DevOps Lab-VIII

500097578
500097695
BTech CSE
B8-CCVT

Date of submission: - 26/4/24
By: -
Manav Khandurie [500097578]
Satvik Tripathi [500097695]

SCENARIO 1

"To-Do List" application
This application should allow users to create, update, delete, and view tasks in a to-do list. Below are the basic features of the To-Do List application:
1. User can add a new task with a title and description.

2. User can mark a task as completed.

3. User can update the details of a task.

4. User can delete a task.

5. User can view the list of tasks.

To implement this project, you'll need to create Java classes for tasks and the main application logic, along with unit tests to ensure the functionality works as expected. You can use a build tool like Maven or Gradle to manage dependencies and build the project.
In your CI/CD pipeline, you can include the following stages:
1. Code Checkout: Checkout the source code from the version control system. [Compulsory]

2. Build: Compile the Java code and package it into a JAR file. [Compulsory]

3. Test: Run unit tests to ensure code quality. [Compulsory]

4. Static Code Analysis: Analyze the code for code smells, bugs, and vulnerabilities. [Optional]

5. Artifact Generation: Generate the deployable artifact (JAR/WAR file). [Compulsory]

6. Deployment to Staging: Deploy the artifact to a staging environment for testing. [Optional]

7. Functional Testing: Run automated tests to verify the functionality of the application.

8. User Acceptance Testing (UAT): Deploy the application to a UAT environment for user acceptance testing. [Optional]

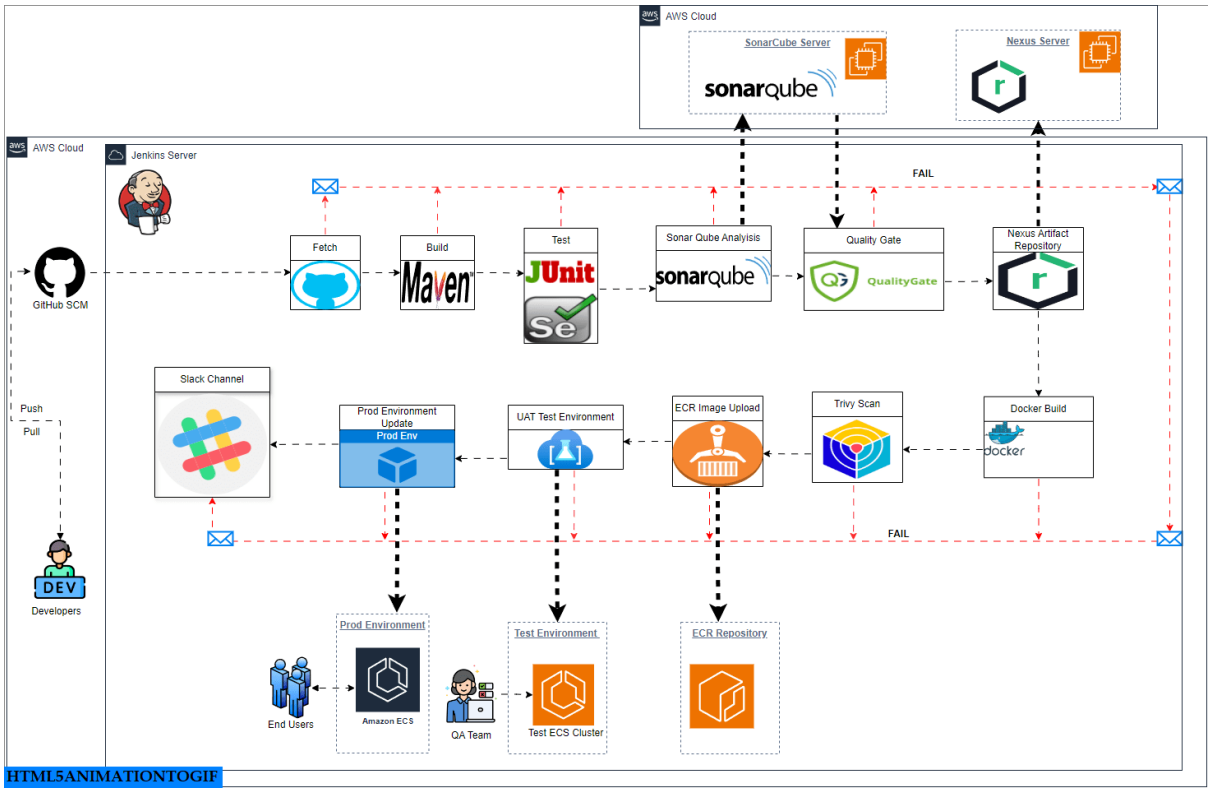
9. Deployment to Production: Deploy the artifact to the production environment. [Compulsory- Use Tomcat]

10. Post-Deployment Verification: Perform smoke tests and monitor the application in the production environment. [Optional]

11. Monitoring and Feedback: Continuously monitor the application for performance, errors, and user feedback. [Optional]

EXPERIMENT-8 [CICD Project]

Creating a end-to-end robust CICD pipeline for the application web app - todo app which is built with Java



PipeLine Overview

Tasks

"To-Do List" application
This application should allow users to create, update, delete, and view tasks in a to-do list. Below are the basic features of the To-Do List application:

1. User can add a new task with a title and description.
2. User can mark a task as completed.
3. User can update the details of a task.
4. User can delete a task.
5. User can view the list of tasks.

