AWS CodePipeline

Instruction

By @Chihiro-001

Scenario

- A development team manages their application in their GitHub repository. Whenever they release their application, they do manual build and deployment checks. The release frequency is expected to increase, and so the team wants to reduce manual operations and tedious tasks.
- The team wants to automate the release pipelines for fast and reliable application and infrastructure updates. They need build, test, and deploy phases for every commit pushed.
- The team wants to use a container service as a deployment destination for consistency across environments on the user side.

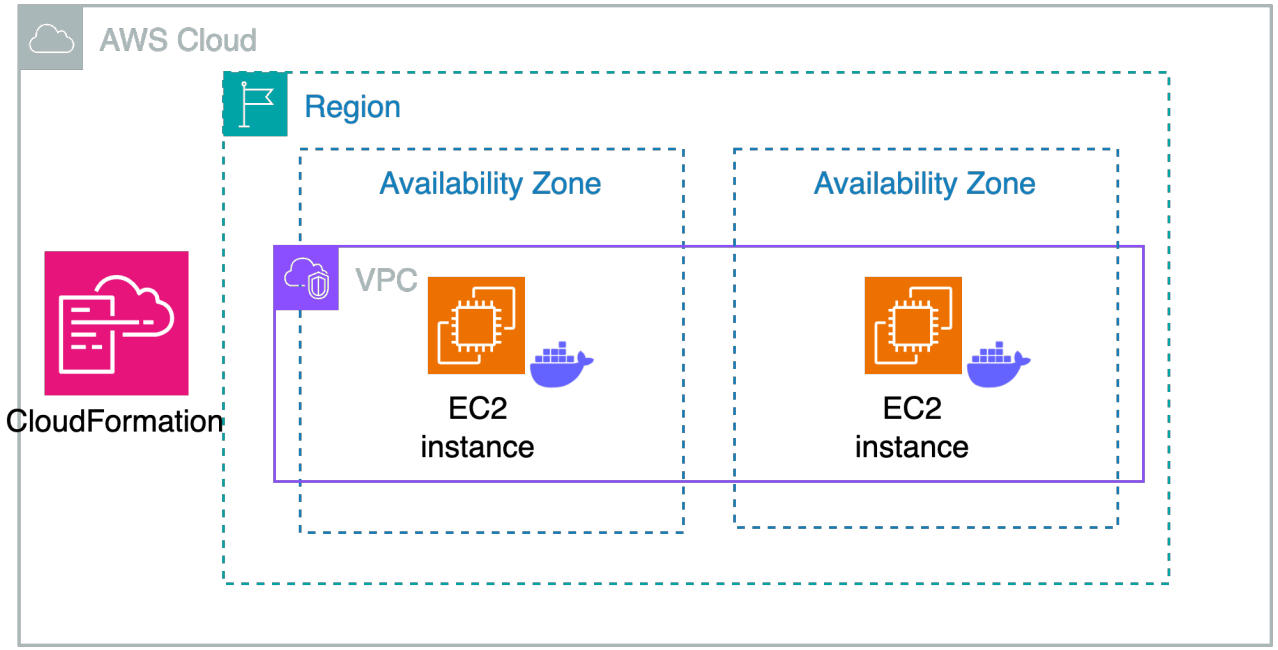
2. Design

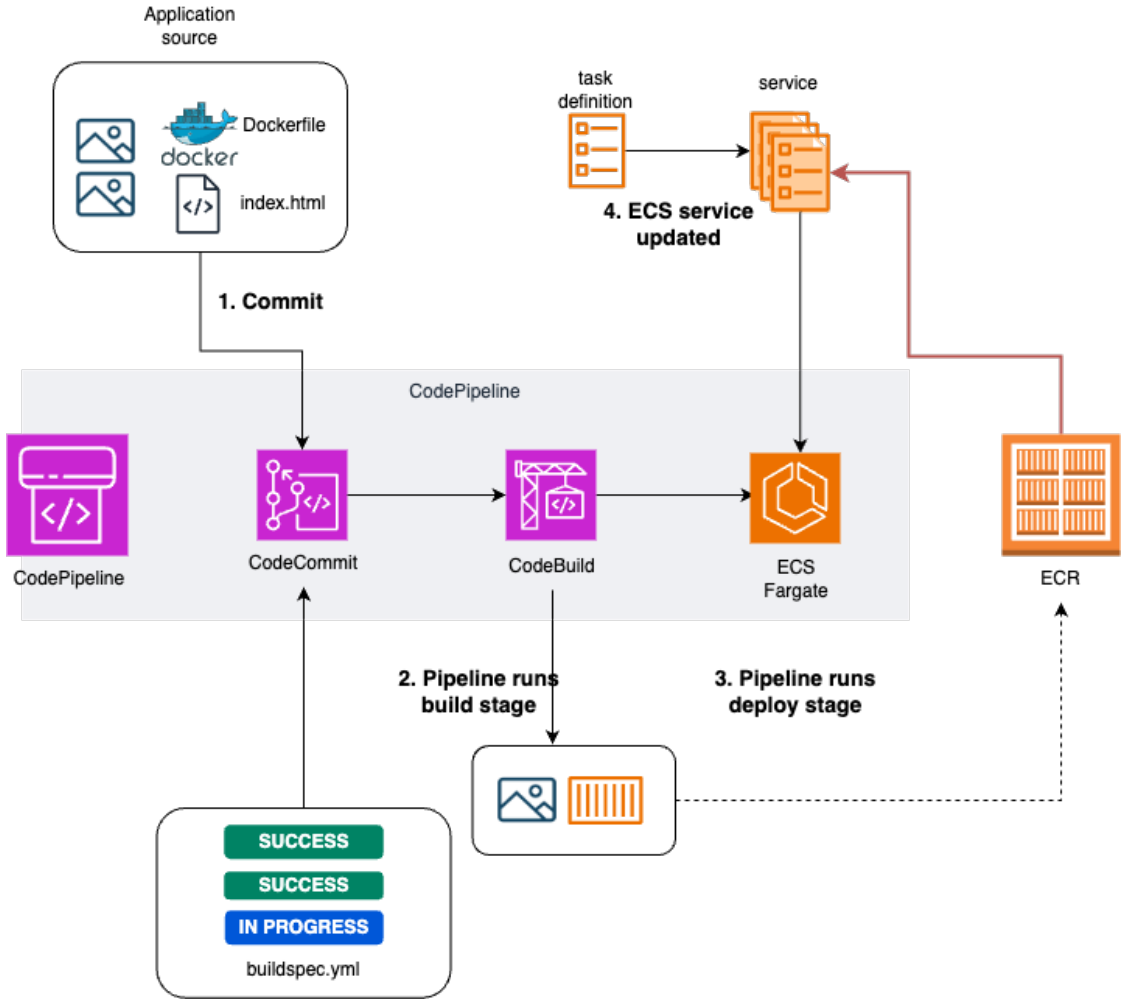
• Specification (Solution) for Requirements:

No	Requirement	Specification (Solution)
1	Automatically build an application's environment that fulfills the following conditions: ① set up a network infrastructure ② set EC2 instances with docker installed in each public subnet, using nginx as the web server.	Use CloudFormation to automatically build the environment. ① create a YAML file to set up the network infrastructure such as VPC, Security Group, etc ② use the same YAML file as the above one to set up EC2 instances.
2	Automatically build and deploy the application	Use CodePipeline with CodeBuild and CodeDeploy. (Use CodeCommit as a source repository) With a build process, use ECR. This is where their docker image is stored and managed
3	Use container service(s) for deployment destination. It should be serverless.	Use ECS Fargate

2. Design

Diagram: CodePipeline + ECR + ECS Fargate with CloudFormation





3. Implementation

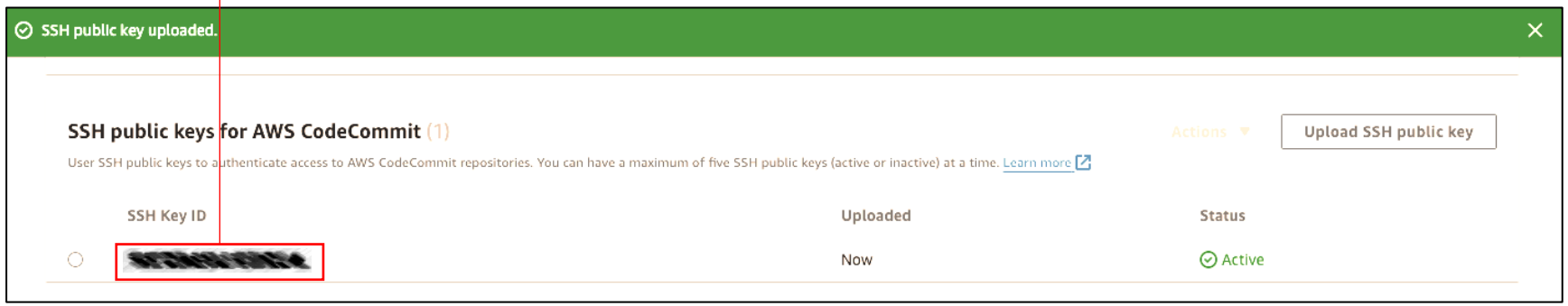
Configure CodeCommit with SSH public key:

1. On your terminal, generate a SSH public key
2. On AWS Console, go to the IAM console, upload the SSH key to Security Credential under the AWS CodeCommmit section
3. Add the SSH Key ID to the config file from your terminal

```
# vi .ssh/config

Host git-codecommit.*.amazonaws.com
  User <YOUR_SSH_KEY_ID>
  IdentityFile ~/.ssh/codecommit
```

```
#run the following command to generate SSH key (This demo is done on Mac)
cd .ssh
ssh-keygen -t rsa -b 4096  #name the key 'codecommit
cat ~/.ssh/codecommit.pub
vi config # add the configuration as left
chmod 600 config
```

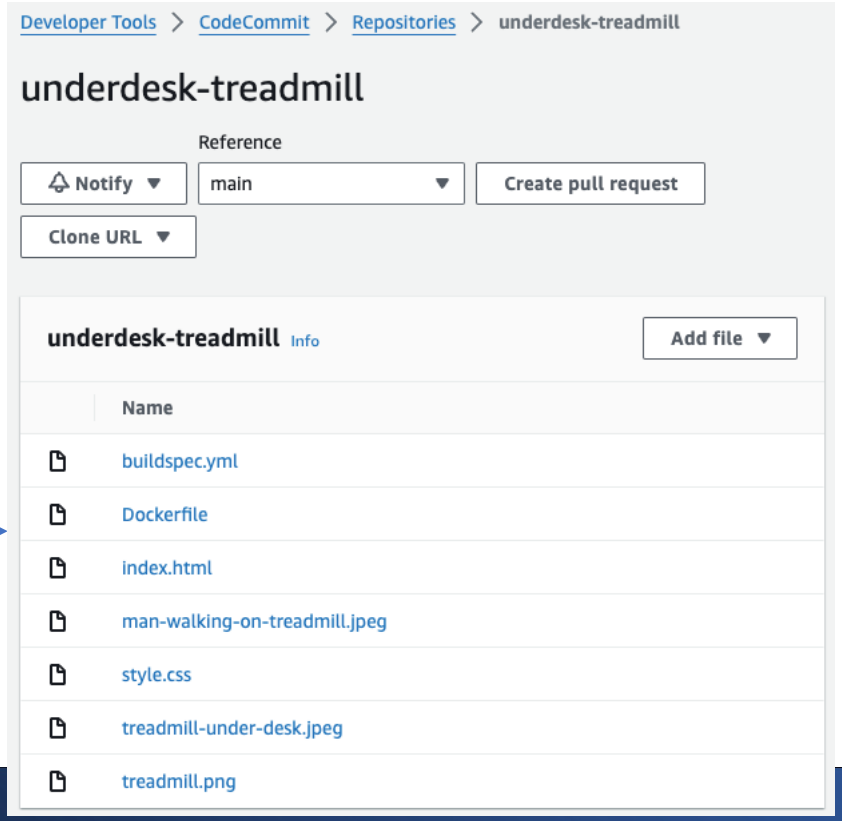


3. Implementation

Setup the source code:

1. On the AWS CodeCommit console, create a CodeCommit repository
2. On your terminal, clone the repository and copy the necessary files (below) to CodeCommit
    1. Dockerfile
    2. index.html
    3. style.css
    4. *.jpeg
    5. treadmill.png
    6. buildspec.yml
3. Git push

```
chihiro@Chihiros-Air repos/underdesk-treadmill main git add -A
chihiro@Chihiros-Air repos/underdesk-treadmill main + git commit -m 'first commit'
[main (root-commit) 7bc3284] first commit
 7 files changed, 100 insertions(+)
 create mode 100644 Dockerfile
 create mode 100644 buildspec.yml
 create mode 100644 index.html
 create mode 100644 man-walking-on-treadmill.jpeg
 create mode 100644 style.css
 create mode 100644 treadmill-under-desk.jpeg
 create mode 100644 treadmill.png
chihiro@Chihiros-Air repos/underdesk-treadmill main
```



