D4.2 Infrastructure Platform v1

Document Identification

<table>
<thead>
<tr>
<th>Status</th>
<th>Final</th>
<th>Due Date</th>
<th>31/10/2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1.0</td>
<td>Submission Date</td>
<td>08/11/2018</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Related WP</th>
<th>WP4</th>
<th>Document Reference</th>
<th>D4.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related Deliverable(s)</td>
<td>D2.2, D4.1, D5.2</td>
<td>Dissemination Level (*)</td>
<td>PU</td>
</tr>
<tr>
<td>Lead Participant</td>
<td>USTUTT</td>
<td>Lead Author</td>
<td>Nico Struckmann</td>
</tr>
<tr>
<td>Contributors</td>
<td>ATOS FR, ATOS ES</td>
<td>Reviewers</td>
<td>Karel Charvát (WRLS), Dimitrios Moshou (CERTH)</td>
</tr>
</tbody>
</table>

Keywords:

This document is issued within the frame and for the purpose of the EUXDAT project. This project has received funding from the European Union’s Horizon2020 Framework Programme under Grant Agreement No. 777549. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

This document and its content are the property of the EUXDAT Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the EUXDAT Consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the EUXDAT Partners.

Each EUXDAT Partner may use this document in conformity with the EUXDAT Consortium Grant Agreement provisions.

(*) Dissemination level: PU: Public, fully open, e.g. web; CO: Confidential, restricted under conditions set out in Model Grant Agreement; CI: Classified, Int = Internal Working Document, information as referred to in Commission Decision 2001/844/EC.
Document Information

List of Contributors

<table>
<thead>
<tr>
<th>Name</th>
<th>Partner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nico Struckmann</td>
<td>USTUTT</td>
</tr>
<tr>
<td>Michael Gienger</td>
<td>USTUTT</td>
</tr>
<tr>
<td>F. Javier Nieto</td>
<td>ATOS ES</td>
</tr>
<tr>
<td>Spiros Michalakopoulos</td>
<td>ATOS ES</td>
</tr>
<tr>
<td>Fabien Castel</td>
<td>ATOS FR</td>
</tr>
</tbody>
</table>

Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Change editors</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>27/09/2018</td>
<td>N.Struckmann (USTUTT)</td>
<td>Structure drafted</td>
</tr>
<tr>
<td>0.2</td>
<td>04/10/2018</td>
<td>N.Struckmann (USTUTT)</td>
<td>Structure finalized, first content populated</td>
</tr>
<tr>
<td>0.3</td>
<td>11/10/2018</td>
<td>M.Spiros (ATOS ES)</td>
<td>Contributions added.</td>
</tr>
<tr>
<td>0.4</td>
<td>12/10/2018</td>
<td>Fabien Castel (ATOS FR)</td>
<td>Contributions added.</td>
</tr>
<tr>
<td>0.5</td>
<td>16/10/2018</td>
<td>N.Struckmann (USTUTT)</td>
<td>Content extended</td>
</tr>
<tr>
<td>0.6</td>
<td>19/10/2018</td>
<td>N.Struckmann (USTUTT)</td>
<td>Review version</td>
</tr>
<tr>
<td>0.7</td>
<td>26/10/2018</td>
<td>N.Struckmann (USTUTT)</td>
<td>First reviewer’s finding addressed.</td>
</tr>
<tr>
<td>0.8</td>
<td>06/11/2018</td>
<td>N.Struckmann (USTUTT)</td>
<td>Second reviewer’s finding addressed.</td>
</tr>
<tr>
<td>0.9</td>
<td>07/11/2018</td>
<td>ATOS ES</td>
<td>Quality review</td>
</tr>
<tr>
<td>1.0</td>
<td>08/11/2018</td>
<td></td>
<td>FINAL VERSION TO BE SUBMITTED</td>
</tr>
</tbody>
</table>

Quality Control

<table>
<thead>
<tr>
<th>Role</th>
<th>Who (Partner short name)</th>
<th>Approval Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliverable leader</td>
<td>Michael Gienger (USTUTT)</td>
<td>06/11/2018</td>
</tr>
<tr>
<td>Technical manager</td>
<td>Fabien Castel (ATOS FR)</td>
<td>06/11/2018</td>
</tr>
<tr>
<td>Quality manager</td>
<td>Susana Palomares (ATOS ES)</td>
<td>08/11/2018</td>
</tr>
<tr>
<td>Project Manager</td>
<td>F. Javier Nieto (ATOS ES)</td>
<td>08/11/2018</td>
</tr>
</tbody>
</table>
Table of Contents

Document Information ............................................................................................................................ 2
Table of Contents .................................................................................................................................... 3
List of Tables........................................................................................................................................... 5
List of Figures ......................................................................................................................................... 6
List of Acronyms ..................................................................................................................................... 7
1. Executive Summary ......................................................................................................................... 8
2. Purpose of the document .................................................................................................................. 9
   2.1 Relation to another project work ............................................................................................. 9
   2.2 Structure of the document ....................................................................................................... 9
   2.3 Glossary adopted in this document........................................................................................ 10
3. EUXDAT Portal Infrastructure ...................................................................................................... 11
   3.1 Orchestrator ........................................................................................................................... 12
      3.1.1 Installation .......................................................................................................................... 13
      3.1.2 Orchestrator Connectors .................................................................................................. 17
      3.1.3 Orchestrator Tests ............................................................................................................ 18
   3.2 Computation and Storage Backends ...................................................................................... 18
      3.2.1 Cloud ............................................................................................................................... 19
      3.2.2 HPC ................................................................................................................................... 21
4. Infrastructure Platform Overview .................................................................................................. 28
   4.1 Staging Concept ....................................................................................................................... 29
      4.1.1 Production Stage ............................................................................................................... 29
      4.1.2 Integration Stage ............................................................................................................. 30
      4.1.3 Development Stage ........................................................................................................ 30
   4.2 Physical Servers ....................................................................................................................... 30
      4.2.1 Frontend and Backup Storage ......................................................................................... 31
      4.2.2 Development Stage ......................................................................................................... 32
      4.2.3 Integration Stage ............................................................................................................... 32
      4.2.4 Production Stage ............................................................................................................. 32
4.3 VMs for Infrastructure Services .................................................................................. 34
  4.3.1 Nginx front-end proxy & router ........................................................................... 34
  4.3.2 GIT ....................................................................................................................... 35
  4.3.3 Gerrit ..................................................................................................................... 35
  4.3.4 Jenkins ................................................................................................................ 35
4.4 Backup Concept ............................................................................................................ 36
5. Development Workflow .................................................................................................. 38
  5.1 GIT Repository, Branches and Workflow .................................................................. 38
    5.1.1 Central repo ....................................................................................................... 39
    5.1.2 Local Repo ......................................................................................................... 39
    5.1.3 Development Repository ................................................................................ 40
    5.1.4 Gerrit Review Repository ................................................................................ 40
  5.2 Jenkins Continuous Integration ................................................................................... 40
    5.2.1 Gerrit Trigger .................................................................................................... 41
    5.2.2 Kubernetes Integration ..................................................................................... 41
  5.3 Gerrit Code Review ..................................................................................................... 41
6. Conclusions ..................................................................................................................... 42
7. References ....................................................................................................................... 43
8. Annexes ........................................................................................................................... 44
  8.1 Orchestrator .yaml files ......................................................................................... 44
    8.1.1 Blueprint.yaml .................................................................................................. 44
    8.1.2 plugin.yaml ...................................................................................................... 48
List of Tables

Table 1: Cray XC40  22
Table 2: Hardware Properties of Backup Storage  31
Table 3: Hardware Properties of Development Stage  32
Table 4: Hardware Properties of Integration Stage  32
Table 5: Kubernetes Cluster Machines  33
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EUXDAT Portal Infrastructure Overview</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>Central End-User Portal</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>Cloudify Console</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>HPC Plugin Test and Connector Classes</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>HPC Batch-System Job Scheduling</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>Cray XC40 architecture</td>
<td>26</td>
</tr>
<tr>
<td>7</td>
<td>EUXDAT Infrastructure</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>EUXDAT Staging Concept</td>
<td>29</td>
</tr>
<tr>
<td>9</td>
<td>EUXDAT Development Environment Servers</td>
<td>31</td>
</tr>
<tr>
<td>10</td>
<td>OTC Cloud environment</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>Physical Frontend</td>
<td>34</td>
</tr>
<tr>
<td>13</td>
<td>Jenkins Overview (src: <a href="https://circleci.com/docs/assets/img/docs/jenkins-ui.png">https://circleci.com/docs/assets/img/docs/jenkins-ui.png</a>)</td>
<td>36</td>
</tr>
<tr>
<td>14</td>
<td>EUXDAT Development Workflow</td>
<td>38</td>
</tr>
<tr>
<td>15</td>
<td>Gerrit Trigger</td>
<td>41</td>
</tr>
</tbody>
</table>
# List of Acronyms

<table>
<thead>
<tr>
<th>Abbreviation / acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>Dx.y</td>
<td>Deliverable number y belonging to WP x</td>
</tr>
<tr>
<td>EC</td>
<td>European Commission</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GridFTP</td>
<td>Parallel high throughput file server</td>
</tr>
<tr>
<td>HPC</td>
<td>High Performance Computing</td>
</tr>
<tr>
<td>NFS</td>
<td>Network File System</td>
</tr>
<tr>
<td>SCP</td>
<td>Secure Copy</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
</tbody>
</table>
1. Executive Summary

This document has the purpose to describe the first version of the EUXDAT Infrastructure Platform as the name implies. It takes into account the previous deliverable “D4.1 Detailed Specification of the Infrastructure Platform” defining the first version of the infrastructure, based on requirements identified in "D2.1 Description of Proposed Pilots and Requirements" and "D2.2 EUXDAT e-Infrastructure Definition v1”.

This document is the second one of a series describing the progress over the project’s runtime in depth. For the complete list of succeeding documents within WP4, please refer to section 2.1.

The recently carried out work on the infrastructure was strongly focused in first place on a collaborative development environment, a workflow including continuous integration and quality checks, besides required services, with the target to ensure a stable end-user production platform separated from development environments.

The main content starts with a description of the current EUXDAT Portal Infrastructure, comprising the central components and building blocks in addition to the computation backends and storage facilities including access mechanisms and properties. As next major building block an overview about the infrastructure platform is provided, highlighting the staging concept applied to ensure a stable production platform alongside with a flexible collaborative development and testing environment. Further, physical server and virtualized services are presented. As last major part the development workflow is explained in detail, comprising the different code repositories, code branches and review workflow, as well as the continuous integration concept for automated building, testing and deployment.

The document ends with a conclusion summarizing the main achievements of the reporting period and discusses future work and challenges.
2. Purpose of the document

This document describes the first version of the EUXDAT Infrastructure Platform. It is the follow up document for D4.1 “Detailed Specification of the Infrastructure Platform” which was focused, as the name implies, on the specification of the platform while this document describes what has been setup and configured, refined and improved so far since then. It describes the actual status rather than plans. It provides an overview about the current state of the EUXDAT portal, including its computation and storage backend facilities. Further, it highlights the physical infrastructure setup, the servers, virtualized services available already, besides the access mechanisms of the Cloud and HPC computing environments. Additionally, the development workflow and code repository structures with their git branches are presented.

2.1 Relation to another project work

This document, Deliverable D4.2 "Infrastructure Platform v1" is taking into account D2.2 "EUXDAT e-Infrastructure Definition v1" and D4.1 "Detailed Specification of the Infrastructure Platform v1" upon which the current infrastructure setup and configuration, described in this document, is aligned to. Further, the EUXDAT e-Infrastructure describing the first release of the End Users’ Platform and the Infrastructure Platform, defined in D5.2 “EUXDAT e-Infrastructure v1”, is considered.

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

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

2.2 Structure of the document

This document is structured in 5 major chapters, divided in subsections containing the important details and technical aspects. Additionally, there is an Annex with further technical details about a central component of the end-user platform and its underlying infrastructure.

Chapter 3 starts with an introduction into the current state of the infrastructure platform and goes into details concerning the end-user portal and available compute and storage backend. Details about the compute environments and access mechanisms are described in its subsections.

Chapter 4 provides an overview about the staging concept in its details, followed by an explanation of the purpose for each of the three stages. Additionally, highlights the physical server infrastructure and available services needed for an effective collaborative development. Further, the backup concept for disaster recovery is introduced.
Chapter 5 presents the collaborative development workflow. Code repositories and versioning aspects, quality assurance by the help of continuous integration. Followed by a description of each of the services in place and how they are utilized.

Chapter 6 finishes the document with a conclusion of the work up to date presented in this document. It outlines the main achievements alongside with challenges addressed so far.

2.3 Glossary adopted in this document

- **Continuous Integration.** Software engineering practice in which frequent, isolated changes are immediately compiled, deployed, tested and reported on when they are added to the code base.

- **Code Review.** Part of the quality assurance takes place before Continuous Integration, and comprises one or several developers checking changes of the code before being merged into the code base.

- **Deployment.** Software Deployment is the process of making software available for use.

- **Stage.** A development environment for implementing and testing installation/configuration/migration scripts and procedures before going into production.

- **Scheduler.** A software/service deciding based on rules, policies and SLAs when a workload is planned to be deployed on which resources.

- **Resource Manager.** A software/service managing any kind of (consumable) IT resources such as compute nodes, accelerator cards and GPUs, besides monitoring and accounting.
3. EUXDAT Portal Infrastructure

The EUXDAT Portal is the central access point for end-users. It provides the core services, which will enable users to utilize all the services offered, while these do not need to bother with HPC and Cloud environments, deployment mechanisms, data staging or other technical details.

The following figure Figure 1 gives an overview of the whole EUXDAT Platform, comprising its central portal, computation and storage backend, as well as use-case related data repositories.

![Figure 1: EUXDAT Portal Infrastructure Overview](image)

The next subsection will highlight the central Portal (blue box in top right corner of the figure) followed by a detailed overview about the Cloud and HPC facilities.
3.1 Orchestrator

The architecture of the EUXDAT Portal comprises several building bricks, presented in the following figure. Greyed out components are part of the designed architecture but not in place, yet, while the coloured box, the Orchestrator, is available already.

As the figure depicts, the first component is in place, as a preliminary version under development. The orchestrator is responsible for scheduling user workloads transparently on the available computation backend.

It accepts end-user workload submitted through the frontends, interfacing with the orchestrator. Details about these front-ends, including access to local data APIs is described in D5.2 [3].

A high-level description of the Hybrid Cloud/HPC Orchestrator used in EUXDAT was introduced in [6]. A closer look at the submodules that make up the Orchestrator were explained in [7]. Here we describe the installation, deployment and parts of its inner workings.

The Orchestrator is based on Cloudify [10], with the addition of Atos’ Cloudify HPC Plugin, first introduced in the project MSO4SC [11].
3.1.1 Installation

Cloudify has been installed on the infrastructure machine, on a Kubernetes pod. It is currently the only container within the pod, and specifically it is the `cloudifyplatform/community:latest` image from Docker hub [12].

```
[root@euxdat-test-0001 ~]# kubectl get pods --all-namespaces -o=jsonpath='{range .items[*]}{"\n"}{.metadata.name}{":\t"}{range .spec.containers[*]}{.image}{", "}{end}{end}' | sort | grep cfy-man
cfy-manager-68dd67cc8-x6hdk: cloudifyplatform/community,
```

It is currently using port 30003 for development purposes, see Figure 3. In the future, this console will not be visible to end users, since jobs will be submitted via the EUXDAT portal.