To implement this project, you'll need to create Java classes for tasks and the main application logic, along with unit tests to ensure the functionality works as expected. You can use a build tool like Maven or Gradle to manage dependencies and build the project.
In your CI/CD pipeline, you can include the following stages:

1. Code Checkout: Checkout the source code from the version control system. [Compulsory]

2. Build: Compile the Java code and package it into a JAR file. [Compulsory]

3. Test: Run unit tests to ensure code quality. [Compulsory]

4. Static Code Analysis: Analyze the code for code smells, bugs, and vulnerabilities. [Optional]

5. Artifact Generation: Generate the deployable artifact (JAR/WAR file). [Compulsory]

6. Deployment to Staging: Deploy the artifact to a staging environment for testing. [Optional]

7. Functional Testing: Run automated tests to verify the functionality of the application.

8. User Acceptance Testing (UAT): Deploy the application to a UAT environment for user acceptance testing. [Optional]

9. Deployment to Production: Deploy the artifact to the production environment. [Compulsory- Use Tomcat]

10. Post-Deployment Verification: Perform smoke tests and monitor the application in the production environment. [Optional]

11. Monitoring and Feedback: Continuously monitor the application for performance, errors, and user feedback. [Optional]

Team Members:

- Manav Khandurie [500097578] [B-8 CCVT NH]
- Satvik Tripathi [500097695] [B-8 CCVT NH]

Source Code

https://github.com/Manav-Khandurie/To-do-app-Maven.git

Tech Stack

- **Source Control Management (SCM):** Version control is handled by GitHub https://git-scm.com/. When a developer pushes code changes to a GitHub repository, the pipeline is triggered.
- **CI/CD Server:** Jenkins acts as the CI/CD server and orchestrates the entire pipeline. It fetches code from the Git repository, initiates builds, runs tests, and deploys the application.
- **Build Tools:** Maven is used to build the application. It manages dependencies, compiles code, and packages the application into a distributable artifact.
- **Testing Tools:** JUnit is used for unit testing the application code. Unit tests ensure individual units of code function as intended. Seliumn for UI Testing.
- **Static Code Analysis**: SonarQube performs static code analysis to identify bugs, code smells, and potential security vulnerabilities.
- **Artifact Repository**: Nexus Repository stores the build artifacts generated by the pipeline. These artifacts can be anything from compiled code to deployment packages.
- **Container Registry**: Amazon Elastic Container Registry (ECR) stores Docker images used to deploy the application to different environments.
- **Security Scanning**: Trivy is used to scan Docker images for vulnerabilities.
- **Cloud Provider**: Amazon Web Services (AWS) provides the cloud infrastructure for running the pipeline and deploying the application.
- **Environments**: The pipeline deploys the application to three different environments:
    - Development (DEV)
    - User Acceptance Testing (UAT)
    - Production (Prod)

Here's a simplified overview of how the pipeline works:

1. Developers push code changes to the GitHub repository.
2. The push triggers Jenkins, which fetches the code from GitHub.
3. Jenkins uses Maven to build the application.

4. JUnit tests are executed to verify the functionality of the code.
5. SonarQube performs static code analysis. If the code quality does not meet the quality gate set by the development team, the pipeline fails.
6. If all the stages pass, the build artifact is uploaded to Nexus Repository.
7. A Docker image is built from the artifact and uploaded to ECR.
8. The image is deployed to the UAT environment where QA performs testing.
9. Once UAT is successful, the image is deployed to the production environment.

PipeLine

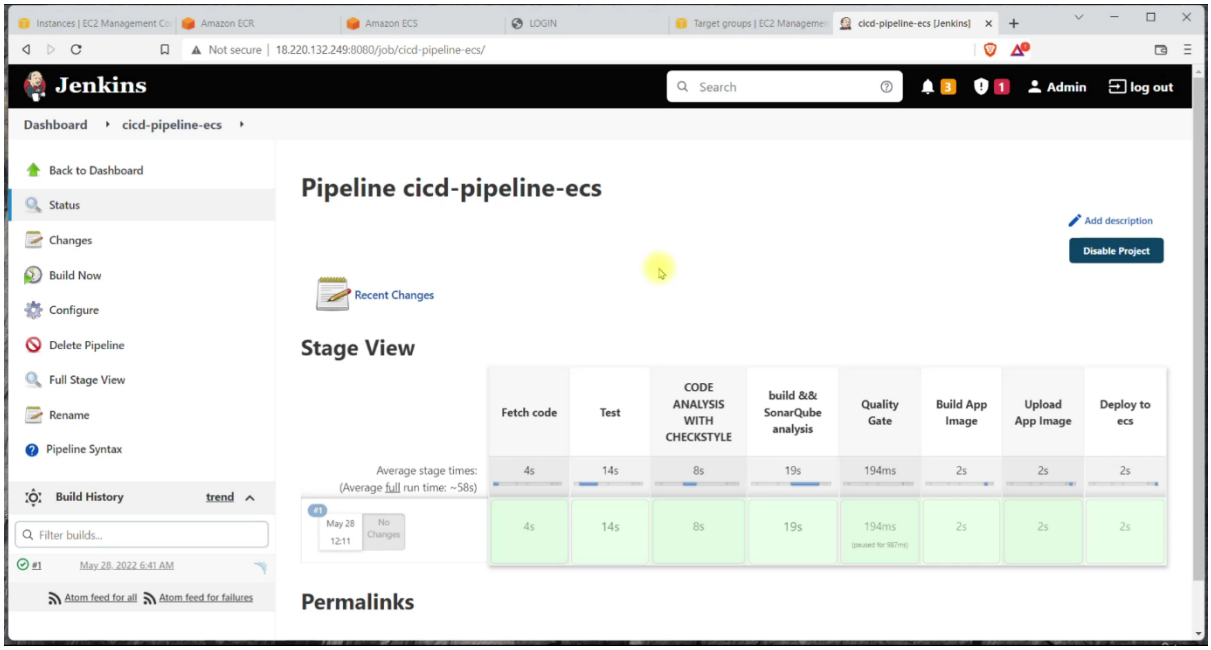
```
def COLOR_MAP = [
   'SUCCESS': 'good',
   'FAILURE': 'danger',
   'REPORT': '#FFFF00'
]

pipeline {
   agent any
   tools {
      maven "MAVEN3"
      jdk "OracleJDK8"
   }

   environment {
      registryCredential = 'ecr:us-east-1:awsiamuser'
      appRegistry = "247477386084.dkr.ecr.us-east-
1.amazonaws.com/devopsproj-todo"
      vprofileRegistry = "https://247477386084.dkr.ecr.us-east-
1.amazonaws.com/"
      clusterTest = "test-cluster"
      serviceTest = "test-service"
      clusterProd = "prod-cluster"
      serviceProd = "prod-service"
      slack_channel = "my-proj-cicd"
```

