CONTINUOUS INTEGRATION
Hello World

RSE Summer School, September 25, 2024 | Jakob Fritz, Maria Lupe Barrios Sazo, Dirk Brömmel, Robert Speck | Jülich Supercomputing Center, Forschungszentrum Jülich

JÜLICH
Forschungszentrum

Overview

Structure of the day

- Laying the foundations
- Give examples using GitLab and/or GitHub

  ☕ Break
- Test and code quality automation

  🍴 Lunch
- Monitor performance

  ☕ Break
- Various applications

Incorporate what you find interesting. Want to chat about ideas? Talk to us!

YAML
What is it and how to write it

```yaml
Key: Value
FirstKey: String
SecondKey: "String with spaces"
ThirdKey: String # followed by a comment
FourthKey: 1234 # Integer
FithKey: 1234.5 # Float
SixthKey: true # Boolean
AList:
  - firstEntry
  - secondEntry
AnotherList: [firstEntry, 2, 3.0]
NewList:
  - NewDict: 1
    AnotherEntry:
    - NestingDictsAndLists
```

JÜLICH
Forschungszentrum

GitHub & GitLab
GitHub-Actions 🐱

- Add a YAML-file to `.github/workflows/`
- GitHub detects files there and starts Actions
- Multiple YAML-files create multiple pipelines
- Pipelines are independent of each other
- Specify docker-image with keyword „container"

```yaml
name: REUSE Compliance Check
on: [push, pull_request]
jobs:
  reuse_compliance:
    runs-on: ubuntu-latest
    container: alpine:3.17
    steps:
    - uses: actions/checkout@v3
    - name: REUSE Compliance Check
      uses: fsfe/reuse-action@v1
```

JÜLICH
Forschungszentrum

GitHub & GitLab
GitLab-CI

- Add a YAML-file `.gitlab-ci.yml` to top level
- GitLab detects the file and starts pipeline
- Only a single file, only a single pipeline
- Multiple stages to order execution of jobs
- Specify docker-image with keyword „image"

```yaml
reuse:
  stage: test
  image:
    name: fsfe/reuse:latest
    entrypoint: [""]
  script:
    - reuse lint
test_python:
  stage: test
  image: python:3.11-alpine
  before_script:
    - pip install .
  script:
    - coverage run -m pytest .
```

JÜLICH
Forschungszentrum

GitHub & GitLab

Where the CI-files differ

GitHub-Actions

- Add a YAML-file to `.github/workflows/`
- GitHub detects files there and starts Actions
- Multiple YAML-files create multiple pipelines
- Pipelines are independent of each other
- Specify docker-image with keyword „container"
- **Need to check out code explicitly**

GitLab-CI

- Add a YAML-file `.gitlab-ci.yml` to top level
- GitLab detects the file and starts pipeline
- Only a single file, only a single pipeline
- Multiple stages to order execution of jobs
- Specify docker-image with keyword „image"
- Code is checked out automatically

JÜLICH
Forschungszentrum

Artifacts

What they are

- Artifacts can be used to expose files to other jobs and the UI
- Specified per job
- Specify paths and/or filenames or extensions
- Exhibit artifacts only on error, on success or always
- Artifacts are available to jobs at later stages

```yaml
test_python:
  stage: test
  image: mambaorg/micromamba:bullseye-slim
  artifacts:
    when: on_success
    paths:
      - htmlcov/*
    expire_in: "30 days"
  script:
    - coverage run -m pytest .
    - coverage report --show-missing
    - coverage html
```

JÜLICH
Forschungszentrum

Containers

A brief introduction

- *Similar to VMs, keywords are: images, containers, Docker*
- CI pipelines are executed on 'a system', containers define this system, i.e. set up the OS environment the pipelines run in. There may be a default.
- Large 'hub' of available images, one for every use-case, library, programming language, …
- Clearly defined starting point, always identical
- Extend by toolchains/libraries you need

Benefits

- Reproducible, extensible, shareable
- Under your control
- Reuse outside of CI and have identical environment

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers – images become containers when they run on Docker Engine. source: docker.com

docker

podman

JÜLICH
Forschungszentrum

Containers

Quickstart

- Large 'hub' of available images, one for every use-case, library, programming language, …
    - Pick one close to your needs or pick one from the actual project (if available)
    - Default registry often DockerHub
    - CI pipeline will download the same, fresh image and start new container every time
    - Perhaps have different images for compiling, testing, deploying code
- Extend by toolchains and libraries you need
    - Depending on chosen image/OS, install additional software (every time during the pipeline)
    - Or: extend image in first step of pipeline, use in steps that follow
- Define your own image for most control (and speed)
- Ship your images, perhaps even your code within an image

JÜLICH
Forschungszentrum

Containers

Quickstart

- Large 'hub' of ⋯ ⋯ning language, …
  - Pick one cⅼ⋯
  - Default regi⋯
  - CI pipeline ⋯ ry time
  - Perhaps ha⋯
- Extend by toolc⋯
  - Depending ⋯ g the pipeline)
  - Or: extend ⋯
- Define your ow⋯
- Ship your imag⋯

gitlab

```
image:
  name: "debian:stretch-slim"
```

github

```
runs:
  using: 'docker'
  image: 'docker://debian:stretch-slim'
```

JÜLICH
Forschungszentrum

source: hub.docker.com

Containers

Quickstart

- Large 'hub' of available images, one for every use-case, library, programming language, …
    - Pick one close to your needs or pick one from the actual project (if available)
    - Default registry often DockerHub
    - CI pipeline will download the same, fresh image and start new container every time
    - Perhaps have different images for compiling, testing, deploying code
- Extend by toolchains and libraries you need
    - Depending on chosen image/OS, install additional software (every time during the pipeline)
    - Or: extend image in first step of pipeline, use in steps that follow
- Define your own image for most control (and speed)
- Ship your images, perhaps even your code within an image

JÜLICH
Forschungszentrum

Containers

Quickstart

- Large 'hub' of available images, one for every use-case, library, programming language, …
  - Pick one close to your needs or pick one from the actual project (if available)
  - Default r
  - CI pipelin
  - Perhaps
- Extend by to
  - Dependin
  - Or: exten
- Define you
- Ship your images, perhaps even your code within an image

```
 98  dry_run_micromamba:
 99    stage: test
100    image: mambaorg/micromamba:1
101    before_script:
102      - micromamba create -n cla_bot
103      - micromamba install -y -f env.yml
104      - micromamba clean --all --yes
105      - eval "$(micromamba shell hook --shell bash)"
106      - micromamba activate cla_bot
107    script:
```

JÜLICH Forschungszentrum

Containers

Quickstart

- Large 'hub' of available images, one for every use-case, library, programming language, ...
  - Pick one close to your needs or pick one from the actual project (if available)
  - Default registry often DockerHub
  - CI pipeline will download the same, fresh image and start new container every time
  - Perhaps have different images for compiling, testing, deploying code
- Extend by toolchains and libraries you need
  - Depending on chosen image/OS, install additional software (every time during the pipeline)
  - Or: extend image in first step of pipeline, use in steps that follow
- Define your own image for most control (and speed)
  - Ship your images, perhaps even your code within an image

JÜLICH
Forschungszentrum

Containers
Quickstart

- Large 'hub' of available images, one for every use-case, library, programming language, …

```
1  ARG CODE_VERSION=latest
2  FROM ubuntu:${CODE_VERSION}
3  COPY ./examplefile.txt /examplefile.txt
4  ENV MY_ENV_VARIABLE="example_value"
5  RUN apt-get update
6
7  # Mount a directory from the Docker volume
8  # Note: This is usually specified in the 'docker run' command.
9  VOLUME ["/myvolume"]
10
11 # Expose a port (22 for SSH)
12 EXPOSE 22
```

source: wikipedia.org

JÜLICH
Forschungszentrum

Containers

Quickstart

- Large 'hub' of available images, one for every use-case, library, programming language, …
  - Pick one close to your needs or pick one from the actual project (if available)
  - Default registry often DockerHub
  - CI pipeline will download the same, fresh image and start new container every time
  - Perhaps have different images for compiling, testing, deploying code
- Extend by toolchains and libraries you need
  - Depending on chosen image/OS, install additional software (every time during the pipeline)
  - Or: extend image in first step of pipeline, use in steps that follow
- Define your own image for most control (and speed)
- Ship your images, perhaps even your code within an image

JÜLICH
Forschungszentrum

Getting things done

Now, enough talking.
You can use the time until the break to get your first CI-Pipeline running.

JÜLICH
Forschungszentrum

CONTINUOUS INTEGRATION
Essentials

RSE Summer School, September 25, 2024 | Jakob Fritz, Maria Lupe Barrios Sazo, Dirk Brömmel, Robert Speck | Jülich Supercomputing Center, Forschungszentrum Jülich

Overview
Upcoming topics in context

- Important elements to include
- Checks on your source code
- Run tests
- What to execute when?
- Default environment variables

JÜLICH
Forschungszentrum

Linting
What it is and why it is useful

A linter is a small program that checks code for stylistic or programming errors. Linters are available for most syntaxes, from Python to HTML.

The idea is to check the code fast and automatically.
Linters analyze the static code (they do not execute it).
Linters are specific for the language to check but not for the code to check.
It checks for syntactic correctness, not for semantic one.

Pros:

- ➕ Fast to execute
- ➕ No (or little) need to adapt to your code
- ➕ Finds e.g. unused variables, …
- ➕ Check the whole code (not only paths)

Cons:

- ➖ Do not find flaws in your logic
- ➖ Do not check if your results are correct

Definition from SublimeLinter

JÜLICH
Forschungszentrum

Running tests

Running tests in CI has a few advantages:

- Tests are run in a reproducible way, as no manual interaction is needed.
- Tests are run frequently with every push to the git-server (GitHub or GitLab)

How to run tests in CI

1. Add job in your CI-file (`.gitlab-ci.yml` or `.github/workflows/WHATEVER.yml`)
2. Select a suitable (docker-)image for your code
3. For GitHub: checkout your code
4. For compiled languages: compile your code
5. Run tests with a framework of your choice (see talk on testing)

Running tests

Running tests in CI has a few advantages:

- Tests are run in a reproducible way, as no manual interaction is needed.
- Tests are run frequently with every push to the git-server (GitHub or GitLab)

How to run tests in CI

1. Add job in your CI-file (`.gitlab-ci.yml` or `.github/workflows/WHATEVER.yml`)
2. Select a suitable (docker-)image for your code
3. For GitHub: checkout your code
4. For compiled languages: compile your code
5. Run tests with a framework of your choice (see talk on testing)

JÜLICH
Forschungszentrum

Running tests

GitLab ᪲:
Entries in `script` are used in shell directly

```
Run_tests:
  image: alpine:3.17
  before_script:
    - PACKAGE MANAGER COMMAND
    - TO INSTALL DEPENDENCIES
  script:
    - YOUR TESTING COMMAND HERE
```

GitHub ○:
Use `run` in `steps`

```
name: Run tests
jobs:
  tests:
    runs-on: ubuntu-latest
    container: alpine:3.17
    steps:
    - uses: actions/checkout@v3
    - name: Install everything needed
      run: PACKAGE MANAGER COMMAND HERE
    - name: Acutal testing
      run: YOUR TESTING COMMAND HERE
```

JÜLICH
Forschungszentrum

Triggering & Conditionals

It is possible to run a workflow/pipeline after different events: after push, PR/MR, scheduled

- On GitHub, a large number of ready-to-use options exist
    - When issue and/or PR is opened, edited, reopened, closed …etc
    - Keywords: **on, types, branches, paths**
    - Use **workflow_dispatch** to manually run
    - Possible to use multiple events and filter combinations
- Distinct sections can be executed under various circumstances
    - Use of expressions can be helpful

```yaml
on:
  pull_request:
    types:
      - opened
    branches:
      - 'releases/**'
    paths:
      - '**.js'
```

```yaml
steps:
- name: if_issue
  if: github.event.issue
  run: |
```

JÜLICH
Forschungszentrum

- On GitLab, keyword: **rules**
  - Can be run manually
  - Possible to use different conditions per job
  - Many rules and combinations: **if, when, changes, exist, needs, allow_failure** …

```yaml
docker build:
  script: docker build -t my-image:$CI_COMMIT_REF_SLUG .
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
      changes:
        - Dockerfile
      when: manual
      allow_failure: true
```

- You are able to configure entire pipeline with **workflow** keyword
- Worthwhile to take a look at predefined CI/CD variables

```yaml
workflow:
  rules:
    - if: $CI_PIPELINE_SOURCE == 'merge_request_event'
    - if: $CI_COMMIT_TAG
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

JÜLICH
Forschungszentrum

Environment variables
GitLab

- Can be available when pipeline is created or when runner runs the job

- Depending on which event starts the pipeline **CI_PIPELINE_SOURCE** gets value, e.g. **merge_request_event, push, web**

Variable	Defined for	GitLab	Runner	Description
`CI_BUILDS_DIR`	Jobs only	all	11.10	The top-level directory where builds are executed.
`CI_COMMIT_AUTHOR`	Pipeline	13.11	all	The author of the commit in `Name <email>` format.
`CI_COMMIT_BEFORE_SHA`	Pipeline	11.2	all	The previous latest commit present on a branch or tag. Is always `0000000000000000000000000000000000000000` for merge request pipelines, the first commit in pipelines for branches or tags, or when manually running a pipeline.
`CI_COMMIT_BRANCH`	Pipeline	12.6	0.5	The commit branch name. Available in branch pipelines, including pipelines for the default branch. Not available in merge request pipelines or tag pipelines.
`CI_COMMIT_DESCRIPTION`	Pipeline	10.8	all	The description of the commit. If the title is shorter than 100 characters, the message without the first line.
`CI_COMMIT_MESSAGE`	Pipeline	10.8	all	The full commit message

JÜLICH
Forschungszentrum

Environment variables
GitHub ⚙

- Default environment variables exist only when runner runs the job
- Contexts available for workflow configuration before job is directed to runner
  - Contexts are objects at hand to access information about runs, variables, jobs, environment
  - To mention a few: GitHub, job, runner
  - Can use with expressions syntax, and functions, e.g. contains, toJSON
- Most default variables have an analogous context property

Property name	Type	Description
github	object	The top-level context available during any job or step in a workflow. This object contains all the properties listed below.
github.action	string	The name of the action currently running, or the `id` of a step. GitHub removes special characters, and uses the name `__run` when the current step runs a script without an `id`. If you use the same action more than once in the same job, the name will include a suffix with the sequence number with underscore before it. For example, the first script you run will have the name `__run`, and the second script will be named `__run_2`. Similarly, the second invocation of `actions/checkout` will be `actionscheckout2`.
github.action_path	string	The path where an action is located. This property is only supported in composite actions.

```yaml
jobs:
  prod-check:
    if: ${{ github.ref == 'refs/heads/main' }}
    runs-on: ubuntu-latest
    steps:
      - run: echo "Deploying to production server on branch $GITHUB_REF"
```

JÜLICH
Forschungszentrum

Getting things done

Now, enough talking.
You can use the time until the break to add tests to your pipeline.

JÜLICH
Forschungszentrum

CONTINUOUS INTEGRATION
Next Steps

RSE Summer School, September 25, 2024 | Jakob Fritz, Maria Lupe Barrios Sazo, Dirk Brömmel, Robert Speck | Jülich Supercomputing Center, Forschungszentrum Jülich

JÜLICH
Forschungszentrum

Overview
Upcoming topics in context

- Other possible components
- Set up customized containers for your needs
- Keep track of performance
- More on variables and handling sensitive info
- Deploy websites and release code versions

JÜLICH
Forschungszentrum

Containers

Building & storing them
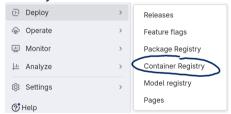
- Heard earlier: interesting to run pipelines with dedicated containers
$\longrightarrow$ Next step is building your own, custom containers

Will not discuss Dockerfiles, instead focus on how to integrate with CI

GitHub offers an easy start: using a Dockerfile instead of an image

