DEEP-HybridDataCloud

First prototype of the DEEP as a Service

Deliverable: D6.3

Document identifier:	DEEP-JRA3-D6.1
Date:	03/12/2018
Activity:	WP6
Lead partner:	CSIC
Status:	Final
Dissemination level:	Public
Permalink:	http://digital.csic.es/handle/10261/168088

Abstract

This document provides an updated description of the prototype implementation of the DEEP as a Service solution that is being developed within the DEEP-Hybrid-DataCloud project Work Package 6 (WP6). As such it provides an overview of the state of the art of the relevant components and technologies, as well as a technology readiness level assessment with regards to the required functionality, the required interactions with other work packages in the project, as well as the detailed work plan and risk assessment for each of the activities.

Copyright Notice

Delivery Slip

	Name	Partner/Activity	Date
From	Álvaro López García	CSIC / WP6	0,/13,2018
Reviewed by	Marcin Płóciennik Alessandro Costantini	PSNC INFN	23/11/2018 01/12/2018
Approved by	Steering Committee		03/12/2018

Document Log

Issue	Date	Comment	Author/Partner
V1.0	06/10/2018	ToC	Álvaro López / CSIC
V2.0	06/10/2018	User requirements relevant for WP6	Álvaro Lopez / CSIC
V3.0	10/10/2018	Alien4Cloud sections	Andy S. Alic / UPV German Moltó / UPV
V4.0	15/10/2018	DEEPaaS API sections	Álvaro López / CSIC
V5.0	20/11/2018	Prototype implementation description	WP members
V5.0	22/11/2018	User requirements relevant for WP6	Álvaro Lopez / CSIC
V7.0	26/11/2018	Review	Marcin Płóciennik / PSNC
V8.0	01/12/2018	Review	Alessandro Costantini / INFN
V8.0	03/12/2018	Changes from review	Álvaro López / CSIC

Table of Contents

Executive Summary

DEEP Hybrid DataCloud (Designing and Enabling E-Infrastructures for intensive data Processing in a Hybrid DataCloud) is a project approved within the EINFRA-21-2017 call of the Horizon 2020 framework program of the European Community. It started with the global objective of promoting the usage of intensive computing services and techniques by different research communities and areas, and their support by the corresponding e-Infrastructure providers and open source projects. The project will integrate and enhance existing components in the cloud (and more specifically in the EOSC) ecosystem, developing an innovative service supporting intensive computing techniques that require access to specialized hardware, such as GPUs or low-latency interconnects, to explore very large datasets. The DEEP as a Service component will provide, in the form of the DEEP catalogue, a set of models and applications ready for reuse and a collection of predefined building blocks, in the form of software assets, that can be composed in order to build complex applications to be transparently executed by the DEEP users on top of the underlying resources.

This document provides an update on the status of the first prototype of the DEEP as a Service solution with the specific implementations and integration activities that have been carried out during the first prototype phase of the project (we provide an updated version of the detailed component description already provided in D6.2 - Design for the DEEP as a Service solution [D6.2]). In this regard, we have developed the needed modules to integrate the Alien4Cloud TOSCA composition tool with the INDIGO-DataCloud orchestrator, providing a seamless tool to compose and submit applications and topologies to the PaaS layer. In close collaboration with WP2, and in order to expose the user applications to other users, we are leveraging the DEEPaaS API component that provides a uniform API for application developers to consume, as well as an intuitive user interface to be exploited by the users. We have integrated all these components (TOSCA topologies, user applications and the DEEPaaS API) under the DEEP Open Catalogue, as a set of ready to use modules that can be exploited on top of any existing infrastructure.

1. Introduction

1.1. Document purpose

This document reports on the work carried out in WP6 towards the delivery of the first prototype of the DEEP as a Service solution. As such, this documents complements D6.2 - Design for the DEEP as a Service solution [D6.2], providing an update on the component status, the work and the implementations that have been carried out for this first prototype.

This initial prototype has mainly focused on the batch task execution, so that a user application is trained and tuned using specialized hardware such as GPUs (as described in deliverable D6.2 - Design for the DEEP as a Service solution). During this initial phase, WP6 has worked closely with WP2 so that the user requirements, initially collected in D2.1 - Initial Plan for Use Cases [D2.1] and further refined in JIRA [JIRA-DEEP], were satisfied by the developed solutions. As such, the work carried out has consisted on the following points:

- Providing users with a **graphical application modeller** (Alien4Cloud) for the composition and execution of complex applications.

- Development of an **INDIGO Orchestrator plugin for Alien4Cloud** as the runtime engine to be used for the transparent deployment of an application.
- Development of a **generic REST API component** (DEEPaaS API) that would expose the application functionality as a web service with minimal modifications for the use cases.
- **Integration of the existing use cases** with the aforementioned tools (Alien4Cloud and DEEPaaS).
- Building **TOSCA templates and Ansible roles** that define the user application to be executed.
- Elaborating the **initial contents of the DEEP Open Catalogue**, consisting on Docker containers and TOSCA templates to be reused.

However, the work carried out during this first phase has not only covered the batch execution (training) of the user applications, but also its deployment as a service. As a matter of fact, through the DEEPaaS API, users are already able to expose their models as services accessible through an HTTP endpoint. Moreover, we have started the initial integration of the DEEPaaS component with a server less framework as a pre-alpha internal preview. Exploiting this framework we have also demonstrated the feasibility of the integration with external storage solutions (coming from the eXtreme-DataCloud project) for automated event-driven processing.

2. DEEP as a Service description

2.1. Overview of components and modules

The high level decomposition of the WP6 DEEP as a Service design was already described in D6.2 - Design for the DEEP as a Service solution and is depicted here again in Figure 1, consisting on the following key components:

- The DEEP open Catalogue where the users, communities, etc. can browse, store and download relevant modules for building up their applications (like ready to use machine learning frameworks, complex application topologies, etc.).
- An application modeller or composition tool, that will be used to build up complex application topologies in an easy way.
- A runtime engine, that will take the defined topology as input, provision the required computing resources and deploy the application.
- The DEEP PaaS layer, that will coordinate the overall workflow execution to choose the appropriate Cloud sites and manage the deployment of the applications to be executed.
- The DEEP as a Service solution, that will offer the application functionality to the user.
- The EINFRA/EOSC data services, to be integrated with the DEEP solutions in order to provide access to any of the data facilities existing in the European Open Science Cloud

The system is designed with extensibility in mind, taking great care in designing a framework which can be updated easily and where a component can be replaced with a new one in case it is needed. Many of the anticipated changes to our system in future phases will only require adding additional functionality on top of existing components, remaining backwards compatible with previous versions.