![Cloudify Console](image)

**Figure 3 Cloudify Console**

Cloudify is built for orchestrating for the cloud. Orchestrating for HPC is provided by the Cloudify HPC plugin which is installed via Cloudify’s TOSCA based blueprints [13]. The installation of the plugin is achieved via an import line such as the following, from the `blueprint.yaml` file (see full file in Annex 8.1.1):

```yaml
... imports: ...
    # HPC plugin
    http://raw.githubusercontent.com/MSO4SC/cloudify-hpc-plugin/master/plugin.yaml ...
```
plugin.yaml (full file in Annex 8.1.2) imports the actual code:

```yaml
plugins:
  hpc:
    ...
    source: https://github.com/MSO4SC/cloudify-hpc-plugin/archive/canary.zip
    package_name: cloudify-hpc-plugin
    package_version: '1.4'
```

The Cloudify manager is controlled via the command line with cfy. The example we demonstrate here describes 4 HPC jobs, as can be seen in the blueprint, of which the 2nd and 3rd are parallel jobs. The blueprint is uploaded to cloudify with the command:

```bash
ID=EUXDAT_DEMO
cfy blueprints upload -b $ID blueprint.yaml # Upload the blueprint
```

We see this reflected in the console:

Next the deployment is created with the `--skip-plugins-validation` flag:

```bash
cfy deployments create -b $ID \
  -i ../local-blueprint-inputs.yaml --skip-plugins-validation # Create the application instance
```
This is also reflected in the console:

Next the application instance is bootstrapped:

```bash
cfy executions start -d $ID install # Bootstrap the app instance
```

Finally, the four jobs described in the blueprint are executed with the command:

```bash
cfy executions start -d $ID run_jobs # Execute the app instance workflow
```

The Cloudify console is configurable, with different views into the deployments and executions. For example, here you can take a look at the events logs and take a peek at the resources being deployed:
The commands to clean up are given next, for completeness:

```bash
cfy executions start -d $ID uninstall # Revert the app instance
cfy deployments delete $ID # Remove the app instance
cfy blueprints delete $ID # Remove the blueprint
```

All the above commands will be executed via the Portal UI, and the intricacies and details will be hidden to most users.

The cloudify plugin was written following the Cloudify documentation [14] and is written in Python and organized in classes. Some of the most important ones are: JobRequester, JobGraphNode, JobGraphInstance, Monitor, and WorkloadManager.
3.1.2 Orchestrator Connectors

The Orchestrator speaks to the cloud virtual machines via Cloudify commands, whereas with the HPC machines via scheduler and resource manager specific commands. The communication in the other direction is done via exporters in the case of HPC and cloudify commands in the case of cloud.

Specifically, Orchestrator → Cloud communication is handled by class Bash, whereas Orchestrator → HPC is handled by classes Slurm and Torque, see Figure 4. In the figure, Slurm, Torque and Bash are all representing Workload Managers. The Workload Managers never initiate “conversations” directly with the machines, instead it is always the Orchestrator who “asks first”. Concerning the cloud, other Cloudify plugins are in charge of creating and destroying virtual machines, but it is still the same scenario: the Orchestrator initiates communication.

![Figure 4: HPC Plugin Test and Connector Classes](image)

Other direction communications are handled by Cloudify in the case of Cloud → Orchestrator, and by external modules for HPC → Orchestrator. Specifically: slurm_exporter (a server that collects metrics from Slurm and exposes them in the Prometheus format), torque_exporter (ditto from Torque), are modules written specifically for this purpose. These modules are written in Go and have not yet been deployed for EUXDAT but are mentioned in this deliverable for completeness.
The exporters are in charge of monitoring the infrastructure and communicating back to the Orchestrator. There is a further exporter module called `exporter_orchestrator` which is the exporter controller for all active exporters in the system.

There is a further exporter module called `exporter_orchestrator`, which is the exporter controller for all active exporters in the system.

### 3.1.3 Orchestrator Tests

The tests can be run using `nose` tests [15], which is automated by the `tox` tool [16]. Through tox, we have various test “environments”, or groupings of tests. The main groups of tests are `py27` and `flake8` which can be executed with the command:

```
    tox -e py27; tox -e flake8
```

The tests are set up to permit the use of a real HPC infrastructure. The input parameters need to be set in a yaml file, usually named `blueprint-inputs.yaml`. Of course, the “simulate: true” label (see 8.1.1) needs to be removed or commented out.

The tests are ready to be integrated into a basic continuous integration system which will allow their automatic execution. It is currently being used with Travis in another project, whenever a git push is done on master or canary, or whenever there’s a push request. Of course, the test blueprints should all have “simulate: true” set, since it is not a good idea to have automated tests on real HPC machines!

The classes responsible for running the tests are `TestPlugin`, `TestSlurm` and `TestTorque` (see Figure 4).

### 3.2 Computation and Storage Backends

Besides the central entry point for end-users, the EUXDAT Portal, there are resources needed to process the workloads. Theirs is a Cloud and an HPC environment transparently available in the background, including data storage facilities. End-users will not have direct access to the compute environments; instead they will utilize the EUXDAT Portal components and services to stage data and deploy applications. The succeeding sections provide an overview about the distinct compute resources.
3.2.1 Cloud

The Cloud environment is in its core properties very different from the HPC environment. Clouds utilize in comparison to traditional HPC shared resources, in terms of several users’ workloads might be deployed on the same physical node. Cloud resources are virtualized and dynamically assigned, workloads can also be migrated to other compute nodes and over-commitment of virtualized resources (assign more than physically avail) is possible.

3.2.1.1 Platform as a Service architecture

The cloud backend of the EUXDAT e-infrastructure is hosted on the OTC (Open Telecom Cloud) cloud facility, provide by T-Systems through the Atos Mundi offer that is part of the ESA DIAS program.

In particular, the EUXDAT end users' platform is hosted on this cloud infrastructure. It is important at this point to differentiate the cloud resources hosting the end users' platform from the cloud resources that are used as elastic computation resources. This distinction can be unclear as all these resources are hosted on the same OTC environment and belong to the same overall infrastructure. What differentiates them is the usage that is performed with these resources:

- VMs and storage hosting the platform are static resources, running continuously, manually provisioned by a platform administrator
- VMs and storage used as computation resources are elastic resources, running only on-demand, automatically provisioned when an algorithm is executed

Static VMs and storage units are provisioned by a platform administrator using the OTC web API. Provisioning scripts have been developed to automatize this process. These scripts are still manually executed by a platform administrator when there is a need for a new instance of the platform. Using scripts enables reproducing identically several platform instances. It also improves the overall process reliability and decrease potential human mistakes.

The platform components are independent application running on a same infrastructure, integrated and communicating together to provide a set of features. Running all these components on same VMs would be challenging (incompatibilities on the dependencies from all the components, high complexity induced by running several applications on the same system, network issues…). Using one VM for each component would require the provisioning of a lot of VMs which would increase the overall hosting cost. The solution that is used for the platform is to use an intermediate PaaS – Platform as a Service – layer to abstract the infrastructure layer for the applications. The hosted applications are managed as containers. The VMs are gathered as nodes into a cluster. Containers are executed in whatever node of the cluster.

Docker is used as the basis containerization technology. Kubernetes is used as the PaaS technology to setup the cluster and ensure the container management.
3.2.1.1.2 Cloud Elastic Resources

The need in terms of computation resources is very variable depending on the algorithms executed on the platform by the users. To adapt to this variable consumption, an elastic provisioning of the cloud computation resources has to be implemented.

The elasticity management on EUXD AT relies on OpenStack [17] APIs provided by the OTC infrastructure [18]. A specific configuration has to be done on the EUXD AT e-infrastructure Kubernetes instance to connect with these APIs, and a logic has to be implemented to describe how the cluster should increase or decrease according to the activity of the hosted applications.

This feature is not implemented on the version 1 of the e-infrastructure but will be a major task on the version 2.

3.2.1.1.3 Application Management

All the components constituting the end users' platform are packaged into Docker containers. This packaging is done by writing Docker specific configuration file (i.e. Dockerfile).

Containers are executed by Kubernetes into the cluster. The way each container is handled by Kubernetes is configured into Kubernetes specific configuration files. In this file all the following features can be configured:

- Network (all containers can be executed on a single virtual network or several sub-networks can be defined if some isolation is required)
- Redundancy (to automatically execute several instances of an application and manage the load-balancing between each instance)
- failure handling (to automatically restart an application if it fails inside the container)
- port mapping (to expose some ports to the outside)
- shared volume (to share folders between a container and the host system and/or between containers)

3.2.1.2 Storage

The OTC infrastructure offers several storage solutions:

- **Object storage**: an object-based data storage service. The data storage is accessed via an Internet link by means of the HTTP and HTTPS protocols. Object storage differs from other storage types by its use of objects as a basis of storage. Every object is clearly identified so that it can be accessed via the network. Object storage offers highly simplified access mechanisms and a high level of scalability. Object storage allows buckets (containers) and storage objects to be created, and objects to be called up and deleted.

- **Elastic volume**: provides the customer with data storage in block level storage capacities. Up to 10 block storage actions of different types may be assigned to each OTC virtual machines. With this service, identical copies are stored on multiple storage nodes, in order to store the data inventory at 99.99995 percent without loss. The following block storage types are available:
  - 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" flavor)

3.2.1.3 Access Mechanisms

Elastic cloud resources are accessed through Kubernetes APIs (container provisioning, execution and monitoring standard APIs) or through higher-level custom APIs developed in the EUXDAT project.

3.2.2 HPC

The HPC environment compared to Clouds is more static and inflexible, restricts run time of jobs, and compute nodes are user-exclusive. This approach provides maximal bare-metal performance and prevents bottlenecks for eg I/O operations, which may occur on shared nodes like Clouds offer. On the other hand, there are unused resources, which need to be taken into account when it comes to meeting a decision where workload should be deployed under consideration of costs.

Another difference to Clouds is a shared file-system amongst compute nodes, and a fast-intermediate storage for computation, besides a long-term tape archive.

3.2.2.1 Computation

The HPC environment is a batch-system, means user workload is not executed the moment it is submitted, but queued and scheduled. Execution takes place as soon as enough resources are free to satisfy job requirements. User’s workload is usually deployed on many compute nodes in parallel.
3.2.2.1.1 System Properties
The Supercomputer provided to the EUXDAT platform by USTUTT is a CRAY XC40. Following table describes its properties in detail. The CRAY has compared to clusters its own networking technology, pre- and post-processing nodes, and workload is started on some kind of master nodes (called MOM nodes) instead of the actual compute nodes.

Table 1: Cray XC40

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>• 7.4 Pflops</td>
</tr>
<tr>
<td></td>
<td>• 5.64 Pflops (76% Peak), November 2015 list rank 8 (2016/06 rank 9, 2016/11 rank 14, 2017/06 rank 17, 2017/11 rank 19)</td>
</tr>
<tr>
<td></td>
<td>• 0.138 Pflops (2% Peak), November 2015 HPCG results rank 6 (2016/06 rank 10, 2016/11 rank 12, 2017/06 rank 13, 2017/11 rank 14)</td>
</tr>
<tr>
<td>Cray Cascade Cabinets</td>
<td>41</td>
</tr>
<tr>
<td>Property</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Number of Compute Nodes</td>
<td>7712 (dual socket)</td>
</tr>
</tbody>
</table>
| Compute Processors                           | • Total number of CPUs: 7712*2= 15424 Intel Haswell E5-2680v3 2.5 GHz, 12 Cores, 2 HT/Core  
• Total number of Cores: 15424*12= 185088                                                                                                                                                               |  |
| Compute Memory on Scalar Processors         | • Memory Type: DDR4  
• Memory per Compute Node: 128GB  
• Total Scalar Compute Memory: 987136GB= 964TB                                                                                                                                                       |  |
| Interconnect                                 | Cray Aries                                                                                                                                                                                                                                                          |  |
| 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 |
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 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</td>
<td></td>
</tr>
<tr>
<td>User Storage</td>
<td>~10 PB</td>
</tr>
<tr>
<td>• Lustre Workspace Capacity</td>
<td></td>
</tr>
<tr>
<td>Cray Linux Environment (CLE)</td>
<td>Yes</td>
</tr>
<tr>
<td>• Compute Node Linux</td>
<td></td>
</tr>
<tr>
<td>• Cluster Compatibility Mode (CCM)</td>
<td></td>
</tr>
<tr>
<td>• Data Virtualization Services (DVS)</td>
<td></td>
</tr>
<tr>
<td>PGI Compiling Suite (FORTRAN, C, C++) including Accelerator</td>
<td>25 user (shared with Step 1)</td>
</tr>
<tr>
<td>Cray Developer Toolkit</td>
<td>Unlimited Users</td>
</tr>
<tr>
<td>• Cray Message Passing Toolkit (MPI, SHMEM, PMI, DMAPP, Global Arrays)</td>
<td></td>
</tr>
<tr>
<td>• PAPI</td>
<td></td>
</tr>
<tr>
<td>• GNU compiler and libraries</td>
<td></td>
</tr>
<tr>
<td>• JAVA</td>
<td></td>
</tr>
<tr>
<td>• Environment setup (Modules)</td>
<td></td>
</tr>
<tr>
<td>• Cray Debugging Support Tools</td>
<td></td>
</tr>
<tr>
<td>o Lgdb</td>
<td></td>
</tr>
<tr>
<td>o STAT</td>
<td></td>
</tr>
<tr>
<td>o ATP</td>
<td></td>
</tr>
<tr>
<td>Cray Programming Environment</td>
<td>Unlimited Users</td>
</tr>
<tr>
<td>• Cray Compiling Environment (FORTRAN, C, C++)</td>
<td></td>
</tr>
<tr>
<td>• Cray Performance Monitoring and Analysis</td>
<td></td>
</tr>
<tr>
<td>o Cray PAT</td>
<td></td>
</tr>
<tr>
<td>o Cray Apprentice2</td>
<td></td>
</tr>
<tr>
<td>• Cray Math and Scientific Libraries</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>o Cray Optimized BLAS</td>
<td></td>
</tr>
<tr>
<td>o Cray Optimized LAPACK</td>
<td></td>
</tr>
<tr>
<td>o Cray Optimized ScaLAPACK</td>
<td></td>
</tr>
<tr>
<td>o IRT (Iterative Refinement Toolkit)</td>
<td></td>
</tr>
<tr>
<td>Alinea DDT Debugger</td>
<td>2048 Processes (shared with Step 1)</td>
</tr>
<tr>
<td>Lustre Parallel Filesystem</td>
<td>Licensed on all Sockets</td>
</tr>
<tr>
<td>Intel Composer XE</td>
<td>10 Seats</td>
</tr>
<tr>
<td>• Intel C++ Compiler XE</td>
<td></td>
</tr>
<tr>
<td>• Intel Fortran Compiler XE</td>
<td></td>
</tr>
<tr>
<td>• Intel Parallel Debugger Extension</td>
<td></td>
</tr>
<tr>
<td>• Intel Integrated Performance Primitives</td>
<td></td>
</tr>
<tr>
<td>• Intel Cilk Plus</td>
<td></td>
</tr>
<tr>
<td>• Intel Parallel Building Blocks</td>
<td></td>
</tr>
<tr>
<td>• Intel Threading Building Blocks</td>
<td></td>
</tr>
<tr>
<td>• Intel Math Kernel Library</td>
<td></td>
</tr>
</tbody>
</table>

The conceptual architecture is presented in the figure below
3.2.2.1.2 Batch System

The Batch system available at HLRS is a Moab/PBS Torque combination. Moab\(^1\) is a sophisticated scheduler, responsible for SLA management and job scheduling, while PBS Torque\(^2\) is a resource manager monitoring and managing any kind of compute resources such as nodes, storage and software licenses. The resource manager is deploying workload scheduled by Moab for execution on compute nodes, further, it prepares the compute node environment and cleans up afterwards.

