Abstract

This document describes the design of the PaaS Layer and the detailed work plan for the WP5 of the DEEP-HybridDataCloud project, which aims at adopting and extending the PaaS orchestration solution for cloud infrastructures realised within the INDIGO-DataCloud project. In particular, the main focus is on the architecture and features of the hybrid cloud solution.
Delivery Slip

<table>
<thead>
<tr>
<th>From</th>
<th>Name</th>
<th>Partner/Activity</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reviewed by</td>
<td>Name</td>
<td>Partner/Activity</td>
<td>Date</td>
</tr>
<tr>
<td>Reviewed by</td>
<td>Name</td>
<td>Partner/Activity</td>
<td>Date</td>
</tr>
<tr>
<td>Approved by</td>
<td>Name</td>
<td></td>
<td>Date</td>
</tr>
</tbody>
</table>

Document Log

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Comment</th>
<th>Author/Partner</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.0</td>
<td>03/04/2018</td>
<td>ToC</td>
<td>Stefano Nicotri / INFN</td>
</tr>
<tr>
<td>V2.0</td>
<td>12/04/2018</td>
<td>Information system components and workplan</td>
<td>Aida Palacio / CSIC</td>
</tr>
<tr>
<td>V3.0</td>
<td>12/04/2018</td>
<td>Hybrid Cloud architecture components</td>
<td>German Molto / UPV</td>
</tr>
<tr>
<td>V4.0</td>
<td>23/04/2018</td>
<td>Information system components and updated workplan</td>
<td>Pablo Orviz / CSIC</td>
</tr>
<tr>
<td>V5.0</td>
<td>23/04/2018</td>
<td>Networking components</td>
<td>Zdeněk Šustr / CESNET</td>
</tr>
<tr>
<td>V6.0</td>
<td>24/04/2018</td>
<td>Hybrid Cloud architecture</td>
<td>Marica Antonacci / INFN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Stefano Nicotri / INFN</td>
</tr>
<tr>
<td>V7.0</td>
<td>24/04/2018</td>
<td>Monitoring information and components</td>
<td>Jose Antonio Sanchez / Atos</td>
</tr>
<tr>
<td>V8.0</td>
<td>27/04/2018</td>
<td>Document reorganization</td>
<td>Marica Antonacci / INFN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Stefano Nicotri / INFN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Giacinto Donvito / INFN</td>
</tr>
<tr>
<td>V9.0</td>
<td>28/04/2018</td>
<td>Implementing review changes</td>
<td>Marica Antonacci / INFN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Stefano Nicotri / INFN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Giacinto Donvito / INFN</td>
</tr>
</tbody>
</table>
Table of Contents

Executive Summary.................................................................................................................4
1. Introduction........................................................................................................................5
  1.1. Document Purpose..................................................................................................5
  1.2. Project Scope..........................................................................................................5
  1.3. WP5 relationships with other work packages....................................................6
  1.4. Document structure...............................................................................................8
2. Description of the hybrid cloud architecture..............................................................8
  2.1. Overview of modules and components.................................................................8
  2.2. Structure and relationships..................................................................................10
  2.3. Authentication and Authorization.......................................................................15
  2.4. User interface issues.............................................................................................15
3. Detailed component description....................................................................................15
  3.1. Component template description..........................................................................15
  3.2. INDIGO PaaS Orchestrator..................................................................................16
  3.3. Infrastructure Manager.........................................................................................18
  3.4. SLA Service...........................................................................................................20
  3.5. CMDB....................................................................................................................21
  3.6. Information System................................................................................................22
  3.7. Monitoring System................................................................................................24
  3.8. Cloud Provider Ranker.........................................................................................26
  3.9. Virtual Router........................................................................................................27
  3.10. Authentication and Authorization......................................................................27
4. Reuse and relationships with other products...............................................................28
5. Design decisions and trade-offs....................................................................................29
6. Work package implementation plan............................................................................30
  6.1. Task structure.........................................................................................................30
  6.2. Coordination of activities.....................................................................................30
  6.3. Task prioritization..................................................................................................31
  6.4. Requirement tracking...........................................................................................31
  6.5. Detailed workplans................................................................................................31
  6.6. Risk assessment.....................................................................................................32
7. References......................................................................................................................33
8. Glossary..........................................................................................................................33
Executive Summary

The DEEP-HybridDataCloud project 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. Moreover, an easy transition path will be provided, following a DevOps approach for developers in order to deploy their applications as a service, to be used by a broader public.

The DEEP project and the WP5 in particular decided to exploit the INDIGO-DataCloud µServices [INDIGO-DataCloud] and others already available framework for the management of the docker containers, in order to easily build the PaaS layer enhancing with the required features, taking into account the user communities requirements.

In particular the focus of the WP5 in the DEEP project is to enable new federations approaches that will allow a more flexible services distribution and the possibility to deal with heterogeneous hardware, in particular with GPU and low latency network.

In order to fulfil the main objectives of DEEP as a Service's approach, the PaaS layer developed in WP5 will provide the capabilities of exploiting IaaS resources in the form of a Platform as a Service. This document is focused on describing the architecture of PaaS layer and the activities needed to implement it. All the components used to implement the PaaS Layer are described, together with the status of each service.

The WP5 layer will provide automation and transparent federation of IaaS resources, to allow the deployment of complex cluster of services, exploiting specialized hardware. In doing so, the details about the heterogeneous resources used in a geographically distributed environment will be hidden.
1. **Introduction**

1.1. **Document Purpose**

This document describes the design and the detailed work plan for the WP5 of the DEEP-HybridDataCloud project, which aims at adopting and extending the PaaS orchestration solution for cloud infrastructures realized within the INDIGO-DataCloud project. WP5 is focused on:

- Supporting hybrid deployments across multiple cloud sites (especially computing clusters)
- Scaling out virtual infrastructures deployed on a given cloud to exploit 3rd party resources
- Building software-based secure virtual networks able to connect resources scattered on multiple pools in a transparent way for the user
- Providing a PaaS layer to WP6/JRA3 for the automated provisioning and configuration of complex hybrid virtual infrastructures (e.g. Apache Mesos or Kubernetes clusters).

1.2. **Project Scope**

The DEEP-HybridDataCloud project started with the global objective of promoting the usage of intensive computing services by different research communities and areas, and their support by the corresponding e-Infrastructure providers and open source projects. Other objectives pursued by the project are:

- Focus on intensive computing techniques for the analysis of very large datasets considering highly demanding use cases.
- Evolve up to production level, intensive computing services exploiting specialized hardware.
- Integrate intensive computing services under a hybrid cloud approach.
- Define a “DEEP as a Service” solution to offer an adequate integration path to developers of final applications.
- Analyse the complementarity with other ongoing projects targeting added value services for the cloud.

The DEEP-HybridDataCloud project aims at providing a bridge towards a more flexible exploitation of intensive computing resources by the research community, enabling access to the latest technologies that require last generation hardware and the scalability to be able to explore large datasets. It is structured into six different work packages, covering Networking Activities (NA) devoted to the coordination, communication and community liaison, Service Activities (SA) focused on the provisioning of services and resources for the execution of the data analysis challenges, and Joint Research Activities (JRAs), dealing with the development of new components and technologies to support data analysis.
In order to achieve these objectives, we propose to evolve existing cloud services, taking into account the following design principles:

1. **Evolve the required services from TRL6 to TRL8 under an open framework** and considering existing standards for interoperability.

2. **Re-use, whenever possible, existing cloud services in production**, and in particular those being adopted for proposed e-infrastructure of the European Open Science Cloud.

3. **Consider the integration of existing specialized resources into cloud services** having in mind the point of view of the current daily management of those resources, as e.g. current HPC data centres.

4. Ensure that the resulting framework will have a **low learning curve** for the developers of the solutions, by delivering a DEEP catalogue that can be directly exploited by users to build their applications.

5. **Assure scalability and performance** of the developed solution, in order to guarantee the interest of both resource providers and users.

WP5 will take care of the provisioning of the platform exploiting the outcomes from WP4 in a hybrid approach, delivering an execution platform for WP6, ensuring that applications can be spawned across several cloud infrastructures. This will be done by enhancing the current orchestration components (especially those from the INDIGO-DataCloud project) enabling fine-grained multi-site orchestration. Following this context, one of the main goals of this work package will be to extend the information published by the provider to match the fine-grained details of user applications requirements as requested by WP6. Moreover, we will work on providing secure network connections between the different sites participating in this multi-cloud deployment, transparently for the user, so that all the provisioned resources can communicate as if they were on the same network segment. This is important when users have data and CPU available in geographically distributed infrastructure. By using a hybrid approach we will be able to transparently access data and hardware resources in a transparent way.

### 1.3. WP5 relationships with other work packages

The DEEP-HybridDataCloud project is organized in six work packages, covering Networking Activities (NA) devoted to the coordination, communication and community liaison; Service Activities (SA) focused on the provisioning of services and resources for the execution of the data analysis challenges; and Joint Research Activities (JRAs), dealing with the development of new components and technologies to support data analysis. In Figure 1, the interaction between the six work packages is shown.
As shown in Figure 1, the development activities are structured in a layer architecture, focusing the WP5 in the middle one "High Level Hybrid Cloud".

WP5 will leverage APIs and features provided by WP4, in order to exploit hardware and software solutions at IaaS level. In particular WP4 solutions will be useful in order to exploit specialized hardware (such as GPUs or low latency networks).

WP5 will provide APIs and features to WP6, in order to provide common building blocks for implementing the "Deep as a Services" approach. These APIs will be implemented by a PaaS layer that will enable automation of services, federation of heterogeneous Computing resources, capability of building complex network, and to manage all the information coming from the IaaS layer. Thus, the resources requested from WP6 will be expressed as application topologies that will involve the orchestration of several components, the usage of hybrid cloud solutions and access to hardware accelerators among other types of requirements developed at WP5 layer. Finally, the WP5 generates accurate information about the status of the resources in order to host the user applications.

The WP2 is gathering the requirements coming from the user communities so that WP6 will build a solution tailored to the needs of the selected use cases, collected in the deliverable [DEEP D2.1]. Requirements to WP5 will come from WP6 in the form of technical requirements, therefore no direct interaction with WP2 is foreseen. Further information on how user requirements are mapped into technical requirements is done in deliverable [DEEP D2.1] as well as Section 6.4.

WP5 will provide WP3 with all the needed information about the services that should be delivered in all the subsequent releases and updates of the DEEP platform. These will include not only code and artefacts but documentation as well. The whole development process will be performed
following agile methodologies (as described in [DEEP D3.1]) in order to ensure that the delivered solutions are aligned with user expectations.

1.4. Document structure

The document is structured following the IEEE STANDARD 1016 [IEEE SDS]: Software Design Specification to the extent applicable to our project, and as such contains the following sections.

- Section 2 provides the overall view of the PaaS Layer architecture of the WP5 work package. It provides an overview of the WP5 components and architecture, its relationships as well as specifications on the interaction between WP5 components and the user.

- Section 3 contains the detailed description of each of the components in the PaaS layer, following the IEEE STANDARD 1016 model.

- In Section 4 we highlight the major decisions and trade-offs we had to make when designing WP6's architecture, motivating those decisions.

- The work package implementation plan is included in Section 5.

- Section 6 details how we are focusing on improving existing software, as well as on code reusability.

- Finally, Section 7 contains all the references and links.

2. Description of the hybrid cloud architecture

2.1. Overview of modules and components

The DEEP PaaS layer is based on the components developed and integrated in the INDIGO-DataCloud project. The architecture is depicted in Figure 2 and the main components are briefly described hereafter:
The PaaS Orchestrator is the core component of the PaaS layer. It receives high-level deployment requests and coordinates the deployment process over the IaaS platforms;

The Identity and Access Management (IAM) Service provides a layer where identities, enrolment, group membership, attributes and policies to access distributed resources and services can be managed in an homogeneous and interoperable way;

The Monitoring Service is in charge of collecting monitoring data from the targeted clouds, analysing and transforming them into information to be consumed by the Orchestrator;

The Cloud Provider Ranker (CPR) is a rule-based engine that allows to rank cloud providers in order to help the Orchestrator to select the best one for the requested deployment. The ranking algorithm can take into account preferences specified by the user and other information like SLAs and monitoring data;

The SLA Management (SLAM) Service allows the handshake between users and a site on a given SLA;

The Managed Service/Application (MSA) Deployment Service is in charge of scheduling, spawning, executing and monitoring applications and services on a distributed DEEP-HybridDataCloud – 777435
infrastructure; the core of this component consists of an elastic Mesos cluster with slave nodes dynamically provisioned and distributed on the IaaS sites;

- The **Infrastructure Manager (IM)** deploys complex and customized virtual infrastructures on a IaaS site providing an abstraction layer to define and provision resources in different clouds and virtualization platforms;

- The **Data Management Services** is a collection of services that provide an abstraction layer for accessing the data storage in a unified and federated way.

- The **Information Provider and Accounting System** collects detailed information from an IaaS provider about the current status of the resources from the amount of resources of CPU, RAM or storage to the availability of a service.

Concerning WP5, the main goal in the DEEP project will consist in improving and extending the aforementioned components, in order to evolve the hybrid cloud orchestration from TRL6 to TRL8. This will include the support for specialized computing hardware resources. Indeed, a prototype demonstration of hybrid deployments across multiple sites was carried out in the framework of the INDIGO-DataCloud project, but not one involving the use of specialized computing hardware. The robustness of the whole deployment workflow implemented in the PaaS layer will be enhanced as well.

As part of the new developments, the PaaS stack will also be extended in order to better address the networking aspects in the hybrid multi-site deployments ensuring proper setup of inter-site network connections. For this purpose the **INDIGO Virtual Router** component will be used as described in the next section.

### 2.2. Structure and relationships

The following Figure 3 shows the main interactions among the WP5 components.

The main component of the WP5 layer is the (Section 3.2 - INDIGO PaaS Orchestrator). The Orchestrator is a service that provides a REST API to receive a deployment request following the TOSCA standard, and is responsible for coordinating the provisioning of the underlying computational and storage resources and their configuration, as indicated in the TOSCA template.

The Orchestrator interacts with different services in the PaaS layer to collect all the information needed to generate the deployment workflow:

- QoS/SLA constraints from the SLA Management System;

- Health status and capabilities of the underlying IaaS platforms and their resource availability from the CMDB and the Monitoring Service;

- The status of the data files and storage resources needed by the service/application and managed by the Data Management Services;

- Priority list of sites sorted by the CPR Service on the basis of pre-defined rules.
This information is used to perform the matchmaking process and to decide where to deploy each service.

Then, the orchestration of the resources provisioning on the chosen site is delegated to the IM or to OpenStack Heat (with TOSCA enabled endpoint). In case of failure on the selected site, the deployment is marked as failed and no working deployment is returned to the user.

In the past INDIGO experience, one of the reasons for a failed deployment process was the quota exceeding on the chosen site. This was due to the missing feedback from the accounting and, even if this feedback were in place, the updated information might arrive late for the Orchestrator to choose the best site for the deployment.

The Figure 4 shows the approach that will be implemented in DEEP to overcome this limitation and to manage the failures: a trial-and-error mechanism will allow the Orchestrator to re-schedule the deployment on the next available cloud provider from the list of candidate sites returned by the CPR.
So, the workflow is the following:

1. the user requests the deployment with some requirements;
2. the orchestrator contacts SLAM, CMDB and Monitoring services to collect info about the cloud providers supporting the user organization and full-filling the user requirements (e.g. cloud images, data location, etc.);
3. the orchestrator contacts the CPR service providing the collected information;
4. CPR returns the Orchestrator with a sorted list of cloud providers;
5. the Orchestrator schedules the deployment on the first provider in the list;
6. the Orchestrator monitors the deployment until it is successful or fails;
7. in case of failure, the Orchestrator schedules the deployment on the next provider in the list, if any.

This approach will help also in case of hybrid multi-site deployments and will allow the PaaS stack to react better to unexpected events or events that were not detected in time by the monitoring pillar or other services (e.g. quota exceeded on a site).
Concerning the multi-site deployments, in INDIGO-DataCloud the deployments of virtual infrastructure across different IaaS cloud sites were restricted to the specific use case of virtual elastic clusters based on the SLURM LRMS.

Virtual elastic clusters consist of a front-end node that is deployed on a cloud site and configured with a Local Resource Management System (LRMS) and an elasticity manager. In INDIGO-DataCloud, CLUES was adopted as the elasticity manager supporting several batch systems such as HTCondor, PBS/Torque, SLURM and Container Orchestration Platforms such as Apache Mesos. Once jobs are submitted to the LRMS in the frontend node, the elasticity manager decides to deploy additional worker nodes on-demand to be able to execute the increased workload. Once the nodes are no longer required the corresponding Virtual Machines are terminated.

The Orchestrator was configured so that worker nodes for a given virtual cluster were always provisioned from the very same IaaS Cloud site on which the front-end node was deployed.

The approach to support virtual elastic clusters across different IaaS Cloud sites involved using the hybrid parameter in a TOSCA template that described the deployment of such cluster (see for example the elastic_cluster.yaml TOSCA template [INDIGO TOSCA types]). Setting the hybrid parameter to true causes:

- The Orchestrator is able to deploy additional worker nodes from any available IaaS Cloud site, regardless of the location of the front-end node.

- The deployment artefacts for the SLURM LRMS include additional steps when the hybrid mode is selected: An OpenVPN server was installed and configured in the front-end node and a VPN client is installed and configured in the worker nodes to connect to the front-end node's OpenVPN server. Further details about how this is automatically deployed and configured are available in the SLURM deployment artefacts created for TOSCA in INDIGO-DataCloud. Therefore, every time a new worker node is requested to be deployed by CLUES, the Orchestrator chooses a cloud site, deploys a VM and configures it so that a VPN tunnel is established between the worker node and the front-end node.

Therefore, the VPN is deployed ad-hoc for the SLURM-based virtual elastic cluster. When the resources (VMs) of the virtual cluster are terminated, no traces of the VPN tunnel are left. This approach does not involve modification of the underlying IaaS cloud site. A downside of this approach is that the front-end node acts as the main bottleneck, since it routes all the connections from the different worker nodes.

In DEEP the current workflow will be extended to other types of clusters (Mesos, Kubernetes, etc.) and the orchestration stack will be extended with inter-site overlay network management capabilities. To this purpose, the INDIGO Virtual Router will be used, a CMF-agnostic appliance intended for routing traffic between isolated local virtual networks. It is designed as a fall-back solution that integrates multiple widely available components (OpenVPN, ISC DHCP Server), whose code mainly consists in configuration, and should work everywhere. It is deployable in user space on Debian/Ubuntu, requiring no coordination with site administrators. It creates overlay
networks (IP over IP VPN, currently OpenVPN, potentially also different technologies such as VXLAN), requiring no support in networking hardware.

The INDIGO Virtual Router can be deployed in three different roles:

- **vRouter**: running OpenVPN client and routing non-local traffic to the Central node
- **Central node**: hosting an OpenVPN server and routing local traffic towards the private topology (connected OpenVPN clients)
- **External central node**: OpenVPN server, running outside the deployment, typically prepared manually by the user pre-deployment.

Each deployment requires at least one central node, resulting in a star topology. Redundant star topologies are also to be supported, meaning that there can be multiple central nodes configured. See figures below.

![Deployment with a single central node: A single OpenVPN server with clients connecting from off-site networks](image1)

![Deployment with multiple central points (two in this case) in active-passive roles](image2)

In the simplest deployment scenario, all traffic between different sites (individual local virtual networks) and also all traffic to and from the Internet is expected to pass through the central point. If that presents a bottleneck, there are approaches that may limit the effects:

- The individual local virtual networks may use NAT, and virtual machines within may download workload directly from any endpoint with a public network address.
- The VPN tool (OpenVPN) may be reconfigured to use a "lightweight" encryption algorithm.

By default, though, this solution favours security over transmission performance, hence fully encrypted VPN is the default option.

The INDIGO Virtual Router integrates multiple widely available components (OpenVPN, ISC DHCP Server). It contains a bare minimum of its own code, mainly for configuration.

The PaaS Orchestrator will decide on the networking topology, designate the site(s) to hold the central node(s), make sure keys (certificates) are exchanged.
2.3. Authentication and Authorization

Although not explicitly present in the described architecture, a key component is the authentication and authorization component. The DEEP project will not develop any Authentication and Authorization Infrastructure (AAI), but all the services will be built and integrated taking into account the AARC Blueprint Architecture (BPA), and any additional EOSC recommendations that may arise, in order to ease the integration path into any existing e-Infrastructure following the same recommendations. Although the AARC Blueprint Architecture is designed to be technology agnostic, OpenID Connect has been chosen as the authentication and authorization technology of choice within the DEEP project. The rationale behind this decision is that OpenID Connect is the driving technology in current existing e-Infrastructures such as the EGI Federated Cloud or the LifeWatch ESFRI. Therefore, all the services requiring authentication and authorization will be built or enhanced in order to support OpenID Connect.

2.4. User interface issues

The PaaS Layer will be mainly accessed by WP6 services. Therefore, end users will not directly exploit the WP5 services and APIs, even if this is possible.

The WP5 layer will provide REST APIs for any of the micro-services, but more in concrete terms, the PaaS Orchestrator will export an authenticated REST API in order to accept TOSCA submission. This will be used by WP6 and, eventually, by expert users via the CLI interface.

Internally, each service will provide well defined REST APIs in order to let the Orchestrator interact with all of them.

3. Detailed component description

According to the components already identified in the description of activities, and the state of the art described in the corresponding section, for each of the generic components that we have already described in this document we have selected the following software assets or services.

3.1. Component template description

The template shown below will be used as the structure for including a complete description of each component in the following submissions.
### 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-HybridDataCloud objectives, in order to fulfil the user requirements and to build the DEEP as a Service functionality.

### Current TRL status

### Expected TRL evolution

### 3.2. INDIGO PaaS Orchestrator

<table>
<thead>
<tr>
<th>Identification</th>
<th>INDIGO Orchestrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Java Application</td>
</tr>
<tr>
<td>Purpose</td>
<td>The Orchestrator receives high-level deployment requests and coordinates the deployment process over the underlying IaaS sites. This component is built around a workflow engine: the implemented workflows include the interactions with other micro-services of the PaaS layer as detailed below.</td>
</tr>
<tr>
<td>Function</td>
<td>The Orchestrator manages the deployment process implementing a complex workflow. The main steps are summarized hereafter: 1. User authentication/authorization 2. User request validation and TOSCA template parsing 3. Retrieve information about the IaaS sites, including SLAs and monitoring</td>
</tr>
</tbody>
</table>
4. Service/Virtual infrastructure deployment exploiting IaaS automation tools like INDIGO Infrastructure Manager, OpenStack Heat, or docker orchestration tool like Apache Mesos (Marathon and Chronos framework are supported) or other tools that can be plugged in as shown in the figure below.

5. Check deployment readiness

This high-level schema is implemented using BPM technology (workflow). The Orchestrator implements a set of workflows to manage the deployment life cycle requests, such as create, update and delete.

The user only has to submit a deployment request providing the description of the topology of the services/infrastructure using the TOSCA language. After the workflow starts, the user can poll the Orchestrator APIs to know the service deployment status.

The following figure shows the high-level architecture of the PaaS Orchestrator:

The INDIGO Orchestrator component is implemented using Java technologies and open-source frameworks such as: jBPM (orchestrator engine), Alien4Cloud TOSCA library (used for the Orchestrator TOSCA parser), Hibernate ORM with MySQL database, etc.

After receiving a deployment request, the Orchestrator interacts with different PaaS services in order to select the best IaaS site for the deployment. The Orchestrator collects information about the user's SLAs (asking the SLAM service), information about the computing and storage services available at the sites (asking the CMDB service), the monitoring data. If the deployment site and resources are available, the deploying process begins with IM/HEAT or a dedicated adapter.

Information about the deployments are stored in a MySQL database.

The Orchestrator depends on the following services:

- INDIGO IAM for user authentication;
- INDIGO SLAM for collecting information about the user's SLAs;
- INDIGO CMDB for collecting information about the IaaS sites (available
virtual images, compute and storage services, etc.);
• INDIGO Monitoring service for collecting services health status and metrics;
• INDIGO Cloud Provider Ranker (CPR) for ranking the cloud providers;
• IaaS orchestrator like INDIGO IM, OpenStack Heat for cloud resources provisioning and configuration;
• Apache Mesos (Marathon/Chronos) for deploying docker containers

Moreover the Orchestrator depends on
• Database: to store information about the deployments

| Interfaces | The Orchestrator exposes RESTful Web Services.
It complies with RFC 2616 standard and uses JSON data format as request and response encoding. All messages and error codes comply with standard status code definition. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Web Services message data are in JSON encoding. Deployments data, endpoints and configurations are stored in MySQL databases and are accessible through RESTful APIs. An additional MySQL schema stores the information of jBPM processes.</td>
</tr>
<tr>
<td>Needed improvement</td>
<td>The Orchestrator needs to be modified in order to support deployments that require the use of specialized computing resources. Another important improvement will be needed in the implemented workflow in order to provide better support for the hybrid multi-site deployments.</td>
</tr>
<tr>
<td>Current TRL status</td>
<td>The INDIGO PaaS Orchestrator is TRL8 for standard deployment on single IaaS instances. It is TRL6 for the deployment of the services across different IaaS or specialized hardware.</td>
</tr>
<tr>
<td>Expected TRL evolution</td>
<td>At the end of the project the Orchestrator will be delivered as TRL8 also for the deployment of the services across different IaaS or specialized hardware, and TRL9 for the basic functionalities.</td>
</tr>
</tbody>
</table>

### 3.3. Infrastructure Manager

<table>
<thead>
<tr>
<th>Identification</th>
<th>Infrastructure Manager (IM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>A service and a client.</td>
</tr>
<tr>
<td>Purpose</td>
<td>The IM is a service for the whole orchestration of virtual infrastructures and applications deployed on it, including resource provisioning, deployment, configuration, re-configuration and termination.</td>
</tr>
<tr>
<td>Function</td>
<td>The service manages the complete deployment of virtual infrastructures or individual components within them. The status of a virtual infrastructure can be: • pending: launched, but still in initialization stage; • running: created successfully and running, but still in the configuration stage; • configured: running and contextualized; • unconfigured: running but not correctly contextualized; • stopped: stopped or suspended; • off: shutdown or removed from the infrastructure; • failed: an error happened during submission;</td>
</tr>
</tbody>
</table>
unknown: unable to obtain the status.
The following figure shows a state transition diagram for a virtual infrastructure.

These are the main requirements concerning the IM in the DEEP-HybridDataCloud project:

- Deploy a TOSCA-specified virtual infrastructure.
- Get information from a deployed infrastructure, including status, specification information, and contextualization logs.
- Reconfigure an existing infrastructure, adding or removing VMs.
- Stop and restart existing infrastructures.
- Terminate an existing deployed infrastructure.

**High level architecture**

The following figure describes the high-level architecture of the IM, including external dependencies.

IM uses the Virtual Machine Image (VMI) Repository & Catalogue. (VMRC) for searching the VMIs. IM integrates a cloud selector that selects the most suitable VMI for the application description. Then, a configuration manager based on Ansible configures the VMs deployed by the cloud connector and installs the
The cloud connector provides the independence to the cloud IaaS platform.

### Dependencies
The IM service requires two additional components to work, IaaS cloud resources and a VMI repository. It supports multiple back-ends and standards (Amazon EC2, Microsoft Azure, Google Cloud, OpenNebula, OpenStack, OCCI, Fogbow, Kubernetes, etc.), which enables interacting with other resources and back-ends. IM uses VMRC as the VMI repository, although this can be skipped in the case the INDIGO PaaS Orchestrator is used to provide VMI selection.

### Interfaces
The IM service supports two APIs:
- The native one in XML-RPC
- A REST interface.
It also includes a command-line Python client which interacts with the XML-RPC API.

### Data
The IM uses three types of information:
- Application descriptions following the OASIS TOSCA Simple Profile in YAML Version.
- Information about the cloud providers end-points and associated metadata to be used by the IM.
- Information about the deployed infrastructures (specifications, IDs, status, end-points, etc.) in a MySQL database or in a file.

Client and server exchange the data through the parameters of the API calls.

### Needed improvement
For the DEEP-HybridDataCloud project, two main improvements are required. On the one hand, include better support for the deployment of virtual infrastructures across hybrid infrastructures which may involve the use of specialized computing devices such as GPUs or low latency interconnects.

### Current TRL status
The IM is considered to be in TRL 8 for its standard features. The IM is part of the EOSC-Hub catalogue of services and it is currently being used in production in the EGI VMOps Dashboard. However, the support for the deployment of infrastructures across multiple Cloud sites involving specialized computing devices and/or low latency interconnects is considered at state TRL 6.

### Expected TRL evolution
Prototypes of deployments across on-premises and public Cloud sites were developed in INDIGO-DataCloud but the level of maturity of this approach will increase during this project to further include these computing devices as part of the life cycle for virtual infrastructure deployment so that applications can seamlessly access them.

## 3.4. SLA Service

<table>
<thead>
<tr>
<th>Identification</th>
<th>SLA Manager (SLAM)</th>
</tr>
</thead>
</table>
| **Type**       | Web portal for the users for composing their SLAs  
|                | Micro-service that provide SLA related details and events to the relevant components (e.g. Orchestrator) |
| **Purpose**    | Allowing the handshake between a user and a site on a given SLA;  
|                | Providing the Orchestrator with the useful information for taking the decision on tasks scheduling according to the agreed and valid SLAs; |
Describing the QoS that a specific user/group has over a given site or, generally, in the PaaS as a whole; this includes a priority of a given users, the capability to access to different QoS at each site (Gold, Silver, Bronze services);

Function

SLAM is central component for SLA operations in the system. The assumption is that all the user actions are executed in context of a clearly identified and valid SLA.

SLAM supports users by collecting all the valid elements for the service they want to obtain. Users specify their request and SLAM provides options to cover it with the available offering from sites. This can be provided in automatic or brokered fashion depending on various configurable conditions. All SLAs are supported with defaults for not specified conditions. The result of SLA closing is a clear specification on what user should expect from the system and what system and sites guarantee to provide.

In the lifetime of a SLA, SLAM notifies relevant actors and technical endpoints about changes in its status. Through REST interfaces SLAM provides a list and details of SLAs requested for specific context (validity, provider, service, etc.).

High level architecture

SLAM consist of the following components:

- Web portal – where all the operation engaging users and other actors are performed;
- SLA Repository – database for all the documents; status and other relevant data are collected;
- Analytics module – component responsible for analysing monitoring/accounting data against SLAs;
- Notification services – component responsible for all outgoing messages from SLAM;
- Information Interface – module provides SLA-related data through REST interface for other services

Dependencies

SLAM uses AAI to authenticate users and providers and recognize their roles in the web interface.

Interfaces

The main interface that SLAM provides is REST Information Interface.

Data

SLAM provides SLAs with JSON and/or XML format.

Needed improvement

In the context of the DEEP project, the SLAM service will need to be extended in order to enrich the SLA description with details about the specialized hardware resources (GPU, low-latency network like Infiniband) provided by the resources providers to the users. This work will consider EOSC-hub project outcomes in order to come up with a joint solution.

Current TRL status

SLAM service is already at TRL8 for the definition of the SLAs on the standard computing and storage resources, while it is TRL5 for what concerns the definition of the specialized hardware support and for multi-IaaS support.

Expected TRL evolution

SLAM service will be TRL8 ad the end of the project also for what concerns the specialized hardware and on multi-IaaS support.

3.5. CMDB

<table>
<thead>
<tr>
<th>Identification</th>
<th>Configuration Management Database (CMDB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Web service</td>
</tr>
</tbody>
</table>
Purpose | Managing information about the cloud providers
---|---
Function | In the good practices on IT service management, like ITIL or ISO 20000 or FitSM, this component is called Configuration Management Database (CMDB), and its function is to contain all the configuration items (CIs) that are valid to manage the infrastructure. CIs may include components, relations between them and responsibilities to operate them. CMDB plays an important role in the majority of operation processes related to infrastructure concerning changes, releases, SLA management, availability, monitoring, and even incident and problem management. INDIGO CMDB service has a role of CMDB according to ITSM definition. However, the challenge of managing infrastructure configuration in federated setups requires specific solutions. Specific requirements come from the fact that different organization contributes to configurations, and boundaries between the configuration of providers and federated infrastructures are blurred.

High level architecture | CMDB consists of two main components:
- database (couchdb)
- REST API server

Dependencies | The information stored in the CMDB can be populated from external sources, like the Cloud Information Provider. The integration with INDIGO IAM allows to manage user's authentication and authorization

Interfaces | This component provides a REST API endpoint

Data | The data managed by CMDB are JSON formatted

Needed improvement | In the context of the DEEP project extensions for this component will be needed in order to support the description of the specialised computing devices and to collect info about other IaaS resources and services (e.g. Mesos/Kubernetes endpoints)

Current TRL status | CMDB service is already at TRL8 for the definition of the service/resources configuration on the standard computing and storage resources, while it is TRL5 for what regards the definition of the specialized hardware support and for multi-IaaS support.

Expected TRL evolution | CMDB service will be TRL8 ad the end of the project also for what regards the specialized hardware and on multi-IaaS support.

### 3.6. Information System

<table>
<thead>
<tr>
<th>Identification</th>
<th>Cloud Information System (IS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Service</td>
</tr>
<tr>
<td>Purpose</td>
<td>Collecting and making available information about status and availability of resources within an IaaS provider.</td>
</tr>
<tr>
<td>Function</td>
<td>The cloud Information System gathers detailed information from an IaaS provider monitoring the current status of the services and collects information as amount of resources (CPU, RAM or storage), or availability of services, which is used to decide whether to allocate a PaaS layer service into a given infrastructure.</td>
</tr>
</tbody>
</table>
The cloud information system has a hierarchical structure composed of three levels, whose building block used in this structure is the Berkeley Database Information Index (BDII), which can be visualized as an LDAP Database. Each of this hierarchy levels are:

- **Top level BDII**: aggregates all the information from all the site level BDII s and hence contains information about all services. The installation of multiple instances of the Top level BDII is needed in order to provide fault tolerant and load balanced features.
- **Site BDII**: aggregates the information from all the core BDII s running on that site.
- **Core BDII**: is usually co-located with the service and provides information about that service.

### Information System current architecture

Within the INDIGO-DataCloud Project, the cloud-info-provider tool was the Cloud Information System used and will be continued in this project. The BDII s are populated with information by running information providers.

### Dependencies

The Information system supports two cloud middleware providers:

- OpenNebula
- OpenStack using Keystone authentication with API v3 only

Furthermore, this service depends on the following services:

- BDII provides information about the status of the services;
- python-novaclient for OpenStack providers

### Interfaces

The cloud-info-provider tool collects information about an IaaS provider and formats it according to the publishing solution, such as JSON for INDIGO-DataCloud's Configuration Management Database (CMDB) consumption. The
information is represented using GLUE Schema v2.

| Data | The cloud-info-provider generates a LDIF according to the parameters defined in a YAML template describing the information of the cloud resources. |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| Needed improvement | In the context of the DEEP project, the cloud-info-provider will extend the current information published by the IaaS provider to match fine-grained details for the user applications requirement (e.g. GPU accelerators and low-latency/high-throughput networks). Moreover, the transport protocol will be moved away from the current LDAP protocol, adopting an asynchronous communication protocol provided by message queues. This particular item will be carried out in coordination with the EOSC-Hub project corresponding activity, and agreed with the EGI Federated Cloud coordination. |
| Current TRL status | The Cloud Information System is already at TRL8 stage for defining the information about standard information about IaaS resources. The Cloud Information System is TRL6 for the definition of complex hardware infrastructure, specialized hardware. |
| Expected TRL evolution | The Cloud Information System will be released at the end of DEEP project as TRL8 also for the specialized hardware. |

### 3.7. Monitoring System

<table>
<thead>
<tr>
<th>Identification</th>
<th>Monitoring System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Web service</td>
</tr>
<tr>
<td>Purpose</td>
<td>Provides metrics needed to evaluate the health of the underlying IaaS sites and of the PaaS services.</td>
</tr>
</tbody>
</table>

**Function**

The Monitoring service will provide a comprehensive REST API to gather information about two main aspects of the INDIGO-DataCloud PaaS:

- On one hand, the monitoring service will provide information about PaaS Core services such as performance indicators, enabling self-monitoring of the Core Service health and service-related availability indicators.
- On the other hand, the monitoring service will provide information about key parameters of the Virtual Machines and Containers provided to the clients, such as CPU usage.

It may also provide information about the state of the IaaS Cloud infrastructures, in terms of available resources, to support brokering on deciding the most appropriate IaaS on which to provision infrastructure.

**High level architecture**

- **Zabbix Server**
  - Zabbix is an open source, enterprise ready monitoring solution which offers high scalability and throughput. It includes a GUI administration tool for easy management that allows an administrator to set which metrics are going to be gathered and create rules for anomaly detection and notification among other features. Once the metrics are set, a series of pods running on different locations gather information about the underlying system, sending them to the central server for storage and analysis.

- **Zabbix Wrapper**
  - Zabbix offers an API that provides more than 200 different methods for server
management and automation, metric gathering and historical data. However, being so comprehensive and based on a custom JSON-RPC protocol, INDIGO-DataCloud developed a REST wrapper around some of its functionality that serves as the API for the PaaS layer.

**Monitoring Probes**

Zabbix supports two different methods for metric gathering:

- **Pull**: some agents distributed around the infrastructure are ready to be executed remotely by the central Zabbix server periodically. These agents will gather some metrics and send them to the server upon request. This method guarantees that data will be collected periodically and if not (in case the agent is inaccessible) it provides a quick response through the administration console or notifications to clients which are subscribed to such events. The disadvantage is that agents need to be running on nodes which is accessible from the outside and has an open port, listening for orders from the server. This is not always possible due to firewall rules and isolation or several parts of an IaaS deployment.

- **Push**: these same agents execute periodically in different nodes and send data to the central server at will. In this case, the server doesn't have any control over the periodicity in which data is gathered but it can set warning messages if it doesn't receive data for some time. This deployment is easier since only the central server needs to be accessible to the different nodes, while these nodes don't need to be listening on open ports, which simplifies the security management.

Since we operate in complex clusters with security as a priority, in INDIGO-DataCloud, and so in DEEP, we decided to go for a push approach in which different agents called probes, are distributed at different parts of the IaaS. They run periodically monitoring the status of different systems and sending the data back to the central server.

So far, the following probes are already developed:

- **OCCI Probe**, which monitors the OCCI interface of an OpenStack installation.
- **Heapster Probe**, which monitors a Kubernetes cluster using the Heapster component to gather different metrics about containers, pods and services.
- **Mesos Probe**, composed of three sub components, each monitoring a part of a Mesos Cluster:
  - Mesos, which gathers information about the cluster using the monitoring API
  - Chronos, which checks periodically the capability to create and run jobs
  - Marathon, which does the same as the Chronos sub probe with Marathon tasks
- **OpenStack Probe**, which checks the capability of an OpenStack deployment to correctly spawn virtual machines
- **Infrastructure Manager Probe**, which checks INDIGO DataCloud's Infrastructure Manager functionality and capability of creating and deleting test infrastructures in OpenNebula deployments.

The data gathered by these probes will be then available to upper PaaS layers through a REST API.

**Dependencies**

The Monitoring system relies on the CMDB for collecting information about the IaaS endpoints to be monitored.
Interfaces
It provides REST API endpoint

Data
XML/JSON

Needed improvement
The current probes will be further improved: the probes for Mesos/Marathon/Chronos will be extended to verify if the GPU support is enabled. The work done in the EOSC-Hub project will be taken into consideration in order to adopt a common solution.

Current TRL status
The current version of the monitoring service is TRL8 for the simple IaaS resources, and the already available PaaS µServices.

Expected TRL evolution
The monitoring services will be upgraded in order to better monitor the new configuration of the IaaS and the upgraded version of the PaaS µServices. This component will be released at TRL8 on all the supported features.

3.8. Cloud Provider Ranker

<table>
<thead>
<tr>
<th>Identification</th>
<th>Cloud Provider Ranker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Web Service</td>
</tr>
<tr>
<td>Purpose</td>
<td>Ranking cloud providers according to rules implemented with the Drools framework [DROOLS].</td>
</tr>
<tr>
<td>Function</td>
<td>It performs the ranking of cloud providers based on an internal algorithm which uses weights and normalization parameters (described in the Ranking Algorithm guide [CPR-Doc]).</td>
</tr>
</tbody>
</table>
| High level architecture | It consists of two main components:  
• The WEB server, implemented using the HttpServer Java library (http://goo.gl/QLBjiP);  
• the rule engine, based on the open-source Drools framework |
| Dependencies   |                      |
| Interfaces     | It provides a REST API endpoint |
| Data           | All the data managed by this service are JSON encoded. |
| Needed improvement | The CPR will be extended in order to support the specialized computing hardware (GPUs and low-latency networks) as additional information to be included in the ranking algorithm. |
| Current TRL status | TRL6 |
| Expected TRL evolution | TRL8 |

3.9. Virtual Router

<table>
<thead>
<tr>
<th>Identification</th>
<th>Networking System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Service</td>
</tr>
</tbody>
</table>
| Purpose        | Routing traffic between isolated local virtual networks, in order to extend the
orchestrator stack with inter-site overlay network management capabilities.

**Function**
CMF-agnostic appliance designed as a fall-back solution that integrates multiple widely available components that should work everywhere. Deployable in user space on Debian/Ubuntu, requiring no coordination with site administrators. It creates overlay networks (IP over IP VPN, currently OpenVPN, potentially also different technologies such as VXLAN), requiring no support in networking hardware.

**High level architecture**
The INDIGO Virtual Router consists of an integration of multiple widely available components (OpenVPN, ISC DHCP Server), and can be deployed in three different roles:
- vRouter: running OpenVPN client and routing non-local traffic to the Central node
- Central node: hosting an OpenVPN server and routing local traffic towards the private topology (connected OpenVPN clients)
- External central node: OpenVPN server, running outside the deployment, typically prepared manually by the user pre-deployment.

**Dependencies**
- OpenVPN 2
- ISC DHCP server

**Interfaces**
- VPN endpoint on instances configured as central nodes
- DHCP, default gw facing the local network/private address range
- SSH interface for configuration (Ansible)

**Data**
Mainly configuration files

**Needed improvement**
The Virtual Router will be enhanced in terms of possible configuration supported, namely to facilitate setup and configuration by automated tools, and in terms of reliability and performance of the services for long term execution.

**Current TRL status**
The Virtual Router is at the moment at TRL6.

**Expected TRL evolution**
It is foreseen to release the Virtual Router at TRL8 by the end of the project.

### 3.10. Authentication and Authorization

<table>
<thead>
<tr>
<th>Identification</th>
<th>INDIGO IAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Service</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>The IAM service provides user identity and policy information to services so that consistent authorization decisions can be enforced across distributed services.</td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td>The INDIGO IAM service provides a layer where identities, enrolment, group membership, attributes and policies to access distributed resources and services can be managed in a homogeneous and interoperable way. It supports the federated authentication mechanisms behind the INDIGO AAI and it is compatible with the AARC Blueprint Architecture. Within DEEP-HybridDataCloud the INDIGO IAM will be used to provide authentication and authorization information for all the components requiring such access control.</td>
</tr>
<tr>
<td>High level architecture</td>
<td>N/A</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>Dependencies</td>
<td>All the components requiring authentication in the system should implement OpenID Connect as the authentication and authorization method.</td>
</tr>
<tr>
<td>Interfaces</td>
<td>The INDIGO IAM implements an OpenID Connect provider and OAuth 2.0 authorization server.</td>
</tr>
<tr>
<td>Data</td>
<td>The IAM stores data about identities, sessions and policies in a persistent database.</td>
</tr>
<tr>
<td>Needed improvement</td>
<td>No improvement or developments foreseen.</td>
</tr>
<tr>
<td>Current TRL status</td>
<td>TRL 8</td>
</tr>
<tr>
<td>Expected TRL evolution</td>
<td>N/A</td>
</tr>
</tbody>
</table>

4. **Reuse and relationships with other products**

The project has along its design principles to reuse existing components as much as possible, contributing the needed changes to the original products when needed and whenever this is possible. The DEEP project has its roots in the solutions developed by the INDIGO-DataCloud project, since several of the production-grade INDIGO components (TRL8, TRL9) will be further enhanced in order to fulfil the requirements of the DEEP use cases, promoting that specific functionality from prototype stages (TRL6) into production grade component (TRL8). Some components are outside the scope of the project consortium. For those cases, we will still seek to contribute our changes, ensuring that these modifications and enhancements are of broad interest.

Moreover, there are some components that are tightly bounded to an existing e-Infrastructure or that are also being extended or developed in other existing projects (like the EGI Federated Cloud e-Infrastructure, the EOSC-Hub project and the information system, monitoring, SLA manager and CMDB or like the INDIGO-DataCloud Orchestrator, whose data management capabilities are being improved in the eXtreme-DataCloud project). For these cases, the DEEP-HybridDataCloud project has established a proper coordination (through the Cooperation Open Board) in order to ensure that the developed functionality or improvements are aligned with any ongoing efforts, ensuring a convergence and integration of solutions.

All the expected enhancements and contributions have been already highlighted in the corresponding "detailed component description" section.

5. **Design decisions and trade-offs**

The design has been guided taking into account the input from the use cases collected in WP2 in the form of technical requirements. This design is subject to change due to further interactions with the DEEP use cases, resulting in an update of this document.
In order to select the best suited component for each of the WP5 components we made a deep scouting starting from the experience of the INDIGO-DataCloud projects. However, based on our experience, some technologies have been chosen without performing such state of the art study. This is the case of the following (a justification is provided for each of them):

- The usage of Docker containers and Ansible roles in order to build up the user applications together with the PaaS Component that has to be automatised. Taking into account the positive experience obtained in the INDIGO-DataCloud project, and the current trend in the scientific computing area of delivering the software as Docker containers, Docker has been chosen as the desired execution method within the DEEP project. Docker container images will be built using Dockerfiles and Ansible. Ansible is an open source software that provides automation for software provisioning, configuration management and application development. By using Ansible within DEEP, it would be possible to express a desired configuration state that will be applied to a given Docker container when it is built. This way it would be possible to create generic Ansible modules, that can be then combined to build more complex configurations, making possible to apply the same installation and configuration (for example for a framework like Mesos or Kubernetes) in different containers and/or virtual machines by simply applying those modules on that Docker container or virtual machine.

- The usage of GitHub and DockerHub for storing the DEEP catalogue and the source code of the PaaS µServices. These components have been selected due to its wide adoption in their corresponding areas and the social features that they provide: forking, user interaction, etc. Maintaining a custom service for this purpose will limit the projects visibility and outreach, hindering the adoption by external user communities.

- The selection of OpenID Connect as the authentication and authorization technology and the usage of the INDIGO IAM as the authentication and authorization component. The DEEP project does not have effort to develop and Authentication and Authorization Infrastructure (AAI) as it is outside of the scope of the project. However, proper authentication and authorization are key aspects of any distributed e-infrastructure exploiting the DEEP products. In order to ease this integration and exploitation plan we have designed our services to follow the AARC Architecture Blueprint and its guidelines (as well as any other relevant recommendations that may arise from the EOSC context). In this regard, OpenID Connect is one of the most widespread technologies in the EOSC, and the INDIGO IAM is a component that provides an OpenID Connect Provider as well as an OAuth 2.0 authorization server.

- The usage of TOSCA as the language for describing the application topologies. During the INDIGO-DataCloud project we already performed a selection of existing methods, languages and standards that would make possible to define complex application topologies. The OASIS TOSCA specification was considered the most suitable one, as it provides an abstraction from the underlying implementation, making it possible to use a TOSCA document across several cloud management frameworks. Moreover, TOSCA’s
extensibility allows an easy definition of custom types, delivering a high degree of flexibility.

- The μServices approach in order to build the PaaS Layer: this approach already exploited during the INDIGO-DataCloud project is very useful to improve the flexibility in the development of the PaaS Layer, and greatly improve the collaboration among different groups of developers.

6. Work package implementation plan

6.1. Task structure

The WP is divided into three tasks:

- **T5.1 - PaaS-level Orchestration Supporting multi-IaaS Hybrid Infrastructures**: aims at realizing PaaS orchestration of resources and complex cluster of services spanning multiple hybrid IaaS through TOSCA templates. The task has two subtasks:
  - **Subtask 5.1.1 – PaaS layer provisioning**.
  - **Subtask 5.1.2 – TOSCA templates definition for complex cluster topologies**.

- **T5.2 - A smart information system to implement a fine grained IaaS description**: has the goal of creating a smart information system to describe the IaaS resources where the PaaS level services must rely on, including status and features, to be used to decide where to host a requested cluster.

- **T5.3 - High level networking orchestration to connect seamlessly to hybrid clouds**: focus on the creation of the software network layer. The task has two subtasks:
  - **Subtask 5.3.1 – Software Tools for Virtual Networking Orchestration**.
  - **Subtask 5.3.2 – Development and Standardization of the Border Router Concept**.

6.2. Coordination of activities

The coordination of this work package follows the structure defined in the Description of Work, the Grant Agreement and the Consortium Agreement documents. The WP coordinator is INFN, who oversees the activities following its progress together with the task leaders (work in each particular task is coordinated by the corresponding task leaders), coordinating the WP planning, reporting and deliverable evaluation. Regular fortnightly remote meetings of task leaders are scheduled for the whole duration of the work package, in order to follow the progresses of tasks and activities.

6.3. Task prioritization

The initial prioritization of tasks in WP5 has been done taking into account the tentative schedule for the delivery of the different solutions that comprise the DEEP as a service final product. On the
first half of the project the priority is to deliver a platform that provides easy access to execute the bulk workloads (batch executions) of the selected use cases in WP2. The second half of the project is more focused on delivering a platform (together with the implementation of a DevOps pipeline) to execute the developed applications as a service. In spite of this prioritization tasks in WP6 are scheduled, as described in the next subsection, to run during the whole lifetime of the project, due to the need of the continuous building and adaptation of the DEEP application catalogue.

In the case of a significant delay in a particular area, the Steering Committee might decide to reassign tasks to different partners, thus reducing the priority and effort on other tasks. Moreover, the continuous interaction that this work package has with WP2 and the related use cases and communities may require a review and update of this plan, resulting in a reprioritization of tasks.

6.4. Requirement tracking

In order to keep track of the user stories, the technical requirements and the involved developments to be carried out by the project JRAs, WP2 and WP3 have defined a set of processes based on the collaborative tools used within the DEEP-HybridDataCloud project [DEEP-requirement-management], Confluence and JIRA. Confluence is being used as the main source of information for the project, holding all the working documents for the project. JIRA is the issue tracker where the development activities are managed and tracked. JIRA and Confluence are tightly linked, facilitating the information flow between the two tools.

The defined processes involve the definition of product blueprints (containing specific requirements) that are linked to individual user stories using the Confluence tool. These user stories are then linked to JIRA Story issues, where the development takes place. JIRA allows the breakdown of these issues into individual working items (subsequent JIRA issues), depending on the complexity of the tasks to be performed. By using these processes, evolving user needs and requirements can be incorporated into the development cycle of the project at any stage. Consolidated reports will be generated accordingly.

6.5. Detailed workplans

This section provides a detailed work plan for the Work Package 5.

6.6. Risk assessment

There are several risks that can hinder the developments described in this document. These risks are now enumerated together with an estimated impact and the mitigation plan to minimise the negative effects.

<table>
<thead>
<tr>
<th>Risks</th>
<th>Impact</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developed software modules are not merged upstream</td>
<td>The software modules may face sustainability issues if the changes are not merged.</td>
<td>To the extent that is possible, software modules will be developed as plugins so that changes are not required to be merged upstream. Otherwise, communications with upstream software</td>
</tr>
<tr>
<td>Issue</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Upstream projects are deprecated</td>
<td>The hybrid Cloud approach solution may face sustainability issues. Developments will be based on open-source products with established communities in order to minimise this risk. In the unlikely event that an upstream project is deprecated during the course of the project, the consortium will undertake the maintenance of the upstream software for the duration of the project.</td>
<td></td>
</tr>
<tr>
<td>Partners do not deliver their contributions on-time</td>
<td>The release of the solution is delayed. Periodic meetings will be carried out among the partners, coordinated by the Work Package leader, with sufficient technical content to track the development progress and introduce corrective countermeasures should they be needed.</td>
<td></td>
</tr>
<tr>
<td>External software and artifact repositories close down</td>
<td>The development would slow down and require additional source code management system and distribution of container images. In the unlikely event that this happens, additional in-house source code management systems would be deployed as a replacement for GitHub and the Docker images stored in Docker Hub would be stored into on-premises Docker registries, in order to mitigate this risk.</td>
<td></td>
</tr>
<tr>
<td>Limited resources of specialized hardware during development</td>
<td>The solution may face this sustainability issue. The probability of this issue is low. The consortium have adequate technical capacity for increasing resources allocated to the project in case of need. The usage of specialized resources will be monitored to identify the need for additional effort.</td>
<td></td>
</tr>
<tr>
<td>Unclear requirements from use cases and user communities</td>
<td>The development would slow down and require additional communications to make clear situations. The probability of this issue is medium. The project development (in general) and WP6 development (in particular) is based on user-driven approach with close communications and interactions with WP2. Periodic meetings will be dedicated to the elicitation of requirements and clarification of required technologies.</td>
<td></td>
</tr>
<tr>
<td>Integration complexity and difficult deployment of software modules</td>
<td>The release is delayed. The probability of this issue is low. The project consortium as well as WP5 partners includes the necessary technological know-how with the detailed work plan to avoid this technical issue. The collaborations with WP2 and WP6 is carefully designed and well established.</td>
<td></td>
</tr>
<tr>
<td>Integration on the EOSC Catalogue failed</td>
<td>The support for the services improved within WP5 could not be sustained by EOSC Community. The WP5 µServices are already part of EOSC-Hub project and the INFN that is leading the WP5 is also leading the evolution of the EOSC Service catalogue. The WP management will assure that the evolution of the µServices is inline with the EOSC plan.</td>
<td></td>
</tr>
<tr>
<td>Few WP5 specific</td>
<td>The services like (SLAM, ) The services are already supported by EOSC-</td>
<td></td>
</tr>
</tbody>
</table>
services will be not fully supported by the partner that developed it

| CMDB, CloudInfoProvider) are quite important for the PaaS Layer | Hub, so the approach in WP5 will be to allow the evolution of the services in a coherent way with EOSC-Hub and DEEP-HybridDataCloud |

7. References

[INDIGO-DataCloud] INDIGO-DataCloud project, http://indigo-datacloud.eu/

[DEEP D2.1] DEEP Consortium, D2.1 - Initial Plan for Use Cases, 2018

[DEEP D3.1] DEEP Consortium, D3.1 - Initial Plan for Software Lifecycle Management,


[INDIGO TOSCA types] https://github.com/indigo-dc/tosca-types/blob/master/custom_types.yaml


[DEEP-requirement-management] https://confluence.deep-hybrid-datacloud.eu/x/KARv

8. Glossary

AAI Authentication and Authorization Infrastructure
API Application Programming Interface
BDII Berkeley Database Information Index
CLUES CLUster Elasticity System
CMDB Configuration Management Database
CMF Cloud Management Framework
CMP Container Management Platform
CPR Cloud Provider Ranker
GOCDB Grid Configuration Database
IM Infrastructure Manager
IS Information System
JRA Joint Research Activities
LRMS Local Resource Management System
REST Representational State Transfer
SLAM SLA Management Service
TOSCA Topology and Orchestration Specification for Cloud Applications
TRL Technology Readiness Level
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
<tr>
<td>YAML</td>
<td>Yet Another Markup Language</td>
</tr>
</tbody>
</table>