UNIVERSITÀ DEGLI STUDI FIRENZE

DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB DISTRIBUTED DATA INTELLIGENCE AND TECHNOLOGIES LAB

SNAP4CITY KM4CITY

Corso: Big Data Architecture

Machine Learning OPerationS with ClearML @SNAP4City

Roadmap

1. Main advantages of MLOPS
2. Adapting ClearML to Snap4City Architecture
3. Basic concepts of ClearML
4. Getting started demo for ClearML at Snap4City

Why MLOPS?

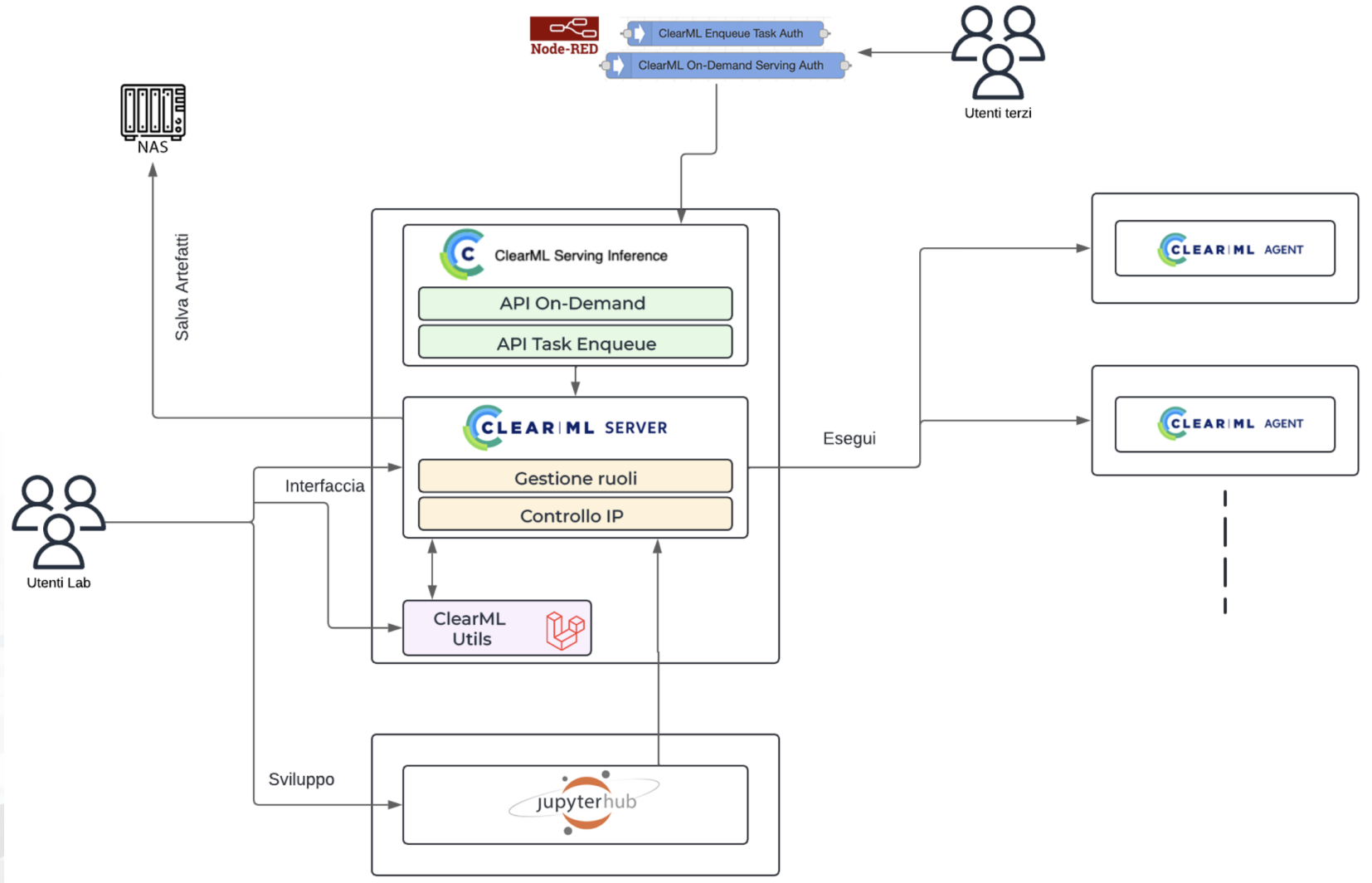
Main advantages in **orchestrating** a **cluster of GPUs**:

- Workload **distribution**
- **Parallel** execution
- Avoiding bottlenecks guarantees execution **speed**

Section I

CLEARML ARCHITECTURE AT SNAP4CITY

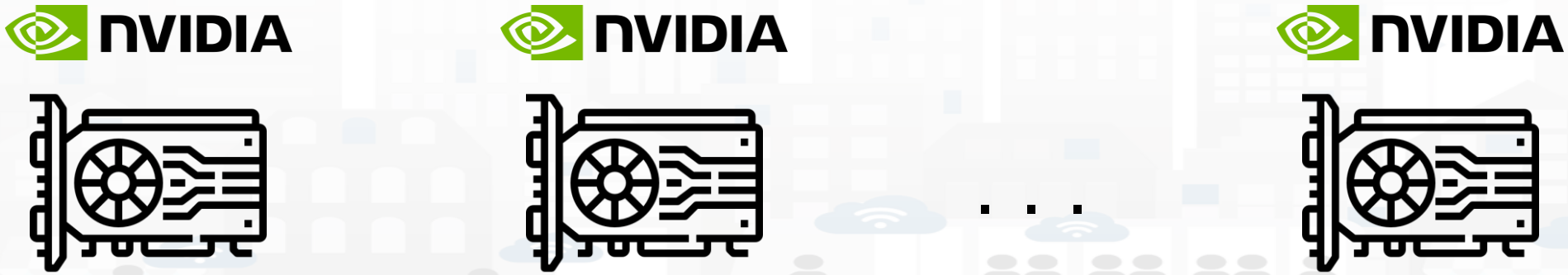
ClearML - Snap4City Architecture



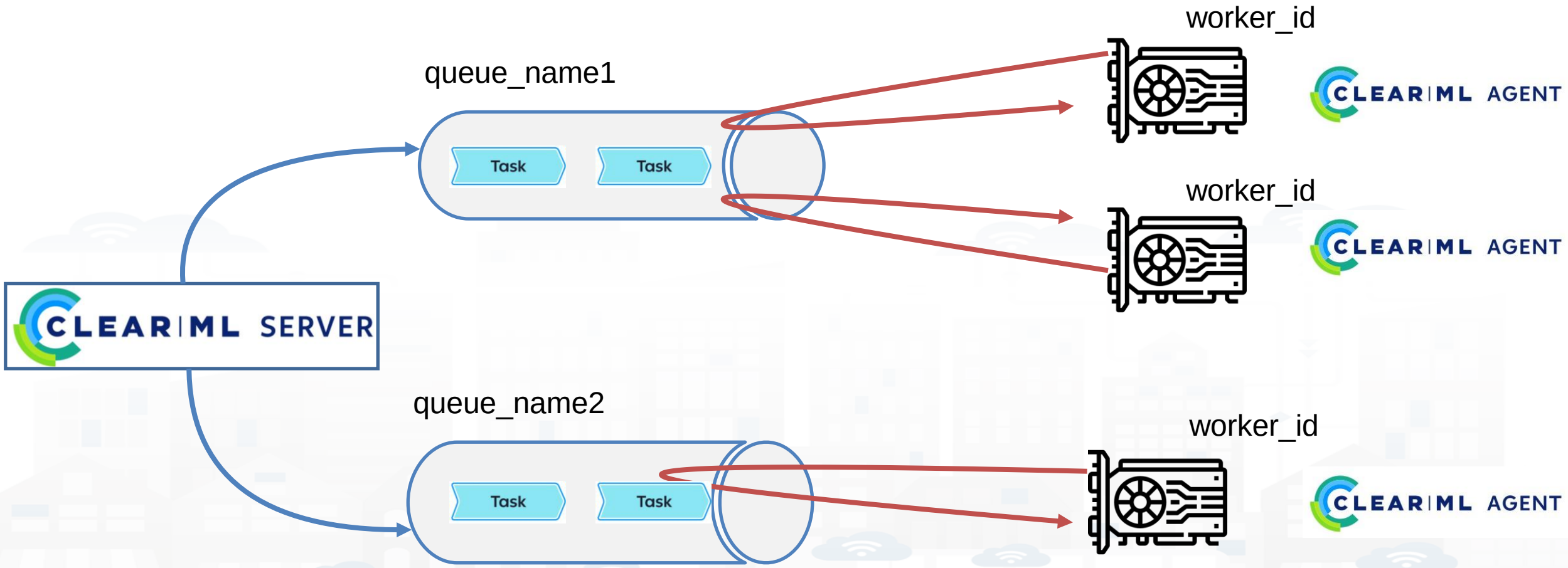
ClearML Agent

ClearML Server is used in order to orchestrate the execution of tasks in the **Snap4City** **GPU** **cluster**.

On each of those machines an instance of **ClearML Agent** is installed. This allows the execution of tasks and ensures workload distribution.



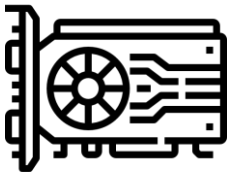
Workers & Queues



Agent execution

queue_name

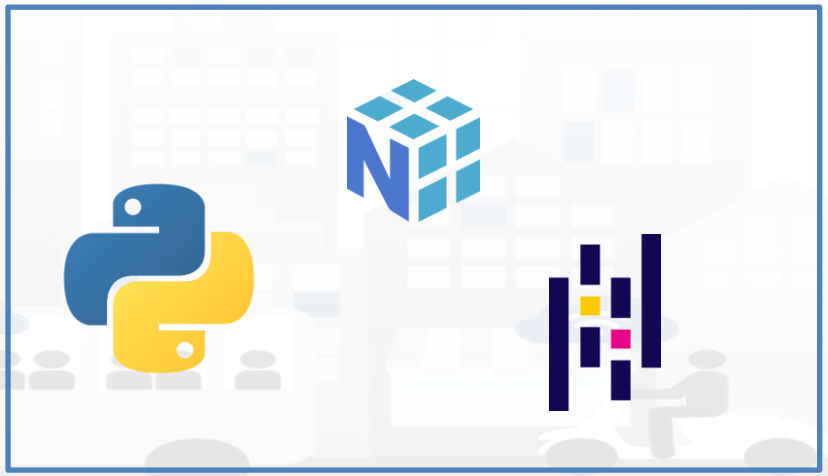
worker_id



CLEARML AGENT

CLEARML SERVER

docker



Roles definition and security management

By default **ClearML** allows every user with credentials to create tasks from all devices.

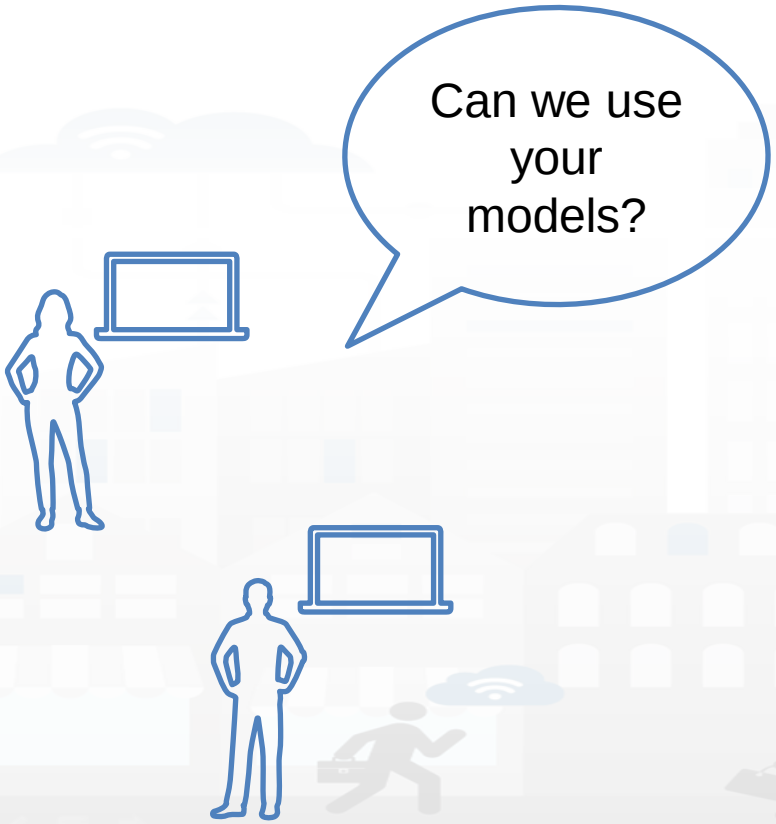
ClearML for **Snap4City** has been designed in order to guarantee that only verified users can access to the GPU cluster. IP verification via ClearML Server implemented.

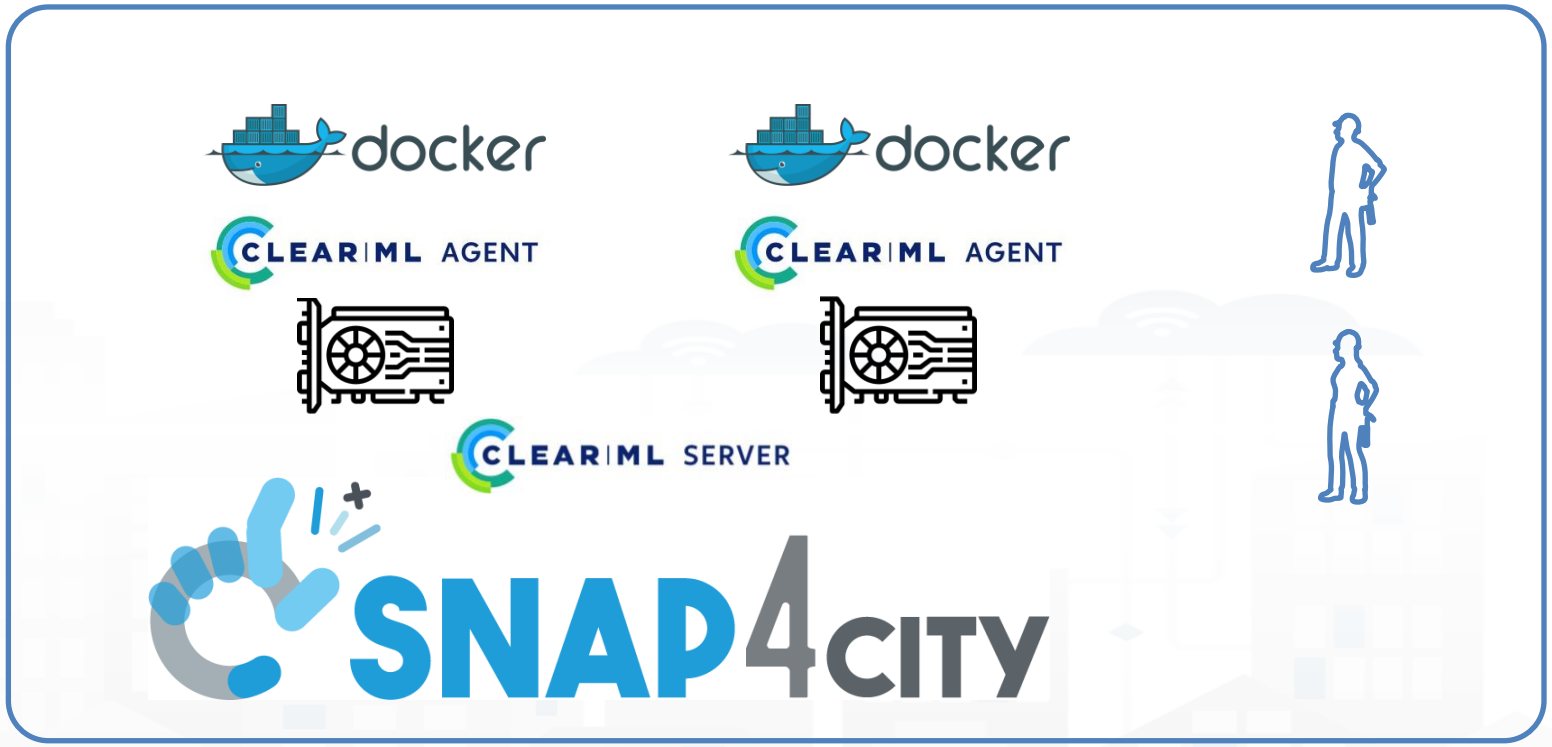
ClearML allows every user to visualize every project and visualize queues.

Defined the **Admin** role in order to differentiate users with privileges for monitoring purpose and guarantee privacy of common users.

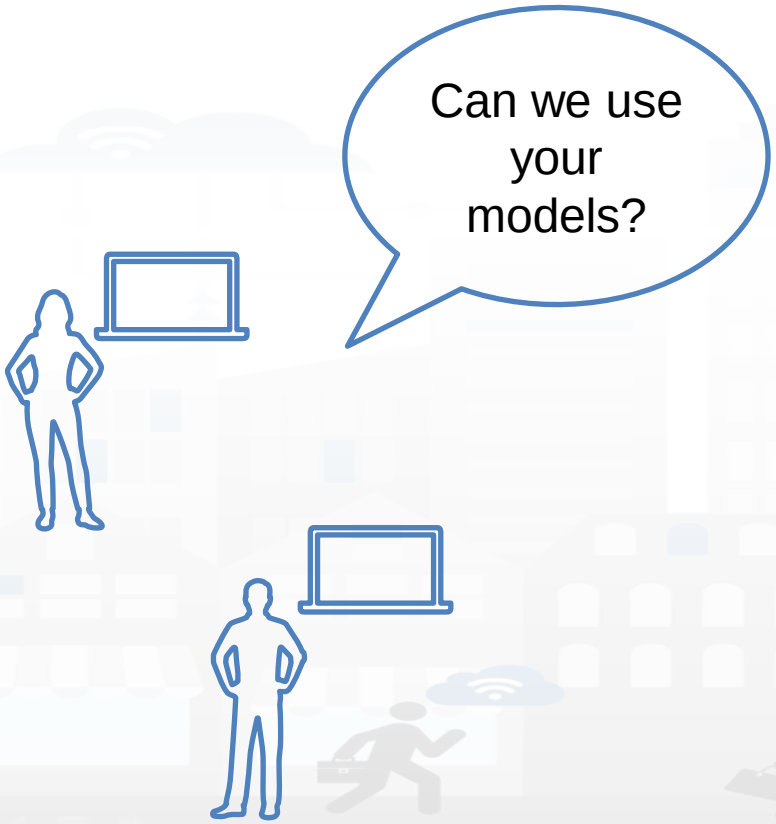
A live **notification system** has been designed to ensure the monitoring of tasks execution and the **resources availability.**

What about inference?

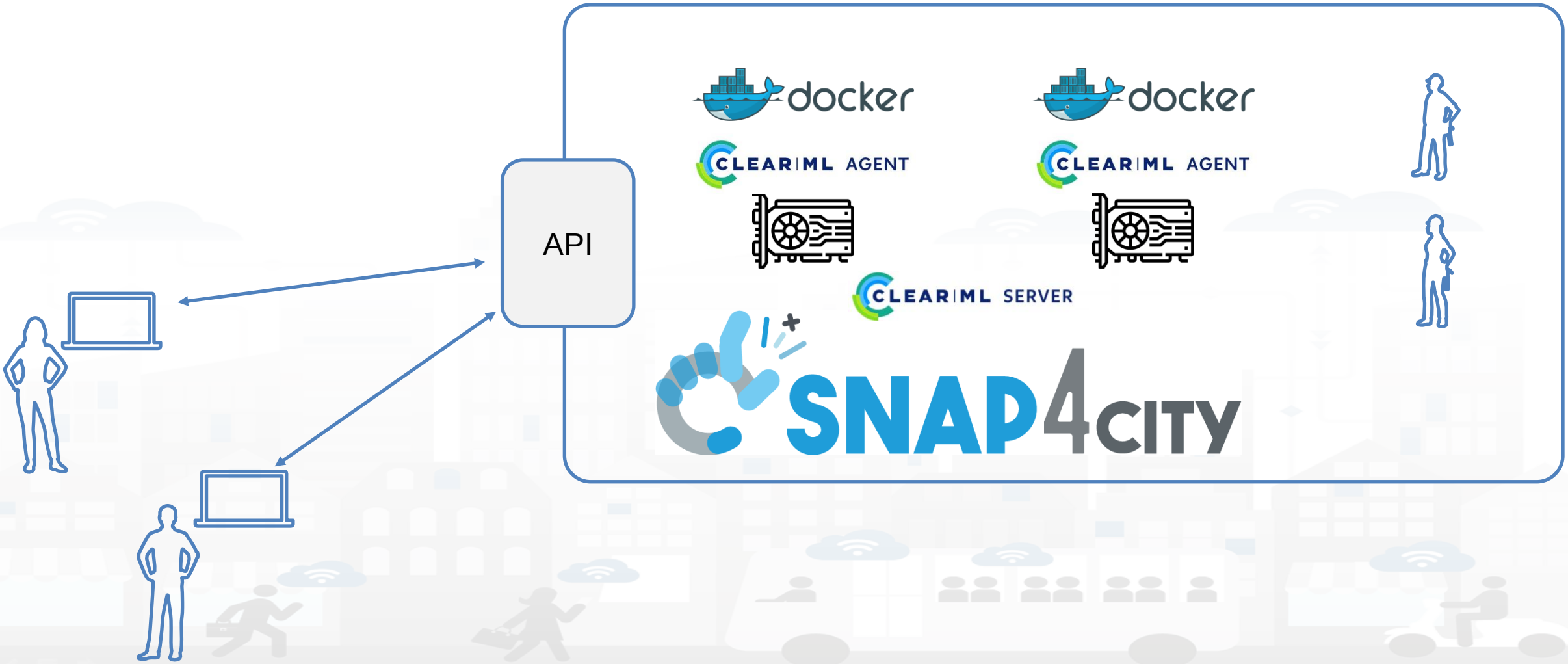




What about inference?



What about inference?



Providing inference services

1. **API Serving On Demand:** real time inference on **pre-trained models**.

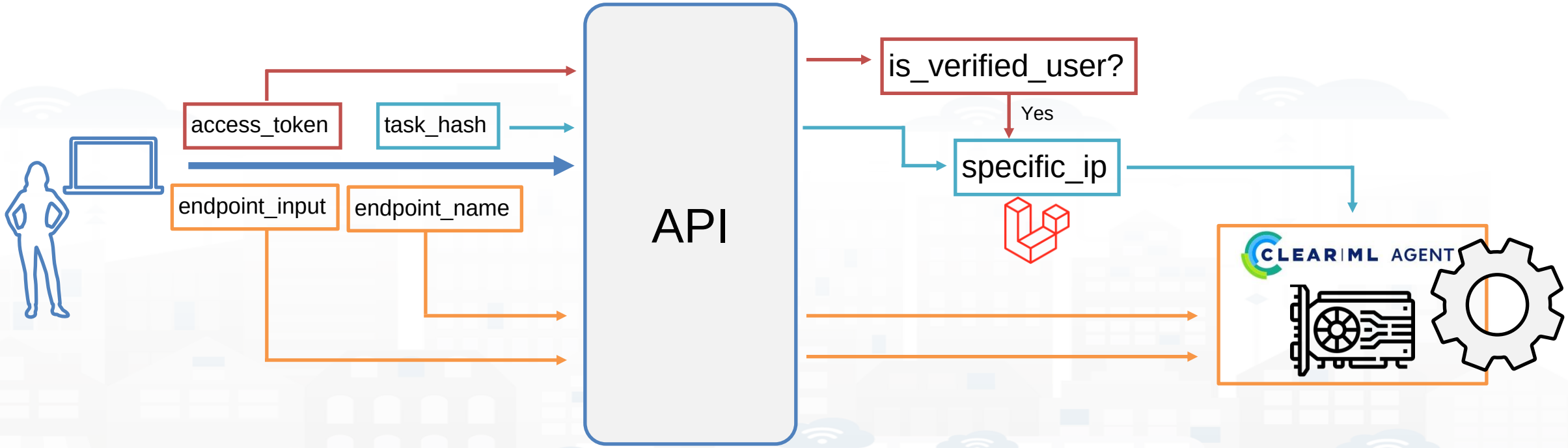
2. **API Task Enqueue:** allowing user to send **task to queues** without knowing the architecture of the system.

Node-RED

ClearML Utils

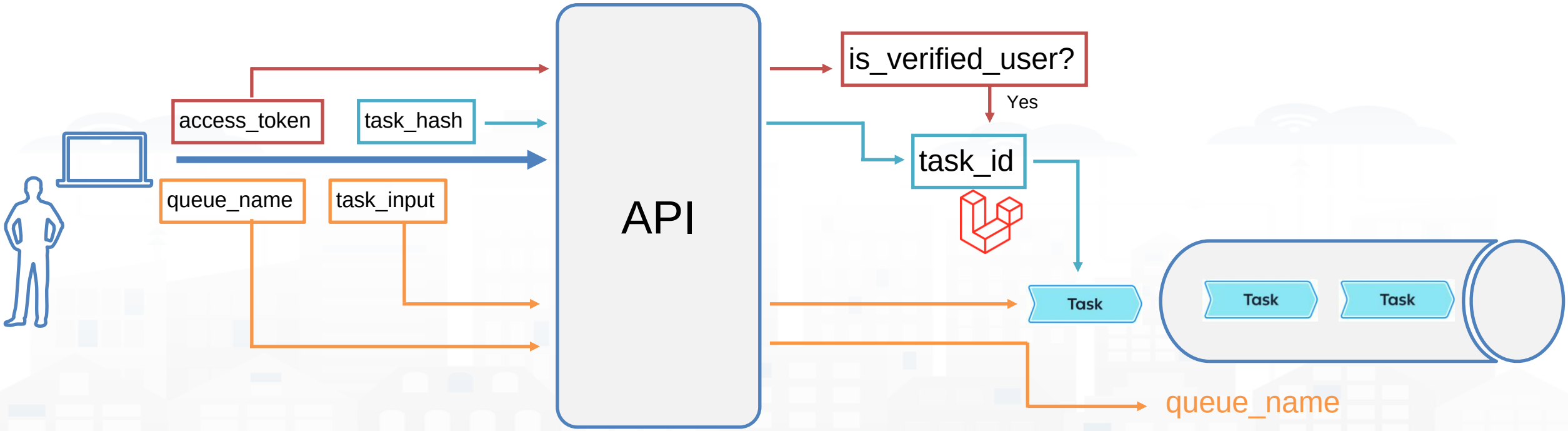
API On Demand

Real time inference on **pre-trained** models



API Task Enqueue

Send task to **queues**



ClearML Utils: What and How

Developed for internal use only:

- **Calls routing**
- **resources handling** for API serving and task enqueue
- **monitoring** calls to services and log tracking/visualization

ClearML Utils

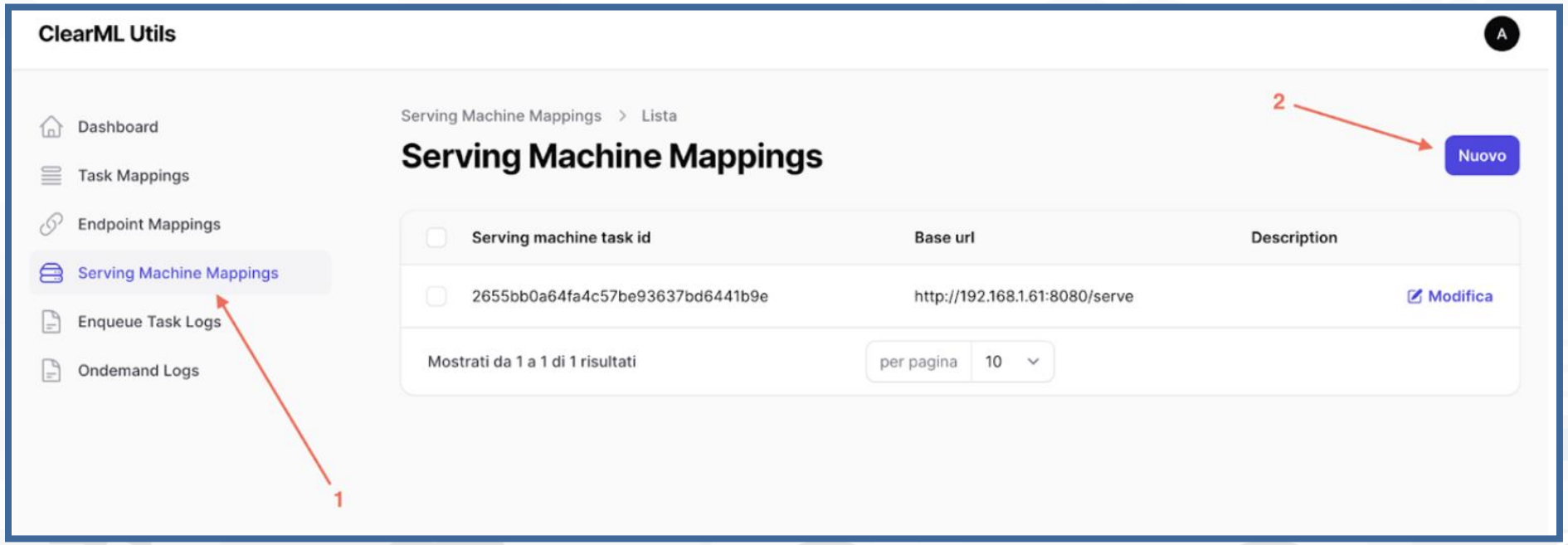
ClearML Utils: Dashboard

Shows **available resources** and **current usages** of services:

- **Number** of **machines** configured for **on-demand serving**
- **Number** of **endpoints** for **on-demand serving**
- **Number** of **available tasks** for **task enqueue**
- Visualization tool for **monitoring the requests trend** on both on-demand and task enqueue

ClearML Utils: Serving

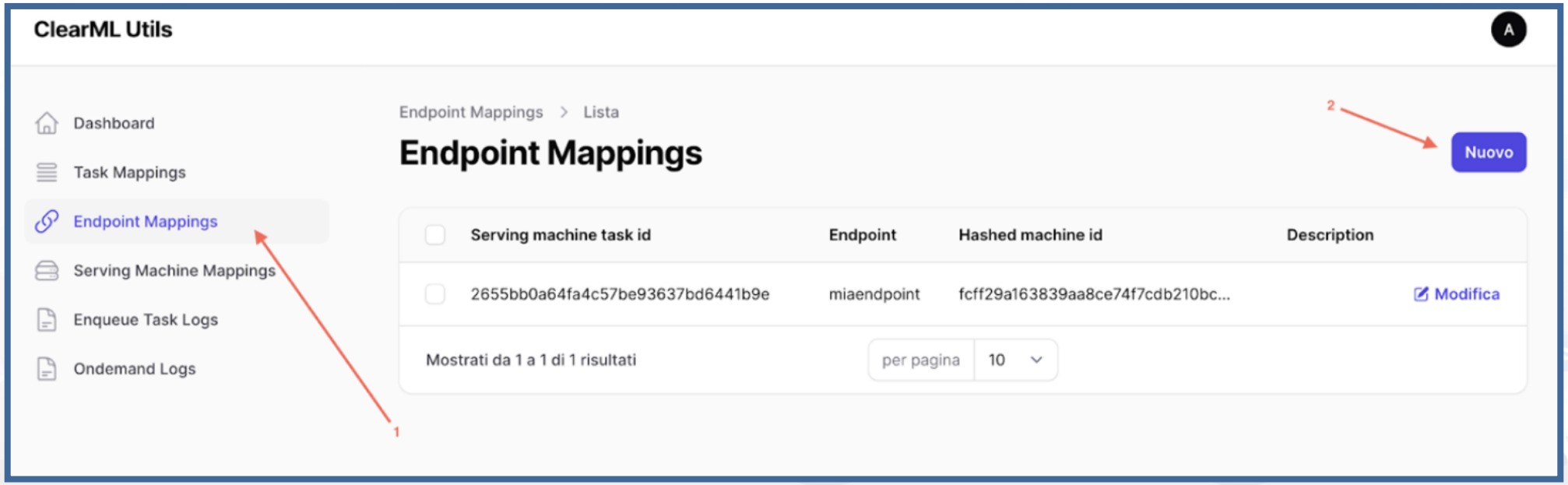
Machine task id is used in order to **map** the specific **machine ip** with the **serving task**. This is done in order to ensure that **the right model is called during serving**.



ClearML Utils: On demand endpoints

Developers create **endpoints** using ClearML Serving then:

1. Define the **endpoint name** and the associated **task_id**
2. An **hashing** of the **serving task is generated**, external users don't need to know the real task_id

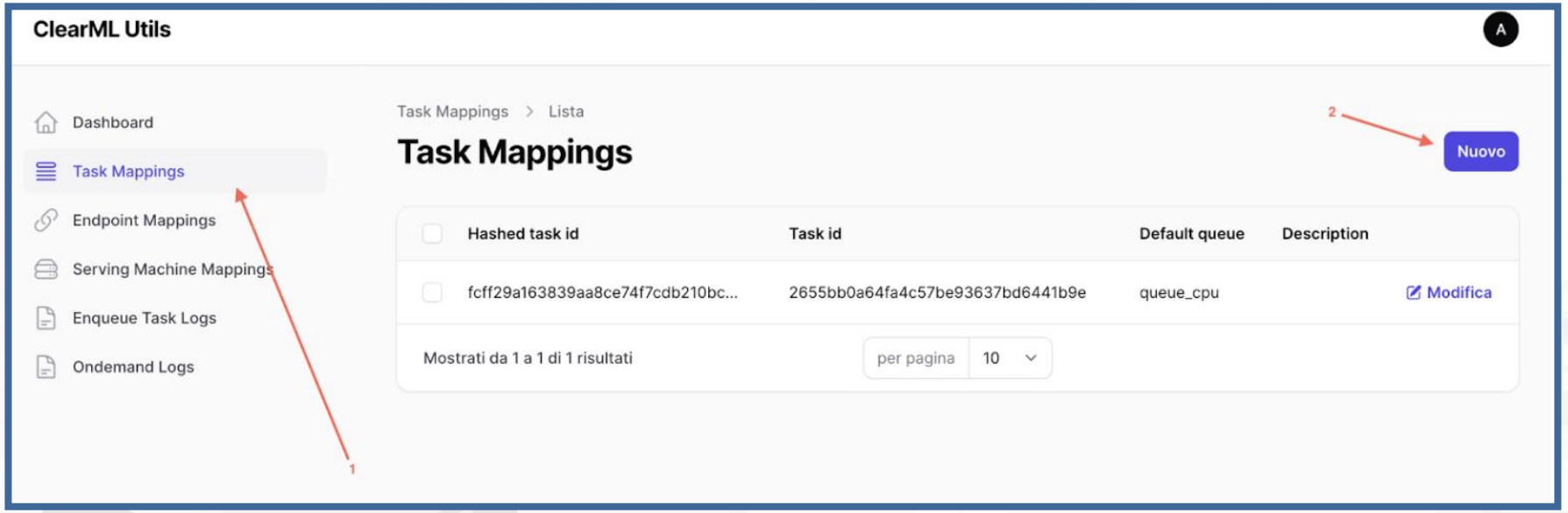


ClearML Utils: Task enqueue

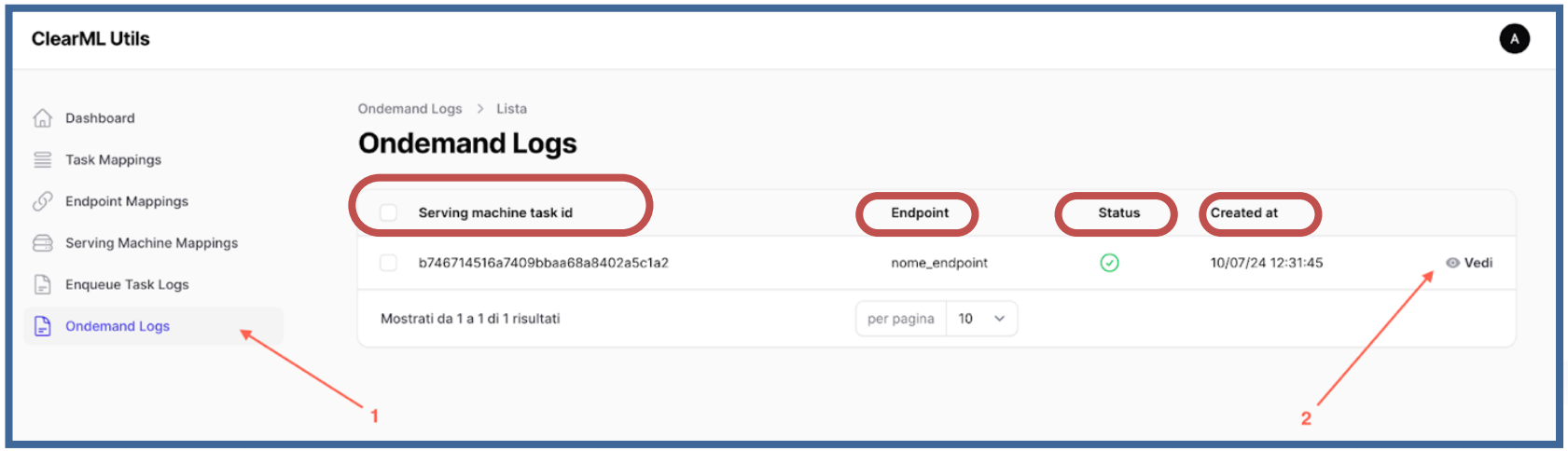
Task enqueue need to clone a task and send it to the right queue.

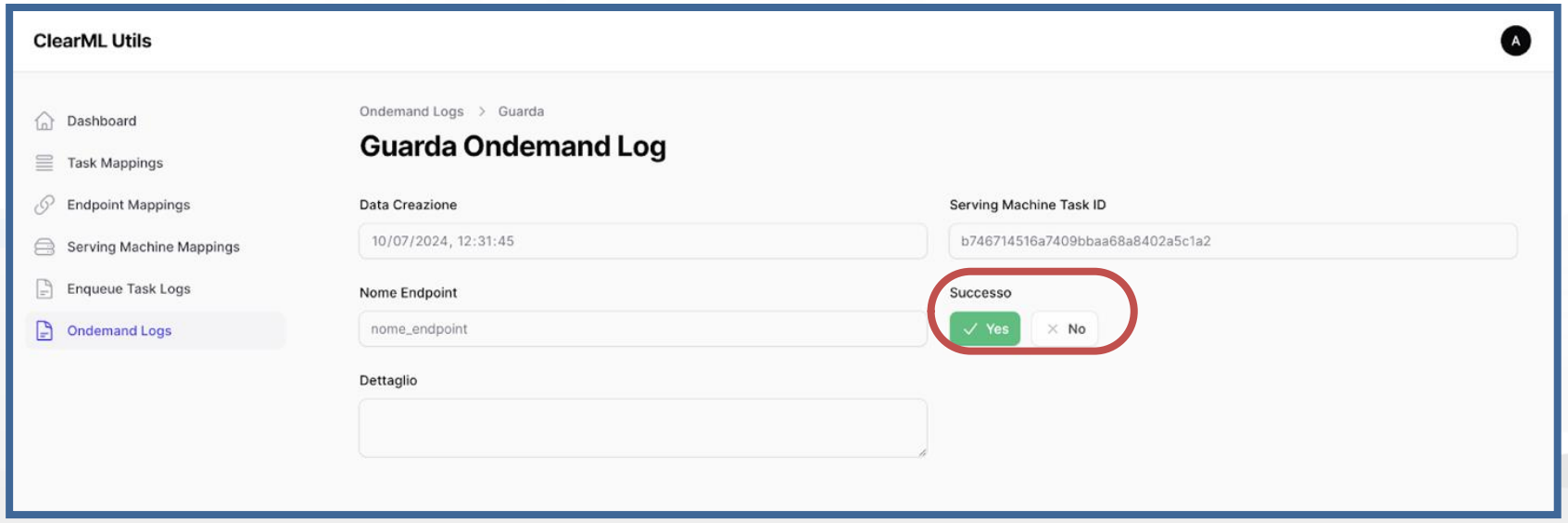
Developers:

1. Insert **task id** and the **queue name**
2. App generates a **task hash id** that allows the user to **enqueue a task**

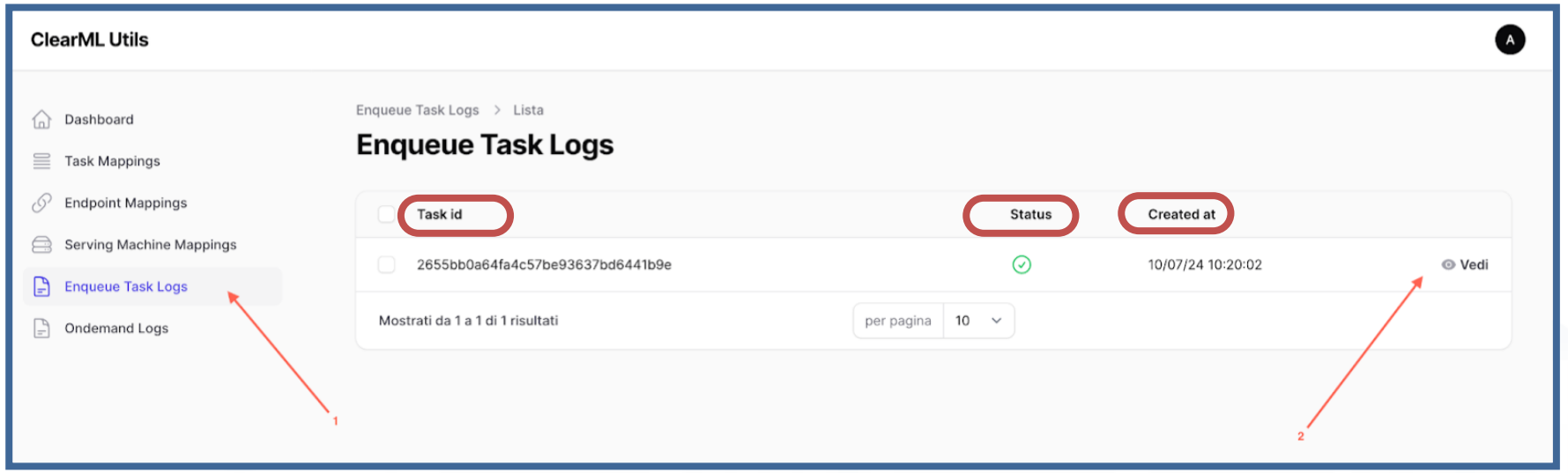


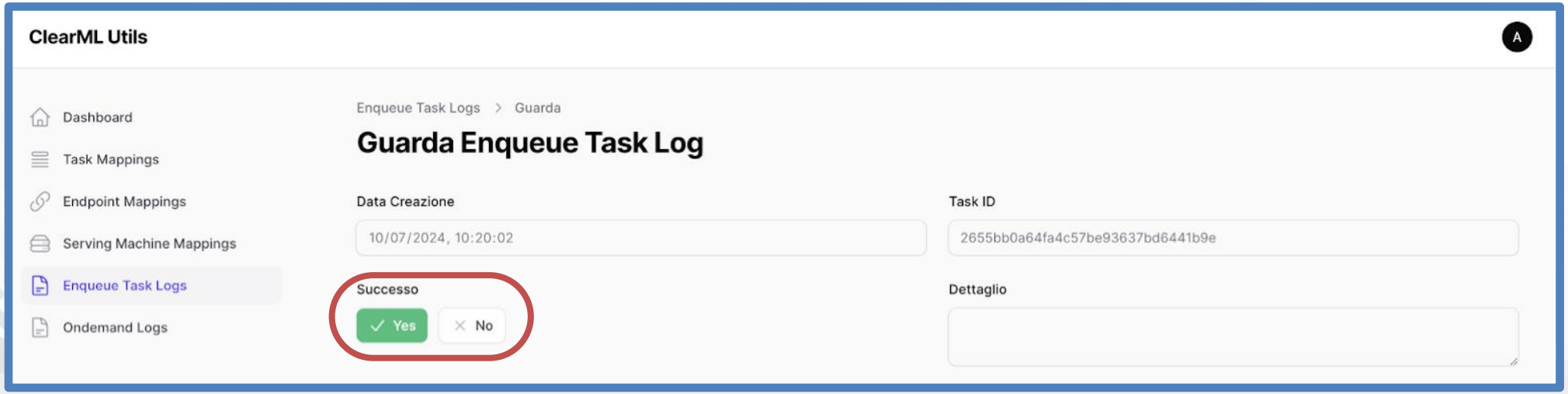
ClearML Utils: On Demand Logging





ClearML Utils: Task Enqueue Logging





Node-RED Microservices

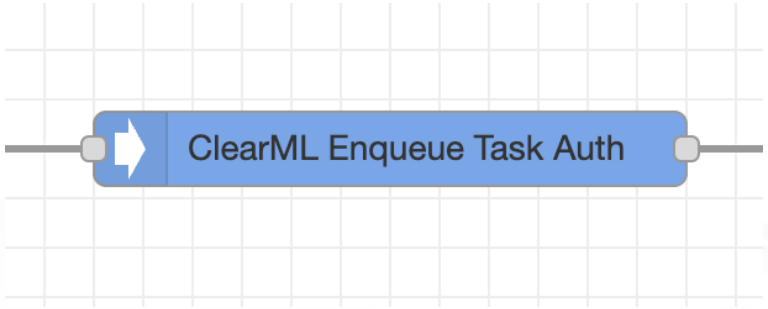
Programming **microservices** using **visual programming** with node.js

- **Task Enqueue block**: sending a task to the queue using json
- **On Demand serving block**: call immediate inference using json

Node-RED

Node-RED: Task Enqueue block

```
{
   “task_id”: “hash_task_id”,
   “queue_name”: “queue”,
   “input”: {
         “param1”: “value_1”,
         “param2”: “value_2”
         }
}
```



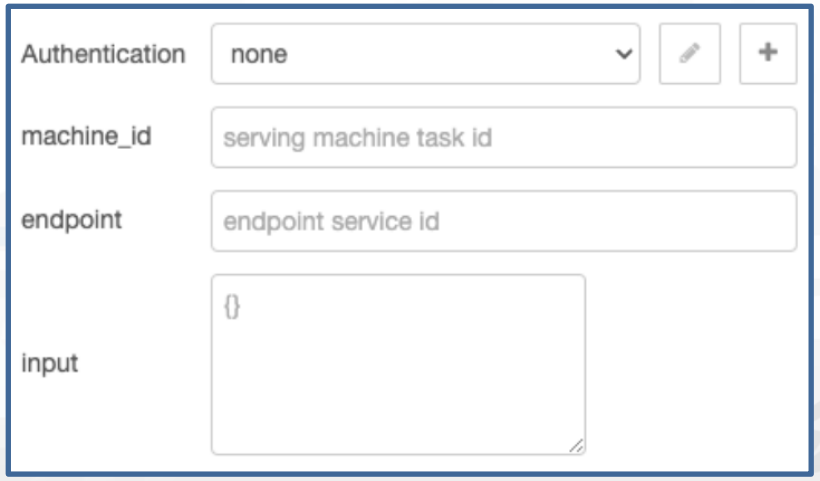

```
{
         “status”: “ok”
}
```

Robust Authentication implemented using authentication_token requested by the block and provided by Snap4City

Node-RED: On Demand block

```
{
   "machine_id": "serving_hash",
   "endpoint": "endpoint_name",
   "input": {
         "param1": "value_1",
         "param2": "value_2"
         }
}
```


```
{
         "status": "ok"
}
```



Robust Authentication implemented using authentication_token requested by the block and provided by Snap4City

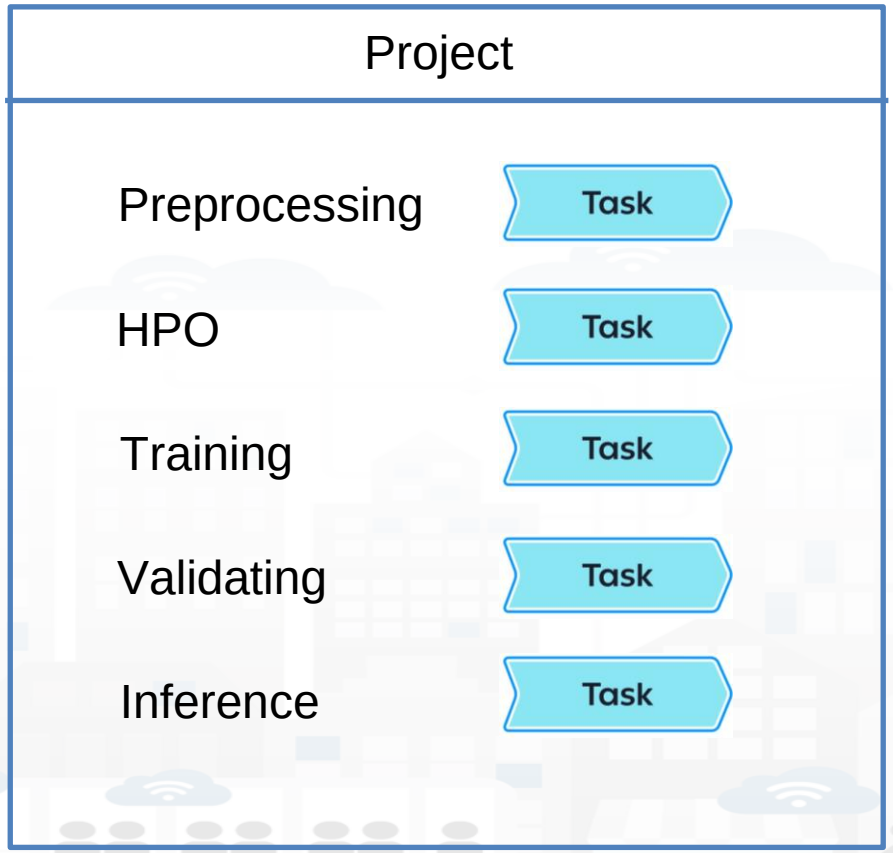
Section II

CLEARML FUNDAMENTAL CONCEPTS

Tasks and Projects

We refer to **Task** as the core ClearML component. Can be viewed as an **experiment instance**.

It is composed of all that code that we are interested in tracking (a training phase for example). Must be **uniquely identified** and all its code must be recoverable and reproducible.



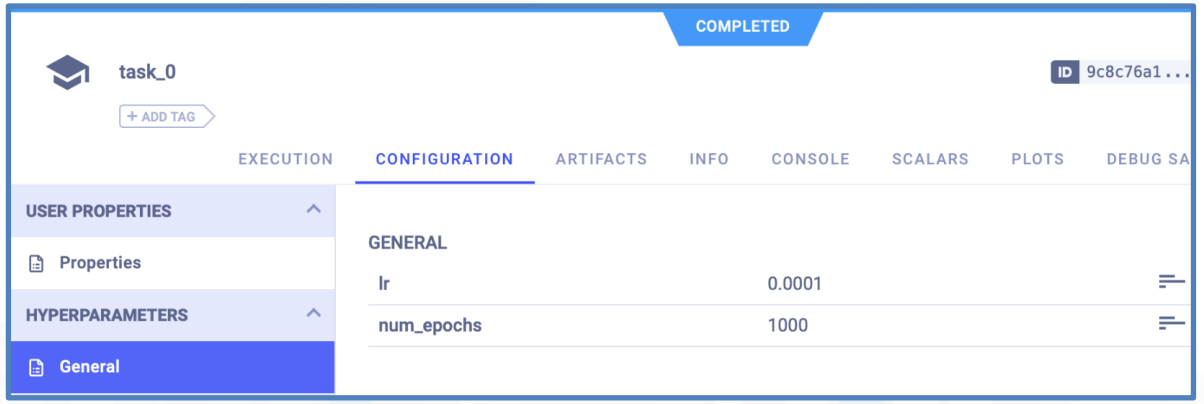
Hyperparameters

ClearML provides **Hyperparameters** handling.

Hyperparams are viewed as experiment (so **Task**) **configurations**.

The user must be able to run a task on a starting set of hyperparameters values, check all the results (so Artifacts) associated with those values.

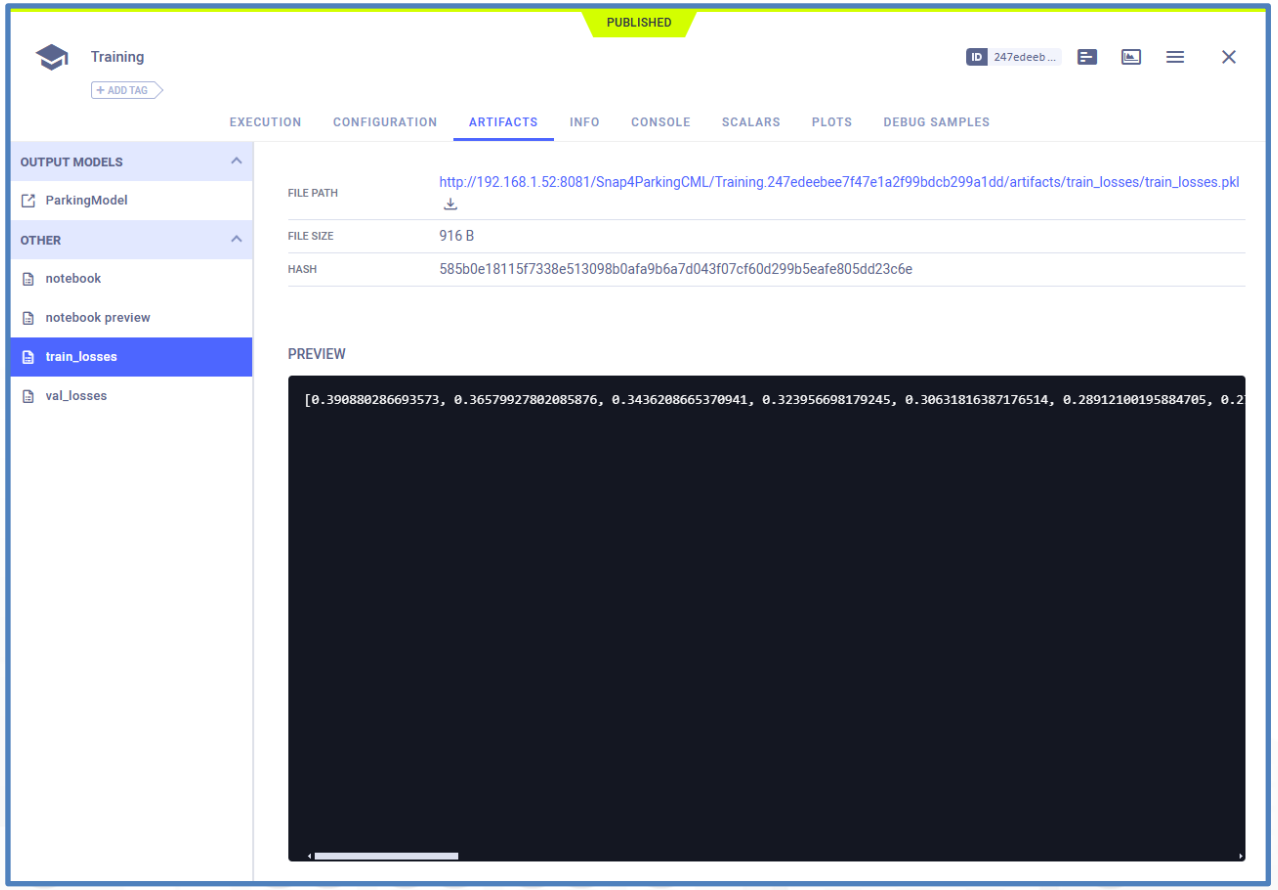
```
hyperparams = {'num_epochs': 1000, 'lr': 0.0001}
task.connect(hyperparams)
```



Artifacts

Code reproducibility doesn't relies only on recoverable lines of code (it's not code versioning). Reproducibility means that ClearML must guarantee to the user the ability to **store**, **visualizing**, **recovering** and eventually **manipulate** everything the **code** **produces**.

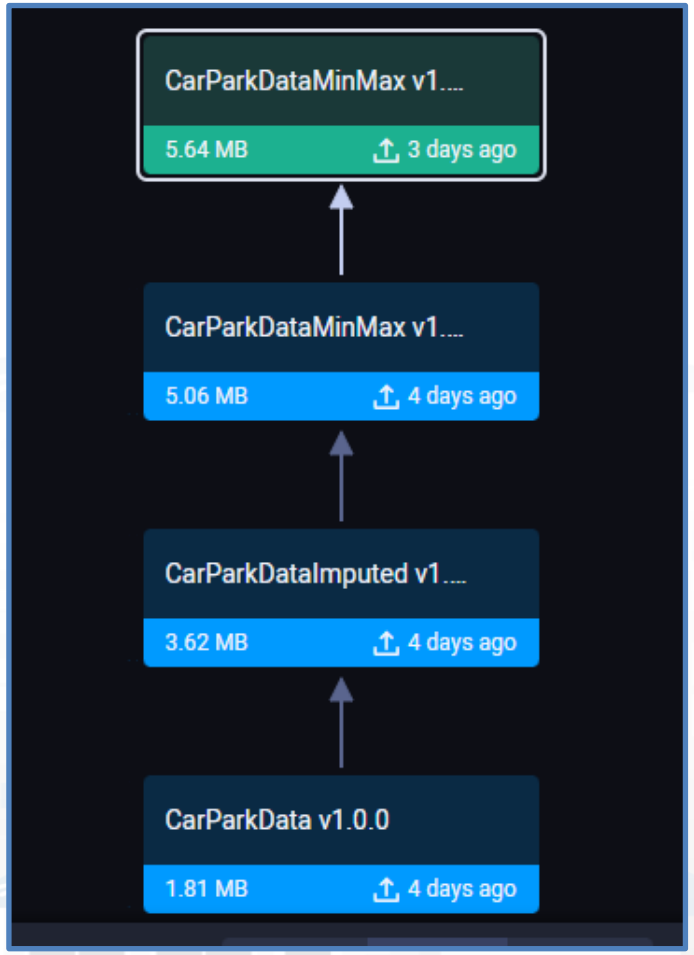
Artifacts are basically the outputs of a Task.



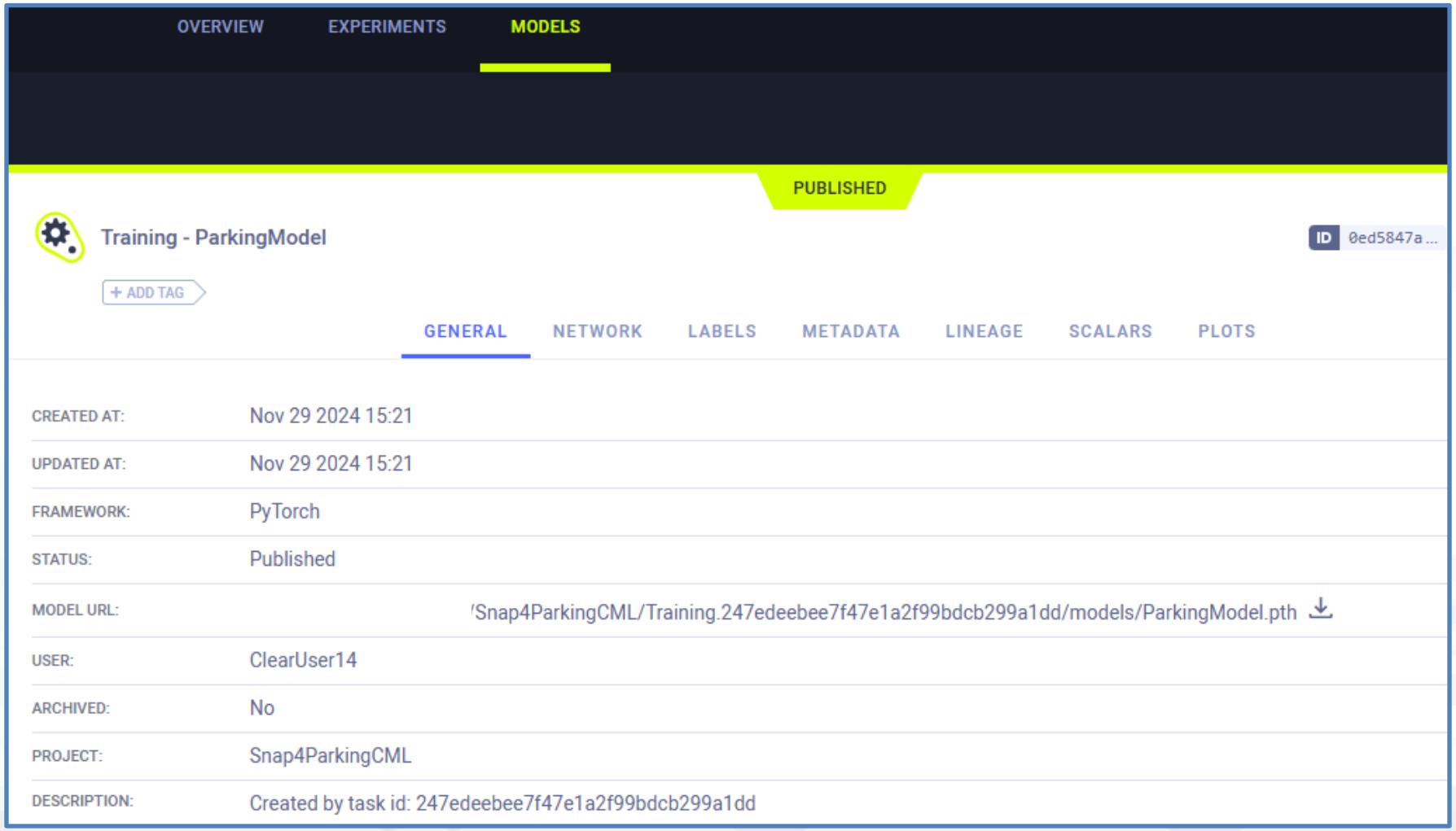
Dataset versioning

ClearML allows developers to **create**, **update**, **upload** and **retrieve datasets**.

Developers can create **different versions** of the same dataset with incremental approach while **keeping track** of every update.



