

# Using the DEEP-Hybrid-Datacloud platform

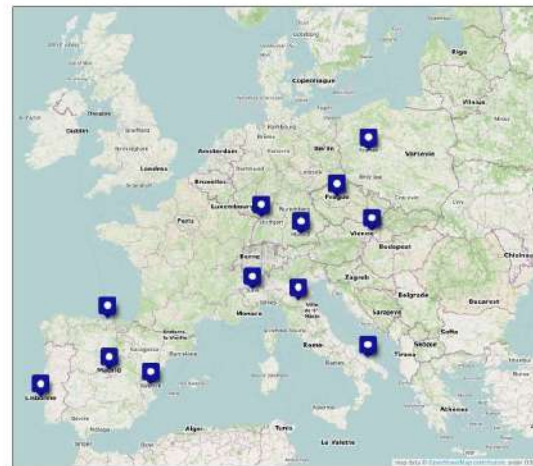
Remote webinar for EGI-ACE  
01 December 2021

Ignacio Heredia  
[iheredia@ifca.unican.es](mailto: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

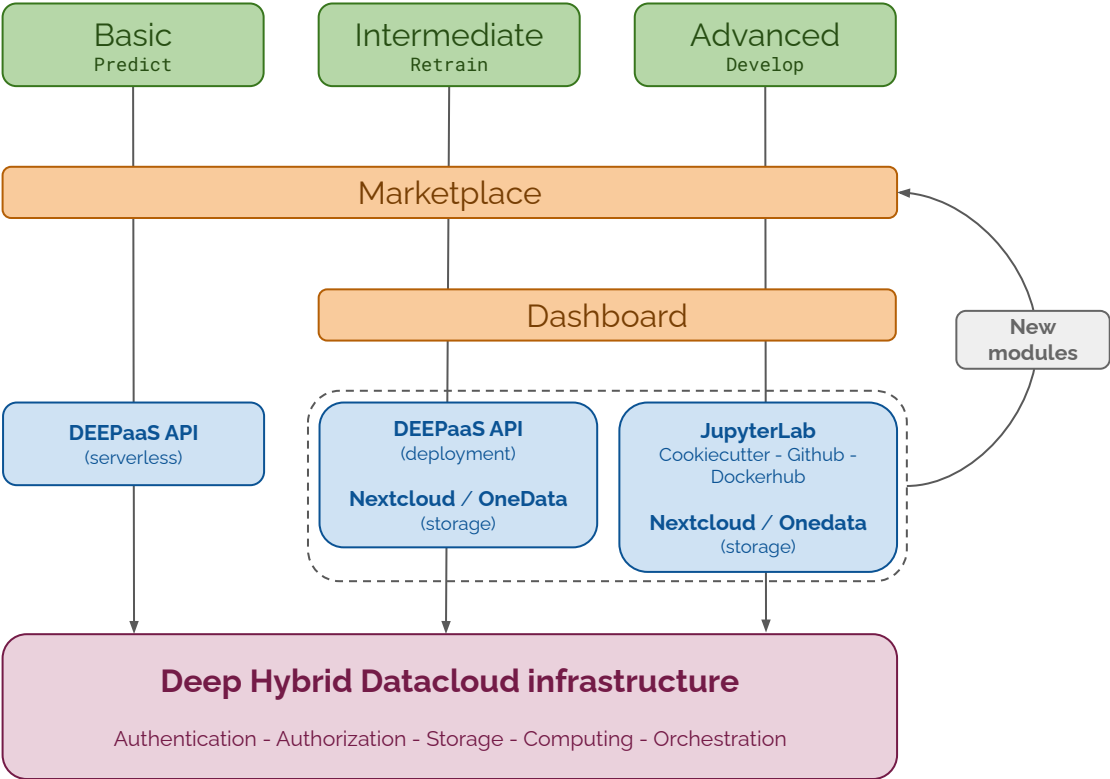
## Advanced

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



# Introduction - Useful links

	<b>Homepage</b>	<a href="https://deep-hybrid-datacloud.eu/"><u>https://deep-hybrid-datacloud.eu/</u></a>
	<b>Marketplace</b>	<a href="https://marketplace.deep-hybrid-datacloud.eu/"><u>https://marketplace.deep-hybrid-datacloud.eu/</u></a>
	<b>Dashboard</b>	<a href="https://train.deep-hybrid-datacloud.eu/"><u>https://train.deep-hybrid-datacloud.eu/</u></a>
	<b>Github</b>	<a href="https://github.com/deephdc"><u>https://github.com/deephdc</u></a>
	<b>DockerHub</b>	<a href="https://hub.docker.com/u/deephdc/"><u>https://hub.docker.com/u/deephdc/</u></a>
	<b>Documentation</b>	<a href="https://docs.deep-hybrid-datacloud.eu/en/latest/"><u>https://docs.deep-hybrid-datacloud.eu/en/latest/</u></a>
	<b>NextCloud</b>	<a href="https://nc.deep-hybrid-datacloud.eu/"><u>https://nc.deep-hybrid-datacloud.eu/</u></a>

(\* 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

DEEP OPEN CATALOG PROJECT PAGE DOCS MODULES CATEGORIES

## DEEP Open Catalog

Welcome to the DEEP Open Catalog

DEEP Hybrid DataCloud delivers a comprehensive platform to easily develop, build, deploy and deploy AI that integrates Machine Learning Description modules on top of distributed infrastructures.

In the DEEP Open Catalog you can find ready-to-use modules in a variety of domains. These modules can be executed on your local box or on a production server or on top of CLOUD INFRASTRUCTURES SUPPORTED BY DEEP-HYBRID DATA CLOUD.

Explore our marketplace!

Artistic style transfer

Model: artistic\_style\_transfer (Pretrained)

4 models to easily artistic style transfer using pytorch.

View Model

Bird sound classifier

Model: bird\_classifier (Pretrained)

Classify audio by learning bird sounds from the internet data sets.

View Model

TF Benchmarks

Model: tf\_benchmarks

tf\_benchmarks executed on GCP VMs.

View Model

Object Detection and Classification with Pytorch

Model: object\_detection\_classification\_pytorch (Pretrained)

A neural Network Convolutional Neural Network (CNN) to detect objects and classification.

View Model

2D semantic segmentation

Model: 2d\_semantic\_segmentation (Pretrained)

2D semantic segmentation trained on the COCO dataset.

View Model

Train an audio classifier

Model: train\_audio\_classifier (Pretrained)

Train your own audio classifier on your custom dataset. It comes also pre-trained on the T5 ImageNet classes.

View Model

OPEN CATALOG PROJECT PAGE DOCS MODULES CATEGORIES

## Train an image classifier

Train your own image classifier with your custom dataset. It comes also pre-trained on the T5 ImageNet classes.

Model: train\_image\_classifier (Pretrained)

Published by DEEP Hybrid DataCloud Consortium  
Created: Tue 01 January 2019 - Updated: Mon 21 September 2020

### Model Description

View

The deep learning revolution has brought significant advances in a number of fields (1), primarily related to image and speech recognition. The standardization of image classification tasks like the [ImageNet Large Scale Visual Recognition Challenge \(ILSVRC\)](#) has resulted in a reliable way to compare top performing architectures.

This Docker container contains the tools to train an image classifier on your personal dataset. It is a highly customizable tool that let's you choose between one of different [state-of-the-art](#) architectures and training parameters.

The container also comes with a pre-trained general purpose image classifier based on ImageNet.

The PRECINCT method expects an RGB image as input (or the list of an RGB image) and will return a JSON with the top 3 predictions.

Categories

License

Try it live!

View Model

API Specification

Configure and train

Training Dashboard

Get the code

GitHub

Docker Hub

Get the data

Dataset

Training Files

Citing this module

Citation





# The Dashboard

# The Dashboard - Module Overview

The screenshot displays the DEEP Dashboard's Marketplace section. The interface includes a dark sidebar on the left with navigation options: Dashboard, Modules (marketplace), Deployments, and OTHER LINKS (Identity and Access, DEEP Marketplace, DEEP Documentation, Project page). The main content area is titled 'Marketplace' and features a search bar. A grid of 12 modules is presented, each with a title, a brief description, and a list of capabilities (Model, Trainable, Inference). The modules are:

- Dogs breed detector**: Identify a dogs breed on the image (133 known breeds)
- DEEP OC Massive Online Data Streams**: Deep learning for proactive network monitoring and security protection.
- DEEP OC Retinopathy Test**: A Tensorflow model to classify Retinopathy.
- Train an image classifier**: Train your own image classifier with your custom dataset. It comes also pretrained on the 1K ImageNet classes.
- Plants species classifier**: Classify plant images among 10K species from the iNaturalist dataset.
- Conus species classifier**: Classify conus images among 70 species.
- Phytoplankton species classifier**: Classify phytoplankton images among 60 classes.
- Seed species classifier**: Classify seeds images among 700K species.
- Upscale multispectral satellites images**: Upscale (superresolve) low resolution bands to high resolution in multispectral satellite imagery.
- Speech keywords classifier**: Train a speech classifier to classify audio files between different keywords.
- Body pose detection**: Detect body poses in images.
- Train an audio classifier**: Train your own audio classifier with your custom dataset. It comes also pretrained on the 527 AudioSet classes.

# The Dashboard - Deploying a module

The screenshot shows the DEEP dashboard interface. The top navigation bar includes the DEEP logo, the user name 'Ignacio Heredia', and a dropdown arrow. The left sidebar contains navigation links: Dashboard, Modules (marketplace), Deployments, OTHER LINKS (Identity and Access, DEEP Marketplace, DEEP Documentation, Project page). The main content area shows the breadcrumb 'Home / Modules / deep-oc-image-classification-tf / train' and the title 'Configure training: Train an image classifier'. Below the title are two configuration sections: 'General Configuration' and 'Specific Configuration'. The 'General Configuration' section includes dropdowns for 'Template' (default), 'Command' (DEEPaaS), 'Hardware configuration' (CPU), and 'Docker tag' (latest). A note below the Docker tag dropdown states: 'You should choose the appropriate tag for your selected hardware. Check module documentation for more details if unsure.' The 'Specific Configuration' section includes 'Docker options' with a text input for 'Docker image to deploy from Docker Hub (docker\_image):' containing 'deephdc/deep-oc-image-classification-tf:latest' and a dropdown for 'Equivalent of --privileged docker flag (docker\_privileged):' set to 'False'. Below this is the 'Jupyter options' section with a text input for 'Password for JupyterLab. Should have at least 9 characters. (jupyter\_password):'.

## 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/imgclas/predict/ Make a prediction given the input data

**Parameters**

Name	Description
<b>data</b> file <i>(formData)</i>	Select the image you want to classify. <i>Default value : null</i>
<b>urls</b> string <i>(query)</i>	Select an URL of the image you want to classify. <i>Default value : null</i>
<b>timestamp</b> string <i>(query)</i>	Model timestamp to use for prediction. Group name: <b>testing</b> Choices: ['default_imagenet'] Type: str <i>Available values : "default_imagenet"</i> <i>Default value : "default_imagenet"</i>
<b>ckpt_name</b> string <i>(query)</i>	Checkpoint inside the timestamp to use for prediction. Group name: <b>testing</b> Type: str <i>Default value : "final_model.h5"</i>



**Response body**

```
{
  "status": "OK",
  "predictions": {
    "labels": [
      "lion",
      "fur_coat",
      "brown_bear",
      "hyena",
      "timber_wolf"
    ],
    "probabilities": [
      0.9763001203536987,
      0.005802886560559273,
      0.00480994675308466,
      0.0014703389024361968,
      0.001060070120729506
    ]
  }
}
```

# 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.

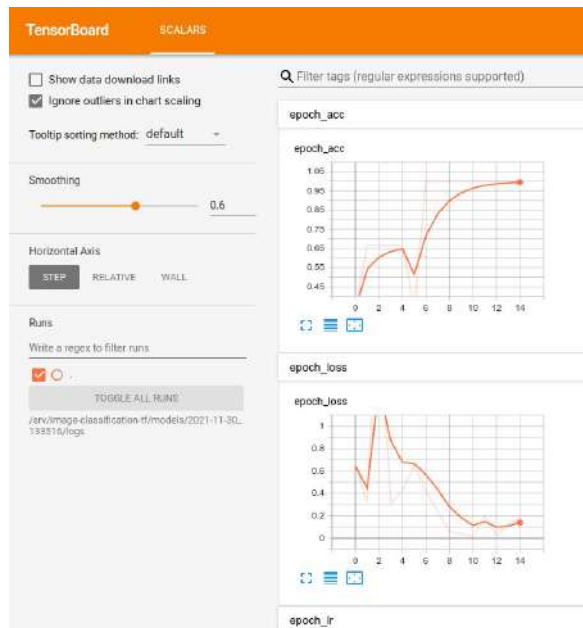
DEEPaaS

Monitor



Training history

POST	<code>/v2/models/imgclas/train/</code>	Retrain model with available data
GET	<code>/v2/models/imgclas/train/</code>	Get a list of trainings (running or completed)
GET	<code>/v2/models/imgclas/train/{uuid}</code>	Get status of a training



Home / Deployments / [11ec51e1-e308-f8be-81df-024250803cfb](#) / history

### 11ec51e1-e308-f8be-81df-024250803cfb details

imgclas Train model Predict

Training uuid	Date	Status	Message	Actions
9cde9418dbb14dc389d333926e7d5435	2021-11-30 13:35:16.621857	done		<span>Training arguments</span> <span>Delete</span>
b8203305b9024e73bed3ad0ff2a48ecb	2021-11-30 13:44:00.431724	done		<span>Training arguments</span> <span>Delete</span>
6f5320e2134a4d8dae20e8dd5db9451	2021-11-30 13:45:54.567297	running		<span>Training arguments</span> <span>Delete</span>

3

**Develop your module**

# Developing - DEEP Development Environment

## DEEP Development Environment

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

#### General Configuration

**Template:** default

**Command:** DEEPaaS

**Hardware configuration:** CPU

**Docker tag:** latest

You should choose the appropriate tag for your selected hardware. Check module documentation for more details if unsure.

