

# DEEP-Hybrid-DataCloud

## FIRST IMPLEMENTATION OF SOFTWARE PLATFORM FOR ACCESSING ACCELERATORS AND HPC

**DELIVERABLE: D4.2**

---

**Document identifier:** DEEP-JRA1-D4.2

**Date:** 30/10/2018

**Activity:** WP4

**Lead partner:** IISAS

**Status:** FINAL

**Dissemination level:** PUBLIC

**Permalink:** <http://hdl.handle.net/10261/168086>

---

### Abstract

This deliverable describes the first implementation of the software platform for accessing accelerators and HPC. The list of components included in the software platform is based on the analysis provided by Deliverable D4.1. This document provides detailed descriptions of software components used in the platforms, the work done on each component and its current status. Evaluation of achieved results and implementation plan for the next periods are also included.

## Copyright Notice

Copyright © Members of the DEEP Hybrid-DataCloud Collaboration, 2017-2020.

## Delivery Slip

|                    | <b>Name</b>                       | <b>Partner/Activity</b> | <b>Date</b>              |
|--------------------|-----------------------------------|-------------------------|--------------------------|
| <b>From</b>        | Viet Tran                         | IISAS / JRA1            | 19/10/2018               |
| <b>Reviewed by</b> | Marcin Plociennik<br>Germán Moltó | PSNC<br>UPV             | 22/10/2018<br>22/10/2018 |
| <b>Approved by</b> | Steering committee                |                         | 30/10/2018               |

## Document Log

| <b>Issue</b> | <b>Date</b> | <b>Comment</b>  | <b>Author/Partner</b>                          |
|--------------|-------------|---|--|
| TOC          | 01/09/2018  | Table of Content  | Viet Tran / IISAS                              |
| V0.9         | 16/10/2018  | Partner contributions                                       | WP members                                     |
| V1.0         | 19/10/2018  | Completed version for internal review                       | Viet Tran / IISAS                              |
| V1.1         | 22/10/2018  | Recommendations from reviewers                              | Marcin Plociennik / PSNC<br>Germán Moltó / UPV |
| V2.0         | 26/10/2018  | Updated version according to recommendations from reviewers | WP members                                     |
| V3.0         | 30/10/2018  | Final version   | Viet Tran / IISAS                              |

## Table of Contents

|  |    |
|--|----|
| Executive Summary.....   | 4  |
| 1. Introduction.....   | 5  |
| 2. Description of software platform for accessing accelerators and HPC.....          | 5  |
| 2.1. Overview.....   | 5  |
| 2.2. Detailed component description.....   | 7  |
| 2.2.1. udocker.....  | 8  |
| 2.2.2. OpenStack.....  | 9  |
| 2.2.3. OpenNebula.....   | 12 |
| 2.2.4. Apache Mesos.....   | 13 |
| 2.2.5. Kubernetes.....   | 20 |
| 2.2.6. Integration of HPC resources with DEEP platform.....                          | 22 |
| 2.2.7. Tools for accessing HPC resources.....  | 24 |
| 2.2.8. Virtual HPC clusters.....   | 26 |
| 3. First implementation of software platform for accessing accelerators and HPC..... | 28 |
| 3.1. udocker.....  | 28 |
| 3.2. OpenStack.....  | 29 |
| 3.3. OpenNebula.....   | 29 |
| 3.4. Mesos.....  | 29 |
| 3.5. Kubernetes.....   | 31 |
| 3.6. HPC.....  | 32 |
| 4. Next steps.....   | 33 |
| 4.1. udocker.....  | 33 |
| 4.2. OpenStack.....  | 34 |
| 4.3 OpenNebula.....  | 34 |
| 4.3. Mesos.....  | 34 |
| 4.5 Kubernetes.....  | 34 |
| 4.6 HPC.....   | 35 |
| 5. Conclusion.....   | 36 |
| 6. Glossary.....   | 37 |
| 7. References.....   | 39 |

## Executive Summary

The DEEP-HybridDataCloud (Designing and Enabling E-Infrastructures for intensive data Processing in a Hybrid DataCloud) is a project approved in July 2017 within the EINFRA-21-2017 call of the Horizon 2020 framework program of the European Community. It will develop innovative services to support intensive computing techniques that require specialized HPC hardware, such as GPUs or low-latency interconnects, to explore very large datasets.

This deliverable describes the first implementation of the software platform for accessing accelerators and HPC. Components included in the software platform are selected on the analysis provided by Deliverable D4.1. For each component, its detailed description, current status, and next implementation plan are presented.

From D4.1, two implementations of container technologies have been selected to be further improved: udocker, as container technology to be used in the case of HPC platforms, and nova-lxd, as a replacement of full hypervisors. In the first implementation of the software platform, support for automatic passing of GPU drivers in udocker has been added. Tests have been carried out with common GPU-enabled Docker containers and they proved that the performance of the GPU-enabled version of udocker is comparable with baremetal. Next work on udocker will be focused on supporting fast interconnection network and compatibility.

OpenStack nova-lxd has been deployed and extensively tested in this implementation of the software platform. It has different compatibility issues and documentation is severely missing. The work done in the first implementation is to create a working version of OpenStack with nova-lxd, integrate it into the DEEP testbed and update documentation. Next work on nova-lxd will be to add GPU support and improve compatibility with other components, mainly block storages.

Support for accelerators in the first software platform is not limited only in the mentioned udocker and nova-lxd, but also in other Cloud middleware, most notably Mesos and Kubernetes. The Mesos schedulers, Marathon and Chronos, have been modified to provide support for GPU. The GPU support for Kubernetes has been also thoroughly tested and documented in a series of How-to articles. Both middleware have been integrated with DEEP IAM and the PaaS orchestration from WP5. The next implementation plan will be to improve compatibility with OpenID connect and better integration with the PaaS orchestrator. For completeness, OpenStack and OpenNebula middleware with KVM PCI-passthrough are included into the first software platform, too.

HPC integration with PaaS approach has been analyzed carefully. Different approaches have been proposed and, finally, the approach using common gateway and token translator for SSH keys has been chosen. The work in the next period will be focused on implementation of the HPC integration with PaaS orchestrator using the proposed approaches and tools.

## 1. Introduction

The DEEP-HybridDataCloud (Designing and Enabling E-Infrastructures for intensive data Processing in a Hybrid DataCloud) is a project approved in July 2017 within the EINFRA-21-2017 call of the Horizon 2020 framework program of the European Community. It will develop innovative services to support intensive computing techniques that require baremetal performance, specialized accelerators such as GPUs and HPC platform in the Cloud, to explore very large datasets.

The objective of WP4 of the DEEP-Hybrid-DataCloud project is to fulfil the requirements mentioned above by working as closely as possible with hardware resources, exploiting the full potential of computational performance provided by the hardware including accelerators, low-latency interconnects and HPC platforms. This workpackage is cooperating tightly with WP5 High Level Hybrid Cloud solutions where the resources provided in this workpackage will be managed and accessed via the PaaS Orchestration service provided by WP5.

This deliverable describes the first implementation of the software platform for accessing accelerators and HPC. The list of components included in the software platform is based on the analysis provided by Deliverable D4.1. This document provides detailed descriptions of software components used in the platform, the work done on each component, and its current status. Evaluation of achieved results and implementation plan for the next periods are also included.

## 2. Description of software platform for accessing accelerators and HPC

### 2.1. Overview

The first software platform in WP4 is focusing on support for containers and accelerators in Cloud and HPC environments. As it has been analyzed in D4.1, we will focus on using udocker for supporting containers on HPC platforms. On the Cloud platforms, we will use nova-lxd plugin with traditional OpenStack Cloud middleware, and the built-in container supports in Kubernetes and Mesos.

The access to accelerators will be supported on all platforms mentioned above. The first implementation of GPU support has been created in udocker within this project. Experimental GPU supports in Kubernetes and Mesos via nvidia-docker runtime have been deployed and integrated into the DEEP testbed. The GPU support for nova-lxd is planned in the next period.

For completeness of the pictures, GPU supports via KVM PCI-passthrough are added in the first software platform as they are being used on some sites in the testbed as a preliminary solution to access accelerators. However, due to several technical issues of PCI-passthrough approach, that have been analyzed in D4.1, we will not invest efforts in improvements of the technology, but only follow its evolution with possible testing/deployment in the DEEP project.

The software platform should be accessed by end-users via the DEEP as a service from WP6 and the PaaS orchestration services from WP5. The overall architectures of components are shown in Fig.1.

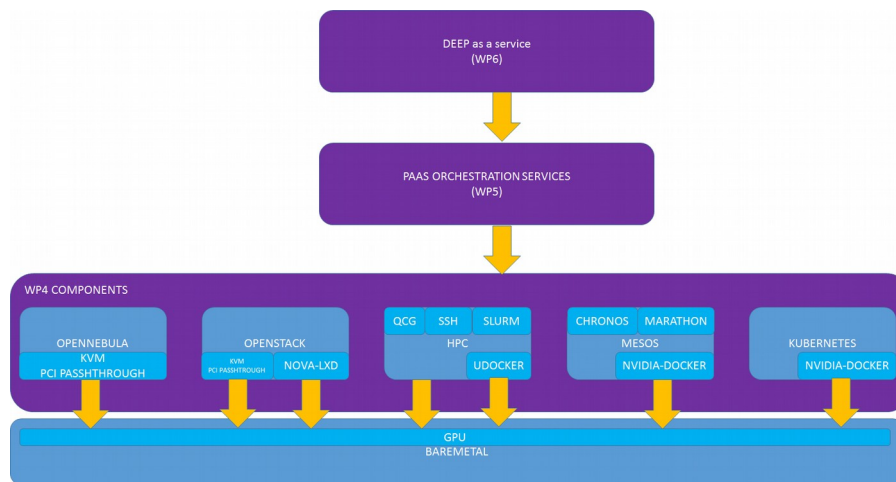


Fig. 1 Architecture of components in WP4

It is worth to mention that some components, specifically Kubernetes, Mesos, and HPC clusters can be deployed in the Cloud instead of on baremetal by the PaaS orchestrator (Fig. 2). In that case, the PaaS orchestration layer takes care of the provisioning of the Cloud resources and of the installation and configuration of the software components. Then it is possible to submit the application services (e.g. DEEPaaS) to the newly deployed components. The software inside the components deployed in the Cloud should be the same as the components deployed on baremetal, including supports for containers and accelerators.

