**WheelsOnDemand-github-actions-terraform-ecs**

1. **Create a GitHub repository**

I’ve named mine “WheelsOnDemand-github-actions-terraform-ecs”

1. **Update the .gitignore file**

Copy and paste the .gitignore file from the shared repository into it

git add .

git commit -m “update gitignore file”

git push

1. **Add the Terraform code into the repository**

Download the Terraform code: <https://github.com/Silas-cloudspace/cicd-projects>

Unzip it and paste the folder into the “WheelsOnDemand-github-actions-terraform-ecs” folder in your local computer

git add .

git commit -m “add iac files”

git push

1. **Create the remote backend**

Now we will create an S3 bucket to store the Terraform state and a DynamoDB table to lock the Terraform state.

Navigate to remote\_backend folder

cd remote\_backend

terraform init

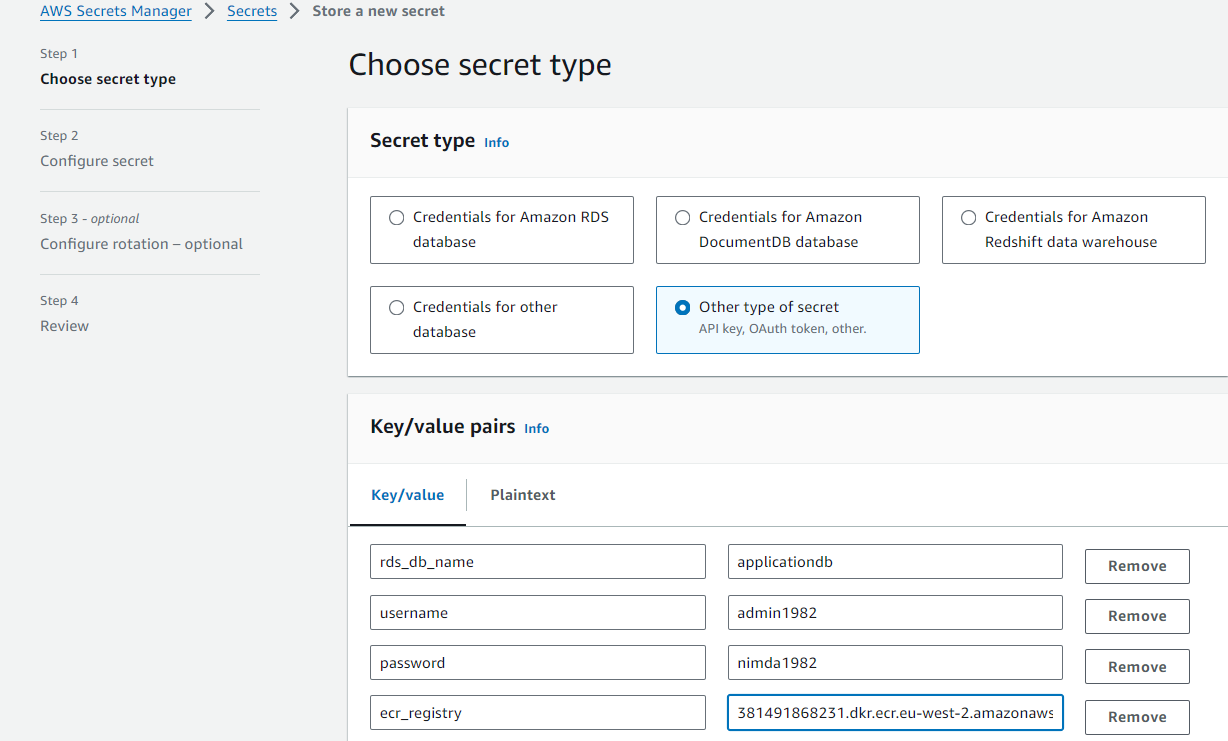
terraform apply

1. **Create Secrets in AWS Secrets Manager**

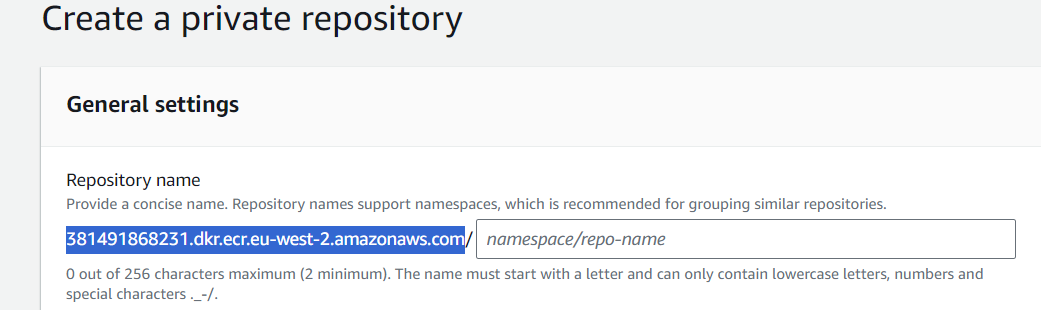
We will now add the value for our RDS database name, username and password, and also our ECR registry as secrets in Secrets Manager.

Go to AWS console and search for Secrets Manager

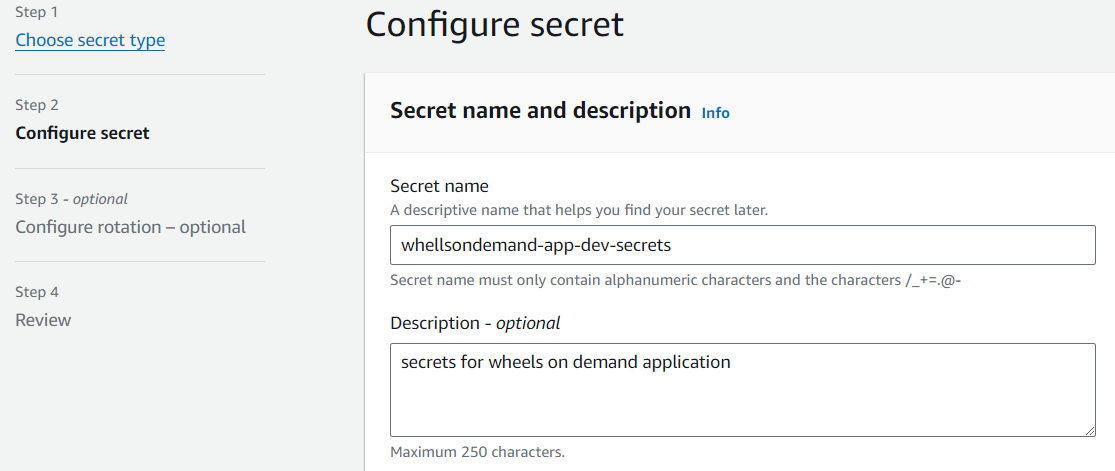
Click on “Store a new secret”



In order to get your Elastic Container Registry, open a new AWS tab, go to ECR and click on “Create repository”



Copy the ECR value and paste it on secrets manager.



You can click next on the following steps.

1. **Register a Domain name**

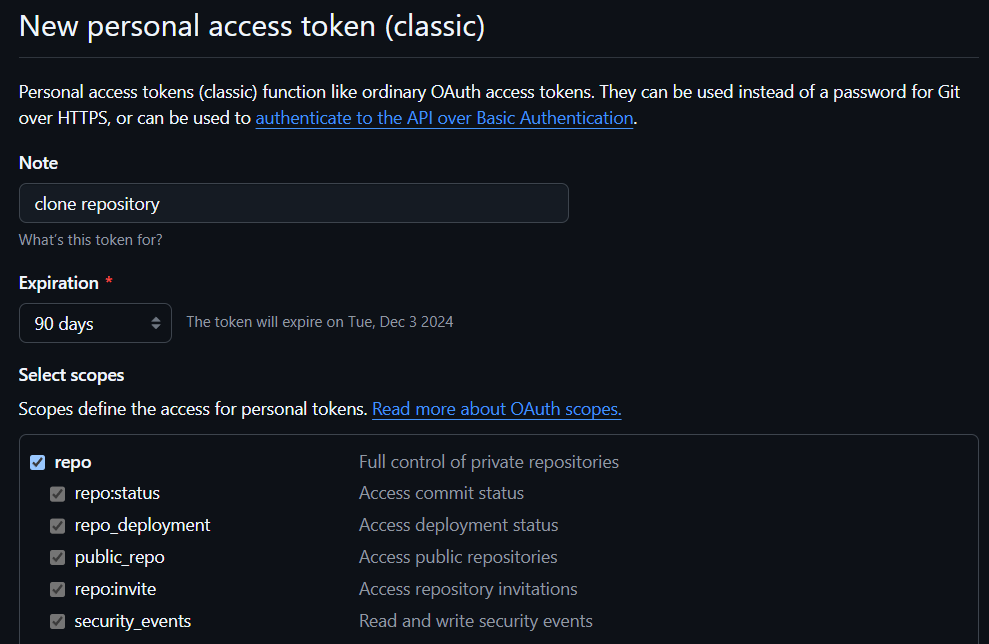
Go to AWS Route 53 and create a new domain name for yourself. It will cost you around 14 dollars.