```
# action.yml
name: 'Hello World'
description: 'Greet someone and record the time'
inputs:
  who-to-greet:  # id of input

outputs:
  time: # id of output
    description: 'The time we greeted you'
runs:
  using: 'docker'
  image: 'Dockerfile'
  args:
    - ${{ inputs.who-to-greet }}
```

GitLab offers per project container registry that is easily accessible

▷ Deploy	>	Releases
◈ Operate	>	Feature flags
▣ Monitor	>	Package Registry
⊫ Analyze	>	Container Registry
⚙ Settings	>	Model registry
⑦ Help		Pages

JÜLICH
Forschungszentrum

Containers

Building & storing them

Running Docker (building an image) inside Docker (in CI) is difficult, may require additional permissions. One suggested, easy way round: do not use Docker directly, us `kaniko`:

```
# Create new image from a changed Dockerfile
docker_image:
  stage: docker
  image:
    name: gcr.io/kaniko-project/executor:debug
    entrypoint: [""]
  script:
    - mkdir -p /kaniko/.docker
    - |
      echo "{\"auths\":{\"$CI_REGISTRY\":{\"username\":\"$CI_REGISTRY_USER\",\"password\":\"$CI_REGISTRY_PASSWORD\"}}}" \
      > /kaniko/.docker/config.json
    - |
      /kaniko/executor --context $CI_PROJECT_DIR \
      --dockerfile $CI_PROJECT_DIR/Dockerfile \
      --destination $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_SLUG:$CI_COMMIT_SHORT_SHA \
      --destination $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_SLUG:latest
  rules:
    # Run when variable is set
    - if: $FORCE_DOCKER_IMAGE == "true"
      when: always
    # Run when committing to a branch or merging and changing Dockerfile
    - if: $CI_COMMIT_BRANCH || $CI_PIPELINE_SOURCE == "merge_request_event"
      changes:
        - Dockerfile
      when: always
```

JÜLICH
Forschungszentrum

Continuous Benchmarking
What is it?
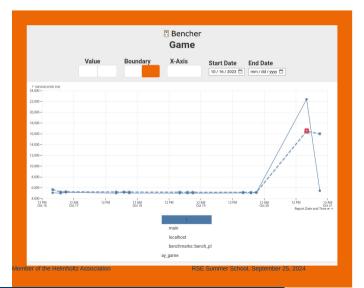
The title gives it away: **continuous monitoring of your code's performance**

$\longrightarrow$ Running automatic benchmarks from the CI pipeline

This does not need to be a benchmark for simulation software, any application is speed 'critical'

JÜLICH
Forschungszentrum

Continuous Benchmarking

What is it?

How It Works

Run your benchmarks

Run your benchmarks locally or in CI using your favorite benchmarking tools. The `bencher` CLI simply wraps your existing benchmark harness and stores its results.

Track your benchmarks
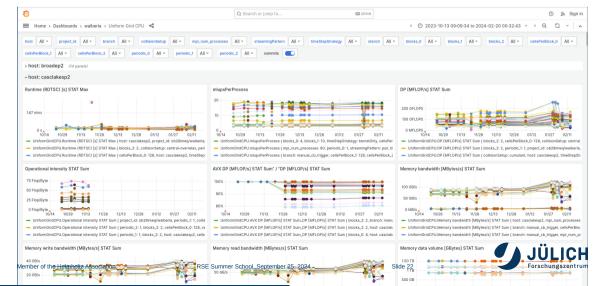
Track the results of your benchmarks over time. Monitor, query, and graph the results using the Bencher web console based on the source branch and testbed.

Catch performance regressions

Catch performance regressions in CI. Bencher uses state of the art, customizable analytics to detect performance regressions before they make it to production.

JÜLICH
Forschungszentrum

Continuous Benchmarking

What is it?

Continuous Benchmarking

Why should I use it?

- Track evolution of performance due to code changes
- Track evolution of performance due to system changes
$\longrightarrow$ Act before users complain about slowdowns
  If you don't know you have a problem, you cannot take action to fix it
- Show users what performance to expect (and how)
- Test automatically on different platforms
- Reduce burden on developers by separating concerns

JÜLICH
Forschungszentrum

Environment variables

Ease their usage, make them available

Sooner or later: too many variables to remember, though some may control your pipeline behaviour
$\longrightarrow$ describe them and make them show up with pull-down option list for manual triggers

To get

Include in `.gitlab-ci.yml`

JÜLICH
Forschungszentrum

Environment variables

Ease their usage, make them available

Sooner or later: too many variables to remember, though some may control your pipeline behaviour
⟶ des

To ge

Run pipeline

Run for branch name or tag

`master ∨`

Variables

| Variable ∨ | FORCE_DOCKER_IMAGE | false ∨ |

Force the creation of a new Docker image for this branch.

Q Search values

✓ false

true

| Variable ∨ | SBATCH_ACCOUNT | |

Account name of compute project to use for JUBE runs

JÜLICH
Forschungszentrum

Environment variables

Ease their usage, make them available

Sooner or later: too many variables to remember, though some may control your pipeline behaviour
⟶ describe them and make them show up with pull-down option list for manual triggers

To ge