3. Implementation

Setup Elastic Container Registry (ECR):
Configure the Elastic Container Registry to manage docker images.
1. Create a private repository

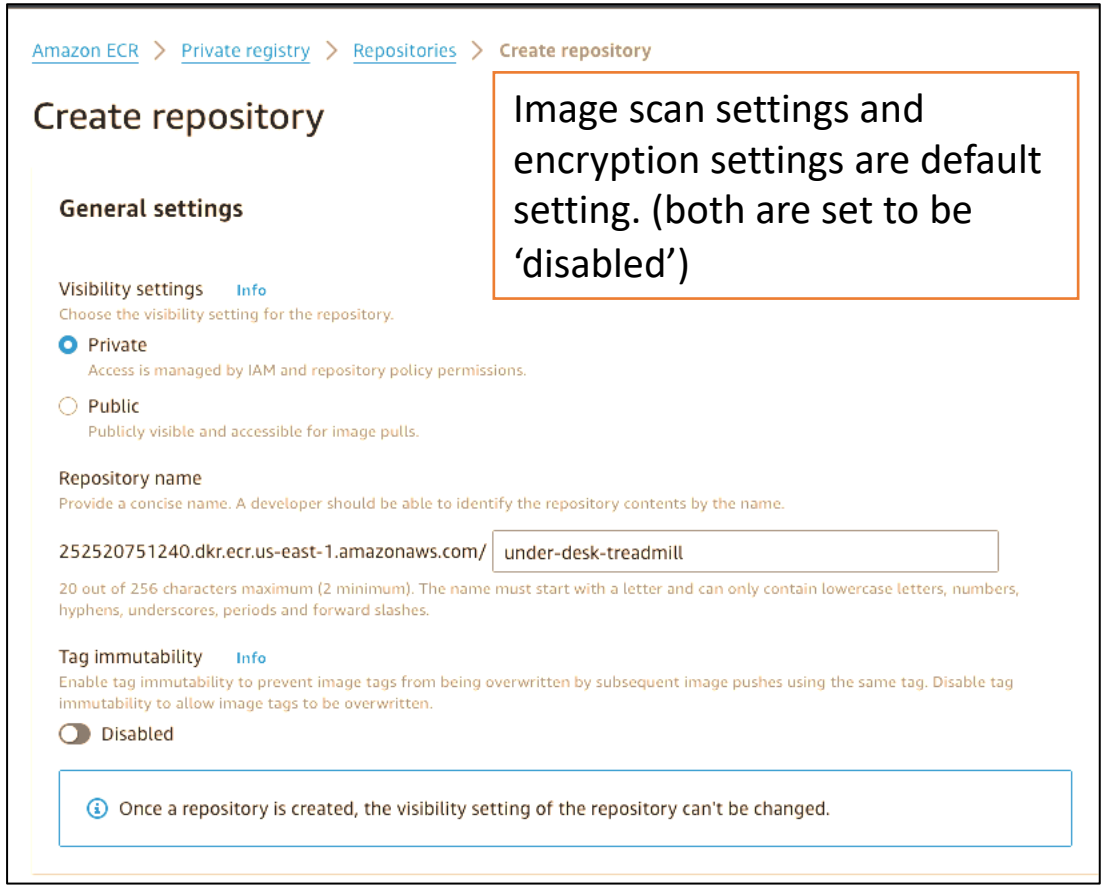
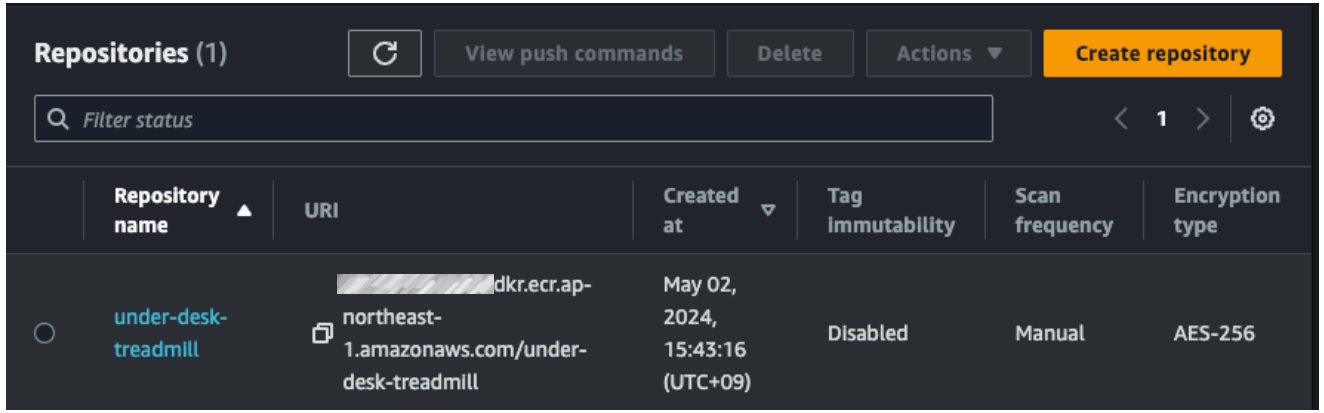


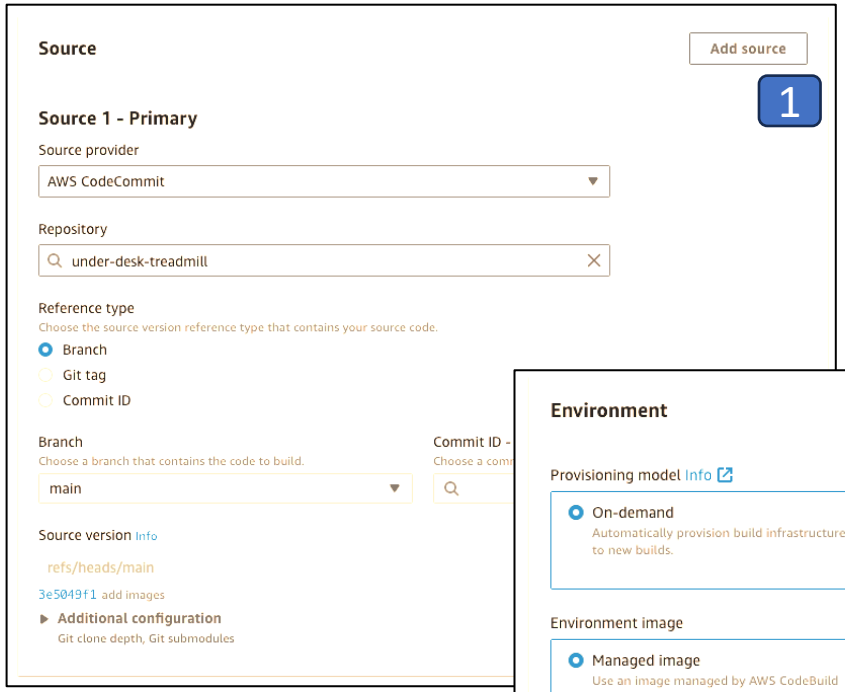
Image scan settings and encryption settings are default setting. (both are set to be 'disabled')

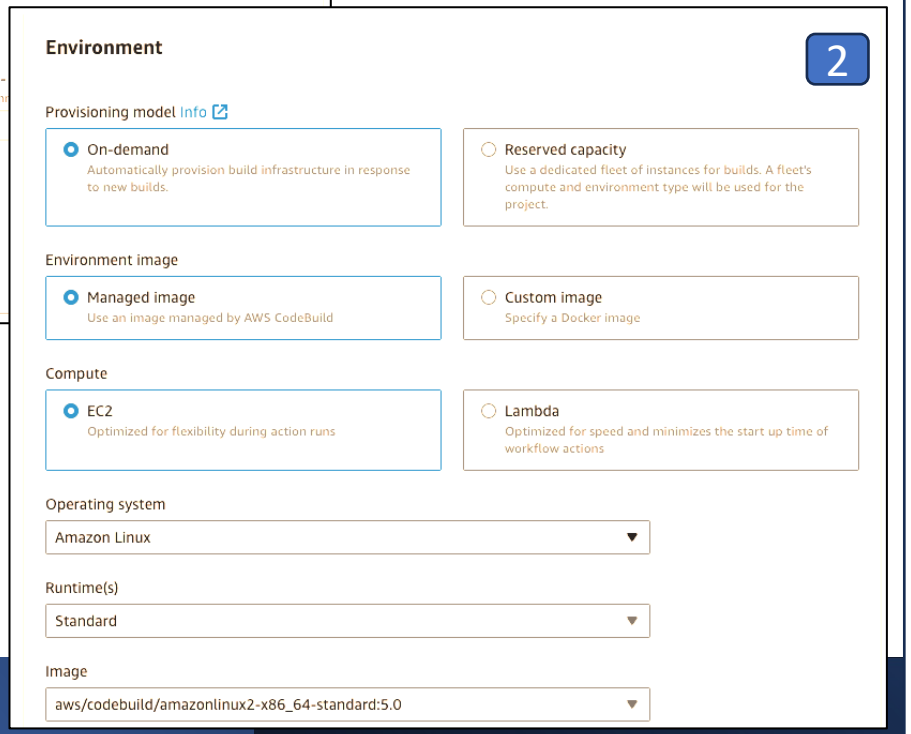


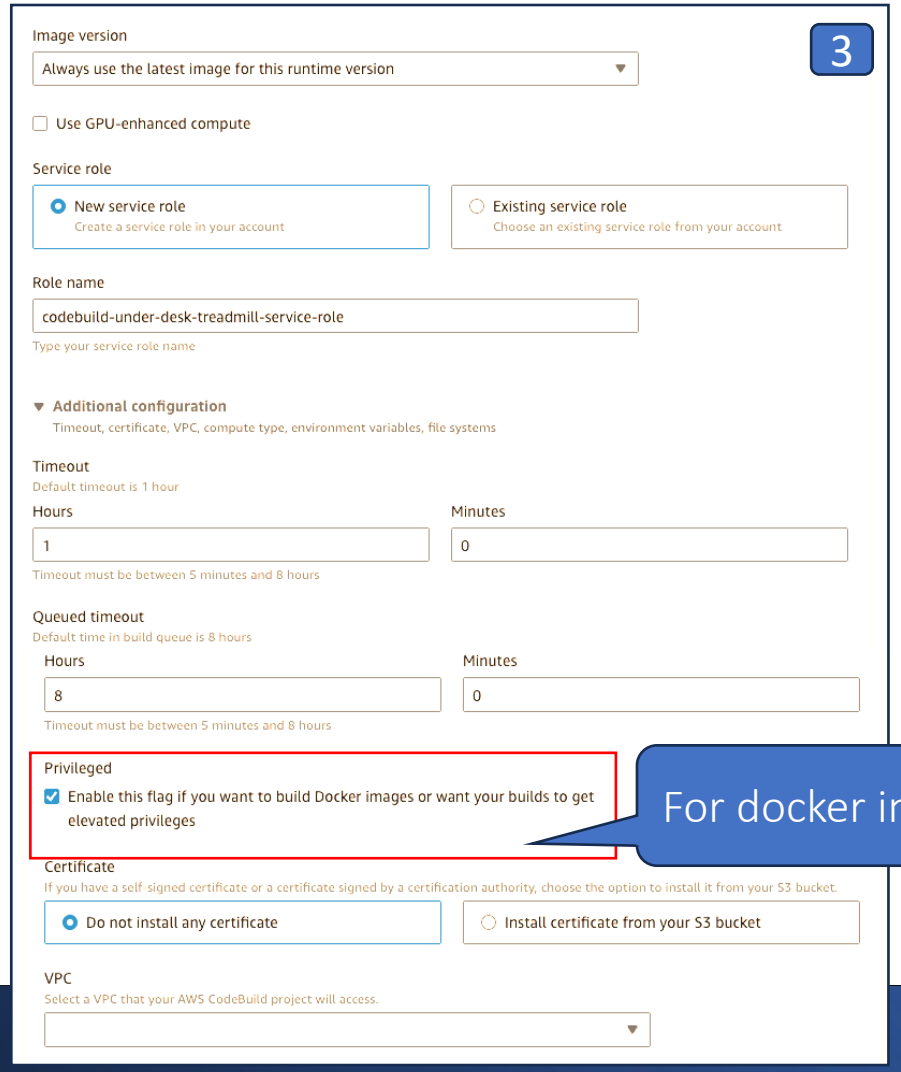
3. Implementation

Setup CodeBuild: (Continue 1/3)

Configure a CodeBuild Project to take the contents of the CodeCommit repository, build a docker image, and store it in the ECR.





