CircleCi Lab 2
Using AWS

Done by: Aseel Al-Khlouf (Malkawi)

Abstract

This report represents the process I went through to deploy a nodejs app in a docker image then run it on AWS EC2 using CircleCi as a CI/CD tool.

Contents

Table of Figures

1. Configuring the EC2

Creation

First, I went to AWS EC2 service and created an EC2 instance. I used ssh keys and saved a copy named “ec2key.pem” of the private key to access the machine later.

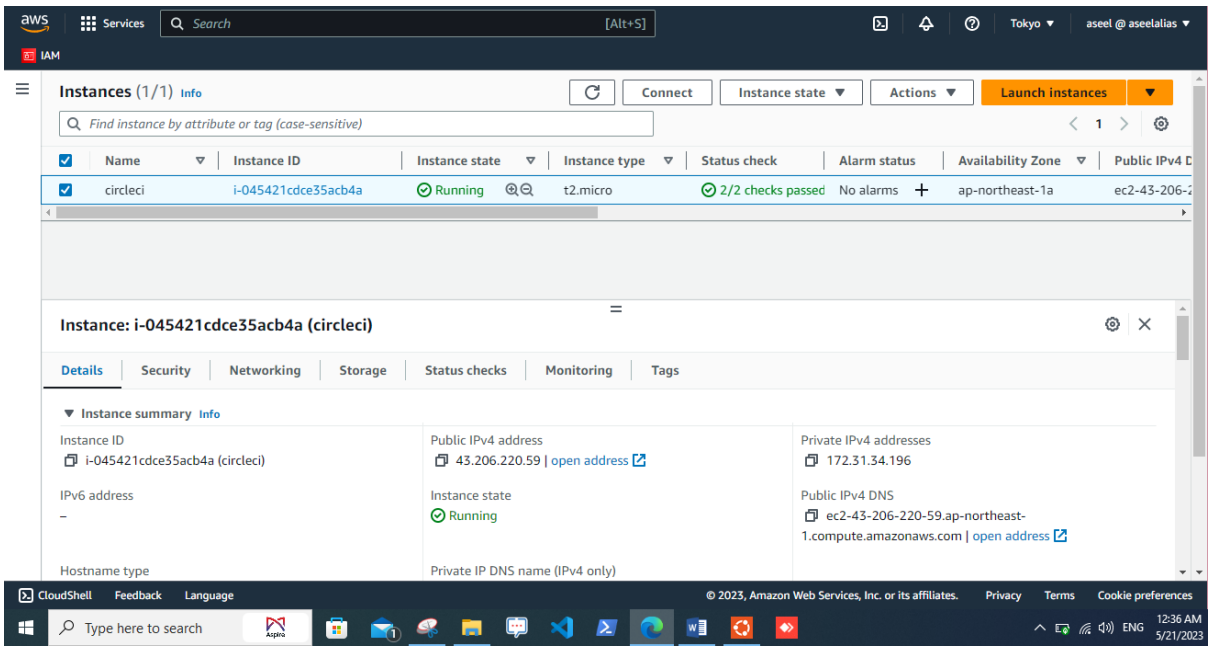

Figure 1: EC2 instance created

Accessing the machine

I opened an Ubuntu terminal in the same private key path and used the following commands:

```
chmond 400 ec2key.pem
ssh -i "ec2key.pem" ubuntu@ec2-43-206-220-59.ap-northeast-1.compute.amazonaws.com
```

After that, I was able to access the machine.

```
aseel@SOS-AALKHLOUF:~$ ssh -i "ec2key.pem" ubuntu@ec2-43-206-220-59.ap-northeast-1.compute.amazonaws.com
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-1025-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sat May 20 21:54:20 UTC 2023

  System load:  0.0               Processes:               107
  Usage of /:   52.2% of 7.57GB   Users logged in:         1
  Memory usage: 37%               IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                IPv4 address for eth0:    172.31.34.196

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

7 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm


Last login: Sat May 20 21:11:54 2023 from 213.139.53.176
ubuntu@ip-172-31-34-196:~$
```