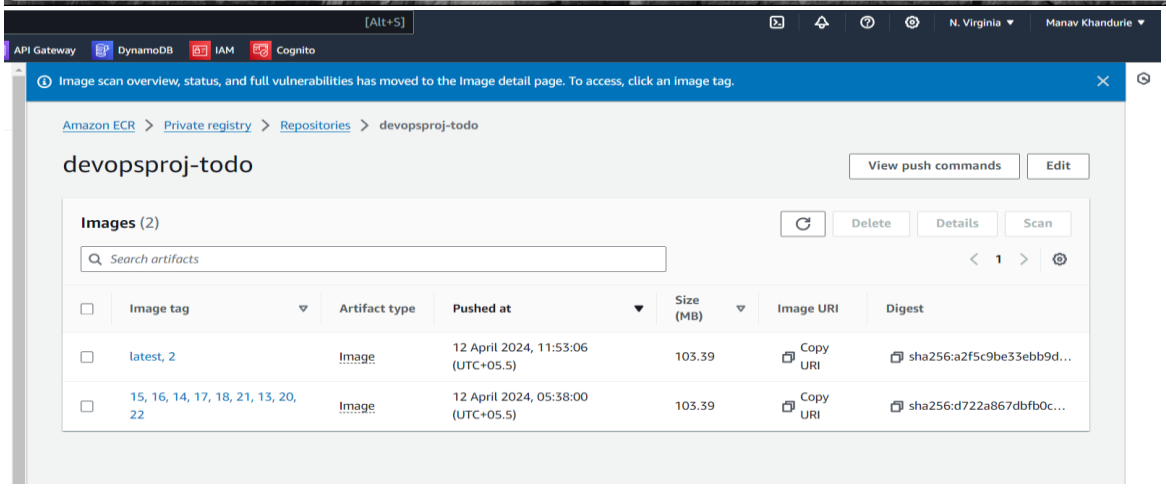
```
    trivyReportPath = "${env.WORKSPACE}/trivy_report.json"
    sonarToken = credentials('sonar-token')
    nexusCredential = credentials('nexus-credential')
  }

  stages {
    stage('Fetch code'){
      steps {
        git branch: 'master', url: 'https://github.com/Manav-Khandurie/To-do-app-Maven.git'
      }
    }

    stage('Build App Image') {
      steps {
        script {
          dockerImage = docker.build( appRegistry + ":$BUILD_NUMBER")
        }
      }
    }

    stage('Upload App Image to AWS ECR') {
      steps {
        script {
          docker.withRegistry( vprofileRegistry, registryCredential ) {
            dockerImage.push("$BUILD_NUMBER")
            dockerImage.push('latest')
          }
        }
      }
    }

    stage('Unit Test and Integration Test'){
      steps {
        sh 'mvn test'
      }
```

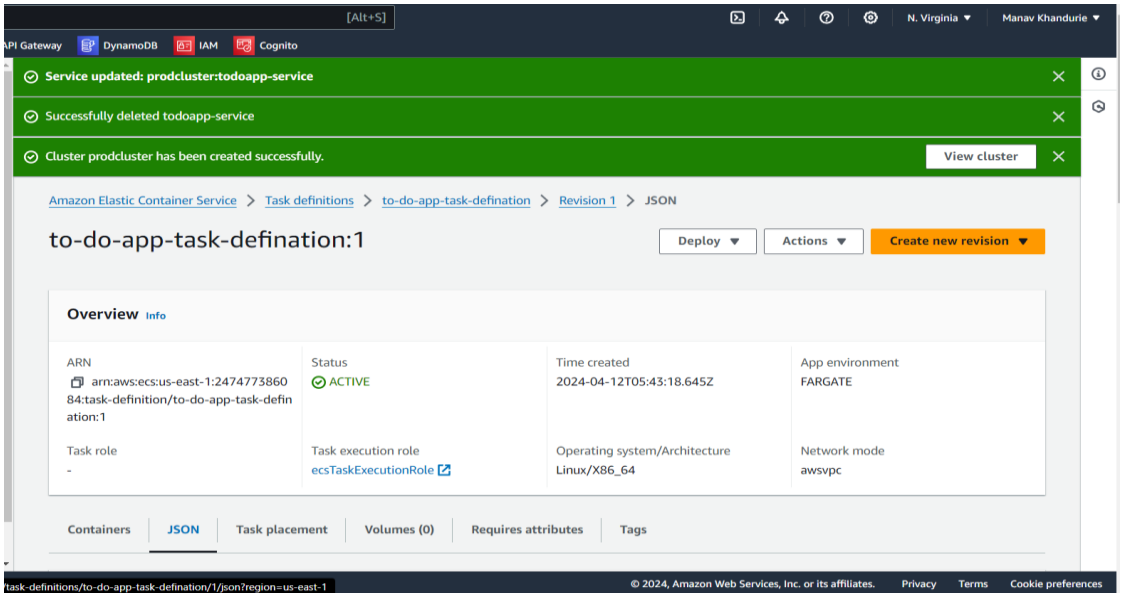
```
        }

        stage('Generate Surefire Reports') {
            steps {
                script {
                    sh 'mvn surefire-report:report'
                }
                junit 'target/surefire-reports/**/*.xml'
            }
        }

        stage('Upload to Nexus') {
            steps {
                withCredentials([usernamePassword(credentialsId: 'nexus-
credential', usernameVariable: 'NEXUS_USERNAME', passwordVariable:
'NEXUS_PASSWORD')]) {
                    sh "mvn deploy:deploy-file -Durl=http://18.30.132.259/
repository/maven-releases/ -DrepositoryId=nexus -Dfile=path/to/artifact.jar -
DgroupId=com.example -DartifactId=my-artifact -Dversion=1.0.0 -
Dpackaging=jar -DgeneratePom=true -DgeneratePom.description='My
Artifact Description' -DgeneratePom.url='http://myartifacturl.com'"
                }
            }
        }

        stage('Static Code Analysis with SonarQube') {
            steps {
                withSonarQubeEnv('SonarQube') {
                    sh "mvn sonar:sonar -Dsonar.host.url=http://18.220.135.129-
Dsonar.login=${sonarToken}"
                }
            }
        }

        stage('Trivy Vulnerability Scan') {
            steps {
                script {
```

