

Get started with GitLab CI/CD

Sonny LION
RPW Operation Centre (ROC)





GitLab

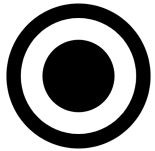
Sign in to a GitLab:

- with your LDAP account: <https://gitlab.obspm.fr/>
- or using your personal gitlab.com account: <https://gitlab.com/>

Demo sources (public repository):

- <https://gitlab.obspm.fr/slion/gitlab-ci-cd-demo>

Development Workflow Summary



Commit

New codes are integrated to the base code



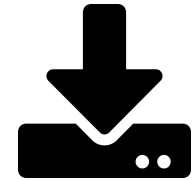
Build

Source code is converted into an executable form



Test

Checks the interaction between builds and if the app is working



Deploy

Deploys the app to production environment



This manual process takes up a huge amount of time and energy, which could have been used for development instead.

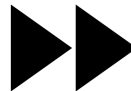
→ **CI/CD** to the rescue !

Continuous integration, delivery and deployment is known collectively as CI-CD

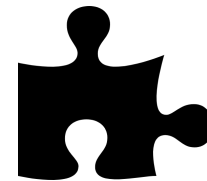
CI-CD essentially involves continuously **building, testing** and **deploying code** changes at every **small iteration**, reducing chance of developing new code based on bugged or failed previous versions.



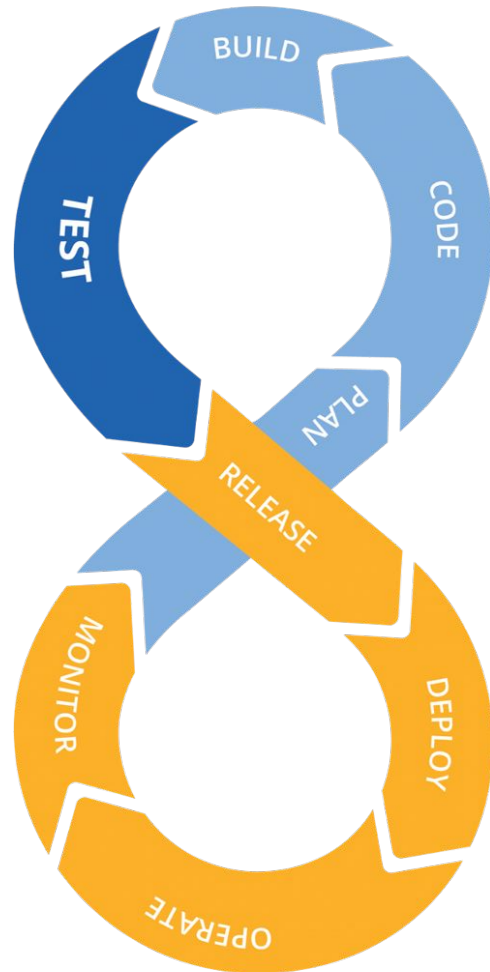
Reduces Errors in
Code



Speeds up Coding
Process



Integrates Code
Seamlessly



Continuous Integration

For **every push** to the repository, you can create a set of scripts to **build and test** your application **automatically**. These scripts help **decrease** the chances that you introduce **errors** in your application.

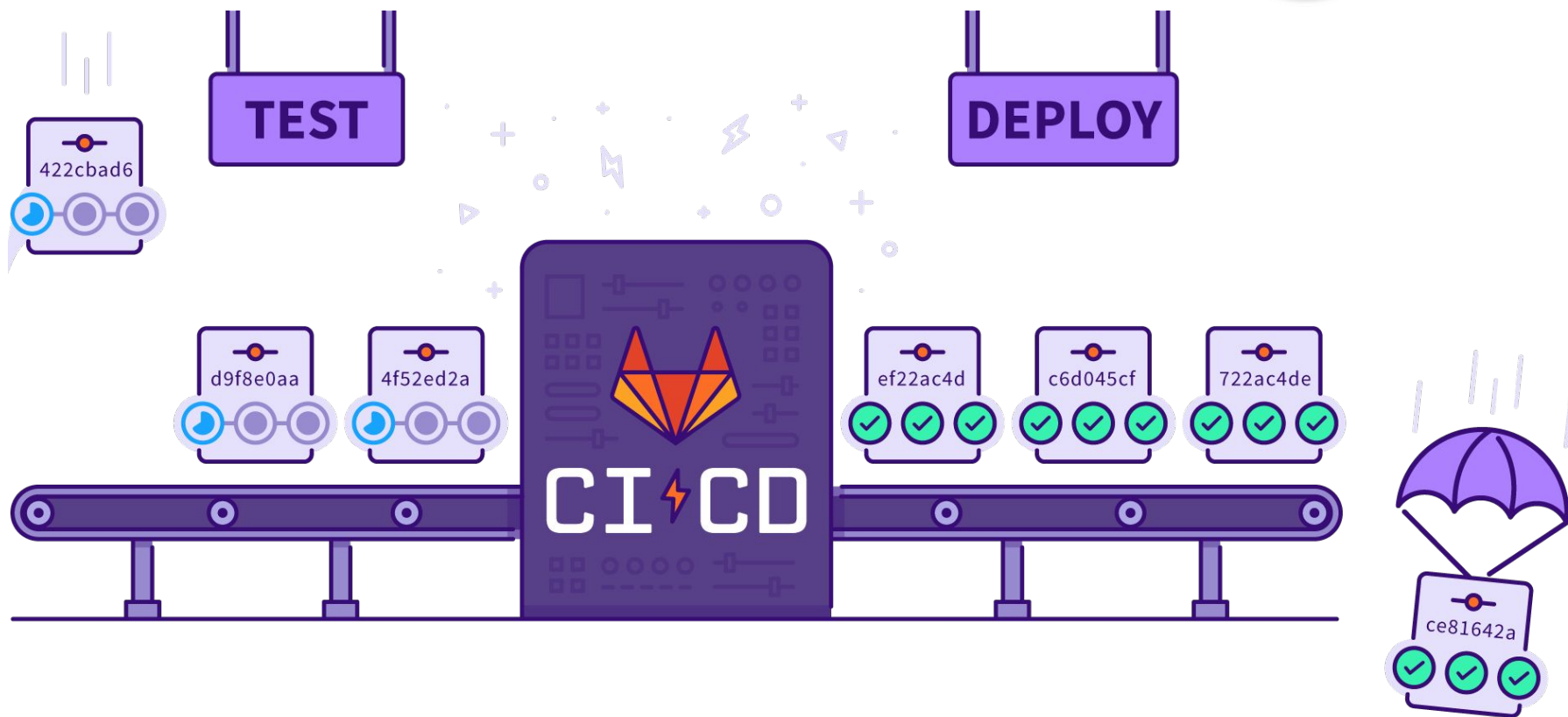
Continuous Delivery

Continuous Delivery is a step beyond Continuous Integration. Not only is your application built and tested each time a code change is pushed to the codebase, the application is also **automatically prepared** for a **release** to production.

