RÉFÉRENCE RAPIDE GITLAB CI/CD

Pipelines, jobs, stages, variables, artefacts, environnements

Bases des pipelines

Fonctionnement des pipelines

`Pipeline`	Conteneur de niveau supérieur ; un par commit/déclencheur
`Stage`	Groupe de jobs qui s'exécutent en parallèle
`Job`	Tâche unique (script) au sein d'un stage
`Runner`	Agent qui exécute les jobs

Déclencher des pipelines

`Push vers une branche`	Automatique (défaut)
`Merge request`	Via workflow:rules ou only: merge_requests
`Planification`	CI/CD → Schedules dans les paramètres du projet
`API`	POST /projects/:id/trigger/pipeline
`Manuel`	Bouton Run Pipeline dans le menu CI/CD

.gitlab-ci.yml

Configuration minimale

```
stages: [build, test, deploy]
build-job:
  stage: build
  script: echo "Compiling..."
```

Mots-clés globaux

`stages`	Définir l'ordre des stages
`default`	Valeurs par défaut pour tous les jobs
`variables`	Variables CI/CD globales
`workflow`	Contrôler quand les pipelines sont créés
`include`	Importer des fichiers YAML externes

Inclure des templates

```
include:
  - template: Auto-DevOps.gitlab-ci.yml
  - local: .ci/lint.yml
  - project: 'group/shared-ci'
    file: '/templates/deploy.yml'
```

Jobs

Définition d'un job

```
test-unit:
  stage: test
  image: node:20
  script:
    - npm ci
    - npm test
```

Mots-clés de job

`script`	Commandes shell à exécuter (requis)
`before_script`	Commandes avant le script principal
`after_script`	Commandes après (même en cas d'échec)
`image`	Image Docker pour le job
`rules`	Conditions d'inclusion du job
`needs`	Dépendances DAG (ignorer l'ordre des stages)
`allow_failure`	Le pipeline continue si le job échoue
`retry`	Nombre de tentatives automatiques (0-2)
`timeout`	Durée maximale du job

Règles

```
deploy:
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
      when: manual
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
      when: never
    - when: on_success
```

Stages

Ordre des stages

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

Stages par défaut

`.pre`	S'exécute toujours en premier
`build`	Stage par défaut 1
`test`	Stage par défaut 2
`deploy`	Stage par défaut 3
`.post`	S'exécute toujours en dernier

DAG avec needs

```
test-api:
  stage: test
  needs: ["build-api"]    # skip waiting for full stage
test-web:
  stage: test
  needs: ["build-web"]    # runs as soon as build-web done
```

Variables

Définir des variables

```
variables:
  NODE_ENV: "production"
  DB_HOST: "postgres"
job:
  variables:
    NODE_ENV: "test"      # job-level override
```

Variables prédéfinies

`CI_COMMIT_SHA`	Hash complet du commit
`CI_COMMIT_BRANCH`	Nom de la branche
`CI_COMMIT_TAG`	Nom du tag (si pipeline de tag)
`CI_PIPELINE_ID`	ID unique du pipeline
`CI_PROJECT_DIR`	Chemin de checkout du dépôt
`CI_MERGE_REQUEST_IID`	Numéro de MR (pipelines MR seulement)
`CI_REGISTRY_IMAGE`	Chemin de l'image dans le registre de conteneurs

Protégées et masquées

`Protected`	Disponible uniquement sur les branches/tags protégés
`Masked`	Masquée dans les logs des jobs
`File`	Écrite dans un fichier temporaire ; chemin dans la variable

Artefacts

Sauvegarder des artefacts

```
build:
  script: npm run build
  artifacts:
    paths: [dist/]
    expire_in: 1 week
```

Types d'artefacts

`paths`	Fichiers/répertoires à stocker
`exclude`	Patterns à ignorer
`expire_in`	Suppression automatique après la durée
`reports:junit`	XML JUnit pour le résumé des tests dans les MR
`reports:coverage_report`	Visualisation de la couverture Cobertura

Rapport JUnit

```
test:
  script: pytest --junitxml=report.xml
  artifacts:
    reports:
      junit: report.xml
```

Cache

Mettre en cache les dépendances

```
test:
  cache:
    key: ${CI_COMMIT_REF_SLUG}
    paths: [node_modules/]
  script: npm ci && npm test
```

Cache vs artefacts

`Cache`	Accélérer les jobs ; non garanti ; réutilisation par même clé
`Artifacts`	Passer des fichiers entre jobs/stages ; garanti

Politiques de cache

`pull-push`	Télécharger + téléverser (défaut)
`pull`	Télécharger seulement (plus rapide pour les consommateurs)
`push`	Téléverser seulement (pour les producteurs)

Environnements

Définir des environnements

```
deploy-staging:
  stage: deploy
  environment:
    name: staging
    url: https://staging.example.com
  script: ./deploy.sh staging
```

Fonctionnalités d'environnement

`name`	Nom de l'environnement (affiché dans l'interface)
`url`	Lien vers l'application déployée
`on_stop`	Job à exécuter quand l'environnement est arrêté
`auto_stop_in`	Arrêt automatique après la durée
`action: stop`	Marque le job comme action d'arrêt

Review Apps

```
review:
  environment:
    name: review/$CI_COMMIT_REF_SLUG
    url: https://$CI_COMMIT_REF_SLUG.example.com
    on_stop: stop-review
    auto_stop_in: 1 week
```

Docker

Construire et pousser une image

```
build-image:
  image: docker:24
  services: [docker:24-dind]
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
$CI_REGISTRY
    - docker build -t $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA .
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
```

Services (conteneurs sidecar)

```
test:
  image: python:3.12
  services:
    - postgres:16
    - redis:7
  variables:
    POSTGRES_DB: testdb
    POSTGRES_PASSWORD: secret
```

Docker-in-Docker

`docker:24-dind`	Image du service DinD
`DOCKER_TLS_CERTDIR`	Définir à '/certs' ou '' pour la config TLS
`DOCKER_HOST`	tcp://docker:2376 (TLS) ou :2375

Patterns courants

Monorepo (changes)

```
test-api:
  rules:
    - changes: [api/**/*]
test-web:
  rules:
    - changes: [web/**/*]
```

Validation manuelle de déploiement

```
deploy-prod:
  stage: deploy
  when: manual
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
```

Matrice parallèle

```
test:
  parallel:
    matrix:
      - PYTHON: ["3.10", "3.11", "3.12"]
        DB: ["postgres", "sqlite"]
  script: tox -e py${PYTHON}-${DB}
```