

---

# **StratusLab Documentation**

***Release 14.06.0-RC6-SNAPSHOT***

**StratusLab Collaboration**

May 06, 2015



<b>1</b>	<b>Tutorials</b>	<b>3</b>
1.1	User Tutorial . . . . .	3
1.2	Installation Tutorial . . . . .	14
<b>2</b>	<b>Guides</b>	<b>29</b>
2.1	User Guide . . . . .	29
2.2	Administrator Guide . . . . .	66
2.3	Contributor Guide . . . . .	100
<b>3</b>	<b>Search</b>	<b>107</b>



StratusLab is a complete cloud distribution that allows you to install an “Infrastructure as a Service” cloud on your own hardware. The StratusLab distribution aims to be simple to use, simple to install, and simple to maintain.

If you can’t find the information you need in the documentation, contact the developers through [GitHub](#) or our [support mailing list](#).



### 1.1 User Tutorial

This tutorial provides a whirlwind tour of the features available to users of StratusLab cloud infrastructures. It covers the installation of the command line client as well as use of the cloud's compute, storage, and network resources.

#### 1.1.1 Prerequisites

To follow this tutorial, you must have a machine running a recent version of Linux, Mac OS X, or Windows and the ability to install and configure software on that machine.

Linux and Mac OS X are fully supported and tested. The core features work also on Windows, although there are a few commands that do not. Those commands that do not work on Windows are pointed out in the documentation.

To take advantage of all of the StratusLab features, you must have the following software installed and configured:

- Python 2 (2.6+)
- `pip`, the python package installer
- Java 1.6+ (any certified distribution)
- SSH client (with an SSH keypair)
- Recent web browser

You may optionally install the python `virtualenv` to isolate the StratusLab python environment from others. All of the supported platforms have widely available versions of these software packages available.

---

**Note:** The commands that require the Java Virtual Machine will not be used in this tutorial, so you do not need to install Java for this tutorial.

---

#### 1.1.2 Command Line Client

The StratusLab command line client is written primarily in Python, although there is a small part related to XML cryptographic signatures that is written in Java.

## Installation

End-users should use the Python package installer (`pip`) to install the client on their machines. Do the following to install the client and all of the required dependencies:

```
$ pip install stratuslab-client
Downloading/unpacking stratuslab-client
...
Successfully installed stratuslab-client dirq httpplib2 requests
Cleaning up...
```

The command should end with a message saying that the installation was successful. By default, `pip` will install the software in the system area to make it available for all users on the system.

---

**Note:** If you want to install a beta version or release candidate, then for newer versions of `pip` you will have to use the `--pre` option to allow pre-release versions to be installed.

---

**Note:** For CentOS, the client can also be installed via the yum package repository. The RPM and `pip` installations conflict with one another, so only one of these methods should be used.

---

**Warning:** On Mac OS X with the latest version of `pip`, the system-wide installation will not work correctly because of a change in where data files are placed. A user-level or `virtualenv` installation is recommended to work around this problem.

If you don't want a system-wide installation of the client, you can do a user-level installation using the option `--user`. However in this case, you will need to modify your environment, adding for example, the appropriate directories to your `PATH`:

```
PATH=$PATH:$HOME/.local/bin
```

Adjust the command appropriately for your operating system and command shell.

## Configuration

You must configure the command line client, providing the service endpoints for the StratusLab cloud that you want to use and your credentials for that cloud.

Create your configuration file by running the following command:

```
$ stratus-copy-config
```

If successful, this will create the file `$HOME/.stratuslab/stratuslab-user.cfg`. You will now need to edit this file to provide the required information. The file itself is extensively commented, showing all of the available options.

Your edited file should be similar to the following:

```
[default]

selected_section = sl-cloud

endpoint_timeout = 5

[sl-cloud]

name = "StratusLab Cloud"
```



```
country = "France"

endpoint = https://cloud.lal.stratuslab.eu/one-proxy/xmlrpc
pdisk_endpoint = https://pdisk.lal.stratuslab.eu/pdisk
marketplace_endpoint = https://marketplace.stratuslab.eu/marketplace

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

Obviously, you will need to replace the username and password values with your actual credentials. You will also need to replace the endpoint values for the cloud you are using. The endpoints given here are for the StratusLab reference infrastructure.

---

**Note:** If your password (or any other value) contains a percent sign (%), you **must** double that character in the configuration file. This is because the Python software used to read the configuration files allows variable substitution with the syntax `%(variable)`.

---

## Testing

To test that the client is installed correctly and that you can contact the cloud services, run the following commands:

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

$ stratus-describe-volumes
No disk to show
```

These should return an empty list of virtual machines and of volumes, respectively. Any error indicates a problem with the installation or configuration of the client. Correct these errors before continuing!

## Getting Help

To see the help for a given command, do the following:

```
$ stratus-describe-instance --help
Usage: stratus-describe-instance [options] [vm-id ...]

Provides information about the virtual machine with the given
identifiers or all virtual machine if no identifier is given.
...
```

All of the StratusLab commands support this option. It provides a summary of the purpose of the command and a detailed list of the available options.

All of the commands also support the `--version` option that prints the version number of the client. When reporting problems, it is very helpful to also provide the exact version number of the client.

In general the command line interface returns a minimum of information to the user. To make the commands more verbose (especially when tracking down errors), you can add the `-v` or `--verbose` option. This can be specified multiple times to increase the verbosity further. All commands support this option.

### 1.1.3 Compute Resources

“Infrastructure as a Service” cloud infrastructures provide computing resources through “virtual machines”. For end users, these virtual machines act just like physical machines, except that they can be provisioned quickly with exactly

the configuration desired.

This section starts with a description of the Marketplace, a catalog of virtual machine images, and then covers how to start, access, and stop virtual machine instances.

## Appliance Marketplace

All virtual machine instances are created from virtual machine images (or appliances) that contain the starting configuration for the virtual machine.

For StratusLab clouds, the Marketplace contains a database of all of the available appliances. These include standard appliances created and maintained by the StratusLab developers as well as appliances created by other StratusLab users.

Use your web browser to view the available appliances by navigating to the central [Marketplace](#). You can find the standard images from StratusLab by putting “images@stratuslab.eu” into the “endorser” field on the right side of the interface.

The summary contains the basic information for an appliance along with a link to find more information. The most important bit of information is the **image identifier**. This is a 27 character Base 64 value that uniquely identifies the appliance. This identifier is used to select the appliance you want to run as a virtual machine.

StratusLab provides minimal distributions of CentOS 6, ScientificLinux 6, Ubuntu 12.04, and Ubuntu 14.04. These follow the best practices for appliance creation and are kept up to date with security patches. They make good starting points for creating your own appliances.

---

### Exercise

Use your web browser to navigate through the Marketplace. Use the search widget on the right to see what appliances are available. Use the search widget to find all of the appliances created by the StratusLab collaboration.

---

## Virtual Machine Lifecycle

The virtual machine lifecycle consists of the following commands:

```
$ stratus-run-instance ${APP_ID} # APP_ID from the Marketplace
                                # returns VM_ID and VM_IP
$ stratus-describe-instance ${VM_ID}

# Connect to the deployed virtual machine
$ stratus-connect-instance ${VM_ID}
$ ssh root@${VM_IP}

$ stratus-kill-instance ${VM_ID}
```

These commands start, list, access, and destroy a virtual machine, respectively.

The `stratus-run-instance` command starts a new virtual machine instance. It requires the appliance or image identifier from the Marketplace (`APP_ID`). This will return the virtual machine identifier (`VM_ID`) and its assigned IP address (`VM_IP`).

The `stratus-describe-instance` command displays the information about virtual machines. Without an argument, it will provide a concise summary for all non-terminated virtual machines. If you provide a virtual machine identifier, then only information for that machine will be returned. More detailed information can be obtained by using the `--verbose` option.

You can log into your virtual machine using SSH. Your public SSH key will have been transferred to the machine and will allow you to log into the machine as `root`. Use `stratus-connect-instance ${VM_ID}` or just `ssh`

`root@${VM_IP}` to log into the machine. Depending on the operating system, resources, etc., it can take several minutes to have access to a virtual machine.

---

**Note:** The `stratus-connect-instance` command does not work on Windows. Use your local SSH command or application to connect to your virtual machine.

---

The `stratus-kill-instance` command stops a virtual machine and releases all of the allocated resources. Virtual machines that have been terminated will no longer appear in the `stratus-describe-instance` output by default.

**Warning:** The `stratus-kill-instance` stops a running virtual machine instantly. This is the equivalent of pulling out the plug. A more graceful shutdown should be done in most cases: halt the virtual machine (with `halt` or `shutdown` within the machine) and then run `stratus-kill-instance`.

---

### Exercise

Choose one of the standard appliances (e.g. Ubuntu 14.04) and use these commands to go through the lifecycle for a virtual machine. How long was it before you could `ping` the virtual machine? How long before you could log in?

---

## Allocated Resources

The resources allocated to virtual machines can be defined with options *at deployment time*. StratusLab does not allow the resource allocations for existing virtual machines to be changed.

A set of predefined “machine types” provide some default configurations.

You can see the predefined configurations with the command:

```
$ stratus-run-instance --list-types
Type      CPU      RAM      SWAP
cl.medium  2 CPU    1536 MB  1536 MB
cl.xlarge  4 CPU    6144 MB  6144 MB
ml.large   2 CPU    6144 MB  6144 MB
ml.medium  1 CPU    3072 MB  3072 MB
* ml.small  1 CPU    1536 MB  1536 MB
ml.xlarge  4 CPU    8192 MB  8192 MB
tl.micro   1 CPU    512 MB   512 MB
```

The values for the CPU, RAM (MiB), and SWAP (MiB) space are listed for each type with the default type marked with an asterisk. You can change which type is selected by using the `--type` option when starting a virtual machine.

Fine-grained control over the resource allocation is also possible. The options `--cpu`, `--ram`, and `--swap` allow you to set these values separately. For values that are not specified explicitly, the value will be taken from the selected machine type.

---

### Exercise

When logged into a virtual machine, can you determine how many CPUs were allocated and how much memory? You can find this information by looking at `/proc/meminfo` and `/proc/cpuinfo`, for example.

---

### Exercise

Use the `--type`, `--cpu`, and `--ram` options to change the allocated resources for a virtual machine. Verify that the correct amount of resources has been allocated.

---

## Contextualization

Contextualization is the process by which a virtual machine discovers characteristics of its environment and properly configures itself. This is used, for example, for network configuration but can also be used for user-level service configuration.

Unfortunately, there is no standard for the contextualization process, although the CloudInit process is slowly becoming a *de facto* standard.

StratusLab supports two contextualization mechanisms: HEPiX/OpenNebula and CloudInit. For historical reasons the HEPiX/OpenNebula mechanism is currently the default.

### HEPiX Contextualization

The HEPiX/OpenNebula contextualization passes information from the user (given with the `stratus-run-instance` command) to the virtual machine via a CD-ROM image. The virtual machine automatically mounts the CD-ROM image and executes a contextualization script using the information from the image.

Your public SSH key is automatically passed to the virtual machine using this mechanism. Additional key-value pairs can be passed to the virtual machine via the `--context` parameter.

The context information can be seen on the client side by using the `stratus-describe-instance -vvv ${VM_ID}` command. This displays all of the information defining a given virtual machine.

From within the virtual machine, you can mount the CD-ROM image (if it isn't already mounted) to see what scripts and what information has been passed from the client to the virtual machine. You can find the image by using the `blkid` command. CD-ROMs have the type "iso9660".

---

### Exercise

Start a virtual machine. Log into the virtual machine, find the context CD-ROM, and mount it. What files are there? How are these executed in the startup process? (Hint: Look in `/etc/init.d/`.)

---

### Exercise

Use the context options to start another virtual machine. How are the key-value pairs you defined passed into the virtual machine? Can you imagine how to use this information to configure a service on the machine?

---

### CloudInit Contextualization

CloudInit is a very flexible contextualization mechanism that is becoming a *de facto* standard. StratusLab supports this mechanism. You can make CloudInit the default contextualization mechanism by setting the `default_context_method` value in your configuration file:

```
default_context_method = cloud-init
```

You can set this for a specific cloud infrastructure or globally in the defaults section.

To start a virtual machine using CloudInit, use the `--cloud-init` option or the `--context-method` option. The following two commands have the same effect:

```
$ stratus-run-instance --cloud-init "" \
    KhGzWhB9ZZv5ZkLSZqm6pkWx7ZF
```

```
$ stratus-run-instance --context-method cloud-init \
    KhGzWhB9ZZv5ZkLSZqm6pkWx7ZF
```

The `--cloud-init` option **requires** a value. Passing the empty string will use the default, which is to pass your SSH public key as for the HEPiX/OpenNebula contextualization.

Ubuntu provides [good documentation](#) for CloudInit describing what can be passed to the virtual machine.

To demonstrate the flexibility, we will show how to use CloudInit to start up a web server on a CentOS virtual machine. Create a file called `run-httpd.sh` with the contents:

```
#!/bin/bash -x

yum install -y httpd

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

chkconfig httpd on

service httpd start
```

This will install, configure, and run the web server on the virtual machine.

Pass this script to a virtual machine based on a CentOS image:

```
$ stratus-run-instance \
    --cloud-init "x-shellscript,run-httpd.sh" \
    KT8gOU8gve_k3UFL7p5Els57My2
```

After a couple of minutes, you should be able to visit the url <http://your-vm.example.com/test.txt> to see a page containing “SUCCESSFUL TEST”.

If you want to pass multiple files, you can separate the mimetype/file pairs with hash (#) characters or use the `stratus-prepare-context` and then the `--context-file` option of `stratus-run-instance`.

---

### Exercise

Start a virtual machine with CloudInit. Log into the virtual machine, find the context VFAT disk, and mount it. What files are there? How are these executed in the startup process?

---

### Exercise

Do the same exercise for an Ubuntu machine? What did you have to change to get this to work? Can you install and configure another service on a virtual machine that would be visible from your web browser?

---

## 1.1.4 Persistent Storage

Raw compute power without persistent storage only supports a limited range of applications. StratusLab like most cloud software also provides services for managing persistent storage.

For StratusLab, persistent storage is in the form of raw block devices (volumes). These volumes can be used in conjunction with virtual machines, but importantly have a lifecycle that is independent of them.

Raw block devices behave similarly to physical hard drives. In particular, they have some of the same limitations:

- They are initially empty and unformatted; they must be initialized when first used.

- They can only be attached on one virtual machine at a time; they must be detached from one machine before being attached to another.

StratusLab does not natively provide file or object-based storage services.

## Lifecycle

The primary commands for a persistent disks are:

```
$ stratus-create-volume --size ${GiB} --tag ${MY_DESC}
$ stratus-describe-volumes ${VOLUME_UUID}
$ stratus-update-volume [opts] ${VOLUME_UUID}
$ stratus-delete-volume ${VOLUME_UUID}
```

These commands create, describe (list), update metadata, and delete persistent disks, respectively.

The `stratus-create-volume` command takes the size of the volume in Gibibytes (GiB) and an optional tag. The tag is useful for remembering the contents of a given volume. A tag be added later with the `stratus-update-volume` command. This command returns the UUID of the created disk.

The `stratus-describe-volumes` command provides a list of persistent disks if no argument is provided. If you provide an argument, then the details for the given disk are provided. The `--filter` option is useful for limiting the number of disks returned in the list.

---

**Note:** The StratusLab caching mechanism for virtual machine appliances uses the persistent disk infrastructure. When virtual machines are running you will see “snapshot” volumes corresponding to the root volumes for those machines.

---

The `stratus-delete-volume` deletes a given volume. This releases the allocated storage space and **permanently deletes any stored data**.

**Warning:** Once a persistent disk has been deleted, all of the data on the disk is deleted and **cannot be recovered**. Be certain that you reference the correct disk before deleting it!

---

### Exercise

Run through the entire persistent disk lifecycle with the command line interface. Try to change the tag associated with the disk after you’ve created it.

---

### Exercise

Create several (small) disks with different tags and sizes. Use the filtering to limit the information returned by the `stratus-describe-volumes` command.

---

## Web Interface

There is also a web interface for this service which allows you to perform the same operations as you can from the command line. (It also allows you to see the current mount status of a disk which the command line client can’t do.)

You can discover the location of the web interface by pointing a browser at the URL for the “`pdisk_endpoint`” in your configuration (or at the “`endpoint`” value if “`pdisk_endpoint`” isn’t defined). Look for the “`svc-pdisk.html`” file in the listing.

