# **AWS: Docker Containers In The Cloud With AWS ECS-Fargate**

Java

# yum provides java

Select the version you want to install.

# yum install java-11-amazon-corretto-11.0.5+10-1.amzn2.x86\_64 –y

# export JAVA\_HOME=/usr/lib/jvm/java-11-amazon-corretto.x86\_64/bin/java

* Git

# yum install git -y

* Maven (optinal,only for maven related jobs )

#cd /opt

#wget <https://www-eu.apache.org/dist/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz>

#sudo tar xzf apache-maven-3.6.3-bin.tar.gz

#sudo ln -s apache-maven-3.6.3 maven

#sudo vi /etc/profile.d/maven.sh

export M2\_HOME=/opt/maven

export PATH=${M2\_HOME}/bin:${PATH}

#source /etc/profile.d/maven.sh

* Docker

#sudo yum update –y

#sudo amazon-linux-extras install docker

#sudo service docker start

#sudo usermod -a -G docker ec2-user

#sudo chkconfig docker on

### [Install Jenkins](https://www.coachdevops.com/2019/07/install-jenkins-on-redhat-enterprise.html)

#sudo yum install wget –y

**Add Jenkins repository to yum repository**

#sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo  
#sudo rpm --import <https://jenkins-ci.org/redhat/jenkins-ci.org.key>

#sudo yum install jenkins –y

**Start Jenkins**  
sudo service jenkins start  
sudo service jenkins status

The manual steps are as follows

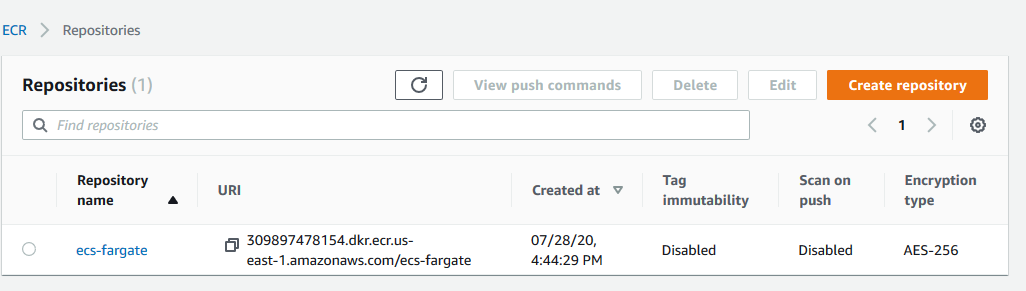
1. Create an ECR
2. Push the Docker image to ECR
3. Create a Cluster
4. Create a task-definition
5. Create a alb
6. Create a Setvice

IAM policy

Create IAM user with appropriate permissions; here I am using admin full access.

1. Create ECR

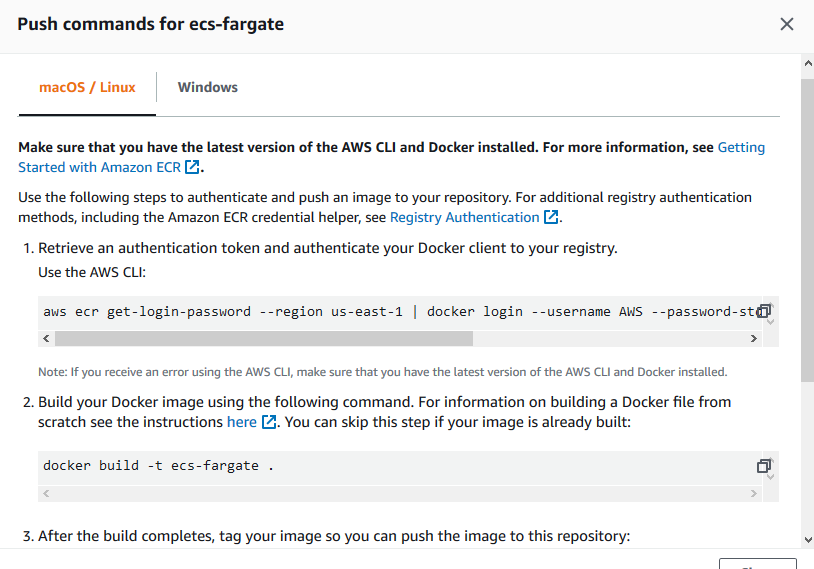
In the ECR service, create a new repository