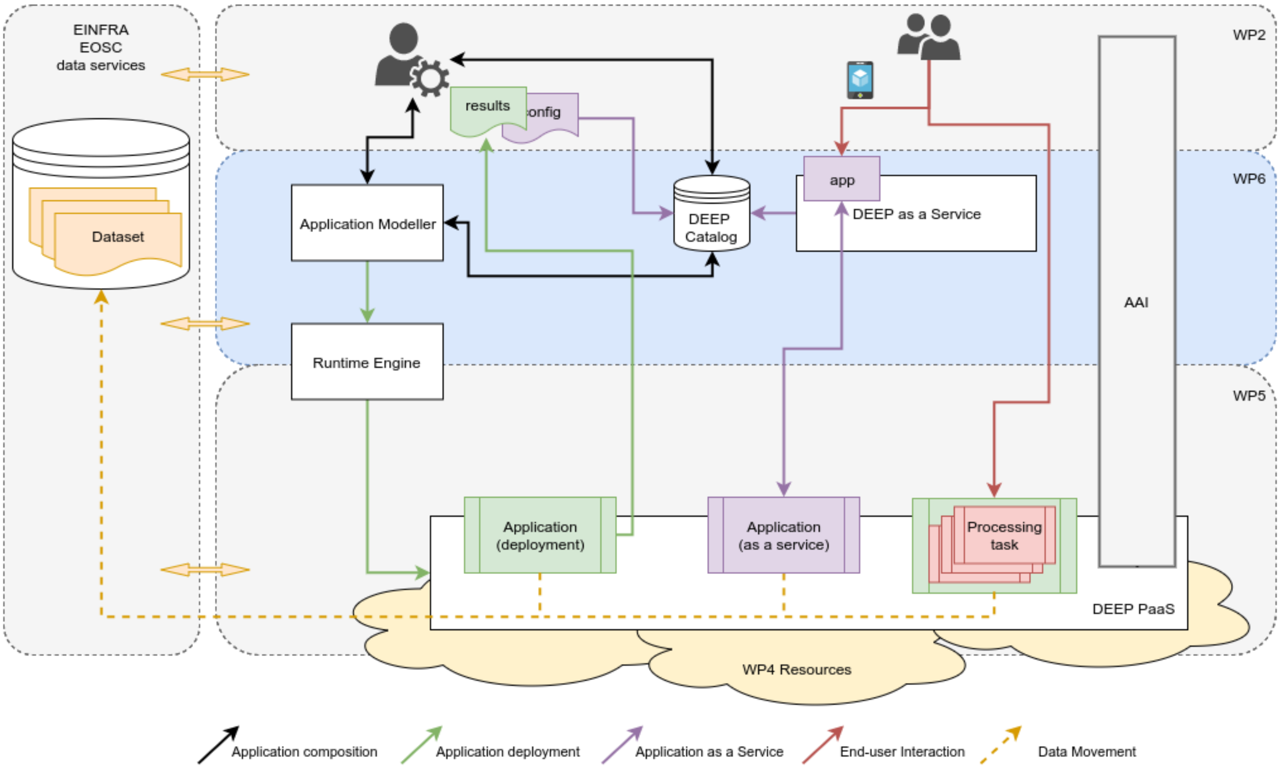


Figure 1: WP6 high level architecture

2.2. Structure and relationships

The high level decomposition of the WP6 DEEP as a Service design has been already described in D6.2 - Design for the DEEP as a Service solution and it follows the design already depicted in Figure 1. Focusing only on interactions within WP6, we can define the following interactions:

- DEEP catalogue.
  - Holds the defined application building blocks.
  - The catalogue interacts with the Application modeller in order to expose any existing building blocks.
  - The catalogue allows the storage and retrieval of the building blocks either from the application modeller, the DEEP as a Service component, or directly by the users.
  - Users can interact with the catalogue to browse, download and upload any module.
  - Interaction with EOSC/EINFRA data services is expected for storing large datasets.
  - Requires user authentication and authorization for writing purposes. Anonymous read-only access is possible.

- Application modeller.
    - Provides functionality to compose several building blocks into a more complex application.
    - Provides functionality to deploy and execute the applications in the underlying PaaS.
    - The application modeller allows to load the pre-defined components from the catalogue
    - The application modeller allows to create new components, and to store them on the catalogue.
    - Interaction with other EINFRA-21 or EOSC-related data services is expected for storing large datasets and referencing them on the built applications.
    - Requires user authentication and authorization for writing purposes. Requires user authentication and authorization for deploying and executing the user applications. Anonymous read-only access is possible.
- DEEP as a Service.
    - Provides application builders with the possibility of deploying an application as a service in order to use the application functionality.
    - Allows to deploy the application without worrying about the underlying resources.
    - Final users can access this service in order to get the provided functionality.
    - Interacts with the catalogue in order to retrieve the required module(s) to be deployed as a service, as long as any configuration parameters.
    - Interaction with EOSC/EINFRA data services may be needed.
    - Requires user authentication and authorization for writing purposes and to create a new application service. Anonymous read-only access may be possible.
- Runtime Engine
    - Deploys an application topology using the underlying DEEP PaaS resources.
    - Interacts with the application modeller in order to receive the application topology and deploy it. Also provides information about the resources and application status.
    - Users will not interact directly with this component, although advanced users will be able to do so.
    - Requires user authentication and authorization.

According to previous analysis of the alternatives, and as we already discussed in D6.2 - Design for the DEEP as a Service solution [D6.2], in order to express the application topologies that will be deployed by WP5 we are leveraging the usage of the TOSCA open standard [TOSCA 2017].

3. User requirements relevant to WP6

The initial collection of user requirements has been performed in D2.1 - Initial Plan for Use Cases [D2.1] and it was further refined using the JIRA tool [JIRA-DEEP] through several iterations with the community representatives The functional and non functional requirements relevant for WP6 are listed in the Table 1.

Key	Summary	Epic Link
DPD-106	Allow constraint for deployment to a specific datacentre	MODS
DPD-255	Build a catalogue of modules	MODS
DPD-227		Plant classification
DPD-224		Deep Learning for satellite monitoring
DPD-248	Expose model functionality to end users	MODS
DPD-242		Retinopathy image classification
DPD-213		Plant classification
DPD-211		Deep Learning for satellite monitoring
DPD-74		Post-processing of QCD simulations
DPD-246	Facilitate the deployment of a trained model for inference	MODS
DPD-240		Retinopathy image classification
DPD-210		Plant classification
DPD-209		Deep Learning for satellite monitoring
DPD-73	Facilitate the deployment of an application	Post-processing of QCD simulations
DPD-251	Facilitate the non-skilled deployment of an existing application	MODS
DPD-218		Plant classification
DPD-217		Deep Learning for satellite monitoring
DPD-252	Provide a containerized version of the application	MODS
DPD-221		Plant classification
DPD-249	Provide a easy way to access a deployed application by end users	MODS
DPD-243		Retinopathy image classification
DPD-214		Plant classification
DPD-212		Deep Learning for satellite monitoring

Key	Summary	Epic Link
DPD-241	Provide a way for the user to keep track of the training status	MODS
DPD-235		Retinopathy image classification
DPD-206		Plant classification
DPD-204		Deep Learning for satellite monitoring
DPD-72		Post-processing of QCD simulations
DPD-260	Provide a way to compose the application from more modules using predefined configuration or settings	MODS
DPD-258	Provide a way to configure deployed components separately	MODS
DPD-261	Provide a way to redeploy existing application topology.	MODS
DPD-254	Provide a way to share a user applications	MODS
DPD-247		Retinopathy image classification
DPD-225		Plant classification
DPD-222		Deep Learning for satellite monitoring
DPD-76		Post-processing of QCD simulations
DPD-256	Provide a way to submit input data to model	MODS
DPD-230	Provide a web interface for the application	Plant classification
DPD-253	Provide an UI for applications	MODS
DPD-223		Plant classification
DPD-220		Deep Learning for satellite monitoring
DPD-77	Provide authenticated and authorized access	Post-processing of QCD simulations
DPD-257		MODS
DPD-228		Plant classification
DPD-226		Deep Learning for satellite monitoring
DPD-78	Provide authenticated and authorized access to monitoring	Post-processing of QCD simulations
DPD-259	Provide monitoring history to all endpoints and services	MODS
DPD-239	Provide resources to efficiently retrain a model with code modification	MODS
DPD-233	Provide resources to efficiently retrain a model with code modification	Retinopathy image classification
DPD-205		Plant classification

Key	Summary	Epic Link
DPD-203		Deep Learning for satellite monitoring
DPD-238	Provide resources to efficiently train a model	MODS
DPD-202		Plant classification
DPD-201		Deep Learning for satellite monitoring
DPD-231	Provide with an automated way to create a smartphone app with the trained application	Plant classification
DPD-250	Support the redeployment of an application	MODS
DPD-245		Retinopathy image classification
DPD-216		Deep Learning for satellite monitoring
DPD-215		Plant classification
DPD-75		Post-processing of QCD simulations
DPD-244	The trained model must be stored permanently and must be available	MODS
DPD-237		Retinopathy image classification
DPD-208		Plant classification
DPD-207		Deep Learning for satellite monitoring

Table 1: WP2 requirements relevant for WP6.

4. Detailed component description

The template shown below is used as the structure to include a complete description of each component in the following submissions. Deliverable D6.2 - Design for the DEEP as a Service solution already included a detailed component description for all the services and components involved in this work package, therefore the original components that are not involved in this prototype are not described here. This deliverable includes an updated template for the specific services that have been improved to support the prototype implementation of DEEP as a Service solution. In this regard, an additional row (Implemented improvements for the prototype) has been included to clearly indicate the implementations carried out. The information in this row is provided as a summary, while section 4 provides additional technical details.

