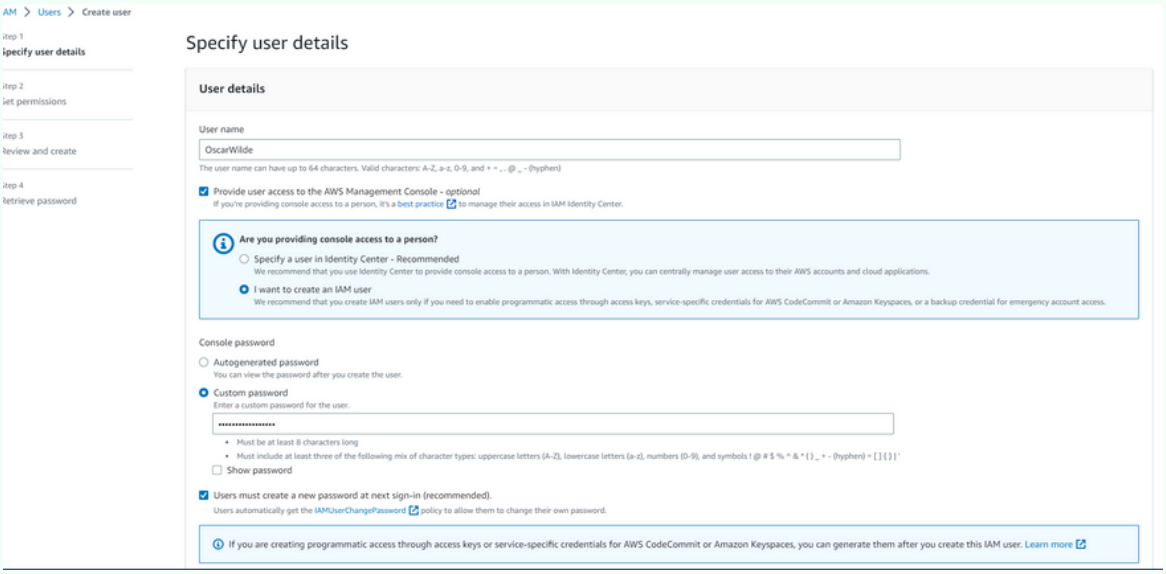
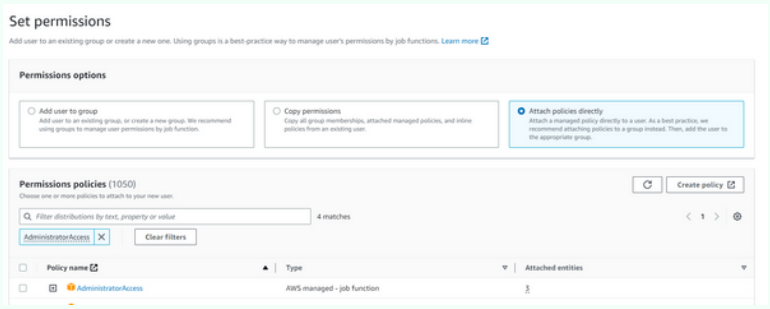
DevOps - SQLABS
Project 1
Mark Kiezhner

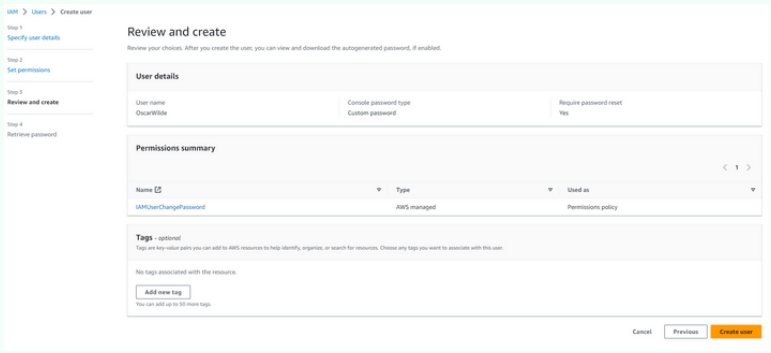
All the code can be found here:

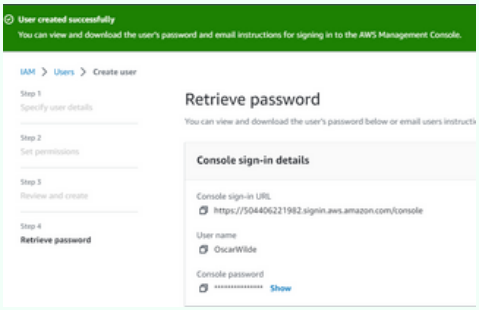
https://github.com/MarkeyBass/todos-docker-compose.git

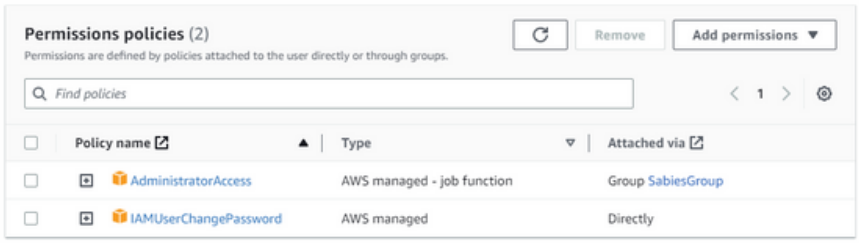
- ***Create an IAM user that you will use for all of the AWS implementations.***



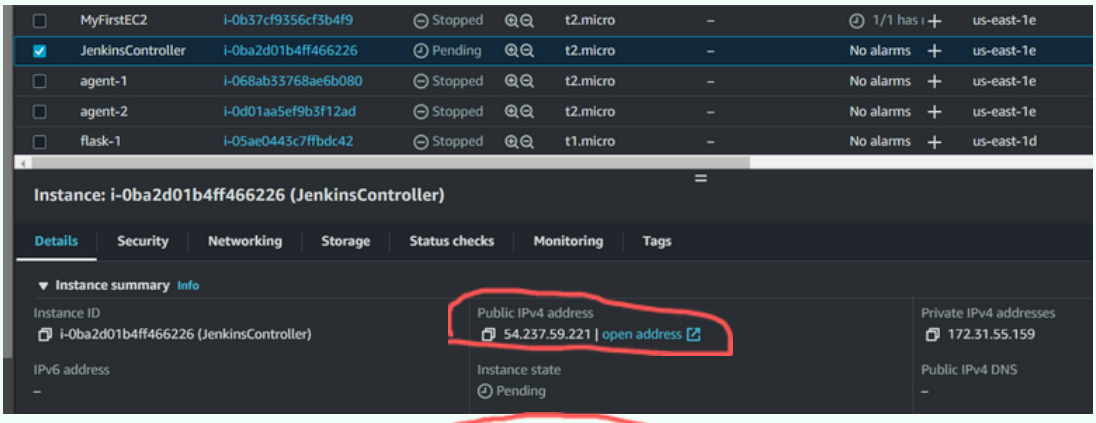


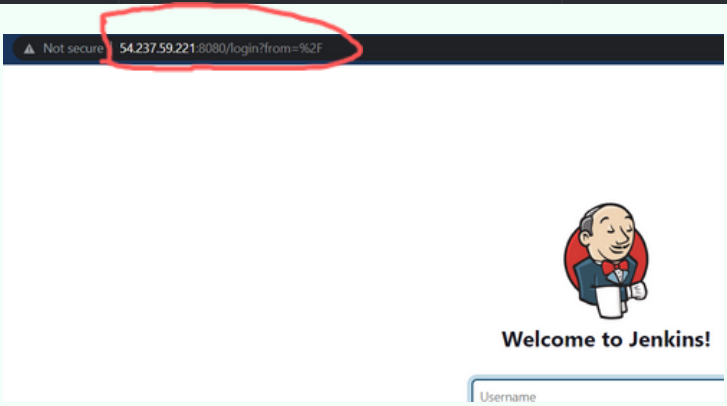






● *Set up a Jenkins server inside a Docker container on an EC2 instance.*





● *Create three different users on your Jenkins server that will be on the same group. You will use only these users for all of the Jenkins implementations.*

JENKINS PLUGINS

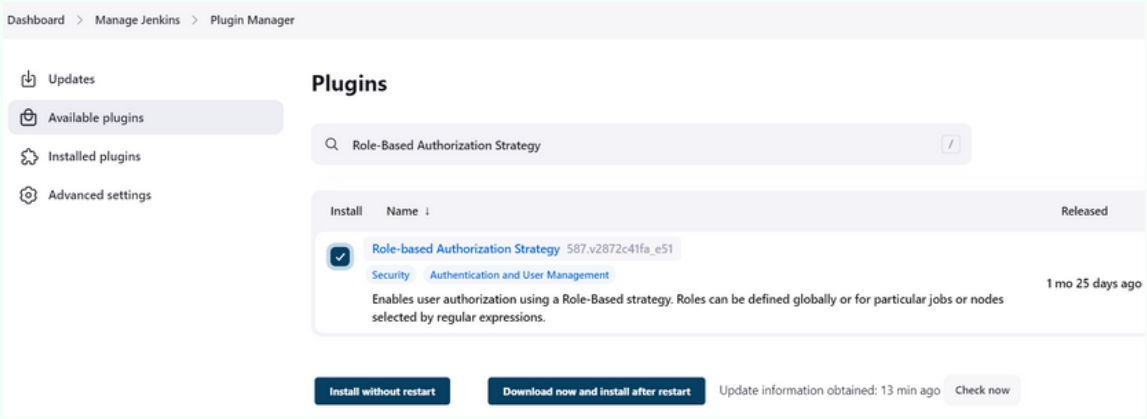
Role-Based Authorization Strategy
---> download without restart of role plugin