1. **Create a personal access token on github**

This token will be used by docker to clone the application codes repository when we build our docker image

Github -> select your profile -> settings -> Developer settings -> Personal access tokens -> Tokens (classic) - > Generate new token -> Generate new token classic

Edit it as you see in the following example:



Remember to copy your personal access token and save it anywhere

ghp\_ZmnBkmyj4cYiIEeB8t8IvupX2kWIlA1pfhHM

1. **Create GitHub repository Secrets**

Now we will create the repository secrets that the GitHub Action job need to build our cicd pipeline for this project.

Go to your GitHub repository

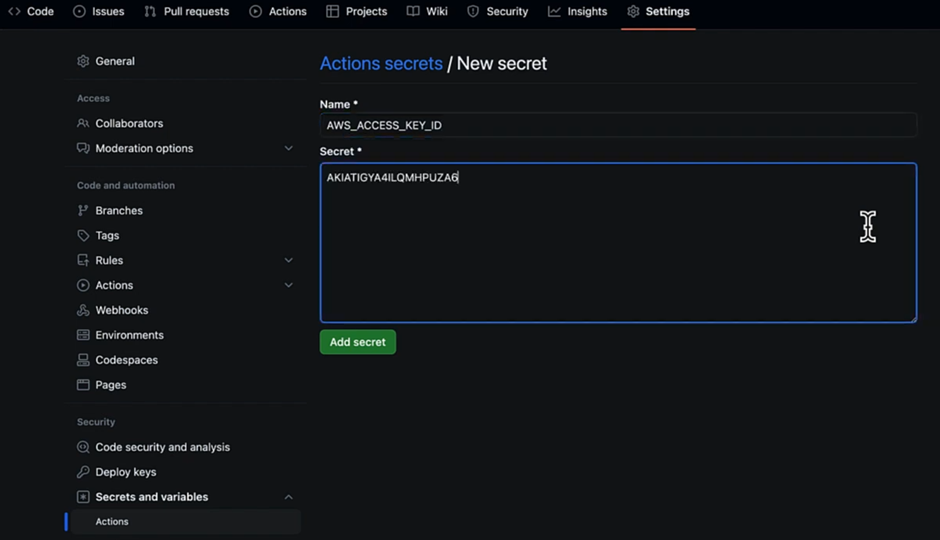
Click on settings

Navigate to “Secrets and variables”

Choose “actions”

Click on “New repository secret”

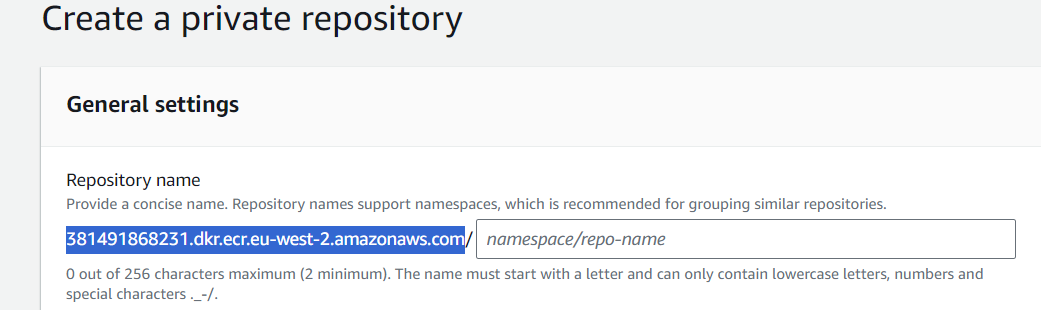
Add 7 secrets:



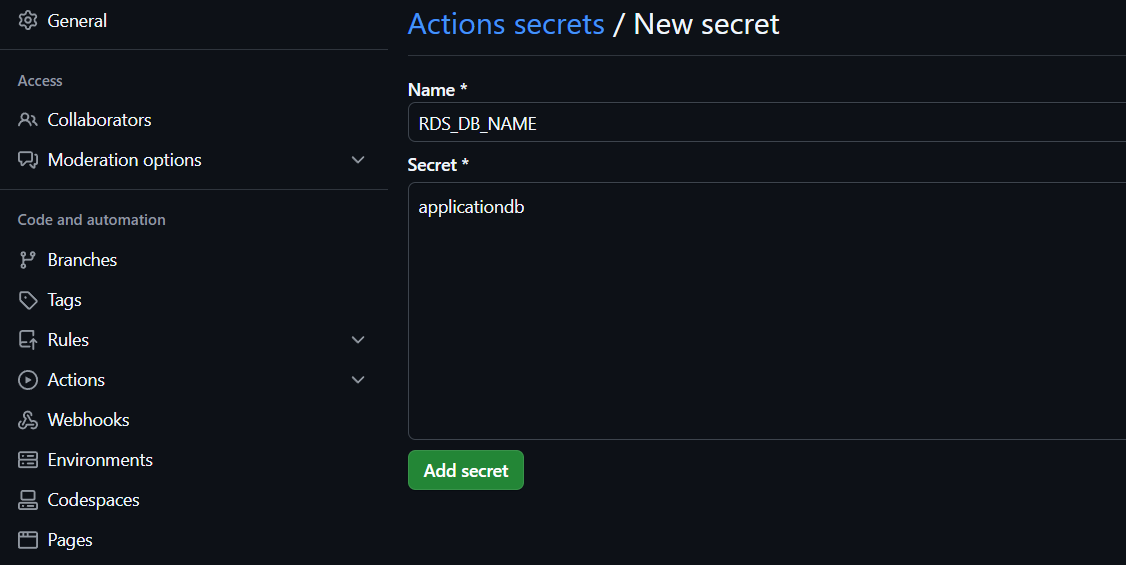
Above you can find an example

* AWS\_ACCESS\_KEY\_ID - “Your AWS access key id”
* AWS\_SECRET\_ACCESS\_KEY – “Your secret AWS access key”
* ECR\_REGISTRY –

In order to get your Elastic Container Registry, open a new AWS tab, go to ECR and click on “Create repository”



* PERSONAL\_ACCESS\_TOKEN – “The personal access token we created on point VII”
* RDS\_DB\_NAME – Go to AWS console > Secrets Manager > select the secret we create before > Under secret value, click on “retrieve secret value”. Paste the rds\_db\_name “applicationdb” as the secret on GitHub.



* RDS\_DB\_USERNAME - RDS\_DB\_NAME – Go to AWS console > Secrets Manager > select the secret we create before > Under secret value, click on “retrieve secret value”. Paste the rds username you choose as the secret on GitHub.
* RDS\_DB\_PASSWORD – RDS\_DB\_NAME – Go to AWS console > Secrets Manager > select the secret we create before > Under secret value, click on “retrieve secret value”. Paste the rds password you choose as the secret on GitHub.

1. **Create the GitHub Action Workflow file**

Navigate to “WheelsOnDemand-github-actions-terraform-ecs” on VS Code

mkdir -p .github/workflows

cd .github/workflows

touch deploy\_pipeline.yml

1. **Create 3 GitHub Actions jobs**

We will now create 3 GitHub Actions jobs:

* Configure AWS credentials
* Build AWS infrastructure
* Create ECR repository

Copy the content from the shared repo into your deploy\_pipeline.yml

Comment everything that is after line 170 by pressing Shift + Alt + A

git add.

git commit -m “added github actions jobs”

git push

Check the pipeline under the “Actions” tab on your GitHub repo

1. **Create a Self-Hosted Runner**

For the next job in our pipeline, we will star a self-hosted ec2 runner in the private subnet. We will use this runner for two things in our pipeline:

* First, we will use the runner to build our docker image and push the docker image to the amazon ecr repository we created previously.
* Then, we will also use this runner to run our database migration with flyway.

