ICT 269978

Integrated Project of the 7th Framework Programme

COOPERATION, THEME 3
Information & Communication Technologies
ICT-2009.5.3, Virtual Physiological Human

Work Package: WP2

Data and Compute Cloud Platform

Deliverable 2.4

Second Prototype of the Cloud Platform

Version: 1v0

Date: 28-Feb-13

DOCUMENT INFORMATION

IST Project Num	FP7 – ICT - 269978	**Acronym**	VPH-Share
Full title	Virtual Physiological Human: Sharing for Healthcare – A Research Environment		
Project URL	http://www.vphshare.org		
EU Project officer	Robert Begier		

Work package	**Number**	2	**Title**	Data and Compute Cloud Platform
Deliverable	**Number**	Deliverable 2.4	**Title**	Second Prototype of the Cloud Platform

Date of delivery	**Contractual**	29-Feb-12	**Actual**	19-Mar-12
Status	Version 1v0		Final ☐	
Nature	Prototype ☒ Report ☐ Dissemination ☐ Other ☐			
Dissemination Level	Public (PU) ☐ Restricted to other Programme Participants (PP) ☐ Consortium (CO) ☒ Restricted to specified group (RE) ☐			

Authors (Partner)	CYFRONET, UCL, UvA, AOSAE			
Responsible Author	Piotr Nowakowski		**Email**	p.nowakowski@cyfronet.pl
	Partner	CYFRONET	**Phone**	+48600280105

Abstract (for dissemination)	This document details the features and technologies used in the implementation of the second prototype of the VPH-Share cloud management platform. It lists the status of each component produced by WP2, ongoing work in each technical task and the specifics of integration with external Work Packages.
Keywords	cloud computing, data storage federation, high performance computing, hybrid clouds

Version Log			
Issue Date	**Version**	**Author**	**Change**
05.12.2012	0.1	Piotr Nowakowski	Initial draft
05.02.2013	0.5	Piotr Nowakowski (CYF; WP2), Marian Bubak (CYF; WP2), Tomasz Bartyński (CYF; T2.1), Tomasz Gubała (CYF; T2.1), Daniel Harężlak (CYF; T2.1), Jan Meizner (CYF; T2.2), Marek Kasztelnik (CYF; T2.7); David Chang (UCL; T2.3), Stefan Zasada (UCL; T2.3), Spiros Koulouzis (UvA; T2.4), Dmitry Vasunin (UvA; T2.4), Krzysztof Styrc (CYF; T2.5); Dario Ruiz Lopez (AOSAE; T2.6); Rodrigo Diaz Rodrigues (AOSAE; T2.6)	Review-ready version
25.02.2013	1.0	Piotr Nowakowski; Marian Bubak; Susheel Varma, Debora Testi	Final version

Contents

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

This document presents the current state of development of Work Package 2 tools and services in the context of the VPH-Share project. The goal of this document is to summarize the existing features of the Cloud application development, enactment and data storage platform, explain how the prototype can be used to deploy existing workflow services and present possible use cases which are already supported by the prototype. In addition, the document outlines future development plans in Work Package 2.

During the second year of the Project, Work Package 2 was able to successfully integrate and deploy an extended version of the cloud platform prototype, as well as utilize it to support a number of VPH-Share application workflows in production mode. The provisioning of this prototype should be viewed as the main achievement of WP2 in the presented period. The presented system provides clear added value both for service developers, supporting them at every step of the service implementation and deployment process, and for end users, by abstracting away all operations which deal with instantiation and management of service instances on cloud-based hardware resources.

More specifically, the following features are now available to end users:

- In Task 2.1 the second year of development focused on implementing user features to fully support the lifecycle of Atomic Services, including management of development versions, storage and management of security keys, adding redirections and publishing application services. T2.1 now provides a full-fledged Atomic Service development environment where new application services can be deployed in the cloud;
- Task 2.2 devoted its attention to extending and upgrading the cloud computing platform supporting VPH-Share services. New computational and storage resources were added, and the infrastructure was migrated to the current release of the OpenStack framework. T2.2 now oversees the functioning of a production-ready infrastructure composed of hardware resources running the OpenStack cloud computing and storage middleware, upon which Atomic Services can be instantiated and used;
- In year 2 Task 2.3 focused on providing a more robust set of programmer's interfaces and integration with project-wide security mechanisms. T2.3 provides a service extension for a set of grid-based high performance computing resources accessible via the Application Hosting Environment;
- In Task 2.4 the metadata storage and management system was reimplemented and extended to support extended descriptions of LOBCDER items. T2.4 now operates a hybrid data storage infrastructure where individual users of the VPH-Share project, as well as applications acting on their behalf, can store and share sensitive data in a secure manner;
- Task 2.5 focused on work on deploying a production-ready monitoring service, integrated with the newest version of LOBCDER. T2.5 now operates a data accessibility and integrity validation service integrated with the platform;
- In year 2 Task 2.6 ensured integration with the OpenID security mechanisms available through the VPH-Share Master Interface and initiated work on management of per-service security policies, including dynamic injection of policies into existing services. As

a result, T2.6 has successfully implemented and provisioned a project-wide authentication and authorization model for VPH-Share services.

Based on the results outlined further on in this document we are confident that WP2 has successfully accomplished all the achievements which make up milestone MS7, as specified in the Project's Description of Work: Verification (Beta release available through PhysiomeSpace; formal report published; project demonstration of Beta release at the 2nd Annual Review).

The scientific results of WP2 have been showcased at the VPH Conference in London (1), eScience'12 in Chicago (2), and at CGW12 in Kraków (3).

This deliverable is meant as a follow-up to Deliverable 2.3, published at the end of Project Month 12 and recapitulates the work that has occurred in the context of WP2 since the end of the design phase. Any deviations from the initial design, whether due to evolving user requirements or discovery of additional technical possibilities, are listed and explained, as is their impact on the overall Project architecture.

The following topics are addressed in this document:

- Features provided by the cloud platform for its users;
- Implemented components of the platform, including cloud computing infrastructure and management tools, data storage federation, HPC frontend and data validation components;
- Technologies applied in the implementation of each component and tool;
- Software engineering methods used.

As the VPH-Share cloud platform is still considered to be a prototype, the user guide section of this deliverable focuses on Consortium members; particularly those representing Work Packages 3, 4 and 5. Nevertheless, the relevant sections of this document are meant as a user-friendly introduction to the functionality of the cloud platform and we do not expect readers to be intimately familiar with the specific features of cloud middleware systems.

This document is divided as follows:

- Section 1 introduces the WP2 prototype by briefly explaining the goals of WP2 and referencing its design document (Deliverable 2.2). An updated architecture diagram is presented and discussed. In addition, this section contains short, introductory descriptions of the work performed in each of technical tasks of WP2.
- Section 2 is meant as a guide for end users (i.e. application workflow teams and Atomic Service maintainers), explaining the use of graphical user interfaces exposed by the VPH-Share Cloud Platform via the Master Interface.
- Section 3 contains an in-depth manual for developers of platform services (particularly representatives of Work Packages 3, 4 and 6), explaining programmatic access to the VPH-Share Cloud Platform with the use of APIs.
- Section 4 contains descriptions of the implementation progress achieved in each technical task of WP2 as well as the features supported by the second prototype (released in Project

Month 24). A subsection has been prepared for each clearly identifiable technical component of WP2, explaining its features, status of prototype releases and ongoing work. In addition, any deviations from the original design (if present) are highlighted and explained.

- Section 5 focuses on software engineering aspects relevant to the development of the Cloud Platform, including remarks on testing and development methodologies applied in the course of the project.
- Section 6 outlines the plans for the third year of development in each technical task of Work Package 2. It serves as a quick primer to WP2 year 3 roadmap; more elaborate descriptions of ongoing and future work can be found in Section 4.
- Section 7 contains closing remarks.

1 Features of the VPH-Share Cloud Platform Prototype

As explained in the Project's Technical Annex (4) and discussed in detail in deliverable D2.2 (5), the goal of Work Package 2 is to provide a backbone for the VPH-Share computational and data storage services, enabling the application workflow representatives from Work Package 5, as well as the data storage and management teams from Work Package 4 to deploy their respective applications and data sources within a Cloud-based distributed computing infrastructure. Specifically, Work Package 2 fulfils the following goals:

- Integrate and oversee the Cloud-based computing and data storage resources made available to the Project;
- Expose a unified, heterogeneous, secure Cloud computing platform where application services and binary data repositories can be deployed and made available to authorised users without the need to install high-performance computing hardware at said users' local sites;
- Facilitate the development and deployment of the VPH-Share Atomic Services (please refer to deliverable D2.1 (6) for an explanation of what constitutes an Atomic Service and what features it offers to end users) on the Cloud resources managed by WP2;
- Provide extensions for traditional (Grid-based) high performance computing resources wherever required by application services;
- Facilitate access to potentially sensitive data sets by managing them with the use of the available Cloud data storage and monitoring their integrity, consistency and availability with automated, configurable tools;
- Collaborate with application and data storage providers as well as with partner projects (specifically, p-medicine) on providing an open, extensible infrastructure into which additional application services can be imported with minimum effort.

Whereas deliverable D2.2 provides an in-depth description of each technical component of the Work Package, the goal of this section is to explain which features (presented in D2.2) have been implemented and are integrated in the second prototype. In order to focus this discussion, we would like to refer the reader to an updated version of the Work Package 2 architecture diagram, which is presented in Figure 1.

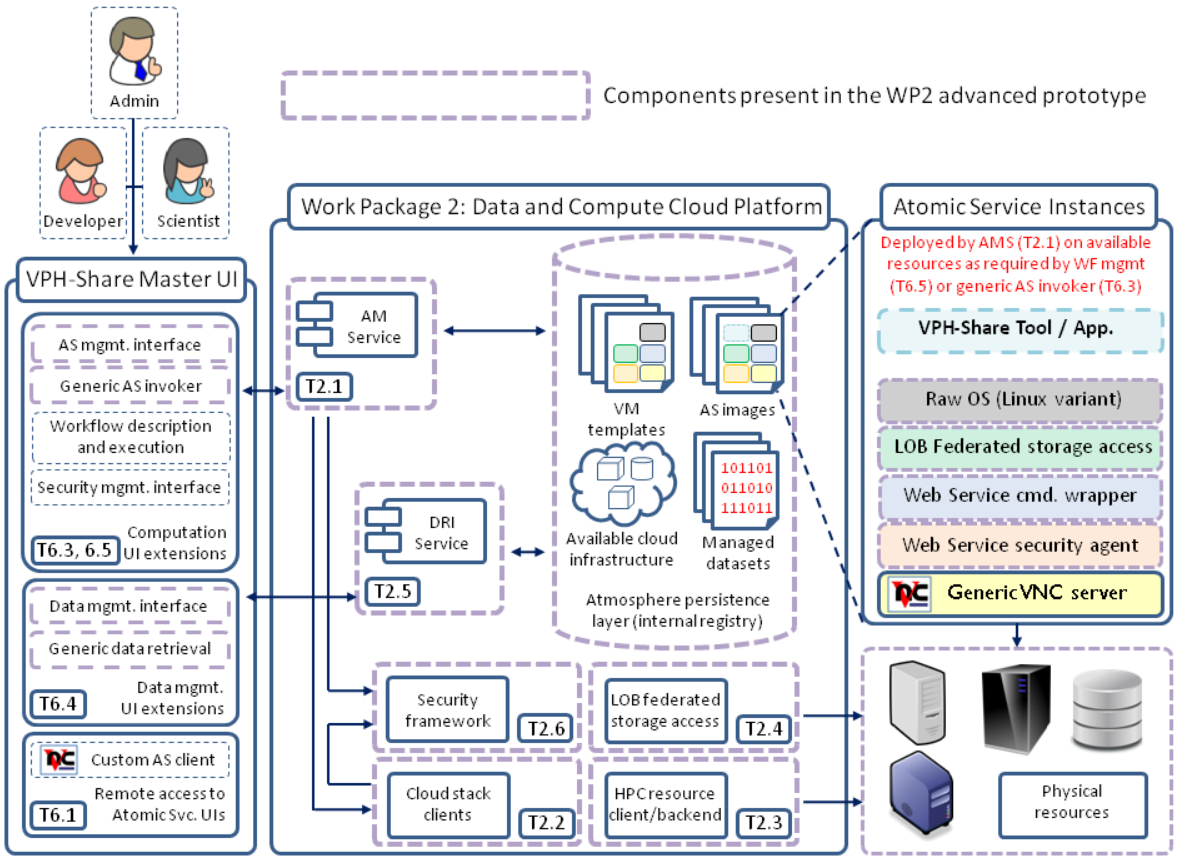

Admin
Developer
Scientist
Components present in the WP2 advanced prototype
VPH-Share Master UI
AS mgmt. interface
Generic AS invoker
Workflow description and execution
Security mgmt. interface
T6.3, 6.5
Computation UI extensions
Data mgmt. interface
Generic data retrieval
T6.4
Data mgmt. UI extensions
Custom AS client
Remote access to Atomic Svc. UIs
T6.1
Work Package 2: Data and Compute Cloud Platform
AM Service
T2.1
DRI Service
T2.5
VM templates
AS images
Available cloud infrastructure
Managed datasets
Atmosphere persistence layer (internal registry)
Security framework
T2.6
LOB federated storage access
T2.4
Cloud stack clients
T2.2
HPC resource client/backend
T2.3
Atomic Service Instances
Deployed by AMS (T2.1) on available resources as required by WF mgmt (T6.5) or generic AS invoker (T6.3)
VPH-Share Tool / App.
Raw OS (Linux variant)
LOB Federated storage access
Web Service cmd. wrapper
Web Service security agent
Generic VNC server
Physical resources

Figure 1: VPH-Share Work Package 2 architecture as of Project Month 24 (conceptual view).

The central part of the diagram lists the WP2 application components and explains how these components map to technical tasks in WP2, according to the Project's Description of Work (4).

Following the initial design period, concluded in Month 6, the technical tasks have commenced implementation of their assigned components, as shown in Figure 1. A general outline is presented below:

- The Cloud Resource Allocation Management framework, known as Atmosphere, is capable of supporting the full development cycle of Atomic Services, starting with selection of OS templates, all the way to sharing and reusing full-fledged AS'.
- The Cloud Execution Environment in place at CYFRONET has been expanded to include 32 four-CPU nodes to which VPH-Share Atomic Service Instances can be deployed. A major upgrade effort has resulted in successful installation of the newest releases of the OpenStack platform (specifically, the Folsom release which became available in September 2012). This cloud site is now fully capable of serving production runs of VPH-Share applications. An upgraded mass storage directory is provided and can be interfaced with the use of the OpenStack Swift protocol.

- As described in deliverable D2.2, VPH-Share application workflows may now delegate HPC computational jobs to legacy grid infrastructures with the use of the AHE service backend.
- The data access framework, known as LOBCDER, is now capable of exposing attached storage resources (including Swift-managed mass storage) as a emulated WebDAV (Web-based Distributed Authoring and Versioning) directory, which can in turn be mounted and accessed as local file systems on the virtual machines used to host Atomic Service Instances. Successful tests have concluded regarding the use of LOBCDER to host and expose application-specific data.
- The Data Reliability and Integrity tool can register and periodically monitor managed datasets in the WP2 framework, ensuring their consistency and availability. In addition, notification mechanisms have been put in place to warn administrators of any data access problems encountered by the validation tools. Finally, a dedicated GUI component of DRI has been implemented.
- The WP2 security framework is now capable of consistently supporting basic usage scenarios, starting with authentication in the Master Interface, through to the delegation of security credentials when invoking Atomic Service and all the way to server-side interception of incoming calls and authorizing (or disallowing) access on the basis of current security policies.

The features implemented during the second year of the Project have enabled us to prepare a number of demonstration scenarios which have been presented to the Consortium and beyond (including external members of the VPH network of excellence). These include the following:

- A general presentation of the Atomic Service development, registration and enactment features with the use of dedicated UI components;
- A presentation of the features of the Data Browser interface to LOBCDER resources;
- A security primer, explaining the scope and definition of VPH-Share security policies applicable to individual Atomic Services;
- A selection of simple demonstrations covering interaction with various types of VPH-Share application components.

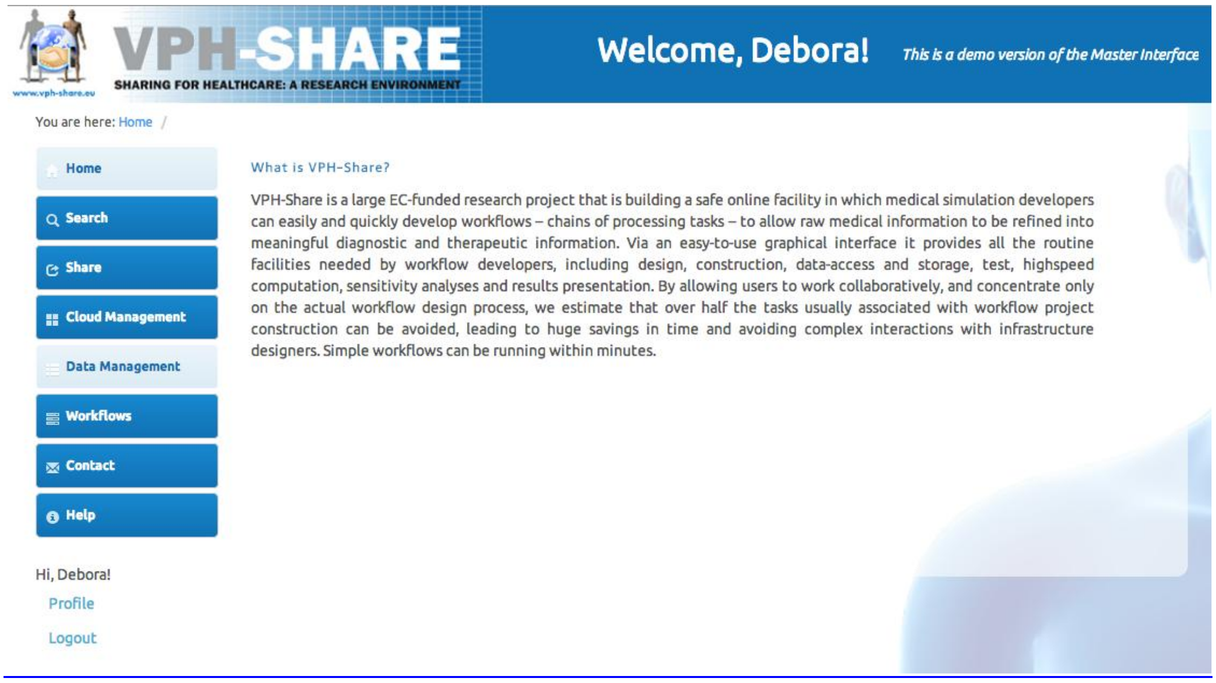
2 END USER ACCESS TO THE CLOUD PLATFORM

The aim of this section is to present the WP2 graphical user interfaces enabling access to the cloud platform in administrator, developer and end-user mode. While these interfaces are being developed in the scope of WP6 (and are not, technically, part of the presented deliverable), we believe that – for completeness' sake – the prototype description should include an overview of cloud platform GUIs as without them access to the WP2 tools would be restricted to developers of application components and other VPH-Share-specific services.

The description will be divided into two parallel streams, one for service developers and one for users interested in invoking a particular Atomic Service. We will begin with the process of creating and registering a new Atomic Service.

2.1 Developer Access – Creating a new Atomic Service

In order to begin work on a new Atomic Service, the developer should first become part of the VPH community (i.e. obtain an account on the BiomedTown portal – which is outside the scope of this deliverable). We will assume that the developer is already a registered user of BiomedTown and has received his/her set of security credentials (login and password). These credentials can be used to obtain authentication with the VPH-Share Master Interface at **masterinterface.vph-share.eu**, a sample screenshot of which is presented in Figure 2.



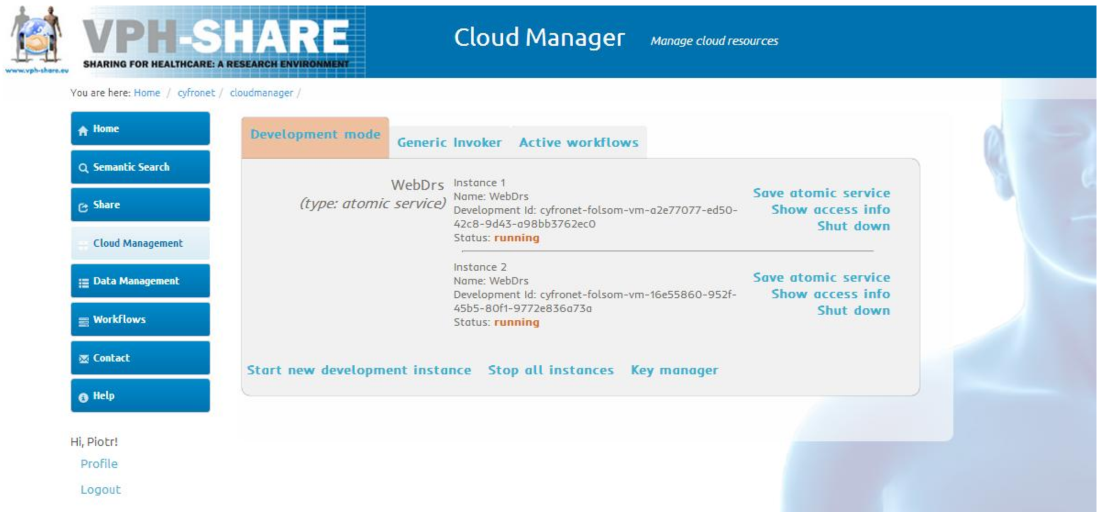
What is VPH-Share?

VPH-Share is a large EC-funded research project that is building a safe online facility in which medical simulation developers can easily and quickly develop workflows – chains of processing tasks – to allow raw medical information to be refined into meaningful diagnostic and therapeutic information. Via an easy-to-use graphical interface it provides all the routine facilities needed by workflow developers, including design, construction, data-access and storage, test, highspeed computation, sensitivity analyses and results presentation. By allowing users to work collaboratively, and concentrate only on the actual workflow design process, we estimate that over half the tasks usually associated with workflow project construction can be avoided, leading to huge savings in time and avoiding complex interactions with infrastructure designers. Simple workflows can be running within minutes.

Figure 2: The VPH-Share master interface.

VPH-Share authentication policies: The VPH-Share Master Interface uses the OpenID authentication standard, which enables users to log in using the credentials, obtained from a OpenID Identity Provider. OpenID is used by a number of well-known organizations, such as Google and Facebook, however in the specific case of the VPH-Share project, the Identity Provider is the BiomedTown portal. By accessing the Master Interface (MI) login feature, the user actually logs into BiomedTown, which is, in all circumstances, the entity responsible for managing VPH-Share user accounts. This is why the MI login request is followed by a query where BiomedTown asks the user to confirm that the requesting application (i.e. the Master Interface) is authorized to access the user's personal information.

Having logged in the user is presented with a variety of options, each represented by an item in the side bar. In order to manage and develop new Atomic Services, the user needs to select the option labelled **Cloud Manager**. This will bring up the Cloud Manager page in the main frame of the MI (see Figure 3).



WebDrs
(type: atomic service)

Instance 1
Name: WebDrs
Development Id: cyfronet-folsom-vm-a2e77077-ed50-42c8-9d43-a98bb3762ec0
Status: running

Save atomic service
Show access info
Shut down

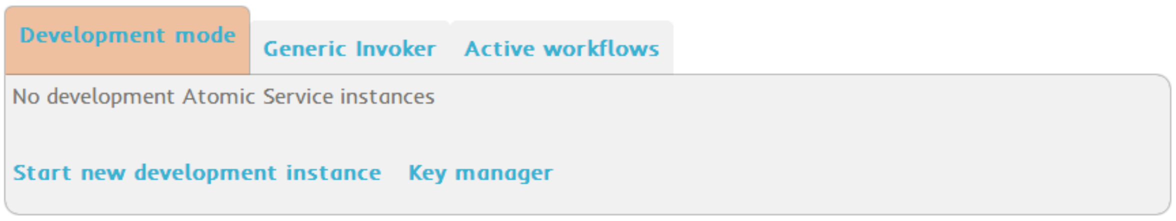
Instance 2
Name: WebDrs
Development Id: cyfronet-folsom-vm-16e55860-952f-45b5-80f1-9772e836a73a
Status: running

Save atomic service
Show access info
Shut down

Start new development instance   Stop all instances   Key manager

Figure 3: The Cloud Manager feature of the Master Interface – sample view

The default view of the Cloud Manager is the Generic Invoker page (which will be discussed in Section 2.2); however VPH-Share application developers will also be able to access a special **development mode**, which enables registered users to manage and modify existing Atomic Services. Clicking this tab will bring up the list of Atomic Service Instances running in development mode (which should initially be empty).



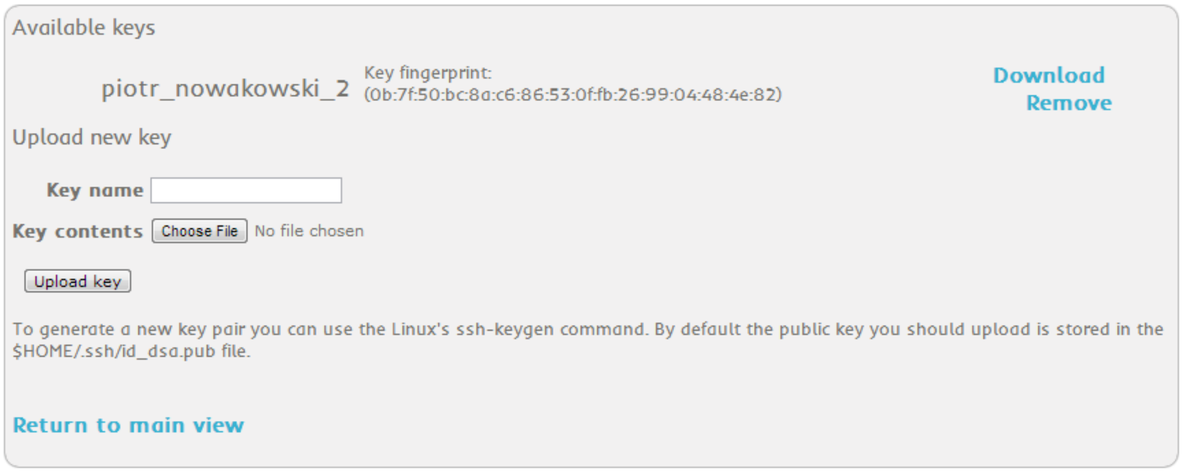
No development Atomic Service instances

Start new development instance   Key manager

Figure 4: Development Mode – initial view

Development mode vs. Invocation mode: Atomic Services (AS) instantiated in development mode follow a special set of policies not applicable to any other mode of AS instantiation. In development mode Atmosphere automatically injects user security keys into running instances to facilitate secure root login for the purposes of modifying the content of Atomic Services. Moreover, Atomic Services not tagged as "published" in the Atmosphere Internal Registry will still be visible in development mode. Finally, development mode provides developers with information regarding the proxy redirection of any external interfaces the Atomic Service may provide (enabling direct access to these interfaces).

In order to be able to interact with services in development mode, the developer must first create a pair of security keys (for instance, by using the Linux **ssh-keygen** command) and upload the public key to Atmosphere with the use of the **Key manager** interface, as shown in Figure 5.


Available keys

piotr_nowakowski_2 — Key fingerprint: (0b:7f:50:bc:8a:c6:86:53:0f:fb:26:99:04:48:4e:82) — Download / Remove

Upload new key

Key name

Key contents [Choose File] No file chosen

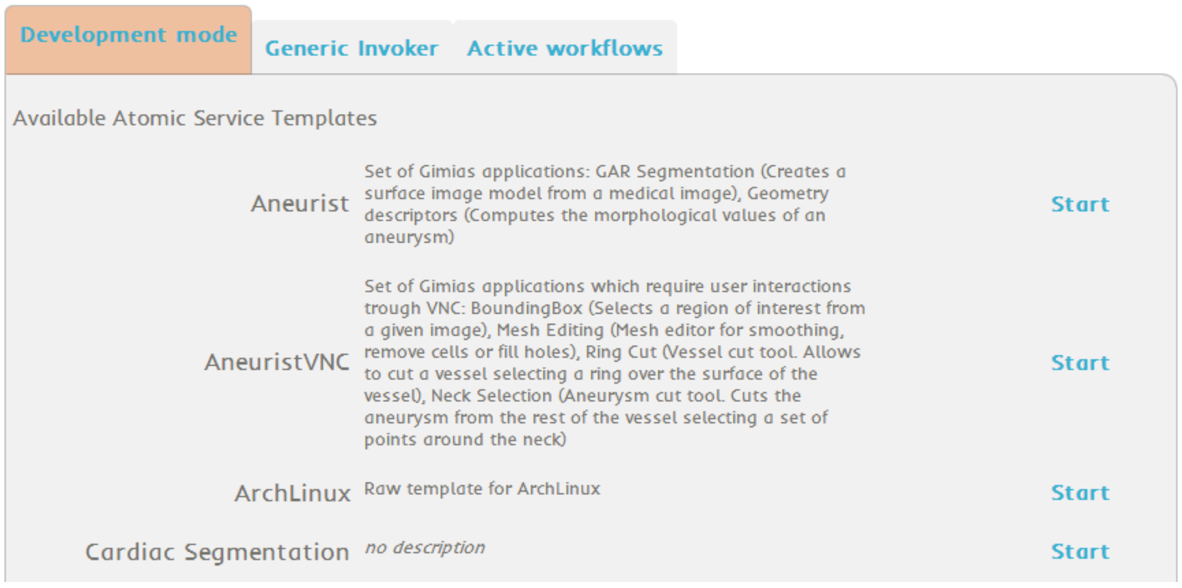
[Upload key]

To generate a new key pair you can use the Linux's ssh-keygen command. By default the public key you should upload is stored in the $HOME/.ssh/id_dsa.pub file.

Return to main view

Figure 5: Atmosphere Key Manager GUI

At least one public key must be stored for each developer to enable secure login to Atomic Service Instances. Once the appropriate public key has been uploaded, the developer may select an Atomic Service for instantiation by clicking the **Start new development instance** button in the development mode view. This will bring up a list of existing Atomic Services and Atomic Service Templates, which can be instantiated (Figure 6).



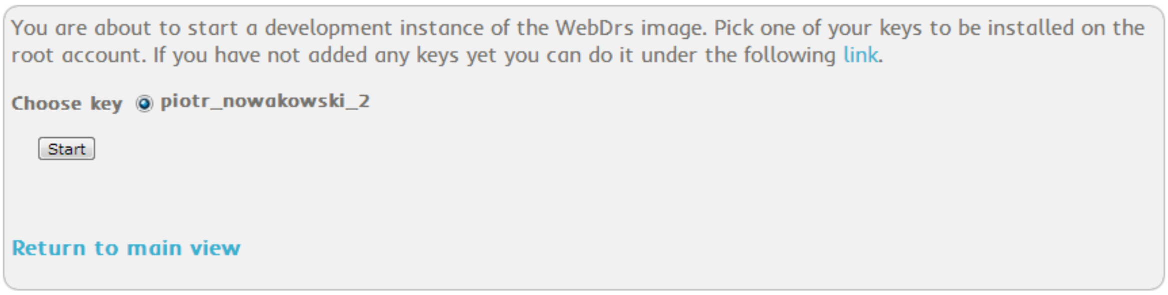
Available Atomic Service Templates

Aneurist	Set of Gimias applications: GAR Segmentation (Creates a surface image model from a medical image), Geometry descriptors (Computes the morphological values of an aneurysm)	Start
AneuristVNC	Set of Gimias applications which require user interactions trough VNC: BoundingBox (Selects a region of interest from a given image), Mesh Editing (Mesh editor for smoothing, remove cells or fill holes), Ring Cut (Vessel cut tool. Allows to cut a vessel selecting a ring over the surface of the vessel), Neck Selection (Aneurysm cut tool. Cuts the aneurysm from the rest of the vessel selecting a set of points around the neck)	Start
ArchLinux	Raw template for ArchLinux	Start
Cardiac Segmentation	*no description*	Start

Figure 6: Selection of Atomic Services for instantiation in development mode.

Atomic Services and Atomic Service Templates: From the point of view of service developers there is no technical difference between services and service templates; the distinction is purely one of convenience: service templates are “raw” OS images which can be used to develop new Atomic Services, while Atomic Services are assumed to include some sort of application payload which can be further extended or modified according to developers’ wishes.

Clicking **Start** next to the name of the selected Atomic Service will display a dialog where the developer is asked to pick one security key to be injected into the new instance, facilitating root login into the instance. Simply pick the preferred key and click the **Start** button again.

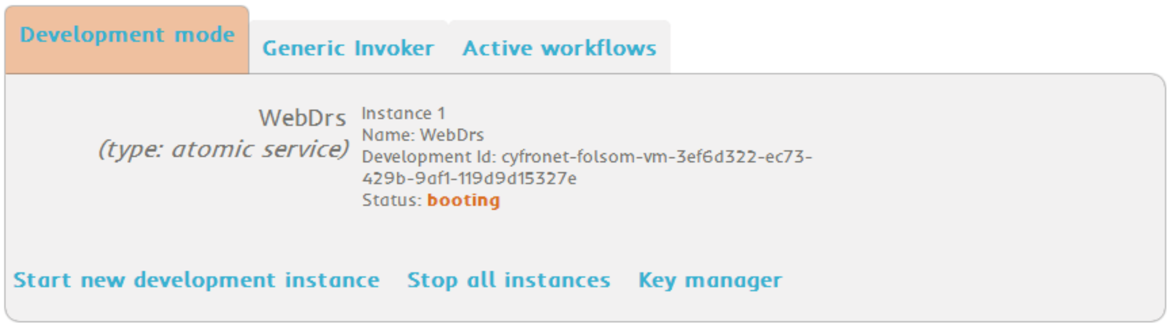


You are about to start a development instance of the WebDrs image. Pick one of your keys to be installed on the root account. If you have not added any keys yet you can do it under the following link.

Choose key ● piotr_nowakowski_2

Figure 7: Selection of public key to be injected into the requested instance.

Once this is done, Atmosphere will begin the process of starting the requested instance. This will be reflected by an appropriate status message in the development mode view (Figure 8).



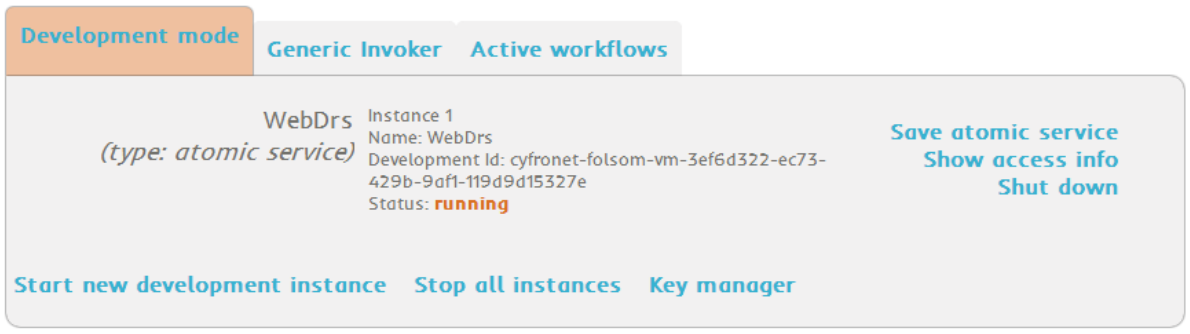
WebDrs
(type: atomic service)

Instance 1
Name: WebDrs
Development Id: cyfronet-folsom-vm-3ef6d322-ec73-429b-9af1-119d9d15327e
Status: **booting**

Figure 8: Booting the selected Atomic Service Instance in development mode.

Booting delay: Depending on their complexity, Atomic Service Instances may take up to several minutes to boot up and become accessible. This is due to the fact that instances are not normally kept in a “running” state (as ready-to-use Virtual Machines) – instead, they reside in a special cloud image repository as binary files. Instantiating an Atomic Service involves copying the selected image file to one of the available cloud hosts, uncompressing it and then booting the operating system within. Once the operating system is up, additional time may be needed to activate any application services residing on the selected VM.

Once the instance is up and running, the status message will be automatically updated, as shown in Figure 9.



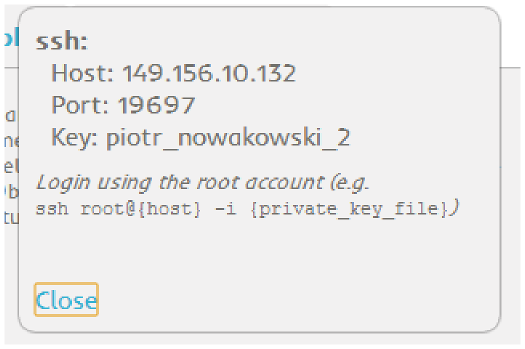
WebDrs
(type: atomic service)

Instance 1
Name: WebDrs
Development Id: cyfronet-folsom-vm-3ef6d322-ec73-429b-9af1-119d9d15327e
Status: running

Figure 9: Atomic service instance up and running in development mode.

Naturally, several different services can be instantiated simultaneously – if you wish to instantiate additional services, simply use the **Start new development instance** again.

Interaction with the running instances may be achieved by communicating with their interfaces. By default, each instance provides SSH access to service developers; however in order to log into the selected instance, we must first learn its address. In a private cloud site this will typically consist of the IP address of the host on which the Reverse Proxy resides and the port used to forward requests to the instance in question. This information can be displayed by clicking the **Show access info** button.



ssh:
Host: 149.156.10.132
Port: 19697
Key: piotr_nowakowski_2

Login using the root account (e.g.
```
ssh root@{host} -i {private_key_file})
```

Figure 10: IP and port used to communicate with the Atomic Service Instance using the SSH protocol.

At this point the developer may use any SSH client to communicate with the instance. The only requirement is that the developer must supply the private key corresponding to the selected public key, which was injected into the instance upon bootup. Root access enables the developer to perform arbitrary changes and modifications to the instance, presumably connected with the installation of the Atomic Service payload.

How persistent are Atomic Service Instances? Editing Atomic Services can be compared to performing work on a text document using word processor software: a file is opened (which corresponds to instantiating an Atomic Service), modified and then saved, either by overwriting the original file or by creating a new file (or, in our case, a new Atomic Service). The metaphor is not entirely accurate since working copies of text files only exist while the word processor is open whereas in our case the Atomic Service Instance persists even if the user closes the "processor" (i.e. the Master Interface). Atomic Service Instances continue to operate and will be accessible the next time you log in. In spite of this fact, it is advisable to perform regular check pointing of your instances (by saving them as Atomic Services) – this will ensure that your work is not lost in case of a cloud site crash or other unforeseen infrastructure problem.

Having concluded implementation work, the developer is free to save the resulting instance as a new Atomic Service. Click the **Save Atomic Service** button to bring up a save dialog (Figure 11).

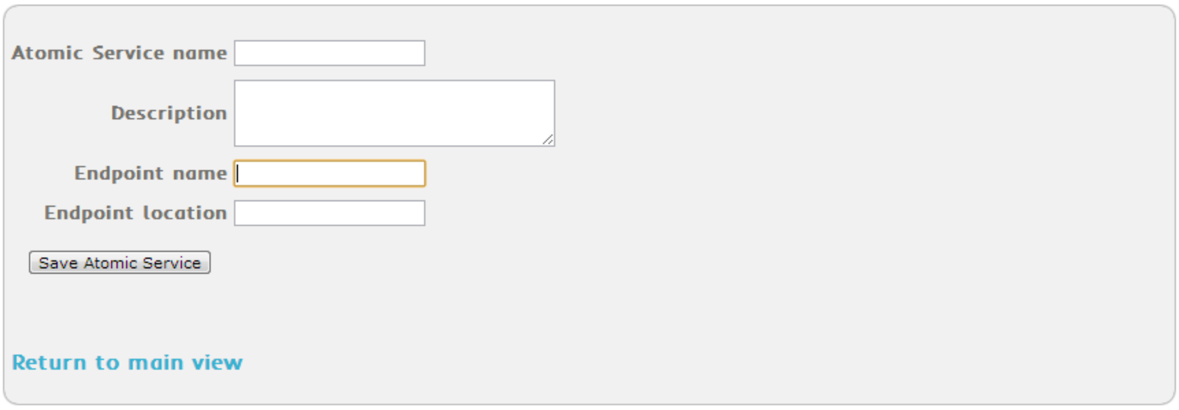


Figure 11: Saving a new Atomic Service – metadata input dialog.

You are required to specify the name of the new AS, as well as (optionally) a description of what it is your service does.

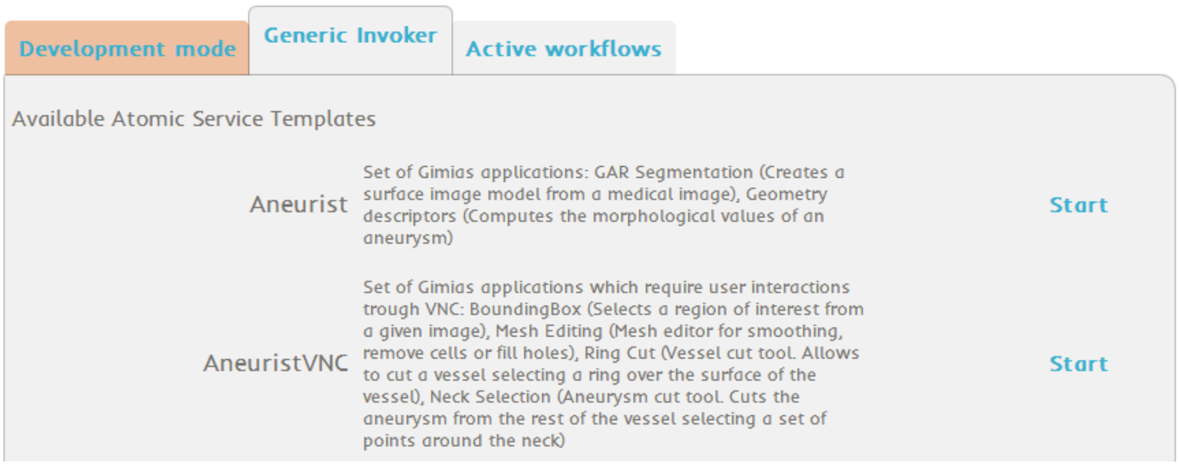
Under construction: The **Endpoint name** and **Endpoint location** fields are used to distinguish services, which provide RESTful (HTTP-based) APIs. If your service is of this type, specify the endpoint path (e.g. **/myservice/hello**) and provide a name for your endpoint (e.g. **hello**). If your AS does not provide RESTful access, you may enter any values in these fields (however, they cannot be left blank in the present prototype). The save dialog will be redesigned and extended with additional options as part of future development of Atmosphere.

Clicking **Save Atomic Service** will cause your new service to be stored (note: this may take several minutes as the instance must be saved to a file in the cloud repository). While the service is being saved, please avoid from further development work on your instance.

Once the service has been saved, it may be instantiated again for a new round of development work. End users may also access the service through the Generic Invoker tab of the Cloud Manager page, which is the subject of the following section.

2.2 End-user access – invoking Atomic Services with the Generic Invoker

The Generic Invoker is a feature provided by Atmosphere to directly communicate with selected Atomic Services. While under most circumstances Atomic Services would be invoked by workflow management tools acting on behalf of end users, there are some application scenarios – such as the ViroLab Drug Ranking System usage – in which it makes more sense to call services directly (for example due to the fact that the entire application is contained within a single service and requires no workflow management). In order to call an Atomic Service, open the Generic Invoker tab and click the **Start Atomic Service** button, which will display a list of accessible Atomic Services (see Figure 12).

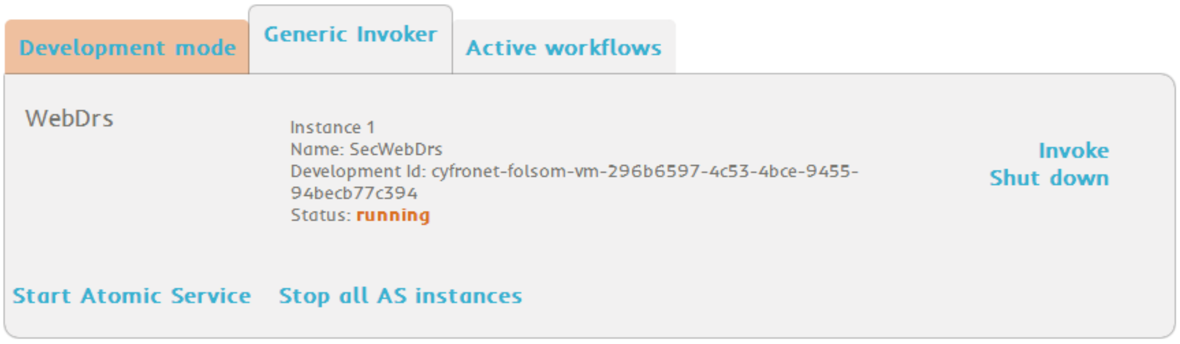


Available Atomic Service Templates

Aneurist	Set of Gimias applications: GAR Segmentation (Creates a surface image model from a medical image), Geometry descriptors (Computes the morphological values of an aneurysm)	Start
AneuristVNC	Set of Gimias applications which require user interactions trough VNC: BoundingBox (Selects a region of interest from a given image), Mesh Editing (Mesh editor for smoothing, remove cells or fill holes), Ring Cut (Vessel cut tool. Allows to cut a vessel selecting a ring over the surface of the vessel), Neck Selection (Aneurysm cut tool. Cuts the aneurysm from the rest of the vessel selecting a set of points around the neck)	Start

Figure 12: Services available for instantiation in the Generic Invoker mode.

Instantiating an Atomic Service through the Generic Invoker is no different than doing so in Development Mode, although in this case the user is not prompted to provide an access key (since the Generic Invoker does not enable users to obtain shell access to the Virtual Machines upon which service instances reside). Once the selected instance has been instantiated the Cloud Manager will display an appropriate status manage along with a set of options (see Figure 13).

