Archetype AI
AWS Marketplace Deployment Guide

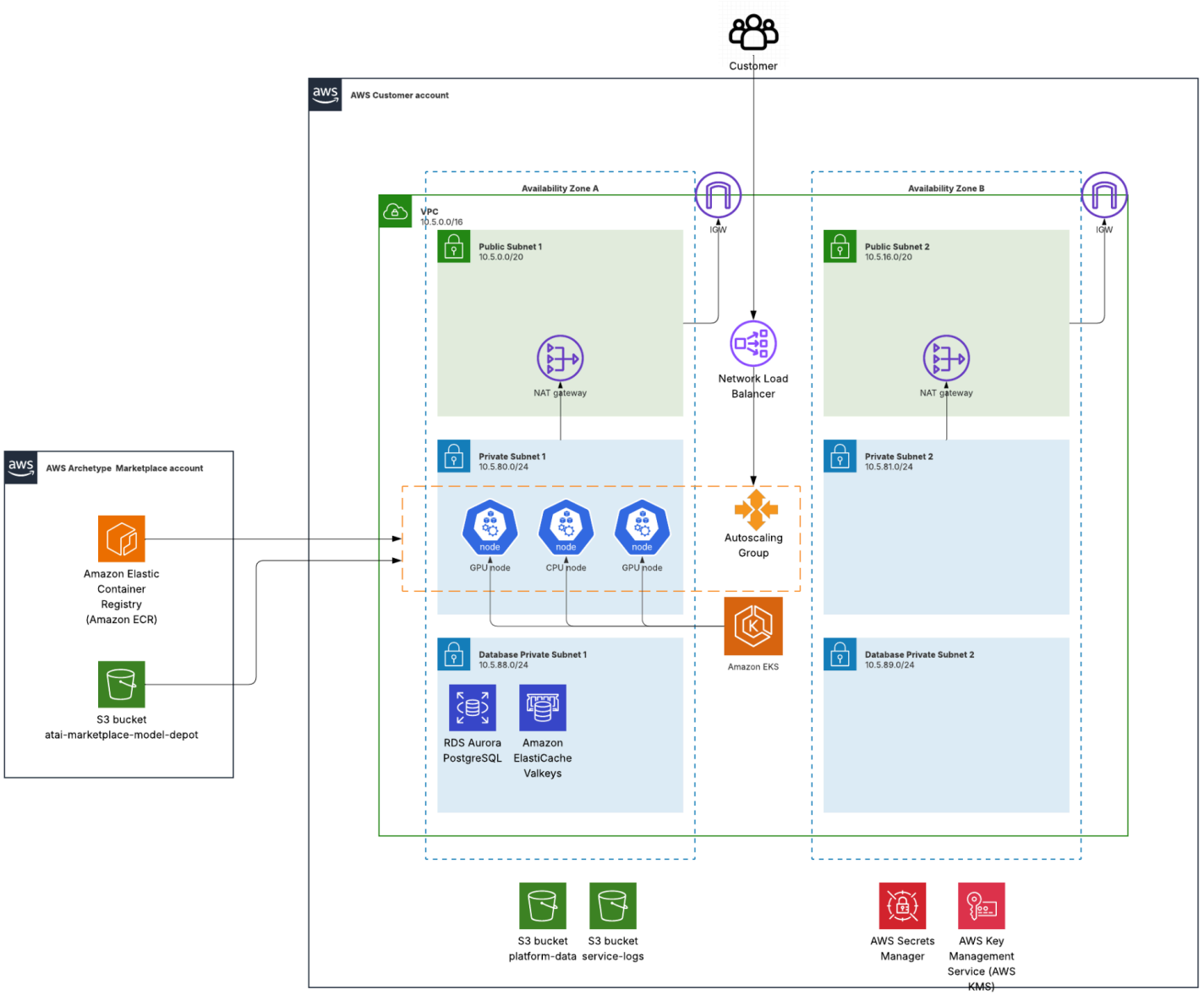
Version: 1.1.3

Executive summary

Archetype exists to unlock the potential of physical AI through Newton, a Large Behavior Model (LBM) that understands and reasons about the physical world using multimodal sensor data. Built on advanced machine learning techniques, Newton processes and interprets data from any sensor type while ensuring data privacy and enabling edge deployment. With a focus on the trillion-dollar sensor economy, Newton is positioned to transform industries such as construction and manufacturing by providing new insights into their physical operations.

Table of contents

Infrastructure Overview



atai Infrastructure

AWS License Manager setup

This Marketplace product uses **AWS License Manager** for proper operation. You **must enable and configure AWS License Manager** in your account before deployment.

Failure to do so will prevent the product from functioning correctly. For more information on how to set up License Manager correctly, see **Appendix: License Manager Setup**.

AWS infrastructure prerequisites

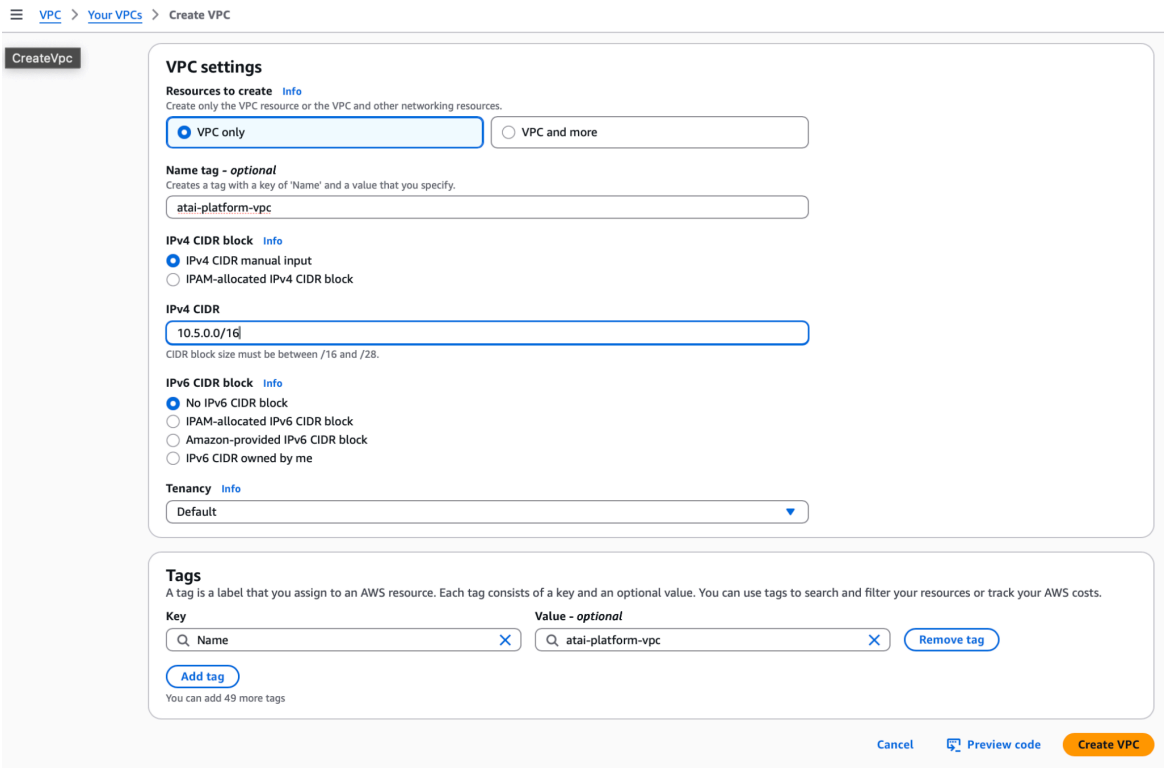
This document outlines the AWS infrastructure prerequisites that must be deployed before installing atai-platform helm chart.

VPC configuration

Create VPC with Public, Private, and Database Subnets

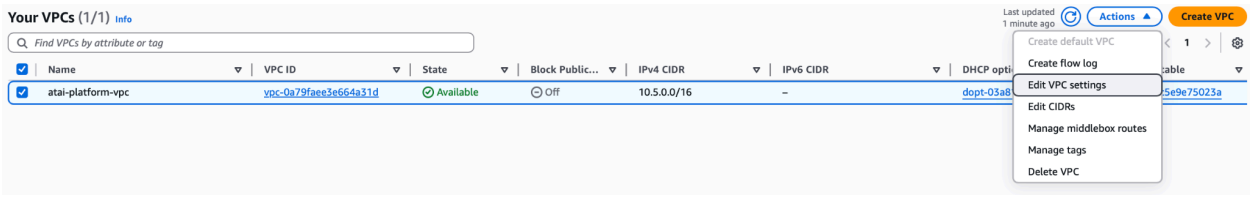
Step 1: Create the VPC (Manual Configuration)

1. Go to VPC Dashboard → Your VPCs → Create VPC
2. Select VPC only (not "VPC and more")
3. Configure:
    a. Name tag: atai-platform-vpc
    b. IPv4 CIDR block: 10.5.0.0/16
    c. IPv6 CIDR block: No IPv6 CIDR block
    d. Tenancy: Default
4. Click Create VPC

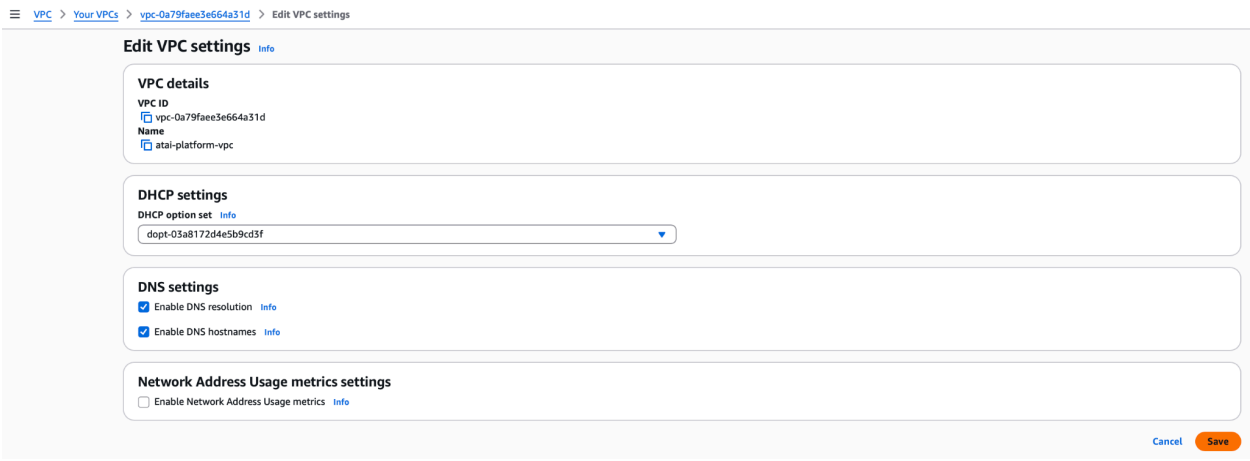


Step 1.1 Enable DNS Hostnames (if not already enabled)

1. Go to Your VPCs → select your VPC
2. Actions → Edit VPC settings

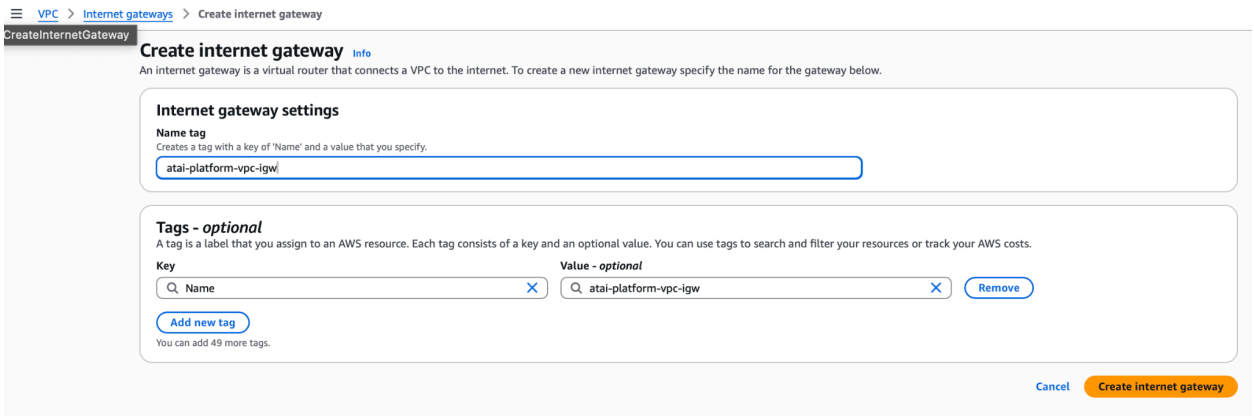


3. Enable DNS hostnames and DNS resolution
4. Save

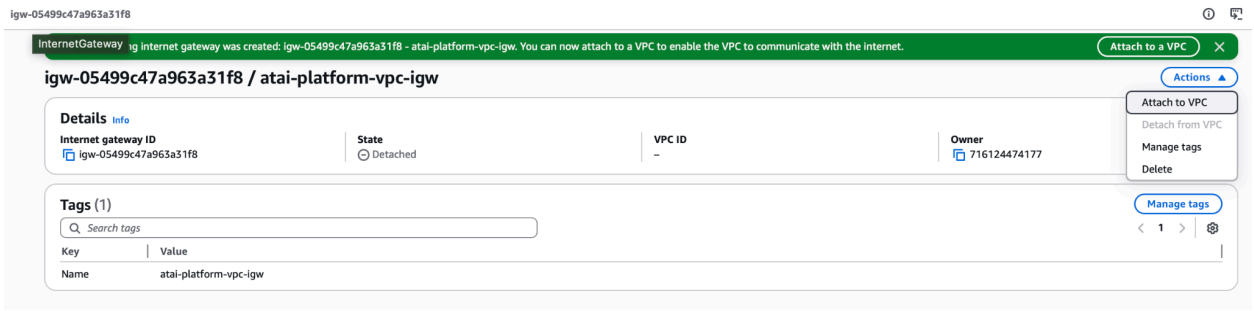


Step 2: Create Internet Gateway

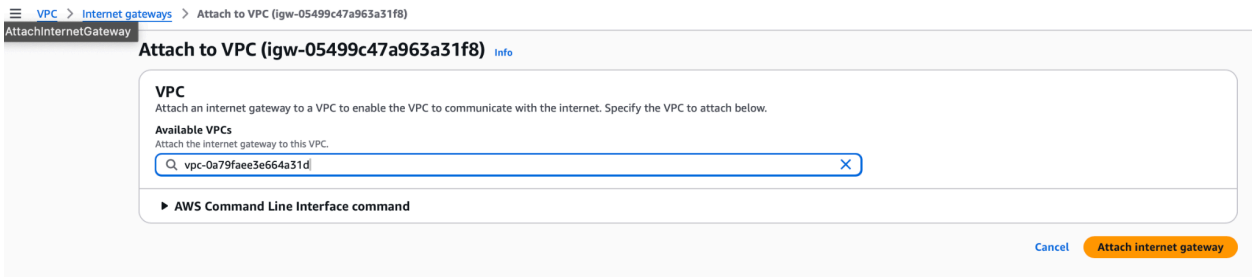
1. Go to VPC Dashboard → Go to Internet Gateways → Create internet gateway
2. Name tag: atai-platform-vpc-igw
3. Click Create internet gateway



4. Select it → Actions → Attach to VPC



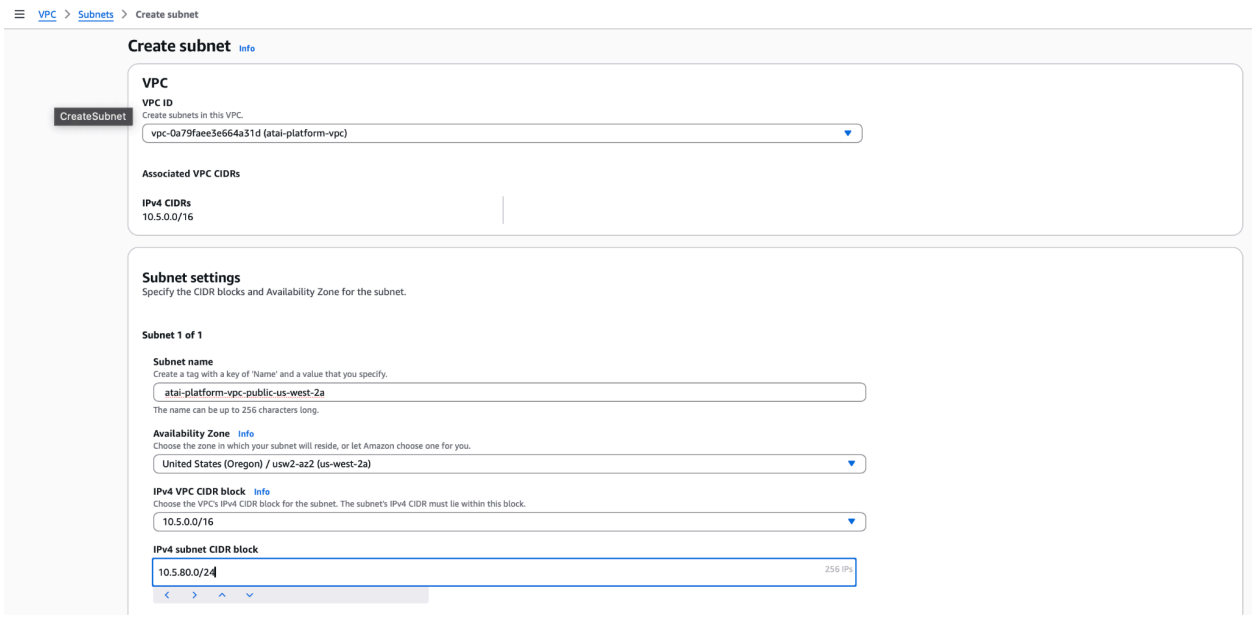
5. Select your VPC → Attach internet gateway

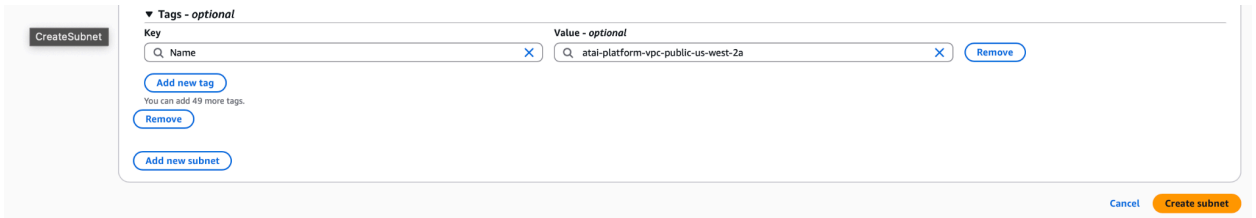


Step 3: Create Public Subnets (with correct CIDRs from the start)

1. Go to VPC Dashboard → Go to Subnets → Create subnet

2. Subnet 1:
    a. Select the VPC created in the Step 1
    b. VPC: Select your VPC
    c. Subnet name: atai-platform-vpc-public-us-west-2a
    d. Availability Zone: us-west-2a (or your AZ1)
    e. IPv4 CIDR block: 10.5.80.0/24
    f. Click Create subnet



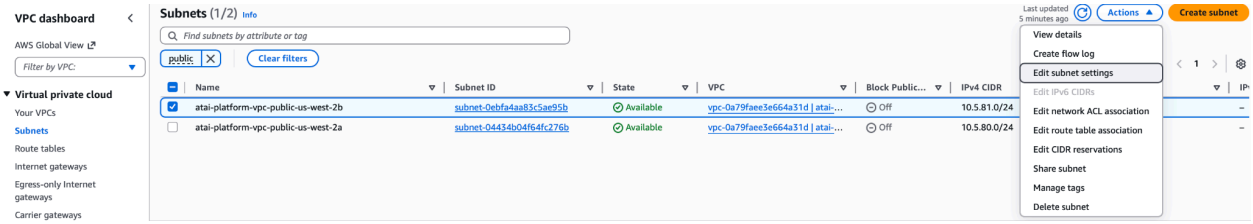


3. Subnet 2:
    a. Select the VPC created in the Step 1
    b. Subnet name: atai-platform-vpc-public-us-west-2b

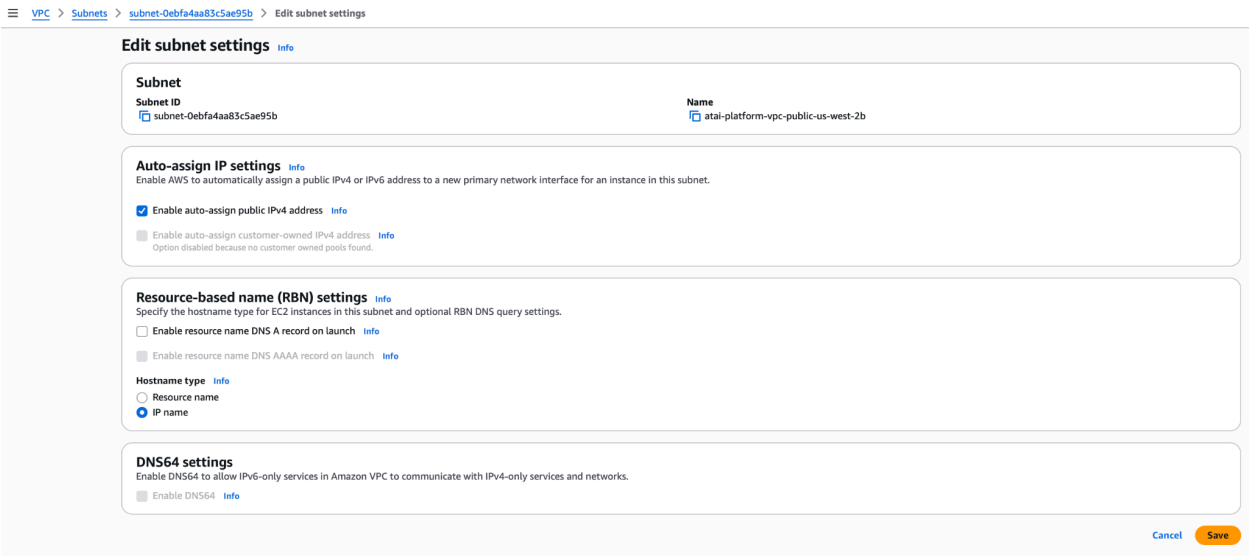
c. Availability Zone: us-west-2b (or your AZ2)
d. IPv4 CIDR block: 10.5.81.0/24
e. Click Create subnet

3.1 Enable Auto-assign Public IPv4 for Public Subnets

1. Go to VPC Dashboard → Go to Subnets → select each public subnet
2. Actions → Edit subnet settings



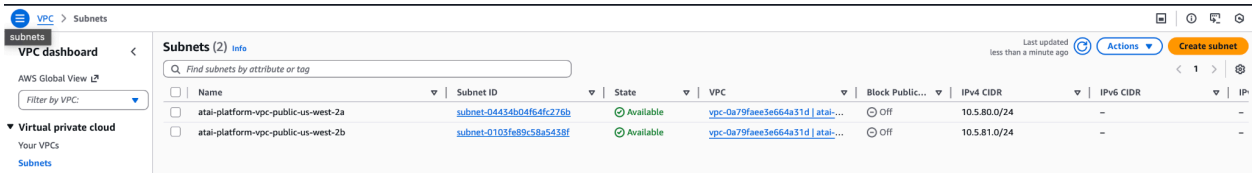
3. Check Enable auto-assign public IPv4 address
4. Save



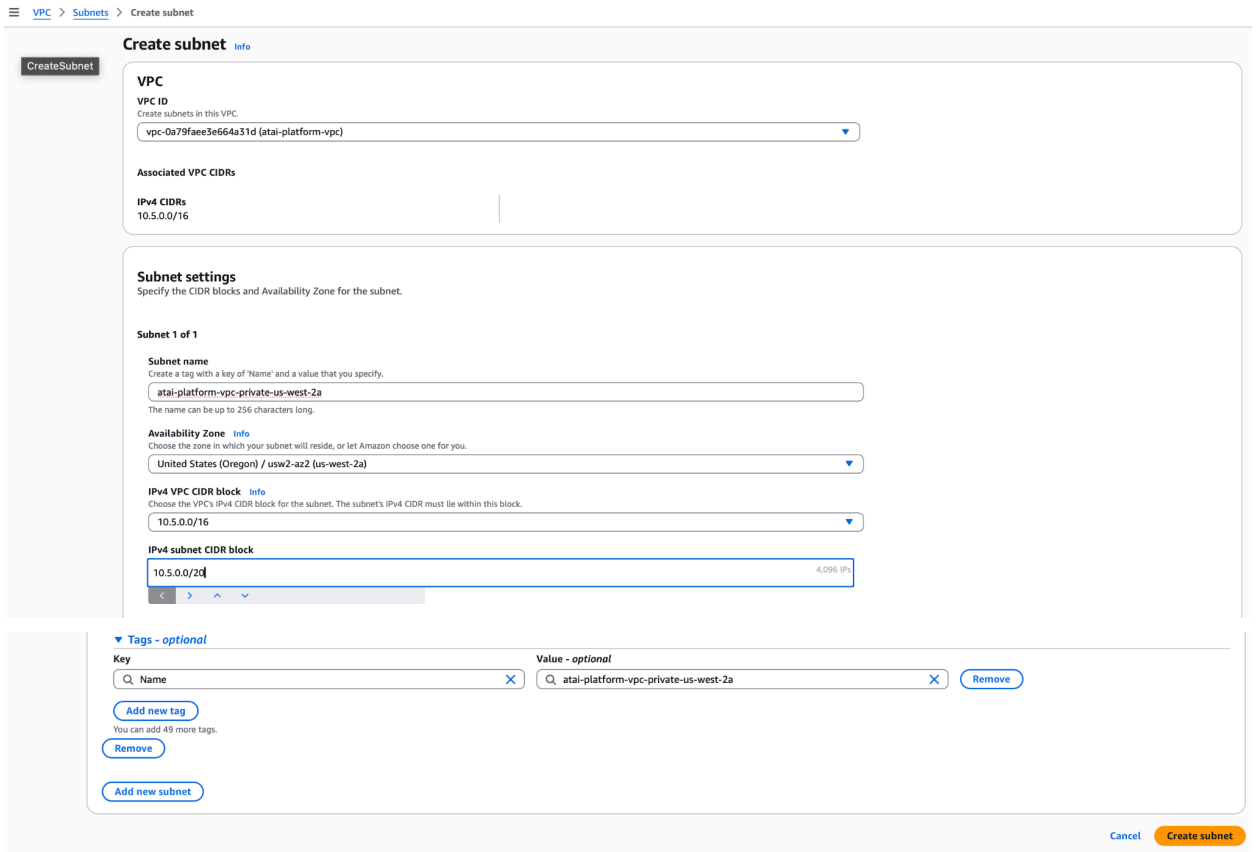
5. Repeat the process for both public subnets.

Step 4: Create Private Subnets

1. Go to VPC Dashboard → Go to Subnets → Create subnet



2. Subnet 1:
    a. Select the VPC created in the Step 1
    b. Name: atai-platform-vpc-private-us-west-2a
    c. Availability Zone: us-west-2a (or your AZ1)
    d. CIDR: 10.5.0.0/20
    e. Click Create subnet

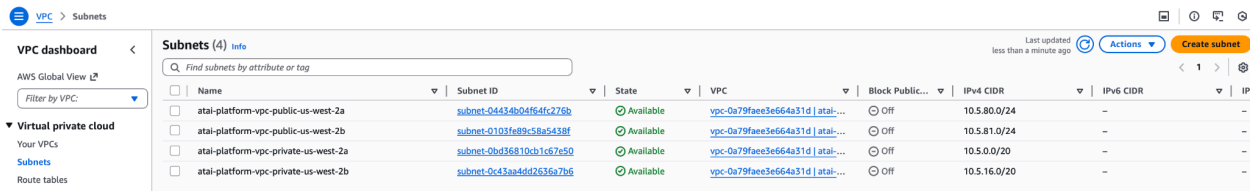


    f. Subnet 2:
    g. Select the VPC created in the Step 1
    h. Name: atai-platform-vpc-private-us-west-2b
    i. Availability Zone: us-west-2b (or your AZ2)
    j. CIDR: 10.5.16.0/20

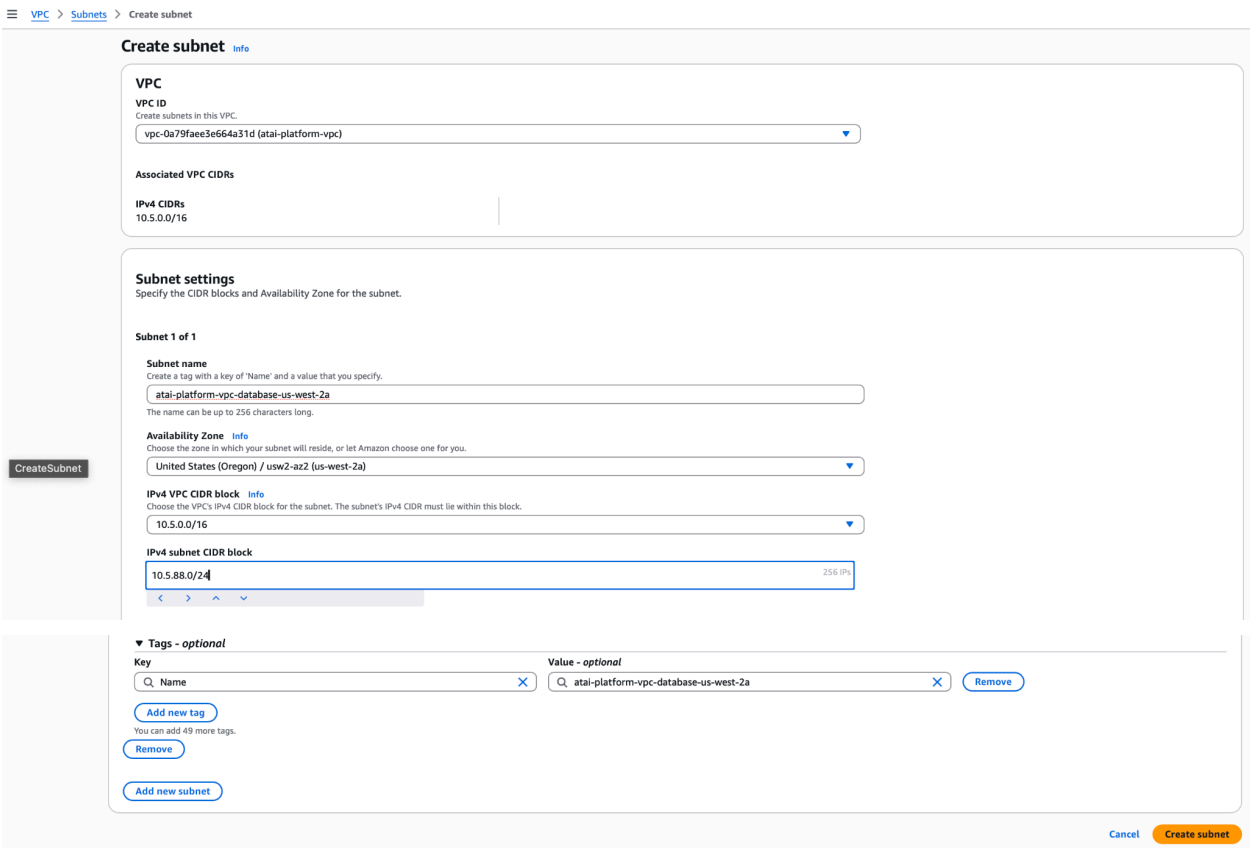
k. Click Create subnet

Step 5: Create Database Subnets

1. Go to VPC Dashboard → Go to Subnets → Create subnet



2. Subnet 1:
    a. Select the VPC created in the Step 1
    b. Name: atai-platform-vpc-database-us-west-2a
    c. Availability Zone: us-west-2a (or your AZ1)
    d. CIDR: 10.5.88.0/24
    e. Click Create subnet

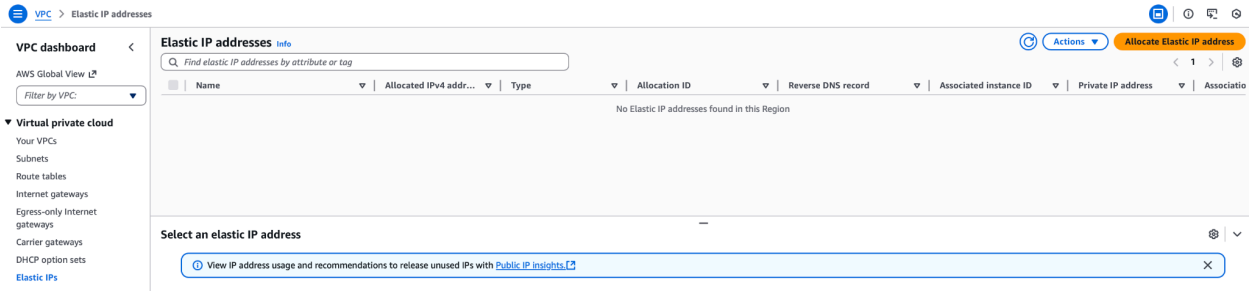


3. Subnet 2:

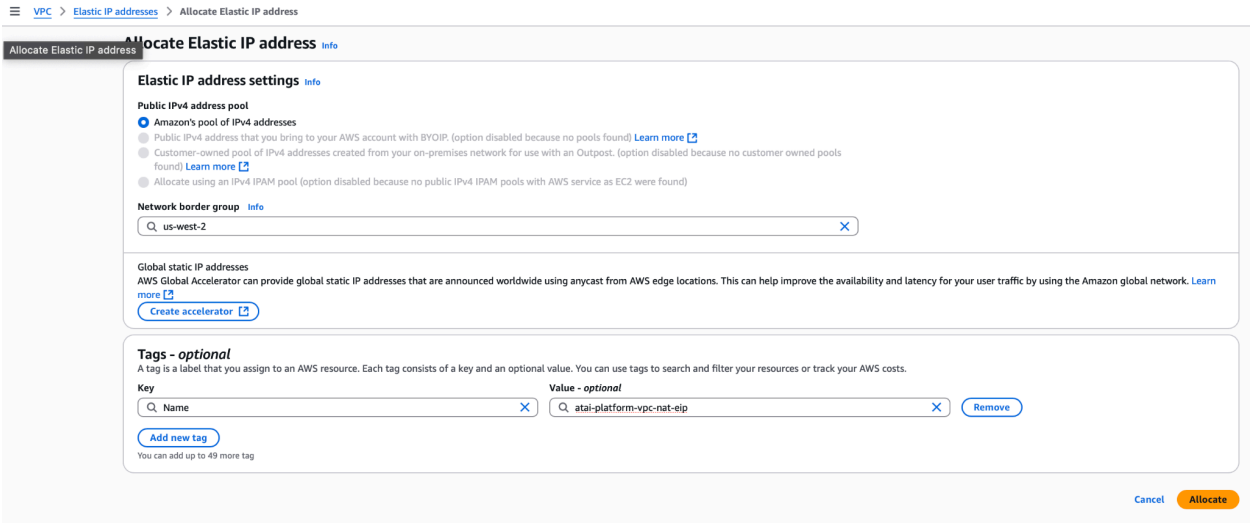
a. Select the VPC created in the Step 1
b. Name: atai-platform-vpc-database-us-west-2b
c. Availability Zone: us-west-2b (or your AZ2)
d. CIDR: 10.5.89.0/24
e. Click Create subnet

Step 6: Allocate Elastic IP for NAT Gateway

1. Go to VPC Dashboard → Go to Elastic IPs → Allocate Elastic IP address

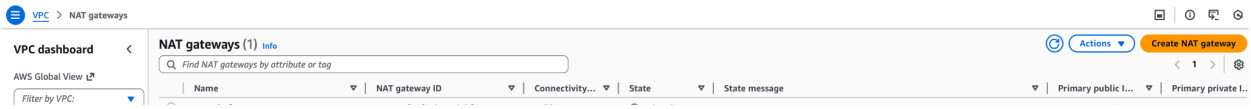


2. Network border group: Select your region
3. Public IPv4 address pool: Amazon's pool
4. Add Name tag: atai-platform-vpc-nat-eip
5. Click Allocate

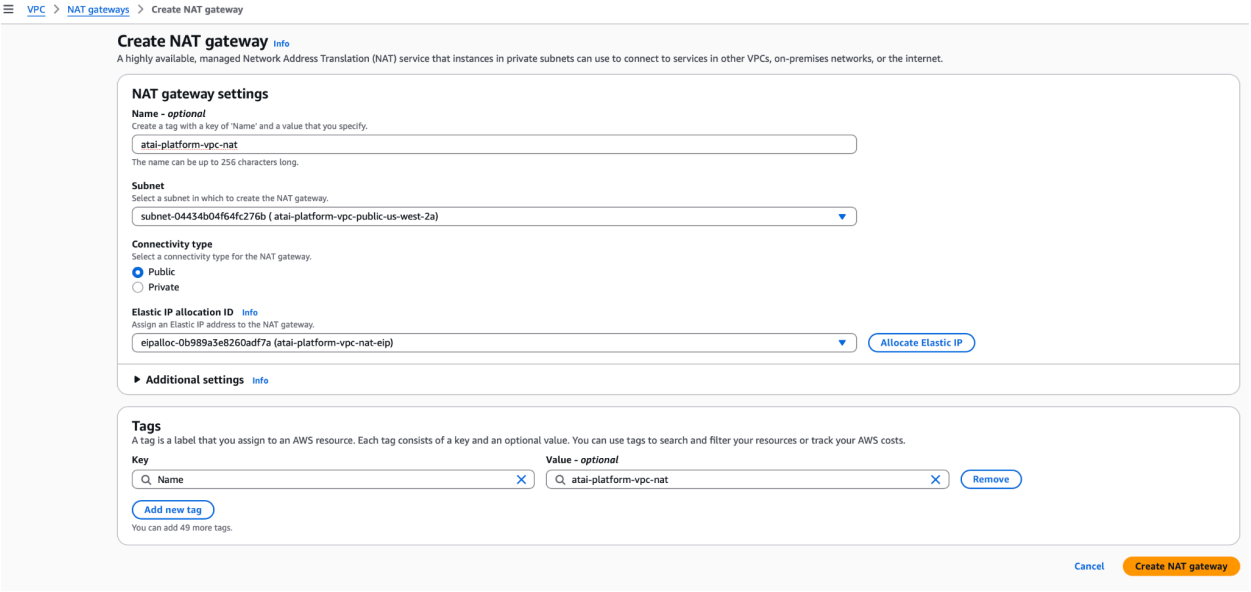


Step 7: Create NAT Gateway

1. Go to VPC Dashboard → Go to NAT Gateways → Create NAT gateway



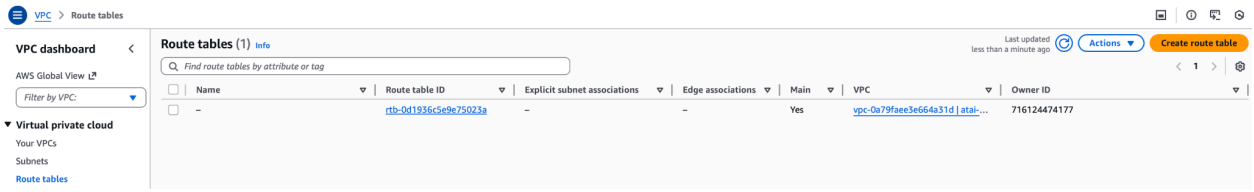
2. Name: atai-platform-vpc-nat
3. Subnet: Select first public subnet (10.5.80.0/24)
4. Elastic IP allocation ID: Select the EIP you just created in Step 6
5. Click Create NAT gateway (wait a few minutes)



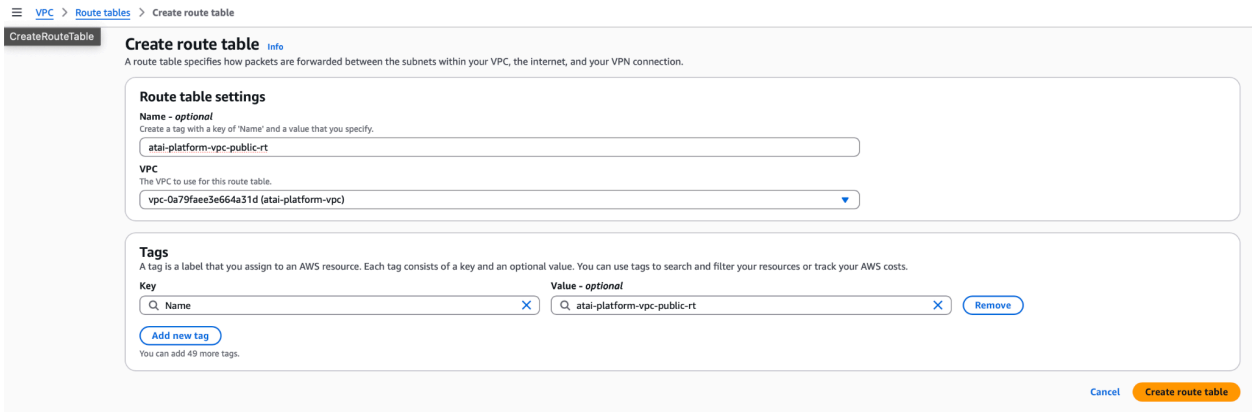
Step 8: Create and Configure Route Tables

Public Route Table:

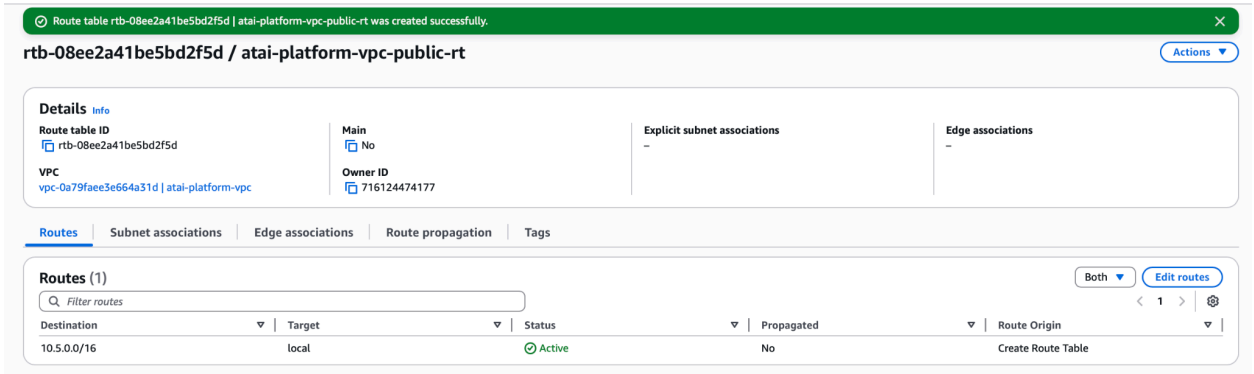
1. Go to VPC Dashboard → Route Tables → Create route table

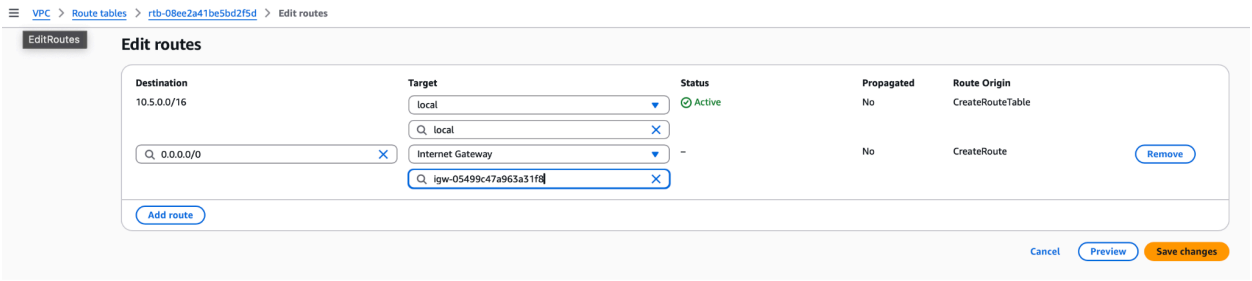


2. Name: atai-platform-vpc-public-rt
3. VPC: Select your VPC from Step 1
4. Click Create route table

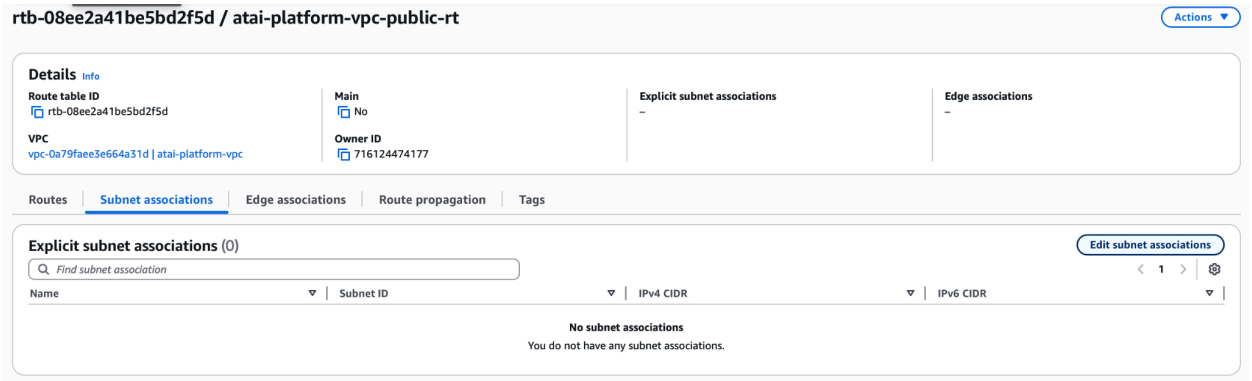


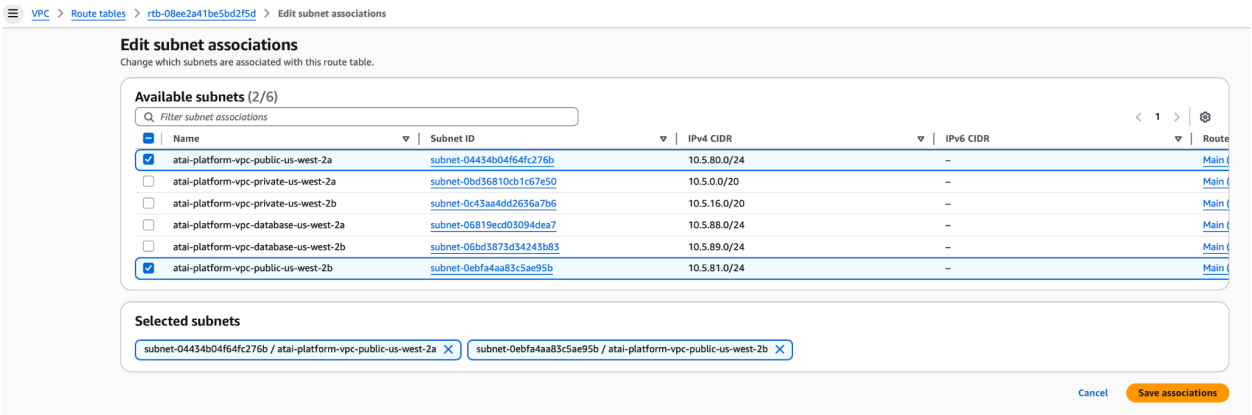
5. Routes → **Edit routes** → Add route:
    a. Destination: 0.0.0.0/0
    b. Target: Internet Gateway → select your IGW from Step 2
    c. Save changes