Using view push commands we can send the image to ECR

Note : $(aws ecr get-login --no-include-email --region us-east-1)

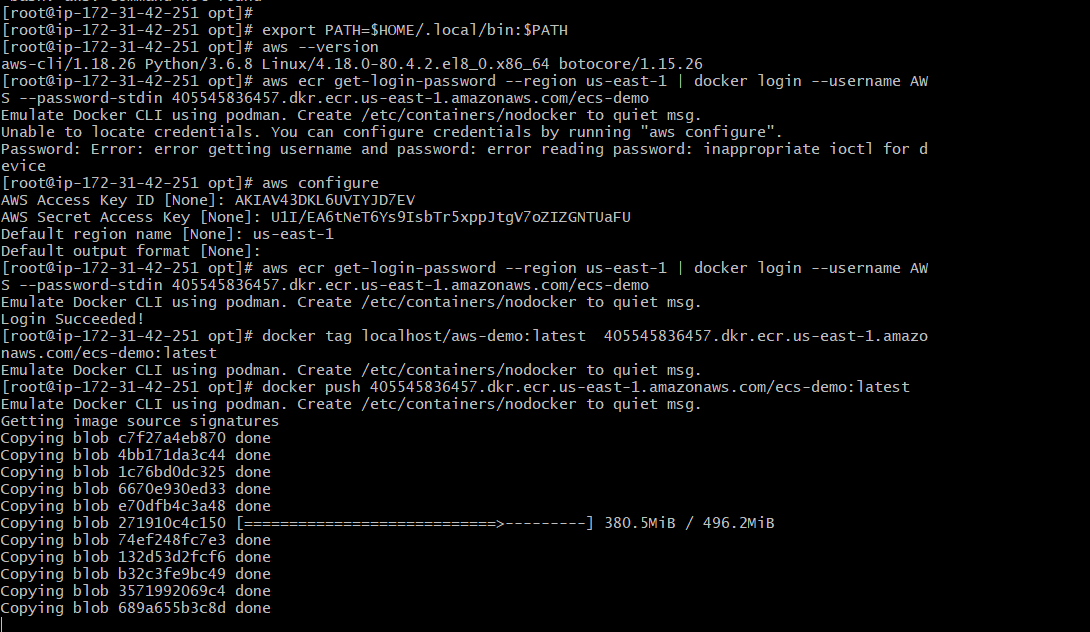
Use the above command to login



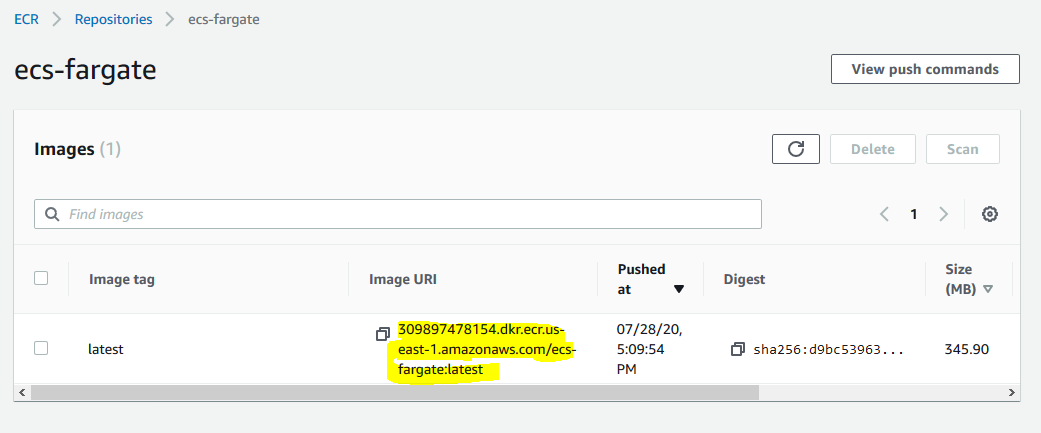
1. Push the Docker image to ECR

Clone the application (we can take any application with working Docker file )

