

Competition Submission Template - Python

This guide will walk you through the process of implementing your model python project starting from the provided python project template.

You can develop your algorithm either on Windows, Linux or macOS Operating System but it is important to keep in mind that the artifact you need to submit for the competition must be a Docker image. We will see below how to package everything for the submission.

Steps:

[0. Context](#)

[1. Technical Prerequisites](#)

[2. Create your own Python project](#)

[Create an empty virtual environment](#)

[Activate your virtual environment](#)

[Install dependencies from requirements.txt file](#)

[Full example](#)

[3. Project configuration & structure](#)

[Project Configuration](#)

○ [Open the Project Settings](#)

○ [Select or Add a Python Interpreter](#)

○ [Apply and Close](#)

[Project Structure](#)

[4. Develop your model](#)

[Available functions](#)

[5. Package the service template plus your model files in a Docker image](#)

[Freeze python dependencies](#)

[Docker build](#)

[6. Submit your code](#)

[Save docker image as TAR file](#)

[Go to our File Store Service Website and upload your archive](#)

0. Context

As the TUPLES consortium, we provide a template to allow the competitor to focus almost exclusively on the development of the model, as the integration with the evaluation platform will be done directly from the web service template.

Depending on the type of competition chosen (deterministic, probabilistic, explainability), there will be a different evaluation flow, and your model will be embedded within a different process.

The template is pre-configured with all the functions of each challenge, it is up to you to implement only those useful for your type of competition.

For the **deterministic** challenge, you will have the following methods available: *setup, plan*.

For the **probabilistic** challenge, you will have the following methods available: *setup, setup_problem, start_simulation, next_action*.

For the **explainability** challenge, you will have the following methods available: *plan, explain*.

Deterministic and **probabilistic** challenges involve the use of an “evaluator system” that interacts with the model, in order to solve and check the provided solutions.

This is why you will also find an instance of the scalability evaluator among the available material (e.g., competition-setup): it will help you test locally before making your submission.

For the **explainability** challenge, you can implement your explainability model using one of two approaches, as described in the official documentation: "explainer only" or "planner and explainer."

Based on your implementation approach, you must use the proper python functions.

Please ensure that you select the correct option based on the approach implemented in the model at the time of submission on the platform.

An evaluator will be used only to assess that the provided solution matches requirements.

In this challenge no local evaluator for development is needed by definition.

