

D8.10 – Deep Learning system implementation

Project Information

Grant Agreement Number	958454
Project Full Title	Intelligent Water Treatment for water preservation combined with simultaneous energy production and material recovery in energy intensive industries
Project Acronym	intelWATT
Funding scheme	IA
Start date of the project	1 st October 2020
Duration	42 months
Project Coordinator	Andreas Sapalidis (NCSR)
Project Website	https://www.intelwatt.eu

Deliverable Information

Deliverable n°	8.10
Deliverable title	Deep Learning system implementation
WP no.	8
WP Leader	Techedge
Contributing Partners	
Nature	Other
Authors	Javier Alejandro Vicente Napolitano, Ciro Navarro Aceto, Manuel Torres Brabo
Contributors	
Reviewers	Andreas Sapalidis
Contractual Deadline	M20
Delivery date to EC	31/05/2022

Dissemination Level

PU	Public	✓
PP	Restricted to other programme participants (incl. Commission Services)	
RE	Restricted to a group specified by the consortium (incl. Commission Services)	
CO	Confidential, only for the members of the consortium (incl. Commission Services)	

Document Log

Version	Date	Author	Description of Change
V1.0	02/05/2022	Javier Alejandro Vicente Napolitano	First release
V1.3	13/05/2022	Javier Alejandro Vicente Napolitano	First draft
V2.0	20/05/2022	Ciro Navarro Aceto	Final draft
V2.1	24/05/2022	Manuel Torres Brabo	Review of the final draft
V3.0	25/05/2022	Manuel Torres Brabo	Final document
V3.1	27/05/2022	Andreas Sapalidis	Final Review



Table of Contents

- 1 Executive Summary..... 5
- 2 Overall Architecture..... 6
- 3 On premises inference implementation 8
 - 3.1 Agent Containerization Architecture 9
 - 3.2 Software requirements 10
 - 3.3 Hardware requirements 10
- 4 Cloud Inference implementation.....11
 - 4.1 General Architecture 11
 - 4.2 Cloud environment definition 11
 - 4.3 Cloud computing instance 12
 - 4.4 Deep Reinforcement Learning agent 13
 - 4.5 Endpoint 13
- 5 Cloud Continuous training implementation.....15
- 6 References.....16



List of figures

- Figure 1. Deep Learning System Implementation Architecture 7
- Figure 2. Hardware and Software components of Docker containerized agents. 8
- Figure 3. 2-containers Agent Implementation. 9
- Figure 4. Necessary libraries to be installed in every container..... 9
- Figure 5. Cloud inference architecture diagram.....11
- Figure 6. Example of environment definition on Azure platform.12
- Figure 7. Computing instance example on Azure.....12
- Figure 8. Model list on Azure cloud13
- Figure 9. Example Azure Endpoint.....14
- Figure 10. Continuous training implementation in Azure.....15



1 Executive Summary

In the overall systems architecture of intelWATT the Deep Learning system is responsible for the control and optimization of membrane processes, providing a guideline for the set point of control variables for the processes involved.

The Deep Learning system is based on the principles of Deep Reinforcement Learning (DRL) and is composed of agents interacting with the environment and adjusting their actions towards the objectives based on the outcomes of previous actions.

The DL system has two different modes of operation: training and inference. The training system uses real pilot data to update the agent's parameters. The inference system can run on the cloud, but in some cases, trained agents must be deployed next to the case study's environment to suggest control actions on the processes with minimum delay.

During training, the main needs of the system are accurate and abundant data for the updates and computing power to speed up the training through hundreds of thousands of iterations. This subsystem is deployed in the cloud and uses scalable and elastic resources to supply that computing power.

For inference, the system must be available as close to the real environments as possible and the main requisites are ease of deployment and portability. The inference system is containerized so that it can be deployed in the cloud or on site if needed and will run on different hardware configurations. For the on-premises deployments data will be read from the sensors (via the on-premises data concentrators) and control actions sent directly to the PLCs.

