Lab: Security Best Practices

Overview

This lab will cover:

1. **Securing GitLab repositories and pipelines**: Protect repositories, limit access, and secure sensitive information.
2. **Best practices for GitLab CI/CD security configurations**: Use secure configurations to prevent unauthorized access and reduce risks.
3. **Demo**: Implement security measures in a CI/CD pipeline, including secret management and access control.

By the end of this guide, you will understand how to configure secure pipelines and manage sensitive information effectively.

Prerequisites

1. A GitLab repository with CI/CD enabled.
2. GitLab admin access or owner-level permissions to configure repository settings.
3. Familiarity with GitLab CI/CD and YAML syntax.

Part 1: Securing GitLab Repositories and Pipelines

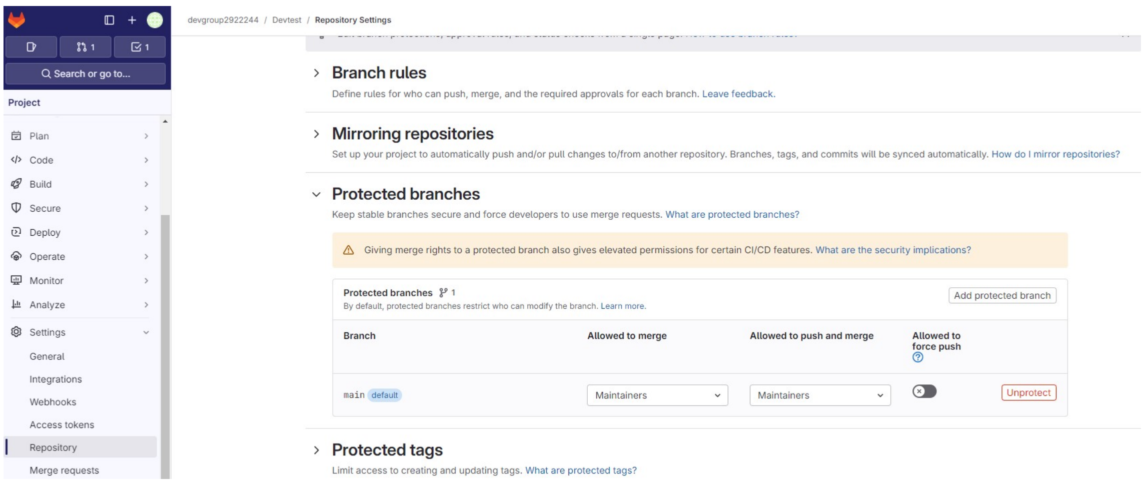
1.1 Repository Access Control

To protect your repository, follow these access control best practices:

- **Limit Access**: Use GitLab's access roles (Guest, Reporter, Developer, Maintainer, Owner) to restrict permissions.
- **Branch Protection**: Enable branch protection for critical branches (e.g., `main` or `production`) to restrict direct pushes and require code review.

Example Steps:

1. Go to **Settings** > **Repository** > **Protected Branches**.
2. Choose the branch you want to protect (e.g., `main`).
3. Set permissions to **allow only Maintainers** to push and require a **merge request** for changes.



Branch rules

Define rules for who can push, merge, and the required approvals for each branch. Leave feedback.

Mirroring repositories

Set up your project to automatically push and/or pull changes to/from another repository. Branches, tags, and commits will be synced automatically. How do I mirror repositories?

Protected branches

Keep stable branches secure and force developers to use merge requests. What are protected branches?

Giving merge rights to a protected branch also gives elevated permissions for certain CI/CD features. What are the security implications?

Protected branches 1

By default, protected branches restrict who can modify the branch. Learn more.

