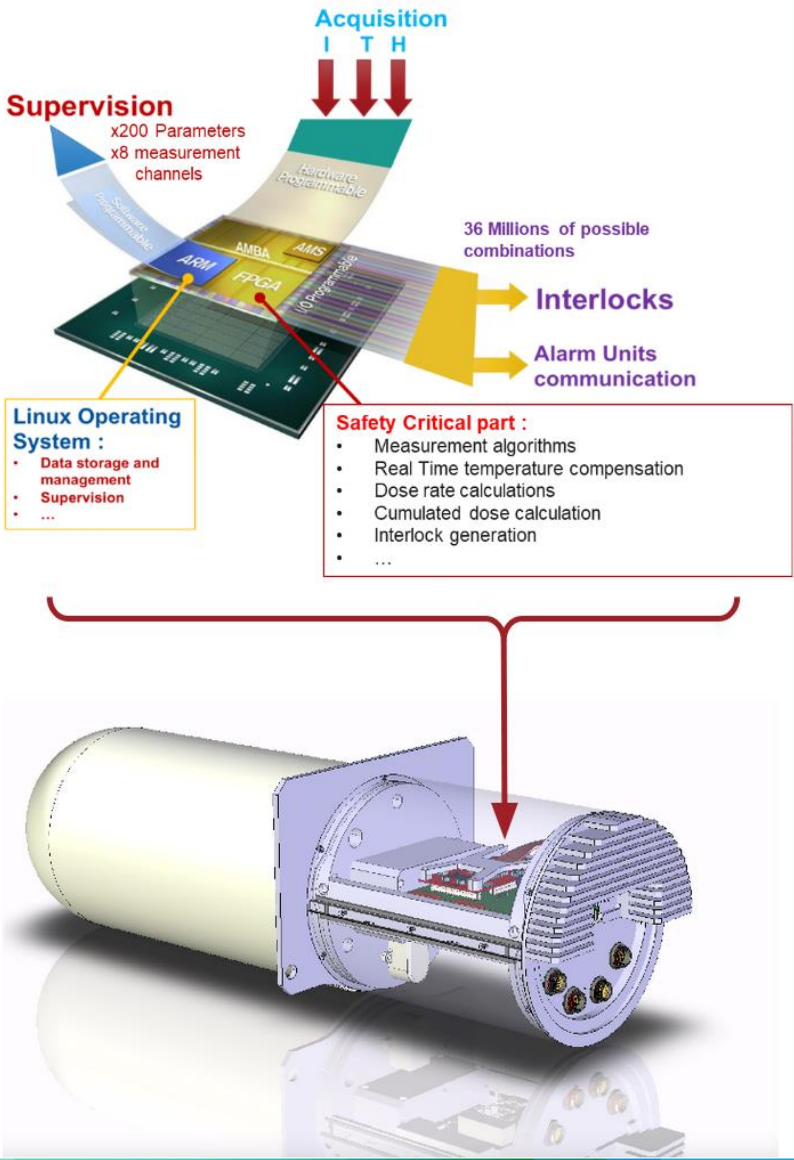
HSE

Occupational Health & Safety
and Environmental Protection unit

Hardware/Software Co-Design with Gitlab CI

CROME: CERN RadiatiOn Monitoring Electronics



Amitabh Yadav, Hamza Boukabache, CROME Team

amitabh.yadav@cern.ch

SoC Interest Group Meeting

23/11/2022

Reference: https://codimd.web.cern.ch/s/7VHssRXbY#

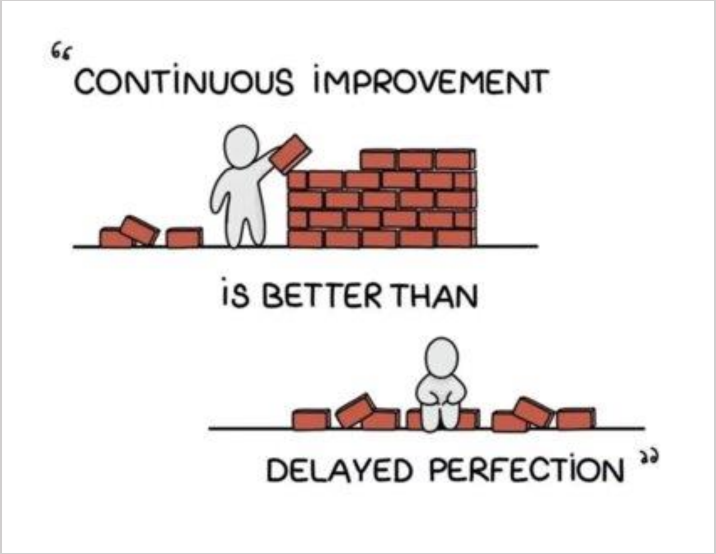
Agenda

- Gitlab CI - what and why?
- CI and Docker Overview
- CROME CI Pipeline
- Key Features – Dependencies and CI Special Variables
- Building Xilinx Petalinux Image through CI
- Future Steps – Verification and Continuous Deployment.

Continuous Integration (CI)

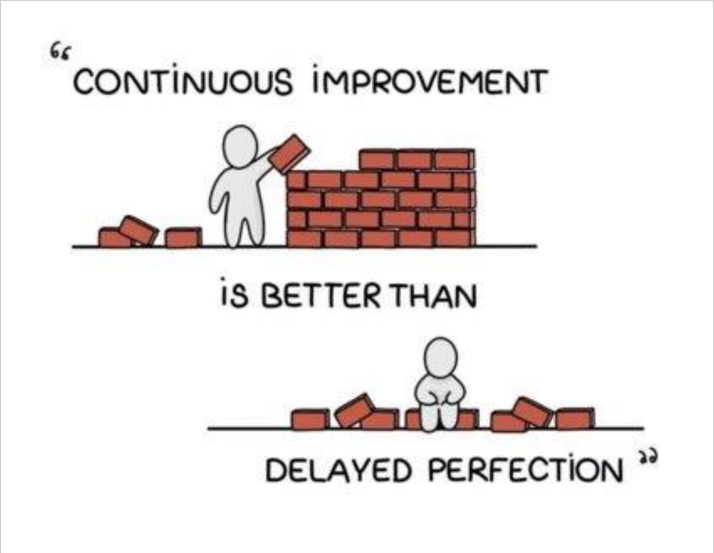
Gitlab CI - what and why?