git clone <https://github.com/yeasy/simple-web.git>

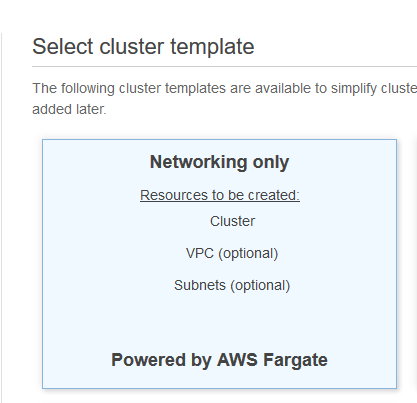




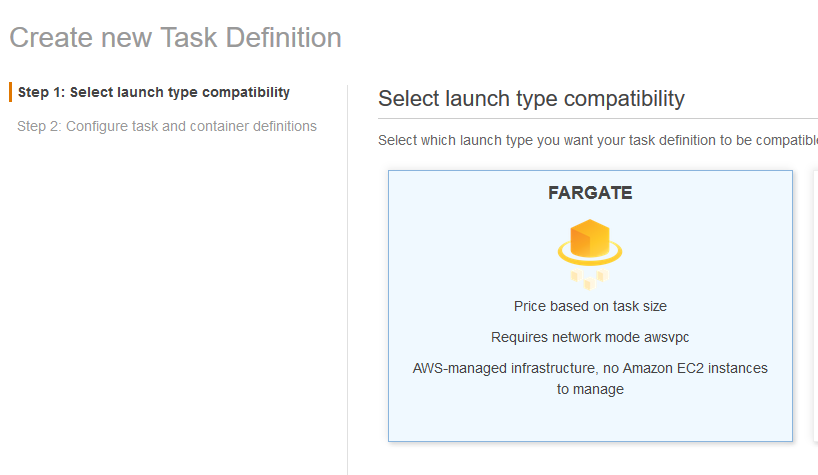


1. Create a Cluster

I used default VPC

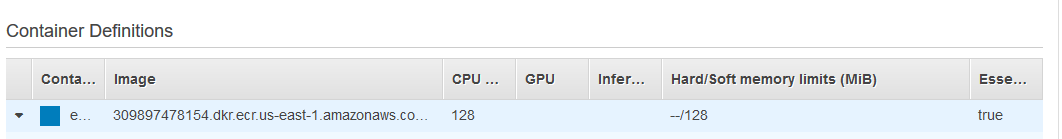


1. create task definition



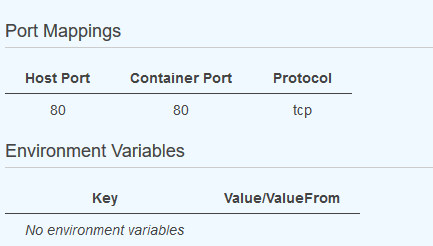
Give task definition name ,it will automatically assigning task role and network mode .

Task size need to selected appropriately and in container section give the image that has been pushed.



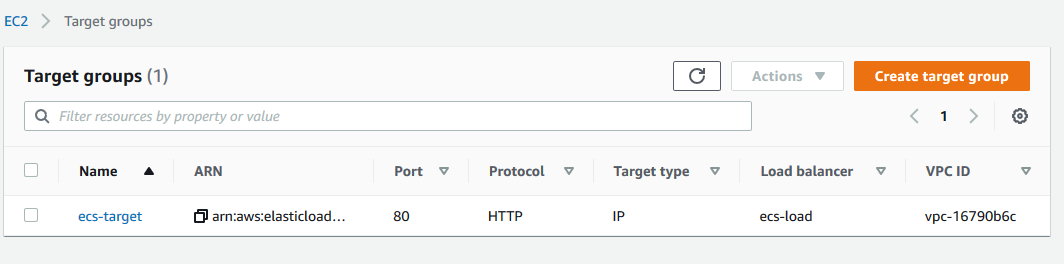
Here we have to select container port in which our application is mapped