```
10 ∨ variables:
11 ∨   FORCE_DOCKER_IMAGE:
12       value: "false"
13 ∨     options:
14         - "false"
15         - "true"
16       description: "Force the creation of a new Docker image for this branch."
17 ∨   SBATCH_ACCOUNT:
18       # use this variable for SBATCH/JUBE to define accounting (or hard code within JUBE file)
19       value: ▓▓▓▓
20       description: "Account name of compute project to use for JUBE runs"
```

JÜLICH
Forschungszentrum

Secrets
Variables you don't want to tell everyone

A secret is anything that you want to tightly control access to, such as API keys, passwords, certificates, and more.

- Any variable defined for a CI pipeline is accessible, can be echoed
- GitHub knows 'variables' and 'secrets', GitLab defines a 'visibility'
  $\longrightarrow$ omitted from log files or the output of your pipelines
- **Secrets may still be leaked by changes to the CI pipeline** $\longrightarrow$ **check any changes**
- Services and tools promise to help with secrets management, e.g. Hashicorp Vault, Infiscal, `git-secret`
  - Manage secrets centrally, also for a team
  - Control access to secrets

A private repository will not ensure secrets are kept safe.

🔗 blog.gitguardian.com/secrets-api-management

Secrets
Variables you don't want to tell everyone

A secret is a ... *potential* ... *words,*
certificates, an ...

- Any vari...
- GitHub kn...
  $\longrightarrow$ omitte...
- **Secrets** ... **nges**
- Services a... /ault, Infiscal,
  git-se...
  - Mana...
  - Contr...

A private repos...

JÜLICH
Forschungszentrum

Secrets

Variables you d...

A secret is any... ...sswords, certificates, an...

- Any vari...
- GitHub kn...
  → om...
- Secrets ...
- Services a...
  git-se...
  - Mana...
  - Cont...

A private repo...

Artifacts

A job artifact is an archive of files and directories saved by a job when it finishes.

Variables

Variables store information that you can use in job scripts. Each project can define a... more.

Variables can be accidentally exposed in a job log, or maliciously sent to a third party... can help reduce the risk of accidentally exposing variable values, but is not a guara... from accessing variables. How can I make my variables more secure?

Variables can have several attributes. Learn more.

- **Protected:** Only exposed to protected branches or protected tags.
- **Masked:** Hidden in job logs. Must match masking requirements.
- **Expanded:** Variables with $ will be treated as the start of a reference to anoth...

CI/CD Variables

Key ↑	Value	Environ...
	There are no variables yet.	

Group variables (inherited)

These variables are inherited from the parent group.

CI/CD Variables

Key	Environments
	There are no variables yet.
	There are no variables yet.

Add variable ×

Type
Variable (default)

Environments ⓘ
All (default)

Visibility
- ● Visible
  Can be seen in job logs.
- ○ Masked
  Masked in job logs but value can be revealed in CI/CD settings. Requires values to meet regular expressions requirements.

Flags ⓘ
- ☐ Protect variable
  Export variable to pipelines running on protected branches and tags only.
- ☑ Expand variable reference
  $ will be treated as the start of a reference to another variable.

Description (optional)

The description of the variable's value or usage.

Key

You can use CI/CD variables with the same name in different places, but the variables might overwrite each other. What is the order of precedence for variables?

Value

Monitor ›
Analyze ›
Settings ⌄
  General
  Integrations
  Webhooks
  Access Tokens
  Repository
  Merge requests
  CI/CD
  Packages and registries
  Monitor
  Usage Quotas

JÜLICH Forschungszentrum

Secrets

Variables you don't want to tell everyone

A secret is anything that you want to tightly control access to, such as API keys, passwords, certificates, and more.

- Any variable defined for a CI pipeline is accessible, can be echoed
- GitHub knows 'variables' and 'secrets', GitLab defines a 'visibility'
  $\longrightarrow$ omitted from log files or the output of your pipelines
- **Secrets may still be leaked by changes to the CI pipeline $\longrightarrow$ check any changes**
- Services and tools promise to help with secrets management, e.g. Hashicorp Vault, Infiscal, `git-secret`