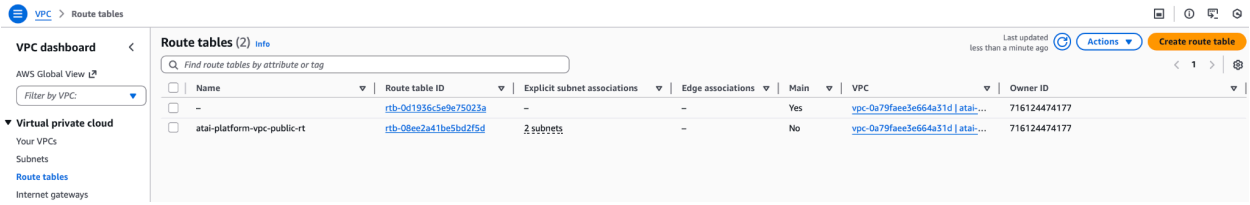
6. Subnet associations → **Edit subnet associations**:
    a. Select both public subnets → Save associations



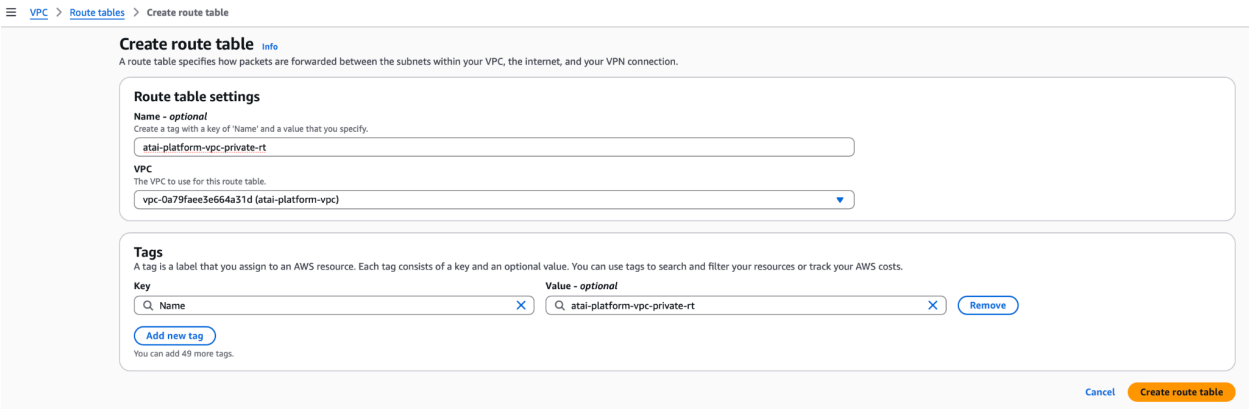


Private Route Table:

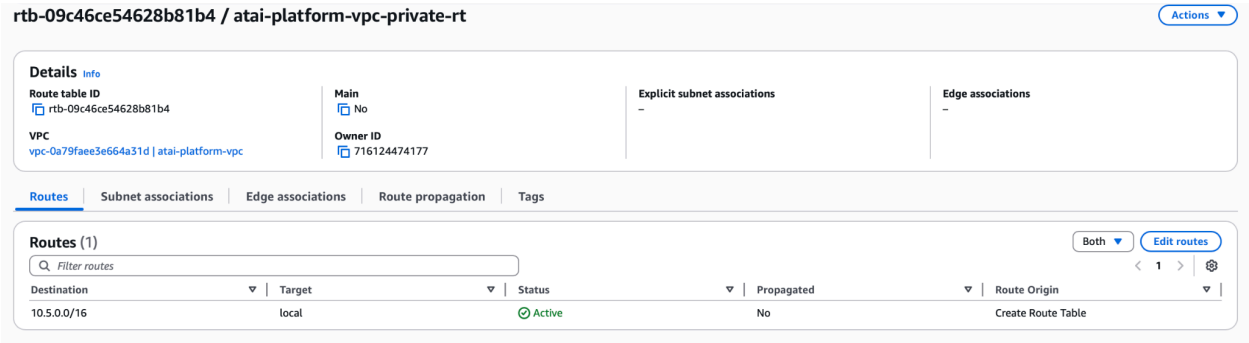
1. Go to VPC Dashboard → Route Tables → Create route table

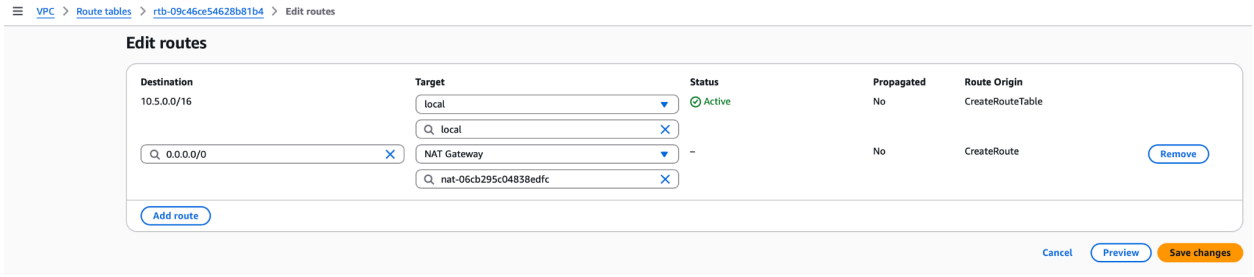


2. Name: atai-platform-vpc-private-rt
3. VPC: Select your VPC from Step 1
4. Click Create route table

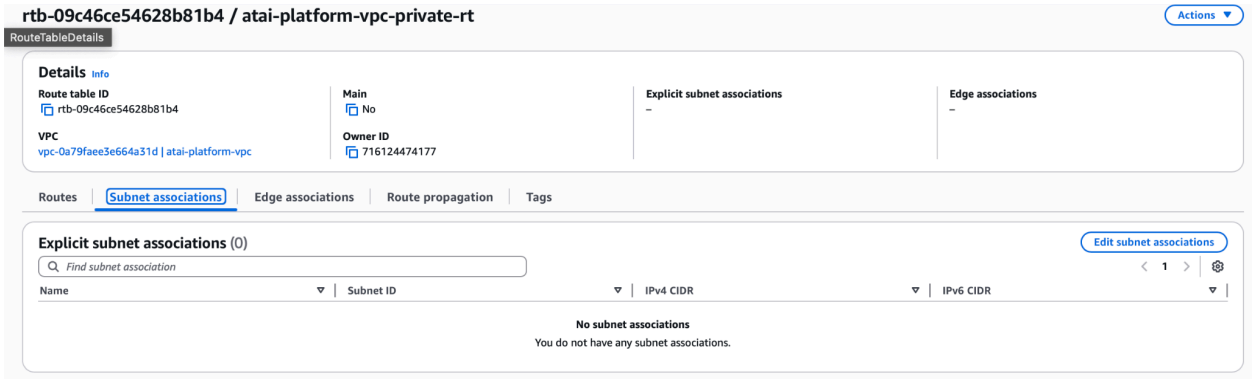


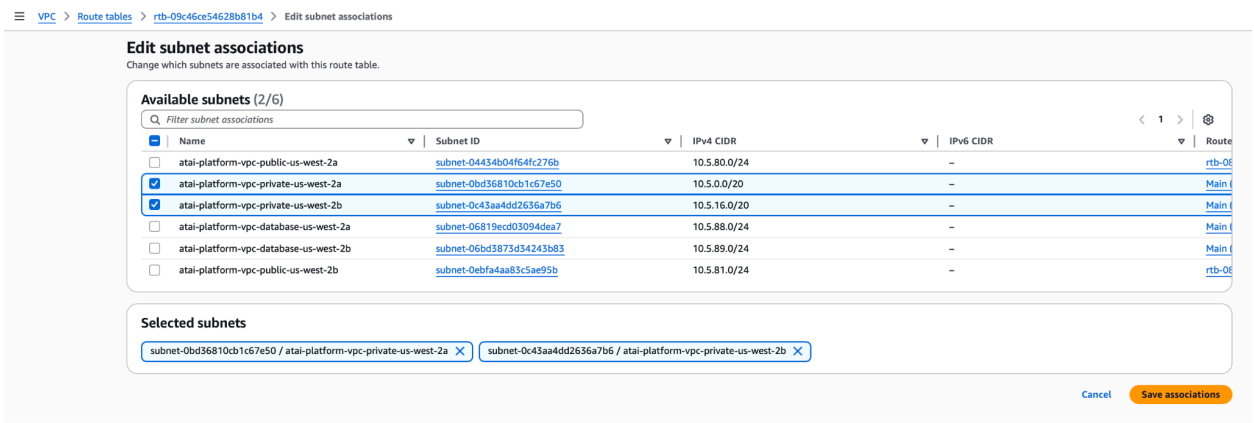
5. Routes → Edit routes → **Add route**:
    a. Destination: 0.0.0.0/0
    b. Target: NAT Gateway → select your NAT Gateway
    c. Save changes





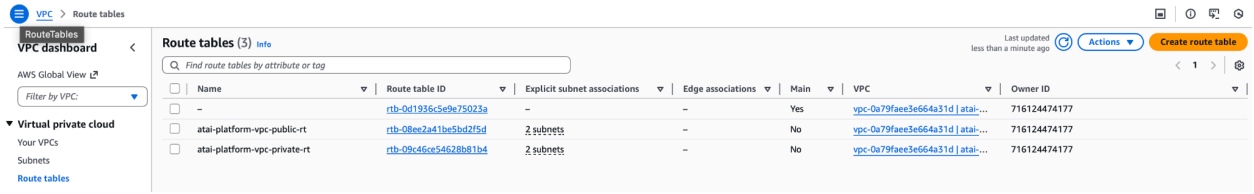
6. Subnet associations → Edit subnet associations:
    a. Select both private subnets → Save associations



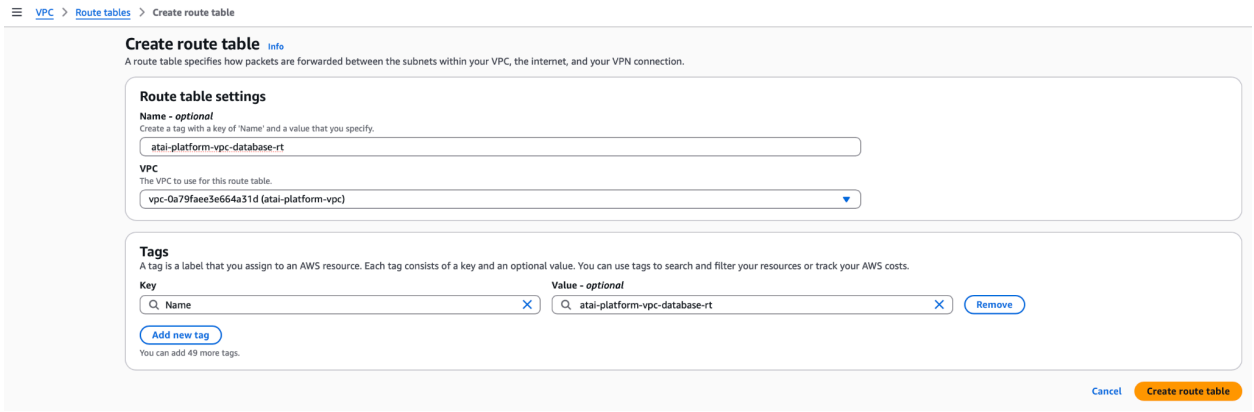


Database Route Table:

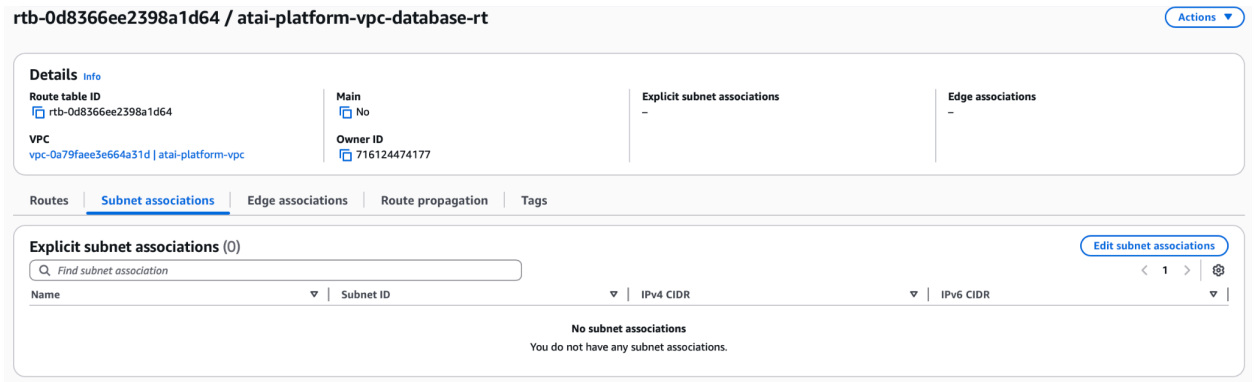
1. Go to VPC Dashboard → Route Tables → Create route table

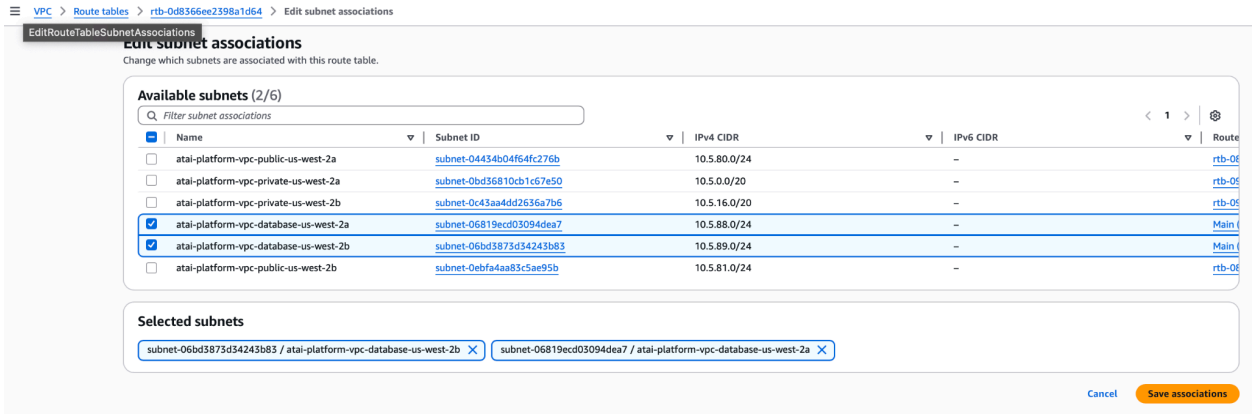


2. Name: atai-platform-vpc-database-rt
3. VPC: Select your VPC from Step 1
4. Click Create route table



5. (No extra routes needed - isolated)
6. Subnet associations → Edit subnet associations:
    a. Select both database subnets → Save associations





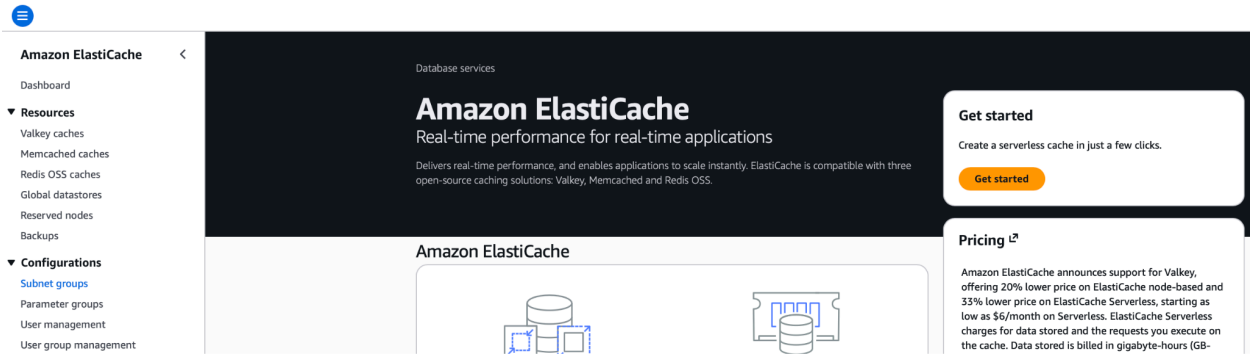
Valkey clusters configuration

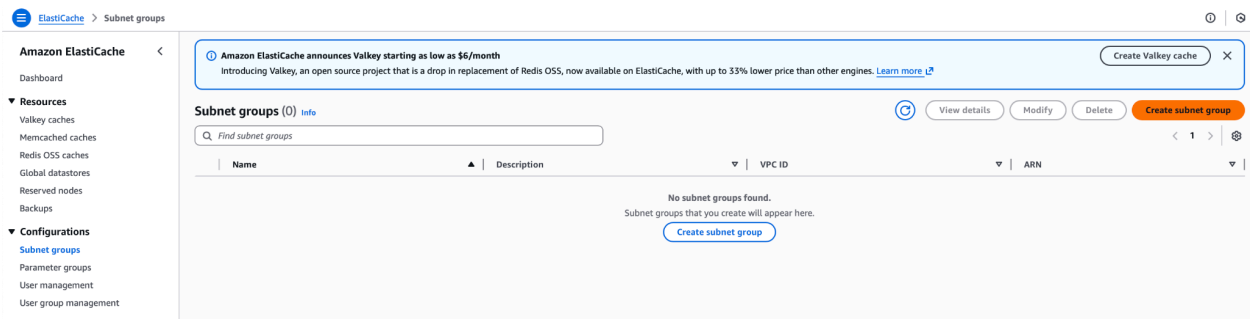
Prerequisites

1. VPC with database subnets
2. Security group allowing access from EKS pods
3. Subnet group for database subnets

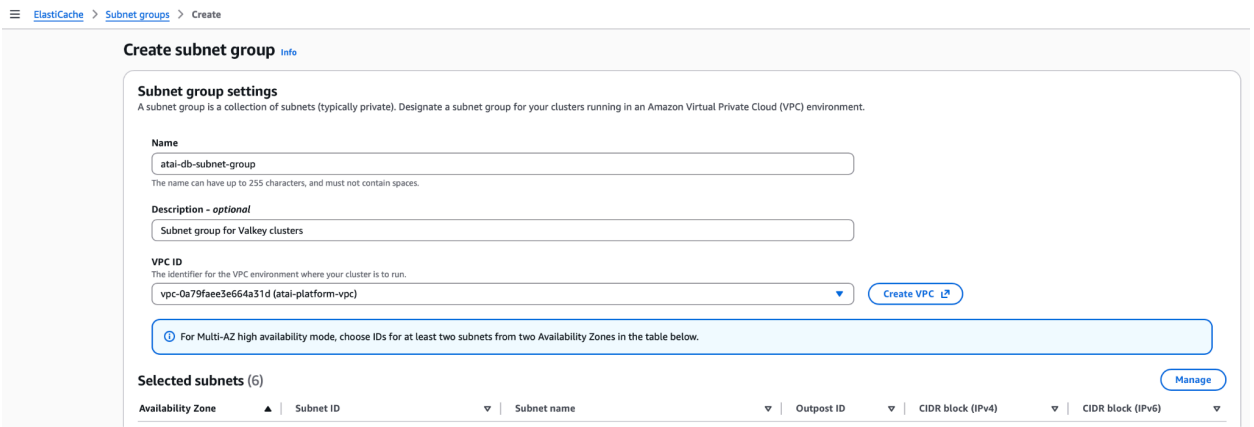
Step 1: Create ElastiCache Subnet Group

1. Go to ElastiCache → Subnet groups → Create subnet group

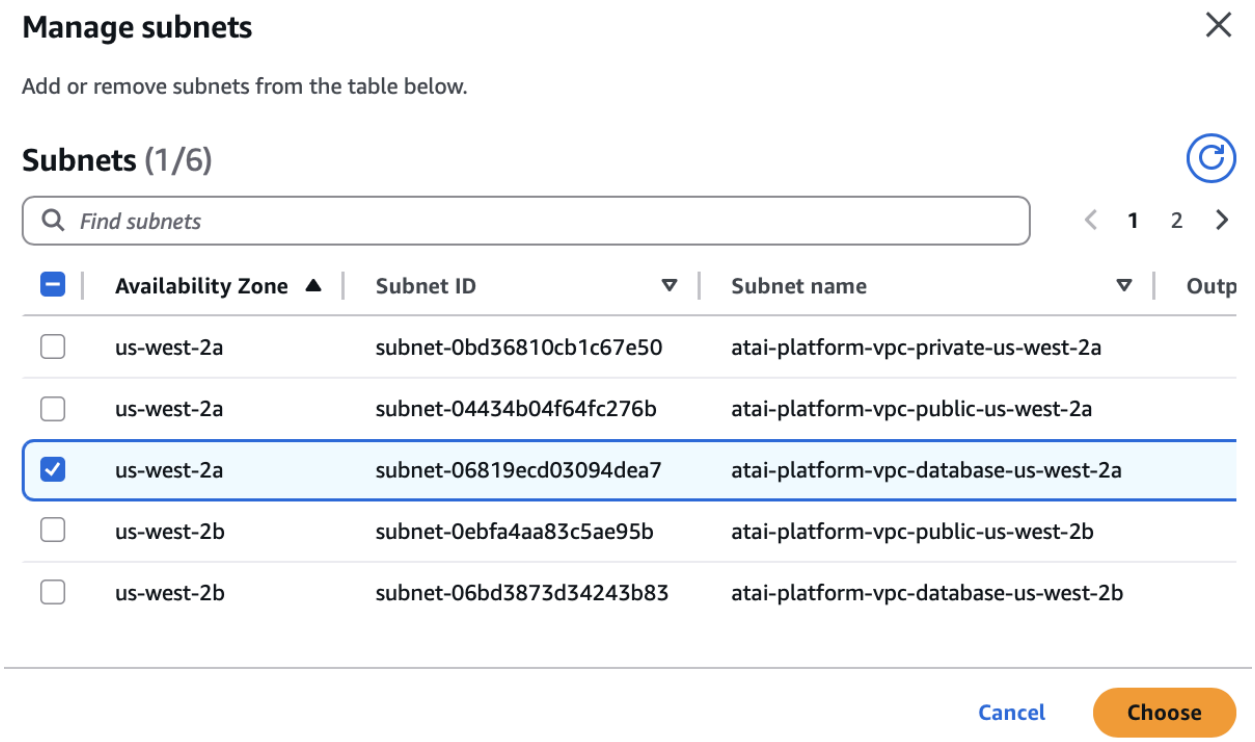




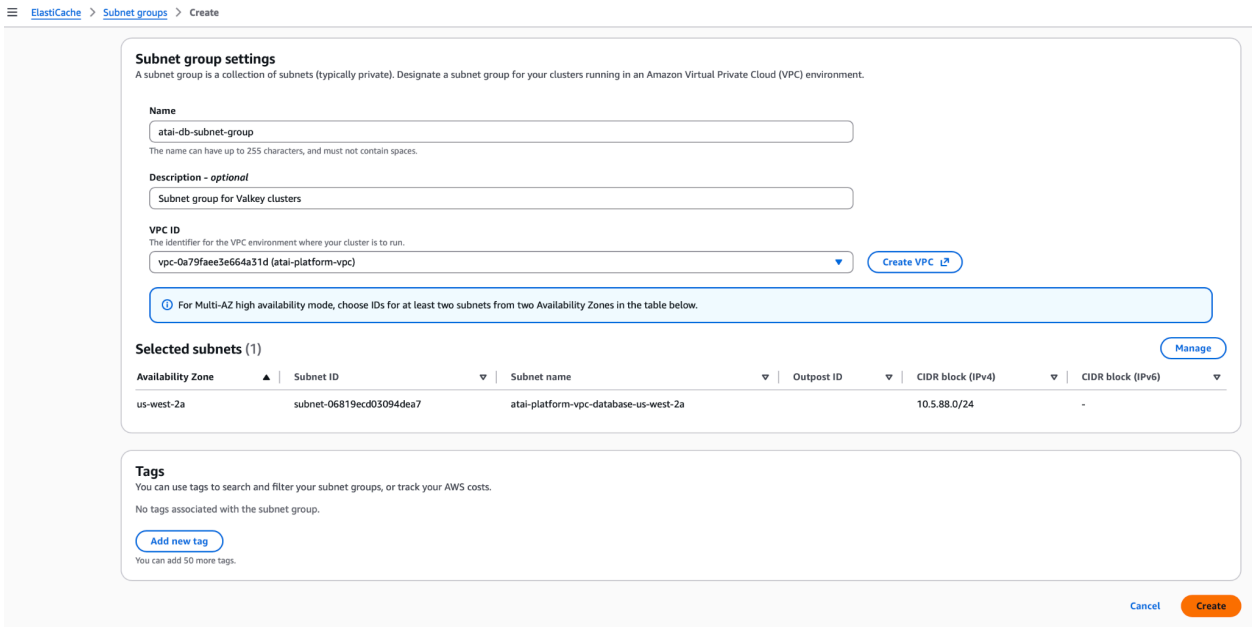
2. Name: atai-db-subnet-group (or your name)
3. Description: Subnet group for Valkey clusters (single AZ for low latency)
4. VPC: Select your VPC



5. In the section **Selected subnets** click on **Manage**
    a. Availability Zones: Select only the first AZ (e.g., us-west-2a) and click on **Choose**
        i. Important: Use only one AZ to reduce latency
        ii. Note: EKS Managed node groups will use a private subnet in the same AZ (different subnet CIDR)



6. Subnets: Select only the first database subnet (e.g., 10.5.88.0/24 in us-west-2a)
    a. Do not select the second database subnet
7. Click Create

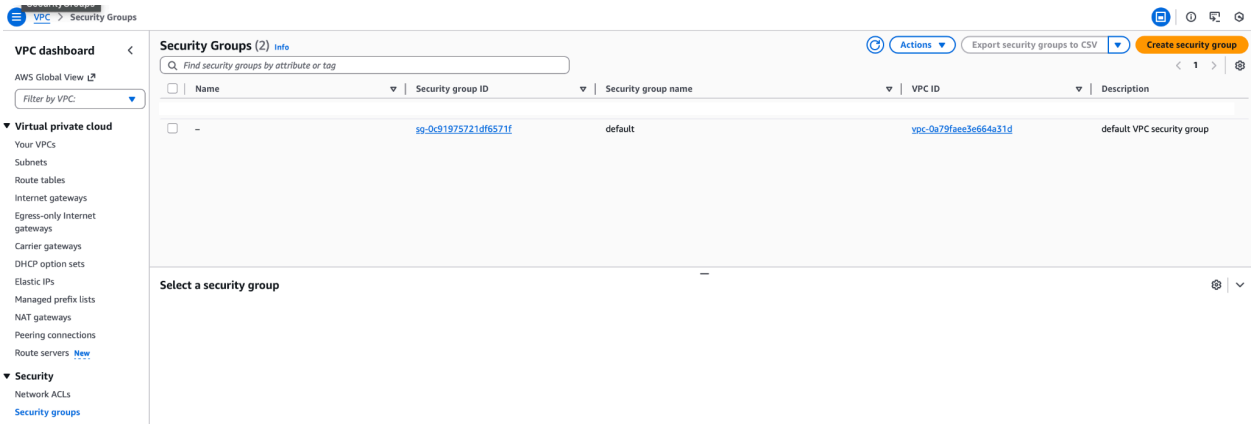


Important Notes:

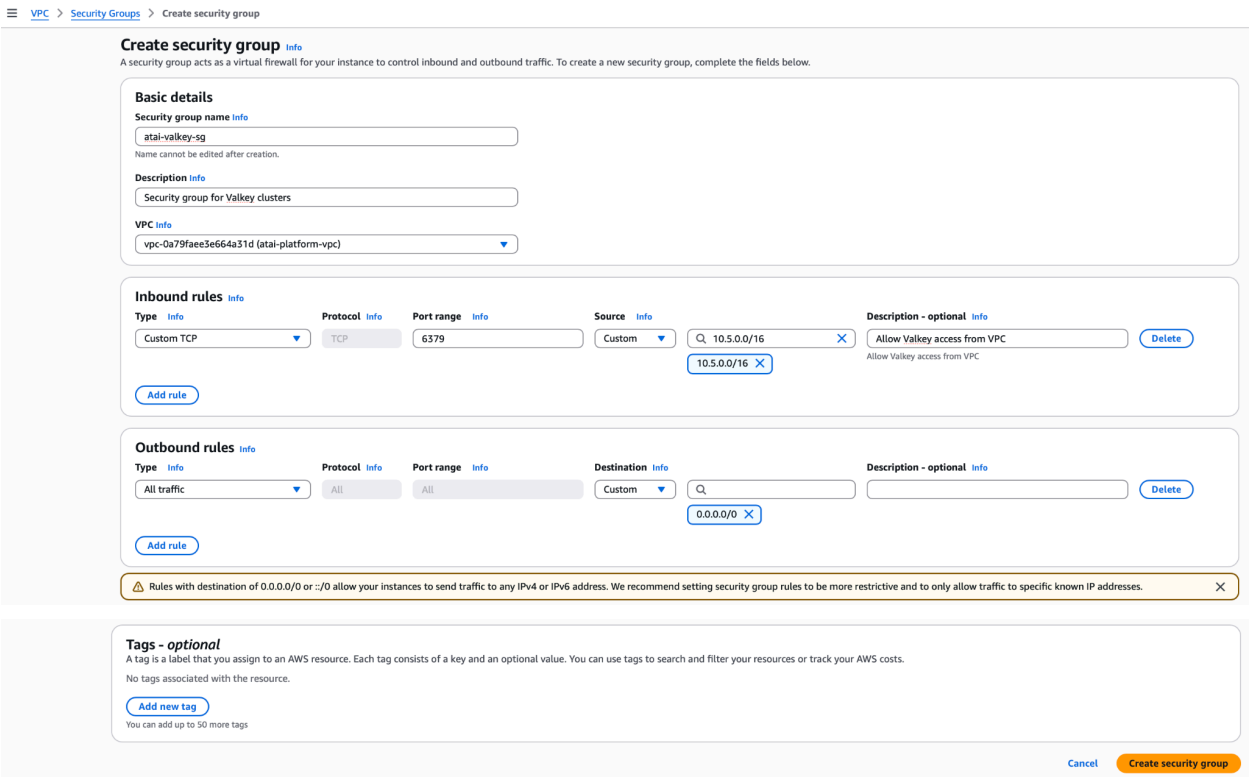
1. Single AZ deployment reduces cross-AZ network latency
2. All Valkey clusters will be created in this single AZ
3. EKS managed node groups should use a private subnet in the same AZ (e.g., 10.5.0.0/20 in us-west-2a) for optimal latency
4. Example: If you use us-west-2a for database subnet 10.5.88.0/24, use us-west-2a for private subnet 10.5.0.0/20 for node groups

Step 2: Create Security Group (if not existing)

1. Go to VPC → Security Groups → Create security group



2. Name: atai-valkey-sg (or your name)
3. Description: Security group for Valkey clusters
4. VPC: Select your VPC from section VPC configuration Step 1
5. Inbound rules: Add rule:
    a. Type: Custom TCP
    b. Port: 6379
    c. Source: Custom → Enter your VPC CIDR (e.g., 10.5.0.0/16)
    d. Description: Allow Valkey access from VPC

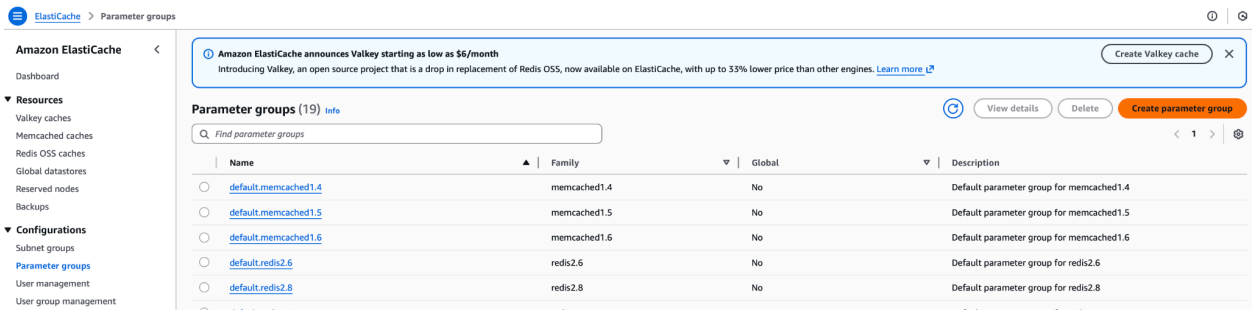


Note: For initial setup, opening to the VPC CIDR simplifies connectivity. Later, restrict to specific security groups (e.g., EKS node group security group) for tighter security.

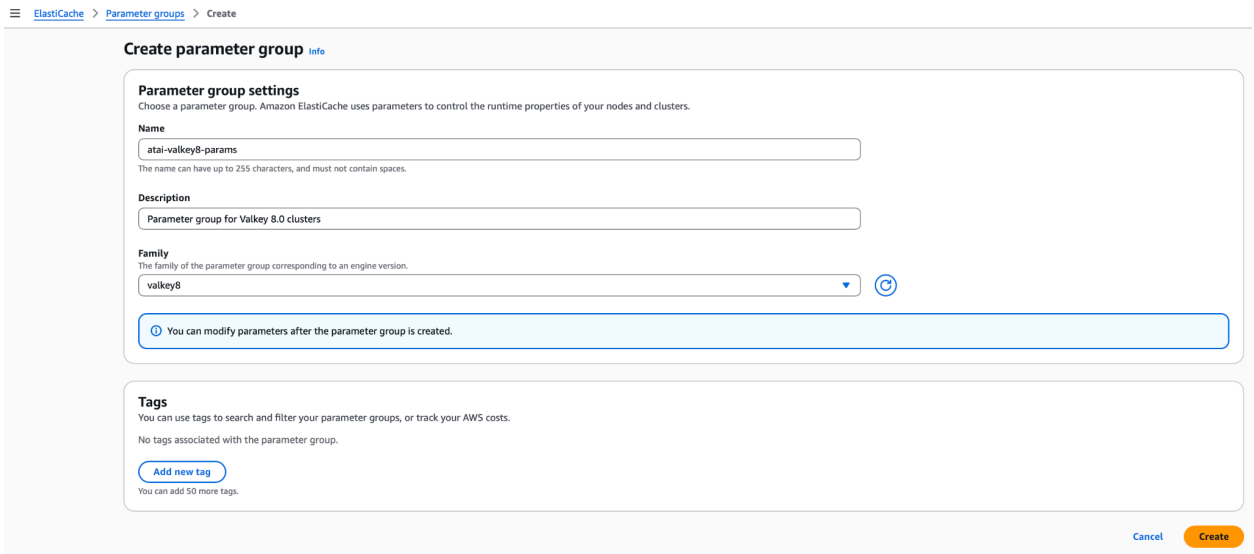
6. Click Create security group

Step 3: Create Parameter Group for Valkey 8.0

7. Go to ElastiCache → Parameter groups → Create parameter group



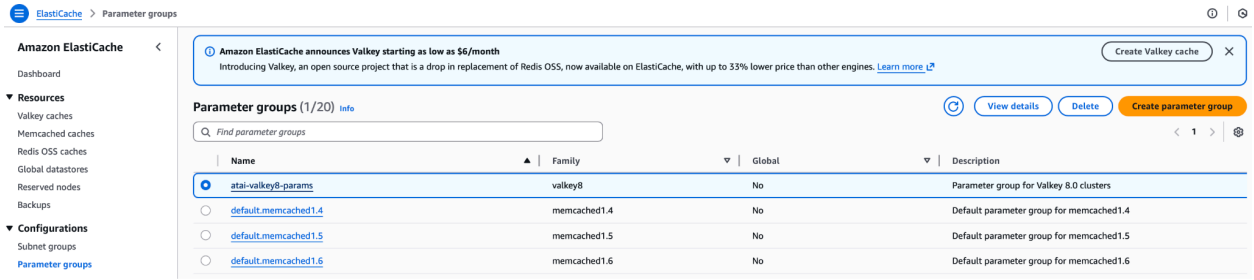
8. Group name: atai-valkey8-params
9. Description: Parameter group for Valkey 8.0 clusters
10. Parameter group family: valkey8



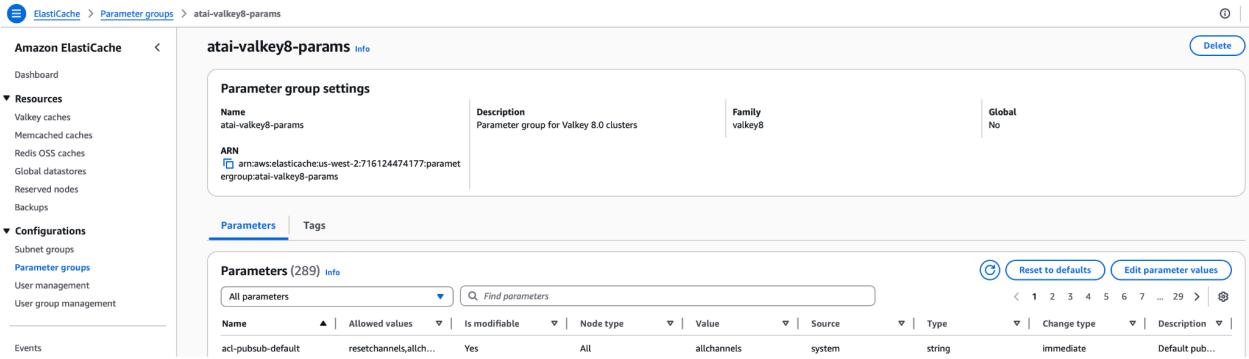
11. Click Create

Step 3.1 After creation, edit the parameter group:

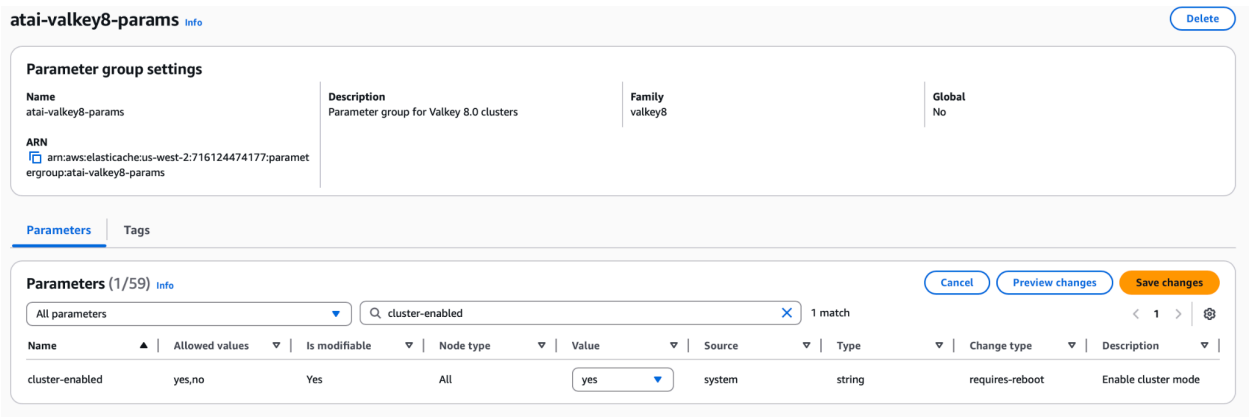
1. Select the parameter group → Edit



2. Click on **Edit parameter values**



3. In search bar type **cluster-enabled**
4. With the dropdown menu set the value to **yes**



5. Click Save changes

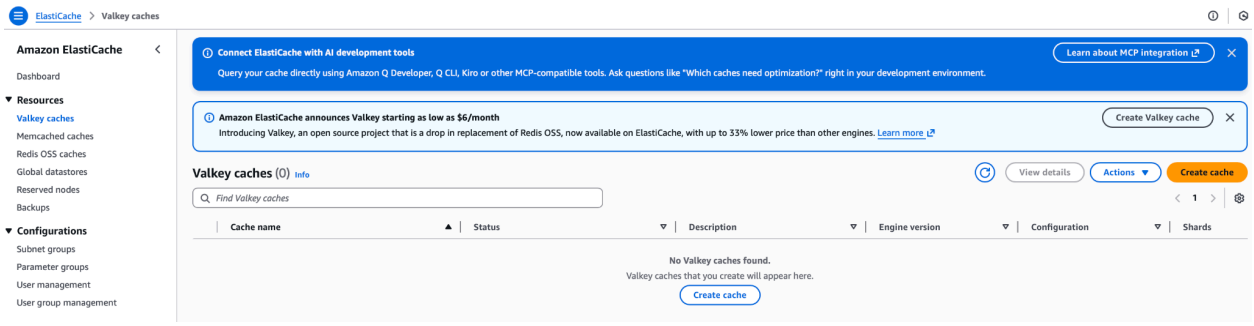
Important: **The cluster-enabled = yes parameter is required for cluster mode to work**. Without it, the clusters won't function properly in cluster mode, even if you enable cluster mode in the cluster settings.

Step 4: Create Valkey Clusters

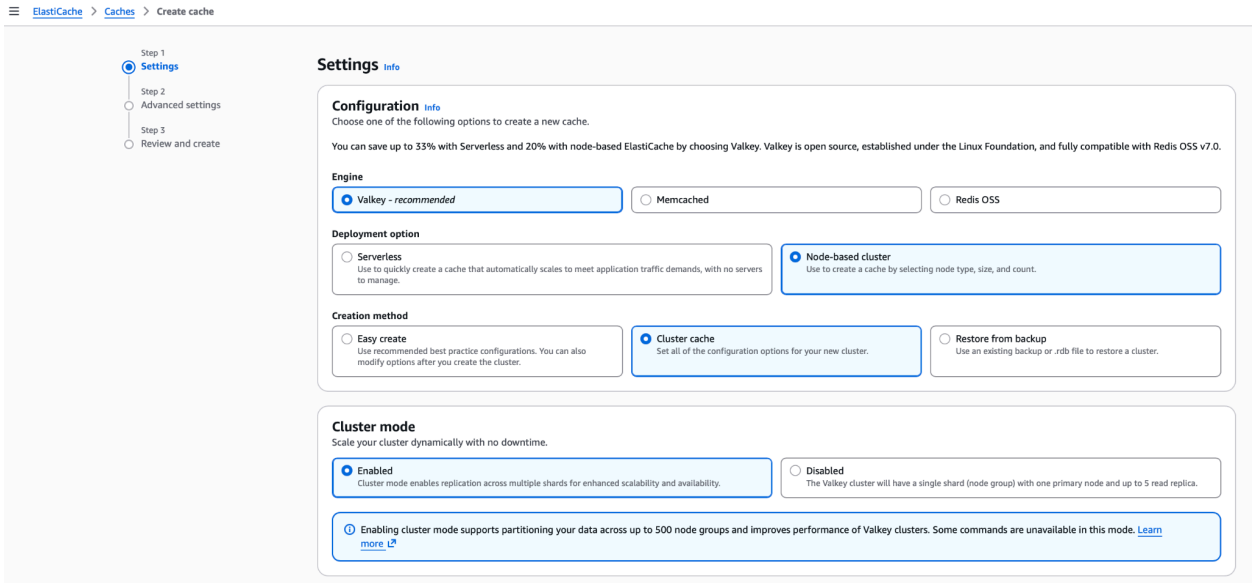
For each cluster, follow these steps.