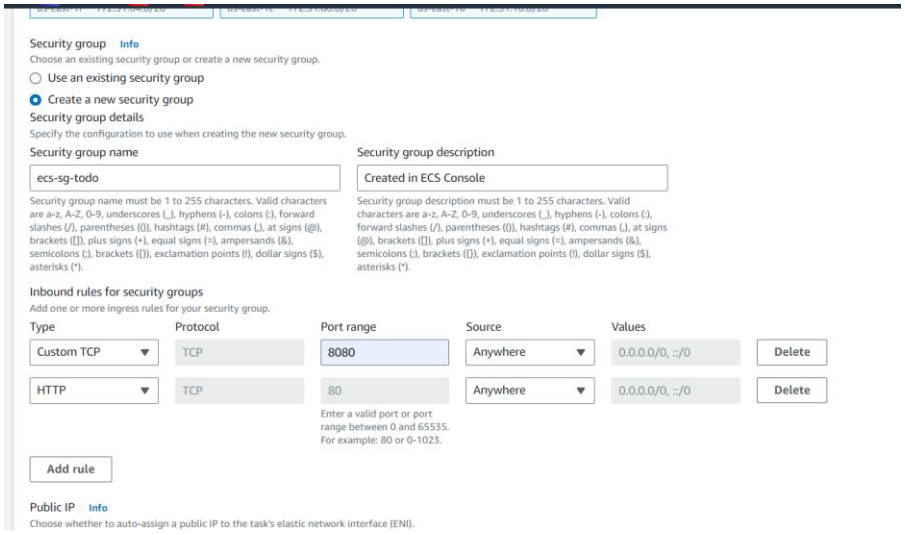
```
                echo "${env.WORKSPACE}"
                sh "trivy image ${appRegistry}:${BUILD_NUMBER} --severity
CRITICAL --no-progress --format json > ${trivyReportPath}"
            }
        }
    }

    stage('Upload to Slack'){
        steps {
            script {
                slackUploadFile channel: slack_channel, filePath:
'${env.WORKSPACE}/trivy_report.json', initialComment: 'Trivy report'
            }
        }
    }

    stage('Deploy to Test Environment') {
        steps {
            withAWS(credentials: 'awsiamuser', region: 'us-east-1') {
                sh "aws ecs update-service --cluster ${clusterTest} --service
${serviceTest} --force-new-deployment"
            }
        }
    }

    stage('Confirm Deployment to Prod') {
        steps {
            timeout(time: 5, unit: 'MINUTES') {
                input message: 'Proceed with deployment to production?',
parameters: [choice(name: 'DeployToProd', choices: 'Yes\nNo', description:
'Do you want to deploy to production?')]
            }
        }
    }

    stage('Deploy to Prod Environment') {
        when {
```