Identification	The unique name for the component and its location in the system
Type	A module, a subprogram, a data file, a control procedure, a class, etc.
Purpose	Function and performance requirements implemented by the design component, including derived requirements. Derived requirements are not explicitly stated in the SRS, but are implied or adjunct to formally stated SDS requirements.
Function	What the component does, the transformation process, the specific inputs that are processed, the algorithms that are used, the outputs that are produced, where the data items are stored, and which data items are modified.
High level architecture	The internal structure of the component, its constituents, and the functional requirements satisfied by each part.
Dependencies	How the component's function and performance relate to other components. How this component is used by other components. The other components that use this component. Interaction details such as timing, interaction conditions (such as order of execution and data sharing), and responsibility for creation, duplication, use, storage, and elimination of components.
Interfaces	Detailed descriptions of all external and internal interfaces as well as of any mechanisms for communicating through messages, parameters, or common data areas. All error messages and error codes should be identified. All screen formats, interactive messages, and other user interface components (originally defined in the SRS) should be given here.
Data	For the data internal to the component, describes the representation method, initial values, use, semantics, and format. This information will probably be recorded in the data dictionary.
Needed improvement	Description of the needed improvements of this tool with regards the DEEP-Hybrid-DataCloud objectives, in order to fulfil the user requirements and to build the DEEP as a Service functionality.

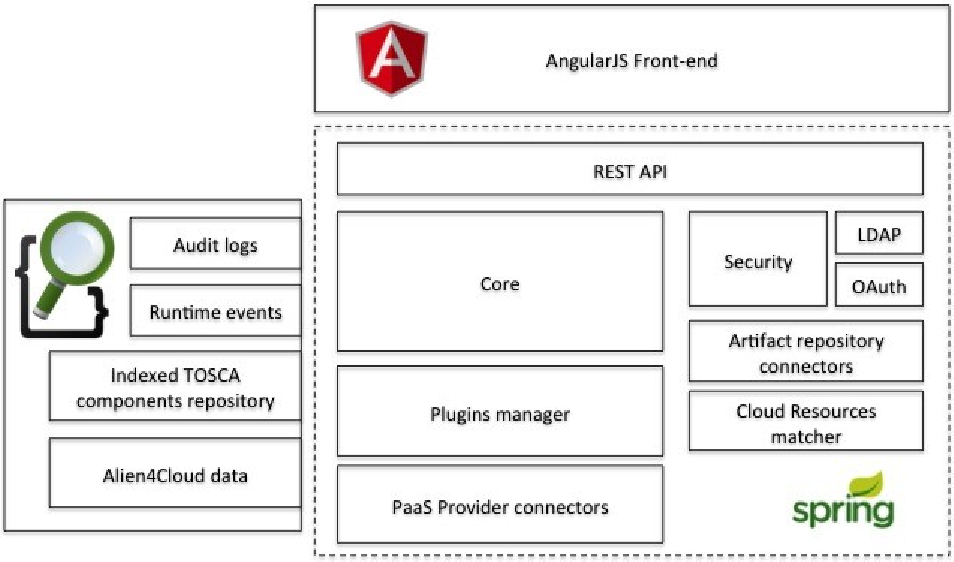
Current TRL status	A detailed description of the status of the Technology Readiness Level, for specific set of features of the services.
Expected TRL evolution	A detailed description of the Technology Readiness Level that DEEP foreseen to reach for a specific set of features of the services.
Implemented Improvements for the Prototype	A description of the implemented improvements in the service in order to support the prototype for hybrid Cloud deployments.

4.1. Application composition

<table>
<tr><th>Identification</th><td>Alien4Cloud</td></tr>
<tr><th>Type</th><td>A Java web-based application</td></tr>
<tr><th>Purpose</th><td>Alien4Cloud is an application that allows people in the enterprise to collaborate in order to provide self-service deployment of complex applications taking in account the different experts through a role based portal by means of application architectures described in the TOSCA Simple Profile in YAML specification.

With this component we partially implement the following requirements.
<table>
<tr><th>Req. key</th><th>Summary</th></tr>
<tr><td>DPD-106</td><td>Allow constraint for deployment to a specific datacentre.</td></tr>
<tr><td>DPD-246</td><td rowspan="4">Facilitate the deployment of a trained model for inference.</td></tr>
<tr><td>DPD-240</td></tr>
<tr><td>DPD-210</td></tr>
<tr><td>DPD-209</td></tr>
<tr><td>DPD-251</td><td rowspan="3">Facilitate the non-skilled deployment of an existing application</td></tr>
<tr><td>DPD-218</td></tr>
<tr><td>DPD-217</td></tr>
<tr><td>DPD-260</td><td>Provide a way to compose the application from more modules using predefined configuration or settings</td></tr>
<tr><td>DPD-258</td><td>Provide a way to configure deployed components separately</td></tr>
<tr><td>DPD-261</td><td>Provide a way to redeploy existing application topology.</td></tr>
<tr><td>DPD-254</td><td rowspan="4">Provide a way to share a user applications</td></tr>
<tr><td>DPD-247</td></tr>
<tr><td>DPD-225</td></tr>
<tr><td>DPD-222</td></tr>
<tr><td>DPD-239</td><td rowspan="4">Provide resources to efficiently retrain a model with code modification</td></tr>
<tr><td>DPD-233</td></tr>
<tr><td>DPD-205</td></tr>
<tr><td>DPD-203</td></tr>
<tr><td>DPD-238</td><td rowspan="3">Provide resources to efficiently train a model</td></tr>
<tr><td>DPD-202</td></tr>
<tr><td>DPD-201</td></tr>
<tr><td>DPD-250</td><td>Support the redeployment of an application</td></tr>
</table></td></tr>
<tr><th>Function</th><td>Alien4Cloud consists on a web-based application that supports i) the definition of multiple user accounts with different roles; ii) importing multiple TOSCA types as building blocks; iii) the composition of TOSCA</td></tr>
</table>

<table>
<tr><td></td><td>templates by means of said building blocks; iv) the deployment of TOSCA templates to a TOSCA runtime engine (Cloudify and Puccini) are supported, and Mesos; v) the life cycle management of said deployment.</td></tr>
<tr><td><b>High level architecture</b></td><td>The internal architecture of Alien4Cloud is described in the following picture (credit: https://alien4cloud.github.io/#/developer_guide/internal-architecture.html):



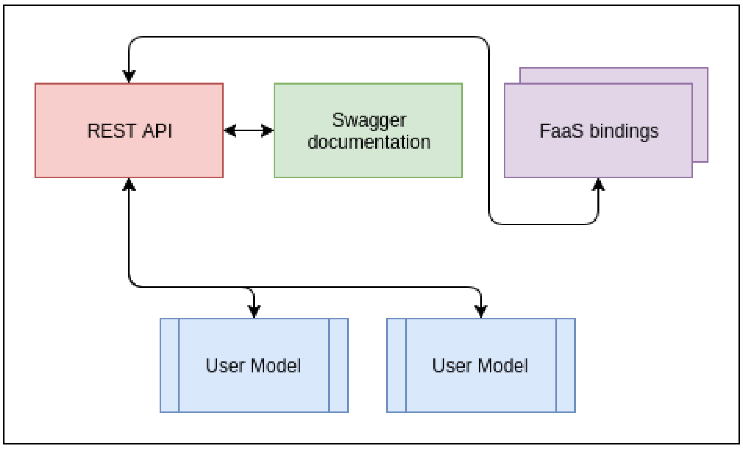
ALIEN is an AngularJS (frontend) + Spring (backend) web-application. Plugins for ALIEN are managed as singular Spring applications context that all share the same parent context (ALIEN core application context). Each plugin uses it’s own class loader to ensure that they don’t collide with each others.</td></tr>
<tr><td><b>Dependencies</b></td><td>Alien4Cloud depends on a Java Runtime Environment (JRE) to deploy the web-based application. It also depends on a set of TOSCA types that are required to perform the composition of TOSCA templates. Finally, it depends on a TOSCA runtime engine, also known as Orchestrator, to perform the provision and deployment of virtual infrastructures on multiple Cloud sites.</td></tr>
<tr><td><b>Interfaces</b></td><td>Alien4Cloud supports a web-based graphical user interface to interact with the software (user management, TOSCA types importing, topology composition, deployment, etc.). It also supports a fully featured REST API to interact with: users, plugins, orchestrators, metaproperties, catalogue, workspaces, applications, applications deployment, topology editor, administrator. The REST API is described in https://alien4cloud.github.io/roadmap/index.html#/documentation/2.0.0/rest/overview.html</td></tr>
<tr><td><b>Data</b></td><td>Data is used at multiple levels in Alien4Cloud but the main focus is set on the Catalogue.
Alien 4 Cloud TOSCA Catalogue is an index of components/elements</td></tr>
</table>

