Using the DEEP-Hybrid-Datacloud platform

Remote webinar for EGI-ACE
01 December 2021

Ignacio Heredia
iheredia@ifca.unican.es
Instituto de Física de Cantabria (CSIC-UC)
Introduction - The project

- The project was carried out with European Horizon 2020 funds.
- The project provides new generation of e-infrastructures that harness latest generation technologies, supporting deep learning and other intensive computing techniques to exploit very large data sources.
- It aims to lower the adoption barriers for new communities and users, satisfying the needs of both research, education communities and citizen science.

Project partners:
## Introduction - The users

### Basic

No machine learning knowledge. Just give me a working model to make predictions.

We offer:

- a **catalogue** full of ready-to-use modules to perform inference with your data
- an **API** to easily interact with the services
- solutions to run the inference in local or **Cloud resources**
- the ability to develop complex topologies by **composing different modules**

### Intermediate

I want to retrain a working model on my personal dataset.

We offer:

- the ability to train out-of-the-box a module of the **catalogue** on your personal dataset
- an **API** to easily interact with the model
- **data storage** resources to access your dataset (DEEP-Nextcloud, OneData, ...)
- the ability to deploy the developed service on **Cloud resources**
- the ability to **share the service** with other users in the user's catalogue

### Advanced

I want to develop my custom Deep Learning model.

We offer:

- a ready-to-use environment with the **main DL frameworks** running in a dockerized solution running on different types of hardware (CPUs, GPUs, etc)
- **data storage** resources to access your dataset (DEEP-Nextcloud, OneData, ...)
- the ability to deploy the developed module on **Cloud resources**
- the ability to share the module with other users in the open **catalogue**
- the possibility to integrate your module with the **API** to enable easier user interaction
Introduction - The users

- Basic Predict
- Intermediate Retrain
- Advanced Develop

Marketplace

Dashboard

Deep Hybrid Datacloud infrastructure
Authentication - Authorization - Storage - Computing - Orchestration

New modules

- DEEPaaS API (serverless)
- DEEPaaS API (deployment)
- Nextcloud / OneData (storage)
- JupyterLab
  - Cookiecutter - Github - Dockerhub
- Nextcloud / Onedata (storage)
Introduction - Useful links

- **Homepage**: [https://deep-hybrid-datacloud.eu/](https://deep-hybrid-datacloud.eu/)
- **Dashboard**: [https://train.deep-hybrid-datacloud.eu/](https://train.deep-hybrid-datacloud.eu/)
- **Github**: [https://github.com/deephdc](https://github.com/deephdc)
- **DockerHub**: [https://hub.docker.com/u/deephdc/](https://hub.docker.com/u/deephdc/)
- **Documentation**: [https://docs.deep-hybrid-datacloud.eu/en/latest/](https://docs.deep-hybrid-datacloud.eu/en/latest/)
- **NextCloud**: [https://nc.deep-hybrid-datacloud.eu/](https://nc.deep-hybrid-datacloud.eu/)

(* these slides are available here)
Introduction - Webinar outline

1. Exploring the Marketplace
2. Using the Dashboard
   a. Deploying a module
   b. Making inference
   c. Retraining a module on a new dataset
3. Developing a new module
   a. Deploying the DEEP development environment
   b. Using the cookiecutter
   c. Integrating it with DEEPaaS API
   d. Adding the model to the CI pipeline
   e. Adding the model to the Marketplace
4. What's next?
   a. New DEEPaaS features
   b. Friendlier UI for module inference
   c. Training Dashboard
Exploring the Marketplace
The Marketplace
The Dashboard
The Dashboard - Module Overview
The Dashboard - Deploying a module

Configurable options

- **docker image** (from deep-oc, but also custom docker images)
- **hardware** (#cpus, #gpus, RAM)
- **storage** (OneData, Nextcloud volumes)
- **services** (DEEPaaS, JupyterLab)
The Dashboard - Making inference

Launch `image-classification-tf` module with DEEPaaS.

POST /v2/models/ingclas/predict/ Make a prediction given the input data

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Select the image you want to classify. Default value: null</td>
</tr>
<tr>
<td>url</td>
<td>Select an URL of the image you want to classify. Default value: null</td>
</tr>
<tr>
<td>timestamp</td>
<td>Model timestamp to use for prediction. Group name: testing, Choice: [&quot;default_imagenet&quot;] Type: str</td>
</tr>
</tbody>
</table>

Response body

```
{
  "status": "OK",
  "predictions": {
    "labels": [
      "lion",
      "far_c coal",
      "brown bear",
      "hyaena",
      "timber wolf"
    ],
    "probabilities": [
      0.9763001283536987,
      0.9553028865605559273,
      0.04880094675388466,
      0.0014793890243631958,
      0.001660078120723568
    ]
  }
}
```
The Dashboard - Retraining a module

1) Launch **image-classification-tf** module with JupyterLab (remember adding password).
2) Copy some demo files to make a mock dataset.
3) Terminal: `deepaas-run --listen-ip 0.0.0.0` to launch DEEPaas.
Develop your module
Developing - DEEP Development Environment

**Configurable options**

- **docker image** (from deep-oc, but also custom docker images). Eg:
  - **Tensorflow docker**
  - **Pytorch docker**
  - ...

- **hardware** (#cpus, #gpus, RAM)

- **storage** (OneData, Nextcloud volumes)

- **services** (DEEPaaS, JupyterLab)

---

The DEEP Development Environment provides a ready to use JupyterLab instance that enables you to develop code using Jupyter notebooks, text editors, terminals, and custom components in a flexible, integrated, and extensible manner.

Create environment

Configure training: DEEP Development Environment

- **Template:**
  - Default

- **Hardware configurations:**
  - CPU

- **Docker tag:**
  - Latest

- **Docker options:**
  - Equivalent of `--privileged docker flag`: False

- **Jupyter options:**
  - Password for JupyterLab: Should have at least 9 characters.
Developing - DEEP Cookiecutter

This is the easiest way to develop any new module from scratch as it will take care of generating all the nitty-gritty details that we will cover in the following slides (entrypoints, files, Jenkinsfile, Dockerfile, etc).

- Use the command: `cookiecutter https://github.com/indigo-dc/cookiecutter-data-science`
- Answer questions:
  - Project name, description, version, license type
  - Author name, email, Github account
  - Dockerhub account, Docker base image
- This will generate two folders. Eg:
  - `mymodule`: This is where the project code is located
    - Example: https://github.com/deephdc/image-classification-tf
  - `DEEP-OC-mymodule`: This contains the Dockerfile of the project
    - Example: https://github.com/deephdc/DEEP-OC-image-classification-tf
Developing - Integrating with DEEPaaS

Head over to mymodule. Any module that wants to integrate with DEEPaaS should have two minimum requirements:

- it should define a file (eg. `mymodule/mymodule/api.py`) with the functions to interact with the module. These functions should define:
  - the model metadata
  - the input args for training
  - the input args for prediction
  - the response structure for prediction
  - the train function
  - the predict function
  - a model warming function for prediction

→ Minimal example: [https://github.com/deephdc/demo_app/blob/master/demo_app/api.py](https://github.com/deephdc/demo_app/blob/master/demo_app/api.py)
→ Full example: [https://github.com/deephdc/image-classification-tf/blob/master/imgclas/api.py](https://github.com/deephdc/image-classification-tf/blob/master/imgclas/api.py)

- it should define an entrypoint in `mymodule/setup.cfg` pointing to that file

Developing - Customizing the Dockerfile

- Head over to DEEP-OC-mymodule and modify the Dockerfile following your needs:
  - install additional packages,
  - change the base image,
  - etc.
Developing - Continuous Integration

- Both `mymodule` and `DEEP-OC-mymodule` have their respective `Jenkinsfile` that define the actions to be taken when a change is committed to the repos.

- Typical workflows:
  - `mymodule/Jenkinsfile` will:
    - run PEP8 style analysis
    - trigger of `DEEP-OC-mymodule/Jenkinsfile`.
    → Example: [https://github.com/deephdc/image-classification-tf/blob/master/Jenkinsfile](https://github.com/deephdc/image-classification-tf/blob/master/Jenkinsfile)
  
  - `DEEP-OC-mymodule/Jenkinsfile` will:
    - build Docker images for different branches (train/test) and different hardware (cpu/gpu)
    - upload the image to DockerHub
    - build Docker images of other dependent modules. For example, changes in the code of image-classification should rebuild all Docker images of applications that were trained with that code (plant classifier, seed classifier, etc).
    - refresh the module page in the Marketplace (see next step)
Developing - Integrating to the Marketplace

- Head over to DEEP-OC-mymodule and modify metadata.json with the info relevant to your module. This is the information that will appear in the Marketplace page.
- Make a Pull Request to add your module here. This will create the Jenkins pipeline for your module and will add the module to the Marketplace and the Training Dashboard.

Congratulations, you're done!
What's next?
What's next? - New DEEPaaS features

- Integration with Dask
  
  **Mature**

- Easier module integration via decorators/hints
  
  **Midterm**

Before (webargs)

```python
from webargs import fields, validate

def get_predict_args():
    arg_dict = {
        "demo-str": fields.Str(required=False, missing='some-string'),
        "demo-int": fields.Int(required=False, missing=1),
    }

def predict(**kwargs):
    return {'demo-list': [1, 2, 3]}
```

After (type hints)

```python
from typing import Dict, List

def predict(**kwargs):
    return {'demo-list': [1, 2, 3]}
```
What's next? - Friendlier inference UI

Before (Swagger UI)

After (Gradio based)
What's next? - Friendlier inference UI

Before (Swagger UI)

After (Gradio based)  
Mature

demo_app
A minimal toy application for demo and testing purposes. We just implemented dummy inference, i.e., we return the same inputs we are fed.

Outputs
What's next? - Training dashboard

- Organizing training run in experiments
  - hyperparameter optimization
  - easier side-by-side comparison of training runs

- Richer module metadata language, to keep track of:
  - training datasets
  - models
  - training execution pipelines