

GitLab CI for FPGA/SoC Projects

**Carmen Marcos Sánchez de la Blanca* (CERN),
Christos Gentsos (CERN)**

8th November 2024

Outline

- **What do we do in the Engineering Software Service?**
- **What is CI, why do we care?**
- **Project purpose and objectives**
- **Gitlab CI basics**
- **Gitlab Runners, our cluster**
- **Defining CI jobs**
- **Lazy pulling**
- **Existing images**
- **Richer images: Pre-Installed Tools and CI Improvements**
- **Monitoring of resources to control access**
- **Documentation**
- **Current state and future plans**

What do we do in the Engineering Software Service?



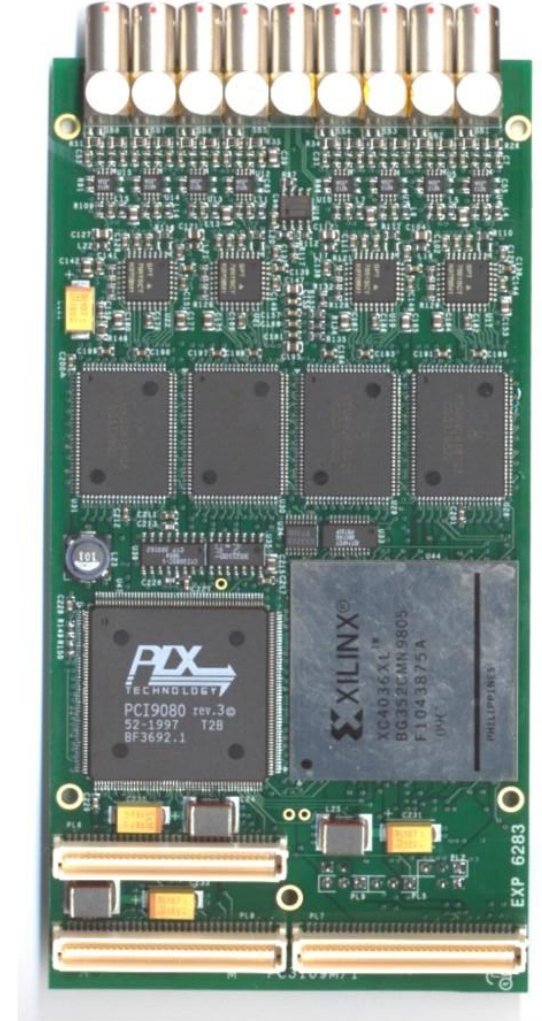
The Engineering Service facilitates access to engineering software for CERN personnel

- Applications in the areas of electronics design, mechanical design, mathematics and multiphysics simulation
- We are in charge of selecting, deploying, configuring software and integrating it to other CERN services
 - Budget management, procurement, license management for commercial software, packaging and installation



What is a FPGA?

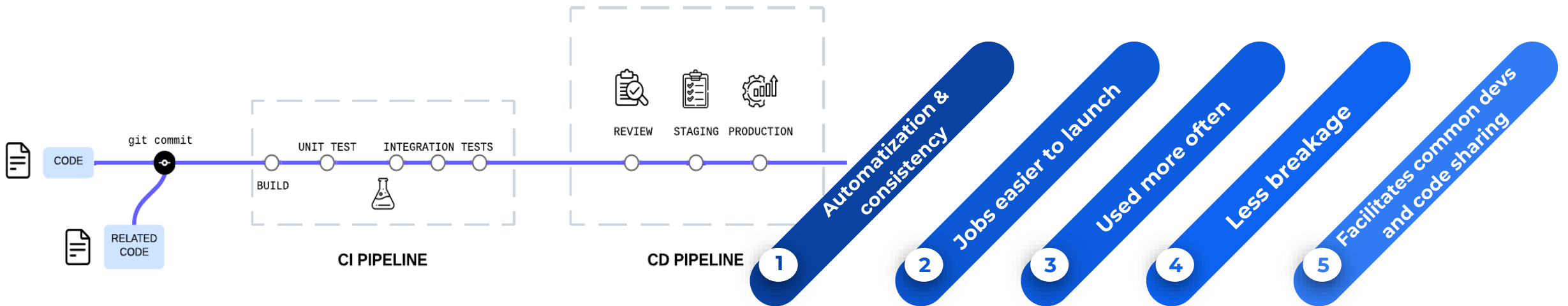
- **Field Programmable Gate Arrays:**
 - an array of gates and flip-flops that can be configured in function and connectivity in very flexible ways
 - add very specialized circuits in the mix to be fast, like SRAM blocks, DSP blocks, multi-Gbits transceivers
 - add I/Os that are widely compatible with many electrical standards
- ...and then make it programmable on the fly, as many times as you wish
- FPGA devices are very widespread from high-end cars to very custom systems
- FPGAs are everywhere at CERN, both experiments and machines:
 - very common in physics experiments (data readout, TDAQ systems, data processing)
 - Sometimes the role is critical for machine safety



