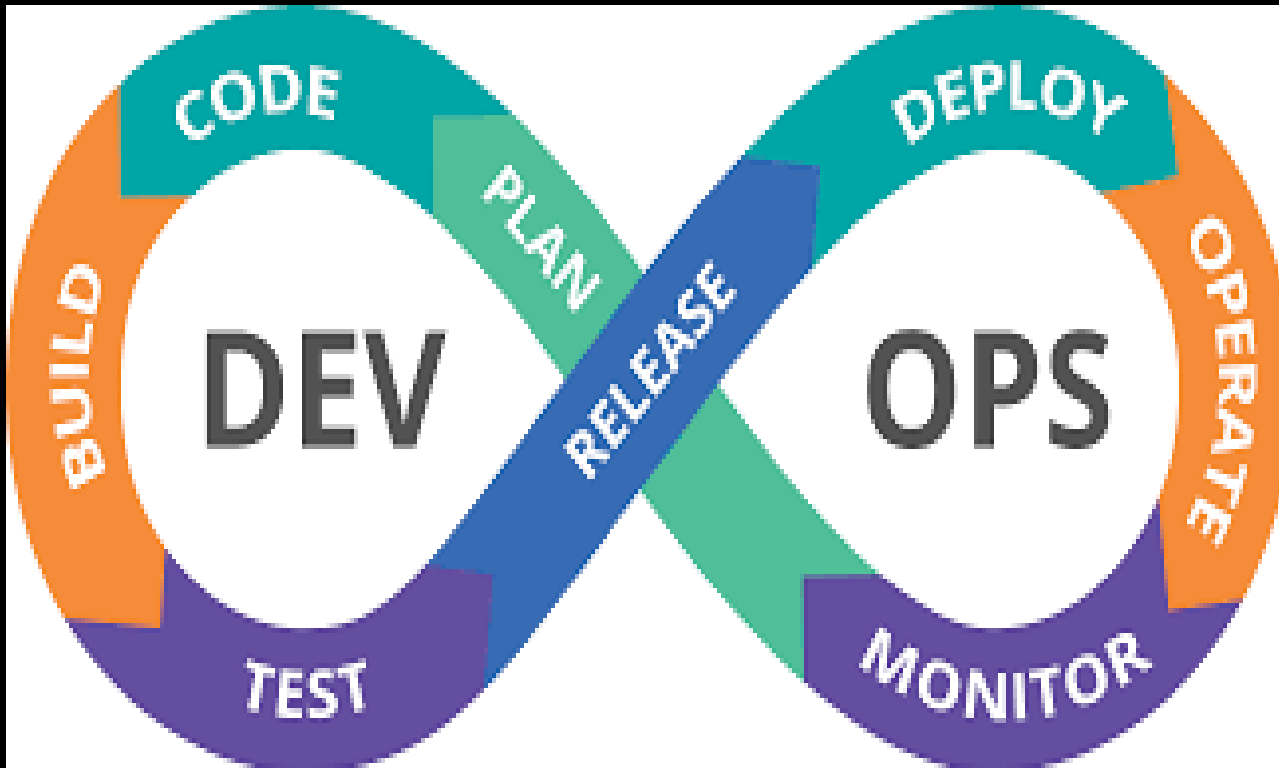




# **Continuous Integration/Continuous Deployment (CI/CD)**

# CI/CD → DevOps



# Continuous Integration (CI) - Definition

- CI is a software development practice where members of a team integrate their work frequently (daily) leading to multiple integrations per day. Each integration is verified by an automated build (including tests) to detect integration errors as quickly as possible.

*Martin Fowler*

- In software engineering, CI implements continuous processes of applying quality control - small pieces of effort, applied frequently. CI aims to improve the quality of software, and to reduce the time taken to deliver it, by replacing the traditional practice of applying quality control after completing all development.

# CI – How and Why.

- How:
  1. Maintain a single (central) source repository.
  2. Automate the build tasks.
  3. Everyone commits 'every day' to the central repository.
  4. Do the build on the CI machine.
  5. Keep the build fast, e.g. caching 3rd party libraries.
  6. Test the application on a clone of the production environment.
  7. Make it easy for everyone to get the latest executable.
  8. Make the process transparent for everyone.

