

# Continuous Integration

## (Oltre Lo) Sviluppo Software

Stefano Antonelli

Diego Michelotto

28/04/2021

---

# Summary

---

- CI/CD in generale
- CI/CD in GitLab
  - Hands-on GitLab CI

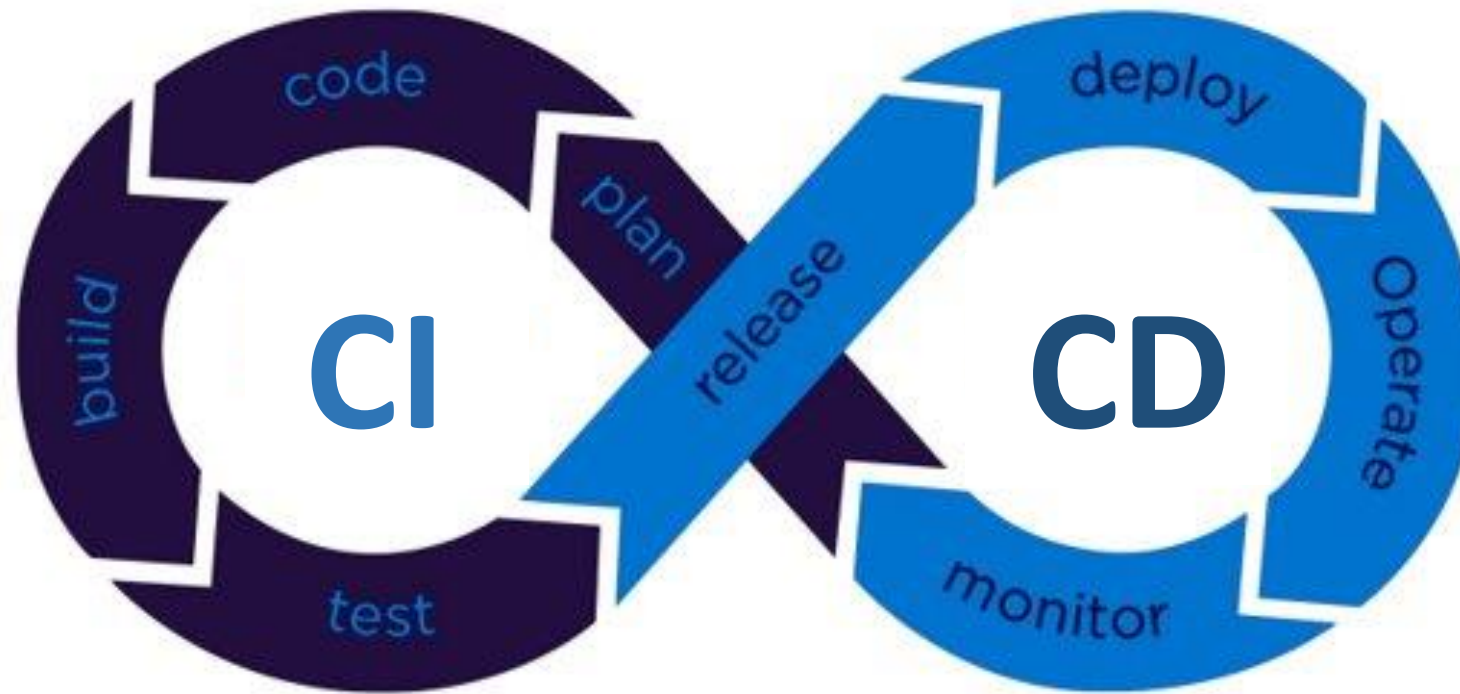
---

CI/CD

---

- La CI/CD (continuous integration/continuous delivery) sono i passaggi che devono essere eseguiti per fornire una nuova versione software.
- Le pipeline di CI/CD sono procedure pensate per ottimizzare l'erogazione di software attraverso l'approccio DevOps o Site Reliability Engineering (SRE).
- Il flusso CI/CD introduce il monitoraggio e l'automazione per ottimizzare il processo di sviluppo delle applicazioni
  - fasi di integrazione
  - test
  - distribuzione
  - deployment
- Il vantaggio principale della CI/CD sta nell'automazione delle procedure

- Una pipeline CI/CD si suddivide in sottoinsiemi distinti di attività. Le fasi di pipeline tipiche includono:
  - **Build:** la fase di compilazione dell'applicazione.
  - **Test:** la fase in cui il codice viene testato. Qui l'automazione può far risparmiare tempo e fatica.
  - **Rilascio:** la fase in cui l'applicazione viene fornita al repository.
  - **Deployment:** in questa fase il software viene distribuito in produzione.
  - **Convalida e conformità:** i passaggi per convalidare una build dipendono dalle esigenze dell'organizzazione. Gli strumenti di scansione per la sicurezza del software confrontandolo con vulnerabilità note (CVE).