My web application will listen to 80 port so selecting container port also 80

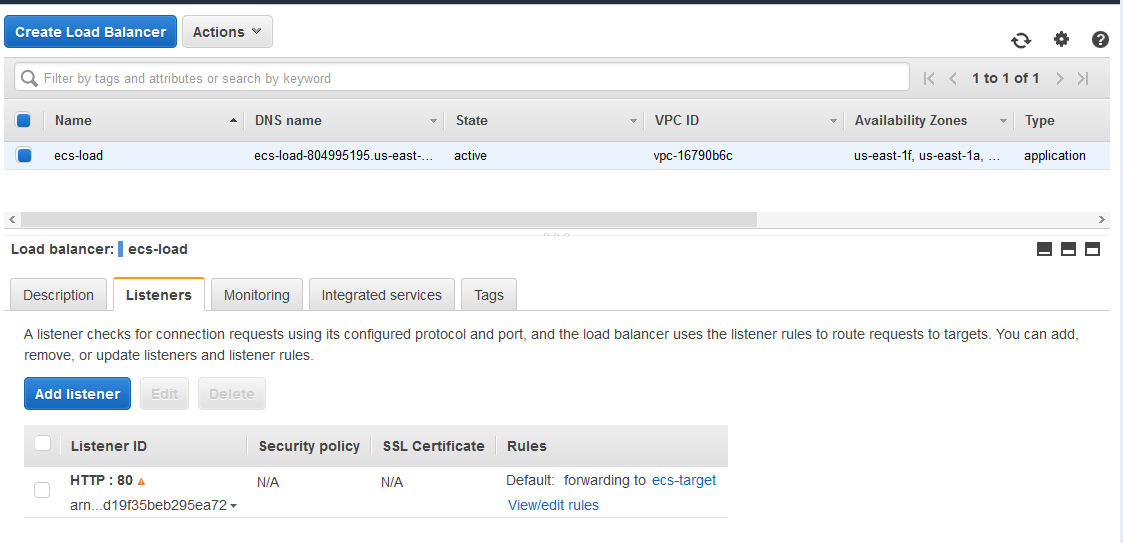


After creating ECR,cluster,taskdefinition we have to create service to maintain the tasks,before creating service we have to create load balancer .

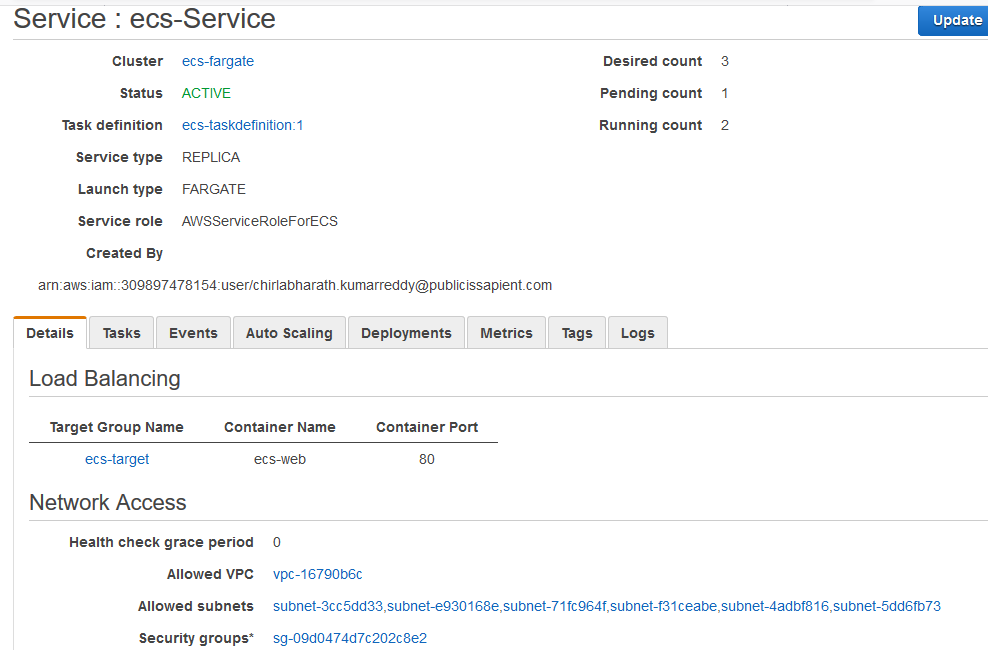
Create a target group with port 80 and target type IP