Figure 2: accessing the EC2 machine

Downloading the necessary software

I downloaded docker using the steps from the docker docs. I also used a few other commands to download aws cli and be able to configure aws.

```
sudo apt install unzip
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
sudo apt install awscli
aws configure
aws ecr get-login-password --region ap-northeast-1 | sudo docker login --username AWS --password-stdin 899112647471.dkr.ecr.ap-northeast-1.amazonaws.com
sudo apt update
```

```
ubuntu@ip-172-31-34-196:~$ aws configure
AWS Access Key ID [****************WFF4]:
AWS Secret Access Key [****************VwYs]:
Default region name [ap-northeast-1]:
Default output format [json]:
ubuntu@ip-172-31-34-196:~$ aws ecr get-login-password --region ap-northeast-1 | sudo docker login --username AWS --passw
ord-stdin 899112647471.dkr.ecr.ap-northeast-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
ubuntu@ip-172-31-34-196:~$
```

Figure 3: configuring AWS

Editing inbound rules

I edited the EC2 inbound rules, so it can accept traffic on all TCP ports to run the docker container successfully later.

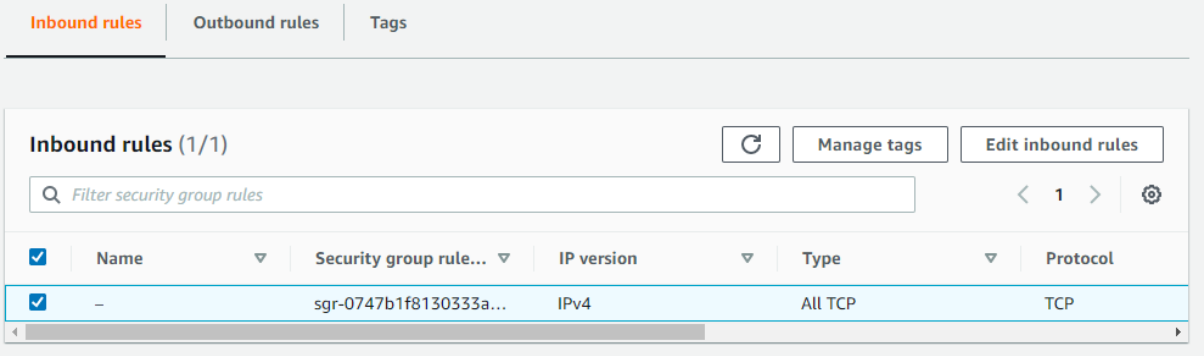

Figure 4: inbound rules

Creating a policy to attach to IAM role

To allow the EC2 to push and pull images on an ECR, I had to create a policy for that, then attach it to an IAM role. It's named AllowPushPull.

