauth2-proxy, which relies on the Ingress Nginx Controller for proper routing and access control
Storage Class	–	Provides persistent storage capabilities
Elasticsearch	`docker.elastic.co/elasticsearch/elasticsearch:x.y.z`	Database component that stores all AI/Run CodeMie data, including datasources, projects, and other application information
Kibana	`docker.elastic.co/kibana/kibana:x.y.z`	Web-based analytics and visualization platform that provides visualization of the data stored in Elasticsearch. Allows monitoring and analyzing AI/Run CodeMie data
Postgres-operator	`registry.developers.crunchydata.com/crunchydata/postgres-operator:x.y.z`	Manages PostgreSQL database instances required by other components in the stack. Handles database lifecycle operations
Keycloak-operator	`epamedp/keycloak-operator:x.y.z`	Manages Keycloak identity and access management instance and it's configuration
Keycloak	`docker.io/busybox:x.y.z`, `quay.io/keycloak/keycloak:`	Identity and access management solution that provides authentication and

	x.y.z, registry.developers.crunchydata.com/crunchydata/crunchy-postgres:x.y.z	authorization capabilities for integration with oauth2-proxy component
Oauth2-Proxy	quay.io/oauth2-proxy/oauth2-proxy:x.y.z	Authentication middleware that provides secure authentication for the AI/Run CodeMie application by integrating with Keycloak or any other IdP
NATS	nats:x.y.z, natsio/nats-server-config-reloader:x.y.z	Message broker that serves as a crucial component of the AI/Run CodeMie Plugin Engine, facilitating communication between services
DIAL Proxy	docker.io/epam/ai-dial-core:x.y.z, docker.io/epam/ai-dial-adapter-openai:x.y.z, docker.io/bitnami/redis-cluster:x.y.z	Optional proxy component that balances requests to Azure OpenAI language models (LLMs), providing high availability and load distribution
Fluentbit	cr.fluentbit.io/fluent/fluent-bit:x.y.z	Fluentbit enables logs and metrics collection from AI/Run CodeMie enabling the Agents observability
PostgreSQL	docker.io/bitnami/postgresql:x.y.z	Database component that stores all AI/Run CodeMie data, including datasources, projects, and other application information

6.2. Scripted AI/Run CodeMie Components Installation

1. Navigate helm-scripts folder:

   ```
   cd ../helm-scripts
   ```

2. Run the following command if using a Unix-like operating system:

   ```
   chmod +x /helm-scripts
   ```

3. Run deployment script:

   ```
   bash ./helm-charts.sh version=x.y.z
   or
   ./helm-charts.sh version=x.y.z
   ```

6.3. Manual Installation AI/Run CodeMie (If the previous step has already been completed, please proceed to skip this step.)

6.3.1. Set up kubectl conig

Run next command

```
aws eks update-kubeconfig --region <REGION> --name <PLATFORM_NAME>
```

6.3.2. Nginx Ingress controller

Install only in case if your EKS cluster does not have Nginx Ingress Controller.

1. Create Kubernetes namespace, e.g. `ingress-nginx` with the command:

   ```
   kubectl create namespace ingress-nginx
   ```

2. Navigate helm-scripts folder

   ```
   cd ../helm-scripts
   ```

3. Install `ingress-nginx` helm chart in created namespace:

```
helm upgrade --install ingress-nginx ingress-nginx/. -n ingress-nginx --values ingress-nginx/values-aws.
yaml --wait --timeout 900s --dependency-update
```

6.3.3. AWS gp3 storage class:

Install only in case if your EKS cluster does not have AWS gp3 storage class:

```
kubectl apply -f storage-class/storageclass-aws-gp3.yaml
```

6.3.4. Install Elasticsearch component:

1. Create Kubernetes namespace, e.g. `elastic` with the command:

   ```
   kubectl create namespace elastic
   ```

2. Create Kubernetes secret:

   ```
   kubectl -n elastic create secret generic elasticsearch-master-credentials \
     --from-literal=username=elastic \
     --from-literal=password="$(openssl rand -base64 12)" \
     --type=Opaque \
     --dry-run=client -o yaml | kubectl apply -f -
   ```

   Secret example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: elasticsearch-master-credentials
   type: Opaque
   data:
     username: <base64-encoded-username>
     password: <base64-encoded-password>
   ```

3. Install `elasticsearch` helm chart in created namespace with the command:

   ```
   helm upgrade --install elastic elasticsearch/. -n elastic --values elasticsearch/values-aws.yaml --wait
   --timeout 900s --dependency-update
   ```

6.3.5. Install Kibana component:

1. Fill in missing values in values.yaml file by replacing `%%DOMAIN%%` with your domain name, e.g. `example.com`
2. Install `kibana` helm chart with the command:

   ```
   helm upgrade --install kibana kibana/. -n elastic --values kibana/values-aws.yaml --wait --timeout 900s
   --dependency-update
   ```

3. Kibana can be accessed by the following URL: `https://kibana.%%DOMAIN%%` , e.g. `https://kibana.example.com`

6.3.6. Install Postgres-operator component:

1. Apply `postgres-operator` chart:

   ```
   helm upgrade --install postgres-operator postgres-operator-helm/. -n postgres-operator --create-
   namespace --wait --timeout 900s --dependency-update
   ```

6.3.7. Install Keycloak-operator component:

1. Create `security` namespace and `keycloak-admin` secret:

```
kubectl create namespace security

kubectl -n security create secret generic keycloak-admin \
  --from-literal=username=admin \
  --from-literal=password="$(openssl rand -base64 12)" \
  --type=Opaque \
  --dry-run=client -o yaml | kubectl apply -f -
```

2. Apply `keycloak-operator` helm chart with the command:

```
helm upgrade --install keycloak-operator-helm keycloak-operator-helm/. -n security --create-namespace --values keycloak-operator-helm/values.yaml --wait --timeout 900s --dependency-update
```

6.3.8. Install Keycloak component:

1. Fill in values in values.yaml and apply `keycloak` helm chart with the command:

```
helm upgrade --install keycloak keycloak-helm/. -n security --values keycloak-helm/values-aws.yaml  --wait --timeout 900s --dependency-update
```

Keycloak Admin UI can be accessed by the following URL: `https://keycloak.%%DOMAIN%%/auth/admin` , e.g. `https://keycloak.example.com/auth/admin`

6.3.9. Install AI/Run CodeMie NATS component:

To deploy a NATS, follow the steps below:

1. Create `codemie-nats-secrets` Kubernetes secret. To set up it, follow these steps to generate and encode the necessary values:
    a. **NATS_URL**
        - Since NATS is deployed in the same namespace as the AI/Run CodeMie and NATS Callout services, use the internal URL `nats://codemie-nats:4222`
        - Base64 encode this URL before using it in the secret.

    b. **CALLOUT_USERNAME**
        - Use the username `callout`.
        - Base64 encode this username before using it in the secret.

    c. **CALLOUT_PASSWORD**
        - Generate a secure password using the command: `pwgen -s -1 25`.
        - Base64 encode this password before using it in the secret.

    d. **CALLOUT_BCRYPTED_PASSWORD**
        - Use the NATS server to generate a bcrypt-hashed password based on the `CALLOUT_PASSWORD`.
        - Command: `nats server passwd -p <CALLOUT_PASSWORD>`
        - Base64 encode the bcrypt-hashed password before using it in the secret.

    e. **CODEMIE_USERNAME**
        - Use the username `codemie`.
        - Base64 encode this username before using it in the secret.

    f. **CODEMIE_PASSWORD**
        - Generate a secure password using the command: `pwgen -s -1 25`.
        - Base64 encode this password before using it in the secret.

    g. **CODEMIE_BCRYPTED_PASSWORD**
        - Use the NATS server to generate a bcrypt-hashed password based on the `CODEMIE_PASSWORD`.
        - Command: `nats server passwd -p <CODEMIE_PASSWORD>`
        - Base64 encode the bcrypt-hashed password before using it in the secret.

    h. **ISSUER_NKEY and ISSUER_NSEED**

- Use the nsc tool to generate NATS account keys. For example: https://natsbyexample.com/examples/auth/callout/cli
- Command: `nsc generate nkey --account`
- Base64 encode the NKEY and NSEED before using them in the secret.

i. **ISSUER_XKEY and ISSUER_XSEED**

- Use the nsc tool to generate NATS curve keys. For example: https://natsbyexample.com/examples/auth/callout/cli
- Command: `nsc generate nkey --curve`
- Base64 encode the XKEY and XSEED before using them in the secret.

Secret example:

```
apiVersion: v1
kind: Secret
metadata:
  name: codemie-nats-secrets
type: Opaque
data:
  NATS_URL: <base64-encoded-nats-url>
  CALLOUT_USERNAME: <base64-encoded-callout-username>
  CALLOUT_PASSWORD: <base64-encoded-callout-password>
  CALLOUT_BCRYPTED_PASSWORD: <base64-encoded-callout-bcrypted-password>
  CODEMIE_USERNAME: <base64-encoded-codemie-username>
  CODEMIE_PASSWORD: <base64-encoded-codemie-password>
  CODEMIE_BCRYPTED_PASSWORD: <base64-encoded-codemie-bcrypted-password>
  ISSUER_NKEY: <base64-encoded-issuer-nkey>
  ISSUER_NSEED: <base64-encoded-issuer-nseed>
  ISSUER_XKEY: <base64-encoded-issuer-xkey>
  ISSUER_XSEED: <base64-encoded-issuer-xseed>
```

Use the following command `echo -n 'your-value-here' | base64` to encode secret or use `kubectl` to create secret from (i.e. `kubectl -n codemie create secret generic --from-literal NATS_URL=nats://codemie-nats:4222 --from-literal CALLOUT_USERNAME=callout ...`)

Alternatively, a Bash script can be used

```
#!/bin/bash

set -euo pipefail

namespace="codemie"
secret_name="codemie-nats-secrets"


log_message() {
    local status="$1"
    local message="$2"
    local timestamp=$(date '+%Y-%m-%d %H:%M:%S')

    case "$status" in
        "success")
            echo -e "[$timestamp] [OK] $message" ;;
        "fail")
            echo -e "[$timestamp] [ERROR] $message" ;;
        "info")
            echo -e "[$timestamp] $message" ;;
        "warn")
            echo -e "[$timestamp] [WARN] $message" ;;
        *)
            echo -e "[$timestamp] $message" ;;
    esac
}

log_message "info" "Creating secret '$secret_name' in namespace '$namespace'..."
callout_password=$(openssl rand -hex 16)
```

```
    codemie_password=$(openssl rand -hex 16)
    bcrypted_callout_password=$(htpasswd -bnBC 10 "" "${callout_password}" | tr -d ':\n' | sed 's/$2y
/$2a/')
    bcrypted_codemie_password=$(htpasswd -bnBC 10 "" "${codemie_password}" | tr -d ':\n' | sed 's/$2y
/$2a/')

    ISSUER_NKEY=""
    ISSUER_NSEED=""
    log_message "info" "Creating secret '$secret_name' in namespace '$namespace'..."
    output_nkey_account=$(nsc generate nkey --account 2>&1)
    log_message "info" "Creating secret '$secret_name' in namespace '$namespace'..."
    while IFS= read -r line; do
        if [[ $line == A* ]]; then
            ISSUER_NKEY="$line"
            log_message "info" "ISSUER_NKEY: 123456789"
        elif [[ $line == S* ]]; then
            ISSUER_NSEED="$line"
            log_message "info" "ISSUER_NKEY: asdfghjk"
        fi
    done <<< "$output_nkey_account"
    if [[ -n $ISSUER_NKEY && -n $ISSUER_NSEED ]]; then
        log_message "info" "ISSUER_NKEY: ${ISSUER_NKEY:0:8}...${ISSUER_NKEY: -8}"
        log_message "info" "ISSUER_NSEED: ${ISSUER_NSEED:0:8}...${ISSUER_NSEED: -8}"
    else
        log_message "fail" "Either ISSUER_NKEY or ISSUER_NSEED is empty."
        exit 1
    fi

    ISSUER_XKEY=""
    ISSUER_XSEED=""
    output_nkey_curve=$(nsc generate nkey --curve 2>&1)
    while IFS= read -r line; do
        if [[ $line == X* ]]; then
            ISSUER_XKEY="$line"
        elif [[ $line == S* ]]; then
            ISSUER_XSEED="$line"
        fi
    done <<< "$output_nkey_curve"
    if [[ -n $ISSUER_XKEY && -n $ISSUER_XSEED ]]; then
        log_message "info" "ISSUER_XKEY: ${ISSUER_XKEY:0:8}...${ISSUER_XKEY: -8}"
        log_message "info" "ISSUER_XSEED: ${ISSUER_XSEED:0:8}...${ISSUER_XSEED: -8}"
    else
        log_message "fail" "Either ISSUER_XKEY or ISSUER_XSEED is empty."
        exit 1
    fi

    kubectl -n "$namespace" create secret generic "$secret_name" \
      --from-literal=NATS_URL="nats://codemie-nats:4222" \
      --from-literal=CALLOUT_USERNAME="callout" \
      --from-literal=CALLOUT_PASSWORD="${callout_password}" \
      --from-literal=CALLOUT_BCRYPTED_PASSWORD="${bcrypted_callout_password}" \
      --from-literal=CODEMIE_USERNAME="codemie" \
      --from-literal=CODEMIE_PASSWORD="${codemie_password}" \
      --from-literal=CODEMIE_BCRYPTED_PASSWORD="${bcrypted_codemie_password}" \
      --from-literal=ISSUER_NKEY="${ISSUER_NKEY}" \
      --from-literal=ISSUER_NSEED="${ISSUER_NSEED}" \
      --from-literal=ISSUER_XKEY="${ISSUER_XKEY}" \
      --from-literal=ISSUER_XSEED="${ISSUER_XSEED}" \
      --type=Opaque -o yaml
```

2. Install `codemie-nats` helm chart in created namespace, applying custom values file with the command:

```
helm repo add nats https://nats-io.github.io/k8s/helm/charts/
helm repo update nats
helm upgrade --install codemie-nats nats/nats --version 1.2.6 \
--namespace codemie --values ./codemie-nats/values-aws.yaml \
--wait --timeout 900s
```

ⓘ In AWS, if TLS termination for Plugin Engine load balancer is handled by NLB (TLS certificate is on LB itself) then Plugin Engine NATS URL should start with `tls` protocol, for example: `tls://codemie-nats.example.com:30422` , otherwise use `nats://codemie-nats.example.com:30422`

6.3.10. Install AI/Run CodeMie NATS Auth Callout component:

To deploy a NATS Auth Callout service, follow the steps below:

1. Create `codemie` namespace with the command:

   ```
   kubectl create namespace codemie
   ```

2. Install `codemie-nats-auth-callout` helm chart, applying custom values file with the command:

   ```
   helm upgrade --install codemie-nats-auth-callout \
   "oci://europe-west3-docker.pkg.dev/or2-msq-epmd-edp-anthos-t1iylu/helm-charts/codemie-nats-auth-callout"
   \
   --version "x.y.z" \
   --namespace "codemie" \
   -f "./codemie-nats-auth-callout/values-aws.yaml" \
   --wait --timeout 600s
   ```

6.3.11. Install AI/Run CodeMie MCP Connect component:

1. Install `mcp-connect` helm chart with the command:

   ```
   helm upgrade --install codemie-mcp-connect-service <Need Update>/helm-charts/codemie-mcp-connect-service
   \
   --version x.y.z \
   --namespace "codemie" \
   -f "./codemie-mcp-connect-service/values.yaml" \
   --wait --timeout 600s
   ```

6.3.12. Install PostgreSQL component:

> Required starting from AI/Run CodeMie 1.0.0 version

1. Create `codemie-postgresql` with postgresql passwords:

   ```
   kubectl create secret generic codemie-postgresql \
   --from-literal=password=$(openssl rand -base64 12) \
   --from-literal=postgres-password=$(openssl rand -base64 12) \
   --namespace codemie
   ```

   Secret example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: codemie-postgresql
     namespace: codemie
   data:
     password: <base64-encoded-password>
     postgres-password: <base64-encoded-postgres-password>
   type: Opaque
   ```

2. Install `PostgreSQL` helm chart with the command:

   ```
   helm repo add bitnami https://charts.bitnami.com/bitnami

   helm repo update

   helm upgrade --install codemie-postgresql bitnami/postgresql \
   --version 16.7.4 \
   --values ./codemie-postgresql/values-aws.yaml \
   ```

```
--namespace codemie \
--wait --timeout 600s \
--dependency-update
```

6.3.13. Install OAuth2 Proxy component:

Authentication middleware that provides secure authentication for the AI/Run CodeMie application by integrating with Keycloak