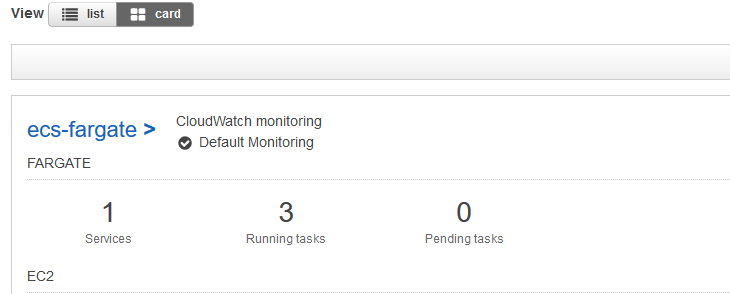


Create a load balancer which listen to the target group crated

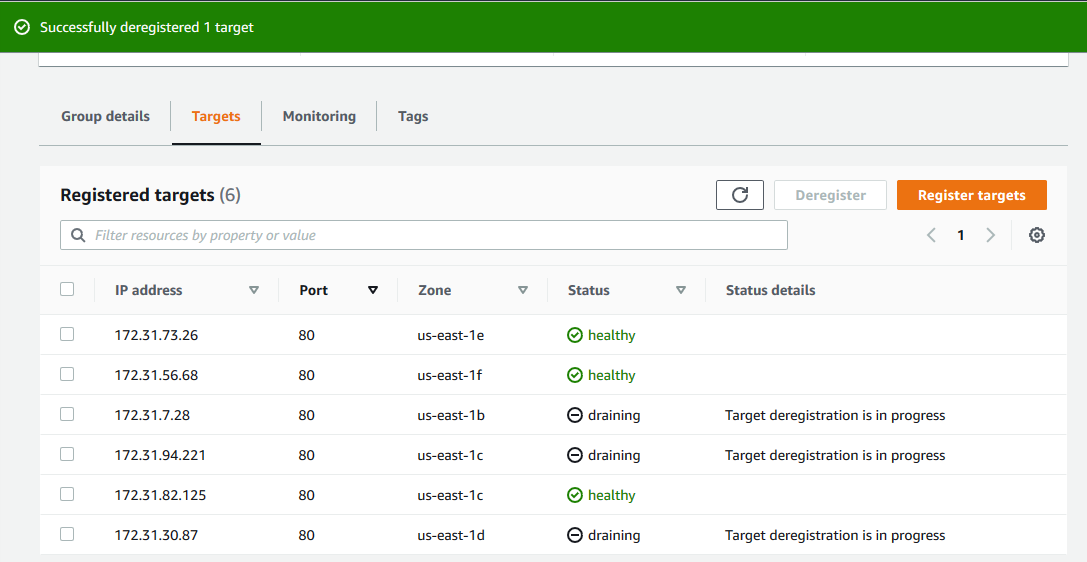


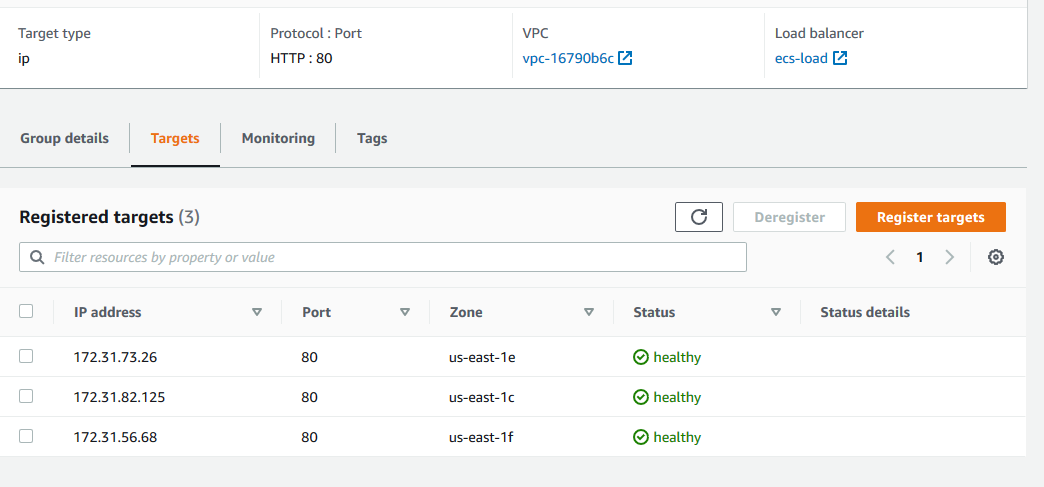
Now create service and select the number of tasks ,load balancer (created one )