Cluster 1: registry

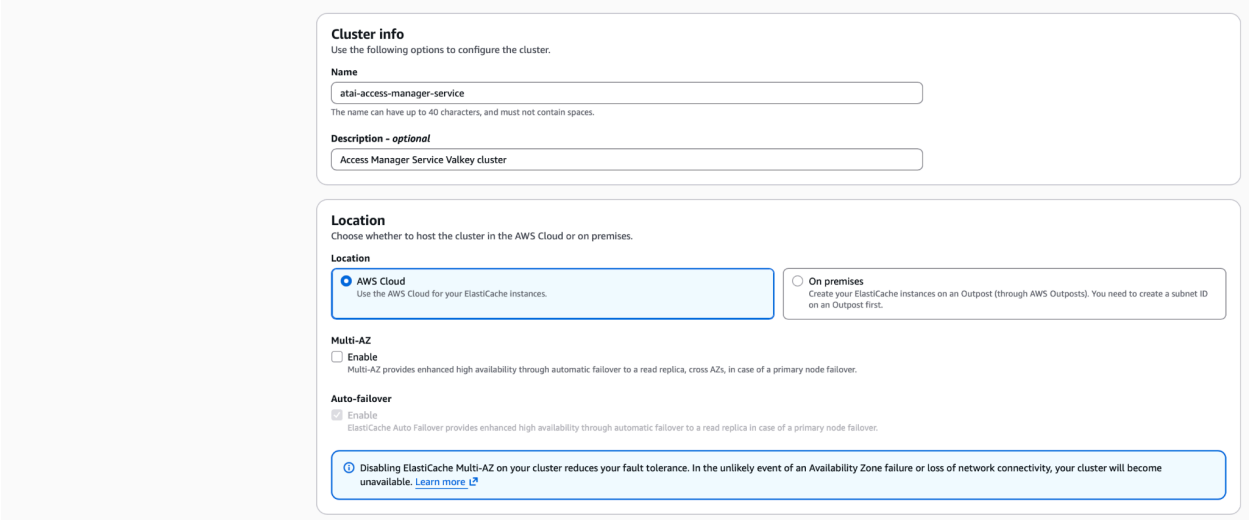
1. ElastiCache → Valkey caches → **Create cache**



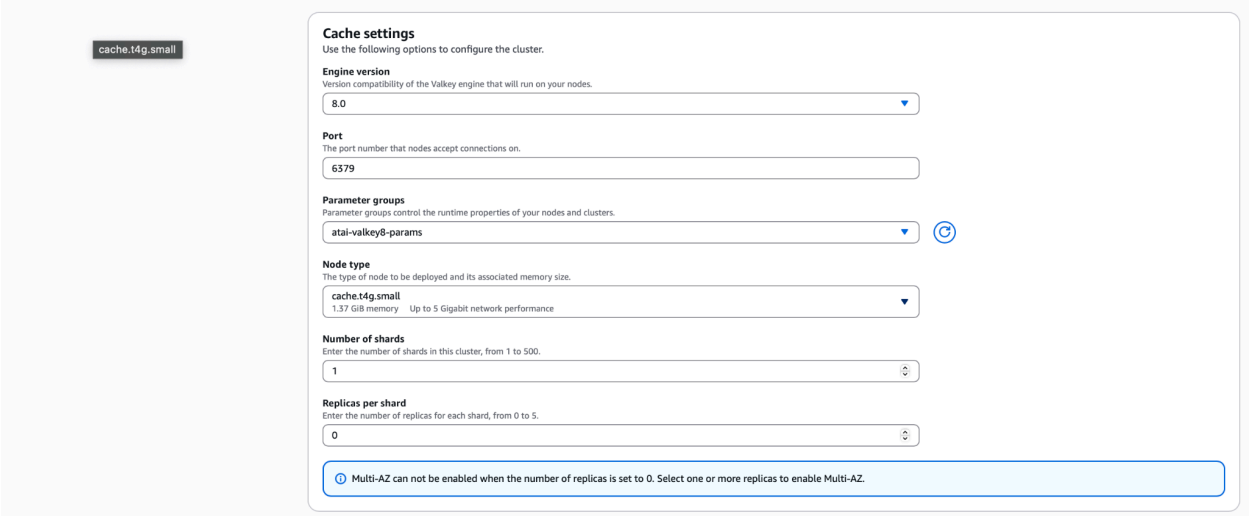
2. Engine: Valkey
3. Deployment option: Node-based cluster
4. Creation method: Cluster cache
5. Cluster mode: Enable



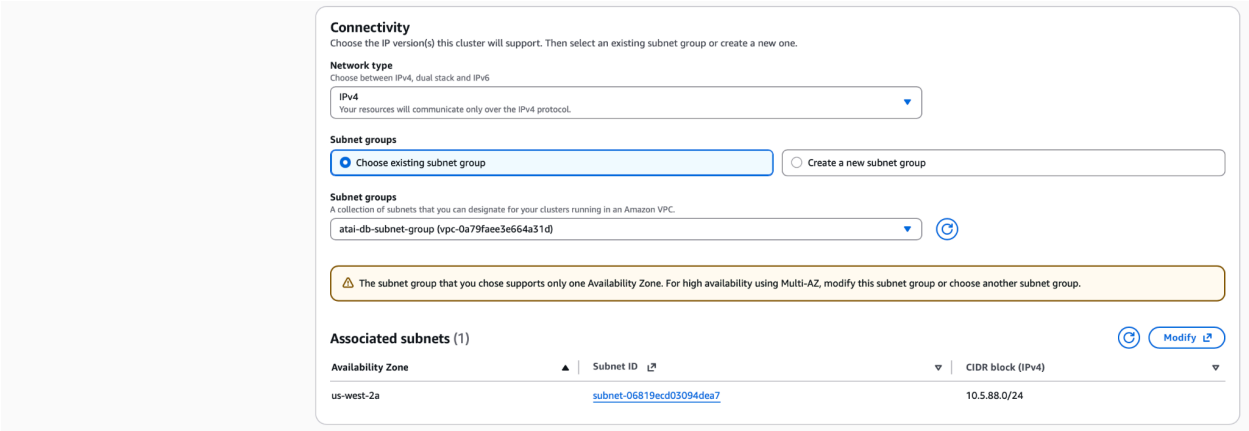
6. Cluster info
    a. Name: atai-registry-service
    b. Description: Registry Service Valkey cluster
7. Location
    a. Location: AWS Cloud
    b. Muti-AZ: Disable (to reduce latency)



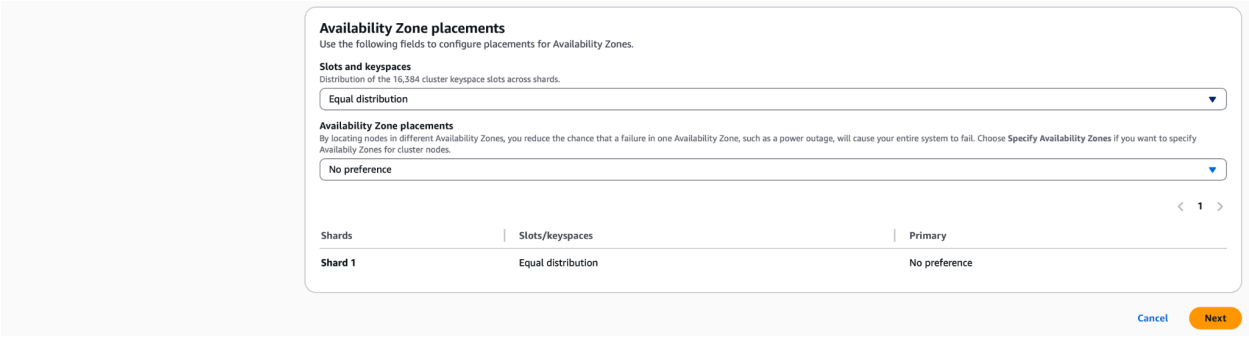
8. Cache setting
    a. Engine version: 8.0
    b. Port: 6379
    c. Parameter group: Select the parameter group created in the Step 3 of this section
    d. Node type: cache.t4g.small
    e. Number of shards: 1
    f. Replicas per shard: 0



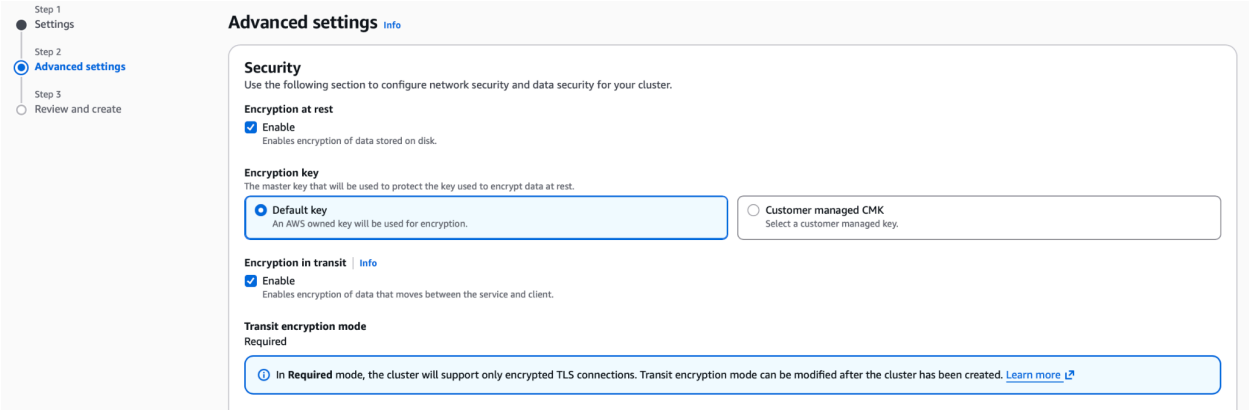
9. Connectivity:
    a. **Subnet groups**: Select your database subnet group (single AZ) from Step 1



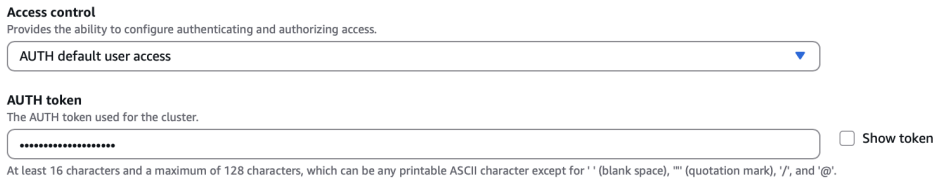
    b. Use the default Availability Zone placements -> Next



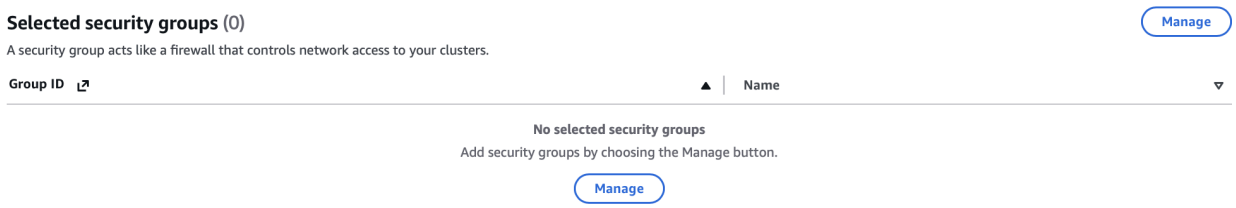
10. Enable encryption at rest and Encryption in transit



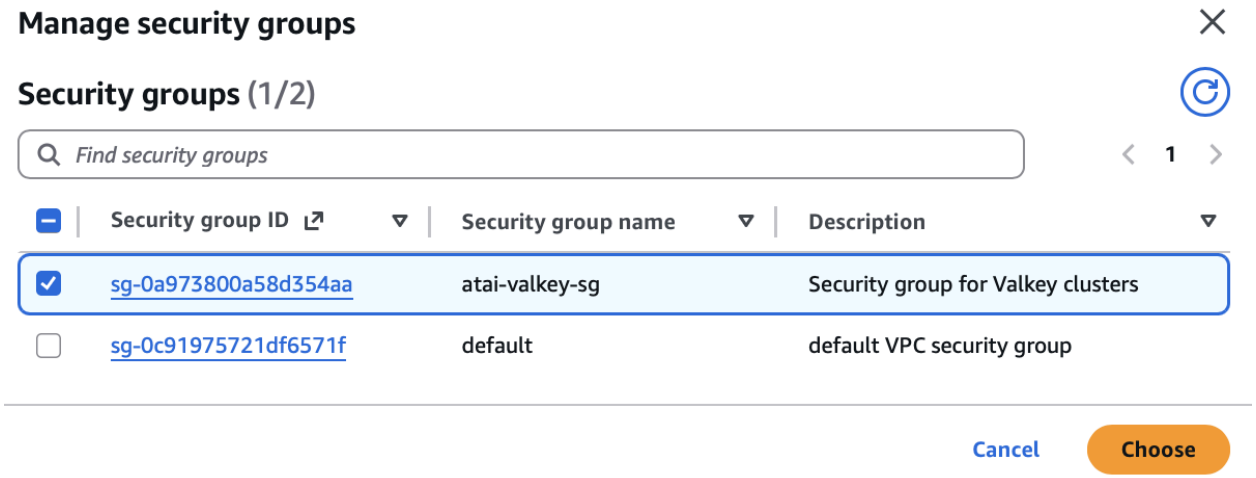
11. For access control choose **AUTH default user access**
    a. Important: Save this auth token securely. It is required later to communicate with your Valkey cluster.
    b. Security recommendation: Create a unique auth token per Valkey cluster (do not reuse the same token across clusters).
    c. Store each token in a secure location such as AWS Secrets Manager.



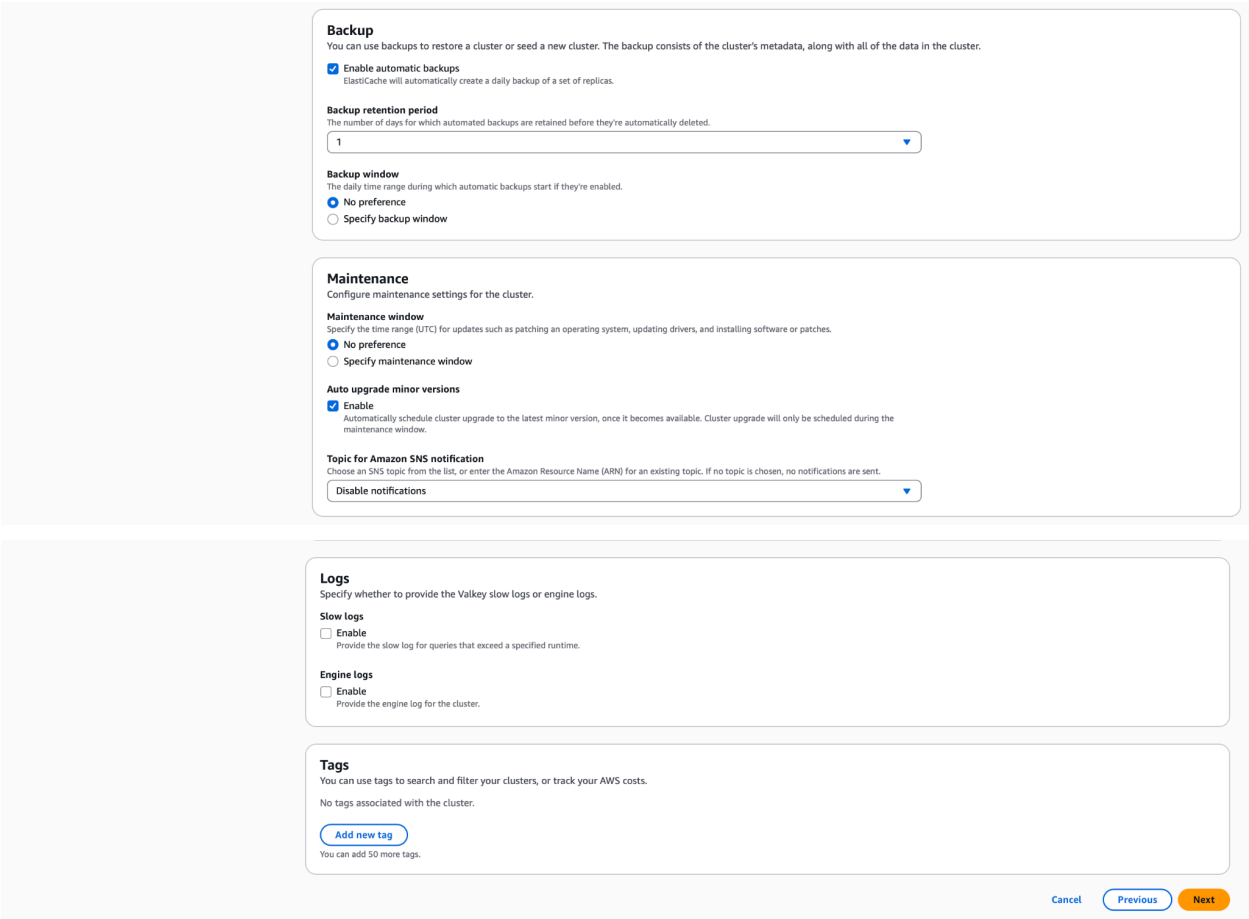
12. Security groups
    a. Click on **Manage**



    b. Select the security group created in Step 2
    c. Click on **Choose**

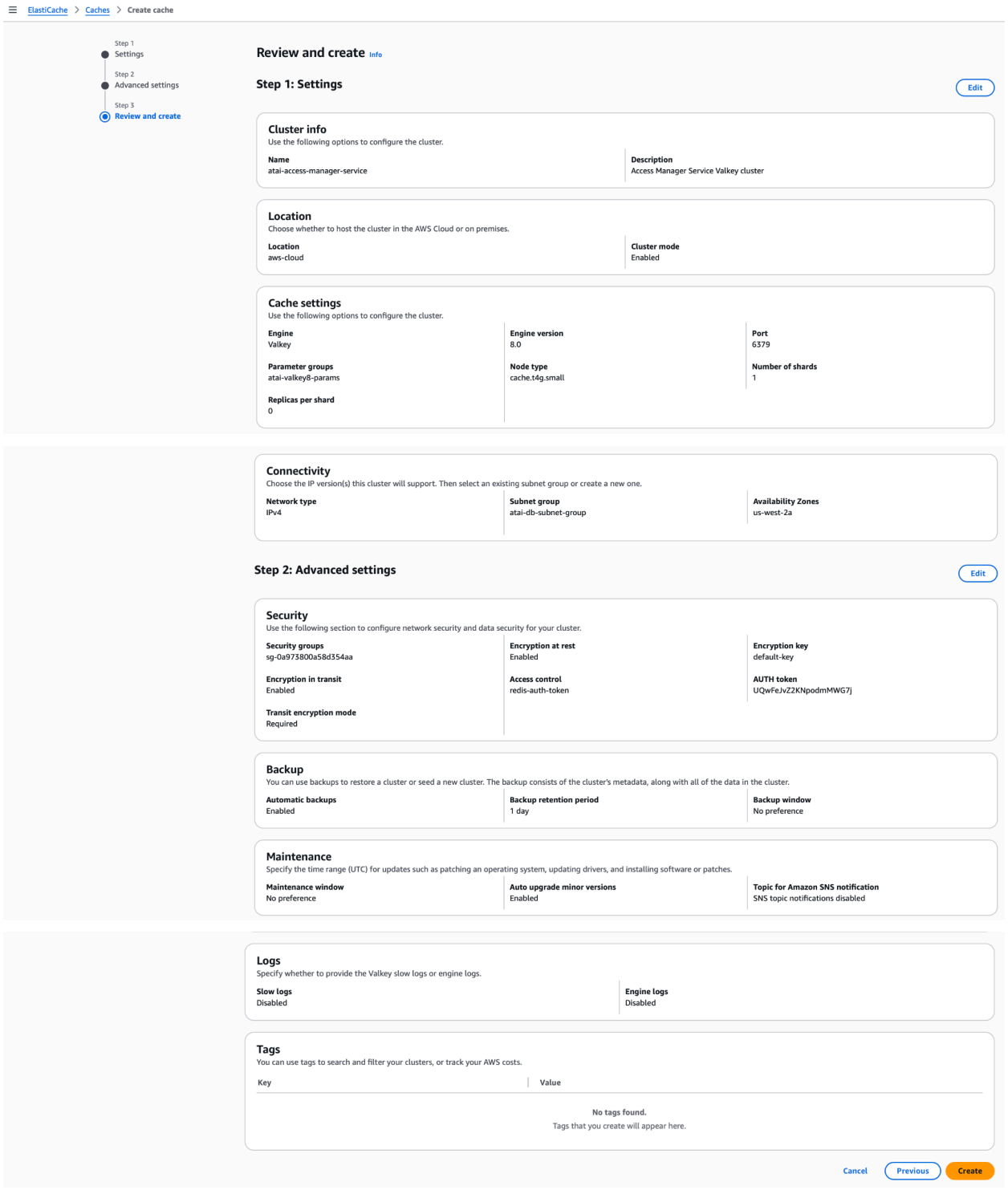


13. Use the default configuration for **Backups**, **Maintenance**, **Logs** and **Tags.**



14. Click Next

15. Review your configuration



16. Click on **Create**

⚠️Store your Valkey host, and authtoken in a secure location. You will need them later in the *atai-platform* prerequisites.

Cluster 2: access-manager

Same as Cluster 1, but:

1. Name: atai-access-manager-service
2. Auth token: Generate a new unique token (different from previous clusters)

⚠️Store your Valkey host, and token in a secure location. You will need them later in the atai-platform prerequisites.

Cluster 3: jos

Same as Cluster 1, but:

3. Name: atai-jos-service
4. Auth token: Generate a new unique token (different from previous clusters)

⚠️ Store your Valkey host, and token in a secure location. You will need them later in the atai-platform prerequisites.

Cluster 4: data-service (10 shards)

Same as Cluster 1, but:

1. Name: atai-data-service
2. Number of shards: 10 (instead of 1)
3. Replicas per shard: 0 (same for the other 10 shard caches)
4. Auth token: Generate a new unique token (different from previous clusters)

⚠️ Store your Valkey host, and token in a secure location. You will need them later in the atai-platform prerequisites.

Cluster 5: gpq (10 shards)

Same as Cluster 1, but:

1. Name: atai-gpq-service
2. Number of shards: 10 (instead of 1)
3. Replicas per shard: 0 (same for the other 10 shard caches)
4. Auth token: Generate a new unique token (different from previous clusters)

⚠️ Store your Valkey host, and token in a secure location. You will need them later in the atai-platform prerequisites.

Cluster 6: health

Same as Cluster 1, but:

1. Name: atai-health-service
2. Auth token: Generate a new unique token (different from previous clusters)

⚠️ Store your Valkey host, and token in a secure location. You will need them later in the atai-platform prerequisites.

Cluster 7: lens (10 shards)

Same as Cluster 1, but:

1. Name: atai-lens-service
2. Number of shards: 10 (instead of 1)
3. Replicas per shard: 0 (same for the other 10 shard caches)
4. Auth token: Generate a new unique token (different from previous clusters)

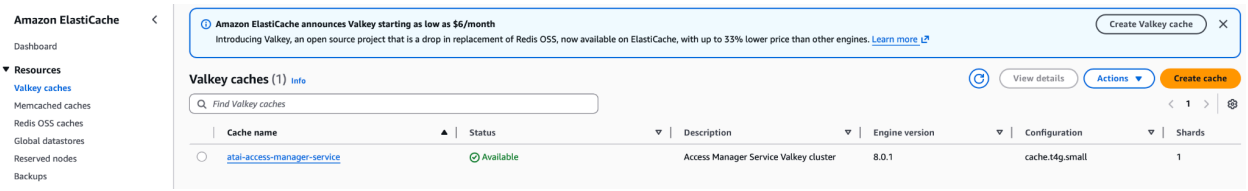
⚠️ Store your Valkey host, and token in a secure location. You will need them later in the atai-platform prerequisites.

Step 5: Store Auth Tokens in AWS Secrets Manager (Recommended)

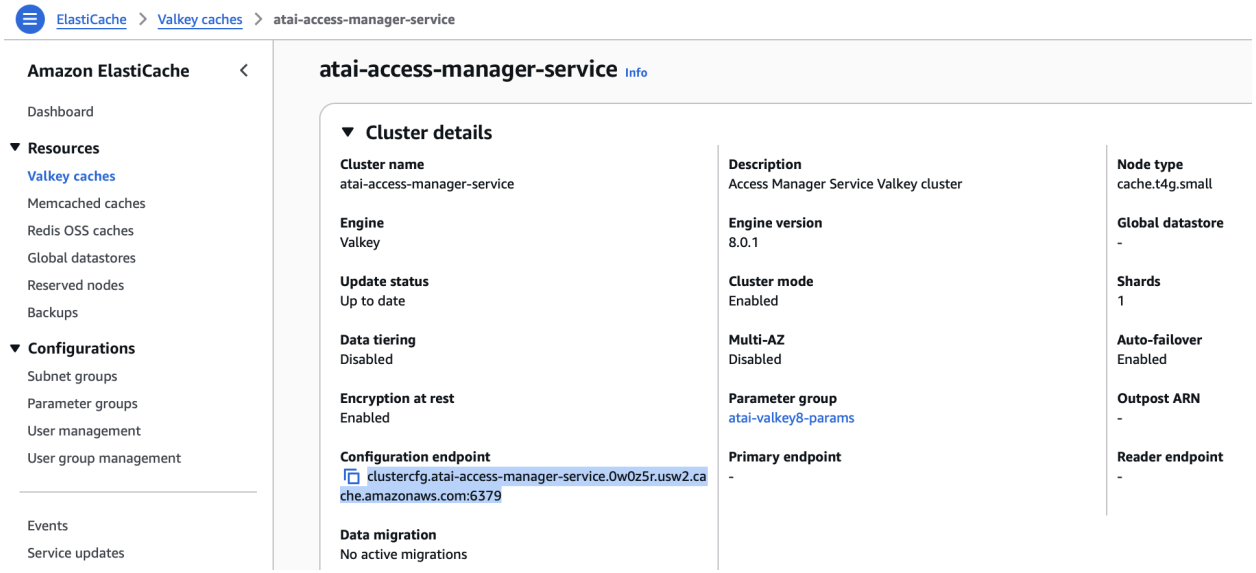
Note: This step is recommended but not mandatory. You can store credentials in Secrets Manager for easier access, better security, and integration with your applications. If you prefer to manage credentials differently, you can skip this step.

For each cluster, store the auth token and connection details:

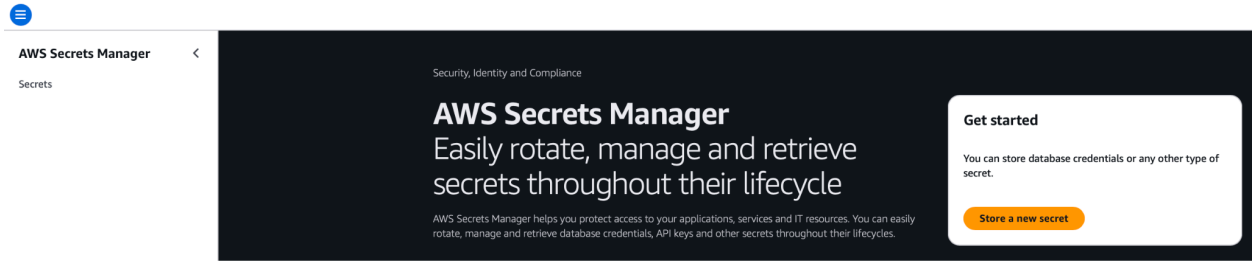
1. Go to ElastiCache → Valkey caches → Click on your Valkey cache cluster



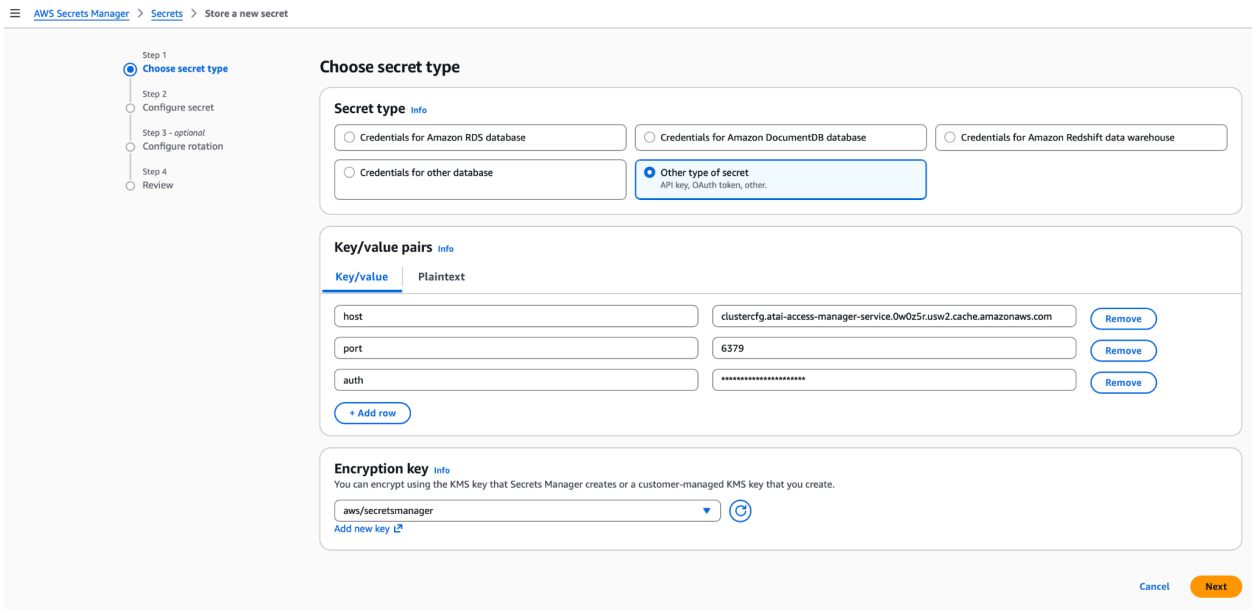
2. Copy the **Configuration endpoint**



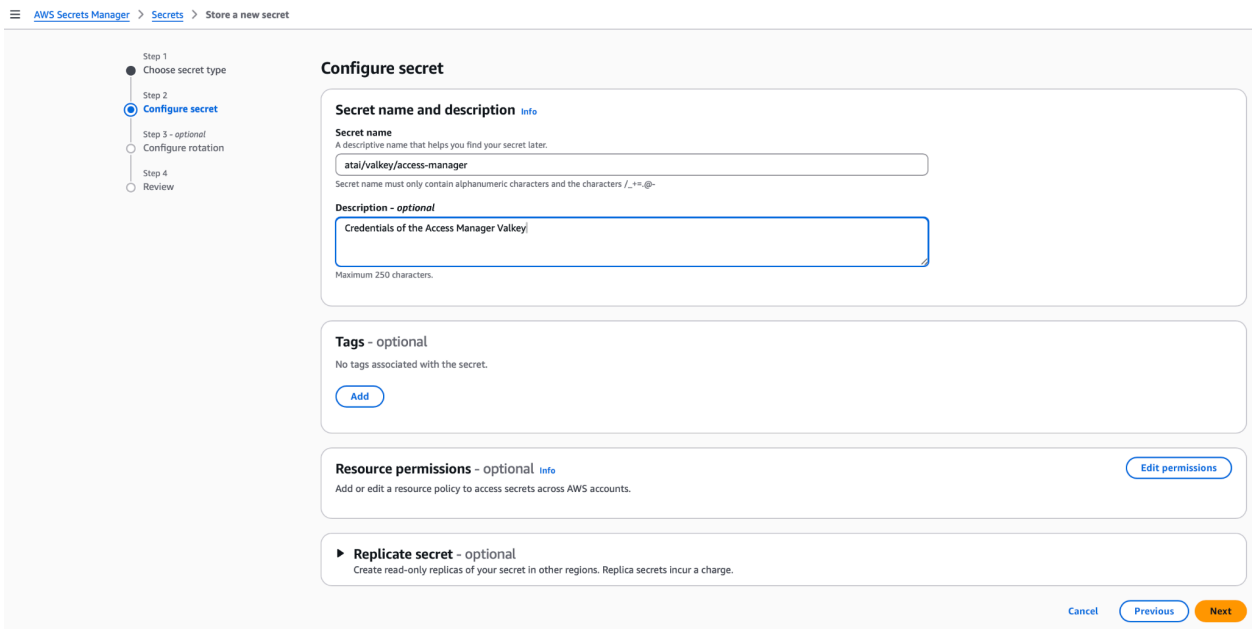
3. Go to Secrets Manager → **Store a new secret**



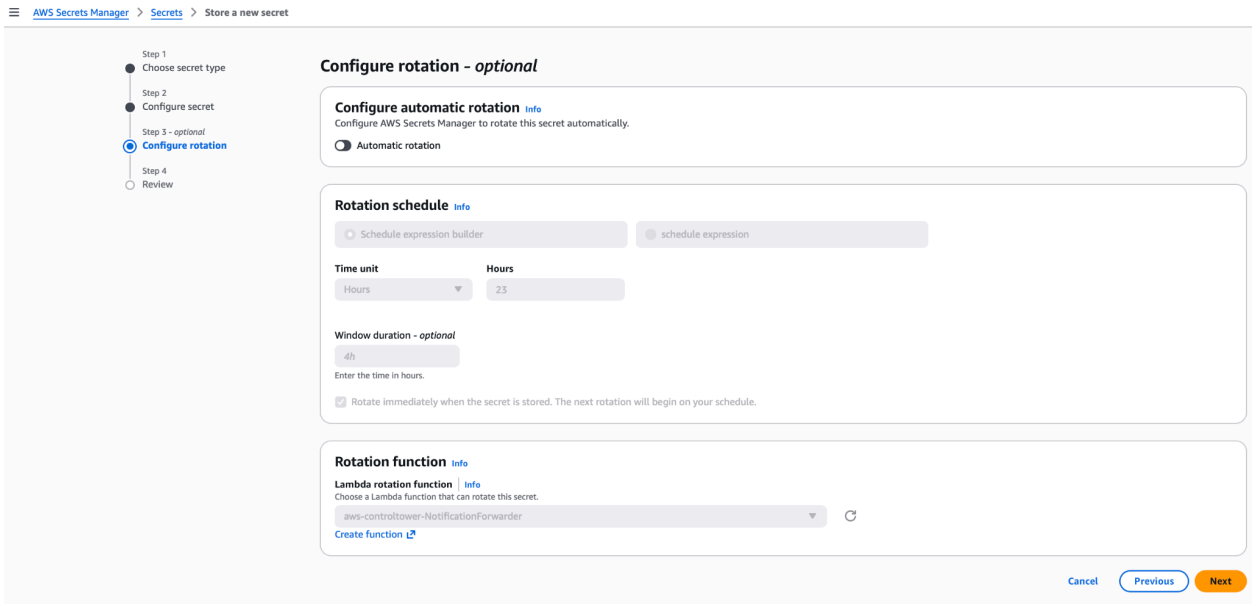
4. Secret type: Other type of secret
5. Key/value pairs: Add:
    a. host: Configuration endpoint address (from cluster details, e.g., atai-access-manager-service.xxxxx.cache.amazonaws.com)
    b. port: 6379
    c. auth_token: The auth token you set for this cluster



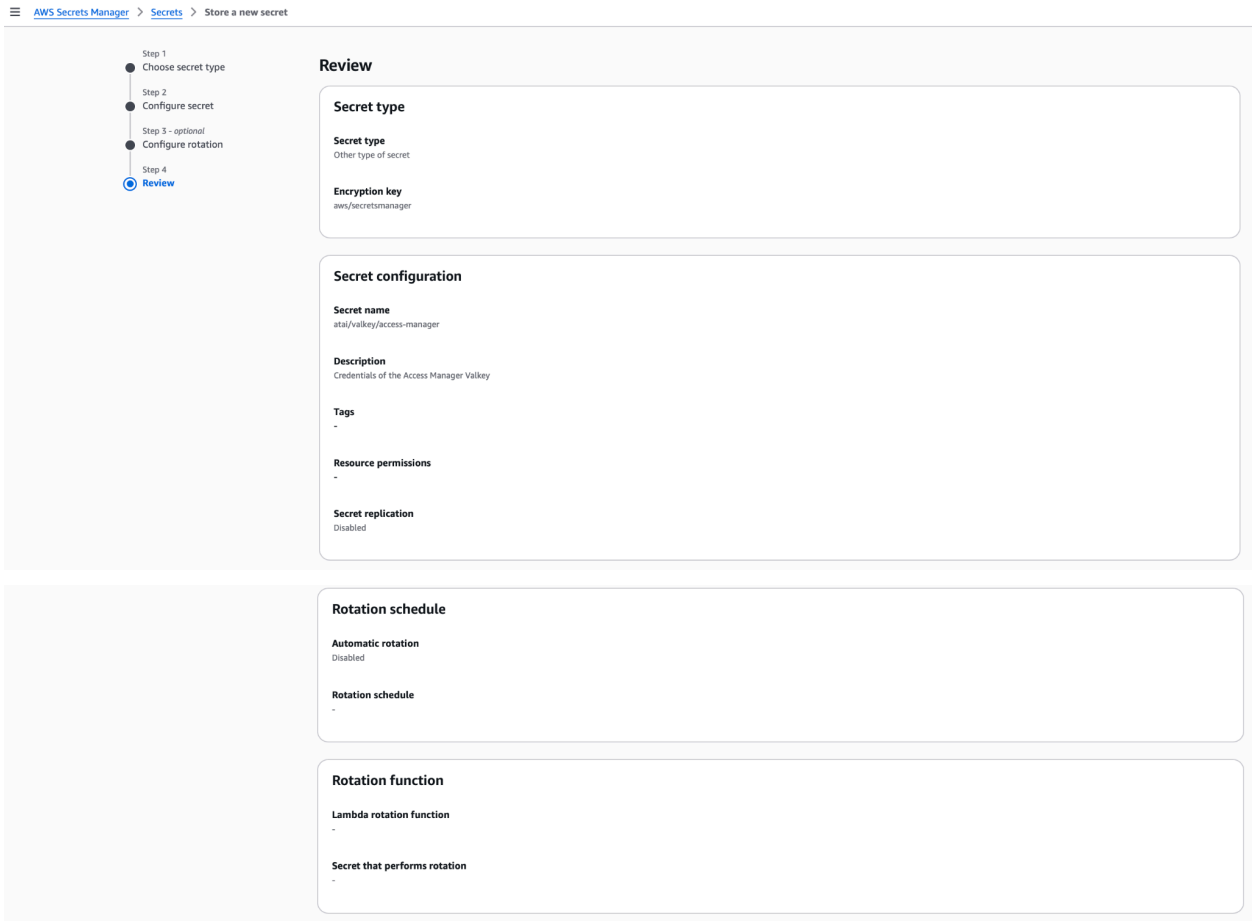
6. Secret name: atai/valkey/{service-name}
   Examples:
   a. atai/valkey/access-manager
   b. atai/valkey/api-events
   c. atai/valkey/dfc
   d. etc.
7. Encryption key: Use AWS managed key (default) or your KMS key
8. Click Next → Next

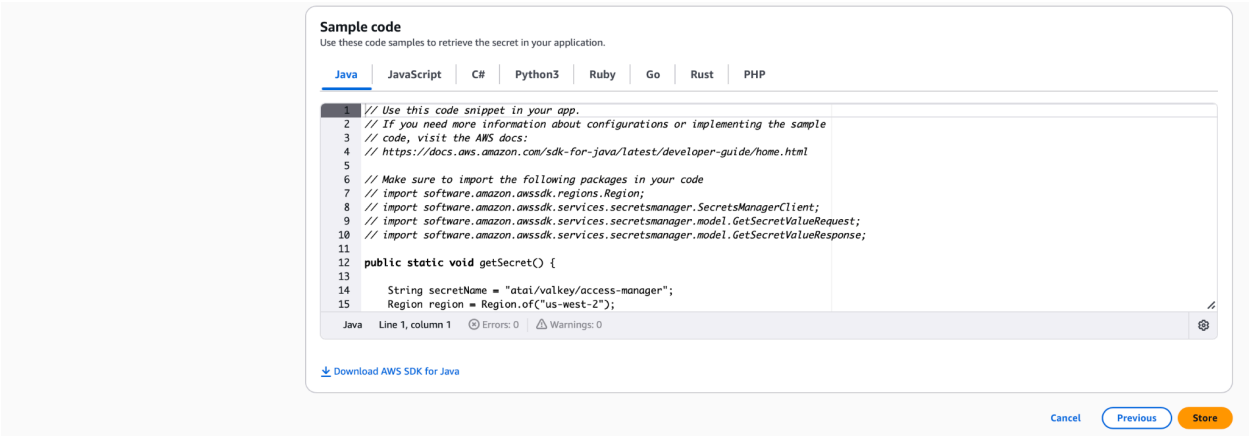


9. Do not configure rotation Click Next



10. Review and click on **Store**





11. Repeat for all 8 clusters with their respective:
    a. Configuration endpoint (host)
    b. Auth token
    c. Service name in the secret path

Benefits of using Secrets Manager:

- Applications can retrieve credentials programmatically
- Credentials are encrypted at rest
- Access is logged for audit purposes
- No need to hardcode credentials in application code

Alternative: If you skip this step, ensure you have the auth tokens and configuration endpoints stored securely elsewhere, as they are required for applications to connect to the Valkey clusters.

⚠️ Store your Valkey host, port, and token in a secure location. You will need them later in the *atai-platform* prerequisites.

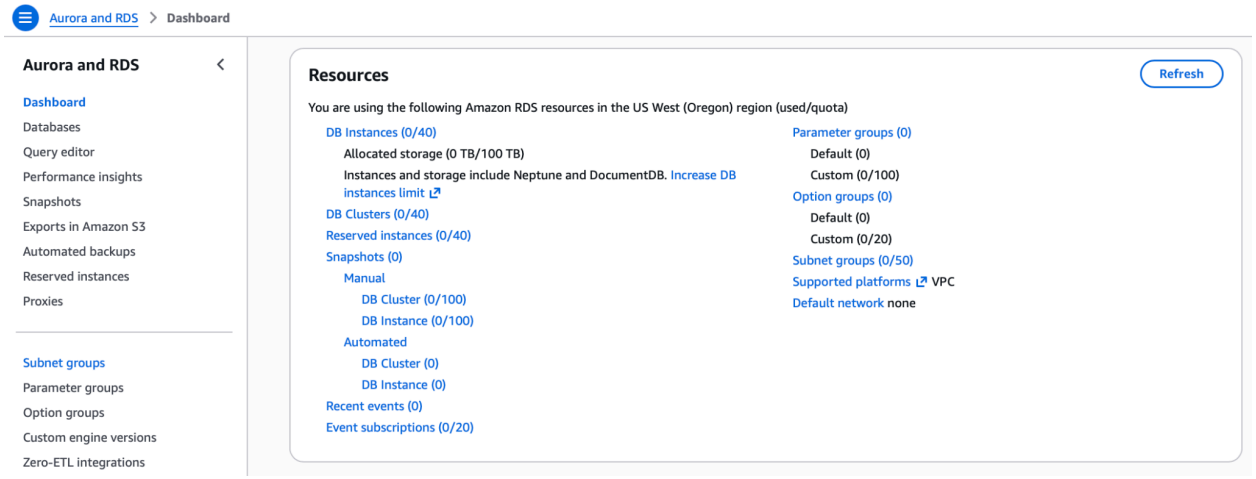
PostgreSQL database configuration

Prerequisites

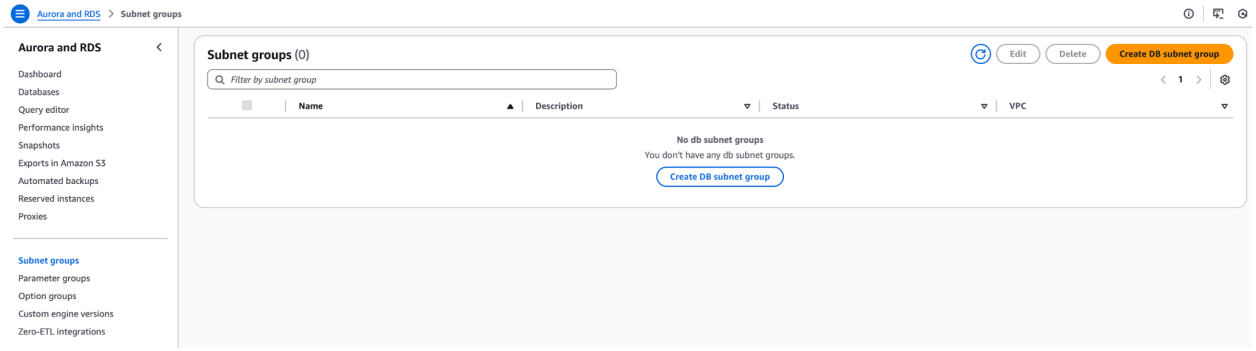
1. VPC with database subnets (at least 2 AZs required for Aurora subnet group)
2. Security group allowing access from EKS pods
3. Database subnet group

Step 1: Create Database Subnet Group

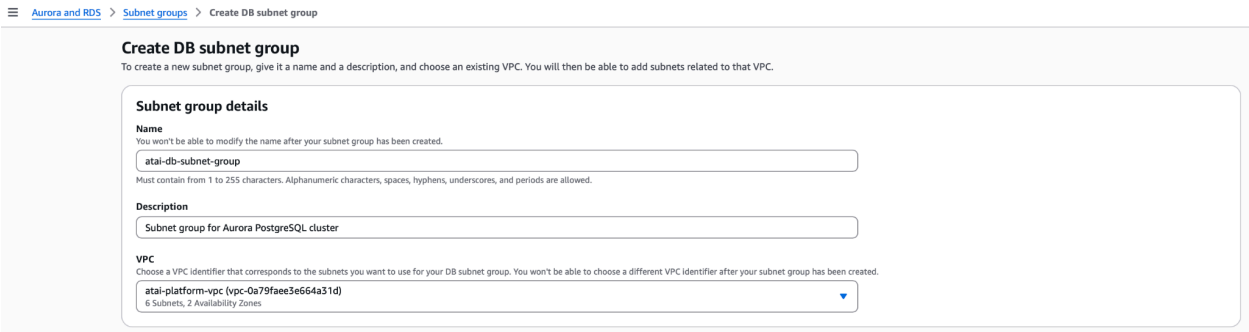
1. Go to RDS dashboard



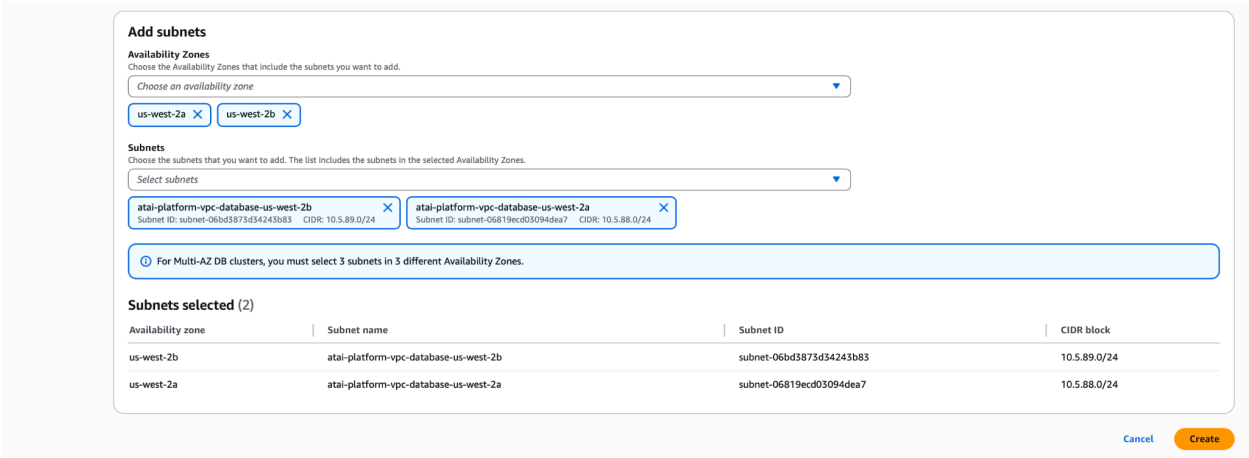
2. Subnet groups → Create DB subnet group



3. Name: atai-db-subnet-group (or your name)
4. Description: Subnet group for Aurora PostgreSQL cluster
5. VPC: Select your VPC



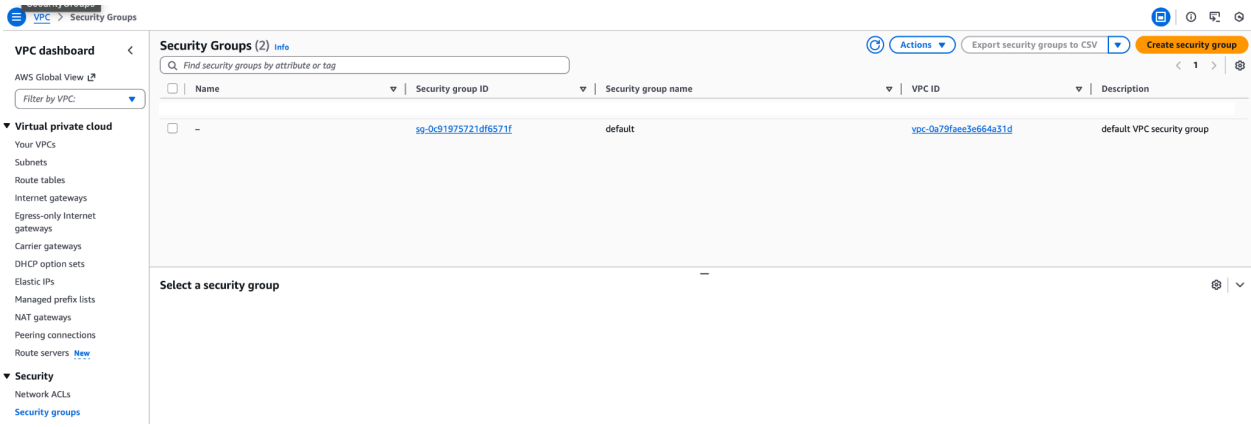
6. Availability Zones: Select at least 2 AZs (AWS requirement for Aurora)
    a. Note: Aurora requires at least 2 AZs for the subnet group, even if you deploy a single instance
    b. Example: Select us-west-2a and us-west-2b
7. Subnets: Select your database subnets atai-platform-vpc-database-us-west-2a and atai-platform-vpc-database-us-west-2b (e.g., 10.5.88.0/24 in us-west-2a and 10.5.89.0/24 in us-west-2b)



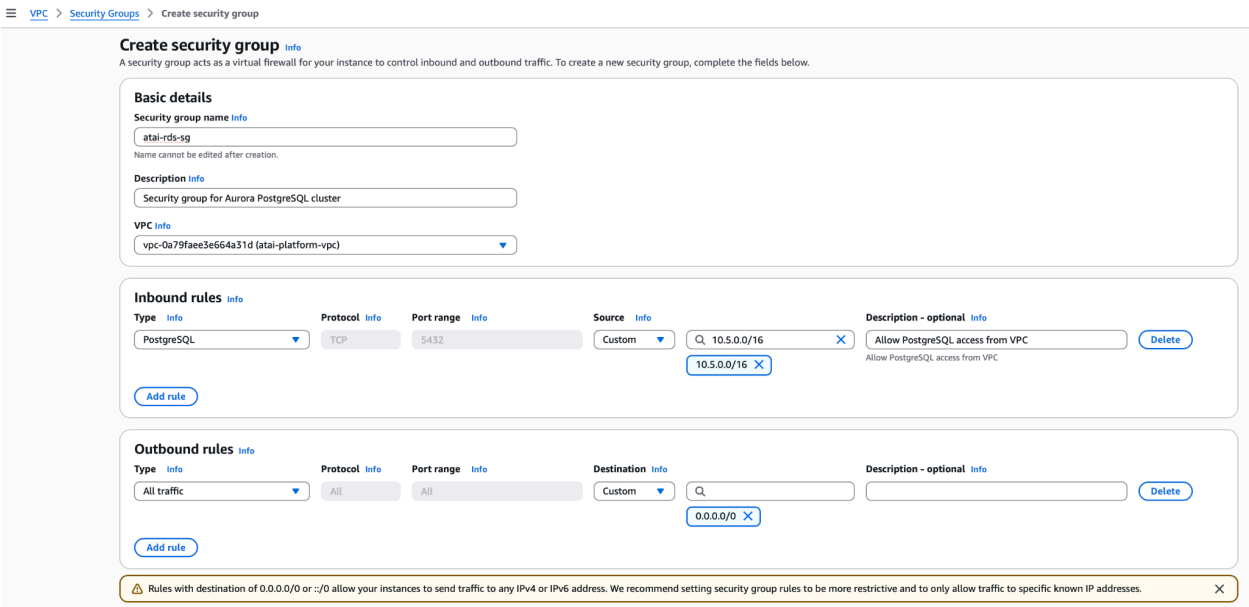
8. Click Create

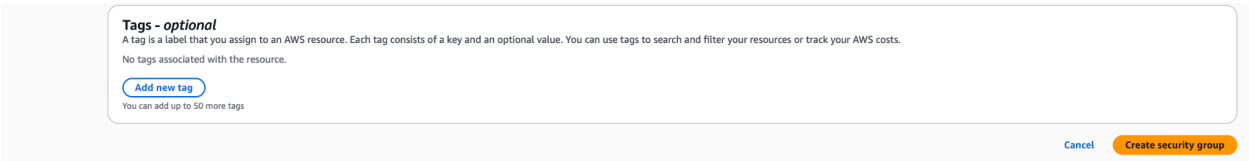
Step 2: Create Security Group (if not existing)

1. Go to VPC → Security Groups → Create security group



2. Name: atai-rds-sg (or your name)
3. Description: Security group for Aurora PostgreSQL cluster
4. VPC: Select your VPC from section VPC configuration Step 1
5. Inbound rules: Add rule:
    a. Type: PostgreSQL
    b. Port: 5432
    c. Source: Custom → Enter your VPC CIDR (e.g., 10.5.0.0/16)
    d. Description: Allow PostgreSQL access from VPC

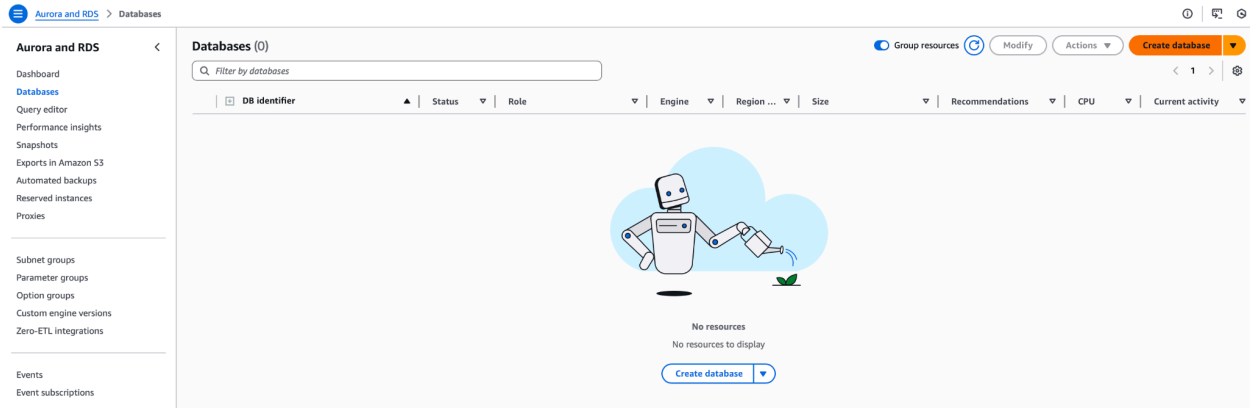




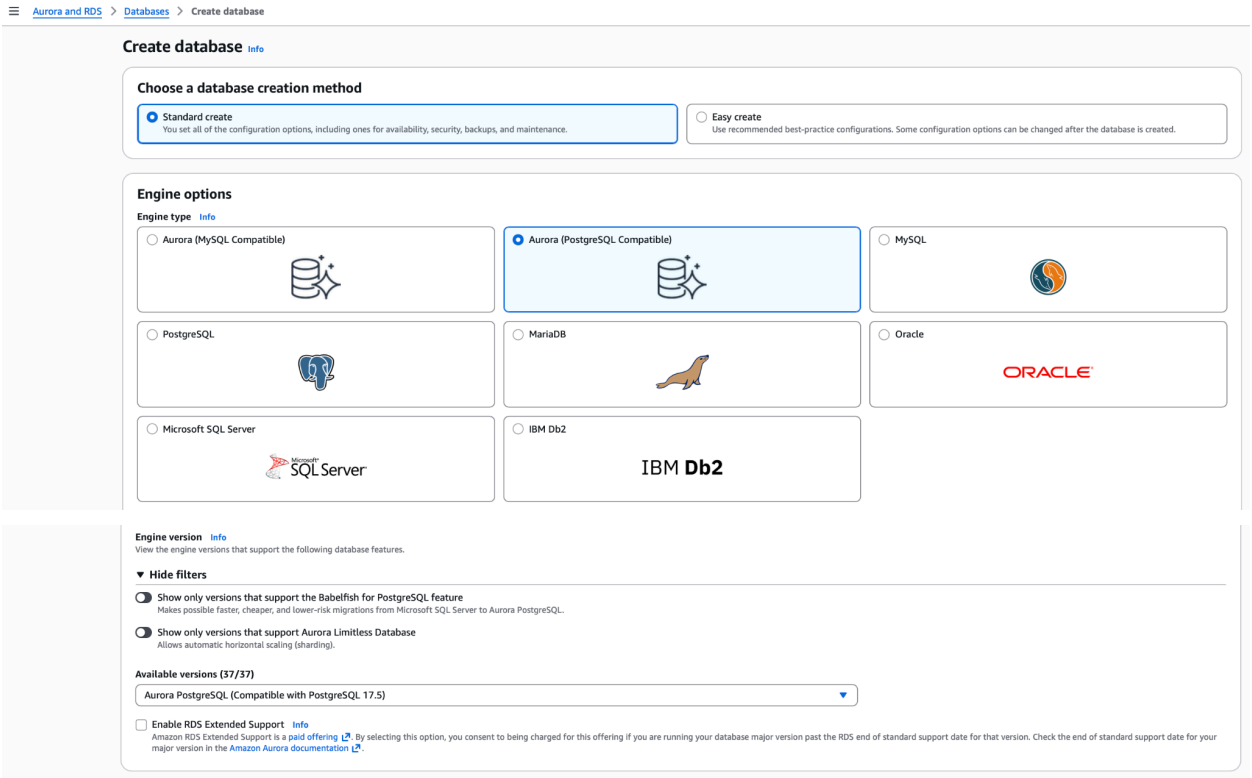
Note: For initial setup, opening to the VPC CIDR simplifies connectivity. Later, restrict to specific security groups (e.g., EKS node group security group) for tighter security.
Click Create security group

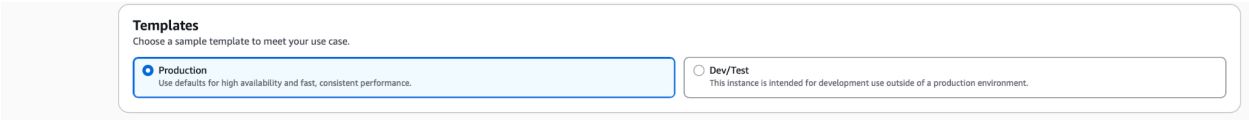
Step 3: Create Aurora PostgreSQL Cluster

1. Go to RDS → Databases → Create database

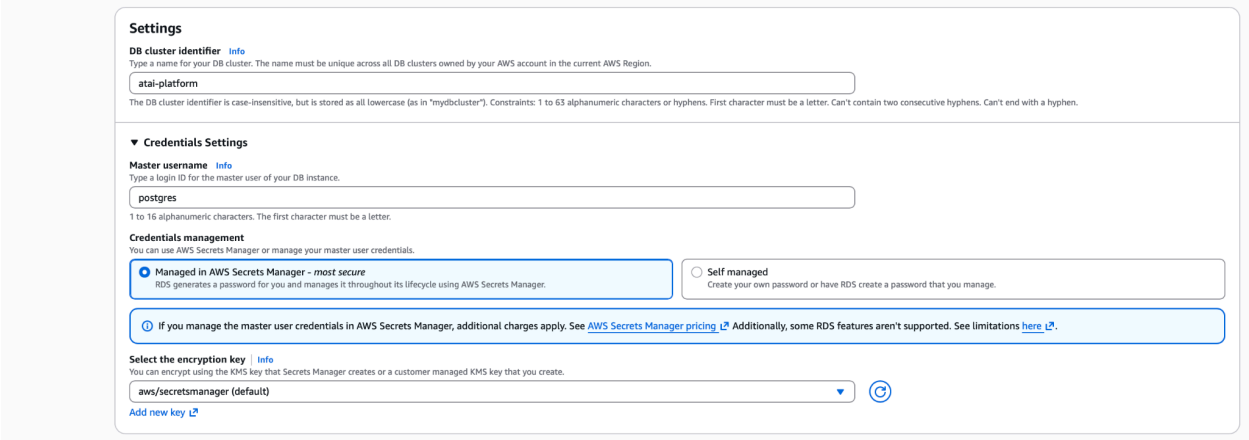


2. Database creation method: Full configuration
3. Engine options:
4. Engine type: Amazon Aurora
    a. Edition: Amazon Aurora PostgreSQL-Compatible Edition
    b. Available versions: Select Aurora PostgreSQL 17.5
    c. Templates: Production (or Dev/Test for non-production)

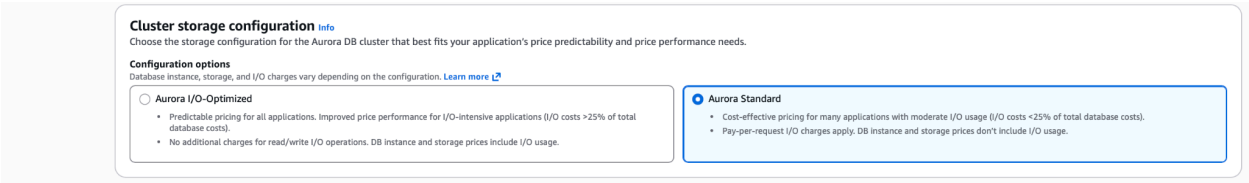




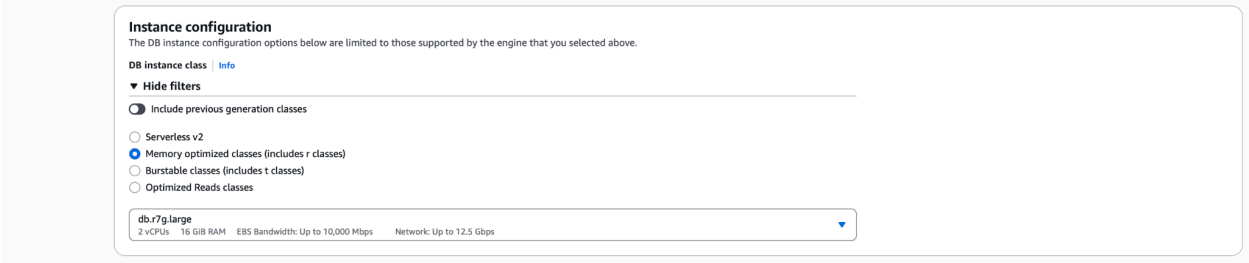
5. Settings:
    a. DB cluster identifier: atai-platform
    b. Master username: postgres (or your preferred username)
    c. Credentials management: Managed in AWS secrets manager



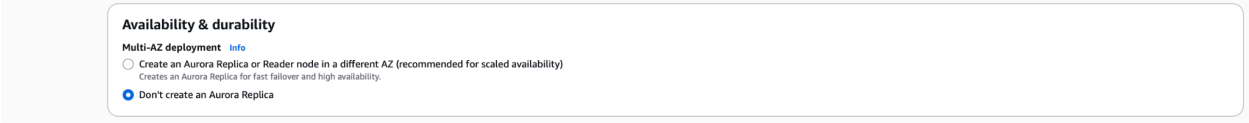
6. Cluster storage configuration
    a. Storage type: Aurora Standard



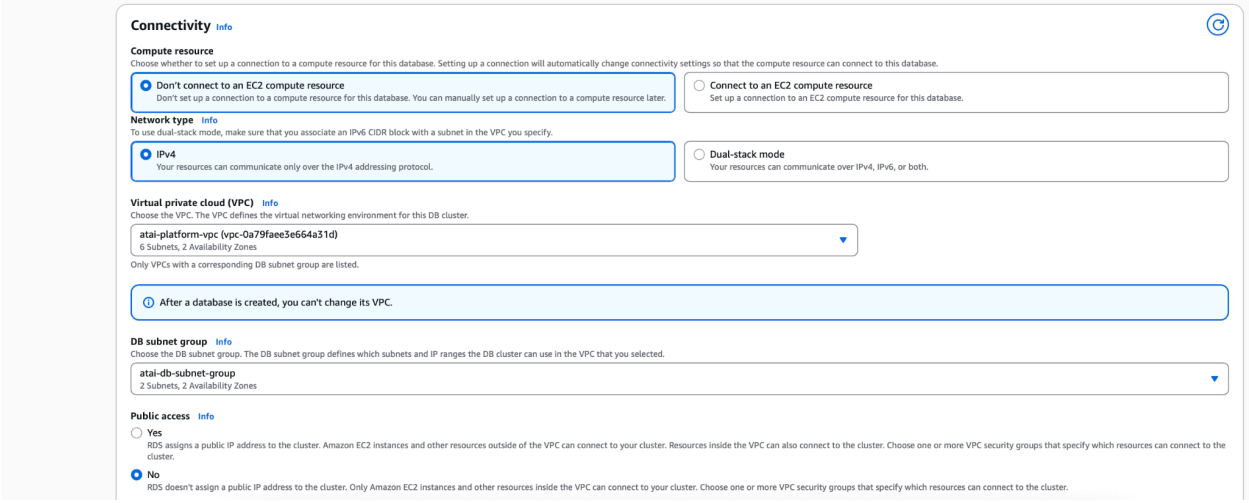
7. Instance configuration
    a. DB instance class: Select your instance class (e.g., db.t4g.medium for dev, db.r7g.large for production)



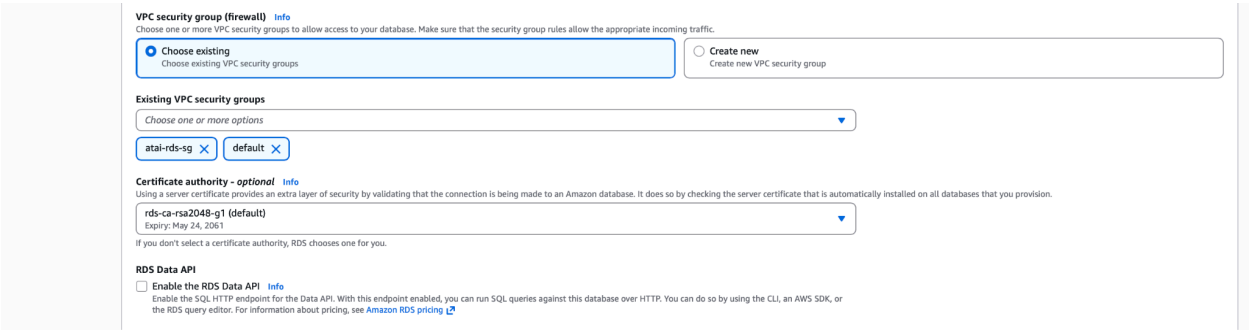
8. For Availability and Durability select Don’t create an Aurora Replica



9. Connectivity:
    a. Virtual private cloud (VPC): Select your VPC
    b. DB subnet group: Select your database subnet group (created in Step 1)
    c. Publicly accessible: No (should be in private subnets)

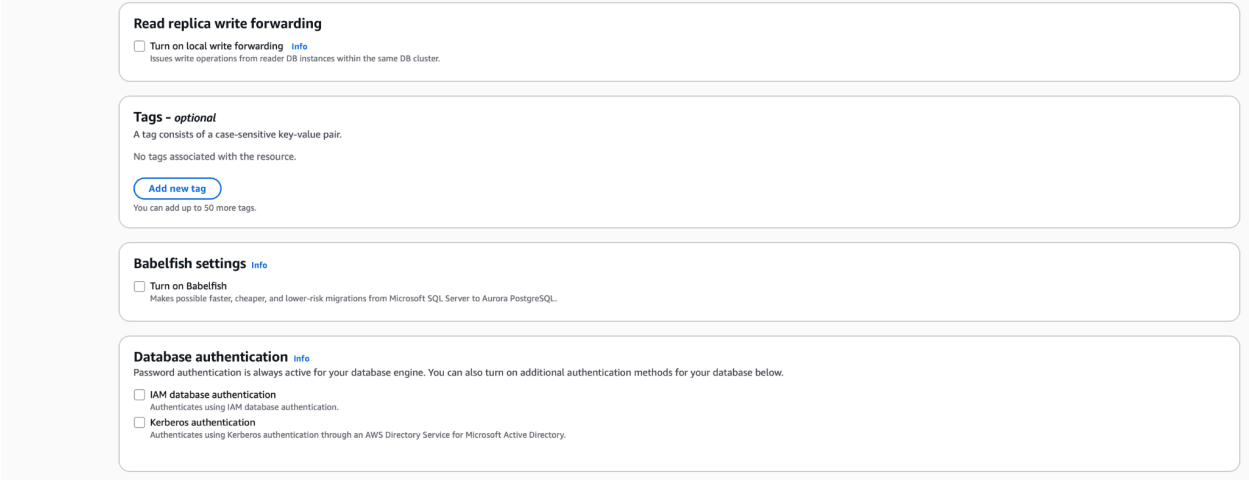


    d. VPC security group: Choose existing → Select atai-rds-sg
    e. Availability Zone: No preference (AWS will choose) or select your preferred AZ for the primary instance

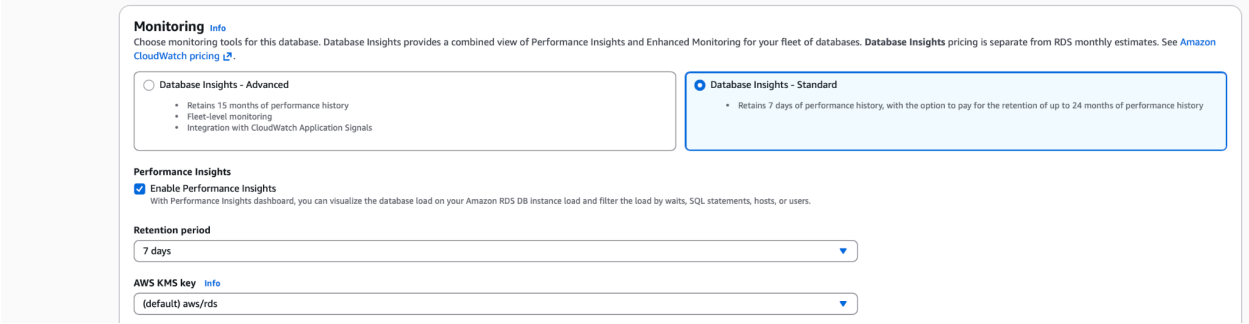


Note: For low latency, you can select the same AZ as your EKS node groups (e.g., us-west-2a)

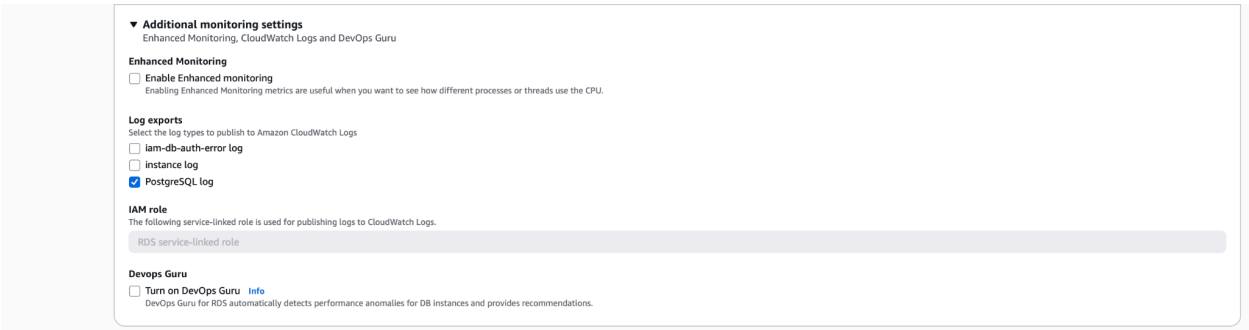
10. Leave the default values for Read replica write forwarding, Tags, Babelfish settings and Database authentication.



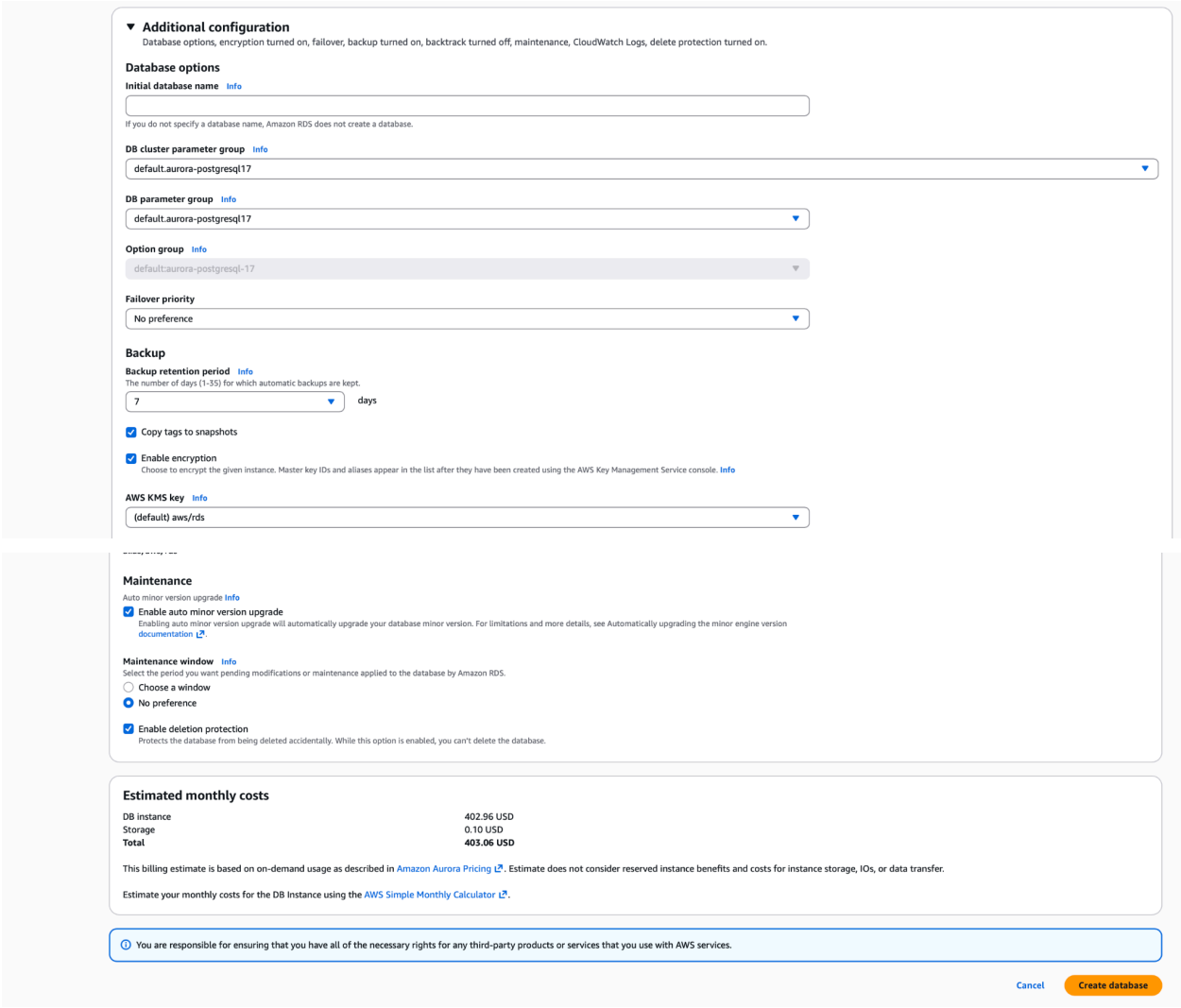
11. **Monitoring**
    a. Select Database insight - Standard



    b. Disable Enhanced monitoring
    c. Select PostgreSQL log under the Logs exports section
    d. You can turn off Devops Guru to avoid extra costs



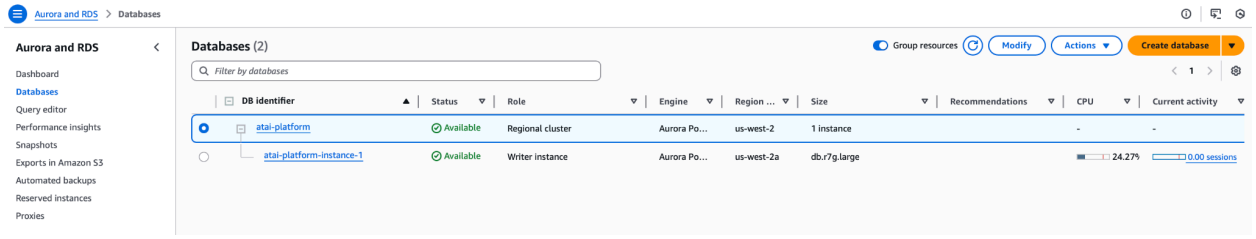
12. Additional configuration:
    a. Initial database name: Leave blank (or enter a name if you want to create an initial database)
    b. DB cluster parameter group: default.aurora-postgresql17 (or create custom if needed)
    c. DB parameter group: default.aurora-postgresql17 (or create custom if needed)
    d. Backup retention period: 7 days (default, or your preference)
    e. Backup window: No preference (default, or set a specific window)
    f. Enable encryption: Enable encryption (it's enabled by default)
    g. Encryption key: Use AWS managed key (default) or your KMS key



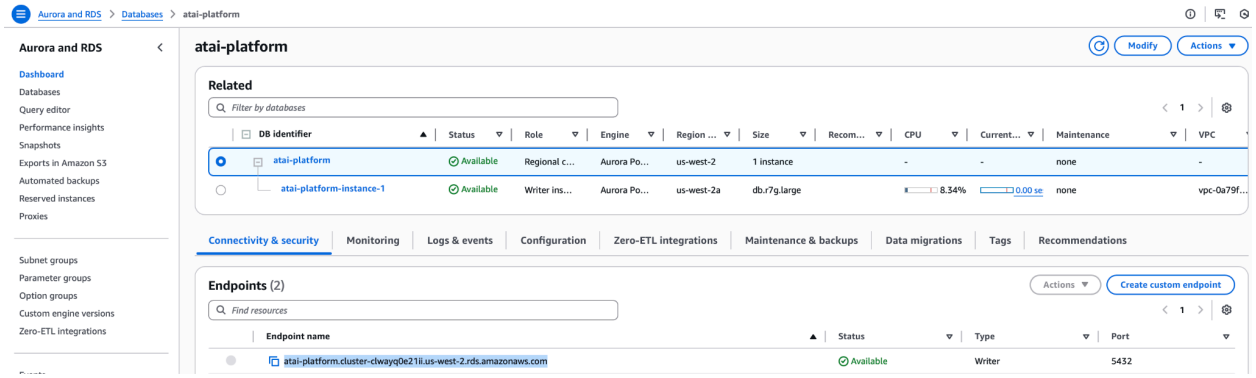
Note: No additional configurations are required — defaults work. You can customize backup retention, monitoring, and other settings if needed.

13. Click on **Create database**

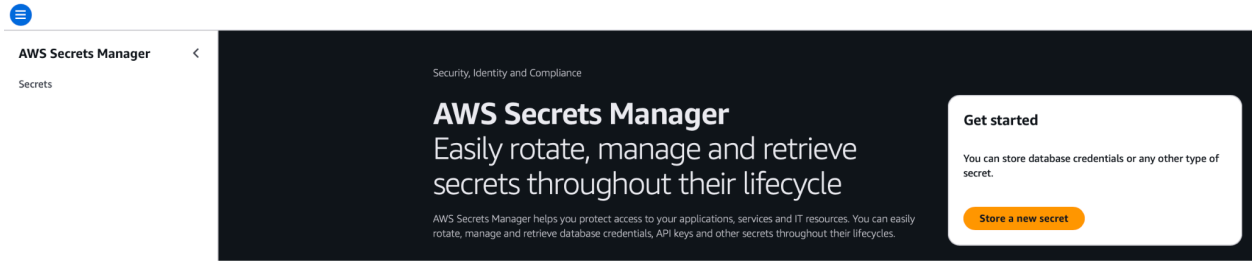
14. Get my database hostname
    a. After you create your database go to the RDS dashboard → **Databases**
    b. Select your database cluster



    c. Save the **Cluster endpoint** - will be needed for the helm values file

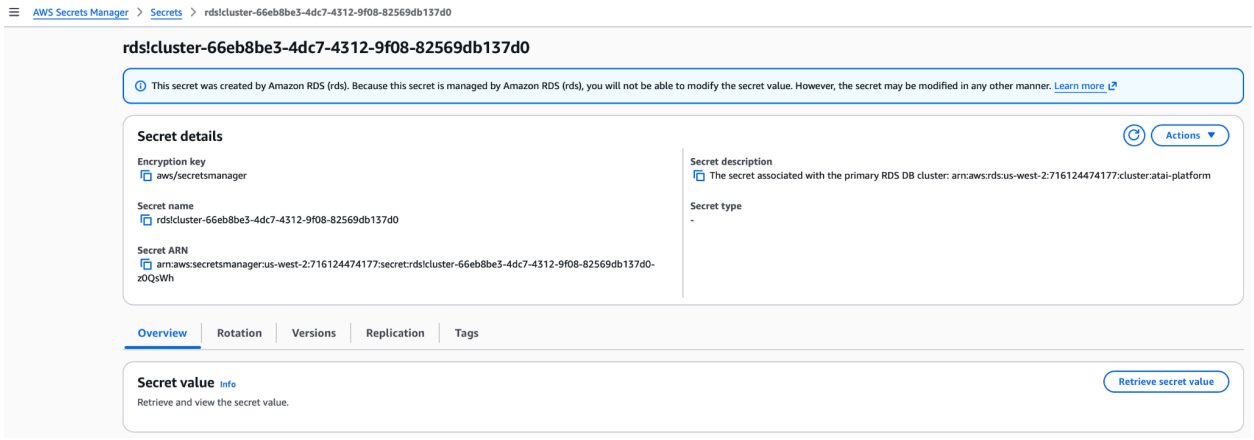


15. Get my database credentials
    a. Go to Secrets Manager → **Secrets**



    b. Click on the new RDS secret

    c. Click on **Get secret value**. You'll be able to see the user and password to connect to your RDS postgreSQL.



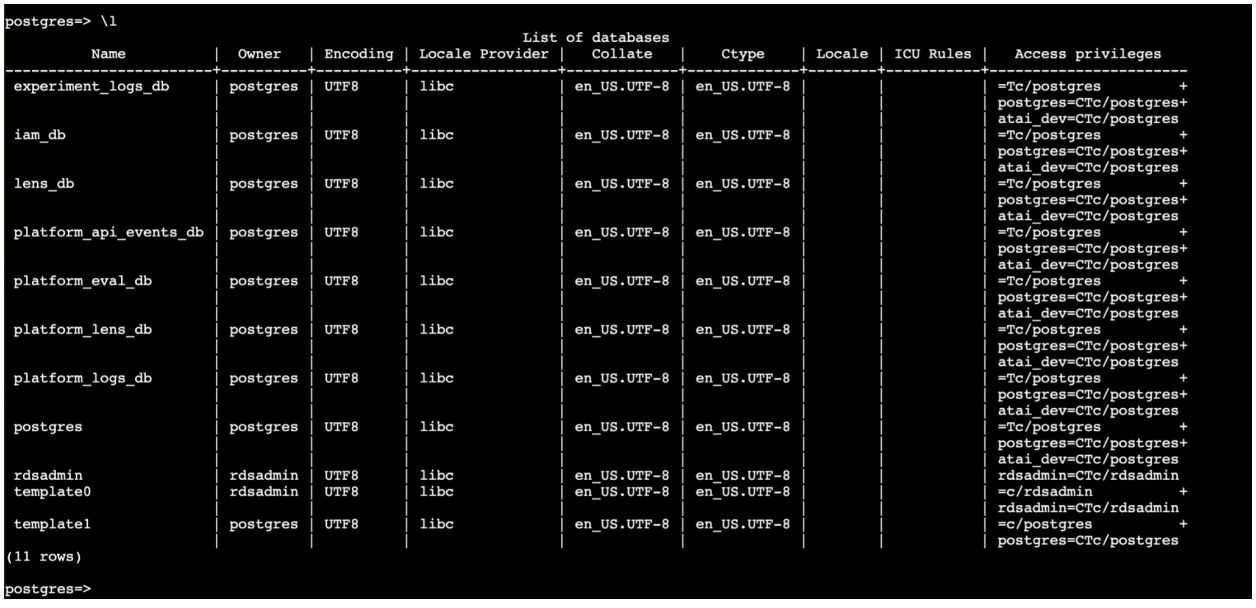
Step 4: Extra Database Configuration steps

Connect to your RDS, in our case we are using bastion host machine:

```
None
psql -h your-rds-endpoint.region.rds.amazonaws.com -U master_username -d postgres
```

Create databases

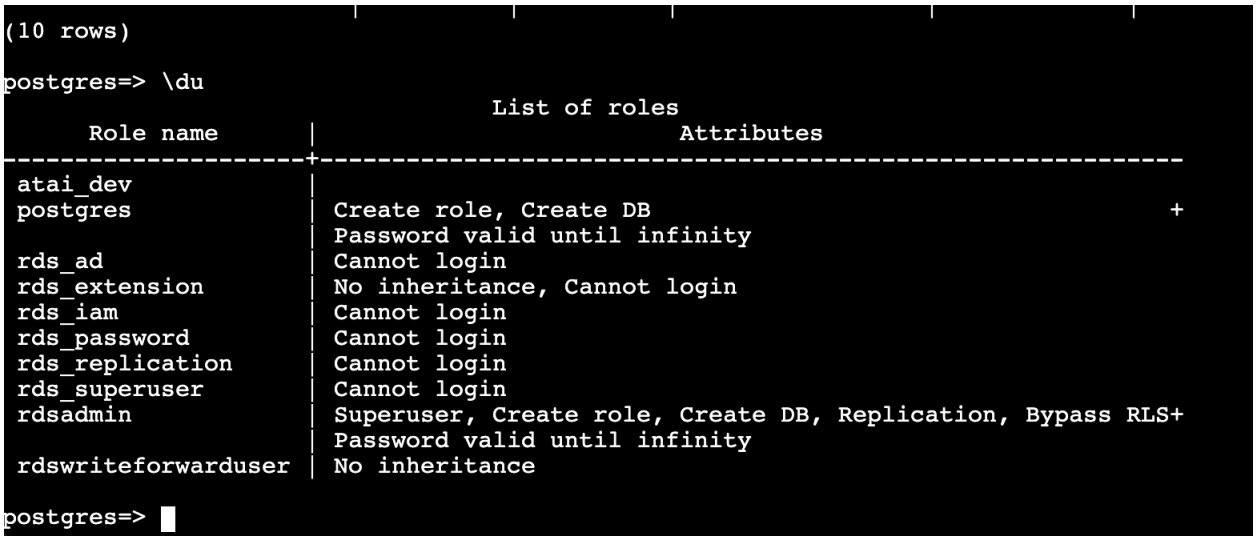
```
None
CREATE DATABASE experiment_logs_db;
CREATE DATABASE iam_db;
CREATE DATABASE platform_api_events_db;
CREATE DATABASE platform_eval_db;
CREATE DATABASE platform_lens_db;
CREATE DATABASE platform_logs_db;
CREATE DATABASE lens_db;
CREATE DATABASE data_service;
CREATE DATABASE jos_db;
```



```
postgres=> \l
                                                    List of databases
        Name         |  Owner   | Encoding | Locale Provider |   Collate   |    Ctype    | Locale | ICU Rules |   Access privileges
---------------------+----------+----------+-----------------+-------------+-------------+--------+-----------+-----------------------
 experiment_logs_db  | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =Tc/postgres         +
                     |          |          |                 |             |             |        |           | postgres=CTc/postgres+
                     |          |          |                 |             |             |        |           | atai_dev=CTc/postgres
 iam_db              | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =Tc/postgres         +
                     |          |          |                 |             |             |        |           | postgres=CTc/postgres+
                     |          |          |                 |             |             |        |           | atai_dev=CTc/postgres
 lens_db             | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =Tc/postgres         +
                     |          |          |                 |             |             |        |           | postgres=CTc/postgres+
                     |          |          |                 |             |             |        |           | atai_dev=CTc/postgres
 platform_api_events_db | postgres | UTF8  | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =Tc/postgres         +
                     |          |          |                 |             |             |        |           | postgres=CTc/postgres+
                     |          |          |                 |             |             |        |           | atai_dev=CTc/postgres
 platform_eval_db    | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =Tc/postgres         +
                     |          |          |                 |             |             |        |           | postgres=CTc/postgres+
                     |          |          |                 |             |             |        |           | atai_dev=CTc/postgres
 platform_lens_db    | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =Tc/postgres         +
                     |          |          |                 |             |             |        |           | postgres=CTc/postgres+
                     |          |          |                 |             |             |        |           | atai_dev=CTc/postgres
 platform_logs_db    | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =Tc/postgres         +
                     |          |          |                 |             |             |        |           | postgres=CTc/postgres+
                     |          |          |                 |             |             |        |           | atai_dev=CTc/postgres
 postgres            | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =Tc/postgres         +
                     |          |          |                 |             |             |        |           | postgres=CTc/postgres+
                     |          |          |                 |             |             |        |           | atai_dev=CTc/postgres
 rdsadmin            | rdsadmin | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | rdsadmin=CTc/rdsadmin
 template0           | rdsadmin | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =c/rdsadmin          +
                     |          |          |                 |             |             |        |           | rdsadmin=CTc/rdsadmin
 template1           | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =c/postgres          +
                     |          |          |                 |             |             |        |           | postgres=CTc/postgres
(11 rows)

postgres=>
```

Create atai_dev database user

```
None
CREATE USER atai_dev WITH PASSWORD 'your_secure_password';
```



⚠️ Store your atai_dev user and password in a secure location. You will need them later in the *atai-platform* prerequisites.

