Reference Architecture for StratusLab Toolkit 2.0

Marc-Elian Bégin, Konstantin Skaburskas, Louise Merifield, Charles Loomis, Eduardo Huedo, Stuart Kenny, Henar Muñoz Frutos

Enhancing Grid Infrastructures with
Virtualization and Cloud Technologies

Reference Architecture for StratusLab Toolkit 2.0

Deliverable D4.4 (V1.0)
3 October 2011

Abstract

The document describes the updated Reference Architecture for StratusLab v2.0, building on D4.1, which described the architecture for StratusLab v1.0. This document contains two main parts, the overall architecture identifying the required services and components of which it is composed, followed by detailed descriptions for each of these elements. For v2.0, the architecture is rationalized and consolidated. For example, the policy validation, caching and cloud storage are integrated as default features. The Appliance Repository is replaced by the cloud storage service. Several services are also added, such as cloud storage, virtual network provisioning and inter-cloud connectivity. From v1.0 to v2.0, incremental versions of the StratusLab distribution will be built and released to provide an increasing amount of the functionality identified in this document, alongside further improvements to the robustness of the distribution.

StratusLab is co-funded by the European Community's Seventh Framework Programme (Capacities) Grant Agreement INFSO-RI-261552.

The information contained in this document represents the views of the copyright holders as of the date such views are published.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED BY THE COPYRIGHT HOLDERS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE MEMBERS OF THE STRATUSLAB COLLABORATION, INCLUDING THE COPYRIGHT HOLDERS, OR THE EUROPEAN COMMISSION BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE INFORMATION CONTAINED IN THIS DOCUMENT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contributors

Name	Partner	Sections
Marc-Elian Bégin	SixSq	All
Konstantin Skaburskas	SixSq	All
Louise Merifield	SixSq	All
Charles Loomis	LAL/CNRS	All
Eduardo Huedo	UCM	All
Stuart Kenny	TCD	All
Henar Muoz Frutos	TID	All

Document History

Version	Date	Comment
0.1	30 August 2011	Initial version for comment.
0.5	27 September 2011	Final version for comment.
1.0	3 October 2011	Final version.

Contents

List of Figures

1 Executive Summary

This document presents the StratusLab reference architecture for v2.0. This document builds on the document D4.1 - Reference Architecture for StratusLab Toolkit 1.0, focusing on the updated architecture that guides work towards v2.0. The overall goal of producing an open source IaaS distribution that can be easily installed and configured remains central to the development of the v2.0 release.

Compared to v1.0, the v2.0 architecture is rationalized and consolidated. For example, the policy validation, caching and cloud storage are integrated as default features, where they were optional in v1.0. The Appliance Repository is replaced by the Cloud Storage service. Several services are also added or significantly augmented, such as Cloud Storage, virtual network provisioning and inter-cloud connectivity. This is also a client for the Service Manager, such that users can interact with this service remotely and securely.

Performance is also improved through integration of the Caching Service which interacts with Cloud Storage, allowing for a scalable solution with minimal VM start times.

The higher-level architecture is described, focusing on the main layers and of the StratusLab architecture as well as groups or categories of services and components. The main groups are: compute, storage, network, Marketplace (or image management), monitoring and billing and finally inter-cloud connection. This description also identifies the main interfaces and protocols used. Most the the users will interact with StratusLab using standard interfaces, e.g. OCCI, CDMI, OVF, TCloud, or popular access mechanisms such as jClouds. When such standards are not available or not appropriate, then RESTful web services are used, with the exception of OpenNebula (providing the function of VM manager) which is accessed via XML-RPC.

The services and components composing the high-level architecture are shown in a decomposition diagram, with a focus on the main interfaces between these elements, such that the main control and data flows are clear. This diagram shows that all user access is performed via proxies, to harmonize access and protect the underlying services. Another noticeable architectural strategy is the central role the new Collector Service plays in v2.0. This aggregator service collects data from different services and components and also provides this data to any service or component, via simple, secured access. Further, the integration of all data persistence requirements are consolidated into a Persistence Storage Service, based on

iSCSI or NFS.

Each service and component is described in more detail, including the interactions between the elements composing the architecture. Also described for each element is its internal composition, focusing on what is reused, adapted and/or newly developed to deliver the required functionality. The license of each service and component is also provided for each element. All elements in the architecture are released under the Apache 2.0 license, with the exception of Claudia, which is released under the Affero GPL license.

From v1.0 to v2.0, incremental versions of the StratusLab distribution are being built and released to provide an increasing amount of the functionality identified in this document, alongside further improvements to the robustness of the distribution.

The ultimate aim of this document is to provide high-level information regarding the architecture of the StratusLab distribution v2.0, such that readers have a better grasp of how the system is structured, how it is integrated, and from where do the different components and services come.

2 Introduction

This document is a complement to D4.1 - Reference Architecture for StratusLab Toolkit 1.0, where it updates the architecture of the system such that StratusLab can successfully support the 'Hybrid Infrastructure' use case identified in D4.1.

The following chapters from D4.1 are directly applicable to this document:

- Chapter 3: Requirements
- Chapter 4: Development Process and Strategy
- Chapter 7: State of the Art
- Chapter 8: Grid / Cloud Technology Gap

whereas the following are superseded by this document:

- Chapter 5: Component Architecture
- Chapter 6: Selected Components

This document introduces the updated architecture by showing the high-level composition of the system; then each service and component in this high-level view is described in detail. The section with the details serves a double purpose, on one hand providing more clarity as to what was reused and integrated versus newly developed software, and on the other hand providing more information such that users can more easily understand the different components and services of the StratusLab system. This information will be converted to pages on the website to make it more accessible to the StratusLab users and administrators.

From v1.0 to v2.0, incremental versions of the StratusLab distribution are being built and released to provide an increasing amount of the functionality identified in this document, while also further improving the robustness of the distribution.

This document describes a complete and ambitious architecture for an IaaS cloud distribution. Given time and resource constraints, the project will concentrate on the most important elements of this architecture; consequently, some elements may be developed during the evolution of the StratusLab distribution after the project ends. It is a roadmap that guides the project's developments to enable to deliver software with relevant cloud functionality. As we realize this architecture, gather further feedback, and investigate new cloud innovations, we may find

alternative and potentially better ways of accomplishing our goals and may consequently alter the development priorities and architecture.

3 Architecture Overview

The StratusLab reference architecture can be viewed as a layered model, grouping of the main components and services composing the system. Figure 3.1 identifies two main blocks: 'Image Management' and 'IaaS Cloud'. The same figure also identifies the main interactions users have with the system. Since image management is not required to be co-hosted with the IaaS cloud service, these are represented in separated high-level blocks. Figure 3.1 also identifies the main protocols and standards that StratusLab aims to support for v2.0.

A pattern in newer services (e.g. Persistent Disk Manager, Marketplace, Dashboard) is that they favor custom RESTful web service interfaces. The rational for this design decision is to be able to deliver functionality immediately (without depending on future standards) while allowing easy adaptation as we go forward.

New for v2.0, are the persistent disk manager, network manager and a control dashboard. The dashboard is an upgrade of the web monitor service, with the important addition of accounting/billing functionality. In order to deliver this feature however, another service, the collector, is required to transversally collect data from several other services and components. Alongside the collector service is an archiver service, which is responsible for archiving all relevant information for later querying and/or (re)processing.

The network manager provides added features such that users can more dynamically create and configure deployment specific virtual networks, in order to provide finer control and isolation of their system deployed in the cloud. The persistent disk manager provides Amazon EBS-like functionality, such that persistent disk devices can be used in conjunction with running virtual machines, either as the main boot device, or as extra data stores that can be attached to virtual machines.

Another important new component is the 'Inter-Cloud Connector', which is responsible for interfacing a given cloud site with another. At the time of writing, we have not completed the design of this important element. It it therefore left at a high-level, while the exact use cases are developed which will allow the detailed design to be done.

Another way of looking at the reference architecture is to represent the different services and components, identifying the main interactions between them. Figure 3.2 does just that. Here we can see that for v2.0, the 'Collector Service' plays a large role in gathering data from several elements, providing a central location from which to query the state of different services and components in the

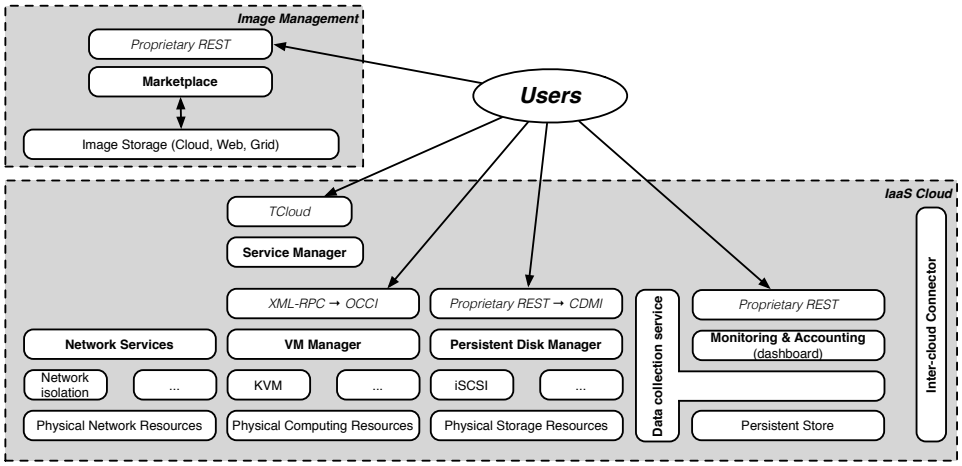

Figure 3.1: *Architecture Overview*

system, from a single interface.

StratusLab v2.0 gives a larger role to the Marketplace to identify virtual machines and disks to instantiate at a given cloud site. In this context, the appliance repository functionality is absorbed in the Cloud Storage Service. Since in the Marketplace each entry is digitally signed including a digital checksum of the virtual image it represents, this approach promotes security and trust, no matter from where the raw files are sourced.

The goal of this tighter integration between the Marketplace and the IaaS layers is to provide a mechanism where trust can be built between the system administrators and users. For this we use cryptographically signed virtual image manifests (or metadata documents) to ensure that we can associate VMs to real users, as well as ensuring that VMs were not modified after creation. This functionality is provided by the Marketplace, together with extended command-line tools. A policy engine and a set of black and white lists related to VM heuristics, integrated at the IaaS VM management level, provide system administrators and users with an automated mechanism to build trust.

Figure 3.2 shows also the facade role that the command-line client plays in abstracting the service interfaces of the system for end-users. The same is true for the dashboard, which abstracts the monitoring and billing services. In v2.0, the Service Manager (Claudia) is also available via a remote command-line client, which will allow users to control and monitor the service from outside the cloud site, as is already available for several other services.

SlipStream has also been added to the overall architecture decomposition. SlipStream is a service providing virtual machine factory functionality and multi-machine deployment orchestration. While this service is not part of the StratusLab distribution, it leverages a significant number of StratusLab services and components. It is a commercial product now available as a SaaS and turn key solution, and part of the StratusLab sustainability and exploitation strategy.

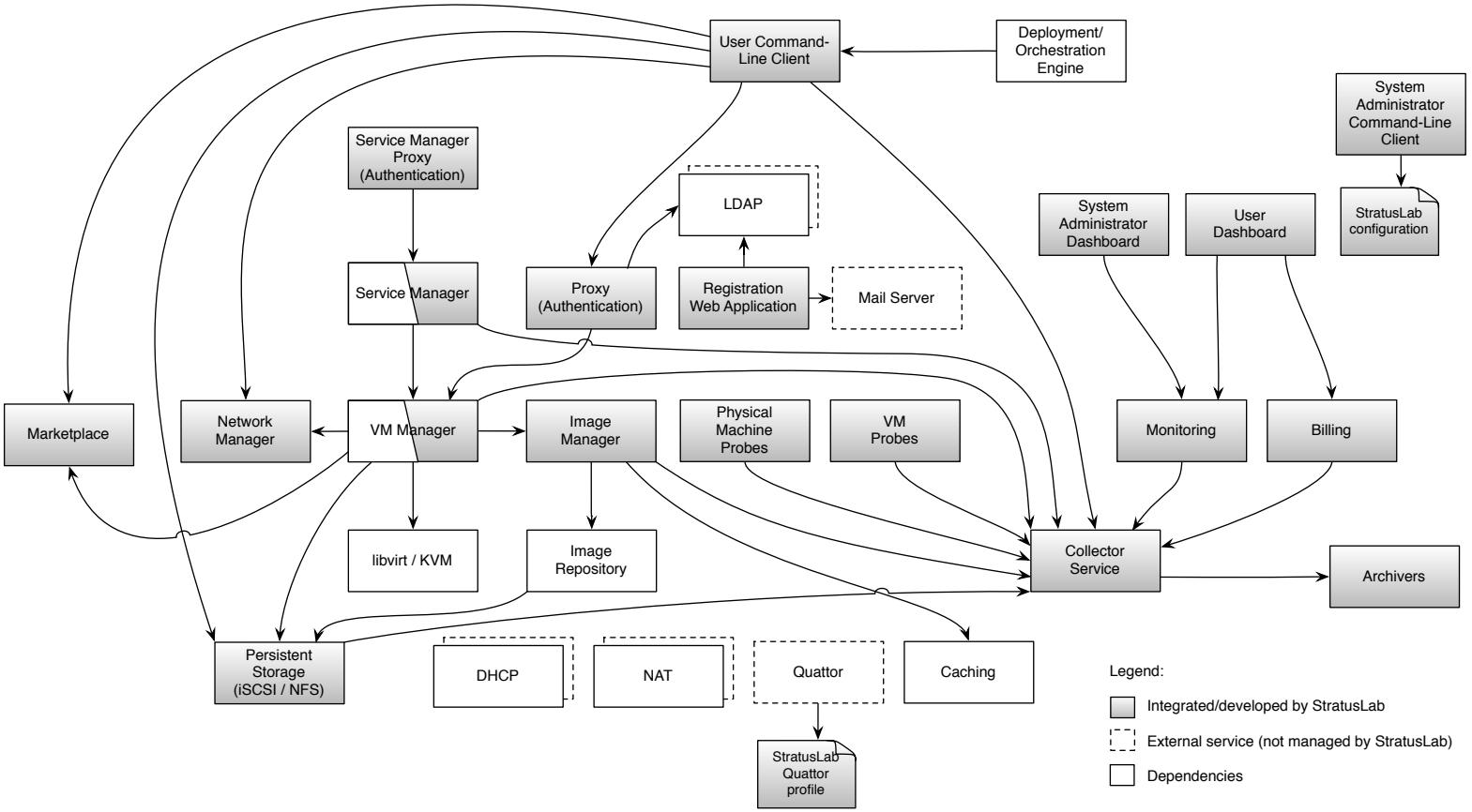

Figure 3.2: *Service and component decomposition*

All of the services and components discussed above are further detailed in the following chapters.

4 VM Manager (OpenNebula) Authentication Proxy

4.1 Overview

The VM Manager (OpenNebula) Authentication Proxy provides a secure, authenticated bridge between external users and the OpenNebula daemon. The service authenticates users based on its configuration, inserts this information into the XML-RPC call to OpenNebula, forwards the modified request to OpenNebula, and then passes the response back to the user.

The service takes advantage of the Java Authentication and Authorization Service (JAAS) implementation with the Jetty web application container, providing a flexible and extensible set of authentication methods. By default, the service is configured to allow authentication with 1) username/password pairs obtained through a simple configuration file or through an LDAP server or 2) digital certificates (e.g. grid credentials in the form of certificates or short-lived proxies).

4.2 Main Interfaces with Other Services/Components

Users access the VM Manager (OpenNebula) service via the authentication proxy. The authentication proxy is accessed programmatically through an XML-RPC interface that mirrors the underlying OpenNebula XML-RPC interface. The only difference is that authentication is handled by the proxy, with this information inserted into the XML-RPC calls passed to the OpenNebula daemon.

The workflow for accessing OpenNebula via the authentication proxy is shown in Figure 4.1. The small core of the service has been developed by the StratusLab project. Standard authentication plugins are used for username/password authentication mechanisms. The plugin supporting grid certificates was created by StratusLab using the cryptographic tools provided by gLite.

The direct software dependencies of the service are listed in Table 4.1. Relevant standards used or supported by the service are shown in Table 4.2.

4.3 License

License	Apache 2
Git repository	stratuslab-authn.git

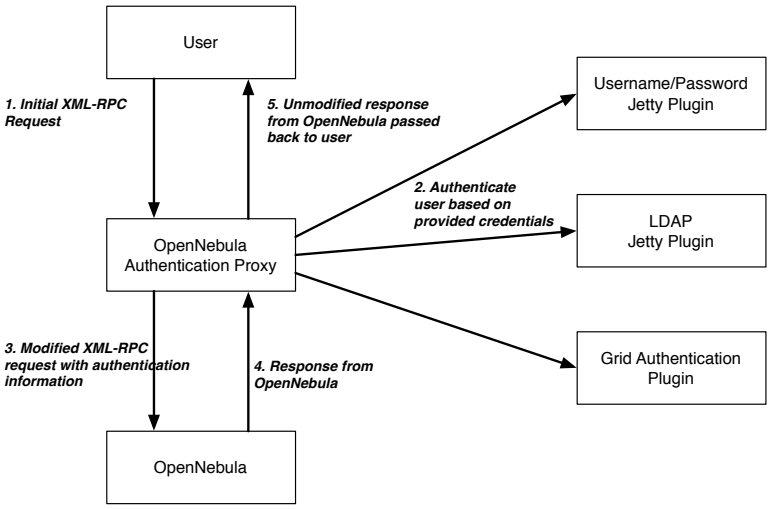

Figure 4.1: *OpenNebula Authentication Proxy Workflow*

Table 4.1: *Dependencies for OpenNebula Authentication Proxy*

Jetty	Java application server
XML-RPC	Apache implementation of XML-RPC protocol

Table 4.2: *Relevant Standards for OpenNebula Authentication Proxy*

JAAS	Java Authentication and Authorization Service
GSI	Grid Security Infrastructure for grid credentials

5 Persistent Disk Service

5.1 Overview

The Persistent Disk Service provides users with persistent storage with a disk-based (or block device) abstraction. The service allows users to create disks, to mount/unmount them on machine instances, and to destroy them. This lifecycle is independent of any particular machine instance, allowing persistent storage of data.

5.2 Main Interfaces with Other Services/Components

The core of the service is a Java-based, RESTful application presenting a proprietary API. Users can access this service via a web browser (HTML representation) or programmatically via the StratusLab command line client (JSON representation).

The service creates user-accessible disks on storage directly attached to the machine running the service. This storage can be accessed via LVM or as a normal file system. LVM-based storage is preferred as this allows faster creation of disks and snapshotting of volumes.

The disks are shared with virtual machines running on the compute hosts either via an iSCSI server or via a shared file system. The iSCSI mechanism is preferred because it provides better performance. Remote access to the defined disks can be provided via HTTP or gsiftp protocols, if they are public and the associated servers are properly configured and running.

This service was entirely developed within the project, although has significant dependencies on other services to provide the complete functionality.

Table 5.1: *Dependencies for Persistent Disk Service*

Jetty	Java application server
Restlet	Java framework for RESTful applications

Table 5.2: *Relevant Standards for Persistent Disk Service*

JAAS	Java Authentication and Authorization Service
LVM	OS-level service for creating block devices
iSCSI	Service for SCSI access to storage over LAN
HTTP Web Server	External access to disks
gsiftp	External access to disks using grid credentials

5.3 License

License	Apache 2
Git repository	stratuslab-storage.git

6 Registration Service

6.1 Overview

The Registration Service provides a web interface that allows new users to register with the infrastructure, or existing users to update their registration information.

The service also provides cloud policy documentation that users must comply with as part of the registration process.

6.2 Main Interfaces with Other Services/Components

Users access this service primarily via a web broswer, although the service could also be accessed programmatically because of the RESTful nature of the interface.

The service keeps all of the user registration information in an LDAP server, running separately from the Registration Service. The StratusLab installation uses the ApacheDS LDAP server implementation, but any other LDAP implementation could also be used.

Interaction with the cloud infrastructure happens through the various authentication proxies for the cloud services, which must be properly configured to access the information in the LDAP server.

6.3 Internal Composition

This service has been entirely developed within the StratusLab project, making use of the software listed in Table 6.1. The service requires access to an external LDAP server to manage the user registration information. The ApacheDS implementation is installed by default, but any other standard LDAP service could be used.

See Table 6.2 for the relevant standards supported or used by the service.

Table 6.1: *Dependencies for Registration Service*

Jetty	Java application server
Restlet	Java framework to simplify development of RESTful applications

Table 6.2: *Relevant Standards for Registration Service*

LDAP	Used to store user information. ApacheDS implementation is used by default.	

6.4 License

License	Apache 2
Git repository	stratuslab-registration.git

7 Network Manager

7.1 Overview

StratusLab v1.0 only supports three networks (public, private and local), which does not provide isolation between virtual machines in the multi-tenancy (multi-machine) cloud environment. This component allows each user (or group) to create, manage and remove sets of isolated LANs (VLAN). Additionally this component allows filtering of virtual machine traffic based on simple rules, e.g. TCP ports.

7.2 Main Interfaces with Other Services/Components

The network manager component interacts with the underlying networking fabric and services to create and manage VLANs. The service is interfaced with the VM management components.

7.3 Internal Composition

This component integrates the work of the Mantychore project into a new service, interfaced with the VM manager.

7.4 License

Apache 2.0.

8 Marketplace

8.1 Overview

StratusLab provides a complete, open-source solution for deploying an "Infrastructure as a Service" cloud infrastructure. Use of the cloud requires the use of prepared machine and disk images. Although StratusLab provides tools to simplify the creation of these images, the procedure for doing so remains a significant hurdle for use of a cloud. Consequently, StratusLab encourages the sharing and reuse of existing images to reduce this barrier.

The Marketplace is at the center of the image handling mechanisms in the StratusLab cloud distribution. It contains metadata about images and serves as a registry for shared images. The Marketplace allows:

1. Users to browse for relevant, pre-existing virtual machine images,
2. System administrators to evaluate virtual machine images against their security policies,
3. Machine creators to publish their work to a larger audience.

8.2 Main Interfaces with Other Services/Components

The core of the service is a Java-based, RESTful application presenting a proprietary API. Users can access this service via a web browser (HTML representation) or programmatically via the StratusLab command line client (XML representation).

8.3 Internal Composition

This service has been entirely developed within the StratusLab project, making use of the software listed in Table 8.1.

See Table 8.2 for the relevant standards supported or used by the service.

8.4 License

License	Apache 2
Git repository	stratuslab-marketplace.git

Table 8.1: *Dependencies for Marketplace Service*

Jetty	Java application server
Restlet	Java framework to simplify development of RESTful applications
Sesame	Java framework for storage and querying of RDF data
Jena	Java framework for building Semantic Web applications
FreeMarker	Java template engine
jQuery	JavaScript library

Table 8.2: *Relevant Standards for Marketplace Service*

RDF/XML	Used to store metadata entries.
SPARQL	Used to query metadata database.

9 Image Manager

9.1 Overview

The Image Manager Service is responsible for resolving Marketplace unique identifiers to physical image files, stored in the Persistent Storage service / Image Repository.

For v2.0, we aim to simplify and streamline the VM submission chain, where the caching, policy validation and image retrieval are integrated into a more coherent and better performing system. The Image Manager provides this functionality, in collaboration with the Image Repository and the Persistent Storage service. This service also absorbs v1.0 services such as the Appliance Repository and several ad-hoc scripts that were integrated with OpenNebula drivers and extension mechanisms.

9.2 Main Interfaces with Other Services/Components

The Image Manager component interacts with the Image Repository, which sits on top of the Persistent Storage Service.

9.3 Internal Composition

This component is developed from scratch, reusing command-line client logic and REST interfaces to other services.

9.4 License

Apache 2.0.

10 VM Manager (OpenNebula)

10.1 Overview

The VM Manager is the core of the system. It orchestrates requests and manages the allocation of resources, as well as the life-cycle of running virtual images. This functionality is provided by OpenNebula, an open-source toolkit for on-premises IaaS cloud computing, offering a comprehensive solution for the management of virtualized data centers to enable private, public and hybrid (cloud bursting) clouds. OpenNebula includes enhancements to address the requirements of the StratusLab project, such as integration with Ganglia, fault tolerance functionality, and virtual network improvements.

10.2 Main Interfaces with Other Services/Components

OpenNebula is a core component of the StratusLab distribution and interfaces with most of the other StratusLab components, mainly: Marketplace, Persistent Disk Service, Authentication Proxy, Network Manager and Service Manager (Claudia). Additionally, OpenNebula interacts with the underlying physical infrastructure (servers, networking fabric and services, storage etc.) to manage virtualized infrastructures.

10.3 Internal Composition

We can distinguish three different areas related to StratusLab in the VM Manager:

- VM Management: It interfaces with physical resource hypervisor, such as KVM, to control (e.g. boot, stop or shutdown) the VMs. It also provides access control to virtual resources and provides fault tolerance.
- Image/Volume Management: Through extended hooks and in collaboration with other services such as the persistent storage and the Marketplace, it transfers the VM images from an image repository to the selected resource and creates on-the-fly temporary images. These also provide caching policies.
- Network Management: Also through hooks, allows each user (or group) to have access to a set of isolated LANs (VLAN) that may be dynamically created. Additionally, this component will allow filtering of virtual machine

Table 10.1: *Relevant Standards for VM Manager*

IEEE 802.1Q	Used for network isolation.

traffic based on simple rules, e.g. TCP ports. These integrate with the Network Manager service.

The components provided for building hybrid or federated clouds are described as part of the Inter-cloud Connector (Chapter 14). Also, OpenNebula provides support for the monitoring and accounting needed in StratusLab.

This component has been extended by the StratusLab project. Specific functionality developed to address StratusLab requirements are:

- VM Management:
  - User groups
  - Access control lists
  - Ganglia information drivers
  - Fault tolerance for virtual machines
  - VM Template repository
- Image/Volume Management:
  - Improved image management
  - Volume management
  - Caching
- Network Management
  - New networking model
  - Dynamic firewalls
  - Network isolation
  - Elastic IPs

See Table 10.1 for the relevant standards supported or used by the VM Manager.

10.4 License

License	Apache 2
Git repository	git://git.opennebula.org/one.git

11 OpenNebula Driver/Extensions

11.1 Overview

StratusLab extends OpenNebula through its driver extensibility mechanism. To this effect, StratusLab ships with a number of upgraded/replaced standard driver and scripts. Here is the list of such extensions:

- Authentication Driver
- Image Downloader
- Quarantine Manager
- Caching Manager
- Policy Validator

Here is an overview of these extensions:

Authentication Driver StratusLab has a unified authentication mechanism, provided by the Authentication Proxy. This means that we require OpenNebula not to perform its normal authentication logic, but trust the proxy. In order for this mechanism to be safe, the StratusLab installation procedure ensures that the OpenNebula daemon cannot be access from outside the machine running it.

Image Downloader StratusLab uses a sophisticated mechanism to allow users to specify which virtual machine they require to be instantiated on the cloud. This feature works in conjunction with the Marketplace service. For this to work, StratusLab also ships with enhanced 'clone' scripts, which provides the logic for accessing the Marketplace to retrieve metadata required to find the physical virtual machine image file, and create a cloned persistent device from which to instantiate the VM.

Quarantine Manager StratusLab supports a quarantine feature, such that system administrators can define a quarantine time during which all terminated machines are kept. This allows them to perform forensic analysis on faulty or abusive virtual machines and to take appropriate actions.

Enhanced Logging To further improve forensic analysis and troubleshooting, improved logging of key events are provided such that, for example, event time correlation can be performed by extracting relevant data from logs.

Table 11.1: *Dependencies for OpenNebula Driver/Extensions*

Ruby	Ruby version supported by OpenNebula
Bash	Standard scripting language

Caching Manager To boost performance, StratusLab supports a caching mechanism, such that the usually large virtual machine image files are not downloaded across the WAN on every request and also are not moved unnecessarily inside the cloud site. By default this is implemented using copy-on-write functionality.

Policy Validator The Marketplace provides rich metadata for each virtual machine and disk image registered with it. This metadata includes digital endorsement and other information on which validation logic can be built. The Policy Validator is integrated as an extension to OpenNebula such that system administrators can have fine control over which virtual machine and disk users are authorized to run on their cloud. This is made possible using a set of black and white lists based on a number of configurable parameters.

11.2 Main Interfaces with Other Services/Components

All of the extensions elements integrate with OpenNebula, but have little or no interaction with OpenNebula.

Authentication Driver None.

Image Downloader Communicates with the Marketplace to retrieve metadata. Once the final endpoint is resolved, it will retrieve (possibly via the caching mechanism) the images and request a clone to be created as a bootable device from the Persistence Service.

Quarantine Manager Persistent storage in which to move the terminated images for a configurable amount of time.

Caching Manager Volatile storage for retrieving cached images.

Policy Validator Communicates with the Marketplace in order to retrieve metadata on which to apply the locally configured policy.

11.3 License

License	Apache 2
Git repository	stratuslab-one.git

12 Collector Service

12.1 Overview

Different types of information are produced across the StratusLab system. For example, information regarding the physical nodes, the virtual machines and the state of the instances from a virtual machine manager.

The Collector Service aggregates the information from these different sources, such that they can be accessed and consumed via a central interface.

To this effect, for example, virtual machine usage information is obtained from OpenNebula information drivers for physical hosts and virtual machines. This information is stored in the OpenNebula database with a proprietary format not necessarily suitable for accounting or billing processing purposes. Therefore, the data is either published or queried in order for the collector service to provide its clients with a uniform REST and/or messaging interface to retrieve the data. This allows high-level services, such as the monitoring and billing service to provide higher value products to the users.

12.2 Main Interfaces with Other Services/Components

The collector is designed as an autonomous component that either periodically extracts information, or better registers with services to be called on important state changes or event triggers.

For example, in the case of OpenNebula, usage information from a running OpenNebula instances can be pushed to the collector service using custom drivers and extensions.

A similar adapter is required for each service from which the collector collects information.

12.3 Internal Composition

This component is developed from scratch making use of existing REST services (e.g. Marketplace, Storage Service) and messaging frameworks (e.g. RabbitMQ or ActiveMQ) and protocols (e.g. Stomp, XMPP, JMS). Other standards such as OGF Usage Record[1] are also used.

[1] More information about Usage Record WG (UR-WG) http://www.ogf.org/gf/group_info/view.php?group=ur-wg.

The different collector modules are developed based on existing interfaces and extensibility mechanisms. For example, the OpenNebula APIs are used.

12.4 License

Apache 2.0.

13 Archivers

13.1 Overview

The archivers are components providing persistence storage for collector service. This can take the form of databases and/or flat files. At the time of writing, this component is still to be better defined and designed.

13.2 Main Interfaces with Other Services/Components

The main provider and consumer of the archivers is the collector service.

13.3 Internal Composition

The archivers are implemented reusing standard database technology, such MySQL, or flat files.

13.4 License

Apache 2.0.

14 Inter-cloud Connector

14.1 Overview

The Inter-cloud Connector allows instantiation of VMs on public clouds (like Amazon EC2 or Flexiscale) as well as partner clouds (other StratusLab clouds). The component will abstract the external clouds' APIs by providing a common interface to them that can be plugged into the VM Manager component allowing a hybrid cloud approach (where the VM manager decides to outsource computation to another cloud) or the Service Manager component with a cloud brokering approach (when the VM is deployed in different cloud providers according to users' policies).

14.2 Main Interfaces with Other Services/Components

The component interfaces with OpenNebula and exposes a public cloud interface to create, manage and monitor VMs on StratusLab sites. The Inter-cloud Connector will be used through OpenNebula, the VM Manager, by means of virtualization and image transfer drivers. These drivers will also interface with Amazon-EC2 compatible clouds.

The component will also interface with Claudia which exposes a TCloud API, considered as an aggregated API, that incorporates different drivers for different Cloud providers (StratusLab-based clouds, Amazon, Flexiscale etc.).

14.3 Internal Composition

For the public cloud interface, a new binding for the jclouds API will be developed. The drivers for StratusLab and Flexiscale clouds are developed from scratch, while the EC2 ones will reuse exiting OpenNebula and Claudia EC2 drivers. These drivers will use the libraries needed to interface each cloud, i.e. the OpenNebula OCA API, the OCCI interface, the Amazon EC2 API or the Flexiscale API.

New functionality for image management will be integrated in order to import disk images from external storage providers, and adapting them so they can benefit from StratusLab's contextualization. For example, an Amazon EBS driver will allow importing Amazon EC2 AMIs into StratusLab sites. And in the other direction, the ability to convert StratusLab images into EC2 compatible images and contextualization will be integrated.

14.4 License

License	Apache 2
Git repository	stratuslab-one.git and stratuslab-claudia.git

15 End-User Command-Line Client

15.1 Overview

The user command-line client is the main user interface for controlling and monitoring StratusLab cloud resources. It allows users to start, stop and query the state of their resources. This remote client also supports a number of authentication mechanisms.

Most StratusLab services provide commands, packaged with this component, such that the user is presented with a uniform mechanism for interacting with the cloud.

The starting point of this component was reused from proprietary code transferred to the StratusLab project from SlipStream by SixSq, with all permissions.

This component is written in portable Python (requires version 2.6 or higher) such that it can be simply installed on a wide range of user machines.

The component was entirely developed within the project, although it has significant dependencies on other services to provide the complete functionality.

A simple configuration file allows the user to configure defaults, which are used by most commands.

15.2 Main Interfaces with Other Services/Components

This component communicates through the proxy services and the Marketplace, using the appropriate protocols, e.g. REST, XML-RPC.

15.3 License

License	Apache 2
Git repository	stratuslab-client.git

Table 15.1: *Dependencies for End-User Command-Line Client*

Python	Python language, version greater or equal to 2.6

Table 15.2: *Relevant Standards for End-User Command-Line Client*

REST	REpresentational State Transfer
XML-RPC	XML Remote Procedure Call

16 System Administrator Command-Line Client

16.1 Overview

The system administration command-line client is one of the two supported mechanisms to configure and install StratusLab.

A single configuration file is used to capture an entire StratusLab installation, from which the system administrator command-line client take configuration information.

16.2 Main Interfaces with Other Services/Components

This component installs and configures all the StratusLab services and components.

16.3 License

License	Apache 2
Git repository	stratuslab-client.git

Table 16.1: *Dependencies for System Administration Command-Line Client*

Python	Python language, version greater or equal to 2.6

Table 16.2: *Relevant Standards for System Administration Command-Line Client*

REST	REpresentational State Transfer
XML-RPC	XML Remote Procedure Call

17 Service Manager (Claudia) Authentication Proxy

17.1 Overview

The Service Manager, provided by Claudia, Authentication Proxy provides a secure, authenticated bridge between external users and Claudia. The service authenticates users based on its configuration, interact with the Claudia API server and then passes the response back to the user.

The service takes advantage of the Java Authentication and Authorization Service (JAAS) implementation with the Jetty web application container, providing a flexible and extensible set of authentication methods. By default, the service is configured to allow authentication with 1) username/password pairs obtained through a simple configuration file or through an LDAP server or 2) grid credentials (certificates or short-lived proxies).

17.2 Main Interfaces with Other Services/Components

In a public infrastructure, users access Claudia via the authentication proxy. The authentication proxy is accessed programmatically through an HTTP interface.

17.3 License

License Apache 2

18 Service Manager (Claudia)

18.1 Overview

The Service Manager is provided by Claudia. The Claudia platform is an advanced service management toolkit that allows service providers to dynamically control the service provisioning and scalability in an IaaS cloud. Claudia manages services as a whole, controlling the configuration of multiple VM components, virtual networks and storage support by optimizing their usage and by dynamically scaling up/down services applying elasticity rules, SLAs and business rules. Claudia can deploy services in a public cloud (Amazon, Flexiscale, GoGrid, etc.) or in a private cloud using a Virtual Infrastructure Manager (such as OpenNebula, Eucalyptus, etc.) through a plug-in driver mechanism that will orchestrate virtual resource allocation.

Claudia has evolved in StratusLab to address the requirements of the StratusLab project, such as the provision and usage of hardware information integration for scalability, scalability and management for grid services, authentication, etc.

18.2 Main Interfaces with Other Services/Components

Claudia, as service manager, can interact with other Cloud providers, like OpenNebula by using an aggregated API, this is the TCloud API. Thus, Claudia can deploy networks or virtual machines. Moreover, other components can interact with Claudia by also using TCloud. Monitoring systems provide monitoring events to Claudia for scalability.

18.3 Internal Composition

The main component in Claudia is Clotho, which includes the Service Lifecycle Manager (which handles service application instantiation and management of services) and the Scalability and Optimization Module (responsible for managing monitoring events and scalability rules). In order to interact with cloud providers, there is an aggregated API implementation, which provides the translations to the cloud providers' APIs. This is an implementation of the TCloud API, aimed to serve as an interface for cloud systems based on Claudia's Service Lifecycle Manager. This application is modular, so it can be tailored to work with any other provider of TCloud services, by creating and using a customized driver.

Table 18.1: *Relevant Standards for Service Manager*

TCloud	Cloud service API for Claudia component
OVF	Service Manifest Format

This component has been extended by the StratusLab project. Specific functionality developed to address StratusLab requirements are:

- Scalability:
  - New policies for scaling according to grid service requirements
  - Expand/Contract a grid site
  - Deployment of a grid site (including grid services, certificates, scripts)
  - Scalability considering VM metrics
- Monitoring:
  - Monitoring at service layer
  - Claudia subscription to services KPI values
- Cloud brokering
  - New Cloud providers drivers
  - Placement Decision Module

See Table 18.1 for the relevant standards supported or used by the Service Manager.

18.4 License

License	Affero GPL
Git repository	stratuslab-claudia.git

19 Claudia Administrator and End-User Command Line Client

19.1 Overview

This component involves a set of tools for administrator and end-user for i) installing and configuring Claudia and ii) interacting with Claudia for deploying, undeploying services.

19.2 Main Interfaces with Other Services/Components

This is a python application, developed from scratch in StratusLab, which it is integrated with the the System Administrator Command Line Client (see chapter 16) for manual installation. It also interacts with the TCloud client for invoking Claudia API and instantiating services, undeploying and so on.

19.3 License

Apache 2.0.

20 System Administrator Dashboard

20.1 Overview

The system administrator dashboard allows system administrators to monitor their infrastructure, both at the level of the physical machines and the level of the virtual machines. Since this dashboard displays data that should be restricted to only site administrators, this dashboard either requires authentication or can only be access from within the organization's network.

20.2 Main Interfaces with Other Services/Components

The system administrator dashboard interfaces with the monitoring service.

20.3 Internal Composition

The dashboard extends the current web monitor, which is a simple set of CGI scripts implemented in Python. This extended functionality builds on the existing logic of the core of the command line clients (end-user and system administrator tools).

20.4 License

Apache 2.0.

21 User Dashboard

21.1 Overview

The user dashboard allows end-users to monitor their virtual machines, and eventually control them. This dashboard requires authentication such that users only see their own resources (e.g. virtual machines, virtual disks).

21.2 Main Interfaces with Other Services/Components

The user dashboard interfaces with the monitoring service, for monitoring, and eventually with the control services (e.g. VM manager, persistent service, network service).

21.3 Internal Composition

Different options exist for this implementation of this web application.

1. The dashboard can either extends the current web monitor, which is a simple set of CGI scripts implemented in Python. This extended functionality builds on the existing logic of the core of the command-line clients (end-user and system administrator tools).

2. Extend the OpenNebula Sunstone application to cover the other StratusLab services, such as network manager, persistence storage and image management.

3. Simply provide javascript-based monitor that interacts directly with the different services (including monitoring).

21.4 License

Apache 2.0.

22 Probes

22.1 Overview

Monitoring probes are the software, installed in the virtual and/or physical machine, which are responsible for measuring metrics. They are the key component in the data gathering mechanisms of the monitoring system. Monitoring information can be obtained from the virtual infrastructure, physical infrastructure, software metrics or KPIs. There are different monitoring probes specialized for each type of metric, and the data they provide is in different formats and offered though different interfaces. Ultimately, the data collected by the probes are collected and pushed to the collector service.

Physical machine probes are installed on the physical machine hosts. These provide the the system monitoring and state transition information required by other services and components to perform their work.

The VM probes, as opposed to the physical machine probes, collect data at the level of the hypervisor, for each virtual machine deployed on the node.

Software probes gather metric information about the software installed in the virtual machine, and are used to monitor this software. For instance, it is possible to measure the number of transactions in a database or the active sessions in a web server.

Finally, the Key Performance Indicators provide metrics related the service, which indicate the status of the service.

22.2 Main Interfaces with Other Services/Components

The probes push data to the collector service. This data is then used by several other services to perform their business.

22.3 Internal Composition

The physical probes are typically written as Nagios/Ganglia scripts, with dependencies corresponding to the transport and protocol used to push the data to their consumer.

The VM probes are highly dependent on the hypervisor used on the physical nodes. StratusLab standardizes on libvirt and KVM, which reduces complexity and dependencies.

Table 22.1: *Dependencies for Persistent Disk Service*

Nagios	Open source computer system and network monitoring software application
Ganglia	scalable distributed system monitor tool for high-performance computing systems
Collectd	Gathering tool which collects, transfers and stores performance data of computers and network equipment.

The software and KPI probes rely on collectd, as gathering tool. It has a set of plugins to which we can add more.

22.4 License

Probes	License
Physical and VM	Apache 2.0
Software and KPI	Affero GPL

23 Monitoring

23.1 Overview

Every distributed system needs to incorporate monitoring mechanisms in order be able to check the health and performance of the system. This is especially true in cloud services that are governed by Service Level Agreements (SLAs) and so the system needs to be able to constantly check that the level of service adheres to the terms of the agreement. As different types of monitoring information are required (e.g. physical, virtual hardware, software, KPIs), corresponding probes (see Section 22), are available to provide this information. Thus, the monitoring service is a RESTful service providing aggregation of monitoring data for system administrators and end-users consumption.

Monitoring differs from accounting and billing, in that it provides a current, real-time view of the cloud service and its main components and services. In comparison, accounting and billing provides aggregated information, over time, on key items.

23.2 Main Interfaces with Other Services/Components

The monitoring service consumes data from the collector service.

23.3 Internal Composition

The monitoring service is composed of a set of components:

- Data Collector and Aggregator: It gathers monitoring information from the collector service, aggregating information according to the data model and storing it in the database.
- Monitoring database: This is a database containing history metrics. These metrics are organized according to a data model, including aggregated information, such as service layer or virtual data center layer.
- API: It is the REST monitoring API implementation which allows querying monitoring information.

This service will be implemented in the project, although has significant dependencies on other services to provide the complete functionality.

Table 23.1: *Dependencies for Monitoring*

Tomcat	application server
Restlet	Java framework for RESTful applications
MySQL	Monitoring database

23.4 License

License	Affero GPL
Git repository	

24 Billing

24.1 Overview

The billing service provides consolidated information on the consumption of resources on the cloud. This data includes compute, storage and network resources. The end-users can generate their own billing statement, and the service can send via email, on a regular basis, billing statements.

Eventually, the billing service could include SLA violations and/or other by-products of resource utilization.

24.2 Main Interfaces with Other Services/Components

The billing service interfaces with the collector services to retrieve consumption data.

24.3 Internal Composition

Like all the web services in StratusLab, the billing service will expose a RESTful web service. In terms of implementation, we are hopeful to be able to reuse results from other projects, such as VENUS-C, in order to reduce development effort.

24.4 License

Apache 2.0.

25 Conclusion

This document describes a complete and ambitious architecture for an IaaS cloud distribution, building on the services provided as part of the v1.0 release delivered in the summer of 2011. From v1.0 to v2.0, incremental versions of the StratusLab distribution will be built and released, providing an increasing amount of the functionality described in this document as well as changes to existing services that improve their robustness and performance.

Given time and resource constraints, the project will concentrate on the most important elements of this architecture; consequently, some elements may be developed during the evolution of the StratusLab distribution after the project ends. It is a roadmap that guides the project's developments to enable to deliver software with relevant cloud functionality. As we realize this architecture, gather further feedback, and investigate new cloud innovations, we may find alternative and potentially better ways of accomplishing our goals and may consequently alter the development priorities and architecture.