The reason why we are using a self-hosted runner to complete these jobs is because launching an ec2 runner in our private subnet will allow the runner to easily access the resources in our private subnet.

In this project we want to migrate our data into the RDS database so, by launching our EC2 in the private subnet, it allows that EC2 runner to easily connect to the RDS in the private subnet and migrate our data into it with flyway.

Once we have successfully migrated our data, we will terminate the ec2 runner immediately.

If we use the GitHub hosted runner, there wouldn’t be a way for that runner to connect to our RDS instance in the private subnet because, as you know, any resources we put in the private subnet may require additional configuration to access resources outside our private subnet.

Steps to create this action: <https://github.com/machulav/ec2-github-runner>

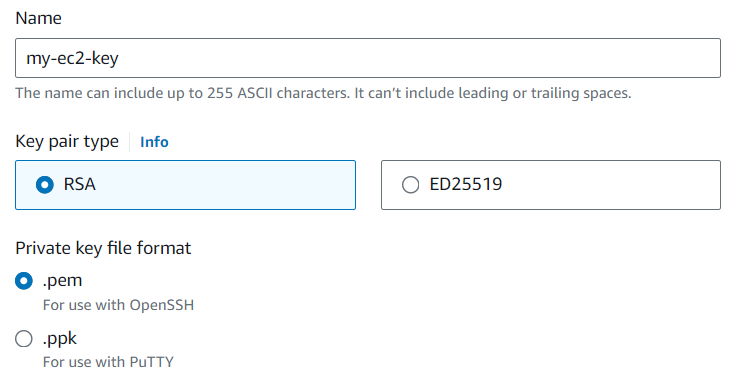
To summary we will need to:

1. Create AWS access keys Pairs
2. Create a GitHub personal access token (we have it)
3. Prepare an EC2 image
4. Use the EC2 instance to create an AMI
5. Terminate the EC2 instance

Once we create our job to start our self-hosted runner, the job will use the AMI we created to start our self-hosted EC2 runner.

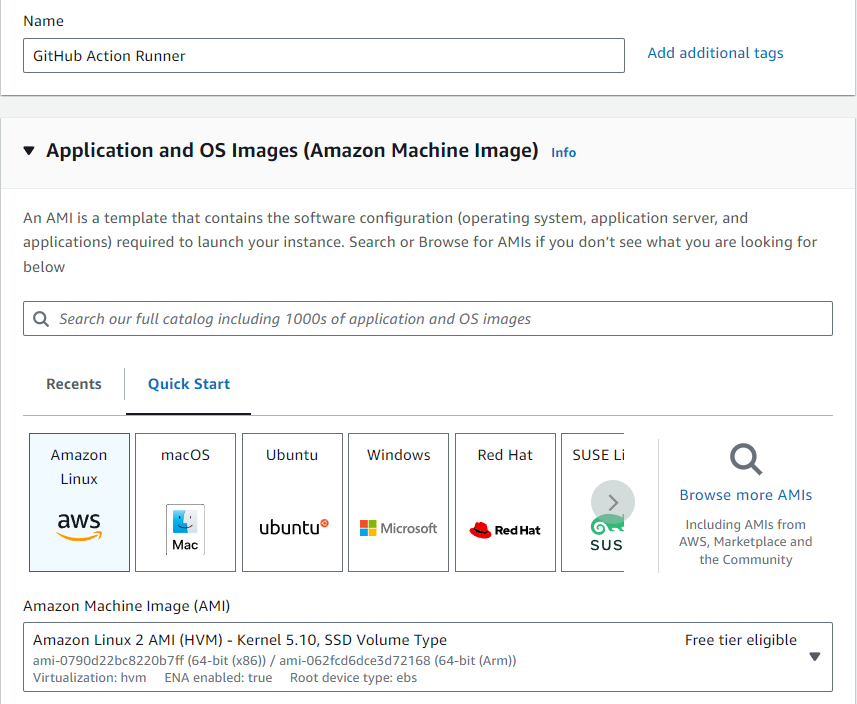
1. **Create AWS access keys Pairs**

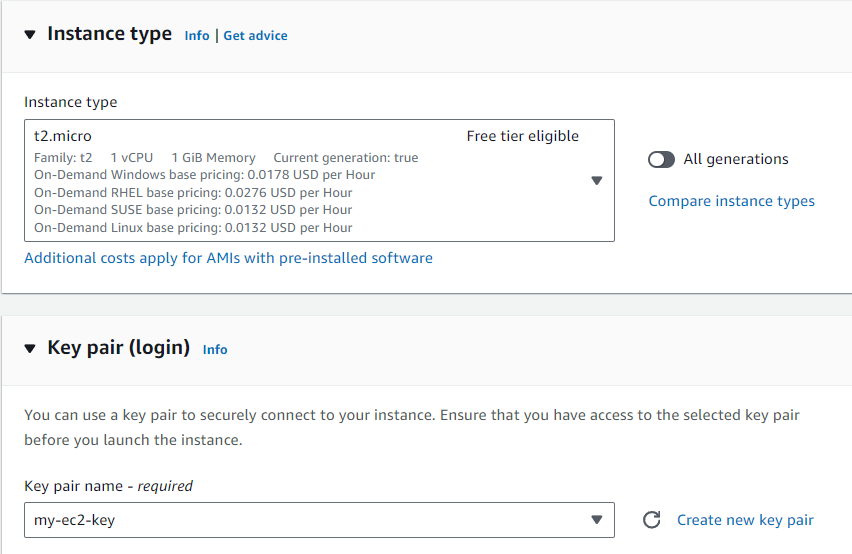
Go to AWS Management Console > EC2 > Key Pairs > Create Key Pair

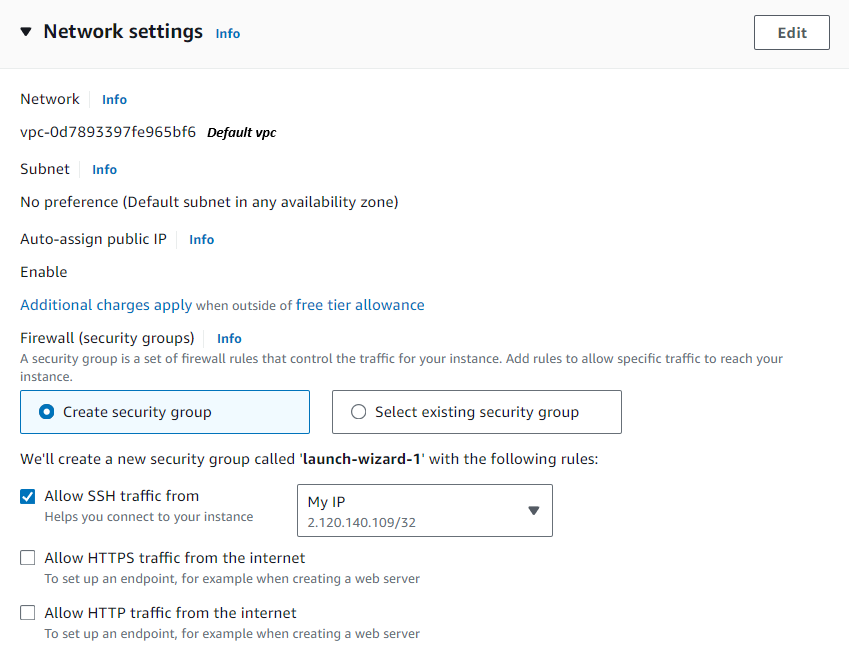


1. **Launch an EC2 instance in a public subnet**

Go to AWS Management Console and create a new EC2 instance:

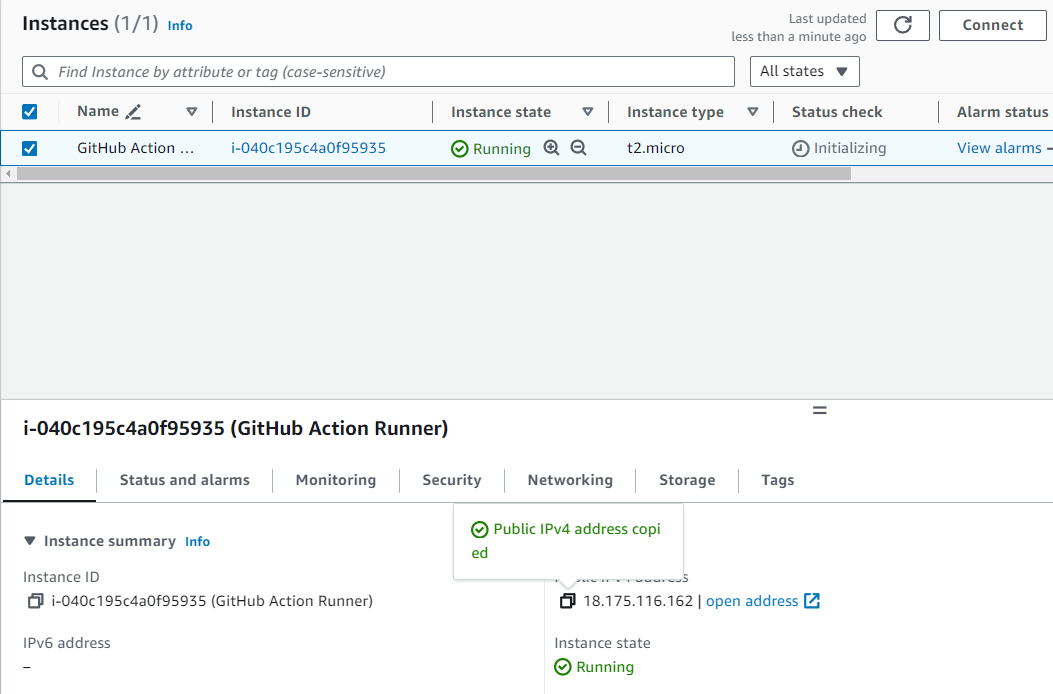






1. **SSHing into an EC2 Instance**

Copy the Public IPv4 address of the EC2 instance:



Open a terminal in the same directory where you have stored your Key Pair and run the following:

ssh -i my-ec2-key.pem ec2-user@<public ipv4 address from ec2 instance>

1. **Install Docker and Git on the EC2 Instance**

Paste, copy, and run the following commands in your ec2 instance:

sudo yum update -y && \

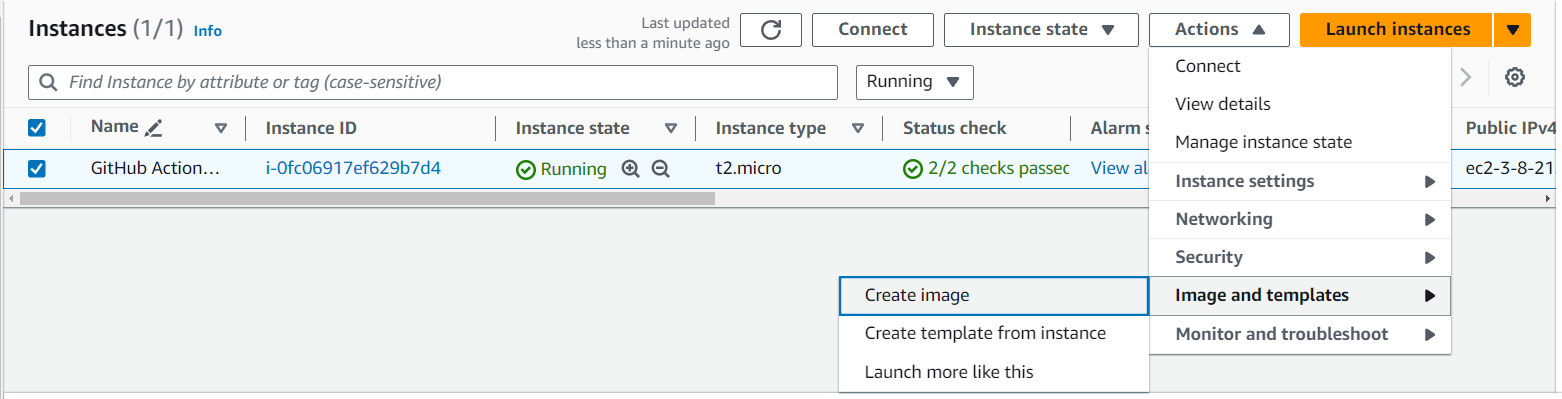
sudo yum install docker -y && \

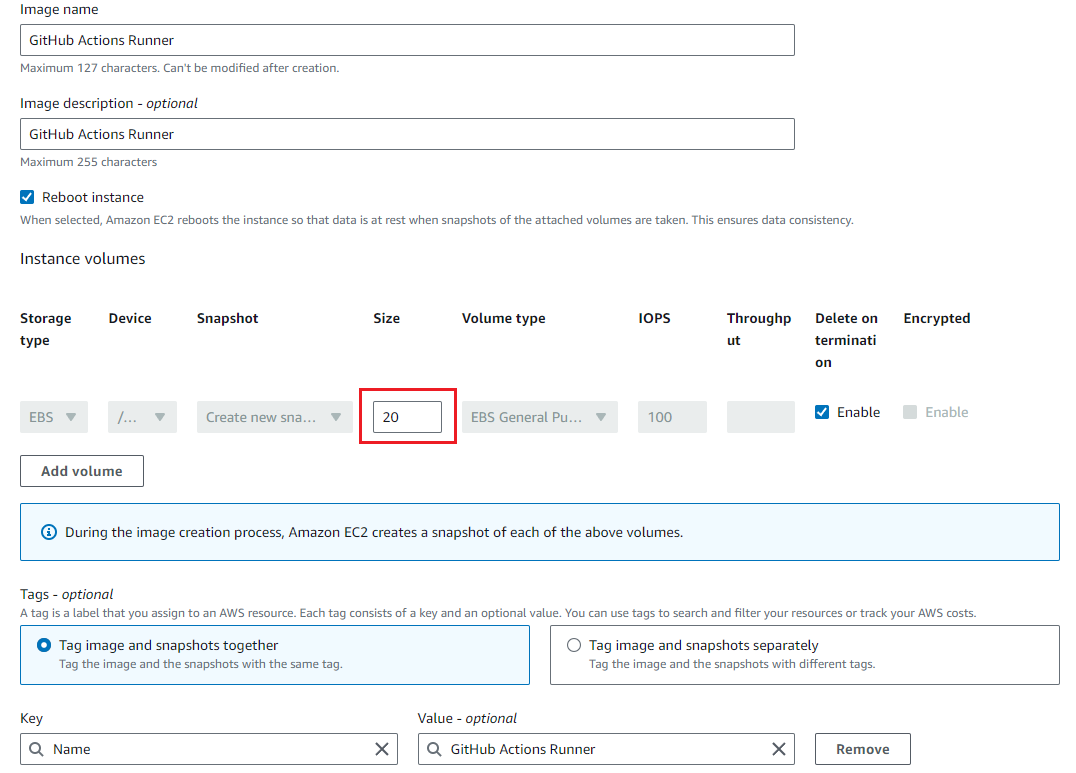
sudo yum install git -y && \

sudo yum install libicu -y && \

sudo systemctl enable docker

1. **Create an AMI and Terminate the EC2 Instance**





Wait for the status to become available and after that, confirm if a snapshot of the AMI was created on Elastic Block Store > Snapshots

Terminate your EC2 Instance

1. **Create a GitHub Actions Job to Start a Self-Hosted Runner**

Remove the comments on the “deploy\_pipeline.yml” file until line 209

git add.

git commit -m “start ec2 runner”

git push

1. **Create a GitHub Actions Job to Build a Dicker Image**

Before creating the job to build the Docker Image, we need to complete these steps first:

1. Set up a repository to store the application code
2. Add the application code to the repository
3. Create a Docker file
4. Create the appserviceprovider.php file

**Create a private repository to store the application codes**

Go to GitHub and create a new repository.

I’ve named mine “application-codes-autorentify-project”

1. Clone the repo to your local computer
2. Download the required file from this link: <https://github.com/Silas-cloudspace/application-codes-autorentify-project>
3. Add it to the repository folder in your local machine
4. Open the repository “application-codes-autorentify-project” on VS code
5. Push it to GitHub “application-codes-autorentify-project”

**Create the Docker file**

Navigate to “WheelsOnDemand-github-actions-terraform-ecs” in VS Code

Create a new file: touch Dockerfile