AWS CLOUD SOLUTION DOCUMENTATION FOR XAI-SERVICE

Electrical and Computer Engineering
Summer 2023

Dr. Yan Liu.

Abstract:

This documentation explains the procedure to deploy the xai-service in two methods. One without using terraform and the other with terraform.

Table of Contents

1. Prerequisite Installations

Note: LINUX Subsystem.

It is better to have WSL in windows than using Ubuntu. Please avoid ARM based processor for this application.

1.1 VMbox and Ubuntu:

STEP 1. Download and install VMBox depending on the OS.
STEP 2. Download the ISO image to install Linux.
STEP 3. Launch VMBox to create a new VM with the image downloaded with the following constraints,
   a. Username and password.
   b. The number of CPUs is 5.
   c. The size of the HDD is about 100GB.

1.2 Anaconda Installation

STEP 1. Open a new terminal.
STEP 2. Download Anaconda for Linux from their official page.
STEP 3. Execute: chmod +x Anaconda3-2023.03-1-Linux-x86_64.sh
STEP 4. Execute: ./Anaconda3-2023.03-1-Linux-x86_64.sh
STEP 5. Accept Terms and Conditions
STEP 6. Enter the installation directory as “home/<user>/anaconda”
STEP 7. Execute: nano ~/.bashrc
STEP 8. Add the path: export PATH="/path/to/anaconda3/bin:$PATH"
STEP 9. Ctrl+X-> y -> Enter.
STEP 10. source ~/.bashrc
STEP 11. Open a new terminal.
STEP 12. Execute: su -> give the user credentials.
STEP 13. Execute conda init bash.
STEP 14. Check the conda installation by executing “conda --version”.
STEP 15. Clone the repository and cd to the working directory.
STEP 16. Create new environment: conda create -n xai39 python=3.9
STEP 17. Execute: conda activate xai39
STEP 18. Try running the application locally before dockerising the application.

1.3 Docker Installation

STEP 1. Open a new terminal in sudo access (“su”).

STEP 2. Execute the following commands consequently,

a. apt update
b. apt install apt-transport-https ca-certificates curl software-properties-common
c. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
d. echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
e. apt update
f. apt install docker-ce docker-ce-cli containerd.io
g. systemctl status docker
h. usermod -aG docker $USER

STEP 3. docker –version

STEP 4. docker login

STEP 5. Verify the login executed successfully.

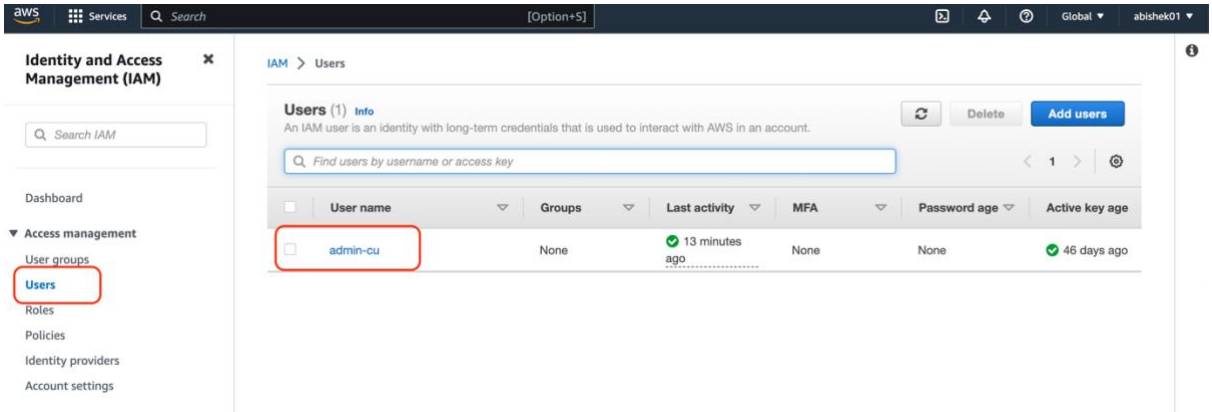
1.4 AWS CLI Installation

STEP 1. Execute the command as sudo user in the new terminal “apt-get install awscli”

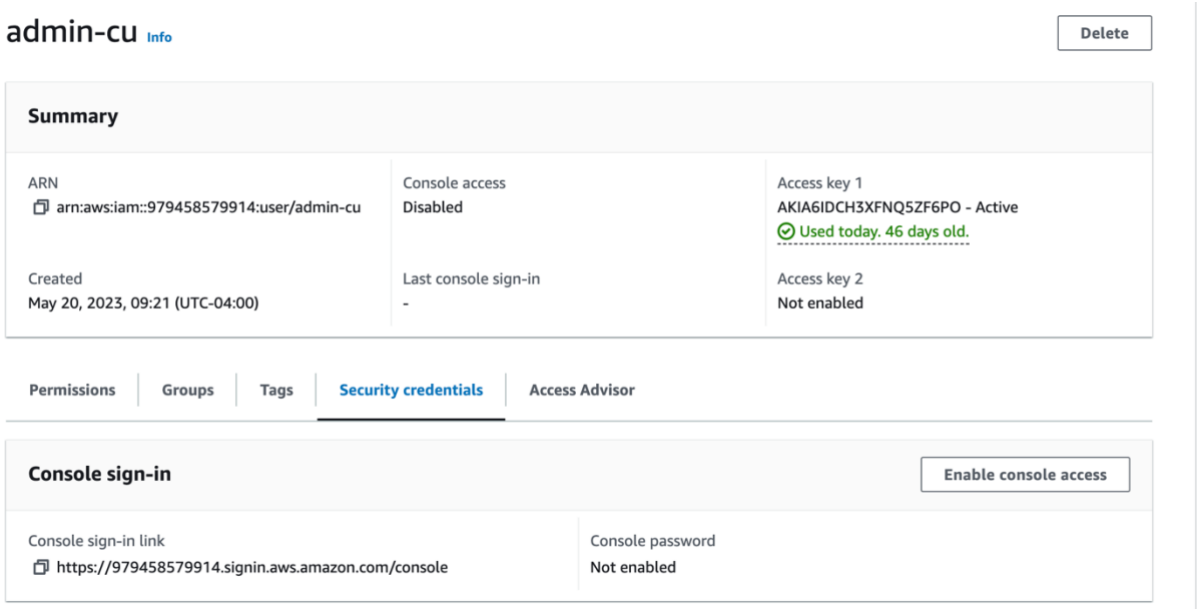
STEP 2. Use “aws configure” to configure the aws credentials for cli access.

STEP 3. How to get the AWS keys for login.

STEP 4. Go to IAM and select Users on the left tab.

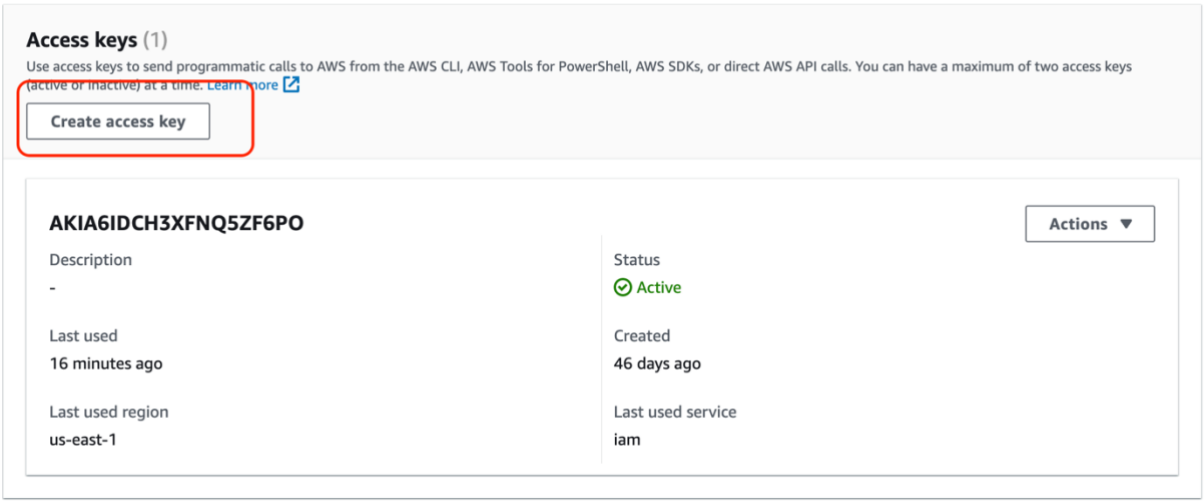


STEP 5. Select the user and go to “Security Credentials”.

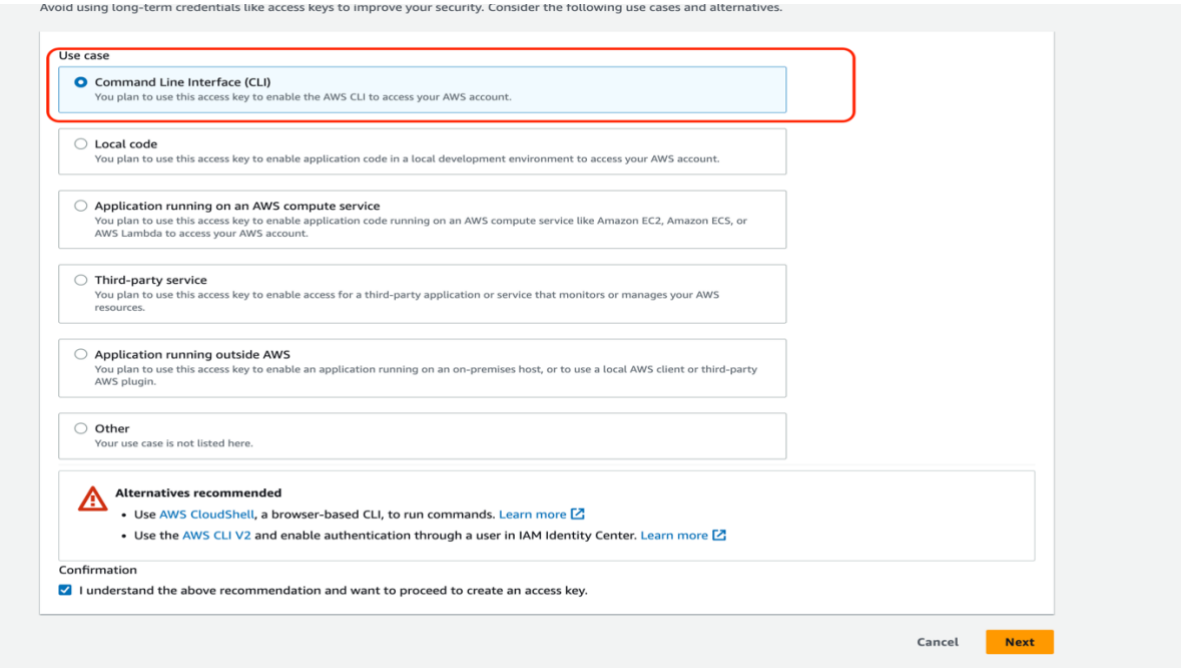


STEP 6. Now scroll to find “Access Keys”.

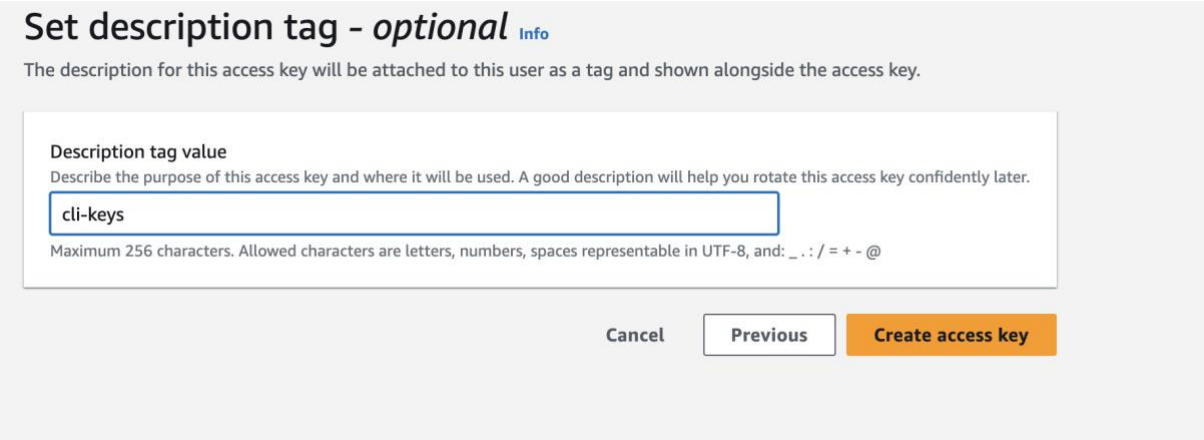
STEP 7. Create access key.



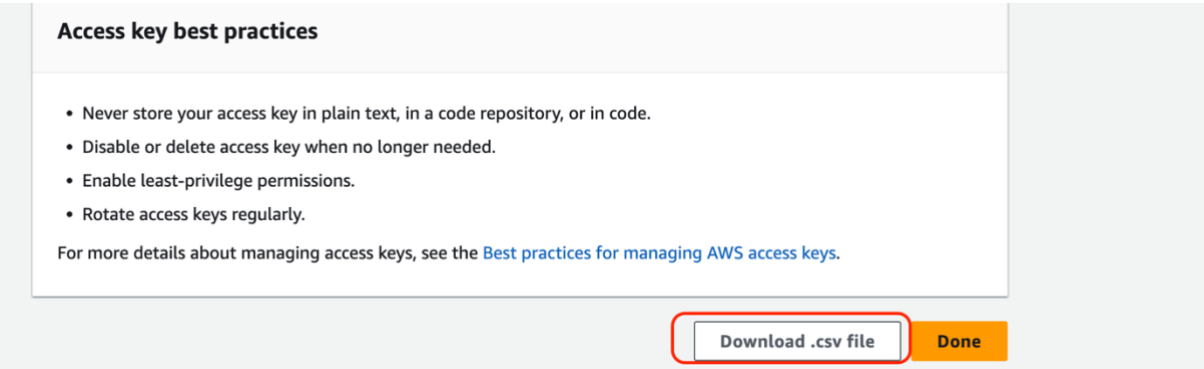
STEP 8. Choose CLI.



STEP 9. Provide a name.



STEP 10. Download the keys to local.



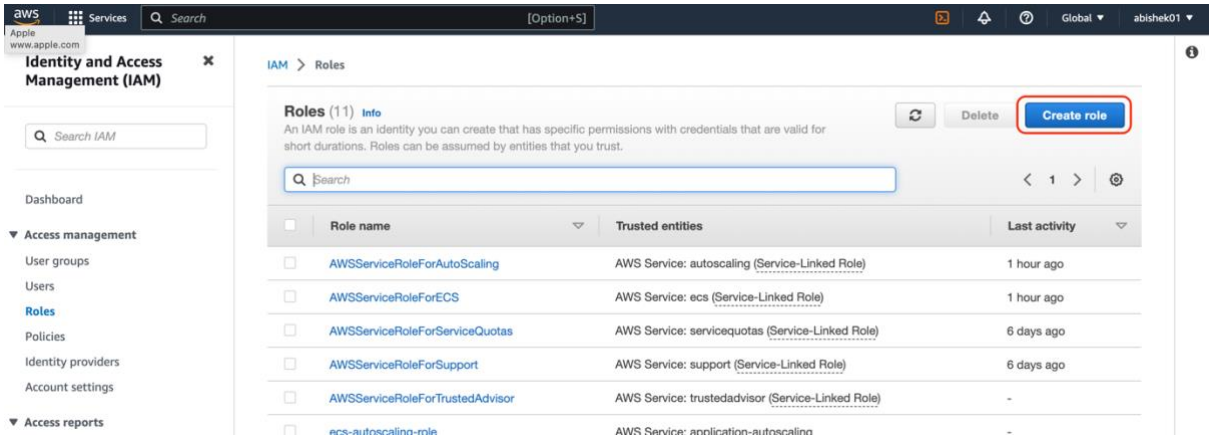
STEP 11. Now use the creds for “aws configure”, with region as us-east-1.