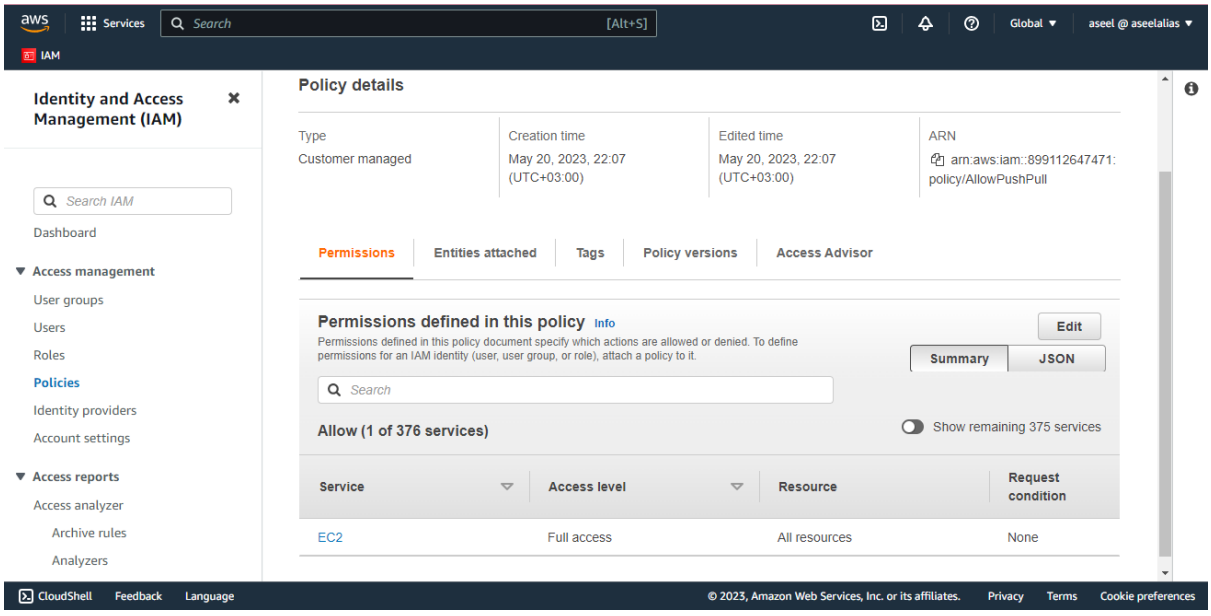

Figure 5: creating a policy

Attaching IAM role to EC2 to push and pull images on the ECR

I created a role and gave it the policy I had just created. It's named ec2.

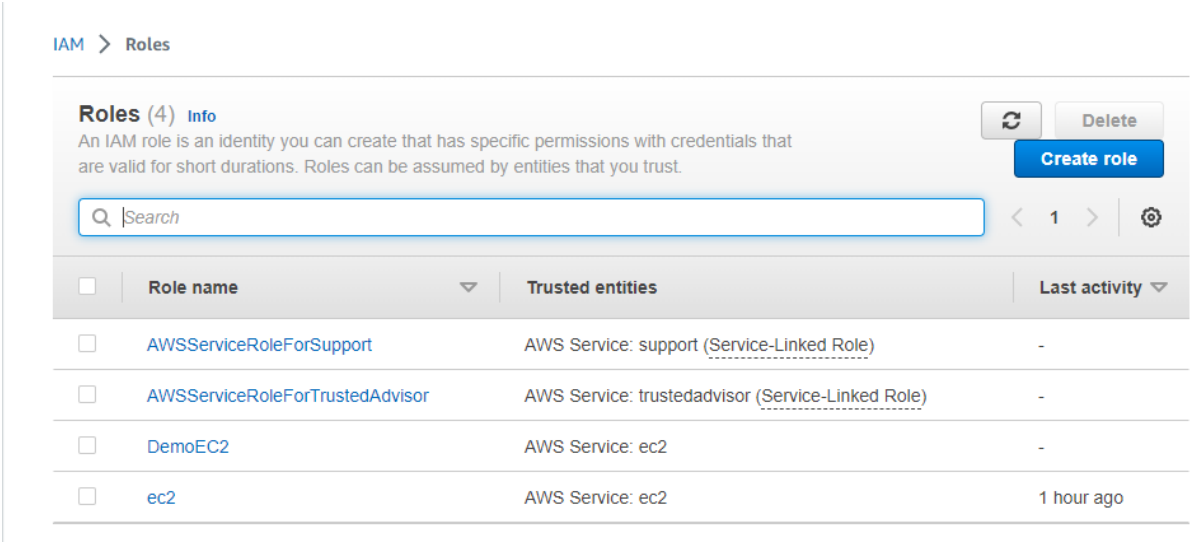

Figure 6: creating an IAM role

Lastly, I modified the IAM role for the EC2 instance from Actions > security > modify IAM role. I chose the role I had just created and hit update.