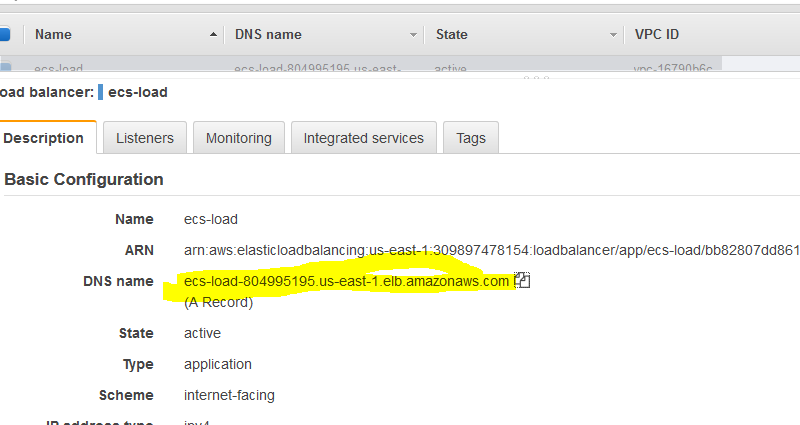


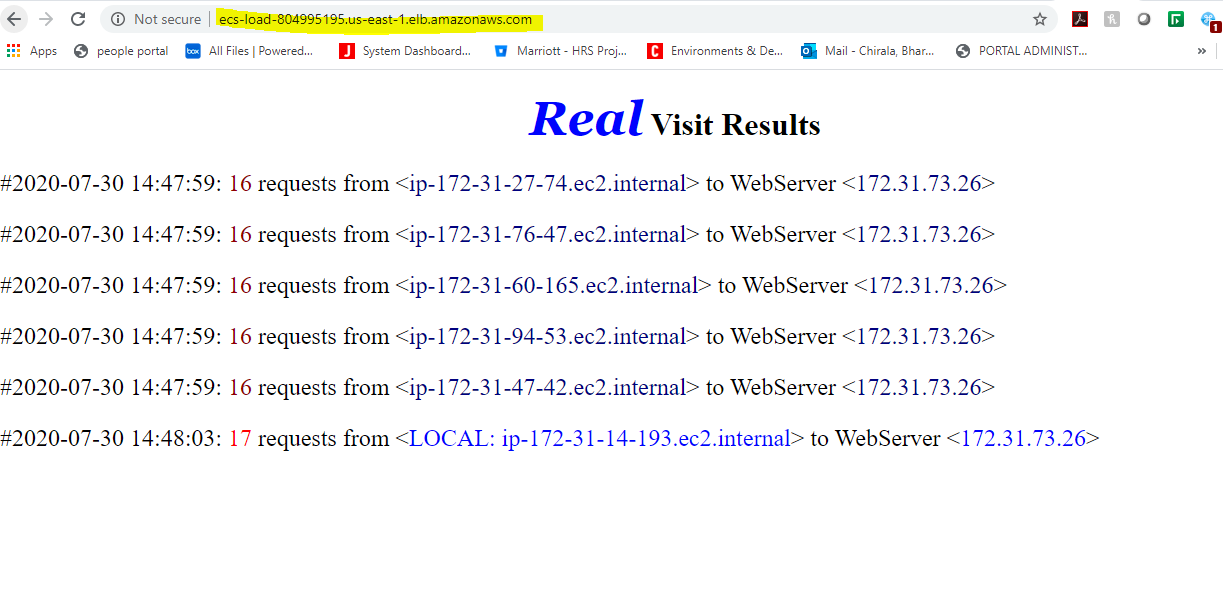


Check the target group is healthy or not ,while creating security groups we need to make sure we opened the correct ports for the application or not .