2. IAM role creation – W/O Terraform.

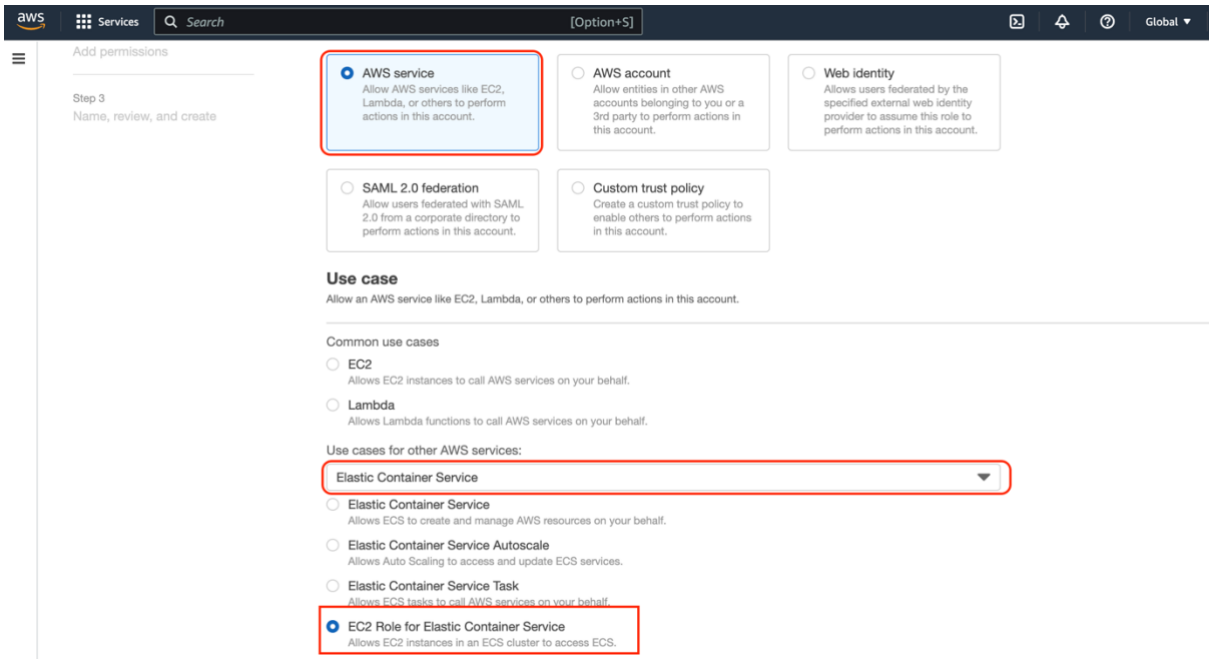
Open IAM in AWS Console.

2.1. ECS – Instance Role

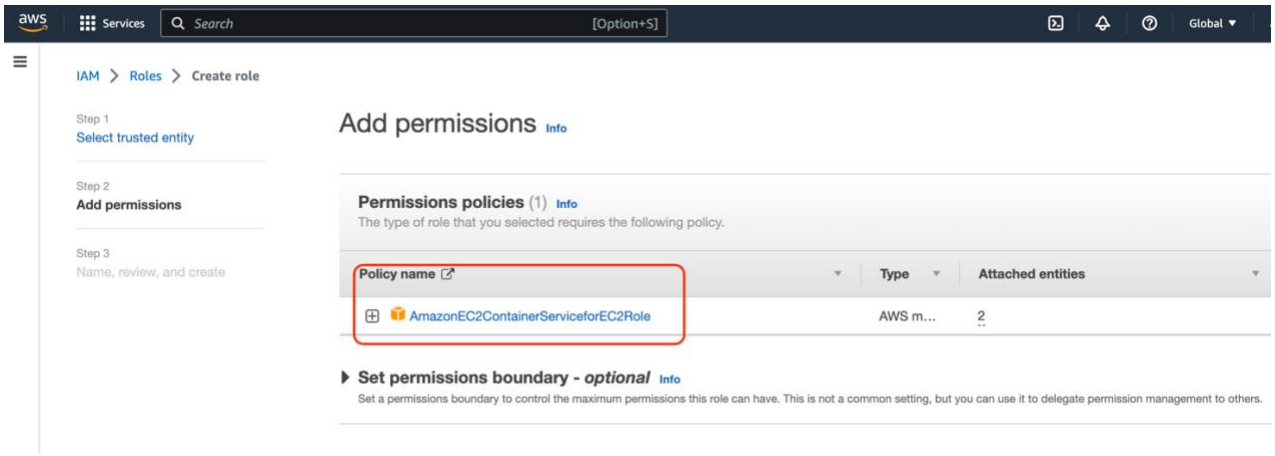
Click on Roles in the left side tab and click “create role”.



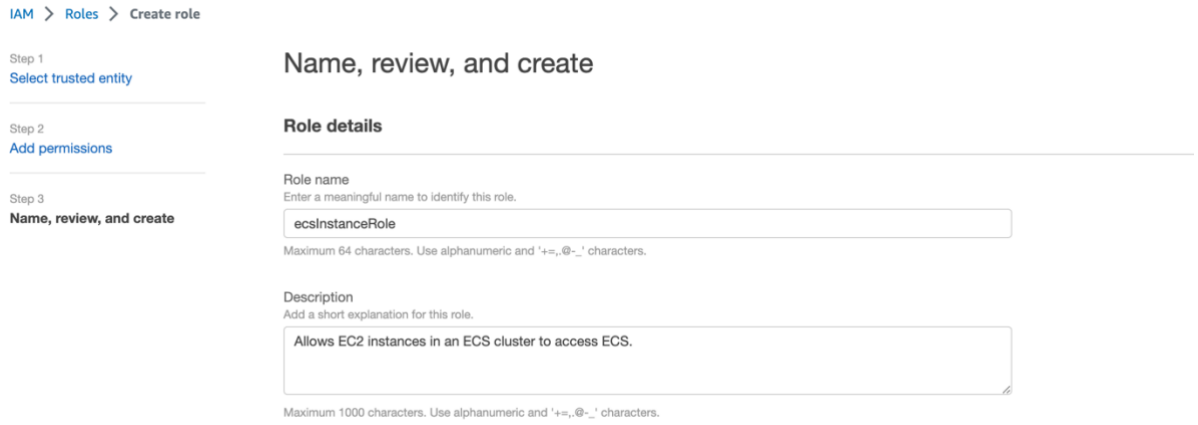
Select AWS Service, choose “Elastic Container Service” and click EC2 role for Elastic Container Service.



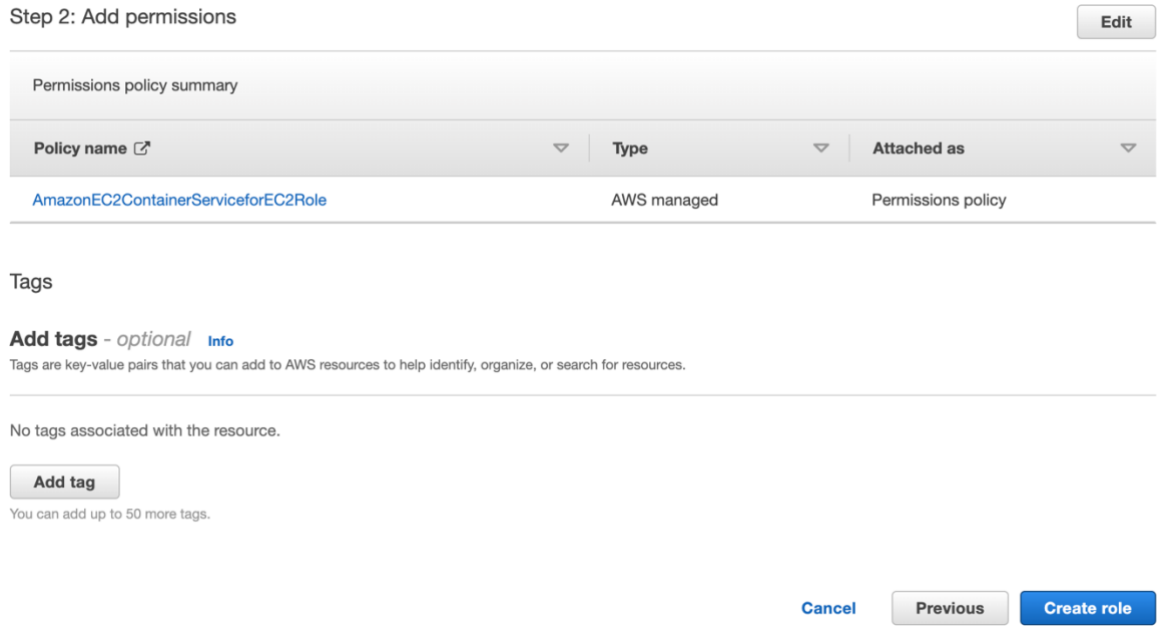
Verify the following policy is added to the role.



Give the role name “ecsInstanceRole”.

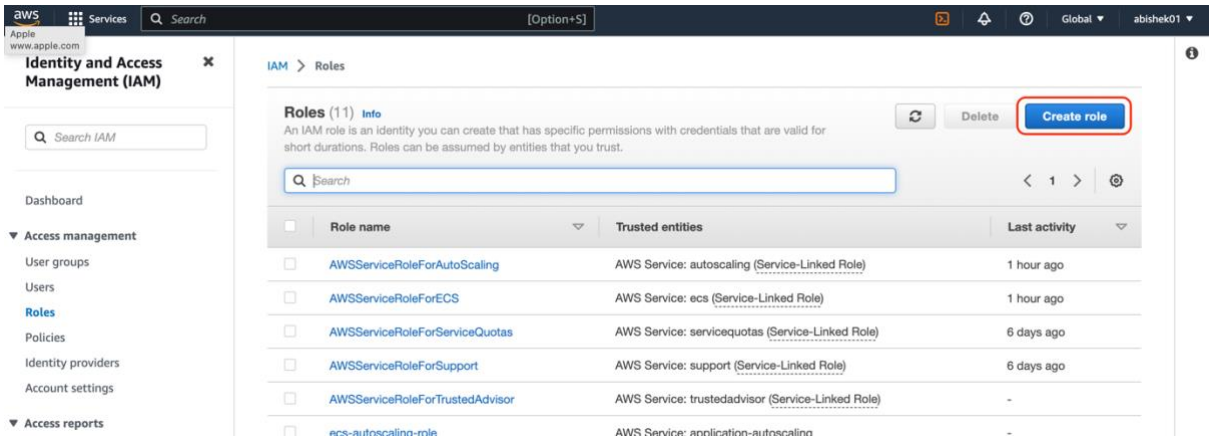


Verify the following in step 2 and create the cluster.

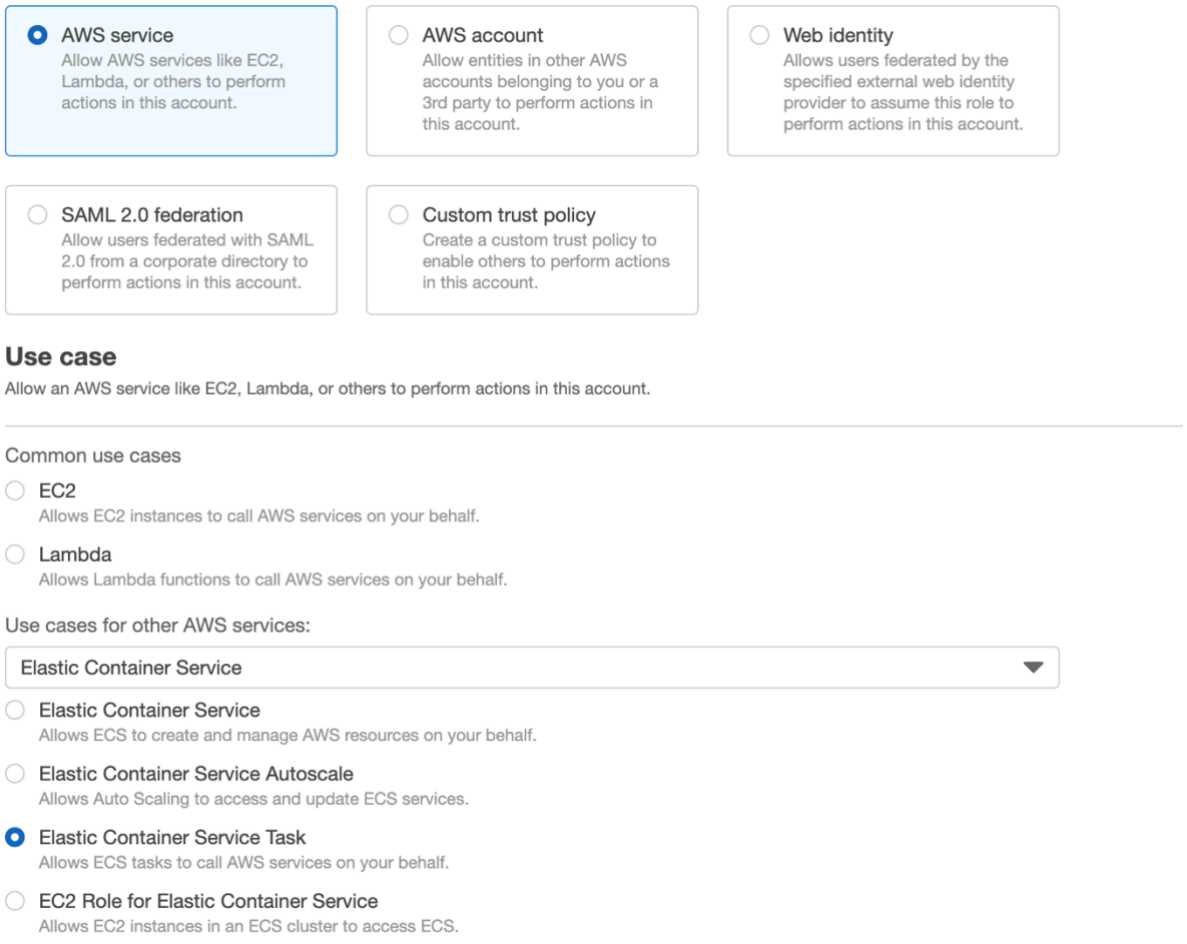


2.2. ECS – Task Execution Role

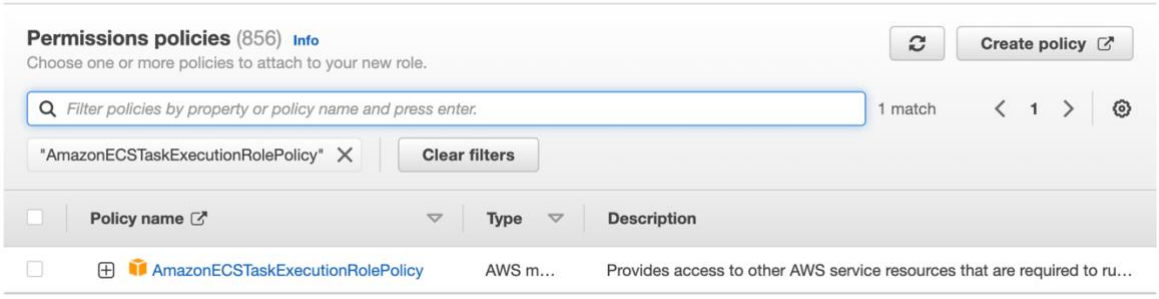
Click on Roles in the left side tab and click “create role”.



Select AWS Service, choose “Elastic Container Service” and click Elastic Container Service Task.



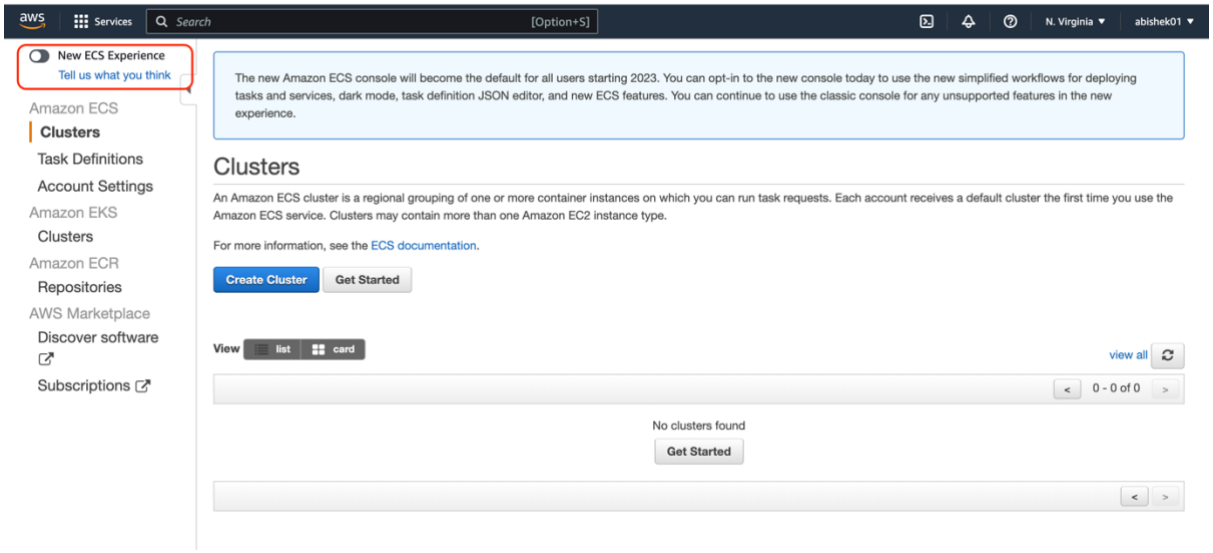
In “Add permissions” select “AmazonECSTaskExecutionRolePolicy”, check the checkbox, and click next. Give the role name “ecsTaskExecutionRole” and click create.


Permissions policies (856) Info
Choose one or more policies to attach to your new role.
Create policy
Filter policies by property or policy name and press enter.
1 match
1
"AmazonECSTaskExecutionRolePolicy"
Clear filters
Policy name
Type
Description
AmazonECSTaskExecutionRolePolicy
AWS m…
Provides access to other AWS service resources that are required to ru…

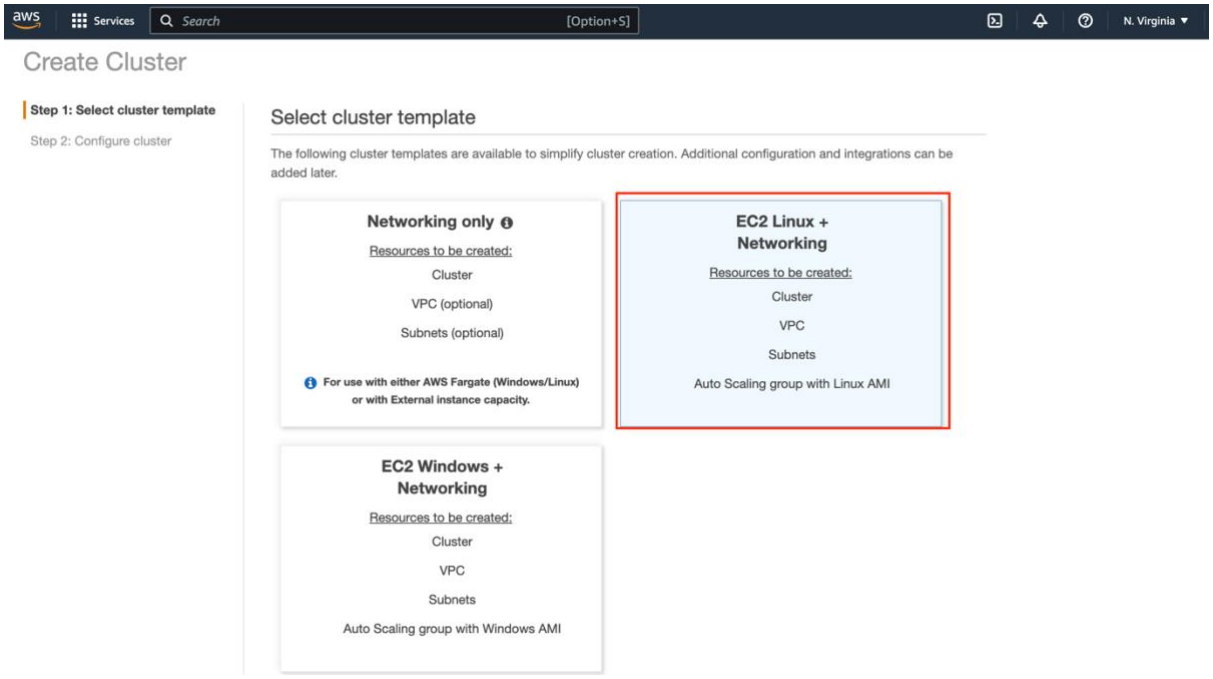
3. Deployment Using AWS console without terraform.

3.1. Creating a cluster.

Login to AWS Console. Open Amazon Elastic Container Service and disable the “New ECS Experience” to create a cluster.



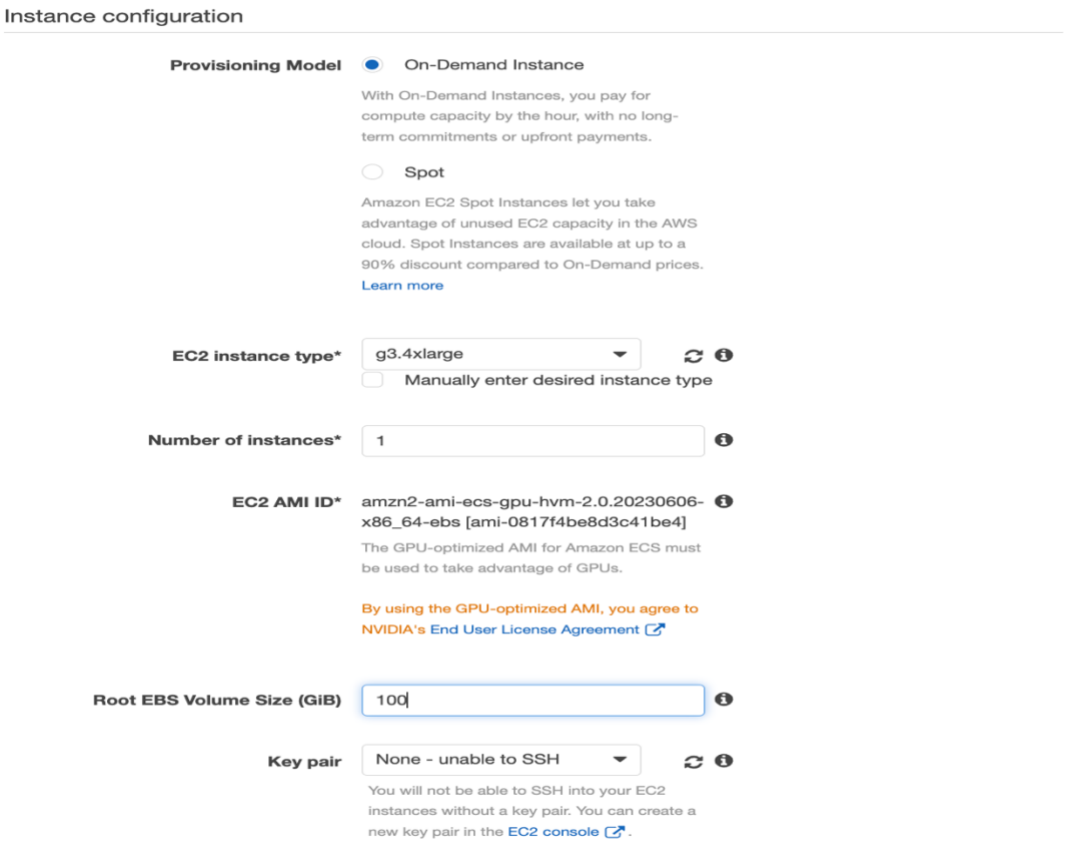
Start creating a cluster by clicking the create cluster. Choose “EC2 Linux + Networking” as the cluster template.



Provide a cluster name, for instance, configuration, and use the “On-Demand Instance” as the provisioning model. EC2 instance type as “**g3.4xlarge/ g3s.xlarge**”. Number

of instances as 1. Root EBS Volume Size (GiB) as 100. For key pair, if you already have .pem key to ssh, we can use it else we can choose “None – Unable to SSH”.

*: **g3.4xlarge/ g3s.xlarge** is used because the instance has GPU which needs XAI-Service.



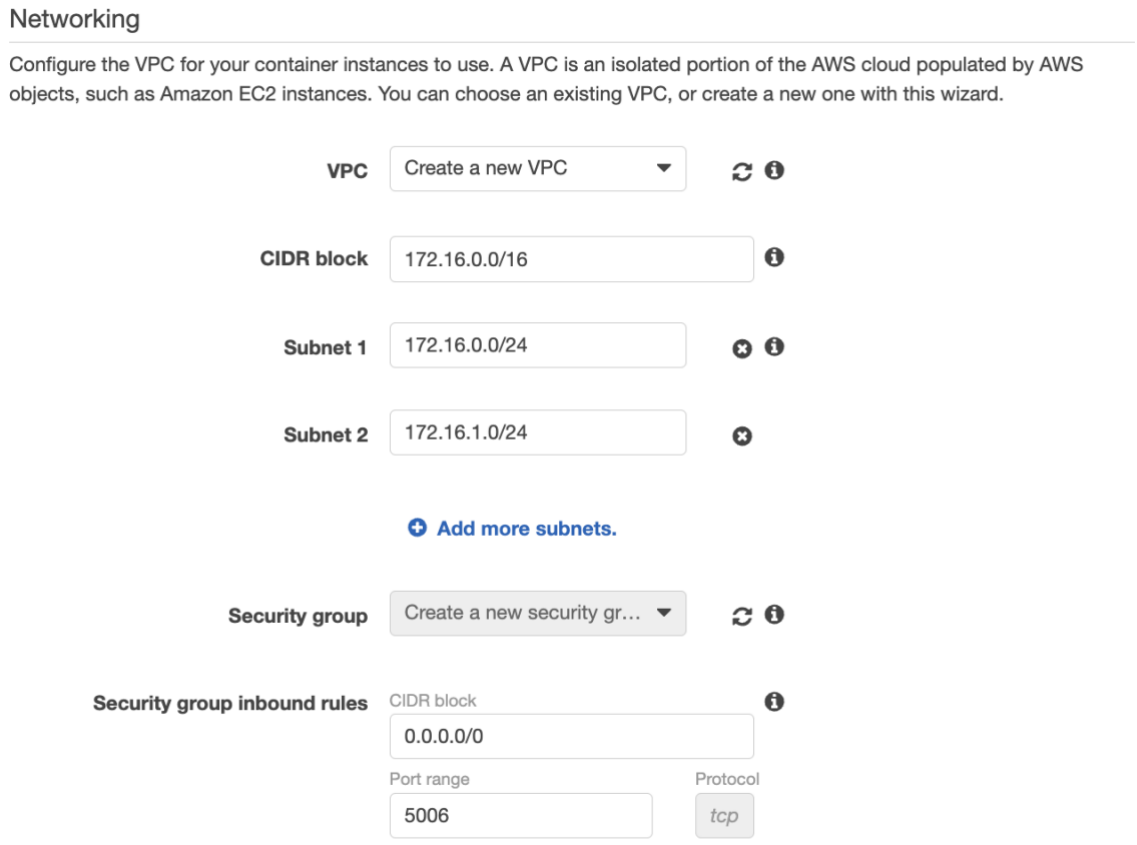
Note: Instance- g3.4xlarge/ g3s.xlarge

Amazon EC2 G3 instances are the latest generation of Amazon EC2 GPU graphics instances that deliver a powerful combination of CPU, host memory, and GPU capacity. G3 instances are ideal for graphics-intensive applications such as 3D visualizations, mid to high-end virtual workstations, virtual application software, 3D rendering, application streaming, video encoding, gaming, and other server-side graphics workloads.

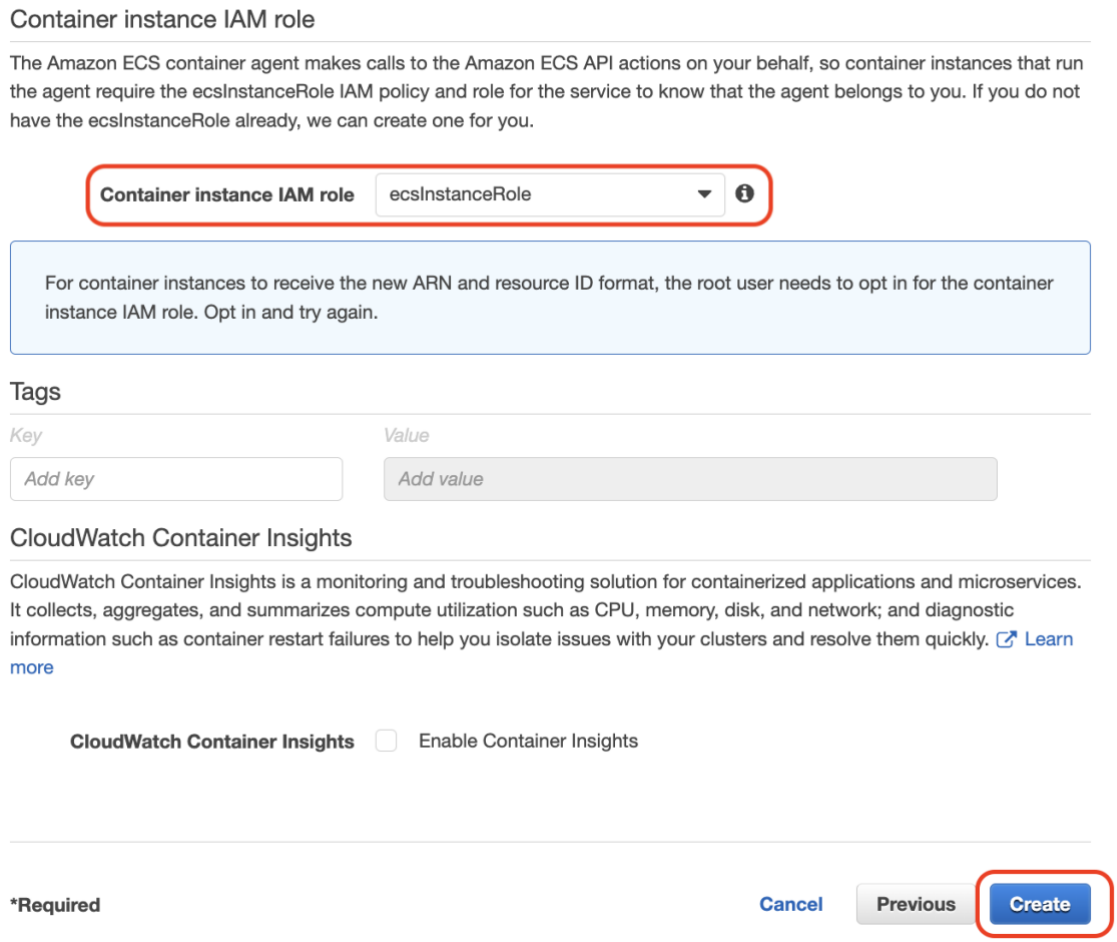
G3 instances provide access to NVIDIA Tesla M60 GPUs, each with up to 2,048 parallel processing cores, 8 GiB of GPU memory, and a hardware encoder supporting up to 10 H.265 (HEVC) 1080p30 streams and up to 18 H.264 1080p30 streams. With the latest driver releases, these GPUs provide support for OpenGL, DirectX, CUDA, OpenCL, and Capture SDK (formerly known as GRID SDK).

Name	GPUs	vCPU	Memory (GiB)	GPU Memory (GiB)	Price/hr* (Linux)
g3s.xlarge	1	4	30.5	8	$0.75
g3.4xlarge	1	16	122	8	$1.14

Continue to Create a cluster, in Network Configuration select vpc as “create a new vpc”. CIDR block as “172.16.0.0/16”. Subnet 1 and Subnet 2 as “172.16.0.0/24” and “172.16.1.0/24” respectively. Security group as “Create new security group”. Security group inbound rules, CIDR block “0.0.0.0/0” and **port range as 5006** depending on microservice. *Once the security group is created, we can give more inbound rules with all the port ranges for the xai-service microservices.

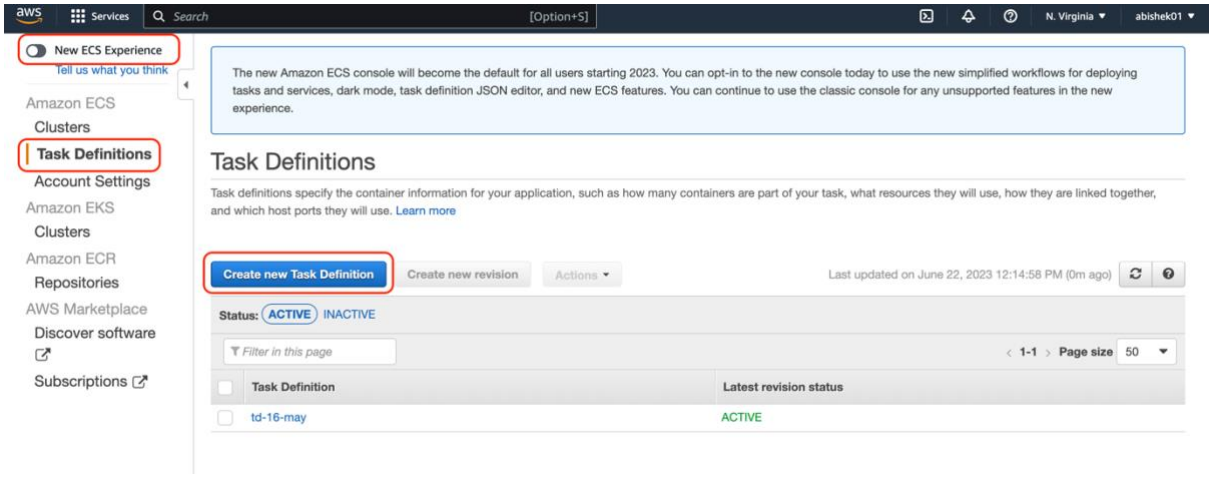


From the Container instance IAM role configuration as “ecsInstanceRole” and click “create”.