# CI – How and Why.

- Why:
  1. Detect development problems earlier.
  2. Find and remove bugs earlier.
  3. Reduce risks of cost, schedule and budget overrun.
  4. Increase project visibility.
  5. Deliver new features and get user feedback more rapidly.
  6. Improve team cohesion.
  7. Have greater availability of deployable software.

# CI – Objective.

- Better:
  - Build better quality software ..... because it's tested early and often ..... adheres to best practices & coding standards.
- Faster:
  - Test in parallel with development (Agile), not at the end (Waterfall).
  - Do not have 'integration days'.
  - Application builds should be a non-event.
- Cheaper:
  - Identify defects earlier.
  - Fix when it's least costly.
  - Testing should be repeatable.

# Continuous Delivery/Deployment

- Continuous Delivery is a development discipline where you build software in such a way that it can be released to production at any time.
- Continuous Deployment means that every software change goes through the pipeline and automatically gets deployed into production, resulting in many production deployments every day.

# CD – How and Why.

- How:
  1. Team must continuously integrating new features into the software.
  2. Run automated tests.
  3. Automatically deploy the application to a production-like environment, termed a staging environment.
- Why:
  1. Reduce deployment risks.
  2. Change the version in production more rapidly.
  3. Able to release new version of the application at the 'push of a button'.
  4. Get user feedback earlier.



# CI/CD - Summary

## Continuous Integration



## Continuous Delivery



## Continuous Deployment



# CI metrics

- Is our CI process successful?
- Possible metrics including:
  - Successful Build Rate.
  - Build Repair Rate.
  - Total Numbers of Static Tool Errors
    - Ex. Lint, Checkstyle, PMD, Findbugs
  - Unit Testing Line Coverage.
  - Functional Testing Line Coverage.

# CD – How and Why.

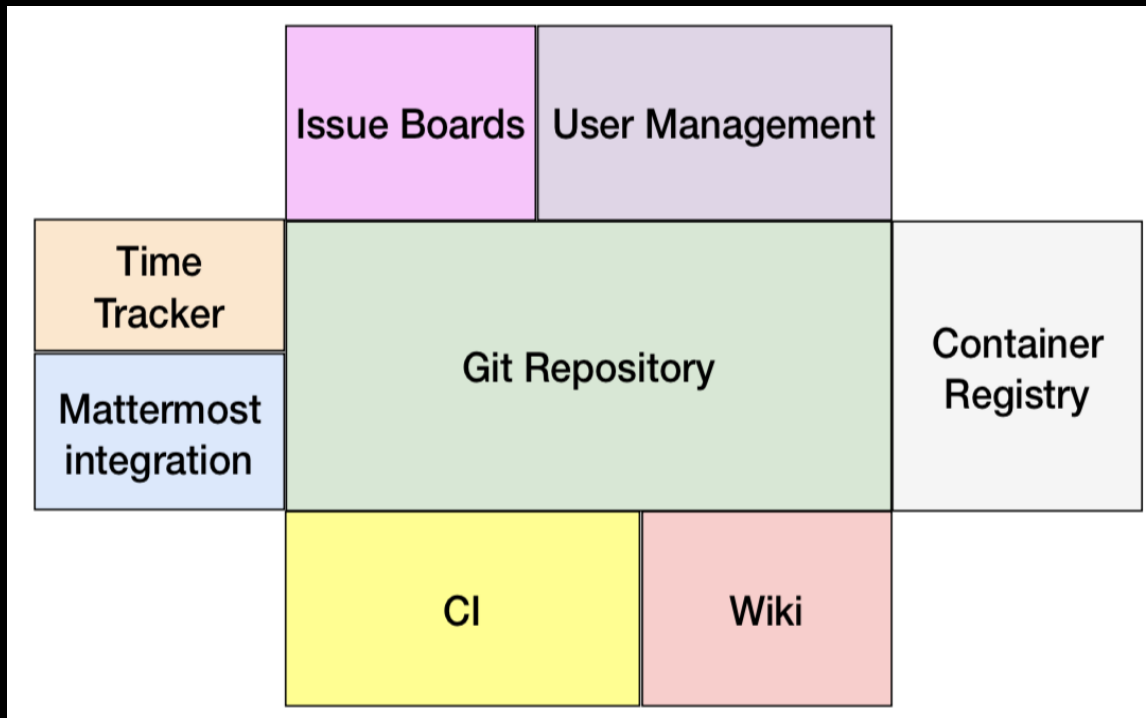
- Pay attention to your builds:
  - They should provide immediate feedback.
  - They should be easily accessible.
  - They should require no developer effort.
- CI Metrics Matter:
  - Identify key metrics and track them visually.
  - Act on them immediately.