Branch	Allowed to merge	Allowed to push and merge	Allowed to force push
main default	Maintainers	Maintainers	✗

Protected tags

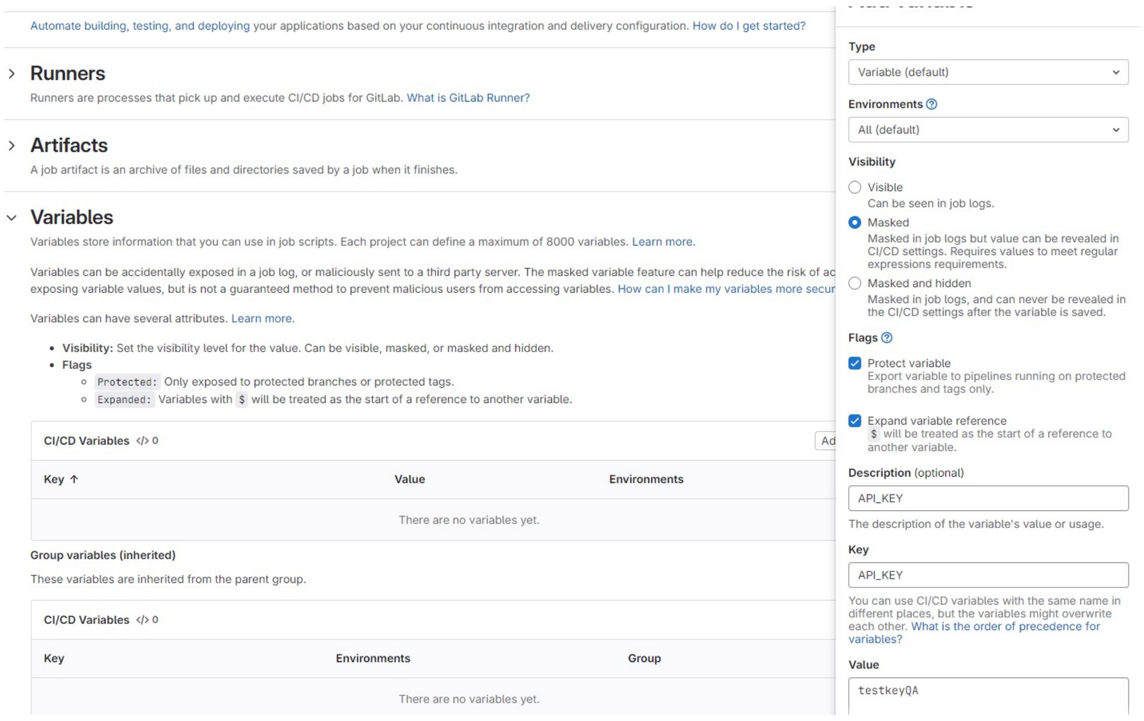
Limit access to creating and updating tags. What are protected tags?

Part 2: Best Practices for GitLab CI/CD Security Configurations

2.1 Secret Management

Avoid hardcoding sensitive information (like API keys) in `.gitlab-ci.yml` files. Instead, use GitLab CI/CD environment variables.

1. Go to **Settings > CI/CD** and expand the **Variables** section.
2. Add sensitive information (e.g., `API_KEY`, `DB_PASSWORD`) as **masked** environment variables to hide values in the CI/CD logs.



Automate building, testing, and deploying your applications based on your continuous integration and delivery configuration. How do I get started?

Runners
Runners are processes that pick up and execute CI/CD jobs for GitLab. What is GitLab Runner?

Artifacts
A job artifact is an archive of files and directories saved by a job when it finishes.

Variables
Variables store information that you can use in job scripts. Each project can define a maximum of 8000 variables. Learn more.

Variables can be accidentally exposed in a job log, or maliciously sent to a third party server. The masked variable feature can help reduce the risk of ac
exposing variable values, but is not a guaranteed method to prevent malicious users from accessing variables. How can I make my variables more secur

Variables can have several attributes. Learn more.

- **Visibility:** Set the visibility level for the value. Can be visible, masked, or masked and hidden.
- **Flags**
  - `Protected:` Only exposed to protected branches or protected tags.
  - `Expanded:` Variables with `$` will be treated as the start of a reference to another variable.

CI/CD Variables </> 0

Key ↑	Value	Environments
There are no variables yet.		

Group variables (inherited)
These variables are inherited from the parent group.

CI/CD Variables </> 0

Key	Environments	Group
There are no variables yet.		

Type: Variable (default)
Environments: All (default)
Visibility
- Visible — Can be seen in job logs.
- [✓ Masked] — Masked in job logs but value can be revealed in CI/CD settings. Requires values to meet regular expressions requirements.
- Masked and hidden — Masked in job logs, and can never be revealed in the CI/CD settings after the variable is saved.