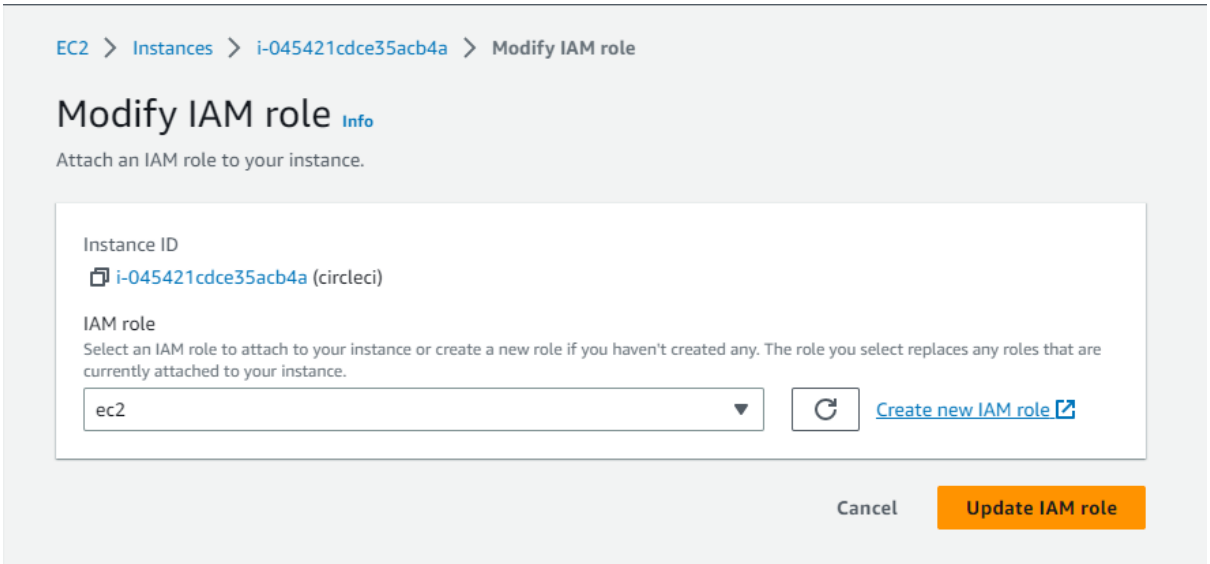

Figure 7: modifying IAM roles for the EC2

2. Creating the ECR

ECR stands for elastic container registry, which is a type of docker registry to store and manage docker images. I created a private ECR named circleci to push the image to it later.

