DevOps adoption in scientific applications: DisVis and PowerFit cases

Pablo Orviz[1], Mikael Trellet[2], Alexandre Bonvin[2]

[1]Instituto de Física de Cantabria, Santander, Spain
[2]Utrecht University, Utrecht, The Netherlands

Plan

- Good development practices in scientific applications
- DevOps culture
- DisVis and PowerFit use cases
- Our continuous delivery pipeline

Good practices in software development

Good development practices in scientific applications

Why?

Scientific/academic software development has similar goals than professional/commercial software

- User satisfaction
- Bug-free end-product
- Coding environment structured
- Seamless release of new features

However, until now, and as far as we know, only few scientific software seem to have a development policy that would professional good development practices.

Good development practices in scientific applications

When?

Several steps of software development must follow good practices:

- Code writing (dependencies, comments, etc.)
- Code checking (e.g. PEP8)
- Code compilation
- Code testing (performances, results, etc.)
- Code release (e.g. SCM)
- Documentation

Good development practices in scientific applications

How?

Nowadays, plethora of tools, servers, libraries, help users to maintain a good software development environment

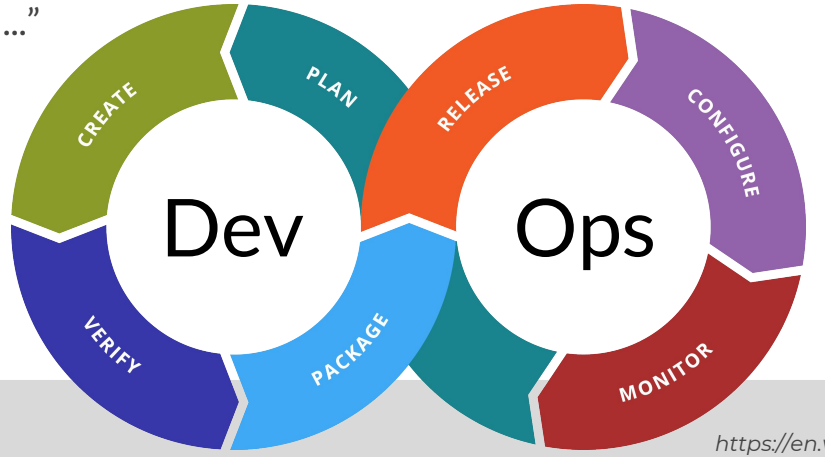
- Code writing ⇒ Developer documentation
- Code checking ⇒ IDE, style conventions checking tools
- Code compilation ⇒ Automated deployment
- Code testing ⇒ Unit tests, integration tests
- Code release ⇒ Integration with SCM tools
- Documentation ⇒ Comments/docstrings scanning tools

DevOps culture

DevOps culture for software development

What is the DevOps culture?

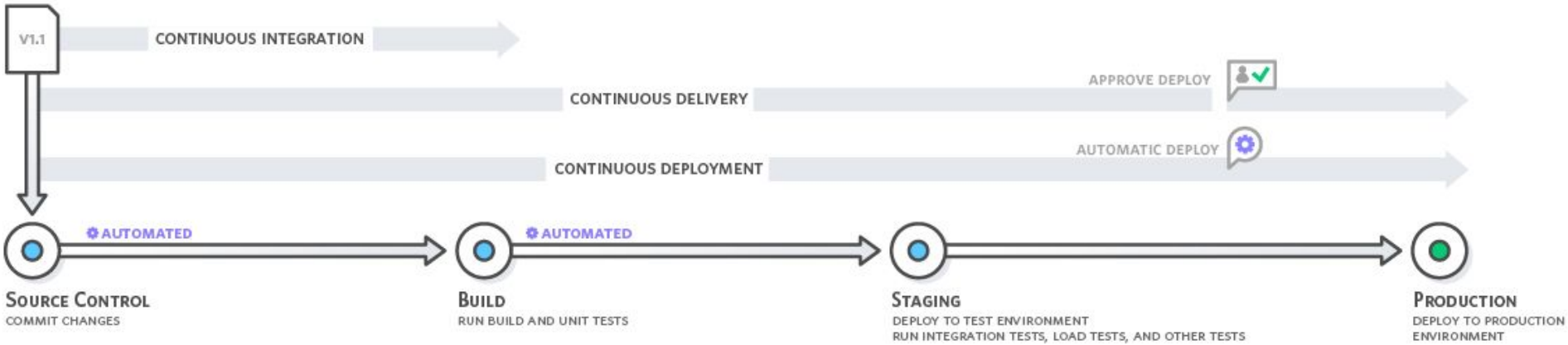
- Wikipedia: "... a software engineering culture and practice that aims at unifying **software development** (Dev) and **software operation** (Ops). The main characteristic of the DevOps movement is to strongly advocate **automation** and **monitoring** at all steps of software construction, from **integration**, **testing**, **releasing** to **deployment** ..."