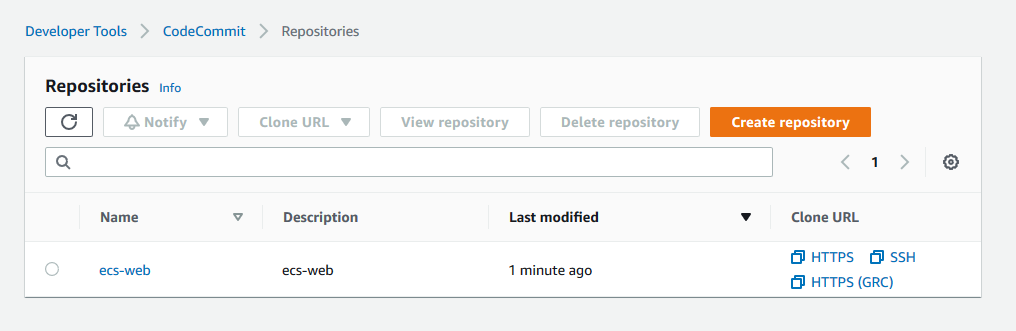


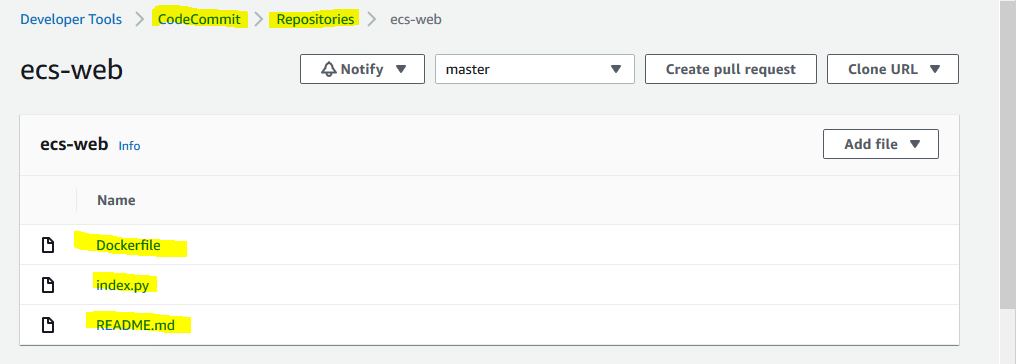
**Code Commit**

Put the sample web source files into CodeComit

Steps

1. Create repository
2. Prepare environment
3. Clone the repository to local PC
4. Fetch sourcecode of sampleweb from github to local PC
5. Add source to cloned repository
6. Commit and push to code commit





**Code Build**

1. Create IAM role and policy
2. Create CodeBuild project
3. Link to codecommit repo
4. Define build environment
5. Create build specification

## create IAM role and policy

CodeBuild will assume that role and therefore needs certain permissions, especially in our case permissions for ECR.