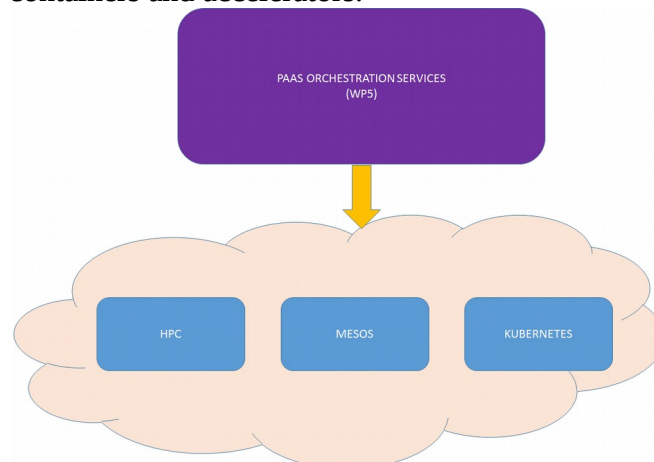


Fig. 2 Components deployed in Cloud via PaaS orchestration services

## 2.2. Detailed component description

The template shown below will be used as the structure for including a complete description of each component in the following submissions.

|                                |   |
|--------------------------------|---|
| <b>Identification</b>          | The unique name for the component and its location in the system  |
| <b>Type</b>                    | A module, a subprogram, a data file, a control procedure, a class, etc.   |
| <b>Purpose</b>                 | 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.   |
| <b>Function</b>                | 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.   |
| <b>High level architecture</b> | The internal structure of the component, its constituents, and the functional requirements satisfied by each part.  |
| <b>Dependencies</b>            | 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. |
| <b>Interfaces</b>              | 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.               |
| <b>Data</b>                    | 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.   |
| <b>Needed improvement</b>      | Description of the needed improvements of this tool with regards the DEEP-Hybrid-DataCloud objectives, in order to fulfill the user requirements and to build the DEEP as a Service functionality.  |
| <b>Current TRL status</b>      | A detailed description of the status of the Technology Readiness Level, for specific set of features of the services.   |
| <b>Expected TRL evolution</b>  | A detailed description of the Technology Readiness Level that DEEP foreseen to reach for a specific set of features of the services.  |

## 2.2.1.udocker

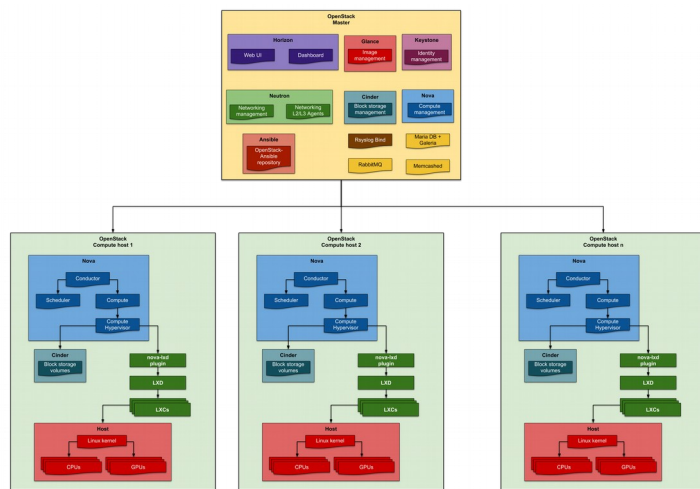
|                                |   |
|--------------------------------|---|
| <b>Identification</b>          | udocker   |
| <b>Type</b>                    | Python client tool  |
| <b>Purpose</b>                 | Tool to execute simple Docker containers in user space without requiring root privileges.   |
| <b>Function</b>                | Enables download and execution of Docker containers by non-privileged users in Linux systems where Docker is not available. It can be used to pull and execute Docker containers in Linux batch systems and interactive clusters that are managed by other entities such as grid infrastructures or externally managed batch or interactive systems.  |
| <b>High level architecture</b> | <p>udocker implements the following commands:</p> <pre> search &lt;repo/image:tag&gt;           :Search Docker Hub for container images pull &lt;repo/image:tag&gt;             :Pull container image from Docker Hub images                             :List container images create &lt;repo/image:tag&gt;           :Create container from a pulled image ps                                 :List created containers rm &lt;container&gt;                    :Delete container run &lt;container&gt;                   :Execute container inspect &lt;container&gt;               :Low level information on container name &lt;container_id&gt; &lt;name&gt;       :Give name to container rmname &lt;name&gt;                     :Delete name from container  rmi &lt;repo/image:tag&gt;              :Delete image rm &lt;container-id&gt;                 :Delete container import &lt;tar&gt; &lt;repo/image:tag&gt;     :Import tar file (exported by Docker) import - &lt;repo/image:tag&gt;         :Import from stdin (exported by Docker) load -i &lt;exported-image&gt;         :Load image from file (saved by Docker) load                               :Load image from stdin (saved by Docker) export -o &lt;tar&gt; &lt;container&gt;       :Export container rootfs to file export - &lt;container&gt;              :Export container rootfs to stdin inspect &lt;repo/image:tag&gt;         :Return low level information on image verify &lt;repo/image:tag&gt;          :Verify a pulled image clone &lt;container&gt;                 :duplicate container  protect &lt;repo/image:tag&gt;          :Protect repository unprotect &lt;repo/image:tag&gt;        :Unprotect repository protect &lt;container&gt;              :Protect container unprotect &lt;container&gt;            :Unprotect container  mkrepo &lt;topdir&gt;                   :Create repository in another location setup                             :Change container execution settings login                              :Login into Docker repository logout                             :Logout from Docker repository  version                            :Shows udocker version and exits  help                               :This help run --help                         :Command specific help </pre> |
| <b>Dependencies</b>            | Written in Python, it has a minimal set of dependencies so that can be executed in a wide range of Linux systems.   |
| <b>Interfaces</b>              | Not applicable  |
| <b>Data</b>                    | Not applicable  |



|                               |  |
|-------------------------------|--|
| <b>Needed improvement</b>     | <p>Automatic use of GPUs: implement code that allows the use of GPUs without the user needing to install and match the NVIDIA drivers in the image or containers.</p> <p>Increase automation of using low latency interconnects such as Infiniband</p> |
| <b>Current TRL status</b>     | <p>TRL 9 - this refers to the current version</p> <p>TRL 6 - support for GPUs and low latency interconnects</p>  |
| <b>Expected TRL evolution</b> | <p>TRL 8 or 9 - support for GPUs and low latency interconnects</p>   |

### 2.2.2. OpenStack

|                       |  |
|-----------------------|--|
| <b>Identification</b> | <p>OpenStack</p>   |
| <b>Type</b>           | <p>A middleware (set of daemons/services)</p>  |
| <b>Purpose</b>        | <p>OpenStack provides management of Cloud resources. OpenStack is able to run on bare metal and/or LXC containers.</p>   |
| <b>Function</b>       | <p>Deployment of virtual machines and other instances into an environment which also contains LXC containers. Because one of the main functional requirements is GPU support, version of LXC and LXD has to be 3.X at least.</p> |

|                                |   |
|--------------------------------|---|
| <b>High level architecture</b> | <div data-bbox="571 1178 1273 1664" data-label="Diagram">  </div> <p>The high-level architecture of the OpenStack cloud environment (depicted below) is composed by:</p> <ul style="list-style-type: none"> <li> <b>Master node:</b> It manages all slave nodes from the OpenStack cloud environment. For the sake of simplification and clarity, the environment has only one master node (it is not a problem to handle the extended case if it will be needed for example due to a better load         </li> </ul> |
|--------------------------------|---|

balance). The master node is an All-in-One installation of OpenStack with limited Nova module and Cinder module. Those modules are responsible for a (global) resource management only. The computing and storage resources are offered by host nodes.

- **Host nodes:** They are registered/recognized by the master node and offering their computing and storage resources to it. According to it, they have installed nova module, cinder module, and nova-lxd plugin.

The main components of an OpenStack environment are:

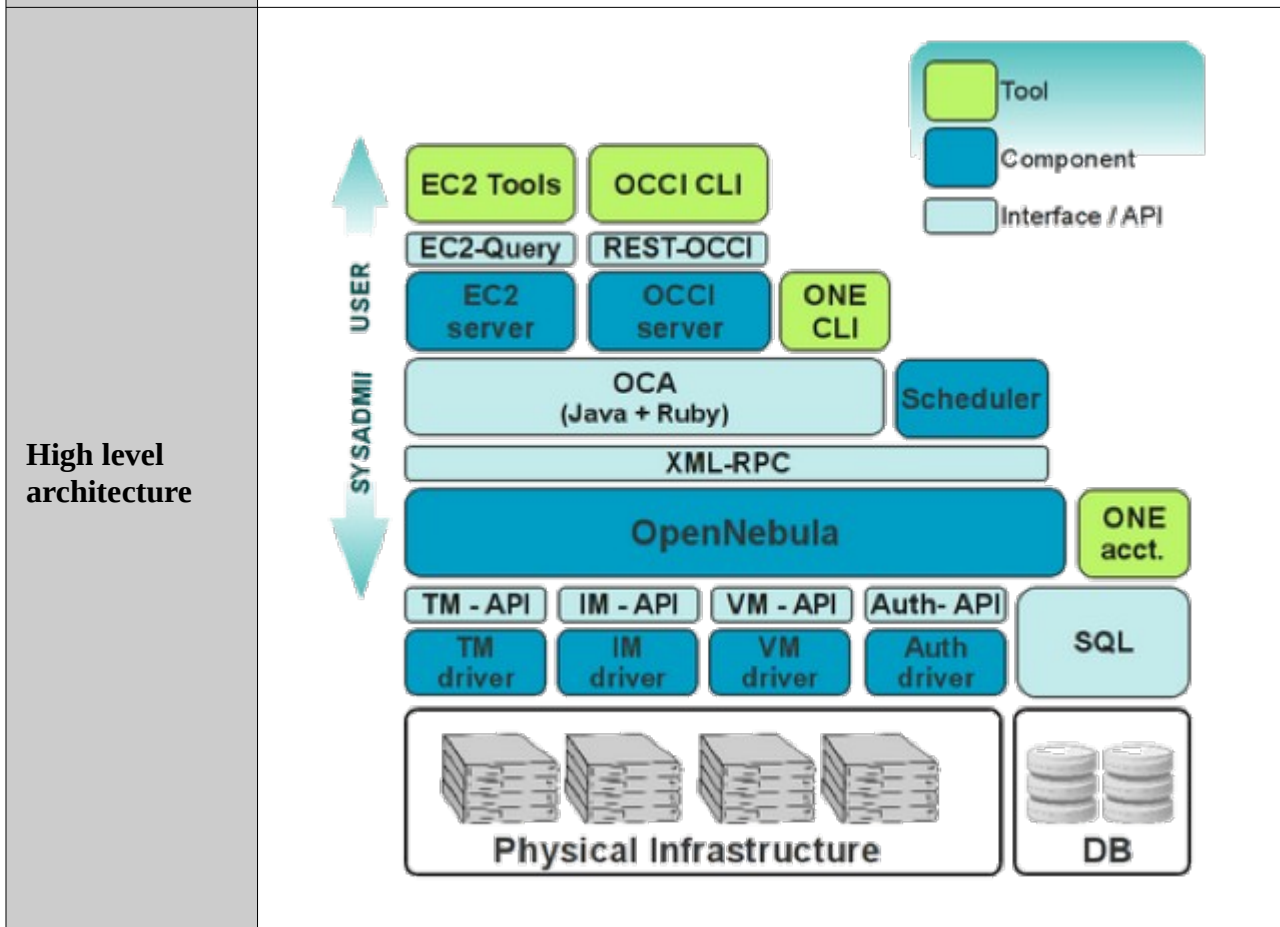
- **Horizon:** OpenStack module responsible for an OpenStack Dashboard. It provides administrators and users with a graphical interface to access, provision, and automate deployment of cloud-based resources.
- **Glance:** OpenStack module for an image management. It provides discovery, registration, and delivery services for disk and server images. Stored images can be used as a template. It can also be used to store and catalog an unlimited number of backup.
- **Keystone:** OpenStack module for an identity management. It provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services
- **Neutron:** OpenStack networking module managing networks and IP addresses. It ensures the network is not a bottleneck or limiting factor in a cloud deployment,[citation needed] and gives users self-service ability, even over network configurations. Neutron provides networking models for different applications or user groups. The module manages IP addresses, allowing for dedicated static IP addresses or DHCP, and associating floating IP addresses let traffic be dynamically rerouted to any resources in the IT infrastructure, so users can redirect traffic during maintenance or in case of a failure.
- **Cinder:** OpenStack module for management of block storage. It provides persistent block-level storage devices for use with OpenStack compute instances. The block storage system manages the creation, attaching and detaching of the block devices to servers. Block storage volumes are fully integrated into OpenStack Compute and the Dashboard allowing for cloud users to manage their own storage needs.
- **Nova:** OpenStack Compute module. It is a cloud computing fabric controller, which manages and automates pools of computer resources and can work with widely available virtualization

|                     |   |
|---------------------|---|
|                     | <p>technologies, as well as bare metal and high-performance computing (HPC) configurations. There are many options for hypervisor technology (virtual machine monitor), we choose nova-lxd (which gives us GPU support in the OpenStack cloud environment).</p> <p>Supporting tools used by an OpenStack cloud environment are:</p> <ul style="list-style-type: none"> <li>• <b>Ansible</b> is open source software that automates software provisioning, configuration management, and application deployment. OpenStack-Ansible provides Ansible playbooks and roles for the deployment and configuration of an OpenStack environment.</li> <li>• <b>Rsyslog</b> (the Rocket-fast SYStem for LOG processing) is an open source software used on UNIX and Unix-like computer systems for forwarding log messages in an IP network. It implements the basic syslog protocol, extends it with content-based filtering, rich filtering capabilities, flexible configuration options and adds important features such as using TCP for transport.</li> <li>• <b>Maria DB</b> is a community-developed fork of the MySQL relational database management system intended to remain free under the GNU GPL. <b>Galera Cluster</b> is a synchronous multi-master database cluster, based on synchronous replication and Oracle’s MySQL/InnoDB.</li> <li>• <b>RabbitMQ</b> is an open source message broker software (sometimes called message-oriented middleware) that originally implemented the Advanced Message Queuing Protocol (AMQP) and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol (STOMP), Message Queuing Telemetry Transport (MQTT), and other protocols.</li> <li>• <b>Memcached</b> is a general-purpose distributed memory caching system. It puts caching data and objects in RAM to reduce the number of times an external data source must be read.</li> </ul> |
| <b>Dependencies</b> | Linux system with Python packages and Ansible (supporting tools used by an OpenStack cloud environment - Rsyslog, MariaDB, Galeria, RabbitMQ, and Memcached are installed/configured by Ansible).   |
| <b>Interfaces</b>   | <p>It has three main interfaces:</p> <ul style="list-style-type: none"> <li>• Web UI, and dashboard (it is provided by a Horizon module)</li> <li>• REST API</li> <li>• CLI</li> </ul>  |
| <b>Data</b>         | JSON messages, configuration files, images, volumes, and snapshots of virtual machines.   |
| <b>Needed</b>       | Simplification of nova-lxd plugin installation/configuration  |

|                               |   |
|-------------------------------|---|
| <b>improvement</b>            | Adding GPU support to nova-lxd  |
| <b>Current TRL status</b>     | TRL 9 - OpenStack system.<br>TRL 6 - nova-lxd plugin and its integration into OpenStack environment |
| <b>Expected TRL evolution</b> | TRL 8 for nova-lxd<br>TRL 8 for accelerators support in nova-lxd                                    |

### 2.2.3. OpenNebula

|                       |  |
|-----------------------|--|
| <b>Identification</b> | OpenNebula   |
| <b>Type</b>           | A middleware (set of daemons/services) for Cloud infrastructure management |
| <b>Purpose</b>        | Cloud management software stack  |
| <b>Function</b>       | Deployment and management of Virtual Machines on physical infrastructures. |



|                               |   |
|-------------------------------|---|
| <b>Dependencies</b>           | Linux systems   |
| <b>Interfaces</b>             | Web-based GUI<br>CLI<br>REST interfaces: OCA (OpenNebula Cloud API), EC2, rOCCI |
| <b>Data</b>                   | Application's internal data stored in SQL database and accessed via CLI.        |
| <b>Needed improvement</b>     | GPU support unsatisfactory, only KVM PCI-passthrough supported                  |
| <b>Current TRL status</b>     | TRL 9 for the system itself, TRL 6-7 for GPU support.                           |
| <b>Expected TRL evolution</b> | TRL 9 for the system itself, TRL 8 for selected GPU configuration               |

#### 2.2.4. Apache Mesos

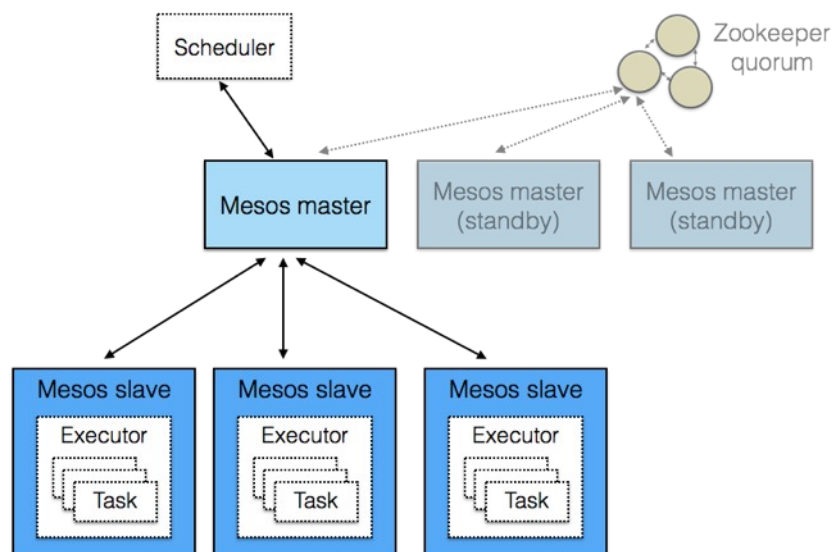
|                       |   |
|-----------------------|---|
| <b>Identification</b> | Apache Mesos [10]   |
| <b>Type</b>           | A middleware (set of daemons/services) for cluster management   |
| <b>Purpose</b>        | Mesos provides efficient resource isolation and sharing across distributed applications (frameworks).<br>Mesos can run on top of Virtual Machines and/or bare metal and/or Docker containers.   |
| <b>Function</b>       | Mesos requires computing resources to be assigned to it, so that it can deploy distributed applications on these resources.<br>Mesos provides the following main functionalities: <ul style="list-style-type: none"> <li>• <b>resource allocation/revocation/re-allocation:</b> Mesos implements a pluggable resource allocation module architecture that allows users to create allocation policies with algorithms that best fit a particular deployment. By default, Mesos includes a strict priority resource allocation module and a modified fair sharing resource allocation module. Resources are generally reallocated when tasks end. However, if a task takes too long, it can be killed to reallocate.</li> <li>• <b>performance isolation</b> among framework executors running on the same slave node through pluggable isolation modules. The default mechanism leverages the Linux container technologies.</li> <li>• <b>framework authorization</b> implemented through configurable ACLs in JSON format that allow 1) frameworks to (re-) register with authorized roles; 2) frameworks to launch tasks/executors as</li> </ul> |

authorized users; 3) Authorized users to shutdown framework(s) through “/shutdown” HTTP endpoint

- **framework rate-limiting** that allows to configure the maximum number of queries per seconds for each framework. This feature aims at protecting the throughput of high-SLA frameworks by having the master throttle messages from other (e.g., development, batch) frameworks.
- **monitoring**: Mesos master and slave nodes report a set of statistics and metrics including details about available resources, used resources, registered frameworks, active slaves, and task state. These metrics are available querying the http endpoints exposed by the master and slave nodes.
- **slave recovery**: this feature allows 1) Executors/tasks to keep running when the slave process is down and allows 2) a restarted slave process to reconnect with running executors/tasks on the slave.
- **native Docker support** that allows users to launch a Docker image as a Task, or as an Executor.

### High level architecture

The high-level architecture of Mesos is depicted below:



The two main components are:

- Mesos **master**
- Mesos **slave**

The Mesos *master* is a daemon that manages *slave* daemons running on each cluster node. The high-availability and fault-tolerance of the master can be achieved using multiple masters and a quorum manager like Zookeeper. Moreover, the master is designed as a soft-state component and is able to

|                           |  |
|---------------------------|--|
|                           | <p>rebuild its internal state from the messages periodically sent by the slaves and from the framework schedulers. The slaves register with the master and offer “resources” i.e. capacity to be able to run tasks.</p> <p>The Executor is responsible for launching tasks in a framework specific way (i.e., creating new threads, new processes, etc). One or more executors from the same framework may run concurrently on the same machine. A dedicated class, “MesosExecutorDriver”, is used both to manage the Executor’s lifecycle (start it, stop it, or wait for it to finish) and to connect the Framework Executor to Mesos.</p> <p>Mesos provides the <i>Scheduler</i> interface to be implemented by each specific framework; this interface includes methods to register, re-register, unregister with the Mesos master and to accept or reject resource offers. The master decides how many resources must be offered to each framework scheduler according to a given organizational policy, such as fair sharing, or strict priority. To support a diverse set of policies, the master employs a modular architecture that makes it easy to add new allocation modules via a plugin mechanism.</p> |
| <b>Dependencies</b>       | <p>Mesos depends on:</p> <ul style="list-style-type: none"> <li>• Zookeeper servers for leader election in high-availability mode;</li> </ul>  |
| <b>Interfaces</b>         | <p>Mesos handles the following main interfaces:</p> <ul style="list-style-type: none"> <li>• <b>Configuration files</b></li> <li>• <b>REST API:</b> the Mesos masters and slaves provide a handful of REST endpoints that can be useful for users and operators.</li> <li>• <b>Web UI:</b> Mesos provides a simple web interface to monitor the cluster state</li> </ul>   |
| <b>Data</b>               | <p>The sandbox is a special directory on each slave node that acts as the execution environment (from a storage perspective) and also contains relevant log files as well as stderr and stdout for the command being executed.</p>   |
| <b>Needed improvement</b> | <p>The integration of Mesos clusters with the PaaS requires that the user requests can be authenticated/authorized via oauth tokens issued by INDIGO IAM. Openid-Connect/OAuth2.0 is not natively supported by Apache Mesos; therefore, we need to explore and investigate ways for enabling it on top of Mesos.</p> <p>Moreover, in order to support the distributed deep-learning applications developed by the project use-cases, it is important to test and verify the support for accelerators in Mesos. The NVIDIA GPU support was introduced since version 1.0 and mimics the support provided by <i>nvidia-</i></p>   |

|                               |   |
|-------------------------------|---|
|                               | <p><i>docker</i> to automatically mount the proper NVIDIA drivers and tools directly into the Docker container. We will provide guides and recipes to install and configure the Mesos components automatically (including the GPU support) in order to ease their deployment at the sites providing infrastructure-level services. GPU sharing and support for RDMA-capable networking (Infiniband) will be investigated as well and, if needed, the missing support will be developed.</p> |
| <b>Current TRL status</b>     | <p>TRL6 for accelerators support<br/>TRL8-9 for the other functions</p>   |
| <b>Expected TRL evolution</b> | <p>TRL8 for accelerators support</p>  |

|                       |   |
|-----------------------|---|
| <b>Identification</b> | <p>Marathon [11]</p>  |
| <b>Type</b>           | <p>Apache Mesos Framework for long-running services</p>   |
| <b>Purpose</b>        | <p>Marathon allows to deploy and execute long-running services on top of a Mesos cluster, providing advanced features like fault-tolerance, high-availability, application dependencies, etc.</p>   |
| <b>Function</b>       | <p>Marathon provides the following main functionalities:</p> <ul style="list-style-type: none"> <li>• <b>Basic Authentication and SSL:</b> it is possible to secure Marathon API endpoints via SSL and limit access to them with HTTP basic access authentication.</li> <li>• <b>Application scaling:</b> users can request to increase/decrease the number of instances of their applications using the API endpoint or the web UI;</li> <li>• <b>Application high-availability:</b> Marathon implements an automated hardware and software failure handling. For example, in case of a node failure, the application is automatically re-scheduled and started on another node.</li> <li>• <b>Health Checks:</b> check your application's health via HTTP or TCP checks.</li> <li>• <b>Application dependencies:</b> Marathon allows to group your applications. Groups make it easy to manage logically related applications and allow you to perform actions on the entire group, such as scaling or restart.</li> <li>• <b>Placement constraints:</b> constraints in Marathon allow restricting where a particular application runs, and thus, you can benefit from</li> </ul> |



|                                       |  |
|---------------------------------------|--|
|                                       | <p>the locality, fault tolerance, or any other constraint.</p> <ul style="list-style-type: none"> <li>• <b>Event Bus Subscription:</b> Marathon's internal bus captures all the events of Marathon, API requests, scaling events, and so on. This can be used to integrate load balancers, monitoring, and other external systems with Marathon.</li> <li>• <b>Metrics:</b> available at /metrics in JSON format</li> <li>• <b>Web UI:</b> allows to deploy and monitor the applications using a very user-friendly interface</li> <li>• <b>JSON/REST API</b> for easy integration and scriptability</li> </ul>  |
| <p><b>High level architecture</b></p> | <p>Marathon implements the Mesos Framework interfaces; it consists of two components: the scheduler and the executors. The scheduler is responsible for coordinating the execution. The executor provides the ability to control the task execution.</p>   |
| <p><b>Dependencies</b></p>            | <p>Marathon behavior strictly depends on the Mesos components (see the diagram below): Marathon Scheduler will be offered resources by the Mesos Master and will accept or reject them; the Scheduler will call the Mesos Master to launch its tasks when the resource offers satisfy their requirements.</p> <div data-bbox="619 1146 1184 1668" data-label="Diagram"> <pre> graph TD     Scheduler[Scheduler]     MesosMaster[Mesos Master]     MesosSlave[Mesos Slave]     Executor[Executor]     Task1[Task]     Task2[Task]      MesosMaster -- "1. resourceOffer()<br/>5. statusUpdate()" --&gt; Scheduler     Scheduler -- "2. launchTasks()" --&gt; MesosMaster     MesosMaster &lt;--&gt; MesosSlave     MesosSlave &lt;--&gt; Executor     Executor -- "3. launchTasks()" --&gt; Task1     Executor -- "4. statusUpdate()" --&gt; Task2     Task1 &lt;--&gt; Executor     Task2 &lt;--&gt; Executor     </pre> </div> <p>Marathon Task Executor will interact with the Mesos slave agent to perform the execution of the real tasks and monitor them.</p> <p>Optionally, the Marathon executor delegates the local Docker daemon for image discovery and management when the user requests to launch a dockerized application.</p> |
| <p><b>Interfaces</b></p>              | <p>The main interfaces of Marathon are:</p>  |

|                               |  |
|-------------------------------|--|
|                               | <ul style="list-style-type: none"> <li>• configuration files</li> <li>• Artifact Store: The artifact store is the location for storing application-specific resources for deployment that an application needs in order to run, like certain files, for example.</li> <li>• Docker Registry: in case of Docker-based application, Marathon executor interacts with the Docker Registry (optionally the registry can be a private Docker Registry) to pull the container images into the slave.</li> <li>• Rest API: Marathon provides a rich set of RESTful API to manage the users' application deployment (start, restart, stop, scale, monitor, etc.).</li> <li>• Web UI: it is a very convenient interface with Marathon. It allows to deploy and monitor the applications.</li> </ul> |
| <b>Data</b>                   | Marathon handles Application definition files in JSON format   |
| <b>Needed improvement</b>     | <p>The improvements foreseen for Mesos apply to Marathon:</p> <ul style="list-style-type: none"> <li>• enable the missing support for OAuth2.0 providing guidelines and configurations;</li> <li>• conduct further tests on accelerators (GPUs, Infiniband) support by running the project deep-learning training applications</li> <li>• provide recipes for the automatic deployment and configuration of the components</li> </ul>  |
| <b>Current TRL status</b>     | <p>TRL6 for accelerators support</p> <p>TRL8-9 for the other functions</p>   |
| <b>Expected TRL evolution</b> | TRL8 for accelerators support  |

|                       |  |
|-----------------------|--|
| <b>Identification</b> | Chronos [12]   |
| <b>Type</b>           | Apache Mesos Framework for recurring job execution   |
| <b>Purpose</b>        | Chronos allows to schedule jobs using ISO8601 repeating interval notation. Typical jobs that do not need to always be running but have to be triggered at a particular time repeatedly are backups, Extract-Transform-Load (ETL) jobs, running other frameworks, and so on. Chronos allows also to run complex jobs pipelines; a typical use-case is the processing of a data set. |
| <b>Function</b>       | Chronos provides the following main functionalities:   |

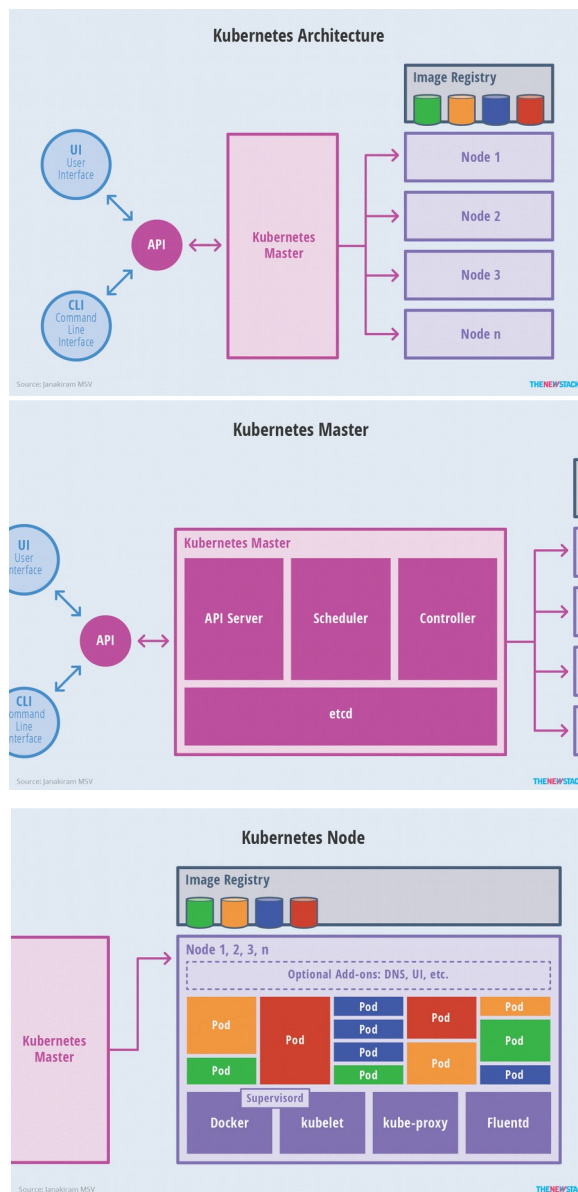
|                                |  |
|--------------------------------|--|
|                                | <ul style="list-style-type: none"> <li>• <b>Basic Authentication and SSL:</b> it is possible to secure Chronos API endpoints via SSL and limit access to them with HTTP basic access authentication.</li> <li>• <b>Job dependencies:</b> Chronos supports the definition of jobs triggered by the completion of other jobs, and it also supports arbitrarily long dependency chains.</li> <li>• <b>Handling of job failures and configurable retries:</b> Chronos allows to specify the number of retries to attempt if a command returns a non-zero exit-code;</li> <li>• <b>Job History</b> (e.g. job duration, start time, end time, failure/success)</li> <li>• <b>Fault Tolerance</b></li> <li>• <b>Native Docker support:</b> Chronos allows to easily schedule Docker containers to run jobs. Launching a Docker container is as simple as performing a POST request to the Chronos scheduler.</li> <li>• <b>Web UI</b> can be used to add, delete, list, modify and run jobs. It can also show a graph of job dependencies.</li> <li>• <b>REST API</b> for the entire job management and monitoring. JSON is used to describe the jobs. Chronos REST endpoint can return the jobs graph in the form of a DOT file</li> </ul> |
| <b>High level architecture</b> | <p>Chronos implements the Mesos Framework interfaces; it consists of two components: the scheduler and the executors. The scheduler is responsible for coordinating the execution. The executor provides the ability to control the job execution.</p>   |
| <b>Dependencies</b>            | <p>Like Marathon, the behaviour of Chronos strictly depends on the Mesos components: Chronos Scheduler will be offered resources by the Mesos Master and will accept or reject them; the Scheduler will call the Mesos Master to launch its tasks when the resource offers satisfy their requirements. Chronos Task Executor will interact with the Mesos slave agent to perform the real task execution and monitor them.</p> <p>Other dependencies:</p> <ul style="list-style-type: none"> <li>• Zookeeper servers for storing state (for high-availability)</li> </ul>  |
| <b>Interfaces</b>              | <p>The main interfaces of Chronos are:</p> <ul style="list-style-type: none"> <li>• configuration files</li> <li>• Docker Registry: in case of Docker-based job, Chronos executor interacts with the Docker Registry (optionally the registry can be a Private Docker Registry) to pull the container images into the slave.</li> <li>• RESTful API: Chronos provides a rich set of RESTful API to add, schedule and monitor the user's jobs;</li> </ul>   |

|                               |  |
|-------------------------------|--|
|                               | <ul style="list-style-type: none"> <li>• Web UI: it is a very convenient interface with Chronos. It allows to add, schedule and monitor the jobs in a very simple way. Moreover, it provides a graph of the jobs dependencies.</li> </ul>  |
| <b>Data</b>                   | Chronos handles the job definition file in JSON format   |
| <b>Needed improvement</b>     | <p>The improvements foreseen for Mesos apply to Chronos:</p> <ul style="list-style-type: none"> <li>• enable the missing support for OAuth2.0 providing guidelines and configurations;</li> <li>• conduct further tests on accelerators (GPUs, infiniband) support by running the project deep-learning training applications;</li> <li>• provide recipes for the automatic deployment and configuration of the components</li> <li>• Unlike Marathon, the GPU support in Chronos is not officially documented and included in the mainstream: there is a pull request [6] that is in good shape but it has not been merged yet since some unit tests required by the review process have not been provided yet. We will verify the correct functioning of the patch and eventually provide the needed tests for including the patch in the mainstream version.</li> </ul> |
| <b>Current TRL status</b>     | <p>TRL6 for accelerators support</p> <p>TRL8 for the other functions</p>   |
| <b>Expected TRL evolution</b> | TRL8   |

### 2.2.5. Kubernetes

|                                |  |
|--------------------------------|--|
| <b>Identification</b>          | Kubernetes   |
| <b>Type</b>                    | An open-source system for managing containerized applications across multiple hosts in a cluster   |
| <b>Purpose</b>                 | Kubernetes provides a platform for automating deployment, scaling, and operations of application containers across clusters of hosts.  |
| <b>Function</b>                | Kubernetes provides mechanisms for application deployment, scheduling, updating, maintenance, and scaling. A key feature of Kubernetes is that it actively manages the containers to ensure that the state of the cluster continually matches the user's intentions. |
| <b>High level architecture</b> | Kubernetes implements a client-server architecture. The <b>master</b> server consists of various components including a <i>kube-apiserver</i> , an <i>etcd</i> storage, a <i>kube-controller-manager</i> , a <i>kube-scheduler</i> , and a DNS server for Kubernetes |

services. **Node** components include *kubelet* and *kube-proxy* on top of Docker.



**Dependencies**

The main dependencies are:

- etcd
- container runner: Docker or rkt

Advanced features may require the interaction with other external components, for example:

- enabling persistent storage requires integration with external storage services (NFS, Cloud block storage services, etc.)

|                               |  |
|-------------------------------|--|
|                               | <ul style="list-style-type: none"> <li>• load-balancing can require external haproxy or nginx or Cloud load-balancer</li> <li>• specific networking: many plugins are available, e.g. calico, weave, flannel, open-vswitch, ect.</li> <li>• GPU support requires nvidia-docker installed on the nodes</li> </ul>   |
| <b>Interfaces</b>             | <ul style="list-style-type: none"> <li>• <b>Kubernetes CLI interface:</b> Kubernetes offers a command line tool named kubectl and it controls the Kubernetes cluster manager.</li> <li>• <b>Kubernetes User Interface:</b> The Kubernetes UI can be used to introspect the cluster, such as checking how resources are used, or looking at error messages. It cannot be used to modify the cluster.</li> <li>• <b>Kubernetes APIs</b></li> </ul>   |
| <b>Data</b>                   | etcd is a highly-available key/value store which Kubernetes uses for persistent storage of all of its REST API objects   |
| <b>Needed improvement</b>     | <p>The adoption of Kubernetes in the DEEP project required the integration with the INDIGO IAM authentication/authorization system. The support for OpenID-Connect in Kubernetes is officially documented but we needed to perform tests in order to verify the correct integration with INDIGO IAM.</p> <p>Moreover, in order to meet the project requirements, the support for GPUs was required. Therefore, the evaluation and testing of such support was carried out and the outcomes of this activities have been documented in order to provide guides and procedures for the resource providers willing to deploy a Kubernetes cluster for the DEEP users.</p> |
| <b>Current TRL status</b>     | <p>TRL6 for GPU support</p> <p>TRL8 for the other features</p>   |
| <b>Expected TRL evolution</b> | TRL8   |

### 2.2.6. Integration of HPC resources with DEEP platform

Adoption of Cloud computing for running HPC workloads is more and more popular and most of the Cloud providers are able to provide virtual instances with accelerators and low latency interconnection network. Several benefits can come from this integration, which facilitates both workload management and interaction with remote resources. Exposing HPC resources to the end-users through the Platform-as-a-Service layer aims to lower barriers in accessing HPC environments. This way of accessing HPC resources can hide the technical details of accessing

resources and therefore can be especially advantageous for some non-IT specialized groups of user and for some parallel applications

Compared to traditional HPC environments, Clouds users can easily adjust their resource pools, via a mechanism known as elasticity, and optimize resource utilization. These features help adding more flexibility to HPC applications but require a different model of using and managing resources. These new trends are leading HPC admins and architects to embrace new approaches: 1) run both containerized and non-containerized workloads on existing HPC clusters; 2) run HPC environments on cloud managed resources.

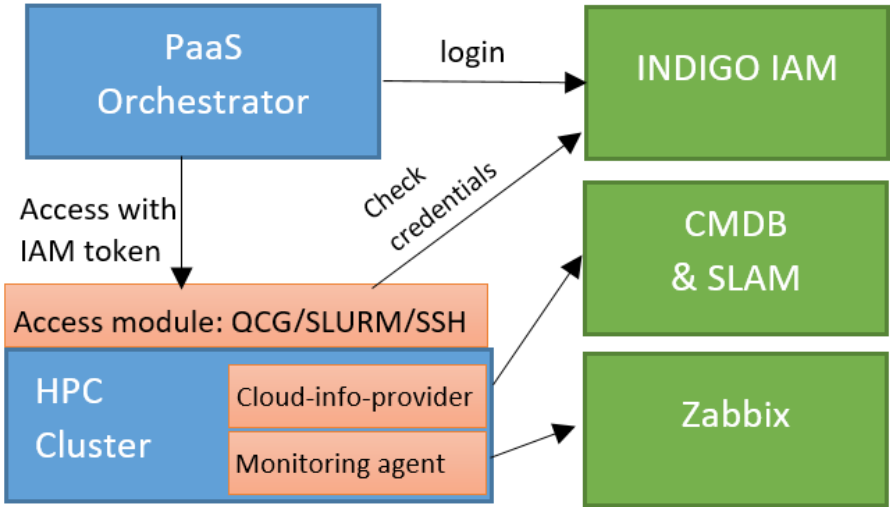
These two approaches are not exclusive and both should be supported if possible, although it is not recommended that the same physical resources will be managed by two different resource management systems. Tightly coupled applications can benefit from better performance running on traditional HPC cluster (with high-performance network); on the other hand, embarrassingly parallel applications can be deployed on non-overbooked cloud ecosystem or on container orchestration platforms without performance penalty. With some additional effort, it is possible to prepare an application that can exploit built-in elasticity features of the cloud environment.

The first approach concerns the submission of job requests to sites not managed by cloud system (e.g. Mesos or Kubernetes). PaaS Orchestrator should be extended with support for submission of containerized jobs and applications to LRMS at HPC site, possibly with using udocker containers to run user applications. Technical problems of accessing and managing HPC must be solved, especially authentication, information system, job and resources monitoring etc. More detailed analysis of HPC specific and problems for PaaS integration was included in deliverable D4.1. This approach is addressed by tools described in 2.2.6.1.

The second approach is more pioneering and it is likely to become more common as the availability of new application frameworks (Artificial intelligence, Big Data, Deep-Learning, etc.) designed and implemented to run natively on these platforms (Kubernetes, Mesos) increases. The most important limitation compared to traditional HPC is that the administrator of the resources has reduced control over the utilization of resources by single users. This kind of resource control was not needed in classical Cloud paradigm as either the resource pool was considered to be infinite or the decisions about the allocation was business driven therefore the requests are limited by the available budget. On the other hand traditionally HPC resources were offered to scientific communities cost-free therefore a mechanism that ensures fair access to the resources was created. If the cloud will be the next generation compute environment for science, a fair, shared access model must be introduced to the Cloud. This approach is addressed in 2.2.6.2

Indeed, the co-existence of the two environments, the traditional on premise HPC cluster and the Kubernetes/Mesos cluster, allows to run mixed workloads and to select the most suited environment depending on the characteristics of the workload itself.

### 2.2.7. Tools for accessing HPC resources

|                                |  |
|--------------------------------|--|
| <b>Identification</b>          | HPC Integration Tools  |
| <b>Type</b>                    | A set of services  |
| <b>Purpose</b>                 | Provides the way to access HPC resources from the PaaS Orchestrator  |
| <b>Function</b>                | <p>One of the approaches for HPC and Cloud integration is to allow PaaS services to submit job requests to existing traditional Local Resource Management Systems (LRMS), such as SLURM. Implementation of this approach must take into account many aspects, especially authentication, interaction with HPC, HPC information system and HPC resources monitoring. A set of tools needs to be prepared to provide the required functionality.</p>   |
| <b>High level architecture</b> | <div data-bbox="491 853 1385 1357" data-label="Diagram">  </div> <p>Integration of HPC resources involves a set of components that provide various functionalities.</p> <p><b>Authentication module</b></p> <p>This module is responsible for the authentication of users on HPC resources. There are two actions that must be supported. The first action is the creation of an account for a new user. Accounts should be created automatically with administrator supervision using information from IAM. Account import module must be tailored to the specific sites needs and configuration. The second action is an actual authentication conducted each time user accesses resources. Authentication process should accept Indigo IAM token and validate it with a project IAM service.</p> <p><b>Interaction with computing services</b></p> |

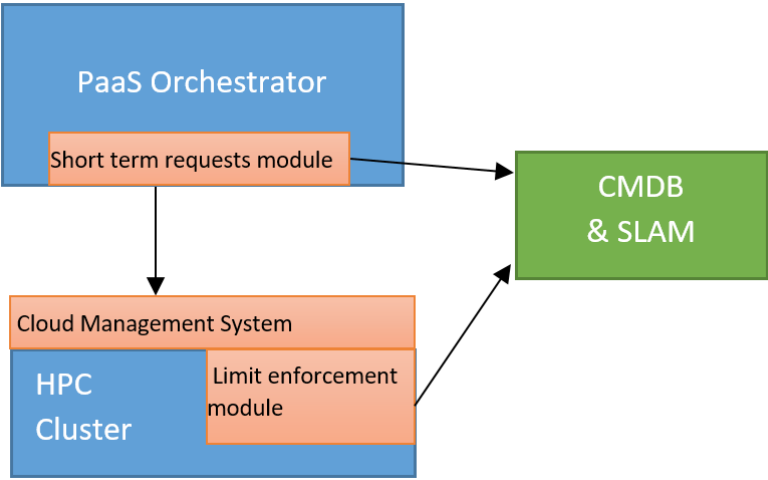


|                            |   |
|----------------------------|---|
|                            | <p>PaaS Orchestrator must be able to submit computational requests to HPC. It is strongly recommended to use a submission service that hides the complexity of HPC resources accessing and works with different LRMS. Such service should offer on-demand access to computing resources and jobs over the REST interface. It allows for remote submission of jobs to scheduling systems as well as remote control over these jobs. In particular, it currently offers methods to:</p> <ul style="list-style-type: none"> <li>• submit a job,</li> <li>• monitor a job’s state,</li> <li>• cancel a submitted job,</li> <li>• query about submitted jobs,</li> <li>• query about available resources.</li> </ul> <p>When possible an existing solution should be used as a submission service, e.g. open source QCG-computing service [8]</p> <p><b>HPC information system provider</b></p> <p>Information about HPC configuration must be passed to DEEP CMDB and SLAM services. This information comes from the local resource management system and from the local accounting system. Configuration can be obtained by querying LRMS system. A specific implementation of cloud-info-system should be prepared for the most popular LRMS (e.g. SLURM). Information about user SLA may depend on the specific solution used by different HPC. Therefore the SLA information provider should be tailored to specific HPC site requirements.</p> <p><b>HPC monitoring probe</b></p> <p>The probe will extend the set of probes prepared by WP5. It will work in push mode: agents execute periodically on HPC resources and send data to the central server at will. Monitoring data for HPC resources can come from resources itself or from local HPC monitoring systems, which collects data locally, aggregates them and pushes to the central DEEP Zabbix server.</p> |
| <p><b>Dependencies</b></p> | <p>CMDB: for collecting HPC configuration information like queue names, number of cores, memory sizes etc.</p> <p>SLAM: if HPC have a service level agreements with users this information should be submitted to SLAM</p> <p>Monitoring System: HPC monitoring probe should submit monitoring data to it</p> <p>IAM: required for the authentication of users</p>  |

|                               |   |
|-------------------------------|---|
| <b>Interfaces</b>             | Job submission interface – it is defined by computing service   |
| <b>Data</b>                   | Computing service: JSON<br>HPC-info-system: Glue2<br>HPC monitoring probe: JSON/XML   |
| <b>Needed improvement</b>     | The components are under development and not yet ready for integration.   |
| <b>Current TRL status</b>     | Authentication module: TRL3<br>Interaction with computing services: TRL2<br>HPC information system provider: TRL4<br>HPC monitoring probe: TRL2 |
| <b>Expected TRL evolution</b> | TRL6: is planned for internal milestone in month 24<br>TRL8: is expected level in the second implementation of software platform.               |

## 2.2.8. Virtual HPC clusters

|                                |  |
|--------------------------------|--|
| <b>Identification</b>          | HPC related Cloud extensions   |
| <b>Type</b>                    | Extensions to other software components  |
| <b>Purpose</b>                 | These components implement the second (first is in 2.2.6) agreed approach for integration PaaS Orchestrator with HPC resources. HPC is managed by a Cloud management system (e.g. OpenStack, Mesos or Kubernetes) and orchestrator submits computation request to it.  |
| <b>Function</b>                | The components extend existing software components (e.g. Orchestrator, Mesos) to provide a better control over the resources and its usage by users. Software components must obey site policies of assigning resources to users, e.g. a maximum number of cores assigned to a single request, a user or a group of users; maximum time length of a request or assigning priorities to users requests.   |
| <b>High level architecture</b> | HPC resources are controlled by a Cloud management system and the PaaS Orchestrator accesses them. Almost all required functionality is provided by INDIGO and DEEP stack components. Extensions are foreseen to provide better control of HPC resources which is a crucial functionality for resources owners. Cloud management system assumes unlimited resources and no restriction on the usage of it. This is contrary to traditional HPC model with LRMS which limits jobs time and resources available for a single user. |

|                               |   |
|-------------------------------|---|
|                               |  <p>Recommended solution would be to modify or extend Cloud system schedulers to support resource allocation policy that takes into account job lifetime. Not all cloud system support this (e.g. OpenStack), therefore in such cases an alternative solution will be implemented. Introduction of a short term Cloud requests, which means that the computational requests from PaaS must be limited in time and killed when the time exceeds.</p> |
| <b>Dependencies</b>           | <p>SLAM – to store requests limit agreed between users and HPC owners</p> <p>Orchestrator – additional module to control the limits of submitted requests and cancel them in case of time is up.</p> <p>Cloud management system – additional module to control jobs execution (if supported by the Cloud)</p>   |
| <b>Interfaces</b>             | <p>Communication with Cloud Management Systems can be done via:</p> <ul style="list-style-type: none"> <li>• configuration files</li> <li>• REST API</li> <li>• Web UI</li> </ul>   |
| <b>Data</b>                   | <p>Module configuration stored in files</p>   |
| <b>Needed improvement</b>     | <p>The components are under development and not yet ready for integration.</p>  |
| <b>Current TRL status</b>     | <p>TLR2 – study of how technologies could be applied in the final solution</p>  |
| <b>Expected TRL evolution</b> | <p>TRL6 - The integration and testing of basic components in a simulated environment in the second implementation of software platform</p>  |

### 3. First implementation of software platform for accessing accelerators and HPC

Since components included in the first implementation of the software platform for accessing accelerators and HPC have different levels of maturity, the work done in the first implementation for each component is different. The udocker has production status regarding to executing containers and no configuration is needed, so we can work on improving GPU support by adding automatic passing GPU drivers to containers. The nova-lxd plugin in OpenStack still need improvements before reaching production level, it only works with certain software combination and documentation for its correct configuration is severely missing. Therefore, the work in nova-lxd has been focused on finding proper ways for deployment/configuration and making documentations about that. GPU support in Kubernetes and Mesos is experimental so most of the efforts have been done in deployment, configuration, integration with DEEP tesbed and some patches for improving GPU support. HPC integration with PaaS orchestration services is still in the phase of design.

#### 3.1. udocker

udocker [1] is a tool which enables the user to execute Linux containers in user mode. Its main aim is to execute applications in environments where users do not have administration/root privileges.

The use of NVIDIA GPUs for scientific computing requires the deployment of proprietary drivers and libraries. In order to use the GPUs inside containers, the devices associated with the GPU have to be visible inside the container. Furthermore, the driver has to be installed in the image and the version has to match the one deployed on the host. This turns the Docker images un-shareable and the image must be built locally for each host. The alternative is to have an image for each version of the driver, which is un-manageable since at each update, many images would have to be built.

The udocker released at the end of the Indigo-Datacloud project does not have such features, as such in order to use GPUs, the image has to have the NVIDIA drivers and libraries matching the host system. On the other hand, it is not necessary to pass the NVIDIA devices to the udocker container since they are visible inside the container, in the same way a non-privileged user can use those devices in the host system.

The work performed during the first months of the DEEP-HybridDataCloud project was to implement such automatism. The development is available in the "devel" branch of the official GitHub repository [9], and scheduled for the first release of the DEEP-HybridDataCloud software stack at the end of October 2018. The libraries and drivers deployed in the host system are made available to the containers.

This version has been tested under several conditions and by several users and use cases. The tests performed in the framework of DEEP-HybridDataCloud project WP3 and WP4 are described in [2].

## 3.2. OpenStack

During deploying and integrating of the nova-lxd plugin into an OpenStack environment, we faced significant configuration problems which prevented us from testing of GPUs in the OpenStack environment. The problems were such heavy that we had been pushed into configuration tests of the nova-lxd plugin (documented in [3]). It is caused that the plugin has not been prepared for the new version of OpenStack, and the new version of LXD/LXC (with the support of a ZFS storage backend). Those facts (and an announcement of a new Ubuntu LST) caused many problems that had not been taken into account during scheduling.

According to the performed test [14], we chose OpenStack Ansible repository for deployment of an OpenStack Cloud. The main reason is that it deploys a standard OpenStack Cloud with LXD/LXC 3.0.1 which supports GPUs.

Our experiences are documented in a series of How-to articles which are published in DEEP-HybridDataCloud documentation repository in GitHub at [7].

## 3.3. OpenNebula

An experimental site with OpenNebula and NVIDIA Tesla K20 card via PCI passthrough has been deployed according to documentation [15]. The support for vGPU in OpenNebula has been requested but so far no development is done for this feature [16].

From OpenNebula version 4.14, GPUs are discovered by the monitoring probe that gets the list of PCI cards and can be added to the VM template. GPUs are then assigned to a VM by the PCI Passthrough driver by first configuring the kernel to support I/O MMU, then loading the vfio-pci driver and blacklisting the drivers for the selected card. The same blacklisting must be done in the system configuration and the vfio-pci driver should be loaded passing the id of the PCI cards one wants to attach to VMs. This means each GPU can be assigned to just one VM and no sharing is currently possible.

However, it has been brought to our attention that, at least for more recent consumer-grade NVIDIA cards, the relevant drivers for KVM virtualization may not be supported any more without additional patching, e.g. [17]. This will be investigated in the next future.

## 3.4. Mesos

In the first period of the project, we have focussed on the following aspects:

1. adding Openid-Connect/Oauth2.0 authentication/authorization to the API endpoints of Apache Mesos and its frameworks, Marathon and Chronos;
2. testing the NVIDIA GPU support in Mesos and its frameworks (Marathon/Chronos).

Both aspects are essential for the integration with the PaaS layer (WP5).

Concerning 1) Mesos/Marathon/Chronos does not support Openid-Connect authentication natively. A very simple solution is to front the Mesos cluster with an Apache server that itself is capable of negotiating authentication for users. We have documented and tested the configuration [5] that can be used to setup a reverse proxy that uses the apache module *mod\_auth\_openidc*.

Concerning 2) we have setup some testing environments in order to verify the support for GPUs in Mesos/Marathon/Chronos. We have documented the procedures [4] needed to add the GPU support in a cluster and developed ansible recipes to automate these installation and configuration steps.

In case of Chronos, we started from the mainstream code and applied the patch [12] available on GitHub to include the GPU support; we have compiled the code and created a Docker image to be used for the deployment of the Chronos framework.

Several tests have been performed to verify the correct management of the available GPUs for Marathon applications (long running services) and Chronos jobs (batch-like processing jobs). In particular, tensorflow Docker images with different versions of CUDA and cuDNN have been used in the tests in order to verify that Mesos is able to mimic the behaviour of *nvidia-docker* mounting automatically the proper NVIDIA drivers and tools directly into the Docker containers.

In Mesos, GPUs are treated as the other resources available in the cluster, i.e. cores, memory, and disk. However, there is an important difference in the containerization mechanism used to run the tasks: in case of tasks requiring one or more GPUs only the MESOS containerizer [13] is supported, whereas, in case of tasks requiring only CPUs, both MESOS and DOCKER containerizers can be used. We have verified that in case of tasks running with the MESOS containerizer we are able to specify all the required features (ports, volumes, etc.) as for tasks running with the DOCKER containerizer. The definition of the task is, of course, different and this has impacts on how the PaaS interacts with the Marathon and Chronos APIs. Examples of JSON definitions for submitting a Marathon application running tensorflow without and with GPUs are provided in [4].

As shown in those examples, there are some differences between the two cases: 1) the container type (DOCKER vs MESOS); 2) the network mode: since the MESOS Containerizer does not support the bridge networking, the host network mode is requested. Anyway, in both cases the service port can be requested and exposed on the cluster load-balancer (haproxy) so that the user will access the Marathon application in the same way both in case of tasks using GPUs and in case

of tasks using only CPUs. The PaaS Orchestrator will be modified in order to submit the proper JSON definition as described above.

Currently, Mesos clusters with GPU support are available both in the development and in the preview testbeds.

### 3.5. Kubernetes

To explore the features of Kubernetes and in particular the built-in batch system, the support to GPGPUs and the integration in the PaaS layer, a pilot cluster have been installed and configured. As an initial work, started from a similar activity in installing and configuring a Kubernetes cluster on top of the Ubuntu Operating System, the procedure for installation and configuration of a Kubernetes cluster on CentOS7 has been provided. The cluster is composed by a Master node and one Worker node. The worker node supports up to 2 GPUs Tesla K40m, CUDA 9.1 and NVIDIA Driver 390.30. For such purpose, a set of additional components have been installed in the workernode hosting the GPUs. Among those components, there is the cuda repo for CentOS.

As part of the work, a specific set of examples have been implemented to test both the correct setup of the cluster and the versatility of the GPUs integration.

The examples are ranging from a very basic sample, that implements element by element vector addition using different Docker images with different versions of CUDA drivers installed, to a more complex example where a bunch of different CUDA jobs with different running time have been submitted with the scope to highlighting the features of the scheduler (FIFO) embedded in the Kubernetes cluster.

To improve the activities carried out so far, the procedure for the integration between a Kubernetes cluster and DEEP-IAM Service (<https://iam.deep-hybrid-datacloud.eu>) has been carried out with the final goals to i) authenticate users with OpenID Connect (OIDC) protocol and ii) authorize them by using token claims (username or groups).

As a first step, some Kubernetes services and the client have been configured to define the IAM service used for OIDC authentication. In the second step, from the IAM dashboard have been created groups with different privileges and some users have been assigned to those groups. The creation in IAM of a new client completes the initial configuration.

Also, in this case, a defined set of examples showing the different authentication/authorization (AuthN/AuthZ) mechanisms both per user and per group have been defined as attribute-based access control (ABAC) policies and implemented to enable or limit the usual set of actions that a user may perform in a Kubernetes cluster.

Both activities have been extensively documented as How-to articles and made public available [7] for the project purposes.

### 3.6. HPC

The integration of HPC resources in the Cloud model assumed by the DEEP project required careful definition of available functionalities, requirements, and existing solutions. Recently the Cloud paradigm evolves and covers new application areas, including those requiring intense computation. Nowadays almost all the Cloud providers are able to provide virtual instances with accelerators and low latency interconnection network, although intense computations are still a domain of HPC centers.

Typical HPC systems consist of at least one head node and a set of working nodes connected to each other via a very fast network (e.g. Infiniband, Ethernet, or proprietary). HPC workloads typically consist of a number of short, computationally intensive and resource demanding jobs spanning tens or in some cases hundreds of physical servers. The resource manager is aware and takes into account physical properties of the supercomputer (such as network topology, memory hierarchy etc) and, in many cases, the application comes with a description of how in the physical layout of the application should look like. Application, libraries, and tools offered for users are prepared by HPC administrator and optimised to the actual hardware resources.

Typical Cloud model assumes an infinite number of resources assigned to services on demand and neglects the physical layout and properties of the hardware. Cloud services usually are long term and less resource demanding. Application, libraries, and tools are provided by users in virtual machine images and job templates. Thus the software components and version exactly meet users' needs but the efficiency may be not optimal. The processing speed can vary unless virtual machines are bounded to the hardware.

Several benefits can come from the integration of both models, which facilitates both workload management and interaction with remote resources. In fact, some resource providers apply a mixed Cloud-HPC where nodes are virtual images deployed on the hardware via some Cloud management system, but the interface for end users is still a queuing system.

DEEP partners represent both user and resource providers, which enabled the fruitful discussion about the possible approaches and technical possibilities of implementation of DEEP services for Cloud and HPC integration. Exchange of ideas via e-mail, cyclic teleconferences and face-to-face meetings allowed to work out the initial design (presented in deliverable D4.1) and implementation plan for services and tools. Two possible approaches were examined and both will be supported by DEEP projects. The problems that need to be solved include:

- The PaaS Orchestrator must have permission to access site Cloud system. Users must be imported before or provisioned during the first request.
- PaaS must know what physical resources are available, e.g. RAM, GPUS, interconnections, CPUs. This can be handled by appropriate cloud-info-provider component and sent to CMDB and SLA Manager.