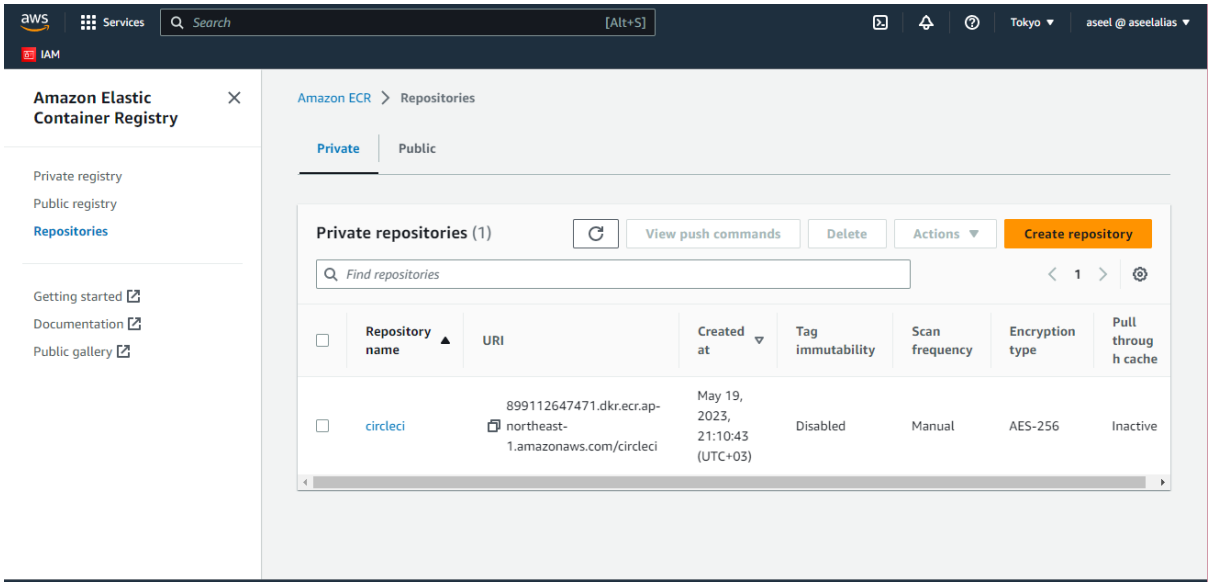

Figure 8: ECR creation

3. Setting up the GitHub repo and creating config.yml

I created the repo from the nodeapp then added the ".circleci" directory. The config file is shown below:

```
version: 2.1
orbs:
  aws-ecr: circleci/aws-ecr@8.2.1
jobs:
  test:
    docker:
      - image: docker:stable
    steps:
      - checkout
      - setup_remote_docker:
          version: 20.10.14
          docker_layer_caching: true
  deploy:
    docker:
      - image: docker:stable

    steps:
      - run:
          name: ssh ec2 and run docker
          command: |
            ssh -o StrictHostKeyChecking=no -i $HOME/.ssh/aws ubuntu@ec2-43-206-220-59.ap-northeast-1.compute.amazonaws.com " aws ecr get-login-password
```

```
--region ap-northeast-1 | docker login --username AWS --password-stdin
899112647471.dkr.ecr.ap-northeast-1.amazonaws.com
            sudo docker run -d -p 3000:3000 --name nodeapp
899112647471.dkr.ecr.ap-northeast-1.amazonaws.com/circleci:lts"
workflows:
  version: 2
  build:
    jobs:
      - test:
          context: Dockerhub
      - aws-ecr/build-and-push-image:
          repo: circleci
          tag: lts
          dockerfile: Dockerfile
          path: .
          registry-id: AWS_ECR_REGISTRY_ID
          requires:
            - test
      - deploy:
          requires:
            - aws-ecr/build-and-push-image
```

Figure 9: config.yml

Lastly, I committed the changes.

4. Setting up the project on CircleCi

I went to the main screen in CircleCi then clicked set up project. I was directed to the screen that displayed the pipelines. There, I clicked project settings and added environment variables and ssh keys.

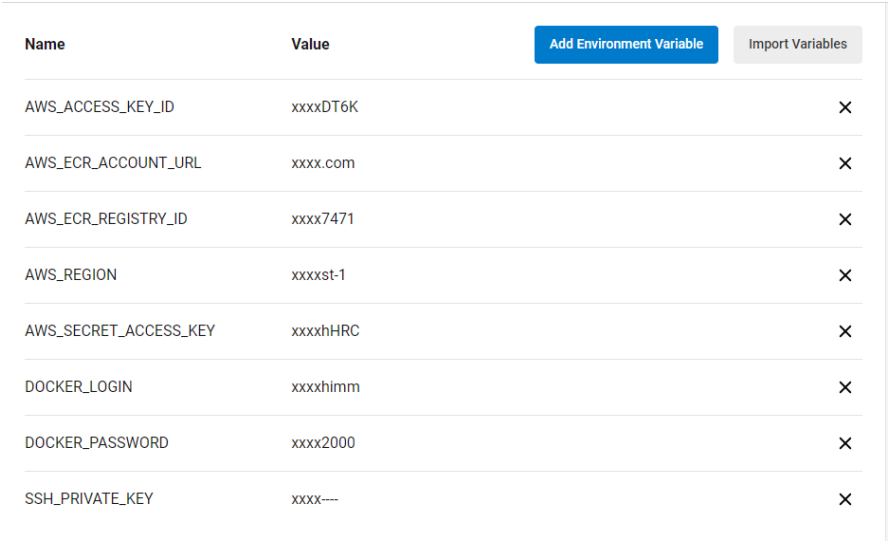

Figure 10: creating env vars

AWS_ACCESS_KEY_ID & AWS_SECRET_ACCESS_KEY

I brought those environment variables from the IAM user console. I made sure to save the secret key because I'd never see it again.

