REANA Snakemake Integration

CERN Summer Student Report

Maria Mafalda S. C. Fernando

Supervisor: Marco Vidal García

15 September 2021

1. Introduction

Science progresses through experimentation, and experiments produce large volumes of data that may or may not be usable for testing new theories in the future. Furthermore, these same results might become lost, corrupted or otherwise inaccessible with time, creating the need for means through which data can be re-analysed and new outputs created in the same conditions as initially intended. In other words, there is a need to preserve aspects of the original analysis such as computing environment, experimental datasets, software utilised and workflow steps followed, as well as a need to accommodate its reproduction at scale.

REANA (REusable ANAlyses) is a platform developed at CERN which allows researchers to structure and perform their data analyses with this reproducibility and scalability in mind. It provides a user interface and a command line client to rerun analysis workflows with different input parameters, with support for container technologies, shared storage systems, cloud infrastructures and workflow engines.

At the time of writing this report, REANA supports sequential (serial), CWL (Common Workflow Language) and Yadage workflow systems, with support for the Snakemake system in progress. The work for this studentship focused on the integration of the latter.

2. Snakemake

Overview

Snakemake is a workflow engine that originated in bioinformatics in 2012. It provides a Python-based workflow definition language and a powerful execution environment. It allows the definition and scalable execution of each step in a workflow, including the deployment of the software stack required for each step.

Amongst other things, Snakemake:

- Allows the definition and execution of each step in a workflow
- Is scalable to server, cluster, grid and cloud environments without the need to change the workflow
- Provides readable text-based workflow definitions
- Provides a portable execution environment
- Automatically infers dependencies between jobs
- Deploys the necessary software stack for each step of a workflow
- Creates interactive analysis reports
- Avoids duplicate work

Workflow structure

Snakemake workflows are defined in Snakefiles, similar in structure to Makefiles. These use Snakemake's domain-specific language (DSL), which is close to standard Python syntax. Snakemake workflows are defined in terms of rules, and each rule specifies:

- Rule name
- List of input files, with support for wildcards

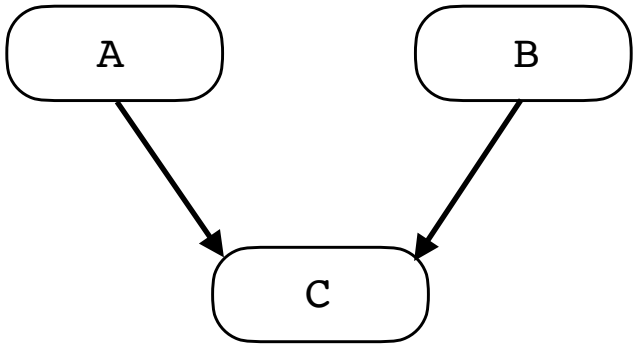
- List of output files, with support for wildcards
- Shell command or Python code to generate output
- Container, number of threads, additional tags, etc

When invoked, Snakemake creates a directed acyclic graph (DAG) from the given workflow, which represents the plan for the jobs, or rule executions. In the DAG, paths are jobs that need to be executed in sequence, and jobs in disjointed paths can be run in parallel, and will be run in parallel if the Snakefile is invoked with the needed resources (ie. number of cores). This plan is a product of dependencies being automatically inferred from different rules' output and input files.

In general, a basic Snakefile may have the following format:

```
rule A:
  input:
        "input_A"
        …
  output:
        "output_A"
        …
  shell:
        …

rule B:
  input:
        "input_B"
        …
  output:
        "output_B"
        …
  shell:
        …

rule C:
  input:
        "output_A"
        "output_B"
  output:
        "output_C"
  shell:
        …
```

And the corresponding DAG, in such a case, would be generated as follows:



3. REANA Integration

Motivation

Snakemake is a mature, widely used tool amongst researchers. In particular in the LHCb experiment which selected Snakemake amongst other workflow systems as the best suited to the collaboration practices. Several groups are using Snakemake to carry out their analyses. The integration of Snakemake into REANA will allow to use the REANA platform to ensure the reproducibility of LHCb analyses.

Integration Steps

Throughout the duration of this studentship, a new repository with the basic skeleton for the REANA Snakemake engine (`reana-workflow-engine-snakemake`) was created and worked on, alongside changes to `reana-client`, `reana-commons` and `reana` for operation handling, report generation, validation, etc. The author focused on initial investigation into Snakemake itself, translation of already existing analysis workflow demos, validation and documentation.

4. REANA Demos

ROOT6 and RooFit

This reproducible analysis example emulates a particle physics analysis where the signal and background data are processed and fitted against a model, using the `RooFit` package of the ROOT framework.

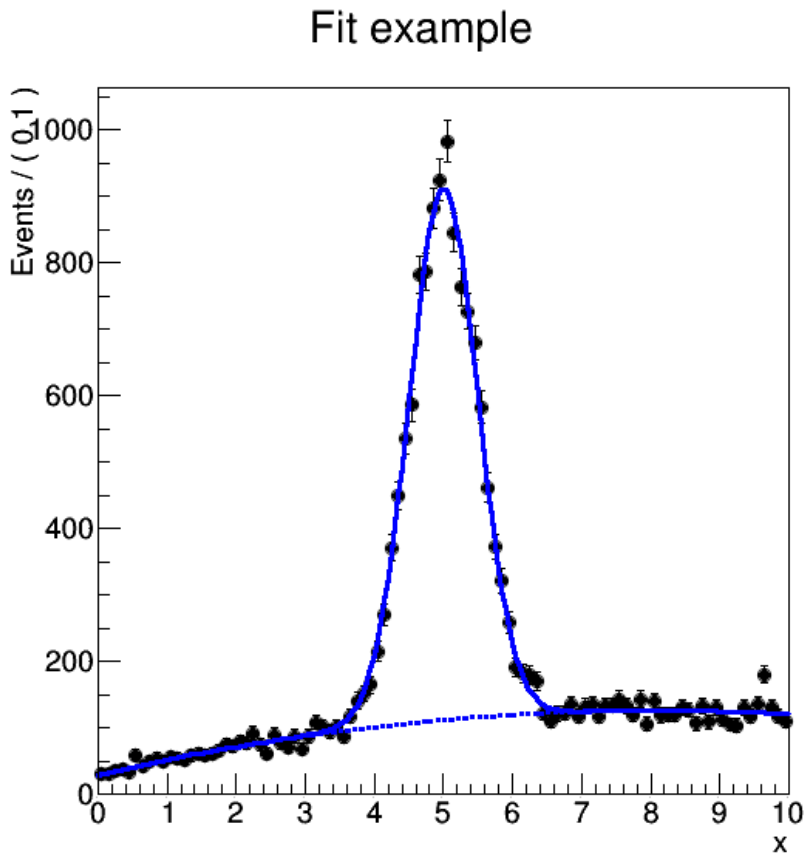
Instead of taking in input data directly, its inputs are generated at runtime. The first stage of the analysis generates the signal and background from the ROOT macros `gendata.C` and `fitdata.C`, and the second stage creates a fit for them.

The computing environment for this analysis is encapsulated within the `reana-env-root6` Docker image, in order to enable its reproduction in the future.

The Snakefile created takes in configuration from an external .yaml file, which specifies the macros that will generate the data and the number of events to consider, and has one rule for each sequential stage.

```
rule gendata:
  input:
        gendata_tool=config[“gendata”]
  output:
        “results/data.root”
  params:
        events=config[“events”]
  container:
        “docker://reanahub/reana-env-root6:6.18.04"
  shell:
        “mkdir -p results && root -b -q “
        “‘{input.gendata_tool}({params.events},\”{output}\”)’”

rule fitdata:
  input:
        fitdata_tool=config[“fitdata”]
        data=“results/data.root”
  output:
        “results/plot.png”
  container:
        “docker://reanahub/reana-env-root6:6.18.04"
  shell:
        "root -b -q ‘{input.fitdata_tool}(\”{input.data}\”,"
        “\"{output}\")'"
```

The output of this analysis is shown below, for a given number of events:



World Population

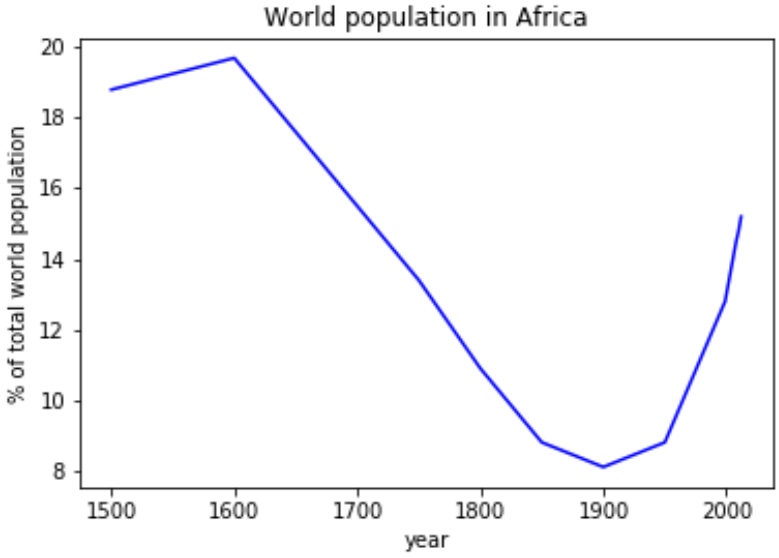
This example uses parametrised Jupyter notebook to analyse world population evolution. It takes as input a dataset with historical and predicted world population numbers (CSV), which the `worldpopulation.ipynb` Jupyter notebook studies in order to print a figure on the evolution of the world population in a given region and time period.

Once again, a Docker image is used in order to enable the reproduction of the analysis in the correct environment and with the correct parameters, such as Jupyter notebook version and notebook kernel used.

The Snakefile created takes in configuration from an external .yaml file, which specifies the file for the input data, parameters and the notebook to be used, as well as the output file to create.

```
rule worldpopulation:
  input:
        notebook=config[“notebook”],
        input_file=config[“input_file”]
  params:
        region=config[“region”],
        year_max=config[“year_max”],
        year_min=config[“year_min”]
  output:
        output_file=config[“output_file”]
  container:
        “docker://reanahub/reana-env-jupyter:2.0.0"
  shell:
        "mkdir -p results && papermill {input.notebook}”
        “ /dev/null -p input_file {input.input_file} -p “
        “output_file {output.output_file} “
        “-p region {params.region} -p year_min “
        “{params.year_min} -p year_max {params.year_max}”
```

The output of this analysis is shown below, for a given region and time period:



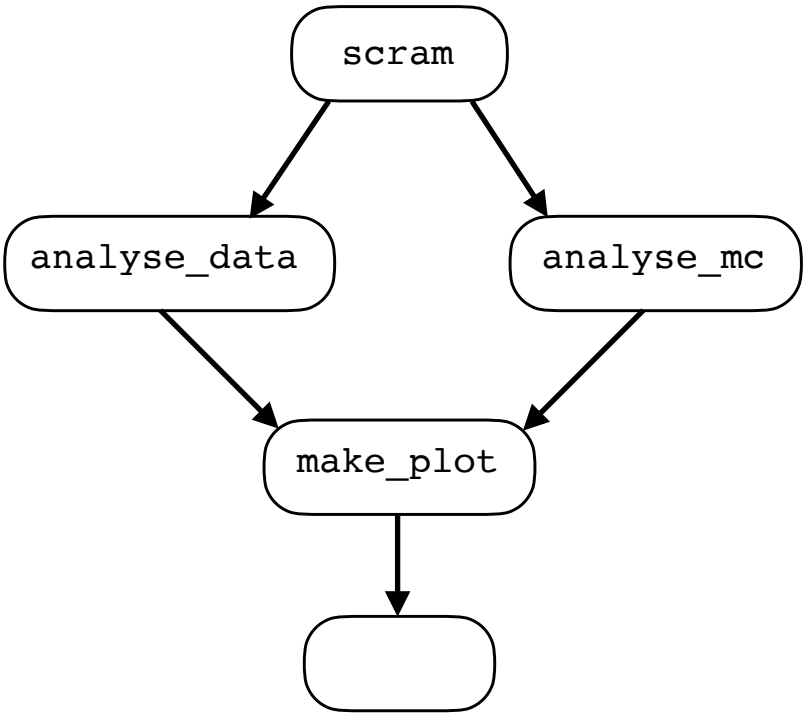
CMS Higgs To Four Leptons

This demo is a study of the Higgs to four lepton decay channel analysis that uses simplified CMS Open Data from 2011 and 2012. As an input, it uses validated runs and ROOT data files as input, as well as collision and simulated data accessed at run time via the CERN Open Data portal.

The analysis is performed in two stages: the processing of the data and the creation of a plot with the results. Within the first stage, there are two processes that can be run in parallel: one for analysing the original data and one for running a Monte Carlo Analysis on the simulated data.

An additional rule was created for the scram process, as this needs to run sequentially before both of the latter. In order to ensure this job order, the analysis rules take as input a flag file created by the scram rule - otherwise, Snakemake would not infer the order automatically.

The DAG for this particular demo looks like this:

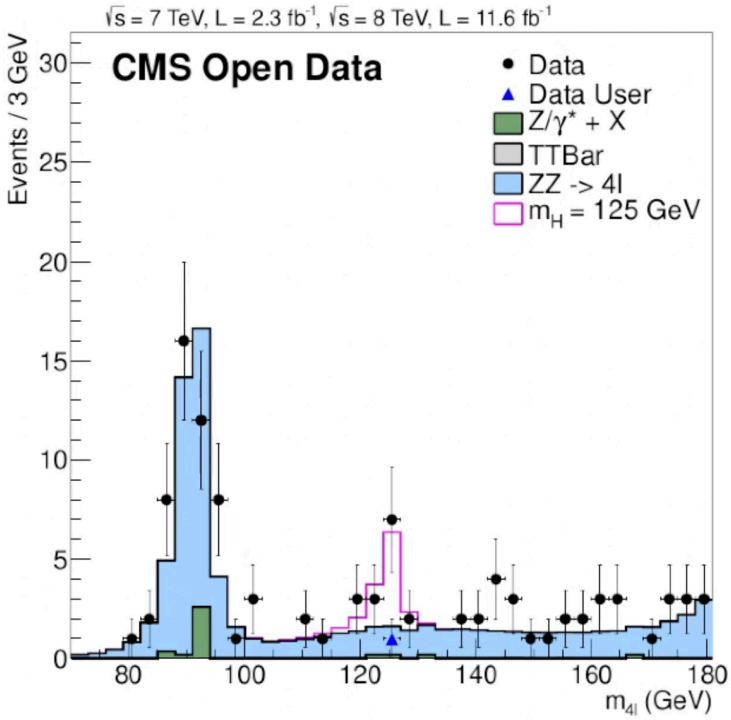


Snakefile:

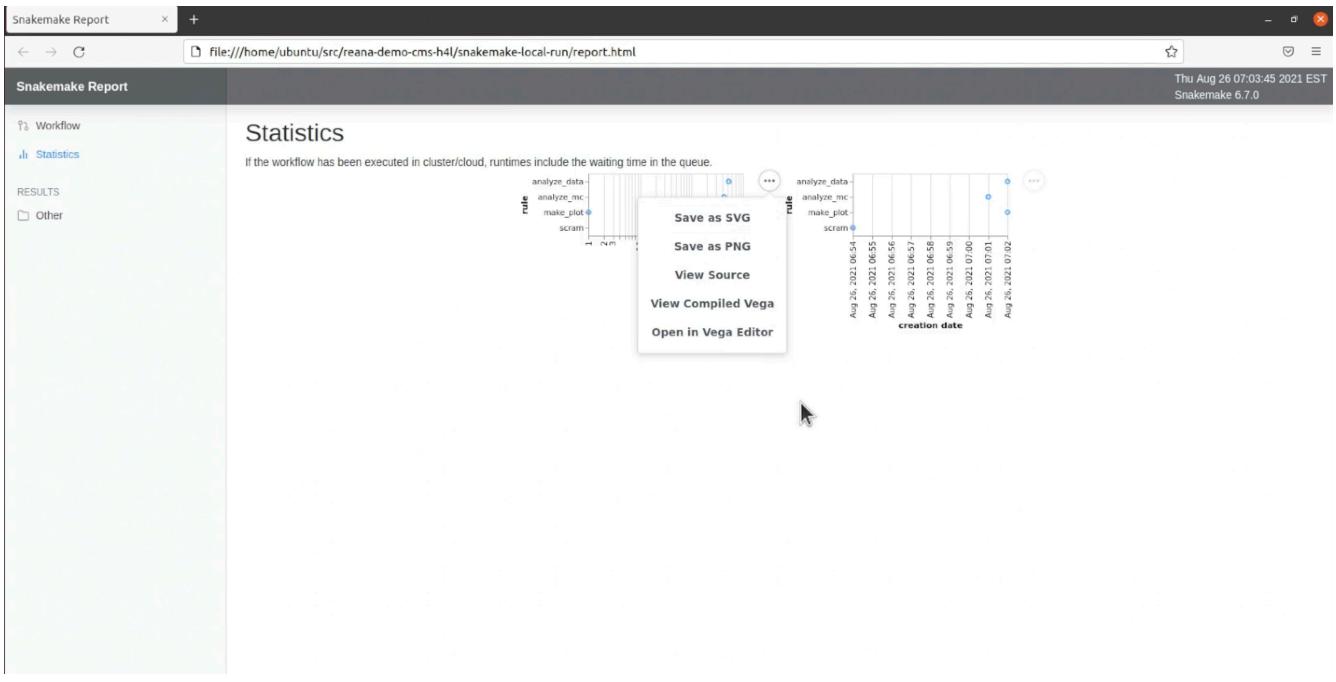
```
rule all:
  input:
        “results/mass4l_combine_userlvl3.pdf”

rule scram:
  input:
        “data”,
        “code”
  output:
        touch(“results/scramdone.txt”)
  container:
        “docker://cmsopendata/cmssw_5_3_32"
  shell:
        “source /opt/cms/cmsset_default.sh “
        . . .

rule analyse_data:
  input:
        “data”,
        “code”,
        “results/scramdone.txt”
  output:
        “results/DoubleMuParked2012C_10000_Higgs.root”
  container:
        “docker://cmsopendata/cmssw_5_3_32"
  shell:
        “source /opt/cms/cmsset_default.sh “
        . . .
rule analyse_mc:
  input:
        “data”,
        “code”,
        “results/scramdone.txt”
  output:
        “results/Higgs4L1file.root”
  container:
        “docker://cmsopendata/cmssw_5_3_32"
  shell:
        “source /opt/cms/cmsset_default.sh “
        . . .

rule make_plot:
  input:
        “data”,
        “code”,
        “results/DoubleMuParked2012C_10000_Higgs.root”,
        “results/Higgs4L1file.root”
  output:
        “results/mass4l_combine_userlvl3.pdf”
  container:
        “docker://cmsopendata/cmssw_5_3_32"
  shell:
```

```
"source /opt/cms/cmsset_default.sh "
. . .
```

The output of this analysis is shown below:



In this particular demo, an interactive HTML report was also generated, including analysis results, runtime statistics, the DAG and job details:



5. Snakefile Validation

Despite not having been successfully carried out by the author, the task of validating a snakefile inside REANA was also worked on. In particular, an effort was made in the direction of parsing snakefiles in a modular manner, to obtain resources (containers and parameters) from each rule. For this purpose, investigation into the Snakemake API and source code in order to understand how their `Workflow` and `Rules` classes worked together, as well as how they were linked to the `snakemake` initialisation function.

It was concluded that having a `Workflow` class object was the best way to get rule information and resources from a snakefile; however, the means through which snakefiles were included in the pre-existing class within Snakemake were not compatible with the REANA validation structure.

6. Conclusion

During the course of this Summer Studentship, the author worked with the REANA full time team in an effort to integrate the Snakemake workflow engine into the existing platform. This involved research into both REANA and Snakemake, and participating in some first steps towards this goal, mostly through translating the existing example demos already written for REANA with other workflow engines.

By the date of this report, the initial implementation of the Snakemake workflow engine in REANA was fully functional and validated by the Snakemake analysis examples developed during the summer studentship.

7. References

https://docs.reana.io/

https://snakemake.github.io/

https://academic.oup.com/bioinformatics/article/28/19/2520/290322

https://inspirehep.net/literature/1761294

https://f1000research.com/articles/10-33/v1

https://github.com/reanahub/reana-demo-root6-roofit

https://github.com/reanahub/reana-demo-worldpopulation

https://github.com/reanahub/reana-demo-cms-h4l

https://github.com/snakemake/snakemake