



Workshop: ML API (Docker, ACR, ECS)
Professor: Fahd KALLOUBI Year:
2022/2023
Level: QFM-Data engineering

In this workshop, we will use the results obtained during the previous workshop. Next, and by building an API of this model with FastAPI, we will try to deploy it using two types of architectures for deploying an ML model:

- A service dedicated to the model via a Rest API
- A dedicated ML API as a microservice

To do this, we will use several inference targets by taking advantage of several cloud technologies (PaaS and IaaS):

- Heroku
- Microsoft Azure through Azure container instances

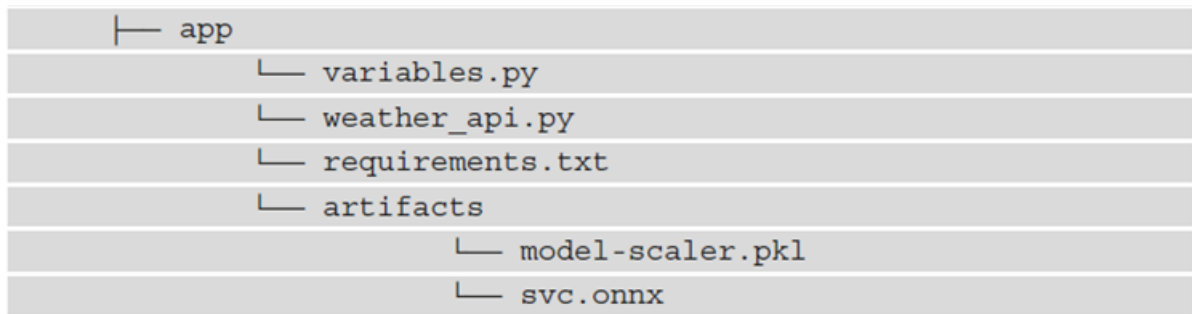
1. Building the API and deploying as a service

In the previous workshop, we were able to run a pipeline to produce a set of models by operating on our "weather_data.csv" dataset. In addition, the produced models can be downloaded from "Microsoft Azure ML Studio".



We will use these models to produce an API. In the workbench folder, open the "API_Microservices" folder.

In this folder, you will find a set of files and folders as shown in the figure below:



-Variables.py

In this file we defined the inputs of our API using the Pydantic library.

Using Pydantic, we created the “WeatherVariables” class which will in turn be used for validating our API variables.

-Weather_api.py

It is in this file that we will define the fastAPI service. The necessary artifacts are imported and used for the model to do the inference.

-Requirements.txt

This file contains all the packages required to run our service.

```
numpy
fastapi
uvicorn
pydantic
pickle5
scikit-learn==1.0.1
pandas
onnx
onnxruntime
```

To run this file, type the following command while in the “app” folder:

```
uvicorn weather_api:app --reload
```

FastAPI used OpenAPI (in know more : <https://www.openapis.org/>, <https://swagger.io/specification/>) Specification to serve the model. The OpenAPI Specification (OAS) is a language-agnostic standard interface for RESTful APIs. Using OpenAPI features, we can access the API documentation and get an overview of the API. You can access the API documentation and test the API at 127.0.0.1:8000/docs and it will take you to a Swagger-based UI (it uses OAS) to test your API.

To see the interactive documentation of the application go to <http://127.0.0.1:8000/docs> . More than that, you can test with other values on the form provided thanks to Swagger UI.

Now that we have executed our locally deployed ML API, we will deploy it in a PaaS namely Heroku.

To do this, add a file named “Procfile” in the workshop folder (already present in the folder) whose contents are as follows:

```
web: uvicorn weather_api:app --host 0.0.0.0 --port $PORT
```

The deployment steps are as follows:

1. Start by creating an account on <https://www.heroku.com>
2. Install Git: <https://git-scm.com/downloads>
3. Install Heroku CLI: <https://devcenter.heroku.com/articles/heroku-cli> , on the command prompt type the following command to ensure that Heroku CLI is installed:

```
heroku --version
```