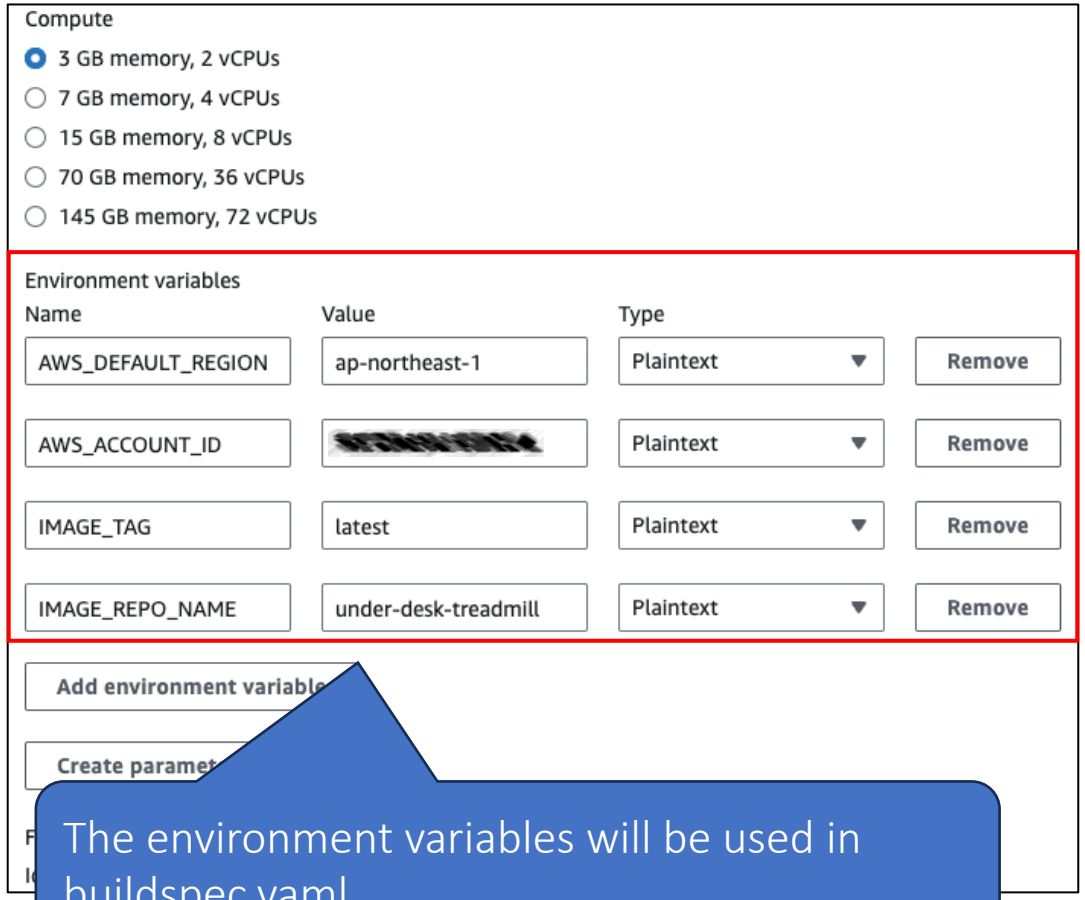


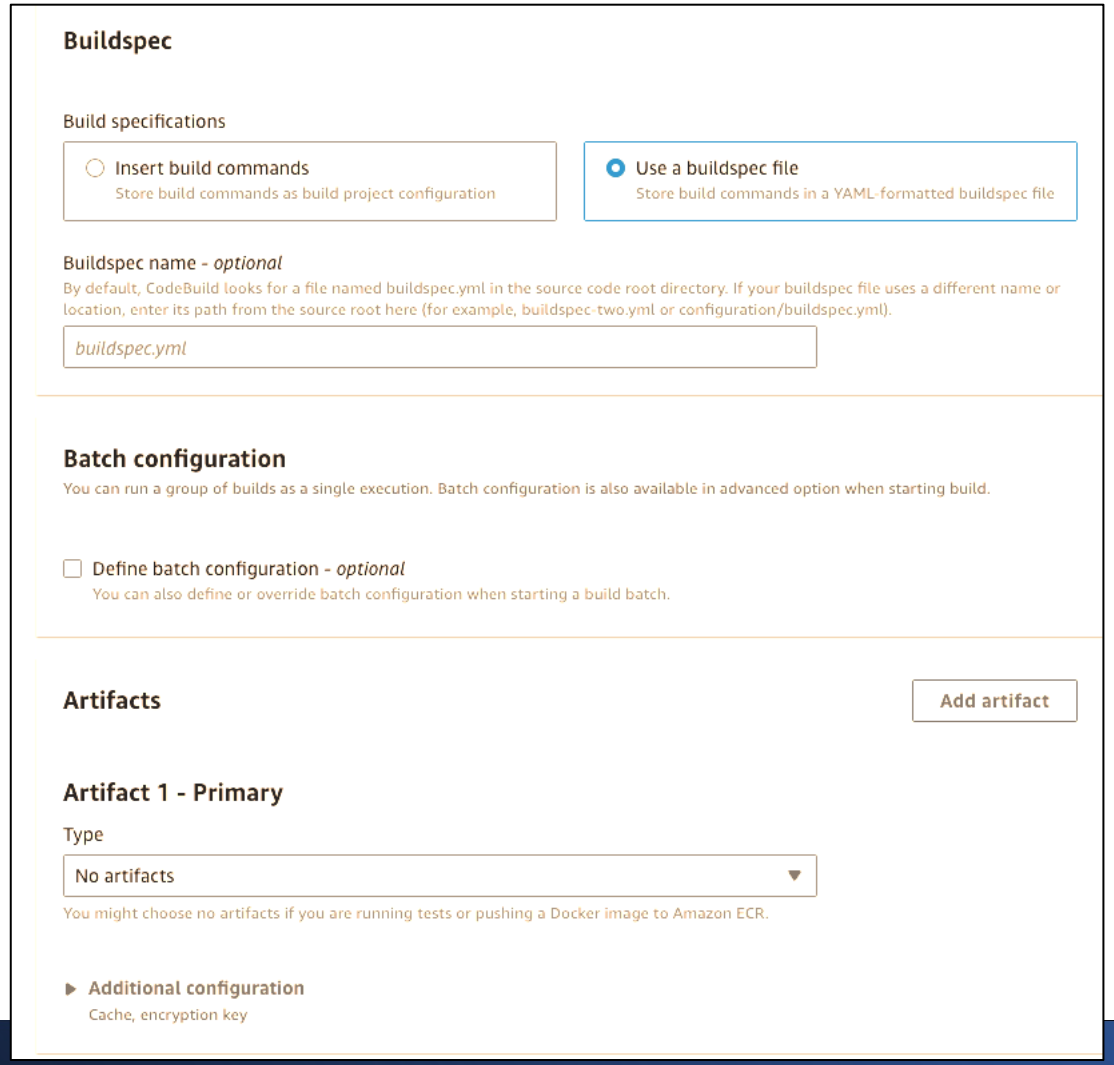
For docker image

3. Implementation

Setup CodeBuild: (Continue 2/3)

Configure a CodeBuild Project to take the contents of the CodeCommit repository, build a docker image, and store it in the ECR.

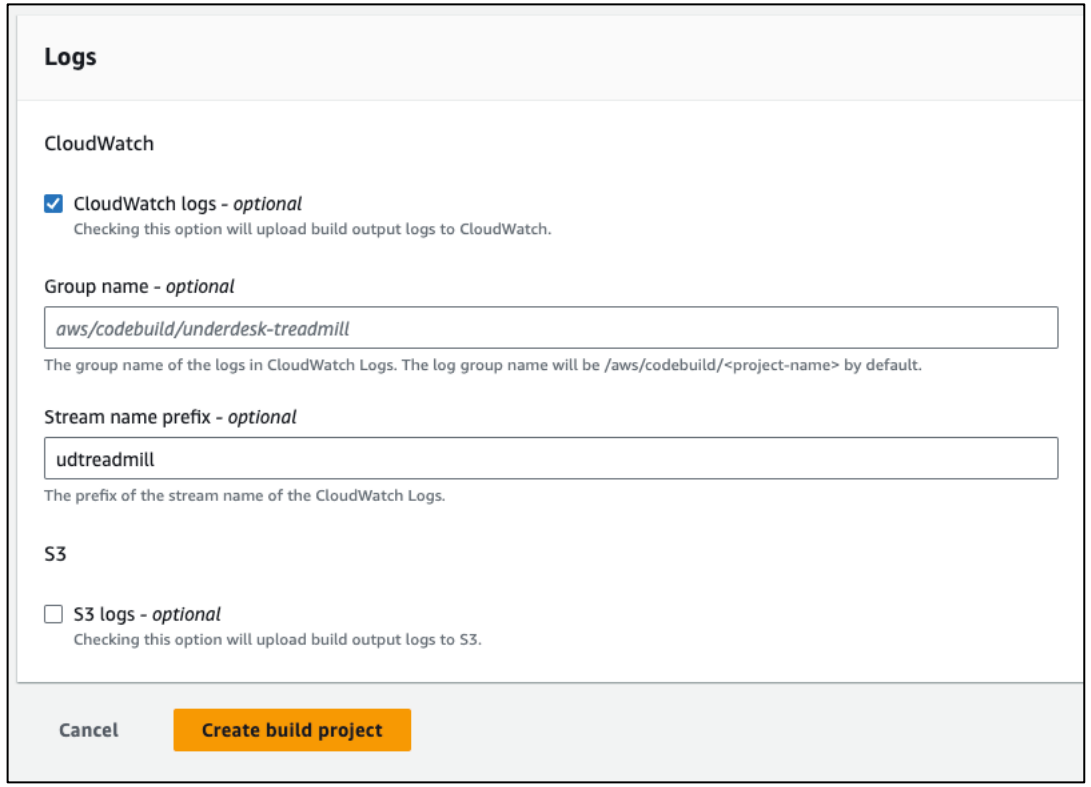




3. Implementation

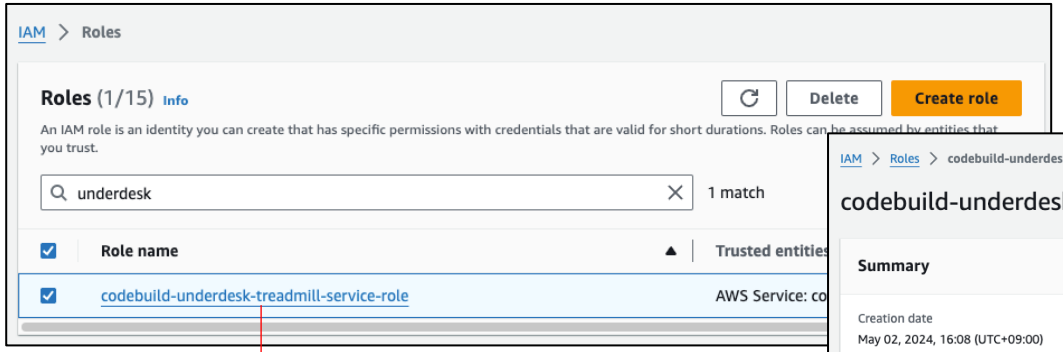
Setup CodeBuild: (Continue 3/3)

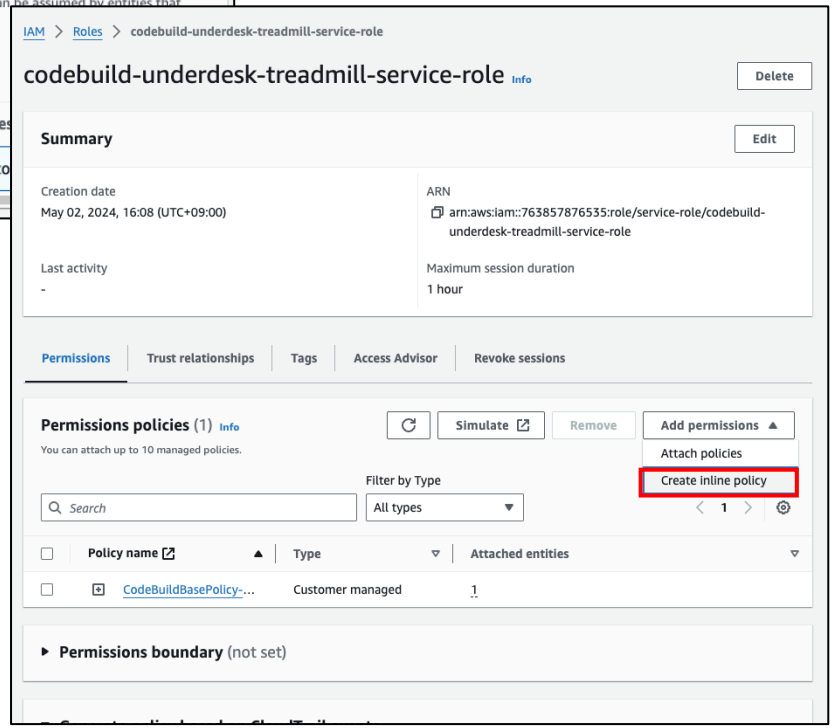
Configure a CodeBuild Project to take the contents of the CodeCommit repository, build a docker image, and store it in the ECR.



