Jenkins Learning Journey – Day 2

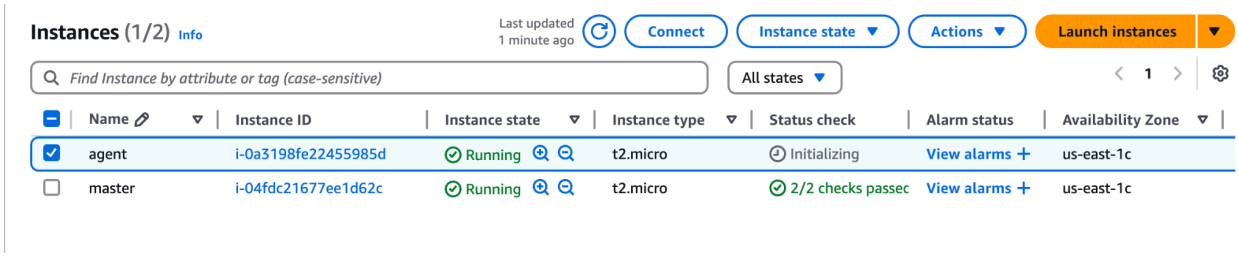
What I Learned Today

Today, I took my Jenkins learning to the next level by working on a small project using a **Declarative Pipeline**. This project involved setting up a **Jenkins Master** and **Agent**, and using **Docker** for building and deploying a project.

Setup Steps

1. Jenkins Master and Agent Setup

- **Created Two EC2 Instances:**
    - One as the **Jenkins Master** and the other as the **Jenkins Agent**.
- **Master EC2 Setup:**
    - Installed **Java** and **Jenkins** on the master instance.
    - Created an **SSH key pair** for secure communication.
- **Agent EC2 Setup:**
    - Installed **Java** (Jenkins not needed on agent).
    - Copied the **public key** from the master EC2 to the agent for SSH authentication.



2. Docker Installation on the Agent EC2

Installed Docker:

```
sudo apt-get install docker.io
```

Installed Docker Compose:

```
sudo apt-get install docker-compose
```

Added User to Docker Group (to run Docker without sudo):

```
sudo usermod -aG docker $USER newgrp docker
```

3. Pipeline Setup in Jenkins

- **Created a Pipeline Job** using YAML syntax, which included 3 stages: **Code**, **Build**, and **Deploy**.

```
pipeline {
    agent any

    stages {
        stage('Code') {
            steps {
                Git url: 'https://github.com/your-repository.git',
branch: ‘main’

            }
        }
        stage('Build') {
            steps {
                sh 'docker build -t image-name:latest .'            }
        }
        stage('Deploy') {
            steps {
                sh 'docker run -d -p 8000:8000 image-name:latest'
            }
```

```
        }
    }
}
```

4. Docker Compose Deployment

For **Docker Compose**, I wrote the deployment step in the pipeline like so:

```
stage('Deploy with Docker Compose') {
    steps {
        sh 'docker-compose up -d'
    }
}
```

5. Testing

- Verified that the Docker container was running and the application was deployed on port 8000.