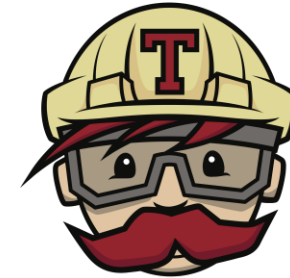
- Una pipeline CI/CD si suddivide in sottoinsiemi distinti di attività. Le fasi di pipeline tipiche includono:
  - **Build:** la fase di compilazione dell'applicazione.
  - **Test:** la fase in cui il codice viene testato. Qui l'automazione può far risparmiare tempo e fatica. I test possono essere di diverso tipo:
    - Unit test: test unità elementari del software
    - Integration test: test di interazione tra più elementari del software
    - Functional test: test che dato un input il Software (Backbox) dia l'output atteso
  - **Rilascio:** la fase in cui l'applicazione viene fornita al repository.
  - **Deployment:** in questa fase il software viene distribuito in staging.
  - **Test:** la fase in cui il software viene testato in esecuzione.
    - Deployment test: test di installazione del software, per esempio su piattaforme differenti
    - Load/Stress test: test di carico del software
  - **Convalida e conformità:**
    - Acceptance test: convalida dei requisiti
    - Security test: confronto con vulnerabilità note (CVE).
  - **Deployment:** il software viene distribuito in produzione
    - Installazione in preproduzione
    - Smoke test
    - Installazione in produzione

- Una pipeline CI/CD si suddivide in sottoinsiemi distinti di attività. Le fasi di pipeline tipiche includono:
  - **Build:** la fase di compilazione dell'applicazione.
  - **Test:** la fase in cui il codice viene testato. Qui l'automazione può far risparmiare tempo e fatica. I test possono essere di diverso tipo:
    - Unit test: test unità elementari del software
    - Integration test: test di interazione tra più elementari del software
    - Functional test: test che dato un input il Software (Backbox) dia l'output atteso
  - **Rilascio:** la fase in cui l'applicazione viene fornita al repository.
  - **Deployment:** in questa fase il software viene distribuito in staging.
    - Deployment test: test di installazione del software, per esempio su piattaforme differenti
    - Load/Stress test: test di carico del software
  - **Convalida e conformità:**
    - Acceptance test: convalida dei requisiti
    - Security test: confronto con vulnerabilità note (CVE).
  - **Deployment:** il software viene distribuito in produzione
    - Installazione in preproduzione
    - Smoke test
    - Installazione in produzione

## La situazione si può complicare ulteriormente



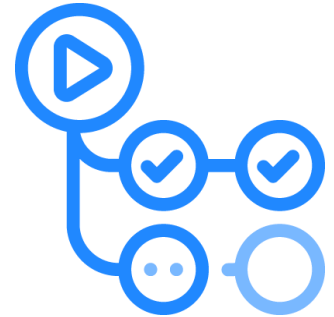
- **GitLab CI**
- **GitHub Actions**
- Jenkins
- Travis CI
- Bamboo/Bitbucket Pipelines (Atlassian)
- ...