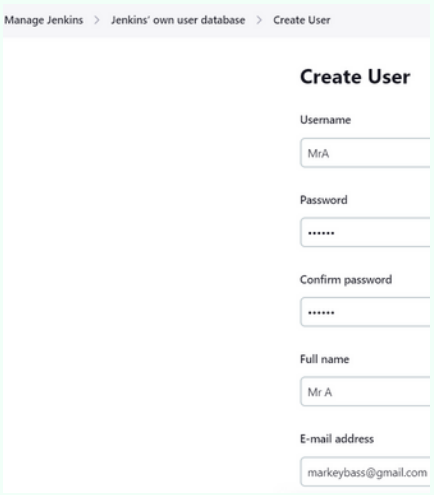
--> Manage Jenkins
---> Configure Global Security
---> Authorization
---> Role-Based Strategy

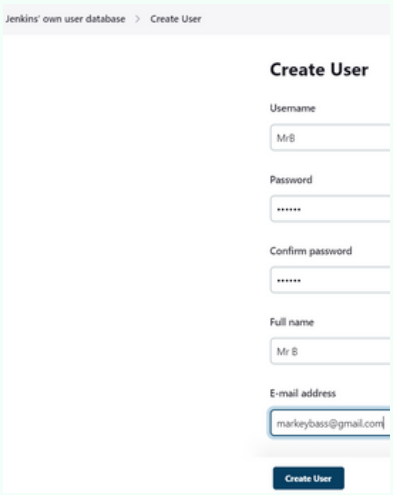
Manage Jenkins
---> Manage and Assign Roles
---> Manage Roles ---> add new role

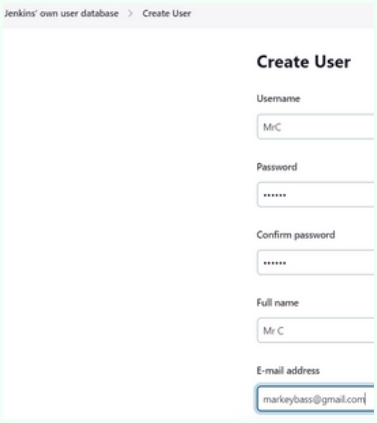
Manage Jenkins
--->Jenkins' own user database
---> create user

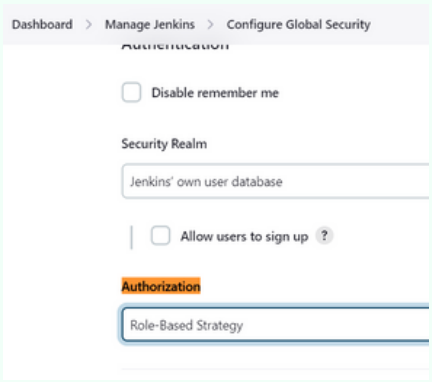
---> Manage and Assign Roles
---> Assign Roles
---> User/group to add (Add the user)

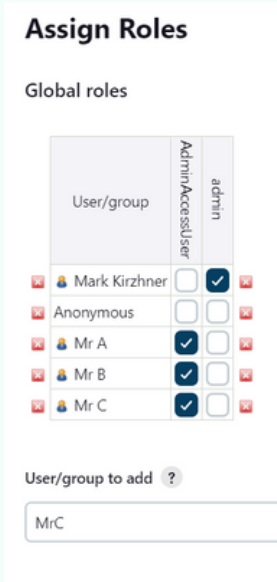






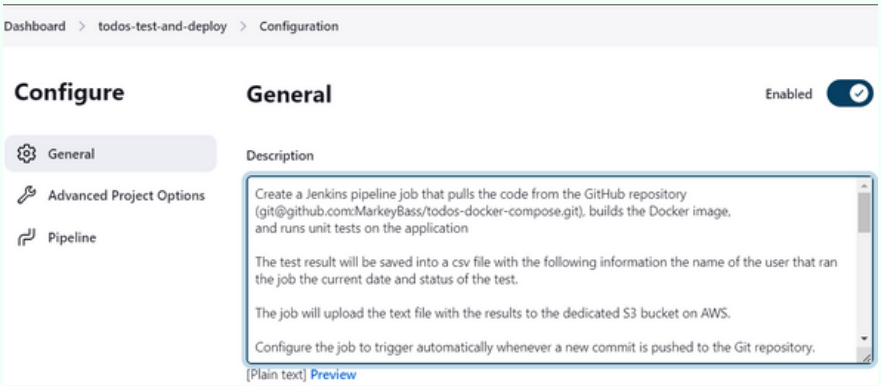


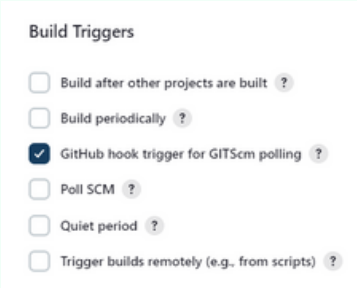




Role base Strategy didn't work. Mabe some BUG - so I Chaned Authorization to Matrix-based security - Now it works

● ***Create a Jenkins pipeline job that pulls the code from the GitHub repository, builds the Docker image, and runs unit tests on the application.***



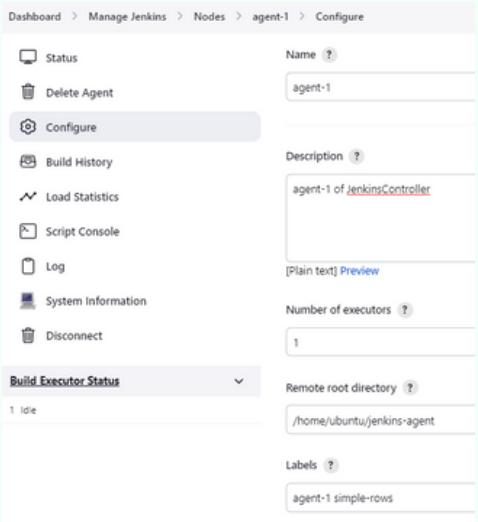


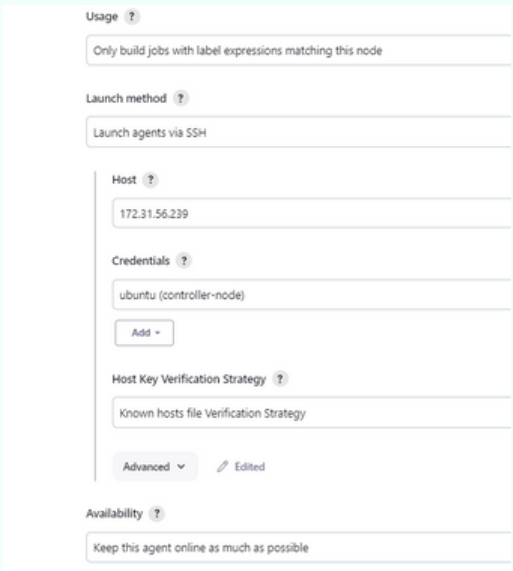
✅ Installong Java on agent-1 and agent-2 EC2 machines
sudo apt-get update
sudo apt install default-jre

✅ Installong Docker on agent-1 and agent-2 machines
https://docs.docker.com/engine/install/ubuntu/

✅ docker check on both instances:
sudo docker run hello-world

inside JenkinsConroller machine:
---
docker exec -it JenkinsController bash
ssh-keygen
cd /var/jenkins_home/.ssh
jenkins@32b8a66b46ff:~/.ssh$ cat id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
-----END OPENSSH PRIVATE KEY-----
add this privet key to the configure credentials --> add this creds ID (controller-node) in the node (agent-1, agent-2, Prod-1-todos, Prod-2-todos) configuration



