Managing ML projects with CI/CD automation

D. Ciangottini, INFN

This is a take-home-messagge session

Papermill

Papermill is a tool for parameterizing, executing, and analyzing Jupyter Notebooks.
Papermill lets you:

- parameterize notebooks
- execute notebooks
  You can programmatically execute a workflow without having to copy and paste from notebook to notebook manually
  To parameterize your notebook designate a cell with the tag parameters:
- Papermill looks for the parameters cell and treats this cell as defaults for the parameters passed in at execution time. Papermill will add a new cell tagged with injected-parameters with input parameters in order to overwrite the values in parameters. If no cell is tagged with parameters the injected cell will be inserted at the top of the notebook.

https://github.com/nteract/papermill

Why should I automate the way I build and distribute my model ?

Reproducibility and Accuracy
Automating the build and distribution of your models ensures that experiments can be consistently reproduced

Efficient Resource Management
efficiently managing computational resources by scheduling and executing tasks without manual intervention, allowing to focus on analysis and interpretation.

Scalability of Experiments
With automated pipelines, you can easily scale your experiments to handle larger datasets and more complex models

Why do we need CI?

Consider an application that has its **code stored in a Git repository in GitLab -->** Developers push code changes every day, multiple times a day.

For every push to the repository, you can create a set of scripts to build and test your application automatically.

This practice is known as Continuous Integration.
Each change submitted to an application, even to development branches, is built and tested automatically and continuously.

These tests ensure the changes pass all tests, guidelines, and code compliance standards you established for your application.

When do you need CD?

Continuous Delivery is a step beyond Continuous Integration.

Not only is your application built and tested each time a code change is pushed to the codebase, **the application is also deployed continuously.**

However, with continuous delivery, you trigger the deployments manually.

Continuous Delivery checks the code automatically, but it requires human intervention to manually and strategically trigger the deployment of the changes.

Do I really need all of this ?

Be aware of the **excel rollercoaster:** `the tools I know have to be all I need to know. I can do everything with that`

CI/CD mechanisms are crucial to achieve:
- reproducibility
- automation (new data, new model, tests)

Non-negligible features for a scientific computing model development :)

Dagger

How a typical CI/CD looks like ?

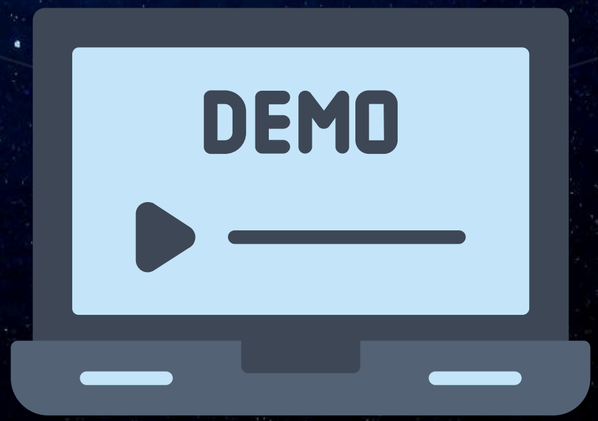
In this session we are going to go through the CI language adopted by Gitlab and Github, as the two main software hosting services out there.

We are going to look at a simple application that should give you already an idea of how a proper workflow can be integrated.

GitHub

Demo time

Some real world ref:
https://github.com/interTwin-eu/interLink/blob/main/.github/workflows/build_images.yaml



All good, then... you are locked in

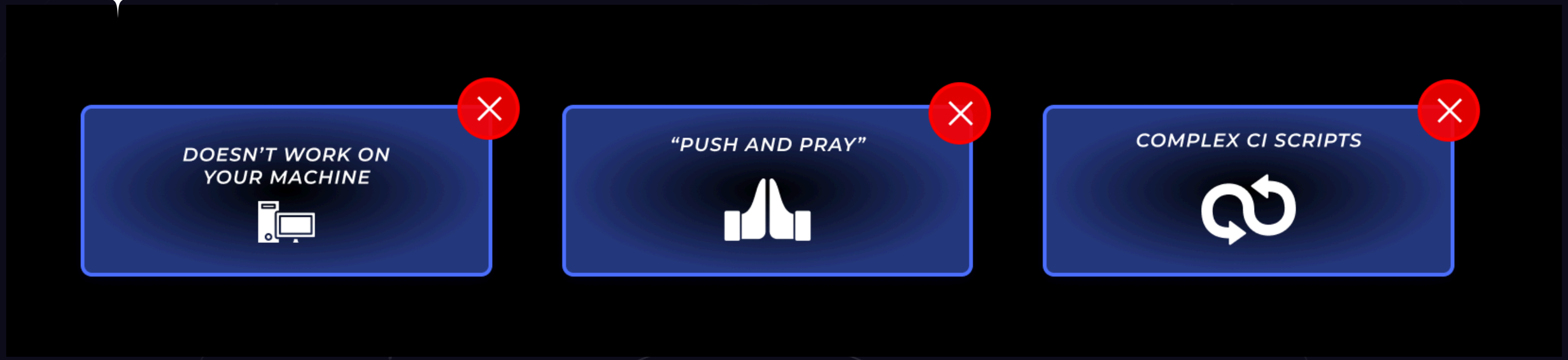
CI/CD tools overview

It does not end here... a LOT more tools have been developed! Each one with its own sintax and behavior.

All of them, very difficult to reproduce on your machine. Not having a simple way to test things locally, translates in a common inefficiency when developing pipelines :

PUSH to git repository.AND PRAY for the test to become green.

Usual issues



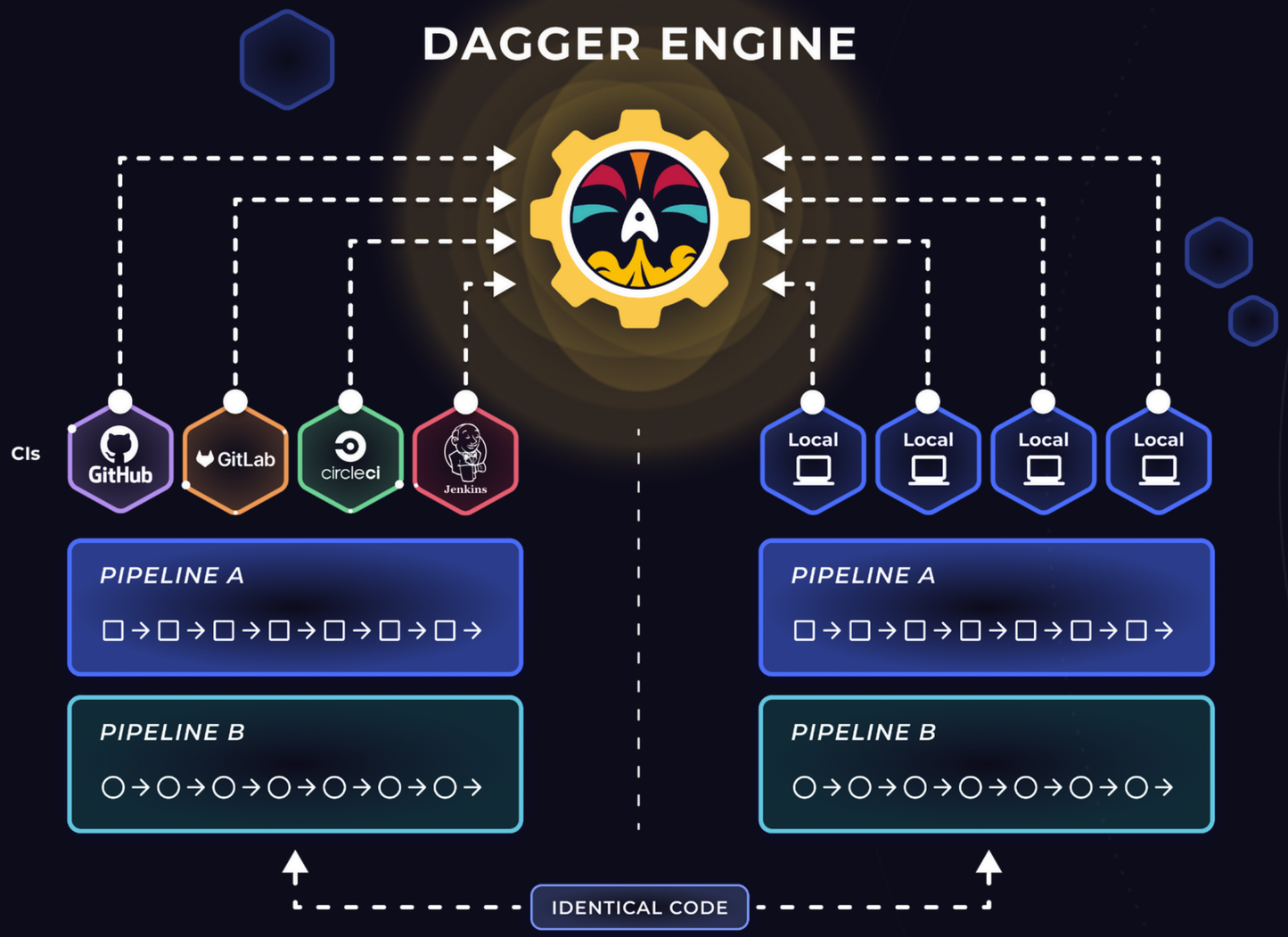
Dagger

It should be better than this

Dagger

CI/CD Solutions with Dagger

Dagger



Eliminate Push And Pray

If it works on your laptop
it'll work in CI

Dagger

This is the result...

Dag on pc, on gh and glab the same!
Just a single command that you can replicate everywhere (GHA, gitlab, your laptop)

```yaml
- name: Integration Test
  uses: dagger/dagger-for-github@v7
  with:
    workdir: ci
    verb: call
    args: -s --name slurm-test build-images new-interlink test stdout
    cloud-token: ${{ secrets.DAGGER_CLOUD_TOKEN }}
    version: "0.13.0"
```

Dagger

And more...

You can literally spin up an environment like the one you are using in a single command

Dagger