Continuous Deployment

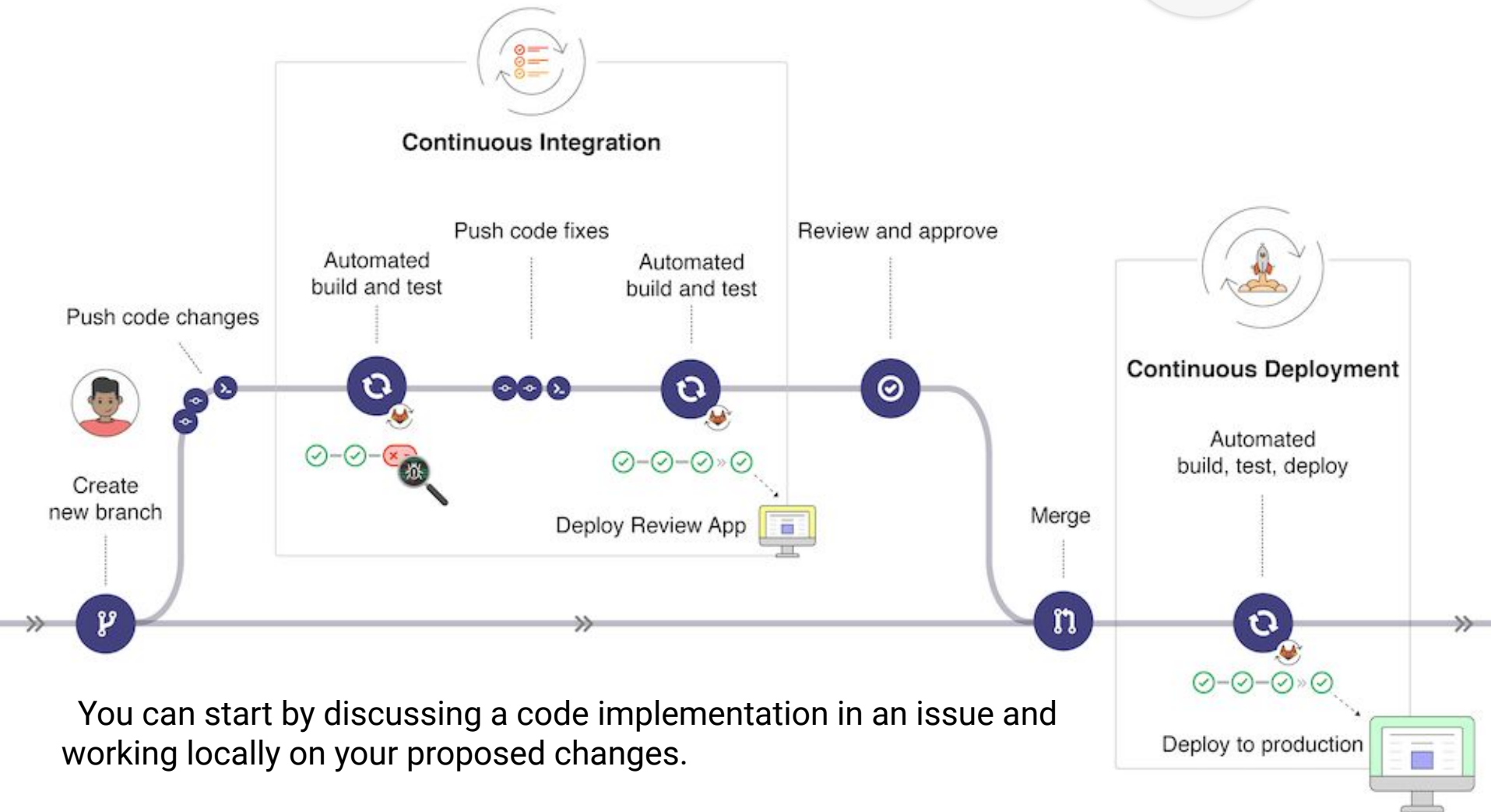
Continuous deployment is like continuous delivery, except that **releases happen automatically**.

GitLab CI-CD ?



GitLab CI/CD is a powerful tool built into Gitlab that allows you to apply Continuous Integration, Continuous Delivery, and Continuous Deployment to your software with **no third-party application** or integration needed. Moreover you can visualize all the steps in the GitLab UI.

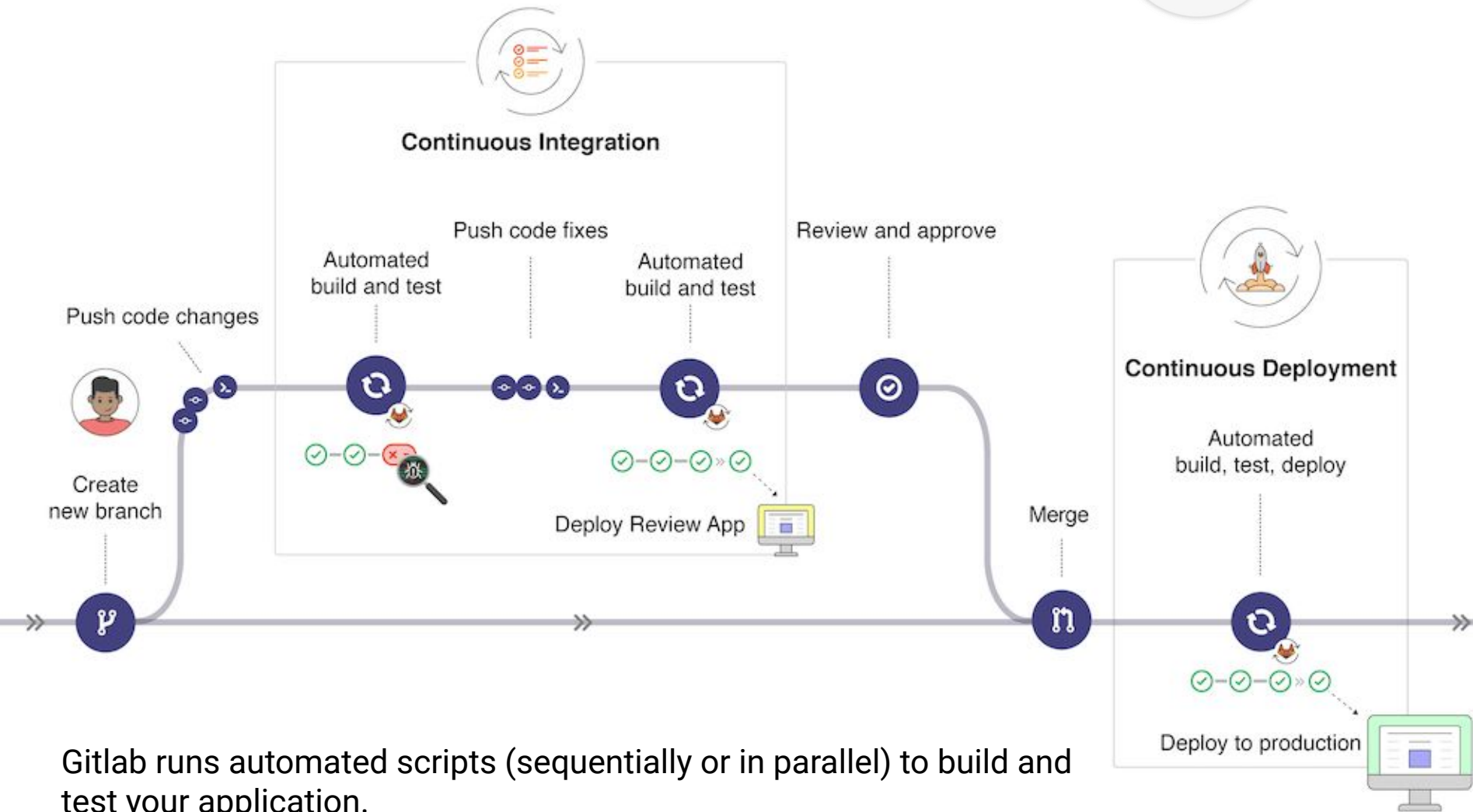
GitLab CI/CD workflow (1/3) - Push code changes