	defined in a TOSCA archive. Among these elements we find two main categories Types (reusable building blocks) and Topologies (Composition and definition of the building blocks to define what a user wants to deploy). When adding or creating a TOSCA archive in Alien 4 Cloud the archive is automatically store on a File System but also indexed to provide browsing and search features. In addition to the types and topology in an archive Alien4Cloud also indexes an object that represents the archive and its meta-data. This is referenced in alien as the CSAR (for Cloud Service ARchive).
Needed improvement	Alien4Cloud is adopted in DEEP-Hybrid-DataCloud as the tool to facilitate the composition of TOSCA Templates out of the TOSCA Types. To this aim, three improvements are required. First, the TOSCA Types that were created in the INDIGO-DataCloud project that already support multiple scientific applications and which are used in the INDIGO-DataCloud PaaS Orchestrator, will be adapted to be used in Alien4Cloud. This involves minor adjustments in the syntax and attributes employed to allow seamless compatibility between Alien4Cloud and the PaaS Orchestrator. Second, a plugin will be developed in order to integrate the INDIGO-DataCloud PaaS Orchestrator as one of the orchestrators supported in Alien4Cloud. Fortunately, the plugin system available in Alien4Cloud will allow to easily modularise the creation of the plugin without requiring to maintain a separate version of Alien4Cloud and push the changes upstream. Finally, support for OpenID Connect should be integrated in order to foster its integration with the IAM solutions available in EOSC.
Current TRL status	Alien4Cloud can be currently considered in TRL 8. The latest version released is 2.0.0 and the web page describes several success stories highlighting its adoption across multiple use cases involving several underlying computing platforms (IaaS Cloud sites and Kubernetes clusters). It is actively developed in GitHub.
Expected TRL evolution	Alien4Cloud will maintain the TRL 8. The plugins developed will be developed and assessed through the proper Software Quality Assurance requirements defined by the project to achieve a similar Technology Readiness Level.
Implemented Improvements for the Prototype	• Bug fixing and adaptation of A4C to better comply with TOSCA standard. • Implementation of INDIGO Orchestrator plugin. • Initial work on AAI integration. • Implemented A4C settings tool.

4.2. DEEP as a Service API

Identification	DEEPaaS API
Type	A Python REST API implementation
Purpose	The main function of the DEEPaaS component is to expose user model

<table>
<tr>
<td></td>
<td>through a REST API. With this component we partially implement the following requirements.
<table>
<tr><th>Req. key</th><th>Summary</th></tr>
<tr><td>DPD-106</td><td>Allow constraint for deployment to a specific datacentre</td></tr>
<tr><td>DPD-246</td><td rowspan="4">Facilitate the deployment of a trained model for inference</td></tr>
<tr><td>DPD-240</td></tr>
<tr><td>DPD-210</td></tr>
<tr><td>DPD-209</td></tr>
<tr><td>DPD-251</td><td rowspan="3">Facilitate the non-skilled deployment of an existing application</td></tr>
<tr><td>DPD-218</td></tr>
<tr><td>DPD-217</td></tr>
<tr><td>DPD-260</td><td>Provide a way to compose the application from more modules using predefined configuration or settings</td></tr>
<tr><td>DPD-258</td><td>Provide a way to configure deployed components separately</td></tr>
<tr><td>DPD-261</td><td>Provide a way to redeploy existing application topology.</td></tr>
<tr><td>DPD-254</td><td rowspan="4">Provide a way to share a user applications</td></tr>
<tr><td>DPD-247</td></tr>
<tr><td>DPD-225</td></tr>
<tr><td>DPD-222</td></tr>
<tr><td>DPD-239</td><td rowspan="4">Provide resources to efficiently retrain a model with code modification</td></tr>
<tr><td>DPD-233</td></tr>
<tr><td>DPD-205</td></tr>
<tr><td>DPD-203</td></tr>
<tr><td>DPD-238</td><td rowspan="3">Provide resources to efficiently train a model</td></tr>
<tr><td>DPD-202</td></tr>
<tr><td>DPD-201</td></tr>
<tr><td>DPD-250</td><td rowspan="4">Support the redeployment of an application</td></tr>
<tr><td>DPD-245</td></tr>
<tr><td>DPD-216</td></tr>
<tr><td>DPD-215</td></tr>
</table>
</td>
</tr>
<tr>
<td><b>Function</b></td>
<td>The DEEPaaS API is intended to expose the user's model functionality by means of a REST API, so that the most common actions (e.g. train, predict, evaluate) can be done over this HTTP endpoint. DEEPaaS interfaces with the user application by loading the user defined entry points within a common namespace, so that minimal modifications are required in the user application. When DEEPaaS is started, it fetches and loads the existing entry points, exposing them in their own namespace (thus several models can be loaded at the same time).</td>
</tr>
</table>

High level architecture	 REST API Swagger documentation FaaS bindings User Model User Model
Dependencies	DEEPaaS has minimal dependencies on Python libraries. There are no special services or components required.
Interfaces	DEEPaaS exposes a REST endpoint based on the OpenAPI specification [OpenAPI], with self-documented and machine readable interfaces.
Data	DEEPaaS is a stateless component, although the underlying models that DEEPaaS will load consume data in different ways. For the training operations, the underlying models require fast efficient access to the data to be used, as well as a data storage to save the results, configurations, etc. For the inference operations, the data is being send (both input and output data) through the HTTP endpoint (requests are synchronous). Asynchronous requests will be implemented in the future, thus data requirements may change.
Needed improvement	DEEPaaS is developed to expose the user model functionality towards end users. This component needs to evolve according to the user needs, fulfilling their requirements, but at the same time it is needed that it remains generic to all the use cases so that one single component can be used for a large number of use cases. The specific improvements that we have collected up to date are the following: • Implement generic and dynamic loading mechanism, so that users do not need to plug manually their model into the API • Implement factory-based loading of models to provide even more dynamic loading mechanisms • Integration with authentication and authorization infrastructures (AAI), namely OpenID Connect. • Ability to expose several models through the same API endpoint. • Ability to specify, with a larger granularity, the parameters that are exposed from the underlying model through the API. • Ability to provide monitoring information of the training status.
Current TRL	TRL6. Currently it is being used in the pilot preview testbed by the DEEP

status	use cases
Expected TRL evolution	TRL8 during second half of the project.
Implemented Improvements for the Prototype	• Dynamic loading of modules, through configurable entry-points. • Ability to load and expose several models into the API namespace, instead of only one. • Implementation of training functionality, metadata fetching and inference/prediction. • Initial prototype (pre-alpha) implementation of integration with server less frameworks.

5. Prototype implementation of the DEEP as a Service solution

In the following section we elaborate on the developments carried out in the aforementioned tools, so that the user requirements were fulfilled form WP6 perspective.

5.1. Alien4Cloud bundle

Alien4Cloud has been elected as the tool able to provide an easy to use and intuitive application composition (see details in D6.2 - Design for the DEEP as a Service solution). We offer Alien4Cloud as part of a bundle [A4C-DEEP] contained in a Docker image, ready to be used without any compilation and installation steps. This bundle contains three different important parts, that are described here: the **Alien4Cloud application**, the **orchestrator plugin** that allows Alien4Cloud to communicate with the INDIGO-DataCloud orchestrator, and a **settings manager** specifically designed to handle the container's environment variables needed for the Alien4Cloud settings. The Alien4Cloud bundle is automatically released through DockerHub following the automated process setup by the project for all the software development activities, as set up by WP3 and available at https://jenkins.indigo-datacloud.eu:8080/view/DEEP/.

We based our developments in the latest stable version of Alien4Cloud, version 2.0.0 as, at the time of writing, the development version, 2.1.0, was not considered stable. The adoption of a newer version of Alien4Cloud will be considered as soon as it will be released. Moreover, we identified several issues and improvements in the official code repository (see Table 2) that would need to be tackled. In order to improve the official product, we fixed some of the issues and submitted upstream merge requests that are still under review by the Alien4Cloud developers. In order to get around the problem and continue our developments, we decided to maintain our own fork of Alien4CLoud [A4C-DEEP] with the aim to merge all the implemented modifications into the upstream repository.