- PaaS must be aware, that resources are limited and can be temporarily inaccessible. Most probably they will be occupied by another users' jobs. This is handled by the orchestrator, the Service Level Agreement Manager (SLAM) and the Cloud Provider Ranker (CPR).
- HPC systems provide resources to many users and must guarantee some level of fairness. E.g. a single user cannot block resources for weeks and system must limit the time of assigning resources to a single resource request.
- The VM and Docker images selected by the PaaS Orchestrator must be tailored to the hardware, so that applications, libraries and tools can be executed efficiently.

## 4. Next steps

### 4.1. udocker

The results and tests of new features in udocker regarding the first months of the DEEP-HybridDataCloud project were described in section 3.1 and [2]. In addition to the implementation of GPU automatic support, several bug fixes and enhancements have been implemented, in particular, the ability to run udocker inside a Docker container where the Docker image is "Alpine", also to run udocker in "ARM" architectures.

The plans for the next months are the following:

- Improve support for MPI applications. By matching MPI libraries both inside and outside of the container, udocker can support MPI applications using OpenMPI over shared memory or TCP/IP. This support will be further tested and improved aiming at accessing low latency interconnects directly.
- Support for Python3: The current version of udocker is supported in Python 2.6 and 2.7, due to the End of Life of Python 2.7 in 2020, a move to Python3 is necessary.
- Modularization of the udocker source code: udocker is currently a single Python script, this eases the download and use by users with minimal dependencies, and its use as a standalone "executable". Modularization of udocker will allow its sustainability, a better support, and compliance with WP3 SQA requirements, code clarity, re-usability as well as the use of the Pypi to distribute the package. On the other hand, udocker should still be supported in older platforms (or Operating Systems such as Centos6) still existing in many HPC clusters, and for that there is still a need to produce a "single" executable binary. This can be accomplished through tools such as "pyinstaller", that can build a single binary with all dependencies included. Such an option is currently under evaluation and initial testing.

## 4.2. OpenStack

According to the configuration tests (see section 3.2), we choose Ansible for deployment of the OpenStack Cloud environment.

The plans for the next months are the following:

- An adoption of OpenStack into the project environment is necessary (an integration with INDIGO IAM authentication/authorization system). Verification of OpenID is the key task within the integration effort
- The next step will be an evaluation of GPU support through LXD/LXC. It is officially supported but we need to perform tests of its usability.

## 4.3 OpenNebula

The next steps will be more practical tests of the use of NVIDIA GPUs with OpenNebula, to check the features and usability. Also, the roadmap for virtualization support by the vendor will be investigated to ensure the long-term sustainability of the solution.

## 4.3. Mesos

The work done for enabling OpenID-Connect authentication and the support for GPUs (as described in Section 3.4) was essential for the implementation of the new features in the PaaS layer (WP5). Indeed, the integration with the PaaS is progressing smoothly thanks to the analysis and tests described in this document.

For the next months, the following activities are planned:

- evaluation of the support of low-latency interconnections (namely Infiniband) and usage of MPI in applications running as Mesos tasks;
- investigation about GPU sharing across Mesos tasks;
- better resource selection according to user requirements;
- upgrade of the components (Mesos/Marathon/Chronos) to the latest stable versions.

## 4.5 Kubernetes

The activities performed in the first part of the project led to the successful integration of Kubernetes with the DEEP INDIGO IAM instance as demonstrated in the clusters available in the project integration and preview testbeds. Moreover, in the same clusters the support of GPUs has been enabled and tested successfully so that users from WP2 managed to run some of their applications requiring GPUs.

For the next months the following activities are expected:

- evaluation of the support of low-latency interconnections (namely Infiniband) and usage of MPI in applications running as Docker containers on top of the kube nodes;
- investigation about GPU sharing across dockerized applications;
- better resource selection according to user requirements;
- upgrade of the components (if needed) to the latest stable versions.

## 4.6 HPC

The implementation of HPC integration components is not completed. During the next months, the focus will be on supporting the integration of PaaS with traditional HPC resources.

- An interface will be prepared to access HPC resources. SSH plugin using Indigo IAM authentication token can be used by PaaS Orchestrator (or user applications) for accessing data stored on HPC sites.
- Import module for importing users from Indigo IAM to local HPC site user bases. The first proof of concept will be prepared for one site.
- PaaS Orchestrator extension will be prepared to submit computation requests to dedicated middleware (QCG computing)
- HPC-information-module will be prepared. The first supported system will be SLURM
- integration with project Zabbix will be prepared. In the first phase some exemplary monitoring probes will be prepared.

It is expected that the basic functionality will be achieved for an internal milestone in month 24 and that final version will be ready for the second implementation of software platform

Implementation of the second approach - HPC extensions to Cloud management system is planned in the later phase of the project

PSNC will dedicate some resources to prepare a small cluster managed by a Cloud middleware to test the cluster on demand approach. The tests can take into account both, open source (e.g. Kubernetes) or commercial software (e.g. Huawei Fusion), depending on the local conditions in HPC centres. Specific points of interest, which will be dealt with during the implementation of the project:

- support for short term requests in PaaS Orchestrator
- limit enforcement modules in existing Cloud management system where possible

- sharing of resources between HPC and Cloud environment (especially file and object stores, integration of sync&share systems with HPC);
- management of templates of working environments (e.g. containers based templates for various classes of execution environment with support for creation, update and optimization);

## 5. Conclusion

In this document, the first software platform for supporting containers and accelerators has been described. The platform consists of components with different types: traditional Cloud management frameworks OpenStack and OpenNebula, pure-containers Kubernetes and Mesos, and HPC platform. In all components, the supports for containers and accelerators are emphasized.

Since it is the first software platform, its components are still in developments and reach different levels of maturity. The udocker is production status regarding to executing containers but GPU support is still experimental (TRL 6). The nova-lxd plugin in OpenStack needs improvements, especially regarding deployment and documentation, before reaching production status and GPU support is planned in the next period. GPU support in Kubernetes and Mesos is experimental and still needs some improvements. Finally, HPC integration with PaaS orchestration is still ongoing.

In the next period, the work in WP4 will be focused on development and improvements of all components according to the plan specified in Section 4. The overall objectives are to reach production levels of all components, both generally and specifically regarding to support for accelerators, better integration, and support with higher software layers (DEEPaaS, PaaS Orchestration services) and delivering complete software platform for accessing accelerators for user communities.

## 6. Glossary

API: Application Programming Interface

CLI: Command-Line Interface

CMDB: Configuration Management Database

CPU: Central Processing Unit

CUDA: Compute Unified Device Architecture

DB: DataBase

DEEPaaS: DEEP as a Service

DNS: Domain Name System

GPU: Graphics Processing Unit

GUI: Graphical User Interface

HPC: High-Performance Computing

IAM: Identity and Access Management

JSON: JavaScript Object Notation

KVM: Kernel-based Virtual Machine

LRMS: Local Resource Management System

LXC: Linux Containers

LXD: Linux Container Daemon

MNIST: Modified National Institute of Standards and Technology

MPI: Message Passing Interface

MQ: Message Queue

PaaS: Platform as a Service

QCG: Quality in Cloud and Grid

REST: Representational State Transfer

SLAM: Service Level Agreement (SLA) Manager

SSH: Secure Shell

SSL: Secure Sockets Layer

TRL: Technology Readiness Levels

UI: User Interface

VM: Virtual Machine

XML: Extensible Markup Language

## 7. References

1. Jorge Gomes, Emanuele Bagnaschi, Isabel Campos, MarioDavid, Luís Alves, João Martins, João Pina, Alvaro López-García, PabloOrviz, "Enabling rootless Linux Containers in multi-user environments: The *udocker* tool", Computer Physics Communications, Volume 232, 2018, Pages 84-97, ISSN 0010-4655, <https://doi.org/10.1016/j.cpc.2018.05.021>.
2. Test and evaluation of new GPU implementation in udocker:  
[https://github.com/indigo-dc/deep-docs/blob/master/docs/udocker/udocker\\_gpu\\_1stimplementation\\_tests.md](https://github.com/indigo-dc/deep-docs/blob/master/docs/udocker/udocker_gpu_1stimplementation_tests.md)
3. Testing of nova-lxd with different software configurations:  
<https://github.com/indigo-dc/deep-docs/blob/master/docs/nova-lxd/nova-lxd-configuration-testing.md>
4. Enabling GPU support in Mesos: <https://github.com/indigo-dc/deep-docs/blob/master/docs/mesos-cluster/enable-gpu-support.md>
5. Enabling open-id connect authentication:  
<https://github.com/indigo-dc/deep-docs/blob/master/docs/mesos-cluster/enable-openid-authentication.md>
6. Add GPU support via --enable-features: <https://github.com/mesos/chronos/pull/810>
7. Documentation repository: <https://github.com/indigo-dc/deep-docs>
8. QCG Computing: <http://www.qoscosgrid.org/trac/qcg-computing>
9. udocker official GitHub repository: <https://github.com/indigo-dc/udocker>
10. Apache Mesos: <http://mesos.apache.org/>
11. Marathon: <https://mesosphere.github.io/marathon/>
12. Chronos: <http://mesos.github.io/chronos/>
13. Mesos Containerizer: <http://mesos.apache.org/documentation/latest/mesos-containerizer/>
14. Deploying nova-lxd with Openstack Ansible: <https://github.com/indigo-dc/deep-docs/blob/master/docs/nova-lxd/nova-lxd-ansible.md>
15. PCI Passthrough – OpenNebula documentaion: [http://docs.opennebula.org/5.6/deployment/open\\_cloud\\_host\\_setup/pci\\_passthrough.html](http://docs.opennebula.org/5.6/deployment/open_cloud_host_setup/pci_passthrough.html)
16. Feature #3028: support shared vgpu, eg: nvidia GRID:  
<https://dev.opennebula.org/issues/3028>
17. nvidia-kvm-patcher: <https://github.com/sk1080/nvidia-kvm-patcher>