MSO4SC

D3.1 Detailed Specifications for the Infrastructure, Cloud Management and MSO Portal

Project Acronym	MSO4SC
Project Title	Mathematical Modelling, Simulation and Optimization for Societal Challenges with Scientific Computing
Project Number	731063
Instrument	Collaborative Project
Start Date	01/10/2016
Duration	25 months (1+24)
Thematic Priority	H2020-EINFRA-2016-1

Dissemination level: Public

Work Package	**WP3 CLOUD TECHNOLOGY**
Due Date:	*M8 (+1)*
Submission Date:	*30/6/2017*
Version:	*1.3*
Status	*Final*
Author(s):	*Carlos Fernández, Victor Sande (CESGA); F. Javier Nieto, Javier Carnero (ATOS); Akos Kovacs, Tamás Budai (SZE)*
Reviewer(s)	*Atgeirr Rasmussen (SINTEF); Johan Hoffman (KTH)*

The MSO4SC Project is funded by the European Commission through the H2020 Programme under Grant Agreement 731063

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	05/05/2017	Preliminary TOC	*Carlos Fernández (CESGA)*
0.2	06/06/2017	Sections 6 and 7	Javier Carnero (ATOS)
0.3	07/06/2017	Containers benchmarking	Akos Kovacs & Tamás Budai (SZE)
0.4	12/06/2017	Software deployment	Victor Sande (CESGA)
0.7	14/06/2017	Orchestrator monitoring	Javier Carnero (ATOS)
0.8	15/06/2017	Sections 2 and 3	Carlos Fernández (CESGA)
1.0	19/06/2017	Typos and figures correction	Carlos Fernández (CESGA)
1.1	26/06/2017	Including comments from reviewers	Javier Carnero (ATOS), Victor Sande, Carlos Fernández (CESGA)
1.2	28/06/2017	More modifications related to received comments	Javier Carnero (ATOS), Victor Sande, Carlos Fernández (CESGA)
1.3	30/06/2017	Minor updates and final version	Javier Carnero (ATOS), Victor Sande, Carlos Fernández (CESGA)

Table of Contents

List of figures

List of tables

Executive Summary

This document contains the detailed description of the components and the detailed design of the MSO4SC e-Infrastructure, taking in account the requirements collected previously and specified in the D2.1, and the design presented in D2.2. Some of these components have already been tested or a pilot implemented in order to verify that the proposed architecture and definition will be technically feasible. In this document we also present some results of these benchmarks and tests.

1. Introduction

1.1 Purpose

Once the first set of requirements have become available and a deep analysis was performed to determine the features and services to be provided through the e-Infrastructure, in D2.2 those features were analyzed, identifying the conceptual layers they belong to, and defining the high level architecture of the e-Infrastructure. Such definition includes some high level components and examples of how they are expected to interact when providing some of the functionalities.

D2.2 provides a detailed design of the high level components of the e-Infrastructure. Such detailed design still is high level and it is the purpose of this document to provide deeper detail as a base for the implementation.

To produce a deeper detail of the components, in many cases a study of the available technologies was performed. In other cases a pilot implementation was performed to verify that the design will be suitable. Also a benchmarking of the technologies was performed to demonstrate that there will be no performance degradation.

In section 2 of this document we present the requirements that were taken in account. Section 3 describes the features and services needed to cover these requirements. Section 4 covers how we propose to integrate and deploy the mathematical software (MADFs) and pilots. Section 5, 6, 7 and 8 cover the rest of the components: Orchestrator, Portal, Software Repository and Data Repository.

Tests and benchmarks were performed using a development infrastructure provided by SZE and a production HPC infrastructure provided by CESGA. These infrastructures are described in section 9. These systems will be the first to be incorporated and available to the users of the project.

1.2 Glossary of Acronyms

All deliverables will include a glossary of Acronyms of terms used within the document.

Acronym	Definition
CA	Consortium Agreement
D	Deliverable
DoA	Description of Action
DRS	Document Review Sheet
EC	European Commission
MADF	Mathematics Application Development Frameworks
PAR	Periodic Activity Report
PC	Project Coordinator
PM	Project Manager

PMB	Project Management Board
PO	Project Officer
STM	Scientific and Technical Manager
SPR	Semester Progress Report
WP	Work Package
WPL	Work Package Leader
WPR	Work Package Report

Table 1. Acronyms

2. E-Infrastructure: requirements, features and services

In this section we provide a view of the expected features for the MSO4SC e-Infrastructure, taking into account the initial design depicted in the proposal, but specially the requirements taken from the users and developers of the mathematical frameworks, as gathered in the document [3]

2.1 E-Infrastructure requirements

At the beginning of the project a questionnaire to collect requirements from the users and stakeholders was distributed. The answers to these questionnaires were processed and some conclusions arrived that were already the basis of the proposal: the traditional HPC/supercomputer environment is not the optimal solution for this end user community. But also that the Cloud is not a viable solution as it is. A combination of the best of each world would be the ideal solution.

Taking into account the inputs collected, there were other important requirements as well, for example the need to provide visualization tools, interact with the running simulations or to provide pre and post processing tools.

According to these requirements, the e-Infrastructure should support a cloud-like deployment, explicitly recognizing the HPC and big-data (data movement and analysis) characteristics of the applications. The platform must be designed taking into account the optimal provisioning configuration of the main mathematical application classes: distributed memory (MPI), shared memory (OpenMP) and embarrassingly parallel. It needs to be easily adaptable to developing technologies and consider the cost as an inherent factor. The design of new services and applications based on the platform must be flexible to adapt to the platform. Thus the platform and its applications must keep pace with the evolving optimum for affordable compute capacity.

These requirements have been collected from the MSO4SC proposal document, the "Cloud and HPC Questionnaire" circulated at the start of the project, from the requirements document [3] and from WP discussions and meeting notes, so here we will not provide the details of these requirements.

2.2 E-Infrastructure features and services

According to the collected requirements, the main components of MSO4SC supporting the corresponding features and services should be:

- **HPC and Cloud Management (MSO Cloud):** The main features to be supported by the MSO Cloud should be the following:

1. It will support heterogeneous, HPC and multi-cloud systems, such as OpenStack and OpenNebula types of clouds (but will also enable others such as Amazon) and Slurm and TORQUE HPC systems [3], avoiding the vendor lock-in problem. Future emerging cloud types

could be easily connected to it (only the suitable cloud plugins should be developed).

2. Its multi-cloud/HPC feature will enable to simultaneously distribute parallel tasks of embarrassingly parallel applications in several clouds, as a way to speed up their execution.
3. It should be based on the usage of containers so that MPI, OpenMP, embarrassingly parallel and Hadoop-like [4] applications can run efficiently, provided the required parallel scalability and single-node performance can be validated.

The MSO Cloud system will have a Platform as a Service (PaaS) level software development platform providing:

1. A workflow-oriented software development environment
2. Simultaneous job submission mechanism for heterogeneous multi-cloud, cluster and supercomputer systems.
3. A transparent storage access mechanism by which several popular storage systems can be accessed.
4. A meta-broker that can schedule parallel computational jobs among various clouds.
5. An orchestrator tool by which even complex virtual infrastructures (service sets) can automatically and dynamically be deployed and managed in the MSO4SC e-Infrastructure, so that their distribution in the nodes benefits from memory sharing and messaging mechanisms, as required. It should enable a distribution mechanism so that certain parts of the mathematical algorithms will go to Cloud resources, while other parts will go to HPC resources, depending on the definition provided by the MADFs. The tool should also be able to receive input while the simulation is running to change the execution parameters "on the go".
6. A REST API to connect the system with the MSO Portal (below) and also with third party applications.

· **Software product catalogue and toolbox (MSO Portal):** A math-related software product catalogue should be setup that will contain the MADFs and the end-user software products, providing visibility and facilitating the search and access to these applications. MSO4SC should provide graphical user interfaces to simplify the use of the portal and the integration of the MADFs, also enabling some configuration inputs.

Specific requirements on the MSO Portal are:

1. An open online database of high quality MSO software
2. An open online database of mathematical models
3. An open online database of benchmarks
4. Integration with existing open source software repositories and services
5. Archival infrastructure for open source software, specifically for the MADFs and Pilots in MSO4SC

6. A high quality web interface for the above services
7. An integrated visualization framework, such as ParaView.
8. An integrated pre-processing framework supporting CAD geometry construction and mesh generation, such as Salome.
9. "One single button to run the whole simulation"-type interface. Additionally the possibility for interactive simulation should be investigated, e.g. to change MADF or Pilot parameters while the simulation is running.

· **Computing Infrastructure:** The project partners should set up an initial infrastructure using HPC and Clouds from ATOS and CESGA. The initial infrastructure may be limited, and further infrastructure should be sought from external organizations such as PRACE and EGI FedCloud.

3. E-Infrastructure architecture and components

The proposed architecture of the e-infrastructure was presented in D2.2 [4] and is based on four main conceptual layers. These layers are represented in figure 1 and described below:

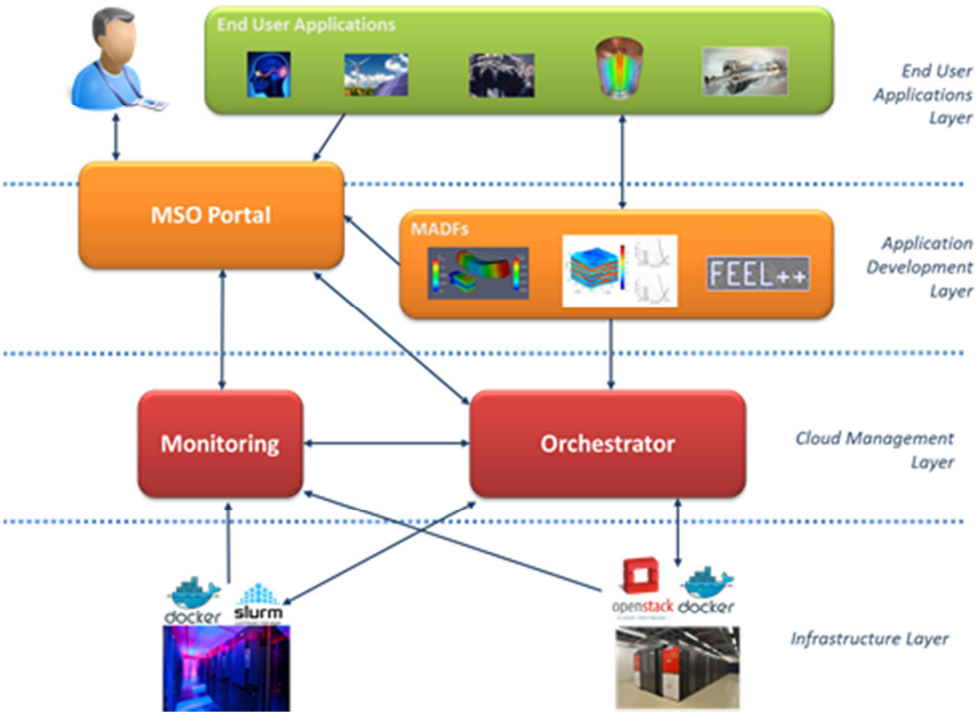


Figure 1. The four layers of the MSO4SC e-Infrastructure

· **End User Applications Layer:** This is the layer in which end users provide their applications, based on the MADFs and other available tools at the Application Development layer. At this layer, basically, it is possible to publish, deploy and monitor complex applications, as well as to design simple experiments for running simulations several ways in an automated way.

- **Application Development Layer:** The purpose of this layer is to facilitate the implementation of applications based on MADFs, by providing not only the MADFs, but also a set of tools which can be also integrated, such as pre/post-processing and visualization. It also provides access to the services of the Cloud Management layer, so it will be possible to know about monitoring, accounting, current deployment, etc.

- **Cloud Management Layer:** This is the layer which maps with those services given usually at the PaaS layer, where services on top of the Infrastructure as a Service (IaaS) are provided, such as monitoring of the applications running, orchestration with load balancing and deployment of the applications.

- **Infrastructure Layer:** This layer corresponds to the typical IaaS layer, where access to computation capabilities is given. These computation capabilities may come from Cloud providers or from HPC centres, enabling a HPC as a Service model.

Taking into account these four layers, the main components have been identified and their relationships are described in figure 2.

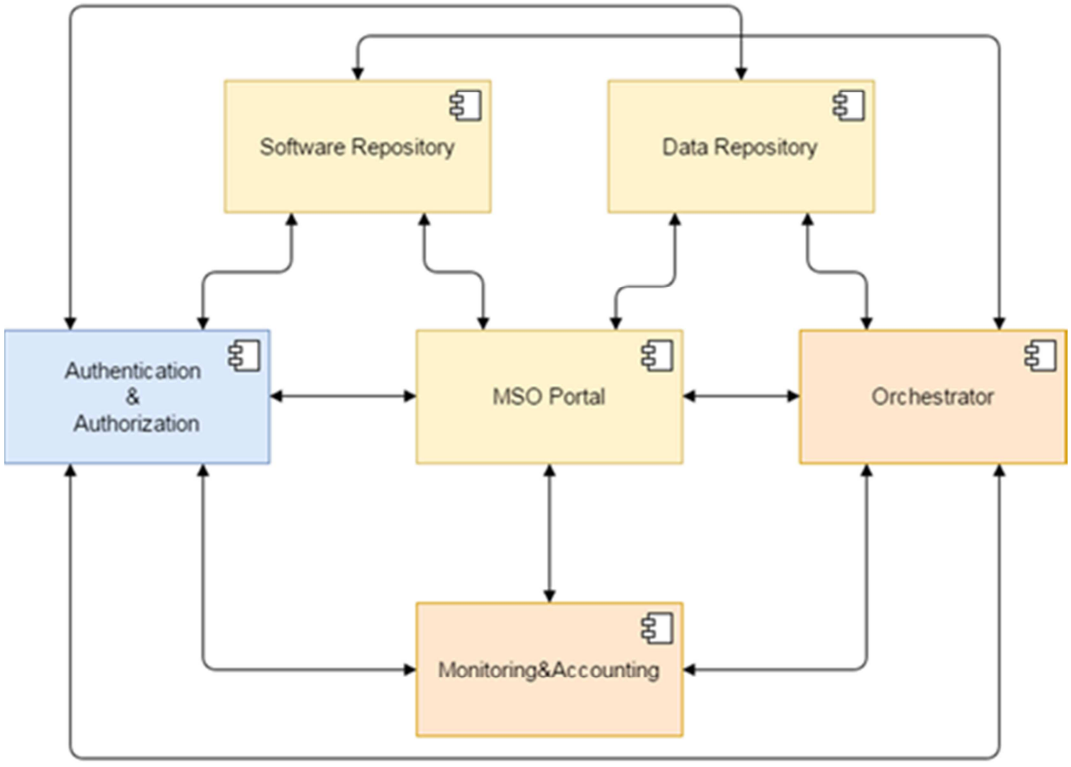


Figure 2. Main components of the MSO4SC e-Infrastructure as described in D2.2 [4]

- **Authentication & Authorization:** This component deals with the security aspects related to users management, single sign-on and authorization. The rest of the components will interact with it in order to confirm users' access to the functionalities, depending on the assigned roles.

- **Data Repository:** It is in charge of the datasets storage and management both for input and output data. Such data will be used by the software to be run in the e-Infrastructure and, therefore, the Orchestrator may request concrete

data movement operations, while the MSO Portal will retrieve information for providing a dataset catalogue.

- **Software Repository:** This repository not only stores the software that can be used in the context of the e-Infrastructure, but also pre-configured containers that can be used by the Orchestrator when deploying applications. It will also facilitate management and testing of the software code whenever possible.

- **MSO Portal:** This component is formed by a front-end and a set of tools available for stakeholders, such as a datasets catalogue, experiments execution, results visualization, data pre/post processing, automated deployment and status monitoring.

- **Monitoring & Accounting:** It retrieves information both about resources usage and about applications execution. It gathers information about the resources spent by users, available resources from infrastructures and current status of the software running.

- **Orchestrator:** This component decides about the most adequate way to deploy the application taking into account resources availability and software characteristics. Moreover, it takes care of requesting data movement and preparing the software so it will be ready to run in the corresponding system.

In the following sections a detailed description of these components is provided. Section 4 will cover the integration of the MADFs and pilots in the infrastructure. Section 5 will describe the Orchestrator. Section 6 the MSO Portal. Section 7 the Software repository and in section 8 we provide the details about the data repository. Finally in section 9 we describe the initial hardware platforms that will be used in the project.

4. Deployment and Integration of MADFs in the e-Infrastructure

In this section we describe how the traditional deployment of software is performed in HPCs and the issues found when trying to use this methodology for the MADFs required in the project. Then we describe how using containerization technology we can provide an alternative that is more flexible with very little impact on performance.

4.1 Installation of the MADFs in the infrastructure

Traditional software deployment and integration consists in performing all the needed steps, like download, configure, build, test and install, to have the software project natively in a production infrastructure. The main goal of this task is to provide the software with good performance and ready to use for end users.

Scientific software and, in particular, mathematical frameworks are extremely complex from an architectural point of view. They are usually composed of

numerous mathematical concepts and features implemented along several software components in order to provide high level abstraction layers. These layers can be implemented in the software project itself or integrated via third party libraries or software components. The whole environment of each mathematical framework is usually composed of a complex dependency matrix and, at least, one programming language and compiler.

Isolation and integration of all dependency matrices at HPC clusters are traditionally managed by environment modules. Environment Modules provide a way to dynamically change the user environment through module files. Lmod, a Lua based module system, is being used at FinisTerrae II.

The key advantage of environment modules is that it allows to use multiple versions of a program or package from the same account by just loading the proper module file. In general, module files are created on per application per version basis. They can be dynamically loaded, unloaded, or switched. Along with the capability of using multiple versions of the same software it also can be used to implement site policies regarding the access and use of applications.

Module files allow managing the loading and unloading of environments to run a particular application, but to manage complex work-flows with environment modules can be sometimes unaffordable, and requires the re-installation of some tools with compatible dependencies. These issues are difficult to manage from the user and the administrator point of view.

Finally, the hardware and software ecosystem of an HPC production infrastructure is different than a development ecosystem, and usually a lot of unexpected issues appear while integrating and deploying the complex environment of mathematical frameworks. To integrate the whole environment of each software project in a production infrastructure is usually hard and time-consuming.

Some issues we found while deploying the MSO4SC mathematical frameworks are detailed below.

4.1.1 FEniCS

FEniCS-HPC is composed by several software modules and interfaces in Python and C++ languages and also depends on several third party libraries that must be integrated to deploy it. FEniCS-HPC is a requirement of Floating wind turbine and 3D Air quality prediction pilots.

After compiling and integrating all the FEniCs-HPC dependencies and to check the proper functioning of the software, several benchmarks were executed and also performance studies were done after each new installation of the software. After each verification and validation stage, when performance issues were detected, support was needed from developers to provide additional information about tuning the compiler flags, and recompilation was performed again until the installation was approved and verified.

This several-week long iterative process results in several installations using several software environments. This means using different compilers, dependency matrices, compilation flags, etc.

4.1.2 Feel++

Feel++ is a huge library and toolbox that uses the latest features of the C++ language standards and depends on multiple third party libraries that must be integrated to deploy it. Feel++ is used by the HiFiMagnet and Eye2Brain pilots.

A complete software environment based in modern C++ compilers has been deployed (more than 50 tools and libraries) at FinisTerrae II in order to satisfy Feel++ requirements.

During the configuration and build steps of Feel++, an issue related with one compiler was detected and also several incompatibilities between the requirements were found resulting in new installations of different versions of some libraries.

Once the base environment was ready, an iterative build process to refine the interaction between Feel++ and its dependencies was done. This process was very time-consuming because the compilation process of Feel++ takes several hours. To complete the whole process took almost 2 months. This process was very similar to the one done with Fenics-HPC, however at the same time very specific and dependent on the software, so no automation of the process was possible

Finally, two different versions of Feel++ must be installed as a requirement of both pilots.

4.1.3 OPM

OPM is composed by several interrelated modules that need a modern C++ compiler to be built. OPM requires a number of libraries and frameworks that are not part of OPM itself to be installed before you can build. The order of compilation of all the involved modules must be strictly satisfied in order to have the pilot OPMFlow natively installed at FinisTerrae II.

After installing all requirements, incompatibilities between some versions of these requirements were found and those dependencies had to be reinstalled. Also some minor fixes in the source code were needed to perform the complete installation of OPM library and the pilot.

The integration and deployment of OPM library and OPMFlow pilot took two weeks.

4.2 Container Technology for the deployment of MADFs and Pilots

As was presented in the previous section, the compilation of this software was very time consuming. Also,as this software is evolving very fast and using the latest technologies and features of the compilers, new versions are provided very frequently. For example in the case of Feel++ two new versions were

released in a period of 6 months. This puts a lot of pressure on the software support team of the infrastructures. Even though there are some tools that could help in this process, like EasyBuild [13], an application to manage software on High Performance Computing (HPC) systems in an efficient way. In this section we propose the usage of containers to solve some of the issues found in the previous section with the integration of the MADFs in FinisTerrae.

4.2.1 Udocker, Singularity, state of the art

Although the containerization techniques is a buzzword nowadays especially in the Datacenter and Cloud industry, the idea is quite old. Container or "chroot" (change root) was a Linux technology to isolate single processes from each other without the need of emulating different hardware for them. Containers are lightweight operating systems within the Host Operating system that runs them. It uses native instructions on the core CPU, without the requirement of any VMM (Virtual Machine Manager). The only limitation is that we have to use the Host operating systems kernel to use the existing hardware components, unlike with virtualization, where we could use different operating systems without any restriction at the cost of the performance overhead. This is the key feature for the project. We could use different software, libraries even different Linux distribution without reinstalling the system. This makes HPC systems more flexible and easy to use for scientists and developers.

Container technology has become very popular as it makes application deployment very easy and efficient. As people move from virtualization to container technology, many enterprises have adopted software container for cloud application deployment. There is a stiff competition to push different technologies in the market. Although Docker is the most popular, to choose the right technology, depending on the purpose, it is important to understand what each of them stands for and does.

Several containerization technologies (like LXC, Docker, Udocker and Singularity) have been tested in the context of the MSO4SC project, but finally, Docker was rejected because of its kernel requirements and security. For example FinisTerrae-II does not have the kernel required by Docker. Udocker and Singularity were developed specifically to be used in HPC environments, as we will describe below. Both of them are Docker-compatible, and help to empower end-users of HPC systems providing a contained location where to manage their installations and custom software. They are also a great solution for developers, one of the biggest benefits for them is to deliver their software in a controlled environment and ready-to-use.

In one hand, Udocker is a basic user tool to execute simple Docker containers in user space without requiring root privileges, which enables basic download and sequential execution of docker containers by non-privileged users in Linux systems. It can be used to access and execute the content of 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.

Although the Udocker development team is working to integrate it with message passing interface libraries (MPI), unfortunately, it is not yet supported.

On the other hand, Singularity was designed focusing on HPC and allows to leverage the resources of whatever host in which the software is running. This includes HPC interconnects, resource managers, file systems, GPUs and/or accelerators, etc.

Singularity was also designed around the notion of extreme mobility of computing and reproducible science. Singularity is also used to perform HPC in the cloud on AWS, Google Cloud, Azure and other cloud providers. This makes it possible to develop a research work-flow on a laboratory or a laboratory server, then bundle it to run on a departmental cluster, on a leadership class supercomputer, or in the cloud.

The simple usage of Singularity allows users to manage the creation of new containers and also to run parallel applications easily. Figure 3 shows the work-flow for creating and running the pilots or MADFs containers using Singularity at FinisTerrae II.

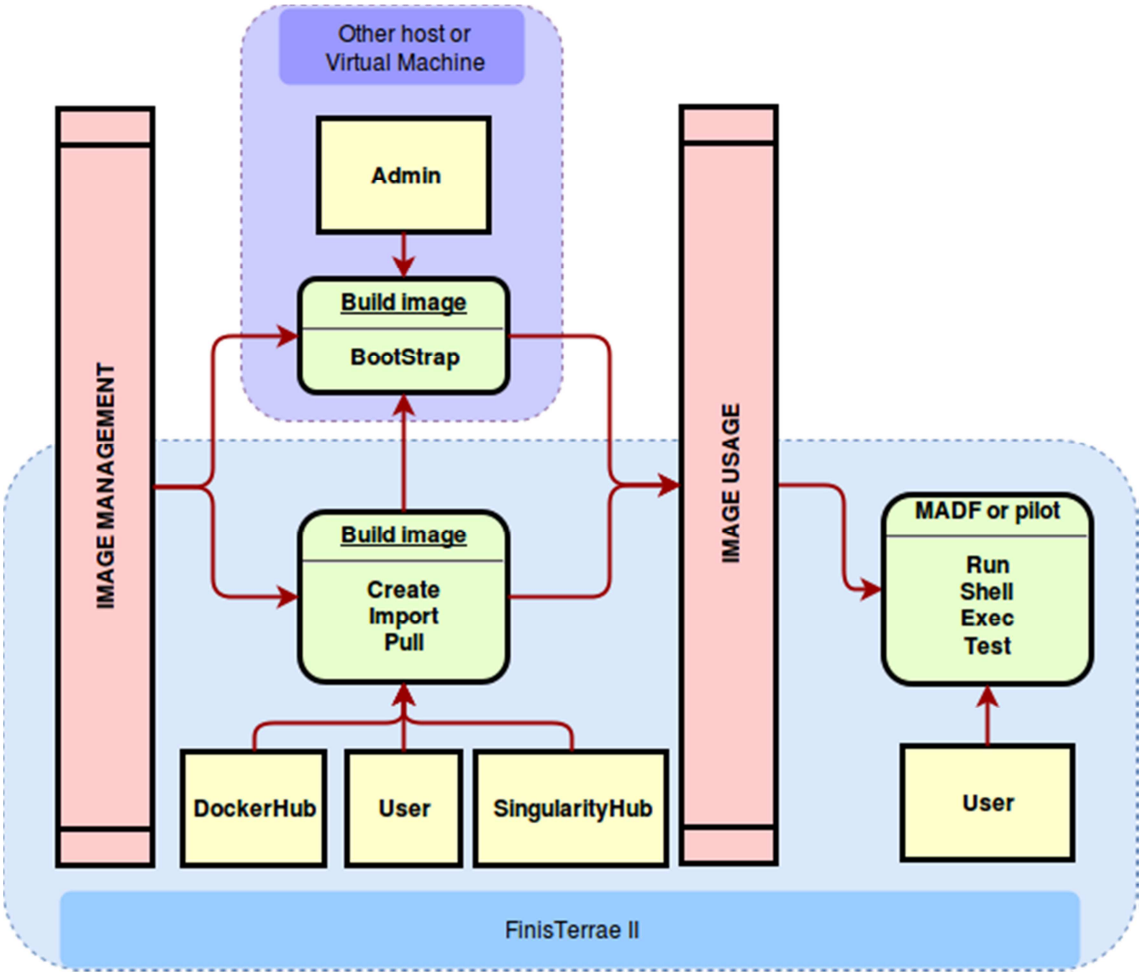


Figure 3. Design of the Singularity images flow in the e-Infrastructure

As we can see in Figure 3, users can create or pull images from public registries (like DockerHub [14] or SingularityHub [15]), and also import images from tar pipes. Once the image is created, singularity allows to execute the container in interactive mode, and test or run any contained application using batch systems. All the work-flow can be managed by a normal user at FinisTerrae II, except the bootstrap process that needs to be called by a superuser. We can use a virtual machine with superuser privileges to modify or adapt and image to the infrastructure using the bootstrap Singularity command.

4.2.2 Containerization technologies comparison and benchmarking

We made some synthetic benchmarks using the industry standard sysbench tool (version 0.4.12) to inspect the CPU overhead of different containerization techniques along with KVM virtualization and native CPU benchmarks. After that we made some test using one of the MADF of the project. We used a HUAWEI CH121 Blade Server with 2x E5-2630 v3 @ 2.40GHz 8 Core CPUs and 128GB of DDR4 memory.

The technologies compared with the native infrastructure include containerization (LXC, Docker and Singularity) and virtualization (KVM). LXC is an operating system-level virtualization method for running multiple isolated Linux systems on a single control host, Docker is a container system making use of LXC containers but with a richer management environment and tools which ease its usage, and Singularity, a containerization system focused in HPC. Finally, KVM (Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware containing virtualization extensions.

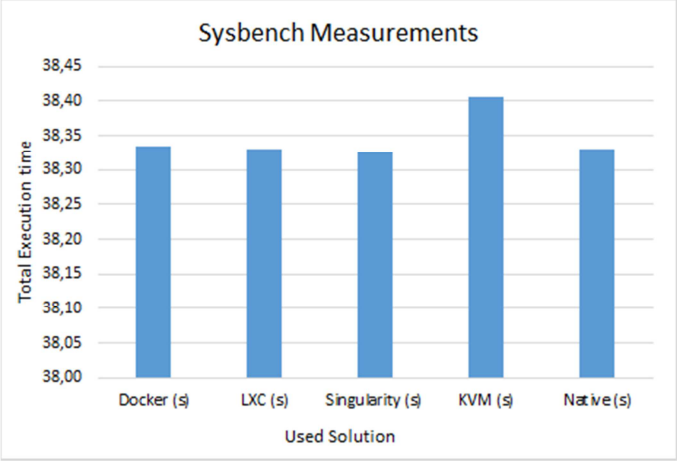


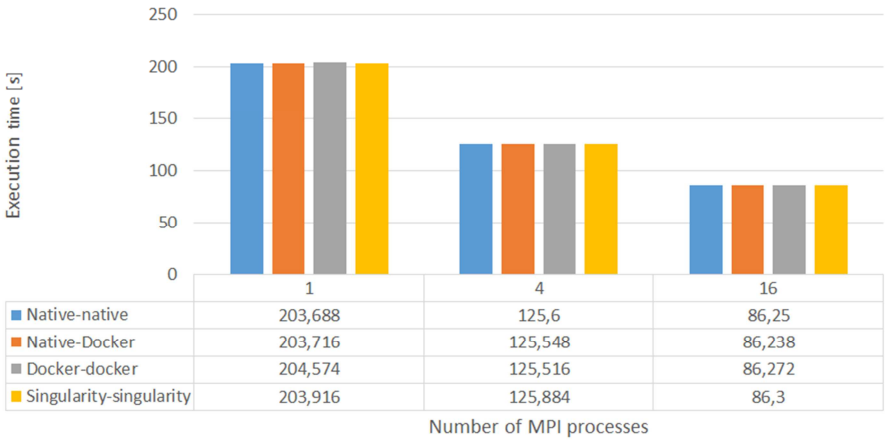
Figure 4. Sysbench benchmark

As we can see in figure 4, all the containerization solutions show almost the same performance as the native execution. The KVM virtualization is a bit slower although this means that KVM virtualization lags behind native execution by less than 1%. The standard deviation results shows that the measurements were pretty the same and stable as well.

After the synthetic benchmarks, we measured the performance with the FEEL++ MADF to validate our synthetic test with a real solver using containers. We used

the Lid driven cavity problem in 2D as a sample simulation using two hosts and MPI between the host and the Containers.

For this benchmark, two different distributed hosts were used and, depending on the technology used, they were named as Native, Docker and Singularity. As shown in the picture below, several combinations of these technologies per host were used.



	1	4	16
Native-native	203,688	125,6	86,25
Native-Docker	203,716	125,548	86,238
Docker-docker	204,574	125,516	86,272
Singularity-singularity	203,916	125,884	86,3

Figure 5. Feel++ Lid driven cavity 2D simulation benchmark

With the Lid driven cavity simulation we proved that the usage of containerization does not provide any overhead to the computing capabilities of the system, as shown in figure 5.

4.2.3 Singularity performance benchmarking at FinisTerrae II

Two different approaches have been used to test the performance while running distributed memory applications using Singularity at FinisTerrae II. We used High-Performance Linpack, a common benchmark tool, and also a benchmark based on FEniCS HPC, a real MADF involved in the project.

High-Performance Linpack (HPL) is a popular benchmark package for solving a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers. The HPL package provides a testing and timing program to quantify the performance of the computation and the accuracy of the obtained solution. The best performance achievable by this software on your system depends on a large variety of factors.

Two different tests cases were designed using HPL, the weak and strong scaling tests. The main goal of these tests was to verify the functioning and to check the correct performance of Singularity while running HPL in distributed-memory computers, but not to reach the peak of performance of the infrastructure.

For these benchmarks we used a Singularity container with an Ubuntu 16.04 (Xenial) OS and HPL 2.2 built over GNU 5.4.0, OpenMPI 1.10.2 and Atlas 3.10.2 as linear algebra package.

Computations were run in 1, 2, 4, 8, 16 and 32 nodes at FinisTerrae II in both cases. The hardware configuration of FinisTerrae II is explained in section 9, "Hardware Infrastructure".

For the strong scaling test we run HPL using a fix-sized square dense matrix with dimension 117824 and for the weak scaling test we run HPL fixing the amount of work per processor using different matrix sizes in order to use almost the 90% of the reserved memory (depending on the number of nodes involved).

Figure 6 shows the runtime of the strong scaling test depending on the number of nodes involved in the computation.

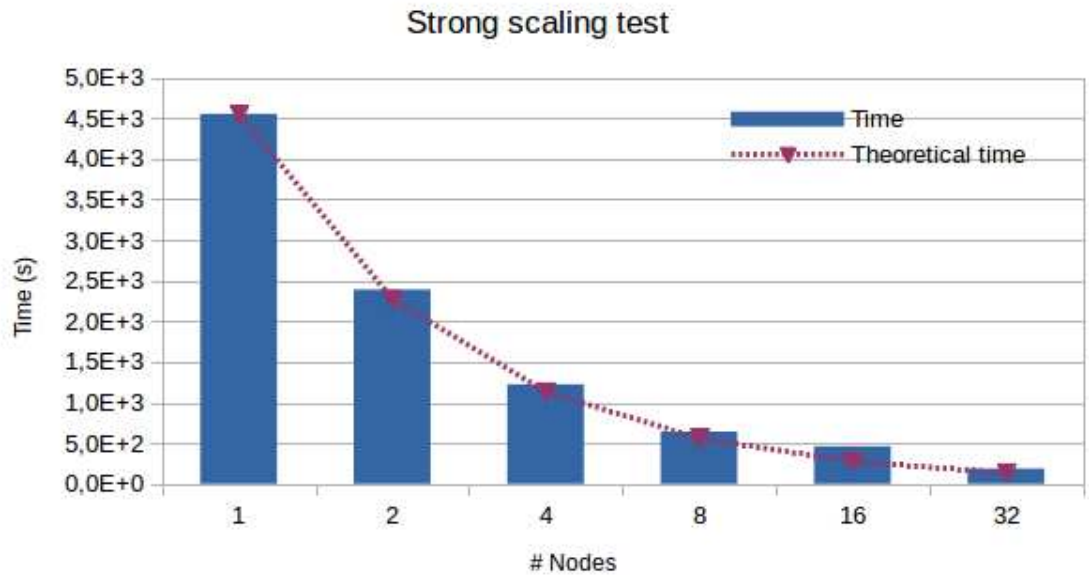


Figure 6. HPL benchmark using singularity

It is important to remark that the casual deviation of almost the 20% occurred running on 16 nodes does not occur while running on 8 or 32 nodes on these benchmarks. The results of the strong scaling tests show in general a good reduction of the execution time, almost directly proportional to the number of nodes used during the execution.

Figure 7 shows the performance (logarithmic scale) of the weak scaling test depending on the number of nodes involved in the computation

The results of the weak scaling tests show an increase of the aggregated performance as we increase the number of nodes involved. Looking at the results we can also see that the performance per node is maintained almost immutable along the different executions. These results are also very close to the expected theoretical values.

.

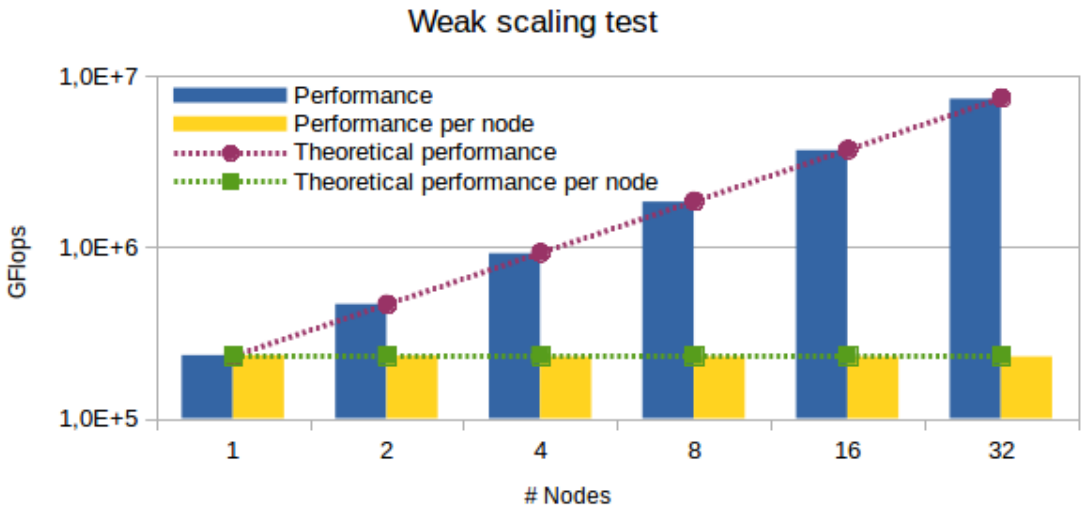


Figure 7. Weak scaling benchmark using singularity

After the HPL benchmarks, in order to assess the accuracy and performance of the singularity approach with real applications using FEniCS HPC, three different benchmarking tests have been prepared and executed in FinisTerrae II cluster:

a) A simple Poisson equation in 3D with known analytical solution. Runs on a single node with 20 cores. The mesh is a unit cube with 329721 vertices.
b) Flow past a cube example on a small mesh of 3351 vertices. Runs on 5 nodes with 120 cores. Due to low number of vertices per core, the running time is dominated by the inter-process communication and the file input output.
c) Flow past a cube example on a bigger mesh of 47586 vertices. Runs on 5 nodes with 120 cores. The setup is designed to have a more realistic floating point arithmetic operation ratio compared to communication and file operations.

Four different setups, using Singularity containers and native installations, have been prepared to run these tests:

1. Using an installation on Finis Terrae II with GNU compilers and running on native operating system (referred later as gnuNative).
2. Using a mirror of the gnuNative installation packed on a container and running it with Singularity (referred later as gnuSingularity).
3. Using an installation on Finis Terrae II with INTEL compilers and running on native operating system (referred later as intelNative).
4. Using a mirror of the intelNative installation packed on a container and running it with Singularity (referred later as intelSingularity).

The four different setups have been first tested for accuracy with test a. After getting satisfactory results the performance has been measured which is described below.

Test case (a)

The Poisson equation with a manufactured solution have been used as a test on a unit cube mesh with 329721 vertices shown in the figure below where half of the domain cut with normal (1,0,0) for better visibility.

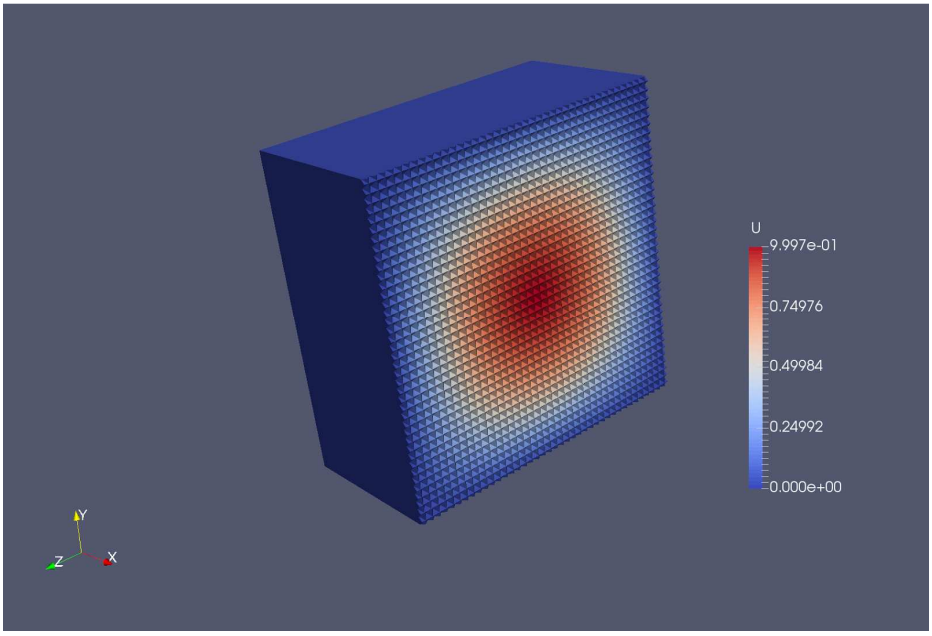


Figure 8. Unit cube mesh

The test for each setup have been repeated 10 times and the average of these 10 running times in seconds for different sections of the code have been listed in Table 2

	gnuNative	gnuSingularity	intelNative	intelSingularity
MPIIO file read	3.96	3.48	3.48	3.46
loop on cells	0.021	0.024	0.017	0.014
Assembly of stiffness matrix	0.422	0.378	0.354	0.354
Assembly of load vector	0.167	0.144	0.105	0.1
Applying boundary conditions	0.009	0.006	0.005	0.008
Solving with GMRES Krylov Solver	1.105	0.917	0.524	0.526
File write	0.932	0.802	0.294	0.298
File write MPIIO	0.144	0.178	0.02	0.027

Table 2. Running times for case a

Test cases (b) and (c)

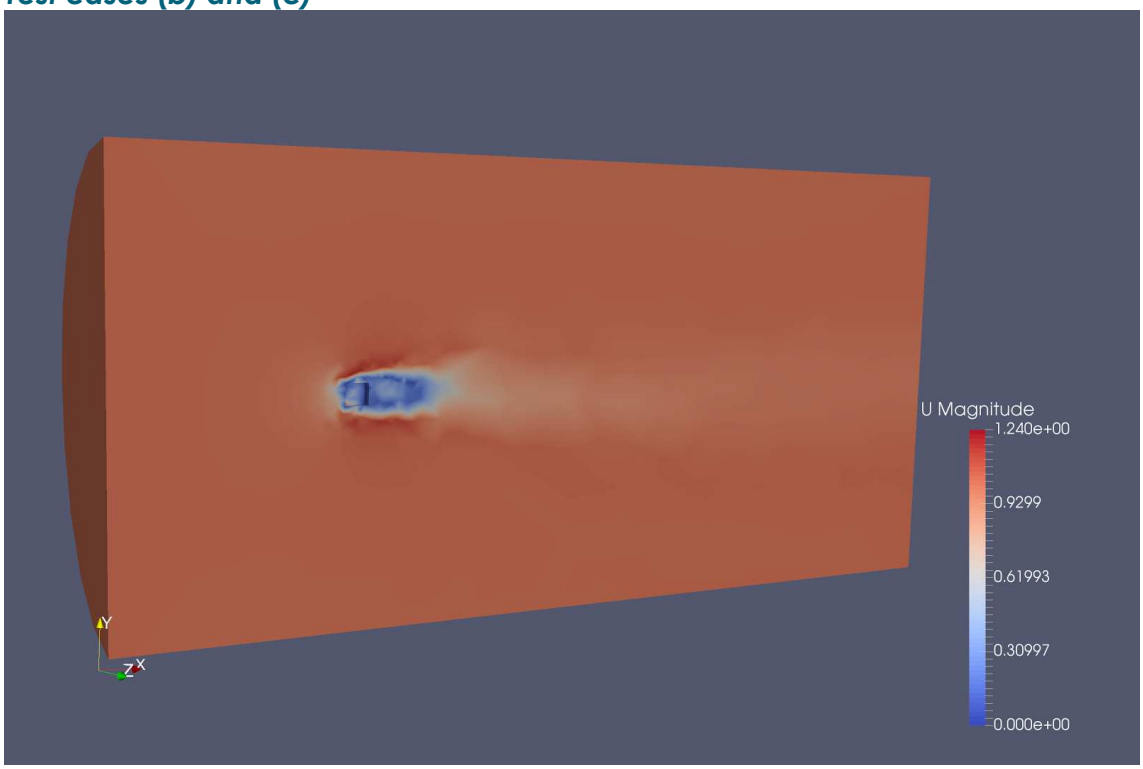


Figure 9. Bluff body cube

The incompressible Navier Stokes equations are solved for the setup of flow past a bluff body (cube). The setup is given in detail at [2]. The difference in these 2 cases is the number of vertices in the case (b) 3351 vertices and the case (c) 47586 vertices where the mesh in the case (c) is constructed by twice uniform refinement of the mesh for the case (b).

Since the simulations are more costly, they are run three times for each setup and average run times in seconds are found in table 3.

	gnu Native	gnu Singularity	intel Native	intel Singularity
case (b)	2045	2098	1942	1943
case (c)	5912	5933	2669	2763

Table 3. Running times for cases b and c.

Although the comparison of the compilers is not the goal of this section, it is important to notice that the binaries produced by the Intel compilers outperform binaries generated by the GNU compilers especially in the case (c). And in general it can be observed that the singularity framework provides close performance to the native environment.

5. The Orchestrator and Monitor

The orchestrator takes decisions about the best way to deploy the applications taking into account the resources availability, software characteristics, and user requirements, based on their experience. This will typically imply operations like data movement and make the software ready to run in the corresponding system. After the deployment, the orchestrator will also run the different components of the applications when needed, managing possible errors and outputs, as well as possible interactions from the end user.

To improve the deployment of the applications and subsequent executions, the orchestrator will be in permanent communication with the monitoring system, to know the status of the different infrastructures and running components (e.g. if there is any issue in the system, or the available storage, among other metrics). Therefore the monitor will be in charge of reading metrics of the HPC infrastructures (queue status, running jobs, etc), as well as extract metrics from applications logs, which will then be sent to the orchestrator and the website portal.

With this information the orchestrator will take the decision about where to send the simulations and, in case of complex simulations, how to use different resources to run the simulation, taking in account dependencies and data movement.

Initial tests on the orchestration were done using Apache Mesos, but design incompatibilities between the behaviour of HPC workload managers and Mesos core implied a broad research that have led to a novel architecture based on TOSCA with Cloudify (with Apache Aria as TOSCA implementation) as an orchestration ground base, plus InfluxDB with other visualization/gathering tools for the monitoring.

From this architecture, described in the next sections, we expect to provide the most versatile and functional HPC+Cloud monitor & orchestrator solution of the scene.

5.1 State of the art

To provide the best architecture and reuse of other open source technologies, the "ground floor" of the design and development of the platform needs to be as rich as possible. As a middle layer, and the layer holding most of the complexity in the platform, extensive research in the field of orchestration and monitoring have been done to take the best informed decisions and reach our goals.

The most relevant results of this research are presented in the next subsections.

5.1.1 Orchestration

It is accepted that the orchestration component of architecture (typically micro-service) provides a more or less automated way of controlling different execution units (services, applications, resources, etc) that at the end performs a higher level complex operation. A simile could be the director of an orchestra, which indicates the entry and rhythm of the musicians to *execute* a song following a score.

However, from a technical point of view, this definition means that the functionalities or tasks that an orchestrator actually performs in a concrete solution vary considerably from one to another. Type of executions (e.g. long running services vs batch applications), metrics to optimize (e.g. orchestrate over network usage vs CPU load) or the concrete problem scope, generate a wide range of orchestrator requirements and architectures.

Therefore, there are several tools that, covering different purposes, act or can act as an orchestrator (or at least fulfil some of its requirements) under certain circumstances. The following subsections present the most relevant ones, divided in functional groups.

Mesos stack

The Mesos stack is defined by a number of applications under the Apache Software Foundation that follow a common high level architecture. Through the Integration of some of these components, the Mesos stack can be shaped to almost any kind of modern cloud solution.

This stack is more-or-less vertical, being the lower component (the one that directly communicates with the hardware) Apache Mesos. As the key component and the one that provides the higher level of abstraction, the stack borrows its name from it.

Apache Mesos essentially transforms a cluster into a pool of resources. To do so, it executes an agent on every machine of the cluster, and, following an offer/consumer pattern, it controls which resources are currently in use and which not, offering the free ones to the upper layers of the stack. That way, upper layers of the system do not need to worry about where to run their tasks, but only the resources needed to run each one (e.g. number of CPUs, memory), reserving those resources when they are offered by Mesos. Also this method simplifies the execution of different applications, or parallel jobs, in the same cluster.

Because of this way of working, Mesos is usually seen as a kernel of a cluster (it abstracts a cluster into a one big machine). While it does not orchestrate anything by itself, it is common that the immediate upper layers implement some kind of orchestration using the pool of resources offered by Mesos.

These applications in the upper layer are identified as Mesos Frameworks. In this section we highlight three of them within the Apache Software Foundation as

examples of orchestration using Mesos. However many more are available from Apache and third parties, covering different functionalities.

Marathon
Through an user-friendly web interface or API, Marathon easily deploys and runs long-run applications in a Mesos cluster. While it has some automatic orchestration features like healing or re-deployment of applications when they fail, most of its orchestration features rely on a human user or an external service to actually execute the orchestration operations, like scaling up and down.

Chronos
It is very similar to Marathon, but built to run batch executions instead. Inspired by the cron system of Unix, it can automate the execution of batch jobs in the Mesos cluster at certain times (e.g. every 5 minutes, once a week)

Aurora
Pretty similar to Marathon in the features it provides, but designed and implemented very differently to manage a great number of scaled instances for every application. It is used to manage applications with great demand, like Twitter.

Container Orchestration

Some solutions focus on orchestration of containers. Instead of dealing with tasks consisting on executable scripts of binaries, they manage container instances. It is a special case of orchestration where its implementation can be simplified and optimized. The most common applications in this field are presented below.

Kubernetes
It orchestrates Docker containers in a cluster. After a petition to execute a specific container, it will deploy it in the cluster and monitor its health. Automatic healing, and also automatic scaling under certain circumstances is possible. It implements a simple way to manage the cluster, but it can also be connected to a Mesos cluster to let Mesos deal with the resources of the individual machines. It is developed by Google.

Docker Swarm
Very similar to Kubernetes but developed by Docker, Docker Swarm can also manage the cluster resources itself or rely on Mesos for it. It integrates better with the Docker stack as it use its same API. Although in the past Swarm had some limitations over Kubernetes, such as software networking or persistent volumes, last releases have added many functionalities that makes Swarm and Kubernetes equivalent choices..

Deployment

Within the functionalities that an orchestrator performs, deployment / configuration operations are common. Because of this, sometimes tools that

automate deployment or configuration processes can be used as orchestrators (if the orchestrator does not have to perform too complex operations) or as components of it where the orchestration software relies on them to perform the deployment (see next subsection).

Ansible, Puppet, Chef, Salt and Fabric are open-source automation engines that automate software provisioning, configuration management, and application deployment. They present different paths to achieve a common goal of managing large-scale server infrastructure efficiently, with minimal input from developers and sysadmins. All five configuration management tools are designed to reduce the complexity of configuring distributed infrastructure resources, enabling speed, and ensuring reliability and compliance.

Another tool is Terraform, that allows users to define a data centre infrastructure in a high-level configuration language, from which it can create an execution plan to build the infrastructure in a service provider such as AWS or OpenStack.

Meta-Schedulers

Schedulers usually manage queues of tasks trying to execute them as optimised as possible using some criteria. These are typically installed as managers of a cluster; for example some Mesos frameworks are considered schedulers of a Mesos cluster. Slurm and TORQUE are schedulers to manage jobs in HPC clusters.

Meta-schedulers are applications that provide an additional abstraction layer to schedulers, typically being able to communicate with different clusters using different schedulers, and also allowing customization to add orchestration capabilities over the clusters they manage. For this reason a meta-scheduler approach is particularly useful in MSO4SC, as we are working with different HPC and Cloud infrastructures, with different managers in each case.

Some interesting ones that have been around for a while are DIRAC, Maui or Moab, all of them built to allow a federation of different HPCs with different schedulers where a user would be able to execute a job without knowing the specifics of each HPC, or even where it was going to be run. They are not very flexible and their customization is hard, suitable mostly for “ad-hoc” solutions in certain organizations.

Following the Cloud trend, many workflow based systems have been developed to serve as deployment / orchestration solutions, designed to work in conjunction with almost any DevOps tools. This flexibility leads to see these systems as workflow based meta-schedulers.

Cloudify is a platform that basically allows the execution of workflows and operations. It supports out-of-the-box almost any well-known development tool and infrastructure, like Ansible, Chef, or OpenStack, OpenNebula, and workflows to configure, install, heal and scale the applications. It uses TOSCA as a description language to define how the applications should be deployed,

monitored and scaled if necessary, relying on Apache Ambari as TOSCA implementation.

Through plugins, Cloudify can be easily extended to support other tools and infrastructures, as well as to implement new workflows that represent different behaviours to perform over the application (e.g a workflow to scale the application reads a TOSCA file and generates new instances of the components).

Very similar to Cloudify, Apache Brooklyn is a modern meta-scheduler that can also deploy and run applications using popular tools in an agent or agentless mode. The main difference with Cloudify would be that it uses a custom description language not as powerful as TOSCA.

CloudSlang (developed by HP and part of its commercial orchestrator solution) and Mistral (developed by OpenStack) are two tools worth mentioning that, as well as Apache Brooklyn, use their own description language. The main difference with the former ones is that they are more general systems, where no standard work-flows or connection with other tools are provided upfront and have to be implemented from scratch.

Heat, developed by OpenStack, implements an orchestration engine to launch multiple composite cloud applications based on the *heat* description language. It is only compatible with OpenStack and CloudFormation clusters.

Finally, Apache Airavata is an interesting software framework to executing and managing computational jobs on distributed computing resources including local clusters, supercomputers, national grids, academic and commercial clouds. Airavata builds on general concepts of service oriented computing, distributed messaging, and workflow composition and orchestration. Airavata bundles a server package with an API, client software development Kits and a general purpose GUI XBaya as an application registration, workflow construction execution and monitoring. While in theory it is a perfect match to resolve the portal, orchestration and monitoring systems in MSO4SC, its complexity and its immaturity (only 48 commits, first line of code on April 2016) makes it a very risky choice as the base of the entire the MSO4SC intermediate and upper layers.

5.1.2 Monitoring

Several tools are currently available to measure both hardware infrastructures and applications status. Due the large amount of software available, a complete presentation of the state of the art would be impossible to manage in this document. Instead the most relevant and known ones are presented (See [5] for an older review of monitoring tools):

Nagios

Nagios is an open source tool that provides monitoring and reporting for network services and host resources. The entire suite is based on the open-source Nagios Core which provides monitoring of all IT infrastructure

components - including applications, services, operating systems, network protocols, system metrics, and network infrastructure. Nagios does not come as a one-size-fits-all monitoring system with thousands of monitoring agents and monitoring functions; it is rather a small, lightweight system reduced to the bare essential of monitoring. It is also very flexible since it makes use of plugins in order to set up its monitoring environment.

Nagios Fusion enables administrators to gain insight into the health of the organisation's entire network through a centralised view of their monitored infrastructure. In addition, they can automate the response to various incidents through the usage of Nagios Incident Manager and Reactor. The Network Analyser, which is part of the suite, provides an extensive view of all network traffic sources and potential security threats allowing administrators to quickly gather high-level information regarding the status and utilisation of the network as well as detailed data for complete and thorough network analysis. All monitoring information is stored in the Log Server that provides monitoring of all mission-critical infrastructure components – including applications, services, operating systems, network protocols, systems metrics, and network infrastructure.

Icinga

Icinga is an open-source network and system monitoring application which was born out of a Nagios fork. It maintains configuration and plug-in compatibility with the latter. Its new features are as follows:

- A modern Web 2.0 style user interface;
- An interface for mobile devices;
- Additional database connectors (for MySQL, Oracle, and PostgreSQL);
- RESTful API.

Currently there are two flavours of Icinga that are maintained by two different development branches: Icinga 1 (the original Nagios fork) and Icinga 2 (where the core framework is being replaced by a full rewrite). In both of them, the metrics support is similar to Nagios.

Sensu

Sensu is a lightweight framework that is simple to extend and use. It has a user-friendly UI and a lot of plugins, being able to run Nagios plugins as well. It relies on local agents to run checks and pushing results to an AMQP broker. A number of servers ingest and handle the result of the health checks from the broker. This model is more scalable than Nagios, as it allows for much more horizontal scaling and a weaker coupling between the servers and agents. However, the central broker has scaling limits and acts as a single point of failure in the system.

Shinken

Shinken is an open source system and network monitoring application. It is fully compatible with Nagios plugins. It started as a proof of concept for a new Nagios architecture, but since the proposal was turned down by the Nagios authors, Shinken became an independent tool. It is not a fork of Nagios; it is a total rewrite in Python. It watches hosts and services, gathers performance data and alerts users when error conditions occur and again when the conditions

clear. Shinken's architecture is focused on offering easier load balancing and high availability capabilities. The main differences and advantages towards Nagios are:

- A more efficient distributed monitoring and high availability architecture
- Graphite integration in the Web UI
- Better performance, mostly due to the use of a distributed database (MongoDB)

Zabbix

Zabbix is an open source, general-purpose, enterprise-class network and application monitoring tool that can be customised for use in mostly any infrastructure. It can be used to automatically collect and parse data from monitored cloud resources. It also provides distributed monitoring with centralised web administration, a high level of performance and capacity, JMX monitoring, SLAs and ITIL KPI metrics on reporting, as well as agent-less monitoring. An OpenStack Telemetry plugin for Zabbix is already available.

Using Zabbix, the administrator can monitor servers, network devices and applications, gathering statistics and performance data. Monitoring performance indicators such as CPU, memory, network, disk space and processes can be supported through an agent, which is available as a native process for Linux, UNIX and Windows platforms.

Graphite

Graphite is an excellent open source tool for handling visualizations and metrics. It has a powerful querying API and a fairly feature-rich setup. In fact, the Graphite metric protocol is often chosen the de facto format for many metrics gatherers. However, Graphite is not always a straightforward tool to deploy and use. It runs into some issues on large scale configurations, due to its design and its use of huge amounts of small I/O operations, and can be hard to deploy.

It focuses on being a passive time series database with a query language and graphing features. Any other concerns are addressed by external components.

It stores numeric samples for named time series, where metric names consist of dot-separated components which implicitly encode dimensions. They are stored on local disk in the Whisper format, an RRD-style database that expects samples to arrive at regular intervals. Every time series is stored in a separate file, and new samples overwrite old ones after a certain amount of time.

Prometheus

Prometheus is a full monitoring and trending system that includes built-in and active scraping, storing, querying, graphing, and alerting based on time series data. It has knowledge about what the world should look like (which endpoints should exist, what time series patterns mean trouble, etc.), and actively tries to find faults.

It features:

- A multi-dimensional data model, where data can be sliced and diced along multiple dimensions like host, service, endpoint and method.

- Operational simplicity: Easiness to set up monitoring anywhere without being an expert through configuration files.
- Scalable and decentralized, for independent and reliable monitoring.
- A powerful query language that uses the data model for meaningful alerting and visualisation.

Prometheus servers scrape (pull) metrics from instrumented jobs. If a service is unable to be instrumented, the server can scrape metrics from an intermediary push gateway. There is no distributed storage. Prometheus servers store all metrics locally. They can run rules over this data and generate new time series, or trigger alerts. Servers also provide an API to query the data.

It encodes dimensions explicitly as key-value pairs (labels) attached to a metric name. This allows easy filtering, grouping, and matching by these labels via in the query language.

Therefore, it is able to expose the internal state of your applications. By monitoring this internal state, we can throw alerts and act upon certain events. For example, if the average request rate per second of a service goes up, or the fifty percent quantile response time of a service suddenly passes a certain threshold, we could act upon this by upscaling the service.

Prometheus servers know which targets to scrape from due to service discovery, or static configuration. Service discovery is more common and also recommended, as it allows you to dynamically discover targets.
Depending on the type of values that will generate the time series, metrics can be defined by some of the following metrics type:

- A counter is a metric which is a numerical value that is only incremented, never decremented. Examples include the total amount of requests served, how many exceptions that occur, etc.
- A gauge is an instantaneous metric value that is created via incrementing, decrementing or accumulation. An example could be memory usage, CPU usage, amount of threads, or perhaps a temperature.
- A histogram is a metric that samples observations. These observations are counted and placed into configurable buckets. Upon being scraped, a histogram provides multiple time series, including one for each bucket, one for the sum of all values, and one for the count of the events that have been observed. A typical use case for a histogram is the measuring of response times.
- A summary is similar to a histogram, but it also calculates configurable quantiles.

Not everything can be instrumented. Third-party tools that do not support Prometheus metrics natively can be monitored with exporters. Exporters can collect statistics and existing metrics, and convert them to Prometheus metrics. An exporter, just like an instrumented service, exposes these metrics through an endpoint, and can be scraped by Prometheus.

Prometheus has large number of exporters that export metrics from several systems such as Nginx, Mongo, Jenkins, Slurm, Mesos or application logs. Those

exporters are written with official client libraries in different common languages. This allows you to generate highly granular data which you can query. However, this technique is not much different than logging.

It has been chosen as the base platform to perform the monitoring part, mainly because it provides wide documentation and possibilities to build new exporters and its data base provides us with a powerful query language and time response that can be easily used by the orchestrator and portal.

Diamond

Firstly developed to publish metrics to Graphite, Diamond is a python daemon that collects system metrics and publishes them into different systems through its handler API. It is capable of collecting CPU, memory, network, I/O, load and disk metrics. Additionally, it features an API for implementing custom collectors for gathering metrics from almost any source.

Its strength lies on its simplicity, which have led into the possibility to get metrics from most common sources and applications, and send it to most common monitoring systems, without developing any code.

collectd, StatsD

Cloud instances may also be monitored by using a collection of separate open source tools. collectd is a daemon which collects system performance statistics periodically and provides mechanisms to store the values in a variety of ways. collectd gathers statistics about the system it is running on and stores this information. These statistics can then be used to find current performance bottlenecks (i.e. performance analysis) and predict future system load (i.e., capacity planning). It is written in C for performance and portability, allowing it to run on systems without scripting language or cron daemon, such as embedded systems. At the same time it includes optimizations and features to handle big amounts of data sets. StatsD is a Node.JS daemon that listens for messages on a UDP to TCP port. StatsD listens for statistics, like counters and timers and then parses the messages, extracts metrics data, and periodically flushes the data to other services in order to build graphs. These tools are commonly used as "gatherers" for more complex tools like Graphite or Prometheus.

Zenoss

Zenoss is an open source monitoring platform released under the GPLv2 license. It provides an easy-to-use web UI to monitor performance, events, configuration, and inventory. Zenoss is one of the best options for unified monitoring as it is cloud-agnostic and is open source. Zenoss provides powerful plug-ins named Zenpacks, which support monitoring on hypervisors (ESX, KVM, Xen and HyperV), private cloud platforms (CloudStack, OpenStack and vCloud/vSphere), and public cloud (AWS).

Ganglia

Ganglia is a scalable distributed system monitor tool for high-performance computing systems such as clusters and grids. Its structure is based on a hierarchical design using a tree of point-to-point connections among cluster

nodes. Ganglia is based on an XML data representation, XDR for compact and RRDtool for data storage and virtualisation. The Ganglia system contains:

- Two unique daemons, gmond and gmetad
- A PHP-based web front-end
- Other small programs

gmond runs on each node to monitor changes in the host state, to announce applicable changes, to listen to the state of all Ganglia nodes via a unicast or multicast channel based on installation, and to respond to requests. gmetad (Ganglia Meta Daemon) polls at regular intervals a collection of data sources, parses the XML and saves all metrics to round-robin databases. Aggregated XML can then be exported.

The Ganglia web frontend is written in PHP. It uses graphs generated by gmetad and provides the collected information like CPU utilisation for the past day, week, month, or year. Ganglia has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes. However, further work is required in order for it to become more cloud-agnostic.

SeaLion

SeaLion is a cloud-based system monitoring tool for Linux servers. It installs an agent in the system, which can be run as an unprivileged user. The agent collects data at regular intervals across servers and this data will be available on your workspace. Sealion provides a high-level view (graphical overview) of Linux server activity. The monitoring data are transmitted over SSL to the SeaLion servers. The service provides graphs, charts and access to the raw gathered data.

MonALISA

MONitoring Agents using a Large Integrated Services Architecture (MonaLISA) is based on Dynamic Distributed Service Architecture and is able to provide complete monitoring, control and global optimisation services for complex systems. The MonALISA system is designed as a collection of autonomous multi-threaded, self-describing agent-based subsystems which are registered as dynamic services, and are able to collaborate and cooperate in performing a wide range of information gathering and processing tasks.

The agents can analyse and process the information in a distributed way, in order to provide optimization decisions in large-scale distributed applications. The scalability of the system derives from the use of a multi-threaded execution engine, that hosts a variety of loosely coupled self-describing dynamic services or agents, and the ability of each service to register itself and then to be discovered and used by any other services, or clients that require such information. The system is designed to easily integrate existing monitoring tools and procedures and to provide this information in a dynamic, customized, self-describing way to any other services or clients.
By using MonALISA the administrator is able to monitor all aspects of complex systems, including:

- System information for computer nodes and clusters;

- Network information (traffic, flows, connectivity, topology) for WAN and LAN;
- Monitoring the performance of applications, jobs or services; and
- End-user systems and end-to-end performance measurements.

OpenStack Telemetry/Ceilometer

The goal of the Telemetry project within OpenStack, is to reliably collect measurements of the utilisation of physical and virtual resources, comprising deployed clouds, store such data for offline usage, and trigger actions on the occurrence of given events. It includes three different services (Aodh, Ceilometer and Gnocchi), providing the different stages of the data monitoring functional chain: Aodh delivers alarming functions, Ceilometer deals with data collection, Gnocchi provides a time-series database with resource indexing.

The actual data collection service in the Telemetry project is Ceilometer. Ceilometer is an OpenStack service which performs collection of data, normalizes and duly transforms them, making them available to other services (starting from the Telemetry ones). Ceilometer efficiently collects the metering data of virtual machines (VMs) and the computing hosts (Nova), the network, the Operating System images (Glance), the disk volumes (Cinder), the identities (Keystone), the object storage (Swift), the orchestration (Heat), the energy consumption (Kwapi) and also user-defined meters.

Monasca

Monasca is an OpenStack project, aiming at developing an open-source multi-tenant, highly scalable, performant, fault-tolerant monitoring-as-a-service solution, which is integrated within the OpenStack framework. Monasca uses a REST API for high-speed metrics processing and querying, and has a streaming alarm and notification engine. Monasca is being developed by HPE, Rackspace and IBM.

Monasca is conceived to scale up to service provider level of metrics throughput (in the order of 100,000 metrics/sec). The Monasca architecture is natively designed to support scaling, performance and high-availability. Retention period of historical data is not less than one year. Storage of metrics values, and metrics database query, use an HTTP REST API. Monasca is multi-tenant, and exploits OpenStack authentication mechanisms (Keystone) to control submission and access to metrics.

The metric definition model consists of a (key, value) pair named dimension. Basic threshold-based real-time alarms are available on metrics. Furthermore, complex alarm events can be defined and instrumented, based on a simple description grammar with specific expressions and operators.

Gnocchi

Gnocchi is a project incubated under the OpenStack Telemetry program umbrella, addressing the development of a TDBaaS (Time Series Database as a Service) framework. Its paramount goal is to fix the significant performance issues experienced by Ceilometer in the time series data collection and storage. The root cause of such issues is the highly generic nature of Ceilometer's data model, which gave the needed design flexibility in the initial

OpenStack releases, but imposed a performance penalty which is no longer deemed acceptable (storing a large amount of metrics on several weeks makes substantially collapse the storage backend). The current data model on one hand encompasses many options never appearing in real user requests, on the other hand does not handle use cases which are over complex or too slow to be run. From the aforementioned remarks, the idea of a brand new solution for metrics sample collection was ignited, which brought to the inception of Gnocchi.

vSphere

The vSphere statistics subsystem collects data on the resource usage of inventory objects. Data on a wide range of metrics is collected at frequent intervals, processed and archived in a database. Statistics regarding the network utilisation are collected at Cluster, Host and Virtual Machine levels. In addition vSphere supports performance monitoring of guest operating systems, gathering statistics regarding network utilisation among others.

Amazon CloudWatch

Amazon CloudWatch is a monitoring service for AWS cloud resources and the applications running on AWS. It provides real-time monitoring to Amazon's EC2 customers on their resource utilisation such as CPU, disk and network. However, CloudWatch does not provide any memory, disk space, or load average metrics without running additional software on the instance. It was primarily designed for use with Amazon Elastic Load Balancing and Auto Scaling with load balancing in mind: the service checks CPU usage on multiple instances and automatically creates additional ones when the load increases.

5.2 Features

As the components in charge of performing the deployments and execution of all applications, and know *what's going on* in the entire platform, the orchestrator and monitor systems play a key role into achieving the goals of the MSO4SC.

Therefore, the features that the orchestrator provides are:

- Hybrid and multi provider cloud (support for multiple HPC and VMs providers).
- Common deployment operations.
    - Build software
    - Data movement
    - Execute/Copy scripts, binary files
    - Virtual Machine and container creation and provisioning
- Deployment and execution requirements.
- Communication with an external monitor system.
- Smart decisions on where to deploy and run what.

- Re-deploy and re-schedule of jobs when infrastructure over infrastructure state changes
- Human interaction to reconfigure the executions "on the fly".
- Output management.
    - Infrastructure and application logs
    - Generated data

On the other side, the monitor functionalities are:

- Collect metrics from different infrastructures, normalized, and gathered into a common storage system
    - Metrics from different HPC infrastructures and workload managers
    - Metrics from different Cloud providers (Virtual Machines).
- Collect custom and normalized metrics from the logs generated by the applications.
- Alerts on relevant events
    - Infrastructure down
    - Deployment/Execution failed/succeeded
- User-friendly visualization
- Efficient monitoring data storage, taking into account the time-series nature of the information

5.3 Design

Figure 10 shows the architecture that follows the monitor and orchestrator high level components, and their interactions with the rest of the platform.

The orchestrator receives the information about deployment and execution through a TOSCA file coming from the MSO Portal. In such file, operations like software compilation, data movements, HPC and Cloud providers to be used, and input datasets and custom parameters of the execution are defined.

With this information, the orchestrator will communicate with the infrastructure to perform the different operations at the correct time, while it exchange information with the monitor to be informed.

The monitor on the other side will get the information from the orchestrator about the jobs that need to be followed, sending metrics about these jobs and the general status of the infrastructure to the orchestrator and the portal. In the MSO Portal, these metrics are presented to the user in a friendly way.

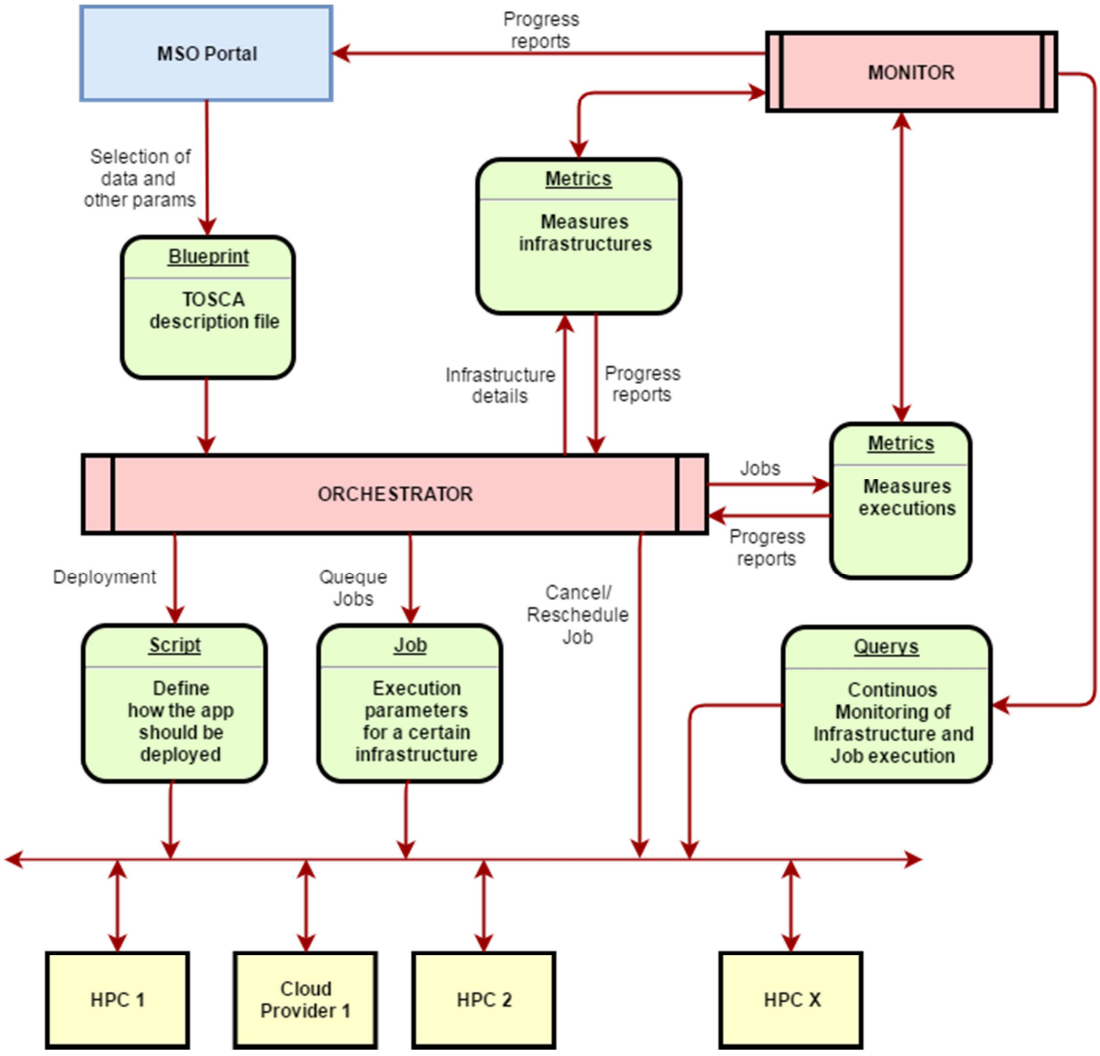


Figure 10. Orchestrator and Monitor Architecture

5.4 Implementation and software components

The Orchestrator & Monitor solution relies on different open source components, as well as our own MSO4SC software, that extends them and enables the overall behaviour we need.

