# Project Overview

* 1. Project Title: End To End CI/CD Pipeline Project
  2. Summary:

A screenshot of a computer

Description automatically generated

This project outlines a comprehensive development and deployment pipeline, leveraging continuous integration and continuous deployment (CI/CD) principles. The process begins with rigorous requirements gathering from the client to ensure a clear understanding of project goals. Subsequently, developers select the most suitable source code and programming language to implement the project's functionalities.

A centralized repository, such as GitHub, serves as the primary hub for code management. Developers commit their code changes to this repository, enabling collaboration and version control. A robust CI/CD pipeline is established to automate the build, test, and deployment processes.

The Continuous Integration pipeline is initiated by a build server, which automatically pulls the latest code from the GitHub repository. Maven, a commonly used build tool for Java, is employed to compile the project code and generate the necessary artifacts. To ensure code quality, SonarQube, a static code analysis tool, is integrated into the pipeline. This tool meticulously examines the code for potential vulnerabilities, bugs, and code quality issues. In the event of any detected problems, developers are promptly notified via email.

Once the code passes the Continuous Integration stage, it transitions to the Continuous Deployment pipeline. A Dockerfile is created to define the Docker image, which encapsulates the application and its dependencies. This Docker image is then pushed to AWS Elastic Container Registry (ECR) for efficient storage and distribution.

A staging environment is set up to simulate the production environment. The Docker image is pulled from ECR and deployed to this staging environment. Rigorous testing is conducted in this environment to validate the application's functionality and performance. Upon successful testing, the Docker image is promoted to the production environment, ensuring a seamless and reliable deployment process.

* 1. Purpose and Goals:

The purpose of this project is to streamline the development and deployment process by implementing a robust CI/CD pipeline. This pipeline aims to enhance efficiency, improve code quality, and accelerate time-to-market for the project.

The specific goals of the project are:

* To establish a centralized repository for efficient code management and collaboration.
* To build, test, and deployment processes through CI/CD.
* To ensure code quality by integrating static code analysis tools like SonarQube.
* To streamline deployment by leveraging containerization technology (Docker) and cloud infrastructure (AWS ECR).
* To provide a reliable and efficient deployment process by utilizing a staging environment for thorough testing before production deployment.
  1. Prerequisites:
* AWS Services:
  + AWS EC2
  + AWS ECR
  + Security Groups
  + IAM Role
* CI Server
  + Docker
  + Java 17
  + Apache Maven
  + Git
  + Sonarscanner plugin
* Sonarqube Server
  + Java 17
  + SonarQube
* CD Server
  + Docker
  + AWS CLI
  + Java 17

# Implementation

* 1. To begin, we must establish a **build server** environment capable of running the project. We will utilize an **Amazon Elastic Compute Cloud (EC2)** instance running **Amazon Linux 2**. To ensure adequate computational resources, a **t2.medium** instance type will be employed. This instance can be configured to use either a newly generated or an existing SSH key pair. To facilitate remote access, the security group associated with the instance must permit incoming **SSH** traffic. Additionally, the server will require the installation of essential software components, including **Java**, **Docker**, **Apache Maven**, and **Git**.

A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A screenshot of a network setting

Description automatically generated  
A close up of a computer screen

Description automatically generated

* 1. To gain remote access to the server, we can leverage either MobaXterm or PuTTY. For this guide, we'll utilize **MobaXterm**. The process involves navigating to the **"Session"** tab within MobaXterm. Here, we'll be prompted to input the server's **public IP address**, the username (which, in this case, is **"ec2-user"**), and the associated **private key file**. To obtain the public IP address, refer to the EC2 console. Once acquired, paste it into the designated field in MobaXterm. Next, specify "ec2-user" as the username. To securely authenticate, navigate to the "Advanced Settings" and select the (.pem) private key file. Finally, confirm these settings by clicking "OK."

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a computer program

Description automatically generated

* 1. To ensure compatibility with the project's dependencies, particularly SonarQube, we will install **Java 17**. Begin by conducting a web search for **"download Java 17 for Amazon Linux 2."** After some experimentation, we've determined that Java 17 provides optimal performance and compatibility with the current versions of the other services. It's important to note that using Java 11 may lead to connectivity issues with SonarQube.