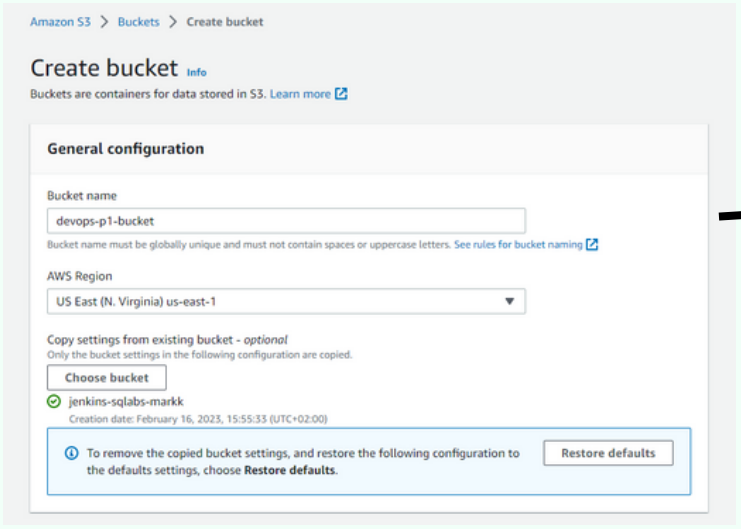


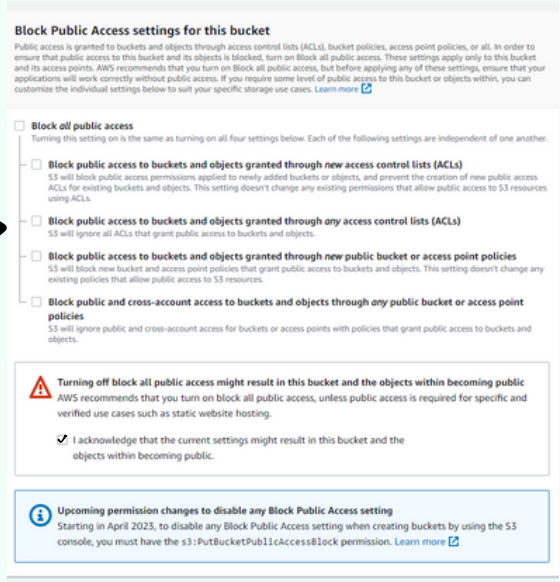
This part of the pipeline the agent connects to GitHub Downloads the repo Performs the tests.

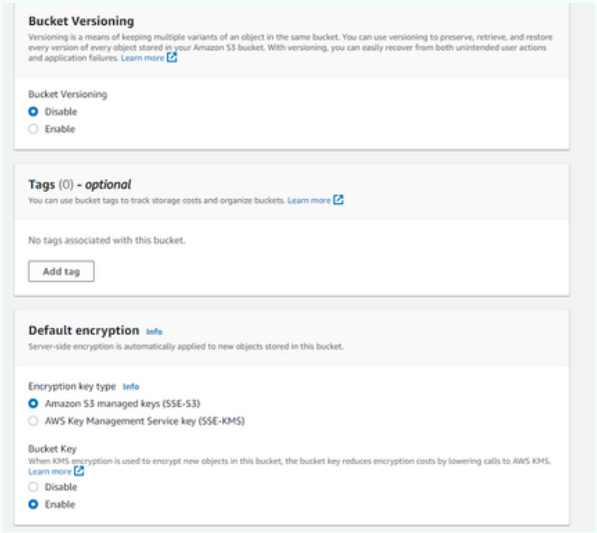
```
pipeline {
  agent {label "agent-1"}

  stages {
    stage('CHECKOUT SCM') {
      steps {
        sshagent(credentials: ['controller-node']) {
          checkout([
            $class: 'GitSCM',
            branches: [[name: 'main']],
            userRemoteConfigs: [[
              url: 'git@github.com:MarkeyBass/todos-docker-compose.git',
              credentialsId: 'controller-node'
            ]]
          ])
        }
      }
    }

    stage('Build') {
      steps {
        sh 'sudo docker compose up -d'
      }
    }

    stage('Test') {
      steps {
        sh 'sudo docker compose exec server python test_server.py > test-results.txt 2>&1'
        sh 'cat test-results.txt'
        script {
          def fileContents = readFile(file: 'test-results.txt', encoding: 'UTF-8').trim()
          def lines = fileContents.split('\n')
          def test_statistics = lines[2].trim()
          def test_status = lines[4].trim()


          def testMap = [:]
          testMap['username'] = "${env.owner}"
          testMap['timestamp'] = new Date().getTime()
          testMap['datetime'] = new Date(testMap['timestamp']).toString()
          testMap['test_statistics'] = test_statistics
          testMap['test_status'] = test_status

          def jsonString = groovy.json.JsonOutput.toJson(testMap)
          writeFile file: 'test-results.json', text: jsonString

          println(jsonString)

          def csvString = "username,timestamp,datetime,test_statistics,test_status\n"
          csvString += "${testMap['username']},${testMap['timestamp']},${testMap['datetime']},${testMap['test_statistics']},${testMap['test_status']}\n"
          writeFile file: 'test-results.csv', text: csvString
        }
      }
    }
```

The test result will be saved into a csv file with the following information the name of the user that ran the job the current date and status of the test.

- ***The job will upload the text file with the results to the dedicated S3 bucket on AWS.***





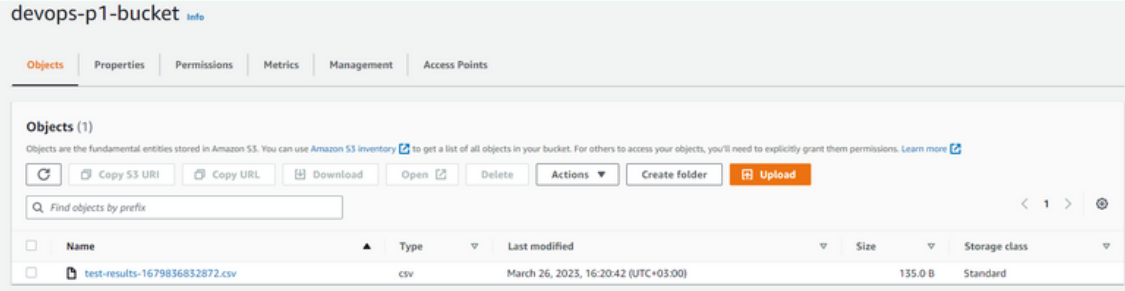


The stage of uploading to S3

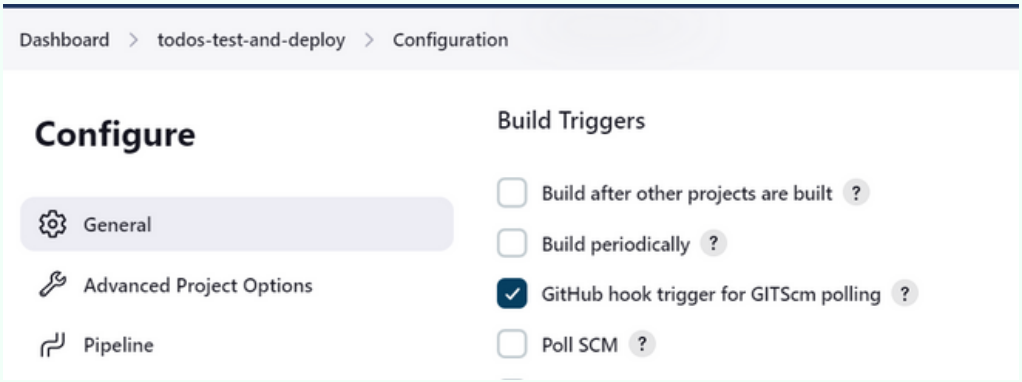
(Instead of the text file I have uploaded the csv file)

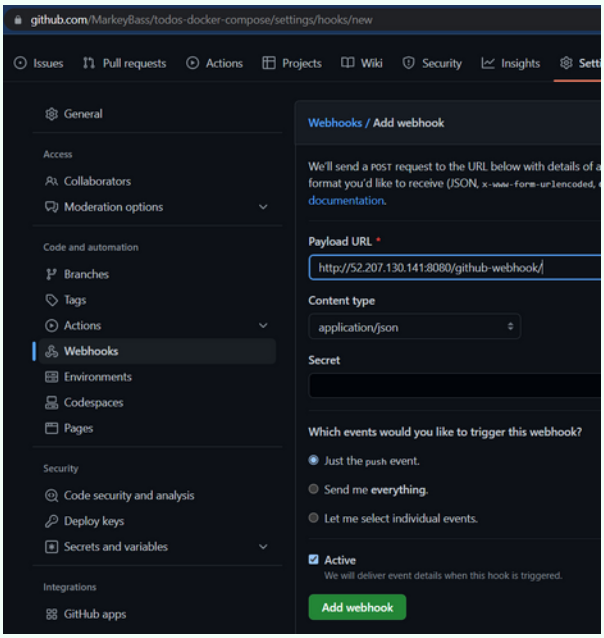
```
stage('Upload Test in csv format to S3') {
   steps {
      script {
         def timestamp
         if (fileExists('test-results.csv')) {
            def fileContents = readFile(file: 'test-results.csv', encoding: 'UTF-8').trim()
            def lines = fileContents.split('\n')
            def csvData = lines[1].trim().split(',')
            timestamp = csvData[1].trim()
         } else {
            error('No test results file found')
         }

         withAWS(credentials: 'awscredentials', region: 'us-east-1') {
            s3Upload(
               file: "test-results.csv",
               bucket: "devops-p1-bucket",
               path: "test-results-${timestamp}.csv"
            )
         }
      }
   }
}
```