Then assign proper permissions to the new PostgreSQL user:

```
None
GRANT ALL PRIVILEGES ON DATABASE experiment_logs_db TO atai_dev;
GRANT ALL PRIVILEGES ON DATABASE iam_db TO atai_dev;
GRANT ALL PRIVILEGES ON DATABASE platform_api_events_db TO atai_dev;
GRANT ALL PRIVILEGES ON DATABASE platform_eval_db TO atai_dev;
GRANT ALL PRIVILEGES ON DATABASE platform_lens_db TO atai_dev;
GRANT ALL PRIVILEGES ON DATABASE platform_logs_db TO atai_dev;
GRANT ALL PRIVILEGES ON DATABASE lens_db TO atai_dev;
GRANT ALL PRIVILEGES ON DATABASE postgres TO atai_dev;
GRANT ALL PRIVILEGES ON DATABASE data_service TO atai_dev;
GRANT ALL PRIVILEGES ON DATABASE jos_db TO atai_dev;
```

Additional permissions required:

```
None
\c database_name (this should be done on all DBs)

GRANT USAGE ON SCHEMA public TO atai_dev;
GRANT CREATE ON SCHEMA public TO atai_dev;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO atai_dev;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO atai_dev; ALTER DEFAULT
PRIVILEGES IN SCHEMA public GRANT ALL ON TABLES TO atai_dev;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL ON SEQUENCES TO atai_dev;
GRANT ALL PRIVILEGES ON ALL FUNCTIONS IN SCHEMA public TO atai_dev; ALTER DEFAULT
PRIVILEGES IN SCHEMA public GRANT ALL ON FUNCTIONS TO atai_dev;
```

Solve a common permission issue:

```
None
-- Issue:psycopg2.errors.InsufficientPrivilege:
--   permission denied for schema public
---  LINE 1: CREATE TABLE IF NOT EXISTS platform_lens_db(
--   Fix: on that specific DB

GRANT CREATE ON SCHEMA public TO atai_dev;
GRANT USAGE ON SCHEMA public TO atai_dev;
```

EKS cluster configuration

Prerequisites

1. An existing VPC and subnets that meet Amazon EKS requirements
2. At least two private subnets in your VPC:
    a. Recommended naming:
        i. atai-platform-vpc-private-us-west-2a,
        ii. atai-platform-vpc-private-us-west-2b (or similar)
    b. Recommended CIDR blocks: /20 per subnet
        i. Example: 10.5.0.0/20 (subnet 1)
        ii. Example: 10.5.16.0/20 (subnet 2)
    c. Subnets must be in different Availability Zones
    d. Subnets must have routes to NAT Gateway or Internet Gateway for outbound internet access
3. The kubectl command line tool v1.32-v1.34 is required.

4. Version 2.12.3 or later or version 1.27.160 or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device.
5. An IAM principal with permissions to create and describe an Amazon EKS cluster

Step 1: Create cluster IAM role

1. If you already have a cluster IAM role, or you're going to create your cluster with eksctl, then you can skip this step. By default, eksctl creates a role for you.
2. Run the following command to create an IAM trust policy JSON file.

```
None
cat > eks-cluster-role-trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

3. Create the Amazon EKS cluster IAM role. If necessary, preface eks-cluster-role-trust-policy.json with the path on your computer that you wrote the file to in the previous step. The command associates the trust policy that you created in the previous step to the role. To create an IAM role, the IAM principal that is creating the role must be assigned the iam:CreateRole action (permission).

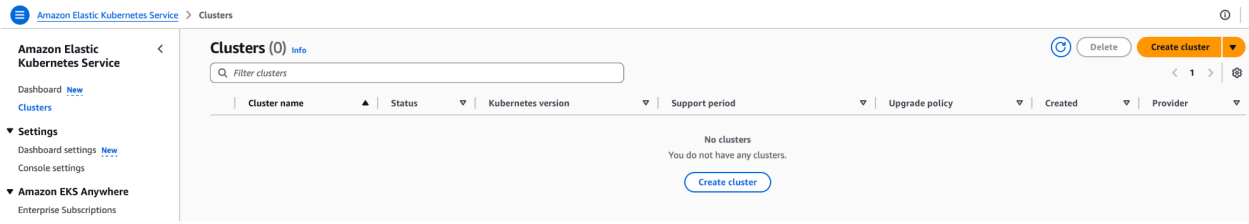
```
None
aws iam create-role --role-name atai-platform-eks-cluster-role
--assume-role-policy-document file://"eks-cluster-role-trust-policy.json"
```

4. You can assign either the Amazon EKS managed policy or create your own custom policy. For the minimum permissions that you must use in your custom policy, see Amazon EKS cluster IAM role.
   Attach the Amazon EKS managed policy named AmazonEKSClusterPolicy to the role. To attach an IAM policy to an IAM principal, the principal that is attaching the policy must be assigned one of the following IAM actions (permissions): iam:AttachUserPolicy or iam:AttachRolePolicy.

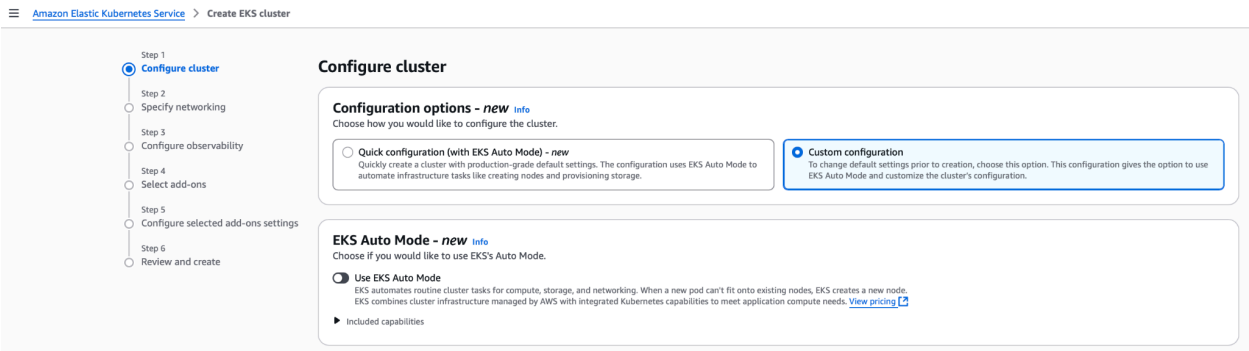
```
None
aws iam attach-role-policy --policy-arn
arn:aws:iam::aws:policy/AmazonEKSClusterPolicy --role-name
atai-platform-eks-cluster-role
```

Step 2: Create cluster

1. Open EKS Console → Click on **Create cluster**

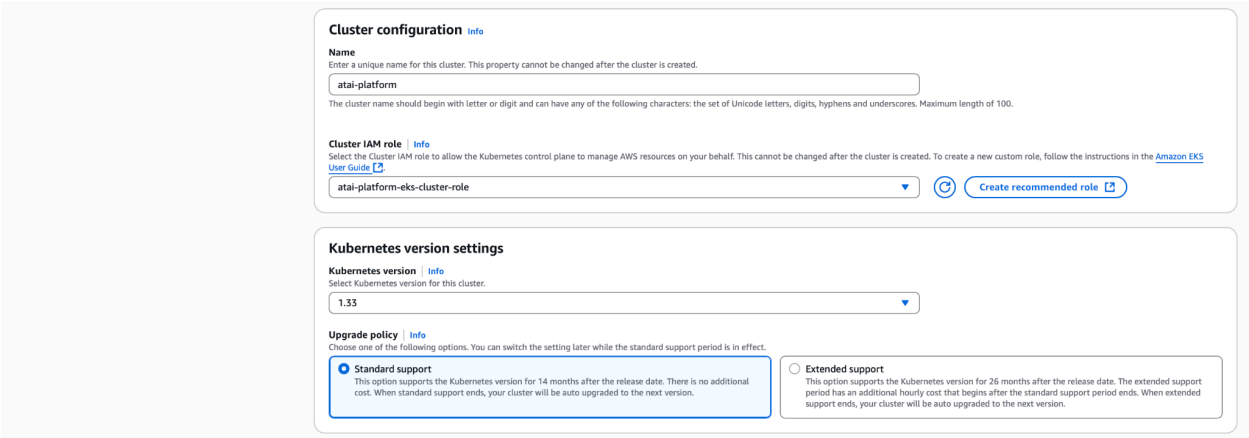


2. Under Configuration options select Custom configuration
3. Under EKS Auto Mode, toggle Use EKS Auto Mode off.

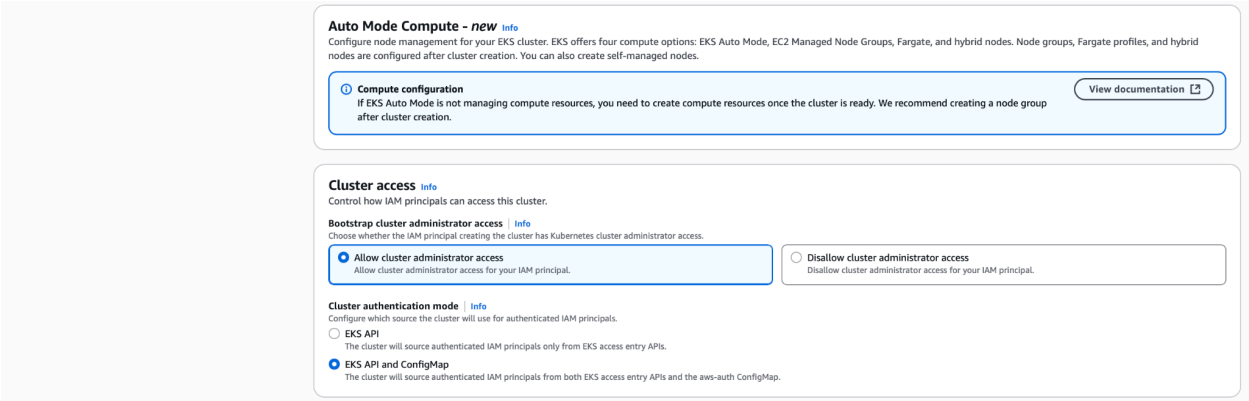


4. On the Configure cluster page, enter the following fields:
    a. Name: atai-platform

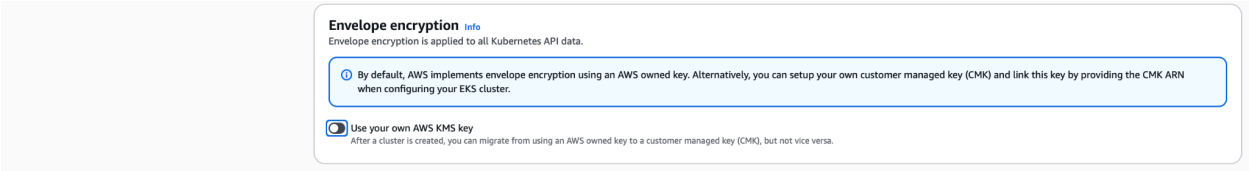
b. Cluster IAM role – Choose the Amazon EKS cluster IAM role that you created in the Step 1 to allow the Kubernetes control plane to manage AWS resources on your behalf.
c. Kubernetes version: 1.33
d. Support type: Standard support



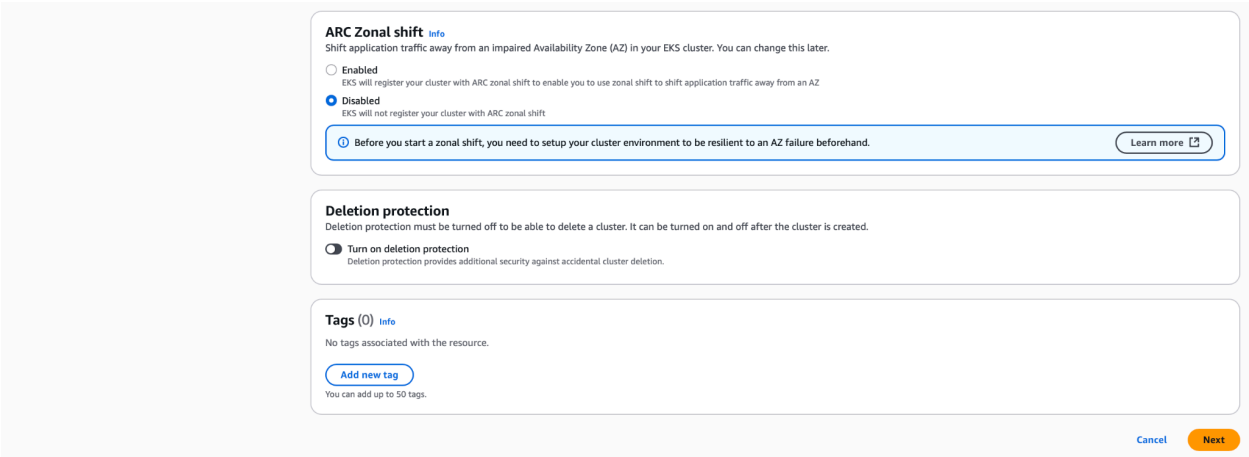
5. Cluster access
    a. Select Allow cluster administrator access
    b. Cluster authentication mode: EKS API and ConfigMap



6. Envelop encryption
    a. By default, AWS implements envelope encryption using an AWS owned key. Alternatively, you can setup your own customer managed key (CMK) and link this key by providing the CMK ARN when configuring your EKS cluster.



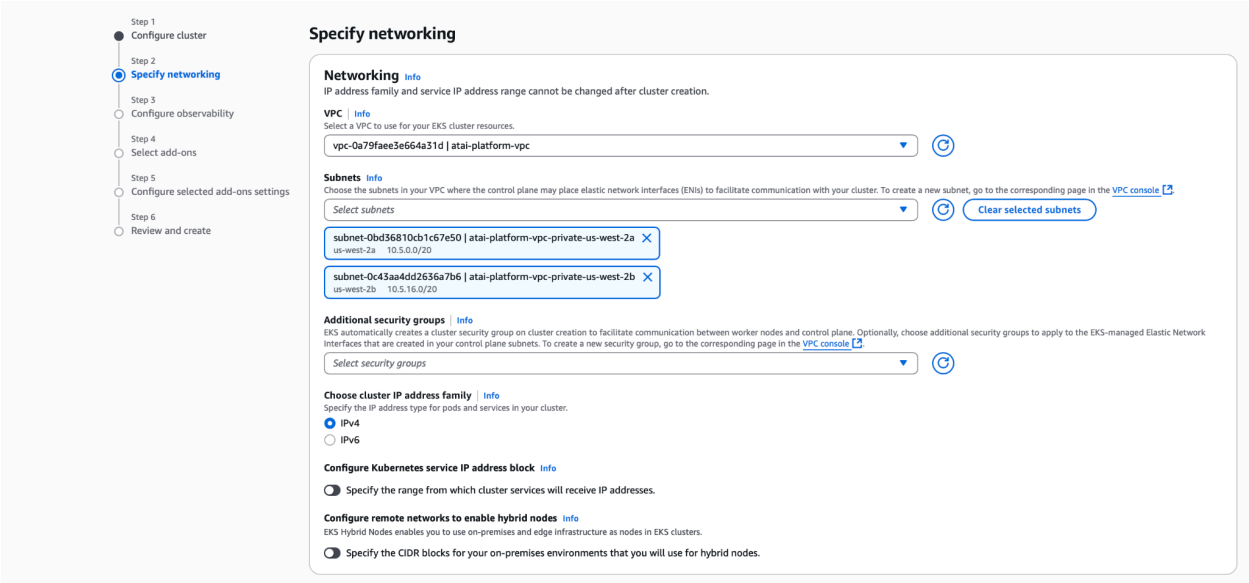
7. Use the default configuration for ARC Zonal Shift (disabled by default).
   (Optional) Enable Deletion protection
   (Optional) Add tags
   Click on **Next**



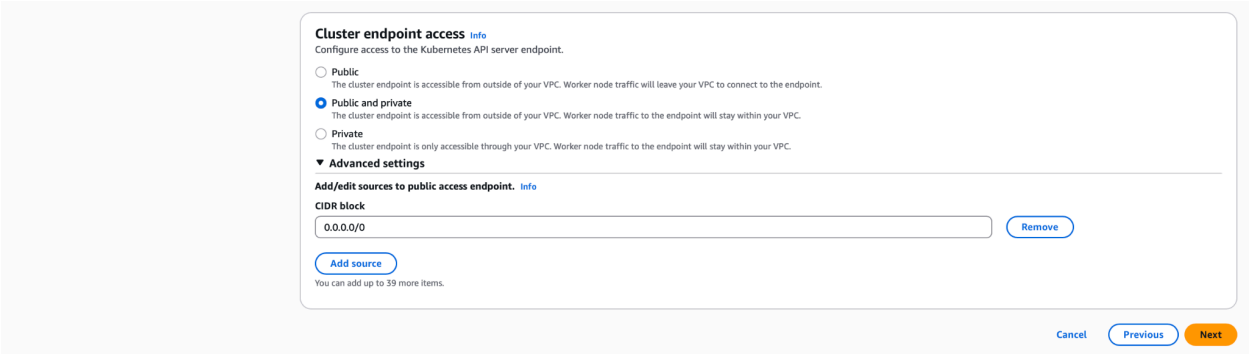
8. Networking
   a. VPC: Select your VPC
   b. Subnets: Select two private subnets:
      i. atai-platform-vpc-private-us-west-2a (10.5.0.0/20)
      ii. atai-platform-vpc-private-us-west-2b (10.5.16.0/20)
   c. (Optional) Additional security groups: EKS automatically creates a cluster security group on cluster creation to facilitate communication between worker nodes and control plane but you can add others as needed.

After the cluster is created, you must **retrieve the security group ID** assigned by AWS.
This ID is required when creating Launch Templates for your Managed Node Groups, to ensure proper networking and access between the control plane and the worker nodes. To learn how to get the security group automatically created by EKS got to Step 4: Get the default cluster security group

   d. Cluster IP address family: IPv4



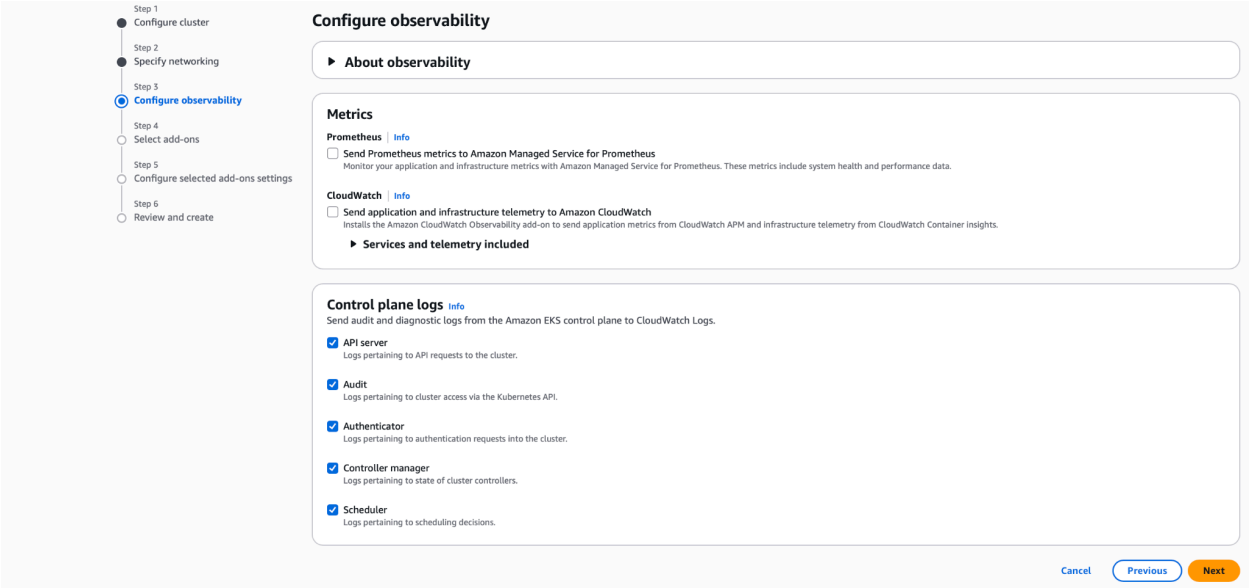
9. Cluster endpoint access
    a. Public and private
    b. Click on **Next**



10. Configure observability page:
    a. Control plane logging: Enable all:
        i. API server
        ii. Audit

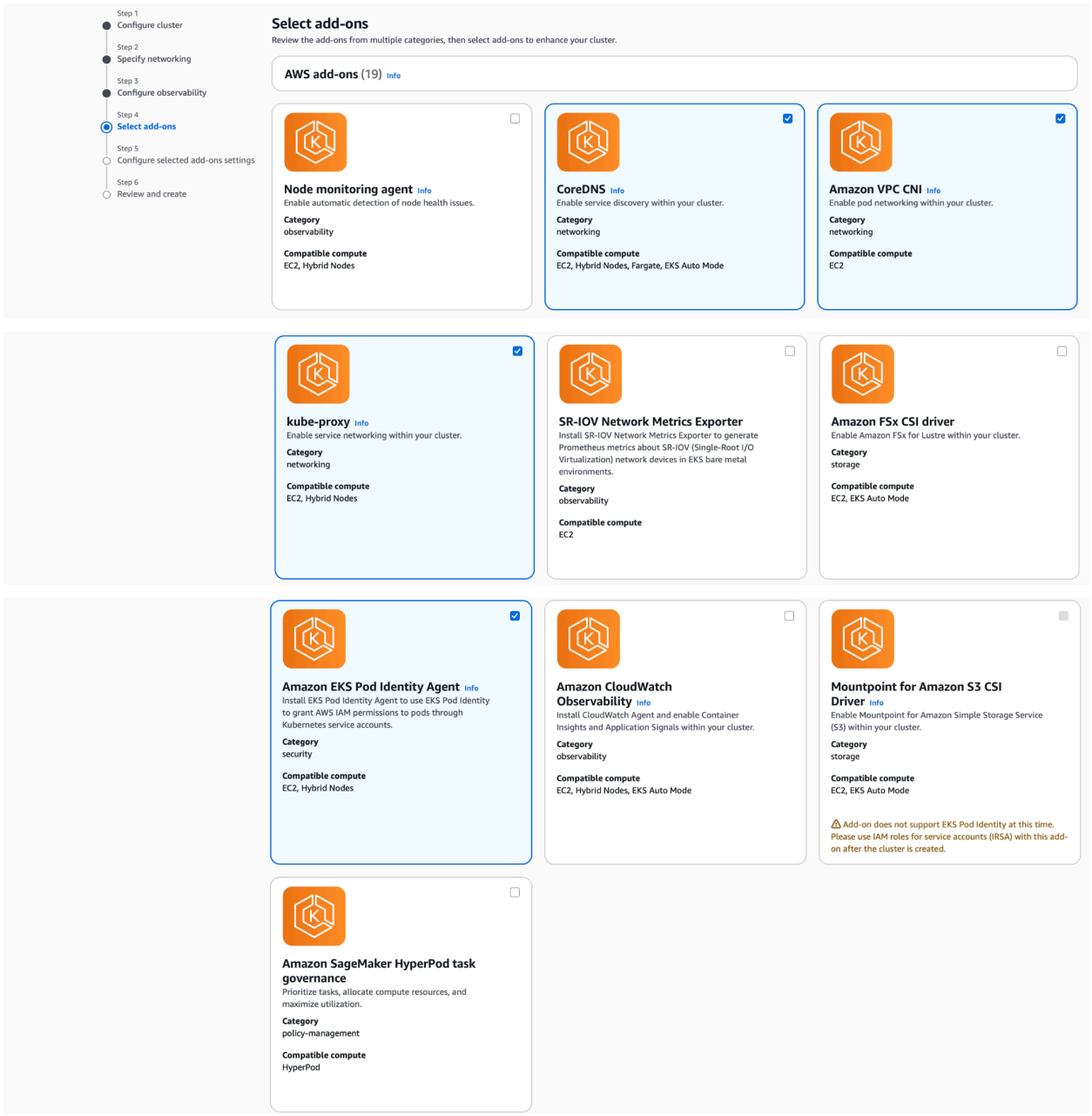
iii. Authenticator
iv. Controller manager
v. Scheduler

b. Prometheus metrics: Optional



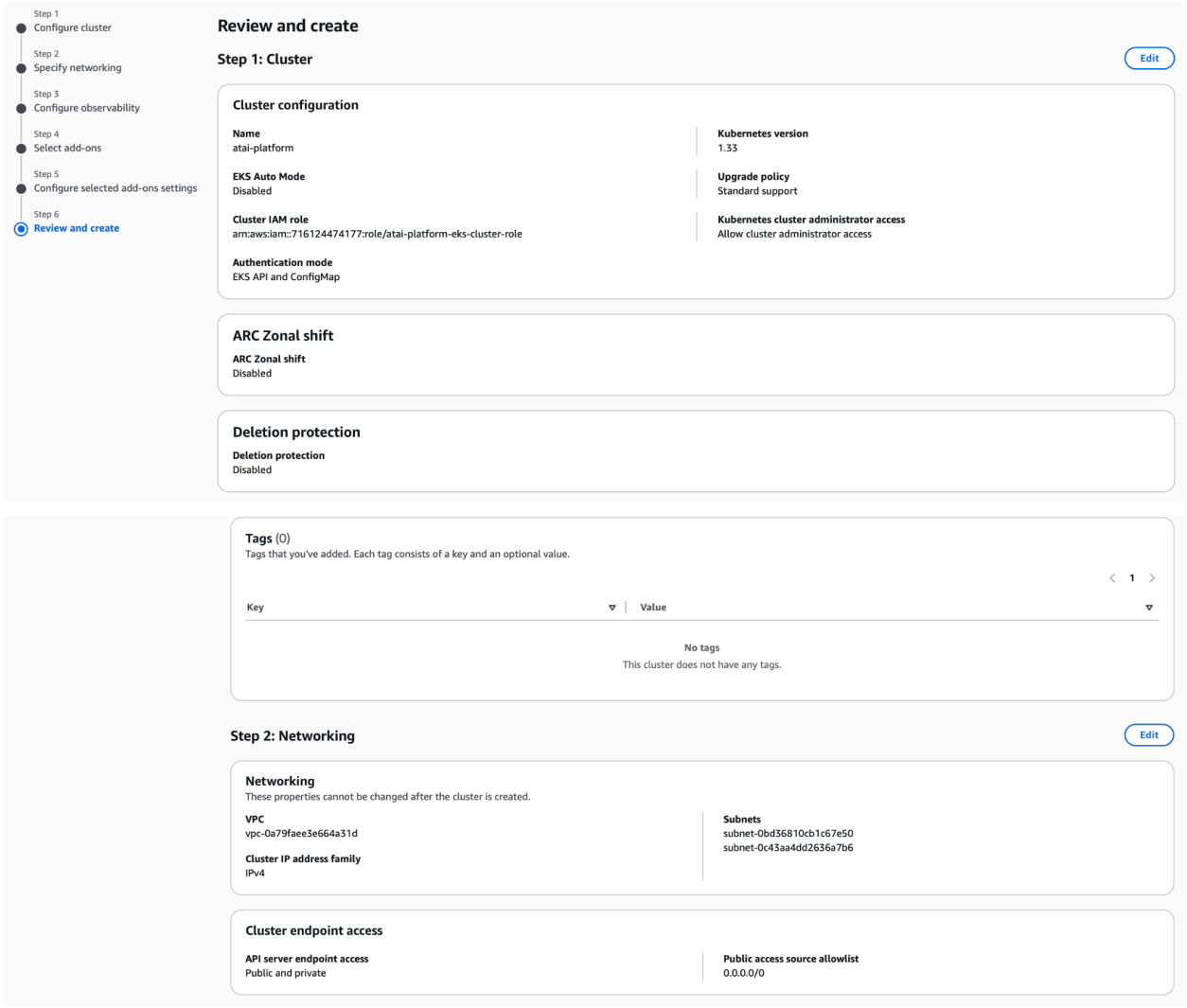
11. Select add-ons: Select these add-ons:
    a. Amazon VPC CNI (required)
    b. CoreDNS (required)
    c. kube-proxy (required)

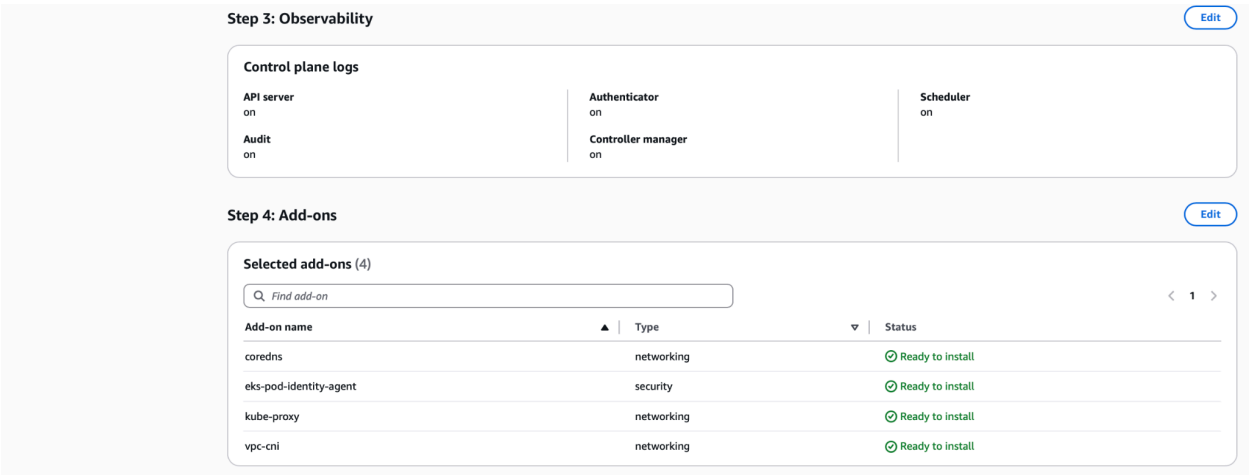
    Click Next

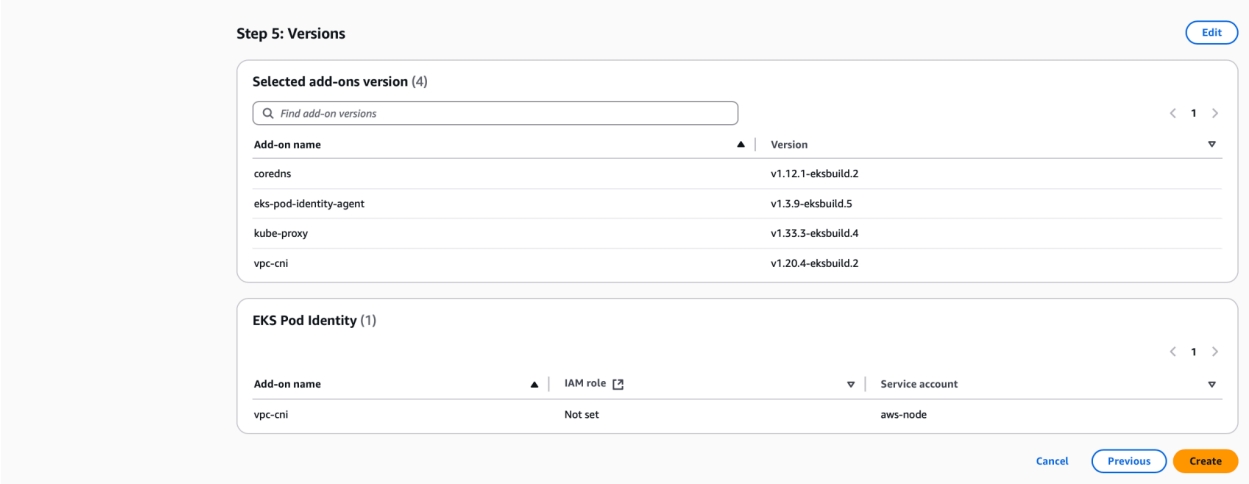


12. Configure selected add-ons settings
    a. VPC CNI: v1.20.4-eksbuild.2 or Default/Current version
    b. CoreDNS: v1.12.1-eksbuild.2 or Default/Current version
    c. kube-proxy: v1.33.3-eksbuild.4 or Default/Current version

13. Review your cluster configuration and click on **Create**







Step 3: Update kubeconfig

1. Enable kubectl to communicate with your cluster by adding a new context to the kubectl config file.

```
None
aws eks update-kubeconfig --region region-code --name atai-platform
```

An example output is as follows.

```
None
Added new context arn:aws:eks:region-code:111122223333:cluster/atai-platform to
/home/username/.kube/config
```

2. Confirm communication with your cluster by running the following command.

```
None
kubectl get svc
```

An example output is as follows.

```
None
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   28h
```

Step 4: Get the default cluster security group

After the cluster is created, you must **retrieve the security group ID** assigned by AWS. This ID is required when creating Launch Templates for your Managed Node Groups, to ensure proper networking and access between the control plane and the worker nodes.

You can determine the ID of your cluster security group in the AWS Management Console under the cluster's Networking section. Or, you can do so by running the following AWS CLI command.

```
None
aws eks describe-cluster --name  atai-platform --query
cluster.resourcesVpcConfig.clusterSecurityGroupId
```

EKS managed node group configuration

Prerequisites

1. EKS cluster is ACTIVE
2. IAM role for node groups created (see Step 1 below)

3. Security group for nodes (see Step 2 below)
4. An existing VPC and subnets that meet Amazon EKS requirements
5. One private subnets in your VPC:
    a. Recommended naming:
        i. atai-platform-vpc-private-us-west-2a,
    b. Recommended CIDR blocks: /20 per subnet
        i. Example: 10.5.0.0/20 (subnet 1)
    c. Subnets must have routes to NAT Gateway or Internet Gateway for outbound internet access
6. The kubectl command line tool is required. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster.
7. Version 2.12.3 or later or version 1.27.160 or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device.
8. An IAM principal with permissions to create and describe an Amazon EKS cluster, and create Managed Node Groups.

Step 1:Creating the Amazon EKS node IAM role

1. Run the following command to create an IAM trust policy JSON file.

```
None
cat > node-role-trust-relationship.json << 'EOF'
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
EOF
```

2. Create the IAM role.

```
None
aws iam create-role \
  --role-name atai-platform-eks-node-role \
  --assume-role-policy-document file://"node-role-trust-relationship.json"
```

3. Attach three required IAM managed policies to the IAM role.

```
None
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name atai-platform-eks-node-role

aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly \
  --role-name atai-platform-eks-node-role

aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name atai-platform-eks-node-role
```

4. Attach the following IAM policy to the IAM role:

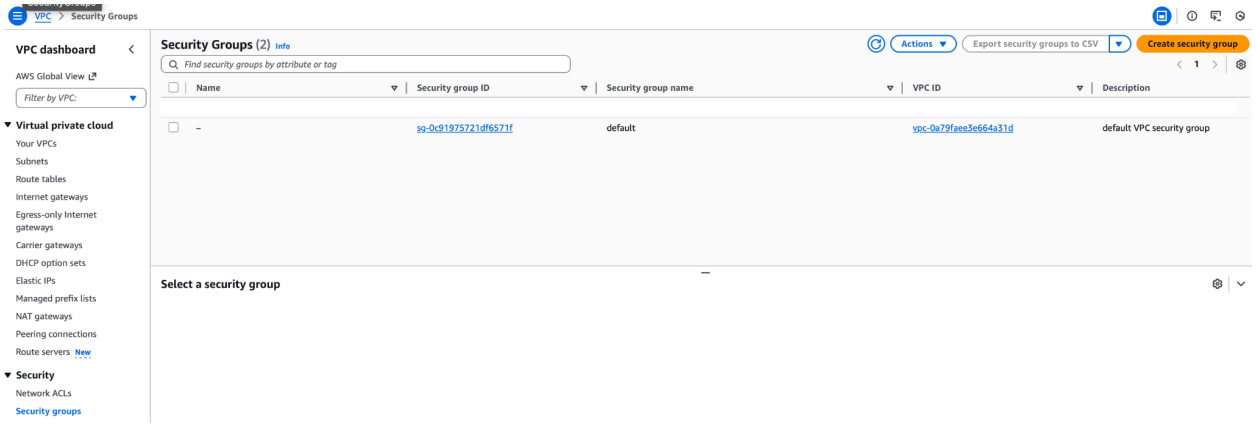
```
None
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name atai-platform-eks-node-role
```

5. Attach the following IAM policy to the IAM role to allow SSH access to the nodes through AWS Session Manager (SSM) for debugging purposes.

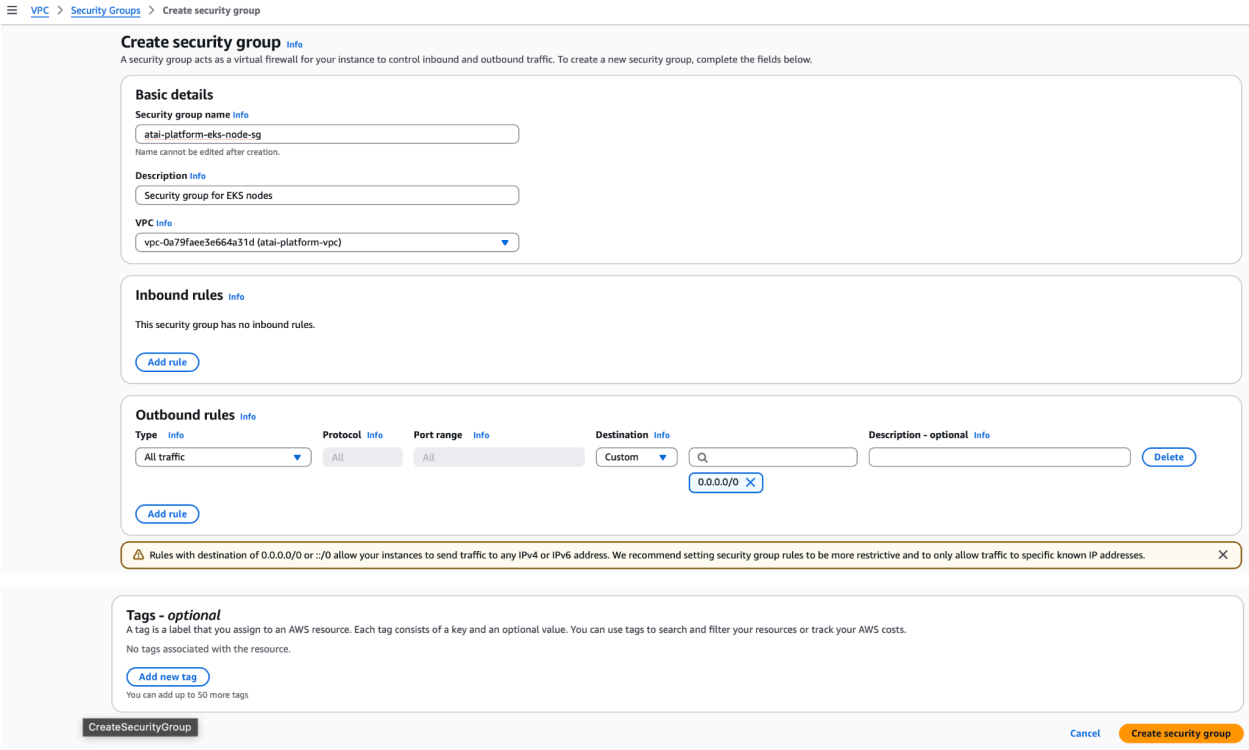
```
None
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore \
  --role-name atai-platform-eks-node-role
```

Step 2: Creating Node Security Group

6. Go to VPC → Security Groups → Create security group



7. Name: atai-platform-eks-node-sg (or your name)
8. Description: Security group for EKS nodes
9. VPC: Select your VPC from section VPC configuration Step 1

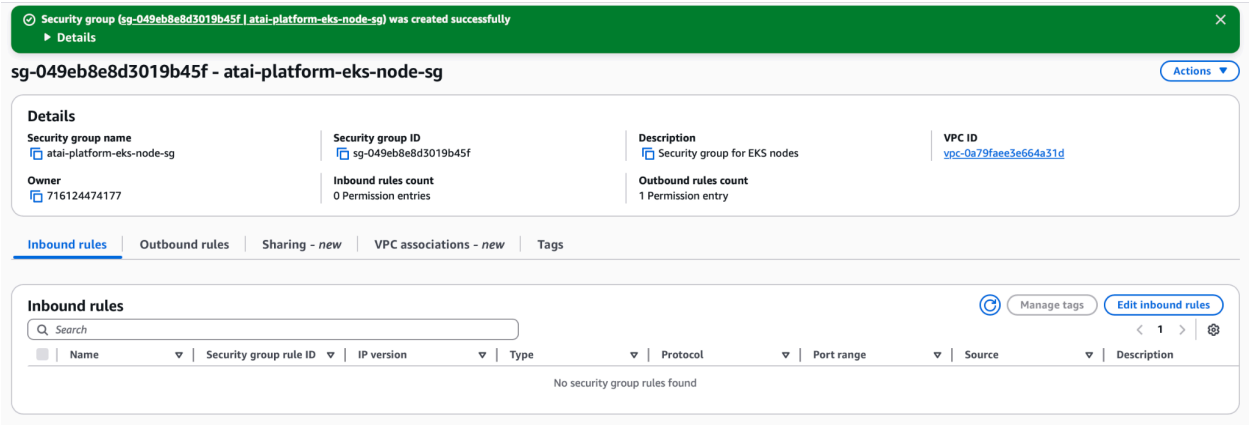


10. Click on **Create security group**

Step 2.1 Add self rules to the node security group

A "self rule" in a security group allows traffic from the same security group.

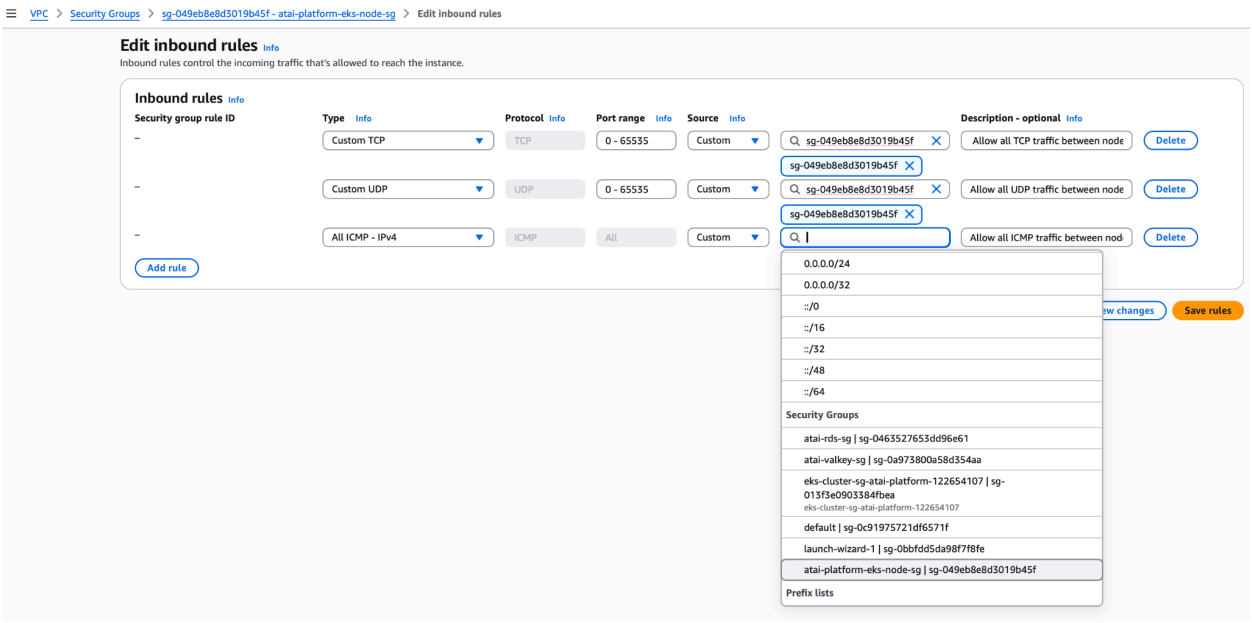
1. Select your security group configured in Step 2 and click on **Edit inbound rules**



2. Inbound rules: Add rule:

a. Rule 1: All TCP
    i. Type: All TCP
    ii. Source: Custom
        1. In the Source field, select Custom (not CIDR blocks or IP addresses).
        2. Click on the blank textbox next to Source
        3. Navigate to Security groups.
        4. Select the same security group you're currently configuring.
    iii. Description: Allow all TCP traffic between nodes
b. Rule 2: All UDP
    i. Type: All UDP
    ii. Source: Custom
        1. In the Source field, select Custom (not CIDR blocks or IP addresses).
        2. Click on the blank textbox next to Source
        3. Navigate to Security groups.
        4. Select the same security group you're currently configuring.
    iii. Description: Allow all UDP traffic between nodes
c. Rule 3: ICMP
    i. Type: All ICMP - IPv4
    ii. Port: None
    iii. Source: Custom
        1. In the Source field, select Custom (not CIDR blocks or IP addresses).
        2. Click on the blank textbox next to Source
        3. Navigate to Security groups.
        4. Select the same security group you're currently configuring.
    iv. Description: Allow all ICMP traffic between nodes

3. Click on **Save rules**



Note: Creating Self-Referencing Security Group Rules

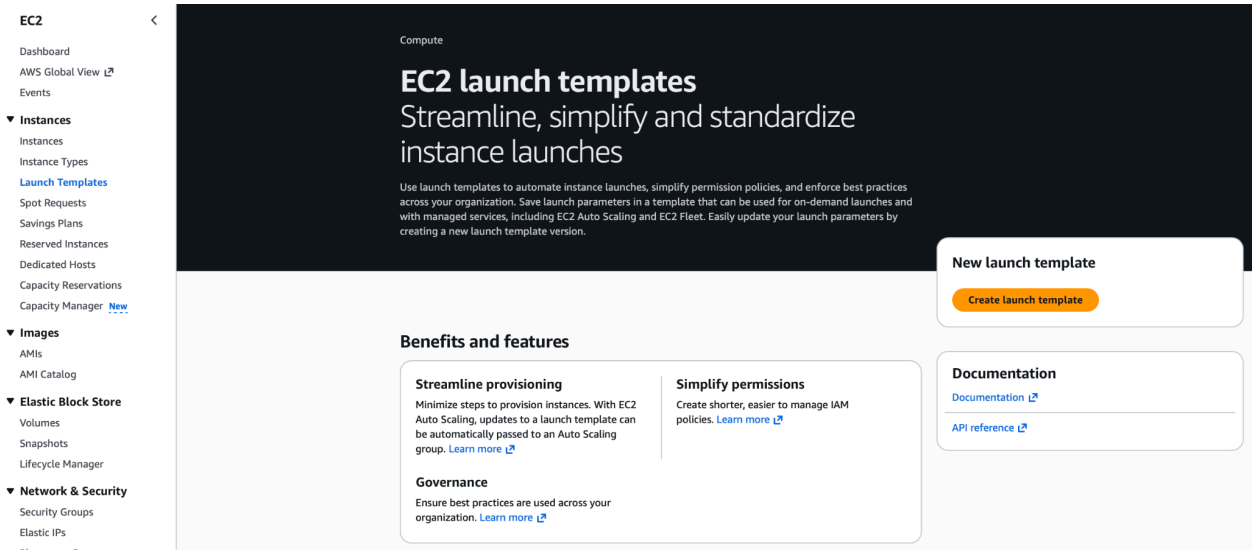
When adding ingress rules that allow traffic from the same security group (self-rule):

1. In the Source field, select Custom (not CIDR blocks or IP addresses).
2. Open the dropdown menu.
3. Navigate to Security groups.
4. Select the same security group you're currently configuring.

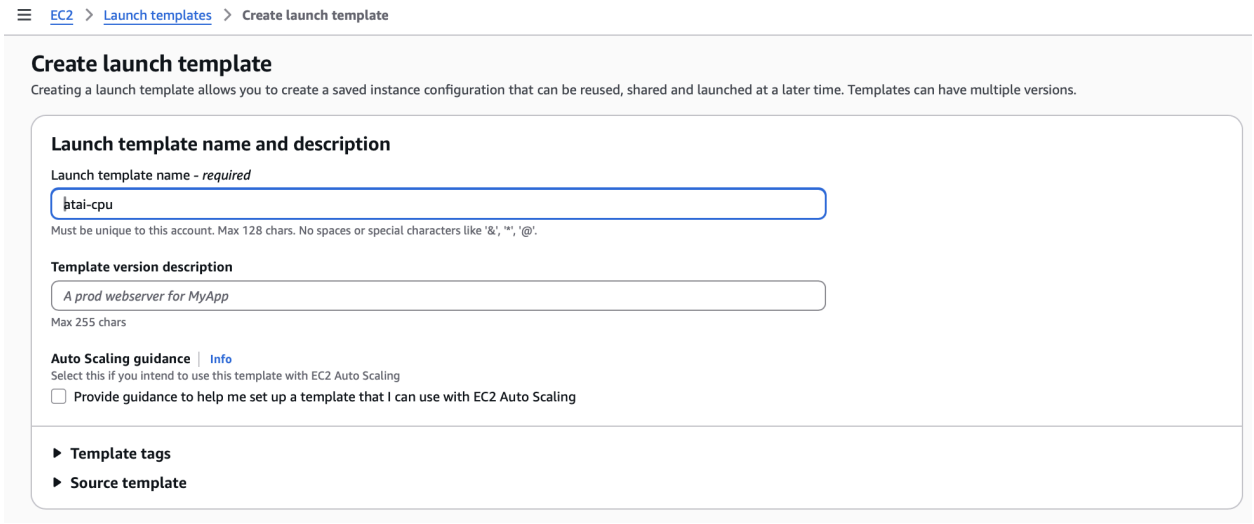
Step 3: Launch templates

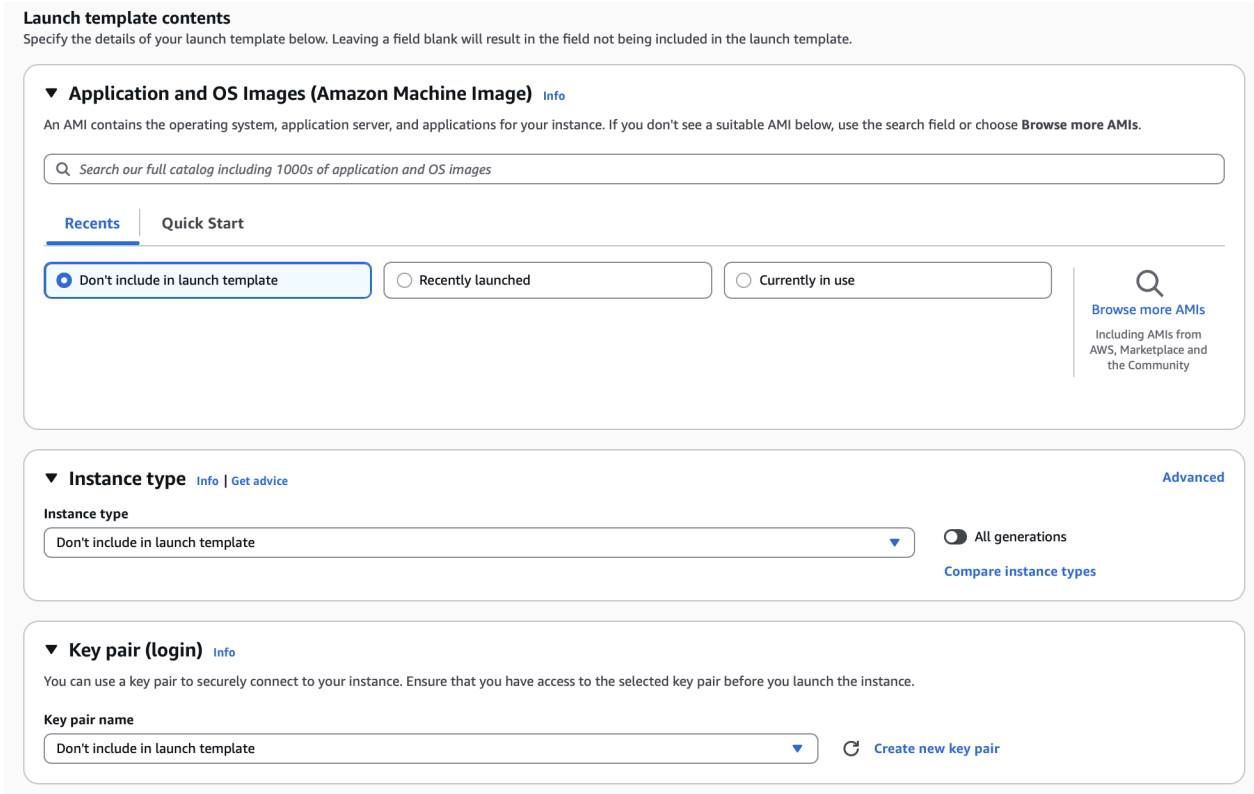
Step 3.1 CPU Node Group

1. Go to EC2 → Launch template → **Create Launch Template**

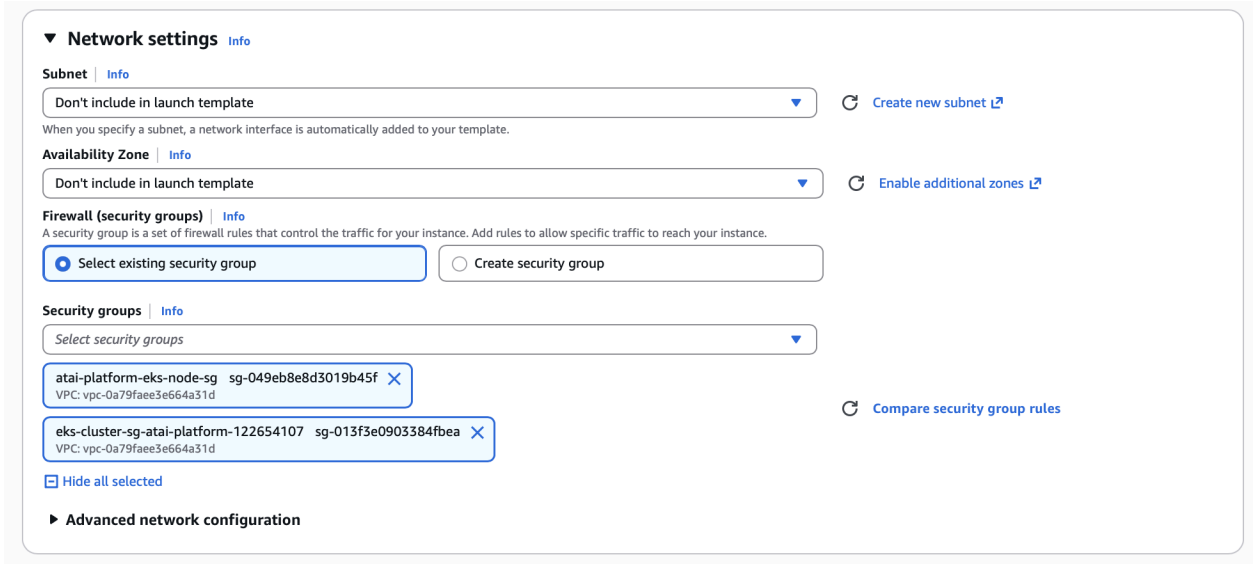


2. Name: atai-cpu

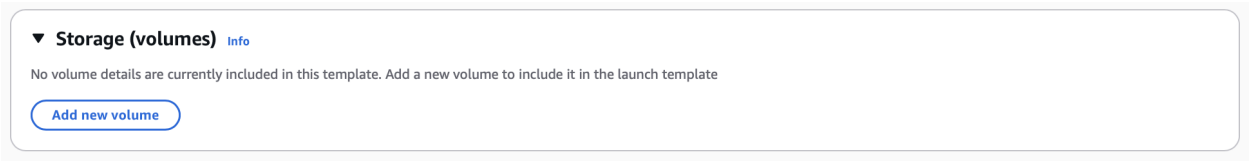




3. Network settings
    a. Select the security group created in the Step 2
    b. Select the default cluster security group that you get from the section **EKS cluster configuration - Step 4: Get the default cluster security group.**

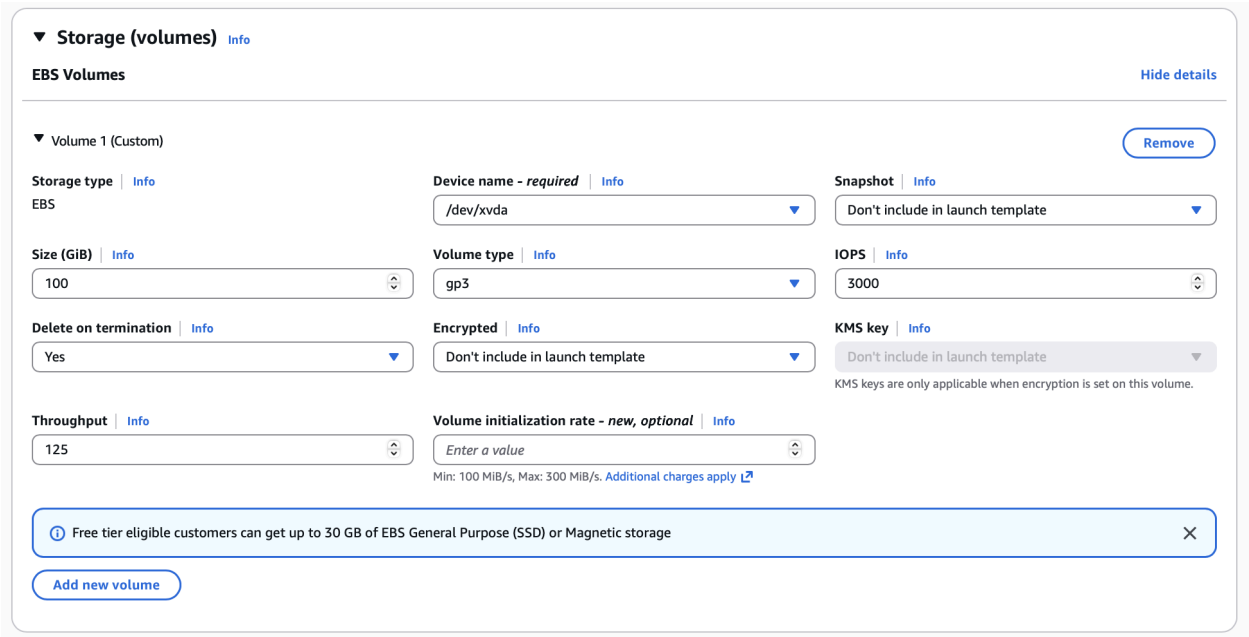


4. Storage (Volumes) - Click on **Add new volume**

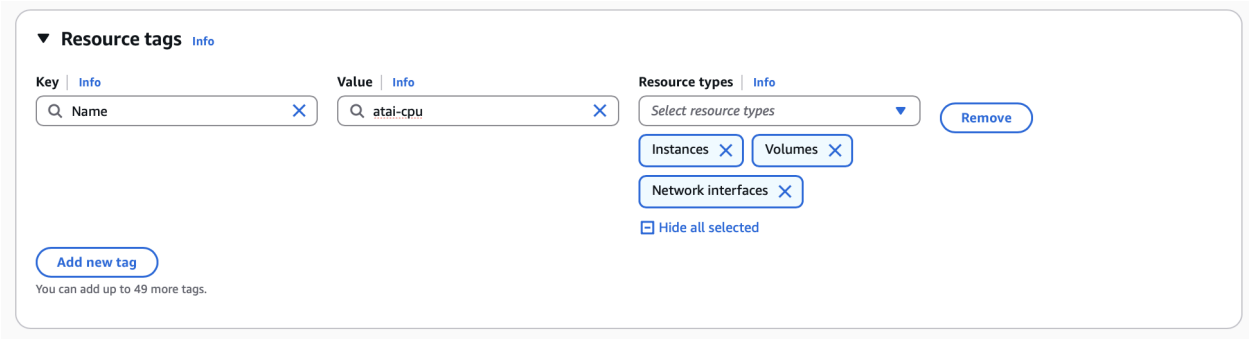


   a. Volume 1 configuration
      i. Size: 100
      ii. Volume type: gp3
      iii. Device name:
         1. Open the dropdown menu and click on **Specify a custom value**
         2. Value: /dev/xvda

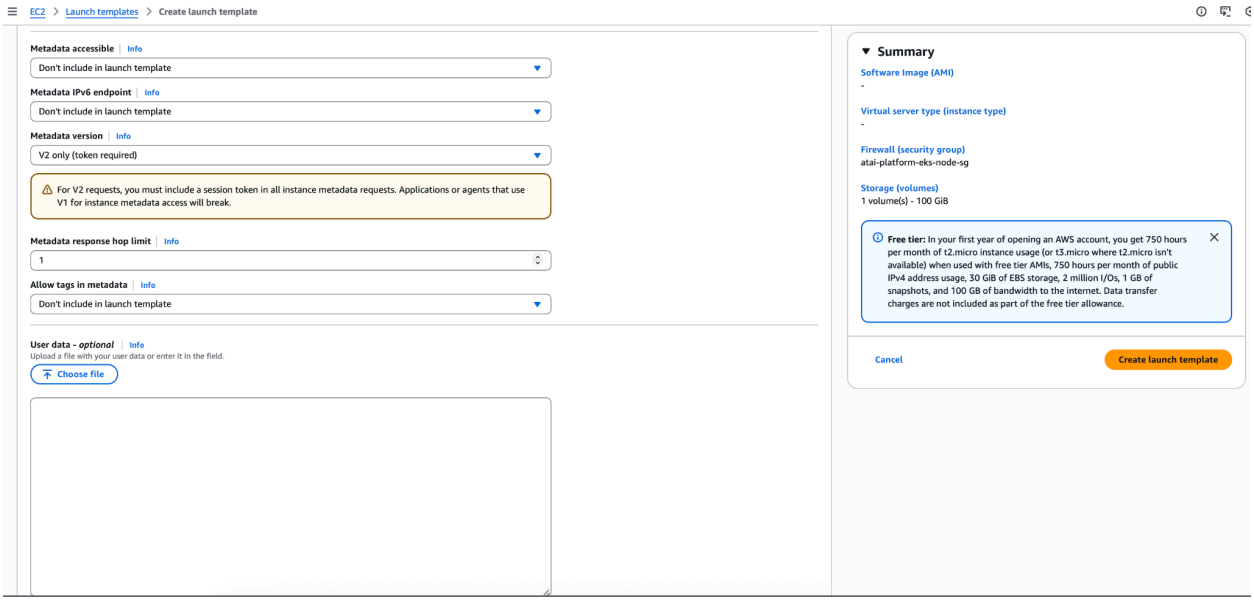
      iv. IOPS: 3000
      v. Throughput: 125



5. Resource tags
    a. Key: Name
    b. Value: atai-cpu
    c. Resource types
        i. Instances
        ii. Volumes
        iii. Network Interfaces



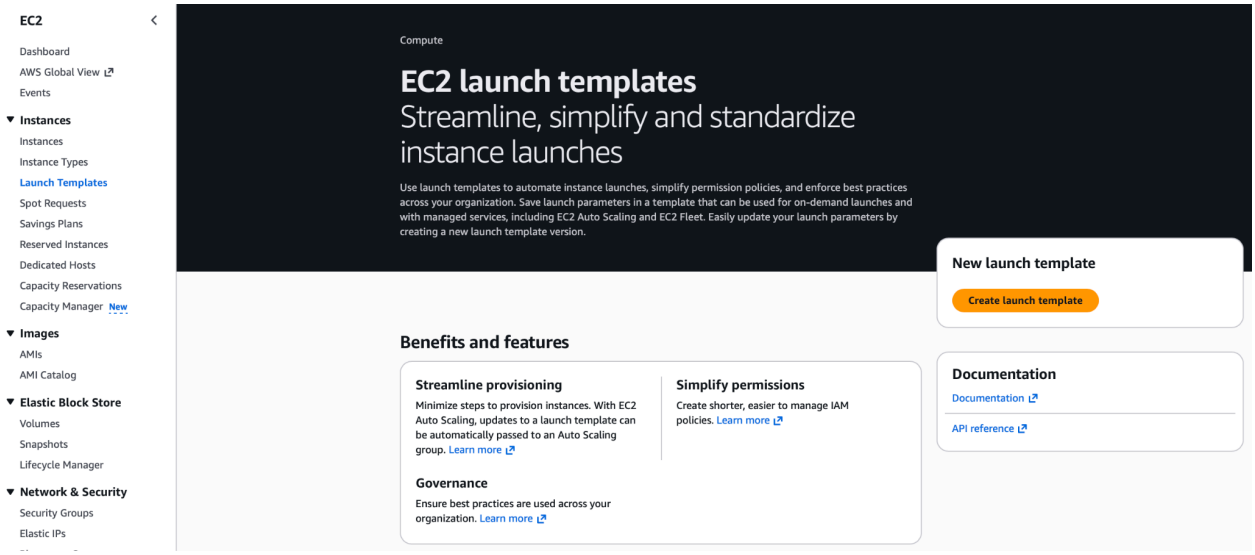
6. Advanced details
    a. Metadata version: V2
    b. Metadata response hop limit: 1



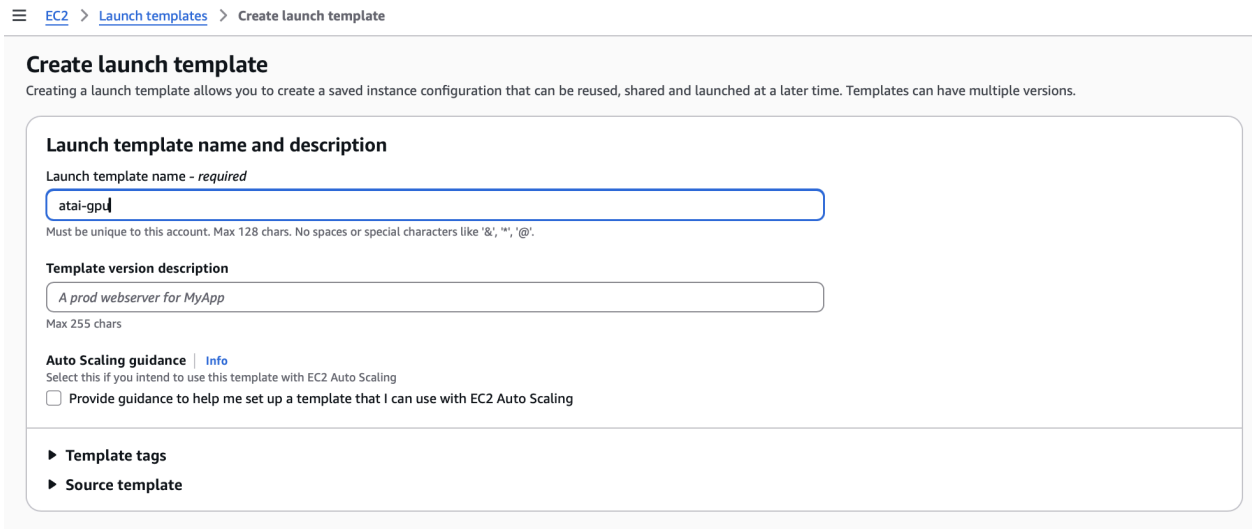
7. Click on **Create Launch Template**

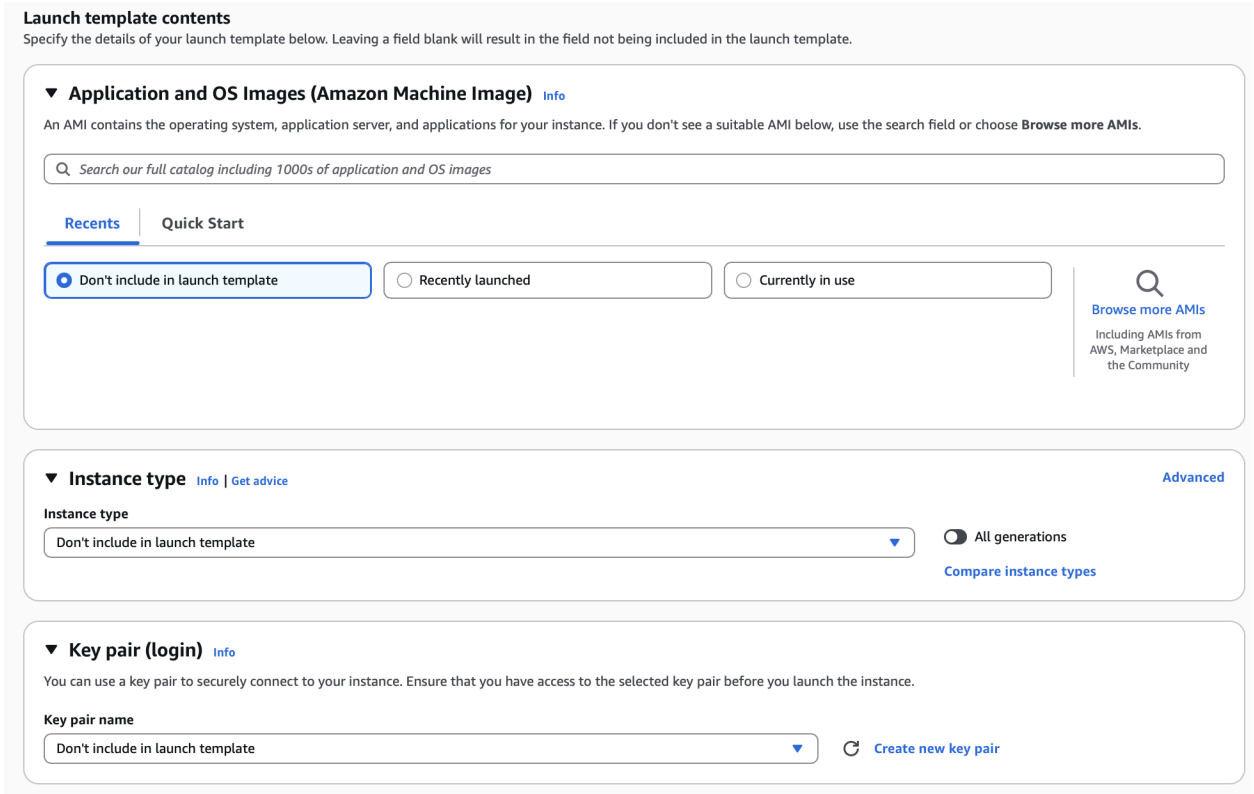
Step 3.2 GPU Node Group

8. Go to EC2 → Launch template → **Create Launch Template**

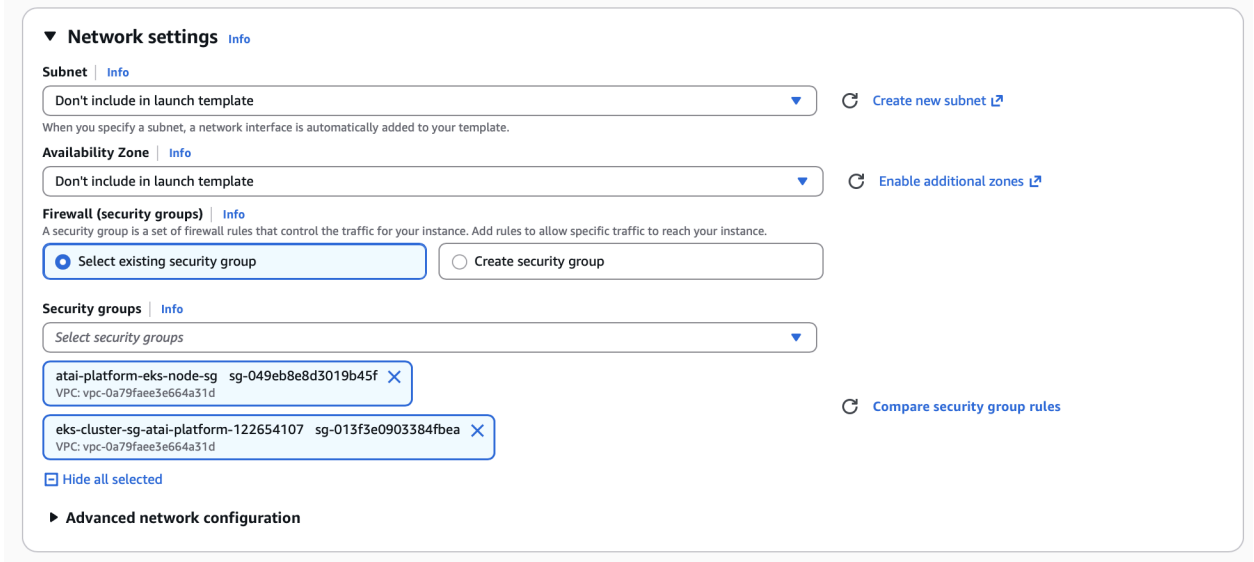


9. Name: atai-gpu

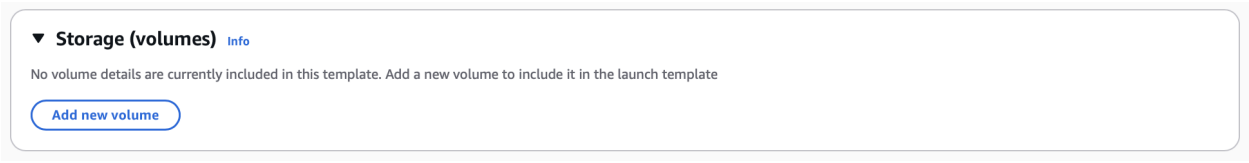




10. Network settings
    a. Select the security group created in the Step 2
    b. Select the default cluster security group that you get from the section EKS cluster configuration Step 4: Get the default cluster security group.

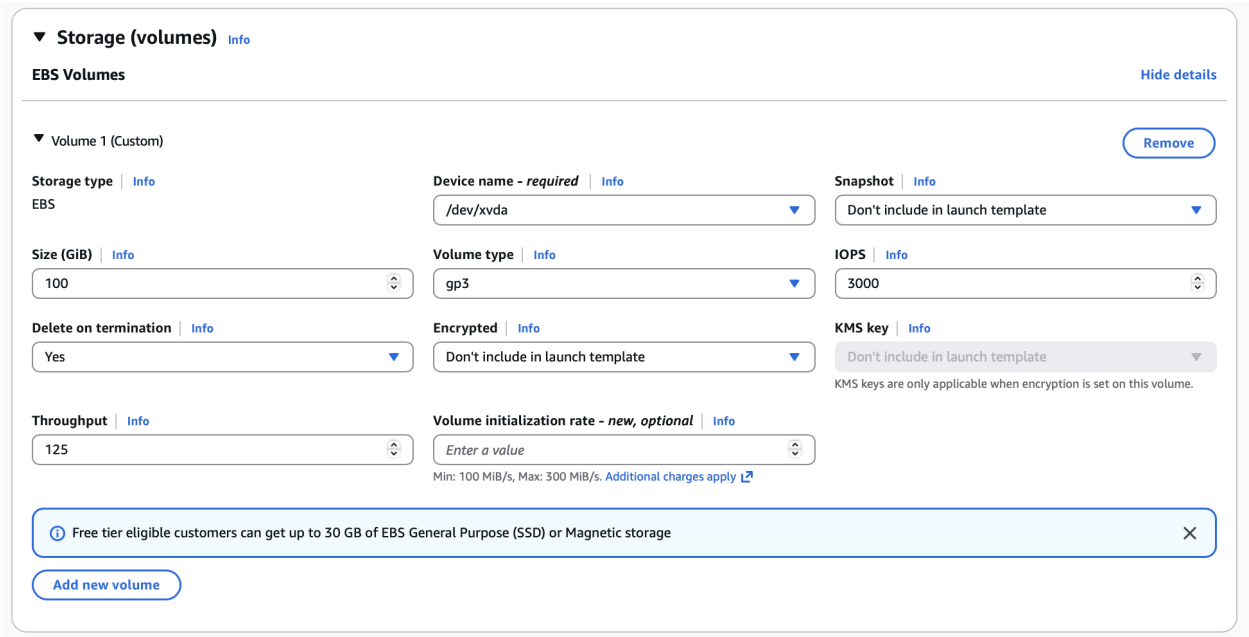


11. Storage (Volumes) - Click on **Add new volume**

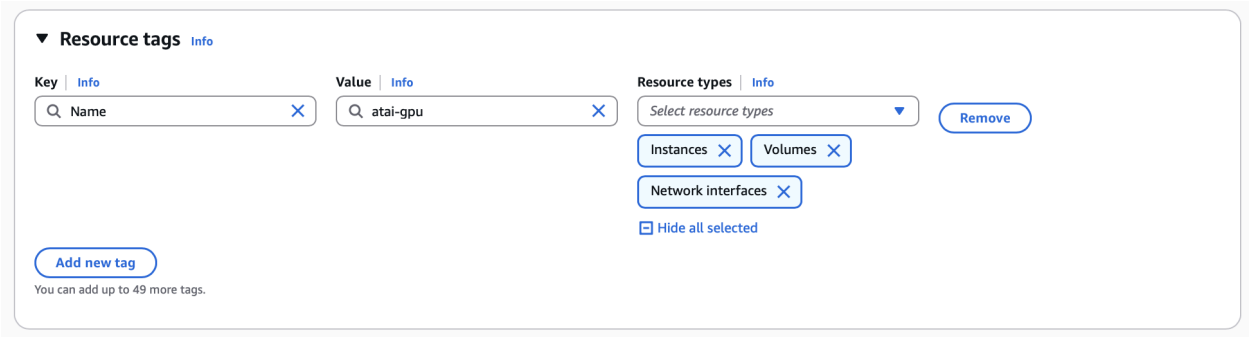


a. Volume 1 configuration
   i. Size: 100
   ii. Volume type: gp3
   iii. Device name:
      1. Open the dropdown menu and click on **Specify a custom value**
      2. Value: /dev/xvda

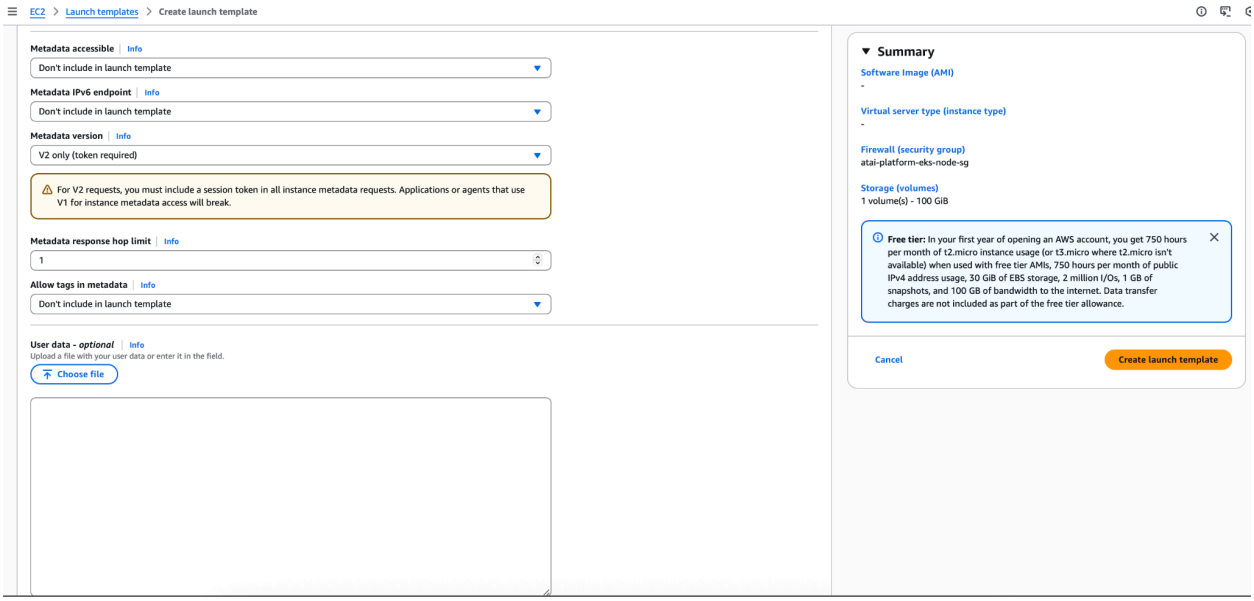
   iv. IOPS: 3000
   v. Throughput: 125



12. Resource tags
    a. Key: Name
    b. Value: atai-gpu
    c. Resource types
        i. Instances
        ii. Volumes
        iii. Network Interfaces



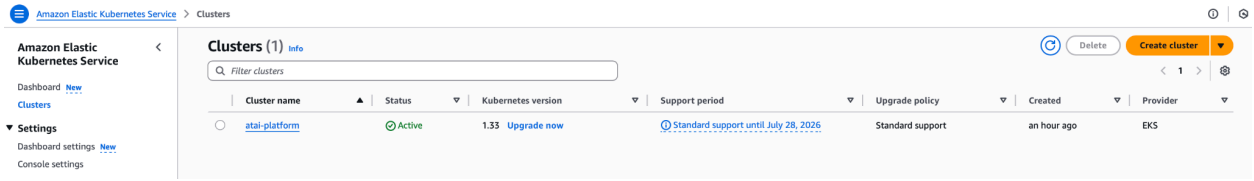
13. Advanced details
    a. Metadata version: V2
    b. Metadata response hop limit: 1



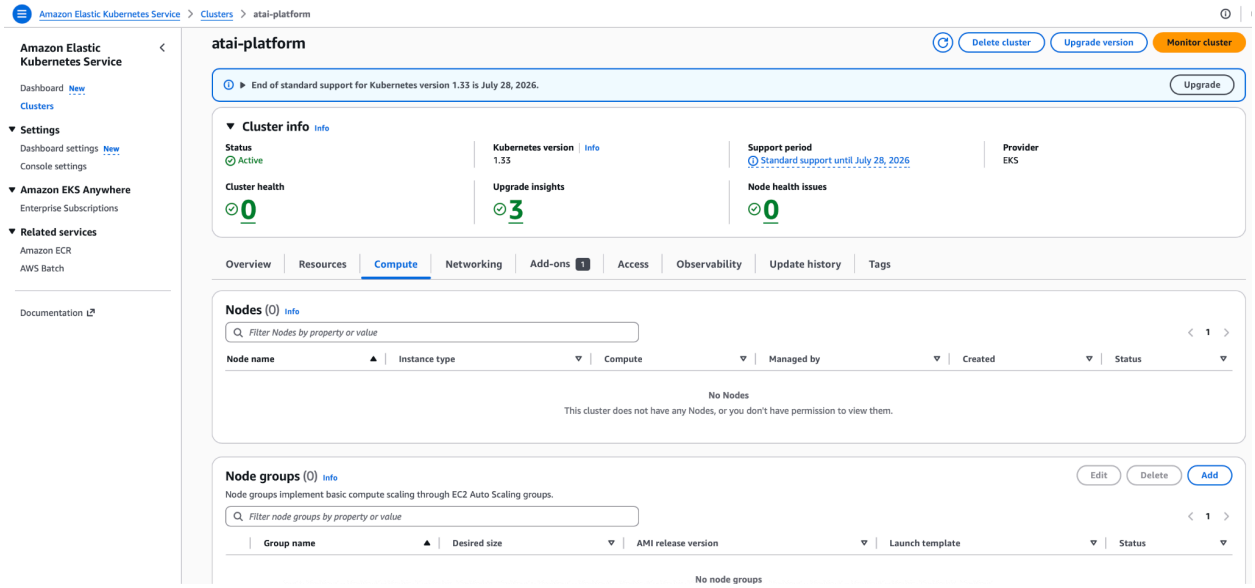
14. Click on **Create Launch Template**

Step 4: Create CPU Node Group

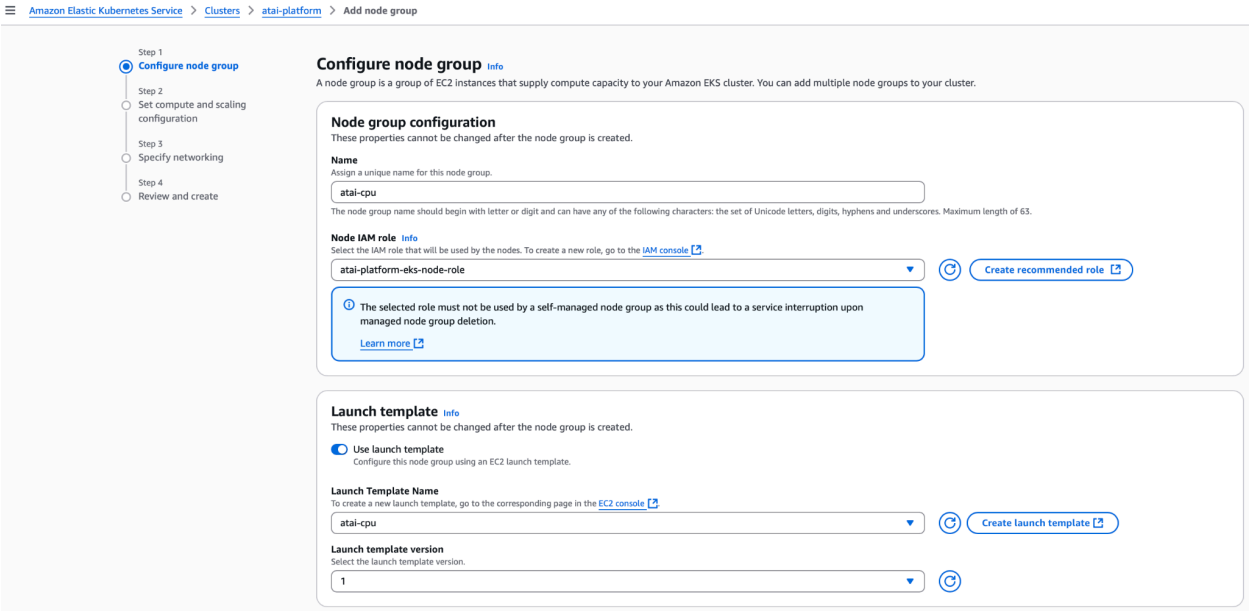
1. EKS Console → Your cluster



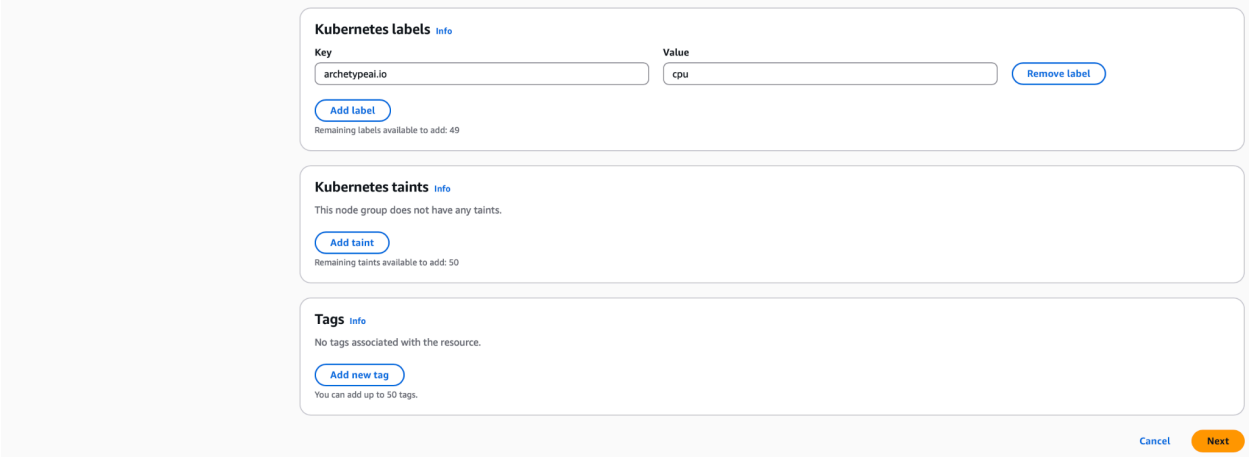
2. Select Compute tab → Node groups section → **Add node group**



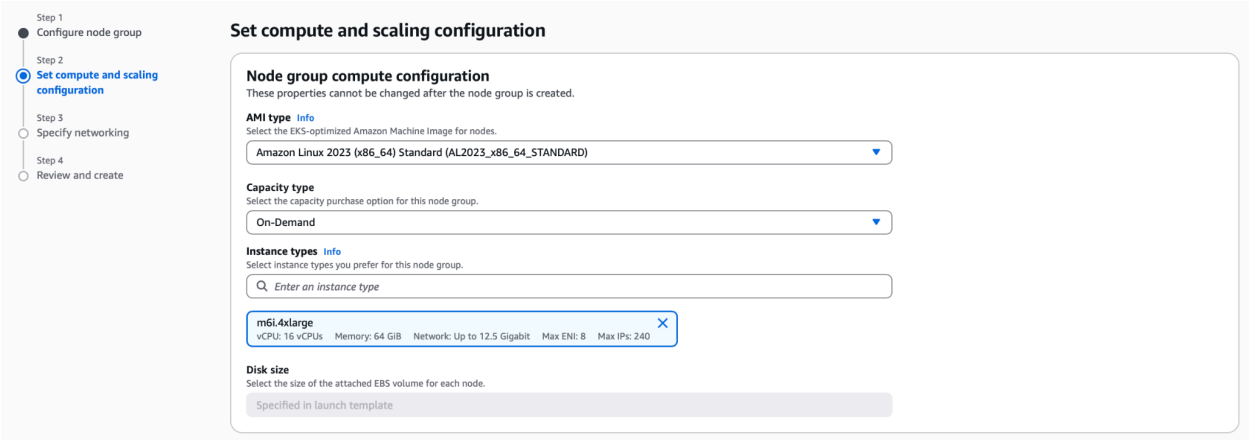
3. Configure node group:
    a. Node group name: atai-cpu
    b. Node IAM role: Select the role from Step 1
    c. Launch template: Select the Launch template created in the Step 3.1



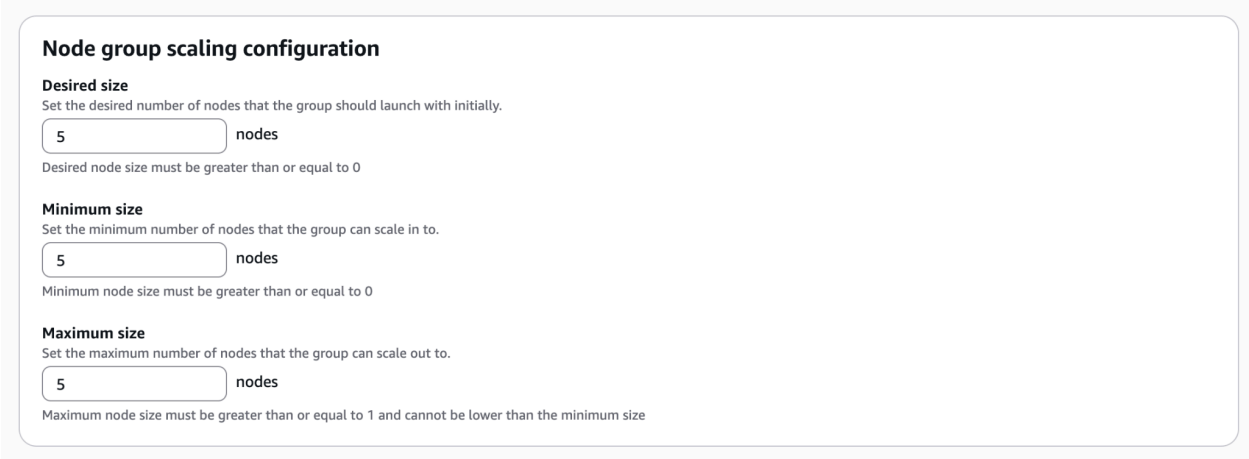
4. Configure Kubernetes labels and taints:
    a. Labels: Add label:
        i. Key: archetypeai.io
        ii. Value: cpu
    b. Taints: None (leave empty)



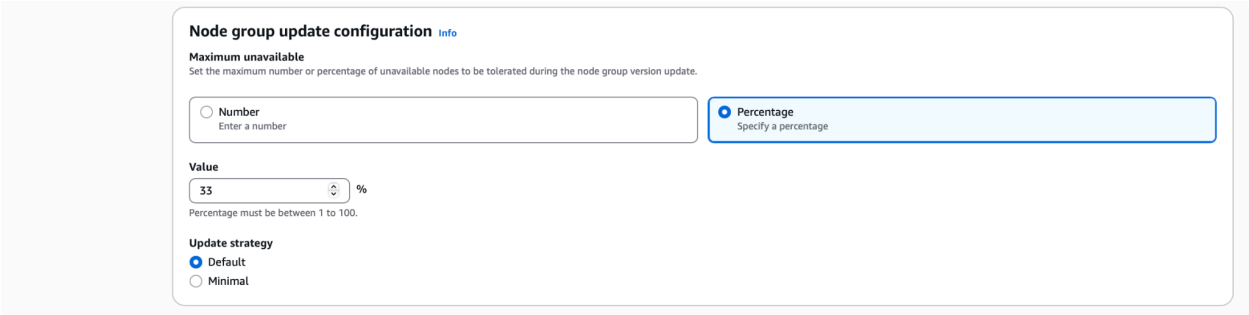
5. Node group compute configuration
    a. AMI type: Amazon Linux 2023 (AL2023_x86_64_STANDARD)
    b. Capacity type: On-Demand
    c. Instance types: m7i.4xlarge
    d. Disk size: Specified in the Launch template



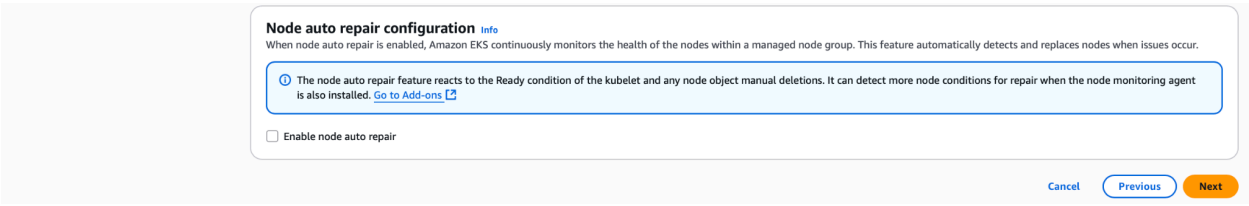
6. Node group scaling configuration:
    a. Desired size: 5
    b. Minimum size: 5
    c. Maximum size: 5



7. Node group update configuration:
    a. Maximum unavailable: Percentage
    b. Value: 33%
    c. Update strategy: Default

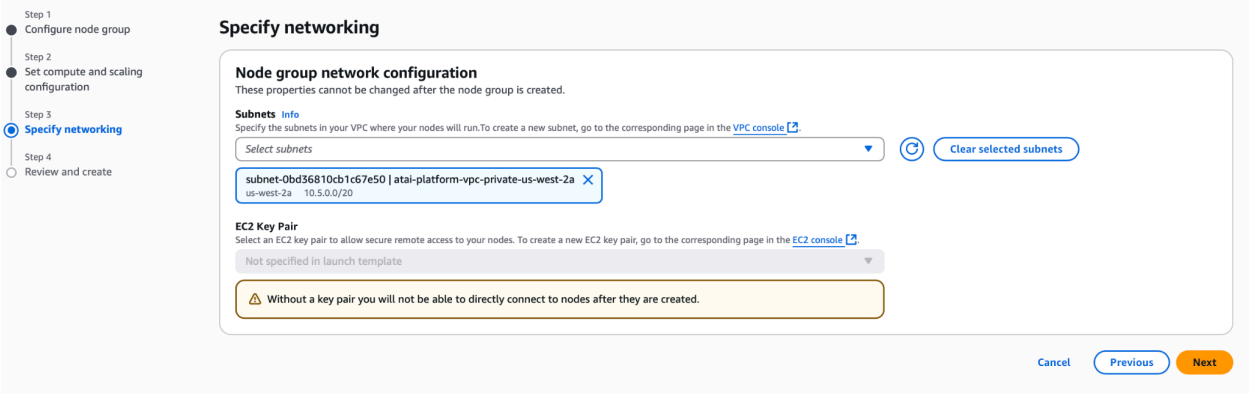


8. Node group auto repair configuration: Disable

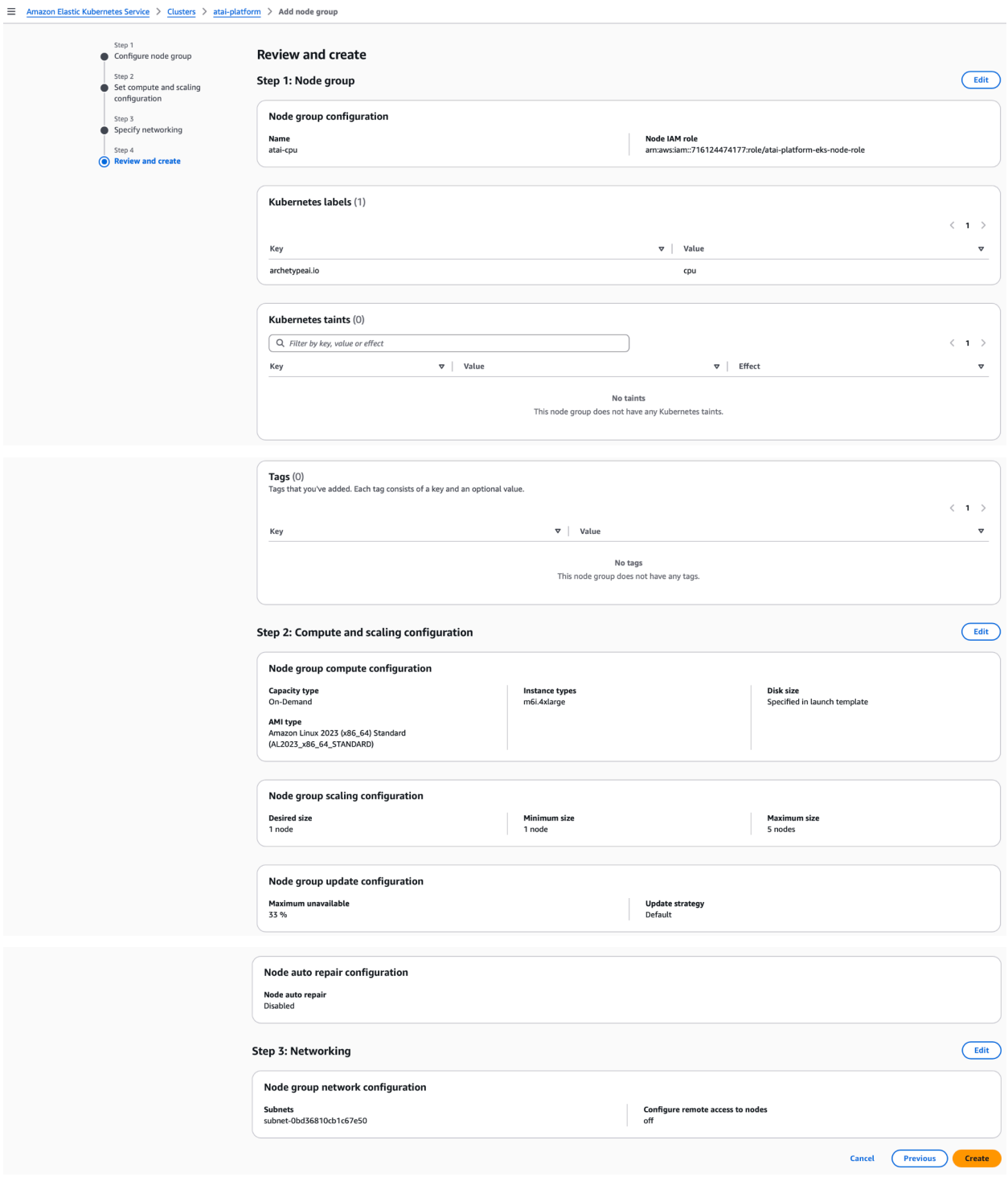


9. Node group network configuration
    a. Subnets: Select your private subnets:
        i. atai-platform-vpc-private-us-west-2a (10.5.0.0/20)

    It's important to select the private subnet in the same AZs as your Valkey clusters and RDS PostgreSQL cluster.

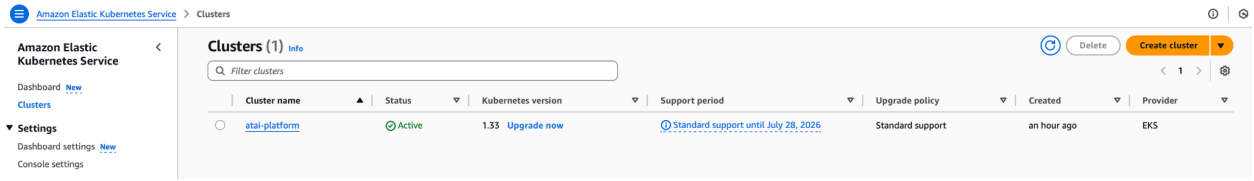


10. Review and click on **Create**
    Wait for node group to be ACTIVE (5-10 minutes)

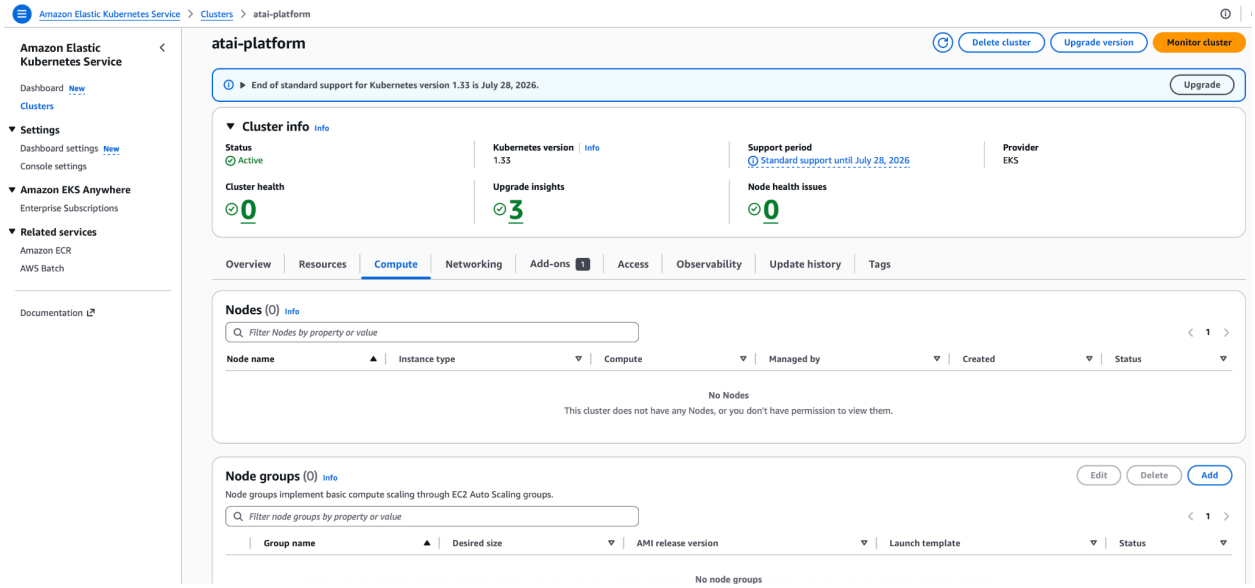


Step 5: Create GPU Node Group

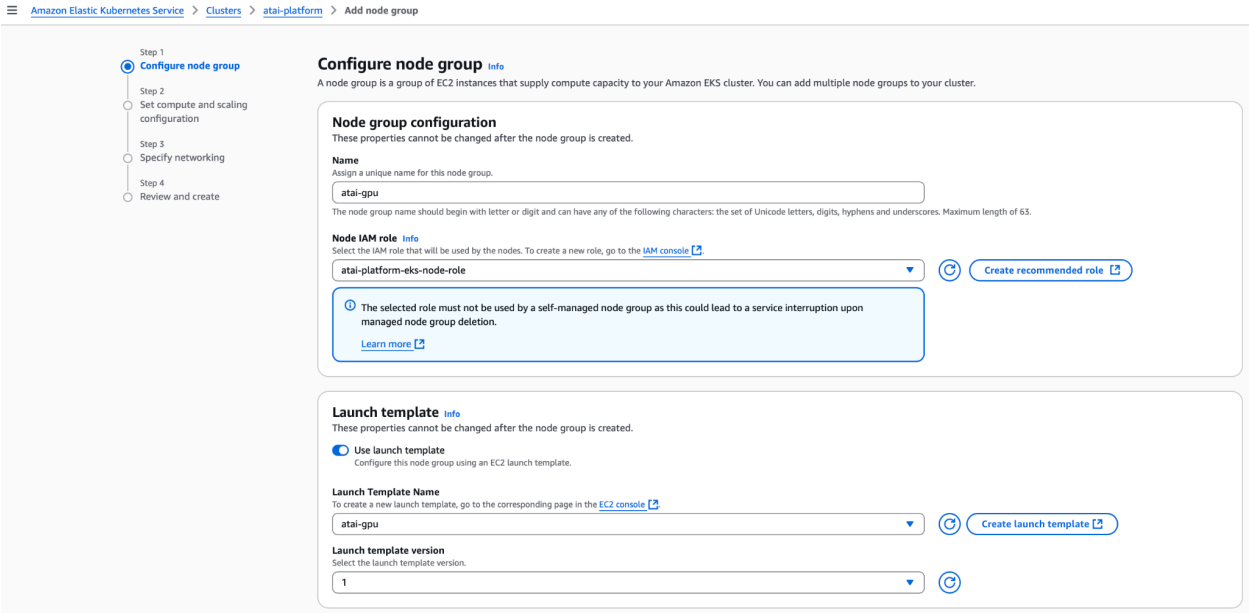
1. EKS Console → Your cluster



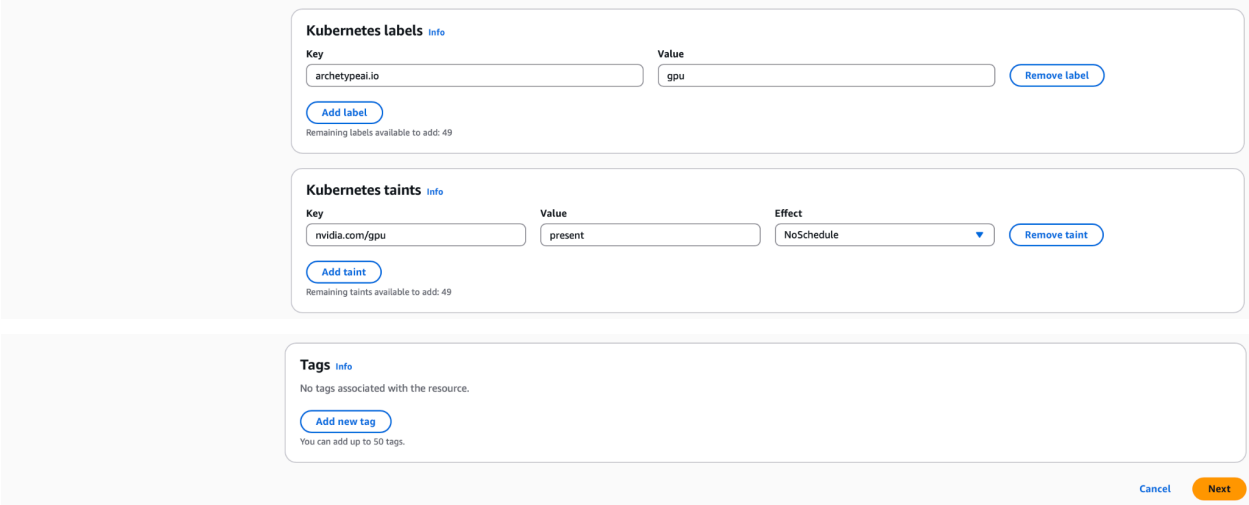
2. Select Compute tab → Node groups section → **Add node group**



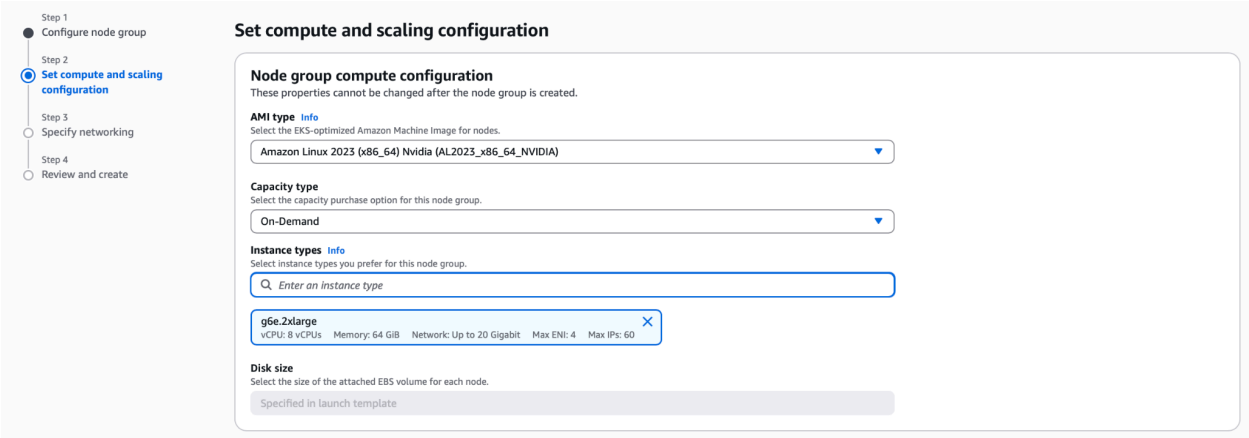
3. Configure node group:
    a. Node group name: atai-gpu
    b. Node IAM role: Select the role from Step 1
    c. Launch template: Select the Launch template created in the Step 3.2



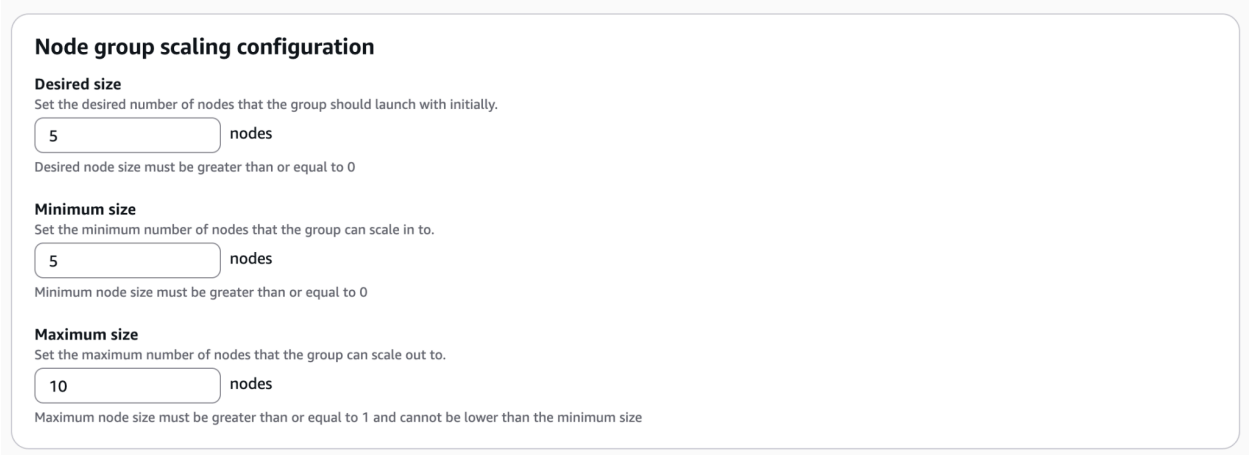
4. Configure Kubernetes labels and taints:
    a. Labels: Add label:
        i. Key: archetypeai.io
        ii. Value: gpu
    b. Taints:
        i. Key: nvidia.com/gpu
        ii. Value: present
        iii. Effect: NoSchedule



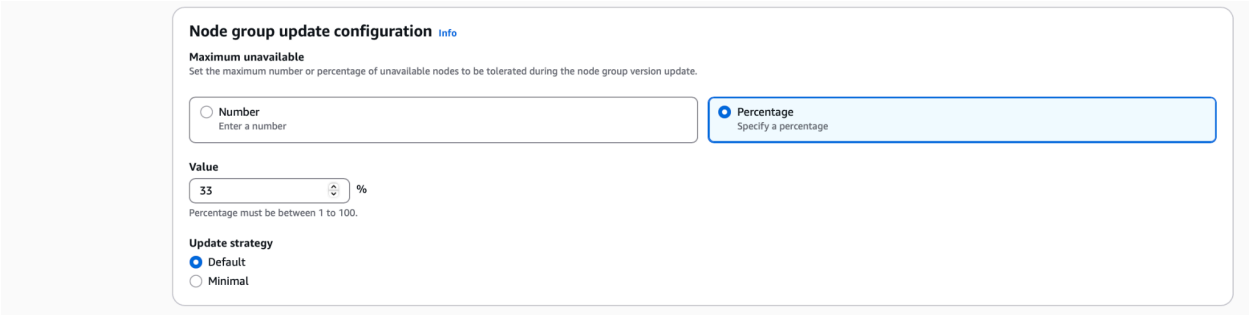
5. Node group compute configuration
    a. AMI type: Amazon Linux 2023 (x86_64) Nvidia (AL2023_x86_64_NVIDIA)
    b. Capacity type: On-Demand
    c. Instance types: g6e.2xlarge
    d. Disk size: Specified in the Launch template



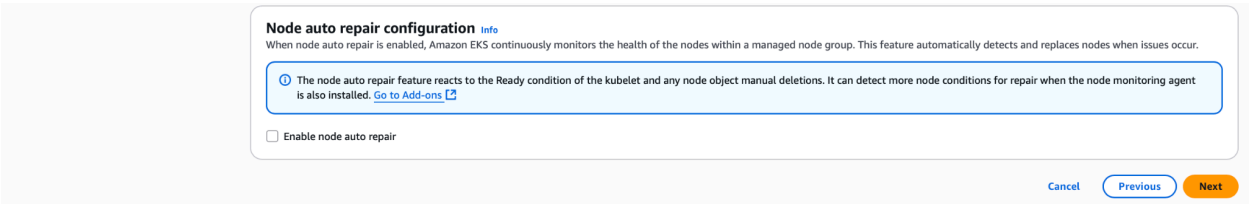
6. Node group scaling configuration:
    a. Desired size: 5
    b. Minimum size: 5
    c. Maximum size: 10



7. Node group update configuration:
    a. Maximum unavailable: Percentage
    b. Value: 33%
    c. Update strategy: Default

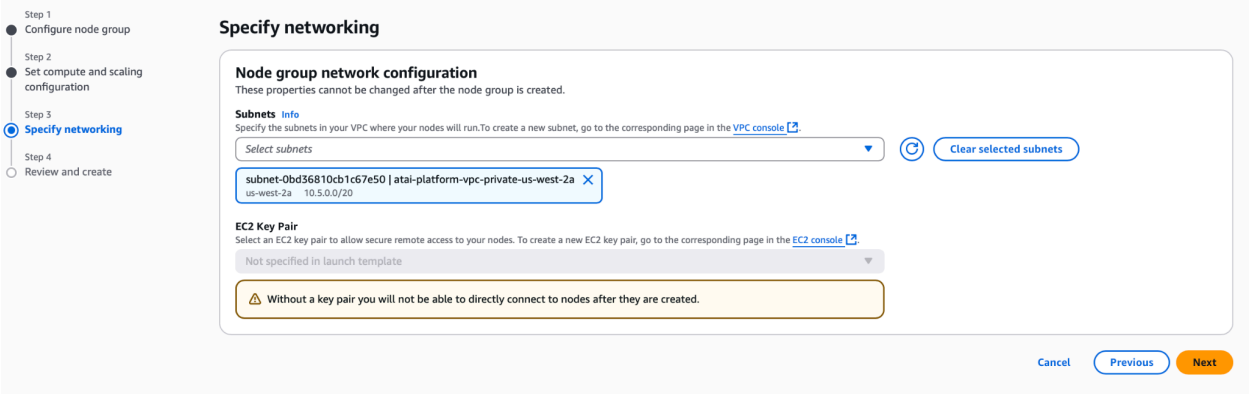


8. Node group auto repair configuration: Disable

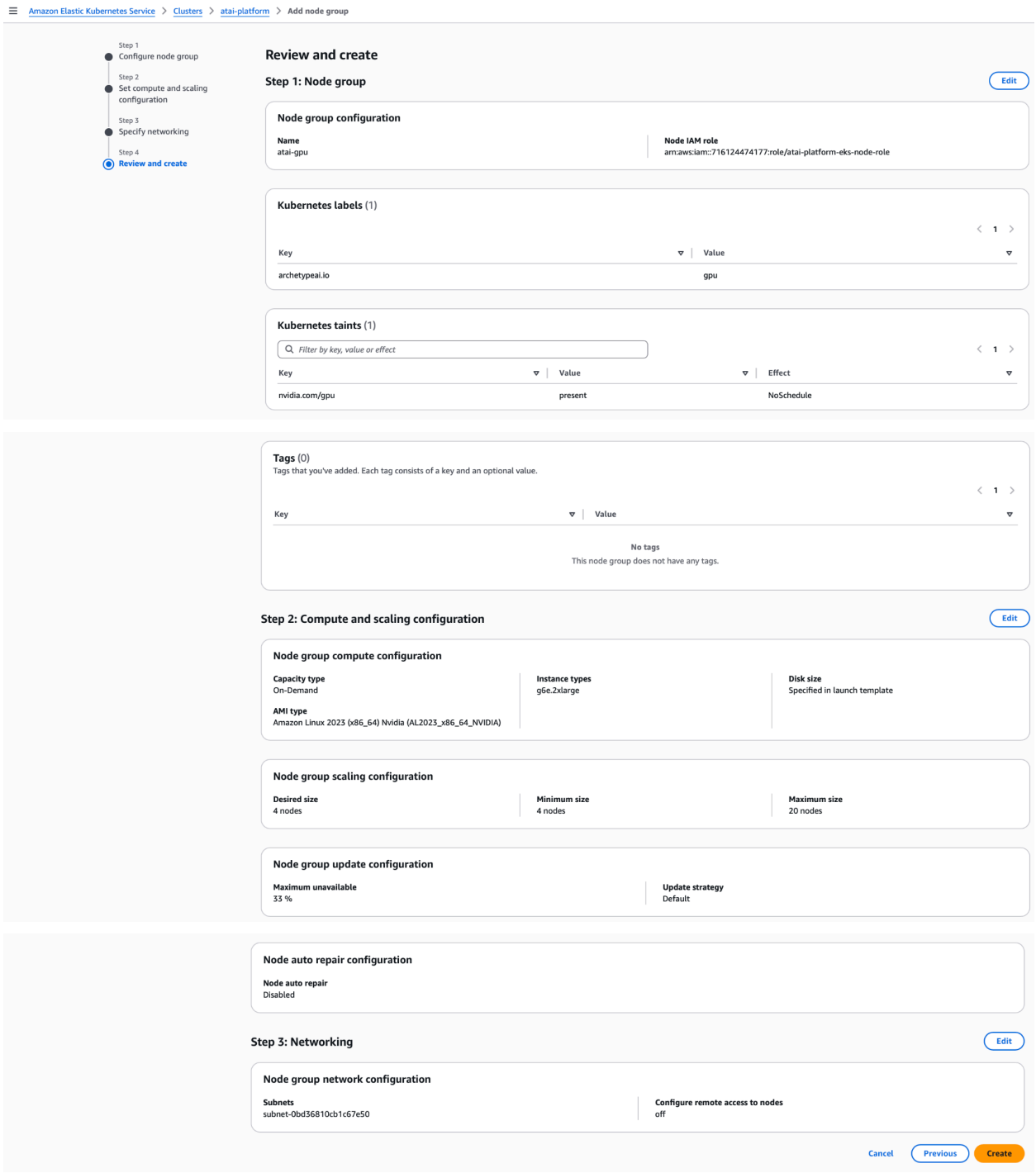


9. Node group network configuration
    a. Subnets: Select your private subnets:
        i. atai-platform-vpc-private-us-west-2a (10.5.0.0/20)

    It's important to select the private subnet in the same AZs as your Valkey clusters and RDS PostgreSQL cluster.



10. Review and click on **Create**
    Wait for node group to be ACTIVE (5-10 minutes)



EKS configuration - Install the NVIDIA Device Plugin

The NVIDIA device plugin DaemonSet to enable GPU resource scheduling in Kubernetes.

Prerequisites

1. EKS cluster is ACTIVE
2. The kubectl command line tool is required. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster.
3. An IAM principal with permissions to create and describe an Amazon EKS cluster

Step 1: Manual installation

1. Direct download from GitHub:

```
None
curl -L -o nvidia-device-plugin-v0.18.0.yml
https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.18.0/deployments/sta
tic/nvidia-device-plugin.yml
```

2. Then apply the Kubernetes manifest:

```
None
kubectl apply -f nvidia-device-plugin-v0.18.0.yml
```

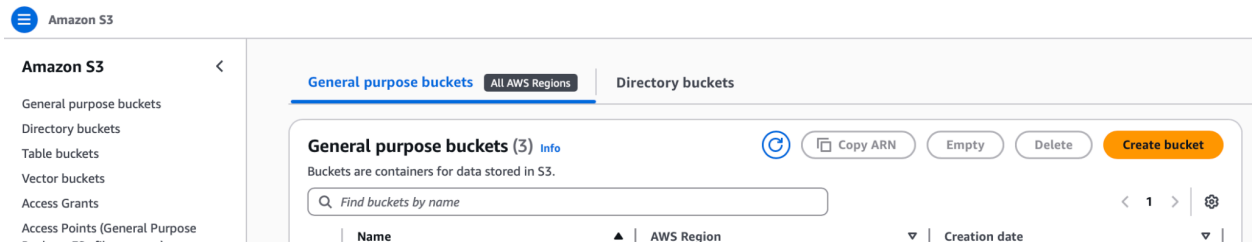
3. Verify nodes are annotated with the nvidia label:

```
None
kubectl get nodes -o
custom-columns="NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

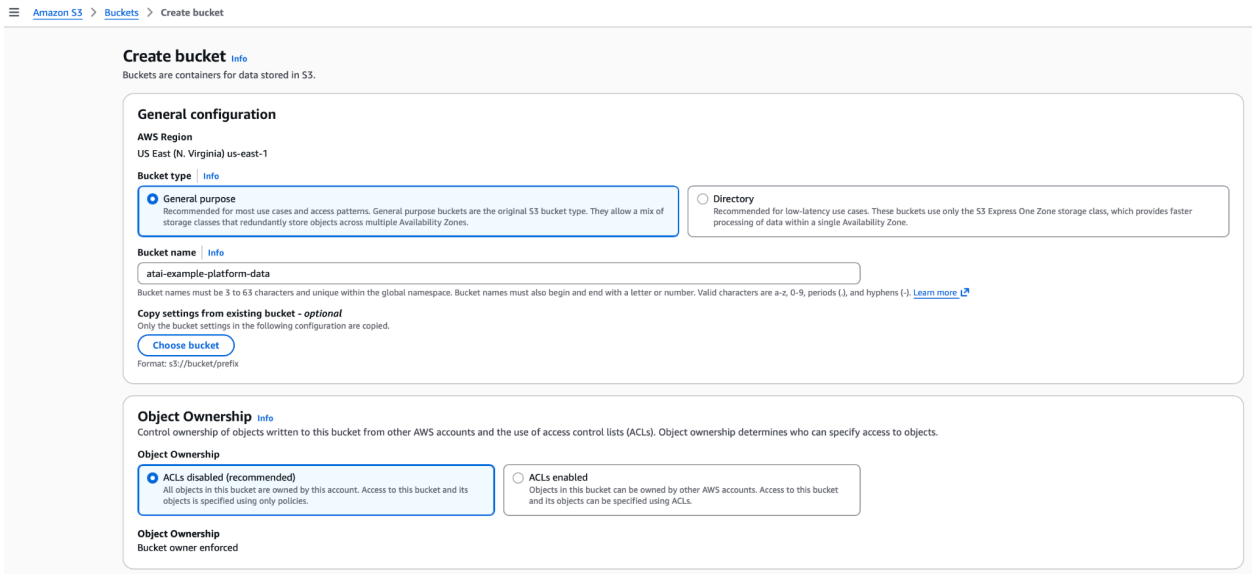
S3 configuration

Step 1: Create the platform-data bucket

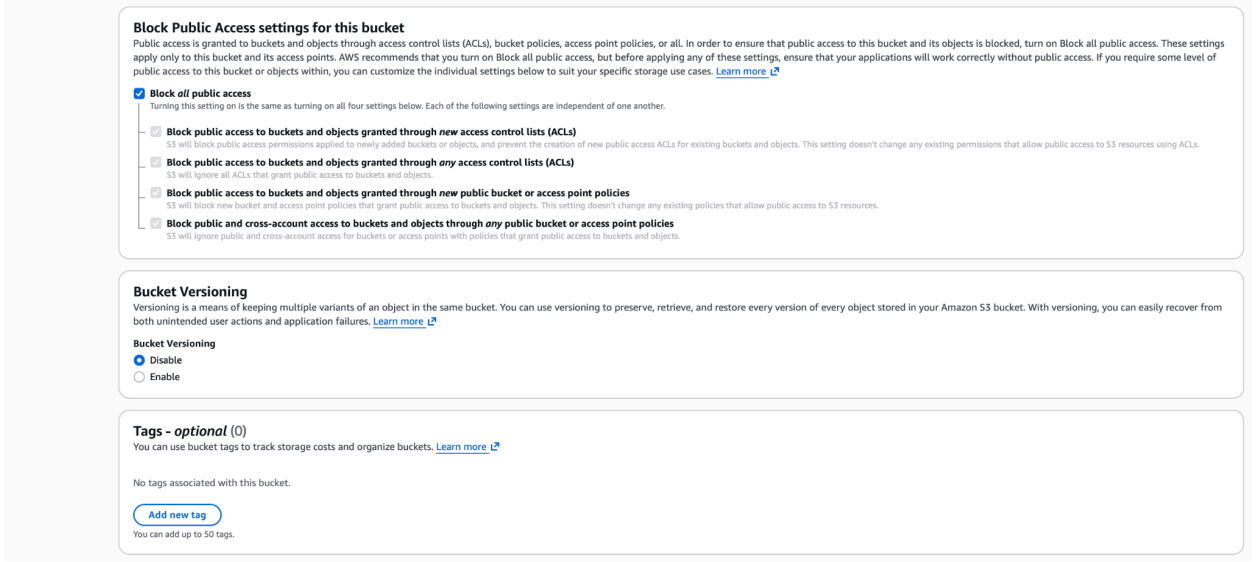
1. Go to the S3 dashboard and click on **Create Bucket**
2. Make sure, you are located in your home AWS region



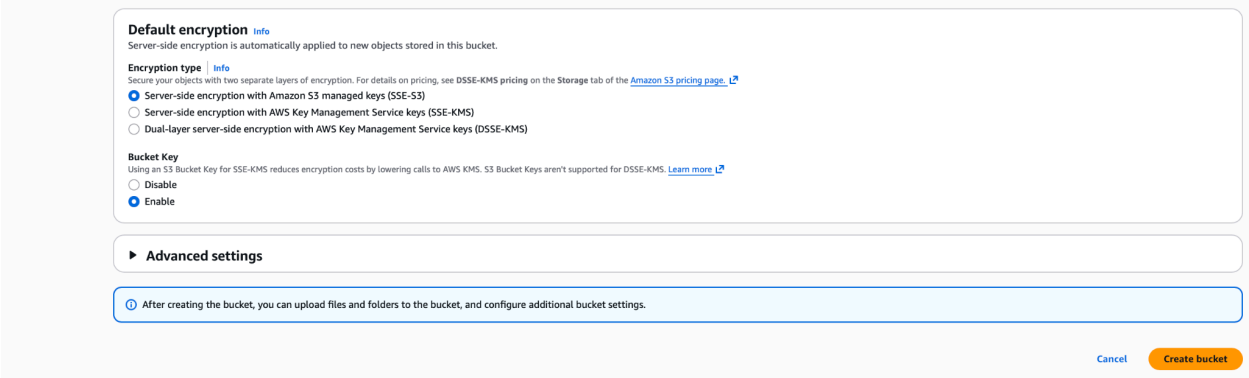
3. Bucket name: atai-<UNIQUE ID>-platform-data



4. Keep the selection in the **Block Public Access setting for the bucket**



5. Use the default Encryption configuration

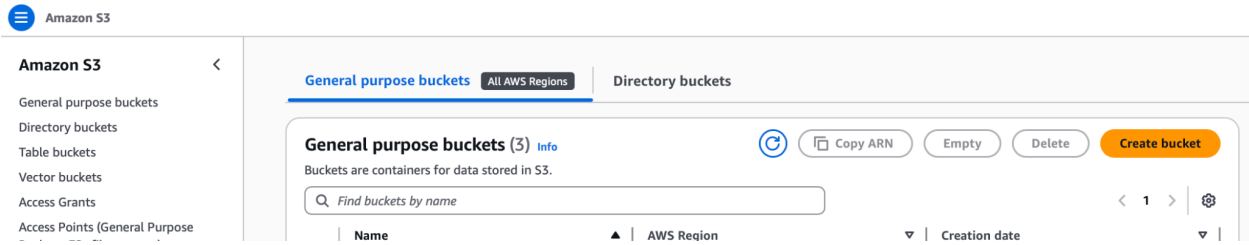


6. Click on **Create bucket**

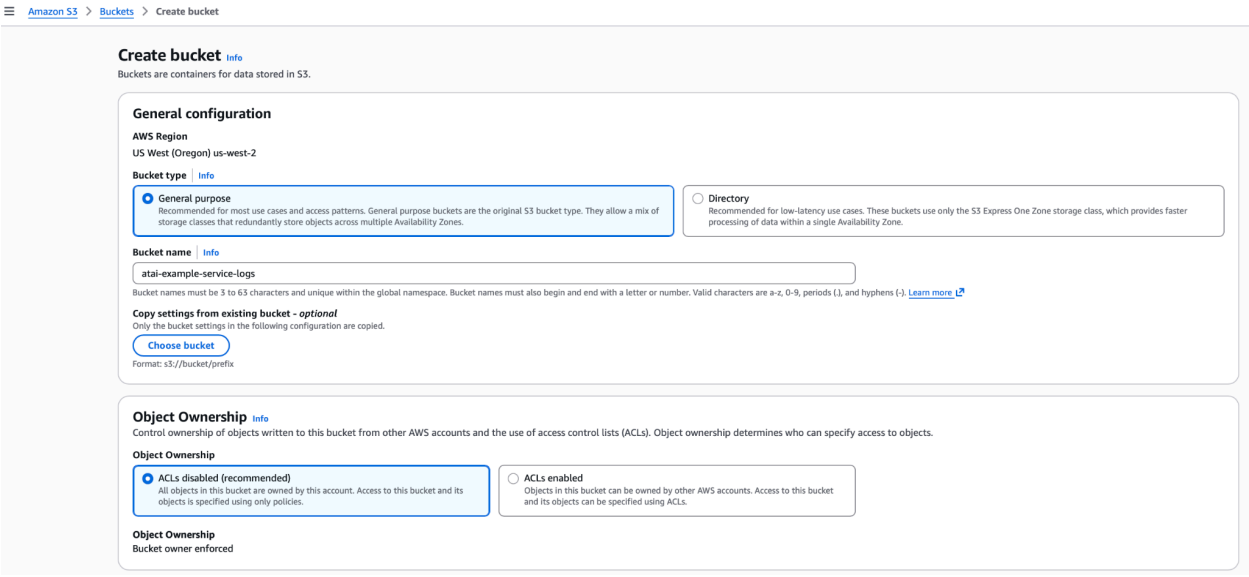
⚠️ Store your bucket endpoint in a secure location. You will need them later in the *atai-platform* prerequisites.

Step 2: Create the service logs bucket

1. Go to the S3 dashboard and click on **Create Bucket**
2. Make sure, you are located in your home AWS region

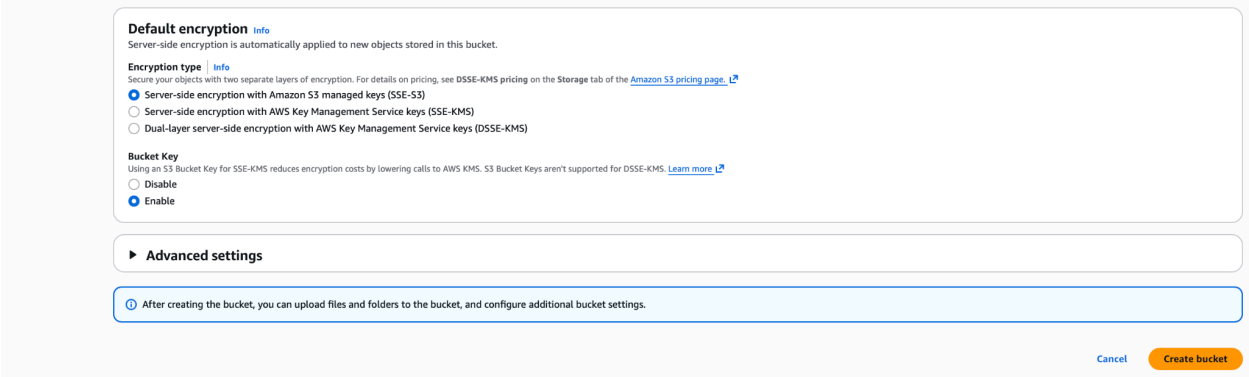


3. Bucket name: atai-<SOME-UNIQUE ID>-service-logs



4. Keep the selection in the **Block Public Access setting for the bucket**

5. Use the default Encryption configuration



6. Click on **Create bucket**

⚠️ Store your bucket endpoint in a secure location. You will need them later in the *atai-platform* prerequisites.

Application Endpoints Configuration

Platform Architecture Overview

The Archetype platform consists of two main entry points that must be publicly accessible:

1. **Console**: The web-based user interface where users interact with the platform
2. **API**: The backend service that handles business logic and data processing

For proper operation, the API must be publicly accessible so the Console can communicate with it, and vice versa. This is a standard configuration for web applications where the frontend (Console) and backend (API) need to exchange data.

Required Public URLs

Once configured, the following URLs must be publicly accessible and secured:

1. **console-archetype.<your-domain>** - Console web application
2. **api-archetype.<your-domain>** - API backend service

Both URLs must be accessible over HTTPS with valid SSL certificates.

How Do I Expose My Services Publicly?

To make your Console and API services publicly accessible, you need to use a Kubernetes Ingress.

What is an Ingress?

An Ingress is a Kubernetes resource that defines routing rules for external traffic. It specifies which domain names should route to which services and on which paths. However, an Ingress resource by itself doesn't handle traffic, you need an Ingress Controller to process these rules and route the actual traffic.

An Ingress Controller acts as a reverse proxy that:

- Receives external HTTP/HTTPS traffic
- Reads Ingress resource rules
- Routes traffic to the appropriate backend services based on domain names and paths

Common Ingress Controllers:Popular Ingress Controller options include:

- AWS Load Balancer Controller - Creates AWS Application Load Balancers (ALB) or Network Load Balancers (NLB)
- NGINX Ingress Controller - A widely-used, feature-rich Ingress Controller
- Traefik - Another popular option

If you don't have an Ingress Controller configured: See Appendix EKS configuration - Install the NGINX ingress controller.

How Do I Secure the Traffic?

Once your services are exposed via Ingress, it's important to secure the traffic with SSL/TLS certificates. This ensures all communications are encrypted and secure.

Common Certificate Solutions: Ingress Controllers typically integrate with certificate management solutions:

- Let's Encrypt through Cert-Manager - Automated, free SSL certificates that are automatically renewed
- AWS Certificate Manager (ACM) - Native AWS-managed certificates

Cert-Manager is a Kubernetes add-on that automatically provisions, renews, and manages SSL certificates. It can integrate with Let's Encrypt to obtain free certificates automatically, or work with AWS ACM for AWS-managed certificates.

If you need help configuring certificates: See Appendix: EKS configuration - Configure Cert-Manager and Let's Encrypt

How Do I Configure DNS?

After your Ingress Controller is configured, it typically creates an AWS Network Load Balancer (NLB) or Application Load Balancer (ALB) as the entry point for your services.

DNS Configuration Steps:

1. Your Ingress Controller creates an AWS load balancer (NLB or ALB)
2. The load balancer receives a public DNS name or IP address
3. You need to add DNS records that point your domain names to this load balancer

Required DNS Records:

1. Add the following DNS records (A records or CNAME records) pointing to the load balancer created by your Ingress solution:
    a. **console-archetype.<your-domain>** → Points to the load balancer
    b. **api-archetype.<your-domain>** → Points to the load balancer

The exact DNS configuration depends on your DNS provider and whether your load balancer provides a DNS name (use CNAME) or an IP address (use A record).

