Automated deployment for better return on investment

Long Vu, DevOps at Ouranos
vu.long@ouranos.ca
2021-July-06
Canadian Research Software Conference
PAVICS (Platform for the Analysis and Visualization of Climate Science)

- [https://pavics.ouranos.ca](https://pavics.ouranos.ca)
- Climate Science as a Service
- Project started in 2016
- Ouranos, CRIM, UofT and PCIC
- Funding: Canarie, ECCC, CEDA, DACCS
PAVICS is a virtual laboratory facilitating the analysis of climate data. It provides access to several data collections ranging from observations, climate projections and reanalyses. It also provides a Python programming environment to analyze this data without the need to download it. This working environment is constantly updated with the most efficient libraries for climate data analysis, in addition to ensuring quality control on the provided data and associated metadata.

- Data and files in netCDF format
- Metadata following CF-conventions
- Parallel computing environment (xarray + dask)

PAVICS climate datasets are hosted on a THREDDS data server at https://paivcs.ouranos.eu/thredds. Although THREDDS provides a user interface for browsing datasets, it is often more practical to navigate the catalogue programmatically. The tutorial introduces the python library to browse the THREDDS catalog, and aims to open a streaming connection to the remote data.

More specifically, the tutorial demonstrates how to access an ensemble of climate simulations, namely Ouranos’ standard ensemble of bias-adjusted climate scenarios version 1.0 (bias-corr 1.0), using python commands. The ensemble contains 22 bias-adjusted CMIP5 simulations in netCDF format, each with three variables (tasmin, tasmax, pr) and dimensions of longitude, latitude and time (1902, 700, 197). The server can provide multiple data access and metadata services but in the following press the tutorial will focus on the CMIP5 service, where instead of downloading huge volumes of data locally, only the relevant portion is accessed using a standard called Data Access Protocol (DAP).

In [13]:
from siphon.catalog import THREDDSCatalog
url = "https://paivcs.ouranos.eu/thredds/catalog/catalogs/simulations/bias_adjust"
cat = THREDDSCatalog(url)
# List of datasets
prislatt (["Number of datasets: 20 (cat.datasets)"])
# Access mechanisms - here we are interested in cdmftp, a data streaming protocol
cd = cat.datasets[0]
prislatt("Access URL: http://esgfaccess.ouranos.eu/cdmftncs/access_urls,keys[13])")
Cashing in with automated deployment

- Order online, getting ROI only when order arrives
- Car dealer, getting ROI only when car is sold
- Time spent fixing bugs, implementing feature, getting ROI only when end user actually gets it (deployed, in production). A package or build not deployed is like a car unsold, still in the dealer parking lot, taking up space.
Automated deployment advantages

- Can deploy much more frequently
  - PAVICS try to deploy nightly
- Smaller changes go into production = less surprise = less stress
  - Each PR is almost deployed by itself, unless 2 PR merged in same day
- If something goes wrong, root cause is much easier to find/revert
Automated deployment advantages for Devs

- Since a PR merge means a deployment, Devs basically have deployment privilege without needing access to production server.

- Autonomy: Devs have full control of what and when to deploy, without needing to wait on someone.
Automated deployment advantages for Org

- Resilience: increase the bus factor: more than one person can perform the deployment
- Traceability: automated deployment have logs, instead of being done by someone, from his personal machine without any trace
Automated deployment advantages for DevOps

- Less interruption
- Can focus on upgrading other components, delegating upgrade of in-house components to Devs
- Avoid being the bottleneck slowing others down
- Have freedom to take off anytime
Automated deployment requirements

- Rigorous code review
- Comprehensive integration test suite
  - PAVICS tests real user end to end workflow using notebooks
- Which means reproducible deployment for testing
  - PAVICS has Vagrant support for spawning a complete VM locally
  - PAVICS has a staging server that checks for deployment every 5 minutes so we can catch error almost instantly after a PR merge
How we did it

**Scheduler**

**Trigger**

**Autodeploy from git repos**

- **Detect new commits**
  - For all repos
  - If at least one repo has new commit on same branch, continue

- **Stop Server**
  - Stop before pull to use current code matching current old state

- **Pull**
  - For all repos

- **Deploy new code**
PAVICS platform deployment

- Platform is made of 3 git repos
  - Plugable
  - Infrastructure-as-code

---

General

Private config

Ouranos override
PAVICS other deploy jobs

- NcML and testdata on Thredds
- The homepage
- Tutorial notebooks in JupyterLab
- GEPS forecast (ECCC)
Questions?

- Contact: vu.long@ouranos.ca
- PAVICS: https://pavics.ouranos.ca

- Resources:
  - Deployment: https://github.com/bird-house/birdhouse-deploy
    + https://github.com/bird-house/birdhouse-deploy-ouranos
  - Test: https://github.com/Ouranosinc/jenkins-config +
    https://github.com/Ouranosinc/PAVICS-e2e-workflow-tests
Reference

- Deployment if new changes on server:
  - [https://github.com/bird-house/birdhouse-deploy/blob/1.13.9/birdhouse/deployment/triggerdeploy.sh](https://github.com/bird-house/birdhouse-deploy/blob/1.13.9/birdhouse/deployment/triggerdeploy.sh) and deployment/deploy.sh
- Triggered nightly by cronjob:
  - [https://github.com/bird-house/birdhouse-deploy/blob/1.13.9/birdhouse/components/scheduler/config.yml.template](https://github.com/bird-house/birdhouse-deploy/blob/1.13.9/birdhouse/components/scheduler/config.yml.template) and scheduler/docker-compose-extra.yml
- All config centralized in
  - [https://github.com/bird-house/birdhouse-deploy/blob/1.13.9/birdhouse/env.local.example](https://github.com/bird-house/birdhouse-deploy/blob/1.13.9/birdhouse/env.local.example)
Reference

- Documentation:
  - [https://github.com/bird-house/birdhouse-deploy/tree/1.13.9/birdhouse/components#scheduler](https://github.com/bird-house/birdhouse-deploy/tree/1.13.9/birdhouse/components#scheduler)

- Vagrantfile and config:
  - [https://github.com/bird-house/birdhouse-deploy/blob/1.13.9/Vagrantfile](https://github.com/bird-house/birdhouse-deploy/blob/1.13.9/Vagrantfile) and [vagrant_variables.yml.example](https://github.com/bird-house/birdhouse-deploy/blob/1.13.9/vagrant_variables.yml.example)

- Battle tested, in production more than a year
Reference

- Generic deployment any files from any git repos

- Testdata deployment
Reference

● Homepage and tutorial notebook deployment
  ○ [https://github.com/bird-house/birdhouse-deploy-ouranos/blob/5dd7608e0639feec4b70cdee6325fad137bc5993/scheduler-jobs/deploy_pavics_landing_notebooks.yml](https://github.com/bird-house/birdhouse-deploy-ouranos/blob/5dd7608e0639feec4b70cdee6325fad137bc5993/scheduler-jobs/deploy_pavics_landing_notebooks.yml)

● Trigger other scripts (GEPS forecast)
  ○ [https://github.com/bird-house/birdhouse-deploy-ouranos/blob/5dd7608e0639feec4b70cdee6325fad137bc5993/scheduler-jobs/retrieve_geps_forecasts.yml](https://github.com/bird-house/birdhouse-deploy-ouranos/blob/5dd7608e0639feec4b70cdee6325fad137bc5993/scheduler-jobs/retrieve_geps_forecasts.yml)