Table of Contents

Introduction

This document provides a step-by-step guide to set up the Sample App on your AWS account. It assumes no prior configuration and is designed to help you get started from scratch. By following the steps, you will be able to have the Sample App on your AWS account and a link for people to access it.

Prerequisites

Before proceeding with the setup, ensure you have the following:

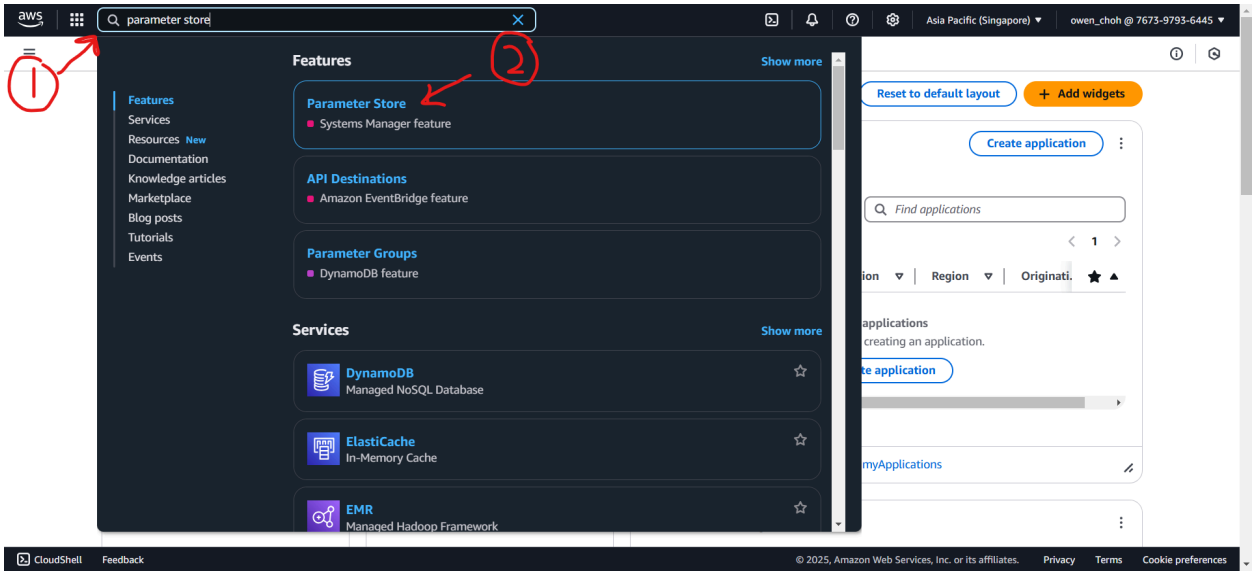
1. **AWS Account**:
   An active AWS account with the necessary permissions to create and manage the following resources:
   - EC2 instances
   - Parameter Store parameters (via AWS Systems Manager)
   - IAM roles
2. **Sample App Source Code**:
   The source code of the Sample App, including its dependencies, available either locally or in a repository.
   - In this guide, we will use the SSG GitHub repository (https://github.com/ssg-wsg/Sample-Codes) as an example.
3. **Sample App Secrets**:
   The secrets used by the Sample App to make the API calls on behalf of our users. There are a total of **3** items:
   - Encryption key
   - Certificate
   - Private Key

Overview of the Setup Process

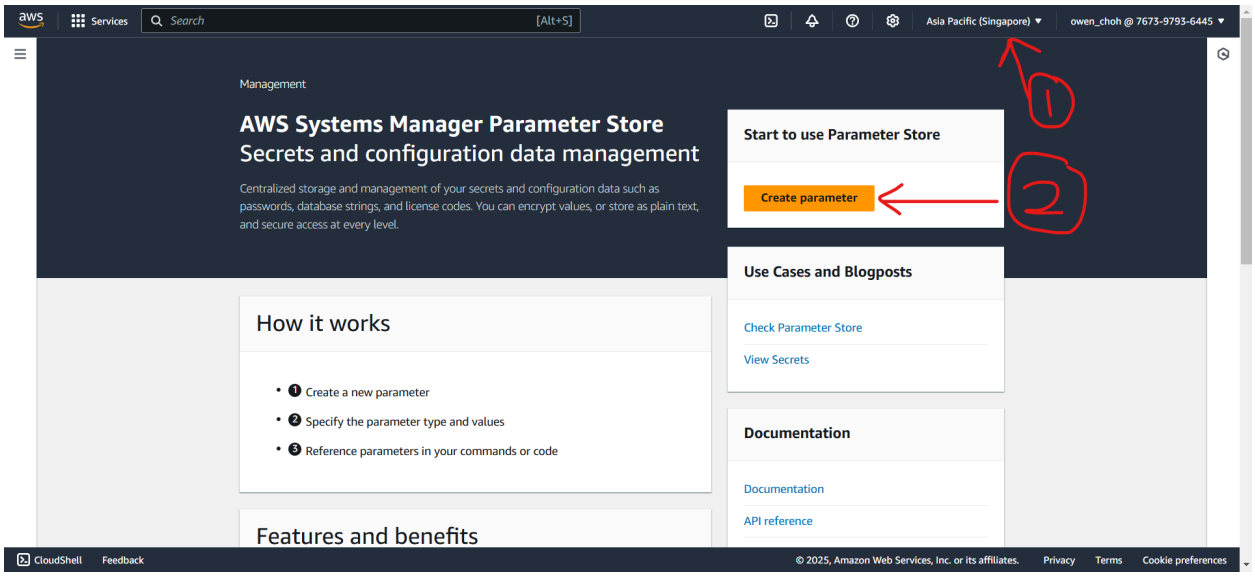
1. **Parameter Store Configuration**: Store the necessary secrets and configuration settings.
2. **IAM Role Setup**: Create an IAM role and attach policies for accessing the parameter store.
3. **EC2 Instance Launch Configuration**: Launch and configure an EC2 instance to host the Sample App.
4. **EC2 Instance installation**: Install dependencies on the EC2 instance.
5. **Sample App Deployment**: Deploy the app on the EC2 instance.
6. **Additional Notes**: Additional information after setting up the instance.

Step 1: Parameter Store Configuration

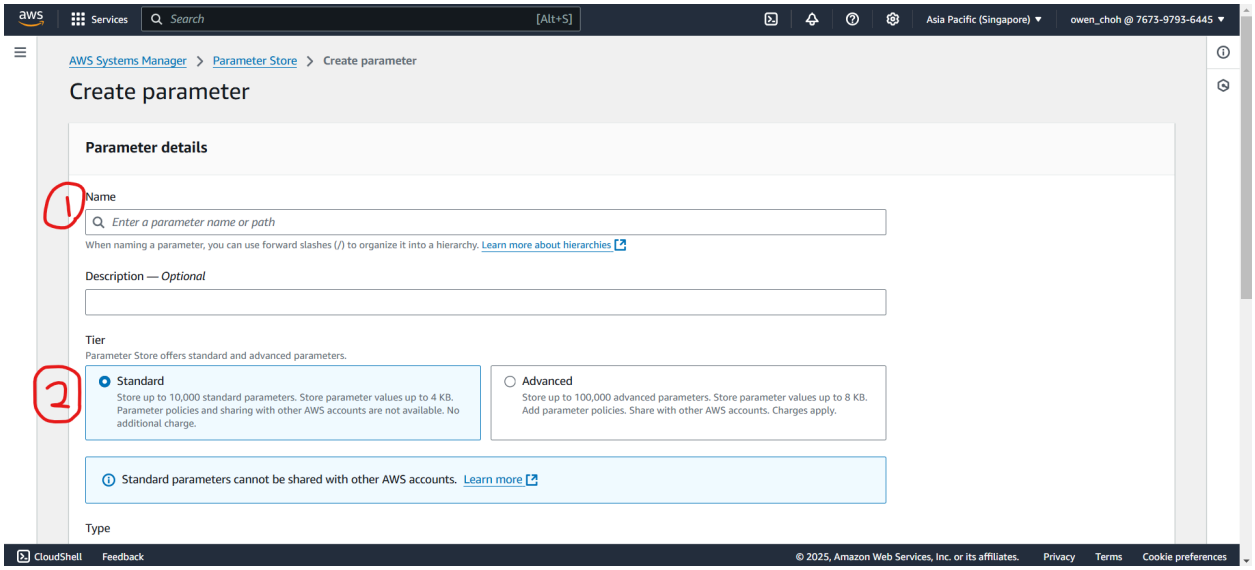
1. From the AWS homepage, navigate to the AWS Parameter Store.



2. On the **parameter store** page, make sure you are in the correct **region** and click **Create parameter**.

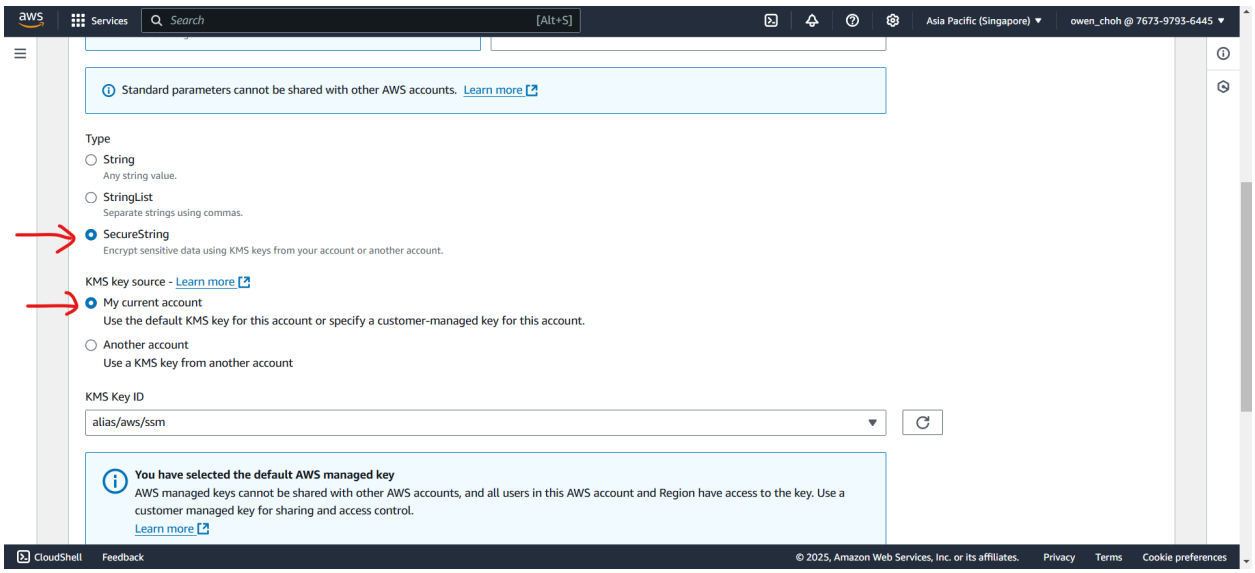


3. For each secret you will need to put a Name and select “**Standard**” tier.

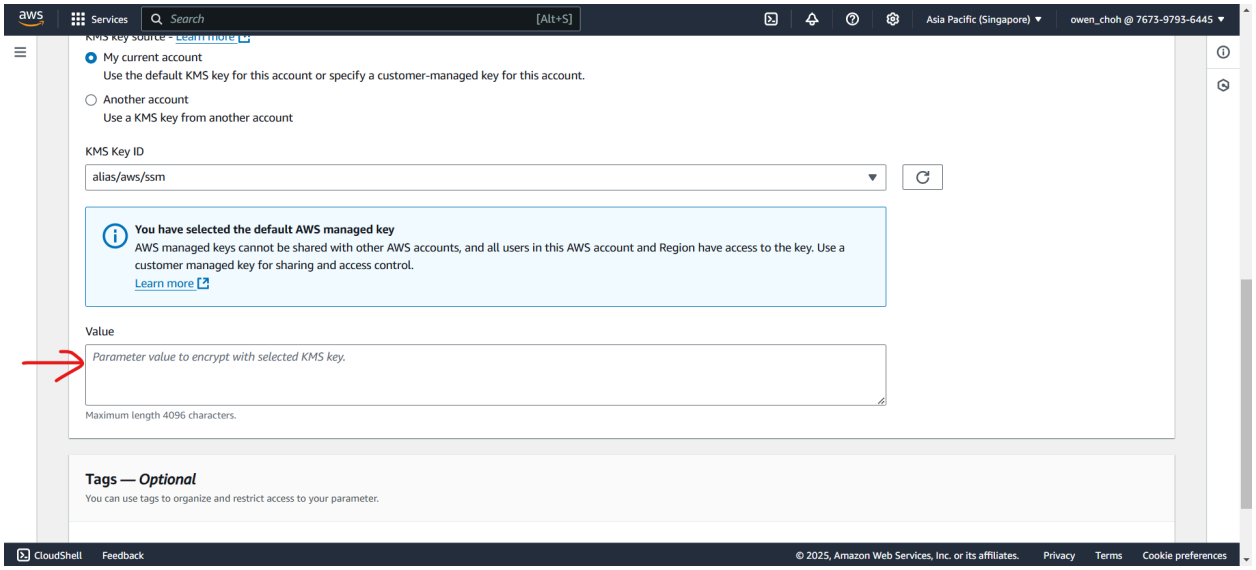


- The names for each of the secret in this example will be as follows:
- Encryption key - name: **/SampleApp/test/secrets/encrypt**
- Certificate - name: **/SampleApp/test/secrets/cert**
- Private Key - name: **/SampleApp/test/secrets/key**

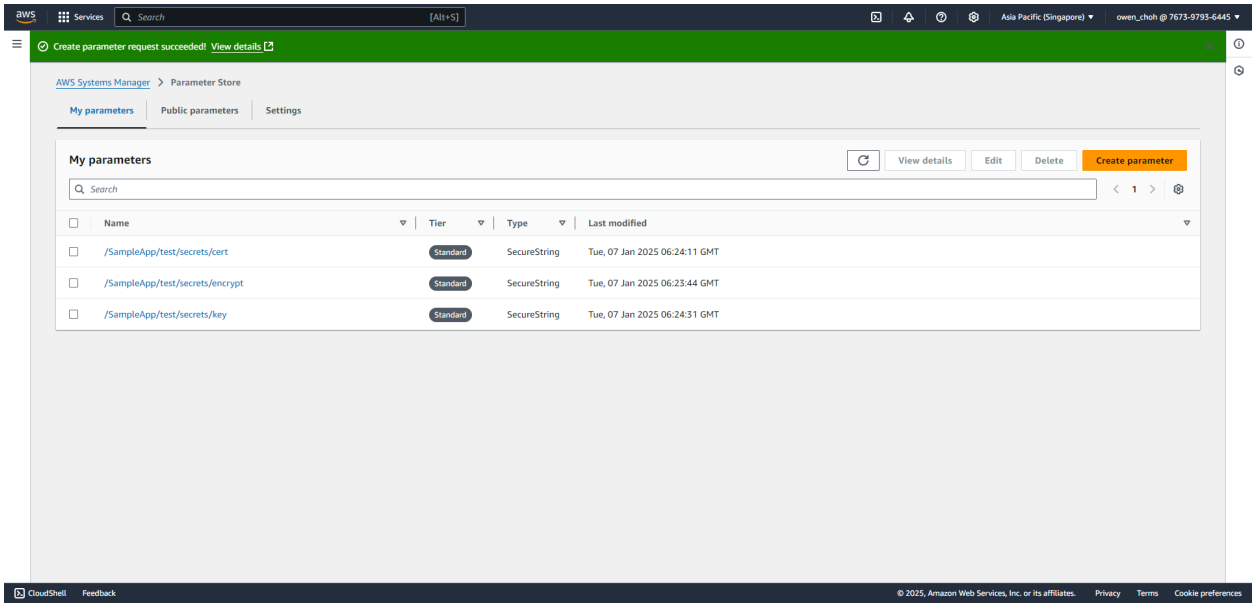
4. Scroll down and select the "**SecureString**" type for each of the secrets. The KMS key source will be the default option.