The result is in the buchet

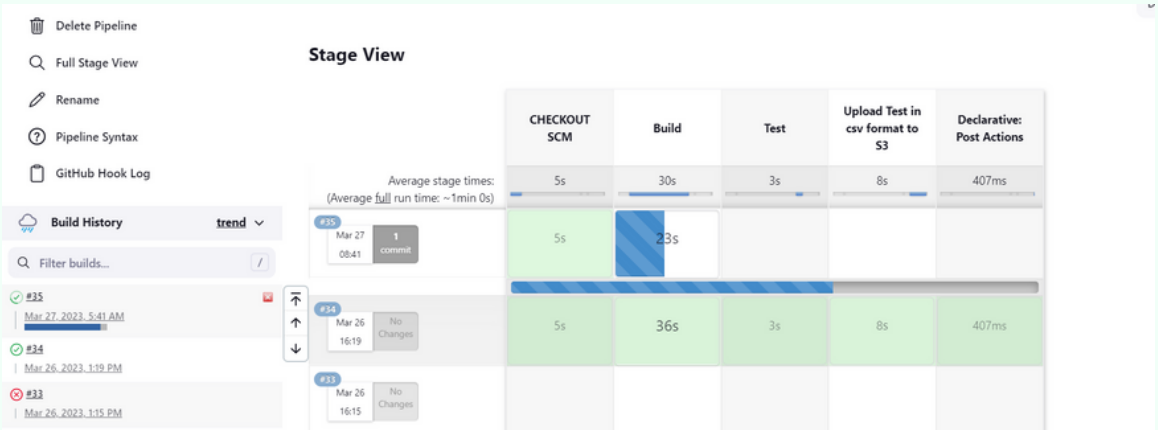


- *Configure the job to trigger automatically whenever a new commit is pushed to the Git repository*

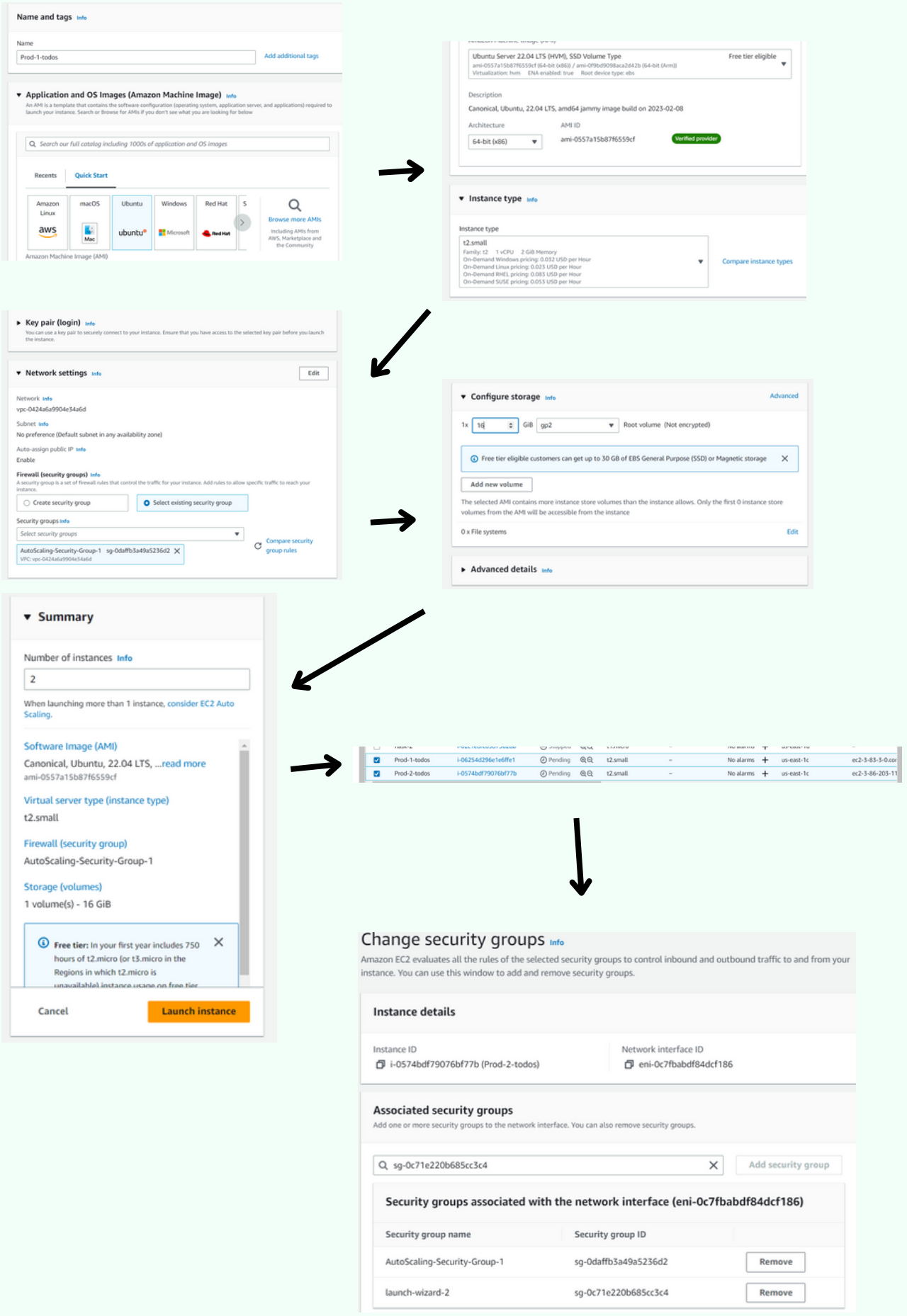





```
marke@DESKTOP-1PI2AB4 MINGW64 /c/dev/DevOps/Projects/todos (main)
$ git push origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 2.43 KiB | 1.22 MiB/s, done.
Total 7 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To github.com:MarkeyBass/todos-docker-compose.git
   826d166..8ff7ef2  main -> main

marke@DESKTOP-1PI2AB4 MINGW64 /c/dev/DevOps/Projects/todos (main)
$
```



- *Set up another EC2 instance to act as the production server.*
- *Create another EC2 production service.*

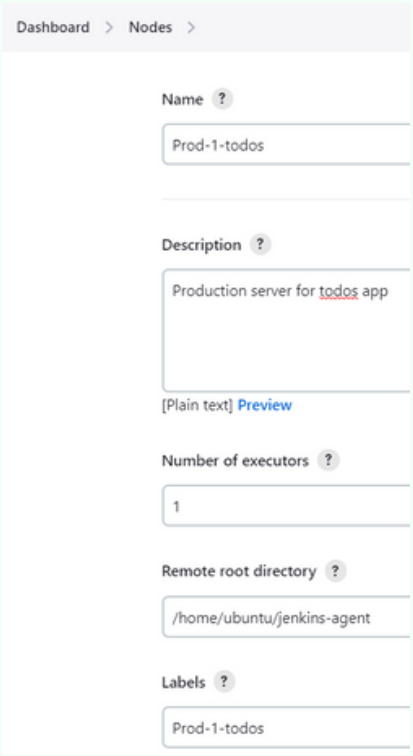


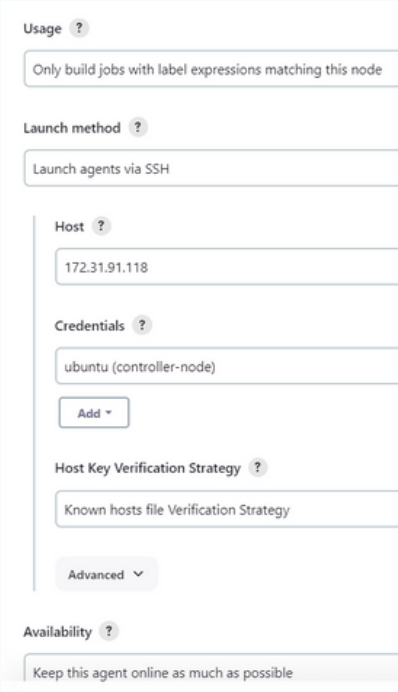
✅ Installing Java on prod-1-todos and prod-2-todos
sudo apt-get update
sudo apt install default-jre

✅ Installong Docker on prod-1-todos and prod-2-todos
https://docs.docker.com/engine/install/ubuntu/