**Warning:** That the web interface will change significantly in the coming releases.

---

## Exercise

Run through the entire persistent disk lifecycle through the web interface. Modify the tag (or add one) to an existing persistent disk.

---

### 1.1.5 Using Storage and Compute Resources Together

Having the ability to use persistent storage on a virtual machine allows you to keep the results of an analysis or to separate the service state (e.g. a database) from the service installation.

Persistent storage can be attached to a virtual machine when it is deployed or dynamically after it has started.

#### Attach a Disk at Deployment Time

The easiest way to attach a persistent disk to machine is to use the option `--persistent-disk` when launching a virtual machine. The command for doing this looks like:

```
$ stratus-run-instance --persistent-disk=${VOLUME_UUID} ${APP_ID}
```

Attaching a disk at deployment time has a couple of limitations. First, only one persistent disk can be attached to the machine. Second, the persistent disk remains attached to the virtual machine for the entire lifetime of the virtual machine.

#### Initializing the Disk

A new persistent disk is an empty, uninitialized block device. Consequently, the first time the disk is used, it must be formatted.

When the virtual machine starts the persistent disk will be attached to the machine. Logging into the virtual machine, you can find the device name for the disk by using the command `fdisk -l`, looking for a disk without a partition table and with the expected size.

A new physical disk is usually partitioned so that the space can be used for different purposes (file storage, swap, etc.). For persistent disks on the cloud, there is really no benefit in doing this unless there is a reason you need more than one file system on the disk. If you do want to partition the disk, you can use the command `fdisk` to do so.

You must however format the disk to make it useful. To format the disk, use the command `mkfs`. It is **very strongly recommended that you provide a label for the file system**.

```
$ mkfs --type ext4 -L MYLABEL /dev/sdc
mke2fs 1.42.9 (4-Feb-2014)
/dev/sdc is entire device, not just one partition!
Proceed anyway? (y,n) y
Filesystem label=MYLABEL
...
Writing superblocks and filesystem accounting information: done
```

If you did not partition the disk, you'll be asked about formatting the entire device. Just answer yes ('y') in this case.

**Warning:** The formatting of the disk should be done **only the first time you use the persistent disk**. If you reformat the disk later, all of the existing data will be lost.

## Mounting the Disk

To use the disk to store data, it must be mounted into the file system. From within the virtual machine do:

```
$ mkdir /mnt/pdisk
$ mount -L MYLABEL /mnt/pdisk
$ ls /mnt/pdisk
lost+found
$ touch /mnt/pdisk/mydata.txt
```

This shows that you can use the disk normally as you would any other disk on the system.

---

**Note:** The benefit in providing a label is that you don't need to search for the device name of the disk. The device name may vary depending on when the disk is attached to the machine and different operating systems may use different naming conventions.

---

When the machine is shutdown, the persistent disk will be released and can then be mounted on another machine.

---

### Exercise

Repeat this procedure yourself. At the end, start up a separate virtual machine with this disk and verify that the file you initially stored is still there.

---

## Attach a Disk Dynamically

Although using the `--persistent-disk` option of `stratus-run-instance` is easy; it isn't always convenient. You can attach and detach disks dynamically from running virtual machines to provide more flexibility and to overcome the limitations stated earlier.

The commands for attaching and detaching persistent disks dynamically are:

```
$ stratus-attach-volume --instance ${VM_ID} VOLUME_UUID
$ stratus-detach-volume --instance ${VM_ID} VOLUME_UUID
```

These can be used at any time while the virtual machine is running. Within the virtual machine, the volume must still be mounted from the file system in order to be accessible.

Even though modern file systems are tolerant of abrupt disconnections of devices, it is always a good idea to unmount the disk from the file system from within the virtual machine before detaching the disk with the `stratus-detach-volume` command.

---

### Exercise

Start a virtual machine, wait for it to come up, and log into it. From another terminal, dynamically attach a persistent disk to the machine. Verify that the disk is visible on the virtual machine (using `blkid` or `fdisk -l`). Mount the disk and verify that any previous data is visible. Unmount the disk and detach it. Verify that the disk is no longer visible on the virtual machine.

---

---

### Exercise

Attach and detach a persistent disk multiple times. Does the device name remain the same or change?

---

---

### Exercise

A volume can be mounted only on one virtual machine at a time. Try to mount a volume on more than one virtual machine to see what error message is given.

---



## 1.1.6 Alternate Storage Types

StratusLab also offers two other storage types to support specific use cases: temporary storage and readonly storage.

### Volatile Storage

Many applications have a need for a large storage space for temporary data. StratusLab provides volatile storage for this purpose. The space will be reclaimed (and the data lost) when the virtual machine is terminated.

When starting a virtual machine a volatile disk can be added to the machine using the `--volatile-disk` option:

```
$ stratus-run-instance \  
  --volatile-disk 10 \  
  KhGzWhB9ZZv5ZkLSZqm6pkWx7ZF
```

You must provide the size (in GiB) of the disk to be allocated.

As for persistent disks, the volatile disks are not formatted. You will need to find and format the disk in the same way as for a persistent disk.

---

### Exercise

Start a virtual machine with a volatile disk. Find, format, mount, and store data on the disk. Verify that the data on the disk will survive a machine reboot. (Reboot the virtual machine using `reboot` within the virtual machine.) Unless you added the disk to the `/etc/fstab` file, you will need to remount the disk after the reboot. The disk and data will disappear when the machine is terminated with `stratus-kill-instance`.

---

### Static Disks

Many applications use datasets that are either fixed or change slowly, for example, input databases for a scientific calculation. Since these are fixed datasets, you can take advantage of the virtual machine image caching mechanisms to efficiently cache and duplicate these datasets.

These static (read-only) disks can be attached when starting a virtual machine. As for virtual machine images, these disk images are also registered in the Marketplace. To start a machine with a static disk:

```
$ stratus-run-instance \  
  --readonly-disk GPAUQFkojP5dMQJNdJ4qD_62mCo \  
  KhGzWhB9ZZv5ZkLSZqm6pkWx7ZF
```

The value used for the `--readonly-disk` is the Marketplace identifier of the disk. The one used here is a CD-ROM image with a small database of “Flora and Fauna”. You can search the Marketplace for it.

You can find, mount, and use the disk:

```
$ blkid  
/dev/sr0: LABEL="_STRATUSLAB" TYPE="iso9660"  
/dev/sda1: UUID="b73f3da6-3ca3-4833-a2ab-8a03d6a47b2c" TYPE="ext4"  
/dev/sdb: UUID="2a76d471-02d3-45e1-859f-eef3d2946dbb" TYPE="swap"  
/dev/sdc: LABEL="CDROM" TYPE="iso9660"  
  
$ mkdir /mnt/data  
$ mount /dev/sdc /mnt/data  
mount: block device /dev/sdc is write-protected, mounting read-only
```

```
$ ls -lR /mnt/data
/mnt/data:
total 4
dr-xr-xr-x 1 root root 2048 Jan 26 2013 animals
dr-xr-xr-x 1 root root 2048 Jan 26 2013 plants

/mnt/data/animals:
total 1
-r-xr-xr-x 1 root root 4 Jan 26 2013 cat.txt
-r-xr-xr-x 1 root root 4 Jan 26 2013 dog.txt

/mnt/data/plants:
total 1
-r-xr-xr-x 1 root root 5 Jan 26 2013 rose.txt
-r-xr-xr-x 1 root root 6 Jan 26 2013 tulip.txt
```

Note that the disk is read-only, so you cannot make any changes to this disk.

The disk will appear in the virtual machine exactly as it has been formatted in the reference image. Usually these images are formatted as a CD-ROM (or DVD) image so that they are portable between different operating systems.

When creating such images it is strongly recommended (unlike here) that they be created with a label so that they can easily be mounted by the user without needing to know what device has been assigned.

---

### Exercise

Run a machine with this disk image. Verify that it is indeed read-only. What happens if you start a second disk with the same image?

---

## 1.1.7 Next Steps

This tutorial has shown you the core functionality available on a StratusLab cloud infrastructure. With just this functionality a large majority of use cases are covered. Nonetheless you are invited to look through the User Guide to acquaint yourself with advanced StratusLab features.

Concretely, some advanced use cases require you to create new appliances or data sets and to register them in the Marketplace. Information about performing those tasks (as well as other advanced features) can be found in the more extensive User Guide.

If you don't find the information or features you need in the documentation, contact StratusLab support for help!

## 1.2 Installation Tutorial

This tutorial demonstrates how to install manually a StratusLab cloud infrastructure using the StratusLab system administrator command line utilities.

The smallest, standard StratusLab cloud consists of two physical machines. You will need to have two relatively modern machines to complete this tutorial.

### 1.2.1 Overview

The StratusLab distribution provides a simple command line client to install, configure and start the StratusLab Cloud services and components.

The deployment described in this tutorial requires two physical machines with a minimal version of CentOS 6 (or compatible) installed. The deployment consists of the two StratusLab machine types:

- **Front-End** - machine for VM management and storage services
- **Node** - machine that hosts virtual machines

The procedure for installing a minimal StratusLab cloud consists of the following steps:

1. Verification of all of the prerequisites.
2. Installation of the StratusLab administrator tools.
3. Definition of all of the StratusLab service parameters.
4. Configuration and installation of the Front End.
5. Configuration and installation of the Node.
6. Validation of the cloud installation.

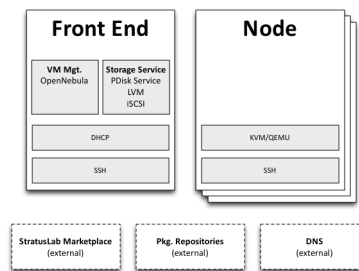


Fig. 1.1: Minimal StratusLab Cloud

## 1.2.2 Prerequisites

This tutorial demonstrates a minimal installation of a StratusLab cloud on two physical machines. Before trying to install the StratusLab software there are many prerequisites that must be satisfied by the hardware, operating system, and software.

### Physical Machines

The physical machines should be relatively modern machines with the following **minimum** characteristics:

- 1 **64-bit multicore** CPU ( $\geq 4$  cores)
- 4 GB of RAM
- 200 GB local disk space

In general cloud infrastructures prefer “fat” machines; that is machines that have the maximum number of CPUs, RAM, and disk space as possible. This is because the maximum size of a single virtual machine is limited by the size of the largest physical machine.

### Virtualization Support

The machine must have a CPU that supports the VT-x virtualization extensions and must have this support enabled in the BIOS. **Many vendors ship machines with these extensions disabled.**

RedHat has a [good document](#) for checking if your machine will support virtualization. In short, check the `/proc/cpuinfo` for the proper flags:

```
$ grep vmx /proc/cpuinfo
```

and check the `dmesg` console for any KVM errors when trying to load the KVM kernel module:

```
$ dmesg | grep kvm
```

looking for any messages about KVM not supporting virtualisation or having it disabled in BIOS.

## Operating System

Be sure to read all of the operating system requirements before installing your machines. This will save you from having to do the installation multiple times!

### Supported Versions

Install a minimal version of [CentOS 6](#) on the two physical machines that will be used for the cloud infrastructure. Other distributions that are compatible with CentOS 6 will likely work; however, only CentOS 6 is systematically tested.

Operating Systems in the Debian family (e.g. Ubuntu) are not currently supported.

### Disable SELinux

The SELinux system must be disabled on **all of the machines** in the cloud infrastructure. The standard CentOS installation procedure normally activates SELinux.

To disable SELinux, ensure that the file `/etc/selinux/config` has the following line:

```
SELINUX=disabled
```

You can also use the commands `getenforce` and `sestatus` to find the current SELinux status. Changes in this configuration are only taken into account after **rebooting the machine**.

You can reboot the machine to have your changes taken into account or disable SELinux temporarily:

```
$ echo 0 > /selinux/enforce
```

This change will be lost at the next reboot unless you have also changed the `/etc/selinux/config` configuration file.

## Disk Configuration

StratusLab allows for a variety of storage options behind the persistent disk service.

This tutorial uses the default storage solution using LVM and iSCSI. Because of this, the machines **must be configured to use LVM for the disk storage**.

The Front End will host the storage service and the physical storage associated with it. It is strongly recommended that the Front End machine be configured with **two LVM groups**: one for the base operating system (~20 GB) and one for the StratusLab storage service (remaining space). One LVM group is possible, but you risk starving normal system services of disk space.

In the tutorial, we assume that the volume group names are “vg.01” for the operating system and “vg.02” for the StratusLab storage service on the Front End. You can use other names, but change the commands and configuration parameters below as necessary.

The “Node” machine can be configured with a single LVM group.

You can see the configured volume groups with the command:

```
$ vgdisplay

--- Volume group ---
VG Name                rootvg
System ID
Format                 lvm2
...
```

You’ll need the volume group name “VG Name” of the volume group you’ll be using for the cloud storage. Verify that the device associated with the group name exists. Here for example, the device is `/dev/rootvg`.

## Software

### Python Version

The default version of Python installed with CentOS should be correct. Verify that the correct version of Python is installed:

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

StratusLab requires a version of Python 2 with a version **2.6 or later**. The StratusLab command line tools **do not work with Python 3**.

### Package Repositories

The StratusLab installation takes packages from four yum repositories:

1. The standard CentOS repository,
2. The [EPEL 6](#) repository,
3. The [StratusLab repository](#), and
4. The [IGTF Root Certificates](#).

The configuration for the CentOS repository is done when the system is installed and the IGTF repository will be configured by the StratusLab tools as necessary. The others require explicit configuration.

The directory `/etc/yum.repos.d` contains the currently configured yum repositories.

**EPEL Repository** Configure **both** the Front End and Node for the EPEL repository. Do the following:

```
$ wget -nd http://mirrors.ircam.fr/pub/fedora/epel/6/i386/epel-release-6-8.noarch.rpm
$ yum install -y epel-release-6-8.noarch.rpm
```

This will add the necessary files to the `/etc/yum.repos.d/` directory. You can find the latest version of the EPEL configuration RPM on the [EPEL wiki](#).

**StratusLab Repository** To configure **both** the Front End and Node for the StratusLab repository, put the following into the file `/etc/yum.repos.d/stratuslab.repo`:

```
[StratusLab-Releases]
name=StratusLab-Releases
baseurl=http://yum.stratuslab.eu/releases/centos-6-v14.06.0_RC5/
gpgcheck=0
```

replacing the URL with the version you want to install.

**Cleanup and Upgrade** Although not strictly necessary, it is advisable to clear all of the yum caches and upgrade the packages to the latest versions:

```
$ yum clean all
$ yum upgrade -y
```

This may take some time if you installed the base operating system from old media.

## Network Setup

### DNS and Hostname

Ensure that the **hostname** is properly setup on the Front End and the Node. The DNS must provide both the forward and reverse naming of the nodes. For example:

```
$ host cloud.lal.stratuslab.eu
cloud.lal.stratuslab.eu is an alias for onehost-4.lal.in2p3.fr.
onehost-4.lal.in2p3.fr has address 134.158.75.4

$ host 134.158.75.4
4.75.158.134.in-addr.arpa domain name pointer onehost-4.lal.in2p3.fr.
```

Ensure that the host resolves to an IP address and that the IP address resolves back to the original name (or alias).

Also ensure that the hostname is properly set for the node:

```
$ hostname -f
```

should return the full hostname (with domain). Set the hostname if it is not correct.

Throughout this tutorial we use the variables `$FRONTEND_HOST` (`$FRONTEND_IP`) and `$NODE_HOST` (`$NODE_IP`) for the Front End and Node hostnames (IP addresses), respectively. Change these to the proper names for your physical machines when running the commands.

### DHCP Server

**You must have a range of free IP addresses that can be assigned to virtual machines.** The range should be large enough to handle the maximum number of virtual machines you expect to have running simultaneously on your infrastructure.

These IP addresses must be publicly visible if the cloud instances are to be accessible from the internet.

In addition, a DHCP server must be configured to assign static IP addresses corresponding to known MAC addresses for the virtual machines. You can use an external DHCP server or if one is not available (or not desired), the StratusLab installation command can be used to properly configure a DHCP server on the Front End for the virtual machines.

This tutorial will start a DHCP server on the Front End by default.

## Network Bridge

A network bridge must be configured on the Node to allow virtual machines access to the internet. You can do this manually if you want, but the StratusLab installation scripts are capable of configuring this automatically.

This tutorial uses the installation scripts to configure the network bridge.