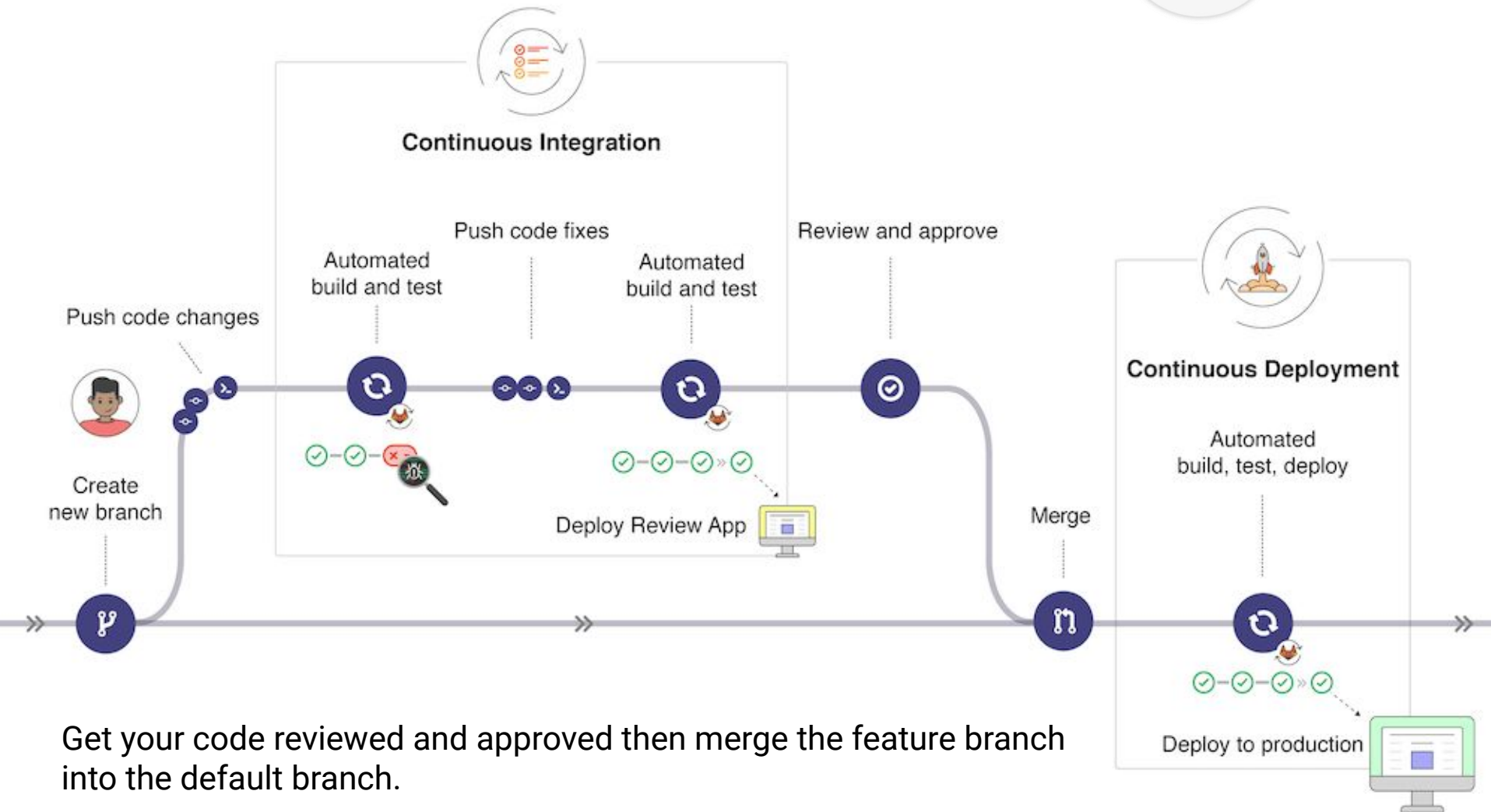
You can start by discussing a code implementation in an issue and working locally on your proposed changes.

Then you can push your commits to a feature branch in a remote repository that's hosted in GitLab. The push triggers the CI/CD pipeline for your project

GitLab CI/CD workflow (2/3) - Continuous Integration



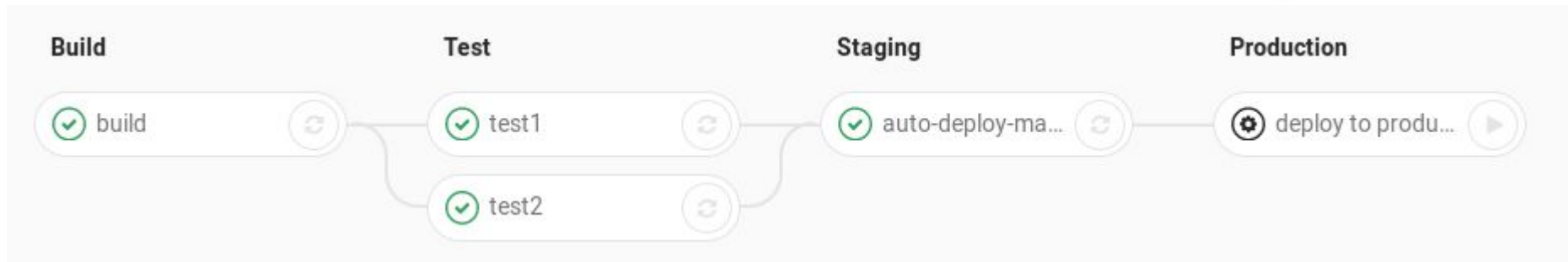
GitLab CI/CD workflow (3/3) - Review, merge and deployment



Get your code reviewed and approved then merge the feature branch into the default branch.