# CI/CD - Platforms



# GitLab.

- A Git-based hosting and collaboration platform
- Open source.
- Hosted (free) or on premise.
- Actively maintained.

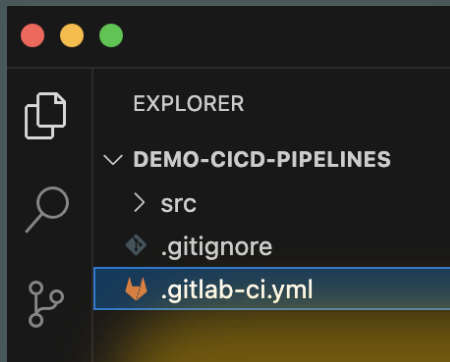


# GitLab CI.

- What:
  - Fully integrated with Gitlab repository
  - Version control of CI/CD scripts.
  - Git hooks.
  - Built on containerization model (optional).
- Why:
  - Code and build scripts in the same repo.
  - Easy to get started.
  - Scalable.
  - Isolated test environment.

# GitLab CI - Concepts.

- Three constructs:
  1. Pipelines,
  2. Stages, and
  3. Jobs.
  - Declare a CI/CD process using these constructs.
  - Pipeline >> Stages >> Jobs.
- A pipeline is a group of jobs that get executed in stages (batches).
  - All jobs in a stage are executed in parallel
  - If all stage jobs succeed, the pipeline moves to the next stage.
  - If a stage job fails, the next is not executed.
- Pipelines are defined in ***.gitlab-ci.yml*** (mandatory) in the project base *folder*.



# Demo

- A pipeline with 4 jobs spread across 3 stages.

```
.gitlab-ci.yml M X
.gitlab-ci.yml
1
2  stages:
3    - stage1
4    - stage2
5    - stage3
6
7  jobA:
8    stage: stage1
9    script:
10     - echo "Start of '$CI_PIPELINE_ID' pipeline "
11     - echo "This is job '$CI_JOB_NAME' in stage '$CI_JOB_STAGE'"
12
13  jobB:
14    stage: stage2
15    script:
16     - echo "This is job '$CI_JOB_NAME' in stage '$CI_JOB_STAGE'"
17     - exit 1
18
19  jobC:
20    stage: stage2
21    script:
22     - echo "This is job '$CI_JOB_NAME' in stage '$CI_JOB_STAGE'"
23
24  jobD:
25    stage: stage3
26    script:
27     - echo "This is job '$CI_JOB_NAME' in stage '$CI_JOB_STAGE'"
28     - echo "End of '$CI_PIPELINE_ID' pipeline "
29
```



# Demo – Pipeline executions

- The pipeline is executed when the repository is updated (git push)

The screenshot shows the GitLab Pipelines interface for the repository `gitlab.com/oconnordiarmuid/demo-cicd-pipelines`. The page displays a list of pipeline executions with columns for Status, Pipeline, Created by, and Stages.

| Status   | Pipeline   | Created by | Stages                                       |
|--|--|------------|--|
| <span>Failed</span><br>00:01:00<br>2 minutes ago | <b>Force job failure</b><br>#1519841833<br>main ed49772b<br>latest |            | <span>✓</span> <span>✗</span> <span>»</span> |
| <span>Passed</span><br>00:01:30<br>3 minutes ago | <b>Correct job stages</b><br>#1519840356<br>main ab7974dc          |            | <span>✓</span> <span>✓</span> <span>✓</span> |
| <span>Passed</span><br>00:01:31<br>7 minutes ago | <b>Use job names</b><br>#1519832208<br>main 3d6444a5               |            | <span>✓</span> <span>✓</span> <span>✓</span> |

# Demo – Job execution

- A job's script is executed inside a container (configurable)

Amazon W x | diarmuidoc x | docker-con x | jobA (#822 x | Mail - Diarmuid x | Predefined x | How to fail x | +

gitlab.com/oconnordiarmuid/demo-cicd-pipelines/-/jobs/8226798573

Diarmuid O Connor / demo-cicd-pipelines / Jobs / #8226798573

Search visible log output

```
9 Getting source from Git repository 00:01
10 Fetching changes with git depth set to 20...
11 Initialized empty Git repository in /builds/oconnordiarmuid/demo-cicd-pipelines/.git/
12 Created fresh repository.
13 Checking out ab7974dc as detached HEAD (ref is main)...
14 Skipping Git submodules setup
15 $ git remote set-url origin "${CI_REPOSITORY_URL}"
16 Executing "step_script" stage of the job script 00:00
17 Using docker image sha256:243309b48f4ab04a5de198e1ef7ec6b224a276924f096f188dd6c616c3a71233 for ruby:3.1 with digest ruby@sha256:ba4d592a4fc6e3f5d2a9a52a1a3bbefde53308e786de4f66ba72237b18b15676 ...
18 $ echo "Start of '$CI_PIPELINE_ID' pipeline "
19 Start of '1519840356' pipeline
20 $ echo "This is job '$CI_JOB_NAME' in stage '$CI_JOB_STAGE'"
21 This is job 'jobA' in stage 'stage1'
22 Cleaning up project directory and file based variables 00:01
23 Job succeeded
```

Duration: 29 seconds  
Finished: 18 minutes ago  
Queued: 0 seconds  
Timeout: 1h (from project) ?  
Runner: #12270807 (j1aLDqxS) 1-blue.saaS-linux-small-amd64.runners-manager.gitlab.com/default

Commit ab7974dc  
Correct job stages

Pipeline #1519840356 Passed for main  
stage1

# Demo – Job container image

```
jobA:
  stage: stage1
  image: busybox
  script:
    - echo "Start of '$CI_PIPELINE_ID' pipeline "
```

Search visible log output

9

Getting source from Git repository

00:01

10

Fetching changes with git depth set to 20...

11

Initialized empty Git repository in /builds/oconnordarmuid/demo-cicd-pipelines/.git/

12

Created fresh repository.

13

Checking out da7cb4e9 as detached HEAD (ref is main)...

14

Skipping Git submodules setup

15

\$ git remote set-url origin "\${CI\_REPOSITORY\_URL}"

16

Executing "step\_script" stage of the job script

00:01

17

Using docker image sha256:27a71e19c95622dce4d60d4a3760707495c9875f5c5322c5bd535214799593ce for busybox with digest busybox@sha256:768e5c6f5cb6db0794eec98dc7a967f40631746c32232b78a3105fb946f3ab83 ...

18

\$ echo "Start of '\$CI\_PIPELINE\_ID' pipeline "

19

Start of '1519885638' pipeline

20

\$ echo "This is job '\$CI\_JOB\_NAME' in stage '\$CI\_JOB\_STAGE'"

21

This is job 'jobA' in stage 'stage1'

22

Cleaning up project directory and file based variables

00:00

23

Job succeeded

Duration: 10 seconds

Finished: 1 minute ago

Queued: 0 seconds

Timeout: 1h (from project) ?

Runner: #32976687 (nN8vMRS9) 6-blue.saas-linux-small-amd64.runners-manager.gitlab.com/default

Commit da7cb4e9

Use busybox image

Pipeline #1519885638 Passed for main

stage1

# Variables in GitLab.

- Three sources:
  1. Predefined – see [https://docs.gitlab.com/ee/ci/variables/predefined\\_variables](https://docs.gitlab.com/ee/ci/variables/predefined_variables)
  2. Local – defined inside a job.
  3. Pipeline – defined at the pipeline level; available to all jobs
  4. Environment – for sensitive data; available to all jobs

# Variables in GitLab.

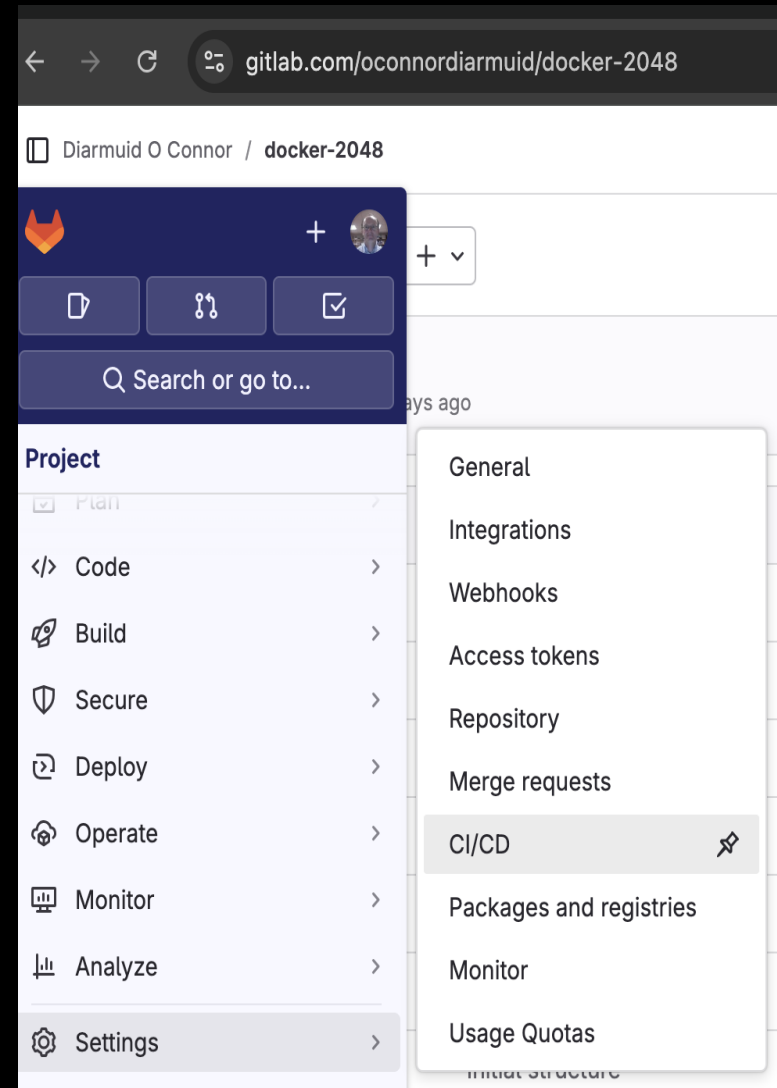
- Pipeline variables..

```
1
2 variables:
3   APP_NAME: user-profile
4   DOCKER_TAG: ${CI_COMMIT_SHORT_SHA}
5   DOCKER_HUB_USERNAME: doconnor
6
7 stages:
8   - stage1
9   - stage2
10  - stage3
11
```

- Reference these variables inside a job using `${var_name}`, e.g. `${DOCKER_HUB_USERNAME}`
- Use uppercase and `_` in names (Convention)

# Variables in GitLab.

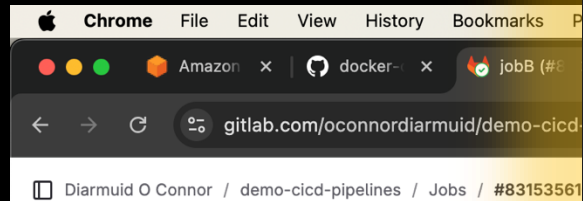
- Environment variables.
- Variables have a key (name) and value.
- Same syntax for referencing inside a job.



# Services.

- A job's script (build script) is executed inside the container whose image is declared with the image keyword.
- An additional container(s) can be created and made available to the build container, using the services keyword.
- The hostname of a service container is based on the service image name, e.g.
  - imageX:2.10-alpine → imageX
  - userA/imageX:2.10-alpine → userA\_\_imageX or userA-imageX

# Services



```
jobB:
  stage: stage2
  image: alpine/curl
  services:
    - nginx:1.15.8-alpine
  script:
    - echo "This job has an additional nginx container"
    - curl http://nginx
```

Search visible log output

cf1798852952

> 3 Preparing the "docker+machine" executor 00:10

> 11 Preparing environment 00:01

> 13 Getting source from Git repository 00:01

> 20 Executing "step\_script" stage of the job script 00:01

21 Using docker image sha256:7c797a007c7fae71f08c5b8e4e1c4ff908166293573f8fb2e72662ee0dc71f97 for alpine/curl with digest alpine/curl@sha256:5adbdde67c2fee2863259fe3eafd44f7a501cbf70ac603936a6241784c9c6188 ...

