**EGI-InSPIRE**

EGI Federated Cloud Blueprint v1

**EU MILESTONE: MS520**

|  |  |
| --- | --- |
| Document identifier: | Document1 |
| Date: | 00/00/0000 |
| Activity: | **SA2** |
| Lead Partner: | **OerC** |
| Document Status: | **DRAFT** |
| Dissemination Level: | **PUBLIC** |
| Document Link: | https://documents.egi.eu/document/1773 |

|  |
| --- |
| Abstract  << The abstract should provide a brief neutral overview of the document and its contents and main conclusions. Once complete the abstract should be copied into the abstract field on the document server.>>  <<document handling and production procedure is provided in <https://documents.egi.eu/document/33> >> |

1. Copyright notice

Copyright © Members of the EGI-InSPIRE Collaboration, 2010. See www.egi.eu for details of the EGI-InSPIRE project and the collaboration. EGI-InSPIRE (“European Grid Initiative: Integrated Sustainable Pan-European Infrastructure for Researchers in Europe”) is a project co-funded by the European Commission as an Integrated Infrastructure Initiative within the 7th Framework Programme. EGI-InSPIRE began in May 2010 and will run for 4 years. This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, and USA. The work must be attributed by attaching the following reference to the copied elements: “Copyright © Members of the EGI-InSPIRE Collaboration, 2010. See www.egi.eu for details of the EGI-InSPIRE project and the collaboration”. Using this document in a way and/or for purposes not foreseen in the license, requires the prior written permission of the copyright holders. The information contained in this document represents the views of the copyright holders as of the date such views are published.

1. Delivery Slip

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Name** | **Partner/Activity** | **Date** |
| **From** | <<The lead author/editor>> |  |  |
| **Reviewed by** | **Moderator:**  **Reviewers:**  <<To be completed by project office on submission to AMB/PMB>> |  |  |
| **Approved by** | **AMB & PMB**  <<To be completed by project office on submission to EC>> |  |  |

1. Document Log

|  |  |  |  |
| --- | --- | --- | --- |
| **Issue** | **Date** | **Comment** | **Author/Partner** |
| 1 |  | First draft |  |
| 2 |  |  |  |
| 3 |  |  |  |

1. Application area

This document is a formal deliverable for the European Commission, applicable to all members of the EGI-InSPIRE project, beneficiaries and Joint Research Unit members, as well as its collaborating projects.

1. Document amendment procedure

Amendments, comments and suggestions should be sent to the authors. The procedures documented in the EGI-InSPIRE “Document Management Procedure” will be followed:  
<https://wiki.egi.eu/wiki/Procedures>

1. Terminology

A complete project glossary is provided at the following page: <http://www.egi.eu/about/glossary/>.

<<The authors should check if the acronyms are covered by the glossary page and if the definition is still correct; all the amendments should be communicated to glossary@egi.eu>>

1. PROJECT SUMMARY

To support science and innovation, a lasting operational model for e-Science is needed − both for coordinating the infrastructure and for delivering integrated services that cross national borders.

The EGI-InSPIRE project will support the transition from a project-based system to a sustainable pan-European e-Infrastructure, by supporting ‘grids’ of high-performance computing (HPC) and high-throughput computing (HTC) resources. EGI-InSPIRE will also be ideally placed to integrate new Distributed Computing Infrastructures (DCIs) such as clouds, supercomputing networks and desktop grids, to benefit user communities within the European Research Area.

EGI-InSPIRE will collect user requirements and provide support for the current and potential new user communities, for example within the ESFRI projects. Additional support will also be given to the current heavy users of the infrastructure, such as high energy physics, computational chemistry and life sciences, as they move their critical services and tools from a centralised support model to one driven by their own individual communities.

The objectives of the project are:

1. The continued operation and expansion of today’s production infrastructure by transitioning to a governance model and operational infrastructure that can be increasingly sustained outside of specific project funding.
2. The continued support of researchers within Europe and their international collaborators that are using the current production infrastructure.
3. The support for current heavy users of the infrastructure in earth science, astronomy and astrophysics, fusion, computational chemistry and materials science technology, life sciences and high energy physics as they move to sustainable support models for their own communities.
4. Interfaces that expand access to new user communities including new potential heavy users of the infrastructure from the ESFRI projects.
5. Mechanisms to integrate existing infrastructure providers in Europe and around the world into the production infrastructure, so as to provide transparent access to all authorised users.
6. Establish processes and procedures to allow the integration of new DCI technologies (e.g. clouds, volunteer desktop grids) and heterogeneous resources (e.g. HTC and HPC) into a seamless production infrastructure as they mature and demonstrate value to the EGI community.

The EGI community is a federation of independent national and community resource providers, whose resources support specific research communities and international collaborators both within Europe and worldwide. EGI.eu, coordinator of EGI-InSPIRE, brings together partner institutions established within the community to provide a set of essential human and technical services that enable secure integrated access to distributed resources on behalf of the community.

The production infrastructure supports Virtual Research Communities (VRCs) − structured international user communities − that are grouped into specific research domains. VRCs are formally represented within EGI at both a technical and strategic level.

1. EXECUTIVE SUMMARY

<< The text should provide a summary of the full report so that the reader can ‘in a page’ understand the problem it has been written to cover. This includes an overview of the background material and motivation for the report, a summary of the analysis, and the report’s main conclusions.>>

TABLE OF CONTENTS

1 Introduction 4

2 XXXX 4

2.1 YYYYYYYYYY 4

2.1.1 ZZZZZZZZ 4

3 Conclusion 4

4 References 4

# Introduction

The current high throughput model of grid computing has proven extremely powerful for a number of different communities, though it has also limited in a number of cases the uptake of e-infrastructure. EGI therefore chose to investigate how to broaden the communities and application design models that would be able to take advantage of EGI core infrastructure and the level of dependency its production infrastructure is able to provide.

It was clear that the utilisation of virtualization and Infrastructure as a Service (IaaS) cloud computing was a clear candidate to enable this transformation though with a number of different open source providers already in use across a number of different resource providers it was clear that the previous approach of a mandated single software stack was not going to be possible. Instead following on from a number of different activities already on-going in the EC including SIENA, an approach that required the utilisation where possible of open standards where available and where not methods that have broad acceptance in the e-infrastructure community were essential.

The Task Force as originally configured has an 18-month mandate from September 2011, which is subdivided into 3 succinct six-month blocks;

1) Setup - Identify resource and technology providers and draft the model

2) Consolidation - Engage exemplar user communities and start configuration of test-bed

3) Integration - Establish test-bed fully with early adopter user communities and document

Overall goals for the activity were:

* Write a blueprint document[[1]](#footnote-1) for EGI Resource Providers that wish to securely federate[[2]](#footnote-2) and share their virtualised environments as part of the EGI production infrastructure;
* Deploy a test bed[[3]](#footnote-3) to evaluate the integration of virtualised resources within the existing EGI production infrastructure for monitoring[[4]](#footnote-4), accounting[[5]](#footnote-5) and information services[[6]](#footnote-6);
* Investigate and catalogue the requirements[[7]](#footnote-7) for community facing services based on or deployed through virtualised resources;
* Provide feedback[[8]](#footnote-8) to relevant technology providers on their implementations and any changes needed for deployment into the production infrastructure;
* Identify and work with user communities[[9]](#footnote-9) willing to be early adopters of the test bed infrastructure to help prioritise its future development;
* Identify issues[[10]](#footnote-10) that need to be addressed by other areas of EGI (e.g. policy, operations, support & dissemination).

# Federation model

The federation of IaaS Cloud resources in EGI is built upon extensive autonomy of Resource Providers in terms of ownership of exposed resources. The current federation model in EGI for exposing Grid resources requires Resource Providers to deploy and operate a specific set of Grid Middleware components before they were integrated into the EGI federated Grid production infrastructure. In contrast to that, the federation model for distributed IaaS Cloud resources allows a lightweight aggregation of local Cloud resources into the EGI Cloud Infrastructure Platform. At the heart of the federation are thus locally deployed Cloud Management stacks. In compliance with the Cloud computing model, the EGI CLIP does not mandate deploying any particular or specific Cloud Management stack; it is the responsibility of the Resource Providers to research, identify and deploy the solution that fits best their individual needs for as long as the offered services implement the required interfaces and domain languages. These interfaces and domain languages, and the interoperability of their implementation with other solutions are the focus of the federation.

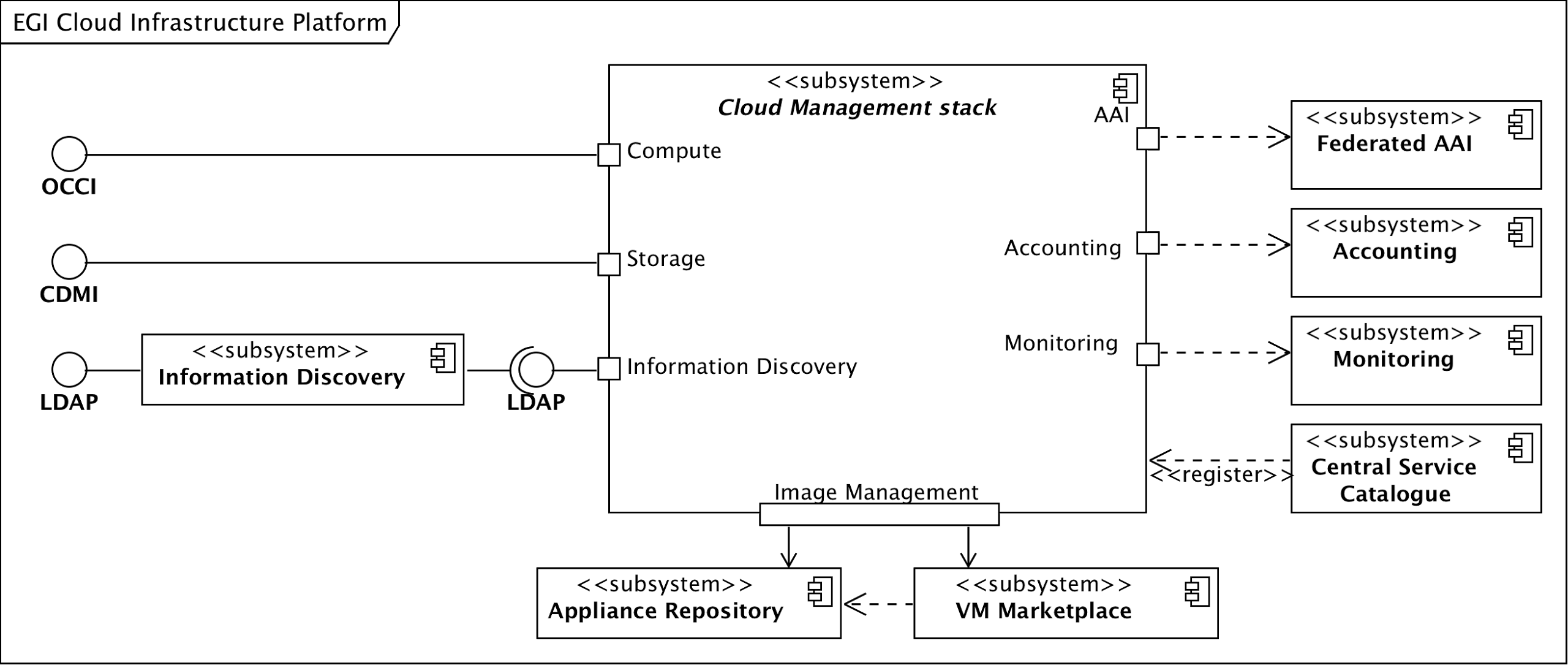
Consequently, the EGI Cloud Infrastructure Platform is modeled around the concept of an *abstract* Cloud Management stack subsystem that is integrated with components of the EGI Core Infrastructure Platform (see Figure 1).

Figure 1: Architecture of the EGI Cloud Infrastructure Platform

This architecture allows EGI to define the CLIP as a relatively thin layer of federation and interoperability around local deployments and integrations of Cloud Management stacks.

This architecture defines interaction ports with a number of services from the EGI Core Infrastructure Platform, and the EGI Collaboration Platform. At the same time, it defines the required external interfaces and corresponding interaction ports. All these ports will have to be realised by local Cloud Management stack deployments.

The main interaction points of Resource Providers must take care of thus are:

* Integrate with the EGI Core AAI
* Integrate with the EGI Core Accounting system
* Integrate with the EGI Core Monitoring system
* Provide a standardised Cloud Computing interface (OCCI)
* Provide a standardised Cloud Storage interface (CDMI)
* Provide a standardised interface to an Information Service

Additionally, by means of using the Appliance Repository and the VM Marketplace from the EGI Collaboration Platform EGI Cloud Infrastructure Platform is providing VM image sharing and re-use across EGI Research Communities.

At the time of writing the Cloud federation platform is maintaining its own, separate Information discovery system. Even though it is using the GLUE2 schema, some extensions and tweaks are not compatible with the canonical GLUE 2 specification. Hence Cloud Resource providers maintain local LDAP endpoints (usually deployed as a resource BDII) aggregated into a Cloud Platform Information Discovery service, which in turn allows access to the data using LDAP v3.

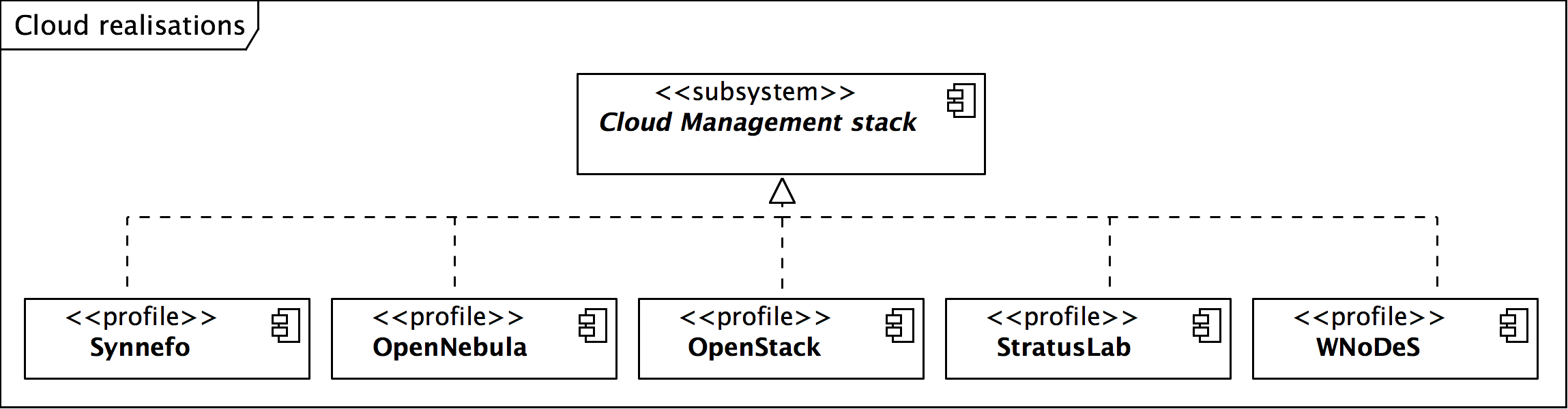
Figure 2 provides an overview of the current realisations of the abstract Cloud Management stack subsystem in the EGI Cloud federation. It illustrates that each existing realisation inherits the obligation to implement the interaction points from the generalised parent Cloud Management stack. At the same time, the EGI Federated Clouds Task (funded through the EGI-InSPIRE project) gives Resource Providers a platform to share their implementation solutions for a commonly deployed specific Cloud Management stack (e.g. OpenNebula and OpenStack). Section 5 is dedicated to the documentation of the steps necessary to integrate a local deployment of a given Cloud Management stack into the EGI Cloud federation.

Figure 2: Current realisations of the abstract Cloud Management stack component

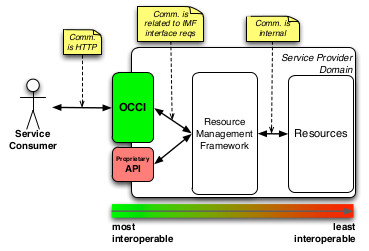
Through this collaboration, Resource Providers gradually develop and mature deployment and configuration profiles around common Cloud Management stacks as illustrated in Figure 2. This blueprint document is the first iteration of the resource provider oriented documentation.

# Management Interfaces

To federate a cloud system there are several functions for which we must have a common interface defined. These are each described below and in all define the method by which a ‘user’ of the service would be able to interact.

## VM management interface: OCCI

**The Open Cloud Computing Interface** (OCCI) is a RESTful Protocol and API designed to facilitate interoperable access to cloud-based resources across multiple resource providers and heterogeneous environments. The formal specification is maintained and still being extended by OGF’s OCCI-WG, for details see <http://occi-wg.org/>. Intended deployment is depicted by Figure 3.

Figure 3Deployment of OCCI in a provider's infrastructure  


OCCI’s specification consists of three basic elements, each covered in a separate specification document:

**OCCI Core** describes the formal definition of the the OCCI Core Model [1]. **OCCI HTTP Rendering** defines how to interact with the OCCI Core Model using the RESTful OCCI API [2]. The document denes how the OCCI Core Model can be communicated and thus serialised using the HTTP protocol. **OCCI Infrastructure** contains the definition of the OCCI Infrastructure extension for the IaaS domain [3]. The document defines additional resource types, their attributes and the actions that can be taken on each resource type. Detailed description of the abovementioned elements of the specification is outside the scope of this document. Simplified description is as follows.

OCCI Core defines base types **Resource**, **Link**, **Action** and **Mixin**. Resource represents all OCCI objects that can be manipulated and used in any conceivable way. In general, it represents provider’s resources such as images (Storage Resource), networks (Network Resource), virtual machines (Compute Resource) or available services. Link represents a base association between two Resource instances, it indicates a generic connection between a *source* and a *target*. The most common real-world examples are Network Interface and Storage Link connecting Storage and Network Resource to a Compute Resource. Action defines an invocable operation tied to a specific Resource instance or a collection of Resource instances. In general, Action is designed to perform complex high-level operations changing the state of the chosen Resource such as virtual machine reboot or migration. The concept of mixins is used to facilitate extensibility and provide a way to define provider-specific features.

In the Federated Cloud environment, OCCI is deployed as a variety of platform-specific implementations. An ongoing EGI-InSPIRE mini-project *TSA 4.4 Providing OCCI support for arbitrary Cloud Management Frameworks* aims to provide a common implementation to further improve interoperability.

[1] R. Nyren, A. Edmonds, A. Papaspyrou, and T. Metsch, “Open Cloud Computing Interface - Core," GFD-P-R.183, April 2011. [Online]. Available: http://ogf.org/documents/GFD.183.pdf

[2] T. Metsch and A. Edmonds, “Open Cloud Computing Interface - HTTP Rendering," GFD-P-R.185, April 2011. [Online]. Available: http://ogf.org/documents/GFD.185.pdf

[3] “Open Cloud Computing Interface - Infrastructure," GFD-P-R.184, April 2011. [Online]. Available: http://ogf.org/documents/GFD.184.pdf

## Data management interface: CDMI (I Livenson)

SNIA Cloud Data Management Interface (CDMI) defines an open standard for operations on storage objects in a RESTful way. Semantically the interface is very close to AWS S3 and MS Azure Blob, but is more open and flexible for implementation.

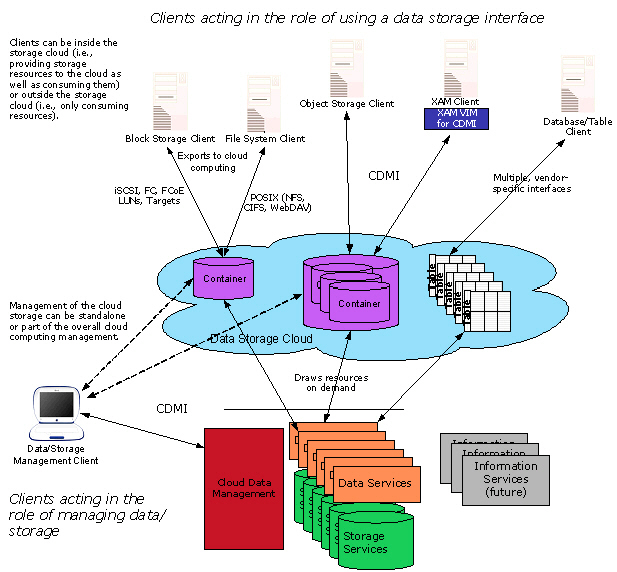
Figure 4: Cloud storage reference model (courtesy to SNIAcloud.com)

Figure 4 shows the conceptual model of a cloud storage. CDMI offers clients a way for operating both on a storage management system and single data items. The exact level of support depends on the concrete implementation and is exposed to the client as part of the protocol.

The design of the protocol is aiming both at flexibility and efficiency.  Certain heavyweight operations, e.g. blob download, can be performed also with a pure HTTP client to make use of the existing ecosystem of tools. CDMI is built around the concept of Objects, which vary in supported operations and metadata schema. Each Object has an ID, which is unique across all CDMI deployments.

### CDMI Objects

There are 4 objects most relevant in the context of EGI Federated cloud TF:

* **Data object**: Abstraction for a file with rich metadata.
* **Container: A**bstraction for a folder. Export to non-HTTP protocols is performed on the container level. Container might have other containers inside of them.
* **Capability:** Exposes information about a feature set of a certain object.
* **Domain:** Deployment specific information.

### Detection of capabilities

CDMI supports partial implementation of the standards by defining optional features and parameters. In order to discover what functionality is supported by a specific implementation, CDMI client can issue a GET request to a fixed url: /cdmi\_capabilities.

### Export protocol

Attachment of the storage items to a VM can often be performed more efficiently using protocols like NFS or iSCSI. CDMI supports exposing of this information via container metadata. A client can make use of this information to attach a storage item to a VM over an OCCI protocol.

More information about the CDMI standard can be found at <http://cdmi.sniacloud.com/>. An ongoing EGI-InSPIRE mini-project *TSA4.5 CDMI Support in Cloud Management Frameworks* aims to provide an implementation of CDMI, which integrates with OCCI-based infrastructure and supports Fedcloud use-cases.

## Virtual Organisation Management & AAI: VOMS

Within the task force, it has been decided to generally allow VOMS proxy certificates of VOs ‘fedcloud.egi.eu’ and ‘ops’, the latter mainly for Nagios monitoring. There are modules available for each cloud middleware that have been developed by TF members. Configuring these modules into a provider’s cloud installation will allow members of these VOs to access the cloud.

Figure 5 shows the main components involved. The user retrieves a VOMS attribute certificate from the VOMS server of the desired VO and thus creates a local VOMS proxy certificate. The VOMS proxy certificate is use in subsequent calls to the OCCI endpoints of OpenNebula or OpenStack using the rOCCI client tool.

The rOCCI client directly talks to OpenNebula endpoints, which map the certificate and VO information to local users. Local users need to have been created in advance, which is triggered by regular synchronizations of the OpenNebula installation with Perun. Perun manages the VOs that participate in the FCTF testbed.

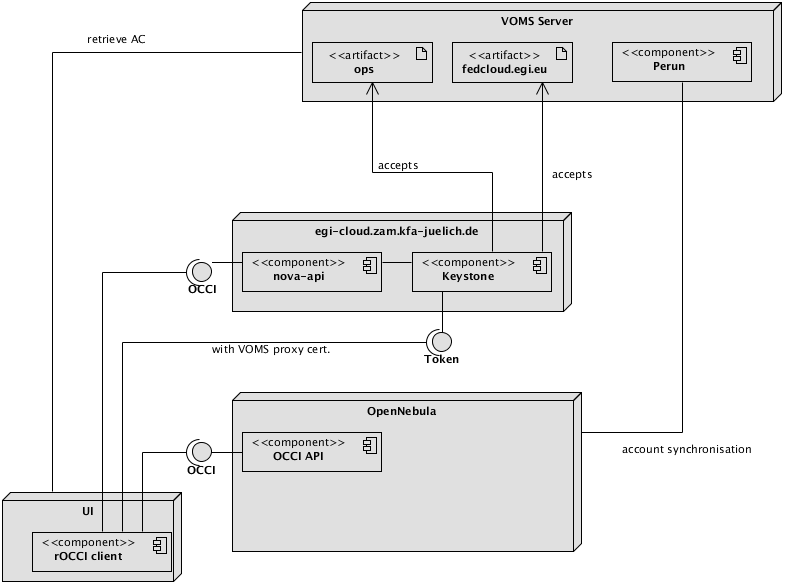
In order to access an OpenStack OCCI endpoint, rOCCI client needs to retrieve a Keystone token from OpenStack Keystone first. The retrieval is transparent to the user and automated in the workflow of accessing the OpenStack OCCI endpoint. It is triggered by the OCCI endpoint rejecting invalid requests and sending back an HTTP header referencing the Keystone URL for authentication. Users are generated on-the-fly in Keystone, it does not need regular synchronization with Perun.

Figure 5: Model of the Federated Cloud authentication architecture

Generic information about how to configure VOMS support for OpenStack Keystone can be found at <http://keystone-voms.readthedocs.org/en/latest/>. Information specific to FCTF is located at <https://wiki.egi.eu/wiki/Federated_AAI_Configuration#OpenStack>.

For OpenNebula, the information can be found here: <https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:_Federated_AAI:OpenNebula>.

Stratus lab provides multiple authentication mechanisms at once. They are documented here: <http://stratuslab.eu//documentation/2012/10/07/docs-syadmin-auth.html>.

## Image management interface: vmcatcher/caster

<< Title shodul be broken down to design, specification, and implementation sections hopefully the main editors can do this (Owen)>>

### Image distribution Design

Images are referenced by an Virtual Machine Image List which contains a secure hash (sha512) signed using x.509 technology. These Virtual Machine Image Lists are published, and interested sites subscribe to the Lists in a catalogue at the site. When an instantiation request for an image is received, the Image validity is checked. If the Virtual Machine Image List is valid, the Image is contextualised and then instantiated. Images that do not pass validation are not instantiated.

**Description**

The idea is that a list of virtual images and the associated metadata is signed with a x.509 certificates.

This provides a mechanism by which a virtual image can be checked for validity: expiry, revocation, whether it has been tampered with etc, and traceability. The image is then distributed to a site using (for example) the software distribution mechanism by the responsible Virtual Organisation, or automatically by the local catalogue. Information about the image (the certificate public keys, Virtual Organisation responsible, a unique ID, a Virtual Organisation tag family and a URI, inital global validity, i.e., the metadata) is registered with a local catalogue. A local validity flag may also be set by the site.

Requests for instantiation are made via the unique ID or the tag. When the local system receives an

instantiation request for a particular image, the catalogue is checked for the location and global and local validity flags. If both are OK, the image is then checked for expiry and revocation using the x.509 credentials. If the chek is passed then the image is contextualised and instantiated. Expired or or revoked images are refused.

**VM Images List signing**

An Image will be created by a Creator and added to an Virtual Machine Image List which is then signed by the Creator’s x.509 certificate. The Virtual Machine Image List will have an appropriate expiry for the Image in the Virtual Machine Image List. The Image may then be distributed by an appropriate mechanism. Images may be revoked by not including the Image in the latest version of the Virtual Machine Image List. The URI to establish the latest version of the Virtual Machine Image List will be contained in the Virtual Machine Image List metadata.

**VM Image distribution**

The Virtual Machine Image List is provided on a host that can be authenticated to prevent

man-in-the-middle attacks blocking update notices. An image may be distributed to Sites by any suitable means including the existing software distribution mechanism, via a URI reference in the metadata. The transfer can be validated using a secure hash which is expected to be sha512.

While any check on the authenticity of the host, would safely prevent man in the middle attacks it may be easier to use x.509 authentication for this function.

**Local catalogue**

A Site will have a Local Catalogue, which will be a registry (database) of Images which may be considered for running at the Site. The Catalogue will contain metadata for each subscribed Image and Virtual Machine Image List. This metadata will at least contain the x.509 certification data, the unique RFC 4122 UUID, a Tag, resource locator, a global validity flag field and a local validity flag field.

The Local Catalogue will subscribe to the appropriate revocation service (CRLs) and flag Images that

have been revoked by setting the global validity flag.

A local validity flag may be set against an Image, to allow local administrations to "blacklist" an Image. The local validity flag may also be set to "run anyway" to allow expired but NOT revoked images to be run.

**Registration/Subscription**

A distributed Virtual Machine Image List will be subscribed into the local Catalogue by the distribution mechanism. The subscriber will write the Virtual Machine Image List metadata to the Catalogue and set the global validity flag by checking the latest CRLs. Images may be subscribed locally.

**Instantiation**

A request to the instantiation system to run an Image will cause the instantiation system to validate the Image against its global and local validity flags and metadata in the Local Catalogue, and to check for the expiry of the Image using its x.509 certificate. If an image passes all the required checks, the Image may be contextualised and then run.

**Image Metadata**

Images will have x.509 signed metadata. Images will have a unique identifier (RFC 4122 UUID). Images will have a Tag name which may be non-unique such that a series of images may belong to a tag family. Requests to run an image may be either via the RFC 4122 UUID or the Tag. In the case of a Tag having more than one RFC 4122 UUID registered the latest dated image will be considered for instantiation.

### Image transfer Specifications

The workflow is based on the concept of Virtual Machine Image List which have a lifecycle. All Virtual Machine Image List have a URI and a RFC 4122 UUID, a creation and expiry date and Version number. If Virtual Machine Image List change either RFC 4122 UUID or URI they are considered different Virtual Machine Image List, and when a Virtual Machine Image List is updated it is expected that the creation data is updated and best to update the version number. It is recommended that the expiry data of an Virtual Machine Image List is 1 month from the creation date.

Endorsers provide signed Virtual Machine Image List, and accept the security responsibilities of producing images as stated in the Image trust section of this document. Subscribers use Virtual Machine Image List and run them confident they can identify the endorser, and decide if they will run the endorsers images. Virtual Machine Image List can be valid or invalid from the perspective of the subscriber. Valid Virtual Machine Image List are signed, with a valid certificate from a CA trusted by the subscriber, the current time is between the created and expiry date, and endorsers subject in the Virtual Machine Image List that matches the endorser, and the current version number is greater or equal to the previous version number. If Virtual Machine Image List do not match any of these conditions the Virtual Machine Image List is invalid. Subscribed Virtual Machine Image List are referenced by their RFC 4122 UUID.

The workflow also has the concept of image life cycles, which are related to image list life cycles. An image maybe available or unavailable, created, and updated. From the subscribers perspective available images cannot exist without a valid Virtual Machine Image List that includes them. Images are secured with a sha512 value in the Virtual Machine Image List that matches the image. Its RFC 4122 UUID and its version number identify an image. Its current state is defined by its presence or absence in a valid Virtual Machine Image List. Valid images always have greater or equal version numbers than previous versions.

RFC 4122 UUID’s of Virtual Machine Image List cannot overlap, and in the same way the RFC 4122 UUID’s of images cannot overlap. It is considered a hostile action if RFC 4122 UUID uniqueness is violated by an endorser. Since Virtual Machine Image List RFC 4122 UUID’s are only added or removed by the subscriber displaying errors and rejecting Virtual Machine Image List with clashing RFC 4122 UUID’s with the subscribers tools is enough to alert the subscriber to such endorser actions. If two Virtual Machine Image List provide images with the same RFC 4122 UUID, the image from the second Virtual Machine Image List will be ignored and an error should be shown by the subscriber’s tools.

Updated Virtual Machine Image List may contain new images referenced by new RFC 4122 UUID’s. These new images will be ignored and WARNINGS shown until the Virtual Machine Image List is re-subscribed. This behavior keeps the subscriber in control of the Image RFC 4122 UUID’s and their association to Virtual Machine Image List RFC 4122 UUID’s. It is always well defined which image belongs to which Virtual Machine Image List and so the subscriber always knows the publisher of an image.

Images RFC 4122 UUID’s no longer present in an Virtual Machine Image List are no longer endorsed by the endorser and so the images are no longer available, and consequently such images are marked as "expired" an event is sent. The subscriber is aware of the complete history of an image through the metadata of that image within the subscription tools, the date of expiry and renewal for each state change for an image in an Virtual Machine Image List.

### Implementations of image list based image transfer.

The above documentation explains how image transfers can work. We have 3 image list publishers and one image list subscriber for use by the EGI Federated Cloud.

**vmcatcher**

The software makes use of a database to store subscriptions to virtual machine images lists and download selected images from the list. It works in a similar way to a podcast reader or a Linux package manager. The tested Database is SqlLight, but it is based upon SQLalchamy so should support many databases. SqlLight has proved more than adequate for the low transaction rate of a image list subscriber and so deployment issues are mealy backing up a database file.

Since the software is made with the Grid in mind it is only natural that the X509 certificate model is used.

1.2.1.1 Introduction

This application allows users to subscribe to virtual machine Virtual Machine Image List{.literal}s, cache the images referenced to in the Virtual Machine Image List{.literal}, validate the images list with x509 based public key cryptography, and validate the images against sha512 hashes in the images lists and provide events for further applications to process updates or expires of virtual machine images without having to further validate the images.

This software is available at:

https://github.com/hepix-virtualisation/vmcatcher{.code}

The software is based upon a simple database that stores subscriptions to Virtual Machine Image List{.literal}s, who can sign theVirtual Machine Image List{.literal}, and which images belong to which subscriptions. It allows images to selected for subscription.

Subscribed images can be downloaded verified and cached. Cached images can be verified, and if invalid or expired they are moved to an expiry directory.

1.2.1.2 Features

• Add and delete multiple subscriptions to Virtual Machine Image List{.literal}s.

• Update subscriptions checking authenticity of the message using x509 based signatures.

• Automation as a cron script.

• Subscribe and unsubscribe to images from Virtual Machine Image List{.literal}s.

• Download verify images into a local cache.

• Expire images to an archive when no longer endorsed or corrupt.

This set of applications are designed to provide a similar work flow from each area of control to theVirtual Machine Image List{.literal} archive.

• vmcatcher\_endorser - Endorsers of Virtual Machine Image List{.literal} subscriptions.

• vmcatcher\_subscribe - Subscription list details.

• vmcatcher\_image - Image details.

• vmcatcher\_cache - Cache images and update events.

They work in conjunction with a database to ease navigation, a local cache of Virtual Machine Image List{.literal} subscriptions. The database is message format agnostic, but it authenticates and validates all messages in import. Because these are just caches ofVirtual Machine Image List{.literal}s they are meant to be used the majority of the time without intervention.

If you are signing a list using the HEPiX Virtual Machine Image List{.literal} signer, you should also install this application and subscribe to your current image.

It is intended to with a couple of cron scripts to be informed at any time if your local images are matching signatures in theVirtual Machine Image List{.literal}.

Anyone curious about this application should consider this application a software application similar to Debian’s ’aptitude’ or Redhats ’yum’, but rather for virtual machines, authenticated by the x509 signatures.

1.2.1.3 vmcatcher\_cache Event interface

Since this application suite is intended to be embedded in a larger application and concerned with downloading and managing updates of VM images into a cloud infrastructure, it is some times beneficial to have an event interface so that applications may embed these applications in larger systems ratehr than polling the local cache.

[user] $ vmcatcher\_cache -x "/usr/bin/VmImageUpdateProcessor \$VMCATCHER\_EVENT\_TYPE"

The events interface launches a shell with a series of environment variables. The event must process its command within 10 seconds or else it will be sent a termination signal. See the following example:

[user] $ vmcatcher\_cache -x 'env ; exit 1'

All Events have a type. This is given to the event handler by setting the variable, VMCATCHER\_EVENT\_TYPE with the following values "AvailablePrefix", "AvailablePostfix", "ExpirePrefix" and "ExpirePosfix".

"Available" events happen when a new image is validated, while "Expire" events occur when an image i no longer the validated image. The "Prefix" events occur before the file changes state, and the "Posfix" events occur after the state change.

The following environment variables may be set by events:

• VMCATCHER\_EVENT\_TYPE

• VMCATCHER\_EVENT\_DC\_DESCRIPTION

• VMCATCHER\_EVENT\_DC\_IDENTIFIER

• VMCATCHER\_EVENT\_DC\_TITLE

• VMCATCHER\_EVENT\_HV\_HYPERVISOR

• VMCATCHER\_EVENT\_HV\_SIZE

• VMCATCHER\_EVENT\_HV\_URI

• VMCATCHER\_EVENT\_SL\_ARCH

• VMCATCHER\_EVENT\_SL\_CHECKSUM\_SHA512

• VMCATCHER\_EVENT\_SL\_COMMENTS

• VMCATCHER\_EVENT\_SL\_OS

• VMCATCHER\_EVENT\_SL\_OSVERSION

• VMCATCHER\_EVENT\_TYPE

• VMCATCHER\_EVENT\_FILENAME

These correspond to the variables within the Virtual Machine Image List{.literal}.

1.2.2 vmcaster

Vmcaster is a simple tool for managing and updating your published virtual machines image lists on th elinux command line using a database to store your old image list and minimise the effort required to update image lists. Following the Hepix image list format.

vmcaster software is available here:

https://github.com/hepix-virtualisation/vmcaster

Users of this application should read "security-related policy requirements for the generation and endorsement of trusted virtual machine (VM) images for use on the Grid". The URL is shown below.

https://edms.cern.ch/document/1080777

This tool is designed to aid "endorsers" as defined in the security policy, and anyone who wants to publish virtual machine images in a way where tampering will be detected.

vmcaster was designed with the realisation that users typically create new virtual machines images rarely but update them frequently. Vmcaster attempts to be the first of a new generation of image list publishers the minimise the data entry for updating images, with imagelists.

The tasks of updating an image and uploading a fresh signed image is now just two quiet short command line operations away. The aim is to make image and imagelist updates as painless as possible as these are the most common tasks.

Internally the application uses a simple sql database, sqlight for storing image lists and multiple back ends managing uploading of images / images lists using a facade pattern. This allows transfer protocol to be derived from the meta data of the image list and a configuration file, updates need much less effort.

1.2.3 hepixvmitrust

1.2.3.1 Introduction

hepixvmitrust is a package that contains a CLI tool, and a minimal implementation, in its documentation for X509 signing lists of virtual machine image metadata. The tools are generally reusable but were developed to satisfy the need to securely exchange virtual machine images between High Energy Physics sites, in a similar way to yum and apt repositories provide for rpms, this software provides for Virtual Machines.

This software is available here:

https://github.com/hepix-virtualisation/hepixvmitrust

Its use has largely been superseded by vmcaster which requires less effort to provide endorsers updating images.

1.2.4 minimal code example for generating image lists.

This is an alternative implantation of an image list signer in python, using the m2crypto wrapper for openssl as does the CLI.

This version is very compact and simplistic. It is intended to be used as part of a release system which has all the variables needed to produce an image list within the release system already, for products like repoman, and the CernVM release system, to see an example of signing their own image lists, using minimal code.

https://github.com/hepix-virtualisation/hepixvmitrust/blob/master/minimal.py

# EGI Core Services for Cloud

## Information discovery: BDII

Users and service managers need tools to retrieve information about the whole infrastructure and filter the huge amount of data to retrieve the subset of the infrastructure that fulfill their requirements. To achieve this target the information about the services in the infrastructure must be structured in a uniform schema and published by a common set of services usable both by automatic tools and human users.

The current standard deployed in EGI for the implementation of the common information system is the Berkeley Database Information Index (BDII). It is a software based on a LDAP server, and it is deployed in an hierarchical structure, distributed over the whole infrastructure.

The information system is structure in three levels: the grid or cloud services publish their information (e.g. specific capabilities, total and available capacity or VO supported by the service) using a standard Open Grid Forum format, the GLUE schema2. The information published by the services are collected by a Site-BDII, a service deployed in almost every site in EGI. The Site-BDIIs are queried by the Top-BDIIs - the higher level of the hierarchy- which contain the information of all the sites in the infrastructure and their services. NGIs usually provide an authoritative instance of Top-BDII, but every Top-BDII, if properly configured, should contain the same set of information. While Site-BDII is configured by site managers to pull the data from the resource information providers of the services, Top-BDII automatically retrieves the list of the production sites from GOCDB and pulls the information form their Site-BDIIs.

Users and tools can use the Top-BDII to look for the services that provide the capabilities and the resources to run their activities. A typical example of Top-BDII query is retrieving the list of services that support a specific VO.

### Technical implementation of the federated cloud information system

Currently the Federated Cloud information system is built starting from the resource provider level. Every resource provider is requested to deploy a LDAP server publishing the information about their services structured used the GLUE schema. The best technical choice is to go for OpenLDAP, which is available in almost all the \*nix machines in the world. On top of that, OpenLDAP is the server used by the gLite BDIIs, therefore it would be easy to use the same configuration files set-up used for the GRIS (Grid Resource Information Service) or the GIIS (Grid Index Information Service).

Cloud services are not yet implementing information providers, therefore the information are published directly by a site-level information provider, comparable to a site-bdii in the structure of the information published. This solution is considered acceptable as the number of cloud services deployed by a single resource provider are usually not as many as the services in grid sites.

The LDIF file to be loaded in the local LDAP server is generated by a prototype custom script, and the information published are the following:

* Cloud computing resources
* Service endpoint
* Capabilities provided by the service, such as: virtual machine management or snapshot taking. The labels that identify the capabilities are agreed within the taskforce.
* Interface, the type of interface – e.g. webservice or webportal – and the interface name and version, for example OCCI 1.2.0
* User authentication and authorization profiles supported by the service, e.g. X.509 certificates
* Virtual machines images made available by the cloud provider
* Operating system, and other environment configuration details
* Maximum number of cores – and physical memory – allocable in a single virtual machine

Clearly, multiple virtual machine types can be associated to a single cloud service. There are no limitations in the number of services published by a resource centre either.Currently the information published is modelled using the latest GLUE2.0 schema definition. An extension of the GLUE schema is under development to address the specific requirements of Cloud resources and add information about storage and network services.

The Federated Cloud taskforce deploys a central Top-BDII that pulls automatically the information from the local LDAP servers of the resource providers. This service can be used as a single entry point to query for all the resource centres supported by the test-bed, by users or other automatic tools.

Currently the central federated cloud Top-BDII is reachable at this address: [ldap://test03.egi.cesga.es:2170](file://localhost/ldap/::test03.egi.cesga.es/2170)

An example of a possible query is:

ldapsearch -x -H ldap://test03.egi.cesga.es:2170 -b o=glue '(objectClass=GLUE2Endpoint)' | perl -p00e 's/\r?\n //g' | grep -E 'GLUE2EndpointURL|GLUE2EndpointInterfaceName|GLUE2EndpointInterfaceVersion|dn\:' | awk '{printf("%s%s", $0, (NR%4 ? " === " : "\n"))}' | awk '{print ""$2" "$5" "$8" "$11}' | awk -F "GLUE2DomainID=" '{print $2}' | awk -F "," '{print $1 " "$3}' | awk '{print $1" "$4" "$3" "$5}' | sort

This query provides the interface names exposed by all the resource providers, the version of the implemented interface, and the endpoint that can be used to contact the service through the specific interface.

From a technical standpoint the resource centre LDAP server must answer to the port 2170, in order to be automatically polled by the Top-BDII

## Central service registry: GOCDB

EGI’s central service catalogue is used to manage the production infrastructure topology. The service is provided using the GOCDB tool that is developed and deployed within EGI. To allow Resource Providers to expose Cloud resources to the production infrastructure, a number of new service types were added to GODCB:

* eu.egi.cloud.accounting
* eu.egi.cloud.information.bdii
* eu.egi.cloud.storage-management.cdmi
* eu.egi.cloud.vm-management.occi
* eu.egi.cloud.vm-metadata.marketplace

For the time-being (i.e. until EGI is integrating federated Cloud resources into production), all registered Cloud resources are maintained in  test-bed mode.

## Monitoring: SAM

The SAM (Service Availability Monitoring) system is a framework consisting of:

Nagios monitoring system (https://[www.nagios.org](http://www.nagios.org/)),

custom databases for topology, probes description and storing results of tests

web interface MyWLCG/MyEGI (<https://tomtools.cern.ch/confluence/display/SAM/MyWLCG>)

Probes used to perform check of services are provided by service developers. SAM team only maintains probes which test internal SAM functions (e.g. communication with messaging system, database synchronization, etc). More information on SAM can be found [here](https://wiki.egi.eu/wiki/SAM).

Current set of probes used for monitoring cloud resources consists of:

OCCI probe - creates an instance of a given image by using OCCI and checks its status

BDII probe - basic LDAP check tries to connects to cloud BDII

Accounting probe - checks if the cloud resource is publishing data to Accounting repository

TCP checks - basic TCP checks used for CDMI services.

Central SAM instance is deployed for monitoring cloud resources (<https://cloudmon.egi.eu/nagios>). Once the set of probes is finalized probes will be included into official SAM release. Adding probes to official SAM will follow procedure “Adding new probes to SAM” (<https://wiki.egi.eu/wiki/PROC07>).

## Accounting: (A. Packer)

<<Describe the key components of the accounting subsystem, with a focus on those components that need to be installed at the Resource Provider’s infrastructure. Link through to other docs where possible for other APEL components>>

To account for resource usage across the resource providers the following have been defined:

* The elements to be accounted for;
* Mechanisms for gathering and publishing accounting data to a central accounting repository;
* How accounting data will be displayed by the EGI Accounting Portal.

A Cloud Accounting Usage Record defines the data elements which resource providers should send to the central Cloud Accounting repository.  These elements are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Value** | **Description** | **Mandatory** |
| VMUUID | string | Virtual Machine's Universally Unique IDentifier | Yes |
| SiteName | string | Sitename, e.g. GOCDB Sitename | Yes |
| MachineName | string | VM Id |  |
| LocalUserId | string | Local username |  |
| LocalGroupId | string | Local groupname |  |
| GlobalUserName | string | User's X509 DN |  |
| FQAN | string | User's VOMS attributes |  |
| Status | string | Completion status - started, completed, suspended |  |
| StartTime | int | Must be set if Status = Started (epoch time) |  |
| EndTime | int | Must be set if Status = completed (epoch time) |  |
| SuspendDuration | int | Set when Status = suspended (seconds) |  |
| WallDuration | int | Wallclock - actual time used (seconds) |  |
| CpuDuration | int | CPU time consumed (seconds) |  |
| CpuCount | int | Number of CPUs allocated |  |
| NetworkType | string | Description |  |
| NetworkInbound | int | GB received |  |
| NetworkOutbound | int | GB sent |  |
| Memory | int | Memory allocated to the VM (MB) |  |
| Disk | int | Disk allocated to the VM (GB) |  |
| StorageRecordId | string | Link to associated storage record |  |
| ImageId | string | Image ID |  |
| CloudType | string | e.g. OpenNebula, Openstack |  |

Scripts have been provided for OpenNebula and Openstack implementations to retrieve the accounting data required in this format ready to be sent to the Cloud Accounting Repository.  These scripts are available from:

* OpenNebula –<https://github.com/EGI-FCTF/opennebula-cloudacc>
* Openstack –<https://github.com/EGI-FCTF/osssm>

The APEL SSM (Secure Stomp Messenger) package is provided by STFC for resource providers to send their messages to the central accounting repository.  It is written in Python and uses the STOMP protocol, the messages contain cloud accounting records as defined above in the following format (example data added) where %% is the record delimiter:

APEL-cloud-message: v0.2

VMUUID: https://cloud.cesga.es:3202/compute/47f74797-e9c9-46d7-b28d-5f87209239eb 2013-02-25 17:37:27+00:00

SiteName: CESGA

MachineName: one-2421

LocalUserId: 19

LocalGroupId: 101

GlobalUserName: NULL

FQAN: NULL

Status: completed

StartTime: 1361813847

EndTime: 1361813870

SuspendDuration: NULL

WallDuration: NULL

CpuDuration: NULL

CpuCount: 1

NetworkType: NULL

NetworkInbound: 0

NetworkOutbound: 0

Memory: 1000

Disk: NULL

StorageRecordId: NULL

ImageId: NULL

CloudType: OpenNebula

%%

...another cloud record...

%%

...

%%

The OpenNebula and Openstack scripts produce the messages to be sent using the SSM package in the correct format.

The SSM package can be downloaded from here: <http://apel.github.io/apel/>

Detail about configuring SSM and publishing records may be found here:

<https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:Scenario4#Publishing_Records>

SSM utilizes the network of EGI message brokers and is run on both the Cloud Accounting server at STFC and on a client at the Resource Provider site.  The SSM running on the Cloud Accounting server receives any messages sent from the Resource Provider SSMs and they are stored in an “incoming” filesystem.

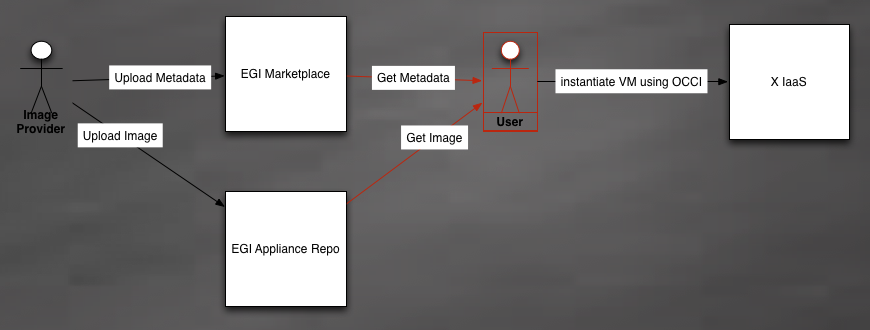
A Record loader package also runs on the Cloud Accounting server and checks the received messages and inserts the records contained in the message into the MySQL database.

A Cloud Accounting Summary Usage Record has also been defined and the Summaries created on a daily basis from all the accounting records received from the Resource Providers is sent to the EGI Accounting Portal.  The EGI Accounting Portal also runs SSM to receive these summaries and the Record loader package to load them in a MySQL database storing the cloud accounting summaries.

The EGI Accounting Portal provides a web page displaying different views of the Cloud Accounting data received from the Resource Providers:

<http://accounting.egi.eu/cloud.php>

## 4.5 Image metadata publishing & repository (K Koumantaros)

<<Describe the key components and concepts of the VM Marketplace, the Appliance repositories. Highlight the various roles and responsibilities of the involved stakeholders.>>

The task force uses the appliance repository and marketplace developed by stratuslab as repositories for images and their metadata.   IaaS providers endorse images that are suitable for their infrastructure by signing their metadata and uploading them on the marketplace (<https://marketpalce.egi.eu>)  and make the image available either to EGI appliance repository (<https://appliance-repo.egi.eu>) or their local appliance repository.  The user is then able to browse the metadata for suitable images to instantiate in one of the federated IaaS.

# 5. Federating Cloud Resources to EGI (M Turilli)

The model of federation chosen in the EGI is one where all resources available to the user are equal. therefore this is therefore suitable for resource consumers who do not have their own cloud infrastructure available to them. Within this section we detail the concepts behind the method of federation chosen and how those technology providers within the task force have been able to adapt or integrate their technologies. We therefore have for each technology provider a section which details work done for integration and any specific configuration a deployer of this technology needs to do to connect to EGI. We also describe technology premise that the activity itself started with.

Add here the criteria for being integrated (and counted) into the testbed.

Mention here also the certification process (that is one step further) to be officially counted in the prod infra figures

<<The key idea of this section is specificity. While the previous sections describe the architectural and design concepts behind the federated Cloud infrastructure, this section lays out the concrete steps a Resource Provider would have to undertake in order to technically integrate their Cloud resources with the EGI Cloud federation Section 5.1 will provide the 6 principal steps as a sort of overview.

The following subsections will list the specific actions pertinent to the described CMF. So for example, if a Resource Provider would want to deploy WNoDeS, section 5.5 would be the right section to read on how to do that.>>

## 5.1 Overview: 6 steps process (M Turilli)

The initial plan for federation of cloud resources within EGI was based on 6 different functional requirements;

1)

<<see above>>

## 5.2 OpenNebula (ON RP team)

(section 5.1 is still empty, I’m not sure what to focus on; here is my attempt to provide a somewhat concise step-by-step guide -- Boris)

A new Resource Provider using OpenNebula or OpenNebula-based CMF has to take the following steps to join the EGI Cloud Federation. There is only one prerequisite and that is fully functional OpenNebula installation capable of deploying, sustaining and shutting down virtual machines. There are no requirements for the underlying architecture, Resource Providers in question may choose the virtualization platform, network and storage configuration according to their preferences and needs. It is highly recommended to install OpenNebula v3.8.x where x denotes the latest security update and coordinate any future upgrades with the task force to avoid infrastructure fragmentation. Resource providers installing OpenNebula from scratch should follow its step-by-step installation and configuration guides available online [1].

The actual integration with the EGI Cloud Federation consists of the following steps:

1. Additional OpenNebula configuration
2. rOCCI-server installation and configuration
3. Integration with user management service -- Perun
4. Integration with accounting service -- APEL
5. Integration with image management service -- vmcaster/vmcatcher
6. Integration with information system -- LDAP/BDII
7. Registration of deployed services in GOCDB

Each of the above-mentioned steps is a requirement for every Resource Provider wishing to join the EGI Cloud Federation. Resource Providers are welcome to deploy and offer additional services such as object storage (CDMI) but this is not a requirement at this time. Detailed description of the listed steps is as follows.

**Additional OpenNebula configuration**

Integration with EGI Cloud Federation requires the use of X.509 authentication mechanism in communication with OpenNebula. Resource Providers are encouraged to follow the step-by-step configuration guide provided by OpenNebula developers available online [2]. There is no need to change authentication driver for the *oneadmin* user or create any user accounts manually at this time.

**rOCCI-server installation and configuration**

The EGI Cloud Federation uses OCCI as its VM management protocol. It is necessary to install a fully compliant OCCI 1.1 server on top of RP’s existing OpenNebula installation. OpenNebula’s OCCI implementation is *not* compliant with the OCCI 1.1 specification. This functionality is provided by the rOCCI-server project. Detailed installation and configuration instructions are available online in the FedCloud wiki [3].

**Integration with Perun**

The current rOCCI-server implementation doesn’t handle user management and identity propagation hence an integration with a third-party service is necessary. The Perun management server developed and maintained by CESNET is used to provide user management capabilities for OpenNebula Resource Providers [4]. It uses locally installed scripts (fully under the control of the Resource Provider in question) to propagate changes in the user pool to all registered Resource Providers. They are required to install and configure (if need be) these scripts and report back to EGI Cloud Federation for registration in Perun. Installation and configuration details are available online in the EGI-FCTF repository on GitHub [5].

**Integration with APEL**

One of the required integration points is accounting. The EGI Cloud Federation employs APEL framework with extended accounting records. Every Resource Provider is required to install the APEL SSM client and OpenNebula accounting script. As with the previous cases, installation and configuration details are available online on GitHub and in the FCTF wiki [6][7].

**Integration with vmcaster/vmcatcher**

Resource Providers are required to integrate their OpenNebula with an image management service used within the federation. As with the previous cases, installation and configuration details are available online in the FCTF wiki [9]. This service ensures that all images are trusted and up-to-date for all Resource Providers across the federation.

**Integration with TopBDII**

Details about services offered by the Resource Provider in question are advertised to the rest of the EGI Cloud Federation using an LDAP server -- BDII. Resource Providers are encouraged to follow instructions available online in the FCTF wiki [8].

**Registration in GOCDB**

(I’m sorry, I have no idea how the process of registration in GOCDB goes or what is required … generally speaking, it should be the same for all Resource Providers regardless of their CMF, right?)[10]

[1] <http://opennebula.org/documentation:archives:rel3.8>

[2] <http://opennebula.org/documentation:archives:rel3.8:x509_auth>

[3] <https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:_Federated_AAI:OpenNebula>

[4] <http://perun.metacentrum.cz/web/>

[5] <https://github.com/EGI-FCTF/fctf-perun>

[6] <https://github.com/EGI-FCTF/opennebula-cloudacc>

[7] (Is there a step-by-step installation and configuration guide for APEL SSM?)

[8] <https://wiki.egi.eu/wiki/Fedclouds_BDII_instructions>

[9] <https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:Scenario8:Configuration#VMcatcher>

[10] (Is there a step-by-step registration guide for a FedCloud site in GOCDB?)

## 5.3 OpenStack (OS RP team)

<<see above>>

Mention glancepush and glancpush-vmcaster

This section describes steps necessary for new Resource Provider (RP) using Openstack middleware to join EGI Cloud Federation. It is strongly recommended using the last Openstack version. Specifically, the VOMS-enable authentication will require Grizzly version of Keystone. The installation and configuration of Openstack are available at [1].

The actual integration with the EGI Cloud Federation consists of the following steps:

a)      VOMS-enable Keystone installation and configuration

b)      OCCI installation and configuration

c)       Integration with accounting service APEL

d)      Integration with vmcaster/vmcatcher

e)      Integration with information system LDAP/BDII

f)       Registration of deployed services in GOCDB

Each of the above-mentioned steps is a requirement for every Resource Provider wishing to join the EGI Cloud Federation. Resource Providers are welcome to deploy and offer additional services such as object storage (CDMI) but this is not a requirement at this time. Detailed description of the listed steps is as follows.

**a)      VOMS-enable Keystone installation and configuration**

The installation and configuration of VOMS-enable Keystone is available at [2]. That will enable X.509 authentication mechanism and allows users with valid VOMS proxy certificate to log in. The actual VO for EGI Cloud Federation fedcloud.egi.eu should be enabled in the configuration. There is an option for automatic creating new users for trusted VO on the fly.

**b)      OCCI installation and configuration**

The steps of installation and configuration of OCCI is available at [3]. The installation and configuration should be done on the machine with Nova server. The OOCI implementation is not perfect; occasionally Nova server needs to be restarted for refreshing OCCI configuration (especially when new images are added).

**c)       Integration with accounting service APEL**

Like RP with OpenNebula, the client for accounting service APEL must be installed and configured. The details of installation and configuration of APEL for Openstack is available at [4], [5].

**d)      Integration with vmcaster/vmcatcher**

Resource Providers are required to integrate their Openstack with an image management service used within the federation. Installation and configuration details are available online in the FCTF wiki [6]. This service ensures that all images are trusted and up-to-date for all Resource Providers across the federation. In addition to vmcaster/vmcatcher, glancepush-vmcatcher [7] uses vmcatcher's event handler to signal glancepush that a new image was updated in vmcatcher's cache and glancepush will check and publish images from vmcatcher cache to glance service in Openstack.

**e)      Integration with information system LDAP/BDII**

Integration with BDII for RP with Openstack is identical as in the OpenNebula case. The instructions are available online in the FCTF wiki [8].

[1] http://docs.openstack.org/install/

[2] http://keystone-voms.readthedocs.org/en/latest/index.html

[3] https://github.com/tmetsch/occi-os

[4] https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:Scenario4

[5] https://github.com/EGI-FCTF/osssm/wiki

[6] https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:Scenario8:Configuration#VMcatcher

[7] https://github.com/EGI-FCTF/glancepush

[8] https://wiki.egi.eu/wiki/Fedclouds\_BDII\_instructions

## 5.4 StratusLab (StratusLab RP team)

A StratusLab cloud based production release offers computing, storage, and networking services as well as the Marketplace, a high-level service to facilitate sharing of machine images.

Actually StratusLab integrates and uses OpenNebula as the Virtual Machine Manager.

In order to technically integrate a StratusLab Cloud with the EGI Cloud Federation, the following steps should be implemented:

Authentication:

X509/VOMS authentication is done by creating users in the OpenNebula VMM service with the X509 driver. This driver should be enabled in the OpenNebula configuration file.

As the VOMS validation is passing through Apache2, grid\_site and then rOCCI-server. Apache2 and grid\_site should be properly configured. In our case, install gridsite packages, then load gridsite module in apache configuration file.

Compute: Nothing to do, rOCCI-server is well integrated with OpneNebula, and StratusLab actually integrates OpenNebula as it’s Virtual Machine Manager.

Network: StratusLab networking service permits the allocation of “public”, “private” and “local” network. When creating VM templates one of these networks should be specified.

Public IP are visible from outside and inside the cloud.

Local IP are visible from the Cloud (VMs running in the Cloud), access external services through NAT, this type of IP addresses could be useful for MPI jobs.

Private IP are visible only fron the host where they are running, go to outside via NAT.

Storage: StratusLab developed it’s own storage service solution based on a disk approach. One of its functionnality is to cache machine images, making deployment of VM very fast.

This service is well integrated with the other StratusLab services, and doesn’t need any additional configuration for rOCCI-server.

Marketplace: We are using StratusLab Marketplace to instantiate VM in the StratusLab Cloud. The StratusLab Marketplace is like a registry of images metadata.

Unlike the other RPs, in the metadata we don’t specify network nor storage elements. Instead in the VM templates it’s mandatory to specify identifier image url  from the marketplace in the SOURCE filed. (E.g. SOURCE=http://marketplace.egi.eu/metadata/HGSxEvjFP0TUo1mMcT-M63Y-2KF/airaj@lal.in2p3.fr/2013-02-12T14:11:55Z).

NB. In StratusLab, OpenNebula is used only as VMM service, and in the near future, it’ll be dropped. Integration will be done directly with libvirt.

More precisely, this configuration will works till the StratusLab 13.05 release, OpenNebula as VMM will be dropped in the  StratusLab 13.08 release, in August 2013.

## 5.5 WNoDeS (E. Ronchetti)

<<see above>>

## 5.6 Synnefo (K Koumantaros)

Synnefo (<http://www.synnefo.org/>) is open source cloud software used to create massively scalable IaaS clouds. It uses Google Ganeti for the low level VM management and also talks to the outside world through the OpenStack APIs with extensions for advanced operations.  Synnefo in conjunction with Google GANETI (<https://code.google.com/p/ganeti/>) is the software thats empowers GRNETs ~Okeanos service (<https://okeanos.grnet.gr>) that has currently supports 2100 users

with 2941 VMs and 10119 Virtual cores.   ~Okeanos is only partialy integrated with the Federated cloud infrastructure using snf-occi (<http://www.synnefo.org/docs/snf-occi/latest/index.html>), an implementation of the OCCI specification on top of synnefo’s API kamaki.   Development for the rest of the modules required is currently foreseen for the near future but due to lack of manpower and parallel developments of the synnefo API there is no estimate for the date of delivery for each module.

# 6. Conclusions (D Wallom)

# Conclusion

# References

|  |  |
| --- | --- |
| R 1 |  |
| R 2 |  |
| R 3 |  |
| R 4 |  |
| R 5 |  |

1. <https://wiki.egi.eu/wiki/Fedcloud-tf:Blueprint> [↑](#footnote-ref-1)
2. <https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:_Federated_AAI> [↑](#footnote-ref-2)
3. <https://wiki.egi.eu/wiki/Fedcloud-tf:Testbed> [↑](#footnote-ref-3)
4. <https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:Scenario5> [↑](#footnote-ref-4)
5. <https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:Scenario4> [↑](#footnote-ref-5)
6. <https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:Scenario3> [↑](#footnote-ref-6)
7. <https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:_Outreach#Requirements> [↑](#footnote-ref-7)
8. <https://wiki.egi.eu/wiki/Fedcloud-tf:Blueprint:Solutions_Intentory> [↑](#footnote-ref-8)
9. <https://wiki.egi.eu/wiki/Fedcloud-tf:WorkGroups:_Outreach> [↑](#footnote-ref-9)
10. <https://wiki.egi.eu/wiki/Fedcloud-tf:Blueprint:Security_and_Policy> [↑](#footnote-ref-10)