## SSH Configuration

The installation scripts will automate most of the work, but the scripts require **password-less root access**:

- From the Front End to each Node and
- From the Front End to the Front End itself

Check to see if there is already an SSH key pair in `/root/.ssh/id_rsa*`. If not, then you need to create a new key pair **without a password**:

```
$ ssh-keygen -q
Enter file in which to save the key (/root/.ssh/id_rsa):
/root/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

Now do the necessary configuration to ensure that you can log into the Front End from the Front End with your SSH key (and without a password). Do the following:

```
$ ssh-copy-id $FRONTEND_HOST
The authenticity of host 'onehost-5.lal.in2p3.fr (134.158.75.5)' can't be established.
RSA key fingerprint is e9:04:03:02:e5:2e:f9:a1:0e:ae:9f:9f:e4:3f:70:dd.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'onehost-5.lal.in2p3.fr,134.158.75.5' (RSA) to the list of known hosts.
root@onehost-5.lal.in2p3.fr's password:
Now try logging into the machine, with "ssh 'onehost-5.lal.in2p3.fr'", and check in:

    .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't
expecting.
```

and then the same thing for the node:

```
$ ssh-copy-id $NODE_HOST
...
```

After these commands your key should have been added to the *authorized\_key* file on both nodes and should allow you to log in without a password.

**Note:** If your machine does not have the *ssh-copy-id* command, then you will have to do the configuration by hand. Append the contents of your *\$HOME/.ssh/id\_rsa.pub* file to the *\$HOME/.ssh/authorized\_keys* file on both the Front End and the Node. You will also have to accept the host's SSH key the first time you log in.

Verify that the password-less access works as expected.

```
$ ssh $FRONTEND_HOST

Last login: Mon May 27 14:26:29 2013 from mac-91100.lal.in2p3.fr
```

```
#
# exit
logout
Connection to onehost-5.lal.in2p3.fr closed.

$ ssh $NODE_HOST

Last login: Mon May 27 14:26:43 2013 from mac-91100.lal.in2p3.fr
#
# exit
logout
Connection to onehost-6.lal.in2p3.fr closed.
```

Now that SSH is properly configured, the StratusLab scripts will be able to install software on both the Front End and the Node.

### 1.2.3 Administrator CLI

The StratusLab administrator command line client will help define parameters and perform the installation.

#### Installation

The first step is to install the StratusLab system administrator command line client from the [StratusLab repository](#) on **the Front End**:

```
$ yum install -y stratuslab-cli-sysadmin
```

This will install the system administrator client and all of the necessary dependencies. You can verify that it is correctly installed by doing the following:

```
$ stratus-config --help

Usage: stratus-config [options] [key [value]]
If the [value] is not provided, the command returns the current value
of the key.
...
```

#### Service Configuration

The command *stratus-config* will help you view and set the various parameters for the StratusLab installation. This is essentially an interface to the configuration file */etc/stratuslab/stratuslab.cfg* and the associated reference file */etc/stratuslab/stratuslab.cfg.ref*.

To list all of the defined parameters (keys) and their values use the command:

```
$ stratus-config --keys
...
```

This will display a long list of keys, their current values, and their default values, by section. You can view a single parameter by naming the parameter:

```
$ stratus-config frontend_system
```

and set it by giving a value:



```
$ stratus-config frontend_system centos
```

Each service has a parameter to determine whether the service will be installed. For example, the registration service:

```
$ stratus-config registration
False
```

is not installed by default. The values that should be set will be discussed in the following sections.

## Service Installation

Once all of the service configuration parameters have been set, you can use the `stratus-install` command to perform the installation. This command should always be executed on the Front End. It will use SSH to access the machines (including the Front End!) where the services will be installed.

Normally, if errors occur in the installation, you can simply correct the configuration and rerun `stratus-install`.

This command is designed to simplify the initial installation of the cloud. It can be used also to update the cloud software, although there are some significant limitations when doing this. Notably there are certain parts of the configuration (like the address ranges) that cannot be updated automatically.

### 1.2.4 Configuration Parameters

There is a fair number of cloud services and a large number of associated parameters. Fortunately, the default values for many of these services will work fine, so only a limited number of parameters actually need to be set.

This section describes the parameters that need to be set by service.

#### Front End Parameter

For the Front End, you need only set the value of its IP address:

```
$ stratus-config frontend_ip $FRONTEND_IP
```

This must be set to the real IP address of the node. The localhost value 127.0.0.1 will not produce a working system.

#### Storage Service

Similar parameters must also be set for the Persistent Disk service.

For this tutorial, this service is installed on the Front End, so the same IP address should be used.

```
$ stratus-config persistent_disk_system centos
$ stratus-config persistent_disk_ip $FRONTEND_IP
$ stratus-config persistent_disk_port 443
$ stratus-config persistent_disk_path pdisk
$ stratus-config persistent_disk_merge_auth_with_proxy True
```

The Persistent Disk service and the Nodes communicate using a strategy defined by the `persistent_disk_storage` and `persistent_disk_share` parameters. The default values (“lvm” and “iscsi”, respectively) will be used for this tutorial.

One needs to specify what device will be used for the physical storage for the Persistent Disk service:

```
$ stratus-config persistent_disk_lvm_device /dev/vg.02

# *** NOTE: Copy the following exactly. Be careful of ***
# *** the opening and closing single quotes! ***

$ stratus-config persistent_disk_backend_sections '
[% (persistent_disk_ip)s]
    type=LVM
    volume_name = /dev/vg.02
    lun_namespace = stratuslab
    volume_snapshot_prefix = pdisk_clone
    initiator_group =
'
```

**Warning:** If your LVM volume group name is not “vg.02”, then change the values in the above commands.

## Network configuration

Use the frontend as the general gateway for the cloud:

```
$ stratus-config default_gateway $FRONTEND_IP
```

Set the IP and mac addresses for virtual machines:

```
# space-separated list
$ stratus-config one_public_network_addr \
    134.158.xx.yy 134.158.xx.yy 134.158.xx.yy

# space-separated list
$ stratus-config one_public_network_mac \
    0a:0a:86:9e:49:2a 0a:0a:86:9e:49:2b 0a:0a:86:9e:49:2c
```

In this example, the Front-End is configured on IP address \$FRONTEND\_IP and three IP/MAC address pairs are defined for virtual machines.

**You must use the real values for the Front End IP addresses and for the range of addresses you will use for the virtual machines.** The mac addresses are arbitrary, but it is a good idea to have the last four fields match the IPv4 network address.

More network parameters are described in the “one-network” section in the reference configuration file.

## DHCP Configuration

Allow the script to automatically configure and start the DHCP server on the Front End. Do the following:

```
$ stratus-config dhcp True
$ stratus-config dhcp_subnet 134.158.75.0
$ stratus-config dhcp_netmask 255.255.255.0
$ stratus-config dhcp_lease_time 3600

$ stratus-config dhcp_one_public_network True
$ stratus-config dhcp_one_public_network_routers $FRONTEND_IP
$ stratus-config dhcp_one_public_network_domain_name lal.in2p3.fr

# comma-separated list
```

```
$ stratus-config dhcp_one_public_network_domain_name_servers \
    134.158.91.80, 134.158.88.149
```

Use **your** values for these parameters!

## Review Parameters

Do a final review of all of the service parameters to make sure that they are correct. If everything looks good, you're ready to do the Front End installation.

## 1.2.5 Front End Installation

Now that we have defined all of the configuration parameters, you can now do the full Front End installation by issuing the following command:

```
$ stratus-install -vv 2>&1 | tee frontend-install.log
```

To get more details on what the command is (because of curiosity or errors), use the option `-v`, `-vv`, or `-vvv`.

If you run into errors, the `stratus-install` command can simply be rerun after adjusting the configuration parameters.

When the installation completes successfully, you should see the following services running on the machine:

```
$ service nginx status
nginx (pid 15215) is running...

$ service one-proxy status
Checking arguments to Jetty:
...
Jetty running pid=15847

$ service pdisk status
Checking arguments to Jetty:
...
Jetty running pid=17099
```

You should also see a web page running at `https://{FRONTEND_HOST}/` that lists these services:

```
$ curl -k https://{FRONTEND_HOST}/
<html>
...
<a href="readme.html">readme.html</a>
...
<a href="svc-one-proxy.html">svc-one-proxy.html</a>
...
<a href="svc-pdisk.html">svc-pdisk.html</a>
...
</html>
```

If the page doesn't appear or any of the services are not running, you should investigate the errors in the log files. The log files for all StratusLab services are in `/var/log/stratuslab`. Each service in a separate subdirectory.

## 1.2.6 Node Installation

The deployment of the StratusLab Nodes is done from the Front End, thus, **all the commands below should be run from the Front End.**

To add a Node to the cloud, specify the Linux distribution of the machine and indicate that the bridge should be configured:

```
$ stratus-config node_system centos
```

Request the automatic configuration of the network bridge:

```
$ stratus-config node_bridge_configure True
$ stratus-config node_bridge_name br0
$ stratus-config node_network_interface eth0
```

**Warning:** Check carefully the name of the interface on the node! Using the wrong interface will cause the network to be misconfigured and you will likely lose remote access to the machine.

Invoke installation with:

```
$ stratus-install -vv -n $NODE_IP 2>&1 | tee node-install.log
```

As before, you can increase the verbosity level by adding the option `-v` or `-vv`.

## 1.2.7 Testing Installation

At this point, you have both the Front End and one Node installed. In principle this is a fully functional installation. To ensure this is the case, we'll run the standard battery of tests on the cloud infrastructure as a normal user.

### Cloud User

We will create a new StratusLab cloud user account. Note that StratusLab accounts are independent of the Unix accounts on the machine itself.

Add the following line to the end of the file `/etc/stratuslab/authn/login-pswd.properties`:

```
$ cat >> /etc/stratuslab/authn/login-pswd.properties
sluser=slpass,cloud-access
```

This creates a new StratusLab user 'sluser' with a password 'slpass'. The group 'cloud-access' is mandatory for the user to have access to the cloud services. (Crypted or hashed password values are also allowed in the configuration.)

The StratusLab distribution supports other authentication methods (LDAP, X509 certificates, X509 proxies, etc.), but a username/password pair is the simplest for this tutorial.

### StratusLab Client

Now we will test that the cloud functions correctly by starting a new virtual machine instance and logging into it. We'll test the cloud service from a normal Unix user account on the Front End.

First, ensure that the StratusLab user client is installed on the machine. Do the following as root:

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

This will do a system-wide installation of the StratusLab client from the RPM package.

**Note:** For normal client installations, it is strongly recommended to use `pip` for the client installation. See the User Tutorial or User Guide for more information.

**Warning:** The RPM and pip installations are incompatible. Use only one method for installing the client!

## Unix Account

Now create a normal Unix user for testing:

```
$ adduser sluser
```

We've chosen the same name as the cloud user, but the unix and cloud user accounts are distinct.

Now log in as the user and setup the account for using StratusLab. An SSH key pair is required to log into your virtual machines and the client requires that a complete client configuration file.

Log in as the user and create an SSH key pair. This is similar to the process used for the root account on the machine:

```
$ su - sluser
$ ssh-keygen -q
...
```

You can create a passphrase for this SSH key if you want. If you do, you'll need to provide the passphrase when logging into virtual machines.

Now copy the reference configuration file into place and edit the parameters:

```
$ stratus-copy-config
$ vi $HOME/.stratuslab/stratuslab-user.cfg
```

You will need to set the “endpoint”, “username”, and “password” parameters in this file. For the “endpoint” use the hostname or IP address of your Front End. For the “username” and “password” use “sluser” and “slpass”, respectively.

## Hello Cloud

Everything should be setup now. So just “ping” the services to ensure that they are running and accessible.

```
$ stratus-describe-volumes
No disk to show

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

If the services do not respond, verify that the services are running and accessible through the nginx proxy. You may need to restart the nginx server after the server configuration.

## Storage Check

Because the virtual machine management infrastructure depends on the storage system. We should first verify that the storage system works.

Run through the complete lifecycle of a persistent disk:

```
$ stratus-create-volume --size 1 --tag HELLO
DISK 87955437-1baa-412b-87ed-71e59e45e0f4

$ stratus-describe-volumes
:: DISK 87955437-1baa-412b-87ed-71e59e45e0f4
   count: 0
   owner: sluser
   tag: HELLO
   size: 1

$ stratus-delete-volume 87955437-1baa-412b-87ed-71e59e45e0f4
DELETED 87955437-1baa-412b-87ed-71e59e45e0f4

$ stratus-describe-volumes
No disk to show
```

If the full lifecycle works, then you're ready to try to deploy a virtual machine.

## Virtual Machine Check

You installed manually the package which provides the load information for the cloud. Check that the Node you configured is visible and not showing any errors:

```
$ curl -k https://${FRONTEND_HOST}/load.txt
ID NAME                RVM   TCPU   FCPU   ACPU   TMEM   FMEM   AMEM   STAT
0 134.158.48.52         0      0      0     100    OK     OK     OK    err
```

If there are no machines listed or there is an error, you will need to correct this before going on.

```
$ stratus-run-instance KhGzWhB9ZZv5ZkLSZqm6pkWx7ZF

::::::::::::::::::::::::::
:: Starting machine(s) ::
::::::::::::::::::::::::::
:: Starting 1 machine
:: Machine 1 (vm ID: 1)
Public ip: 134.158.75.42
:: Done!
```

Check the status of the machine as it starts:

```
# Check its status. Pending -> not yet assigned to a Node
$ stratus-describe-instance
id state   vcpu memory   cpu% host/ip                name
1  Pending  1      0           0   vm-42.lal.stratuslab.eu one-1

# Check again. Prolog -> resources for VM are being initialized
$ stratus-describe-instance
id state   vcpu memory   cpu% host/ip                name
1  Prolog   1      0           0   vm-42.lal.stratuslab.eu one-1

# Check again. Running -> hypervisor has started machine
$ stratus-describe-instance
id state   vcpu memory   cpu% host/ip                name
1  Running  1      0           0   vm-42.lal.stratuslab.eu one-1
```

When the machine reaches the 'running' status, the virtual machine is running in the hypervisor on the Node. It will probably take some additional time for the operating system to boot.

Verify that the machine has fully booted and is accessible from the network:

```
# Ping the virtual machine to see if it is accessible.
$ ping vm-42.lal.stratuslab.eu
PING vm-42.lal.stratuslab.eu (134.158.75.42) 56(84) bytes of data.
From onehost-5.lal.in2p3.fr (134.158.75.5) icmp_seq=2 Destination Host
  Unreachable
...
From onehost-5.lal.in2p3.fr (134.158.75.5) icmp_seq=8 Destination Host
  Unreachable
64 bytes from vm-42.lal.stratuslab.eu (134.158.75.42): icmp_seq=9
  ttl=64 time=1.44 ms
...

# Now login to the machine as root.
$ ssh root@vm-42.lal.stratuslab.eu

The authenticity of host 'vm-42.lal.stratuslab.eu (134.158.75.42)'
  can't be established.
RSA key fingerprint is
  6a:bd:f7:2d:b6:82:39:61:e6:ca:3f:c7:61:9d:72:31.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'vm-42.lal.stratuslab.eu,134.158.75.42'
  (RSA) to the list of known hosts.

#           # <- we're logged into the virtual machine
# exit     # just logout of the session
logout
Connection to vm-42.lal.stratuslab.eu closed.
```

Now the machine can be terminated:

```
$ stratus-kill-instance 1
```

Going through the full lifecycle of a machine shows that all of the services are working.

## 1.2.8 Conclusions

You've successfully installed a minimal StratusLab cloud. You can checkout the [documentation](#) to see what other configuration parameters are available or try the user tutorials to discover more of the StratusLab services.

You can get help on the installation or use of StratusLab through the [support mailing list](#). You can also report bugs and provide feedback on the same list.





## 2.1 User Guide

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.

### 2.1.1 Preface

#### 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.

#### Typographic Conventions

This guide uses several typographic conventions to improve the readability.

links	<a href="#">some link</a>
filenames	<code>\$HOME/.stratuslab/stratuslab-user.cfg</code>
commands	<code>stratus-run-instance</code>
options	<code>--version</code>

Table: 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.

## 2.1.2 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.

## 2.1.3 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 thoroughly.

### Verifying the Prerequisites

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.

## 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.

## 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.

## Testing the Installation

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

### 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 `ttlinux` 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
ttlinux_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 you 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.

## 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.

## 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.

## 2.1.4 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.

### 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.

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

Table: Overview of StratusLab commands.