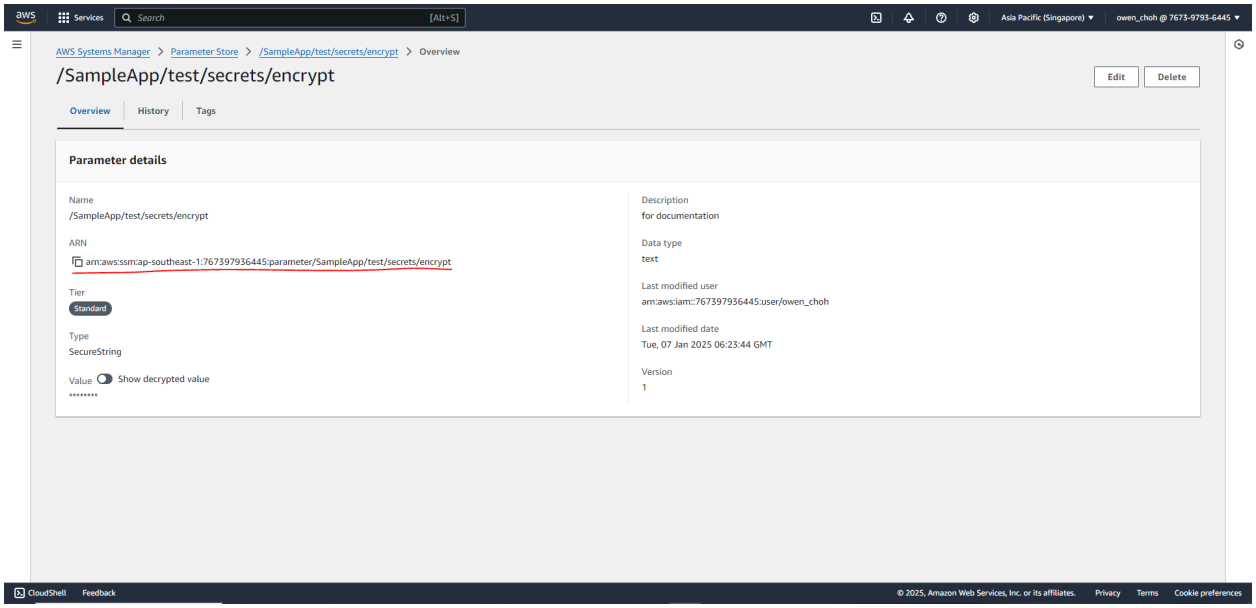
5. Scroll down and paste the secret into the "**Value**" field.



6. Perform the previous steps to create parameters for all 3 secrets. This is the screen you should see after you created them.



7. Copy the arn of one of the secrets you created for reference in the later steps.
    - Example used in the steps will be
    - arn:aws:ssm:ap-southeast-1:767397936445:parameter/**SampleApp/test/secrets/encrypt**



Step Recap

After completing this step, you should have:

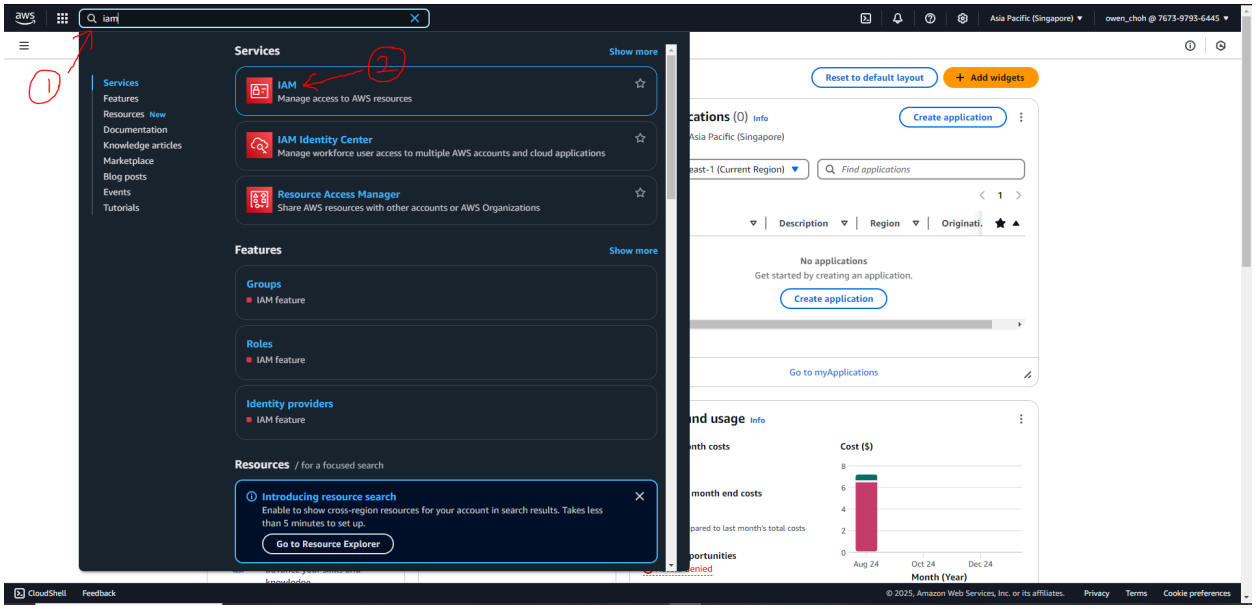
1. Created 3 parameters containing the secrets.
    - Encryption key - name: **/SampleApp/test/secrets/encrypt**
    - Certificate - name: **/SampleApp/test/secrets/cert**
    - Private Key - name: **/SampleApp/test/secrets/key**

2. Recorded one of the arn of the parameters for reference later.
    - For example, “arn:aws:ssm:ap-southeast-1:767397936445:parameter/**SampleApp/test/secrets/encrypt**”.

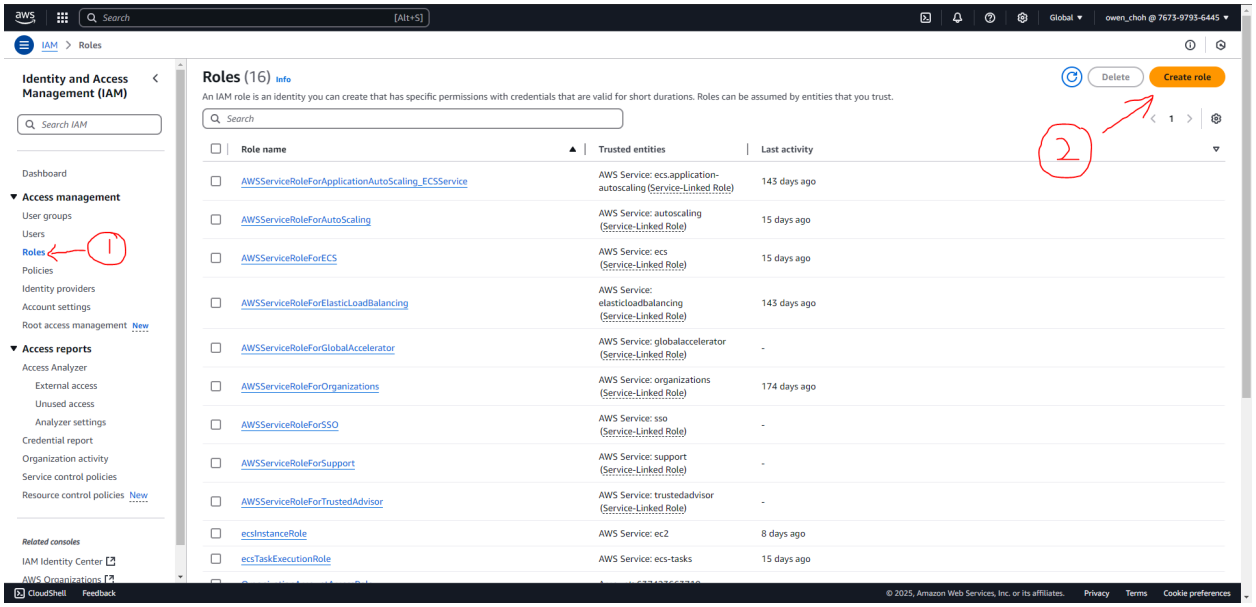
Step 2a: IAM Role Setup

The Sample App will need two roles to function, one for the EC2 instance and one to retrieve the secret from the parameter store to align with best practices.

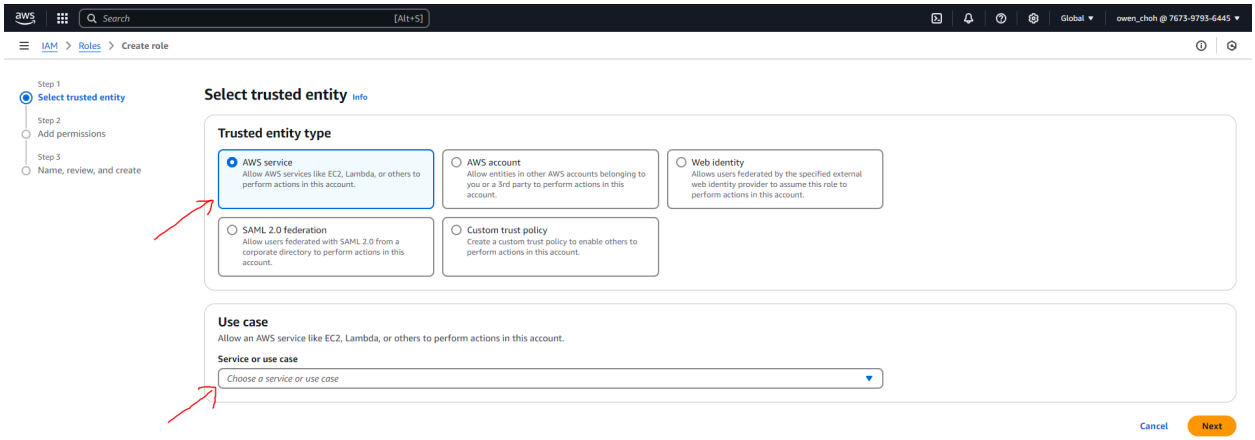
1. Navigate to the **IAM** console.

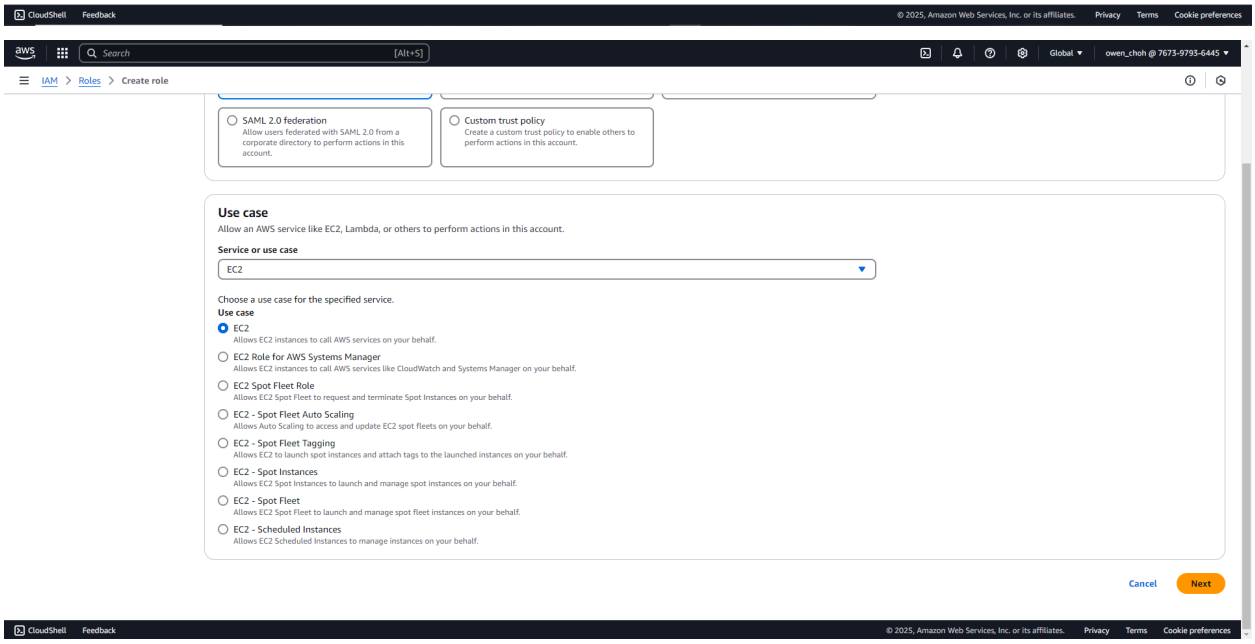


2. Click on “**Roles**” under Access management and click “**Create role**”

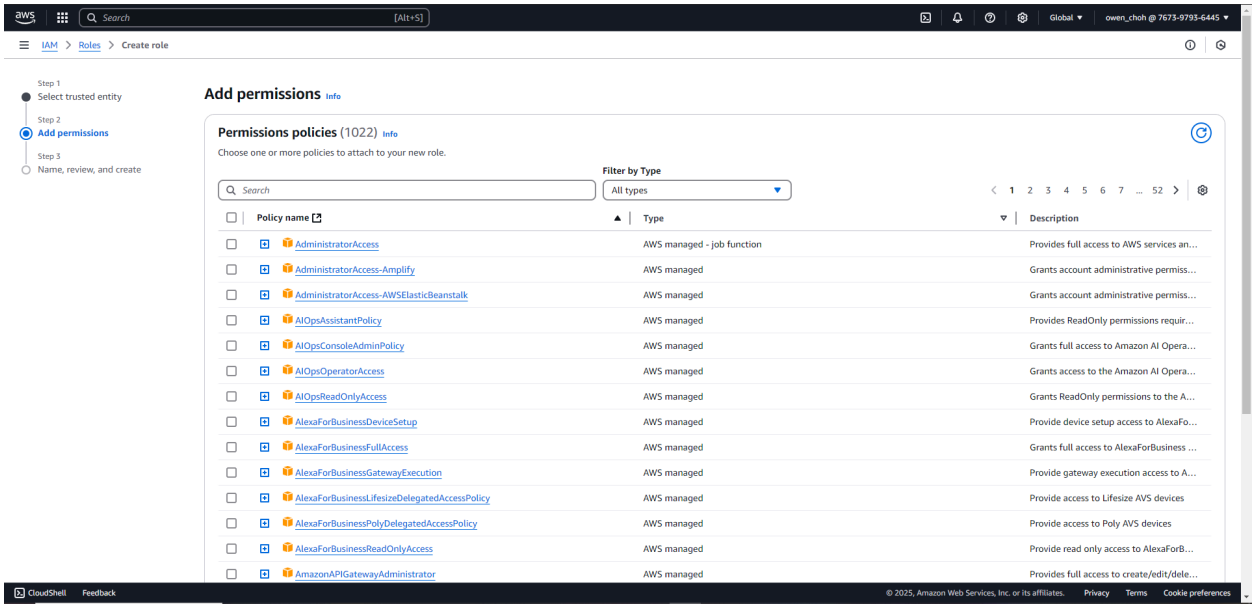


3. This IAM role is for the EC2 instance later. Select “**AWS service**” under Trusted entity type and select “**EC2**” under Use case. Then click on “**Next**” at the bottom of the screen to proceed.

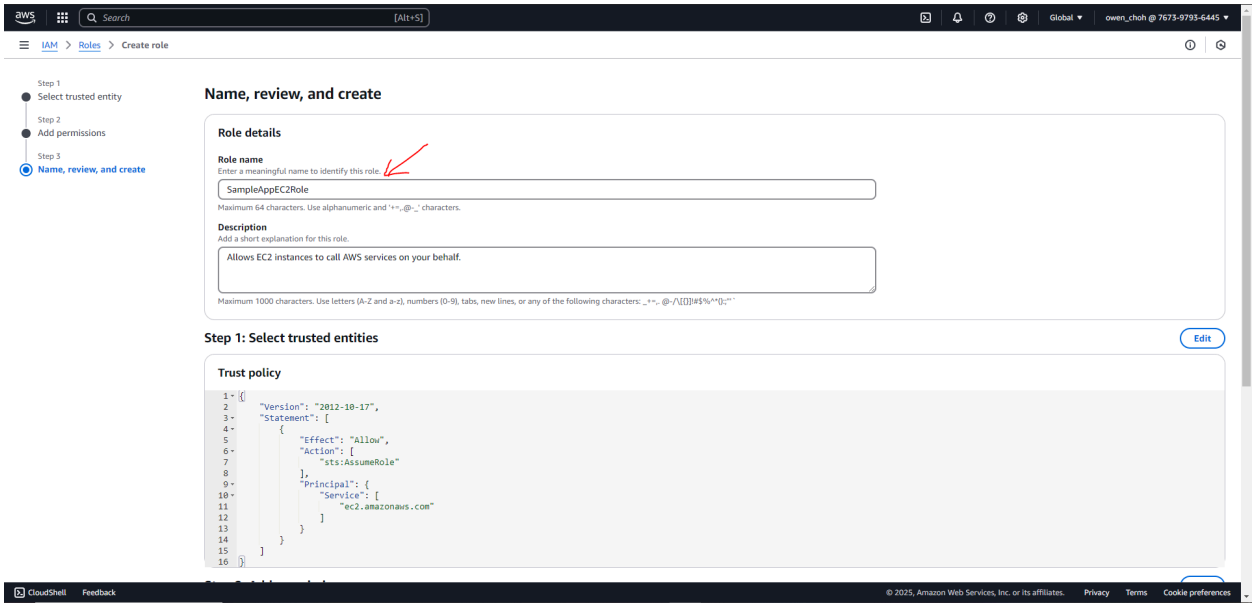




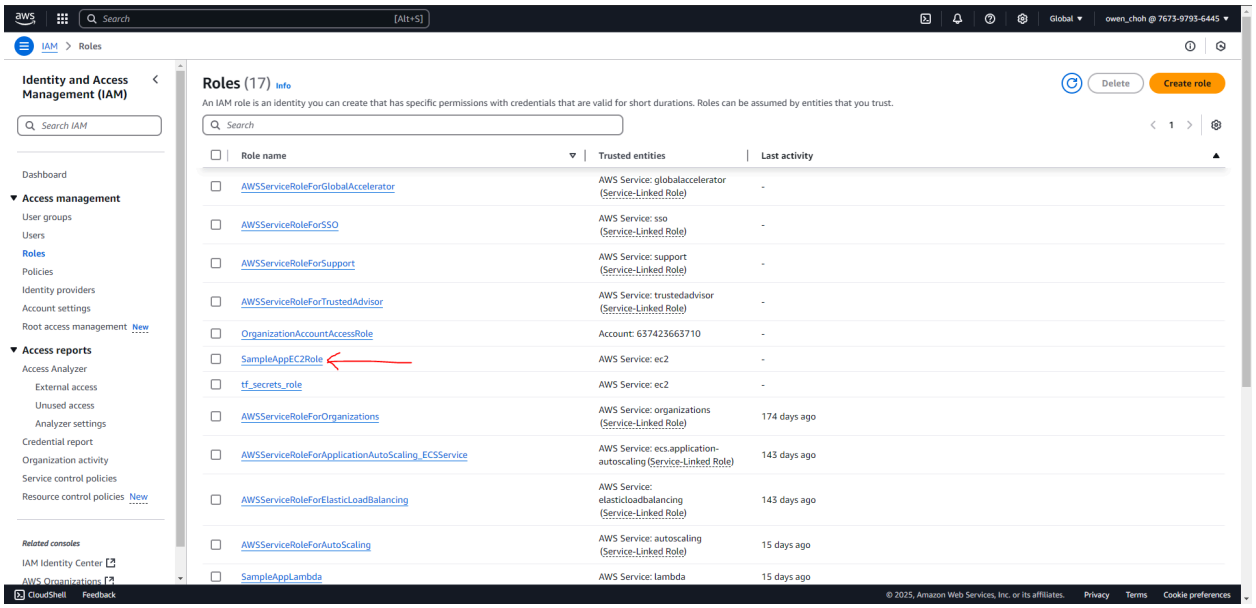
4. On this screen, scroll to the bottom and click next



