www.interpreter.eu

INTERPRETER

# SOFTWARE REPLICABILITY TOOLS AND METHODOLOGY

Deliverable D6.4

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 864360 (INTERPRETER). This output reflects only the author's view and the European Union cannot be held responsible for any use that may be made of the information contained therein.

Co-funded by the Horizon 2020
Framework Programme of the European Union

# Deliverable D6.4
# SOFTWARE REPLICABILITY TOOLS AND METHODOLOGY

Version 1.2

Organisation: CIRCE

Main authors: Aitor Alcrudo, Hans Bludszuweit

Date: 21/02/2021

# DELIVERABLE 6.4 – VERSION 1.2

## WORK PACKAGE N° 6 – Integration of modelling tool and SW applications in a common framework and existing platforms

## TASK N° 4 – External processes for migration of the results to standard platforms

| Nature of the deliverable | | |
|---|---|---|
| R | Document, report (excluding the periodic and final reports) | X |
| DEM | Demonstrator, pilot, prototype, plan designs | |
| DEC | Websites, patents filing, press & media actions, videos, etc. | |
| OTHER | Software, technical diagram, etc. | |

| Dissemination Level | | |
|---|---|---|
| PU | Public, fully open, e.g. web | X |
| CO | Confidential, restricted under conditions set out in Model Grant Agreement | |
| CI | Classified, information as referred to in Commission Decision 2001/844/EC | |

Document history

| Date | Version | Reviewer | Nature of the revision |
|---|---|---|---|
| 04/11/2021 | TOC | Aitor Alcrudo | Table of contents, first content added |
| 17/02/2022 | 1.0 | CIRCE | Final deliverable to review |
| 21/02/2022 | 1.1 | ATOS | Review |
| 21/02/2021 | 1.2 | CIRCE | Final comments addition and quality check |

## Disclaimer of warranties

*This report is part of the deliverables from the project INTERPRETER which has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 864360. More information on the project can be found at https://www.interpreter.eu*

## Executive summary

**D6.4 – Software replicability tools and methodology** is the main output of **Task 6.4 - External processes for migration of the results to standard platforms**. This document explains the approach taken to ease integration between components and the migration of these components to other servers.

To make integration between components easier without fundamentally changing the workflow of each partner, we proposed using automatic testing with Jenkins on the public endpoints of the components. This means we can detect breaking changes to interfaces as well as a standardized testbed to integrate against.

In order to facilitate the migration of components to different servers and environments the approach is to use container technology. Docker is the technology selected to develop the containers needed for the project. Docker provides a way of packaging not only components, but also all the dependencies of the component in a standard way. This means that each partner could develop their components in the programming language and environment of their choosing and still be able to run seamlessly in third party servers running the docker service capabilities and the access to the distributed storage systems.

## List of abbreviations

| Abbreviation | Full name |
|---|---|
| API | Application Programming Interface |
| CI/CD | Continuous Integration and Delivery |
| CSV | Comma separated values |
| GMT | Grid Modelling Tool |
| ICT | Information and Communications Technology |
| JSON | JavaScript Object Notation |
| LV | Low Voltage |
| NaN | Not a Number |
| REST | Representational State Transfer |
| WP | Work Package |

## Partners short names

CIRCE: FUNDACIÓN CIRCE CENTRO DE INVESTIGACIÓN DE RECURSOS Y CONSUMOS ENERGÉTICOS

RDN: CENTRO DE INVESTIGACAO EM ENERGIA REN - STATE GRID SA – R&D NESTER

CERTH: ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS

DTU: DANMARKS TEKNISKE UNIVERSITET

CARTIF: FUNDACION CARTIF

ATOS: ATOS SPAIN SA

ATOS IT: ATOS IT SOLUTIONS AND SERVICES IBERIA SL (LTP ATOS)

ORES: OPERATEUR DE RESEAUX D'ENERGIES

CUERVA: MONTAJES ELECTRICOS CUERVA S.L.

TT: TURNING TABLES S.L. (LTP CUERVA)

EQY: EUROQUALITY SARL

## List of Figures

# Contents

# 1 INTRODUCTION

The objective of the present deliverable is to describe the training tools and examples created in the project to make the replicability of the results easier, as well as to provide a set of recommendations and best practices. It also explains the use of Jenkins for automatic testing with the goal of making integration easier.

The focus is the usage of container technology, allowing the encapsulation of different execution environments in a way that is transparent to the deployment environment. Specifically, Docker [1] is used because it is a stablished technology and is also used in the FUSE platform.

## 1.1 RELEVANCE WITH OTHER TASKS

This deliverable is strongly related to WP3 (Tool for low-voltage grid modelling). The examples provided have served as a basis for the integration of some of the components of this WP, such as the LV modelling tool and the isolated areas algorithm. Software made to translate the data from the pilots, collected in Task 2.5 (Data collection from pilot networks selected for the use cases) into the Grid Modelling Tool (GMT) and Power Factory formats used in the WP7 (Testing and validation for selected use cases) is also encapsulated in Docker containers.

## 1.2 STRUCTURE OF THE DOCUMENT

In addition to the Executive Summary and the introduction sections, the deliverable has the following chapters:

- Chapter 2 – Discuses usage of the Jenkins automation server
- Chapter 3 – Containers General background information about container and docker as well introduction to examples and tutorials generated for the project
- Chapter 4 –Services provided by CIRCE as a Docker container.

# 2 APPROACH TO CONTINUOUS INTEGRATION

Continuous Integration and Delivery (CI/CD) is used to ease integration of new code and components using automation to test integrate and deploy component through the lifecycle of the software. It is commonly used together with agile development methodologies. The underlying idea is that by forcing integration at the earliest stage possible errors will be detected earlier.

Creating a CI/CD structure for a project would have required tackling it at the start of the project as well as a tighter integration between contributors. CI/CD requires changing the development methodology, which is hard, particularly with different contributors, each with different processes and work culture. In this case intellectual property protection concerns make it even harder to use CI/CD for the whole lifecycle of the project.

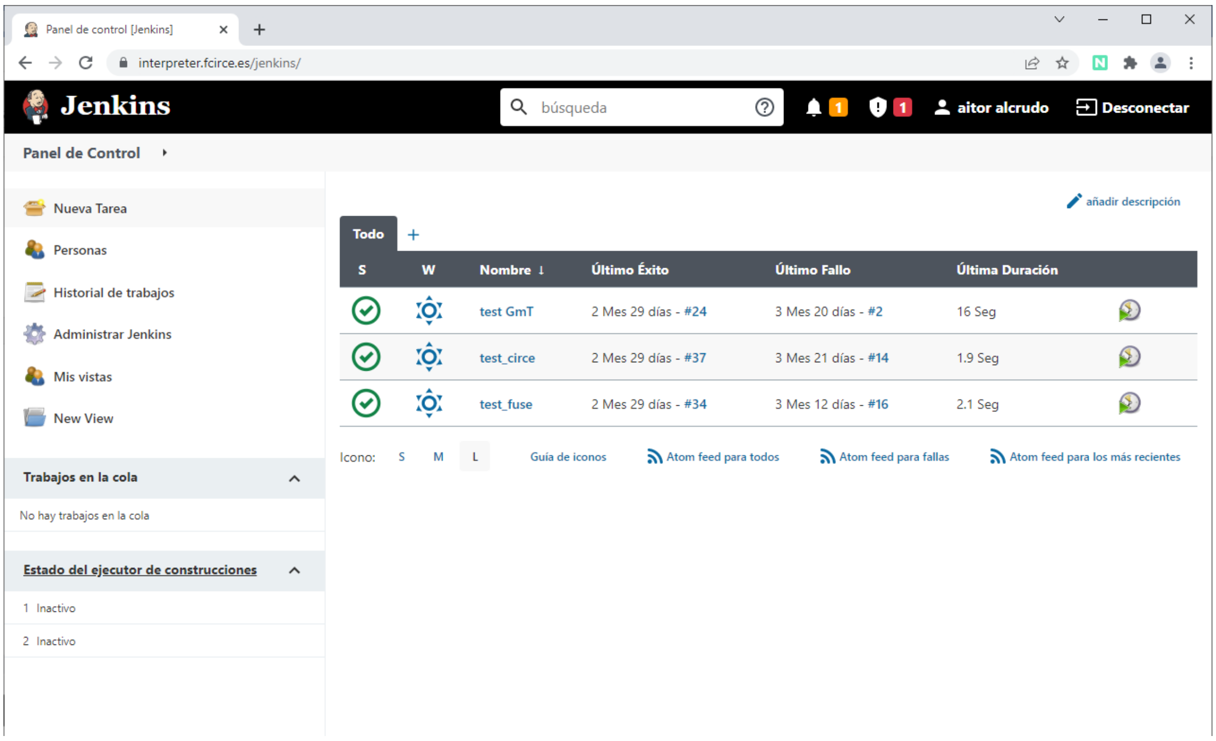
What has been done is to adapt some of the ideas of CI/CD to the project with practicality in mind. In practice, INTERPRETER focuses on automatically testing of the public endpoints and API documentation, as these areas would be useful in the project both at the current stage and in further steps. This in turn would ease the integration of the different components of the architecture.

A Jenkins automation server is deployed in order to perform automated testing of the web API endpoints of the project. These endpoints are critical because they are used by the different partners as a way of communicating different services and can be used and tested without having knowledge of the code of the different provides. The Jenkins [2] server provides daily information on the state of these services and of the test provided on them.

## 2.1 JENKINS SERVER

Jenkins is a self-contained, open-source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software [2]. In this case we are using Jenkins to automate testing of the web API endpoints of the project. This allows us to know the state of the API when doing integrations, as well as having early warning when an update to the endpoints creates an error. This is useful in helping to detect and solve issues in a fast way.

In Figure 1 it is shown a screenshot of the Jenkins server deployed in the CIRCE servers. It shows the state of the test for the GMT (Grid Modelling Tool), FUSE services and the test for the services deployed by CIRCE.


Panel de control [Jenkins]
interpreter.fcirce.es/jenkins/
Jenkins
búsqueda
aitor alcrudo
Desconectar
Panel de Control

Nueva Tarea
Personas
Historial de trabajos
Administrar Jenkins
Mis vistas
New View

Trabajos en la cola
No hay trabajos en la cola

Estado del ejecutor de construcciones
1 Inactivo
2 Inactivo

añadir descripción
Todo

| S | W | Nombre ↓ | Último Éxito | Último Fallo | Última Duración |
|---|---|---|---|---|---|
| ✓ | ☼ | test GmT | 2 Mes 29 días - #24 | 3 Mes 20 días - #2 | 16 Seg |
| ✓ | ☼ | test_circe | 2 Mes 29 días - #37 | 3 Mes 21 días - #14 | 1.9 Seg |
| ✓ | ☼ | test_fuse | 2 Mes 29 días - #34 | 3 Mes 12 días - #16 | 2.1 Seg |

Icono: S M L
Guía de iconos
Atom feed para todos
Atom feed para fallas
Atom feed para los más recientes

FIGURE 1: JENKINS SERVER

# 3 CONTAINERS AND DOCKER

A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. [3]

Containers encapsulate software in a way that is mostly transparent to the platform that is using it. It is ideal for projects with various suppliers and different technology stacks as it enables deployment in a single environment/server without cumbersome adaptations.

Containers can be migrated and/or replicated in different servers as-is without having to change the underlying software easing deployment in different clients/and or environments.

Docker is the most widespread container technology. It is integrated with several popular tools, and it has a vibrant ecosystem of free software containers which offers a huge variety of services such as databases, web servers and other tools. As an example, the Jenkins server can be run as a Docker container, and this is how it was done in the project.

With Docker also comes Docker-Compose which can be used to manage several containers as a single entity, orchestrating the communications and configurations between the containers.

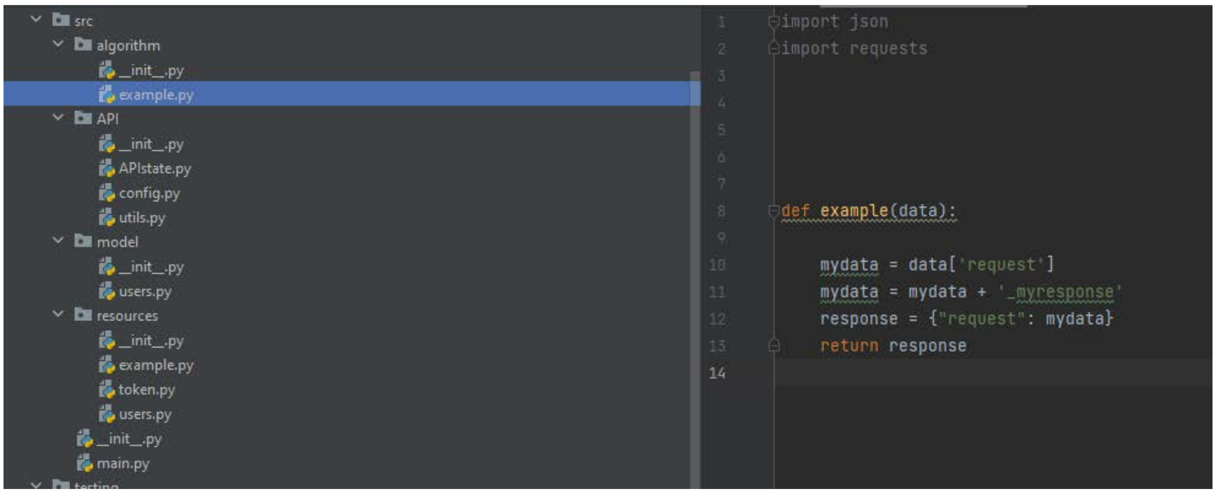
## 3.1 API EXAMPLE

As several partners have developed algorithms as standalone desktop applications without further integration, CIRCE prepared a basic example illustrating how to create a REST API which would allow to offer such algorithms as a service.

The example was developed in Python as it is one of the more popular languages for algorithm prototyping. Flask framework was used to provide a lightweight REST solution, database and authentication services.

CERTH used this example to create a REST API which offers the GMT LV grid modelling tool as a service.

The example includes an SQLite database, which is used to store user information, authentication management using tokens and an example endpoint which calls the example algorithm. In the Figure 2 a screenshot of the example project structure is presented.



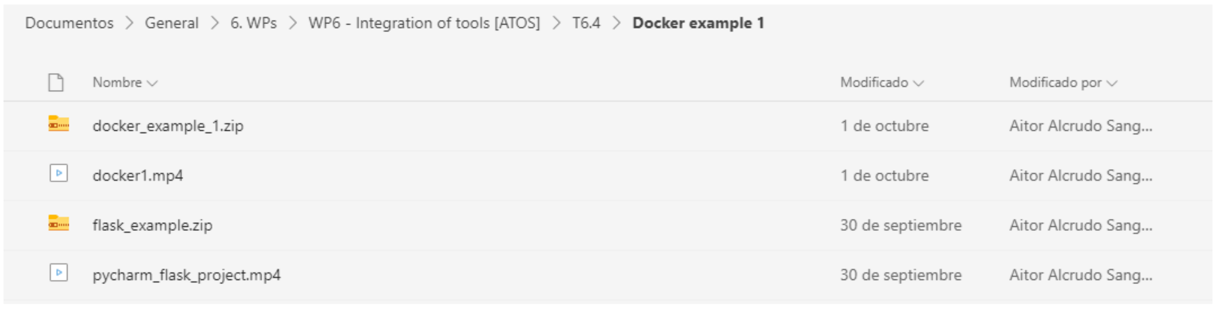
```
src
  algorithm
    __init__.py
    example.py
  API
    __init__.py
    APIstate.py
    config.py
    utils.py
  model
    __init__.py
    users.py
  resources
    __init__.py
    example.py
    token.py
    users.py
  __init__.py
  main.py
```

```python
import json
import requests




def example(data):

    mydata = data['request']
    mydata = mydata + '_myresponse'
    response = {"request": mydata}
    return response
```

FIGURE 2: API EXAMPLE

The example is available to all partners in the project repository for their use. An explanatory video tutorial was made providing guidelines on how to use and expand the example. It includes details on how to adapt the example to algorithms, how to adapt the configuration such as changing the database location or how to further use the database capabilities of the framework. The video also includes comments on how to test the REST API using Postman [4].

In the Figure 3 we can see the location of the example code inside the project SharePoint available to all partners.


Documentos > General > 6. WPs > WP6 - Integration of tools [ATOS] > T6.4 > **Docker example 1**

| Nombre | Modificado | Modificado por |
|---|---|---|
| docker_example_1.zip | 1 de octubre | Aitor Alcrudo Sang... |
| docker1.mp4 | 1 de octubre | Aitor Alcrudo Sang... |
| flask_example.zip | 30 de septiembre | Aitor Alcrudo Sang... |
| pycharm_flask_project.mp4 | 30 de septiembre | Aitor Alcrudo Sang... |

FIGURE 3: EXAMPLE LOCATION

## 3.2 DOCKER EXAMPLE

The Docker example adds to the API example by showing how to deploy the previous example as a Docker service. It explains how to create a configuration file that builds all the components and coordinates them. The code is available in the repository as well as an explanatory video that outlines how to use the example and how to expand upon it.

It explains basic usage of Docker-compose, leveraging on both "of the self" public containers and containers purpose built for the project to create a REST API. It includes how to build containers and how to start and stop them. In the Figure 4 the content of the example Docker-compose file is presented.

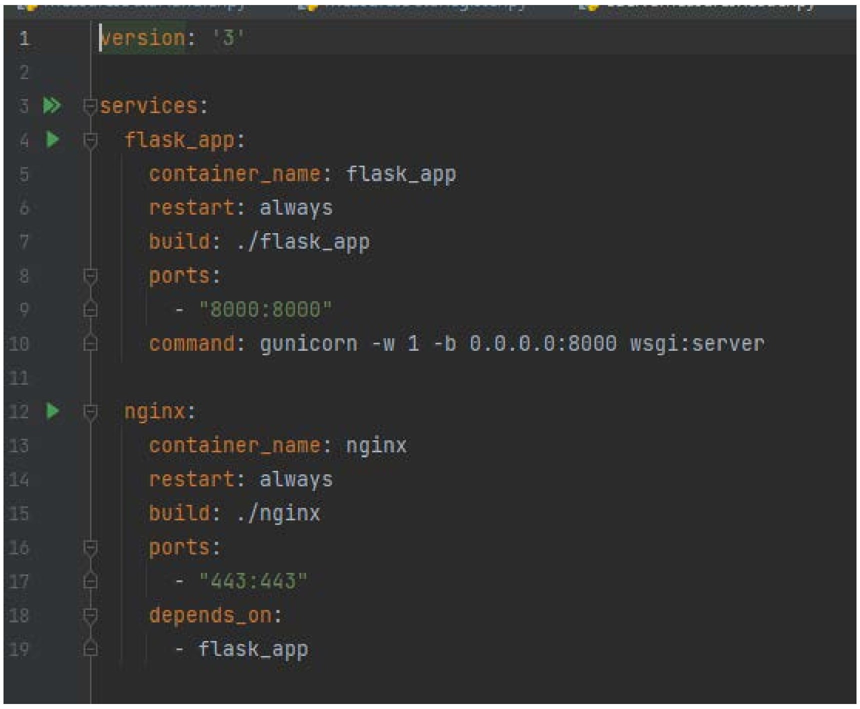
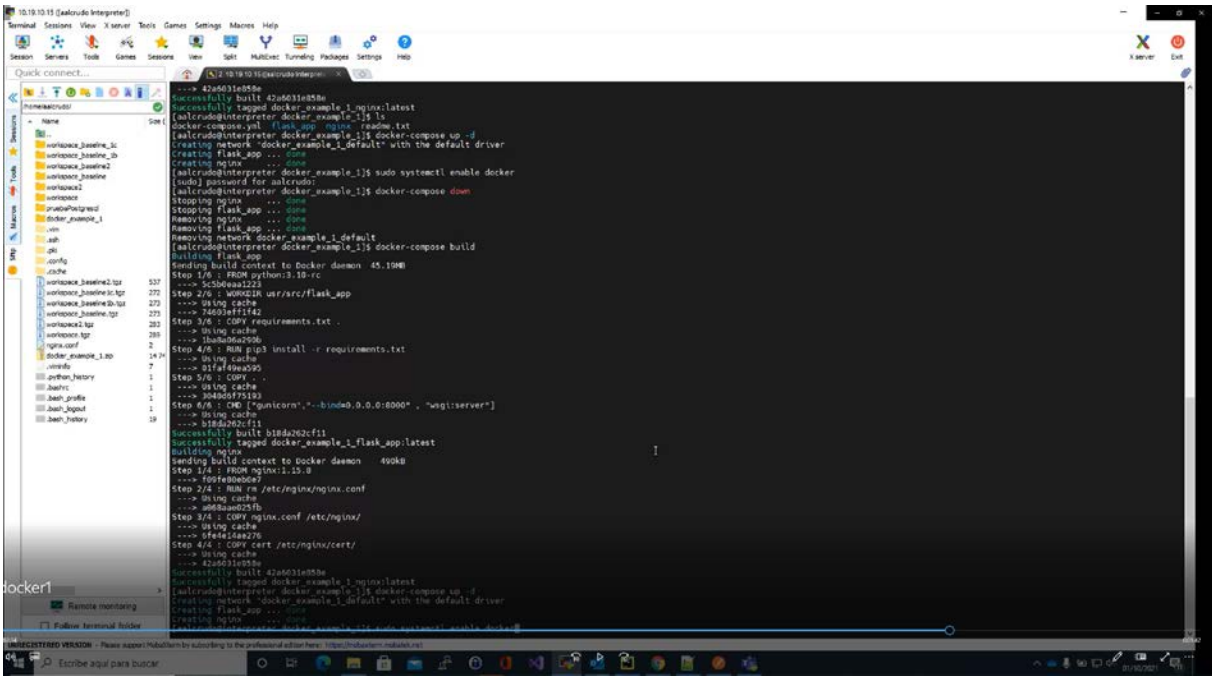

```
version: '3'

services:
  flask_app:
    container_name: flask_app
    restart: always
    build: ./flask_app
    ports:
      - "8000:8000"
    command: gunicorn -w 1 -b 0.0.0.0:8000 wsgi:server

  nginx:
    container_name: nginx
    restart: always
    build: ./nginx
    ports:
      - "443:443"
    depends_on:
      - flask_app
```

FIGURE 4: EXAMPLE DOCKERFILE

Figure 5 presents a screenshot of the video tutorial made for the Docker example in which its use is explained. How to build the example on a server and how to start and stop it.



```
 ---> 42a6031e858e
Successfully built 42a6031e858e
Successfully tagged docker_example_1_nginx:latest
[aalcrudo@interpreter docker_example_1]$ ls
docker-compose.yml  flask_app  nginx  readme.txt
[aalcrudo@interpreter docker_example_1]$ docker-compose up -d
Creating network "docker_example_1_default" with the default driver
Creating flask_app ... done
Creating nginx     ... done
[aalcrudo@interpreter docker_example_1]$ sudo systemctl enable docker
[sudo] password for aalcrudo:
[aalcrudo@interpreter docker_example_1]$ docker-compose down
Stopping nginx     ... done
Stopping flask_app ... done
Removing nginx     ... done
Removing flask_app ... done
Removing network docker_example_1_default
[aalcrudo@interpreter docker_example_1]$ docker-compose build
Building flask_app
Sending build context to Docker daemon  45.19MB
Step 1/6 : FROM python:3.10-rc
 ---> 5c5b0eaa1223
Step 2/6 : WORKDIR usr/src/flask_app
 ---> Using cache
 ---> 74603eff1f42
Step 3/6 : COPY requirements.txt .
 ---> Using cache
 ---> 1ba8a06a290b
Step 4/6 : RUN pip3 install -r requirements.txt
 ---> Using cache
 ---> 01faf49ea595
Step 5/6 : COPY . .
 ---> Using cache
 ---> 3040d6f75193
Step 6/6 : CMD ["gunicorn","--bind=0.0.0.0:8000" , "wsgi:server"]
 ---> Using cache
 ---> b18da262cf11
Successfully built b18da262cf11
Successfully tagged docker_example_1_flask_app:latest
Building nginx
Sending build context to Docker daemon    490kB
Step 1/4 : FROM nginx:1.15.8
 ---> f09fe80eb0e7
Step 2/4 : RUN rm /etc/nginx/nginx.conf
 ---> Using cache
 ---> a068aae025fb
Step 3/4 : COPY nginx.conf /etc/nginx/
 ---> Using cache
 ---> 6fe4e14ae276
Step 4/4 : COPY cert /etc/nginx/cert/
 ---> Using cache
 ---> 42a6031e858e
Successfully built 42a6031e858e
Successfully tagged docker_example_1_nginx:latest
[aalcrudo@interpreter docker_example_1]$ docker-compose up -d
Creating network "docker_example_1_default" with the default driver
Creating flask_app ... done
Creating nginx     ... done
[aalcrudo@interpreter docker_example_1]$ sudo systemctl enable docker
```

FIGURE 5: VIDEO TUTORIAL SCREENSHOT

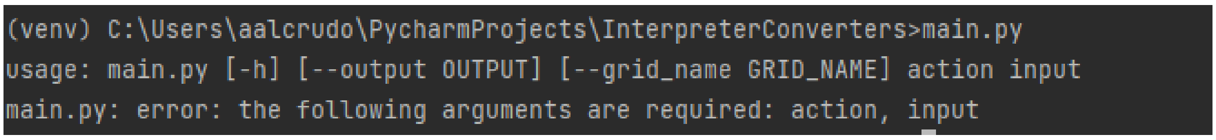
## 4 DOCKERIZED SERVICES

This section introduces some of the services that have been dockerized during de project. Those developed for the platform integration and for the internal cloud operation.

### 4.1 SERVICES DEVELOPED FOR PLATFORM INTEGRATION

Software for data management has been developed keeping in mind the integration with cloud platforms such as FUSE. It offers functionalities to translate data formats, converting the pilot information in their native formats to the agreed project format, that is a JSON based format. It also allows to transform the project data in the JSON format, to a CSV format, which is the one used by PowerFactory.

Executing the code without any options will yield a *help message* as seen on Figure 6:



```
(venv) C:\Users\aalcrudo\PycharmProjects\InterpreterConverters>main.py
usage: main.py [-h] [--output OUTPUT] [--grid_name GRID_NAME] action input
main.py: error: the following arguments are required: action, input
```

FIGURE 6: DATA TRANSLATION HELP MESSAGE

The valid options while executing the command are the following:

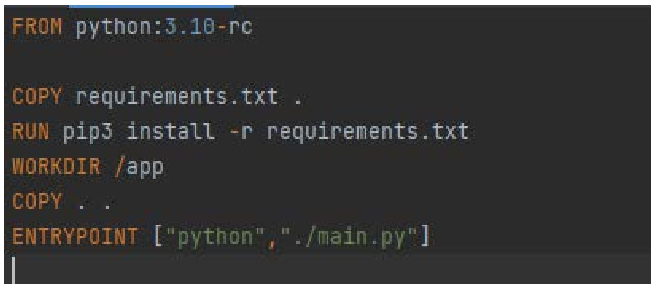
- **Actions**: this includes the format options possible, that are: 'ores', 'cuerva' and 'json'. Depending on the options selected, the executable will translate the input from ORES format to json, from CUERVA format to json or from json format to csv.
- **Input**: this indicates the input target file that will be processed.

- **Grid_name**: this is an optional parameter used when translating CUERVA files. Since input files do not include a grid name, we can manually input which is the Grid corresponding to the demand data.
- **Output**: this is also an optional parameter for the name of the output file name. If no input is given, the result of the program will be output to the *data.out* file.

This code was developed by CIRCE and made available for all partners on the project SharePoint.

### 4.1.1 DOCKER INTEGRATION

The Docker container is based on a solution developed for python on which we add our own code. The *main.py* is the entry point which orchestrates all the different functionalities of the container. In Figure 7 we can see the Dockerfile for the creation of this container.



```
FROM python:3.10-rc

COPY requirements.txt .
RUN pip3 install -r requirements.txt
WORKDIR /app
COPY . .
ENTRYPOINT ["python","./main.py"]
```

FIGURE 7: DATA CONVERTER DOCKERFILE

The Docker container can be built using the following command:

*docker build -t interpreter_converter:interpreter_converter .*

Then we can run the container by using:

*docker run --name converter -v /home/aalcrudo/datos/:/mydata interpreter_converter:interpreter_converter ores "/mydata/ores/PQ_Delivery 2021_10_17.csv" /mydata/output.json*

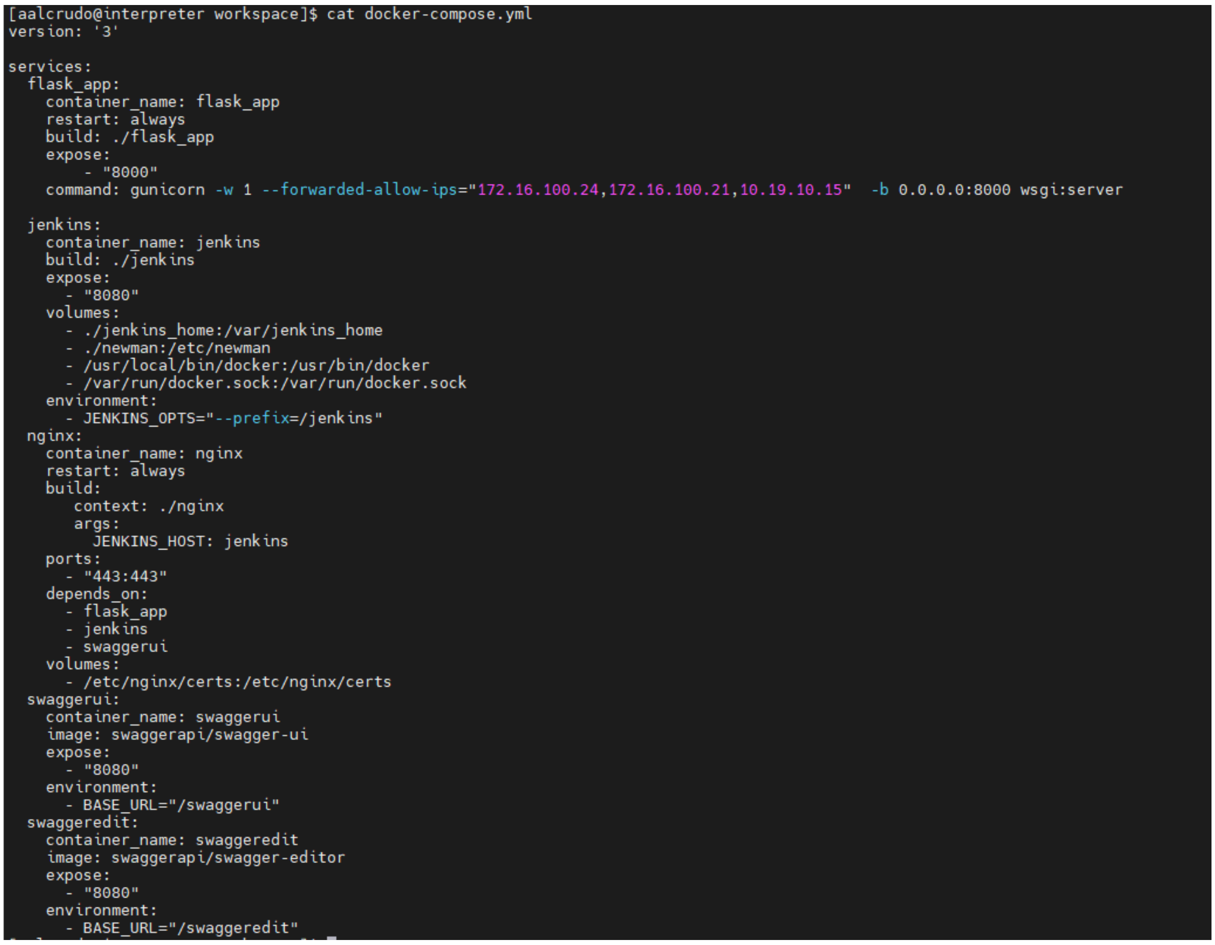
Notice that a volume must be bounded to be able to share a folder between the container and the host machine, which is necessary to deliver input files and access output files.

## 4.2 SERVICES DEVELOPED FOR INTERNAL CLOUD OPERATION

Some of the services developed by CIRCE are offered as REST API services. All those services have been dockerized applying the provided tutorials. Other services like Jenkins and Swagger have also been dockerized in a bundle to demonstrate how to coordinate different services.

In total, 5 services have been bundled together in the CIRCE server using Docker. A flask API with Python code, a Nginx web server, a Jenkins server, and two Swagger components. The *base_url* environment variable has been used to have different services on different parts of the server.

Figure 8 presents all the different services that have been deployed on the CIRCE cloud, from REST API algorithms to Jenkins and Swagger servers. Use of Swagger was considered as part of the continuous integration efforts but was finally not used.



```
[aalcrudo@interpreter workspace]$ cat docker-compose.yml
version: '3'

services:
  flask_app:
    container_name: flask_app
    restart: always
    build: ./flask_app
    expose:
        - "8000"
    command: gunicorn -w 1 --forwarded-allow-ips="172.16.100.24,172.16.100.21,10.19.10.15"  -b 0.0.0.0:8000 wsgi:server

  jenkins:
    container_name: jenkins
    build: ./jenkins
    expose:
      - "8080"
    volumes:
      - ./jenkins_home:/var/jenkins_home
      - ./newman:/etc/newman
      - /usr/local/bin/docker:/usr/bin/docker
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      - JENKINS_OPTS="--prefix=/jenkins"
  nginx:
    container_name: nginx
    restart: always
    build:
       context: ./nginx
       args:
         JENKINS_HOST: jenkins
    ports:
      - "443:443"
    depends_on:
      - flask_app
      - jenkins
      - swaggerui
    volumes:
      - /etc/nginx/certs:/etc/nginx/certs
  swaggerui:
    container_name: swaggerui
    image: swaggerapi/swagger-ui
    expose:
      - "8080"
    environment:
      - BASE_URL="/swaggerui"
  swaggeredit:
    container_name: swaggeredit
    image: swaggerapi/swagger-editor
    expose:
      - "8080"
    environment:
      - BASE_URL="/swaggeredit"
```

FIGURE 8: CIRCE SERVICES DOCKER-COMPOSE

## 5 CONCLUSIONS

Containers are an easy way to solve the typical “works on my machine problem” by turning the environment required by the software in one of the components delivered and packaging everything together.

With Docker and Docker-compose we can also package several tools like databases and web servers together as a single entity that can be deployed in any server without the need for further configuration.

The example code and videos provided are a good starting point to produce software as reusable components that expose services as REST APIs.

Methodological changes, such as continuous integration, require members of the development teams to be fully on board to be successful which is particularly hard in environments with different partners each with its own work methodology and culture. Furthermore, to be successful they require being adopted at the start of the project. In this case a compromise was made to use some of the underlying ideas without disrupting the work process of each partner.

# 6 REFERENCES

[1] ‘Docker’. https://www.docker.com/ (accessed May 27, 2021).
[2] ‘Jenkins’. https://www.jenkins.io/doc/ (accessed January 11, 2022).
[3] ‘Docker Container definition. https://www.docker.com/resources/what-container (accessed December 09, 2021).
[4] ‘Postman | The Collaboration Platform for API Development’. https://www.postman.com/ (accessed Jun. 03, 2021).