### Installation

#### Verifying the Prerequisites

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
```

## Local Installation

For local installations of the StratusLab client, for example within a non-privileged 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.

## 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.

## 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.

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

Table: 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.

## 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: 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`.

## 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: 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.

## 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: 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`.

## 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.

## 2.1.5 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!).

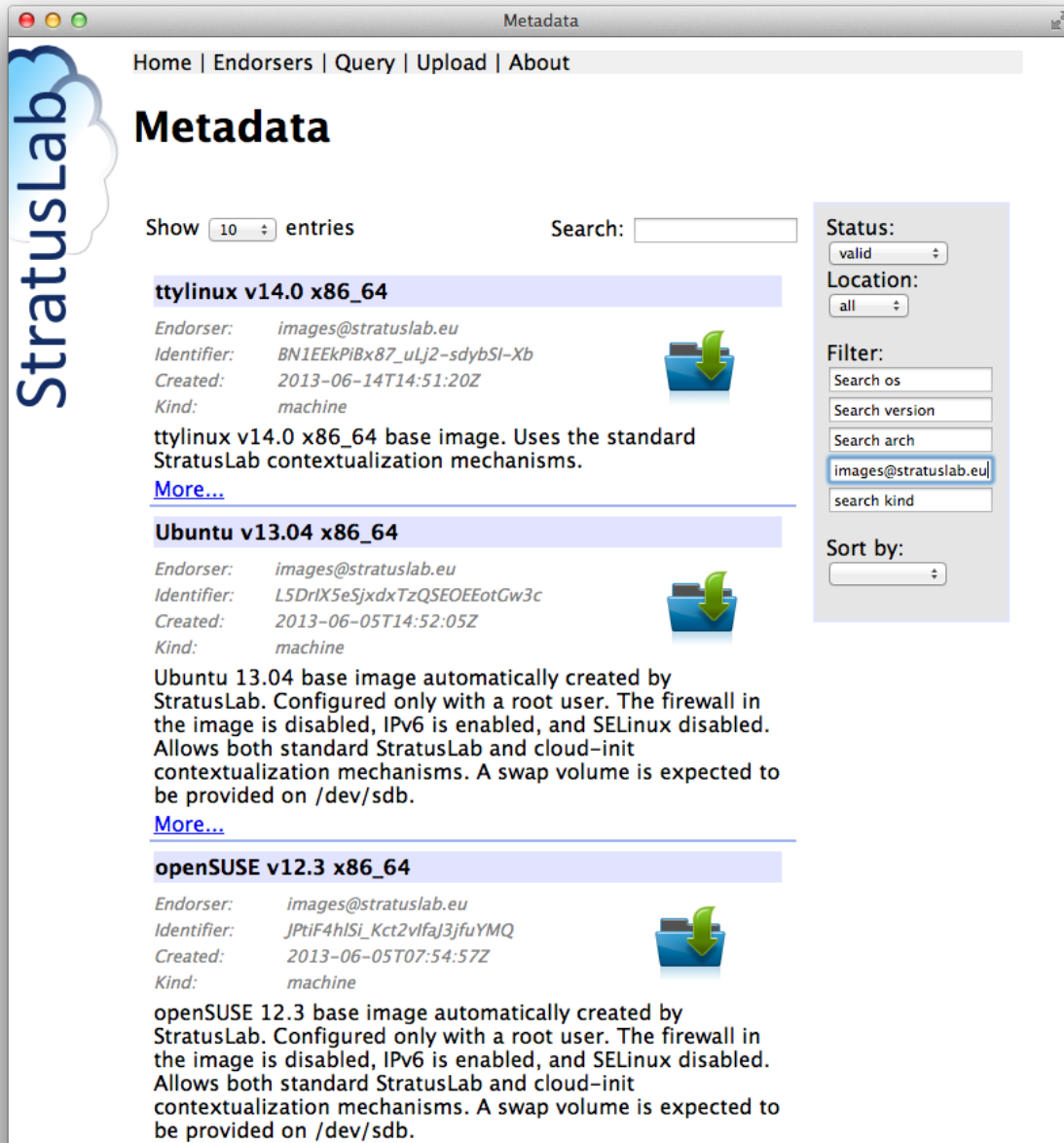
Eventually StratusLab will provide a unified portal to allow access to all services via a web browser, but this is not yet available. Nevertheless, several of the StratusLab services provide a web interface. These interfaces are essentially a thin veneer over the services’ REST interfaces.

### 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.

### 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.



Metadata

Home | Endorsers | Query | Upload | About

## Metadata

Show  entries      Search:

**ttylinux v14.0 x86\_64**

Endorser: *images@stratuslab.eu*  
 Identifier: *BN1EEkPiBx87\_uLj2-sdybSI-Xb*  
 Created: *2013-06-14T14:51:20Z*  
 Kind: *machine*

ttylinux v14.0 x86\_64 base image. Uses the standard StratusLab contextualization mechanisms.  
[More...](#)

**Ubuntu v13.04 x86\_64**

Endorser: *images@stratuslab.eu*  
 Identifier: *L5DrIX5eSjxdxTzQSEOEotGw3c*  
 Created: *2013-06-05T14:52:05Z*  
 Kind: *machine*

Ubuntu 13.04 base image automatically created by StratusLab. Configured only with a root user. The firewall in the image is disabled, IPv6 is enabled, and SELinux disabled. Allows both standard StratusLab and cloud-init contextualization mechanisms. A swap volume is expected to be provided on /dev/sdb.  
[More...](#)

**openSUSE v12.3 x86\_64**

Endorser: *images@stratuslab.eu*  
 Identifier: *JPtiF4hISi\_Kct2vlfaj3jfuYMQ*  
 Created: *2013-06-05T07:54:57Z*  
 Kind: *machine*

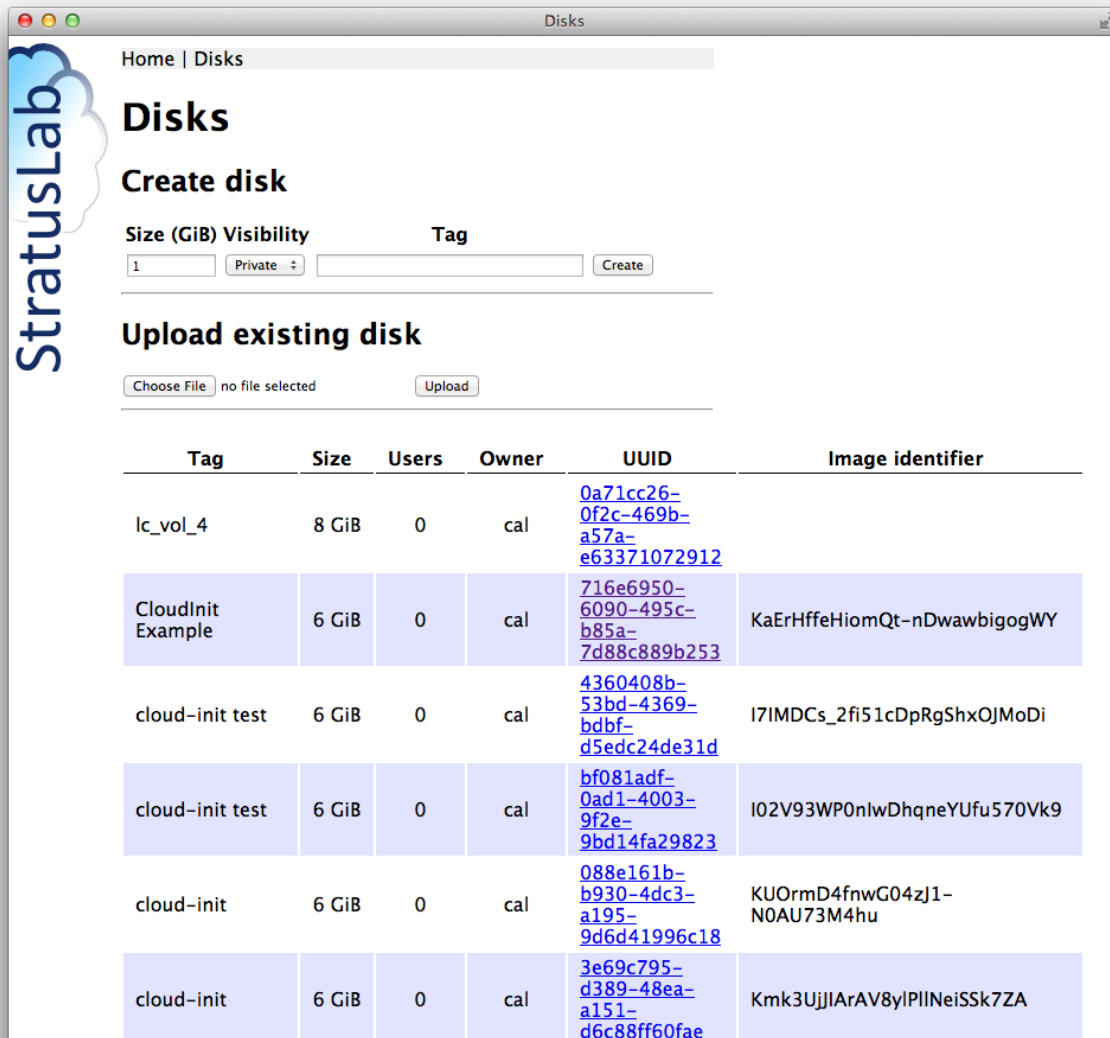
openSUSE 12.3 base image automatically created by StratusLab. Configured only with a root user. The firewall in the image is disabled, IPv6 is enabled, and SELinux disabled. Allows both standard StratusLab and cloud-init contextualization mechanisms. A swap volume is expected to be provided on /dev/sdb.

**Filter:**

Status:   
 Location:   
 Search os:   
 Search version:   
 Search arch:   
 images@stratuslab.eu  
 search kind:

**Sort by:**

Fig. 2.1: Marketplace Screenshot



Home | Disks

## Disks

### Create disk

Size (GiB) Visibility Tag

1 Private  Create

### Upload existing disk

Choose File no file selected Upload

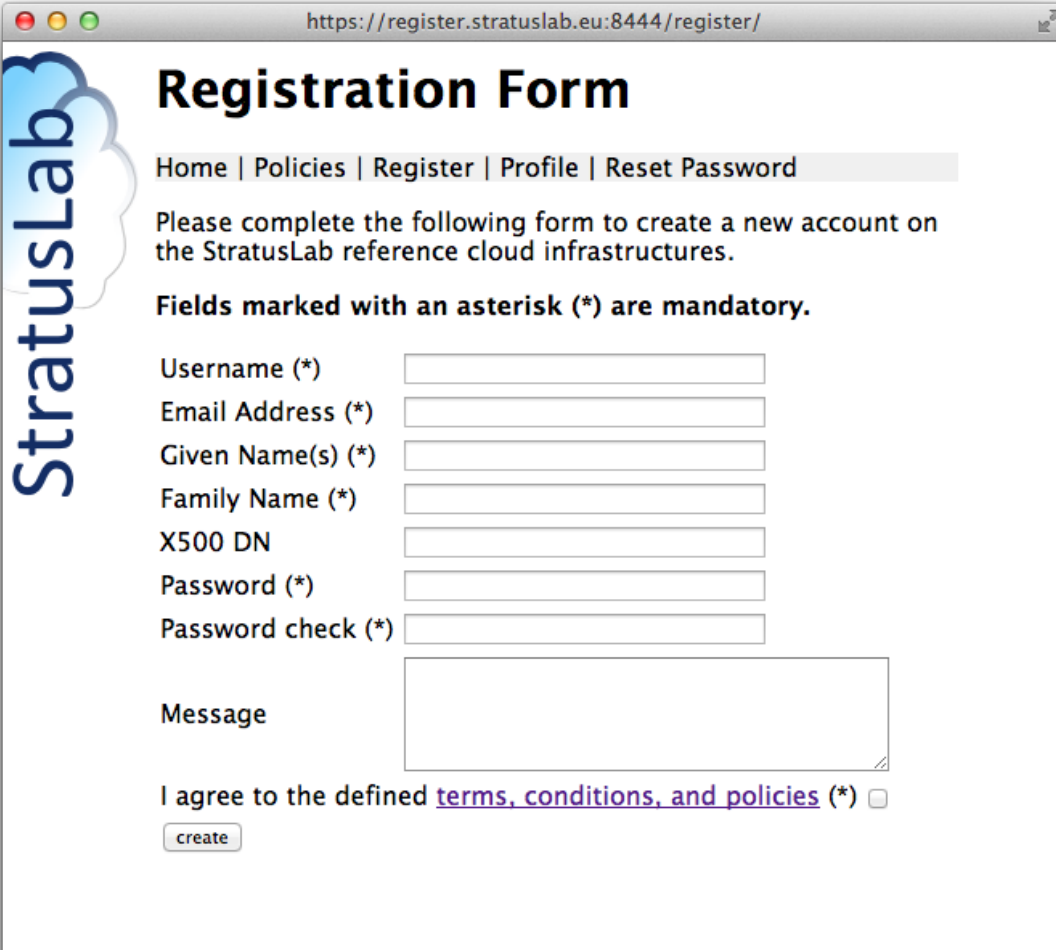
Tag	Size	Users	Owner	UUID	Image identifier
lc_vol_4	8 GiB	0	cal	<a href="#">0a71cc26-0f2c-469b-a57a-e63371072912</a>	
CloudInit Example	6 GiB	0	cal	<a href="#">716e6950-6090-495c-b85a-7d88c889b253</a>	KaErHffeHiomQt-nDwawbigogWY
cloud-init test	6 GiB	0	cal	<a href="#">4360408b-53bd-4369-bdbf-d5edc24de31d</a>	I7IMDCs_2fi51cDpRgShxOJMoDi
cloud-init test	6 GiB	0	cal	<a href="#">bf081adf-0ad1-4003-9f2e-9bd14fa29823</a>	I02V93WP0nlwDhqneYUfu570Vk9
cloud-init	6 GiB	0	cal	<a href="#">088e161b-b930-4dc3-a195-9d6d41996c18</a>	KUOrmD4fnwG04zJ1-N0AU73M4hu
cloud-init	6 GiB	0	cal	<a href="#">3e69c795-d389-48ea-a151-d6c88ff60fae</a>	Kmk3UjJIaArAV8ylPIlNeiSSk7ZA

Fig. 2.2: Storage Service Screenshot

## 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](#).



The screenshot shows a web browser window with the URL `https://register.stratuslab.eu:8444/register/`. The page features the StratusLab logo on the left and a navigation bar with links: Home | Policies | Register | Profile | Reset Password. The main heading is "Registration Form". Below it, a message states: "Please complete the following form to create a new account on the StratusLab reference cloud infrastructures." A note indicates: "Fields marked with an asterisk (\*) are mandatory." The form includes input fields for: Username (\*), Email Address (\*), Given Name(s) (\*), Family Name (\*), X500 DN, Password (\*), and Password check (\*). There is a large text area for a "Message". At the bottom, there is a checkbox for "I agree to the defined [terms, conditions, and policies](#) (\*)" and a "create" button.

Fig. 2.3: Registration Screenshot

### 2.1.6 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.

## 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.

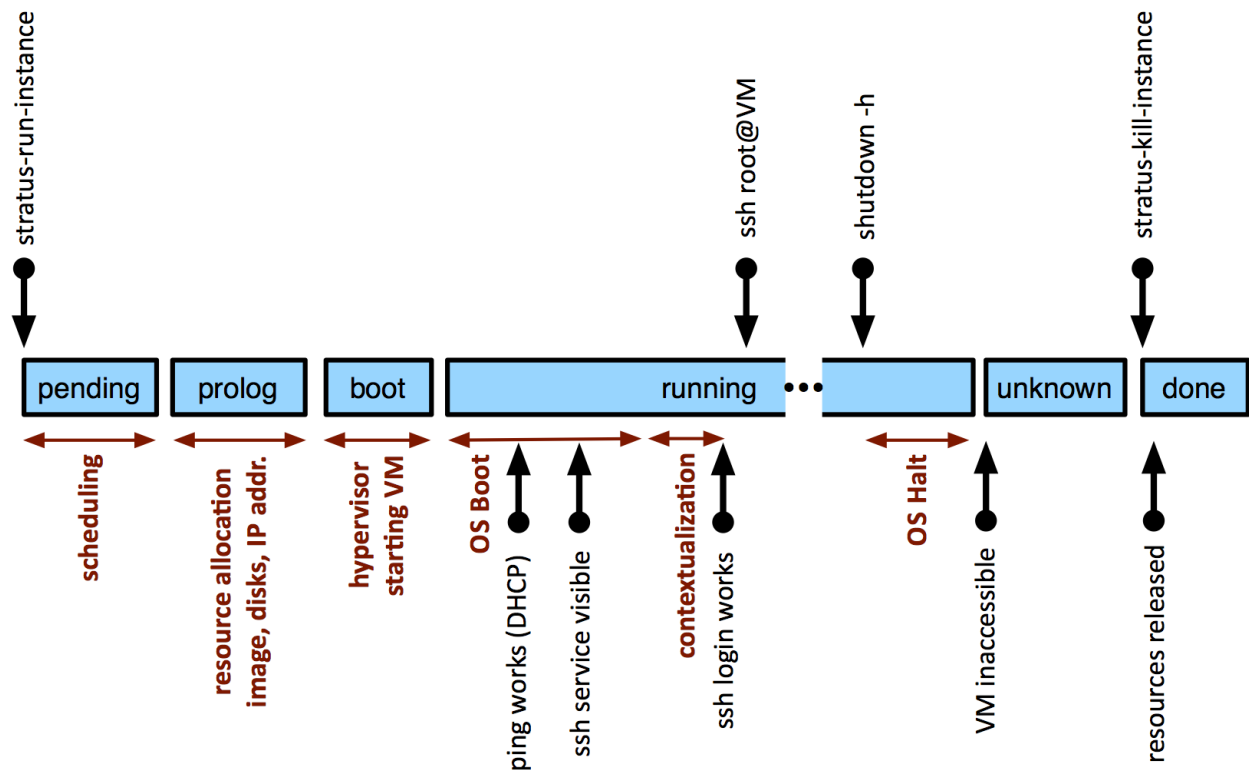


Fig. 2.4: Virtual machine timeline and states.

## 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.

### 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](mailto:images@stratuslab.eu)”. For our case, the ttylinux image has the identifier: “BN1EEkPiBx87\_uLj2-sdybSI-Xb”.

### 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.

### 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.

### 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.

### 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 forcibly 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 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.

### 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 configurations. You can obtain a list of these with the command:

```
$ stratus-run-instance --list-type
Type      CPU      RAM      SWAP
cl.medium  1 CPU    256 MB   1024 MB
cl.xlarge  4 CPU    2048 MB  2048 MB
ml.large   2 CPU    512 MB   1024 MB
* ml.small  1 CPU    128 MB   1024 MB
ml.xlarge  2 CPU    1024 MB  1024 MB
tl.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 (“`ml.small`”).

