React App Containerized and Pushed to Amazon ECR locally and using Jenkins for complete automation

Part 1 - prepare docker image

Forked project to my repo:
https://github.com/yrenamm/nfx-react-clone

Create new working repo:

- **mkdir nfx-react-clone**

Clone repo to my local pc

- **git clone https://github.com/yrenamm/nfx-react-clone.git**

Remove yarn.lock

- **rm -rf yarn.lock**

Install dependencies listed in the package.json file of your project. (Executing it within the directory of your project where the package.json file is located, yarn.lock must be created)

- **nvm use 16.10.0**
- **yarn install**

Run application:

- **yarn start**

Build docker image:

- **docker build -t yrenamm/nfx-react .**

Run docker container

- **docker run -itd -p 3000:80 yrenamm/nfx-react:latest**
- **docker ps**

http://localhost:3000/

Stop docker container:

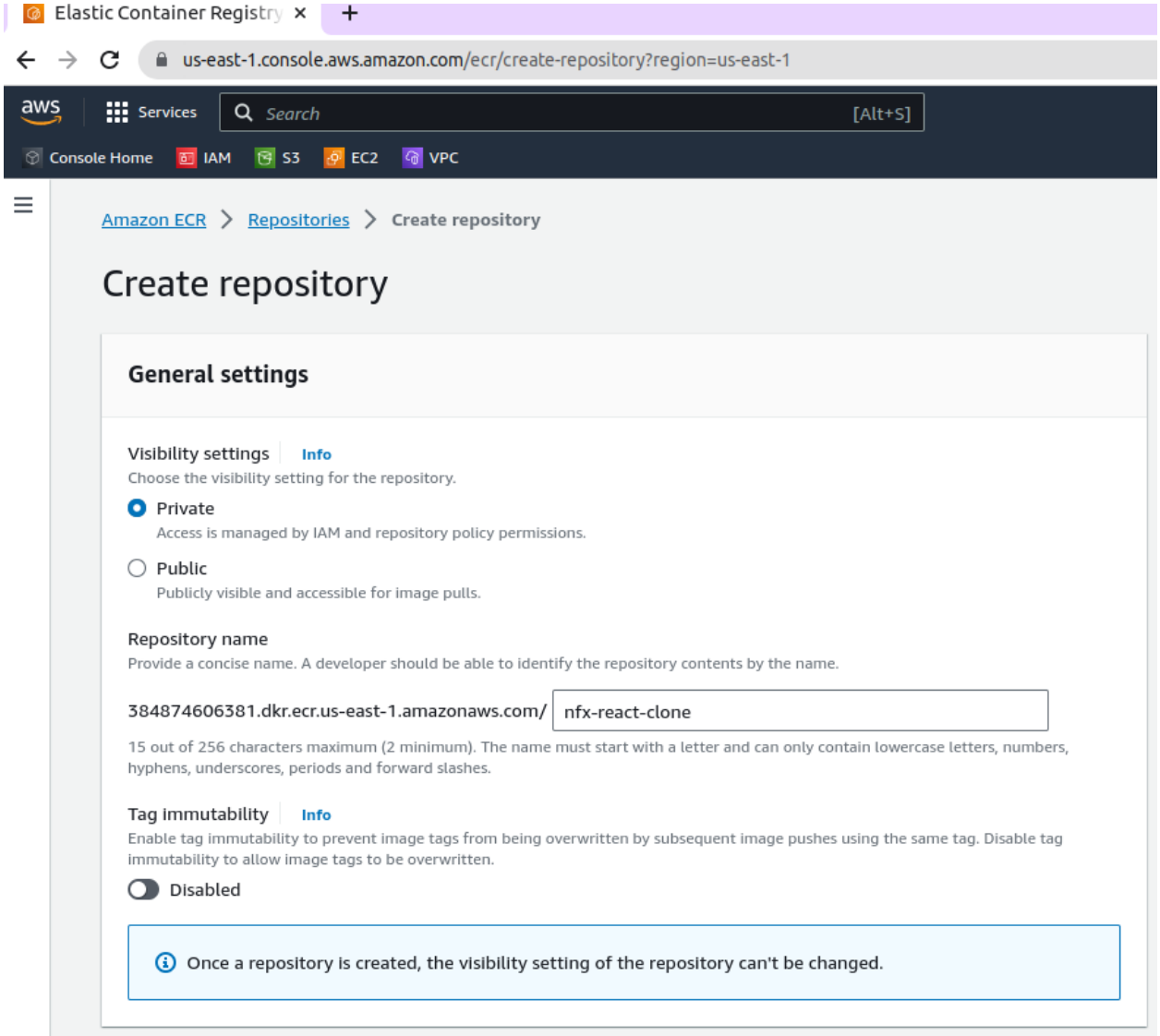
- **docker stop 8767cdf2f048**

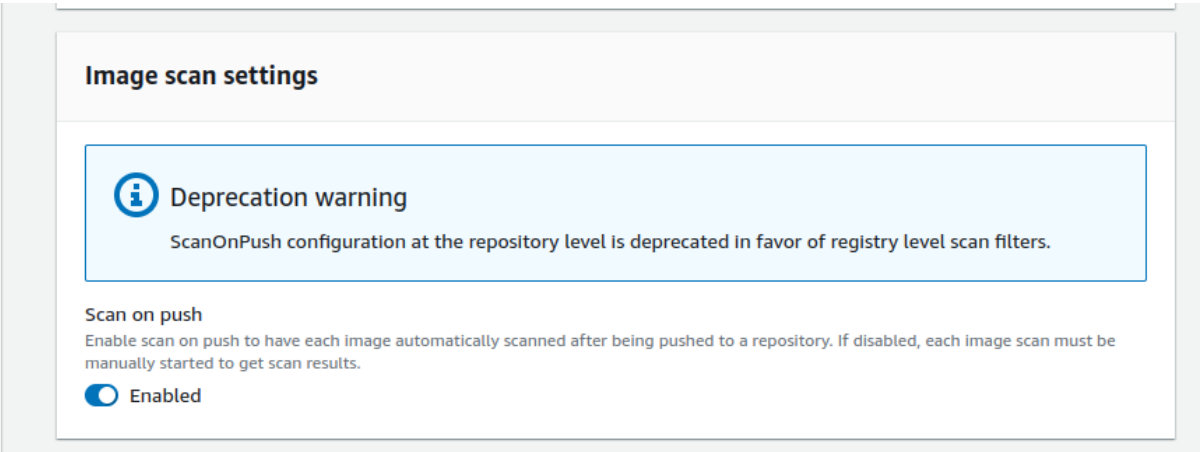
Remove container:

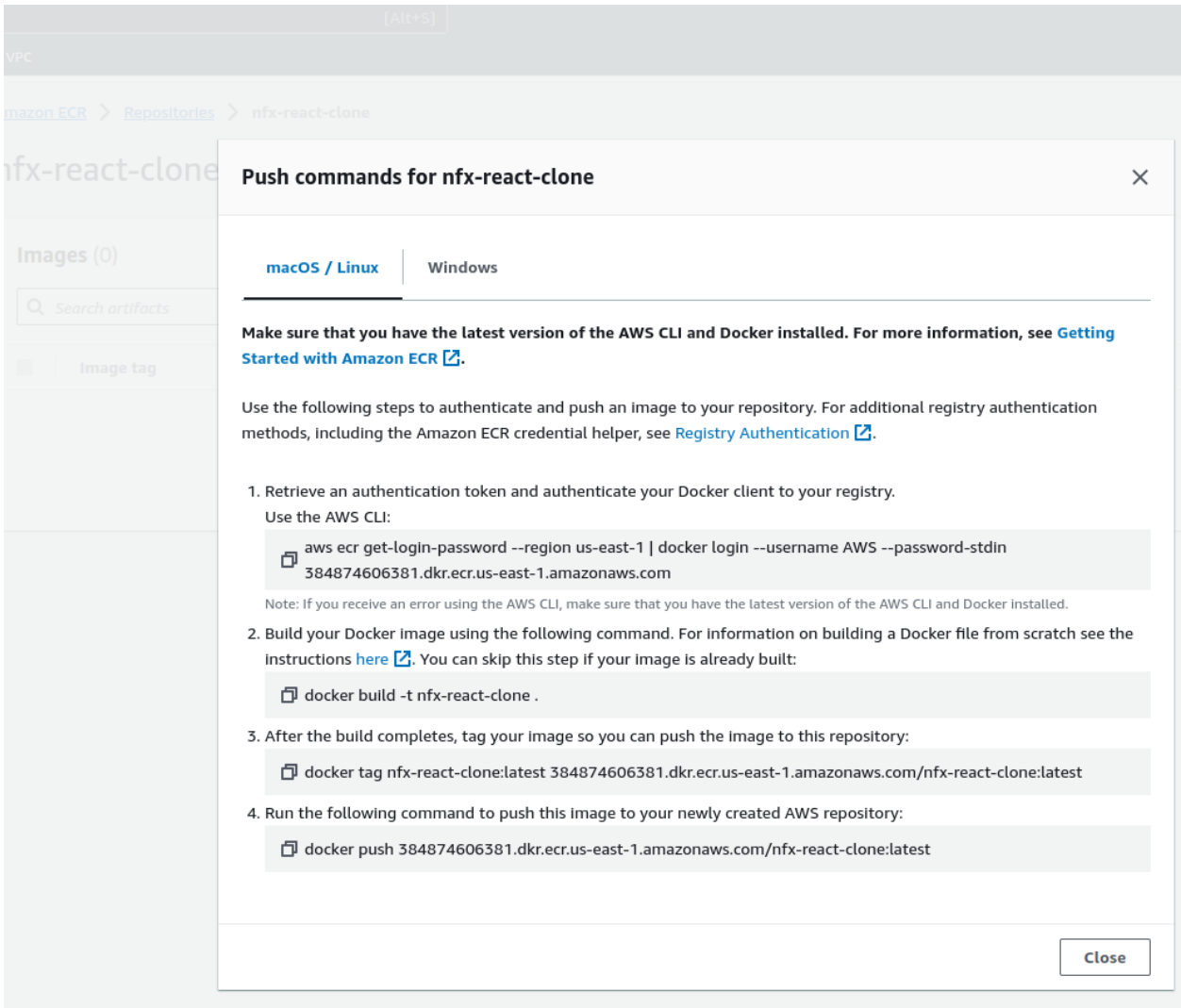
- **docker remove 8767cdf2f048**
- **docker ps** (will not appear)

Part 2 - deploy docker image to AWS

Search for AWS Elastic Container Registry







Retrieve an authentication token and authenticate your Docker client to your registry.
Use the AWS CLI:

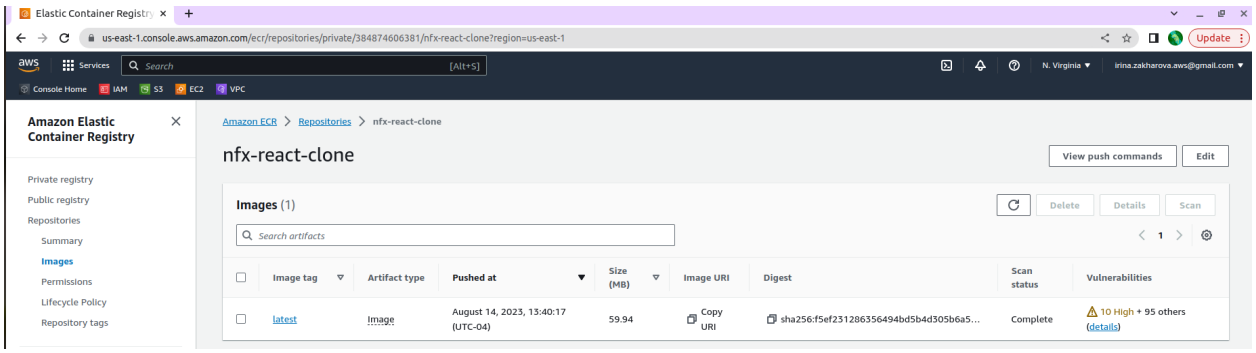
- **aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 384874606381.dkr.ecr.us-east-1.amazonaws.com**

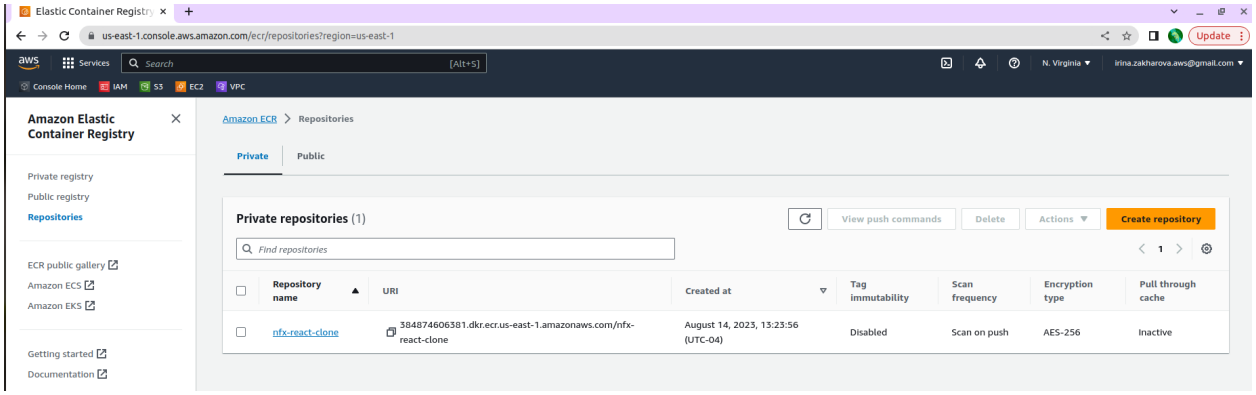
After the build completes, tag your image so you can push the image to this repository:

- **docker tag yrenamm/nfx-react:latest 384874606381.dkr.ecr.us-east-1.amazonaws.com/nfx-react-clone:latest**

Run push command:

- **docker push 384874606381.dkr.ecr.us-east-1.amazonaws.com/nfx-react-clone:latest**





Part 3 - Automation with Jenkins

Work with .git

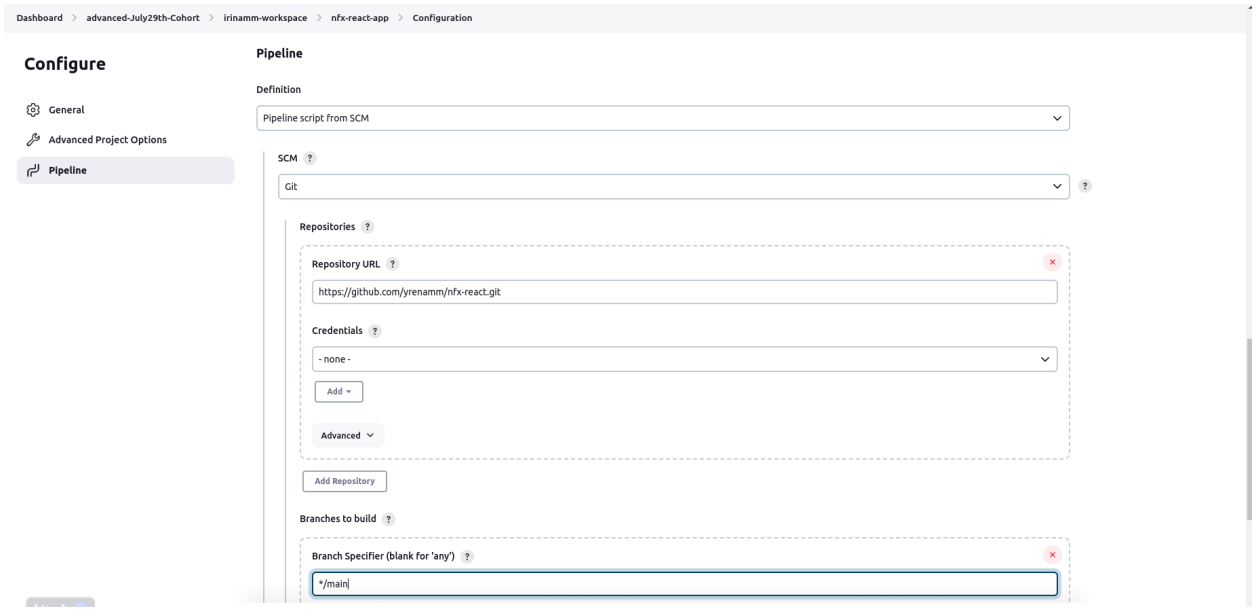
- **rm -rf .git**
- **git status**
- **git init**
- **git status**
- **git add .**
- **git commit**