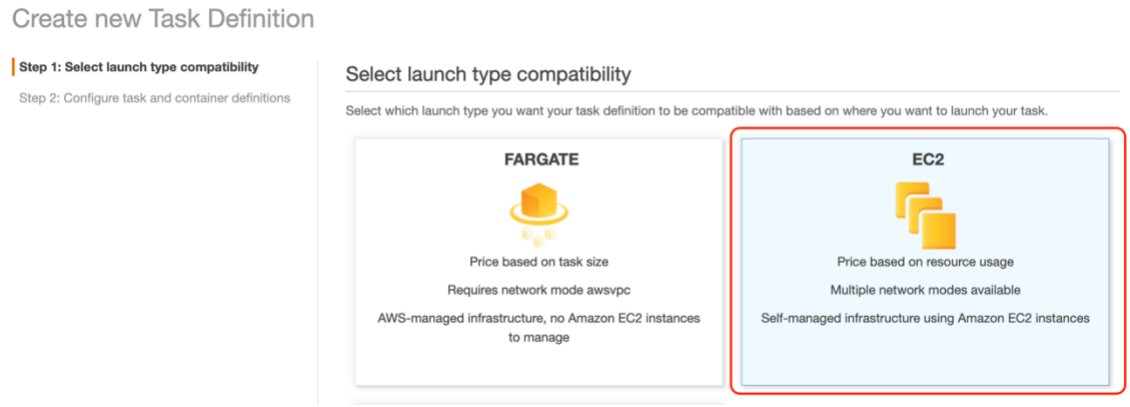


3.2. Creating a task definition - TD.

Login to AWS Console. Open Amazon Elastic Container Service and disable the “New ECS Experience”. Click the Task Definitions on the left tab of ECS to create a TD.

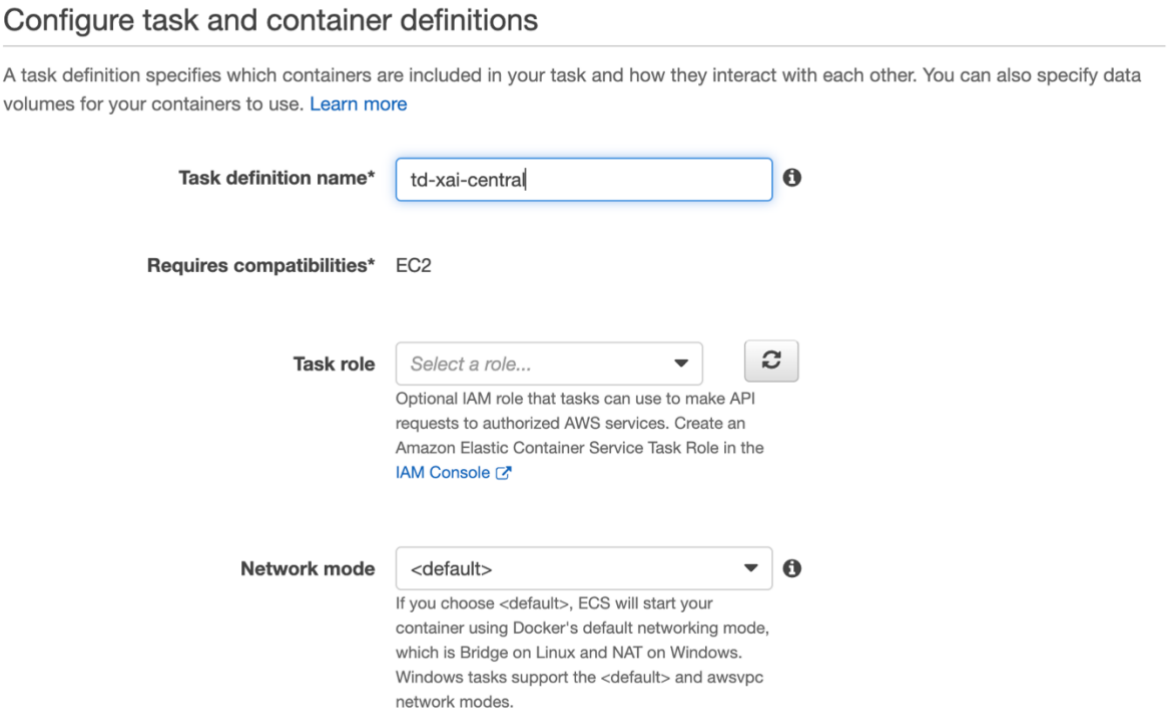


Click “Create a new Task Definitions” and select launch type compatibility as “EC2”.



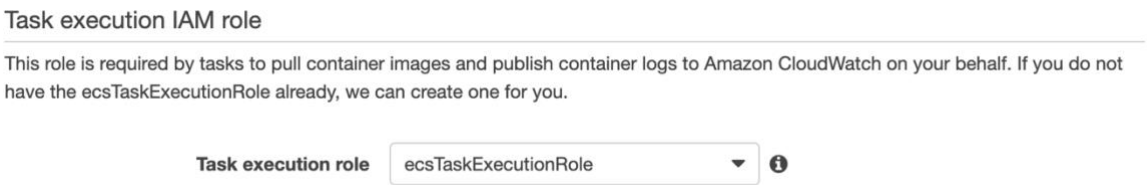
3.2.1. Configure task and container definitions.

1. The task definition name is “td-xai-central”.
2. Task Role is none or left untouched. [If an issue arises create an IAM role as None, better left untouched].
3. Network mode as default.



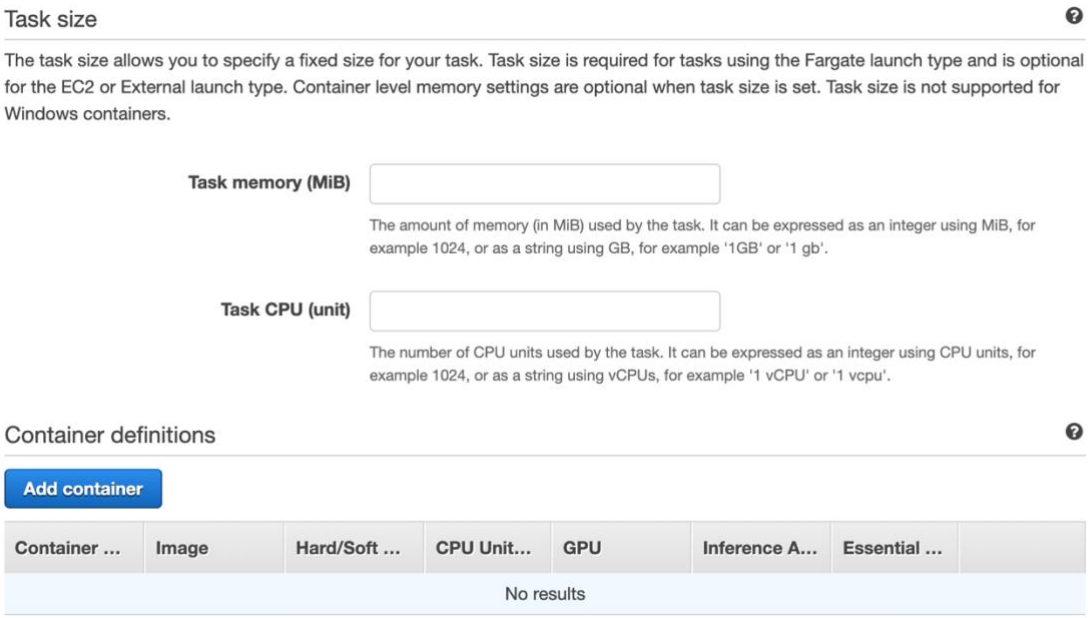
3.2.2. Task execution IAM role

1. Task execution role as “ecsTaskExecutionRole”.

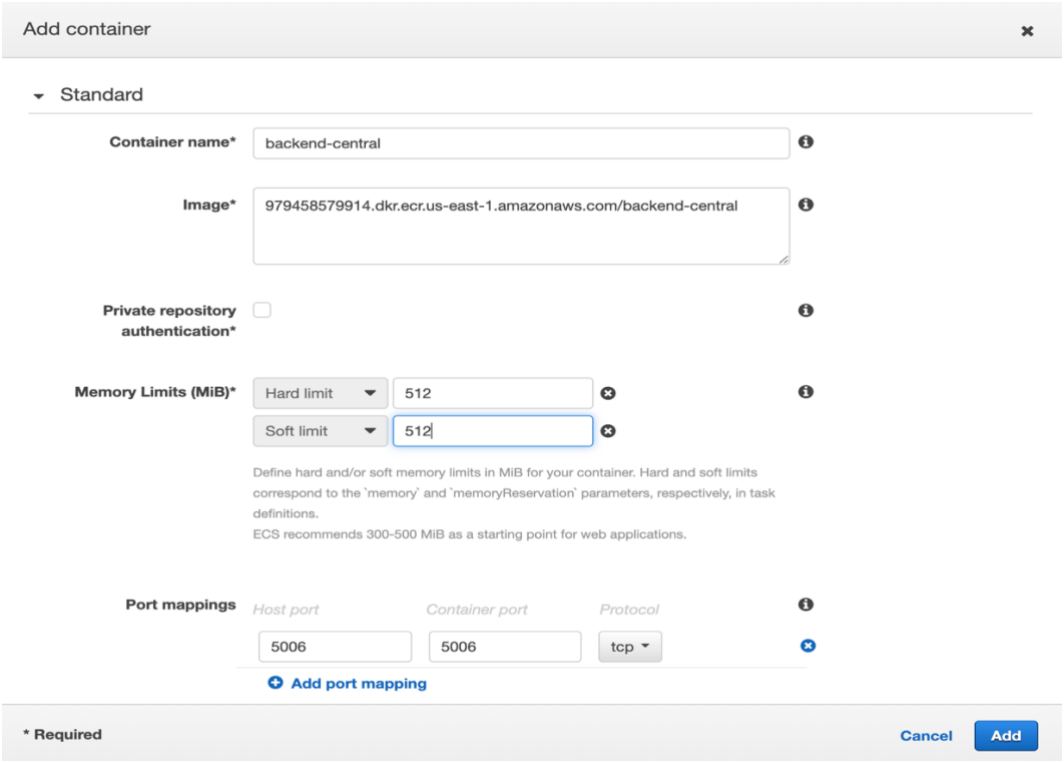


3.2.3. Container Definitions

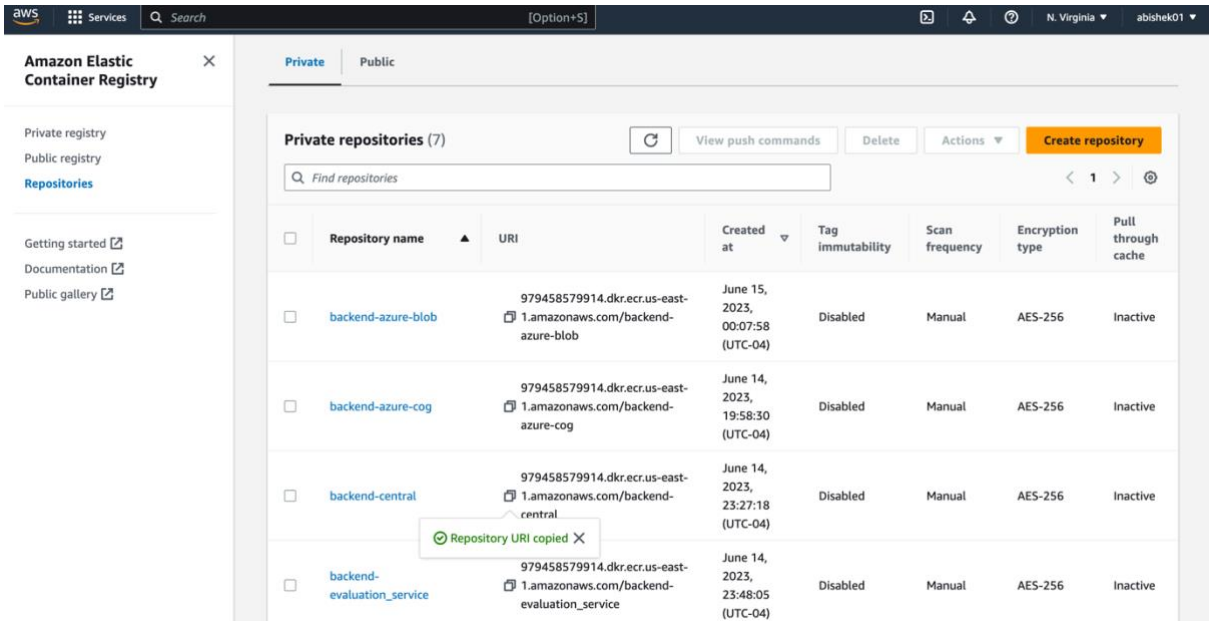
1. Click on Add Container.



2. Use the following container configurations. Note*, we use ports 5006 in our container image exposed in 5006.



3. Please note that in the earlier project stage, we created the image using docker and the image is pushed to ECR(Only for reference for step 2).



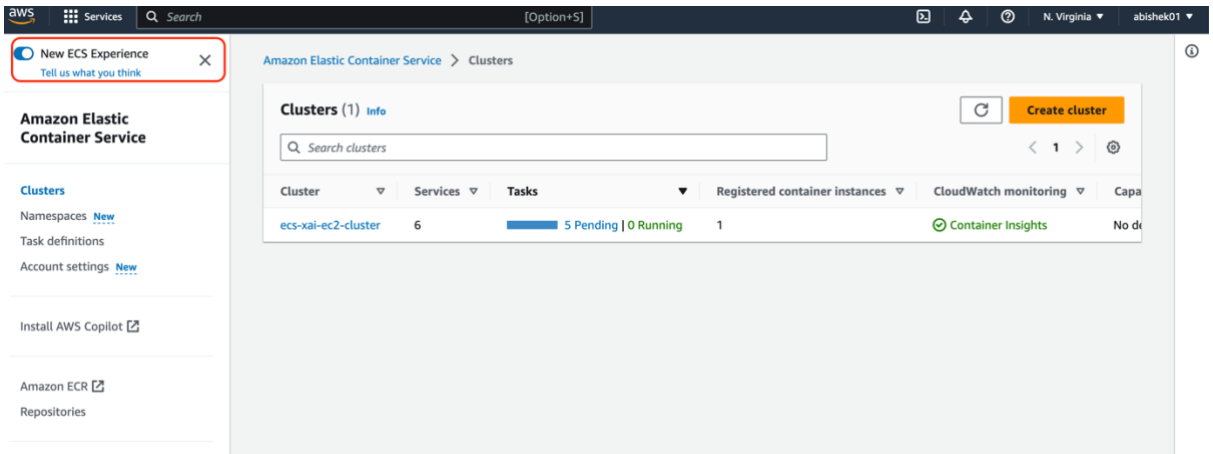
4. Click Create to create a TD.

3.3. Creating a Service.

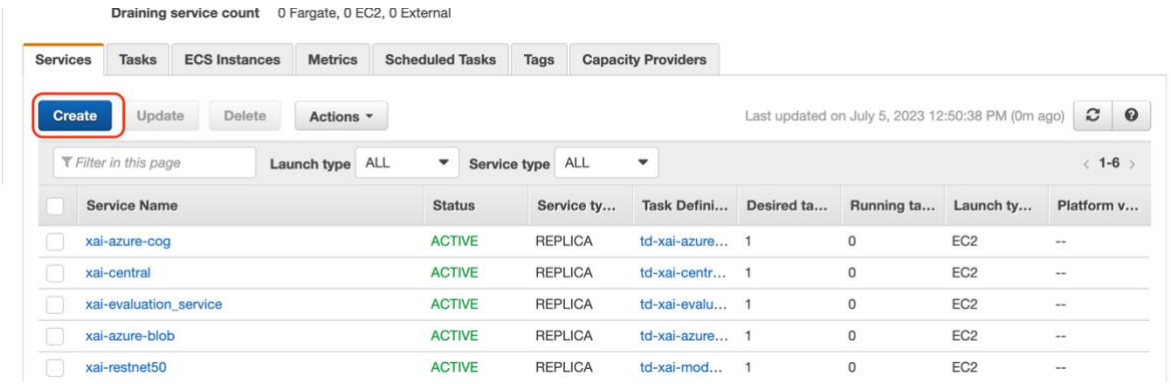
Creating one microservice and follow the same procedure for rest of the microservices.

STEP 1. In AWS Console, go to ECS and select the cluster created in section 3.1.

