Lab 6: Troubleshooting Pipelines and Job Failures

Overview

This lab will cover:

1. Using **rules** to run jobs on failure (e.g., cleaning up resources after a failed deployment).
2. Handling **job failures** and **pipeline timeouts**.
3. A **demo** illustrating YAML rules to clean up after a failed VM deployment.

By the end of this guide, you will have a better understanding of how to handle job failures and ensure resources are properly managed.

Prerequisites

1. Basic understanding of GitLab CI/CD and YAML syntax.
2. A GitLab repository with CI/CD enabled.
3. Familiarity with deploying resources (e.g., VMs) using GitLab CI/CD.

Part 1: Using Rules to Run Jobs on Failure

1.1 What Are Rules?

Rules in GitLab CI/CD determine when jobs are executed. You can set conditions to run jobs based on the success or failure of previous jobs, making it easier to manage workflows, especially in error scenarios.

1.2 Running Cleanup Jobs on Failure

For example, you may want to run a cleanup job if a deployment fails. This helps ensure that resources are released and not left hanging.

Example Configuration

```
stages:
  - deploy
  - cleanup

deploy_job:
  stage: deploy
  script:
    - echo "Deploying the application..."
    - exit 1  # Simulate a failure

cleanup_job:
  stage: cleanup
  script:
    - echo "Cleaning up resources..."
  rules:
    - if: '$CI_JOB_STATUS == "failed"'
```

Explanation: In this example, cleanup_job runs if deploy_job fails, ensuring that any necessary cleanup occurs.

Part 2: Handling Job Failures and Pipeline Timeouts

2.1 Monitoring Job Failures

Job Failure: A job fails when any command in the job script returns a non-zero exit code. Pipeline Timeout: Pipelines can timeout if they exceed the configured duration for any job.

2.2 Best Practices for Handling Failures

1. **Use Exit Codes**: Ensure that your scripts return appropriate exit codes.
2. **Retry Failed Jobs**: You can use the retry keyword to automatically retry a failed job.

Example of Retrying a Job

```
retry_job:
  script:
    - echo "This job may fail occasionally."
    - exit 1  # Simulated failure
  retry: 2  # Retry this job up to 2 times on failure
```

3. **Set Timeout for Jobs**: You can specify a timeout to prevent jobs from running indefinitely.

Example Timeout Configuration

```
timeout_job:
  script:
    - echo "This job has a timeout."
  timeout: 1m  # Set timeout to 1 minute
```

Part 3: Demo – Example of YAML Rules to Clean Up After a Failed VM Deployment

3.1 Scenario

In this demo, we will create a pipeline that simulates a VM deployment and runs a cleanup job if the deployment fails.

3.2 Lab Setup

Define Stages and Jobs in `.gitlab-ci.yml`:

```
stages:
  - deploy
  - cleanup

deploy_vm:
  stage: deploy
  script:
    - echo "Deploying VM..."
    - exit 1  # Simulating failure

cleanup_vm:
  stage: cleanup
  script:
    - echo "Cleaning up VM resources..."
  rules:
```

```
    - if: '$CI_JOB_STATUS == "failed"'
      when: always
```

Explanation:

- `deploy_vm` simulates a VM deployment that fails.
- `cleanup_vm` runs only if `deploy_vm` fails, ensuring resources are cleaned up.

Run the Pipeline:

- Push the configuration to your GitLab repository.
- Trigger the pipeline and observe the behavior when the deployment job fails. The cleanup job should execute afterward.

Summary

This lab covers:

Using rules to manage job execution based on previous job status. Strategies for handling job failures and timeouts. A demo of using rules to clean up resources after a failed deployment.