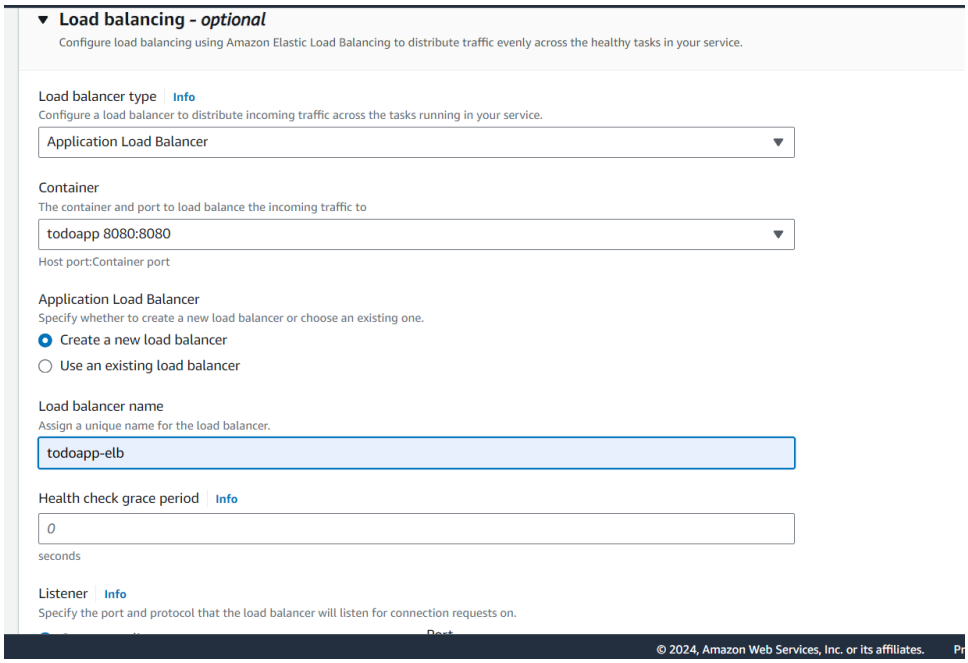
```
            expression {
                return params.DeployToProd == 'Yes'
            }
        }
        steps {
            withAWS(credentials: 'awsiamuser', region: 'us-east-1') {
                sh "aws ecs update-service --cluster ${clusterProd} --service
${serviceProd} --force-new-deployment"
            }
        }
    }
  }

  post {
    always {
        echo 'Slack Notifications.'
        slackSend channel: slack_channel,
            color: COLOR_MAP[currentBuild.currentResult],
            message: "*${currentBuild.currentResult}:* Job ${env.JOB_NAME}
build ${env.BUILD_NUMBER} \n More info at: ${env.BUILD_URL}"
    }
  }
}
```

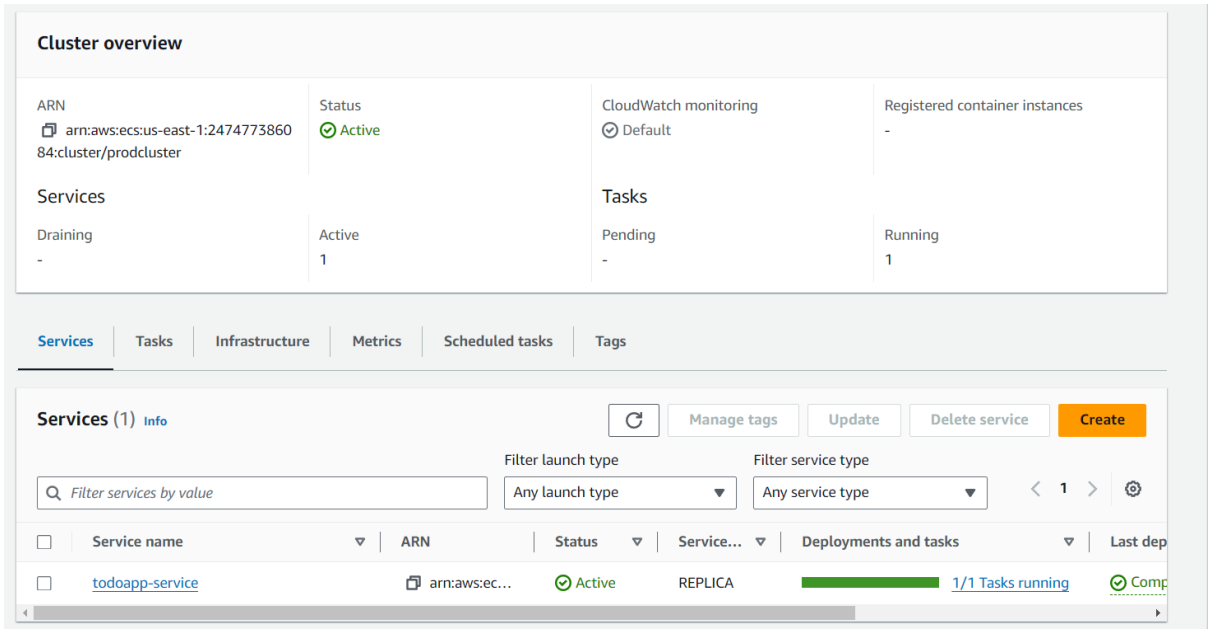

Jenkins
Search
Admin
log out
Dashboard
cicd-pipeline-ecs
Back to Dashboard
Status
Changes
Build Now
Configure
Delete Pipeline
Full Stage View
Rename
Pipeline Syntax
Build History
trend
Pipeline cicd-pipeline-ecs
Add description
Disable Project
Recent Changes
Stage View
Fetch code
Test
CODE ANALYSIS WITH CHECKSTYLE
build && SonarQube analysis
Quality Gate
Build App Image
Upload App Image
Deploy to ecs
Average stage times:
(Average full run time: ~58s)
4s
14s
8s
19s
194ms
2s
2s
2s
Permalinks