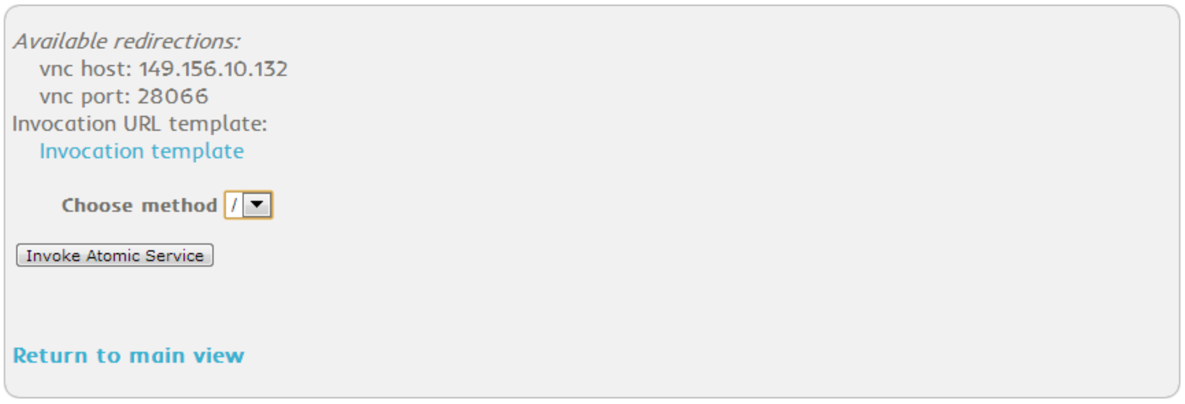


WebDrs

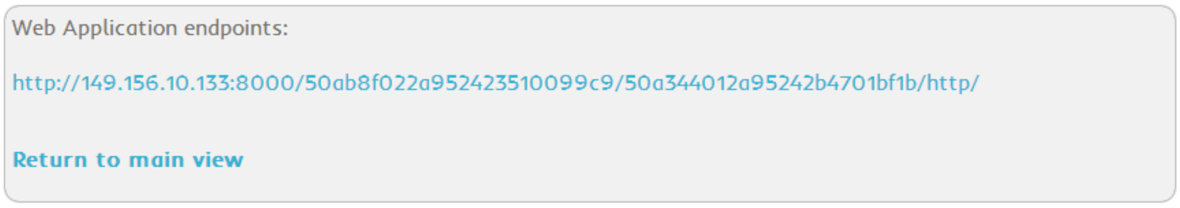
Instance 1
Name: SecWebDrs
Development Id: cyfronet-folsom-vm-296b6597-4c53-4bce-9455-94becb77c394
Status: running

Figure 13: Atomic Service Instance ready for operation in the Generic Invoker mode.

Clicking the **Invoke** button will display information on how to call the specific Atomic Service Instance, depending on the external interfaces it provides. Sample invocation templates for RESTful services and web applications are presented in Figure 14.


Available redirections:
vnc host: 149.156.10.132
vnc port: 28066
Invocation URL template:
Invocation template

Choose method /


Web Application endpoints:

http://149.156.10.133:8000/50ab8f022a952423510099c9/50a344012a95242b4701bf1b/http/

Figure 14: Invocation templates for (1) a RESTful service with VNC access (top); (2) a service which provides a web application endpoint (bottom).

If the service provides a RESTful interface, the Generic Invoker may be used to call its methods directly. Simply select the required method, enter the input data in the field provided and click **Invoke Atomic Service**. For web applications, such as the Drug Ranking Service, Atmosphere provides a URL link to the application root – click this link to access the application. For other types of interfaces, Atmosphere will provide a redirection port and IP which needs to be contacted using an external client, appropriate for the protocol in question (such as a VNC desktop streaming client), to communicate with the given service instance.

Security and authorization: Remember that Atomic Service Instances are secured by the Security Proxy and therefore in order to communicate with them the user must present a valid security token. If you click the URL provided by the Generic Invoker to access a web application service (or use the REST invocation interface), Atmosphere will automatically attach your security token to the request header; if however you decide to use an external client (for instance, if you copy and paste the instance URL to a different browser) then you will be prompted for a set of login credentials. In these circumstances you will want to leave the username field blank and use your current token as password. In order to extract your security token from the Master Interface click the **Profile** option next to your username – the Master Interface will display your token, enabling you to copy it to the clipboard for later use.

Atomic Service Instances that are no longer needed, may be shut down by clicking the **Shut down** option next to their names. It is also possible to close all running instances via the **Stop all AS instances** option.

Remember to clean up after yourself: Running instances consume resources and occupy space on cloud hosts. It is therefore unadvisable to leave instances running when they are no longer needed. If you use workflow management tools to access Atomic Services, cleanup operations are handled automatically, but in Development and Generic Invoker modes you are responsible for cleaning up your own workspace.

3 Programmatic Access to the Cloud Platform (for WPs 3, 4 and 6)

For developers of VPH-Share infrastructure components and external clients that wish to interact directly with the Atmosphere framework WP2 offers programmatic access with a dedicated service-based API. This API is known as the **cloud facade** and is exposed by the Atmosphere core host at **149.156.10.133**.

A detailed specification of the Atmosphere API follows.

3.1 Management of Application Workflow Service Instances

Atmosphere provides an API for management of application workflow service instances. This API enables users to manage Atomic Service Instances bound to specific application workflows.

Atomic Service Instances in the context of application workflows: Each Atomic Service Instance is executed in the context of a specific workflow. Workflows are the principal mechanism by which Atomic Service Instances are grouped and a user may run any number of workflows. A typical example of a workflow is the @neurIST use case scenario where several service instances need to be run as a coherent package. (Note: in the context of Atmosphere a workflow is understood as a set of Atomic Services rather than a graph) The Development Mode and Generic Invoker pages make use of special (hidden) workflows to which all services running in these modes are automatically attached.

The following API commands are used to manage Atmosphere workflows:

`GET /workflow/list` – gets the JSON structure of all user workflows. The structure of the returned JSON includes the following flags:

- **username**
- **workflows** (list of user workflows):
  - (required) **id** – workflow ID
  - (required) **name** – workflow name
  - (required) **type** – workflow type (possible values: **workflow**, **development**, **portal**)

`GET /workflow/{workflowId}` – gets the JSON structure of the workflow identified by `workflowId`

- (required) **workflowId** – workflow identifier (acquired when starting the workflow)

The structure of the returned JSON includes:

- **name** - workflow name
- **type** - workflow type (possible values: **workflow**, **development**, **portal**)
- **atomicServiceInstances** - list of Atomic Service instances created in the scope of this workflow:
  - **id** – Atomic Service instance id
  - **name** – Atomic Service instance name
  - **status** – Atomic Service instance status (**running**, **paused**, **booting**, **stopping** or **stopped**)
  - **message** – optional diagnostic message
  - **atomicServiceId** - id of the Atomic Service used to create this instance
  - **credential** - credentials enabling login to Atomic Service instance (empty in development mode workflow)
  - **redirections** - list of redirections created for this Atomic Service instance:
    - **name** - redirection name
    - **http** - if true, HTTP traffic is redirected
    - **host** - public entry point host for the redirection. Full redirection is host:fromPort

- **fromPort** - public entry point port for the redirection. Full redirection is host:fromPort
- **toPort** - port to which traffic from the public entry point will be forwarded in the Atomic Service instance

Redirections: Redirections are added to Atomic Service Instances to enable external communication with services deployed in a private IP address space. When a redirection is registered for an Atomic Service, Atmosphere will automatically reconfigure the reverse proxy host, which accompanies the cloud site to forward requests to the instance of this service. Thus, instead of communicating directly with the ASI host (which may be impossible), external clients should talk to the proxy host (**149.156.10.132** in the CYFRONET cloud site) on a specific port (which is defined in the redirection configuration).

`POST /workflow/start` – starts a new workflow for a given user

The structure of the JSON body posted must include:

- (required) **name** - workflow name
- (optional) **description** - workflow description
- (optional) **priority** - workflow priority
- (optional) **type** - workflow type (possible values: **workflow**, **development**, **portal**)
- (optional) **asConfigIds** - list of required workflow Atomic Service configurations

If successful, the operation returns status code **200** and the identifier of the new workflow instance. In case of failure, code **400** is returned, along with an explanation of the problem.

Initial configurations: Atomic Service initial configurations are sets of data that is injected upon each Atomic Service Instance upon startup. If the ASI needs to perform some initialization actions (e.g. subscribe to monitoring), the initial configuration can be used to pass the necessary monitoring host endpoint and port number. In most cases, however, initial configurations are not necessary. Atmosphere requires each atomic service to possess an initial configuration but does not restrict its contents. An empty initial configuration is automatically created when a new Atomic Service is saved.

`DELETE /workflow/{workflowId}` - stops a workflow

- (required) **workflowId** - workflow identifier (acquired when starting the workflow)

If successful, the operation returns status code **200**. In case of failure, code **400** is returned, along with an explanation of the problem.

`PUT /workflow/{workflowId}/as/{asConfigId}/{name}` – adds an Atomic Service to a workflow, which also instantiates the Atomic Service.

- (required) **workflowId** – workflow id

- (required) **asConfigId** – Atomic Service configuration id
- (required) **name** – Atomic Service Instance name
- (optional) **key** – identifier of the public key which should be injected when starting the ASI. Only valid in development mode workflow.

If successful, the operation returns status code **200**. In case of failure, code **400** is returned, along with an explanation of the problem.

Developer keys: Security keys enable Atomic Service creators to obtain shell access to the virtual machines on which their Atomic Service Instances reside. This feature is only of concern to developers.
In order to login to your ASI as a root user, you need to create a pair of public/private RSA security keys (for instance using the Linux **ssh-keygen** command) and then upload your public key to Atmosphere using the Master Interface or a dedicated API (see Section 3.4 for more information). When instantiating a new AS in development mode you have the option to inject this key into the new instance, enabling you to log into the ASI as **root** with the use of your private key).

`DELETE /workflow/{workflowId}/as/{asConfigId}` - remove Atomic Service from workflow. This may also shut down the relevant Atomic Service Instance, although the decision on whether to shut down the instance or preserve it for other clients depends on the internal logic of the Atmosphere Management Service.

- (required) **workflowId** – workflow identifier
- (required) **asConfigId** – Atomic Service configuration identifier

If successful, the operation returns status code **200**. In case of failure, code **400** is returned, along with an explanation of the problem.

3.2 Atomic Service Management

This portion of the Cloud Facade interface is dedicated to management of individual Atomic Services (as opposed to Atomic Service Instances, which are managed via the workflow API described in the previous section).

Atomic Services and Atomic Service Instances: Atomic Services are images of Virtual Machines with preinstalled "payload" (i.e. user applications). They exist as files in a dedicated cloud image store (such as the Glance repository in OpenStack-based cloud sites) and need to be instantiated on the available computing resources before they can be accessed. Each Atomic Service may therefore have an arbitrary number of associated Atomic Service Instances.

The following methods are provided:

`GET /as/list` - gets the JSON representation of all registered Atomic Services

The structure of the returned JSON includes an array of:

- **atomicServiceId** - Atomic Service identifier
- **name** - Atomic Service name
- **description** - Atomic Service description
- **vnc** - true if the Atomic Service provides a VNC connection
- **http** - true if the Atomic Service provides a HTTP service
- **shared** - true if the Atomic Service can be shared among workflows
- **scalable** - true if the Atomic Service can be replicated
- **published** - true if the Atomic Service is published (and thus visible to end users)
- **endpoints** – endpoints registered for this Atomic Service:
  - **description** - endpoint description
  - **descriptor** - endpoint descriptor (e.g. WSDL)
  - **invocationPath** - endpoint invocation path
  - **port** - endpoint port
  - **serviceName** - service name
  - **type** - endpoint type (**WS**, **REST**, **WebApp**)

`POST as/create_from/{atomicServiceInstanceId}` - create a new Atomic Service from an existing instance

- **atomicServiceInstanceId** – Source instance for the new Atomic Service

The structure of the JSON request includes:

- (required, unique) **name** - atomic service name
- **description** - Atomic Service description
- **vnc** - true if the Atomic Service provides a VNC connection
- **http** - true if the Atomic Service provides a HTTP service
- **shared** - true if the Atomic Service can be shared among workflows
- **scalable** - true if the Atomic Service can be replicated
- **published** - true if the Atomic Service is published (and thus visible to end users)
- **inProxy** - true if atomic service endpoints are to be automatically registered in the reverse proxy
- **endpoints** – endpoints registered for this Atomic Service:
  - (required) **invocationPath** - endpoint invocation path
  - (required) **port** - endpoint port
  - (required) **serviceName** - service name
  - **type** - endpoint type (**WS**, **REST**, **WebApp**)
  - **description** - endpoint description
  - **descriptor** - endpoint descriptor (e.g. WSDL)

If successful, the operation returns status code **200**. In case of failure, code **400** is returned, along with an explanation of the problem.

`GET /as/{atomicServiceId}/configurations` - gets the available initial configurations for an Atomic Service

The structure of the returned JSON includes:

- **id** - configuration id
- **name** - configuration name

`POST /as/{atomicServiceId}/configurations` - Add a new initial configuration to an Atomic Service

- atomicServiceId – target Atomic Service

The structure of the JSON request includes:

- (required, unique) **name** - configuration name
- (required) **payload** - configuration payload

If successful, the operation returns status code **200**. In case of failure, code **400** is returned, along with an explanation of the problem.

`GET /as/services_set` - Get the available Atomic Services in a Taverna-compliant format. This method is implemented for the purposes of WP6 workflow management integration. Only WS endpoints are currently returned, using the XML template illustrated in Figure 15.

```
<serviceDescriptions
xmlns="http://taverna.sf.net/2009/xml/servicedescription">
  <providers>
    <provider>
      <providerId
xmlns="http://taverna.sf.net/2008/xml/t2flow">http://taverna.sf.net/2010/
service-provider/wsdl</providerId>
      <configBean            xmlns="http://taverna.sf.net/2008/xml/t2flow"
encoding="xstream">

<net.sf.taverna.t2.activities.wsdl.servicedescriptions.WSDLServiceProvide
rConfig xmlns="">
          <uri serialization="custom">
            <java.net.URI>
              <default>
                <string>http://path/to/wsdl</string>
              </default>
            </java.net.URI>
          </uri>

</net.sf.taverna.t2.activities.wsdl.servicedescriptions.WSDLServiceProvid
erConfig>
      </configBean>
    </provider>
    ...
  </providers>

</serviceDescriptions>
```

Figure 15: Sample structure returned by GET /as/services_set

`GET /as/{atomicServiceId}/endpoint/{servicePort}/{invocationPath}` – Get descriptor of a specific Atomic Service endpoint

- (required) **atomicServiceId** - atomic service id
- (required) **servicePort** - endpoint port
- (required) **invocationPath** - encoded invocation path (/ should be changed to %2f)

If successful, the operation returns status code **200** and the descriptor value. In case of failure, code **400** is returned, along with an explanation of the problem.

`GET /{descriptorPath}/get_as_id` - Get Atomic Service identifier for a given descriptor

- (required) **descriptorPath** - descriptor endpoint (see above for details)

If successful, the operation returns status code **200** and the Atomic Service id. In case of failure, code **400** is returned, along with an explanation of the problem.

3.3 Security Policy Management

This section of the Cloud Facade API is dedicated to management of Atomic Service Security Policies.

Security policies: Security policies are documents which define authorization patterns for each Atomic Service. Atomic Services contain inbuilt security proxies which intercept attempts to contact the services' HTTP endpoints and make decisions on whether to permit or disallow access on the basis of the requestor's identity and roles. These decisions are made in accordance with the contents of security policy documents.

The following methods are provided:

`GET /securitypolicy` - lists the names of all stored security policies

If successful, the operation returns status code **200** and the names of the available security policies.

`GET /securitypolicy/{policyName}` – retrieve a security policy

- (required) **policyName** - security policy name

If successful, the operation returns status code **200** and the content of the requested policy. If the policy is not found, code **404** is returned.

`POST /securitypolicy/{policyName}` - upload or update a security policy

- (required) **policyName** - security policy name
- (required) **overwrite** - if true, the existing policy will be updated, otherwise code 409 code will be returned if the policy already exists
- (required) **body** - security policy content

If successful, the operation returns status code **200**. Code **409** is returned if the user attempts to update an existing security policy without setting the **overwrite** query parameter to **true**.

`DELETE /securitypolicy/{policyName}` - removes a security policy and purges its content.

- (required) **policyName** - security policy name

If successful, the operation returns status code **200**. If the policy is not found, code **404** is returned.

3.4 Security Key Management

The Cloud Facade also supports a dedicated API for management of user security keys. As explained in Section 2.1 these keys can be generated and uploaded by registered developers to enable them to log into Atomic Service Instances as root users, in order to perform development or maintenance work.

The following methods are provided:

`GET /key/list` - lists the public keys belonging to the current user

If successful, the operation returns status code **200** and an array of basic data about user keys. The structure of the returned JSON includes the following items:

- **keyName** - key name
- **fingerprint** - key fingerprint
- **id** - key id

`POST /key` – register a new public key for the current user

The structure of the request form includes:

- (required, unique) **keyName** - key name
- (required) **publicKey** - key payload

If successful, the operation returns status code **200** and an identifier of the newly registered. If a key with the given name is already registered, code **409** (conflict) is returned. Malformed keys trigger error code **400** (bad request).

`GET /key/{keyId}` - get content of public key

- (required) **keyId** - key identifier

If successful, the operation returns status code **200** and the payload of the specified public key. If the key is not found, code **404** is returned.

`DELETE /key/{keyId}` - delete public key

- (required) **keyId** – id of key to be deleted

If successful, the operation returns status code **200**. If the key is not found, code **404** is returned.

4 VPH-Share Cloud Platform Modules: Status Update

The goal of this section is to detail the features of each of the technical components, which comprise the WP2 platform, present their integration with external components and outline on-going development activities in each technical task as of Project Month 24.

4.1 Cloud Resource Allocation Management

At the core of the Cloud resource management infrastructure lies the Atmosphere Management Service (AMS). AMS is responsible for optimising utilisation of computational resources and will be used by:

- Application providers, who need to select an appropriate virtual machine template, install and configure their software and save the Virtual Machine as an Atomic Service.
- Scientists, who will specify their requirements in terms of the required functionality (as a list of Atomic Services and their configurations) and will be provided with a pool of resources that is monitored and managed automatically by AMS. This includes interaction with single Atomic Service Instances as well as running workflows, which access a set of Atomic Service Instances.

For a more detailed description of AMS features please refer to Section 4.1.1 of deliverable D2.2 (5).

The internal architecture of AMS and its interactions with other WP2 subsystems are illustrated in Figure 1. The main component of AMS is the Manager, which supervises the process of preparing an optimal deployment plan and provisioning resources. It exposes the AMS functionality to the Cloud Facade and accepts requests from clients. The Manager is also responsible for maintaining a representation of the infrastructure, available Atomic Services and AS Instances in a dedicated registry called the Atmosphere Internal Registry (AIR), described in Section 4.2. When a new request arrives, the Manager queries AIR for available resources and invokes the Optimizer to prepare a deployment plan that will ensure optimal resource allocation. The Manager then enacts the deployment plan by using the Cloud Client to manage resources in the Cloud Execution Environment (CEE), and the Proxy Controller Client to register Atomic Service Instances in the HTTP reverse proxy. A detailed description of this process can be found in Section 4.1.7 of deliverable D2.2 (5).

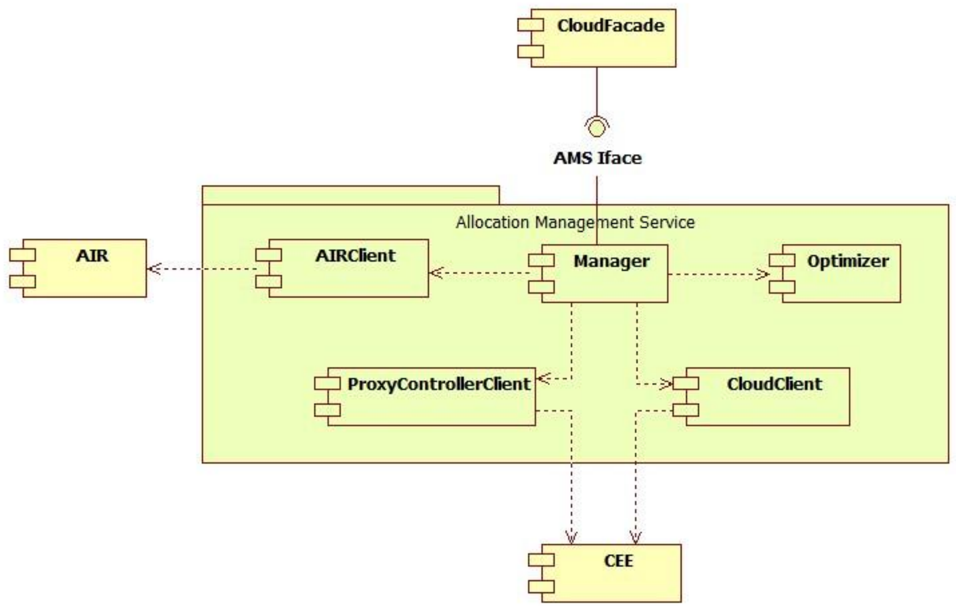


Figure 16: Architecture diagram of Allocation Management Service in the scope of WP2 tools. The light green box indicates internal AMS components while yellow boxes indicate other WP2 subsystems that AMS depends on (i.e. the Atmosphere Internal registry and the Cloud Execution Environment) as well as the Cloud Facade which exposes AMS functionality through REST and Web Service interfaces.

4.1.1 Implementation status

The initial prototype of the framework includes all Allocation Management Service components depicted in Figure 16. The candidate technologies mentioned in Section 4.1.8 of deliverable D2.2 (5) are being used to implement the actual prototype. All AMS components are implemented in Java as OSGI (7) bundles and deployed in a Karaf (8) container. Additionally, AIR and Proxy Controller Clients employ the Camel integration framework (9) that facilitates communication with external services. The Cloud Client is based on the JClouds (10) library. The Optimizer component implements a simple load balancing algorithm in Java that relies on the number of clients using a specific Atomic Service Instance.

The Allocation Management Service prototype provides all the features required by users to start interacting with the WP2 Data and Compute Platform. It implements end-user services (see Section 4.1.1 of deliverable D2.2 (5) for a full list of AMS features), such as:

- Browsing the available templates in AIR
- Instantiating Virtual Machines from selected templates
- Saving Virtual Machines as Atomic Services
- Requesting instances of specified Atomic Services

The prototype supports all the identified use cases, albeit in a restrictedmanner. This can be explained by considering the fourth item in the above list. The AMS prototype is able to start Atomic Service Instances only in a private Cloud infrastructure, and register them in an HTTP proxy to make them publicly accessible. It does not yet collect monitoring data from

Atomic Service Instances and does not take into account required data replication and security constraints – thus, the deployment plans prepared by the Optimiser are far from being optimal and Atomic Service Instances are not subject to automatic scaling.

4.1.2 Allocation Management Service metadata schema

In order to fulfil its responsibilities, AMS maintains a representation of the Cloud infrastructure, Atomic Services and Atomic Service Instances. A simplified view of this internal representation is depicted in Figure 17. At the topmost level lies the **ExecutionEnvironment**, composed of **DataSources** and **ComputeSites**. The former represent data storage services such as Amazon S3 or OpenStack Swift, while the latter model commercial data centres, private cloud deployments, and HPC infrastructures. Each **DataSource** and **ComputeSite** may be associated with a **CostModel**, which defines the expenses for storing and transferring data or running virtual machines of a specific size. **ComputeSites** are composed of **Hosts**, which, in the case of private cloud providers, represent physical machines and their capacity. A **Host** should be treated as a physical computational resource, which is capable of hosting VPH-Share Atomic Services. Since information about physical machines is not available for public Cloud infrastructures each **ComputeSite** of this type will have only one **Host** that represents the resources available for single cloud client (i.e. Atmosphere). **Hosts** run virtual machines with VPH-Share applications (Atomic Service Instances) represented by **Appliances**. Virtual Machines are characterised by **Load** (CPU, memory and disk usage statistics) and **Performance** (duration of a single request or number of requests per second). Each **Appliance** is assigned a certain amount of resources (CPU, memory and disk storage, expressed as Cloud instance size) represented by **ResourceAllocation**. **Appliances** may be instantiated from and saved as **ApplianceTemplates**. **AvailableAppliance** is a base class for **Appliance** and **ApplianceTemplate** classes. It encapsulates attributes and methods common for both of these classes. Examples of such common attributes include **ApplianceState**, which represents the condition of a resource (whether it is running, stopped etc.) and **ApplianceType**, determined on the basis of what type of Application (i.e. arbitrary VPH-Share process) is installed. An **Appliance** requires an **InitialConfiguration** which contains the initialisation context for a given Virtual Machine (for instance a root public SSH key), data required to properly configure and start the hosted **Application** as well as a list of **DataSources** that will be used by the application (required for deployment plan optimisation). Applications consume and produce data represented by **LogicalData** items that are grouped in **DataSets** retrieved from/stored in **DataSources**. Applications and **LogicalData** may have **SecurityConstraints** associated with them that will – for instance – restrict the list of **CloudSites** at which sensitive data may be processed. **DependentWorkflow** models any application that requires the availability of specific Atomic Service Instances.

In Figure 17, Classes that are coloured light green are persisted in the Atmosphere Internal Registry.

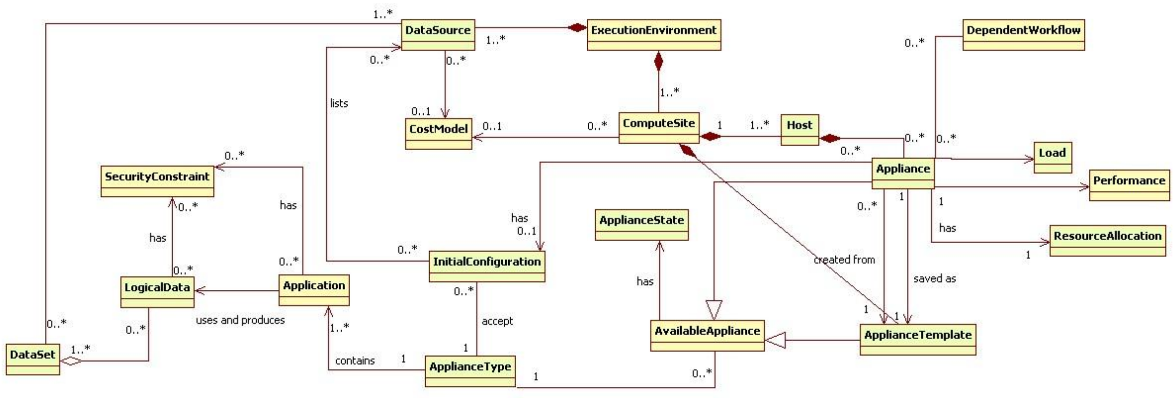


Figure 17: UML class diagram of the internal representation of the Cloud infrastructure, Atomic Services and Atomic Service Instances used by the Allocation Management Service. Classes coloured light green are persisted in the Atmosphere Internal Registry.

4.1.3 Deviations from proposed design

In the original design (see Section 4.1.2 in deliverable D2.2 (5)) it was stated that AIR is a component of the Allocation Management System; however we have since discovered that AIR functionality is also useful for other WP2 tools, such as DRI. Therefore, AIR has been extracted from AMS as a standalone subsystem and will be separately described in the next section of this deliverable. It was also assumed that AMS would expose its interface as a REST or Web Service endpoint. During development it turned out that it is more convenient to provide a single entry point for all WP2 remote endpoints and publish it through the Cloud Facade. Hence, AMS will only provide a local OSGI (7) service for the Cloud Facade.

4.1.4 Future work

During the coming months work on AMS will focus on collecting online monitoring data about the availability and load of Virtual Machines, and storing such data in AIR. A dedicated AMS component, called the Monitoring Controller Client will be developed. It will be responsible for registering and unregistering Atomic Service Instances in the Zabbix monitoring framework deployed in CEE. We will also focus on enhanced optimisation algorithms, taking into account the load of Atomic Service Instances, rather than just their availability.

4.2 Atmosphere Internal Registry

The Atmosphere Internal Registry (AIR) is responsible for delivering the persistence layer and inter-component exchange mechanism for metadata regarding the Project's computing and data cloud. This information is stored, shared and consumed by components of the VPH-Share software; particularly by elements of the Atmosphere cloud computing and data provisioning platform (see Section 4.1.2 for details). Conceptually, AIR should be treated as part of Task 2.1; however due to the fact that it is implemented as a standalone component and its functionality is not limited to supporting the AMS service, we have decided to describe it in a separate top-level subsection of this deliverable.

The main duties involve establishing a common domain model for VPH-Share metadata exchange, providing a suitable persistence mechanism to securely store and manage that metadata and exposing a set of appropriate API (Application Programming Interface) operations for the other entities in VPH-Share to use that functionality for their purpose.

The following sections provide an up to date description of AIR functionality at the end of Year 2 of the Project. They present both the internal structure of the registry and the external programmable interfaces for other software to interact with the data. The tool is available at (11).

4.2.1 Description of the 2nd AIR version

The initial AIR prototype was deployed in late 2011 and the registry has been periodically updated since. Such incremental development enables the developers to fix existing shortcomings quicker and introduce new features more smoothly. They also support incremental updates of any AIR-dependent tools' interfaces.

As of Project Month 24 AIR has gone through a dozen such iterations. Below we present the current architecture of the registry.

4.2.2 Architecture and Domain Model Division

The internal architecture of AIR was considerably extended when compared to the first prototype (see Section 2.2.2 of deliverable D2.3 (12)).

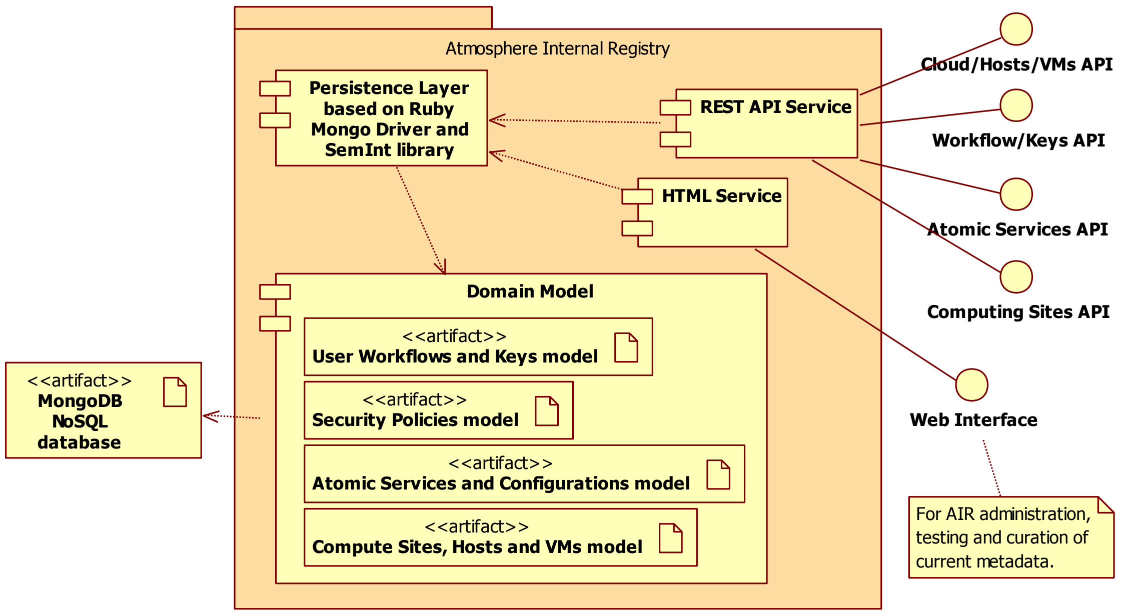

Atmosphere Internal Registry
Persistence Layer based on Ruby Mongo Driver and SemInt library
REST API Service
HTML Service
Domain Model
<<artifact>> User Workflows and Keys model
<<artifact>> Security Policies model
<<artifact>> Atomic Services and Configurations model
<<artifact>> Compute Sites, Hosts and VMs model
<<artifact>> MongoDB NoSQL database
Cloud/Hosts/VMs API
Workflow/Keys API
Atomic Services API
Computing Sites API
Web Interface
For AIR administration, testing and curation of current metadata.

Figure 18: Architecture, APIs and domain models of the second version of Atmosphere Internal Registry. The most notable changes with respect to the first prototype include the addition of several domain submodels, removal of the Logic Dataset model and further API extensions.

As presented in Figure 18, the current version of AIR is capable of storing and publishing the following metadata elements:

1. Public security keys registered by users and the workflows executed in the VPH-Share Portal. This model is user-scoped – it stores information for specific users of Atmosphere.
2. Security policies fetched from AIR and uploaded to Atomic Service Instance in order to establish proper local authorisation mechanisms. This model has a global scope – it stores registered policies for the entire VPH-Share infrastructure.
3. Definitions of the available Atomic Service types and configurations, which may be used when deploying new instances of these services to the cloud. These definitions and initial configuration files are also global in scope.
4. Infrastructure information concerning the available Compute Sites, the hardware resources available for deployment of new Atomic Service Instances and any running Virtual Machines. This model, apart from the topmost sites layer, is subdivided into cloud sites.

Regarding the technologies used inside AIR, the situation is similar to the first prototype. The persistence layer is provided with use of the MongoDB schemaless database, while the domain models, defined using a Semantic Integration (13) methodology to provide semantic abstractions (concepts) over the data inside that database. Web interfaces utilize the Sinatra web toolkit deployed on top of an Apache2 server(14).

4.2.3 Interfaces

AIR provides two types of interfaces – an HTTP API through which other applications and clients can use AIR programmatically, and a basic, internal Web UI for human users (especially system administrators).

The current set of API calls supported by the second AIR version is listed in Table 1. Please note this is simply an overview of what is possible with the current version of AIR. A detailed list of parameters, JSON structures and return codes is available online (11).

Table 1: Complete list of all API calls available in the second version of AIR (February 2012). A detailed manual on the supported API operations is available online as part of the AIR user interface. The deprecated DataSource API is omitted in this table, however for the sake of backward compatibility it is still supported by the registry.

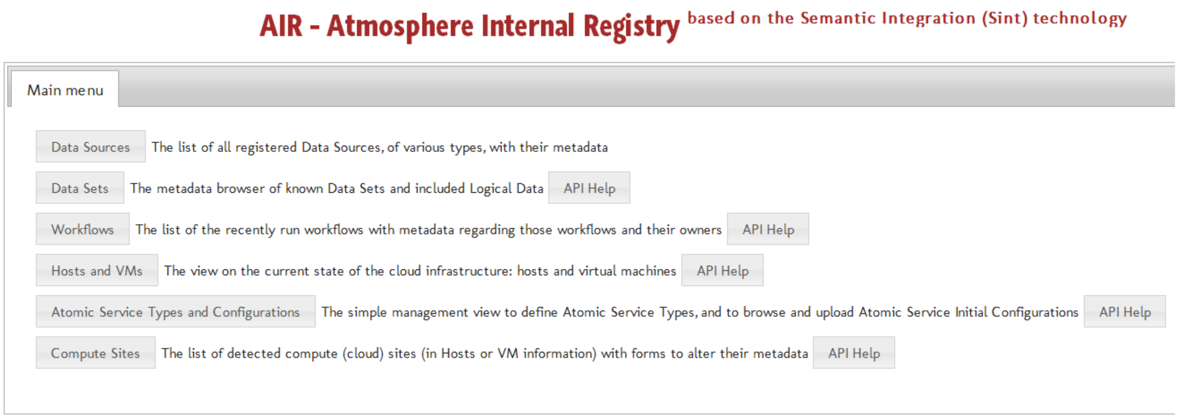
AIR model	API operation call header	Meaning
DataSource management	`---`	**Deprecated**. See following sections of this document for the reason of removing DataSource model from AIR and further information on where the DataSources metadata is available now.
User Workflows	`GET: /running_workflows`	Gets the JSON structure of all running workflows, grouped by user names of their owners. Among others, the returned metadata contains workflow context id, its execution state, the type and priority of the workflow and the list of all Atomic Service Instances (VMs) it uses.

AIR model	API operation call header	Meaning
	`POST: /workflow/start`	Creates a new workflow entry on behalf of a specific VPH-Share user.
	`POST: /workflow/stop`	Deletes a workflow entry.
	`GET: /workflow/(context_id)`	Gets the JSON structure of a running workflow identified with the `context_id`. The returned information set is similar to the `running_workflows` operation result.
	`GET: /workflow/ get_user_workflows/ (vph_username)`	Gets the list (as a JSON structure) of all workflows owned by a user identified with a `vph_username`.
	`PUT: /workflow/associate_vms/ (context_id)/(vms_id)`	Associates a virtual machine with a workflow instance. From now on the infrastructure assumes the workflows depends on the `vms_id` VM.
	`DELETE: /workflow/ remove_vms_association/ (context_id)/(vms_id)`	Removes the VM-workflow association previously added with the PUT `associate_vms` operation.
	`GET: /user_key/get_user_keys/ (vph_username)`	Gets the list (as a JSON structure) of all public keys stored for a given `vph_username` user.
Key management	`GET: /user_key/get_public_key/ (vph_username)/(id)`	Gets the public key payload.
	`DELETE: /user_key/1 (vph_username)/(id)`	Removes the public key from AIR.
	`POST: /user_key/add`	Registers a new public key on behalf of a specific VPH-Share user.
Compute Site management	`GET: /get_compute_sites/`	Returns a JSON structure with full metadata regarding the available Compute Sites. Please note that new Compute Sites are automatically registered when incoming `add_host` or `add_vms` API calls use a new `site_id` identifier. There is no need to explicitly register new Compute Sites.
	`GET: /get_hosts_for_compute_site/ (site_id)`	Returns a JSON structure with full metadata regarding Hosts which belong to the Compute Site specified by `site_id`.
Host and VM management	`POST: /add_host`	*'Upserts'* another host to the topology – adds it if no such host is found in the data or updates altered fields if the host has been registered before.
	`POST: /add_vms`	*'Upserts'* another virtual machine instance (or template of such) to the topology – adds it if no such VM is found in the data or updates altered fields if the VM has been registered before.
	`GET: /get_running_vm_specs`	Generates a JSON file with detailed list of hosts and running virtual machines, along with the specification and used/free resources.
	`DELETE: /remove_vms/(vms_id)`	Removes metadata for the virtual machine

AIR model	API operation call header	Meaning
		identified by `vms_id`.
	`DELETE: /remove_host/(host_id)`	Removes the host metadata from the registry, where the host is identified by `host_id`.
Atomic Service management	`GET: /get_appliance_types/`	Returns a JSON structure with full metadata regarding the registered Atomic Service Types and uploaded Atomic Service Configurations inside AIR.
	`POST: /add_appliance_type`	Creates a new Atomic Service Type according to the input arguments.
	`DELETE: /appliance_type/(apl_type_id)`	Deletes an Atomic Service. To be removable the Atomic Service cannot be associated with any active instances (VMs) and should not have any configurations uploaded.
	`GET: /get_vms_by_appliance_type/(conf_id)`	Gets the JSON metadata of the virtual machines which are currently set to represent a certain Atomic Service.
	`GET: /get_appliance_config/(conf_id)`	Downloads the payload of the configuration file with configuration ID equal to `conf_id`.
	`POST: /upload_appliance_config`	Uploads a configuration string as a new Atomic Service configuration.
	`DELETE: /appliance_config/(conf_id)`	Removes the configuration of the `conf_id` identifier from AIR.
	`GET: /get_appliance_type_for_config/(conf_id)`	Gets a JSON description of a specific Atomic Service. The Atomic Service is chosen according to the passed Initial Configuration ID.
	`GET: /get_appliance_type_for_vm/(vms_id)`	Gets a JSON description of a specific Atomic Service. The Atomic Service is chosen according to the identified instance (VM).
Security Policy management	`GET: /list_security_policies`	Lists metadata about all current security policies (in JSON format).
	`GET: /security_policy`	Downloads the payload of the security policy with a given name.
	`POST: /upload_security_policy`	Uploads a new Atomic Service security policy.
	`DELETE: /security_policy`	Removes the security policy identified by `policy_name`.

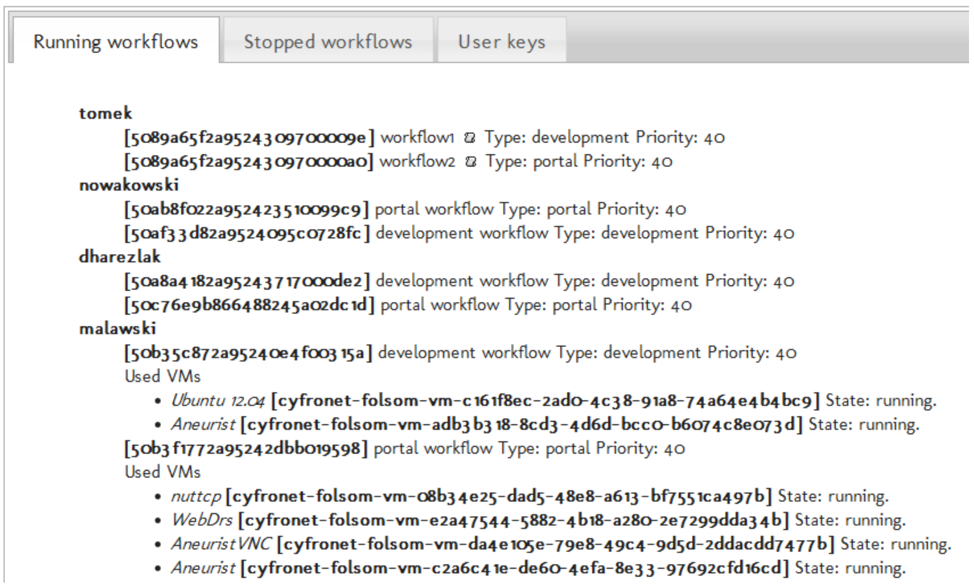
Although the list in Table 1 is already quite long, it is still expected to grow and change during the future implementation cycles of AIR, according to the evolution of VPH-Share user requirements and software. All the listed operations are secured with the basic authentication (HTTP) protocol.

The second type of interface is an administrator Web UI delivered as a set of HTML views over HTTP. Internally the controller tier of AIR uses the same metadata to produce HTML views for the human user. The following figures (Figure 19, Figure 20, Figure 21, Figure 22) show a glimpse of the interface (it is not feasible to include screenshots of all AIR views here).


AIR - Atmosphere Internal Registry based on the Semantic Integration (Sint) technology
Main menu
Data Sources
Data Sets
API Help
Workflows
API Help
Hosts and VMs
API Help
Atomic Service Types and Configurations
API Help
Compute Sites
API Help

The list of all registered Data Sources, of various types, with their metadata
The metadata browser of known Data Sets and included Logical Data
The list of the recently run workflows with metadata regarding those workflows and their owners
The view on the current state of the cloud infrastructure: hosts and virtual machines
The simple management view to define Atomic Service Types, and to browse and upload Atomic Service Initial Configurations
The list of detected compute (cloud) sites (in Hosts or VM information) with forms to alter their metadata

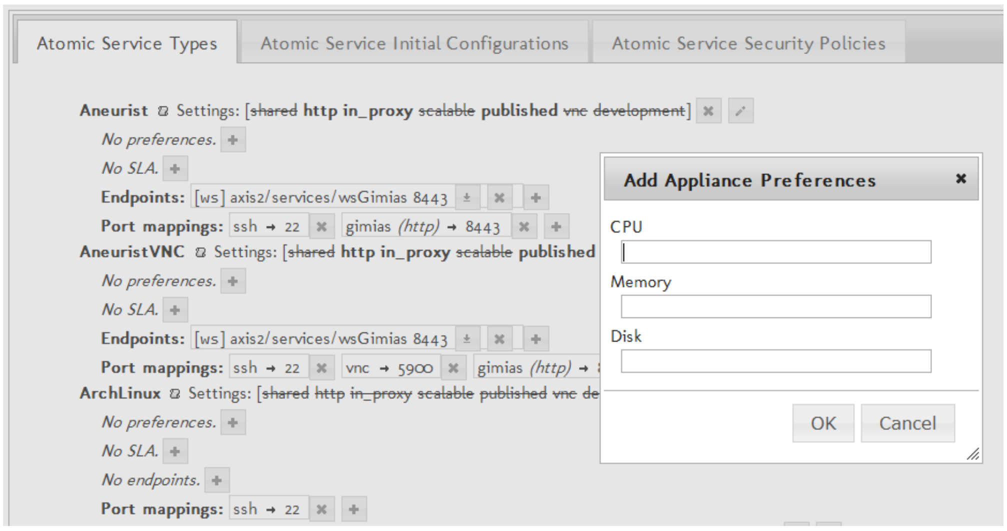
Figure 19: The main menu of the AIR Web user interface. The subsections of the interface correspond to the domain division of the models. Detailed API help is available for each section. Please note the Data Sources and Data Sets are left for compatibility reasons and will be removed as soon as all client tools switch to the LOBCDER metadata solution.



tomek
[**5089a65f2a952430970009e**] workflow1 Type: development Priority: 40
[**5089a65f2a9524309700000a0**] workflow2 Type: portal Priority: 40
nowakowski
[**50ab8f022a95242351009 9c9**] portal workflow Type: portal Priority: 40
[**50af33d82a9524095c0728fc**] development workflow Type: development Priority: 40
dharezlak
[**50a8a4182a952437170000de2**] development workflow Type: development Priority: 40
[**50c76e9b866488245a02dc1d**] portal workflow Type: portal Priority: 40
malawski
[**50b35c872a95240e4f00315a**] development workflow Type: development Priority: 40
Used VMs
- *Ubuntu 12.04* [**cyfronet-folsom-vm-c161f8ec-2ad0-4c38-91a8-74a64e4b4bc9**] State: running.
- *Aneurist* [**cyfronet-folsom-vm-adb3b318-8cd3-4d6d-bcc0-b6074c8e073d**] State: running.

[**50b3f1772a95242dbb019598**] portal workflow Type: portal Priority: 40
Used VMs
- *nuttcp* [**cyfronet-folsom-vm-08b34e25-dad5-48e8-a613-bf7551ca497b**] State: running.
- *WebDrs* [**cyfronet-folsom-vm-e2a47544-5882-4b18-a280-2e7299dda34b**] State: running.
- *AneuristVNC* [**cyfronet-folsom-vm-da4e105e-79e8-49c4-9d5d-2ddacdd7477b**] State: running.
- *Aneurist* [**cyfronet-folsom-vm-c2a6c41e-de60-4efa-8e33-97692cfd16cd**] State: running.

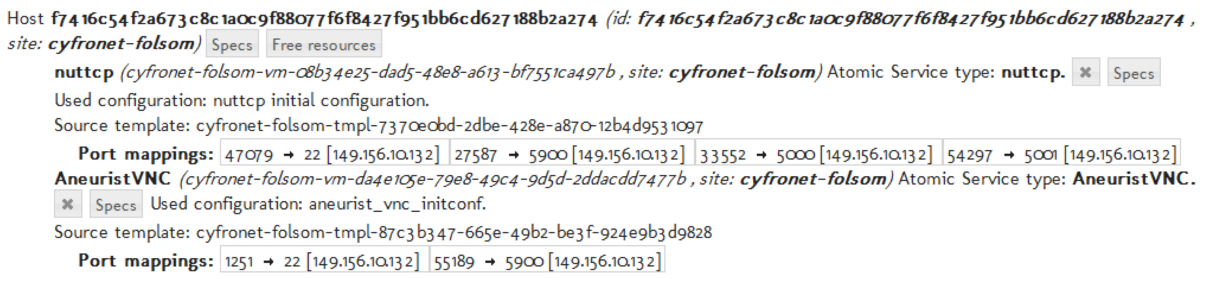
Figure 20: An excerpt from the user workflow management area. The administrator is able to see the workflows executed in the VPH-Share infrastructure at any given moment. In case of problems, the administrator can also view the owner of the workflow in order to contact that person. Stopped (past, archived) workflows are also stored here, as are public user keys.


Atomic Service Types
Atomic Service Initial Configurations
Atomic Service Security Policies

Aneurist Settings: [~~shared~~ **http in_proxy** ~~scalable~~ **published** ~~vnc development~~]
No preferences.
No SLA.
Endpoints: [ws] axis2/services/wsGimias 8443
Port mappings: ssh → 22 gimias *(http)* → 8443
AneuristVNC Settings: [~~shared~~ **http in_proxy** ~~scalable~~ **published**
No preferences.
No SLA.
Endpoints: [ws] axis2/services/wsGimias 8443
Port mappings: ssh → 22 vnc → 5900 gimias *(http)* →
ArchLinux Settings: [~~shared http in_proxy scalable published vnc de~~
No preferences.
No SLA.
No endpoints.
Port mappings: ssh → 22

Add Appliance Preferences
CPU
Memory
Disk
OK Cancel

Figure 21: An example of an administrator dialog for management of registered Atomic Services, their properties, optimization characteristics etc. Some runtime information such as interface endpoints or IP port mappings required by the service is also stored here. The administrator uses this section of AIR to publish new security policies and virtual machine configuration files.


Host **f7416c54f2a673c8c1a0c9f88077f6f8427f951bb6cd627188b2a274** *(id:* ***f7416c54f2a673c8c1a0c9f88077f6f8427f951bb6cd627188b2a274*** *, site:* ***cyfronet-folsom****)* Specs Free resources

nuttcp *(cyfronet-folsom-vm-08b34e25-dad5-48e8-a613-bf7551ca497b , site:* ***cyfronet-folsom****)* Atomic Service type: **nuttcp.** Specs
Used configuration: nuttcp initial configuration.
Source template: cyfronet-folsom-tmpl-7370e0bd-2dbe-428e-a870-12b4d9531097
Port mappings: 47079 → 22 [149.156.10.132] 27587 → 5900 [149.156.10.132] 33552 → 5000 [149.156.10.132] 54297 → 5001 [149.156.10.132]
AneuristVNC *(cyfronet-folsom-vm-da4e105e-79e8-49c4-9d5d-2ddacdd7477b , site:* ***cyfronet-folsom****)* Atomic Service type: **AneuristVNC.** Specs Used configuration: aneurist_vnc_initconf.
Source template: cyfronet-folsom-tmpl-87c3b347-665e-49b2-be3f-924e9b3d9828
Port mappings: 1251 → 22 [149.156.10.132] 55189 → 5900 [149.156.10.132]

Figure 22: Metadata concerning a single host which belongs to the VPH-Share infrastructure. For a given machine and the virtual machines running there Atmosphere requires a considerable amount of information in order to support optimized, effective cloud provisioning.

The AIR web UI is not intended for the actual end users in the VPH-Share community, i.e., the scientists. Rather, the aim is to provide system administrators with the ability to register new entities inside AIR, check if the stored metadata is valid and set up important parameters of the Atmosphere cloud provisioning mechanisms. Administrators may also monitor the current state of running user workflows and the registered infrastructure elements: Compute Sites, Hosts and Virtual Machines.

4.2.4 Relocation of Metadata Related to Cloud Data Source

One important change with respect to the first prototype of AIR (see Section 2.2 of deliverable (12)) is the removal of the metadata related to Data Sources. This was performed in cooperation with the Project Implementation Group, the LOBCDER developers and the persons responsible for semantic metadata management in WP4.

All the tools and users who require access to metadata regarding VPH-Share Data Sources, will be able to retrieve such metadata from either the LOBCDER metadata service (information related to the runtime properties of the files: checksums, sizes, full paths etc.) or

WP4 Semantic Services (metadata regarding semantics, or "meaning" of the stored files). This change has occurred for two reasons: performance requirements related to accessing file data and the need for better separation of responsibility between WP2 and WP4. Please consult Section 4.5 for a more thorough description of how LOBCDER handles metadata.

4.3 Cloud Execution Environment

4.3.1 Structure of the existing Cloud platform

The Cloud Execution Environment (CEE) constitutes a specialized platform, which is interfaced by:

- The Atmosphere component described in Section 4.1, to enable low-level operations on Virtual Machines, such as managing the VM lifecycle or preparing snapshots that can be used as templates.
- The WP3 tools, as a Private Cloud storage backend used by such tools to store and manipulate data.

CEE is composed of several components, shown in Figure 23, some of which are off-the-shelf while others are custom-developed.

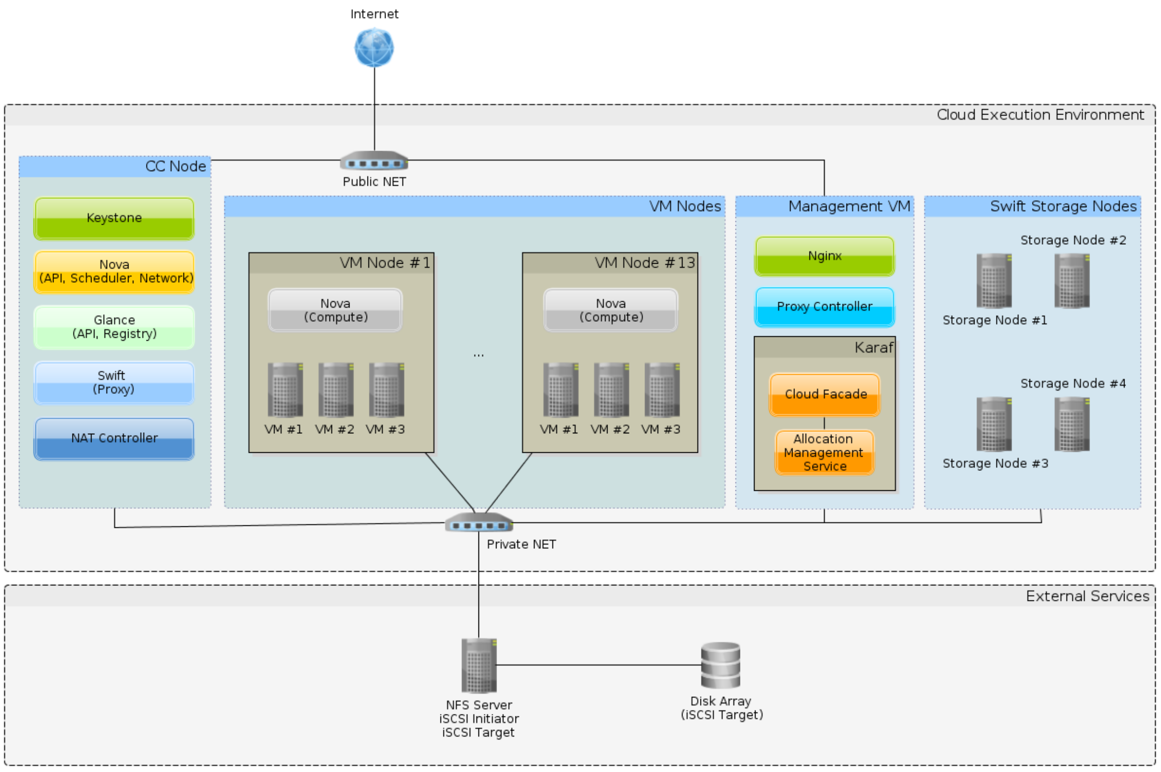


Figure 23: Architecture of the Cloud Execution Environment, including OpenStack nodes (CC, VM, Swift), additional management VM (for the Nginx-based reverse proxy) as well as external components.

The main components are elements of the OpenStack (15) middleware suite (described in more detail later on in this document) responsible for providing computational and storage features for the private cloud installation.

During the last year we have doubled the number of nodes used by the computational part of the cloud (from 7 to 14 nodes). All deployed nodes are homogenous with regard to hardware (HP ProLiant BL2x220c G5, 2 x Intel Xeon L5420, 16GB RAM, 120 GB internal HDD). One of them acts as the Cloud Controller (CC) and the rest are used to run VMs. All nodes have access to approximately 3 TB of external shared storage space (NFS on iSCSI volume) backed by a disk array with fast (15000 RPM) SAS hard drives. This shared space is used to store VM templates and images of running Atomic Service Instances. We have also deployed infrastructure-monitoring tools, which allow us to judge that at the current stage of the project the presented resources are more than sufficient. However we are ready to extend our resource pool if demand increases. All mentioned nodes are connected to an Ethernet switch with support for 802.1Q VLANs using 1 Gbps ports (switch uplink port is 10 Gbps). As such, the physical network layout is consistent with the description and diagram presented in Section 4.2.3 of deliverable D2.2 (5). According to the description contained in that document this layout enables us to choose any network mode available for OpenStack. As a result we have decided to use a VLAN-based network setup, which provides 2 VLANs – one for physical nodes and one for VPH VMs, which provides L2 level separation. However, if required (e.g. for security reasons), this deployment allows us to provide additional VLANs for multiple VM groups that should be separated. Finally, in addition to internal LANs (using private IP addresses) the CC Node is connected to the Internet (WAN) and has a public IP address. Thus, the CC node can act as a NAT-enabled router for the remaining physical nodes and VMs, including both SNAT (for all IP outbound traffic) and DNAT (for some inbound traffic on predefined TCP/UDP ports) between cloud VMs in a private LAN and the Internet.

The DNAT (Destination Network Address Translation) mechanism is remotely configurable via the **NAT Controller** – a lightweight service written in Ruby, exposing a RESTful API, and is used by **AMS** to provide TCP/UDP port redirection for any network service (other than Atomic Service Instance HTTP(s) endpoints) such as SSH or VNC. ASI, on the other hand, required a more intelligent solution enabling redirection of traffic based on deeper packet inspection (in the application layer) for which we have proposed a solution based on the **Nginx** (16) server acting as a reverse proxy. It is configured using the **Proxy Controller**, a lightweight service written in Ruby that exposes a RESTful API (with the help of the Sinatra library). The Proxy Controller is used by the **Proxy Controller Client** (part of AMS) to register and unregister ASIs. All components of the proxy are deployed on a dedicated VM (marked as "management VM"), directly connected to both the Internet (public IP) and all LANs (private IPs). Both mechanisms (NAT and the proxy) facilitate conservation of the public IP address space. Such mechanisms may also prove critical for (pre-IPv6) future development of scientific clouds especially given the exhaustion of IANA's IPv4 pool and near-exhaustion of RIRs pools, such as RIPE NCC responsible for allocations in Europe (17), which, as of 14 Sep 2012, began allocation of IPv4 space from the last /8 block (https://www.ripe.net/internet-coordination/news/announcements/ripe-ncc-begins-to-allocate-ipv4-address-space-from-the-last-8). In turn, RIPE is now distributing IPv4 addresses in accordance with the section 5.6 of "IPv4 Address Allocation and Assignment Policies for the

RIPE NCC Service Region", which prevents any non-LIR provider from getting new independent public IPv4 resources, as well as imposes a hard limit even on LIRs by allowing only a single allocation of /22 subnet (1024 addresses) provided the LIR meets special conditions. Unfortunately, direct use of an IPv6-only network for our cloud installation is not possible as IPv6 support in the public Internet is still lagging; however OpenStack already supports this protocol and it remains a possible option for the future. Note that the use of dual-stack (private IPv4 network and public IPv6 addresses) solutions would preclude direct access from IPv4 clients (NAT/Proxy mechanisms still would be required).

The storage component of our private cloud is currently deployed on 5 physical nodes. Its frontend is deployed on the CC node, while storage nodes are provided as 4 dedicated physical nodes – typical 1RU form-factor servers with single Intel Xeon CPU (dual core, 2.80GHz) and 2x2TB 7200 RPM SATA HDD per node. As suggested by OpenStack developers, the XFS file system is used on those volumes. As data is stored by our installation of OpenStack in a triple-redundant manner, ca. 5.3 TB of space is available; however, just like the computational part of our cloud, the storage space can be flexibly increased if required.

4.3.2 Cloud middleware stack

As foreseen during the design phase, we have deployed the OpenStack middleware suite in our private Cloud infrastructure. The currently installed release is Folsom, which at the time of preparation of this document, is the most current stable version of OpenStack.

More specifically, we have deployed four of the "core" services of this release:

- Identity (Keystone) – responsible for centralized management user credentials for other services that are used for authentication and authorization, as well as acting as a service catalogue (storing other service endpoints). It provides a REST-based API (current version 2.0) however this API is only used internally by the stack and not exposed to the rest of the platform.
- Computing (Nova) – directly responsible for managing the lifecycle of VMs and ASIs. It provides an external REST-based OpenStack Compute API (a.k.a. Nova API, currently in version 2) which is used by the **Cloud Client** component of Atmosphere (by way of the JClouds (10) library).
- Image Service (Glance) – this enables storage of VM templates used to instantiate Atomic Service Instances. It also provides a REST-based API (called the OpenStack Image Service API; a.k.a. Glance API, currently in version 1.0/1.1) however this API is only used internally by the Task 2.1 middleware stack and not exposed to the rest of the platform as it is not required by external clients.
- Object Storage (Swift) – for the storage part of the Private Cloud used by WP3 tools and WP5 services. Like other parts it also provides a REST-based "OpenStack Object Storage" API (a.k.a. Swift API, currently in version 1.0), which is exposed for external clients and can be interfaced by the LOBCDER storage federation described in Section 4.5.

We have performed standard manual software installation using official package repositories as described in the project documentation. In accordance with this description we have used Ubuntu (12.04 LTS) on all nodes.

4.3.3 Infrastructure provisioning status

While migrating the CYFRONET cloud site to the Folsom release we have also taken the opportunity to purge old and unneeded AS images, while preserving the actual application services belonging to VPH-Share workflows. Table 2 summarises the application services currently provided on the CYFRONET site.

Table 2: Virtual Machines provisioned to VPH-Share developers as of Project Month 24.

VM no.	Allocated resources	Recipient	Purpose
1	1 VCPU, 2 GB RAM, 5 GB HDD	[UVA]	VM for development and testing of the ViroLab workflow
2	1 VCPU, 2 GB RAM, 5 GB HDD	Martin Steghöfer [UPF]	VM for the Taverna Server
3	1 VCPU, 2 GB RAM, 5 GB HDD	Xavier Planes [UPF]	Non-interactive part of the @neurIST workflow
4	1 VCPU, 2 GB RAM, 5 GB HDD	Xavier Planes [UPF]	Interactive part of the @neurIST workflow
5	1 VCPU, 2 GB RAM, 10 GB HDD	Eric Kerfoot	VM for Atomic Services used as part of the euHeart workflow
6	1 VCPU, 2 GB RAM, 25 GB HDD	Xavier Planes [UPF]	VM for the ANSYS simulation package
7	1 VCPU, 2 GB RAM, 10 GB HDD	[ATOS]	euHeart workflow VM clone to enable development and testing of the security framework
8	1 VCPU, 2 GB RAM, 6 GB HDD	[P-Medicine]	The OncoSimulator application
9	1 VCPU, 2 GB	[UvA]	DataFluo application components

VM no.	Allocated resources	Recipient	Purpose
	RAM, 6 GB HDD		(Master Node)
10	1 VCPU, 2 GB RAM, 6 GB HDD	[UvA]	DataFluo application components (Worker Node)

In addition to these resources we have also allocated space on storage part of the cloud for:

- Testing the WP3 tools during their development process.
- Storage of data required to test the @neurIST workflow.

4.3.4 Future work

The process of migration from Diablo to Folsom was smooth; however, similarly to initial installation, we had to overcome several minor glitches and some outstanding issues may still require further work.

The issues already addressed are as follows:

- There is still no built-in mechanism for redirecting TCP/UDP ports. As a result, we are using the previously described NAT Controller, which has performed its function well and would most likely be a permanent solution in the VPH-Share platform.
- Contextualization mechanisms do not automatically remove old keys. We're verifying that the key is removed before creating a template however the process needs to be automated so that no previously injected credentials slip into the template.
- Template saving progress is not reported – this is not a real issue as templates are properly created and information about their readiness is communicated to the client – however the lack of a progress indicator mars the user interface.

The problem with creating templates using JClouds (reported in D2.3) has been solved and this feature is now fully supported.

We have also deployed a dedicated Zabbix-based monitoring infrastructure for the VMs and ASIs. We had initially planned to deploy a solution based on Nagios, however we have since changed those plans mainly due to the fact that the current version of Zabbix offers a JSON-RPC based API that allows smooth integration with the AMS (including the ability to register/unregister ASIs and extract monitoring data). Additionally, our research indicates that Zabbix is also a reliable platform, hence its deployment in place of Nagios would not degrade our infrastructure.

4.4 High-Performance Execution Environment

The Application Hosting Environment (AHE) 3.0 is a lightweight middleware suite which virtualises grid applications and exposes their features as RESTful web services. Grid middleware tools are complicated applications with a steep learning curve. This is often a troublesome issue for scientific end users. AHE attempts to hide the complexity of the underlying HPC resources by providing a lightweight layer between the grid middleware and the user. An expert user is required to setup AHE with information on how and where to execute an application. Once this is completed, the scientific end user can execute the application through a set of simple RESTful web service commands.

The Audited Credential Delegation application (ACD) provides a secure solution that audits, authenticates and authorises user commands. It also provides virtual organisation management as well as credential delegation features. ACD is implemented as a RESTful web service and, in conjunction with AHE, provides a simple way to handle grid credentials, and manage data and computational jobs.

AHE and ACD can be accessed using RESTful web service. AHE accesses the underlying grid middleware using a number of Java middleware client libraries, including UCC (unicore), JGlobus (Globus) and QcG to launch applications that have already been installed in the relevant grid infrastructure. AHE also supports file transfers using WebDAV and GSI-FTP protocols. AHE can use ACD to authenticate and authorise the user and generate proxy certificates on the user's behalf. AHE itself is able to check and auto-generate MyProxy certificates if required.

4.4.1 Current Status of the Prototype

During the second year of the project (following the issuance of Deliverable 2.3) a newly designed version of AHE 3.0 has been implemented in Java. AHE 3.0 can be deployed in two different versions: as a standalone Java application using an embedded Jetty Server, or as a servlet that can be deployed on a servlet-compliant server such as Apache Tomcat. AHE 3.0 can be categorized into two major components: the AHE core server which consists of a public RESTful API, internal data registries to store descriptions of users, applications, application instances, workflow management engines, and AHE module servers. The other major component is a set of AHE module servers. These modules generally provide a set of features such as job submission for specific middleware, or data transfer. Each module server is exposed using an internal RESTful API. In summary, AHE 3.0 consists of the AHE core server with a public API and many AHE module servers, which communicate with the AHE core servers to perform specific functions. This architect allows AHE to be more scalable and resilient to module failures. It also overcomes the JAR dependency conflicts seen in the previous version of AHE.

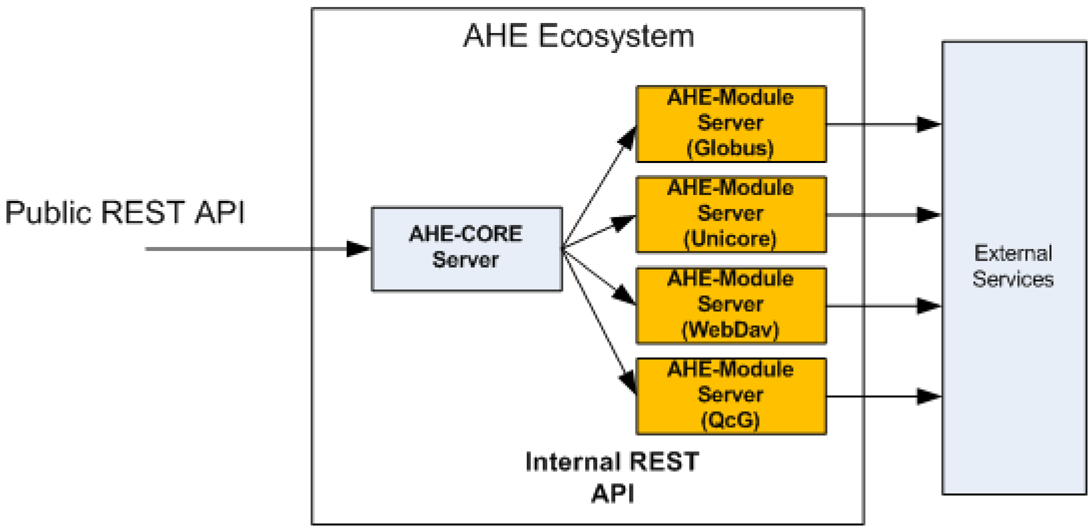


Figure 24 AHE 3.0 web service oriented architecture

The AHE core server consists of the core data structures and libraries that implement AHE functionality – these include user, security, application, resource, and workflow management. AHE is currently able to create and edit user information and security credentials. This allows AHE to authenticate or map the user to ACD for authentication and authorisation. It is also able to set up a set of credentials to connect to the underlying grid middleware. An application and resource registry has been implemented in AHE using the Hibernate ORM (Object-Relational Mapping) framework, which allows AHE to set up virtual applications on corresponding resources. These applications translate into VPH-Share Atomic Services and can be managed by Atmosphere. The main features enabling AHE to create and initiate virtual applications have also been implemented. This allows the user to find the application they wish to launch and submit it, along with the required input data, to the resource they desire. When the user initiates a virtual application or workflow, the persistent workflow engine starts a new workflow and proceeds through successive workflow stages. The AHE workflow engine enables simple job submission with pre- and post-processing stages to be implemented, it also allows more complicated workflows for virtual applications (such as error recovery) to be created.

As of Project Month 24, additional features has been added to AHE 3.0 include support for VPH-Share authentication as well as implementation of an AHE Web client using the Google Web Toolkit. A number of new features have also been added to AHE 3.0, including session token management, WebDAV support and file upload.

AHE currently includes a number of middleware connectors: QcG (using the QcG Java SDK), Unicore (using the UCC Java library) and Globus (using the JGlobus library). AHE is able to generate a submission object compliant with those connectors from the application execution details and configurations provided by the user. AHE is also able to handle GSIFTP and WebDAV file transfer protocols. The WebDAV transfer protocol is implemented to integrate AHE with the VPH-Share LOBCDER storage mechanism. Each of these middleware and data transfer handlers are implemented in a dedicated AHE module server.

The RESTful web API for AHE has been updated and streamlined since Project Month 12. The interface of this service is presented in Table 3.

Table 3: AHE service API.

HTTP Method	URI Resource	Comment
Post	/profile	Add User
Post	/profile/{profile_id}	Edit user
Get	/profile	List all user
Delete	/profile	Remove User
Post	/cred	Add credential
Post	/cred/{cred_id}	Edit credential
Get	/cred	List credentials
Delete	/cred/{cred_id}	Remove credential
Post	/profile/{profile_id}/cred/{cred_id}	Add credential to user
Delete	/profile/{profile_id}/cred/{cred_id}	Remove credential from user
Get	/appinst	Get status of application instance
Post	/appinst/{appinst_id}/transfer	Set data staging for both stage in and out
Get	/appinst/{appinst_id}/transfer	Get data staging information
Post	/appinst	Prepare virtual application
Post	/appinst/{appinst_id}/runtime	Start application / workflow
Delete	/appinst/{appinst_id}	Terminate
Get	/appinst	List all jobs
Post	/appinst/{appinst_id}/property	Set application instance property
Get	/appinst/{appinst_id}/property	Get application instance property
Get	/appinst/{appinst_id}/property	List all application instance properties
Post	/resource	Create a resource in the resource registry
Post	/resource/{resource_id}	Edit a resource in the registry
Delete	/resource/{resource_id}	Delete a resource in the registry
Get	/resource	List all resources in the registry
Post	/app	Create an application in the application registry
Post	/app/{app_id}	Edit an application in the application registry

HTTP Method	URI Resource	Comment
Get	/app	List all applications in the registry
Delete	/app/{app_id}	Delete an application in the application registry
Post	/session	Generate temporary session token

The Audited Credential Delegation (ACD) application has also been implemented as a RESTful web service. Currently, ACD can be deployed as a standalone application using an embedded Jetty server or as a servlet container deployable on servlet compliant servers such as Apache Tomcat.

ACD is currently able to audit commands issued by the user. It is also able to set up virtual organisations (VO) including member management and group certificate setup, allowing ACD to generate proxy certificates for each member of the VO. Finally, ACD is able to authenticate and authorise users based on their roles (administrator or researcher). It supports Shibboleth authentication as well as local username/password database authentication.

A summary of ACD RESTful web service commands can be found in Table 4.

Table 4: ACD RESTFUL API.

Command	HTTP Method	Resource	Comment
`createlocalACDAccount`	Post	User	Create new ACD account
`createNewVO`	Post	CMD	Create new Virtual Organisation
`generateProxies`	Post	CMD	Generate proxy
`registerUser`	Post	CMD	Register user in VO
`getACDAudit`	Get	CMD	Get ACD audit
`getCertificateDetail`	Get	VO	Get certificate details
`getRoleAssignment`	Get	ACD Credential	Get role assignment
`getuserACDAccounts`	Get	User	Get user ACD account
`getUserVOs`	Get	ACD Credential	Get user VO

Command	HTTP Method	Resource	Comment
`viewAllRoles`	Get	CMD	View all roles
`viewCurrentVOs`	Get	CMD	View current VO
`assignP12CertToVO`	Post	VO	Assign P12 certificate to VO
`assignUserToRole`	Post	ACD Credential	Assign user to role
`assignUserToVO`	Post	VO	Assign user to VO
`changelocalACDPassword`	Post	Login	Change user local ACD password
`updateP12VOCert`	Post	VO	Update P12 certificate of VO
`updateRegisteredUserDetails`	Post	User	Update user details
`resetACDPassword`	Post	Login	Rest user ACD password
`RevokeUserRole`	Delete	ACD Credential	Revoke user role
`deactivateACDUser`	Delete	Login	Deactivate ACD user
`Deactiveteall`	Delete	User	Deactivate all
`removeUserVOs`	Post	VO	Remove user from VO

4.4.2 Deviations from Proposed Design

Several changes in AHE have occurred since the publication of Deliverable 2.2. First, JavaGAT will not be used in the Connector module, as AHE does not currently require the features provided by JavaGAT or general-purpose grid SDK libraries. Instead, it is simpler to use UCC for Unicore, JGlobus for Globus and additional middleware Java client libraries, as required.

Since Project Month 24, AHE 3.0 had to be rewritten using a web service-oriented architecture to improve performance and reliability as well as overcome the Java JAR file conflict issue. Several components have been added as extensions to the presented design, particularly in the context of user credential management. This is due to the fact that AHE may have to be deployed without ACD. Proxy Delegation support has also been added to AHE, enabling AHE to automatically obtain proxy certificates on behalf of the user.

In ACD, one major change is that ACD will no longer intercept commands from AHE and authorise them. Instead, all commands will be authorised by AHE, while ACD is used to authenticate the user and – if necessary – generate proxy certificates.

4.4.3 Future work

Since Project Month 24 AHE has been re-implemented with the introduction of a streamlined RESTful API and numerous bug fixes and additional features. On-going work involves testing the AHE 3.0 VPH-Share authentication system with the VPH-Share Security Proxy. The introduction of a new AHE web client also calls for integration of the AHE client with the VPH-Share master user interface.

Other development plans for AHE involve implementation of several use cases to ensure that AHE is able to run real-world scenarios. As ascertained in the course of discussions with WP5 representatives, this includes using GROMAC for HIV studies in ViroLab, and euHeart simulations. AHE will also implement more complex workflow support, enabling error recovery, batch job submissions and application scheduling. Error recovery will allow users to reconfigure a virtual application when AHE has discovered an error during job submission: once an error is detected AHE waits for the user to correct the error or terminate the application. Batch job workflows will allow users to run simulations which exploit the same application but with a different range of variables per submission. The job scheduler will allow users to run applications concurrently, creating a coupled simulation. We will also investigate more intuitive complex workflow management methods and integration with the workflow engine.

Additionally, AHE will be integrated with external grid libraries, including Steering, SPRUCE emergency submission and advance reservation functions.

Currently, there are no major issues with ACD. However, we are investigating the implementation of OpenID in ACD for authentication as well as implementing database access using Hibernate ORM. This will allow ACD to be more easily deployed.

4.5 Data Access for Large Binary Objects

The Large OBject Cloud Data storagE fedeRation (LOBCDER) is a storage federation service that aims to ensure reliable, managed access to distributed scientific data stored in various storage frameworks and providers. LOBCDER needs to expose and share large amount of data within the VPH-Share community without having to modify existing applications and services.. Not only because the complexity of managing and maintaining the existing software stack is already high, but also because some storage that holds datasets is outside our administrative domain. Therefore, LOBCDER loosely couples with a variety of storage resource that may use different technologies such as S3, Swift, and GridFTP. As a result, LOBCDER transparently integrates multiple autonomous storage resources and exposes all available storage as a single namespace to provide benefits such as:

- distribution of data to ensure accessibility and reduce latency
- dynamic addition and removal of storage resources
- provisioning virtually limitless storage capacity
- improved collaboration through data sharing
- simplifying the process of developing scientific applications

The LOBCDER service is divided into three main layers. Figure 25 shows the conceptual design of LOBCDER, which includes the Frontend, the Resources and the Backend. In the top layer LOBCDER presents a logical file system as a Web-based Distributed Authoring and Versioning repository (WebDAV). This logical file system is stored and queried in the persistence layer, which is a database holding only the metadata of the files stored in the physical storage resources. Moreover, every logical file can be physically stored in more than one storage resource.

More specifically, the purpose of the Frontend is to provide access and control through standardized interfaces. This removes the need for developing and maintaining specialized clients. LOBCDER's Frontend is a WebDAV servlet, which presents the entire available storage space via HTTP. Although HTTP is not specifically designed as a file transfer protocol, it has proven suitable for file transfers. Content Delivery Networks, such as YouTube, are able to upload and download large files[1] with the use of HTTP. Additionally, the LOBCDER frontend provides a RESTful service that allows clients to get and set extended attributes and metadata.

The purpose of the Resource layer is to create a logical representation of the physical storage space. The first part of the Resource layer contains an implementation of the WebDAV resources and the resource catalogue. The WebDAV resources implement the WebDAV specification while the catalogue is responsible for querying the persistence layer for logical resources. The second part of the Resource layer contains the logical resources' metadata and their storage resources. Finally, the Backend layer provides the necessary abstraction to uniformly access physical storage resources. The main component is a virtual resource system client, which is able to access any physical resource system thus providing a uniform API to the components above it.

[1] According to Youtube users can upload files up to 20 GB each

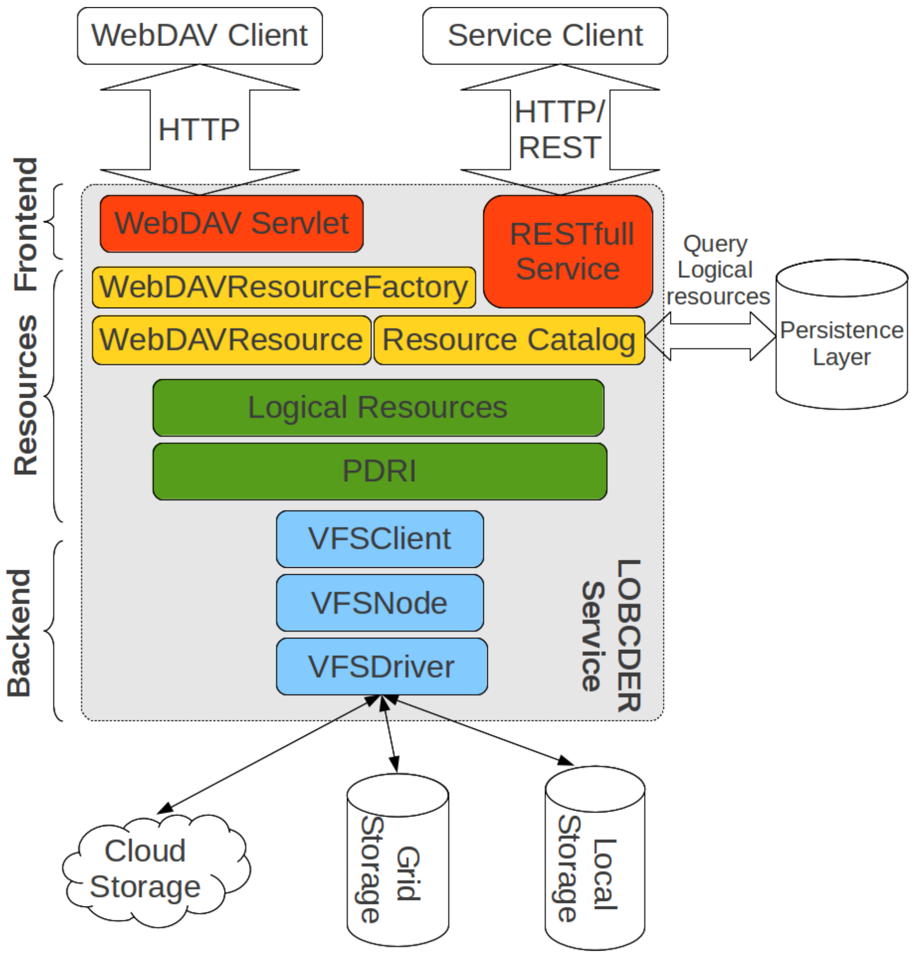


Figure 25: Conceptual design of LOBCDER.

4.5.1 Status of the Prototype

As described in deliverable D2.2 (5), the Frontend enables the following operations on the WebDAV resources:

- **Copy** creates a duplicate of a source dataset (identified by its Logical Data Resource Identifier or LDRI, which is an URI (Universal Resource Identifier) pointing to the dataset in LOBCDER), at a destination data resource
- **Delete** removes a data resource
- **Lock/Unlock** locks and unlocks access to a data resource while its owner is modifying it
- **Make Collection** creates a new collection (or bucket) at the location of the specified LDRI
- **Move** moves a data resource to the location specified by an LDRI
- **Get Properties** retrieves properties for a data resource such as mime type, and length
- **Upload/Download** enables basic (non-DAV-specific) interaction with storage resources.

Currently LOBCDER is able to provide all of these operations, except for the lock/unlock operation. More specifically, the current state of the LOBCDER prototype is depicted in the class diagram shown in Figure 26. In addition to the WebDAV frontend LOBCDER also provides a RESTful service that enables clients to get and set extended attributes and metadata. An extension to the file attributes handled by LOBCDER is the **supervised** property and **lastValidationDate**. The **supervised** property means that a file is supervised by the DRI (see Section 4.6), while **lastValidationDate** is the date when the file was validated DRI. More specifically the service has the following features:

- A client can get the list of metadata related to a set of files by a GET query in the form of http://host/lobcder/rest/Items?path=/path/to/parent/dir in JSON or XML format. The result is the metadata for all files, which are located under this parent directory and its subdirectories.
- To set the "supervised" property for all files located under a specified directory, the client can send a PUT query in the following form: http://host/lobcder/rest/Items/TRUE?path=/path/to/parent/dir
- To reset the "supervised" property for all files located under a specified directory, the client can send a PUT query in the following form: http://host/lobcde/rest/Items/isSupervised/FALSE?path=/path/to/parent/dir
- To get the metadata of a file specified by its UID (in JSON or XML format) the client can send a GET query: http://host/lobcder/rest/Item/{uid} (where **uid** is the unique file identifier).
- To set the "supervised" flag for a specific file the client can send a PUT query: http://host/lobcder/rest/Item/{uid}/supervised/{flag} (where **uid** is the unique file identifier and **flag** is either TRUE or FALSE).
- To set the checksum value for a file the client can send a PUT request: http://host/lobcder/rest/Item/{uid}/checksum/{checksum} (**checksum** is an integer)
- To set the last validation date value for a file the client can send a PUT request: http://host/lobcder/rest/Item/{uid}/lastValidationDate/{lastValidationDate (**lastValidationDate** is the number of seconds elapsed since the beginning of the current Unix epoch)

In addition to introducing the RESTful API, LOBCDER has updated the following:

- The catalogue is built up from scratch based on JDBC and extensively uses indexes and SQL for efficient query operations.
- Operations that do not involve physical data are performed on the catalogue level only. Here, SQL is extensively used to perform most operations on the database side in order to minimize remote procedure call (RPC) impact.
- For the COPY WebDAV operation a COPY-ON-WRITE strategy is used: this operation only updates the catalogue. A new entry is created with references to the same physical data. If the file is subsequently modified, a new physical record is created for it and the references are updated.
- For the DELETE WebDAV operation, the catalogue uses delayed delete. This means that only catalogue entries are deleted. Here SQL is used as much as possible to minimize RPC impact. Physical entries are asynchronously deleted in the background.

At this point we will provide a description of the classes listed in Figure 26.

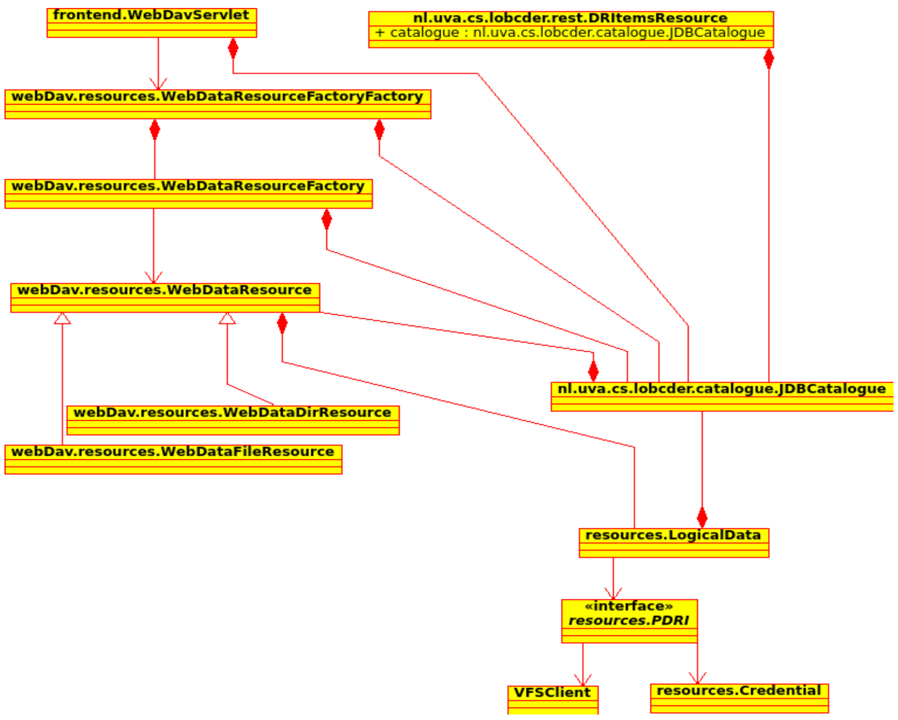

frontend.WebDavServlet
nl.uva.cs.lobcder.rest.DRItemsResource
+ catalogue : nl.uva.cs.lobcder.catalogue.JDBCatalogue
webDav.resources.WebDataResourceFactoryFactory
webDav.resources.WebDataResourceFactory
webDav.resources.WebDataResource
nl.uva.cs.lobcder.catalogue.JDBCatalogue
webDav.resources.WebDataDirResource
webDav.resources.WebDataFileResource
resources.LogicalData
«interface»
resources.PDRI
VFSClient
resources.Credential

Figure 26: LOBCDER class diagram.

WebDataResourceFactoryFactory: As the name suggests, this class is responsible for creating WebDataResourceFactory instances. The **WebDataResourceFactoryFactory** class is instantiated when a new request comes from client through the **WebDavServlet**. Moreover, this class is meant for configuring and instantiating the **WebDataResourceFactory** class.

WebDataResourceFactory: This class's role is to create **WebDataResource** resources, and return them to the **WebDavServlet**. Thus, the main method of this class is **getResource**. This method locates an instance of a **WebDataResource** for a given URL (Universal Resource Location).

JDBCatalogue: The instance of this class is created on service startup. Then it is available through JNDI. The role of this class is to query the persistence layer for **LogicalData** and **PDRI** entries. Most frequently used methods of this class are **registerOrUpdateResourceEntry** and **getResourceEntryByLDRI**. **registerOrUpdateResourceEntry** registers new instance of a **LogicalData** in the persistence layer or updates it. **getResourceEntryByLDRI** is quite straightforward: given a Logical Data Resource Identifier (LDRI), the method should return a single one LogicalData entry.

WebDataResource: This is a superclass for all the WebDAV resources (file and folder). The methods not currently implemented in this class are **authenticate**, **authorise** and **checkRedirect**. WebDAV provides the ability to authenticate and authorise each resource separately, providing better granularity for user permissions. These two methods are implemented in subclasses in order to reflect the permissions each member of the VPH-Share community has with respect to data resources. Additionally, the **checkRedirect** method will be implemented at a later stage, when LOBCDER is deployed on multiple hosts. Under this setup LOBCDER will be able to redirect incoming calls to the nearest LOBCDER instance.

WebDataDirResource/**WebDataFileResource**: These two classes are subclasses of **WebDataResource**, and provide a representation of the logical and physical data to a WebDAV client.

LogicalData: This class encapsulates the logical entries held on the persistence layer together with the physical data stored on the cloud services. Hence, this class holds a LDRI, which uniquely identifies the logical resource, and a set of **PDRIs** that hold replicas of the physical data. When a new **LogicalData** object is created, it is not necessary to have any physical data associated with it either because the client may create an empty file, or because this instance represents a folder or a collection. Moreover, this class holds a metadata that provide creation modification dates, mime types, etc.

PDRI: This class is a representation of the storage resource that holds the actual data. Since there are many different storage resources this class needs to have a **Credential** member that will provide access to storage resources. This class can hold credentials such as passwords, certificates, etc.

VFSClient: In order to be able to interact with the physical storage resources, each **PDRI** uses a **VFSClient**. This class provides numerous methods for manipulating and managing data on the remote storage resource.

To obtain a better understanding of how LOBCDER works, the specific sequence of operations involved in processing user requests is presented below.

1. A WebDAV client sends a request to the **WebDavServlet**.
2. The **WebDavServlet** calls the **WebDataResourceFactoryFactory** to create a **WebDataResourceFactory** instance.
3. A **WebDataResourceFactory** instance is returned to the **WebDataResourceFactoryFactory**.
4. The **WebDataResourceFactory** class creates an **IDLCatalogue** in order to be able to query the requested resource.
5. The **WebDavServlet** will now call the **getResource** method from the **WebDataResourceFactory** class.
6. The **WebDataResourceFactory** will use its **IDLCatalogue** to query the requested resource from the persistence layer.
7. The returned entry is an instance of **LogicalData** class which contains a **DataSource** that indicate the physical location of the requested resource.
8. At this moment the **WebDataResourceFactory** will get the **DataSource** from the **IDLCatalogue** the requested user has access to.
9. The **WebDataResourceFactory** will instance the **WebDataResource**, and set its DataSource.
10. Finally, the **WebDataResourceFactory** will return the requested **WebDataResource** back to the **WebDavServlet**, where it will respond to the WebDAV client.

4.5.2 Future work

4.5.2.1 Performance

Since the design goal for LOBCDER is to federate data located in the cloud, the aim is to optimize the implementation to handle extensive datasets prior to developing distributed solutions in order to address scale issues. Moreover, tests indicate that LOBCDER performance is heavily dependent on the performance of its backend and VFSDriver. This is especially true for uploading files to the cloud. To circumvent this bottleneck we will use a local cache so that when a client uploads a file the data is first uploaded to a local cache and from there transferred to the appropriate storage using a batch job.

4.5.2.2 Authentication

In parallel with this effort we are developing a security mechanism that protects LOBCDER resource handles against unauthorised access. This implies that for each request, we need to know who is accessing a given resource (credentials) and make a decision on whether the requestor has sufficient rights (permissions). For a typical set of credentials consisting of a username and a password, the HTTPS authorisation a scenario is depicted in Figure 27.

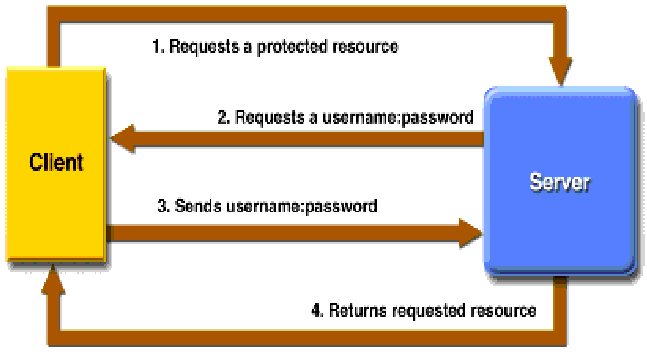

1. Requests a protected resource
2. Requests a username:password
Client
Server
3. Sends username:password
4. Returns requested resource

Figure 27: HTTP basic authentication.

According to the VPH-Share security convention, each request includes a security token passed along with the request parameters. The security subsystem (which is also built into LOBCDER) validates user credentials and makes authentication decisions.

The security token is extracted from the HTTP request and passed to the validation service running on the front node. The token is checked by the service and the result contains the VPH user name of the client and the list of the roles associated with this client. The result is cached locally to minimize the number of RPC calls.

4.5.2.3 Authorisation

When an authenticated user tries to access a LOBCDER resource, the system has to decide if this user is authorised. Any logical path in LOBCDER (and therefore associated **LogicalData** resource) has an Access Control List (ACL). ACL consists of the list of the roles that are allowed to access the resource and permissions set by the resource owner for these roles. The request is considered authorized if the user belongs to a specified role and this role is allowed to perform the requested operation (read or write), or if the user is an owner (creator) of the requested resource. Here "user role" is a synonym of the user's group.

This ACL is kept in an internal metadata table. A resource may have as many roles in its ACL as needed. To manipulate the content of ACL, LOBCDER may require an additional extension, going beyond its data access (WebDAV) interface. The technology candidate for implementing this extension is a RESTful service.

For bulky WebDAV operations such as MOVE, COPY and DELETE performed on the collections (logical folders) the permission checks are performed on the database side using stored procedures. These operations involve catalogue updates only and do not involve synchronous operations on the physical layer (DELETE does it in the background). In this way LOBCDER avoids performing an RPC operation on each entry in a bulk request, which results in a dramatic performance improvement.

4.5.2.4 Data encryption

We propose to use LOBCDER for permanent or temporary data storage. It would be mounted (using davfs2) on each instance including those in the public cloud. This would allow processing chunks of data in memory without the need to write them, even temporarily, to any form of "local instance storage" provided by the cloud service provider (and over which we have no control). Combining this strategy with the immediate sanitization of used parts of RAM and security monitoring to detect intruders/malware would make it very difficult to steal sensitive data. We would also ensure that only small parts of the data would be exposed (unencrypted data currently being processed in memory). Finally, as we wouldn't need to store data on the instance's local hard drive, it would not become necessary to ensure removal of said data.

Of course LOBCDER itself would need to store the data. As backend it may use either private storage clouds (e.g. Swift-based) in which case data is stored in a trusted environment, as well as public ones (Amazon S3, Rackspace CloudFiles etc.) in which case additional actions are needed to ensure security. We propose two encryption-based solutions to satisfy this goal.

LOBCDER encryption

The LOBCDER encryption scenario is shown in Figure 28.

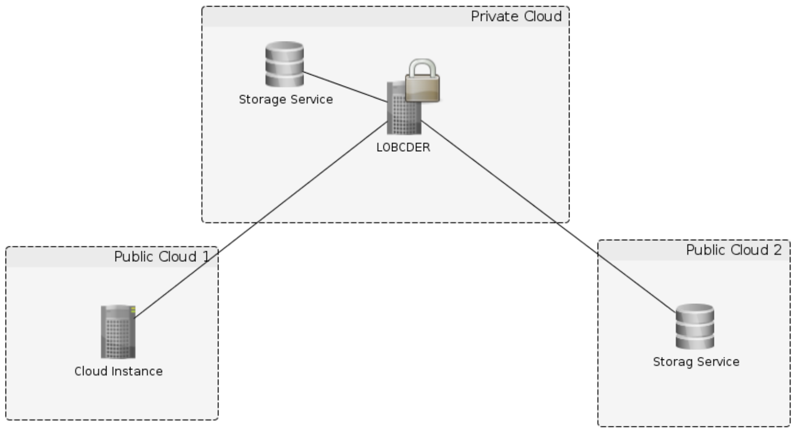


Figure 28: LOBCDER based encryption - data owner needs to trust LOBCDER however the encryption process itself is transparently handled by the VPH platform.

In this scenario LOBCDER itself is responsible for encrypting the data. The symmetric key used for encryption/decryption will be entered during startup and stored in memory. This, combined with placing LOBCDER in the trusted zone, should provide an additional layer of security for most of the processed data. In this scenario the platform would offer full functionality, such as seamless access to the data using DAV client/davfs2 as well as the Master Interface. LOBCDER will also control access to the data so the only authorized entities could obtain decrypted data. Of course in this solution the data provider needs to trust LOBCDER.

End to end encryption

This scenario is depicted in Figure 29.

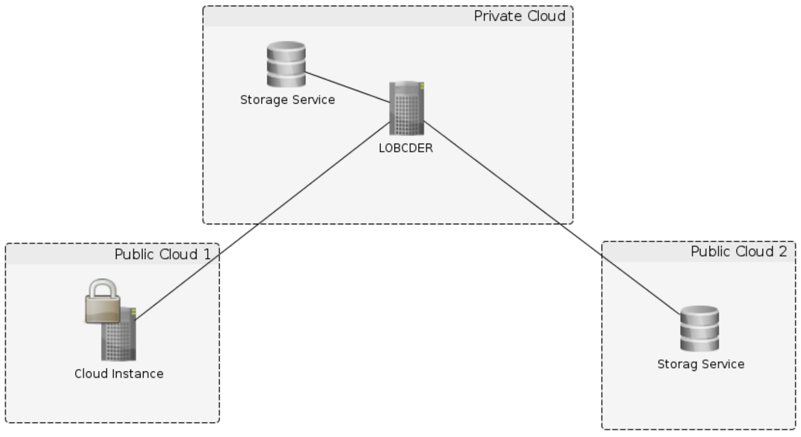


Figure 29: End-to-end encryption – nobody except the data owner (even WP2 administrators) can decode information stored by the platform. The data owner needs to encrypt/decrypt the data.

If mandated by the data provider, e.g. due to stricter confidentiality requirements, an alternate solution could be applied. The data could be encrypted and decrypted by the Atomic Services operating in the cloud. The project could assist developers in this respect by suggesting some standard tools (such as OpenSSL) that would be helpful in executing this scenario. LOBCDER encryption could then be deactivated (so that data encrypted in the "end to end" fashion wouldn't be needlessly re-encrypted by LOBCDER). In this use case only the data provider knows the key and can therefore decrypt the data. An obvious drawback of this solution is that standard VPH tools (such as the Master Interface) wouldn't be able to assist the user in a decryption process, however they may still allow retrieval of encrypted data, to be decrypted later by the owners themselves.

Selection of one of the above mentioned scenarios will be made on the basis of consultations with application providers and will form part of the work in T2.4 during the third year of the Project.

4.6 Data Reliability and Integrity

As presented in Deliverable 2.2 (5), the goal of the Data reliability and integrity (DRI) tool is to ensure reliable use of the biomedical datasets manipulated with the use of VPH-Share applications and tools. Simulations results and inferred medical outcomes must be based on reliable data. Due to the large size and long-term persistence of medical data files, special reliability and integrity mechanisms should be enforced on top of Cloud storage. Thus, the infrastructure developed in Task 2.5 needs to be able to perform the following tasks:

- periodic integrity checks on data objects with the use of hash algorithms,
- facilitating storage of multiple copies of data on various Cloud platforms,
- tracking the history and origin of binary datasets.

4.6.1 Status of the Prototype

The DRI Runtime is responsible for enforcing Task 2.5 data management policies. It keeps track of managed components and periodically verifies the accessibility and integrity of the managed data. As designed, the Runtime assumes the form of a generic (i.e. non-application-specific) Atomic Service in the WP2 infrastructure. Thus, it can be managed and deployed by Atmosphere tools, just like any other type of Atomic Service. For scalability purposes, multiple instances of DRI Runtime may coexist in the system, integrated into a coherent platform by sharing a common registry (the WP2 persistence layer), although at present only a single instance of the prototype service has been deployed on the computing resources contributed by CYFRONET (see Section 4.3 for details).

In line with the Atomic Service specification, the DRI service has been deployed into a virtual machine and further registered with Atmosphere mechanisms for automatic management. It is currently able to monitor and validate the data sets registered with the Atmosphere Internal Registry and present in the Swift data storage that is part of the VPH-Share cloud federation. Figure 30 presents the architecture of DRI Runtime.

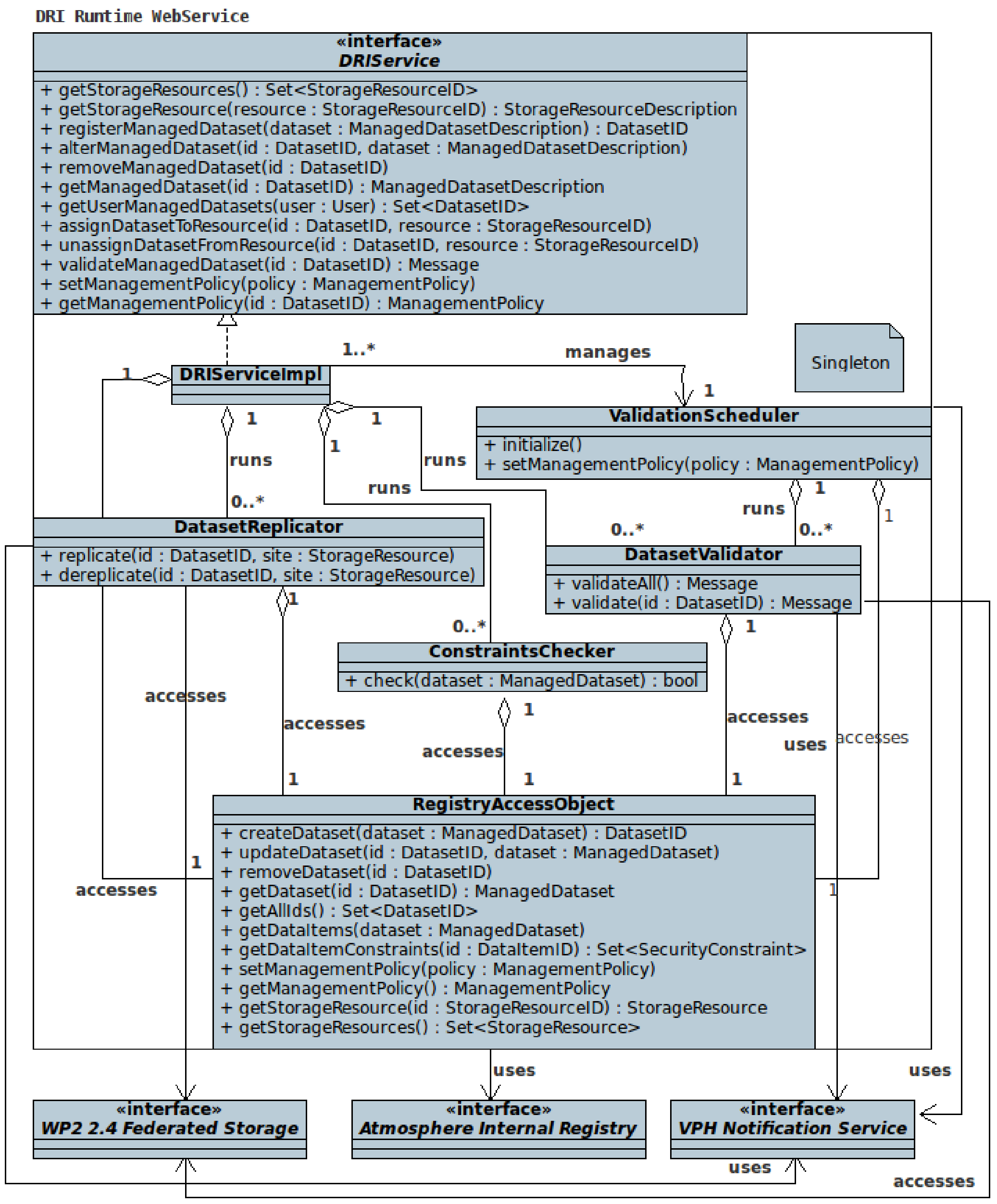

DRI Runtime WebService
«interface»
DRIService
+ getStorageResources() : Set<StorageResourceID>
+ getStorageResource(resource : StorageResourceID) : StorageResourceDescription
+ registerManagedDataset(dataset : ManagedDatasetDescription) : DatasetID
+ alterManagedDataset(id : DatasetID, dataset : ManagedDatasetDescription)
+ removeManagedDataset(id : DatasetID)
+ getManagedDataset(id : DatasetID) : ManagedDatasetDescription
+ getUserManagedDatasets(user : User) : Set<DatasetID>
+ assignDatasetToResource(id : DatasetID, resource : StorageResourceID)
+ unassignDatasetFromResource(id : DatasetID, resource : StorageResourceID)
+ validateManagedDataset(id : DatasetID) : Message
+ setManagementPolicy(policy : ManagementPolicy)
+ getManagementPolicy(id : DatasetID) : ManagementPolicy
Singleton
DRIServiceImpl
1..*
manages
1
ValidationScheduler
+ initialize()
+ setManagementPolicy(policy : ManagementPolicy)
runs
0..*
DatasetReplicator
+ replicate(id : DatasetID, site : StorageResource)
+ dereplicate(id : DatasetID, site : StorageResource)
DatasetValidator
+ validateAll() : Message
+ validate(id : DatasetID) : Message
ConstraintsChecker
+ check(dataset : ManagedDataset) : bool
accesses
uses
RegistryAccessObject
+ createDataset(dataset : ManagedDataset) : DatasetID
+ updateDataset(id : DatasetID, dataset : ManagedDataset)
+ removeDataset(id : DatasetID)
+ getDataset(id : DatasetID) : ManagedDataset
+ getAllIds() : Set<DatasetID>
+ getDataItems(dataset : ManagedDataset)
+ getDataItemConstraints(id : DataItemID) : Set<SecurityConstraint>
+ setManagementPolicy(policy : ManagementPolicy)
+ getManagementPolicy() : ManagementPolicy
+ getStorageResource(id : StorageResourceID) : StorageResource
+ getStorageResources() : Set<StorageResource>
«interface»
WP2 2.4 Federated Storage
«interface»
Atmosphere Internal Registry
«interface»
VPH Notification Service

Figure 30: DRI Runtime architecture.

At the core of the prototype lies the **DatasetValidator** class, which performs periodic validation of data items represented in the Atmosphere Internal Registry. This class is configurable by a dedicated set of parameters stored in the Registry. As part of our ongoing development work, we are implementing an end-user interface that will enable administrators to manage the runtime parameters of the DRI service. Furthermore, the DRI prototype provides a service frontend that can be used to register, unregister and query the status of managed datasets.

Figure 31 presents the persistence schema upon which DRI Runtime bases its operation. This schema is managed by the Atmosphere Internal Registry which thus provides a uniform persistence layer for DRI (as well as for other core components of WP2).

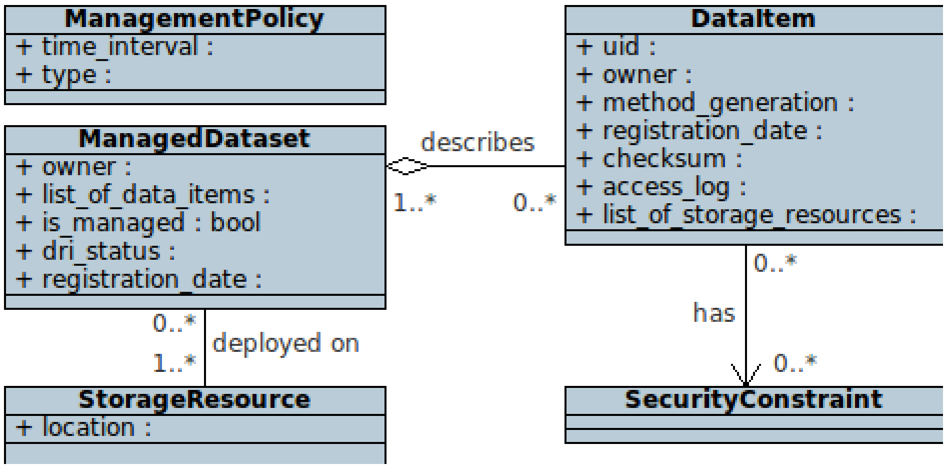


Figure 31: DRI data model.

DataItem is a virtual concept shared between DRI and LOBCDER. It can describe either a single file or a collection of files. A **DataItem** is optionally associated with a **ManagedDataset** object, which implies that it needs to be validated by DRI, in accordance with the parameters stored within that instance of **ManagedDataset**. **ManagementPolicy** specifies the general operating characteristics of DRI and can be configured by system administrators.

The DRI Runtime prototype is developed in Java. The service is deployed to a an application server residing on a virtual machine currently provided by CYFRONET and exposes a RESTful Web Service API (Jersey implementation) backed by the DRI component internally. Quartz Scheduler (18) is utilised for scheduling validation routines periodically.

4.6.2 Future work

Having deployed the first prototype of DRI our focus will now shift from low-level conceptual and implementation-oriented details to providing a more robust set of interfaces and measuring the performance of various data validation algorithms. The DRI component has been integrated with the Data Browser extension to the Master Interface and can be managed via this extension. We intend to deploy a notification service (in collaboration with WP6) where any emerging problems could be communicated to system administrator and resource owners, either synchronously (e.g. via e-mail) or through notifications stored in AIR and displayed in the Master Interface whenever an authorised user has logged in.

The service is also in the process of being secured with authenticity tokens provided by task T2.6. When deployed in production mode, it will contact the common Policy Decision Point to guard against unauthorised access.

4.7 Security Framework

The security framework is responsible to ensure that all Atomic Services in the system are properly secured. To do so, any communications with these services are properly encrypted and have their authorisation checked before actually invoking any of the services.

4.7.1 Security requirements

4.7.1.1 Scope

In order to explain the scope of the requirements of the security framework, it is important to remember that VPH-Share is a platform that integrates a dynamic set of heterogeneous applications, each of them with its own security requirements. As such, the security framework does not aim to address all the security requirements that are specific to each integrated application, especially when new applications with new requirements can be added dynamically.

Hence the VPH-Share Security Framework will deal with the security issues resulting from the integration of these applications into a common and publicly accessible framework, but the intrinsic security requirements that are specific to each application will remain as the responsibility of the associated application developer.

VPH-Share will provide a generic mechanism to enable application developers to define the security constraints of each application based on information relative to the user, which in turn will be stored and retrieved from a common authentication platform.

4.7.1.2 Authentication

One of the goals of VPH-Share is to help augment the VPH initiative by providing a reliable and consolidated infostructure. To help foster and cement trust relationships with existing VPH users, the system will try to reuse the same authentication platform used for the BiomedTown portal. This will provide a single sign-on mechanism for the users of both BiomedTown and VPH-Share applications.

4.7.1.3 Security perimeter

As explained in Section 4.7.1.1, the security framework will deal with the security issues resulting from the integration of applications. This, in practice, means that the security framework will protect the integrated platforms from unauthorised access from outside the VPH-Share platform.

4.7.1.4 Atomic services

VPH-Share applications, also referred to as Atomic Services in this document, will be supported with:

- **Privacy assurance** by encrypting all incoming and outgoing communication to/from the VPH-Share platform.

- **Master interface authentication** – LOBCDER will reuse the security proxy to carry away the authorization process. Specifically, it will use:
  - a permissions mechanism based on a role attribute stored in the attributes of the user logged through the master interface
  - An expiration limit for the security token coming from the master interface
  - A verification of the signature of the security token from the master interface

The security framework aims to transparently incorporate these features as a wrapper around the Atomic Services. Note, however, that both the security roles and the access rules are specific to each Atomic Service and hence it is the responsibility of the application developers to define such rules and the user attributes needed to validate these rules..

4.7.1.5 LOBCDER / WebDAV

The LOBCDER service provides a standardised WebDAV interface to unified distributed file system and with respect to security, it will at least provide:

- **Privacy assurance** by encrypting all incoming and outgoing communication to/from the platform;
- **Basic user/password authentication** – LOBCDER cannot reuse any of the complex authorisation features provided by the Atomic Services because the WebDAV standard does not support them. Thus, LOBCDER will alternatively use basic user/password mechanisms to ensure compatibility with the WebDAV standard and third-party WebDAV clients.

Although LOBCDER authentication requirements differ slightly from the approach suggested for Atomic Services, a common model can be applied by taking these inconsistencies into account.

4.7.1.6 Workflow Management / Taverna

The Taverna workflow system will have the same security requirements as the Atomic Services, with the only addition that applications may be launched from a command line instead of a web client. It is acceptable for the authentication process to be performed from the command line, by having the Taverna workflow system supply authentication parameters to the identity platform through a secure API.

4.7.1.7 Performance

The security layer should not introduce significant overhead within the system or, at least, it should not be larger than its other working components within ASI. The initial goal for the security proxy execution time is set on 1 second, subject to server workload and the impact of other system components.

4.7.1.8 External services

VPH-Share includes a search engine to look for additional external services to be used conjunctly with VPH-Share Application Services. However, VPH-Share has no way to guarantee the security of these external services at all. Therefore, VPH-Share will follow a classic trust/deny mechanism, meaning that external services will be initially labelled as untrusted, and access to them banned from the firewall configuration of the system. However, the system will also maintain a list of trusted external services. Hence, the correct procedure for accessing these services will be to ask the system administrator for access to a given service. An administrator will be responsible for granting such access (or denying it) and configuring the firewall appropriately. Of course, open (unsecured) external services deployed under a known IP can always be accessed by VPH-Share Atomic Service Instances and the applications deployed therein.

4.7.1.9 Security policies

As explained above, the system provides the ability to define access criteria for services based on the attributes assigned to the user. These policies can be as simple as a role-based mechanism or as complex as a combination of policy rules depending on more complex attributes.

An example of a simple role-based policy could be: "**If** the user **has the role** *paediatrics* assigned, **then** they should be granted access to the Paediatrics database service".

An example of a complex policy rule could be: "**If** the user **has at least** a bachelor's degree in medicine **and has** signed a confidence agreement **then** grant access to all medical stores".

Security policies can be a very powerful mechanism for developers to express accessibility constraints via rules for services. However, there is no way that the VPH-Share platform can know them beforehand, as they are specific to each service. As a result, it will be the responsibility of application developers to infer these criteria from the application requirements and user attributes provided by the Master Interface.

The administration of the security system will have to be integrated with the Atmosphere Internal Registry via the Master Interface, allowing application developers to update the policy rules associated with their services (even already instantiated ones). It will therefore be possible to maintain and update the access criteria for services without the need to define a new service template in each case.

To express policy rules the project has chosen XACML (eXtensible Access Control Markup Language) (19) for the following reasons:

- It is an already established industry standard
- It is specifically designed for defining access control rules and policies, which is exactly what our policy engine is meant to evaluate
- It is feature-rich; in fact it provides a number of features not yet supported by our system, which enables future improvements if needed
- There exist several open source implementations of XACML
- The design of the security proxy is a match for the conceptual architecture proposed in the XACML specification
- It is not proprietary

The current implementation of the system already includes an XACML compliant engine, which supports complete XACML policies. However, for the moment only role-based policies are included in the system. More complex policies are foreseen to be included in the future.

4.7.2 Security for Atomic Service invocations: the Security Proxy

The Security Proxy, which is preinstalled on any Virtual Machine which hosts an Atomic Service works jointly with the Reverse Proxy, which is in charge of forwarding all incoming requests from a public IP address to private addresses and balancing the workload of the system. Note, that, as mentioned in Section 4.3.1, the actual Atomic Service Instances sit behind an IP forwarding Reverse Proxy so that the ASIs do not expose their private IP addresses.

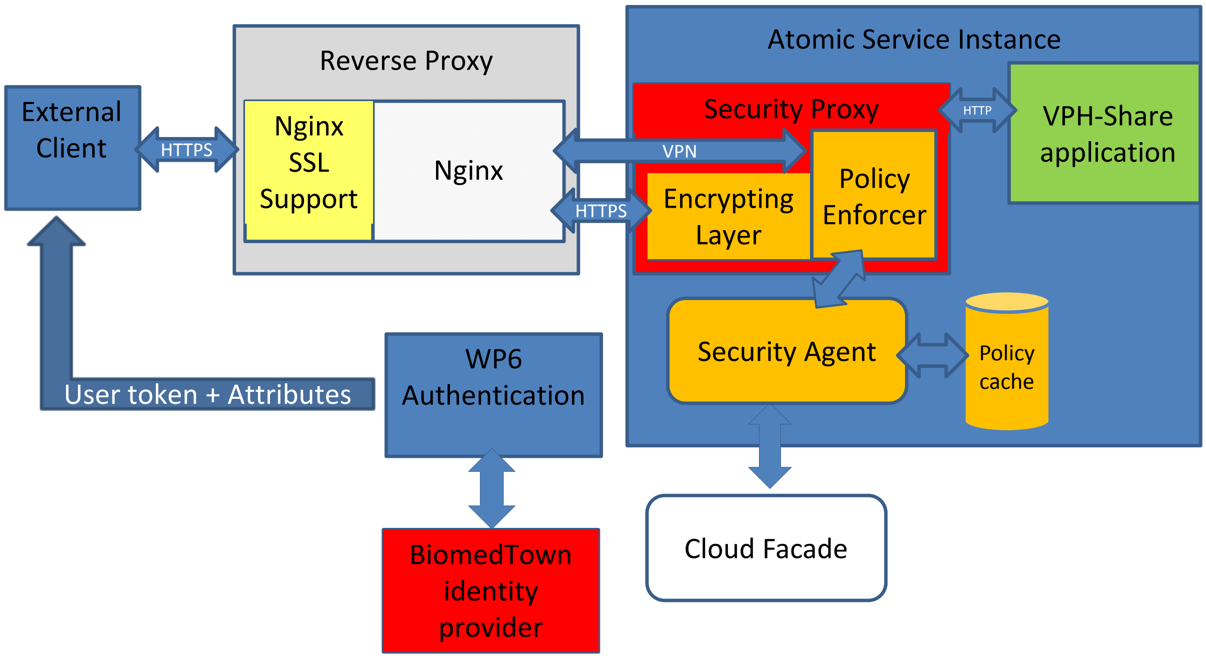


Figure 32: Overview of WP2 security mechanisms.

Figure 32 presents a schematic view of how the Security Proxy integrates with the Reverse Proxy, external clients, Atomic Services and WP6.

As the first step, the user authenticates via the Master Interface, which performs the necessary actions, including forwarding the authentication request to the BiomedTown identity provider and retrieving all user attributes. Subsequently, this authentication information (user + attributes) is signed by the Master Interface and included in the password field of the header of the HTTP request, which will travel through the whole invocation.

To further secure data transmission, all communication between the different servers is encrypted at the HTTP level by using the SSL (Secure Socket Layer) protocol over HTTP (HTTPS). This implies that the Reverse Proxy decrypts the message, performs any calculations required for message forwarding, and encrypts it again prior to contacting the service recipient. Dealing with SSL can be easily performed on the Nginx server by configuring the Nginx HTTP SSL module (see the Nginx HTTPS Module (20) for details).

On the Atomic Service Instance side, all HTTP requests are intercepted by the Security Proxy, which listens to incoming requests on configurable ports and performs two tasks: decrypting messages and checking for authorisation for a given service based on the attributes of the user provided along with the request into the VPH-Share platform. It is capable of extracting user attributes from the HTTP header and confirming their integrity by checking the signature of the field containing it. These user attributes are then used jointly with the XACML security policies applicable to that service that are retrieved from the cloud facade to decide whether to grant or deny access to the service.

If the request is granted, the invocation is forwarded to the service on a local host address. Therefore, all services (including the Apache server) are required to communicate through the local host. An advantage of this design is that it also allows different services to invoke each other on different networks, by passing through the Nginx server responsible for properly forwarding the request. Figure 33 shows how such communication can be accomplished.

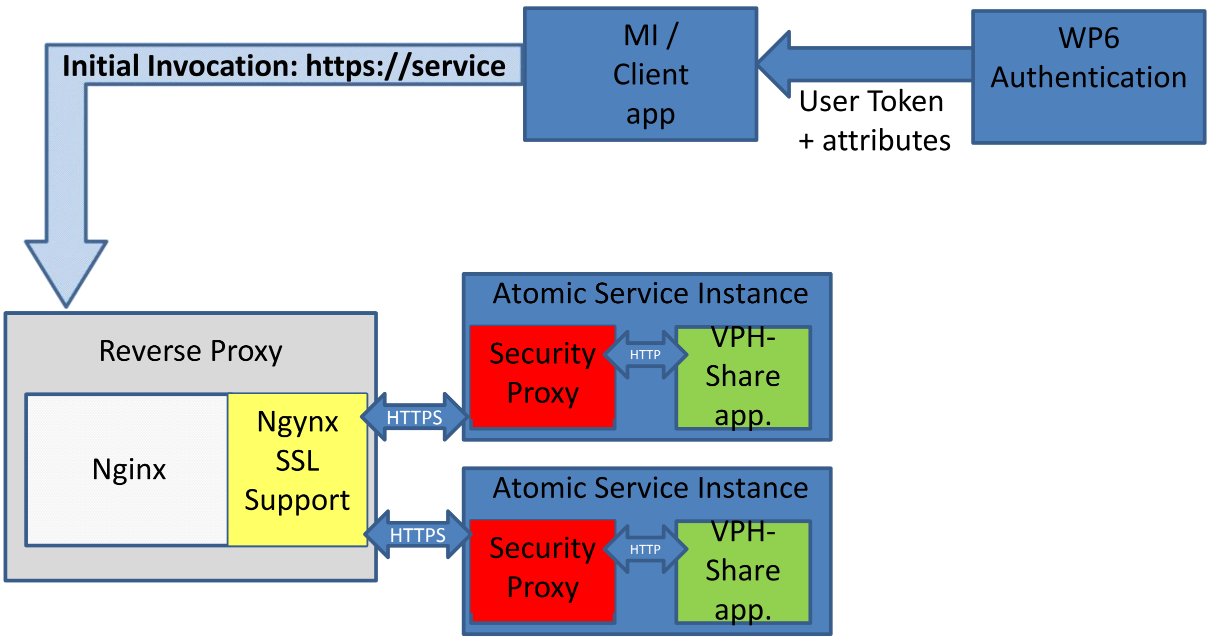


Figure 33: Inter-service communication with Nginx.

The sequence of steps a request takes within the VPH-Share platform is shown in Figure 33. The initial request to invoke any Atomic Service will first be authenticated in Work package 6, which will properly include the user token and attributes. This user token and attributes then need to travel along with the request to the Security Proxy in the HTTP header, and will reach the corresponding VPH-Share application service. When the VPH-Share application wishes to invoke another Atomic Service, it will invoke its corresponding public IP address and follow the same path again without the WP6 authentication, with the original user token travelling through all the invocations.

The presented architecture allows any HTTP-based services, including REST and SOAP services to be secured in a transparent manner. Regarding the Reverse Proxy, the design facilitates the required encryption by simply adapting the configuration files of the Nginx server.

Regarding security policies, they will be retrieved from a central service in the cloud but cached locally in a temporary folder of the Atomic Service Template. Atomic Service policies can be modified either when defining the template or once it has been instantiated through the Atmosphere Internal Registry management console. To do so, each Atomic Service Instance will have a security agent installed, which will download configuration files periodically and whenever they are changed in the cloud facade. This will enable the application developer to update policy rules and define access criteria for the security proxy on a given machine. Figure 34 presents this process.

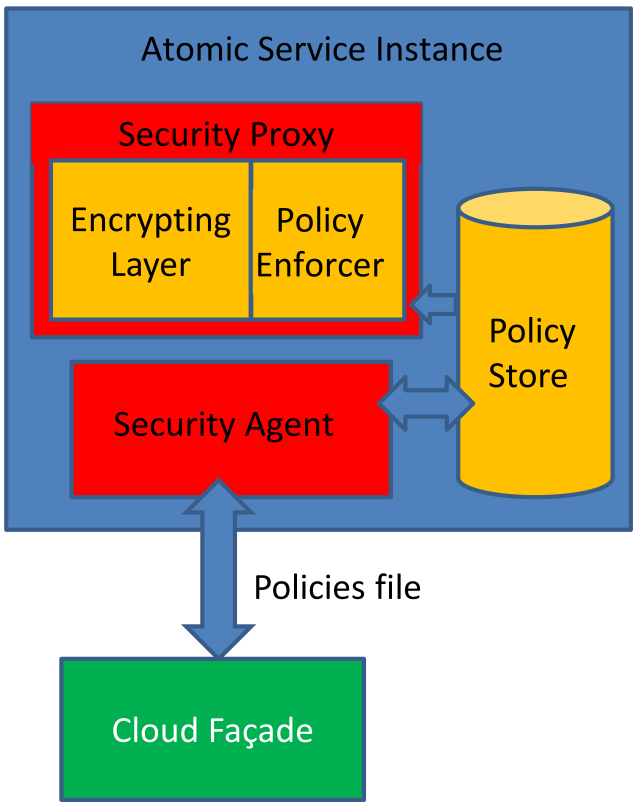


Figure 34: Updating policy rules of the Security Proxy.

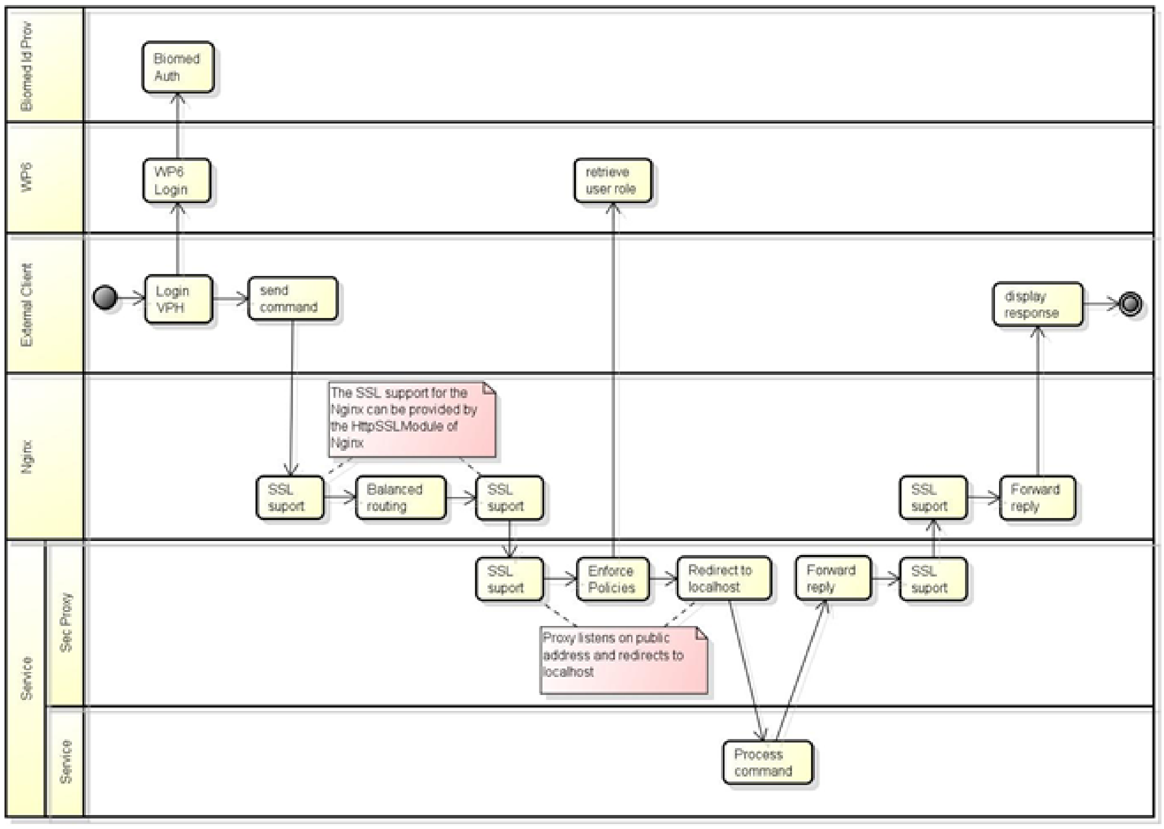


Figure 35: User authentication and authorisation in the context of invoking Atomic Services deployed in a private network.

Figure 35 shows the sequence diagram involved in communication with an Atomic Service, for an arbitrary process launched in any external client application (or within the context of the Master Interface).

The process starts with the external client (or the service which launches a given application in a terminal session on the service host). The client logs into VPH-Share and subsequently issues a service invocation request to a public IP address, which is properly encrypted with SSL. The Nginx proxy decrypts the request, selects the appropriate private IP of the Atomic Service Instance and forwards the request, having again encrypted it with SSL.

Once the request arrives at the local (instance-bound) Security Proxy, it is decrypted, and its security attributes used to decide whether the user is authorised to perform the given operation. If so, the HTTP request is redirected to a local host address of the Atomic Service. Once the service produces a result, it is properly encrypted and sent back to the reverse proxy, which can finally deliver it to the client application.

4.7.3 Interoperability with p-Medicine on Security

The project held a meeting with p-Medicine project with a specific topic of security, whose main conclusion was both p-Medicine and VPH-Share will build a bridge to enable mutual access. The main objective of this bridge will be to translate the security token that each system uses into the format appropriate for the other project so that a service belonging to one system may invoke services on the opposite side. However, because the very nature of p-Medicine requires that each user accessing p-Medicine services is certified by the p-Medicine certification authority, it will remain necessary for each VPH-Share user needing to access p-Medicine services to apply for a p-Medicine account.

4.7.4 Status of the prototype and future work

The initial prototype of the security framework includes the Security Proxy has been deployed on the first Atomic Services produced by WP5. Thus, service communication proceeds in a secure manner and the Security Proxy can be replicated to additional Atomic Services (by inclusion in the host VM).

Current version of the prototype fully supports XACML policy files to be provided by the service developers. Additionally, the system also includes a tool to automatically generate XACML policy files that support RBAC policies and assign them to the correct user.

During the next 12 months, work will focus on:

- Defining XACML policy files for the main features which might require more advanced support from XACML
- Studying the feasibility of Cross Site Scripting protection in the security proxy
- Studying the feasibility of adding auditing capabilities to the security proxy
- Developing a VPH-Share – p-Medicine bridge to ensure interoperability between both systems

5 Software Engineering Aspects

As far as software engineering methods are concerned, we follow the rapid prototyping approach, with short development cycles and frequent validation of results against the expectations of their intended users (whether the WP5 workflow teams or other technical tasks of VPH-Share). Progress reports are collected from WP2 developers on a monthly basis, and corrective actions introduced whenever requested by WP2 management. In addition, CYFRONET operates a WP-wide code repository at https://gforge.cyfronet.pl where the source code of individual components can be collected and managed. We intend to carry on with this scheme in the second year of the Project, focusing in particular on the specifics of end-user interfaces (which are developed as part of WP6, but which involve WP2 to a significant degree).

Table 5 presents a generalised list of technologies exploited by WP2 components. A more detailed technical description of each component can be found in deliverable D2.2 (5), while any discrepancies or deviations from the WP2 design documentation will be explained in Section 4 of this deliverable.

Table 5: List of technologies applicable to each component of the WP2 architecture.

Component	Applicable technologies
Cloud Resource Allocation Management (Atmosphere core services)	Java SE6 components deployed as OSGi bundles to an Apache Karaf container. Nagios framework for instance monitoring. Integration layer based on Apache Camel.
Atmosphere Internal Registry	Semantic registry layer implemented in Ruby on top of MongoDB storage. REST-based service endpoint and custom jQuery-based GUI.
Cloud Execution Environment	OpenStack (Diablo release) private cloud stack deployed at CYFRONET (HP ProLiant BL2x220c G5, 2 x Intel Xeon L5420, 16GB RAM, 120 GB internal HDD); 3 TB attached storage (NFS).
High Performance Execution Environment	Middleware connections for QcG (using the QcG Java SDK), Unicore (using the UCC java library) and Globus (using the JGlobus library). SOAP-based service endpoint.
Data Storage Federation	Support for OpenStack Swift storage resources; data exposed by a WebDAV-like API (mimicking a true WebDAV server). Service-based control endpoint for management actions.
Data Reliability and Integrity	Standalone Java-based service, integrated with OpenStack storage. Managed dataset metadata stored in AIR. Service-based control endpoint for management actions.
WP2 security components	Java-based Security Proxy implemented as plugin for the Apache server framework, deployable on any Atomic Service template.

6 Features to be Implemented in the Third Year of the Project

Since the software components presented in this deliverable are still considered to form a prototype deployment, the third year of the Project will be primarily devoted to ensuring their robustness and enhancing performance. During this time fewer new backend features will be implemented – instead, more attention will be paid to the end user interfaces, making sure that the platform can be exploited by the end users without support from developers.

In Task 2.1 a redesign of the UI frontend is planned. In the development mode, users will be provided with control over some aspects of the Atmosphere services which have hitherto relied on default values, such as selection of appropriate service flags (tagging services as shared and selection of appropriate port redirection options for private cloud sites). In addition, more sophisticated resource monitoring and performance optimization algorithms will be integrated with the Atmosphere Management Service.

In Task 2.2 care will be taken to ensure integration of additional private cloud sites and resources procured from public cloud providers. Existing cloud sites, such as the one at CYFRONET, will be extended and hardware performance adjustments will be made to increase the end-users' Quality of Experience when interacting with the cloud platform.

Task 2.3 will provide a more robust set of interfaces for the AHE service, ensuring that the HPC infrastructure overlaid by this service can be easily accessed by VPH-Share application components.

In Task 2.4 further performance optimization and extension of the storage infrastructure is planned. In addition, LOBCDER intends to ensure runtime data encryption and more fine-grained control over the security constraints applicable to binary files.

In Task 2.5 the DRI tool will be supplied with a new set of user interfaces targeted specifically at resource administrators and owners of datasets who need to be appraised of the status of their hardware and data. The service will also be subjected to performance improvements and further integrated with the LOBCDER metadata schema.

Task 2.6 will continue work on tools and services which will enable AS developers to prepare and fine-tune security policies. The Security Proxy component will be extended to handle XACML security constraints and a set of user interfaces will be contributed to the VPH-Share Master Interface.

Detailed descriptions of the plans for the third year of the Project in each WP2 tasks can be found in Section 4.

7 Summary

The advanced prototype of the cloud platform is fully capable of running the VPH-Share pilot application services with the use of private cloud resources contributed to the Project by partner institutions. In addition, a number of p-Medicine application services are also being deployed in the infrastructure on a trial basis. During the third year of the project our focus will be to extend the cloud infrastructure to include public (bought-in) cloud resources and enable computations to be performed on hardware procured from commercial cloud vendors. We will also continue our work on further refinement of WP2 interfaces – for end users as well as for developers of external services and application clients.

8 References

1. VPH Consortium. VPH2012 workshop, September 2012, London. [Online].; 2012. Available from: http://www.vph-noe.eu/vph-events/details/199-vph-share-workshop-in-vph2012.

2. Bubak M, Nowakowski P, Bartyński T, Gubała T, Harężlak D, Kasztelnik M, et al. Cloud Platform for Medical Applications. In Proceedings of eScience 2012; 2012; Chicago.

3. Nowakowski P, Bartyński T, Gubała T, Harężlak D, Kasztelnik M, Meizner J, et al. Managing Cloud Resources for Medical Applications. In Proceedings of the Cracow Grid Workshop 2012; 2012; Krakow.

4. VPH-Share Project Consortium. Grant Agreement for Collaborative Project "VPH-

Share", Annex I: Description of Work, 2nd revision. 2011..

5. VPH-Share Work Package 2. Deliverable D2.2: Design of the Cloud Platform. Internal Project Deliverable. ACC CYFRONET AGH; 2011.

6. VPH-Share Project Consortium. Deliverable D2.1: Analysis of the State of the Art and Work Package Definition. Internal Project Deliverable. ACC CYFRONET AGH; 2011.

7. OSGI Alliance. OSGi Alliance Homepage. [Online].; 2011. Available from: http://www.osgi.org/Main/HomePage.

8. Apache Karaf OSGi Container. [Online].; 2011. Available from: http://karaf.apache.org/.

9. Apache Camel Integration Framework. [Online].; 2011. Available from: http://camel.apache.org/.

10. JClouds. [Online].; 2011. Available from: http://code.google.com/p/jclouds/.

11. VPH-Share Project Consortium. Atmosphere Internal Registry. [Online].; 2013. Available from: http://vph.cyfronet.pl/air.

12. VPH-Share Project Consortium. Deliverable D2.3: First Prototype of the Cloud Platform..

13. Sloot PMA, Gubała T, Bubak M. Semantic Integration of Collaborative Research Environments. Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare. 2009: p. 514-530.

14. The Apache Software Foundation. The Apache Server Project. [Online].; 2013. Available from: http://httpd.apache.org/.

15. OpenStack Project Home. [Online]. Available from: http://openstack.org/.

16. Nginx Project Home. [Online]. Available from: http://wiki.nginx.org/Main.

17. IPv4 Exhaustion. [Online]. Available from: http://www.ripe.net/internet-coordination/ipv4-exhaustion.

18. Quartz Scheduler homepage. [Online].; 2012 [cited 2012 2 29. Available from:

http://quartz-scheduler.org/.

19. OASIS. eXtensible Access Control Markup Language. [Online].; 2013. Available from: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

20. Nginx HTTPS Module. [Online]. Available from: http://wiki.nginx.org/HttpSslModule.

21. Davidoff F, Godlee F, Hoey J, Glass R, Overbeke J, Utiger R, et al. Uniform requirements for manuscripts submitted to biomedical journals. JAOA: Journal of the American Osteopathic Association. 2003; 103(3).

22. Jetspeed 2 Web Portal. [Online].; 2012. Available from: http://portals.apache.org/jetspeed-2.

23. Alliance O. OSGI Web Portal. [Online]. Available from: http://www.osgi.org/Main/HomePage.

1. VPH Consortium. VPH2012 workshop, September 2012, London. [Online].; 2012. Available from: http://www.vph-noe.eu/vph-events/details/199-vph-share-workshop-in-vph2012.

2. Bubak M, Nowakowski P, Bartyński T, Gubała T, Haręźlak D, Kasztelnik M, et al. Cloud Platform for Medical Applications. In Proceedings of eScience 2012; 2012; Chicago.

3. Nowakowski P, Bartyński T, Gubała T, Haręźlak D, Kasztelnik M, Meizner J, et al. Managing Cloud Resources for Medical Applications. In Proceedings of the Cracow Grid Workshop 2012; 2012; Krakow.

4. VPH-Share Project Consortium. Grant Agreement for Collaborative Project "VPH-Share", Annex I: Description of Work, 2nd revision. 2011..

5. VPH-Share Work Package 2. Deliverable D2.2: Design of the Cloud Platform. Internal Project Deliverable. ACC CYFRONET AGH; 2011.

6. VPH-Share Project Consortium. Deliverable D2.1: Analysis of the State of the Art and Work Package Definition. Internal Project Deliverable. ACC CYFRONET AGH; 2011.

7. OSGI Alliance. OSGi Alliance Homepage. [Online].; 2011. Available from: http://www.osgi.org/Main/HomePage.

8. Apache Karaf OSGi Container. [Online].; 2011. Available from: http://karaf.apache.org/.

9. Apache Camel Integration Framework. [Online].; 2011. Available from: http://camel.apache.org/.

10. JClouds. [Online].; 2011. Available from: http://code.google.com/p/jclouds/.

11. VPH-Share Project Consortium. Atmosphere Internal Registry. [Online].; 2013. Available from: http://vph.cyfronet.pl/air.

12. VPH-Share Project Consortium. Deliverable D2.3: First Prototype of the Cloud Platform..

13. Sloot PMA, Gubała T, Bubak M. Semantic Integration of Collaborative Research Environments. Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare. 2009: p. 514-530.

14. The Apache Software Foundation. The Apache Server Project. [Online].; 2013. Available from: http://httpd.apache.org/.

15. OpenStack Project Home. [Online]. Available from: http://openstack.org/.

16. Nginx Project Home. [Online]. Available from: http://wiki.nginx.org/Main.

17. IPv4 Exhaustion. [Online]. Available from: http://www.ripe.net/internet-coordination/ipv4-exhaustion.

18. Quartz Scheduler homepage. [Online].; 2012 [cited 2012 2 29. Available from: http://quartz-scheduler.org/.

19. OASIS. eXtensible Access Control Markup Language. [Online].; 2013. Available from: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

20. Nginx HTTPS Module. [Online]. Available from: http://wiki.nginx.org/HttpSslModule.

21. Davidoff F, Godlee F, Hoey J, Glass R, Overbeke J, Utiger R, et al. Uniform requirements for manuscripts submitted to biomedical journals. JAOA: Journal of the American Osteopathic Association. 2003; 103(3).

22. Jetspeed 2 Web Portal. [Online].; 2012. Available from: http://portals.apache.org/jetspeed-2.

23. Alliance O. OSGI Web Portal. [Online]. Available from: http://www.osgi.org/Main/HomePage.

LIST OF KEY WORDS/ABBREVIATIONS

ACD	Audited Credential Delegation
ACL	Access Control List
AHE	Application Hosting Environment
AIR	Atmosphere Internal Registry
AMS	Atmosphere Management Service
AS	Atomic Service
AS	Authentication Service
ASI	Atomic Service Instance
API	Application Programming Interface
CC	Cloud Controller
CEE	Cloud Execution Environment
DNAT	Destination Network Address Translation
DRI	Data Reliability and Integrity
EC2	Amazon Elastic Computing Cloud v2
HPC	High Performance Computing
HTTP	HyperText Transfer Protocol
LDRI	Logical Data Resource Identifier
LOBCDER	Large OBject Cloud Data storagE fedeRation
MI	Master Interface
NAT	Network Address Translation
ORM	Object-Relational Mapping
OSGI	Open Service Gateway Initiative
REST	REpresentational State Transfer

SaaS	Software as a Service
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
SSH	Secure SHell
SSL	Secure Socket Layer
SSO	Single Sign-On
TLS	Transport-Layer Security
URI	Universal Resource Identifier
URL	Universal Resource Location
VM	Virtual Machine
VCN	Virtual Network Computing
WebDAV	Web-based Distributed Authoring and Versioning
WP	Work Package