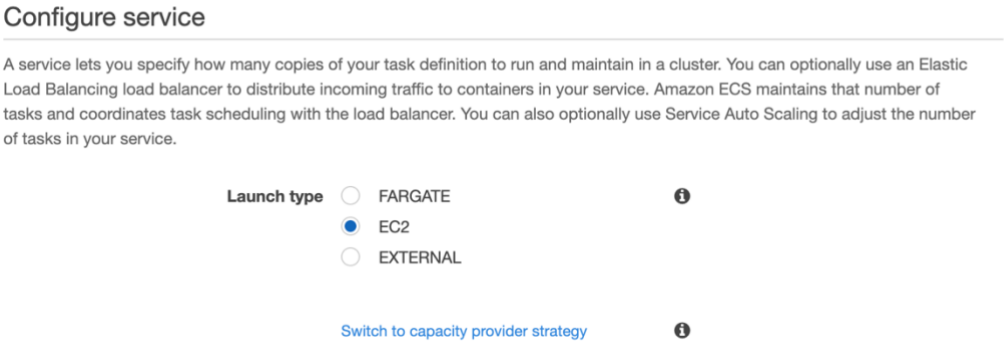
STEP 2. Uncheck the new user experience.



STEP 3. In services tab click create.



STEP 4. Launch type as EC2.

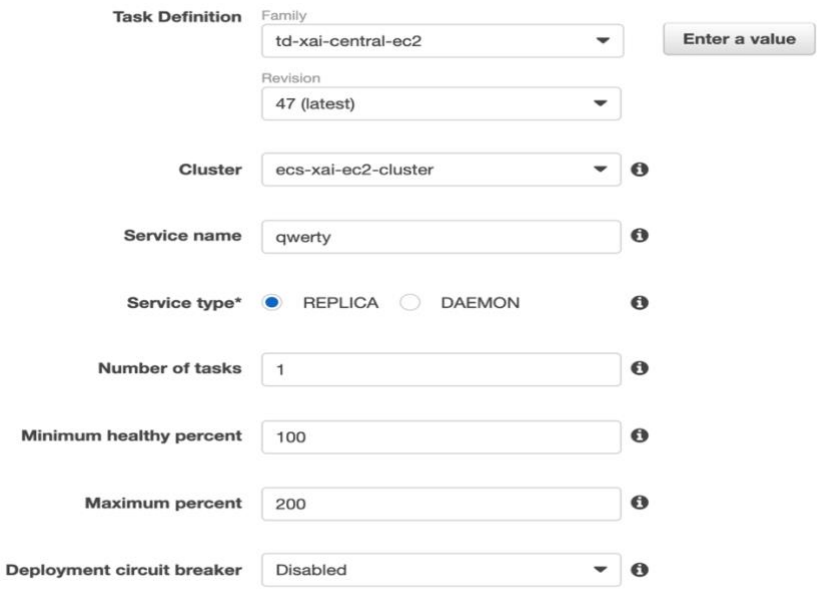


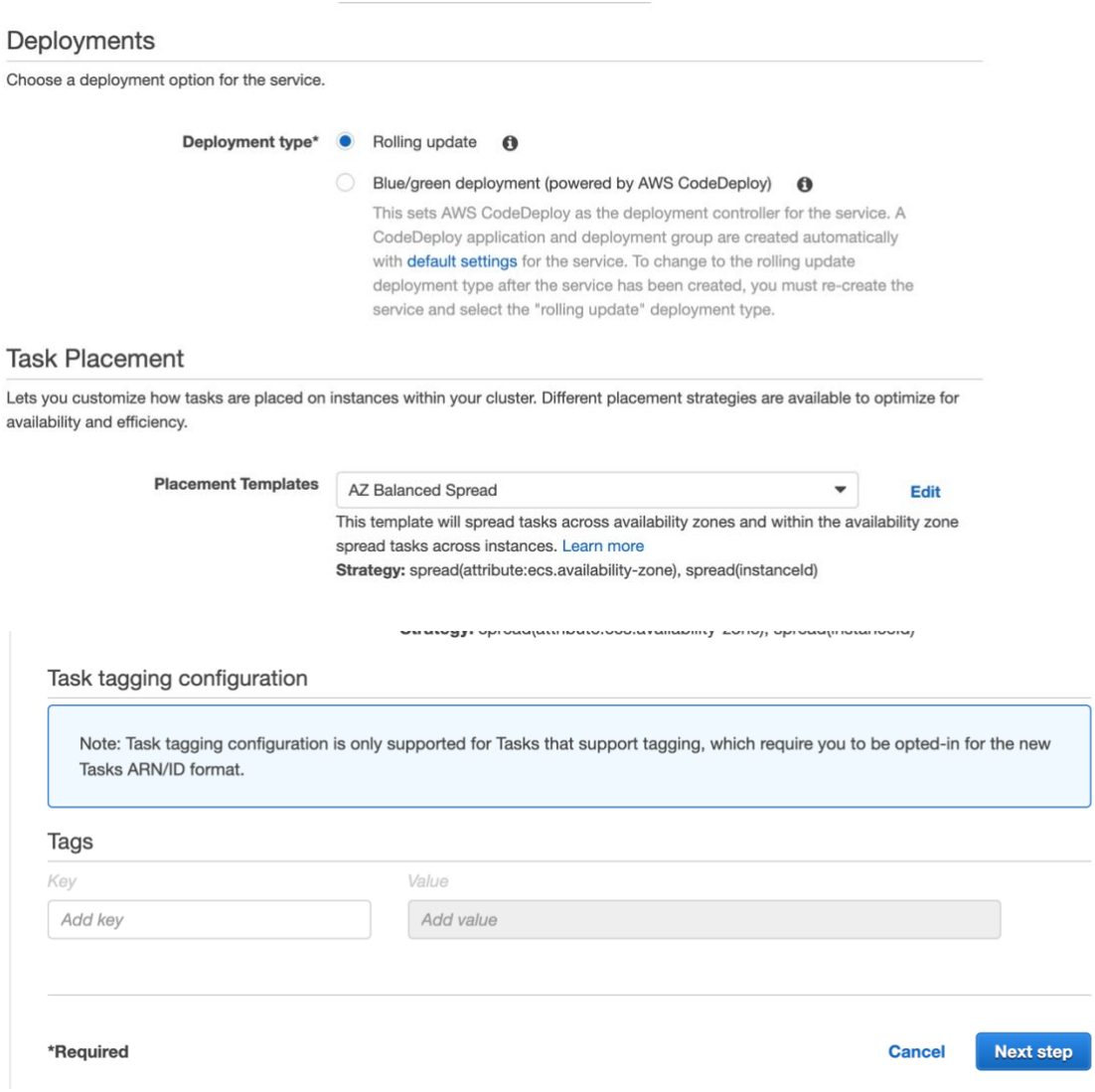
STEP 5. Use the TD created in section 3.2. and use the latest revision.

STEP 6. Use the cluster created in section 3.1.

STEP 7. Service name is user defined.

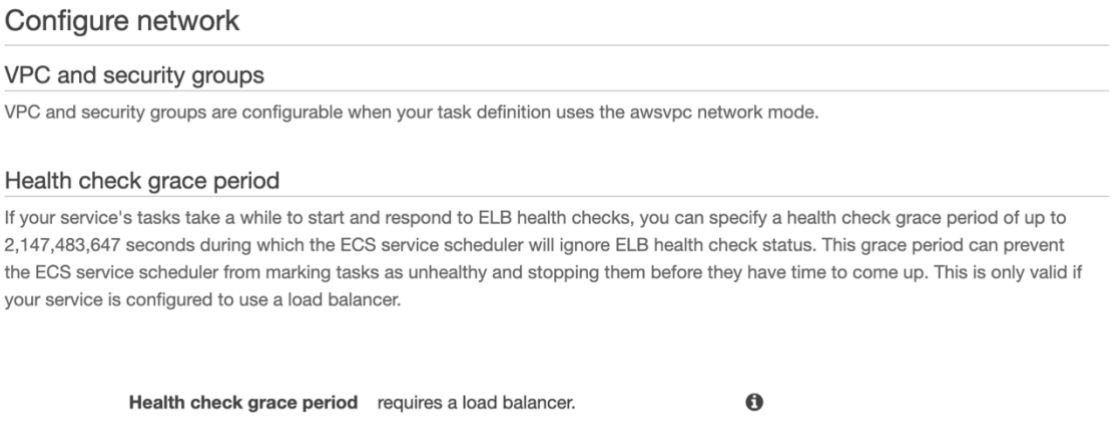
STEP 8. Number of tasks is 1 and rest are defaults.





STEP 9. Click next.

STEP 10. In “Configure Network” all are defaults and proceed to next.



STEP 11. In “Set Auto Scaling” all are defaults and proceed to next.

STEP 12. Create Service.

4. CI-CD Deployment using AWS-Terraform

4.1. Prerequisites:

4.1.1. buildspec.yml:

The following file in the repository, stores the flow of series of commands to execute to deploy our application.

4.1.2. S3 bucket:

Create an S3 bucket to store the terraform state. We need the state to be saved, because on every ci-cd deployment conflict will arise as the state is not saved.

4.1.3. CodeBuild Configuration:

Here we set the location of our buildspec, the values for environment variables and the runtime for performing the build.

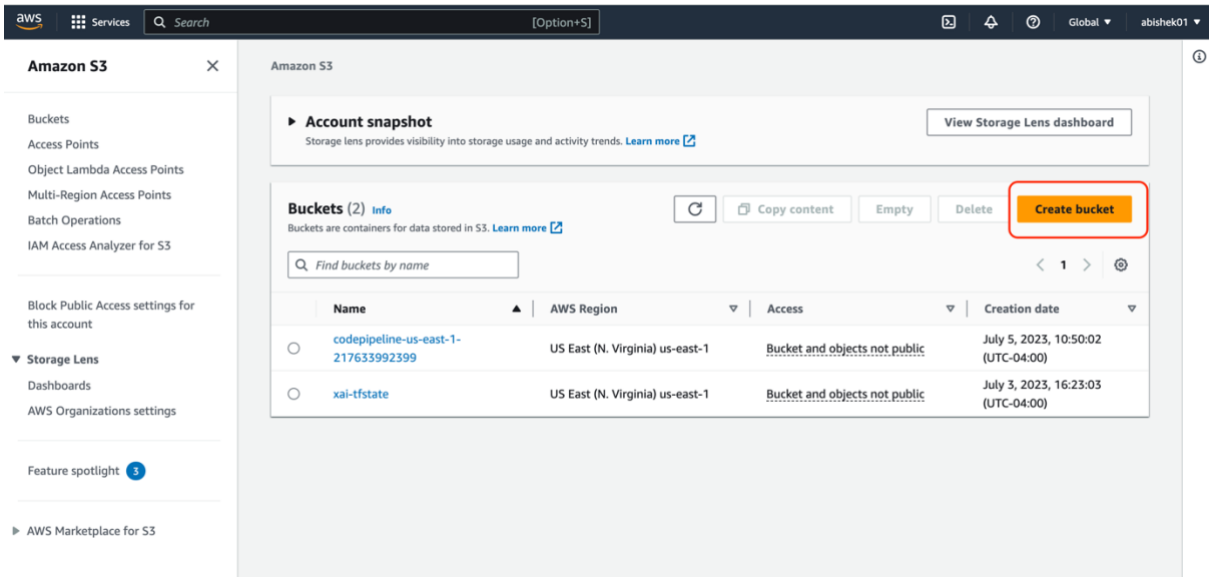
4.1.4. Connection Strings:

We need to save the azure and mongodb connection strings in AWS Systems Manager to access in out buildspec securely.

4.2. S3 Configuration:

STEP 1. In AWS, go to S3.

STEP 2. Create a bucket with name “xai-tfstate”, because out “provider.tf” has configured under this name.



STEP 3. Follow the below screenshots to and click create.

Create bucket Info

Buckets are containers for data stored in S3. Learn more

General configuration

Bucket name

xai-tfstate

Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

AWS Region

US East (N. Virginia) us-east-1

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Choose bucket

Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

- (●) ACLs disabled (recommended)
  All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.
- ( ) ACLs enabled
  Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. Learn more

Bucket Versioning

- (●) Disable
- ( ) Enable

Tags (0) - *optional*

You can use bucket tags to track storage costs and organize buckets. Learn more

No tags associated with this bucket.

Add tag

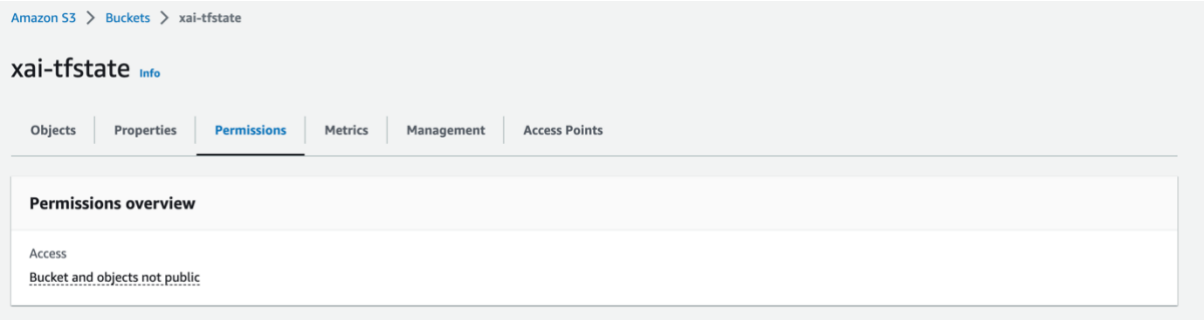
Default encryption Info

Server-side encryption is automatically applied to new objects stored in this bucket.

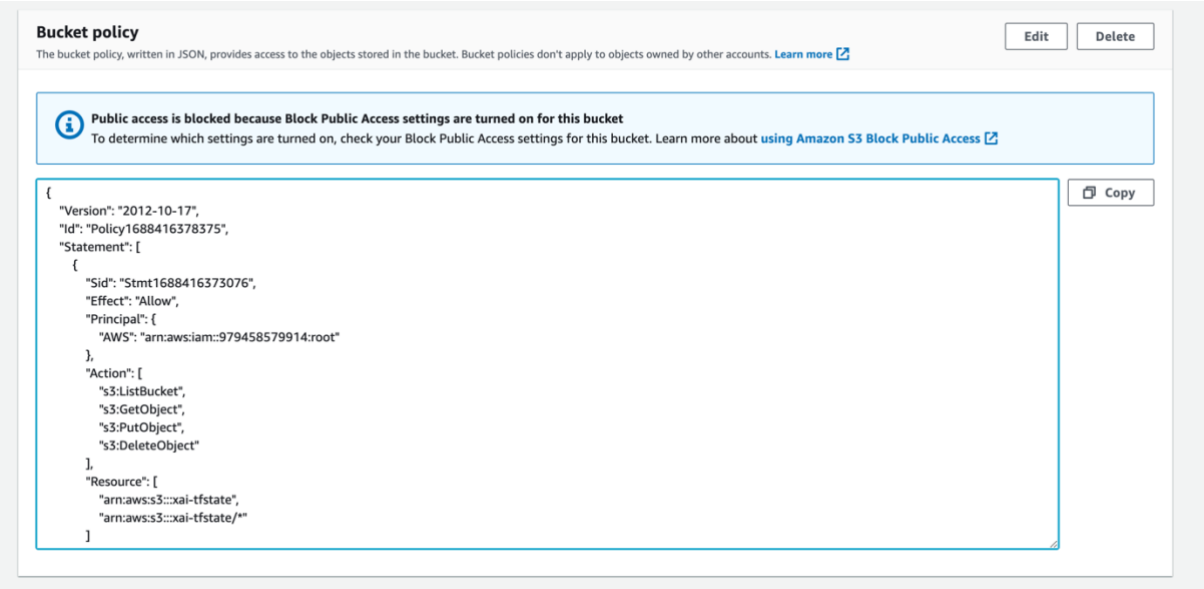
Encryption type Info

- (●) Server-side encryption with Amazon S3 managed keys (SSE-S3)
- ( ) Server-side encryption with AWS Key Management Service keys (SSE-KMS)
- ( ) Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)

STEP 4. Click the bucket created and go to permissions tab.



STEP 5. Under bucket policy add the following json policy.