A screenshot of a computer

Description automatically generated

* 1. To install Java 17 on the server, we will follow the official instructions provided by Amazon. Copy the first step and paste it to the server. Switch to root user in the server before installing the java package.

A screenshot of a computer

Description automatically generated

A computer screen with white text and colorful text

Description automatically generated

A computer screen shot of a computer program

Description automatically generated

* 1. To facilitate version control and project management, we will install Git. This can be achieved by executing the following command in the terminal: **sudo yum install git -y**

****

A screen shot of a computer

Description automatically generated

****

* 1. To initiate the project setup, we will clone the designated repository from GitHub. First, retrieve the repository's URL. Then, navigate to the root directory of the server using the command **cd /root**. Finally, execute the following Git command to clone the repository: **git clone <repository\_url>**

  
A black background with white text

Description automatically generated  
A computer screen shot of a program

Description automatically generated

* 1. To initiate the build process, **Maven**, a popular build automation tool, is required. The initial step involves navigating to a search engine to locate the Maven repository. From the repository, the desired version, in this case 3.9.9, is selected, and the corresponding download link for the compressed tar file is copied. Subsequently, the terminal is utilized to access the /opt directory on the build server. This directory is a common location for installing software packages. The **wget** command is then employed to download the tar file from the previously copied link.

A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A computer screen shot of a program

Description automatically generated

* 1. Once the tar file containing the Maven distribution is downloaded, the extraction process is initiated using the command **tar -xvf <tar file name>**. This uncompresses the file and creates the necessary directories and files. To ensure that the Maven executable can be accessed from the terminal, the environment variable PATH is modified. This is accomplished by executing the command export **PATH=$PATH:/opt/<name>/bin**, which appends the directory containing the Maven binaries to the existing PATH variable. This allows the system to locate and execute Maven commands from any directory.

A screenshot of a computer

Description automatically generated  
A screen shot of a computer

Description automatically generated

* 1. To commence the build process, the project repository, which was previously cloned, is navigated to. The **mvn clean package** command is executed within the terminal to initiate the build. This command triggers a series of actions, including cleaning the project, compiling the source code, running tests, and packaging the application. Upon successful completion, a new directory named **target** is generated within the project directory. This directory houses the final artifact, which is configured according to the specifications outlined in the **pom.xml** file.

A screenshot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

* 1. To establish a **SonarQube** service, a new instance is launched on AWS EC2. This instance, similar to the build server, will employ **Ubuntu** as the operating system. Once the instance is up and running, remote access is facilitated through MobaXterm, using the default username **'ubuntu'** for authentication. This process mirrors the steps undertaken for the Amazon Linux server, with the exception of the underlying **operating system** and the **default user credentials**.

A screenshot of a computer

Description automatically generated  
A screenshot of a computer application

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A screenshot of a network settings

Description automatically generated  
  
A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated

* 1. To initiate SonarQube installation, the SonarQube distribution is downloaded from the official website. As the **'ubuntu'** user, the terminal is used to navigate to the **‘home/ubuntu’** directory. The **wget** command is then employed to download the SonarQube package from the copied link. Once the download is complete, the package is **unzipped** to extract the necessary files for the SonarQube installation.

A screenshot of a computer

Description automatically generated  
A screenshot of a computer program

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A computer screen shot of a computer program

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A black background with white text

Description automatically generated  
A screenshot of a computer

Description automatically generated

* 1. To ensure compatibility and optimal performance, the SonarQube server requires a specific Java version. In this case, **Java 17** is installed. The **sudo apt update** command is executed to refresh the package lists, followed by **sudo apt -y install openjdk-17-jdk** to install the Java Development Kit (JDK). Additionally, the **sudo apt -y install openjdk-17-jre** command is used to install the Java Runtime Environment (JRE), which is optional but recommended for comprehensive functionality. By installing both the JDK and JRE, the SonarQube server is equipped with the necessary components to execute and analyze code effectively.
  2. To integrate SonarQube with the Maven build process, the **SonarScanner plugin** is added to the **pom.xml** file of the build server. This plugin facilitates communication between Maven and the SonarQube server, enabling code analysis and quality gate enforcement. The latest version of the SonarScanner plugin is obtained from the Maven repository and incorporated into the **<plugins>** section of the **pom.xml** file. This configuration ensures that the build process includes the necessary steps to send code to SonarQube for analysis.