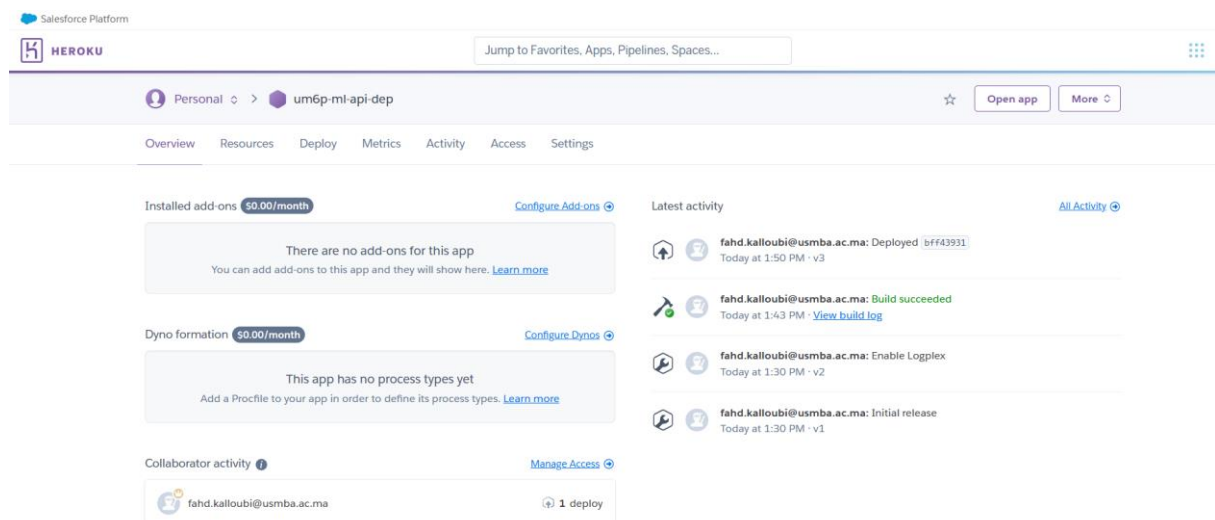
4. While in the “app” folder, run the following command to create a Heroku application (choose your own name)

```
// by launching heroku create, heroku will choose a random name //  
choose another name of your choice  
heroku create um6p-ml-api-dep
```

5. Run the following commands one after one so that we can upload our code to Heroku

- Git init
- heroku git:remote -a um6p-ml-api-dep git
- add .
- git commit -am "ML API um6p " git
- push heroku master

On the Heroku dashboard, click on your created application.



You must wait for Heroku to install all the required dependencies. Subsequently, and once the status changes to “build succeeded”, click on the “open app” button or in the command line type “heroku open”.

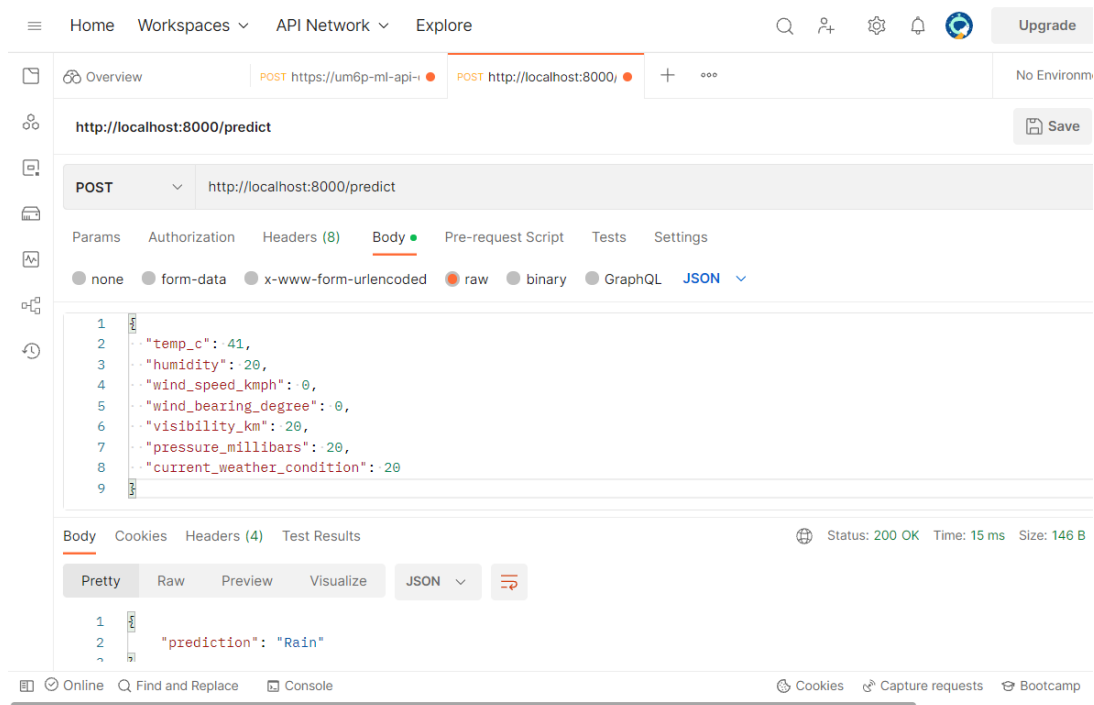


We have indeed deployed our service (ie, API) on the “Heroku” PaaS that we can consume it either using Swagger UI by visiting:<https://um6p-ml-apidep.herokuapp.com/docs> or via a dedicated application.

NB: Unfortunately, the Heroku PaaS has become paid, so we will use the service deployed locally (ie, <http://localhost:8000>)

1.1 Let's consume our API using Postman

We will now try a tool to query our API using curl requests. Postman is an easy-to-use GUI (download link: <https://www.postman.com/downloads/>).



1.2 Let's consume our API using a dedicated application

Like flask, FastAPI also has the same jinja2 template engine. Therefore, we will create an application based on FastAPI in order to consume our service through a user-friendly HTML interface.

Open the “app_consuming” folder.

Install the dependencies by running the command: `pip install -r requirements.txt`

As you notice, the content of the “weather_app.py” file is the same as that of “weather_api.py” except that we retrieve the content from an Html form located in the “Template/index.html” page. Furthermore, this content is then sent using the “Requests” library to our API “https://um6p-ml-api-dep.herokuapp.com or http://localhost:8000”.

To run the application (while in the “app_consuming” folder:

```
uvicorn weather_app:app --reload --port 5000
```

To test the application, open <http://localhost:8000/>. Try entering a few values to test whether the model works correctly.

The screenshot shows a web browser window with the address bar displaying 'localhost:8000/predict'. The page content is a dark blue form titled 'Weather Turku Port Prediction'. The form contains seven input fields, each with a label and a text input box: 'Temperature_C', 'Humidity', 'Wind_speed_kmph', 'Wind_bearing_degrees', 'Visibility_km', 'Pressure_millibars', and 'Current_weather_condition'. Below these fields is a large blue button labeled 'Predict'.

2. Deploy an ML API as a microservice

Now we will package the FastAPI service in a standardized way using Docker. This way we can deploy the Docker image or container to the deployment target of our choice.

We created a “Dockerfile” in which we used an official fastAPI image (tiangolo/uvicorn-gunicorn-fastapi) from Docker Hub.

```
FROM tiangolo/uvicorn-gunicorn-fastapi:python3.7
COPY ./app /app
RUN pip install -r requirements.txt
EXPOSED 80
CMD ["uvicorn", "weather_api:app", "--host", "0.0.0.0", "--port", "80"]
```

First, we use an official fastAPI Docker image from Docker Hub using the FROM command and pointing to the image - tiangolo/uvicorn-gunicornfastapi:python3.7. The image uses Python 3.7, which is compatible with fastAPI. Next, we copy the app folder to a directory named app inside the docker image/container. Once the app folder is copied to the Docker image/container, we will install the necessary packages listed in the requirements.txt file using the RUN command.

Next, we will EXPOSE port 80 for the Docker image/container. Finally, we will spin up the server inside the Docker image/container using the CMD command “uvicorn weather_api:app --host 0.0.0.0 --port 80”. This command points to the weather_api.py file to access the fastAPI app object and host the service on port 80 of the image/container.

2.1 Let's run our API as a local container

To test our API, we will follow the following steps:

1. We will start by building our Docker image. To do this, you must have Docker installed. While being in the folder “API_Microservices”, run the following command to build the Docker image.

```
docker build -t fastapi.
```

Following the execution of this command we will proceed to build our Docker image by following the steps prescribed in the Dockerfile. To check the creation of our image we can run the “docker images” command.

```

C:\Windows\System32\cmd.exe
=> => extracting sha256:aebd27b2d86a5a3a2cbe186247911047a7e432b9d17daad8f226597c0ea4276 1.3s
=> => extracting sha256:3b81dfff756ff433c0aa6bdd25335679c0e5f7baafba663151e57610912b564 2.9s
=> => extracting sha256:6e4f59a23b58b034aca9e284e24eb048a5bd8a08dfaf009cfd664804b5422e06 0.0s
=> => extracting sha256:5cdb61eb416d184070e2a3179d0fbf5195a965fb1a4051ba4ac497a42f711002 0.9s
=> => extracting sha256:41575650c078c48027e842a05cc34353cd2faee9c229c8353b56c67e72ab36c 0.0s
=> => extracting sha256:82af7648a68a3cdd74a95f3b8e467a68ad164374f726184cdedca6369af31c1c 1.7s
=> => extracting sha256:697e3cdf6fdb54685dd4b6ef30199efb83c70d5ba9060cd7beae411eef6c0664 0.0s
=> => extracting sha256:8d414b25a011a87ef0532614a30e5bfa150ccb1f6307ee8c9cea3d4dbc5be59 0.0s
=> => extracting sha256:3e3cfa737f563f22f21cab6f758afa05a8095c325ffbd1ce11060e1f4f51aa0 0.0s
=> => extracting sha256:1595db7eb9b17f8f89289166cfad8ff9dc7e39776bd627408ba5816d062c87da 0.0s
=> => extracting sha256:d2d3fcd4b870b1af2eb777f132032774d461109aa700325a3dcb351bc9aa14e6 0.0s
=> => extracting sha256:35f87fd78dc8b33cd8339a58d1da5f75fe2fd8c85478891e5e70e02d793e4c42 0.0s
=> => extracting sha256:146e266752fc4dc9c2073f27ca8769e23189df00d9b57b56aa59340e90b67a0 0.0s
=> => extracting sha256:57df244389c008c329cd8ee89a74f8f36e3984715e65607caca22c30d21efe83 2.1s
=> => extracting sha256:e8391aa94779ce81d31831f53777c61b634a0e7df78be5549e41edf932679f48 0.0s
=> [2/3] COPY ./app /app 0.3s
=> [3/3] RUN pip install -r requirements.txt 97.4s
=> exporting to image 12.5s
=> exporting layers 12.4s
=> writing image sha256:c4ec509ea1e18277d0d821449c2df4bd86f82ee2115580b93e61dbb0b5f29e28 0.0s
=> naming to docker.io/library/fastapi 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

C:\migration\Cours UM6P\MLops DataOps\Labs\Lab 6\API_Microservices>docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
fastapi              latest          c4ec509ea1e1   22 seconds ago 1.37GB
docker/getting-started latest          26d80cd96d69   10 days ago    28.5MB

C:\migration\Cours UM6P\MLops DataOps\Labs\Lab 6\API_Microservices>

```

2. We will run a container locally by executing the following command:

```
docker run -d -p 80:80 --name weathercontainer fastapi
```

To display the created “weathercontainer” image, you can run the “docker container ps --all” command. However, you can see that the service is served on port 80.

```

Administrateur : Invite de commandes
C:\migration\Cours UM6P\MLops DataOps\Labs\Lab 6\API_Microservices>docker build -t fastapi .
[+] Building 109.0s (8/8) FINISHED
=> [internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 225B 0.0s
=> [internal] load .dockerignore 0.1s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/tiangolo/uvicorn-gunicorn-fastapi:python3.7 20.7s
=> CACHED [1/3] FROM docker.io/tiangolo/uvicorn-gunicorn-fastapi:python3.7@sha256:f5770422a8875fe3d677e1bda724ee 0.0s
=> [internal] load build context 0.1s
=> => transferring context: 436B 0.0s
=> [2/3] COPY ./app /app 0.2s
=> [3/3] RUN pip install -r requirements.txt 83.3s
=> exporting to image 4.3s
=> exporting layers 4.2s
=> writing image sha256:33d9c2c6ff3f9695556aa90765d0fc72ea91a7a6dd5c3ea4c4e7496b945e3028 0.0s
=> naming to docker.io/library/fastapi 0.0s

C:\migration\Cours UM6P\MLops DataOps\Labs\Lab 6\API_Microservices>docker run -d -p 80:80 --name weathercontainer fastapi
bb249c128052b86b9b4af07d0ae02fa3ff63f8cdb529b91e65b831b785c83eec

C:\migration\Cours UM6P\MLops DataOps\Labs\Lab 6\API_Microservices>docker container ps --all
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
bb249c128052   fastapi       "uvicorn weather_api..." 22 seconds ago Up 21 seconds  0.0.0.0:80->80/tcp       weathercontainer

C:\migration\Cours UM6P\MLops DataOps\Labs\Lab 6\API_Microservices>docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
fastapi              latest          33d9c2c6ff3f   About an hour ago 1.46GB

C:\migration\Cours UM6P\MLops DataOps\Labs\Lab 6\API_Microservices>

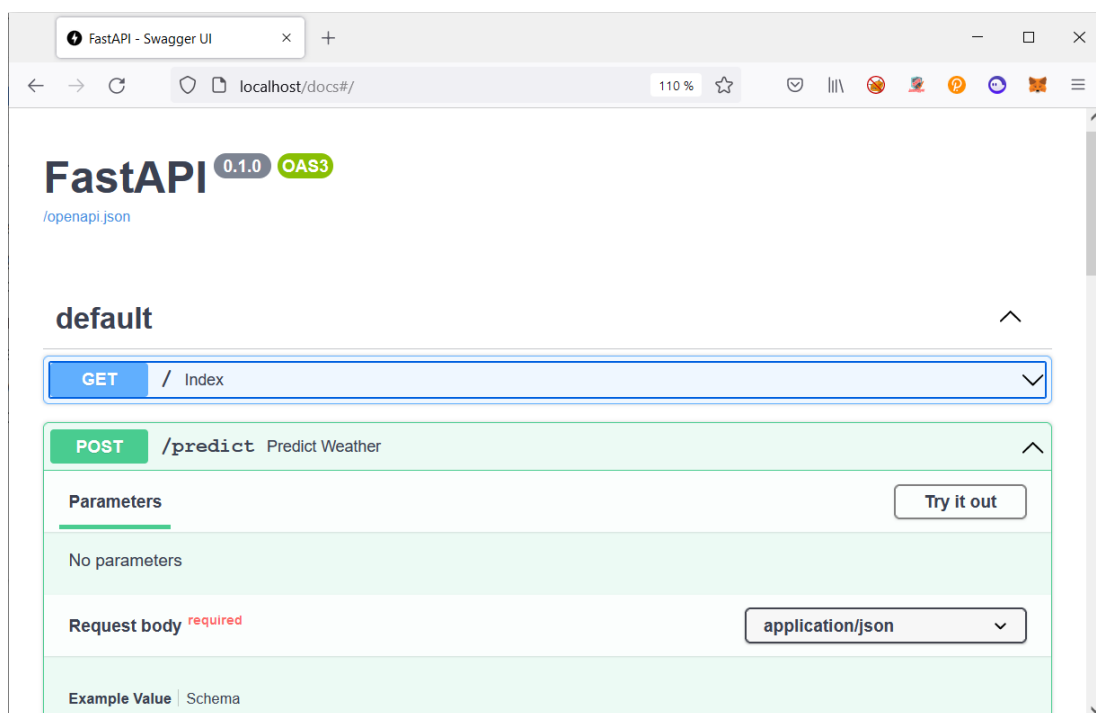
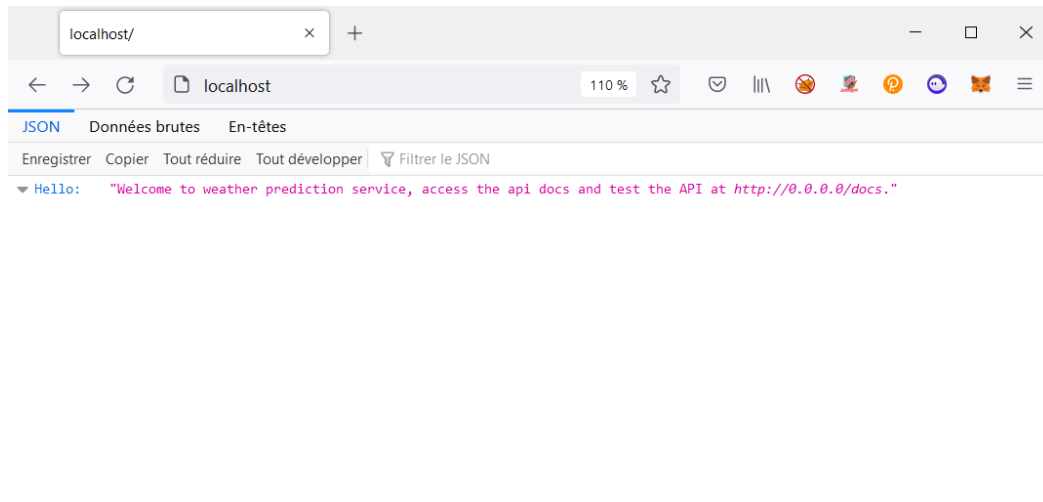
```

Below are some useful Docker commands:

docker container rm <container-id>	To delete a container
docker container ps --all	To display all containers
docker logs <container-id>	To display the logs of a container (useful for debugging)

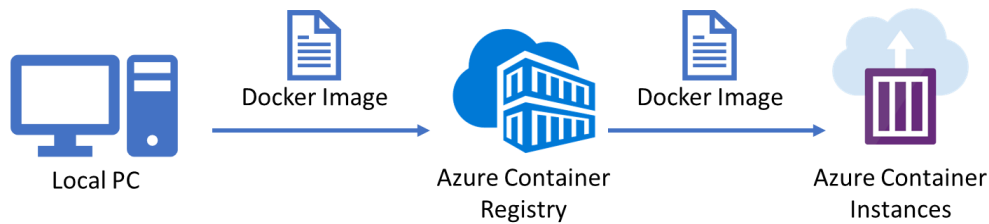
<code>docker rmi <image-ID></code>	To delete an image
<code>docker images</code>	To view all images

You can access and test the service from the browser by visiting "0.0.0.0:80".



3. Let's deploy our container on Azure

So far we have been able to build our image and we have run it as a local container. The process that we will follow to deploy our image on Azure is described in the figure below:



The following steps will be implemented in order to deploy our ML API as a microservice on Azure:

1. Create an Azure Container Registry instance
2. Mark the image
3. Upload the image to Azure Container Registry
4. Deploy the container from ACR to Azure Container Instances
5. Show the API on the browser
6. View container logs

Before you begin:

- Install Azure CLI (> 2.0.29): <https://learn.microsoft.com/en-us/cli/azure/installazure-cli> (the .msi file is given in the workshop folder), run the following command to ensure that Azure CLI is installed.

```
az --version
```

-Make sure Docker is installed

3.1 Creating an Azure Container Registry instance

In the Azure portal, click on “create resource” and then choose “containers” in the “categories” section and then click on “Container Registry”.

Name your container registry “um6plab1” and choose your resource group and click “create”.

The screenshot shows the Azure portal interface for the 'um6plab1' Container Registry. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Quick start, Events, Settings, Access keys, Encryption, and Identity. The main content area displays the 'Essentials' section with the following details:

- Resource group (move): [Um6p-ds](#)
- Location: France Central
- Subscription (move): [Azure for Students](#)
- Subscription ID: 513875a3-c381-4a6b-949e-564892a2304e
- Login server: um6plab1.azurecr.io
- Creation date: 9/25/2022, 3:19 PM GMT+1
- SKU: Standard
- Provisioning state: Succeeded
- Soft Delete (Preview): Disabled

Below the Essentials section, there are two cards:

- Usage:** A bar chart showing storage usage. Included in SKU: 100 GiB, Used: 0.55 GiB, Additional storage: 0.00 GiB.
- ACR Tasks:** A card with a trash icon and text: 'Build, Run, Push and Patch containers in Azure with ACR Tasks. Tasks supports Windows, Linux and ARM with QEMU.' with a 'Learn more' link.

On the command prompt, we will connect to our container registry. To do this, type the following command:

```
az acr login --name um6plab1 //  
output: Login Succeeded
```

3.2 Mark image for ACR

In order to put the image in a private registry (ACR), one must first mark the image with the full name of the registry connection server.

-Let's display the full name of the connection server first

```
az acr show --name um6plab1 --query loginServer --output table //  
um6plab1.azurecr.io
```

-Let's display the image that we were able to build (ie, fastapi)

Docker images

- We will now tag the "fastapi" image with the full name of the server (um6plab1.azurecr.io)

```
docker tag fastapi um6plab1.azurecr.io/fastapi:v1
```

Run the docker images command to see that another image has been created with the appropriate tag

-Push the marked image to ACR

```
docker push um6plab1.azurecr.io/fastapi:v1
```

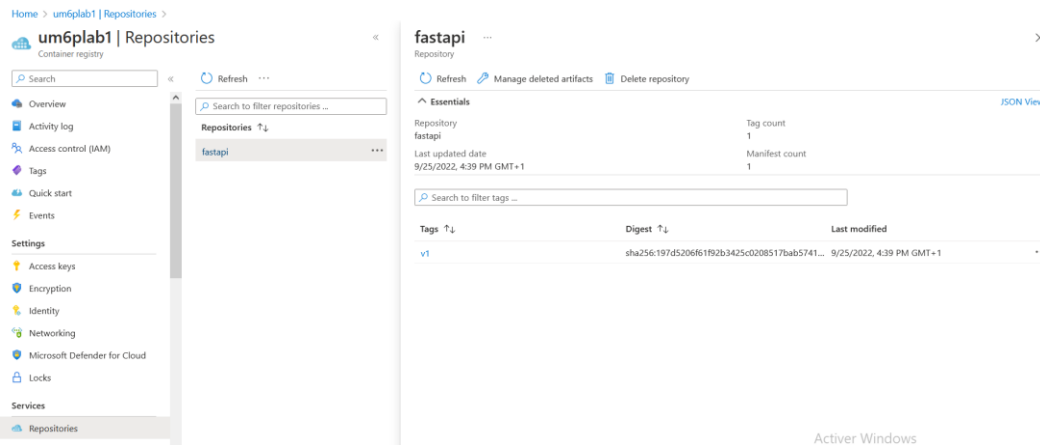
- Let's display the images in ACR

```
az acr repository list --name um6plab1 --output table
```

- Let's display the tag for our image in ACR

```
az acr repository show-tags --name um6plab1 --repository fastapi --output table
```

In the Azure portal, click on your ACR "um6plab1" and then on "repository" to display the image thus created



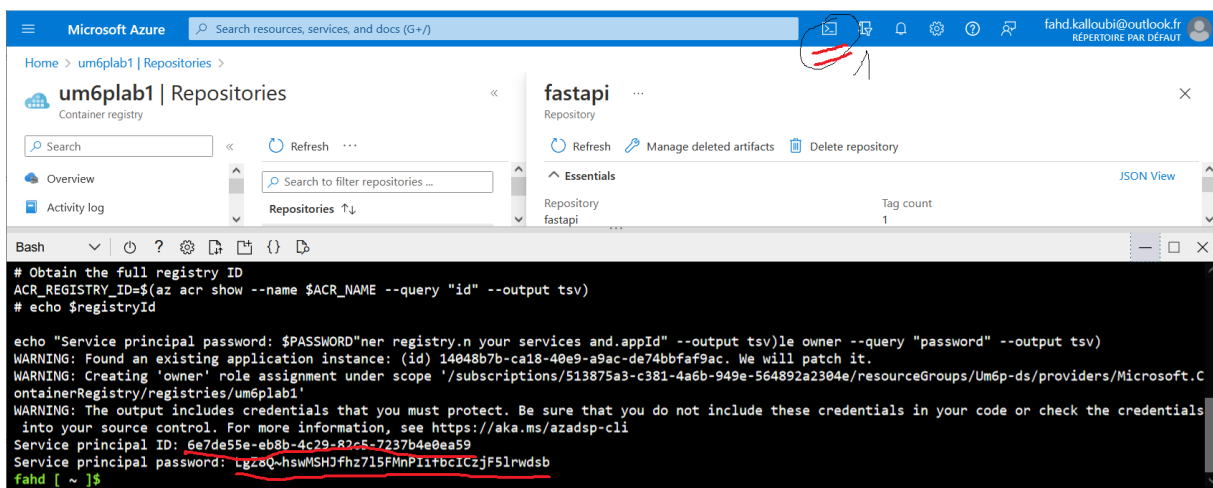
-Let's display the full name of the container registry connection server

```
az acr show --name um6plab1 --query loginServer
```

3.3 Let's deploy the container

In order to pull our image from ACR to ACI we need to create and configure an Azure AD “active directory” backend service with pull permissions on your registry. Next, you start a container in Azure Container Instances (ACI) which pulls its image from your private registry, using the service created for authentication. For more information visit the following link:<https://learn.microsoft.com/enus/azure/container-registry/container-registry-auth-aci>

In the Azure portal, click on the “Cloud Shell” command prompt to launch it. Then, copy the code from the “azure-service-principal.sh” file provided in the workshop folder and paste it into the “cloud shell” command prompt. After running this file, you will output the backend service authentication information that we will use to connect to ACR as shown in the figure below.



