# PLATFORM FOR REPRODUCIBLE ANALYSES

## AUGUST 2021

**AUTHOR(S):**
Diamantis Patsidis

CERN openlab

**SUPERVISOR(S):**
Anna Ferrari

CERN openlab

# PROJECT SPECIFICATION

During the recent years, the never-ending integration of computers and the Internet in our daily life has resulted in an unprecedented amassment of software and information. Nearly 250 million Github repositories [1], close to two million academic publications each year [2] and the total amount of data created, captured, copied, and consumed globally is forecast to increase rapidly, reaching 79 zettabytes in 2021 [3]. But growth, creates new challenges. Maintaining applications require continuous and expansive work, many technologies become obsolete in a short amount of time, and it is nearly impossible for software to execute successfully, even in the near future, without keeping it up to date.

In this scenario, it is clear that we need to tackle the challenges related to how to create persistent and sustainable researches, despite technological changes so that the research community can focus on producing new results rather than constantly maintaining or try to reproduce the work that has been done previously. Even more, the exponential growth of data availability led to the constant evolution of data analyses and scientific needs, so researchers nowadays need to work fast and efficiently, and research reusability can significantly reduce the time and effort needed. The aim of this project is to tackle these challenges by providing a platform for reproducible analyses. This platform must be easy in use, without requiring specialized knowledge in different technologies. It must be intuitive for users from a wide range of scientific fields and backgrounds. At the same time, it must be flexible, providing different levels of abstraction for different use cases and requirements and provide highly specialized level of technology under the hood. The amount of redundant software will be minimized as well as the development of similar processes. As a consequence, the user will be empowered to reuse and parametrize new or previously developed analyses, together with ready to use tools and services for data visualization and machine learning applications which dominate in today's research and development practices.

# ABSTRACT

This report investigates how to create reproducible analyses by providing a systemic approach for general users to follow best research practises. It bases on REANA, a dedicated tool for analysis reproducibility developed by CERN. Despite its importance, REANA setup and use remain inaccessible for the majority part of the research community. As its adoption in everyday research require significant additional time and knowledge which is discouraging. This project tackles these challenges, in defining and implementing a web interface that guides the user to the best practises of reproducibility for analyses, with REANA under the hood. New libraries and API have been implemented for this purpose.

This tool is certainty innovative in the research community and feels the gap between the results provided by the research community and highly performant infrastructures that already exist. The use at large scale will represent a new and effective form of communication between researchers from different fields and backgrounds.

# TABLE OF CONTENTS

## 1. INTRODUCTION

During the recent years, the never-ending integration of computers and the Internet in our daily life has resulted in an unprecedented amassment of software and information. Nearly 250 million Github repositories [1], close to two million academic publications each year [2] and the total amount of data created, captured, copied, and consumed globally is forecast to increase rapidly, reaching 79 zettabytes in 2021 [3]. But growth, creates new challenges. Maintaining applications require tedious work since the amount of content to be managed has significantly increased. Many technologies become obsolete in a short amount of time due to the world's and the scientific community's ever-changing needs. This makes it nearly impossible for software that relies on them to execute successfully, even in the near future, without keeping it up to date. Migrating applications to new technologies in order to keep them running has become a common practice but provides only a temporal solution.

The aim of this project is to tackle these challenges by providing a platform for reproducible analyses. Earlier work done, includes containerization technologies for creating immutable environments capable of running applications even several years in the future or complex frameworks like REANA from CERN [4] which combines os-level virtualization technologies, container orchestration software and the workflow engines to provide reproducible analyses, as described in chapter 2. But these solutions usually come with a usage barrier due to the wide range of prerequisite knowledge needed, limiting their scalability to domain specific applications. The platform must be easy in use, without requiring specialized knowledge in different technologies. It must be intuitive for users from a wide range of scientific fields and backgrounds. At the same time, it must be flexible, providing different levels of abstraction for different use cases and requirements and provide highly specialized level of technology under the hood. The amount of redundant software will be minimized as well as the development of similar processes. As a consequences, the user will be empowered to reuse and parametrize new or previously developed analyses, together with ready to use tools and services for data visualization and machine learning applications which dominate in today's research and development practices.

