StratusLab User's Guide

StratusLab Collaboration

Version 14.03.0

Contents

1 Preface **7**

1.1 Target Audience 7

1.2 Typographic Conventions 7

2 Introduction **9**

3 Quick Start **11**

3.1 Verifying the Prerequisities 11

3.2 Client Installation 12

3.3 Client Configuration 13

3.4 Testing the Installation 14

3.4.1 Virtual Machines 14

3.4.2 Persistent Disks 15

3.5 Conclusions 16

4 Command Line Client **17**

4.1 Overview 17

4.2 Installation 19

4.2.1 Verifying the Prerequisities 19

4.2.2 Local Installation 20

4.2.3 System Wide Installation 21

4.3 Client Configuration 21

4.3.1 Credentials 22
4.3.2 Service Endpoints 23
4.3.3 Other Credentials 24
4.4 Multi-cloud Configuration 24

5 Web Interfaces **27**
5.1 Marketplace 27
5.2 Storage 27
5.3 Registration 30

6 Virtual Machine Lifecycle **31**
6.1 Lifecycle Overview 31
6.2 Manage a VM 32
6.2.1 Find the Image Identifier 33
6.2.2 Start a Virtual Machine 33
6.2.3 Virtual Machine Status 33
6.2.4 Connect to the Virtual Machine 34
6.2.5 Terminating a Virtual Machine 34
6.3 Virtual Machine Resources 35

7 Storage Management **37**
7.1 Create persistent disk 37
7.2 List persistent disks 38
7.3 Using Persistent Disks 39
7.3.1 Launch VM with a persistent disk attached 39
7.3.2 Format attached disk 40
7.3.3 Mount disk, store data, unmount disk 40
7.3.4 Launch VM with the modified persistent disk attached 40
7.4 Hot-plug Persistent Disks 41
7.5 Delete Persistent Disks 43

7.6 Read Only Volumes 43
7.6.1 Create a Disk Image 44
7.6.2 Registering the Image 45
7.6.3 Using the Data Image 45
7.6.4 Summary 46
7.7 Volatile Storage 47

8 Image Management **49**
8.1 Finding Available Images 49
8.2 Building New Images from Existing Images 50
8.2.1 Automated Process 50
8.2.2 Manual Process 54
8.3 Building Images from Scratch 56
8.4 Contextualization 57
8.4.1 OpenNebula and HEPiX Contextualization 57
8.4.2 CloudInit 57
8.4.3 CloudInit-Enabled Images 57
8.4.4 Web Server Example 58
8.4.5 Future Work 60
8.4.6 Building an Image with CloudInit 60
8.5 Converting a VirtualBox Image 62

9 Programmatic Access **63**
9.1 REST Interfaces 63
9.1.1 StratusLab API 64
9.1.2 CIMI API 64
9.2 Libcloud API 64
9.2.1 Installing the Driver 64
9.2.2 Configuring the StratusLab Client 65
9.2.3 Using the Driver 65
9.2.4 Driver Status 66

A Python and Python Utilities **69**
A.1 Python 69
A.1.1 Linux Operating Systems 69
A.1.2 Mac OS X 70
A.1.3 Windows 70
A.2 Pip 70
A.3 Virtualenv 70

B SSH Client **73**
B.1 Linux Operating Systems 73
B.1.1 Client Installation 73
B.1.2 Creating an SSH Key Pair 73
B.1.3 SSH Agent 74
B.2 Mac OS X 74
B.2.1 Client Installation 74
B.2.2 Generating an SSH Key Pair 75
B.2.3 SSH Agent 75
B.3 Windows 75
B.3.1 Client Installation 75
B.3.2 Generating an SSH Key Pair 75
B.3.3 Logging into a VM with PuTTY 76
B.3.4 SSH Agent 76

C Java **77**
C.1 Linux Operating Systems 77
C.2 Mac OS X 77
C.3 Windows 78

D Windows PowerShell **79**

Chapter 1

Preface

1.1 Target Audience

This guide provides information of interest to researchers, scientists, and engineers looking to use an existing StratusLab cloud infrastructure. It also provides information for application developers looking to port their software to a StratusLab cloud environment.

System administrators wanting to install a StratusLab cloud on their own resources should start with the StratusLab Administrator's Guide, although they will also want to use the information in this guide to test their cloud after installation.

Those wishing to contribute to the StratusLab software or documentation should consult the StratusLab Contributor's Guide.

1.2 Typographic Conventions

This guide uses several typographic conventions to improve the readability.

links	some link
filenames	`$HOME/.stratuslab/stratuslab-user.cfg`
commands	`stratus-run-instance`
options	`--version`

Table 1.1: Typographic Conventions

Extended examples of commands and their outputs are displayed in the monospace Courier font. Within these sections, command lines are prefixed with a '$' prompt. Lines without this prompt are output from the previous command. For example,

```
$ stratus-run-instance -q BN1EEkPiBx87_uLj2-sdybSI-Xb
5507, 134.158.75.75
```

the `stratus-run-instance` is the command line which returns the virtual machine identifier and IP address.

Chapter 2

Introduction

StratusLab, an international collaboration, provides a complete, open source cloud distribution, allowing the installation of public or private "Infrastructure as a Service" cloud infrastructures. It aims to be both simple to use and simple to install.

Cloud infrastructures provide many benefits to developers and end-users (scientists and engineers). End-users appreciate the cloud's ability:

- To provide a customized execution environment,
- To make available pre-installed and pre-configured applications,
- To provision new resources (CPU, storage, etc.) rapidly, and
- To allow complete control of those resources.

Application and service developers also appreciate:

- Easy access to the services through simple APIs and
- The elasticity of the cloud to respond to peaks in demand.

StratusLab provides a couple mechanisms for accessing cloud resources: a command line interface written in portable python and web interfaces (for a subset of services). Developers can also use the Libcloud API, the StratusLab Python API, or the service REST APIs for programmatic access.

Chapter 3

Quick Start

This chapter provides the bare-bones instructions for getting up and running with the StratusLab command line client. The client provides a set of commands for interacting with all of the StratusLab services. Getting started involves just four steps:

1. Verifying the prerequisites,
2. Installation of the client,
3. Configuring the client, and
4. Testing the installation.

Each is covered in a section below. More detailed information on installing and configuring the client are in subsequent chapters; similarly later chapters also cover the utilization of the StratusLab services more throughly.

3.1 Verifying the Prerequisities

Before starting, you must verify that the prerequisites for the StratusLab command line are satisfied:

- Python 2 (2.6+), virtualenv and pip are installed.
- Java 1.6 or later is installed.

- An SSH client is installed with an SSH key pair.
- You have an active StratusLab account and connection parameters.

For the first three points, more information can be found in the following chapter and in the appendix.

For the StratusLab account, you must contact the administrator of your cloud infrastructure. The following parts of this chapter presume that you have a username/password pair for credentials and are using the StratusLab reference infrastructure at LAL.

3.2 Client Installation

First create a virtual environment to hold the StratusLab client and its dependencies.

```
$ virtualenv $HOME/env/SL
New python executable in /home/sluser/env/SL/bin/python
Installing setuptools.............done.
Installing pip...............done.
```

You may want to choose a different name or location for your virtual environment. Now you will need to activate that environment.

```
$ source $HOME/env/SL/bin/activate
(SL)$
```

The prompt should change to include the name of the virtual environment.

Now use pip to install the StratusLab client:

```
(SL)$ pip install stratuslab-client
Downloading/unpacking stratuslab-client
  Downloading stratuslab-client-13.05.0.RC1.tar.gz (1.2MB)
     : 1.2MB downloaded
...
Successfully installed stratuslab-client dirq ...
Cleaning up...
```

You can verify that the client is installed and accessible with by searching for one of the StratusLab commands:

```
(SL)$ which stratus-copy-config
~/env/SL/bin/stratus-copy-config
```

All of the StratusLab commands begin with "'stratus-". On systems that support it, you can use tab completion to see all of the available commands.

3.3 Client Configuration

Now that the StratusLab client is installed, it needs to be configured. You will need to have your credentials and the cloud service endpoints available.

Copy the reference configuration file into place and verify it is present:

```
(SL)$ stratus-copy-config

(SL)$ ls $HOME/.stratuslab/
stratuslab-user.cfg
```

This configuration file contains descriptions of all of the parameters that can be set. There are only three or four that must be set.

Set the service endpoints for the cloud entry point and the storage (pdisk):

```
endpoint = cloud.lal.stratuslab.eu
pdisk_endpoint = pdisk.lal.stratuslab.eu
```

substituting the values for your cloud infrastructure. (These are the values for the StratusLab reference cloud infrastructure at LAL.) Also set the values for your username and password:

```
username = your.username
password = your.password
```

again substituting your values for these parameters.

You can now see if you have any running machines on the cloud to test if the client is correctly installed:

```
(SL)$ stratus-describe-instance
id  state     vcpu memory    cpu% host/ip
   name
(SL)$
```

This should return an empty list of machines. If it returns any errors, then you'll need to correct whatever went wrong in the installation. See the more detailed documentation.

3.4 Testing the Installation

To test the installation more throughly and to give you an idea how to use the cloud, we will start a virtual machine and create a persistent disk.

3.4.1 Virtual Machines

Normally, you would browse the StratusLab Marketplace to find an image that is useful for you. You then use the image identifier to start a copy of that image.

We use the image identifier `BN1EEkPiBx87_uLj2-sdybSI-Xb` for a ttylinux image. This is a very minimal linux distribution usually intended as an embedded operating system. It is extremely small and boots quickly, making it ideal for tests.

Launch a virtual machine instance using this image:

```
(SL)$ stratus-run-instance BN1EEkPiBx87_uLj2-sdybSI-Xb

 ::::::::::::::::::::::::
 :: Starting machine(s) ::
 ::::::::::::::::::::::::
 :: Starting 1 machine
 :: Machine 1 (vm ID: 4710)
 Public ip: 134.158.75.152
 :: Done!
```

This gives you the VM identifier (4710) and the IP address from where the machine can be accessed. Afterwards, check the status of the machine.

```
(SL)$ stratus-describe-instance
id   state     vcpu memory    cpu% host/ip
                  name
4710 Running   1    1572864   8    vm-152.lal.stratuslab.
   eu one-4710
```

You may have to wait a little while until it is in a running state. Then verify that the machine is accessible with `ping`. Once it is visible try to log into the machine:

```
(SL)$ stratus-connect-instance 4710
...
Enter passphrase for key '/home/sluser/.ssh/id_rsa':
# hostname
ttylinux_host
# exit
```

Note that the password is the password for your SSH key. You can also log in directly using ssh:

```
(SL)$ ssh root@vm-152.lal.stratuslab.eu
...
Enter passphrase for key '/home/sluser/.ssh/id_rsa':
#
```

Note that your must use the username defined by the person that created the image. This is almost always "root". In both these cases, information about the SSH host key have been suppressed for clarity.

The machine can then be killed (stopped) with the command:

```
(SL)$ stratus-kill-instance 4710
(SL)$
(SL)$ stratus-describe-instance 4710
id   state     vcpu memory    cpu% host/ip
                  name
4710 Done      1    1572864   1    vm-152.lal.stratuslab.
   eu one-4710
```

The resources allocated to the machine are only released when the machine is killed. If you shutdown the machine while inside the operating system with `halt` or `shutdown`, you must still use the StratusLab command to kill the machine to release the resources.

3.4.2 Persistent Disks

The lifecycle for a persistent disk (or volume) is also rather simple.

To create a disk:

```
(SL)$ stratus-create-volume --size 1 --tag=mydisk
DISK 08f59022-463a-4662-8136-c0cee5517f17
```

This creates a new disk with the given tag and a size of 1 GiB. The UUID is the identifier for the disk.

The disks can be listed with the command:

```
(SL)$ stratus-describe-volumes
:: DISK 08f59022-463a-4662-8136-c0cee5517f17
   count: 0
   owner: cal
   tag: mydisk
   size: 1
```

Normally at this point, it would be attached to a virtual machine instance, formatted, and then used to store data. We will leave that for the detailed chapters below.

When the disk is no longer needed, it can be deleted with the command:

```
(SL)$ stratus-delete-volume 08f59022-463a-4662-8136-
   c0cee5517f17
DELETED 08f59022-463a-4662-8136-c0cee5517f17
```

That is the complete lifecycle for a persistent disk.

3.5 Conclusions

This chapter has shown you the procedure for installing the client and then, the basic lifecycles for starting machines and for creating persistent disks. Hopefully, you are intrigued enough to read following chapters that provide more detail on the StratusLab services and their functionality.

Chapter 4

Command Line Client

StratusLab provides a simple command line client that is easy to install on all platforms. This client provides access to all of the StratusLab services.

4.1 Overview

The command line client is the principal method for accessing the StratusLab services. It is written almost entirely in portable python and is easy to install and configure on all platforms (Linux, Mac OS X, Windows).

A list of the commands along with brief descriptions are provided in the table. All of the commands start with the `stratus-` prefix. On platforms that support it, tab completion can be used to find all of the commands. All of the commands support the `--help` option, which provides detailed information about the command and its options. All of the commands also support the `--version` option that will display the version of the client being used.

`stratus-copy-config`	copy reference configuration file into the correct location
`stratus-describe-instance`	list VM instances
`stratus-run-instance`	start a new VM instance
`stratus-kill-instance`	destroy and release resources for a given VM instance
`stratus-shutdown-instance`	stop and save a VM instance, must be used with `--save` option when starting VM instance
`stratus-describe-volumes`	list persistent disk volumes
`stratus-create-volume`	create a new persistent disk volume
`stratus-delete-volume`	destroy an existing persistent disk volume
`stratus-attach-volume`	dynamically attach a persistent disk volume to a VM instance
`stratus-detach-volume`	dynamically detach a persistent disk volume from a VM instance
`stratus-update-volume`	update the metadata for a persistent disk volume
`stratus-build-metadata`	create a Marketplace entry for a VM image
`stratus-sign-metadata`	cryptographically sign a Marketplace entry for a VM image
`stratus-validate-metadata`	verify that a VM image entry is syntactically correct and signed
`stratus-upload-metadata`	upload a VM image entry to the Marketplace
`stratus-deprecate-metadata`	deprecate a Marketplace entry for a VM image
`stratus-create-image`	create a new VM image from an existing image
`stratus-upload-image`	(deprecated) upload an image to the appliance repository
`stratus-connect-instance`	connect via ssh to a given VM instance
`stratus-hash-password`	hash a password to give to cloud administrator
`stratus-prepare-context`	prepare a CloudInit contextualization file
`stratus-run-cluster`	utility to run a batch cluster on the cloud

Table 4.1: Overview of StratusLab commands.

4.2 Installation

4.2.1 Verifying the Prerequisities

Before starting, you must verify that the prerequisites for the StratusLab command line are satisfied:

- Python 2 (2.6+), virtualenv and pip are installed.
- Java 1.6 or later is installed.
- An SSH client is installed with an SSH key pair.
- You have an active StratusLab account and connection parameters.

Quick recipes for checking the first three points are below, with more detailed information in the appendices.

For the StratusLab account, you must contact the administrator of your cloud infrastructure. The administrator must give you 1) your credentials and 2) the service endpoints of the cloud infrastructure.

You can quickly verify the availability of Python and utilities with the following commands:

```
$ python --version
Python 2.6.6

$ which virtualenv
/usr/bin/virtualenv
```

Note that pip is always installed with virtualenv.

Similarly for java use the following:

```
$ java -version
java version "1.7.0_19"
OpenJDK Runtime Environment (rhel-2.3.9.1.el6_4-x86_64)
OpenJDK 64-Bit Server VM (build 23.7-b01, mixed mode)
```

Note that there is only one hyphen in the option.

For SSH check to see if the directory `$HOME/.ssh/` contains files starting with "`id_`".

```
$ ls $HOME/.ssh/id_*
/home/sluser/.ssh/id_rsa  /home/sluser/.ssh/id_rsa.pub
```

4.2.2 Local Installation

For local installations of the StratusLab client, for example within a non-priviledged user account or on a user's laptop, an installation using virtualenv and pip is very strongly recommended.

Assuming that the prerequisites are installed, the installation is just a matter of a few commands. First create a virtual environment to hold the StratusLab client and its dependencies.

```
$ virtualenv $HOME/env/SL
New python executable in /home/sluser/env/SL/bin/python
Installing setuptools.............done.
Installing pip...............done.
```

You may want to choose a different name or location for your virtual environment. Now you will need to activate that environment.

```
$ source $HOME/env/SL/bin/activate
(SL)$
```

The prompt should change to include the name of the virtual environment.

Now use pip to install the StratusLab client:

```
(SL)$ pip install stratuslab-client
Downloading/unpacking stratuslab-client
  Downloading stratuslab-client-13.05.0.RC1.tar.gz ...
...
Successfully installed stratuslab-client dirq ...
Cleaning up...
```

This will also install all of the required Python dependencies for the client as well. (The above output has been abridged.)

You can verify that the client is installed and accessible with by searching for one of the StratusLab commands:

```
(SL)$ which stratus-copy-config
~/env/SL/bin/stratus-copy-config
```

All of the StratusLab commands begin with `stratus-`. On systems that support it, you can use tab completion to see all of the available commands.

Once the client is installed, it must be configured. See the instructions below.

4.2.3 System Wide Installation

The above method can also be used for system wide installations for multi-user machines. Simply use pip directly without using virtualenv.

Additionally, StratusLab provides client packages for RedHat Enterprise Linux (RHEL) systems. These packages work also on derivatives of these systems like CentOS, ScientificLinux, and OpenSuSE.

You must have root access to your machine to install these packages. For RHEL and RHEL-like systems, it is recommended to do the installation with yum. The configuration for yum tells yum where to find the StratusLab packages. Choose the `centos-6` repository. Use the command:

```
$ yum install stratuslab-cli-user
```

to install the latest version of the client tools.

For SuSE, configure zypper for the StratusLab OpenSuSE repository ("opensuse-12.1") and use it to install the package.

Users must then configure the client within their accounts.

4.3 Client Configuration

The values of the configuration parameters for the client can be provided in several different ways. The various mechanisms in order of precedence are:

- Command line parameters
- Environment variables (`STRATUSLAB_*`)
- Configuration file parameters
- Defaults in the code

The names of the command line parameters and the environmental variables can be derived from the name of the configuration file parameter. The table gives an example for one parameter and the algorithm for deriving the other names.

`pdisk_endpoint`	configuration file parameter
`--pdisk-endpoint`	command line option: change underscores to hyphens and prefix with two hyphens (`--`)
`STRATUSLAB_PDISK_ENDPOINT`	environmental variable: make all letters uppercase, prefix with `STRATUSLAB_`

Table 4.2: Deriving command line option and environmental names from the configuration file parameter.

The configuration file is in the standard INI format. The file **must** contain the `[default]` section. It may also contain additional sections describing parameters for different cloud infrastructures.

A minimal configuration file, assuming that a username/password pair is used for credentials is:

```
[default]
endpoint = cloud.lal.stratuslab.eu
pdisk_endpoint = pdisk.lal.stratuslab.eu
username = sluser
password = slpass
```

The values will obviously have to change to correspond to your credentials and the cloud infrastructure that you are using.

4.3.1 Credentials

StratusLab provides a very flexible authentication system, supporting username/password pairs, X509 certificates, Globus/VOMS certificate proxies, and PKCS12 certificates.

username	user's StratusLab username
password	user's password
pem_certificate	X509 or proxy certificate file
pem_key	X509 or proxy key file

Table 4.3: Parameters for supplying user credentials.

Users should specify either the username/password or the pem_certificate/pem_key parameters but not both sets. If both are specified, then the username/-password will take precedence.

The default for the pem certificate and key are the files `usercert.pem` and `userkey.pem`, respectively in the directory `$HOME/.globus`. Contrary to the usual rules, the command line parameter for `pem_certificate` is `--pem-cert`.

4.3.2 Service Endpoints

The user must also specify the cloud service endpoints. These will be provided by the cloud administrator.

endpoint	cloud entry point (VM mgt.)
pdisk_endpoint	pdisk entry point (storage mgt.)
marketplace	Marketplace URL

Table 4.4: Cloud service endpoints.

If the `pdisk_endpoint` parameter is not specified, then the value for `endpoint` will be used. The default value for the `marketplace` parameter is `https://marketplace.stratuslab.eu/`, the central StratusLab Marketplace.

4.3.3 Other Credentials

Various other credentials are used to access running virtual machines and for signing information for the Marketplace.

`user_public_key_file`	user public SSH key file
`p12_certificate`	PKCS12-formatted certificate
`p12_password`	password for PKCS12 certificate

Table 4.5: Other user credentials.

The default for the public SSH key file is `$HOME/.ssh/id_rsa.pub`. Contrary to the usual rules, the environmental variable and command line parameter are `STRATUSLAB_KEY` and `--key`, respectively.

The PKCS12 certificate is used to sign image metadata entries before uploading them to the Marketplace. Contrary to the usual rules, the command line option for the parameter `p12_certificate` is `--p12-cert`.

4.4 Multi-cloud Configuration

If more than one StratusLab cloud infrastructure is being used, then the configuration for all of these can be kept in a single file. This allows you to quickly switch between the various clouds.

In the configuration file it is possible to create uniquely named user specific sections to define any of the variables described above. Example of 'endpoint'

```
[my-section]
endpoint = <another.cloud.frontend.hostname>
username = <another.username>
password = <another.password>
```

which can be activated using the `selected_section` parameter in the `[default]` section of the configuration file:

```
selected_section = my-section
```

It can also be activated using command-line options `--user-config-section` or `-S` or via the environmental variable `STRATUSLAB_USER_CONFIG_SECTION`.

This example configuration file defines custom 'fav-cloud' section with endpoint/credentials and a custom SSH key:

```
[fav-cloud]
endpoint = favorit-cloud.tld
username = clouduser
password = cloudpass
user_public_key_file = /home/clouduser/.ssh/id_rsa-
   favcloud.pub
```

Values for parameters not specified in this section will be taken from the `[default]` section.

Chapter 5

Web Interfaces

Web interfaces are often more intuitive than command line interfaces. Moreover, they do not require the installation of software on the user's machine (aside from a web browser!).

Eventually StratusLab will provide a unified portal to allow access to all services via a web browser, but this is not yet available. Nevertheless, several of the StratusLab services provide a web interface. These interfaces are essentially a thin veneer over the services' REST interfaces.

5.1 Marketplace

For the Marketplace, the primary interface is via a web browser. This interface allows users to search for available images, to access the metadata in several formats (HTML, XML, and JSON), and to upload new metadata entries.

5.2 Storage

The storage service also provides a web browser interface that mirrors the underlying REST interface. This interface allows a users to see the complete list of disks, manage the metadata of the disks, and to mount and dismount them from virtual machines dynamically.

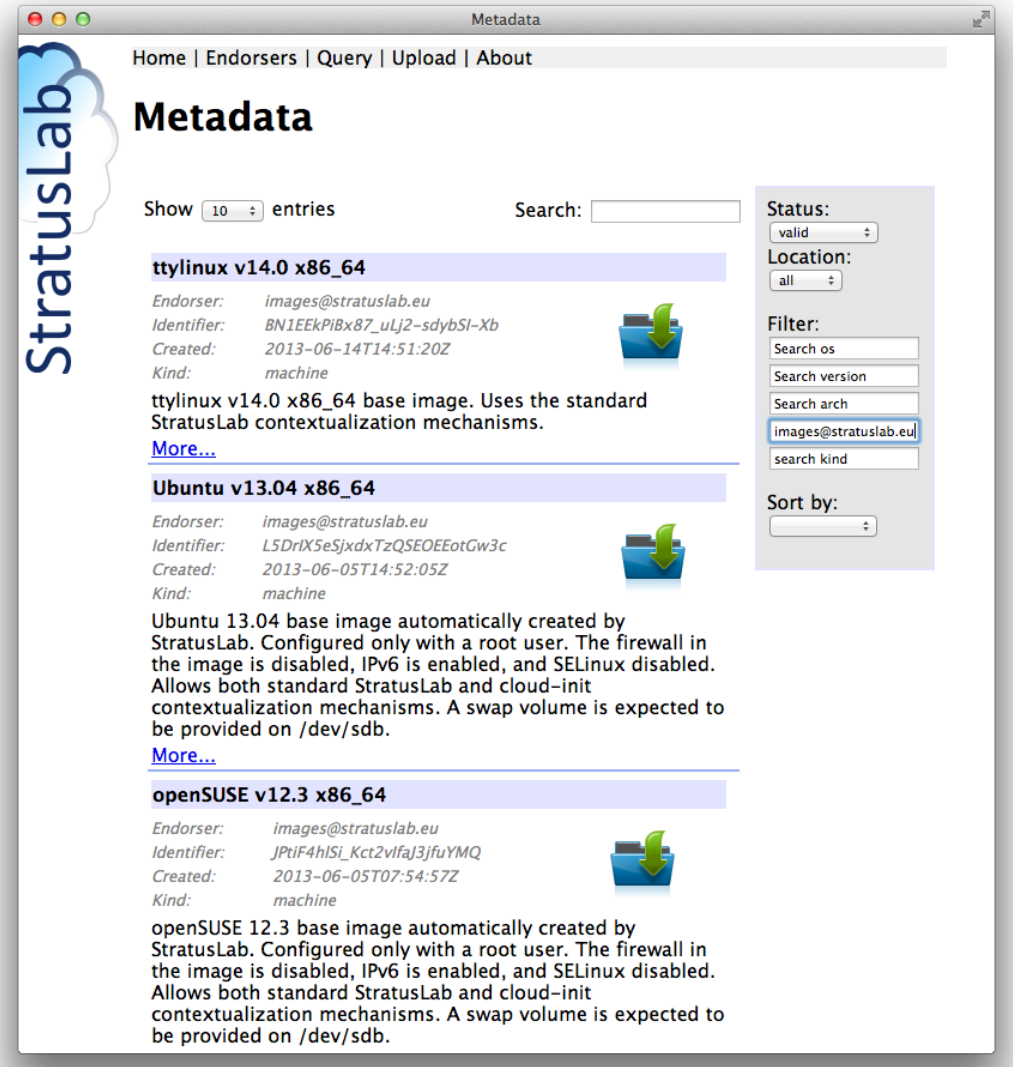

Figure 5.1: Marketplace Screenshot

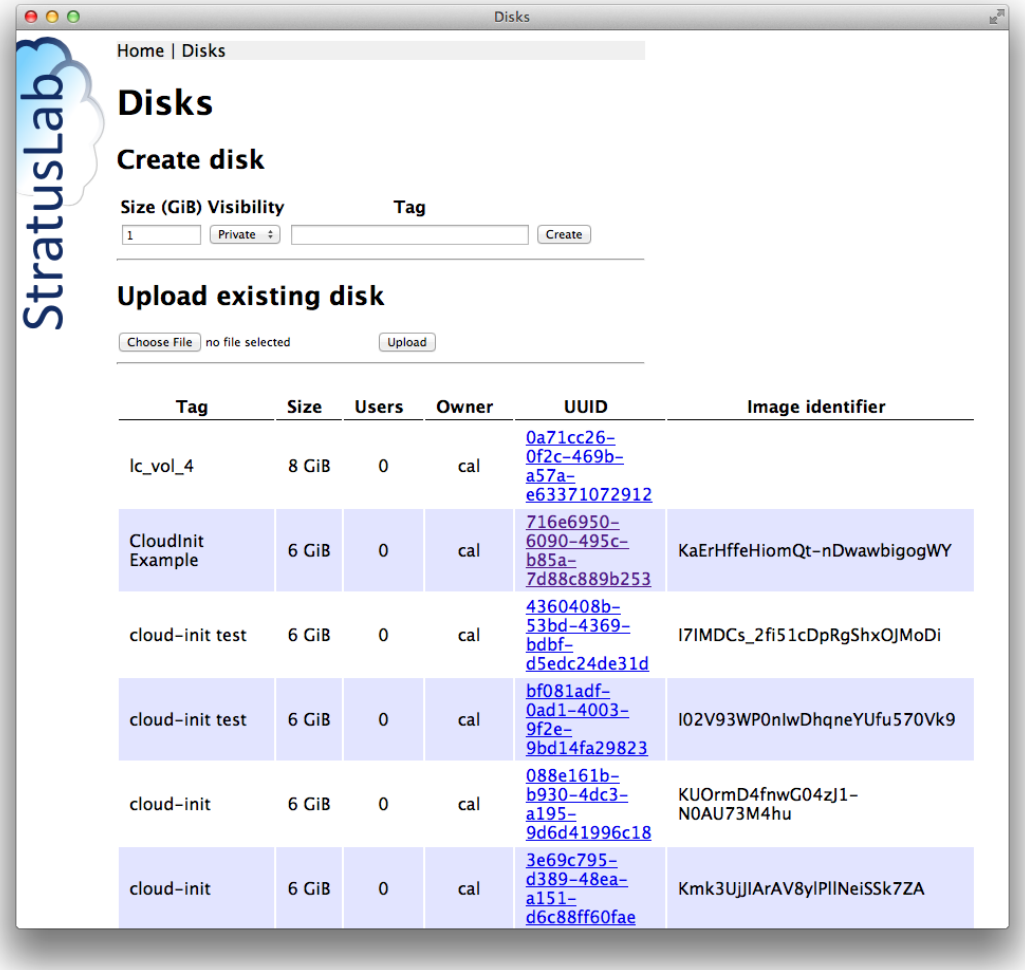

Figure 5.2: Storage Service Screenshot

5.3 Registration

The registration service is an optional service that allows users to register for access to a cloud infrastructure and to manage the information associated with their account. If used, the cloud administrator validates the account requests.

This service is indeed used for the StratusLab reference cloud infrastructure and users may register via this service instance.

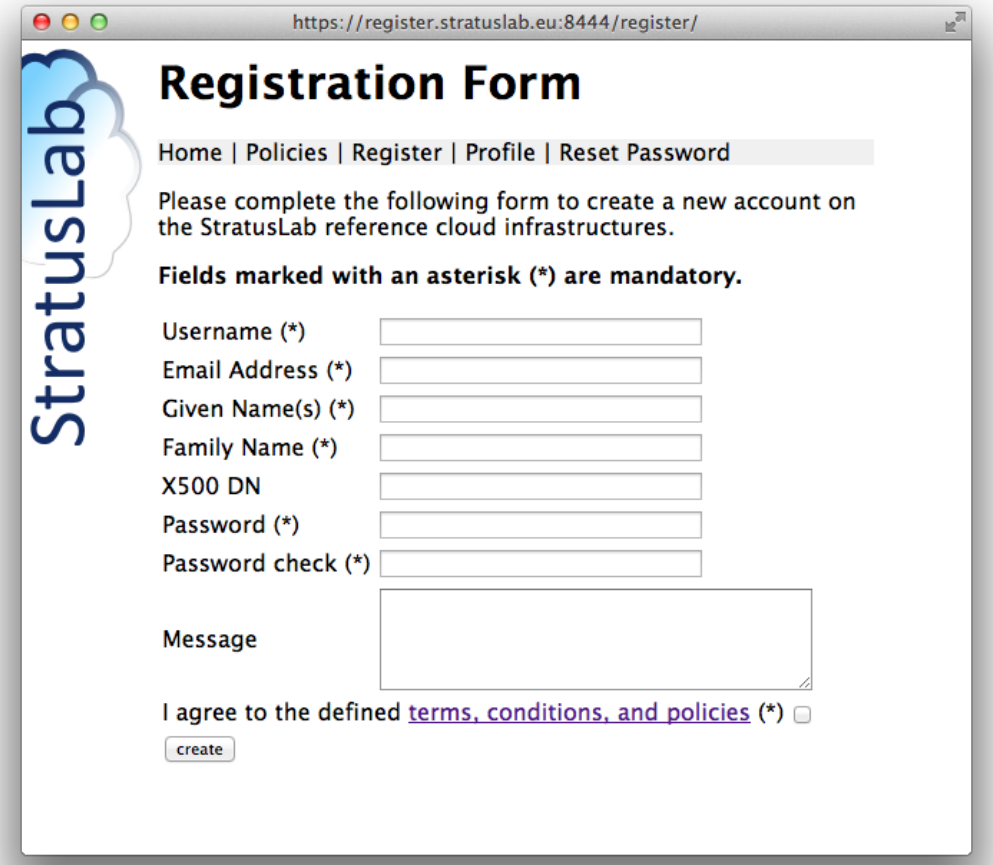

Figure 5.3: Registration Screenshot

Chapter 6

Virtual Machine Lifecycle

Managing virtual machines (VMs) is the core functionality associated with IaaS cloud infrastructures. StratusLab is no different, providing the commands to start and stop virtual machines. Users can define the resources allocated to these VMs–CPUs, RAM, swap space, and volatile disk space.

6.1 Lifecycle Overview

From the user's perspective, the VM lifecycle is rather simple. It consists of the following steps:

1. Search the Marketplace for virtual machine image to run on the cloud.
2. Launch a machine instance via the cloud entry point using the VM image identifier.
3. Obtain the machine instance's network address.
4. Use and control the VM, usually for example, logging into the VM as root via SSH.
5. Shutdown the virtual machine and release the resources.

For the first, you need to use a web browser to select an appropriate image. The remaining steps each correspond to a StratusLab command:

- `stratus-run-instance`: deploy a VM given the Marketplace identifier of the image
- `stratus-describe-instance`: find the state of all of the active VMs or of a single VM
- `stratus-connect-instance`: connect via SSH to the machine (raw SSH commands can also be used)
- `stratus-kill-instance`: stop the machine and deallocate all of its resources

The detailed lifecycle of a machine is more complicated. The diagram shows the full lifecycle and describes what is happening behind the scenes in each of these cases.

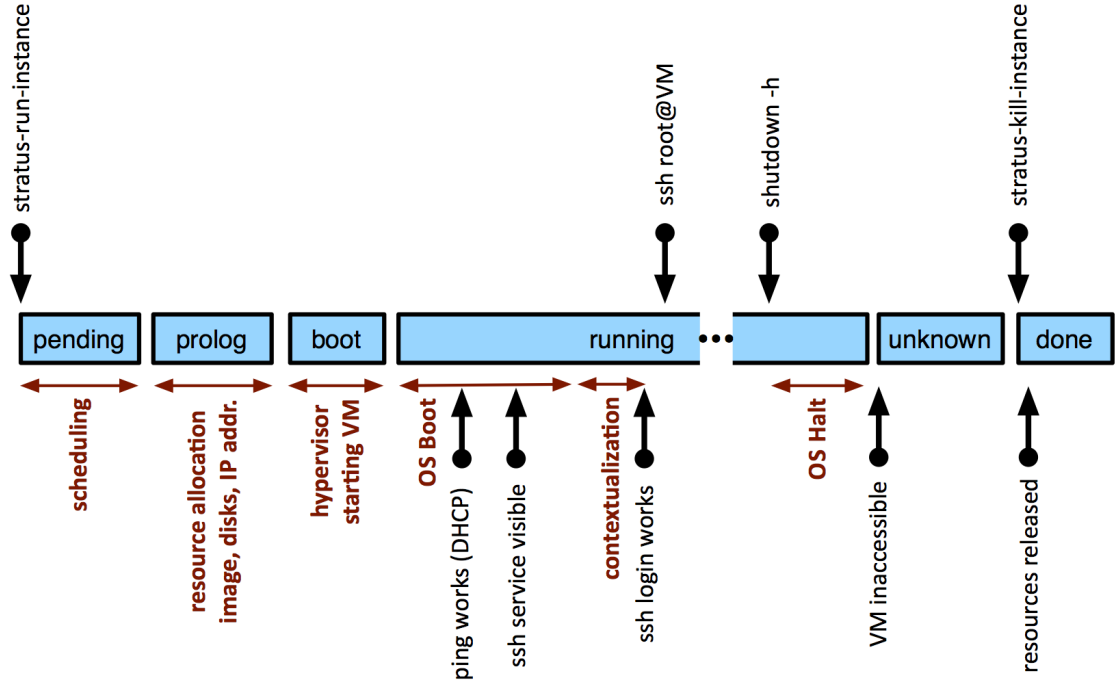

Figure 6.1: Virtual machine timeline and states.

6.2 Manage a VM

Probably the easiest way to see how this works is to run through a complete example. We will go through a complete lifecycle for a ttylinux

machine. The ttylinux distribution is a small linux distribution mostly intended for embedded systems. Its small size and fast boot make it ideal for tests.

6.2.1 Find the Image Identifier

First you would normally browse the Marketplace to find a suitable image. You can find the images provided by StratusLab by searching for the endorser "images@stratuslab.eu". For our case, the ttylinux image has the identifier: "BN1EEkPiBx87_uLj2-sdybSI-Xb".

6.2.2 Start a Virtual Machine

To start the virtual machine, use the `stratus-run-instance` command:

```
$ export TTYLINUX_ID=BN1EEkPiBx87_uLj2-sdybSI-Xb
$ stratus-run-instance ${TTYLINUX_ID}
 ::::::::::::::::::::::::
 :: Starting machine(s) ::
 ::::::::::::::::::::::::
 :: Starting 1 machine
 :: Machine 1 (vm ID: 165)
     Public ip: 134.158.75.201
 :: Done!
```

This provides the virtual machine identifier (165 in this case) and the IP address at which the machine will be visible.

6.2.3 Virtual Machine Status

To find the status of all active virtual machine, you can use the `stratus-describe-instance` command without any parameters:

```
$ stratus-describe-instance
id  state     vcpu memory    cpu% host/ip
                    name
165 Running   1    0         0    vm-201.lal.stratuslab.eu
    one-165
166 Pending   1    0         0    vm-202.lal.stratuslab.eu
    one-166
```

The status of a single machine can be found by giving the VM identifier:

```
$ stratus-describe-instance 165
id  state     vcpu memory    cpu% host/ip
                    name
165 Running   1    131072    1    vm-201.lal.stratuslab.eu
    one-165
```

More details will be provided if you increase the verbosity of the commands with the options `-v`, `-vv`, or `-vvv`. More letters provide increasingly more verbosity. This is especially helpful when virtual machines fail.

6.2.4 Connect to the Virtual Machine

You can use `ping` to determine when the machine becomes visible on the network. Once it is visible (and the SSH daemon has started on the VM), you can connect to the machine directly with SSH:

$ ssh root@vm-201.lal.stratuslab.eu # # echo $USER root #

or you can use the command `stratus-connect-instance` with the VM identifier. This command is a simple wrapper around the SSH commands.

6.2.5 Terminating a Virtual Machine

To safely stop all services and halt a virtual machine, use the standard `shutdown` or `halt` commands from within the virtual machine.

```
# shutdown -h
#
Connection to vm-201.lal.stratuslab.eu closed by remote
   host.
Connection to vm-201.lal.stratuslab.eu closed.
```

The machine will stop and the status will eventually become an "unknown" state.

```
$ stratus-describe-instance 165
id  state     vcpu memory    cpu% host/ip
                    name
```

```
165 Unknown    1    131072    0    vm-201.lal.stratuslab.eu
    one-165

$ stratus-kill-instance 165
$
```

This mechanism ensures that all resources (especially data volumes) are shut down cleanly and released. Note that the VM resources are **not** released until the `stratus-kill-instance` command is run.

You can also forcably stop and remove machine by just running the `stratus-kill-instance` command:

```
$ stratus-kill-instance 166
$
$ stratus-describe-instance 166
id  state     vcpu memory    cpu% host/ip
                  name
166 Done      1    131072    0    vm-202.lal.stratuslab.eu
    one-166
```

This is the essentially the equivalent of pulling the power cord out of a physical machine, so be careful when doing this, especially if persistent data volumes are attached to the virtual machine.

6.3 Virtual Machine Resources

You can control the number of CPUs, amount of RAM and size of the swap space allocated to a virtual machine. StratusLab provides a number of predefined machine cnofigurations. You can obtain a list of these with the command:

```
$ stratus-run-instance --list-type
  Type              CPU        RAM       SWAP
  c1.medium       1 CPU     256 MB    1024 MB
  c1.xlarge       4 CPU    2048 MB    2048 MB
  m1.large        2 CPU     512 MB    1024 MB
* m1.small        1 CPU     128 MB    1024 MB
  m1.xlarge       2 CPU    1024 MB    1024 MB
  t1.micro        1 CPU     128 MB     512 MB
```

You can select the configuration you want by using the `--type` option to `stratus-run-instance` and providing the name of the type. The default is the type marked with an asterisk ("m1.small").

You can also individually specify the CPU, RAM, and swap space with the `--cpu`, `--ram`, and `--swap` options. These will override the corresponding in the value in the selected type.

Note that the maximum values are determined by the largest physical machine in the cloud infrastructure. The cloud administrator of your infrastructure can provide these limits.

Chapter 7

Storage Management

Persistent Disk Storage is the StratusLab service that physically stores machine and disk images as volumes on the cloud site. It facilitates quick startup of VMs and hot-plugging of disk volumes as block devices to the VMs.

7.1 Create persistent disk

Before creating persistent disks (or volumes)[1], you should define the persistent disk storage endpoint by either of

- in your HOME/.stratuslab/stratuslab-user.cfg
- set the environment variable STRATUSLAB_PDISK_ENDPOINT
- pass it as **--pdisk-endpoint** command when creating one.

Lets create a persistent disk of 5GB and named "myprivate-disk"

```
$ stratus-create-volume --size=5 --tag=myprivate-disk
DISK 9c5a2c03-8243-4a1b-a248-0f0d22d948c2
```

The command returned the UUID of the created persistent disk. Newly created disk is by default

- private - can be read, written and deleted only by their owner

[1] "disk" and "volume" are used interchangeably.

- of type **read-write data image**

To create public persistent disk, pass --public argument to stratus-create-volume

```
$ stratus-create-volume --public --size=10 --tag=mypublic-
   disk
DISK d955fda6-bf9c-4aa8-abc4-5bbcdb83021b
```

or update the disk with

```
stratus-update-volume --public <UUID>
```

To get the list of available disk types and the properties that can be updated on the disk run

```
stratus-update-volume -h
```

7.2 List persistent disks

stratus-describe-volumes allows you to query the list of all your public and private persistent disks, and also all the public persistent disks created by other users.

```
$ stratus-describe-volumes
:: DISK a4324f26-39e0-4965-8c8f-3287cd0936e5
        created: 2011/07/20 16:37:10
        visibility: public
        tag: mypublic-disk
        owner: testor2
        size: 5
        users: 0
:: DISK 9c5a2c03-8243-4a1b-a248-0f0d22d948c2
        created: 2011/07/20 16:10:37
        visibility: private
        tag: myprivate-disk
        owner: testor1
        size: 5
        users: 0
:: DISK d955fda6-bf9c-4aa8-abc4-5bbcdb83021b
        created: 2011/07/20 16:26:31
```

```
        visibility: public
        tag: mypublic-disk
        owner: testor1
        size: 5
        users: 0
```

The above command lists 'testor1' public and private persistent disks, and also 'testor2' public ones.

7.3 Using Persistent Disks

Workflow:

- Launch a virtual machine instance referencing a persistent disk
- Format the disk in the running VM
- Write data to the disk
- Dismount the disk or halt the machine instance
- Disk with persistent data is available for use by another VM
- Launch another VM referencing the modified persistent disk

7.3.1 Launch VM with a persistent disk attached

To Launch a VM we will be using the ttylinux image identifier GOaxJF-doEXvqAm9ArJgnZ0_ky6F from StratusLab Marketplace (default one).

--persistent-disk=UUID option when used with stratus-run-instance, tells StratusLab to attach the referenced persistent disk(UUID) to the VM.

Instantiate ttylinux image with reference to your private persistent disk 9c5a2c03-8243-4a1b-a248-0f0d22d948c2.

```
$ stratus-run-instance \
    --persistent-disk=9c5a2c03-8243-4a1b-a248-0f0d22d948c2
        \
    GOaxJFdoEXvqAm9ArJgnZ0_ky6F

 :::::::::::::::::::::::::
```

```
 :: Starting machine(s) ::
 ::::::::::::::::::::::::
 :: Starting 1 machine
 :: Machine 1 (vm ID: 3)
        Public ip: 134.158.75.35
```

Log into your VM using ssh, depending in the linux kernel and distribution version of your VM, your persistent disk will be referenced as /dev/hdc or /dev/sdc.

In ttylinux, it will be /dev/hdc.

Make sure that your disk was attached to your VM

```
$ ssh root@134.158.75.35
# fdisk -l
.........
Disk /dev/hdc: 5368 MB, 5368709120 bytes
255 heads, 63 sectors/track, 652 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
.........
```

7.3.2 Format attached disk

```
# mkfs.ext3 /dev/hdc
```

7.3.3 Mount disk, store data, unmount disk

```
# mount /dev/hdc /mnt
# echo "Testing␣Persistent␣Disk" > /mnt/test_pdisk
# umount /mnt
```

Your persistent disk is ready to be used by another VM.

7.3.4 Launch VM with the modified persistent disk attached

Instantiate new VM ttylinux with the same reference to your private persistent disk 9c5a2c03-8243-4a1b-a248-0f0d22d948c2.

```
$ stratus-run-instance \
    --persistent-disk=9c5a2c03-8243-4a1b-a248-0f0d22d948c2
        \
    GOaxJFdoEXvqAm9ArJgnZ0_ky6F
```

```
 ::::::::::::::::::::::::
 :: Starting machine(s) ::
 ::::::::::::::::::::::::
 :: Starting 1 machine
 :: Machine 1 (vm ID: 4)
        Public ip: 134.158.75.36
```

Log into your VM using ssh, verify existence of your persistent disk

```
$ ssh root@134.158.75.35
# fdisk -l
...........
Disk /dev/hdc: 5368 MB, 5368709120 bytes
255 heads, 63 sectors/track, 652 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
...........
```

Mount your persistent disk

```
# mount /dev/hdc /mnt
# ls /mnt
lost+found  test_pdisk
# cat /mnt/test_pdisk
Testing Persistent Disk
```

7.4 Hot-plug Persistent Disks

StratusLab storage also provides hot-plug feature for persistent disk. With stratus-attach-instance you can attach a volume to a running machine and with stratus-detach-instance you can release it.

To use the hot-plug feature, the running instance needs to have acpiphp kernel module loaded. Image like ttylinux doesn't have this feature, you have to use base image like Ubuntu, CentOS or Fedora.

Before hot-plugin a disk, make sure acpiphp is loaded. In your VM execute

```
modprobe acpiphp
```

To attach two volumes to the VM ID 24 with the UUIDs 1e8e9104-681c-4269-8aae-e513c6723ac6 and 5822c376-9ce1-434e-95d1-cdaa240cd47c

```
$ stratus-attach-volume -i 24 1e8e9104-681c-4269-8aae-
   e513c6723ac6 5822c376-9ce1-434e-95d1-cdaa240cd47c
ATTACHED 1e8e9104-681c-4269-8aae-e513c6723ac6 in VM 24 on
   /dev/vda
ATTACHED 5822c376-9ce1-434e-95d1-cdaa240cd47c in VM 24 on
   /dev/vdb
```

Use the fdisk -l command as above to see the newly attached disks.

Make sure to unmount any file systems on the device within your operating system before detaching the volume. Failure to unmount file systems, or otherwise properly release the device from use, can result in lost data and will corrupt the file system.

```
umount /dev/vda
umount /dev/vdb
```

When you finish using your disks, you can detach them from the running VM

```
$ stratus-detach-volume -i 24 1e8e9104-681c-4269-8aae-
   e513c6723ac6 5822c376-9ce1-434e-95d1-cdaa240cd47c
DETACHED 1e8e9104-681c-4269-8aae-e513c6723ac6 from VM 24
   on /dev/vda
DETACHED 5822c376-9ce1-434e-95d1-cdaa240cd47c from VM 24
   on /dev/vdb
```

On running instance detaching not hot-plugged disks or disks that were not or are no longer attached to the instance will result in an error

```
$ stratus-detach-volume -i 41 2a17226f-b006-45d8-930e-13
   fbef3c6cdc
DISK 2a17226f-b006-45d8-930e-13fbef3c6cdc: Disk have not
   been hot-plugged
```

If you have attached the volume at instance start-up, it cannot be detached while the instance is in the 'Running' state. To detach the volume, stop the instance first.

7.5 Delete Persistent Disks

To delete a persistent disk use the stratus-delete-volume command, note that you can delete only your disks.

To delete 'myprivate-disk' with UUID 9c5a2c03-8243-4a1b-a248-0f0d22d948c2

```
$ stratus-delete-volume 9c5a2c03-8243-4a1b-a248-0
   f0d22d948c2
DELETED 9c5a2c03-8243-4a1b-a248-0f0d22d948c2
```

Check that the disk is no longer there

```
$ stratus-describe-volumes
:: DISK a4324f26-39e0-4965-8c8f-3287cd0936e5
        created: 2011/07/20 16:37:10
        visibility: public
        tag: mypublic-disk
        owner: testor2
        users: 0
        size: 5
:: DISK d955fda6-bf9c-4aa8-abc4-5bbcdb83021b
        created: 2011/07/20 16:26:31
        visibility: public
        tag: mypublic-disk
        owner: testor1
        users: 1
        size: 5
```

Now try to delete 'mypublic-disk' of 'testor2' user persistent disk

```
$ stratus-delete-volume a4324f26-39e0-4965-8c8f-3287
   cd0936e5
  [ERROR] Service error: Not enough rights to delete disk
```

7.6 Read Only Volumes

In addition to persistent and volatile data volumes, StratusLab also supports read-only volumes. These come in handy when you have fixed

(or slowly changing) data sets that you would like to use with multiple machines.

Many scientific and engineering applications require access to fixed (or slowly changing) data sets. This includes versioned copies of databases (e.g. protein databases) or calibration information. These are often inputs to calculations that analyze larger, more varied data sets.

Persistent disk volumes and volatile disk volumes are not well-suited for this. The persistent disks cannot be shared between machine instances, so multiple copies of the disk need to be maintained or the data needs to be shared through a file server (e.g. NFS). Use of volatile storage would require copying the data each time a new machine instance starts.

Read-only disks allow users to create a fixed disk image containing the data. It is then registered in the Marketplace and can then be attached to multiple machine instances. This mechanism takes advantage of the caching and snapshotting infrastructure used for machine images. Making the initial copy of the data image and subsequent snapshotting for individual machine instances completely transparent to the user.

7.6.1 Create a Disk Image

Disk images must contain a formatted file system that can be read by the chosen operating system for your machine images. Although this could be any file system, for a couple of reasons the best choice is an ISO9660 (CDROM) image.

- These images are easy to create using the `mkisofs` utility (or variants) available in most operating systems.
- It can be universally read by all operating systems.
- It ensures that the operating system is aware that this is a read-only file system.
- Normally these volumes can be mounted and accessed by non-root users.

Using the `mkisofs` utility, create a disk is easy. The procedure is just two steps: 1) create a directory containing all of the data for the disk and 2) run `mkisofs` on this directory to create the CDROM image.

The only downside is that this requires enough disk space to hold the directory with the original data and also the created CDROM image. Using the persistent or volatile storage makes finding a big enough playground easy.

It is strongly recommended that you provide a label for the disk image. This will allow you to mount the image in a machine instance without having to know the hardware device name within the machine instance.

7.6.2 Registering the Image

Just like for machine images, the data images must be registered in the Marketplace. You need to create an image metadata entry and then upload it into the Marketplace.

Before anything else, you need to put the image on a webserver. The URL of the image will then be used in the 'location' element of the metadata.

Then you'll need to create the metadata itself. This can be done with the `stratus-build-metadata` command in the standard way. However, there are a few 'gotchas'. The OS, OS version, and OS architecture must be provided; just use 'none' for these values. You should gzip the image and use 'gz' for the compression. The 'format' of the image should be 'raw'. Lastly, the image should have a filename that ends with '.iso.gz' after the compression.

If the disk has a label (it does right?!), then you should add this information in the metadata description element.

Once you've created the metadata entry, sign it with `stratus-sign-metadata` and upload it into the Marketplace.

7.6.3 Using the Data Image

Using the image itself should be straight-forward. Use `stratus-run-instance` as you normally would but add the `--readonly-disk` option with the Marketplace identifier of the data image. An example is:

```
$ stratus-run-instance \
    --readonly-disk=GPAUQFkojP5dMQJNdJ4qD_62mCo \
    GJ5vp8gIxhZ1w1MQF16R6MIcNoq
```

This disk image is a standard image ("Flora and Fauna") used for tests of the system. It contains a hierarchy of files named after animals and plants.

Once the machine boots, you can use the command `blkid` to find the image. For this instance, the output looks like the following:

```
$ blkid
/dev/sda1: UUID="2fb85561-3fc4-4258-b3cf-abd8ae53d18a"
   TYPE="ext4"
/dev/sda5: UUID="ae3f7fb4-7d0e-4f6a-b91a-2f261be6a75a"
   TYPE="swap"
/dev/sr0: LABEL="_STRATUSLAB" TYPE="iso9660"
/dev/sdb: UUID="84e95f5f-dd31-4452-beca-2ab2cfd1bb87" TYPE
   ="swap"
/dev/sdc: LABEL="CDROM" TYPE="iso9660"
```

In this case, the disk we're looking for is `/dev/sdc`. The other CDROM image with a label '_STRATUSLAB' is the contextualization information.

Mount the data disk and look at the data:

```
$ mount /dev/sdc /mnt
mount: warning: /mnt seems to be mounted read-only.

$ ls -l /mnt
total 4
dr-xr-xr-x 1 root root 2048 Jan 26 20:01 animals
dr-xr-xr-x 1 root root 2048 Jan 26 20:01 plants

$ cat /mnt/animals/cat.txt
cat
```

If you create your disk with a label, then the device can be mounted without having to know the actual device ID. This makes it easier for automated scripts to mount the disk.

7.6.4 Summary

Use of read-only disks is convenient for sharing fixed datasets between multiple machine instances. Doing so reduces the size of customized machine images and decouples the updates of the machine images from the updates of the datasets. Because this takes advantage of

the extensive caching mechanisms of StratusLab, the transfer of such images and the creation of disks for each machine instance is completely transparent to the user.

7.7 Volatile Storage

In addition to persistent volumes and read only volumes, StratusLab also offers the possibility of volatile storage. These disks are allocated when a virtual machine starts and are destroyed automatically when the VM terminates. Because of the volatile nature of these disks, they are ideal for temporary storage.

Users can request a volatile disk when starting a virtual machine with the `--volatile-disk` option to `stratus-run-instance`, giving the required size of the disk. Note that this storage is allocated on the physical machine where the VM is running, so requesting a very large volatile disk may reduce the number of physical machines which can host the virtual machine.

As for the persistent volumes, these are raw devices, so the user is responsible for partitioning and/or formatting the volumes before using them. They also must be subsequently mounted on the VM's file system.

Again, these volumes are appropriate **only for temporary data storage**. The data on these volumes will be destroyed when the virtual machine is terminated.

Chapter 8

Image Management

Being able to fully customize the execution environment is one of the strongest attractions of a IaaS cloud. StratusLab provides tools to find existing customized virtual machine images and to create new ones if necessary.

8.1 Finding Available Images

Building new virtual machine images can be a tedious and time-consuming task. Your first instinct should be to first look to see if someone else has done the work for you!

The StratusLab Marketplace provides a registry for available, public virtual machine images. These are created by the people within the community as well as by StratusLab partners to help people get started using the cloud quickly.

The StratusLab partners themselves provide "base" images for ttylinux, CentOS (a RHEL derivative), Ubuntu, Debian, and OpenSuSE. These are minimal, but functional, installations of these operating systems. They can be used directly or customized to create personalized machines. These can be found in the Marketplace by looking for "hudson.builder" as the endorser of the images.

The Marketplace also contains images for various scientific disciplines. IGE has prepared images for testing Globus services and for running tutorials with the services. IBCP has created appliances with numerous

bioinformatics applications already installed. Search the Marketplace to find appliances that interest you.

8.2 Building New Images from Existing Images

Although there are many images in the Marketplace, sometimes a suitable image cannot be found. In this case, try to find a suitable appliance as a starting point and create a new appliance by customizing the initial one.

8.2.1 Automated Process

StratusLab provides the `stratus-create-image` command to automate the production of a new image based on an existing one. (NOTE: This command does not work on Windows. See manual process section.) This takes three inputs:

- The Marketplace identifier of the starting image,
- A list of additional packages to install, and
- A script to configuring the image.

In addition, some information about the new information will be required– such as a description, the author, and the author's email address.

As an example, let's use the StratusLab Ubuntu base image, adding an Apache web server, and customizing the home page. To do this, we will need to add the "apache2" and "chkconfig" packages to the image. We will also need to run a script that modifies the server's home page.

First, create a script `setup-ubuntu.sh` that contains the following commands:

```
#!/bin/bash

#
# Workaround to ensure old networking information isn't␣
   cached
#
```

```
rm -f /lib/udev/rules.d/*net-gen*
rm -f /etc/udev/rules.d/*net.rules

#
# Modify the web server's home page.
#
cat > /var/www/cloud.txt <<EOF
Cloudy Weather Expected
EOF
```

This will modify the server's home page. When we eventually start the modified image, we can use this to ensure that the modifications have been correctly made.

Now use the `stratus-create-image` command to create the new image:

```
$ stratus-create-image \
  -s setup-ubuntu.sh \
  -a apache2,chkconfig \
  --type m1.xlarge \
  --comment "ubuntu create image test" \
  --author "Joe Builder" \
  --author-email builder@example.org \
  HZTKYZgX7XzSokCHMB60lS0wsiv
```

Note that the necessary packages are included and the configuration script has been referenced. In addition, information about the author and the new image has been provided. The argument is the Marketplace identifier of the image to start with; in this case, it is a base Ubuntu image.

Warning: Be sure to provide a correct email address. The results of the process will be sent to that address!

Running this command will produce output like the following:

```
 ::::::::::::::::::::::::::::
 :: Starting image creation ::
 ::::::::::::::::::::::::::::
 :: Checking that base image exists
 :: Retrieving image manifest
 :: Starting base image
  [WARNING] Image availability check is disabled.
```

```
 ::::::::::::::::::::::::
 :: Starting machine(s) ::
 ::::::::::::::::::::::::
 :: Starting 1 machine
 :: Machine 1 (vm ID: 1655)
 Public ip: 134.158.75.239
 :: Done!
 :: Waiting for machine to boot
............
 :: Waiting for machine network to start
....
 :: Check if we can connect to the machine
 :: Executing user prerecipe
 :: Installing user packages
 :: Executing user recipe
 :: Executing user scripts
Connection to 134.158.75.239 closed.

 ::::::::::::::::::::::::::::::::::::::::
 :: Finished building image increment. ::
 ::::::::::::::::::::::::::::::::::::::::

 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

 :: Please check builder@example.org for new image ID and
    instruction. ::
 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

 :: Shutting down machine
```

At this point if you check the running machines, you'll see something like this:

```
$ stratus-describe-instance
id   state     vcpu memory    cpu% host/ip
                    name
1655 Epilog    4    0         0    vm-239.lal.stratuslab.
   eu creator: 2012-12-04T07:58:25Z
```

For a normal machine, the "Epilog" state flashes by very quickly because it just deletes the virtual machine's resources. In this case however, the "Epilog" process actually saves the modified image to a new volume in

the persistent disk service. Because these are generally multi-gigabyte files, this process can take several minutes.

At the end of the "Epilog" process, an email will be sent to the user with a subject like "New image created IOeo3R5qEdCas5j_r1HxVne3JMk". The body of the email contains:

- The location of the created image,
- The identifier of the created image, and
- A draft metadata entry for the new image.

There will also be a temporary entry created in the Marketplace to allow private testing of the image after creation. You can search for the image identifier to find the metadata entry.

You can also find the created disk by searching the persistent disk service:

```
$ stratus-describe-volumes
:: DISK 410b7fb4-973b-4b6d-82a7-e637a5103f4d
   count: 0
   tag:
   owner: builder
   identifier: IOeo3R5qEdCas5j_r1HxVne3JMk
   size: 6
```

Now we will try to deploy the new machine and verify that the web service responds. Ubuntu takes several minutes to go through the full boot process and to start the web service, so a little patience is required.

```
$ stratus-run-instance --type c1.medium
   IOeo3R5qEdCas5j_r1HxVne3JMk

 ::::::::::::::::::::::::
 :: Starting machine(s) ::
 ::::::::::::::::::::::::
 :: Starting 1 machine
 :: Machine 1 (vm ID: 1657)
 Public ip: 134.158.75.58
 :: Done!

$ # after waiting a few minutes...
```

```
$ curl http://vm-58.lal.stratuslab.eu/cloud.txt
Cloudy Weather Expected
```

After testing the image, you'll need to take a few more steps to make the image accessible for more than 2 days or to make it public.

To make the image public, the contents will need to be copied to a public server. Mount the define on a VM and copy the contents to a suitable location. (A future version will allow you to expose the disk contents directly without copying them.) For a private disk, you do not need to make any copies.

In both cases, modify the draft image metadata, especially providing a longer validity period for the image. A resonable value is 6 months. Sign the metadata with `stratus-sign-metadata` and upload it to the Marketplace with `stratus-upload-metadata` or via the web interface.

8.2.2 Manual Process

The `stratus-create-image` automates the interactions with the new machine, but you may want to make modifications by hand (or be working on Windows).

To repeat the above exercise with the manual process, start with the command `stratus-run-instance`:

```
$ stratus-run-instance \
  --save \
  --type m1.xlarge \
  --comment "manual␣ubuntu␣test␣image" \
  --author "Joe␣Builder" \
  --author-email builder@example.org \
  --image-version=2.0 \
  HZTKYZgX7XzSokCHMB60lS0wsiv

  [WARNING] Image availability check is disabled.

 ::::::::::::::::::::::::::
 :: Starting machine(s) ::
 ::::::::::::::::::::::::::
 :: Starting 1 machine
 :: Machine 1 (vm ID: 1659)
```

```
 Public ip: 134.158.75.62
 :: Done!
```

The important option is the **–save** option. This will trigger the copy of the image to a persistent disk at when the machine is shutdown.

Once the machine is accessible via SSH, log into the machine and execute the following commands:

```
$ rm -f /lib/udev/rules.d/*net-gen*
$ rm -f /etc/udev/rules.d/*net.rules
$ apt-get install -y apache2 chkconfig
$ cat > /var/www/cloud.txt
Cloudy Weather Expected
```

See the previous section for information about what these commands do.

After all of the modifications have been made, log out of the machine. **Use the command `stratus-shutdown-instance` to stop the machine.** If you use `stratus-kill-instance` the changes you've made will be lost.

```
$ stratus-shutdown-instance 1659
$ stratus-describe-instance
id   state     vcpu memory    cpu% host/ip
                  name
1659 Shutdown  4    8388608   7    vm-62.lal.stratuslab.eu
    creator: 2012-12-04T09:16:35Z

$ stratus-describe-instance
id   state     vcpu memory    cpu% host/ip
                  name
1659 Epilog    4    8388608   7    vm-62.lal.stratuslab.eu
    creator: 2012-12-04T09:16:35Z
```

The machine will entry the "Shutdown" then the "Epilog" states. The image is copied during the "Epilog" state. When completed, you will receive an email with the image metadata.

You can check that the image functions correctly:

```
$ stratus-run-instance --type c1.medium J-
   zVxEV5vfFscoKPLOHjtubmrJF

 :::::::::::::::::::::::::
```

```
 :: Starting machine(s) ::
 ::::::::::::::::::::::::
 :: Starting 1 machine
 :: Machine 1 (vm ID: 1664)
 Public ip: 134.158.75.70
 :: Done!

$ # after waiting a few minutes...

$ curl http://vm-70.lal.stratuslab.eu/cloud.txt
Cloudy Weather Expected
```

Follow the instructions in the previous section to make this a public image.

8.3 Building Images from Scratch

Sometimes a suitable starting image cannot be found and building an image from scratch is required. Usually it is easiest to build new images with desktop virtualization solutions. The results must be converted to a format suitable for KVM.

Generating an image from scratch can be tedious and there are lots of pitfalls along the way. Keep in mind the following points:

- Images must support the StratusLab contextualization scheme
- Ensure DHCP network configuration (and turn off udev persistent net rules)
- All private information (keys, passwords, etc.) must be removed
- Remote access must only be via SSH keys, not by password
- Activate firewall blocking all unused ports
- Minimize installed software and services

As there are many places to run into problems, you're advised to contact the StratusLab support before starting.

8.4 Contextualization

Contextualization allows a virtual machine instance to learn about its cloud environment (the 'context') and to configure itself to run correctly there. StratusLab now supports CloudInit contextualization in addition to the OpenNebula and HEPiX contextualization schemes.

8.4.1 OpenNebula and HEPiX Contextualization

To be done.

8.4.2 CloudInit

The CloudInit mechanism is gaining traction within the cloud ecosystem and is moving towards becoming a de facto standard for the process. Because it handles both web-server and disk based contextualization mechanisms it is fairly straightforward for cloud software developers to implement.

For the appliance developers it provides a convenient modular framework for allowing both system and user-level service configuration. Being written in python with OS packages for most systems, makes it easy for those developers to include and use the software.

Ubuntu provides good documentation for CloudInit. The software itself can be downloaded from launchpad or installed from standard repositories for your operating system (e.g. EPEL for CentOS).

8.4.3 CloudInit-Enabled Images

All of the latest versions of the base virtual machine images maintained by StratusLab now support CloudInit. Using one of those images is the easiest way to see how CloudInit works. However, you can build your own image with CloudInit support; see the appendix of this document for instructions.

8.4.4 Web Server Example

To show how users can pass CloudInit information to the appliance, we will work through an example with the latest CentOS base image. We will use a script to install and configure a web server on the example machine. This script is generated by the user, sent via the `stratus-run-instance` command and then executed in the machine instance by the CloudInit contextualization.

The script that we want to execute on the machine instance is the following:

```
#!/bin/bash -x

yum install -y httpd

cat > /var/www/html/test.txt <<EOF
SUCCESSFUL TEST
EOF

chkconfig httpd on

service httpd start
```

This installs, configures, and starts a web server on the machine. We've named this script `run-http.sh`. Create this script on the machine where you've installed the StratusLab client commands.

The context information must be defined and passed to the virtual machine when starting it. The context is defined by a set of mimetype,file pairs:

```
ssh,$HOME/.ssh/id_rsa.pub
x-shellscript,run-http.sh
```

For ssh keys, use the pseudo-mimetype of 'ssh'. A single file can be included literally (instead of being embedded in a multipart message) with a pseudo-mimetype of 'none'.

This information can be passed directly to `stratus-run-instance` with the `--cloud-init` option:

```
$ stratus-run-instance \
     --cloud-init \
```

```
    'ssh,$HOME/.ssh/id_rsa.pub#x-shellscript,run-http.sh'
        \
    ... other options ...
```

Note that in this case, multiple pairs are separated by a hash sign. You can also create a `cloud-init.txt` file containing the context information:

```
$ stratus-prepare-context \
    ssh,$HOME/.ssh/id_rsa.pub \
    x-shellscript,run-http.sh
```

and then pass this to the `stratus-run-instance` command with the `--context-file` option:

```
$ stratus-run-instance --context-file cloud-init.txt
```

The first option is generally the most convenient option unless further (non-cloud-init) options need to be passed to the virtual machine.

The context can contain multiple public ssh keys and multiple scripts or other files. See the CloudInit documentation for what is permitted for 'user data'.

Warning: No ssh keys are included by default. You must specify explicitly any keys that you want to include.

Warning: Be sure that the contextualization information you are passing can be used by the CloudInit version within the appliance itself. **Be particularly careful because multipart inputs do not work if the python version in the virtual machine is less than 2.7.3.**

This is the case with the CentOS image used here, so we'll include the script literally in the user data with context information:

```
ssh,$HOME/.ssh/id_rsa.pub
none,run-http.sh
```

Note the use of the 'none' pseudo-mimetype. If 'none' is used, only the last file marked as 'none' will be included in the user data; it will be included literally, that is without encapsulating it in a multipart form.

If you use the `stratus-prepare-context` command, you can see what key-value pairs are passed to the virtual machine. The `cloud-init.txt` will look like:

```
CONTEXT_METHOD=cloud-init
CLOUD_INIT_AUTHORIZED_KEYS=c3NoLXJzYS...
CLOUD_INIT_USER_DATA=H4sIAGHZzV...
```

The last two parameters contain base64 encoded representations of the ssh keys and the `run-http.sh` script.

The machine can then be started with the command:

```
$ stratus-run-instance \
    --cloud-init \
    ssh,$HOME/.ssh/id_rsa.pub'#none,run-http.sh' \
    IRei7LKvxoWVRsiiup2cz3-sSsk
```

After this machine starts it should be possible to see the configured file in the appliance's web server:

```
$ curl http://vm.example.org/test.txt
SUCCESSFUL TEST
```

Of course you need to change the node name to point to the machine instance that you've started.

8.4.5 Future Work

StratusLab support for CloudInit should make it easier for users to create appliances that will run on StratusLab as well as on other clouds using an implementation compatible with CloudInit. This will be an important gain for users as they move to a federated cloud environment.

This preview support for CloudInit demonstrates the utility of this contextualization method. The collaboration is interested in hearing your feedback on the implementation so that we can improve it in upcoming releases. This will likely become the default contextualization method in a future release.

8.4.6 Building an Image with CloudInit

The standard StratusLab commands for creating appliances (e.g. `stratus-create-image`) can be used to create an appliance using CloudInit. See the image creation document for details on the commands.

The following script can be fed to `stratus-create-image` to create an example appliance with CloudInit support. Note that all of the recent base images maintained by StratusLab already contain CloudInit support.

```
#!/bin/bash -x

#
# Turn off udev caching of network information.
#
rm -f /lib/udev/rules.d/*net-gen*
rm -f /etc/udev/rules.d/*net.rules

#
# Configure the machine for the EPEL repository.  The
   CloudInit
# package is available from there.
#
wget -nd http://fr2.rpmfind.net/linux/epel/6/i386/epel-
   release-6-8.noarch.rpm
yum install -y epel-release-6-8.noarch.rpm

#
# Upgrade all package on the machine and install cloud-
   init.
yum clean all
yum upgrade -y
yum install -y cloud-init

#
# Change configuration to allow root access.  Signal that
   the 'user'
# account should also be configured.
#
sed -i 's/user: ec2-user/user: root/' /etc/cloud/cloud.cfg
sed -i 's/disable_root: 1/disable_root: 0/' /etc/cloud/
   cloud.cfg
```

This script was named `create-cloud-init-appliance.sh` for the command to create the machine:

```
$ stratus-create-image \
    -s create-cloud-init-appliance.sh \
```

```
    --type c1.medium \
    --title 'example title' \
    --comment 'example comment' \
    --author 'Jane Creator' \
    --author-email 'jane@example.org' \
    Jd3yRF6x4ofxfCeVK6BmCkuHc0m
```

The image ID Jd3yRF6x4ofxfCeVK6BmCkuHc0m refers to a standard CentOS 6.2 machine without CloudInit support. The configuration commands and the package to be installed will depend on what base operating system you choose.

The image generated with the above procedure will work with the CloudInit tutorial described above.

8.5 Converting a VirtualBox Image

If you used VirtualBox to create a VM, you can convert it to a raw images to run on StratusLab. The easiest way to make your VM compatible with StratusLab is to convert your vdi disk into raw disk. This can be done with standard VirtualBox tools with the following command:

```
$ VBoxManage internalcommands converttoraw mydisk.vdi
   mydisk.img
$ gzip mydisk.img
```

Now, you ve got a img.gz, you can put this in a public location and register the image in the Marketplace as usual.

Chapter 9

Programmatic Access

Application developers often desire to have programmatic access to cloud services. This allows them to easily integrate those services into existing applications and analysis frameworks.

StratusLab provides several mechanisms by which application developers can access its services. All services provide REST interfaces, making them easily accessible from all languages via standard HTTP(S) client libraries. The collaboration also provides the python APIs that are used internally by the command line interface and a Libcloud plug-in.

9.1 REST Interfaces

Services with a resource-oriented architecture expose a set of resources or objects via well defined URLs, standard CRUD (create, read, update, and delete) operations are then supported for those resources. Although not required, nearly all such services use the HTTP(S) protocol and reuse the standard HTTP actions for implementing the CRUD functionality.

Using the standard HTTP protocol and having a well defined hierarchy of URLs representing resources, makes access easy from any programming language with a HTTP client library. Because of this universal accessibility and the intuitive mapping between resources and URLs, StratusLab using a resource-oriented architecture for all of its services.

However, because of significant underlying changes in the StratusLab service implementations at this time, it is recommended that application

developers either use the Libcloud API or script the command line interface. This will protect them from the significant, incompatible changes in the APIs that are currently taking place.

9.1.1 StratusLab API

As mentioned above, the REST APIs and the corresponding StratusLab python API are undergoing significant changes at the moment. Application developers should avoid these interfaces if possible. Contact the StratusLab support for more information about the current status.

9.1.2 CIMI API

The CIMI API, a standard REST API for IaaS clouds, will become the standard interface for all of the StratusLab services. Application developers planning to port their software to the StratusLab cloud should target this API. This API should appear over the next couple of StratusLab releases (i.e. 3-6 months).

9.2 Libcloud API

The Libcloud API provides a generic python interface for controlling cloud resources from multiple providers. StratusLab now provides a Libcloud compute driver allowing users to access StratusLab cloud infrastructures via this API.

Application developers who desire a stable interface should prefer this API. It is available now and will remain stable despite the underlying changes to the StratusLab service implementations.

9.2.1 Installing the Driver

The driver is intended to be installed with pip. You should be able to simply do the following:

```
pip install stratuslab-libcloud-drivers
```

which will install the StratusLab Libcloud driver, Libcloud itself (0.12.1), and the StratusLab client. If you want to use the `deploy_node()` function, you'll also need to install paramiko, a python SSH library, as well.

```
pip install paramiko
```

If pip is configured to do system-wide installations, then the PYTHONPATH and PATH should already be set correctly. If it is setup for user area installations, you will likely need to set these variables by hand.

You can download the package directly from PyPi. The name of the package is "stratuslab-libcloud-drivers". You will also need to download and install all of the dependencies. **Using pip is very strongly recommended.**

9.2.2 Configuring the StratusLab Client

The Libcloud drivers for StratusLab use the same configuration file as for the command line client.

To copy a reference configuration file into place, use the command `stratus-copy-config` to copy an example configuration file into place. The command will print the location of your configuration file. The example configuration file contains extensive documentation on the parameters. Edit the file and put in your credentials and cloud endpoints.

More detailed documentation can be found in the StratusLab documentation area on the website.

9.2.3 Using the Driver

Once you've downloaded, installed, and configured the necessary dependencies, you are ready to start using the driver.

From the Python interactive shell do the following:

```
from libcloud.compute.providers import set_driver

set_driver('STRATUSLAB',
           'stratuslab.libcloud.compute_driver',
           'StratusLabNodeDriver')
```

This registers the driver with the Libcloud library. This import must be done **before** asking Libcloud to use the driver! Once this is done, then the driver can be used like any other Libcloud driver.

```
# Obtain an instance of the StratusLab driver.
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver
StratusLabDriver = get_driver('stratuslab')
driver = StratusLabDriver('default')

# Use the Libcloud methods to find, create and control
   nodes.
nodes = driver.list_nodes()
```

There are a couple examples in the test area of the GitHub repository for this driver lc-sl-examples. You can also find general information on the Apache Libcloud website.

9.2.4 Driver Status

The driver is functionally complete and should work with all of the standard Libcloud workflows. Problems encountered when using the driver should be reported via the StratusLab support mailing list.

In detail, the following functions have working implementations:

- `list_images`: list all valid images in Marketplace
- `list_locations`: list of sections in configuration file
- `list_sizes`: list of standard machine instance types
- `create_node`: start a virtual machine
- `deploy_node`: start a VM and run a script (see notes below)
- `destroy_node`: terminate a virtual machine
- `list_nodes`: list of active virtual machines
- `create_volume`: create persistent disk
- `destroy_volume`: destroy a persistent disk

- `attach_volume`: attach a volume to node
- `detach_volume`: remove a volume from a node

The `list_volumes` function is specific to the StratusLab driver and is not part of the Libcloud standard abstraction.

The `reboot_node` function will not be implemented as the required functionality is not provided by a StratusLab cloud.

Notes for `deploy_node`:

1. The SSH library used by Libcloud seems to only work correctly with DSA SSH keys. You can have both RSA and DSA keys available in parallel.
2. This function uses sftp to transfer the script between the client and the virtual machine. Consequently, SSH implementations that do not support sftp will not work. This include, notably, ttylinux.

Appendix A

Python and Python Utilities

The StratusLab command line client and other tools are written Python and are packaged to take advantage of the Python installation tools.

An environment with an acceptable version of Python must be used. In addition, it is strongly recommended that `pip` and `virtualenv` be used to manage the installation and configuration of the StratusLab tools.

A.1 Python

All of the StratusLab python code requires a recent version of Python 2, version 2.6 or later. The StratusLab code is **not** compatible with Python 3.

A.1.1 Linux Operating Systems

Python packages are a standard part of all Linux distributions and can be installed via the distribution's standard package management tools.

Once installing Python verify that a version compatible with StratusLab has been installed:

```
$ python --version
Python 2.6.6
```

If the version is not a recent Python 2 version (2.6+), then you must install a separate version that is compatible with StratusLab and modify

your environment. You will find the `virtualenv` section below useful in this case.

A.1.2 Mac OS X

All recent versions of Mac OS X provide a version of Python that works with StratusLab. No special configuration is needed for Python itself.

A.1.3 Windows

Python is not a standard part of the Windows distributions. Consequently, you must install and configure Python yourself. You can find installation packages on the python.org website. Be sure to select an installer for Python 2 and **not** Python 3.

A.2 Pip

Pip is a tool for installing and managing Python packages. It allows packages to be installed, listed, queried, upgraded, and uninstalled. It uses information in the downloaded packages to automatically resolve dependencies.

If pip is not already installed on your system, then you can find out how to download and install it on the pip-installer.org website. This website also contains information on how to use it.

StratusLab **strongly recommends using pip** to manage the installation of the its Python-based tools. The `easy_install` command can also be used, but pip provides better management capabilities.

A.3 Virtualenv

The virtualenv package allows users to create customized virtual Python environments. This avoids having to modify the global system environment, permits use of different Python versions, and avoids potential dependency conflicts between installed packages.

The virtualenv page in pypi provides complete information on installing and using virtualenv.

StratusLab recommends using virtualenv, especially if you are on a machine with a shared environment that you cannot easily change.

Appendix B

SSH Client

Nearly always, virtual machine instances are accessed remotely via an SSH (secure shell) session. This requires that the user has an SSH client installed and has generated an SSH key pair.

B.1 Linux Operating Systems

B.1.1 Client Installation

An SSH client, usually OpenSSH, comes as a standard part of all Linux operating systems. It is normally installed by default, but can be installed via the distribution's package management system if necessary.

B.1.2 Creating an SSH Key Pair

To use the client to connect to virtual machine instances, the user must have an SSH key pair consisting of a "public" key and a "private" key. These keys are usually located in the `~/.ssh/` directory and have names like `id_rsa` (private) and `id_rsa.pub` (public).

You can generate them with the following command

```
$ ssh-keygen
```

The default values are appropriate in most cases, but you should provide a passphrase and not leave it empty.

Verify the generated key pair permissions. The `id_rsa` should have permissions 0600 (read/write access for owner only) and the `id_rsa.pub` should have permissions 0644 (read access for all; write access for owner).

Be sure to remember the passphrase that you have used! This passphrase can (and usually is) different from the password for the user's account.

B.1.3 SSH Agent

SSH agents allow users to provide the passphrase once per session, caching the passphrase and providing it automatically after the first request. This makes use of SSH more convenient when multiple connections are being made to a virtual machine.

Some operating systems start an SSH agent automatically when a user logs in. If this is the case, be sure that the agent uses the correct key and the correct password for that key.

You can check if an SSH agent is running by looking at the `SSH_AGENT_PID` variable.

```
$ printenv SSH_AGENT_PID
```

If this isn't empty, then the agent is running. You can add your key to the agent with the command:

```
$ ssh-add
```

Providing the passphrases for your keys when prompted for them. See the manpages for `ssh-agent` and `ssh-add` for more information.

B.2 Mac OS X

B.2.1 Client Installation

The SSH client is a standard part of Mac OS X. No installation is necessary.

B.2.2 Generating an SSH Key Pair

The commands and procedure are the same as for the Linux operating systems. Follow the instructions there.

B.2.3 SSH Agent

An SSH agent is started automatically when logging into the machine. It will automatically ask for your passphrase the first time and then remember it for all future sessions.

B.3 Windows

B.3.1 Client Installation

Windows does not ship with an SSH client, so you must install one. Although there are some other solutions (especially for recent versions of Windows), most people install and use PuTTY. Binaries and installation instructions can be found on the PuTTY website.

It is recommended to install the full PuTTY distribution, but at a minimum the `putty` and `puttygen` executables need to be available.

B.3.2 Generating an SSH Key Pair

To use PuTTY with the cloud machines, you must generate a certificate. The most recent version of PuTTY allows you to do this on your machine, using the executable PuTTYGen.

In the PuTTYGen interface do the following:

- Click “generate”.
- Provide key passphrase if you want
- Click “save public key” to save as file (e.g. in the .ssh folder)
- Click “save private key” to save as file (e.g. in the .ssh folder)

- Copy the text in the "Public key for pasting into OpenSSH..." box to the clipboard
- Save this text in the file $HOME/.ssh/id_rsa.pub as a **plain text file**

B.3.3 Logging into a VM with PuTTY

To log in your virtual machine using PuTTY:

1. Start PuTTY,
2. In the "session" category provide the hostname or IP address
3. In Connection/SSH/Auth category, in "Private key for authentication" field, browse to your private key.
4. Open

Be sure to login with the correct username for the virtual machine; this is nearly always "root".

If you are using X11 for graphical interfaces, you must also check the following:

- Connection/SSH/Auth panel: click "Allow agent forwarding" and select the private key file you saved above
- Connection/SSH/X11 panel: click "Enable X11 forwarding"

The X11 server on your machine must be started before making the connection to the virtual machine.

More information on how to "Connecting to Linux/UNIX Instances from Windows Using PuTTY" can be found in the Amazon documentation.

B.3.4 SSH Agent

PuTTY supports the SSH agent functionality through the Pageant executable.

Appendix C

Java

Although Python is the primary programming language used by the StratusLab client, a small portion uses Java–specifically the portions related to the creation, validation, and signing of metadata for the Marketplace.

For these features, an installation of a certified Java 1.7 or later distribution is required.

C.1 Linux Operating Systems

Most Linux distributions contain an acceptable set of Java packages–usually OpenJDK. These can be installed with the distribution's standard package management tools.

Oracle's Java distributions can also be used. Instructions for obtaining and installing these distributions are available from the Java website.

The Java compiler created from GNU (GJC) will **not** work. If this is installed, either remove it from your system or ensure that it is not the default version of Java on your system.

C.2 Mac OS X

Oracle provides a distribution of Java for recent versions of Mac OS X. You can download and install this version from the primary Java website.

Older versions of Mac OS X also incorporate their own version of java that will work with the StratusLab client, although the more recent Oracle releases are preferred.

C.3 Windows

Download and install Java from the primary Java website.

Appendix D

Windows PowerShell

In recent versions of Windows, there are two command line interfaces: the traditional “cmd” shell and the newer “PowerShell”.

If available, PowerShell is recommended. It can be started directly from the graphical interface or indirectly from the “cmd” shell by typing the command `powershell`.

For security reasons, PowerShell does not allow scripts to be executed by default. As the StratusLab command line interface consists of Python scripts, these cannot be executed without changing the default configuration.

To allow PowerShell to run scripts, execute the following within the PowerShell:

```
Set-ExecutionPolicy Unrestricted -Scope CurrentUser
```

The shell does not need to be restarted. This configuration change only needs to be done once.