What is CI, why do we care?

What is CI?

- Continuous Integration, commonly used in software development
- Involves *regularly* and *automatically* integrating code changes from multiple contributors into a shared codebase
- The term has come to be mostly used to refer to automatic testing and building at the repository level



Project purpose and Objectives

Why this project?

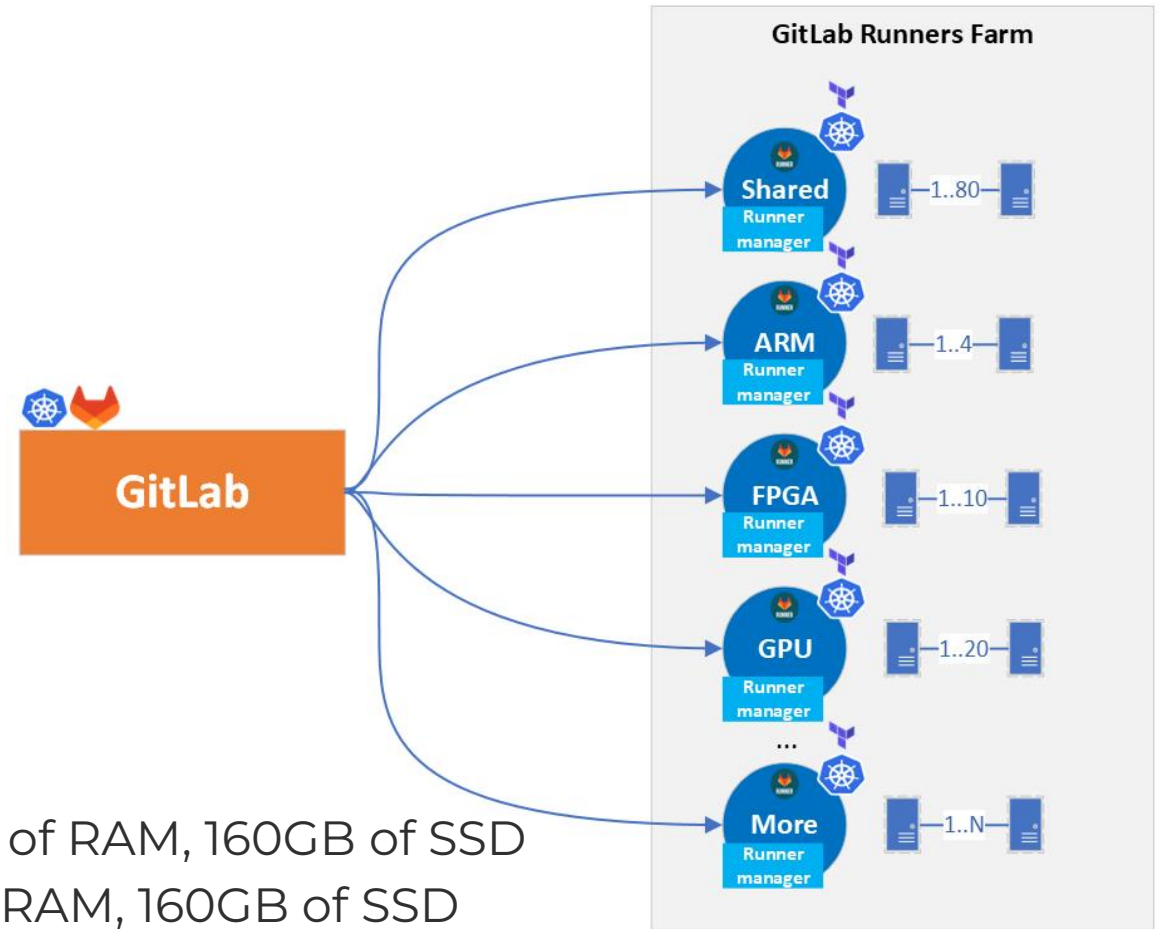
- Some teams have already done considerable work to get there, but there is a lack of centralized infrastructure
 - Hard to efficiently set up and maintain build clusters
 - Huge tools O(100GB) don't make it any easier
 - Effort multiplied across all the different teams, very inefficient at scale