## 2. REANA FOR REPRODUCIBILITY

REANA is a platform for creating reusable and reproducible data analysis [5]. It was developed for the requirements and processes of the particle physics community and gradually expanded, supporting a wide range of reproducibility practices and diverse scientific needs [4]. The platform achieves the preservation of data analysis by utilizing container technologies such as Docker [6], to instantiate runtime environments containing all the necessary packages and software required for the analysis. These environments are OS containers, like a virtual machine, in which can be installed, configured, and run any applications. In order to support complex data analysis pipelines with multiple environments, container orchestration technologies are used (Kubernetes, HTCondor) which automate the managing of the different environments.

An important step needed to achieve the reusability and reproducibility of the code is the definition of the computational workflow of the data analysis. That is, the collection of the environments, scripts, parameters, commands, and analysis pipelines required for driving the analysis. REANA supports several workflow

engines, from simple sequential workflows (Serial), with a serial execution order of the different steps, to more complicated workflows with parallel steps (Yadage, CWL). [7] [8]. On top of these technologies, REANA provides a set of micro-services, responsible for instantiating, launching, and monitoring the data analysis pipelines, with which the user can interact by a command-line client or a web user interface.

A common scenario of interaction with REANA is presented in Figure 1. Usually, it consists of the user connecting to the REANA cluster, creating a new analysis, uploading the necessary files (code, data, and workflow files) and finally initializing the execution. The following example in Figure 1 describe the above scenario.

```
$ export REANA_SERVER_URL=http://example.org # connect to REANA cloud
$ export REANA_ACCESS_TOKEN=XXXXXXX
$ reana-client create -n myanalysis          # create new analysis
$ export REANA_WORKON=myanalysis
$ reana-client upload ./code ./data          # upload code and data
$ reana-client start                         # start workflow run
```

Figure 1: Common REANA interaction for running an analysis. [4]

REANA is an important step forward from the traditional practices of software development for data analysis, which required frequent maintenance of the code to facilitate its future reuse or its redevelopment. Indeed, it empowers the user to

1. organize environments and provide workflows,
2. parametrize the analysis in different steps
3. easily run the analysis with multiple environments

Nevertheless, it requires substantial understanding in specialized software technologies and defining the necessary runtime environments and computational workflows can take significant effort and time. These requisites limit REANA's adoption in user's daily practices and making it a suitable choice mostly for large-scale analysis and research teams. The following Section 3 proposes a solution to simplify the process of creating the necessary environments, and Section 4 for defining the computational workflows. A more intuitive and user-friendly approach, by a graphical user interface or general-purpose programming language will make it easier for a diverse audience to write reproducible analysis pipelines.

## 3. CONTAINERIZATION

Containerization represents a crucial step in creating reproducible and sustainable analyses. Containerized environments can be thought of as virtual machines with all the needed software pre-installed. The advantage of containers lies in their characteristic of sharing the OS kernel of the host machine. This allows for better resource management and subsequently multiply containers can be served with a single operating system installation. In particular, containerized environments are a collection of dependencies, packages and software necessary for the execution of the analyses independent from future updates and new-found conflicts. Regardless of the technological changes the contents of the containerized environments will not be affected, and the reproducibility of the analysis will persist in time.

Many different containerization technologies have been developed over the last years. The most prominent and widely used are Docker and Singularity. Despite their benefits in creating long lasting analyses, container technologies introduce new challenges and increase the complexity in data analysis and software

development. The additional requirements of understanding a container technology and the time needed to create the necessary environments can discourage developers and consequently the persistency and reproducibility of their analyses is most of the time not ensured, especially when they must deal with tight deadlines or complex environment setups.

Every container technology has its own format to define the environments and usually some knowledge on operating systems and bash scripting is required. In order to help developers in defining the necessary environments and ease the creation of reproducible analyses, an API responsible for automating the process, currently supporting Docker environments has been proposed. The user through a web interface or high-level programming language can interact with the API and easily choses the necessary base environment, packages and software. As default, a new extended version of the base environment is automatically created to support the user's requisites. This simplifies the process by limiting the use cases developers will need to manually create an environment and provides a more familiar and intuitive way interacting with the Docker engine than bash scripting. Currently, system packages or Python packages (Python 2, Python 3) are supported to extent an environment, but more technologies can be added in the future.

Below an example using the API is depicted.

```
new_image = DockerImage()

new_image.setBaseImage('reanahub/reana-env-root6','6.18.04')

new_image.setSystemPackages(['libyaml-dev', 'python-numpy', 'zip'])

new_image.setPackages({'hftools': '0.0.6'}, 'python2')

new_image.setWorkdir('/code')
new_image.addFiles('code', '/code')
```

Figure 2: Example of API extending an environment.

The user creates an environment based on the version 6.18.04 of the "*reana-env-root6*" image by the REANA team and adds additional packages and files [8]. The added packages include system packages like 'libyaml-dev', 'python-numpy', 'zip' and Python2 packages like 'hftools'. Also, a folder named 'code' is loaded on the environment and set as the working directory. The API creates the new image by communicating through subprocesses with the Docker engine and passes the information describing the image to the engine.

A Dockerfile is created from the above process, and will be used to create the extended environment. Figure 3 shows an example.

```
FROM reanahub/reana-env-root6:6.18.04

RUN apt-get update -y && apt-get install -y --no-install-recommends \
        libyaml-dev \
        python-numpy \
        zip && \
        apt-get clean && rm -rf /var/lib/apt/lists/*

RUN pip install --upgrade pip
RUN pip install hftools==0.0.6

ADD code /code
WORKDIR /code
```

Figure 3: Automatically created Dockerfile for the above example in Figure 2.

## 4. WORKFLOWS

Workflows define a common language to describe the end-to-end analysis pipeline and it mainly encompasses steps and information about its execution. Generally, workflows include all details related to the analysis, such as commands needed to be run at the specific step, input data, code files, inputs and outputs across different step, the environment used for the execution of the step and many more. Workflows are defined in a YAML file, based on a workflow engine with its specific syntax format. Some of the most common ones are the Serial, for creating simple serial analyses and Yadage or CWL, supporting parallel and data intensive analyses. It is clear the use of such technologies requires a specific background as well.

For this reason, an API supporting the Yadage workflow engine has been developed. Yadage originates from the physics community and uses a JSON schema to define the parametrized workflows as a collection of different stages. Each stage represents a single parametrized step of the workflow and packages information about the environment, parameters, the files that are referenced and its dependencies regarding the other stages. The API uses a template based on the syntax format of a single Yadage stage and can be populated with information that the user defines through a web interface or a high-level programming language. This results in a YAML file describing the workflow. The automation through user interface of such a technology represents an important step forward for the entire research community.

In Figure 4, the contents of a YAML file using the Yadage workflow engine are presented:

```yaml
stages:
- name: image_filter1
  dependecies:
  - init
  scheduler:
    scheduler_type: singlestep-stage
    parameters:
      input_image: data/test_image.png
      filter_type: median
      ksize: 9
      output_image: '{workdir}/output_image1.png'
      notebook: code/image_filter.ipynb
    step:
      process:
        process_type: string-interpolated-cmd
        cmd: 'papermill "{notebook}" /dev/null -f "{notebook_parameters}" -p input_image
          "{input_image}" -p filter_type "{filter_type}" -p ksize "{ksize}" -p output_image
          "{output_image}" '
      publisher:
        publisher_type: frompar-pub
        outputmap:
          output_image: output_image
      environment:
        environment_type: docker-encapsulated
        image: diamantispa/reana-demo-imagefilter
- name: image_filter2
  dependecies:
  - image_filter1
  scheduler:
    scheduler_type: singlestep-stage
    parameters:
      input_image:
        stages: image_filter1
        output: output_image
        unwrap: true
      filter_type: gaussian
      rows: 3
      cols: 3
      output_image: '{workdir}/output_image2.png'
      notebook: code/image_filter.ipynb
    step:
      process:
        process_type: string-interpolated-cmd
        cmd: 'papermill "{notebook}" /dev/null -f "{notebook_parameters}" -p input_image
          "{input_image}" -p filter_type "{filter_type}" -p rows "{rows}" -p cols
          "{cols}" -p output_image "{output_image}" '
      publisher:
        publisher_type: frompar-pub
        outputmap:
          output_image: output_image
      environment:
        environment_type: docker-encapsulated
        image: diamantispa/reana-demo-imagefilter
```

Figure 4: Example of a Yadage workflow

In the example in Figure 4, the workflow comprises of two stages, ‘image_filter1’ and ‘image_filter2’. Both use the same Jupyter notebook, which implements different filters in OpenCV [10]. In the first stage a Median filter with kernel size (9,9) is used to filter the image, and the produced image is then passed to the

second stage to be filtered with a Gaussian filter of size (3,3). The image file and the filter parameters are defined by the user as parameters for each workflow stage, along with other information about the environments and the dependencies.

Instead of manually writing the YAML file to describe the workflow, the same can be achieved through the API.

```
workflow = Workflow()
```

```
stage = Stage('image_filter1')
```

```
stage.setDependencies(['init'])
```

```
parameters = {}
parameters['inputs'] = {}
parameters['outputs'] = {}
parameters['notebook'] = 'code/image_filter.ipynb'
parameters['inputs']['input_image'] = 'data/test_image.png'
parameters['inputs']['filter_type'] = 'median'
parameters['inputs']['ksize'] = 9
parameters['outputs']['output_image'] = '{workdir}/output_image1.png'
stage.scheduler.setParameters(parameters)
```

```
workflow.addStage(stage)
```

```
stage = Stage('image_filter2')
```

```
stage.setDependencies(['image_filter1'])
```

```
parameters = {}
parameters['inputs'] = {}
parameters['outputs'] = {}
parameters['notebook'] = 'code/image_filter.ipynb'
parameters['inputs']['input_image'] = {'stages': 'image_filter1', 'output': 'output_image', 'unwrap': True}
parameters['inputs']['filter_type'] = 'gaussian'
parameters['inputs']['rows'] = 3
parameters['inputs']['cols'] = 3
parameters['outputs']['output_image'] = '{workdir}/output_image2.png'
stage.scheduler.setParameters(parameters)
```

```
workflow.addStage(stage)
```

```
workflow.build()
```

Figure 5: Implementation of the workflow in Figure 4, using the API.

Yadage parameters that can be defined through the API, include name of stage, stage dependencies, parameters and the environment used in the stage. The API was developed having in mind analyses consisting of Jupyter notebooks, so currently the workflows created support shell commands only for executing Jupyter notebooks.

## 5. PLATFORM (WEB APPLICATION)

As stated before, a platform providing a user interface has been developed. It integrates the ~~above~~ API explained in chapter 3 and 4 for automating the workflows. Users are guided in ~~can~~ creating their own analysis workflows, through an intuitive user interface.

The frontend was developed using the React framework, whereas the backend bases on Python API explained in chapter 4. Some additional functionalities have been added for uploading data and code files, inspecting Jupyter notebooks to determine parameters and endpoints for automatically specifying the dependencies between the stages and of their parameters, depending on the inputs and outputs the user defined for each stage.

In Figure 6, Figure 7 and Figure 8, the pages of the web application. First, the user defines the stages comprising his analysis using a drag and drop menu, Figure 6.

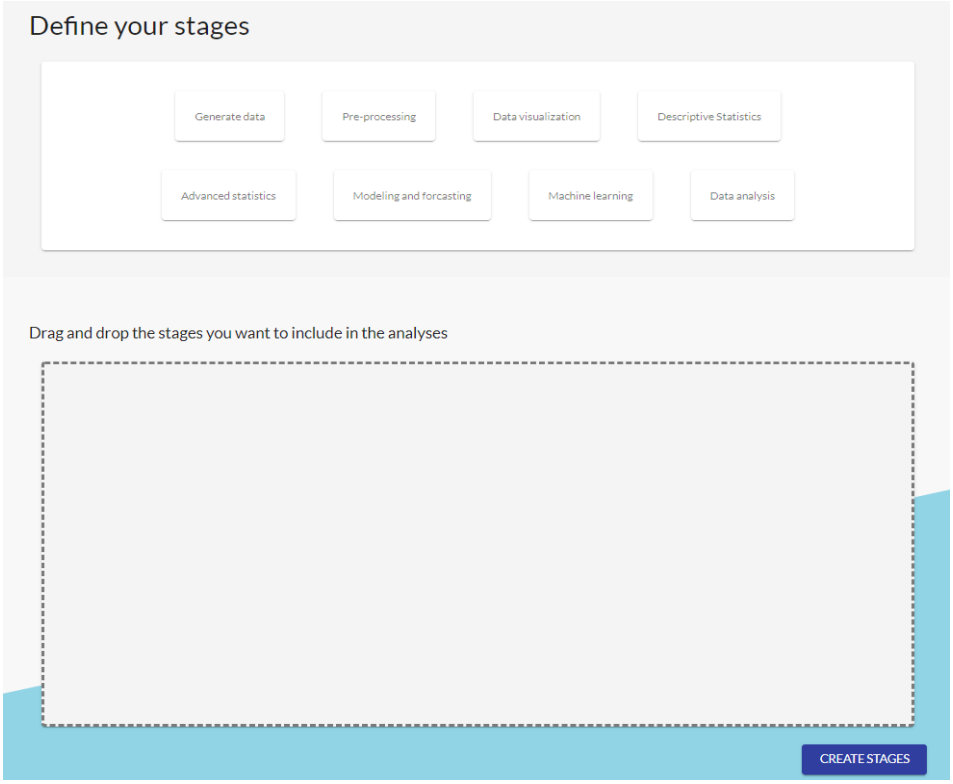


Figure 6: Drag and drop menu of the platform for defining the stage of the workflow

After the stages are defined, the data and code files used in each of them can be uploaded. The user can specify the parameters and the expected outputs for each stage. From the dropdown menu “INPUTS” and “OUTPUTS”, the user can choose, respectively the default inputs and outputs parameters of the stage related to the selected Jupyter notebook. The default input parameters can be overridden by the user. She/he can pass as value either the location of a data file, or an output from another stage or an inline value, see Figure 7.

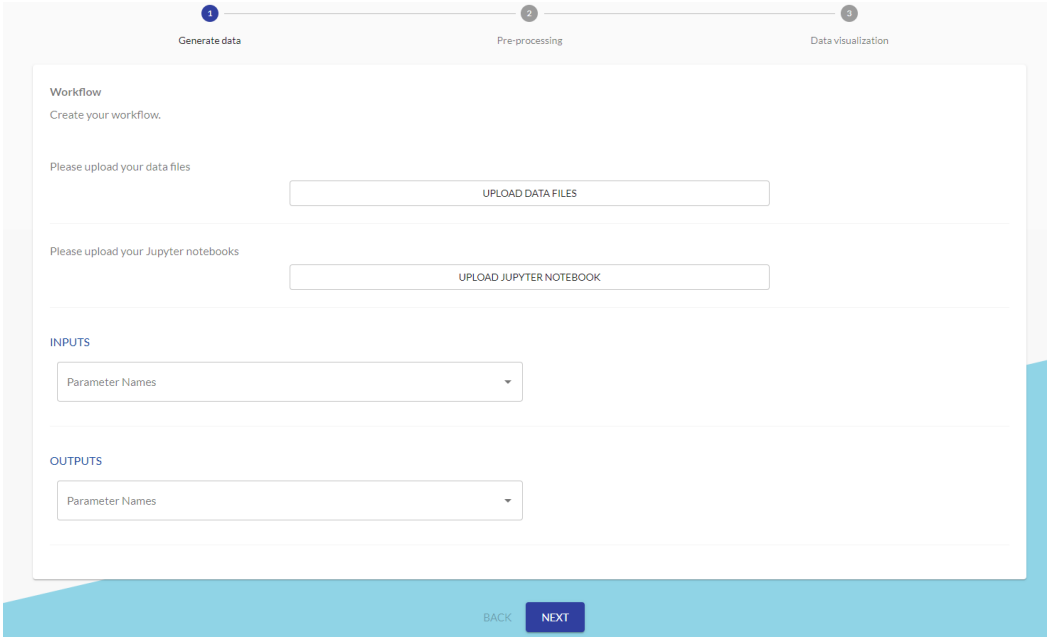


Figure 7: The page of the platform for defining the information for each stage of the analysis

Finally, the workflow, i.e. the REANA YAML file, is automatically created. In Figure 8 the YAML file is shown in the bottom part. The user will be able to use this file to execute its analysis.

A graph in the upper part of Figure 8, has been provided as well to visualize the workflow in a more friendly way. In a few words, each node of the graph depicts a stage in the workflow. If a stage is dependent on a previous one, waiting for some output, the stages are connected with an edge.

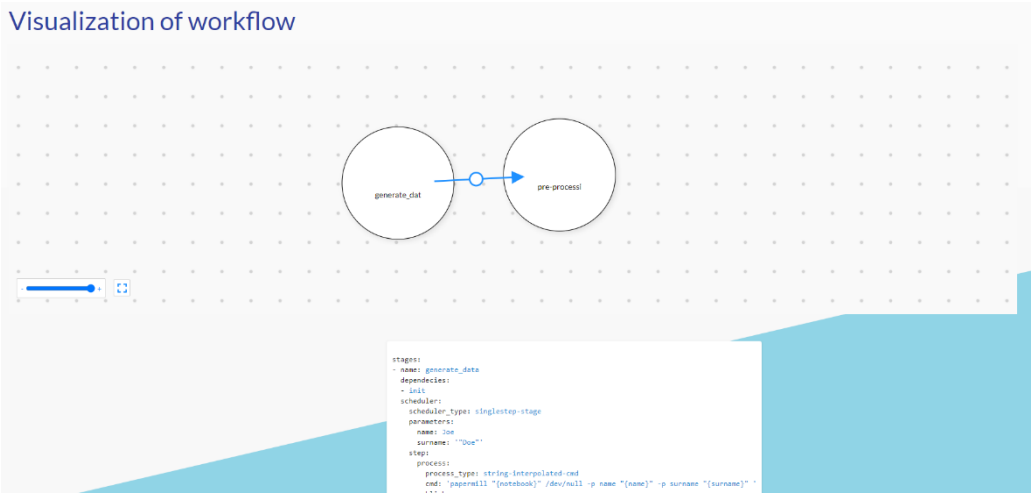


Figure 8: Visualization of the created workflow.

## 6. CONCLUSION

REANA offers the ability to make reproducible the research generated by the scientific community, by providing a systematic approach through a platform that integrates different technologies. But the actual challenge in enhancing the reproducibility in the scientific community will be the adoption of these tools and practises. A combination of policymaking and technological support for more user-friendly and intuitive tools are needed in order to foster actual widespread reproducibility. The work presented in this report, proposes, and implements an innovative attempt in that direction. APIs were built to provide a higher level of communication between REANA technologies and a user-friendly web interface. An additional effort has been made to investigate how to guide users to deploy reproducible analyses without significantly increasing the time invested in it.

The presented approach represents a viable solution for everyday practises. Nevertheless, it can be extended with additional crucial features. Currently, the API responsible for extending a base environment with additional packages supports only the installation of system or Python packages. Furthermore, the automated workflows can execute only Jupyter notebooks, restricting the use cases of the platform. The possibility to execute different source code formats, like Java, C++ or Python code files and support for different technologies when extending an environment will increase the scalability and practicality of the platform. The support for a wide range of technologies is also needed to enrich it with the flexibility to create diverse reproducible analyses.

## 7. REFERENCES


1. https://github.com/bugout-dev/mirror/blob/master/notebooks/rise-of-github.ipynb
2. Too much academic research is being published, (Philip G Altbach and Hans de Wit 07 September 2018)
3. https://www.statista.com/statistics/871513/worldwide-data-created/
4. REANA: A System for Reusable Research Data Analyses (Tibor Šimko, Lukas Heinrich , Harri Hirvonsalo , Dinos Kousidis , and Diego Rodríguez)
5. https://reana.cern.ch/
6. https://docs.docker.com/
7. K. Cranmer, L. Heinrich, “Yadage and Packtivity — analysis preservation using parametrized workflows” (2017), arXiv:1706.01878.
8. P. Amstutz, M. R. Crusoe, N. Tijanic (editors), B. Chapman, J. Chilton, M. Heuer, ´ A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, M. Scales, S. Soiland-Reyes, L. Stojanovic, “Common Workflow Language, v1.0” Specification, Common Workflow Language working group (2016), doi:10.6084/m9.figshare.3115156.9.v2.
9. https://github.com/reanahub/reana-env-root6
10. https://docs.opencv.org/4.5.3/d4/d86/group__imgproc__filter.html
11. https://ec.europa.eu/info/sites/default/files/research_and_innovation/reana.pdf