REFERENCIA RÁPIDA DE GITLAB CI/CD

Pipelines, jobs, stages, variables, artefactos, entornos

Fundamentos del pipeline

Cómo funcionan los pipelines

`Pipeline`	Contenedor principal; uno por commit/trigger
`Stage`	Grupo de jobs que se ejecutan en paralelo
`Job`	Tarea única (script) dentro de un stage
`Runner`	Agente que ejecuta los jobs

Activar pipelines

`Push a rama`	Automático (por defecto)
`Merge request`	Con workflow:rules o only: merge_requests
`Programado`	CI/CD → Schedules en la configuración del proyecto
`API`	POST /projects/:id/trigger/pipeline
`Manual`	Botón Run Pipeline en el menú CI/CD

.gitlab-ci.yml

Configuración mínima

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

Palabras clave globales

`stages`	Definir el orden de los stages
`default`	Valores por defecto para todos los jobs
`variables`	Variables globales de CI/CD
`workflow`	Controlar cuándo se crean los pipelines
`include`	Importar archivos YAML externos

Incluir plantillas

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

Jobs

Definición de job

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

Palabras clave de job

`script`	Comandos shell a ejecutar (requerido)
`before_script`	Comandos antes del script principal
`after_script`	Comandos después (incluso si falla)
`image`	Imagen Docker para el job
`rules`	Condiciones para incluir el job
`needs`	Dependencias DAG (omitir orden de stage)
`allow_failure`	El pipeline continúa si el job falla
`retry`	Contador de reintentos automáticos (0-2)
`timeout`	Duración máxima del job

Rules

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

Stages

Orden de stages

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

Stages por defecto

`.pre`	Siempre se ejecuta primero
`build`	Stage por defecto 1
`test`	Stage por defecto 2
`deploy`	Stage por defecto 3
`.post`	Siempre se ejecuta al último

DAG con needs

```
test-api:
  stage: test
  needs: ["build-api"]    # no esperar al stage completo
test-web:
  stage: test
  needs: ["build-web"]    # ejecuta tan pronto como termine build-
web
```

Variables

Definir variables

```
variables:
  NODE_ENV: "production"
  DB_HOST: "postgres"
job:
  variables:
    NODE_ENV: "test"      # sobreescribir a nivel de job
```

Variables predefinidas

`CI_COMMIT_SHA`	Hash completo del commit
`CI_COMMIT_BRANCH`	Nombre de la rama
`CI_COMMIT_TAG`	Nombre del tag (si es pipeline de tag)
`CI_PIPELINE_ID`	ID único del pipeline
`CI_PROJECT_DIR`	Ruta de checkout del repo
`CI_MERGE_REQUEST_IID`	Número de MR (solo en pipelines de MR)
`CI_REGISTRY_IMAGE`	Ruta de imagen del registro de contenedores

Protegidas y enmascaradas

`Protected`	Solo disponibles en ramas/tags protegidos
`Masked`	Ocultas en los logs de jobs
`File`	Escritas en archivo temporal; ruta en la variable

Artefactos

Guardar artefactos

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

Tipos de artefacto

`paths`	Archivos/directorios a almacenar
`exclude`	Patrones a omitir
`expire_in`	Auto-eliminar después de la duración
`reports:junit`	XML JUnit para resumen de pruebas en MR
`reports:coverage_report`	Visualización de cobertura Cobertura

Reporte JUnit

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

Caché

Cachear dependencias

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

Caché vs Artefactos

`Cache`	Acelerar jobs; no garantizado; reutiliza la misma clave
`Artifacts`	Pasar archivos entre jobs/stages; garantizado

Políticas de caché

`pull-push`	Descargar + subir (por defecto)
`pull`	Solo descargar (más rápido para consumidores)
`push`	Solo subir (para productores)

Entornos

Definir entornos

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

Características de entornos

`name`	Nombre del entorno (visible en la UI)
`url`	Enlace a la app desplegada
`on_stop`	Job a ejecutar cuando se detiene el entorno
`auto_stop_in`	Auto-detener después de la duración
`action: stop`	Marca el job como la acción de parada

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

Build y push de imagen

```
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 (contenedores sidecar)

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

Docker-in-Docker

`docker:24-dind`	Imagen del servicio DinD
`DOCKER_TLS_CERTDIR`	Establecer en '/certs' o '' para configuración TLS
`DOCKER_HOST`	tcp://docker:2376 (TLS) o :2375

Patrones comunes

Monorepo (changes)

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

Compuerta de despliegue manual

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

Matriz paralela

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