# FORTISSIMO

## D3.5

## Operational Sustainability Best Practice Final Report

| | | |
|---|---|---|
| **Workpackage:** | 3 | Core Service Deployment and Facility Operation |
| **Author(s):** | Jochen Buchholz | USTUTT |
| | | |
| **Authorized by** | Bastian Koller | USTUTT |
| **Reviewer** | Terry Sloan | EPCC |
| **Reviewer** | Ivan Spisso | CINECA |
| **Reviewer** | Aleksander Grum | ARCTUR |
| **Dissemination Level** | PU | |

| Date | Author | Comments | Version | Status |
|---|---|---|---|---|
| 2016-12-12 | J. Buchholz | Initial draft based on the first report | V0.1 | Draft |
| 2016-12-13 | J. Buchholz | Adding Drupal Chapter, extending marketplace chapter | V0.2 | Draft |
| 2016-12-14 | J. Buchholz | Updating all sections | V0.3 | Draft |
| 2016-12-14 | J. Buchholz | Formatting | V0.4 | Draft |
| 2016-12-20 | J. Buchholz | Addressing reviewer comments | V0.5 | Draft |
| 2016-12-28 | J. Buchholz | Addressing minor comments | V0.6 | Draft |
| 2016-12-29 | J. Buchholz | Finalizing | V1.0 | Final |

## Executive Summary

This Operational Sustainability Best Practice Report provides guidelines on the creation and maintenance of HPC Cloud environments, to allow others to benefit from the knowledge obtained when the Fortissimo [1] HPC Cloud Marketplace was created.

HPC Clouds are collections of HPC resources offered for use on marketplaces. The HPC resources provided by different widely distributed organisations may differ significantly especially in access methods and policies. A platform for providing access to HPC Clouds that meets all requirements from users and providers therefore seems to be very complex and difficult to achieve.

This report describes a generic approach for technical provision of HPC resources in an HPC Cloud. This report also explains the reasons because under certain circumstances the Fortissimo HPC Cloud Marketplace has deviated from this generic strategy.

The intended audience for this report are project members and managers as well as system operators facing the challenge of implementing HPC Clouds. We present the generic approach and details about tools, services and configurations, especially at the network and system layer as implemented in the Fortissimo Cloud Marketplace. In addition, the report can provide valuable input for third parties that are interested in the design, implementation and operation of HPC Clouds.

This report is published at the end of the Fortissimo project after a duration 42 months and is an extension to the first version published at the original mid-term point of the project at month 18. The Fortissimo project focus has primarily been on the design and implementation of the Fortissimo Marketplace and on the start of its operation. As more experience has been gathered during operation for a wider and wider set of users, additional material on “best practices” has been gathered and is now published in this extended version of the Deliverable.

Nevertheless, the design and implementation phases are critical, since they lay the foundations of the Marketplace. Therefore, it is worthwhile to document the rationale behind the decisions taken and complement that with the initial experience in provisoning and operating the Marketplace.

# Table of Contents

# Table of Figures



# Table of Tables

# 1 Introduction

A great deal of knowledge in a variety of areas is necessary to build and sustainably operate a HPC Cloud environment. Since the term sustainability is not self-explanatory and is often used to refer to the preservation of the natural environment and its ecosystems, we define in sub-section 1.1 what we mean by sustainability in the context of IT systems. This definition is then used as the basis for this report.

The majority of this document is concerned with the details of the overall installation, development and operational processes necessary for creating and operating an HPC Cloud sustainably. It is important to note that these processes must satisfy a number of different types of user, in particular the end-user who wishes to exploit the resources available via this Marketplace and the providers who wish to make resources available via this Marketplace.

The functional dimension represents different user groups and their involvement in the creation of the platform (e.g. administrators of the hosting environment, developers, testers, resources providers and end users). As shown in Figure 1, the time dimension is divided into design, implementation and documentation. These are executed in this order for most tasks. Regarding the functional dimension, Figure 1 shows that the activities associated with design and documentation are fairly homogeneous. Implementation activities in this functional dimension however are closely interlinked and often dependent upon one another.

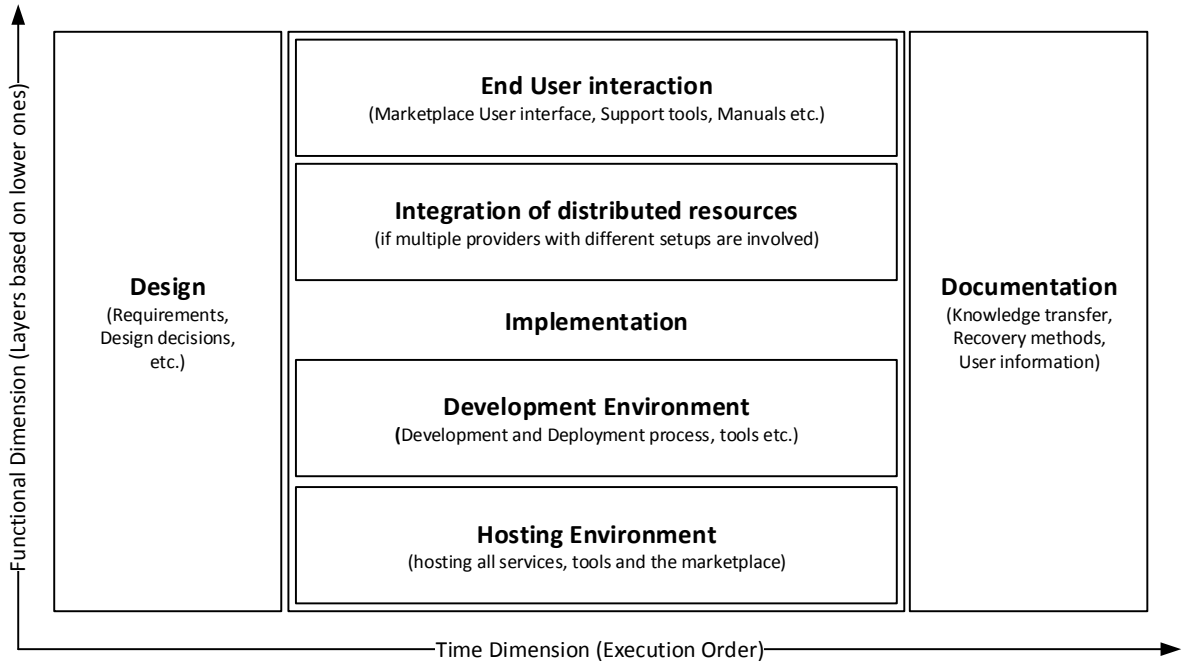


**Figure 1 Schematic structure for creating the platform.**

Section 2 of this document is concerned with the design phase.

Section 3 covers the different functional areas related to the different user groups/views of an HPC Cloud Marketplace. The sub-sections of this describe the hosting environment (3.1), the development environment (3.2), the Marketplace itself (3.3) including integration of distributed (HPC) resources (3.3.3) and end user interactions (3.4), respectively.

Section 4 discusses documentation and how it can help to achieve sustainability.

Finally, section 5 summarises our ideas and thoughts on how to create a sustainable HPC Cloud Marketplace.

## 1.1 About Sustainability

To achieve a sustainable HPC Cloud environment we must first define what we mean by the term sustainability.

In the context of this Fortissimo report sustainability refers primarily to two aspects of the HPC Cloud:

- Reduce the effort needed for ensuring Marketplace reliability and availability in the long term as much as possible;
- Create a standardized and well documented way for adding new services to the marketplace enriching it with new functionality for HPC users.

It also refers to a lesser extent to each of the following:

- Simplicity of architecture in order to reduce maintenance overheads.
- User experience with easy to use interfaces and procedures that minimise the entry barriers for new users. New users need to be able to learn and understand how to use the Marketplace quickly without the help of experts or detailed documentation.
- Ensuring traceability of changes and design decisions to allow developers (in particular if they are new) easy familiarization with the solution. This also helps guaranteeing a highly available and reliable infrastructure.

Whilst this definition of sustainability is quite generic, it does help Fortissimo focus on the following important and pragmatic goals:

- Reducing long-term costs (including staff costs) to increase competitiveness.
- Minimizing service outages to reduce user frustration and increase income.
- Reducing the number of user complaints in order to limit support workload, ensure quick turnaround and improve user satisfaction.
- Attracting additional users and increasing revenue by providing good user experience.
- Ensuring flexibility to allow the integration of new features as quickly as possible so leading to better competitiveness.

# 2 Design – Towards a Sustainable HPC Cloud

In Fortissimo, it has become apparent that preparation is key towards achieving sustainability. This means that during the design phase possible conflicts have to be detected and solved. In addition, a flexible, easily implementable architecture has to be designed. This section describes three aspects of the Fortissimo design where this has been observed.

## 2.1 Rapid Development and Prototyping

Due to the need for rapid development of a prototype at the start of Fortissimo, the development environment and its accompanying integration deployment stage did change significantly over time as the need for new capabilities became apparent. These permanent changes and reconfigurations resulted in changing behaviour on the integration stage. Instead of initially copying the integration stage to preproduction, we instead set-up the preproduction stage based solely on the default installation of the required software components and copied only the necessary, traceable changes from the integration stage. This way it could be ensured that the behaviour of the HPC Cloud Marketplace was as expected. Whilst this required more effort to do, the long term results justified it.