In IAM section of AWS console

\* click \_Policies\_

\* click \_create Policy\_ ==> choose \*\*JSON\*\*

\* paste the following JSON as policy content

```

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "CloudWatchLogsPolicy",

"Effect": "Allow",

"Action": [

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Resource": [

"\*"

]

},

{

"Sid": "CodeCommitPolicy",

"Effect": "Allow",

"Action": [

"codecommit:GitPull"

],

"Resource": [

"\*"

]

},

{

"Sid": "S3GetObjectPolicy",

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:GetObjectVersion"

],

"Resource": [

"\*"

]

},

{

"Sid": "S3PutObjectPolicy",

"Effect": "Allow",

"Action": [

"s3:PutObject"

],

"Resource": [

"\*"

]

},

{

"Sid": "ECRPullPolicy",

"Effect": "Allow",

"Action": [

"ecr:BatchCheckLayerAvailability",

"ecr:GetDownloadUrlForLayer",

"ecr:BatchGetImage"

],

"Resource": [

"\*"

]

},

{

"Sid": "ECRAuthPolicy",

"Effect": "Allow",

"Action": [

"ecr:GetAuthorizationToken"

],

"Resource": [

"\*"

]

},

{

"Sid": "S3BucketIdentity",

"Effect": "Allow",

"Action": [

"s3:GetBucketAcl",

"s3:GetBucketLocation"

],

"Resource":

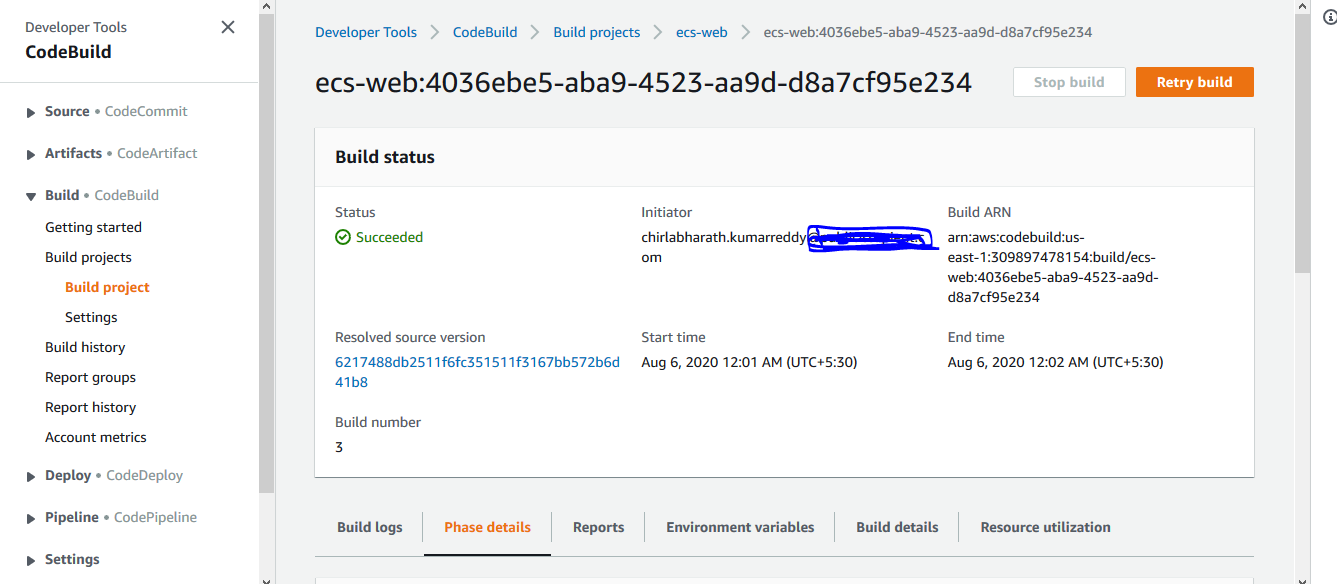
"\*"

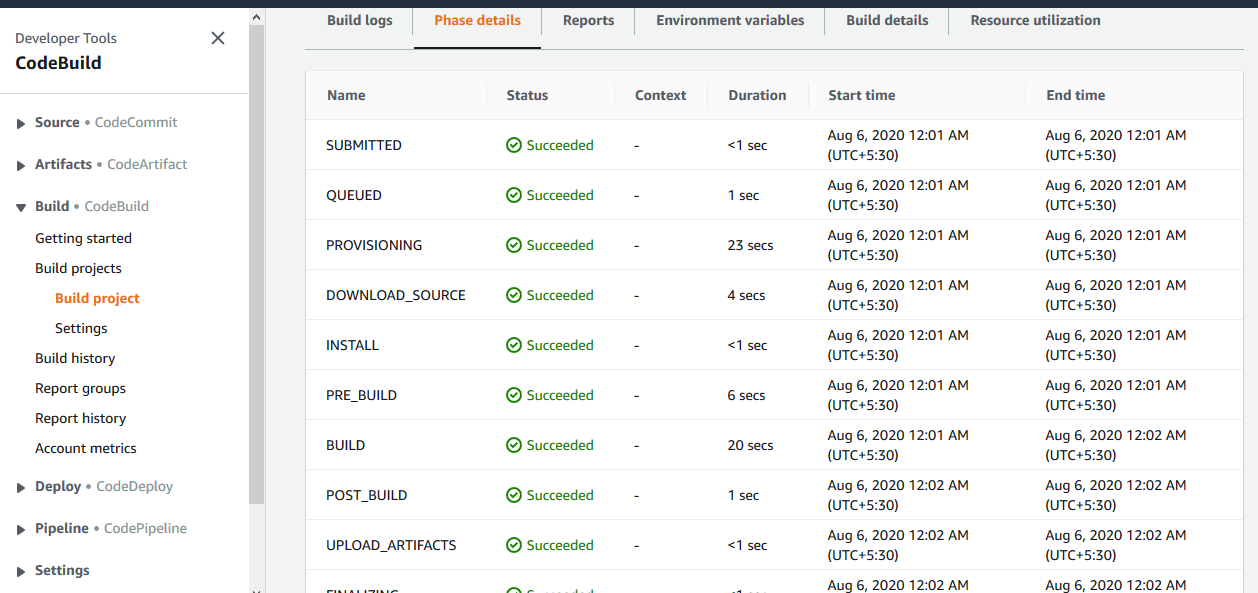
}

]

}

```





CodePipeline

1. Update buildspec.yml to create an artifact,containing container name and image
2. Create codepipeline
3. Link to ecs-web CodeCommit repository
4. Link to ecs-web CodeBuild Project
5. Configure Amazon ECS as Deploy provider