5.4.1 Orchestrator implementation

The orchestrator kernel is based on Cloudify (Apache ARIA), the open source TOSCA description language implementation of reference. Over it, different types of nodes and relationships are defined and implemented to model HPC infrastructures and executions. Also deployment operations as data movement or software management are implemented as TOSCA operations. Finally, as batch executions (typical HPC-like executions) are different from long-term executions (typical cloud executions), special workflows have been implemented to deploy and run the simulations, enabling monitoring while the execution is performed.

The result is an orchestrator that, receiving a TOSCA description file, cannot only execute long-term operations (like running a server), but also execute batch

operations (short or long duration), while monitoring them. Moreover, apart of using VMs as infrastructure, it can now use HPC infrastructures as computation nodes as well, making a smart use of the available resources.

Besides, the orchestrator implements two different REST APIs to connect and interact with the MSO Portal and Monitoring System, running as a service within the MSO4SC architecture.

5.4.2. Monitor implementation

The storage of metrics coming from different parts of the platform is implemented by InfluxDB. InfluxDB is a database designed specifically to store time-series data. On top of it, two well-known open source tools, Prometheus and Grafana, are used to collect the metrics and visualize the results respectively.

In the case of Prometheus, it will connect with different *exporters* developed within the project to collect the pertinent metrics. Each exporter works as an independent application that remotely gathers information about the infrastructure, application logs or other interesting data to collect, accumulating this data and exposing it to Prometheus through HTTP. Moreover, alerts are defined using Prometheus Alert Manager, allowing the platform to warn users when relevant events occur.

The orchestrator will gather the last metrics from Prometheus DB, while Grafana will be embedded into the MSO Portal for end-user metrics consultation (see next section).

6. Portal

The MSO Portal will be the user-friendly interaction mechanism between the end users and the MSO4SC platform. From its frontend the user will be able to use all the functionalities the project provides: run the MSO4SC experiments software with pre and post operations and monitor it while executing, apart from login into the system, manage the data available, visualize it, etc. Its components are described in detail in D.2.2 [4] and shown in figure 11.

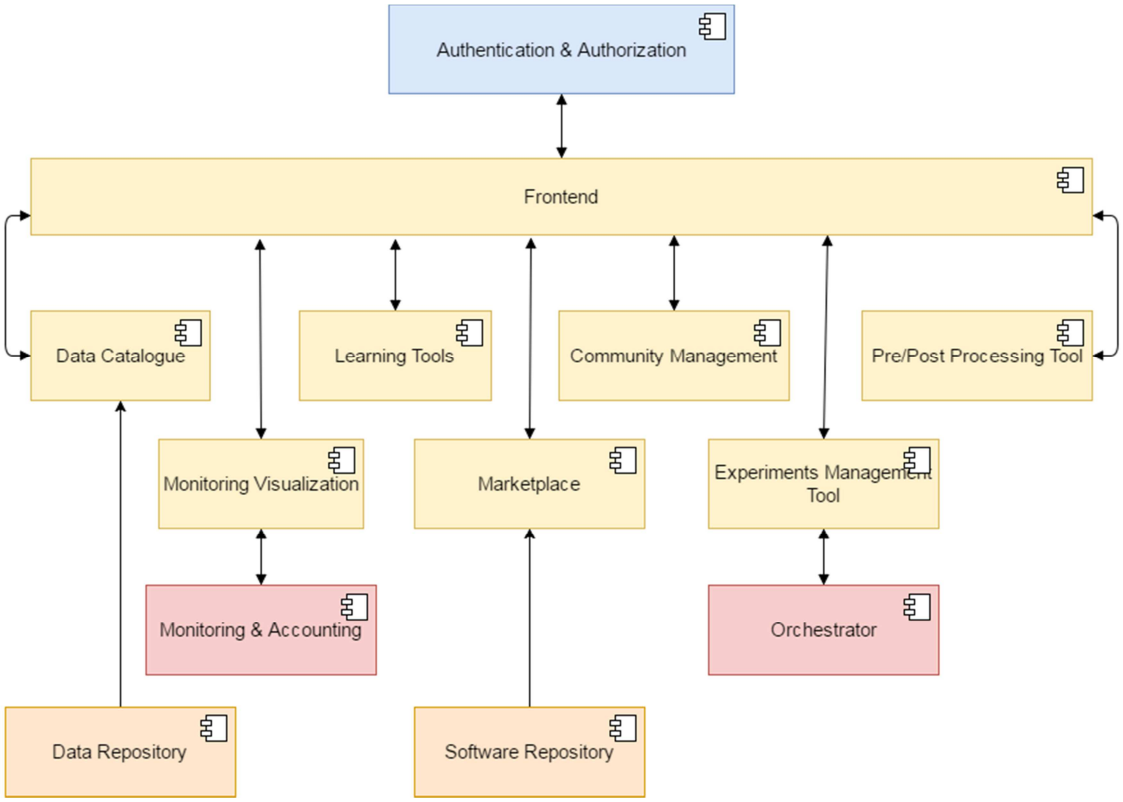


Figure 11. MSO4SC Portal high-level architecture [4]

6.1 Frontend

This component serves the user with a nice interface in which the user will be able to access to the different functionalities in the portal. It is implemented using Django, well-known python web framework. It embeds the other components of the portal, acting as the "landing page" and user management (register, login, etc) of the entire platform.

6.2 Data Catalogue

It presents the data available in the system, no matter where it is actually stored, providing easy ways to manage and select the datasets to be used by the rest of the modules. FIWARE CKAN catalogue is implementing this module.

In the image shown in figure 12, a real dataset representing an oil field is displayed. Datasets belong to an organization and can be formed of more than one data file. Following the DCAT format, several tags are used to describe it (author, last updated, license…) where one of those tags is "source" representing the real storage location. All the different sources will be part of the data repository, designed to deal with heterogeneous storage systems. Other custom tags can also be defined, providing the catalogue with strong categorization and filtering capabilities.

In order to customize the CKAN with the features we need, the MSO instance will include several extensions: ckanext-oaimph (for improving harvesting capabilities), disqus (for enabling comments), dcat (for metadata import/export) and ldap (for LDAP integration). Also, the consortium will analyse the need of implementing new extensions (i.e. for easing data movement).

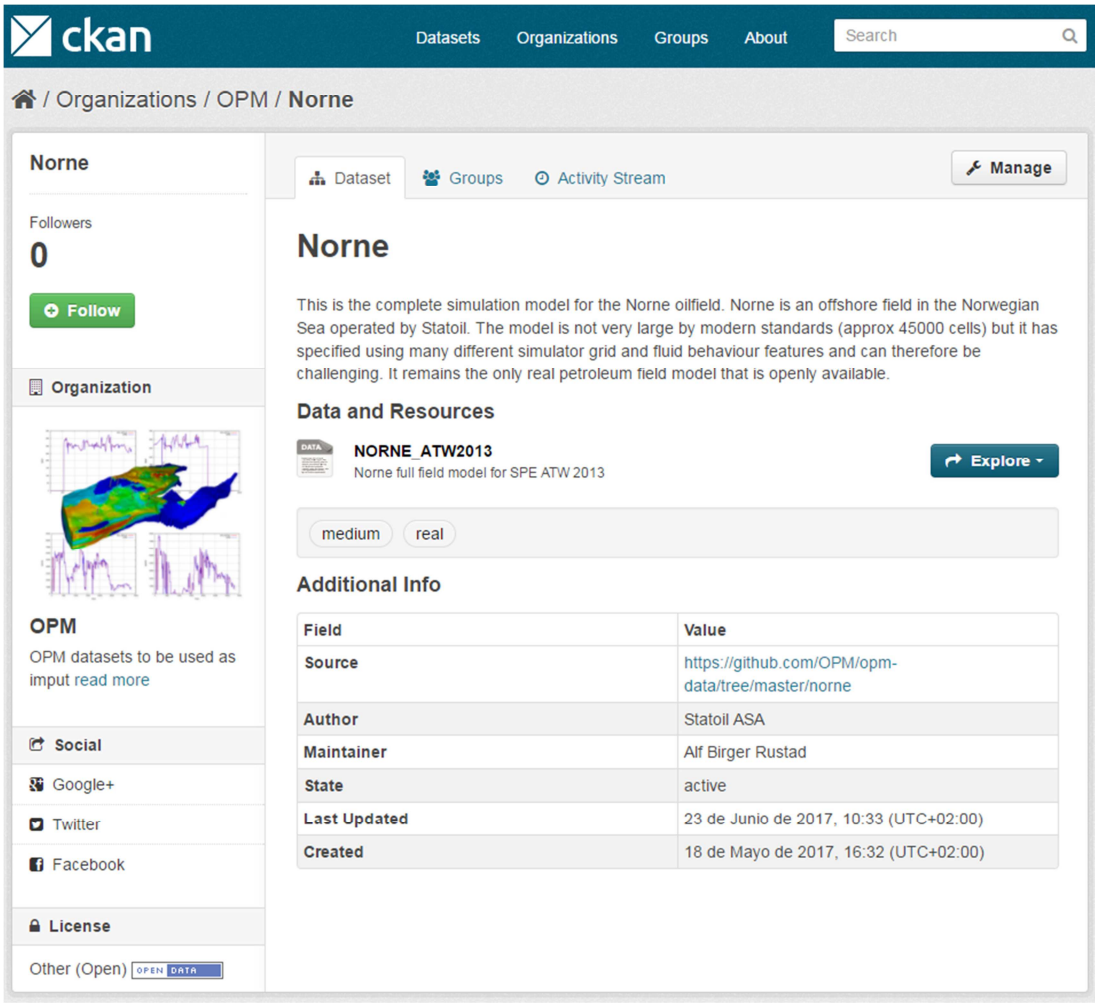


Figure 12. Norne Oil Filed dataset in CKAN

6.3 Monitoring dashboard

This component is in charge of rendering and presenting to the user the monitor data generated by the monitor service, so he/she will be able to know what is happening in any moment and actuate in concordance, controlling the simulation execution.

Figure 13. Example of FinisTerrae-II infrastructure dashboard with Grafana

The implementation relies on Grafana, a tool that shows graphically the data stored in the monitor (InfluxDB). From it different "dashboards" can be set for every application and infrastructure, while the users can also create (or edit) new dashboards to adapt the visualization to their needs. An example of a dashboard is shown in figure 13.

6.4 Marketplace

In this section of the MSO4SC portal the user will find a catalogue of the applications available in the platform, and will be able to upload, update and select each of them for execution.

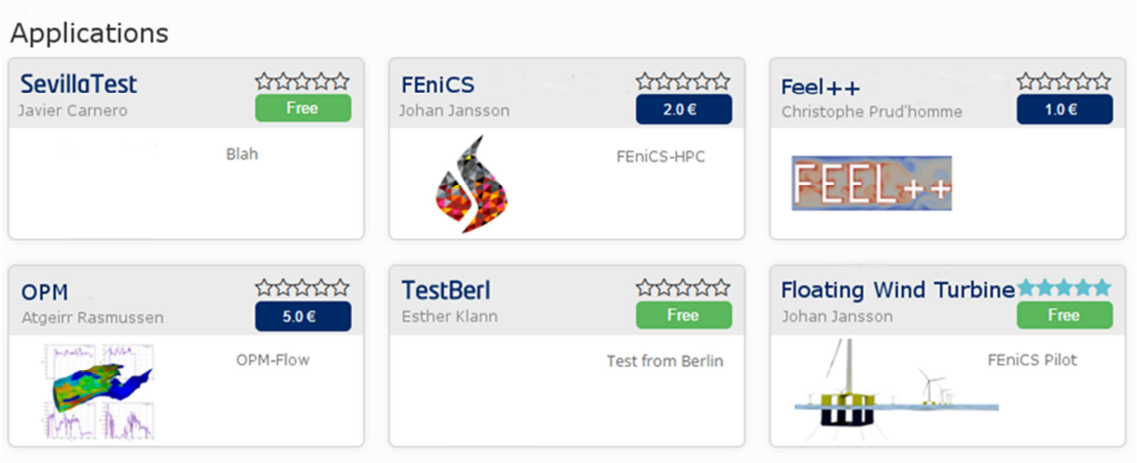


Figure 14. MSO4SC Marketplace using FIWARE Business Framework

FIWARE Business framework will be used to implement this service. It will communicate with the software repository to perform the operations, like for example provide the path where the software is stored when an application is selected to be executed. Figure 14 shows the prototype for the portal.

We will implement a custom connector with our orchestrator, instead of using a normal Cloud deployment and the framework will be customized according to the styles and product categories that the MSO Portal requires.

6.5 Community Management

The MSO4SC platform has to be aware of the different scientific communities that use the system. To achieve that, this module manages the information, datasets and end user applications that are presented to the user of the portal.

It is implemented within the Django framework, integrated in the frontend, and works with the authorization module to give access to above resources taking into account the user organization and privileges. Askbot will also be used in order to enable spaces for discussion between community members.

6.6 Learning Tools

Two different tools configure the learning functionalities of the portal. One is Moodle, a learning platform that allow us to provide tutorials, courses, videos, and any relevant content that could help the end-user to fast use of the platform and concrete applications. The other is Askbot, a Q&A system thought to provide quick support and proximity between users and developers/application maintainers. Figure 15 and 16 show different screenshots of Moodle and Askbot.

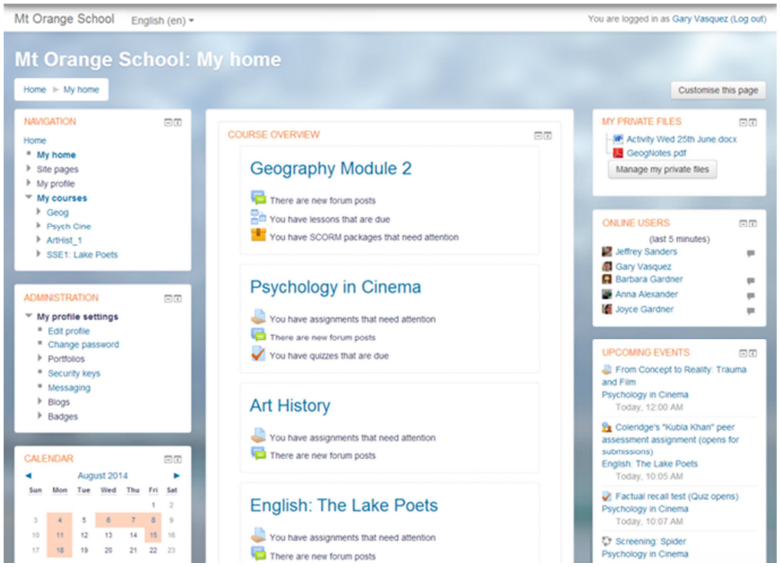


Figure 15. Moodle

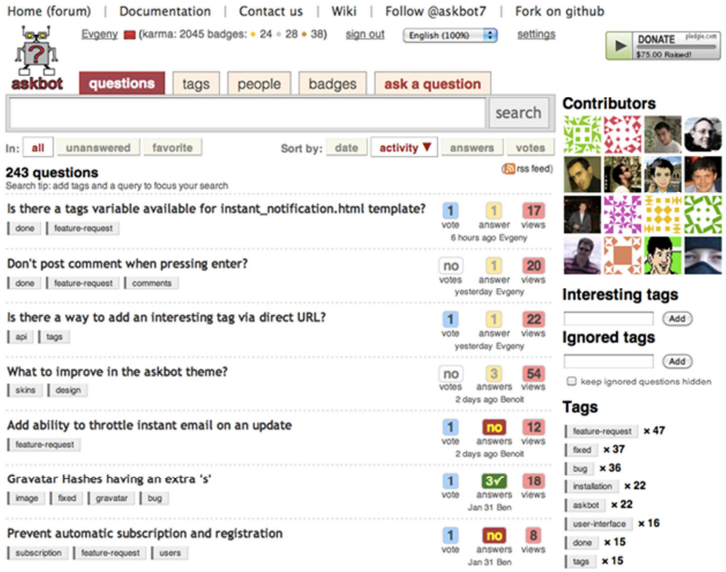


Figure 16. Askbot

6.7 Experiments Management Tool

This module supports the deployment and execution workflows of an application, communicating with the orchestrator through a REST API.

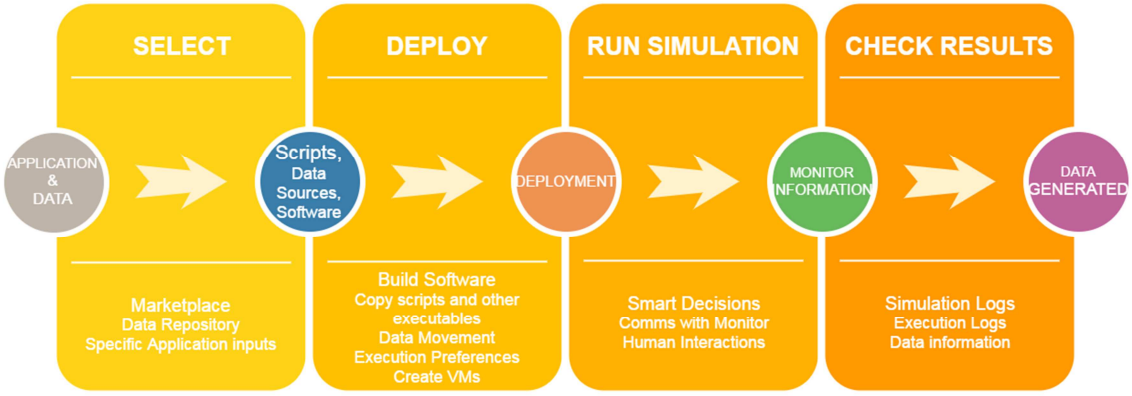


Figure 17. Experiments Workflow

It is built again using the Django framework. The module let choose an application from the marketplace, as well as dataset(s) from the data catalogue and other input information. Then it composes all this information into a TOSCA blueprint and sends it to the orchestrator. Also the deployment and execution of a concrete simulation are started from here, as well as further interactions with the running application. Figure 17 show the workflow for the deployment of the experiments.

6.8 Visualization and Pre and Post Processing tools

Scientific software is solving bigger and bigger problems every day. The resolution of these kinds of problems usually produces a large amount of data that needs to be generated, stored and analysed, frequently through scientific visualization software.

The full life-cycle of analysis with mathematical frameworks usually involves three separated stages: pre-processing, processing and post-processing.

The pre-processing stage generates the input data required by the simulation. The processing stage takes the input data, performs the simulation and produces the results, and the post-processing stage deals with the representation and visualization of results.

Some tools like Salome, ParaView and ResInsight have been installed in order to avoid costly data movement of huge datasets and to satisfy this life-cycle from the infrastructure. These three applications provide graphical user interfaces to interact, modify, render and visualize the datasets.

A common solution to integrate this heterogeneous environment of tools and improve the user experience is to use remote desktop technologies. The solution adopted in this project is noVNC.

These technologies have been tested with a demonstrator and are ready to be integrated within the portal.

noVNC

noVNC is a VNC implementation based on HTML5 that supports modern browsers, including mobile browsers.

VNC (Virtual Network Computing) is a popular graphical desktop sharing system that uses the Remote Frame Buffer protocol (RFB), an open simple protocol for remote access to graphical user interfaces. It shares the keyboard and mouse events from the host in order to remotely control another computer, relaying the graphical screen updates back in the other direction, over a network.

noVNC has a client-server architecture. The server is the one installed in the remote machine which allows share the screen and to control it. The web client shows the remote screen from a web browser, receives updates from it and captures the user interaction to communicate and interact with the server.

Salome

Salome is open source software that provides a generic platform for pre-processing and post-processing for numerical simulation. It can generate geometric models, prepare data for numerical calculations and visualize the calculation results, but it also can integrate third party numerical codes to produce new applications.

Salome provides a wide set of features and catalogue of algorithms to deal with geometric models and finite element meshes, and it also manages common and widely extended file formats.

Figure 18 shows a screenshot of Salome.

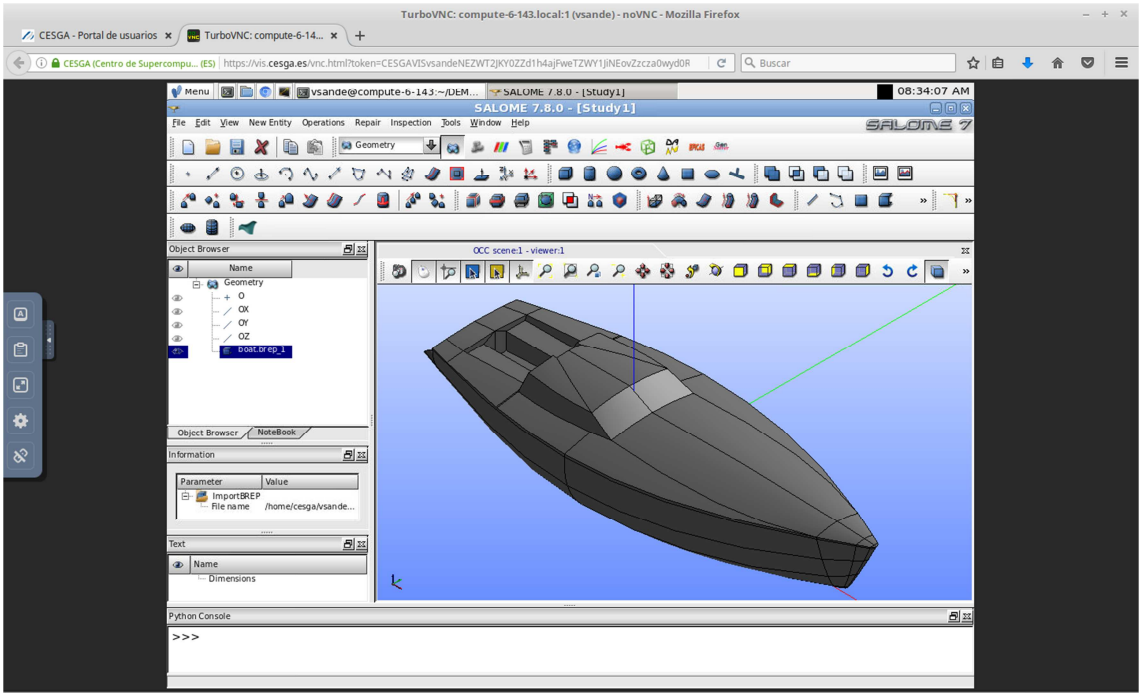


Figure 18. Salome running in a noVNC remote desktop web

The user interaction can be done interactively, by using the graphical user interface (GUI), or in non-interactive mode, with the text user interface (TUI) through Python scripts.

Paraview

ParaView is popular open source data analysis and visualization software for scientific visualization.

ParaView was designed for data parallelism, using parallel and distributed file system, and is able to visualize and analyse extremely large datasets using distributed memory computing resources, although it can be run on distributed and shared memory parallel and single processor systems.

The user interaction can be done interactively or programmatically using ParaView's batch processing capabilities. Figure 19 shows Paraview running in a web browser via noVNC.

Paraview also provides several options to perform remote visualizations. It is designed to work well in client/server mode. In this way, users can have the full advantage of using a shared remote high-performance rendering cluster in order to visualize in their personal computers.

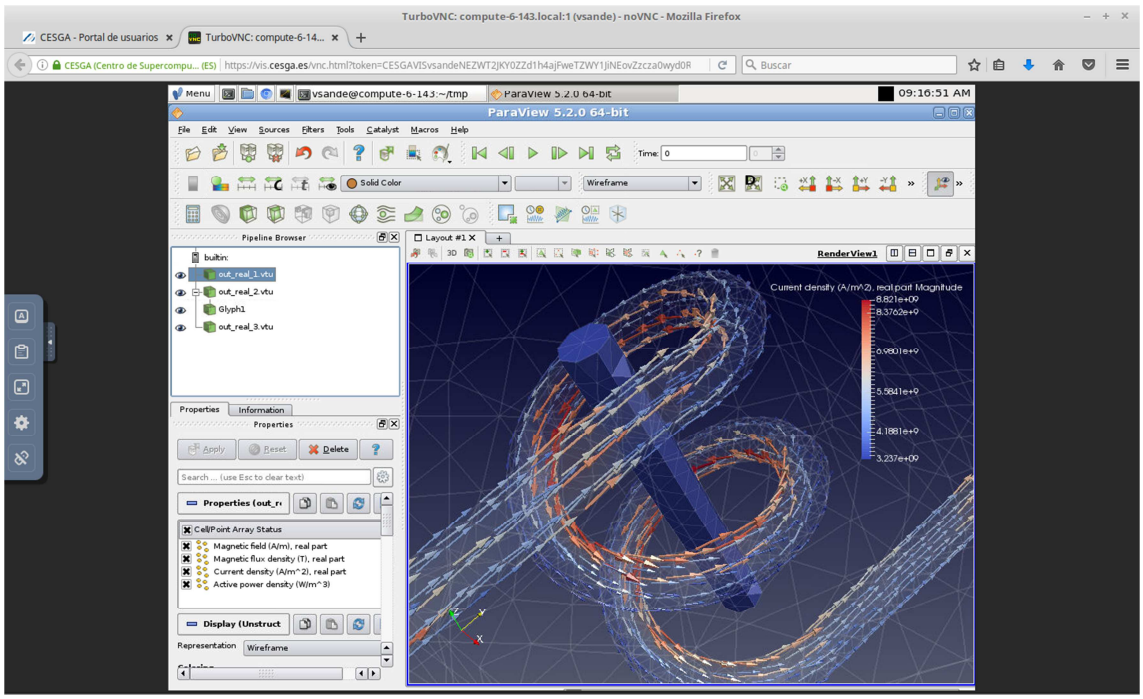


Figure 19. Paraview running in a noVNC remote desktop web

ParaView project also provides ParaViewWeb, a web framework to build applications with interactive scientific visualization inside the Web browser. Those applications can leverage a VTK and/or ParaView backend for large data processing and rendering, but can also be used on static Web server, a high-performance HTTP server or even locally with command line based application using your browser.

ResInsight

ResInsight is an open source, cross-platform 3D visualization, curve plotting and post processing tool for Eclipse reservoir models and simulations. It can also be configured to visualize geomechanical simulations from ABAQUS.

The system also constitutes a framework for further development and can be extended to support new data sources and visualization methods, e.g. additional solvers, seismic data, CSEM, and more.

The user interface is tailored for efficient interpretation of reservoir simulation data with specialized visualizations of properties, faults and wells. It enables easy handling of a large number of realizations and calculation of statistics. To be highly responsive, ResInsight exploits multi-core CPUs and GPUs. Efficient

plotting of well log plots and summary vectors are available through selected plotting features.

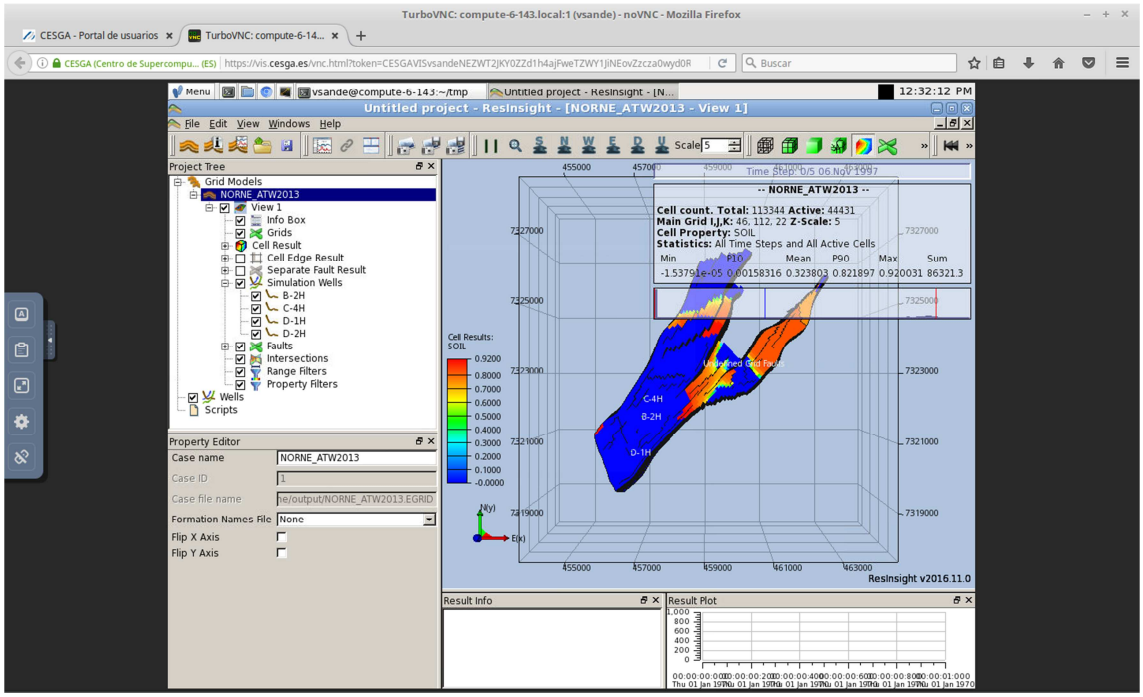


Figure 20. ResInsight running in a noVNC remote desktop via web

Pre and post operations needed to be done over the datasets of a simulation will be managed by this tool, controlled by the end user through the frontend. Figure 20 is a screenshot of ResInsight running in a web browser.

6.9 Authentication & Authorization

While this module is horizontal, and it is used by mostly all submodules and components in the system, we can consider it part of the MSO Portal as it is a requirement that arises from its design as a compound of different services.

The module is in charge of authenticating users and services in the portal, and then give permission or not to perform operations in the platform and communicate with other services.

For the authentication part, LDAP is used in combination with custom software. As it is commonly implemented, a user or service will provide credentials that will be verified using LDAP. If the item is successfully authenticated, a token will be returned.

On the other hand the authorization mechanism is performed using the well-known OAuth2 protocol and the Django framework. Using the token from the previous step, the services or users will provide it when asking to perform an operation on other service. The receiver will check with the authorization module if the service with the token provided have permissions to perform the operations. Finally the operation will be performed or rejected depending on the answer by the authorization part.

As tool for facilitating single-sign-on, the consortium is considering to use Shibboleth, an open source tool with the features the MSO Portal needs.

7. Software Repository and Automated Integration and Deployment

The Software repository is intended to be the place where to store the applications-related environment like meta-data, test suites, benchmarks, software, containers and also the e-infrastructure. It is a storage location from which software packages may be accessed and executed by end-users.

Providing a software repository accessible from a single place (the portal) will help to homogenize applications usage and to increase the visibility and the impact of the provided data and applications. The repository will be distributed along the involved infrastructures, but all data and applications stored in the repository will be accessed/distributed via computer networks. This data will be public or protected by control access through the previously exposed authentication and authorization methods.

The software repository will be supported in a backup system, storing redundant information. This information could be retrieved in case of a catastrophic problem, avoiding data losing and helping to mitigate other possible risks.

The repository will also include technologies and tools in order to ease management, development, integration and deployment of MADFs and Pilots in the infrastructure. At least, it will integrate a version control system like Git, containerization technologies like Singularity and third party tools to provide automatic integration and deployment.

The correct functioning of every contained MADF and Pilot must be checked before publishing it on the Portal. Automatic integration and deployment processes will automate this applications validation and deployment. After a successful deployment, applications will be available and ready-to-use in the proper production infrastructure and through the Marketplace.

The use of containers has been adopted as the way of interacting between the MADFs and different components of the project. The designed high-level flow for automating the integration and deployment of Singularity containers can be seen in Figure 21.

MADFs and Pilots developers will provide the container itself or a way to have it available from the infrastructure, like public container repositories (DockerHub, SingularityHub, etc.) or bootstrap definition files. Once the container is created, it will be automatically ported to the infrastructure and tested. If tests are successful the new container will be available and ready-to-use.

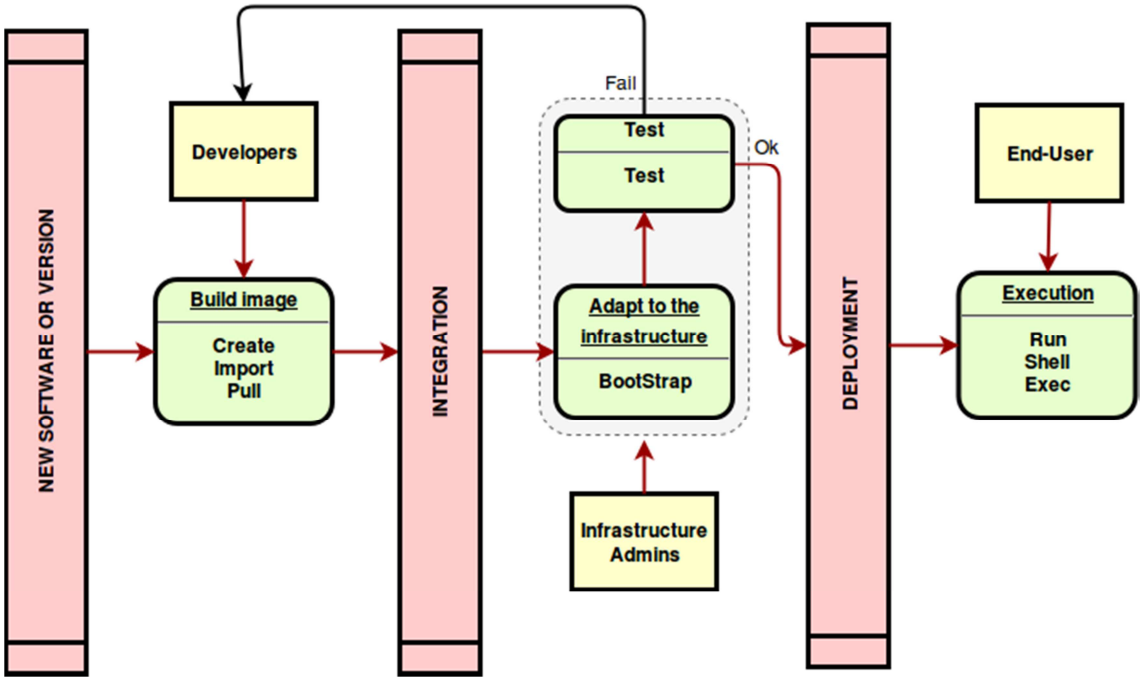


Figure 21. Automatic integration and deployment flow chart

As a part of the communication mechanisms supporting the project structure, a set of public domain repositories has been created taking advance of the services and collaborative tools (like wikis, issue tracking, continuous integration, etc.) provided by GitHub.

These repositories are currently being used to share the software, meta-data and technical documentation, like the deployment process of the frameworks. They are also intended to be the place where to publish the benchmarks for the MADFs. Moreover a special documentation repository using asciidoc format contains general documentation about the project and each MADF and pilot, in a user-friendly presentation as a book.

8. Data Repository

The data repository is composed by two different parts: the data storage and the data movement tool. The first will show the data available in the different storage units (Data catalogue), while the second move datasets from/to the computing infrastructure following the instructions of the orchestrator.

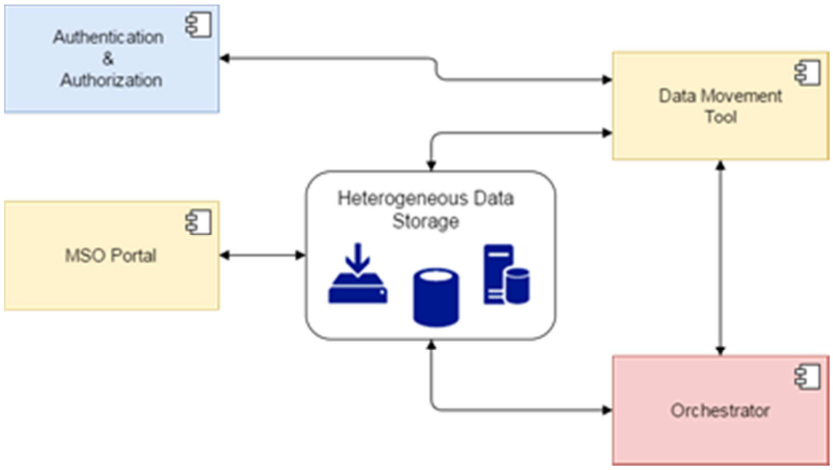


Figure 22. Data Repository architecture [4]

To adequate the repository to the different characteristics and formats of the datasets, the data storage will be formed of several storage units based on different paradigms, such as array databases, relational and NoSQL databases, storage servers, etc. Those will be typically the same storage systems that the users uses to store their data in the infrastructure the usually use.

Relying on specific protocols and tools as GridFTP, the data movement tool will take orders from the orchestrator to move data from and to the different storage units as efficient as possible. By using a standard protocol like GridFTP we will provide encryption of the data transmitted and also higher bandwidth that with other standards, by using multiple simultaneous TCP streams. To connect to the heterogeneous data storage, this component will use the Authentication & Authorization module. During the implementation of this component, in the second implementation, we will analyse the possibility to use other solutions already available, like the ones provided by EUDAT (like B2STAGE and B2DROP) for data movement.

9. Hardware Infrastructure

For the testing, execution and development of the e-Infrastructure, a development and production infrastructure will be available. CESGA will provide access to the FinisTerrae HPC cluster, which is a Singular Research Infrastructure part of the Spanish Supercomputing Network and a Tier-1 PRACE system. This system will be an example on how the complex MADFs and pilots can be deployed in a production HPC system. SZE will provide a test and preproduction infrastructure for testing the software during its development phase and all the changes that cannot be implemented in the production infrastructure. ATOS will be providing also a test a production infrastructure. In the next sections we provide more details about these systems.

9.1 FinisTerrae-II HPC cluster

FinisTerrae-II is the main supercomputing system provided by CESGA. It is a Bull/ATOS HPC supercomputer with 306 servers, each of them with 24 cores

Haswell 2680v3 Intel processor and 128GB of main memory per server. It is connected to a shared Lustre High-performance Filesystem with 768TB of disk space. The servers are interconnected with a low latency Infiniband FDR with a peak bandwidth of 56Gbps. Additionally, the system has 4 GPU servers with GPUs (NVIDIA K80) and 2 servers with Intel Xeon Phi accelerators. There is also one "Fat" node with 8 Intel Haswell 8867v3 processors, 128 cores and 4TB of main memory

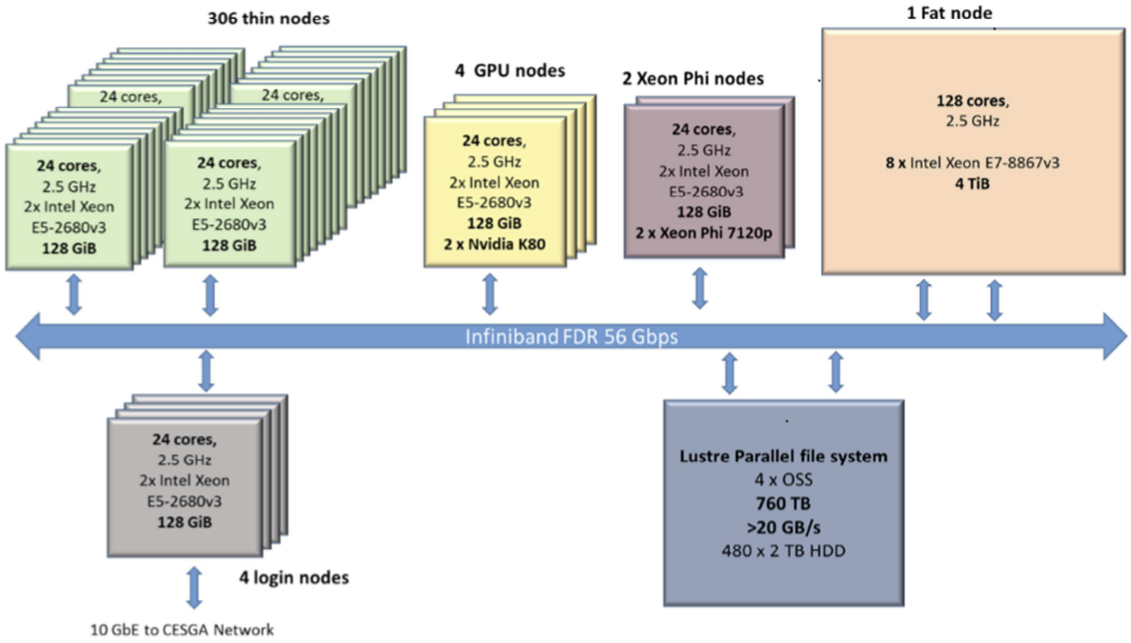


Figure 23. FinisTerrae-II schematic diagram with the configuration of servers and network

9.2 SZE HPC cluster

The SZE HPC cluster called "plexi", consist 26 computing node, which could be divided into two separate group. There are 20 normal compute nodes with 12 Cores and 48 GB Memory each, and 6 GPU node which are housing more than 12 Nvidia Tesla cards M2050 and M2090 with total of 5888 GPU cores. The nodes are connected with Infiniband QDR interconnect which provides 32Gb/s connection speed. These compute nodes are diskless, so we have and 12TB IBM Fibre-Channel 4Gb/s storage which are used to store the boot images and simulation results of the system.

For testing purposes we use a HUAWEI CH140 Blade Server with 24 Haswell CPU Cores and 128GB DDR4 ECC Memory with VMware virtualization. We generated a virtual infrastructure with a head node and many relatively small worker nodes. This virtual infrastructure is ideal for testing the horizontal scalability of the MSO cluster.

9.3 ATOS HPC cluster

During the execution of the project ATOS will incorporate an HPC cluster based on their Sequana system. It is expected that this cluster will be incorporated on the second year of the project.

9.4 CESGA Cloud

In addition to the HPC resources, CESGA will provide access to cloud resources available in the center. This cloud infrastructure is based on OpenNebula cloud management system and delivers a virtual infrastructure, configurable to the requirements of the final users: operating system, number of processors, memory, disk and number of nodes are configured to user´s needs in a dynamic way. This cloud will be used not only for those parts of the pilots that are not suitable to be run in an HPC infrastructure, but also for the services needed in the development of the services and for running these service. For example, to provide a highly available Portal and Orchestrator, two virtual machines running in this cloud will be used.

9,5 Other Infrastructures: PRACE and EGI

During the project we expect to increase the number of physical infrastructures available to the users, including some of the main cloud and HPC research infrastructures in Europe, like PRACE and EGI. These activities will start in the second implementation phase, once the main services are available.

10. Summary and Conclusions

This document presents a detailed description of the components that will be part of the e-Infrastructure and how they are going to be implemented. Some of them have already been deployed and there are pilots and proofs of concepts working in the infrastructure. For example most of the MADFs are already available as containers on the pilot infrastructure. With these descriptions we plan to implement the cloud components according to the WP3 roadmap, in order to have a first version of the software ready by the end of the first year of the project. This first version will be the D3.2 Integrated Infrastructure, Cloud Management and MSO Portal deliverable which should be ready in October 2017.

References

[1] MSO4SC Description of Work (DoA). Annex I to the EC Contract.

[2] Johan Hoffman. Computation of mean drag for bluff body problem using adaptive DNS/LES. SIAM J. Sci. Comput ., 27(1):184-207, 2005.

[3] MSO4SC D2.1End Users' Requirements Report

[4] MSO4SC D2.2 MSO4SC e-Infrastructure Definition

[5] TNOVA D4.42 Monitoring and Maintenance

[6] GridFTP: http://toolkit.globus.org/toolkit/docs/latest-stable/gridftp/

[7] Feel++: http://www.feelpp.org/

[8] FEniCS project: https://fenicsproject.org/

[9] OPM: http://opm-project.org/

[10] Salome: http://www.salome-platform.org/

[11] Paraview: https://www.paraview.org/

[12] ResInsight: http://resinsight.org/

[13] EasyBuild: https://hpcugent.github.io/easybuild/

[14] DockerHub: https://hub.docker.com/

[15] SingularityHub: https://singularity-hub.org/