Freezing a design too early in a rapid prototyping project can lead to extremely complex structures that are static, hard to change and include untraceable content. This happens especially when successive requirements appear and have to be met without the chance to revisit the basic design. By taking stock of the content and make-up of the integration prior to migration to pre-production and only implementing traceable changes we have minimised this danger.

In the last year of the project we decided to replace the underlying portal software completely by switching from the Java based Liferay to the PHP based Drupal. The main reason was that Liferay caused continuously increasing problems, especially in content staging etc. where some of the features were relied upon from the beginning. To reduce the time for migration to Drupal a web development company was employed to assist in transforming all designs to Drupal and to set up a basic environment. This took some time and so to prevent delays, we also used some additional test Drupal instances for development and testing of new features. These test instances were never intended to be used for later development or in the staging process, but these helped hasten the first part of the migration which mainly involved getting familiar with Drupal and the concepts.

## 2.2 Exploiting Virtual Machines

Fortissimo uses three deployment stages, integration, pre-production and production, for the building and testing of candidate services in their transition from development to operational use. This separates the development of the HPC Cloud Marketplace from its operation.

For the major part of the project we developed the services based on the Liferay platform. Each of these Liferay stages used three virtual machines (VMs), one each for the proxy, database and application server. Whilst this may seem to be too much complexity for the overall Fortissimo goal, it provided more flexibility and offered higher scalability. For example, it offered the option of simply adding new application servers or resources or delivering static content directly through the proxy. Similarly, a separate database server with very fast disks and large memory and a Tomcat server [2] with fast CPUs could be more effective than an all-in-one machine. Using virtual machines also allowed simple addition of further centralized services on demand without additional hardware. Only in the case of a regular heavy load were additional resources needed. Virtual machines can also easily be migrated to new hardware which we did during the project to improve performance.

Later in the project we moved away from Liferay towards Drupal. First we intended to use the same hosting environment as before simply by replacing the Tomcat server with a standard Apache. But the involvement of a web development company and their proposal to use the web hosting company Acquia [3] changed our plans. The main reason was that Acquia already had the whole environment ready for Drupal including the three stages, content migration and monitoring. Also the experience of the web development company with this environment and the higher availability were taken into account. Up to now we do not regret the change to an external hosting company. Maybe we later host it again locally but it is not planned. Nevertheless, the development support regarding build chain and other aspects are still hosted locally on virtual machines and will remain there.

## 2.3 Context information

In Fortissimo, the importance of collecting early as much contextual information as possible about both the development and operational aspects of the HPC Cloud Marketplace has been apparent. Such input included:

- *The functional requirements:* This included in particular, the features to provide the end users with, and, the features necessary for the internal connection between the HPC Cloud Marketplace and the HPC resource providers. Creating a well-balanced pool of features is critical for attracting customers. During the project we repeatedly collected requirements from the included experiments. The experiments represent exemplary use cases and implement their use case based on HPC resources. So we were in the advantageous situation that we received new requirements condensed into a few use cases instead of having to collect this information from many users who often have the same problems.
- *The frequency and intensity of planned tests and changes to the HPC Cloud Marketplace:* This may influence the design of the different deployment stages. We found that the integration stage is the most important stage and also the most complex one for the whole development. Specific settings or additions need to be made there to support debugging and continuous integration. The integration stage is used heavily especially when dealing with potentially problematic issues. So you should spend much effort to set up this stage. The production stage is also critical but from a completely different view point. Changes are quite rare compared to integration, only well tested and working modules are included. New content is added directly on production but it is based on predefined content types and normally causes no problems. For the production stage security and performance are much more important and need increased attention. The preproduction stage is the least critical and strongly depends on how you use it, see section 3.2.1.
- *Deployment stages:* We decided to set up three deployment stages in order to separate development installations from preproduction and production. The Integration stage is where all developers are free to play, test and integrate their software with the others. In preproduction we set up the final look and feel for the Marketplace so that this should not be influenced by changing items in the integration stage. In pre-production we also ran the final tests with data from the production stage to ensure everything worked fine. The production stage is the environment offered to the end users. For other projects more deployment stages for the test process or even for specific services might be useful. With the switch to Drupal the workflow suggested by Acquia includes effectively a fourth stages, the local development stage. By providing the Dev Desktop application the developer has a local Drupal environment where they can implements their components, test them and then deploy them on the

integration/development stage. This allows each developer to work independently from the central stages and so not affect others during basic development. An automated build chain only operates between the local stage and the development stage to enforce build rules and tests.

- *Support developers of candidate services:* The need to support developers who will provide candidate services has an impact on the development process and the choice of tools required. It is advisable to provide some code templates with the basic structure for all components. This reduces or even solves problems that can occur with such developers.
    - Naming of components of all components must follow the same guidelines.
    - Dependencies can be simplified by using the same level of inclusion and reference to depending packages.
    - When separate database constructs are needed (especially additional new tables) the way to create them can be reduced to a single mechanism. We found that some candidate services used a separate database, others separate tables, some used a manually configured database connection, other used the tomcat included connection.
    - Examples of access rights enforcement can reduce the learning overhead for all developers since a working solution is already provided.
- *Level of user knowledge:* The target user groups and their knowledge greatly influence the user interfaces and user centric documentation. For example, experts from IT domains may need less support but when they do specialized knowledge is required.
- *Resource availability:* Which resources (hardware, software, human resources, knowledge) are available to realize the project? Inclusion of an unknown (to the Fortissimo staff) tool will require more effort and may cause delays or even temporary unavailability of the HPC Cloud Marketplace. This risk has to be managed.
- *The expected usage intensity, loads, data sizes etc.:* Without such information, it is not possible to predict or decide how to deal with scalability issues or define the set of resources to be made available in the form of virtual machines, distinct physical machines or specialized hardware.
- *Security requirements:* These must be described. The security policies and guidelines must be documented to avoid ad hoc solutions. These include data and operational security, traceability, incident response, privacy protection, legal issues, etc.

Early on in the collection of contextual information some basic architecture ideas can be discounted or improved for prioritized evaluation later on. When all the information is available it should be possible to sort it into different categories related to the different functional areas described in the later sections of this document. The finer grained and complete the information is, the more specific the architecture can and should be. It is important to cover all aspects and collect proper information in order to define an architecture that can also potentially fulfil future emerging requirements. If something is not yet known, the range of possible situations should be described.

# 3 Implementation

In this chapter we discuss approaches to implementation that can support or improve sustainability. Some sections refer to aspects where the details may be unclear until you read the whole document. Since many areas partially overlap it was not possible to avoid this inconvenience.

At first, we describe the Hosting environment in section 3.1 where we install our systems for supporting developers. Developer support itself is described in section 3.2. The Marketplace (described in section 3.3) is also partially installed on top of the Hosting environment and the outputs from development. We finish this chapter by describing the end user interactions in section 3.4.

## *3.1 Hosting Environment*

The hosting environment provides the basic system infrastructure for the HPC Cloud Marketplace. It is operated by system administrators via secure console connections (SSH [4], [5]). Developers of HPC Cloud Marketplace services are considered as users of the hosting environment.

The hosting environment (see Figure 2 in sub-section 3.1.3) is quite generic for all kinds of development purposes. The construction and operation of this environment has been based on the following principles. After the switch to Drupal the three stages associated with Liferay became less important. Only the integration stage was still used for further development of Java components, while the other stages were only kept for archive purpose. They were replaced by the stages hosted externally now at Acquia. The basic principles however remained the same, more details are explained in a separate section (3.1.7).

The remainder of this section contains information about different aspects of the hosting environment. It provides useful information you should be aware of depending on your requirements when setting up an HPC Cloud. We start with centralized services in sub-section 3.1.1, since a single instance is enough and even more powerful than having it replicated for each stage. In sub-section 3.1.2 we briefly describe the advantages of using virtualization for such a setup followed in sub-section 3.1.3 with the network layout supporting strong separation between the stages but offering enough flexibility for future requirements. Sub-section 3.1.4 covers the user management to have consistent account information everywhere. The remaining sub-sections include information about data caches to reduce network traffic (3.1.5), backup and restore functionality in case of data loss (3.1.6) and the already mentioned information about the new Drupal hosting(3.1.7),

### 3.1.1 Centralized Services

In Fortissimo a number of services (see Table 1) have been instantiated only once. These are referred to as the centralized services. Thus we simplified their use, reduced load (e.g. network traffic) or kept data consistent. The services that can be centralised include the following:

- *User Management*: this is discussed in more detail in sub-section 3.1.4. For the user management we have two different areas: the development environment itself, and; the Marketplace. We separated these to gain more flexibility and so not rely on the capability of a component to support both areas.
- *Logging facilities:* By default logs are written to a specific directory on the machine implementing a service. Log analysis is difficult when more than one service is affected and is also very problematic if clocks are not synchronized. When using a

centralized logging host, all logs may be consolidated into one file or filtered to fit your needs. It is also easier to store and backup a single log facility instead of many of them.