Now we will use the “az container create” command to deploy the container.

- “um6plab1.azurecr.io”:is the value obtained by running the previous command
- The parameter values “--registry-username»And “--registry-password »are the values obtained by running the previous script (underlined in the figure)

- "um6pdsaci": is the value of the DNS server I chose

```
az container create \  
  -- resource-group um6p-ds \  
  -- name fastapi\  
  -- image um6plab1.azurecr.io/fastapi:v1 \  
  -- cpu 1 \  
  -- memory 1 \  
  -- registry-username 6e7de55e-eb8b-4c29-82c5-7237b4e0ea59 \  
  -- registry-password 67P8Q~0zWjB7fcmUYfE77NW5DLT1SrtWYucHkc_U \  
  -- registry-login-server um6plab1.azurecr.io \  
  -- ip-address Public \  
  -- dns-name-label um6pdsaci \  
  -- ports 80
```

- To check the deployment status, run the following command (replace Um6p-ds with the name of your resource group)

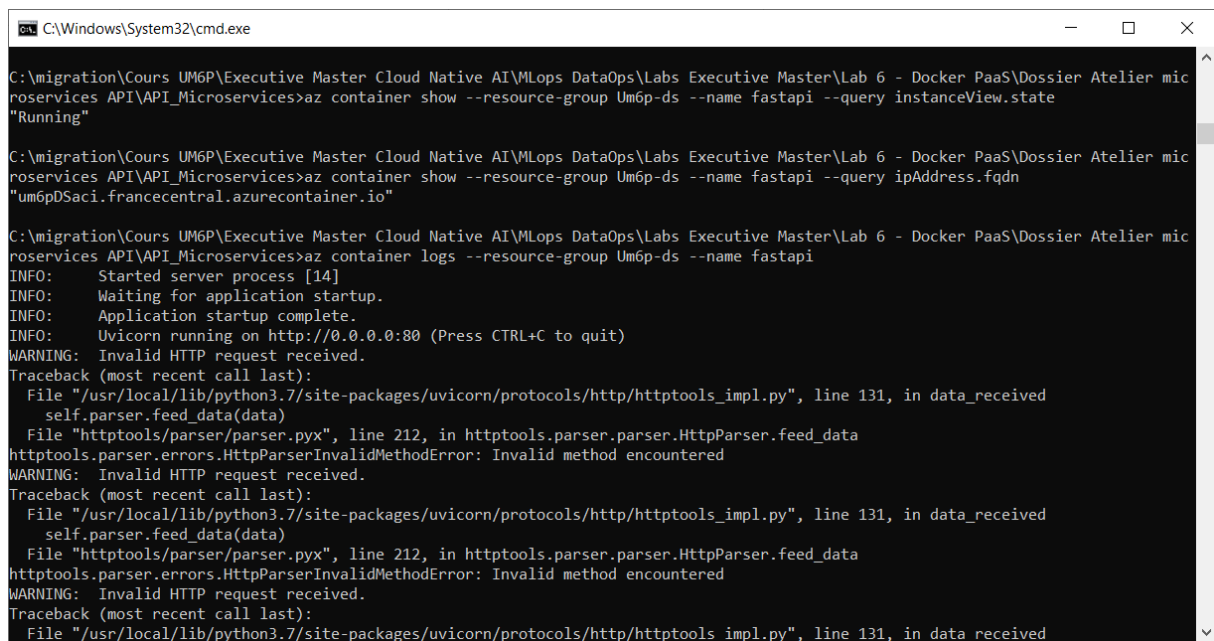
```
az container show -- resource-group Um6p-ds --name fastapi --query  
instanceView.state
```

- To view the app (app link)

```
az container show --resource-group Um6p-ds --name fastapi --query ipAddress.fqdn
```