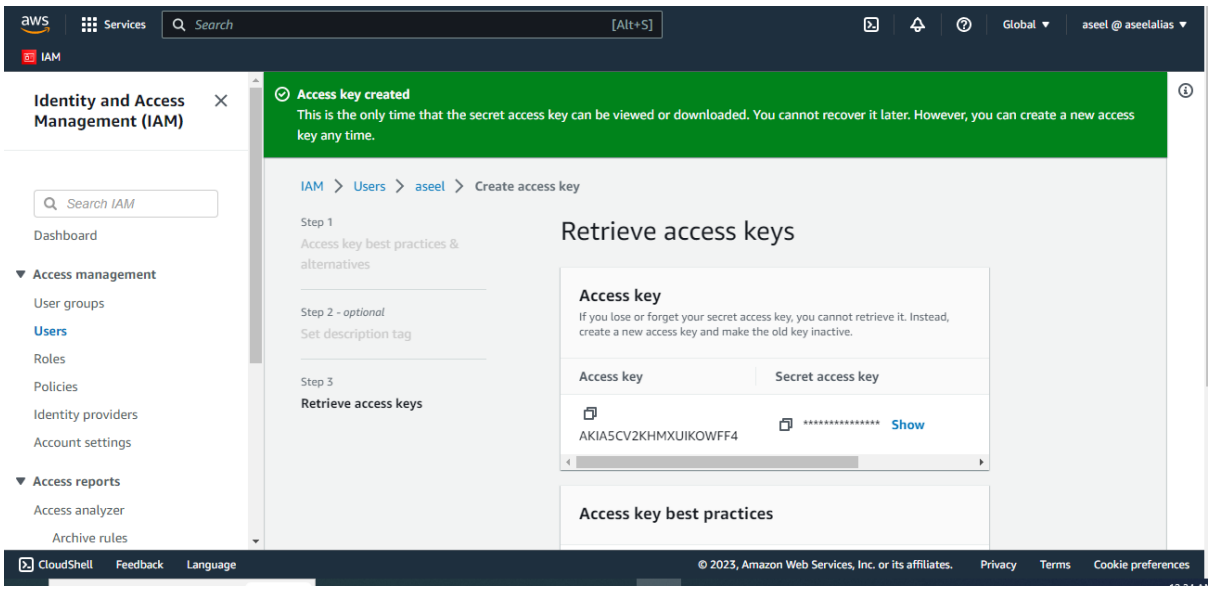

Figure 11: access key & secret access key

SSH_PRIVATE_KEY

This is exactly the same from the ec2key.pem file that I saved when creating the EC2.

AWS_ECR_REGISTRY_ID & AWS_ECR_REGISTRY_URL

Those are from the main screen of the ECR creation.

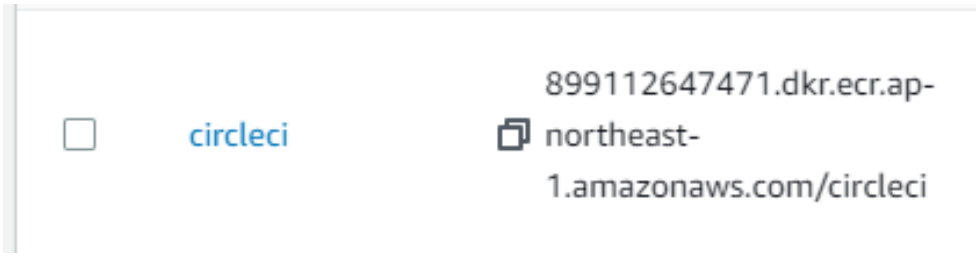

Figure 12: ECR ID & URL

AWS_REGION

Basically the region from the bar on top of the screen.