Travis CI



**Jenkins**



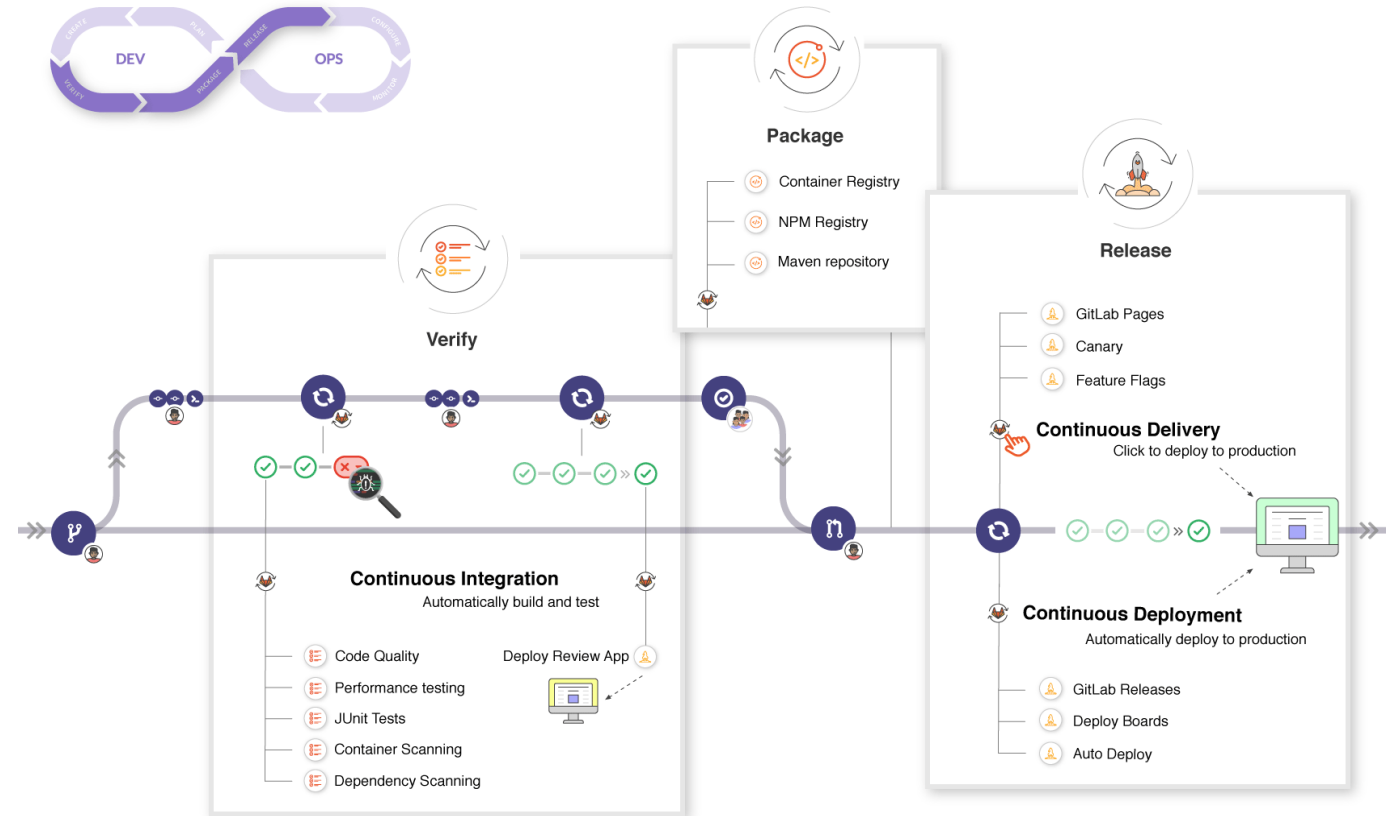
---

# CI/CD in GitLab

---

- La CI/CD è integrata in GitLab e consente lo sviluppo software secondo le metodologie
  - Continuous integration CI  
Uno sviluppatore ha il suo repository in GitLab ed ad ogni "push" sul repository, eventualmente anche per i branch di sviluppo, parte una serie di scripts che compilano e testano l'applicazione
  - Continuous delivery CD
    - Un passo ulteriore rispetto alla CI; l'applicazione viene rilasciata per il repository
  - Continuous deployment CD
    - Il rilascio dell'applicazione viene messo in produzione automaticamente senza intervento manuale

[https://docs.gitlab.com/ee/ci/quick\\_start/index.html](https://docs.gitlab.com/ee/ci/quick_start/index.html)



# CI/CD in GitLab

Cosa serve? Un repository ed un file.

