GitLab Continuous Integration (GitLab CI/CD)

Dan MAGIER

Dan Magier

CEO

https://www.linkedin.com/in/dan-magier-67a9374/

@MagierDan

dan@heiwa-it.com

REMERCIEMENTS

Gitlab Continuous Integration (Gitlab CI/CD)

CONTINUOUS INTEGRATION

- Continuous Integration is a software development practice in which you **build and test** software every time a developer pushes code to the application, and it happens several times a day.

Continuous Integration: TEST - BUILD

CONTINUOUS DELIVERY

- Continuous Delivery is a software engineering approach in which **continuous integration**, **automated testing**, and **automated deployment** capabilities allow software to be developed and deployed rapidly, reliably and repeatedly with minimal human intervention. Still, the **deployment to production** is defined strategically and **triggered manually**.

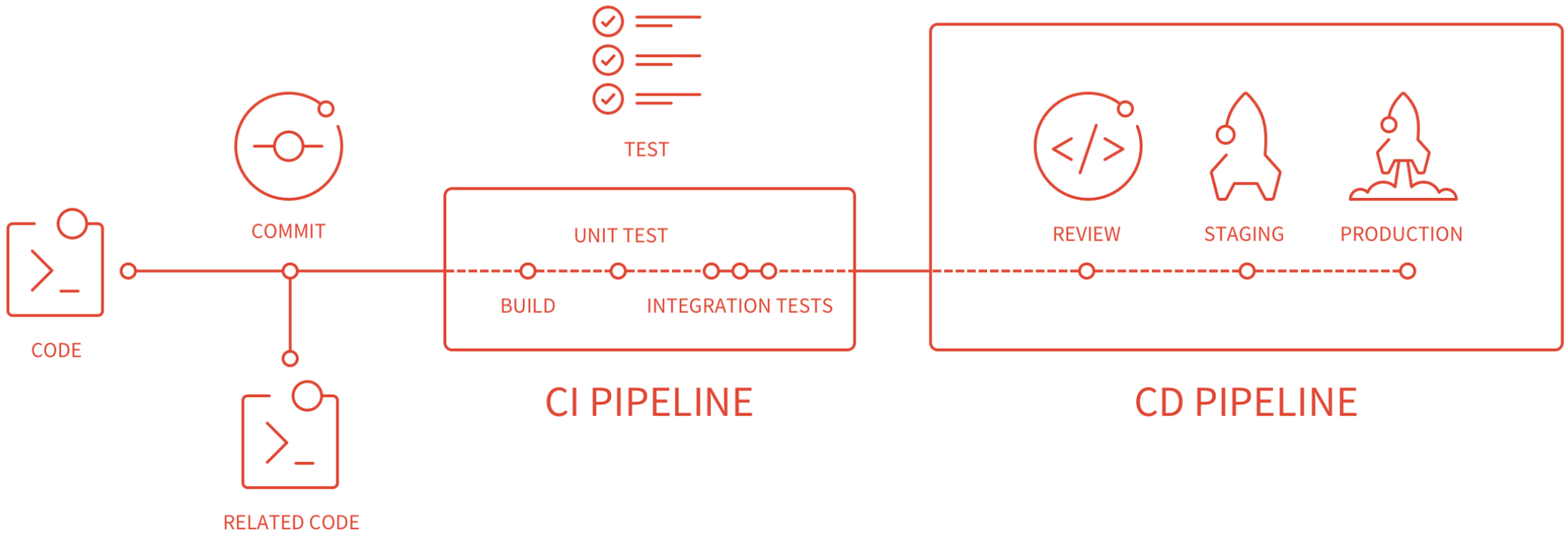
Continuous Delivery: TEST - BUILD - DEPLOY MANUALLY

CONTINUOUS DEPLOYMENT

- Continuous Deployment is a software development practice in which every code change goes through the entire pipeline and is put **into production automatically**, resulting in many production deployments every day. It does everything that Continuous Delivery does, but the process is fully automated, there's **no human intervention** at all.

Continuous Deployment: TEST - BUILD - DEPLOY AUTOMATICALLY

CI/CD PIPELINE



https://docs.gitlab.com/ce/ci/

Gitlab CI/CD Configuration File

Pipelines

PIPELINE

A pipeline is a group of jobs that get executed in stages.

All of the jobs in a stage are executed in parallel (if there are enough concurrent Runners)

If all the jobs succeed, the pipeline moves on to the next stage.
If one of the jobs fails, the next stage is not (usually) executed.

You can access the pipelines page in your project's **Pipelines** tab.