A screenshot of a computer

Description automatically generated  
A blue and white screen with text

Description automatically generated  
A screenshot of a computer program

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A black background with white text

Description automatically generated  
A computer screen shot of text

Description automatically generated

* 1. The integration process continues by establishing a connection between the build server and the SonarQube service. To achieve this, the SonarQube service needs to be running. Locate the **sonar.sh** script, typically found by following the instructions provided in accompanying visuals. Once identified, execute the command **sh sonar.sh start** to initiate the SonarQube service. It's crucial to verify the service status using **sh sonar.sh status** after starting. Multiple checks might be necessary as there are instances where the service might stop unexpectedly. This behavior could be attributed to launching the service with root privileges. Remember, the SonarQube service can be started by **non-root users**, eliminating the need for elevated access.

A screen shot of a computer

Description automatically generated  
A screen shot of a computer

Description automatically generated  
A screen shot of a black and white screen

Description automatically generated  
A screen shot of a number

Description automatically generated  
A black background with white numbers

Description automatically generated

* 1. Once the SonarQube service is successfully started, the next step is to access its web interface. To do so, you'll need to identify the public IP address assigned to the server hosting the SonarQube service. This IP address can typically be found in the AWS Management Console or other relevant infrastructure management tools. With the IP address in hand, construct a URL in the following format: **http://<public\_ip\_address>:9000**. Ensure that you use the **HTTP** protocol (http) instead of HTTPS. Replace **<public\_ip\_address>** with the actual IP address of your SonarQube server. For example, if the IP address is 3.83.219.83, the URL would be [**http://3.83.219.83:9000**](http://3.83.219.83:9000). Open a web browser and enter this URL into the address bar. This will take you to the SonarQube web interface, where you can manage projects, analyze code quality, and track technical debt.

A screenshot of a computer

Description automatically generated

* 1. To proceed with the SonarQube integration, you'll need to generate a token that will be used to authenticate the build server with the SonarQube server. To obtain this token, navigate to your SonarQube account by accessing the web interface. Once logged in, locate your profile icon, typically represented by your initials or a user icon. Click on this icon, and from the dropdown menu, select **"My Account."** This will open your user settings page. Within this page, locate the **"Security"** tab and click on it. Here, you'll find an option to generate a token. Click on the **"Generate"** button to create a new token. This token will be used to authorize the build server to interact with the SonarQube server during the build process.

A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated

* 1. Having secured the SonarQube token, return to the build server and navigate to the directory where your project resides. This is typically the location where you execute the build commands. Now, it's time to initiate the actual code analysis by SonarQube. To achieve this, execute the following command in your terminal: **mvn sonar:sonar -Dsonar.host.url=http://3.83.219.83:9000 -Dsonar.login=sqa\_142e5f1586b97303e49c7e67a6b2f60808f67d02**

A black background with white text

Description automatically generated  
A screen shot of a computer

Description automatically generated

* 1. Once the code analysis is performed by SonarQube, you can access the SonarQube web interface to review the results. Navigate to the "Projects" section of the SonarQube dashboard. Here, you'll find a list of projects that have been analyzed, including your recently built project. Click on your project to view its detailed analysis report. The report provides a comprehensive overview of the code quality, including metrics such as code coverage, technical debt, potential vulnerabilities, and other relevant information. The overall quality gate status will be displayed, indicating whether the project has passed or failed the quality checks. You can delve deeper into the report to identify specific issues, prioritize remediation efforts, and improve the overall code quality of your project.

A screenshot of a computer

Description automatically generated

* 1. To containerize the built artifact, a Dockerfile is created to define the image configuration. The Dockerfile is shown as below:

**FROM java:8**

**COPY target/springboot-maven-course-micro-svc-0.0.1-SNAPSHOT.jar app.jar**

**ENTRYPOINT [“java”,”-jar”,”app.jar”]**

Explanation:

This file specifies the base image, which in this case is a Java 8 image, and the subsequent instructions for building the container. The **COPY** instruction transfers the built JAR file, **target/springboot-maven-course-micro-svc-0.0.1-SNAPSHOT.jar**, from the build context to the container's file system and renames it to app.jar. This JAR file will be the primary component of the container. The ENTRYPOINT instruction sets the default command to be executed when the container starts. In this case, the command is java -jar app.jar, which launches the Java application within the container. This ensures that the application is immediately accessible and operational when the container is started.