La CI/CD è configurata tramite un file `.gitlab-ci.yml` presente nella *root* del repository. Ad un *push* sul repository, il file esegue una *pipeline*, un set di istruzioni che eseguono dei *jobs* eseguiti su dei *runner*

Il file `.gitlab-ci.yml` è scritto in *YAML*

- <https://en.wikipedia.org/wiki/YAML>
- <https://yaml.org/>

Se utilizzo CI/CD alternative (e.g. Jenkins, Drone) è consigliabile disabilitare la CI in GitLab per evitare conflitti

The screenshot shows the GitLab repository interface for 'Stefano Antonelli > mutt\_config\_files > Repository'. The left sidebar shows the repository structure with 'Repository' selected. The main content area shows the commit history for the `.gitlab-ci.yml` file. A table lists the commits:

Name	Last commit
<code>.gitlab-ci.yml</code>	Update .gitlab-ci.yml
<code>muttrc</code>	aggiornato
<code>muttrc-cnaf</code>	disabilitata CI/CD in Settings
<code>muttrc-presid</code>	aggiornato

Below the table, the 'Edit file' interface for `.gitlab-ci.yml` is shown, displaying the YAML configuration:

```
1 # This file is a template, and might need editing before it works on your project.
2 # see https://docs.gitlab.com/ee/ci/yaml/README.html for all available options
3
4 # you can delete this line if you're not using Docker
5 # image: busybox:latest
6
7 before_script:
8   - echo "Before script section"
9   - echo "For example you might run an update here or install a build dependency"
10  - echo "Or perhaps you might print out some debugging details"
11
12 after_script:
13   - echo "After script section"
14   - echo "For example you might do some cleanup here"
15
16 build:
17   stage: build
18   script:
19     - echo "Do your build here"
20
21 test1:
22   stage: test
23   script:
24     - echo "Do a test here"
25     - echo "For example run a test suite"
26
27 test2:
28   stage: test
```

...e controllare la CI/CD dalla GUI di GitLab

The screenshot shows the GitLab navigation menu with 'CI/CD' highlighted. Other options include 'Issues', 'Merge requests', 'Pipelines', 'Editor', 'Jobs', 'Schedules', and 'Security & Compliance'.

Status	Pipeline	Triggerer	Commit	Stages	Duration
passed	#44662 latest	🌐	👤 master ~ 41a36ca7 📄 Update .gitlab-ci-y...	🟢🟢🟢	🕒 00:03:19 📅 3 weeks ago
passed	#44659	🌐	👤 master ~ 0869affd 📄 test commit e CI/...	🟢🟢🟢	🕒 00:01:36 📅 3 weeks ago
passed	#44568	🌐	👤 master ~ 1074284e 📄 Add new file	🟢🟢🟢	🕒 00:01:35 📅 3 weeks ago

# CI/CD in GitLab

## File .gitlab-ci.yml

Esistono diversi template di .gitlab-ci.yml disponibili dalla GUI

<https://gitlab.com/gitlab-org/gitlab-foss/tree/master/lib/gitlab/ci/templates>



Posso editare il file dall'editor di GitLab

URL per keywords per pipeline (e.g. uno stage è def. dalla keyword stage)

<https://docs.gitlab.com/ee/ci/yaml/README.html#variables>

Bash.gitlab-ci.yml	Add latest changes from gitlab...	6 months ago
C++.gitlab-ci.yml	Add yaml lint	1 year ago
Chef.gitlab-ci.yml	Add latest changes from gitlab...	1 month ago
Clojure.gitlab-ci.yml	Add latest changes from gitlab...	6 months ago
Code-Quality.gitlab-ci...	Refactor Code Quality templat...	1 year ago
Composer.gitlab-ci.yml	Add latest changes from gitlab...	9 months ago
Crystal.gitlab-ci.yml	Add latest changes from gitlab...	6 months ago
Dart.gitlab-ci.yml	Add latest changes from gitlab...	9 months ago
Deploy-ECS.gitlab-ci.y...	Add latest changes from gitlab...	9 months ago
Django.gitlab-ci.yml	Add latest changes from gitlab...	6 months ago
Docker.gitlab-ci.yml	Add latest changes from gitlab...	4 weeks ago
Elixir.gitlab-ci.yml	Add latest changes from gitlab...	6 months ago
Flutter.gitlab-ci.yml	Add latest changes from gitlab...	4 months ago
Go.gitlab-ci.yml	Add yaml lint	1 year ago
Gradle.gitlab-ci.yml	Tidy up CI templates	2 years ago
Grails.gitlab-ci.yml	Add yaml lint	1 year ago
Helix-Wvrtl nrlab-ci vml	Add latest changes from gitlab...	1 week ago
dd latest changes from gitlab...		1 week ago
dd latest changes from gitlab...		1 month ago
dd latest changes from gitlab...		1 month ago