1. Create Kubernetes namespace, e.g. `oauth2-proxy` with the command:

   ```
   kubectl create namespace oauth2-proxy
   ```

2. Create `oauth2-secret` with keycloak client data:

   ```
   kubectl create secret generic oauth2-proxy \
   --namespace=oauth2-proxy \
   --from-literal=client-id='codemie' \
   --from-literal=client-secret="$(openssl rand -base64 12)" \
   --from-literal=cookie-secret=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64 | tr -d -- '\n' |
   tr -- '+/' '-_' ; echo) \
   --type=Opaque
   ```

   Secret example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: oauth2-proxy
     namespace: oauth2-proxy
   data:
     client-id: <base64-encoded-client-id>
     client-secret: <base64-encoded-client-secret>
     cookie-secret: <base64-encoded-cookie-secret>
   type: Opaque
   ```

3. Copy `keycloak` secret to `oauth2-proxy` namespace:

   ```
   kubectl get secret keycloak-admin -n security -o yaml | sed '/namespace:/d' | kubectl apply -n oauth2-
   proxy -f -
   ```

4. Fill in missing values in values.yaml file by replace `%%DOMAIN%%` with your domain name, e.g. `example.com`
5. Install `oauth2-proxy` helm chart in created namespace with the command:

   ```
   helm upgrade --install oauth2-proxy oauth2-proxy/. -n oauth2-proxy --values oauth2-proxy/values-aws.yaml
   --wait --timeout 900s --dependency-update
   ```

6.3.14. Install AI/Run CodeMie UI component:

1. Fill in missing values in values.yaml file in `codemie-helm-charts/codemie-ui`by replacing `%%DOMAIN%%` with your domain name, e.g. `example.com`
2. Install `codemie-ui` helm chart in created namespace, applying custom values file with the command:

   ```
   helm upgrade --install codemie-ui <Need Update>/helm-charts/codemie-ui \
   --version x.y.z \
   --namespace "codemie" \
   -f "./codemie-ui/values-aws.yaml" \
   --wait --timeout 180s
   ```

3. Deploy AI/Run CodeMie API component.

6.3.15. Install AI/Run Mermaid Server component

1. Install mermaid-server helm chart with the command:

```
helm upgrade --install mermaid-server oci://europe-west3-docker.pkg.dev/or2-msq-epmd-edp-anthos-t1iylu
/helm-charts/mermaid-server \
--version x.y.z \
--namespace "codemie" \
-f "./mermaid-server/values.yaml" \
--wait --timeout 600s
```

6.3.16. Install AI/Run CodeMie API component:

1. Fill in missing values in values.yaml file in `codemie-helm-charts/codemie-api` :
   a. Replace `%%DOMAIN%%` with your domain name, e.g. `example.com`
   b. Replace `%%AWS_DEFAULT_REGION%%` with your AWS region, e.g. `us-west-2`
   c. Replace `%%EKS_AWS_ROLE_ARN%%` with your AWS IAM Role arn, e.g. `arn:aws:iam::0123456789012:role/AWSIRSA_AI_RUN`
   d. Replace `%%AWS_KMS_KEY_ID%%` with your KMS Key ID, e.g. `50f3f093-dc86-48de-8f2d-7a76e480348c`
2. Copy Elasticsearch credentials to the application namespace with the command:

```
kubectl get secret elasticsearch-master-credentials -n elastic -o yaml | sed '/namespace:/d' | kubectl
apply -n codemie -f -
```

3. Install `codemie-api` helm chart, applying custom values file with the command:

```
helm upgrade --install codemie-api <Need Update>/helm-charts/codemie \
--version x.y.z \
--namespace "codemie" \
-f "./codemie-api/values-aws.yaml" \
--wait --timeout 600s
```

4. AI/Run CodeMie UI can be accessed by the following URL: `https://codemie.%%DOMAIN%%` , e.g. `https://codemie.example.com`

6.3.17. Install Fluentbit component

If you do not have your own logging system then consider installing Fluentbit component to store historical log data.

1. Create `fluentbit` namespace:

```
kubectl create ns fluentbit
```

2. Copy Elasticsearch credentials to the `fluentbit` namespace with the command:

```
kubectl get secret elasticsearch-master-credentials -n elastic -o yaml | sed '/namespace:/d' | kubectl
apply -n fluentbit -f -
```

3. Install `fluentbit` with the command:

```
helm upgrade --install fluent-bit fluent-bit/. -n fluentbit --values fluent-bit/values.yaml --wait --
timeout 900s --dependency-update
```

4. Go to Kibana and setup `codemie_infra_logs*` index to view historical logs.

6.3.18. Install Kibana Dashboards

AI/Run CodeMie supports custom metrics about Assistants usage, including token consumption, costs, and engagement patterns. To view these metrics and gain valuable insights into your AI assistant interactions, the Kibana dashboard installation is required.

1. Run the script with the next arguments:

```
bash ./kibana-dashboards/import-kibana-dashboards.sh --url "https://kibana.url"
```

use `--force` to recreate existing resources in Kibana.

7. AI/Run CodeMie post-installation configuration

7.1. Required Steps

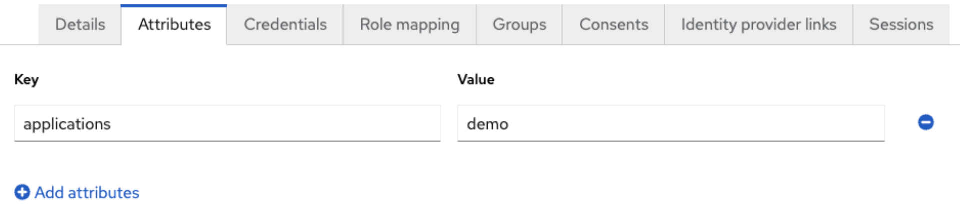
Before onboarding users few additional configuration steps are required:

7.1.1. Keycloak AI/Run CodeMie Realm configuration

It's crucial to make additional realm configurations.

Enable realm unmanaged attributes:

1. Open Keycloak console
2. Choose `codemie-prod` realm
3. Click on Realm Settings
4. Select `Enabled` for "Unmanaged Attributes" parameter.
5. Create users in Keycloak (see instruction) or configure SSO (see instruction), assign them `developer` or `admin` roles, and add a custom attributes named `applications` and `applications_admin` containing comma-separated project names to grant users access to specific AI/Run CodeMie projects. It's recommended to start with `demo,%%username%%` projects. For example:



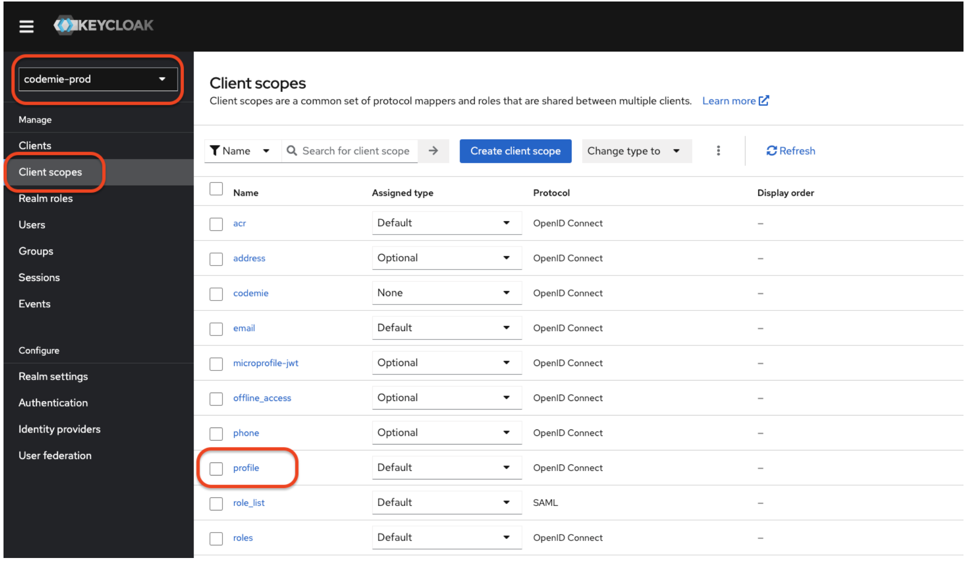
Key	Value
applications	demo

> When you assign a user access to a project that matches their Keycloak username (from the username claim), the system will automatically create this personal project in AI/Run CodeMie. Other projects must be created by AI/Run CodeMie admin.

Add user attributes to JWT token in Keycloak:

To include the added `applications` unmanaged attribute as an additional claim to the token it's necessary to configure protocol mappers. Follow the step:

1. Navigate to "Client Scopes" and update the client scope "profile" to include the newly added attribute.

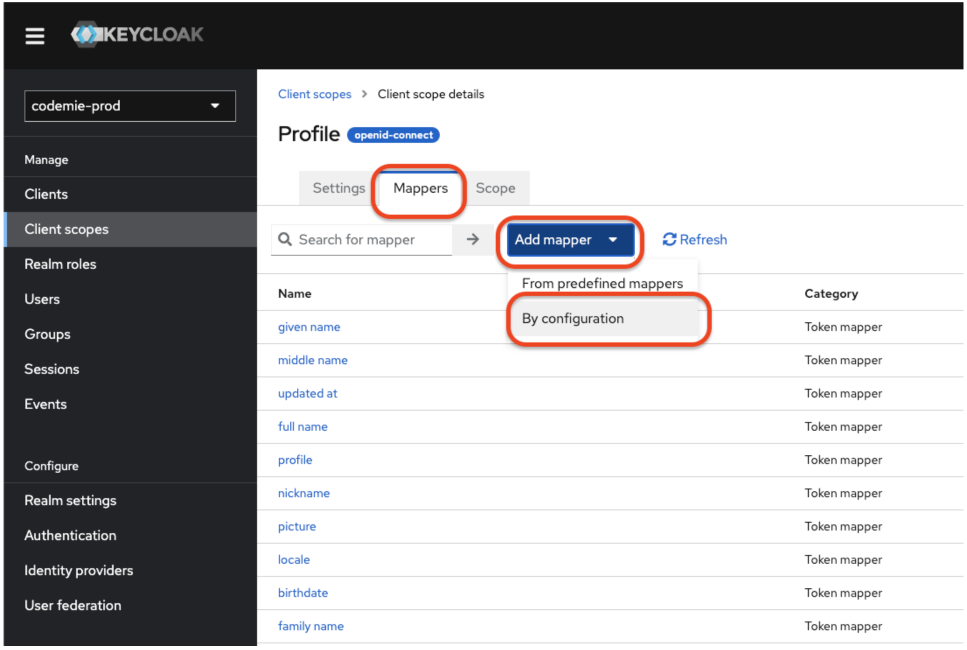


Client scopes

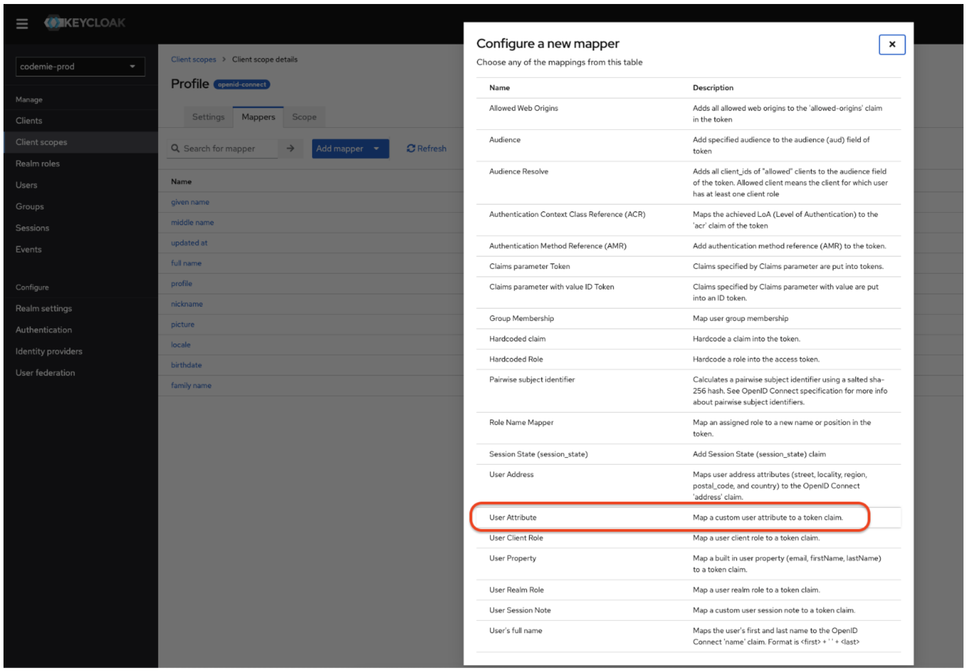
Client scopes are a common set of protocol mappers and roles that are shared between multiple clients. Learn more

	Name	Assigned type	Protocol	Display order
□	acr	Default	OpenID Connect	–
□	address	Optional	OpenID Connect	–
□	codemie	None	OpenID Connect	–
□	email	Default	OpenID Connect	–
□	microprofile-jwt	Optional	OpenID Connect	–
□	offline_access	Optional	OpenID Connect	–
□	phone	Optional	OpenID Connect	–
□	profile	Default	OpenID Connect	–
□	role_list	Default	SAML	–
□	roles	Default	OpenID Connect	–

2. Configure a mapper, selecting the mapping type as "User Attribute", then set `applications` as the field name, user attribute, and token claim name. Finally, save the changes.