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



## 2.1.7 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.

### Create persistent disk

Before creating persistent disks (or volumes) <sup>1</sup>, 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
- 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
```

### 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
```

<sup>1</sup> “disk” and “volume” are used interchangeably.

```
users: 0
:: DISK d955fda6-bf9c-4aa8-abc4-5bbcd83021b
  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.

## 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

### Launch VM with a persistent disk attached

To Launch a VM we will be using the ttylinux image identifier GOaxJFdoEXvqAm9ArJgnZ0\_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
.....
```

### Format attached disk

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

### 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.

### 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
```

## 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-13fbef3c6cdc
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.

## 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-a248-0f0d22d948c2
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-3287cd0936e5
[ERROR] Service error: Not enough rights to delete disk
```

## 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.

## 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.

## 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.

## 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 \
  GJ5vp8gIxhZ1w1MQF16R6M1cNoq
```

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.

## 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.

## 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.

## 2.1.8 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.

### 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.

### 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.

### 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" \
--title "My first image" \
--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-239.1a1.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 drive 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 reasonable 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.

## 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 cl.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.

## 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.

## 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.

### OpenNebula and HEPIX Contextualization

To be done.

### 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).

### 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.

### 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.

## 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.

## 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 cl.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.

## 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.

## 2.1.9 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.

### 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.

### 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.

### 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).

### 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.

### 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.**

### 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.

### 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.

## 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`.

## 2.1.10 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.

### 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.

### 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.

### 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.

### 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](http://python.org) website. Be sure to select an installer for Python 2 and **not** Python 3.

### 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](http://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.

## 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.

### 2.1.11 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.

## Linux Operating Systems

### 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.

### 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.

### 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.

## Mac OS X

### Client Installation

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

### Generating an SSH Key Pair

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

### 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.

## Windows

### 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.

### 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**

## 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](#).

## SSH Agent

PuTTY supports the SSH agent functionality through the Pageant executable.

## 2.1.12 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.

## 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.

## 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.

## Windows

Download and install Java from the primary [Java website](#).

### 2.1.13 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.

## 2.2 Administrator Guide

This guide provides information for system administrators wanting to install and maintain a StratusLab cloud.

### 2.2.1 Preface

#### Target Audience

This guide provides information for system administrators wanting to install and maintain a StratusLab cloud.

Those interested in using the resources in a StratusLab cloud should instead consult the StratusLab User’s Guide.

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

#### Typographic Conventions

This guide uses several typographic conventions to improve the readability.

links	<a href="#">some link</a>
filenames	<code>\$HOME/.stratuslab/stratuslab-user.cfg</code>
commands	<code>stratus-run-instance</code>
options	<code>--version</code>

Table: 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.

## 2.2.2 Introduction

Describe all of the services and how they fit together in detail. Provide a detailed view of a cloud deployment.

How StratusLab fits into the various cloud service models. It is a IaaS!

Different types of deployments: public, private, and community.

### Organization

The guide starts with this introduction and then is followed by an installation tutorial showing all of the steps to install a minimal, two-machine StratusLab cloud infrastructure. The subsequent chapter provide more detail on various aspects of the software.

### Terminology

The older terminology, still used in most parts of this (and other) documents is the following:

- **Frontend:** A physical machine that runs the services for virtual machine management and storage services. This is the machine which is contacted by the cloud users for managing their cloud resources.
- **Node:** A physical machine that hosts virtual machines.

The newer terminology that has been adopted is:

- **Cloud Entry Point:** One or more physical machines that run the CIMI service. Users use this management interface to acquire, control, and release all of their cloud resources.
- **VM Host:** One or more physical machines that host virtual machines.
- **Storage Controller:** One or more physical machines that act as a gateway to a storage services.
- **Couchbase Node:** A physical machine that hosts a Couchbase instance running as part of a Couchbase cluster. This may be combined with either the Cloud Entry Points or the VM Hosts to minimize the number of physical machines used for the supporting services of the cloud.

The software and documentation is being gradually updated to consistently use this newer terminology.

## 2.2.3 Installation

This tutorial demonstrates how to install manually a StratusLab cloud infrastructure using the StratusLab system administrator command line utilities. A minimal StratusLab cloud consists of two physical machines, although additional machines may be necessary if the internet cannot be accessed.

### Installation Overview

The StratusLab distribution provides a simple command line client to install, configure and start the StratusLab Cloud services and components.

The default deployment has two types of machines:

1. **Front-End** - machine for VM management and storage services
2. **Node** - machine that hosts virtual machines

A quick overview of the procedure is:

1. Ensure all prerequisites are satisfied.

2. Define all of the StratusLab service parameters.
3. Install and configure the Front End, containing the VM management service (OpenNebula) and the storage management (Persistent Disk) service.
4. Install and configure the Node(s) via SSH from the Front End. By default **KVM** is used for the hypervisor on the Node(s).
5. Validate the installation by starting a virtual machine.

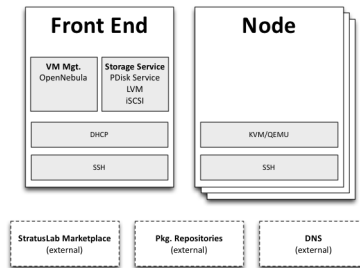


Fig. 2.5: Minimal StratusLab Cloud

## Prerequisites

### Physical Machines

This tutorial demonstrates a minimal installation of a StratusLab cloud on two physical machines. The physical machines should be relatively modern machines with the following **minimum** characteristics:

- 1 **64-bit multicore** CPU ( $\geq 4$  cores) with **VT-x extensions**
- 4 GB of RAM
- 200 GB local disk space

**The hardware virtualization extensions must be enabled in the BIOS on the “Node” machine.** Many vendors ship machines with these extensions disabled.

In general cloud infrastructures prefer “fat” machines, that is machines that have a maximum number of CPUs, RAM, and disk space as possible. This is because the maximum size of a single virtual machine is limited by the size of the largest physical machine.

### Operating System

Install a minimal version of **CentOS 6** on the two physical machines that will be used for the cloud infrastructure.

### Disable SELinux

The SELinux system must be disabled on **all of the machines**. Normally this is enabled by default. To disable SELinux, ensure that the file `/etc/selinux/config` has the following line:

```
SELINUX=disabled
```

**You must reboot the machine for this to take effect.**



## Python Version

The default version of Python installed with CentOS should be correct. StratusLab requires a version of Python 2 with a version **2.6 or later**. The StratusLab command line tools **do not work with Python 3**.

Verify that the correct version of Python is installed:

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

## Disk Configuration

StratusLab allows for a variety of storage options behind the persistent disk service. The tutorial uses the defaults using LVM and iSCSI.

**The machines must be configured to use LVM for the disk storage.**

The Front End must be configured with two LVM groups: one for the base operating system (~20 GB) and one for the StratusLab storage service (remaining space).

The “Node” machine can be configured with a single LVM group.

Below, we assume that the volume group names are “vg.01” for the operating system and “vg.02” for the StratusLab storage service. You can use other names, but then change the commands below as necessary.

## Package Repositories

The StratusLab installation takes packages from four yum repositories:

1. The standard CentOS repository,
2. The [EPEL 6](#) repository,
3. The [StratusLab repository](#), and
4. The [IGTF Root Certificates](#).

The configuration for the CentOS repository is done when the system is installed. The others require additional configuration.

To configure **both** the Front End and Node for the EPEL repository, do the following:

```
$ wget -nd http://mirrors.ircam.fr/pub/fedora/epel/6/i386/epel-release-6-8.noarch.rpm
$ yum install -y epel-release-6-8.noarch.rpm
```

This will add the necessary files to the `/etc/yum.repos.d/` directory.

To configure **both** the Front End and Node for the StratusLab repository, put the following into the file `/etc/yum.repos.d/stratuslab.repo`:

```
[StratusLab-Releases]
name=StratusLab-Releases
baseurl=http://yum.stratuslab.eu/releases/centos-6.2-v13.02/
gpgcheck=0
```

replacing the URL with the version you want to install.

Although not strictly necessary, it is advisable to clear all of the yum caches and upgrade the packages to the latest versions:

```
$ yum clean all
$ yum upgrade -y
```

This may take some time if you installed the base operating system from old media.

## DNS and Hostname

Ensure that the **hostname** is properly setup on the Front End and the Node. The DNS must provide both the forward and reverse naming of the nodes. This is required for critical services to start.

You can verify this on both the Front End and the Node with the command:

```
$ hostname -f
```

Set the hostname if it is not correct.

Throughout this tutorial we use the variables `$FRONTEND_HOST` (`$FRONTEND_IP`) and `$NODE_HOST` (`$NODE_IP`) for the Front End and Node hostnames (IP addresses), respectively. Change these to the proper names for your physical machines when running the commands.

## SSH Configuration

The installation scripts will automate most of the work, but the scripts require **password-less root access**:

- From the Front End to each Node and
- From the Front End to the Front End itself

Check to see if there is already an SSH key pair in `/root/.ssh/id_rsa*`. If not, then you need to create a new key pair **without a password**:

```
$ ssh-keygen -q
Enter file in which to save the key (/root/.ssh/id_rsa):
/root/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

Now ensure that you can log into the Front End from the Front End without needing a password. Do the following:

```
$ ssh-copy-id $FRONTEND_HOST
The authenticity of host 'onehost-5.lal.in2p3.fr (134.158.75.5)' can't be established.
RSA key fingerprint is e9:04:03:02:e5:2e:f9:a1:0e:ae:9f:9f:e4:3f:70:dd.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'onehost-5.lal.in2p3.fr,134.158.75.5' (RSA) to the list of known hosts.
root@onehost-5.lal.in2p3.fr's password:
Now try logging into the machine, with "ssh 'onehost-5.lal.in2p3.fr'", and check in:

    .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.
```

Do the same thing for the node:

```
$ ssh-copy-id $NODE_HOST
...
```

And verify that the password-less access works as expected.

```
$ ssh $FRONTEND_HOST

Last login: Mon May 27 14:26:29 2013 from mac-91100.lal.in2p3.fr
#
# exit
logout
Connection to onehost-5.lal.in2p3.fr closed.

$ ssh $NODE_HOST

Last login: Mon May 27 14:26:43 2013 from mac-91100.lal.in2p3.fr
#
# exit
logout
Connection to onehost-6.lal.in2p3.fr closed.
```

Now that SSH is properly configured, the StratusLab scripts will be able to install software on both the Front End and the Node.

## DHCP Server

A DHCP server must be configured to assign static IP addresses corresponding to known MAC addresses for the virtual machines. These IP addresses must be publicly visible if the cloud instances are to be accessible from the internet.

If an external DHCP server is not available, the StratusLab installation command can be used to properly configure a DHCP server on the Front End for the virtual machines.

This uses a DHCP server on the Front End.

## Network Bridge

A network bridge must be configured on the Node to allow virtual machines access to the internet. You can do this manually if you want, but the StratusLab installation scripts are capable of configuring this automatically.

This tutorial allows the installation scripts to configure the network bridge.

## Front End Deployment

### Deployment tool installation

The first step is to install the StratusLab system administrator command line client from the [StratusLab repository](#) on the Front End:

```
$ yum install -y stratuslab-cli-sysadmin
```

This will install the system administrator client and all of the necessary dependencies. You can verify that it is correctly installed by doing the following:

```
$ stratus-config --help

Usage: stratus-config [options] [key [value]]
If the [value] is not provided, the command returns the current value
of the key.
...
```

## Configuration file customization

The entire StratusLab Cloud is configured from a single configuration file `/etc/stratuslab/stratuslab.cfg`. This file contains many options, but only a few are required.

StratusLab ships with a default configuration file in the standard location and a reference configuration file located in `/etc/stratuslab/stratuslab.cfg.ref`.

To simplify viewing the configuration parameters and changing them, the `stratus-config` command can be used.

To list the content of the configuration, and show the differences between the `stratuslab.cfg` file and the reference configuration, you can use the `-k` or `--keys` option:

```
$ stratus-config -k
... lots of parameter values! ...
```

To change a value, specify the key and the new value. To view a single value, simply specify the key.

We will use this command to set the various configuration parameters below.

## VM Management Service

The parameters for the frontend and VM management:

```
$ stratus-config frontend_system centos
$ stratus-config frontend_ip $FRONTEND_IP
```

## Storage Service

Similar parameters must also be set for the Persistent Disk service.

For this tutorial, this service is installed on the Front End, so the same IP address should be used.

```
$ stratus-config persistent_disk_system centos
$ stratus-config persistent_disk_ip $FRONTEND_IP
$ stratus-config persistent_disk_merge_auth_with_proxy True
```

The Persistent Disk service and the Nodes communicate using a strategy defined by the `persistent_disk_storage` and `persistent_disk_share` parameters. The default values (“lvm” and “iscsi”, respectively) will be used for this tutorial.

One needs to specify what device will be used for the physical storage for the Persistent Disk service:

```
$ stratus-config persistent_disk_lvm_device /dev/vg.02

# Provide detailed parameters for storage backend plugins.
# (NOTE: The opening and closing single quotes!)
$ stratus-config persistent_disk_backend_sections '
[% (persistent_disk_ip)s]
    type=LVM
    volume_name = /dev/vg.02
    lun_namespace = stratuslab
    volume_snapshot_prefix = pdisk_clone
    initiator_group =
'
```

If you’ve used another name for the LVM volume group, then change the above command.

## Network configuration

Use the frontend as the general gateway for the cloud:

```
$ stratus-config default_gateway $FRONTEND_IP
```

Set the IP and mac addresses for virtual machines:

```
$ stratus-config one_public_network_addr \
    134.158.xx.yy 134.158.xx.yy 134.158.xx.yy

$ stratus-config one_public_network_mac \
    0a:0a:86:9e:49:2a 0a:0a:86:9e:49:2b 0a:0a:86:9e:49:2c
```

In this example, the Front-End is configured on IP address \$FRONTEND\_IP and three IP/MAC address pairs are defined for virtual machines.

**You must use the real values for the Front End IP addresses and for the range of addresses you will use for the virtual machines.**

More network parameters are described in the “one-network” section in the reference configuration file.

## DHCP Configuration

Allow the script to automatically configure and start the DHCP server on the Front End. Do the following:

```
$ stratus-config dhcp True
$ stratus-config dhcp_subnet 134.158.75.0
$ stratus-config dhcp_netmask 255.255.255.0
$ stratus-config dhcp_lease_time 3600

$ stratus-config dhcp_one_public_network True
$ stratus-config dhcp_one_local_network_routers $FRONTEND_IP
$ stratus-config dhcp_one_local_network_domain_name lal.in2p3.fr
$ stratus-config dhcp_one_local_network_domain_name_servers \
    134.158.91.80, 134.158.88.149
```

Use **your** values for these parameters!

