Lab Exercise: CI/CD Basics with GitLab for DevOps Beginners

Objective:

In this lab, you will learn the fundamental tasks for setting up **CI/CD pipelines** using **GitLab**. You will create a repository in GitLab, define a simple CI/CD pipeline using a .gitlab-ci.yml file, and use GitLab's CI/CD tools to automate testing and deployment. By the end of this lab, you will have a basic understanding of how CI/CD pipelines help automate software testing and delivery in a DevOps environment.

Prerequisites:

- A GitLab account (Sign up at GitLab if you don't have one).
- Basic knowledge of version control and Git.
- Basic understanding of software development lifecycle (SDLC) and DevOps practices.

Step 1: Create a GitLab Repository

1. **Log in to GitLab**:
    - Navigate to GitLab and log in with your credentials.
2. **Create a new project**:
    - In the GitLab dashboard, click **New Project**.
    - **Project name**: Enter a name for your project (e.g., **devops-demo**).
    - **Visibility level**: Choose **Public**.
    - Leave the default settings and click **Create project**.

Step 2: Set Up a Basic CI/CD Pipeline

1. **Create a .gitlab-ci.yml file**:

- In your GitLab repository, click **New file**.
- Name the file .gitlab-ci.yml. This file defines the stages, jobs, and scripts for your CI/CD pipeline.

2. **Define the CI/CD pipeline**:

   Here's what each part simulates:

   Build stage: Mimics compiling or setting up your project.
   Test stage: Simulates running tests to check if the code works correctly.
   Deploy stage: Mimics deploying the project to a server or production environment.

   - Add the following basic CI/CD configuration to the .gitlab-ci.yml file:

```
stages:
  - build
  - test
  - deploy

# Build Stage
build-job:
  stage: build
  script:
    - echo "Building the project..."

# Test Stage
test-job:
  stage: test
  script:
    - echo "Running tests..."
    - echo "All tests passed!"

# Deploy Stage
deploy-job:
  stage: deploy
  script:
    - echo "Deploying the project..."
    - echo "Deployment complete!"
```

   - This simple pipeline has three stages: build, test, and deploy, and each stage contains a job that prints a message.

3. **Commit the file**:
   - After adding the YAML content, click **Commit changes** to save the .gitlab-ci.yml file to the repository.

Step 3: Run the CI/CD Pipeline

1. **Trigger the pipeline**:
   - Once the .gitlab-ci.yml file is committed, GitLab automatically triggers the CI/CD pipeline. You can view the pipeline status under the **CI/CD** > **Pipelines** section in your project.
2. **View the pipeline stages**:
   - Go to **Build** > **Pipelines** and click on the running pipeline.
   - You will see the three stages (build, test, and deploy) and the jobs being executed one after the other.
   - Click on each stage to view the output logs (e.g., "Building the project...", "Running tests...", etc.).

Step 4: Simulate a Test Failure

1. **Add a test case**:
   - Modify the .gitlab-ci.yml file to add a simple test command:

```
stages:
  - build
  - test
  - deploy

# Build Stage
build-job:
  stage: build
  script:
    - echo "Building the project..."

# Test Stage
test-job:
  stage: test
  script:
    - echo "Running tests..."
    - exit 1  # Simulating a test failure

# Deploy Stage
deploy-job:
  stage: deploy
  script:
    - echo "Deploying the project..."
    - echo "Deployment complete!"
  when: on_success  # Only deploy if tests pass
```

- In this updated file, the test-job simulates a test failure using exit 1. The deploy-job will only run if the test-job passes (when: on_success).

2. Commit the changes:
   - Commit the changes to the repository. This will trigger a new pipeline.
3. Review the pipeline execution:
   - Go to the CI/CD > Pipelines section and review the new pipeline.
   - The pipeline should fail at the test-job stage, and the deploy-job will not run because the test failed.

   In a real project, the test stage of the CI/CD pipeline typically involves running automated tests to verify that the code works as expected. These tests can vary in complexity and depth, depending on the project's requirements. Here's what usually happens in the test stage:
   - *Execute the Code and Monitor for Errors:*
     1. *The simplest form of testing involves running the code to check for syntax errors or obvious runtime issues (e.g., does the code compile or execute without crashing?).*
     2. *These are often called unit tests, which focus on testing small parts of the application (individual functions or components) to ensure they work correctly.*
   - *Deeper Levels of Testing:*
     1. *Integration tests: Test how different parts of the code (modules, components) work together. For example, verifying that a function interacts correctly with a database.*
     2. *End-to-end (E2E) tests: Simulate real user interactions, testing the full application from start to finish, including the user interface, APIs, and back-end.*
     3. *Regression tests: Ensure that new changes don't break existing functionality by running previously written test cases.*
     4. *Performance tests: Monitor how well the code performs under specific conditions, such as high traffic or data loads, to ensure the system can handle expected usage.*
     5. *Security tests: Look for vulnerabilities in the code, such as potential exploits or weak authentication.*

Step 5: Set Up a Simple Deployment

1. **Add a deployment script**:
   - Modify the .gitlab-ci.yml file to deploy your project to a web server. For this lab, we will simulate deployment using a placeholder script.

```
stages:
  - build
  - test
  - deploy

# Build Stage
build-job:
  stage: build
  script:
    - echo "Building the project..."

# Test Stage
test-job:
  stage: test
  script:
    - echo "Running tests..."
    - echo "All tests passed!"

# Deploy Stage
deploy-job:
  stage: deploy
  script:
    - echo "Deploying the project to the server..."
    - echo "Project successfully deployed!"
```

   - In this case, the deploy-job simulates a successful deployment.
2. **Commit the changes**:
   - Commit the changes, which will trigger the pipeline again.
3. **Verify successful deployment**:
   - The pipeline should now run through all stages (build, test, and deploy) and finish successfully.
   - You can view the deployment logs by clicking on the deploy-job stage in the pipeline.

   Here's what usually happens in the deployment stage:
   - *Automated Deployment:*
     1. *The code is packaged and automatically installed or deployed to a target environment (e.g., a web server, cloud infrastructure like AWS or Google Cloud, or a container platform like Kubernetes).*
     2. *This can involve copying files, running deployment scripts, or using deployment tools to push the new version of the software.*
   - *Deployment to Different Environments:*
     1. *Staging environment: This is a pre-production environment where the final version of the software is deployed to perform final checks, including user acceptance testing (UAT) and performance tests, before going live.*

        2. *Production environment: Once everything is validated, the code is deployed to the live environment, where real users interact with the application.*
    - *Other Deployment Tasks:*
        1. *Database migrations: The deployment might involve running scripts to update or migrate databases to support the new version of the software.*
        2. *Configuration updates: Update environment variables, API keys, and other configuration details for the new code to run correctly.*
        3. *Rollback plans: In case of failure, many deployment processes include a rollback mechanism to revert to a previous stable version of the software.*
        4. *Notifications: After deployment, notifications might be sent to team members or DevOps engineers to inform them that the deployment was successful.*

Step 6: Clean Up (Optional)

1. **Delete the GitLab project**:
    - If you no longer need the project, go to **Settings** > **General** > **Advanced**.
    - Scroll down to the **Remove project** section and click **Delete project**.