ID	Title	Link
#167	Version 2.0.0 no outputs for Attributes	https://github.com/alien4cloud/alien4cloud/issues/167
#165	Plugin from init dir replaced when a4c restarted	https://github.com/alien4cloud/alien4cloud/issues/165
#148	get_attribute error	https://github.com/alien4cloud/alien4cloud/issues/148
#143	Outputs YAML errors	https://github.com/alien4cloud/alien4cloud/issues/148

Table 2: List of A4C open issues from DEEP.

5.1.1.Alien4Cloud

Figure 2 shows the internal architecture of Alien4Cloud. There are three main components that work together: **the web client** (the view), **the web server** (the controller), and **the data storage** (the model). All these are packed in a self-executable WAR file. We present an overview of the aforementioned components in the following paragraphs.

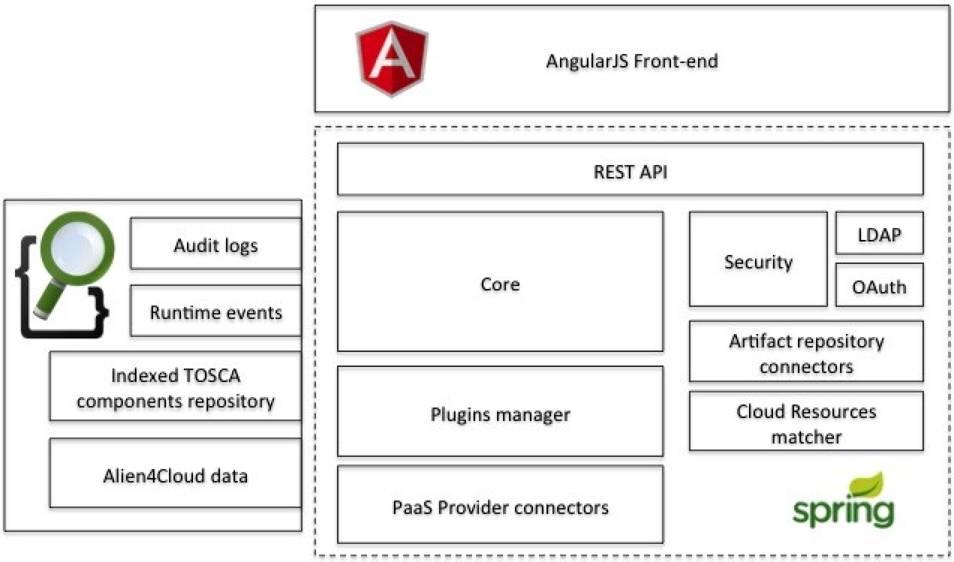
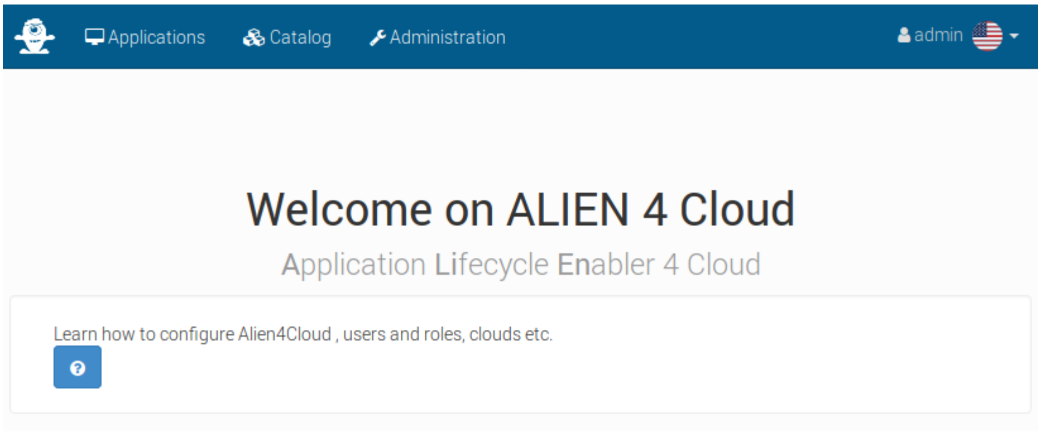


Figure 2: Alien4Cloud internal architecture.

The web client is built around AngularJS 1. It allows the user to interact with the web server. There are three main features (as seen in Figure 3) grouped in the "Applications", "Catalog", and "Administration" menus. The first menu allows the user to create and manage applications. These applications can be edited and then launched on different instances of the orchestrators supported by Alien4Cloud (through plugins). Users can check the available components, add new, or remove existing ones through the Catalog UI. Alien4Cloud supports uploading CSARs directly through its interface, as well as specifying the Github repository where the TOSCA definitions are kept.



Welcome on ALIEN 4 Cloud

Application Lifecycle Enabler 4 Cloud

Learn how to configure Alien4Cloud , users and roles, clouds etc.

Figure 3: Alien4Cloud main page

The web server consists of different components, with one of the most important being the REST API. It is used by the Alien4Cloud web client (running in a browser) and it can also be used directly by the user. The core is the main part, being responsible for the TOSCA parsing, application management (like keeping track of the created applications, stored into the ElasticSearch storage and deployment), Alien4Cloud management (like server information, starting up the application, and loading the pre-installed plugins and TOSCA templates), etc. The plugin manager represents the interface made available to the developers that allows the creation of external plugins. We use this component for our own plugin, as described in the next section. Another important part of the web server is the security component. It deals with the authentication (local, but also involved in the LDAP and OAuth).

The data storage saves and holds the persistent information such as user-created applications, settings, and TOSCA components. Alien4Cloud implements two methods of storage: plain and ElasticSearch. The former holds plugins, CSARs and other data sources that don't necessarily need indexing and a DB engine (at least for the data sources themselves, they can be unpacked and indexed though to improve performance). Our plugin is stored in the plain storage, in the init directory, and then loaded by Alien4Cloud during startup. The former storage deals with information like users and available applications, information that lends itself very well to this type of storage.

5.1.2.Orchestrator plugin

The second component present in the Docker image is the actual plugin that enables the communication between Alien4Cloud and the DEEP orchestrator. This plugin is compiled when the image is built, and then installed in the Alien4Cloud's *init* directory. We use the same *init* directory to install the TOSCA custom types defined in the INDIGO-DataCloud project. The plugin allows configuration through a number of parameters, modifiable when an instance of it is created. We explain the use of these parameters in the README file found in the Github repository at [A4C-DEEP]. Our users have to set the client ID and client secret, because each new instance of Alien4Cloud should have a different pair. An IAM instance requires these values (along with the user name and the password) to generate a token that allows the user to deploy with the orchestrator.

At the time of writing, the administrator has to create local users in the Alien4Cloud instance that have the same name and password as the ones registered with IAM. As a temporary solution, in order to allow the authentication during the application deployment, we had to modify the original code of Alien4Cloud to store the user's password and not just its hash. It is a temporary solution for our users, and we already are in the initial phases of the development of a feature/plugin that would authenticates the user against an IAM instance. Once the authentication component development is completed, the plugin can be installed on any Alien4Cloud instance that was compiled from the original source code. This is one of our most important goals, the compatibility with the original source code.