Deployment Checklist

Before Helm Chart Installation:

- ☐ **VPC and subnets** deployed with proper CIDR blocks
- ☐ **8 Valkey instances** created with correct names and versions
- ☐ **1 RDS PostgreSQL** instance created with Aurora engine
- ☐ **EKS cluster** deployed with Kubernetes 1.33
- ☐ **1 CPU node group** deployed with m7i.4xlargeinstances
- ☐ **4 GPU node group** deployed with g6e.2xlarge instances and taints
- ☐ **S3 bucket** atai-{customer-prefix}-platform-data and atai-{customer-prefix}-service-logs created
- ☐ **Network connectivity** verified between pods and databases
- ☐ **k8s Ingress** solution installed in the EKS cluster

Support

For questions about infrastructure requirements, contact the Archetype team (support@archetypeai.dev) for validation of instance types and scaling configurations.

atai-platform

Prerequisites

1. The kubectl command line tool is required. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster.
2. The eksctl command line tool is required. For more information visit Installation options for Eksctl.
3. Version 2.12.3 or later or version 1.27.160 or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device.
4. An IAM principal with permissions to create and describe an Amazon EKS cluster

Download the deploy kit files

Download and extract the deploy kit, which contains the configuration files and scripts required for the atai-platform configuration:

```shell
curl -O
https://archetypeai-marketplace-assets.s3.us-west-2.amazonaws.com/atai-platform-deploy-kit.tar.gz && \
tar -xzf atai-platform-deploy-kit.tar.gz
```

This will create an ***atai-platform-deploy-kit/*** directory with the following structure:

```shell
atai-platform-deploy-kit/
├── scripts/
│   ├── 6.create-irsa.sh
│   └── policies/
│       ├── platform-data-access.json.tpl
│       ├── service-logs-access.json.tpl
│       └── model-depot-access.json
└── templates/
    └── values.yaml
```

Step 1: Kubernetes namespaces

1. Create the namespace to install all the components of the atai-platform

```shell
$ kubectl create namespace atai-platform
```

Step 2: Kubernetes Service account for IAM roles (IRSA)

1. Associated an IAM OIDC provider

```shell
$ eksctl utils associate-iam-oidc-provider \
    --region <AWS_REGION> \
    --cluster atai-platform \
    --approve
```

2. Run the setup script to create IAM policies, the IAM role, and the Kubernetes service account.

Before running, gather the following values:

Parameter	Description
CUSTOMER_NAME	Short identifier for your organization (max 15 characters). Used as a prefix for IAM resource names.
EKS_CLUSTER_NAME	Name of the EKS cluster created in the previous section.
PLATFORM_DATA_BUCKET_NAME	Name of the S3 bucket created in "S3 configuration". Bucket name only (do not include s3:// or the full ARN).
SERVICE_LOGS_BUCKET_NAME	Name of the S3 bucket created in "S3 configuration". Bucket name only (do not include s3:// or the full ARN).

Set the variables and run the script:

```shell
CUSTOMER_NAME="<CUSTOMER_NAME>" \
EKS_CLUSTER_NAME="<EKS_CLUSTER_NAME>" \
PLATFORM_DATA_BUCKET_NAME="<PLATFORM_DATA_BUCKET_NAME>" \
SERVICE_LOGS_BUCKET_NAME="<SERVICE_LOGS_BUCKET_NAME>" \
&& cd atai-platform-deploy-kit/scripts \
&& ./6.create-irsa.sh \
    --region us-west-2 \
    --customer-name "$CUSTOMER_NAME" \
    --cluster-name "$EKS_CLUSTER_NAME" \
    --platform-data-bucket "$PLATFORM_DATA_BUCKET_NAME" \
```

```
    --service-logs-bucket "$SERVICE_LOGS_BUCKET_NAME"
```

Upon successful completion, you should see output similar to:

```
Shell
[INFO] Setup completed successfully!
[INFO]
[INFO] Summary:
[INFO]   - Created namespace: atai-platform
[INFO]   - Created 3 IAM policies
[INFO]   - Using S3 buckets:
[INFO]     * Platform Data: atai-<CUSTOMER_NAME>-platform-data
[INFO]     * Service Logs:  atai-<CUSTOMER_NAME>-service-logs
[INFO]   - Created IAM role: atai-platform-role
[INFO]   - Created service account: atai-platform-sa
```

Step 3: Kubernetes secrets required for the atai-platform services

Step 3.1 Generate values for the IAM service secret

Master Key (`IAM_MASTER_KEY`)

Used to authenticate **administrative operations**, such as creating and managing organizations and keys.

Purpose:

- Solves the *bootstrapping problem* — allows you to populate an empty database.

To generate:

```
None
openssl rand -base64 32
```

Server Salt (`IAM_SERVER_SALT`)

Used as the **salt input for Argon2** when hashing API keys.

Key Points:

- Must remain **persistent** for the lifetime of the service.
  (Changing it invalidates all existing keys.)
- Leaking it doesn't immediately compromise security **if keys are service-generated** (not manually uploaded).

To generate:

```
None
openssl rand -base64 48
```

After generation stores the secret in a secure place

IAM db name (`IAM_DB_NAME`)
Postgres database that was previously created called iam_db in the section PostgreSQL database configuration Step 4: Extra Database Configuration steps.

IAM db user (`IAM_DB_USER`)
atai_dev postgres user that was created in the PostgreSQL database configuration Step 4: Extra Database Configuration steps.

IAM db password (`IAM_DB_PASSWORD`)
atai_dev postgres user password that was created in the PostgreSQL database configuration Step 4: Extra Database Configuration steps.

IAM db port (`IAM_DB_PORT`)
Port of your PostgreSQL instance

IAM db host (`IAM_DB_HOST`)
Host of your PostgreSQL instance

⚠️ Store your user and password in a secure location. You will need them later in the *atai-platform* prerequisites.

Helm chart installation

Prerequisites

1. The kubectl command line tool is required. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster.
2. Version 2.12.3 or later or version 1.27.160 or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device.
3. Helm 3.20.0 or higher. Learn more about Helm installation here.

Step 1: Installation

1. Retrieve an authentication token and authenticate your clients. Enter the AWS CLI:

```
None
aws ecr get-login-password \
    --region us-west-2 | helm registry login \
    --username AWS \
    --password-stdin 337756366293.dkr.ecr.us-west-2.amazonaws.com
```

2. Create a tmp folder:

```
None
mkdir awsmp-chart && cd awsmp-chart
```

3. Create your ***values.yaml*** from the template included in the deploy kit:

```
Shell
cp atai-platform-deploy-kit/templates/values.yaml values.yaml
```

Note: Open ***values.yaml*** and follow the instructions at the top of the file to replace all placeholders with your actual configuration values. Each placeholder includes inline comments with format examples to guide you.

4. Install the **atai-platform** helm chart:

```
Shell
$ helm upgrade atai-platform atai-platform-1.0.3-163-a6c5863.tgz \
```

```
--namespace atai-platform \
--values values.yaml \
--install

Release "atai-platform" does not exist. Installing it now.
I1111 11:06:28.291728   35840 warnings.go:110] "Warning:
spec.template.spec.containers[0].env[9]: hides previous definition of
\"REDIS_USE_INSECURE_TLS\", which may be dropped when using apply"
NAME: atai-platform
LAST DEPLOYED: Tue Nov 11 11:06:23 2025
NAMESPACE: atai-platform
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Getting Started with the Archetype Platform

To get started with the Archetype Platform, please visit following documentation page:
https://docs.archetypeai.app/introduction/overview

Appendix

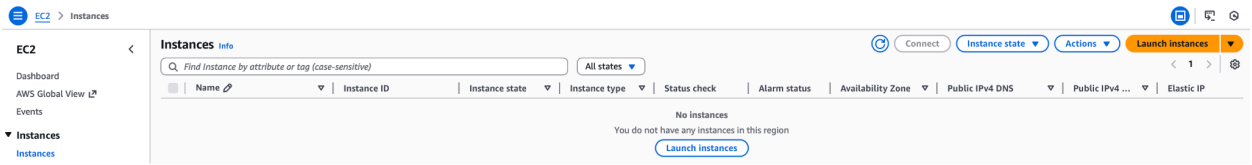
Bastion host configuration

Prerequisites

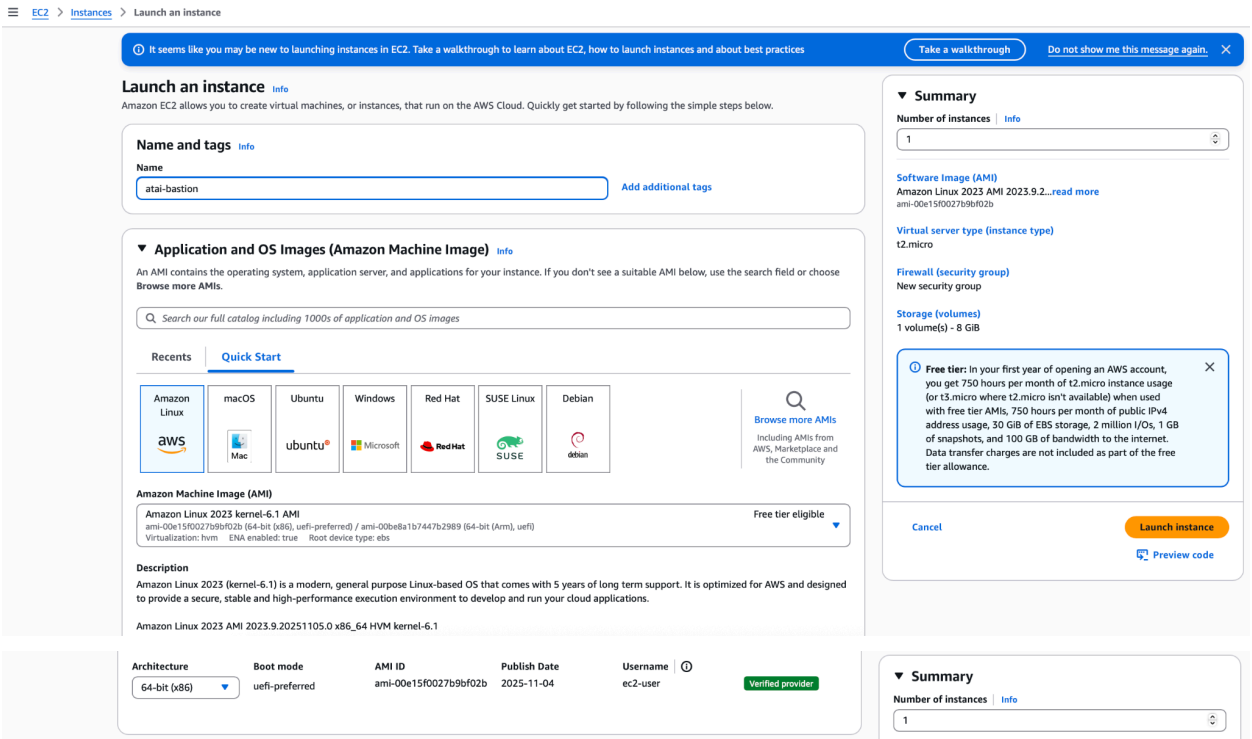
1. VPC with public subnet
2. Internet Gateway attached to VPC
3. Route table configured for public subnet

Step 1: Launch EC2 Instance (Bastion Host)

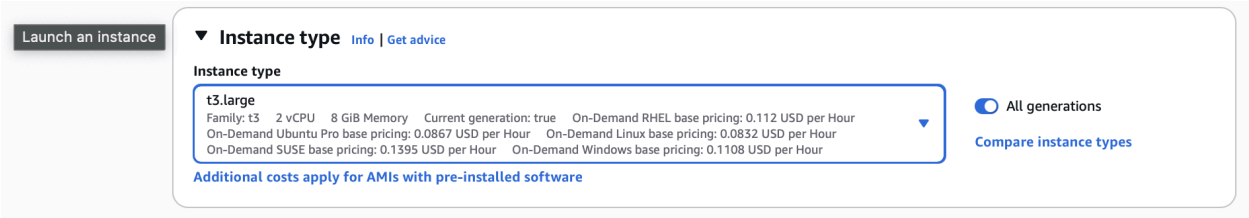
1. Go to EC2 → Instances → Launch instance



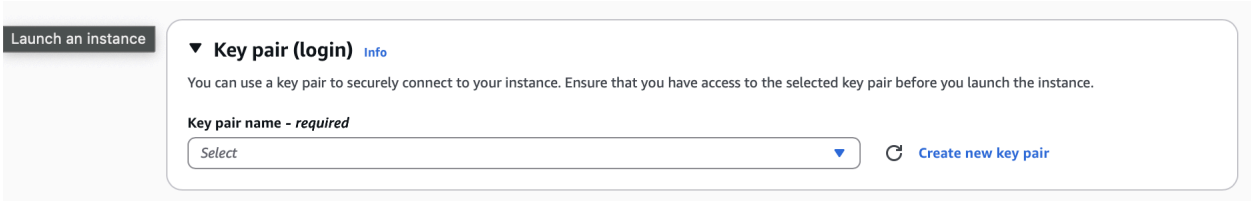
2. Name: atai-bastion
3. Application and OS Images (Amazon Machine Image):
    a. Search for: Amazon Linux 2023 AMI
    b. Select: Amazon Linux 2023 AMI (x86_64, HVM, kernel 6.1)



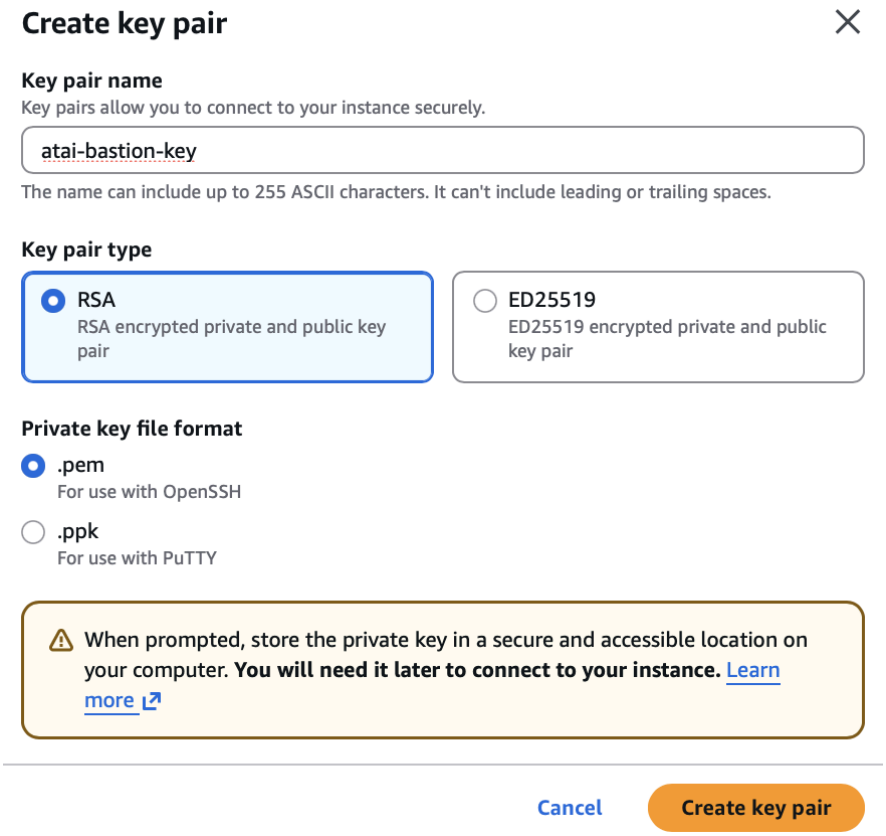
4. Instance type: Select your instance type (e.g., t3.medium for dev, t3.large for production)



5. Select an existing key pair or click on **Create a new key pair**

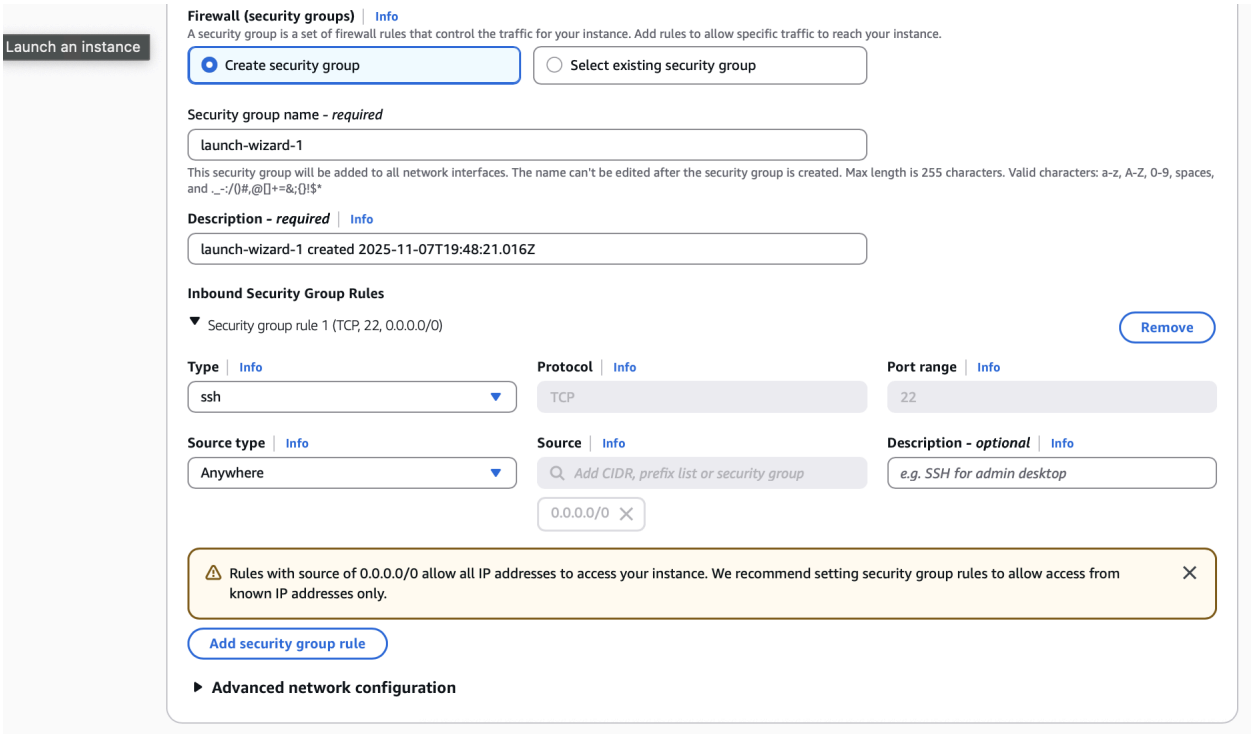


6. Assign a new atai-bastion-key, then click on **Create key pair**

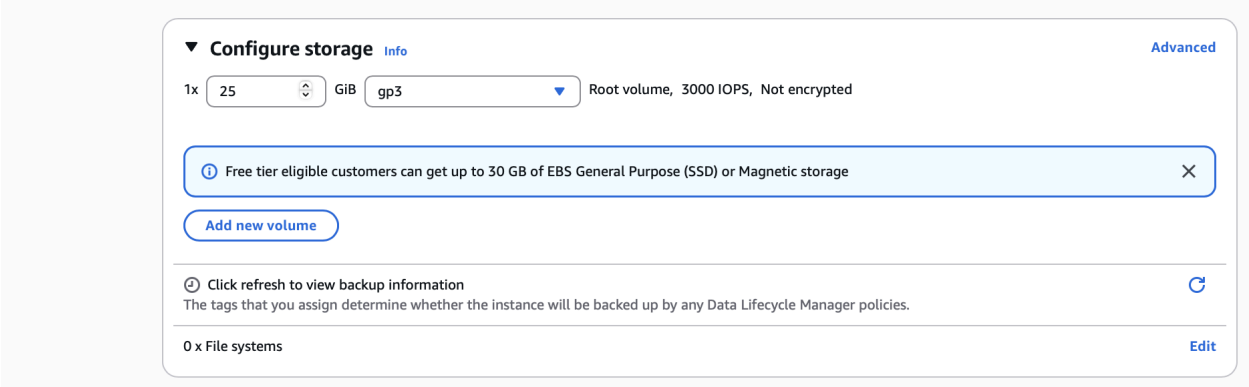


7. Network settings:
    a. VPC: Select your VPC
    b. Subnet: Select a public subnet (e.g., atai-platform-vpc-public-us-west-2a)
    c. Auto-assign public IP: Enable (or use Elastic IP from Step 4)

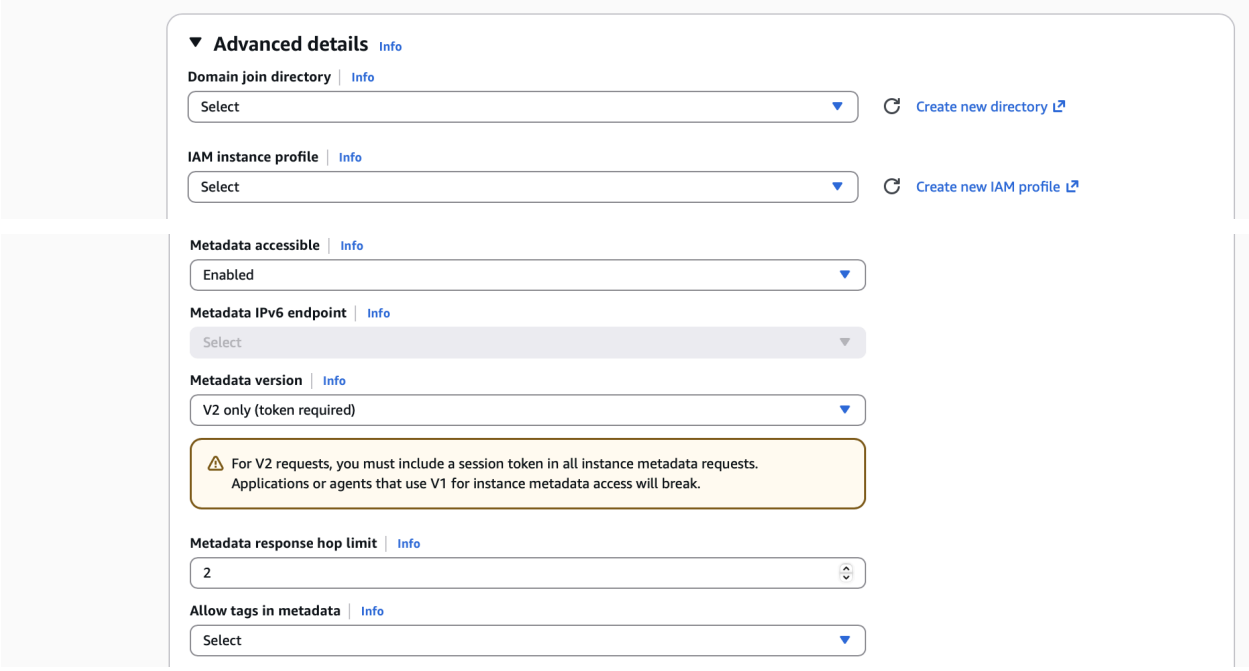
    d. Firewall (security groups): Select **Create security group**
        i. By default AWS will add an SSH rule



8. Configure storage
    a. Default (8 GiB gp3)
    b. Recommended 25 GB, but you can adjust based on your requirements

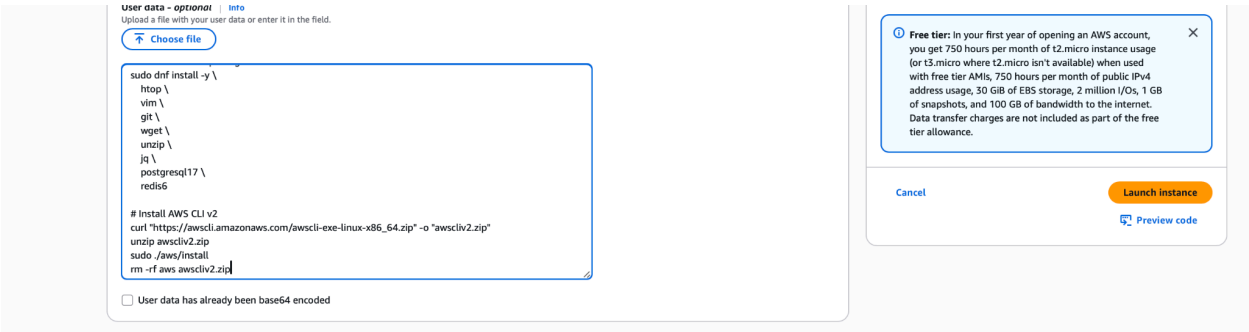


9. Advanced details (expand):
    a. Metadata accessible: Enable
    b. Metadata version: V2 only (token required) (IMDSv2)
    c. Metadata token response hop limit: 2

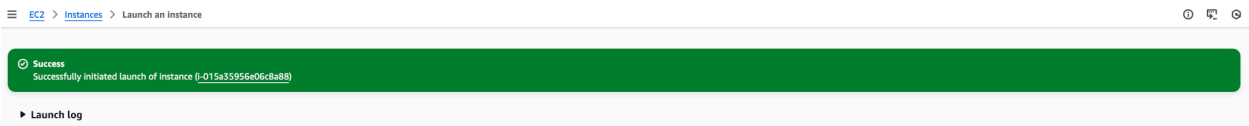


d. User data: Paste the following script:

```
None
#!/bin/bash

# Update system packages
sudo dnf update -y

# Install essential packages
sudo dnf install -y \
    htop \
    vim \
    git \
    wget \
    unzip \
    jq \
    postgresql17 \
    redis6

# Install AWS CLI v2
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
rm -rf aws awscliv2.zip
```

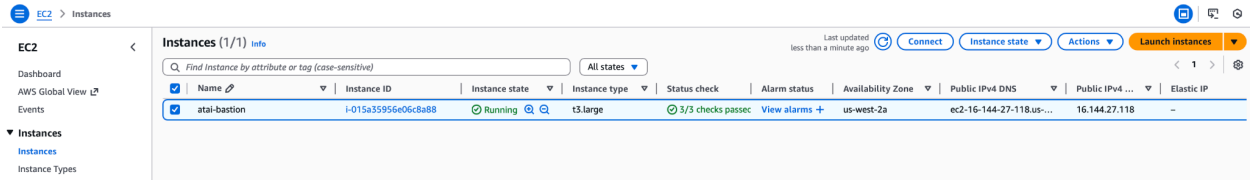


10. Click on **Launch instance**.

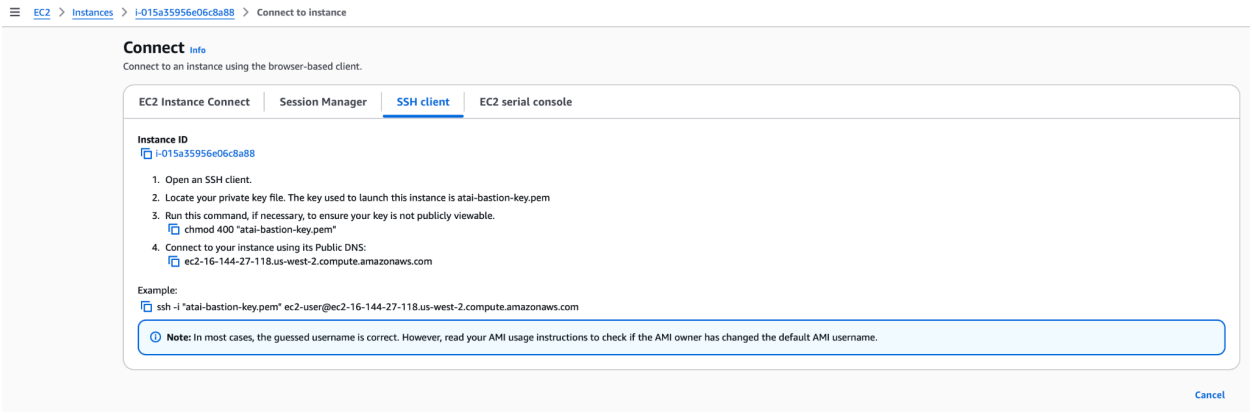


Step 2: Connect to your Bastion host

1. Go to EC2 → Instances → Select your atai-bastion instance and click on **Connect**



2. Click on the tab **SSH client**



   a. Open an SSH client.
   b. Locate your private key file. The key used to launch this instance is atai-bastion-key.pem
   c. Run this command, if necessary, to ensure your key is not publicly viewable.

```
None
chmod 400 "atai-bastion-key.pem"
```

   d. Connect to your instance using its Public DNS e.g,:

```
None
ec2-16-144-27-118.us-west-2.compute.amazonaws.com
```

Example

```
None
ssh -i "atai-bastion-key.pem"
ec2-user@ec2-16-144-27-118.us-west-2.compute.amazonaws.com
```

AWS Service Quotas

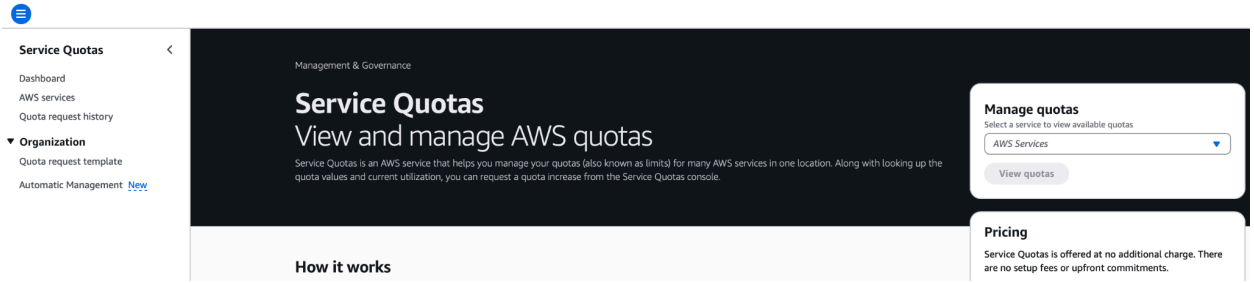
Running On-Demand G and VT instances

Requirements:

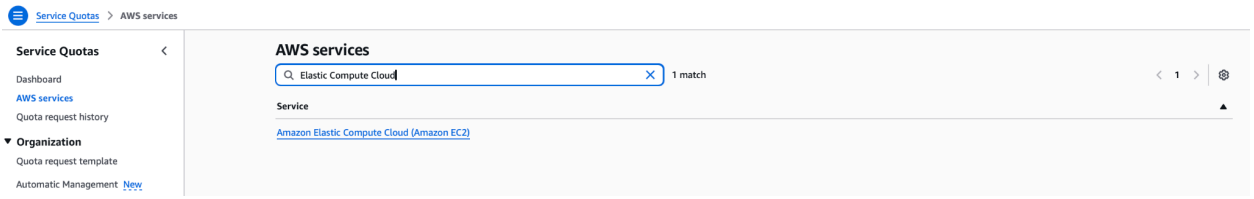
- Instance Type: g6e.2xlarge
- vCPUs per instance: 8 vCPUs
- Minimum instances required: 10
- Total vCPUs needed: 10 × 8 = 80 vCPUs

Service Quota to check:

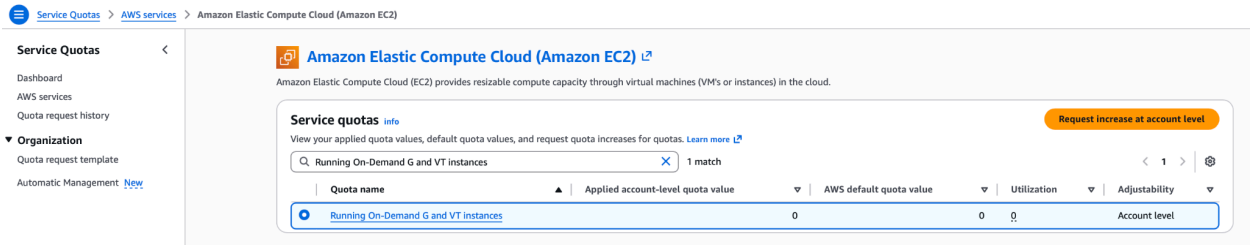
3. Go to AWS Service Quotas → In the left panel click on **AWS Services**



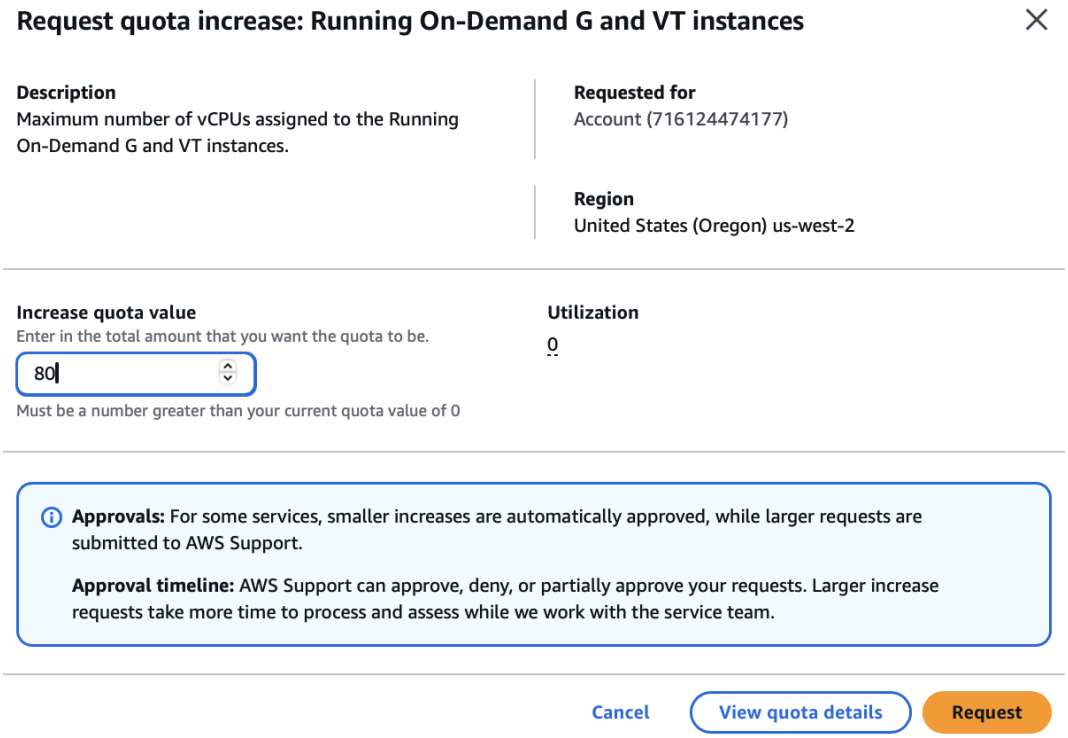
4. In the search bar type AWS Elastic Compute Cloud



5. In the search bar type **Running On-Demand G and VT instances**, select the quota and click on **Request increase at account level.**



6. Set the **Increase account value** to 80 vCPUs and click on **Request**



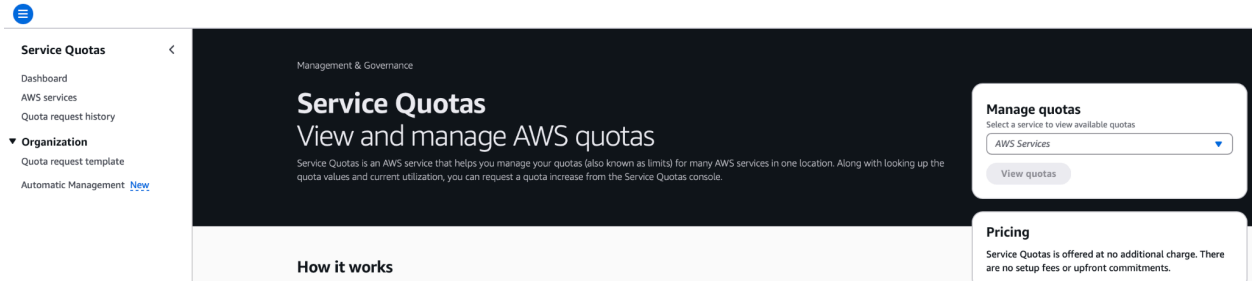
Running On-Demand Standard (A, C, D, H, I, M, R, T, Z) instances

Requirements:

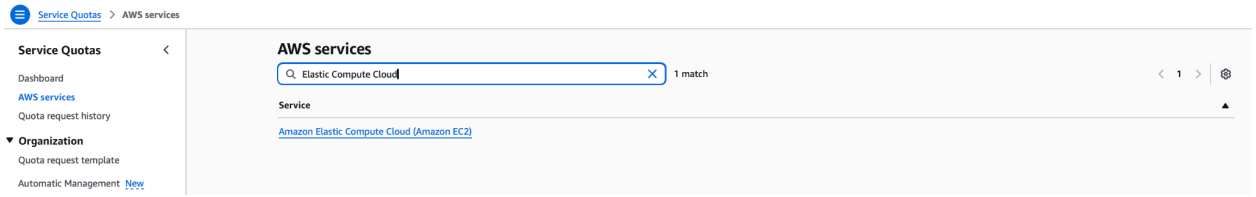
- Instance Type: m6i.4xlarge
- vCPUs per instance: 16 vCPUs
- Minimum instances required: 6
- Total vCPUs needed: 6 × 16 = 96 vCPUs

Service Quota to check:

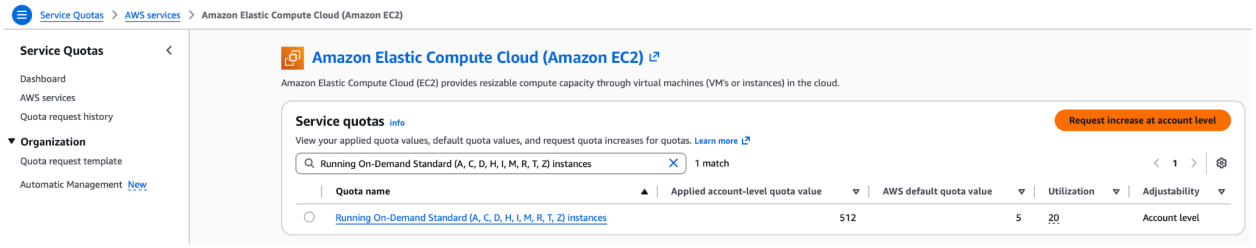
1. Go to AWS Service Quotas → In the left panel click on **AWS Services**



2. In the search bar type AWS Elastic Compute Cloud



3. In the search bar type **Running On-Demand Standard (A, C, D, H, I, M, R, T, Z) instances**, select the quota and click on **Request increase at account level.**



4. Set the **Increase account value** to 96 vCPUs and click on **Request**

Note: If your existing quota for Standard Instances (A, C, D, H, I, M, R, T, Z family) already exceeds 96 vCPUs, no increase is needed for this family.

EKS configuration - Install the NGINX ingress controller

The **atai-platform** Helm chart already bundles the NGINX Ingress Controller as a dependency subchart. There is no need to install a separate Helm chart or add external repositories.

Prerequisites

Before installing NGINX Ingress Controller, ensure you have:

1. EKS Cluster - Your Kubernetes cluster must be running and accessible
2. Public Subnets - You need the subnet IDs where the load balancer will be created
3. VPC ID - The VPC ID where your cluster is running

Enabling the controller

In your values.yaml, locate the ***ingress-nginx*** section and set enabled to ***true***:

```
None
ingress-nginx:
  enabled: true    # change from false to true
```

Then replace the two placeholders in the same section:

Placeholder	**Example**
<PLACEHOLDER_NLB_SECURITY_GROUP_ID>	sg-0a1b2c3d4e5f6g7h8
<PLACEHOLDER_PUBLIC_SUBNET_IDS>	subnet-xxxxxxxxx, subnet-yyyyyyyyy

The controller will be deployed automatically when you run ***helm upgrade --install***.

Note: Already have an NGINX Ingress Controller? Keep ***ingress-nginx.enabled: false*** and update the ***className*** field in each ingress block (***api-service-backend***, ***console-2-service-frontend***, ***iam-service-backend***) to match your existing IngressClass name.

EKS configuration - Configure Cert-Manager and Let's Encrypt

The atai-platform Helm chart already bundles cert-manager as a dependency subchart. There is no need to install a separate Helm chart or add external repositories.

Enabling cert-manager

In your values.yaml, locate the ***cert-manager*** section and set enabled to ***true***:

```
None
cert-manager:
  enabled: true    # change from false to true
```

Cert-manager will be deployed automatically when you run ***helm upgrade --install***. No additional placeholders are required.

Create a ClusterIssuer for Let's Encrypt

After the Helm install completes, create a ClusterIssuer to enable automatic SSL certificate provisioning:

```
None
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: <YOUR_EMAIL>
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
      - http01:
          ingress:
            class: atai-nginx
```

Replace ***<YOUR_EMAIL>*** with a valid email address for certificate expiration notifications.

Apply it:

```shell
kubectl apply -f cluster-issuer.yaml
```

MANUAL SETUP INSTRUCTIONS FOR EKS IRSA (IAM ROLES FOR SERVICE ACCOUNTS)

PREREQUISITES

Before starting, gather the following information:

- AWS Region: (e.g., us-west-2)
- Customer Name: Maximum 15 characters (e.g., acme)
- EKS Cluster Name: (e.g., atai-platform)
- Platform Data Bucket Name: (e.g., atai-acme-platform-data)
- Service Logs Bucket Name: (e.g., atai-acme-service-logs)
- AWS Account ID: Your 12-digit AWS account ID

STEP 1: VERIFY PREREQUISITES

1. Confirm the EKS cluster exists and is accessible
2. Verify both S3 buckets exist:
   - Platform Data Bucket
   - Service Logs Bucket
3. Ensure the EKS cluster has an OIDC provider associated (required for IRSA)

STEP 2: ASSOCIATE IAM OIDC PROVIDER (IF NOT ALREADY DONE)

If your cluster doesn't already have an OIDC provider associated:

1. Navigate to EKS Console → Select your cluster
2. Go to the Configuration tab
3. Under OpenID Connect provider URL, note the OIDC provider URL
4. In IAM Console → Identity Providers, verify an identity provider exists for this OIDC URL
5. If not, create one:
   - Provider Type: OpenID Connect
   - Provider URL: (from EKS cluster)
   - Audience: sts.amazonaws.com

STEP 3: CREATE IAM POLICIES

You need to create THREE IAM policies. You'll need the policy JSON documents from the policies/ directory.

POLICY 1: PLATFORM DATA ACCESS

Policy Name: atai-{customer-name}-platform-data-access
Policy Document (substitute {PLATFORM_DATA_BUCKET_ARN}):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::{platform-data-bucket-name}"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:::{platform-data-bucket-name}/*"
    }
  ]
}
```

POLICY 2: SERVICE LOGS ACCESS

Policy Name: atai-{customer-name}-service-logs-access
Policy Document (substitute {SERVICE_LOGS_BUCKET_ARN}):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::{service-logs-bucket-name}"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:::{service-logs-bucket-name}/*"
    }
  ]
}
```

POLICY 3: MODEL DEPOT ACCESS

Policy Name: atai-{customer-name}-model-depot-access
Policy Document (uses fixed bucket: atai-marketplace-model-depot):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::atai-marketplace-model-depot"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::atai-marketplace-model-depot/*"
    }
  ]
}
```

NOTE: Record the ARN of each policy after creation. The format will be:
arn:aws:iam::{account-id}:policy/{policy-name}

STEP 4: CREATE KUBERNETES NAMESPACE

Create a namespace in your EKS cluster:
Namespace Name: atai-platform

Using kubectl:

```
kubectl create namespace atai-platform
```

Or apply via YAML:

```
apiVersion: v1
    kind: namespace
    metadata:
      name: atai-platform
```

STEP 5: CREATE IAM ROLE FOR SERVICE ACCOUNT

Create an IAM role with the following configuration:

Role Name: atai-platform-role
Trust Relationship (substitute values):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated":
"arn:aws:iam::{account-id}:oidc-provider/oidc.eks.{region}.amazonaws.com/id
/{oidc-id}"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.{region}.amazonaws.com/id/{oidc-id}:sub":
"system:serviceaccount:atai-platform:atai-platform-sa",
          "oidc.eks.{region}.amazonaws.com/id/{oidc-id}:aud":
"sts.amazonaws.com"
        }
      }
    }
  ]
}
```

Where:

- {account-id}: Your AWS account ID
- {region}: Your AWS region
- {oidc-id}: The OIDC provider ID (find in EKS cluster details)

ATTACH POLICIES TO ROLE

Attach the following SIX policies to the IAM role:

1. arn:aws:iam::{account-id}:policy/atai-{customer-name}-platform-data-access
2. arn:aws:iam::{account-id}:policy/atai-{customer-name}-service-logs-access
3. arn:aws:iam::{account-id}:policy/atai-{customer-name}-model-depot-access
4. arn:aws:iam::aws:policy/AWSMarketplaceMeteringFullAccess (AWS managed)
5. arn:aws:iam::aws:policy/AWSMarketplaceMeteringRegisterUsage (AWS managed)
6. arn:aws:iam::aws:policy/service-role/AWSLicenseManagerConsumptionPolicy (AWS managed)

STEP 6: CREATE KUBERNETES SERVICE ACCOUNT

Create a Kubernetes service account with the IAM role annotation:
Service Account Name: atai-platform-sa
Namespace: atai-platform

Apply the following YAML (substitute {role-arn}):

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: atai-platform-sa
  namespace: atai-platform
  annotations:
    eks.amazonaws.com/role-arn:
arn:aws:iam::{account-id}:role/atai-platform-role
```

STEP 7: VERIFICATION

Verify the setup is complete:

1. Check IAM Policies exist: Navigate to IAM → Policies and confirm all 3 custom policies are created

2. Check IAM Role exists: Navigate to IAM → Roles and verify atai-platform-role has 6 policies attached

3. Check Kubernetes resources:

```
kubectl get namespace atai-platform
kubectl get serviceaccount atai-platform-sa -n atai-platform
kubectl describe serviceaccount atai-platform-sa -n atai-platform
```

4. Verify the service account has the correct annotation with the IAM role ARN

SUMMARY OF CREATED RESOURCES

- Namespace: atai-platform
- Service Account: atai-platform-sa
- IAM Role: atai-platform-role
- IAM Policies:
  - atai-{customer-name}-platform-data-access
  - atai-{customer-name}-service-logs-access
  - atai-{customer-name}-model-depot-access

IMPORTANT NOTES

- Replace {customer-name} with your actual customer name throughout
- The customer name must be 15 characters or less
- The OIDC provider ID can be found in your EKS cluster's configuration page
- If policies already exist, you can create a new version instead of creating a new policy
- The service account annotation eks.amazonaws.com/role-arn is critical for IRSA to work