#### Specific Configuration

**Docker options:**

**Docker image to deploy from Docker Hub (docker\_image):** deepcdc/deep-oc-generic-dev:latest

**Equivalent of --privileged docker flag (docker\_privileged):** False

**Jupyter options:**

**Password for JupyterLab. Should have at least 9 characters. (jupyter\_password):**

### 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**)

# 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

```
get_metadata()
```

```
get_train_args()
```

```
get_predict_args()
```

```
schema
```

```
train()
```

```
predict ()
```

```
warm()
```

→ 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>

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

→ Example: [https://github.com/deephdc/demo\\_app/blob/master/setup.cfg#L25-L27](https://github.com/deephdc/demo_app/blob/master/setup.cfg#L25-L27)

# Developing - Customizing the Dockerfile



- Head over to `DEEP-0C-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>
  - `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)

→ Example: <https://github.com/deephdc/DEEP-OC-image-classification-tf/blob/master/Jenkinsfile>

# 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!**



4

**What's next?**

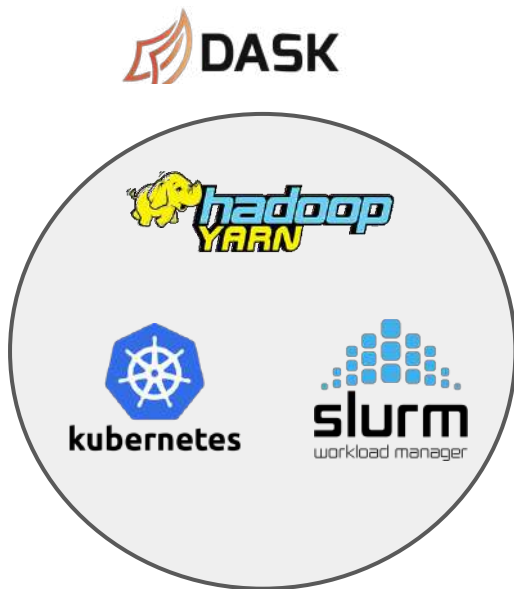
# What's next? - New DEEPaaS features

- Integration with Dask

Mature

- Easier module integration via decorators/hints

Midterm



Before (webargs)

```
3 from webargs import fields, validate
4
5 def get_predict_args():
6     arg_dict = {
7         "demo-str": fields.Str(
8             required=False,
9             missing='some-string',
10        ),
11        "demo-int": fields.Int(
12            required=False,
13            missing=1,
14        ),
15    }
16
17
18 schema = {
19     "demo-list": fields.List(
20         fields.Float()
21     ),
22 }
23
24
25 def predict(**kwargs):
26     return {"demo-list": [1, 2, 3]}
27
```

After (type hints)

```
32 def predict("demo-str": str,
33            "demo-int": int,
34            ) -> dict:
35     return {"demo-list": [1, 2, 3]}
36
```

# What's next? - Friendlier inference UI



Before (Swagger UI)

Swagger UI for demo\_app. Endpoints include GET /v2/models/demo\_app/train/{uuid}, DELETE /v2/models/demo\_app/train/{uuid}, and POST /v2/models/demo\_app/predict/. The POST endpoint has parameters: demo-str (string, some-string), demo-str-choice (string, choice2), demo-int (integer, 1), demo-image (file, image), demo-audio (file, audio), and demo-video (file, video). An 'Execute' button is at the bottom.

Inputs

After (Gradio based)

Mature

demo\_app

A minimal toy application for demo and testing purposes. We just implemented dummy inference, ie. we return the same inputs we are feed.

Gradio-based demo\_app interface. It features interactive input fields for: DEMO-STR (some-string), DEMO-STR-CHOICE (choice2), DEMO-INT (1), DEMO-INT-RANGE (50), DEMO-FLOAT (0.1), DEMO-BOOL (checked), DEMO-DICT ({"a": 0, "b": 1}), DEMO-LIST-OF-FLOATS (0.1, 0.2, 0.3), and DEMO-IMAGE (image with Edit button). On the right, there are preview areas for DEMO-IMAGE, DEMO-INT-RANGE (50), DEMO-VIDEO (video player), and DEMO-AUDIO (audio player).





# 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

Midterm

Project

# Questions



European  
Commission

Horizon 2020  
European Union funding  
for Research & Innovation



**CSIC**  
Spanish Council of Research

DEEP-Hybrid-DataCloud has received funding from the European Horizon 2020 research and innovation programme under grant agreement N°777435