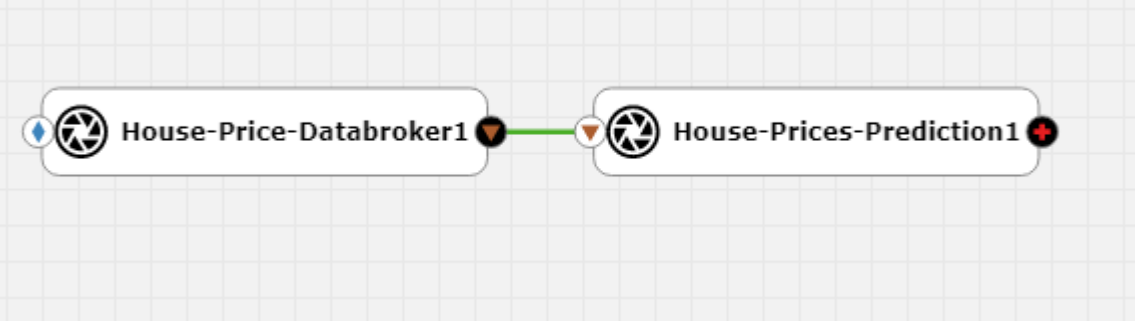
AI4EU Experiments Container Format



1. Introduction

This document 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 comopse 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 procecure calls supporting a broad range of programming languages and it is optimized for performance and high throughput.

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 used for piplines as long as it exposes a set of public methods via gRPC.

2. Define model.proto

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

- It should be self contained, thus contain the service definitions with all input and output data structures and no imports can be used
- The full feature set of protobuf v3 can be used (except import and enums)
- a container can define serveral methods, but all in the .proto file
- it must not have a package declaration, as this blocks automatic message dispatching when used as part ofa 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{`
there must not be curly braces after a rpc line	`rpc predict (AggregateData) returns (Prediction);`	`rpc predict (AggregateData) returns (Prediction) {}`

3. Create the gRPC docker container

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.

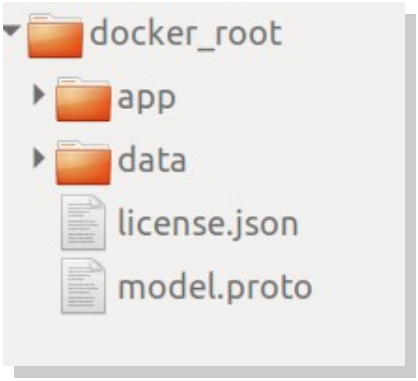
The gPRC server must listen on port 8061.

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 are 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:

https://docs.acumos.org/en/clio/submodules/license-manager/docs/user-guide-license-profile-editor.html

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 unpriviledged user should be created that runs the application, here is an example snippet from a Dockerfile:

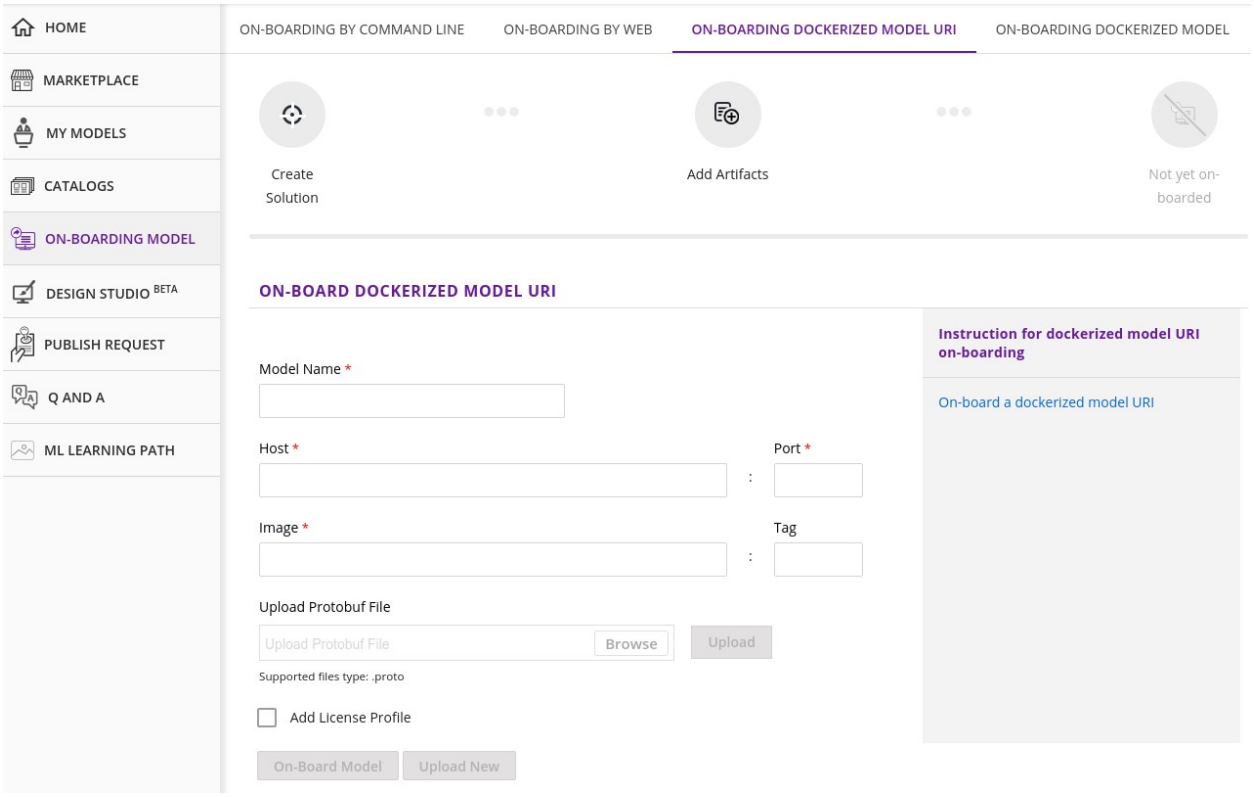
```
RUN useradd app
USER app
CMD ["java", "-jar", "/app.jar"]
```

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

4. Onboarding

The final step is to onboard the model. There are several was to onboard a model into AI4EU Experiments but currently the only recommended way is to use **"On-boarding dockerized model URI"**:

1. Upload your docker container to a public registry like Docker Hub
2. Start the onboarding process like in the screenshot below
3. Upload the protobuf file
4. Add license profile



5. First Node Parameters (e.g. for Databrokers)

Genereally 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” input datatype, in this case it could be google.protobuf.Empty or an Empty message type. Typically this concerns nodes of type Databroker as the usual starting point of a pipeline.

```
syntax = "proto3";
import "google/protobuf/empty";

message NewsText {
  string text = 1;
}
```

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

6. Scalability, GPU Support and Training

The potential execution environments range from Minikube on a Laptop over small Kubenetes 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)
- Some AI frameworks like PyTorch or Tensorflow can be used in a way to work with or without GPU with the same code.
- even training is possible, if the model exposes the corresponding methods in the protobuf interface