European Commission

Directorate General for Communications Networks, Content and Technology
Sustainable and Secure Society - Health and Well-being

H2020 PHC-30-2015689617
Research and Innovation Action

Work Package: WP2

Infrastructure

Deliverable: 2.2

Infrastructure Design Recommendations

Version:1v2

Date: 31-May-16

DOCUMENT INFORMATION

Project Num	H2020 PHC-30-2015 689617	**Acronym**	EurValve
Full title	Personalised Decision Support for Heart Valve Disease		
Project URL	http://www.eurvalve.eu		
EU Project officer	Carmen LAPLAZA SANTOS (CNECT/H/01)		

Work package	**Number**	2	**Title**	Infrastructure
Deliverable	**Number**	2.2	**Title**	Infrastructure Design Recommendations

Date of delivery	**Contractual**	31-May-16	**Actual**	31-May-16
Status	Version 1v2		Final ☑	
Nature	Prototype ☐ Report ☑ Dissemination ☐ Other ☐			
Dissemination Level	Public (PU) ☑ Consortium(CO) ☐	Restricted to other Programme Participants (PP) ☐ Restricted to specified group (RE) ☐		

Authors (Partner)	M Bubak (CYFRONET)			
Responsible Author	M Bubak		**Email**	bubak@agh.edu.pl
	Partner	CYFRONET	**Phone**	+48 32 833 56

Abstract (for dissemination)	The main contribution of the WP2 will be a flexible and easy to use research computing and clinical infrastructures. On the basis of currently available requirements as well as on analysis of state-of-the-art in this area, we propose an architecture and detailed design recommendations for the research environments consisting of the File Store, Model Execution Environment, Real-Time Visualization, and Security Systems. We assess the applicability of available technologies for implementation, and provide recommendations for high-quality software development. We also describe the design of interfaces between the research and clinical environments.
Keywords	Large-scale simulations, files system, execution environments, visualisation, security, design recommendations, HPC, cloud computing

Version Log			
Issue Date	**Version**	**Author**	**Change**
09-May-2016	0v1	Marian Bubak	Initial draft
09-May-2016	0v2	Marek Kasztelnik	Initial requirements tables created. Content for section 6.1.3 (Containers) added. Description of AWS Lambda
09-May-2016	0v3	Tomasz Gubała	Content for 6.1.1 section.
10-May-2016	0v4	Marek Kasztelnik	References extended and moved from footnotes into References section for 6.1.3 section. Docker architecture figure description extended.
10-May-2016	0v5	Maciej Malawski	Section 8: interfaces and technology matrix.
10-May-2016	0v6	Tomasz Gubała	Contribution to sections 6.2 and 6.3.
10-May-2016	0v7	Piotr Nowakowski	Content for sections 3 and 9.
10-May-2016	0v8	Tomasz Bartyński	Added section about cloud computing.
10-May-2016	0v9	Marek Kasztelnik	Add application characteristics into 6.2. Recommendations for container usage, technologies recommendation for containers (sections 6.2 and 6.3).
10-May-2016	0v10	Maciej Malawski	Section 9: diagram
10-May-2016	0v11	Jan Meizner	Content for section 6 (Security)
10-May-2016	0v12	Daniel Harężlak	Partial content for sections 4 and 7
11-May-2016	0v13	Tomasz Bartyński	Added design recommendations for cloud services.
11-May-2016	0v14	Piotr Nowakowski	Minor updates
11-May-2016	0v15	Daniel Harężlak	Updated references and section 4
12-May-2016	0v16	Daniel Harężlak	Updated section 7
12-May-2016	0v17	Maciej Malawski	Updates to section 9
14-May-2016	0v18	Piotr Nowakowski	Executive Summary, Introduction and Summary sections filled in
17-May-2016	0v19	Marian Bubak	Abstract, keyword, and minor updates of sections
23-May-2016	0v20	Tomasz Bartyński	Applied remarks from internal review in sections on cloud services
24-May-2016	0v21	Marek Kasztelnik	Remarks from internal review in section 2 and in all sections connected with container technologies applied
24-May-2016	0v22	Tomasz Gubała	Applied remarks from internal review in HPC sections
24-May-2016	0v23	Piotr Nowakowski	Applied remarks from internal review in introduction, overview, software development/QA and summary sections
25-May-2016	0v24	Daniel Harężlak	Added risk paragraphs to sections 4 and 7
25-May-2016	0v25	Daniel Harężlak	Applied minor changes in sections 4 and 7
25-May-2016	0v26	Jan Meizner	Minor updates to section 6
25-May-2016	0v27	Maciej Malawski	Applied proofreading comments to section 8
29-May-2016	0v28	Piotr Nowakowski	Final proofreading
30-May-2016	1v0	PMO	Release candidate
31-May-2016	1v1	PMO	Minor corrections
31-May-2016	1v2	Bubak/PMO	Final corrections

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

The goal of this document is to provide an introduction to the computational infrastructure which will be developed and deployed in the EurValve project. We begin with a discussion of the requirements facing the proposed system, and then provide an overview of technical solutions which can be applied in addressing these requirements, as well as the design of a platform which will implement the required functionality.

This document is structured as follows:

- Section 1 presents the basic characteristics of the computational environment to be developed in EurValve.
- Section 2 provides a list of requirements which the proposed infrastructure must fulfil. These requirements have been gathered from the platform's prospective users.
- Section 3 presents a high-level vision of how the infrastructure is expected to operate, and enumerates some basic building blocks of the infrastructure, along with their intended functionality.
- Section 4 provides information on the design of data warehousing and management components of the infrastructure.
- Section 5 details the Model Execution Environment – a core element of the platform which enables execution of digital patient models using a variety of computational resources.
- Section 6 provides an in-depth introduction to security aspects involved in designing and deploying the EurValve infrastructure, and presents the corresponding design recommendations.
- Section 7 discusses how the proposed infrastructure can visualise results obtained by the Model Execution Environment and stored in the data warehouse.
- Section 8 provides a more in-depth presentation of the design of research and clinical environments, along with their interfaces.
- Section 9 presents the QA policies and tools which will be applied in developing the components introduced in the preceding sections.
- Section 10 summarises the document with a recapitulation of its most important aspects.

The document provides a starting point for the implementation of prototype EurValve software components, which will be reported in a later deliverable to be produced by WP2 in Month 15 of the Project.

The infrastructure architecture and design will be updated as new requirements come from other Work Packages (especially WP3 and WP5) and currently available requirements are refined.

1 INTRODUCTION

The goal of Work Package 2 is to elaborate and operate a flexible, easy to use environment for the development, deployment and execution of large-scale simulations required for learning process development and for sensitivity analyses (a Model Execution Environment, hereafter referred to as MEE), and for the associated data storage. The simulations will produce decision rulesets that will be transferred from the research infrastructure to the Decision Support System (DSS), while built-in visualisation facilities will enable access to cloud services from the local workstation.

The simulations will be run on external federated compute and storage resources exploiting recent advances in distributed computing, especially in cloud and container technologies. WP2 will closely collaborate with WP3 in the optimisation of their software modules.

The specific objectives are:

- To provide and develop the necessary infrastructure to collect, represent, annotate and publish core data; give secure access to the participating clinical centres and development partners to the necessary data, and execute the models in the most appropriate computational environment, supporting real-time multiscale visualisation.
- To develop an integrated security solution, supporting authentication, authorisation and data encryption for secure processing in public clouds.
- To deploy and operate the infrastructure, ensuring quality of software components and quality of service (i.e. aspects such as availability, responsiveness and cost efficiency).

The requirements, recommendations and suggested technologies listed in this document are the result of a series of discussions between Project participants – particularly the technical partners responsible for development and operation of the EurValve platform, providers of computational tools and algorithms which will be plugged into the platform, and the final consumers of data produced by the Decision Support System.

2 REQUIREMENTS

The following is a list of user requirements, defined in EurValve Data Transactions Matrix and Tasks requirements surveys, which have impact on design of the tools created in the scope of WP2.

Req. no.	Description
1	Execution environment delivers access to commercial numerical computing environments such as Matlab and Ansys CFD solvers (it will be used to deliver OD system model (T3.3) and ROM creation of 4D CFD (T3.6))
2	Execution environment delivers access to significant computer power and storage (ROM creation requires a lot of computer power and storage, >1000 CPU, many TB of HDD)
3	Some of the computations will be executed outside the research environment (e.g. Segmentation – T3.2 – will be run locally with node-locked license)
4	Data is transferred from data warehouse to compute environment
5	Sensitivity analyses are visualised while the computation takes place
6	The environment should support interactive and batch processing
7	Interaction with selected services should be automated by scripting or Web UI interfaces

3 VISION OF RESEARCH AND CLINICAL EXECUTION ENVIRONMENTS

According to the Description of Work, the goal of the EurValve project is to combine a set of complex modelling tools to deliver a workflow which will permit the evaluation of medical prospects and outlook for individual patients presenting with cardiovascular symptoms suggesting valvular heart disease (VHD).

First and foremost, it should be noted that the project must fulfil a set of requirements which may, at first glance, appear contradictory. While simulating surgical interventions (such as valve replacement), their outcomes and the patient's long-term outlook involves a set of highly complex algorithms designed to run in HPC (high-performance computing) environments, processing vast quantities of data and often taking a substantial amount of time to complete, the doctor who evaluates the patient at a clinic will typically have no access to such infrastructures and will not be able to marshal and dispatch the required data for processing at an external computing facility. What is more, the decision on whether to pursue surgical intervention (and if so – in what manner) must be taken in a timely fashion, leaving no time for complex HPC simulation runs, even if the required resources could be procured. Accordingly, the EurValve project must be able to reconcile the following goals:

- Accurate simulations of the outcome of surgical interventions (in this case – different treatment strategies for aortic and mitral valve diseases)
- Providing a decision support system (DSS) which can be applied in clinical practice, deliver timely results and be usable with no in-depth knowledge in the area of high performance computing or manipulation of large datasets.

In light of the above, the vision pursued by Work Package 2 is based on splitting the proposed system into two distinct components, each of which will be designed and built separately, only interacting with the other component by means of producing and consuming suitable input data, and sharing a common security/visualisation scheme. This is schematically depicted in Figure 1.

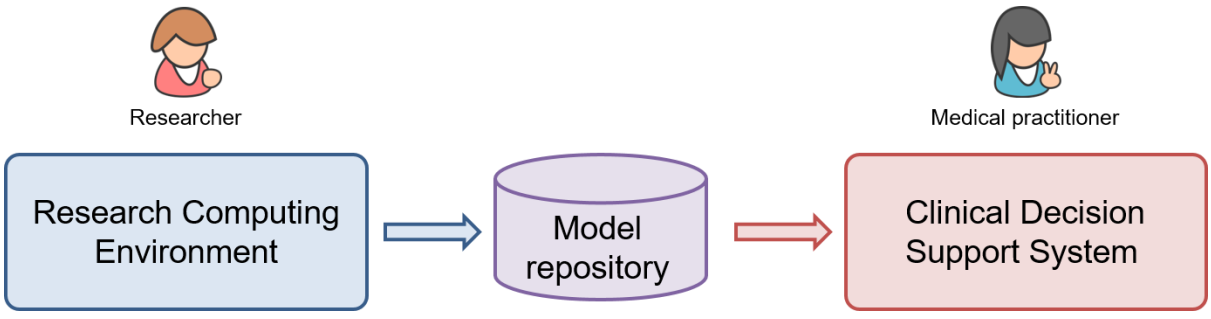

Figure 1: Conceptual view of the EurValve computational environment

In order to provide a repository of actionable models which would feed into the projected clinical decision support system (the so-called Reduced Order Models), EurValve intends to perform in-depth processing of a number of patient cases deemed representative for the VHD spectrum. This is a computationally heavy task which will be handled by HPC resources contributed to the Project by technical partners, most notably ACC CYFRONET AGH. In order to process each case, a number of steps is required:

- Medical scans describing a patient must be fed into a segmentation tool which creates mesh models with sufficient resolution for subsequent volumetric meshing.
- Tissue samples that are gained during surgery provide proteomic description of the cardiomyocyte. Models that use such patient-specific proteomics as input parameters allow the simulation of myocyte contractility in the context of the physiology of a given patient. This information can be integrated back to the models at an organ level.
- Computational optimisation with mechanistic models derives values of parameters needed to build a predictive description of heart load. Machine learning will reproduce the model values derived in this way.
- The above simulations produce input data which is then processed by CFD simulations (using the ANSYS software package), in a variety of modes (0D, 3D, Reduced Order Model).
- A sensitivity analysis tool is being developed by Philips to assess the sensitivity of simulation outcomes to specific input parameters, and to enable the system to narrow down the parameter space in search of variables which have particularly profound impact on the outcome of a given case.
- The end result is a set of parameterised models which will be stored in the model repository as the primary output produced by the Research Computing Environment.

The above procedure is schematically depicted in Figure 2.

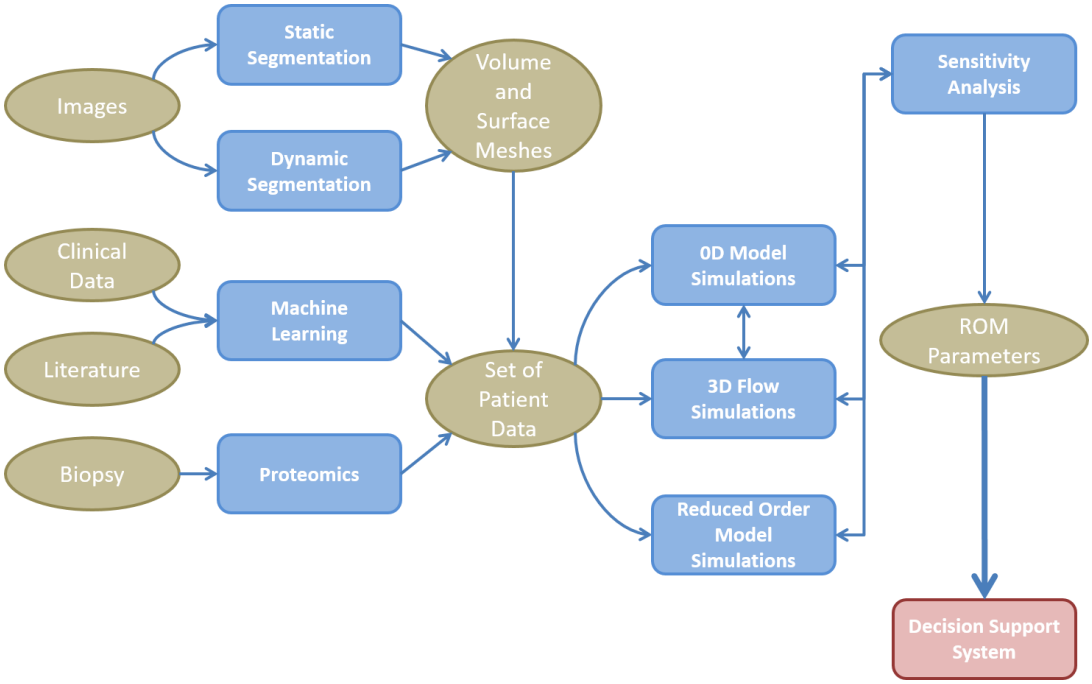

Figure 2: Generation of Reduced Order Models for processing by the Clinical Decision Support System

Having a repository of 0D/3D/ROMs in hand enables the Decision Support System to compute predictions for a given patient without repeating all of the above-mentioned steps. The DSS will work by processing a reduced set of patient-specific parameters (describing their cardiovascular system as well as ancillary data such as age, blood pressure etc.) in order to match them to one of the available models. This matching can be performed on a standard workstation, requiring no significant investment in computational resources at participating

medical institutions, and would provide a starting point for the clinical assessment of a patient as well as the prognosis of various forms of interventional treatment.