Continuous Integration (CI) is the practice of continuously integrating and verifying the code changes automatically through CI pipelines.



Gitlab CI - what and why?

Continuous Integration (CI) is the practice of continuously integrating and verifying the code changes automatically through CI pipelines.



Advantages:

- Ensures successful HW and SW build/compilation.
- Automatic code quality and performance testing.
- Early detection of errors, bug tracking, reduced integration problems, and faster deployment.
- Reproducible builds using Docker containers [1]
- Code packaging and deployment.

[1] SY-EPC-CCE Gitlab container-registry: https://gitlab.cern.ch/cce/docker_build/container_registry

CI Pipeline

- **CI Pipeline:**
  - A pipeline is a sequence of scripted steps that will be executed on the code in repository.
  - The steps are defined in the file **.gitlab-ci.yml** and is placed in the root of the project repository.
  - Gitlab detects the YAML file and initiates the GitlabCI script.
  - No modifications to the project repository.
  - At the end, any new files/outputs created during the process of execution of a CI script are called **artifacts**.

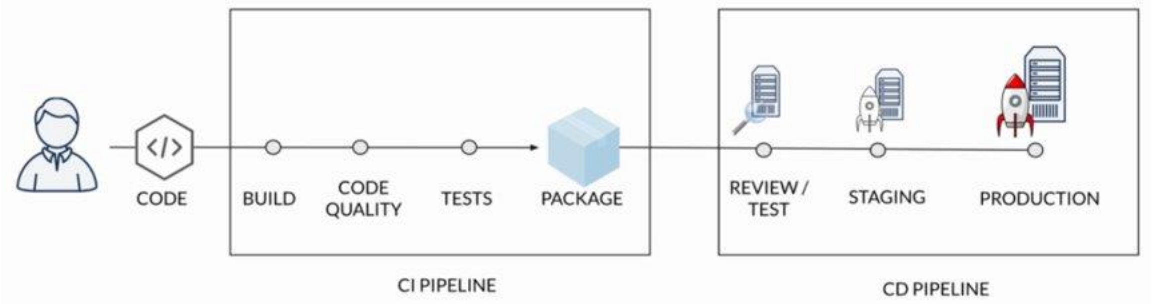


Image source: docs.gitlab.com

CI Pipeline - continued

- **CI pipeline components - Jobs and Stages:**
  - A pipeline is composed of independent **jobs** that execute scripts.
  - Jobs are grouped into **stages**.
  - Stages run in sequential order, but jobs within stages run in parallel.
- **Artifacts:**
  - Archive of generated output files and directories.
  - Accessible through GitLab UI or the API.