- *DNS (Domain Name System) service:* This may be helpful to keep names up-to-date on all client systems especially in large installations with frequent name changes. In Fortissimo we have created a static list of all machines together with some aliases for possible future usage, and we decided to use the default file-based host resolving mechanism on each machine.
- *OS (Operating System) repository cache:* In Fortissimo many packages are installed on several machines and we needed to update all VMs regularly. This can cause high and unnecessary network traffic at update time. We therefore created a local repository cache from the official repository, which all VMs download their updates from. Of course, since all packages for the specific OS distribution have to be downloaded into the cache first, the overall amount of data may turn out to be even higher. The external traffic for the repository cache can be moved to happen when traffic is low (i.e. during the night to reduce peak traffic). Besides that we are also more independent from external network problems or downtimes of the external repository. Additional installations also do not cause any additional outside traffic (new VMs as well as new packages on the existing machines).
- *Version control system:* Whilst different organisations are creating services and associated code for the Fortissimo HPC Cloud Marketplace, it is important to have only one source code repository for all services that will be deployed. Otherwise code has to be moved from one code base under version control to another or potentially added from a code base without any underlying version control system. In Fortissimo we have a project wide SVN (Subversion) [6] that is located outside the hosting environment. To reduce connection and bandwidth problems we mirror the SVN read-only in the hosting environment so that build tools can access it easily. With the migration to Drupal at Acquia an additional Git repository is provided (by Acquia) which is fullly integrated into the deployment cycle defined by Acquia. Thus we will commit further development to the Acquia Git. At the same time it was announced that the old SVN will be shut down soon. So that development (including still used java components) is not lost we will create a new repository in the hosting environment and update the build chain accordingly. This Git may later be used for the local development by developers too so as to not overstretch the Acquia repository with changes far ahead of deployment. The repository in the hosting environment will also be used for the updated build chain for Acquia.
- *Build tools and related services:* In Fortissimo these tools and services are part of the deployment stages. Nevertheless, there is no need to have multiple instances of these services. For Fortissimo Liferay-based development we used a Jenkins [7] server for continuous integration and as a code artefact repository, and Nexus [8] for storing build results. Especially for deploying services on the pre- and production stages it was advisable to use the identical deployment file (build artefact) accepted by QA (quality assurance) to prevent inadvertent use of different and untested build options. With the migration to Drupal the old functionality of Jenkins is still used, but only for Java components. The full build and test support for PHP will be added in future with the same level of automation as before.
- *Test tools:* All tools for testing services and validating code can be used in similar ways as the build tools. They are not used steadily without interruption but only after

builds and then intensively but not in parallel for e.g. different stages. The test tools for PHP are not in place at the end of Fortissimo but will be needed in the successor project Fortissimo 2.

The status of possible centralized services in Fortissimo is described in the following tables together with a short notice about why we decided as we did.

| Service | Centralized | Reason |
|---|---|---|
| SSH-Gateway | Yes | Security, only one IP address exposed. |
| LDAP (User Mgmt.) | Yes | Ensure Data consistency. |
| Home directory | Yes | Data consistency, reducing needed storage. |
| XMPP | Yes | Jabber [9] server, installed only once. |
| SVN | Yes | Reduce remote data transfer. |
| Jenkins | Yes | Needed mainly for integration stage and possibly preproduction and production stages. |
| Mail relay | Yes | To have full and easy control capabilities over email. |
| Database | No | Configuration for the database connections is simple. Enables strict separation of the deployment stages. |
| Application server | No | Deployment stages might need different performance. Also development activities must not influence production. |
| Logging | Partially | The basic system logging is centralised while some of the services especially the application server still run their local logging. Main reason is that they don't use the system wide logging, instead they directly use log files |

**Table 1 Centralized services in Fortissimo**

### 3.1.2 Virtualization

In Fortissimo due to uncertainties about usage loads only one physical machine was used with multiple virtual machines (VM) installed on it to separate the different services.

Using VMs provides a number of advantages:

- A reduced number of physical servers and hence less hardware maintenance effort and reduced power consumption. (A single server usually consumes less energy than several smaller servers whose combined CPU power and memory match it).
- The possibility to easily add new VMs without the need for more space or power as long as the machine is powerful enough for it. Especially for the Drupal migration we created some additional test VMs by cloning a template and installing Drupal. Within less than an hour the new instance was completely functional.
- Better efficiency in terms of CPU usage and I/O. The overall CPU performance (and

so power consumption) compared to multiple machines can be reduced without affecting the usability of the system.

- The flexibility to replace a single heavily used VM with a physical server.
- Easy backup/snapshot capabilities without the need to configure backups for each host. In Fortissimo the amount of data is small on some machines and so we created a backup that is executed per host with different collections of directories and backup routines.
- Network filtering can be performed by the physical machine on all traffic instead of separately configuring the network of all client VMs. Within Fortissimo we do the main network setup including network separation and firewalling on the physical machine while service specific filtering is still done on the proper service VM.
- Easy replacement of underlying hardware. During the project we moved all VMs to a new hardware base and did that one by one without any problems during migration. The downtime for the VM was limited to a bit more than the downtime during normal reboot of a VM.

Of course there are also some disadvantages:

- Single point of hardware failure due to only one server being used.
- A powerful server is needed to support many VMs.
- Virtualization leads in principle to a decreased maximum performance since everything has to be transferred through the virtualization layer. In a few cases this can lead to a performance increase i.e. different caching behaviour of the virtualization layer etc. but in general you should expect up to 5% loss (10% under really bad circumstances).

As a conclusion in Fortissimo we think virtualization is worthwhile since the advantages are significant, especially for the development and early production phase where the load will be limited or precise load prediction is not possible.

### 3.1.3 Network Layout

The simplest network structure is a flat one where all machines are connected to the same default network. All machines are then connected and can access each other directly. In Fortissimo, the Liferay-based deployment stages were separated into independent network segments realized with IP subnets that had no influence on each other even if a developer tried to do so, accidently or intentionally. This ensured that the production environment cannot be affected even if there was a total crash in the integration stage. This separation gave developers more freedom to test without adverse consequences. To achieve this separation we assigned a specific network segment to each stage. Connections were allowed only within a segment and to the centralised services. Communication between stages had to be routed and were configured only if needed, for example to allow content staging as done in Liferay when the instances directly connect to each other.

The centralised services (see sub-section 3.1.1) of course remain as single points of failure across all deployment stages. These services are however, simple and widely used installations (mail relay, DNS, proxies) and in Fortissimo we believe the risk is controllable. As a consequence, the network is split into different logical networks and access is restricted between these on the host. Additionally, we use mainly private IP addresses for the machines for the following reasons:

- It reduces the number of addresses used. There are not many IPv4 addresses left [10]. The most important point here is that the network segmentation causes a loss of many addresses hence the need for private ones.
- It provides more flexibility when changes are necessary since the private addresses are only visible within the whole development environment and not to the world, so only a few machines are affected by any changes. (i.e. hosts file instead of DNS + caches worldwide).
- It provides no accessibility from outside and thus reduces the security risks. Only machines and services for which external IP access is required are bound to external IP addresses directly or via proxies.

As shown in Figure 2 we created one network slice per deployment stage and another one for the centralized services and restricted any access between the stages.

Regarding the migration to Drupal, the hosting at Acquia can also be treated like the Liferay stages before, the main difference being the network addresses since these are external and publically available.