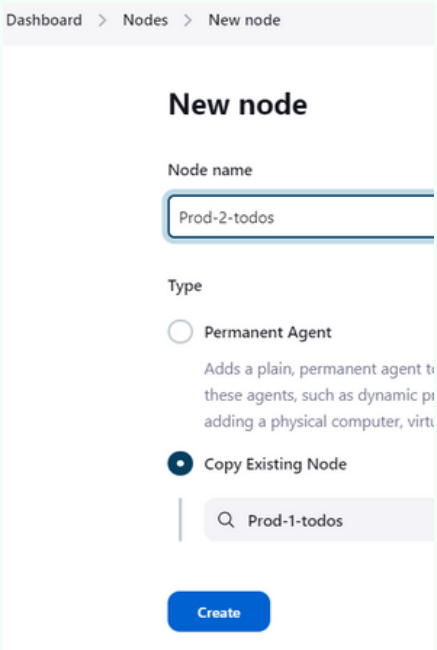
Setting an agent connection between JenkinsController and prod-1-todos and prod-2-todos

Configiring agent:

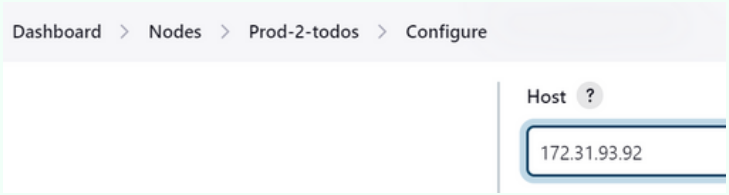




Copy configuration to Prod-2-todos agent



Change ip adress and Labels to Prod-2-todos agent



connect to prod-1-todos and prod-2-todos instances

add to .ssh/authorised_keys the Public Key of JenkinsController container (!important - get it from inside the container)

Get it in advanced from the ---> docker exec -it JenkinsController bash --> jenkins@32b8a66b46ff:~/.ssh$ cat ~/.ssh/id_rsa.pub

```
ubuntu@ip-172-31-55-159:~$ sudo docker exec -it JenkinsController bash
jenkins@32b8a66b46ff:/$ cat ~/.ssh/id_rsa.pub~
cat: /var/jenkins_home/.ssh/id_rsa.pub~: No such file or directory
jenkins@32b8a66b46ff:/$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDWR0xewaR16ucGCg9xX/aOBDBiyhoKJoq
je8i4jhs4bah/KCaoEsR0EfXCmLOjUE/XIAnOa7OnHKrhxnaPatyUeCZN2OlEzAtQ6fP0kN
rHGFTV8KsxB862kogWR0vDZSmkYwJWmuFq8/6DAQlAlU4fAx/5fKjU9hNLHgINcMZPm9FX2
```

```
Last login: Mon Mar 27 06:46:41 2023 from 2.52.151.17
ubuntu@ip-172-31-91-118:~$ cd ~/.ssh
ubuntu@ip-172-31-91-118:~/.ssh$ nano authorized_keys
ubuntu@ip-172-31-91-118:~/.ssh$
```

```
Last login: Mon Mar 27 06:47:37 2023 from 2.52.151.17
ubuntu@ip-172-31-93-92:~$ cd ~/.ssh
ubuntu@ip-172-31-93-92:~/.ssh$ nano authorized_keys
ubuntu@ip-172-31-93-92:~/.ssh$
```

THEN

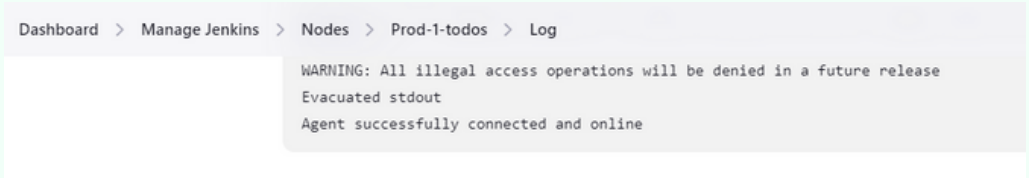
---> conect from the Jenkins Container (JenkinsController) via ssh to the agent (prod-1-todos and prod-2-todos) and leave a fingerprint

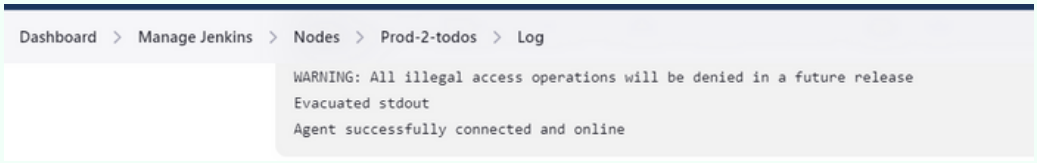
Only now after leaving a fingerprint the JenkinsController will be able to connect to its agent

```
jenkins@32b8a66b46ff:/$ ssh ubuntu@172.31.91.118
The authenticity of host '172.31.91.118 (172.31.91.118)' can't be established.
ECDSA key fingerprint is SHA256:ozKT7G1ONYAr1WmluiAsLErbfgKXgPB6egPG3dDy5d4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.31.91.118' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-1028-aws x86_64)
```

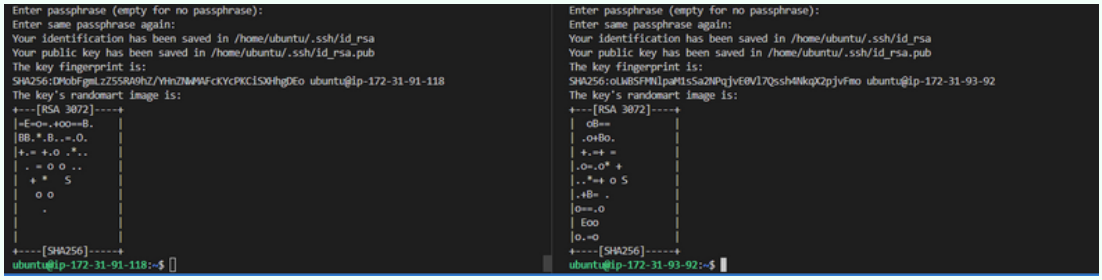
```
jenkins@32b8a66b46ff:/$ ssh ubuntu@172.31.93.92
The authenticity of host '172.31.93.92 (172.31.93.92)' can't be established.
ECDSA key fingerprint is SHA256:JiHJIzb4OoBpG0BhL/IW3SZ/q2TmPrE+24Fhuj6RYBA.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.31.93.92' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-1028-aws x86_64)
```

Launch the agents



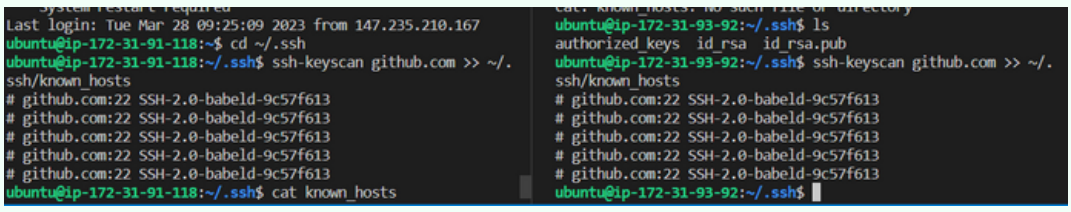


Adding prod-1-todos and prod-2-todos ssh pub keys to github

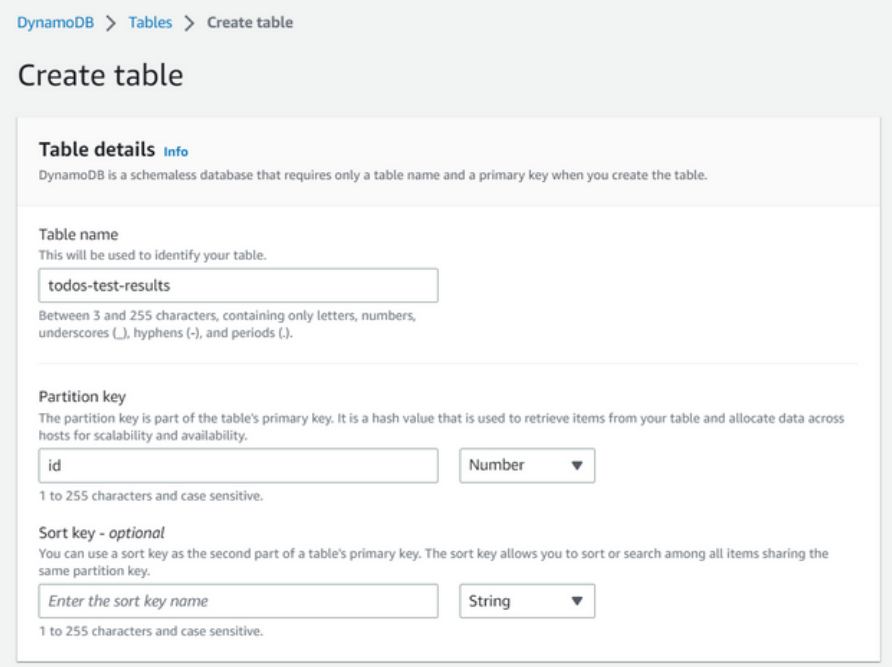


Adding GitHub to the agents known_hosts file:

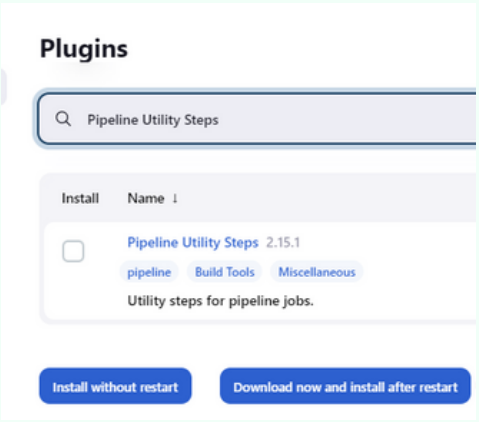
ssh-keyscan github.com >> ~/.ssh/known_hosts



- *Upload the data from the S3 bucket csv file with the tests status into the DynamoDB service. And validate that you can see the collection with the whole results..*



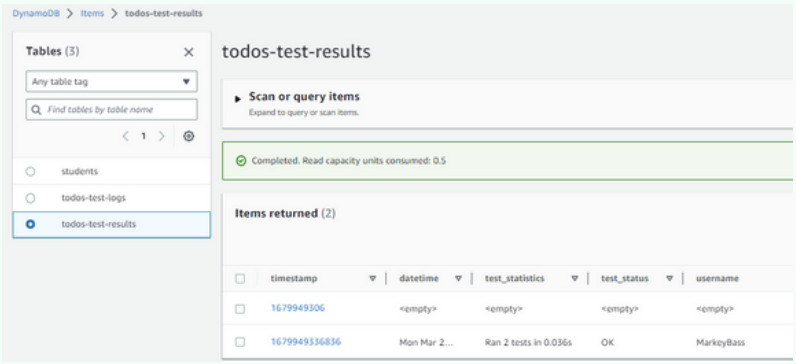
Installing "Pipeline Utility Steps" plugin to allow parsing Map from JSON (def props = readJSON text: jsonString)

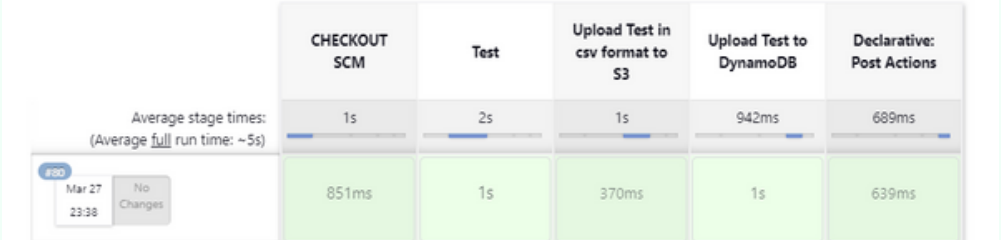


Installing AWS CLI inside my agent (agent-1 EC2)

```
sudo apt-get update
sudo apt-get install awscli
aws --version
```

```
stage('Upload Test to DynamoDB') {
        steps {
          script {
            if (fileExists('test-results.csv')) {
              def testResultMap = readJSON file: './test-results.json'


              def timestamp = testResultMap['timestamp']
              def datetime = testResultMap['datetime']
              def username = testResultMap['username']
              def test_statistics = testResultMap['test_statistics']
              def test_status = testResultMap['test_status']

              withAWS(credentials: 'awscredentials', region: 'us-east-1') {
                sh """
                  aws dynamodb put-item \
                  --table-name todos-test-results \
                  --item '{
                    \"timestamp\": {\"S\": \"${timestamp}\"},
                    \"datetime\": {\"S\": \"${datetime}\"},
                    \"username\": {\"S\": \"${username}\"},
                    \"test_statistics\": {\"S\": \"${test_statistics}\"},
                    \"test_status\": {\"S\": \"${test_status}\"}
                  }'
                """
              }

            } else {
              error('No test results file found')
            }
          }
        }
      }
```



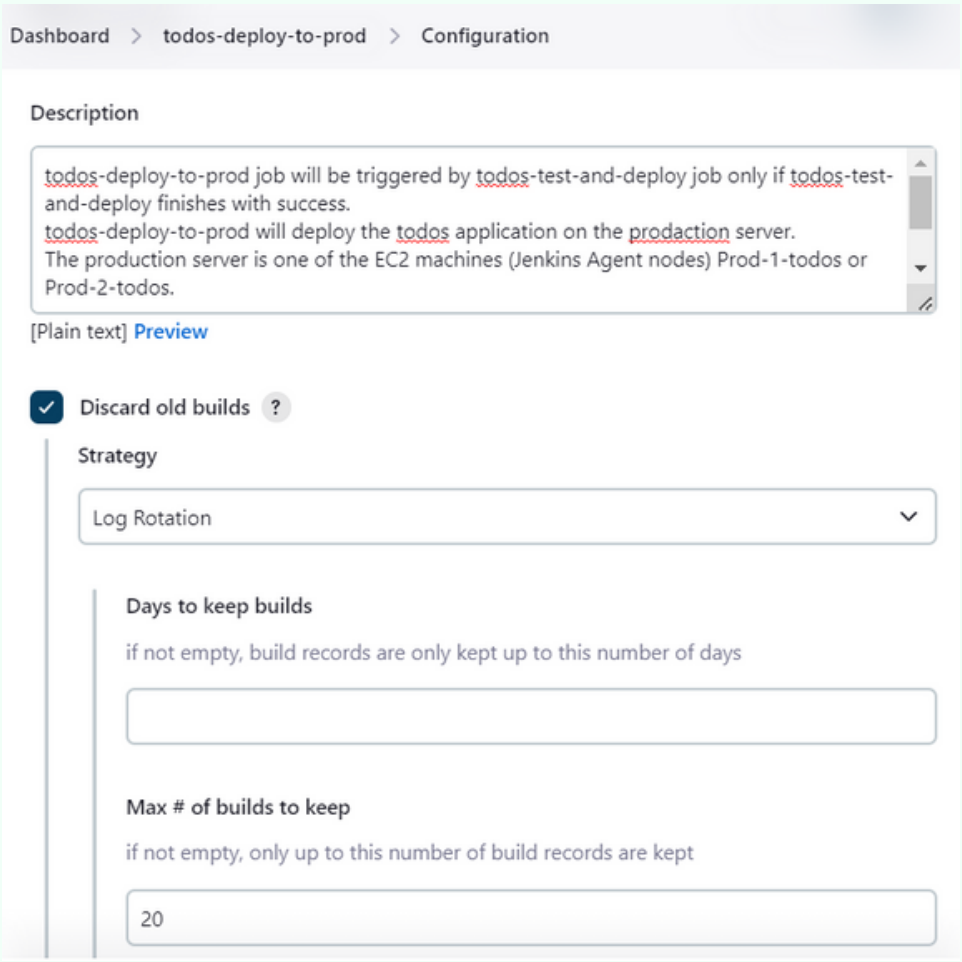


- *Create a Jenkins job that deploys the Docker image to the production server.*
- *Configure the job to trigger automatically whenever a new test build is successfully tested*

Added the trigger line to the success clause in the post clause:

```
post {
 success {
  build job: "${DEPLOY_JOB_ON_SUCCESS}", wait: false
 }
}
```

Create the deploy to prod job:



The deploy to prod job pipeline:

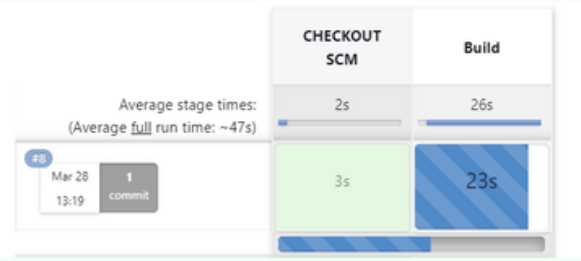
```
pipeline {
  agent {label 'Prod-1-todos'}
  environment {
    GIT_REPO_URL_SSH = 'git@github.com:MarkeyBass/todos-docker-compose.git'
  }

  stages {
    stage('CHECKOUT SCM') {
      steps {
        sshagent(credentials: ['controller-node']) {
          checkout([
            $class: 'GitSCM',
            branches: [[name: 'main']],
            userRemoteConfigs: [[
              url: "${GIT_REPO_URL_SSH}",
              credentialsId: 'controller-node'
            ]]
          ])
        }
      }
    }
    stage('Build') {
      steps {
        sh 'sudo docker compose -f docker-compose-prod.yml down'
        sh 'sudo docker compose -f docker-compose-prod.yml up -d'
      }
    }
  }
}
```