  - Manage secrets centrally, also for a team
  - Control access to secrets

A private repository will not ensure secrets are kept safe.

$\mathcal{O}$`blog.gitguardian.com/secrets-api-management`

Deployment
How to deploy websites (pages)

- You can use GitHub pages to build and host project, user, and organization websites
- Site can be published by changes pushed to specific branch which holds static generated website, or with a workflow that holds the site content as an artifact
- In the repository settings one should specify further details in Pages options
- In the branch case, when push is done in a development branch one can automatically build the site and push content on the gh-pages branch, actions-gh-pages can help
- In GitLab, Pages option enabled in Settings, General, Visibility, project, features, permissions.
- Use specific **pages** job in `.gitlab-ci.yml`, artifact with content in folder named **public**

JÜLICH
Forschungszentrum

GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at https://guadabsb15.github.io/summer_2021_report/   Visit site   ...

Build and deployment

Source

Deploy from a branch ▼

Branch

Your GitHub Pages site is currently being built from the /docs folder in the gh-pages branch. Learn more about configuring the publishing source for your site.

⌥ gh-pages ▼   📁 /docs ▼   Save

Learn how to add a Jekyll theme to your site.

Deployments 500+

✓ github-pages 5 days ago

+ more deployments

Merge requests

🔧 Manage   ›
📋 Plan   ›
</> Code   ›
⚙ Build   ›
🛡 Secure   ›
🚀 Deploy   ›
⚙ Operate   ›
📈 Monitor   ›
📊 Analyze   ›
⚙ Settings   ⌄
    General
    Integrations

Model registry

Manage machine learning models. Learn more.

⬤ Everyone With Access

Pages

With GitLab Pages you can host your static websites on GitLab. GitLab Pages uses a caching mechanism for efficiency. Your cache is invalidated, which usually takes less than a minute. Learn more.

⬤

Monitor

Monitor the health of your project and respond to incidents.

⬤ Everyone With Access

Environments

Every project can make deployments to environments either via CI/CD or API calls. Non-project members have read-only acc

⬤ Everyone With Access

Feature flags

Roll out new features without redeploying with feature flags. Learn more.

📄 README
🔧 CI/CD configuration
🔗 GitLab Pages

JÜLICH
Forschungszentrum

Deployment
Release code versions

- Some of this points will be cover later in the software publication session
- It is possible to use workflows and pipelines to do code releases
- An option is to trigger workflow/pipeline when a tag is created
- Release can include comments with changes and links to binaries
- During code releases some of the options to explore include
  - Create or update (Digital Object Identifier) DOI in data archiving tool like Zenodo, with GitHub integration, Hermes
  - Save artifacts and upload, link to assets
  - Automate packaging
  - Publish python code to package repo e.g. TestPyPI, PyPI

JÜLICH
Forschungszentrum

Getting things done
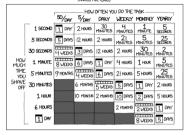
Now, enough talking.
You can use the time until the break to extend your CI.

JÜLICH
Forschungszentrum

source: xkcd.com

CONTINUOUS INTEGRATION
Outlook

RSE Summer School, September 25, 2024 │ Jakob Fritz, Maria Lupe Barrios Sazo, Dirk Brömmel, Robert Speck │ Jülich Supercomputing Center, Forschungszentrum Jülich

JÜLICH
Forschungszentrum

Overview
Upcoming topics in context

- Miscellaneous ideas
- Couple pipelines
- Need more freedom and control of runners?
- Verify commit author
- Automate potential agreement by contributors
- Host websites
- Create LaTeX documents
- Include features/tools from other projects
- Combine with team chat apps
- Syncing repositories

JÜLICH
Forschungszentrum

Coupling pipelines
In GitLab ⅄

Multiple pipelines can be coupled using the `trigger` keyword.

The triggered pipelines can be in the same repository or in another repository. This is then called parent-child or multi-project pipeline

Parent-child pipelines help to structure jobs and to run multiple similar pipelines from a single file.

Multi-project pipelines can help if multiple repositories need to work together (e.g. in microservices).

JÜLICH
Forschungszentrum

Coupling pipelines
In GitLab 🦊

```
.gitlab-ci.yml

...
echo_downstream:
  trigger:
    include:
      - local: downstream.yml

...
```

```
downstream.yml

stages:
  - build
  - test
mimic_build:
  stage: build
  image: alpine:latest
  script:
    - echo "This mimics a build job"
```

JÜLICH
Forschungszentrum

Coupling pipelines
In GitLab 🦊

```
.gitlab-ci.yml

...
echo_downstream:
  trigger:
    include:
      - local: downstream.yml

...
```

```
downstream.yml

stages:
  - build
  - test
mimic_build:
  stage: build
  image: alpine:latest
  script:
    - echo "This mimics a build job"
```

JÜLICH
Forschungszentrum

Coupling pipelines
In GitLab & GitHub

- Triggering pipelines via the API is possible.
- Checking status and downloading artifacts can also be done via API.

This makes it possible to couple pipelines/workflows. E.g. waiting in GitHub for another workflow to finish in order to combine results from both workflows.

JÜLICH
Forschungszentrum

Coupling pipelines

Syncing GitHub and GitLab

Combine the benefits of GitHub (large community and visibility) with the benefits of GitLab (possibility of self-hosted instances and long history of self-hosted runners)

🔗 https://github.com/jakob-fritz/github2lab_action

Coupling pipelines

Syncing GitHub and GitLab

Combine the benefits of GitHub (large community and visibility) with the benefits of GitLab (possibility of self-hosted instances and long history of self-hosted runners)
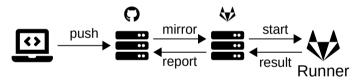

🔗 https://github.com/jakob-fritz/github2lab_action

Using your own runners

For special usecases

GitHub and GitLab offer runners to projects. On self-hosted GitLab-Instances, the admins can provide runners to all projects of that instance.

Furthermore, own runners can be set up and registered at the platform (GitHub or GitLab).

Pros:

- ➕ Known what hardware is used for the runners and if/when they change
- ➕ Can be decided, if multiple jobs can run in parallel on the hardware

Cons:

- ➖ Additional work for set-up and maintenance
- ➖ Often, hardware architecture is not relevant in containers

Bottom line:

- It is possible to have runners, that are only available to a group or a single project
- May be helpful when doing performance analysis or when needing resources not provided by other runners
- If not needed: save yourself the effort of creating and maintaining

JÜLICH Forschungszentrum

What CI can also be used for
Automizing repetitive tasks

Also for repetitive tasks, the benefits of CI hold true. These include repeatability and explicit documentation of how processes are done.

CI can therefore also be used for (small) jobs that occur in a specific frequency. This can include updates to a Database, updates of a website, …

The downside of this approach is, that the environment needs to be created new every time, as you use a new container each run.

Also be aware, that you use external services/computers, so be nice with the usage (regarding duration and frequency of execution).

An example is a database and website, I update every month.

JÜLICH
Forschungszentrum

What CI can also be used for

Hosting websites

In Addition to hosting a website with the documentation of your code, GitHub and GitLab can also be used to host websites, that are not documentation.

This approach offers a few advantages:

- Version controlled website (You can track and roll-back changes made)
- Collaborative creation/updates with multiple people and even PRs/MRs
- No need to take care of hosting yourself
- Known system (you already know how to use git and GH, GL)

However, when using GitLab, the admins of the instance need to have that feature enabled (called pages). The Helmholtz Codebase has this feature enabled.

An example is the HiRSE_PS Website that is maintained in a GitHub repository.

JÜLICH
Forschungszentrum

What CI can also be used for

Compiling LaTeX

Versioning LaTeX in git can be a good idea to know when something changed and keep the previous versions.

Also, CI of the platforms (GitHub & GitLab) can compile the document. This helps, as you know when some change broke your document. Also, the document is build the same way regardless, who made changes. The compiled document can be made available as artifact.

This presentation is an example for this approach.

Furthermore, linting the document is possible.

JÜLICH
Forschungszentrum

Usage of Git(Hub|Lab)
Signing commits

Commit authors are defined in a project's (or global) `git` settings and can be chosen at will

$\longrightarrow$ Thus no reliable information

$\longrightarrow$ Do not use to judge security implications by this

Signing commits will verify who committed what, a CI pipeline can be used to check that.

- `git` supports signing either via GPG, `x.509`, or `ssh`
- Requires settings for `git` (set up of format and key, allowed keys, trust)
- Requires providing matching information to Git(Hub|Lab) (upload keys)

JÜLICH
Forschungszentrum

Usage of Git(Hub|Lab)
Signing commits

Commit author and committer can be changed at will

$\longrightarrow$ Thus no real ...

$\longrightarrow$ Do not use ...

Signing commits ... that.

- git supports ...
- Requires ...
- Requires ...

Signed with GPG
Dirk Brömmel authored 21 hours ago — Verified — d68e2592

Signed with ssh
Dirk Brömmel authored 21 hours ago — Verified — c422a0a0

Signed w/ x.509
Dirk Brömmel authored 21 hours ago — Verified — 75114cab

Verified commit

This commit was signed with a verified signature and the committer email was verified to belong to the same user.

GPG Key ID: 728DC7...

Learn about signing commits

Verified

Verified commit

This commit was signed with a verified signature and the committer email was verified to belong to the same user.

SSH key fingerprint:
xic8b2q3BJ4aqsEfu...
vcwZ85GY

Learn about signing commits with SSH keys.

Verified

This commit was signed with a verified signature and the committer email is verified to belong to the same user.

Certificate Subject
- CN=Dirk Broemmel
- OU=JSC
- O=Forschungszentrum Juelich GmbH
- Subject Key Identifier:
  75 A5 38 A6 03 5E 86 EB B9 27 00 C2 CD E5 B7 6B 1F 1D 5E A3

Certificate Issuer
- CN=DFN-Verein Global Issuing CA
- OU=DFN-PKI
- O=Verein zur Foerderung eines Deutschen Forschungsnetzes e. V.
- Subject Key Identifier:
  6B 3A 98 8B F9 F2 53 89 DA E0 AD B2 32 1E 09 1F E8 AA 3B 74

Learn more about X.509 signed commits

Verified

JÜLICH
Forschungszentrum

Usage of Git(Hub|Lab)

Signing commits

Commit auth... ...en at will

$\longrightarrow$ Thus no r...

$\longrightarrow$ Do not us...

Signing comm... ...that.

- `git` supp...
- Require...
- Require...


```
commit d68e25923d9ead286ce71dffb66bb8245bbf8363
gpg: keyserver option 'honor-http-proxy' is unknown
gpg: Signature made Tue 13 Aug 2024 16:29:24 CEST
gpg:                using RSA key 30EDB907C8601385C
gpg: Good signature from "Dirk Broemmel <d.broemmel@fz-juelich.de>" [ultimate]
Author: Dirk Brömmel <d.broemmel@fz-juelich.de>
Date:   Tue Aug 13 16:29:24 2024 +0200

    Signed with GPG

commit c422a0a0b0d13d238460dbb7fdcfc6698521a6d6
Good "git" signature for d.broemmel@fz-juelich.de with ED25519 key SHA256:x1c8b2q3BJ4aq
Author: Dirk Brömmel <d.broemmel@fz-juelich.de>
Date:   Tue Aug 13 16:22:44 2024 +0200

    Signed with ssh

commit 75114cab1835b5ce0818563fff139a2a21d95ec7
gpgsm: Signature made 2024-08-13 14:19:28 UTC
gpgsm:                using rsa4096 key D4EFB3A1D7ED65946F4DD12B53FB5D4A3E0322D2
gpgsm: Note: non-critical certificate policy not allowed
gpgsm: Note: non-critical certificate policy not allowed
gpgsm: Note: non-critical certificate policy not allowed
gpgsm: Good signature from "/CN=Dirk Broemmel/OU=JSC/O=Forschungszentrum Juelich GmbH/C=DE/GN=Dirk/SN=Broemmel"
gpgsm:                aka "d.broemmel@fz-juelich.de"
Author: Dirk Brömmel <d.broemmel@fz-juelich.de>
Date:   Tue Aug 13 16:19:28 2024 +0200

    Signed w/ x.509
```

JÜLICH
Forschungszentrum

Usage of Git(Hub|Lab)

CLAs & DCOs ✹

Your code has a(n outgoing) license and you have (external) contributors. Now what?

$\longrightarrow$ You might want to ensure you can include content and keep your license.

This is about ensuring necessary rights are transferred to your project.

- CLA: Contributor License Agreement (an incoming license)
- DCO: Developer Certificate of Origin

Either requirement needs to be included/stated in repo.

🔗contributoragreements.org/ca-cla-chooser/   🔗developercertificate.org/

JÜLICH
Forschungszentrum

Usage of Git(Hub|Lab)

CLAs & DCOs ✿

Choice may depend on your code's (future?) license. It may depend on whether you are (really!) dealing with individuals or institutions. To quote:

- CLAs are more formal and legally binding, while DCOs are more informal and rely on the community's trust and good faith.
- CLAs require contributors to sign a legal agreement, which can be a barrier to entry for some contributors.
- DCOs, on the other hand, require only a simple sign-off line, which is easier for contributors to understand and comply with.

*ᴼ*osr.finos.org/docs/bok/artifacts/clas-and-dcos

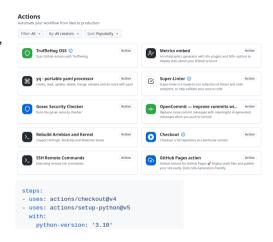IANAL…see your trusted lawyer for advice, speak to your institution

Management tooling does exist (mostly on GitHub, see us for GitLab)**. Merge request may be conditional on CLA/DCO being present.**

Integration and not reinventing the wheel

Market Place in GitHub

- Find countless actions, developed by others, to include in your workflow.
- You can add them in your workflows with keyword **uses**
- For specifying parameters **with**
- Examples that you might already use, and can be helpful:
  - checkout, **actions/checkout@v4**
  - python, setup-python
  - auto commit, git-auto-commit
- You can also create your own actions!

Actions
Automate your workflow from idea to production

Filter: All ▾   By: All creators ▾   Sort: Popularity ▾

TruffleHog OSS ⊘ Action Scan Github Actions with TruffleHog	**Metrics embed** Action An infographics generator with 40+ plugins and 300+ options to display stats about your GitHub account
yq - portable yaml processor Action create, read, update, delete, merge, validate and do more with yaml	**Super-Linter** ⊘ Action Super-linter is a ready-to-run collection of linters and code analyzers, to help validate your source code
Gosec Security Checker Action Runs the gosec security checker	**OpenCommit — improve commits wi...** Action Replaces lame commit messages with meaningful AI-generated messages when you push to remote
Rebuild Armbian and Kernel Action Support Amlogic, Rockchip and Allwinner boxes	**Checkout** ⊘ Action Checkout a Git repository at a particular version
SSH Remote Commands Action Executing remote ssh commands	**GitHub Pages action** Action GitHub Actions for GitHub Pages ✔️ Deploy static files and publish your site easily. Static-Site-Generators-friendly

```
steps:
- uses: actions/checkout@v4
- uses: actions/setup-python@v5
  with:
    python-version: '3.10'
```

JÜLICH
Forschungszentrum

Integration and not reinventing the wheel

Integrating external YAML-files in GitLab-CI

- Don't repeat yourself, include `.yml` files created by you or others
- Add **include** with project, remote, local, template, depending on where the files are located
- Can utilize or create components in a CI/CD catalog, with possibility of specifying parameters and share within GitLab instance
- Supports the use of inputs and rules

```
include:
  - component: $CI_SERVER_FQDN/$CI_PROJECT_PATH/jube-benchmark
    inputs:
      stage: continuous_benchmarking
      benchmark_system: jusuf
      compute_project: ctsma
```

```
include:
  # stages needed: "test" and later ".post"
  # Add as URL to work with other Gitlab-Instances than Jugit
  - remote:
      "https://jugit.fz-juelich.de/j.fritz/continuous-integration/-/raw/main/linter.gitlab-ci.yml"
  - remote:
      "https://jugit.fz-juelich.de/j.fritz/continuous-integration/-/raw/main/sast.gitlab-ci.yml"
  - template: "Security/Secret-Detection.gitlab-ci.yml"
```

JÜLICH
Forschungszentrum

Integration and not reinventing the wheel
Integration into messengers

- Some groups collaborate through team chat applications like slack, rocketchat, matrix
- Getting notifications from GitHub or GitLab activity can be practical
- Feasible to receive details about CI success or failures
- Chat platforms provide integrations
- For more customized reports:
  - gh actions that provide support like action-slack-notify or slack-GitHub-action
  - Needs a token or webhook for permission to post on the chat
  - Interact via curl

JÜLICH
Forschungszentrum

Integration and not reinventing the wheel

Integration into

- Some gr
- Getting n
- Feasible t
- Chat platfo
- For more
  - gh act
  - Need
  - Interac

Combining Repositories

Git submodules

- In Git it is possible to have a repository within a repository while keeping commits separated
- A potential way to handle libraries, dependencies
- In GitHub workflows, one can build project with explicit call to `git submodule update --init` or option with checkout action
- In GitLab pipelines, one can set variable GIT_SUBMODULE_STRATEGY to recursive
- Access tokens if content from other non-public repo needs to be used
- CI can also be used to update submodule

```
- name: Get submodules
  run: |
    git submodule update --init
    cd external/Microphysics
    git fetch; git checkout development
    cd ../amrex
    git fetch; git checkout development
```

```
variables:
  GIT_SUBMODULE_STRATEGY: recursive
```

JÜLICH
Forschungszentrum

That's it

Thank you for your attention. Feel free to ask questions now and later this week!

JÜLICH
Forschungszentrum

✓ Passed

⏱ 06:00:00
📅 just now

CONTINUOUS INTEGRATION
Goodbye

RSE Summer School, September 25, 2024 | Jakob Fritz, Maria Lupe Barrios Sazo, Dirk Brömmel, Robert Speck | Jülich Supercomputing Center, Forschungszentrum Jülich

JÜLICH
Forschungszentrum