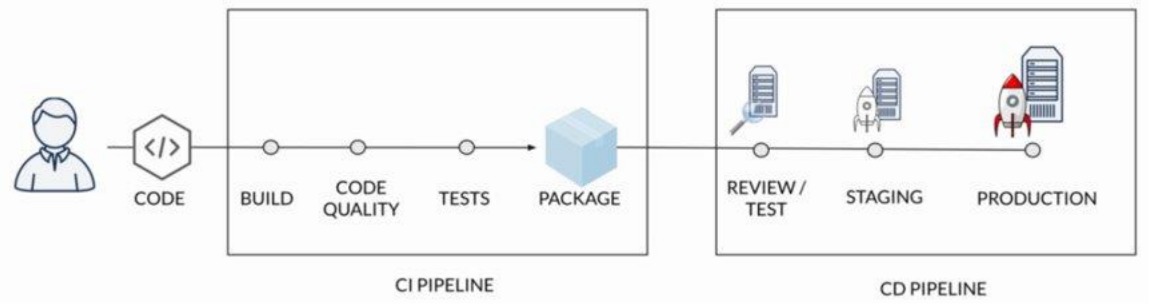


Image source: docs.gitlab.com

Gitlab CI Workflow

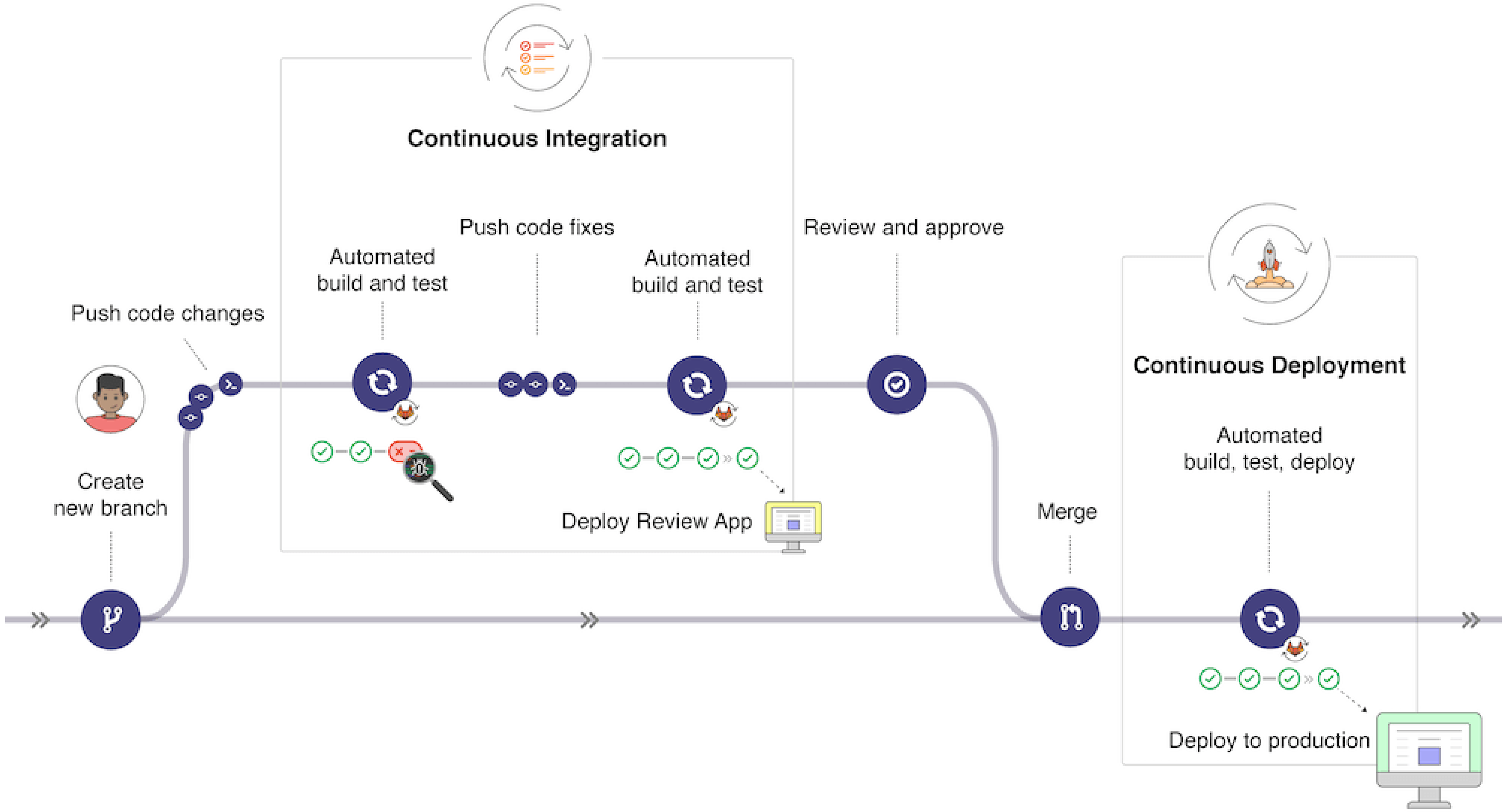
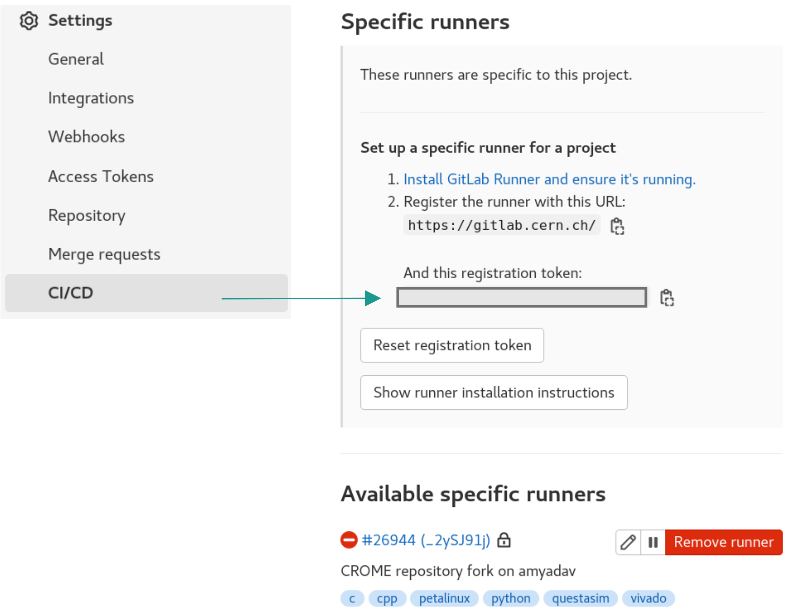


Image source: docs.gitlab.com

Gitlab CI Backend

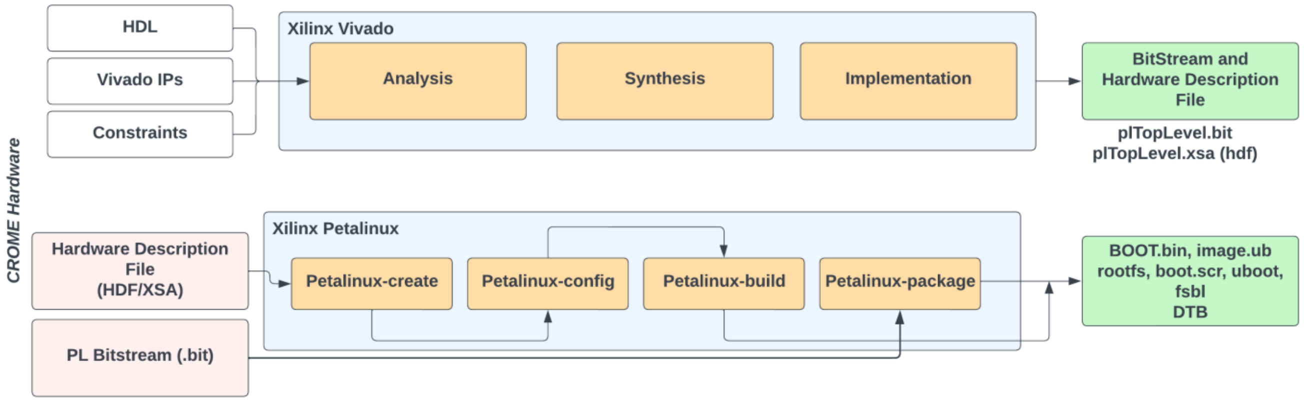
- Gitlab Runner:
  - Runners are computers/virtual machines where we Gitlab CI scripts get executed.
  - For specialized/large software that require configurations of our own, we make use of CERN OpenStack virtual machine.
  - For this we need to install a runner on the system and register it as a gitlab-runner for our repository.



https://docs.gitlab.com/runner/install/
https://docs.gitlab.com/runner/register/

An example CI Pipeline

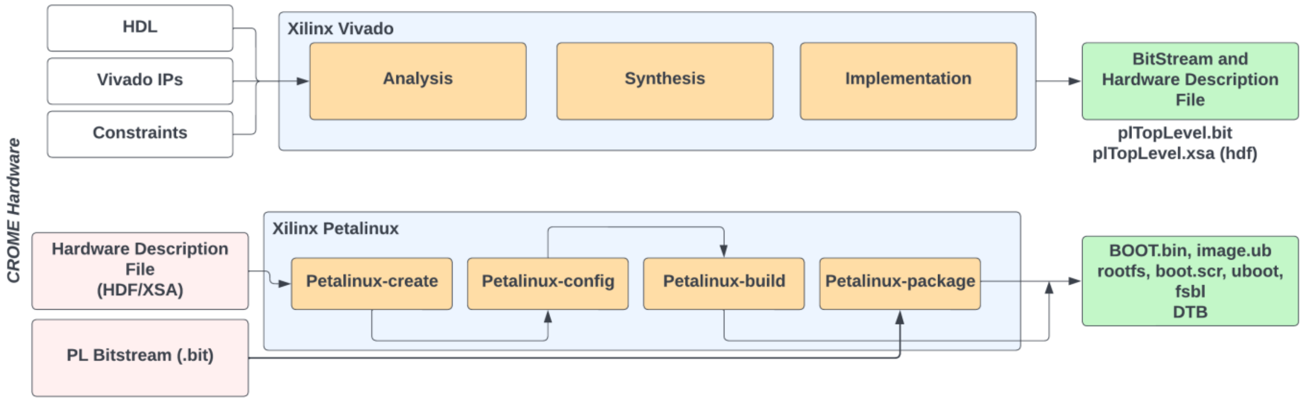
An example CI Pipeline



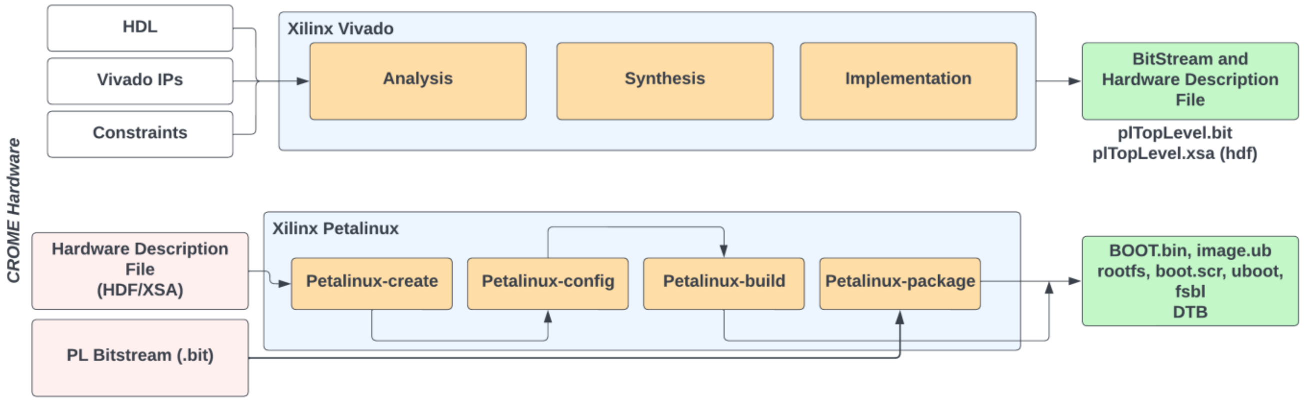
Amitabh Yadav, amitabh.yadav@cern.ch

An example CI Pipeline


```
stages:           # List of stages for jobs, and their order of execution
  - build
  - test

build-software:       # This job runs in the build stage, which runs first.
  stage: build
  before_script:
    - yum -y install doxygen
  script:
    - gcc --version
    - cd CMPU/zynq/sw/ROMULUSlib/
    - make RUN_ON_PC=1
    - echo "Building ROMULUSlib for x86 architecture."
    - export LD_LIBRARY_PATH="CMPU/zynq/sw/ROMULUSlib/"
    - cd ../embedded_linux_userspace_app
    - make RUN_ON_PC=1
    - echo "Embedded_Application and ROMULUSlib Compiled Successfully"
  artifacts:
    paths:
      - CMPU/zynq/sw/ROMULUSlib/
      - CMPU/zynq/sw/embedded_linux_userspace_app/
    expire_in: 7d
```



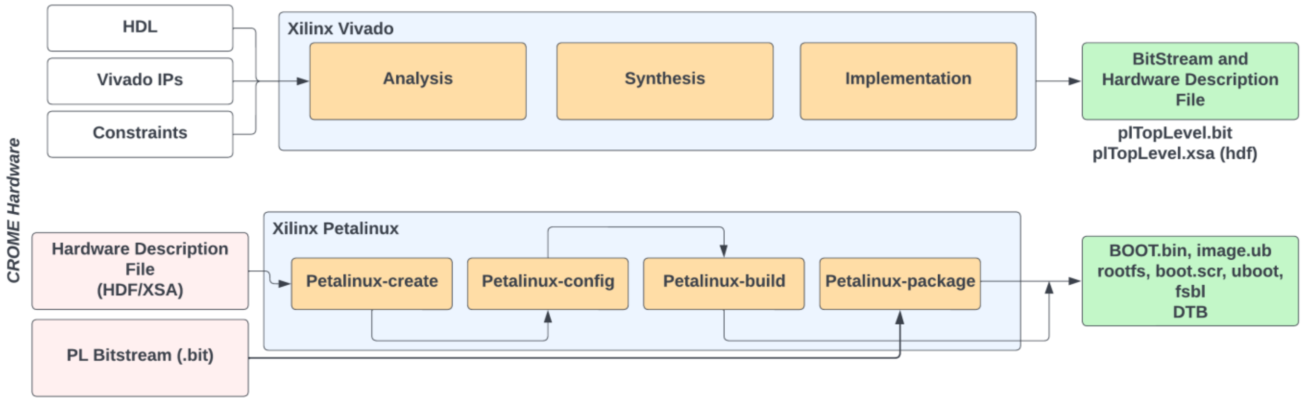
An example CI Pipeline




```
stages:          # List of stages for jobs, and their order of execution
  - build
  - test

build-software:       # This job runs in the build stage, which runs first.
  stage: build
  before_script:
    - yum -y install doxygen
  script:
    - gcc --version
    - cd CMPU/zynq/sw/ROMULUSlib/
    - make RUN_ON_PC=1
    - echo "Building ROMULUSlib for x86 architecture."
    - export LD_LIBRARY_PATH="CMPU/zynq/sw/ROMULUSlib/"
    - cd ../embedded_linux_userspace_app
    - make RUN_ON_PC=1
    - echo "Embedded_Application and ROMULUSlib Compiled Successfully"
  artifacts:
    paths:
      - CMPU/zynq/sw/ROMULUSlib/
      - CMPU/zynq/sw/embedded_linux_userspace_app/
    expire_in: 7d
```

An example CI Pipeline


```
stages:           # List of stages for jobs, and their order of execution
  - build
  - test

build-software:       # This job runs in the build stage, which runs first.
  stage: build
  before_script:
    - yum -y install doxygen
  script:
    - gcc --version
    - cd CMPU/zynq/sw/ROMULUSlib/
    - make RUN_ON_PC=1
    - echo "Building ROMULUSlib for x86 architecture."
    - export LD_LIBRARY_PATH="CMPU/zynq/sw/ROMULUSlib/"
    - cd ../embedded_linux_userspace_app
    - make RUN_ON_PC=1
    - echo "Embedded_Application and ROMULUSlib Compiled Successfully"
  artifacts:
    paths:
      - CMPU/zynq/sw/ROMULUSlib/
      - CMPU/zynq/sw/embedded_linux_userspace_app/
    expire_in: 7d
```




```
# Build Hardware job defined to run in Docker container based on Vivado 2021.2 image.
build_hw:
    stage: build
    image: gitlab-registry.cern.ch/cce/docker_build/vivado:2021.2
    script:
        - echo "Creating Project"
        - cd CMPU/zynq/hw/
        - make create_project
        - echo "Vivado 2021.2 project created successfully."
        - make synthesis
        - echo "Vivado 2021.2 Project SYNTHESIS completed successfully."
        - make implementation
        - echo "Vivado 2021.2 Project IMPLEMENTATION completed successfully."
        - echo "Bitstream generated successfully."
    artifacts:
      paths:
        - CMPU/zynq/hw/outputfiles
```

An example CI Pipeline

In a simple .gitlab-ci.yml file, we define:

- The commands need to run in sequence (Stages) and those that need to run in parallel (Jobs).
- The scripts that need to be run.
- before_script, after_script and variables.
- Artifacts: the built files that needs to be saved.
- Specification of whether to run the scripts automatically or trigger manually.
- Specification on which specific branch the pipeline should be executed automatically.

```
stages:          # List of stages for jobs, and their order of execution
  - build
  - test

build-software:       # This job runs in the build stage, which runs first.
  stage: build
  before_script:
    - yum -y install doxygen
  script:
    - gcc --version
    - cd CMPU/zynq/sw/ROMULUSlib/
    - make RUN_ON_PC=1
    - echo "Building ROMULUSlib for x86 architecture."
    - export LD_LIBRARY_PATH="CMPU/zynq/sw/ROMULUSlib/"
    - cd ../embedded_linux_userspace_app
    - make RUN_ON_PC=1
    - echo "Embedded_Application and ROMULUSlib Compiled Successfully"
  artifacts:
    paths:
      - CMPU/zynq/sw/ROMULUSlib/
      - CMPU/zynq/sw/embedded_linux_userspace_app/
    expire_in: 7d
```

```
# Build Hardware job defined to run in Docker container based on Vivado 2021.2 image.
build_hw:
    stage: build
    image: gitlab-registry.cern.ch/cce/docker_build/vivado:2021.2
    script:
        - echo "Creating Project"
        - cd CMPU/zynq/hw/
        - make create_project
        - echo "Vivado 2021.2 project created successfully."
        - make synthesis
        - echo "Vivado 2021.2 Project SYNTHESIS completed successfully."
        - make implementation
        - echo "Vivado 2021.2 Project IMPLEMENTATION completed successfully."
        - echo "Bitstream generated successfully."
    artifacts:
      paths:
        - CMPU/zynq/hw/outputfiles
```

Docker Executor

Using image keyword executes the CI jobs in Docker containers.

```
build_sw:
  stage: build
  image: gitlab-registry.cern.ch/cce/docker_build/petalinux:2018.1
```

The Docker executor divides the job into multiple steps:

- Prepare: Create and start the services.
- Pre-job*: Clone, restore cache and download artifacts from previous stages.
- Job: User build. This is run on the user-provided Docker image.
- Post-job*: Create cache, upload artifacts to GitLab.

*pre-job and post-job are run on a special docker container based on Alpine Linux.

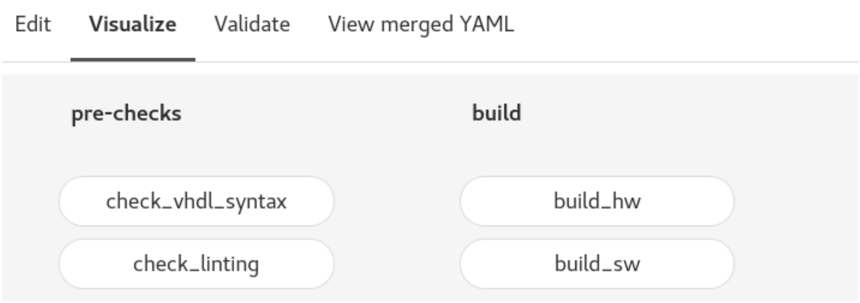
```
Running with gitlab-runner 15.5.1 (7178588d)
  on cs8crome1;ansible-started M1qHTXyW
Resolving secrets
Preparing the "docker" executor
Using Docker executor with image gitlab-registry.cern.ch/cce/docker_build/petalinux:2018.1 ...
Using locally found image version due to "if-not-present" pull policy
Using docker image sha256:b0bfa0820c48e652f034f4c85ac0[redacted]f for gitlab-
t gitlab-registry.cern.ch/cce/docker_build/petalinux@sh[redacted]de59901d9dd6b
Preparing environment
Running on runner-m1qhtxyw-project-27912-concurrent-1 via [redacted]rn.ch...
Getting source from Git repository
Fetching changes...
Reinitialized existing Git repository in /builds/CROME/CROME/.git/
Checking out ab38b2ef as refs/merge-requests/41/head...
Removing CMPU/zynq/hw/aclocal.m4
Removing CMPU/zynq/hw/autom4te.cache/
Removing CMPU/zynq/hw/build/
Removing CMPU/zynq/hw/configure
Skipping Git submodules setup
Downloading artifacts
Downloading artifacts for check_linting (25921706)...
Downloading artifacts from coordinator... ok        id=25921706 responseStatus=200 OK token=ocLhtsa8
Executing "step_script" stage of the job script
```

Prebuilt docker images for Vivado, Petalinux, QuestaSim, ModelSim, Doxygen etc. are available through Gitlab Container Registry at SY-EPC-CCE's repository[1]

[1] SY-EPC-CCE Gitlab container-registry: https://gitlab.cern.ch/cce/docker_build/container_registry

CROME CI Pipeline

CROME CI Pipeline

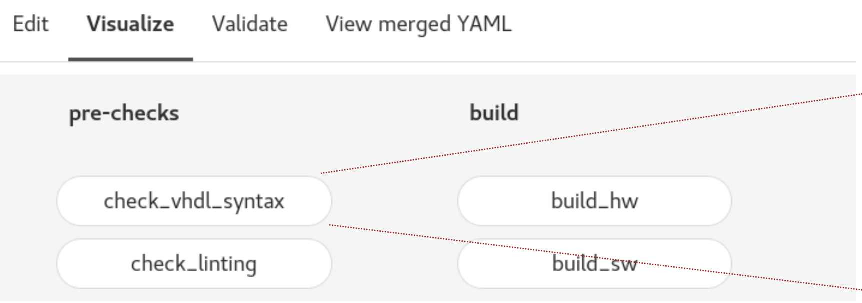



```yaml
stages:          # List of stages for jobs, and their order of execution
  - pre-checks
  - build

workflow:   # dictates pipeline behaviour
  rules:
    - if: $CI_PIPELINE_SOURCE == 'merge_request_event'
```

merge request pipelines runs when a merge request is open for the branch.

CROME CI Pipeline: Syntax Check and Linting

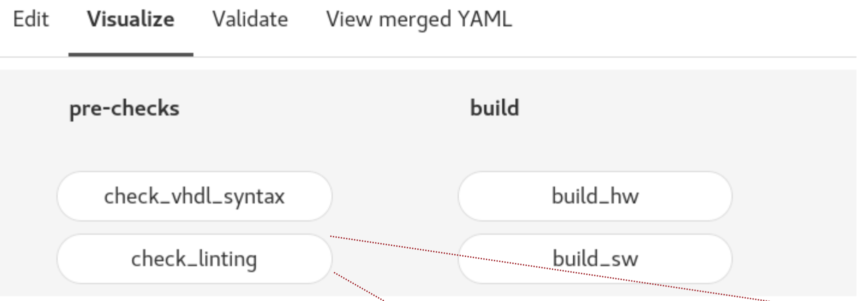
```yaml
stages:                # List of stages for jobs, and their order of execution
  - pre-checks
  - build

workflow:    # dictates pipeline behaviour
  rules:
    - if: $CI_PIPELINE_SOURCE == 'merge_request_event'
```




```yaml
check_vhdl_syntax:
  stage: pre-checks
  image: gitlab-registry.cern.ch/cce/docker_build/vivado:2018.1
  allow_failure: false
  before_script:
    - sudo yum -y install autoconf automake
  script:
    - cd CMPU/zynq/hw/
    - autoreconf -i
    - mkdir -p build
    - cd build
    - ../configure --enable-triplication --enable-frontend=ion --enable-sem=repair
    - make check_syntax
```

before_script: use for initial setup of docker container and resolve missing dependencies. Upon final testing this can be packaged in the original docker image.

CROME CI Pipeline: Syntax Check and Linting




```yaml
stages:          # List of stages for jobs, and their order of execution
  - pre-checks
  - build

workflow:
  rules:
    - if: $CI_PIPELINE_SOURCE == 'merge_request_event'
```

```yaml
check_vhdl_syntax:
  stage: pre-checks
  image: gitlab-registry.cern.ch/cce/docker_build/vivado:2018.1
  allow_failure: false
  before_script:
    - sudo yum -y install autoconf automake
  script:
    - cd CMPU/zynq/hw/
    - autoreconf -i
    - mkdir -p build
    - cd build
    - ../configure --enable-triplication --enable-frontend=ion --enable-sem=repair
    - make check_syntax
```

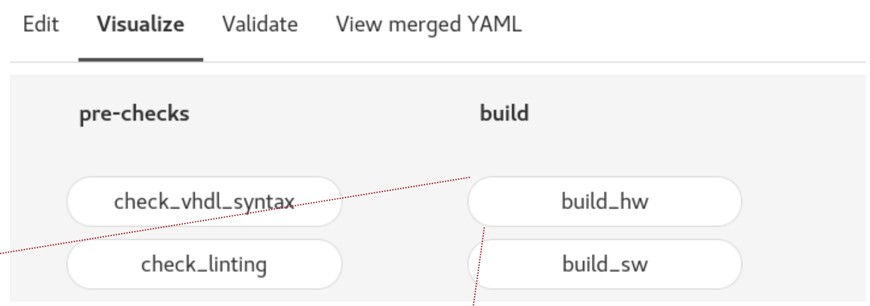
```yaml
check_linting:
  stage: pre-checks
  image: vsg:latest
  allow_failure: false
  script:
    - cd CMPU/zynq/hw/
    - autoreconf -i
    - mkdir -p build
    - cd build
    - ../configure --enable-triplication --enable-frontend=ion --enable-sem=repair
    - make lint
  artifacts:
    when: always
    paths:
    - CMPU/zynq/hw/build/linting/lint_junit.xml
    reports:
      junit: CMPU/zynq/hw/build/linting/lint_junit.xml
      codequality: CMPU/zynq/hw/build/linting/lint_quality_report.xml
```

allow_failure: allows for a CI job to fail and continue executing the pipeline.

artifacts: built files and executables that are passed on to the next job.

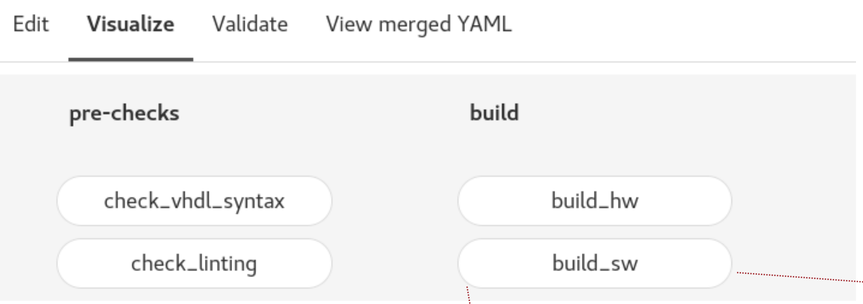
Amitabh Yadav, amitabh.yadav@cern.ch

CROME CI Pipeline: Build Stage




```
build_hw:
  stage: build
  image: gitlab-registry.cern.ch/cce/docker_build/vivado:2018.1
  allow_failure: false
  before_script:
    - sudo yum -y install autoconf automake
  script:
    - cd CMPU/zynq/hw/
    - autoreconf -i
    - mkdir -p build
    - cd build
    - ../configure --enable-triplication --enable-frontend=ion --enable-sem=repair
    - make implementation
    - echo "Vivado 2018.1 Project IMPLEMENTATION completed successfully."
  artifacts:
    paths:
     - CMPU/zynq/hw/build/CMPU.hdf
```

CROME CI Pipeline: Build Stage




```yaml
build_hw:
  stage: build
  image: gitlab-registry.cern.ch/cce/docker_build/vivado:2018.1
  allow_failure: false
  before_script:
    - sudo yum -y install autoconf automake
  script:
    - cd CMPU/zynq/hw/
    - autoreconf -i
    - mkdir -p build
    - cd build
    - ../configure --enable-triplication --enable-frontend=ion --enable-sem=repair
    - make implementation
    - echo "Vivado 2018.1 Project IMPLEMENTATION completed successfully."
  artifacts:
    paths:
      - CMPU/zynq/hw/build/CMPU.hdf
```

```yaml
build_sw:
  stage: build
  image: gitlab-registry.cern.ch/cce/docker_build/petalinux:2018.1
  allow_failure: false
  before_script:
    - sudo yum -y install autoconf automake
  script:
    - source /opt/Xilinx/petalinux/settings.sh
    - cd CMPU/zynq/sw/cromeSuite
    - autoreconf -i
    - mkdir -p build
    - cd build
    - ../configure --host arm-linux-gnueabihf
    - make -j8
    - echo "Embedded_Linus_Userspace_Application and ROMULUSlib Compiled Successfully"
  artifacts:
    paths:
      - CMPU/zynq/sw/ROMULUSlib/libROMULUS.*
      - CMPU/zynq/sw/cromeSuite/build/cromeApp
      - CMPU/zynq/sw/cromeSuite/build/plreset
      - CMPU/zynq/sw/cromeSuite/build/setPCAP
      - CMPU/zynq/sw/cromeSuite/apps/cromeApp/local_parameters.dat
      - CMPU/zynq/sw/cromeSuite/apps/cromeApp/romulus_parameters.dat
      #- CMPU/zynq/sw/embedded_linux_userspace_app/doc
```

Key Features

needs is used needs to execute jobs out-of-order.

```yaml
build_sw:        # This job runs in the build stage, which runs first.
  stage: build
  needs: [sw_linting]
  image: gitlab-registry.cern.ch/cce/docker_build/vivado:2021.2
  allow_failure: false
  variables:
    XILINX_PATH: /opt/Xilinx/
    XILINX_VERSION: "2021.2"
    ROMULUSLIB_MAJOR_VERSION: "7"
    ROMULUSLIB_MINOR_VERSION: "2"
  before_script:
    #- yum -y update
    - yum -y install doxygen
  script:
    - source $XILINX_PATH/Vitis/$XILINX_VERSION/settings64.sh
```

Key Features

needs is used needs to execute jobs out-of-order.

variables can be defined in .gitlab-ci.yml or in the project settings. They can be made global or local to a job and are used in similar way as shell variables.

```yaml
build_sw:        # This job runs in the build stage, which runs first.
  stage: build
  needs: [sw_linting]
  image: gitlab-registry.cern.ch/cce/docker_build/vivado:2021.2
  allow_failure: false
  variables:
    XILINX_PATH: /opt/Xilinx/
    XILINX_VERSION: "2021.2"
    ROMULUSLIB_MAJOR_VERSION: "7"
    ROMULUSLIB_MINOR_VERSION: "2"
  before_script:
    #- yum -y update
    - yum -y install doxygen
  script:
    - source $XILINX_PATH/Vitis/$XILINX_VERSION/settings64.sh
```

Key Features

needs is used needs to execute jobs out-of-order.

variables can be defined in .gitlab-ci.yml or in the project settings. They can be made global or local to a job and are used in similar way as shell variables.

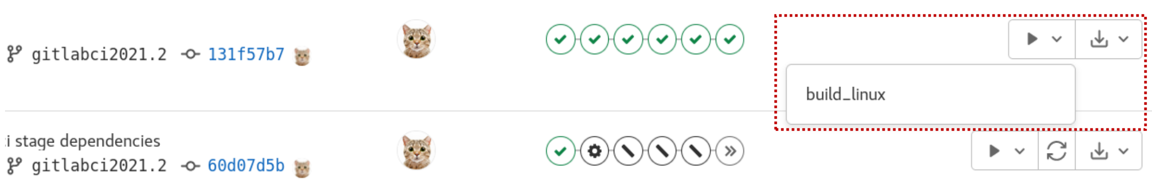
when is used within a job to specify if the job needs manual intervention to start.

- Stages such as formal_verification for license availability reasons.

- Linux image build.

```yaml
build_sw:       # This job runs in the build stage, which runs first.
  stage: build
  needs: [sw_linting]
  image: gitlab-registry.cern.ch/cce/docker_build/vivado:2021.2
  allow_failure: false
  variables:
    XILINX_PATH: /opt/Xilinx/
    XILINX_VERSION: "2021.2"
    ROMULUSLIB_MAJOR_VERSION: "7"
    ROMULUSLIB_MINOR_VERSION: "2"
  before_script:
    #- yum -y update
    - yum -y install doxygen
  script:
    - source $XILINX_PATH/Vitis/$XILINX_VERSION/settings64.sh
```

```yaml
when: manual
allow_failure: true
rules:
  - if: '$CI_COMMIT_BRANCH == "gitlabci2021.2"'
artifacts:
  when: on_success
  paths:
    - images/linux/
```



CROME CI Pipeline: Artifacts

Status	Pipeline	Triggerer	Stages
passed 00:32:33 3 days ago	Dummy commit #4780028 41 ab38b2ef latest merge request		✓ ✓
passed 00:32:58 4 days ago	Dummy commit #4778234 41 4a121f43 merge request		✓ ✓
passed 00:32:45 6 days ago	cosim/README.md: updated prepare_sdcard, minor impr... #4768540 allModifClyde 361a0ca5		✓ ✓
failed 00:33:31 6 days ago	sw: WIP change of file structure #4764416 allModifClyde 898e3ce4		✓ ✗

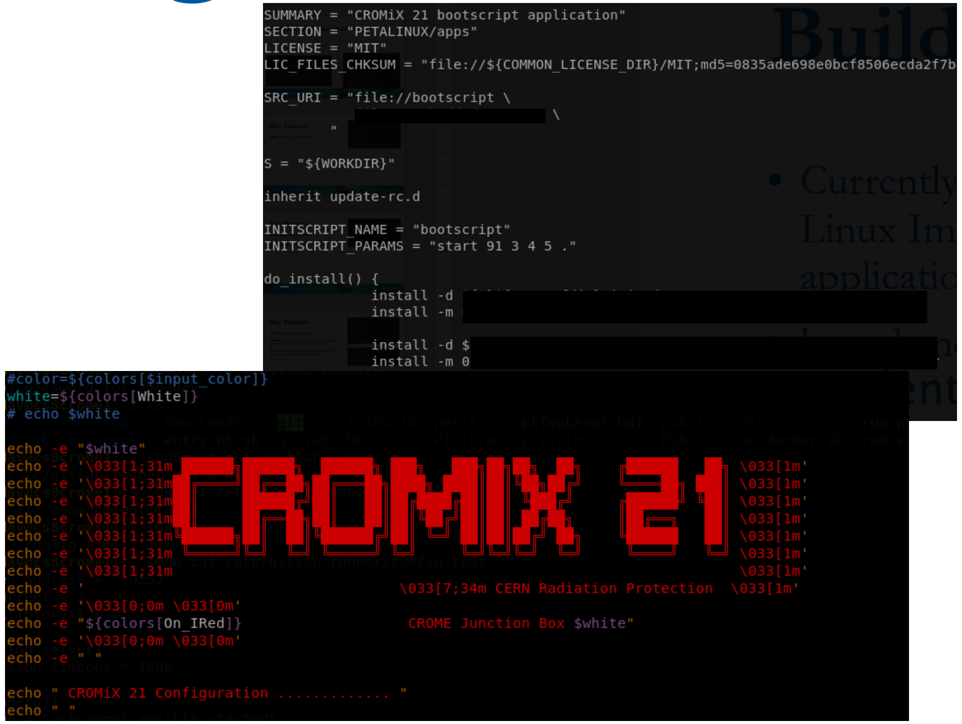
Download artifacts

Search

- build_hw:archive
- build_sw:archive
- check_linting:archive
- check_linting:junit
- check_linting:codequality

Building Linux Images through Gitlab CI

- Currently testing: Building of Embedded Linux Image alongwith bit bake bootscript applications.



Building Linux Images through Gitlab CI

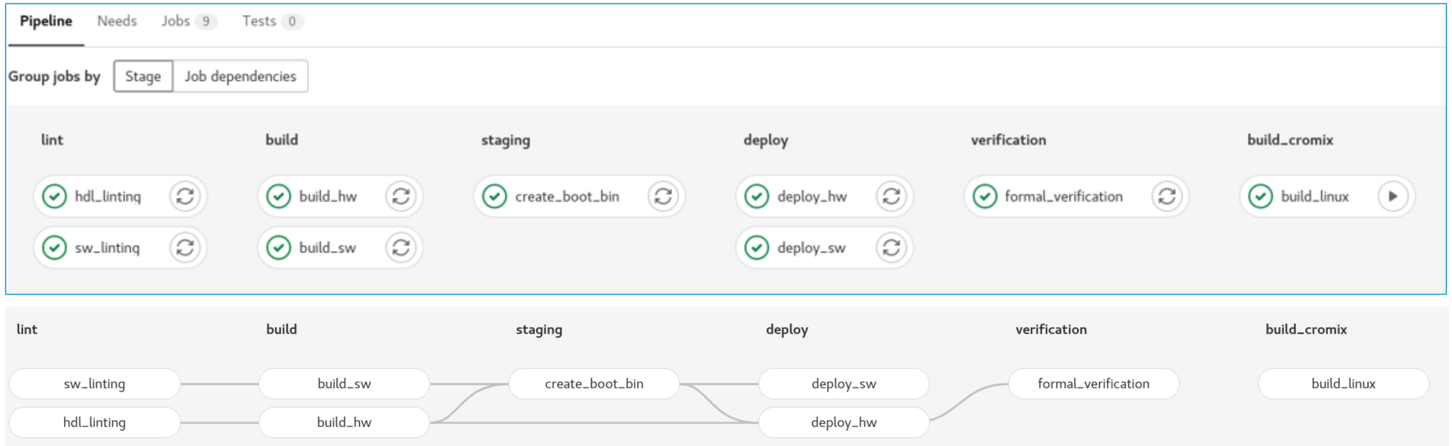
- Currently testing: Building of Embedded Linux Image alongwith `bit bake` bootscript applications.
- kernel and hw configuration through `--silentconfig` option on Xilinx/Petalinux 2021.2

```yaml
build_linux:
    stage: build_cromix
    needs: []
    image: gitlab-registry.cern.ch/cce/docker_build/petalinux:2021.2
    script:
      - echo "Run cromix21 Build Petalinux Image"
      - cd CMPU/zynq/hw
      - petalinux-create --type project --template zynq --name cromix21
      - cd cromix21
      - petalinux-config --get-hw-description ../hdf/ --silentconfig
      - petalinux-config -c kernel --silentconfig
      - petalinux-build
    when: manual
    allow_failure: true
    rules:
      - if: '$CI_COMMIT_BRANCH == "gitlabci2021.2"'
    artifacts:
      when: on_success
      paths:
        - images/linux/
      expire_in: 7d
```

*in development

Conclusion

- CI can be efficiently adopted for heterogeneous development for SoCs.
- We have currently deployed Gitlab CI in development branch is currently used for HDL linting using vsg docker image.
- Gitlab CI is proving elemental during migration of our Xilinx HDL codebase + IPs from version 2018.1 to 2021.2
- Has helped in resolving dependency issues through GNU Autotools and Docker containers to execute CI jobs.



In the works is Gitlab CI for:

- Formal Verification scripts to be added to the CI pipelines to be executed automatically.
- Continuous Deployment through gitlab-runner through ssh into devices.

www.cern.ch