https://en.wikipedia.org/wiki/DevOps

Continuous Integration/Delivery/Deployment

Continuous Integration (CI)	Code merged in repository ⇒ Automated builds and code tests performed
Continuous Delivery (CD)	**CI** success ⇒ Deploy in testing environments and performs integration tests
Continuous Deployment (CD')	**CD** success ⇒ Deploy to production environment (e.g. as a stable release)



https://aws.amazon.com/devops/continuous-integration/

DevOps culture for scientific/academic software development

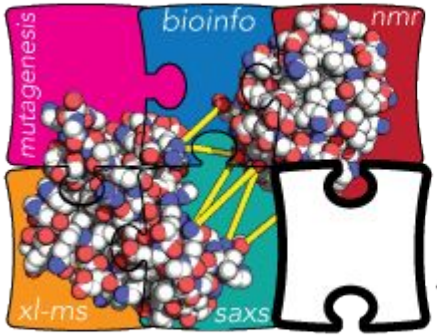
- **Limited** number of contributors ⇒ Development/operations often made by the **same person/people**
- Not all scientists are familiar with **best practices** in operations
- **Reproducibility** must be at the core of Science, setting up **automated** and **systematic** checking points reduces chances of results divergence
- Cost of **defect solving** is reduced if detected **early** in the software development process
- Allows for **faster** and **more frequent** updates, **reduces time** between algorithms improvement and their availability for users
- More **frequent updates** ⇒ **smaller changes** ⇒ **less risk of disruption** in operations
- Automating **test/build/stage** steps allow to focus on the **core** development, the **algorithm(s)**

DisVis & PowerFit

DisVis/PowerFit test cases

Bonvin Lab

DisVis



HADDOCK
High-Ambiguity Driven Docking

Prodigy

PowerFit

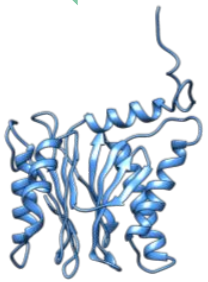
CPORT

SpotON

haddock.science.uu.nl

3D-DART

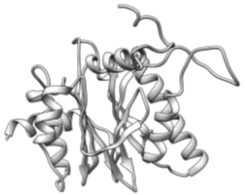
DisVis - visualising accessible interaction space



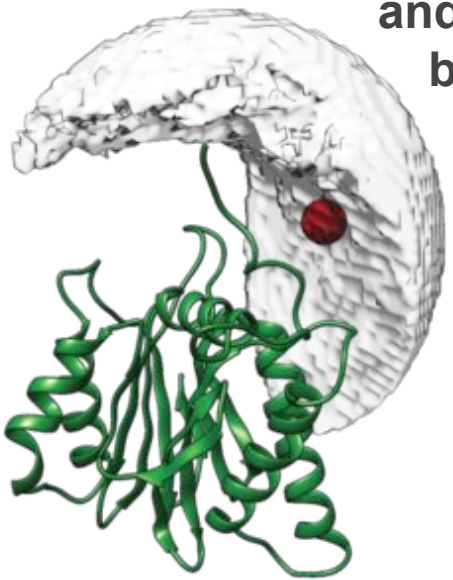
+

Restraints (XL-MS)

+



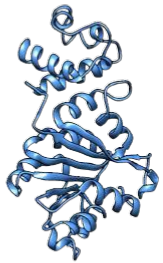
6D exhaustive search



Given 2 interacting structures and a set of distance constraints between them, are there any solutions that satisfy *N* constraints?

Van Zundert and Bonvin, BioInformatics, 2015

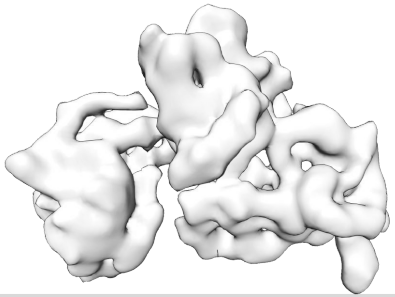
PowerFit - fitting structures into 3D maps