22 \$ echo "This is job '\$CI\_JOB\_NAME' in stage '\$CI\_JOB\_STAGE'"

23 This is job 'jobB' in stage 'stage2'

24 \$ curl http://nginx

25 % Total % Received % Xferd Average Speed Time Time Time Current

26 Dload Upload Total Spent Left Speed

27 100 612 100 612 0 0 637k 0 --:--:-- --:--:-- --:--:-- 597k

28 <!DOCTYPE html>

29 <html>

30 <head>

31 <title>Welcome to nginx!</title>

32 <style>

33 body {

34 width: 35em;

35 margin: 0 auto;

Duration: 15 seconds

Finished: 20 minutes ago

Queued: 0 seconds

Timeout: 1h (from project)

Runner: #12270837 (J2nyww-s) 4-blue.saas-linux-small-amd64.runners-manager.gitlab.com/default

Commit 9cfcab7

Services array

Pipeline #1534656668 Passed for main

stage2

Related jobs

→ jobB



# Services – Use case.

- The Docker in Docker (DinD) image – contains the Docker daemon/engine/server installation.

```
jobC:
  stage: stage2
  image: docker:27.0.2-cli-alpine3.20
  services:
    - docker:27.0.2-dind-alpine3.20
  script:
    - docker pull doconnor/movies-api:1.0
    - docker images
```

```
90 Digest: sha256:90aebf2ca5179a96f0a23d44aa606856900985b066b7e8ba7abec0c4fd6da020
91 Status: Downloaded newer image for doconnor/movies-api:1.0
92 docker.io/doconnor/movies-api:1.0
93 $ docker images
94 REPOSITORY          TAG          IMAGE ID       CREATED        SIZE
95 doconnor/movies-api  1.0          536d4758124e   2 weeks ago   2.84GB
96 Cleaning up project directory and file based variables
97 Job succeeded
```

00:00

# Job Structure

|               |   |
|---------------|---|
|               | <code>job1:</code>  |
|               | <code>stage: build</code>   |
| Variables     | <code>variables:</code><br><code>  DATABASE_URL: "test"</code>  |
| Before script | <code>before_script:</code><br><code>  - execute-before-script-for-job1</code>  |
| Script        | <code>script:</code><br><code>  - execute-script-for-job1</code><br><code>  - something else</code>                   |
| After Script  | <code>after_script:</code><br><code>  - execute-after-script-for-job1</code>  |
| Artifacts     | <code>artifacts:</code><br><code>  paths:</code><br><code>    - .variables</code><br><code>  expire_in: 1 week</code> |
| Only/ Except  | <code>only:</code><br><code>  - master</code><br><code>except:</code><br><code>  - develop</code>                     |
| Tags          | <code>tags:</code><br><code>  - ruby</code><br><code>  - postgres</code>  |
| When          | <code>when: manual</code>   |

# Job rules

- Replaces the deprecated only/except option.
- Use rules to specify when jobs run.

```
jobB:
  stage: stage2
  image: alpine/curl
  services:
    - nginx:1.15.8-alpine
  script:
    - echo "This job has an additional nginx container"
    - curl http://nginx
  rules:
    - if: $CI_COMMIT_BRANCH == 'develop'
```

# Job rules

- Examples: Run a job only when:
  - The commit is for a certain branch
  - The commit is a merge request
  - The commit tag is a certain value
  - The commit tag matches a pattern
- Can have multiple rules.
  - If any rule is true then the job is executed

# GitLab CI/CD Architecture

- GitLab CI/CD service uses registered Runner nodes to execute pipeline jobs.
- GitLab Runner:
  - An application to run jobs and send results back to GitLab
  - Installed on cloud / on-premise machines.
  - Written in Go and can be run as a single binary.
  - Works with Linux, Windows and OSX.
  - Acts as an agent (worker) for GitLab CI/CD service
- GitLab Executor:
  - A Runner passes a job to an Executor.
  - Executor determines the environment a job runs in .
  - Lots of Executor types:
    - PowerShell, VirtualBox, Docker, Kubernetes, Custom, etc,