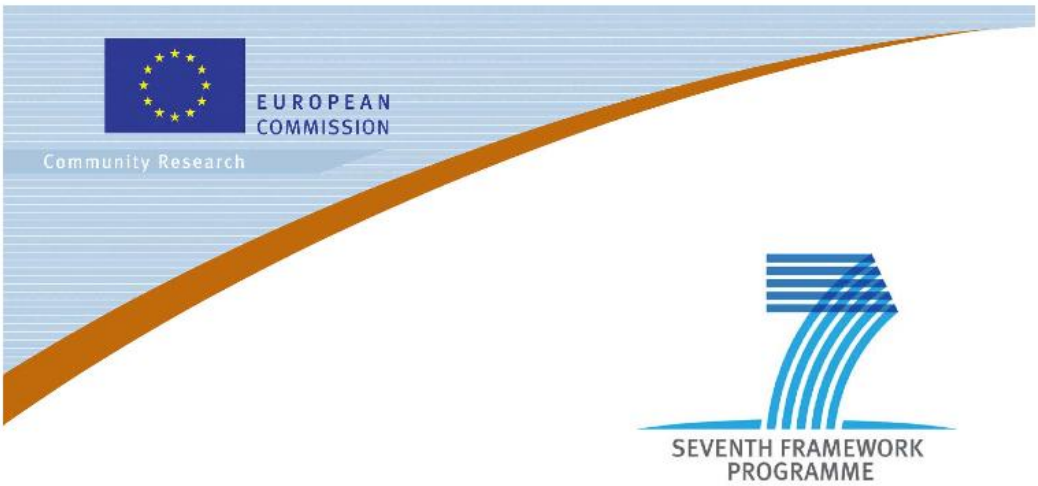


Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)

D.4.1.3: FI-WARE GE Open Specification

Project acronym: FI-WARE
Project full title: Future Internet Core Platform
Contract No.: 285248
Strategic Objective: FI.ICT-2011.1.7 Technology foundation: Future Internet Core Platform
Project Document Number: ICT-2011-FI-285248-WP4-D.4.1.3
Project Document Date: 2014-06-15
Deliverable Type and Security: Public
Author: FI-WARE Consortium
Contributors: FI-WARE Consortium

1.1 Executive Summary

This document describes the Generic Enablers in the Cloud Hosting chapter, their basic functionality and their interaction. Each GE Open Specification is first described at a generic level, describing the functional and non-functional properties and is supplemented by a number of specifications according to the interface protocols, API and data formats.

1.2 About This Document

FI-WARE GE Open Specifications describe the open specifications linked to Generic Enablers GEs of the FI-WARE project (and their corresponding components) being developed in one particular chapter.

GE Open Specifications contain relevant information for users of FI-WARE to consume related GE implementations and/or to build compliant products, which can work as alternative implementations of GEs developed in FI-WARE. The later may even replace a GE implementation developed in FI-WARE within a particular FI-WARE instance. GE Open Specifications typically include, but not necessarily are limited to, information such as:

- Description of the scope, behavior and intended use of the GE
- Terminology, definitions and abbreviations to clarify the meanings of the specification
- Signature and behavior of operations linked to APIs (Application Programming Interfaces) that the GE should export. Signature may be specified in a particular language binding or through a RESTful interface.
- Description of protocols that support interoperability with other GE or third party products
- Description of non-functional features

1.3 Intended Audience

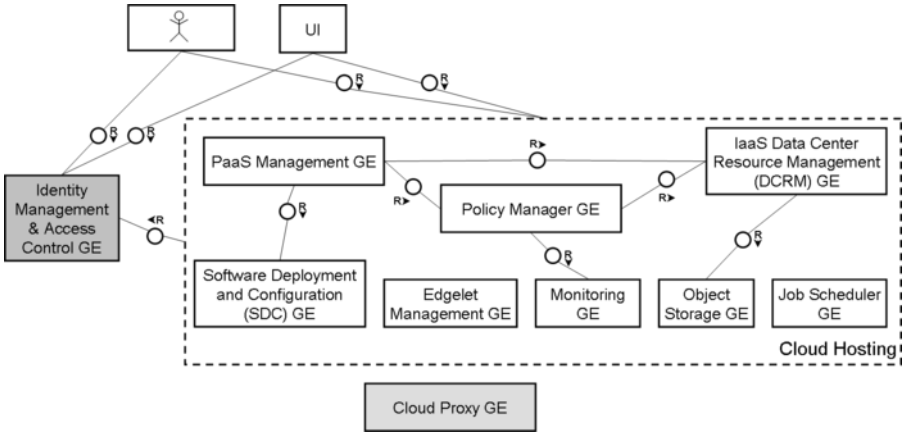
The document targets interested parties in architecture and API design, implementation and usage of FI-WARE Generic Enablers from the FI-WARE project.

1.4 Chapter Context

The Cloud Chapter offers Generic Enablers that comprise the foundation for designing a modern cloud hosting infrastructure that can be used to develop, deploy and manage Future Internet applications and services, as outlined in Materializing Cloud Hosting in FI-WARE.

The capabilities available in the second release of FI-WARE Cloud Hosting platform are outlined in Roadmap of Cloud Hosting.

The following diagram shows the main components (Generic Enablers) that comprise the second release of FI-WARE architecture.



The architecture comprises a set of Generic Enablers that together provide hosting capabilities of several kinds and at several levels of resource abstraction -- aiming at the needs of different applications hosted on the cloud platform. **IaaS Data Center Resource Management (DCRM) GE** is offering provisioning and life cycle management of virtualized resources (compute, storage, network) associated with **virtual machines**, which can run general purpose Operating Systems as well as arbitrary software stacks. Application developers and providers can use these virtual machines to develop and deploy their own software components that comprise their application stacks. **Object Storage GE** offers provisioning and life cycle management of **object**-based storage containers and elements, which can be efficiently used to store unstructured fixed content (such as images, videos, etc) as well as accompanying metadata. **Job Scheduler GE** offers the application to submit and manage computational jobs in a unified and scalable manner. **Edgelet Management GE** offers the capability to host lightweight application components, called *edgelets*, on devices typically located outside of the Data Center, such as those provided by the **Cloud Proxy GE** (developed jointly by the Cloud chapter and the Interfaces to Network and Devices chapter). **Software Deployment and Configuration (SDC) GE** offers a flexible framework for installation and customization of software products within individual virtual machines. **Policy Manager GE** provides a framework for rule-based management of cloud resources, including application auto-scaling based leveraging metrics collected by **Monitoring GE**. Lastly, **PaaS Management GE** uses the above capabilities to offer holistic provisioning and ongoing management of complex workloads comprising sophistical combination of interdependent VMs and associated resources (such as multi-tier web applications or even complete custom-built PaaS environments), as well as configuration and management of software components within the VMs.
Each of the above GEs provides a REST API that can be used programmatically. The human actor represents the programmatic user of the different capabilities of the Cloud GEs via REST APIs. Moreover, the Cloud chapter provides a Web-based **Portal** (part of of the **UI** layer) , which surfaces main capabilities in an interactive manner --such as provisioning and monitoring of VM instances and services.
Cloud Hosting Generic Enablers are using the **Identity Management and Access Control** framework provided by the Security chapter, as outlined in the Cloud Security Architecture.

1.5 Structure of this Document

The document is generated out of a set of documents provided in the public FI-WARE wiki. For the current version of the documents, please visit the public wiki at http://wiki.fi-ware.eu/

The following resources were used to generate this document:

D.4.1.3 FI-WARE GE Open Specifications front page

FIWARE.OpenSpecification.Cloud.DCRM
DCRM OpenStack Open RESTful API Specification
DCRM OCCI Open RESTful API Specification
FIWARE.OpenSpecification.Cloud.SelfServiceInterfaces
FIWARE.OpenSpecification.Cloud.ObjectStorage
Object Storage Open RESTful API Specification
FIWARE.OpenSpecification.Cloud.SDC
SDC Open RESTful API Specification
FIWARE.OpenSpecification.Cloud.PaaS
PaaS Open RESTful API Specification
FIWARE.OpenSpecification.Cloud.Monitoring
Monitoring Open RESTful API Specification
FIWARE.OpenSpecification.Cloud.PolicyManager
Policy Manager Open RESTful API Specification
FIWARE.OpenSpecification.Cloud.JobScheduler
Job Scheduler Open RESTful API Specification
FIWARE.OpenSpecification.Cloud.Edgelets
Edgelets Open API Specification

1.6 Typographical Conventions

Starting with October 2012 the FI-WARE project improved the quality and streamlined the submission process for deliverables, generated out of the public and private FI-WARE wiki. The project is currently working on the migration of as many deliverables as possible towards the new system.

This document is rendered with semi-automatic scripts out of a MediaWiki system operated by the FI-WARE consortium.

1.6.1 Links within this document

The links within this document point towards the wiki where the content was rendered from. You can browse these links in order to find the "current" status of the particular content.

Due to technical reasons not all pages that are part of this document can be linked document-local within the final document. For example, if an open specification references and "links" an API specification within the page text, you will find this link firstly pointing to the wiki, although the same content is usually integrated within the same submission as well.

1.6.2 Figures

Figures are mainly inserted within the wiki as the following one:

```
[[Image:....|size|alignment|Caption]]
```

Only if the wiki-page uses this format, the related caption is applied on the printed document. As currently this format is not used consistently within the wiki, please understand that the rendered pages have different caption layouts and different caption formats in general. Due to technical reasons the caption can't be numbered automatically.

1.6.3 Sample software code

Sample API-calls may be inserted like the following one.

```
http://[SERVER_URL]?filter=name:Simth*&index=20&limit=10
```

1.7 Acknowledgements

The following partners contributed to this deliverable: IBM, TID, INTEL, UPM, TRDF, Inria, THALES

1.8 Keyword list

FI-WARE, PPP, Roadmap, Reference Architecture, Generic Enabler, Open Specifications, I2ND, Cloud, IoT, Data/Context Management, Applications/Services Ecosystem, Delivery Framework, Security, Developers Community and Tools

1.9 Changes History

Release	Major changes description	Date	Editor
v1	Draft of deliverable submission	2014-05-25	IBM
v2	Updated document	2014-06-15	IBM

1.10 Table of Contents

2 FIWARE OpenSpecification Cloud DCRM

2.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.org and similar pages in order to understand the complete context of the FI-WARE project.

2.2 Copyright

Copyright © 2011-2014 by IBM and Intel Corporations

2.3 Legal notice

Please check the following Legal Notice to understand the rights to use these specifications.

2.4 Overview

This specification describes the DataCenter Resource Management (DCRM) GE, which is a key enabler to build a cloud solution.

The DCRM GE provides the basic Virtual Machine (VM) hosting capabilities, as well as management of the corresponding resources within the DataCenter that hosts a particular FI-WARE Cloud Instance.

The baseline for the DCRM GE is **OpenStack**. Hence, DCRM offers all the capabilities that OpenStack natively provides to cloud hosting users and cloud hosting providers plus some unique extended capabilities.
The main capabilities provided for a **cloud hosting user** are:

- Browse VM template catalogue and provision a VM with a specified virtual machine image
- Manage life cycle of the provisioned VM
- Manage network and storage of the VM
- Resource monitoring of the VM
- Resiliency of the persistent data associated with the VM
- Manage resource allocation (with guarantees)
- Secure access to the VM

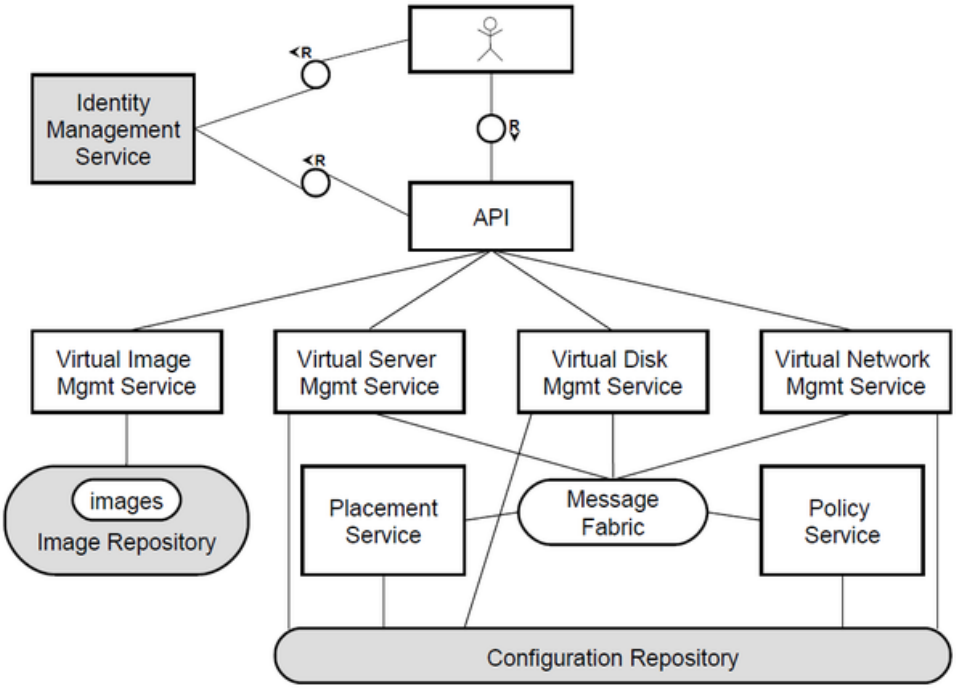
For a **cloud hosting provider**, the following capabilities are provided:

- Resource optimization and over-commit (aimed at increasing the utilization and decreasing the hardware cost)
- Capacity management and admission control (allowing to easily monitor and control the capacity and the utilization of the infrastructure)
- Multi-tenancy (support isolation between VMs of different accounts)
- Automation of typical admin tasks (aimed at decreasing the admin cost)

- Resiliency of the infrastructure and of the management stack (aimed at reducing outage due to hardware failures)

2.4.1 DCRM components

The following diagram shows the main components of the DCRM GE, as well as its main interactions.



Main components of the DCRM Architecture

On the above diagram, the *API* component is the front-end of DCRM. It can be implemented either as a set of endpoints handling each of the resource types -- virtual servers, virtual disks, virtual networks and virtual images, or a single endpoint dispatching the incoming requests to the corresponding 'backend' runtime. At the back-end, different aspects of resource management are handled by corresponding internal services, such as policy service and placement service.

2.4.2 DCRM-specific features

With respect to the OpenStack baseline, DCRM provides in addition the following set of high-level advanced features:

- Shared storage configuration enabling live VM migration and related scenarios
- VM High Availability
- Adaptive scheduling for optimized resource utilization
- Support for QoS guarantees for workloads
- Support for placement policies
- Support of concurrent management and deployment workflows in a scalable consistent manner
- Unified management of heterogeneous environments

- Support for policy-based virtual network connectivity

2.4.3 Extended capabilities with respect to OpenStack

The high-level features listed above are enabled by the combination of a set of extended capabilities that can also be used in isolation. They are:

- Host failure detection (Zookeeper): enables the automatic recovery of VMs from a failed host
- Advanced Scheduler:
  - Flexible global resource optimization based on large and extensible set of metrics, including resource utilization: enables the implementation and configuration of several resource utilization optimization policies for infrastructure providers
  - Ongoing placement optimization using live migration and a solver: live migration of VMs across hosts allows continuous optimization of VMs placement according to configurable policies with no service disruption
  - Placement support for automated HA of VM instances: in an HA scenario, if a host fails, the placement logic will automatically provide correct placement for all recovered VMs
  - Host evacuation support: this capability simplifies infrastructure management tasks by automatically moving out all VMs from one host (host maintenance scenario)
  - Support for anti-affinity / placement policies: this capability allows cloud users to specify constraints and requirements with respect to VMs placement. This capability enables both service availability and VM proximity scenarios for cloud applications
  - HA-aware admission control: this mechanism is used to guarantee that at all times the system will have sufficient host spare capacity to recover from a single host failure
  - Unified support for heterogeneous performance of the underlying HW: a translation layer is used to assign correct VCPU nominal performance on heterogeneous hosts. This prevents wasting capacity and enables optimal resource utilization
  - Fine-grained compatibility verification: it prevents VMs from being deployed on incompatible HW
  - Flexible resource allocation policies and adaptive resource over-commit based on idleness detection: VM admission control adapts to the number of idle VMs in the system, maximizing resource utilization

2.5 Basic Concepts

The key concepts visible to the cloud user are:

- **Virtual server**, a virtualized container that can host an arbitrary Operating System and arbitrary software stack on top, installed within the virtual server. Virtual servers are also referred as Virtual Machines (VMs) or **(virtual) image instances** (see definition of virtual image below). DCRM GE supports provisioning and life cycle management of Virtual Servers.

- **Virtual disk**, representing a persistent virtual disk that can be potentially attached to an arbitrary virtual server. DCRM GE supports provisioning of virtual disks, as well as their attachment to virtual servers.
- **Virtual network**, representing a logical network abstraction that would typically represent a network segment at layer 2 of the OSI model. DCRM GE supports provisioning of virtual networks, as well as attachment of virtual NICs of virtual servers to them.
- **Computing Flavor**, a computing flavor is a hardware configuration that can be associated to a virtual server. Each flavor has a unique combination of CPU cores, memory capacity and disk space. An example of this combination could be 8 virtual CPUs, 16 Gb RAM, 10 Gb HDD and 160 Gb of ephemeral disk (a disk that is stored locally on the hypervisor host). Computing flavors must be registered and made available prior to their association to virtual servers.
- **Virtual image**, an image is a collection of packaged files used to create or rebuild a virtual server. Basically, a virtual image is a snapshot of a virtual server from which you can create new virtual servers (i.e., image instances). Each virtual server derived from a virtual image hosts the Operating System and software stack associated to the virtual image and is assigned one among the set of available computing flavors, each of which maps to a configuration of computing resources (memory, CPU, etc.). A number of pre-built virtual images can be made available to cloud users but they may also create their own images using tools defined for that purpose. These custom images are useful for backup purposes or for producing "gold" server images if you plan to deploy a particular server configuration frequently. DCRM GE supports life cycle of virtual images, as well as provisioning of virtual servers based on virtual images.

2.6 Main Interactions

DCRM provides a wide variety of operations to provision and manage the life cycle of cloud resources. The most important ones are listed below as conceptional operations, the API itself might use different syntax and is specified in the API specification section.

2.6.1 Virtual Images

- ***listVirtualImages*** -- Returns a list of all available virtual images (visible by the authenticated user)
- ***queryVirtualImages*** -- Returns a list of available virtual images, filtered by given query criteria
- ***getVirtualImageDetails*** -- Returns details of a virtual image (type, size, creation details, etc.)
- ***uploadVirtualImage*** -- Uploads a new virtual image into the virtual image repository

2.6.2 Virtual Servers

2.6.2.1 ***Provisioning***

- ***createVirtualServer*** -- Provisions a new virtual server with the given properties (virtual hardware, policy parameters, access, etc). Returns unique ID of the virtual server.
- ***destroyVirtualServer*** -- Removes a virtual server

2.6.2.2 ***Power Management***

- ***powerOnVirtualServer*** -- Powers on a virtual server
- ***powerOffVirtualServer*** -- Powers off a virtual server
- ***RestartVirtualServer*** -- Restarts a virtual server
- ***ShutdownVirtualServer*** -- Shuts down a virtual server (note: the ability to perform this operation on the fly depends on the capabilities of the underlying virtualization platform)

2.6.2.3 ***Reconfiguration***

- ***resizeVirtualServer*** -- Changes the virtual hardware allocation for a virtual server, e.g., allocated RAM or number of CPUs (note: the types of resources for which the reconfiguration can be done on the fly depends on the capabilities of the underlying virtualization platform)

2.6.2.3.1 *Inventory*

- ***getVirtualServerDetails*** -- Returns details of a virtual server (virtual hardware specification, state, associated policy parameters, access details, etc.)

2.6.3 Virtual Disks

2.6.3.1 ***Provisioning***

- ***createVirtualDisk*** -- Provisions a new virtual disk with the given properties (size, capabilities, etc.). Returns unique ID of the virtual disk.
- ***destroyVirtualDisk*** -- Removes a virtual disk

2.6.3.2 ***Attachment***

- ***attachVirtualDisk*** -- Attaches a given virtual disk to a given virtual server (note: the ability to perform this operation on the fly depends on the capabilities of the underlying virtualization platform)
- ***detachVirtualDisk*** -- Detaches a given virtual disk from a given virtual server (note: the ability to perform this operation on the fly depends on the capabilities of the underlying virtualization platform)

2.6.3.3 ***Inventory***

- ***getVirtualDiskDetails*** -- Returns details of a given virtual disk (size, capabilities, attachment details, etc.)

2.6.4 Virtual Networks

2.6.4.1 *Provisioning*

- ***createVirtualNetwork*** -- Provisions a new virtual network with the given properties (e.g., VLAN ID, capabilities, etc.). Returns unique ID of the virtual network.
- ***destroyVirtualNetwork*** -- Removes a virtual network

2.6.4.2 *Attachment*

- ***attachVirtualServerToNetwork*** -- Attaches a virtual network interface of a given virtual server to a given virtual network
- ***detachVirtualServerFromNetwork*** -- Detaches a virtual network interface of a given virtual server from a given virtual network

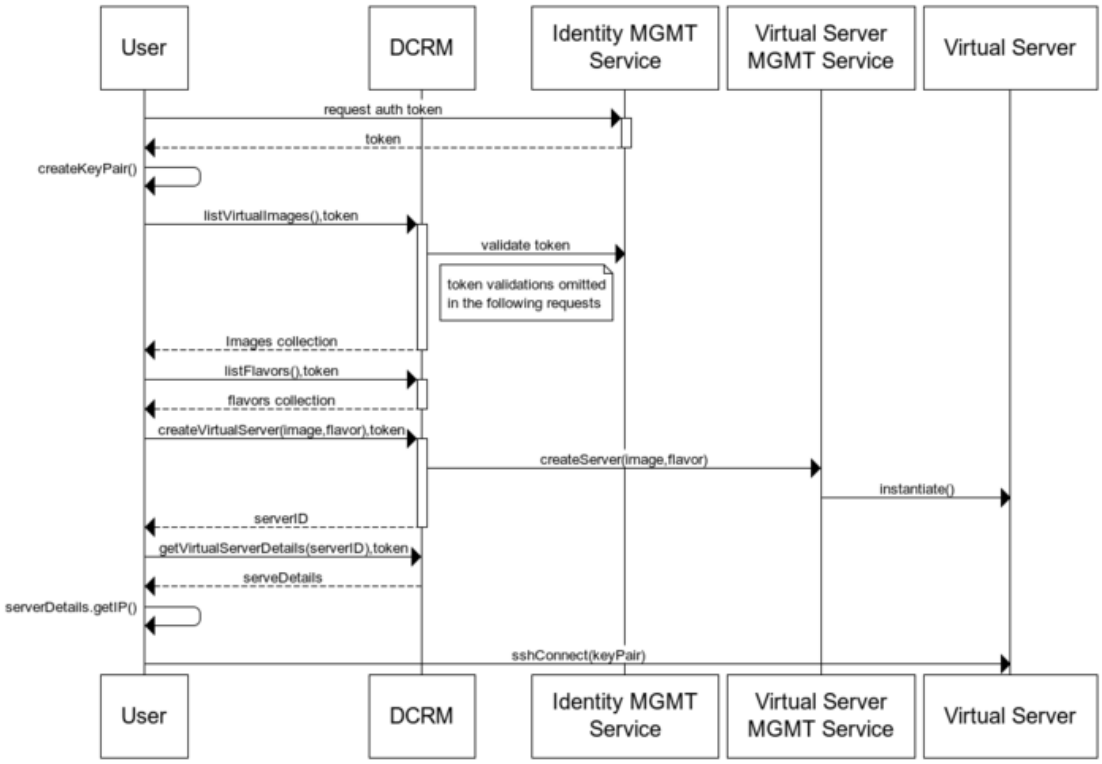
2.6.4.3 *Inventory*

- ***getVirtualNetworkDetails*** -- Returns details of a given virtual network (ID, capabilities, attachment details, etc.)

2.6.5 Example Scenario

The following sequence of operations describes a typical (simple) scenario of provisioning of a virtual server hosted in the Cloud:

- User authenticates with Identity Management GE, receives a token
- User creates ssh key-pair, to be used to authenticate with the guest OS within the virtual server instances
- User retrieves a list of available images and of virtual server flavors
- User requests a new virtual server
- User verifies that the virtual server creation has completed
- User retrieves the IP address allocated for the virtual server
- User connects to the virtual server using ssh



2.7 Basic Design Principles

When applied to DCRM, the general design principles outlined at Cloud Hosting Architecture can be translated into the following key design goals:

- **Fully-automated provisioning and life cycle of compute, storage and network resources, requested, managed and released via a standards-based REST API:** The REST API allows management of the provisioned resources both through a Web-based user interface or direct API invocation. The API is designed to be abstract and "declarative": a tenant specifies "what" he needs, while the "how" of the provisioning is left to the infrastructural policies and goals. The goal is to provide a standard interface to consume the virtual resource service regardless of the underlying technology used to implement the provisioning infrastructure.
- **High resource utilization, while providing the necessary levels of isolation, availability and performance of provisioned resources:** Improved utilization and automation of resources allow greater cost efficiencies for both infrastructure providers and tenants.
- **Ability to dynamically control the amount of allocated resources, as well as to monitor the actual resource usage:** Dynamic control of resource provisioning is at the core of application elasticity, enabling the correct sizing of applications' components and operating costs to the varying load conditions.
- **High availability and scalability of the management stack:** The infrastructure management components provide availability and scalability through the most advanced current design and development practices, including: fully-distributed shared-nothing architectures to naturally support horizontal scalability, asynchronous communication mechanisms, and extensive automated testing cycles for each contribution.
- **Non-disruptive, automated administrative tasks (e.g., infrastructure maintenance):** when scale grows, partial hardware and software failures are

the norm rather than the exception. Infrastructure providers require mechanisms to automate administrative tasks reducing the needed effort and preventing any disruption to the tenants' **services and applications**.

- **Avoid non-authorized access to resources and workloads:** Role Based Access Control (RBAC), coupled with an Identity Management service, ensure security by user, role and project.

FIWARE.OpenSpecification.Details.Cloud.DCRM

2.8 Re-utilised Technologies/Specifications

The DCRM OpenStack API is a RESTful, resource-oriented API accessed via HTTP that uses XML-based and/or JSON-based representations for information interchange. It builds on top of the Compute, Identity, Images, and Network OpenStack API v2.0, but it extends the original specifications to provide support to additional DCRM-specific management features currently not available in OpenStack.

2.9 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be help to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FIWARE Global Terms and Definitions

FIWARE.Glossary.DCRM

3 DCRM OpenStack Open RESTful API Specification

3.1 Introduction to the DCRM OpenStack API

3.1.1 DCRM OpenStack API Core

The DCRM OpenStack API is a RESTful, resource-oriented API accessed via HTTP that uses XML-based and/or JSON-based representations for information interchange. It builds on top of the Compute, Identity, Images, and Network OpenStack API v2.0, but it extends the original specifications to provide support to additional DCRM-specific management features currently not available in OpenStack.

As in OpenStack, these APIs allow direct management of compute, network, and storage infrastructure. In addition, they have been enhanced with the following advanced management features:

- Live VM migration
- Host evacuation
- Ongoing optimization
- VM High Availability (HA)
- VM Groups
- Advancement placement engine (e.g., group-based anti-affinity)
- Smart resource over-commit
- Virtual networking

3.1.2 Intended Audience

This specification is intended for software developers aiming at interfacing their software with the DCRM. This document provides a full specification of how to manage the main entities of DCRM, which are **virtual servers**, **virtual disks**, **virtual networks**, **virtual images**, and **policies**.

In order to use this specifications, the reader should have a general understanding of the DCRM Generic Enabler.

To use this information, the reader should also be familiar with:

- the OpenStack Compute API Specification version 2.0
- the OpenStack Identity API Specification version 2.0
- the OpenStack Image API Specification version 2.0
- the OpenStack Network API Specification version 2.0
- RESTful web services
- HTTP/1.1 (RFC2616)
- JSON and/or XML data serialization formats.

3.1.3 API Change History

This version of the DCRM OpenStack API Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
Feb 25, 2013	R2
Apr 23, 2014	R3

3.1.4 How to Read This Document

All FI-WARE RESTful API specifications will follow the same list of conventions and will support certain common aspects. Please check Common aspects in FI-WARE Open Restful API Specifications.

For a description of the terms used along this document, see DCRM Generic Enabler.

You may also read the OpenStack API reference (v2.0)

3.1.5 Additional Resources

You can download the most current version of this document from the FI-WARE API specification selecting **PDF Version** from the Toolbox menu (left side), which will generate the file to download it. For more details about the **DCRM Generic Enabler** that this API is based upon, please refer to FI-WARE Cloud Hosting. Related documents, including an Architectural Description, are available at the same site.

3.2 General DCRM OpenStack API Information

3.2.1 Endpoints

With the aim of preserving the API separation that is part of the OpenStack architecture, the DCRM Openstack API also supports different endpoint configuration for each of the underlying APIs.

In a generic deployment, each API will have its own endpoint with the following URL template: http://{serverRoot}:{serverPort}/v2.0

For reference we will identify them in the following with these endpoints:

API	Associated Endpoint
Compute	http://{computeAPIserver}:{computeAPIport}/v2.0
Block Storage	http://{volumeAPIserver}:{volumeAPIport}/v2.0
Identity	http://{identityAPIserver}:{identityAPIport}/v2.0
Image	http://{imageAPIserver}:{imageAPIport}/v2.0

Network	http://{networkAPIserver}:{networkAPIport}/v2.0

3.2.2 Resources Summary

Each API manages a specific set of resources. With the aim of providing both conciseness and precision when describing the DCRM API, we deal with each of the underlying APIs separately. We also take a differential approach in which we describe only the changes and extensions that were made to the original APIs in order to support the advanced management features introduced by DCRM, rather than providing an extensive replication of publicly available content.

3.2.2.1 *Compute API*

DCRM R3 API is compatible with OpenStack Compute API (v2.0), and supports standard extensions delivered in Havana release of OpenStack, as well as specific scheduler hints outlined in the Compute API Specification section below.

The base URL is http://{computeAPIserver}:{computeAPIport}/v2.0.

3.2.2.2 *Block Storage API*

DCRM R3 API is compatible with OpenStack Block Storage API (v2.0). The base URL is http://{volumeAPIserver}:{volumeAPIport}/v2.0.

3.2.2.3 *Network API*

Release R3 of the DCRM fully supports the OpenStack Network API Specification version 2.0 with extensions specified in the Network API Specification section below.

The base URL is http://{networkAPIserver}:{networkAPIport}/v2.0.

The different Uniform Resource Identifiers (URIs) that can be used in the Network API are networks, subnets, ports, routers and floatingIPs.

3.2.3 Authentication

Each HTTP request to the DCRM OpenStack API requires the inclusion of specific authentication credentials.

The default authentication mechanism uses the OpenStack Keystone Identity API. As all FI-WARE RESTful APIs, the DCRM OpenStack API will evolve to support an authentication mechanism based on OAuth relying on the capabilities of a product in compliance with the Identity Management GE Open Specifications.

3.2.4 Representation Format

The DCRM OpenStack API supports both XML and JSON request formats. The request format is specified using the Content-Type header and is required for operations that have a request body.

The response format can be specified in requests using either the Accept header or adding an .xml or .json query extension to the request URI. If there is a conflict between Accept header and a query extension, the query extension takes precedence.

While there is no default request format, if no response format is specified, JSON is the default.

3.2.5 Resource Identification

As in the underlying OpenStack API implementations, resources are unambiguously identified by providing an ID or a URL. "When providing an ID, it is assumed that the resource exists in the current OpenStack deployment" [1]

For HTTP transport, this is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

3.2.6 Links and References

Resources often lead to refer to other resources. In those cases, we have to provide an ID or an URL to a remote resource.

In the DCRM OpenStack API, resources contain links to themselves. This allows client to avoid constructing resource URIs by composing IDs. Three link types can be associated with a resource:[2]

- **`self` link** is a versioned link to the resource
- **`bookmark` link** provides a permanent link to the resource
- **`alternate` link** provides another URL (possibly used in another context) for the same resource

3.2.7 Limits

Limits follow the standard specification for FI-WARE APIs Common aspects in FI-WARE Open Restful API Specifications

3.2.8 Versions

The DCRM OpenStack API uses both a URI and a MIME type versioning scheme, following the commin versioning policies adopted in all OpenStack APIs. See for instance the Openstack Compute API V2.0

3.2.9 Extensions

Extensions follow the standard specification for FI-WARE APIs Common aspects in FI-WARE Open Restful API Specifications

3.2.10 Faults

Synchronous[3] and asynchronous faults[4] are dealt with according to the common OpenStack policy

3.3 API Operations

In this section we go in depth for each operation. In order to preserve the OpenStack API organization, we also subdivide this section in compute, image, and network APIs.

3.3.1 Compute API Extensions

DCRM R3 supports standard OpenStack extensions shipped with OpenStack Havana release. Moreover, the FI-WARE Scheduler supports the following features:

- host aggregates (AggregateExtraSpec filter)
- availability zones
- instance groups (see below)

3.3.1.1 *Group Affinity Filters*

When creating a VM, specify it's group with group scheduler_hints

```
$ nova boot --image cedef40a-ed67-4d10-800e-17455edce175 --
flavor 1 \
 --hint group=foo server-1
```

By default, the group foo will be considered anti-affinity type. So in the above example, a VM named server-1 would not be placed with any VMs belonging to group foo. To specify affinity group, use affinity namespace (i.e. --hint group=affinity:foo). Notice that you could specify several affinity groups by stacking them in a list (i.e. --hint group=foo --hint group=foo1)

Or if you use the api:

```
{
'server': {
'name': 'server-1',
'imageRef': 'cedef40a-ed67-4d10-800e-17455edce175',
'flavorRef': '1'
},
'os:scheduler_hints': {
'group': 'foo'
}
}
```

```
{
'server': {
'name': 'server-1',
'imageRef': 'cedef40a-ed67-4d10-800e-17455edce175',
'flavorRef': '1'
},
'os:scheduler_hints': {
'group': ['foo', 'foo1']
}
}
```

3.3.2 Network API Extensions

Release R3 of the DCRM fully supports the OpenStack Network API Specification version 2.0, including its Layer-3 Networking Extension.

NOTE: in order to get external connectivity for VMs, the DC administrator needs to use the Neutron API and follow the Common L3 Workflow.

4 DCRM OCCI Open RESTful API Specification

4.1 Introduction to the Open Cloud Computing Interface (OCCI) API

Please check the FI-WARE Open Specifications Legal Notice (Intel) to understand the rights to use this FI-WARE Open Specifications.

This specification is based on OCCI, published by OGF. Check the OCCI Legal Notice to understand the relevant usage rights.

OCCI is an open, standardised, extendable, RESTful interface designed to manage arbitrary resources. Whilst OCCI is used in FI-WARE to manage virtual machines, OCCI can also be used to manage storage resources, network resources, and even software services.

The OCCI walkthrough presentation provides a useful introduction to OCCI. Complete technical details of the standard are available in the formal specification documents:

- OGF183: “Open Cloud Computing Interface - Core” - Describes the formal definition of the the OCCI Core Model
- OGF184: “Open Cloud Computing Interface - Infrastructure” - Describes the OCCI Infrastructure extension for the IaaS domain.The document defines additional resource types, their attributes and the actions that can be taken on each resource type.
- OGF185: “Open Cloud Computing Interface - RESTful HTTP Rendering” - Defines how to interact with the OCCI Core Model using the RESTful OCCI API. The document defines how the OCCI Core Model can be communicated and thus serialised using the HTTP protocol.

Any observations, feedback or questions on OCCI can be posted to the OCCI community via the OCCI Mailing list.

4.1.1 OCCI API Core

The OCCI core model forms the basis of its type system. Types made available by OCCI implementations are defined by the Category construct. There are two specialised forms of the `Category` construct: `Kind` and `Mixin`. `Kind` defines the basic capabilities (attributes and functionality) of a type of resource. `Mixin` defines a means to further modify and extend a particular `Kind`’s capabilities. `Action`’s define the executable functionality (e.g. methods) of either. Categories are self-descriptive, they can be discovered through a Query interface. The Query Interface allows for all service provider supported Categories to be discovered and described. The core model forms the basis for the Infrastructure as a Service specification. This specification extends the core model and it in itself provides an example of how one can extend the OCCI model to suit particular needs.

4.1.2 Intended Audience

This specification is intended for both software developers and Cloud Providers. For the former, this document provides a full specification of how to interoperate with Cloud Platforms that implement the OCCI API. For the latter, this specification describes the interface to be provided in order for clients to interoperate with the Cloud Platform to provide the described functionalities. To use this information, the reader should firstly have a general understanding of the Data Centre Resource Manager Generic Enabler service. The reader should also be familiar with:

- ReSTful web services.
- HTTP 1.1.
- Text and JSON data serialisation formats.

4.1.3 API Change History

This version of the DCRM Open RESTful API Specification replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
Apr 30, 2012	• Initial Version
Sept 4, 2012	• Updated Templated sections
April 29, 2013	• Restructured content

4.1.4 How to Read This Document

It is assumed that the reader is familiar with the RESTful architectural style. This document uses the following notation:

- A bold, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).
- An italic font is used to represent document titles or some other kind of special text, e.g., URI.

For a description of some of the terms used within this document, see the Data Center Resource Management Architecture.

4.1.5 Additional Resources

- OCCI walkthrough
- InfoQ.com article on OCCI
- OGF site
- OCCI web site
- OCCI mailing list
- FIware Cloud Hosting Product Vision

4.2 General OCCI API Information

4.2.1 Resources Summary

To understand what infrastructural resources are specified by OCCI see GFD184. To understand the core resource model within OCCI see GFD183 Sections 3 and 4. As providers of the API may wish to use different URI schemes and hierarchies, OCCI does not prescribe a static configuration. To understand how the hierarchy can be discovered through OCCI's query interface see GFD183. The OCCI Query Interface is advertised using the well-known location IETF RFC5785. OCCI is highly extensible and various extension mechanisms and extension points are documented in GFD183 Section 4.6.

4.2.2 Authentication

Each HTTP request against OCCI requires the inclusion of specific authentication credentials. The specific implementation of this API may support multiple authentication schemes (OAuth, Basic Auth, Token) as such mechanisms are orthogonal to the API. This will be determined by the specific provider that implements the GE. Please contact with it to determine the best way to authenticate against this API. Remember that some authentication schemes may require that the API operate using SSL over HTTP (HTTPS). See GFD185 for further details.

4.2.3 Representation Format

The OCCI API supports plain text:

- `text/plain` - this is the default content type
- `text/occi`
- `text/uri-list`

See GFD185 for precise Augmented Backus-Naur Form (ABNF) specifications of these content types. JSON support is currently being developed. The request format is specified using the Content-Type header and is required for operations that have a request body. The response format can be specified in requests using the standard HTTP `Accept` header.

4.2.4 Representation Transport

Resource representation is transmitted between client and server by using HTTP 1.1 protocol, as defined by IETF RFC–2616. Each time an HTTP request contains payload, a Content-Type header shall be used to specify the MIME type of wrapped representation. In addition, both client and server may use as many HTTP headers as they consider necessary. For OCCI specifics see GFD185.

4.2.5 Resource Identification

To understand how resource identification is achieved in OCCI, see GFD183.

4.2.6 Linking Resources

Resources often lead to refer to other resources. To review how this is done in OCCI (using the `Link` construct), see GFD183 Section 4.5.3 and GFD184 on their application to infrastructural resources.

4.2.7 Pagination of Resource Collections

This is being developed through the JSON specification.

4.2.8 API Limits

This is implementation specific. In the **OpenStack** OCCI implementation, within the `nova-api` module, a `WSGI` middleware is included to enable rate limiting. The OCCI API can also use this and as such be part of its middleware pipeline as is done for all OpenStack API services. This is configured through the `/etc/nova/api-paste.ini` file.

4.2.9 Versions

To get the version that an OCCI service is using, see GFD185. Associated semantics with version mismatches etc are also dealt with there.

4.2.10 Extensibility

See GFD183 and GFD184. It should be noted that GFD184 is an extension itself and so offers a comprehensive example OCCI extensibility. The CompatibleOne project uses OCCI as their core model. How they extend the OCCI core model can be seen in the CORDS reference manual.

4.2.11 Faults

For the full set of faults that an OCCI service can return please also see GFD185. The most common error codes are listed here:

HTTP Return code	Description
200 OK	Indicates that the request was successful. The response MUST contain the created resource instance's representation.
201 OK	Indicates that the request was successful. The response MUST contain a HTTP Location header to the newly created resource instance.
400 Bad Request	Used to signal parsing errors or missing information (e.g. an attribute that is required is not supplied in the request).
401 Unauthorized	The client does not have the required permissions/credentials.
404 Not Found	Used to signal that the request had information (e.g. a kind, mixin,

	action, attribute, location) that was unknown to the service and so not found.
500 Internal Server Error	The state before the request should be maintained in such an error condition. The implementation MUST roll-back any partial changes made during the erroneous execution.

4.3 API Operations

See GFD183, GFD184 and GFD185 for all OCCI specifications on specific aspects. The FI-WARE programmer guide will also provide examples of how to use the API. Only operations related to OCCI extensions will be discussed and described here. In the sections below a summary of the extension, its purpose and example usage will be given. The example usages will use the `text/occi` content type for brevity.

4.4 Architectural Operation Mapping

These operations are listed in the Data Centre Resource Management architectural specification. A high level mapping of these operations to OCCI requests are presented below. The operations described here are all listed with mandatory inputs.

- `createVirtualServer`
  - **Description**: provision a new virtual server with the given properties (virtual hardware, policy parameters, access, etc). Returns unique ID of the virtual server.
  - **OCCI Mapping**: A HTTP POST using the `compute`, `OsTemplate` and `ResourceTemplate` categories.
- `destroyVirtualServer`
  - **Description**: remove a virtual server
  - **OCCI Mapping**: A HTTP DELETE issued against the HTTP URL identifying the virtual server instance.
- `powerOnVirtualServer`
  - **Description**: turn on a virtual server
  - **OCCI Mapping**: A HTTP POST using the `start` action.
- `createVirtualServer`
  - **Description**: provision a new virtual server with the given properties (virtual hardware, policy parameters, access, etc). Returns unique ID of the virtual server.
  - **OCCI Mapping**: A HTTP POST using the `compute`, `OsTemplate` and `ResourceTemplate` categories.
- `destroyVirtualServer`
  - **Description**: remove a virtual server
  - **OCCI Mapping**: A HTTP DELETE issued against the HTTP URL identifying the virtual server instance.
- `powerOnVirtualServer`
  - **Description**: turn on a virtual server
  - **OCCI Mapping**: A HTTP POST using the `start` action.

- `powerOffVirtualServer`
  - **Description**: turn off a virtual server
  - **OCCI Mapping**: A HTTP POST using the `stop` action, parameterised with the method=`acpioff`.
- `restartVirtualServer`
  - **Description**: restart a virtual server
  - **OCCI Mapping**: A HTTP POST using the `restart` action.
- `shutdownVirtualServer`
  - **Description**: shut down a virtual server (note: the ability to perform this operation on the fly depends on the capabilities of the underlying virtualisation platform)
  - **OCCI Mapping**: A HTTP POST using the `stop` action, parameterised with the method=`graceful`.
- `resizeVirtualServer`
  - **Description**: change the virtual hardware allocation for a virtual server (note: the types of resources for which the reconfiguration can be done on the fly depends on the capabilities of the underlying virtualization platform)
  - **OCCI Mapping**: A HTTP POST (partial update) against the the HTTP URL identifying the virtual server instance and with the `ResourceTemplate` specified indicating the new hardware allocation profile.
- `getVirtualServerDetails`
  - **Description**: returns details of a virtual server (virtual hardware specification, state, associated policy parameters, access details, etc)
  - **OCCI Mapping**: A HTTP GET against the the HTTP URL identifying the virtual server instance.
- `createVirtualDisk`
  - **Description**: provision a new virtual disk with the given properties (size, capabilities, etc). Returns unique ID of the virtual disk.
  - **OCCI Mapping**: A HTTP POST using the `storage` category.
- `destroyVirtualDisk`
  - **Description**: remove a virtual disk
  - **OCCI Mapping**: A HTTP DELETE issued against the HTTP URL identifying the virtual disk instance.
- `attachVirtualDisk`
  - **Description**: attach a given virtual disk to a given virtual server (note: the ability to perform this operation on the fly depends on the capabilities of the underlying virtualization platform)
  - **OCCI Mapping**: A HTTP POST using the `storagelink` link category.
- `detachVirtualDisk`
  - **Description**: detach a given virtual disk from a given virtual server (note: the ability to perform this operation on the fly depends on the capabilities of the underlying virtualization platform)

- **OCCI Mapping**: A HTTP DELETE issued against the HTTP URL identifying the storage link associating the virtual server with the virtual disk instances.