3.2.2.1.3 Application Management

Applications are managed in the HPC environment as shared binaries and several versions are in parallel available by the help of the Modules System\(^3\). User have defaults defined by administrators but are free to load any other version of an application or library suiting their needs the best.

---

\(^3\) [http://modules.sourceforge.net/](http://modules.sourceforge.net/)
In addition, users can compile any application in user space on the frontends, which have the same environment and packages available as the compute nodes. Via the shared file-system, compiled binaries do not need to be transferred to compute hosts allocated for an user.

Workload is expected to be submitted as job script, started on the first node of a job allocation.

### 3.2.2.2 Storage

The HPC environment provides three different storages, each serving a particular purpose. There is a NFS\(^4\) based shared user \$HOME\, mounted on the front-ends as well as on each compute node. Data stored there is immediately accessible to the user on all nodes. It is intended to hold input files and store job results.

For intermediate data written out during a computation, there is a fast-parallel workspace file-system in place, a Lustre\(^5\) capable to handle huge amounts of data and files in parallel. These workspaces need to be requested/created by the user and expire after 30 days. These are not intended to store any valuable data after a job is done. Users are supposed to copy it to their home and download it or if required archive it on the long-term storage.

The long-term storage provided is a tape library with a capacity of many Petabytes but very slow access times as tapes have to be mounted first usually. It is not intended for anything else than archiving valuable data needed at a later point in time.

### 3.2.2.3 Access Mechanisms

The HPC environment is accessed securely via the frontends by the help of SSH\(^6\). After login users have a Linux terminal to check the queues with command `qstat`, submit their job scripts to Moab/Torque by the help of PBS Torque command line tools `qsub`, cancel a job with `qdel` or manage their workspaces with command `ws_list`, `ws_allocate`, `ws_release`, `ws_extend` and un/load applications and libraries with command `module`.

For data staging there are two possibilities available, either via SCP\(^7\) connection to the common user frontends accessible also via SSH, and as second way there are data frontends with GridFTP and 10GBit connectivity available, capable to handle huge data volumes and allows parallel transfers.

---

\(^5\) [http://lustre.org/](http://lustre.org/)
\(^6\) [https://www.openssh.com/](https://www.openssh.com/)
\(^7\) [http://www.scp-wiki.net/](http://www.scp-wiki.net/)
4. Infrastructure Platform Overview

This chapter highlights the physical infrastructure platform of the EUXDAT Portal, with a focus on the server setup for the different stages, the infrastructure servers and services. The following figure provides an overview of physical and virtualized servers the infrastructure platform is composed of.

![Figure 7: EUXDAT Infrastructure](image)

There are four bigger building blocks, from a more abstract view, namely the EXUDAT Portal (all three stages), a Cloud computation backend, an HPC computation backend and a separate hosting environment with services required for an efficient collaborative development.

In the succeeding subsections all three stages of the Portal, as well as the separate hosting infrastructure for (virtualized) services is presented. For details about the Cloud and HPC backend, please refer to section 3.2.
4.1 Staging Concept

The EUXDAT development infrastructure is managed in different stages allowing to test and develop in parallel without affecting end-users on the platform. In this section the staging concept and its benefits will be presented, differences and purpose of each stage is explained and an overview of the whole setup is given. The development workflow is prominently presented in section 5 while this section focuses on the stages, their configuration, interconnectivity, underlying hardware and purpose.

![Figure 8: EUXDAT Staging Concept](image)

The staging concept comprises in short three different environments with equal configurations and capabilities. It is intended to provide an environment for functional ad-hoc testing, an integration testing environment to ensure interoperability with other components as well as a distinct environment for running regressing tests to ensure previous functionality is not broken by a bug fix, new feature or other update of a component’s code base. Development and testing can take place without affecting end-users or risking instability of the production environment.

All three stages use the same Cloud and HPC backend as Figure 7 depicts, but are as highlighted distinct environments not impacted by each other if something goes wrong.

4.1.1 Production Stage
The production stage represents the end-user platform. No development nor testing is supposed to take place here. Only tested and approved component versions are deployed in order to provide a stable platform to end-users.

The production stage is not a single physical server like the integration and development stages, but instead hosted in the Cloud provided by ATOS, which consists of several physical compute nodes and a controller. Kubernetes is used to manage the Cloud backend as well as the deployment of services the EUXDAT Portal is comprised of.

For further details about the Cloud and its setup and hardware properties, please refer to section 3.2.1 and for more details about the production stage please refer to section 4.2.4.

4.1.2 Integration Stage

The integration stage is a copy of the production environment with all available components deployed in a stable version. It is the place where code is supposed to be tested after a review has been passed and before it goes into production.

The integration stage is a single physical machine with Docker support, and compared to the production stage, which is hosted in the Cloud backend it comes there is no need for Kubernetes as deployment mechanism. For system properties, please refer to section 4.2.3.

4.1.3 Development Stage

The purpose of the development stage is as the name implies, solely for development. Developers are supposed to test their written code on this stage and fix problems, which were not occurring on his local machine. Compared to the integration stage single components are tested, and compared to the production stage there are no end-users.

The development stage is as the integration stage a single physical machine with identical properties. It comes also with Docker support and has no Kubernetes deployment available. For system properties, please refer to section 4.2.2.

4.2 Physical Servers

Besides the production infrastructure, there is a development and testing infrastructure required in addition to ensure an efficient collaborative development in parallel to a stable production platform. The following figure depicts the physical server setup for the different stages and services.
4.2.1 Frontend and Backup Storage

The Frontend machine is hosting several VMs in which the services needed for collaborative development are hosted. For details about these services deployed in VMs, please refer to section 4.3. There is a storage system attached via NFS to the physical frontend, to VMs and to the two stages hosted at HLRS. It is utilized for backups of system configurations, VM snapshots and such data, in order to enable a quick disaster recovery in case of failing hardware or else.

**Table 2: Hardware Properties of Backup Storage**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2x Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.4GHz, in total 12 cores/24 threads</td>
</tr>
<tr>
<td>RAM</td>
<td>126 GB</td>
</tr>
<tr>
<td>Local Storage</td>
<td>2TB system storage and 42TB NFS storage</td>
</tr>
<tr>
<td>OS</td>
<td>CentOS 7</td>
</tr>
<tr>
<td>Relevant Software</td>
<td>SSH, virsh, ZFS raid, NFS server</td>
</tr>
</tbody>
</table>
4.2.2 Development Stage

The development stage is mirroring the production environment it its important aspects, more about the staging concept is to be found in section 4.1.

Table 3: Hardware Properties of Development Stage

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel(R) Xeon(R) CPU X5560 @ 2.80GHz, in total 8 cores/16 threads</td>
</tr>
<tr>
<td>RAM</td>
<td>96 GB</td>
</tr>
<tr>
<td>Local Storage</td>
<td>700 GB</td>
</tr>
<tr>
<td>OS</td>
<td>Debian GNU/Linux 9 (stretch)</td>
</tr>
<tr>
<td>Relevant Software</td>
<td>SSH, git, docker</td>
</tr>
</tbody>
</table>

4.2.3 Integration Stage

The server for the integration stage is a 1:1 copy of the development stage, with the only difference that it is not used by developers, but by Jenkins in first place. Besides a git repository, there is no difference to the development stage, thus for details please refer to the previous section.