Our objectives:

- Scalable, maintained VM-based k8s cluster to run the CI jobs, adapted to the resource requirements of EDA tools
- Easy to use Docker images for EDA tools, like simulators and toolchains
- Document the above clearly to support the 100+ projected users

GitLab Runners, our cluster

- A runner is a service running on a computer or a k8s cluster that will deploy CI jobs
- We offer *Instance* (shared) runners, *visible to all projects in gitlab.cern.ch*
- Our runners will pick up the jobs that carry specific tags: fpga-mid, fpga-large
- fpga-large: m4.2xlarge VMs, 16 VCPUS, ~57.5GB of RAM, 160GB of SSD
fpga-mid: m2.2xlarge VMs, 16 VCPUS, ~29GB of RAM, 160GB of SSD



[\(Kubernetes official docs at CERN\)](#)

Example CI job currently running:

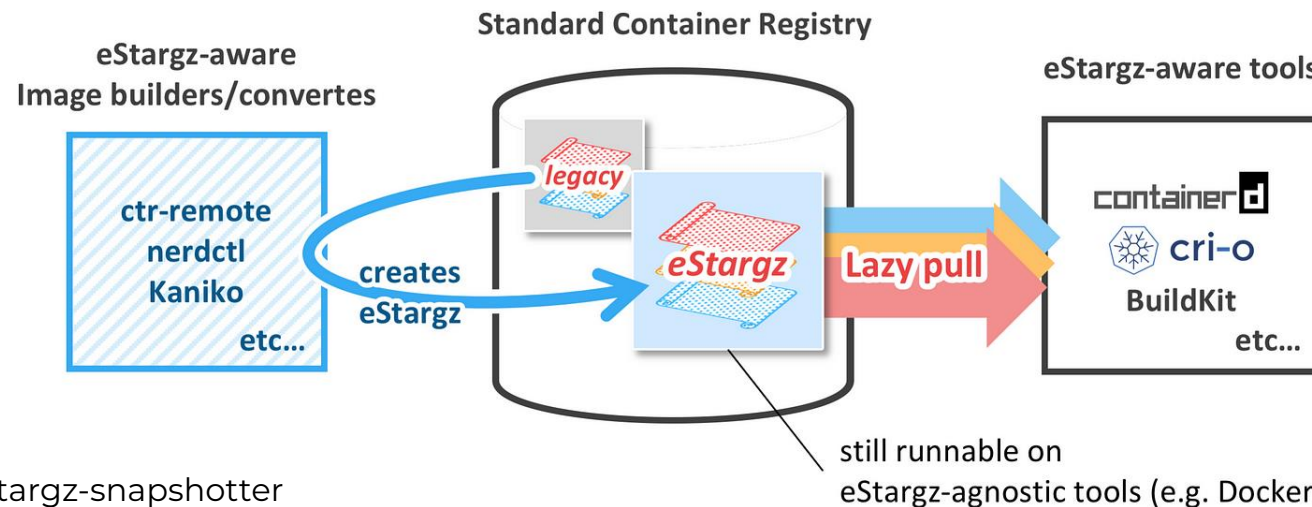
```
stages:
  - test

test-job:
  stage: test
  tags:
    - fpga-mid
  image: registry.cern.ch/ci4fpga/vivado:2022.2
  script:
    - vivado -mode batch -source build.tcl
  artifacts:
    paths:
      - "*.rpt"
      - "*.bit"
```