The plugin has three main functions:

- Manage the life cycle of a deployment (deploy, undeploy, get status)
    - When a user presses the deploy button, the plugin takes charge
    - The plugin sends the topology and the additional necessary information (authentication token , input parameters etc.) to the DEEP orchestrator
    - It asks for the status of the deployment through the DEEP orchestrator API
        - Each time the Alien4Cloud web page is refreshed
        - We plan to implement automatic status updates in the near future
    - It sends the undeploy command
- Convert between Alien4Cloud's TOSCA variant and the DEEP orchestrator's
    - Some elements the Alien4Cloud specific TOSCA have to be converted
        - e.g. the imports section has to use an external file not a version as it is used by Alien4Cloud
- Connect the user to the orchestrator, and allowing graphical management of the deployment
    - The plugin retrieves a token used from the DEEP IAM instance
        - Token is sent with every REST call to the DEEP orchestrator
    - User password and name are loaded from the Alien4Cloud running instance each time a REST operation which involves the DEEP orchestrator is performed
    - Alien4Cloud sends the UI actions related to the deployment of a topology to our plugin
        - The internal workings of the plugin are hidden from the user by the Alien4Cloud web UI

Settings manager

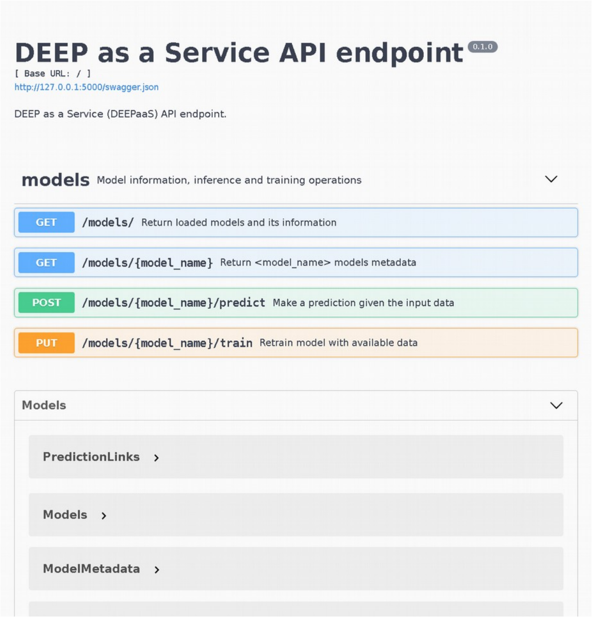
The third component is a utility used to change the settings of Alien4Cloud. It is a Java application (therefore it doesn't require additional runtime in the container) that uses the ENV variables of the container to alter the configuration files. This application runs each time the container starts, before executing the Alien4Cloud web server.

This utility allows the configuration of the instance without the need of editing the configuration files individually. This way, the administrator (who manages the Alien4Cloud deployment and its life afterwards) can launch the container and configure Alien4Cloud just by setting some variables.

5.2. DEEPaaS API

In order to easily expose the user application functionality we decided to develop a generic REST API component that could be plugged on top of the WP2 applications. This component will therefore expose the underlying functionality with minor modifications on the users' code. We based the development on the OpenAPI Specification developed by the OpenAPI Initiative. The OpenAPI Specification (OAS) [OpenAPI] defines a standard, programming language-agnostic interface description for REST APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic. When properly defined via OpenAPI, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.

The DEEPaaS API component defines a set of entry points that need to be declared at the user application side. When installed together with the application, DEEPaaS looks up those entry points, loading the user application and exposing the underlying functionality via the a REST endpoint. Since it follows the OpenAPI specification, a Swagger documentation UI [Swagger] is automatically provided (Figure 5), as well as the JSON OpenAPI specification (Figure 4).



DEEP as a Service API endpoint 0.1.0

[Base URL: /]
http://127.0.0.1:5000/swagger.json

DEEP as a Service (DEEPaaS) API endpoint.

models Model information, inference and training operations

GET /models/ Return loaded models and its information

GET /models/{model_name} Return <model_name> models metadata

POST /models/{model_name}/predict Make a prediction given the input data

PUT /models/{model_name}/train Retrain model with available data

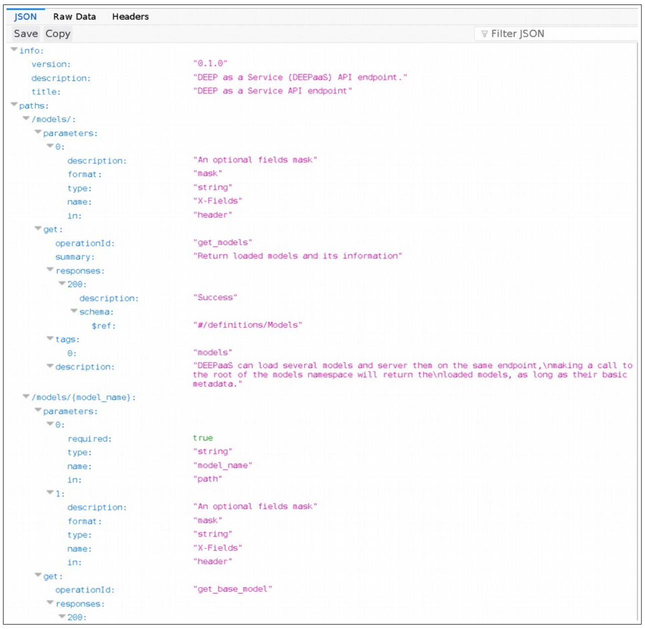
Models

PredictionLinks ›

Models ›

ModelMetadata ›

Figure 5: Swagger UI for a DEEPaaS.



```
▾info:
   version:                    "0.1.0"
   description:                "DEEP as a Service (DEEPaaS) API endpoint."
   title:                      "DEEP as a Service API endpoint"
▾paths:
  ▾/models/:
    ▾parameters:
      ▾0:
         description:          "An optional fields mask"
         format:               "mask"
         type:                 "string"
         name:                 "X-Fields"
         in:                   "header"
    ▾get:
       operationId:            "get_models"
       summary:                "Return loaded models and its information"
      ▾responses:
        ▾200:
           description:        "Success"
          ▾schema:
             $ref:             "#/definitions/Models"
      ▾tags:
         0:                    "models"
      ▾description:            "DEEPaaS can load several models and server them on the same endpoint,\nmaking a call to
                               the root of the models namespace will return the\nloaded models, as long as their basic
                               metadata."
  ▾/models/{model_name}:
    ▾parameters:
      ▾0:
         required:             true
         type:                 "string"
         name:                 "model_name"
         in:                   "path"
      ▾1:
         description:          "An optional fields mask"
         format:               "mask"
         type:                 "string"
         name:                 "X-Fields"
         in:                   "header"
    ▾get:
       operationId:            "get_base_model"
      ▾responses:
        ▾200:
```

Figure 4: OpenAPI JSON Specification.

This API component provides the following functionality:

- Loading any installed model, exposing several applications through the same endpoint.
- Fetching metadata from the installed modules.
- Triggering a training of a model, with the possibility of passing down training parameters to the model.
- Performing the evaluation of a trained model.
- Triggering an inference/prediction/classification using the loaded model, therefore exposing its functionality.
- Preliminary support for server less components.

Once an application is properly integrated with DEEPaaS API (i.e. defining which entry points should be invoked for any of the aforementioned items) it is possible access the model's functionality via the REST API in any execution environment that is available. Apart from the entry point definition, this integration is being carried out by installing DEEPaaS API and the user application together into the same Docker container. This allows to cover a wide range of execution environments, like the local computer of the scientist, a Mesos instance (as provided via WP5), a Kubernetes deployment (created by the user), a server less component (with preliminary support for the time being) or an HPC installation via uDocker.

5.3. DEEP Open Catalogue

One of the objectives of WP6 is the definition and elaboration of the DEEP Open Catalogue, containing reusable modules both for the end-users and application developers. The DEEP Open Catalogue has been structured as a set of Docker images (stored under the DEEP-Hybrid-DataCloud DockerHub [INDIGO-Docker-Hub] organization), source code repositories (stored under the INDIGO-DataCloud GitHub Organization), TOSCA templates (stored under the INDIGO-DataCloud GitHub Organization [INDIGO-DC]) and so on, that can be composed together via the application Alien4Cloud. In this way users can either reuse any existing application (and topology), modify an existing one or develop one from scratch. Currently the DEEP Open Catalogue is composed by the components described in Table 3.