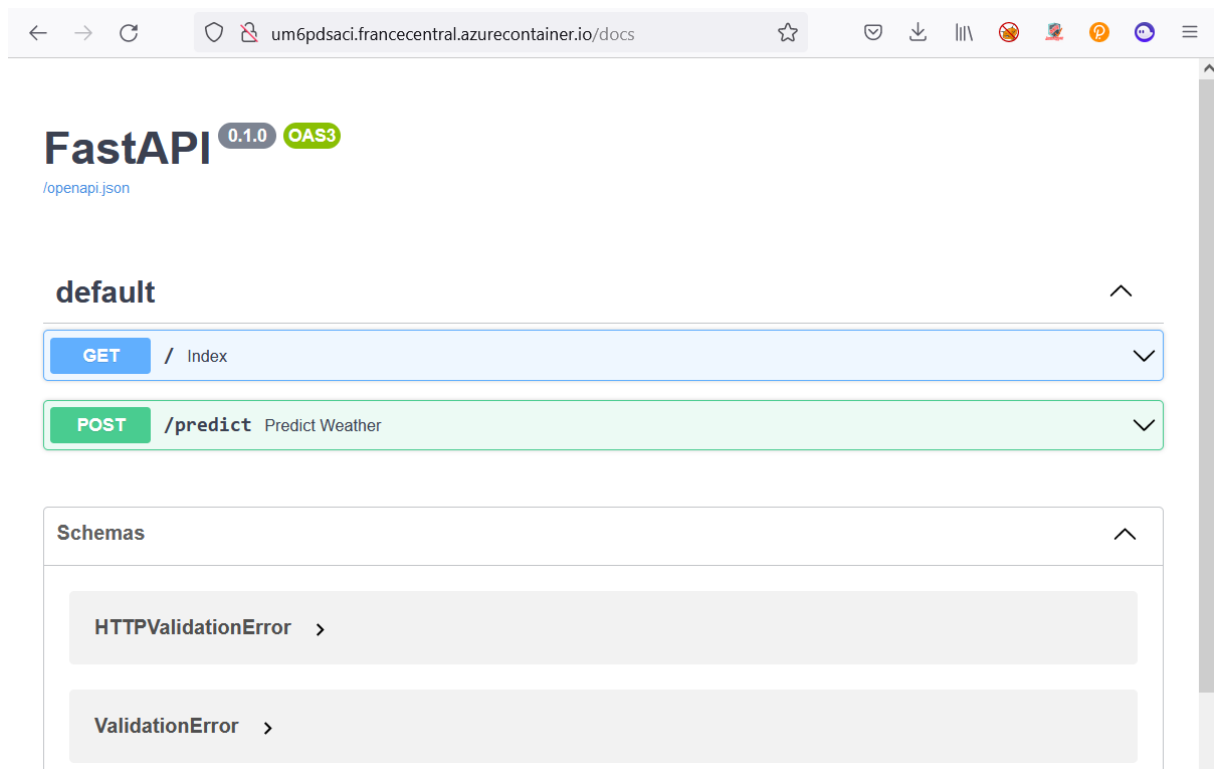
- To display container log output

```
az container logs --resource-group Um6p-ds --name fastapi
```



```
C:\Windows\System32\cmd.exe  
  
C:\migration\Cours UM6P\Executive Master Cloud Native AI\MLops DataOps\Labs Executive Master\Lab 6 - Docker PaaS\Dossier Atelier mic  
roservices API\API_Microservices>az container show --resource-group Um6p-ds --name fastapi --query instanceView.state  
"Running"  
  
C:\migration\Cours UM6P\Executive Master Cloud Native AI\MLops DataOps\Labs Executive Master\Lab 6 - Docker PaaS\Dossier Atelier mic  
roservices API\API_Microservices>az container show --resource-group Um6p-ds --name fastapi --query ipAddress.fqdn  
"um6pDSaci.francecentral.azurecontainer.io"  
  
C:\migration\Cours UM6P\Executive Master Cloud Native AI\MLops DataOps\Labs Executive Master\Lab 6 - Docker PaaS\Dossier Atelier mic  
roservices API\API_Microservices>az container logs --resource-group Um6p-ds --name fastapi  
INFO: Started server process [14]  
INFO: Waiting for application startup.  
INFO: Application startup complete.  
INFO: Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)  
WARNING: Invalid HTTP request received.  
Traceback (most recent call last):  
  File "/usr/local/lib/python3.7/site-packages/uvicorn/protocols/http/httptools_impl.py", line 131, in data_received  
    self.parser.feed_data(data)  
  File "httptools/parser/parser.pyx", line 212, in httptools.parser.parser.HttpParser.feed_data  
httptools.parser.errors.HttpParserInvalidMethodError: Invalid method encountered  
WARNING: Invalid HTTP request received.  
Traceback (most recent call last):  
  File "/usr/local/lib/python3.7/site-packages/uvicorn/protocols/http/httptools_impl.py", line 131, in data_received  
    self.parser.feed_data(data)  
  File "httptools/parser/parser.pyx", line 212, in httptools.parser.parser.HttpParser.feed_data  
httptools.parser.errors.HttpParserInvalidMethodError: Invalid method encountered  
WARNING: Invalid HTTP request received.  
Traceback (most recent call last):  
  File "/usr/local/lib/python3.7/site-packages/uvicorn/protocols/http/httptools_impl.py", line 131, in data_received
```

In order to view our containerized API deployed to ACI, paste the output of the application view command into your browser.



Using the application created previously (ie, app_consuming), consume this service using this application.

3.4 Resource cleaning

To avoid further overhead and if you do not need the resources created in this lab, you can run the following command:

```
az group delete --name um6p-ds
```

4. Deploy an ML model to ACI using Azure ML and MLflow

In the "Azure MLflow Deploy" folder open the "Train & Deploy local and ACI.ipynb" notebook and follow the steps described there.