Adding the EC2 SSH key

I also added the ssh key from ec2key.pem to the project ssh keys.

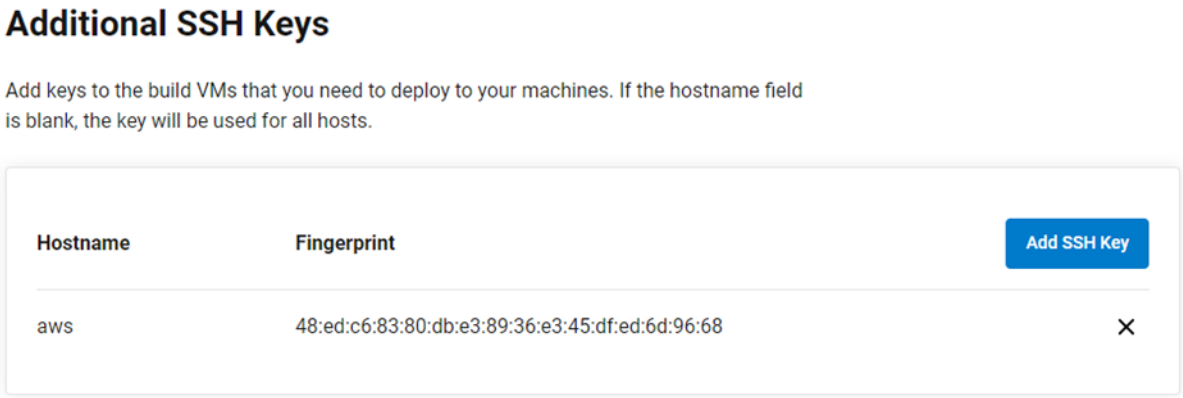

Figure 13: EC2 SSH key

5. Running the pipeline

After everything is set, I ran the pipeline and got a successful result.

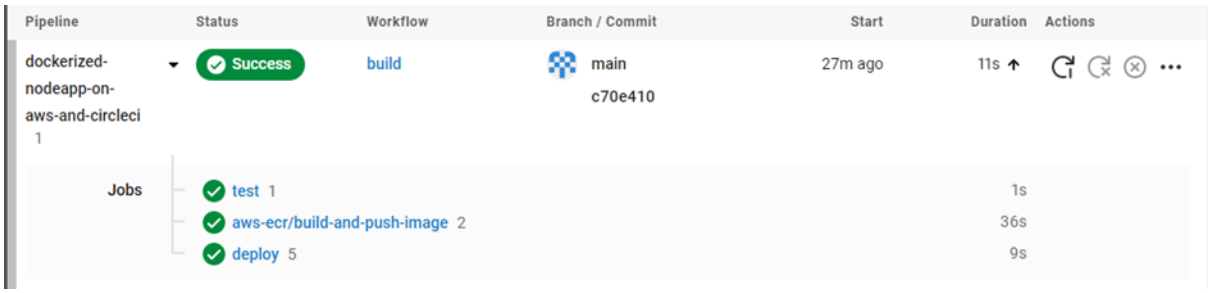

Figure 14: the pipeline ran successfully

6. Checking up the running container on the EC2

To verify, here's the running container:

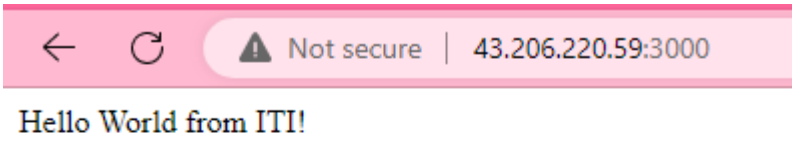

Figure 15: the container