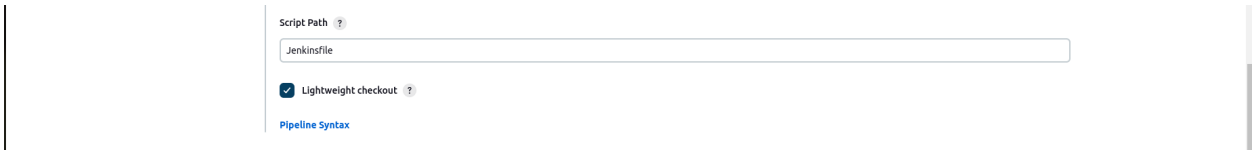
Create new repository **nfx-react** and:
https://github.com/yrenamm/nfx-react

- **git remote add origin https://github.com/yrenamm/nfx-react.git**
- **git branch -M main**
- **git push -u origin main**

Create Jenkins job:

Jenkins -> New Item -> nfx-react-app -> pipeline:

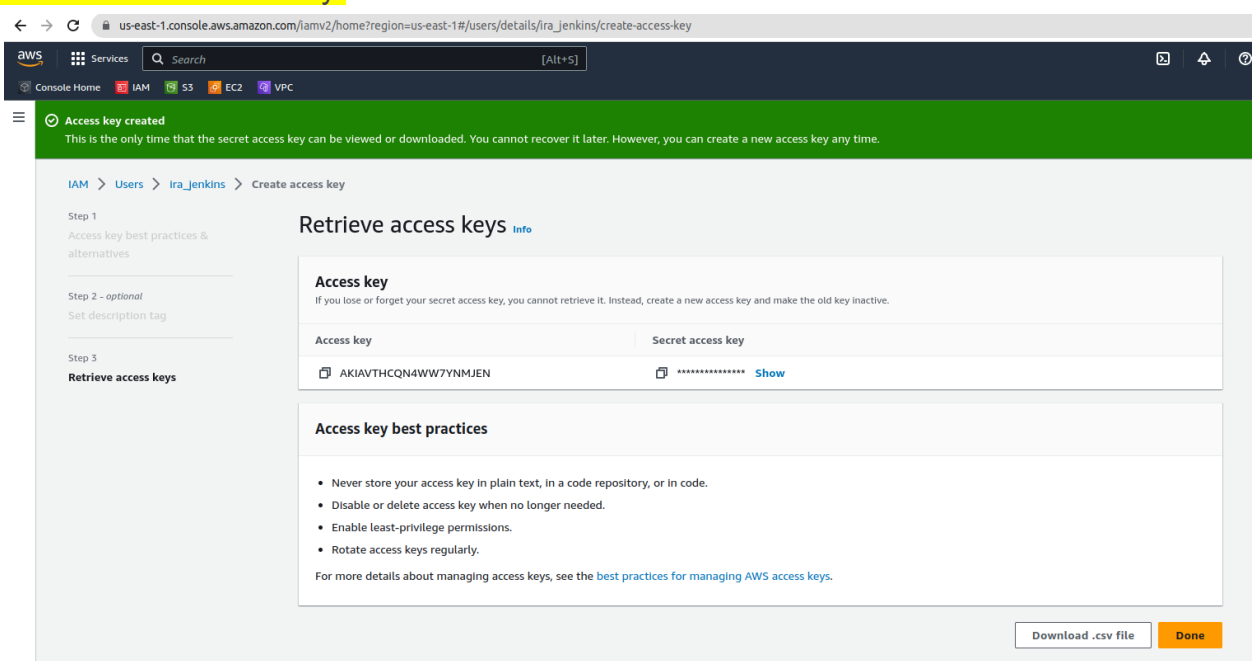




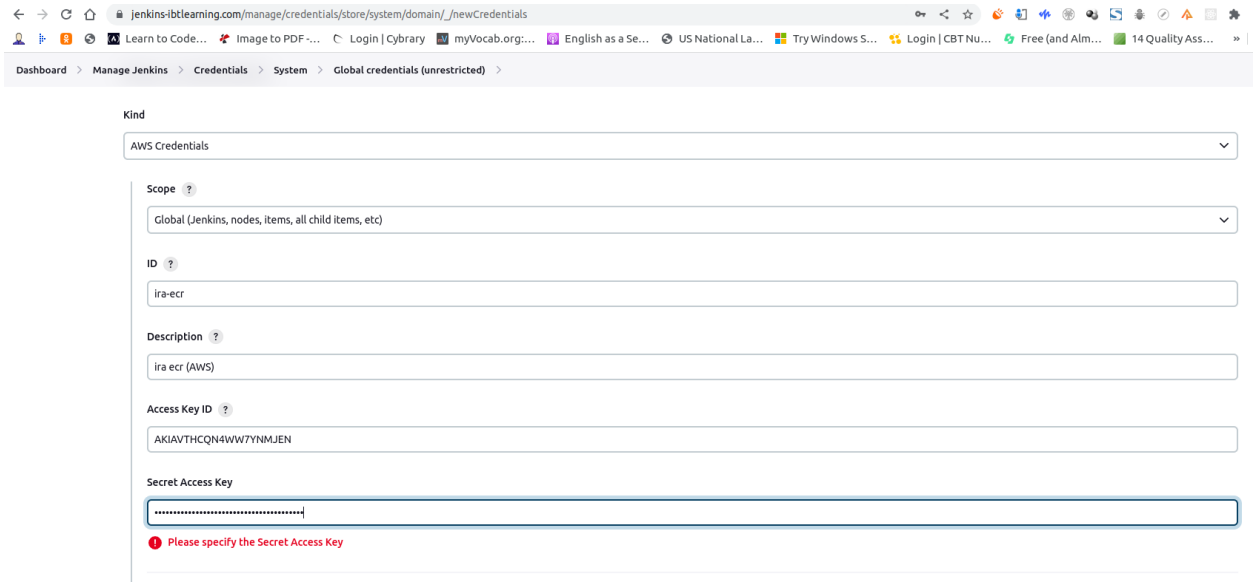
Jenkinsfile:

```
pipeline {
    agent any
    options {
        timeout(time: 20, unit: 'MINUTES')
    }
    stages{
        // NPM dependencies
        stage('pull npm dependencies') {
            steps {
                sh 'npm install'
            }
        }
       stage('build Docker Image') {
            steps {
                script {
                    // build image
                    docker.build("384874606381.dkr.ecr.us-east-1.amazonaws.com/nfx-react-clone:v1.0.0.RELEASE")
                }
            }
        }
        stage('Trivy Scan (Aqua)') {
            steps {
                sh 'trivy image --format template --output trivy_report.html 384874606381.dkr.ecr.us-east-1.amazonaws.com/nfx-react-clone:v1.0.0.RELEASE'
            }
       }
        stage('Push to ECR') {
            steps {
                script{
                    //https://<AwsAccountNumber>.dkr.ecr.<region>.amazonaws.com/netflix-app', 'ecr:<region>:<credentialsId>
                    docker.withRegistry('https://384874606381.dkr.ecr.us-east-1.amazonaws.com/nfx-react-clone', 'ecr:us-east-1:ira-ecr') {

                    // build image
                    def myImage = docker.build("384874606381.dkr.ecr.us-east-1.amazonaws.com/nfx-react-clone:v1.0.0.RELEASE")
                    // push image
                    myImage.push()
                    }
                }
            }
        }

    }
}
```