Write pipeline configuration Visualize Lint View merged YAML

```
1 # This file is a template, and might need editing before it works on your project.
2 # see https://docs.gitlab.com/ee/ci/yaml/README.html for all available options
3
4 # you can delete this line if you're not using Docker
5 # image: busybox:latest
6
7 before_script:
8   - echo "Before script section"
9   - echo "For example you might run an update here or install a build dependency"
10  - echo "Or perhaps you might print out some debugging details"
11
12 after_script:
13   - echo "After script section"
14   - echo "For example you might do some cleanup here"
15
16 build:
17   stage: build
18   script:
19     - echo "Do your build here"
20
21 test1:
22   stage: test
23   script:
24     - echo "Do a test here"
25     - echo "For example run a test suite"
26
27 test2:
28   stage: test
```

```
1 # This file is a template, and might need editing before it works on your project.
2 # see https://docs.gitlab.com/ee/ci/yaml/README.html for all available options
3
4 # you can delete this line if you're not using Docker
5 # image: busybox:latest
6
7 before_script:
8   - echo "Before script section"
9   - echo "For example you might run an update here or install a build dependency"
10  - echo "Or perhaps you might print out some debugging details"
11
12 after_script:
13   - echo "After script section"
14   - echo "For example you might do some cleanup here"
15
16 build:
17   stage: build
18   script:
19     - echo "Do your build here"
20
21 test1:
22   stage: test
23   script:
24     - echo "Do a test here"
25     - echo "For example run a test suite"
26
27 test2:
28   stage: test
29   script:
30     - echo "Do another parallel test here"
31     - echo "For example run a lint test"
32
33 deploy:
34   stage: deploy
35   script:
36     - echo "Do your deploy here"
```

Editare la pipeline

Visualizzare le pipeline

Verificare la sintassi

Pipeline: è il componente di livello più alto della CI/CD. Una pipeline comprende

- *Jobs* cosa viene fatto
- *Stages* quando viene fatto

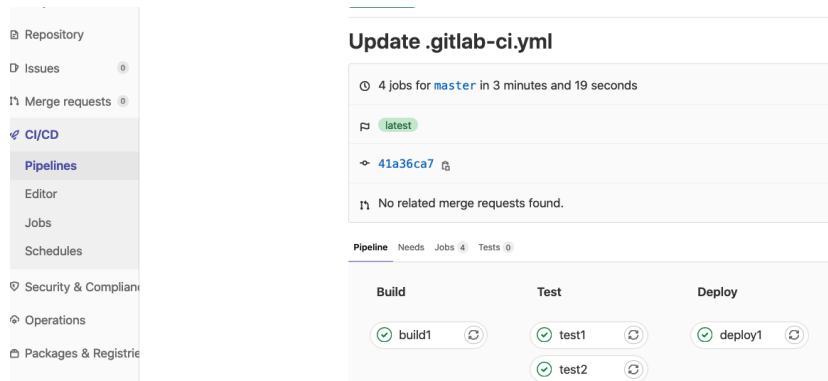
In generale parte in automatico in seguito ad un push ma può essere eseguita anche manualmente

URL che mostrano i tipi di pipeline configurabili

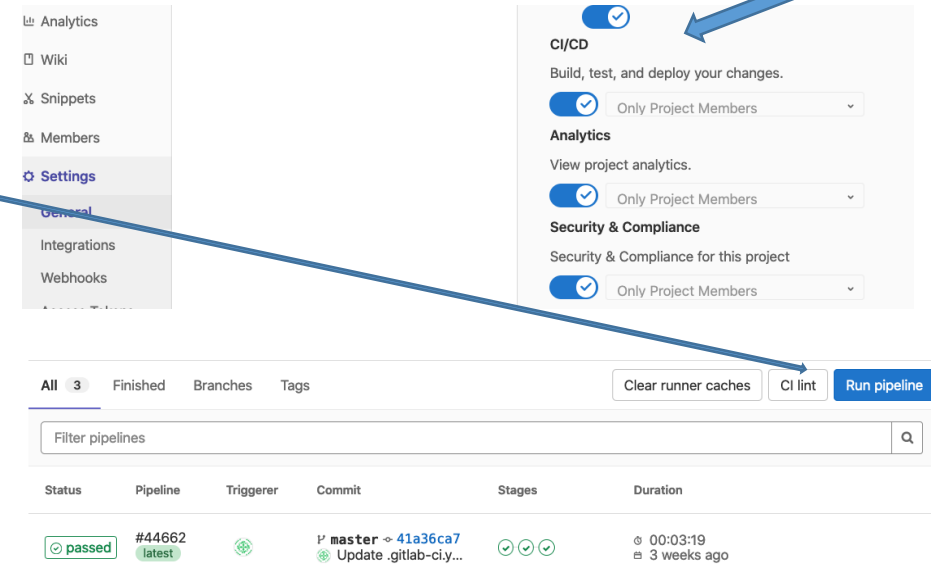
<https://docs.gitlab.com/ee/ci/pipelines/index.html#types-of-pipelines>