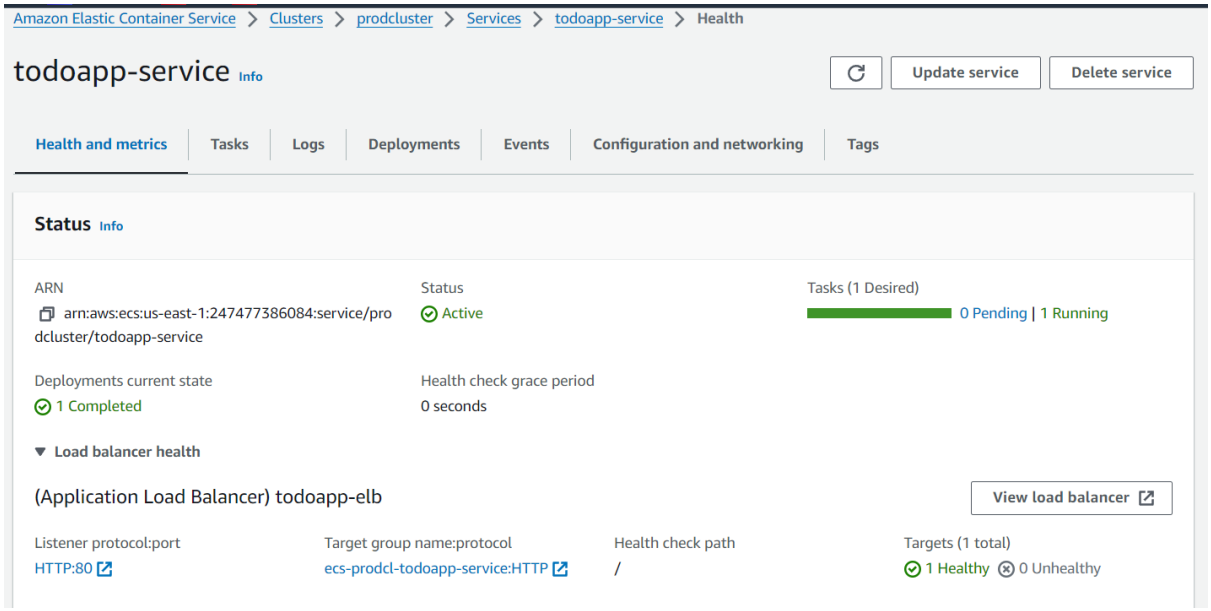

API Gateway
DynamoDB
IAM
Cognito
N. Virginia
Manav Khandurie
Image scan overview, status, and full vulnerabilities has moved to the Image detail page. To access, click an image tag.
Amazon ECR
Private registry
Repositories
devopsproj-todo
devopsproj-todo
View push commands
Edit
Images (2)
Delete
Details
Scan
Search artifacts
Image tag
Artifact type
Pushed at
Size (MB)
Image URI
Digest
latest, 2
Image
12 April 2024, 11:53:06 (UTC+05.5)
103.39
Copy URI
sha256:a2f5c9be33ebb9d...
15, 16, 14, 17, 18, 21, 13, 20, 22
Image
12 April 2024, 05:38:00 (UTC+05.5)
103.39
Copy URI
sha256:d722a867dbfb0c...

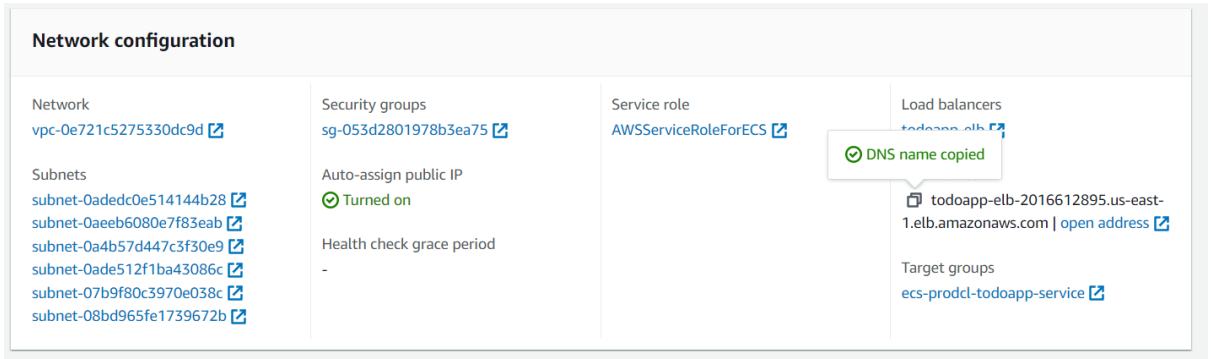

API Gateway
DynamoDB
IAM
Cognito
N. Virginia
Manav Khandurie
Service updated: prodcluster:todoapp-service
Successfully deleted todoapp-service
Cluster prodcluster has been created successfully.
View cluster
Amazon Elastic Container Service
Task definitions
to-do-app-task-defination
Revision 1
JSON
to-do-app-task-defination:1
Deploy
Actions
Create new revision
Overview Info
ARN
arn:aws:ecs:us-east-1:247477386084:task-definition/to-do-app-task-defination:1
Status
ACTIVE
Time created
2024-04-12T05:43:18.645Z
App environment
FARGATE
Task role
-
Task execution role
ecsTaskExecutionRole
Operating system/Architecture
Linux/X86_64
Network mode
awsvpc
Containers
JSON
Task placement
Volumes (0)
Requires attributes
Tags
/task-definitions/to-do-app-task-defination/1/json?region=us-east-1
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