- In a special file (.gitlab-ci.yml) we can declare the stages and jobs that make up a pipeline
- For each job, we assign a runner tag and a docker image name, the commands to be ran and any resulting files to be kept
- The right runner type will be instantiated based on the tag, downloading the docker image and running the specified commands
- If all goes well, the resulting files can be found on the gitlab webpage

Lazy Pulling

- To pull the image can be one of the most time-consuming steps in the container lifecycle
 - But in many cases only a small part of the pulled data is ever read
 - Imagine a case of an FPGA toolchain installation, full of 10s of GBs of device data for all families – but we only need one family in a run!
- Lazy pulling allows a container to be started with just the necessary data pulled
 - 100GB image as an example: 15 secs vs 15 minutes (with a 1Gbps connection)
 - eStargz format, allows chunks of data to be fetched on-demand



<https://github.com/containerd/stargz-snapshotter>

Existing Images

Toolchains

- **Xilinx ISE**
- **Xilinx Vivado**
 - 2018.1
 - 2018.3
 - 2019.1 – 2019.2
 - 2021.1 – 2024.1
- **Microsemi Libero**
 - 11.8 – 11.9
 - 2021.2
 - 2023.2
- **Intel Quartus Prime**
 - 13.0-ii
 - 18.1-std, 20.1-std, 22.1-std
 - 22.3-pro, 23.3-pro
- **Lattice Diamond**
 - 3.11, 3.13

Simulators

- **Aldec Riviera-Pro**
 - 2022.04 – 2024.04
- **Mentor ModelSim**
 - 10.7g, 2021.1 – 2021.3
- **Mentor Questasim**
 - 10.7g, 2019.4, 2021.1, 2022.3, 2023.4
- **Ghdl**
 - 3.0.0, 4.0.0, 4.1.0, dev
- **NVC**
 - 1.14.0, dev
- **Icarus Verilog**
 - 12.0, dev

Others

- **Doxygen**
 - 1.9.2, 1.12.0
- **VHDL Style Guide**
 - 3.20.0, 3.26.0
- **YosysHQ OSS CAD Suite**
 - 2024.04, 2024.10, dev
- **PetaLinux**
 - 2018.1, 2019.2
 - 2021.1 – 2022.2
 - 2024.1
- **Visual Elite**
 - 2018.06, 2019.3

Richer images: Pre-Installed Tools and CI Improvements

- **General improvements:**

- Many tools pre-installed (embedded compilers, bootgen, hdlmake...)
- Username spoofing to have license servers aware of the user
- License availability check on image start prevent CI jobs from draining the license pool and annoying us. Ensures at least 2 licenses are free, limits CI jobs to 30% of licenses, waits until it's OK
- No need to run setup scripts or set env. variables
- Jobs start quickly because of lazy pulling
- Simulation libraries pre-built for all versions of almost all toolchains loadable by simply copying modelsim.ini or just using the path

Monitoring of resources to control access

We automatically push job-related data to an OpenSearch instance for real-time usage monitoring and analytics. The data includes:

```
31/10/24-15:01:59, start: image registry.cern.ch/ci4fpga/ghdl:4.1.0, triggered as web by ***** from job check-***** (ID: *****)  
in project CROME***
```

```
31/10/24-15:03:59, start: image registry.cern.ch/ci4fpga/vivado:2018.1, triggered as web by ***** from job build-hw***** (ID:  
*****) in project CROME***
```

```
31/10/24-15:19:14, start: image registry.cern.ch/ci4fpga/vivado:2024.1, triggered as push by ***** from job generate-package  
**** (ID: *****) in project cms-tracker*****
```

```
31/10/24-16:50:15, end: image registry.cern.ch/ci4fpga/vivado:2022.2, triggered as web by ***** from job device***** (ID:*****)  
in project be*****
```

How do we create our documentation ?

Everytime that we add a new toolchain, version, modules, software to images...