[https://docs.gitlab.com/ee/ci/pipelines/pipeline\\_architectures.html](https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html)

Dalla GUI posso controllare lo stato delle *pipeline*



è abilitata di default ma posso disabilitarla



# CI/CD in GitLab

Pipeline: è il componente di livello più alto della CI/CD.

- Una pipeline semplice
- Una pipeline complessa
  - <https://gitlab.com/gitlab-org/gitlab/blob/master/.gitlab-ci.yml>

```
1 stages:
2   - sync
3   - prepare
4   - build-images
5   - fixtures
6   - test
7   - post-test
8   - review-prepare
9   - review
10  - dast
11  - qa
12  - post-qa
13  - pages
14  - notify
15
```

```
53 variables:
54   RAILS_ENV: "test"
55   NODE_ENV: "test"
56   # we override the max_old_space_size to prevent OOM errors
57   NODE_OPTIONS: "--max_old_space_size=3584"
58   SIMPLECOV: "true"
59   GIT_DEPTH: "20"
60   GIT_SUBMODULE_STRATEGY: "none"
61   GET_SOURCES_ATTEMPTS: "3"
62   KNAPSACK_RSPEC_SUITE_REPORT_PATH: knapsack/report-master.json
63   FLAKY_RSPEC_SUITE_REPORT_PATH: rspec_flaky/report-suite.json
64   RSPEC_TESTS_MAPPING_PATH: crystalball/mapping.json
65   RSPEC_PACKED_TESTS_MAPPING_PATH: crystalball/packed-mapping.json
66   BUILD_ASSETS_IMAGE: "false"
67   ES_JAVA_OPTS: "-Xms256m -Xmx256m"
68   ELASTIC_URL: "http://elastic:changeme@elasticsearch:9200"
69   DOCKER_VERSION: "20.10.1"
70   CACHE_CLASSES: "true"
71
```

```
include:
- local: .gitlab/ci/build-images.gitlab-ci.yml
- local: .gitlab/ci/cache-repo.gitlab-ci.yml
- local: .gitlab/ci/cng.gitlab-ci.yml
- local: .gitlab/ci/docs.gitlab-ci.yml
- local: .gitlab/ci/frontend.gitlab-ci.yml
- local: .gitlab/ci/global.gitlab-ci.yml
- local: .gitlab/ci/memory.gitlab-ci.yml
- local: .gitlab/ci/pages.gitlab-ci.yml
- local: .gitlab/ci/qa.gitlab-ci.yml
- local: .gitlab/ci/reports.gitlab-ci.yml
- local: .gitlab/ci/rails.gitlab-ci.yml
- local: .gitlab/ci/vendored-gems.gitlab-ci.yml
- local: .gitlab/ci/review.gitlab-ci.yml
- local: .gitlab/ci/rules.gitlab-ci.yml
- local: .gitlab/ci/setup.gitlab-ci.yml
- local: .gitlab/ci/dev-fixtures.gitlab-ci.yml
- local: .gitlab/ci/test-metadata.gitlab-ci.yml
- local: .gitlab/ci/yaml.gitlab-ci.yml
- local: .gitlab/ci/releases.gitlab-ci.yml
- local: .gitlab/ci/notify.gitlab-ci.yml
- local: .gitlab/ci/dast.gitlab-ci.yml
- local: .gitlab/ci/workhorse.gitlab-ci.yml
- local: .gitlab/ci/graphql.gitlab-ci.yml
```

```
29
30 ### Test simple pipeline
31
32 simple:
33   script:
34     - echo "bellashell"
```

```
14 Skipping Git submodules setup
15 Executing "step_script" stage of the job script
16 Using docker image sha256:300e315adb2f96afe5f0b2b8c04efc1 ...
17 $ echo "bellashell"
18 bellashell
19 Cleaning up file based variables
20 Job succeeded
```