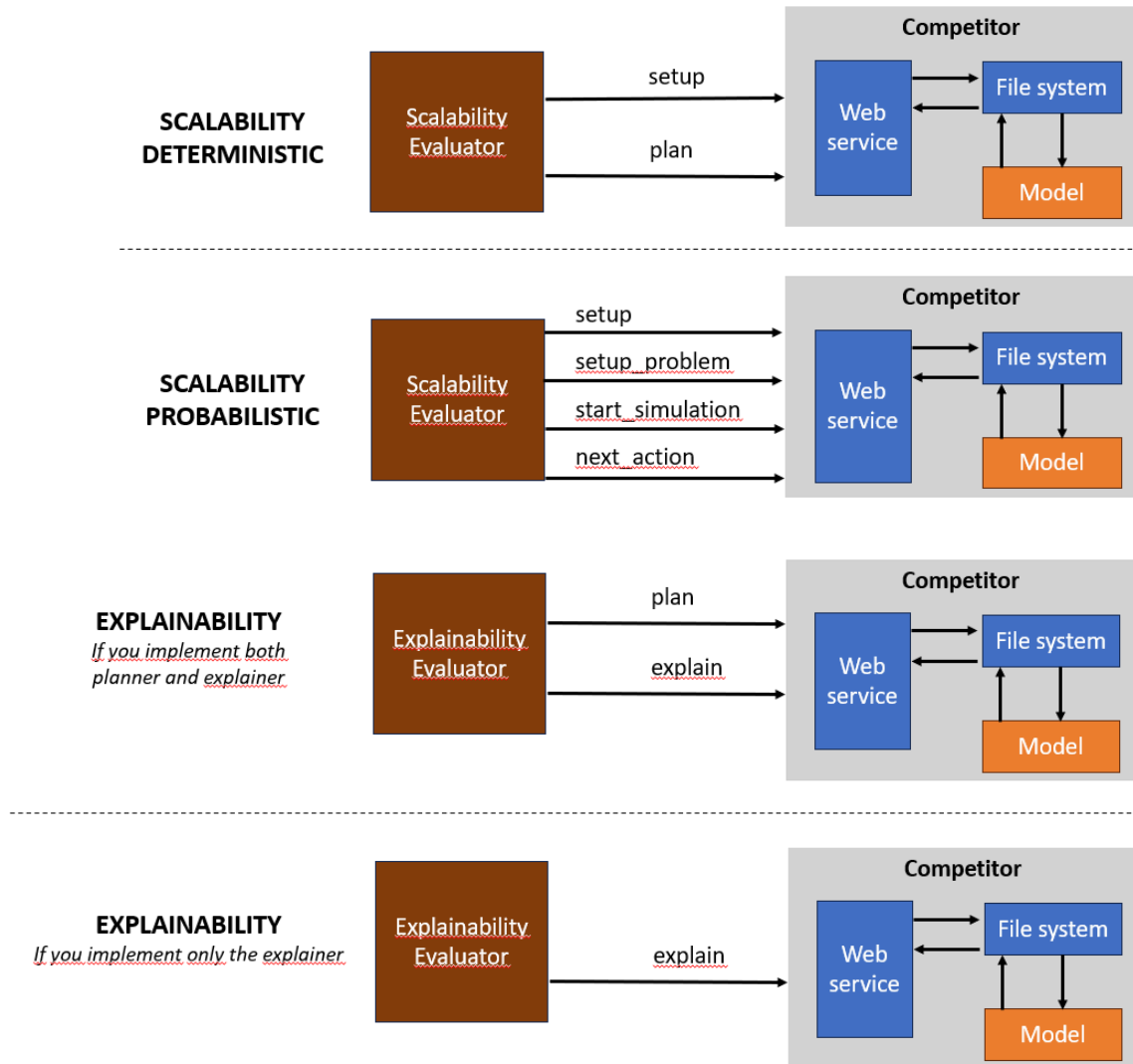
Below is the architectural diagram that describes both evaluators and competitor-model.

- folder “evaluator” is the brown block
- folder “submission” is the grey block.
 - Into the python template you downloaded you already find the webservice and file system handler
 - The ‘Model’ orange block within the architectural schema is the one you will have to implement.

This guide will give you directions on how to encapsulate / wrap your model within the template in just a few steps in order to be compliant with the evaluation system .

You will be able to use the evaluator during local testing as said before. Further details on how to test locally before submission are described in the following chapters.

After an overview of the methods available for each challenge, you can move on to the next chapters with the focus on the implementation of the algorithm and the required methods.



1. Technical Prerequisites

This template leverages on Python 3.11+ and the web service is built using Flask, no knowledge of Flask is required to run and package your model for the competition submission.

Prerequisites:

- Python 3.12 (<https://www.python.org/downloads/release/python-3127/>)
- Postman API Platform (<https://www.postman.com/>)
- Docker (<https://www.docker.com/>) - to build the artifact that will be submitted to the competition

2. Create your own Python project

The goal of this section is to install and run the empty Python template, enabling you to start writing your code with the confidence that all components are functioning correctly.

Please clone the repository into a local directory where you have the right permissions to work, for example:

- Windows OS → D:/workspaces/competition-template-ws-python/
- Linux → /home/username/workspaces/competition-template-ws-python/.

Assuming that you have Python 3.11+ installed on your computer, please open a shell and navigate to the folder where you clone the repository.

Create an empty virtual environment

Subsequently, use the following commands to create an empty virtual environment in the project folder.

```
python -m venv .\env
```

You can find all cmd commands useful to create a virtual environment for Unix/Windows here:

<https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/>

Activate your virtual environment

```
.\env\Scripts\activate
```

If the activation of the virtual environment was successful, you will see "(env)" preceding the path in your shell.

<https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/#activate-a-virtual-environment>

Install dependencies from requirements.txt file

After cloning the project into the designated folder, you should find the "requirements.txt" file in the current directory of your shell. You may now execute the command to activate the newly created virtual environment.

```
python -m pip install -r requirements.txt
```

Use the provided `requirements.txt` file to install all required dependencies.

<https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/#using-a-requirements-file>

Full example

```
C:\competition-template-ws-python>python -m venv .\env
C:\competition-template-ws-python>.\env\Scripts\activate
(env) C:\competition-template-ws-python>python -m pip install -r requirements.txt
Collecting annotated-types==0.7.0 (from -r requirements.txt (line 1))
  Obtaining dependency information for annotated-types==0.7.0 from https://files.pyth...
```

3. Project configuration & structure

Project Configuration

Open your project folder with your favorite IDE (in the examples below we will use PyCharm Community Edition).

Please, note that if you are not using an IDE that adds the “project folder” to the PYTHONPATH (as PyCharm do) then you have to do it manually.

Here are the steps to configure the Python interpreter in a PyCharm CE project:

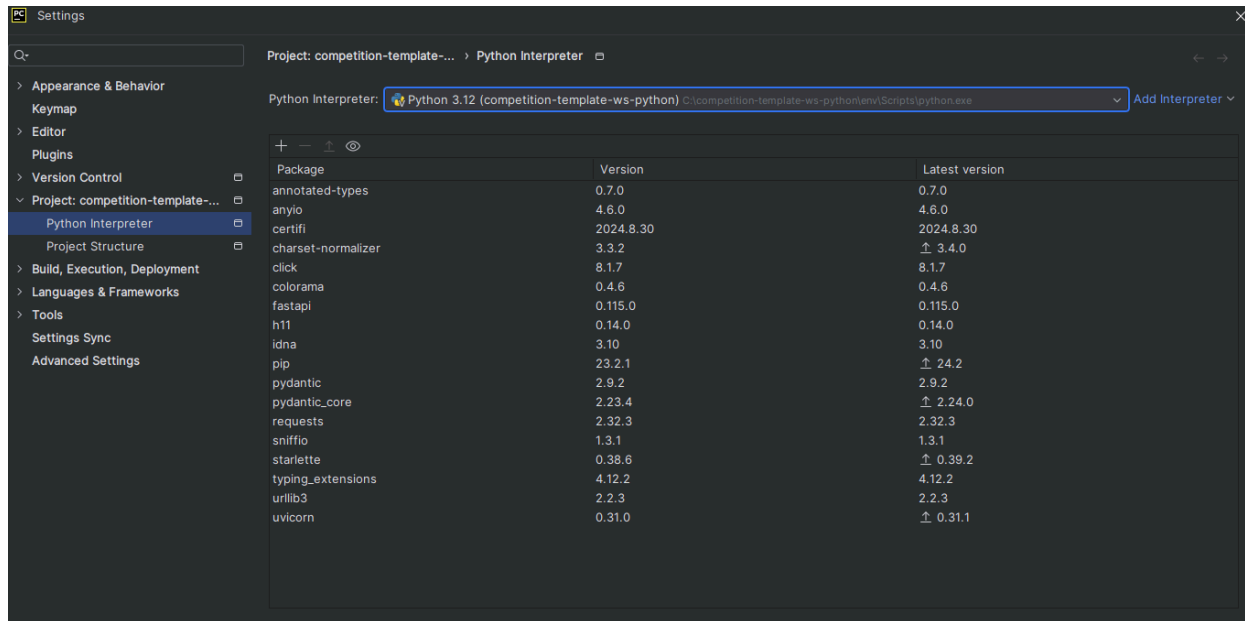
- Open the Project Settings

In the top menu, navigate to **"File" > "Settings"** (on Windows/Linux) or **"PyCharm" > "Preferences"** (on macOS).

In the **Settings/Preferences** window, go to the **"Project: [Your Project Name]"** section on the left side, then select **"Python Interpreter"**.

- Select or Add a Python Interpreter

In the **Python Interpreter** section, **select the interpreter** of your virtual environment created in the previous step (`.\env\Scripts\python.exe`)



○ Apply and Close

After selecting or configuring your Python interpreter:

- Click "OK" or "Apply" to save the settings.

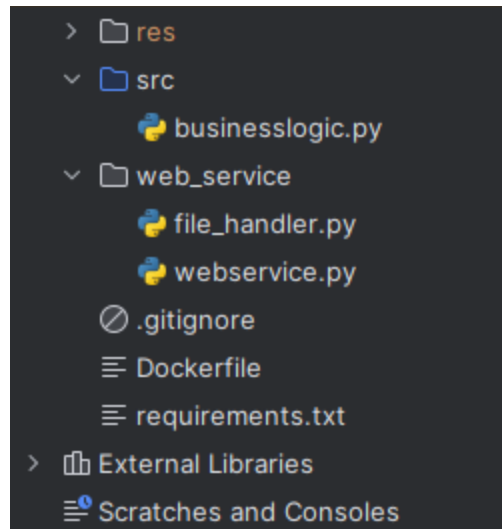
The selected interpreter will now be used for your project, and you should see it displayed in the bottom right corner of the PyCharm window.

Project Structure

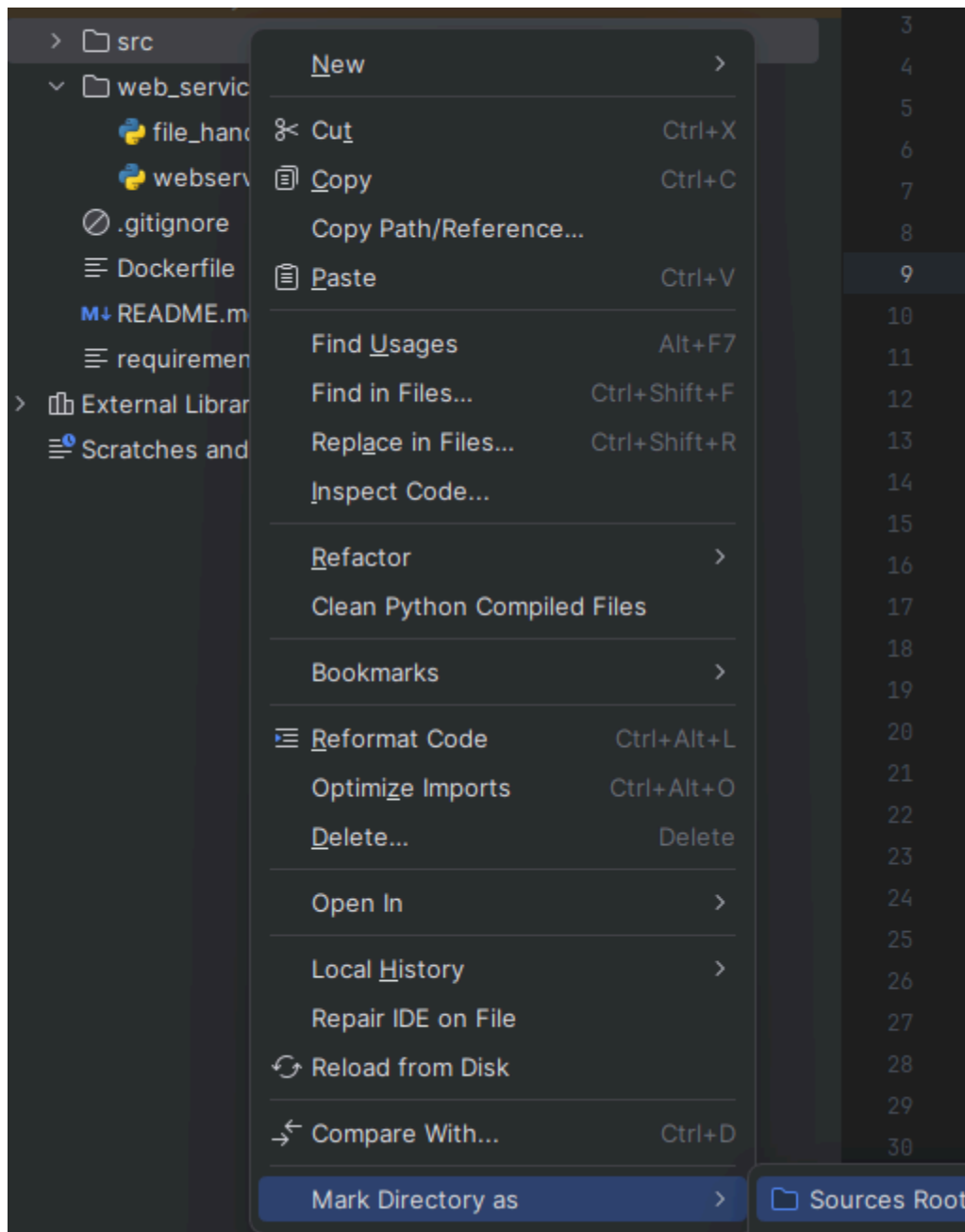
Here is the project structure, not editable.

- **res** folder: This folder is automatically managed by the web service logic.
 - You may add input files here for local testing purposes.
- **src** folder: This is your primary workspace.
 - **business_logic.py**: This is the file you will edit to implement your algorithm's business logic.
 - You may optionally add other Python files in this folder that are useful for your algorithm.
- **web_service** folder: This is a system folder.
 - You are not permitted to modify any files within this folder.
- **Dockerfile**: This file serves as an example of a Dockerfile in use.
 - Instructions for building the Docker image will be provided in the following chapters.

- **requirements.txt:** You will need to update this file if you require additional libraries beyond those already present. Instructions for doing so will be included in the following sections.



Mark the src directory as source root if not already done by PyCharm.



Based on your competition type (deterministic, probabilistic, explainability) all the functions that you must implement are located within the `business_logic.py` file.

According to each function signature, some function will provide you one or two parameters: `input_path` and `output_path`.

- Inside `input_path` you will automatically receive from the master web service all the files required by the competition.

- In the `output_path`, you are required to write all output files specified by the competition. The master web service will automatically read all files in the `output_path` and send them to the competition platform.

You can find all the detailed signatures in the “Develop your model” section.

The paths for `input_path` and `output_path` will be provided to you by the web service with each execution request, be sure to use the `input_path` and `output_path` variable + filename to be sure that you will read/write properly all files as you can see in the code examples.

4. Develop your model

The integration of your model into the template is essential for seamless interaction with the evaluation platform. It is mandatory to implement the class corresponding to the selected challenge (see README.md)

Starting from the “business_logic.py” python file:

- You can add all the files and dependencies you need in the project, but **the main structure of the business_logic.py file must remain the same.**
- **You must not modify the signature of the functions**, otherwise the code will not run correctly in the Tuples Competition Platform.
- **You must not modify all files located in the web_service folder (the exposed port must be 80)**

Available functions

Depending on the type of competition you are participating in, you will have to use certain methods among those available.

Below are the methods provided for each type of competition.

- **Scalability Deterministic**
 - setup:
 - IN: no file
 - OUT: no file
 - plan:
 - IN: problem.json
 - OUT: plan.json
- **Scalability Probabilistic**
 - setup:
 - IN: no file
 - OUT: no file
 - setup_problem

- IN: problem.json
 - OUT: no file
- start_simulation
 - IN: no file
 - OUT: no file
- next_action:
 - IN: state.json, metadata.json
 - OUT: action.json
- **Explainability - Explainer Only:**
 - explain:
 - IN: see explainability challenge documentation
 - OUT: see explainability challenge documentation
- **Explainability - Planner and Explainer:**
 - plan
 - IN: problem.json
 - OUT: plan.json
 - explain
 - IN: see explainability challenge documentation
 - OUT: see explainability challenge documentation

Pay attention to the input and output requirements, the schema above summarizes the expected input and output scheme for each method.

For example, the setup method has no input and no output, while the plan method has both an input and an output file.

For each method that expects one or more input files, you will find an 'input_path' parameter representing the string corresponding to the path to the folder where you will find the file(s) expected by your method.

(e.g. in the plan method, inside input_path you will find a file called problem.json, and so on.)

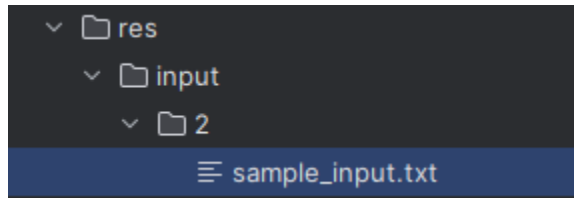
For each method that expects output, you will find a parameter in the method signature called output_path. This parameter represents the path where you will have to create the file for the method (e.g. for the plan method, you will have to create a file called plan.json inside the output_path)

- It will be the web service that takes care of making the JSON files available inside input_path.
- It is always the web service that will automatically retrieve all the JSON files from the output_path and send them back to the platform for evaluation.

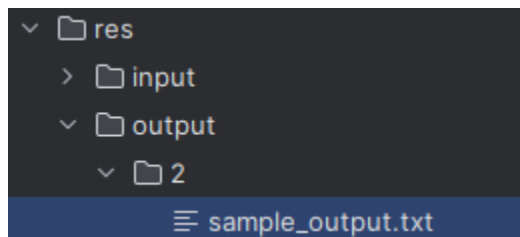
Here is an example, assuming that the chosen competition requires input files named problem.json and the expected output is a json file named plan.json.

- If the chosen competition expects a problem.json file as input → you will find the problem.json file inside the folder defined as 'input_path' + /problem.json, so that you can read it within the proper function.

This is because the master web service, when receiving an execution request, will automatically insert all the files provided by the competition directly within the folder defined in the 'input_path' variable.



- If the output template is to produce a file called plan.json → the plan.json file must be written into the path defined as 'output_path' + /plan.json



You can keep the same principle in mind if you need to manually insert some files to test the model locally.

5. Package the service template plus your model files in a Docker image

Upon completion of your development work, or whenever it is deemed necessary to test the code as a Docker image, please adhere to the following steps:

Freeze python dependencies

Generate a new requirements file if you added packages to your local environment created in chapter n.1,

Use this command to overwrite the `requirements.txt` file adding all the libraries and packages necessary for the execution of the web service and your Python script.

We recommend creating a backup of the "original" file before executing this command before using the `pip freeze` command.

```
python -m pip freeze > requirements.txt
```

Here you can find further details about this command:

<https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/#freezing-dependencies>

Docker build

Before any docker build operation, please be sure that:

- The copied `requirements.txt` file contains all the necessary libraries and the correct versions.
- Any additional files you have added to the project are copied and/or installed correctly.

By following the previous instructions, both of these steps are automatically satisfied: "place any additional files within the `businesslogic` folder" and "freeze requirements."

In case you need to edit the Dockerfile in order to add useful commands to install your algorithm, you can do this.

You must not edit the last 3 commands in Dockerfile.

(In case you change the port for local testing purposes, remember to set it to 80 before submitting.)

```
# Set the working directory to the src folder
WORKDIR /app/web_service

# Expose the port on which the app will run
EXPOSE 80

# Command to run the app
CMD ["python", "-u", "webservice.py"]
```

Go to the root of the Python project and execute the docker build command

```
docker build -t <image-tag>:<image-version> -f <path-to-docker-file> <context-path>
```

Example considering current path as context, and assuming that Dockerfile is saved in current path.

```
docker build -t competition-platform-ws-python:1 .
```

6. Submit your code

This section contains technical instructions for packing the docker image and submitting it.

Please look at the general rules to make sure you have read all the submission procedures and general submission rules.

We expect that each competitor upload a docker image into our Jirafeau website.

In order to load a docker image to Jirafeau, it is required that each competitor use the command "docker save" to create a tar file of the docker image.

Summarizing, the technical action required to package and submit your docker image is:

- Create tar file using docker save command
- Go to our file sharing platform and upload the file

Save docker image as TAR file

You can package your docker image using this single command that creates the **.tar** file.

<https://docs.docker.com/reference/cli/docker/image/save/>

Syntax

```
docker save --output <dest-path> <image-tag>:<image-version>
```

Example

```
docker save --output ./myimage_latest.tar myimage:latest
```

Below is an example of the expected tree structure.

```
myimage_latest.tar
├── blobs
├── index.json
├── manifest.json
├── oci-layout
└── repositories
```


Go to our File Store Service Website and upload your archive


The “Direct download link” that Jirafeau will return should be placed within the platform, indicating where the evaluation system can retrieve the zip containing your docker image.

Please ensure that your zip file is not larger than 15Gb.

- Go to the website of our file store service (<https://www.tuplesfile.store/>).
- Upload your TAR archive
- After uploading the file, copy and paste the DIRECT DOWNLOAD LINK to competition submission website, fill all required submission fields and start your new submission request.

File uploaded.

Download page 

[http://tuples\[redacted\]](http://tuples[redacted]) 

Direct download link

[http://tuples\[redacted\]&d=1](http://tuples[redacted]&d=1) 

Delete link

[http://tuples\[redacted\]](http://tuples[redacted]) 