GitLab CI/CD deploys your changes automatically to a production environment.

CI/CD pipelines



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

Pipelines comprise:

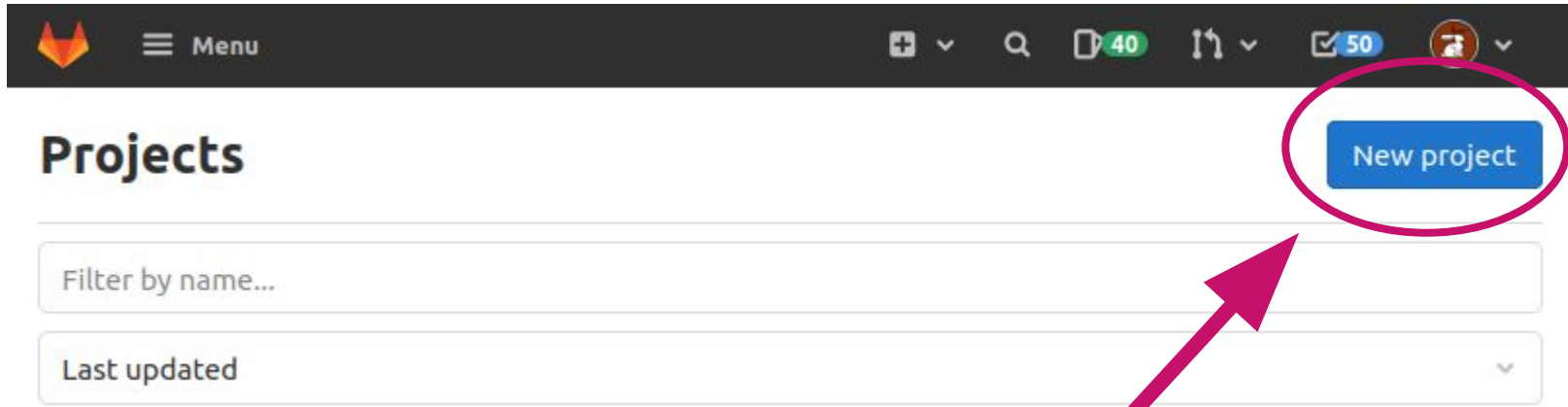
- **Jobs**, which define what to do. For example, jobs that compile or test code.
- **Stages**, which define when to run the jobs. For example, stages that run tests after stages that compile the code.

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.

Get started - New project (1/2)



Create a blank project using the Gitlab UI

Get started - New project (2/2)



Create blank project

Create a blank project to house your files, plan your work, and collaborate on code, among other things.

New project > **Create blank project**

Project name




Project URL

Project slug

Want to house several dependent projects under the same namespace? [Create a group.](#)

Project description (optional)

Visibility Level

- ☒  **Private**
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.
- ☐  **Internal**
The project can be accessed by any logged in user except external users.
- ☐  **Public**
The project can be accessed without any authentication.

☒ **Initialize repository with a README**

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Get started - First CI/CD pipeline with Gitlab



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, including conditional execution.



Optimize your workflow with CI/CD Pipelines

Create a new `.gitlab-ci.yml` file at the root of the repository to get started.

Create new CI/CD pipeline

Step 2

Get started - The YAML file (.gitlab-ci.yml)

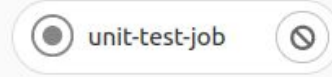
✓ demo/01_simple_example

```
stages:  
  - build  
  - test  
  - deploy
```

Build



Test



Deploy



```
build-job:      # This job runs in the build stage, which runs first.  
  stage: build  
  script:  
    - echo "Compiling the code..."  
    - echo "Compile complete."  
  
unit-test-job:  # This job runs in the test stage.  
  stage: test    # It only starts when the job in the build stage completes successfully.  
  script:  
    - echo "Running unit tests... This will take about 60 seconds."  
    - sleep 60  
    - echo "Code coverage is 90%"  
  
lint-test-job:  # This job also runs in the test stage.  
  stage: test    # It can run at the same time as unit-test-job (in parallel).  
  script:  
    - echo "Linting code... This will take about 10 seconds."  
    - sleep 10  
    - echo "No lint issues found."
```

Demo 01

https://gitlab.obspm.fr/slion/gitlab-ci-cd-demo/-/tree/demo/01_simple_example

The screenshot shows the GitLab web interface for the repository 'gitlab-ci-cd-demo'. The top navigation bar includes the GitLab logo, a menu icon, a search bar, and various status icons. The left sidebar contains navigation icons for repository, merge requests, issues, and other features. The main content area shows the repository path 'Lion Sonny > gitlab-ci-cd-demo > Repository'. A modal titled 'Switch branch/tag' is open, displaying a search bar and a list of branches. The branches listed are 'demo/04_python_with_coverage', 'develop', 'demo/03_postgres_database', 'demo/02_fix_pending_job', 'demo/01_simple_example' (which is selected with a checkmark), and 'master'. The repository name 'gitlab-ci-cd-demo' is also visible at the bottom of the modal.

GitLab Menu

Search GitLab

Lion Sonny > gitlab-ci-cd-demo > Repository

demo/01_simple_... gitlab-ci-cd-demo / +

History Find file Web IDE Clone

Switch branch/tag

Search branches and tags

Branches

- demo/04_python_with_coverage
- develop
- demo/03_postgres_database
- demo/02_fix_pending_job
- ✓ demo/01_simple_example
- master

gitlab-ci-cd-demo

Last update

	16 hours ago
	1 week ago

Pipeline stuck on pending

M

my_project

Project information

Repository

Issues0

Merge requests0

CI/CD

Pipelines

Editor

Jobs

Schedules

Lion Sonny > my_project > Pipelines

All1FinishedBranchesTags

Clear runner cachesCI lintRun pipeline

Filter pipelines

Show Pipeline ID

Status	Pipeline ID	Triggerer	Commit	Stages	Duration
<div>⏸ pending</div>	<div>#11906</div> <div>latest</div> <div>stuck</div>		<div>main ↗ 76ac8de9</div> <div>Update .gitlab-ci.y...</div>	<div>⏸</div> <div>●</div> <div>●</div>	<div>⚠ In progress</div>

⏸ pending

 Job #29139 created 5 minutes ago by Lion Sonny

This job is stuck because the project doesn't have any runners online assigned to it.
Go to project [CI settings](#)

GitLab Runners ?



GitLab Runner is the open source project written in Go that is used to run your CI/CD jobs and send the results back to GitLab

It can be run as a single binary; no language-specific requirements are needed.

You can install GitLab Runner on several different supported operating systems.

Runner registration

After you install the application, you have to register individual runners. When you register a runner, you are setting up communication between your GitLab instance and the machine where GitLab Runner is installed.

GitLab Runner implements a number of executors that can be used to run your builds in different scenarios:

- **Docker** > In a separate and isolated Docker container
- Shell > Locally on the machine where GitLab Runner is installed
- SSH > On a remote machine by executing commands over SSH
- Kubernetes > on a Kubernetes cluster
- etc.