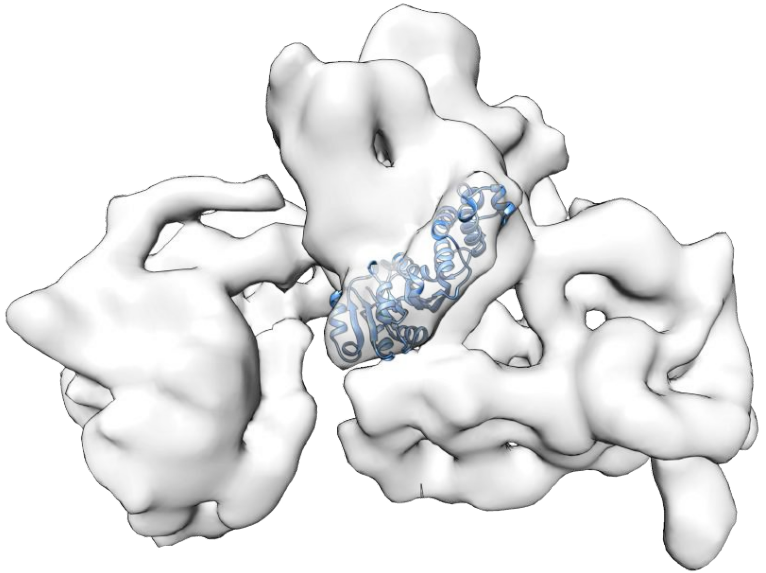
3D structure

+

Low-resolution density map



6D exhaustive search



Best Fit of structure in density map

Van Zundert and Bonvin, AIMS Biophysics, 2015

GRID-enabled web portals

Architecture

User DB

User not found

Input error

Validation

Pre-processing + Input files packaging

Submission to local nodes

OR

Submission to grid node

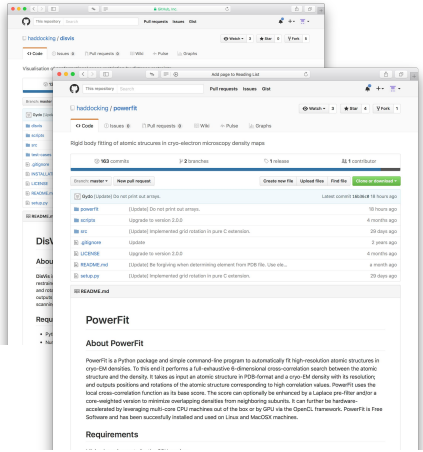
CPU-calculation

Output files packaging + submission of image generation

Chimera image generation

Post-processing + Results formatting

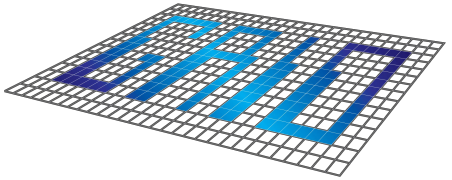
From web form to GPGPU resources



udocker

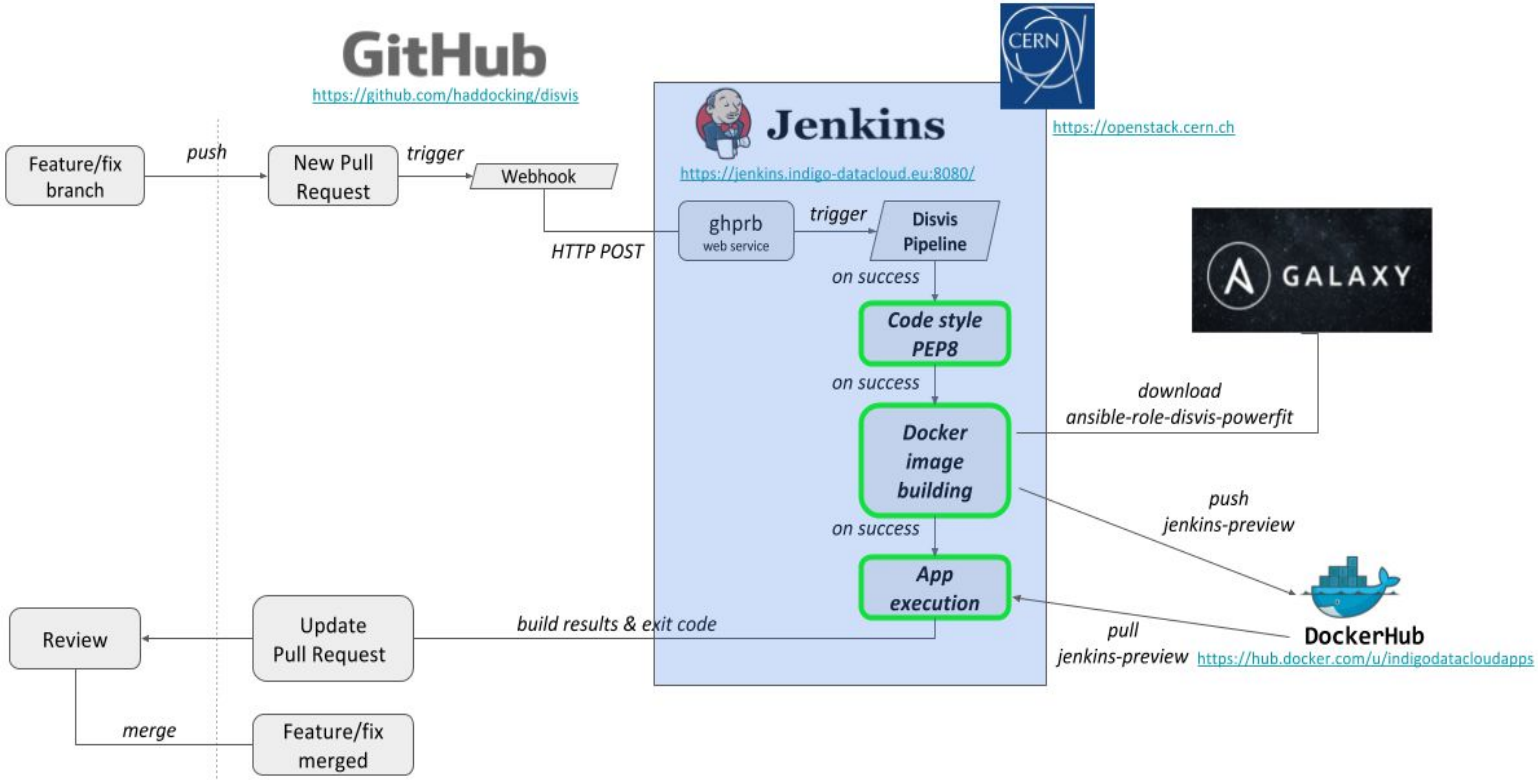
indigodatacloudapps/**disvis**

indigodatacloudapps/**powerfit**



Our continuous delivery pipeline

DevOps workflow for DisVis/PowerFit



Step-by-step

Setup

- **Github repository** setup to trigger "**webhooks**" when certain event occurs (PR, comment on PR, push, etc.)
- **Jenkins server** with (1) GitHubPullRequestBuilder (**GHPRB**) plugin allows to handle webhooks received from Github and triggers the pipeline, (2) pipeline as set of command-line instructions

Continuous Delivery pipeline (1/2)

1. **Fetch** the branch to test (source of the Pull Request)
2. Run **pep8** (pycodestyle, soon to be flake8) for code style checking of the whole project
3. **Build** docker images (GPU driver & application) with **Ansible** roles
4. Send to **indigodatacloudapps** (hud.docker.com) with proper **tag** for testing (**not** production yet)

Step-by-step

Continuous Delivery pipeline (2/2)

