AI4EU Deliverable D3.3

Platform Management, AI-Resource Publication and Enrichment Manual

Deliverable abstract

The goal of the AI4EU On Demand Platform is to make AI and machine learning accessible to a wide audience by creating an extensible marketplace of reusable solutions, sourced from a variety of AI toolkits and languages that ordinary developers, who are not machine-learning experts or data scientists, can easily use to create their own applications.

This document describes the publication and review process that tool providers must follow to onboard and publish their AI resources. Moreover, the system architecture and platform management of AI4EU Experiments is explained.

Contents

1. Introduction

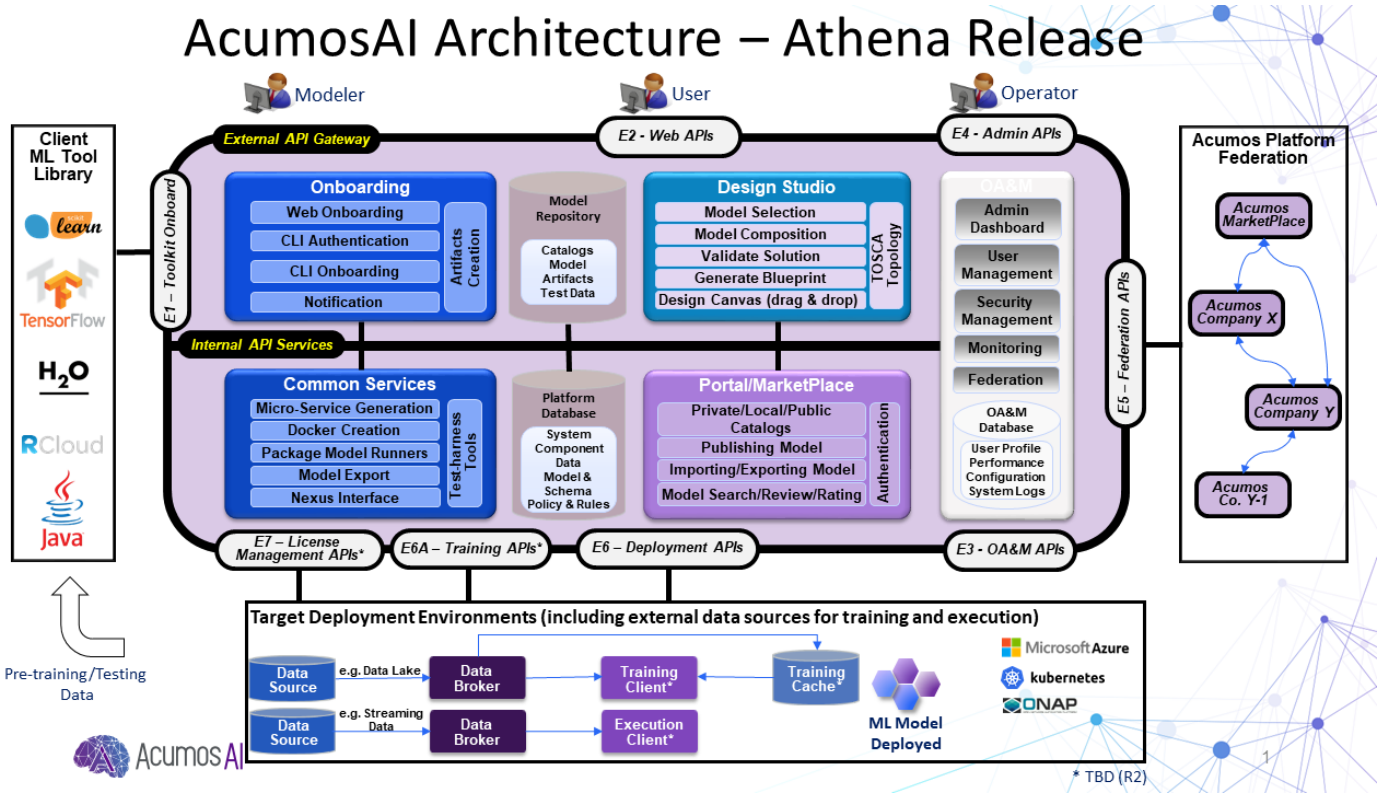
The goal of the AI4EU On Demand Platform is to make AI and machine learning accessible to a wide audience by creating an extensible marketplace of reusable solutions, sourced from a variety of AI toolkits and languages that ordinary developers, who are not machine-learning experts or data scientists, can easily use to create their own applications.

The three main subsystems of the platform are as follows:

1. Content Management System (CMS)
2. AI4EU Experiments
3. Search

For each of them there will be a dedicated deliverable. This one is for AI4EU Experiments. All of them will be accessible publicly in the CMS.

2. AI4EU Experiments Platform Management



source: https://docs.acumos.org/en/boreas/architecture/platform_architecture.html

2.1. System Overview

AI4EU Experiments is based on the Acumos AI project (https://www.acumos.org/). From this, it inherits a clean microservice architecture with a dozen microservices using RESTful communication to interact. The data is stored in two databases: MariaDB for relational data and Nexus for files.

To support easy scalability, the microservices run as pods in a Kubernetes Cluster using the NGINX-Ingress-Controller to make the services externally accessible. The most important Microservices are:

- **Portal-Marketplace:** Web-UI providing the basic HTML based pages and dialogs of the catalog, including the publication process.

- **Design-Studio:** A JavaScript based application to visually compose AI-Pipelines from models in the catalog.
- **Onboarding:** Implements the serverside functions to onboard models. It offers a public API to onboard models via scripts / CLI.
- **Common-Data-Service:** The common dataservice is the interface to the persistent catalog object model.
- **Kubernetes-Client:** The Kubernetes-Client creates kubernetes deployments for models and pipelines.
- **Federation:** Acumos instances can be loosely coupled by Federation to exchange models.

Additionally, there is the **orchestrator** that is controlling the execution of a pipeline after deployment into the customer's IT environment.

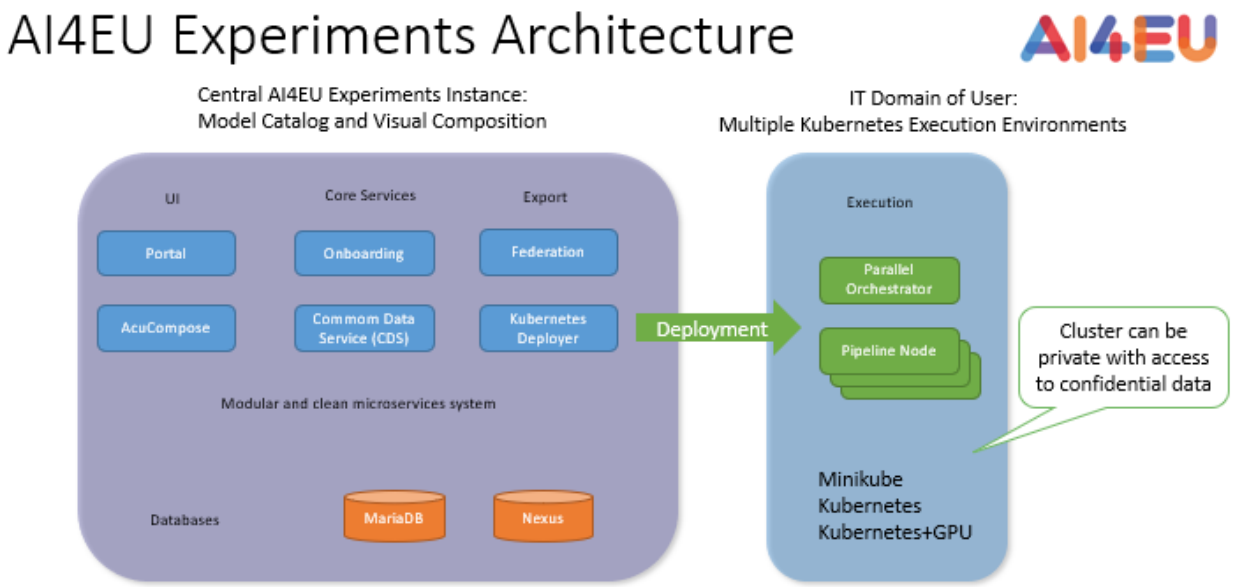
2.2. Acumos Roots and Important Differences

The project AI4EU included the goal to use Acumos AI (https://www.acumos.org/) as a place to experiment with AI tools from a catalog. However, in the course of the project, we forked from Acumos a new branch now called AI4EU Experiments (https://github.com/ai4eu) which deviates significantly from Acumos in several areas:

- Unlike Acumos, AI4EU Experiments does NOT build docker images of models internally. It turned out that building the docker images internally results into many restrictions and limitations for the image providers and also poses a security risk, because the image construction script ("Dockerfile") is executed inside the same context as AI4EU Experiments itself.
- As a consequence, the Acumos bundle format is not supported, which turned out to be not interoperable. Instead, all resources must provide gRPC/Protobuf interfaces and follow the AI4EU Experiments Container Specification: https://github.com/ai4eu/tutorials/tree/master/Container_Specification
- AI4EU Experiments only stores references to docker images (docker image URIs), never the images directly. This not only saves storage space, but more important, it avoids legal implications for commercial tools and confidential data.
- Similar approach for datasets: they are not uploaded to the platform. They are only made accessible by so called Databrokers that make the data available as one of the input nodes for a pipeline.
- AI4EU Experiments requires the models to support gRPC communication according to the container specification (see section 3.2).
- Points 1-4 together ensure maximum flexibility for the model providers and also maximum interoperability between the models (both of which were not possible with the original Acumos approach).
- The AI4EU Experiments design studio supports more topologies and gRPC streaming.
- We have removed Acumos features that have never been finished like the Workbench and NIFI integration.
- Unlike Acumos, AI4EU Experiments does not allow models to be executed inside AI4EU Experiments itself, for security and scalability reasons.

2.3. Architecture

Looking at the whole picture, it must be distinguished between the central instance of AI4EU Experiments and the separate execution environments, which are typically in the IT domain of the user.



The AI4EU Experiments instance offers the catalog, collaboration and the visual pipeline composition editor. The connection to an execution is made by the Kubernetes Deployer, which creates from the pipeline definition (topology) the corresponding Kubernetes configuration files. Please note that this means there are many execution environments, typically in an IT domain controlled by the user. This has the important advantage that confidential data is accessible for the pipeline and also that commercial models can be securely executed.

Nevertheless, it is possible to have public execution environments like the KI.NRW Playground that enables the user to try out models quickly or work collaboratively on solutions and proof of concepts.

Thanks to the modular architecture of AI4EU Experiments, it is easy to support different types of execution environments, like HPC or docker-compose, just by adding the corresponding deployment service to the platform.

To support many different open source and commercial scenarios, the docker images of the models are not stored inside AI4EU Experiments, but in public or private docker registries. Only the references to the images are stored (the so called docker image tag). Similar for datasets: not the dataset is uploaded, but a databroker component that can make the connection between the pipeline and the data. This is essential when it comes to support very large datasets or confidential data: in both cases the pipeline needs to be deployed close to the data.

2.4. Installation and Resource Planning

The audience for the following two sections is system administrators that are familiar with the Linux command line, networking concepts and Kubernetes.

The installation scripts are managed in the following public repository, which has always the latest version and contains also additional information.

https://github.com/ai4eu/system-integration

The installation and operating are based on Kubernetes which is an execution environment focused on high availability and scalability. It is easy to add new resources to a running system and configure it for minimum downtime. For further information please consult: https://kubernetes.io/

A Kubernetes-Cluster consists of several nodes. For the worker nodes used in production, we recommend the following specs:

- 10 CPU Cores
- 64 GB RAM
- 2 TB local storage

Software versions:

- Kubernetes 1.19.16
- Kube CNI 0.8.7
- Kubernetes dashboard 2.3.1 (optional for operation)
- Calico 3.16
- Helm 3.6.3

Preconditions:

- Ubuntu 20.04 server, one big disk partition
- Docker as part of the Ubuntu installation and docker service enabled and started
- At least 20GB of disk space available for a minimal installation (/var/lib alone will require more than 10GB)
- Installation user created that belongs to groups docker and sudo, in this example the user is ai4eu
- /etc/hosts has exactly one entry for the FQHN pointing to the external ipv4 interface and make sure that the hostname resolvable using DNS
- for https optionally have letsencrypt certificates already installed (currently only letsencrypt is supported)

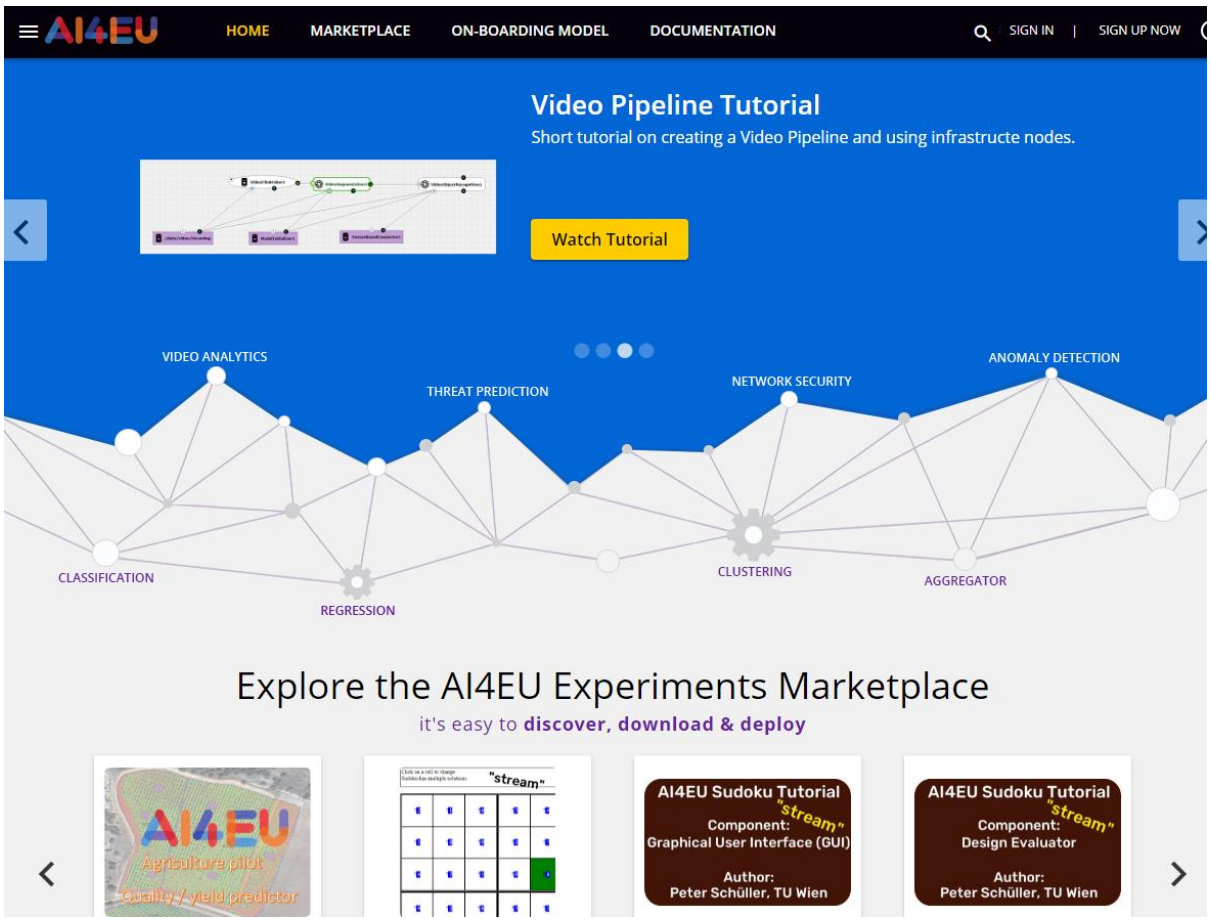
Become user ai4eu (installation user)

Clone this repo https://github.com/ai4eu/system-integration into $HOME and then:

```
# Use the installation user without sudo (you will be asked for sudo if neccesary).
cd system-integration/tools
bash setup_k8s.sh
# dashboard access is optional[1]
bash setup_helm.sh
cd $HOME
# replace FQHN appropriately
bash system-integration/AIO/setup_prereqs.sh k8s FQHN $USER generic | tee log_prereqs.txt
cd system-integration/AIO/
bash oneclick_deploy.sh | tee log_oneclick.txt
```

Some of those scripts might take several minutes to complete execution. The last script should end with output showing the URLs to use, e.g. among others:

Portal: https://(your FQHN):443



[1] https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

2.5. Periodic Maintenance Tasks

Backups

As part of the system-integration repository comes the make_backup.sh script. It makes rotating backups of the two databases for the last 7 days. The MariaDB is the storage for structured, tabular data whereas the Nexus repository is the storage for files. In Acumos, it used to be the place where big docker images were stored, but for several reasons AI4EU Experiments only stores references to images which take no space in Nexus at all.

```
#!/bin/bash

set -x
trap 'fail' ERR

export AIO_ROOT=$HOME/system-integration/AIO
source $AIO_ROOT/utils.sh
source $AIO_ROOT/acumos_env.sh

export TODAY=$(date +"%a")
export TARGET_BDIR=$HOME/backups/$TODAY
export BDIR=$HOME/backups/current
mkdir -p $BDIR
cd $BDIR
cp $AIO_ROOT/nexus_env.sh .


mysqldump -h $ACUMOS_MARIADB_DOMAIN -P $ACUMOS_MARIADB_NODEPORT --user=root --
password=$ACUMOS_MARIADB_PASSWORD  $ACUMOS_CDS_DB > acumosdb.sql

export NEXUS_DATA_DIR=/mnt/acumos/$(kubectl -n $ACUMOS_NEXUS_NAMESPACE get pvc |
tail -1 | sed 's/ \+/ /g' | cut -d' ' -f3)
cd $NEXUS_DATA_DIR
tar -czf $BDIR/nexus-data.tgz --exclude=cache --exclude=tmp --exclude=log --
exclude=javaprefs .

cd $HOME
rm -rf $TARGET_BDIR
mv $BDIR $TARGET_BDIR
```

Restore

To restore a backup, both of the aforementioned databases need to be initialized.

First, the SQL-database can be read with the mysql CLI-Tool from the file acumosdb.sql.

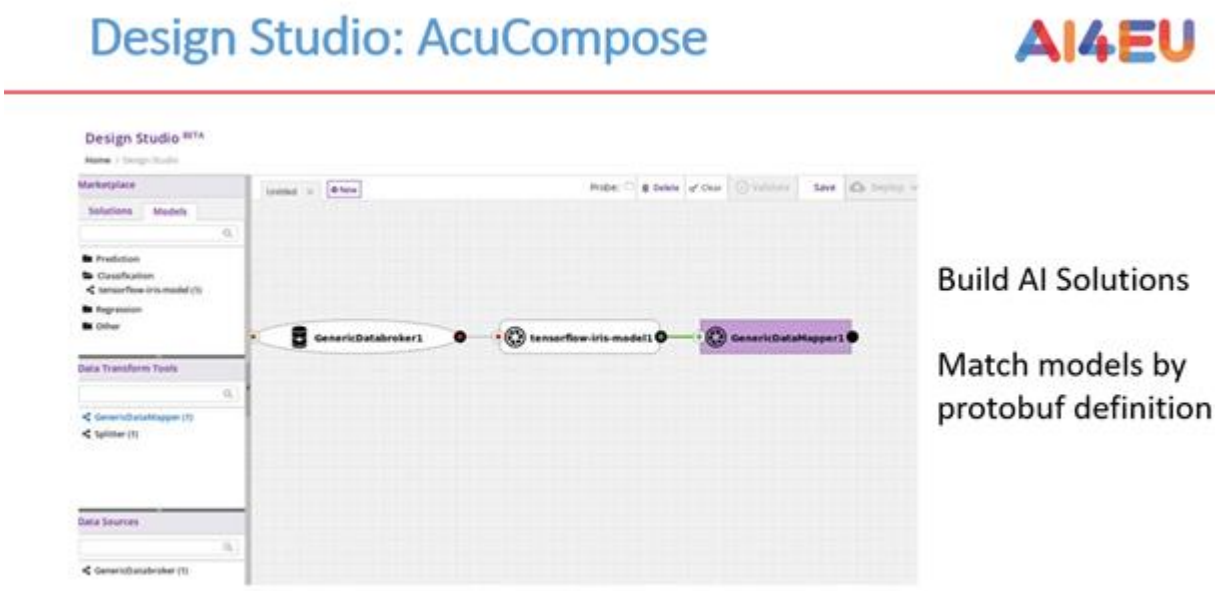
For the Nexus data, we need to find the location of the corresponding persistent volume on the filesystem with the following command:

```
export NEXUS_DATA_DIR=/mnt/acumos/$(kubectl -n $ACUMOS_NEXUS_NAMESPACE get pvc |
tail -1 | sed 's/ \+/ /g' | cut -d' ' -f3)
```

Then change directory to $NEXUS_DATA_DIR and extract the file nexus-data.tgz

3. AI-Resource Publication and Enrichment

AI4EU Experiments is a design and distribution framework for integrating solutions from modular components. It provides a launchpad for training[2] and validating both individual components and composite solutions, and then securely distributing the results to targeted communities through an electronic catalog, from which components can be selected. AI4EU Experiments also provides the deployment interfaces that allow solutions to be trained or executed in several runtime environments, mostly kubernetes-based.



AI4EU Experiments includes a graphical tool, called AcuCompose, for chaining together multiple models, data translation tools, filters and output adapters into a full end-to-end solution that can be deployed into the aforementioned runtime environments. AI4EU Experiments only requires a container management facility, like Kubernetes, to deploy and execute portable general-purpose applications. At the very core of interoperability is the AI4EU container specification:

[2] https://github.com/ai4eu/tutorials/tree/master/news_training

AI4EU Container Specification

- Docker container
- Protobuf specification of public interface
- gRPC communication
- optional Web-UI for human interaction
- Based on free / open source technologies
- Recommendations for scalability, training and GPU-Support

```
// set used version of protobuf
syntax = "proto3";


// define input data structure
message IrisDataFrame {
  repeated double sepal_length = 1;
  repeated double sepal_width = 2;
  repeated double petal_length = 3;
  repeated double petal_width = 4;
}

// define output data structure
message ClassifyOut {
  repeated int64 value = 1;
}

// define exposed service
service Model {
  rpc classify (IrisDataFrame) returns (ClassifyOut);
}
```

AI4EU Experiments also has the means for collaboration in closed groups on dedicated projects in mixed teams (i.e., building a pipeline together, or working through an auditing process by building pipelines and store audit documentation and execution results).

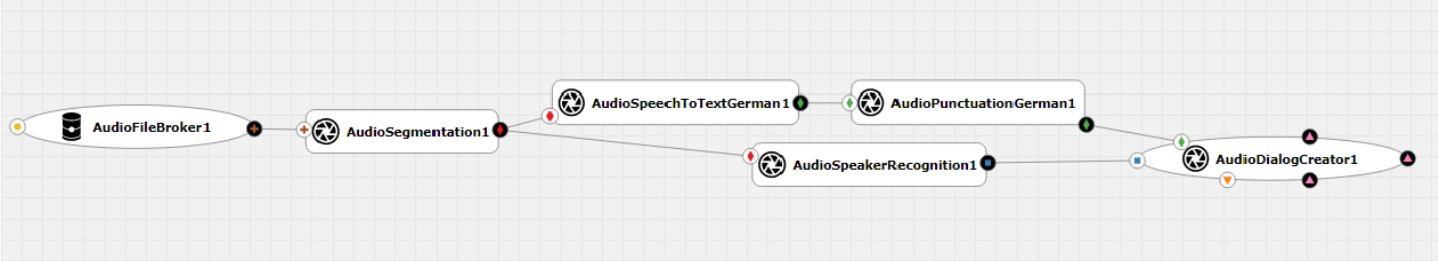
AI4EU Experiments supports many hardware infrastructures in order to maximize the utility of the solutions being deployed. This makes AI4EU Experiments-compatible solutions portable and flexible. AI4EU Experiments offers a mechanism for packaging, sharing, licensing, and deploying AI models in the form of portable, containerized microservices, which are interoperable with one another. It provides a publication mechanism for creating shared, secure catalogs and a mechanism for deployment onto any suitable runtime infrastructure.

3.1. The Concept of Re-Usable AI Building Blocks

To increase the usage of AI in industries but also facilitate the exchange of models in the AI community, the concept of re-usable AI building blocks plays a key role. The inner workings of a model should be encapsulated and only a small subset of methods, called the public interface, should be exposed in a standard way.

These public interfaces could be common for similar tasks, like speech to text conversion for audio data or object detection in video data. If we have a well-defined interface for speech to text conversion, it can be implemented for many different models: for different languages (French or English) but as well using different technologies internally (e.g. convolutional neural networks or Transformer based neural networks). The user can choose the model the best fit the needs of the use case at hand.

Let's have a look at an audio pipeline:



We can see that the pipeline uses segmentation, then speech to text German and speaker recognition and adding punctuation. Each of the models can be replaced by an alternative, be it for better precision or performance or different languages. And using the AI4EU Experiments visual composition editor, even a domain user for audio processing is able to edit pipelines without the necessity to be an AI expert. Many of the aspects of this concept will be described in more detail in subsequent sections.

3.2. AI4EU Experiments Container Specification

The audience for this chapter is software developers that are familiar with Docker[3] containers and command line tools.

In recent years, the IT landscape has seen the rise of Docker Containers and Kubernetes and both of those technologies have changed fundamentally the way software components are packaged, installed and scaled.

AI4EU Experiments builds on the features and benefits of both technologies to define the format for re-usable AI building blocks.

This chapter specifies the docker container format for tools and models that can be onboarded on the AI4EU Experiments platform so they can be used in the visual composition editor as re-usable, highly interoperable building blocks for AI pipelines.

In short, the container should define its public service methods using protobuf v3 and expose these methods via gRPC. All these technologies are open source and freely available. Here are the respective home pages for documentation and reference:

- https://docs.docker.com/reference/
- https://developers.google.com/protocol-buffers/docs/overview
- https://developers.google.com/protocol-buffers/docs/proto3
- https://www.grpc.io/docs/

Because the goal is to have re-usable building blocks to compose pipelines, the main reason to choose the above technology stack is to achieve the highest level of interoperability:

- docker is today the defacto standard for serverside software distribution including all dependencies. It is possible to onboard containers for different architectures (x86_64, GPU, ARM, HPC/Singularity).
- gRPC together with protobuf is a proven specification and implementation for remote procedure calls supporting a broad range of programming languages and it is optimized for performance and high throughput.

Define model.proto

Please note that the tools and models are not limited to deep learning models. Any AI tool from any AI area like reasoning, semantic web, symbolic AI and of course deep learning can be used for pipelines as long as it exposes a set of public methods via gRPC.

The public service methods should be defined in a file called model.proto:

- it should be self contained, thus contains the service definitions with all input and output data structures and no imports can be used
- a container can define serveral rpc methods, but all in one .proto file and in the same service { } block

[3] Docker is a Trademark of Docker Inc.

- it must not have a package declaration, as this precludes automatic message dispatching when used as part of a pipeline

```
//Define the used version of proto

syntax = "proto3";


//Define a message to hold the features input by the client

message Features {

	float MSSubClass = 1 ;

	float LotArea = 2 ;

	float YearBuilt = 3 ;

	float BedroomAbvGr = 4 ;

	float TotRmsAbvGrd = 5 ;

}

//Define a message to hold the predicted price

message Prediction {

	float salePrice = 1 ;

}


//Define the service

service Predict {

	rpc predict_sale_price(Features) returns (Prediction);

}
```

Important: The parser for .proto-files inside AI4EU Experiments is much less flexible than the original protobuf compiler, so here are some rules. If the rules are not followed, it prohibits the model from being usable inside the visual editor AcuCompose. **Moreover, the enum keyword is not yet supported!**

Rule	Good	Bad
syntax spec must be in double quotes	**syntax = "proto3";**	**syntax = 'proto3';**
there must always be a space before an opening curly brace	**service Predictor {**	**service Predictor{**
the closing curly brace must be in a separate line	**message Empty {** **}**	**message Empty {}**
there must not be curly braces after a rpc line	**rpc predict (AggregateData) returns (Prediction);**	**rpc predict (AggregateData) returns (Prediction) {}**

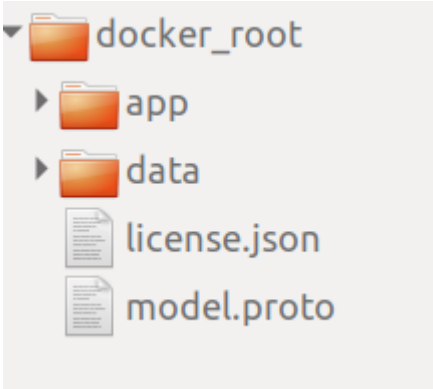
Based on model.proto, you can generate the necessary gRPC stubs and skeletons for the programming language of your choice using the protobuf compiler **protoc** and the respective protoc-plugins. Then create a short main executable that will read and initialize the model or tool and starts the gRPC server. This executable will be the entrypoint for the docker container. **The gPRC server must listen on port 8061.**

If the model also exposes a Web-UI for human interaction, which is optional, it must listen on **port 8062**.

The filetree of the docker container should look like below. In the top-level folder of the container should be the files

- model.proto
- license.json

And also the folders for the microservice like app and data, or any supplementary folders:



The license file is not mandatory and can be generated after onboarding with the License Profile Editor in the AI4EU Experiments Web-UI.

There are several detailed tutorials on how to create a dockerized model in this repository: https://github.com/ai4eu/tutorials

Important recommendation: for security reasons, the application in the container should **not** run as root (which is the default). Instead, an unprivileged user should be created that runs the application, here is an example snippet from a Dockerfile:

```
RUN useradd app
USER app
CMD ["java", "-jar", "/app.jar"]
```

This will also allow the docker container to be converted into a Singularity container for HPC deployment.

Status and Error Codes

The models should use gRPC status codes according to the spec:
https://grpc.github.io/grpc/core/md_doc_statuscodes.html

For example if no more data is available, the model should return status 5 (NOT_FOUND) or 11 (OUT_OF_RANGE).

First node parameters (e.g. Databrokers)

Generally speaking, the orchestrator dispatches the output of the previous node to the following node. A special case is the first node, where obviously no output from the previous node exists. In order to be able to implement a general orchestrator, the first node must define its services with an Empty message type. Typically, this concerns nodes of type Databroker as the usual starting point of a pipeline.

```
syntax = "proto3";

message Empty {
}

message NewsText {
      string text = 1;
}

service NewsDatabroker {
      rpc pullData(Empty)returns(NewsText);
}
```

To indicate the end of data, a Databroker should return status code 5 or 11.

Scalability, GPU support and training

The potential execution environments range from Minikube on a Laptop over small Kubernetes clusters to big Kubernetes clusters and even HPC and optional GPU acceleration. **It is possible to support all those environments with a single container image** taking into account some recommendations:

- let the model be flexible with memory usage: use more memory only if available
- let the model be scalable if more cpu cores are available (allow for concurrency): https://en.wikipedia.org/wiki/Concurrency_(computer_science)
- some AI frameworks like PyTorch or Tensorflow can be used in a way to work with or without GPU with the same code. Here is an example with PyTorch: https://stackoverflow.com/a/56975325
- even training is possible if the model exposes the corresponding methods in the protobuf interface

Shared Folder for Pipeline Nodes

To be compatible with the shared folder concept of AI4EU Experiments, please use an environment variable to pass the absolute path of the shared folder to the processes running inside the container: so it should be the path from the inside the container point of view. Let's assume the shared folder is mapped as "/data/shared" into the container, then the container could be started like

```
docker run --env SHARED_FOLDER_PATH=/data/shared ...
```

How this shared folder is actually set up, depends on your container runtime and is different for docker, docker-compose or Kubernetes. For AI4EU Experiments, the Kubernetes deployment client will take care of it.

Streaming support for gRPC services

The “stream” keyword is supported for both: RPC input and output.

That means a RPC can have the following forms, assuming `Input` and `Output` are Protobuf message types:

- Each RPC call consumes one input message and returns one output message:

```
rpc call(Input) returning (Output);
```

- Each RPC call gets at least one input message, decides by itself when to stop consuming the input stream, after which it returns a single output message:

```
rpc call(stream Input) returning (Output);
```

- Each RPC call gets at one input message and decides by itself how many output messages to produce before closing the stream:

```
rpc call(Input) returning (stream Output);
```

- Each RPC call gets at least one input message, decides by itself how many output messages to produce and when to stop consuming input before closing input and output streams:

```
rpc call(stream Input) returning (stream Output);
```

Special cases of the above are the following, where `Empty` is a message type without any fields.

- RPC is called immediately when orchestration starts and is restarted whenever it closes the output stream:

```
rpc call(Empty) returning (stream Output);
```

  This pattern is useful for GUIs, with one RPC call for each type of User Event that should trigger a computation in other components.

  This pattern is useful for sensors, with one RPC call for each type of sensor reading that should trigger a computation in other components.

- RPC is consuming messages only:

```
rpc call(stream Input) returning (Empty);
```

  This pattern is useful for GUIs, with one RPC call for each type of input to display.

Some notes about how streaming works:

- Connecting “stream” outputs with “non-stream” inputs and vice versa is allowed.
- Orchestrator creates a Queue for each connection between two ports.
- Orchestrator creates a Thread for each RPC call. Essentially orchestration is parallel.
- An RPC is started as soon as there is a message in the input queue, or immediately for the message called `Empty`.

- Cycles among components are possible.
- For example, a cycle from a GUI that returns User Events as streaming output to a computation component with a simple non-streaming RPC which feeds back into the GUI into a RPC that displays the result.

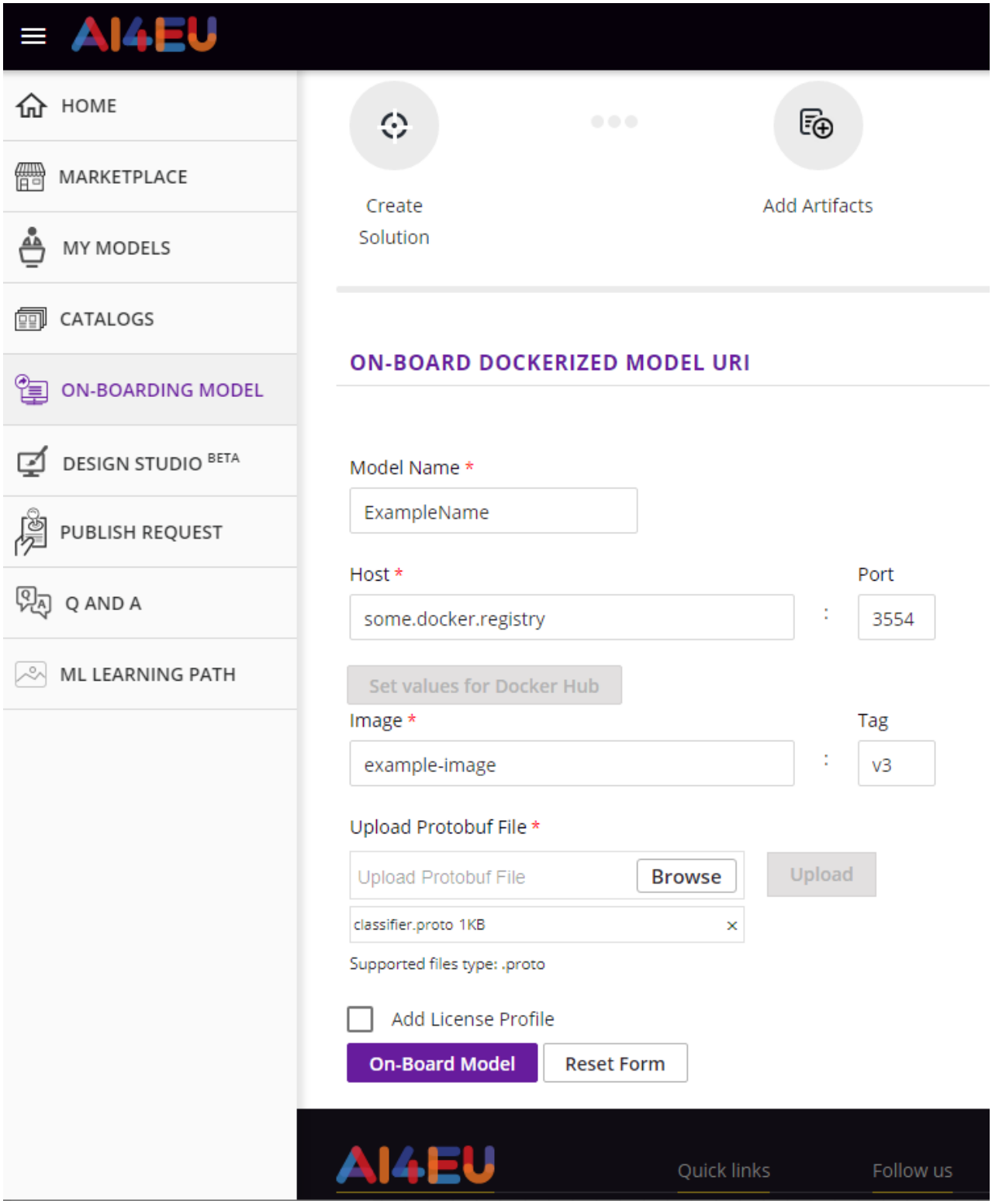
GUI RPC1 (`Empty` input, stream output) → Computation RPC → GUI RPC2 (stream input)

The Sudoku Tutorial streaming branch contains such a GUI (a webinterface), see https://github.com/peschue/ai4eu-sudoku/blob/streaming/gui/sudoku-gui.proto.

3.3. Onboarding

To add a model to the AI4EU Experiments catalog, two steps are needed: the first step is called onboarding and saves the technical properties of the model in the database. An onboarded model is not visible in the catalog, but it can be used in the design studio and shared with specific users via the "share model" functionality. The second step is publication which adds all the properties to the model to make it a complete catalog entry.

The picture below shows the onboarding dialog with the input fields filled with example values:



Each model must have a name. Then there must be a reference to a docker image, that resides in a Docker registry. It can be a private registry, a public registry or Docker Hub. Since Docker Hub has some specialties, there is a button to set the input fields accordingly. Note that the docker image is not stored inside AI4EU Experiments and will only be referenced when it is deployed to an execution environment.

Fowllowing the AI4EU Experiments Container Specification, each model must expose the public interface via gRPC defined in Protobuf. The .proto file must be uploaded into AI4EU Experiments because it is needed to connect only matching model ports in the design studio.

A license is mandatory for publishing and can optionally already added here. During the publication process (see section 3.4) it can also be added.

Datasets

For datasets, the same rules apply as for containers: they are not uploaded to AI4EU Experiments. Instead, only so called Databrokers are onboarded, that can make the connection between the pipeline nodes and the data in the execution environment and transmit it via gRPC.

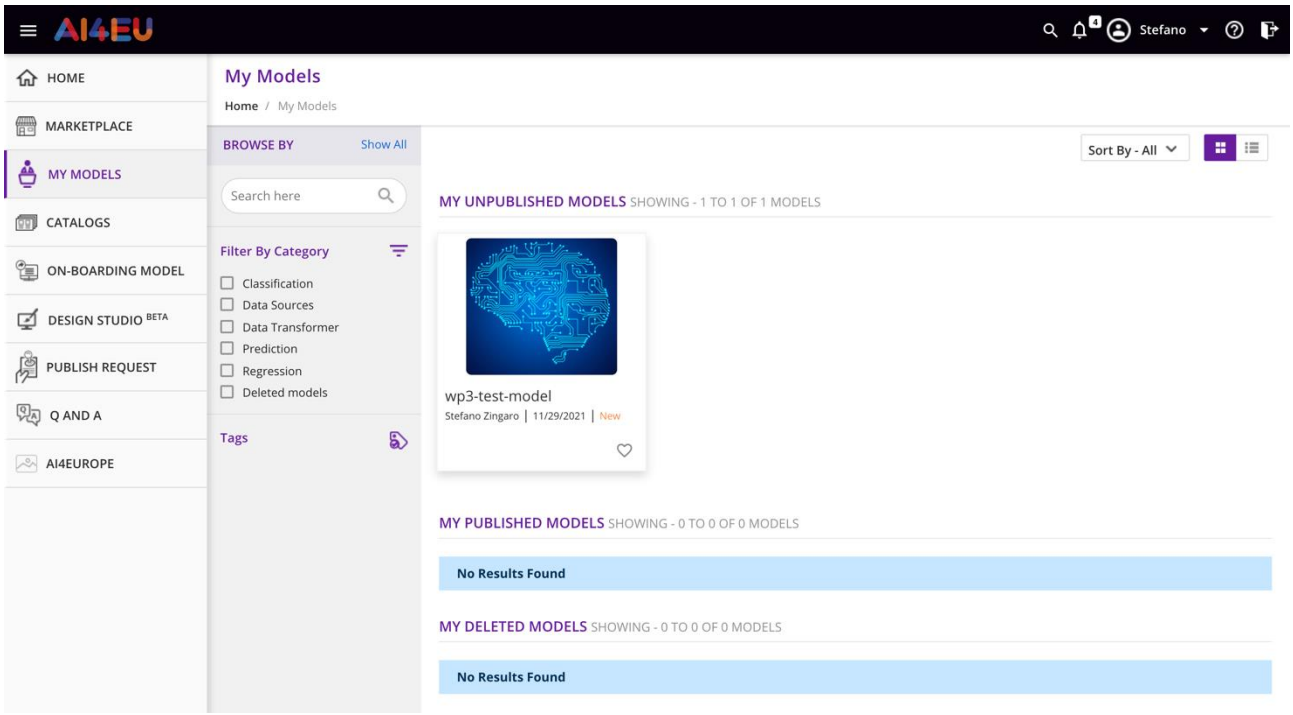
3.4. Publication and Review Process

In this section, we will describe the prerequisites for publishing a resource on the AI4EU platform and provide a step-by-step guide to complete the publication on the AI4EU marketplace [4].

Prerequisites

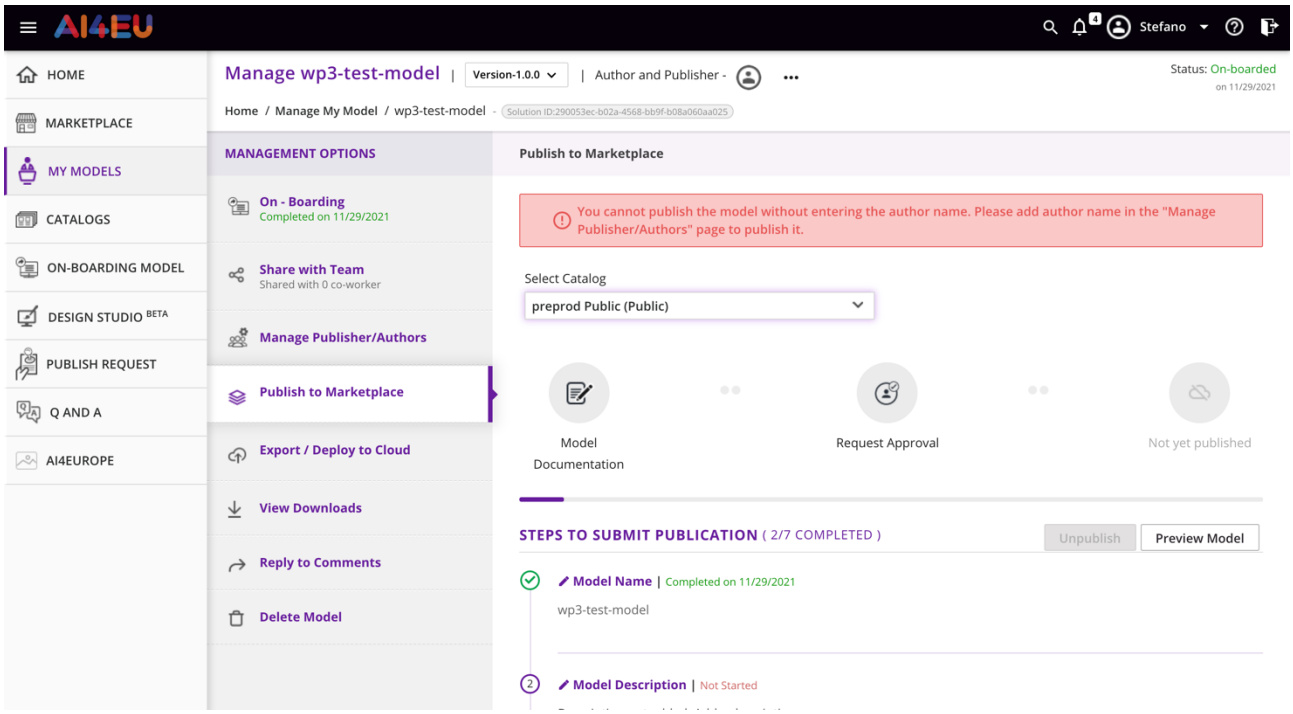
The publication process involves the following prerequisites:

- The model for the resource must have been previously onboarded (see Section 3.3 for a description of the onboarding process). Unpublished models ready for publication can be viewed under the “My unpublished models” section on the "My Models" page (see below).

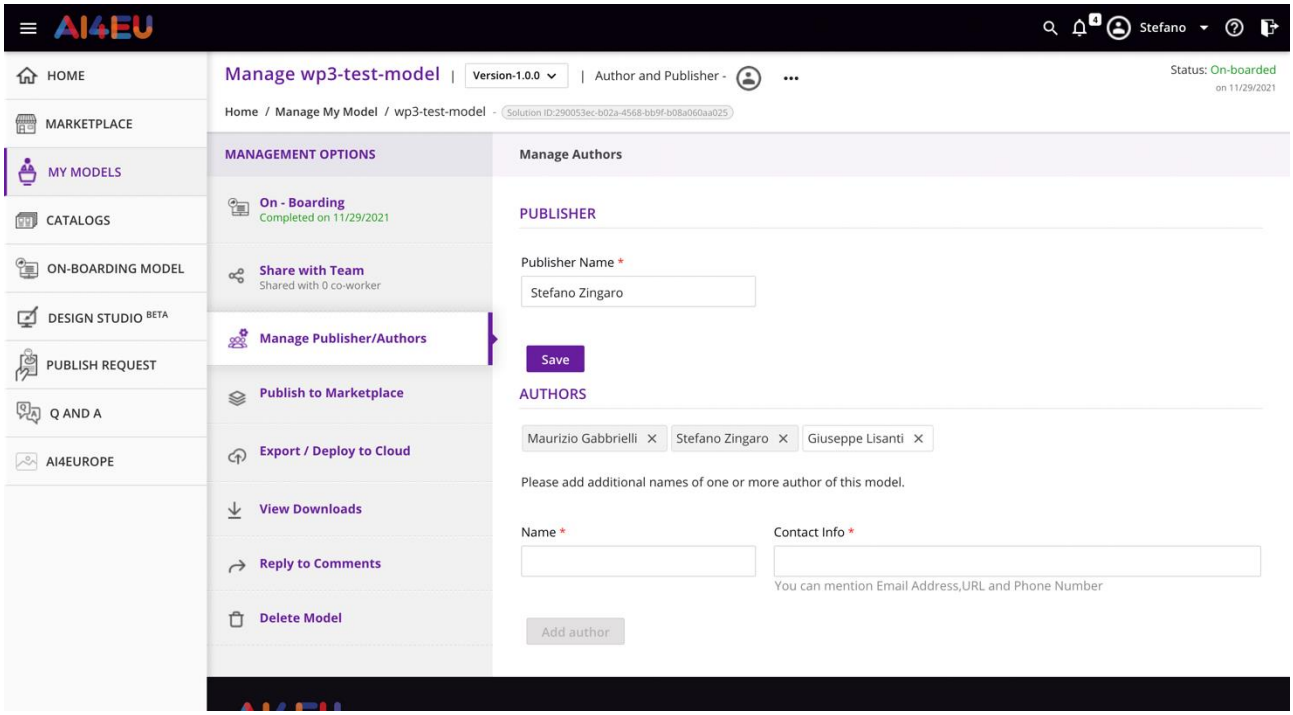


- The metadata of the template must be completed with information regarding 'Author(s)' and 'Publisher' of the resource. In case that information are missing, a warning message will be displayed on the model management page under the section "Publish to Marketplace" (see Figure below).

[4] *All the tests presented were performed on the pre-production platform (https://aiexp-preprod.ai4europe.eu/); any differences with the production environment will be made known in subsequent versions of this documentation.*



- The name of the "Publisher" and the name(s) of the "Author(s)" can be added on the dedicated page in the "Manage Publisher/Authors" section of the model's management options (see Figure below).



Overview of the publication process

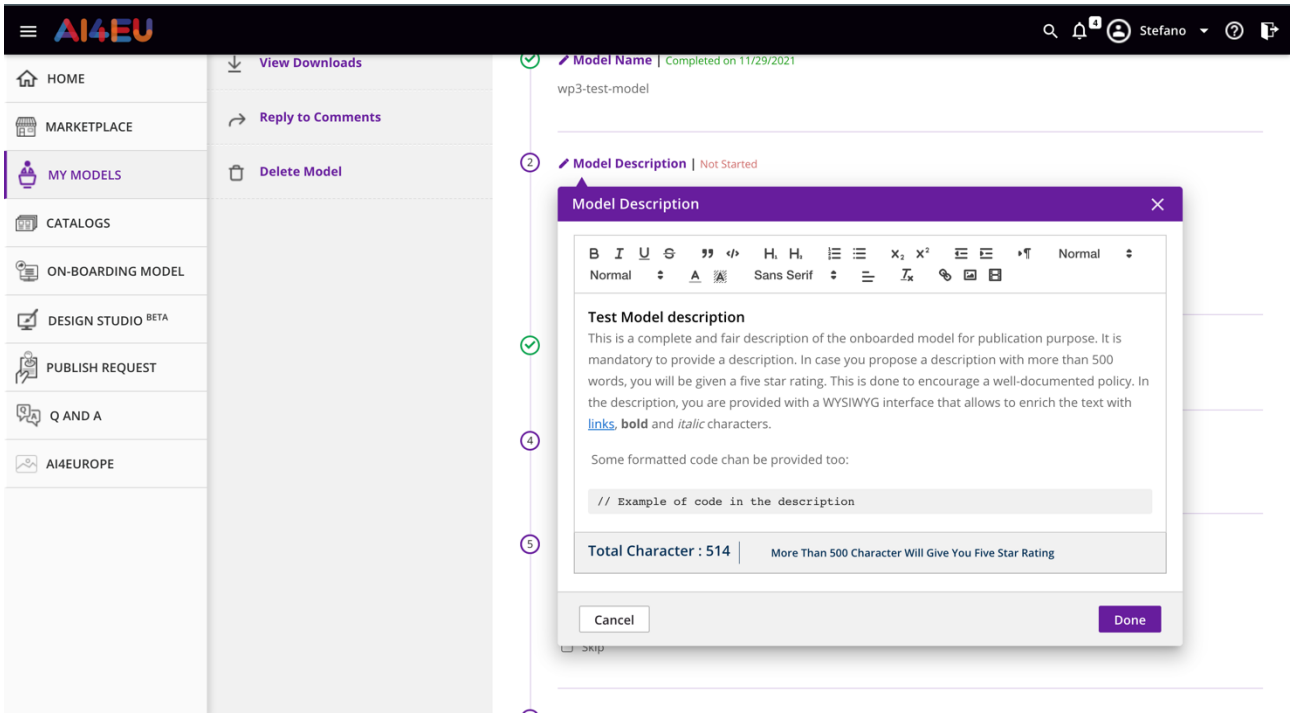
The publication process follows community-oriented principles. The resource store aims to be a "One Stop Shop of European AI resources" where each item is published and reviewed by the community of suppliers and consumers. In the infographic below, the macro-steps of the publication process are presented.

In this summary, several figures in the publication process are identified: the “AI Resource Provider” is the person or organization providing an AI Resource; he/she must be registered in the AI4EU platform; the “AI4EU Resource Reviewer” performs the review of a submitted AI Resource and is a qualified and trusted person of the community, and, finally, the AI4EU community is the set of registered users of the AI4EU platform.

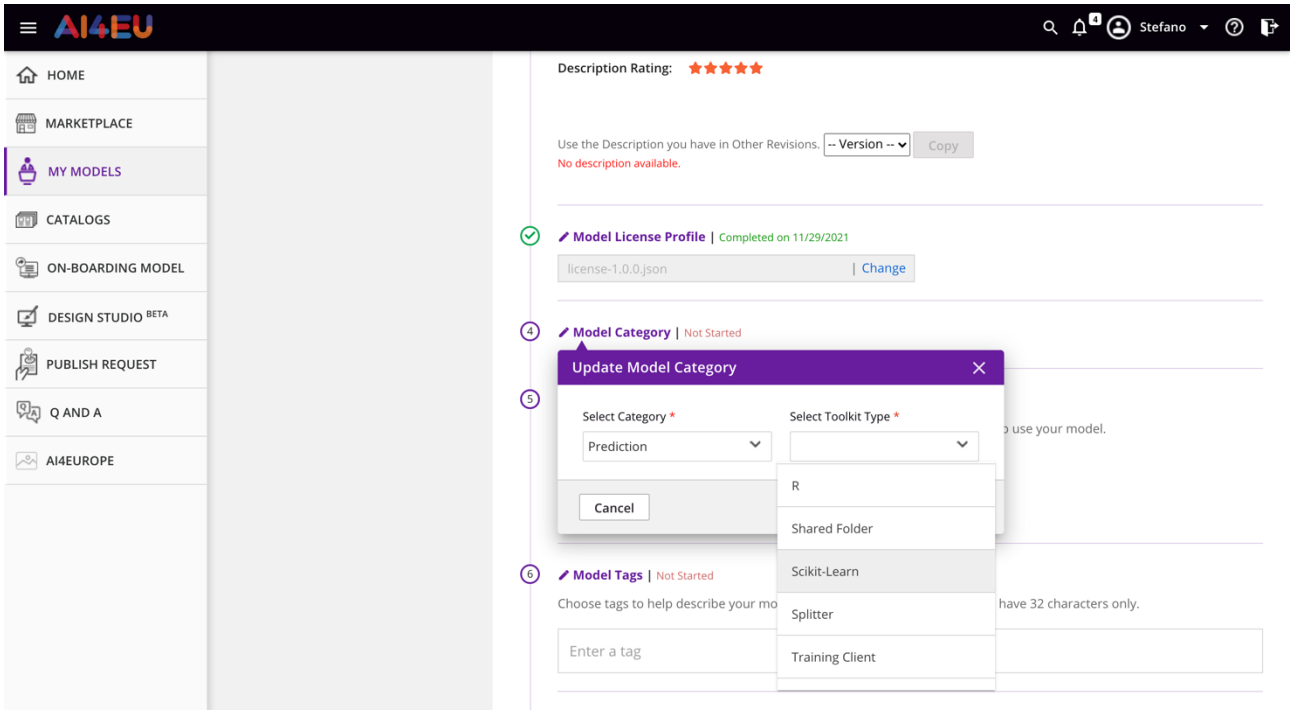
Step-by-Step submission process

In order to complete the publication process, a few more steps are required, i.e., model’s metadata must be added.

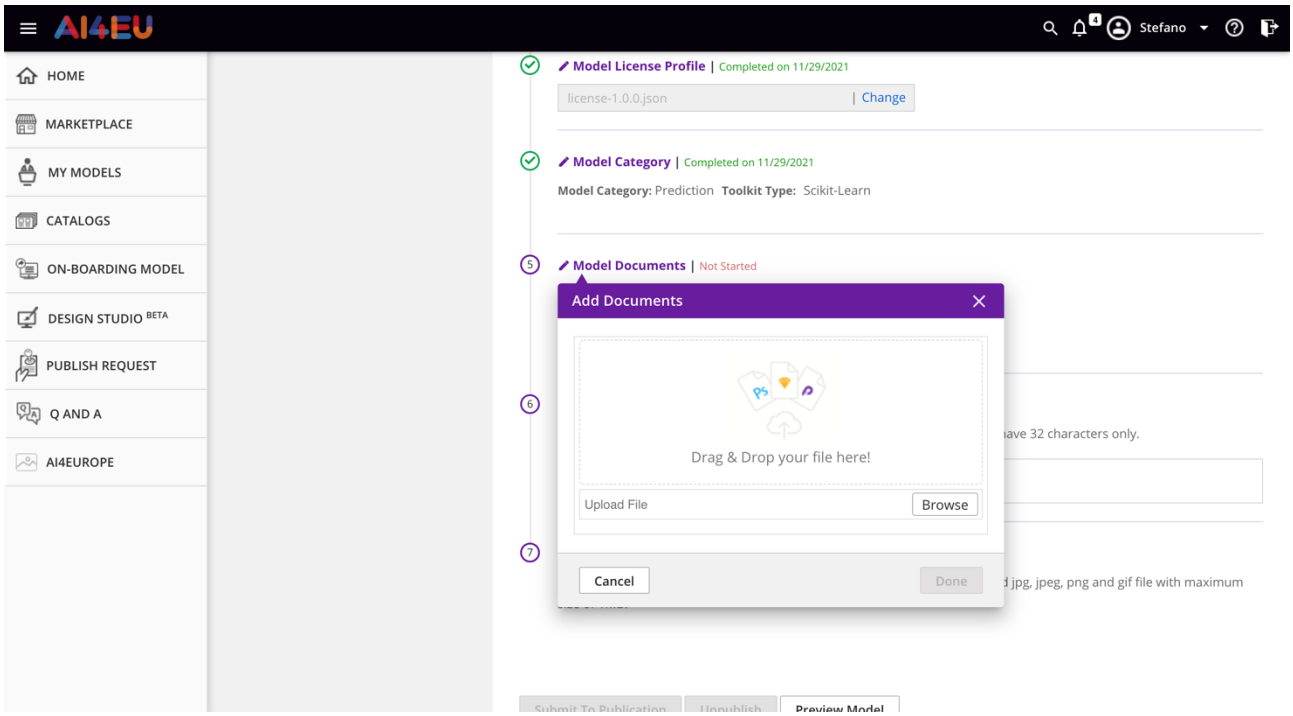
- The description of the model provides information concerning the use of the artefacts loaded during the onboarding process. It is mandatory to provide a description. In case you propose a description with more than 500 words, you will be given a five-star rating. This is done to encourage well-documented resources. The information can be uploaded through the appropriate interface, which can be accessed by clicking on the "Model Description" in the model management menu (see Figure below).



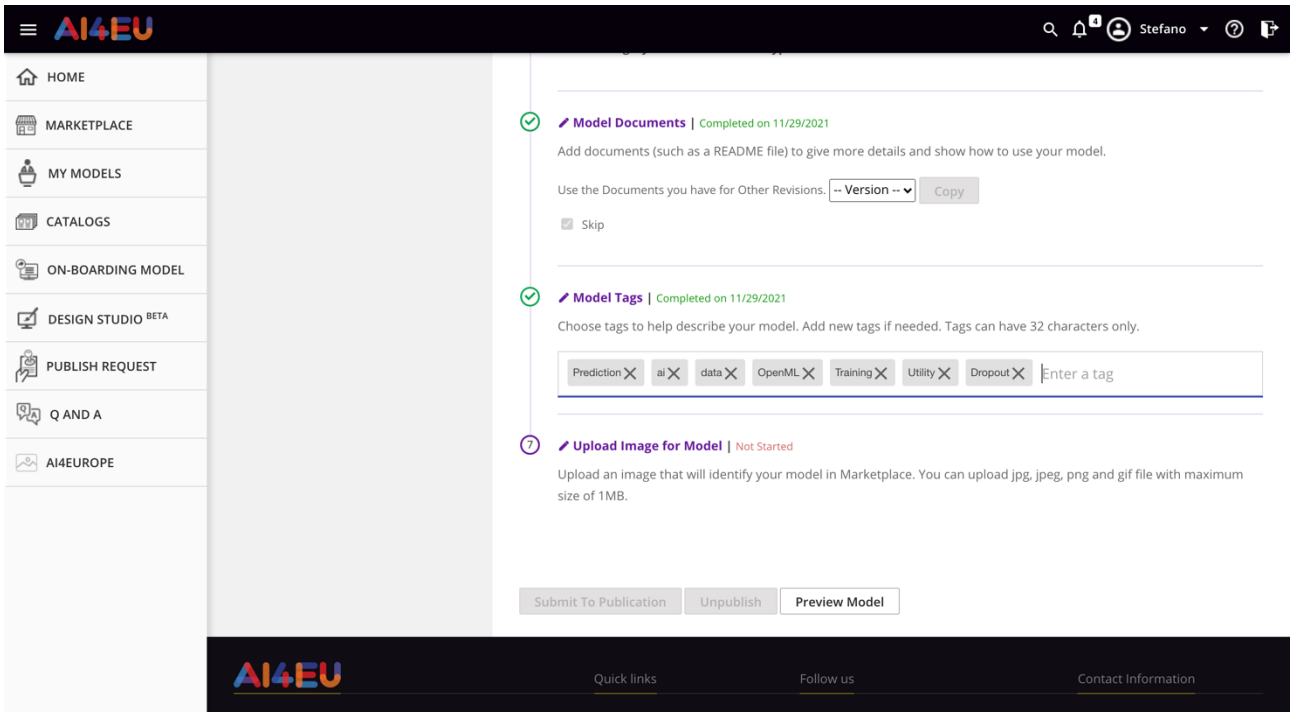
- For the resource to be easily found in the Marketplace, using the platform's search system, the user is asked to select a category, e.g., "prediction" or "regression", and the implementation kit for the model, e.g., JavaScript or C/C++.



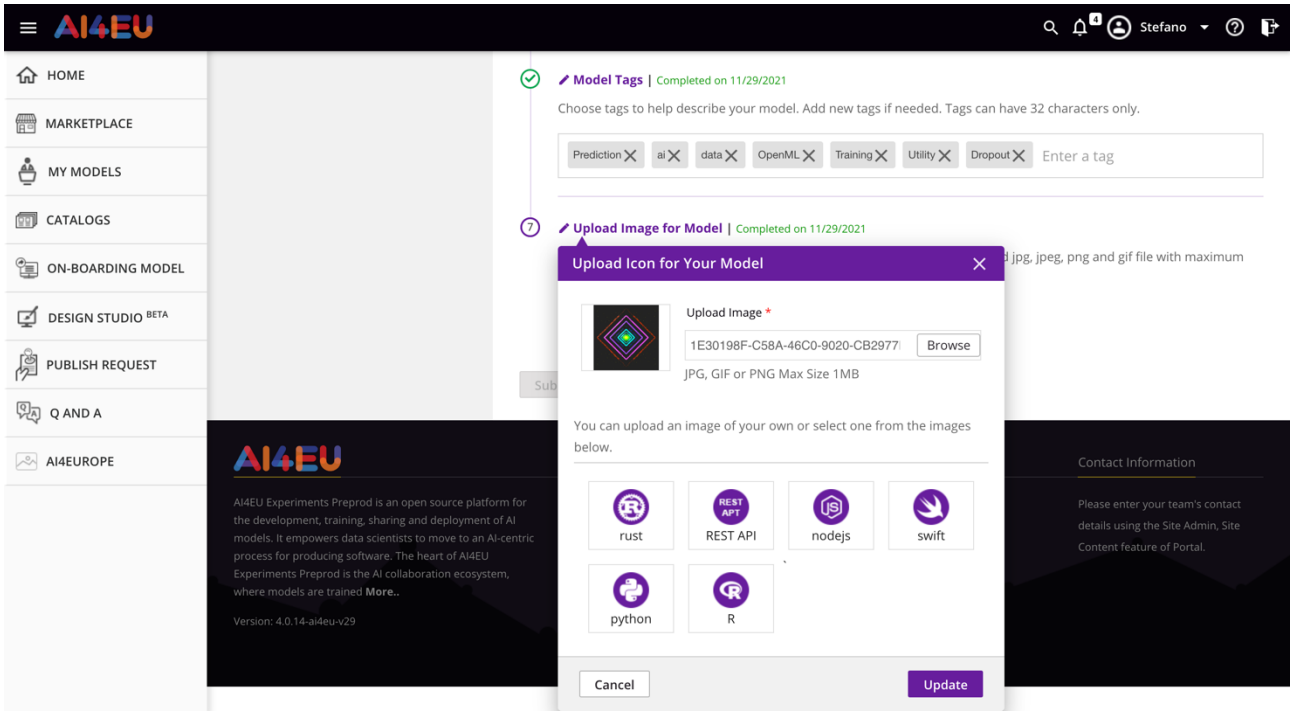
- However, it is optional to add documentation, e.g., README files, or other resources useful to the consumer of the resource.



- Tags should be inserted to provide a quick way of navigating through the resources in the Marketplace. You can select from those already in the system using the auto-complete function, or you can add new ones.



- It is mandatory to choose an image, a logo, or an icon to better identify the model in the Marketplace.



Once the prerequisites have been met and all the necessary metadata entered, you can preview the model or proceed with the submission for publication using the appropriate button.

It is important to note that the submission can be made both for publication in the "public" catalogue and for the "private" catalog (for internal use). The option can be set at the very beginning of the model management dedicated page.

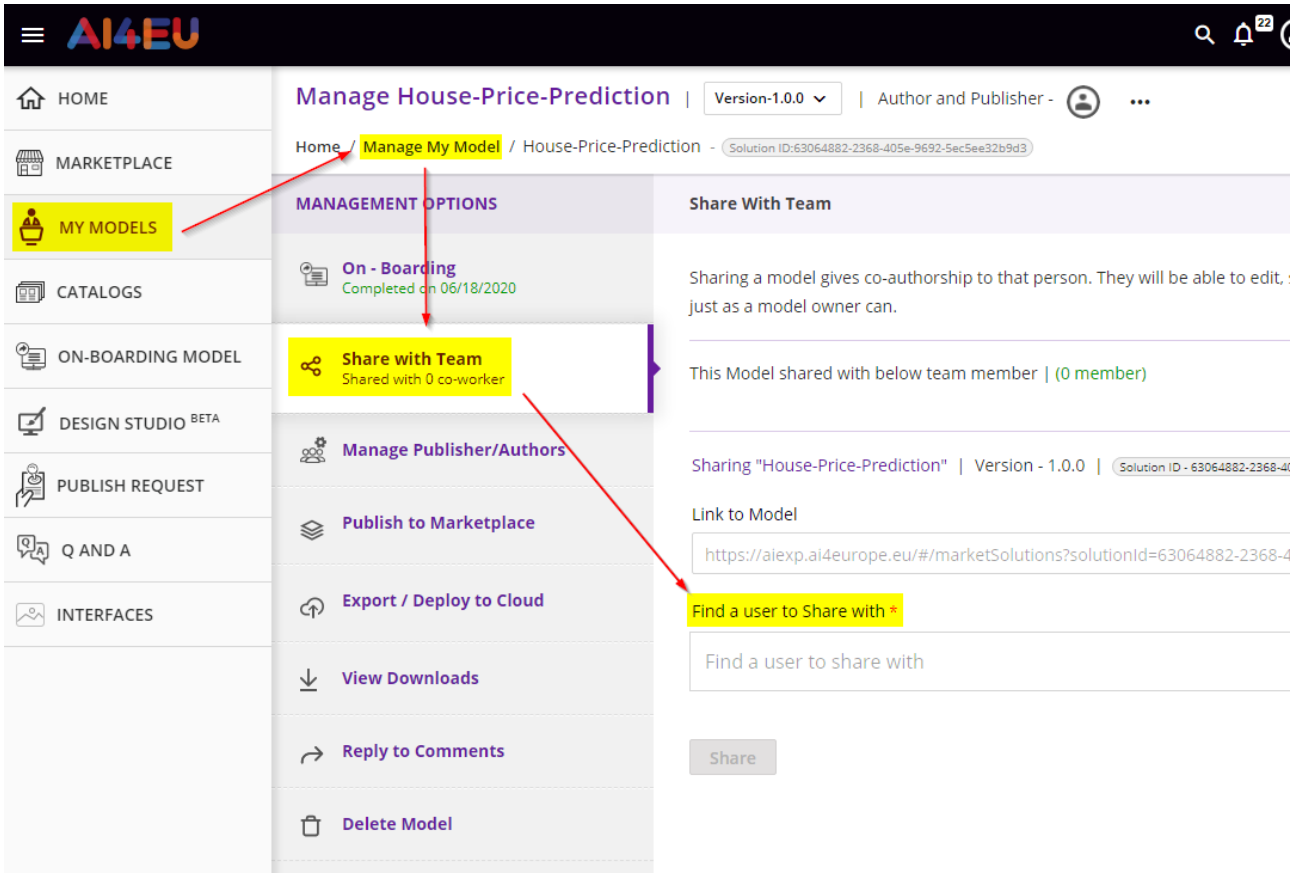
From the moment the submission request is made, the resource enters the "Request Approval" state.

If necessary, e.g., to correct the information entered, the approval request can be cancelled using the appropriate button “Withdraw Request”.

3.5. Collaboration

One of the important features of AI4EU Experiments is the possibility to work with other users on models and solutions. This can significantly improve know-how transfer and reduce the time to bootstrap AI adoption in companies. Typically, a mixed team consisting of external experts and company employees can collaborate on a proof of concept for to solve a problem with AI tools. This includes the creation of pipelines as well as the deployment and execution to assess and verify the results. The created models and solutions in the course of a collaboration need not to be published, so it is supported to work in private teams, where only the selected users see the models and results.

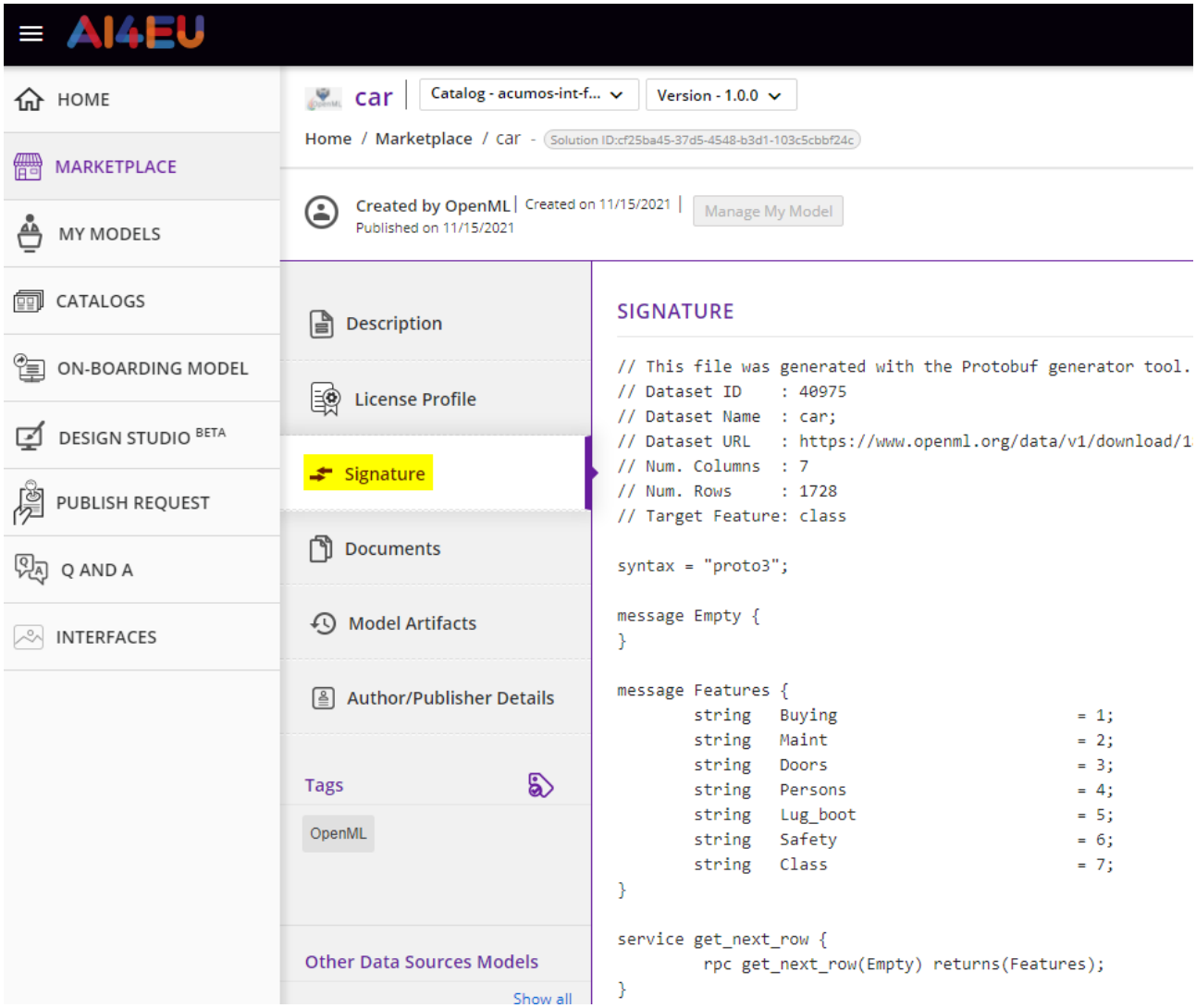
The most important element to enable this kind of collaboration is the "Share Model" feature: if the user goes to the "My Models" page and selects a model (or solution), and then clicks on "Manage My Model", the "Share with Team" dialog can be chosen:



Please be aware that sharing a model gives co-authorship to that person. They will be able to edit, share and publish, just as a model owner can.

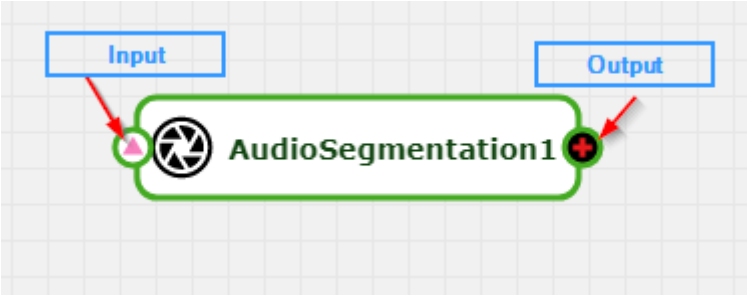
3.6. Visual Pipeline Composition

Models are the basic, re-usable building blocks in the Design Studio. It is these models that are combined together by the user to create complex AI application pipelines. Models offer a standard contract – a public interface definition to the external world. This contract specifies the details of the operation performed by the model, the input request (message) consumed by the model and the output response (message) produced by the model. Each model may support one or more operations – corresponding to the functions, such as prediction, classification, planning etc., performed by the model. In AI4EU Experiments, this contract is specified in the Protobuf file, which can easily be viewed on the Signature tab of the model details:



How to use the visual composition editor

Each operation consumes an input message and produces an output message which are represented by two ports – an input port and an output port. A model may have more than two ports if it provides (exposes) multiple operations (aka services).



1. Input Port – consumes the input message and provides the service, such as prediction or classification or regression to the caller/client. The input port represents the capability of the model. The client that needs a service to be performed need to send a request to input or the capability port of the model. Input ports have a white background.
2. Output Port – produces the output (response) message. Note that the output produced by an operation (say the Prediction message) needs not necessarily be consumed by the caller/client, but in fact needs to be fed to another Model which provides another service, such as classification (of the Prediction message). So, from a composition perspective, the output port represents a requirement that is satisfied by classification service. Output ports have a black background.

Similar symbols in the ports represent similar message types and hence can be connected. A connection must always connect an input and an output port. A port can have multiple connections.

In the Design Studio, a pair of ports are compatible if the requirement of one port can be matched with the capability of another port. Or if the output of one model can be consumed by the input port of another model, so as to get some service from the latter.

The matching criterion is based on comparing the Protobuf message signature of the output port to the message signature of the input port of another model.

A pair of output and input messages are compatible if all the following conditions are satisfied:

1. The number of tags in both their message signatures is the same
2. For each tag number, the fields on both the sides are of the same type
3. For each tag number, the fields on both the sides have the same role – repeated, optional, etc.

NOTE: the field names and message names are not taken into consideration for determining compatibility.

Elements of a pipeline

Models or Nodes have different roles in a pipeline:

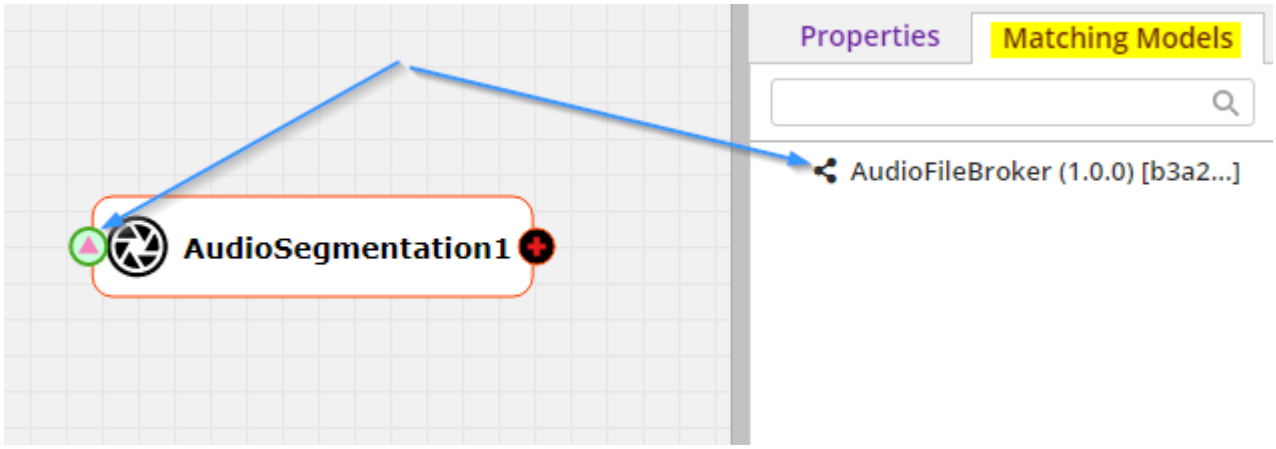
Databrokers are shallow components that make the data accessible for the pipeline via gRPC/protobuf. Typically, the dataset is not included or contained in the databroker, it only makes the connection. Often, a databroker is quite specific to an execution environment.

AI Models provide modular AI services that can be combined.

Shared folders are accessible by all connected nodes can be used to share big binary files among pipeline nodes, like video or audio files. Shared folders can have subfolders and thus provide a complete filesystem structure to the pipeline.

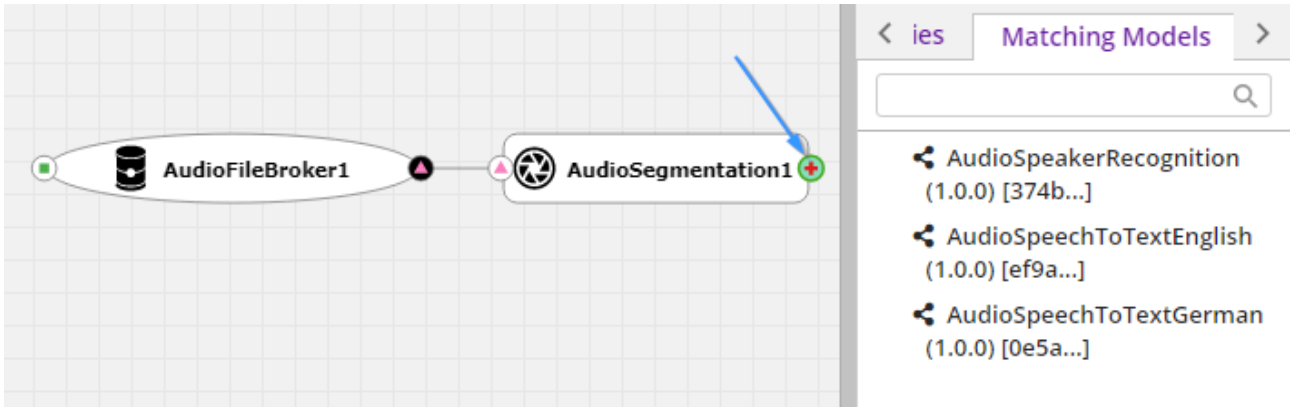
Composing a pipeline leveraging the matching models feature

As more and more models get published in the catalog, it becomes even more important to find matching models when building a pipeline. The design studio supports this via the "Matching Models" panel. After a click on the port that needs to be connected, in this example the input port of the AudioSegmentation1-Node on the left-hand side of the node, the matching models panel shows the AudioFileBroker as a match.

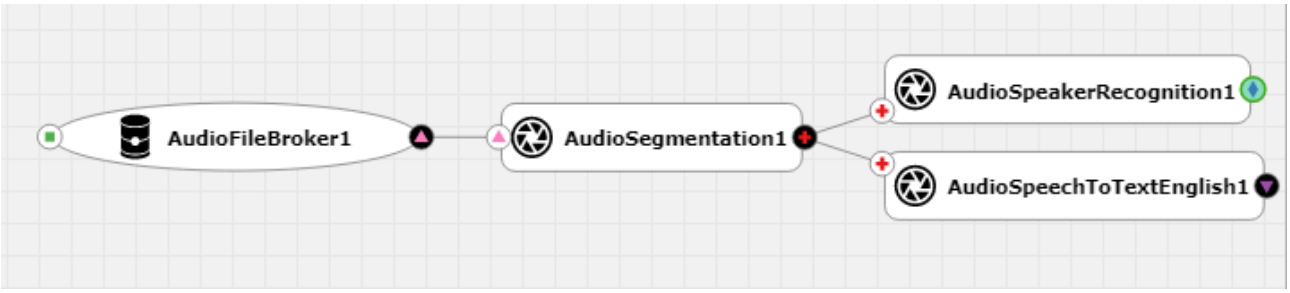


The node can directly be dragged from the matching models panel to the canvas and then connected. The next step is to continue the pipeline to the right, so we click on the output port of the AudioSegmentation1-node, and we can see three matching models in the panel:

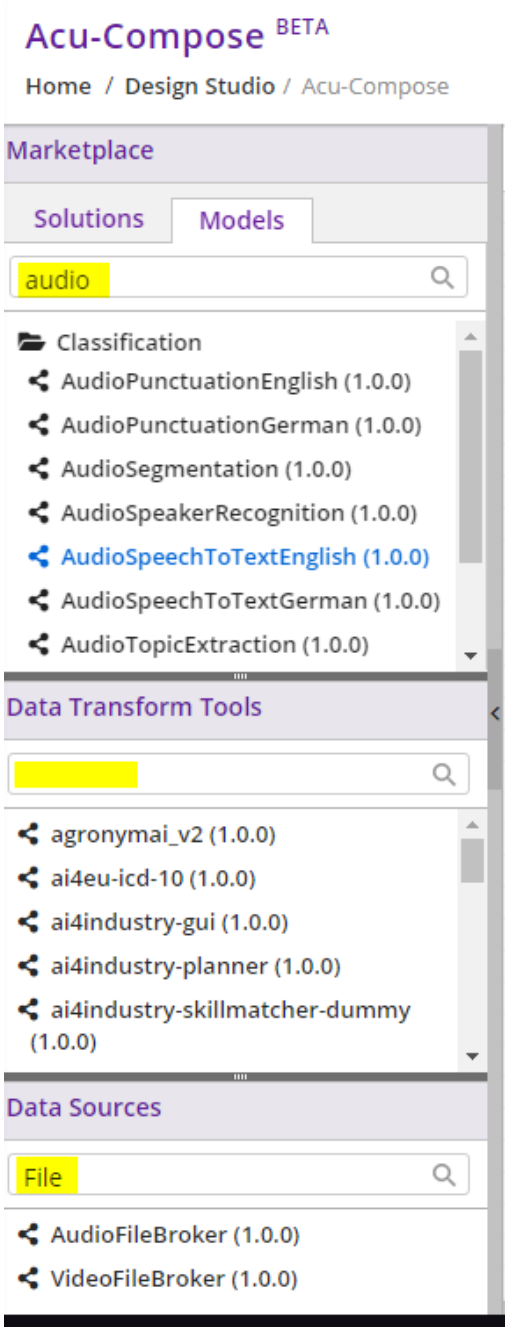
- AudioSpeakerRecognition
- AudioSpeechToTextEnglish
- AudioSpeechToTextGerman



All three of them have a matching input port and we continue the pipeline by opening two parallel paths for speech-to-text and speaker-recognition:



When looking for models, the three panels on the left-hand side of the canvas holding the complete lists of models, can be used to filter or search:

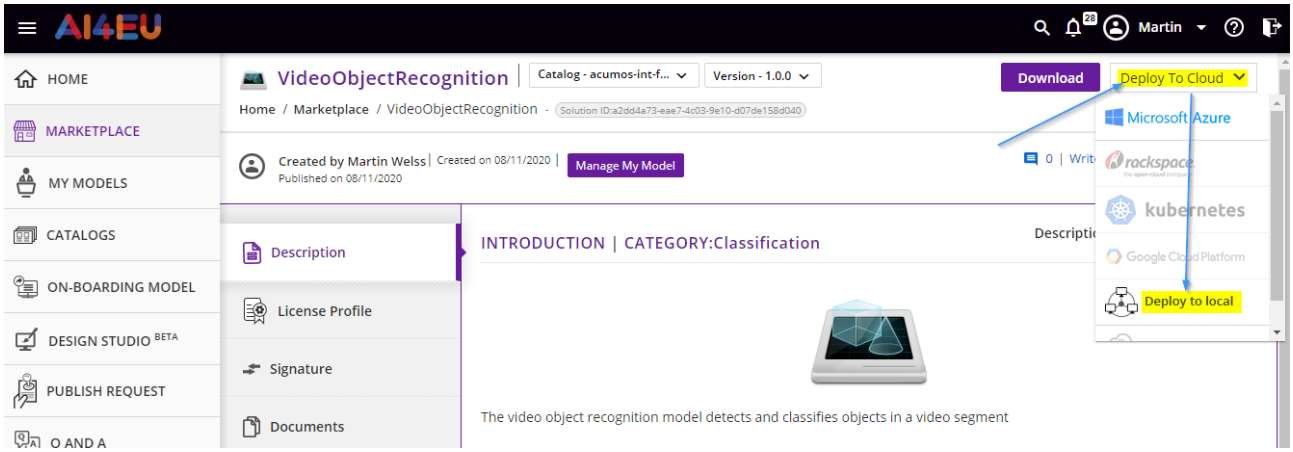


3.7. Deployment to an Execution Environment

An instance of AI4EU Experiments hosts the catalog of AI models and pipelines, offers collaboration and visual composition, but it is not the place to execute a pipeline. To do that, a pipeline or model must be deployed to an execution environment (see section 2.3). The architecture allows to support many different execution environments, it is quite easy to add support for new ones that even fit into local or customer specific MLOps workflows. Currently, the deployment to Kubernetes is supported.

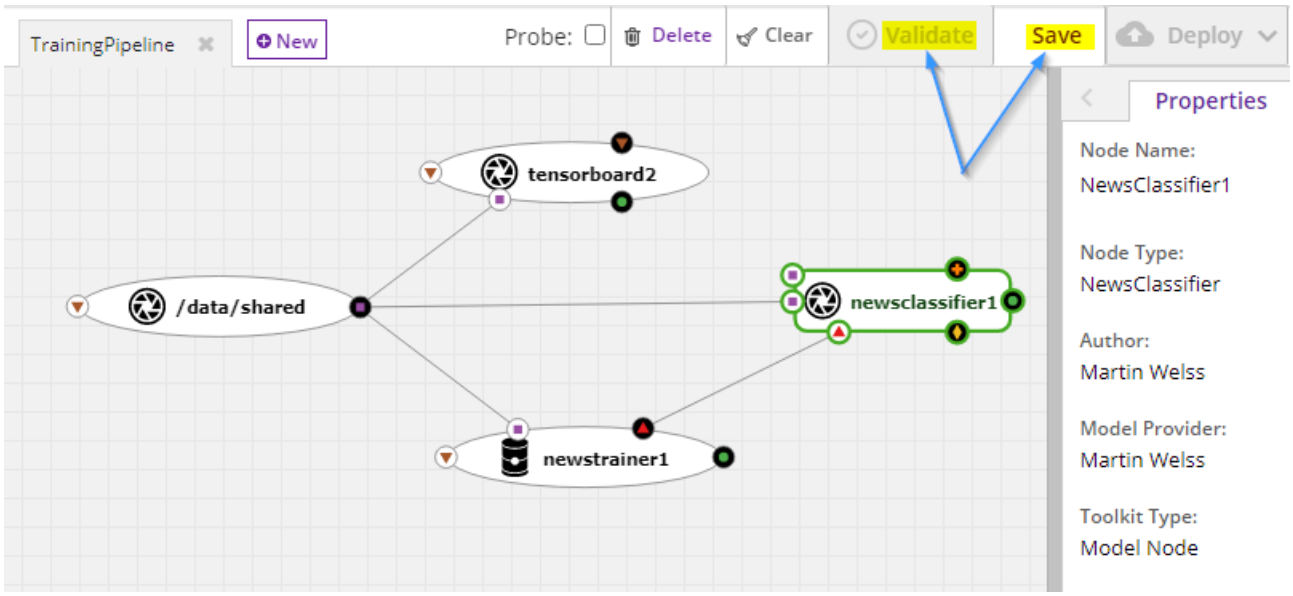
If a pipeline is to be deployed, then the advanced AI4EU orchestrator is automatically added to the deployment. The orchestrator dispatches the messages between the nodes according to the pipeline definition (blueprint.json).

The deployment of a single model can be initiated in the model detail page:

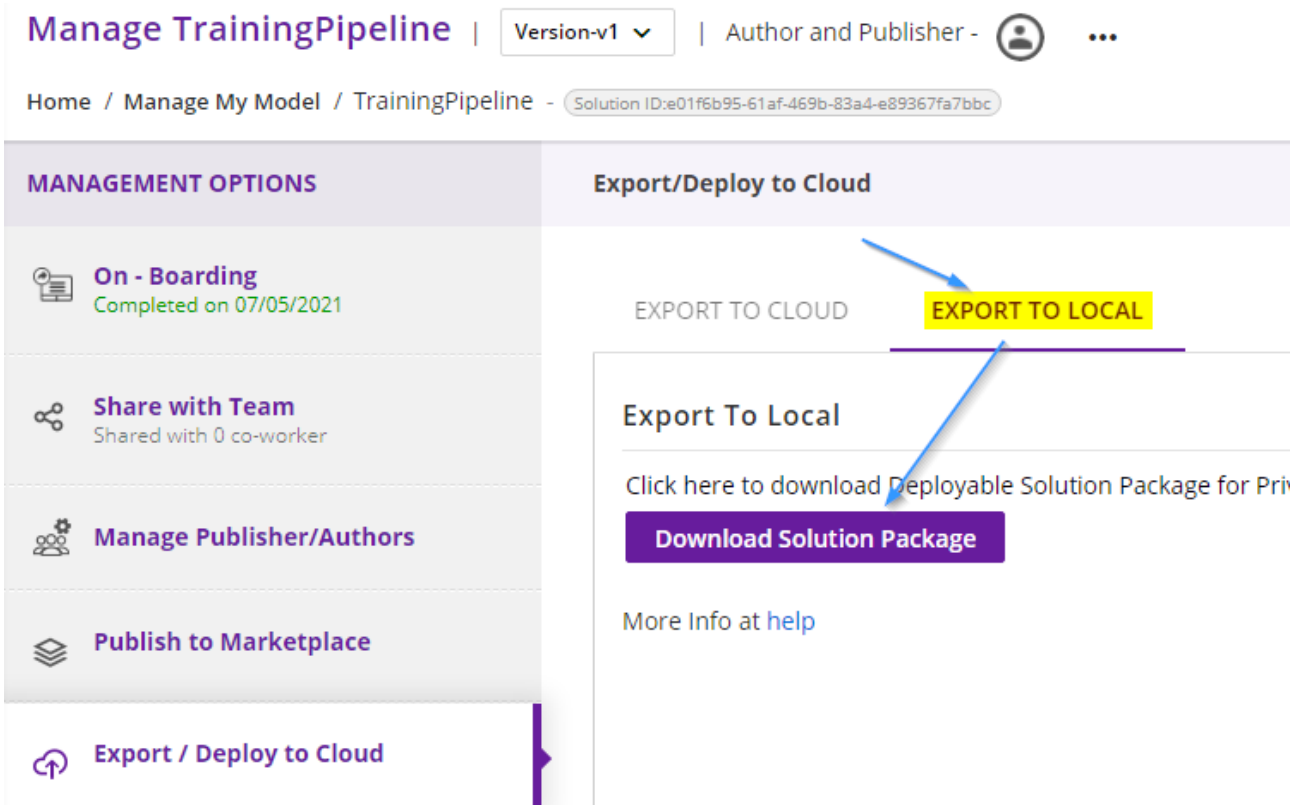


Then a dialog will open to download a file called soltion.zip which contains all necessary artifacts and scripts to deploy the model to a private Kubernetes cluster.

To deploy a pipeline, it must be saved and validated first:



Then choose "Deploy" -> "Deploy to local" and a new dialog will open. There you have to click on "Export to local", to download the solution package:



The solution.zip must be copied to a Kubernetes cluster where the user can use the kubectl command to deploy artifacts. After the solution.zip is extracted, the following files and folders are created:

```
mw@mwtest:~/demo$ ls -1
blueprint.json
deployment-readme.txt
deployments
dockerinfo.json
kubernetes-client-script.py
microservice
orchestrator_client
protobuf
requirements.txt
solution.zip
```

- the file deployment-readme.txt contains instructions on how to deploy and run the pipeline
- blueprint.json is the pipeline definition
- the folder deployments contains the templates for Kubernetes yaml-files like deployments and services
- the file dockerinfo.json contains the network addresses and ports of the pipeline nodes (used by the orchestrator)
- the kubernetes-client-script.py completes the deployment templates according to local resources, especially free ports, and installs them on the cluster in the given namespace
- the folder microservice contains the protobuf files for the pipeline nodes
- the folder protobuf contains the protobuf file for the orchestrator

- the folder orchestrator_client holds the script to start pipeline execution by triggering the orchestrator

Step 1:

Create a Kubernetes namespace for the pipeline

```
kubectl create ns demo
```

Step 2:

Make sure python yaml and grpc is installed.

You might use

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

to install necessary requirements for Kubernetes deployment script and orchestrator client.

Step 3:

Now run the python script kubernetes-client-script.py. You have to provide namespace as parameter to the script.

```
python3 kubernetes-client-script.py -n demo
get_namespaces: output NAME              STATUS   AGE
demo              Active   119s
type <class 'str'>
Given namespace is active
apply_deployment_services file_name= /home/mw/demo/deployments/newsclassifier1_deployment.yaml
set_image_pull_policy setting imagePullPolicy to Always
  apply got ['deployment.apps/newsclassifier1', 'created\n']
web_ui_service file_name = /home/mw/demo/deployments/tensorboard1_service.yaml node_port = 30013
  added webui suffix
apply_deployment_services file_name= /home/mw/demo/deployments/tensorboard1_service_webui.yaml
set_node_port in /home/mw/demo/deployments/tensorboard1_service_webui.yaml to 30013
  apply got ['service/tensorboard1webui', 'created\n']
apply_deployment_services file_name= /home/mw/demo/deployments/tensorboard1_service.yaml
set_node_port in /home/mw/demo/deployments/tensorboard1_service.yaml to 30012
  apply got ['service/tensorboard1', 'created\n']
apply_deployment_services file_name= /home/mw/demo/deployments/newstrainer1_deployment.yaml
set_image_pull_policy setting imagePullPolicy to Always
  apply got ['deployment.apps/newstrainer1', 'created\n']
apply_deployment_services file_name= /home/mw/demo/deployments/orchestrator_deployment.yaml
set_image_pull_policy setting imagePullPolicy to Always
  apply got ['deployment.apps/orchestrator', 'created\n']
web_ui_service file_name = /home/mw/demo/deployments/orchestrator_service.yaml node_port = 30015
  added webui suffix
apply_deployment_services file_name= /home/mw/demo/deployments/orchestrator_service_webui.yaml
set_node_port in /home/mw/demo/deployments/orchestrator_service_webui.yaml to 30015
  apply got ['service/orchestratorwebui', 'created\n']
apply_deployment_services file_name= /home/mw/demo/deployments/orchestrator_service.yaml
set_node_port in /home/mw/demo/deployments/orchestrator_service.yaml to 30014
  apply got ['service/orchestrator', 'created\n']
web_ui_service file_name = /home/mw/demo/deployments/newstrainer1_service.yaml node_port = 30017
  added webui suffix
apply_deployment_services file_name= /home/mw/demo/deployments/newstrainer1_service_webui.yaml
set_node_port in /home/mw/demo/deployments/newstrainer1_service_webui.yaml to 30017
  apply got ['service/newstrainer1webui', 'created\n']
apply_deployment_services file_name= /home/mw/demo/deployments/newstrainer1_service.yaml
set_node_port in /home/mw/demo/deployments/newstrainer1_service.yaml to 30016
  apply got ['service/newstrainer1', 'created\n']
apply_deployment_services file_name= /home/mw/demo/deployments/tensorboard1_deployment.yaml
set_image_pull_policy setting imagePullPolicy to Always
  apply got ['deployment.apps/tensorboard1', 'created\n']
web_ui_service file_name = /home/mw/demo/deployments/newsclassifier1_service.yaml node_port = 30019
  added webui suffix
apply_deployment_services file_name= /home/mw/demo/deployments/newsclassifier1_service_webui.yaml
set_node_port in /home/mw/demo/deployments/newsclassifier1_service_webui.yaml to 30019
  apply got ['service/newsclassifier1webui', 'created\n']
```

```
apply_deployment_services file_name= /home/mw/demo/deployments/newsclassifier1_service.yaml
set_node_port in /home/mw/demo/deployments/newsclassifier1_service.yaml to 30018
  apply got ['service/newsclassifier1', 'created\n']
apply deployment services file name= /home/mw/demo/deployments/pvc.yaml
WARNING: set_image_pull_policy encountered incompatible input file /home/mw/demo/deployments/pvc.yaml
  apply got ['persistentvolumeclaim/pipeline', 'created\n']
{'tensorboard1webui': 30013, 'tensorboard1': 30012, 'orchestratorwebui': 30015, 'orchestrator': 30014,
'newstrainer1webui': 30017, 'newstrainer1': 30016, 'newsclassifier1webui': 30019, 'newsclassifier1':
30018}

Start updating the docker info Json :
update_node_port: [{'container_name': 'tensorboard1', 'ip_address': 'tensorboard1', 'port': 30012},
{'container_name': 'newstrainer1', 'ip_address': 'newstrainer1', 'port': 30016}, {'container_name':
'newsclassifier1',    'ip_address':    'newsclassifier1',    'port':    30018},    {'container_name':
'orchestrator', 'ip address': 'orchestrator', 'port': 30014}]

 Docker info file is successfully updated
Node IP-address : 202.61.249.225
Orchestrator Port is : 30014
Please  run  python  orchestrator_client/orchestrator_client.py  --endpoint=202.61.249.225:30014  --
basepath=./
```

Step 4:

You need to verify that deployments and services are deployed successfully.

```
mw@mwtest:~/demo$ kubectl -n demo get all
NAME                                    READY   STATUS             RESTARTS   AGE
pod/newsclassifier1-6679b46d59-7cgvv    0/1     Running   0          3m44s
pod/newstrainer1-6dd65f587-gwsvc        1/1     Running            0          3m43s
pod/orchestrator-7869f67ddd-6f5x2       1/1     Running            0          3m43s
pod/tensorboard1-686875bcf9-4rnnj       1/1     Running            0          3m43s

NAME                              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/newsclassifier1                 NodePort      10.106.187.96        <none>
30018:30018/TCP   3m42s
service/newsclassifier1webui        NodePort        10.110.35.208          <none>
30019:30019/TCP   3m42s
service/newstrainer1                    NodePort      10.102.147.146       <none>
30016:30016/TCP   3m43s
service/newstrainer1webui             NodePort       10.102.195.76        <none>
30017:30017/TCP   3m43s
service/orchestrator                    NodePort      10.102.151.172       <none>
30014:30014/TCP   3m43s
service/orchestratorwebui             NodePort       10.107.24.36         <none>
30015:30015/TCP   3m43s
service/tensorboard1                    NodePort      10.98.25.121         <none>
30012:30012/TCP   3m44s
service/tensorboard1webui             NodePort       10.107.161.255       <none>
30013:30013/TCP   3m44s

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/newsclassifier1      0/1     1            0           3m44s
deployment.apps/newstrainer1         1/1     1            1           3m43s
deployment.apps/orchestrator         1/1     1            1           3m43s
deployment.apps/tensorboard1         1/1     1            1           3m43s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/newsclassifier1-6679b46d59    1         1         0       3m44s
replicaset.apps/newstrainer1-6dd65f587        1         1         1       3m43s
replicaset.apps/orchestrator-7869f67ddd       1         1         1       3m43s
replicaset.apps/tensorboard1-686875bcf9       1         1         1       3m43s
```

After running this command on terminal, you can see that corresponding pods are running successfully in given namespace.

If a shared folder is part of the pipeline, then the script will map an existing, free persistent volume to the shared folder and create the corresponding persistent volume claims in the node deployments. The persistent volume should be setup to support read-write-many. The persistent volume itself may have to be created in advance by the cluster administrator.

Then, finally, start the execution of the pipeline by running the orchestrator client:

```
python3 orchestrator_client/orchestrator_client.py --endpoint=202.61.249.225:30014
--basepath=./
```

The command has been printed as the last output line of the kubernetes-client-script.py, but can as well be found by running:

```
kubectl -n demo get service orchestrator
```

4. Annex

Collection of useful Links

- AI4EU Experiments: https://aiexp.ai4europe.eu
- AI4EU Experiments Tutorials: https://github.com/ai4eu/tutorials
- Youtube Playlist https://www.youtube.com/playlist?list=PLL80pOdPsmF6s6P6i2vZNoJ2G0cccwTPa
- Container Specification https://github.com/ai4eu/tutorials/tree/master/Container_Specification