Component	Purpose	Link
DEEP as a Service Dogs breed determination	Example application used by WP2 to show integration steps and testbed validation.	https://github.com/indigo-dc/DEEP-OC-dogs_breed_det
		https://hub.docker.com/r/deephdc/deep-oc-dogs_breed_det/
DEEP as a Service container for Conus (Marine Snails) classification	WP2 deep learning application for biodiversity classification use cases	https://github.com/indigo-dc/DEEP-OC-conus-classification
		https://hub.docker.com/r/deephdc/deep-oc-conus-classification/

DEEP as a Service container for seeds classification	WP2 deep learning application for biodiversity classification use cases	https://github.com/indigo-dc/DEEP-OC-seeds-classification https://hub.docker.com/r/deephdc/deep-oc-seeds-classification/
DEEP as a Service container for plant classification	WP2 deep learning application for biodiversity classification use cases	https://github.com/indigo-dc/DEEP-OC-plant-classification-theano https://hub.docker.com/r/deephdc/deep-oc-plant-classification-theano/
DEEP as a Service container for retinopathy detection	WP2 deep learning application for retinopathy detection	https://github.com/indigo-dc/DEEP-OC-retinopathy_test https://hub.docker.com/r/deephdc/deep-oc-retinopathy_test/
DEEP as a Service MODS container	WP2 deep learning application for anomaly detection using massive online data streams	https://github.com/indigo-dc/DEEP-OC-mods https://hub.docker.com/r/deephdc/deep-oc-mods/
DEEP as a Service biodiversity classificator	WP2 deep learning bundle containing all biodiversity applications into a single container	https://github.com/indigo-dc/DEEP-OC-engine https://hub.docker.com/r/deephdc/deep-oc-engine/
DEEP as a Service container for phytoplankton classification	External use case (from LifeWatch Belgium) leveraging DEEP solutions for automatic classification of phytoplankton	https://github.com/indigo-dc/DEEP-OC-phytoplankton-classification https://hub.docker.com/r/deephdc/deep-oc-phytoplankton-classification/
DEEP as a Service generic container	Generic container with DEEPaaS API installation and configuration, to be used by other use cases	https://github.com/indigo-dc/DEEP-OC-generic-container https://hub.docker.com/r/deephdc/deep-oc-generic-container/
DEEP Cookiecutter Data Science	Repository template for WP2 applications	https://github.com/indigo-dc/cookiecutter-data-science
Ansible roles	Ansible roles for installing Data Analytic Stack (DAS) in OS container	https://github.com/indigo-dc/ansible-role-jupyterhub https://github.com/indigo-dc/DEEP-OC-mods-ansible
TOSCA templates and types	TOSCA templates and the needed types for the deployment of the DEEP-OC components	https://github.com/indigo-dc/tosca-templates https://github.com/indigo-dc/tosca-types

Table 3: Contents of the DEEP Open Catalogue for the prototype.

This set of initial contents has been created in close collaboration with the networking activity (NA) Work Package 2. As a matter of fact, both work packages elaborated and adopted a templated repository for the use cases, so that the adoption of new user applications follow the same repository structure and pattern, facilitating the integration of the applications into the final solution. This templated repository includes configuration for automated builds and software quality assurance processes, tasks that are carried out under the service activity (SA) of WP3. This way, the corresponding Docker container is created automatically in the DEEP-Hybrid-DataCloud DockerHub organization, as shown in Figure 6, whenever a change is merged into the master branch of the source code repository and all the required tests are passed. Moreover, since all these components are integrated with the DEEPaaS API (as explained before) any change in this component will trigger an update on the user application container. This is currently being done using the DockerHub repository links. We expect in the next period to perform this task by adopting the tools operated by WP3 in order to have more control on the creation of the new container.

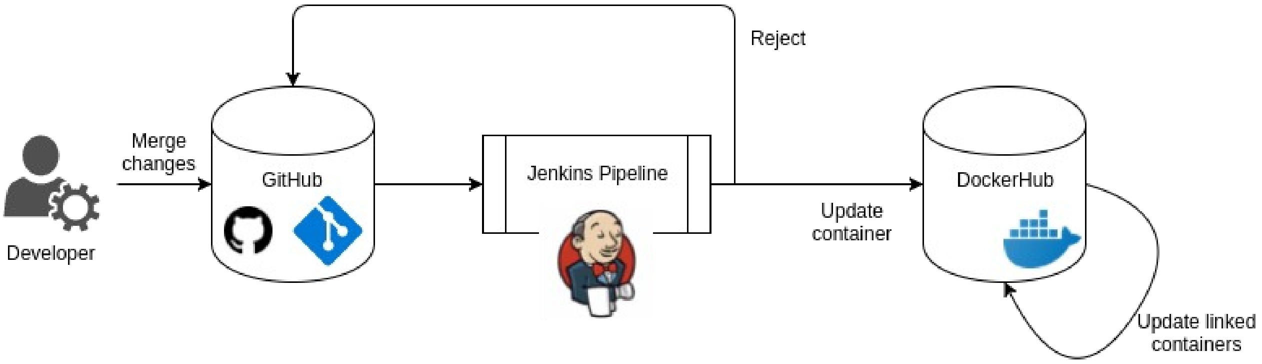


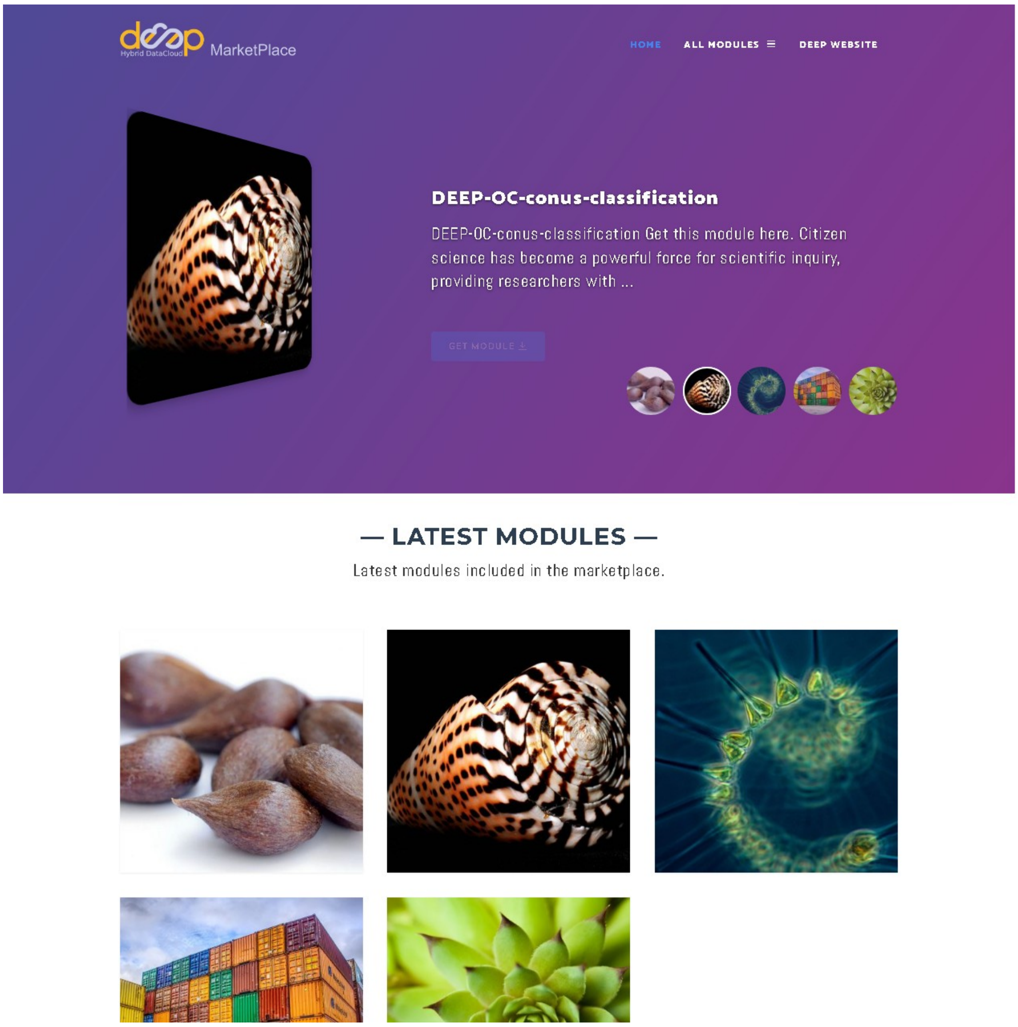
Figure 6: DevOps pipeline currently implemented

As already stated, all the DEEP Open Catalog (DEEP-OC) modules that contain a user application are based in the DEEPaaS API described before. By exploiting Docker containers together with DEEPaaS we provide a way to the users to execute their applications in different computing infrastructures and platforms:

- A user can download a module an execute it in their local computers using Docker.
- The Docker application can be executed on a production-grade server, using any of the common deployment tools.
- The Docker application can be executed on any container orchestration engine, such as Mesos (as it is being done through the Alien4Cloud and the Orchestrator) or Kubernetes.