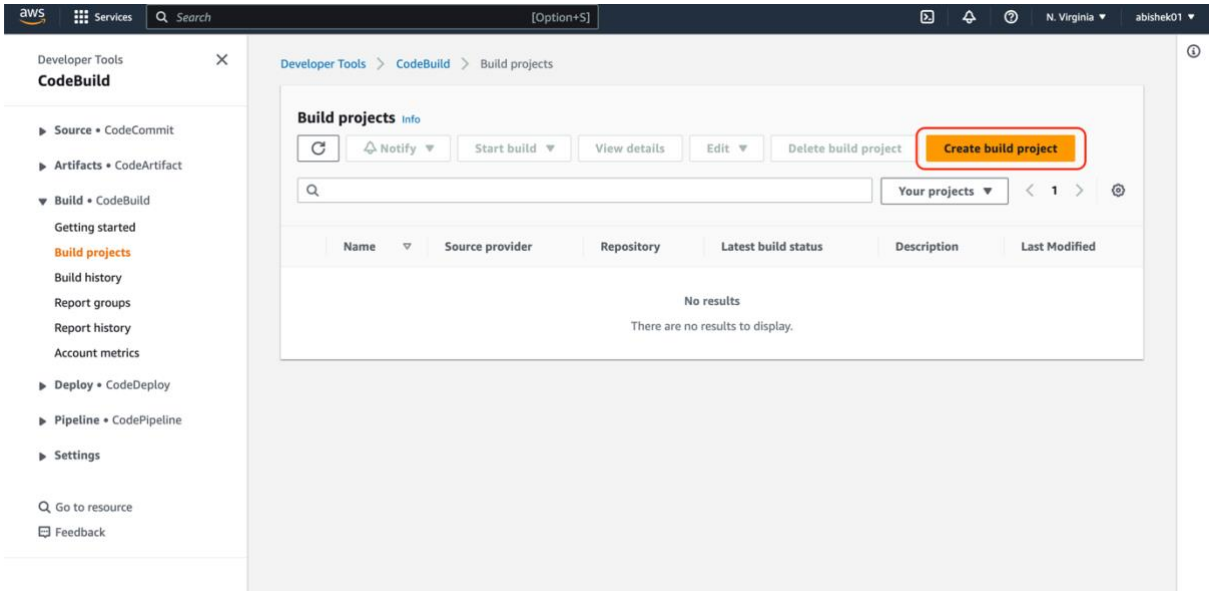

```
        {
"Version": "2012-10-17",
"Id": "Policy1688416378375",
"Statement": [
    {
        "Sid": "Stmt1688416373076",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::<account-id>:root"
        },
        "Action": [
            "s3:ListBucket",
            "s3:GetObject",
            "s3:PutObject",
            "s3:DeleteObject"
        ],
```

```
        "Resource": [
            "arn:aws:s3:::xai-tfstate",
            "arn:aws:s3:::xai-tfstate/*"
        ]
    }
  ]
}
```

4.3. CodeBuild Configuration:

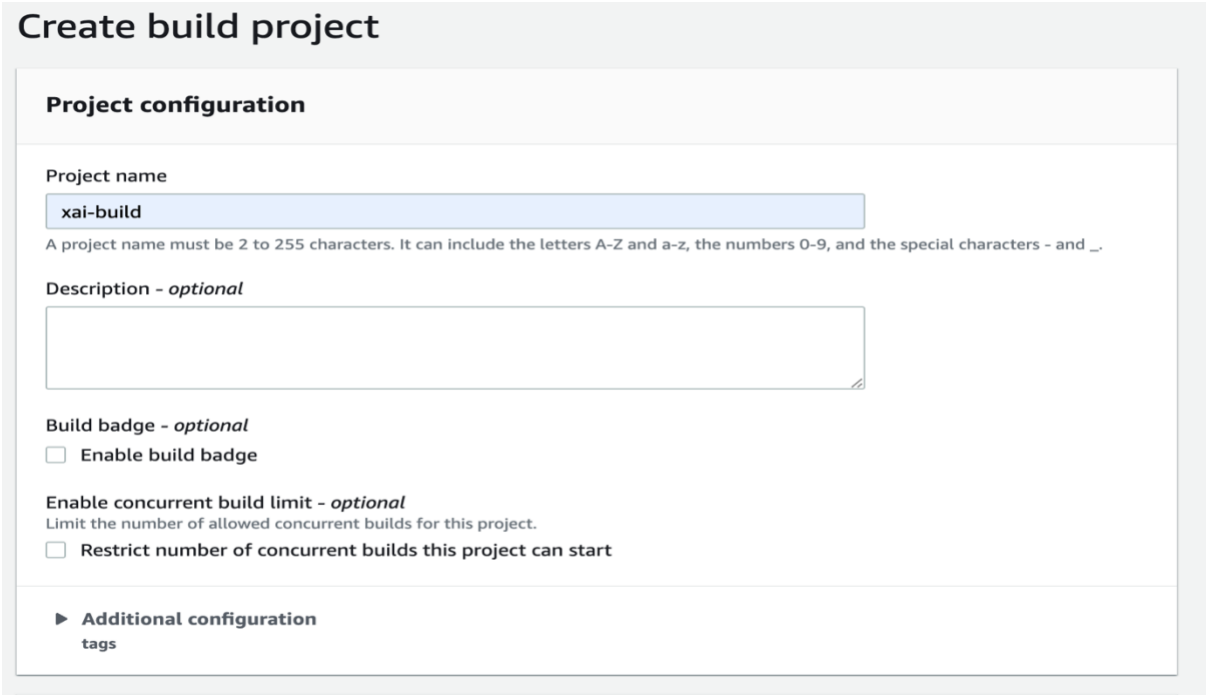
STEP 1. In AWS console, go to CodeBuild.

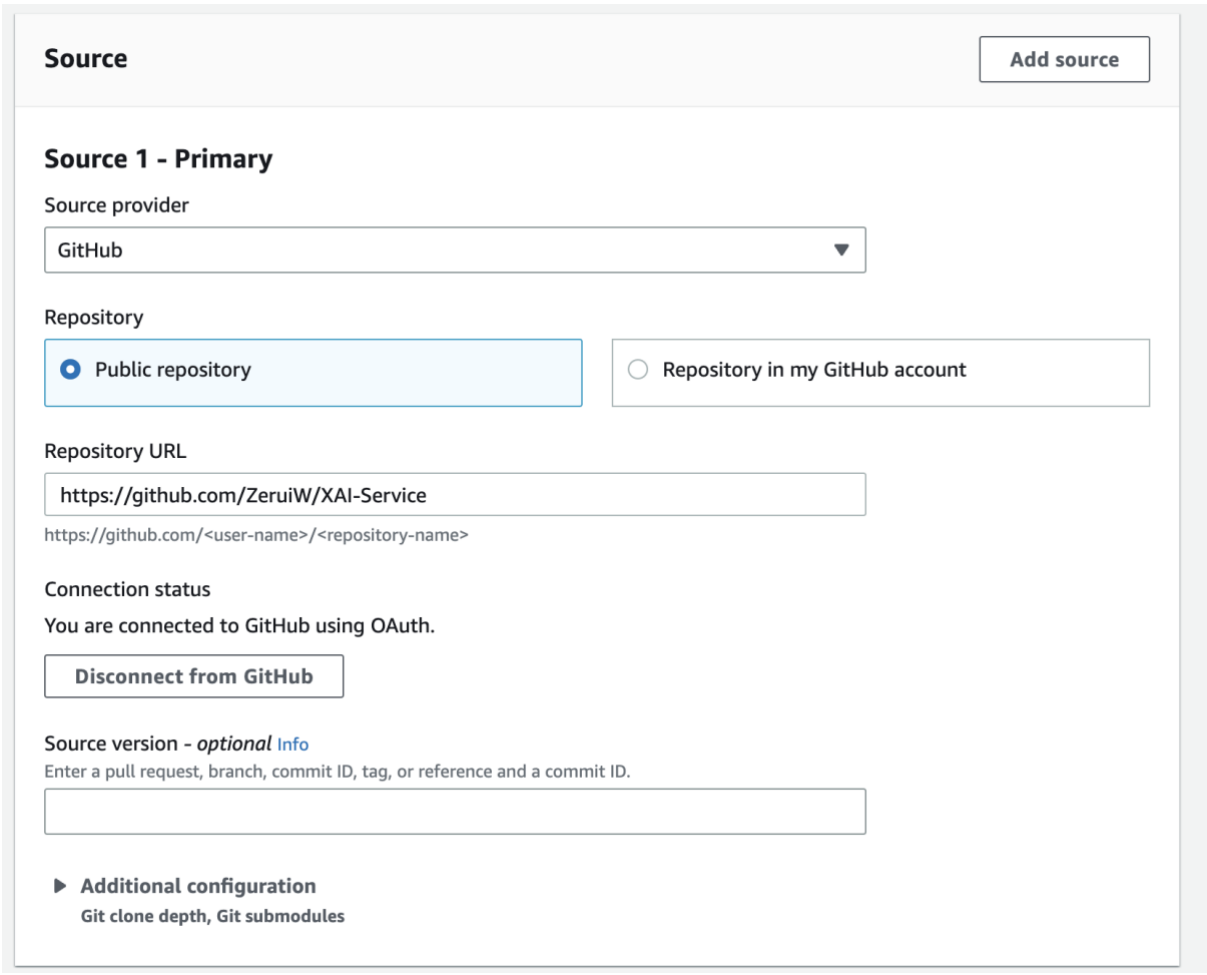
STEP 2. Click create build project.

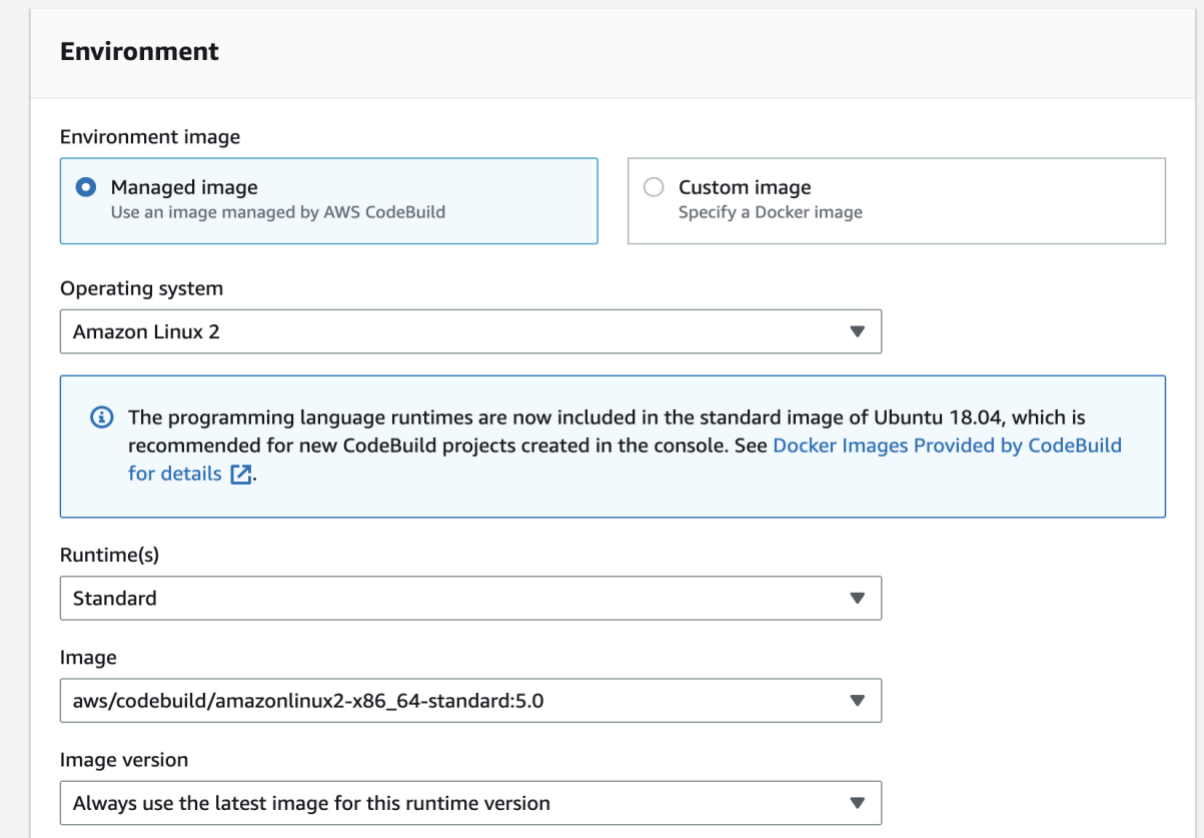


STEP 3. Give the project name as “xai-build”.

STEP 4. Follow the steps shown in below images.


Create build project
Project configuration
Project name
xai-build
A project name must be 2 to 255 characters. It can include the letters A-Z and a-z, the numbers 0-9, and the special characters - and _.
Description - optional
Build badge - optional
Enable build badge
Enable concurrent build limit - optional
Limit the number of allowed concurrent builds for this project.
Restrict number of concurrent builds this project can start
Additional configuration
tags


Source
Add source
Source 1 - Primary
Source provider
GitHub
Repository
Public repository
Repository in my GitHub account
Repository URL
https://github.com/ZeruiW/XAI-Service
https://github.com/<user-name>/<repository-name>
Connection status
You are connected to GitHub using OAuth.
Disconnect from GitHub
Source version - optional Info
Enter a pull request, branch, commit ID, tag, or reference and a commit ID.
Additional configuration
Git clone depth, Git submodules


Environment
Environment image
Managed image
Use an image managed by AWS CodeBuild
Custom image
Specify a Docker image
Operating system
Amazon Linux 2
The programming language runtimes are now included in the standard image of Ubuntu 18.04, which is recommended for new CodeBuild projects created in the console. See Docker Images Provided by CodeBuild for details.
Runtime(s)
Standard
Image
aws/codebuild/amazonlinux2-x86_64-standard:5.0
Image version
Always use the latest image for this runtime version

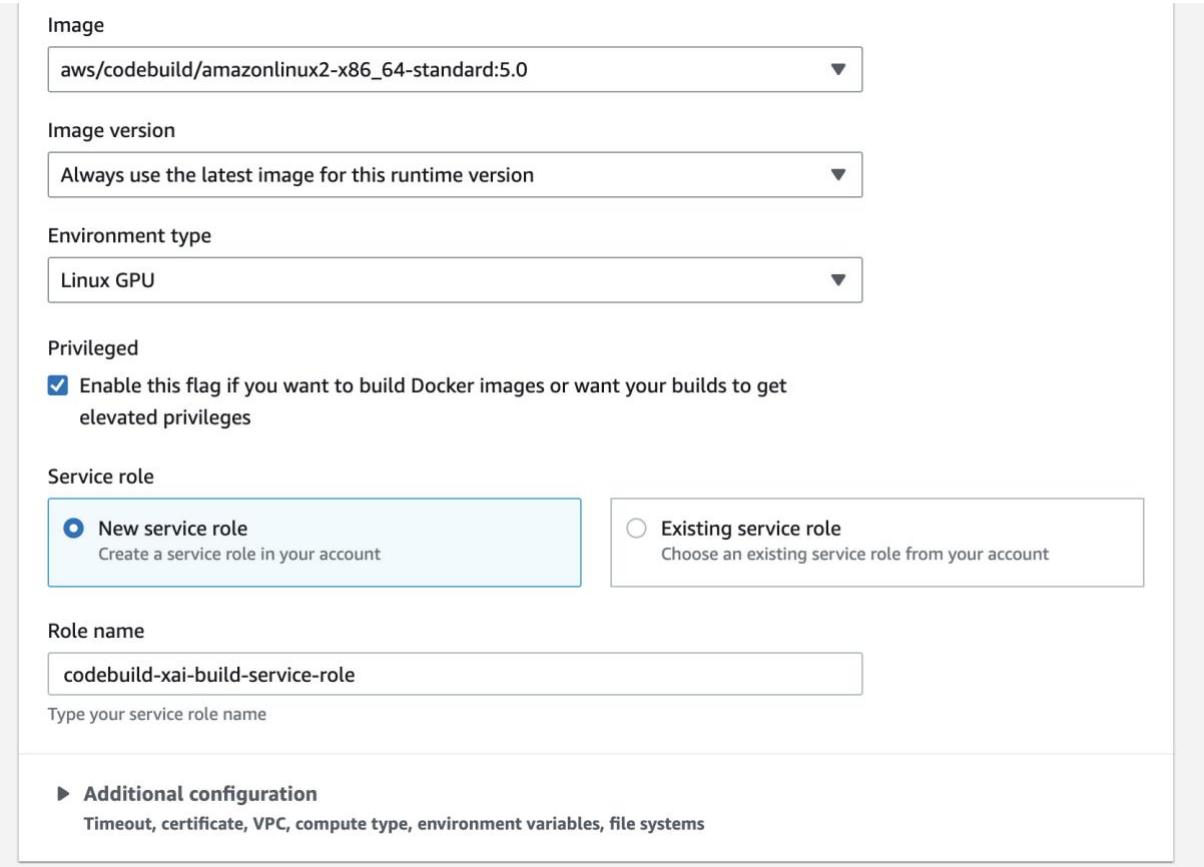
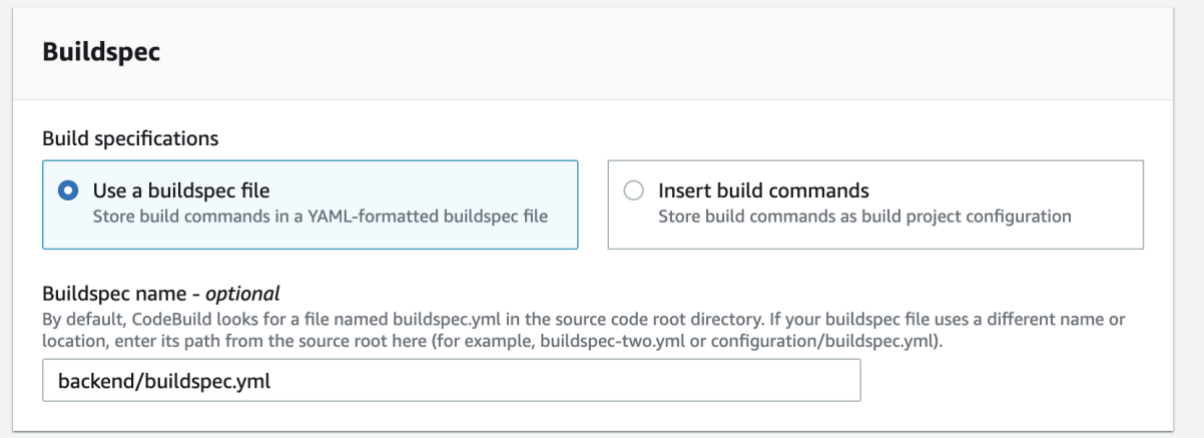
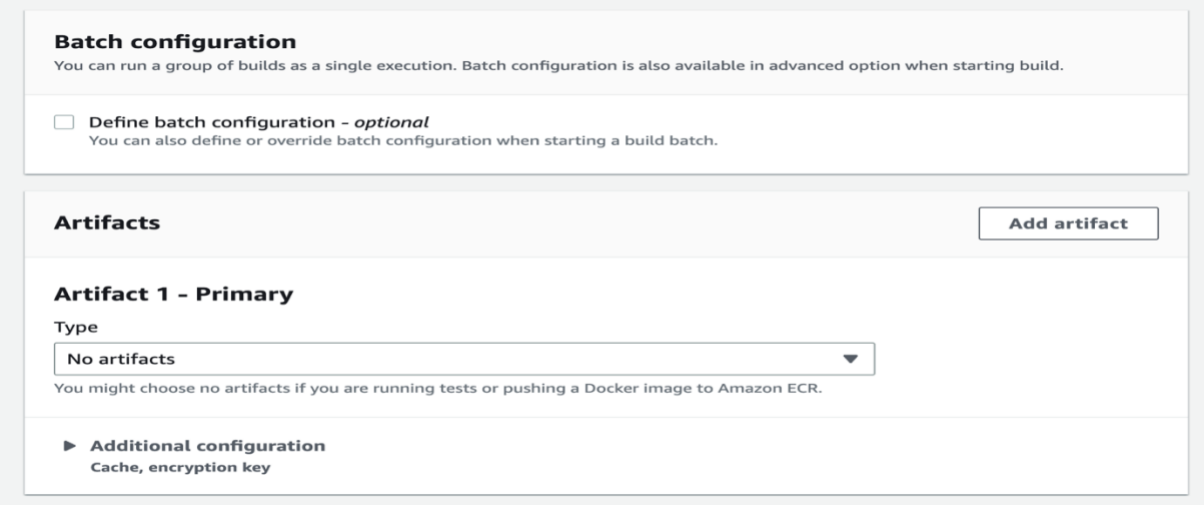
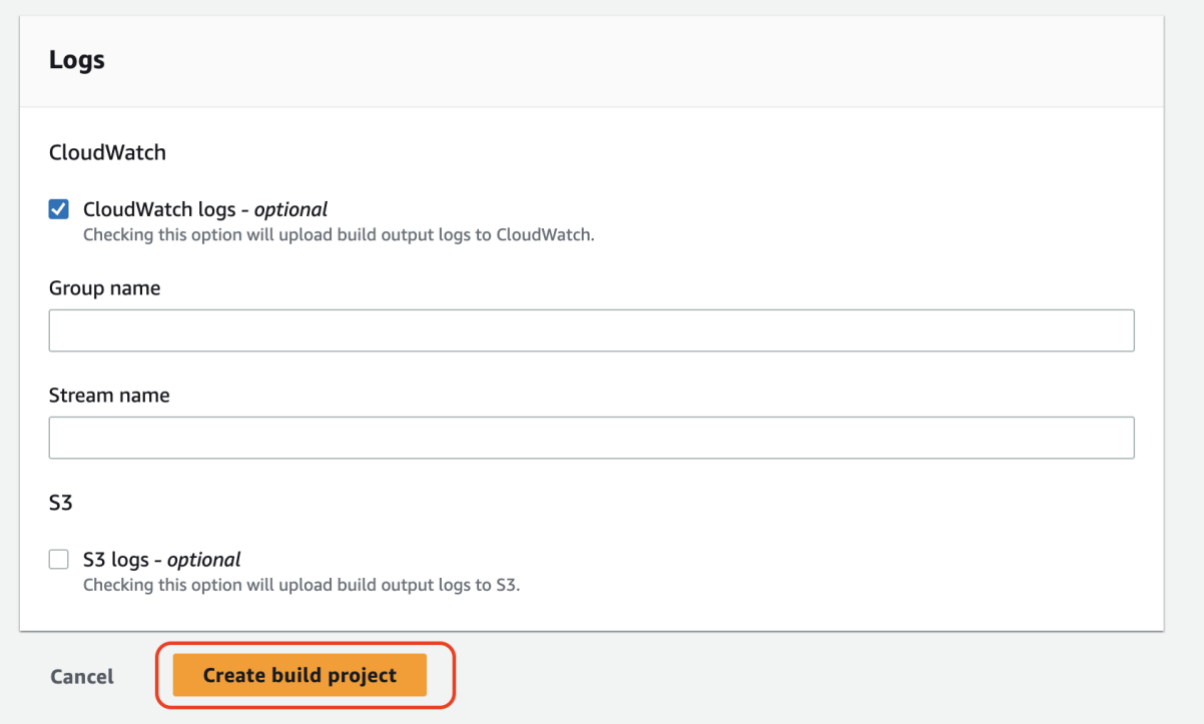


Image
aws/codebuild/amazonlinux2-x86_64-standard:5.0
Image version
Always use the latest image for this runtime version
Environment type
Linux GPU
Privileged
Enable this flag if you want to build Docker images or want your builds to get elevated privileges
Service role
New service role
Create a service role in your account
Existing service role
Choose an existing service role from your account
Role name
codebuild-xai-build-service-role
Type your service role name
Additional configuration
Timeout, certificate, VPC, compute type, environment variables, file systems







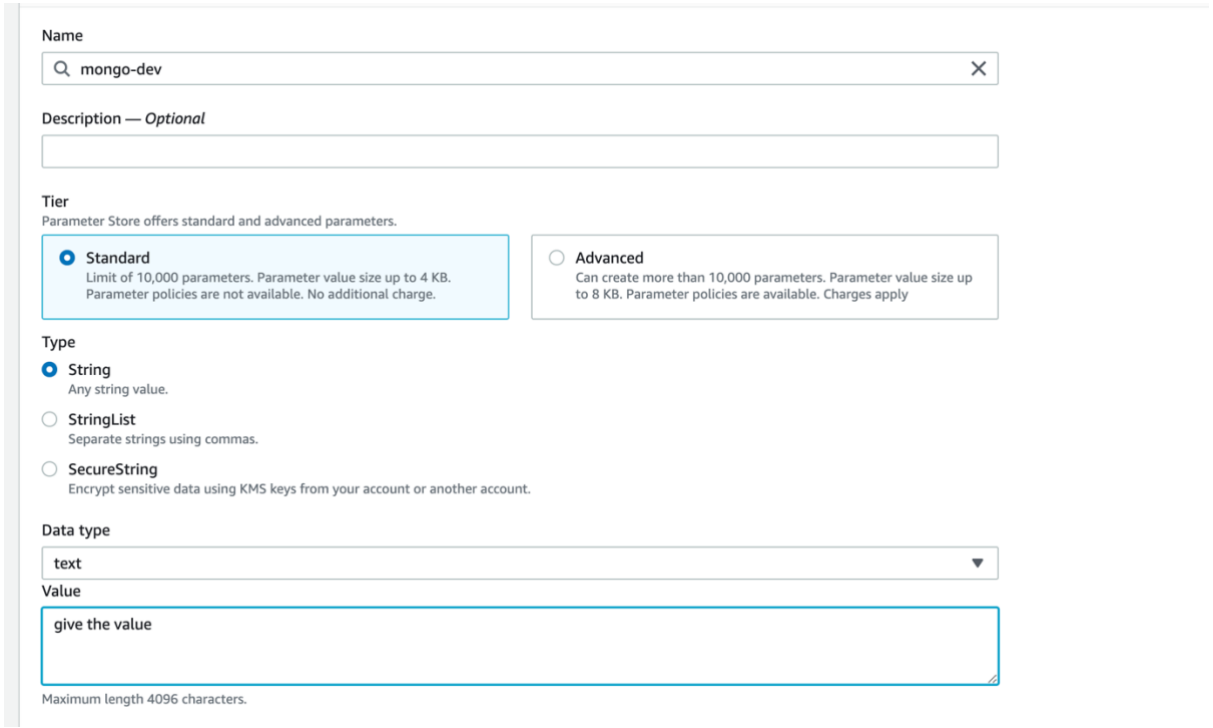
STEP 5. Click create.

4.3. Connection String Configuration:

STEP 1. In AWS Console, go to “Systems Manager”.

STEP 2. Go to parameter Store.

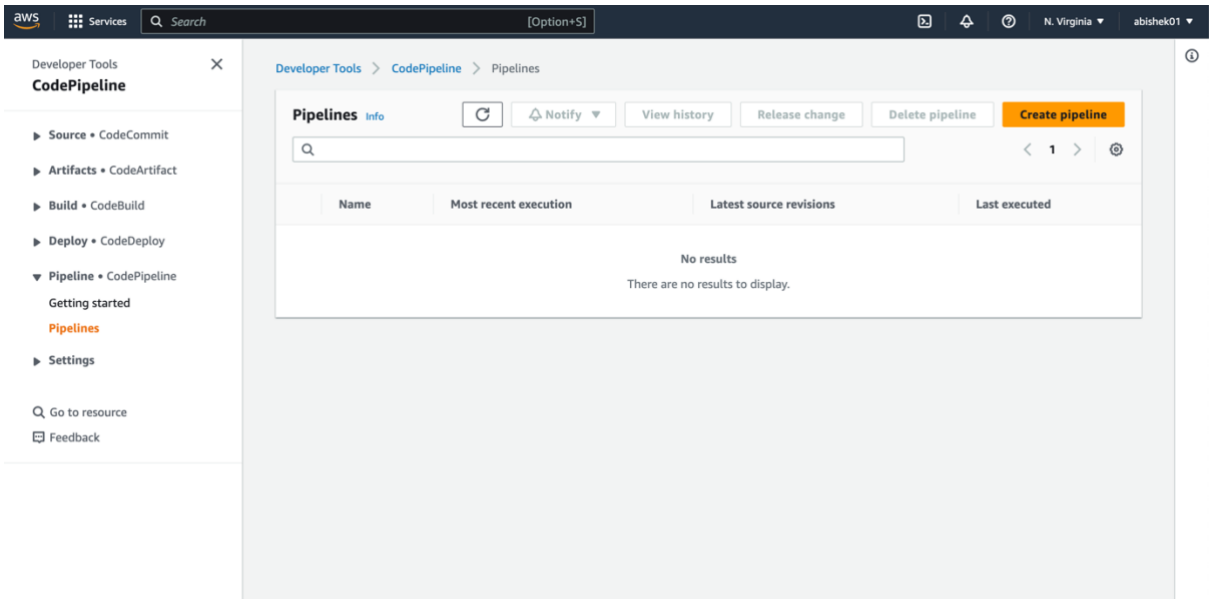
STEP 3. Use the name of the file as “azure-con-str” and “mongo-dev” as we have configured the name in buildspec.



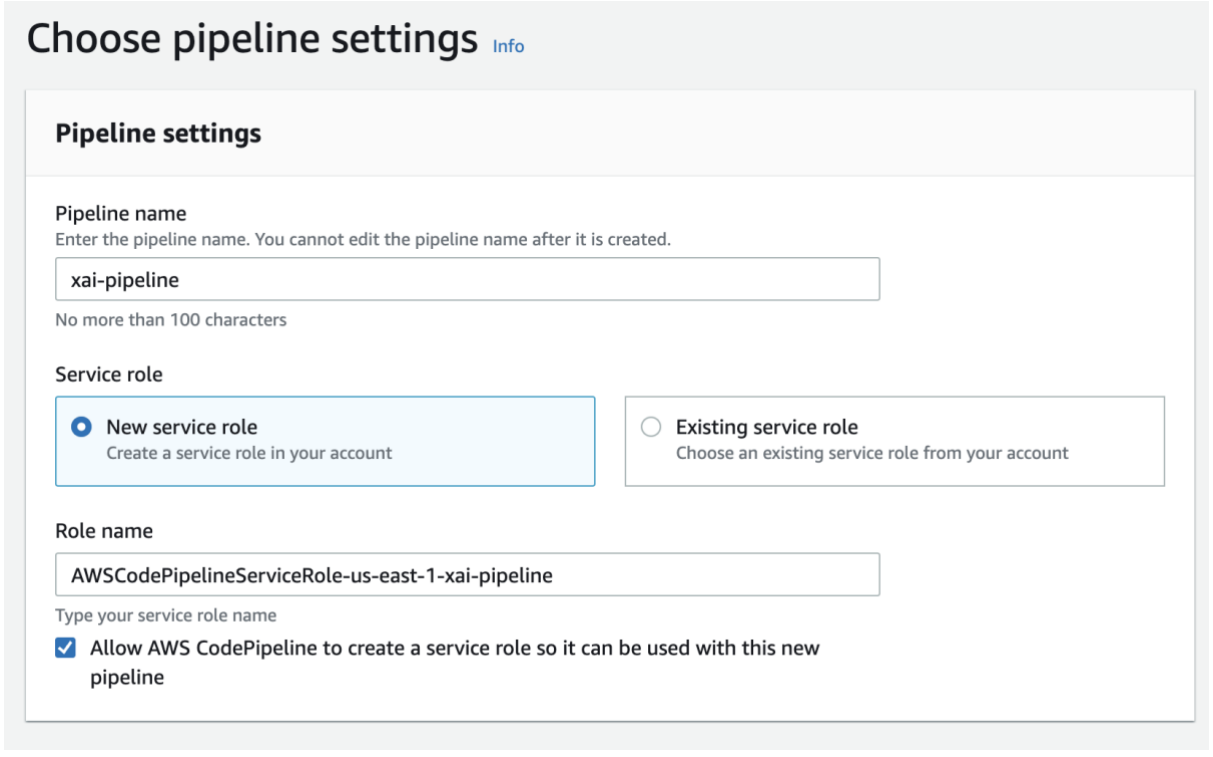
4.4. CodePipeline Configuration:

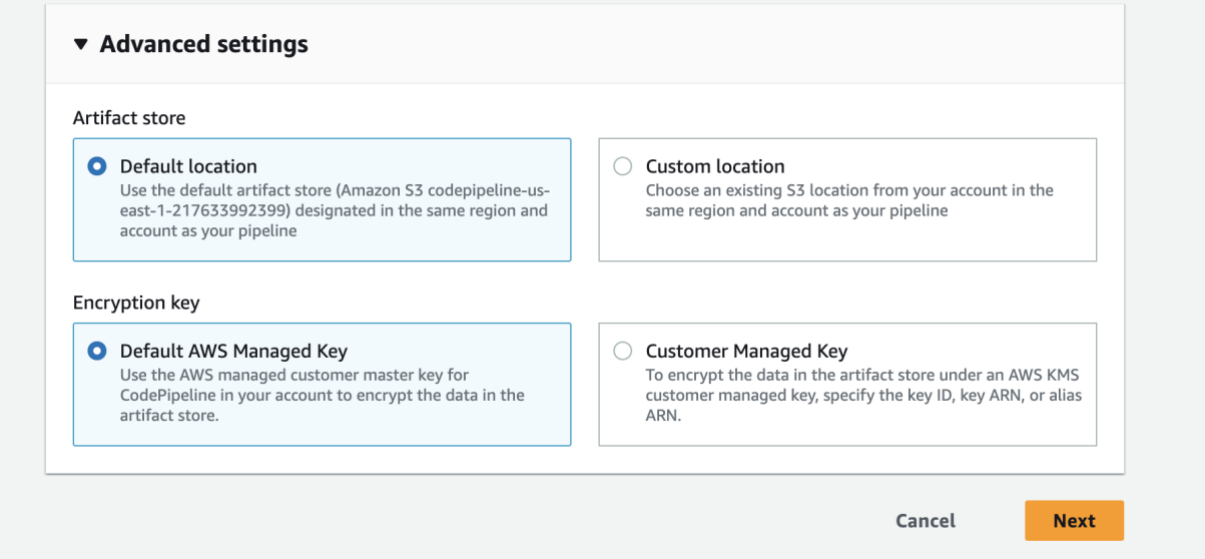
STEP 1. In AWS Console, go to CodePipeline.