Stages

STAGES

`stages` is used to define stages that can be used by jobs and is defined globally.

The specification of `stages` allows for having flexible multi stage pipelines. The ordering of elements in `stages` defines the ordering of jobs' execution:

1. Jobs of the same stage are run in parallel.
2. Jobs of the next stage are run after the jobs from the previous stage complete successfully.

Jobs

JOBS

The YAML file defines a set of jobs with constraints stating when they should be run. You can specify an unlimited number of jobs which are defined as top-level elements with an arbitrary name and always have to contain at least the `script` clause.

Jobs are picked up by Runners and executed within the environment of the Runner. What is important, is that each job is run independently from each other.

Each job must have a unique name, but there are a few **reserved `keywords` that cannot be used as job names:**

- `image`, `services`, `stages`, `types`, `before_script`, `after_script`, `variables` and `cache`
-

Gitlab RUNNERS

RUNNER

GitLab Runner is the open source project that is used to run your jobs and send the results back to GitLab.

It is written in GO.

It is used in conjunction with GitLab CI, the open-source continuous integration service included with GitLab that coordinates the jobs.

RUNNER FEATURES

- Allows to run:
  - multiple jobs concurrently
  - use multiple tokens with multiple server (even per-project)
  - limit number of concurrent jobs per-token

- Jobs can be run:
  - locally
  - using Docker containers
  - using Docker containers and executing job over SSH
  - using Docker containers with autoscaling on different clouds and virtualization hypervisors
  - connecting to remote SSH server

RUNNER FEATURES

- Is written in Go and distributed as single binary without any other requirements
- Supports Bash, Windows Batch and Windows PowerShell
- Works on GNU/Linux, OS X and Windows (pretty much anywhere you can run Docker)
- Allows to customize the job running environment
- Automatic configuration reload without restart
- Easy to use setup with support for Docker, Docker-SSH, Parallels or SSH running environments
- Enables caching of Docker containers
- Easy installation as a service for GNU/Linux, OSX and Windows
- Embedded Prometheus metrics HTTP server

CI/CD Pipelines

CLASSICAL INFRASTRUCTURE

Non Production Environments

DEV / CI

INT

MOE

Production Environments

MOA

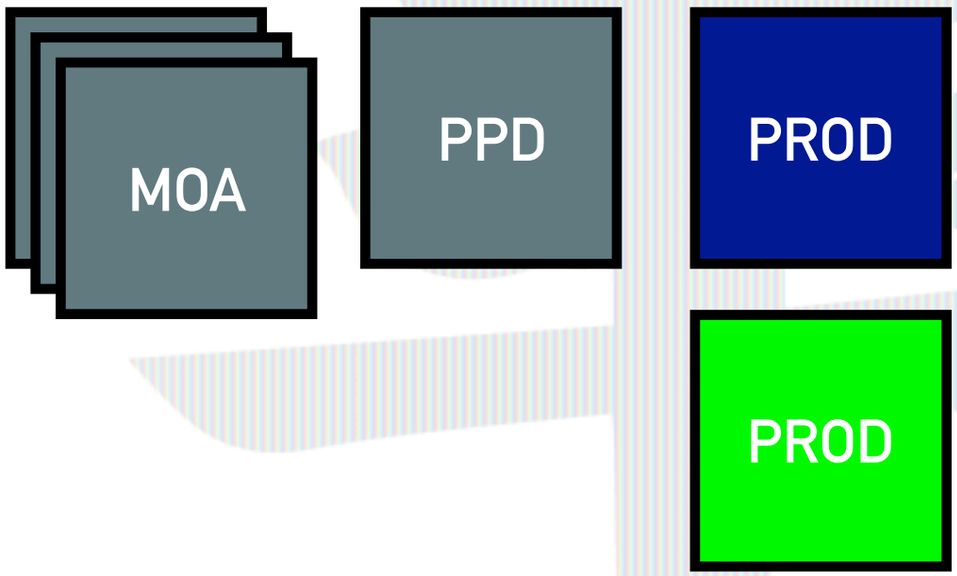
PPD

PROD

CLOUD INFRASTRUCTURE

Non Production Environments

Production Environments



WHY USING GITLAB ?

PRO

- Integrated in Gitlab
- Docker and kubernetes integration
- Parallel builds
- CI/CD by code
- No fu…g plugins ^^
- Open to other source control tools : GitHub
- Scripting is the base

CON

- Only available in Gitlab

REFERENCES

https://docs.gitlab.com/

THANK YOU

Q&A