## Pipeline

Oltre le *keywords* da utilizzare per "costruire" la *pipeline*, ci sono delle variabili ambientali che vengono definite nell'esecuzione dei *jobs* e sono utili per

- controllare il comportamento dei *jobs* e della *pipeline*
- assumere un valore che posso riutilizzare nel *job*
- evitare valori *hard-coded* nel file `.gitlab-ci.yml`
- [https://docs.gitlab.com/ee/ci/variables/predefined\\_variables.html](https://docs.gitlab.com/ee/ci/variables/predefined_variables.html)
- posso anche definire delle variabili *custom*
- La *keyword stage*: definisce quale *job* viene eseguito durante quello *stage*. Qui viene descritta la *keyword stage* <https://docs.gitlab.com/ee/ci/yaml/README.html#stage>

```
test_variable:
  stage: test
  script:
    - echo $CI_JOB_STAGE
```

```
variables:
  TEST_VAR: "All jobs can use this variable's value"

job1:
  variables:
    TEST_VAR_JOB: "Only job1 can use this variable's value"
  script:
    - echo $TEST_VAR and $TEST_VAR_JOB
```

Esempio di:

- *pipeline* che genera un *artifact* (un prodotto del job)

L'*artifact* è un pdf che sarà rimosso dopo una settimana *expire\_in* ed è nella directory *paths* dove *paths* è relativo al repository dove è eseguito il job

**N.B.** In caso di *artifact* inserire il più possibile la *keyword expire\_in*: (seguita da xx hr, day, week etc.) per evitare che gli *artifact* occupino spazio e restino lì per un tempo indeterminato

```
pdf:
  script: xelatex mycv.tex
  artifacts:
    paths:
      - mycv.pdf
    expire_in: 1 week
```



## Keywords *stages* & *stage*

### Stages

- Definisce gli *stage* che contengono gruppi di *job* →
- L'ordine degli argomenti definisce l'ordine di esecuzione dei *job*
- E' definita globalmente
- Se non è definita, gli stage di default sono build, test, deploy

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

### Stage

- Definisce quale *job* è eseguito in quello *stage* →
- Se *stages* non è definita ci sono 5 stage di default (eseguiti in ordine) *.pre*, *build*, *test*, *deploy*, *.post*
- Ad un *job* senza *stage* viene assegnato lo stage *test*

```
stages:  
- build  
- test  
- deploy  
  
job 0:  
  stage: .pre  
  script: make something useful before build stage  
  
job 1:  
  stage: build  
  script: make build dependencies  
  
job 2:  
  stage: build  
  script: make build artifacts  
  
job 3:  
  stage: test  
  script: make test  
  
job 4:  
  stage: deploy  
  script: make deploy
```

Chi esegue il codice definito in `.gitlab-ci.yml`?

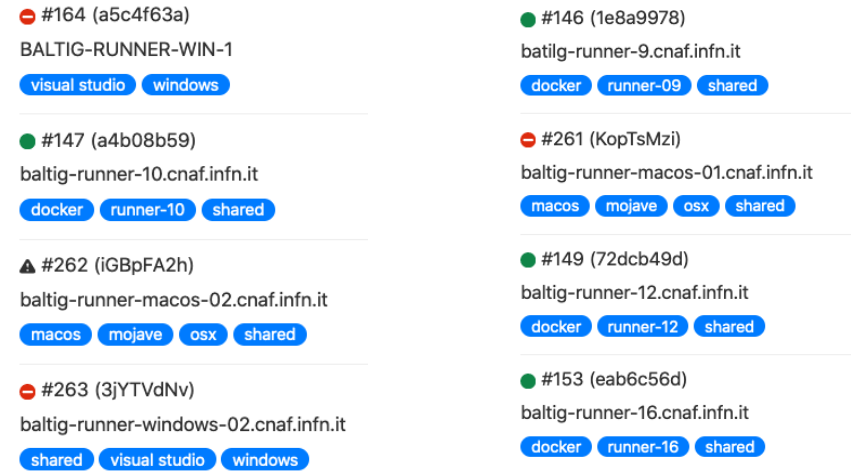
- Questo compito è svolto dai *runner*
- In GitLab INFN ci sono gli *shared runners*, 16 runner a disposizione degli utenti

Quando utilizzo la CI controllo se ci sono *runner* disponibili

[https://docs.gitlab.com/ee/ci/quick\\_start/index.html#ensure-you-have-runners-available](https://docs.gitlab.com/ee/ci/quick_start/index.html#ensure-you-have-runners-available)