- The Docker application can be executed on a server less infrastructure. This work is currently undergoing, but a pilot preview is provided to some use cases.
- The user application can be executed on an HPC facility, by means of the uDocker tool.

In order to provide a graphical user interface to browse all the components of the DEEP-OC catalogue we created a "marketplace" web page (http://marketplace.deep-hybrid-datacloud.eu/) where all the developed modules have been included, as shown in Figure 7. This is a simple web portal that lists, links and provides information about the developed modules, but it does not provide any extra functionality like submission of an application to the infrastructure, as this is task of Alien4Cloud. However, as part of the exploitation plans of the project, we expect to integrate these modules into a production marketplace of the European Open Science Cloud, through the EOSC-Hub project or the EGI.eu Application Database (or similar tools).



DEEP-OC-conus-classification

DEEP-OC-conus-classification Get this module here. Citizen science has become a powerful force for scientific inquiry, providing researchers with ...

— LATEST MODULES —

Latest modules included in the marketplace.

Figure 7: The DEEP-OC marketplace.

5.4. Integration with storage services

For this first half of the project our approach was based on the utilization of a local deployment of NextCloud to provide access to the users to a sync and share solution, and the rClone [RCLONE] tool to gain seamless access to the data from the execution environment. From the user's perspective, the sync and share approach is the most natural way to store and retrieve their data, therefore with this approach we will be able to replace NextCloud [NEXTCLOUD] for any similar solution deployed in the EOSC (like the storage solutions provided by the eXtreme-DataCloud project). Moreover, by leveraging rClone we are able to get seamless access to any cloud storage API (WebDav, S3, etc.) transparently for the users. Obviously, this first approach does not provide the required efficient data access for the application training, but we do expect to solve this issue during the second half of the project with the close collaboration with the eXtreme-DataCloud project (XDC) that we have set up.

In this regard, in order to adopt advanced storage solutions from XDC into DEEP so as to provide data access to the WP2 use cases we have leveraged a common and external use case (common in the sense that both projects should try to satisfy their requirements, external in the sense that it was not covered by any of the projects before). This use case is based on the automatic and marker-less pose estimation using deep learning techniques and requires from efficient data storage to perform the training, as well as a way to perform automatic and asynchronous classifications of large video data sets. As a preliminary outcome of this collaboration we set up a demo at the Digital Infrastructures for Research 2018 event (https://www.digitalinfrastructures.eu/) where we performed an automated classification of a video using the DEEP-Hybrid-DataCloud tools, reacting to storage events produced by the eXtreme-DataCloud solutions.

6. Conclusions and next steps

This document describes the initial development and integration tasks that have been carried to provide the use cases with an appropriate solution for the training of machine learning and deep learning models, exploiting accelerators through the WP5 and WP4 developments. Taking into account the milestones that were setup for this work package (listed in Table 4), at present this activity has completed the MS4, MS9 and MS14 milestones. Currently the MS19 is partially implemented and a pre-alpha preview is available for use cases to verify the required functionality.

We developed a plugin for Alien4Cloud that allows the deployment of applications directly from its graphical user interface, exploiting the INDIGO-DataCloud orchestrator. We created the corresponding TOSCA documents and templates that allow the deployment of all the modules from the A4C tool. In this regard, we integrated the use cases in Docker containers, together with the DEEPaaS API as a way to expose the user functionality in a generic way that can be either used directly by the users (by means of its web interface) or exploited by any tool (by means of its REST API).

No.	Milestone title	Month	Status
MS4	DEEP Catalog base content	3	Completed (reported in D6.1 - State-of-the-art Deep Learning, Neural Networks and Machine Learning frameworks and libraries)
MS9	DEEP-as-a-Service architecture and API description	6	Completed (reported in D6.2 - Design for the DEEP as a Service solution)
MS14	Batch executions in the DEEP-as-a-Service solution	12	Completed
MS19	Application-as-a-service implementation	24	In progress
MS22	DEEP-as-a-Service solution	24	In progress

Table 4: List of milestones for work package 6.

Moreover, although the focus has been put on the training task through batch executions, we have also carried out developments activities towards the deployment as a service of the user applications. The DEEPaaS API provides enough functionality to deploy it as a service on different execution environments and platforms (local computer, server, container orchestration engine). We expect to work further in this regard during the second half of the project, as detailed in the next paragraph.

Beyond the expected improvement activities carried out during the project lifetime, the next steps for WP6 are the following:

- Integration of Alien4Cloud with OpenID connect. Currently A4C does not support OpenID connect authentication, therefore a temporary solution was implemented for the pilot preview. We have put our effort in the support for the INDIGO-DataCloud orchestrator, as well as unexpected bug fixing in order to support our use cases, therefore we could not implement this support yet. We foresee to implement this functionality in a proper way during the next phases of the project.
- Creation of Ansible roles for the configuration and deployment of the components. Several of the DEEP Open Catalogue Docker containers install and configure the same components (like DEEPaaS API) manually. We expect to create Ansible roles to that these components can be deployed and configured using them.
- Addition of components into the DEEP-OC. We will continue creating and adding components into the catalogue, as requested by our use cases and project stakeholders.
- Integration of Big Data analytics tools into the DEEP-OC. The catalogue currently does not contain any Big Data analytics tools, but we do expect to incorporate them during the second half of the project, when the WP2 use cases will require this kind of components. Moreover, we expect to also incorporate tools from the eXtreme-DataCloud project.

- Definition of the automated DevOps pipeline for the automated deployment of the user applications as a service. We have performed preliminary steps towards the implementation of the DevOps pipeline for the user applications. Currently the user containers are created automatically whenever a change is merged into the repository but we will incorporate additional testing for the user applications. Moreover, we will define the pipeline to update the deployed user services whenever a change is done in the underlying model, meaning that a change in the code or model configuration will trigger an update of the services.
- Integration of DEEP-OC applications into production marketplaces. As part of the exploitation plans we expect to start activities in order to integrate our marketplace and the rest of the tools into production marketplaces or the corresponding tools, like the EGI.eu Application Database.
- Evolution of DEEPaaS API. We expect that the DEEPaaS API component evolves together with the use cases. We expect to elaborate a major release during the second half of the project in order to incorporate all the use cases feedback.
- OpenID Connect integration into DEEPaaS API. Currently this component does not support OpenID connect authentication. As this feature has been requested by some use cases, it is expected to be implemented in the next period.

7. Glossary

A4C Alien4Cloud

AAI Authentication and Authorization Infrastructure

API Application Programming Interface

CLI Command Line Interface

CMF Cloud Management Framework

CMP Container Management Platform

CSAR Cloud Service Archive

DEEP-OC DEEP Open Catalogue

EOSC European Open Science Cloud

GUI Graphical User Interface

IAM Identity and Access Management service

IM Infrastructure Manager

IS Information System

JRA Joint Research Activities

REST Representational State Transfer

SAML Security Assertion Markup Language

TOSCA Topology and Orchestration Specification for Cloud Applications

TRL Technology Readiness Level

VM Virtual Machine

VPC Virtual Private Cloud

VPN Virtual Private Network

WP Work Package

YAML Yet Another Markup Language

8. References

[A4C-DEEP] https://github.com/indigo-dc/alien4cloud-deep/

[D2.1] Initial Plan for Use Cases http://hdl.handle.net/10261/164311

[D6.1] State-of-the-art Deep Learning, Neural Networks and Machine Learning frameworks and libraries https://confluence.deep-hybrid-datacloud.eu/download/attachments/3145850/DEEP-JRA3-D6.1.pdf?api=v2

[D6.2] Design for the DEEP as a Service solution http://hdl.handle.net/10261/164314

[JIRA-DEEP] https://jira.deep-hybrid-datacloud.eu/

[INDIGO-DC] https://github.com/indigo-dc

[INDIGO-Docker-Hub] https://hub.docker.com/u/deephdc/

[NEXTCLOUD] https://nextcloud.com/

[OpenAPI] http://www.openapis.org/

[RCLONE] https://rclone.org/

[Swagger] https://swagger.staging.wpengine.com/swagger-ui/

[TOSCA 2017] Crandall, John, and Paul Lipton. “OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC.” https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca.