Workflow management system

Valentin Pestel

LPC Caen Bootcamp - 06/11/2024

Why a workflow management system

To achieve your goal, you need complexe workflows:

- Getting the data, pre-processing, extracting information, all that for each runs, then merging
- You can use bash script, but…

In the ideal world, you'd like your worfklow:

- Clear and understandable
  - everything is explicit, I understand and can explain what happen in which order
- Flexible
  - easy re-usability of already existing steps
- Performant and scalable
  - parallelize task that can be, offer solution to monitor the resources needed
- Portable and reproducible
  - Easy to port on another system, easy to reproduce by somebody else

Very smart people have put effort in that, and they found solutions:

nextflow

snakemake

Snakemake

The Snakemake workflow management system is a tool to create reproducible and scalable data analyses. Workflows are described via a human readable, Python based language. They can be seamlessly scaled to server, cluster, grid and cloud environments, without the need to modify the workflow definition. Finally, Snakemake workflows can entail a description of required software, which will be automatically deployed to any execution environment.

From Snakemake documention

To walk you through the key features, let's go through a small set of examples:

- Available here: https://gitlab.in2p3.fr/lpc-dev/snakemake-examples/snakemake_101/

Level 01: my first workflow

Per default, snakemake execute the first rule of the workflow.
Let's start with a dry-run (`-n`)

- `snakemake -n`

And now executing it:

- `snakemake`

The work directory can be set with `-d`:

- `snakemake -d my_workdir`

Level 02: DAG and wildcards

First, let's take a look at the Directed Acyclic Graph (DAG):

- `snakemake -d workdir --dag | dot -Tpdf > dag.pdf`
- Try out `--rulegraph` and `--filegraph` as well

Now let's play with the number of core (-c or --cores):

- `snakemake -d workdir -c 4`

Now let's play a bit with rerunning options

- `snakemake -d my_workdir -n`
- `echo "test" >> workdir/generated_data/data_C_1.bin`
- `snakemake -d my_workdir -n`

Level 03: config files

Ok, now we have a config file, let's run with it:

- `snakemake -d workdir --configfile config.yaml`

Try modifying it and re-running

Level 04: providing scripts

Functionally, the workflow is the same than before

- `snakemake -d workdir --configfile config.yaml`

But now the script used for scripts/generate_data.sh is “tracked”:

- `snakemake -d workdir --configfile config.yaml -n`
- echo “echo DONE” > scripts/generate_data.sh
- `snakemake -d workdir --configfile config.yaml -n`

Level 05: log and performances

Functionally, the workflow is the same than before, again

- `snakemake -d workdir --configfile config.yaml`

There is also a global way of monitoring ressources, through the report:

- `snakemake -d workdir --configfile config.yaml --report report.html`

Let’s make histograms

But first… let’s add dependencies I forgot:

- `micromamba install docopt boost-histogram`

Good, let’s produce a new environment spec file:

- `micromamba env export --from-history > my_env.yaml`

Now let’s take a look to examples:

- https://gitlab.in2p3.fr/lpc-dev/python-examples/root_tree_histogrammer
- 2 solutions provided, using ROOT through pyROOT, or full python uproot and boost-histograms
  - Not really one better than the other, depends on context/person

Now, your turn to build a workflow

- Create the histogramming script from example
- Create the Snakefile that runs `analysis` and the histogram script for a run
- Generate a rule that trigger analysis for every runs and merge the histograms together
  - Tips, look at ROOT command line tool `hadd`

But Snakemake is vast…

And contains solution to problem you haven't met yet:

- File flags, like `temp`, which can be applied on an output to have it deleted as soon as possible
- Possibility to run in a temporary directory, if e.g. the code produce a lot of temporary file

Can submit jobs to a cluster by itself:

- `cluster` interface that can be defined with a `profile` file

Snakemake can manage rule-wise environment:

- Not relevant here, but can be when some steps requires a particular environment

```
rule NAME:
    input:
        "table.txt"
    output:
        "plots/myplot.pdf"
    conda:
        "envs/ggplot.yaml"
    script:
        "scripts/plot-stuff.R"
```