Table 4: Hardware Properties of Integration Stage

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel(R) Xeon(R) CPU X5560 @ 2.80GHz, in total 8 cores/16 threads</td>
</tr>
<tr>
<td>RAM</td>
<td>96 GB</td>
</tr>
<tr>
<td>Local Storage</td>
<td>700 GB</td>
</tr>
<tr>
<td>OS</td>
<td>Debian GNU/Linux 9 (stretch)</td>
</tr>
<tr>
<td>Relevant Software</td>
<td>SSH, git, docker</td>
</tr>
</tbody>
</table>

4.2.4 Production Stage

The production environment, hosted on Cloud resources, is organized as follow:
The bounce server is the only access point from the outside. An SSH connection has to be initiated to this machine. This step requires transmitting an SSH public key to the platform administrator so that the SSH connection could be established.

Once connected to the bounce server, all the other machines can be accessed through SSH without any more constraints. The computing cluster is composed of a Kubernetes Master and 2 Kubernetes Workers. The following table recap the characteristics of the different machines.

**Table 5: Kubernetes Cluster Machines**

<table>
<thead>
<tr>
<th>Type</th>
<th>Machine Name</th>
<th>CPU</th>
<th>RAM</th>
<th>Local Disk Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>euxdat-test-0001</td>
<td>4</td>
<td>8 GB</td>
<td>100 GB</td>
</tr>
<tr>
<td>Worker</td>
<td>euxdat-test-0002</td>
<td>8</td>
<td>16 GB</td>
<td>100 GB</td>
</tr>
<tr>
<td>Worker</td>
<td>euxdat-test-0003</td>
<td>8</td>
<td>16 GB</td>
<td>100 GB</td>
</tr>
</tbody>
</table>
4.3 VMs for Infrastructure Services

All services, needed for a development and testing infrastructure for the EUXDAT project, are hosted in virtual machines as they do not need a lot of compute resources and for a more convenient maintainability by the help of snapshots and the ability to migrate these virtualized services to another host if the physical host hardware is degrading.

As the figure Figure 11 shows, the physical host is not reachable from the internet, but a virtualized frontend proxy host, only. All other services are behind this frontend acting as SSH and HTTP proxy. Following subsection explains each of the services in place and their purpose.

4.3.1 Nginx front-end proxy & router

The frontend VM is the only reachable access point from the internet. It runs a nginx-webserver acting as HTTP(s) proxy for Gerrit (https://eufe.hlrs.de/gerrit) and Jenkins (https://eufe.hlrs.de/jenkins),
besides acting as router forwarding gerrits SSH port as well as the development and integration stages’ SSH port.

### 4.3.2 GIT

The central GIT repository server for EUXDAT is a FusionForge\(^8\) v6.0 instance hosted at [https://projects.hlrs.de](https://projects.hlrs.de). This service is a central service and not dedicated to the EUXDAT infrastructure but for completeness also mentioned. For further info about existing git repositories, branches and workflow, please refer to section 5.1.

### 4.3.3 Gerrit

Gerrit is the state of the art review tool for git commits and comes along with a web user-interface for web-browser access and offers in addition SSH access for git clients. Gerrit enables developers to review each other’s code, attach comments to code parts and approve or reject a commit. It can be considered a git based repository of pending changes to be merged into the code base.

![Gerrit Code Review](https://www.gerritcodereview.com/images/sbs.png)

**Figure 12: Gerrit Code Review** (src: [https://www.gerritcodereview.com/images/sbs.png](https://www.gerritcodereview.com/images/sbs.png))

Figure 12 shows an example of a review in Gerrit’s Web UI. Comments can be attached to any code part and replied to.

### 4.3.4 Jenkins

Jenkins is a well-known self-contained open source Continuous Integration service providing by the help of hundreds of different plugins, support for building code, deploying build binaries and automate regression testing for software development projects. It comes with a Gerrit plugin that enables to fetch code automatically after a review has been successfully passed and run defined actions like a regression testing suite and summarize the results in a report.

It is usually deployed in the background supporting developers with their daily work without manual interaction. Jenkins is written in Java and has a web-based user-interface.

---

\(^8\) [https://fusionforge.org/](https://fusionforge.org/)
Figure 13: Jenkins Overview (src: https://circleci.com/docs/assets/img/docs/jenkins-ui.png)

Figure 13 provides an example how Jenkin’s project overview looks like, providing the status for the last execution (e.g. a build with regression test) and when it took place.

4.4 Backup Concept

In order to mitigate failing hardware and broken operating systems or services, as well as to provide disaster recovery, different kinds of backups are taken frequently. The backup concept consists of three parts.

1. **Operating system and services configuration**
   Configurations of operating systems as well as deployed services are versioned by the help of a local git repository, recording all changes. After each update or configuration change, the files are committed.

2. **VM snapshots**
   All virtualized hosts are backed up with snapshots. Older snapshots will be cleaned up after a while, since the latest working state is the only desired one for a recovery.

3. **Backup File-system**
There is a dedicated storage for backups, a ZFS raid mounted on all physical servers. Copies of the server’s operating system and services configuration (local git repo) and service data (eg databases) are stored there. Further, redundant copies of VM snapshots are stored there as well. The File-system itself is setup with striping and backup disks and thus is tolerant against single drives failing and capable to restore data reliably in such a case.

4. **Automated, cron job steered Backups**

All backups of local git repositories, service data and VM snapshots are copied frequently to the ZFS backup file-system. These tasks are automated by the help of Linux cron jobs and need no manual interaction. Older backups are kept for some time before being wiped, as their value and importance lowers with each succeeding backup of working services and machines taken.
5. Development Workflow

In this section the workflow for EUXDAT is explained in detail. This comprises the different code repositories and their branches, as well tools and services utilized for development and testing, ensure a stable production platform and a productive collaborative development environment.

![EUXDAT Development Workflow Diagram](image)

As first, a developer pulls the latest master from the central FusionForge repository. On the local machine, a branch is created (develop/feature/fix) and pushed to Dev-Stage where it is ad-hoc tested. After successful testing code is pushed into Gerrit for review. After review has been passed Jenkins fetches the updated master branch and deploys it on integration stage where further (regression) tests are executed. After passing it successful, code is pushed into the central repository and can be manual deployed on the production stage.

5.1 GIT Repository, Branches and Workflow

For the code development there are git repositories utilized. The overall workflow is organized in branches, enabling a robust and convenient parallel development process.
In the following list each of the branches used in the development infrastructure and workflow, including its purpose is explained:

- **release-***
  Are intended to bundle a new version of the platform, may branch off develop, but must merge into **master**. It contains stable major releases of the overall platform

- **master**
  The master branch holds production code; all developments are merged into master after approval

- **develop**
  Contains pre-production code under development, features are merged into develop, and develop itself is merged into master

- **feature-***
  These branches are utilized for development of features, which are then merged into develop

- **hotfix-***
  Hotfixes are resolving urgent issues with the production version and may branch off latest master, must merge into master

Not each repository in the EUXDAT infrastructure holds all of the branches, in the following chapters there is pointed out which repo has which purpose and branches.

### 5.1.1 Central repo

The central repository is hosted at [https://projects.hlrs.de](https://projects.hlrs.de) (Fusion Forge) and its purpose is solely to hold stable production code intended to be fetched by EUXDAT developers to base their changes on, as well as being published. The central repository holds the following branches:

- **release**
- **master**

Besides SSH based access, the fusion forge server also provides a web UI where one can browse the code and download it by the help of a web-browser, too.