Pushing to github

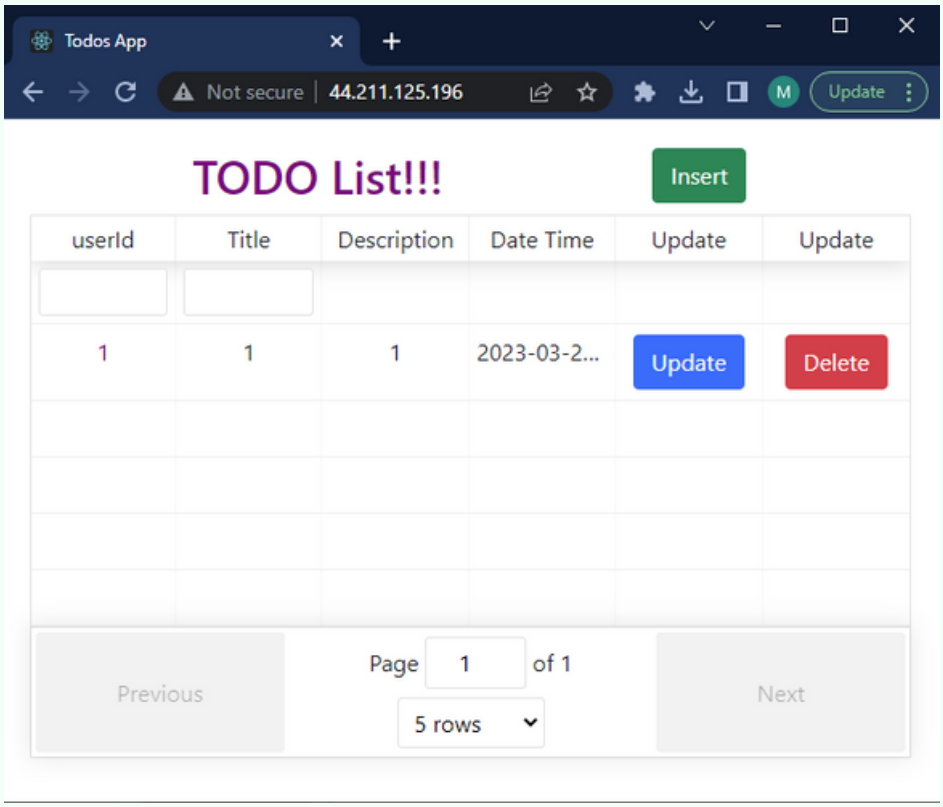
```
marke@DESKTOP-1PI2AB4 MINGW64 /c/dev/DevOps/Projects/todos (main)
$ git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 1.20 KiB | 1.20 MiB/s, done.
Total 5 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To github.com:MarkeyBass/todos-docker-compose.git
   83a3cb6..1a4f8ec  main -> main
```

todos-test-and-deploy job triggered

todos-deploy-to-prod job triggered



Production app is running on port 80



● *Create a parameter in your Jenkins job that will allow for you to choose on which of the production services you want to deploy.*
○ *The first one*
○ *The second one*
○ *Both*

todos-deploy-to-prod pipeline modified

The parameters block defines the parameter and its available options
The input step inside 'Manually add agent lable' stage ensures that the parameter is set

```
def gitCheckoutSCM(gitRepoUrl) {
  sshagent(credentials: ['controller-node']) {
    checkout([
      $class: 'GitSCM',
      branches: [[name: 'main']],
      userRemoteConfigs: [[
        url: "${gitRepoUrl}",
        credentialsId: 'controller-node'
      ]]
    ])
  }
}

def deployToProd() {
  sh 'sudo docker compose -f docker-compose-prod.yml down'
  sh 'sudo docker compose -f docker-compose-prod.yml up -d'
}

pipeline {
  agent none

  environment {
    GIT_REPO_URL_SSH = 'git@github.com:MarkeyBass/todos-docker-compose.git'
  }

  parameters {
    choice(name: 'KEY', choices: ['ONE', 'TWO', 'BOTH'], description: 'Select on which agent to run the deploy')
  }

  stages {
    stage('Input Key') {
      when {
        expression { params.KEY == null }
      }
      steps {
        input message: 'Please select KEY value', parameters: [choice(name: 'KEY', choices: ['ONE', 'TWO', 'BOTH'], description: 'Select on which agent to run the deploy')]
      }
    }
    stage('Run on Prod-1-todos') {
      when {
        expression { params.KEY == 'ONE' || params.KEY == 'BOTH' }
      }
      agent {
        label 'Prod-1-todos'
      }
      steps {
        gitCheckoutSCM(GIT_REPO_URL_SSH)
        deployToProd()
      }
    }
    stage('Run on Prod-2-todos') {
      when {
        expression { params.KEY == 'TWO' || params.KEY == 'BOTH' }
      }
      agent {
        label 'Prod-2-todos'
      }
      steps {
        gitCheckoutSCM(GIT_REPO_URL_SSH)
        deployToProd()
      }
    }
  }
}
```

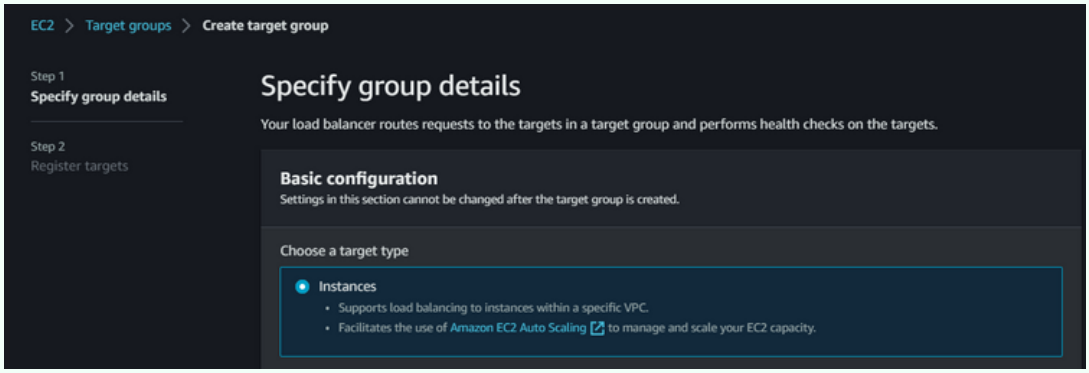
todos-test-and-deploy modified

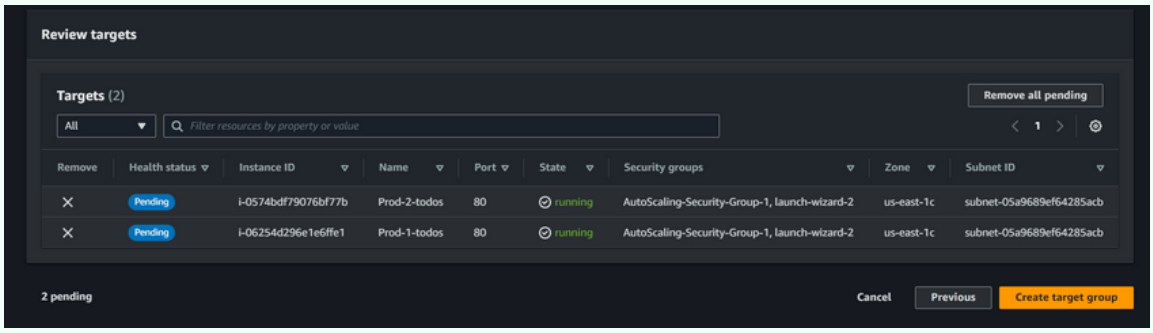
added parameter when triggering the next job

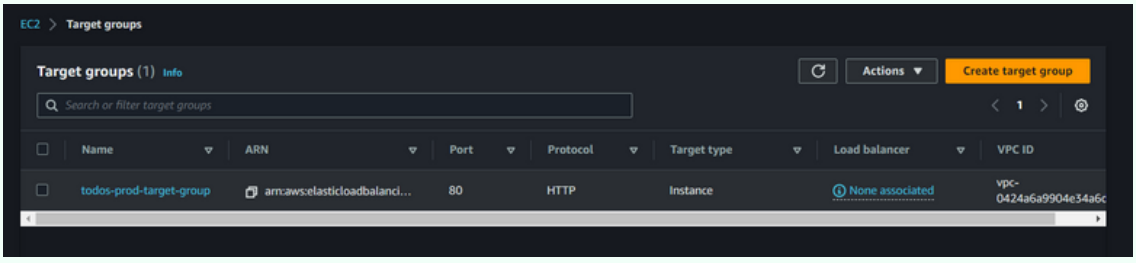
```
build job: "${DEPLOY_JOB_ON_SUCCESS}", wait: false, parameters: [
  [$class: 'StringParameterValue', name: 'KEY', value: 'BOTH']
]
```

● *Create a load balancer between your both production servers. Separate equally the traffic between both of these instances.*

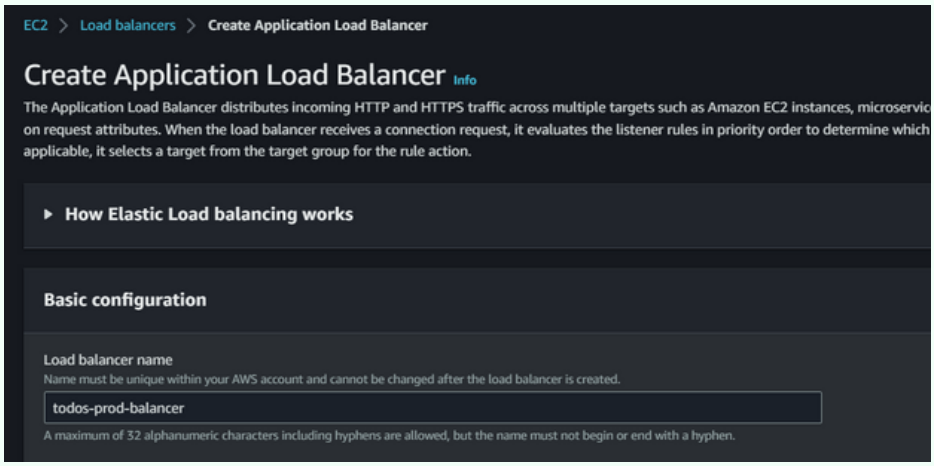
Creating a target group

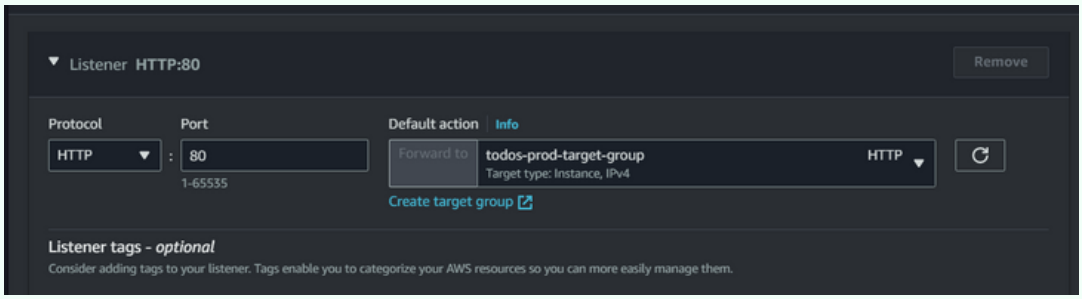




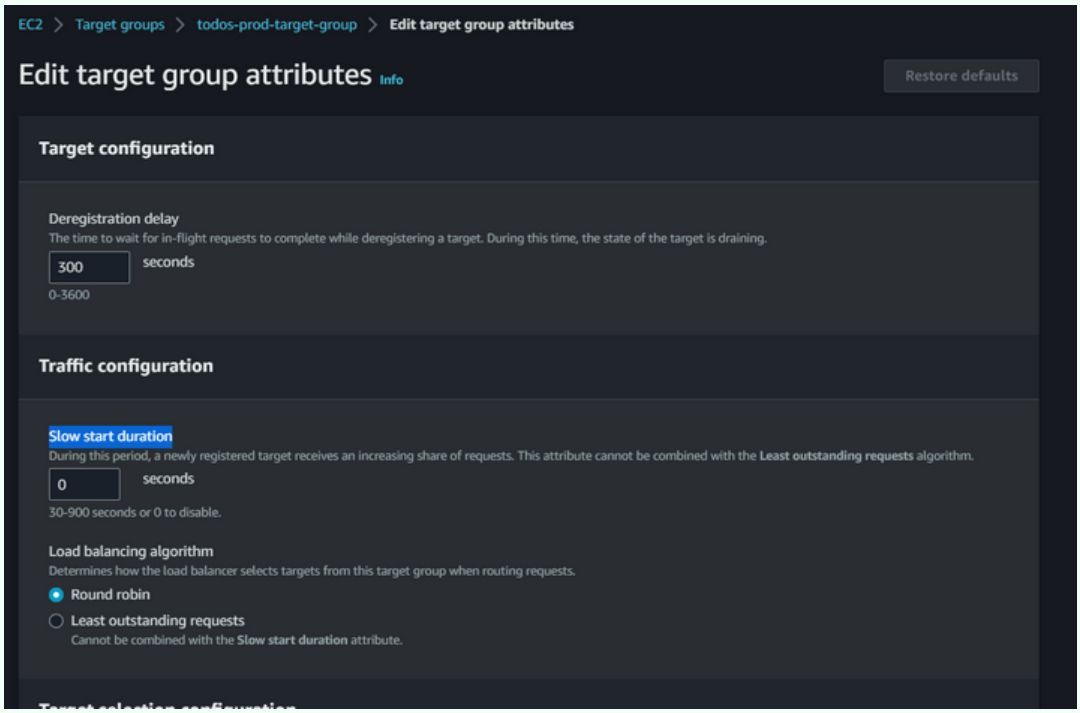


Creating a Load Balancer



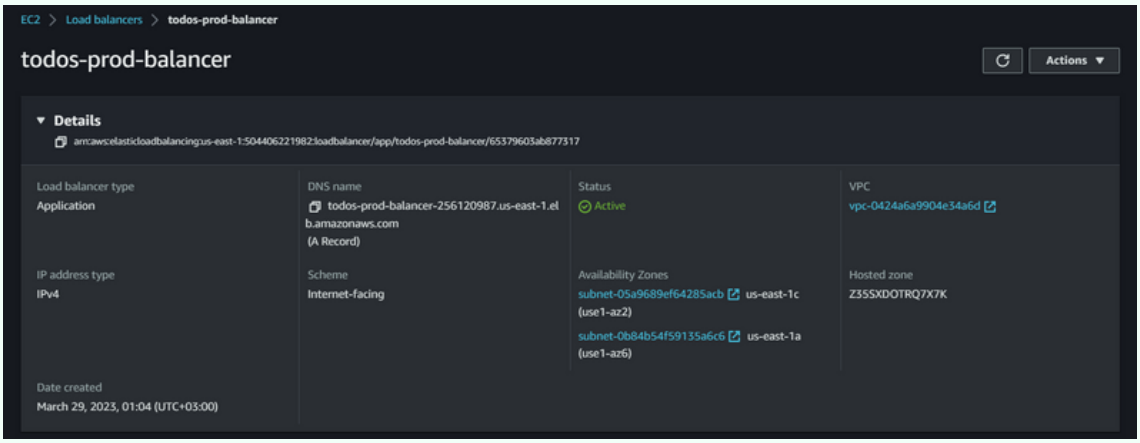


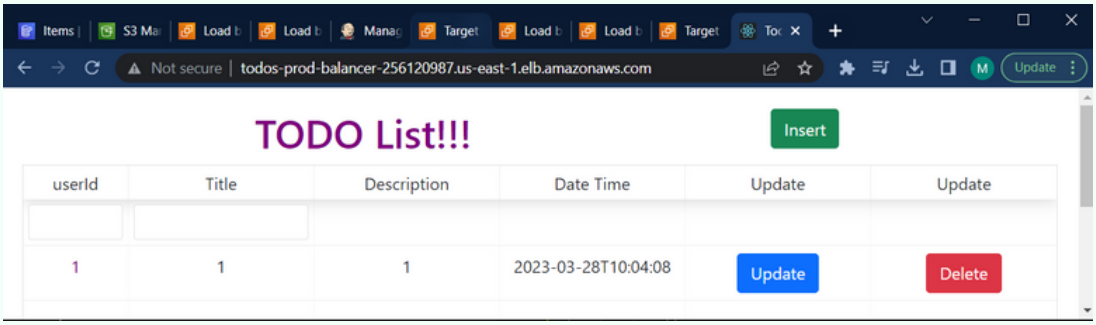
Round Robon algorithm is configured - Separate equally the traffic between both of these instances.

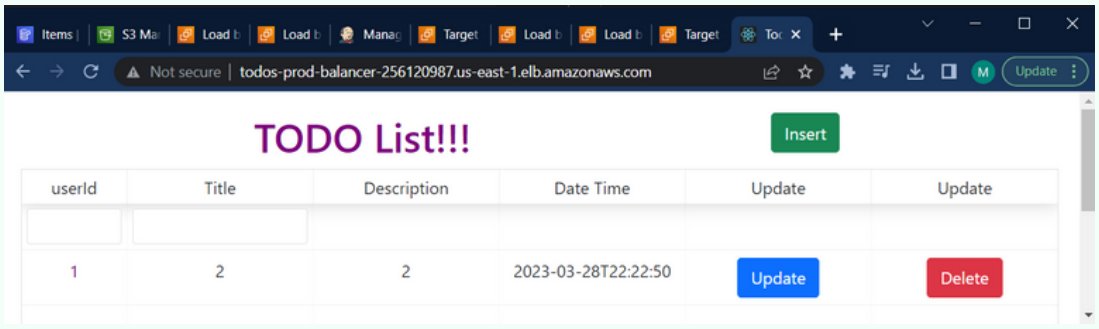


Entering the application from the load balancer DNS name

todos-prod-balancer-256120987.us-east-1.elb.amazonaws.com







● *Create a cloud watch service that will inspect your billing, the CPU of your Jenkins EC2 instance and the CPU of your production service.*