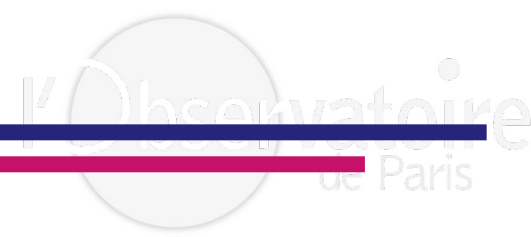
The executors support different platforms and methodologies for building a project.

Examples

If you want your CI/CD job to run PowerShell commands, you might install GitLab Runner on a Windows server and then register a runner that uses the shell executor.

If you want your CI/CD job to run commands in a custom Docker container, you might install GitLab Runner on a Linux server and register a runner that uses the Docker executor.

Runners access and Tags



Who has access to runners in the GitLab UI

There are three types of runners, based on who you want to have access:

- Shared runners are for use by all projects
- Group runners are for all projects and subgroups in a group
- Specific runners are for individual projects

When you register a runner, you specify a token for the GitLab instance, group, or project. This is how the runner knows which projects it's available for.

Tags

When you register a runner, you can add tags to it.

When a CI/CD job runs, it knows which runner to use by looking at the assigned tags.

For example, if a runner has the ruby tag, you would add this code to your project's .gitlab-ci.yml file:

```
my-job:  
  tags:  
    - ruby
```

When the job runs, it uses the runner with the ruby tag

Check available runners



Issues0

Merge requests0

CI/CD

Security & Compliance

Deployments

Monitor

Infrastructure

Packages & Registries

Analytics

Wiki

Snippets

Settings

- General
- Integrations
- Webhooks
- Access Tokens
- Repository
- CI/CD
- Monitor
- Pages
- Packages & Registries

Runners

Collapse

Runners are processes that pick up and execute CI/CD jobs for GitLab. [How do I configure runners?](#)

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine. Runners are either:

- active - Available to run jobs.
- paused - Not available to run jobs.

Specific runners

These runners are specific to this project.

Set up a specific Runner for a project

- [Install GitLab Runner and ensure it's running.](#)
- Register the runner with this URL:
`https://gitlab.obspm.fr/`

And this registration token:
`fsMzeZNTySLkix1Qy8Wj`

Reset registration token

Show Runner installation instructions

Shared runners

These runners are shared across this GitLab instance.

The same shared runner executes code from multiple projects, unless you configure autoscaling with [MaxBuilds](#) set to 1 (which it is on GitLab.com).

Enable shared runners for this project



Available shared runners: 2

#35 (FGmL_Pwr)
docker

`docker_dio`

#18 (4147eff3)
Docker DIO pour DIND

`docker_dio_dind`

Fix the pipeline

```
default:
  image: alpine
  tags:
    - docker_dio

stages:      # List of stages for
              # jobs, and their order of execution
  - build
  - test
  - deploy
```

Available shared runners: 2

● #35 (FGmL_Pwr)

docker

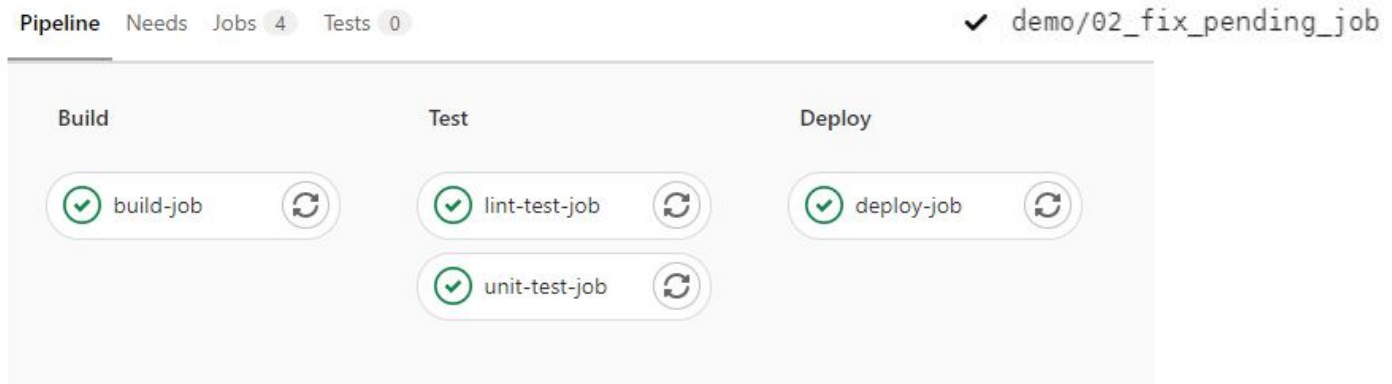
[docker_dio](#)

● #18 (4147eff3)

Docker DIO pour DIND

[docker_dio_dind](#)

Set the default docker image to **alpine** and add the **docker_dio** tag



Demo 02

https://gitlab.obspm.fr/slion/gitlab-ci-cd-demo/-/tree/demo/02_fix_pending_job

GitLab Menu Search GitLab 41 51

Lion Sonny > gitlab-ci-cd-demo > Repository

demo/02_fix_pen... gitlab-ci-cd-demo / +

History Find file Web IDE Clone

Switch branch/tag

Search branches and tags

Branches

- demo/04_python_with_coverage
- develop
- demo/03_postgres_database
- ✓ demo/02_fix_pending_job
- demo/01_simple_example
- master

gitlab-ci-cd-demo

312a3dd9

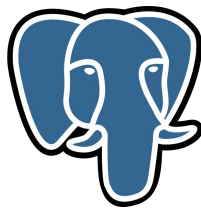
Last update
16 hours ago
1 week ago

Services

The services keyword defines a **Docker image** that runs during a job **linked** to the **Docker image** that the image keyword defines. This allows you to access the service image during build time.

The service image can run any application, but the most common use case is to run a **database container**, for example:

- MySQL
- PostgreSQL
- Redis
- etc.



It's easier and faster to use an existing image and run it as an additional container than to install mysql, for example, every time the project is built.

You're **not limited to only database services**. You can add as many services you need to .gitlab-ci.yml or manually modify config.toml. Any image found at Docker Hub or your private Container Registry can be used as a service.



Services are **not shared between jobs**

Demo 03

https://gitlab.obspm.fr/slion/gitlab-ci-cd-demo/-/tree/demo/03_postgres_database

The screenshot shows the GitLab web interface for the repository 'gitlab-ci-cd-demo' by user 'Lion Sonny'. A modal window titled 'Switch branch/tag' is open, displaying a list of branches. The branch 'demo/03_postgres_database' is selected with a checkmark. Other visible branches include 'demo/04_python_with_coverage', 'develop', 'demo/02_fix_pending_job', 'demo/01_simple_example', and 'master'. The main content area shows the repository's commit history with a table of recent updates.

