

Course 3 – Sprint 2: Continuous Integration

History

Grady Booch, proposed the term CI in 1994.

In 1997, Kent Beck and Ron Jeffries invented Extreme Programming (XP) while on the Chrysler Comprehensive Compensation System project, including continuous integration.

CruiseControl, one of the first open-source CI tools, was released in 2001.

Continuous Integration:

Continuous Integration works by pushing small code chunks to our Git repository, and to every push, run a pipeline of scripts to build, test, and validate the code changes before merging them into the main branch.

CI Concepts:

GitLab CI/CD uses several concepts to describe and run your build and deploy.

Concept	Description
Pipelines	Structure your CI/CD process through pipelines.
CI/CD variables	Reuse values based on a variable/value key pair.
Environments	Deploy your application to different environments (e.g., staging, production).
Job artifacts	Output, use, and reuse job artifacts.
Cache dependencies	Cache your dependencies for a faster execution.
GitLab Runner	Configure your own runners to execute your scripts.

Pipeline efficiency	Configure your pipelines to run quickly and efficiently.
Test cases	Configure your pipelines to run quickly and efficiently.

CI/CD process overview

To use GitLab CI/CD:

1. Ensure you have runners available to run your jobs. If you don't have a runner, install GitLab Runner and register a runner for your instance, project, or group.
2. Create a gitlab-ci.yml file at the root of your repository. This file is where you define your CI/CD jobs.

When you commit the file to your repository, the runner runs your jobs. The job results are displayed in a pipeline.

NOTE:

For installation GitLab Runner, refer <https://docs.gitlab.com/runner/install>

For registering your GitLab Runner, refer <https://docs.gitlab.com/runner/register/>

In GitLab, runners are agents that run your CI/CD jobs. For Shared runners refer (https://docs.gitlab.com/ee/ci/runners/runners_scope.html).

To view available runners:

- Go to **Settings > CI/CD** and expand **Runners**.

As long as you have at least one runner that's active, with a green circle next to it, you have a runner available to process your jobs.

If no runners are listed on the **Runners** page in the UI, you or an administrator must create one.

If you are testing CI/CD, you can install GitLab Runner and register runners on your local machine. When your CI/CD jobs run, they run on your local machine.

CI/CD Configuration file: **.gitlab-ci.yml file**

The **.gitlab-ci.yml** file is a [YAML](#) file where you configure specific instructions for GitLab CI/CD.

In this file, you define:

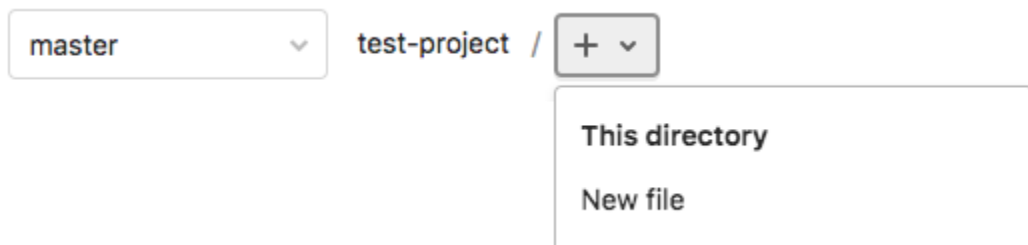
- The structure and order of jobs that the runner should execute.
- The decisions the runner should make when specific conditions are encountered.

For example, you might want to run a suite of tests when you commit to any branch except the default branch. When you commit to the default branch, you want to run the same suite, but also publish your application.

All of this is defined in the `.gitlab-ci.yml` file.

To create a `.gitlab-ci.yml` file:

1. On the left sidebar, select **Project information > Details**.
2. Above the file list, select the branch you want to commit to, click the plus icon, then select **New file**:



3. For the **Filename**, type `.gitlab-ci.yml` and in the larger window, paste this sample code:

```
build-job:
  stage: build
  script:
    - echo "Hello, $GITLAB_USER_LOGIN!"

test-job1:
  stage: test
  script:
    - echo "This job tests something"

test-job2:
```

```

stage: test

script:
  - echo "This job tests something but takes more time than test-job1."
  - echo "After the echo commands complete, it runs the sleep command for 20
seconds"
  - echo "which simulates a test that runs 20 seconds longer than test-job1"
  - sleep 20

deploy-prod:
  stage: deploy
  script:
    - echo "This job deploys something from the $CI_COMMIT_BRANCH branch."
  
```

`$GITLAB_USER_LOGIN` and `$CI_COMMIT_BRANCH` are predefined variables that populate when the job runs.

4. Click **Commit changes**.

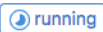


The pipeline starts when the commit is committed.