Model deployment



Section III

A GETTING STARTED DEMO

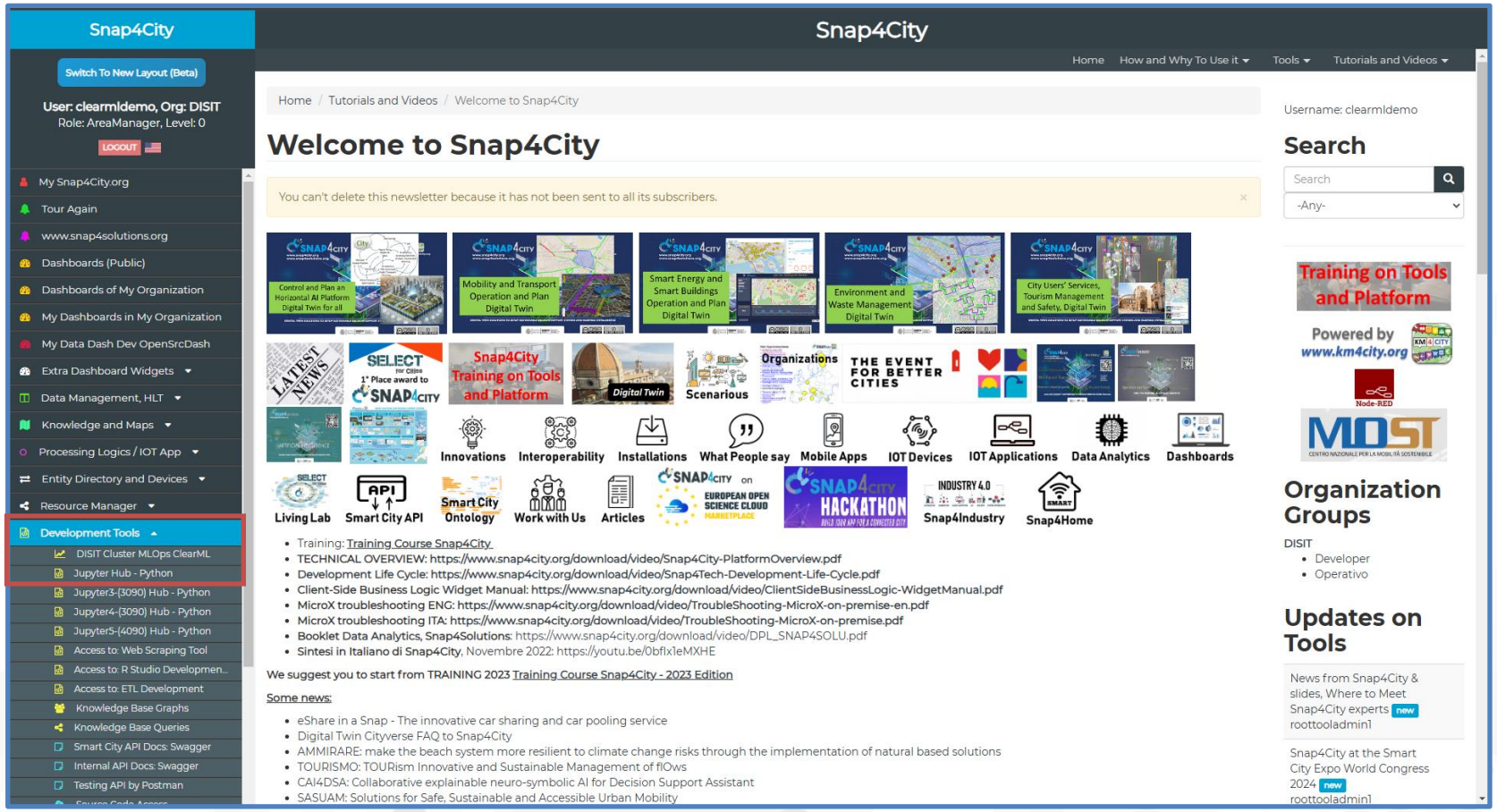
UNIVERSITÀ DEGLI STUDI FIRENZE

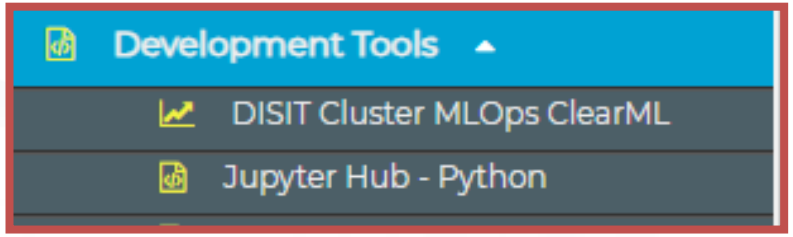
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB DISTRIBUTED DATA INTELLIGENCE AND TECHNOLOGIES LAB

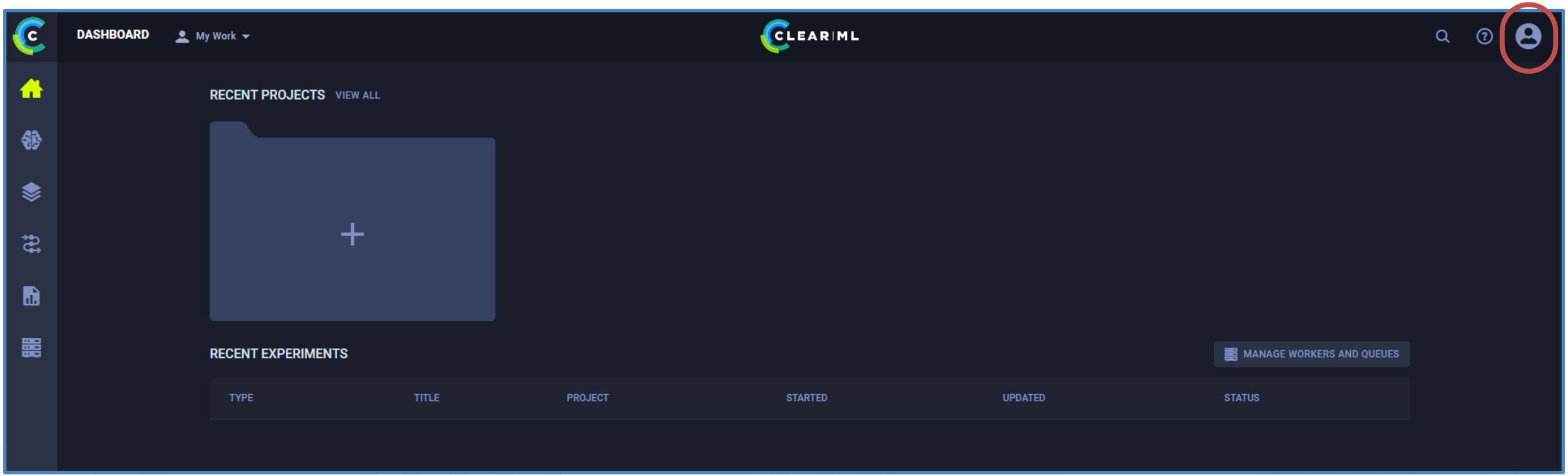
SNAP4CITY KM4CITY

Authentication with Snap4City

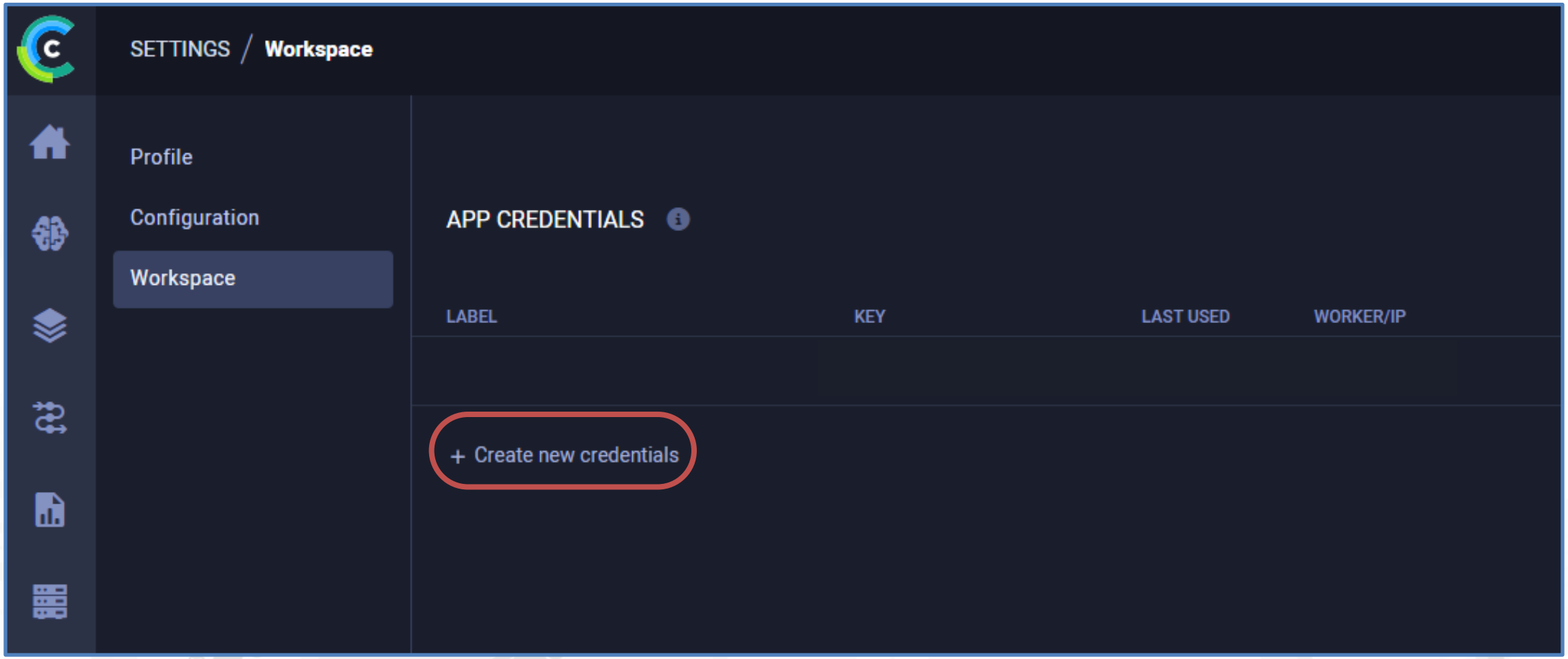




ClearML Server account



Getting API keys



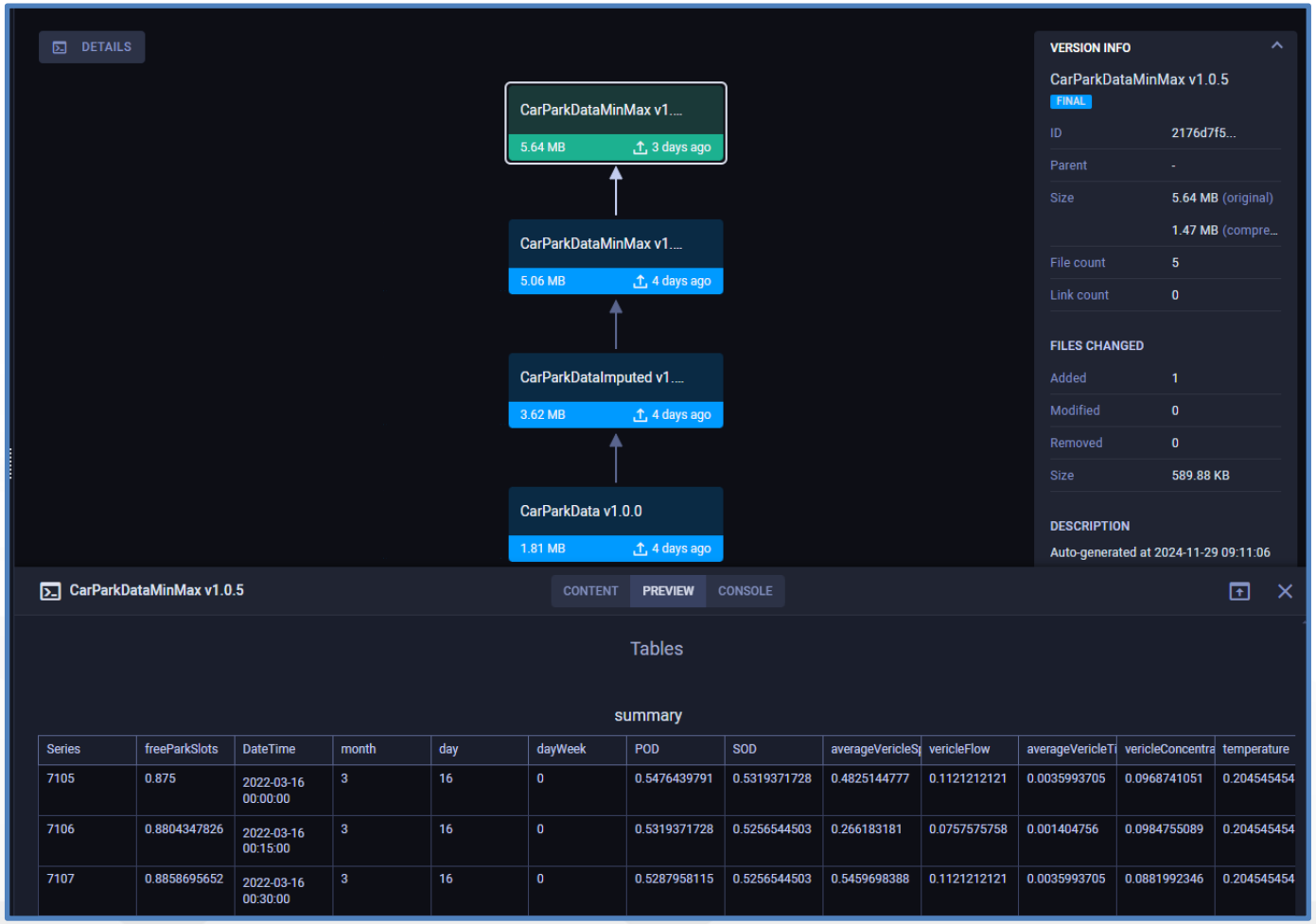
Data uploading

```
dataset = Dataset.create(dataset_project = <YOUR_PROJECT>, dataset_name = <DATASET_NAME>)

dataset.add_files(path = <DATA_PATH>)

dataset.upload()

dataset.finalize()
```