STEP 2. Click Create Pipeline.

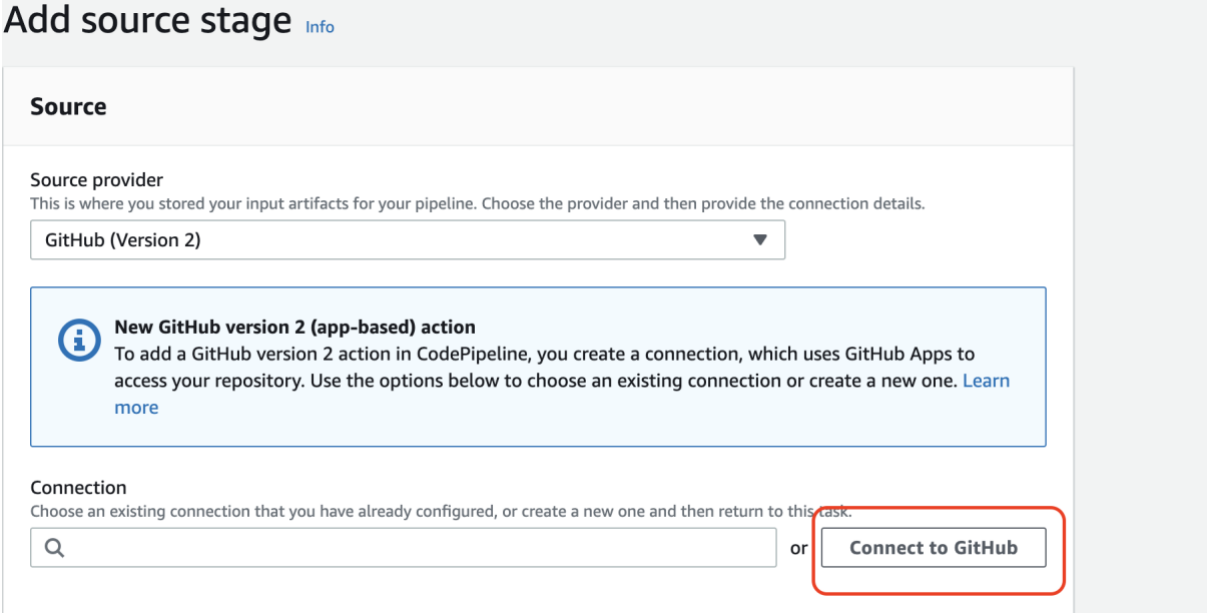


STEP 3. Give the pipeline name as “xai-pipeline”, rest as default and click next.



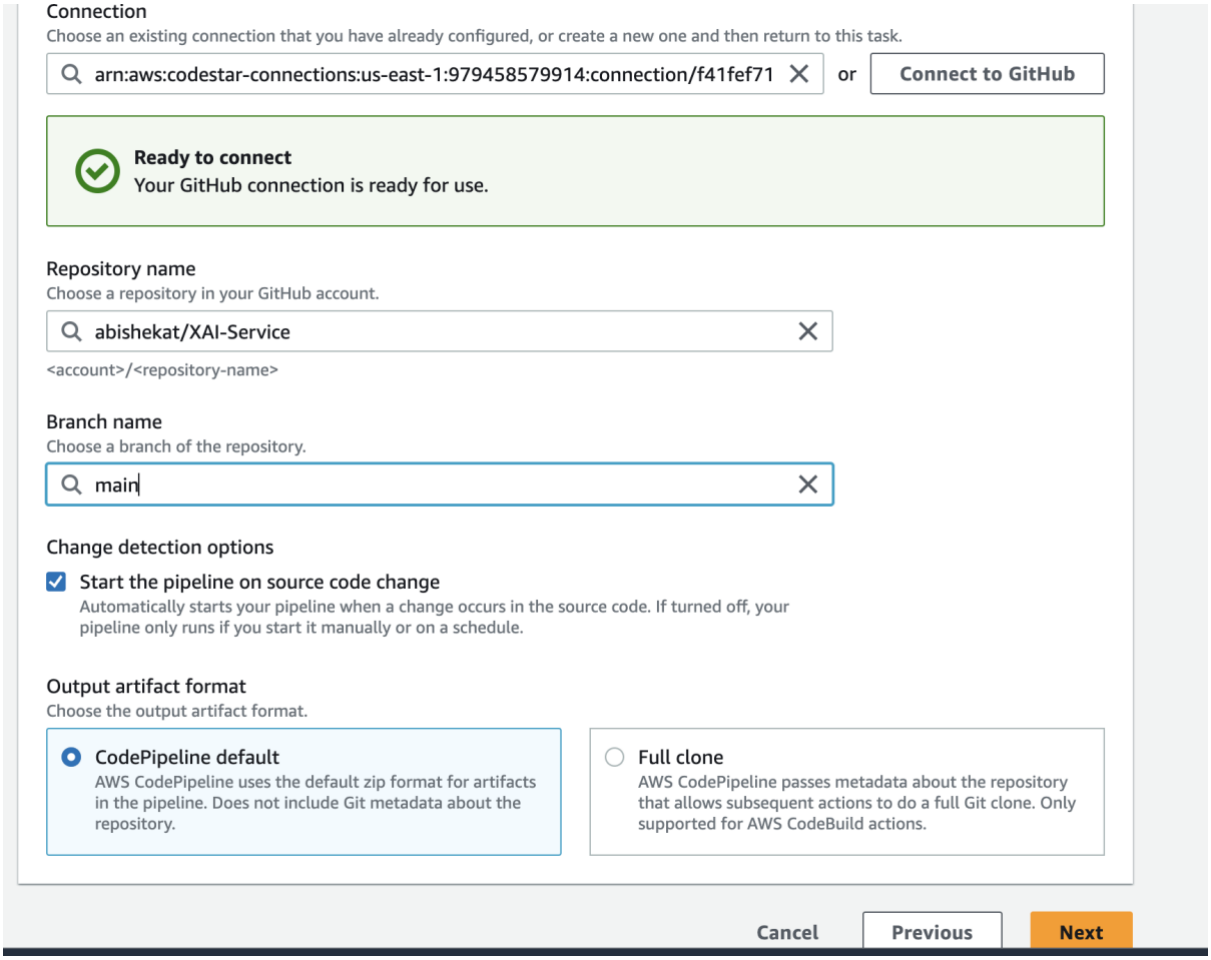


STEP 4. Source provider as GitHub (Version 2).

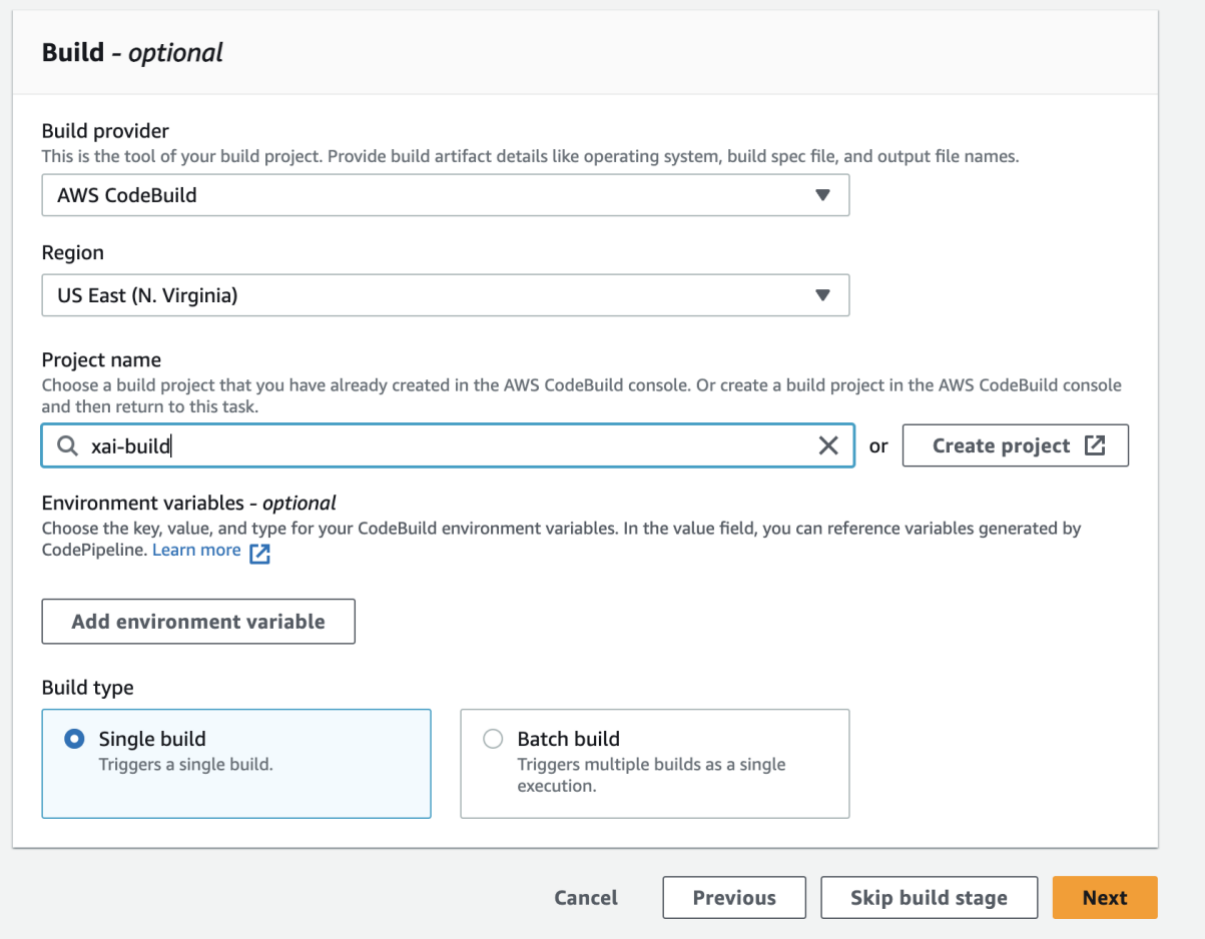


STEP 5. Connect to Github, give your github username when a window popup.

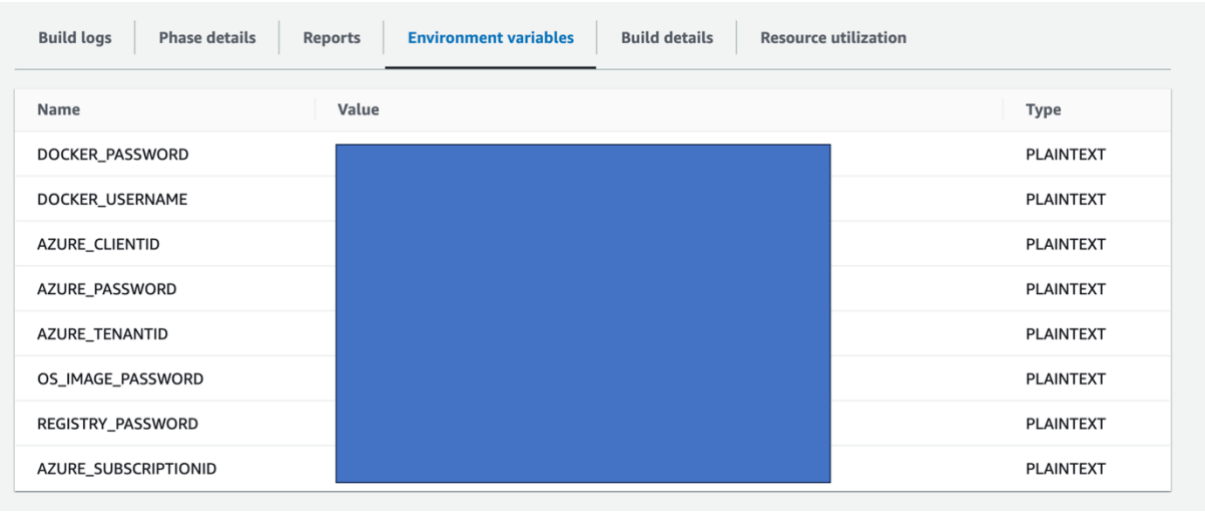
STEP 6. Follow the image for rest of the fields and click next.



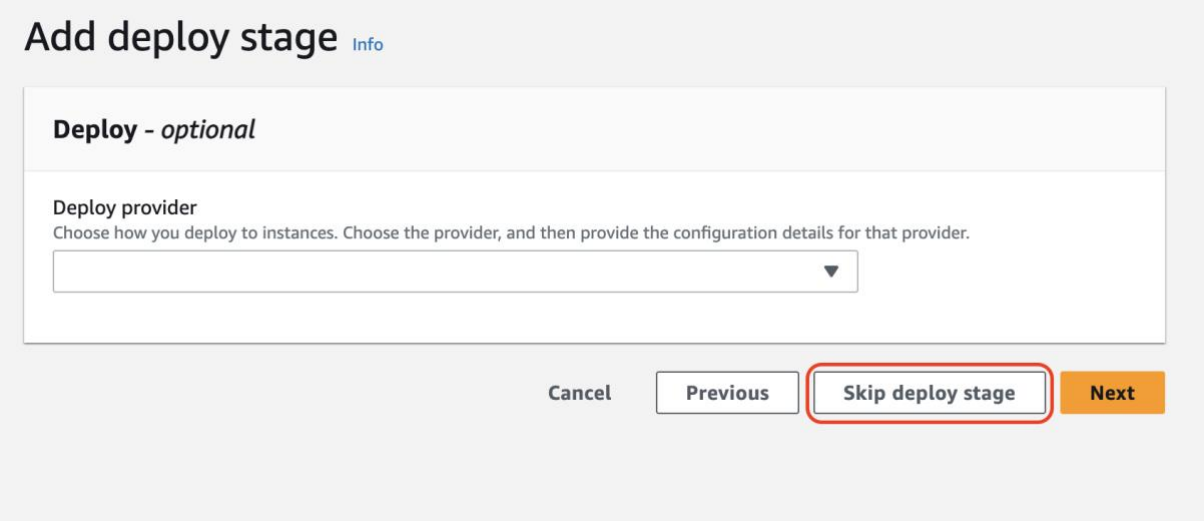
STEP 7. Build provider as AWS CodeBuild and give the project name as “xai-build” created in section 4.2. Click on Next.



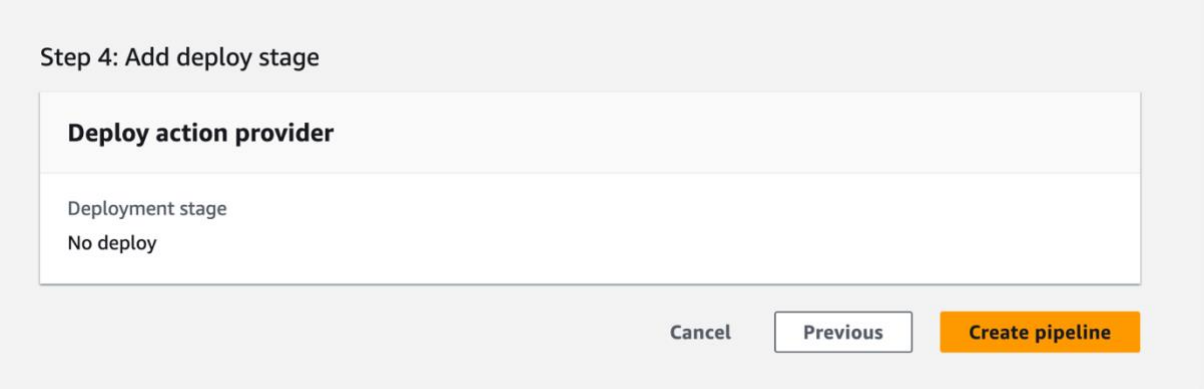
STEP 8. Add environment variables.



STEP 9. Skip the deploy stage.



STEP 10. Click create pipeline.



4.5. *Resource clean-up:

Once you have successfully deployed using terraform and want to remove the resources created. Follow the below steps.

STEP 1. Go to the local setup of your xai-service.
STEP 2. Open terminal and “cd backend/terraform”.
STEP 3. Use the command “terraform init”. It copies the terraform state file from s3.
STEP 4. Use the command “terraform destroy” and type yes on prompt.
STEP 5. All the resources created by CodeBuild will be destroyed.