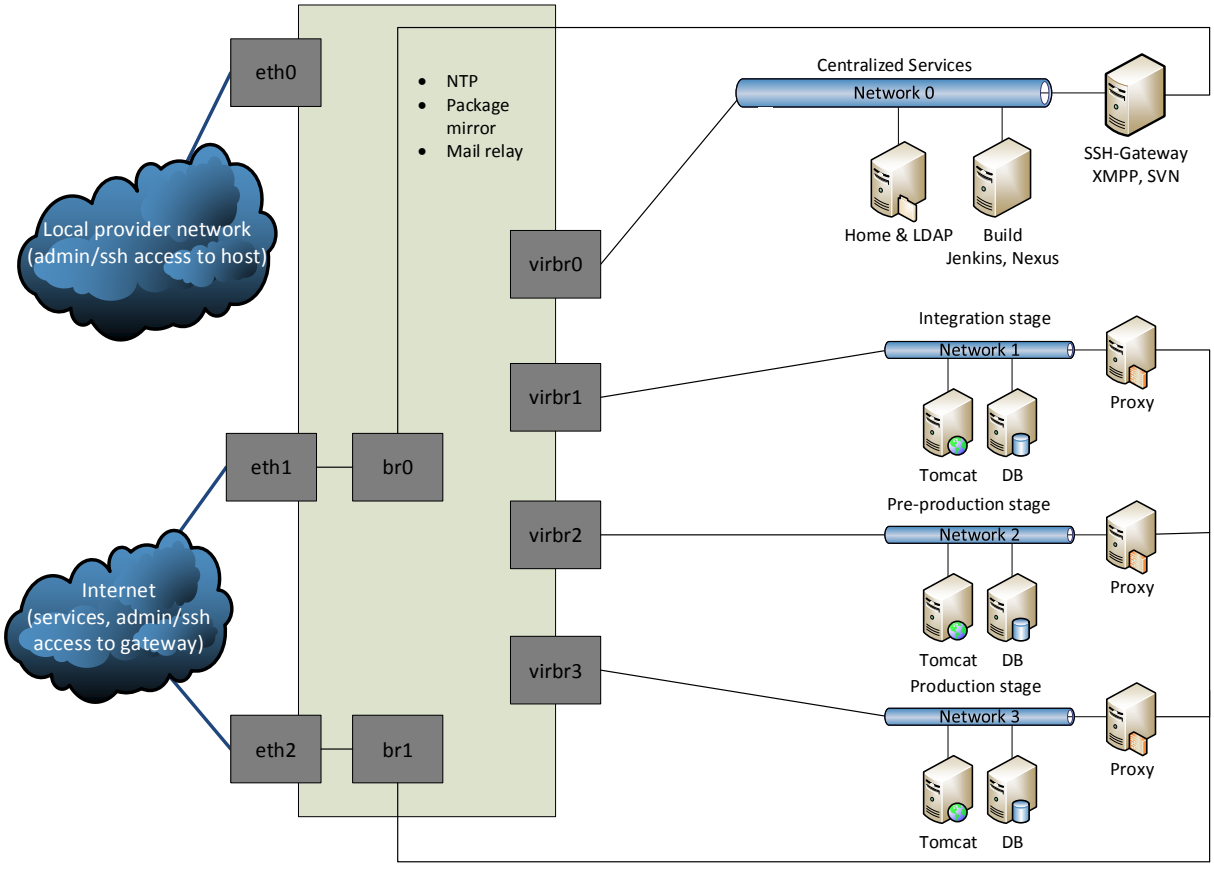


**Figure 2 The hosting environment network structure**

### 3.1.4 User Management

It is possible to have one centralized user management system used by all services with user interfaces (e.g. SSH [4]). The complexity however of storing all user related information in one system/schema and the configuration of all related services depends on the variety of different requirements from these services. In Fortissimo we had two main areas:

- The Marketplace with an arbitrary number of users, roles and rights managed primarily by the Marketplace platform (Liferay / Drupal). Connected to this the offered services using the same user accounts/credentials if possible by Single-Sign-On (SSO).
- The hosting environment where only Developers have access including build chain, server directly, code repository and more.

Putting everything in one user directory would cause some problems. Either, we manage the hosting rights in the Marketplace too but without gaining any advantage, or, we must manually mark or activate a specific user from the Marketplace to be allowed to access the hosting environment. In this second case, if the Marketplace has problems it would mean that nothing would be accessible! Some additional reasons why we use separate user databases for the development environment and the marketplace are.

- We wanted to grant all developers direct SSH access to the machines to be able to control the necessary services (database, application server). This also allows them to react quickly to problems.
- The Fortissimo services need to act without user interaction and so also need user accounts.
- For the Web portal that forms the basis of the Fortissimo HPC Cloud Marketplace we wanted to have only one database per deployment stage. Thus developers can use the same system with the same configuration as all other users in order to test their services.

In Fortissimo we therefore decided to have two different user management systems, one for shell based access including the automated deployment process and another for the Web portal. This might change in the future when we have a more advanced user management. The first is mostly done with standard Linux tools and settings (users, groups, rights) and the second is integrated within our enterprise portal software Liferay [11] and therefore is specific. It would be difficult to have one combined system supporting all required attributes and allow changes from both sides to get propagated.

For the shell based access and the OS (Linux) based issues we use LDAP (Lightweight directory access protocol) [12], [13] and all machines/services are linked to it. Changes are made directly in the main LDAP directory while nodes can run caches (for e.g. user/group numbers) of this to reduce the number of interactions.

### 3.1.5 Proxies, Mirrors and Caches

Whilst these services are not strictly necessary within Fortissimo, they are however helpful. Some of the following services have already been mentioned but they share a caching capability as a common functionality, and it is important to realize that this can help you when providing your services. The services that share a caching capability are:

- The OS repository mirror(s)
- The source code version controlled mirror (SVN/GIT)
- The code artefact storage (Nexus)
- The proxies we use for each stage to hide the application servers (Nginx [14])

In principle all these “caches” can be removed and the system will still work. The existence of these “caches”, however has the following benefits for Fortissimo:

- Shorter response time (Nexus)

- Avoid duplicate effort/load/traffic (Nexus, SVN/GIT)
- Less bandwidth usage, and independence from outside connections (OS repository, SVN/GIT)
- Flexibility (Filter/rewrite requests) (Nginx)
- Easier administration and extensibility (Nginx, OS repository when switching mirror or version, additional repositories)
- Reduced workload (Nginx in case a lot of static content is directly delivered)
- Help implement security considerations (Nginx)

In conclusion the introduction of caches in different areas can be beneficial depending on their actual usage.

### 3.1.6 Data integrity / Backup

Standard IT operational procedures and especially sustainable operations require the existence of a fault tolerant hardware and backups. For the server used to host the virtual machines we have a collection of RAID controlled discs. Using RAID level 6 ensures even a two disc failure does not cause any trouble since redundancy information is stored twice to enable recovery from all problems. In case of a total system damage - either RAID controller problems or fire or electrical damage – additional backups are needed. In principle a simple image backup (in our case VM images, otherwise system images) fulfils this requirement, since all data can be recovered. Nevertheless, a backup strategy should consider not only the capability to recover data but also the following:

- Minimize the amount of data backed up to avoid having to regularly delete old backups to free up storage.
- Allowing easy access to small pieces of the backed up data. If only a certain service / information gets lost, this will reduce the downtime. With image backups it is not easy to access just small parts of data.
- Being prepared for disaster recovery if large parts or the whole hosting system is out of operation.

In Fortissimo there are many different VMs (about 15 altogether) with only a few containing large data sets. Most contain rather small configuration directories thus we decided not to use image backups. Instead we back up important directories and have additional configuration documentation to allow recovery from most problems.

For all VMs concerned with the Web portal that forms the basis of the HPC Cloud Marketplace, we created custom scripts to backup only the portal and its related contents and code.

For the Fortissimo hosting environment, we rely on an externally maintained wiki where the installation process and further changes are documented. Recovery from small problems only requires finding the appropriate section in the documentation. In the case of disaster recovery all the steps described in the wiki are executed. Since we sometimes recreated the stages with content from another stage we used the backup mechanism we already had and simply added backup restore procedures that could also be used to real backup scenarios. This recovery process is not ready to run unattended, but in contrast to backup, the recovery is executed only in emergency cases and then normally attended. This also reduces the recovery time which is quite important once a significant set of users relies on the Marketplace.

In the worst case we need to follow the documentation in the wiki to setup again the VM hosting and all VMs, then copy the data back from the backup server. It is important to not host the backups on the same server or at least not only on the same server. Using a temporary local file is useful to reduce remote interactions.

### 3.1.7 Hosting for Drupal

After the decision to move to Drupal, we faced a lot of issues regarding the hosting and development. It was not clear how this would work since the change included a non-trivial shift from Java to PHP, so any development might be affected. Also the time limitations were quite challenging and our experience with Drupal was limited. So it was also decided to use the services of a web development company. They introduced the hosting at Acquia [3], a specialized hosting company for Drupal based on Amazon AWS [15]. After getting familiar with this platform we were quite confident that it fulfils our needs since it also used the three stages concept, extended by a fourth local stage for each developer. Additionally, Acquia had the whole staging cycle already realized and well documented.

The concept of Acquia also uses three different stages as we did before, for development, testing and production.

The concept of Acquia also uses three different stages as we did before, for development, testing and production.

- The development environment (DEV) is used for deploying and testing new components the first time on a production like environment and together with all the other components from the wider group of developers. Also new content types, setting etc. can be tested here in combination, since it is not possible for developers to do this in their local development environment. From time to time DEV is overwritten with the whole production environment to have an updated version similar to the one offered on production. Thus, DEV is at most times an old version of production with changes.
- The production environment (PROD) is the stage that is publically available and used by the end users. This also includes the most recent updates of user accounts, content, setting etc. The production environment is for performance reasons supported by caches and load-balancers.
- The staging environment (TEST/STAGE) can be used in different ways. Either for testing a new release with specific versions of all components, or a completely new development branch, or also as a mirror of production for debugging.

But there are some differences in the concept and in the configuration of the platform compared to our prior Liferay installation. First, they optionally offer an additional development path – Acquia Dev Desktop – where they provide a full software stack that provides a local Drupal installation for each developer. For Liferay this was done with an individual local setup causing different paths within a project. From this Acquia Dev Desktop code is submitted to a Git [16] versioning system that is provided by Acquia. So far, the differences to our previous installation are not that large. However, Acquia also uses the Git for management of the three stages.

In Drupal there are in principle three different pieces of data

- The Database containing all pages, users, settings (also for Drupal modules) and much more.
- The Code including the Drupal core components, all code for modules, basic Drupal setting (e.g. for the webserver, underlying security, database connection etc. that is

needed before Drupal can start). The code is managed by Git, so any change is tracked and any prior version can simply be restored.

- Files covering images used in pages, uploaded files, etc.

Acquia offers a web interface (Figure 3) where the site admin can easily manage all their stages. The idea behind this is that each of the three pieces can simply be copied to another stage on demand.

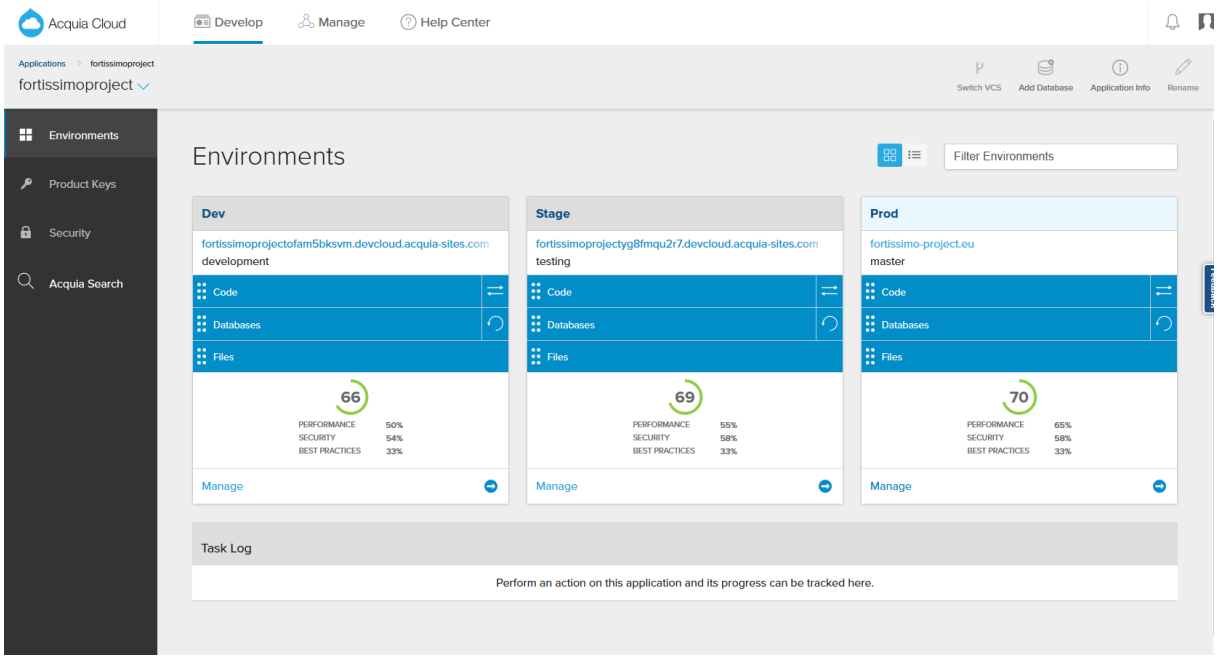


**Figure 3 Acquia Web interface**

To understand the following concepts the basic Git concepts are important. The versioning system supports tags and branches. Branches are used for a longer development phase where code is checked in several times and the HEAD revision of a branch includes always the latest changes. Branches are later on merged to the master branch if the development has finished. Tags are simply names for versions and are normally used to mark a specific milestone etc. (e.g. a new release, a daily named snapshot, …). The HEAD revision can be seen as a special tag since it also is a clearly defined version, but always the newest and so it changes always.

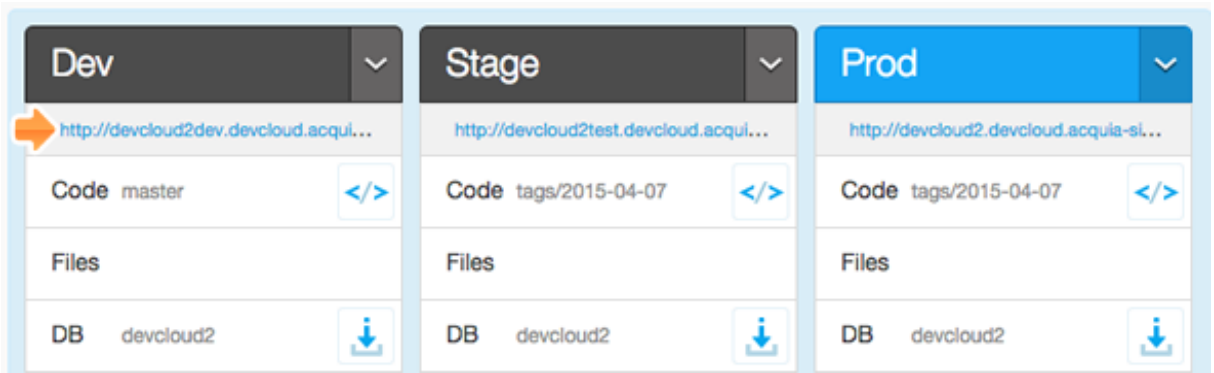


**Figure 4 The different stages and connected Git tags/branches (Acquia example picture)**

As suggested by Acquia the PROD environment should be a stable version of all developments and therefore be connected to a tagged version in the Git. The advantage to this is that under normal circumstances a tagged version will never be changed. Technically it can be changed but that is a special feature only to be used in special circumstances. Normally a new tag is created when the code for components has reached a specified maturity level, i.e. a

new release is published. The tagged version can simply be selected in the Web interface and the deployment is done in the background. For the Dev environment it is suggested to use a branch so that any committed code is directly deployed to this environment, both references to the git can be seen in Figure 4 as the "code" attribute.

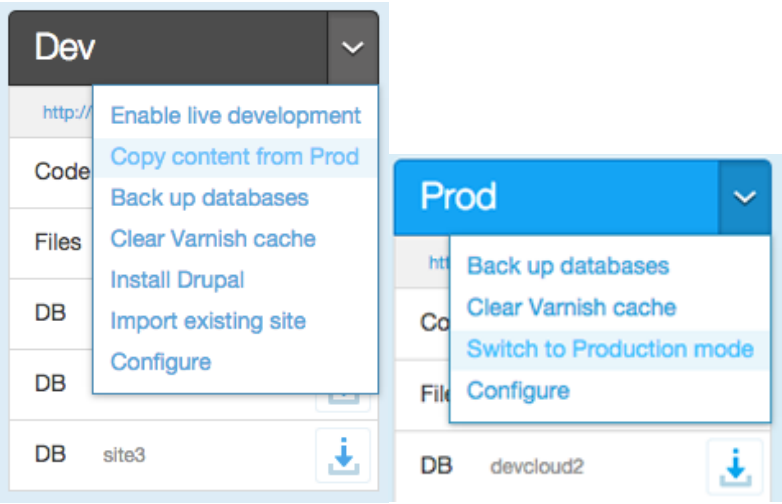


**Figure 5 Example pictures for DEV/PROD and actions**

Within the Web interface also snapshots from all three data pieces are possible either for backup or for local replication. All data parts can easily be moved by Drag & Drop to a different stage. For PROD there are additional features available (Figure 5) like a load balancer for secured connections or locking of the environment to prevent accidental changes in the web interface. For Dev also additional features are available like the live mode where changes is made directly on the server without Git. That is useful for debugging, but all changes made there need to be committed back to Git afterwards to not get lost.

So with the infrastructure provided by Acquia we have a powerful platform where we can easily collaborate and develop the necessary components for the marketplace.

## *3.2 Developer Support*

This subsection describes the activities undertaken within Fortissimo to help developers produce a sustainable HPC Cloud Marketplace and related services.

We start by describing the different stages in the first sub-section, followed by the automated build chain, supporting developers and in the end describe the most important aspect of how to migrate content to production.

### 3.2.1 Deployment Stages

The number of deployment stages you set up depends on your needs. In Fortissimo we use mainly three stages but more may be useful for particular applications, separate tests, different independent developer groups etc.

In Fortissimo we initially decided on the following three parallel deployment stages:

- The Integration stage for developers to work on the integration and tests of all functions and services of the Web portal that form the basis of the HPC Cloud Marketplaces.
- The preproduction stage for preparing
    - The next release of the production environment with data from the production stage,

  - The optimized Web portal look and feel,
  - User driven tests for the final checks before releasing,
  - Debugging in a copy of production stage without affecting production
- The production stage is the real production environment that should not be affected by the other stages. Security and performance issues needs to be addressed for this stage.

In Fortissimo we also considered adding another stage exclusively for automated testing. This stage would not be accessible by developers in order to prevent manual concurrent developer driven tests affecting it. Due to the switch to Drupal, however, and the Acquia hosting we stopped that. Instead Acquia introduced another stage partially, the local stage by providing the Acquia Dev Desktop application. Thus a local Drupal installation is available where the user develops his components and tests it. In Liferay this was also possible but more difficult due the high system requirements and long start up times. This fourth stage reduces the number of incomplete components in development since they have already been tested to some extent. This, thus leads to more successful builds.

### 3.2.2 Automating the Build Chain

It is possible to reduce the developer involvement in uploading code and reading test results by automating the workflow starting from the first commit to the code repository up to automatic deployment to any stage after a successful build and test. Basically the whole workflow can be described:

- A developer implements some code and commits the code to the repository. This does not necessarily indicate that the code is complete or even can be built successfully. So the real workflow starts when the developer think their code may run through the process successfully.
- The developer may upload additional test procedures.
- Depending on the configuration either the workflow needs to be triggered or it is executed regularly. The developer may also trigger it attached to a successful commit request.
- The build support system then fetches the code from the repository
- The component build process is initiated. If it finished without errors, the code artefact is then stored for later usage.
- The result artefact is then deployed within the build support system and tested against the provided test procedures.
- Again only on a successful test is the new artefact deployed on the integration stage for further testing.
- The developer or specific test person executes the defined test and human interactions. In case it fails, reports to the developer are generated.
- The artefact is marked as ready for the next stage according to the test result.
- This ensures that only proven code can reach the next stage.

In Fortissimo we implemented for Liferay the whole chain. In the end we were able to even test the database functionality directly by either accessing a dummy database or the integration stage database. Some of the components used the whole process, other reduced it

to a smaller set of actions. Mainly the separate artefact storage was not used that often since Jenkins stores the build itself already.

Regarding the switch to Drupal and therefore from Java to PHP we did not yet manage a similar process. Since PHP is used directly as source code, and not precompiled like Java, the build itself is not that critical as for Java where smaller errors cause the whole process to stop. Also we had different approaches to create code due to the short time we had for performing the migration. So build support was not that important.

### 3.2.3 Migration from Preproduction to Production

Any component developed should at some point be migrated from integration to production. Normally the process in the case of a release cycle and not a rolling release would include prior testing on preproduction (on Acquia this is named stage or testing). In Fortissimo this can only occur under the following conditions:

- The latest successfully tested versions of all services are deployed to the preproduction stage. Components failing some test are not ready and therefore excluded or held back.
- Data from the current production stage is available on the preproduction stage. To test the whole release under production conditions, page content and maybe user generated data from production needs to be used for realistic tests.
- The Web portal is set up with all required configurations performed based on scripts.
- Additional tests on the available production data are executed and the package is marked according to the test results.

Under these conditions successfully tested packages are ready for deployment to the production stage. This migration is complex and so administrator or developer interactions are needed to check intermediate results.

In Fortissimo we supported migration for Liferay with tasks being executed with scripts but intermediate reports and error logs needed to be evaluated manually to ensure a successful migration. For the new Drupal environment the migration is currently executed manually by either committing the source to the Git repository into the branch connected to the target stage. The deployment to the stage is then done automatically.

## *3.3 Marketplace*

The Marketplace is the publically available part of the whole environment and equals the production stage together with services located at the provider's sites. This may include basic services like the helpdesk or services directly related to HPC resources like simulation services.

In this section we first start with describing the basic Marketplace provisioning in sub-section 3.3.1. On top of this, the Marketplace user management (sub-section 3.3.2) is quite important since all resources providers need to be connected to it. In the third sub-section we describe which and how external resources can be connected to the Marketplace.

### 3.3.1 Marketplace provisioning

The production stage should get more attention regarding stability than the other stages since developers will have also problems when the stages are not working properly, but they are part of the Marketplace providers and can handle that somehow. The end users in contrast will be unsatisfied and may switch to another platform. Regarding downtimes and maintenance periods we had some bad experiences with Liferay, since the start up process is quite slow and

a restart is necessary for deploying new code. We improved that somehow by granting more resources for the production environment and for sure, reducing the number for restarts. But the platform was not that stable that we could do it without restart, we even needed to do it every few days. In the end we did it daily overnight, since also for backups we needed a shutdown to not backup directories including temporary changes that causes crashes if restored. For example, Liferay renamed files during start-up and renamed them back on shutdown. By backing up this change, Liferay would try a second time to rename it and fail. For the Drupal platform all these disadvantages are gone, since it directly uses the source code and interprets it, it does not need to load a lot of code in advance, the start-up times are fast. Also it is possible to replace files during operation since Drupal will reload the files immediately. Smaller changes can be done without interrupting operation.

In principle you should in detail check the requirements and limitation of the technology you are using (here i.e. JAVA class loading problems on start-up) and the concrete platform (Liferay include a huge number of classes by default) to not be surprised later on when you get into trouble. We thought that Liferay was a good decision but in the end we experienced the opposite. For others it might be completely different.

Regarding the Acquia hosting, the production environment is provided differently to the other stages in a way that the performance can massively be improved by using a proxy in front, in such way caching the static content and also covering the secure connection handling. This specialization leads to reduced load on the Drupal instance itself and therefore higher performance. The other stages are not equipped in the same way to the proxy and are therefore slower. Proxies and caches are more useful for a large number of users and cache updates are easier for rarely changing content. Both obviously fit better with a production environment like our Marketplace than to the development environment. For a large Marketplace with much user generated content this might be different due to the more difficult cache updates.

### 3.3.2 User Management

The user management is a crucial point for a Marketplace, especially for our Fortissimo Marketplace since many services are not completely included but need to use the same user accounts (or mapped user accounts). This means not everything is based on the same technology, nor the same network. This indicates that we need a central user management system that is populated by the Marketplace. Since most of the providers use LDAP for their resources and LDAP is widely used with support in many tools and applications (for user management, system information etc.) we decided to use LDAP. So we installed a separate LDAP virtual machine. This LDAP is then mirrored by the HPC providers in their local environment. This gives them much flexibility on how to connect their resources to the LDAP. The main advantages of this setup are:

- The providers can access all the user accounts and also adjust their LDAP if necessary to add local attributes or settings. So the need to adjust the local environment is reduced.
- Providers can also use mapping between the LDAP and their local user management. This may be needed to deal with conflicting naming schemes.
- LDAP is widely used and a huge community can provide support, most problems with LDAP can easily be solved since others have often had the same problems before.

- Extending our user management-related services is possible since many related tools support LDAP. This includes Single Sign On. (SSO). We were able to realize SSO with a CAS server (Centralized Authentication System).
- Flexible configuration options allow the use of LDAP in changing environments. It is not easy to configure it every time but it is adjustable to almost all requirements.

For Liferay we wanted to use the included LDAP connector, but it failed in several ways when we tried to use more than the minimal functionality. Many of the provided features did not work as expected and support was not available since the public community is quite small. After migration to Drupal we faced the question of whether to add an LDAP interface to Drupal. There is an LDAP module that provides LDAP server functionality. This means a close connection between Drupal and LDAP would be possible and would allow the chance to provide many attributes for the users. We decided however to keep the stand alone LDAP for several reasons.

- The uncertainty about the provided and possible user attributes.
- The uncertainty about the configuration options to access this LDAP.
- The unavailability in case of Marketplace problems.
- No need for larger changes (except changing the LDAP path) related to the HPC providers. All changes are covered by the Marketplace to LDAP connection.

We still are considering about migrating to this LDAP, but that is not final yet.

For services which do not need to map any users, especially new services from vendors in the Marketplace, these should be able to use the user account from the Marketplace directly and not need to host a synced LDAP server. Normally vendors should also not be able to access the user credentials, neither by forwarding the credential, not by accessing the hashed passwords. But we also want that the users do not need to login on every service again, they should stay logged in with their marketplace credential. This can be solved by using Single Sign On, one of the targeted features for the marketplace. In the end we installed a CAS server at the same machine as the LDAP directory itself, connected to it internally. The idea behind this is that services forward the user in a transparent way to the CAS server where the user logs in. After that the user is redirected back to the targeted services along with a special token. The server can identify the CAS server based on this token, the CAS is trusted and so the user account can access the services. Any further required additional attributes can be requested by the service directly from CAS on the basis of this token. That is, without user interactions and without knowing the credential. We configured for now the Marketplace and the helpdesk to support CAS, so only a single login is required. When a user is already logged in and is redirected to the CAS, the CAS server knows the user session and directly redirects the user back to the service without additional login.

### 3.3.3 Integration of External / Distributed Resources

In Fortissimo we want to allow users to access HPC resources offered by different providers and therefore different locations and substantially different configurations and access characteristics. So we need a way to integrate all of them so that the users can access these without high entry barriers. We want to solve this by mapping user names between the Marketplace and the provider user management systems without the user's knowledge, in effect providing the same user experience as for a true single sign-on setup.

So the basic problem is that the "remote" resources (all pre-existing HPC resources that should be made available through the platform and therefore not created especially for the

platform) have their own access requirements which normally cannot be changed easily. Small adjustments may be possible but changing the overall process is not possible due to the terms and policies of the involved providers. The only change enabling the platform to connect all providers and present them in similar ways is to hide some of the differences and try to harmonize the others.

We identified some problem areas, this list may not cover all issues but the major ones we identified:

- *User account management:* Users have different accounts for different providers due to existing accounts, different naming conventions, etc. So there is no possibility to match all the names. Instead user name mapping can solve many issues. For security reasons the mapping has to be done on the provider side, so that each provider has full control of changes. Otherwise the Fortissimo HPC Cloud Marketplace provider could change all user mappings for all providers.
- *The registration process:* For legal reasons users have to sign the terms and policies for each resource provider they want to use, unless the providers agree on a common version of the conditions for all. This is not possible. Instead separate policies for the HPC Cloud Marketplace can be used which cover most of the common issues and are aligned with the terms of the platform providers. When users want to access an HPC provider the first time, they have to agree to the local terms. Depending on the signed policies, the users can be offered preselected items in the Marketplace Web portal or status information for these systems.
- *Job submission and monitoring:* The large number of different job submission systems is a challenge regarding common access. For some providers we managed to hide the access details by using a common generic job submission system. Others made their own adjustments to provide their resources in the marketplace. The same holds true for job monitoring commands and interfaces.
- *Accounting:* A tricky part is that on the one hand data privacy has to be ensured while on the other the user should be able to see all his accounting data in one place to have a better overview and to be able to easily switch between providers and different accounting systems. We, therefore decided that the accounting data should be transferred to the Marketplace with mapped user names on a regular basis (daily if possible).

Since the differences between the various existing HPC resources provided in Fortissimo are very large, we decided to allow different possible implementations for the connection to the job submission systems:

- A generic approach using OpenStack [17] as a cloud based solution offering frontends for the users, where the user can work as on the default cluster frontends. A middleware layer checks for “submitted” jobs on the VM and forwards the jobs to the cluster based on the account mapping table and writes back status information for the user. This allows a very generic user interface that can be enriched by adding Web interfaces, specific application depending configuration forms, workflow support and much more. In our scenario the OpenStack is only loosely coupled to the cluster without direct access to the file systems. The disadvantage of this is increased transfer time for the data since it is temporarily stored in the OpenStack environment. For this approach we also built a generic job submission web frontend
- Direct integration of provider specific access portlets with the Marketplace. They should be aligned to give the user a common look and feel depending on the different

needed parameters. This tight integration allows perfect alignment of provider and portlet, but needs also the most effort since this must be done for each provider individually.

- Use existing portlets for commonly used middleware that is already supported by providers like Unicore [18]. Some modifications are probably necessary. Clear advantage is the existing usage base (the interface will be familiar to many users), but at the same time flexibility towards supporting new features may be strongly limited.

The decision on which possible solution should be selected does not only depend on functional arguments for the overall platform. The interests of each provider have to be respected to not overburden any of them. A mixture of the different solutions might be the best way to achieve a sustainable Cloud infrastructure even if then the user may see some differences and tend to get confused. This usability drawback has to be balanced against the high effort needed for creating and maintaining a truly "seamless" integration.

## 3.4 End User Interactions

The functionality of the platform is one side of the coin; the other is the end user as the most important actor in the Marketplace. After all, the user is the customer and pays for the services. Achieving user satisfaction is important to convince him to extend his stay or attract even more customers. To achieve this three aspects are important:

- The capabilities of the platform which need to cover the user's requirements.
- Well-structured and intuitive user interfaces that enable easy use of all functions and lower the entry barrier for new users.
- Different kinds of support facilities ranging from manuals, to question and answer sections and issue tracking systems for upcoming problems to match internal processes for error handling and escalation. These structures can also be used for operating and maintaining the systems.

Providing the underlying capabilities was already covered in the previous sections.

### 3.4.1 User Interfaces

The entry point for all users is the Marketplace Web portal and the provided module for the different providers and their resources. Thus the Marketplace is designed to be intuitive and additionally we provide documentation material on different topics so users (especially a beginner) can follow them for often used functionalities. Each possible field or button within the Marketplace as well as the structure / hierarchy and even the design needs to be planned well:

- *Menus:* Which type and location fit the planned number of menu entries? Are they grouped in a way they can easily be found? A footer menu is only feasible for items that should be presented independent from the specific currently shown page (i.e. legal or contact information). The menu decisions depend on the number of items and their grouping and also the content that will be displayed, for some a left side menu would be much better that a top menu and vice versa. Are all important features available in the menus and unimportant ones removed?
- *Naming/Topic*. Is the text or name of a specific item specified to avoid misunderstandings, does it contain enough information for the users? Synonyms and similar words should not be used, also the naming should match the user's vocabulary and not be too technical. Is the naming of fields descriptive enough or is it necessary

to add some information (i.e. tooltip text or similar)?

- *Pages*. Is the style of all pages the same? Arbitrary differences could cause irritation. Is the quality of all pages similar? Badly written content is not interesting for users, since they need more time to understand it.

Especially for the names of menu items and other functional elements we had several discussion since we already had different view and understanding of some words. For an arbitrary user this might be different in another direction.

For the permanent development and new version of components there are additional recommendations to minimize confusion for the users and support sustainability:

- Keep all design elements including menus untouched if possible. At least try to avoid moving or renaming of menu items. If it is really necessary to rename something, this is probably caused by changing content or the intention to change it, then you should also change it immediately thus the user can detect the change even if the planned functionality is not yet fully available. The other possibility is to mark new features for a certain time to allow users to notice and get familiar with them quicker.
- Find possible documentation pages where the new content can be referenced and create the proper links. Make sure that old documentation is updated – e.g. by a description why a certain feature is not yet available, or how an interface element has changed. For this, we aligned quality checks for code and documentation, so that the updated documentation is deployed at the same time.

### 3.4.2 User Support

Inexperienced users especially are usually not able to intuitively use all Marketplace capabilities, and they need support in learning and understanding the system. One can expect that there will be different situations where the users will need help – properly combining different tools might be a better way than trying to produce perfectly matching point solutions:

- FAQ section. This can be done in different ways. The easiest is a collection of pages for different categories of questions (general questions, technical problems, accounting, …) or you may have multi-level selections based on different drop down lists or search patterns (here the number of FAQ items can be much higher). Depending on how deeply users participate in helping each other forums might be an alternative, where some users are allowed to create new topics and other users can comment on the proposed solution. FAQs are very useful if the question can be answered explicitly. The answers should normally be short and cover only the content of one question, otherwise it should link to other questions. On the technical side error messages that appear quite often and have a clear solution that is already proven can be presented easily in a FAQ. Growing the FAQ material over time to cover the vast majority of user requests can significantly limit the support effort needed and thereby contribute to sustainability of the Marketplace in face of growing number of end users. The option to create several specific sections is also important to limit the displayed questions for specific sections.
- How-to sections. Especially for beginners some hands-on explanations for standard issues are useful. This strongly depends on the intended usage of the Marketplace and the experience of the users. For job submission for example we created documentation for each provider even in case they use the same access method to cover also smaller differences.

- For our HPC Cloud Marketplace a how-to would be helpful for the registration process until the user can have access to the real resources or how he can submit a job from the data upload to validate the results.
- Guided tour. Getting in touch with the main Marketplace features with additional context information and description. The tour can be realized in different ways, using a video is the most common and also very simple way. More beneficial would be an integrated tour that points to the specific item with explanations, and is interactive, but that causes also the most effort.

Within Fortissimo we created a FAQ that will be extended beyond the project duration. Additionally, for beginners and for typical workflows we added manuals and step by step instructions, since it can be hard to understand complex interactions by simple questions/answers alone.

The previous parts can partially be prepared in advance, with developers and support staff foreseeing typical user needs. Such documentation will support users immediately at the start of a new service.

Also many cases exist where the user cannot solve the situation on their own; not with any information provided within the Marketplace. Not even by searching the internet for a solution. Using search is not the suggested way for users, but some of them might do it before contacting the helpdesk. For direct user interactions a bug/issue/request tracking system should be used if you expect more than a few actions in parallel. Otherwise you get confused by too many requests or some get lost, which has to be avoided.

We introduced regular shifts (weekly) with fixed persons per shift for a long period in advance. Over time we have seen that the weekly periods are very short and the learning overhead for each shift is too high. So we extended the shifts to biweekly and made very good experience with that.

For larger installations and depending on the number of resource providers the support needs to be separated into different levels and maybe groups. The first level support always comes in touch with the user requests first and tries to solve them (which should be possible in all major and often occurring situations). A first level supporter does not need to have strong experience with the system, as most of the requests should be solved either based on the public user documentation or with an extended version for internal usage only. If the variety of support requests covers several domains, groups for the different topics need to be formed, i.e. for financial problems a separate team might be necessary that has full access to accounting. With difficult problems normally the second level of support comes into play, which consists of experienced trouble-shooters that know the system very well. They should be familiar with the specific implementation of the Marketplace and the known problems and need to have more rights on the system to track down problems. If even these people can't solve the problem, the last small group of real experts (level three) will deal with the problem. They are normally operating the system and have full admin rights or are experts in a specific domain. They should be able to solve all problems or at least find the cause and a possible solution even if this solution is only an individual fix or something needs to be changed in an unusual and not normally attempted way.

Depending on the situation you can offer additional capabilities:

- Telephone support (depending on the type of problem this is similar to a tracking system and normally the call centre agent adds the request to the tracking system directly).

- Mailing lists (only with internal or even with interested people to include in the community). This is widely used in open source software since they often don't have professional support and a wide user community can participate.
- Consultation capabilities either to give lectures for groups or help on individual implementations.
  For an HPC Cloud this might be useful for integrating new ISV providers since they need to adjust their code in some cases or to any other user group that wants to use the platform for creating their own services.

# 4 Documentation and Backup

The previously mentioned actions are sufficient to install and maintain a HPC cloud Marketplace in principle. However, over time errors will occur that affect or even interrupt the Marketplace availability and functionality. To provide a Marketplace in a sustainable way documentation is essential. A short list of reasons describes why documentation and backup are important. Both areas are connected by the possible loss of information.

- When developers leave, the knowledge loss cannot be compensated since newcomers depend on experienced developers to get objective information. No reliable information is available (documentation is persistent and can be handed over to new developers).
- Same for admins but since this group is the smallest and most experienced normally the impact will be stronger.
- Changes over time are not known to all developers or admins and therefore the knowledge differs and even might result in conflicting responses.
- Hardware failures with data loss (configuration/applications, not user data) probably lead to a different setup of the new environment.
- Recovery after any incident with data loss is difficult to impossible and in case of system configuration may last a long time during which the marketplace is non-operational.
- The list can be continued but we think it is obvious that documentation is needed and it is important to have the right amount and well selected documentation. Too much documentation is hard to maintain.

## *4.1 Documentation*

We suggest different types of content to be included in the overall documentation:

- The above mentioned user related documentation (FAQ, how-to …) which has been detailed previously. In most cases this information is part of the content managed by the CMS used for the marketplace. A separate storing in the documentation is then not necessary.
- List of used software and most important the current settings of all configuration options. For large configurations and those including additional code or plugins this should be part of the backup strategy, for small information pieces this can also be handled in other forms (files, wiki, …).
- Installation process as a step-by-step guide ideally. Recovery can then be done quite quickly. The major problem without such a detailed guide is that many smaller configuration options are not obvious and hard to be resolved if they are skipped. So this is a crucial issue to speed up recovery or also introduction new participants
- Development Process. Write down the overall process as a guide for all developers on how they should develop and test their code and how they can use the development support tools for this.

  In case using Jenkins and Nexus the backup of tested and validated packages can be done automatically by fetching the nexus content.

The mentioned content types differ in their size, structure and lifetime or validity period. Different documentation formats and systems storage system therefore should be used for this.

- For long living documents like the how-to and the development process etc., an Office or better a PDF document would be appropriate.
- For temporary documents (i.e. creating or updating long living documents) a flexible solution might be a better way in particular when several people are participating in writing the document.

  We used for this Google Docs but any similar solution is also possible.
- For changing documentation (network settings or during setup) a wiki based approach might be useful, where many people can edit the documentation.
- For sure for the configuration files it can be very important to have a history to be able to go back in time if something goes wrong without the need to fetch this information from a backup system. So a version control system for configurations is advisable.

Overall good documentation can prevent many problems while offering additional capabilities, not just for the user directly but also for the development of the platform:

- Lowering the entry barrier for new developers/admins and also users, depending on the type and details of documentation.
- Upcoming problems could be solved by comparing the symptoms and the documentation. Maybe different client settings cause trouble.
- Suddenly emerging problems may be solved easily by comparing recent changes.
- Full and well-structured documentation may help in case of disaster recovery or to reduce the backup overhead (by not storing the complete configuration of, e. g. network/mail/DNS and repeating the initial setup process for recovery).
- If the documentation also contains information about tested and failed settings (in particular those with dramatic consequences), similar problems occurring in the future may be circumvented.
- Risks are reduced dramatically.

## 4.2 Backup

Aligned with the documentation the backup needs to be configured. It does not make any sense to have one server where the production environment is stored, all the documentation and, if available, local zipped backups. In the case of an electric discharge or fire everything is then lost. Thus we need a backup strategy and a detailed backup procedure that strongly depends on your overall setup and requirements.

We had in Fortissimo several independent services where information was stored.

- Project management solution where persons, effort, documents, work plans etc. are stored.
- A wiki where most of the short term documentation was done and also everything that was not part of deliverables like protocols, checklists etc.
- The hosting environment itself with many services and complex data collections including configuration and the marketplace

- Explicit documentation within the marketplace like the FAQ, Guidance documents and more.

Since parts of these were stored at different locations and responsibilities, an overall backup was not feasible. Instead we defined our backup strategy to reduce backup overhead while preventing critical damages. This caused increasing effort in case of lost/damaged data.

- No full backup of any virtual machine or server. Most of the data within a virtual machine is software out of the distribution repository and can be reinstalled quite fast. The immanent effort for offering and maintaining large backup volumes (for full backups) needs to be compared with the effort for reinstalling the basic operating system and restoring smaller data sets from backup combined with the probability of a related heavy damage. In most cases smaller restore operations are enough. For Fortissimo we decided to use only partial backups.
- Partial backup of system configuration limited to the core application for providing the Marketplace or the development process. The basic system configuration with a high number of configuration files was covered by the setup documentation in the wiki. So instead of using the effort for including many different files in a backup and restore process, we simply documented the setup quite well. Even if the time is the same, the documentation can be used for many different issues related to debugging, the backup files not.
- Full backup of the basic portal software folder to include all portlets/modules, configuration of the main active software pieces
- For the LDAP, a full dump plus a copy of the database files. Here it is obvious that the data volume is very limited and caused no problems while successful recovery is the most important part in here, since otherwise the data is completely lost. For most other parts unsuccessful recovery would cause more work. The two different backup were created to have a readable version for debugging and manual recovery and the binary version to have s simplified recovery that can be automated.
- Remote storage (at least other server) to prevent the unlikely case of electric discharge or hardware controller failure.

Starting with this strategy we created a daily backup running overnight when most likely nobody uses the marketplace.

- Backup initiated by Cron scheduling daemon [19] at the central server.
- Each server is consecutively visited and a specific backup script for this server of type of server (i.e. same service in all stages) is executed
- For specific services like databases a dump is created with a specific command
- Depending on the data the services is stopped shortly for getting consistent data on disc and started after the data dump.
- The files are backed up. For many small files a collective zipped file is generated locally and then copied to the remote backup server. For others the file is generated directly remotely to reduce disc usage and reduce execution time.
- Along with the backup progress, log information is collected and appended to a temporary file. At the end of each server task this file is also transferred to the backup server.
- At the end of the global loop, the main log is stored at the backup server and also sent

by mail to the admin for daily checks. This is mainly for errors in case something fails or to be informed about space left on the backup device.

- The whole process takes about 15 minutes for all machines.

# 5 Concluding Remarks

Many projects try to reach a solution as quickly as possible or at least skip some aspects in the beginning, believing that these can be picked up later. In most cases this is either difficult or even impossible, or these aspects are just forgotten. Even when you achieve it you will have delays later to implement it which is not completely anticipated Also if the overall architecture has been designed focusing only on a part of the overall projected solution, it cannot be extended later on or only with high effort. The conclusion is that:

**Preparation is half the way to success**

During the preparation you should also think about ideas and possible functions that are not yet on the plan, but to assess their implications on the architecture. Maybe with more or less the same effort you can create an architecture capable of adding the features later quite easily.

**Think about it before implementing**

We would also suggest not to create structures too complex, since they are difficult to debug and analyse if problems occur and also extensions might be complicated by the structure itself.

**Keep a clear structure for architecture and processes**

Creating documentation is annoying but it is the best way to recover and solve many problems and to allow new users to start with the platform/Marketplace.

**Write down any actions and changes**

In combination with the documentation you should create or ensure backup for all aspects and all possible threats ranging from human mistakes (changing a file or deleting one accidently) to disaster recovery if the infrastructure is completely destroyed. By combining different strategies and tools the effort for backup can be reduced dramatically.

**Create an intelligent backup concept**

In case when you experience more and more trouble with a specific component increasing over time preventing you from progress, think early about alternatives and possible consequences. We were confident we could handle the Liferay problems and decided very late on to switch to Drupal, but we do not regret it even if it caused a lot of additional work.

**Make decisions even if they seem to be hard**

# 6 References and Applicable Documents

[1] “Fortissimo Project,” [Online]. Available: http://fortissimo-project.eu/.

[2] “Apache Tomcat,” 14 12 2016. [Online]. Available: http://tomcat.apache.org/.

[3] “Acquia,” [Online]. Available: https://www.acquia.com/.

[4] “Secure Shell,” [Online]. Available: http://en.wikipedia.org/wiki/Secure_Shell.

[5] “OpenSSH,” [Online]. Available: http://en.wikipedia.org/wiki/Secure_Shell.

[6] “Subversion,” [Online]. Available: http://subversion.apache.org/.

[7] “Jenkins - An extensible open source continuous integration server,” [Online]. Available: https://jenkins-ci.org/.

[8] “Nexus - Artefact Repository Manager,” [Online]. Available: http://www.sonatype.org/nexus/.

[9] “Jabber,” [Online]. Available: http://www.jabber.org/.

[10] “IPv4 address exhaustion,” [Online]. Available: http://en.wikipedia.org/wiki/IPv4_address_exhaustion.

[11] “Liferay Portal,” [Online]. Available: http://www.liferay.com/.

[12] “LDAP,” [Online]. Available: http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol.

[13] “OpenLDAP,” [Online]. Available: http://www.openldap.org/.

[14] “Nginx - HTTP and proxy server,” [Online]. Available: http://nginx.org/.

[15] “Amazon web services,” [Online]. Available: http://aws.amazon.com/.

[16] “GIT - Version control system,” [Online]. Available: https://en.wikipedia.org/wiki/Git.

[17] “Openstack - Open Source Cloud Computing Software,” [Online]. Available: https://www.openstack.org/.

[18] “Unicore,” [Online]. Available: https://www.unicore.eu/.

[19] “Cron,” [Online]. Available: https://en.wikipedia.org/wiki/Cron.