3. Implementation

Configure IAM Role for CodeBuild security and permissions: (Continue 1/2)
The build project will use IAM Role which is automatically created by CodeBuild. For the Build Project to access ECR to store a docker image, update the role permissions with ALLOWs for ECR.





Create a new inline policy in JSON. Delete the skelton JSON and replace with this.
Name the new policy ‘codebuild-ecr’

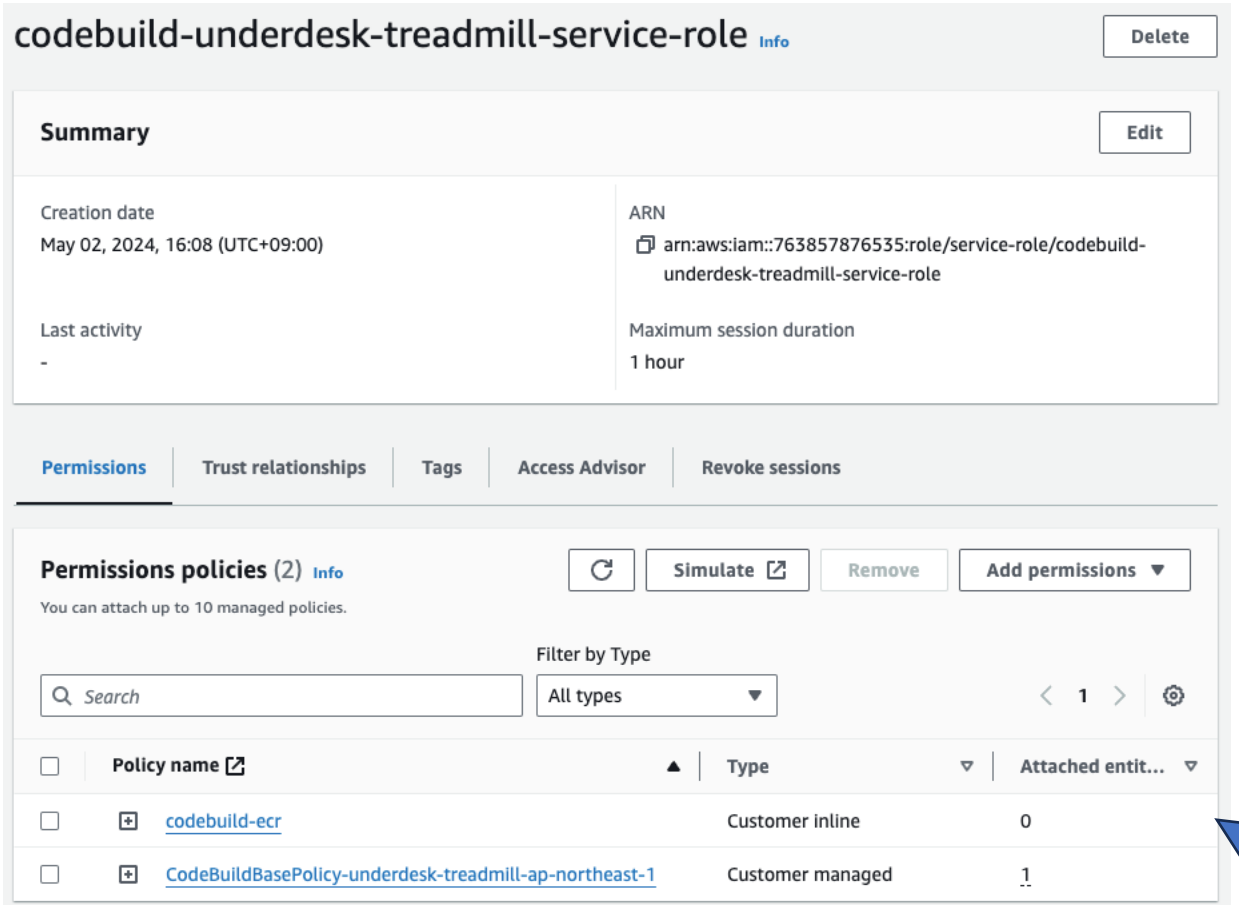
Policy editor

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:CompleteLayerUpload",
                "ecr:GetAuthorizationToken",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:UploadLayerPart"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

How to test CodeBuild will be presented in the latter slides.

3. Implementation

Configure IAM Role for CodeBuild security and permissions: (Continue 2/2)
The build project will use IAM Role which is automatically created by CodeBuild. For the Build Project to access ECR to store a docker image, update the role permissions with ALLOWs for ECR.



The IAM Role for CodeBuild should look like this.

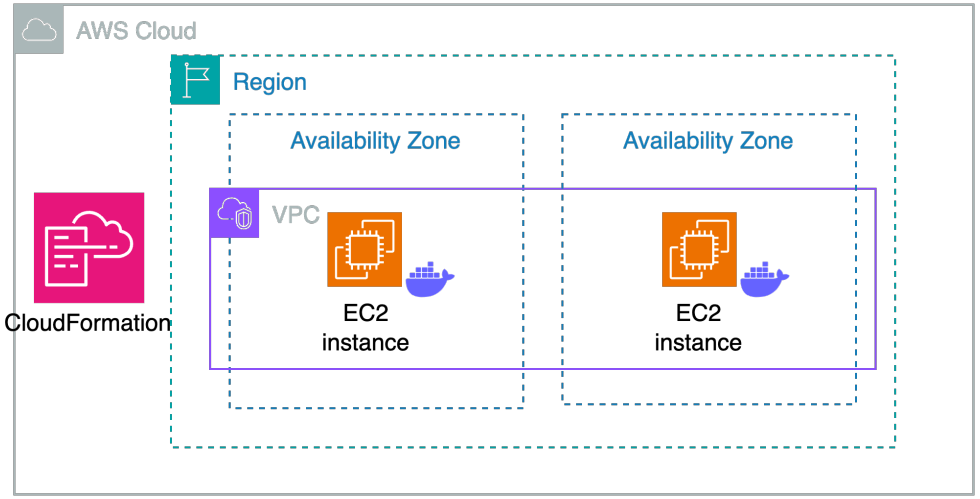
3. Implementation

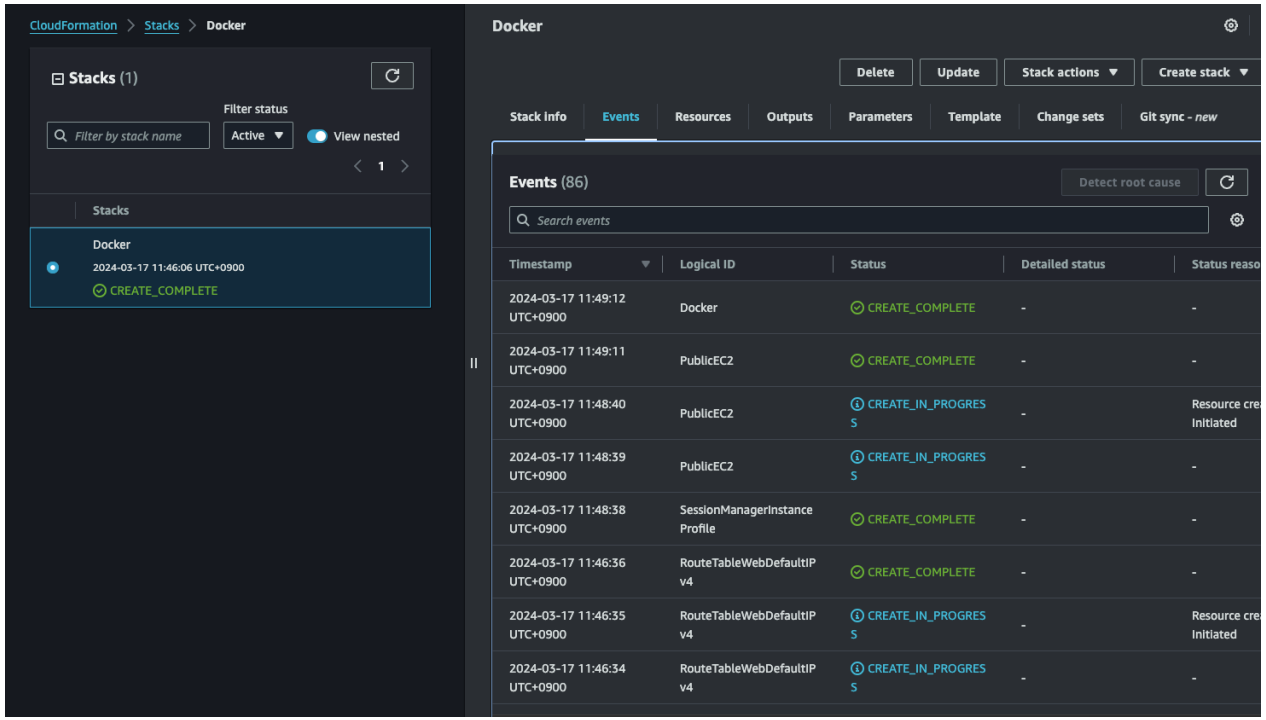
Use CloudFormation to automatically build an application's environment that fulfills the following conditions:

① set up a network infrastructure

② set EC2 instances with docker installed in each public subnet, using nginx as the web server.

Use ec2docker.yaml to build the stack. This template will launch EC2 instances with Docker installed like below:

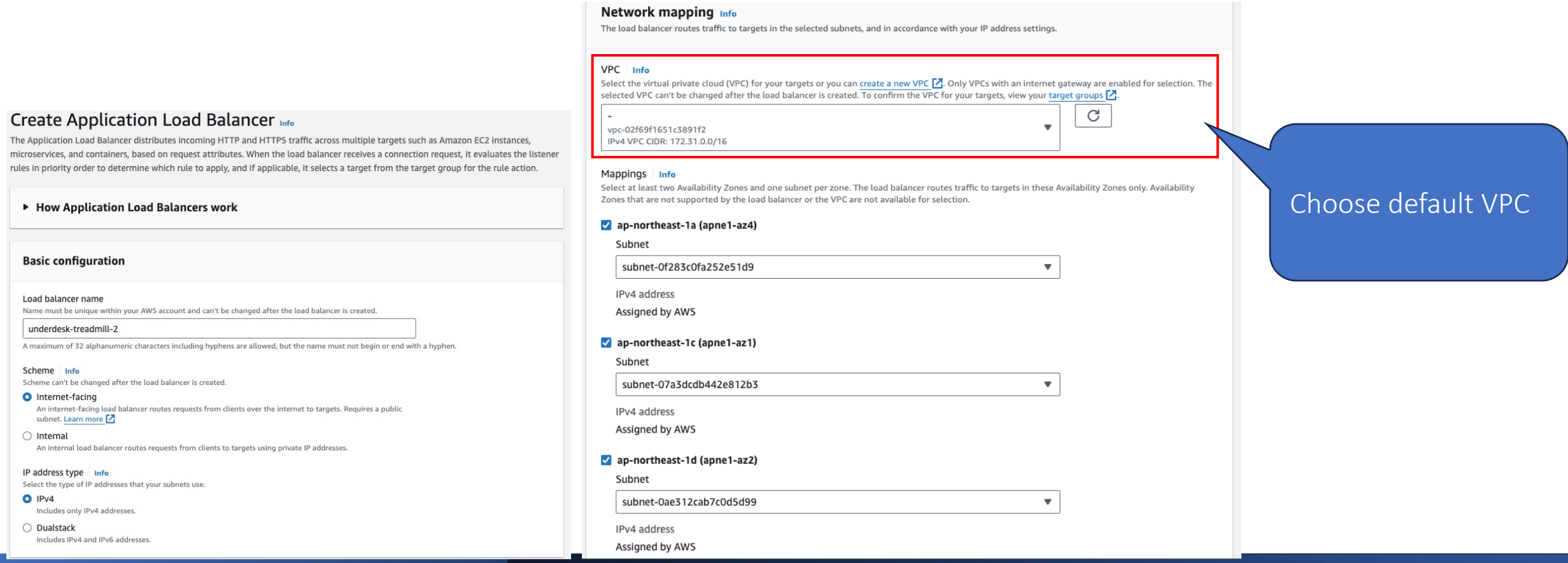




How to test the docker images will be presented in the latter slides.

3. Implementation

Setup CodeDeploy:Configure automated deployment of the application to ECS Fargate.
Before setting up CodeDeploy, set up an application load balancer (ALB). Docker will consume AWS resources.
Using ALB is a key to load off the stress from them. (Continue)

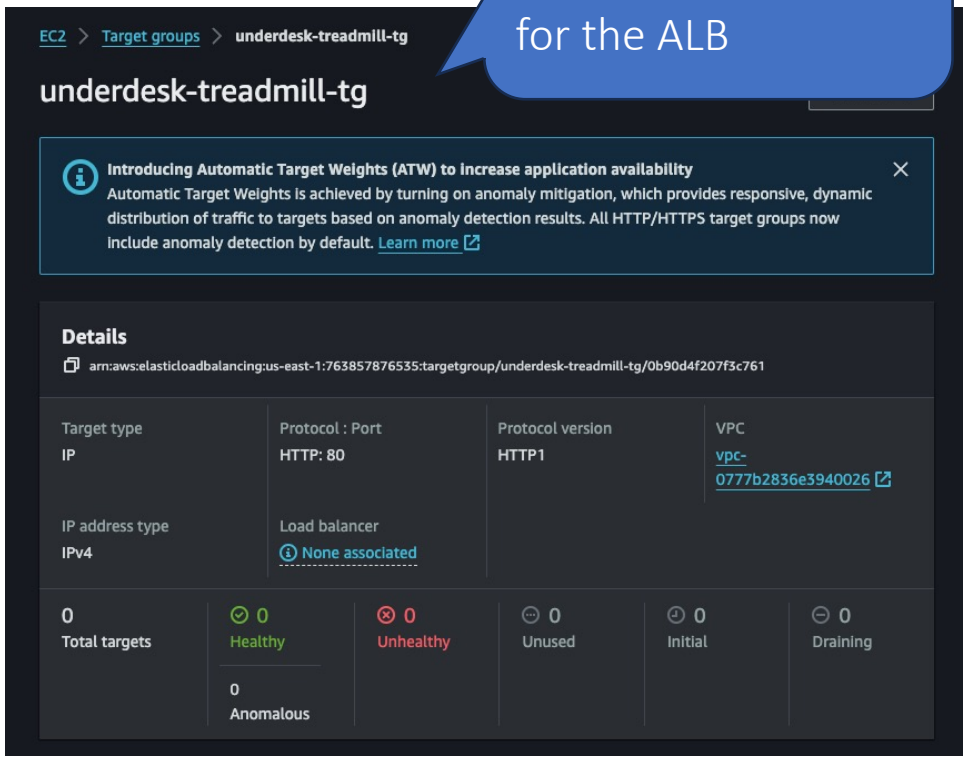


Choose default VPC

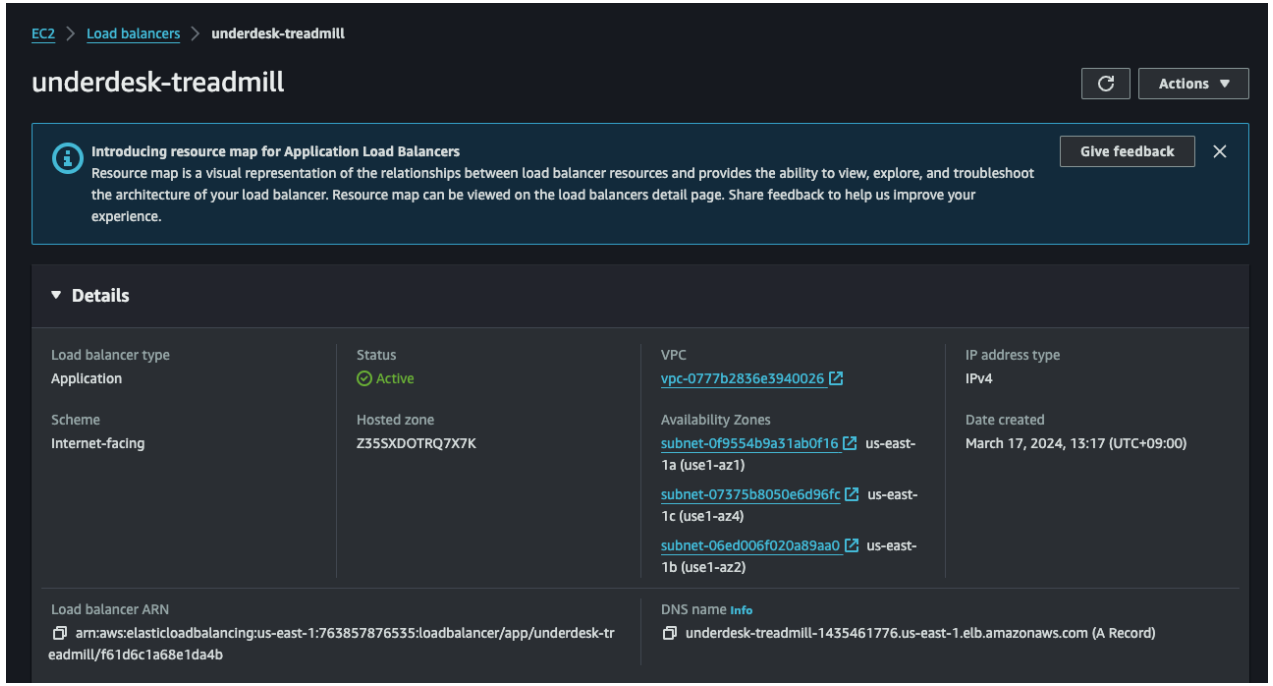
3. Implementation

Setup CodeDeploy:
Configure automated deployment of the application to ECS Fargate.
Set up an application load balancer (ALB). Docker will consume AWS resources. Using ALB is a key to load off the stress from them.

Create ALB Target Group and pick this for the ALB



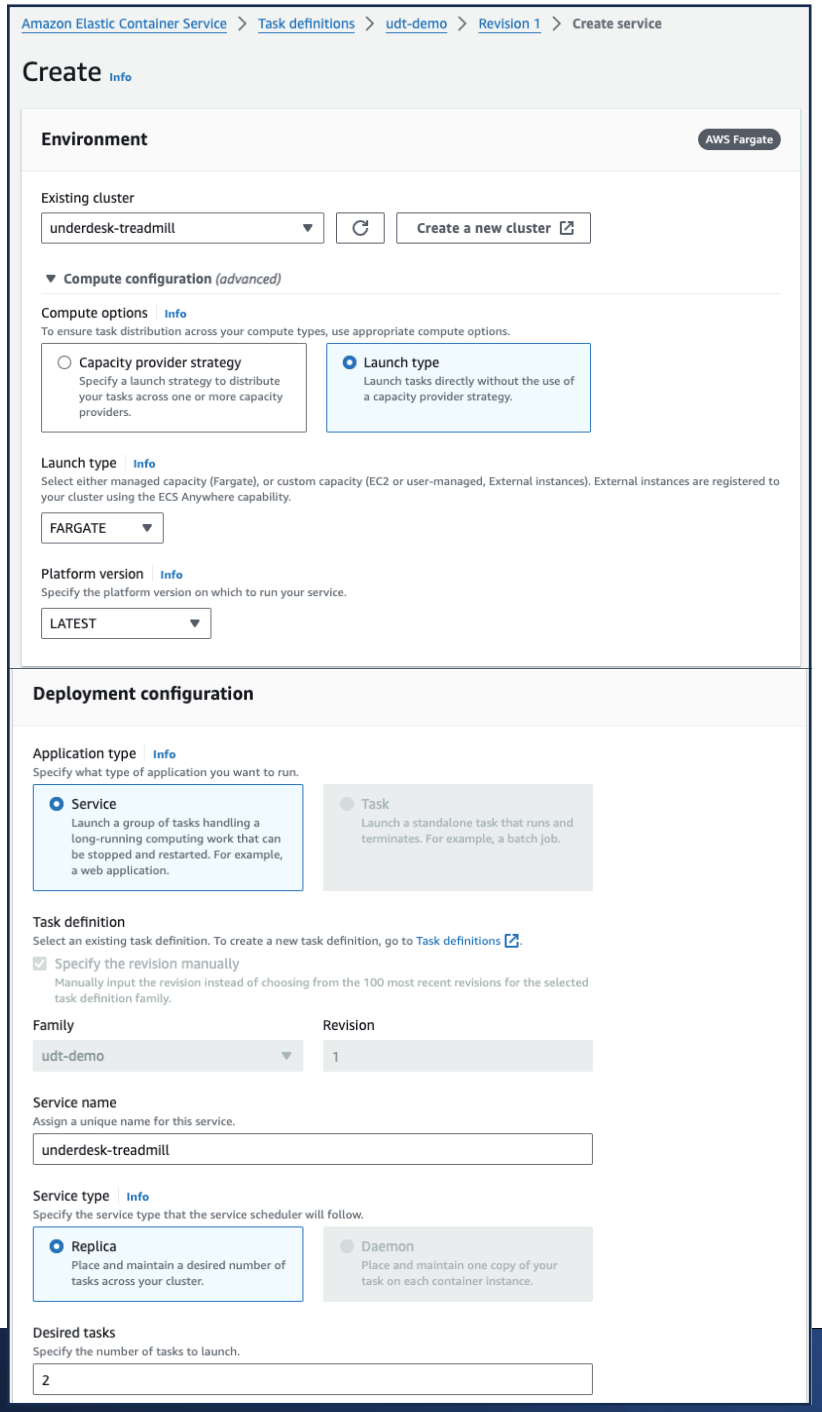
ALB detail

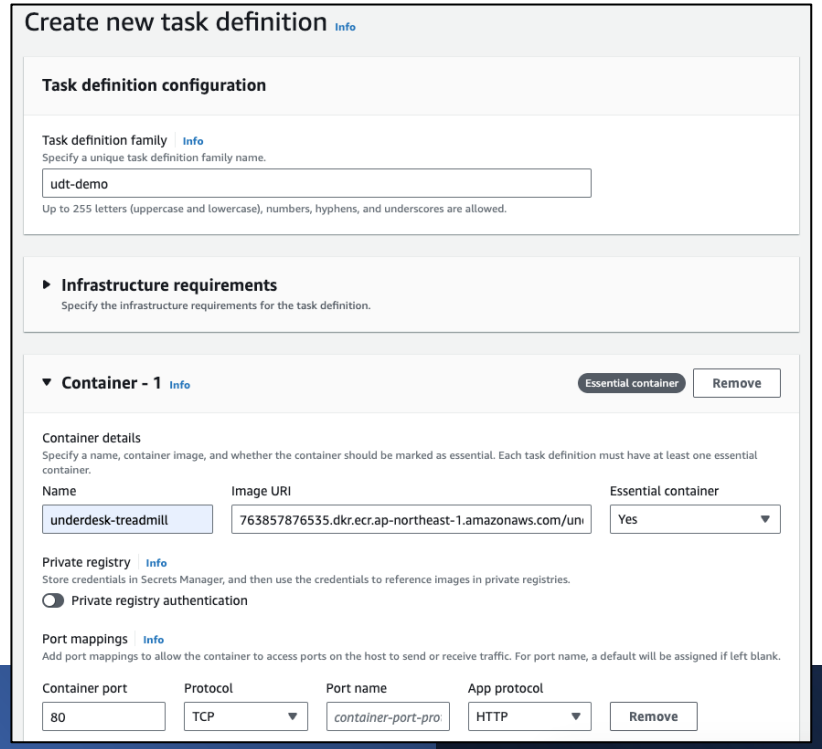


3. Implementation

Setup ECS Fargate:

1. Create the ECS Cluster
2. Create Task and Container Definitions

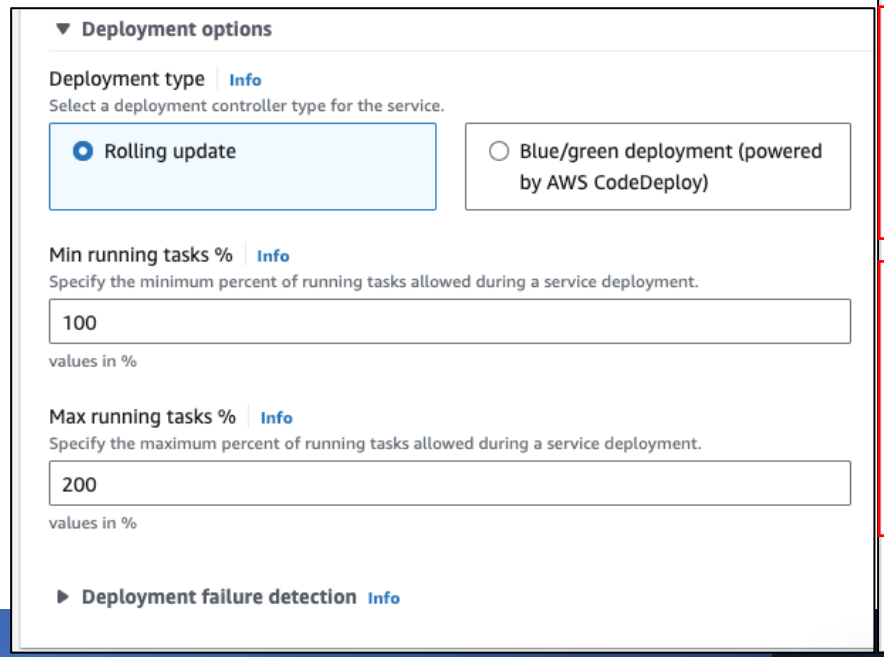


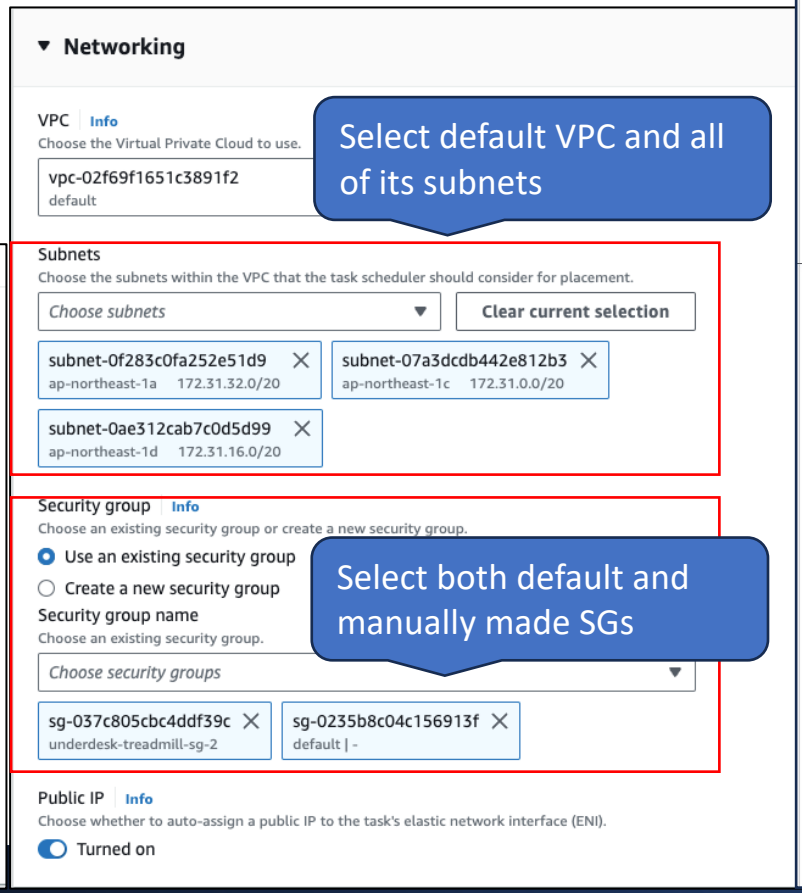


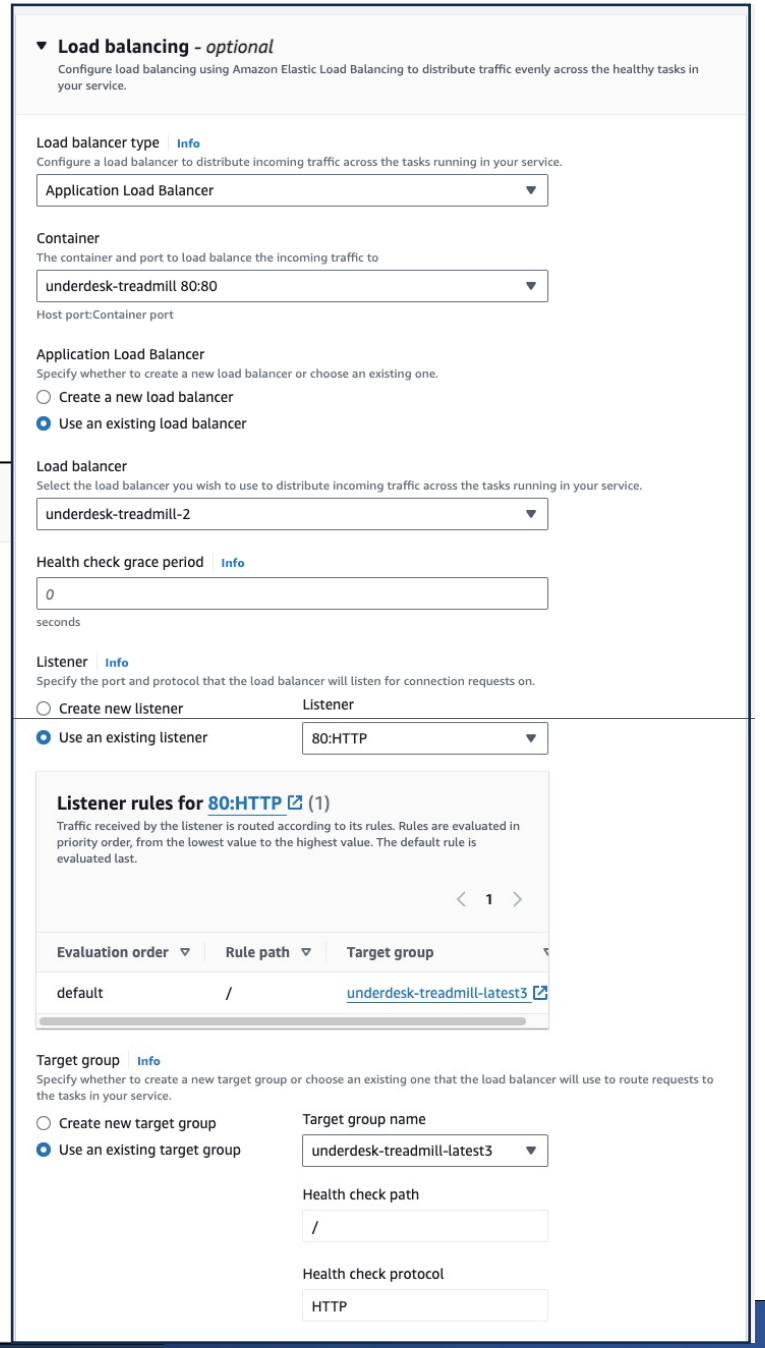
3. Implementation

Setup ECS Fargate:

1. Create a service from Container Definitions



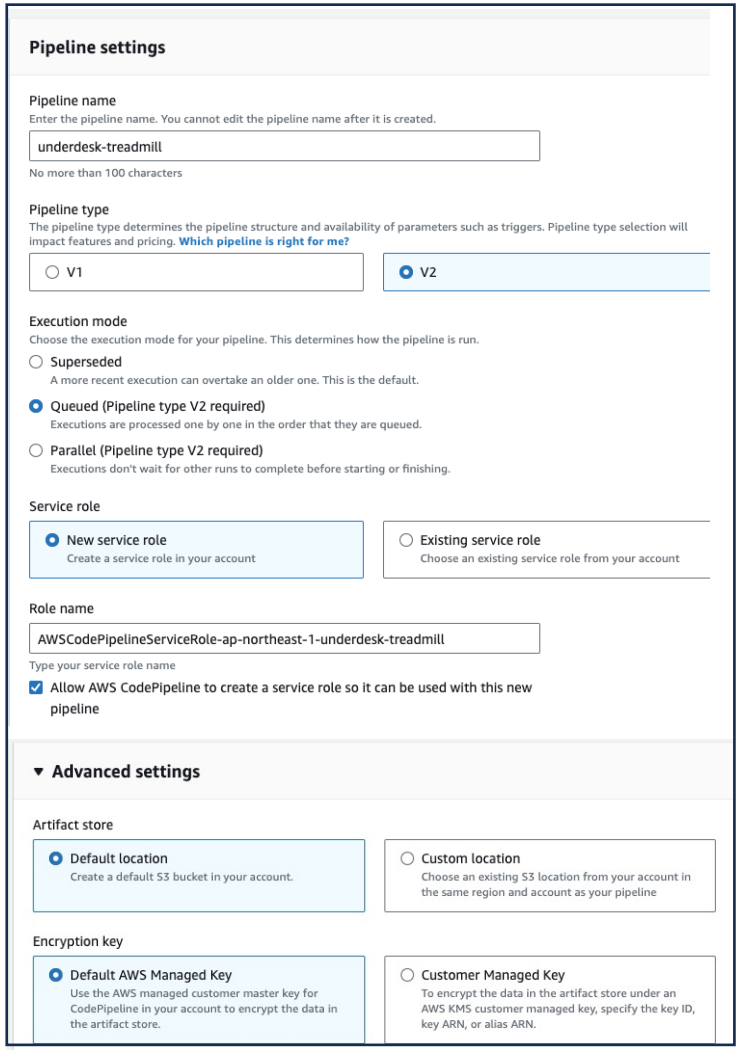


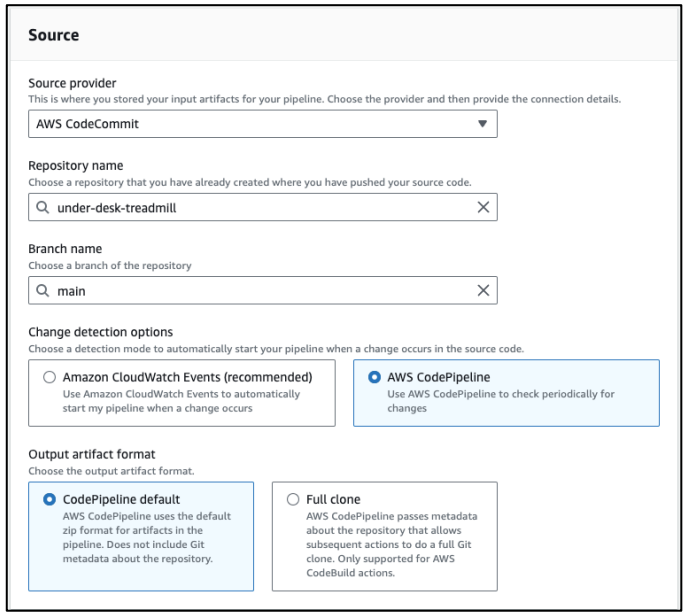


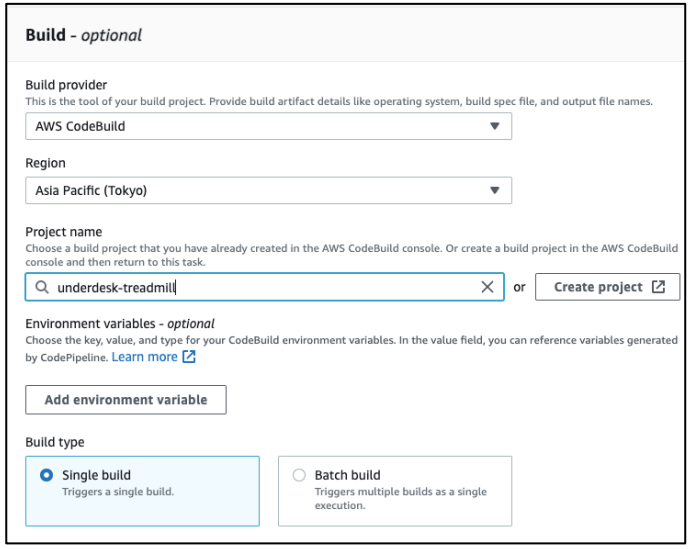
3. Implementation

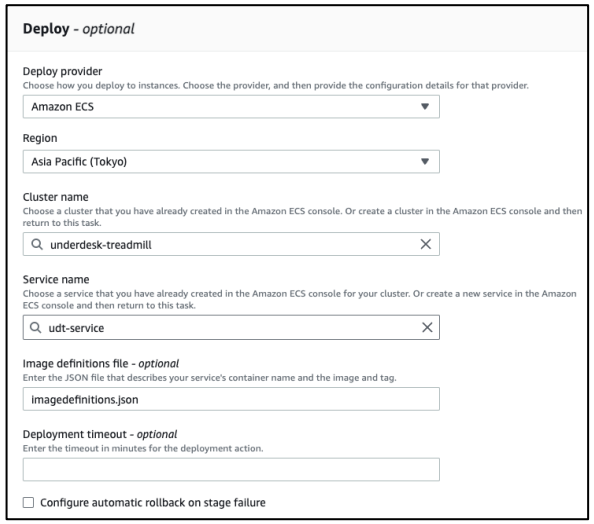
Create CodePipeline:

1. Create a pipeline
2. Add source stage
3. Add build stage
4. Add deploy stage







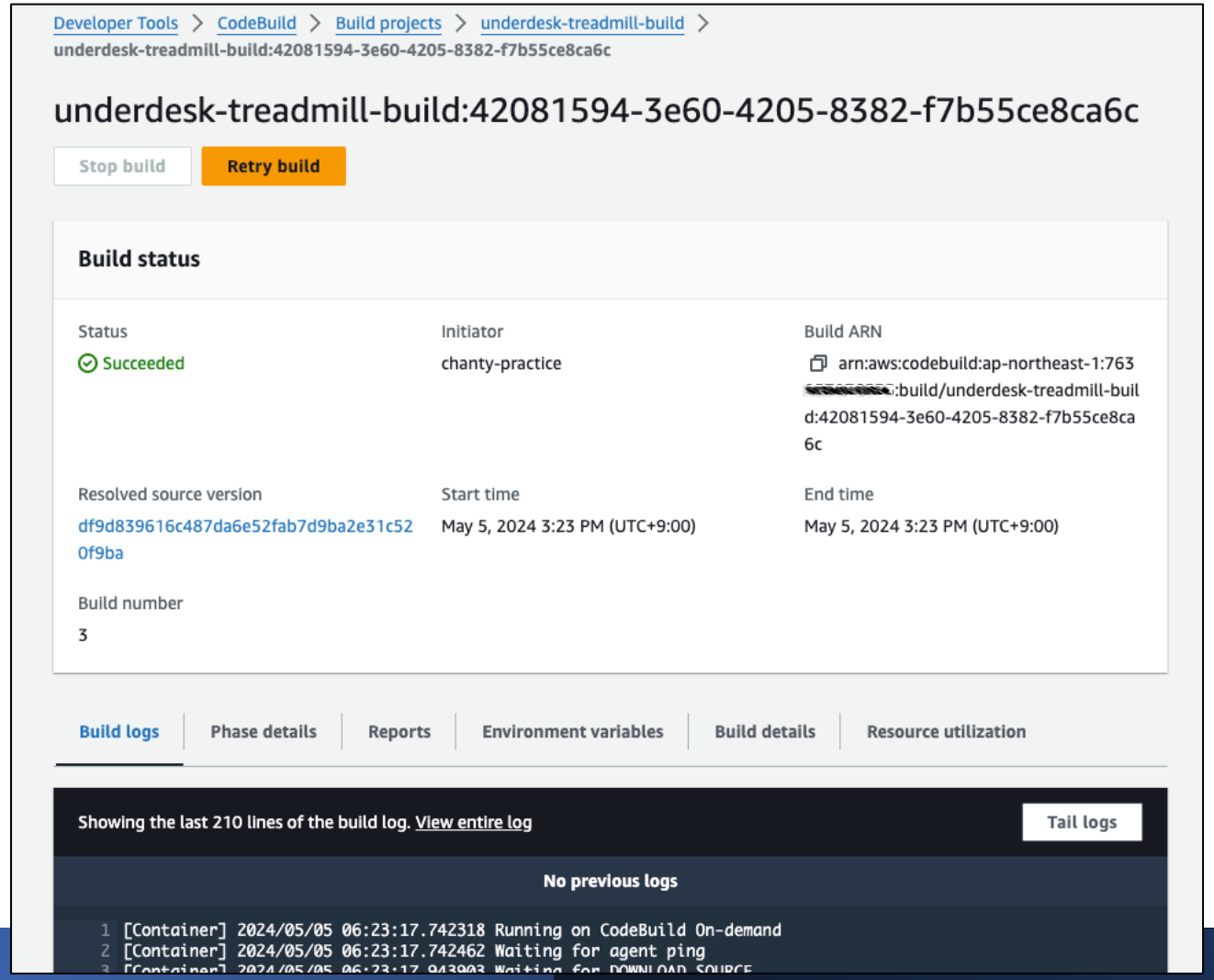


That's it for implementation.

4. Test Results:

Test build:

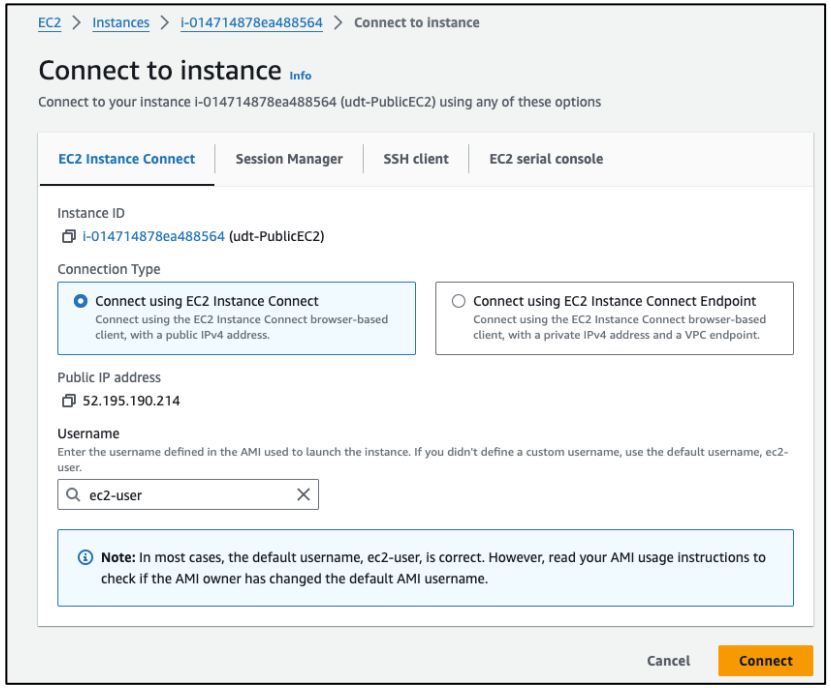
1. Go to CodeBuild Console, select “underdesk-treadmill-build” and click ”Start build”



4. Test Results:

Test your Docker image: (Continue 1/3)

1. Go to EC2 Instance Console and click the EC2 instance created by the CloudFormation
2. Use EC2 Instance Connect to login to your instance




```
# run the following commands:

aws ecr get-login-password --region ap-northeast-1 | docker login --username AWS --password-stdin <YOUR-AWS-ACCOUNT-ID>.dkr.ecr.ap-northeast-1.amazonaws.com
```

```
[ec2-user@ip-10-16-48-197 ~]$ aws ecr get-login-password --region ap-northeast-1 | docker login --username AWS --password-stdin
ap-northeast-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[ec2-user@ip-10-16-48-197 ~]$
```

4. Test Results:

Test your Docker image: (Continue 2/3)

run the following commands:

docker pull <YOUR-AWS-ACCOUNT-ID>. dkr.ecr.ap-northeast-1.amazonaws.com/under-desk-treadmill

docker images # copy the image ID

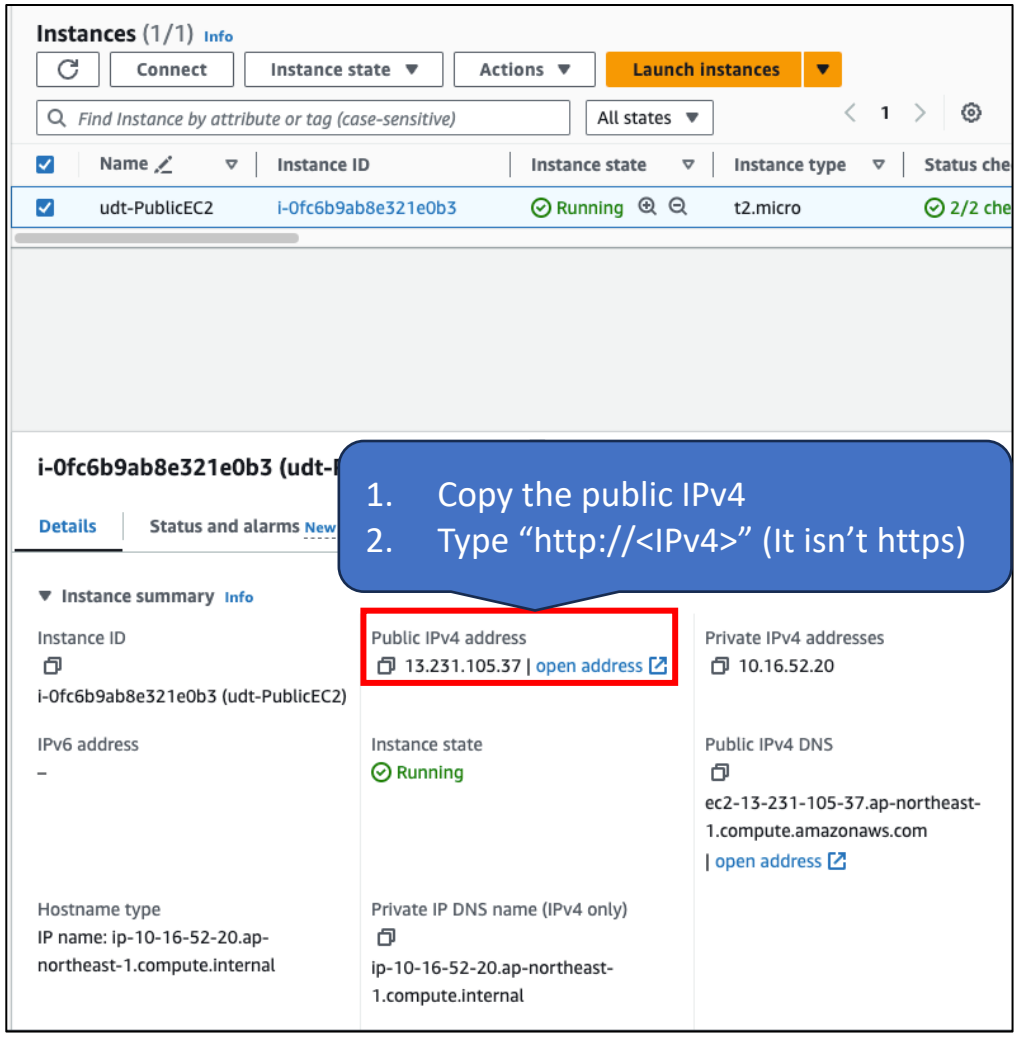
Docker run –p 80:80 <IMAGE_ID>

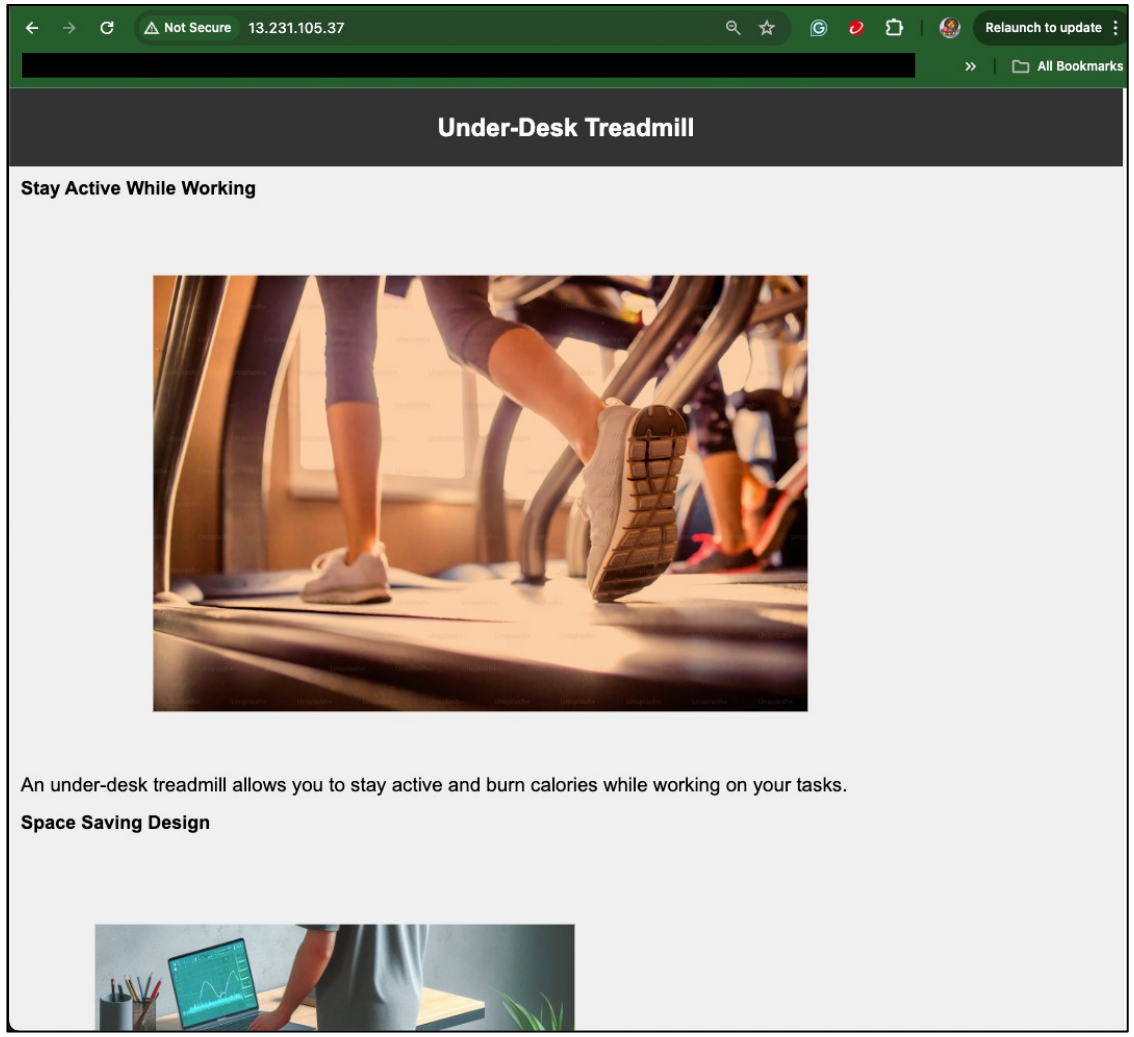
```
[ec2-user@ip-10-16-52-20 ~]$
[ec2-user@ip-10-16-52-20 ~]$ docker pull [redacted].dkr.ecr.ap-northeast-1.amazonaws.com/underdesk-treadmill
Using default tag: latest
latest: Pulling from underdesk-treadmill
b0a0cf830b12: Pull complete
8ddb1e6cdf34: Pull complete
5252b206aac2: Pull complete
988b92d96970: Pull complete
7102627a7a6e: Pull complete
93295add984d: Pull complete
ebde0aa1d1aa: Pull complete
3fd9ff9e3799: Pull complete
3b508517b633: Pull complete
a9ec7b0141fd: Pull complete
cd89e971cb4b: Pull complete
Digest: sha256:df5073f1d69fe6ca4a8f5e0e1203ea9d1d2658d04de21eb12714c
Status: Downloaded newer image for [redacted]kr.ecr.ap-northeast
763857876535.dkr.ecr.ap-northeast-1.amazonaws.com/underdesk-treadmil
```

```
[ec2-user@ip-10-16-52-20 ~]$ docker run -p 80:80 f267e7eb7b85
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/05/05 06:58:01 [notice] 1#1: using the "epoll" event method
2024/05/05 06:58:01 [notice] 1#1: nginx/1.25.5
2024/05/05 06:58:01 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/05/05 06:58:01 [notice] 1#1: OS: Linux 4.14.336-257.568.amzn2.x86_64
2024/05/05 06:58:01 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 32768:65536
2024/05/05 06:58:01 [notice] 1#1: start worker processes
2024/05/05 06:58:01 [notice] 1#1: start worker process 29
```

4. Test Results:

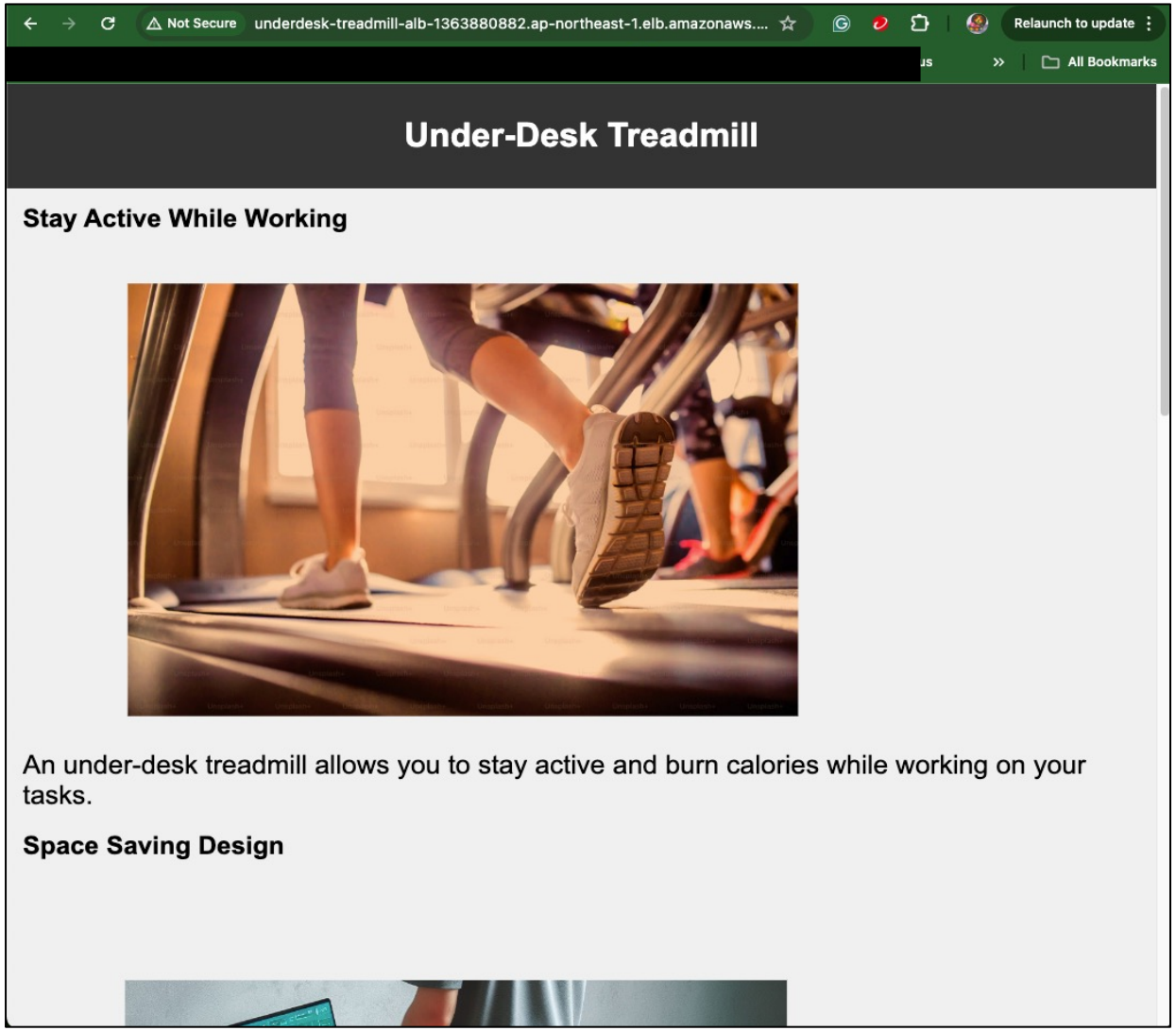
Test your Docker image: (Continue 3/3)





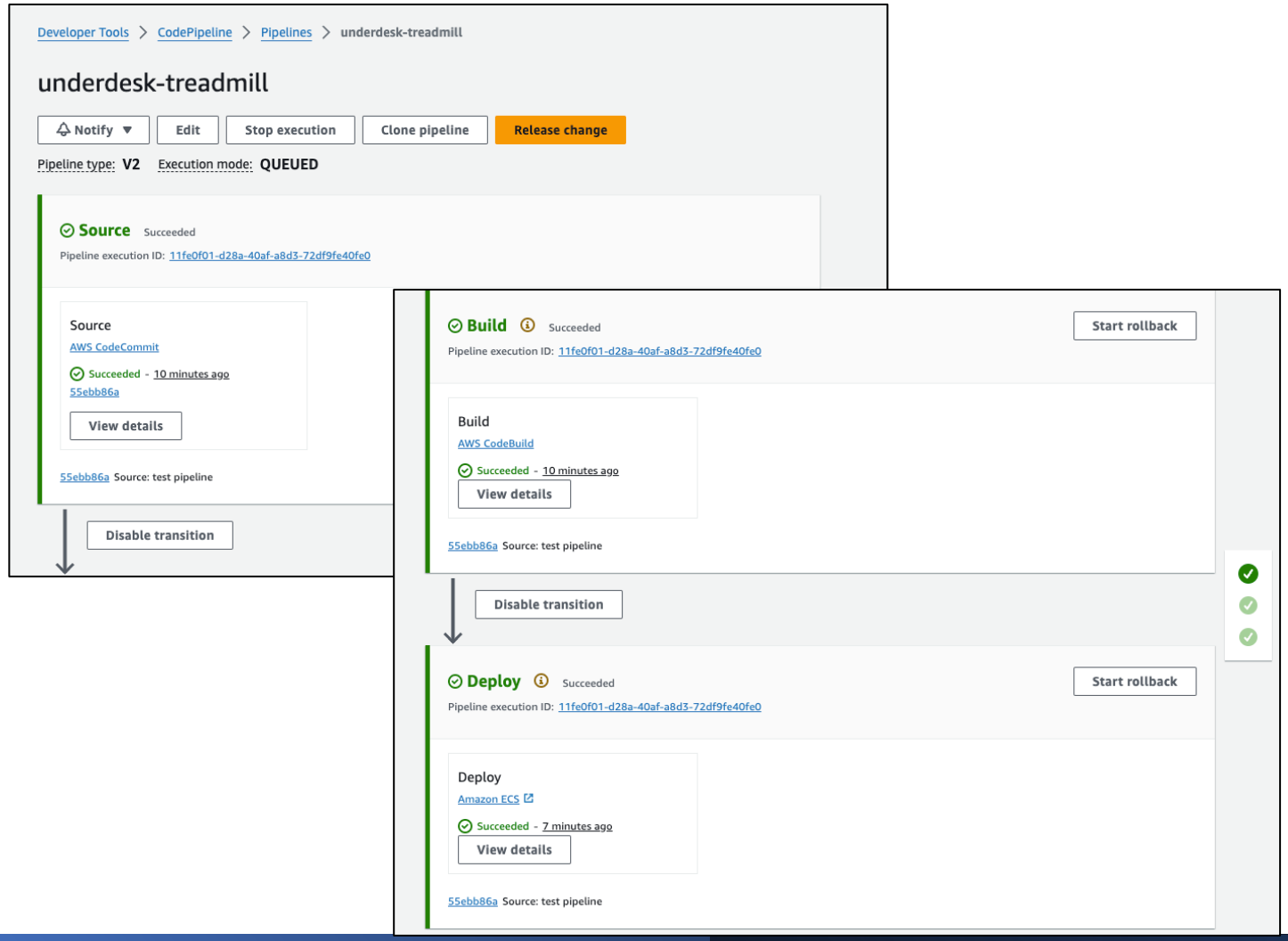
4. Test Results:

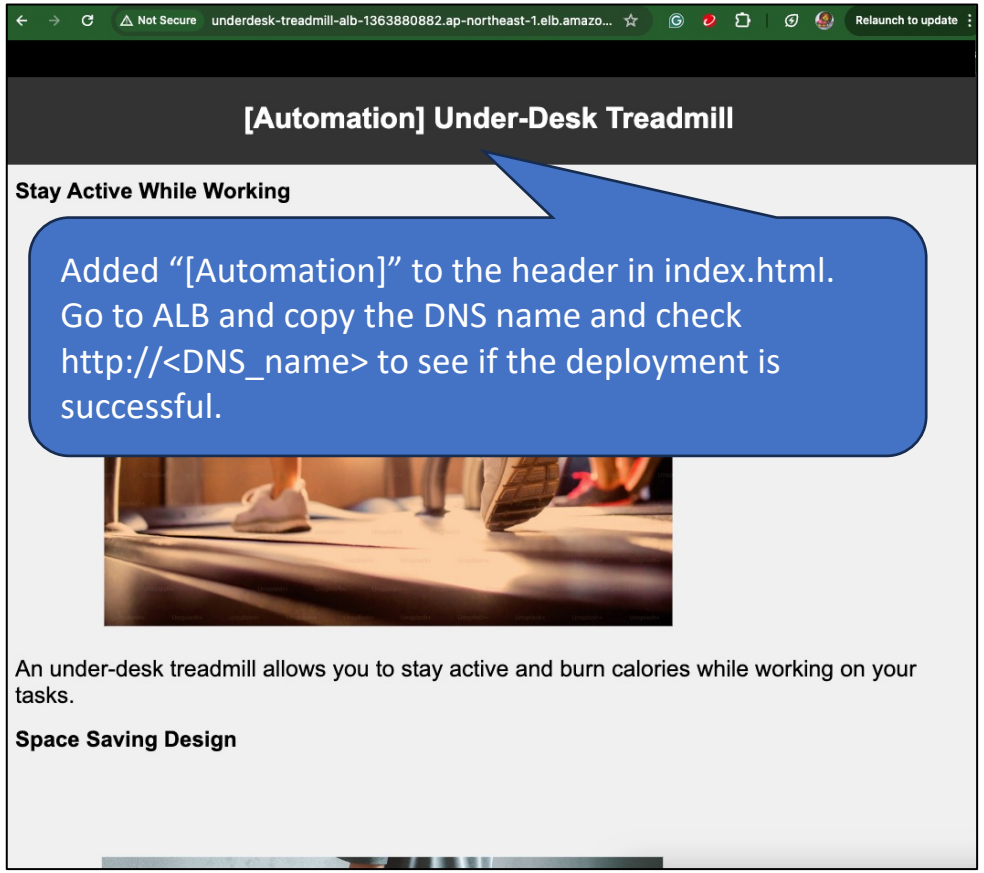
After creating the task definition and service for the ECS, test your Application Load Balancer.



4. Test Results:

After creating the CodePipeline with Deploy stage, add one or two modifications to index.html to test the pipeline.





Fin.