Last update	
6ad85af5	13 hours ago
	1 week ago
	14 hours ago

Cache and artifacts

Use cache for dependencies, like packages you download from the internet. Subsequent jobs that use the same cache don't have to download the files again, so they execute more quickly. Cache is stored where GitLab Runner is installed.

Artifacts are generated by a job, stored in GitLab, and can be downloaded. Use artifacts to pass intermediate build results between stages.

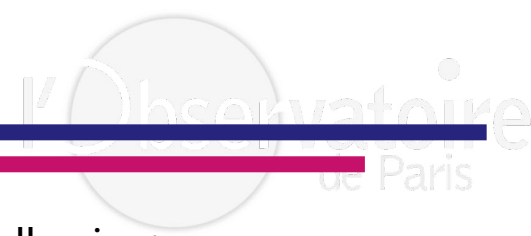
Cache

- Define cache per job by using the cache: keyword. Otherwise it is disabled.
- Subsequent pipelines can use the cache.
- Subsequent jobs in the same pipeline can use the cache, if the dependencies are identical.
- Different projects cannot share the cache.

Artifacts

- Define artifacts per job.
- Subsequent jobs in later stages of the same pipeline can use artifacts.
- Different projects cannot share artifacts.
- Artifacts expire after 30 days by default. You can define a custom expiration time.
- The latest artifacts do not expire if keep latest artifacts is enabled.
- Use dependencies to control which jobs fetch the artifacts.

Good caching practices



To ensure maximum availability of the cache, do one or more of the following:

- Tag your runners and use the tag on jobs that share the cache.
- Use runners that are only available to a particular project.
- Use a key that fits your workflow. For example, you can configure a different cache for each branch.

For runners to work with caches efficiently, you must do one of the following:

- Use a single runner for all your jobs.
- Use multiple runners that have distributed caching, where the cache is stored in S3 buckets. Shared runners on GitLab.com behave this way. These runners can be in autoscale mode, but they don't have to be.
- Use multiple runners with the same architecture and have these runners share a common network-mounted directory to store the cache. This directory should use NFS or something similar. These runners must be in autoscale mode

Demo 04

https://gitlab.obspm.fr/slion/gitlab-ci-cd-demo/-/tree/demo/04_python_with_coverage

The screenshot shows the GitLab web interface for the repository 'gitlab-ci-cd-demo' by user 'Lion Sonny'. A 'Switch branch/tag' modal is open, displaying a list of branches. The 'demo/04_python_with_coverage' branch is selected. The main content area shows a file list with columns for file name, commit message, and last update time.

Repository: gitlab-ci-cd-demo / +

Buttons: History, Find file, Web IDE, Clone

Switch branch/tag modal:

- Search branches and tags
- Branches:
 - ✓ demo/04_python_with_coverage
 - develop
 - demo/03_postgres_database
 - demo/02_fix_pending_job
 - demo/01_simple_example
 - master

	Last update
te gitlab-ci.yml	16 hours ago
te gitlab-ci.yml	16 hours ago
te gitlab-ci.yml	16 hours ago
te gitlab-ci.yml	16 hours ago
.gitlab-ci.yml	Fix regex 13 hours ago
.pre-commit-config.yaml	Add python app and update gitlab-ci.yml 16 hours ago
README.md	Apply pre-commit 1 week ago
poetry.lock	Add python app and update gitlab-ci.yml 16 hours ago
pyproject.toml	Add python app and update gitlab-ci.yml 16 hours ago

File list: README.md

Test and deploy a Python package

We start from a repository containing the sources of a simple python application, some tests, the documentation and some coding conventions to check.

	Name	Last commit	Last update
sources	demo_app	Apply pre-commit	1 week ago
docs	docs	Apply pre-commit	1 week ago
tests	tests	Apply pre-commit	1 week ago
	.gitignore	Apply pre-commit	1 week ago
	.gitlab-ci.yml	Add junit report	1 week ago
guidelines	.pre-commit-config.yaml	Add python src code, gitlab-ci an...	1 week ago
	README.md	Apply pre-commit	1 week ago
requirements	poetry.lock	Apply pre-commit	1 week ago
	pyproject.toml	Apply pre-commit	1 week ago

Registries

Package Registry



The GitLab Package Registry acts as a private or public registry for a variety of common package managers (npm, PyPI, Ruby gems, etc.). You can publish and share packages, which can be easily consumed as a dependency in downstream projects.

Container Registry



docker

The GitLab Container Registry is a secure and private registry for container images. It's built on open source software and completely integrated within GitLab. Use GitLab CI/CD to create and publish images. Use the GitLab API to manage the registry across groups and projects.



Check if the package registry feature is enabled

- Project information
- Repository
- Issues 0
- Merge requests 0
- CI/CD
- Security & Compliance
- Deployments
- Monitor
- Infrastructure
- Packages & Registries
- Analytics
- Wiki
- Snippets
- Settings
- General**
- Integrations
- Webhooks
- Access Tokens
- Repository
- CI/CD
- Monitor
- Pages
- Packages & Registries

« Collapse sidebar

Visibility, project features, permissions

Collapse

Choose visibility level, enable/disable project features and their permissions, disable email notifications, and show default award emoji.

Project visibility ?

Public

The project can be accessed by anyone, regardless of authentication.

☒ Users can request access

Issues ?

Flexible tool to collaboratively develop ideas and plan work in this project.

☒ Everyone With Access

Repository

View and edit files in this project. Non-project members will only have read access.

☒ Everyone With Access

Merge requests

Submit changes to be merged upstream.

☒ Everyone With Access

Forks

Users can copy the repository to a new project.

☒ Everyone With Access

Git Large File Storage (LFS) ?

Manages large files such as audio, video, and graphics files.

☒

Packages ?

Every project can have its own space to store its packages.

☒

Artifacts, test reports and coverage

```
# test the code and display coverage
pytest:
  stage: test
  tags:
    - docker_dio
  before_script:
    - python --version
    - pip install --upgrade pip
    - pip install poetry
    - poetry config virtualenvs.in-project true --local
    - poetry install
  coverage: '^TOTAL.+?(\\d+\\%)$'
  script:
    - poetry run pytest --cov=demo_app
    - poetry run coverage xml
  artifacts:
    reports:
      junit: report.xml
      cobertura: coverage.xml
```

Coverage report · coverage 80.00%

00:01:15

1 hour ago

Download artifacts

Download pytest:junit artifact

Download pytest:cobertura artifact

Pipeline Needs Jobs 2 Tests 1

Summary

1 tests 0 failures 0 errors 100% success rate 3.00ms

Jobs

Job	Duration	Failed	Errors	Skipped	Passed	Total
pytest	3.00ms	0	0	0	1	1

Test coverage visualization

Once configured, if you create a merge request that triggers a pipeline which collects coverage reports, the coverage is shown in the diff view. This includes reports from any job in any stage in the pipeline.

The coverage displays for each line:

- covered (green)
- no test coverage (orange)
- no coverage information

The screenshot shows a GitLab merge request interface. On the left, a file explorer lists files with their coverage status: `demo_app/` (0 lines covered, 0 lines not covered), `demo_app/__init__.py` (0 lines covered, 0 lines not covered), `demo_app/app.py` (11 lines covered, 0 lines not covered), `docs/` (1 line covered, 0 lines not covered), `.gitignore` (1 line covered, 0 lines not covered), `Makefile` (20 lines covered, 0 lines not covered), `conf.py` (82 lines covered, 0 lines not covered), `index.rst` (13 lines covered, 0 lines not covered), `make.bat` (35 lines covered, 0 lines not covered), and `requirements.txt` (3 lines covered, 0 lines not covered). The main area shows the diff view for `demo_app/app.py`, which has 11 lines covered (green) and 0 lines not covered (orange). The code in the diff view is as follows:

```
1 + # -*- coding: utf-8 -*-
2 + import sys
3 +
4 +
5 + def print_python_version():
6 +     """Print the Python version
7 +     """
8 +     print(f"{sys.version}")
9 +
10 + if __name__ == '__main__':
11 +     print_python_version()
```


Tag-triggered deployment

You can also use GitLab CI/CD to build and publish packages

```
# publish python package on gitlab registry
publish:
  stage: deploy
  tags:
    - docker_dio
  before_script:
    - python --version
    - pip install --upgrade pip
    - pip install poetry

  script:
    # publish on a dedicated and centralized repository
    - poetry config repositories.gitlab
      ${CI_API_V4_URL}/projects/${CI_PROJECT_ID}/packages/pypi
    - poetry build
    - poetry publish -r gitlab -u gitlab-ci-token -p $CI_JOB_TOKEN

  only:
    - tags
    - web
```

The screenshot shows the GitLab web interface. The top navigation bar includes the GitLab logo, a menu icon, a search bar, and user profile information. The left sidebar contains a list of project features: Project information, Repository, Issues (0), Merge requests (1), CI/CD, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Package Registry (selected), Container Registry, and Infrastructure Registry. The main content area is titled 'Package Registry' and shows '1 Package'. Below this, it says 'Publish and share packages for a variety of common package managers. [More information](#)'. A search bar with 'Filter results' and a search icon is present. Below the search bar, a package named 'demo-app' is listed. It has a version '0.1.0' published by 'Lion Sonny' using 'PyPI'. To the right of the package name, it shows '2.0.0' and a commit hash '39ca8712'. Below this, it says 'Created 2 minutes ago' and there is a trash icon.

Exploring GitLab Pages

With GitLab Pages, you can publish static websites directly from a repository in GitLab.

The screenshot shows the GitLab web interface. The top navigation bar includes the GitLab logo, a menu icon, a search bar, and several status icons. The left sidebar contains a list of project settings: Project information, Repository, Issues (0), Merge requests (0), CI/CD, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, Snippets, and Settings. The 'Settings' menu is expanded, showing sub-options: General, Integrations, Webhooks, Access Tokens, Repository, CI/CD, Monitor, and Pages. The 'Pages' option is selected and highlighted. The main content area shows the 'Pages' settings for the project 'gitlab-ci-cd-demo'. It includes a search bar for settings, a 'Pages' section with a description and a 'Learn more' link, an 'Access pages' section showing the URL 'https://slion.pages.obspm.fr/gitlab-ci-cd-demo', a 'Domains' section with a message that domain support is disabled, and a 'Remove pages' section with a red header and a 'Remove pages' button.

GitLab Menu

Search GitLab

g gitlab-ci-cd-demo

Project information

Repository

Issues 0

Merge requests 0

CI/CD

Security & Compliance

Deployments

Monitor

Infrastructure

Packages & Registries

Analytics

Wiki

Snippets

Settings

General

Integrations

Webhooks

Access Tokens

Repository

CI/CD

Monitor

Pages

Lion Sonny > gitlab-ci-cd-demo > Pages

Search settings

Pages

With GitLab Pages you can host your static website directly from your GitLab repository. [Learn more.](#)

Access pages

Your pages are served under:

<https://slion.pages.obspm.fr/gitlab-ci-cd-demo>

Domains

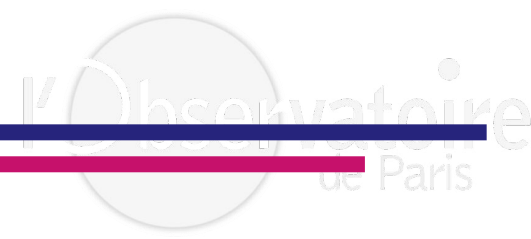
Support for domains and certificates is disabled. Ask your system's administrator to enable it.

Remove pages

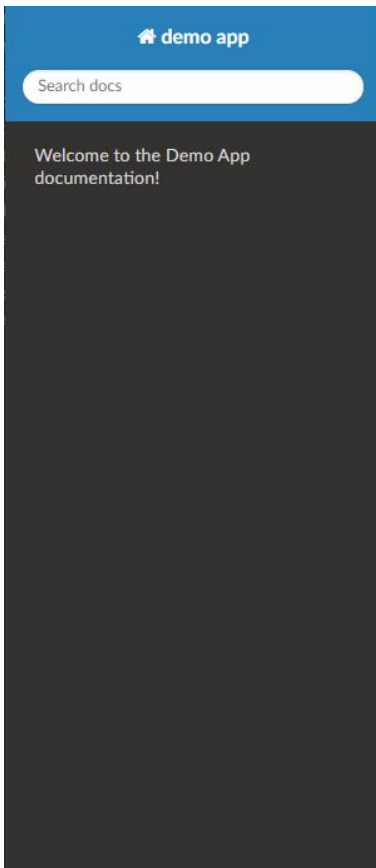
Removing pages will prevent them from being exposed to the outside world.

Remove pages

Generate the documentation



A specific job called pages in the configuration file makes GitLab aware that you're deploying a GitLab Pages website.



» Welcome to the Demo App documentation!
[View page source](#)

Welcome to the Demo App documentation!

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus lacus quam, accumsan at orci vel, varius venenatis arcu. Mauris suscipit euismod dui, eu molestie lectus pharetra ut. Suspendisse et erat vehicula purus iaculis aliquet. Quisque ultrices iaculis lobortis. Sed cursus, turpis ut lacinia iaculis, lorem libero interdum dolor, non dignissim arcu nunc ac massa. Fusce eu semper mauris. Nullam sollicitudin at nulla a imperdiet. Cras iaculis nibh vitae ex dignissim rhoncus. Ut metus urna, condimentum in dolor quis, elementum scelerisque augue. Fusce sollicitudin dolor non cursus malesuada. Quisque faucibus massa metus, sit amet pretium lacus pharetra et. Proin at sapien pretium, venenatis sem quis, lobortis elit. Maecenas ut semper felis, a posuere nunc. Nam rutrum sagittis massa, molestie egestas ex tincidunt euismod. Cras lectus felis, vulputate vitae eros ac, feugiat malesuada felis.