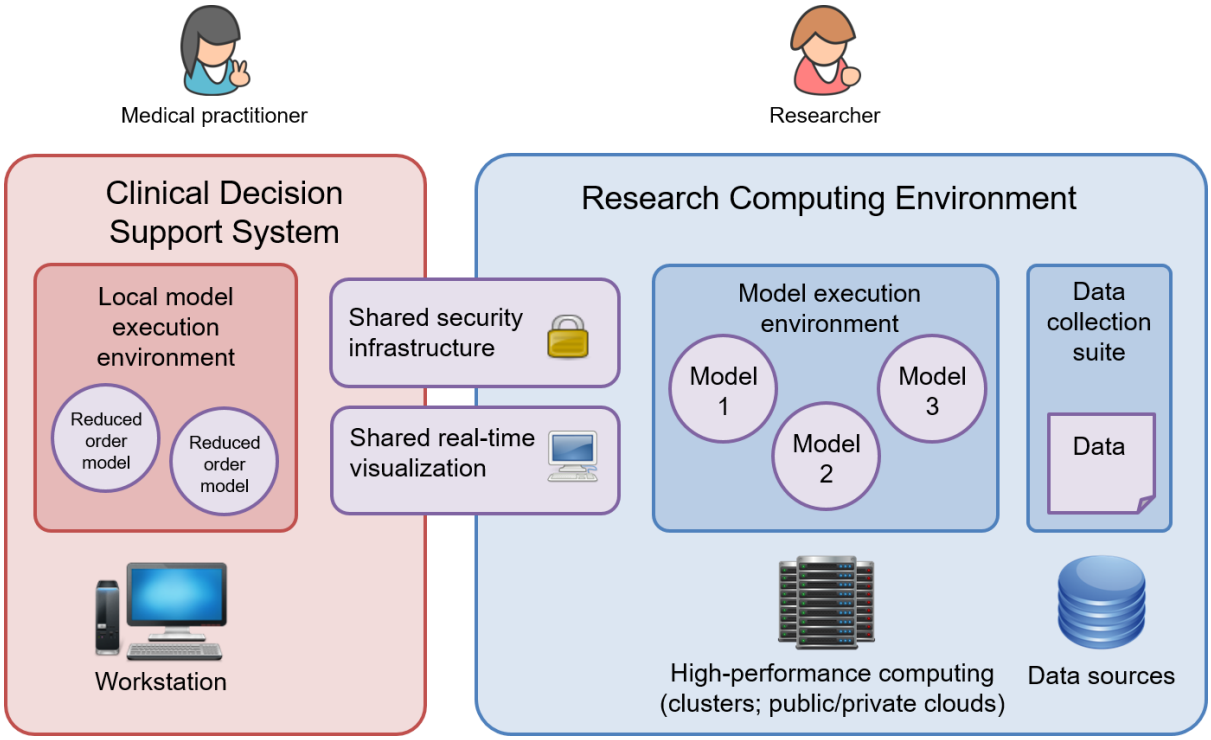

Figure 3: General overview of the computational architecture of EurValve

Figure 3 provides a generalised overview of the division of the EurValve computational architecture into two distinct subsystems, as well as a depiction of the interplay between both subsystems. As can be seen, the generation of models will be the task of the Research Computing Environment which is meant to provide access to HPC resources, including public/private computational clouds as well as HPC infrastructures. Subsequent sections of the document will provide more detailed descriptions on the intended functionality and preliminary design of each component presented in this introduction.

4 DATA WAREHOUSE AND DATA COLLECTION AND PUBLICATION SUITE

This section presents a description of the envisioned data management and presentation mechanisms. A thorough description of the Data Warehouse component can be found in Deliverable 2.1 of the Project.

4.1 File Store requirements

The File Store component is a critical part of the Data Warehouse system responsible for storing and provisioning of file-based data in various data processing scenarios of the EurValve platform. A survey of the technical work packages showed that this component will be used in areas such as image segmentation, system models, sensitivity analysis, reduced order models and potentially in machine learning if the literature documents from which knowledge is derived are to be stored and made available to the decision support system. The File Store component should satisfy the following requirements:

4.1.1 Integration support in various systems/tools

File access in the EurValve platform will be required by many parties with various technical skills so a broad support for existing tools is needed. In particular, standard file browsers present in popular operating systems should be able to connect to the File Store component, access from a web-based user interface should be available and finally the file transfer protocols selected should allow for integration of other services and tools via an API with user credential delegation.

4.1.2 File access restriction management

Medical data, even if anonymised, is considered sensitive and so the File Store component should allow data owners to limit its use to approved users or groups of users. Management of file access policies should be available through one of the components of the EurValve platform. The system should take care of applying restrictions to files uploaded by any application in a reasonable and predictive manner. For instance, uploading file resources to a folder should inherit the restrictions of that folder, copying file resources within the File Store space should also copy their restrictions, etc.

4.1.3 Integration with a centralised authorisation system

The security infrastructure of the EurValve platform is based on a centralised approach with a single policy decision point (PDP) making authorisation decisions. Accessing File Store component's resources should also be a part of the PDP authorisation pipeline. This approach allows for convenient management of file access restrictions and enables user credential delegation in which any EurValve component can access File Store resources on behalf of a user whose credentials it possesses thus being a part of a larger data flow with just a single user sign-on action (SSO).

4.1.4 External resource reference and local mapping

File resources of the File Store component should be easily addressed by external components. This is especially important for computation and database services which keep only references to files being processed. In cases where computation services retrieve large amounts of file data it may be necessary to access the physical file storage directly in order to improve files transfer rates. Such use cases should be possible with the underlying storage systems, especially when the computation and storage facilities are hosted at the same compute centre.

4.1.5 Support for direct communication from web agents

Web applications are restricted with the same-origin policy which means that communication between a web agent (web browser) and a server is limited to a single domain. This constraint is relaxed with the use of the CORS (Cross-Origin Resource Sharing) [CORS] protocol, however, the server side has to support it. In the case of the File Store component server side is an in-house product so this approach is feasible. As a result, web applications can benefit from making direct calls to the File Store component without relaying the network traffic through an intermediate server which improves transfer rates and resource utilisation.

4.1.6 Convenience tools for file web access

Operating on complex file structures (many files, deep directory structures) from web browsers is difficult without support for directory uploads and downloads. The support for such operations is improving, however, not all major browsers have support yet. If possible the web file browser should support such cases to improve user experience.

The above list is not exhaustive and may be changed in the course of implementation and user verification. Based on the presented requirements in the following sections design proposals and implementation recommendations are given.

4.2 File Store design recommendations

In this section design recommendations for the File Store component are described by listing benefits and drawbacks of subsequent approaches basing the design on the WebDAV protocol.

4.2.1 WebDAV specification

Based on the requirements listed in the previous section and experience from another clinically-focused project (LOBCDER), the WebDAV protocol specification [WEBDAV]is recommended as the underlying technology for building the File Store component. This protocol is HTTP-based (Hypertext Transfer Protocol) [HTTP] and conforms to the widely adopted REST (Representational State Transfer) [REST] standards which are well-supported in various programming platforms. Its maturity allows integration with many file browsing tools (e.g. Windows Explorer for Windows operating system or DavFS for Linux systems). All file and directory operations are requested by using different HTTP verbs (listed below) so integration with an external authorisation system is straightforward and referencing file resources is done by unique URLs (Uniform Resource Locator). Depending on the WebDAV

implementation used mapping to physical resources may differ but generally it should be easy with standard file system-based storage facilities (more details are given in the implementation recommendations section).

Among the many HTTP methods which WebDAV supports, the list below details the most important from the perspective of an authorisation system. The methods are grouped according to file resource read/write permissions.

WebDAV methods allowing file read:

- PROPFIND: This method retrieves file properties such as name, size, etc. and for directories the method lists all the contained resources. Generally, this method allows browsing of the directory structure of the File Store component. Restricting this method limits access to certain parts of the directory tree.
- GET: This method retrieves file contents. Restricting combination of this method and the previous one (PROPFIND) might allow a user to view the contents of a certain directory but forbids files in that directory to be retrieved.

WebDAV methods allowing file write:

- MKCOL: This method is used to create new directories identified by the URL to which the request is sent.
- PUT: Method used to upload new file resources to a directory specified by a URL.
- DELETE: Method invoked on either a file or directory resource to remove it. Where a directory is removed all containing resources are removed as well.

The basic set of methods listed above allows management of access restrictions by an external policy decision point which should accept a resource URL and the requested method. It is assumed that the external policy decision point supports regular expressions in order to improve file access management. Organising the structure of the File Store component in a multi-user environment is a separate matter and several strategies to address this concern are presented in the next section.

4.2.2 Directory mapping schemas

WebDAV directory structure is similar to a standard file system structure in which directories can contain files and other directories within a root directory. Depending on the data access restrictions, user experience and implementation complexity, different combinations of directory structure and permissions can be imposed by the system. Below, three approaches to the problem are presented. In the diagrams users and groups are mapped to certain directories with specific WebDAV operations to show the permissions required in an external policy decision point. Regular expressions are simplified and a single asterisk (*) symbol means that preceding methods apply to direct child resources while a double asterisk (**) allows given methods to be invoked on all descendant resources.

4.2.2.1 Single policy approach

To create a common file exchange space a single permission policy attached to the root node of the WebDAV store is sufficient (see Figure 4). Only two groups of users are recognised.

Members of the readers group are able to browse and read all the content of the File Store (PROPFIND and GET methods) while those who belong to the writers group additionally can create directories, upload files and delete both types of resources (additional MKCOL, PUT and DELETE methods). The policy stored and validated by the policy decision point against WebDAV resource actions is applied to all resources by specifying the URL with a regular expression matching all resources. If required, a third group with only the DELETE method can be created to prevent any data loss in the system. Directory structure can be created by any member of the writers group without any limitations. If the MKCOL and DELETE methods are removed from the readers and writers sets of methods, a predefined structure for the file repository can be established to impose a desired directory layout. In such cases only browsing, downloading and uploading of files would be possible.

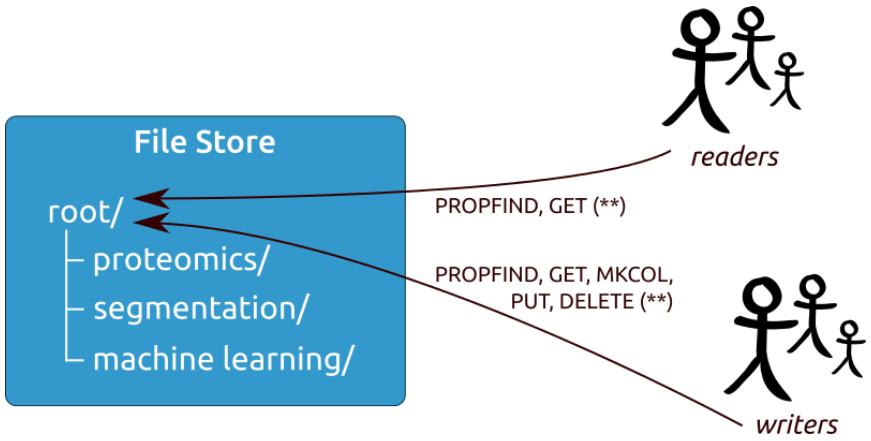

Figure 4. Single policy approach to building the File Store component. File access policy is attached only to the root directory and distinguishes two groups of users with read and write permissions.

In its simplicity the proposed approach gives quite a few configuration options to build a file repository. The benefits include trivial configuration of authorisation policy, lack of communication from the repository with the authorisation framework (in order to build new policies) and simple group management. Drawbacks include lack of maintaining user-private or group-private directories, abundance of resources as every user has access to every directory and file, as well as, necessity to deploy several file repositories to ensure data isolation among different groups of users.

4.2.2.2 Multi policy approach

The approach presented in Figure 5 allows for more flexibility in managing permissions to access file resources. At the same time a system-imposed structure is in place to control how the resources are created. The approach is limited to allow creation of directories only in the root directory to prevent storing mixed file content in one place (the root directory). After creating an initial directory, a user is free to create subsequent resources as they like. Top directories can be shared with a group to create group-private file spaces. It is also possible within a group-shared directory to distinguish writers and readers of file resources (e.g. by defining two groups such as my-group-readers with PROPFIND and GET methods and my-group-writers with PUT, and MKCOL methods). Directories and files at any depth can be shared, however, it is necessary to ensure that all parent directories allow at least the PROPFIND method to be invoked by clients which browse resources from the root directory

(if the URL for the deeply embedded resource is known, direct access should also be possible). The flexibility of this approach comes at the price of implementation complexity. Creating top directories has to invoke an action on the authorisation framework to create a new policy for the new resource initially limiting access to the resource owner. An alternative might be to create user-private directories while account registration takes place, however, such directory structure is very limiting and forces creation of directories which might be never used.

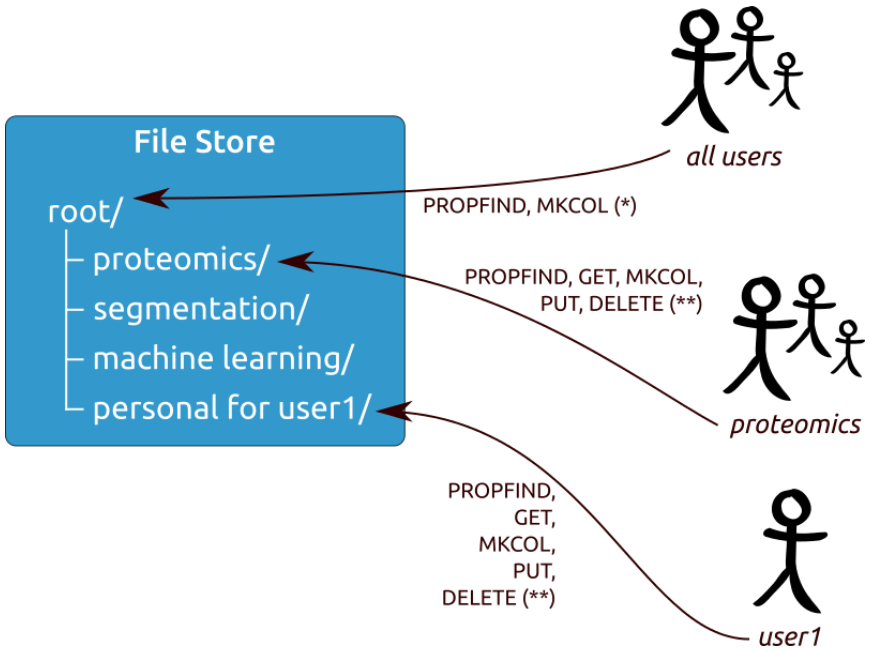

Figure 5. Multi policy approach to building the File Store component. Access policies are attached to different nodes according to user sharing actions. Private spaces can be created for individual users and groups.