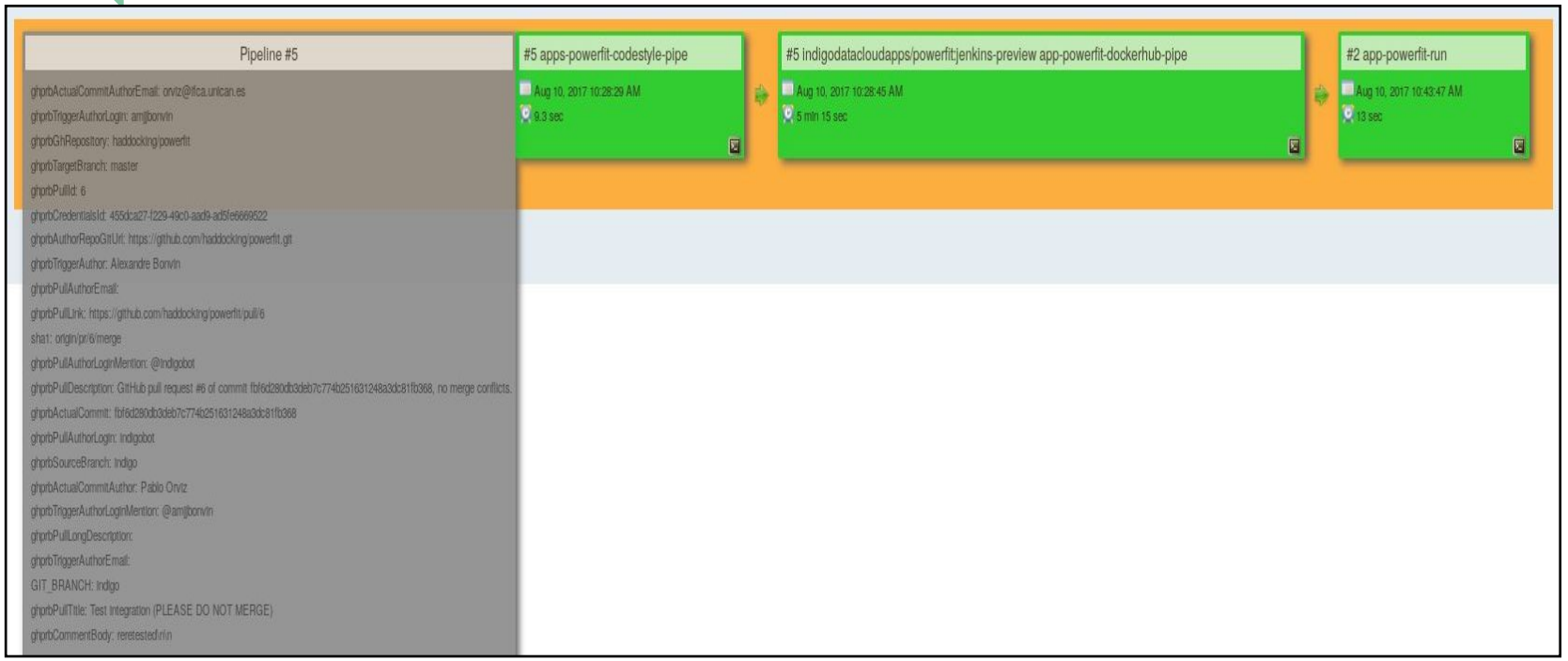
5. Execute app on a running node with **GPU capability**
6. **Validate** results with expected values (**integration test**)

Length of the pipeline: **~7minutes**

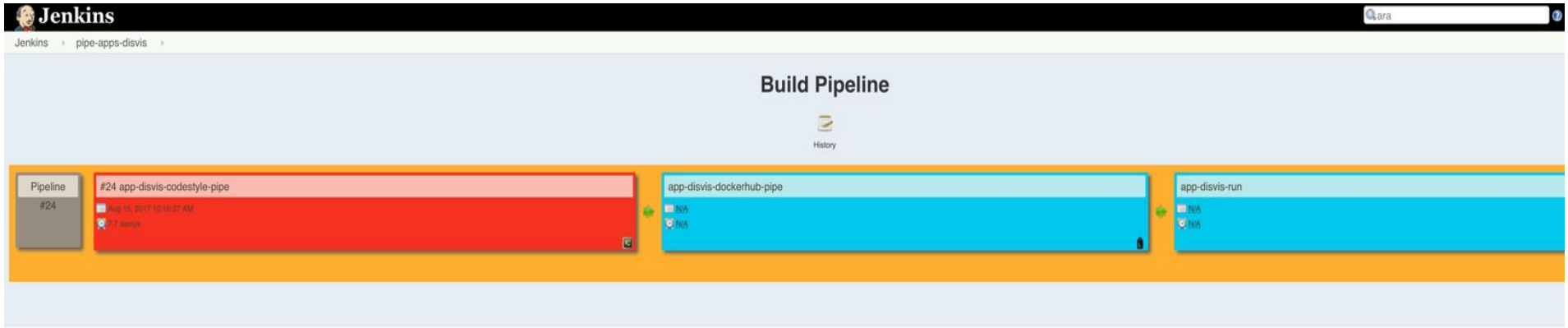
Deployment

- Feedback sent to Github PR that triggered the pipeline (using GitHub API)
- If needed/wanted, the PR is merged
- The indigodataclouds docker image gets its tag updated for production

Jenkins server



Failed step in Jenkins server



In a nutshell

This pipeline is highly configurable and we are showcasing a test case that might be more complete:

- Add **unit tests** in step 2 (as of today, only code style is checked with pep8)
- Also test **multi-CPU** results (as of today, only the GPU version is checked because more error-prone and the one used through docker images in production)
- Implements **Continuous Deployment** where docker images are production-ready

Our short-term view:

- Apply this pipeline to more projects (**PDB-tools**, **HADDOCK-tools**, web servers, etc.)
- Improve the **feedback loop** when a step is failing (remove docker images when integration tests are failing, provide pipeline output to developer in GitHub, etc.)

Acknowledgments

CSB group

Alexandre Bonvin
Jorg Schaarschmidt
Adrien Melquiond

INDIGO datacloud

Pablo Orviz

THANK YOU FOR YOUR ATTENTION

Failed step in GitHub