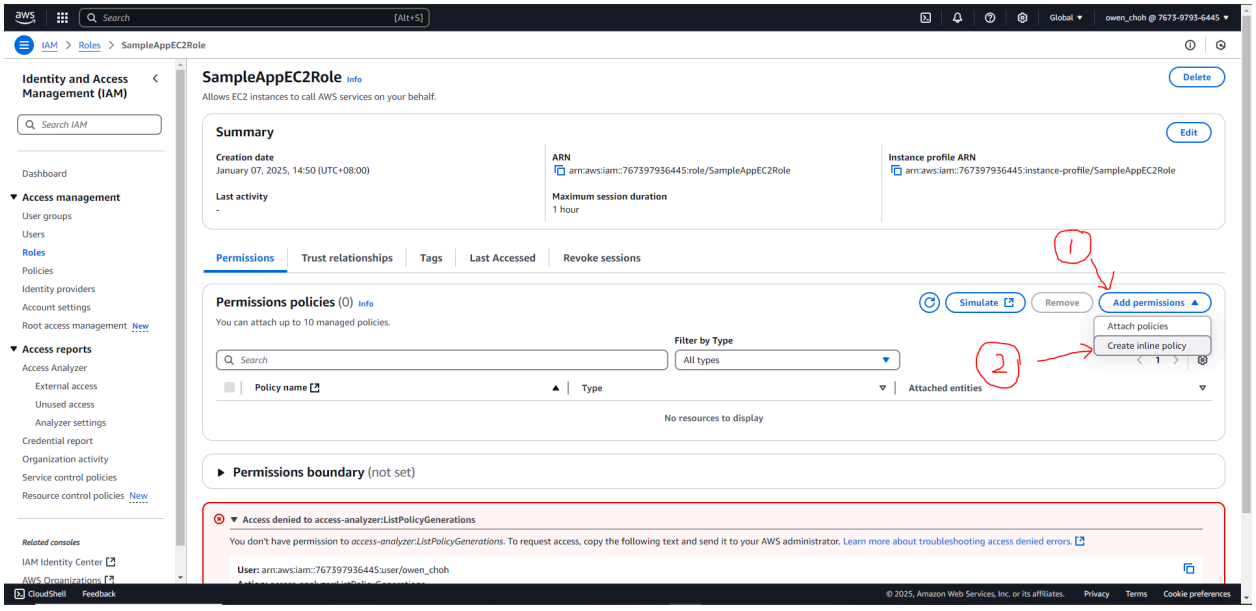
5. Fill in the role name and scroll to the bottom and click “Create role”. It will be named “**SampleAppEC2Role**” in this example.



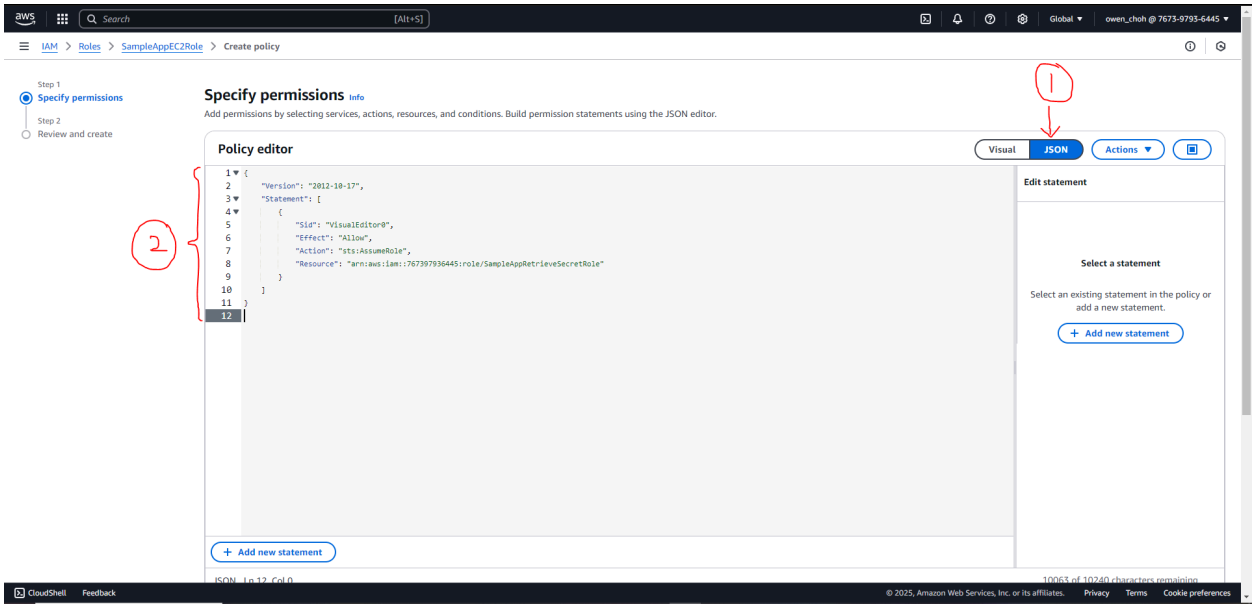
6. Look for the role you just created in the list of roles and click on it.



7. On the role permissions screen, click on “Add permissions” and “Create inline policy”.



8. On the create policy screen, click on “**JSON**” and overwrite the lines in the “**Policy editor**” box. The lines are included below for your convenience. Click “**Next**” at the bottom of the screen once you are done.

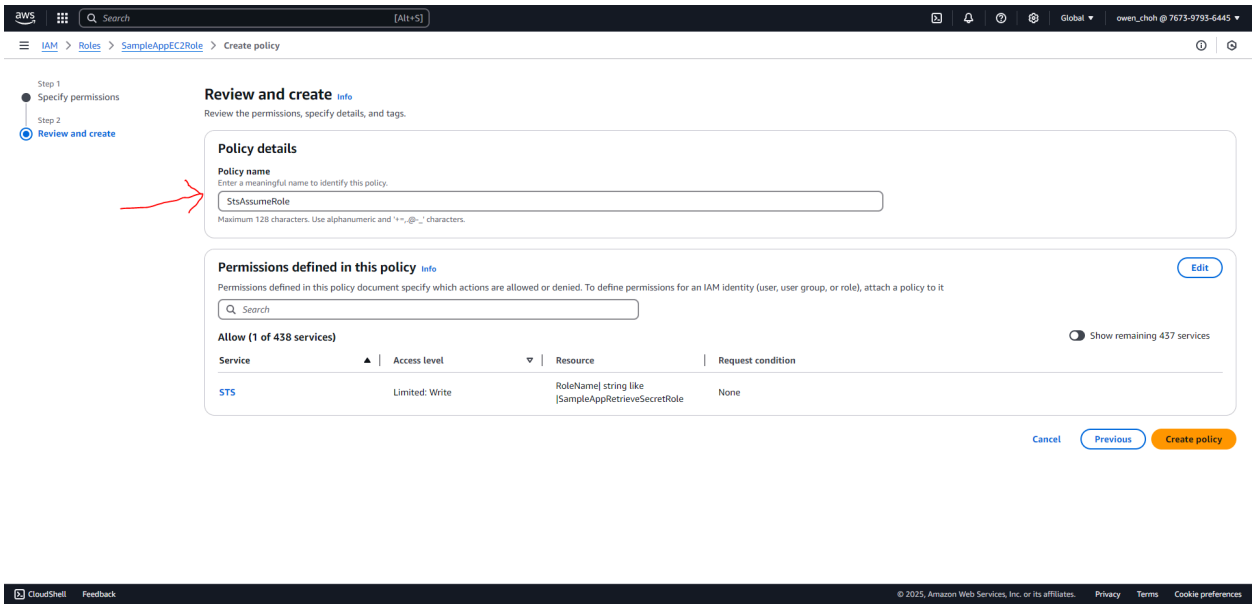


The following are the lines to include in the policy editor. Do take note:

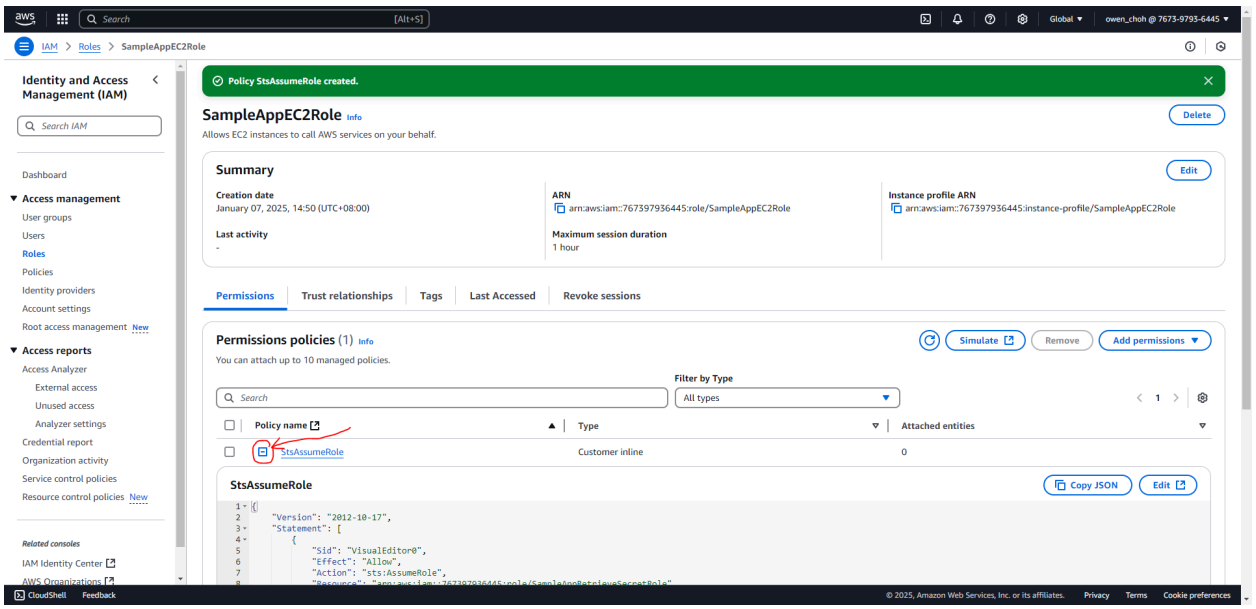
- The numbers in **green** should be your account number (found beside your username at the top right corner of the screen)
- The words in **blue** will be the name of the role (to be created in the later steps) to retrieve secrets from the parameter store.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "sts:AssumeRole",
         "Resource": "arn:aws:iam::767397936445:role/SampleAppRetrieveSecretRole"
      }
   ]
}
```

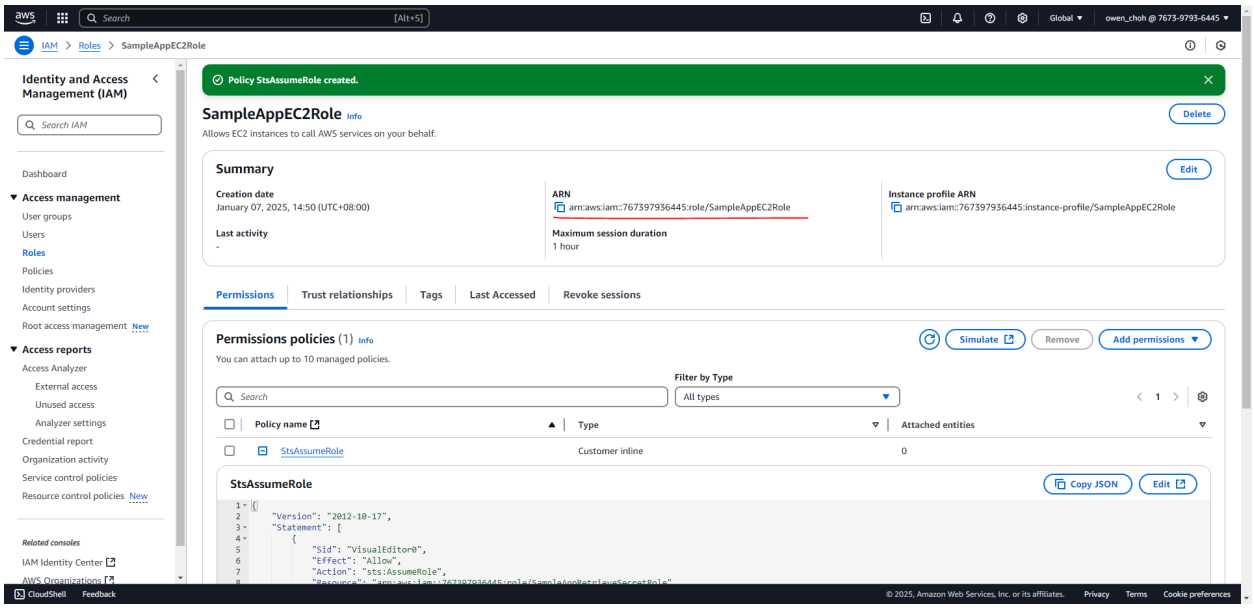
9. Input a name for this policy. It will be named “**StsAssumeRole**” in this example. Click on “**Create policy**” once you are done.



10. This is the screen you will see once it is created. You can click on the button beside the policy name to check if the lines you pasted earlier are correct.

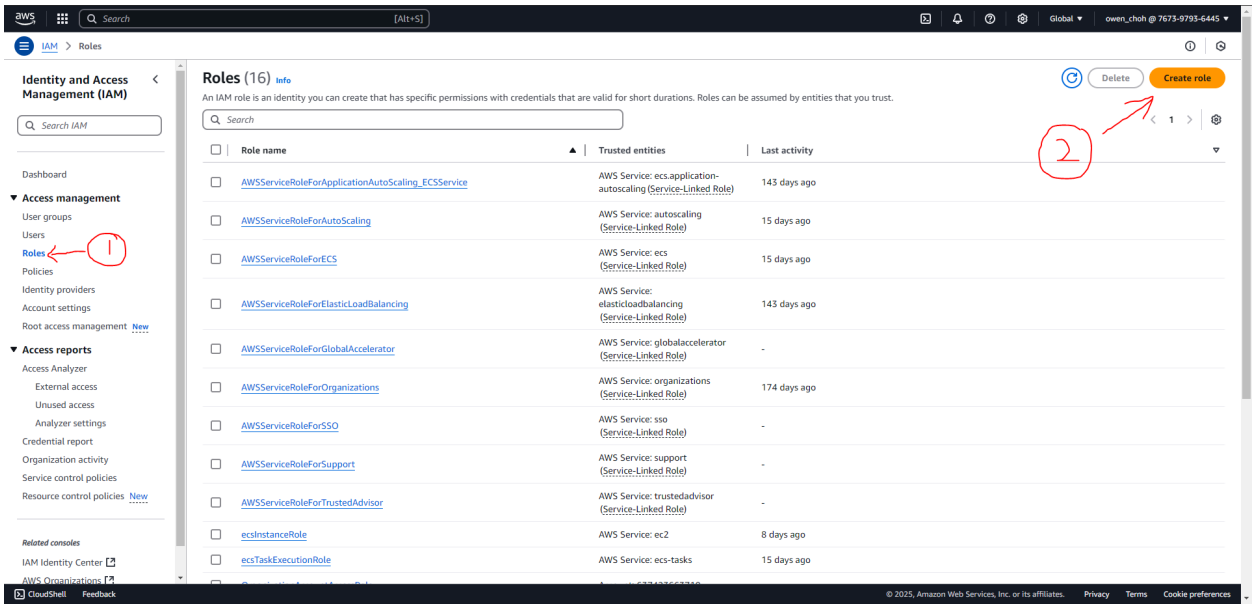


11. Copy the arn of the role for the EC2. It will be "**arn:aws:iam::767397936445:role/SampleAppEC2Role**" in this example.

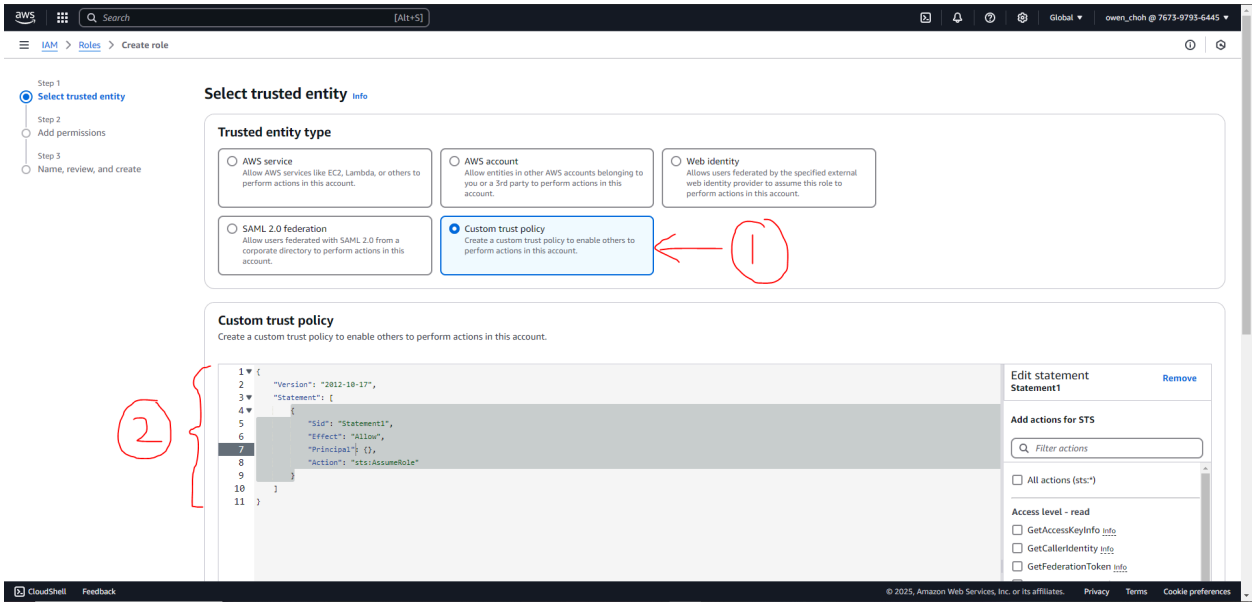


Step 2b: IAM Role to retrieve secrets

12. Now we need to create a new IAM role to retrieve secrets with the same role name as the one provided when creating an inline policy (It should be **SampleAppRetrieveSecretRole** if you followed the example given). Please proceed to click on create a new role from the role screen.



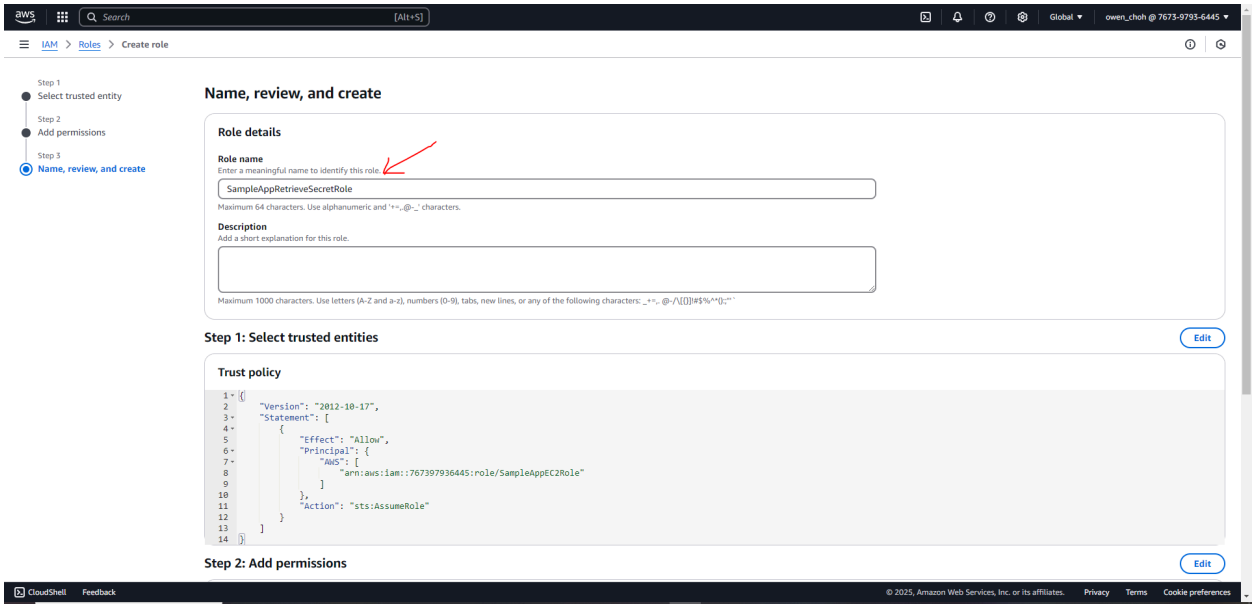
13. On the create role screen, click on “Custom trust policy”. You should see the box labeled as (2) below.



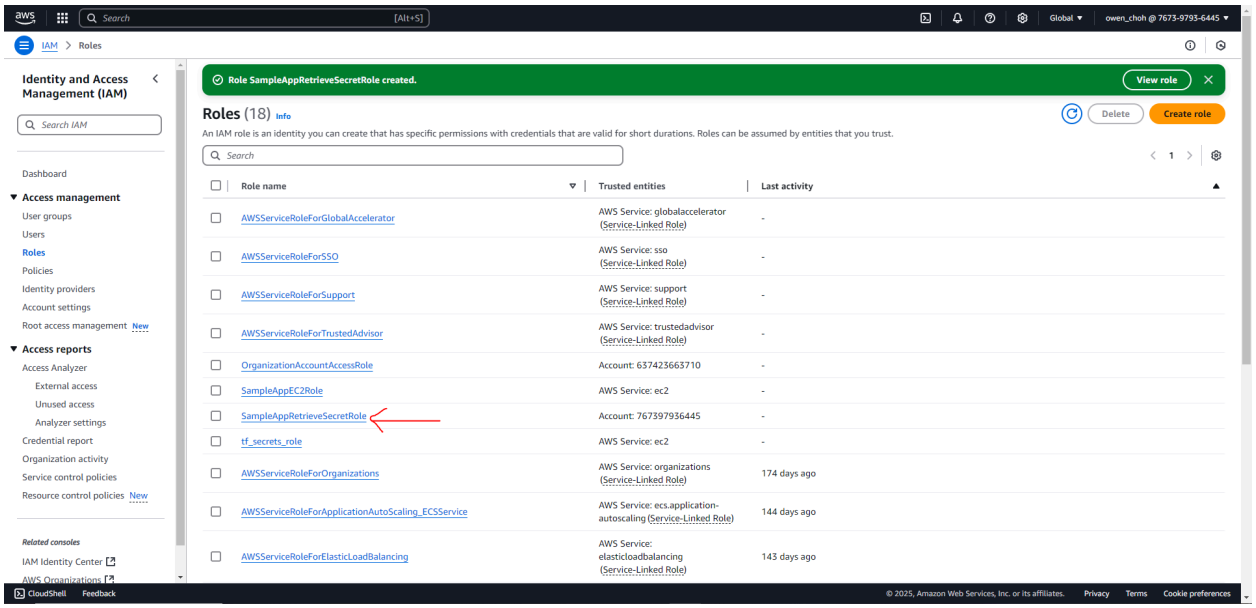
14. Put in the arn of the EC2 role created earlier to allow it to access this role that we are creating. Below are the lines you should see after editing the policy. Take note that the numbers in **green** should be your account number instead. Click "**Next**" when you are done and click "**Next**" again when you are on the "Add permissions" screen.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "AWS": [
               "arn:aws:iam::767397936445:role/SampleAppEC2Role"
            ]
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

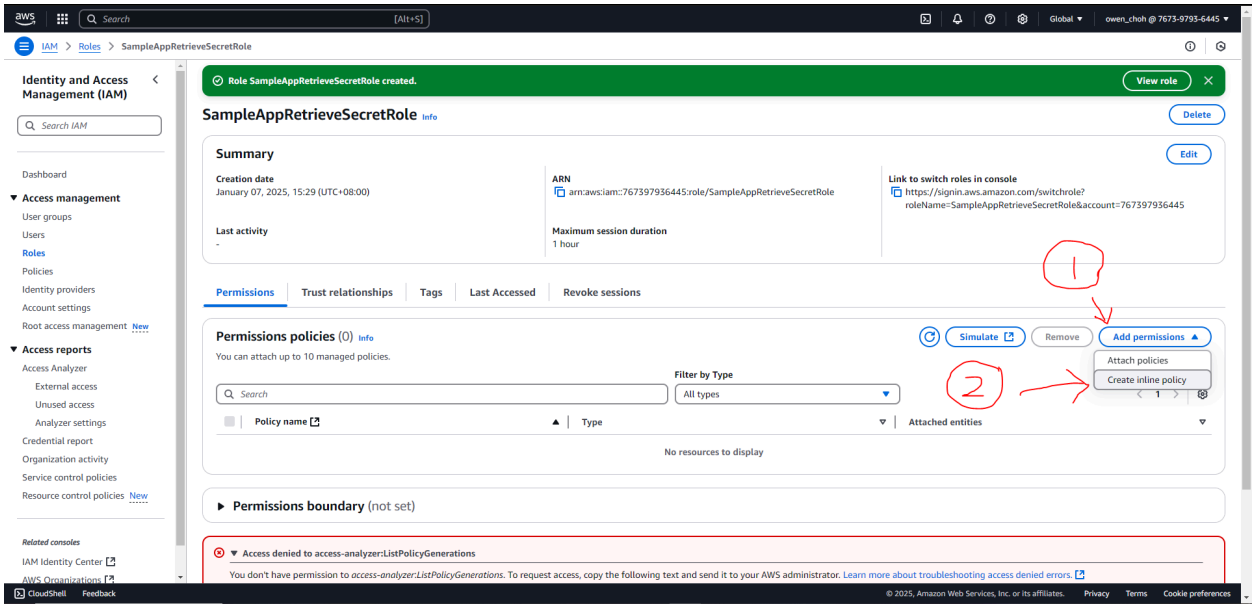
15. Input the same role name as the one provided when creating an inline policy for the EC2 role (It should be **SampleAppRetrieveSecretRole** if you followed the example given). Scroll down and click on "Create role" once you are done.



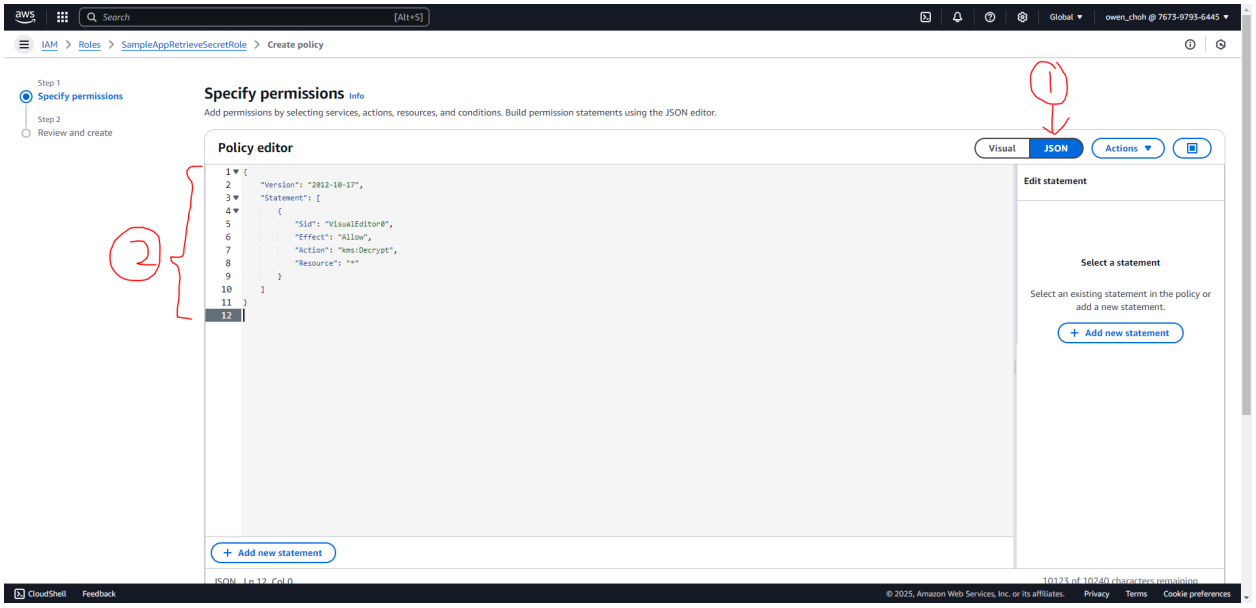
16. Look for the role you just created in the list of roles and click on it.



17. On the role permissions screen, click on “**Add permissions**” and “**Create inline policy**”.



18. On the create policy screen, click on "**JSON**" and overwrite the lines in the "**Policy editor**" box. The lines are included below for your convenience. Click "**Next**" at the bottom of the screen once you are done.

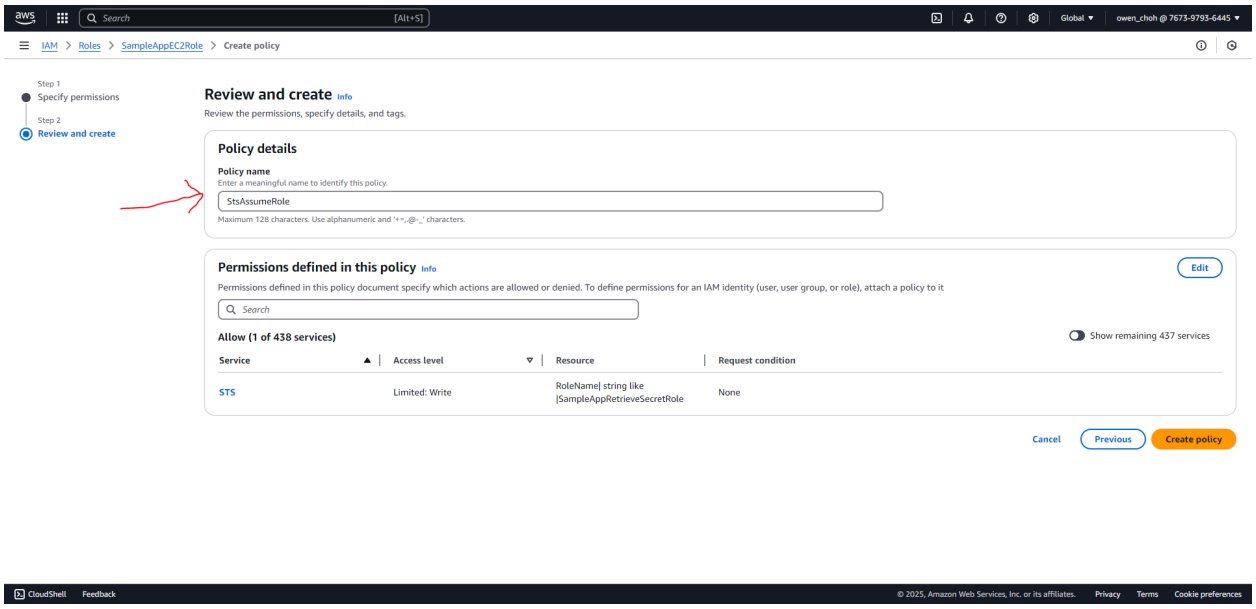


The following are the lines to include in the policy editor to decrypt secrets when retrieving from the parameter store.

```
{
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "*"
        }
    ]
}
```

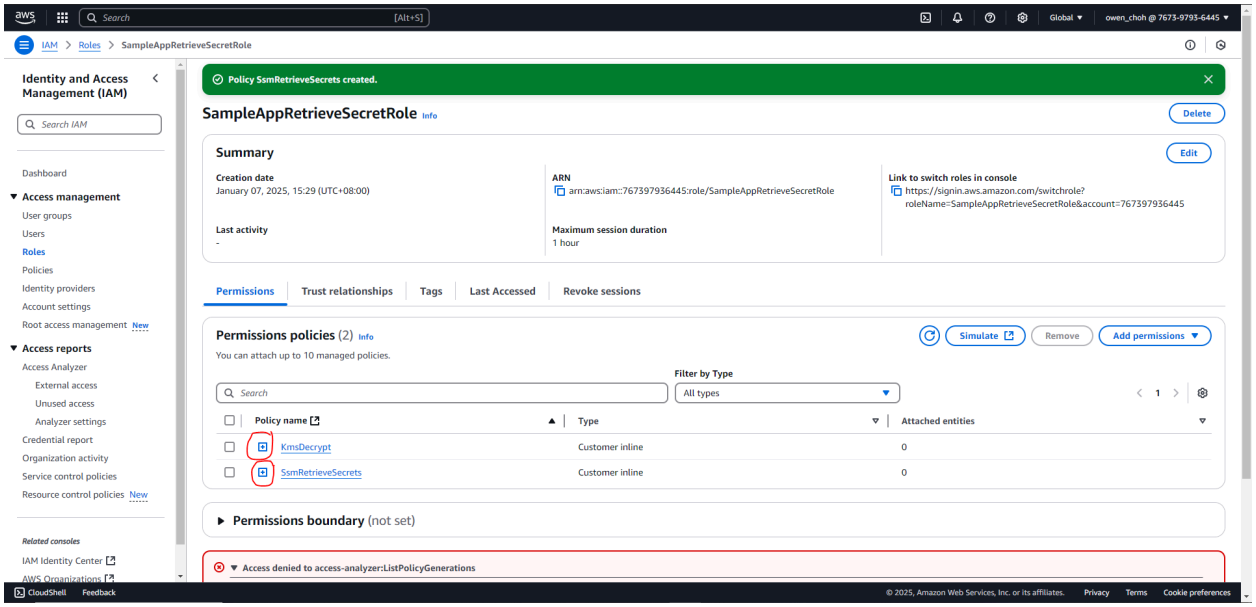
19. Input a name for this policy. It will be named “**KmsDecrypt**” in this example. Click on “**Create policy**” once you are done.

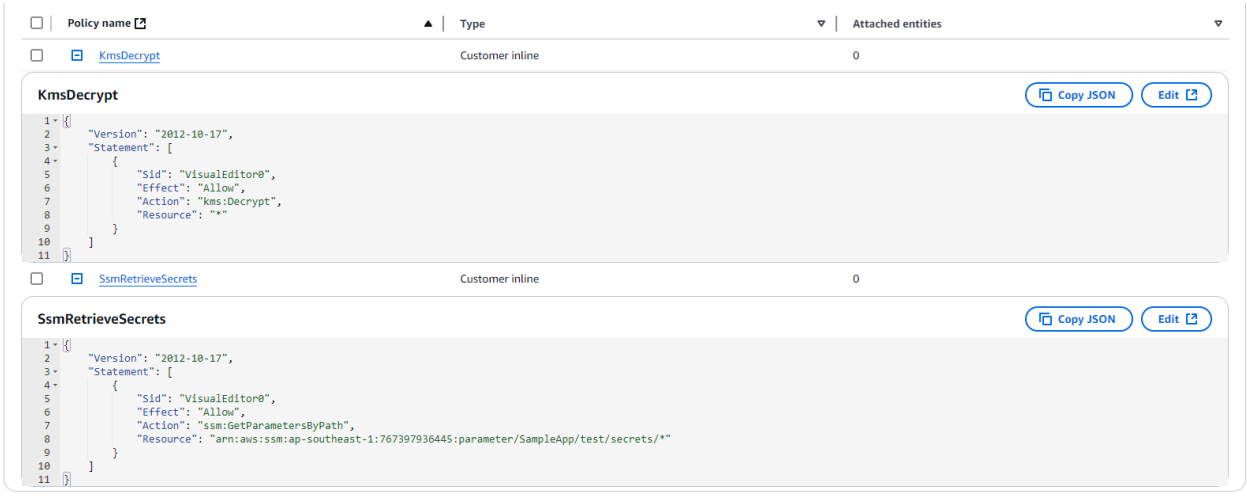


20. Please repeat the same steps to add the **inline policy** provided below to retrieve secrets from the parameter store. It will be named as “**SsmRetrieveSecrets**” in this example.
    - Do take note that the “resource” value is dependent on the arn of the secret in the parameter store. In our example at the end of “Step 1: Parameter Store Configuration”, the arn is “arn:aws:ssm:ap-southeast-1:767397936445:parameter/**SampleApp/test/secrets/encrypt**”. Thus the resource value here should be “.../SampleApp/test/secrets/*****”.
    - Please also remember to change the numbers in **green** to your account number.

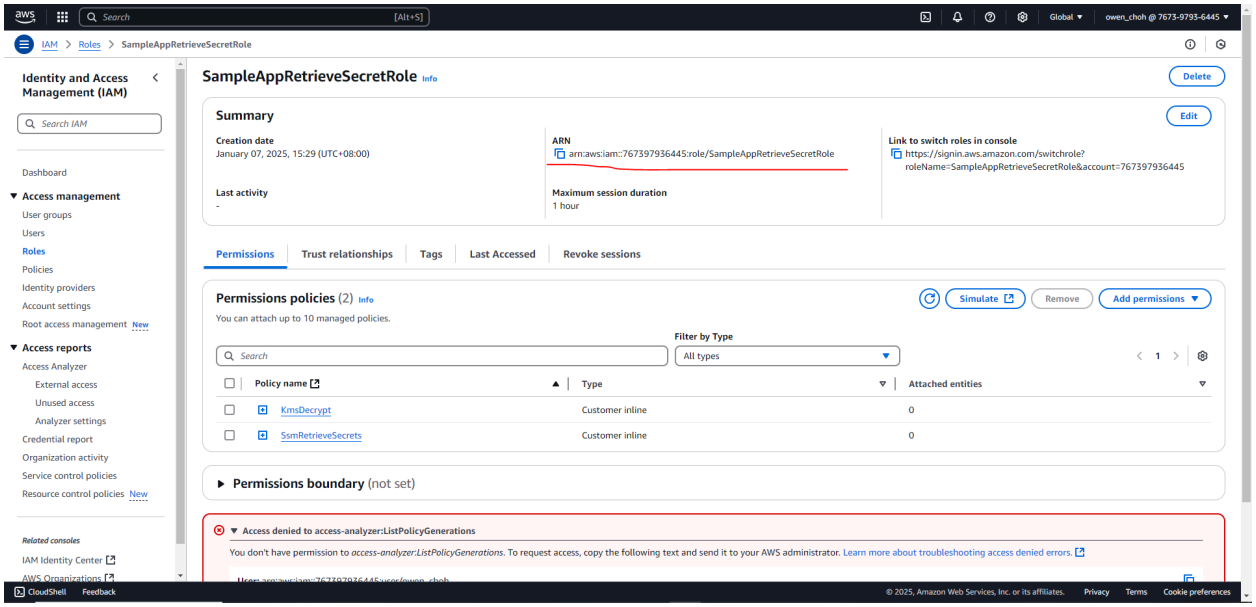
```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "ssm:GetParametersByPath",
         "Resource":
"arn:aws:ssm:ap-southeast-1:767397936445:parameter/SampleApp/test/secrets/*"
      }
   ]
}
```

21. This is the screen you will see once both policies are created. You can click on the button beside the policy name to check if the lines you pasted earlier are correct.





22. Copy the arn of the role to retrieve secrets as you will need it later.



Step Recap

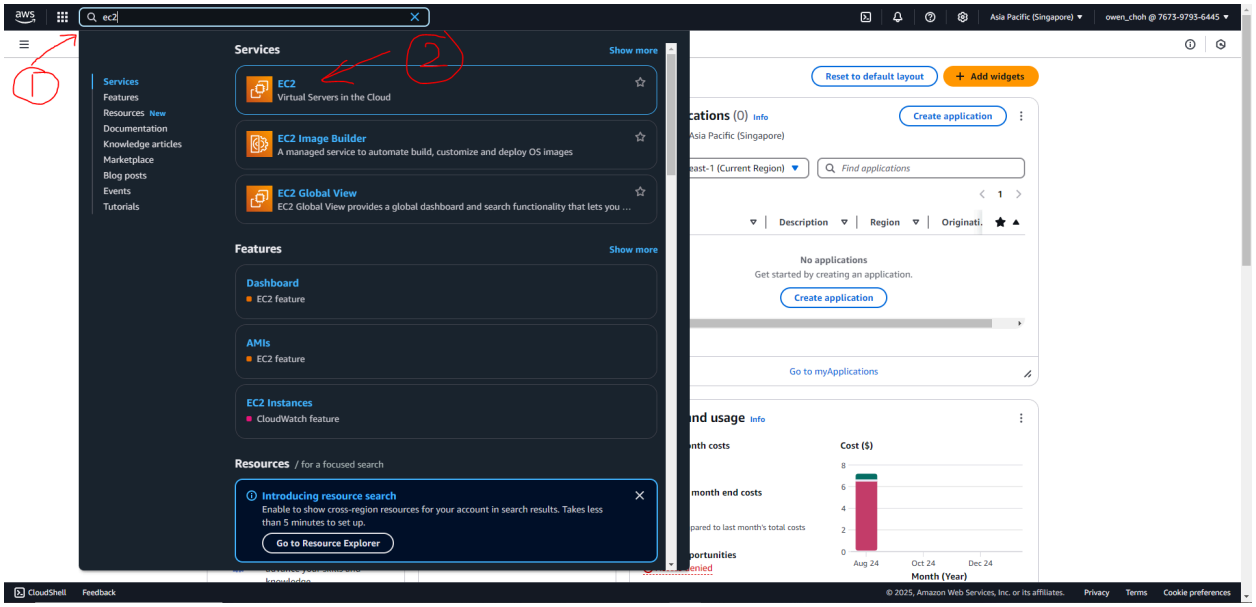
After completing step 2A and 2B, you should have:

1. Created two roles:
    a. “**SampleAppEC2Role**” for the EC2 instance
    b. “**SampleAppRetrieveSecretRole**” for the EC2 to retrieve secrets
2. Copied the arn of the “**SampleAppRetrieveSecretRole**” e.g. arn:aws:iam::767397936445:role/SampleAppRetrieveSecretRole

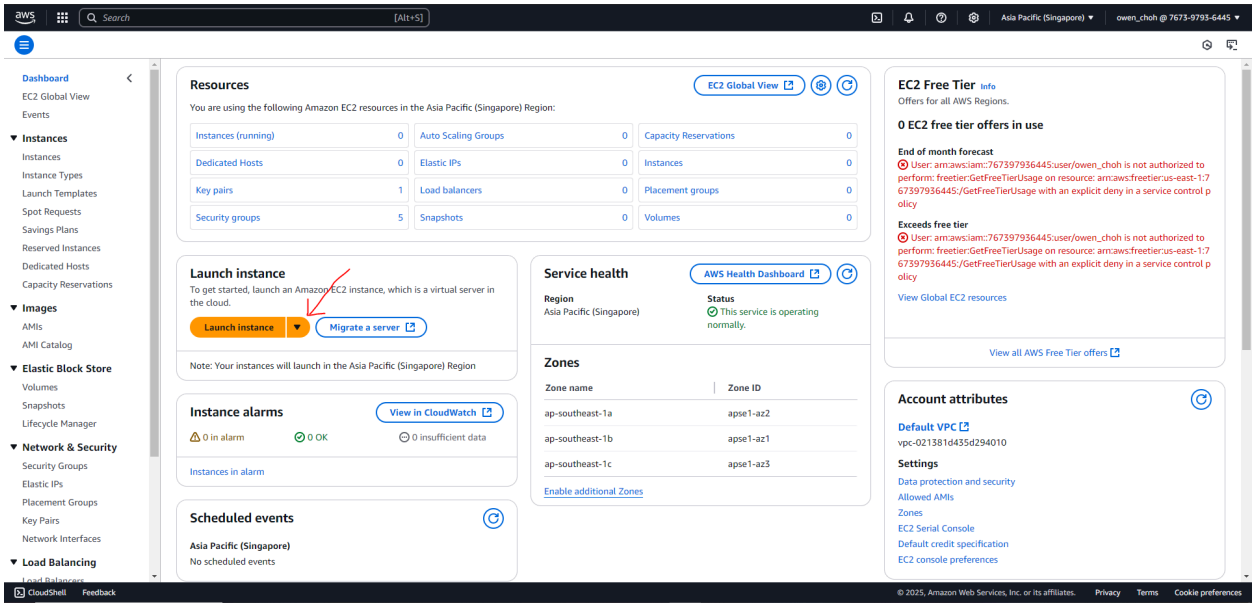
Step 3: EC2 Instance Launch Configuration

This step is to start an EC2 instance on AWS which will host the Sample App for people to use.

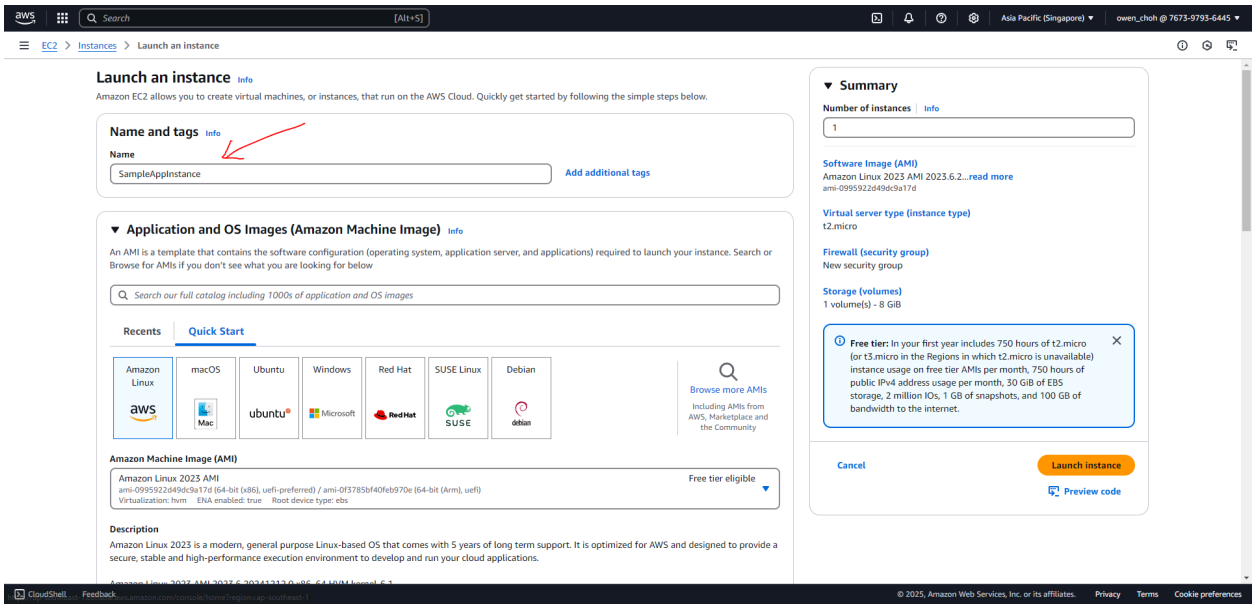
1. Navigate to the EC2 dashboard



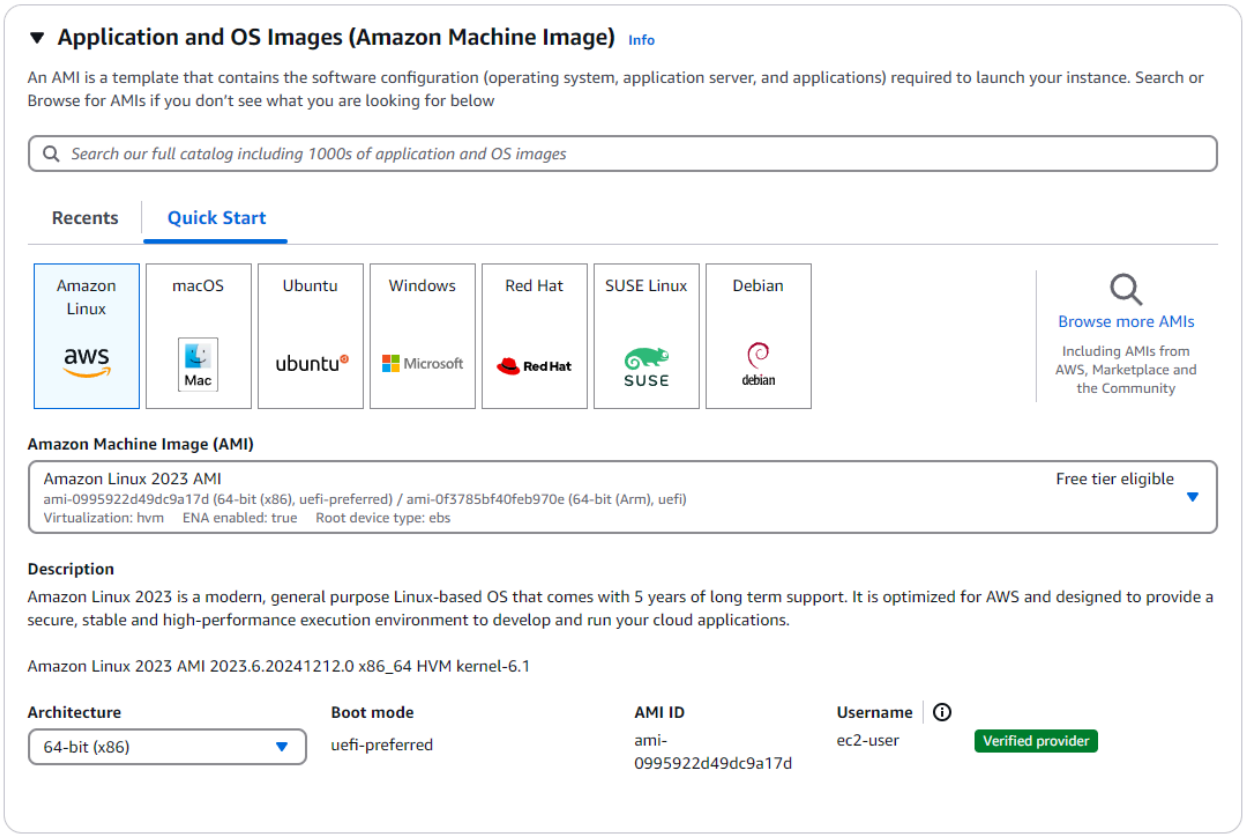
2. Launch an EC2 instance

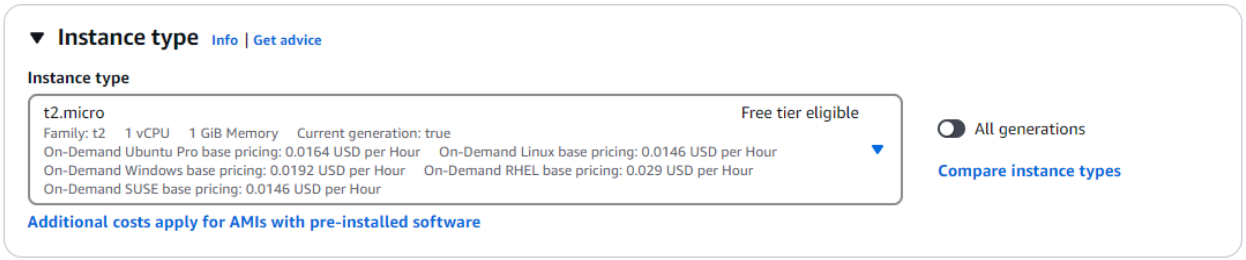


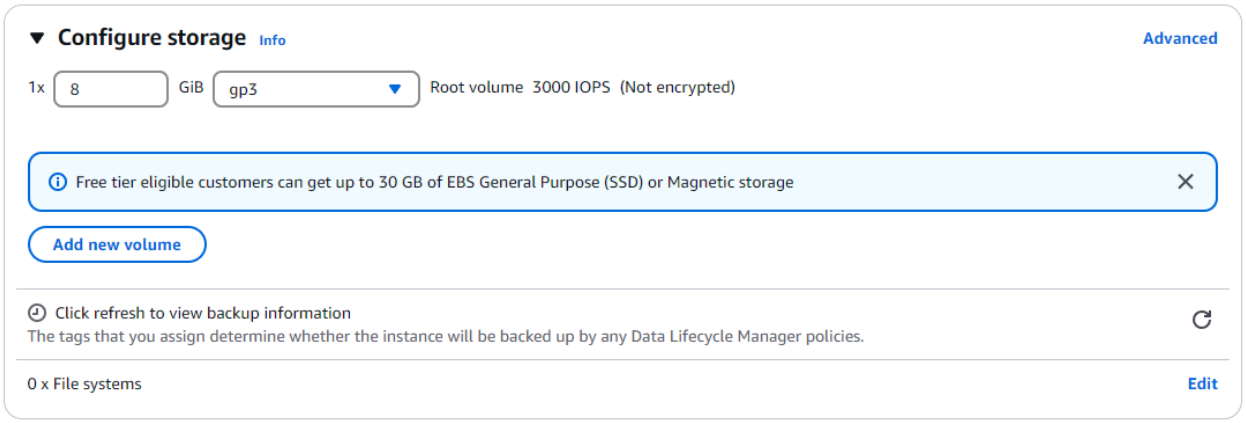
3. Input a name for the instance such as “**SampleAppInstance**”



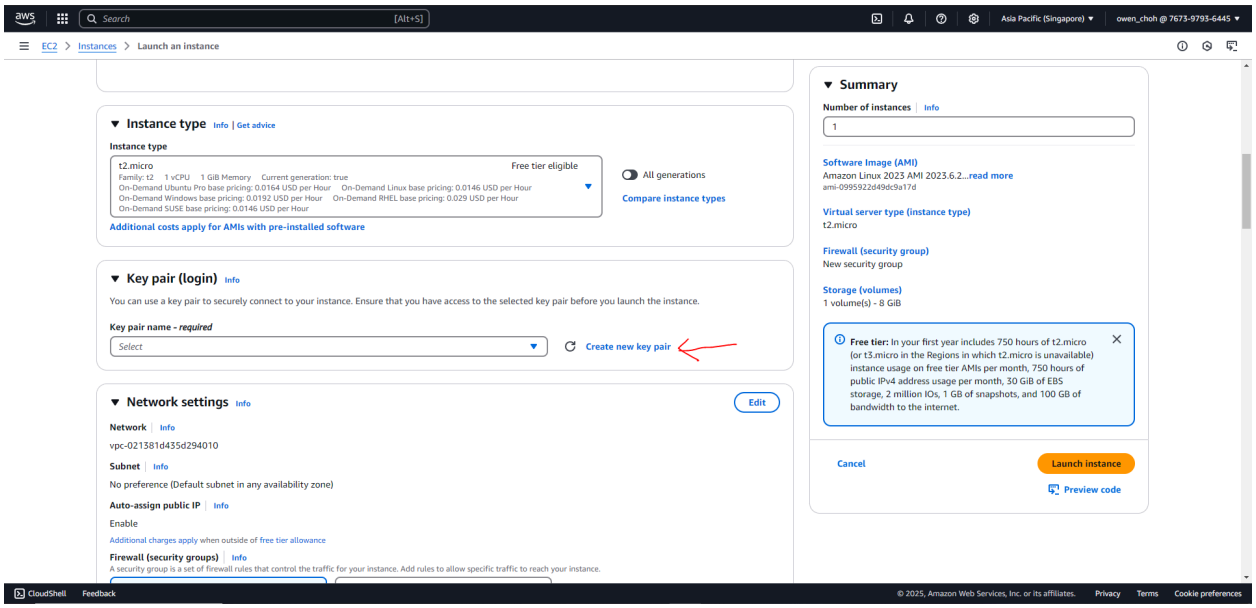
4. In this example, we will use the default options for most of the options but you can choose other options if needed. Screenshots are provided in case you like to verify them.





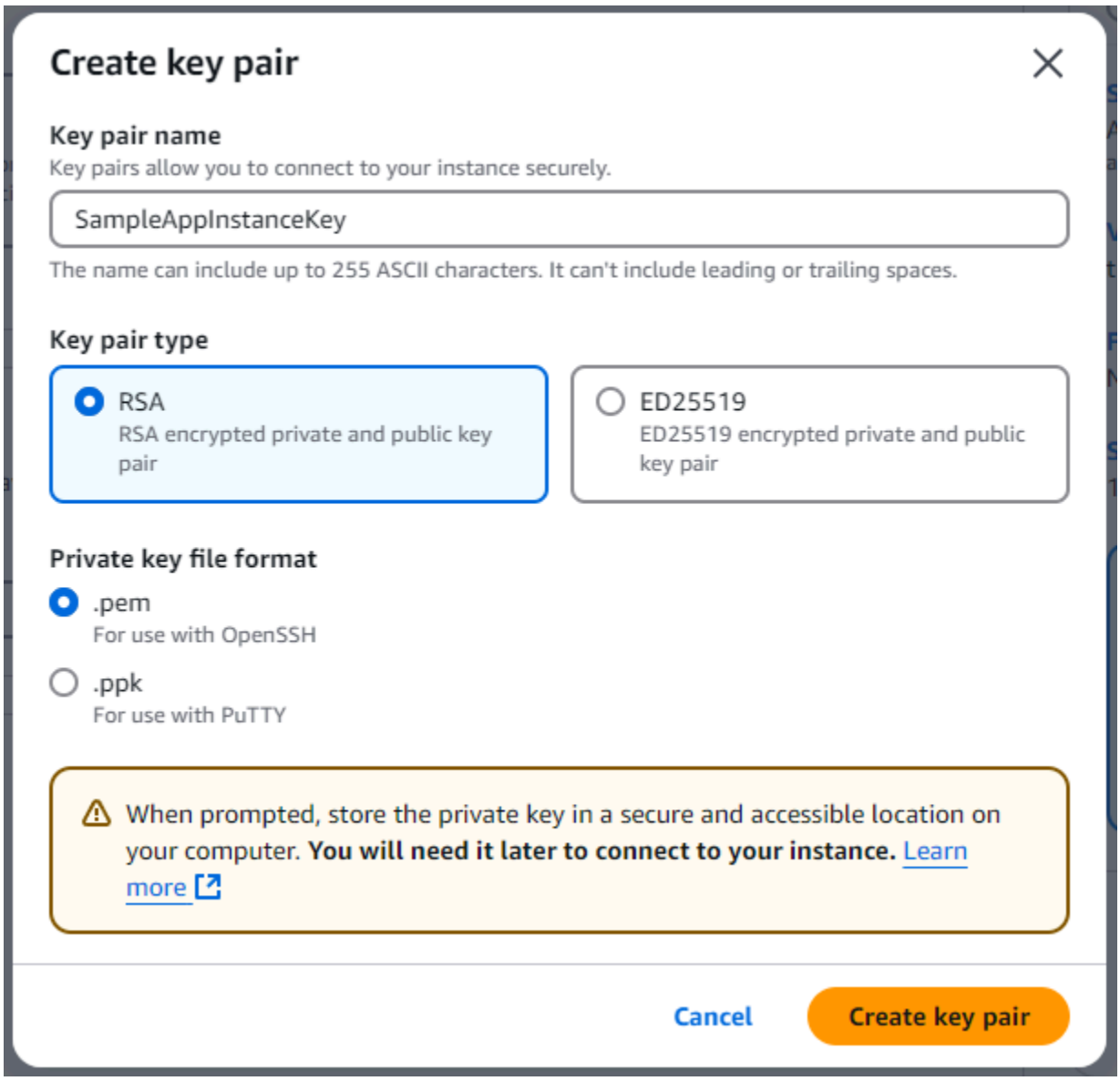


5. Click on create a new key pair.

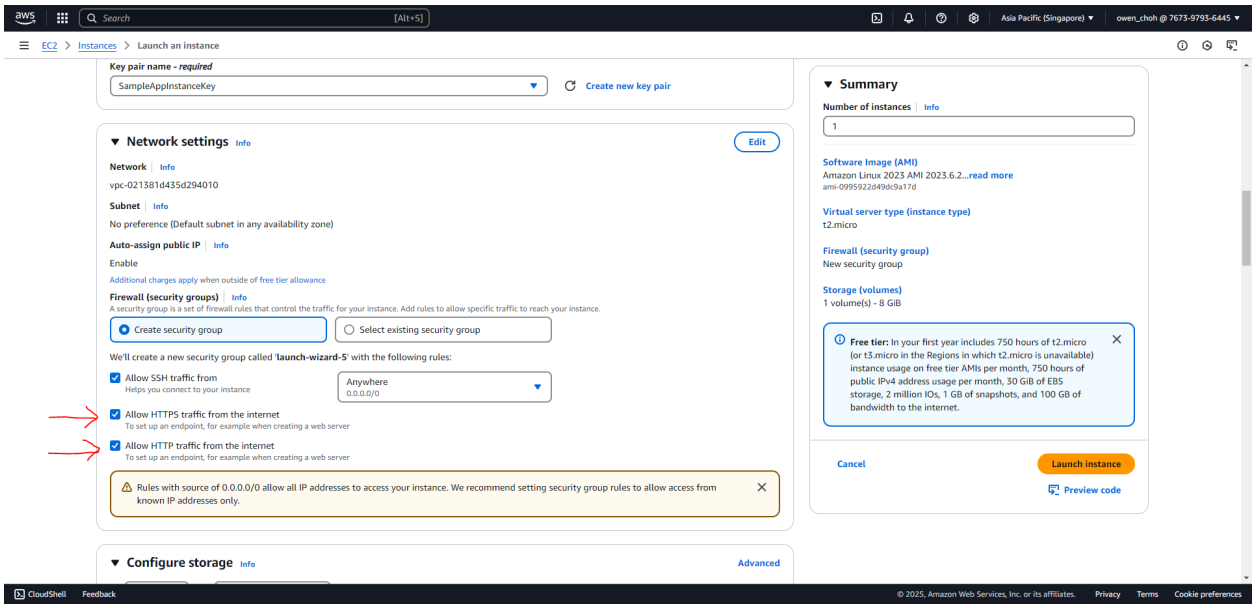


6. Input a name for the key pair such as “SampleAppInstanceKey” and click “Create key pair”. Your browser should prompt you to download a file (named SampleAppInstanceKey.pem in this example) automatically, please save this file as it is

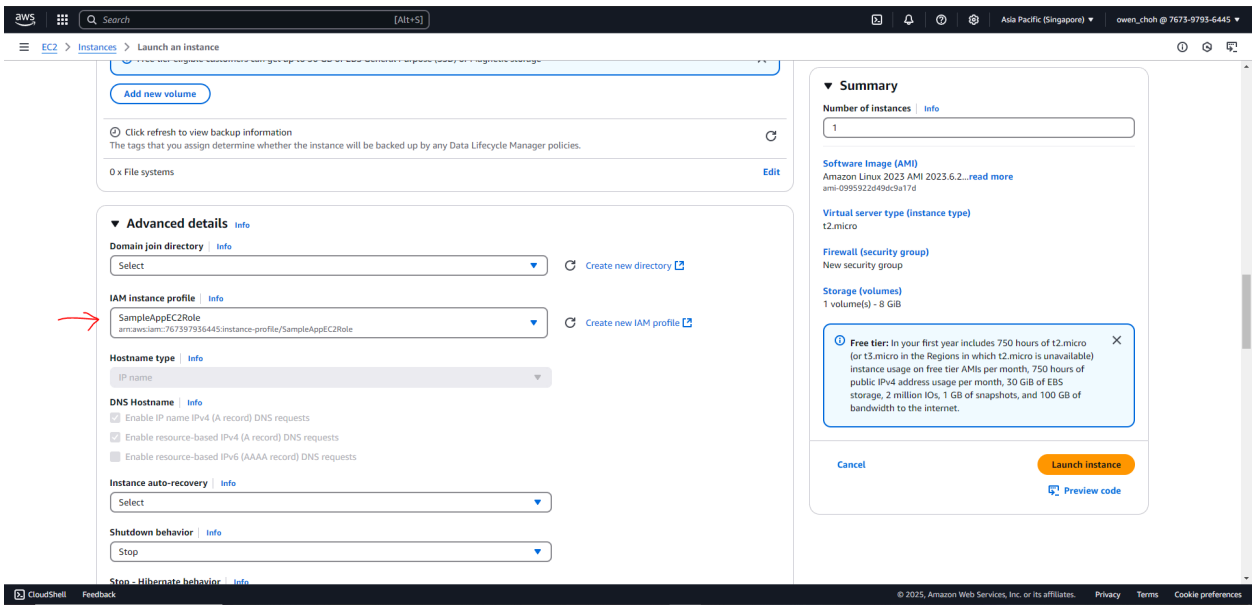
required to access the instance once it is running and there is no way to obtain this file again should you lose it.



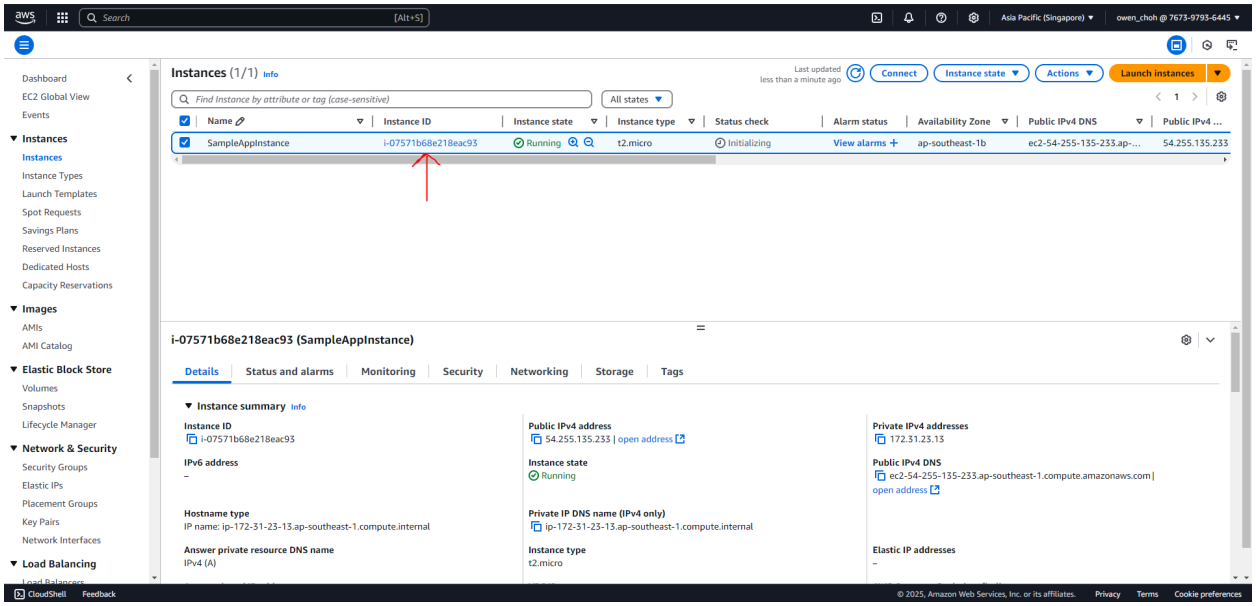
7. Check these two boxes under the “Network settings”



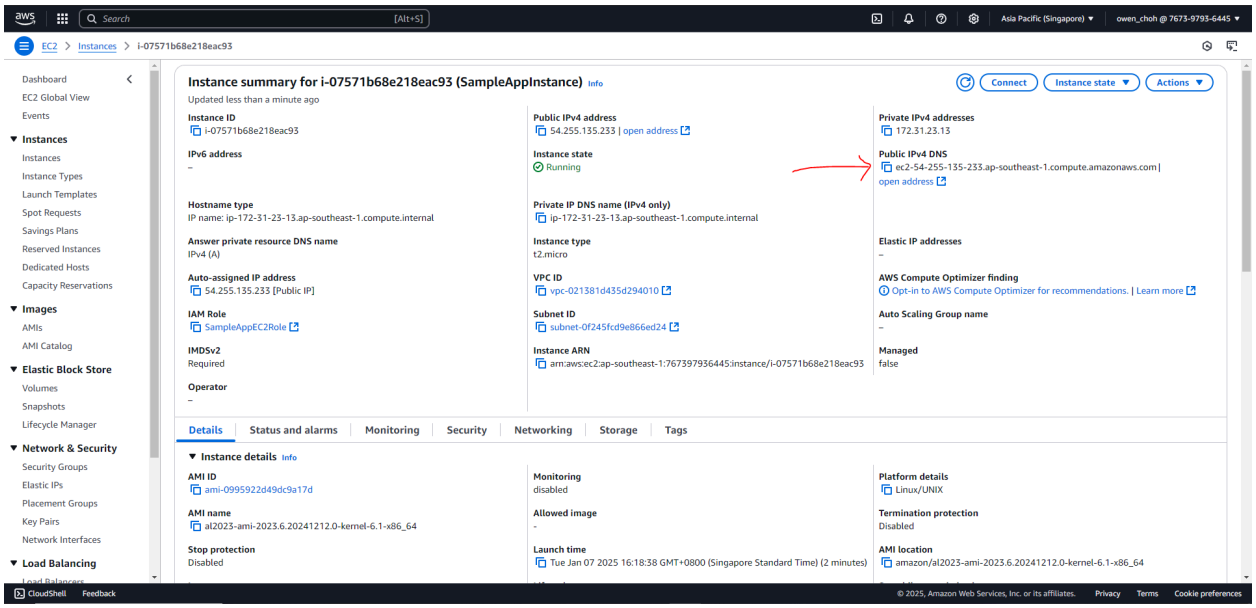
8. Assign the IAM role created in Step 2 to the instance under the “Advanced details” tab. Click on “Launch instance” on the right once you are done.



9. Click on the instance you just created.



10. Copy the “Public IPv4 DNS”, this is the link that users will use to access the sample app once it is running. Please take note that you need to append “http://” to the front of the link to use it (e.g. **http://**ec2-52-77-81-200.ap-southeast-1.compute.amazonaws.com).



Step Recap

After completing this step, you should have:

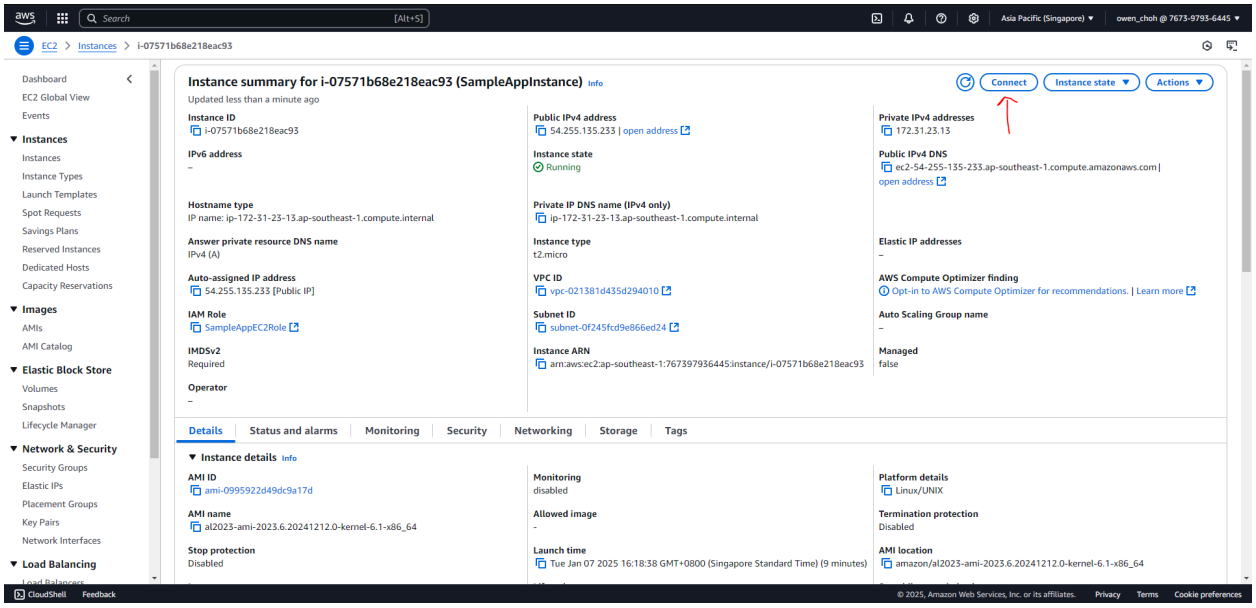
1. An EC2 instance running with the correct IAM role
2. Obtained a key to access the instance (e.g. SampleAppInstanceKey.pem)

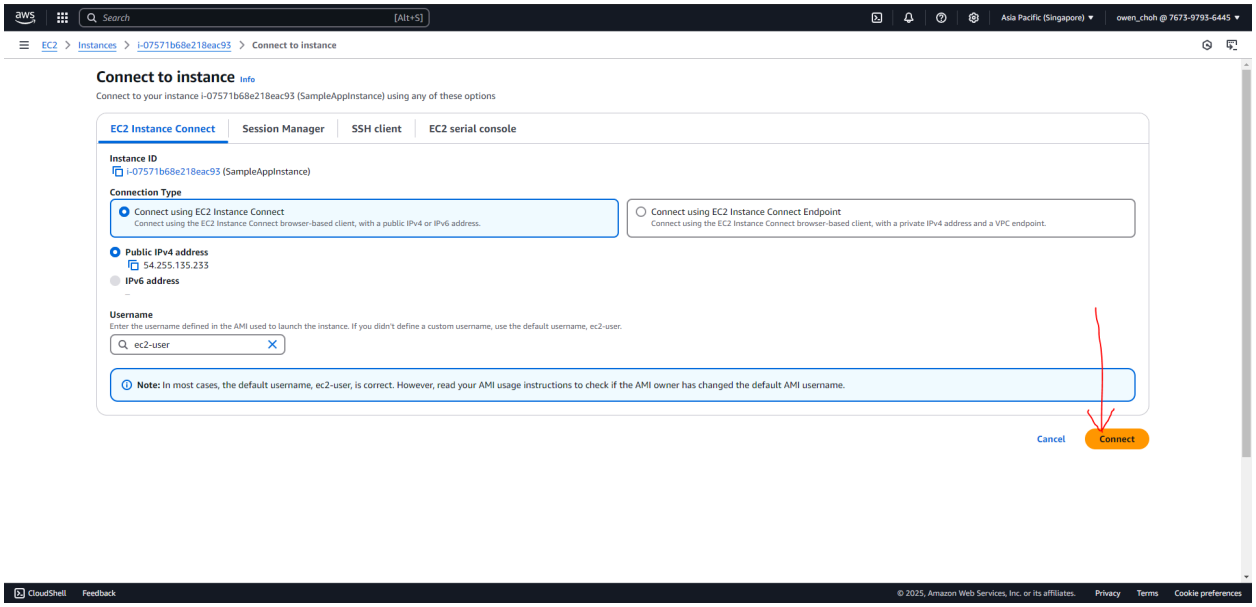
3. A “Public IPv4 DNS” for users to use to access the sample app once it is running (e.g. **http://ec2-52-77-81-200.ap-southeast-1.compute.amazonaws.com**)

Step 4: EC2 Instance installation

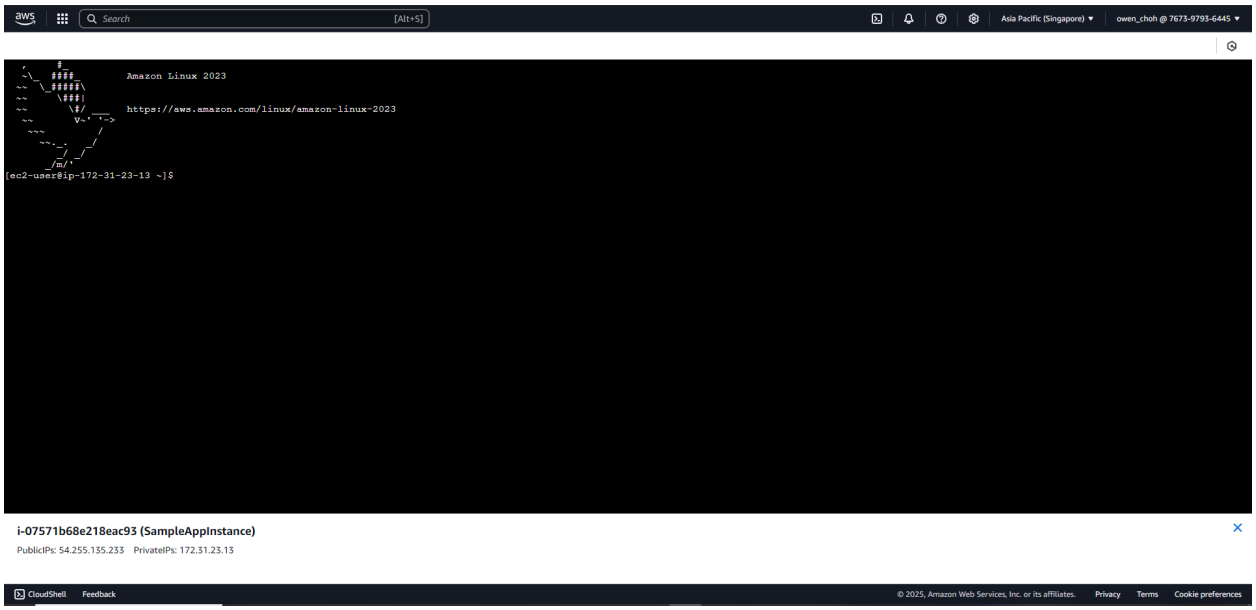
This step is to install the dependencies in order to run the Sample App.

1. Connect to the instance via one of two methods. Both are equivalent.
    - Using the aws console



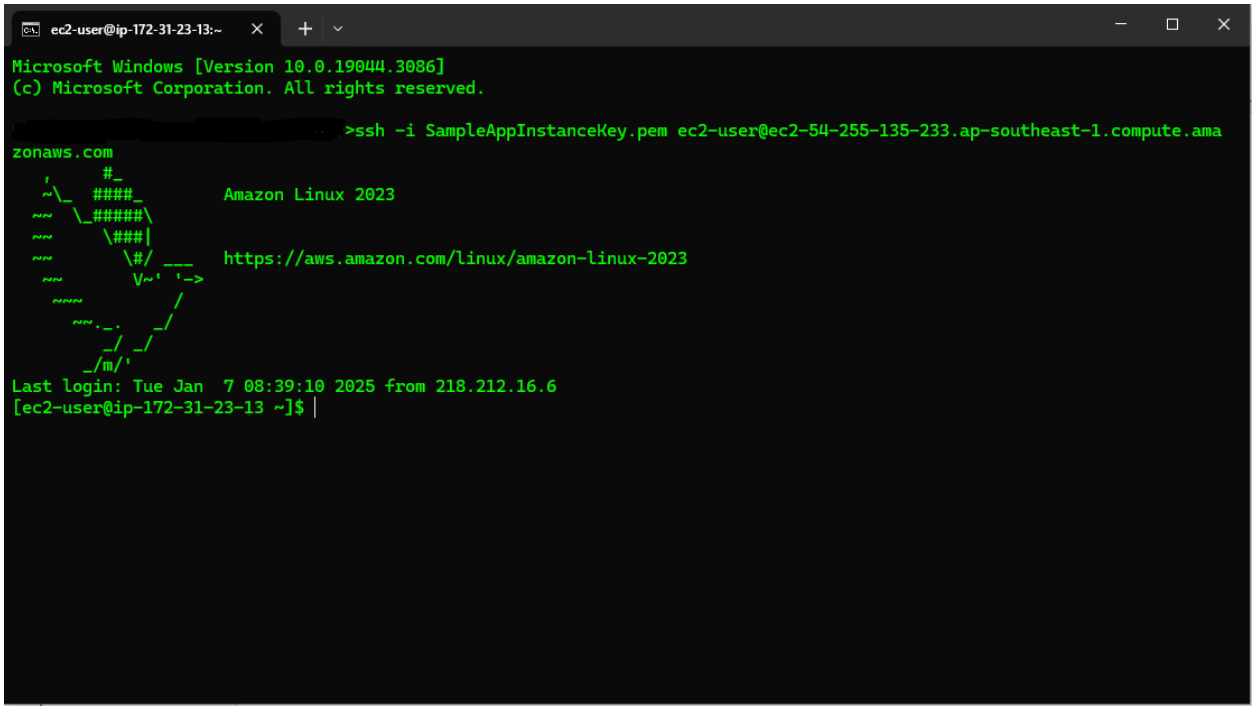
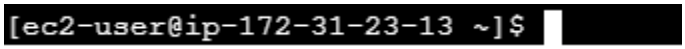
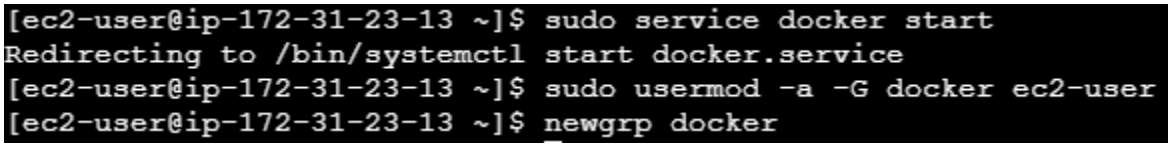
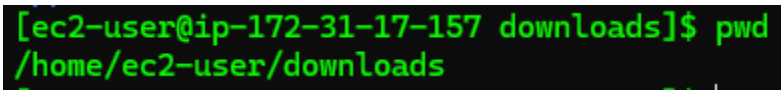
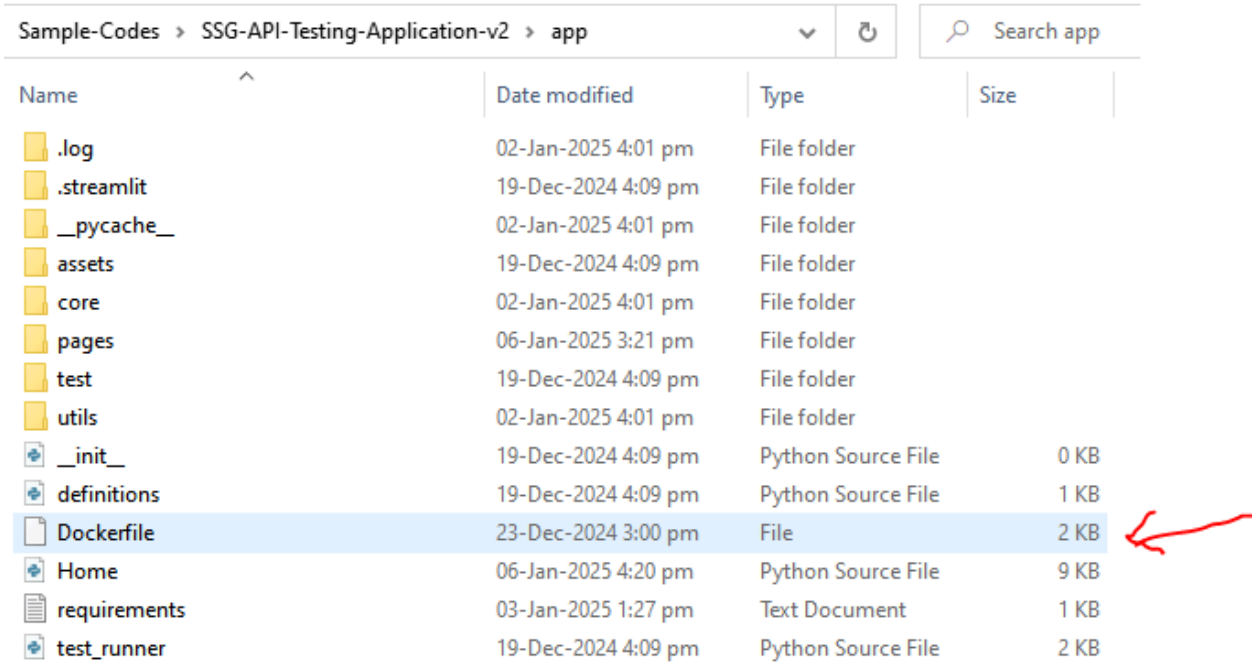
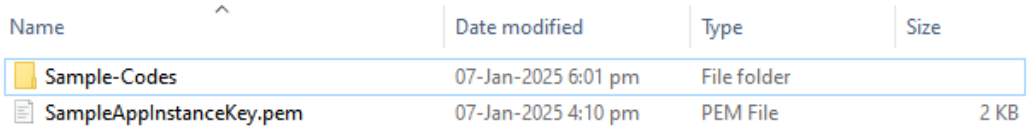
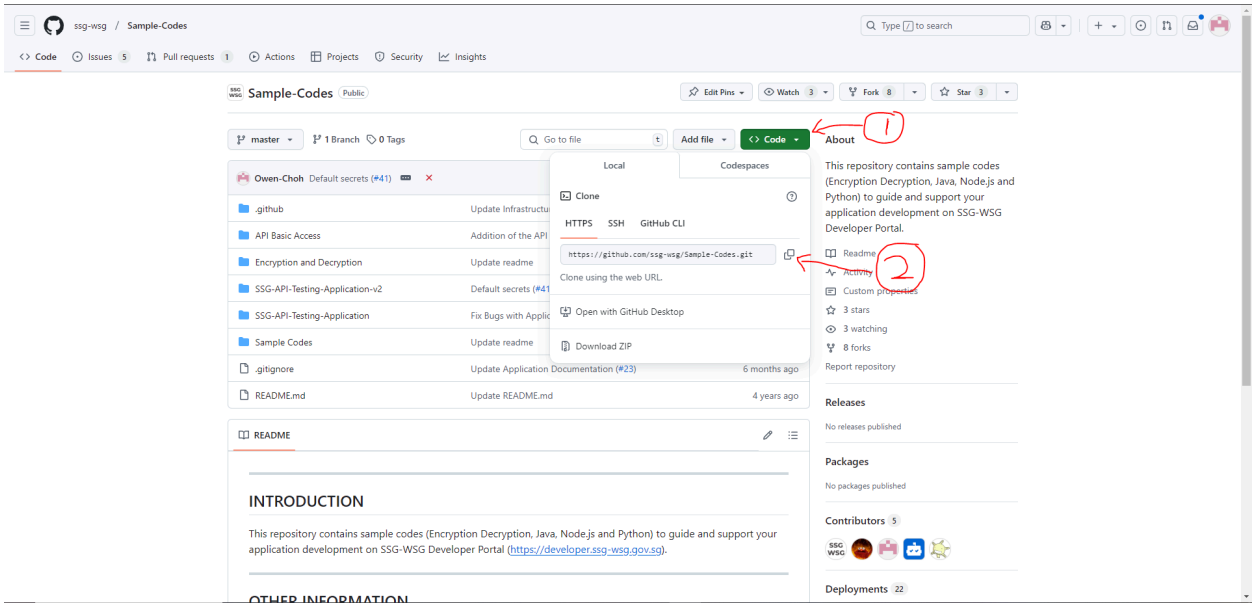


This is the screen once connected via the aws console:

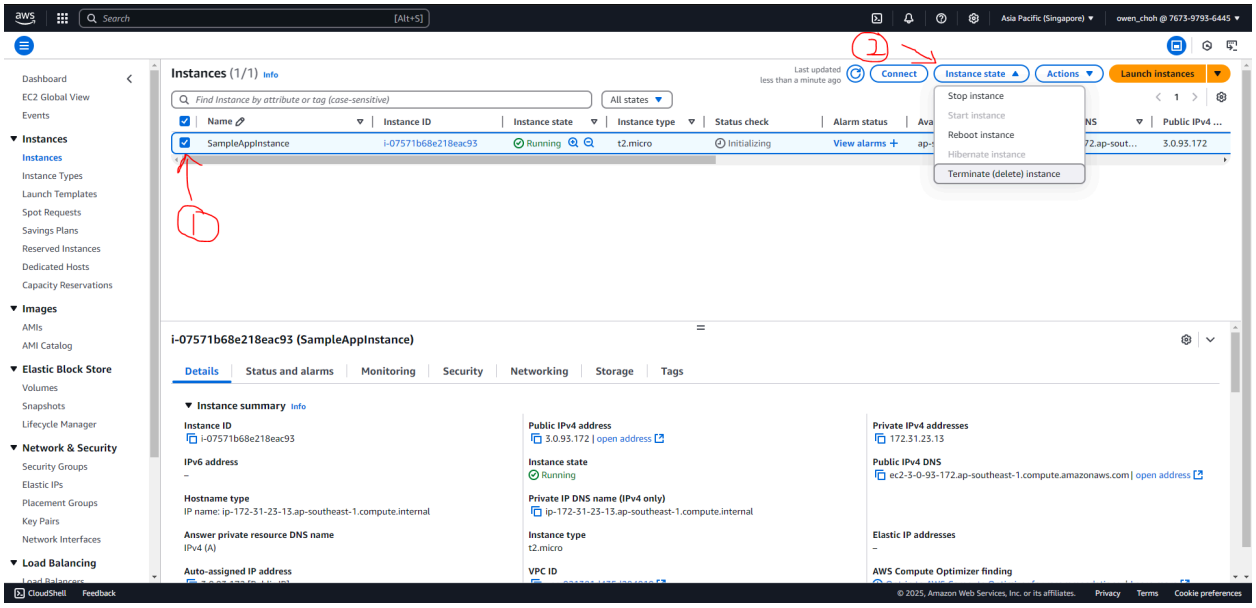


- Using the ssh command “ssh -i <relative path to the instance key> ec2-user@<dns of the instance>”
  - Type “yes” if the ssh command produces a warning about a new fingerprint
  - Refer to this page (https://superuser.com/questions/1296024/windows-ssh-permissions-for-private-key-are-too-open) if there is a warning about the key being unprotected
  - ```
    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    @         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    Permissions for 'private-key.ppk' are too open.
    It is required that your private key files are NOT accessible by others.
    This private key will be ignored.
    Load key "private-key.ppk": bad permissions
    ```