The main benefit of this approach is the creation of a flexible directory structure which suits the requirements of data handling in the EurValve platform. Moreover, private file spaces for users and groups can be established and different levels of access maintained for given resources. The downside is that additional implementation cost is involved in integrating the File Store component and the authorisation framework. The following sequence of steps has to be followed when creating top-level directories:

- User creates a new top-level directory using any available WebDAV client (MKCOL method allowed on the root directory permits this action).
- The File Store component contacts the authorisation framework to create a new policy allowing full access to the created directory to the user (communication authorisation required, REST API exposed by the authorisation framework required).
- If the policy creation succeeded the whole operation finishes with success, otherwise an appropriate error message is returned.

Another drawback is that all users have access to all top-level directories (they allow the PROPFIND method on the root directory), which may clutter the initial view when browsing the file repository.

4.2.2.3 Dedicated view approach

In this approach (Figure 6) each user is presented with a dedicated view of the file repository which contains only private resources and resources shared with the user or with groups the user belongs to. Such organisation of the repository is possible using one of the available WebDAV extensions which is described in detail in an online RFC document [WEBDAVBIND]. To implement this a tight integration loop has to be built between the repository and the authorisation framework as each action of adding or removing a new account as well as changing group ownership has to initiate an update operation on links within the repository. For instance, when a user is removed from a group all links in his repository space shared with that group have to be removed as the user has lost permission to access resources shared with a group from which he was removed. Another requirement for the authorisation framework is to recognise linked WebDAV locations for which permissions have to be retrieved from the linked location. WebDAV clients have to mount repository resources to a certain depth level by convention. The mounting point is unique to each user as the path ends with user's identifier.

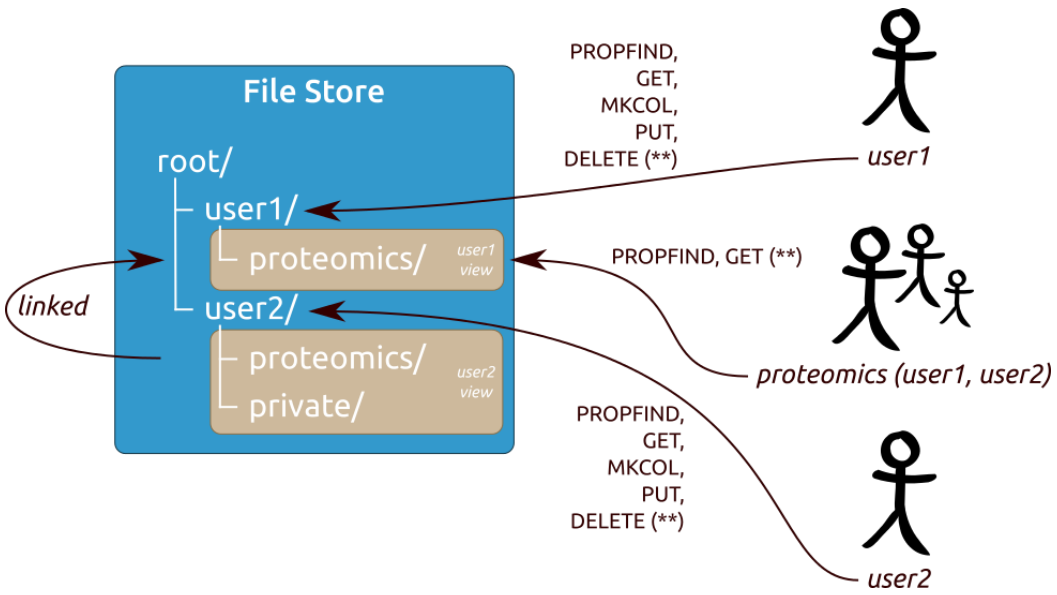

Figure 6. Dedicated view approach to building the File Store component. The view of the repository is limited only to a private user space. Shared resources are linked to that space.

With this approach the number of items each user is presented with is minimal, only items accessible to each user are shown. This gives a more modern user experience similar to existing file sharing services such as Dropbox [DROPBOX] or ownCloud [OWNCLOUD]. However, implementation complexity greatly increases as it is necessary to integrate user and group lifecycle management with file resource management lifecycle, which requires both the authorisation framework and the file repository to expose appropriate APIs. Let's consider a case in which a user is removed from a group:

- An action to remove a user from a group is invoked on the authorisation framework (it is assumed that there are file resources shared for this group).
- The authorisation framework analyses the permission database to see if any file resources are shared for the given group.

- The File Store component is contacted to remove all links from the user's repository space for resources shared within the group the user is removed from.
- After a successful repository link update operation, the authorisation framework commits the removal operation.

Another concern that has to be addressed is maintaining coherency in the linking structure so that there is no situation where dead links exist. Operations which lead to the creation of a link with an empty target should be prevented by the repository component.

4.2.3 Authorisation framework integration

The File Store component delegates authorisation decisions to a centrally managed policy decision point by submitting a resource URL and an HTTP method invoked on a given file resource. Depending on the decision result sent back by the authorisation framework access to the resource is either granted or rejected. Integration with the authorisation framework can be accomplished in two ways.

In Figure 7 on the left usage of an internal PDP client is depicted. Each request to the WebDAV repository is blocked by the client until an authorisation result is retrieved from the policy decision point. The client is an integral part of the File Store component and its implementation is provided by the File Store component's developers.

Another possibility, presented on the right of Figure 7 is to use a PEP filter provided by the security development team. This is deployed on a proxy server (e.g. Nginx) and mapped to the address space handled by the File Store component. Requests which are properly authorised are pushed to the File Store and those declined by PDP are returned with an error code, never reaching the File Store component.

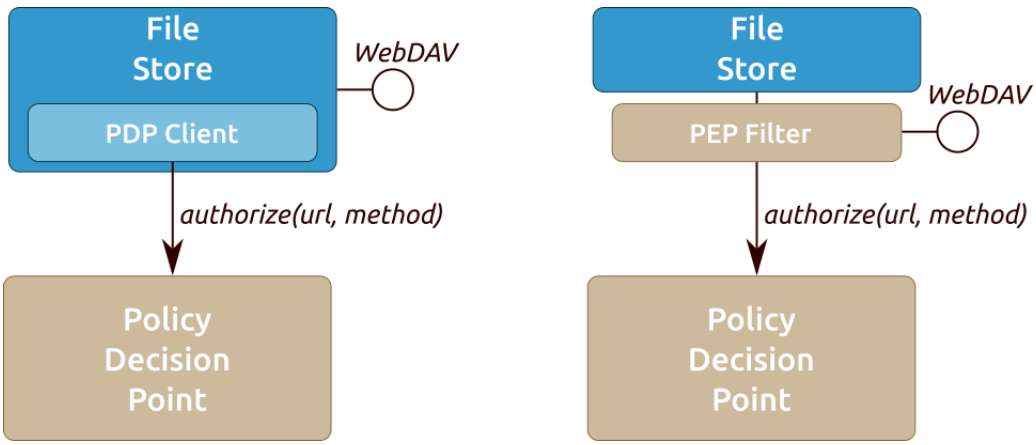

Figure 7. Integration between the File Store components and the authorisation framework's Policy Decision Point. On the left an internal PDP Client contacts PDP to authorise WebDAV requests. On the right a PEP filter proxies and authorises all WebDAV requests pushing only those accepted by the PDP to the File Store component.

Choosing between these two options has its pros and cons. The first option ensures that the File Store component is always secured as the PDP client is an integral part of the deployment but on the other hand each request has to be handled by this component, which may influence its performance. The option with the PEP filter is based on a lightweight implementation using a

module from one of the available front-end HTTP servers and relieves the File Store component from processing requests not allowed by the authorisation framework. If, however, the repository implements an audit log only requests allowed by the PEP filter will be recorded. Also, it is critical to correctly configure the filter to ensure the file resources are secured.

4.3 File Store implementation recommendations

To implement any of the presented approaches to building the File Store component existing systems and libraries which are listed and characterised in this section should be used to save development time and shorten the component delivery schedule. As the WebDAV protocol is a well-established standard, available implementations are robust and well-tested.

Name	Function in EurValve	Setup/mode of operation
Nginx/Apache WebDAV module	These modules allow establishment of a test infrastructure for preliminary integration of the WebDAV repository with the authorisation framework in order to evaluate overall performance and PDP overheads. Communication with PDP can only be accomplished with the PEP filter approach.	In case of Nginx two additional modules need to be compiled in the final server build: ngx_http_dav_module, nginx-dav-ext-module. For Apache httpd and the mod_dav module have to be enabled. Both servers need to be deployed on a machine with access to the storage element.
Apache Tomcat WebDAV servlet	May be used as the basis for the final implementation of the File Store component. Both integration options with the PDP can be implemented. Custom extensions can be easily added by embedding the servlet in a standard web application.	The servlet is deployed as part of a standard web application which requires a Java Virtual Machine to be present on the target machine. Access to storage is maintained through a mounted file system in which repository files are stored.
Apache Jackrabbit	May be used as the basis for the final implementation of the File Store component. Implements a WebDAV binding extension required by the third presented approach to building the File Store. Additionally, Jackrabbit supports handling of WebDAV properties.	Jackrabbit can be deployed as a standalone server or be embedded on a web application in case custom modification is required. The only dependency is a Java Virtual Machine installed on the target system and file access to the storage element.

All presented technologies are open source projects and can easily be modified if custom changes are required. In the course of the implementation phase performance analysis should be performed to evaluate if the combination of the File Store component and the authorisation system is fast enough to handle typical EurValve use cases. Proper metrics should be identified while the system is being implemented.

Integration with a central and remotely deployed policy decision point may lead to significant delays in performing large number of file operations. This can be caused by either network issues between the file store and PDP servers or by a huge request load on the PDP service and

as a consequence it may make the file service unresponsive. If such state occurs during file store operations the following measures might be undertaken.

- A local short-term cache may be implemented as part of the file store component which depending on the security policies would cache PDP decisions for a certain amount of time (e.g. a couple of seconds to take care of a high upload rate of small files). Such cache mechanism has to properly process requests for file downloads/uploads from/to a specific directory as each request will carry a different file URL from which the directory path has to be extracted and become a key in the cache registry. Policies for other directories may differ but it can be assumed by the implementation that if a file can be uploaded to specific directory other files can be uploaded as well.
- A replica of the PDP service might be deployed near the file store server to improve the communication between the components. Such approach would require a synchronisation mechanism to be implemented which would retrieve current policies by the local replica. From the file store components' point of view this solution does not need any modifications except for changing the PDP address, however, implementing the synchronisation mechanism depending on the security requirement might pose a challenge.

5 MODEL EXECUTION ENVIRONMENT

One of the main objectives of WP2 is elaboration of a flexible and easy to use environment (a kind of virtual laboratory, a problem solving framework) for development, deployment, and execution of applications for mesh generation, learning processes, large scale flow simulations and sensitivity analyses.

5.1 State of the Art

5.1.1 High Performance Computing

Scientists usually access high-capacity and high-capability computing through a remote shell. This requires them to log into an access node of the selected computing infrastructure, and to issue any further instructions from there, using the textual command line interface. This way the users of HPC computers stage their input file transfer, conduct required simulations and retrieve the output files, either to their local desktop machines, or to some kind of remote/cloud storage.

Therefore, the basic mode of access to HPC is through the SSH protocol [SSH], or an extension thereof, a GSI-SSH protocol [GSI]. The latter provides the possibility to use the so-called X509 proxy certificates [PROXY], to temporarily get access to a computing element (or to delegate rights of access to a trusted third party). Basic file transfer is usually performed through the SCP command (which also relies on similar transport mechanisms like SSH), or a more advanced GridFTP high-performance transfer protocol [GFTP]. The latter also requires the user to be able to issue the X509 proxy certificate for credential delegation.

Most available scientific HPC resources provide computational power on a fair-share basis. They are multi-user, multi-team systems, that, in order to maintain consistent rules of access to common resources, utilise some kind of job queuing mechanism. Implementations of such queues (examples are SLURM [SLURM] or Torque [TORQUE]) require the user to:

1. **describe** in detail as a kind of "recipe" file which computation should be run, what amount of computational power and storage capacity should be used, and which input files have to be accessed,
2. **stage in** (i.e. copy) all required input files to the computing resource's file system,
3. **submit** the job description to a selected queue,
4. **wait** for the queuing system to allocate resources to that job and for the job to complete execution using these resources,
5. **collect** output files, optionally copying them (stage out) to either a personal computer or some dedicated storage.

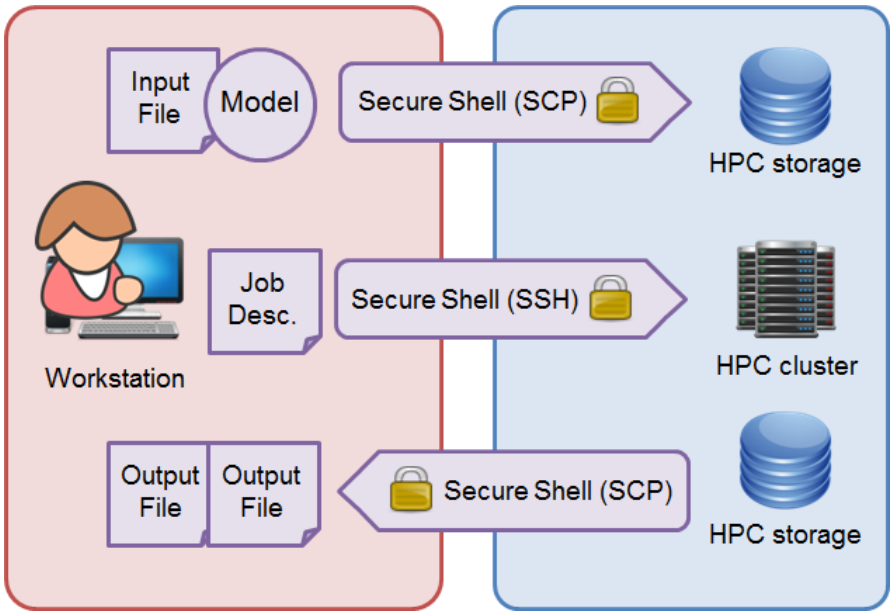

Figure 8. A user, utilising a series of secure shell connections, transfers files (e.g., simulation input files, heart valve 3D model) to the HPC cluster's file system, submits a computational job execution (for instance, a fluid dynamics simulation) and retrieves job execution output (e.g., the visualisation of the performed blood flow simulation through the uploaded heart model).

As explained before, the above procedure (see Figure 8) is conducted, in the basic scenario, using a shell prompt and a set of commands provided by the queuing system. Please note this procedure introduces a factor that is outside of the user's control - the job queue waiting time. Basically, the more in demand the computing system is, the longer the expected wait time. On the other hand, the advantage of such a setup is that the resource allocation subsystem takes care of proper MPI configuration for parallel jobs, usually optimising possible networking bottlenecks (e.g. by preferring single system setup, over multiple system setup, when possible) - which is especially important for EurValve as listed in Section 2.

In order to streamline the described basic mode of access, various software services, collectively called middleware, have been created. Examples of such middleware are Globus Toolkit [GLOBUS], Unicore [UNICORE], QosCosGrid [QCG], or rimrock [RR]. The most fundamental element that they provide is a middle layer, that works as a kind of proxy between the user and the computing element (e.g. a computing HPC machine), which, to a lesser or greater extent, automates and simplifies the data management and job submission procedure described earlier in this section. Being a middle layer, these solutions require some kind of rights delegation - usually the (already mentioned) X509 proxy certificates are used for this purpose. This delegation allows middleware to take care of file staging, job submission and status querying (for job completion signal) and retrieval of output files.

What is more, middleware combined with a mechanism for credential delegation, allows another layer of graphical user interfaces to be provided. Examples include domain-specific science gateways (e.g., InSilicoLab [ISL], Galaxy Server [GLXY]), or generic UI tools (e.g., PLG-Data [PLGD], UnicorePortal [UNIP]). Alternatively, libraries exist to allow developers to build their own, specific graphical research environments (e.g., plgapp [PLGAPP], rimrock [RR]).

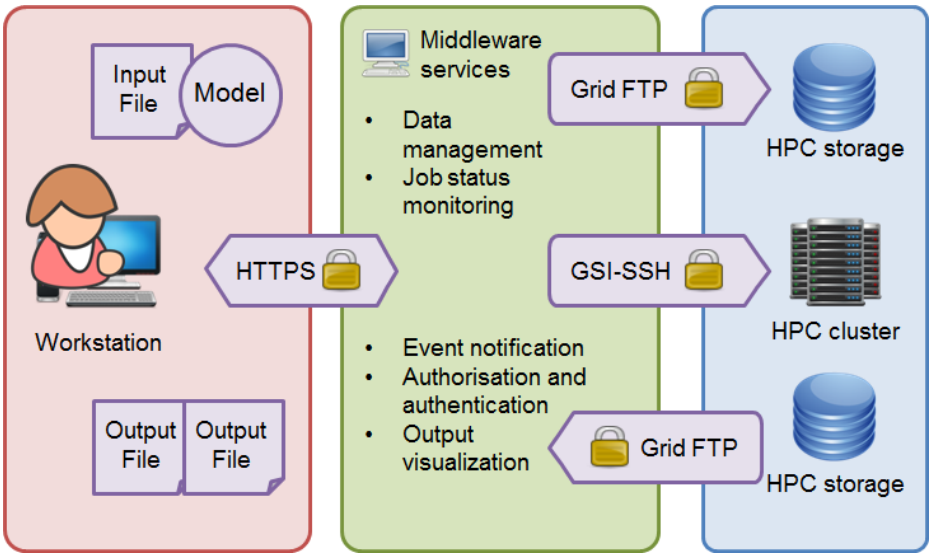

Figure 9. An alternative scenario where the user delegates one's credentials to middleware services, which in turn manage user's input and output data, create and submit the computational job description, actively monitor job execution on the remote computational cluster, and notify the user of completion status. Sometimes such services are also able to visualise simulation execution outputs to the user.

Being equipped with such a rich set of easy-to-use tools (as shown in Figure 9) and solutions, today's HPC scientist is able to conduct complex computational research in an organised, simplified and, to some extent, automated way. In Section 5.2 we describe our recommendations for middleware tools for the Model Execution Environment, in order to meet the requirements listed in Section 2.

5.1.2 Cloud computing

Cloud computing compared to HPC features a different approach. Resources (CPU power, memory, storage, etc.) and services are available on demand. Users can easily acquire exactly what they need, when they need and are charged on a pay-per-use basis. In a cloud model physical resources are maintained by a provider that usually expose both an API (for instance REST interface) and graphical web-based consoles to allow users to request and manage resources. Provisioning time varies from seconds to minutes. The cloud offers great elasticity and scalability thus users can easily take advantage of large computing power and services without the burden of building their own hardware infrastructure.

The cloud provides a wide range of services on different levels. At the lowest level there is Infrastructure as a Service (IaaS), that enables management of virtual machines and templates. Users can manage the full life cycle of servers. A variety of templates can be used to instantiate a server (a raw OS image of the preferred kind or one with a pre-configured software stack). What's more, the user can save his/her templates and choose to make them publicly available. IaaS gives the user root access to servers and makes them responsible for system administration. There are currently a number of commercial IaaS providers including Amazon, Google, Rackspace [RACKSPACE] and Microsoft.

One level higher is Platform as a Service (PaaS) that provides users (usually developers) a hosting environment for applications implemented in a specific technology. PaaS may provide additional services such as databases, file storage and queuing systems but it may also impose

some limitations on developers (for instance a request must be served within a given time limit).

Software as a Service (SaaS) offers ready to use software services. Providers take full responsibility for the infrastructure, from hardware to the software stack, and for hosting a service. Users are charged on an agreed basis (monthly, yearly, per request etc.).

In addition, there is a multitude of services that cannot easily be assigned to the categories described above. At the moment of writing this deliverable, Amazon, which is a leading cloud vendor, provides in total sixty-seven distinct services [AWS]. Microsoft offers fifty-nine cloud services within the Azure platform while Google provides forty-seven [GOOGLE-SERVICES]. AWS Lambda [AWS-LAMBDA] is an example of such a service that might be of interest to EurValve users. It provides the possibility to create a stateless function (in one of the supported languages, currently Node.js, Java, Python), which will be executed on Amazon scalable infrastructure in two scenarios:

1. When an event occurs (e.g. change in Amazon S3 Bucket or in Amazon DynamoDB table)
2. Following an HTTP request to Amazon API Gateway or an AWS API request is made

Each lambda can have RAM preferences assigned (max 1536MB, CPU will be assigned automatically). Amazon takes care of the scaling up/down of the infrastructure needed to handle all lambda requests. The user pays for time when the Lambda function is running and the RAM used by the Lambda instance.

Apart from distinguishing the type of service, cloud computing can either be performed at a commercial vendor site that is publicly accessible or using private infrastructure. The former outsources all administration and operation work to provider but may be inappropriate in case of handling sensitive data, licensing issues or maybe be cost inefficient. In such cases a private cloud infrastructure can be built and managed in house. An extensive survey on cloud stacks [VPH-SHARE-D2.1] and our experience with providing cloud services proved that OpenStack [OPENSTACK] is the most mature and feature-rich software for building private cloud infrastructure.

A “silver bullet” solution is to use a hybrid cloud consisting of private and public (commercial) providers. Atmosphere [ATMOSPHERE] is a cloud platform that federates resources from both commercial providers (such as AWS, Google Compute or Rackspace) and private OpenStack deployments. It facilitates uniform usage of infrastructure and cloud application life-cycle management. Atmosphere allows exposing services deployed in sites using private addressing to the outside world as well as HTTP and HTTPS load-balancing. It supports cost optimisation by selecting the cheapest servers that meet user requirements and by sharing servers among users when possible. Furthermore, it has a built-in billing mechanism that provides control over expenses. Deployment of the platform is depicted in Figure 10.

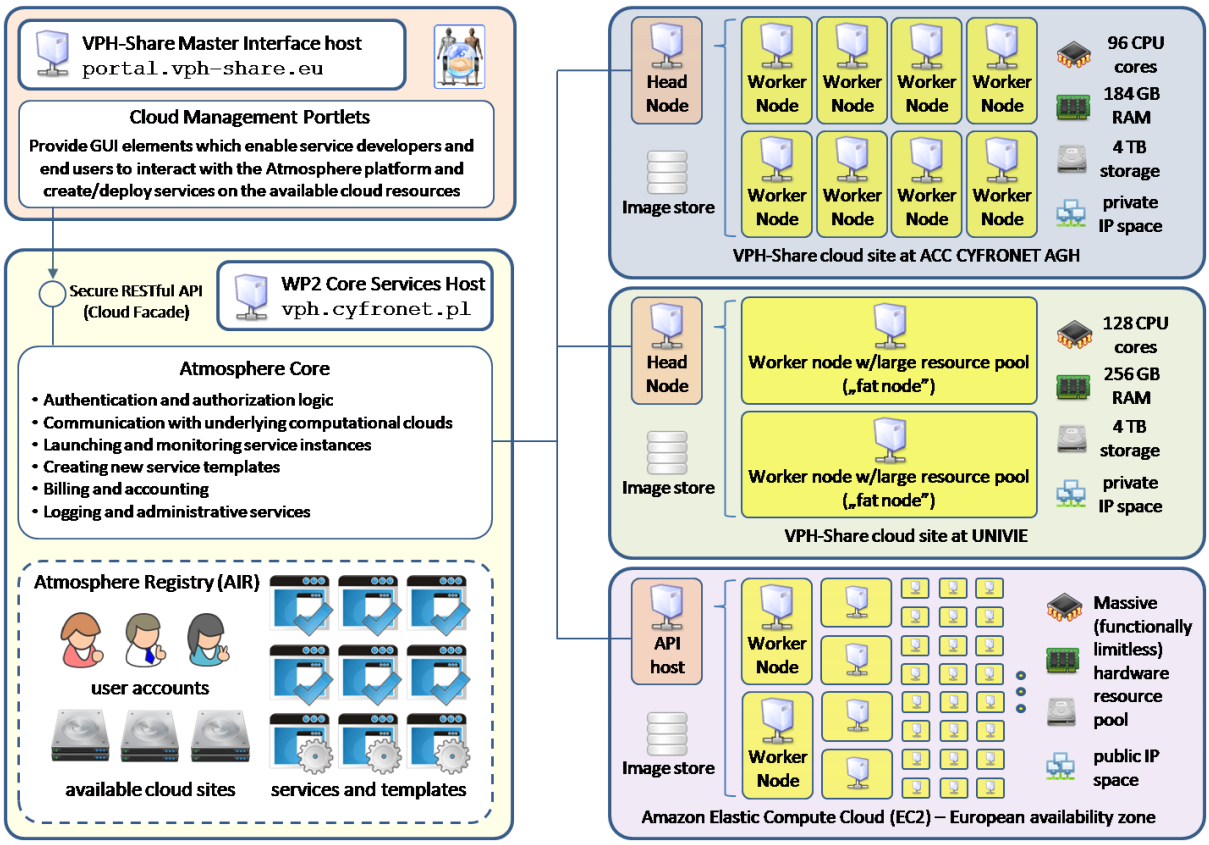

Figure 10. Deployment of Atmosphere Cloud Platform for the VPH Share project. Web-based client applications access Atmosphere functionality via a secured REST interface. Atmosphere manages two private cloud sites (in Cracow and Vienna) and the commercial AWS site in Ireland.

The Atmosphere Cloud Platform provides a REST API and web-based interface. It is a mature solution that has proven its qualities for various groups of users. It has been successfully deployed and operated in VPH-Share [VPH-SHARE], PL-Grid [PLGRID] and ISMOP [ISMOP] projects. Besides those it has external user groups from VPH-Dare community [VPH-DARE] and the Jagiellonian University Medical College [CMUJ].

5.1.3 Containers

The main drawback of the cloud virtualisation technology is connected with the fact that, once started, Virtual Machines share nothing with the underlying host. Thus, starting a new VM is quite expensive and time consuming. As a result, start time can range from a few seconds to several minutes. Containers (based on operating system level virtualisation) try to deliver the same level of encapsulation as the Cloud (based on multiple isolated user-space instances), but do not require a separate operating system. Instead containers are based on kernel functionality and use resource isolation. By using containers resources are isolated, services restricted and individual processes are unaware of the fact that the operating system is shared. In conclusion, overheads for containers are usually low to non-existent, because programs started in containers use the operating system's normal system call interface – no emulation is needed as in the Cloud computing situation. As a result, container start-up time is very short, typically insignificant compared with application start-up time.

Container solutions are available on the Linux operating system starting from 1982, when the *chroot* solution was introduced, but the hype for using containers started with the Docker [DOCKER] release in 2013. In the next part of this section we will focus mainly on Docker and solutions built on top of Docker, it is the most mature solution and the ecosystem for creating and managing containers.

5.1.3.1 Docker

Docker is an open-source project, with main focus on automation and application deployment inside containers. It uses kernel features (such as *cgroups*, *namespaces*, *aufs*) to start containers without the need to start and maintain virtual machines. Figure 11 illustrates the architecture of Docker.

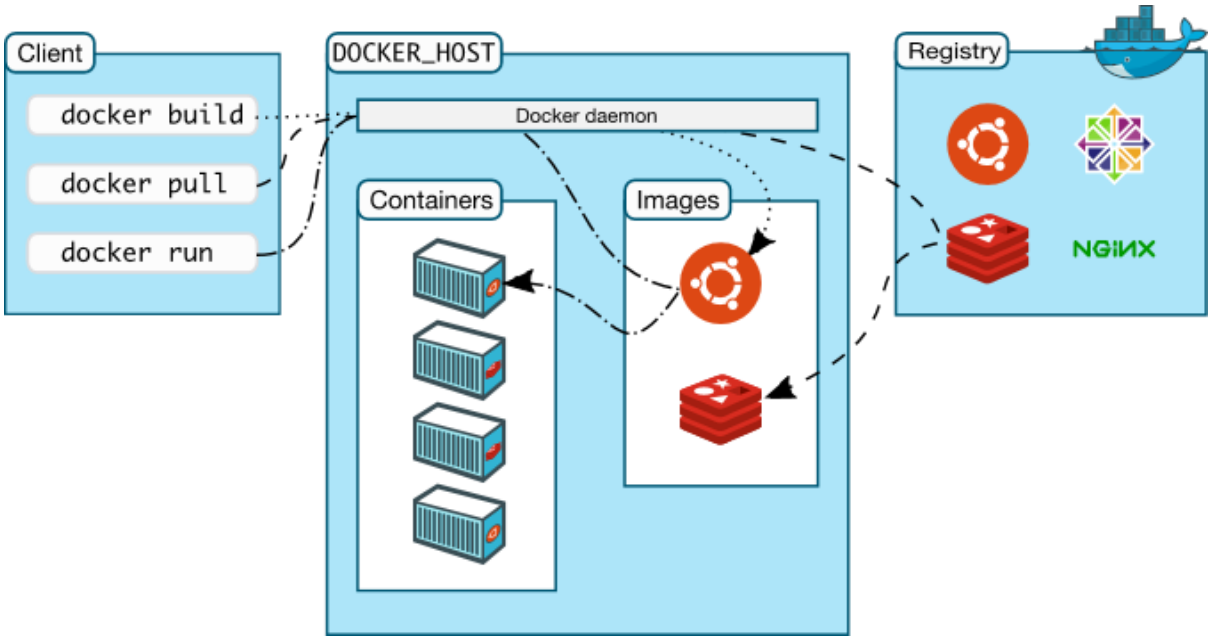

Figure 11. Docker architecture. Docker is created in client-server architecture, thus for the user it is simple to switch from using a local or remote Docker server. An image repository is used to find container images. If the image is not found in the local repository it is fetched from a remote image repository. (Source https://docs.docker.com/v1.8/introduction/understanding-docker)

Docker uses client-server architecture. The client talks to the Docker daemon, which can be started locally or remotely. The daemon is responsible for building, running and distributing Docker containers.

The Docker build process takes an image build description file (usually called *Dockerfile*) and produces a new container image, based on another container image. When storing the new image only the difference between the source and target images is stored. As a result, the size of the new image can be relative small. An example of a container with MPI installed is presented below based on the latest stable Ubuntu image, with additional MPI libraries are installed.

```
FROM ubuntu:latest

RUN apt-get update && \
```

```
    apt-get --no-install-recommends install -y \
mpichlibmpich-dev make openssh-client openssh-server \
&& apt-get clean

RUN mkdir -p /root/.ssh&&\
printf "Host *\nStrictHostKeyChecking no\n" > /root/.ssh/config
COPY id_rsa /root/.ssh
COPY id_rsa.pub /root/.ssh
RUN cat /root/.ssh/id_rsa.pub > /root/.ssh/authorized_keys&& \
chmod 400 ~/.ssh/id_rsa
RUN mkdir /var/run/sshd

RUN mkdir -p /mpich/src/app
WORKDIR /mpich/src/app
ENV MPIEXEC_PORT_RANGE 1000:1000
EXPOSE 1000

COPY Makefile /mpich/src/app/
COPY . /mpich/src/app
RUN make

CMD ["/usr/sbin/sshd", "-D"]
```

From such a Dockerfile we can build a new image:

```
Docker build –t dice-cyfronet/mpi-hello .
```

The newly created image can be stored locally, or deployed remotely, e.g. into Docker hub [DOCKER-HUB]:

```
docker push dice-cyfronet/mpi-hello
```

The last step is to start the newly created container:

```
docker run –i -t dice-cyfronet/mpi-hello
```

5.1.3.1.1 Composing multi-container applications

Usually applications are not composed from one container, but from a set of containers. For example, a classical web application is usually composed of the database and application serving web pages. In such a way the database can be installed inside the first container and the web server on the second container. There should be a way to link these two containers together. Docker delivers such a solution, which simplifies creation of complex distributed applications deployed as containers – Docker compose [DOCKER-COMPOSE]. An example presented below is taken from the smogmapper[1] application, which is created as a Ruby on Rails application with a PostgreSQL database. It presents how two containers can be linked together:

```
db:
  image: postgres
web:
```

[1] https://github.com/Nuanda/smogmapper

```
build: .
command: bundle exec rails s -p 3000 -b '0.0.0.0'
volumes:
  - .:/smogmapper
ports:
  - "3000:3000"
environment:
  SMOGMAPPER_DB_USERNAME: postgres
  SMOGMAPPER_DB_HOST: db
links:
```

- db

Docker compose allows injection of information about other containers into application by defining links between containers. As a result the application can be started by simply executing:

```
docker-compose up
```

As a result the *db* image will be started from the *postgres* image. The second image with the web application will be built based on *Dockerfile* from the current directory and the defined command (`bundle exec rails s -p 3000 -b '0.0.0.0'`) will be started. Another advantages is that the network is configured in such a way that the web application can access the database instance.

5.1.3.1.2 Container clusters

Running containers on one physical host is usually not enough for complex applications. That is why solutions for clustering containers or more generally distributed application orchestration are gaining popularity nowadays. There are many existing solutions solving these problems, such as Docker Swarm [DOCKER-SWARM], Kubernetes [KUBERNETES], TerraForm [TERRA-FORM], TOSCA [TOSCA], Heat [HEAT]. A comparison of these technologies can be found here: http://getcloudify.org/2015/06/11/orchestration-docker-cloud-automation-openstack-heat-tosca-kubernetes.html. In the next part of this section we will present only Docker Swarm, because it is based on Docker and it seems to be the most promising for the EurValve use cases.

Docker swarm is a native clustering for Docker. It turns a set of Docker hosts into a single, virtual Docker host. The user can request a container start for such a virtual Docker host and swarm will decide the most optimum physical place to start the container. What is more swarm will also configure the container network in such a way that two containers started on different physical machines will be able to see each other. Below we present the procedure for starting simple multi worker MPI application on a swarm cluster:

1. Swarm should be configured with multi-host networking according to this instruction: https://docs.docker.com/engine/userguide/networking/get-started-overlay/
2. Next, an overlay network needs to be created (this network will be configured on all swarm hosts):
   `docker network create --driver overlay mpi`
3. Start multiple docker containers with MPI installed:

```
docker run -itd --name workerN --net=mpi --env="constraint:node==mhs-
demoM" cyfronet/mpi-hello-world
```

4. Run the MPI application, which will use as many worker nodes as defined in step 3:
```
docker run -it --name master --net=mpi --rm=true --
env="constraint:node==mhs-demo0" cyfronet/mpi-hello-world mpirun -n N
-hosts worker1,...,workerN ./mpi_hello_world
```

5.1.3.2 Conclusions

Container solutions look promising for the class of applications which do not require a dedicated network layer (such as InfiniBand delivered by Prometheus supercomputer) and reservation of a whole physical host. The biggest advantage in comparison to Cloud solutions is the reuse of the OS from the host machine, avoiding the need to start a new operating system. As a result, start-up time can be minimised. An additional advantage of using a container is the simplification of the process from application development to production runs as the production environment can simply be recreated on the developer's machine. Last, but not least, Docker and container solutions are likely to be available in HPC environments in the near future. Existing experiments, where Docker containers have been started on such infrastructures, e.g. [DOUGLAS2015] or [KNIEP], demonstrate quite good performance of clustering containers at scale [NICKOLOFF].

Currently, Docker and Docker Swarm appear to be the best container and container clustering solutions and we are recommending this pairing for EurValve applications which can take advantage of container technologies.

5.2 Design recommendations

In the scope of EurValve many applications will be created by many vendors. Characteristics and requirements of such applications will be different and as a result different execution environment elements should be used. The table below shows generic application requirements and the recommended environment where it should be hosted:

Scenario	Execution environment element
Require large computing power	HPC
Require standard numerical packages with license already configured (such as Matlab, ANSYS Fluent etc.)	HPC
Dedicated, guaranteed interconnect (application is communication bound, e.g. MPI)	HPC
Require root access to install and configure non-standard application	Cloud or containers
Require scalability and elasticity, isolation	Cloud or containers

Scenario	Execution environment element
Resources required on demand (immediately)	Cloud or containers
Require multiuser system (e.g. to mount NFS with different access policies)	Cloud
Avoid cloud vendor lock	Cloud federation (e.g. Atmosphere)
Require portability (similar stack on development, testing production environments)	Containers
Simple releasing and version management	Containers
Legal issues (data confidentiality of software ownership not guaranteed by HPC, public Cloud or containers)	In house solutions or private cloud, private containers cluster

For the part of EurValve's research pipeline that requires HPC access, to execute CFD simulations and sensitivity analysis study, taking into account the requirements listed in Section 2, and the current state of the art described in Section 5.1.1, we recommend the following:

1. Users who plan to overview the execution of simulations, should get a user account in the PL-Grid Infrastructure: this will give them the access to computational power and storage capacity adequate to the requirements of these large scale simulations. Moreover, required CFD solvers are installed and available in PL-Grid, so no additional setup effort is required. A suitable computational grant should be negotiated with the PL-Grid's Operations Centre to cover EurValve HPC needs.
2. Each of these users should also have issued a X509-compliant user certificate, so a GSI-based credentials delegation could be used when accessing computational and storage resources through middleware (see Section 5.1.1).
3. Any tool to be deployed in the scope of EurValve's Model Execution Environment, which has a direct link with the HPC part of the case study pipeline, should be integrated with PL-Grid's adequate security infrastructure element (for instance: web UIs should integrate with PL-Grid's OpenId identity provider, resource access tools should be able to pass user's X509 proxy certificate to PL-Grid resources).
4. It is recommended that transfers of large files to/from the computing clusters are performed using the GridFTP protocol, authorised with user's X509 proxy certificate, especially if they are the so-called *'3rd party transfers'* (not involving user's own workstation on any transfer end). Otherwise, a secure, encrypted SCP/HTTPS protocol should be utilised.
5. Regarding the number of patient cases involved, and the complex nature of sensitivity study setup, all study input and output files should be organised in a efficient and tidy

hierarchical structure, imposed by EurValve middle layer access tools. Having a consistent naming and addressing schema throughout the entire project will help various member of the Project's consortium to efficiently collaborate on the computational part of the patient case pipeline. It will also help results provenance tracking.

If one chooses to use cloud services, the following guidelines apply:

1. Instance size and service capacities should be chosen wisely. Bigger instance types provide better performance at higher cost.
2. Virtual machines should be as small as possible to reduce the overhead on service performance and minimise provisioning time.
3. Log rotation should be configured for deployed services as disk space on a single virtual machine is limited.
4. If the service is developed in in an iterative manner the virtual machine should not be permanently running. Instead the machine should be saved as a template (using Save or Save as functionality in Atmosphere), destroyed and instantiated again when required.
5. Cloud elasticity and scalability should be leveraged, starting with minimal resource allocation. If required servers can be scaled horizontally and vertically.
6. Resources should be conserved where possible. Servers that are not required should be stopped and old templates and files should be deleted. As resources are shared across the consortium suboptimal utilisation of resources means someone else is not able to use them, infrastructure performance suffers and costs increase.

For EurValve applications more suited to container technologies (lower computing power required, fast execution, e.g. Web Services with Image Segmentation services) we recommend the following:

1. Install Docker on a developer machine and test the application there before deploying it into the staging and production environment.
2. Use cloud resources managed by Atmosphere to start virtual machine(s) where Docker is installed and Docker Swarm (with overlay network(s)) is configured. This can be used to push a production version of the application composed of one or more containers.
3. For every application (composed of single or multiple containers) create a separate network with limited ports open to the outside world.
4. If the number of resources used by running containers varies over during time use cloud auto-scaling capability to add/remove virtual machines (with Docker installed) when needed.
5. Try to keep the container image as small as possible (as a result the container image will be fetched from the repository and started faster).
6. Choose carefully the source image for your container, since it has the biggest impact on image size and container start-up time (simplest source image which meets application requirements is the best).

5.3 Recommended technologies

Name	Function in EurValve	Setup/mode of operation
GSI	Accessing HPC resources. Providing user credentials delegation mechanism.	Pre-installed on the HPC resources. Integrated in th middle layer or user end tools which connect directly to HPC resources.
PL-Grid account management	Accessing HPC resources. Providing authentication and authorisation for the HPC part of EurValve MEE.	Server side provided by the PL-Grid Infrastructure. Client side to be integrated with any user UI that connects directly to HPC resources.
GridFTP	Transferring large files to/from the HPC storage.	Not required but recommended for higher transfer efficiency. Server side provided by the PL-Grid Infrastructure. Client side to be integrated with any tool that send/retrieves files to/from HPC storage.
rimrock	Submitting simulation jobs to a HPC cluster.	A REST service provided by the PL-Grid Infrastructure.
PLG-Data	Files management directly on HPC storage.	Both a REST service and a web application provided by the PL-Grid Infrastructure.
AWS or Google	Commercial cloud services	Leading vendors providing a publicly accessible range of cloud services
OpenStack	Building and managing private cloud infrastructure	The software stack to be deployed and managed on partners' own hardware infrastructure.
Atmosphere	Federating cloud resources and cloud application life-cycle management	To be decided. It is possible to deploy an instance dedicated specifically for the EurValve community or to use existing deployments.
Docker	Container execution and management	Installed locally (for development purpose) and remotely (on either private or public cloud resources or bare metal machines)
Docker compose	Multi-containers applications management	Not required but recommended to simplify multi-container applications. Installed in the same places where Docker is installed
Docker swarm	Containers cluster management	Not required but recommended to manage multiple containers and optimise resource usage. Installed either on private or public cloud resources or bare metal machines.
Docker hub	Repository for Docker container images	Not required but recommended to enhance container images accessibility. The public Docker repository can be used or a private repository can be configured to host private containers.

6 INTEGRATED SECURITY AND DATA ENCRYPTION

An integrated authentication and authorisation infrastructure will be provided for the EurValve platform. This section discusses the basic concepts underpinning this infrastructure, and suggests potential implementation options.

6.1 State of the Art

Planning an Integrated Security System, particularly one dealing with confidential data like EurValve, requires deep analysis of the current State of the Art. During this process we focus on multiple aspects of the system such as identity management and authentication, authorisation, policy management and data security.

6.1.1 Authentication and Authorisation

Authentication and authorisation could be done based on simple locally stored credentials however for ease-of-use Single Sign On (SSO) solutions are preferred. We have analysed multiple SSO schemas based on different protocols:

SAML based (Shibboleth) – SAML is an XML-based protocol for federated Authentication and Authorisation. It's prominently used by the Shibboleth framework which allows building of the tightly linked federations of Identity and Service providers. Although very powerful this solution is quite heavy-weight and rigid so we have decided not to implement it in our solution.

OpenID – is a very popular and well established Identity Provider which allows simple and lightweight authentication mechanism. As OpenID does not offer authorisation it needs to be augmented by other authorisation solution or replaced by newer frameworks like OpenID Connect.

OAuth 2.0 –is a lightweight authorisation solution (successor to OAuth 1.0) is used by major providers such as Google. Despite the fact that OAuth 2.0 itself doesn't provide authentication solutions such a solution could be built on top of it using some custom solution or standard like OpenID Connect.

OpenID Connect –is an authentication protocol standardised by the OpenID Foundation which enables authentication mechanisms based on the OAuth 2.0 authorisation schema. It allows use of OAuth 2.0 providers such as Google for Identity Management.

Another important aspect is the mechanism to encode and securely pass user data between the platform components.

Numerous solutions exist to provide authenticity of data, both in the XML domain (such as already mentioned **SAML**) and in the JSON domain – like **JSON Web Token (JWT)** which is based on **JSON** with its payload signed using the **JSON Web Signature (JWS)** schema.

Confidentiality of the data is usually ensured at transport level (TLS protocol) but might also be achieved at message level through solutions like XML Encryption or **JSON Web Encryption (JWE)**.

6.1.2 Policy Management

For complex systems like the one required for EurValve the final authorisation decision needs to be based on a policy defining not only which users have access to the services but also specifying the scope and access level of the access grant.

We have analysed solutions based on two distinct paradigms:

Attribute Based Access Control (ABAC) – this complex access control paradigm is based on atomic attributes attached to users. The decision is made based on evaluation of the whole set of policies. This gives the policy manager freedom to define required attributes, and ways the policies are combined. During our analysis we determined that this paradigm might be too complex for the EurValve needs. Its complexity would not only negatively impact efficiency during the policy assessment but also would require maintaining a highly complex set of policies. As many users (including resource managers) of the platform are domain specialists from non-IT fields maintenance of a complex policy set could be too complex.

Role Based Access Control (RBAC) – in this paradigm access control is performed based on a defined set of roles for the users. It offers a simpler yet still powerful mechanism for defining and enforcing authorisation of users. During our research we determined that the RBAC paradigm will be sufficient for the project needs as it allows linking of users, resources and actions performed on those resources.

Any access control paradigm requires a mechanism to store and manage rules and policies. ABAC is usually modelled in **XACML** language, which is very powerful but could be too complex for some use-cases. XACML might also be used for modelling the RBAC approach but in this case simpler solutions like LDAP or RDBMS are usually sufficient. There are a few examples, both commercial and open source, such as OpenLDAP (Open Source), MS Active Directory for LDAP and PostgreSQL, MySQL, Oracle Database or MS SQL Server in case of RDBMS.

6.1.3 Data Security

Of course many common scenarios require moving data between locations where they're stored or produced and where they will be processed. Various mechanisms could be used to ensure security, the most basic one being standard **Transport Layer Security (TLS)**, which is now commonly used for HTTP. This protocol is the default for the new **HTTP/2** standard for which encryption is not mandatory but strongly recommended.

If encryption at transport layer is not available (legacy applications/protocols) or an additional layer of security is required lower level techniques could be used such as TCP or UDP based **OpenVPN** tunnels, best suited for VPNs between client machines and Linux servers or an **IPSec** based solution best suited for Site-to-Site VPNs between network equipment. IPSec might also be used for Cloud provider assisted solutions such as the **Amazon Virtual Private Cloud (VPC)**.

If the volume of data is huge or additional privacy is required between private and public Cloud solutions physical connections between the organisation LAN and the provider might be used

such as **Amazon AWS Direct Connect**. This solution establishes a physical network connection in one of the Points of presence (PoP) supported by Amazon.

6.2 Detailed design

The EurValve platform will be composed of many distinct modules based on various technologies. In turn the proposed security framework needs to be flexible enough to support various API types which includes plain HTTP-based ones (like REST), XML-based (like SOAP WS used by .NET components) and HTTP extensions like WebDav used by File Storage components. On the other hand, the security framework itself needs to be homogenous and centralised to provide a good level of manageability.

6.2.1 Basic infrastructure

The heart of the proposed security system for the Project is the integrated Authentication and Authorisation system based on the Role Based principal (RBAC). Its main goal is to provide a straight-forward solution for policy definition and enforcement.

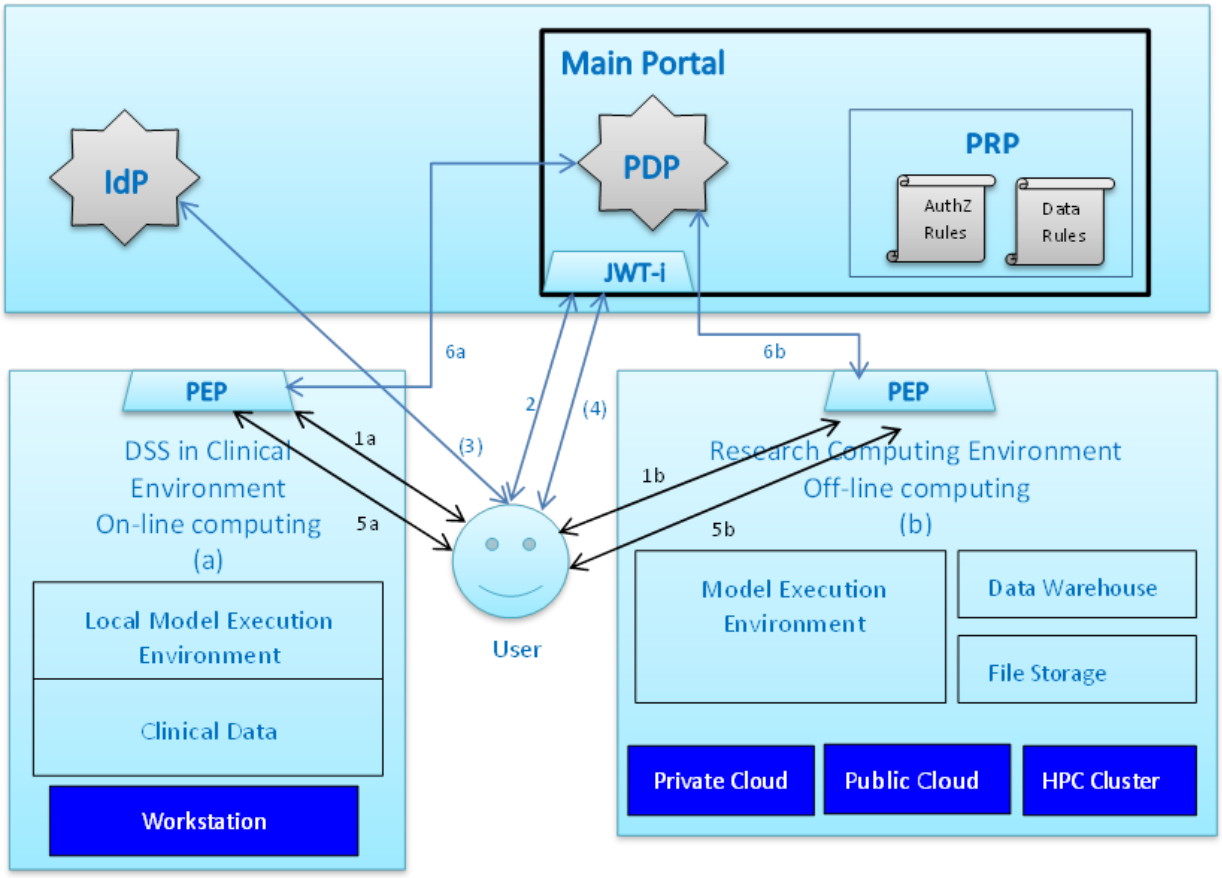

Figure 12. General architecture of the proposed security solution including Central Portal containing Policy Decision Point (PDP), Policy Retrieval Point (PRP); Policy Enforcement Points (PEP) deployed by the service providers as well as external Identity Providers (IdP).

The system will be composed of:

- A set of trusted external Identity Providers (IdP) based on OpenID including the PL-Grid IdP, but also any arbitrary providers that could be trusted.
- Core of the system contained in the EurValve Security Web Platform (EV SWP) which includes:
    - IdP assertions consumer
    - JSON Web Tokens issuer (JWT-i)
    - Policy Decision Point (PDP)
    - Policy Retrieval Point (PRP)
- Generic Policy Enforcement Point (PEP) for HTTP(S) services based on NGINX extensions
- (Optional) Specific PEPs developed by service providers based on JWT tokens and queries to the PDP
- (Optional) Specific PEPs for the locally hosted applications and services when the external constraints (such as license requirements) forces them to be bound to the given node.

As the team responsible for the security platform we will host and maintain the mandatory components described above as well as assisting in creation of optional PEPs where necessary by providing know-how for the developers. We will also provide best practices and assist in evaluation of security requirements of locally installed software as well as help in integration with the rest of the platform (including access control, credential delegation if needed).

The centralised PDP is required for easy policy management, however it may be replicated locally for redundancy or efficiency reasons. The generic overview of the system is shown in Figure 12.

The authentication process consists of following steps:

1. A user request to the service is intercepted by the PEP.
2. The user accesses EV SWP to get a token and authenticates with simple credentials (after which process continues from pt. 5 forward) or external IdP.
3. (Optional) User is redirected to the external IdP for authentication.
4. (Optional) EV SWP access is granted on denied based on assertion from the IdP.
5. User accesses the Service using token obtained form JWT-i.
6. PEP contacts PDP to get authorisation decision based on the user token, required actions and the users' roles. The decision is returned to the user. If access is denied the returned code indicates the reason (such as wrong token or insufficient rights) which allows appropriate action to be taken (e.g. re-authentication or sending request for sharing of the resources).

Details of the algorithm used by the PDP to reach the authorisation decision are shown in Figure 13.

The process is composed of the following basic phases:

(a) Request from the Service Provider's (SP) PEP to the PDP

(b) Validation of the SP itself to minimise chance of information leak – if invalid: terminate connection

(c) Validation of the ticket:
- i if ticket issuer is not trusted -> deny
- ii if issuer signature is invalid -> deny
- iii if user is not authorised to access service at any level -> deny,
- iv otherwise -> go to (d);

(d) Validation of the user roles:
- i if user role is sufficient to perform required action on the resource ->**grant access**
- ii otherwise -> deny

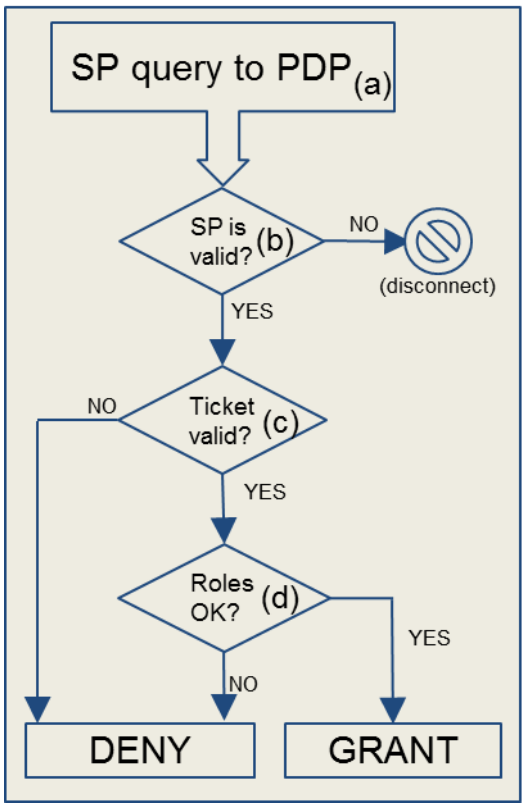

Figure 13. PDP algorithm used for authorisation process.

The proposed solution addresses issues mentioned earlier in this document such as the need to use multiple Identity Providers (IdP) including external (like Google), and allow foreign credentials delegation.

Access may be granted based on the required trust level by linking resources, actions and roles of the users. If access is denied, the user may automatically request it from the service manager.

6.3 Technologies

Choice of technologies is crucial for any IT system. The security system imposes special requirements as the technologies used need to be both mature enough to provide the required level of stability and security and should also be lightweight, at least for critical parts like the PEP.

After careful consideration we have decided to base our EurValve Security Web Platform on the Ruby on Rails framework. This mature platform allows rapid development of complex Web-based systems. In the realm of security, software stacks based on Rack and Rails offer a well-tested versatile security solution called Devise which allows various authentication methods, both local and integrated with external Identity Providers.

The PEP part needs to be adapted to the specific technologies used by the service itself, however as HTTP(S) based services are very common we plan to provide a universal PEP for this type of services. As the PEP is called constantly it needs to be as lightweight as possible. As a result, we decided to choose NGINX – a lightweight HTTP(S) server/proxy solution proven to efficient, stable and secure. The processing of the request will be done by an NGINX module written in the C language for maximum efficiency.

7 REAL-TIME MULTISCALE VISUALISATION

In this section analysis of existing solutions and recommendations for building a real-time visualisation system are presented. The final section contains a list of recommended technologies to be used for the implementation phase.

7.1 State of the art

A reliable visualisation facility working on-demand is one of the most important components of a computer-aided medical decision support system. Provided with a graphical insight into simulated phenomena the decision making process can be significantly improved.

Application deployment through web platforms has not been common for highly interactive 3D software with natively installed applications directly accessing the rendering hardware preferred. With the introduction of WebGL [WEBGL] technology this trend has changed and with the availability of supporting technologies such as WebSocket [WEBSOCKET] for data transmission and modelling abstraction libraries such as three.js [THREEJS] it is possible to utilise the web for robust and platform independent application delivery and build highly interactive decision support systems based on the enlisted stack.

More formal studies such as [DATACUBE] and practical implementations such as [GIMIAS] exist which allow visualisation of multiscale phenomena, however, this work focuses only on static data available locally to the rendering platform. For cloud-based simulation setups data for different model scales is produced in real time and possibly on distributed resources. Collecting the data and feeding the rendering process in the browser poses a challenge. Current solutions (e.g. ParaView [PARAVIEW]) focus on server-side rendering and transmitting raster images to be presented to the user which harms interactivity and introduces latency for high-resolution imaging.

The visualisation task in WP2 aims at implementing a basic infrastructure for visualisation, data extraction and storage, as well as at providing a set of visualisation widgets executed in a web environment. Design recommendations for this work are presented in the next section.

7.2 Design recommendations

In Figure 14 a visualisation pipeline is depicted. It shows how the data being visualised is obtained, stored and transferred for rendering. The first step of the process (left side on the diagram) is data extraction done by a dedicated data extractor deployed close to the process producing data. This process can be a simulation, a database or any code managing a parameter sweep and sensitivity analysis process. The transferred data is filtered to select only parts relevant for the visualisation. This may be relevant for applications which produce large amounts of data from which only small chunks are intended for rendering. The data extractor can be preinstalled on a container which handles the computations for easier integration by application providers.

Figure 14. Visualisation pipeline consists of a data extractor for retrieving data relevant for visualisation, visualisation server used for storing and buffering visualisation data and a visualisation widget which takes care of the final renderings.

In the second step of the pipeline the visualisation server processes the retrieved data by storing, buffering and transferring it to visualisation widgets in case a real-time insight is required. Before storing the data, the visualisation server may convert data formats to optimise storage utilisation and to pre-process the data to conform with data formats supported by the widgets. Storing of visualisation data will also improve rendering in the offline mode after the computing source has finished its work.

The last step of the pipeline takes place in the web browser where the transmitted data is rendered by a dedicated visualisation widget. Data sent by the server should already be in the supported format to minimise processing on the client side (browsers make only one thread available to the UI libraries). A combination of visualisation widgets may be used to present different aspects of the visualised process and appropriate configurations should be developed during the implementation phase of the EurValve platform.

Deployment scenarios of the visualisation system are depicted in Figure 16. Both environments envisioned by WP2 are considered. In blue all components external to the visualisation system are presented while the brown components are internal for the presented visualisation system.

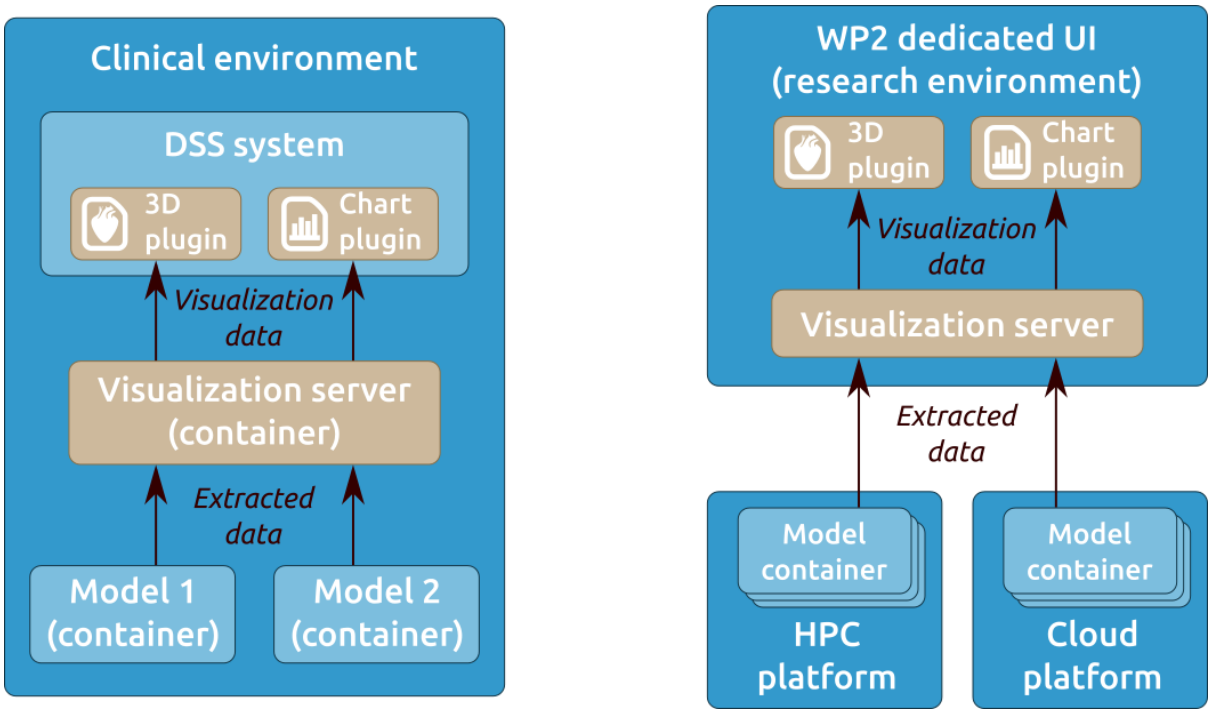

Figure 15. Deployment diagrams of the visualisation components in both clinical and research environments.

On the left the clinical environment may use the visualisation system package as a container for convenient deployment and embed the visualisation widgets by using the CEF framework (more in the next section) if a native UI toolkit is used for building the DSS. Data extractors are present in the containers handling model execution. It should be possible to duplicate the whole visualisation infrastructure if the administrative domain where the DSS is installed is closed to the outside networks. In critical cases where sensitive data cannot leave the premises of a single machine the deployment of the visualisation system should also be possible.

The research environment, managed by a web portal, should be able to utilise the visualisation system simply by embedding the provided visualisation widgets and using a centrally deployed visualisation server. Data extractors can be scattered among different computing platforms (cloud, HPC, etc.) and push data to a configured visualisation server (in practice a single server instance should suffice).

7.3 Recommended technologies

Depending on the type of visualisation different libraries can be used. For two-dimensional charts there exists a large number of plotting functions which run entirely in the Web Browser. Three-dimensional visualisation can also take place in the user's browser with the WebGL extensions introduced a few years ago. Finally, visualisation tools supporting concrete file formats (e.g. ParaView) are used to enrich the end-user visualisation experience. The table below gives an overview of libraries which could potentially be used in the EurValve platform.

Name	Function in EurValve	Setup/mode of operation
Highcharts, D3.js, Chart.js	These libraries can be used to visualise any process or data which can be presented as a two-dimensional chart. There are many chart types available (e.g. for sensitivity analysis results the so called tornado diagrams can be easily created using bar charts).	The libraries are standalone JavaScript modules embedded into web applications using a script tag within a web page. If it is required they can be used together to support different types of charts. Also, depending on the integration framework used wrappers exist for convenient use (e.g. Google Web Toolkit or scala.js wrappers).
Three.js	For three-dimensional visualisations the three.js wrapper can be used over WebGL. The library can be used to present simulated valve blood flows or to show wall shear stress with colour gradients. In cases where high demand on graphics is required two-dimensional visualisations with hardware support can also be handled with this library.	Embedding the library is similar to chart libraries. It is however important to ensure that the underlying hardware system is equipped with a hardware-accelerated graphics card. Whilst fall-back to software acceleration is supported the final renderings may differ as not all 3D features are supported.

Name	Function in EurValve	Setup/mode of operation
ParaViewWeb	For visualising VTK files the ParaView library can be used. Usually segmentation results are stored using this file format. The library also handles other formats such as CSV and is able to render point geometries.	The library requires a server-side component to be deployed in the same domain in which the browser counterpart is operating. The performance is dependent on the connection bandwidth so large renderings may be sluggish.
WebSockets	Real-time data streaming can be handled by this library to minimise overheads and latency of the HTTP protocol. A two-way connection is available between a web browser and the server after establishing a WebSocket link. This mode of operation can be used to visualise long-running computation tasks such as parameter sweep analysis where a constant monitoring panel is required.	WebSocket extension is available in all web browsers and can be used to feed data to the visualisation widgets. Various web frameworks support it with a set of dedicated extensions. Special care needs to be devoted to proxy server configurations as they do not always correctly forward WebSocket protocol negotiation headers.
Chromium Embedded Framework (CEF)	The visualisation widgets developed in task 2.4 can be executed in a web environment which basically requires a working web browser. If integration with a desktop library is required, the CEF component allows embedding of web components in native applications. Support for WebGL and all web APIs is ensured and integration wrappers for various desktop frameworks are available.	Deployment of CEF components can be done with an external or internal web server in place to serve the web content. The internal approach makes the final application independent of any external servers, however, it requires that all web content is bundled with the application. An approach with an external server simplifies web content updates and allows for small application packages.

All presented libraries are open source software with active community support. No additional fees are required for use in open source and non-profit projects. The CEF framework listed above makes the visualisation components developed in task 2.4 usable in native applications so that any EurValve development platform can be easily integrated.

A significant development challenge in the scope of providing a real-time visualisation system id the integration of the web-based components with desktop applications through solutions such as CEF. Depending on the native technology used the CEF wrappers supported by a community-based teams have different levels of software maturity and may become an issue that not all the features allowing for seamless web application embedding are supported. To mitigate this a semi-integration approach may be used to simply provide hyperlinks in the native application which will point to the required visualisation view in the web browser installed separately on the user's computer. Communication with the visualisation widget, if needed, can be established through the visualisation server.

8 DESIGN OF RESEARCH AND CLINICAL EXECUTION ENVIRONMENTS

8.1 Interfaces to the Model Execution Environment

The Model Execution Environment should support a wide range of interaction types, as required by the application users. The nature of the tasks performed in the research environment includes both interactive and batch processes, some of which need to be invoked manually, whilst others can and should be automated, as they require repetitive use of similar computing steps (e.g. in the case of sensitivity analysis). A typical workflow of the user is thus to invoke some computations on the cluster, fetch the data to analyse it locally, launch an interactive cloud service to process the selected data, subsequently run a batch of jobs on the resulting datasets, etc. This combination of interactions poses a significant challenge to the design of the execution environments, since the research process in this case is exploratory and requires manual intervention at multiple phases, which precludes the use of fully automated tools such as scientific workflow engines or science gateways. We need thus to design the system as a bag of tools that can be easily integrated ad-hoc to the current needs of skilled users, who can then use scripting tools to automate some of the processing steps, whilst also taking advantage of Web interfaces to access the data and services in a user friendly way if needed. The important aspect will be to provide a unified security mechanism so that the access to all the services will be possible in a seamless way using a single sign-on mechanism.

In order to meet these objectives to build a flexible set of tools, we propose to build the Model Execution Environment using a set of interfaces:

- **REST application programming interfaces.** All the services such as access to HPC (Rimrock), access to cloud (Atmosphere) and data (PLG-Data) should provide RESTful APIs with a unified authentication mechanism. This will provide a standard way of accessing the services and building the higher-level client interfaces.
- **Command-line interfaces**. This is an optional interface, facilitating the access to the services for advanced users who are used to working with HPC resources. Job submission and data management tasks should be simplified by providing a clear and coherent CLI interface. The CLI can support syntax completion and various output formats (tabular, CSV, JSON, etc.) so that it can be easily used in scripts which automate repetitive tasks. The CLI implementation should be based on the REST interface.
- **Client-side API**. If needed, the services can provide APIs in popular scripting languages and high-level languages (e.g. Python, Java). We consider this interface optional, since there are a large variety of languages and unless there is a common agreement on the preferred programming language used in the project, we are not planning to implement these APIs.
- **Web User Interfaces.** For easy access to services and data, we plan to provide a Web UI for the selected services. E.g. PLG-Data provides a web browser of user files, or using Rimrock it is possible to create a Web component showing running jobs, etc. We consider Web interfaces optional, since there is currently no explicit requirement regarding portal access to all the tools and services in the project. Since

the REST interfaces provide a natural way of building Web user frontends, it will be possible to create such UIs if required.

In order to make all these interfaces coherent and easy to compose into the application-specific workflows, we must ensure that common security will be used, and that the proposed interfaces (REST, CLI, API, Web UI) complement each other. E.g. the URLs to files returned from the data access services should be usable in the job submission system, etc. Moreover, when designing the REST, CLI or Web UIs, we should enable their interplay, e.g. from the Web UI the users should be able to generate the relevant REST request, or a command to execute in a CLI, while the REST API or CLI should return URLs that can be directly opened in the Web browser for user friendly visualisation or interaction.

Based on the current requirements, we plan to provide the REST interfaces first, together with selected Web UIs to demonstrate their usage. During the course of the project and based on the requirements of the users, we will be able to provide additional interfaces (CLI, API, Web UI) taking into account their priorities and the resources available in WP2.

The architecture of the environment together with the interfaces is shown in Figure 16.

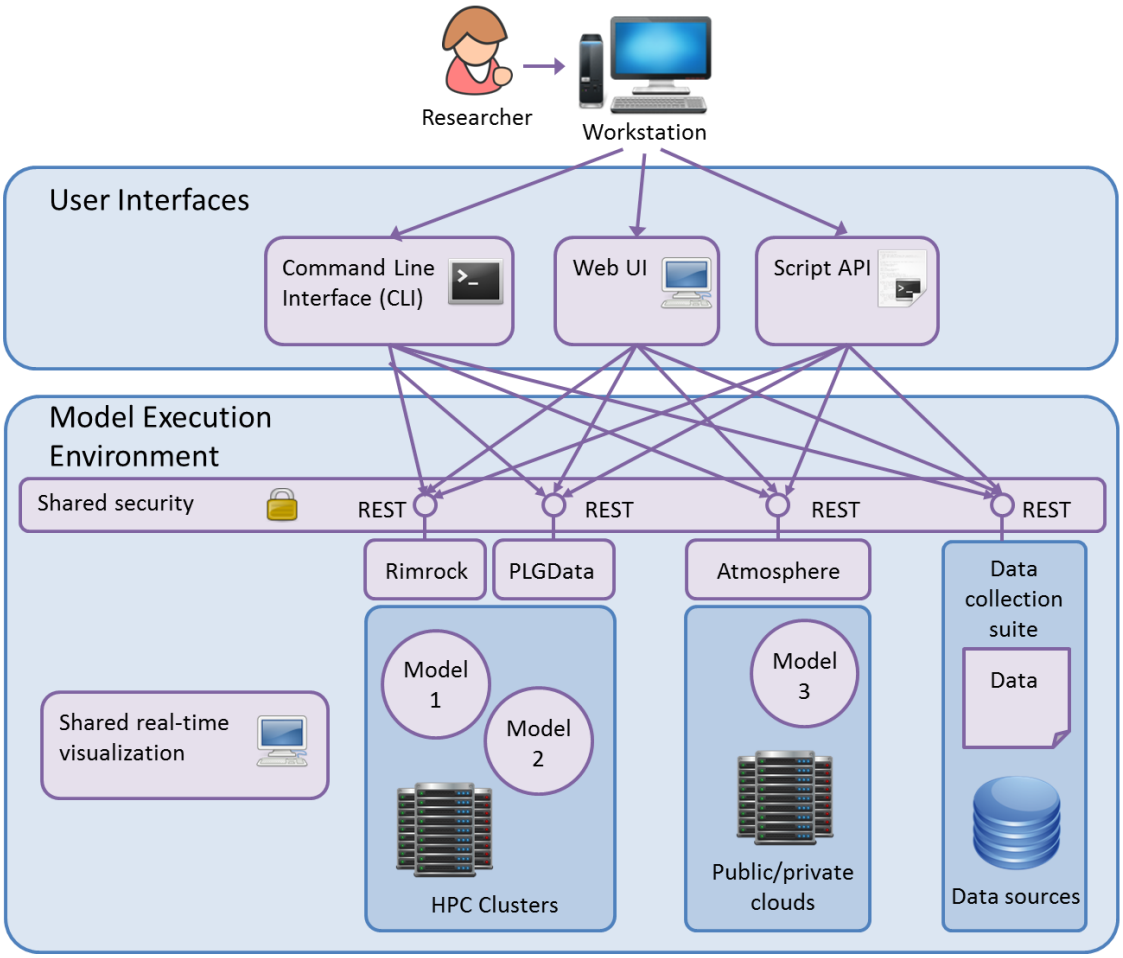

Figure 16. Architecture of the Model Execution Environment

8.2 Summary of technologies recommended

Table 1 below presents the selected technologies that will constitute the Model Execution Environment. We give their purpose, reference, a short description, the current usage and deployment status, as well as interfaces available. These building blocks will be used as a basis for creating ad-hoc or more advanced solutions supporting the execution of models in the project.

Table 1 Main technologies for model execution environment

Purpose	Tool	Description	Usage	Interface
Access to HPC resources	**Rimrock**http://dice.cyfronet.pl/products/rimrock	Robust Remote Process Controller. It allows executing applications in a batch or interactive mode using a simple REST interface.	Rimrock has been deployed into production in PL-Grid infrastructure after passing the operations and security audits.	REST, Web UI
Access to cloud resources	**Atmosphere** http://dice.cyfronet.pl/products/atmosphere	Atmosphere integrates resources from privately-deployed open-source cloud platforms (e.g. OpenStack) and commercial IaaS providers (e.g. Amazon EC2).	Within the VPH community Atmosphere is used in a production environment, providing access to cloud services for VPH-Share project and is the main interface to cloud in PL-Grid.	REST, Web UI
Data access for HPC infrastructure	**PLG-Data**https://data.plgrid.pl/?locale=en	A tool to access the folders and files that are available in the PL-Grid Infrastructure. These include both personal resources and also those shared by collaborators.	Within PL-Grid infrastructure PLGData has been deployed as a web service that uses PLGrid's proxy certificates and GridFTP protocol to deliver file management platform for Zeus and Prometheus clusters.	REST, Web UI

In addition to these core technologies of which we have previous experience, we recommend use of the following technologies as presented in Table 2.

Table 2 List of other recommended technologies

Name	Function in EurValve
Nginx/Apache WebDAV module	The modules for setting up a testing infrastructure for preliminary integration of the WebDAV repository with the authorisation framework.
Apache Tomcat WebDAV servlet	May be used as the basis for the final implementation of the File Store component.
Apache Jackrabbit	May be used as the basis for the final implementation of the File Store component.
GSI	Accessing HPC resources. Providing user credentials delegation mechanism.
PL-Grid account management	Accessing HPC resources. Providing authentication and authorisation for the HPC part of EurValve MEE.
GridFTP	Transferring large files to/from the HPC storage.
RimRock	Submitting simulation jobs to a HPC cluster.
PLGData	File management directly on HPC storage.
AWS or Google	Commercial cloud services
OpenStack	Building and managing private cloud infrastructure
Atmosphere	Federating cloud resources and cloud application life-cycle management
Docker	Container execution and management
Docker compose	Multi-containers applications management
Docker swarm	Containers cluster management
Docker hub	Repository for Docker container images
Highcharts, D3.js, Chart.js	Visualisation of any process or data which can be presented as a two-dimensional chart
Three.js	Three-dimensional visualisations
ParaViewWeb	Visualising VTK files
WebSockets	Real-time data streaming
Chromium Embedded Framework (CEF)	For integration with a desktop library

9 SOFTWARE DEVELOPMENT METHODS AND QUALITY ASSURANCE RECOMMENDATIONS

All software to be developed in the context of the computational infrastructure presented in this deliverable – in particular the Research Computational Environment (RCE) and its constituent components – will be subject to modern Computer-Aided Software Engineering (CASE) mechanisms and tools facilitating collaborative development work and ensuring consistent quality of the resulting code. It is the intent of WP2 developers to develop the RCE as an open-source project (even though the software which implements selected models processed by the RCE may be proprietary and subject to copyright restrictions). To this end, WP2 intends to pursue the following approach:

- **Use a shared code repository**. This will be based on the Git versioning and collaborative development platform – the current *de facto* standard in development of shared IT systems by geographically distributed teams.
- **Where feasible, provide early prototypes**. As in any scientific project with substantial IT involvement, the exact requirements of end users (exact in terms of the specific functionality of each software component to be developed) may be difficult to elucidate in advance, and new requirements may emerge as the project progresses. In order to accommodate this variability, WP2 will attempt to implement an early prototyping approach where the core components of the platform are made available at an early stage of the project in "draft" form and then fleshed out with additional features as required.
- **Create a separate development and production testbed**. While ongoing implementation work on the RCE will require a development "playground" for service developers, we are also aware of the need to provide a stable production platform upon which other Work Packages of the EurValve project may base their work. Consequently, we intend to split the available hardware infrastructure into two subsets, one of which (consisting of a rudimentary set of representative resources) would be used for development and testing, while the other will power the production platform. New versions of the production platform would be rolled out at regular intervals, along with a concise documentation of the available features and changes.
- **Use automated mechanisms for ensuring code quality**. EurValve will be based, to an extent, on existing software and technology stacks. This necessitates consistent management of software dependencies. Additionally, WP2 intends to make use of a variety of automatic CASE plug-ins offered by popular Git platforms – see below for a description of several tools which we currently plan to utilise.
- **Institute a code review procedure based on push requests**. All development will be done in a separate Git branch and the developer is encouraged to create a push request immediately upon forking the branch. Once a push request is created, free tools integrated with the Git repository may kick in (as remarked above). We intend to use Hound to validate code formatting. This allows us to maintain a unified format so that other developers don't have to focus on this aspect during the review. The next tool currently proposed for use is Travis CI. It builds working versions of the platform and executes unit and integration tests every time code is pushed to the repository. At the end of each test, we propose calculating code coverage and code quality using Codeclimate. Once a given branch is ready to be merged, the developer can add a "ready

for review" label to their push request and notifies the team to begin the review. Internal code reviews are a continuous process which lasts until all issues reported by the team are fixed. A push request can be merged into master when all test and integration tests are green and at least two other developers accept the changes by issuing +1s in the push request comment section.

- **Keep track of infrastructure status with monitoring tools.** Alongside each version of the platform (the development version as well as the production version) we will provide a monitoring framework with which each deployment can communicate in order to track issues and report any errors. The specific tools which we plan to integrate with the RCE platform include Sentry (to collect and aggregate information about any platform error) and New Relic (to track platform performance).

10 SUMMARY

The document presents a detailed introduction to the basic components of the EurValve computational platform, in both its incarnations – the Research Computing Environment and the Decision Support System. Information contained in this document will drive implementation of the first (beta) version of the platform, which is scheduled for deployment by Project Month 15.

This presented design is intended as the preliminary step towards implementation of EurValve infrastructural components, and is expected to evolve over time as new requirements come to light and existing requirements are refined.

An updated design of the infrastructure will be published alongside its beta release, in Month 15 of the Project (in Deliverable 2.4).

11 REFERENCES

[ATMOSPHERE] Atmosphere Cloud Platform on GitHub, https://github.com/dice-cyfronet/atmosphere (access 10.05.2016)

[AWS] Amazon Web Services (AWS) - Cloud computing services, https://aws.amazon.com/products/ (access 10.05.2016)

[AWS-LAMBDA] AWS Lambda homepage, http://aws.amazon.com/lambda (access 10.05.2016)

[AZURE] Microsoft Azure: Cloud Computing Platform and Services, https://azure.microsoft.com (access 10.05.2016)

[CMUJ] Jagiellonian University Medical College homepage, https://www.cm-uj.krakow.pl/indexen.php (access 10.05.2016)

[CORS] W3C. Cross-Origin Resource Sharing. [Online].; 2016. Available from: http://www.w3.org/TR/cors/.

[DATACUBES] Stolte, C.; Tang, D.; Hanrahan, P., "Multiscale visualization using data cubes," Visualization and Computer Graphics, IEEE Transactions on , vol.9, no.2, pp.176,187, April-June 2003, doi: 10.1109/TVCG.2003.1196005

[DOCKER] Docker homepage, https://docker.com (access 10.05.2016)

[DOCKER-COMPOSE] Docker compose documentation, https://docs.docker.com/compose (access 10.05.2016)

[DOCKER-HUB] Docker hub homepage, https://hub.docker.com (access 10.05.2016)

[DOCKER-SWARM] Docker swarm documentation, https://docs.docker.com/swarm (access 10.05.2016)

[DOUGLAS2015] Douglas M. Jacobsen, Richard Shane Canon. Contain This, Unleashing Docker for HPC. Cray User Group 2015 (CUG 2015) Proceeding, April 2015.

[DROPBOX] Dropbox. Dropbox. [Online]. Available from: https://www.dropbox.com.

[GFTP] W.E. Allcock, I. Foster, R. Madduri. Reliable Data Transport: A Critical Service for the Grid. Building Service Based Grids Workshop, Global Grid Forum 11, June 2004.

[GIMIAS] Debora Testi, Daniele Giunchi, Gordon Clapworthy, Stephen Aylward, Xavier Planes, and Richard Christie. 2012. New interactive visualisation of multiscale biomedical data. In ACM SIGGRAPH 2012 Posters (SIGGRAPH '12). ACM, New York, NY, USA, , Article 76 , 1 pages. DOI=10.1145/2342896.2342987 http://doi.acm.org/10.1145/2342896.2342987

[GLOBUS] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp. 2-13, 2006.

[GLXY] B. Giardine, C. Riemer, R.C. Hardison, R. Burhans, L. Elnitski, P. Shah, Y. Zhang, D. Blankenberg, I. Albert, J. Taylor, W. Miller, W.J. Kent, A. Nekrutenko. Galaxy: a platform for interactive large-scale genome analysis. Genome Research 15(10), pp. 1451-5, 2005

[GOOGLE-SERVICES] - Products and services - Google Cloud Platform, https://cloud.google.com/products/ (access 10.05.2016)

[GSI] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. A Security Architecture for Computational Grids. Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.

[HEAT] Heat: OpenStack orchestration, https://wiki.openstack.org/wiki/Heat (access 10.05.2016)

[HTTP] Group NW. Hypertext Transfer Protocol -- HTTP/1.1. [Online].; 1999. Available from: https://tools.ietf.org/html/rfc2616.

[ISL] J. Kocot, T. Szepieniec, P. Wójcik, M. Trzeciak, M. Golik, T. Grabarczyk, H. Siejkowski, M. Sterzel. A Framework for Domain-Specic Science Gateways. In: M. Bubak, J. Kitowski, K. Wiatr (eds.) eScience on Distributed Computing Infrastructure, Achievements of PLGrid Plus. LNCS, vol. 8500, Springer pp. 130-146, 2014

[ISMOP] ISMOP - Computer System for Monitoring River Embankments, http://www.ismop.edu.pl/en (access 10.05.2016)

[KNIEP] Christian Kniep. Containerization of High Performance Compute Workloads using Docker, November 2014, available onliehttp://doc.qnib.org/2014-11-05_Whitepaper_Docker-MPI-workload.pdf (access 10.05.2016)

[KUBERNETES] Kubernetes homepage, http://kubernetes.io (access 10.05.2016)

[LOBCDER] Koulouzis S, Vasyunin D, Cushing R, Belloum A, Bubak M. Cloud Data Federation for Scientific Applications. In Euro-Par 2013: Parallel Processing Workshops.: Springer Berlin Heidelberg; 2014. p. 13-22.

[NICKOLOFF] Jeff Nickoloff. Evaluating Container Platform at Scale, available online https://medium.com/on-docker/evaluating-container-platforms-at-scale-5e7b44d93f2c#.j2p8rlty8 (access 10.05.2016)

[OPENSTACK] OpenStack Open Source Cloud Computing Software, https://www.openstack.org/ (access 10.05.2016)

[OWNCLOUD] ownCloud. ownCloud. [Online]. Available from: https://owncloud.org/.

[PARAVIEW] Kitware. ParaView. [Online]. Available from: http://www.paraview.org.

[PLGAPP] Plgapp Scientific application development made easier, https://app.plgrid.pl(access 10.05.2016)

[PLGD] PLG-Data - a simple tool for files management inPLGrid, https://data.plgrid.pl(access 10.05.2016)

[PLGRID] Atmosphere in PLGrid, https://cloud.plgrid.pl (access 10.05.2016)

[PROXY] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, F. Siebenlist. X.509 Proxy Certificates for Dynamic Delegation. 3rd Annual PKI R&D Workshop, 2004.

[QCG] B. Bosak, P. Kopta, K. Kurowski, T. Piontek, M. Mamoński, New QosCosGrid Middleware Capabilities and Its Integration with European e-Infrastructure, In eScience on Distributed Computing Infrastructure, Springer, pp. 34-53, 2014

[RACKSPACE] Rackspace: Managed Dedicated and Cloud Computing Services, https://www.rackspace.com/ (access 10.05.2016)

[REST] Fielding RT, Taylor RN. Principled design of the modern Web architecture. In ICSE '00 Proceedings of the 22nd international conference on Software engineering. New York: ACM; 2000. p. 407-416.

[RR] Rimrock Robust Remote Process Controller, https://submit.plgrid.pl(access 10.05.2016)

[SLURM] M. Jette and M. Grondona. Slurm: Simple Linux Utility for Resource Management. Proceedings of ClusterWorld Conference and Expo, San Jose, California, June 2003.

[SSH] Securre shell access protocol, https://pl.wikipedia.org/wiki/Secure_Shell, (access 10.05.2016)

[TERRA-FORM] TerraForm homepage, https://www.terraform.io (access 10.05.2016)

[THREEJS] three.js. three.js library. [Online]. Available from: http://threejs.org.

[TORQUE] Torque Resource Manager, http://www.adaptivecomputing.com/products/open-source/torque(access 10.05.2016)

[TOSCA] OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca (access 10.05.0216)

[UNICORE] K. Benedyczak, M. Stolarek, R. Rowicki, R. Kluszczynski, M. Borcz, G. Marczak, M. Filocha, P. Bała. Seamless access to the PL-Grid e-infrastructure using UNICORE middleware. In: M. Bubak, T. Szepieniec, K. Wiatr (eds.) Building a National Distributed e-Infrastructure–PL-Grid, Springer, pp. 56-72, 2012

[UNIP] M. Petrova, V. Huber, B. Demuth, K. Benedyczak, B. Schuller. The Unicore Portal. UNICORE Summit 2013: Proceedings, Leipzig, Germany, pp. 47-54, 2013

[VPH-DARE] Virtual Physiological Human: Dementia Research Enabled by IT homepage, http://www.vph-dare.eu/ (access 10.05.2016)

[VPH-SHARE] About Vph-Share, https://portal.vph-share.eu/about/ (access 10.05.2016)

[VPH-SHARE-D2.1] M. Bubak, T. Bartyński, M. Kasztelnik, M. Malawski, J. Meizner, P. Nowakowski, S. Koulouzis, D. Chang, S. Zasada, E.Sarries. Data and Compute Cloud Platform Deliverable: D2.1 Analysis of the State of the Art; Work Package Definition, VPHShare project report, http://dice.cyfronet.pl/projects/details/VPH-Share-files/deliverables/vph-share-wp2-d2.1-v13.pdf

[WEBDAV] Network Working Group. HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). [Online].; 2007. Available from: http://www.webdav.org/specs/rfc4918.html.

[WEBDAVBIND] Internet Engineering Task Force. Binding Extensions to Web Distributed Authoring and Versioning (WebDAV). [Online]. Available from: https://tools.ietf.org/html/rfc5842.

[WEBGL] KHRONOS. OpenGL ES 2.0 for the Web, [Online]. Available from: https://www.khronos.org/webgl.

[WEBSOCKET] Internet Engineering Task Force. The WebSocket Protocol. [Online]. Available from: https://tools.ietf.org/html/rfc6455.

LIST OF KEY WORDS/ABBREVIATIONS

CASE	Computer-Aided Software Engineering
CFD	Computational Fluid Dynamics
CI	Continuous Integration
CORS	Cross-Origin Resource Sharing
DSS	Decision Support System
GSI	Grid Security Infrastructure
HPC	High-Performance Computing
HTTP	Hypertext Transfer Protocol
MEE	Model Execution Environment
PDP	Policy Decision Point
RCE	Research Computing Environment
REST	Representational Stata Transfer
SLURM	Simple Linux Utility for Resource Management
SSH	Secure shell
SSO	Single Sign-On
URL	Uniform Resource Locator
VHD	Valvular Heart Disease