2 Overall Architecture

The Deep Learning (DL) system is composed of two different processes: inference and continuous training. Until the pilots are finished and ready to run, the first agent, trained with simulators and laboratory data, will be deployed for each of the membrane process within the different case studies. The agents are responsible for the inference process, meaning the output of these agents will optimize the reward function and therefore the process itself. As the simulator's precision is limited, the first version of the agents only has a partial understanding of the environment. The continuous training is necessary to gain a deeper understanding of the process and achieve better results, regarding the defined reward function.

There are several possible components and structures that can be used to define the overall architecture of both the inference and the continuous training processes. Taking into consideration this fact, the desired characteristics of the implementation must be listed:

- Automated: the system must work autonomously, with no human intervention required. The inferences will be suggested/applied to the process after a fixed amount of time (depending on process requirements). The continuous training is performed after N inference steps through batches of data. The old agent version is stored and the new one deployed into production.
- Available: the system must ensure the availability of inferences during the pilot's operating time. Delays on the deployment of new agents are acceptable.
- Easy to maintain: if any problem comes up, it should not require a substantial effort to solve it and start operating again.
- Adaptable: due to the variety of possible solutions, the possibility to upgrade/improve the system cannot be discarded. The needs of the process will drive the development of the system.

The requirement of being adaptable is essential to the whole DL system. Several different technologies will be tested during the project and the best ones will be adopted. The current proposed solution is the basis for further tests and development that are expected during the evolution of the project. Hence, the final system can be completely different from the one described in this document. However, the underlying operation of the system, the inference and continuous training, will be maintained.

To ensure the availability of the DL system, the agent will be deployed both on premises and in the cloud. This setup allows inferences to be available even if the internet connection is not working. Moreover, if any process has low latency requirements, the on-premises implementation will be able to fulfil these needs. The specific requirements for the on-premises implementation will be detailed in the corresponding section. To reduce the investment on hardware, only the inference process will be carried out on-premises.

The cloud implementation allows the system to scale the inference and continuous training processes as needed and to store the sequence of trained agents. Figure 1 shows a diagram of the dataflow between pilot sensors, on-premises inference implementation and cloud inference and training implementations. Blue arrows correspond to data or inferences, whereas green ones denote a new model

Sensor data will be sent both to the on-premises system and to the data lake in the cloud. The on-premises system can directly send inferences to the pilot actuator. On the other hand, the cloud system can also infer the same output, as the agent deployed in both systems will be exactly the same. After N inference steps the current agent is updated (continuous training). After the training process is finished, the new agent is deployed on-premises and stored and deployed in the cloud. The new agent just takes over when the system is completely updated, meanwhile the old version is operating.

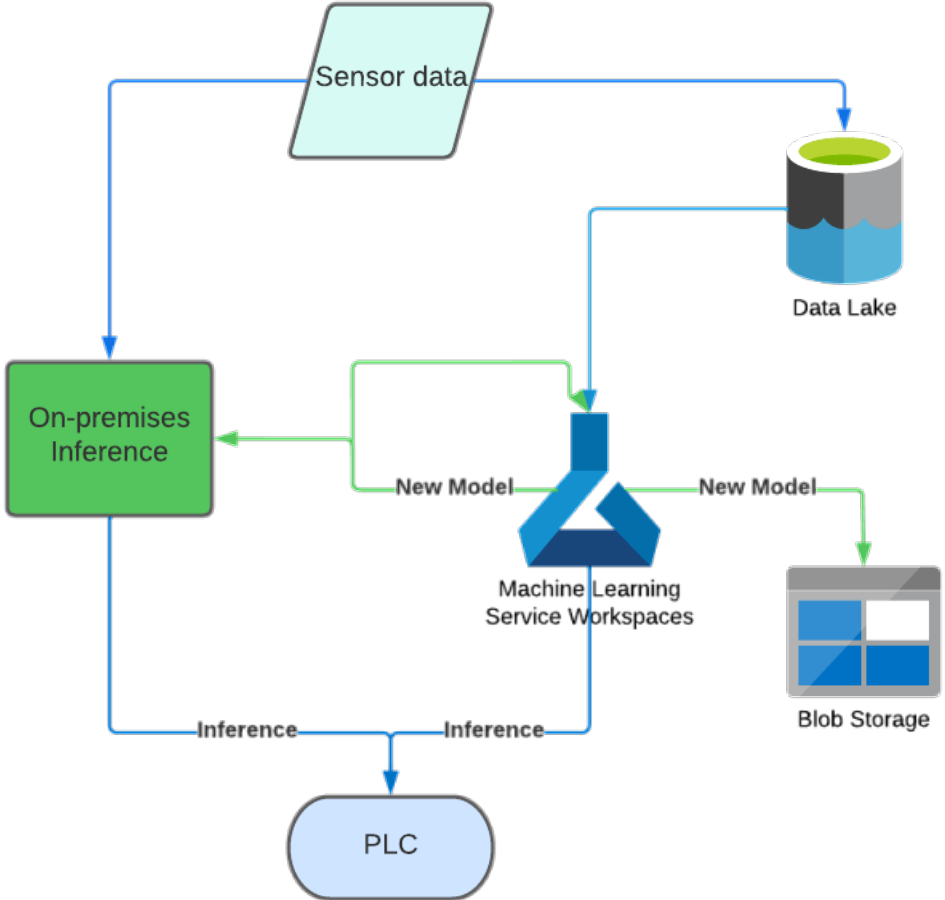


Figure 1. Deep Learning System Implementation Architecture



3 On premises inference implementation

Currently one of the most widely used ways of deploying applications and Machine Learning or Deep Learning models is through containers. Docker is one of the available solutions to containerize applications. To be clear about what a container means, the Docker’s website definition of container is the following:

“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.

Container images become containers at runtime and in the case of Docker containers – images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.”

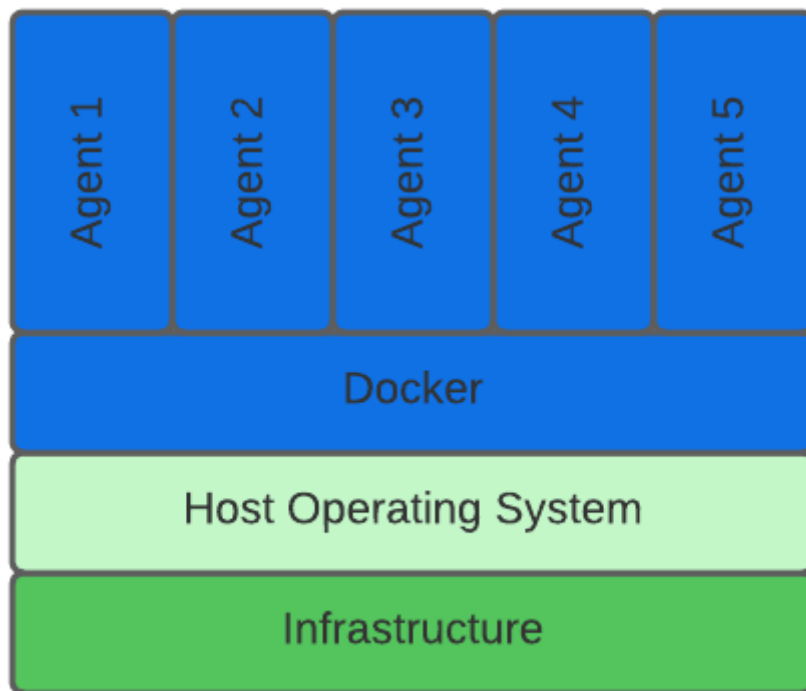


Figure 2. Hardware and Software components of Docker containerized agents.

Due to the several advantages and the ease of deployment of containers, they are the best choice to the on-premises implementation. All the necessary Docker containers will be provided by Techedge to be implemented in the pilots.

3.1 Agent Containerization Architecture

For each agent represented in Figure 2, there are actually two agents that can be a part of two different containers or a single one. These two agents, that can work in parallel, are two different versions of the same agent. The old agent is the last version before the update of the parameters, whereas the new agent is the updated version. This configuration was chosen for some reasons. If the new agent, by any reason, has a degraded performance or any problem occur during the update, the old agent can start controlling the system again. Almost no time is lost during this operation switch. On the other hand, after each update of the parameters, the new agent must be deployed in the container of the old agent. After this process, the new agent will be the old agent, and the old agent is now the new agent. This setup allows the update of the agent while the other agent is currently operating the control of the membrane process.

Long story short, the 2-containers agent implementation allows the update of one agent during the operation of the control system and the switch back to the old version if anything unexpected occur.

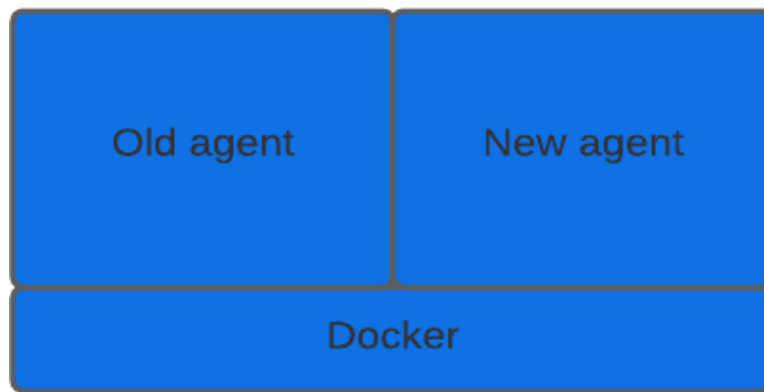


Figure 3. 2-containers Agent Implementation.

Other than the agent, each container has all the necessary libraries and its dependencies installed. As all agents are trained in Python, the basic programming language is Python. Every Reinforcement Learning library uses one of the most used Deep Learning libraries, TensorFlow or Pytorch. These libraries are the primary tool that runs on Python. The whole agent architecture, neural network structure and parameters, are saved in a specific format that depends on the library used to train it. By now, stable-baselines3 and RLlib are the ones that are being considered for the training of the agents. As a last step, for the inference to be performed by the agent, the input data must be manipulated and fed into the correct format to the agent. Figure 4 shows the necessary libraries in each container to output the control inferences of the membrane processes.

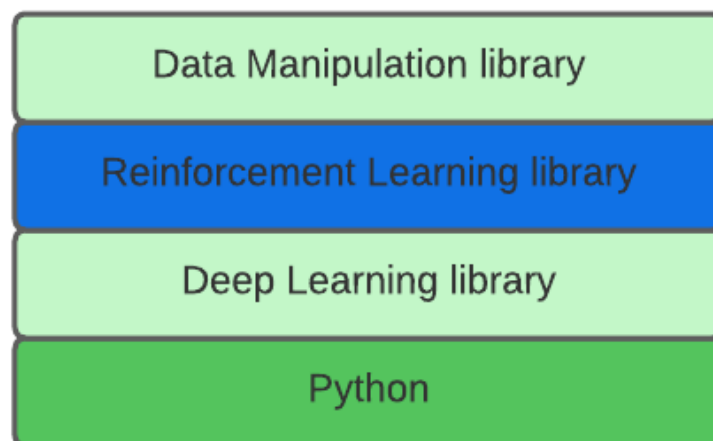


Figure 4. Necessary libraries to be installed in every container.

3.2 Software requirements

Figure 2 shows the necessary components to run a containerized application. The host operating system corresponds to the only software apart from the docker container that is needed in the system. As docker is not dependent on the operating system (OS), the selected OS will not be a limitation to the on-premises implementation. Linux would be the easiest option.

3.3 Hardware requirements

The following requirements were tested to ensure a low run time of the models. Depending on the requirements of the other software that will share this same hardware, these minimum requirements must be increased to avoid any slowdown of the system.

CPU: 2 cores, 2.3 GHz

RAM: 16 GB

Memory: up to 5 GB per Docker container.

4 Cloud Inference implementation

The cloud inference implementation allows the pilot to receive control recommendations or automatic actions from the agent in the cloud when internet connection is available. To carry out the implementation of the agent in Azure Cloud there are some components that must be defined. A computing instance is necessary to run the agent in the cloud. The OS of the computing instance and the necessary libraries and dependencies are specified in a cloud environment. The deep reinforcement learning agent must be previously trained and registered into the model section. Lastly, the endpoint to send the inferences to the pilot is created in around 10-12 minutes. The whole process is performed within the Azure Machine Learning Studio.

4.1 General Architecture

The cloud definition of the endpoint is shown in Figure 5. The Environment is defined as a Docker image and executed in a Virtual Machine (VM) or computing instance. The model is loaded from the Azure Blob Storage to the model section of Azure Machine Learning Studio. The endpoint is then ready to run. The PLC carries out the API REST call and the endpoint sends back the recommendation/action for the control system.

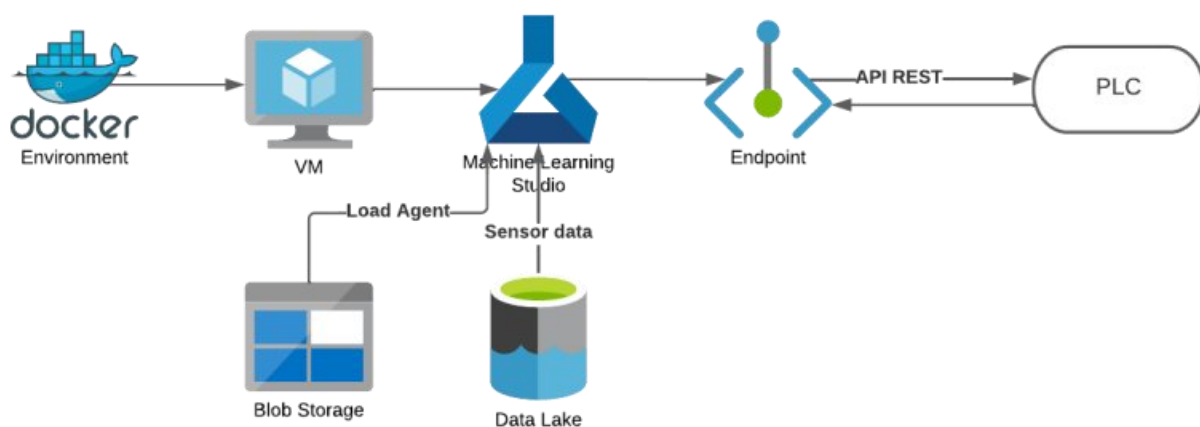


Figure 5. Cloud inference architecture diagram.

4.2 Cloud environment definition

The cloud environment must contain all the necessary libraries and dependencies for the deep reinforcement learning agent to be executed. For this purpose, a dockerfile is defined. The following figure shows the definition of a simple environment. The left column shows some details about the environment, such as the name, version, who created it and the creation date. The definition of the dockerfile is the right column, where the OS and main libraries are specified.

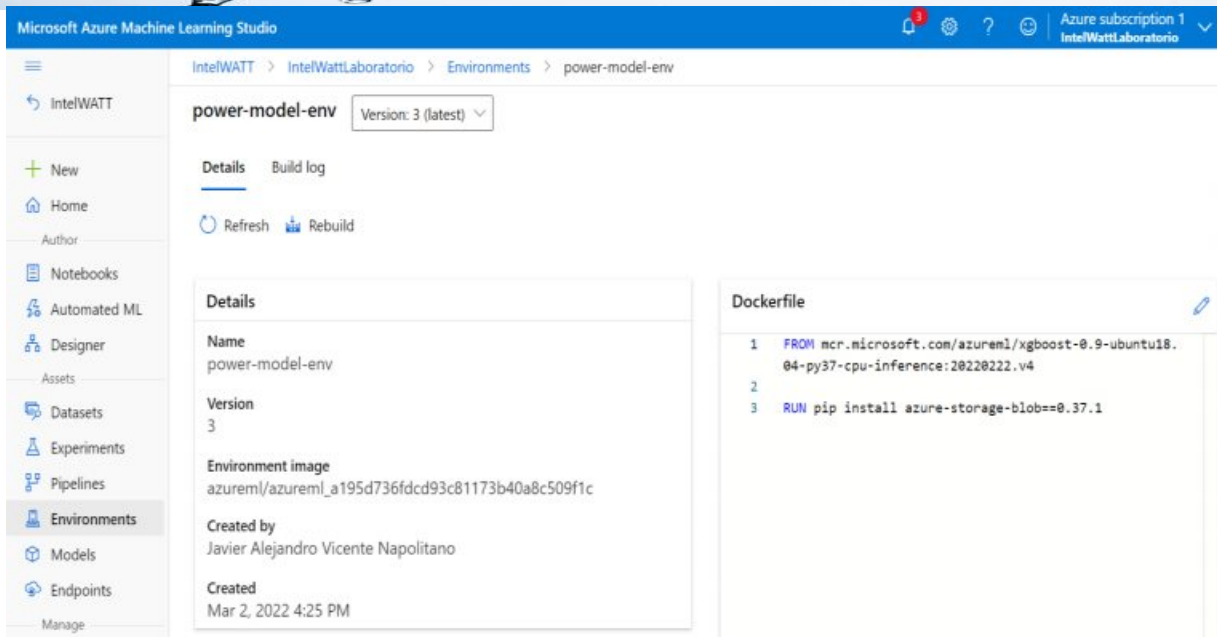


Figure 6. Example of environment definition on Azure platform.

4.3 Cloud computing instance

The previous environment can only be executed if a computing instance or virtual machine (VM) is assigned to it. Azure cloud provides several choices of computing instances, both CPU and GPU with different sizes (CPU and GPU cores) and RAM memory. The computing instance can be scaled as needed, increasing its size, or creating a cluster of computer instances to adapt to the computing power demand.

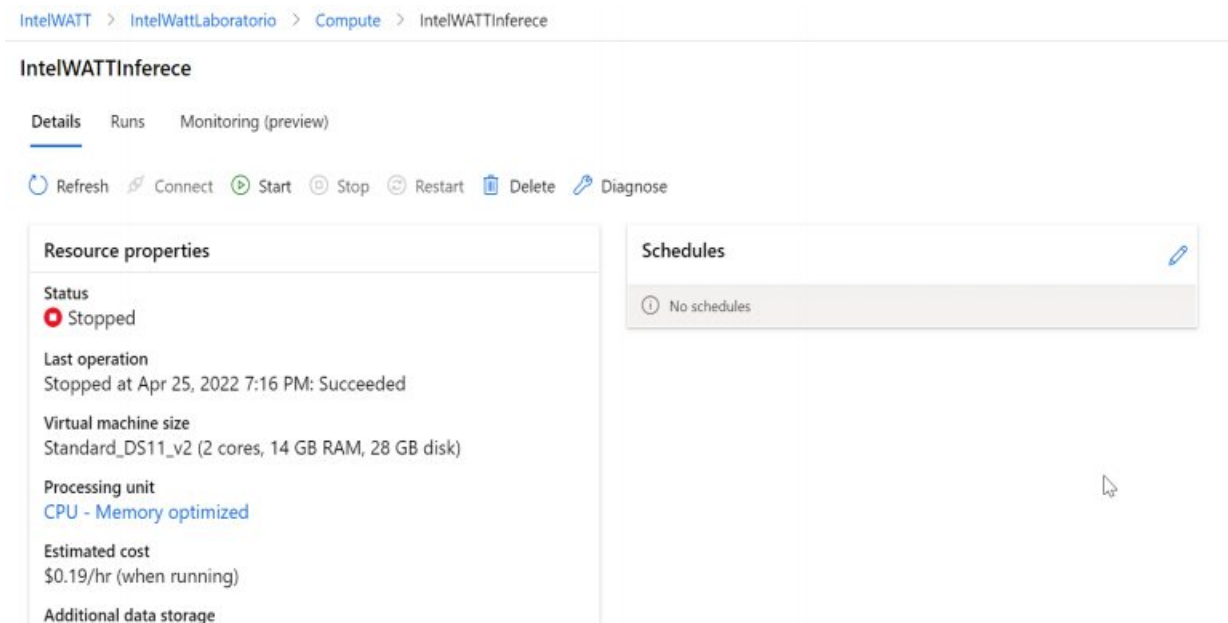


Figure 7. Computing instance example on Azure.

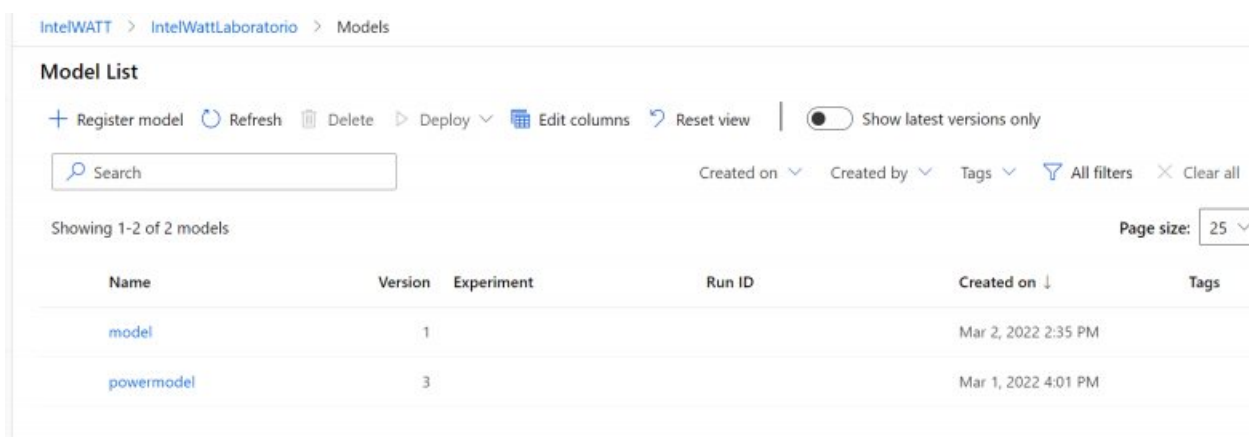


‘intelWATTinference’ VM has the resource properties on the left column. The status, the VM size, process unit, and estimated cost is some of the information shown for each new computing instance created. Any computing instance can be started and stopped whenever it is needed or not.

4.4 Deep Reinforcement Learning agent

The DLR agent is the responsible for the control and optimization of the membrane processes. After the agent training through theoretical python models and the fine tuning of the algorithm hyperparameters (parameters related to the algorithm, not the actual model), the result is the first version of the neural network responsible for the control and optimization. This neural network will be continuously trained with real pilot data, during the development of intelWATT project.

The previously described neural network or the agent, is the one that will be registered into the Azure Machine Learning Studio as the model. The model is the key part of the endpoint. Environment and computing resources are just defined to allow the model to be executed.



Name	Version	Experiment	Run ID	Created on ↓	Tags
model	1			Mar 2, 2022 2:35 PM	
powermodel	3			Mar 1, 2022 4:01 PM	

Figure 8. Model list on Azure cloud

4.5 Endpoint

The endpoint is a remote computing device that communicates back and forth with a network to which it is connected. It will allow the recommendations to be sent from the cloud to the pilot through internet. The data sent from the pilot and then stored in the data lake is the input of this endpoint. The output is the recommended action to be taken in each of the membrane process (e.g., increase/decrease flow rate in a certain amount).

After the definition of the environment, computing instance and model, the endpoint can now be deployed. The geographic zone, permissions and access keys are some of the necessary information to the endpoint to be deployed. This service can scale as demand grows. If more virtual machines (computing instances) are needed, the endpoint allows them to be created during its operation. The traffic that each of the computing instances are responsible for is balanced, so that there are no idle instances during the operation of the endpoint.



Figure 6 shows a defined endpoint in Azure platform. Service ID, a brief description about the endpoint, the provisioning state, and the compute type are displayed when any endpoint is checked out.

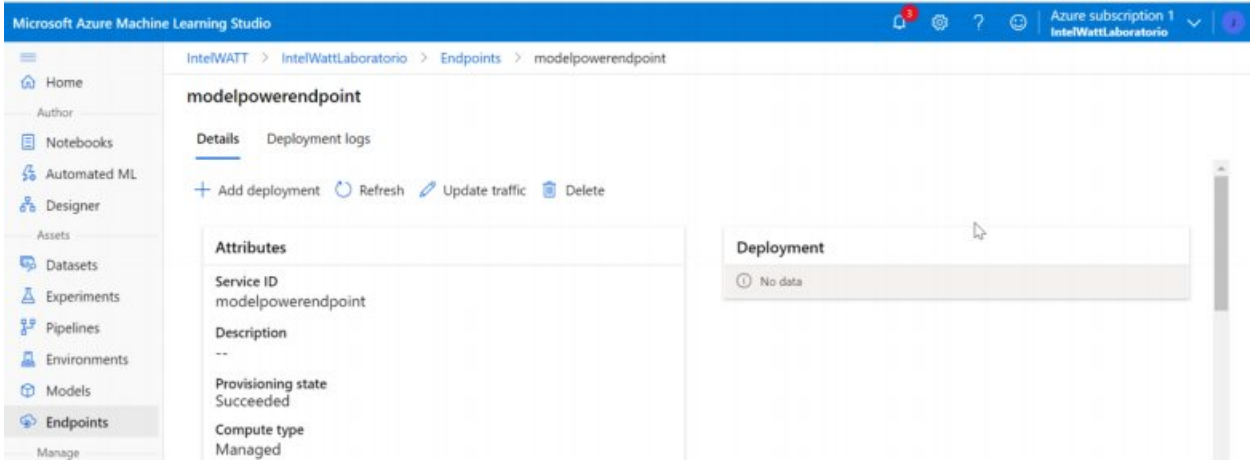


Figure 9. Example Azure Endpoint

5 Cloud Continuous training implementation

Figure 9 shows the general diagram of the continuous training implementation in Azure Cloud.

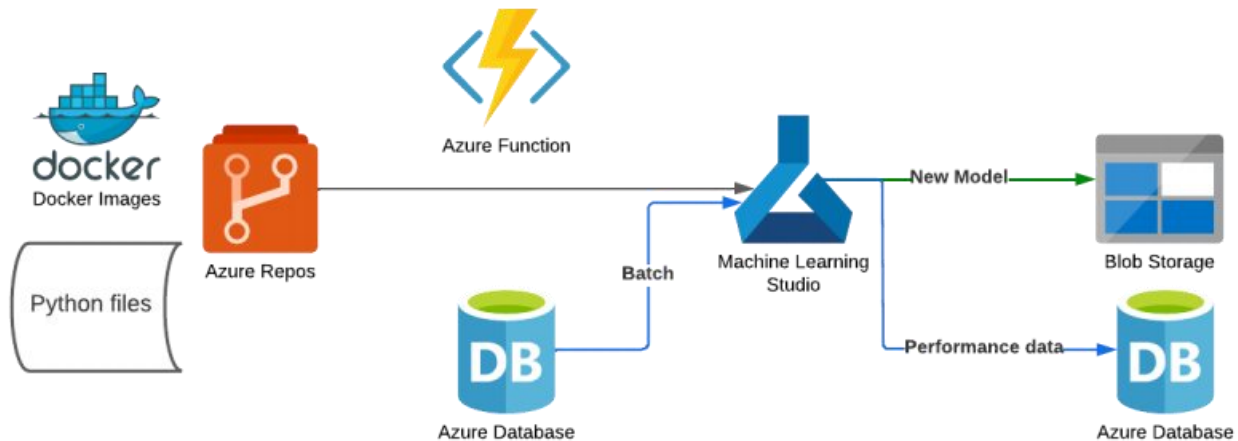


Figure 10. Continuous training implementation in Azure.

The training Python scripts, and Docker images will be in Azure Repos, a private Git repository. A Git repository tracks and saves the history of all changes made to the files in a Git project. The training process will be performed after N actions taken by the agent, that is to say, after N interactions of the agent with the real pilot system. An Azure Function will be used to count the number of interactions of the agent. It will be responsible for triggering the next training phase.

After the triggering of the training phase, Azure Machine Learning Studio sets up a VM to compute the agent parameter's update, using both the training Python script and the Docker imagen from Azure Repos. The data used to perform this updated is stored in a database. The output of the training phase is a new version of the agent.

Azure Blob storage will store every agent version. On the one hand, storing different agent versions allow an easy way to compare the performance of agents. Although the comparison between agents is not straightforward and the best way for doing it is still not defined, data about the agent's performance will be stored in a database. On the other hand, in case any update leads to a worsening in the agent's performance, or any problem occurs during training, an old version of the agent can be reused.

6 References

- [1] Docker website, <https://www.docker.com/>
- [2] Azure Cloud, <https://azure.microsoft.com/en-us/>