- `getVirtualDiskDetails`
  - **Description:** return details of a given virtual disk (ID, capabilities, attachment details, etc)
  - **OCCI Mapping:** A HTTP GET against the the HTTP URL identifying the virtual disk instance.

- `createVirtualNetwork`
  - **Description**: provision a new virtual network with the given properties (e.g., VLAN ID, capabilities, etc). Returns unique ID of the virtual network.
  - **OCCI Mapping**: A HTTP POST using the `network` and `ipnetwork` categories.

- `destroyVirtualNetwork`
  - **Description**: remove a virtual network
  - **OCCI Mapping**: A HTTP DELETE issued against the HTTP URL identifying the virtual network instance.

- `attachVirtualServerToNetwork`
  - **Description**: attach a virtual network interface of a given virtual server to a given virtual network
  - **OCCI Mapping**: A HTTP POST using the `networkinterface` link category and `ipnetworkinterface` mixin.

- `detachVirtualServerFromNetwork`
  - **Description**: detach a virtual network interface of a given virtual server from a given virtual network
  - **OCCI Mapping**: A HTTP DELETE issued against the HTTP URL identifying the network link associating the virtual server with the virtual network instances.

- `getVirtualNetworkDetails`
  - **Description**: return details of a given virtual network (ID, capabilities, attachment details, etc)
  - **OCCI Mapping**: A HTTP GET against the the HTTP URL identifying the virtual network instance.

- `listVirtualImages`
  - **Description**: return a list of all available virtual images (visible by the authenticated user)
  - **OCCI Mapping**: HTTP GET on the Query Interface

- `queryVirtualImages`
  - **Description**: return a list of available virtual images, filtered by given query criteria.
  - **OCCI Mapping**: HTTP GET on the Query Interface with a categories filter applied.

- `getVirtualImageDetails`
  - **Description**: return details of a virtual image (type, size, creation details, etc)

- **OCCI Mapping**: image details are present in all HTTP GETs on the Query Interface.

- `uploadVirtualImage`
  - **Description**: upload a new virtual image into the virtual image repository
  - **OCCI Mapping**: HTTP POST against the query interface with the `OsTemplate` category supplied (Note not currently supported in OCCI V1.1).

4.4.1 OCCI Kind Extensions

These extensions extend the OCCI core model `Kind` construct (See Core Model Section 4.4.2). As such they can have full create, retrieve, update and delete functionality.

4.4.2 Security Group Extension

Security Rules are associated with a Collection defined by a Mixin. Rules can be removed likewise. The semantics of this are set out in the OCCI Core Model document Section 4.6.3, where user-supplied `Mixin`s are described. This is a candidate extension to be offered to OCCI for inclusion in main spec.

- term: `USER_DEFINED`
- scheme: `USER_DEFINED`
- rel: `http://schemas.ogf.org/occi/infrastructure/security#group`

Example Text Rendering:

```
Category: my_grp; scheme="http://www.mystuff.org/sec#";
class="mixin";
rel="http://schemas.ogf.org/occi/infrastructure/security#group
```

4.4.3 Security Rule Extension

Using this extension network security ingress rules can be applied to VMs. The definition of such rules are a requirement in satisfying NIST cloud computing SAJACC use cases. EC2, OpenStack and OpenNebula all share this functionality. This is a candidate extension to be offered to OCCI for inclusion in main spec.

- term: `rule`
- scheme: `http://schemas.openstack.org/occi/infrastructure/network/security#`
- attributes:
  - `occi.network.security.protocol`: enumeration of strings, range: tcp, ip, icmp
  - `occi.network.security.to`: integer, range: 0–65535
  - `occi.network.security.from`: integer, range: 0–65535
  - `occi.network.security.range`: CIDR string

Example text rendering:

```
Category: rule;
scheme="http://schemas.openstack.org/occi/infrastructure/netwo
rk/security#"; class="kind"

X-OCCI-Attribute: occi.network.security.protocol = "tcp"

X-OCCI-Attribute: occi.network.security.to = 22

X-OCCI-Attribute: occi.network.security.from = 22

X-OCCI-Attribute: occi.network.security.range = "0.0.0.0/24"
```

4.4.4 VNC Console Extension

This extension is used only to relay information back to the client about console endpoint information. It only supports HTTP `GET` or retrieve functionality.

- term: `vnc_console`
- scheme: http://schemas.openstack.org/occi/infrastructure/compute#
- attributes:
  - `org.openstack.compute.console.vnc`: URI string

Example text rendering:

```
Category: vnc_console;
scheme="http://schemas.openstack.org/occi/infrastructure/compu
te#"; class="mixin"

X-OCCI-Attribute:
org.openstack.compute.console.vnc="http://10.234.54.23:5901"
```

4.4.5 SSH Console Extension

This extension is used only to relay information back to the client about console endpoint information. It only supports HTTP `GET` or retrieve functionality.

- term: `ssh_console`
- scheme: http://schemas.openstack.org/occi/infrastructure/compute#
- attributes:
  - `org.openstack.compute.console.ssh`: URI string

Example text rendering:

```
Category: ssh_console;
scheme="http://schemas.openstack.org/occi/infrastructure/compu
te#"; class="mixin"

X-OCCI-Attribute:
org.openstack.compute.console.ssh="ssh://142.123.45.42:22"
```

4.4.6 OCCI Mixin Extensions

These extensions extend the OCCI core model `Mixin` construct (See Core Model Section 4.4.3). As such they can only have full create and delete functionality.

4.4.7 TCP - Trusted Compute Pool Extension

This mixin extension signals to the backend implementation that the requested VM to be provisioned should be placed on hardware that has TCP capabilities.

- term: `tcp`
- scheme: `http://schemas.fi-ware.eu/occi/infrastructure/compute#`
- attributes:
  - `eu.fi-ware.compute.tcp`: boolean

Example text rendering:

```
Category: floating_ip;
scheme="http://schemas.openstack.org/instance/network#";
class="mixin"

X-OCCI-Attribute:
org.openstack.credentials.publickey.name="my_pub_key"
```

4.4.8 Administrative Password Extension

This mixin extension allows for authenticated users to specify the administrative password of the VM.

- term: `admin_pwd`
- scheme: `http://schemas.openstack.org/instance/credentials#`
- attributes:
  - `org.openstack.credentials.admin_pwd`: string

Example text rendering:

```
Category: admin_pwd;
scheme="http://schemas.openstack.org/instance/credentials#";
class="mixin"

X-OCCI-Attribute:
org.openstack.credentials.admin_pwd="pussinboots"
```

4.4.9 Access IP Extension

This mixin extension allows for authenticated users to specify an additional access IP and is used as a means to route from that IP to the target VM's internal IP.

- term: `access_ip`
- scheme: `http://schemas.openstack.org/instance/network#`
- attributes:
  - `org.openstack.network.access.ip`: string

- `org.openstack.network.access.version`: enumeration of strings, range: ipv4, ipv6

Example text rendering:

```
Category: access_ip;
scheme="http://schemas.openstack.org/instance/network#";
class="mixin"

X-OCCI-Attribute:
org.openstack.network.access.ip="188.12.132.58"

X-OCCI-Attribute: org.openstack.network.access.version="ipv4"
```

4.4.10 Key Pair Extension

This mixin extension allows for authenticated users to set a SSH public key for passwordless authentication (via SSH) against the provisioned VM.

- term: `public_key`
- scheme: http://schemas.openstack.org/instance/credentials#
- attributes:
  - `org.openstack.credentials.publickey.name`: string
  - `org.openstack.credentials.publickey.data`: string

Example text rendering:

```
Category: floating_ip;
scheme="http://schemas.openstack.org/instance/network#";
class="mixin"

X-OCCI-Attribute:
org.openstack.credentials.publickey.name="my_pub_key"

X-OCCI-Attribute:
org.openstack.credentials.publickey.datap="ssh-dss
AAAAB3NzaC1kc3MAAACBAML/WvvbXBF1DWTou0ISmDJGo4OsfU6qW94vyZuiM0
5LbChnPrBVlzv1p2gWw/PlC6jkD/aNaMJwbuQoS/1J2G6cHgcX6oeNM+mo/BtH
KZGUmMjvKeeNuP8BqB0YX4OMSQ5E0GLfFmLC0P4QpjZ0jiuE4K8AM3Ht75p9XC
tdTpF3AAAAFQC6E882UjUeZGbv8zja97gfFbLTGwAAAIBERL187u5siYAf2Cq4
pBZ3YSoeWjrn68bU5h9DWXlRnMb1G0waPhh4MCbYfKefqstu/mwuq9w2gIGYuz
Y2NBHX2BYDtC2cfKh0v1SzFv1U6UtOg9WT34eNuHNkCrgE1p+JuSaZQ8rO864G
abxwSWxMQe53DwpaznCzcuK6KL4VGQAAAIBybl3bv7S1UsW51LmnasshlVVaMF
5i1pS5qFYjK829"
```

4.4.11 Floating IP Extension

This mixin extension allows for authenticated users know what the allocated floating IP address of a VM is.

- term: `floating_ip`
- scheme: http://schemas.openstack.org/instance/network#
- attributes:
  - `org.openstack.network.floating.ip`: valid IP address rendered as string

Example text rendering:

```
Category: floating_ip;
scheme="http://schemas.openstack.org/instance/network#";
class="mixin"

X-OCCI-Attribute:
org.openstack.network.floating.ip="172.156.89.12"
```

4.4.12 OCCI Action Extensions

These extensions extend the OCCI core model `Action` construct (See Core Model Section 4.5.4). As such they only implement the executive aspect of the construct (via `HTTP`POST`).

4.4.13 Change Password Action Extension

This action allows authenticated users to change the administrative user's password.

- term: `chg_pwd`
- scheme: `http://schemas.openstack.org/instance/action#`
- attributes:
  - `org.openstack.credentials.admin_pwd`: string

Example text rendering:

```
Category: chg_pwd;
scheme="http://schemas.openstack.org/instance/action#";
class="action"

X-OCCI-Attribute:
org.openstack.credentials.admin_pwd="new_pass"
```

4.4.14 Revert Action Extension

This action allows authenticated users to rollback a VM resize operation.

- term: `revert_resize`
- scheme: `http://schemas.openstack.org/instance/action#`
- attributes: None

Example text rendering:

```
Category: revert_resize;
scheme="http://schemas.openstack.org/instance/action#";
class="action"
```

4.4.15 Confirm Resize Action Extension

This action allows authenticated users to confirm that a VM resize operation was successful and met their needs.

- term: `confirm_resize`
- scheme: `http://schemas.openstack.org/instance/action#`

- attributes: None

Example text rendering:

```
Category: confirm_resize;
scheme="http://schemas.openstack.org/instance/action#";
class="action"
```

4.4.16 Create Image Action Extension

In order to create a bootable VM image from an existing running VM, a user can use the `create_image` Action. The resultant image MUST and will be exposed only to the owning user through the OCCI query interface.

- term: `create_image`
- scheme: http://schemas.openstack.org/instance/action#
- attributes:
  - `org.openstack.snapshot.image_name`: string

Example text rendering:

```
Category: create_image;
scheme="http://schemas.openstack.org/instance/action#";
class="action"

X-OCCI-Attribute:
org.openstack.snapshot.image_name="my_image_name"
```

4.4.17 Allocate IP Action Extension

This action allows authenticated users to allocate a floating (static) IP to a VM. The pools (pool names are obtained from the Query Interface) from which IPs can be obtained is listed in the OCCI Query Interface.

- term: `alloc_float_ip`
- scheme: http://schemas.openstack.org/instance/action#
- attributes:
  - `org.openstack.network.floating.pool`: string

Example text rendering:

```
Category: alloc_float_ip;
scheme="http://schemas.openstack.org/instance/action#";
class="action"

X-OCCI-Attribute: org.openstack.network.floating.pool="nova"
```

4.4.18 Deallocate IP Action Extension

This action allows authenticated users to deallocate a floating (static) IP to a VM and release back to the originating IP pool.

- term: `dealloc_float_ip`
- scheme: http://schemas.openstack.org/instance/action#

- attributes: None

Example text rendering:

```
Category: dealloc_float_ip;
scheme="http://schemas.openstack.org/instance/action#";
class="action"
```

5 FIWARE OpenSpecification Cloud SelfServiceInterfaces

5.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.org and similar pages in order to understand the complete context of the FI-WARE project.

5.2 Copyright

Copyright © 2013 by UPM. All Rights Reserved.

5.3 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

5.4 Overview

This specification describes the Self Service Interfaces GE which is enabler for user/admin to better interact with the cloud infrastructure. The main goal of the Self Service Interfaces is to provide a user interface and libraries as a support for the administrators and the users to easily manage their services deployed in the Fi-Ware cloud. This GE consist of several components that permit to that enable to perform the following actions: Create/Delete/Edit Image/Flavor/Project/User/Snapshots/Volumes/Containers/Objects, Create/Terminate/Reboot/Resize/Delete Instance etc.

5.4.1.1 *GE Description*

The Self Services Interfaces is divided in two parts: User Portal and Toolkit.

The **User Portal** is implemented in a form of a Web GUI following the example of the portals that today's common cloud infrastructure managers like Amazon EC2, Eucalyptus,Cloud Sigma, Rackspace, etc. have. In concrete it bases its design principles on the OpenStack Horizon Dashboard. The basic objective of the user portal is to facilitate the user of the cloud perform operations over the underlying infrastructure. This includes perform actions such as: create user, manage projects, lunch instances on a base of images, create images in the image repository, retrieve flavors from the resource, etc. Moreover the portal facilitates management of a Virtual Data centers (VDCs), Services and Service Data Centers (SDCs), PaaS management and will offer monitoring statistics of the physical and virtual machines.
The **Toolkit** is aimed for administrators and experienced users and it consist of various scripts that permit to perform the same actions the user portal does and some more advanced options.

5.4.1.2 *GE Components*

The following diagram shows the main components of the Self Service Interfaces GE, as well as their main interactions.

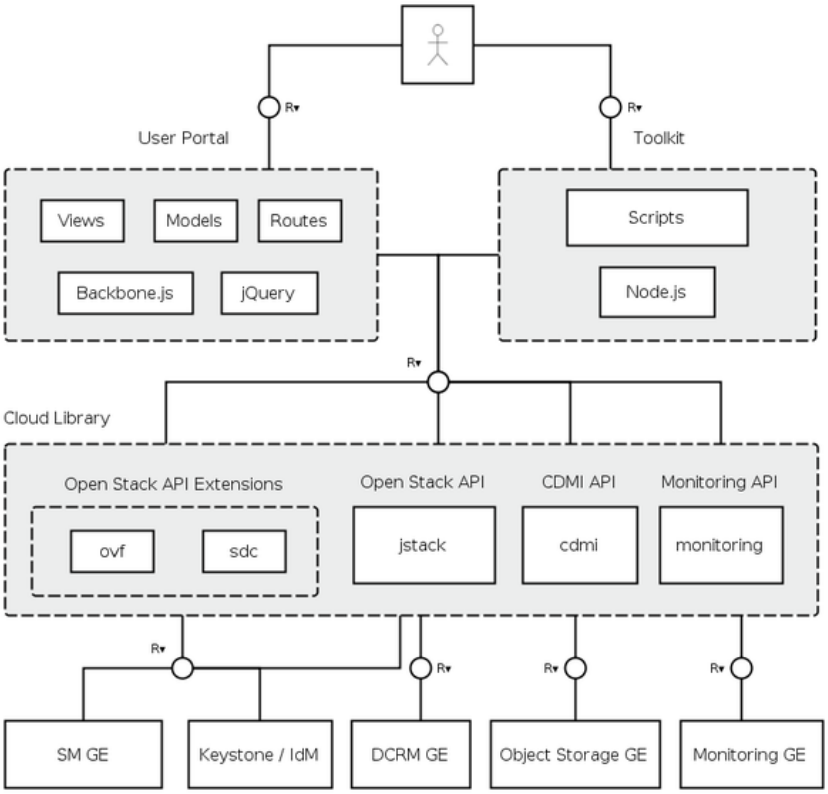


Figure: Self Service Interfaces GE

5.4.1.2.1 *User Portal*

A web client-side application implemented in JavaScript, that follows the structure as the OpenStack Horizon component, but has extended functionality to meet the requirements of the SM GE components. It is a Backbone-based application that follows the (Model-View-Controller) MVC design principles and aims at improving the user experience by using AJAX. The portal uses the underlying Cloud Library.

5.4.1.2.2 *Toolkit*

A direct command line interface to the cloud infrastructure and platform aimed for experienced users. The toolkit uses the cloud library and its different APIs to perform different calls to the SM GE, DCRM GE, Object Storage GE, Keystone/IdM GE and Monitoring GE.

5.4.1.2.3 *Cloud Library*

This library consist of sub-libraries developed in JavaScript. Each of them implement specific APIs to enable the use of the underlying GE from the User Portal or with the Toolkit.

5.5 Basic Design Principles

The **User Portal** design has the following key features:

- Client side app accessible within an HTML5 Page (no Web server is needed to interact with back-end)
- Internationalization: i18n (supports currently 4 languages)
- Responsive design: adaptable to multiple device screens (desktop, smartphone, tablet, etc)
- Customizable Object Oriented CSS
- Dynamic web app refreshed automatically as backend changes

The **Toolkit** contains Node.js based scripts as a JavaScript implementation of different APIs.
The **Cloud Library** comprises of libraries that contain various JavaScript APIS: OpenStack APIs, OpenStack API extensions, CDMI APIs and Monitoring APIs. Each of these APIs are organized in the following libraries: ovf, sdc, jstack, cdmi and the monitoring library.

5.5.1 References

[Keystone]	Open Stack Keystone. http://keystone.openstack.org, 2012
[Dashboard]	Open Stack Dashboard. http://wiki.openstack.org/OpenStackDashboard, 2012
[Glance]	Open Stack Glance image repository documentation. http://glance.openstack.org, 2012
[Amazon EC2]	Amazon Management Console. http://aws.amazon.com/console, 2012
[Eucalyptus]	Eucalyptus web-based front-end. http://open.eucalyptus.com/wiki/EucalyptusManagement_v1.5, 2012
[Rackspace]	Rackspace control panel. http://en.wikipedia.org/wiki/File:Rackspace_Cloud_control_panel_screenshot.png
[Cloud Sigma]	Cloud Sigma web interface. http://cloudsigma.com/en/platform-details/intuitive-web-interface, 2012

5.6 Re-utilised Technologies/Specifications

The Self Service Interfaces GE is based on RESTful Design Principles. The technologies and specifications used in this GE are:

- RESTful web services
- HTTP/1.1 (RFC2616)
- JSON and XML data serialization formats.

- OpenStack Compute API v2.0

5.7 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be helpful to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FI-WARE Global Terms and Definitions

- **Runtime Execution Container** (**REC**) -- a VM with all the software stack required to deploy a complete node in an application architecture. It usually comprises one or more VMs, middleware, monitoring probes, and Chef client and SDC agent to support installation and configuration.
- **Software Deployment and Configuration GE** (**SDC GE**) -- a GE in devoted to the automated installation and configuration of software on VMs through the execution of recipes in the corresponding nodes. It relies on Opscode Chef technology.
- **Software Deployment and Configuration Interface** (**SDCI**) -- the interface offered by the SDC GE to be managed by the PaaS Manager or a Cloud Portal.
- **Product Instance** (**PI**) -- an installed software in a VMs, usually referring to middleware or platform software (E.g.: Apache Tomcat, MySQL, etc.).
- **Application Component** (**AC**) -- a component (or configuration artifact) that implements usually one or more components of an application architecture. These ACs are installed on, and differentiated from, existing PIs.
- **PaaS Manager Interface** (**PMI**) -- the interface offered by the PaaS Manager GE to be used by a Cloud Portal to manage both the catalogue and the lifecycle of the platform resources and applications.

6 FIWARE OpenSpecification Cloud ObjectStorage

6.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.org and similar pages in order to understand the complete context of the FI-WARE project.

6.2 Copyright

The FI-WARE ObjectStorage Specification is Copyright © 2014 INTEL. Please note that this specification adopts the CDMI standard, which is published by and copyright SNIA.

6.3 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

6.4 Overview

Object Storage is one of the Generic Enablers within FI-WARE. It offers persistent storage for digital objects, important cloud-based functionality that has been specifically requested by Use Cases. Objects can be files, databases or other datasets which need to be archived. Objects are stored in named locations known as containers. Containers can be nested thus objects can be stored hierarchically.

Containers and objects can have Metadata associated with them, providing details of what the data represents. Similar to files in a traditional filesystem - objects in an Object store belong to a certain user (account).

The following sections provide more information on all of these topics and introduce the CDMI standard. Rather than develop an entirely new interface, this Generic Enabler is based on CDMI.

The users of the Object Storage Generic Enabler include both FI-WARE Cloud Instance Providers and FI-WARE Cloud Instance Users.

- Provider usage: Cloud Instance Providers can both provide Object Storage as a service to Cloud Instance Users, and consume the Object Storage service themselves. In terms of providing the service, the Cloud Instance Providers will require a system that demands as little maintenance as is possible. This entails that any:
    - stale data be purged,
    - deactivated accounts be removed,
    - corrupt data is replaced with a valid replica,
    - Issues are escalated to an automated service that will attempt to resolve them (if they cannot be resolved then notifications to the Provider should be sent),

- relevant statistics should be available to support inspection of the system and the User's utilisation of the system,
- additional requirements for hardware (storage capability) can be easily added to the system without any drop in service. This will allow the storage capacity to grow over time.

In terms of consuming the service themselves, the Cloud Instance Providers will want to store certain types of data such as monitoring, reporting and auditing data to support their offering. This data can be made available to the Cloud Instance Users depending on requirements. The Object storage service can also be used as a virtual machine staging area. A Cloud Instance User may upload their custom virtual machine to the Object Store from which location the provider will make it available.

- User usage: The User will use the object storage service as a means to distribute static content rather than incur the additional load of serving static content from an application. Taking this approach allows the Provider to optimize the distribution of those files. The Provider can also use this as a building block to offer further content distribution network capabilities. The User could also use the object storage service as a means to supply a customized virtual machine that only they have access to (the storage is isolated by user). This would operate in much the same way as how customized virtual machine images are supplied on services like Amazon EC2.

6.5 Basic Concepts

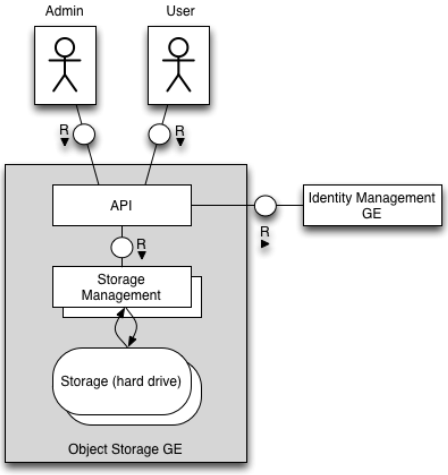
Implementations of the Object Storage Generic Enabler (GE) should provide a highly available, distributed and eventually consistent object store. The object store is a collection of objects that are structured in a simple hierarchy. The object store presents itself as a service that has multi-tenant capabilities such that the service can be offered to many users and organisations and that their data is safely partitioned. The object storage service does not have a traditional POSIX-type file system and as it has a simple hierarchical structure it really has little notion of true directory semantics. In fact, the Object Storage service allows for levels of hierarchy, where a container can reside within another container so allowing for more flexible organisation of data. Notably the Object Storage GE adopts the Cloud Data Management Interface (CDMI) specification.

The key abstract entities identified that need to be considered by this GE are:

- **Object**: opaque piece of data with associated meta-data. This directly maps to the CDMI object.
- **Container**: collection of objects with associated meta-data. This directly maps to the CDMI container. In CDMI, a container can contain many other containers. A container cannot have more than one parent container.
- **Policy**: meta data associated with an object or container that dictates the use of the data by the object storage service provider. Policies will be expressed through the meta-data facilities offered by CDMI.
- **Account**: a collection of containers assigned to a user. As the Object Storage GE uses the CDMI specification the concept of Account will map to the highest level of container.
- **User**: the actor accessing and managing the above entities through the GE's API. At a minimum the actors of end-user (human or external service) and administrator are considered. The security information related to User is managed by the Identity Management GE.

Access to and management of Object Storage entities is performed through the defined API. As standardised and open interfaces to GEs is an important aspect to consider in the specification of all GEs, the Cloud Data Management Interface (CDMI) has been adopted as the API specification for the Object Storage GE.

The overall *internal* architecture of the Object Storage GE is shown below:



The main elements in the functional block diagram are as follows:

- **Admin**, **User**: These are Actors interacting with the service. Both might have different privileges associated to their accounts.
- **Identity Management GE**: This is the entity which handles the privileges of the users.
- **API**: this entity is what exposes the interface of the Object Storage GE and allows the administrator and user user-types interact and manage their service instances.
- **Storage Management**: this is the entity that manages the resources associated with a user's service instance. Here entities such as containers, meta-data, objects and policies are managed.
- **Storage**: this is the storage device where the objects are physically persisted

6.5.1 Cloud Data Management Interface (CDMI)

CDMI is the de-facto standard for manipulating data in the cloud. Developed by the Storage Networking Industry Association (SNIA), it has now been designated by ISO as a formal international standard.

CDMI considers the management of both cloud storage systems and those of enterprise systems. It implements the SNIA's Storage Industry Resource Domain Model (SIRDM).

At the core of CDMI are the basic management operations of Create, Retrieve, Update and Delete. This functionality is exposed via a RESTful API that:

- Enables clients to discover capabilities of the object storage offering
- Manage containers and the objects that are placed within them
- Assigns and manipulates metadata to containers and objects

Meta-data is core to enabling richer management of data within CDMI. Meta-data can be associated with both system-managed objects and of course user-managed objects. Indeed, according to the specification "metadata is interpreted by the cloud

offering as data requirements that control the operation of underlying data services for that data." This focus of CDMI makes it as a very flexible and suitable specification for the Object Storage GE, especially as providing quality of service and experience is a core focus in FI-ware. It is through CDMI's meta-data facilities that various data handling policies and QoS parameters can be specified and how the abstract entity of Policy can be represented.

The core entities in the CDMI model are:

- **Capabilities**: allows a client to discover features of the CDMI standard implemented by a provider. This is required for basic CDMI compliance.
- **Object**: opaque piece of data with associated meta-data. This is required for basic CDMI compliance. This CDMI entity will allow for the representation of Object.
- **Container**: collection of objects with associated meta-data. This is required for basic CDMI compliance. This CDMI entity will allow for the representation of Container and Account.
- **Domain**: represent the concept of administrative ownership of stored data within a CDMI storage system. This is an optional entity and is not required to be implemented for basic CDMI compliance. For the first release of the Object Storage GE this feature will not be supported.
- **Queue**: is a first-in, first-out (FIFO) access when storing and retrieving data. This is an optional entity and is not required to be implemented for basic CDMI compliance. For the first release of the Object Storage GE this feature will not be supported.

6.6 Main Interactions

CDMI is a RESTful interface and all interactions are via well-known HTTP methods such as GET, PUT and DELETE. Some typical CDMI operations are introduced in this section. For complete details on how a client interacts with the CDMI interface, please refer to the CDMI specification document.

6.6.1 Creating a Container

The following request and response illustrate how a container named "mycontainer" can be created. The service responds with a HTTP 201, indicating the container was created successfully.

Request:

```
> PUT /mycontainer/HTTP/1.1
> Host: os.fi-ware.eu
> Accept: application/cdmi-container
> Content-Type: application/cdmi-container
> X-CDMI-Specification-Version: 1.0.1
```

Response:

```
< HTTP/1.1 201 Created
< Content-Type: application/cdmi-container
```

```
< X-CDMI-Specification-Version: 1.0.1
```

6.6.2 Persisting an Object

The following request and response illustrate how an object called "simpleobject.json" containing the string "Hello CDMI World!" can be stored in container named "mycontainer". The service responds with a HTTP 201, indicating the object was created successfully. Various object metadata is also returned.

Request:

```
> PUT /mycontainer/simpleobject.json HTTP/1.1
> Host: os.fi-ware.eu
> Accept: application/cdmi-object
> Content-Type: application/cdmi-object
> X-CDMI-Specification-Version: 1.0.1
> {
>     "mimetype" : "text/plain",
>     "metadata" : {
>     },
>     "value" : "Hello CDMI World!"
> }
```

Response:

```
< HTTP/1.1 201 Created
< Content-Type: application/cdmi-object
< X-CDMI-Specification-Version: 1.0.1
< {
<     "objectType" : "application/cdmi-object",
<     "objectID" : "00007E7F0010BD1CB8FF1823CF05BEE4",
<     "objectName" : "simpleobject.json",
<     "parentURI" : "/mycontainer/",
<     "parentID" : "00007E7F00102E230ED82694DAA975D2",
<     "domainURI" : "/cdmi_domains/MyDomain/",
<     "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
<     "completionStatus" : "Complete",
<     "mimetype" : "text/plain",
<     "metadata" : {
<         "cdmi_size" : "17"
<     }
```

```
< }
```

6.6.3 Retrieving an Object

The following request and response illustrate how an object called "simpleobject.json" can be retrieved from a container named "mycontainer". The service responds with a HTTP 200, indicating the object was retrieved successfully. Various object metadata and the content of the file are returned in the HTTP response body.

Request:

```
> GET /mycontainer/simpleobject.json HTTP/1.1
> Host: os.fi-ware.eu
> Accept: application/cdmi-object
> X-CDMI-Specification-Version: 1.0.1
```

Response:

```
< HTTP/1.1 200 OK
< Content-Type: application/cdmi-object
< X-CDMI-Specification-Version: 1.0.1
< {
<     "objectType": "application/cdmi-object",
<     "objectID": "00007E7F0010BD1CB8FF1823CF05BEE4",
<     "objectName": "simpleobject.json",
<     "parentURI": "/mycontainer/",
<     "parentID" : "00007E7F00102E230ED82694DAA975D2",
<     "domainURI": "/cdmi_domains/MyDomain/",
<     "capabilitiesURI": "/cdmi_capabilities/dataobject/",
<     "completionStatus": "Complete",
<     "mimetype": "text/plain",
<     "metadata": { "cdmi_size": "17"
<     },
<     "valuetransferencoding": "utf-8",
<     "valuerange": "0-16",
<     "value": "Hello CDMI World!"
< }
```

FIWARE.OpenSpecification.Details.Cloud.ObjectStorage

6.7 Re-utilised Technologies/Specifications

The ObjectStorage GE implements SNIA's CDMI standardised API.

An implementation of this Generic Enabler has been developed on top of OpenStack swift. The implementation is available at https://github.com/osaddon/cdmi.

6.8 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be help to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FIWARE Global Terms and Definitions

- **Runtime Execution Container** (**REC**) -- a VM with all the software stack required to deploy a complete node in an application architecture. It usually comprises one or more VMs, middleware, monitoring probes, and Chef client and SDC agent to support installation and configuration.
- **Software Deployment and Configuration GE** (**SDC GE**) -- a GE in devoted to the automated installation and configuration of software on VMs through the execution of recipes in the corresponding nodes. It relies on Opscode Chef technology.
- **Software Deployment and Configuration Interface** (**SDCI**) -- the interface offered by the SDC GE to be managed by the PaaS Manager or a Cloud Portal.
- **Product Instance** (**PI**) -- an installed software in a VMs, usually referring to middleware or platform software (E.g.: Apache Tomcat, MySQL, etc.).
- **Application Component** (**AC**) -- a component (or configuration artifact) that implements usually one or more components of an application architecture. These ACs are installed on, and differentiated from, existing PIs.
- **PaaS Manager Interface** (**PMI**) -- the interface offered by the PaaS Manager GE to be used by a Cloud Portal to manage both the catalogue and the lifecycle of the platform resources and applications.

7 Object Storage Open RESTful API Specification

7.1 Intended Audience

This specification is intended for both software developers and reimplementors of this API. For the former, this document provides a full specification of how to interoperate with Object Storage Platforms that implement the CDMI API. For the latter, this specification indicates the interface to be provided to allow clients interoperate with the Object Storage Platform. To use this information, the reader should firstly have a general understanding of the Object Storage Generic Enabler service. You should also be familiar with:

- RESTful web services.
- HTTP 1.1.
- JSON data serialisation formats.

7.2 Introduction to the Cloud Data Management Interface (CDMI) API

This specification is based on CDMI published by SNIA Cloud Storage TWG. Please, refer to SNIA IP Policy to understand the relevant usage rights.

7.2.1 CDMI API Core

At the core of CDMI are the basic management operations of Create, Retrieve, Update and Delete. The key CDMI entities that one can possibly interact with through CDMI are:

- **Capabilities**: allows a client to discover features of the CDMI standard is implemented by a provider. This is required for basic CDMI compliance.
- **Object**: opaque piece of data with associated meta-data. This is required for basic CDMI compliance. This CDMI entity will allow for the representation of Object.
- **Container**: collection of objects with associated meta-data. This is required for basic CDMI compliance. This CDMI entity will allow for the representation of Container and Account.
- **Domain**: represent the concept of administrative ownership of stored data within a CDMI storage system. This is an optional entity and is not required to be implemented for basic CDMI compliance. For the first release of the Object Storage GE this feature will not be supported.
- **Queue**: is a first-in, first-out (FIFO) access when storing and retrieving data. This is an optional entity and is not required to be implemented for basic CDMI compliance. For the first release of the Object Storage GE this feature will not be supported.

More information on these entities can be found in Sections 12, 8, 9, 10 and 11 respectively in the CDMI specification

7.2.2 API Change History

This version of the Object Storage Open RESTful API Specification replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
Apr 30, 2012	• Initial Version
Sept 4, 2012	• Updated Templated sections
April 29, 2013	• Restructured content and updated API Operations and implementation details

7.2.3 How to Read This Document

It is assumed that the reader is familiar with the RESTful architectural style. This document uses the following notation:

- A bold, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).
- An italic font is used to represent document titles or some other kind of special text, e.g., URI.

For a description of some terms used along this document, see the Object Storage Architecture.

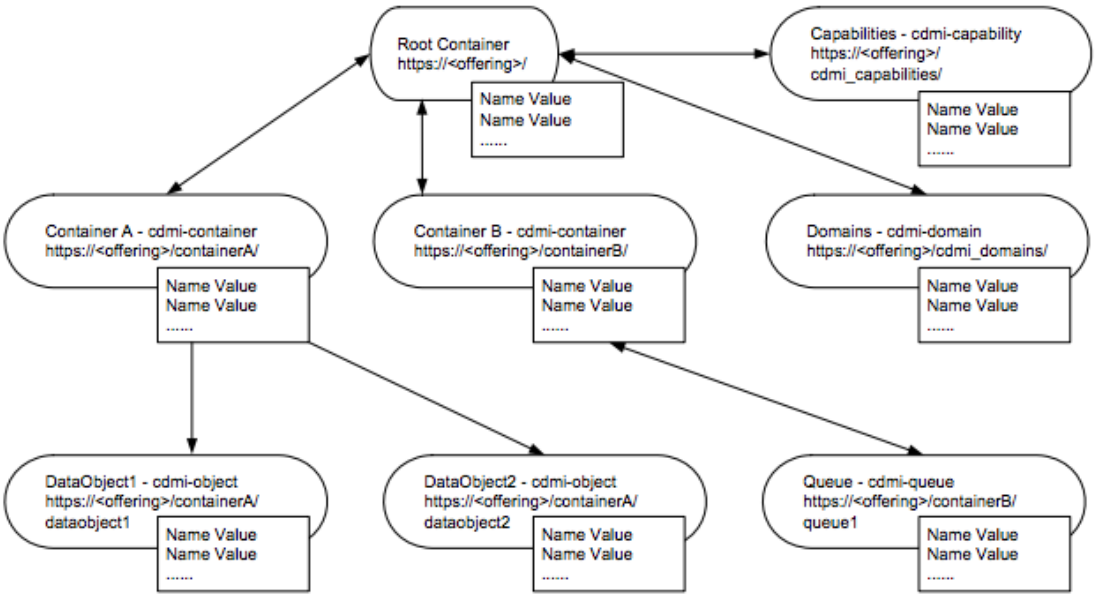
7.2.4 Additional Resources

You can download the most current version of this document from the FIWARE API specification website at https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/Summary_of_FI-WARE_Open_Specifications. For more details about the Cloud hosting architecture in which this Generic Enabler fits, please refer to high level cloud hosting architecture description. Related documents, including an introduction to the overall FI-WARE architecture, are available at the same site.

7.3 General CDMI API Information

7.3.1 Resources Summary

Graphically the main CDMI entities as listed above are related to each other as is shown in the image below.



For the purposes of data storage a CDMI Client only needs to know about container and data objects. CDMI functionality is exposed via a RESTful API that:

- Enables clients to discover capabilities of the object storage offering
- Manage containers and the objects that are placed within them
- Assigns and manipulates metadata to containers and objects

Meta-data is core to enabling richer management of data within CDMI. Meta-data can be associated with both system-managed objects and of course user-managed objects. Indeed, according to the specification “metadata is interpreted by the cloud offering as data requirements that control the operation of underlying data services for that data.” This focus of CDMI makes it as a very flexible and suitable specification for the Object Storage GE, especially as providing quality of service and experience is a core focus in FI-ware. It is through CDMI’s meta-data facilities that various data handling policies and QoS parameters can be specified and how the abstract entity of Policy can be represented.

7.3.2 Authentication

Each HTTP request against the OCCI requires the inclusion of specific authentication credentials. The specific implementation of this API may support multiple authentication schemes (OAuth, Basic Auth, Token) as such mechanisms are orthogonal to the API. This will be determined by the specific provider that implements the GE. Please contact it to determine the best way to authenticate against this API. Remember that some authentication schemes may require that the API operate using SSL over HTTP (HTTPS). See Section 5.12 and Annex A in the CDMI specification for more information.

7.3.3 Representation Format

The CDMI API represents its model in the JSON format. The CDMI specification details this further through examples.

7.3.4 Representation Transport

Resource representation is transmitted between clients and servers by using the HTTP 1.1 protocol, as defined by IETF RFC–2616. Each time a HTTP request contains payload, a Content-Type header is used to specify the MIME type of wrapped representation. In addition, both client and server may use as many HTTP headers as they consider necessary.

7.3.5 Resource Identification

Resource identification in CDMI is achieved using the defined Object ID that is detailed in Section 5.10 and 5.11

7.3.6 Versions

The CDMI specification uses the `X-CDMI-Specification-Version` header to relay the supported version of the CDMI implementation.

7.3.7 Faults

See the CDMI Specification Section 7 for details on Status Codes and Faults.

7.4 API Operations

Please refer to the CDMI specification for definitive details of all CDMI API operations and comprehensive examples. There are specific document sections dedicated to each CDMI object as listed below.

- CDMI Specification Section 8: Data Object Resource Operations
- CDMI Specification Section 9: Container Object Resource Operations
- CDMI Specification Section 10: Domain Object Resource Operations
- CDMI Specification Section 11: Queue Object Resource Operations
- CDMI Specification Section 12: Capability Object Resource Operations

Rather than duplicate documentation of the CDMI standard, a summary of some of the key CDMI operations is presented below.

7.4.1 discoverCapabilities

Allows the technical capabilities and installed features of a CDMI deployment to be discovered at runtime.

Verb	URI	Description
GET	/cdmi_capabilities/	Returns details of capabilities of the CDMI installation.

Normal Response Code(s): OK (200)

Error Response Code(s): badRequest (400), unauthorized (401), forbidden (403), notFound(404), notAcceptable(406) Example discoverCapabilitiesResponse:

```
HTTP/1.1 200 OK
Content-Type: application/cdmi-capability
X-CDMI-Specification-Version: 1.0.1
{
"objectType" : "application/cdmi-capability",
"objectID" : "00007E7F0010CEC234AD9E3EBFE9531D",
"objectName" : "cdmi_capabilities/",
"parentURI" : "/",
"parentID" : "00007E7F0010DCECC805FB6D195DDBCB",
"capabilities" : {
"cdmi_domains" : "true",
"cdmi_queues" : "true",
"cdmi_query" : "true",
"cdmi_metadata_maxsize" : "4096",
"cdmi_metadata_maxitems" : "1024",
"cdmi_size" : "true",
"cdmi_list_children" : "true",
    },
...
}
```

7.4.2 createContainer

Creates a container in which other containers and objects can be stored.

Verb	URI	Description
PUT	/{container name}/	Creates a container named {container name}.

Normal Response Code(s): created (201), accepted (202)

Error Response Code(s): badRequest (400), unauthorized (401), forbidden (403), notFound(404), conflict (409)

Example createContainer response:

```
HTTP/1.1 201 Created
Content-Type: application/cdmi-container
X-CDMI-Specification-Version: 1.0.1
{
"objectType" : "application/cdmi-container",
"objectID" : "00007E7F00102E230ED82694DAA975D2",
"objectName" : "MyContainer/",
```

```
"parentURI" : "/",
"parentID" : "00007E7F0010128E42D87EE34F5A6560",
"domainURI" : "/cdmi_domains/MyDomain/",
"capabilitiesURI" : "/cdmi_capabilities/container/",
"completionStatus" : "Complete",
"metadata" : {
    "cdmi_size" : "0"
    },
"childrenrange" : "",
"children" : [
]
}
```

7.4.3 createObject

Stores a binary object in the specified location.

Verb	URI	Description
PUT	/{container name}/{object name}	Creates an object named {object name} in the container named {container name}. The content of the object is supplied in the request body.

Normal Response Code(s): created (201), accepted (202)

Error Response Code(s): badRequest (400), unauthorized (401), forbidden (403), notFound(404), conflict (409)

Example createObject Request:

```
PUT /MyContainer/MyDataObject.txt HTTP/1.1
Host: cloud.example.com
Accept: application/cdmi-object
Content-Type: application/cdmi-object
X-CDMI-Specification-Version: 1.0.1
{
    "mimetype" : "text/plain",
    "metadata" : {
    },
    "value" : "Hello CDMI World!"
}
```

Example createObject Response:

```
HTTP/1.1 201 Created
Content-Type: application/cdmi-object
X-CDMI-Specification-Version: 1.0.1
{
"objectType" : "application/cdmi-object",
"objectID" : "00007E7F0010BD1CB8FF1823CF05BEE4",
"objectName" : "MyDataObject.txt",
"parentURI" : "/MyContainer/",
"parentID" : "00007E7F00102E230ED82694DAA975D2",
"domainURI" : "/cdmi_domains/MyDomain/",
"capabilitiesURI" : "/cdmi_capabilities/dataobject/",
"completionStatus" : "Complete",
"mimetype" : "text/plain",
"metadata" : {
    "cdmi_size" : "17"
    }
}
```

7.4.4 listContainerContents

Returns a list of the contents of a container.

Verb	URI	Description
GET	/{container name}/	Lists the contents of the container named {container name}.

Normal Response Code(s): created (201), found (302)

Error Response Code(s): badRequest (400), unauthorized (401), forbidden (403), notFound(404), notAcceptable (406)

Example listContainerContents Response:

```
HTTP/1.1 200 OK
Content-Type: application/cdmi-container
X-CDMI-Specification-Version: 1.0.1
{
"objectType" : "application/cdmi-container",
"objectID" : "00007E7F00102E230ED82694DAA975D2",
"objectName" : "MyContainer/",
"parentURI" : "/",
"parentID" : "00007E7F0010128E42D87EE34F5A6560",
"domainURI" : "/cdmi_domains/MyDomain/",
```

```
"capabilitiesURI" : "/cdmi_capabilities/container/",
"completionStatus" : "Complete",
"metadata" : {
    "cdmi_size" : "83"
    },
"childrenrange" : "0-0",
"children" : [
    "MyDataObject.txt"
    ]
}
```

7.4.5 readObjectContents

Retrieves a specified object from the storage system.

Verb	URI	Description
GET	/{container name}/{object name}	Returns the contents of the object named {object name} in the container named {container name}. The contents are returned in the response body.

Normal Response Code(s): ok (200), accepted (202), found (302)

Error Response Code(s): badRequest (400), unauthorized (401), forbidden (403), notFound(404), notAcceptable (406)

Example readObjectContents Response:

```
HTTP/1.1 200 OK
Content-Type: application/cdmi-object
X-CDMI-Specification-Version: 1.0.1
{
"objectType": "application/cdmi-object",
"objectID": "00007E7F0010BD1CB8FF1823CF05BEE4",
"objectName": "MyDataObject.txt",
"parentURI": "/MyContainer/",
"parentID" : "00007E7F00102E230ED82694DAA975D2",
"domainURI": "/cdmi_domains/MyDomain/",
"capabilitiesURI": "/cdmi_capabilities/dataobject/",
"completionStatus": "Complete",
"mimetype": "text/plain",
"metadata": { "cdmi_size": "17"
},
```

```
"valuetransferencoding": "utf-8",
"valuerange": "0-16",
"value": "Hello CDMI World!"
}
```

7.4.6 deleteObject

Deletes a specified object from the storage system.

Verb	URI	Description
DELETE	/{container name}/{object name}	Deletes the object named {object name} in the container named {container name}.

Normal Response Code(s): noContent (204)

Error Response Code(s): badRequest (400), unauthorized (401), forbidden (403), notFound(404), conflict (409)

Example deleteObject Response:

```
HTTP/1.1 204 No Content
```

7.4.7 Implementation Specific Caveats

The current CDMI implementation for OpenStack Swift offered by FI-WARE implements all basic elements of the CDMI interface. At the time of writing, **capabilities**, **container** and **object** object types are all supported. Containers can be nested and the retrieval of a limited sub-section of an object is also supported.

8 FIWARE OpenSpecification Cloud SDC

8.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.org and similar pages in order to understand the complete context of the FI-WARE project.

8.2 Copyright

Copyright © 2012 by Telefónica I+D. All Rights Reserved.

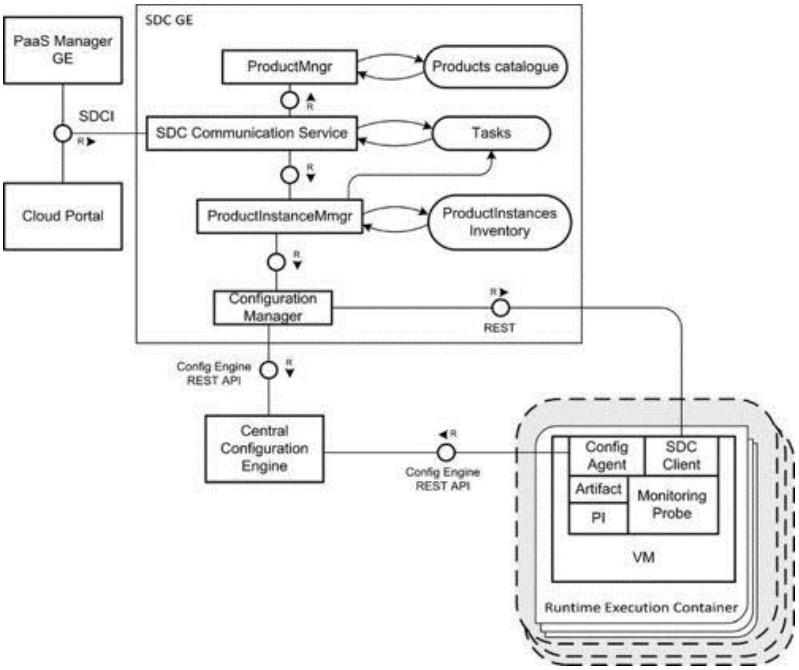
8.3 Legal Notice

Please check the following FI-WARE Open Specifications Legal Notice to understand the rights to use these specifications.

8.4 Overview

This specification describes the Software Deployment and Configuration (SDC) GE, which is the key enabler used to support automated deployment (installation and configuration) of software on running virtual machines. As part of the complete process of deployment of applications, the aim of SDC GE is to deploy software product instances upon request of the user using the SDC API or through the Cloud Portal, which in turn uses the PaaS Manager GE. After that, users will be able to deploy artifacts, that are part of the application, on top of the deployed product instances

The following diagram shows the main components of the Service Deployment and Configuration GE (SDC GE)



SDC architecture specification

In order to deploy product instances and artifacts, the SDC GE operates on SDC-enabled virtual machines (also referred as Runtime Execution Containers, RECs) that include a preinstalled SDC client. The SDC GE relies on the existence of a Configuration Management Engine, like the Opscode Chef or Puppet system integration frameworks, to automate software remote installation processes. Those Configuration Management engines are used in a client/server mode: a central Configuration Management Engine manages all the nodes (RECs), keeping information about their location and the operations that have to be executed on every node in a given time. These operations are described in form of recipes that describe how the software to be placed on a node (such as Apache Tomcat, MySQL, or HAProxy) has to be deployed. They describe a series of resources that should be configured to start running in a particular state: packages that should be installed, services that should be running, or files that should be copied. The Configuration Management Engine makes sure each resource is properly configured, and provides a safe, flexible, easily-repeatable mechanism that ensures all the nodes are always running exactly the way they were expected. Recipes belonging to a single product are aggregated in a cookbook for that product, which is stored and managed from the central Configuration Management Engine. Elements in the Product Catalogue must be synchronized with the cookbooks that are available in the central Configuration Management Engine.

All the available recipes needed for the management of a given product are aggregated in a cookbook following a naming convention. Every product supported by the SDC must have a cookbook with all the required recipes. Some instructions about how to build cookbooks and recipes can be found in Recipes for GEs.

8.4.1 Target Usage

The SDC GE automates the (un)-installation, configuration and management of product instances on a REC and the installation, configuration and management of artifacts on a previously installed product instance. The primary usage of the SDC GE is to provide an execution layer to be used by the PaaS Manager GE which ultimately deals with all the processes associated to software management. Nevertheless, the recipes managed by the SDC GE can be easily defined for complete products. Therefore, the SDC GE can be used for the deployment and configuration of complete applications on virtual machines, using the SDC GE APIs directly or through an on-demand application deployment portal.

8.5 Main Concepts

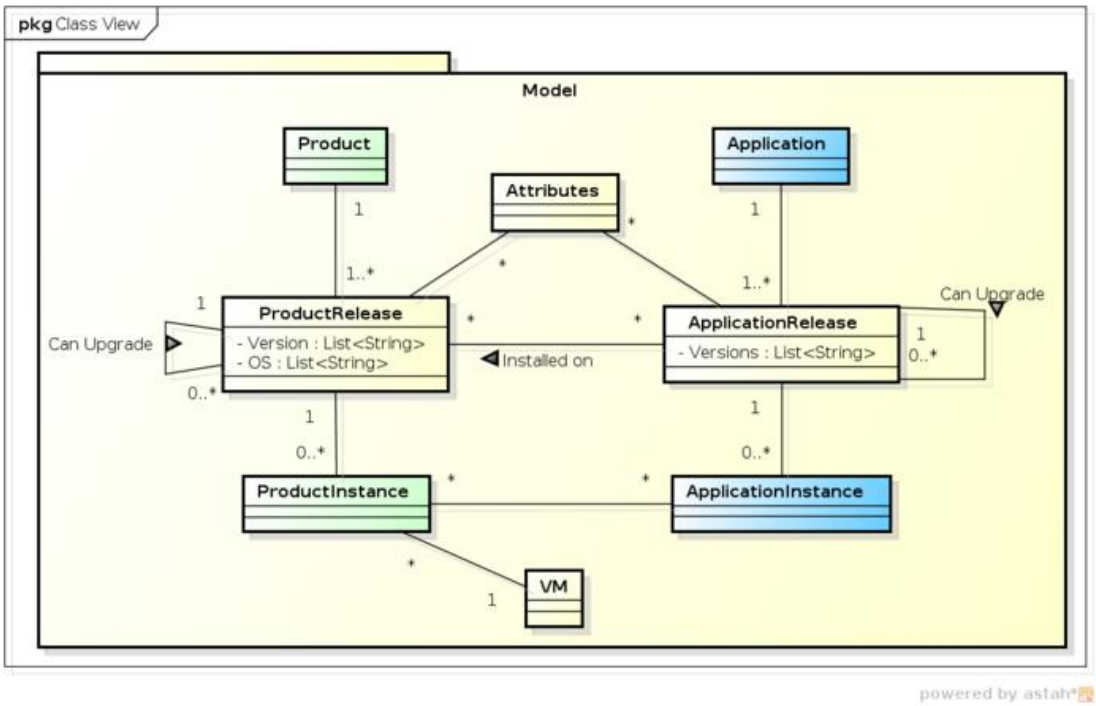
Following the previous FMC diagram of the SDC, this section introduces the main concepts related to this GE through the definition of their interfaces and components, and by means of an example of their use.

The SDC is responsible for carrying out the provisioning and maintenance of the software in virtual machines. The key concepts visible to the cloud user could be differentiated between the interfaces and the components. All of them are described below.

8.5.1 Entities

To use the SDC effectively, the following concepts have to be explained, which are illustrated in next figure. Take into account that these concepts are just a subset of the concepts defined for the PaaS Manager GE.

- **Software Product**. A software product is a software piece to be installed in an operating system that can work by itself (it does not need another software to run). Both Tomcat and MySQL are example of software products.
- **Software Product Release**. A software product release involves a concrete release (version) of a software product. Tomcat 7.0 or Tomcat 5.5 are examples of two releases of the tomcat product. Both software product and product release information are stored in the Product Catalogue.
- **Software Product Instance**. It is the instantiation of a software product release in a VM or server.
- **Application**. An application is a set of configured artifacts, supporting a concrete set of functions, deployed over one or more VMs or servers and software product instances deployed on top of that VMs and servers.
- **Application Release**. An application release represents the specification of a given version of an application, through the specification of all the artifacts and attributes that are required for its deployment.
- **Artifact**. An artifact refers to each one of the application components that made up an application. Both a war or a properties file are examples of artifacts.
- **Application Instance**. An application instance is an actual deployment of all the application artifacts of a given Application Release on one or more RECs.
- **Attributes**. The attributes associated to an application or a software product release that are used to help in their configuration during the deployment process.
- **VM**. A virtual machine or server represents the infrastructure where a REC is created.



SDC Class Diagram

8.5.2 Interfaces

The SDC GE is accessible through the Service Deployment and Configuration Interface (SDCI). This interface is a REST interface that allows End Users or Developers to cope with the deployment and lifecycle management of software products and applications defined to run in virtual machines. In addition, it offers to Software Providers the possibility to add recipes about their software in the SDC catalogue. The SDCI API is specified at SDC Open RESTful API Specification (PRELIMINARY)

In addition, the SDC interacts with other components through well-defined interfaces:

- The **SDC Client Interface**, which is used to obtain information from a VM or server and request Configuration agent executions.
- The **Configuration Engine Interface**, which is the API for interacting with the configuration engine, whose API (for Chef server) is found at [1]

8.5.3 Components

The SDC GE internal components are:

- **Communication Service.** It implements the REST API interface (SDCI) and manages the asynchronous tasks status.
- **ProductManager.** It manages the catalogue of products.
- **ProductInstanceManager.** It is in charge of managing and executing the installation of products and artifacts through the Configuration Engine.
- **ConfigurationManager.** This component is in charge of managing the communication with the Configuration Engine for the assignation of new recipes to the REC execution queue, and to trigger the actual execution through the SDC client in the REC.
- **Repositories.** There is a repository for task statuses, a catalogue of product releases (and applications) and an inventory of product instances.

Used by the SDC GE but separated from it, three components have to be highlighted:

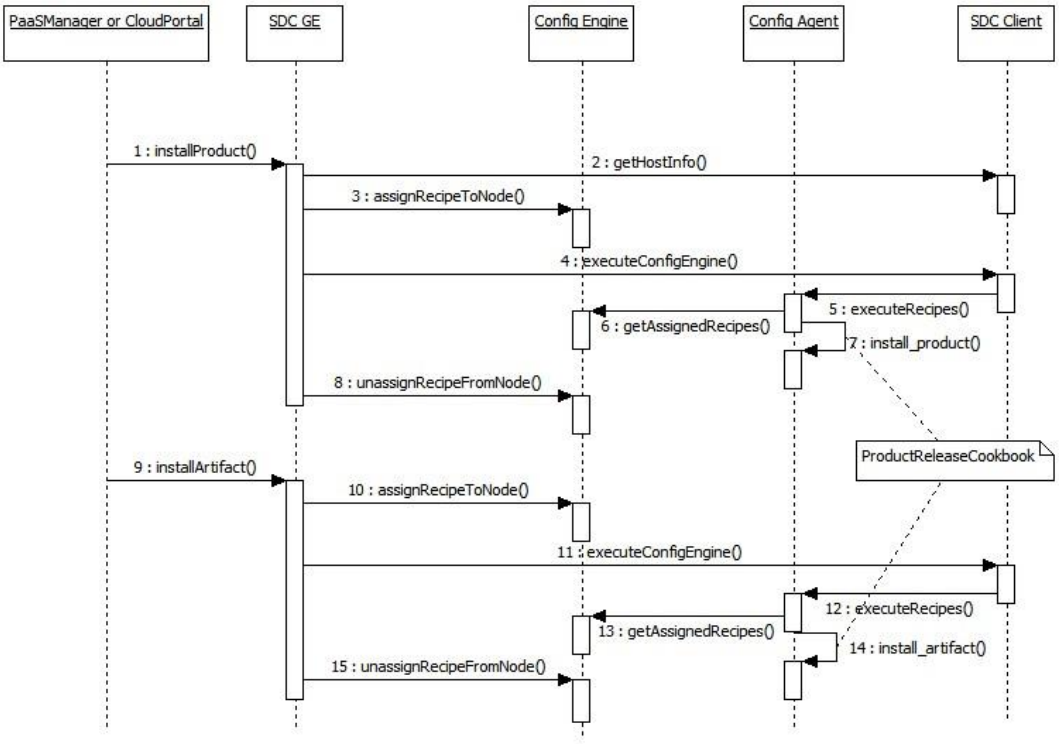
- **Central Configuration Engine,** which manages the execution of recipes on the nodes (RECs). One example is the Chef server.
- **Configuration agent,** which executes the recipes in the local RECs.
- **SDC client,** which serves as support and intermediary between the SDC and the configuration agent.

8.5.4 Example Scenario

Apart from querying information about the defined or installed products, there are two main usages of the SDC GE: installing products and installing artifacts. Installing a product involves to specify either a VM or server details and credentials, and a product (previously defined and with a proper cookbook). Installing an artifact requires identifying the product instance over which it has to be installed and configured, and to trigger the execution of the proper recipe. Both scenarios run asynchronously, and when they are requested, the SDC GE returns a task identifier that can be used to track the status of the operation.

8.6 Main Interactions

Although the scenarios of the SDC are simple, the interactions are complex due to the number of components involved. The following figure depicts how it works.



SDC interactions

Usually, the SDC GE will receive a request for installing a product from the PaaS Manager (1), although it may be also be used by a client application or from a web

portal. The SDC GE knows the IP address of the host where the product has to be installed, but does not know the hostname and host domain, which are required by the Configuration Engine. Therefore, it obtains that information from the SDC Client running in the REC (2).

After that, the SDC GE assigns the execution of a recipe to the corresponding node (3) and notifies the SDC client (4) to request the Configuration Agent in the same VM or server (5) to execute the pending recipes for that node, as informed by the Configuration Engine (6). With that information, the Configuration Agent executes the recipe needed to install the software product instance (7). Finally, the SDC GE requests the Configuration Engine to unassign the recipe to the node (8).

The second scenario starts when the PaaS Manager requests to install an artifact (9). The SDC GE receives information about the product instance and therefore, knows the node where the recipe has to be executed. The process to install the artifact from that moment on is exactly as described above, except because the name of the recipe for installing an artifact will be different from the one in the recipe used to install a software product.

8.7 Basic Design Principles

8.7.1 Design Principles

The SDC GE has to comply with the following technical requirements:

- Decouple the management of the catalogue (specifications of what can be deployed) from the inventory (instances of what has been already deployed).
- Support an Asynchronous interface with polling mechanism to obtain information about the deployment status.
- Deal with ad-hoc interfaces definition, given that no real or de-facto standard exists for PaaS.

8.7.2 Resolution of Technical Issues

When applied to SDC GE, the general design principles outlined above can be translated into the following design approaches:

- REST based automated self-provisioning interface for both products and artifacts.
- Dependency on the Configuration Management Engine (Opscode Chef or Puppet) to automate the installation processes; the virtual machines have to be based on images that include the SDC Client and the Configuration Agent to be usable by the SDC.
- Multitenancy is not covered by the SDC GE. The entire lifecycle of software products and applications is managed at a higher level, by the PaaS Manager or by an administrator.
- SDC GE does not manage the installation and configuration of the infrastructure required for monitoring or measuring the service; these must be provided by the cookbooks or the VM or server images used.

8.8 Detailed Specifications

8.8.1 Open API Specifications

Following is a list of Open Specifications linked to this Generic Enabler. Specifications labeled as "PRELIMINARY" are considered stable but subject to minor changes derived from lessons learned during last interactions of the development of a first reference implementation planned for the current Major Release of FI-WARE. Specifications labeled as "DRAFT" are planned for future Major Releases of FI-WARE but they are provided for the sake of future users.

- SDC Open RESTful API Specification (PRELIMINARY)

8.9 Re-utilised Technologies/Specifications

The SDC GE is based on RESTful Design Principles. The technologies and specifications used in this GE are:

- RESTful web services
- HTTP/1.1 (RFC2616)
- XML data serialization formats.

8.10 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be helpful to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FI-WARE Global Terms and Definitions

- **Runtime Execution Container** (**REC**) -- a VM with all the software stack required to deploy a complete node in an application architecture. It usually comprises one or more VMs, middleware, monitoring probes, and Chef client and SDC agent to support installation and configuration.
- **Software Deployment and Configuration GE** (**SDC GE**) -- a GE in devoted to the automated installation and configuration of software on VMs through the execution of recipes in the corresponding nodes. It relies on Opscode Chef technology.
- **Software Deployment and Configuration Interface** (**SDCI**) -- the interface offered by the SDC GE to be managed by the PaaS Manager or a Cloud Portal.
- **Product Instance** (**PI**) -- an installed software in a VMs, usually referring to middleware or platform software (E.g.: Apache Tomcat, MySQL, etc.).
- **Application Component** (**AC**) -- a component (or configuration artifact) that implements usually one or more components of an application architecture. These ACs are installed on, and differentiated from, existing PIs.
- **PaaS Manager Interface** (**PMI**) -- the interface offered by the PaaS Manager GE to be used by a Cloud Portal to manage both the catalogue and the lifecycle of the platform resources and applications.

9 SDC Open RESTful API Specification

9.1 Introduction to the Software Deployment and Configuration (SDC) API

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

9.1.1 SDC API Core

The Software Deployment and Configuration (SDC) API is a RESTful, resource-oriented API accessed via HTTP/HTTPS that uses XML-based and/or JSON-based representations for information interchange that provide management of software in the Cloud. The SDC offers some mechanisms to install, uninstall, and configure products and applications in a running operating system.

9.1.2 Intended Audience

This specification is intended for both software developers and final users. For the former, this document provides a full specification of how to include new software in the catalogue of the SDC. For the latter, this specification indicates the interface to be provided in order to clients to manage software in their virtual machines. To use this information, the reader should firstly have a general understanding of the SDC Generic Enabler and also be familiar with:

- RESTful web services
- HTTP/1.1 (RFC2616)
- JSON and/or XML data serialization formats.

9.1.3 API Change History

This is the first version for the SDC API

Version	Changes Summary
Oct 09, 2012	• First version of the SDC API, API uses for managing software on top of server.

9.1.4 How to Read This Document

In the whole document the assumption is taken that the reader is familiarized with REST architecture style. Along the document, some special notations are applied to differentiate some special words or concepts. The following list summarizes these special notations.

- A **bold**, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).

- An *italic* font is used to represent document titles or some other kind of special text, e.g., *URI*.
- The variables are represented between brackets, e.g. {id} and in italic font. When the reader find it, can change it by any value.

For a description of some terms used along this document, see[1].

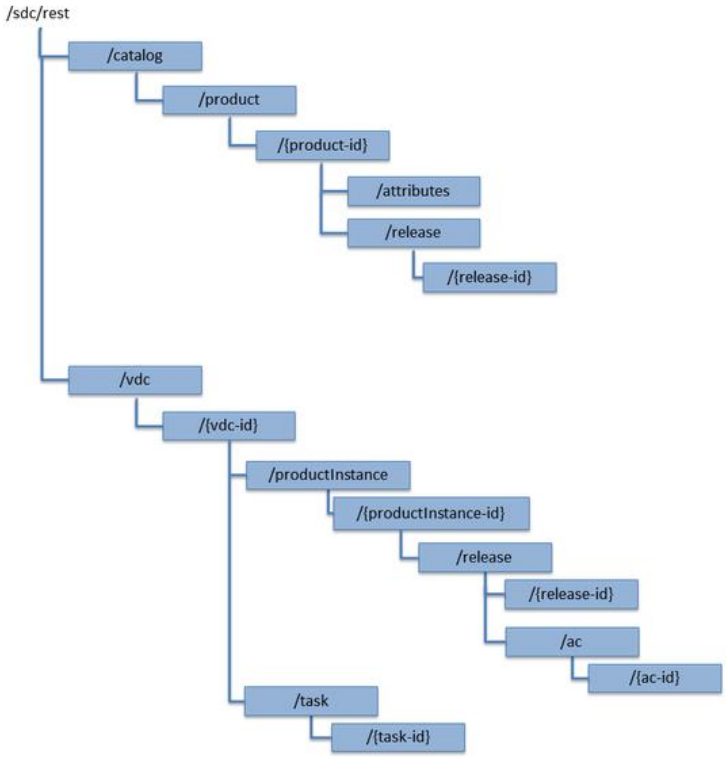
9.1.5 Additional Resources

You can download the most current version of this document from the FIWARE API specification selecting **PDF Version** from the Toolbox menu (left side), which will generate the file to download it. For more details about the **SDC** that this API is based upon, please refer to FI-WARE Cloud Hosting.

9.2 General SDC API Information

9.2.1 Resources Summary

Graphical diagram in which the different URNs that can be used in the API is shown here. The PaaS Manager API can start with http://SDC_IP:port/sdc/rest



SDC Open RESTful API resource summary

9.2.2 Authentication

Not implemented

9.2.3 Representation Format

The SDC API resources are represented by hypertext that allows each resource to reference other related resources. More concisely, XML or JSON format are used for resource representation and URLs are used for referencing other resources by default. The request format is specified using the Content-Type header and is required for operations that have a request body. The response format can be specified in requests using either the Accept header with values application/json or application/xml or adding an .xml or .json extension to the request URI. In the following examples we can see the different options in order to represent format.

```
POST /catalog/product HTTP/1.1

Host: sdc.paas.tid.org

Content-Type: application/json

Accept: application/xml

X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

```
POST /catalog/product HTTP/1.1

Host: sdc.paas.tid.org

Content-Type: application/xml

X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

```
POST /catalog/product HTTP/1.1

Host: sdc.paas.tid.org

Content-Type: application/json

X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

9.2.4 Representation Transport

Resource representation is transmitted between client and server by using HTTP 1.1 protocol, as defined by IETF RFC-2616. Each time an HTTP request contains payload, a Content-Type header shall be used to specify the MIME type of wrapped representation. In addition, both client and server may use as many HTTP headers as they consider necessary.

9.2.5 Resource Identification

API consumer must indicate the resource identifier while invoking a GET, PUT, POST or DELETE operation. SDC API combines both identification and location by terms of URL. Each invocation provides the URL of the target resource along the verb and any required input data. That URL is used to identify unambiguously the resource. For HTTP transport, this is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

SDC API does not enforce any determined URL pattern to identify its resources. Anyway the SM Extension API follows the HATEOAS principle (Hypermedia As The Engine Of Application State). This means that resource representation contains the URLs of the related resources (e.g., book representation contains hyperlinks to its chapters; chapter representation contains hyperlinks to its pages...). API consumer obtains the VDC representation as its following point, which in turn provides hyperlinks that directly or indirectly take to other resources like Services and/or Servers.

SDC API entities provide an instance identifier property (instance ID). This property is used to identify unambiguously the entity but not the REST resource used to manage it, which is identified by its URL as described above. It is common that most implementations make use of instance ID to compose the URL (e.g., the book with instance ID 1492 could be represented by resource http://.../book/1492), but such an assumption should not be taken by API consumer to obtain the resource URL from its instance ID.

9.2.6 Links and References

Resources often lead to refer to other resources. In those cases, we have to provide an ID or an URL to a remote resource.

9.2.7 Paginated Collections (Optional)

In order to reduce the load on the service, we can decide to limit the number of elements to return when it is too big. In order to do it, we use the limit, which is the maximum number of element to return, and last parameter, which was the last element to see. In the response JSON, we include an atom "next" links to follow to the next group of data. The last page in the list will not contain a "next" link. If there is an over limit error, the API will return a 413 message (over limit error) or 404 message (item not found error).

9.2.8 Efficient Polling with the Changes-Since Parameter (Optional)

In this case we can specify the parameter changes-since in a GET method in order that the response will give us only the changed information from the previous request specified through a dateTime format ISO 8601 (2011-01-24T17:08Z).

9.2.9 Limits

Not implemented

9.2.10 Versions

This is the first version.

9.2.11 Extensions

No apply yet.

9.2.12 Faults

The error code is returned in the body of the response for convenience. The message section returns a human-readable message that is appropriate for display to the end user. The details section is optional and may contain information—for example, a stack trace—to assist in tracking down an error. The detail section may or may not be appropriate for display to an end user.

SDC API Faults

Fault Element	Associated Error Codes	Description
Fault	500, 400, other codes possible	Error in the operation
serviceUnavailable	503	The service is not available
unauthorized	401	You are not authorized to access to that operation. The token is not correct.
forbidden	403	It is forbidden
badRequest	400	The request has not been done correctly
badMediaType	415	The payload media is not correct
itemNotFound	404	It is not exist
badMethod	405	Method not allowed
overLimit	413	Request entity too large

Besides the previous errors, when there is a error in the SDC operation, the SDC prints this error in the task result although the request error code is 200. In this case, the task status is ERROR and there is a message indicating its error message.

```
<task href="http://130.206.80.112:8080/sdc/rest/vdc/{vdc-
id}/task/81" startTime="2012-11-19T08:20:24.844+01:00"
endTime="2012-11-19T08:20:50.700+01:00" status="ERROR">
```

```
  <error message="The product tomcat-7 can not be installed in
server since the server does not exists"
minorErrorCode="SdcRuntimeException"/>
  <result href="http://130.206.80.112:8080/sdc/rest/vdc/{vdc-
id}/product/82"/>
  <description>Install product tomcat in server</description>
  <vdc>{vdc-id}</vdc>
</task>
```

In this case the main controlled exceptions are:

1. **ProductNotFoundException**: The product not exist in the catalogue
2. **ProductInstanceNotFoundException**: The product instance has not been deployed in the server
3. **ProductReleaseNotFoundException**: The product release in not stored in the catalogue
4. **InvalidInstallProductRequestException**: The product to be installed in the server is not valid
5. **InvalidProductReleaseException**: The product release to be introduced in the catalogue is not valid
6. **IncompatibleProductsException**: The products to be installed are incomptibles
7. **AlreadyExistsProductReleaseException**: The product release ist already in the catalogue
8. **AlreadyInstalledException**: The product is already deployed in the server
9. **NotInstalledProductsException**: The product to be undeployed in the not installed in the server
10. **InvalidProductReleaseUpdateRequestException**: The product cannot be updated
11. **CanNotDeployException**: It is not possible to do any action in the server (install, uninstall and so no).
12. **CanNotCallChefException**: It is not possible to call the Chef server to execute any action.
13. **NodeExecutionException**: There is an error in the execution of chef in the node
14. **FSMViolationException**: It is not possible to execute an action to the node. The node is not in the right status
15. **ShellCommandException**: There is an error in executing a command in the node
16. **SdcRuntimeException**: An internal SDC error has happened.

9.3 API Operations

In this section we go in depth for each operation. These operations have been described in theSDC Architecture. The FIWARE programmer guide will also provide examples of how to use the API. The SDC API is divided among two funciotnalities: the catalogue of the software to be deployed and the provisioning of this software. Due to it, we make two main divisions in the API explanation. All the payload examples in this section are formalized in XML, but the user could use also JSON as representation format.

9.3.1 Product Catalogue

9.3.1.1 *List Products in the catalogue*

Verb	URI	Description
GET	*/sdc/rest/catalog/product*	Lists all Product in the catalogue.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This operation does not require a request body.

This operation lists the products stored in the catalogue.

The following example shows an XML response for the list Product API operation:

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
   <product>
      <name>tomcat</name>
      <description>tomcat J2EE container</description>
      <attributes>
         <key>port</key>
         <value>8080</value>
         <description>The listen port</description>
      </attributes>
      <attributes>
         <key>ssl_port</key>
         <value>8443</value>
         <description>The ssl listen port</description>
      </attributes>
   </product>
</product>
```

9.3.1.2 *Add a product to the catalogue*

Verb	URI	Description
POST	*/sdc/rest/catalog/product*	Add the selected Product Release into the SDC's catalog. If the Product already exists, then add the new Release. If not, this method also creates the product.

Normal Response Code(s): 200

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

This operation stores a new product in the catalogue. This call creates a new Product in the catalogue.

```
<?xml version="1.0" encoding="UTF-8"?>
<product>
  <name>postgresql</name>
  <description>db manager</description>
  <attributes>
    <key>username</key>
    <value>postgres</value>
    <description>The administrator usename</description>
  </attributes>
  <attributes>
    <key>password</key>
    <value>postgres</value>
    <description>The administrator password</description>
  </attributes>
</product>
```

9.3.1.3 *Get Product Details*

Verb	URI	Description
GET	*/sdc/rest/catalog/product/{product-id}*	Lists details of the specified product.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

Specify the Product ID as productId in the URI. This operation does not require a request body and returns the details of a specific product by its ID.

Product Response: XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<product>
   <name>tomcat</name>
   <description>tomcat J2EE container</description>
```

```
    <attributes>
       <key>port</key>
       <value>8080</value>
       <description>The listen port</description>
    </attributes>
    <attributes>
      <key>ssl_port</key>
      <value>8443</value>
      <description>The ssl listen port</description>
    </attributes>
</product>
```

9.3.1.4 *Delete Product in the Catalogue*

Verb	URI	Description
DELETE	*/sdc/rest/catalog/product/{product-id}*	Deletes the product with the id *(product-id)* in the catalogue

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

It specifies the ID of the product in the URI. This operation does not require a request body and returns without any body.

9.3.1.5 *Get Product Attributes*

Verb	URI	Description
GET	*/sdc/rest/catalog/product/{product-id}/attributes*	Get the attributes fot the product.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

It specifies the ID of the product in the URI. This operation does not require a request body and returns the details of a specific product by its ID.

Attributes XML:

```
<attributes>
```

```
  <attribute>
    <key>port</key>
    <value>8080</value>
    <description>The listen port</description>
  </attribute>
  <attribute>
    <key>ssl_port</key>
    <value>8443</value>
    <description>The ssl listen port</description>
  </attribute>
</attributes>
```

9.3.2 Product Provisioning

9.3.2.1 ***Install a product***

Verb	URI	Description
POST	*/sdc/rest/vdc/{vdc-id}/productInstance*	Install a product in a server

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This call deploys a Product (existing in the catalogue) in a server for the vdc *{vdc-id}*. It request provides server features like IP are in the payload as well as the product characteristics to be installed.

```
<?xml version="1.0" encoding="UTF-8"?>
<productInstanceDto>
  <vm>
    <ip>130.206.80.114</ip>
    <fqn>fqnname</fqn>
    <osType>95</osType>
    <hostname>rhel-5200ee66c6</hostname>
  </vm>
  <product>
    <productDescription/>
    <productName>tomcat</productName>
    <version>7</version>
```

```
    <supportedOOSS>
        <osType>Debian 5</osType>
        <name>Debian 5</name>
        <version>5</version>
      </supportedOOSS>
 </product>
</productInstanceDto>
```

Once executed the request, the SDC returns a Task to manage this operation. This task specifies the status of the operation, a description and more information. Thus, in this case the product provisioning Response is a XML like that

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task href="http:/130.206.80.112:8080/sdc/rest/vdc/{vdc-
id}/task/{task-id}" startTime="2012-11-08T09:13:18.311+01:00"
status="RUNNING">
  <description>Install product tomcat in  VM rhel-
5200ee66c6</description>
  <vdc>{vdc-id}</vdc>
</task>
```

Then, when the product has been installed asking for the task status

```
  GET http:/130.206.80.112:8080/sdc/rest/vdc/{vdc-
id}/task/{task-id}
```

It is possible to obtain, the sucess status.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task href="http:/130.206.80.112:8080/sdc/rest/vdc/{vdc-
id}/task/{task-id}" startTime="2012-11-08T09:13:18.311+01:00"
status="SUCCESS">
  <description>Install product tomcat in  VM rhel-
5200ee66c6</description>
  <vdc>{vdc-id}</vdc>
</task>
```

9.3.2.2 *Get information about the products installed (product instances) in a server*

Verb	URI	Description
GET	*/sdc/rest/vdc/{vdc-id}/productInstance*	Install a product in a server

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This operation does not require a request body and returns the details of the list of product instances deployed in the server.

Product Instance List Response: XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<productInstances>
  <productInstance>
    <id>56</id>
    <date>2012-11-07T15:56:30.590+01:00</date>
    <status>INSTALLED</status>
    <vm>
      <ip>130.206.80.114</ip>
      <hostname>rhel-5200ee66c6</hostname>
      <fqn>fqn</fqn>
      <osType>95</osType>
    </vm>
    <vdc>{vdc-id}</vdc>
    <product>
      <releaseNotes>Tomcat server 7</releaseNotes>
      <version>7</version>
      <product>
        <name>tomcat</name>
        <description>tomcat J2EE container</description>
        <attributes>
           <key>port</key>
           <value>8080</value>
        </attributes>
      </product>
   </product>
  </productInstance>
</productInstances>
```

9.3.2.3 *Get details about a product installed in a server*

Verb	URI	Description

GET	*/sdc/rest/vdc/{vdc-id}/productInstance/{productInstance-id}*	Lists of Products installed in the server

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

This operation does not require any payload in the request and provides a productInstances XML response.

Product Instances Response: XML

```
<productInstance>
   <id>{productInstance-id}</id>
   <date>2012-11-08T15:30:56.764+01:00</date>
   <status>INSTALLED</status>
   <vm>
      <ip>130.206.80.114</ip>
      <hostname>testchef</hostname>
      <domain>.test</domain>
      <fqn>es.tid.dd</fqn>
   </vm>
   <vdc>{vdc-id}</vdc>
   <product>
      <name>tomcat</name>
      <description>tomcat J2EE container</description>
      <attributes>
         <key>port</key>
         <value>8080</value>
         <description>The listen port</description>
      </attributes>
      <attributes>
         <key>ssl_port</key>
         <value>8443</value>
         <description>The ssl listen port</description>
      </attributes>
   </product>
</productInstance>
```

9.3.2.4 *Uninstall product in the server*

Verb	URI	Description
DELETE	*/sdc/rest/vdc/{vdc-id}/productInstance/{productInstance-id}*	Uninstall the product belonging the the product instance {productInstanceId} in the server.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

It is required to specify the Product Instance to be undeployed. This operation does not require a request body and returns the details of a generated task.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task href="http:/130.206.80.112:8080/sdc/rest/vdc/{vdc-id}/task/72" startTime="2012-11-08T09:45:44.020+01:00" status="RUNNING">
  <description>Uninstall product tomcat in  VM rhel-5200ee66c6.</description>
  <vdc>{vdc-id}</vdc>
</task>
```

9.3.2.5 *Update product version in the server*

Verb	URI	Description
PUT	*/rest/vdc/{vdc-id}/productInstance/{productInstance-id}/release/{release-id}*	Update the version of the product deployed in the server

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

Specify the Product Instance ID to be updated in the URI. This operation does not require a request body and returns the details of a generated task.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task href="http:/130.206.80.112:8080/sdc/rest/vdc/{vdc-id}/task/72" startTime="2012-11-08T09:45:44.020+01:00" status="RUNNING">
  <description>Updating product tomcat in  VM rhel-5200ee66c6.</description>
  <vdc>{vdc-id}</vdc>
```

```
</task>
```

9.3.2.6 *Reconfigure product (attributes) in the server*

Verb	URI	Description
PUT	*/rest/vdc/{vdc-id}/productInstance/{productInstance-id}/*	Update the version of the product deployed in the server

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

It specifies the Product Instance ID as *{productInstance-id}* in the URI. The payload of this request contains the new attributes to be overwriteen.

```
<attributes>
  <key>port</key>
  <value>8082</value>
  <description>The listen port</description>
</attributes>
```

This operation returns the details of a generated task.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task href="http:/130.206.80.112:8080/sdc/rest/vdc/{vdc-
id}/task/72" startTime="2012-11-08T09:45:44.020+01:00"
status="RUNNING">
  <description>Uninstall product tomcat in  VM rhel-
5200ee66c6.</description>
  <vdc>{vdc-id}</vdc>
</task>
```

9.3.3 Task Management

Verb	URI	Description
GET	*/rest/vdc/{vdc-id}/task/{task-id}*	Get the status of a task.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

This operation recovers the status of a task created previously. It does not need any request body and the response body in XML would be the following.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task href="http:/130.206.80.112:8080/sdc/rest/vdc/{vdc-
id}/task/{task-id}" startTime="2012-11-08T09:13:18.311+01:00"
status="SUCCESS">
  <description>Install product tomcat in  VM rhel-
5200ee66c6</description>
  <vdc>{vdc-id}</vdc>
</task>
```

The value of the status attribute could be one of the following:

Value	Description
QUEUED	The task is queued for execution.
PENDING	The task is pending for approval.
RUNNING	The task is currently running.
SUCCESS	The task is completed successfully.
ERROR	The task is finished but it failed.
CANCELLED	The task has been cancelled by user.

10 FIWARE OpenSpecification Cloud PaaS

10.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.org and similar pages in order to understand the complete context of the FI-WARE project.

10.2 Copyright

Copyright © 2012 by Telefónica I+D. All Rights Reserved.

10.3 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

10.4 Overview

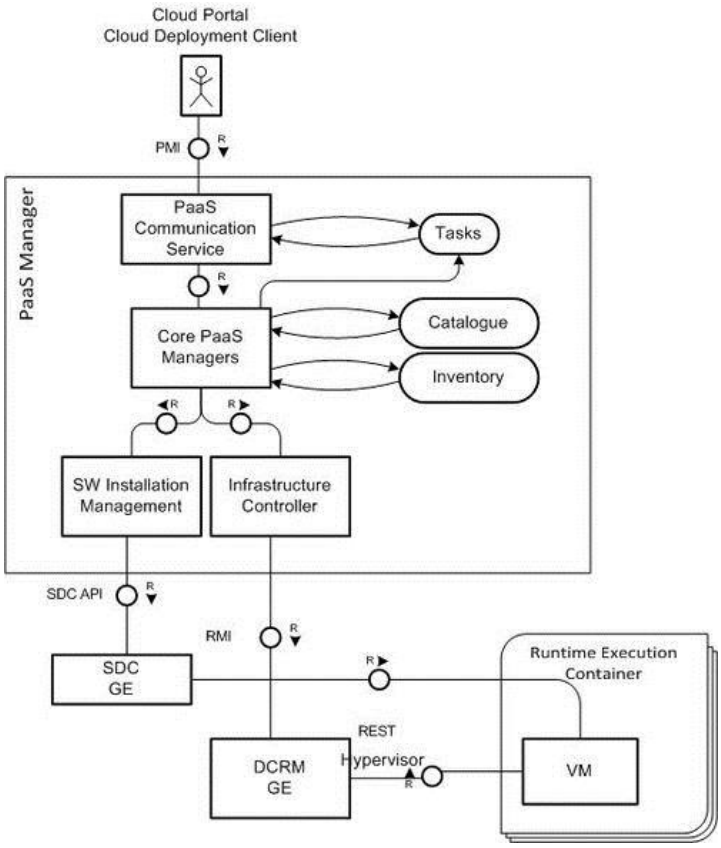
The main functionality that the PaaS Manager provides is:

- Management of Application Environments, which involves the provisioning and configuration of IaaS resources (eventually including NaaS features), and installation, configuration and management of the Products Instances (PIs) required for the application components to be deployed.
- Management of Application Components (ACs) (lifecycle and configuration) with the help of SDC GE for the installation and configuration of ACs.

The PaaS Manager executes orchestration logic that is specified in the Environment element specified as part of the PaaS Open RESTful API Specification (PRELIMINARY). Environment information can be previously registered in the Environment Catalogue or it can be provided by the user.

The PaaS Manager GE interacts with the DCRM GE for the deployment of the hardware infrastructure (virtual machines and networks) and with the SDC GE for the installation and configuration of the software.

The figure below shows the architecture of the PaaS Manager. There are three layers and a persistence mechanism: the implementation of the PaaS interface, the set of managers in charge of executing the tasks, and the clients to interact with other GEs (DCRM GE and SDC GE). By using these GEs, the PaaS Manager GE orchestrates the provisioning of VMs or servers and the deployment of the corresponding software products over which the applications can be deployed.



PaaS Manager architecture specification

10.4.1 Target Usage

The PaaS Manager GE provides a new layer over the DCRM GE in the aim of easing the task of deploying applications on a Cloud infrastructure. Therefore, it orchestrates the provisioning of the required virtual resources at IaaS level through the DCRM , and then, the installation and configuration of the whole software stack of the application, taking into account the underlying virtual infrastructure. It provides a flexible mechanism to perform the deployment, enabling multiple deployment architectures: everything in a single VM or server, several VMs or servers, or elastic architectures based on load balancers and different software tiers.

10.5 Main concepts

Following the above FMC diagram of the PaaS Manager, in this section we introduce the main concepts related to this GE through the definition of their interfaces and components and finally an example of their use.

10.5.1 Basic Concepts

The PaaS Manager manages the lifecycle of applications and platforms software where they are deployed on top of VMs. There is a distinction between specification of resources and the actually deployed resources. For most of the resources, there is a software component that implements its lifecycle management.

10.5.2 Entities

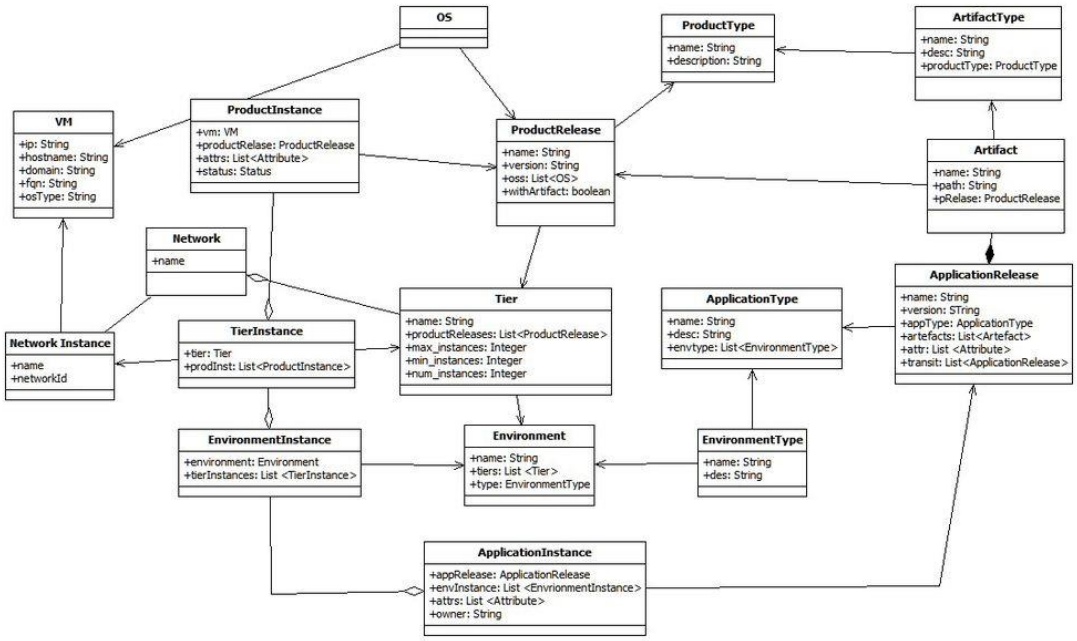
The following figure depicts the main entities managed by the PaaS Manager:

- **Software Product Release & Product Instance (PI)**. They represent an installable software (usually middleware) that is installed previous to the deployment of the application components, e.g., Apache Tomcat, MongoDB, MySQL, etc. The software product release contains the information about the software to be installed, while the product instance refers to the product release already instantiated.
- **Tier & Tier Instance**. An application is structured into tiers, each one comprising a set of VMs that share the same virtual image and where the same set of software products is installed. Nodes in a tier are clonable, typically in order to handle elasticity. A tier instance is the result of instantiating a given Tier definition (Tier template). Example of a tier is the one associated to the farm of web servers serving static web pages in the given portal associated to a CRM application
- **Network/Subnetwork & Network/Subnetwork Instance**. The application can be deployed into different Network and Subnetworks. Thus, a tier can be associate one or more networks, where their VMs are going to be deployed. The instantiation of these network and subnetworks correspond to the instances.
- **Environment & Environment Instance**, which represent the complete software stack, including all the tiers, required for the deployment of an application. As an example, an on-line shop application maybe structured into three tiers, the first tier associated to the farm of web servers, a second tier associated to application servers running business logic and a third layer hosting a NoSQL data management layer. Nodes in the first tier would typically host Apache Web Server software. Nodes in the second tier may host some given Java application server (e.g., JBoss) while the third tier may comprise nodes hosting shard daemons of MongoDB.
- **Application Release & Application Instance**, represent an application specification with all the artefacts and configuration attributes, and its deployment over an Environment Instance. E.g.: Mediawiki.

There are two components in the architecture of the PaaS Manager where these entities are handled:

- **Resources Catalogue**, which contains all the specifications of resources that can be managed by PaaS Manager, including environment, product release information to be deployed. This catalogue can be populated by using an API.
- **Resources Inventory**, which represents the actual instantiation of resources for a specific deployment. For each element in the inventory (called *-Instance), there is an equivalent in the catalogue.

It is important to highlight that the ApplicationType's and EnvironmentType's are catalogued in order to provide typical deployment environments for applications (Java-Web, Java-Python, etc.). ProductType's and ArtifactType's are also catalogued in order to provide common management mechanisms (deployment, configuration, etc.). Thus, besides having a good catalogue for typical environments, the GE can efficiently manage new application patterns that haven't been catalogued. That way, the PaaS Manager GE is flexible to manage new contexts.



PaaS Manager class diagram

10.5.3 Interfaces

The PaaS Manager Interfaces are:

- Northbound interface. The PaaS Manager GE offers an interface (PMI) to its users to manage environments and applications. This interface is specified in PaaS Open RESTful API Specification (PRELIMINARY).
- Southbound interfaces. The PaaS Manager implementation of the enabler invokes the following interfaces:
  - DCRM interfaces (RMI): DCRM OpenStack Open RESTful API Specification (PRELIMINARY)
  - SDC interface (SDCI): SDC Open RESTful API Specification (PRELIMINARY)

10.5.4 Components

The PaaS Manager GE includes three main blocks of components, a data repository, which keeps the definition of the environments and products persistent and the inventory of deployed environments and applications.

- **PaaS Communication Service (PMI)** is responsible of offering a RESTful interface to the PaaS Manager GE users and to handle the asynchronicity of the calls through a set of asynchronous tasks. It triggers the appropriate manager to handle the request. PMI manages synchronous calls by providing a task id which can be checked to obtain its status.
- **Core PaaS Managers**. A number of specialized manager components are responsible of managing the different entities.
  - **ApplicationInstanceManager**, is responsible for the installation of application components on an existing environment.

- **ApplicationReleaseManager**, is responsible for managing the specifications of applications in the PaaS Manager GE.
- **ArtifactManager**, is responsible of managing application components and to find the appropriate product to be installed in and configured in an environment.
- **EnvironmentInstanceManager**, is responsible of deploying and managing environment instances.
- **EnvironmentManager**, is in charge of the management of environment specifications.
- **ProductInstanceManager**, is in charge of managing the installation and configuration of a product instance on a VM or server through the SWInstallationManager
- **ProductReleaseManager**, in in charge of managing the catalogue of available products that can be used to define environments.
- **TemplateManager**, responsible for creating templates from existing tier instances to be replicated during elasticity.
- **TierInstanceManager**, which orchestrates the installation of all the products that are required in a VM or server to install the application components.

- **InfrastructureManager (Service Manager Controller)**, manages the creation and lifecycle of required IaaS resources.
- **SWInstallationManager**, provides the mechanisms to coordinate with the SDC GE the installation of products and artifacts on VMs or servers, and to configure them.

10.5.5 Example Scenario

The PaaS Manager GE is involved in three different phases:

- Management of the catalogue of products and environments and the lifecycle of the environments.
- Deployment of applications.
- Management of applications at runtime.

10.5.5.1 *Environment Management*

The management of environments involves several operations that have to do the configuration and provisioning of the resources required for the deployment of applications.

First of all, the environments have to be defined. The definition of an environment includes the specification of the products that will be supported. This functionality is related to the capacities of the SDC GE, that is, the list of products that can be actually installed, configured and used. Therefore, the definition of environments involves the specification of products and their supported releases as well as the specification of tiers.

Secondly, in order to enable the deployment of applications, an environment has to be deployed. This can be performed on demand, just previous to the deployment of an application, or in advance, in order to enable faster deployments.

Finally, the environments and environment instances can be retired, redefined or evolved.

10.5.5.2 *Application Deployment*

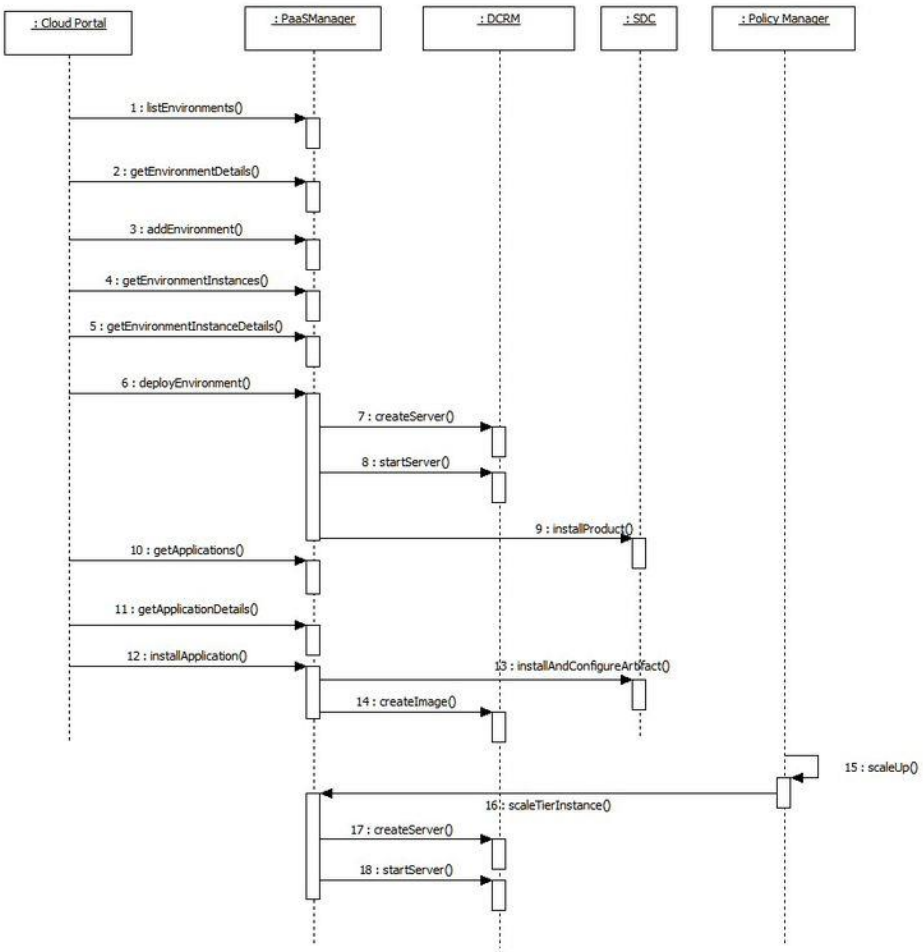
Applications are specified together with the specification of the environment that has to be used for actual deployment. An application is described by the different artifacts that compose it, and the products and configuration attributes required for managing the deployment and configuration. The PaaS Manager checks if the application specification is compatible with the environment specification, essentially by checking the compatibility between the product requirements associated to the application and the environment specification. If they are compatible, the PaaS Manager GE coordinates the actual deployment and configuration of artifacts in the environment with the SDC GE. Once an application is deployed, a VM or server image is created for scaling the nodes at each tier.

10.5.5.3 *Runtime Management*

During the runtime of an application, the Policy Manager GE can detect the need to scale up the resources allocated to a given application. If the deployment architecture is scalable, the PaaS Manager GE determines whether a tier node has to be cloned and sends the proper request for instantiating the corresponding virtual image to the DCRM GE. This mechanism, based on the VM or server image created for each tier node during deployment, allows for a fast scale up, since the whole software installation and configuration process is omitted.

10.6 Main Interactions

The following picture depicts some interactions between the PaaS Manager, the Cloud Portal as main user, the Service Manager and the SDC, in a typical scenario.



PaaS Manager sequence diagram

The interactions are:

1. The Cloud Portal gets information about all the possible environments that can be deployed.
2. The Cloud Portal gets detailed information about a specific environment (tiers, products, etc.).
3. The Cloud Portal requests the creation of a new environment to be used for instantiation lately, providing all the details. The user can customize existing environments and create a new one based on them.
4. The Cloud Portal can get information about the already deployed environments instances.
5. The Cloud Portal can get the detailed information about a given environment instance.
6. If there are no appropriate ready to deploy environment instances, the Cloud Portal requests to deploy an environment in the testbed, which is an environment instance. This request can take some time since the instantiation of an environment involves the deployment of virtual machines and the installation of the software on top of it. For avoid blocking the PaaS Manager component, the request is asynchronous. Thus, the Cloud Portal will be checking the status of the instantiation task.

7. The PaaS Manager GE requests the DCRM GE to create the different servers involved in the enviornment..
8. The PaaS Manager GE requests the DCRMto start up of all servers by asking for start service.
9. The PaaS Manager GE requests the SDC GE to install the appropriate products for the environment in the different provisioned servers.
10. It is possible for the Cloud Portal to get the complete list of applications installed in a given environment instance.
11. The Cloud Portal may obtain and show the details of an application deployed on an environment. This might be useful to decide if another application can be deployed on the same environment instance.
12. The Cloud Portal may request the installation of an application on a given environment instance.
13. The PaaS Manager GE requests the SDC GE to deploy the artifacts of an application on an environment instance. This operation is usually requested several times.
14. The PaaS Manager GE requests the DCRM GE to create a server or VM or server image upon an installed and configured tier instance (node) for future elasticity.
15. The Policy Manager GE takes the decision to scale up based on monitoring information.
16. The Policy Manager GE requests to scale a tier.
17. The PaaS Manager GE clones the tier instance node by requesting a new server to the DCRM GE.
18. The PaaS Manager GE starts up the new server.

10.7 Basic Design Principles

10.7.1 Design Principles

The PaaS Manager GE has to support the following technical requirements:

- Decoupling the management of the catalogue (specifications of what can be deployed) and the management of the inventory (instances of what has been already deployed).
- Asynchronous interface with polling mechanism to obtain information about the deployment status.
- Decoupling the management of environments from the management of applications, since there could be uses cases where the users of those functionalities could be different ones.
- Fast elasticity mechanisms, avoiding overhead or repeated work as much as possible during elasticity.

10.7.2 Resolution of Technical Issues

When applied to PaaS Manager, the general design principles outlined at Cloud Hosting Architecture can be translated into the following design approaches:

- REST based automated self-provisioning interfaces, both for environments and applications.
- The PaaS Manager GE relies on the service manager for service provisioning, and therefore, the multitenancy is offered at the level of IaaS, and not PaaS. Nevertheless, the environments are created per customer on demand.
- The PaaS Manager GE has been designed for elasticity and supports fast elasticity on demand, as requested by the DCRM GE.
- The PaaS Manager GE inventory keeps control of all the resources provisioned for each user.

10.8 Detailed Specifications

10.8.1 Open API Specifications

Following is a list of Open Specifications linked to this Generic Enabler. Specifications labeled as "PRELIMINARY" are considered stable but subject to minor changes derived from lessons learned during last interactions of the development of a first reference implementation planned for the current Major Release of FI-WARE. Specifications labeled as "DRAFT" are planned for future Major Releases of FI-WARE but they are provided for the sake of future users.

- PaaS Open RESTful API Specification (PRELIMINARY)

10.9 Re-utilised Technologies/Specifications

The PaaS Manager GE is based on RESTful Design Principles. The technologies and specifications used in this GE are:

- RESTful web services
- HTTP/1.1 (RFC2616)
- JSON and XML data serialization formats.

10.10 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be helpful to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FI-WARE Global Terms and Definitions

- **Runtime Execution Container** (**REC**) -- a VM with all the software stack required to deploy a complete node in an application architecture. It usually comprises one or more VMs, middleware, monitoring probes, and Chef client and SDC agent to support installation and configuration.
- **Software Deployment and Configuration GE** (**SDC GE**) -- a GE in devoted to the automated installation and configuration of software on VMs through the execution of recipes in the corresponding nodes. It relies on Opscode Chef technology.
- **Software Deployment and Configuration Interface** (**SDCI**) -- the interface offered by the SDC GE to be managed by the PaaS Manager or a Cloud

Portal.

- **Product Instance** (**PI**) -- an installed software in a VMs, usually referring to middleware or platform software (E.g.: Apache Tomcat, MySQL, etc.).
- **Application Component** (**AC**) -- a component (or configuration artifact) that implements usually one or more components of an application architecture. These ACs are installed on, and differentiated from, existing PIs.
- **PaaS Manager Interface** (**PMI**) -- the interface offered by the PaaS Manager GE to be used by a Cloud Portal to manage both the catalogue and the lifecycle of the platform resources and applications.

11 PaaS Open RESTful API Specification

11.1 Introduction to the PaaS Manager API

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

11.1.1 PaaS Manager API Core

The PaaS Manager API is a RESTful, resource-oriented API accessed via HTTP/HTTPS that uses XML-based and/or JSON-based representations for information interchange that provide management of software in the Cloud. The PaaS Manager offers some mechanisms for creating environment for deploying application. The environment involves both the virtual machines and the software required.

11.1.2 Intended Audience

This specification is intended for both software developers and final users. For the former, this document provides a full specification about a environment catalogue. For the latter, this specification indicates the interface to be provided in order to clients to manage their environment in the testbed as well as deploying applications on top of these environments. To use this information, the reader should firstly have a general understanding of thePaaS Manager GE and also be familiar with:

- RESTful web services
- HTTP/1.1 (RFC2616)
- JSON and/or XML data serialization formats.

11.1.3 API Change History

This version of the PaaS Manager API Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Version	Changes Summary
Nov 14, 2012	• First version of the PaaS Manager API, API uses for managing environments and applications in the Cloud.
Dic 18, 2013	• Second version of the PaaS Manager API including network and regions in the tier definition

11.1.4 How to Read This Document

In the whole document the assumption is taken that the reader is familiarized with REST architecture style. Along the document, some special notations are applied to differentiate some special words or concepts. The following list summarizes these special notations.

- A **bold**, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).
- An *italic* font is used to represent document titles or some other kind of special text, e.g., *URI*.
- The variables are represented between brackets, e.g. {id} and in italic font. When the reader find it, can change it by any value.

For a description of some terms used along this document, see [1].

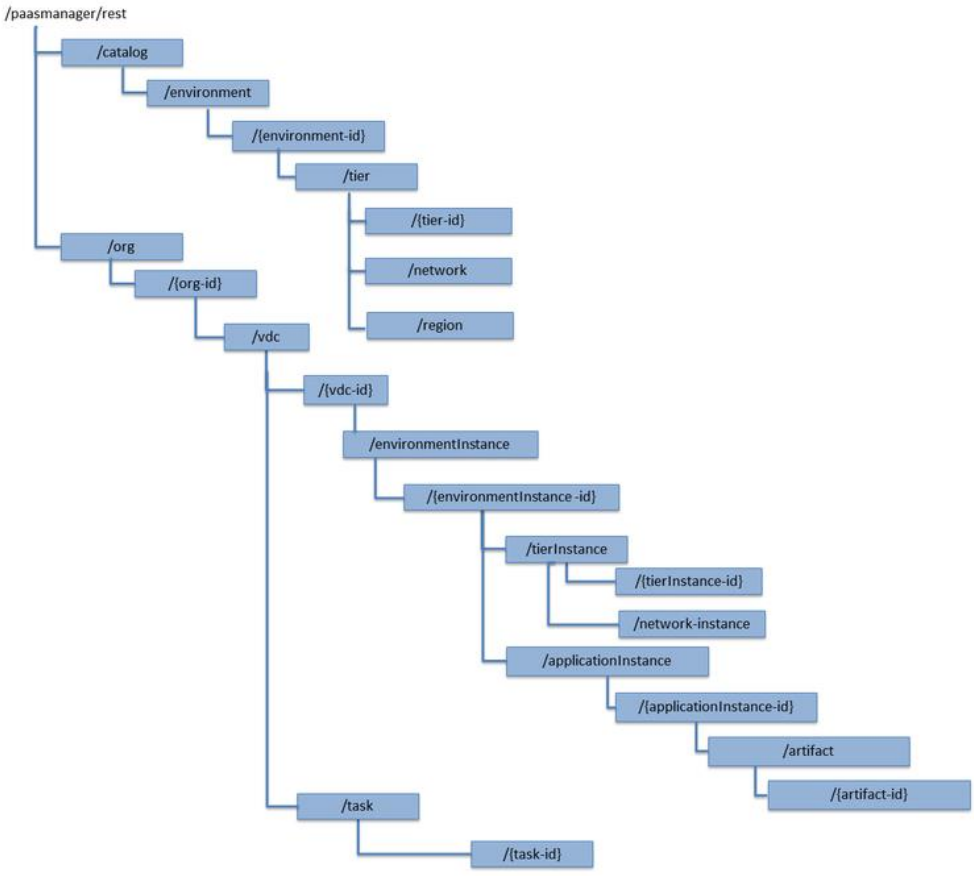
11.1.5 Additional Resources

You can download the most current version of this document from the FIWARE API specification selecting **PDF Version** from the Toolbox menu (left side), which will generate the file to download it. For more details about the **PaaS Manager** that this API is based upon, please refer to Cloud Hosting.

11.2 General PaaS Manager API Information

11.2.1 Resources Summary

A diagram, in which the different URNs that can be used in the API, is shown here. The URL for accessing to the API is something like: *http://IP_host:port/paasmanager/rest*.



PaaS Manager Open RESTful API resources summary

11.2.2 Authentication

It is using Oauth for authentication. It means including the X-Auth-Token header with a valid token like it

```
 X-Auth-Token: X-Auth-Token: eaaafd18-0fed-4b3a-81b4-
663c99ec1cbb
```

11.2.3 Representation Format

The PaaS Manager API resources are represented by hypertext that allows each resource to reference other related resources. More concisely, XML or JSON format are used for resource representation and URLs are used for referencing other resources by default. The request format is specified using the Content-Type header and is required for operations that have a request body. The response format can be specified in requests using either the Accept header with values application/json or application/xml or adding an .xml or .json extension to the request URI. In the following examples we can see the different options in order to represent format.

```
POST /catalog/environment HTTP/1.1

Host: paasmanager.tid.org

Content-Type: application/json

Accept: application/xml

X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

```
POST /catalog/environment HTTP/1.1

Host: paasmanager.tid.org

Content-Type: application/xml

X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

```
POST /catalog/environment HTTP/1.1

Host: paasmanager.tid.org

Content-Type: application/json

X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

11.2.4 Representation Transport

Resource representation is transmitted between client and server by using HTTP 1.1 protocol, as defined by IETF RFC-2616. Each time an HTTP request contains payload, a Content-Type header shall be used to specify the MIME type of wrapped representation. In addition, both client and server may use as many HTTP headers as they consider necessary.

11.2.5 Resource Identification

API consumer must indicate the resource identifier while invoking a GET, PUT, POST or DELETE operation. PaaS Manager API combines both identification and location by terms of URL. Each invocation provides the URL of the target resource along the verb and any required input data. That URL is used to identify unambiguously the resource. For HTTP transport, this is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

PaaS Manager API does not enforce any determined URL pattern to identify its resources. Anyway the PaaS Manager API follows the HATEOAS principle (Hypermedia As The Engine Of Application State). This means that resource representation contains the URLs of the related resources (e.g., book representation contains hyperlinks to its chapters; chapter representation contains hyperlinks to its pages...). API consumer obtains the VDC representation as its following point, which in turn provides hyperlinks that directly or indirectly take to other resources like Services and/or Servers.

PaaS Manager API entities provide an instance identifier property (instance ID). This property is used to identify unambiguously the entity but not the REST resource used to manage it, which is identified by its URL as described above (although this URL can contain the instance ID). It is common that most implementations make use of instance ID to compose the URL (e.g., the book with instance ID 1492 could be represented by resource http://.../book/1492), but such an assumption should not be taken by API consumer to obtain the resource URL from its instance ID.

11.2.6 Links and References

Resources often lead to refer to other resources. In those cases, we have to provide an ID or an URL to a remote resource.

11.2.7 Paginated Collections (Optional)

In order to reduce the load on the service, we can decide to limit the number of elements to return when it is too big. In order to do it, we use the limit, which is the maximum number of element to return, and last parameter, which was the last element to see. In the response JSON, we include an atom "next" links to follow to the next group of data. The last page in the list will not contain a "next" link. If there is an over limit error, the API will return a 413 message (over limit error) or 404 message (item not found error).

11.2.8 Efficient Polling with the Changes-Since Parameter (Optional)

In this case we can specify the parameter changes-since in a GET method in order that the response will give us only the changed information from the previous request specified through a dateTime format ISO 8601 (2011-01-24T17:08Z).

11.2.9 Versions

This is the first version.

11.2.10 Faults

The error code is returned in the body of the response for convenience. The message section returns a human-readable message that is appropriate for display to the end user. The details section is optional and may contain information—for example, a stack trace—to assist in tracking down an error. The detail section may or may not be appropriate for display to an end user.

PaaS Manager API Faults

Fault Element	Associated Error Codes	Description
Fault	500, 400, other codes possible	Error in the operation
serviceUnavailable	503	The service is not available
unauthorized	401	You are not authorized to access to that operation. The token is not correct.
forbidden	403	It is forbidden
badRequest	400	The request has not been done correctly
badMediaType	415	The payload media is not correct
itemNotFound	404	It is not exist

Besides the previous errors, when there is a error in the PaaS Manager operation, the PaaS Managerprints this error in the task result although the request error code is 200. In this case, the task status is ERROR and there is a message indicating its error message.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task
href="http:/130.206.80.112:8080/paasmanager/rest/org/{org-
id}/vdc/{vdc-id}/task/{task-id}" startTime="2012-11-
08T09:13:18.311+01:00" status="ERROR">
  <description>Deploy environment {emvironment-
name}</description>
  <vdc>{vdc-id}</vdc>
</task>
```

In this case the main controlled exceptions are:

1. **InvalidApplicationReleaseException**: The application release provided in the payload is not correct.
2. **InvalidEnvironmentRequestException**: The environment information is not right.
3. **InvalidProductInstanceRequestException**: The information about the product instance is created incorrectly
4. **EnvironmentInstanceNotFoundException**: The environment instance is not found.
5. **ApplicationInstanceNotFoundException**: The application instance does not exist
6. **ProductReleaseNotFoundException**: The product release is not found.
7. **ProductInstallatorException**: There is a problem to install the product
8. **ApplicationInstallatorException**: There is a problem to install the application
9. **IPNotRetrievedException**: There is a problem to obtain the IP from the VM
10. **VMStatusNotRetrievedException**: There is a problem to obtain the status of the VM
11. **InfrastructureException**: There is a problem with a operation in the IaaS level
12. **PaasManagerServerRuntimeException**: There is an problem in the PaaS Manager works

11.3 API Operations

In this section we go in depth for each operation. These operations have been described in the PaaS Manager Architecture. The FIWARE programmer guide will also provide examples of how to use the API. In this page, only XML examples are going to be provided.

The PaaS Manager API is divided among two functionalities:

- The catalogue of available environments to be deployed.
- The provisioning of environments and applications.

Due to it, we make two main divisions in the API explanation.

11.3.1 Environment Catalogue

It involves the catalogue of environments to be deployed.

11.3.1.1 *List Environments in the catalogue*

Verb	URI	Description
GET	*/paasmanager/rest/catalog/environment*	Lists all Environments in the catalogue.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This operation does not require a request body.

This operation lists the environments stored in the catalogue. The following example shows an XML response for the list Environment API operation. It is possible to see it contains a list of tiers including products to be installed.

```
<environmentDtoes>
  <environment>
  <name>{emvironment-name}</name>
     <tiers>
        <tier>

<initial_number_instances>1</initial_number_instances>

<maximum_number_instances>1</maximum_number_instances>

<minimum_number_instances>1</minimum_number_instances>
           <name>{tier-id}</name>
           <productReleases>
              <product>postgresql</product>
              <version>0.0.3</version>
              <withArtifact>true</withArtifact>
              <productType>
                 <id>5</id>
                 <name>Database</name>
              </productType>
           </productReleases>
                      ...
       </tier>
    </tiers>
  </environment>
  <environment>
  <name>{emvironment-name}</name>
     <tiers>
        <tier>
          ...
        <tier>
     </tiers>
  </environment>
</environmentDtoes>
```

11.3.1.2 *Add a Environment to the catalogue*

Verb	URI	Description
POST	*/paasmanager/rest/catalog/environment*	Add the Environment into the PaaS Manager's catalog. If the Environment already exists, then it updates it. If not, this method also creates the environment.

Normal Response Code(s): 200 Ok

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

This operation stores a new environment in the catalogue. This call creates a new environment in the catalogue and does not return any response body.

```
<?xml version="1.0" encoding="UTF-8"?>
<environment>
    <name>{environment-name}</name>
    <tiers>
        <tier>

<initial_number_instances>1</initial_number_instances>

<maximum_number_instances>1</maximum_number_instances>

<minimum_number_instances>1</minimum_number_instances>
           <name>{tier-id}</name>
           <productReleases>
              <product>postgresql</product>
              <version>0.0.3</version>
              <withArtifact>true</withArtifact>
              <productType>
                 <id>5</id>
                 <name>Database</name>
              </productType>
          </productReleases>
                     ...
        </tier>
    </tiers>
</environment>
```

The network and region information are including also in the payload of the enviornment. The following lines show a example.

```
<tier>
    <name>{tier-id}</name>
    <region>Spain</region>
    <network>Internet</network>
    <network>private_network</network>
    <productReleases>
       ...
    </productReleases>
</tier>
```

11.3.1.3 *Get Environment Details*

Verb	URI	Description
GET	*/paasmanager/rest/catalog/environment/{environment-id}*	Lists details of the specified environment.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

This operation does not require a request body and returns the details of a specific environment by its ID.

Environment Response: XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<environment>
    <name>{emvironment-name}</name>
    <tiers>
        <tier>

<initial_number_instances>1</initial_number_instances>

<maximum_number_instances>1</maximum_number_instances>

<minimum_number_instances>1</minimum_number_instances>
           <name>{tier-id}</name>
           <productReleases>
              <product>postgresql</product>
              <version>0.0.3</version>
              <withArtifact>true</withArtifact>
              <productType>
```

```
                    <id>5</id>
                    <name>Database</name>
                </productType>
            </productReleases>
        </tier>
        <tier>

<initial_number_instances>1</initial_number_instances>

<maximum_number_instances>5</maximum_number_instances>

<minimum_number_instances>1</minimum_number_instances>
            <name>{tier-id}</name>
            <productReleases>
                <product>tomcat</product>
                <version>7</version>
                <withArtifact>true</withArtifact>
                <productType>
                    <id>6</id>
                    <name>webserver</name>
                </productType>
            </productReleases>
        </tier>
    </tiers>
</environment>
```

11.3.1.4 *Modify Environment Details*

Verb	URI	Description
PUT	*/paasmanager/rest/catalog/environment/{environment-id}*	Modify the details of the specified environment.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

This operation requires the request payload containing the environment details and does not return an response entity.

Environment Request: XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<environment>
```

```
    <name>{emvironment-name}</name>
    <tiers>
        <tier>

<initial_number_instances>1</initial_number_instances>

<maximum_number_instances>1</maximum_number_instances>

<minimum_number_instances>1</minimum_number_instances>
           <name>{tier-id}</name>
           <productReleases>
              <product>postgresql</product>
              <version>0.0.3</version>
              <withArtifact>true</withArtifact>
              <productType>
                 <id>5</id>
                 <name>Database</name>
              </productType>
          </productReleases>
        </tier>
        <tier>

<initial_number_instances>1</initial_number_instances>

<maximum_number_instances>5</maximum_number_instances>

<minimum_number_instances>1</minimum_number_instances>
           <name>{tier-id}</name>
           <productReleases>
              <product>tomcat</product>
              <version>7</version>
              <withArtifact>true</withArtifact>
              <productType>
                 <id>6</id>
                 <name>webserver</name>
              </productType>
          </productReleases>
        </tier>
    </tiers>
```

```
</environment>
```

Environment Response: None

11.3.1.5 *Delete Environment in the Catalogue*

Verb	URI	Description
DELETE	*/paasmanager/rest/catalog/environment/{environment-id}*	Deletes the product with the id (environment-id) in the catalogue

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

It specifies the environment ID as *{environment-id}* in the URI. This operation does not require a request body and returns the details of a specific environment by its ID.

11.3.2 Environment Provisioning

These operations involve all the operations for deploying environments in the Cloud and its management.

11.3.2.1 *Deploy an environment*

Verb	URI	Description
POST	*/paasmanager/rest/org/{org-id]/vdc/{vdc-id}/environmentInstance*	Deploy an environment including both virtual machines and required software.

Normal Response Code(s): 200 (Ok), 203 (No Service available)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This call deploys all the servers required (according to the Tier information) for the vdc *{vdc-id}*. On top of these servers, it installs the software specified.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<environment>
  <name>{emvironment-name}</name>
     <tiers>
        <tier>

<initial_number_instances>1</initial_number_instances>

<maximum_number_instances>1</maximum_number_instances>

<minimum_number_instances>1</minimum_number_instances>
```

```
            <name>{tier-id}</name>
            <productReleases>
               <product>postgresql</product>
               <version>0.0.3</version>
               <withArtifact>true</withArtifact>
               <productType>
                  <id>5</id>
                  <name>Database</name>
               </productType>
            </productReleases>
                     ...
      </tier>
   </tiers>
</environment>
```

Environment Instance Response: XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task
href="http:/130.206.80.112:8080/paasmanager/rest/org/{org-
id}/vdc/{vdc-id}/task/{task-id}" startTime="2012-11-
08T09:13:18.311+01:00" status="RUNNING">
  <description>Deploy environment {emvironment-
name}</description>
  <vdc>{vdc-id}</vdc>
</task>
```

With the URL obtained in the href in the Task, it is possible to monitor the operation status (you can check Task Management). Once the environment has been deployed, the task status should be SUCCESS.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task
href="http:/130.206.80.112:8080/paasmanager/rest/org/{org-
id}/vdc/{vdc-id}/task/{task-id}" startTime="2012-11-
08T09:13:19.567+01:00" status="SUCCESS">
  <description>Deploy environment {emvironment-
name}</description>
  <vdc>{vdc-id}</vdc>
</task>
```

11.3.2.2 *Get information about the environments deployed (environment instance)*

Verb	URI	Description

GET	*/paasmanager/rest/org/{org-id}/vdc/{vdc-id}/environmentInstance*	Get information about the environments deployed

Normal Response Code(s): 200 (Ok), 203 (No Service available) Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This operation does not require a request body and returns the details of the list of environments instances deployed.

Environment Instance List Response: XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<environmentInstanceDtoes>
   <environmentInstance>
      <environmentInstanceName>{environmentInstance-
id</environmentInstanceName>
      <vdc>{vdc-id}</vdc>
      <environment>
         <name>{emvironment-name}</name>
         <tiers>
            <tier>

<initial_number_instances>1</initial_number_instances>

<maximum_number_instances>1</maximum_number_instances>

<minimum_number_instances>1</minimum_number_instances>
               <name>{tier-id}</name>
               <productReleases>
                  <product>postgresql</product>
                  <version>0.0.3</version>
                  <withArtifact>true</withArtifact>
                  <productType>
                     <id>5</id>
                     <name>Database</name>
                     </productType>
               </productReleases>                          ...
            </tier>
        </tiers>
      </environment>
      <tierInstances>
```

```
            <id>35</id>
            <date>2012-10-31T09:24:45.298Z</date>
            <name>tomcat-</name>
            <status>INSTALLED</status>
            <vdc>{vdc-id}</vdc>
            <tier>
               <name>{tier-id}</name>
            </tier>
            <productInstances>
               <id>33</id>
               <date>2012-10-31T09:14:33.192Z</date>
               <name>postgresql</name>
               <status>INSTALLED</status>
               <vdc>{vdc-id}</vdc>
               <productRelease>
                  <product>postgresql</product>
                  <version>0.0.3</version>
               </productRelase>
               <vm>
                  <fqn>vmfqn</fqn>
                  <hostname>rehos456544</hostname>
                  <ip>109.231.70.77</ip>
               </vm>
            </tierInstances>
         </environmentInstance>
</environmentInstanceDtoes>
```

11.3.2.3 *Get details about an environment deployed*

Verb	URI	Description
GET	*/paasmanager/rest/org/{org-id}/vdc/{vdc-id}/environmentInstance/{environmentInstance-id}*	Get the details about the environments deployed

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

This operation does not require any payload in the request and provides a environmentInstance XML response.

EnvironmentInstances Response: XML

```
<environmentInstance>
   <environmentInstanceName>{environmentInstance-
id</environmentInstanceName>
   <vdc>{vdc-id}</vdc>
   <environment>
      <name>{emvironment-name}</name>
      <tiers>
         <tier>

<initial_number_instances>1</initial_number_instances>

<maximum_number_instances>1</maximum_number_instances>

<minimum_number_instances>1</minimum_number_instances>
            <name>{tier-id}</name>
            <productReleases>
               <product>postgresql</product>
               <version>0.0.3</version>
               <withArtifact>true</withArtifact>
               <productType>
                  <id>5</id>
                  <name>Database</name>
                  </productType>
            </productReleases>
                ...
         </tier>
     </tiers>
   </environment>
   <tierInstances>
      <id>35</id>
      <date>2012-10-31T09:24:45.298Z</date>
      <name>tomcat-</name>
      <status>INSTALLED</status>
      <vdc>{vdc-id}</vdc>
      <tier>
         <name>{tier-id}</name>
      </tier>
```

```
      <productInstances>
         <id>33</id>
         <date>2012-10-31T09:14:33.192Z</date>
         <name>postgresql</name>
         <status>INSTALLED</status>
         <vdc>{vdc-id}</vdc>
         <productRelease>
            <product>postgresql</product>
            <version>0.0.3</version>
         </productRelase>
        <vm>
            <fqn>vmfqn</fqn>
            <hostname>rehos456544</hostname>
            <ip>109.231.70.77</ip>
         </vm>
      </productInstance>
   </tierInstances>
</environmentInstance>
```

11.3.2.4 *Undeploy an Environment Instance*

Verb	URI	Description
DELETE	*/paasmanager/rest/org/{org-id}/vdc/{vdc-id}/environmentInstance/{environmentInstance-id}*	Undeploy the environment deployed in the testbed (involving both servers and software)

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

Specify the Environment Instance ID as *{environmentInstance-id}* in the URI. This operation does not require a request body and returns the details of a generated task.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task
href="http:/130.206.80.112:8080/paasmanager/rest/org/{org-
id}/vdc{vdc-id}/task/{task-id}" startTime="2012-11-
08T09:45:44.020+01:00" status="RUNNING">
  <description>Uninstall environment</description>
  <vdc>{vdc-id}</vdc>
</task>
```

With the URL obtained in the href in the Task, it is possible to monitor the operation status (you can checkTask Management). Once the environment has been undeployed, the task status should be SUCCESS.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task
href="http:/130.206.80.112:8080/paasmanager/rest/org/{org-
id}/vdc/{vdc-id}/task/{task-id}" startTime="2012-11-
08T09:13:19.567+01:00" status="SUCCESS">
  <description>Undeploy environment {emvironment-
name}</description>
  <vdc>{vdc-id}</vdc>
</task>
```

11.3.3 Application Provisioning on top of an Environment

These operations involve all the operations for deploying applications on top of environment already deployed in the Cloud and its management.

11.3.3.1 ***Deploy an application***

Verb	URI	Description
POST	*/paasmanager/rest/org/{org-id]/vdc/{vdc-id}/environmentInstance/{environmentInstance-id}/applicationInstance*	Deploy an application (together their artefacts on top of an environment already deployed)

Normal Response Code(s): 200 (Ok), 203 (No Service available)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This call deploys an application on top of an environment already deploying, including the deployment of all the artefacts in the associated product releases. The request involves information about the Application to be deployed mainly composed by a name, version and a set of artefacts. Current example shows the deployment of a java web application which runs on top of a Tomcat and PostgreSQL. Mainly, its instantiation involves the deployment of a war and a properties file on top of the tomcat and the execution of a .sql script in the PostgreSQL.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<applicationReleaseDto>
  <applicationName>{application-name}</applicationName>
  <version>{application-version}</version>
  <artifacts>
     <artifact>
        <name>thewarfile</name>
        <attributes>
```

```
<key>webapps_url</key><value>http://Artefacts/WAR/tomcatFixedL
ocalPostgresDB/flipper.war</value>
            <key>webapps_name</key><value>flipper.war</value>
         </attributes>
         <productRelease>
            <version>7.0</version>
            <product>tomcat</product>
         </productRelease>
      </artifact>
      <artifact>
         <name>the properties file</name>
         <attributes>

<key>properties_url</key><value>http://configuration.prperties
</value>

<key>properties_name</key><value>flipper.war</value>
         </attributes>
         <productRelease>
            <version>7.0</version>
            <product>tomcat</product>
         </productRelease>
      </artifact>
      <artifact>
         <name>a .sql script</name>
         <attributes>

<key>sql_script_url</key><value>http://script.sql</value>
         </attributes>
         <productRelease>
            <version>1.2</version>
            <product>postgreSQL</product>
         </productRelease>
      </artifact>
   </artifacts>
<applicationReleaseDto>
```

The result for provisioning the application is a Task indicating the operation status:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<task
href="http:/130.206.80.112:8080/paasmanager/rest/org/{org-
id}/vdc/{vdc-id}/task/{task-id}" startTime="2012-11-
08T09:13:18.311+01:00" status="RUNNING">
  <description>Deploy application {application-
name}</description>
  <vdc>{vdc-id}</vdc>
</task>
```

With the URL obtained in the href in the Task, it is possible to monitor the operation status. Once the application has been deployed, the task status should be SUCCESS.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task
href="http:/130.206.80.112:8080/paasmanager/rest/org/{org-
id}/vdc/{vdc-id}/task/{task-id}" startTime="2012-11-
08T09:13:19.567+01:00" status="SUCCESS">
  <description>Deploy application {application-
name}</description>
  <vdc>{vdc-id}</vdc>
</task>
```

11.3.3.2 *Get information about the applications already deployed in a environment (application instance)*

Verb	URI	Description
GET	*/paasmanager/rest/org/{org-id]/vdc/{vdc-id}/environmentInstance/{environmentInstance-id}/applicationInstance*	Get information about the applications deployed in the environment

Normal Response Code(s): 200 (Ok), 203 (No Service available) Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This operation does not require a request body and returns the details of the list of application instances deployed.

Application Instance List Response: XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<applicationInstanceDtoes>
   <applicationInstance>
      <id>206</id>
      <date>2012-12-03T08:42:21.294+01:00</date>
      <name>{application-name}</name>
      <status>APPLICATION_INSTALLED</status>
```

```
      <vdc>{vdc_id}</vdc>
      <applicationRelease>
         <name>{application-name}</name>
         <version>{application-version}</version>
         <artifacts>
            <artifact>
               <name>thewarfile</name>
               <attributes>

<key>webapps_url</key><value>http://Artefacts/WAR/tomcatFixedL
ocalPostgresDB/flipper.war</value>

<key>webapps_name</key><value>flipper.war</value>
               </attributes>
               <productRelease>
                  <version>7.0</version>
                  <product>tomcat</product>
               </productRelease>
            </artifact>
            <artifact>
               <name>the properties file</name>
               <attributes>

<key>properties_url</key><value>http://configuration.prperties
</value>

<key>properties_name</key><value>flipper.war</value>
               </attributes>
               <productRelease>
                  <version>7.0</version>
                  <product>tomcat</product>
               </productRelease>
            </artifact>
            <artifact>
               <name>a .sql script</name>
               <attributes>

<key>sql_script_url</key><value>http://script.sql</value>
               </attributes>
               <productRelease>
```

```
                <version>1.2</version>
                <product>postgreSQL</product>
            </productRelease>
        </artifact>
    </artifacts>
</applicationRelease>
<environmentInstance>
    <id>202</id>
    <date>2012-11-30T13:40:55.941+01:00</date>
    <name>environmetntomcat</name>
    <status>INSTALLED</status>
    <vdc>{vdc-id}</vdc>
    <environment>
        <name>{environment_name}</name>
        <tiers>
            <name>tomcat</name>
                ...
    </environment>
    <tierInstances>
        <id>201</id>
        <date>2012-11-30T13:40:55.941+01:00</date>
        <status>INSTALLED</status>
        <tier>
            <name>tomcat</name>
            <productReleases>
                <product>tomcat</product>
                <version>7.0</version>
            </productReleases>
        </tier>
        <fqn>{environment_name}_tomcat fqn</fqn>
        <name>{environment_name}_tomcat</name>
        <productInstances>
            <id>200</id>
            <date>2012-11-30T13:40:50.786+01:00</date>
            <status>ARTEFACT_DEPLOYED</status>
            <productRelease>
                <product>tomcat</product>
                <version>7.0</version>
```

```
                </productRelease>
                <vm>

<fqn>4caast.customers.test4.services.testtomcatsap16.vees.tomc
at.replicas.1</fqn>
                   <hostname>relhot2345</hostname>
                   <ip>130.206.80.114</ip>
                </vm>
             </productInstances>
          </tierInstances>
       </environmentInstance>
    </applicationInstance>
</applicationInstanceDtoes>
```

11.3.3.3 *Get details about an application already deployed*

Verb	URI	Description
GET	*/paasmanager/rest/org/{org-id}/vdc/{vdc-id}/environmentInstance/{environmentinstance-id}/applicationInstance/{applicationInstance-id}*	Get the details about the application deployed

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

This operation does not require any payload in the request and provides an applicationInstance XML response.

ApplicationInstance Response: XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<applicationInstance>
   <id>206</id>
   <date>2012-12-03T08:42:21.294+01:00</date>
   <name>{application-name}</name>
   <status>APPLICATION_INSTALLED</status>
   <vdc>{vdc_id}</vdc>
      <applicationRelease>
         <name>{application-name}</name>
         <version>{application-version}</version>
         <artifacts>
            <artifact>
               <name>thewarfile</name>
```

```
                <attributes>

<key>webapps_url</key><value>http://Artefacts/WAR/tomcatFixedL
ocalPostgresDB/flipper.war</value>

<key>webapps_name</key><value>flipper.war</value>
                </attributes>
                <productRelease>
                   <version>7.0</version>
                   <product>tomcat</product>
                </productRelease>
             </artifact>
             <artifact>
                <name>the properties file</name>
                <attributes>

<key>properties_url</key><value>http://configuration.prperties
</value>

<key>properties_name</key><value>flipper.war</value>
                </attributes>
                <productRelease>
                   <version>7.0</version>
                   <product>tomcat</product>
                </productRelease>
             </artifact>
             <artifact>
                <name>a .sql script</name>
                <attributes>

<key>sql_script_url</key><value>http://script.sql</value>
                </attributes>
                <productRelease>
                   <version>1.2</version>
                   <product>postgreSQL</product>
                </productRelease>
             </artifact>
          </artifacts>
       </applicationRelease>
       <environmentInstance>
```

```
          <id>202</id>
          <date>2012-11-30T13:40:55.941+01:00</date>
          <name>environmetntomcat</name>
          <status>INSTALLED</status>
          <vdc>{vdc-id}</vdc>
          <environment>
             <name>{environment_name}</name>
             <tiers>
                <name>tomcat</name>
                      ...
          </environment>
          <tierInstances>
             <id>201</id>
             <date>2012-11-30T13:40:55.941+01:00</date>
             <status>INSTALLED</status>
             <tier>
                <name>tomcat</name>
                <productReleases>
                   <product>tomcat</product>
                   <version>7.0</version>
                </productReleases>
             </tier>
             <fqn>{environment_name}_tomcat fqn</fqn>
             <name>{environment_name}_tomcat</name>
             <productInstances>
                <id>200</id>
                <date>2012-11-30T13:40:50.786+01:00</date>
                <status>ARTEFACT_DEPLOYED</status>
                <productRelease>
                   <product>tomcat</product>
                   <version>7.0</version>
                </productRelease>
                <vm>

<fqn>4caast.customers.test4.services.testtomcatsap16.vees.tomc
at.replicas.1</fqn>
                   <hostname>relhot2345</hostname>
                   <ip>130.206.80.114</ip>
```

```
                </vm>
            </productInstances>
        </tierInstances>
    </environmentInstance>
</applicationInstance>
```

11.3.3.4 *Undeploy an Application Instance*

Verb	URI	Description
DELETE	*/paasmanager/rest/org/{org-id}/vdc/{vdc-id}/environmentInstance/{environmentinstance-id}/applicationInstance/{applicationInstance-id}*	Undeploy the application deployed in the environment instance

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

It specifies the Environment Instance ID as *{environmentInstance-id}* and the application instance ID as *{applicationInstance-id}* in the URI. This operation does not require a request body and returns the details of a generated task.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task
href="http:/130.206.80.112:8080/paasmanager/rest/org/{org-
id}/vdc{vdc-id}/task/{task-id}" startTime="2012-11-
08T09:45:44.020+01:00" status="RUNNING">
  <description>Uninstall application</description>
  <vdc>{vdc-id}</vdc>
</task>
```

With the URL obtained in the href in the Task, it is possible to monitor the operation status (you can check Task Management). Once the environment has been deployed, the task status should be SUCCESS.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task
href="http:/130.206.80.112:8080/paasmanager/rest/org/{org-
id}/vdc/{vdc-id}/task/{task-id}" startTime="2012-11-
08T09:46:44.020+01:00" status="SUCCESS">
  <description>Uninstall application</description>
  <vdc>{vdc-id}</vdc>
</task>
```

11.3.4 Task Management

Verb	URI	Description
GET	*/rest/vdc/{vdc-id}/task/{task-id}*	Get the status of a task.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), badMediaType (415)

This operation recovers the status of a task created previously. It does not need any request body and the response body in XML would be the following.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task href="http:/130.206.80.112:8080/sdc/rest/vdc/{vdc-
id}/task/{task-id}" startTime="2012-11-08T09:13:18.311+01:00"
status="SUCCESS">
  <description>Install product tomcat in  VM rhel-
5200ee66c6</description>
  <vdc>{vdc-id}</vdc>
</task>
```

The value of the status attribute could be one of the following:

Value	Description
QUEUED	The task is queued for execution.
PENDING	The task is pending for approval.
RUNNING	The task is currently running.
SUCCESS	The task is completed successfully.
ERROR	The task is finished but it failed.
CANCELLED	The task has been cancelled by user.

12 FIWARE OpenSpecification Cloud Monitoring

12.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.org and similar pages in order to understand the complete context of the FI-WARE project.

12.2 Copyright

Copyright © 2012-2014 by Telefónica I+D. All Rights Reserved.

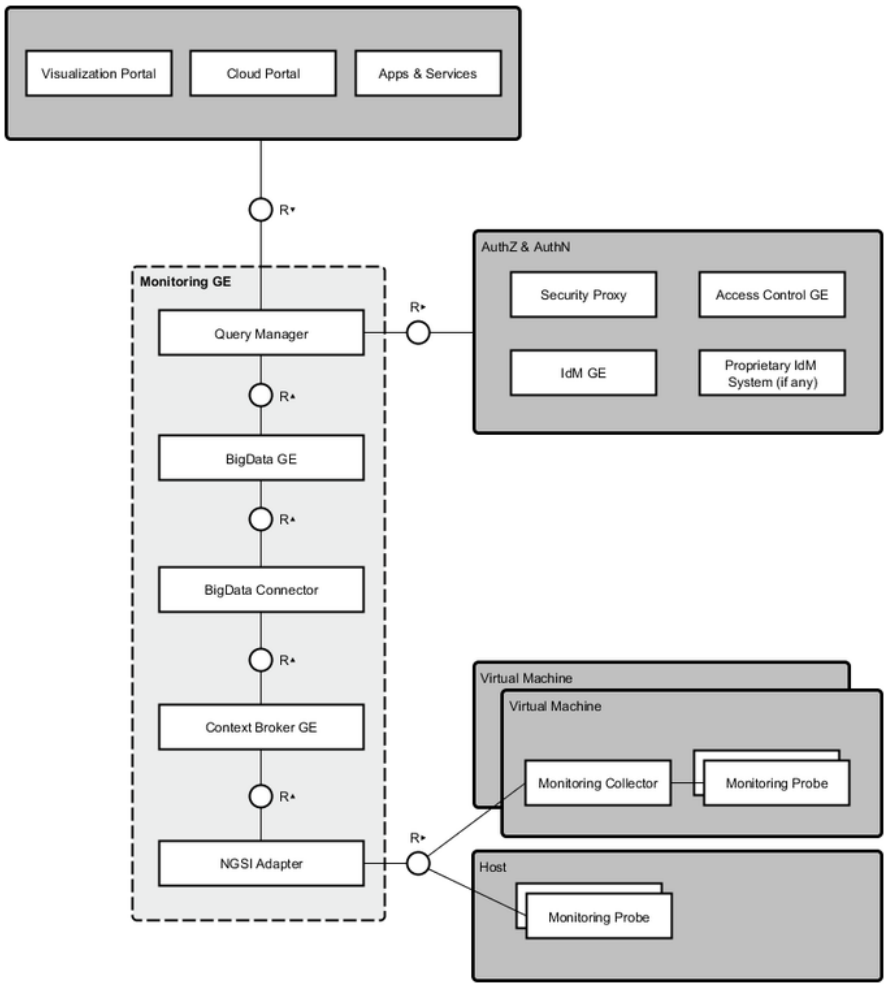
12.3 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

12.4 Overview

Every distributed system needs to incorporate monitoring mechanisms in order be able to constantly check the performance. Monitoring involves gathering operational data in a running system. There might be many consumers, which might use this information for various purposes. SLA management, where the system needs to be able to constantly check that the performance adheres to the terms signed, could be one of them. Monitoring data can also be used in a variety of ways, for example as optimization of virtual machines, products and applications, alarms detection, recommendations, etc.

This specification describes the Monitoring GE, which is the key enabler to provide monitoring information to the rest of GEs. Its architecture can be seen in the following figure:



Monitoring GE architecture overview

The Monitoring GE works once the resource has been deployed. The architecture of the Monitoring GE requires having monitoring probes distributed in the VMs and Hosts. This is valid for both IaaS or PaaS. This information is pushed to an adaptation layer either directly by the probes or through a custom ***monitoring collector*** responsible for that on behalf the probes.

Adaptation layer (*NGSI Adapter*) expresses raw monitoring data in terms of updates of entities' context, where the resources being monitored are such entities, and therefore sends requests to Context Broker GE. This GE holds the last available context of monitored entities, but lacks historical information. Through a connector component subscribed to Context Broker, every context update is written to BigData GE storage, thus building such historical information.

This information is offered to the cloud management enablers following both **pull** and **push** models. On the one hand, a *query manager* server implements a query API that allows gathering the last records of a measurement at different levels (vApps, VM, and even deployed software); besides, by using map-reduce mechanisms at BigData GE, we are able to aggregate certain measurements into higher-level data that can be used for more precise management of the resources (for instance, aggregating all the measurements from all the VMs hosting an application/service into a more meaningful KPI at the service level). On the other hand, it is still possible to subscribe to Context Broker notifications about context updates, thus being pushed with new monitoring data.

12.4.1 Target Usage

The monitoring system is used by different Cloud GEs in order to track the status of the resources. They use gathered data to take decisions about elasticity or for SLA management. Whenever a new resource is deployed in the cloud, the proper monitoring probe is set up and configured to start providing monitoring data. The GE offers a query interface to allow other GEs to poll for information relevant to any KPI associated to resources.

12.5 Main Concepts

Following the above FMC diagram of the Monitoring GE, in this section we introduce the main concepts related to this GE through the definition of their interfaces and components and finally and example of their use.

The key concepts visible to the cloud user could be differentiated between the interfaces and the components, each of them are described below.

12.5.1 Entities

The following entities are considered:

- **Measure**. A value that corresponds to the value of a metric for a resource in a given moment. These metrics are collected by monitoring probes, i.e., software which have been installed inside the VMs.
- **Source**. A probe that generates data for different metrics of a given resource.
- **Resource**. A cloud resource for which a cloud metric can be generated. It can apply to IaaS level resources and their aggregations (Organization, Project, vApp, VMs) and PaaS level resources (PIs and ACs).
- **Context**. In the NGSI vocabulary from Context Broker, a set of *attributes* (measures) from an *entity* (resource) in a given moment.

12.5.2 Interfaces

Two different models are supported for accessing monitoring data:

- **Pull model**. The client fetches monitoring data from resources using the Monitoring API offered by its query manager.
- **Push model**. The client subscribes to Context Broker GE (registering a callback URL) and receives notifications whenever new monitoring data is available.

12.5.3 Components

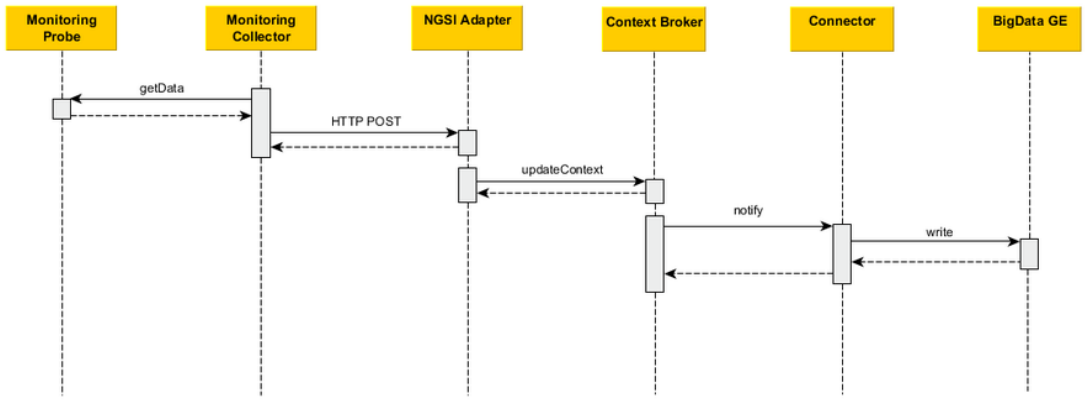
The Monitoring GE comprises a set of distributed components:

- **Monitoring Probes**, part of the software tool used to gather metrics. They are installed in the resource to be measured (virtual machine, physical node, etc.) and configured to provide the monitoring information. Such information has to be pushed to the Adapter. Monitoring GE should be agnostic to the concrete monitoring software chosen. Alternatives are Nagios, Zabbix, openNMS, perfSONAR, collectd, mBeanCmd, Ganglia, etc.

- **Monitoring Collector**: in case probes aren't able to push data directly to Adapter, then a custom component has to be deployed as part of the monitoring tool to gather probe data and issue HTTP requests to Adapter.
- **NGSI Adapter**: responsible for translating probe raw data into a common format (NGSI), and issuing update requests to Context Broker.
- **Context Broker GE**: publish/subscribe broker managing context updates.
- **BigData GE**: this is the storage system for saving the history of metrics for several years. These metrics have been organized according to the data model, that is, it will include also aggregated information.
- **BigData Connector**: subscriber of Context Broker resposible for writing context updates into storage.
- **Query Manager**: implementation of the Monitoring API (pull model).

12.5.4 Example Scenario

When a VM or server has been deployed, probes installed on it start sending monitoring data according to a defined schedule. Either directly or through a monitoring collector, data reach the adapter, which in turn publish them into Context Broker. BigData GE stores such data by means of a *connector* already subscribed to Context Broker for updates.



Monitoring GE example scenario

At this point, clients may use both querying modes:

1. The pull mode, by means of using the API implemented by Query Manager.
2. The push mode, by means of subscribing a client to Context Broker for updates.

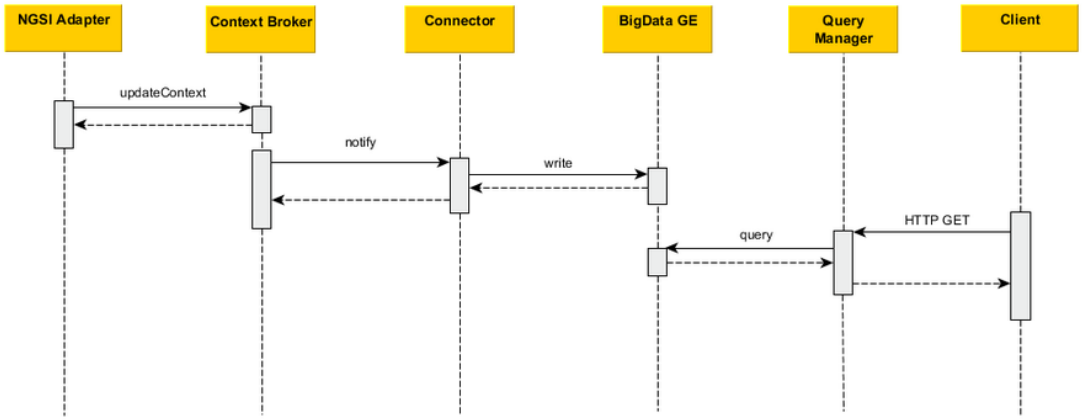
12.6 Main Interactions

The following section represents the main interactions in both modes. Concrete information about URL, schemas and so on can be seen on Monitoring Open RESTful API Specification (PRELIMINARY).

12.6.1 Pull model

This set of operations involves querying information about a measurable resource once the resource has been deployed. A measurable resource can be anything being measured. It can involve both a virtual machine and a physical node. Inside the

virtual machine, it can consider also software installed. Monitoring probes have to be installed at the monitored resource, in order to generate a set of metrics. Using the operations of the query API, it is possible to obtain this information.



Monitoring GE pull model

12.6.1.1 *Query metrics for all resources of a given type*

Client obtains the list of metrics for all available monitored resources of a given type

- INPUT: resource type (corresponding to NGSI *entityType*)
- OUTPUT: list resources and their metrics

12.6.1.2 *Query metrics for at most n resources of a given type*

Client obtains the list of metrics for at most `n` monitored resources of a given type

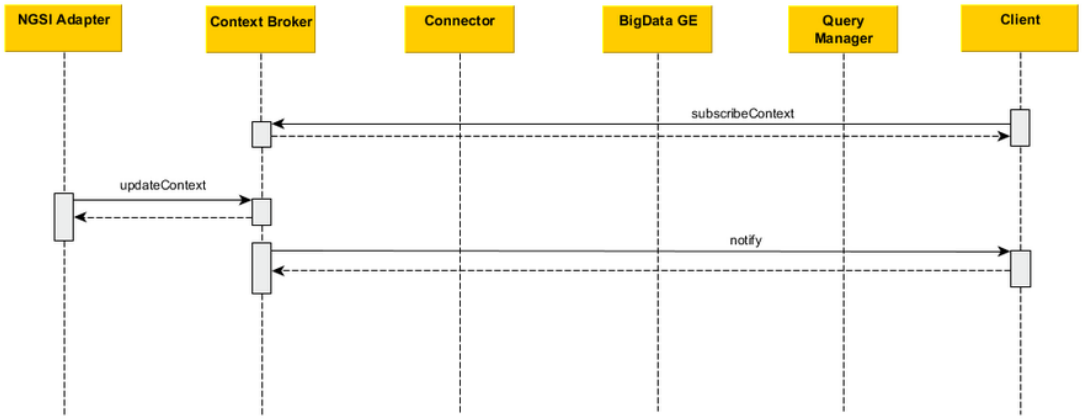
- INPUT: resource type (corresponding to NGSI *entityType*); the maximum number `n` of results
- OUTPUT: list resources and their metrics

12.6.1.3 *Query metrics for a specific resource*

Client obtains the list of metrics for a specific resource

- INPUT: resource type (corresponding to NGSI *entityType*) and resource identifier (NGSI *entityId*)
- OUTPUT: resource and its metrics

12.6.2 Push model



Monitoring GE push model

12.6.2.1 *Subscribe a client*

It subscribes a client to Context Broker for updates, providing a callback URL to receive notifications (see more details in Context Broker User and Programmers Guide)

- INPUT: `subscribeContext` request specifying: entities to subscribe to, attributes, notify conditions and callback URL
- OUTPUT: `subscribeContext` response including `subscriptionId`

12.6.2.2 *Process notification*

Client processes input data on Context Broker notification

- INPUT: `notifyContext` request including `subscriptionId` and context elements
- OUTPUT: none

12.7 Basic Design Principles

12.7.1 Design Principles

This section specifies a set of requirements for the Cloud monitoring framework:

- ***Non-intrusiveness on resource functionality and performance***: monitoring needs to be as non-intrusive as possible. This means that, although monitoring probes are installed within the monitored resources, they should not affect neither the rest of resource functionality nor performance.
- ***Deal with metric heterogeneity***: monitoring system has to deal with different kind of metrics (infrastructure, KPI, applications and product metrics), different virtualization technologies, different products, applications, etc.
- ***Scalability in monitored resources***: the system needs to be able to scale to large numbers of monitored nodes and resources.
- ***Scalability in number of measures***: when the number of monitored resources increases, the number of information to be included in the storage also increases. Taking into account that we collect information frequently (say

at a 5 seconds rate) and we store it for keeping historical data, a normal relational database can fail.

- ***Data aggregation***: monitoring system should be able to aggregate the information at application/service level, which means that it has to be able to aggregate metrics from VMs or hardware resources at service level.

12.7.2 Resolution of Technical Issues

Several approaches can be adopted to convey the design principles described in the previous section. Here, some of them are suggested:

- ***Probes for non-intrusive solution***: the approach would consist in introducing probes installed in the same virtual machines or host where the resource to be measure is placed, but not being part of the resource software itself.
- ***Collector for solving heterogeneity in metrics***: the approach would consist in including a monitoring collector component as part of the architecture, which is in charge of collecting the different metrics from the different probes.
- ***BigData storage for metrics scalability***: a promising approach would consist in gaining scalability in the number of metrics handled by means of using BigData GE as storage.
- ***BigData GE for data analysis***: the approach would consist in using map-reduce techniques to perform data aggregation.

12.8 Detailed Specifications

12.8.1 Open API Specifications

Following is a list of Open Specifications linked to this Generic Enabler. Specifications labeled as "PRELIMINARY" are considered stable but subject to minor changes derived from lessons learned during last interactions of the development of a first reference implementation planned for the current Major Release of FI-WARE. Specifications labeled as "DRAFT" are planned for future Major Releases of FI-WARE but they are provided for the sake of future users.

- Monitoring Open RESTful API Specification (PRELIMINARY)

12.9 Re-utilised Technologies/Specifications

The Monitoring Manager GE is based on RESTful Design Principles. The technologies and specifications used in this GE are:

- RESTful web services
- HTTP/1.1 (RFC2616)
- JSON and XML data serialization formats.

12.10 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be helpful to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FI-WARE Global Terms and Definitions

- **Runtime Execution Container** (**REC**) -- a VM with all the software stack required to deploy a complete node in an application architecture. It usually comprises one or more VMs, middleware, monitoring probes, and Chef client and SDC agent to support installation and configuration.
- **Software Deployment and Configuration GE** (**SDC GE**) -- a GE in devoted to the automated installation and configuration of software on VMs through the execution of recipes in the corresponding nodes. It relies on Opscode Chef technology.
- **Software Deployment and Configuration Interface** (**SDCI**) -- the interface offered by the SDC GE to be managed by the PaaS Manager or a Cloud Portal.
- **Product Instance** (**PI**) -- an installed software in a VMs, usually referring to middleware or platform software (E.g.: Apache Tomcat, MySQL, etc.).
- **Application Component** (**AC**) -- a component (or configuration artifact) that implements usually one or more components of an application architecture. These ACs are installed on, and differentiated from, existing PIs.
- **PaaS Manager Interface** (**PMI**) -- the interface offered by the PaaS Manager GE to be used by a Cloud Portal to manage both the catalogue and the lifecycle of the platform resources and applications.

13 Monitoring Open RESTful API Specification

13.1 Introduction to the Monitoring API

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

In a Cloud, there is a set of heterogeneous components will have to be monitored, including infrastructure elements, products and applications. To access monitoring services inside FIWARE, a TCloud REST-based API is proposed. TCloud is a Cloud-oriented API released by Telefónica and submitted to the DMTF [1] to be incorporated in the CIMI specification. TCloud API is based on vCloud API specification 0.8 [2], as published by VMware. In essence, compatibility for the main operations and data types defined in vCloud are maintained in TCloud, but it provides extensions on advanced Cloud Computing management capabilities including additional shared storage for service data, network element provisioning (different flavours of load balancers and firewalls), monitoring, snapshot management, and so on.

TCloud is focused on adding network intelligence, reliability and security features to Cloud Computing empowered by enhanced Telecom network integration. Moreover, TCloud aims to extend current Cloud Computing models providing more flexibility and control to Cloud Computing customers. TCloud provides extensions on advanced Cloud Computing management capabilities including additional shared storage for service data, network element provisioning (different flavours of load balancers and firewalls), monitoring, snapshot management, and so on. TCloud API offers RESTful web services: it is resource oriented, accessed via HTTP, and using XML-based representations for information interchange.

13.1.1 Monitoring API Core

The Monitoring API is a RESTful, resource-oriented API accessed via HTTP/HTTPS that uses XML-based representation for information interchange. This API, as described here, will be used basically to get information about monitored metrics, adding a subscription interface. However, it is considered interesting to offer access to the advanced analysis capabilities to be leveraged inside FI-WARE; with that in mind, some basic extensions to TCloud will be proposed at this time.

13.1.2 Intended Audience

This specification is intended for both software developers and implementers of this API. For the former, this document provides a full specification of how to interoperate with Cloud Platforms that implements Monitoring API. For the latter, this specification indicates the interface to be provided to clients to interoperate with the Monitoring GE within the Cloud Platform to provide the described functionalities. To use this information, the reader should first have a general understanding of the Monitoring GE and also be familiar with:

- RESTful web services
- HTTP/1.1 (RFC2616)
- JSON and/or XML data serialization formats.

13.1.3 API Change History

This version of the Monitoring API Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
09/11/2012	• First version of the Monitoring API based on TCloud specification

13.1.4 How to Read This Document

In the whole document the assumption is taken that the reader is familiarized with REST architecture style. Along the document, some special notations are applied to differentiate some special words or concepts. The following list summarizes these special notations.

- A **bold**, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).
- An *italic* font is used to represent document titles or some other kind of special text, e.g., *URI*.
- The variables are represented between brackets, e.g. {id} and in italic font. When the reader find it, can change it by any value.

For a description of some terms used along this document, see [3].

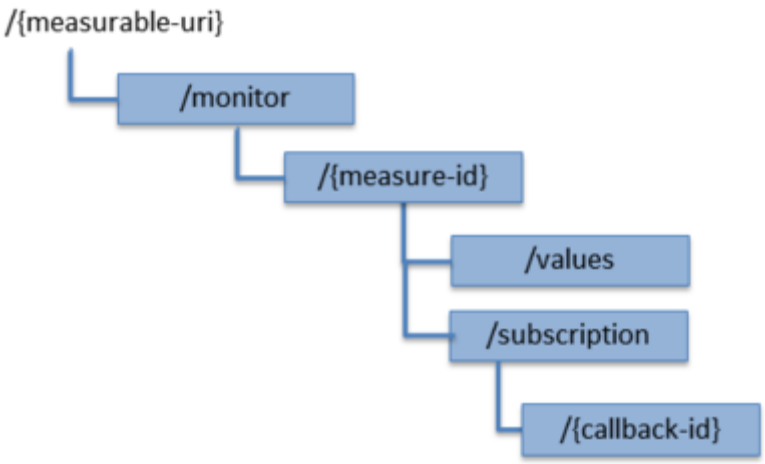
13.1.5 Additional Resources

You can download the most current version of this document from the FI-WARE API specification selecting **PDF Version** from the Toolbox menu (left side), which will generate the file to download it. For more details about the **Monitoring** that this API is based upon, please refer to Cloud Hosting. Related documents, including an Architectural Description, are available at the same site.

13.2 General Monitoring API Information

13.2.1 Resources Summary

A diagram in which the different Uniform Resource Names (URNs) that can be used in the API is shown here. The URL is http://{serverRoot}:{serverPort} and the main root element is the measurable-uri is any entity which can be measure, like a virtual machine, a service, a product and so on.



Monitoring Open RESTful API resource summary

13.2.2 Authentication

No defined yet.

13.2.3 Representation Format

The Monitoring API resources are represented by hypertext that allows each resource to reference other related resources. More concisely, XML is used for resource representation and URLs are used for referencing other resources by default. The request format is specified using the Content-Type header and is required for operations that have a request body. The response format can be specified in requests using either the Accept header with values application/xml or adding an .xml extension to the request URI. The following lines show the examples about it.

```
GET /{measure-uri}/monitor HTTP/1.1

Host: api.monitoring.org

Content-Type: application/xml

Accept: application/xml

X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

13.2.4 Representation Transport

Resource representation is transmitted between client and server by using HTTP 1.1 protocol, as defined by IETF RFC-2616. Each time an HTTP request contains payload, a Content-Type header shall be used to specify the MIME type of wrapped representation. In addition, both client and server may use as many HTTP headers as they consider necessary.

13.2.5 Resource Identification

API consumer must indicate the resource identifier while invoking a GET, PUT, POST or DELETE operation. Monitoring API combines both identification and location by terms of URL. Each invocation provides the URL of the target resource along the verb and any required input data. That URL is used to identify unambiguously the resource. For HTTP transport, this is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

Monitoring API does not enforce any determined URL pattern to identify its resources. Anyway the Monitoring API extension follows the HATEOAS principle (Hypermedia As The Engine Of Application State). This means that resource representation contains the URLs of the related resources (e.g., book representation contains hyperlinks to its chapters; chapter representation contains hyperlinks to its pages...). API consumer obtains measure and values as its following point, which in turn provides hyperlinks that directly or indirectly take to other resources like measure resources.

Some Monitoring API entities provide an instance identifier property (instance ID). This property is used to identify unambiguously the entity but not the REST resource used to manage it, which is identified by its URL as described above. It is common that most implementations make use of instance ID to compose the URL (e.g., the book with instance ID 1492 could be represented by resource http://.../book/1492), but such an assumption should not be taken by API consumer to obtain the resource URL from its instance ID.

13.2.6 Links and References

Resources often lead to refer to other resources. In those cases, we have to provide an ID or an URL to a remote resource.

13.2.7 Paginated Collections (Optional)

n.a.

13.2.8 Efficient Polling with the Changes-Since Parameter (Optional)

In this case we can specify the parameter changes-since in a GET method in order that the response will give us only the changed information from the previous request specified through a dateTime format ISO 8601 (2011-01-24T17:08Z).

13.2.9 Limits

n.a

13.2.10 Versions

This is the first version.

13.2.11 Extensions

No apply yet.

13.2.12 Faults

The error code is returned in the body of the response for convenience. The message section returns a human-readable message that is appropriate for display to the end user. The details section is optional and may contain information—for example, a stack trace—to assist in tracking down an error. The detail section may or may not be appropriate for display to an end user.

Monitoring API Faults

Fault Element	Associated Error Codes	Description
Fault	500, 400, other codes possible	Error in the operation
serviceUnavailable	503	The service is not available
unauthorized	401	You are not authorized to access to that operation. The token is not correct.
forbidden	403	It is forbidden
badRequest	400	The request has not been done correctly
badMediaType	415	The payload media is not correct
itemNotFound	404	It is not exist
badMethod	405	Method not allowed
overLimit	413	Request entity too large

13.3 API Operations

In this section we go in depth for each operation. In order to provide good comprehensive of the API operations, these operations were described in the Monitoring Manager Architectural Specification. The FI-WARE programmer guide will also provide examples on how to use the API. The operations, which are not described here are under discussion and will be included in the following releases.

This relates to the operations admitted by cloud resources for monitoring actions. Each operation is described using the same notation as TCloud API core operations. URIs are abbreviated using the <item-uri> form, where item may be an organization, a VDC, a service, a database instance… in summary, any computing resource which can be potentially measured.

Measurable resource

```
<measurable-uri> ::= <vdc-uri> | <vapp-uri> | <hwitem-uri> |
<net-uri> | …
```

Measurable resource with monitoring abilities

```
<measure-uri> ::= <measurableuri>/monitor/<measure-id>
```

The monitoring operations have two main purposes or two working modes:

- a mode to obtain monitoring information
- a subscription mode

13.3.1 Monitoring mode

13.3.1.1 *List metrics of a measurable resource*

Verb	URI	Description
GET	*<measurable-uri>/monitor*	Obtains the list of available measures for a resource.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This operation does not require a request body and lists the metrics associated to a measurable resource, in the MeasureDescriptorList element.

The following examples show an MeasureDescriptorList XML response:

```
<MeasureDescriptionList
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/{measurable-
uri}/monitor">
   <Link
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/{measurable-uri}"
rel="up" type="application/vnd.telefonica.tcloud.vapp+xml"/>
   <MeasureDescriptor
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/{measurable-
uri}/monitor/diksUsed" name="diksUsed">
      <Link
href="http://{IP_MONITORING_SYSTEMS}:PORT}/{measurable-
uri}/monitor/diksUsed/subscription"   rel="monitor:subscribe"
type="application/vnd.telefonica.tcloud.monitoringCallback+pla
in"/>
      <Link
href="http://{IP_MONITORING_SYSTEMS}:PORT}/{measurable-
uri}/monitor/diksUsed/values"  rel="monitor:poll"
type="application/vnd.telefonica.tcloud.measure+xml"/>
      <ValueType>bytes</ValueType>
      <MinValue>0</MinValue>
      <MaxValue>100</MaxValue>
      <Description>diksUsed</Description>
   </MeasureDescriptor>
```

```
   <MeasureDescriptor
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/{measurable-
uri}/monitor/requestDelay" name="diskFree">
      <Link
href="http://{IP_MONITORING_SYSTEMS}:PORT}/{measurable-
uri}/monitor/requestDelay/subscription"
rel="monitor:subscribe"
type="application/vnd.telefonica.tcloud.monitoringCallback+pla
in"/>
      <Link
href="http://{IP_MONITORING_SYSTEMS}:PORT}/{measurable-
uri}/monitor/requestDelay/values" rel="monitor:poll"
type="application/vnd.telefonica.tcloud.measure+xml"/>
      <ValueType>seconds</ValueType>
      <MinValue>0</MinValue>
      <MaxValue>1000</MaxValue>
      <Description>requestDelay</Description>
   </MeasureDescriptor>
</MeasureDescriptionList>
```

13.3.1.2 *Obtain information about a metric*

Verb	URI	Description
GET	*<measurable-uri>/monitor/<measure-id>*	Obtains the description for a measure.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This operation does not require a request body and describe the metric.

The following examples show an MeasureDescriptor XML response for getting information about the requestDelay metrics shown in the example before:

```
<MeasureDescriptor
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/{ID_MEASURABLE_RES
OURCE}/monitor/requestDelay" name="requestDelay">
   <Link
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/{ID_MEASURABLE_RES
OURCE}/monitor/requestDelay/subscription"
rel="monitor:subscribe"
type="application/vnd.telefonica.tcloud.monitoringCallback+pla
in"/>
   <Link
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/{ID_MEASURABLE_RES
```

```
OURCE}/monitor/requestDelay/values" rel="monitor:poll"
type="application/vnd.telefonica.tcloud.measure+xml"/>
   <ValueType>miliseconds</ValueType>
   <MinValue>0</MinValue>
   <MaxValue>0</MaxValue>
   <Description>requestDelay</Description>
</MeasureDescriptor>
```

13.3.1.3 *Obtain Metric Value*

Verb	URI	Description
GET	*<measurable-uri>/monitor/<measure-id>/values*	Obtains values for a measure.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This operation does not require a request body and provides the metric values.

The following example show a MeasureValues XML response:

```
<Measure
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/{measurable-
uri}/monitor/requestDelay/values">
   <Sample timestamp="2012-11-12T10:39:04" unit="miliseconds"
value="100"/>
</Measure>
```

In addition, adding the ?samples, it is possible to specify the number of values it is required. In the following example, we are asking for 3: http://{IP_MONITORING_SYSTEMS}:{PORT}/{measurable-uri}/monitor/requestDelay/values?samples=3

```
<Measure
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/{measurable-
uri}/monitor/requestDelay/values">
   <Sample timestamp="2012-11-12T10:40:14" unit="miliseconds"
value="100"/>
   <Sample timestamp="2012-11-12T10:40:04" unit="miliseconds"
value="98.9"/>
   <Sample timestamp="2012-11-12T10:39:54" unit="miliseconds"
value="97.0"/>
</Measure>
```

13.3.2 Subscription Mode

This mode allows having an asyncronous way to obtain the information. Thus, the user is subscribed to a concrete measure obtained a callback (or callback id in the API). Through this callback it is possible to obtain the monitoring information.

13.3.2.1 ***Register a subscription***

Verb	URI	Description
POST	*<measure-uri>/subscription*	Subscribes a new callback to monitor given measure.

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This operation does not require the URL for returning the callback in the request.

```
http://example.com/monitoring/do.ReceiveMeasure
```

The response is a MonitoringSubscription element, which its information.

```
<MonitoringSubscription
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/<measure-
uri>/monitor/{measure-id}/subscription/1" status="accepting">
   <Link rel="up"
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/<measure-
uri>/monitor/{measure-id}" />
   <Callback
href="http://example.com/monitoring/do.ReceiveMeasure"
type="application/vnd.telefonica.tcloud.measure+xml" />
</MonitoringSubscription>
```

13.3.2.2 ***Obtain details for a Subscription***

Verb	URI	Description
GET	*<measure-uri>/subscription/<callback-id>*	Obtains detailed information for a previously subscribed callback (identified by the callback-id).

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This operation does not require a request body.

The following examples show a MonitoringSubscription XML response:

```
<MonitoringSubscription status="accepting">
   <Link rel="up"
href="http://{IP_MONITORING_SYSTEMS}:{PORT}/<measure-
uri>/monitor/{measure-id}" />
   <Callback
href="http://example.com/monitoring/do.ReceiveMeasure"
type="application/vnd.telefonica.tcloud.measure+xml" />
</MonitoringSubscription>
```

13.3.2.3 ***Unsubcribe a metric***

Verb	URI	Description
DELETE	*<measure-uri>/subscription/<callback-id>*	Removes a previously registered subscription to monitor given measure deleting its callback (callback-id).

Normal Response Code(s): 200, 203

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), overLimit (413), serviceUnavailable (503), itemNotFound (404)

This operation does not require a request body.

14 FIWARE OpenSpecification Cloud PolicyManager

14.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.org and similar pages in order to understand the complete context of the FI-WARE project.

14.2 Copyright

Copyright © 2012 by Telefónica I+D. All Rights Reserved.

14.3 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

14.4 Overview

This specification describes the Policy Manager GE, which is a key enabler to scalability and to manage the cloud resources based on defined policies or rules.

The Policy Manager GE provides the basic management of cloud resources based on rules, as well as management of the corresponding resources within the FI-WARE Cloud Instance like actions based on physical monitoring or infrastructure, security monitoring of resources and services or whatever that could be defined by a facts, actions and rules.

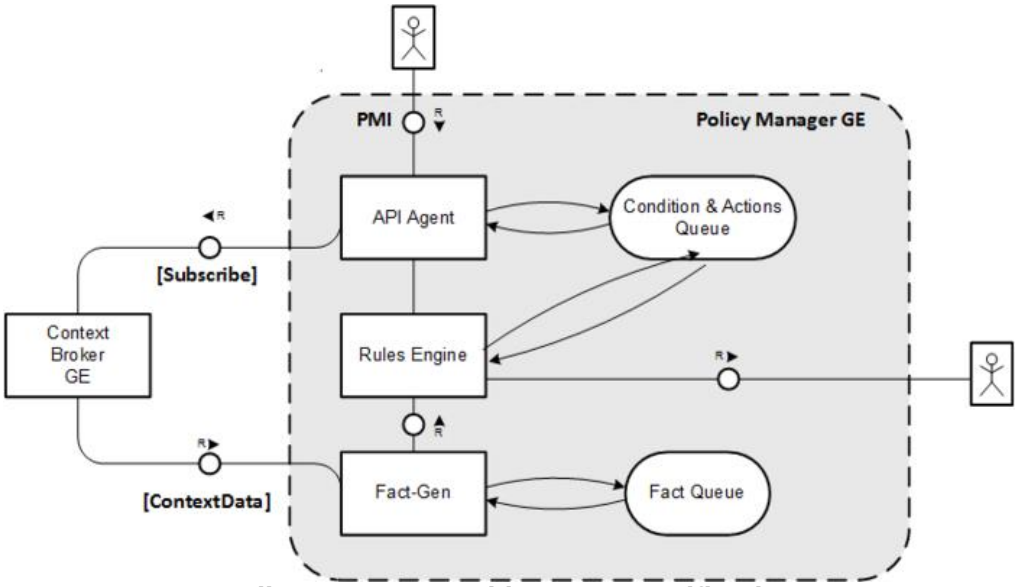
The baseline for the Policy Manager GE is PyCLIPS, which is a module to interface CLIPS expert system and python language. The reason to take PyCLIPS is to extend the OpenStack ecosystem with a expert system written in the same language that the rest of the OpenStack services. Hence, Policy Manager offers the decision-making ability, independently of the type of resource (physical/virtual resources, network, service or whatever), able to solve complex problems within the Cloud field by reasoning about the knowledge base, represented by facts and rules.

The main functionality that the Policy Manager GE provides is:

- Management of scalability rules. It is possible to manage rules whose target is not to scale and this is also included in the main functionality of component.
- Management of different facts related to virtual machines and other facts in order to launch actions from the rules whose conditions are met.

The Policy Manager needs interaction with the user who provides the specification of the rules and actions that compound the knowledge system following a CLIPS language format. The facts are received from any producer of information that monitors the different resources of the cloud system. Context Broker GE, like publish/subscribe/notify system, interacts with the Policy Manager GE to suscribe to the information (facts) of Virtual Machines or whatever in order to get updated usage status of resources (ej. cpu, memory, or disk) or resources that we want to monitor.

These facts are used by the inference engine to deduce new facts based on the rules or infer new actions to take by third parties.



Policy Manager architecture specification

14.4.1 Target Usage

The Policy Manager GE is an expert system that provides independent server in the OpenStack ecosystem which evaluates the current state of the knowledge-base, applied the pertinent rules and infers new knowledge into the knowledge-base. Currently, the actions are designed to scale up and down Virtual Machines according to facts received from them (memory, cpu, disk or whatever). There are more kind of usage for these rules and is the user who defines conditions and actions he wants for. It is the user when specify the rule and actions who specify which is the use that we want to give to this GE.

14.5 Main concepts

Following the above FMC diagram of the Policy Manager, in this section we introduce the main concepts related to this GE through the definition of their interfaces and components and finally an example of their use.

14.5.1 Basic Concepts

The Policy Manager manages a set of rules which throws actions when certain conditions are activated when some facts are received. These rules can be associated with a specific virtual machine or be a general rule that affects the entire system. The key concepts, components and interfaces associated to the Policy Manager GE and visible to the cloud user, are described in the following subsections.

14.5.2 Entities

The main entities managed by the Policy Manager are as follows:

- **Rules**. They represent the policy that will be used to infer new facts or actions based on the facts received from the Context Broker GE. Usually, rules are some type of statement of the form: if <x> then <y>. The if part is the rule premise or antecedent, and the then part is the consequent. The rule fires when the if part is determined to be true or false. They are compound of 2 types of rules:
  - **General Rules**. They represent a global policy to be considered regardless specific virtual machines. Each rule is compound of a name to identify it and the condition and action which is fired. GeneralRules entities are represented as RuleModel.
  - **Specific Rules**. They represent a policy associated to a specific virtual machine. SpecificRules entities are represented as SpecificRuleModel.
- **Information**. It represent the information about the Policy Manager API and tenant information. Tenant information contains the window size, a modificable value for manage the minimal number of measures to consider a real fact for Rules Engine.
- **Facts**. They represent the measurement of the cloud resources and will be used to infer new facts or actions. an average of measures from a virtual machine trough the Context Broker GE. The are the base of the reasoning process.
- **Actions**, They are the output of the knowledge system related to a sense input and the are the implementation of the response rule or consequent.

14.5.3 Interfaces

The Policy Manager GE is currently composed of two main interfaces:

- **The Policy Manager interface (PMI)** is the exposed REST interface that implements all features of the Policy Manager exposed to the users. The PMI allows to define new rules an actions together with the activation of a specific rule asociated to a resource. Besides, this interface allow to get the information about this GE (url documentation, windows size, owner and time of the last server start). Besides, the PMI implements the NGSI-10 interface in order to receive the facts provided by Context Broker (notification of the context data) related to a virtual server.
- **Context Broker Manager Interface (NGSI)** is invoked in order to subscribe the Policy Manager to a specific monitoring resource. See NGSI-10 Open RESTful Api Specification for more details.

14.5.4 Architecture Components

The Policy Manager includes a data repository which keeps the rules stored and information about the server, tenants.

- **API-Agent (PMI)** is responsible of offering a RESTful interface to the Policy Manager GE users. It triggers the appropriate manager to handle the request.
  - **InfoManager**, is responsible for the management of general information about the server running and specific tenant information like the window size.
  - **RuleManager**, is responsible for the management of all related with general rules and rules for specified virtual machines.
- **Rules Engine**. Is responsible for handling when a condition is satisfied based on the facts received and launch the associated actions.

    - **RuleEngineManager**, provides management for access the rule engine based on CLIPS, adding the new facts to the Rule Engine and check rule conditions.
    - **DbManager**, provides connection to the Data Base.
- **Fact-Gen**, provides the mechanisms to insert facts into the rule Engine from context data received.
    - **FactGenManager**, is responsible for the management of all related with data context build facts from this data.
- **Condition & Actions Queue**, which contains all the rules and actions that can be managed by Policy Manager, including the window size for each tenant.
- **Facts Queue**, which represents the actual instantiation of resources for a specific resource. For each element in the inventory (called *-Instance), there is an equivalent in the catalogue. This queue is implemented with a list on a data structure server in order to obtain a rapid response of the system.

14.5.5 Example Scenario

The Policy Manager GE is involved in three different phases:

- Management of the rules provided by users.
- Populate rule engine with facts collected from the data context.
- Management of rules status at runtime.

14.5.5.1 *Rules Management*

The management of rules involves several operations to prepare the scalability system working. First of all, the rules have to be defined. The definition of a rule includes the specification of the actions to be launched, the conditions that must be inferred and a descriptive name so user can easily recognize the rule. This rule can also be specified for a single virtual machine.

Secondly, to get facts, it must subscribe the virtual machine to Context Broker GE in order to receive notifications of the resources status. Context Broker GE updates the context of each virtual machined to which we are subscribed and the Policy Manager stores this information in a Queue system in order to get a stable monitored value without temporal oscillation of the signal.

Finally, the rules can be deleted or redefined. When a rule is deleted, Policy Manager unsubscribe the virtual machine from Context Broker if rule is a Specific Rule.

14.5.5.2 *Collecting data*

The Context Broker has subscribed a number of virtual machines. Each virtual machine publishes the status of its resources in the Context Broker GE and Policy Manager receives this notifications. After that, Policy Manager is in charge of build facts and insert them into de Rule Engine. When we receive a number of Facts equal to the window size, the Policy Manager calculates the arithmetic mean of the data and insert its value into the Rule Engine. Finally, Policy Manager discards the oldest value in the queue.

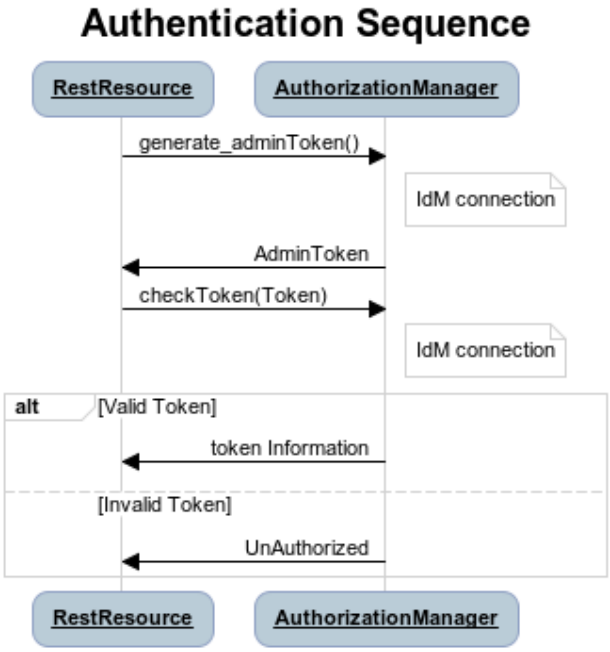
14.5.5.3 *Runtime Management*

During the runtime of an application, the Policy Manager can detect if a rule condition is inferred and is in charge of launch actions associated with, this action will be communicated to the users that was subscribed to this specific rule.

14.6 Main Interactions

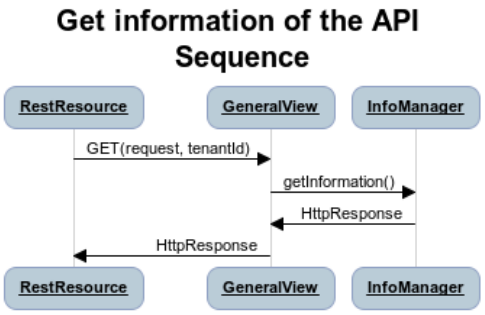
The following pictures depicts some interactions between the Policy Manager, the Cloud Portal as main user in a typical scenario. For more details about the Open REST API of this GE, please refer to the Open Spec API specification.

First of all, every interaction need Authentication sequence before starting. Authentication sequence follows like this:



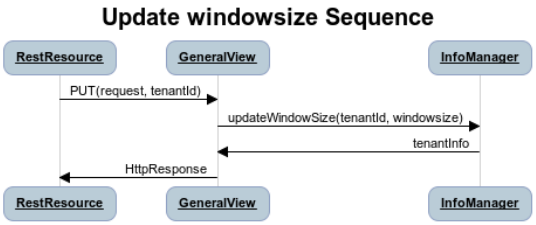
1. The Policy Manager requests a new administration Token from IdM in order to validate the future token received from the Cloud Portal through **generate_adminToken()** interface.
2. The IdM returns a valid administration token that will be used to check the *Token* received from the Cloud Portal requested message through the **checkToken(Token)** interface.
3. The IdM could return 2 options:
    1. If the *Token* is valid, the IdM returns the information related to this token.
    2. If the *Token* is invalid, the IdM returns the message of unauthorized token.

The next interactions gets information about the Policy Manager server:



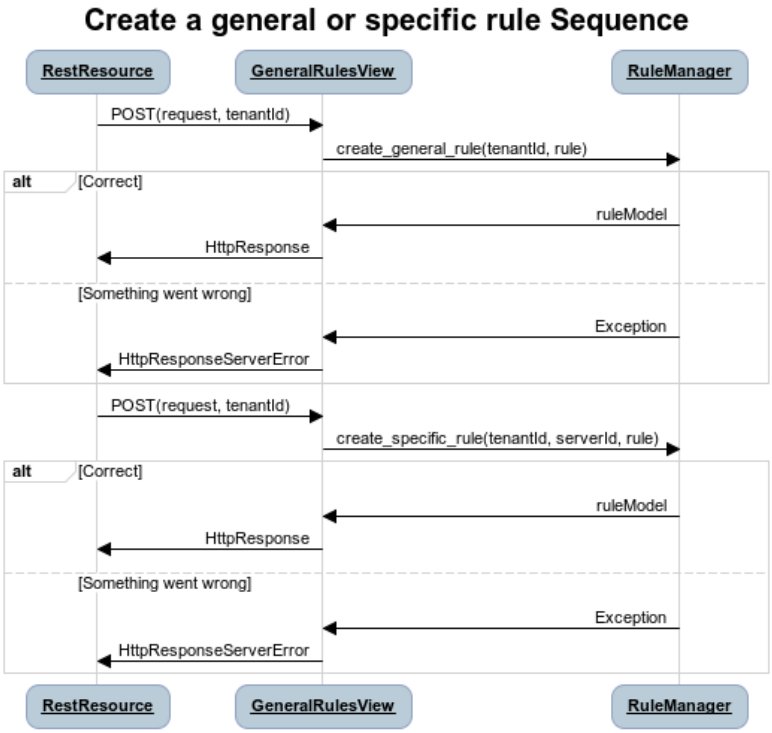
1. The User through Cloud Portal or CLI sends a GET operation to request information about the Policy Manager through **getInformation()**.
2. The InfoManager returns the information related to the Policy Manager GE associated to this tenant.
    1. Owner of the GEi.
    2. Time and date of the last wake up of the Policy Manager GE.
    3. URL of the open specification specification.
    4. Window size of the facts stabilization queue.

Following, you can see request to update the window size.



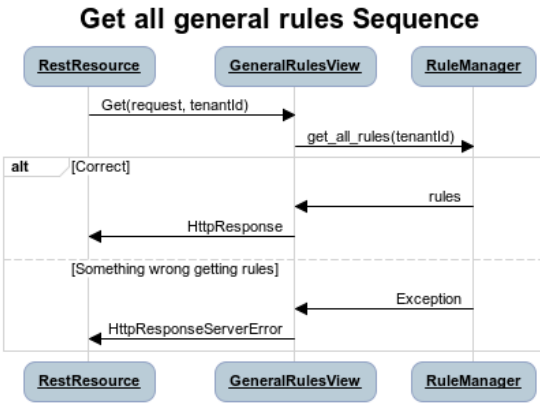
1. The User through Cloud Portal or CLI sends a PUT message to the Policy Manager GE to update the window size of the tenantId through the **updateWindowSize()** message.
2. The Policy Manager returns a message with the information associated to this tenantId in order to confirm that the change was made.

Next, you can see the interactions to create general or specific rule sequence



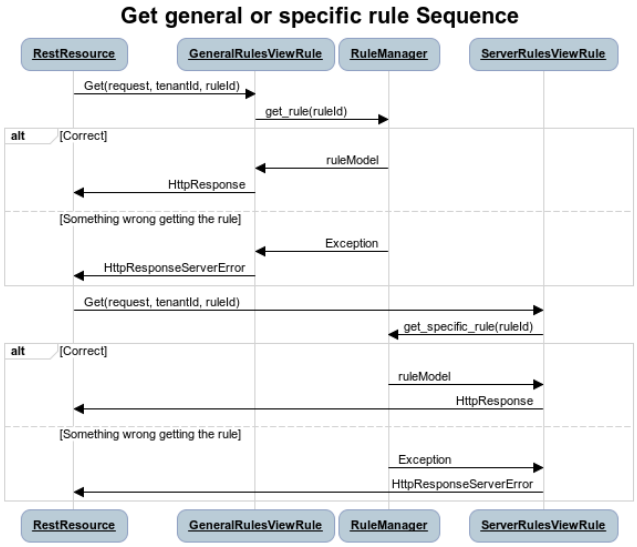
1. The User through Cloud Portal or CLI requests a POST operation to create a new general/specific rule to the Policy Manager.
    1. In case of general one, the **create_general_rule()** interface is used, with params *tenantId*, the OpenStack identification of the tenant, and the rule description.
    2. In case of specific one, the **create_specific_rule()** interface is used, with params *tenantId*, the OpenStack identification of the tenant, the *serverId*, the OpenStack identification of the server, and the rule description.
2. The Rule Manager returns the new ruleModel associated to the new requested rule and the Policy Manager returns the respense to the user.
    1. If something was wrong, due to incorrect representation of the rule, a *HttpResponseServerError* is returned in order to inform to the user that something was wrong.

Afterward, you could see the interactions to get information about already created general rules:



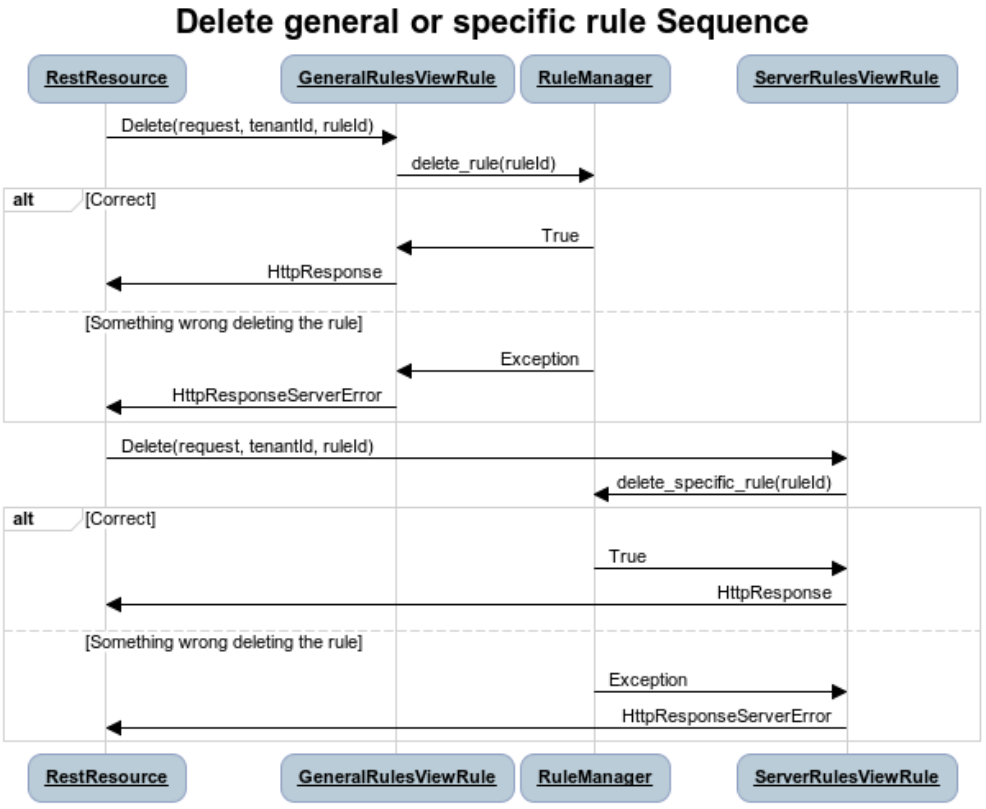
1. The User through Cloud Portal or CLI requests a GET operation to the Policy Manager in order to receive all the general rules associated to a tenant through **get_all_rules()** interface with parameter *tenantId*
2. The Rule Manager component of the Policy Manager responses with the list of general rules.
3. If the tenant identify is wrong or whatever the Rule Manager responses a HttpResponseServerError.

Following, the interactions to get detailed information about getting general or specific rule sequence.



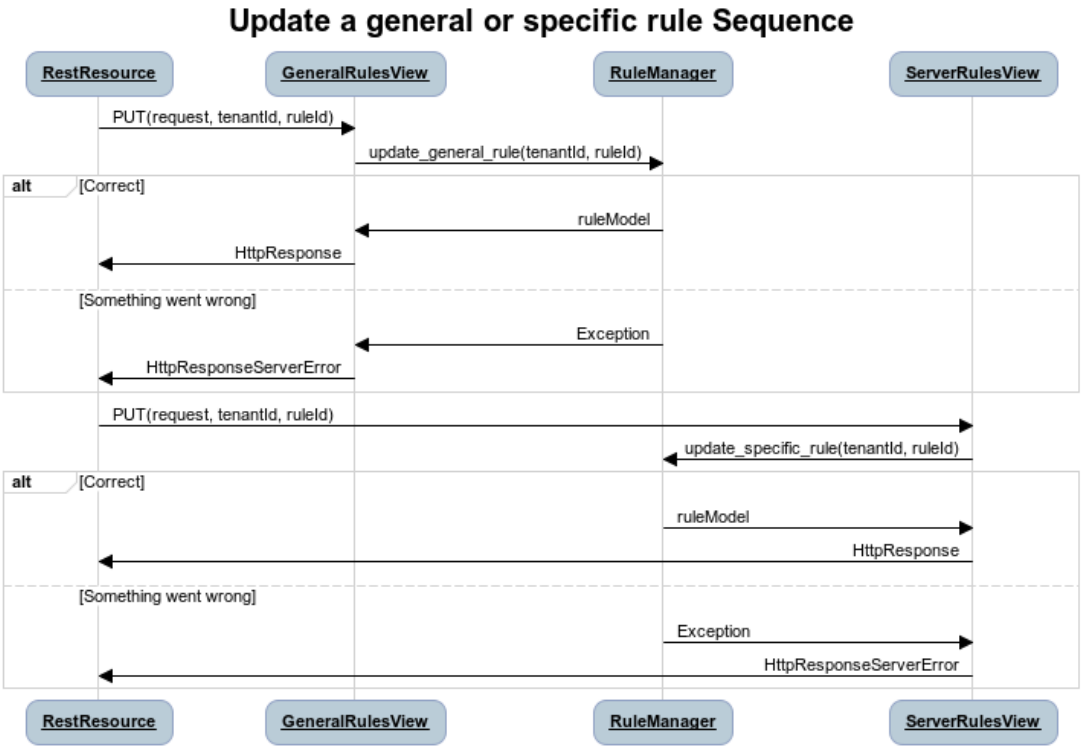
1. The User through Cloud Portal or CLI requests a GET operation to recover the rules.
    1. If we decide to recover a general rule, the **get_rule()** interface should be used with *ruleId* parameter
    2. Otherwise, if you decir to recover a specific rule, the **get_specific_rule()** interface should be used with the *ruleId* parameter.
2. The Rule Manager of the Policy Manager will return the ruleModel that it is stored in the Rule & Action Queue. If something was wrong, Policy Manager will return **HttpResponseServerError** to the user.

Next off, the interactions to delete general or specific rule.



1. The User through Cloud Portal or CLI requests the deletion of a general or specific rule to the Policy Manager with the identity of the tenant and rule.
    1. The view sends the request to the RuleManager by calling the **delete_rule()** interface with identity of the rule as parameter of this interface to delete it.
    2. Otherwise, if the rule is specific for a server, the views sends the request to the RuleManager by calling the **delete_specific_rule()** interface, with identity of the rule as parameter of this interface to delete it.
2. If the operation was ok, the RuleManager responses a *HttpResponse* with the ok message, by contrast, if something was wrong, it returns a *HttpResponseServerError* with the details of the problem.

Finally, the interactions to update a specific or general rule



1. The User through Cloud Portal or CLI requests the update of a general or specific rule to the Policy Manager with the identity of the tenant and rule.
    1. The view sends the request to the RuleManager by calling the **update_general_rule()** interface with identity of the tenant and rule as parameters of this interface to delete it.
    2. Otherwise, if the rule is specific for a server, the views sends the request to the RuleManager by calling the **update_specific_rule()** interface, with identity of the tenant and rule as parameters of this interface to delete it.
2. If the operation was ok, the RuleManager responses with a new ruleModel class created and the API returns a *HttpResponse* with the ok message, by contrast, if something was wrong, it returns a *HttpResponseServerError* with the details of the problem.

14.7 Basic Design Principles

14.7.1 Design Principles

The Policy Manager GE has to support the following technical requirements:

- The condition to fire the rule could be formulated on several facts.
- The condition to fire the rule could be formulated on several interrelated facts (the values of certain variables in those facts match).
- User could add facts "in runtime" via API (without stop server).
- User could add rules "in runtime" via API (without stop server).
- That part of the implementation of the rule would:

    - Update facts.
    - Delete facts.
    - Create new facts.
- Actions can use variables used in the condition.
- Actions implementation can invoke REST APIs.
- Actions can send an email.
- The Policy Manager should be integrated into the OpenStack without any problem.
- The Policy Manager should interact with the IdM GE in order to offer authentication functionality to this GE.
- The Policy Manager should interact with the Context Broker GE in order to receive monitoring information from resources.

14.7.2 Resolution of Technical Issues

When applied to Policy Manager GE, the general design principles outlined at Cloud Hosting Architecture can be translated into the following key design goals:

- Rapid Elasticity, capabilities can be quickly elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.
- Availability, Policy Manager should be running all the time without interruption of the service due to the nature of itself.
- Reliability, Policy Manager should assure that the activations of rule was produce by correct inference based on facts received from a Context Broker GE.
- Safety, is the Policy Manager has any problem, it should continue working without any catastrophic consequences on the user(s) and the environment.
- Integrity, Policy Manager does not allow the alteration of the facts queue and/or rules and actions queue.
- Confidentiality, Policy Manager does not allow the access to facts, rules and actions associated to a specitic tenant.

Regarding the general design principles not covered at Cloud Hosting Architecture, they can be translated into the following key design goals:

- REST based interfaces, for rules and facts.
- The Policy Manager GE keeps stored all rules provisioned for each user.
- The Policy Manager GE manage all facts and checks when actions should be fired.

14.8 Detailed Specifications

14.8.1 Open API Specifications

Following is a list of Open Specifications linked to this Generic Enabler. Specifications labeled as "PRELIMINARY" are considered stable but subject to minor changes derived from lessons learned during last interactions of the development of a first reference implementation planned for the current Major Release of FI-WARE. Specifications labeled as "DRAFT" are planned for future Major Releases of FI-WARE but they are provided for the sake of future users.

- Policy Manager Open RESTful API Specification

14.9 Re-utilised Technologies/Specifications

The Monitoring Manager GE is based on RESTful Design Principles. The technologies and specifications used in this GE are:

- RESTful web services
- HTTP/1.1 (RFC2616)
- JSON data serialization formats.

14.10 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be helpful to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FI-WARE Global Terms and Definitions

- **Runtime Execution Container** (**REC**) -- a VM with all the software stack required to deploy a complete node in an application architecture. It usually comprises one or more VMs, middleware, monitoring probes, and Chef client and SDC agent to support installation and configuration.
- **Software Deployment and Configuration GE** (**SDC GE**) -- a GE in devoted to the automated installation and configuration of software on VMs through the execution of recipes in the corresponding nodes. It relies on Opscode Chef technology.
- **Software Deployment and Configuration Interface** (**SDCI**) -- the interface offered by the SDC GE to be managed by the PaaS Manager or a Cloud Portal.
- **Product Instance** (**PI**) -- an installed software in a VMs, usually referring to middleware or platform software (E.g.: Apache Tomcat, MySQL, etc.).
- **Application Component** (**AC**) -- a component (or configuration artifact) that implements usually one or more components of an application architecture. These ACs are installed on, and differentiated from, existing PIs.
- **PaaS Manager Interface** (**PMI**) -- the interface offered by the PaaS Manager GE to be used by a Cloud Portal to manage both the catalogue and the lifecycle of the platform resources and applications.

15 Policy Manager Open RESTful API Specification

15.1 Introduction to the PMI Scalability Extension API

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

15.1.1 PMI Scalability Extension API

The PMI Scalability API is a RESTful, resource-oriented API accessed via HTTP/HTTPS that uses JSON-based representations for information interchange that provide functionalities to the Policy Manager GE. This document describes the FI-WARE-specific features extension, which allows cloud user to extend the basic functionalities offered by Policy Manager GE in order to cope with elasticity management.

15.1.2 Intended Audience

This specification is intended for both software developers and Cloud Providers. For the former, this document provides a full specification of how to interoperate with Cloud Platforms that implements PMI API. For the latter, this specification indicates the interface to be provided in order to create policies and actions associated the facts received from cloud resources (currently associated to servers but not only oriented to them). To use this information, the reader should first have a general understanding of the Policy Manager Generic Enabler and also be familiar with:

- RESTful web services
- HTTP/1.1 (RFC2616)
- JSON data serialization formats.

15.1.3 API Change History

This version of the PMI Scalability Extension API Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
Oct 17, 2012	• First version of the PMI Scalability Extension API.

15.1.4 How to Read This Document

In the whole document the assumption is taken that the reader is familiarized with REST architecture style. Along the document, some special notations are applied to differentiate some special words or concepts. The following list summarizes these special notations.

- A **bold**, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).
- An *italic* font is used to represent document titles or some other kind of special text, e.g., *URI*.
- The variables are represented between brackets, e.g. {id} and in italic font. When the reader find it, can change it by any value.

For a description of some terms used along this document, see [1].

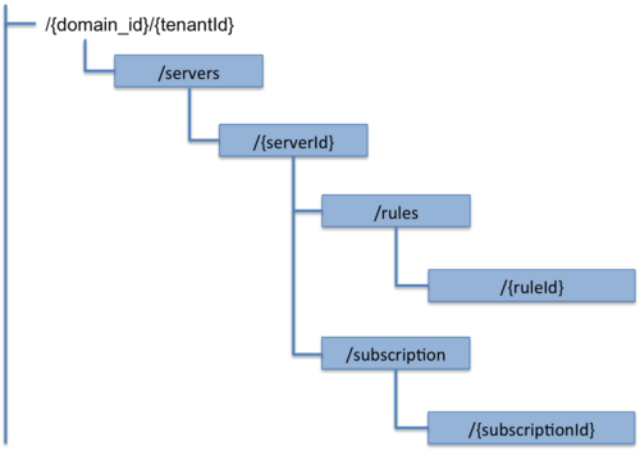
15.1.5 Additional Resources

You can download the most current version of this document from the FI-WARE API specification selecting **PDF Version** from the Toolbox menu (left side), which will generate the file to download it. For more details about the **Policy Manager** that this API is based upon, please refer to FI-WARE Cloud Hosting.

15.2 General PMI Scalability Extension API Information

15.2.1 Resources Summary

A graphical diagram, including the different Uniform Resource Names (URNs) that can be used in the API, is shown here. The URL is http://{serverRoot}:{serverPort}.



Policy Manager Open RESTful API resource summary

15.2.2 Authentication

Each HTTP request against the **PMI** requires the inclusion of specific authentication credentials. The specific implementation of this API supports OAuth v2.0 authentication schemes and will be determined by the specific provider that implements this GE and Interface. Please contact with it to determine the best way to authenticate against this API. Remember that some authentication schemes may require that the API operate using SSL over HTTP (HTTPS).

15.2.3 Representation Format

The PMI Scalability Extension API resources are represented by hypertext that allows each resource to reference other related resources. More concisely, JSON format are used for resource representation and URLs are used for referencing other resources by default. The request format is specified using the Content-Type header and is required for operations that have a request body. The response format can be specified in requests using either the Accept header with values application/json or adding a .json extension to the request URI. In the following examples we can see the different options in order to represent format.

```
POST /v1.0/d3fdddc6324c439780a6fd963a9fa148/servers/15520fa6dc914f97bd1e54f8e1444d41 HTTP/1.1

Host: servers.api.openstack.org

Content-Type: application/json

Accept: application/json

X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

```
POST /v1.0/d3fdddc6324c439780a6fd963a9fa148/servers/15520fa6dc914f97bd1e54f8e1444d41.json HTTP/1.1

Host: servers.api.openstack.org

Content-Type: application/json

X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

15.2.4 Representation Transport

Resource representation is transmitted between client and server by using HTTP 1.1 protocol, as defined by IETF RFC-2616. Each time an HTTP request contains payload, a Content-Type header shall be used to specify the MIME type of wrapped representation. In addition, both client and server may use as many HTTP headers as they consider necessary.

15.2.5 Resource Identification

API consumer must indicate the resource identifier while invoking a GET, PUT, POST or DELETE operation. PMI Scalability Extension API combines both identification and location by terms of URL. Each invocation provides the URL of the target resource along the verb and any required input data. That URL is used to

identify unambiguously the resource. For HTTP transport, this is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

PMI Scalability Extension API does not enforce any determined URL pattern to identify its resources. Anyway the SM Scalability Extension API follows the HATEOAS principle (Hypermedia As The Engine Of Application State). This means that resource representation contains the URLs of the related resources (e.g., book representation contains hyperlinks to its chapters; chapter representation contains hyperlinks to its pages...). API consumer obtains the server representation as its following point, which in turn provides hyperlinks that directly or indirectly take to other resources like scalability rules.

Some PMI Scalability Extension API entities provide an instance identifier property (instance ID). This property is used to identify unambiguously the entity but not the REST resource used to manage it, which is identified by its URL as described above. It is common that most implementations make use of instance ID to compose the URL (e.g., the book with instance ID 1492 could be represented by resource http://.../book/1492), but such an assumption should not be taken by API consumer to obtain the resource URL from its instance ID.

15.2.6 Links and References

Resources often lead to refer to other resources. In those cases, we have to provide an ID or an URL to a remote resource. see OpenStack Compute Developer Guide on their application to infrastructural resources.

15.2.7 Limits

n.a.

15.2.7.1 ***Rate Limits***

n.a.

15.2.7.2 ***Absolute Limits***

n.a.

15.2.7.3 ***Determining Limits Programmatically***

n.a.

15.2.8 Versions

This section shows the version of this API. You can see the historical change of the API at the beginning of this document. Currently, the version of this API is the 1.0.

15.2.9 Extensions

This document is a description itself of an extension, we have no possibilities to add extensions inside an extensions.

15.2.10 Faults

n.a.

15.3 API Operations

In this section we go in depth for each operation. These operations were described in the Policy Manager Architectural description. The FI-WARE programmer guide will also provide examples of how to use this API. The specify operations of this extensions are related to the management of scalability rules.

15.3.1 General Operations

This section has the general operations related to this service.

15.3.1.1 ***Get the information of the API***

Verb	URI	Description
GET	/{tenantId}/	Get information about this current API.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

This operation does not require a request body and lists the information of the current version of the API. The following examples show a JSON response for the API operation:

Response:

```
{
    "owner": "TELEFONICA I+D",
    "windowsize": <windows_size>,
    "version": "<API_version>",
```

```
    "runningfrom": "<last_launch_date>
    "doc": "<URL_DOCUMENTATION>"
}
```

The descriptions of the returned values are the following:

- **owner** is the key whose value is the company name that develops this API. Its value is fixed to "Telefonica I+D".
- **windowsize** is the key that represents the window size (<windows_size>) to stabilize the values of the measures probes to checking rules and taking actions. This value is very important due to allow resolving false positives that could launch the action to scaling up and down a server.
- **version** is the key whose value is the version (<API_version>) of the API currently in execution.
- **runningfrom** is the key whose value is the date of the last launch (<last_launch_date>) of the service. This value takes the ISO 8601 an example of this value 2013-10-04 20:32:17.
- **doc** is the key whose value is the link to this API specification.

15.3.1.2 *Update the window size*

Verb	URI	Description
PUT	/{tenantId}/	Update the window size of the service.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

This call updates the window size of the service in order to change the stabilization window size to be applied to the monitoring data received from the Monitoring GE. The request is in JSON format and the response has no body.

Request:

```
{
    "windowsize": <windows_size>
}
```

Where **windowsize** is the key whose value is the size of the windows to stabilized the values of the measures probes to checking rules and taking actions. This value is very important due to allow resolving false values that could launch the action to scaling up and down a server.

Response:

```
{
    "windowsize": <windows_size>
}
```

15.3.2 Servers

This section has the operations related to the subscription to the platform together with the rules associated to the servers to be analyzed by the rules engine.

15.3.2.1 *Get the list of all servers' rules*

Verb	URI	Description
GET	/{tenantId}/servers	Get the list of all servers registered in the platform.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

Returns a list of servers with their rules. There is no body in the request and the response is the following one:

Response:

```
{
    "servers": [
        {
            "serverId": "<serverId>",
            "rules": [
                {
                    "condition": <CONDITION_DESCRIPTION>,
                    "action": <ACTION_ON_SERVER>,
                    "ruleId": "<RULE_ID>"
                },
                {
                    "condition": <CONDITION_DESCRIPTION>,
                    "action": <ACTION_ON_SERVER>,
                    "ruleId": "<RULE_ID>"
                }
            ]
        },
        {
            "serverId": "<serverId>",
            "rules": [
                {
                    "condition": <CONDITION_DESCRIPTION>,
```

```
                    "action": <ACTION_ON_SERVER>,
                    "ruleId": "<RULE_ID>"
                },
                {
                    "condition": <CONDITION_DESCRIPTION>,
                    "action": <ACTION_ON_SERVER>,
                    "ruleId": "<RULE_ID>"
                }
            ]
        }
    ]
}
```

The values that you receive are the following:

- **serverId** is the key whose value specifies the server ID in the URI, following the OpenStack ID format. An example of it is the id 52415800-8b69-11e0-9b19-734f6af67565.
- **condition** is the key whose value is the description of the scalability rule associated to this server. It could be one or more than one and the format of this rule is the following:
- **action** is the key whose value represents the action to take over the server. Its values are up and down.
- **ruleId** is the key that represents the id of the rule, following the OpenStack Id format (e.g. 52415800-8b69-11e0-9b19-734f6f006e54).

15.3.2.2 *Get the list of all rules of a server*

Verb	URI	Description
GET	/*{tenantId}*/servers/*{serverId}*	Get all rules related to specified server.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

This operation returns the list of elasticity rules associated with a server identified with its *{serverId}*. This operation does not require a body and the response is in JSON format.

Response:

```
{
    "serverId": "<serverId>",
    "rules": [
                {
```

```
                        "name": <NAME>,
                        "condition": <CONDITION_DESCRIPTION>,
                        "action": <ACTION_ON_SERVER>,
                        "ruleId": "<RULE_ID>"
                    },
                    {
                        "name": <NAME>,
                        "condition": <CONDITION_DESCRIPTION>,
                        "action": <ACTION_ON_SERVER>,
                        "ruleId": "<RULE_ID>"
                    }
        ]
}
```

The values that you receive are the following:

- **serverId** is the key whose value specifies the server ID in the URI, following the OpenStack ID format. An example of it is the id 52415800-8b69-11e0-9b19-734f6af67565.
- **condition** is the key whose value is the description of the scalability rule associated to this server. It could be one or more than one and the format of this rule is the following:
- **action** is the key whose value represents the action to take over the server. Its values are up and down.
- **ruleId** is the key that represents the id of the rule, following the OpenStack Id format (e.g. 52415800-8b69-11e0-9b19-734f6f006e54).

15.3.2.3 *Update the context of a server*

Verb	URI	Description
POST	/*{tenantId}*/servers/*{serverId}*	Update Context of a specific server.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

This operation updates the context related to a specific server, identified with its *serverId*. The context information contains the description of the CPU, Memory, Disk and/or Network usages. This message follows the NGSI-10 information model but using JSON format and the response has no body.

Request:

```
{
        "subscriptionId": "<SubscriptionId>",
        "originator": "http://localhost/test",
```

```
        "contextResponses": [
                {
                        "contextElement": {
                                "type": "Server",
                                "isPattern": "false",
                                "id": "<ServerId>",
                                "attributes": [
                                        {
                                                "name": "CPU",
                                                "type": "Probe",
                                                "value": "0.75",
                                        },
                                        {
                                                "name": "Memory",
                                                "type": "Probe",
                                                "value": "0.83",
                                        },
                                        {
                                                "name": "Disk",
                                                "type": "Probe",
                                                "value": "0.83",
                                        },
                                        {
                                                "name": "Network",
                                                "type": "Probe",
                                                "value": "0.83",
                                        }
                                ],
                        },
                        "statusCode": {
                                "code": "200",
                                "reasonPhrase": "Ok",
                                "details": "a message"
                        }
                }
        ]
}
```

The values that you receive are the following:

- **SubscriptionId**, is the identifier of a subscription process following the id schemas of OpenStack.
- **type**, is the element type, in our case, it is always "Server".
- **isPattern**, is used to define some type of pattern in order to search the information in the list of attributes. In our case, this attribute is not used and is always fixed to "false".
- **id**, is the id of a server, the same id of ServerId of OpenStack.
- **attributes**, this is a list of attributes:
  - **type** is the type of attribute, for our case, this key has always the value "Probe".
  - **value**, is the value of the attribute expressed in percentage.
  - **name** is the name of the attribute. In our case, this key takes one of the following values:
    - **CPU**, amount of used CPU of a server.
    - **Memory**, amount of used Memory of the same server.
    - **Disk**, amount of used disk (HDD) of the same server.
    - **Network**, amount of used network interface of the same server.
- **statusCode**, in NGSI-10 this key shows the information that the system should return when it receives this message. Currently, our implementation does not take into consideration this information but have to be defined following the standard. Its values are always the same in that case how you can see in the previous example.

15.3.3 Elasticity rules

15.3.3.1 *Create a new elasticity rule*

Verb	URI	Description
POST	/{tenantId}/servers/{serverId}/rules	Create a new rule associated to the server.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

This operation creates a new elasticity rules associated to a server, which is identified by {serverId}. The request specifies the rule to be activated and the action associated to it (increase or decrease the number of servers). The response returns a 200 Ok message together with the id of the new rule created.

Request:

```
{
   "name": <NAME>,
   "condition": <CONDITION_DESCRIPTION>,
   "action": <ACTION_ON_SERVER>
}
```

The values that you receive are the following:

- **name** is the key whose value represents the name of the rule.
- **condition** is the key whose value is the description of the scalability rule associated to this server. It could be one or more than one and the format of this rule is the following:
- **action** is the key whose value represents the action to take over the server. Its values are up and down.

Response:

```
{
    "serverId": <serverId>,
    "ruleId": <RULE_ID>
}
```

The values that you receive are the following:

- **serverId** is the key whose value specifies the server ID in the URI, following the OpenStack ID format. An example of it is the id 52415800-8b69-11e0-9b19-734f6af67565.
- **ruleId** is the key that represents the id of the rule, following the OpenStack Id format (e.g. 52415800-8b69-11e0-9b19-734f6f006e54).

15.3.3.2 *Update an elasticity rule*

Verb	URI	Description
PUT	/{tenantId}/servers/{serverId}/rules/{ruleId}	Update an elasticity rule.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

This operation allows to update the rule condition, the action or both or a specific server identified by its {serverId} and a specific rule identified by its {ruleId}. This operation requires a request context and the response has no body on it.

Request:

```
{
    "name": <NAME>,
    "condition": <CONDITION_DESCRIPTION>,
    "action": <ACTION_ON_SERVER>
}
```

Where:

- **name** is the key whose value represents the name of the rule.

- **condition** is the key whose value is the description of the scalability rule associated to this server. It could be one or more than one and the format of this rule is the following:
- **action** is the key whose value represents the action to take over the server. Its values are up and down.

Response:

```
{
    "name": <NAME>,
    "condition": <CONDITION_DESCRIPTION>,
    "action": <ACTION_ON_SERVER>
 }
```

15.3.3.3 *Delete an elasticity rule*

Verb	URI	Description
DELETE	/{tenantId}/servers/{serverId}/rules/{ruleId}	Delete an elasticity rule.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

This operation deletes a specific rule, identified by its {ruleId}, within a server, identified by its {serverId}. This operation does not require a request body and response body. The response is a 200 Ok if it was deleted without any problem or error message in other case.

15.3.3.4 *Get an elasticity rule*

Verb	URI	Description
GET	/{tenantId}/servers/{serverId}/rules/{ruleId}	Get an elasticity rule.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

This operation gets a specific rule, identified by its {ruleId}, within a server, identified by its {serverId}. This operation does not require a request body and response body is in JSON format.

Response:

```
 {
    "name": <NAME>,
```

```
    "condition": <CONDITION_DESCRIPTION>,
    "action": <ACTION_ON_SERVER>,
    "ruleId": "<RULE_ID>"
}
```

Where:

- **name** is the key whose value represents the name of the rule.
- **condition** is the key whose value is the description of the scalability rule associated to this server. It could be one or more than one and the format of this rule is the following:
- **action** is the key whose value represents the action to take over the server. Its values are up and down.
- **ruleId** is the key that represents the id of the rule, following the OpenStack Id format (e.g. 52415800-8b69-11e0-9b19-734f6f006e54).

15.3.4 Subscription to rules

15.3.4.1 *Create a new subscription*

Verb	URI	Description
POST	/{tenantId}/servers/{serverId}/subscription/	Create a new subscription for the server.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

This operation creates a new subcription rules associated to a rule, which is identified by {ruleId}. The request specifies the rule to be activated and the action associated to it (increase or decrease the number of servers). The response returns a 200 Ok message together with the id of the new subscription created.

Request:

```
{
    "ruleId": <RULE_ID>,
    "url": <URL_TO_NOTIFY>,
}
```

The values that you receive are the following:

- **ruleId** is the key whose value identifies the rule associated to this server.
- **url** is the key whose value is the url to notify the action when the rule is fired.

Response:

```
{
    "subscriptionId": <SUBSCRIPTION_ID>
}
```

The values that you receive are the following:

- **subscriptionId** is the key that represents the id of the subscription, following the OpenStack Id format (e.g. 52415800-8b69-11e0-9b19-734f6f006e54).

15.3.4.2 *Delete a subscription*

Verb	URI	Description
DELETE	/{tenantId}/servers/{serverId}/subscription/{subscriptionId}	Delete a subscription.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

This operation deletes a subscription, identified by its {subscriptionId}, within a server, identified by its {serverId}. This operation does not require a request body and response body. The response is a 200 Ok if it was deleted without any problem or error message in other case.

15.3.4.3 *Get a subscription*

Verb	URI	Description
GET	/{tenantId}/servers/{serverId}/subscription/subscriptionId}	Get a subscription.

Normal Response Code(s): 200 (Ok)

Error Response Code(s): identityFault (400, 500, …), badRequest (400), unauthorized (401), forbidden (403), badMethod (405), serviceUnavailable (503)

This operation gets a subscription, identified by its {subscriptionId}, within a server, identified by its {serverId}. This operation does not require a request body and response body is in JSON format.

Response:

```
{
   "subscriptionId": <SUBSCRIPTION_ID>,
   "url": <URL_TO_NOTIFY>,
   "serverId": <SERVER_ID>,
   "ruleId": "<RULE_ID>"
}
```

Where:

- **subscriptionId** is the key that represents the id of the subscription, following the OpenStack Id format (e.g. 52415800-8b69-11e0-9b19-734f6f006e54).
- **url** is the key whose value is the url to notify the action when the rule is fired.

- **serverId** is the key whose value specifies the server ID in the URI, following the OpenStack ID format. An example of it is the id 52415800-8b69-11e0-9b19-734f6af67565.
- **ruleId** is the key that represents the id of the rule, following the OpenStack Id format (e.g. 52415800-8b69-11e0-9b19-734f6f006e54).

15.4 Elasticity Rules

In this section we explain how it is represented an elasticity rule.

15.4.1 Rules Engine

Rules are described using JSON, and contain information about CPU and Memory usage, in first instance.

15.4.2 Example Rule

The rule is compound of three parts, name, conditions and actions. In this case, the name will be "AlertCPU"

Every fact is like "(server (server-id 12345-abcd)(cpu 50)(mem 33))"

In this case, the condition defined expects all server with cpu usage more than 98.3

Actions will create an HTTP POST notification to an url specified on every subscription to this rule. In this case the notification will be that server should be scaled up because CPU usage is greater than limit.

This is the rule as is expected to:

```
{
    "action": {
          "actionName": "notify-scale",
          "operation": "scaleUp"
    },
    "name": "AlertCPU",
    "condition": {
           "cpu": {
                  "value": 98.3,
                  "operand": "greater"
           },
           "mem": {
                  "value": 95,
                  "operand": "greater equal"
           }
   }
}
```

16 FIWARE OpenSpecification Cloud JobScheduler

16.1 Preface

Within this document you find a self-contained open specification of a FI-WARE generic enabler, please consult as well the FI-WARE Product Vision, the website on http://www.fi-ware.org and similar pages in order to understand the complete context of the FI-WARE project.

16.2 Copyright

Copyright © 2013 by INRIA. All Rights Reserved.

16.3 Legal Notice

Please check the following FI-WARE Open Specifications Legal Notice to understand the rights to use these specifications.

16.4 Overview

This specification describes the Job Scheduler GE, which is the key enabler to execute a generic job over distributed multiple heterogeneous computer systems, both physical and virtual ones.

The Job Scheduler GE integrates to major internal services, namely the Resource Manager (RM) Service and the Scheduler Service. Thanks to the internal RM Service, the Job Scheduler GE abstracts computer systems as computing resources where job can be executed, called nodes, offering the following main features:

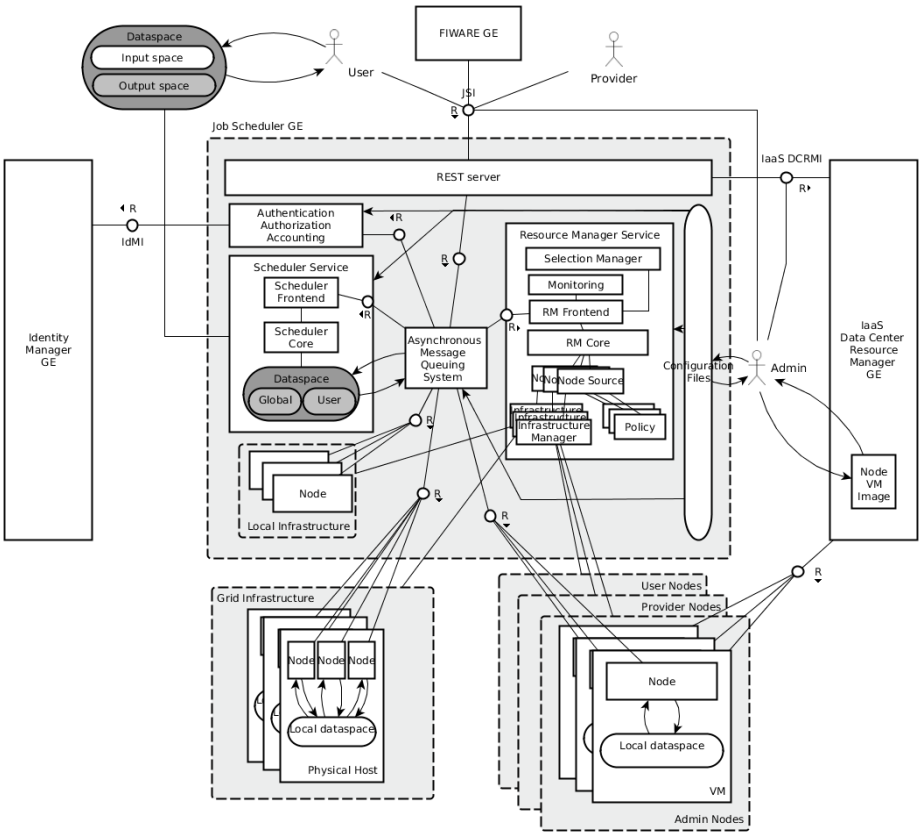
- infrastructures management
- nodes provisioning based on users criteria
- nodes life-cycle management
- monitoring

Thanks to the internal Scheduler Service, the Job Scheduler GE finally puts those resources at the disposal of applications, users and other FI-WARE GEs, by giving the possibility of

- submitting a job for execution
- handling its life-cycle
- specifying the data locations.

So, the Job Scheduler GE can act as a general purpose GE that helps to save valuable time when a high amount of computation is required for data processing. It also increases the average computing resources usage, when underutilized, by offering the possibility to add dynamically underutilized resources through a registration mechanism. That, thanks to the embedded AAA (Authentication, Authorization, Accounting) system, is of particular interest for those who would like to play the nodes *provider* role, other than the *user* and *administrator* one.

The following diagram shows the main components of the Job Scheduler Generic Enabler.



Job Scheduler GE architecture specification

In the above diagram, the REST Server implements the API front-end to the Job Scheduler GE and offers a representation of both the internal Scheduler Service and the Resource Manager Service to the Cloud User. Those two services are the backbone of the Job Scheduler GE back-end.

Because the computing resources being handled are distributed, an Asynchronous Message Queuing System is needed to allow the communication between all components to take place in a non-blocking way. Such a system relies on a middleware layer, whose role is to fill the gap that exists between all the heterogeneous computing systems, where nodes handled by the Job Scheduler GE can be available.

16.4.1 Target Usage

The Job Scheduler GE enriches the FI-WARE Cloud Architecture with an intuitive and powerful enabler, based on simple abstractions, by introducing the concepts of job and computing node.

Thanks to the middleware layer, it finds naturally place in the Cloud Hosting Chapter, since cloud computing itself is just one of the possible computing systems.

The Job Scheduler GE offers the way of gathering, aggregating and monitoring resources for computing purposes, by playing a key role for attracting users/enterprises that may already have physical resources -other than virtual ones-

at their disposal. So, the Job Scheduler GE actually enables them to achieve a hybrid (physical and virtual) approach concerning computing resources usage, by increasing its average value. In fact, that might be the case when a physical computing resource, such as a workstation, is assigned to a just one developer and, at the same time, be configured to host tasks computations, as well.

Finally, the Job Scheduler GE is a general purpose GE to process data, available to users, enterprises and all the other FI-WARE GEs, especially to those having low computing performance at their disposal, as it happens to the Cloud Edge GE.

16.5 Main concepts

Following the above FMC diagram of the Job Scheduler, in this section we introduce the main concepts related to this GE through the definition of their interfaces and components; finally, an example of their use.

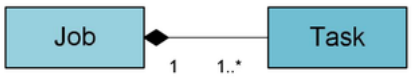
The Job Scheduler GE allows the job submission and its life-cycle control, by taking into account resources that are free, meaning available from Resource Manager perspective. Leveraging on computing resources abstraction achieved by these entities called nodes, it handles the dynamic addition/subtraction of resources, which could be desktop computers, clusters or clouds.

The key components visible to the cloud user could be differentiated between the interfaces and the components, together with the explication of the concepts used on it, each of them is described below.

16.5.1 Entities

In order to use the Job Scheduler GE, users should be familiar with the following key concepts:

- **Job.** A Job is the entity to be submitted to the Scheduler, according with a given priority. It is composed of one or more tasks. More concretely, a job can be described through an XML which reflects the Job XSD schema.
- **Task.** A Task is the smallest schedulable entity. It is included in a Job and will be executed in accordance with the scheduling policy on the available resources. It might be a *Java* task, which implements a specific interface, or a *native* task that is any user program, a compiled C/C++ application, a shell or batch script. Moreover, since a task might require more Computing Nodes at the same time, a *multi-node* task could be considered, as well.

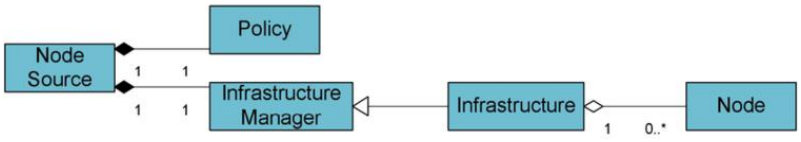


Scheduler concepts class diagram

- **Computing Node.** A Computing Node, briefly Node, is a logical container for computing tasks. More concretely, it might be thought as a software agent (such a JVM), running on the computing resource, able to leverage its operating system to compute tasks and to extend it with any customized library, in order to be part of the middleware.
- **Nodes Deployment.** Nodes Deployment, briefly Deployment, is the process by which at least one computing resource is enabled to host one or

more Nodes. It depends highly on the middleware nature, but a simple example might be achieved by implementing *ssh* commands.

- **Infrastructure.** An Infrastructure is an aggregation of Nodes representing a target environment to host the execution of specific tasks. An example is the *Local Infrastructure* in the architecture picture, which represents the aggregation of some nodes, locally deployed together with the Job Scheduler GE implementation. Being local, the nodes deployment does not require *ssh* commands.
- **Infrastructure Manager.** It is in charge of main Infrastructure management. Its behaviour could be extended by the Infrastructure.
- **Policy.** A Policy defines a strategy of the Deployment. An example of Policy might be "deploy the Nodes at the moment of the Infrastructure creation and never remove them" (static deployment policy) or "deploy the Nodes at a particular time" (time slot policy).
- **Node Source.** A Node Source is composed by an Infrastructure Manager and a Policy. All Nodes under the same Node Source will be launched on the same Infrastructure, according with the given Policy.



Resource Manager concepts class diagram

- **Nodes Selection Script.** A Nodes Selection Script, briefly Selection Script, is the tool through which the user performs Nodes selection mechanism, according to some available criteria of his interest. The most common scripting languages (Javascript, Python, Ruby and so on) could be used. A Javascript concrete example is available here.
- **Node Registration.** Node Registration, briefly Registration, is the process by which a Node is registered to a Node Source. After that, the Node belongs officially to the computing resources pool visible at Resource Manager level.

16.5.2 Interfaces

The Job Scheduler GE is currently composed of one main interface:

- The **Job Scheduler Interface** (JSI) is the REST API that, at the same time, provides RESTful representation of both internal Scheduler and Resource Manager Services, which together constitute its backbone. It utilizes JavaScript Object Notation (JSON) to serialize state objects and transports them over HTTP. Usage of HTTP, as the transport protocol, eliminates most of the restrictions imposed by corporate firewalls; while JSON provides a lightweight, widely used mechanism to serialize/deserialize state objects.

16.5.3 Components

By referring to the Job Scheduler GE Architecture, here follows the description of the five main modules characterizing the Job Scheduler GE, the most of them subjected to the admin configuration:

- **REST Server**, that relieves the Scheduler and the RM Service from the eventual overload due to the connection of large groups of simultaneous clients. In fact, thanks to a built-in caching mechanism, clients operations involving the state (task, job or the Scheduler and so on) are served according to the local (cached) state objects, periodically updated. Hence, REST Server effectively reduces the communication load due to multiple/concurrent requests. For the back-end integration, this component uses the native middleware to communicate with both the Scheduler Service and the Resource Manager Services.
- **Asynchronous Message Queuing System**, that assures non-blocking messages delivery across heterogeneous computing systems. The way that could be accomplished depends on the embedded middleware architecture, which could leverage on a distributed or centralized communication system between its components. In the last case, it acts basically like a software router.
- **Authentication, Authorization, Accounting System**, that is the first component contacted when the user wants to log in. It is in charge of authenticating the user and allowing him to access (or not) to the Scheduler Service or Resource Manager Service features. Moreover, it regulates the accounting aspects related to the nodes usage.
- **Resource Manager Service**, that is the software for coupling distributed resources in order to solve large-scale problems. It provides a single point of access to all resources by enabling an effective way of selecting and aggregating them for computations with required criteria. In order to accomplish all the above, Resource Manager relies on the following embedded sub-components:
  - **RM Front-end**, that offers the interface for accessing all the other sub-components.
  - **RM Core**, that keeps an up-to-date list of nodes able to perform the Scheduler tasks; gives nodes to the Scheduler asked by its user; dialogs with Node Sources for adding/removing nodes; performs creation and removal of Node Source; treats nodes addition/removal request; creates and launches events concerning nodes and node sources to Monitoring
  - **Monitoring**, that provides a way for a monitor to ask the Resource Manager to throw events, generated by nodes and nodes sources management.
  - **Selection Manager**, that is responsible for nodes selection from a pool of free nodes for further scripts execution. User requests of getting nodes are processed by Selection Manager, which may contact nodes at the request time and execute some code there in order to know whether the node is suitable. Once the user has obtained nodes, he contacts them directly without involving the RM.
  - **Node Source**, that manages acquisition, monitoring and creation/removal of a set of nodes in the Resource Manager.
  - **Infrastructure Manager**, that is the part of Node Source responsible for node deployment/release to/from the actual underlying infrastructure. For instance, it may launch a node over ssh or by submitting a specific job to the native scheduler of the system.
  - **Node Source Policy**, that is the part of node source defining rules and limitations of node source utilization. All policies require to define an administrator of the node source and a set of its users, so that you can limit nodes utilization. Moreover, the policy defines rules of nodes deployment, like static deployment (all nodes are launched at the

moment of node source creation and never removed) or time slot deployment (nodes are deployed for particular time) or others.

- **Scheduler Service**, that is the main entity and is not a GUI daemon, which -acting as client- is connected to the Resources Manager Service. It is composed of the following sub-components:
  - **Scheduler Front-end**, that is responsible for the management of the Scheduler. All authenticated users requests are treated by this front-end. Before transmitting requests to the core, the front-end checks if the users have the required authorization. This interface allows users to submit jobs, get scheduling state, and retrieve job results.
  - **Scheduler Core**, that is the main entity of the Scheduler Service. The Core is responsible for the Scheduler implementation and communicates with the RM Service to acquire nodes. It is in charge of scheduling Jobs according with the policy (FIFO by default), retrieving scheduling events to the user and making storage. Users cannot interact directly with the Scheduler Core, but they need to pass through the Scheduler Service Front-end.
  - **Dataspaces**, that allow user to handle files during the scheduling process. In fact, as part of the Scheduler Service infrastructure, they define from where the nodes, in order to accomplish a given task, can pick files up and where they can put the produced files. Each dataspace might have its own scope and, mainly, we propose the following classification:
    - the *GLOBALSPACE* is a virtual place, under the Scheduler service control domain, shared among all the users, where anyone has the read and write permissions;
    - the *USERSPACE* is a virtual place, under the Scheduler service control domain, whose access and manipulation is limited to the user in question only;
    - the *INPUTSPACE* and *OUTPUTSPACE* are virtual, additional places where users can put/pull/delete data keeping them under their own control domain, which might be remote with respect to the Scheduler host location. Those add flexibility to fit the needs of the most exigent users, who cannot/do not want move their data from their premises.

16.5.4 Example Scenario

This section provides three main use cases from user, enterprise and FI-WARE GEs perspective, in order to show:

- how the Job Scheduler GE could cover in FI-WARE the needs of a short time or temporary business model, whereas cloud computing uses to address users towards a long term one.
- how the Job Scheduler GE could push to get into FI-WARE Cloud ecosystem, by allowing hybrid resources aggregation for jobs scheduling.
- how the Job Scheduler GE could play in FI-WARE the role of a general purpose GE.

16.5.4.1 *User Use Case*

1. A worker needs to have heavy computation and many tries are required to get the best result, to be achieved as soon as possible. By doing that with just his own laptop or with his current VM in the Cloud, it will require hours of processing for each try and, so, he loses valuable time.
2. From a colleague, that had similar constraints, he finds the solution by designing an ad-hoc job to submit to the Job Scheduler GE, available in the cloud.
3. After job design phase, he fills his credentials, waiting for Identity Management GE validation.
4. Once approved, he submits the job which will create a dedicated virtual nodes infrastructure by leveraging his own VM image, built starting from a template, at the IaaS Data Center Resource Manager GE. In few minutes, he fetches the first results. If not satisfied, he tries again by tuning those parameters that define the job.
5. Finally, the worker is so happy that shares his experience on his blog

16.5.4.2 *Enterprise Use Case*

1. An enterprise has some business in the field of computing intensive data rendering and is thinking to get into the cloud. The only drawback with such an approach is that all its physical machines might be wasted, since nothing acting like a "glue" between distributed resources seems to be available.
2. So, it receives the nice news about the existence of the Job Scheduler GE, which enables the usage of distributed heterogeneous computing resources for data processing.
3. Once deployed into the cloud, the administrator of the enterprise proceeds as follows:
    1. first, he discovers which types of infrastructures and policies are available;
    2. then, creates new node sources, according with his infrastructure and policy requirements;
    3. finally, runs nodes on each available physical machine and registers them under the same RM Service, running at a given endpoint in the cloud;
    4. eventually, checks if nodes are actually available from the RM Service perspective and proceeds by assigning them the rendering tasks.

16.5.4.3 *FI-WARE GEs Use Case*

Both the previous scenarios could be easily rethought in such perspective. In particular, the Cloud Edge GE could have some computing issue due to not being a high performance device. As well as, any Edglet could leverage the Job Scheduler GE computing resources, for example to manipulate audio and video files.

16.6 Main Interactions

The Job Scheduler GE provides intuitive operations to manage the Resource Manager Service, the Scheduler Service and the other main entities, previously described.

16.6.1 Scheduler Service

- ***createCredential*** - creates user credentials as function of his username, password and the public ssh key of the Job Scheduler GEi server
- ***login*** - enables user to access the Scheduler with his credentials
- ***loginWithCredential*** - enables the user to login to the scheduler by submitting his credential file
- ***disconnect*** - disconnects user from the Scheduler
- ***isConnected*** - tests whether or not the user is connected to the Scheduler.
- ***startScheduler*** - starts the Scheduler
- ***pauseScheduler*** - pauses the Scheduler
- ***freezeScheduler*** - freezes the Scheduler
- ***resumeScheduler*** - resumes the Scheduler
- ***stopScheduler*** - stops the Scheduler
- ***killScheduler*** - kills the Scheduler
- ***getSchedulerStatus*** - returns the current Scheduler status.
- ***getSchedulerStats*** - returns statistics about the Scheduler
- ***getMySchedulerStats*** - returns statistics about the Scheduler usage of the current user
- ***getConnectedUsers*** - returns users currently connected to the Scheduler
- ***getJobs*** - returns jobs list
- ***linkRM*** - connects the Scheduler to a given the RM endpoint
- ***getSchedulerVersion*** - returns the current REST Server API and the Scheduler version.

16.6.1.1 *Jobs*

- ***submitJob*** - submits a job to the Scheduler
- ***killJob*** - kills the job execution
- ***deleteJob*** - deletes the job information
- ***getLiveLogs*** - returns only the currently available logs of a job
- ***removeLiveLogs*** - disables live logs relate to a job
- ***getServerLogs*** - returns job server logs
- ***pauseJob*** - pauses the job execution
- ***resumeJob*** - resumes the job execution
- ***getJobState*** - returns the job state
- ***getJobsInfo*** - returns a subset of the Scheduler state
- ***changeJobPriority*** - changes the priority of a job under execution
- ***getJobResult*** - returns the job result and related logs
- ***getTasks*** - returns the list of all the tasks belonging to the job
- ***getTasksState*** - returns the list of the state of all tasks related to the job

16.6.1.2 *Tasks*

These operations are used to manage tasks within a job:

- ***killTask*** - kills a task within a job
- ***restartTask*** - restarts the task

- ***preemptTask*** - preempts a task within a job
- ***getTaskResult*** - returns the task result
- ***getTaskState*** - gets task state

16.6.1.3 *Dataspaces*

These operations are used to manipulate data in the dataspaces:

- ***pushData*** - pushes a file from the local file system into the given dataspace.
- ***pullData*** - either pulls a file from the given dataspace to the local file system or lists the content of a directory.
- ***deleteData*** - deletes a file or recursively deletes a directory from the given dataspace

16.6.2 Resource Manager Service

- ***login*** - enables user to access the RM with his credentials
- ***loginWithCredential*** - enables the user to login to the RM by submitting his credential file
- ***disconnect*** - disconnects user from the RM and release all the nodes taken by user for computations
- ***isActive*** - tests if the RM is operational.
- ***shutdownRM*** - kills the RM
- ***getRMInfo*** - retrieves specific information relate to the RM
- ***getMonitoring*** - gets the initial state of the RM
- ***getRMStatHistory*** - returns the RM statistic history
- ***getRMState*** - returns an overview about current free/alive/total nodes number of the RM
- ***getRMVersion*** - returns the current RM version

16.6.2.1 *Node Source*

- ***createNodeSource*** - creates a new node source in the RM, specifying infrastructure and policy, with related parameters
- ***removeNodeSource*** - removes a new node source from the RM
- ***getSupportedInfrastructures*** - returns the list of supported node source infrastructures descriptors
- ***getSupportedPolicies*** - returns the list of supported node source policies descriptors

16.6.2.2 *Nodes*

- ***isNodeAvailable*** - tests if a node is registered to the RM
- ***lockNode*** - prevents other users from using a set of locked nodes
- ***unlockNode*** - allows other users to use a set of nodes previously locked
- ***releaseNode*** - releases a node, previously reserved for computation
- ***addNode*** - adds a node to a particular node source. If not specified, add it to the default node source of the RM
- ***removeNode*** - removes a node from the RM

16.7 Basic Design Principles

In order to address a technical audience interested in the design of the Job Scheduler GE, the following basic principles should be taken into account:

- Job Scheduler supports a RESTful interface;
- Job Scheduler should be addressed to gather the most heterogeneous computing resource as possible;
- Job Scheduler should be deployed on a virtual machine/physical host with a public IP address, in order to gather resources across internet;
- Job Scheduler components communication must be asynchronous;
- Job Scheduler relies on a middleware layer, able of behaving in an adaptive way with respect to network layer and devices (especially firewalls);
- Job Scheduler's RM Service, being a single point of access to resources, should be highly stable.
- Job Scheduler should have nodes available, in order to launch jobs, so that the RM Service should be launched before Scheduler Service;
- Job Scheduler enables several users to share the same pool of resources and also to manage issues related to distributed environment, such as failing resources;
- Job Scheduler should offer the possibility to aggregate nodes in different type of infrastructures, each one having a policy, and abstract each pair infrastructure-policy in a node source;
- Job Scheduler must offer a dynamic node addition/subtraction to a node source;
- Job Scheduler should maintain and monitor the list of resources;
- Job Scheduler must supply computing nodes to users based on user criteria (i.e. specific operating system, available resources or licenses);
- Job Scheduler is in charge of scheduling submitted jobs in accordance with the scheduling policy;
- Job Scheduler should have the global dataspace as default for input and output data, defined by the administrator;
- Job Scheduler should give the possibility to the user to specify all the possible dataspaces location when a job is defined;
- Cloud Users could either deploy it or could register nodes to an already existing the Job Scheduler;
- Cloud Users could deploy nodes on any physical host, laptop, workstation or virtual machine.

16.8 Detailed Specifications

Following is a list of Open Specifications linked to this Generic Enabler. Specifications labeled as "PRELIMINARY" are considered stable but subject to minor changes derived from lessons learned during last interactions of the development of a first reference implementation planned for the current Major Release of FI-WARE.

Specifications labeled as "DRAFT" are planned for future Major Releases of FI-WARE but they are provided for the sake of future users.

16.8.1 Open API Specifications

- Job Scheduler Open RESTful API Specification (PRELIMINARY)

16.9 Re-utilised Technologies/Specifications

The Job Scheduler GE is based on RESTful Design Principles. The technologies and specifications used in this GE are:

- REST web services architecture
- HTTP/1.1 (RFC2616)
- JSON.
- ProActive Scheduling & Resourcing RESTful API

16.10 Terms and definitions

This section comprises a summary of terms and definitions introduced during the previous sections. It intends to establish a vocabulary that will be helpful to carry out discussions internally and with third parties (e.g., Use Case projects in the EU FP7 Future Internet PPP). For a summary of terms and definitions managed at overall FI-WARE level, please refer to FI-WARE Global Terms and Definitions

- **Runtime Execution Container** (**REC**) -- a VM with all the software stack required to deploy a complete node in an application architecture. It usually comprises one or more VMs, middleware, monitoring probes, and Chef client and SDC agent to support installation and configuration.
- **Software Deployment and Configuration GE** (**SDC GE**) -- a GE in devoted to the automated installation and configuration of software on VMs through the execution of recipes in the corresponding nodes. It relies on Opscode Chef technology.
- **Software Deployment and Configuration Interface** (**SDCI**) -- the interface offered by the SDC GE to be managed by the PaaS Manager or a Cloud Portal.
- **Product Instance** (**PI**) -- an installed software in a VMs, usually referring to middleware or platform software (E.g.: Apache Tomcat, MySQL, etc.).
- **Application Component** (**AC**) -- a component (or configuration artifact) that implements usually one or more components of an application architecture. These ACs are installed on, and differentiated from, existing PIs.
- **PaaS Manager Interface** (**PMI**) -- the interface offered by the PaaS Manager GE to be used by a Cloud Portal to manage both the catalogue and the lifecycle of the platform resources and applications.

17 Job Scheduler Open RESTful API Specification

17.1 Introduction to the JSI API

Please check the following FI-WARE Open Specifications Legal Notice to understand the rights to use these specifications.

17.1.1 JSI API

The JSI API is a RESTful, resource-oriented API accessed via HTTP/HTTPS that uses JSON-based representation for information exchange. This API allows Cloud User to access basic functionalities of the internal Scheduler Service and Resource Manager Service, described at Job Scheduler GE Architecture Specifications.

17.1.2 Intended Audience

This specification is intended for both software developers and reimplementers of this API. For the former, this document provides a full specification of how to interoperate with Cloud Platforms that implements JSI API (derived from ProActive Scheduling & Resourcing RESTful API).

For the latter, this specification indicates the interface to be provided to clients to interoperate with the Job Scheduler GE within the Cloud Platform to provide the described functionalities. To use this information, the reader should first have a general understanding of the Job Scheduler Generic Enabler and also be familiar with:

- REST web services architecture
- HTTP/1.1 (RFC2616)
- JSON.

17.1.3 API Change History

This version of the JSI API Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Changes Summary
Apr 22, 2013	• 2.3 version of the JSI API
Jan 31, 2014	• 3.2 version of the JSI API. Added the following resources: ○ createCredential ○ loginWithCredential ○ removeLiveLogs ○ deleteJob ○ pushData ○ pullData

	○ deleteData

17.1.4 How to Read This Document

In the whole document the assumption is taken that the reader is familiarized with REST architecture style. Along the document, some special notations are applied to differentiate some special words or concepts. The following list summarizes these special notations.

- A **bold**, mono-spaced font is used to represent code or logical entities, e.g., HTTP method (GET, PUT, POST, DELETE).
- An *italic* font is used to represent document titles or some other kind of special text, e.g., *URI*.
- The variables are represented between brackets, e.g. {id} and in italic font. When the reader find it, can change it by any value.

For a description of some terms used along this document, see FIWARE.ArchitectureDescription.Cloud.JobScheduler.

17.1.5 Additional Resources

- FI-WARE Cloud Hosting, where the whole Cloud Hosting Architecture is available
- For any information may seem missed here, we invite you to dig into ProActive Parallel Suite documentation, we offered while introducing Job Scheduler GE Baseline Assets

17.2 General JSI API Information

17.2.1 Resources Summary

An incremental view, where Uniform Resource Names (URNs) of each resource are shown starting from the root resource, is browsable at ProActive Scheduling & Resourcing REST API (v1.3.2) web page.

In order to improve readability, the same resources may be rearranged in an exhaustive lexico-graphical order, by achieving ProActive Scheduling & Resourcing REST API (v1.3.2) Index.

Finally, in order to know which is the path that represents the prefix for the root of the resources defined by the Job Scheduler GE Open RESTful Specification, the most appropriate resource is Job Scheduler - User and Programmers Guide. If such information was not present over there, by default we let you assume the following URL: *http://<hostname>:<port>/rest/rest*

17.2.2 Authentication

Each HTTP request against the **JSI** requires the inclusion of specific authentication credentials. As required at Inter-dependencies and Interaction Between GEs, the specific implementation of this API may support multiple authentication schemes (OAuth, Basic Auth, Token) according to Identity Management GE Architecture Specifications. Those will be determined by the specific provider that implements this GE and Interface. Please contact with it to determine the best way to authenticate against this API. Remember that some authentication schemes may require that the API operate using SSL over HTTP (HTTPS).

17.2.3 Representation Format

The JSI API resources are represented by hypertext that allows each resource to reference other related resources. More concisely, JSON and XML format are used for resource representation and URLs are used for referencing other resources by default. The request format is specified using the Content-Type header and is required for operations that have a request body. The response format can be specified in requests using either the Accept header with values *application/json* or *.json* extension to the request URI. In the following examples we can see the different options in order to represent format.

17.2.4 Representation Transport

Resource representation is transmitted between client and server by using HTTP 1.1 protocol, as defined by IETF RFC-2616. Each time an HTTP request contains payload, a Content-Type header shall be used to specify the MIME type of wrapped representation. In addition, both client and server may use as many HTTP headers as they consider necessary.

17.2.5 Resource Identification

API consumer must indicate the resource identifier while invoking a GET, PUT, POST or DELETE operation. JSI API combines both identification and location by terms of URL. Each invocation provides the URL of the target resource along the verb and any required input data. That URL is used to identify unambiguously the resource. For HTTP transport, this is made using the mechanisms described by HTTP protocol specification as defined by IETF RFC-2616.

Some JSI API entities provide an instance identifier property (instance ID). This property is used to identify unambiguously the entity but not the REST resource used to manage it, which is identified by its URL as described above. It is common that most implementations make use of instance ID to compose the URL (e.g., the book with instance ID 1492 could be represented by resource http://.../book/1492), but such an assumption should not be taken by API consumer to obtain the resource URL from its instance ID.

17.2.6 Links and References

Resources often lead to refer to other resources. In those cases, we have to provide an ID or an URL to a remote resource, as it will happen for identifying jobs and computing nodes resources.

17.2.7 Versions

In order to get the actual compatibility between the REST API and the back-end services of a Job Scheduler GE implementation, it is important to check their versions. For such a purpose, we invite you to refer to **getRMVersion** and **getSchedulerVersion** operations, at API Operations section.

17.2.8 Extensions

17.2.9 Faults

By default, the set of faults, which a **JSI** service can return, are derived from Built-in Internally-Thrown Exceptions of RESTEasy framework and generated when errors in dispatching or marshalling are encountered during client requests processing. Here follow the most important ones:

Exception	HTTP Code	Description
BadRequestException	400	Bad Request. Request wasn't formatted correctly or problem processing request input.
UnauthorizedException	401	Unauthorized. Security exception thrown if you're using Resteasy's simple annotation-based role-based security
InternalServerErrorException	500	Internal Server Error.
MethodNotAllowedException	405	Method Not Allowed. There is no method for the resource that can handle the invoked HTTP operation.
NotAcceptableException	406	Not Acceptable. There is no method that can produce the media types listed in the Accept header.
NotFoundException	404	Not Found. There is no method that serves the request path/resource.

Anyway, in order to offer a customized and reasonable exceptions handling, useful for debugging and troubleshooting, previous behavior may be overrided by implementing the following exceptions mapping:

Exception	Mapped Error
ConnectionException	HTTP_NOT_FOUND
InternalSchedulerException	HTTP_INTERNAL_ERROR
IOException	HTTP_NOT_FOUND

JobAlreadyFinishedException	HTTP_NOT_FOUND
JobCreationException	HTTP_NOT_FOUND
JobCreationException	HTTP_NOT_FOUND
KeyException	HTTP_NOT_FOUND
LoginException	HTTP_NOT_FOUND
NotConnectedException	HTTP_UNAUTHORIZED
PermissionException	HTTP_FORBIDDEN
MessageQueuingRuntimeException (alias ProActiveRuntimeException)	HTTP_NOT_FOUND
RuntimeException	HTTP_INTERNAL_ERROR
SchedulerException	HTTP_NOT_FOUND
SubmissionClosedException	HTTP_NOT_FOUND
UnknownJobException	HTTP_NOT_FOUND
UnknownTaskException	HTTP_NOT_FOUND
MappingException (alias ThrowableMapperException)	HTTP_SERVER_ERROR

17.3 API Operations

While at Job Scheduler Architectural Specification we had an introduction to main interactions doable via **JSI**, in this section we go in depth for each operation and introduce additional ones. Please, note that operations may not be described here, since under discussion and, therefore, may be included in future releases.
In order to actually get started to interact with the Job Scheduler GE API, we invite you to refer to the Job Scheduler - User and Programmers Guide, where you can find per-resource examples. There, you may notice that the integration with Identity Management GE will reasonably consist in storing in the *sessionid* parameter, required for each request against ProActive Scheduling & Resourcing REST API, the content of the *id* of the *token* returned after the successful user authentication, by providing *username* and *password* parameters. Please, refer to **login** on the following subsections.

Finally, we would like to stress how, in most cases, a boolean value might be back: in case of success, the value will be *true*; *false* otherwise.

17.3.1 Scheduler Service

17.3.1.1 ***Common***

17.3.1.1.1 *createCredential*

verb	URI	description
POST	/scheduler/createcredential	create user credentials as function of his username, password and the public ssh key of the Job Scheduler GEi server

HTTP Example:

```
POST /scheduler/createcredential
```

API Example:

```
SchedulerStateRest.getCreateCredential({'$entity': /*
generates a credential file from user provided credentials
*/});
```

Input:Login form
Output: byte[] - the credential file generated by the scheduler
Produces: */*
Consumes: multipart/form-data

17.3.1.1.2 *login*

verb	URI	description
POST	/scheduler/login	login to the scheduler using an form containing 2 fields (username & password)

HTTP Example:

```
POST /scheduler/login username=…&password=…
```

API Example:

```
SchedulerStateRest.login({'username': /* username username
*/,
'password': /* password password */});
```

Form parameters:

- ***username*** - username
- ***password*** - password

Output: String - the session id associated to the login
Produces: application/json
Consumes: application/x-www-form-urlencoded

17.3.1.1.3 *login with credential*

verb	URI	description
POST	/scheduler/login	login to the scheduler using a multipart form can be used

		either by submitting - 2 fields username & password - a credential file with field name 'credential'

HTTP Example:

```
POST /scheduler/login
```

API Example:

```
SchedulerStateRest.loginWithCredential({'$entity': /* login
to the scheduler using a multipart form can be used either by
submitting - 2 fields username & password - a credential file
with field name 'credential' */});
```

Input:Login form
Output: String - the session id associated to the login
Produces: application/json
Consumes: multipart/form-data

17.3.1.1.4 *disconnect*

verb	URI	description
PUT	/scheduler/disconnect	terminates the session id sessionId

HTTP Example:

```
PUT /scheduler/disconnect sessionid: …
```

API Example:

```
SchedulerStateRest.disconnect({'sessionid': /* sessionId a
valid session id */});
```

Output: void
Header parameters:
- ***sessionid*** - a valid session id

Produces: */*

17.3.1.1.5 *isConnected*

verb	URI	description
GET	/scheduler/isconnected	Tests whether or not the user is connected to the ProActive Scheduler

HTTP Example:

```
GET /scheduler/isconnected sessionid: …
```

API Example:

```
SchedulerStateRest.isConnected({'sessionid': /* sessionId the
session to test */});
```

Output: *boolean* - true if the user connected to a Scheduler, false otherwise.
Header parameters:
- ***sessionid*** - the session to test

Produces: application/json

17.3.1.1.6 *startScheduler*

verb	URI	description
PUT	/scheduler/start	starts the scheduler

HTTP Example:

```
PUT /scheduler/start sessionid: …
```

API Example:

```
SchedulerStateRest.startScheduler({'sessionid': /* sessionId
a valid session id */});
```

Output: *boolean* - true if success, false otherwise.
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.1.7 *pauseScheduler*

verb	URI	description
PUT	/scheduler/pause	pauses the scheduler

HTTP Example:

```
PUT /scheduler/pause sessionid: …
```

API Example:

```
SchedulerStateRest.pauseScheduler({'sessionid': /* sessionId
a valid session id */});
```

Output: *boolean* - true if success, false otherwise.
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.1.8 *freezeScheduler*

verb	URI	description
PUT	/scheduler/freeze	freezes the scheduler

HTTP Example:

```
PUT /scheduler/freeze sessionid: …
```

API Example:

```
SchedulerStateRest.freezeScheduler({'sessionid': /* sessionId
a valid session id */});
```

Output: *boolean* - true if success, false otherwise.

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.1.9 *resumeScheduler*

verb	URI	description
PUT	/scheduler/resume	resumes the scheduler

HTTP Example:

```
PUT /scheduler/resume sessionid: …
```

API Example:

```
SchedulerStateRest.resumeScheduler({'sessionid': /* sessionId
a valid session id */});
```

Output: *boolean* - true if success, false otherwise.

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.1.10 *stopScheduler*

verb	URI	description
PUT	/scheduler/stop	stops the scheduler

HTTP Example:

```
PUT /scheduler/stop sessionid: …
```

API Example:

```
SchedulerStateRest.stopScheduler({'sessionid': /* sessionId a
valid session id */});
```

Output: *boolean* - true if success, false otherwise.

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.1.11 *killScheduler*

verb	URI	description
PUT	/scheduler/kill	kills and shutdowns the scheduler

HTTP Example:

```
PUT /scheduler/kill sessionid: …
```

API Example:

```
SchedulerStateRest.killScheduler({'sessionid': /* sessionId a valid session id */});
```

Output: *boolean* - true if success, false otherwise.
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.1.12 *getSchedulerStatus*

verb	URI	description
GET	/scheduler/status	returns the status of the scheduler

HTTP Example:

```
GET /scheduler/status sessionid: …
```

API Example:

```
SchedulerStateRest.getSchedulerStatus({'sessionid': /* sessionId a valid session id */});
```

Output: *SchedulerStatusData* - the scheduler status
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.1.13 *getSchedulerStats*

verb	URI	description
GET	/scheduler/stats	returns statistics about the scheduler

HTTP Example:

```
GET /scheduler/stats sessionid: …
```

API Example:

```
SchedulerStateRest.getStatistics({'sessionid': /* sessionId the session id associated to this new connection */});
```

Output: *Map<String,String>* - a string containing the statistics
Header parameters:

- ***sessionid*** - the session id associated to this new connection

Produces: application/json

17.3.1.1.14 *getMySchedulerStats*

verb	URI	description
GET	/scheduler/stats/myaccount	returns a string containing some data regarding the user's account

HTTP Example:

```
GET /scheduler/stats/myaccount sessionid: …
```

API Example:

```
SchedulerStateRest.getStatisticsOnMyAccount({'sessionid': /*
sessionId the session id associated to this new connection
*/});
```

Output: *Map<String,String>* -a string containing some data regarding the user's account

Header parameters:

- ***sessionid*** - the session id associated to this new connection

Produces: application/json

17.3.1.1.15 *getConnectedUsers*

verb	URI	description
GET	/scheduler/users	Users currently connected to the scheduler

HTTP Example:

```
GET /scheduler/users sessionid: …
```

API Example:

```
SchedulerStateRest.getUsers({'sessionid': /* sessionId the
session id associated to this new connection\ */});
```

Output: *List<SchedulerUserData>* -list of users

Header parameters:

- ***sessionid*** - the session id associated to this new connection

Produces: application/json

17.3.1.1.16 *getJobs*

verb	URI	description
GET	/scheduler/jobs?index=…&range=…	Returns the ids of the current jobs under a list of string.

HTTP Example:

```
GET /scheduler/jobs?index=…&range=… sessionid: …
```

API Example:

```
SchedulerStateRest.jobs({'index': /* index optional, if a
sublist has to be returned the index of the sublist */,

'range': /* range optional, if a sublist has to be returned,
the range of the sublist */,

'sessionid': /* sessionId a valid session id */});
```

Output: *List<String>* - a list of jobs' ids under the form of a list of string

Query parameters:

- ***index*** - optional, if a sublist has to be returned the index of the sublist
- ***range*** - optional, if a sublist has to be returned, the range of the sublist

Header parameters:

- ***sessionid*** - the session id associated to this new connection

Produces: application/json

17.3.1.1.17 *linkRM*

verb	URI	description
POST	/scheduler/linkrm	Reconnect a new Resource Manager to the scheduler.

HTTP Example:

```
POST /scheduler/linkrm sessionid: … rmurl=…
```

API Example:

```
SchedulerStateRest.linkRm({'sessionid': /* sessionId a valid
session id */,
'rmurl': /* rmURL the url of the resource manager */});
```

Output: *boolean* - true if success, false otherwise.

Form parameters:

- ***rmurl*** - the url of the resource manager

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.1.18 *getSchedulerVersion*

verb	URI	description
GET	/scheduler/version	returns the version of the rest api

HTTP Example:

```
GET /scheduler/version
```

API Example:

```
SchedulerStateRest.getVersion({});
```

Output: *String* - returns the version of the rest api

17.3.1.2 ***Jobs***

17.3.1.2.1 *submitJob*

verb	URI	description
POST	/scheduler/submit	Submits a job to the scheduler

HTTP Example:

```
POST /scheduler/submit sessionid: …
```

API Example:

```
 SchedulerStateRest.submit({'sessionid': /* sessionId a valid
session id */,
 '$entity': /* Submits a job to the scheduler */});
```

Input: *MultipartFormDataInput*
Output: JobIdData - the jobid of the newly created job
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json
Consumes: multipart/form-data

17.3.1.2.2 *killJob*

verb	URI	description
PUT	/scheduler/jobs/{jobid}/kill	Kill the job represented by jobId.

HTTP Example:

```
 PUT /scheduler/jobs/{jobid}/kill sessionid: …
```

API Example:

```
 SchedulerStateRest.killJob({'jobid': /* jobId the job to
kill. */,
  'sessionid': /* sessionId a valid session id */});
```

Output: *boolean* - true if success, false otherwise.
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.2.3 *deleteJob*

verb	URI	description
DELETE	/scheduler/jobs/{jobid}	Delete a job

HTTP Example:

```
 DELETE /scheduler/jobs/{jobid} sessionid: …
```

API Example:

```
 SchedulerStateRest.removeJob({'jobid': /* jobId the id of the
job to delete */,
  'sessionid': /* sessionId a valid session id */});
```

Output: *boolean* - true if success, false if the job not yet finished (not removed, kill the job then remove it)
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.2.4 *deleteJob*

verb	URI	description
GET	/scheduler/jobs/{jobid}/livelog	return only the currently available logs of job identified by the id *jobid*

HTTP Example:

```
GET /scheduler/jobs/{jobid}/livelog sessionid: …
```

API Example:

```
SchedulerStateRest.getLiveLogJob({'jobid': /* jobId the id of
the job to retrieve */,
  'sessionid': /* sessionId a valid session id */});
```

Output: *String*
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.2.5 *removeLiveLogs*

verb	URI	description
DELETE	/scheduler/jobs/{jobid}/livelog	return true if the live logs of job identified by the id *jobid* have been disabled

HTTP Example:

```
DELETE /scheduler/jobs/{jobid}/livelog sessionid: …
```

API Example:

```
SchedulerStateRest.deleteLiveLogJob({'jobid': /* jobId the id
of the job to retrieve */,
  'sessionid': /* sessionId a valid session id */});
```

Output: *boolean*
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.2.6 *getServerLogs*

verb	URI	description
GET	/scheduler/jobs/{jobid}/log/server	return job server logs by means of *jobid*

HTTP Example:

```
GET /scheduler/jobs/{jobid}/log/server sessionid: …
```

API Example:

```
SchedulerStateRest.jobServerLog({'jobid': /* jobId the id of
the job */,
```

```
  'sessionid': /* sessionId a valid session id */});
```

Output: *String* -job traces from the scheduler and resource manager
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.2.7 *pauseJob*

verb	URI	description
PUT	/scheduler/jobs/{jobid}/pause	pause the job execution by means of *jobid*

HTTP Example:

```
PUT /scheduler/jobs/{jobid}/pause sessionid: …
```

API Example:

```
SchedulerStateRest.pauseJob({'jobid': /* jobId the id of the
job */,
  'sessionid': /* sessionId a valid session id */})
```

Output: *boolean* - true if success, false if not
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.2.8 *getJobState*

verb	URI	description
GET	/scheduler/jobs/{jobid}	return the job state by providing *jobid*.

HTTP Example:

```
GET /scheduler/jobs/{jobid} sessionid: …
```

API Example:

```
SchedulerStateRest.listJobs({'jobid': /* jobId the id of the
job to retrieve */,
  'sessionid': /* sessionId a valid session id */});
```

Output: JobStateData
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json, application/xml

17.3.1.2.9 *getJobsInfo*

verb	URI	description
GET	/scheduler/jobsinfo?index=…&range=…	Returns a subset of the scheduler state, including pending, running,

		finished jobs (in this particular order).

HTTP Example:

```
GET /scheduler/jobsinfo?index=…&range=… sessionid: …
```

API Example:

```
SchedulerStateRest.jobsinfo({'index': /* index optional, if a sublist has to be returned the index of the sublist */,
'range': /* range optional, if a sublist has to be returned, the range of the sublist */,
'sessionid': /* sessionId a valid session id */});
```

Output: *List<UserJobData>* - a list of UserJobData
Query parameters:

- ***index*** - optional, if a sublist has to be returned, the index of the sublist
- ***range*** - optional, if a sublist has to be returned, the range of the sublist

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json, application/xml

17.3.1.2.10 *changeJobPriorityByName*

verb	URI	description
PUT	/scheduler/jobs/{jobid}/priority/byname/{name}	change the previous job priority by providing *jobid* and priority *name*

HTTP Example:

```
PUT /scheduler/jobs/{jobid}/priority/byname/{name} sessionid: …
```

API Example:

```
SchedulerStateRest.schedulerChangeJobPriorityByName({'jobid': /* jobId the job id */,
'name': /* priorityName a string representing the name of the priority */,
'sessionid': /* sessionId a valid session id */});
```

Output: *void*
Header parameters:

- ***sessionid*** - a valid session id

17.3.1.2.11 *changeJobPriorityByName*

verb	URI	description
PUT	/scheduler/jobs/{jobid}/priority/byname/{name}	/scheduler/jobs/{jobid}/priority/byvalue/{value}

HTTP Example:

```
 PUT /scheduler/jobs/{jobid}/priority/byvalue/{value}
sessionid: …
```

API Example:

```
SchedulerStateRest.schedulerChangeJobPriorityByValue({'jobid':
/* jobId the job id */,
 'value': /* priorityValue a string representing the value of
the priority */,
 'sessionid': /* sessionId a valid session id */});
```

Output: *void*
Header parameters:

- ***sessionid*** - a valid session id

17.3.1.2.12 *getJobResult*

verb	URI	description
GET	/scheduler/jobs/{jobid}/result	return the job result associated to *jobid*

HTTP Example:

```
 GET /scheduler/jobs/{jobid}/result sessionid: …
```

API Example:

```
 SchedulerStateRest.jobResult({'jobid': /* Returns the job
result associated to the job referenced by the id jobid */,
 'sessionid': /* sessionId a valid session id */});
```

Output: JobResultData - the job result of the corresponding job
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.2.13 *getTasks*

verb	URI	description
GET	/scheduler/jobs/{jobid}/tasks	a list of the name of the tasks belonging to job, by providing *jobid*.

HTTP Example:

```
 GET /scheduler/jobs/{jobid}/tasks sessionid: …
```

API Example:

```
 SchedulerStateRest.getJobTasksIds({'jobid': /* jobId jobid
one wants to list the tasks' name */,
 'sessionid': /* sessionId a valid session id */});
```

Output: *List<String>* - a list of tasks' name

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.2.14 *getTasksState*

verb	URI	description
GET	/scheduler/jobs/{jobid}/taskstates	return the list of all the tasks state related to the job, by providing *jobid*.

HTTP Example:

```
GET /scheduler/jobs/{jobid}/taskstates sessionid: …
```

API Example:

```
SchedulerStateRest.getJobTaskStates({'jobid': /* jobId the
job id */,
'sessionid': /* sessionId a valid session id */});
```

Output: *List<TaskStateData>* - a list of task's states of the job jobId

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.3 ***Tasks***

17.3.1.3.1 *killTask*

verb	URI	description
PUT	/scheduler/jobs/{jobid}/tasks/{taskname}/kill	kill a task within a job.

HTTP Example:

```
PUT /scheduler/jobs/{jobid}/tasks/{taskname}/kill sessionid:
…
```

API Example:

```
SchedulerStateRest.killTask({'jobid': /* jobid id of the job
containing the task to kill */,
'taskname': /* taskname name of the task to kill */,
'sessionid': /* sessionId current session */});
```

Output: *boolean*

Header parameters:

- ***sessionid*** - current session

17.3.1.3.2 *restartTask*

verb	URI	description

PUT	/scheduler/jobs/{jobid}/tasks/{taskname}/restart	restart the task.

HTTP Example:

```
 PUT /scheduler/jobs/{jobid}/tasks/{taskname}/restart
sessionid: …
```

API Example:

```
 SchedulerStateRest.restartTask({'jobid': /* jobid id of the
job containing the task to kill */,
 'taskname': /* taskname name of the task to kill */,
 'sessionid': /* sessionId current session */});
```

Output: *boolean*
Header parameters:

- ***sessionid*** - current session

17.3.1.3.3 *preemptTask*

verb	URI	description
PUT	/scheduler/jobs/{jobid}/tasks/{taskname}/preempt	preempt a task within a job.

HTTP Example:

```
 PUT /scheduler/jobs/{jobid}/tasks/{taskname}/preempt
sessionid: …
```

API Example:

```
 SchedulerStateRest.preemptTask({'jobid': /* jobid id of the
job containing the task to preempt */,
 'taskname': /* taskname name of the task to preempt */,
 'sessionid': /* sessionId current session */});
```

Output: *boolean*
Header parameters:

- ***sessionid*** - current session

17.3.1.3.4 *getTaskResult*

verb	URI	description
GET	/scheduler/jobs/{jobid}/tasks/{taskname}/result	return the task result and related logs.

HTTP Example:

```
 GET /scheduler/jobs/{jobid}/tasks/{taskname}/result
sessionid: …
```

API Example:

```
 SchedulerStateRest.taskresult({'jobid': /* jobId the id of
the job */,
 'taskname': /* taskname the name of the task */,
 'sessionid': /* sessionId a valid session id */});
```

Output: TaskResultData - the task result of the task taskName
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.3.5 *getTaskResultAsErrorLogs*

verb	URI	description
GET	/scheduler/jobs/{jobid}/tasks/{taskname}/result/log/err	Returns the standard error output (stderr) generated by the task

HTTP Example:

```
 GET /scheduler/jobs/{jobid}/tasks/{taskname}/result/log/err
sessionid: …
```

API Example:

```
 SchedulerStateRest.tasklogErr({'jobid': /* jobId the id of
the job */,
 'taskname': /* taskname the name of the task */,
 'sessionid': /* sessionId a valid session id */});
```

Output: *String* - the stderr generated by the task or an empty string if the result is not yet available
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.3.6 *getTaskResultAsOutputLogs*

verb	URI	description
GET	/scheduler/jobs/{jobid}/tasks/{taskname}/result/log/out	Returns the standard output (stdout) generated by the task

HTTP Example:

```
 GET /scheduler/jobs/{jobid}/tasks/{taskname}/result/log/out
sessionid: …
```

API Example:

```
 SchedulerStateRest.tasklogout({'jobid': /* jobId the id of
the job */,
 'taskname': /* taskname the name of the task */,
```

```
 'sessionid': /* sessionId a valid session id */});
```

Output: *String* - the stdout generated by the task or an empty string if the result is not yet available
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.3.7 *getTaskResultAsAllLogs*

verb	URI	description
GET	/scheduler/jobs/{jobid}/tasks/{taskname}/result/log/all	Returns all the logs generated by the task (either stdout and stderr)

HTTP Example:

```
 GET /scheduler/jobs/{jobid}/tasks/{taskname}/result/log/all
sessionid: …
```

API Example:

```
 SchedulerStateRest.tasklog({'jobid': /* jobId the id of the
job */,
 'taskname': /* taskname the name of the task */,
 'sessionid': /* sessionId a valid session id */});
```

Output: *String* - all the logs generated by the task (either stdout and stderr) or an empty string if the result is not yet available
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.3.8 *getTaskResultAsSerializedValue*

verb	URI	description
GET	/scheduler/jobs/{jobid}/tasks/{taskname}/result/serializedvalue	Returns the value of the task result of the task *taskName* of the job *jobId* This method returns the result as a byte array whatever the result is.

HTTP Example:

```
 GET
/scheduler/jobs/{jobid}/tasks/{taskname}/result/serializedvalu
e sessionid: …
```

API Example:

```
 SchedulerStateRest.serializedValueOftaskresult({'jobid': /*
jobId the id of the job */,
 'taskname': /* taskname the name of the task */,
 'sessionid': /* sessionId a valid session id */});
```

Output: *byte[]* - the value of the task result as a byte array.
Header parameters:

- ***sessionid*** - a valid session id

Produces: */*

17.3.1.3.9 *getTaskResultAsNotSerializedValue*

verb	URI	description
GET	/scheduler/jobs/{jobid}/tasks/{taskname}/result/value	Returns the value of the task result of task *taskName* of the job *jobId* the result is deserialized before sending to the client, if the class is not found the content is replaced by the string 'Unknown value type' .

HTTP Example:

```
 GET /scheduler/jobs/{jobid}/tasks/{taskname}/result/value
sessionid: …
```

API Example:

```
 SchedulerStateRest.valueOftaskresult({'jobid': /* jobId the
id of the job */,
 'taskname': /* taskname the name of the task */,
 'sessionid': /* sessionId a valid session id */});
```

Output: *Serializable* - the value of the task result
Header parameters:

- ***sessionid*** - a valid session id

Produces: */*

17.3.1.3.10 *getTaskState*

verb	URI	description
GET	/scheduler/jobs/{jobid}/tasks/{taskname}	get task state, identified by *taskname*.

HTTP Example:

```
 GET /scheduler/jobs/{jobid}/tasks/{taskname} sessionid: …
```

API Example:

```
 SchedulerStateRest.jobtasks({'jobid': /* jobId the id of the
job */,
 'taskname': /* taskname the name of the task */,
 'sessionid': /* sessionId a valid session id */});
```

Output: TaskStateData - the task state of the task taskname of the job jobId
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.4 *Dataspaces*

17.3.1.4.1 *pushData*

verb	URI	description
POST	/scheduler/dataspace/{spaceName}{filePath}	pushes a file from the local file system into the given DataSpace.

HTTP Example:

```
POST /scheduler/dataspace/{spaceName}{filePath} sessionid: …
```

API Example:

```
SchedulerStateRest.pushFile({'spaceName': /* spaceName the name of the DataSpace */,
'filePath': /* filePath the path inside the DataSpace where to put the file e.g. */,
'sessionid': /* sessionId a valid session id */,
'$entity': /* multipart the form data containing : - fileName the name of the file that will be created on the DataSpace - fileContent the content of the file */});
```

Input: *MultipartFormDataInput* - the form data containing : - fileName the name of the file that will be created on the DataSpace - fileContent the content of the file
Output: *boolean* - true if the transfer succeeded
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.1.4.2 *pullData*

verb	URI	description
GET	/scheduler/dataspace/{spaceName}{filePath}	either pulls a file from the given DataSpace to the local file system or lists the content of a directory, if the path refers to a directory.

HTTP Example:

```
GET /scheduler/dataspace/{spaceName}{filePath} sessionid: …
```

API Example:

```
SchedulerStateRest.pullFile({'spaceName': /* spaceName the name of the data space involved (GLOBAL or USER) */,
```

```
 'filePath': /* filePath the path to the file or directory
whose content must be received */,
 'sessionid': /* sessionId a valid session id */});
```

Output: *InputStream*
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/octet-stream

17.3.1.4.3 *pullData*

verb	URI	description
DELETE	/scheduler/dataspace/{spaceName}{filePath}	deletes a file or recursively deletes a directory from the given DataSpace.

HTTP Example:

```
 DELETE /scheduler/dataspace/{spaceName}{filePath} sessionid:
…
```

API Example:

```
 SchedulerStateRest.deleteFile({'spaceName': /* spaceName the
name of the data space involved (GLOBAL or USER) */,
 'filePath': /* filePath the path to the file or directory
which must be deleted */,
 'sessionid': /* sessionId a valid session id */});
```

Output: *boolean*
Header parameters:

- ***sessionid*** - a valid session id

17.3.2 Resource Manager Service

17.3.2.1 ***Common***

17.3.2.1.1 *login*

verb	URI	description
POST	/rm/login	enable user to access the RM with his credentials.

HTTP Example:

```
 POST /rm/login username=…&password=…
```

API Example:

```
 RMRest.rmConnect({'username': /* Log into the resource
manager using an form containing 2 fields */,
 'password': /* Log into the resource manager using an form
containing 2 fields */});
```

Output: *String* - the sessionid of the user if succeed
Form parameters:

- ***username***
- ***password***

Produces: application/json

17.3.2.1.2 *disconnect*

verb	URI	description
POST	/rm/disconnect	allow the user to disconnect from the RM.

HTTP Example:

```
 POST /rm/disconnect sessionid: …
```

API Example:

```
 RMRest.rmDisconnect({'sessionid': /* sessionId a valid
session id */});
```

Output: *void*
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.1.3 *isActive*

verb	URI	description
GET	/rm/isactive	test if the RM is operational.

HTTP Example:

```
 GET /rm/isactive sessionid: …
```

API Example:

```
 RMRest.isActive({'sessionid': /* sessionId a valid session id
*/});
```

Output: *boolean* - true if the resource manager is operational.
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.1.4 *shutdownRM*

verb	URI	description
GET	/rm/info/{name}?attr=…	Initiate the shutdowns the resource manager.

HTTP Example:

```
GET /rm/info/{name}?attr=… sessionid: …
```

API Example:

```
RMRest.getMBeanInfo({'attr': /* attrs attributes to enumerate
*/,
'name': /* name mbean's object name */,
'sessionid': /* sessionId a valid session */});
```

Output: *Object* - returns the attributes of the mbean
Query parameters:
- ***attr*** - attributes to enumerate

Header parameters:
- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.1.5 *getRMInfo*

verb	URI	description
GET	/rm/shutdown	retrieve specific RM information by furnishing the *name* of available resource.

HTTP Example:

```
GET /rm/shutdown?preempt=… sessionid: …
```

API Example:

```
RMRest.shutdown({'preempt': /* preempt if true shutdown
immediatly whithout waiting for nodes to be freed, default
value is false */,
'sessionid': /* sessionId a valid session */});
```

Output: *boolean* - true if the resource manager is operational.
Header parameters:
- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.1.6 *getMonitoring*

verb	URI	description
GET	/rm/monitoring	get the initial state of the RM, included current deployed Node Sources, Nodes and Policies info.

HTTP Example:

```
GET /rm/monitoring sessionid: …
```

API Example:

```
RMRest.getInitialState({'sessionid': /* sessionId a valid
session id */});
```

Output: *RMInitialState* - the initial state of the resource manager
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.1.7 *getRMStatHistory*

verb	URI	description
GET	/rm/stathistory?range=…	return the statistic history.

HTTP Example:

```
GET /rm/stathistory?range=… sessionid: …
```

API Example:

```
RMRest.getStatHistory({'range': /* range a String of 5 chars,
one for each stat history source, indicating the time range to
fetch for each source. */,
'sessionid': /* sessionId a valid session */});
```

Output: *String* - a JSON object containing a key for each source
Query parameters:

- ***range*** - a String of 5 chars, one for each stat history source, indicating the time range to fetch for each source. Each char can be:
  - 'a' 1 minute
  - 'm' 10 minutes
  - 'h' 1 hour
  - 'H' 8 hours
  - 'd' 1 day
  - 'w' 1 week
  - 'M' 1 month
  - 'y' 1 year

Header parameters:

- ***sessionid*** - a valid session

Produces: application/json

17.3.2.1.8 *getRMState*

verb	URI	description

GET	/rm/state	return an overview about the current free/alive/total nodes number of the RM.

HTTP Example:

```
GET /rm/state sessionid: …
```

API Example:

```
RMRest.getState({'sessionid': /* sessionId a valid session id
*/});
```

Output: *RMState* - Returns the state of the scheduler
Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.1.9 *getRMVersion*

verb	URI	description
GET	/rm/version	return the current REST server API and the RM version.

HTTP Example:

```
GET /rm/version
```

API Example:

```
RMRest.getVersion({});
```

Output: *String* - returns the version of the rest api

17.3.2.2 ***Node Source***

17.3.2.2.1 *createNodeSource*

verb	URI	description
POST	/rm/nodesource/create	create a new node source in the RM, specifying infrastructure and policy, with related parameters.

HTTP Example:

```
POST /rm/nodesource/create sessionid: …
nodeSourceName=…&infrastructureType=…&infrastructureParameters
=…&infrastructureFileParameters=…&policyType=…&policyParameter
s=…&policyFileParameters=…
```

API Example:

```
RMRest.createNodeSource({'sessionid': /* sessionId current
session id */,
```

```
 'nodeSourceName': /* nodeSourceName name of the node source
to create */,

 'infrastructureType': /* infrastructureType fully qualified
class name of the infrastructure to create */,

 'infrastructureParameters': /* infrastructureParameters
String parameters of the infrastructure, without the
parameters containing files or credentials */,

 'infrastructureFileParameters': /*
infrastructureFileParameters File or credential parameters */,

 'policyType': /* policyType fully qualified class name of the
policy to create */,

 'policyParameters': /* policyParameters String parameters of
the policy, without the parameters containing files or
credentials */,

 'policyFileParameters': /* policyFileParameters File or
credential parameters */});
```

Output: *boolean* - true if a node source has been created
Form parameters:

- ***nodeSourceName*** - name of the node source to create
- ***infrastructureType*** - fully qualified class name of the infrastructure to create
- ***infrastructureParameters*** - String parameters of the infrastructure, without the parameters containing files or credentials
- ***infrastructureFileParameters*** - File or credential parameters
- ***policyType*** - fully qualified class name of the policy to create
- ***policyParameters*** - String parameters of the policy, without the parameters containing files or credentials
- ***policyFileParameters*** - File or credential parameters

Produces: application/json

17.3.2.2.2 *removeNodeSource*

verb	URI	description
POST	/rm/nodesource/remove	remove a new node source from the RM.

HTTP Example:

```
 POST /rm/nodesource/remove sessionid: … name=…&preempt=…
```

API Example:

```
 RMRest.removeNodeSource({'sessionid': /* sessionId a valid
session id */,

 'name': /* sourceName a node source */,

 'preempt': /* preempt if true remove node source immediatly
whithout waiting for nodes to be freed */});
```

Output: *boolean* - true if the node is removed successfully, false or exception otherwise

Form parameters:

- ***name*** - a node source
- ***preempt*** - if true remove node source immediatly whithout waiting for nodes to be freed

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.2.3 *getSupportedInfrastructures*

verb	URI	description
GET	/rm/infrastructures	return the list of supported node source infrastructures descriptors.

HTTP Example:

```
GET /rm/infrastructures sessionid: …
```

API Example:

```
RMRest.getSupportedNodeSourceInfrastructures({'sessionid': /*
sessionId a valid session */});
```

Output: *Collection<PluginDescriptor>* - the list of supported node source infrastructures descriptors

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.2.4 *getSupportedPolicies*

verb	URI	description
GET	/rm/policies	return the list of supported node source policies descriptors.

HTTP Example:

```
GET /rm/policies sessionid: …
```

API Example:

```
RMRest.getSupportedNodeSourcePolicies({'sessionid': /*
sessionId a valid session */});
```

Output: *Collection<PluginDescriptor>* - the list of supported node source policies descriptors

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.3 ***Nodes***

17.3.2.3.1 *isNodeAvailable*

verb	URI	description
GET	/rm/node/isavailable?nodeurl=…	test if a node is registered to the RM.

HTTP Example:

```
GET /rm/node/isavailable?nodeurl=… sessionid: …
```

API Example:

```
RMRest.nodeIsAvailable({'nodeurl': /* url the url of the node
*/,
 'sessionid': /* sessionId a valid session id */});
```

Output: *boolean* - true if the node nodeUrl is registered and not down
Query parameters:

- ***nodeurl*** - the url of the node

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.3.2 *lockNode*

verb	URI	description
POST	/rm/node/lock	prevent other users from using a set of locked nodes.

HTTP Example:

```
POST /rm/node/lock sessionid: … nodeurls=…
```

API Example:

```
RMRest.lockNodes({'sessionid': /* sessionId current session
*/,
 'nodeurls': /* nodeUrls set of node urls to lock */});
```

Output: *boolean* - true when all nodes were free and have been locked
Form parameters:

- ***nodeurls*** - set of node urls to lock

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.3.3 *unlockNode*

verb	URI	description
POST	/rm/node/unlock	allow other users to use a set of nodes previously locked.

HTTP Example:

```
POST /rm/node/unlock sessionid: … nodeurls=…
```

API Example:

```
RMRest.unlockNodes({'sessionid': /* sessionId current session
*/,
'nodeurls': /* nodeUrls set of node urls to unlock */});
```

Output: *boolean* - true when all nodes were locked and have been unlocked
Form parameters:

- ***nodeurls*** - set of node urls to unlock

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.3.4 *releaseNode*

verb	URI	description
POST	/rm/node/release	release a node, previously reserved for computation.

HTTP Example:

```
POST /rm/node/release sessionid: … url=…
```

API Example:

```
RMRest.releaseNode({'sessionid': /* sessionId a valid session
id */,
'url': /* url node's URL */});
```

Output: *boolean* - true of the node has been released
Form parameters:

- ***url*** - node's URL

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.3.5 *addNode*

verb	URI	description
POST	/rm/node	add a node to a particular node source. If not specified, add it to the default node source of RM

HTTP Example:

```
POST /rm/node sessionid: … nodeurl=…&nodesource=…
```

API Example:

```
RMRest.addNode({'sessionid': /* sessionId a valid session id
*/,
'nodeurl': /* url the url of the node */,
```

```
 'nodesource': /* nodesource the node source, can be null
*/});
```

Output: *boolean* - true if new node is added successfully, runtime exception otherwise
Form parameters:

- ***nodeurl*** - the url of the node
- ***nodesource*** - the node source, can be null

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.2.3.6 *removeNode*

verb	URI	description
POST	/rm/node	remove a node from the RM.

HTTP Example:

```
 POST /rm/node/remove sessionid: … url=…&preempt=…
```

API Example:

```
 RMRest.removeNode({'sessionid': /* sessionId a valid session
id */,
 'url': /* nodeUrl node's URL */,
 'preempt': /* preempt if true remove node source immediatly
whithout waiting for nodes to be freed */});
```

Output: *boolean* - true if the node is removed successfully, false or exception otherwise
Form parameters:

- ***nodeurl*** - the url of the node
- ***preempt*** - if true remove node source immediatly whithout waiting for nodes to be freed

Header parameters:

- ***sessionid*** - a valid session id

Produces: application/json

17.3.3 Common Data structure

17.3.3.1 ***exception***

Elements

name	type	required	nillable
errorMessage	xsd:string	false	false

exception	java.lang.Throwable	false	false
exceptionClass	xsd:string	false	false
httpErrorCode	xsd:int	false	false
stackTrace	xsd:string	false	false

17.3.3.2 *JobIdData*

Elements

name	type	required	nillable
id	xsd:long	false	false
readableName	xsd:string	false	false

XML Example:

```
<JobIdData>
  <id>xsd:long</id>
  <readableName>xsd:string</readableName>
</JobIdData>
```

JSON Example:

```
{"JobIdData":
 {
  "id": Number,
  "readableName": String,
 }
}
```

17.3.3.3 *JobInfoData*

Elements

name	type	required	nillable
finishedTime	xsd:long	false	false
jobId	JobIdData	false	false

jobOwner	xsd:string	false	false
numberOfFinishedTasks	xsd:int	false	false
numberOfPendingTasks	xsd:int	false	false
numberOfRunningTasks	xsd:int	false	false
priority	LOWEST | LOW | NORMAL | HIGH | HIGHEST	false	false
removedTime	xsd:long	false	false
startTime	xsd:long	false	false
status	RUNNING | STALLED | FINISHED | PAUSED | CANCELED | FAILED | KILLED	false	false
submittedTime	xsd:long	false	false
totalNumberOfTasks	xsd:int	false	false

XML Example:

```
 <JobInfoData>
  <finishedTime>xsd:long</finishedTime>
  <jobId>JobIdData</jobId>
  <jobOwner>xsd:string</jobOwner>
  <numberOfFinishedTasks>xsd:int</numberOfFinishedTasks>
  <numberOfPendingTasks>xsd:int</numberOfPendingTasks>
  <numberOfRunningTasks>xsd:int</numberOfRunningTasks>
  <priority>IDLE | LOWEST | LOW | NORMAL | HIGH |
HIGHEST</priority>
  <removedTime>xsd:long</removedTime>
  <startTime>xsd:long</startTime>
  <status>PENDING | RUNNING | STALLED | FINISHED | PAUSED |
CANCELED | FAILED | KILLED</status>
  <submittedTime>xsd:long</submittedTime>
  <totalNumberOfTasks>xsd:int</totalNumberOfTasks>
 </JobInfoData>
```

JSON Example:

```
  {"JobInfoData":
   {
   "finishedTime": Number,
   "jobId": JobIdData,
   "jobOwner": String,
   "numberOfFinishedTasks": Number,
   "numberOfPendingTasks": Number,
   "numberOfRunningTasks": Number,
   "priority": 'IDLE' | 'LOWEST' | 'LOW' | 'NORMAL' | 'HIGH' |
'HIGHEST',
   "removedTime": Number,
   "startTime": Number,
   "status": 'PENDING' | 'RUNNING' | 'STALLED' | 'FINISHED' |
'PAUSED' | 'CANCELED' | 'FAILED' | 'KILLED',
   "submittedTime": Number,
   "totalNumberOfTasks": Number,
   }
  }
```

17.3.3.4 *JobResultData*

Elements

name	type	required	nillable
allResults	java.util.Map	false	false
id	JobIdData	false	false

XML Example:

```
 <JobResultData>
  <allResults>java.util.Map</allResults>
  <id>JobIdData</id>
 </JobResultData>
```

JSON Example:

```
 {"JobResultData":
  {
  "allResults": java.util.Map,
  "id": JobIdData,
```

```
  }
}
```

17.3.3.5 *JobStateData*

Elements

name	type	required	nillable
id	xsd:long	false	false
jobInfo	JobInfoData	false	false
name	xsd:string	false	false
owner	xsd:string	false	false
priority	xsd:string	false	false
projectName	xsd:string	false	false
tasks	java.util.Map	false	false

XML Example:

```
<JobStateData>
  <id>xsd:long</id>
  <jobInfo>JobInfoData</jobInfo>
  <name>xsd:string</name>
  <owner>xsd:string</owner>
  <priority>xsd:string</priority>
  <projectName>xsd:string</projectName>
  <tasks>java.util.Map</tasks>
 </JobStateData>
```

JSON Example:

```
{"JobStateData":
 {
 "id": Number,
 "jobInfo": JobInfoData,
 "name": String,
 "owner": String,
```

```
   "priority": String,
   "projectName": String,
   "tasks": java.util.Map,
   }
  }
```

17.3.3.6 *JobUsageData*

Elements

name	type	required	nillable
jobDuration	xsd:long	false	false
jobId	xsd:string	false	false
jobName	xsd:string	false	false
taskUsages	zero or N[TaskUsageData]	false	false

XML Example:

```
 <JobUsageData>
  <jobDuration>xsd:long</jobDuration>
  <jobId>xsd:string</jobId>
  <jobName>xsd:string</jobName>
  zero or N[<taskUsages>TaskUsageData</taskUsages>]
 </JobUsageData>
```

JSON Example:

```
 {"JobUsageData":
  {
  "jobDuration": Number,
  "jobId": String,
  "jobName": String,
  "taskUsages": [TaskUsageData],
  }
 }
```

17.3.3.7 *Login form*

a class that represent a mean to submit credential through the rest api. A credential can be either - a username/password couple - a file submitted through an HTTP form and whose name is 'credential'

Elements

name	type	required	nillable
credential	java.io.InputStream	false	false
password	xsd:string	false	false
sshKey	xsd:base64Binary	false	false
username	xsd:string	false	false

XML Example:

```
<LoginForm>
  <credential>java.io.InputStream</credential>
  <password>xsd:string</password>
  <sshKey>xsd:base64Binary</sshKey>
  <username>xsd:string</username>
</LoginForm>
```

JSON Example:

```
{"LoginForm":
 {
 "credential": java.io.InputStream,
 "password": String,
 "sshKey": [Number],
 "username": String,
 }
}
```

17.3.3.8 *ParallelEnvironmentData*

Elements

name	type	required	nillable
nodesNumber	xsd:int	false	false

XML Example:

```
<ParallelEnvironmentData>
 <nodesNumber>xsd:int</nodesNumber>
</ParallelEnvironmentData>
```

JSON Example:

```
{"ParallelEnvironmentData":
 {
 "nodesNumber": Number,
 }
}
```

17.3.3.9 *SchedulerUserData*

Elements

name	type	required	nillable
connectionTime	xsd:long	false	false
hostName	xsd:string	false	false
lastSubmitTime	xsd:long	false	false
submitNumber	xsd:int	false	false
username	xsd:string	false	false

XML Example:

```
<SchedulerUserData>
  <connectionTime>xsd:long</connectionTime>
  <hostName>xsd:string</hostName>
  <lastSubmitTime>xsd:long</lastSubmitTime>
  <submitNumber>xsd:int</submitNumber>
  <username>xsd:string</username>
</SchedulerUserData>
```

JSON Example:

```
{"SchedulerUserData":
 {
"connectionTime": Number,
"hostName": String,
```

```
  "lastSubmitTime": Number,
  "submitNumber": Number,
  "username": String,
   }
  }
```

17.3.3.10 *TaskIdData*

Elements

name	type	required	nillable
id	xsd:long	false	false
readableName	xsd:string	false	false

XML Example:

```
 <TaskIdData>
  <id>xsd:long</id>
  <readableName>xsd:string</readableName>
 </TaskIdData>
```

JSON Example:

```
  {"TaskIdData":
   {
    "id": Number,
    "readableName": String,
   }
  }
```

17.3.3.11 *TaskInfoData*

Elements

name	type	required	nillable
executionDuration	xsd:long	false	false
executionHostName	xsd:string	false	false
finishedTime	xsd:long	false	false

numberOfExecutionLeft	xsd:int	false	false
numberOfExecutionOnFailureLeft	xsd:int	false	false
startTime	xsd:long	false	false
taskId	TaskIdData	false	false
taskStatus	PENDING | PAUSED | RUNNING | WAITING_ON_ERROR | WAITING_ON_FAILURE | FAILED | NOT_STARTED | NOT_RESTARTED | ABORTED | FAULTY | FINISHED | SKIPPED	false	false
username	xsd:string	false	false

XML Example:

```
 <TaskInfoData>
  <executionDuration>xsd:long</executionDuration>
  <executionHostName>xsd:string</executionHostName>
  <finishedTime>xsd:long</finishedTime>
  <numberOfExecutionLeft>xsd:int</numberOfExecutionLeft>

<numberOfExecutionOnFailureLeft>xsd:int</numberOfExecutionOnFa
ilureLeft>
  <startTime>xsd:long</startTime>
  <taskId>TaskIdData</taskId>
  <taskStatus>SUBMITTED | PENDING | PAUSED | RUNNING |
WAITING_ON_ERROR | WAITING_ON_FAILURE
  | FAILED | NOT_STARTED | NOT_RESTARTED | ABORTED | FAULTY |
FINISHED | SKIPPED</taskStatus>
 </TaskInfoData>
```

JSON Example:

```
 {"TaskInfoData":
  {
  "executionDuration": Number,
  "executionHostName": String,
  "finishedTime": Number,
  "numberOfExecutionLeft": Number,
```

```
  "numberOfExecutionOnFailureLeft": Number,
  "startTime": Number,
  "taskId": TaskIdData,
  "taskStatus": 'SUBMITTED' | 'PENDING' | 'PAUSED' | 'RUNNING'
| 'WAITING_ON_ERROR' | 'WAITING_ON_FAILURE' |
  'FAILED' | 'NOT_STARTED' | 'NOT_RESTARTED' | 'ABORTED' |
'FAULTY' | 'FINISHED' | 'SKIPPED',
  }
 }
```

17.3.3.12 *TaskResultData*

Elements

name	type	required	nillable
id	TaskIdData	false	false
serializedValue	xsd:base64Binary	false	false

XML Example:

```
 <TaskResultData>
  <id>TaskIdData</id>
  <serializedValue>xsd:base64Binary</serializedValue>
 </TaskResultData>
```

JSON Example:

```
 {"TaskResultData":
  {
  "id": TaskIdData,
  "serializedValue": [Number],
  }
 }
```

17.3.3.13 *TaskStateData*

Elements

name	type	required	nillable
description	xsd:string	false	false

iterationIndex	xsd:int	false	false
maxNumberOfExecution	xsd:int	false	false
maxNumberOfExecutionOnFailure	xsd:int	false	false
name	xsd:string	false	false
numberOfNodesNeeded	xsd:int	false	false
parallelEnvironment	ParallelEnvironmentData	false	false
replicationIndex	xsd:int	false	false
taskInfo	TaskInfoData	false	false

XML Example:

```
 <TaskStateData>
  <description>xsd:string</description>
  <iterationIndex>xsd:int</iterationIndex>
  <maxNumberOfExecution>xsd:int</maxNumberOfExecution>

<maxNumberOfExecutionOnFailure>xsd:int</maxNumberOfExecutionOn
Failure>
  <name>xsd:string</name>
  <numberOfNodesNeeded>xsd:int</numberOfNodesNeeded>

<parallelEnvironment>ParallelEnvironmentData</parallelEnvironm
ent>
  <replicationIndex>xsd:int</replicationIndex>
  <taskInfo>TaskInfoData</taskInfo>
 </TaskStateData>
```

JSON Example:

```
 {"TaskStateData":
  {
   "description": String,
   "iterationIndex": Number,
   "maxNumberOfExecution": Number,
   "maxNumberOfExecutionOnFailure": Number,
   "name": String,
```

```
   "numberOfNodesNeeded": Number,
   "parallelEnvironment": ParallelEnvironmentData,
   "replicationIndex": Number,
   "taskInfo": TaskInfoData,
  }
 }
```

17.3.3.14 *TaskUsageData*

Elements

name	type	required	nillable
taskExecutionDuration	xsd:long	false	false
taskFinishedTime	xsd:long	false	false
taskId	xsd:string	false	false
taskName	xsd:string	false	false
taskNodeNumber	xsd:int	false	false
taskStartTime	xsd:long	false	false

XML Example:

```
 <TaskUsageData>
  <taskExecutionDuration>xsd:long</taskExecutionDuration>
  <taskFinishedTime>xsd:long</taskFinishedTime>
  <taskId>xsd:string</taskId>
  <taskName>xsd:string</taskName>
  <taskNodeNumber>xsd:int</taskNodeNumber>
  <taskStartTime>xsd:long</taskStartTime>
 </TaskUsageData>
```

JSON Example:

```
 {"TaskUsageData":
  {
  "taskExecutionDuration": Number,
  "taskFinishedTime": Number,
  "taskId": String,
```

```
  "taskName": String,
  "taskNodeNumber": Number,
  "taskStartTime": Number,
  }
 }
```

17.3.3.15 *UserJobData*

A class that contains a subset of the information available in a scheduler state. It is mostly used to provide a fast access to meaningful data within the scheduler state without having to manage the complete state

Elements

name	type	required	nillable
jobid	xsd:string	false	false
jobInfo	JobInfoData	false	false
jobOwner	xsd:string	false	false

XML Example:

```
  <UserJobData>
   <jobid>xsd:string</jobid>
   <jobInfo>JobInfoData</jobInfo>
   <jobOwner>xsd:string</jobOwner>
  </UserJobData>
```

JSON Example:

```
  {"UserJobData":
   {
   "jobid": String,
   "jobInfo": JobInfoData,
   "jobOwner": String,
   }
  }
```

18 FIWARE OpenSpecification Cloud Edgelets

18.1 Copyright

Copyright © 2013 Thales Communications & Security

18.2 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

18.3 Overview

This specification describes the Edgelets Generic Enabler, which is designed for improving the end-users web experience by allowing some of web interactions (media upload or download, logical interaction, ...) to happen on devices located near the user with high data rates and low latency.

18.3.1 GE Description

The Edgelets GE provides the means to manage and deploy a set of distributed pieces of applications called "edgelets". The edgelets are software pieces used to move some of the logic or ressources for a web application close to the user.

Two main use cases are envisioned:

18.3.1.1 *Logic Distribution Network*

This use case can be seen as some kind of "dynamic CDN" (Content Delivery Network). Whereas CDNs are mainly targetting at delivering static content to end-users, edgelets can be used to deliver logical pieces of a web application, like captchas, buttons (like the twitter or facebook buttons which displays number of tweets or likes), or to realize some services like transcoding of videos to a format that is best suited for the end-user.

18.3.1.2 *Local Webapp Companion*

In this use cas, edgelets are installed proactively by the end-user (for example on the user's set-top box) in order to enhance its web experience twoards a specific website using capabilities offered by the edgelet (storage, computation, ...). This allows to quickly upload movies or pictures to the local edgelet, that are then synchronized with the remote web site, while keeping the website usage flow untouched (It should introduce as less extra complexity and differences in the interface as possible).

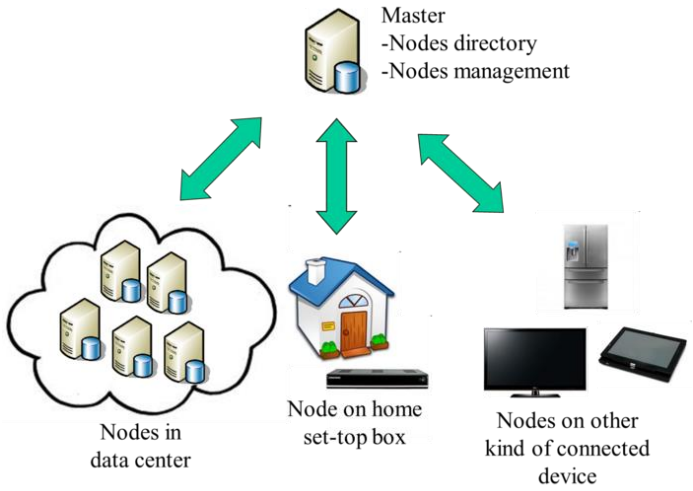
18.3.2 GE Architecture and Components

18.3.2.1 *Edgelets Infrastructure*

The edgelet infrastructure is composed of:

- A master node that manages slave nodes

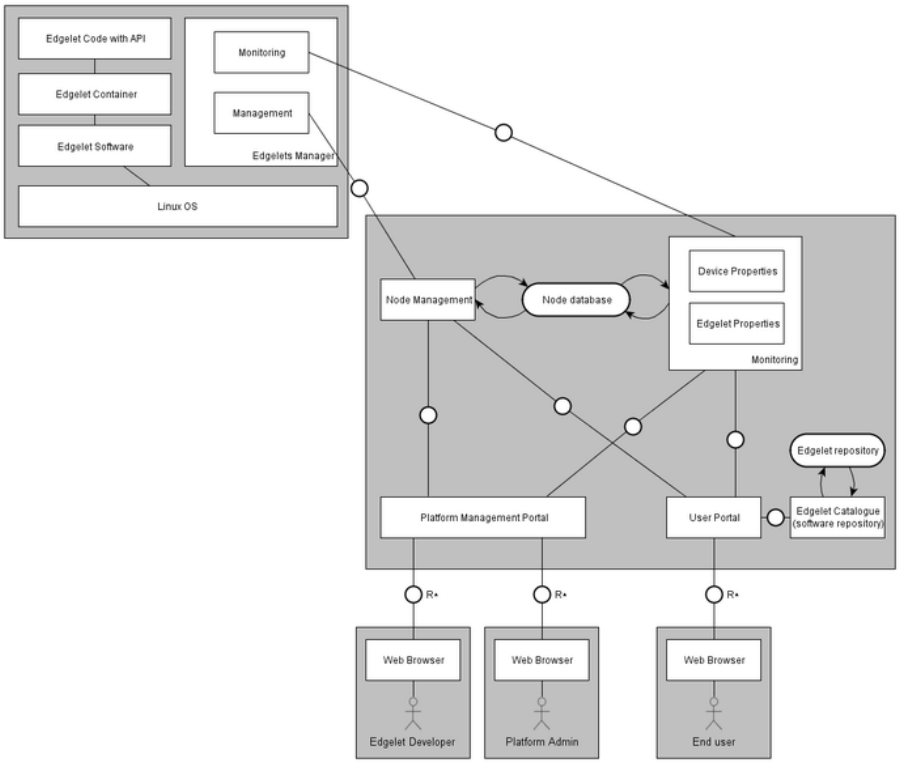
- A set of slaves nodes, distributed accross the internet and running the edgelets



Basic infrastructure of the Edglets GE

18.3.2.2 *Edgelets Main Components*

The following diagram shows the main components of the Edgelets GE:



High level architecture of the Edglets GE

18.3.2.2.1 *Master Node Components*

The master node is the central point for managing the Edgelets GE infrastructure.

The Node Management component is the most important components since it handles:

- which nodes belong to the Edgelets infrastructure

- which edgelets should run on the different slave nodes
- from a centralized point

The Monitoring component aggregates the monitoring information from the different slave nodes and provides an API for requesting such information.

These two components are highly interacting with two web portals:

- the User Portal used by end-users for proactively managing edgelets deployed on nodes when relevant (especially for the local webapp companion use case)
- the Platform Management Portal is used by Edgelet Developers and the Platform Administrator for:
    - Edgelet Developer
        - managing its own software releases
        - managing and monitoring the deployment of the edgelets on the slave nodes for the logic distribution network use case
    - Platform Administrator
        - managing the list of available edgelets
        - managing the deployment and life-cycle of the edgelets
        - managing the slave nodes
        - monitoring the edgelets and nodes

Note that the monitoring component and the portals are not available in the first release.

18.3.2.2.2 *Slave node components*

A slave node only has minimal software requirements in order to be able to run edgelets (A Linux based OS and some build tools).

The Management component is in charge of managing the edgelets running on the device:

- Synchronize with the master node in order to know which edgelets should run
- Download and build the software used to run a specific edgelet
- Run an edgelet

The Monitoring component is used to report usage and statistics on the device and the edgelets to the master node.

The edgelets themselves are instances of software releases running inside a container.

18.4 Basic Concepts

The key concepts of the GE are:

- The edgelets can be deployed on a wide variety of devices, with only minimal requirements. They could be as well deployed in a datacenter, on set-top boxes, or any kind of connected device (tablets, smart tvs, ...), as long as they fullfil the basic requirements. The overhead of the edgelets is minimal

compared to running several virtual machines on a device with limited resources.

- The edgelets are useful when they are close to the end-user (the idea is the same as the one behind CDNs). In order to achieve this, slave nodes are spread in various locations over the entire Internet, and the node management has knowledge about the location of the slave nodes. It enables a fine deployment of the edgelets in the different use cases.
- Several edgelets can be run on each of the slave nodes
- Monitoring is used to ensure the optimal use and deployment of the edgelets throughout the distributed infrastructure

18.5 API Operations

The main interfaces of the Edglets GE for this release are listed below. These are the interfaces provided by the Master node and exposed to the users (Platform admins, edgelet developers, end users, ...)

18.5.1 Slave nodes

- **listSlaveNodes** -- return a list of registered slave nodes
- **getSlaveNodeDetails** -- return information on the slave node (state, capabilities, ...)

18.5.2 Edgelets

- **listEdgelets** -- return a list of edgelets available on a node
- **getEdgeletDetails** -- retrieve information on an edgelet on a node (state, metrics, ...)
- **provisionEdgelet** -- build the software necessary to run an edgelet
- **startEdgelet** -- start and edgelet
- **stopEdgelet** -- stop an edgelet
- **updateEdgelet** -- update an edgelet to a newer version
- **deleteEdgelet** -- delete an edgelet

18.5.3 Software repository

- **listSoftwareReleases** -- return a list of available edgelets releases
- **getSoftwareReleaseDetails** -- return information on a edgelet release (author, version, ...)
- **uploadSoftwareRelease** -- upload a new edgelet release in the catalogue

18.6 Basic Design Principles

When applied to DCRM, the general design principles outlined at Cloud Hosting Architecture can be translated into the following key design goals:

- Ability to dynamically control the deployed edgelets as well as to monitor the actual usage
- Avoid non-authorized access to the slave nodes and deployed edgelets
- Automated provisioning and life cycle of the edgelets

- Broad network access -- portals can be accessed by several heterogeneous end-devices

18.7 Detailed Specifications

Following is a list of Open Specifications linked to this Generic Enabler. Please note that the API is subject to small changes for next versions.

18.7.1 Open API Specifications

- Edgelets Open API Specification (DRAFT)

18.8 Re-utilised Technologies/Specifications

The Edgelets GE reuses internally the slapos software stack https://www.slapos.org/, which aims at providing decentralized cloud computing

19 Edgelets Open API Specification

19.1 Introduction

Please check the FI-WARE Open Specifications Legal Notice to understand the rights to use FI-WARE Open Specifications.

19.2 Edgelets Management API

Please note that this is a very draft version of the API which is lacking many features. Moreover, due to technical limitations in the current implementation, the current API lacks consistency and will be subject to changes. Future releases will provide a more user-friendly and clear API

19.2.1 How to Read This Document

All FI-WARE RESTful API specifications will follow the same list of conventions and will support certain common aspects. Please check Common aspects in FI-WARE Open Restful API Specifications.

19.2.2 Intended Audience

This specification is intended (with its current state) for Cloud Operators. It indicates how to manage an edgelet network. For a concrete example of using these APIs, please look Edgelets - User and Programmers Guide

19.2.3 API Change History

This is the first draft version of the API. It is expected to change significantly with next releases.

19.2.4 Additional Resources

More documentation related to the architecture of the GE is available in the Edgelet Open Specification

19.3 General API information

19.3.1 Authentication

Each API operation requires authentication. In order to authenticate, you have to provide a token as an Authorization HTTP header. This token is retrieved by posting username and password to the /authenticate URL.

19.3.2 Implemented methods

- getServers is used to retrieve the list of servers available to the user
- getServer is used to retrieve specific information on a sevrer

- getServices is used to retrieve a list of deployed edgelets
- getService is used to retrieve specific information of an edgelet, especially the endpoints
- newServer is used to register a new node to the network
- newSoftwareRelease is used to request the installation of a software release (the base code of an edgelet) on a node
- newInstance is used to request the instanciation of an edgelet on a node
- listSoftwareReleases returns a list of available edgelets releases
- getSoftwareReleaseDetails returns information on a edgelet release (author, version, ...)

19.3.3 Representation format

The API supports JSON-based representation

19.3.4 Representation transport

Resource representation is transmitted between client and server by using HTTPs 1.1 protocol.

19.4 API operations

19.4.1.1 *Get a list of servers*

verb	URI	description
GET	/servers	list of available servers

Normal response code: 200

This operation does not require a request body.

This operation returns a list of available servers

Example:

[

```
 {
   "title": "myserver",
   "reference": "COMP-12",
   "id": "20130130-4C766B"
 }
```

]

19.4.2 Get a list of edgelets

verb	URI	description

GET	/services	list of available edgelets

Normal response code: 200

This operation does not require a request body.

This operation returns a list of available edgelets

Example:

```
[
 {
   "title": "myedgelet",
   "id": "20130130-4C766B"
 },
 {
   "title": "myedgelet2",
   "id": "20130130-4C7xx"
 }
]
```

19.4.3 Get information on an edgelet

verb	URI	description
GET	/service/:id	information on an edgelet

Normal response code: 200

This operation does not require a request body.

This operation returns information on an edgelet (status, endpoints, ...)

Example:

```
{
  "id": "20130130-4C766B",
  "url": "",
  "status": "Instance correctly started",
  "params": [
    {
      "key": "backend_url",
      "value":
"http://[2001:470:1f14:169:719a:e730:b364:3faf]:50000/"
    }
  ]
}
```

19.4.4 get information on a server

verb	URI	description
GET	/server/:id	information on the server

Normal response code: 200

This operation does not require a request body. The id must be a valid one.

This operation returns information on the server.

Example:

```
{
 "id": "20130130-4C766B",
 "title": "myserver",
 "software_releases": [
   {
     "title": "edgeletproduct1",
     "version": "0.1",
     "state": "Installation requested"
   }
 ]
}
```

19.4.5 Register a new server

verb	URI	description
POST	/servers	Register a new server

Normal response code: 200

The following params must be passed as POST parameters

name	example	description
name	"myserver"	Server title

This operation returns a key and a certificate used to authenticate the new node to the master.

Example:

```
{
 "key": <key text>,
 "certificate": <certificate text>
}
```

19.4.6 Request a software release on a server

verb	URI	description
POST	/software_releases	Request software release installation

The following params must be passed as POST parameters

name	example	description
server	"COMP-XXX"	Server reference (not id)
software_release_url	"https://raw.github.com/besson/slapos/master/software/simple-example2/software.cfg"	Server release configuration file

Normal response code: 200. This operation returns nothing.

19.4.7 List available edgelets on the catalog

verb	URI	description
GET	/catalog/edgelets	List available edgelets

Normal response code: 200

This operation does not require a request body.

This operation returns a list of available edgelets

Example: [

```
 {
   "title": "myedgelet1",
   "url":
"https://raw.github.com/besson/slapos/master/software/simple-
example2/software.cfg",
   "id": "20130130-4C766B"
 },
 {
   "title": "myedgelet2",
   "url":
"https://raw.github.com/besson/slapos/master/software/simple-
example1/software.cfg",
   "id": "20140530-4C766B"
```

]

19.4.8 Get information on an edgelet in the catalog

verb	URI	description
GET	/catalog/edgelets/:id	Information on the edgelet

Normal response code: 200

This operation does not require a request body.

This operation returns a list of metadata on the edgelet

19.4.9 Instanciate an edgelet on a node

verb	URI	description
POST	/services	Request edgelet instantiation

The following params must be passed as POST parameters

name	example	description
name	"myedgeletinstance"	instance name
server_reference	"COMP-XXX"	Server reference (not id)
software_release_url	"https://raw.github.com/besson/slapos/master/software/simple-example2/software.cfg"	Server release configuration file

Normal response code: 200. This operation returns nothing.