Example ssh command:

```
ssh -i SampleAppInstanceKey.pem
ec2-user@ec2-52-77-81-200.ap-southeast-1.compute.amazonaws.com
```
```



2. Once connected, type in the following commands
    - To update the instance package manager
        - sudo yum update -y
    - To install docker
        - sudo yum install -y docker

(you will be able to type once its done)



    - Once installed, start docker
        - sudo service docker start
    - Once started, give yourself permissions
        - sudo usermod -a -G docker ec2-user
        - newgrp docker



    - Check that you have permissions and docker is running
        - docker ps

Step Recap

After this step, you should have installed docker on the EC2 instance where the Sample App will run.

Step 5: Sample App Deployment

This step is to get the source code to the instance and run it using the docker service installed in the previous step.

Step 5a: Transferring the Sample App files

This sub step is to describe how to transfer the files to the EC2 instance.

1. Connect to the instance (Please refer to the previous “Step 4: EC2 Instance installation” if you need help with this)
2. (Optional) Make a new folder called “downloads” in the instance so that it is more tidy by typing the commands below.
    - Make a new directory
        - mkdir downloads
    - Navigate to the directory
        - cd downloads
    - The commands above do not produce any outputs but you can check that you are in the right directory by typing “pwd”.
    - 

3. Copy the Sample App code to the EC2 instance via one of two methods.
    a. Using scp command to transfer files from your laptop to the instance
    b. Downloading directly from github

3a. Transferring files from your laptop

This is the preferred method as there will be less unnecessary files and dependencies in the instance.

1. Download the files from github if you do not have it on your laptop
2. Navigate to the directory with the source code on your laptop (look for the location with the dockerfile as shown below)



3. Open a terminal / command prompt and send the files to the instance using the command below
    - scp -i <relative location of the instance key> -r <relative location of the source code> ec2-user@<dns of the instance>:<where to put your files on the instance>
    - It should look like the command below if the terminal is in this directory



    -
    - scp -i mykey.pem -r ./Sample-Codes/SSG-API-Testing-Application-v2/app/ ec2-user@ec2-52-77-81-200.ap-southeast-1.compute.amazonaws.com:/home/ec2-user/downloads/

4. Once it stops running, you can connect to the instance and navigate to the directory specified above to check that it is there. (screenshot below assumes you are connected and navigated to the “downloads” directory)

```
[ec2-user@ip-172-31-17-157 downloads]$ ls
app
[ec2-user@ip-172-31-17-157 downloads]$ cd app
[ec2-user@ip-172-31-17-157 app]$ ls
Dockerfile  __init__.py  assets  definitions.py  requirements.txt  test_runner.py
Home.py     __pycache__  core    pages           test              utils
[ec2-user@ip-172-31-17-157 app]$
```

5. Proceed to “Step 5b: Starting the Sample App” once this is done

3b. Downloading from github

1. Copy the link (2) in the image from the SSG repository on github (https://github.com/ssg-wsg/Sample-Codes)



2. Connect to the instance (Please refer to the previous “Step 4: EC2 Instance installation” if you need help with this step)
3. Install git using the command
    - sudo yum install git -y
4. Download the files from the link you copied earlier using the command (it may take a while since it will need to download the entire repository)
    - git clone https://github.com/ssg-wsg/Sample-Codes.git
5. Type in the following commands to check that the files are there and to navigate to the correct directory

```
[ec2-user@ip-172-31-17-157 test]$ git clone https://github.com/ssg-wsg/Sample-Codes.git
Cloning into 'Sample-Codes'...
remote: Enumerating objects: 1100, done.
remote: Counting objects: 100% (441/441), done.
remote: Compressing objects: 100% (338/338), done.
remote: Total 1100 (delta 259), reused 121 (delta 103), pack-reused 659 (from 2)
Receiving objects: 100% (1100/1100), 708.64 MiB | 17.73 MiB/s, done.
Resolving deltas: 100% (391/391), done.
Updating files: 100% (429/429), done.
[ec2-user@ip-172-31-17-157 test]$ ls
Sample-Codes
[ec2-user@ip-172-31-17-157 test]$ cd Sample-Codes/SSG-API-Testing-Application-v2/app
[ec2-user@ip-172-31-17-157 app]$ ls
Dockerfile  Home.py  __init__.py  assets  core  definitions.py  pages  requirements.txt  test  test_runner.py  utils
[ec2-user@ip-172-31-17-157 app]$
```

6. Proceed to “Step 5b: Starting the Sample App” once this is done

Step 5b: Starting the Sample App

This sub step assumes that you have:

1. Placed the secrets in the parameter store.
2. Created the required IAM roles.
3. Created an EC2 instance.
4. Transferred the Sample App files to the instance and you are in the correct directory.

Please refer to the previous steps in this document if there are missing items.

This sub step is to describe how to start the Sample App once the files are in the instance.

1. Build the container image that the Sample App will run in using the command below
    - Please note that the strings in bold are following the examples provided in the previous steps. Do make sure to amend them if your values are different else the app will not be able to retrieve the secrets from the parameter store.
    - docker build --build-arg SECRET_PATH="**/SampleApp/test/secrets/**" --build-arg SECRET_ENCRYPTION_KEY_PATH="**/SampleApp/test/secrets/encrypt**" --build-arg SECRET_CERT_PATH="**/SampleApp/test/secrets/cert**" --build-arg SECRET_KEY_PATH="**/SampleApp/test/secrets/key**" --build-arg ROLE_ARN="**arn:aws:iam::767397936445:role/SampleAppRetrieveSecretRole**" --build-arg REGION_NAME="**ap-southeast-1**" -t ssg/sample-app .
        i. SECRET_PATH="**/SampleApp/test/secrets/**" is the path where all 3 secrets are stored.

```
[ec2-user@ip-172-31-17-157 app]$ docker build --build-arg SECRET_PATH="/SampleApp/test/secrets/" --build-arg SECRET_ENCR
YPTION_KEY_PATH="/SampleApp/test/secrets/encrypt" --build-arg SECRET_CERT_PATH="/SampleApp/test/secrets/cert" --build-ar
g SECRET_KEY_PATH="/SampleApp/test/secrets/key" --build-arg ROLE_ARN="arn:aws:iam::637423663710:role/TestSecretRole" --b
uild-arg REGION_NAME="ap-southeast-1" -t ssg/sample-app .
[+] Building 1.9s (10/10) FINISHED                                                                        docker:default
 => [internal] load build definition from Dockerfile                                                               0.0s
 => => transferring dockerfile: 1.35kB                                                                             0.0s
 => [internal] load metadata for docker.io/library/python:3.12-slim                                                1.8s
 => [internal] load .dockerignore                                                                                  0.0s
 => => transferring context: 2B                                                                                    0.0s
 => [1/5] FROM docker.io/library/python:3.12-slim@sha256:10f3aaab98db50cba827d3b33a91f39dc9ec2d02ca9b85cbc5008220  0.0s
 => [internal] load build context                                                                                  0.0s
 => => transferring context: 26.62kB                                                                               0.0s
 => CACHED [2/5] WORKDIR /app                                                                                      0.0s
 => CACHED [3/5] COPY requirements.txt requirements.txt                                                            0.0s
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt                                                0.0s
 => CACHED [5/5] COPY . .                                                                                          0.0s
 => exporting to image                                                                                             0.0s
 => => exporting layers                                                                                            0.0s
 => => writing image sha256:7e63b558b859d63c97b74a1293621b4c90cfc5b474b2c94b15e9dbbe35cc0347                       0.0s
 => => naming to docker.io/ssg/sample-app                                                                          0.0s
[ec2-user@ip-172-31-17-157 app]$
```

The screenshot is for reference only. It takes about a minute to download items and complete building if this is your first time running the command.

2. Once your image is built, run the command below to start the container
    - docker run -d --rm -p 80:80 --name sampleapp ssg/sample-app
    - The value in magenta must be the same as the build command above.

- The value in orange is the name of the container that the Sample App is running in.
- The output of the command does not matter
- Please refer to https://docs.docker.com/reference/cli/docker/container/run/ for detailed explanation of the command.

```
[ec2-user@ip-172-31-17-157 app]$ docker run -d --rm -p 80:80 --name sampleapp ssg/sample-app
b1f17c4ba9e90fd092205373ad134a420bd77eb094c0579429e76a5a51febb6f
[ec2-user@ip-172-31-17-157 app]$
```

Step Recap

After this step, you should have:

- Transferred the Sample App files to the instance
- Started the Sample App on the instance
- Able to access the Sample App via the "**Public IPv4 DNS**" of the instance (e.g. http://ec2-52-77-81-200.ap-southeast-1.compute.amazonaws.com)
    - You can think of a "**Public IPv4 DNS**" as a website link.

Additional Notes

To stop the container that is running the app, run the command below

- Please note that you can **skip this step** if you want to stop the instance as this will only stop the Sample App.
- docker stop sampleapp
- ```
  [ec2-user@ip-172-31-17-157 app]$ docker stop sampleapp
  sampleapp
  ```

Associate an elastic ip address to your EC2

- Please note that this step is **not necessary** for the Sample App to function and only helps to stop the "**Public IPv4 DNS**" of the instance from changing.
- If you want your EC2 instance to use an IP address you have in your AWS account, please refer to the steps in the link below.
- https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/working-with-eips.html#using-instance-addressing-eips-associating

Disassociate the elastic ip address to your EC2

- Please note that you can **skip this step** if you want to stop the instance as this will only "unlink" the IP address from the EC2 and change the "**Public IPv4 DNS**" of the instance.
- If you associate an IP address to the EC2 instance and you want to disassociate it, please refer to the steps in the link below.
- https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/working-with-eips.html#using-instance-addressing-eips-associating-different
- If you terminate an instance, AWS will automatically disassociate the IP address from the instance and **will not** release your IP address.

To stop the EC2 instance

- Choose the state that you require from this dropdown on the EC2 dashboard
- "**Stop instance**" will allow you to "**Start instance**" later but will still incur charges from:
  - The instance storage - the "hard disk" of the instance
  - You may still incur charges for the ip address if you reserved one under the "**Elastic IPs**" tab on the left. Otherwise, the "**Public IPv4 DNS**" of the instance will change after you "**Start instance**" later.
  ```

- “**Terminate (delete) instance**” will delete the instance and storage forever (It may remain on the list for a while after deletion). You will not be able to start this instance again and will need to install and deploy the app again. You may still incur charges for the ip address if you reserved one under the “**Elastic IPs**” tab on the left.



Delete parameters from parameter store

- It is currently under the “**always free**” tier to store “**standard**” tier secrets in the parameter store but you can choose to delete them if they are not needed.