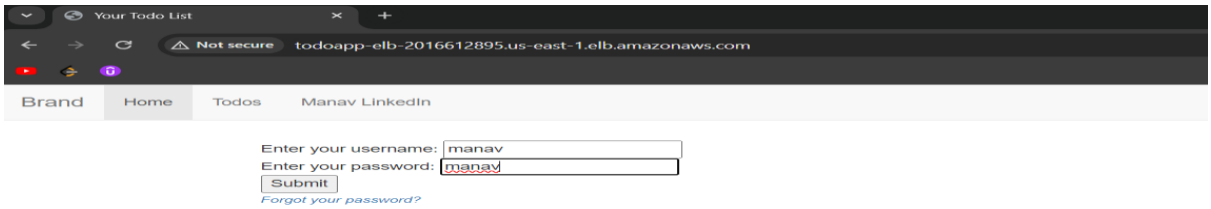

Security group Info
Choose an existing security group or create a new security group.
Use an existing security group
Create a new security group
Security group details
Specify the configuration to use when creating the new security group.
Security group name
ecs-sg-todo
Security group name must be 1 to 255 characters. Valid characters are a-z, A-Z, 0-9, underscores (_), hyphens (-), colons (:), forward slashes (/), parentheses ((), hashtags (#), commas (,), at signs (@), brackets ([]), plus signs (+), equal signs (=), ampersands (&), semicolons (;), brackets ({}), exclamation points (!), dollar signs ($), asterisks (*).
Security group description
Created in ECS Console
Security group description must be 1 to 255 characters. Valid characters are a-z, A-Z, 0-9, underscores (_), hyphens (-), colons (:), forward slashes (/), parentheses ((), hashtags (#), commas (,), at signs (@), brackets ([]), plus signs (+), equal signs (=), ampersands (&), semicolons (;), brackets ({}), exclamation points (!), dollar signs ($), asterisks (*).
Inbound rules for security groups
Add one or more ingress rules for your security group.
Type
Protocol
Port range
Source
Values
Custom TCP
TCP
8080
Anywhere
0.0.0.0/0, ::/0
Delete
HTTP
TCP
80
Anywhere
0.0.0.0/0, ::/0
Delete
Enter a valid port or port range between 0 and 65535. For example: 80 or 0-1023.
Add rule
Public IP Info
Choose whether to auto-assign a public IP to the task's elastic network interface (ENI).

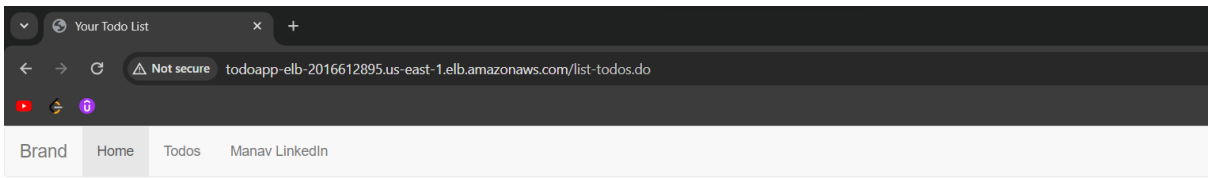

Load balancing - optional
Configure load balancing using Amazon Elastic Load Balancing to distribute traffic evenly across the healthy tasks in your service.
Load balancer type Info
Configure a load balancer to distribute incoming traffic across the tasks running in your service.
Application Load Balancer
Container
The container and port to load balance the incoming traffic to
todoapp 8080:8080
Host port:Container port
Application Load Balancer
Specify whether to create a new load balancer or choose an existing one.
Create a new load balancer
Use an existing load balancer
Load balancer name
Assign a unique name for the load balancer.
todoapp-elb
Health check grace period Info
0
seconds
Listener Info
Specify the port and protocol that the load balancer will listen for connection requests on.
© 2024, Amazon Web Services, Inc. or its affiliates.


Cluster overview
ARN
arn:aws:ecs:us-east-1:247477386084:cluster/prodcluster
Status
Active
CloudWatch monitoring
Default
Registered container instances
-
Services
Draining
-
Active
1
Tasks
Pending
-
Running
1
Services
Tasks
Infrastructure
Metrics
Scheduled tasks
Tags
Services (1) Info
Manage tags
Update
Delete service
Create
Filter launch type
Any launch type
Filter service type
Any service type
Filter services by value
1
Service name
ARN
Status
Service...
Deployments and tasks
Last dep
todoapp-service
arn:aws:ec...
Active
REPLICA
1/1 Tasks running
Comp







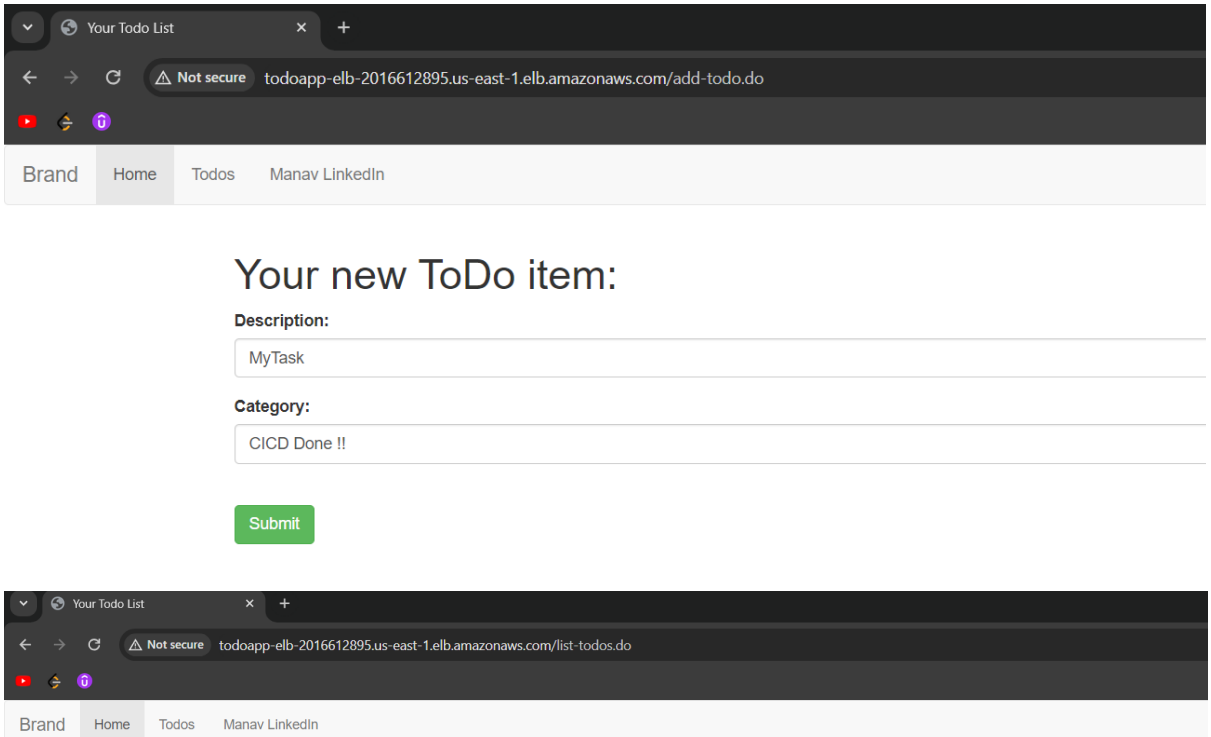


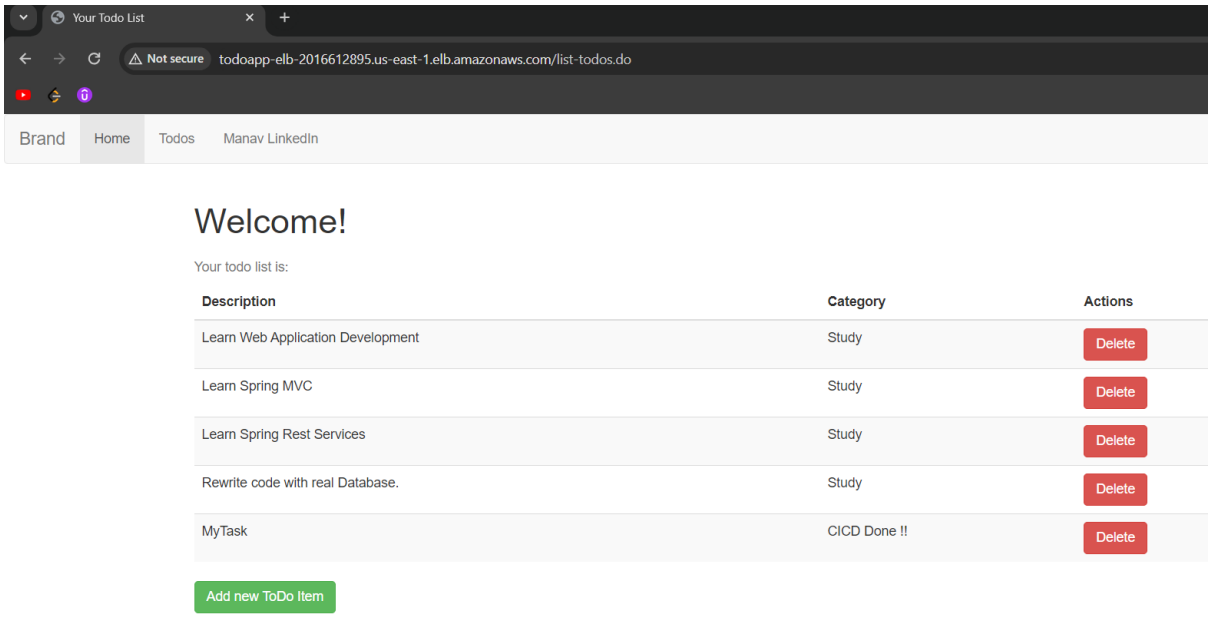
Welcome!

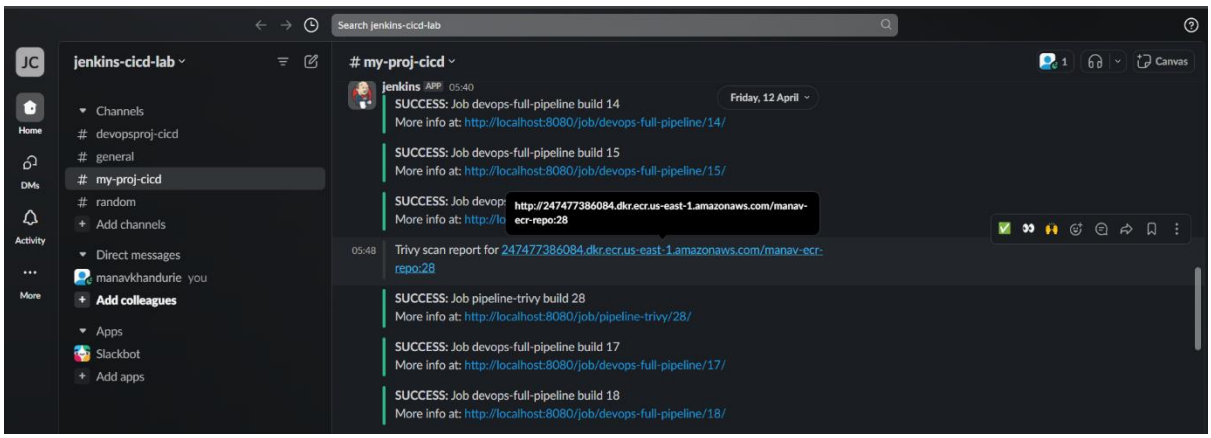
Your todo list is:

Description	Category	Actions
Learn Web Application Development	Study	Delete
Learn Spring MVC	Study	Delete
Learn Spring Rest Services	Study	Delete
Rewrite code with real Database.	Study	Delete

Add new ToDo Item







As Seen Above the entire deployment to the prod environment is done.