A screenshot of a computer

Description automatically generated

* 1. Once the Dockerfile is in place, the next step is to establish a secure repository for the Docker image, **Amazon Elastic Container Registry (ECR)** is an ideal choice for this purpose. ECR provides a fully managed Docker container registry that makes it easy to store, manage, and deploy Docker container images. By creating an ECR repository, you can securely store the Docker image and grant access to authorized users, such as administrators, to pull the image and deploy it to different environments like staging or production.

A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated

* 1. To deploy the Docker image to the newly created AWS ECR repository, the **docker push** command is utilized. The full commands can be accessed by clicking on the **‘view push command’**. Before executing this command, it's crucial to ensure that the build server has the necessary permissions to access the ECR repository. In a typical production scenario, this is achieved by creating IAM roles and policies that grant specific permissions to the build server. However, due to the limitations of student accounts, leveraging existing roles or permissions, such as the provided **labrole**, is often necessary. This role should grant the required permissions to interact with ECR, including pushing and pulling Docker images. By following these steps and ensuring proper role-based access control, the Docker image can be successfully deployed to the ECR repository.

A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated  
  
A screen shot of a computer

Description automatically generated  
A computer screen with blue and white text

Description automatically generated  
A screen shot of a computer

Description automatically generated  
A screen shot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated

* 1. Next, a new server is provisioned. This server, configured with a **t2.micro** instance type and **Ubuntu** OS, will be responsible for pulling the Docker image from the **AWS ECR** repository and deploying it. To prepare this server, **Docker** and **AWS CLI** are installed. These tools are essential for interacting with Docker images and the AWS services, respectively. Once the server is configured, it's ready to be used as a deployment node. Open the server in MobaXterm.

A screenshot of a computer

Description automatically generated  
A screenshot of a software application

Description automatically generated  
A screenshot of a web page

Description automatically generated  
A screenshot of a chat

Description automatically generated  
A screenshot of a computer

Description automatically generated

* 1. To empower the newly created deployment server to interact with Docker containers and AWS services, two crucial tools are installed: Docker and the AWS CLI. The provided reference (<https://www.xda-developers.com/how-install-aws-cli-ubuntu/>) offers a detailed guide on installing the AWS CLI on Ubuntu. The images below will show how these packages are installed.

  
  
A black background with white text

Description automatically generated  
A screen shot of a computer

Description automatically generated  
A black background with white text

Description automatically generated  
A screen shot of a computer code

Description automatically generated

* 1. To enable the Continuous Deployment (CD) server to access the Docker image stored in the AWS ECR repository, it's necessary to grant the server appropriate permissions. This is typically achieved by assigning an IAM role with the required permissions to the server's instance. In this case, the labrole is assigned, which provides the necessary access to ECR.

Once the server has the required permissions, the first step from the "View push command" in the AWS ECR console is followed to authenticate the server with ECR. This involves configuring the AWS CLI with the appropriate credentials and region.

Additionally, a security group rule is created to allow inbound traffic on port 8080 from any source. This port will be exposed by the Docker container and used to access the deployed application.

Finally, the Docker container is started using the following command:

**docker container run -name=<repo\_name> -p 8080:8080 -d <image\_name>:<tag>**

This command creates a new container, assigns it a name, maps port 8080 of the host machine to port 8080 of the container, and runs the container in detached mode. The <image\_name>:<tag> specifies the image to be pulled from the ECR repository.

A screenshot of a computer

Description automatically generated  
A screen shot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A screen shot of a computer

Description automatically generated  
A screen shot of a computer

Description automatically generated

* 1. To access the deployed application, the public IP address of the CD server is utilized, along with the specified port (8080) and the appropriate context path and endpoint. In this case, the URL **34.234.96.254:8080/course-svc/getAllDevopsTools** is used to access the **/getAllDevopsTools** endpoint of the **course-svc** application. This URL combines the public IP address of the CD server, the port on which the application is listening, and the specific endpoint within the application. By accessing this URL in a web browser, the desired application functionality can be observed.

A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated