Enhancing Grid Infrastructures with
Virtualization and Cloud Technologies

StratusLab D4.1 Architecture

Deliverable D4.1 (V0.1)
3 September 2010

Abstract

This document presents the initial architecture of the StratusLab Toolkit (or distribution). This architecture defines the foundation that will form the v1.0 of StratusLab. This document is a starting point and is expected to evolve. It will be updated at XX with D4.X.

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

Contributors

Name	Partner	Sections
Marc-Elian Bégin	SixSq	All
Eduardo Huedo	UCM	3.1, 3.2, 3.3, 3.5.1 and 6
Rubén S. Montero	UCM	3.1, 3.2, 3.3, 3.5.1 and 6

Document History

Version	Date	Comment
0.1	16 July 2010	Initial version for comment.
0.2	8 August 2010	Chapters 3 and 6.

Contents

List of Figures

List of Tables

1 Introduction

This document presents the initial architecture of the StratusLab Distribution. This architecture defines the foundation that forms the v1.0 of StratusLab, and the version that will follow over the next few months. This document is a starting point and is expected to evolve. It will be updated at XX with D4.X.

The project continuously follows and tracks trends in the domain of cloud computing, grid computing and the communities that gravitate around these fields. While we need a certain level of consistency in our development work, we also need to remain flexible and agile in order to better leverage the latest relevant development and better support our user communities.

The overall goal of the project is to identify existing technologies, standards and implementations, as well as what is required in order to on the one hand deliver to infrastructure system administrators an easy to deploy, install and configure local cloud system, and on the other hand, provide end-users with the ability to leverage the benefit of using cloud computing in a grid context.

This document reviews the requirements driving the StratusLab project, including identification of potential user categories, user stories (as functional requirements), constraint requirements, performance requirements and security requirements. We then describe the anatomy of a cloud in the Service Decomposition chapter. In that chapter we review the functionality already provided by identified distributions, tools and implementations such as OpenNebula. We also discuss in the Service Decomposition chapter missing elements that have to be provided, or existing systems that have to be configured to implement a given missing function (e.g. using LDAP and Apache2 to provide the functionality of the Appliance Repository in which to store virtual images, or Squid to provide caching of popular virtual images).

The reader of this document also needs to understand the development process used in StratusLab. Although unusual in the context of FP7 projects, agile development processes are gaining in popularity in industry and government development. Faced with a challenging and fast moving eco-system around web and distributed technologies, we need to be able to quickly react to changes without loosing focus nor impacting our performance with constant perturbation on our programme of work. Most project members have also a long history of collaboration. We therefore have decided to apply an agile methodology (REF) called Scrum (REF) to drive the development, integration and release efforts in the project. This allows

us to define small units of work, able to add value to our users, on a quick turn-around time scale. Users can then provide valuable feedback which we can use to improve the project. This process includes the entire chain from collecting user requirements, architecture, design, implementation and testing, but also dissemination of our releases, and active pursuit of new user communities and management of feedback.

In this context, the architecture document is there to provide a high-level overview of the StratusLab, leaving the details of implementation, integration and testing to the technical tasks and experts that perform the work. This iterative process causes the architecture to evolve, change and improve, and is therefore not set in stone. However, it is fundamental that the architecture of the system remains crystal clear in all project members' head, and that any changes to the architecture be discussed and agreed by the Technical Board and disseminated to all project members.

The project wiki is also an important source of up-to-date technical, including architectural, information. Each end of development cycle requires that the data provided on the project wiki be updated, such that we can always refer to accurate information.

2 Requirements

The development of the StratusLab distribution is driven by user requirements. These requirements are or different natures - e.g. functional, constraints, performance, security. Te following sections describe our methodology for defining and handling these requirements.

2.1 Users

We have identified several categories of users (or actors) that will interact with the system. While real users might fulfil several roles at once, it is important to separate and identify the concerns that these different types of users will have in the context of StratusLab.

- **Scientists**: End-users that take advantage of existing machine images to run their scientific analyses.
- **Software Scientists and Engineers**: Scientists and engineers that write and maintain core scientific community software and associated machine images.
- **Community Service Administrators**: Scientists and engineers that are responsible for running community-specific data management and analysis services.
- **System Administrators**: Engineers or technicians that are responsible for running grid and non-grid services in a particular resource center.
- **Hardware Technicians**: Technicians that are responsible for maintaining the hardware and infrastructure at a resource centre.

Adding to this list of external users, we add the StratusLab team members, since we are our first users. This allows us Further, since several project activities consists of engineering tasks to better

2.2 User Stories

The functional requirements are expressed a user stories. User stories are a high-level description of a need the different actors of the system have, and a description of way the system fulfils this need. Most user stories are captured according to the

following template: ”As a ¡actor/user¿ i can ¡action¿ in order to ¡benefit¿. The scope and complexity of each story is such that it can be implemented within a single development iteration (or sprint in the Scrum agile methodology we are using). This guarantees that entire stories can be implemented and delivered, such it can be demonstrated and evaluated by project team members and/or target users.

We don't list all user stories in this document since it would represent a significant effort and would cause duplication. StratusLab uses a collaboration tool called JIRA (from XX), augmented by the GreenHopper plugin which provides JIRA with the added features for agile and Scrum support, including management of user stories. The user stories are assigned to sprints by the technical board, at the start of every sprint, which is every three weeks. At the end of every sprint, completed and accepted user stories are declared 'Done' and closed. Others are either rejected as not complete and rescheduled if still relevant.

2.3 Constraint Requirements

This section describes the constraint requirements that have been identified on the system which must be taken into consideration. The StratusLab initial system administrators community will come from the grid world. This community largely uses Scientific Linux (REF) as the base operating system. While this is likely to remain the case initially, it is important that the StratusLab distribution be multi-operating system. Therefore, right from the start, StratusLab shall support more than one operating systems. The logical candidates are: CentOS (close cousin of Scientific Linux and also Red-Hat based), Ubuntu (a Debian based operating system), but also SUSE (an other RPM based operating system as CentOS).

Ease of installation and configuration is an important tenet of StratusLab. This probably means that StratusLab will have to be opinionated, which means that while customisation and extension shall be possible, assumptions will be made to facilitate the installation and configuration of the system. An important trade-off is required between flexibility and usability, where our user communities will be instrumental in reaching. It therefore shall be possible to install and configure StratusLab using manual tools and automated fabric management tools (e.g. Quattor (REF)).

It is important that StratusLab is viewed as a high quality distribution. Building credibility is an important driver for our dissemination activities. It is also important to realise that StratusLab will largely be composed of existing tools, libraries and implementations, which have to be carefully selected. It is therefore important that each constituent of the distribution be well supported and mature. Ideally members of the StratusLab will be part of or in contact with the developers communities behind these.

We have pledged that StratusLab would be an open source (REF) distribution. While it should be possible to substitute open source compoenents with commercial ones, it is critical that the entire feature set provided by StratusLab be available under open source licenses. Whenever possible, every effort should be taken to se-

lect harmonious licenses. If components only available under commercial license are required, then all project members should be given equal access to the license to the duration of the project.

2.4 Performance Requirements

This section describes the performance requirements that have been identified on the system which must be taken into consideration.

The survey conducted earlier and reported in D2.1 illustrate clearly that system administrators are expecting to be able to handle in the order of tens of thousands of virtual machines over thousands of physical hosts. These numbers are therefore defining clearly what is the expectation in terms of expected performance of the system.

Another important performance parameter is the need for the system on which StratusLab is deployed to remain elastic, from a user perspective. While there are physical limitations that limit the elasticity of any system, it is important that the system behaves elastic, which means a reasonable response time to all requests, without exposing to the user the semantic of queues.

Cloud computing is based on virtualization and virtualization means virtual images, which can be large files. We are therefore, faced with manipulating larges files, over potentially wide area networks. Since these images are largely 'write once and read many', a caching strategy is appropriate to avoid having to transfer several times the same files over the network.

2.5 Security Requirements

While the grid doesn't have as stringent security requirements as some commercial and military applications, it is important that the system provide a reasonable level of security.

It is important that the system administrators trust that running virtual machines on their infrastructure is not a threat to that same infrastructure. Similarly, end-users deploying virtual machines on a remote infrastructure via the StratusLab cloud API must also feel certain that the execution of their machine will not be tempered with and that data used in that context can also remain secure.

Virtualisation technologies by there very nature already provide a powerful level of separation between the physical hosts, under the control of the system administrators, and the running virtual machine instances, manages by the end-users. However, virtualisation technologies are complex, it is therefore important that their installation and configuration is performed such that it doesn't leave back doors open to malicious exploitation.

An important aspect of security is establishing a trust relationship between the system administrators and the end-users, such that a foreign virtual machine can be deployed on a remote infrastructure. For the cloud model to work in supporting the grid, it is fundamental that this trust relationship can be established and negotiated without the constant required intervention of humans. It is therefore required that

virtual machines be signed in some way such that the pedigree of the machine, it's provenance and composition be tracked back to a trusted user.

Another important aspect to security that requires our attention is regarding credential management. All the services integrated in StratusLab shall integrate with a centralised and shared secure credential server, for example LDAP. This means that the same credentials can be used to authenticate and authorise access to the different StratusLab integrated services.

3 Service Decomposition

In general, a IaaS cloud consists of several main components, namely:

- **virtualization layer**: on top of the physical resources including network, storage and compute
- **virtual infrastructure manager (VIM)** that control and monitor the VMs over the distributed set of physical resources
- **appliance repository** as the source of VMs and container for newly created
- **cloud interface** that provides the users with a simple abstraction to manage VMs.
- **credentials manager** that provide credentials authentication and authorisation services to the StratusLab integrated services.

In the last years, a constellation of technologies that provide one or more of these components have emerged. So, a variety of hypervisors have been developed and greatly improved, most notably KVM, Xen and VMWare. Also several VIM technologies that cover the functionality outlined above have appeared, like Platform VM Orchestrator, VMware DRS, or Ovirt. On the other hand, projects like Globus Nimbus [9], Eucalyptus [8] or OpenNebula [6] (developed by UCM in the context of the RESERVOIR project), or products like VMware vSphere, that can be termed cloud toolkits, can be used to transform existing infrastructure into an IaaS cloud with cloud-like interfaces.

While the virtualization eco-system is now rich and reasonably mature, a gap still exists in order integrate a single IaaS cloud distribution, accessible from outside the infrastructure, over a clear service API. This is critical since, for example in a grid context, user cannot be granted direct remote access to the machines running the virtualisation management engine (OpenNebula in the case of StratusLab).

The contextualisation of virtual machines is also an important aspect in which we need to fill a gap. This is important in order to be able to reuse virtual images between sites running StratusLab. Contextualisation currently lacks standardisation. While this is an aspect where StratusLab will engage the standardization organisations over the project lifetime, we currently need to define conventions

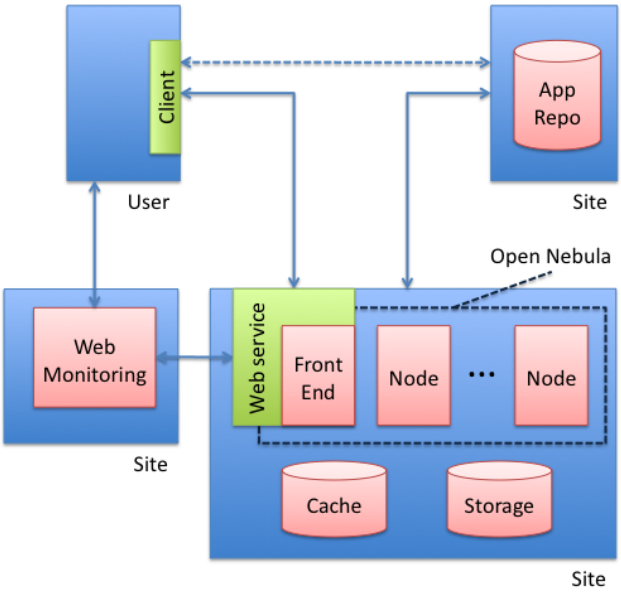

Figure 3.1: *High-level architecture.*

that will guarantee virtual machine interoperability between sites running the StratusLab distribution in the first place, and eventually develop solutions for letting users and sites utilise other cloud services beyond the StratusLab frontier.

The following image shows a high-level view of the composition of the envisaged StratusLab cloud deployment.

The heavy-lifting in terms of virtual machine management is provided by OpenNebula. The Appliance Repository is based on a simple Apache Web Server (i.e. HTTPD), but could also be implemented

Key differentiating factors of OpenNebula with other commercial virtual infrastructure managers are its open and flexible architecture and interfaces, which enable its integration with any existing product and service in the Cloud and virtualization ecosystem, and its support for building any type of Cloud deployment. On the other hand, compared to other open-source alternatives, OpenNebula provides superior functionality on a wider range of virtualization technologies for building private and hybrid clouds.

Therefore, OpenNebula has been chosen as the VIM of the StratusLab cloud toolkit. OpenNebula is an open-source toolkit to easily build any type of cloud: private, public and hybrid. It has been designed to be integrated with any networking and storage solution and so to fit into any existing data center.

OpenNebula manages VMs and performs life-cycle actions by orchestrating three different management areas, namely: **networking** by dynamically creating

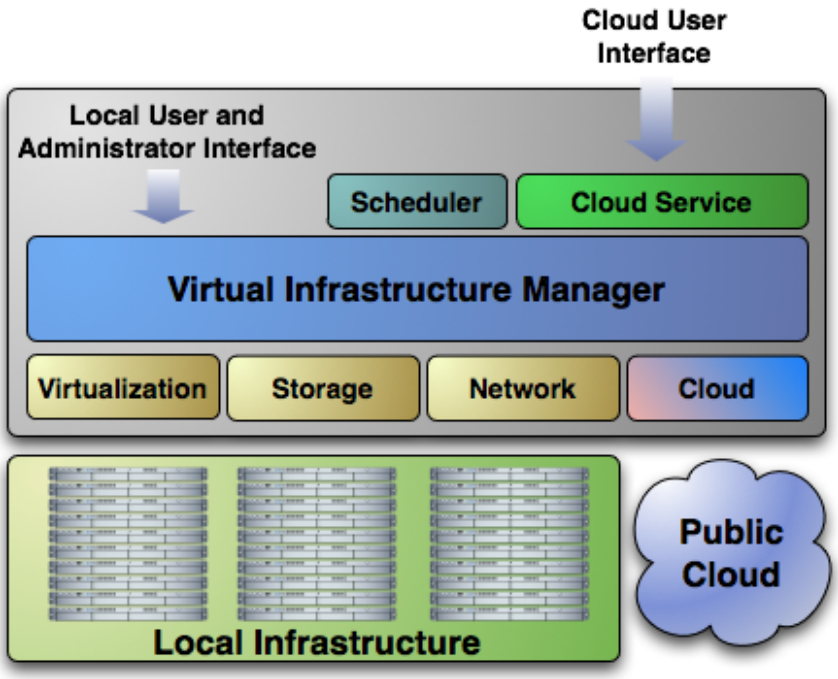

Figure 3.2: *OpenNebula architecture.*

local area networks (LAN) to interconnect the VMs and tracking the MAC addresses leased in each network; **image management** (storage) by transferring the VM images from an image repository to the selected resource and by creating on-the-fly temporary images; and **virtualization** by interfacing with physical resource hypervisor, such as XEN or KVM, to control (e.g. boot, stop or shutdown) the VMs. Moreover, it is able to contact cloud providers to combine local and remote resources according to allocation policies. Figure 3.2 shows the architecture of OpenNebula. Phase I of StratusLab focus on the virtualization of grid services, phase II will focus on the use of novel resource provisioning methods using cloud interfaces.

OpenNebula uses a set of managers to orchestrate the management of VMs. In turn, these managers are helped by a set of pluggable modules that decouple the managing process from the underlying technology, such as virtualization hypervisors, operating systems, file transfer mechanisms or information services. These modules are called drivers in the OpenNebula jargon, and they communicate with the OpenNebula core using a simple ASCII protocol; this simplifies the development of new drivers.

3.1 Computing

A VM within the OpenNebula system consists of:

- Capacity in terms memory and CPU.
- A set of NICs attached to one or more virtual networks (see Section 3.2).
- A set of disk images. In general, it could be necessary to transfer some of these image files to/from the execution host (see Section 3.3).
- A state file (optional) or recovery file, that contains the memory image of a running VM plus some hypervisor specific information.

The above items, plus some additional VM attributes like the OS kernel and context information to be used inside the VM, are specified in a VM template file. OpenNebula manages VMs by interfacing with the physical resource virtualization technology (e.g. Xen or KVM).

The scheduler module is in charge of the assignment between pending VMs and known hosts. The OpenNebula scheduling framework is designed in a generic way, so it is highly modifiable and can be easily replaced by third-party developments. This way, it can be used to develop virtual resource placement heuristics to optimize different infrastructure metrics (e.g. utilization or energy consumption) and to fulfill grid service constraints (e.g. affinity of related virtual resources or SLA).

Whenever the VM request enters OpenNebula, it is placed in PENDING state. In this state, the VM is eligible for placement by the scheduler. At a user request, the machine can be kept on hold, and won't be deployed until released. The VM enters then the HOLD state, going back to PENDING upon release. As soon as the scheduler finds a suitable physical resource for the VM, OpenNebula invokes its Virtual Machine Manager which, aided by a pluggable driver, writes the deployment descriptor suitable for the hypervisor running in the chosen physical resource. Afterwards,the same manager, aided by the same driver, starts the VM invoking the hypervisor with the deployment descriptor. At this point, the VM enters the ACTIVE state.

In order to comply with certain policies, the scheduler may decide to migrate the VM to another physical host (this is done again by the Virtual Machine Manager component). During this migration, the VM stays in the ACTIVE state.

The user or the scheduler might decide to stop the VM, putting it in the STOPPED state. Upon restoring, the VM will fall back to the PENDING state. A similar procedure is followed whenever a VM suspension is triggered, although the SUSPENDED state is different because it returns to the ACTIVE state whenever it is decided to get restored. At the end of the life cycle, if all the process went fine, the VM will end in the DONE state, after a shutdown or a cancellation. Otherwise, if for whatever reason there is a failure in the process, the VM will be placed in the FAILED state.

The whole state diagram is shown on Figure 3.3

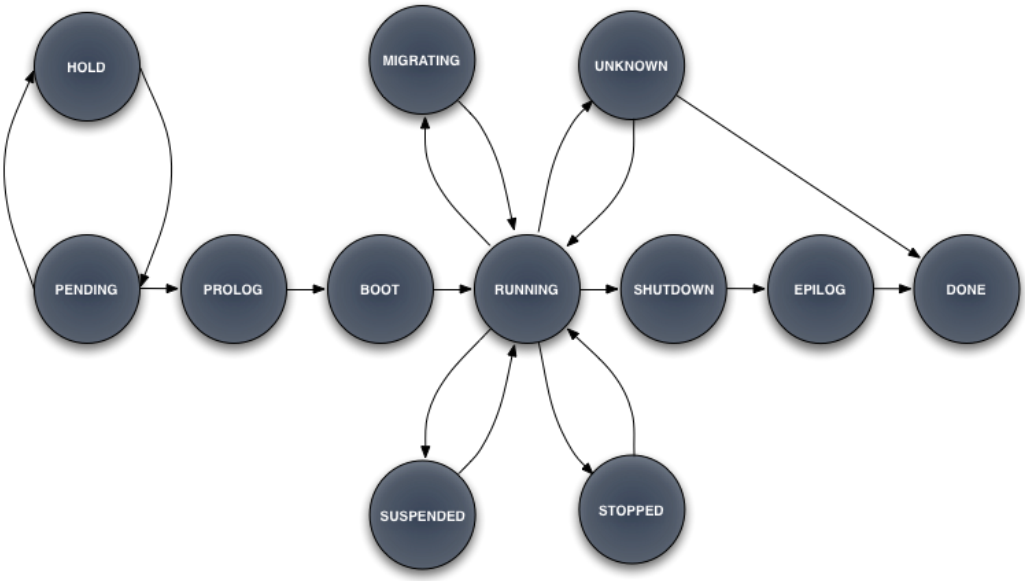

Figure 3.3: *VM state diagram.*

3.2 Networking

VMs are the basic building blocks used to deliver IT services of any nature, from a computing cluster to the classical three-tier business application. In general, these services consists of several interrelated VMs, with a Virtual Application Network (VAN) being the primary link between them. OpenNebula dynamically creates these VANs and tracks the MAC addresses leased in the network to the service VMs. Other TCP/IP services such as DNS, NIS or NFS, are the responsibility of the service (i.e. the service VMs have to be configured to provide such services).

The physical hosts that will conform the fabric of the virtual infrastructures will need to have some constraints in order to effectively deliver virtual networks to the VMs. Therefore, from the point of view of networking, we can define our physical cluster as a set of hosts with one or more network interfaces, each of them connected to a different physical network.

Figure 3.4 shows two physical hosts with two network interfaces each, thus there are two different physical networks. There is one physical network that connects the two hosts using a switch, and another one that gives the hosts access to the public internet. This is one possible configuration for the physical cluster, and it is the recommended one since it can be used to make both private and public VANs for the VMs. Moving up to the virtualization layer, we can distinguish three different VANs. One is mapped on top of the public internet network, and a couple of VMs take advantage of it. Therefore, these two VMs will have access to the internet. The other two (red and blue) are mapped on top of the private physical

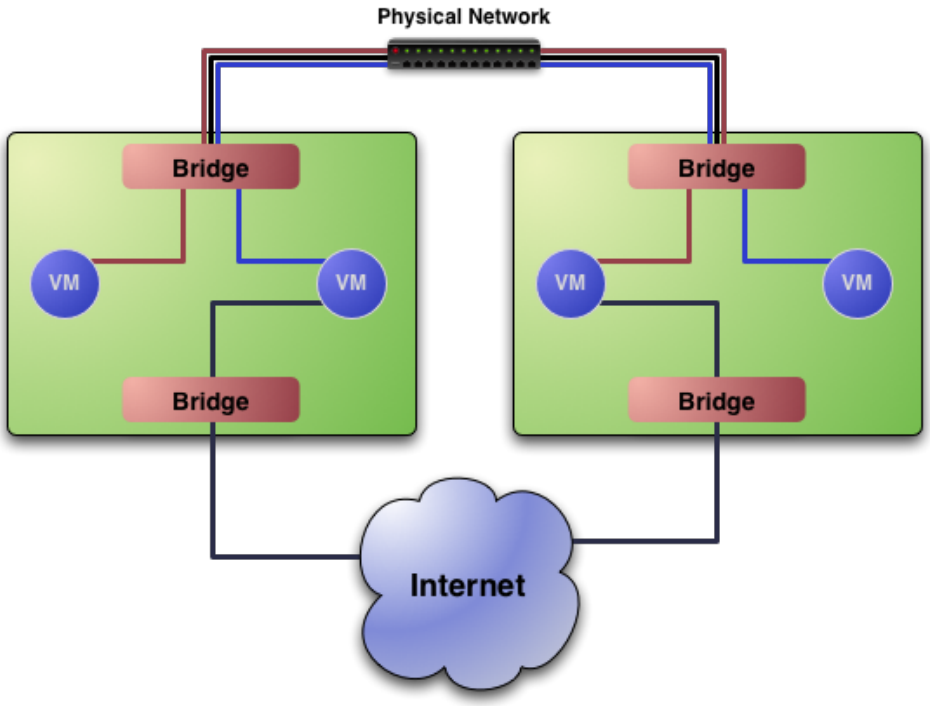

Figure 3.4: *Networking model.*

network. VMs connected to the same private VAN will be able to communicate with each other, otherwise they will be isolated and won't be able to communicate.

3.2.1 Virtual Network

OpenNebula allows for the creation of Virtual Networks by mapping them on top of the physical ones. All Virtual Networks are going to share a default value for the MAC prefix, set in the oned.conf file.

There are two types of Virtual Networks in OpenNebula:

- Fixed, which consists of a set of IP addresses and associated MACs, defined in a text file.
- Ranged, which allows for a definition supported by a base network address and a size, either as a number or as a network class (B or C).

Other approaches, like configuring a DHCP server for the datacenter, are also possible.

3.2.2 IP Address Assignation

When virtual...

3.2.3 Firewall Configuration

This section describes the performance requirements that have been identified on the system which must be taken into consideration.

3.3 Storage

VMs are supported by a set of virtual disks or images, which contains the OS and any other additional software needed by the service. There will be an image repository (see Section 3.4) that holds the base image of the VMs. Also, the images can be shared through NFS between all the hosts, or transferred through SSH between them.

OpenNebula uses the following concepts for its image management model:

- Image Repositories, refer to any storage medium, local or remote, that hold the base images of the VMs. An image repository can be a dedicated file server or a remote URL from an appliance provider, but they need to be accessible from the OpenNebula front-end. See Section 3.4 for more details about the image repository to be used in StratusLab.
- Virtual Machine Directory, is a directory on the cluster node where a VM is running. This directory holds all deployment files for the hypervisor to boot the machine, checkpoints, and images being used or saved?all of them specific to that VM. This directory should be shared for most hypervisors to be able to perform live migrations.

Any given VM image goes through the following steps along its life cycle:

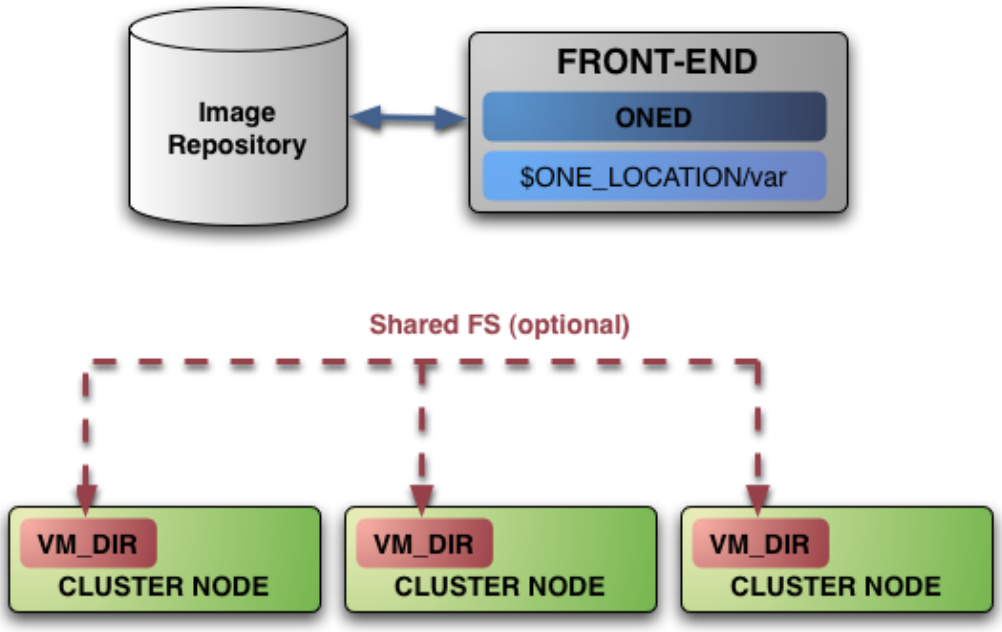

Figure 3.5: *Image management.*

- Preparation implies all the necessary changes to be made to the machine's image so it is prepared to offer the service to which it is intended. OpenNebula assumes that the image(s) that conform to a particular VM are prepared and placed in the accessible image repository.
- Cloning the image means taking the image from the repository and placing it in the VM's directory in the physical node where it is going to be run before the VM is actually booted. If a VM image is to be cloned, the original image is not going to be used, and thus a copy will be used. There is a qualifier (`clone`) for the images that can mark them as targeting for cloning or not.
- Save or Remove. If the save qualifier is disabled, once the VM has been shutdown the images and all the changes thereof are going to be disposed of. However, if the save qualifier is activated, the image will be saved for later use.

3.3.1 Persistent Storage

It is important to be able to persist data independently from running virtual images. Amazon has a feature called Elastic Block Store, which allows a cloud user to define a persistent volume, attached to the running instance via a device and mounted on the local file system. Even if the instance stops or crashes, the data behind this volume is persisted and can be re-attached to a different running instance. This

feature also allow cloud users to be able to create complex data-sets and manage them independently from the machine images. For example, a data-set might be maintained for testing, which can be applied to different versions of a system and composed at runtime, without being 'baked' into each image needing this data-set. This feature can be achieved in StratusLab using the current contextualisation feature of OpenNebula, where this disk image is saved and made available to the user via the appliance repository. If true persistence is required (data surviving beyond a physical node crash) the disk image could be associated with a distributed or replicated file system. Performance and network consideration have to be take into account for this feature. Further, other options also exists, which we will explored.

3.3.2 Caching

...

3.4 Appliance Repository

The StratusLab Applicance Repository is leveraging a simple web enabled file system. The current version of the Appliance Repository is implemented using Apache2 (httpd), configured to authenticate with the project LDAP server. Here's an example of the structure of the repository, following the Maven structure convention: `https://appliances.stratuslab.org/images/base/ubuntu-10.04-i686-bas` In this directory, we find the manifest file (`ubuntu-10.04-i686-base-1.0.img.manifest.xml`) and the image (`ubuntu-10.04-i686-base-1.0.img.gz`). The image file should be compressed as a single file (i.e. not archived).

The currently proposed manifest format is an XML document. Here's an example of a manifest file:

```
<manifest>
    <created>2010-08-18 20:34:28.334763</created>
    <type>base</type>
    <version>1.0</version>
    <arch>i686</arch>
    <user>A. Joseph</user>
    <os>ubuntu</os>
    <osversion>10.04</osversion>
<compression>gz</compression></manifest>
```

As we explore features like the Persistent Storage (3.3.1), we might then extend the definition of 'Appliance', which could mean to also add data images to the appliance repository.

3.5 Service Image Creation Engine

Since in cloud the fundamental building block for users is the virtual machine, we need to facilitate the creation of virtual machine images, which will work with our

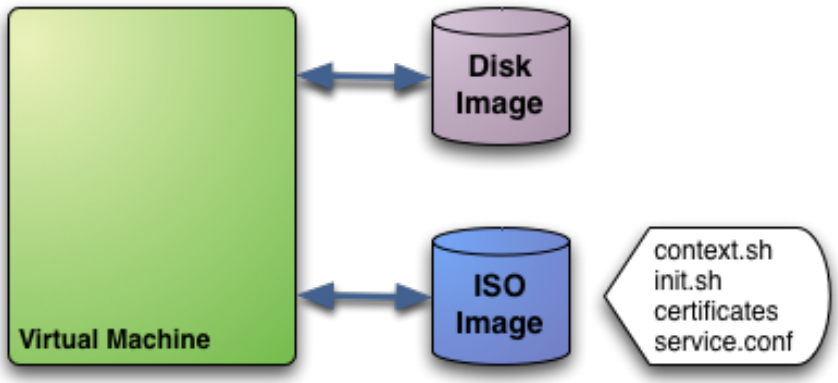

Figure 3.6: *VM Contextualisation.*

contextualisation. This section describes several aspects of this creation and how can StratusLab provide support to users for image creation.

3.5.1 Virtual Machine Contextualisation

In general, a VM consists of one or more disk images, which contain the operating system and any additional software or data required. When a new node is needed, the images are transferred (cloned) to a suitable physical resource and a new VM is booted. During the boot process the VM is contextualized, i.e. the base image is specialized to work in a given environment by, for example, setting up the network, the machine hostname or registering a new worker node in a cluster service (e.g. job queuing system). Different techniques are available to contextualize a VM, for example, using an automatic installation system (like Quattor [1], or a context server [4], or accessing a disk image with the context data for the VM (Open Virtual Format, OVF [3], recommendation).

OpenNebula uses an ISO image (OVF recommendation) to give configuration parameters to a newly started VM. This method is network agnostic so it can be used also to configure network interfaces. In the VM description file, the user can specify the contents of the ISO file (files and directories), tell the device the ISO image will be accessible and specify the configuration parameters that will be written to a file for later use inside the VM.

Figure 3.6 shows an example with a VM and two associated disks. The Disk Image holds the filesystem where the Operating System will run from. The ISO image has the contextualization for that VM, including the following files:

- `context.sh`: file that contains configuration variables, filled by OpenNebula with the parameters specified in the VM description file.
- `init.sh`: script called by VM at start that will configure specific services for this VM instance.

- `certificates`: directory that contains certificates for some service.
- `service.conf`: service configuration.

This is just an example of what a contextualization image may look like. Only `context.sh` is included by default. The user has to specify the values that will be written inside `context.sh` and the files that will be included in the image.

In VM description file the user can tell OpenNebula to create a contextualization image and to fill it with values using the `CONTEXT` parameter. For example:

```
CONTEXT = [
  hostname   = "$NAME",
  ip_private = "$NIC[IP, NETWORK=\"Private LAN\"]",
  ip_gen     = "10.0.0.$VM_ID",
  files      = "/service/init.sh /service/certificates /service/service.c
  target     = "sdc"
]
```

Variables inside the `CONTEXT` section will be added to `context.sh` file inside the contextualization image. The variables starting with $ are substituted by the values that this specific VM instance has (you can check the values with `onevm show`). In this case `$NAME` gets its value from the NAME specified in the VM description file. `$NIC[IP, NETWORK="Private LAN"]` will get the IP assigned to the interface that associated to `Private LAN` network. The file generated will be something like this:

```
# Context variables generated by OpenNebula
hostname="somename"
ip_private="192.168.0.5"
ip_gen="10.0.0.85"
files="/service/init.sh /service/certificates /service/service.conf"
target="sdc"
```

Some of the variables have special meanings: the `files` attribute contains the files and directories that will be included in the contextualization image, and the `target` attribute contains the device where the contextualization image will be available to the VM instance (note that the proper device mapping may depend on the guest OS, e.g. ubuntu VMs should use `hd*` as the target device).

The VM should be prepared to use the contextualization image. First of all it needs to mount the contextualization image somewhere at boot time. Also a script that executes after boot will be useful to make use of the information provided.

The file `context.sh` is compatible with bash syntax, so users can easily source it inside a shell script to get the variables that it contains.

3.5.2 Base Images

...

3.5.3 Image Creation Service

...

3.6 User Management

All services used by StratusLab must be integrated with a single identity system. OpenNebula is already been extended to support LDAP. It is important to ensure that all services use such strategy to limit the propagation of specific credential mechanism across the distribution. Further, the grid uses a digital certificate mechanism to ensure proper integration of authentication and authorisation across its services and sites. StratusLab will need to integrate this certificate-based mechanism with its own mechanism.

3.7 Accounting

Both cloud and grid services require adequate tracking of resource usage, per user. While OpenNebula maintains several key accounting metrics, they are not readily available (see 3.8) to accounting services. In a grid context, it is important to be able to extract these values and aggregate. It is therefore important to make the raw accounting parameters available via a convenient API, such that they can be securely queried and retrived.

3.8 Extensibility

As the StratusLab experience with OpenNebula grows, we are also gaining experience and confidence in its capability. Several members of the StratusLab project are now contributors on the OpenNebula project. The latest development on the Web Monitor, which add monitoring capability of an OpenNebula installation over the web, showed the power of the XMLRPC API exposed by OpenNebula. Being able to extend more easily the data available via that interface will improve the rate at which we can extend and better integrate OpenNebula, StratusLab's core virtualization management engine, with existing and new services. Other extensibility mechanisms are likely to be identified over the course of the project, which we hope we will be able to exploit in collaboration with the main developers of these systems.

4 Build, Packaging, Deployment, Installation, Configuration and Testing Strategy

StratusLab is composed of several components, integrated into a coherent whole. However, it is important for the success of the distribution, and hence the project, that StratusLab be seen, from an deployment, installation, configuration and testing point-of-view as a well integrated system. For this reason, we are releasing StratusLab as a set of packages, including well identified dependencies, in the native format of the different operating system distribution that we support. It is also important that StratusLab be well separated from the grid services and other applications that will be used on top of it in a grid site. Yet, we want to be sure that it is easy to deploy the system such that it integrates well with these know grid services. We therefore need to strike the right balance between a generic cloud setup and facilitating the life of grid site system administrator, such that the specific needs of grid services are well catered for by the under-lying cloud layer.

4.1 Reference Deployment Model

The following logical components will be deployed as part of a standard site running StratusLab:

- **Front-End**: OpenNebula Virtual Machine Manager
- **Nodes**: on which VM are deployed by the VM Manager, including the necessary virtualisation support software (e.g. kvm, xen, vmware)
- **Monitoring**: Web Application providing monitoring service
- **Accounting**: Accounting service available over a web application and/or service.
- **Caching**: HTTP caching service providing cache of popular requests, especially interesting for large files such as VMs.

While part of the StratusLab distribution, other services in a grid context will be treated as singleton (i.e. only one instance of the service will be deployed and remotely accessed by all other sites). There services are:

- **Appliance Repository**: Store for trusted appliances (i.e. virtual machine images)
- **Credential Manager**: Service providing credentials management

This setup will be expected when deploying a StratusLab installation. OpenNebula and virtualisation libraries and tools provide a very wide range and rich set of parameter settings. In order to reduce the range of these parameters exposed to the system administrator, we are proposing to expose only a subset of parameters, taking into account reasonable assumptions on the setup.

4.2 Installation

The preferred installation strategy for StratusLab will be from packages. We will support RPM and DEB packages, popular with Red-Hat, Debian and SUSE-like operating system distribution.

All required dependencies for StratusLab will be expressed as dependencies in the StratusLab packages. Further, all dependencies will be on packages, either pointing to official and supported packages repository (preferred option), or in the case were these dependencies are not available in package format, or if the wrong versions are available, we will package and maintain these package ourselves. All the StratusLab packages will be available in our dedicated package repository (i.e. YUM and APT repositories).

The installation itself will be available via two mechanisms:

- **Automated**: Using the Quattor fabric management system, already the defact standard for several grid sites
- **Manual**: Using dedicated `stratus-*` system administrator commands

The source of metadata for both systems will be shared and maintained together with each release of StratusLab

4.3 Configuration

The configuration of the StratusLab system will be performed via a single configuration file, including parameters grouped in sections. No duplication will exist in the configuration file, reducing the chances of conflicts, resulting in a faulty system. Further, since the certain parameter, for example the choice of the hypervisor or share strategy (e.g. NFS, SSH), could result in installing different packages, some installation technique could decide to make those choices up-front, reducing therefore flexibility for the system administrator but also reducing complexity in installing the system.

4.4 Testing

Testing the StratusLab distribution is required to ensure that the advertised functionality works, and as we release new features and fix bugs, that we do not intro-

duce regressions. The best way to ensure that each release of StratusLab is of the expected quality and that no regression is introduced is to have a test-suite, built over each sprint, which covers the main features. In order to execute this test-suite often, such that mistake committed in our version control system, or released in a new version of the dependencies we integrate in StratusLab, is to have the test-suite execution automated. However, a challenge in testing distributed systems, and worst a cloud system designed to deploy distributed systems, is the setup required for the test-suite to perform on a meaningful deployment of the system.

StratusLab already has a series of tasks automated using a continuous integration server called Hudson (REF). This server fires build, deployment and test tasks (called jobs in Hudson) soon after new or updated files are committed in our version control system.

Further, once more complex deployment scenario are required, SlipStream will be used to deploy complex multi-node systems automatically, without having to write special code for this. The execution of the SlipStream deployment scenario will also be integrated in Hudson jobs, which in turn will be triggered by commits in our version control system.

5 Grid Service Support

This document is a simple example for producing StratusLab reports. It should be used for formal deliverables and milestones that require a written report. It should also be used for general reports intended for the wider scientific community or general public.

6 Cloud Interoperability

StratusLab sites will expose elastic behaviour exploiting the underlying capabilities of the StratusLab cloud in order to adapt to peak loads observed from various grid applications. The infrastructure will also be able to utilize external resources residing outside the borders of the participating institutes. Such resources will be for instance large public cloud providers, like Amazon's EC2 service, therefore demonstrating that the EGI infrastructure can operate in a hybrid public–private cloud platform, able to tap on external cloud resources when peak demand requires it.

A hybrid cloud is an extension of a private cloud to combine local resources with resources from remote cloud providers. The remote provider could be a commercial cloud service, such as Amazon EC2 or ElasticHosts, or a partner infrastructure running the StratusLab cloud distribution. Such support for *cloudbursting* enables highly scalable hosting environments.

An hybrid cloud deployment powered by OpenNebula is fully transparent to infrastructure users. Users continue using the same private and public cloud interfaces, so the federation is not performed at service or application level, but at infrastructure level. It is the infrastructure administrator who takes decisions about the scale out of the infrastructure according to infrastructure or business policies.

However, the simultaneous use of different providers to deploy a virtualized service spanning different clouds involves several challenges, related to the lack of a cloud interface standard; the distribution and management of the service master images; and the inter-connection links between the service components. The following sections briefly analyze these issues.

6.1 Cloud interfaces

Currently, there is no standard way to interface with a cloud, and each provider exposes its own APIs. Moreover the semantics of these interfaces are adapted to the particular services that each provider offers to its costumer (e.g. firewall services, additional storage or specific binding to a custom IP).

A traditional technique to achieve interoperation in this scenario is the use of adapters [5]. The pluggable and modular architecture exhibited by OpenNebula, makes easy the integration with specialized adapters to interoperate with different hypervisor technologies (Xen, KVM, etc.), or different cloud providers, like

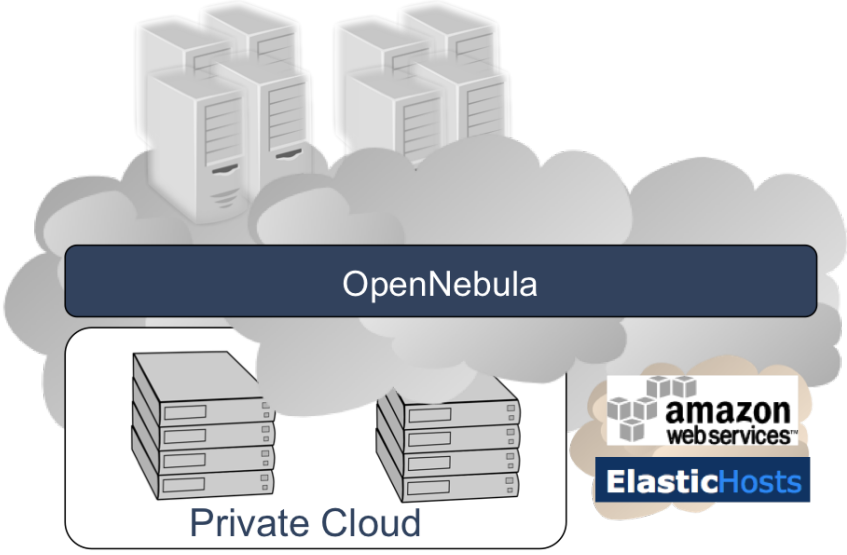

Figure 6.1: *Hybrid cloud.*

Amazon EC2 and Elastic Hosts (EH) (see Figure 6.2).

Several standardization bodies, like the Open Grid Forum (OGF) or the Distributed Management Task Force (DMTF), include new working groups to produce an standard cloud interface. For example, the Open Cloud Computing Interface (OCCI) specification is being developed by the OCCI Working Group inside the OGF community. The OCCI effort was initiated as the need for a global, open, non-propietary standard to define the infrastructure management interfaces in the context of cloud computing was made clear. OpenNebula supports this standard.

The OCCI specification defines a simple (about 15 commands) and extensible RESTful API. It defines three types of resources: compute, storage and network. Each resource is identified by a URI, has a set of attributes and is linked with other resources. Resources can be represented in many formats such as OCCI descriptor format, Open Virtualization Format (OVF) or Open Virtualisation Archive (OVA). Create, Read, Update and Delete (CRUD) methods are available for each resource, which are mapped to the typical REST methods over the HTTP protocol: POST, GET, PUT and DELETE. Other HTTP methods are also included by OCCI, like COPY, HEAD, MOVE and OPTIONS. Requests are used to trigger state changes and other operations (create backup, reconfigure, etc). A request is sent by a POST command to the resource URI, of which body contains the request and its parameters.

6.2 Image management

In general, a virtualized service consists in one or more components each one supported by one or more VMs. Instances of the same component are usually obtained

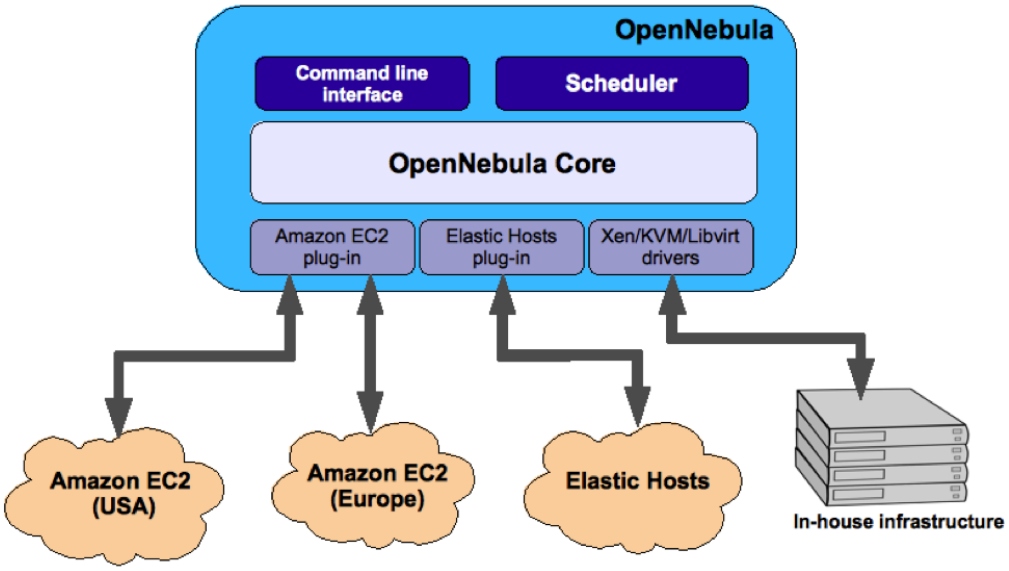

Figure 6.2: *Cloud brokering.*

by cloning a master image for that component, that contains a basic OS installation and the specific software required by the service.

Cloud providers use different formats and bundling methods to store and upload these master images. We could assume that suitable service component images has been previously packed and registered in each cloud provider storage service. So when a VM is to be deployed in a given cloud the image adapters skip any image transfer operation. This approach minimizes the service deployment time as no additional transfers are needed to instantiate a new service component. However, there are some drawbacks associated to the storage of one master image in each cloud provider: higher service development cycles as images have to be prepared and debug for each cloud; higher costs as clouds usually charge for the storage used; and higher maintenance costs as new images have to be distributed to each cloud.

Therefore, the project will investigate the feasibility of offering a repository of reference images for cloud users, with demonstrated interoperability among the supported public cloud infrastructures, and following the existing standards in the areas of VM images and virtual appliances.

TODO... StratusLab contextualisation strategy

6.3 Network management

Resources running on different cloud providers are located in different networks, and may use different addressing schemes (public addresses, private addresses with NAT, etc.). However, some kind of services require all their components to follow a uniform IP address scheme (for example, to be located on the same local net-

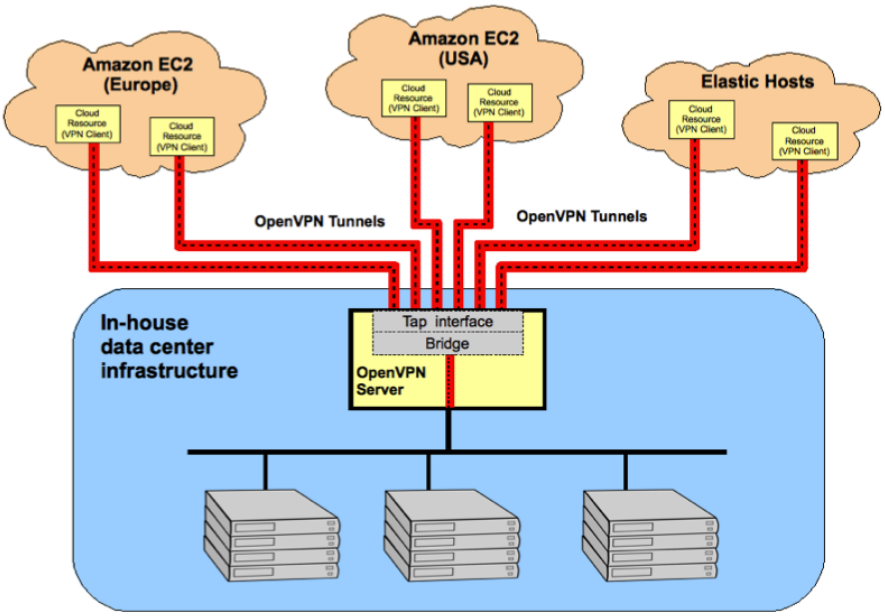

Figure 6.3: *VPN-based network configuration for a multi-cloud infrastructure*

work), so it could be necessary to build some kind of overlay network on top of the physical network to communicate the different service components. In this context, there are some interesting research proposals like ViNe [11], CLON [7], etc., or some commercial tools, like VPN-Cubed [2], which provide different overlay network solutions for grid and cloud computing environments.

Virtual Private Network (VPN) technology could interconnect the different cloud resources with the in-house data center infrastructure in a secure way. In particular, OpenVPN [10] software allows implementing Ethernet tunnels between each individual cloud resource and the data center LAN, as shown in Figure 6.3. In this setup, which follows a client-server approach, the remote cloud resources, configured as VPN clients, establish an encrypted VPN tunnel with in-house VPN server, so that each client enables a new network interface which is directly connected to the data center LAN. In this way, resources located at different clouds can communicate among them, and with local resources, as they were located in the same logical network, and they can access to common LAN services (NFS, NIS, etc.) in a transparent way, as local resources do.

7 Summary

This document serves as both a template for StratusLab official documents and as a guide on using that template. Feedback on this document should be sent to the author as well as the TSCG.

Glossary

Virtual Machine / VM	Running and virtualized operating system
Instance	see Virtual Machine / VM
Machine Image	Virtual machine file and metadata providing the source for Virtual Images or Instances
Appliance	Virtual machine containing preconfigured software or services
Appliance Repository	Repository of existing appliances
Regression	Features previously working which breaks in a new release of the software containing this feature
Front-End	OpenNebula server machine, which hosts the VM manager
Node	Physical host on which VMs are instantiated
Web Monitor	Web application providing basic monitoring of a single StratusLab installation
DCI	Distributed Computing Infrastructure
EGEE	Enabling Grids for E-sciencE
EGI	European Grid Infrastructure
EGI-TF	EGI Technical Forum
GPFS	General Parallel File System by IBM
Hybrid Cloud	Cloud infrastructure that federates resources between organizations
iSGTW	International Science Grid This Week
NFS	Network File System
NGI	National Grid Initiative
Public Cloud	Cloud infrastructure accessible to people outside of the provider's organization
Private Cloud	Cloud infrastructure accessible only to the provider's users
VM	Virtual Machine
VO	Virtual Organization
VOBOX	Grid element that permits VO-specific service to run at a resource center
Worker Node	Grid node on which jobs are executed

References

[1] CERN. Quattor. Online resource., 2010. http://www.quattor.org.

[2] cohesiveFT. VPN-Cubed. Online resource. http://www.cohesiveft.com/vpncubed.

[3] DMTF. Virtualization Management Initiative (VMAN). Online resource. http://www.dmtf.org/initiatives/vman_initiative.

[4] T. Freeman and K. Keahey. Contextualization: Providing One-Click Virtual Clusters. In *Proceedings of the eScience08 Conference*, December 2008.

[5] E. Huedo, R. Montero, and I. Llorente. A modular meta-scheduling architecture for interfacing with pre-WS and WS Grid resource management services. *Future Generation Computer Systems*, 23(2):252–261, 2007.

[6] I. Llorente, R. Moreno-Vozmediano, and R. Montero. Cloud Computing for on-Demand Grid Resource Provisioning. *Advances in Parallel Computing, IOS Press*, 2009.

[7] M. Matos, A. Sousa, J. Pereira, and R. Oliveira. Clon: overlay network for clouds. In *WDDM '09: Proceedings of the Third Workshop on Dependable Distributed Data Management*, pages 14–17, New York, NY, USA, 2009. ACM.

[8] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *Cloud Computing and Its Applications*, October 2008.

[9] U. of Chicago. Globus Nimbus. Online resource. http://workspace.globus.org.

[10] OpenVPN. Online resource. http://openvpn.net.

[11] M. Tsugawa and J. Fortes. A virtual network (ViNe) architecture for grid computing. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 10 pp., 25-29 2006.