Il *job* viene eseguito su uno dei *runner* a disposizione

La *dashboard* segnala il tempo di esecuzione di ogni *job*



ID	Name	Tags
#164 (a5c4f63a)	BALTIG-RUNNER-WIN-1	visual studio windows
#147 (a4b08b59)	baltig-runner-10.cnaf.infn.it	docker runner-10 shared
#262 (iGBpFA2h)	baltig-runner-macos-02.cnaf.infn.it	macos mojave osx shared
#263 (3jYTVdNv)	baltig-runner-windows-02.cnaf.infn.it	shared visual studio windows
#146 (1e8a9978)	batilg-runner-9.cnaf.infn.it	docker runner-09 shared
#261 (KopTsmzi)	baltig-runner-macos-01.cnaf.infn.it	macos mojave osx shared
#149 (72dcb49d)	baltig-runner-12.cnaf.infn.it	docker runner-12 shared
#153 (eab6c56d)	baltig-runner-16.cnaf.infn.it	docker runner-16 shared

```
1 Running with gitlab-runner 13.11.0 (7f7a4bb0)
2   on baltig-runner-8 3242bd47
3   Preparing the "docker" executor
4   Using Docker executor with image centos:latest ...
5   Pulling docker image centos:latest ...
6   Using docker image sha256:300e315adb2f96afe5f0b2780b87f28ae95231fe3bdd1e16b9ba606307728f55 for centos:latest
7   b8c04efc1 ...
8   Preparing environment
9   Running on runner-3242bd47-project-3564-concurrent-0 via baltig-runner-8.cnaf.infn.it...
11  Getting source from Git repository
```

## La dashboard consente la gestione della CI (pipeline, job...)

The screenshot displays the GitLab CI/CD dashboard. On the left, there are summary cards for failed pipelines: #45938 (failed) and #45931 (failed) with the error 'yml invalid error'. A central sidebar menu includes 'Merge requests', 'CI/CD', 'Pipelines', 'Editor', 'Jobs', and 'Schedules'. The main area shows a table of pipelines with columns for Status, Pipeline ID, Triggerer, Commit, Stages, and Duration. A dropdown menu for pipeline #45968 offers options to 'Download pages:archive artifact' and 'Download build:archive artifact'. Below this, a 'Jobs' table lists individual jobs with columns for Status, Job ID, Pipeline, Stage, Name, Duration, and Coverage. A terminal window on the right shows the execution of a script, including the command 'cp pdf/slide\_ci\_artifact.pdf public/' and the output 'Uploading artifacts for successful job'.

Status	Pipeline	Triggerer	Commit	Stages	Duration
passed	#45968	🌐	P master -> 6f94a661 Update index.html	✓✓✓	00:01:08 4 days ago
passed	#45967	🌐	P master -> ab032976 Update .gitlab-ci-y...	✓✓✓	00:00:00 4 days ago
passed	#45966	🌐	P master -> f42480a3 Update .gitlab-ci-y...	✓✓✓	00:08:25 4 days ago

Status	Job	Pipeline	Stage	Name	Duration	Coverage
passed	#165441 v master -> 6f94a661	#45968 by 🌐	pages	pages	00:00:36 4 days ago	📄 C
passed	#165440 v master -> 6f94a661	#45968 by 🌐	build	build	00:00:31 4 days ago	📄 C
passed	#165438 v master -> ab032976	#45967 by 🌐	pages	pages	00:00:30 4 days ago	📄 C
passed	#165437 v master -> ab032976	#45967 by 🌐	build	build	00:06:30 4 days ago	📄 C
passed	#165435 v master -> f42480a3	#45966 by 🌐	pages	pages	00:00:37 4 days ago	📄 C

```
38 public:
39 total 4
40 -rw-rw-rw- 1 root root 184 Apr 23 21:42 index.html
41 $ cp pdf/slide_ci_artifact.pdf public/
42 $ ls -ltr pdf/ public
43 pdf/:
44 total 240
45 -rw-r--r-- 1 root root 242302 Apr 23 21:41 slide_ci_artifact.pdf
46 public:
47 total 244
48 -rw-rw-rw- 1 root root 184 Apr 23 21:42 index.html
49 -rw-r--r-- 1 root root 242302 Apr 23 21:42 slide_ci_artifact.pdf
51 Uploading artifacts for successful job
52 Uploading artifacts...
```