https://github.com/stevekafack/devopexam.git

To create a CI/CD pipeline for converting a legacy development process to a DevOps process, you'll need to follow a series of steps. Here's an outline of the process:

1. **Assess the existing development process**: Understand the current development process, identify pain points, and determine the areas that can be improved through automation and DevOps practices.
2. **Define the goals and requirements**: Clearly define the goals you want to achieve with the CI/CD pipeline. This could include faster software delivery, improved quality, increased collaboration, and enhanced feedback loops. Identify the specific requirements of your project, such as the programming languages, frameworks, and deployment environments you're using.
3. **Choose the right CI/CD tool**: Select a suitable CI/CD tool that aligns with your requirements. Popular choices include Jenkins, GitLab CI/CD, CircleCI, Travis CI, and Azure DevOps. Evaluate their features, integrations, scalability, and ease of use.
4. **Design the pipeline architecture**: Define the stages and tasks that the CI/CD pipeline will consist of. Typically, a CI/CD pipeline includes stages such as code compilation, testing, code quality analysis, artifact creation, deployment, and monitoring. Plan how these stages will be executed and determine the required tools and configurations for each stage.
5. **Implement version control**: Set up a version control system (e.g., Git) if you don't already have one. Create a repository for your project and establish branching strategies (e.g., GitFlow) to manage parallel development efforts.
6. **Configure build automation**: Integrate the CI/CD tool with your version control system and configure it to automatically trigger builds whenever changes are pushed to specific branches. Set up build scripts or configuration files (e.g., Jenkinsfile or .gitlab-ci.yml) to define the build steps and dependencies.
7. **Add automated testing**: Integrate testing frameworks (e.g., JUnit, Selenium, or Cypress) and configure automated tests to run as part of the CI/CD pipeline. Include unit tests, integration tests, and end-to-end tests to ensure code quality and prevent regressions.
8. **Implement code quality checks**: Integrate code analysis tools (e.g., SonarQube or ESLint) to enforce coding standards, detect vulnerabilities, and identify technical debt. Configure the CI/CD pipeline to perform static code analysis and generate reports for developers to review and address.
9. **Create artifact management**: Set up artifact repositories (e.g., Nexus or JFrog Artifactory) to store built artifacts, such as compiled binaries or container images. Configure the CI/CD pipeline to publish artifacts to the repository for future deployments.
10. **Enable deployment automation**: Define deployment scripts or use deployment tools (e.g., Ansible, Kubernetes, or AWS CloudFormation) to automate the deployment process. Configure the CI/CD pipeline to deploy the built artifacts to various environments, such as development, staging, and production, based on predefined triggers.
11. **Implement monitoring and feedback loops**: Integrate monitoring tools (e.g., Prometheus, Grafana, or ELK Stack) to collect and visualize application and infrastructure metrics. Configure the CI/CD pipeline to provide feedback on deployment status, test results, and performance metrics.
12. **Enable continuous feedback and collaboration**: Foster a culture of collaboration by integrating collaboration tools (e.g., Slack or Microsoft Teams) and establishing notification mechanisms to keep the team informed about pipeline status, test results, and deployment activities.
13. **Iterate and improve**: Continuously monitor and evaluate the CI/CD pipeline's performance, gather feedback from the development team, and make improvements based on lessons learned. Implement feedback loops to address issues, enhance automation, and optimize the delivery process

**Implementation of requirement** :

1. Intstall and configure the jenkis architecture on aws
2. **Launch an EC2 instance**: Sign in to the AWS Management Console and navigate to the EC2 service. Launch a new EC2 instance using an Amazon Linux or Ubuntu Server AMI. Ensure that the instance has appropriate resources allocated based on your requirements.
3. **Set up security groups**: Configure security groups to allow inbound traffic on port 8080 (default Jenkins port) from your IP address or specific IP ranges. This will enable access to the Jenkins web interface.
4. **Connect to the EC2 instance**: Once the instance is running, connect to it using SSH. You can use a tool like PuTTY (Windows) or the SSH command (Linux/Mac) to establish the connection.
5. **Install Java**: Jenkins requires Java to run. Install Java Development Kit (JDK) on your EC2 instance by executing the appropriate commands for your Linux distribution. For example, on Amazon Linux, you can use the following command: