REANA Tutorial

Beginner Examples

Example 1

“Hello World!”

“Hello World!” – structure

The most simple code you can think of (**helloworld.py**)

```python
print("Hi from Reana!")
```

These are all the information we need to pass to REANA (**reana.yaml**):

```yaml
inputs:
  files:
    - helloworld.py
workflow:
  type: serial
  specification:
    steps:
      - environment: 'docker.io/library/python:3.10-bookworm'
        commands:
        - python helloworld.py
```

“Hello World!” – yaml file

The most simple code you can think of (**helloworld.py**)

```
print("Hi from Reana!")
```

These are all the information we need to pass to REANA:

```yaml
inputs:
  files:
    - helloworld.py
workflow:
  type: serial
  specification:
    steps:
      - environment: 'docker.io/library/python:3.10-bookworm'
        commands:
        - python helloworld.py
```

All input files and parameters to pass to REANA (in this case only the file with the python source code)

Property	Type	Mandatory?
directories	*list*	*optional*
files	*list*	*optional*
parameters	*dictionary*	*optional*
options	*dictionary*	*optional*

“Hello World!” – yaml file

The most simple code you can think of (**helloworld.py**)

```
print("Hi from Reana!")
```

These are all the information we need to pass to REANA:

```yaml
inputs:
  files:
    - helloworld.py
workflow:
  type: serial
  specification:
    steps:
      - environment: 'docker.io/library/python:3.10-bookworm'
        commands:
        - python helloworld.py
```

Computational steps necessary for the analysis:

- **language <u>type</u> can be cwl, serial, yadage, snakemake**

“Hello World!” – yaml file

The most simple code you can think of (**helloworld.py**)

```python
print("Hi from Reana!")
```

These are all the information we need to pass to REANA:

```yaml
inputs:
  files:
    - helloworld.py
workflow:
  type: serial
  specification:
    steps:
      - environment: 'docker.io/library/python:3.10-bookworm'
        commands:
        - python helloworld.py
```

All the workflow steps to be run sequentially to obtain the results:

- **environment: runtime container image where the workflow step commands will be run**

“Hello World!” – yaml file

The most simple code you can think of (**helloworld.py**)

```
print("Hi from Reana!")
```

These are all the information we need to pass to REANA:

```yaml
inputs:
  files:
    - helloworld.py
workflow:
  type: serial
  specification:
    steps:
      - environment: 'docker.io/library/python:3.10-bookworm'
        commands:
        - python helloworld.py
```

All the workflow steps to be run sequentially to obtain the results:

- **environment: runtime container image where the workflow step commands will be run**
- **all the commands to be run in the environment container when the given workflow step is executed. Each command is executed as a separate containerised job**

“Hello World!” – yaml file

The most simple code you can think of (**helloworld.py**)

```python
print("Hi from Reana!")
```

These are all the information we need to pass to REANA:

```yaml
inputs:
  files:
    - helloworld.py
workflow:
  type: serial
  specification:
    steps:
      - environment: 'docker.io/library/python:3.10-bookworm'
        commands:
        - python helloworld.py
```

Full docs on yaml files:
https://docs.reana.io/reference/reana-yaml/

“Hello World!” – run on REANA

After installation, `reana-client` should be ready to run.

Test the connection with:

```
reana-client ping
```

You should get something like:

REANA server: https://reana-p4n.aip.de
REANA server version: 0.9.2
REANA client version: 0.9.2
Authenticated as: XXX <xxx@xxx.xx>
Status: Connected

“Hello World!” – run on REANA

To check whether your yaml file is valid, you can run:

```
reana-client validate
```

If all is correct, you will get:

```
==> Verifying REANA specification file... <path>/reana.yaml
  -> SUCCESS: Valid REANA specification file.
==> Verifying REANA specification parameters...
  -> SUCCESS: REANA specification parameters appear valid.
==> Verifying workflow parameters and commands...
  -> SUCCESS: Workflow parameters and commands appear valid.
==> Verifying dangerous workflow operations...
  -> SUCCESS: Workflow operations appear valid.
```

“Hello World!” – run on REANA

To create a new workflow with your desired name, run:

```
reana-client create -n <WF_name>
```

You can set the environment variable `REANA_WORKON` to set the default name:

```
export REANA_WORKON=<WF_name>
```

Then upload all the files specified in the yaml file to the workspace:

```
reana-client upload
```

And start the analysis:

```
reana-client start
```

ALTERNATIVE! These steps can be called all together with:

```
reana-client run -w <WF_name>
```

“Hello World!” – run on REANA

Two other useful commands are:

`reana-client status`

to verify the job status (this one should be finished quickly!)

`reana-client logs`

to check all logs and terminal output

Full docs on `reana-client`: https://reana-client.readthedocs.io/

Go to the web interface (https://reana-p4n.aip.de/) to see all the files uploaded, job logs, and workflow status!

Example 2

Plot a sine function

Sine Plot – structure

INPUT:

- python script `sine_plot.py`

ANALYSIS:

- create a sine function
- plotting the results

Sine Plot – code

sine_plot.py

```python
import numpy as np
import matplotlib.pyplot as plt

# Generate x values from 0 to 2*pi with 100 points
x_values = np.linspace(0, 2 * np.pi, 100)

# Generate y values using the sine function
y_values = np.sin(x_values)

# Create a plot for the sine function
plt.plot(x_values, y_values, label='sin(x)')

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Sine Plot')

# Add a legend
plt.legend()

# Save the plot as a PNG file
plt.savefig('sine_plot.png')

```

reana.yaml

```yaml
version: 0.9.0
inputs:
  files:
    - sine_plot.py
workflow:
  type: serial
  specification:
    steps:
      - environment: 'docker.io/library/python:3.10-bookworm'
        commands:
          - python sine_plot.py
```

Sine Plot – run on REANA

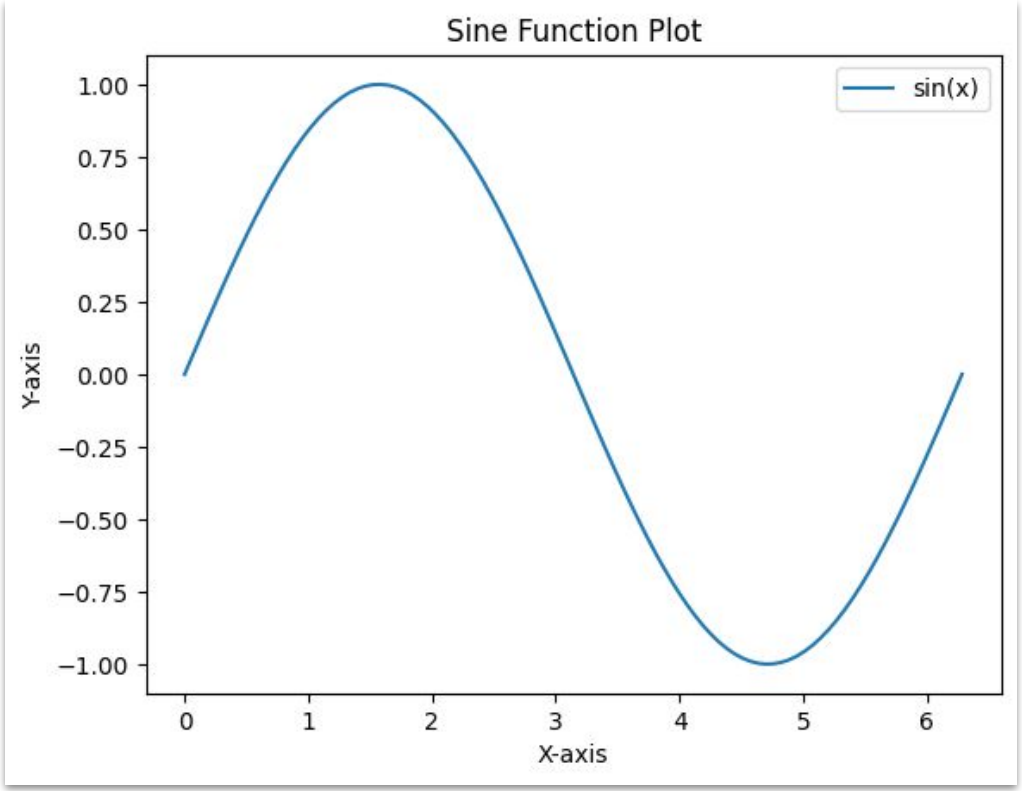
We can run the workflow as usual:

```
reana-client create -n <WF_name>
export REANA_WORKON=<WF_name>
reana-client upload
reana-client start
```

OR:

```
reana-client run -w <WF_name>
```

Sine Plot – result



Intermediate Examples

Example 1

Upload & Download files

Up/Download – structure

INPUT:

- Two .csv tables

ANALYSIS:

- Reading and matching the tables
- Saving and plotting the results

OUTPUT:

- Table with common object
- Plot

Up/Download – code

up_down.py

```python
import pandas as pd
import matplotlib.pyplot as plt

# Read the result of 2 Gaia queries with pandas

table1 = pd.read_csv('gaia_table_1.csv')
table2 = pd.read_csv('gaia_table_2.csv')

# Find common objects

table_match = pd.merge(table1, table2, on='source_id')

# Save plot of the results

fig, ax = plt.subplots(1,1, figsize=(7,5))

ax.plot(table1.ra, table1.dec, '.', label='Table 1')
ax.plot(table2.ra, table2.dec, '.', label='Table 2')
ax.plot(table_match.ra_x, table_match.dec_x, 'o', mfc='None', label='Match')
ax.invert_xaxis()
ax.set_xlabel('RA')
ax.set_ylabel('Dec')
ax.legend()

fig.tight_layout()
plt.savefig('results/table_match.png', format='png', dpi=150)

# Save the results in a new table

table_match.to_csv('results/table_match.csv', index=False)
```

reana.yaml

```yaml
inputs:
  files:
    - updown.py
    - gaia_table_1.csv
    - gaia_table_2.csv
workflow:
  type: serial
  specification:
    steps:
      - environment: 'jupyter/scipy-notebook'
        commands:
        - mkdir -p results
        - python updown.py
outputs:
  files:
    - results/table_match.png
    - results/table_match.csv
```

Up/Download – code

up_down.py

```python
import pandas as pd
import matplotlib.pyplot as plt

# Read the result of 2 Gaia queries with pandas

table1 = pd.read_csv('gaia_table_1.csv')
table2 = pd.read_csv('gaia_table_2.csv')

# Find common objects

table_match = pd.merge(table1, table2, on='source_id')

# Save plot of the results

fig, ax = plt.subplots(1,1, figsize=(7,5))

ax.plot(table1.ra, table1.dec, '.', label='Table 1')
ax.plot(table2.ra, table2.dec, '.', label='Table 2')
ax.plot(table_match.ra_x, table_match.dec_x, 'o', mfc='None', label='Match')
ax.invert_xaxis()
ax.set_xlabel('RA')
ax.set_ylabel('Dec')
ax.legend()

fig.tight_layout()
plt.savefig('results/table_match.png', format='png', dpi=150)

# Save the results in a new table

table_match.to_csv('results/table_match.csv', index=False)
```

reana.yaml

```yaml
inputs:
  files:
    - updown.py
    - gaia_table_1.csv
    - gaia_table_2.csv
workflow:
  type: serial
  specification:
    steps:
      - environment: 'jupyter/scipy-notebook'
        commands:
        - mkdir -p results
        - python updown.py
outputs:
  files:
    - results/table_match.png
    - results/table_match.csv
```

Now we also have an output section!

Up/Download – run on REANA

We can run the workflow as usual:

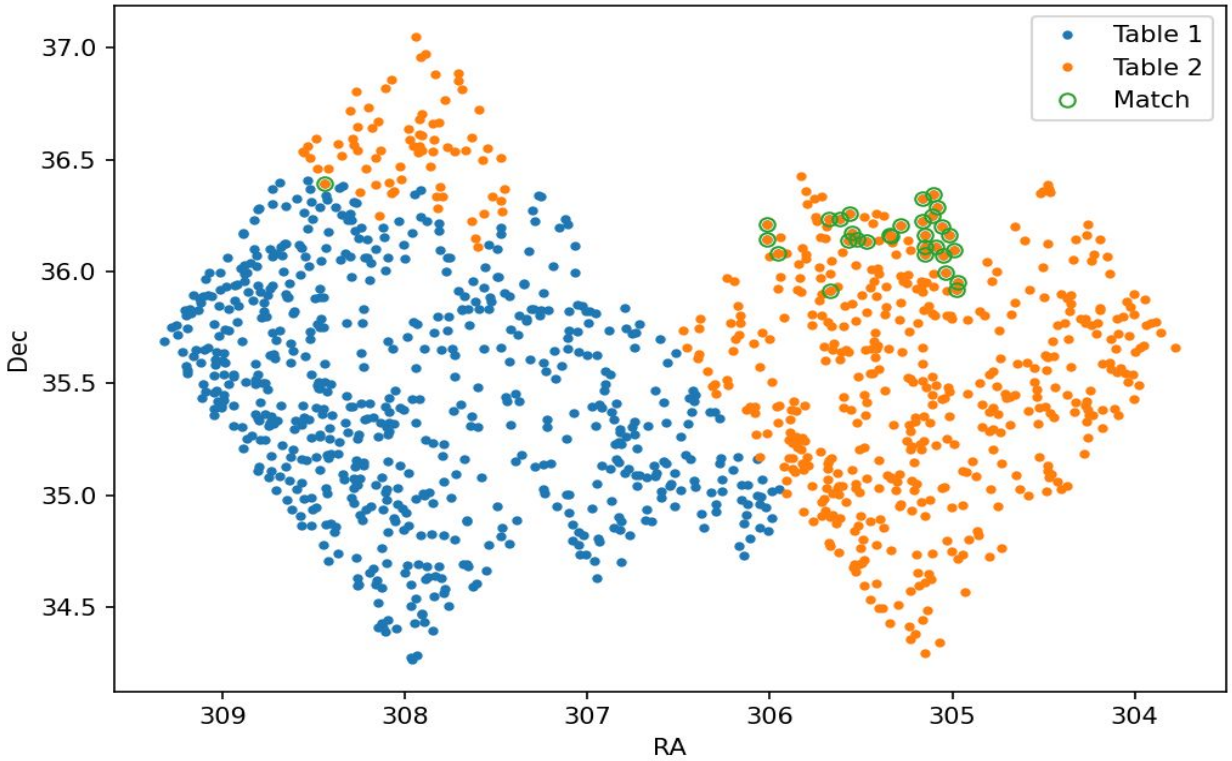
```
reana-client run -w <WF_name>
```

And then **download** the table and plot with:

```
reana-client download results -w <WF_name>
```

! If you set `export REANA_WORKON=<WF_name>`
there is no need to repeat the WF name.

Up/Download – results



Example 2

Access remote public data (S3 storage)

Remote data – structure

INPUT:

- Data from a public S3 bucket (link provided)

ANALYSIS:

- Reading the 10 tables into a single dataframe
- Plotting the results in aitoff galactic projection

OUTPUT:

- Plot

Remote data – yaml file

reana.yaml

```yaml
inputs:
  files:
    - remote_data.py
workflow:
  type: serial
  specification:
    steps:
      - environment: 'gitlab-p4n.aip.de:5005/p4nreana/reana-env:py311-astro.9845'
        commands:
        - mkdir -p results
        - python remote_data.py
outputs:
  files:
    - results/galactic_plot.png
```

Notice the customized environment!

Remote data – run on REANA

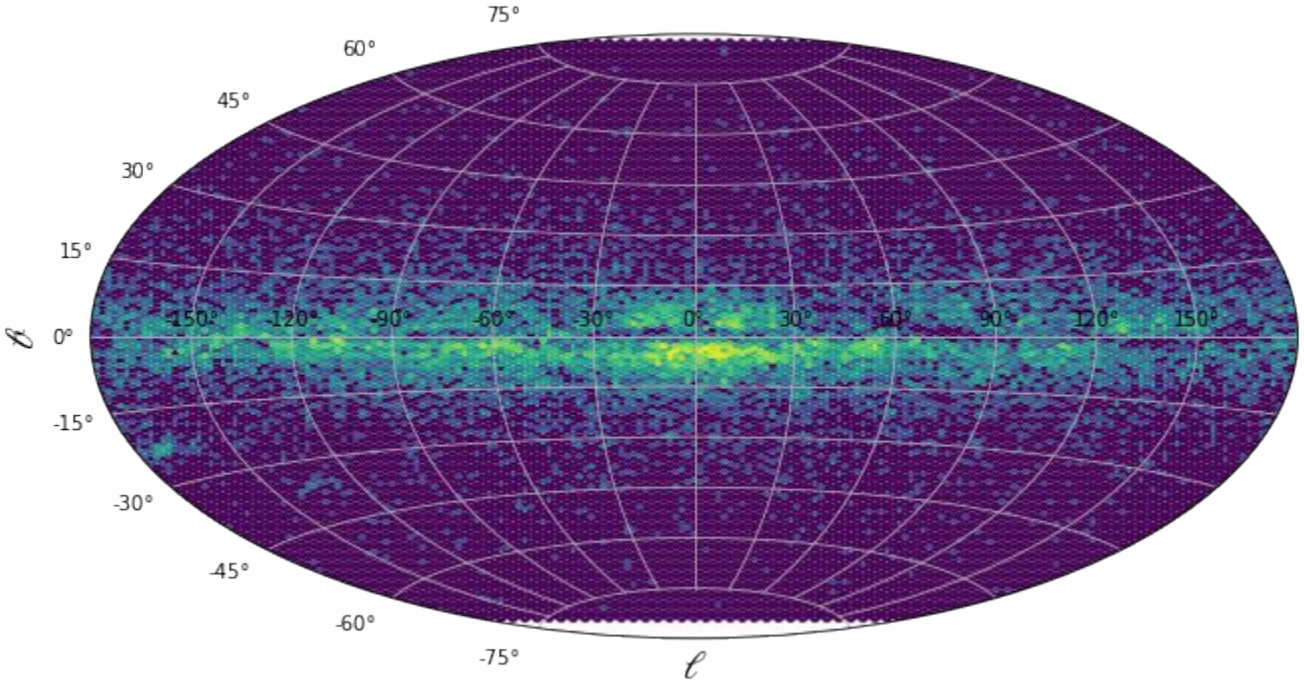
We can run the workflow as usual:

```
reana-client run -w <WF_name>
```

And then download the plot with:

```
reana-client download results -w <WF_name>
```

Remote data – results



Example 3

Data from TAP/VO queries

TAP query

- **Table Access Protocol**

  - a service protocol for accessing general table data, including astronomical catalogs as well as general database tables

- **APPLAUSE**

  - archives of digitized photographic plates, along with metadata about scientific content the plates, digitized observation logbooks, and table data extracted from the images

  - distribution of extracted source detections in the sky

- **SQL query**

TAP query – structure

INPUT

- data from TAP query
- python script `plotplates.py`

ANALYSIS:

- retrieve the data
- prepare archive access
- query archive table for all available archives.
- loop through the archives

OUTPUT:

- `archive_id.csv`
- plot

TAP query – code

plot_plates.py

```
# Prepare archive access
x_url = 'https://www.plate-archive.org/tap'
tap_session = requests.Session()
tap_service = vo.dal.TAPService(x_url, session=tap_session)
lang='PostgreSQL'

# Query archive table for all available archives
qry = "Select archive_id, archive_name, num_plates from applause_dr4.archive order by archive_id"
tap_result = tap_service.run_sync(qry, language=lang)
dfa = tap_result.to_table().to_pandas()
dfa.to_csv('archive_id.csv', index=False)
```

TAP query – code

You can change this to any archive_id of your wish from previously created archive_id.csv file

```
# Loop through the archives
for index, row in dfa.iterrows():
    if(row['archive_id']==401):
        create_plot(row['archive_id'], row['archive_name'], row['num_plates'])

```

TAP query – code

reana.yaml

```yaml
version: 0.9.0
inputs:
  files:
  - plotplates.py
workflow:
  type: serial
  specification:
    steps:
        - environment: 'gitlab-p4n.aip.de:5005/p4nreana/reana-env:py311-astro.10125'
          commands:
          - mkdir -p imgdr4
          - python plotplates.py
outputs:
  files:
    - archive_id.csv
    - imgdr4/dr4_archive_401.png
```

TAP query – code

reana.yaml

```yaml
version: 0.9.0
inputs:
  files:
  - plotplates.py
workflow:
  type: serial
  specification:
    steps:
        - environment: 'gitlab-p4[illegible]ip.de:5005/p4nreana/reana-env:py311-astro.10125'
          commands:
          - mkdir -p imgdr4
          - python plotplates.py
outputs:
  files:
    - archive_id.csv
    - imgdr4/dr4_archive_401.png
```

It makes a directory called imgdr4

TAP query – code

reana.yaml

```yaml
version: 0.9.0
inputs:
  files:
  - plotplates.py
workflow:
  type: serial
  specification:
    steps:
        - environment: 'gitlab-p4n.aip.de:5005/p4nreana/reana-env:py311-astro.10125'
          commands:
          - mkdir -p imgdr4
          - python plotplates.py
outputs:
  files:
    - archive_id.csv
    - imgdr4/dr4_archive_401.png
```

Saves plot in the created directory.

TAP query – result

archive_id	archive_name	num_plates
1	Zeiss Triplet (Potsdam)	4921
2	Carte du Ciel (Potsdam)	979
3	Great Schmidt Camera (Potsdam)	508
4	Small Schmidt Camera (Potsdam)	113
5	Ross Camera (Potsdam)	64
6	Einstein Turm Solar Plates	3613
101	Lippert-Astrograph (Hamburg)	8750
102	Grosser Schmidt-Spiegel (Hamburg)	5323
103	1m-Spiegelteleskop (Hamburg)	7643
104	Hamburger Schmidt-Spiegel (Calar Alto)	3255

TAP query – run on REANA

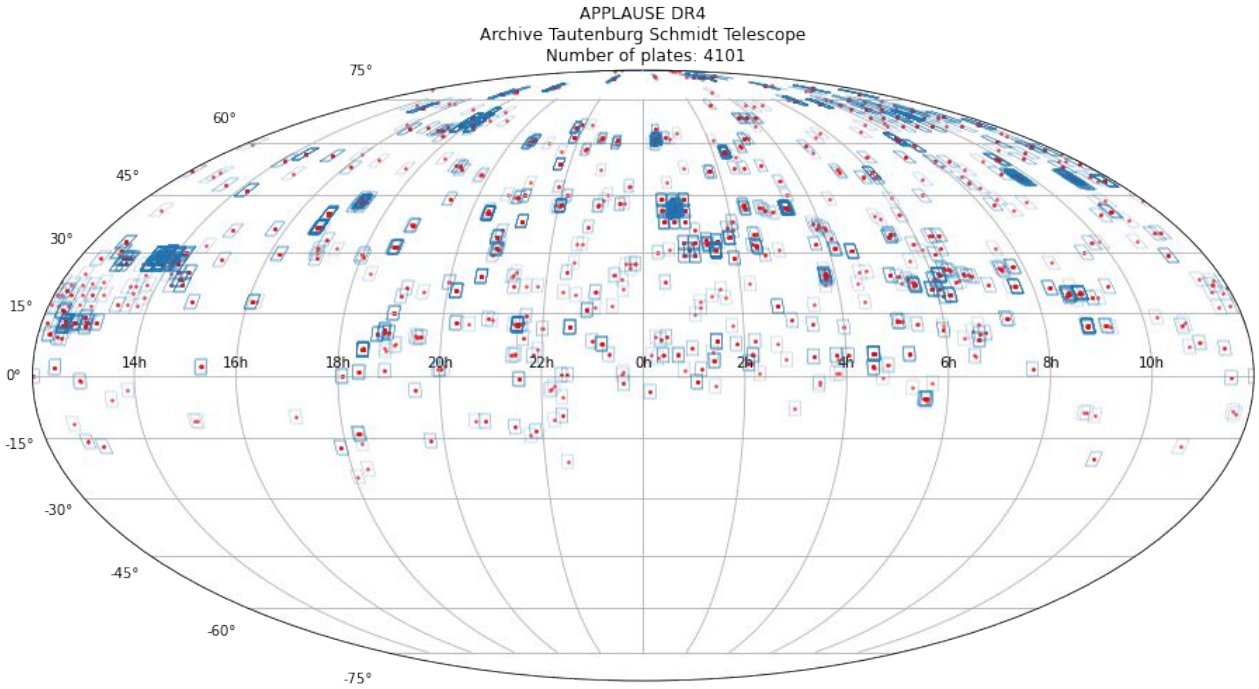
We can run the workflow as usual:

```
reana-client run -w <WF_name>
```

And then download the plot with:

```
reana-client download imgdr4 -w <WF_name>
```

TAP query – results



Advanced Examples

Example 1

Using Jupyter Notebooks on REANA

Using Notebooks – structure

We will use the same code as the Remote Data example (intermediate/ex2) but within an interactive Jupyter Notebook instead of a pure python script.

In order to do so on REANA, we will need papermill, a python library that allows to parameterize and execute notebooks.

Papermill takes a **source notebook**, applies some **parameters** to it, executes the notebook with the specified kernel, and saves the output in the **destination notebook**.

Using Notebooks – structure

The source notebook needs to contain a cell tagged **parameter** where we specify some default values which may be overridden by values specified at execution time (in this case the path to the output plot.

Each parameter can be passed with the flag `-p`.

See more here: https://papermill.readthedocs.io/en/latest/usage-parameterize.html

Using Notebooks – yaml file

```yaml
inputs:
  files:
    - remote_data.ipynb
  parameters:
    notebook_in: remote_data.ipynb
    notebook_out: results/output_notebook.ipynb
    output_plot: results/galactic_plot.png
workflow:
  type: serial
  specification:
    steps:
      - environment: 'gitlab-p4n.aip.de:5005/p4nreana/reana-env:py311-astro.9845'
        commands:
          - mkdir -p results
          - papermill ${notebook_in} ${notebook_out} -p output_file ${output_plot} -k python3
outputs:
  files:
    - results/galactic_plot.png
```

These we need to define and later pass them to papermill

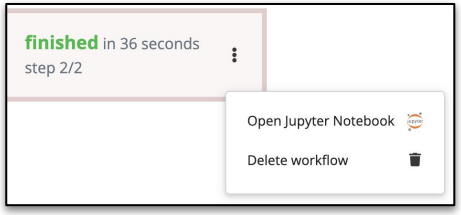
Using Notebooks – yaml file

```yaml
inputs:
  files:
    - remote_data.ipynb
  parameters:
    notebook_in: remote_data.ipynb
    notebook_out: results/output_notebook.ipynb
    output_plot: results/galactic_plot.png
workflow:
  type: serial
  specification:
    steps:
      - environment: 'gitlab-p4n.aip.de:5005/p4nreana/reana-env:py311-astro.9845'
        commands:
          - mkdir -p results
          - papermill ${notebook_in} ${notebook_out} -p output_file ${output_plot} -k python3
outputs:
  files:
    - results/galactic_plot.png
```

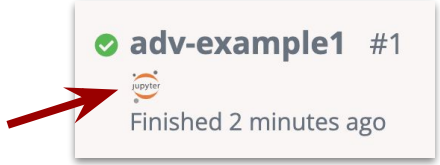
This we need to avoid conflicts with REANA's default kernel

Using Notebooks – run on REANA

After running the analysis through REANA with the usual commands, we can start a Jupyter Notebook by clicking on the 3 dots and selecting Open Jupyter Notebook.



After waiting a few seconds, we can click on the new notebook image that should have appeared next to the workflow name.



Notice that this notebook runs in a **different environment** compared to the one specified in the yaml file, so we might need to reinstall some libraries (see the first commented cell in the notebook).

<u>Check the differences between input and output notebook!</u>

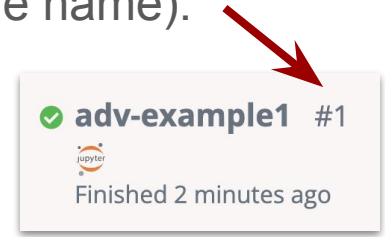
Using Notebooks – run on REANA

Alternatively, you can open the notebook from command line and specify a custom image, in this case:

```
reana-client open -w adv-example1.1 -i
gitlab-p4n.aip.de:5005/p4nreana/reana-env:py311-astro.9845 jupyter
```

Notice you need to use the **workflow name** you chose when running the analysis (here "adv-example1") with the correct **tag** (the progressive number automatically added by REANA that you can see on the web interface next to the name).

You can open notebooks in both ways for all your workflows.



Example 2

Create an image for a custom environment

Create an image – requirements

The first step is to figure out all you need inside your image, e.g. the list of python libraries required for your analysis to run.

We will put this list in a file called **requirements.txt**, e.g.:

```
scipy
numpy
pandas
matplotlib
astropy
seaborn
papermill
```

Create an image – definition

Then we need to define the image in a file called **.gitlab-ci.yml** that will contain different GitLab variables:

- `$CI_REGISTRY_IMAGE`: the address of the project's Container Registry (in this case the gitlab repository where we are building the image, e.g., gitlab-p4n.aip.de:5005/p4nreana/reana-env)
- `$CI_COMMIT_REF_SLUG`: the branch or tag name where the project is built (e.g., py311-astro)
- `$CI_JOB_ID`: a serial number with the image version (e.g., 9845)

```
variables:
  IMAGE_TAG: $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG.$CI_JOB_ID
```

Create an image – definition

We call now the `build-push-docker-image-job` command and add these steps:

- `image`: a Docker image to run the job in
- `services`: additional services needed to run the job
- `before_script`: commands that should run before each job's script commands
- `script`: run all the commands to create the image

```
build-push-docker-image-job:
  # Specify a Docker image to run the job in.
  image: docker:20.10.16
  # Specify an additional image 'docker:dind' ("Docker-in-Docker") that
  # will start up the Docker daemon when it is brought up by a runner.
  services:
    - docker:20.10.16-dind
  before_script:
    - docker info
  script:
    - export
    - ls -latr
    - echo  $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER $CI_REGISTRY
    - docker build -t $IMAGE_TAG .                                          [...]
    - docker push $IMAGE_TAG
```

Create an image – Dockerfile

Finally, we need to create a file called **Dockerfile** containing:

- the base image, e.g. from a public repository like DockerHub or JupyterHub
- the packages we need, from **requirements.txt**
  (notice that Docker is a separate container, so we need to copy the file from our local machine to the container, using the command `FROM`)
- the commands to install everything on top of the base image (e.g. `pip`)

```
FROM jupyter/scipy-notebook

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
```

Create an image – CI/CD

Now we need to run git's Continuous Integration/Continuous Delivery pipeline.

This should happen automatically, given that we have a runner enables for this project. If not, on the left sidebar go to `settings > CI/CD > Runners`, expand and enable a runner.

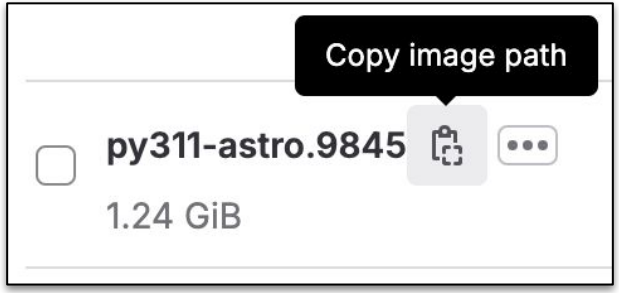
Wait until the pipeline finishes running, and if successful your image is created!

Create an image – use the new image!

Go to `Deploy > Container registry` and you should see a new tag with the name you defined and a version number.

Copy the image path and you can now use it as, e.g., a REANA environment. (It might be necessary to make it public if it's not.)

To run it locally, you can use the `docker run` command with the same path.



Example 3

Dimensionality reduction & access to private data

Dimensionality reduction – structure

We show an example of 3 different algorithms to perform dimensionality reduction:

- UMAP (Uniform Manifold Approximation and Projection)
- PCA (Principal Component Analysis)
- t-SNE (t-distributed Stochastic Neighbor Embedding)

The data are managed between different pipelines using S3 private storage:

- download data from remote TAP service
- analyze and plot the results
- save and upload to S3
- download plots from S3 if they exist
- combine them and save the result

Dimensionality reduction – REANA secrets

To access private S3 storage, we use `reana-secrets`, a way to store tokens in REANA environment. For S3, we need two keys, that we can add with:

```
reana-client secrets-add --env access_key=XXX
reana-client secrets-add --env secret_key=XXX
```

Now we can call them within the python script with:

```
os.environ['access_key']
os.environ['secret_key']
```

See more here:
https://docs.reana.io/reference/reana-client-cli-api/#secret-management-commands

Dimensionality reduction – yaml file

The inputs are the 2 python scripts and some useful parameters:

```yaml
inputs:
  files:
    - reduce.py
    - combine_plots.py
  parameters:
    user_folder: new_user
    n_test: 5
```

Dimensionality reduction – yaml file

The inputs are the 2 python scripts and some useful parameters:

```yaml
inputs:
  files:
    - reduce.py
    - combine_plots.py
  parameters:
    user_folder: new_user
    n_test: 5
```

Change this to your name

Dimensionality reduction – yaml file

The inputs are the 2 python scripts and some useful parameters:

```yaml
inputs:
  files:
    - reduce.py
    - combine_plots.py
  parameters:
    user_folder: new_user
    n_test: 5
```

Number of iterations

Dimensionality reduction – yaml file

The inputs are the 2 python scripts and some useful parameters:

```yaml
inputs:
  files:
    - reduce.py
    - combine_plots.py
  parameters:
    user_folder: new_user
    n_test: 5
```

The final output is the combined pdf with all plots from the `n_test` iterations:

```yaml
outputs:
  files:
    - results/merged_plots.pdf
```

Dimensionality reduction – yaml file

The data round trip is performed in 2 steps:

```yaml
workflow:
  type: serial
  specification:
    steps:
      - name: make-projections
        environment: 'gitlab-p4n.aip.de:5005/p4nreana/reana-env:py311-astro-ml.10134'
        commands:
          - mkdir -p results
          - python reduce.py -d ${user_folder} -n ${n_test}
      - name: combine-plots
        environment: 'gitlab-p4n.aip.de:5005/p4nreana/reana-env:py311-astro-ml.10134'
        commands:
          - python combine_plots.py -d ${user_folder} -n ${n_test}
```

This runs the 3 algorithms and uploads the plots on S3

Dimensionality reduction – yaml file

The data round trip is performed in 2 steps:

```yaml
workflow:
  type: serial
  specification:
    steps:
      - name: make-projections
        environment: 'gitlab-p4n.aip.de:5005/p4nreana/reana-env:py311-astro-ml.10134'
        commands:
          - mkdir -p results
          - python reduce.py -d ${user_folder} -n ${n_test}
      - name: combine-plots
        environment: 'gitlab-p4n.aip.de:5005/p4nreana/reana-env:py311-astro-ml.10134'
        commands:
          - python combine_plots.py -d ${user_folder} -n ${n_test}
```

This retrieves the plots from S3 and combines them