GitLab always deploys your website from a very specific folder called **public**

```
# deploy doc pages
pages:
  stage: deploy
  tags:
    - docker_dio
  script:
    - pip install -r docs/requirements.txt
    - sphinx-build -b html docs public
  artifacts:
    paths:
      - public
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

Using **rules** we specify that the website should be deployed only from the default branch

Pipeline schedules

Pipelines are normally run based on certain conditions being met. For example, when a branch is pushed to repository.

Pipeline schedules can be used to also **run pipelines at specific intervals**. For example:

- Every month on the 22nd for a certain branch.
- Once every day.

In addition to using the GitLab UI, pipeline schedules can be maintained using the Pipeline schedules API.

Schedule timing is configured with **cron notation**

```
# * * * * * command to execute
```

```
# | | | | |
# | | | | |
# | | | | |
# | | | | |
# | | | | | day of week (0 - 7)
# | | | | | month (1 - 12)
# | | | | | day of month (1 - 31)
# | | | | | hour (0 - 23)
# | | | | | min (0 - 59)
```



Triggering pipelines through the API



Triggers can be used to force a pipeline rerun of a specific *ref* (branch or tag) with an API call.

Adding a new trigger

Go to your **Settings > CI/CD** under **Triggers** to add a new trigger. The **Add trigger** button creates a new token which you can then use to trigger a rerun of this particular project's pipeline.

Every new trigger you create, gets assigned a different token which you can then use inside your scripts or `.gitlab-ci.yml`. You also have a nice overview of the time the triggers were last used.

Manage your project's triggers

Description

Add trigger

Token	Description	Owner	Last used	
e0d577983a539433d4ba1cc57f4650	other_project		Never	

In the following examples, you can see the exact API call you need to make in order to rebuild a specific *ref* (branch or tag) with a trigger token.

Revoking a trigger

You can revoke a trigger any time by going at your project's **Settings > CI/CD** under **Triggers** and hitting the **Revoke** button. The action is irreversible.

Trigger a pipeline using cURL



Passing plain text tokens in public projects is a security issue. Potential attackers can impersonate the user that exposed their trigger token publicly in their `.gitlab-ci.yml` file. Use CI/CD variables to protect trigger tokens.

To trigger a pipeline you need to send a **POST** request to the **GitLab API endpoint**:

```
POST /projects/:id/trigger/pipeline
```

The required parameters are the **trigger's token** and the Git **ref** on which the trigger is performed. Valid refs are branches or tags. The **:id** of a project can be found by querying the API or by visiting the CI/CD settings page which provides self-explanatory examples.

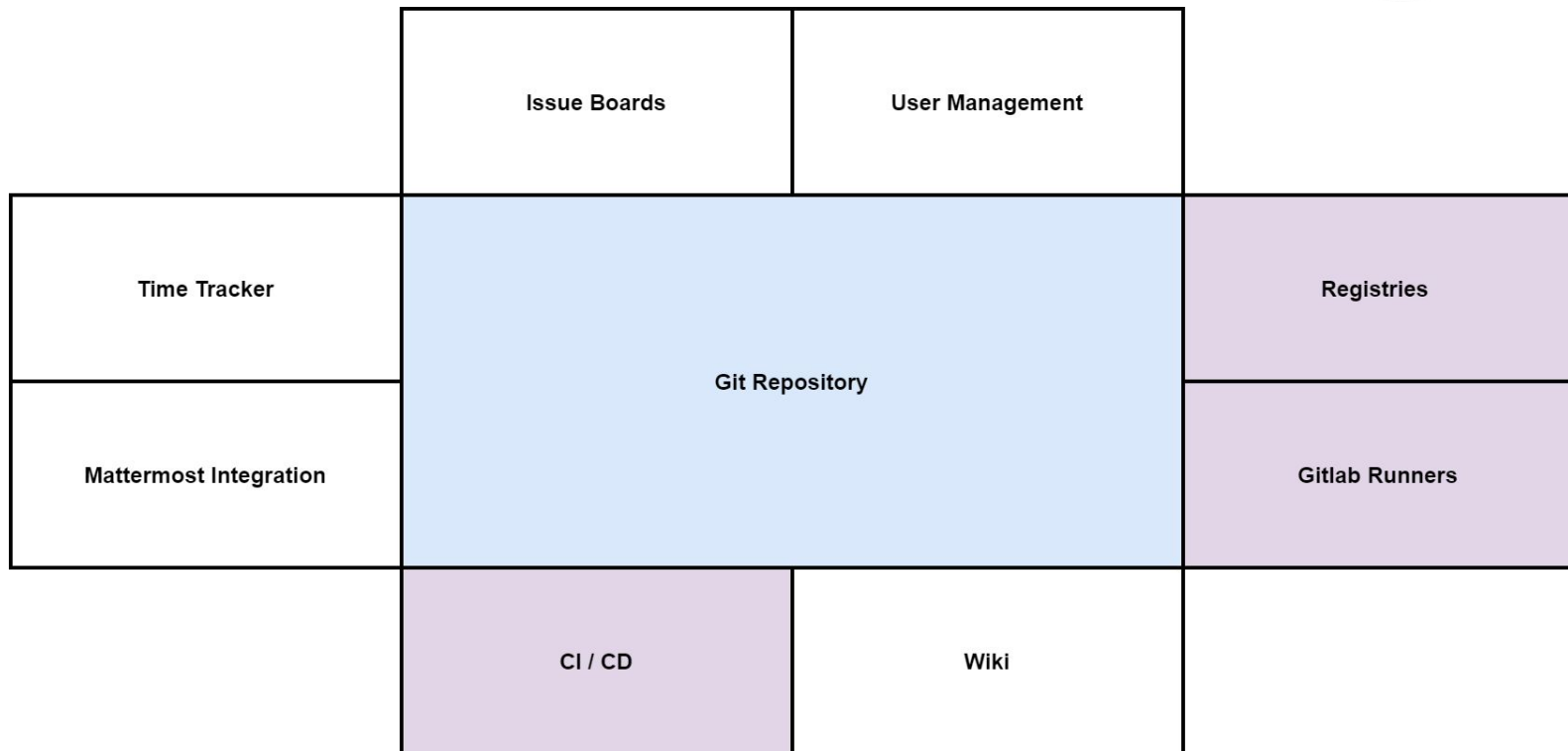
By using **cURL** you can trigger a pipeline rerun with minimal effort, for example:

```
curl -X POST \  
  -F token=<my-token> \  
  -F "ref=<ref-name-like-branch-or-tag>" \  
  https://gitlab.obspm.fr/api/v4/projects/<project-id>/trigger/pipeline
```

Alternatively, you can pass the token and ref arguments in the query string:

```
curl --request POST \  
  "https://gitlab.example.com/api/v4/projects/<project-id>/trigger/pipeline?token=TOKEN&ref=main"
```

This is just the beginning...



Gitlab comes with a lot of built-in features and you still have a lot to discover...

➔ <https://docs.gitlab.com/>