### 5.1.2 Local Repo

The local repository is a cloned master branch from the central repository. A develop, feature, hotfix or release branch is forked off the master and code is written to implement new functionality, a bug fix, or bundle a new release. The local repository resides on the developers working machine, storing local changes to be submitted for review after being tested on the development stage. It may contain a few or all of the following branches:

- **master**
- **develop**
• feature-*
• hotfix-*
• release-*

5.1.3 Development Repository

The development stage is hosted on the development stage and reachable via SSH, only, under the following path:

eufe.hlrs.de:22550/repo/euxdat.git

Developers push their local changes during development on this stage in order to carry out functional ad-hoc test and debugging. After code is considered stable and ready, it is pushed into gerrit and reviewed. The available branches are obviously the same as in the local repository, since it can be considered the upstream repo for new developments or bug fixes:

• master
• develop
• feature-*
• hotfix-*
• release-*

5.1.4 Gerrit Review Repository

Gerrit is the GIT review tool utilized in the development workflow. The moment a developer has finished his task and considers his code stable and suitable to be merged into master, he pushed it either from local repository or from the development stage as signed commit to gerrit. Since all branches under active development are reviewed before merged into the master after passing review and automated testing, this repository holds the same branches as the local or development repositories do:

• master
• develop
• feature-*
• hotfix-*
• release-*

5.2 Jenkins Continuous Integration

Jenkins is in place as Continuous Integration service, coupled with gerrit. It is utilized to automatically fetch code which passed a review and needs to be compiled for the Cloud and/or HPC environment. Code which fails to compile or which does not pass automated regression tests will be rejected and not merged into the master. If it passes the review, compilation is successful and test suite has been passed.
as well, code is pushed upstream to the central repository. Deployment on the production stage is not carried out automatically but triggered manually instead. Since Jenkins only fetches approved changes only from gerrit, which had been merged into the master or release-* branch, there are only these two-branched available:

- master
- release-*

### 5.2.1 Gerrit Trigger

Jenkins offers a plugin called Gerrit Trigger, which is configured to fetch code after passing a review. The code is then compiled in a next step and deployed on integration stage where regression tests are carried out and a report is generated.

![Gerrit Trigger](image)

**Figure 15: Gerrit Trigger**

### 5.2.2 Kubernetes Integration

In order to enable Jenkins to carry out regression testing of EUXDAT Portal components in the Cloud hosting environment, it must be able to deploy these builds automatically. The Kubernetes plugin for Jenkins provides this kind of automated interaction with the Cloud.

### 5.3 Gerrit Code Review

After developers have finished their code and carried out functional ad-hoc tests on development stage, they submit their changes for review to gerrit. Gerrit sends a notification to the assigned reviewers, who then can check and comment the changes. If changes are considered fine and are approved, Jenkins is triggered to fetch the code for automated building and testing.
6. Conclusions

This deliverable describes the first version of the EUXDAT e-infrastructure platform and is based on the previous deliverable D4.1 “Detailed Specification of the Infrastructure Platform”.  
It comprises the state of the central portal for end-user access, its interconnection to computation and storage backend and available development infrastructure and services. As this document is written at the end of the first year, the focus is mostly on the development infrastructure in terms of physical and virtualized servers, services and environments required during the project runtime for a collaborative and effective development of the EUXDAT platform in combination with a stable production setup for end-users.

The document starts with a presentation of the current state of the central portal and its connectivity to the Cloud and HPC backend, describing these in detail including access mechanisms and system properties. In the succeeding part the staging concept together with the whole physical and virtualized infrastructure is highlighted, addressing the needs of a flexible development and stable production environment.

Followed by how this is workflow is designed and configured, existing repos, code branches and their purpose, as well as the code review process and continuous integration for automated building, testing and deployment.

The future work focuses mostly on the missing central components the EUXDAT end-user’s portal is comprised of.
## 7. References

[1] EUXDAT. *D2.2 EUXDAT e-Infrastructure Definition*. Nieto, F. Javier. 2018


[3] EUXDAT. *D5.2 EUXDAT e-Infrastructure v1*. Castel, Fabien. 2018


[7] [https://cloudify.co/guide/3.0/understanding-blueprints.html](https://cloudify.co/guide/3.0/understanding-blueprints.html), retrieved 2018-10-11

[8] [https://docs.cloudify.co/4.0.0/plugins/creating-your-own-plugin/](https://docs.cloudify.co/4.0.0/plugins/creating-your-own-plugin/), retrieved 2018-10-11


8. Annexes

8.1 Orchestrator .yaml files

8.1.1 Blueprint.yaml

```
# Copyright (c) 2018-2020 EUXDAT - spiros.michalakopoulos.external@atos.net
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

tosca_definitions_version: cloudify_dsl_1_3

inputs:
  # Monitor
  monitor_entrypoint:
    description: Monitor entrypoint IP
    default: ""
    type: string

# Job prefix name
```
job_prefix:
  description: Job name prefix in HPCs
  default: "cfyhpc"
  type: string

partition:
  description: Partition in which the jobs will run
  default: "public"
  type: string

euxdat_hpc_primary:
  description: Configuration for the primary HPC to be used
  default: {}

euxdat_dataset_model:
  description: Model dataset
  default: ""

euxdat_datacatalogue_entrypoint:
  description: entrypoint of the data catalogue
  default: "http://193.144.35.207"

euxdat_datacatalogue_key:
  description: API Key to publish the outputs
  default: ""
euxdat_outdataset_outputs_at:
  description: ID of the CKAN output dataset
  default: ""

node_templates:
  first_hpc:
    type: hpc.nodes.Compute
    properties:
      config: { get_input: euxdat_hpc_primary }
      external_monitor_entrypoint: { get_input: monitor_entrypoint }
      job_prefix: { get_input: job_prefix }
      base_dir: "$LUSTRE"
      workdir_prefix: "four"
      skip_cleanup: True
      simulate: True  # COMMENT to test against a real HPC
first_job:
    type: hpc.nodes.job
    properties:
        job_options:
            type: 'SRUN'
            modules:
                - gcc/5.3.0
            partition: { get_input: partition }
            command: 'touch fourth_example_1.test'
            nodes: 1
            tasks: 1
            tasks_per_node: 1
            max_time: '00:01:00'
        deployment:
            bootstrap: 'scripts/bootstrap_example.sh'
            revert: 'scripts/revert_example.sh'
            inputs:
                - 'first_job'
            skip_cleanup: True
        relationships:
            - type: job_contained_in_hpc
              target: first_hpc

second_parallel_job:
    type: hpc.nodes.singularity_job
    properties:
        job_options:
            modules:
                - gcc/5.3.0
                - openmpi/1.10.2
                - singularity/2.3.1
            partition: { get_input: partition }
            image: '$LUSTRE/openmpi_1.10.7_ring.img'
            home: '$HOME:/home/$USER'
            volumes:
                - '/scratch'
            command: 'ring > fourth_example_2.test'
            nodes: 1
            tasks: 1
            tasks_per_node: 1
            max_time: '00:01:00'
deployment:
  bootstrap: 'scripts/singularity_bootstrap_example.sh'
  revert: 'scripts/singularity_revert_example.sh'
inputs:
  - '$LUSTRE'
  - 'openmpi_1.10.7_ring.img'
skip_cleanup: True
relationships:
  - type: job_contained_in_hpc
target: first_hpc
  - type: job_depends_on
target: first_job

third_parallel_job:
type: hpc.nodes.singularity_job
properties:
  job_options:
    modules:
      - gcc/5.3.0
      - openmpi/1.10.2
      - singularity/2.3.1
    partition: { get_input: partition }
  image: '$LUSTRE/openmpi_1.10.7_ring.img'
  home: '$HOME:/home/$USER'
  volumes:
    - '/scratch'
  command: 'ring > fourth_example_3.test'
  nodes: 1
  tasks: 1
  tasks_per_node: 1
  max_time: '00:01:00'
deployment:
  bootstrap: 'scripts/singularity_bootstrap_example.sh'
  revert: 'scripts/singularity_revert_example.sh'
inputs:
  - '$LUSTRE'
  - 'openmpi_1.10.7_ring.img'
skip_cleanup: True
relationships:
  - type: job_contained_in_hpc
target: first_hpc
- type: job_depends_on
target: first_job

fourth_job:
type: hpc.nodes.job
properties:
  job_options:
    type: 'SBATCH'
    command: "touch.script fourth_example_4.test"
  deployment:
    bootstrap: 'scripts/bootstrap_sbatch_example.sh'
    revert: 'scripts/revert_sbatch_example.sh'
  inputs:
    - 'fourth_job'
    - { get_input: partition }
  skip_cleanup: True

8.1.2 plugin.yaml

plugins:
  hpc:
    # Could be 'central_deployment_agent' or 'host_agent'.
    # If 'central_deployment_agent', this plugin will be executed on the
    # deployment dedicated agent, otherwise it will be executed on the host
    executor: central_deployment_agent

    # URL to archive containing the plugin or name of directory containing
    # the plugin if it is included in the the blueprint directory under the
    # "plugins" directory.
    source: https://github.com/MSO4SC/cloudify-hpc-plugin/archive/canary.zip
    package_name: cloudify-hpc-plugin
    package_version: '1.4'

workflows:
  run_jobs:
    mapping: hpc.hpc_plugin.workflows.run_jobs

node_types:
  hpc.nodes.WorkloadManager:
    derived_from: cloudify.nodes.Compute
    properties:
config:
    description: credentials, timezone and workload manager
  
  external_monitor_entrypoint:
    description: Monitor entrypoint, port and orchestrator port
    default: ""
    type: string
  
  external_monitor_port:
    description: Monitor entrypoint, port and orchestrator port
    default: "":9090"
    type: string
  
  external_monitor_type:
    description: Monitor type, e.g PROMETHEUS
    default: "PROMETHEUS"
    type: string
  
  external_monitor_orchestrator_port:
    description: Monitor entrypoint, port and orchestrator port
    default: "":8079"
    type: string
  
  job_prefix:
    description: Job name prefix for this HPC
    default: "cfyhpc"
    type: string
  
  base_dir:
    description: Root directory of all executions
    default: "$HOME"
    type: string
  
  workdir_prefix:
    description: Prefix of the working directory instead of blueprint name
    default: ""
    type: string
  
  monitor_period:
    description: Seconds to check job status.
    default: 60
    type: integer
  
  simulate:
    description: Set to true to simulate job without sending it
    type: boolean
    default: False
  
  skip_cleanup:
    description: True to not clean all files at deployment removal
    type: boolean
default: False

agent_config:
default:
  install_method: none

interfaces:
cloudify.interfaces.lifecycle:
  configure:
    implementation: hpc.hpc_plugin.tasks.configure_execution
    inputs:
      config:
        default: { get_property: [SELF, config] }
      base_dir:
        default: { get_property: [SELF, base_dir] }
      workdir_prefix:
        default: { get_property: [SELF, workdir_prefix] }
      simulate:
        default: { get_property: [SELF, simulate] }

delete:
    implementation: hpc.hpc_plugin.tasks.cleanup_execution
    inputs:
      config:
        default: { get_property: [SELF, config] }
      skip:
        default: { get_property: [SELF, skip_cleanup] }
      simulate:
        default: { get_property: [SELF, simulate] }

cloudify.interfaces.monitoring:
  start:
    implementation: hpc.hpc_plugin.tasks.start_monitoring_hpc
    inputs:
      config:
        default: { get_property: [SELF, config] }
      external_monitor_entrypoint:
        default: { get_property: [SELF, external_monitor_entrypoint] }
      external_monitor_port:
        default: { get_property: [SELF, external_monitor_port] }
      external_monitor_orchestrator_port:
        default: { get_property: [SELF, external_monitor_orchestrator_port] }
      simulate:
default: { get_property: [SELF, simulate] }

stop:
  implementation: hpc.hpc_plugin.tasks.stop_monitoring_hpc
  inputs:
    config:
      default: { get_property: [SELF, config] }
    external_monitor_entrypoint:
      default: { get_property: [SELF, external_monitor_entrypoint] }
    external_monitor_port:
      default: { get_property: [SELF, external_monitor_port] }
    external_monitor_orchestrator_port:
      default: { get_property: [SELF, external_monitor_orchestrator_port] }
    simulate:
      default: { get_property: [SELF, simulate] }

hpc.nodes.Job:
  derived_from: cloudify.nodes.Root
  properties:
    deployment:
      description: Deployment script and inputs
      default: {}
    job_options:
      description: Job main command and options
    publish:
      description: Config to publish its outputs
      default: []
    skip_cleanup:
      description: True to not clean after execution (debug purposes)
      type: boolean
      default: False
  interfaces:
    cloudify.interfaces.lifecycle:
      start: # needs to be start to have the hpc credentials
        implementation: hpc.hpc_plugin.tasks.bootstrap_job
        inputs:
          deployment:
            description: Deployment scripts and inputs
            default: { get_property: [SELF, deployment] }
          skip_cleanup:
default: { get_property: [SELF, skip_cleanup] }

stop:
  implementation: hpc.hpc_plugin.tasks.revert_job
  inputs:
    deployment:
      description: Undeployment script and inputs
      default: { get_property: [SELF, deployment] }
    skip_cleanup:
      default: { get_property: [SELF, skip_cleanup] }

hpc.interfaces.lifecycle:
  queue:
    implementation: hpc.hpc_plugin.tasks.send_job
    inputs:
      job_options:
        default: { get_property: [SELF, job_options] }
  publish:
    implementation: hpc.hpc_plugin.tasks.publish
    inputs:
      publish_options:
        default: { get_property: [SELF, publish] }
  cleanup:
    implementation: hpc.hpc_plugin.tasks.cleanup_job
    inputs:
      job_options:
        default: { get_property: [SELF, job_options] }
      skip:
        default: { get_property: [SELF, skip_cleanup] }
  cancel:
    implementation: hpc.hpc_plugin.tasks.stop_job
    inputs:
      job_options:
        default: { get_property: [SELF, job_options] }

hpc.nodes.SingularityJob:
  derived_from: hpc.nodes.Job

relationships:
  wm_contained_in:
    derived_from: cloudify.relationships.contained_in
    source_interfaces:
      cloudify.interfaces.relationship_lifecycle:
preconfigure:
  implementation: hpc.hpc_plugin.tasks.preconfigure_wm
inputs:
  config:
    default: { get_property: [SOURCE, config] }
simulate:
    default: { get_property: [SOURCE, simulate] }
job_managed_by_wm:
  derived_from: cloudify.relationships.contained_in
source_interfaces:
  cloudify.interfaces.relationship_lifecycle:
    preconfigure:
      implementation: hpc.hpc_plugin.tasks.preconfigure_job
inputs:
  config:
    default: { get_property: [TARGET, config] }
  external_monitor_entrypoint:
    default: { get_property: [TARGET, external_monitor_entrypoint] }
  external_monitor_port:
    default: { get_property: [TARGET, external_monitor_port] }
  external_monitor_type:
    default: { get_property: [TARGET, external_monitor_type] }
  external_monitor_orchestrator_port:
    default: { get_property: [TARGET, external_monitor_orchestrator_port] }
  job_prefix:
    default: { get_property: [TARGET, job_prefix] }
  monitor_period:
    default: { get_property: [TARGET, monitor_period] }
simulate:
  default: { get_property: [TARGET, simulate] }
job_depends_on:
  derived_from: cloudify.relationships.depends_on