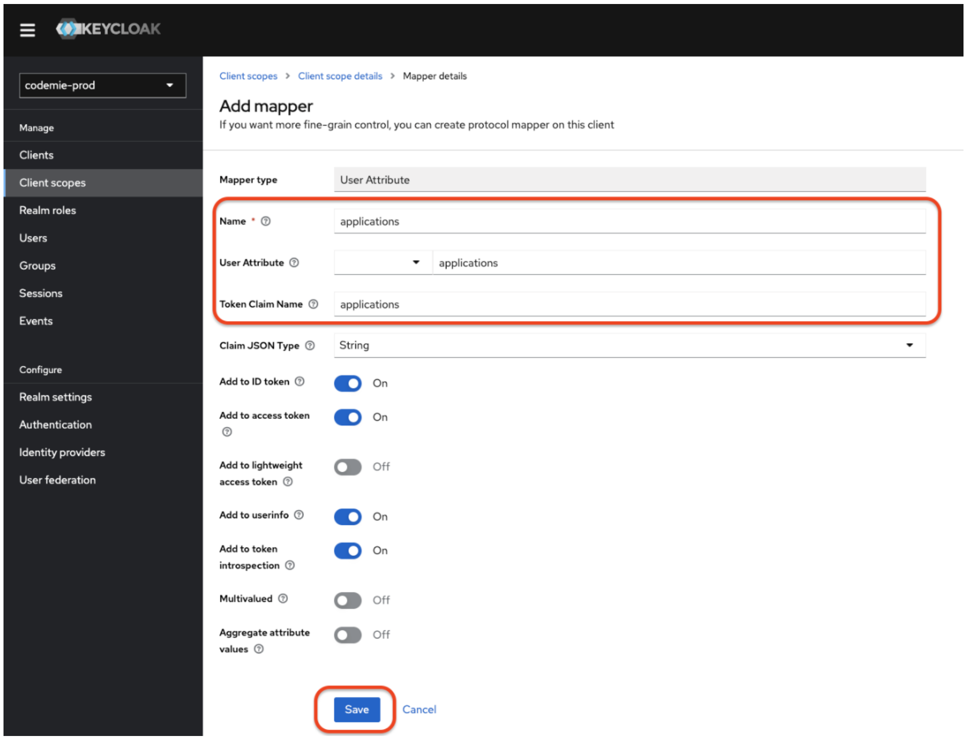
Name	Category
given name	Token mapper
middle name	Token mapper
updated at	Token mapper
full name	Token mapper
profile	Token mapper
nickname	Token mapper
picture	Token mapper
locale	Token mapper
birthdate	Token mapper
family name	Token mapper



Configure a new mapper

Choose any of the mappings from this table

Name	Description
Allowed Web Origins	Adds all allowed web origins to the 'allowed-origins' claim in the token
Audience	Add specified audience to the audience (aud) field of token
Audience Resolve	Adds all client_ids of "allowed" clients to the audience field of the token. Allowed client means the client for which user has at least one client role
Authentication Context Class Reference (ACR)	Maps the achieved LoA (Level of Authentication) to the 'acr' claim of the token
Authentication Method Reference (AMR)	Add authentication method reference (AMR) to the token.
Claims parameter Token	Claims specified by Claims parameter are put into tokens.
Claims parameter with value ID Token	Claims specified by Claims parameter with value are put into an ID token.
Group Membership	Map user group membership
Hardcoded claim	Hardcode a claim into the token.
Hardcoded Role	Hardcode a role into the access token.
Pairwise subject identifier	Calculates a pairwise subject identifier using a salted sha-256 hash. See OpenID Connect specification for more info about pairwise subject identifiers.
Role Name Mapper	Map an assigned role to a new name or position in the token.
Session State (session_state)	Add Session State (session_state) claim
User Address	Maps user address attributes (street, locality, region, postal_code, and country) to the OpenID Connect 'address' claim.
User Attribute	Map a custom user attribute to a token claim.
User Client Role	Map a user client role to a token claim.
User Property	Map a built in user property (email, firstName, lastName) to a token claim.
User Realm Role	Map a user realm role to a token claim.
User Session Note	Map a custom user session note to a token claim.
User's full name	Maps the user's first and last name to the OpenID Connect 'name' claim. Format is <first> + ' ' + <last>

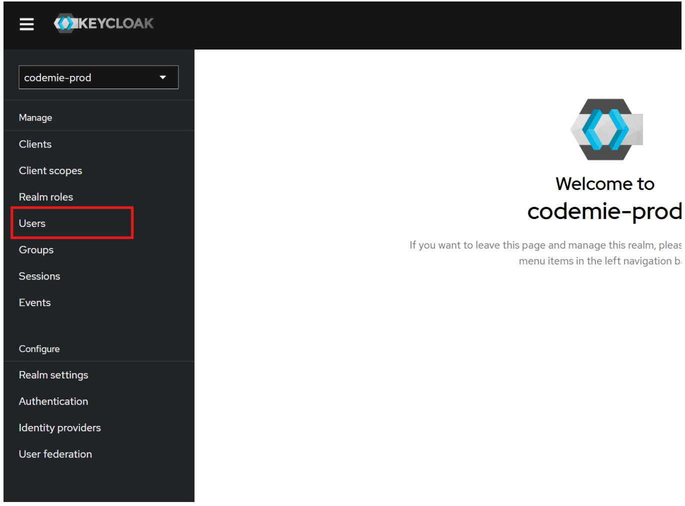


Add mapper

If you want more fine-grain control, you can create protocol mapper on this client

Mapper type	User Attribute
Name *	applications
User Attribute	applications
Token Claim Name	applications
Claim JSON Type	String
Add to ID token	On
Add to access token	On
Add to lightweight access token	Off
Add to userinfo	On
Add to token introspection	On
Multivalued	Off
Aggregate attribute values	Off

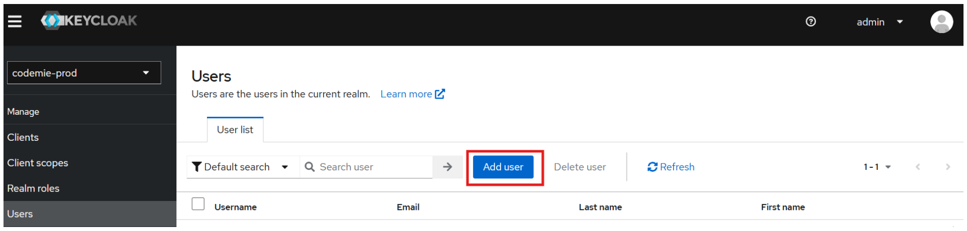
3. Open Users list page



Welcome to
codemie-prod

If you want to leave this page and manage this realm, pleas
menu items in the left navigation b

4. Click on "Add user" button

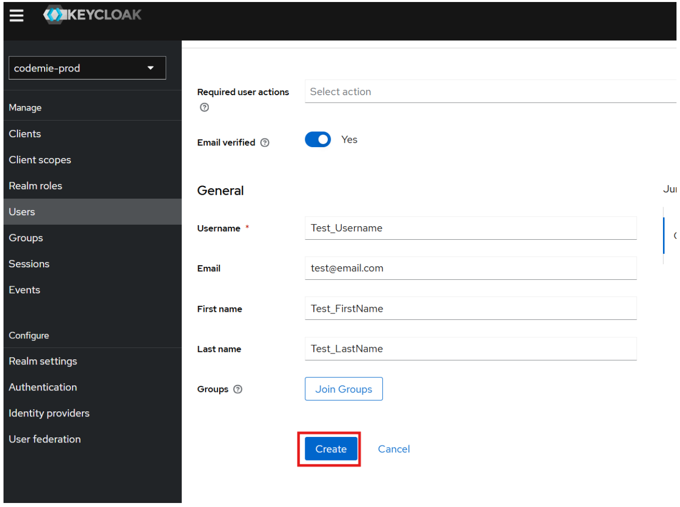


Users

Users are the users in the current realm. Learn more

Username	Email	Last name	First name

5. Fill all necessary fields and click on "Create" button



Required user actions: Select action

Email verified: Yes

General

Username *: Test_Username

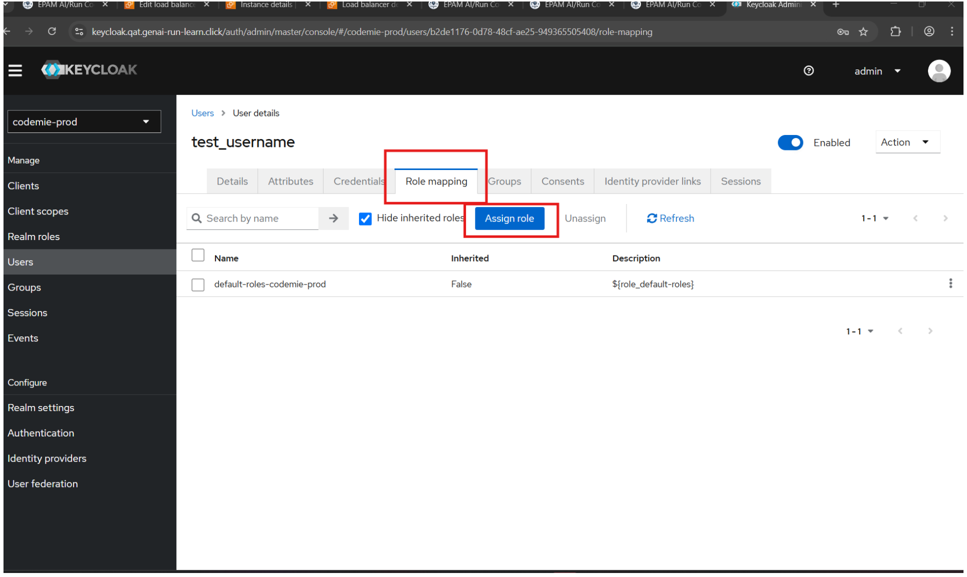
Email: test@email.com

First name: Test_FirstName

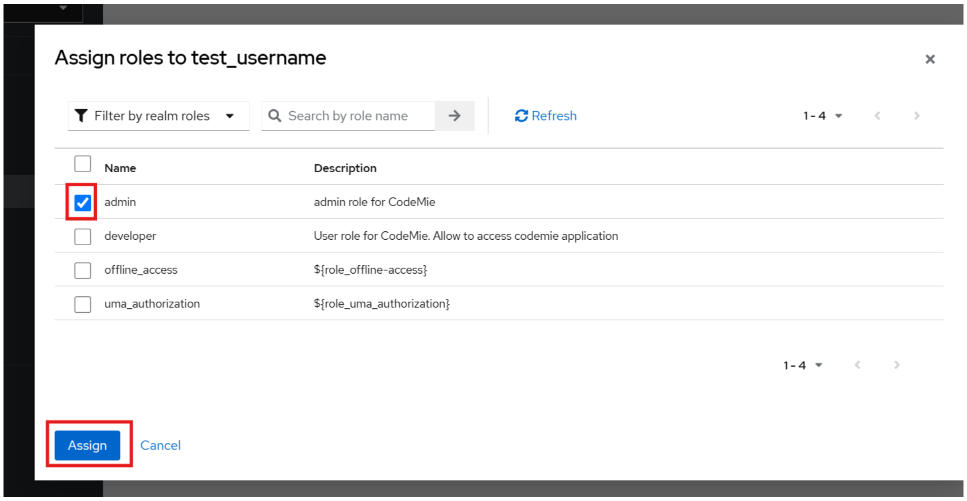
Last name: Test_LastName

Groups

6. Assign role

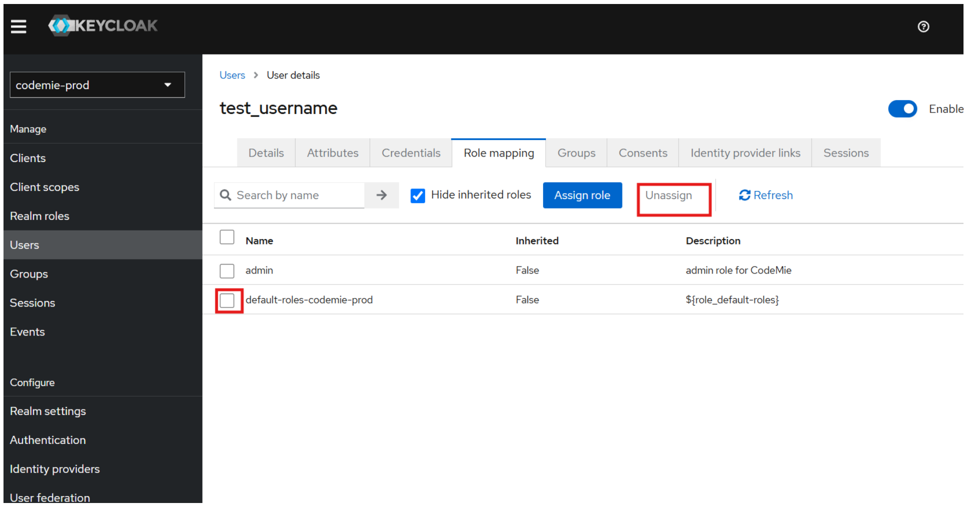


□ Name	Inherited	Description
□ default-roles-codemie-prod	False	${role_default-roles}



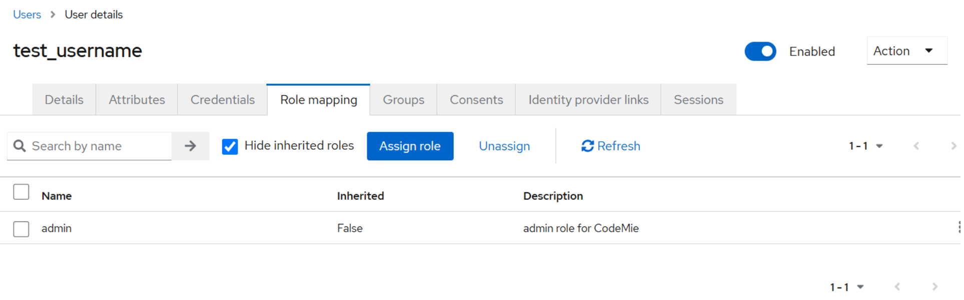
Assign roles to test_username

	Name	Description
☑	admin	admin role for CodeMie
□	developer	User role for CodeMie. Allow to access codemie application
□	offline_access	${role_offline-access}
□	uma_authorization	${role_uma_authorization}



test_username

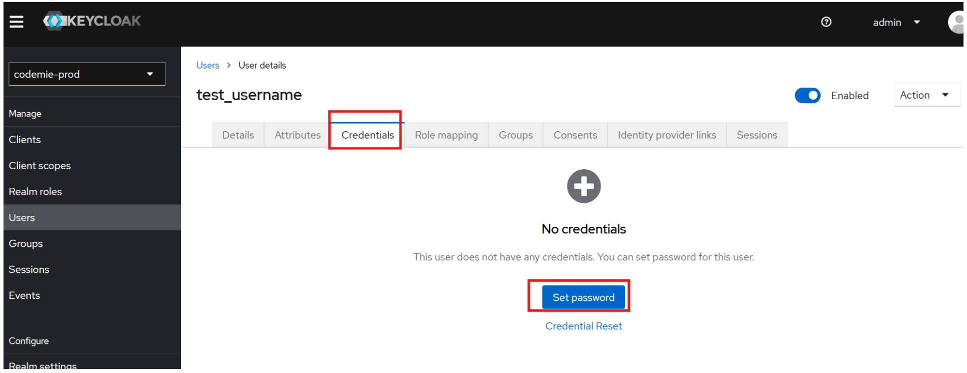
□	Name	Inherited	Description
□	admin	False	admin role for CodeMie
□	default-roles-codemie-prod	False	${role_default-roles}



test_username

□	Name	Inherited	Description
□	admin	False	admin role for CodeMie

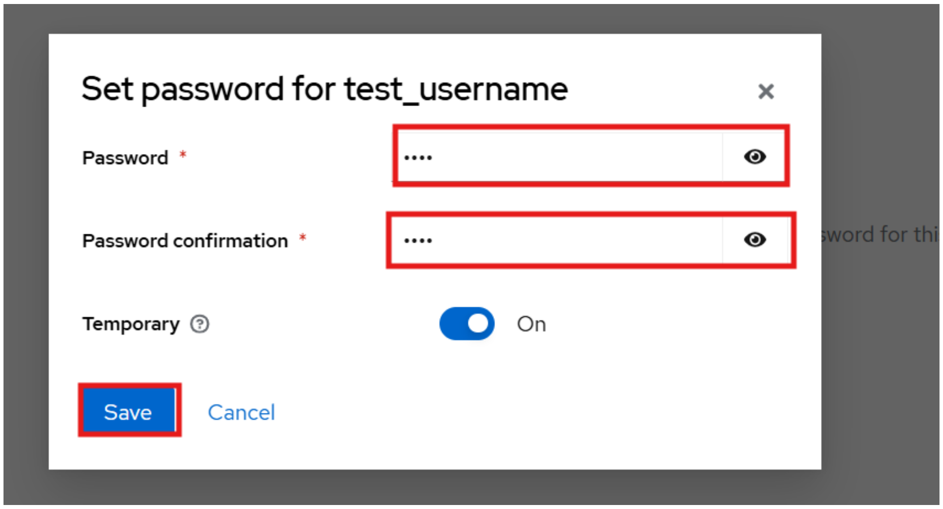
7. Set up credential



test_username

No credentials

This user does not have any credentials. You can set password for this user.



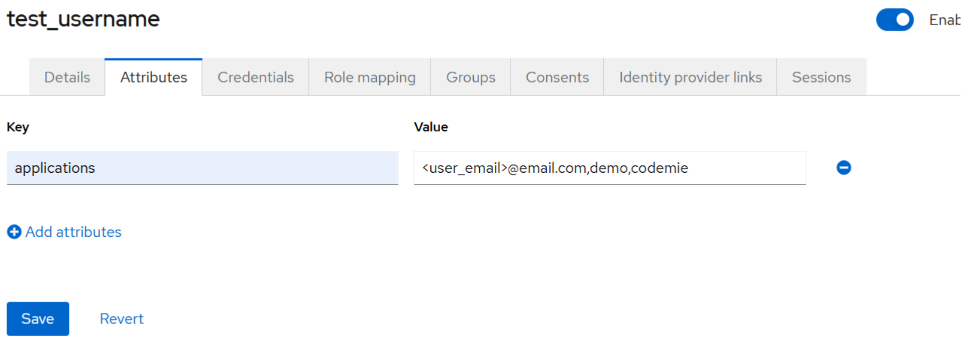
Set password for test_username

Password * ••••

Password confirmation * ••••

Temporary On

8. Set up attributes



test_username

Key	Value
applications	<user_email>@email.com,demo,codemie

9. Verify login and access to AI/Run CodeMie application.

7.1.2. UI custom configuration (optional)

To meet customer requirements, some of the UI elements on the AI/Run CodeMie can be hidden or changed via customer config. The `customer-config.yaml` file follows a YAML format with a specific structure:

```
components:
  - id: "componentId"
    settings:
      name: "Component Display Name"
      enabled: true|false
      url: "https://example.com/resource"
```

Below are the standard components that can be configured or hidden/enabled in AI/Run CodeMie:

7.1.2.1. Video Portal

Provides links to tutorial videos for users.

```
- id: "videoPortal"
  settings:
    name: "Video Portal"
    enabled: true
    url: "https://example-video-portal.com"
```

Setting	Type	Required	Description
name	string	Yes	Display name for the video portal link
enabled	boolean	Yes	Set to `true` to show the video portal link, `false` to hide it
url	string	Yes	URL to your video tutorial content

7.1.2.2. User Guide

Provides a link to user documentation.

```
- id: "userGuide"
  settings:
    name: "User Guide"
    enabled: true
    url: "https://example-tutorial-portal.com"
```

Setting	Type	Required	Description
name	string	Yes	Display name for the user guide link
enabled	boolean	Yes	Set to `true` to show the user guide link, `false` to hide it
url	string	Yes	URL to your user documentation

7.1.2.3. Admin Actions

Controls whether administrative actions are available.

```
- id: "adminActions"
  settings:
    enabled: true
```

Setting	Type	Required	Description
enabled	boolean	Yes	Set to `true` to enable admin actions, `false` to disable them

7.1.2.4. Feedback Assistant

Controls whether the feedback feature is available.

```
- id: "feedbackAssistant"
  settings:
    enabled: true
```

Setting	Type	Required	Description
enabled	boolean	Yes	Set to `true` to enable the feedback assistant, `false` to disable it

7.1.2.5. Workflow Documentation

Provides a link to workflow-specific documentation.

```
- id: "workflowDocumentation"
  settings:
    name: "Workflow Documentation"
    enabled: true
    url: "https://example-documentation.com"
```

Setting	Type	Required	Description
name	string	Yes	Display name for the workflow documentation link
enabled	boolean	Yes	Set to `true` to show the workflow documentation link, `false` to hide it
url	string	Yes	URL to your workflow documentation

7.1.2.6. Configuration

To configure it add the following blocks with specific for you configuration to the `codemie-helm-charts/codemie-api/values.yaml`:

```
extraObjects:
  - apiVersion: v1
    kind: ConfigMap
    metadata:
      name: customer-config
    data:
      customer-config.yaml: |
        ---
        components:
          - id: "videoPortal"
            settings:
              name: "Video Portal"
              enabled: false
              url: "https://example-video-portal.com"
          - id: "userGuide"
            settings:
              name: "User Guide"
              enabled: false
              url: "https://example-tutorial-portal.com"
          - id: "adminActions"
            settings:
              enabled: true
          - id: "feedbackAssistant"
            settings:
              enabled: false
          - id: "workflowDocumentation"
            settings:
              name: "Workflow Documentation"
              enabled: false
              url: "https://example-documentation.com"
```

```
extraVolumes: |
...
  - name: customer-config
    configMap:
      name: customer-config
...
```

```
extraVolumeMounts: |
...
  - name: customer-config
    mountPath: /app/config/customer/customer-config.yaml
    subPath: customer-config.yaml
...
```

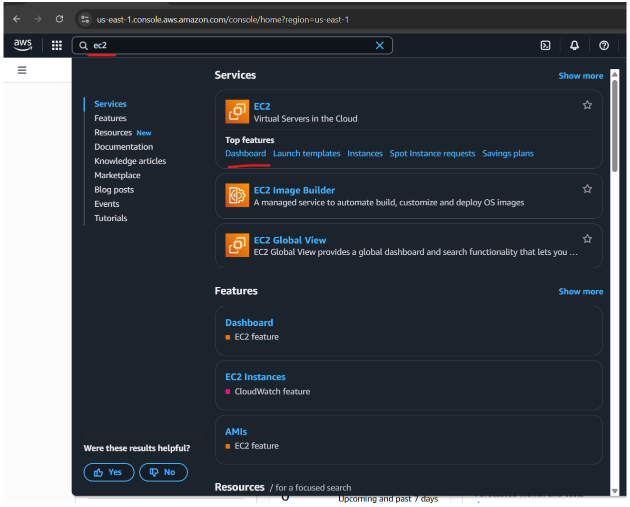
and apply helm chart with the command:

```
helm upgrade --install codemie-api oci://europe-west3-docker.pkg.dev/or2-msq-epmd-edp-anthos-t1iylu/helm-charts
/codemie \
--version x.y.z \
--namespace "codemie" \
-f "./codemie-api/values-aws.yaml" \
--wait --timeout 600s
```

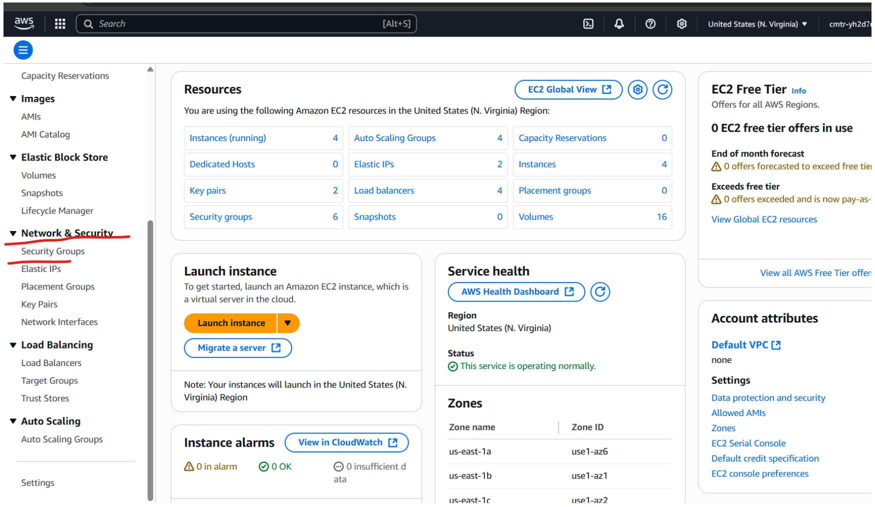
8. Provide access to the application from the Internet

8.1. Create new security group

8.1.1. Open EC2 service group



8.1.2. Open "Security Groups"

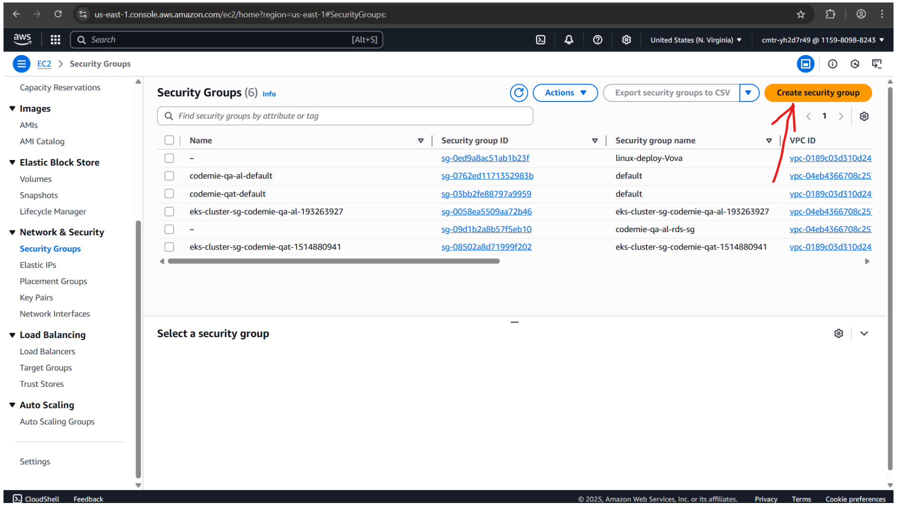


You are using the following Amazon EC2 resources in the United States (N. Virginia) Region:

Instances (running)	4	Auto Scaling Groups	4	Capacity Reservations	0
Dedicated Hosts	0	Elastic IPs	2	Instances	4
Key pairs	2	Load balancers	4	Placement groups	0
Security groups	6	Snapshots	0	Volumes	16

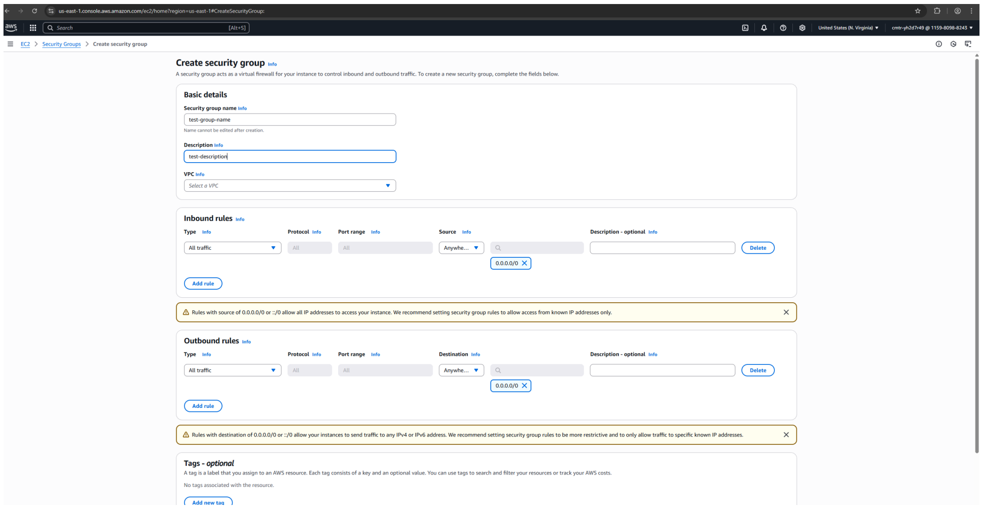
Launch instance
To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.
Launch instance
Migrate a server
Note: Your instances will launch in the United States (N. Virginia) Region
Instance alarms
View in CloudWatch
0 in alarm
0 OK
0 insufficient data
Service health
AWS Health Dashboard
Region
United States (N. Virginia)
Status
This service is operating normally.
Zones
Zone name
Zone ID
us-east-1a
use1-az6
us-east-1b
use1-az1
us-east-1c
use1-az2
EC2 Free Tier Info
Offers for all AWS Regions.
0 EC2 free tier offers in use
End of month forecast
0 offers forecasted to exceed free tie
Exceeds free tier
0 offers exceeded and is now pay-as-
View Global EC2 resources
View all AWS Free Tier offer
Account attributes
Default VPC
none
Settings
Data protection and security
Allowed AMIs
Zones
EC2 Serial Console
Default credit specification
EC2 console preferences

8.1.3. Create new "Security Groups""



Security Groups (6)

Name	Security group ID	Security group name	VPC ID
–	sg-0ed9a8ac51ab1b23f	linux-deploy-Vova	vpc-0189c03d310d24
codemie-qa-al-default	sg-0762ed1171352983b	default	vpc-04eb4366708c25
codemie-qat-default	sg-03bb2fe88797a9959	default	vpc-0189c03d310d24
eks-cluster-sg-codemie-qa-al-193263927	sg-0058ea5509aa72b46	eks-cluster-sg-codemie-qa-al-193263927	vpc-04eb4366708c25
–	sg-09d1b2a8b57f5eb10	codemie-qa-al-rds-sg	vpc-04eb4366708c25
eks-cluster-sg-codemie-qat-1514880941	sg-08502a8d71999f202	eks-cluster-sg-codemie-qat-1514880941	vpc-0189c03d310d24



Create security group

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

Security group name: test-group-name

Name cannot be edited after creation.

Description: test-description

VPC: Select a VPC

Inbound rules

Type	Protocol	Port range	Source		Description - optional
All traffic	All	All	Anywhe...	0.0.0.0/0	

Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Outbound rules

Type	Protocol	Port range	Destination		Description - optional
All traffic	All	All	Anywhe...	0.0.0.0/0	

Rules with destination of 0.0.0.0/0 or ::/0 allow your instances to send traffic to any IPv4 or IPv6 address. We recommend setting security group rules to be more restrictive and to only allow traffic to specific known IP addresses.

Tags - *optional*

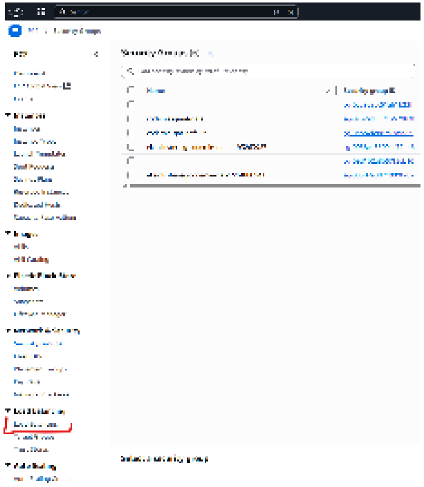
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

No tags associated with the resource.

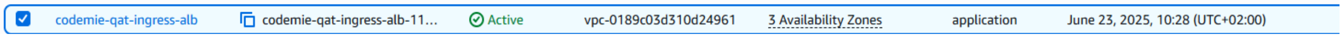
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateSecurityGroup:
EC2 > Security Groups > Create security group
Delete
Add rule
Delete
Add rule
Add new tag

8.2. Add security group to Load Balancers

8.2.1. Open Load Balancers

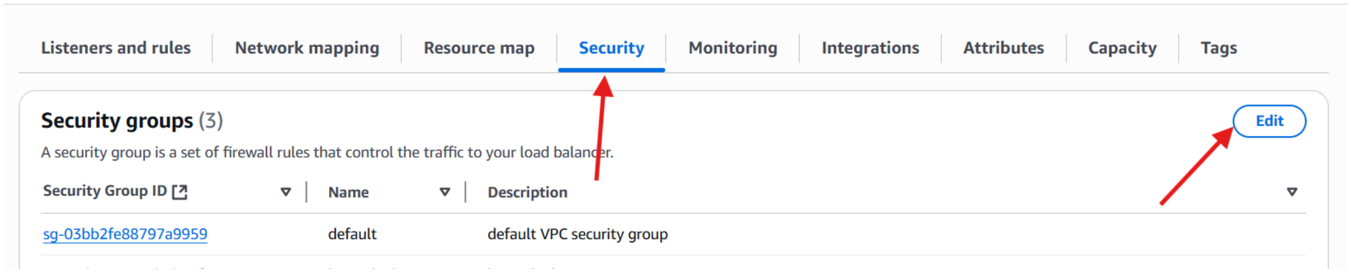


8.2.2. Find and open <some name>-ingress-alb balancer to cluster which was created



☑	Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
☑	codemie-qat-ingress-alb	codemie-qat-ingress-alb-11…	Active	vpc-0189c03d310d24961	3 Availability Zones	application	June 23, 2025, 10:28 (UTC+02:00)

8.2.3. Navigate to security tab and click "edit" button

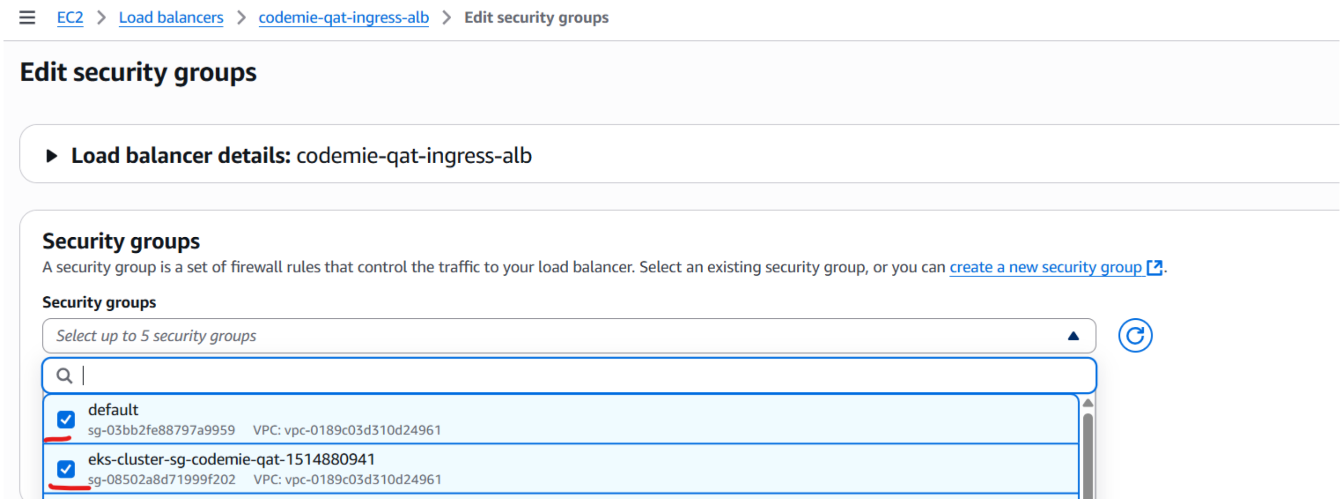


Security groups (3)

A security group is a set of firewall rules that control the traffic to your load balancer.

Security Group ID	Name	Description
sg-03bb2fe88797a9959	default	default VPC security group

8.2.4. Add new security group and save changes



EC2 > Load balancers > codemie-qat-ingress-alb > Edit security groups

Edit security groups

▶ **Load balancer details:** codemie-qat-ingress-alb

Security groups

A security group is a set of firewall rules that control the traffic to your load balancer. Select an existing security group, or you can create a new security group.

Security groups

Select up to 5 security groups

☑ default
sg-03bb2fe88797a9959 VPC: vpc-0189c03d310d24961

☑ eks-cluster-sg-codemie-qat-1514880941
sg-08502a8d71999f202 VPC: vpc-0189c03d310d24961