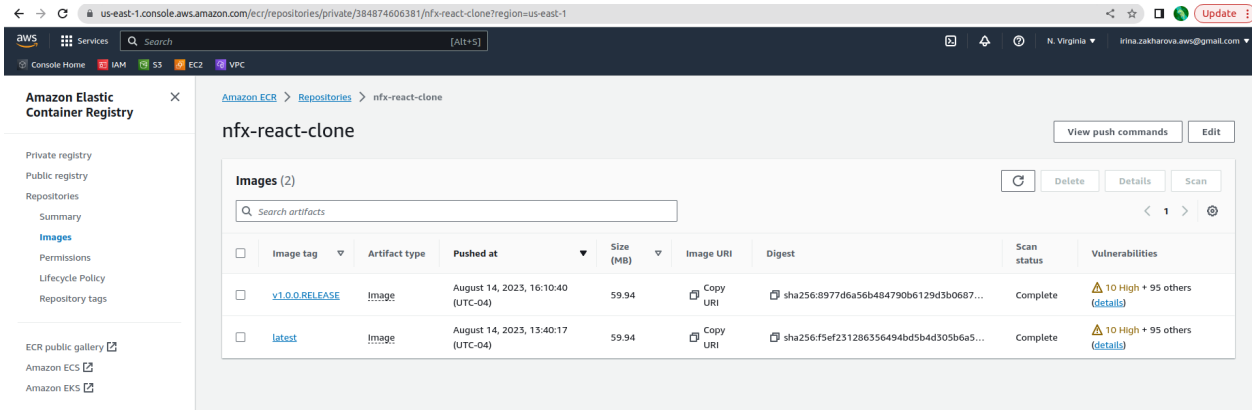
AWS create access key:



Jenkins add credential: (AWS credentials as a type)



Run Jenkins job and check in AWS:



1. Fork or Clone Repository
2. cd into the app directory
3. Install dependencies
   use Yarn install
4. Run app locally
   use Yarn start
5. Check the app on your Browser using localhost or server ip and the port it is listening on.

Dockerize Application

6. Create Dockerfile for your react app, use nginx to serve the static assets

7. Build Docker Image by using the docker build command
   a. docker build -t repoName/imageName:tag .

8. Run the container by using the docker run command to confirm application still works
  a. docker run -d -p 3000:80 nameOfImage
  b. Go to Browser and confirm your containerised app works using your localhost or server IP and the port e.g localhost:3000

9. Log onto your AWS Console and type ECR on the search bar and click on it. ECR = Elastic Container Registry

10. Create a repository,
   a. click on private
   b. type in repo name
   c. Enable Scan on Push
   and Click on Create

11. Click or enter into your repository and click on the view push commands

12. Go to your terminal and follow each steps provided by aws

13. Congratulations, You have made your first ECR push

AUTOMATE THE ENTIRE STEPS TO PUSH YOUR IMAGE TO ECR USING JENKINS

1. Create an ECR key in Jenkins Key Storage using your aws access and secret key
2. Create a Jenkinsfile or Add one
3. Go to your Jenkins workspace and Create a Pipeline Job
4. Build your Job
5. Go to ECR and confirm Image was succesfully pushed