Flags
- ✓ Protect variable — Export variable to pipelines running on protected branches and tags only.
- ✓ Expand variable reference — `$` will be treated as the start of a reference to another variable.

Description (optional): API_KEY
The description of the variable's value or usage.

Key: API_KEY
You can use CI/CD variables with the same name in different places, but the variables might overwrite each other. What is the order of precedence for variables?

Value: testkeyQA

`.gitlab-ci.yml` Usage:

```
stages:
  - build

build_job:
  stage: build
  script:
    - echo "Building the application..."
    - echo "Using API key $API_KEY"  # $API_KEY is a masked variable from GitLab
settings
```

2.2 Limit Job Permissions

Use rules to restrict jobs to specific branches or pipeline events.

```
deploy_job:
  stage: deploy
  script:
    - echo "Deploying to production..."
  rules:
    - if: '$CI_COMMIT_REF_NAME == "main"'  # Only allow deployment on the main branch
```

Part 3: Demo – Security Measures in a CI/CD Pipeline

In this demo, you'll set up a secure pipeline with secret management and access control.

3.1 Demo Setup

1. Create Environment Variables:

- Go to **Settings > CI/CD > Variables**.
- Add a variable `SECRET_TOKEN`, mark it as **masked**, and use it in your pipeline.

2. Configure the Pipeline:

Add the following `.gitlab-ci.yml` file to your repository.

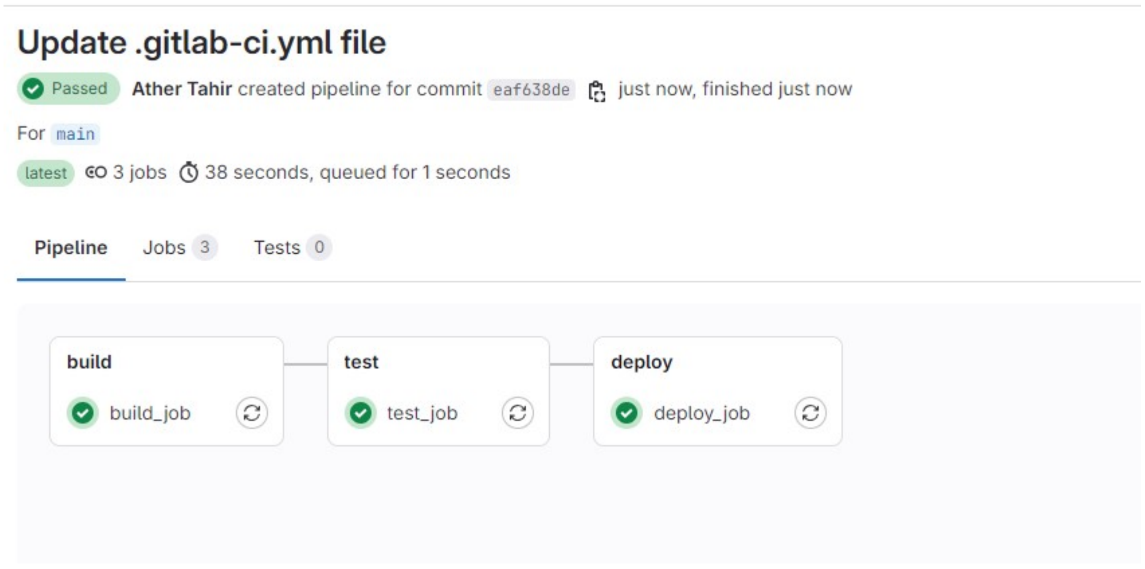
```
stages:
  - build
  - test
  - deploy

build_job:
  stage: build
  script:
    - echo "Building application..."

test_job:
  stage: test
  script:
    - echo "Running tests..."
  rules:
    - if: '$CI_COMMIT_REF_NAME == "main"'
    - if: '$CI_COMMIT_REF_NAME =~ /^release-.*$/'

deploy_job:
  stage: deploy
  script:
    - echo "Deploying with secret token..."
    - echo $SECRET_TOKEN  # Uses the masked variable SECRET_TOKEN
  rules:
    - if: '$CI_COMMIT_REF_NAME == "main"'  # Deploy only on main branch
    - when: manual  # Manual trigger to control deployments
```

3. **Explanation:**

- `build_job`: Runs unconditionally as a basic build step.
- `test_job`: Executes on the main branch and release branches only.
- `deploy_job`: Runs manually on the main branch, using SECRET_TOKEN securely.

4. **Run the Pipeline:**

- Push changes to the repository and observe how jobs behave according to branch and rule restrictions.
- Check that `SECRET_TOKEN` is masked in logs.

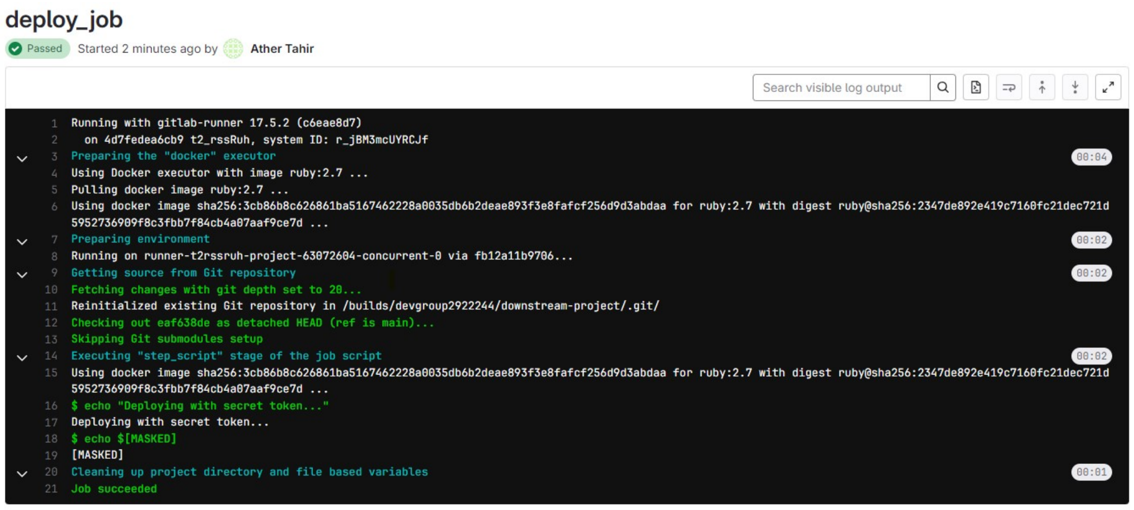


Update .gitlab-ci.yml file

Passed Ather Tahir created pipeline for commit eaf638de just now, finished just now

For main

latest 3 jobs 38 seconds, queued for 1 seconds



deploy_job

Passed Started 2 minutes ago by Ather Tahir

```
Running with gitlab-runner 17.5.2 (c6eae8d7)
  on 4d7fedea6cb9 t2_rssRuh, system ID: r_jBM3mcUYRCJf
Preparing the "docker" executor                                                     00:04
Using Docker executor with image ruby:2.7 ...
Pulling docker image ruby:2.7 ...
Using docker image sha256:3cb86b8c626861ba5167462228a0035db6b2deae893f3e8fafcf256d9d3abdaa for ruby:2.7 with digest ruby@sha256:2347de892e419c7160fc21dec721d
5952736909f8c3fbb7f84cb4a07aaf9ce7d ...
Preparing environment                                                               00:02
Running on runner-t2rssruh-project-63072604-concurrent-0 via fb12a11b9706...
Getting source from Git repository                                                  00:02
Fetching changes with git depth set to 20...
Reinitialized existing Git repository in /builds/devgroup2922244/downstream-project/.git/
Checking out eaf638de as detached HEAD (ref is main)...
Skipping Git submodules setup
Executing "step_script" stage of the job script                                     00:02
Using docker image sha256:3cb86b8c626861ba5167462228a0035db6b2deae893f3e8fafcf256d9d3abdaa for ruby:2.7 with digest ruby@sha256:2347de892e419c7160fc21dec721d
5952736909f8c3fbb7f84cb4a07aaf9ce7d ...
$ echo "Deploying with secret token..."
Deploying with secret token...
$ echo $[MASKED]
[MASKED]
Cleaning up project directory and file based variables                              00:01
Job succeeded
```

Summary

This lab covered:

1. Security practices for access control and branch protection.
2. Configurations for securely handling secrets and setting rules on sensitive jobs.
3. A practical demo on using secure variables and controlled job triggers.