View the status of your pipeline and jobs

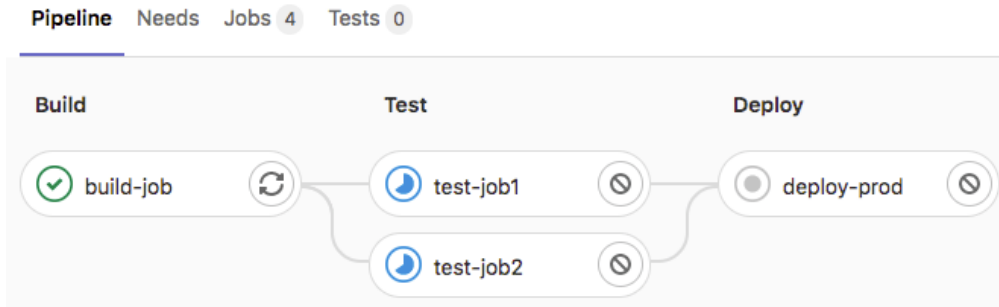
When you committed your changes, a pipeline started.

- 1) To view your pipeline:
 - Go to **CI/CD > Pipelines**.

A pipeline with three stages should be displayed:

Status	Pipeline	Triggerer	Commit	Stages
	#217408060 latest		master -> 271ad0cd Update .gitlab-ci.yml	

2) To view a visual representation of your pipeline, click the pipeline ID.



3) To view details of a job, click the job name, for example, **deploy-prod**.

HYPERLINK "https://docs.gitlab.com/ee/ci/quick_start/img/job_details_v13_6.png"

✓ passed Job #855275091 triggered 23 minutes ago by Suzanne Selhorn

```

1 Running with gitlab-runner 13.6.0-rc1 (d83ac56c)
2 on docker-auto-scale ed2dce3a
3 ✓ Preparing the "docker+machine" executor
4 Using Docker executor with image ruby:2.5 ...
5 Pulling docker image ruby:2.5 ...
6 Using docker image sha256:b7280b81558d31d64ac82aa66a9540e04baf9d15abb8fff
  ed62cd60e4fb5bf4132943d6fa2688 ...
7
8 ✓ Preparing environment
9 Running on runner-ed2dce3a-project-16381496-concurrent-0 via runner-ed2dce3a
10
11 ✓ Getting source from Git repository
12 $ eval "$CI_PRE_CLONE_SCRIPT"
13 Fetching changes with git depth set to 50...
14 Initialized empty Git repository in /builds/sselhorn/test-project/.git/
15 Created fresh repository.
16 Checking out 7353da73 as master...
17 Skipping Git submodules setup
18
19 ✓ Executing "step_script" stage of the job script
20 $ echo "This job deploys something from the $CI_COMMIT_BRANCH branch."
21 This job deploys something from the master branch.
22
23 ✓ Cleaning up file based variables
24
25 Job succeeded
  
```

If the job status is **stuck**, check to ensure a runner is properly configured for the project.

CI/CD PIPELINES

Pipelines are the top-level component of continuous integration, delivery, and deployment.

Pipelines comprise:

- Jobs, which define *what* to do. For example, jobs that compile or test code.
- Stages, which define *when* to run the jobs. For example, stages that run tests after stages that compile the code.

Jobs are executed by **runners**. Multiple jobs in the same stage are executed in parallel, if there are enough concurrent runners.

If *all* jobs in a stage succeed, the pipeline moves on to the next stage.

If *any* job in a stage fails, the next stage is not (usually) executed, and the pipeline ends early.

In general, pipelines are executed automatically and require no intervention once created. However, there are also times when you can manually interact with a pipeline.

A typical pipeline might consist of four stages, executed in the following order:

- A **build** stage, with a job called **compile**.
- A **test** stage, with two jobs called **test1** and **test2**.
- A **staging** stage, with a job called **deploy-to-stage**.
- A **production** stage, with a job called **deploy-to-prod**.

To Configure, Run and View Pipeline refer this link <https://docs.gitlab.com/ee/ci/pipelines/>

JOBS

Pipeline configuration begins with jobs. Jobs are the most fundamental element of a **.gitlab-ci.yml** file.

Jobs are:

- Defined with constraints stating under what conditions they should be executed.
- Top-level elements with an arbitrary name and must contain at least the **script** clause.
- Not limited in how many can be defined.

For example:

job1:

script: "execute-script-for-job1"

job2:

script: "execute-script-for-job2"

For More details about Jobs please refer to <https://docs.gitlab.com/ee/ci/jobs/>

For Environment and deployment details, refer to <https://docs.gitlab.com/ee/ci/environments/>