- Do we have to manually modify the documentation? ❌

```
UBUNTU_BASE_PKGS = ['apt-uti
                    'python3
                    'libxren
                    'xvfb',
                    'unzip',
                    'autogen
                    'gperf',
                    'wget',
                    'netcat'
                    'coccine
                    'liblz4-
                    'lz4',
                    'python3
                    'python3
                    'python3
                    'python3
                    'u-boot-
                    'libxcb-
                    'libxcb-
                    'libxcb-
                    'llvm-de
                    'libboos

# Define base packages to in
PACKAGES_TO_INSTALL = {'c6':
                       'c7':
                       'cs8'
                       'c8':
                       'ubun

{% set OS = 'ubuntu' %}
FROM docker.io/{{OS}}:{{ OSVER[0]

{{LABEL_AUTHORS}}
{{LABEL_SOURCE}}

{{EXPORT_TZ}}

{{INSTALL_BASE_PACKAGES[OS]}}
{{PM_CLEANUP[OS]}}

{{UBUNTU_USE_BASH}}
{{UBUNTU_GEN_LOCALE}}

{{INSTALL_HDLMAKE}}
{{INSTALL_CHEBY}}
{{INSTALL_FUSESOC}}
{{INSTALL_VUNIT}}
{{INSTALL_COCOTB}}
{{INSTALL_BOOTGEN}}
{{INSTALL_UHAL}}

{{INSTALL_LM32_GCC_453}}
{{INSTALL_RISCV_GCC_1120}}
{{INSTALL_RISCV_GCC_1320[OS]}}

{{INSTALL_LMUTIL}}
{{INSTALL_SETUP_SCRIPT}}
{{INSTALL_SUDO_SCRIPT}}
```

- Jinja Dockerfile templates
- Render the dockerfiles
- Define the packages to install
- Generation of docs with python scripts
- There the documentation pipeline is triggered

Documentation [\(link\)](#)

Electronics

images on my computer?

What kind of support is available if I encounter issues with the CI4FPGA container images?

How do I modify or update the container images provided in CI4FPGA for custom use cases?

How does CI4FPGA handle tool version updates in its Docker images?

Can I run multiple FPGA tools simultaneously in a single CI job using CI4FPGA?

[.gitlab-ci.yml examples](#)

[ISE image](#)

[Riviera-PRO image](#)

[GHDL image](#)

[NVC image](#)

[Doxygen image](#)

[VHDL Style Guide image](#)

[YosysHQ OSS CAD Suite](#)

[Libero image](#)

[Modelsim image](#)

[Questasim image](#)

[Diamond image](#)

[Quartus image](#)

[Petalinux image](#)

[Visual Elite image](#)

[Vivado image](#)

[Icarus Verilog image](#)

[CentOS 6 base OS image](#)

[CentOS 7 base OS image](#)

[CentOS 8 base OS image](#)

Continuous Integration for FPGA development (CI4FPGA)

Welcome to the (still a work-in-progress) Continuous Integration (CI) to FPGA development.

CI helps automate testing, verification, and pre-built Docker images we've prepared on your local workstation, without any special

How to use

Two simple steps: first target our cluster, then add its path to the job's `image keyword`. Next, the `PATH`. Some common utilities (like `hd` except the Centos 6-based ISE), you can

List of tools available as images

The following tools are available in the repository:

- [Xilinx ISE](#)
- [Riviera-PRO](#)
- [GHDL](#)
- [NVC](#)
- [Doxygen](#)
- [VHDL Style Guide](#)
- [YosysHQ OSS CAD Suite](#)
- [Microsemi Libero](#)
- [ModelSim](#)

Xilinx Vivado CI Docker Image

Vivado has been Xilinx's main toolchain for HDL design

Since version 2020.2, Vivado comes packages with Vitis combination of Vivado for hardware design and Vitis for

Note

This documentation is about the `ci4fpga` container image of this

You can use this image by setting the `image` key in your job with the version you need). For more details on how to

The [CERN CentOS 7.9](#) base image is used for the following

- 2018.1
- 2018.3
- 2019.1
- 2019.2
- 2020.2
- 2021.1
- 2021.2
- 2022.1

The [Ubuntu 22.04 LTS](#) base image is used for the following

- 2022.2
- 2023.1
- 2023.2
- 2024.1

To learn what other tools are pre-installed (e.g. `hdlmake` or `VUnit`), please visit the base OS image documentation pages, above.

Ubuntu 22.04 LTS CI base docker image

Ubuntu is a popular open-source Linux distribution, based on the Debian architecture, that happens to be supported for a number of EDA tools. Its 22.04 LTS release will be supported until April 2027.

List of installed tools:

- Python 3.10.12
- Git 2.34.1
- Hdlmake 3.4dev1 [tool:unx cmd:unx]
- Cheby 1.7.dev0
- FuseSoC 2.3
- VUnit 4.7.0
- cocotb 1.9.1
- Bootgen v2023.2
- IPBus uHAL 2.8.15
- GCC 4.5.3 for LM32
- GCC 11.2.0 for RISC-V

List of packages installed explicitly:

Current adoption, challenges, future plans

Announced the beta release

- 22 users
- 27 projects
- 21 images

Currently

overcommitting nodes to maximize efficiency but this causes unexpected failures



Very positive feedback

- Jobs are mostly trivial to migrate from other setups
- We can quickly add anything missing to the images

- **Cluster sizing**
- **Balancing resource/efficiency**
- **User convenience**

Future plans

- **Add more clusters for better resource management without failures**
- **Direct users to the right cluster at the end of the job according to real usage (cgroups v2 gives peak RAM info)**

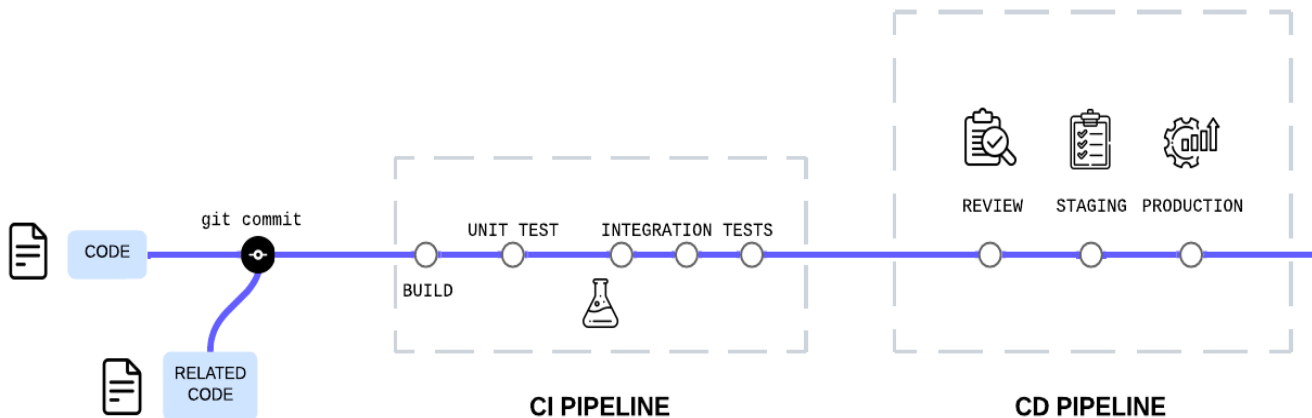
Many thanks!



Any questions?

Gitlab CI basics

- Code changed often → sometimes changes reach the main branch many times/day
- Each change can introduce a bug, especially when touching fundamental building blocks in complex systems
- Ideally one should run a “full” test suite and build after each change to the main branch



Gitlab CI gives tools to do exactly that

- Pipeline can be started after each push or merge request
- A pipeline can bundle many jobs together: tests, builds, doc generation

Cluster data

- Currently we have 2 different clusters: fpga-mid and fpga-large
 - fpga-large:
 - 3 nodes
 - Gitlab runner can run up to 15 jobs concurrently (5 jobs per node)
 - Running on m4.2xlarge VMs w/ 57GBRAM, 16 VCPUs, 160GB SSD
 - fpga-mid:
 - 3 nodes
 - GitLab runner can run up to 15 jobs concurrently (5 jobs per node)
 - Running on m2.2xlarge VMs w/29GBRAM, 16 VCPUs, 160GB SSD
- New logic for allocating jobs:
 - First try of allocation can result in error, second will redirect to the right cluster
 - cgroups v2 gives peak RAM info
- For every cluster, the job concurrency limit would be set to the number of machines
 - Avoiding jobs to get killed by the OOM killer

Cluster monitoring:

