D4.1 Detailed Specification of the Infrastructure Platform

Document Identification			
Status	Final	**Due Date**	30/06/2018
Version	1.0	**Submission Date**	18/07/2018

Related WP	WP4	**Document Reference**	D4.1
Related Deliverable(s)	D2.1, D2.2	**Dissemination Level (*)**	PU
Lead Participant	USTUTT	**Lead Author**	Michael Gienger
Contributors	ATOS FR ATOS ES	**Reviewers**	Dimitrios Moshou CERTH

Keywords:

e-Infrastructure, Specification, Architecture, Platform, Design, High Performance Computing, High Performance Data Analytics, Big Data, Data Analytics

Document Information

List of Contributors	
Name	Partner
Nico Struckmann	USTUTT
Michael Gienger	USTUTT
F. Javier Nieto	ATOS ES
Fabien Castel	ATOS FR

Document History			
Version	Date	Change editors	Changes
0.1	17/04/2018	N. Struckmann (USTUTT)	Initial draft
0.2	13/06/2018	Michael Gienger (USTUTT)	Template refinement and structure
0.3	20/06/2018	Fabien Castel (ATOS FR) F. Javier Nieto (ATOS ES)	Content in various sections populated
0.4	29/06/2018	N. Struckmann (USTUTT)	Contents finalized
0.5	03/07/2018	N.Struckmann (USTUTT)	Document finalized for Review
0.6	13/07/2018	D,Moshou (CERTH)	Review
0.9.0	18/07/2018	N.Struckmann (USTUTT)	Final version for quality review
1.0	18/07/2018	ATOS	Quality review
FINAL	18/07/2018		Final version ready for submission

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Michael Gienger (USTUTT)	18/07/2018
Technical manager	Fabien Castel (ATOS FR)	18/07/2018
Quality manager	Susana Palomares (ATOS ES)	18/07/2018
Project Manager	F. Javier Nieto (ATOS ES)	18/07/2018

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
API	Application Programming Interface
CI	Continuous Integration
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
HPC	High Performance Computing
HPDA	High Performance Data Analytics
OTC	Open Telekom Cloud
PO	Project Officer
QoS	Qualit of Service
SCP	Secure Copy
SLA	Service Level Agreement
WP	Work Package
WS-Agreement	Web Service Agreement Specification

1 Executive Summary

This document has the purpose to enhance initial ideas described in the EUXDAT Proposal and specify the first version of the EUXDAT e-infrastructure platform. Since this deliverable is due end of M6 it is obvious further ones will follow up during the projects runtime, for a complete list please refer to section 2.1.

It takes WP2 deliverables "D2.1 Description of Proposed Pilots and Requirements [1]" and "D2.2 EUXDAT e-Infrastructure Definition v1 [2]" into account containing identified requirements and also defines the platform's major building blocks.

The initial specification of the infrastructure, jointly discussed amongst involved and affected parties, is highlighted at first in chapter 3. Including computation backend systems and their integration with the central platform services, as well as application management and available software storage and other properties. Followed in chapter 4 by a description of the infrastructure's features mapped to dedicated components and how they interact with each other.

The development concept, deployment mechanisms and chosen software tools are outlined in chapter 5, besides repositories for code management with quality assurance and distinct stages maintaining flexibility for the platform developers while a stable production is ensured.

2 Purpose of the document

This document describes the EUXDAT Infrastructure Platform in its various aspects, such as properties of the computation backends (Cloud and HPC), including hardware properties, available software and libraries, access mechanisms, as well as code repositories and deployment workflows with QA.

It describes in detail the first set of solutions to be implemented related to the hybrid orchestration, the management of data and VMs/jobs, the profiling component, the SLA management, the monitoring infrastructure and the accounting component.

2.1 Relation to other project work

This document, Deliverable "D4.1 Detailed Specification of the Infrastructure Platform v1" is taking into account the "D2.1 Description of Proposed Pilots and Requirements [1]" and "D2.2 EUXDAT e-Infrastructure Definition v1 [2]" to satisfy the identified requirements towards the infrastructure appropriately. It supports WP3 on its mission to build the end-user platform and contributes input to D3.1 "Detailed Specification of the End Users' Platform v1".

This WP4 deliverable is the first of a series over the project runtime. It is followed by:

- D4.2 Infrastructure Platform v1 - Demonstrator (M12)
- D4.3 Detailed Specification of the Infrastructure Platform v2 - Report (M16)
- D4.4 Infrastructure Platform v2 - Demonstrator (M23)
- D4.5 Detailed Specification of the Infrastructure Platform v3 - Report (M26)
- D4.6 Infrastructure Platform v3 - Demonstrator Public (M31)

2.2 Structure of the document

This document is structured in 6 major chapters, with various subsections explaining in-depth several aspects crucial for a full understanding.

Chapter 2 describes the actual purpose of this document and how it is structured.

Chapter 3 is explaining the e-Infrastructure for the EUXDAT Platform and its individual building blocks. The three major building blocks are the platform and associated development environments, besides Cloud and HPC computation backends for the workload processing. The backends are highlighted and detailed information provided about available hardware, applications and libraries.

Chapter 4 describes all core components of the targeted EUXDAT end-user e-Infrastructure Platform, such as workload orchestrator, monitoring solutions and SLA management.

Chapter 5 highlights the development and deployment infrastructure and required services, like git, gerrit, jenkins. Besides explanation of software used, git workflows, code branches and repositories, and the three distinct stages are focused, including how the deployment of code from developer machines onto the infrastructure is intended to be carried out alongside with proper quality assurance and ensuring a stable production.

Chapter 6 finishes the document with a conclusion of the work up to date, challenges addressed and presents the main achievements.

3 Infrastructure Platform Specification

The EUXDAT infrastructure not only comprises the targeted portal for end-users, but in addition also the infrastructure required for development and testing, as well as the connectivity to computing backends, namely Cloud and HPC systems.

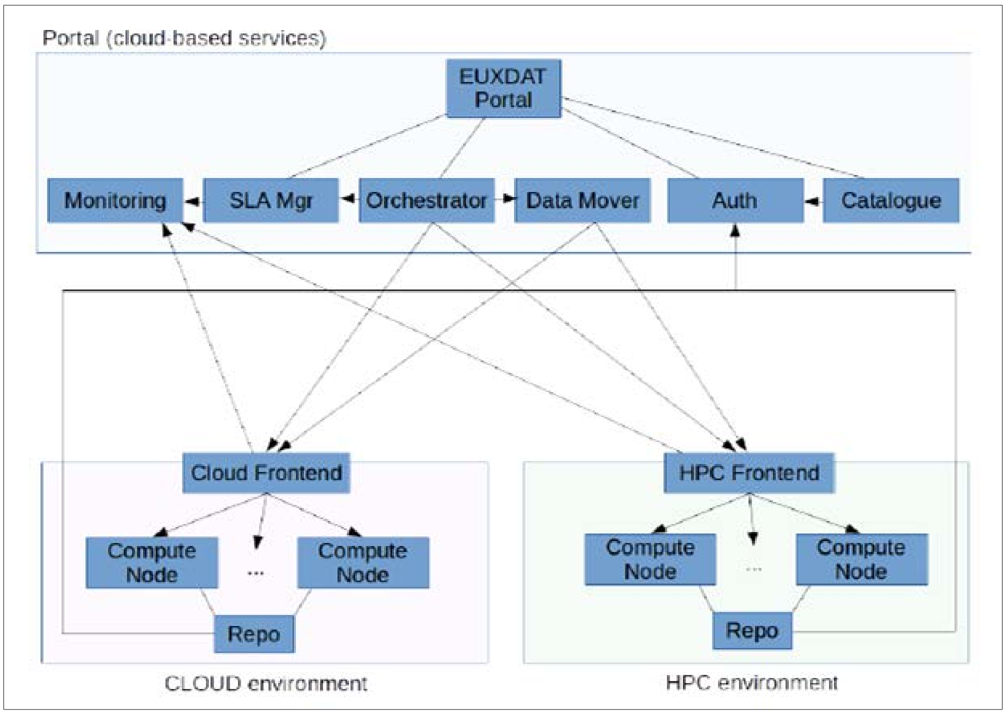


Figure 1 Platform Infrastructure Overview

There are 3 environments coming into play, as first there is the end-user portal where also all central services will be deployed and running. The two other major building blocks of the platform's infrastructure are the computation backends, Cloud and HPC/HPDA systems, connected to and utilized by the help of the central services.

3.1 EUXDAT Platform Services

There has been a previous work in WP2 with respect to EUXDAT requirements and high level architecture definition that already listed certain features expected from the e-Infrastructure (see D2.1 [1] and D2.2 [2]). WP4 has taken those features, identifying which ones depend on the Infrastructure Platform part of the project, which is focused on providing the adequate resources for running the large data analytics tasks EUXDAT is designed for. We have identified five main features that WP4 must design in detail and implement, in order to fulfil the received requirements.

Software Execution Management

The main feature that the EUXDAT Platform must provide is related to the management of the applications that are executed through the e-Infrastructure. EUXDAT assumes that data analytics applications have certain requirements and constraints for running as they are expected to. Therefore, EUXDAT has to guarantee that, once such indications are received, the platform will deal with them, assigning the adequate resources and following up the execution, in order to complete the application execution lifecycle.

We know that applications based on large data analysis require to run several steps, so EUXDAT enables the usage of simple workflows by means of TOSCA, which allows to determine the different parts of the application to run while, at the same time, it is possible to indicate the kind of resources that are necessary and other aspects that should be taken into account. EUXDAT will execute each task with optimal conditions, monitoring the process in case it is necessary to apply any fault tolerance action.

Resources Management

The optimal assignment of resources is very important, especially when we want to guarantee certain quality levels which will make applications usable and valuable for end users. EUXDAT proposes a solution in which HPC and Cloud resources are mixed, as a way to optimize the execution as much as possible, exploiting HPC computing power and Cloud flexibility and scalability.

EUXDAT will generate profiles of the tasks and applications executed, in order to understand how software behaves and makes use of the assigned resources. Then, using those profiles, declared tasks requirements and information about resources available (and their characteristics), EUXDAT will assign the adequate resources to each task.

Data Management

In the same way that computing resources are managed, data storage and movement is also managed. Together with the components implemented in WP3 (as described in D3.1 [3]), the EUXDAT platform will facilitate storage resources and means to move data between computing centres, in such a way that data movement and storage will be as much optimal as possible.

Taking into account that moving large amounts of data is costly and might be problematic (i.e. data chunks that are lost), EUXDAT will provide mechanisms that guarantee that data location and usage is also taken into account when assigning resources and when using storage capacities.

SLA management

Since EUXDAT wants to guarantee a certain level of Quality of Service (QoS) to its end users, it is necessary to set up a mechanism which will define the quality levels and that will deal with this aspect. That means, to take care of Service Level Agreements negotiation and their management (storage, monitoring and application of corrective actions).

The SLA management in EUXDAT will work at application level (instead of task level), based on pre-defined templates and flavours (i.e. gold, silver, etc) which will influence the resources and data management actions, taking into account that any SLA breach has to be managed as stipulated in the negotiation. For doing so, it is important to guarantee that the appropriate monitoring mechanism is available, so QoS aspects can be analysed in real-time.

Monitoring

In order to take the right decisions and to know the current status of the whole system, it is necessary to set up a monitoring mechanism that will be retrieving information about the EUXDAT platform. Such EUXDAT monitoring mechanism will retrieve information from resources providers (HPC and Cloud) and from the applications' behaviour (how they use the resources), storing the information in databases that will facilitate the time series analysis of data.

The monitoring solution will allow the definition of thresholds which will trigger alerts, as a way to facilitate raising issues, like SLA breaches because of one of the QoS aspects.

Finally, the monitoring information will be organized in such a way that it will be possible to show it to different groups of users, since each of them will have different interests and access rights (i.e. administrators vs end users running applications).

3.2 Cloud Backend

One kind of computation backend is the Cloud environment, connected to the portal, from which end-users utilize provided resources transparently, based on their workload policies.

3.2.1 Services provided

EUXDAT cloud infrastructure is based upon Open Telekom Cloud (OTC) which provides an Infrastructure-as-a-Service service on the basis of OpenStack technology.

The infrastructure services of OTC can be configured via a self-service portal. OTC offers the following functions:

• Computing:	Virtual server types with different computing powers
• Storage:	Virtual volume storage and object storage
• Network:	Virtual network services with public and private IP addresses

In addition to the OTC self-service portal, users may also deploy OpenStack APIs while using standard OpenStack command line tools in order to provide new and manage existing resources via web service interfaces. Via REST API calls, this API supports users in the fully-automated provisioning of cloud resources. The APIs are based on standard OpenStack implementation enriched with additional features specifically developed for OTC. The APIs can always be retrieved in the latest version online at https://docs.otc.t-systems.com.

3.2.2 Computing resources

OTC provides automatic provisioning of Elastic Cloud Servers, i.e. virtual computing servers consisting of processor(s) (vCPU), memory (RAM), OS image (operating system, public or private image) and storage resources. The customer can choose between pre-assembled Elastic Cloud Server types. When the customer selects vCPU, RAM, storage, and image, OTC automatically provides it with an Elastic Cloud Server of the adequate type.

Different server types are listed in the following table.

Table 1: OTC Computing Server Types

Category	Elastic Cloud server Type	vCPU	RAM (GB)
General Purpose:	s1.medium	1	4

Category	Elastic Cloud server Type	vCPU	RAM (GB)
vCPU/RAM ratio – 1:4	s1.large	2	8
	s1.xlarge	4	16
	s1.2xlarge	8	32
	s1.4xlarge	16	64
	s1.8xlarge	32	128
Compute 1: vCPU/RAM ratio – 1:1	c1.medium	1	1
	c1.large	2	2
	c1.xlarge	4	4
	c1.2xlarge	8	8
	c1.4xlarge	16	16
	c1.8xlarge	32	32
Compute 2: vCPU/RAM ratio – 1:2	c2.medium	1	2
	c2.large	2	4
	c2.xlarge	4	8
	c2.2xlarge	8	16
	c2.4xlarge	16	32
	c2.8xlarge	32	64
Memory Optimized: vCPU/RAM ratio – 1:8	m1.medium	1	8
	m1.large	2	16
	m1.xlarge	4	32
	m1.2xlarge	8	64
	m1.4xlarge	16	128
Memory Optimized 2: vCPU/RAM ratio – 1:8	m2.8xlarge.8	32	256
High Performance 1: 1 vCPU is equal to one physical core	h1.large	2	4
	h1.xlarge	4	8
	h1.2xlarge	8	16
	h1.4xlarge	16	32
	h1.8xlarge	32	64
	h1.large.4	2	8
	h1.xlarge.4	4	16
	h1.2xlarge.4	8	32
	h1.4xlarge.4	16	64

Category	Elastic Cloud server Type	vCPU	RAM (GB)
	h1.8xlarge.4	32	128
	h1.large.8	2	16
	h1.xlarge.8	4	32
	h1.2xlarge.8	8	64
	h1.4xlarge.8	16	128
	h1.8xlarge.8	32	256
High Performance 2: Optimized with InfiniBand network adapters and local SSD hard disks directly in the physical host for scientific calculations. Based on local hard drives, note that this type of machine is also charged if the Elastic Cloud Server has been shut down.	h2.3xlarge.10	12	128
	h2.3xlarge.20	12	256
Large Memory: Optimized flavor for highly-scaled Enterprise applications of the type "in-memory computing." Volume storage SSD – Latenz optimiert	e1.xlarge	4	128
	e1.2xlarge	8	256
	e1.4xlarge	16	470
	e1.8xlarge	32	940
	e2.2xlarge	8	128
	e2.3xlarge	12	256
	e2.4xlarge	18	445
	e2.9xlarge	36	890
Disk Intensive: Optimized with hard disks directly in the physical host for big data applications and applications with high read/write performance requirements. Based on local hard drives, note that this type of machine is also charged if the Elastic Cloud Server has been shut down.	d1.xlarge local hard disk 3x 1.8 TB SAS	4	32
	d1.2xlarge local hard disk 6x 1.8 TB SAS	8	64
	d1.4xlarge local hard disk 12x 1.8 TB SAS	16	128
	d1.8xlarge local hard disk 24x 1.8 TB SAS	36	256
GPU optimized: Optimized for computationally intensive applications	g1.xlarge	4	8
	g1.2xlarge	8	16

Category	Elastic Cloud server Type	vCPU	RAM (GB)
1 GPU NVIDIA M60-1Q			

3.2.3 Image Management Service

The Image Management Service provides images pre-configured and fixed images in the form of operating systems for use on Elastic Cloud Servers. It also offers the opportunity to use the customer's own images Every Elastic Cloud Server must be assigned an image by the user.

The following table provides the list of the available pre-configured images and the compatibility with the different server types:

Table 2: Public Images Compatibility

OS	General purpose	Compute 1 & 2	Memory Optimized	High Performance 1	High Performance 2	GPU	Large Memory	Disk Intensive
OpenSUSE	X	X	X	X			X	X
CentOS	X	X	X	X	X		X	X
Debian	X	X	X					
Fedora	X	X	X					
Ubuntu	X	X	X					
EulerOS	X	X	X	X				
SUSE Enterprise Linux	X	X	X	X	X		X	X
Oracle Linux	X	X	X					
Red Hat Enterprise Linux	X	X	X	X	X		X	X
Microsoft Windows	X	X	X	X		X		

It is also possible to upload its own so-called private images to the Image Management Service or to create them on the basis of an Elastic Cloud Server.

3.2.4 Storage resources

OTC provides different types of storage solutions:

- **Object Storage**: object-based data storage. The data storage is reached via an Internet link by means of the HTTP and HTTPS protocols.
- **Elastic Volume**: The Elastic Volume Service provides the customer with data storage in block level storage capacities. Customers can use the Elastic Volume Service separately or connect it to storage capacities for use on Elastic Cloud Servers. The following block storage types are available to the customer for selection:
  - Common I/O: SATA disk; IOPS: up to 1000; data throughput rate: up to 40 MB/s; Response time: 10 – 15 ms
  - High I/O: SAS disk; IOPS: up to 3000; data throughput rate: up to 120 MB/s; Response time: 6 – 10 ms
  - Ultra-high I/O: SSD disk; IOPS: up to 20000; data throughput rate: up to 320 MB/s; Response time: 1 – 3 ms
  - Ultra-high I/O (optimized latency): SSD disk; IOPS: up to 20000; optimized data throughput through InfiniBand: up to 400 MB/s; response time: 1 ms – for use with the "large memory" server type.
- **Volume Backup**: a full backup to restore local system and storage data. A backup is a "snapshot copy" of an Elastic Cloud Server or Elastic Volume.

3.2.5 Computation framework

Two kinds of "Big Data" processing can be considered: the "batch" processing and the "streaming" processing.

A batch processing architecture enables dealing with a data set until the exhaustion of the source. As long as input data is available, processing continue until reaching a consistent state. The result is available when all the processing tasks are over. Computation frameworks to implement this kind of architecture are the following (not an exhaustive list):

- Hadoop MapReduce[1]
- Apache Spark[2]
- Apache Flink[3]
- Apache Tez[4]
- Apache Apex[5]

A streaming architecture enables dealing with data in real time, keeping in mind that the notion of "real time" could vary depending on the use case from a few seconds to several minutes. With this kind of processing, when a data come, it is processed in order to produce a result as soon as possible. Quite similar to the streaming, the micro-batch approach also exists, in which data are processed by small and

[1]https://github.com/apache/hadoop-mapreduce
[2]https://spark.apache.org/
[3]https://flink.apache.org/
[4]https://tez.apache.org/
[5]https://apex.apache.org/

periodic batches. Computation frameworks to implement this kind of architecture are the following (not an exhaustive list):

- Apache Storm[6]
- Apache Flink
- Apache Apex
- Apache Spark (with the streaming module)

If a distinction exists between these two kinds of processing tasks, they are not exclusive: most of the Big Data architectures combine them together in the frame of a same infrastructure, as in the Lambda architecture.

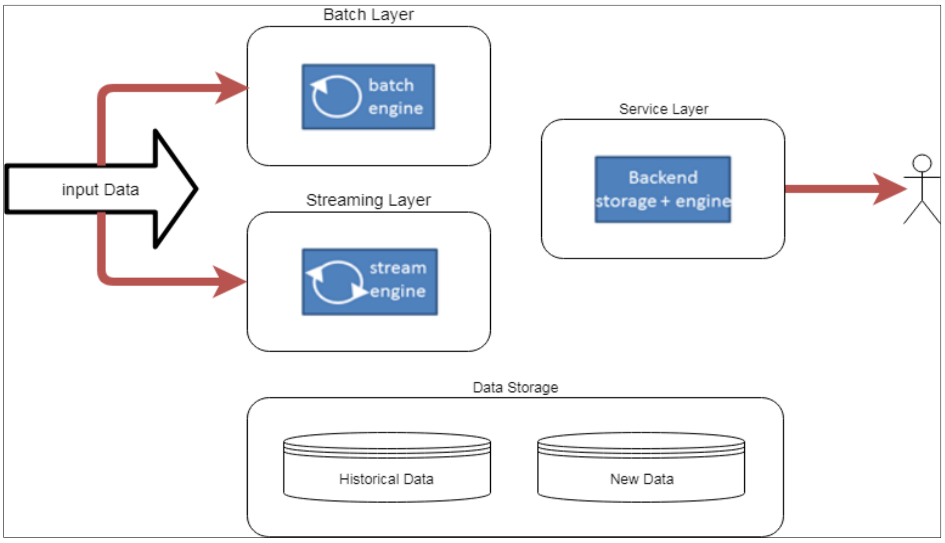


Figure 2 Lambda Architecture

Taking advantage that some frameworks can handle batch and streaming processing (Spark, Apex, Flink...), the Kappa architecture provides a simplification by fusing batch and streaming layers, and thus getting rid of the necessity to implement two processing chains.

This kind of architecture provides the more flexibility and adaptability to the various use cases that are planned to be implemented in EUXDAT. As a partial conclusion, we can state that a framework that can handle both batch and streaming processing should be used in the EUXDAT cloud platform. The considered solutions are the following:

- Apache Apex
- Apache Flink
- Apache Spark

A brief presentation of the considered framework is provided in the next chapters. The final statement requires more evaluation and testing and is not decided in the frame of this document.

[6]http://storm.apache.org/

3.2.5.1 Apache Spark

Apache Spark is a general-purpose and fast cluster computing system. Apache Spark is a tool for running Spark applications, i.e. application developed using the Spark API provided in Java, Scala, Python and R. That can be seen as a downside as the Spark API is quite specific and not very easy to use. Executing already existing algorithms on Spark can require major refactoring of the code.

Apache Spark doesn't implement strictly a streaming architecture but can act in quite a similar way by enabling micro-batch processing using the Spark Streaming module.

Spark uses master/slave architecture, with a central master coordinating many distributed workers. The master runs its own Java process. It communicates with a potentially large number of workers, each one executing a separate Java process. With the help of a cluster manager (i.e Kubernetes, Mesos, Hadoop Yarn…) a Spark application runs on a set of machines.

Apache Spark comes with an ecosystem of modules and libraries. We can mention:

- MLlib implementing machine learning algorithms in a distributed way
- GraphX for graphs and graph-parallel computation

3.2.5.2 Apache Flink

Apache Flink is an open source streaming data flow engine that provides communication, fault-tolerance, and data-distribution for distributed computations over data streams. It acts as a scalable data analytics framework fully compatible to Hadoop.

The main focus of Apache Flink is to reduce the complexity that has been faced with other distributed processing engine such as Spark.

Flink is mainly based on the streaming model: it iterates data by using a native streaming architecture. This pipelined architecture allows processing the streaming data faster with lower latency than a micro-batch architecture like Spark.

3.2.5.3 Apache Apex

Apache Apex is a platform and framework enabling developers to write stream and batch oriented distributed applications. It is designed to process data in a distributed, highly performant and fault tolerant way. It is an in-memory processing engine, highly scalable with very low end to end latency.

The underlying architecture of Apex favours streaming first. It is designed to run continuously. It uses check-pointing and replays to ensure fault tolerance, and no acknowledgement in order to keep a very low latency while still scaling up linearly with resources.

It is a Hadoop native platform that can run on YARN and HDFS.

It provides an easy to use Java API that enables users to write their code with limited knowledge of stream processing notions. Developers can focus on writing user defined functions without having to think about how they will operate in distributed environment.

3.3 High Performance Computing Backend

The second computation backend for the EUXDAT e-infrastructure is an HPC batch system and a BigData capable system, both with different characteristics than Clouds come with. This comprises for example that HPC compute nodes, and applications running on them will not be directly accessible from the outside. Further, applications may not start immediately, depending on SLAs, but instead when the batch system scheduler allocated free resources. Another important aspect is the batch-based workload processing, which does not allow in contrast to clouds to run services or applications continuously, but are limited e.g. to 24h.

3.3.1 Services provided

The HPC environments hosted by HLRS are a traditional High Performance Computing (HPC) batch system and an High Performance Data Analytics (HPDA) system. The HPC system is run with Torque/PBS in combination with the sophisticated scheduler Moab, besides many other features it allows enforcement of SLAs. While the HPDA system is run with Mesos, interfacing transparently with Spark, Hadoop and Cray Graph Engine.

- Computing: Bare metal servers with homogenous characteristics
- Storage: Highly parallel Lustre workspace file-system and a HPSS with tapes

Computation capabilities are requested and utilized by the help of the schedulers, while the storage for user homes is provided by default with a fixed quota, and in addition but required to be requested manually (as part of a job script) there are Lustre workspaces limited in time (30days) and a HPSS long term storage system (tape library).

3.3.2 Computing Resources

There are two HPC systems available for the EUXDAT platform, one is a batch-based while the other one is a stream processing system.

3.3.2.1 Batch Processing System

Batch-Systems automatically allocate requested resources, defined as defaults by admins or by users in their job script and/or at submission time. Depending on free slots or SLAs in place the processing may start immediately or is queued until a set of required resources becomes available.

Table 3: HPC System Properties

Cray Cascade XC40 Supercomputer	
Performance	7.4 Pflops

Cray Cascade Cabinets	41
Number of Compute Nodes	7712 (dual socket)
Compute Processors	15424 Intel Haswell E5-2680v3 2,5 GHz, 12 Cores, 2 HT/Core, in total 185088 cores
Compute Memory on Scalar Processors	DDR4 128GB, in total 964TB
Interconnect	Cray Aries Network
Service Nodes (I/O and Network)	90
External Login Servers	10
Pre- and Post-Processing Servers	3 Cray CS300: each with 4x Intel(R) Xeon(R) CPU E5-4620 v2 @ 2.60GHz (Ivy Bridge), 32 cores, 512 GB DDR3 Memory (PC3-14900R), 7,1TB scratch disk space (4x ~2TB RAID0), NVidia Quadro K6000 (12 GB GDDR5), single job usage
	5 Cray CS300: each with 2x Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz, 16 cores, 256GB DDR3 Memory (PC3-14900R), 3,6TB scratch disk space (2x ~1,8TB), NVidia Quadro K5000 (4 GB GDDR5), single job usage
	3 Supermicro Superserver: each with 4x Intel Xeon X7550 (Nehalem EX OctCore), 2.00GHz (4*8=32 Cores for 32*2=64 HyperThreads) 128GB RAM, 5,5TB scratch disk space (10x ~600GB), NVidia Quadro 6000 (GF100 Fermi) GPU, 14 SM, 448 Cuda Cores, 6 GB GDDR5 RAM (384bit Interface with 144 GB/s), single job usage
	1 Supermicro Superserver: with 8x Intel Xeon X7550 (Nehalem EX OctCore), 2.00GHz (4*8=32 Cores for 32*2=64 HyperThreads) 1TB RAM, 6,6TB scratch disk space (14x ~600GB), NVidia Quadro 6000 (GF100 Fermi) GPU, 14 SM, 448 Cuda Cores, 6 GB GDDR5 RAM (384bit Interface with 144 GB/s), multi job usage
	2 Cray CS300: each with 4x Intel(R) Xeon(R) CPU E5-4620 v2 @ 2.60GHz (Ivy Bridge), 32 cores, 1536 GB DDR3 Memory (PC3-14900R), 15 TB scratch disk space (4x ~4TB RAID0), NVidia Quadro K6000 (12 GB GDDR5), multi job usage
User Storage	Lustre Workspace with ~10PB, 30days default run-down time
	NFS Home with default user Quota of 20GB, and group quota of 200GB
	HPSS with a total capacity of 4PB long term tape storage
Operating System	Cray Linux Environment (includes SUSE Linux SLES11, HSS and SMW software)
	Extreme Scalability Mode (ESM) and Cluster Compatibility Mode (CCM)

Compilers, Libraries & Tools	Cray Compiler Environment, Intel Compiler, PGI Compiler, GNU compiler
	Support for the ISO Fortran standard (2008) including parallel programming using coarrays, C/C++ and UPC, support for Java
	MPI 3.0, Cray SHMEM, other standard MPI libraries using CCM. Cray Apprentice and CrayPAT™ performance tools. Intel Parallel Studio Development Suite (option)
	Cray Debugging Support Tools: Lgdb,STAT,ATP
Batch Job Management	Torque, PBS job management system
	Moab, Adaptive Computing Suite job management system
	SLURM – Simple Linux Unified Resource Manager
External I/O Interface	10GBit

3.3.2.2 HPDA System

HPDA systems serve another paradigm and focus on uses cases concerning BigData, machine learning, deep learning, stream processing, and such.

Table 4: HPDA System Properties

Urika GX	
Number of Compute Nodes	48 (7admin nodes, 41 compute nodes)
Compute Processors	36 cores per node, in total 2304 cores
Compute Memory	512GB per node, in total ~21TB
User storage	Hadoop Distributed File System (HDFS) with 2TB per node, 128TB in total
	Local SSDs with 1.6 TB per node
	Dedicated Lustre Workspace with 240TB
	NFS for user homes
Interconnect	Cray Aries Network
Operating System	CentOS7.0
Compilers, Libraries & Tools	Glibc (2.17), gcc (4.8.5), Java (1.8.0_131), Scala (2.11.8), sbt (0.13.9), Jupyter (4.2.0) and Jupyter Notebook (4.2.3), Python (2.7.5, 3.4.3, Anaconda Python 3.5.2), R-language (3.4.3), ant (1.9.2), git (1.8.3.1), Maven (3.3.9), Grafana (3.0.1), InfluxDB (0.12.2)
	Marathon (1.1.1), Zookeeper (3.4.6), Mesos, Yarn, Flex, Cray Graph Engine (CGE), Cobbler (2.6), collectl (3.7.4)
Resource Management	Apache Mesos as top layer

	Apache Hadoop (via YARN), Spark (via Mesos) or Cray Graph Engine (via SLURM) as second transparent resource management layers

3.3.3 Application Management

Applications management is done by the help of an Environment Modules System[7] which allows to deploy multiple versions of different software tools on a system and enables users for self-service of picking the best matching ones for their needs.

Besides globally installed software and tools, managed by the modules system, users can also deploy any applications, software, libraries and tools within their user space, as long as no root rights are required to properly run.

3.3.4 Storage

There are three different types of storages available, for short/mid/long term.

Short term storage, also referred to as workspace, is a highly parallel Lustre file-system, connected to both machines, the data stream processing and batch system, and should be utilized for any I/O intensive workloads and intermediate data. Such a workspaces need to be actively allocated by the user and those have a default life time of 30days.

Mid term storage is a NFS for the user $HOME filesystem, reachable from both machines, like workspaces. There is a default quota in place, which might be further extended if reasonable.

Long term storage in an High Performance Storage System (HPSS) designed to manage petabytes of data stored on disks and in tape libraries. HLRS manages 500 TB of disk storage and more than 4PB tape storage held in two copies.

3.3.5 Data Transfer

Data transfers from and to HLRS can be done via SCP directly into the user's $HOME for smaller datasets, such as source code or smaller input files that in summary have a up to a few hundred MBs at maximum.

A more efficient approach for huge data volumes is a direct upload/download from/to user's fast Lustre-based workspace, via two GridFTP endpoints, each capable of 10GBits. GridFTP is a high performance FTP service, a component of the Globus middleware suite[8].

3.3.6 Computation Frameworks

The HPDA system Urika-GX provides several engines for data and stream processing.

[7] http://modules.sourceforge.net/
[8] http://www.globus.org/toolkit

- Spark
  - Spark Core, DataFrames, and Resilient Distributed Datasets (RDDs) - Spark Core provides distributed task dispatching, scheduling, and basic I/O functionalities.
  - Spark SQL, DataSets, and DataFrames - The Spark SQL component is a layer on top of Spark Core for processing structured data.
  - Spark Streaming - The Spark Streaming component leverages Spark Core's fast scheduling capabilities to perform streaming analytics.
  - MLlib Machine Learning Library - MLlib is a distributed machine learning framework on top of Spark.
  - GraphX - GraphX is a distributed graph processing framework on top of Spark. It provides an API for expressing graph computations.
- Hadoop
  - HUE[9]
  - HDFS10
  - Hortonworks Data Platform (HDP)11
  - Hive[12]
- Cray Graph Engine

4.3.6 Connectivity

The HPC and HPDA systems can be accessed through SSH preferably with keys rather than passwords, only. Resources are only accessible under an user's account, so either application providers have to execute workload under their credentials on behalf of the EUXDAT platform users, basically resell resources and take care of accounting, or each user has to signup first at HLRS as HPC user.

3.3.7 Application Deployment

Binaries can only be installed by administrators after a security audit, which is a longer lasting and effort intensive process. Thus, the preferred way to run EUXDAT applications is in user space, without the need of root rights. Application developers can provide compiled binaries for both target systems, the HPC and HPDA one, in the central platform repository from where it is pulled on request and after authorisation checks automatically and transparent to the platform end-user.

[9]https://github.com/cloudera/hue
[10]https://hortonworks.com/apache/hdfs/
[11]https://hortonworks.com/products/data-platforms/hdp/
[12]https://hive.apache.org/

4 Infrastructure Features

In the following subsections all major features of the infrastructure platform are described. Features are mapped on dedicated individual components, described in depth in the succeeding subsections.

4.1 Orchestrator

The brains of the EUXDAT system, is the Orchestrator. The name says it all, since this component is responsible for coordinating the whole system, by making sure that all the other components (the orchestra, you could say) work together to produce the final output of the system. Specifically, the Orchestrator will decide which resources to use, after a User[13] request, and will then act upon this decision by distributing the data and application to the selected resources. This decision will be based on the following information:

- The User's profile, permissions and agreement with the EUXDAT platform.
- The requested application's profile.
- The service level agreement (SLA).
- The current status of available HPC and Cloud resources.

In Figure 3, a high level architectural diagram of the Orchestrator is presented. The Internal sub-components of the Orchestrator component are depicted in orange boxes, whereas the other coloured boxes are EUXDAT components that are external to the Orchestrator. The Orchestrator Interface receives user requests via the EUXDAT Portal.

Initially, the Orchestrator needs to negotiate with the SLA Manager with regards to the requested by the User service level for the specific application and job, and once there's an agreement, the Interface sends all the relevant information to the Engine. This is where the decision will be made which HPC or Cloud resource to use for the specific job.

The Engine obtains the User's credentials from the I&A Manager and information pertaining to the data needed for the job is retrieved from the Data Manager. The latter information will include size and type of data, as well as possibly a profile of the specific dataset. Finally, to support the Engine's decision making process, the status of the available resources is retrieved from the Monitoring component via the Monitoring Connector. Finally, the Profiles Manager in the diagram refers to application profiles and not to User profiles, (which, as mentioned above, are managed by the I&A Manager), and is also taken into account to decide what the optimal resource to use given the requested application, should be. Note that the Profiles Manager can also query the Monitoring component (via its Connector) to determine historical data (in the form of application logs, curated or raw) about the specific application.

[13] The term User has a broad sense here, since it does not need to be a physical user making this request, but could be a batch job, a service request, or any such scheduled task, etc.

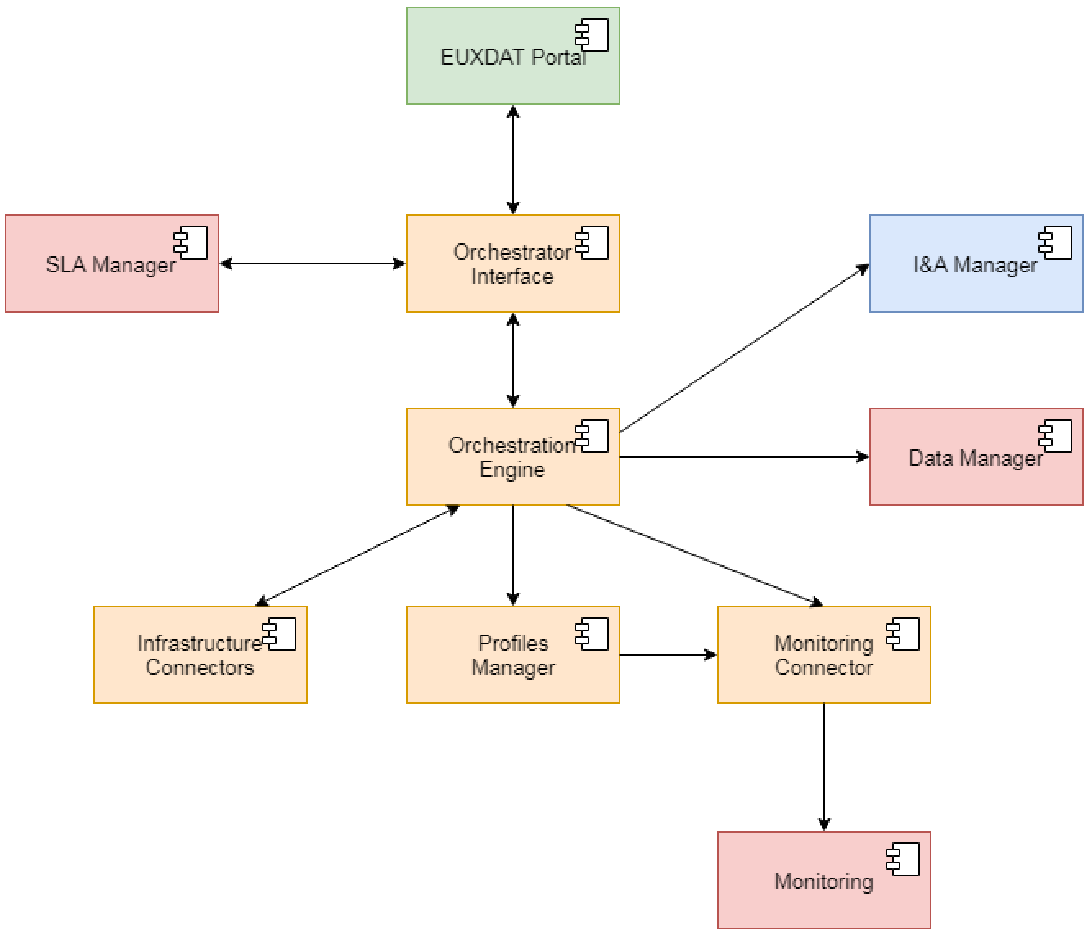


Figure 3: Orchestrator High Level Architecture

The Orchestrator is not only responsible for the decision, but also for managing running jobs. The Infrastructure Connectors add a level of abstraction for the physical HPC resources, by tying each physical cluster with one Infrastructure Connector, and similarly for each Cloud resource. The Engine is then also responsible for transferring the data, retrieved from storage via the Data Manager, through the Infrastructure Connector, to the chosen resource. Finally, the Orchestrator is in constant communication with the Monitoring component, during the execution of the application. More details about this monitoring is presented in the following section.

Orchestration solutions

The project MSO4SC required a similar Orchestrator, to decide given various sources of information, what the optimal resource for execution of mathematical modelling software, would be, given the choice of HPC and Cloud resources. Thus, an evaluation of orchestrator-like software, and frameworks that could be modified and used as a basis for a novel implementation of such an orchestrator, was carried out. This state of the art analysis was published in the MSO4SC D3.1 deliverable [4].

Solutions discussed there included Apache Mesos in conjunction with the Mesos Frameworks: Marathon, Chronos and Aurora. These Mesos-based solutions, you could say, provide a sort of supervised orchestration. A couple of container technologies that come with a form of orchestration for containers looked at were Kubernetes and Docker Swarm. The automation engines: Ansible, Chef, Fabric, Puppet, and Salt were also considered, as building blocks for MSO4SC's orchestration solution.

HPC specific meta-schedulers (on top of Slurm or TORQUE schedulers for example) such as DIRAC, Maui and Moab, have a similar philosophy to our envisioned Orchestrator, i.e., they introduce a layer of abstraction between the user and the physical machines/clusters, but they are very tightly coupled to specific HPC technologies and making them flexible enough to satisfy our needs in EUXDAT would be a daunting task.

There are also Cloud based solutions, that facilitate the definition and execution of workflows on most popular infrastructures and tools, such as Ansible, Chef, OpenNebula, and OpenStack. One such platform is Cloudify, which was eventually chosen as the best fit for adjustment for the needs of MSO4SC's Orchestrator. Workflows are defined for Cloudify by using the TOSCA description language, which is one of the main reasons why Cloudify was chosen over another modern framework looked at, Apache Brooklyn, which uses its own custom language. TOSCA is more extensive and is an approved standard, which makes it more appetizing for our needs in EUXDAT as well. Finally, another possible solution looked at was Heat, which can do similar orchestration tasks as Cloudify, but is also based on a custom description language (heat), and is tied to OpenStack and CloudFormation technologies.

Apache Airavata, a software framework to define and manage workflows on various resources, is an ambitious idea which may in the future be able to fulfil requirements for projects such as EUXDAT. It was however judged to not be mature enough for the needs of MSO4SC, and was discarded on this basis.

The solution proposed and implemented in that project took a novel approach and is based on Cloudify with TOSCA. Cloudify already allows the use of Cloud (VMs) resources, so a plugin was implemented which allows HPC resources to be defined and used in executions as well. In EUXDAT we plan to extend and improve this solution to allow for more intelligent decision making on the Orchestrator side, based on better and wider information gathered by the Monitoring component, discussed next

4.2 Monitoring

The Monitoring component provides information regarding all assets in the system: HPC and Cloud resources, algorithms and applications, the datasets, and the SLAs of running jobs. The information transmitted to other components could be upon request, periodically, or even in the form of alerts when certain conditions are met.

Specifically:

- The Orchestrator needs to know the state of its infrastructure resources (HPC hardware and proxies, and Cloud VMs) to decide where to execute a job requested by a user. In some cases, it may need to communicate this back to the user, e.g., if there is currently no availability for the job the user is attempting to execute. In general, though, this information should be transparent to the user, i.e., the Orchestrator with information from the Monitoring component, will decide where to execute the user's requested job(s) and act accordingly.

- The Orchestrator needs to periodically be updated with regards to executing jobs, including when a certain job has terminated.
- The Orchestrator needs to be alerted in exceptional situations with regards to executing jobs, and/or resource failures or unusual circumstances, especially when they are critical.
- The SLA Manager similarly needs to be alerted by the Monitoring component if there is a breach in contract for a specific SLA being used for a specific job.

The Portal also needs monitoring information. Depending on the type of User accessing the system, monitoring information will be displayed in a user-friendly way. Typically, this can be implemented using Grafana, which provides a nice visual representation of the monitoring metrics being gathered by the Monitoring component.

Figure 4 was presented in EUXDAT D3.1 [3] and displays the high-level architecture of the monitoring module that will be implemented in EUXDAT. The internal sub-components of Monitoring are depicted in orange boxes, whereas the external to Monitoring components in red. As can be seen in the diagram, the Monitoring component has an Interface to communicate with the outside world. This module is responsible for responding to user queries through the Portal and for providing the Orchestrator and the SLA Manager with the necessary metrics and alerts. Any visualization software, such as Grafana being considered for this project, will also live in this module. The Collector is responsible for gathering the information from the various sources and efficiently preparing it for storage. The Storage sub-component receives the data from the Collector, and persists this data. The Connectors are responsible for providing the communication mechanism between the implementation specific exporters, and Monitoring.

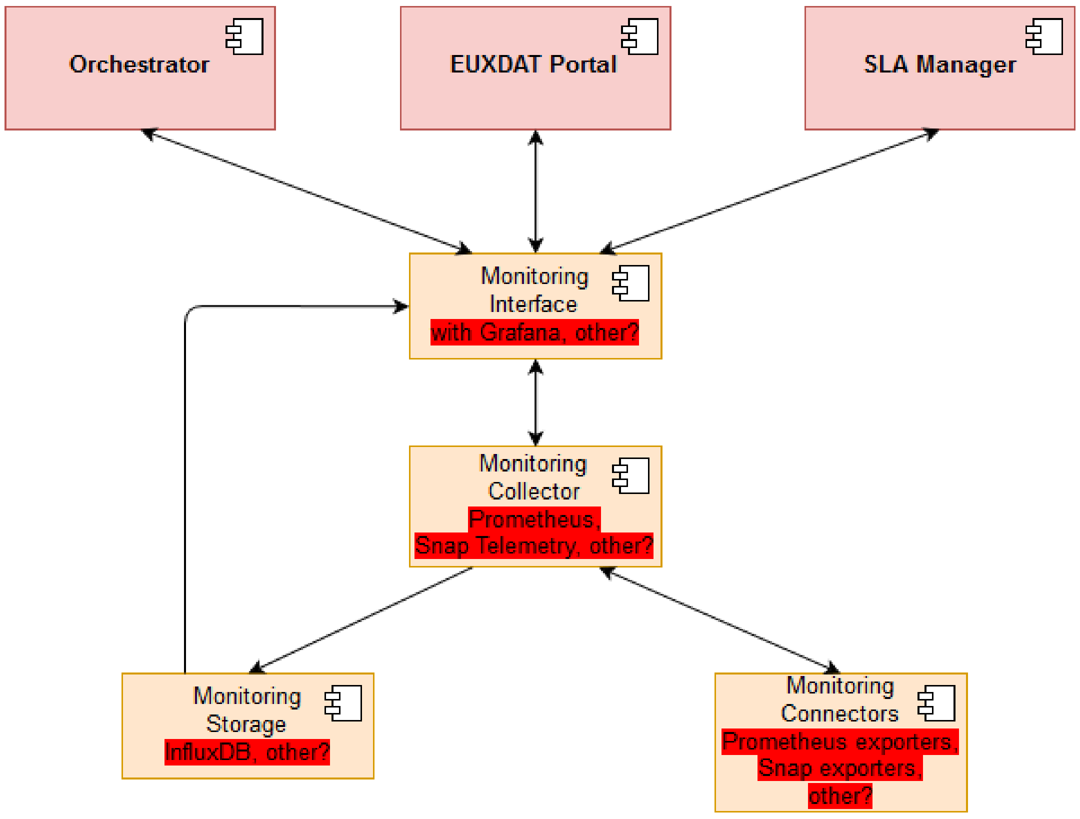


Figure 4: Monitoring High Level Architecture

Monitoring solutions

An extensive, yet far from complete, review of monitoring frameworks and software was done in [4] for the MSO4SC[14] project, and a slightly older review in [5]. Monitoring applications and software evaluated there include Nagios, Icinga, Sensu, Shinken, Zabbix, Prometheus, Zenoss, Ganglia, SeaLion, MonALISA, OpenStack Telemetry/Ceilometer, Monasca, Gnocchi, vSphere, and Amazon CloudWatch.

Some of those monitoring applications include more functionalities than others, straight out of the box (Prometheus), whereas others have a default implementation that is rather lightweight (Nagios, Icinga, Sensu, Shinken) but that can be extended through the inclusion of plugins. Some are specialized for use on HPC resources (Ganglia), while others for the Cloud (e.g. Amazon CloudWatch, OpenStack Telemetry/Ceilometer, Gnocchi).

Visualization software that can take the raw data published from the metrics gathering and monitoring software mentioned above, and present it in a user-friendly, configurable environment, was also evaluated in [4], namely Graphite and Grafana, and a dedicated Graphite publisher called Diamond was also looked at.

In MSO4SC, Prometheus was chosen as the monitoring software and Grafana for visualization, whereas the storage is provided by InfluxDB. Prometheus works with exporters, i.e. software that lives on a

[14] http://mso4sc.eu/

specific machine and feeds back the metrics in a format that Prometheus understands. For example, a cluster running Slurm as its scheduler, can have a slurm-exporter that gathers information from the scheduler, converts it to Prometheus readable format, and forwards the information onto the Monitoring Collector. Prometheus Alert Manager allows you to define alerts, e.g. when an SLA breach has occurred.

Snap is a telemetry framework that can collect system data from a variety of sources (VMs, servers, applications and Oss), includes some processing capabilities such as data filtering and injection and can publish data to a variety of databases, flat files, and scheduling software. Snap integrates well with Grafana, which provides a rich user-friendly visualization environment. The Snap developers consider its open architecture with plugins to be its strongest selling point, combined of course with a growing number of plugin developers and releases.

We have decided to postpone getting tied into one of the above technologies until later, giving us time to evaluate in certain depth the above proposed solutions. This will also allow us to have a clearer idea about how such a system would fit in within the bigger EUXDAT picture, and see how these proposed solutions handle our specific needs. This is reflected in the diagram in Figure 5 in text with red background.

4.3 SLA Manager

The SLA Manager is a component which deals with the QoS expected when running applications which implement large data analytics. It has to deal with the QoS aspects negotiation, the agreements storage, the monitorization of the agreement and the management of corrective actions and penalties.

According to the SLA management solutions analysed in [5], SLA management processes have several phases, namely, negotiation, provisioning, monitoring, assessment and termination. Additionally, in some cases, we may find re-negotiation as a way to solve issues and adapt dynamically the service provision, depending on predictions and context changes.

In Figure 5 we show the planned internal architecture of the SLA Manager for EUXDAT, together with the external to the SLA Manager components: Orchestrator and Monitoring. The Internal components are drawn with the orange background boxes, and the external with red. In TNOVA D4.42 [6] the Service Level Agreement Management (SLAM) life cycle for a service provided by a system is made up of 5 states:

1. Negotiation of the contract
2. Resource provisioning
3. Service delivery monitoring
4. Service performance assessment
5. Service termination

Augmented by a six state:

6. Renegotiation of the contract, which can occur after state 4, i.e., before termination, and once the renegotiation is completed, take the life cycle back to state 2.

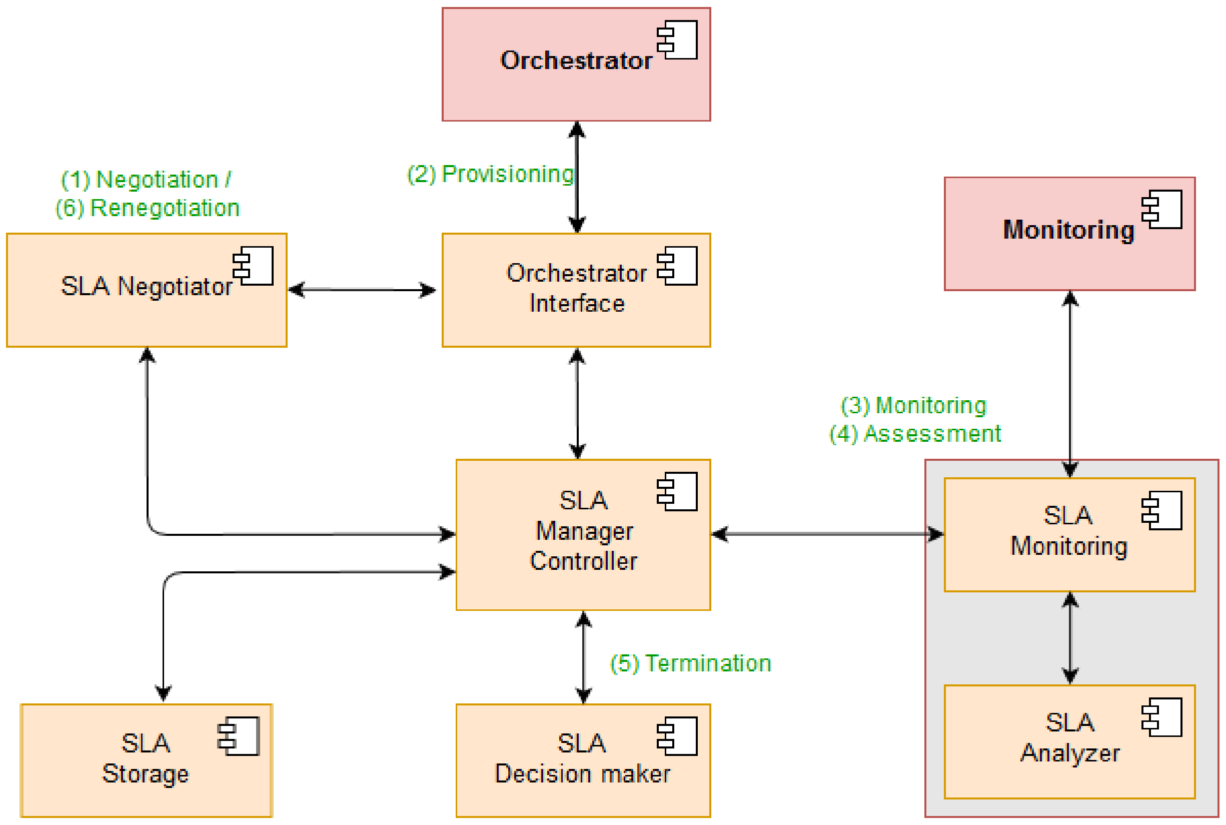


Figure 5: SLA Manager High Level Architecture

The central sub-component of the SLA Manager is a controller in charge of program flow and ensuring the SLAs life cycle states are transitioned accordingly.

Negotiation and potentially renegotiation is performed in the SLA Negotiator. There are several SLA negotiation strategies, with broker-based negotiation and runtime SLA negotiation among the more popular. In the former, the broker will choose suitable service providers and do the negotiation. With runtime negotiation, there is an ongoing negotiation process while the service is running which benefits from the various changes occurring to the environment, to get the best deal. Many SLA managers provide a catalogue to the customer, who can choose their preferred service. This is called offering and the available options are known as offers. The SLA Negotiator may include an offers manager with an offers collector.

The next state of the SLAM life cycle, provisioning, refers to providing and initiating resources, such as an HPC cluster, or a Cloud VM, and will be implemented by the Orchestrator in EUXDAT. The SLA Manager will thus need a way of communicating information back and forth with the Orchestrator, hence the Orchestrator Interface module in the diagram. The SLA Manager will provide the Orchestrator with negotiation info via this interface, as well as data pertaining to any SLA breaches detected.

The SLA Monitoring module is a form of connector to the Monitoring component. However, whereas the Monitoring component will be collecting metrics from many measurable aspects of the various resources, the SLA Monitoring is only interested in the QoS aspects pertaining to the SLA.

The SLA Analyzer takes monitoring information and detects SLA violations, breaches, and predictions of them occurring. The SLA Monitoring and SLA Analyzer modules are often combined into one component on certain systems, hence the grouping in the diagram. These components together oversee states 3 and 4, service delivery monitoring and service performance assessment.

The SLA Decision Maker does as its name says, i.e., it takes decisions related to launching renegotiation (and under what terms), applying penalties, notifications, or stopping the execution. This module can thus decide to terminate a given SLA, or put it back into its negotiating state, as initially, only under different terms and conditions.

Finally, the SLA Storage module stores the information pertaining to the SLA agreed terms, detected violations during its life time, etc., as well as any other information which may support the SLA Management in the future.

4.3.1 SLA Manager Solutions

There are many solutions for SLA management, in fact, SLA negotiation, monitoring, etc. as you would expect are performed by most software and infrastructures that provide a service, for example AWS and GCP. Some of these systems have open source versions, that would allow us to isolate the SLA related parts, and use what we need for our purposes.

Many of the open source solutions we looked at briefly, are IT services ticketing-like systems, with decoupled SLA management modules, for example OpenSmart [7]specializes in monitoring and reporting for IT services. Combodo [8] has a (open source) community edition and one of its major features is a CMDB (configuration management database) with SLA management capabilities, whereas Citsmart [9] also has an open source community version focusing on ITSM (information technology service management), i.e., IT change and incident management, which includes an SLA Manager. Finally, Project-open [10]is an open source project management suite, with an IT service management component which contains an SLA manager package [11]The more interesting solutions we've looked at are Mosaic [12] OTRS [13], and the ATOS SLA Manager and are discussed next.

mOSAIC

This EU project (Open Source ApI and platform for multiple Clouds) implements a multi-agent brokering mechanism that searches for a service that matches the requested one, and if it doesn't find a good match, it can compose one. It has built into it the ability to facilitate competition between various Cloud providers, which benefits them because they can thus reach more customers that could have possibly not even been aware of their existence. The main drawback with mOSAIC is that with the termination of the project in 2013 there doesn't seem to have been any further development or active support provided. This would probably constitute it not fit for our purposes.

OTRS

This software also started out as an IT ticketing system (in 2001), hence the name OTRS (Open Ticket Request System). Nowadays the application has grown into a full blown business service management system, but an open source version is still actively maintained. OTRS implements an offering negotiation strategy, and has a powerful monitoring and assessment component.

ATOS SLA Manager

This solution is the only one we could confirm is WS-Agreement [14] compliant. The other evaluated applications, may be WS-Agreement compliant, but it is rather difficult to verify this without extensive exposure to the code, since OTRS does not have an online search facility, and Mosaic, as we mentioned, appears rather obsolete.

Figure 5 depicts an overview of the ATOS SLA Manager architecture. The correspondence (or not) between those components in Figure 6 are depicted in Table 5.

Table 5: SLA Manager Correspondence

SLA Manager	
ATOS SLA Manager architecture	EUXDAT proposed SLA Manager Internal Architecture (Fig. 6)
Smart Monitoring & Monitoring	Monitoring
Evaluation	SLA Monitoring & SLA Analyzer
Assessment	SLA Decision Maker
Repository	SLA Storage
Factory + (negotiate)	SLA Negotiator: the ATOS SLA Manager implements a one-shot negotiation, whereas in EUXDAT the negotiation policy is still to be decided.
Accounting	(will probably live in) Monitoring
n/a	Orchestrator & Orchestrator Interface
Not depicted as such	SLA Manager Controller

We've decided to adopt this solution, the ATOS SLA Manager, because of its WS-Agreement compliance, but even more importantly because we are aware that it is under active development and we can get support if it is required. The development team in charge of the ATOS SLA Manager have also released a light-weight SLA manager for use in Edge-computing, which is implemented in Go, is based on the WS-Agreement but is not compliant (it is a JSON implementation instead of XML), and is, appropriately, named SLA Lite.

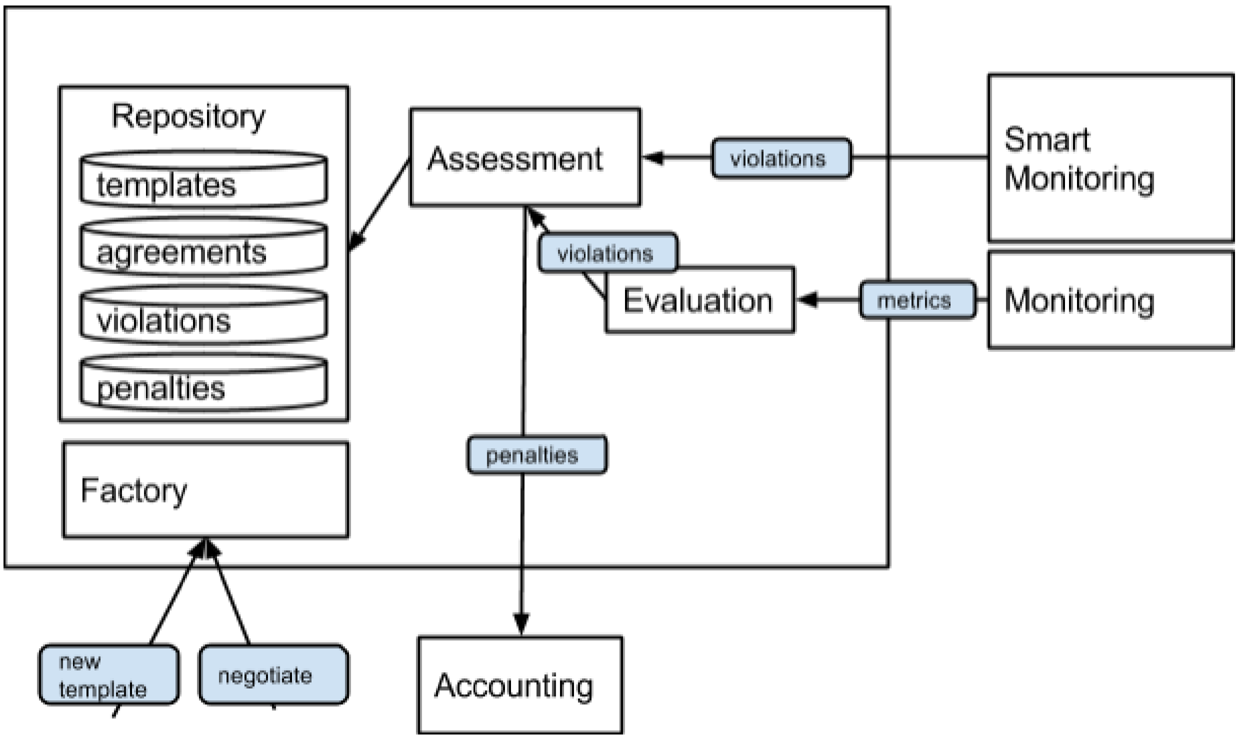


Figure 6: ATOS SLA Manager Architecture

5 Development and Deployment

This chapter highlights the development and deployment services and infrastructure, as well as the intended workflow during development. Further, used software is described in detail, besides the different stages which are ensuring stable production during development.

5.1 Development Concept

The development in this project will be done by a number of developers in different geographical locations. It is thus essential that we set up and configure the most appropriate tools for remote work, as well as ensuring a high quality of development by defining system-wide guidelines. Coding and style guidelines will of course depend on the specific programming languages and technologies used, which will differ in the various EUXDAT components, so it is recommended that these remain as consistent as possible throughout the project, but are detailed and agreed upon for each component before implementations start in earnest, on each component.

We will implement a Continuous Integration (CI) strategy, which aims to solve small problems more often, to avoid larger problems later. The technologies involved in the development, Git, GitLab, Gerrit, Jenkins, and SonarQube, are explained in the next sections.

5.1.1 Repositories

Code is managed by the help of git repositories, where each of the stages has a dedicated code branch, such as *development, integration* and *master*. Confidential information like user name, port, host and such are hidden to the rest of the world by the help of environment variables for GIT.

Figure 7 explains the different repositories in place, and how the workflow for upstreaming code is designed.

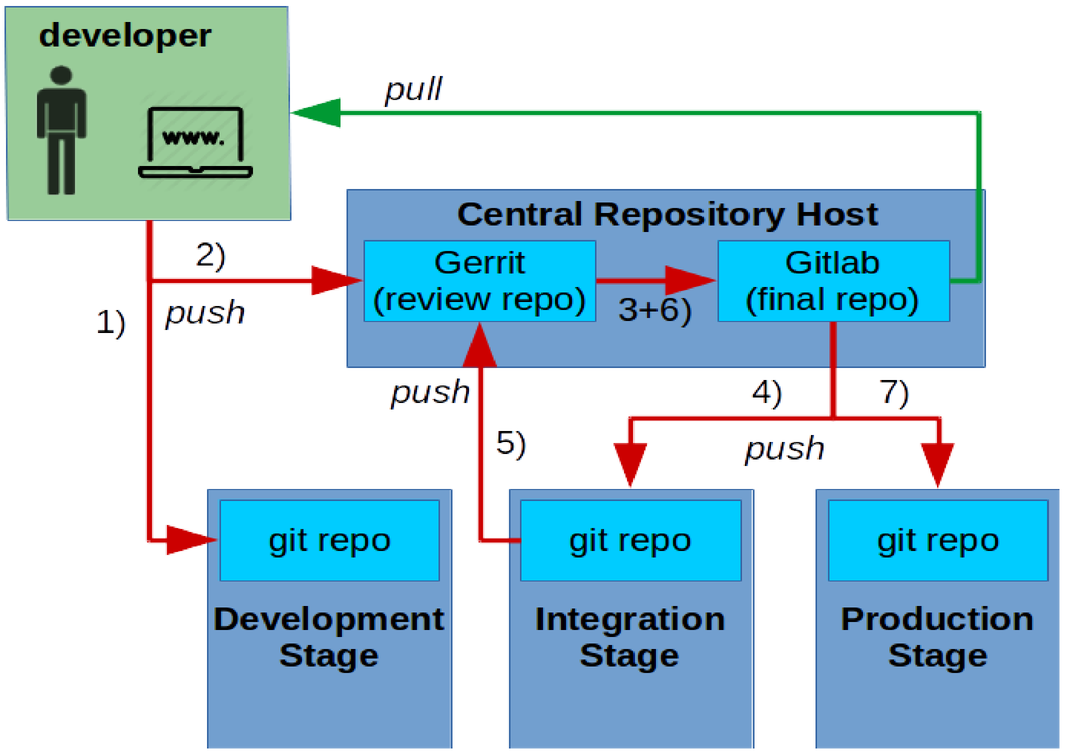


Figure 7: Git Repositories and Upstreaming Workflow

There are 3 different repositories, one for each stage and with dedicated branches, besides the central one managed by git+gerrit+jenkins and cloned local ones on developer machines.

Developers push code as first step (1) their code onto the development stage, where initial implementation is tested and debugged. As soon as a component is working, it will be submitted for review (2) and when passed it will be automatically merged into integration stage's code branch (3) and auto-deployed (4) on the integration stage for further testing and quality assurance. As next step code is pushed (into final review and on approval then merged into master

A submission to the development stage's branch triggers an automatic re-deployment on the development environment, same for commits merged into integration stage's branch after passing a code review. Code on production is only deployed manually after regression tests and final QA has been passed successfully.

5.1.2 Continuous Integration

Here we explain the development workflow for EUXDAT from the moment a developer wants to commit their changes (or to use Git terminology: a developer makes a Merge Request), i.e., without going into the specifics of Change Requests, Issues, or Branch management in Git.

An overview is depicted in Figure 8. When a developer working on an issue is satisfied with the development and their tests have been passed, he/she makes a merge request in GitLab. This will trigger Gerrit to establish a code review for the specified code. When the code review has passed, Jenkins is triggered, and when Jenkins is happy with its own (system-wide) tests, the new code is deployed to the specified environment(s). The details are as follows.

Gerrit & GitLab

Gerrit is a code collaboration tool for Git, that will be integrated into the GitLab setup for the project, which allows remotely located developers to perform code reviews online. It is a Java-based tool and is a fork of Rietveld, Guido von Rossum's original Python-based code review tool for Subversion.

When a developer does a Git push, the output from the Git command will include a Gerrit host link to a web page which will have a similar look to that in Figure 9.[15]

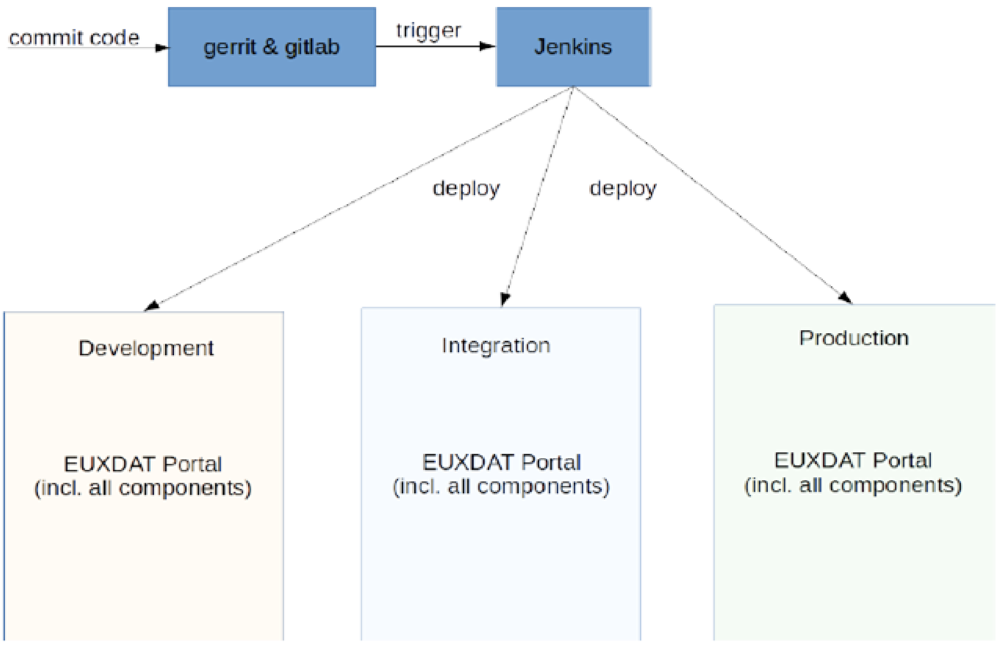


Figure 8: Continous Integration Workflow

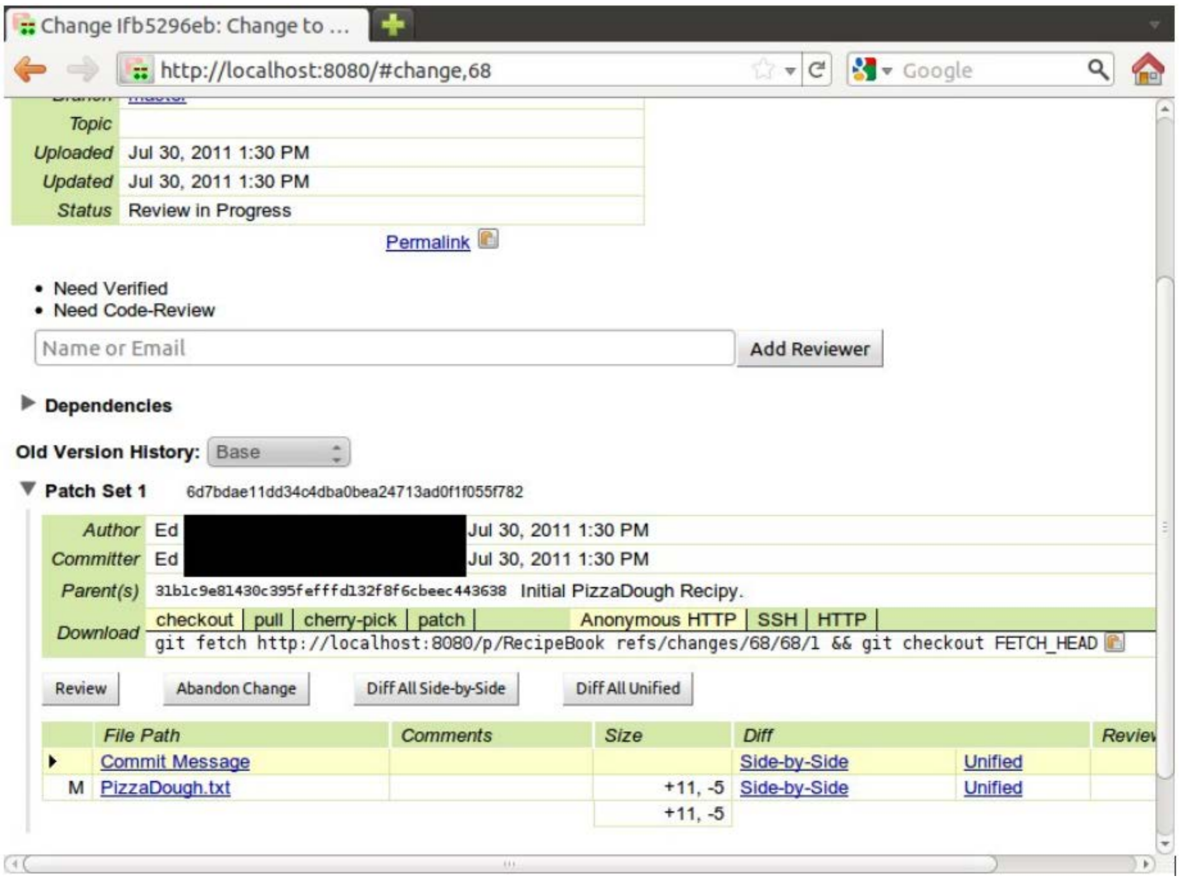
On this Gerrit hosted page, the developer doing the Git push, can add other developers to perform the code review. Gerrit can be configured to automatically assign reviewers, and or simple observers, if someone wants to know how a certain issue is progressing without actually doing a review, and includes

[15] Figure 9 screenshot taken from https://gerritcodereview-test.gsrc.io/intro-gerrit-walkthrough.html

email notifications within its options. It is an interactive page where the developers can exchange comments and make improvements to the code directly on the page.

In the screenshot, there are two bullet points "Need Verified" and "Need Code-Review". The Code-Review need means that another developer must look at the code and make sure it is per the project's (and components) style and guidelines. On the other hand, the Verified need means that the code compiles, executes and passes testing and performance criteria. The latter is usually automated, using for example the Gerrit Trigger Jenkins Plugin.

The developer performing the review can add comments, and eventually give a score to the changed code. The scoring is configurable, but by default is a range from -2 (Do not submit) to +2 (Looks good to me, approved), and includes the option of suggesting that someone else also approves the change, +1 (Looks good to me, but someone else must approve).


Change Ifb5296eb: Change to ...

http://localhost:8080/#change,68

Topic	
Uploaded	Jul 30, 2011 1:30 PM
Updated	Jul 30, 2011 1:30 PM
Status	Review in Progress

Permalink

- Need Verified
- Need Code-Review

Name or Email | Add Reviewer

Dependencies

Old Version History: Base

Patch Set 1 6d7bdae11dd34c4dba0bea24713ad0f1f055f782

Author	Ed [redacted]	Jul 30, 2011 1:30 PM
Committer	Ed	Jul 30, 2011 1:30 PM
Parent(s)	31b1c9e81430c395fefffd132f8f6cbeec443638	Initial PizzaDough Recipy.
Download	checkout | pull | cherry-pick | patch | Anonymous HTTP | SSH | HTTP	

```
git fetch http://localhost:8080/p/RecipeBook refs/changes/68/68/1 && git checkout FETCH_HEAD
```

Review | Abandon Change | Diff All Side-by-Side | Diff All Unified

	File Path	Comments	Size	Diff		Review
▸	Commit Message			Side-by-Side	Unified	
M	PizzaDough.txt		+11, -5	Side-by-Side	Unified	
			+11, -5			

Figure 9: Sample Gerrit Host Page

The Verified part is a simple binary yes or no (or -1 and +1 by default) and is usually automated as mentioned, but could be done manually by another developer if so required.

Jenkins

Jenkins is the foremost used CI tool and is open source. In the case of our planned development workflow, Jenkins is triggered when the code has successfully passed the code review and tests in Gerrit. Jenkins takes over and performs its own (configurable) tests. These are more system wide and regression tests than the unit tests performed during the code review process and will ensure that the modified code hasn't broken other parts of the system, be that its compilation, execution, or system-wide guidelines or other criteria. Adhering to quality assurance guidelines can also be automated, with tools such as SonarQube, Cast, or Squale.

When Jenkins is satisfied that the system code passes its criteria, and regression tests are successfully passes, it is deployed on the corresponding stage (Development, Integration, Production). Note that there will be separate repositories with dedicated branches for each stage, with code being introduced in the order depicted from left to right in the figure.

Jenkins can also be configured to allow periodic (many projects do it nightly) builds and deployments, and can also be executed on demand. It produces log files with each execution, which can be looked at when things go wrong, and can also be used for various development metrics, and analytics.

5.2 Deployment Infrastructure

To ensure a smooth production and proper quality assurance of developments, there are three environments planned. One for component development and testing, one for system and integration tests and a dedicated production environment. All three stages use the same computation backends.

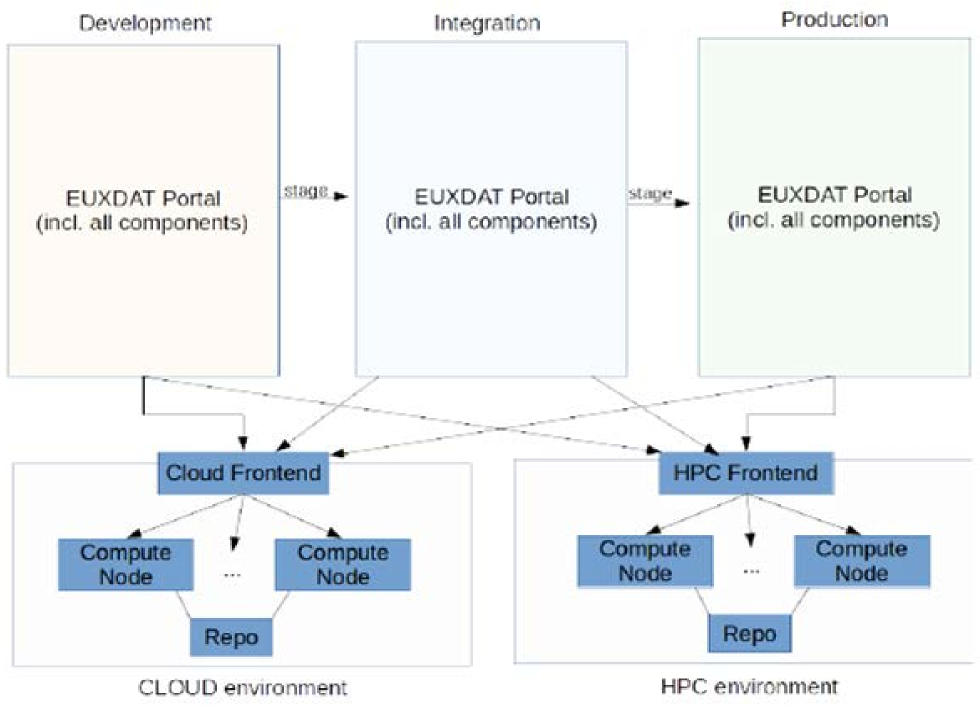


Figure 10: EUXDAT e-Infrastructure Overview

The stages are clones of each other to ensure a smooth migration to the next stage. The production stage is here the reference for the development and integration stages. However, the integration and development stages may be virtualized completely and will have weaker hardware resource demands than the production platform.

5.2.1 Development Infrastructure

The development stage is intended for initial component developments in an environment with the same setup like the production platform, in terms of base components and backends. Developers implement their components here and run first ad-hoc tests.

All content on the development platform is synthetic and should cover edge cases, for example very short and long person, service or product names.

After implementation is done and targeted functionality has been verified, ad-hoc tested and is basically working, a component is ready for migration to the integration stage.

5.2.2 Integration Infrastructure

The integration stage is where initial working components are further tested in combination with other components in their latest or in-development versions. Also regression testing is carried out here, and must be passed successfully before a component can be considered stable.

Content is synchronized with production platform automatically once a day or at least once a week, in terms of all new and updated content is copied over from production environment.

The moment a component has passed integration and system testing, it is ready for a final review and on approval a manually triggered deployment on the production environment can be carried out in order to make the new or updated component available to end-users and put it in production.

5.2.3 Production Infrastructure

Production stage is where the end-users of the platform are to be found. It will have more resources compared to development and integration stages used internally.

All stable components that have passed regression testing and quality assurance, are deployed here and in production. Content on this stage is real world content and will be mirrored to the integration stage.

The component versions, except during development of a component, is predetermined by the stable version deployed on the production environment.

6 Conclusions

This deliverable describes the initial architecture and design of the EUXDAT e-infrastructure platform based on identified requirements summarized in "D2.1 Description of Proposed Pilots and Requirements [1]" and also takes into account "D2.2 EUXDAT e-Infrastructure Definition v1 [2]".

It is comprising the several components of the platform, connectivity to backends and available software in the computation environments. As this document is written at the begin of the project, the platform's e-infrastructure may evolve further in the future and adaption of the presented design may occur.

It starts with an in-depth explanation of the platform specification, the central platform services and connected computation backend systems, namely Cloud, HPC, and HPDA including their hardware properties and storage, data transfer and networking, software environment and available computation frameworks. Followed by a detailed description of the e-infrastructures features and main components, such as workload orchestrator, monitor and SLA manager.

In the succeeding part the development environment setup is highlighted. It points out how a stable production is ensured while development actively takes place, and further, the structure of repositories, branches and distinct stages, as well as the deployment is intended in combination with continuous integration and quality assurance.

This document provides the basis for the implementation and setup of the infrastructure prototype, due in M12 of the project, and accompanied by "D4.2 Infrastructure Platform v1".

7 References

[1] EUXDAT, D2.1 Description of Proposed Pilots and Requirements, Karel Jedlička et al., 2018

[2] EUXDAT, D2.2 EUXDAT e-Infrastructure Definition v1, F. Javier Nieto et al., 2018

[3] EUXDAT, D3.1 Detailed Specification ofthe End Users' Platform v1, Fabien Castel et al., 2018

[4] MSO4SC, D3.1 Detailed Specifications for the Infrastructure, Cloud Management and MSO Portal, Carlos Fernández et al, 2017

[5] Hani, A. F. M., Paputungan, I. V., & Hassan, M. F. (2015), Renegotiation in Service Level Agreement Management for a Cloud-Based System, ACM Computing Surveys

[6] TNOVA, D4.42 Monitoring and Maintenance, I. Koutras et al., 2016

[7] OpenSmart, http://opensmart.sourceforge.net, retrieved 2018-06-28

[8] Combodo, https://www.combodo.com/itop-193, retrieved 2018-06-28

[9] Citsmart, https://sourceforge.net/projects/citismart, retrieved 2018-06-28

[10] Project-open, http://www.project-open.com, retrieved 2018-06-28

[11] Project-open SLA Management package, http://www.project-open.com/en/package-intranet- sla-management, retrieved 2018-06-28

[12] mOSAIC, http://www.mosaic-cloud.eu/, retrieved 2018-06-28

[13] OTRS, https://otrs.com, retrieved 2018-06-28

[14] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, Ming Xu, Web Services Agreement Specification, Open Grid Forum, March 2007