Series	freeParkSlots	DateTime	month	day	dayWeek	POD	SOD	averageVericleSp	vericleFlow	averageVericleTi	vericleConcentra	temperature
7105	0.875	2022-03-16 00:00:00	3	16	0	0.5476439791	0.5319371728	0.4825144777	0.1121212121	0.0035993705	0.0968741051	0.204545454
7106	0.8804347826	2022-03-16 00:15:00	3	16	0	0.5319371728	0.5256544503	0.266183181	0.0757575758	0.001404756	0.0984755089	0.204545454
7107	0.8858695652	2022-03-16 00:30:00	3	16	0	0.5287958115	0.5256544503	0.5459698388	0.1121212121	0.0035993705	0.0881992346	0.204545454

Jupyter notebook: setup

```
import torch
import os
import pandas as pd
import numpy as np
import torch.nn as nn
import torch.optim as optim
import json
import matplotlib.pyplot as plt
from clearml import Task, Logger, Dataset, OutputModel, StorageManager
from enum import Enum
from torch.utils.data import DataLoader, TensorDataset
from typing import Tuple

os.environ['CLEARML_WEB_HOST'] = "http://
os.environ['CLEARML_API_HOST'] = "http://
os.environ['CLEARML_FILES_HOST'] = "http://
os.environ['CLEARML_API_ACCESS_KEY'] =
os.environ['CLEARML_API_SECRET_KEY'] =

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

Task.add_requirements("numpy", "1.23.5")
Task.add_requirements("pandas", "1.3.3")
Task.add_requirements("accelerate", "0.31.0")
Task.add_requirements("bitsandbytes", "0.43.1")

task = Task.init(project_name = "Snap4ParkingCML", task_name="Training", output_uri = True)
output_model = OutputModel(task = task, framework = "PyTorch")
#output_model.set_upload_destination("models/ParkingModel/")
task.set_base_docker("                                    ")
task.execute_remotely(queue_name=                        )
#StorageManager.upload_folder("./models", '/models/')
hyperparams = dict()
criterion = nn.MSELoss()
hyperparams['num_epochs'] = 100
task.connect(hyperparams)
```

Jupyter notebook: retrieving data

```
dataset = Dataset.get(dataset_id=                                    )
dataset_folder = dataset.get_local_copy()
```

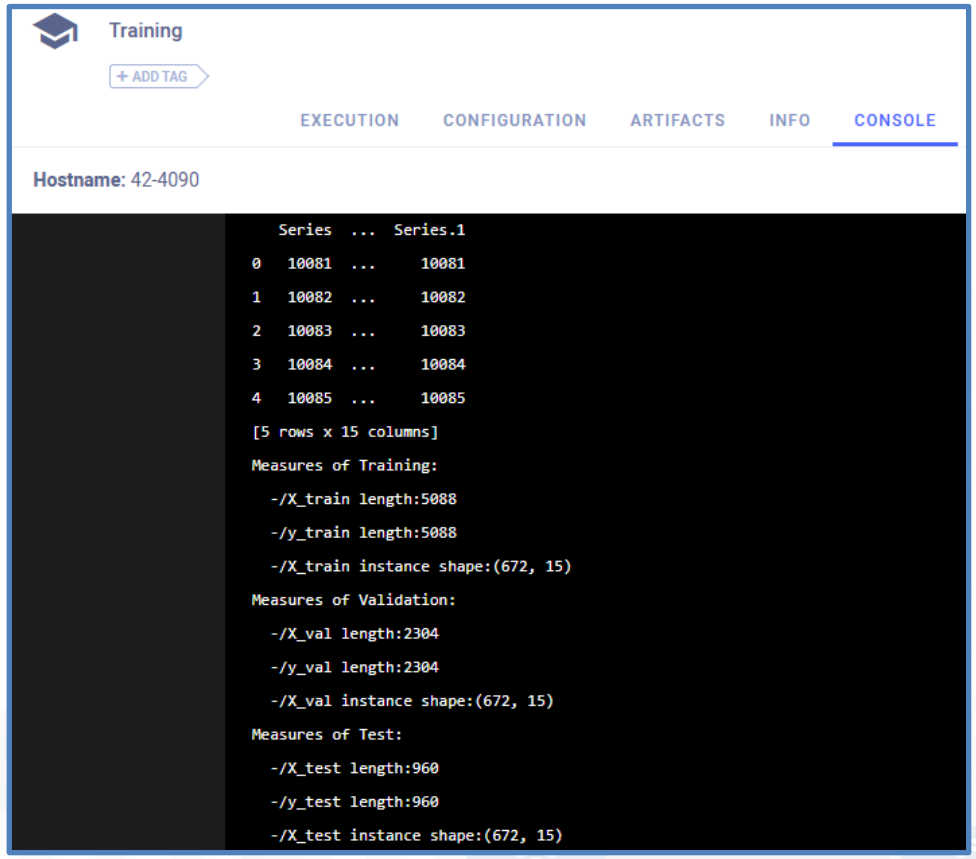
```
train_data = pd.read_csv(dataset_folder+"/train_data.csv")
val_data = pd.read_csv(dataset_folder+"/validation_data.csv")
test_data = pd.read_csv(dataset_folder+"/test_data.csv")
print(train_data.head())
print(test_data.head())
```

Jupyter notebook: uploading artifacts and saving model

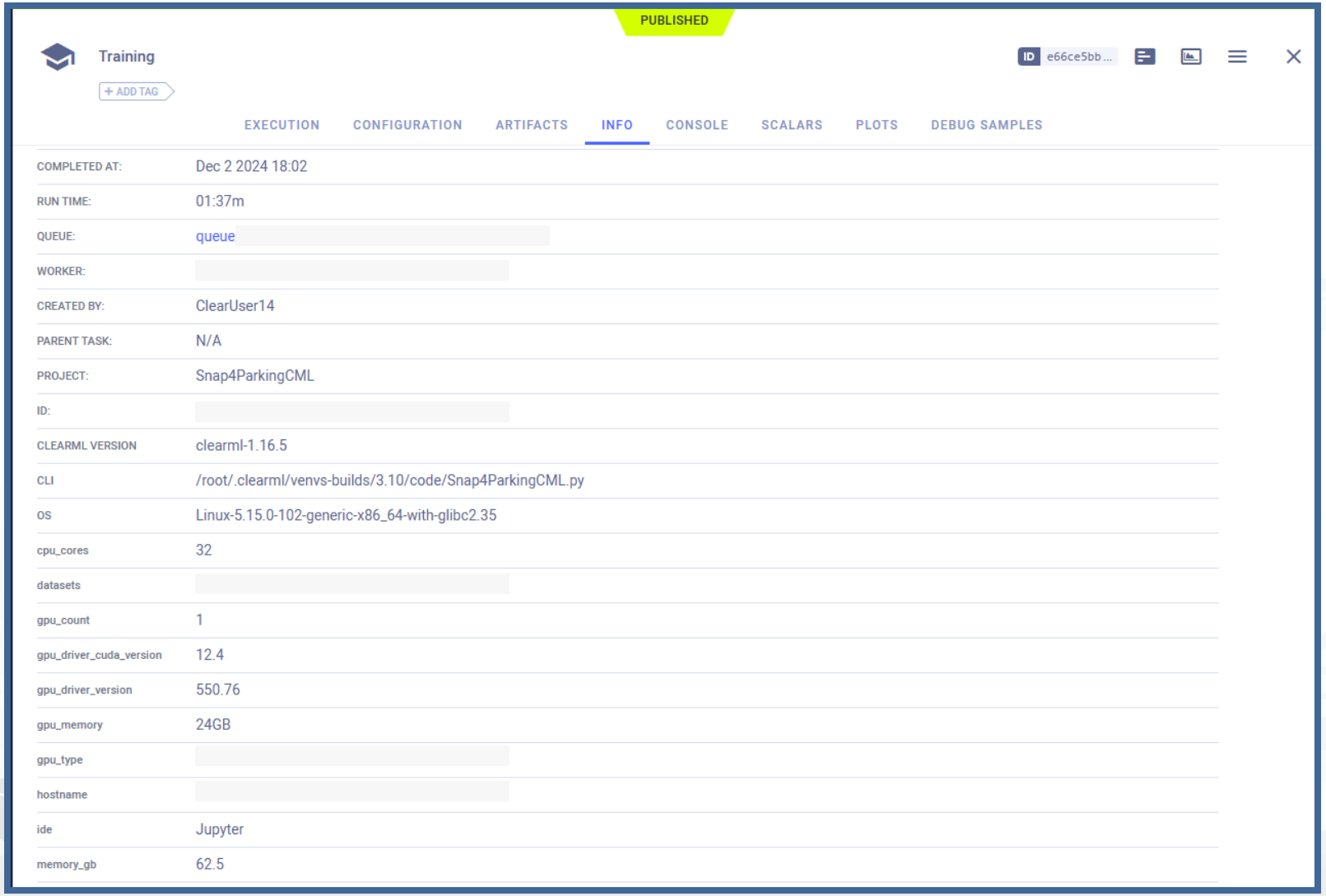
```
torch.save(model, "ParkingModel.pth")
task.upload_artifact(name = "train_losses", artifact_object = train_losses)
task.upload_artifact(name = "val_losses", artifact_object = val_losses)
plt.figure(figsize=(10, 6))
plt.title("Training vs Validation Loss")
plt.xlabel("epoch")
plt.ylabel("loss")
plt.plot(range(len(train_losses)), train_losses, label="training loss", color="red")
plt.plot(range(len(val_losses)), val_losses, label="validation loss", color="blue")
plt.legend()
plt.show()

task.mark_completed()
task.close()
task.publish()
```

ClearML: monitoring execution



ClearML: task ending savings 1



ClearML: task ending savings 2