## Finalize Front End Installation

Now that we have defined all of the configuration parameters, you can now do the full Front End installation by issuing the following command:

```
$ stratus-install -vv
```

To get more details on what the command is (because of curiosity or errors), use the option `-v`, `-vv`, or `-vvv`.

If you run into errors, the `stratus-install` command can simply be rerun after adjusting the configuration parameters.

## Node Deployment

The deployment of the StratusLab Nodes is done from the Front End, thus, **all the commands below should be run from the Front End.**

To add a Node to the cloud, specify the Linux distribution of the machine and indicate that the bridge should be configured:

```
$ stratus-config node_system centos
```

Request the automatic configuration of the network bridge:

```
$ stratus-config node_bridge_configure True
$ stratus-config node_bridge_name br0
$ stratus-config node_network_interface eth0
```

Check carefully the name of the interface on the node!

Invoke installation by

```
stratus-install -vv -n $NODE_IP
```

As before, you can increase the verbosity level by adding the option `-v` or `-vv`.

## User Configuration

At this point, you have both the Front End and one Node installed. This is a functional installation, but you have not yet authorized any users for the cloud. Here we will create a new StratusLab user account. Note that StratusLab accounts are independent of the Unix accounts on the machine itself.

Add the following line to the end of the file `/etc/stratuslab/authn/login-pswd.properties`.

```
$ cat >> /etc/stratuslab/authn/login-pswd.properties
sluser=slpass,cloud-access
```

This creates a new StratusLab user ‘sluser’ with a password ‘slpass’. The group ‘cloud-access’ is mandatory for the user to have access to the cloud services. (Crypted or hashed password values are also allowed in the configuration.)

The StratusLab distribution supports other authentication methods (LDAP, X509 certificates, X509 proxies, etc.), but a username/password pair is the simplest for this tutorial.

## StratusLab Client

Now we will test that the cloud functions correctly by starting a new virtual machine instance and logging into it. We’ll test the cloud service from a normal Unix user account on the Front End.

First, ensure that the StratusLab user client is installed on the machine. Do the following as root:

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

It is very likely that the user client commands are already installed.

(Note: For normal client installations, it is strongly recommended to use `pip` or `easy_install` with `virtualenv`. See the usual [client installation instructions](#).)

Now create a normal Unix user for testing:

```
$ adduser sluser
```

Now log in as the user and setup the account for using StratusLab. An SSH key pair is required to log into your virtual machines and the client requires that a complete client configuration file.

Log in as the user and create an SSH key pair. This is similar to the process used for the root account on the machine.

```
$ su - sluser
$ ssh-keygen -q
...
```

Now copy the reference configuration file into place and edit the parameters.

```
$ mkdir ~/.stratuslab
$ cd ~/.stratuslab
$ cp /etc/stratuslab/stratuslab-user.cfg.ref ~/.stratuslab/stratuslab-user.cfg
$ vi ~/.stratuslab/stratuslab-user.cfg # endpoint, username, password
```

You will need to set the “endpoint”, “username”, and “password” parameters in this file. For the “endpoint” use the hostname or IP address of your Front End. For the “username” and “password” use “sluser” and “slpass”, respectively.

Everything should be setup now. So try deploying a virtual machine. You can look in the Marketplace to find an interesting machine to deploy. We’ll use a ttylinux image here. This is a micro distribution that boots very quickly and is ideal for tests.

```
# Deploy a ttylinux virtual machine.
$ stratus-run-instance BN1EEkPiBx87_uLj2-sdybSI-Xb

::::::::::::::::::::::::::
:: Starting machine(s) ::
::::::::::::::::::::::::::
:: Starting 1 machine
:: Machine 1 (vm ID: 1)
Public ip: 134.158.75.42
:: Done!
```

Check the status of the machine as it starts:

```
# Check its status. Pending -> not yet assigned to a Node
$ stratus-describe-instance
id  state      vcpu memory    cpu% host/ip                name
1   Pending    1     0          0    vm-42.lal.stratuslab.eu one-1

# Check again. Prolog -> resources for VM are being initialized
$ stratus-describe-instance
id  state      vcpu memory    cpu% host/ip                name
1   Prolog     1     0          0    vm-42.lal.stratuslab.eu one-1

# Check again. Running -> hypervisor has started machine
$ stratus-describe-instance
id  state      vcpu memory    cpu% host/ip                name
1   Running    1     0          0    vm-42.lal.stratuslab.eu one-1
```

When the machine reaches the ‘running’ status, the virtual machine is running in the hypervisor on the Node. It will probably take some additional time for the operating system to boot.

Verify that the machine has fully booted and is accessible from the network:

```
# Ping the virtual machine to see if it is accessible.
$ ping vm-42.lal.stratuslab.eu
PING vm-42.lal.stratuslab.eu (134.158.75.42) 56(84) bytes of data.
From onehost-5.lal.in2p3.fr (134.158.75.5) icmp_seq=2 Destination Host
  Unreachable
...
From onehost-5.lal.in2p3.fr (134.158.75.5) icmp_seq=8 Destination Host
  Unreachable
64 bytes from vm-42.lal.stratuslab.eu (134.158.75.42): icmp_seq=9
```

```
tty=64 time=1.44 ms
...

# Now login to the machine as root.
$ ssh root@vm-42.lal.stratuslab.eu

The authenticity of host 'vm-42.lal.stratuslab.eu (134.158.75.42)'
can't be established.
RSA key fingerprint is
6a:bd:f7:2d:b6:82:39:61:e6:ca:3f:c7:61:9d:72:31.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'vm-42.lal.stratuslab.eu,134.158.75.42'
(RSA) to the list of known hosts.

#           # <- we're logged into the ttylinux virtual machine
# exit     # just logout of the session
logout
Connection to vm-42.lal.stratuslab.eu closed.
```

Now the machine can be terminated:

```
$ stratus-kill-instance 1
```

Going through the full lifecycle of a machine shows that all of the services are working.

## Conclusions

You've successfully installed a minimal StratusLab cloud. You can checkout the [documentation](#) to see what other configuration parameters are available or try the user tutorials to discover more of the StratusLab services.

You can get help on the installation or use of StratusLab through the [support mailing list](#). You can also report bugs and provide feedback on the same list.

## 2.2.4 Network Configuration

The StratusLab network services do not make dynamic changes to the network configuration, greatly simplifying the integration of the cloud into the local computing environment.

### Network Services

The StratusLab network configuration relies on standard networking services and resources already available on your site.

#### IP Addresses

The StratusLab configuration parameters allow for three classes of IP addresses: public, local, and private. The public IPs are accessible from the WAN, local IPs are only visible within the cloud infrastructure itself, and private IPs are only visible on the physical machine hosting the virtual machine.

Each class you want to configure must have an associated range of free IP addresses. The number of addresses for each range must be large enough to cover the number of virtual machines you expect to be running simultaneously.

At least one class of IP addresses must be configured. This is usually the “public” class, which the StratusLab client uses by default.



## DHCP

There must be a DHCP server available that will assign IP addresses to virtual machines when they start. This DHCP server maintains a mapping between the MAC addresses (managed by OpenNebula) and the IP addresses. The DHCP server must be configured to send all of the standard networking parameters (gateway, broadcast, domain, etc.) to the DHCP clients.

## DNS

All of the IP addresses for the virtual machines must have associated hostnames. These hostname must be registered in a DNS server, where both forward and reverse lookups work correctly.

## Isolation

A StratusLab cloud can work correctly with a single physical network shared between virtual machines and the physical machines running the StratusLab services. For a private cloud, this is probably sufficient.

For clouds with a larger user base, it is better to isolate the network for the virtual machines from that for the physical machines. This can either be done with VLANs or by physically segmenting the network (assuming that you have multiple network interface cards).

For large cloud infrastructures, it is also worthwhile to segment the traffic to the storage service. This reduces interference between high-bandwidth data access and the lower-bandwidth for control messages.

## Services and Ports

All of the StratusLab services now sit behind a nginx web proxy. This allows all of the services on a particular machine to share port 443 as well as the server certificate. This also means that port 443 is the only port that must be open to the WAN.

Nonetheless, it is useful to know what ports and users are used by the StratusLab services when debugging problems. The following table summarizes those ports. The name of the *init.d* script to control the service is the same as the name in the first column.

oned (OpenNebula)	oneadmin	localhost:2634
cimi	slcimi	localhost:9200
registration	slreg	localhost:9202
marketplace	slmkpl	localhost:9204
pdisk	root	localhost:9206
one-proxy	slauth	localhost:9208

## 2.2.5 Authentication and Authorization

### Authentication Methods

StratusLab supports a wide range of different authentication methods and account storage options, allowing it to fit into a wide range of sites with pre-existing authentication systems.

All of the configuration files for authentication are located in the directory `/etc/stratuslab/authn`. The main configuration file controlling the active authentication methods and their parameters is `login.conf`. Properties files with username/password or certificate information are also located in this directory.

## **Username and Password**

Users can be identified through username/password pairs. Accounts associated with these credentials can be stored in:

- A java properties file or
- An LDAP server

If you use an LDAP server, the StratusLab Registration service can be used to allow users to register for an account.

## **X509 Credentials**

Users can also be identified via client X509 credentials. Raw X509 certificates, RFC3820 certificate proxies, or VOMS proxies can all be used to connect to and to authenticate with the StratusLab services.

As above, these credentials can be authorized in:

- A java properties file or
- An LDAP server

The registration service can also manage the required X509 Distinguished Names (DNs) if using an LDAP server.

## **Authorization**

Apart from the super user, users can only see and control resources that they create themselves.

## **Registration Service**

StratusLab provides an optional service, the “Registration Service”, that allows users to register for an account on your cloud infrastructure. Through a web-accessible interface, users can register for an account and update the information associated their accounts.

The registration workflow proceeds through the following stages:

1. User submits form with required information,
2. Service sends email validation message to user,
3. User confirms email address by visiting confirmation link,
4. Service sends account validation request to administrator,
5. Administrator accepts or rejects account request, and
6. An email is sent to the user with the decision.

When users update their email addresses, the new email address is also validated through an email and confirmation link.

The user information is stored in a separate LDAP server. The registration service must have full access to that server, so that it can manage the user entries.

To use the accounts managed by the Registration Service on your cloud infrastructure, you must configure the authentication service to use the LDAP server as a source of information. Users who supply a certificate DN, can also authenticate with the cloud using their certificate.

## 2.2.6 Running a Cloud

Information on running the cloud as a service.

### Service Logging Information

Where are the service logs?

### Monitoring Activity

Where to find monitoring information.

### Security Concerns

Various topics.

## 2.2.7 Troubleshooting

Information about common problems.

## 2.2.8 Architecture

The architecture of the StratusLab software has evolved to make the overall system more scalable and more robust. At the center of the system is a distributed database that removes single points-of-failure in the system and permits redundant deployments of StratusLab service components for better reliability.

StratusLab also now exposes the [CIMI interface](#)—a standard from [DMTF](#) that provides a coherent, unified API for all of the StratusLab cloud resources and that follows the usual REST patterns. It simplifies programmatic access to the cloud and provides a good foundation for browser-based access.

### StratusLab Components

The diagram shows a high-level view of the various components in the StratusLab architecture. The distributed database is at the core of the system. On the user-facing side are machines that provide the CIMI interface to the system. On the service side are a set of controllers for different types of resources (storage, VMs, etc.). Behind the controllers are the physical resources used by the cloud.

The distributed database contains the complete state of the cloud. Consequently, the CIMI interfaces and the various controllers can be completely stateless, allowing redundant instances to be deployed as necessary.

As all of the communication between controllers takes place through the database, specialized controllers can be added to the cloud infrastructure easily, such as:

- Different type of storage (backed up, shared, fast, etc.)
- Support for Linux containers as well as virtual machines
- Dynamic network configurations

The controllers react to jobs and resources in the database, and update those entries when completing tasks or making adjustments to resources.

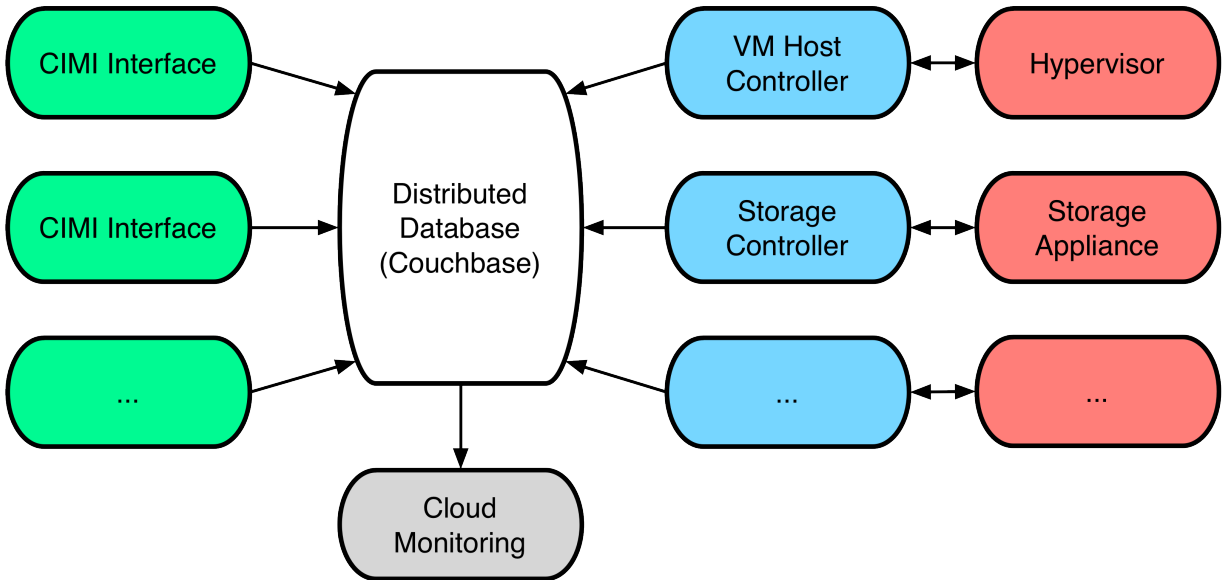


Fig. 2.6: StratusLab Architecture

## Service Configuration

Generally, the configuration of the StratusLab cloud services also resides within the database. This allows deployment of redundant services while maintaining a consistent configuration of those services across machines.

The configuration information within the database is split into a series of JSON-formatted documents. Each document has an document identifier of the form `ServiceConfiguration/service-instance`, where the “service” corresponds to the name of the service and the “-instance” is optional and used only if the configuration of an instance differs from the general service configuration.

Most configuration exists within the database, but there are two notable exceptions: third-party services and the Couchbase access parameters.

To minimize the StratusLab development effort, third-party services used by StratusLab are not modified to use the database. Consequently, these continue to use configuration files on the local file system. One example are the certificate authorities used for PKI authentication in the European Grid Infrastructure. When deploying multiple service instances, these files must be coordinated between physical machines.

The StratusLab services must discover the contact parameters for the Couchbase database. This information resides in a file on each node (`/etc/stratuslab/couchbase.cfg`). If this file doesn’t exist, then services will use the default bucket and assume that the local machine is a member of the Couchbase cluster.

The standard “ini” format is used for the file:

```
[DEFAULT]
host=localhost
bucket=default
username=default
password=

[service]
config=ServiceConfiguration/service-instanceX
```

This example shows the default values. You can define a different bucket and location for the database. However, the same database must be shared by all of the StratusLab cloud services. **If changes are made to this file, then the**

**StratusLab service must be restarted.**

The example also shows the possibility of providing instance-specific configuration documents for services. Generally, these should not be needed but are available for allow for specialized service configurations.

Despite the exceptions, having the majority of configuration information in the database will make configuring and maintaining the services simpler for system administrators. Generally, it is just a matter of updating a document in the database to change the behavior of the cloud infrastructure.

## 2.2.9 Planning the Deployment

The components of the StratusLab distribution are quite modular and allow for a variety of different deployment layouts. This chapter provides some advice when considering the deployment of StratusLab in your data center.

The chapter concludes with a description of the minimal deployment of StratusLab on two machines. This deployment serves as a good testbed for understanding the installation process and how the cloud services work.

### Your Cloud Users

When considering your deployment, you must consider who are your intended users. The level of trust you have in those users will impact further decisions on the network configuration, service layout, and authentication methods.

For clouds at the “Infrastructure as a Service” level, like StratusLab, there are generally three types of cloud deployments: private, community, and public.

Private cloud deployments target a limited set of known and trusted users. An example is using a cloud deployment to provide a flexible infrastructure for an institute’s services. These types of clouds are often run by and used by the same system administrators. Because of the high-level of trust in the users, less needs to be done to isolate cloud services from the cloud’s virtual machines.

On the other end of the spectrum are public cloud infrastructures. These target external and possibly unknown users outside of the institute’s administrative domain. These generally require a higher level of isolation of service and more fine-grained network segmentation.

Between the two are “community” clouds. These function like public clouds but have a more limited set of users. A cloud infrastructure destined for scientists at a number of collaborating institutes is a good example of a community cloud infrastructure.

### Network Configuration

StratusLab puts very few constraints on the network configuration supporting the cloud infrastructure. At its simplest, it can work on a single network tying together all cloud services and resources. It can also be made to work on complicated network configurations with multiple VLANs, protocols, etc.

The only network constraints for StratusLab are:

- Range of addresses to allocate to virtual machines,
- DHCP server configured to serve those addresses, and
- DNS configured with reverse-DNS lookups for those addresses.

The range of addresses must be IPv4 addresses, but each address can also have an IPv6 address associated with it.

There are three different types of network traffic on a StratusLab cloud infrastructure:

- Control messages between the cloud services (and through the Couchbase database),
- Network access to virtual machines in the cloud, and

- Traffic between nodes hosting virtual machines and physical resources (e.g. storage).

The level of isolation between these different types of traffic depends on your type of cloud deployment and your desire for a secure infrastructure.

For a private cloud infrastructure, mixing these three types of traffic on the same network doesn't pose any particular problem and is the easiest to configure.

However, for a public cloud infrastructure it is a good idea to separate the three types of traffic on separate VLANs. Even better, physical segregation of the virtual machine traffic from the other traffic prevents any possible compromise of the cloud services from the running virtual machines. This is obviously more complicated to configure. You must weight the additional complexity against the potential threats from your users.

## Mapping Services to Machines

There are three classes of services that need to be mapped to physical machines: Couchbase instances, CIMI interfaces, and resource controllers.

The usual mapping for Couchbase instances and CIMI interfaces is to deploy these on the same physical machine. For scalability and redundancy, two or more such physical machines are deployed.

Resource controllers are generally deployed on the same physical nodes that provide the underlying resources. For example, the VM controller is deployed directly on the machine(s) running the hypervisors. This arrangement minimizes the number of physical machines that need to be deployed.

You can alter the mapping as necessary to adapt the needs of your data center. For example, you may have resources (like a storage appliance) that require having the controller hosted on a different machine than the resource it is controlling.

## Minimal Deployment

The examples in this guide use a minimal deployment of two physical machines. One machine (the “cloud entry point”) contains the Couchbase database and the CIMI interface. The other contains all of the resources controllers and resources used for the cloud. A single network for all traffic types is assumed for simplicity in this minimal deployment. The following diagram summarizes this deployment.

### 2.2.10 Prerequisites

#### Physical Machines

This tutorial demonstrates a minimal installation of a StratusLab cloud on two physical machines. The physical machines should be relatively modern machines with the following **minimum** characteristics:

- 1 **64-bit multicore** CPU ( $\geq 4$  cores) with **VT-x extensions**
- 4 GB of RAM
- 200 GB local disk space

**The hardware virtualization extensions must be enabled in the BIOS on the “Node” machine.** Many vendors ship machines with these extensions disabled.

In general cloud infrastructures prefer “fat” machines, that is machines that have a maximum number of CPUs, RAM, and disk space as possible. This is because the maximum size of a single virtual machine is limited by the size of the largest physical machine.

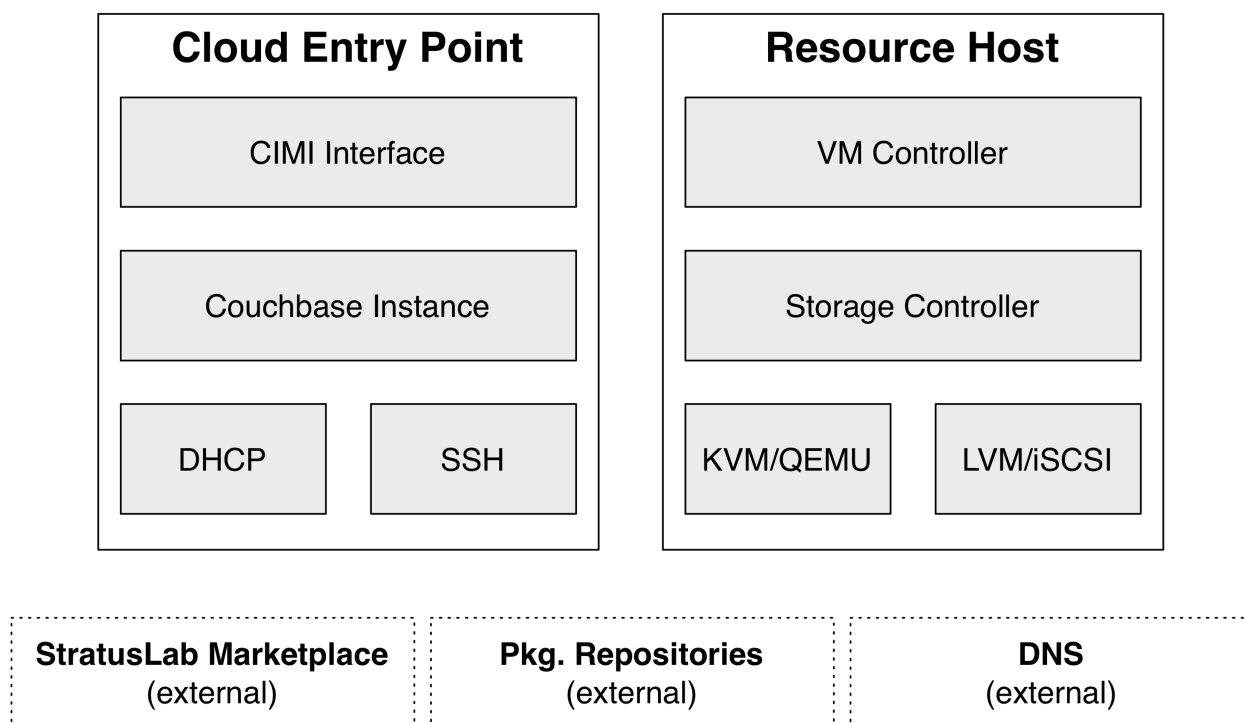


Fig. 2.7: Minimal StratusLab Deployment

## Operating System

Install a minimal version of [CentOS 6][centos] on the two physical machines that will be used for the cloud infrastructure.

## Disable SELinux

The SELinux system must be disabled on **all of the machines**. Normally this is enabled by default. To disable SELinux, ensure that the file `/etc/selinux/config` has the following line:

```
SELINUX=disabled
```

**You must reboot the machine for this to take effect.**

## Python Version

The default version of Python installed with CentOS should be correct. StratusLab requires a version of Python 2 with a version **2.6 or later**. The StratusLab command line tools **do not work with Python 3**.

Verify that the correct version of Python is installed:

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

## Disk Configuration

StratusLab allows for a variety of storage options behind the persistent disk service. The tutorial uses the defaults using LVM and iSCSI.

**The machines must be configured to use LVM for the disk storage.**

The Front End must be configured with two LVM groups: one for the base operating system (~20 GB) and one for the StratusLab storage service (remaining space).

The “Node” machine can be configured with a single LVM group.

Below, we assume that the volume group names are “vg.01” for the operating system and “vg.02” for the StratusLab storage service. You can use other names, but then change the commands below as necessary.

## Package Repositories

The StratusLab installation takes packages from four yum repositories:

1. The standard CentOS repository,
2. The [EPEL 6][epel] repository,
3. The [StratusLab repository][stratuslab-yum], and
4. The [IGTF Root Certificates][igtf-certs].

The configuration for the CentOS repository is done when the system is installed. The others require additional configuration.

To configure **both** the Front End and Node for the EPEL repository, do the following:

```
$ wget -nd http://mirrors.ircam.fr/pub/fedora/epel/6/i386/epel-release-6-8.noarch.rpm
$ yum install -y epel-release-6-8.noarch.rpm
```

This will add the necessary files to the `/etc/yum.repos.d/` directory.

To configure **both** the Front End and Node for the StratusLab repository, put the following into the file `/etc/yum.repos.d/stratuslab.repo`:

```
[StratusLab-Releases]
name=StratusLab-Releases
baseurl=http://yum.stratuslab.eu/releases/centos-6.2-v13.02/
gpgcheck=0
```

replacing the URL with the version you want to install.

Although not strictly necessary, it is advisable to clear all of the yum caches and upgrade the packages to the latest versions:

```
$ yum clean all
$ yum upgrade -y
```

This may take some time if you installed the base operating system from old media.

## DNS and Hostname

Ensure that the **hostname** is properly setup on the Front End and the Node. The DNS must provide both the forward and reverse naming of the nodes. This is required for critical services to start.

You can verify this on both the Front End and the Node with the command:



```
$ hostname -f
```

Set the hostname if it is not correct.

Throughout this tutorial we use the variables `$FRONTEND_HOST` (`$FRONTEND_IP`) and `$NODE_HOST` (`$NODE_IP`) for the Front End and Node hostnames (IP addresses), respectively. Change these to the proper names for your physical machines when running the commands.

## SSH Configuration

The installation scripts will automate most of the work, but the scripts require **password-less root access**:

- From the Front End to each Node and
- From the Front End to the Front End itself

Check to see if there is already an SSH key pair in `/root/.ssh/id_rsa*`. If not, then you need to create a new key pair **without a password**:

```
$ ssh-keygen -q
Enter file in which to save the key (/root/.ssh/id_rsa):
/root/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

Now ensure that you can log into the Front End from the Front End without needing a password. Do the following:

```
$ ssh-copy-id $FRONTEND_HOST
The authenticity of host 'onehost-5.lal.in2p3.fr (134.158.75.5)' can't be established.
RSA key fingerprint is e9:04:03:02:e5:2e:f9:a1:0e:ae:9f:9f:e4:3f:70:dd.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'onehost-5.lal.in2p3.fr,134.158.75.5' (RSA) to the list of known hosts.
root@onehost-5.lal.in2p3.fr's password:
Now try logging into the machine, with "ssh 'onehost-5.lal.in2p3.fr'", and check in:

    .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.
```

Do the same thing for the node:

```
$ ssh-copy-id $NODE_HOST
...
```

And verify that the password-less access works as expected.

```
$ ssh $FRONTEND_HOST

Last login: Mon May 27 14:26:29 2013 from mac-91100.lal.in2p3.fr
#
# exit
logout
Connection to onehost-5.lal.in2p3.fr closed.

$ ssh $NODE_HOST

Last login: Mon May 27 14:26:43 2013 from mac-91100.lal.in2p3.fr
#
```

```
# exit
logout
Connection to onehost-6.lal.in2p3.fr closed.
```

Now that SSH is properly configured, the StratusLab scripts will be able to install software on both the Front End and the Node.

## DHCP Server

A DHCP server must be configured to assign static IP addresses corresponding to known MAC addresses for the virtual machines. These IP addresses must be publicly visible if the cloud instances are to be accessible from the internet.

If an external DHCP server is not available, the StratusLab installation command can be used to properly configure a DHCP server on the Front End for the virtual machines.

This uses a DHCP server on the Front End.

## Network Bridge

A network bridge must be configured on the Node to allow virtual machines access to the internet. You can do this manually if you want, but the StratusLab installation scripts are capable of configuring this automatically.

This tutorial allows the installation scripts to configure the network bridge.

### 2.2.11 Couchbase

The StratusLab architecture uses a distributed database at the core to hold the full state of the cloud. This simplifies all of the other components of the system by allowing them to be stateless and by acting as a means for coordinating the components of the system.

StratusLab uses [Couchbase](#) for this distributed database. It was chosen because of its ability to store and to index JSON-formatted documents efficiently, closely matching the needs of managing cloud resources through the CIMI data model. It is easy to deploy for small database, while retaining the ability to scale to very large systems.

Because all other StratusLab components rely on having access to the Couchbase database, it is the first component that must be installed.

## Service Overview

Couchbase is available as a standard RPM package from the Couchbase website. The company distributes two versions of the database:

- The **community version** is open source and released under the Apache 2 license (the same as for StratusLab). StratusLab uses this version for all of its own deployments and tests. For convenience, the RPM package for Couchbase is included in the StratusLab yum repository.
- The **enterprise version** requires the purchase of a commercial license for anything other than small test deployments. The commercial support may be interesting for mission-critical cloud infrastructures. StratusLab should work with the enterprise version, but does not require it. The enterprise version is not systematically tested by the StratusLab collaboration.

init.d script	couchbase-server
language	erlang
APIs	Java, Python, and C
log file(s)	/opt/couchbase/var/lib/couchbase/logs/*
initial password	/opt/couchbase/cluster-password.txt
8091	web administration port
8092	Couchbase API port
11209*, 11210	internal cluster ports
4369*	Erlang Port Mapper
21100-21199*	Node data exchange

Table: Couchbase Server Characteristics

All of the listed ports must be open to communication between the nodes participating in the Couchbase cluster. All ports except those with an asterisk must be open to cloud services accessing the database.

Couchbase is written in Erlang, but StratusLab uses the Java and Python APIs to access the database. The Python API depends on the C API, so it must also be installed.

## Installation and Configuration

The usual service mapping puts the Couchbase instances on the “cloud entry point” nodes, along with the CIMI interface. The instances on these nodes form the Couchbase cluster for the full StratusLab cloud. The minimal deployment has only one “cloud entry point” node and hence only one Couchbase instance.

Production StratusLab deployments should have more than one Couchbase instance in the Couchbase cluster for reliability and redundancy. Although the default location for these instances is on the “cloud entry point” nodes, you can deploy them on dedicated nodes or other cloud services nodes, if needed.

The installation of Couchbase consists of installing the RPM package and then initializing a Couchbase cluster. The StratusLab installation commands automate the process.

Log into the node that will function as the “cloud entry point” node for your cloud infrastructure as “root”. Verify that all of the prerequisites detailed in the previous chapter are satisfied.

Install the StratusLab system administrator command line interface (CLI). This installs the commands that simplify and automate the installation of all of the StratusLab components. Assuming that you have already configured the machine for using the StratusLab yum repository (see “Prerequisites”), this should be as simple as:

```
$ yum install -y stratuslab-cli-sysadmin
```

Once this completes, you should have a set of StratusLab commands in your path. All of the StratusLab commands start with the prefix `stratus-`. You may want to look at the help for the `stratus-install` command:

```
$ stratus-install --help

Usage: stratus-install [options]

Install selected services of the StratusLab cloud distribution.
...
```

This is the command that we will use to automate the installation of the cloud services.

We will now use this command to install and initialize the Couchbase database on the machine. Do the following:

```
$ stratus-install --couchbase

...
Starting couchbase-server          [OK]
```

You can get more detailed output if you add the `-vvv` option. This command installs the necessary packages and then sets up the database. The last line should indicate that Couchbase has been started; if successful you will see an “OK” indication.

---

**Note:** The installation creates an administrator account for the database called ‘admin’. The randomly generated password for this account is available in `/opt/couchbase/cluster-password.txt`.

---

## Verification

The easiest way to verify that Couchbase has been correctly installed is to use the Couchbase web interface. This interface is available on port 8091, but by default, is only accessible locally on the machine. To view it remotely you will need to tunnel to the machine:

```
$ ssh -L2000:cep.example.org:8091 -N root@cep.example.org
```

in a separate terminal window. You can then connect to the interface on the “<http://localhost:2000/>” URL. **You must use the “admin” account with the generated password to log in.**

If everything has been installed correctly, you should see a display similar to the screenshot below.

### 2.2.12 CIMI

StratusLab uses **CIMI** as its native user interface. CIMI is a standard from **DMTF** that provides a coherent, unified, RESTful API. The API covers all of the StratusLab cloud resources, including:

- Virtual machines
- Volumes for data storage
- Machine images or appliances

StratusLab has extended the interface to provide information to cloud users in “ServiceMessage” resources, configuration of the StratusLab services, and user management.

As the CIMI specification follows the usual REST patterns, the standard CRUD actions (create, read, update, and delete) are represented by:

- Create: HTTP POST to a resource collection URL,
- Read: HTTP GET to a resource URL,
- Update: HTTP PUT to a resource URL, and
- Delete: HTTP DELETE to a resource URL.

All of the resources are represented as JSON documents and contain metadata to allow users to understand what operations they can perform on the resource.

## Service Overview

The CIMI service is the only StratusLab service accessible by users on a cloud infrastructure. All interactions between the user and the underlying resources occur through this service. This service also handles all of the user authentication for the cloud, passing verified authentication information to the underlying services.

The service is written in **clojure** and uses the **Ring** and **Compojure** frameworks for implementing the web service. The **Friend** framework is used for authentication.

The characteristics of the service are summarized in the following table.

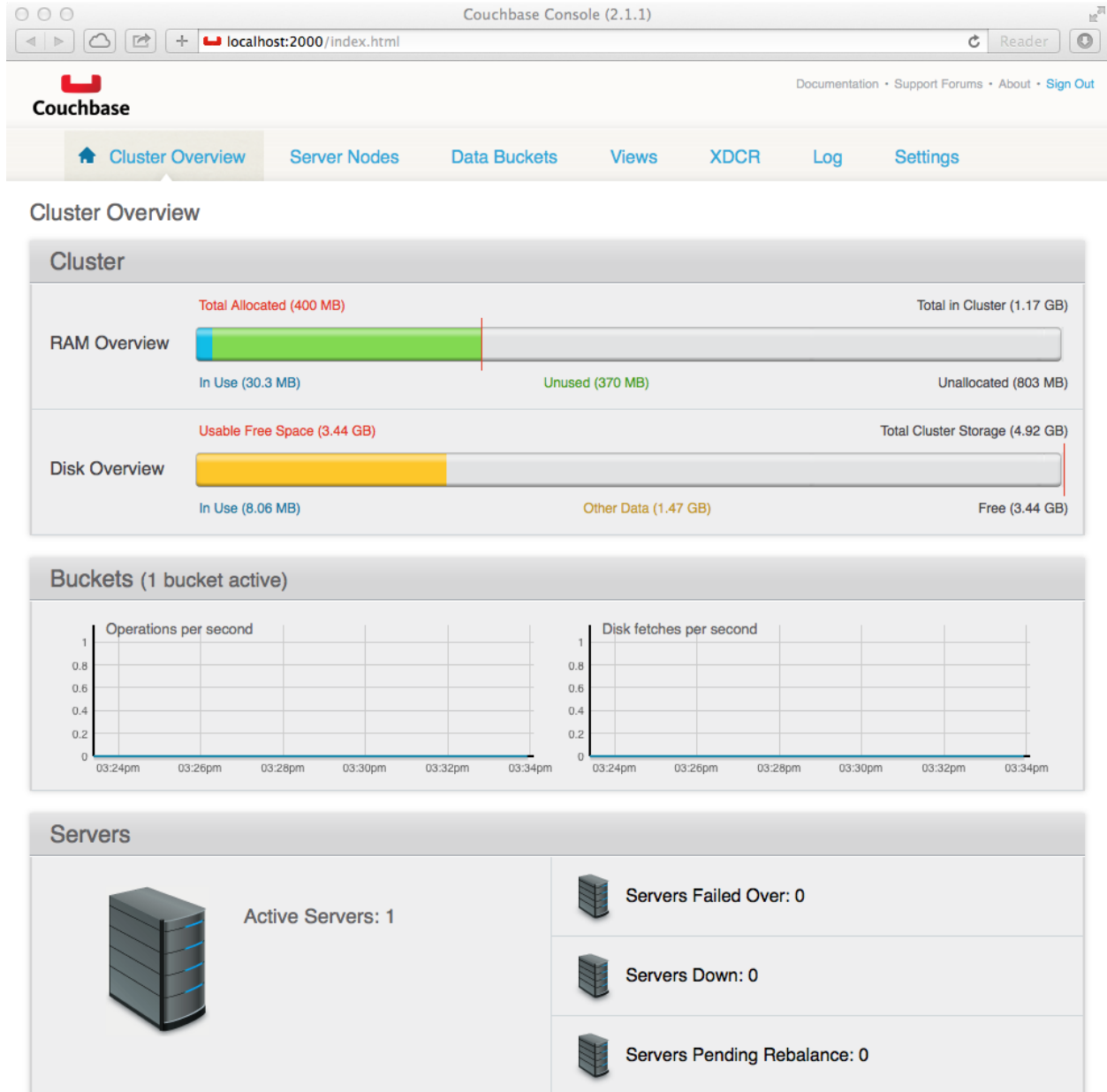


Fig. 2.8: Couchbase Management Console

init.d script	cimi
language	clojure (lisp on the Java VM)
APIs	REST, python, and libcloud
log file(s)	/var/log/stratuslab/cimi/*
initial password	see <code>cimi.log</code> log file
9200	open only on localhost (HTTP)

Table: CIMI Server Characteristics

## Installation

As for all of the StratusLab services, the installation consists of the installation of an RPM package followed by configuration of the service. This process is automated via the `stratus-install` command.

**Warning:** The CIMI service depends on the Couchbase database. You must install and configure Couchbase before trying to install the CIMI service.

Logged in as “root” on the “cloud entry point” machine, execute the following command to install the CIMI service:

```
$ stratus-install --cimi
```

This should complete without error and start the cimi service. You can check that it is running with the command:

```
$ service cimi status

START_INI      = /opt/stratuslab/cimi/start.ini
JETTY_HOME     = /opt/stratuslab/cimi
...
Jetty running pid=2099
```

This gives a lot of information about the service configuration and ends with giving the PID of the running service. If this is not running, then no PID will be given.

## Configuration

### nginx

The CIMI service runs behind an nginx proxy. The installation of nginx and the necessary configuration will all be done automatically by the StratusLab RPM packages.

The default installation will generate a self-signed certificate for the service. If you want to use a certificate signed by an accredited certificate authority, you must install it in the `/etc/stratuslab/nginx-proxy/` directory. The certificate and key must be in the `cert.pem` and `cert.key` files, respectively.

### Administrator Account

The details for configuring the authentication for the service are explained in the next chapter. For now, it is enough to know that an administrator account is created the first time the service starts. The username is “admin”; the randomly-generated password is available in the service log `/var/log/stratuslab/cimi/cimi.log`.

## Testing the CIMI Service

The “CloudEntryPoint” resource as well as a few others are visible to anyone, even those without an account on the cloud. We can verify that the service is working correctly by retrieving the CloudEntryPoint.

To do this via the command line, just use `curl` on the base URL of the service:

```
$ curl -s --insecure https://cimi.example.org/ \
python -mjson.tool

{
  ...

  "baseURI": "https://onevm-142.lal.in2p3.fr:443/",
  "created": "2013-11-12T16:10:47.990Z",
  "id": "CloudEntryPoint",
  "jobs": {
    "href": "Job"
  },
  "machineConfigs": {
    "href": "MachineConfiguration"
  },
  "resourceURI": "http://schemas.dmtf.org/cimi/1/CloudEntryPoint",
  ...
}
```

This resource (in JSON format) contains the list of all of the cloud resource collections supported by this cloud infrastructure, along with relative URLs (in the “href” field) for those resource collections. It also contains metadata concerning the cloud infrastructure itself.

---

**Note:** The first access to the server takes some time to respond because the server is dynamically compiling the source clojure files and initializing the database. Subsequent accesses to the service should be much faster.

---

There is also a rudimentary web browser interface provided by the service. Point a browser at the URL <http://cloud.example.org/cimi/webui>, replacing the hostname with your own. You should see an HTML representation of the CloudEntryPoint as in the following screenshot.

## Verify Administrator Account

You will be using the administrator account to update the service configuration. To verify that it works, first recover the administrator’s account password from the service log. You should find a message in the log like the following:

```
... User/admin entry created; initial password is 6GfRtIeWVyGK
```

The username of this initial account is always “admin”; the “6G...” value is the generated password. Use the value from your log file.

To login as the administrator from the web interface, click on the “login” link in the upper right corner, fill in the username and password on the form, and then click the “login” button. If the login was successful, then you should be redirected back to the CloudEntryPoint, but you will see your login information on the right side of the header.

---

**Note:** You can always see your full authentication information by visiting the URL <https://cloud.example.org/cimi/authn>. The most important fields are the “identity” field (giving your username) and the “roles” field (giving your authorizations).

---

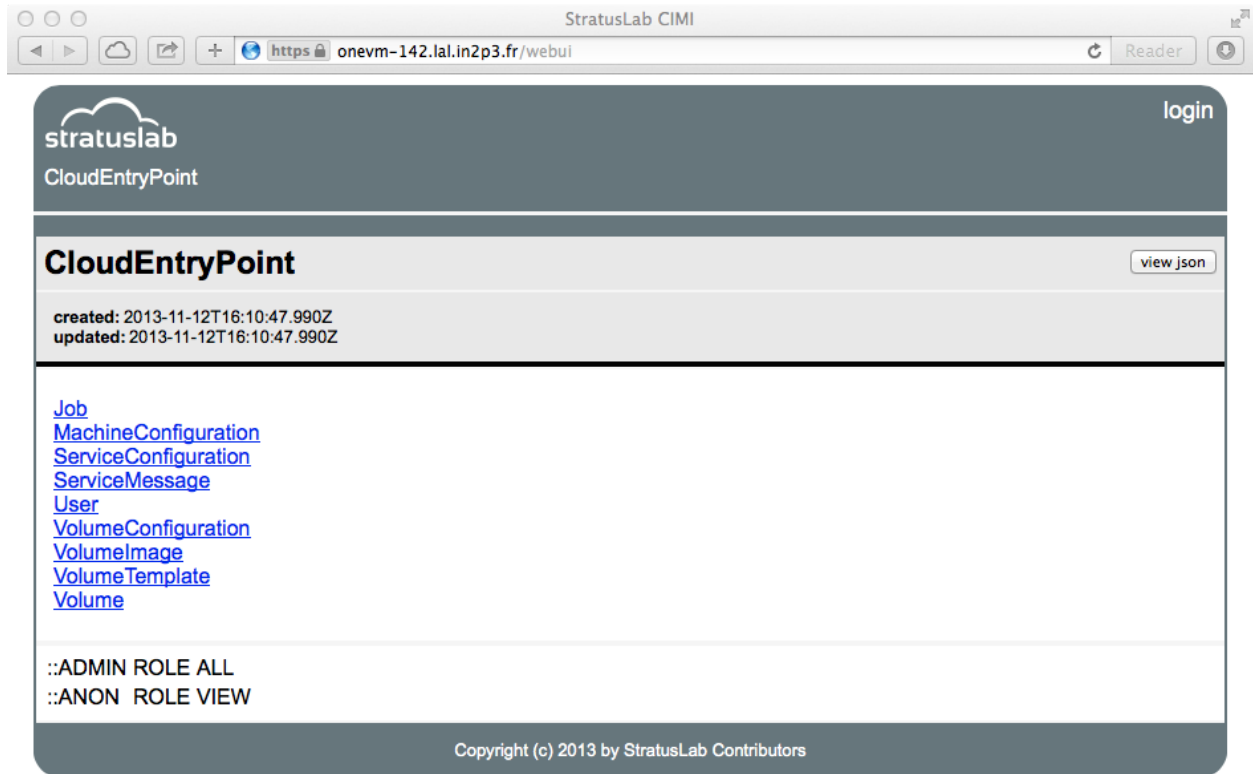


Fig. 2.9: CloudEntryPoint Viewed in CIMI Web Browser Interface



Fig. 2.10: Logged in User Information



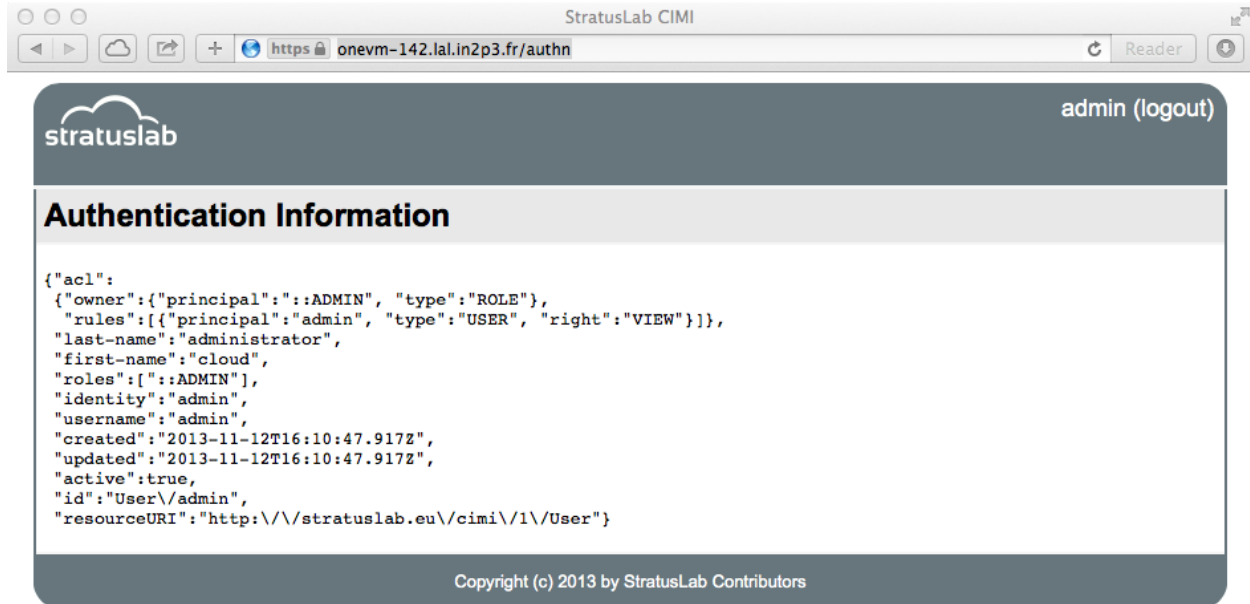


Fig. 2.11: Full Authentication Information

If you can see pages similar to the screenshots, the administrator of the CIMI is correctly configured. However, you will likely want to **change the password of the administrator account**. Now that you are logged into the server, you can do this.

Return to the CloudEntryPoint using the web browser interface (i.e. the URL ending with “webui”). From there, click on “User”. This brings up the list of user records; only the “admin” account should be listed. Then click on “admin” to view the user record. You should see a page listing characteristics of the “admin” account, notably there will be a field “password” containing the bcrypt hash of the current administrator password.

You should see three buttons on the right of the page: “view json”, “edit”, and “delete”. You will want to click on the “edit” button which will bring up a JSON editor with the current contents of the “admin” user.

However, before doing this, you want to generate the bcrypt hash for a new (memorable) password. This can be done with python using the following command:

```
$ python -c "
> import bcrypt
> h=bcrypt.hashpw('hello', bcrypt.gensalt())
> print h
> "
$2a$12$zvS7axGrws6/YH2AuIyXpufc174KV5bjBTp.vo400sGZsehP7CpFS
```

You may have to install the package “py-bcrypt” on CentOS for this to work. It returns the hash of your password. Change the ‘hello’ in the example to the password you want to use.

Now that you have a new password, click on the “edit” button, change the value of the password field to the hash value you’ve generated and click on the “save” button. You should be redirected back to the same page, but the password field will have been updated.

You can now logout via the “logout” link and log back into the service (with your new password!) using the same procedure as before.

## Service Messages

As a further example of how to use the web interface (which you will use to handle service configuration), you can create “ServiceMessage” resources on the server.

The ServiceMessage resources are visible to anyone but can only be created by the administrator. These messages are intended to provide general service information to users, like the MOTD (message of the day) text on many operating systems.

From the CloudEntryPoint, click on the “ServiceMessage” link. This should bring up an empty list of ServiceMessage resources. Click on the “add” button, which will bring up the same JSON editor you saw previously.

Add something like the following to the editor panel:

```
{
  "name": "StratusLab is Alive!",
  "description": "Deploying StratusLab clouds is fun."
}
```

and then click on the “save” button. You should then see a summary panel of the message along with metadata that was added to the entry. You can view JSON for the entry with the “view json” button, update it with the “edit” button, or delete it with the “delete” button.

If you go back to the ServiceMessageCollection, you will see the entry in the list.

For ServiceMessage resources the “name” field is treated like a title and the “description” gives the full message.

### 2.2.13 Authentication

As mentioned in the previous chapter, the CIMI server also handles all of the user authentication for the cloud. The server supports the use of internal or external databases of users.

The internal database, with user records stored in Couchbase, is convenient because all information is contained within the cloud infrastructure and user records can be managed in the same way as other configuration files.

Using an external database, such as LDAP, allows the user information to be exported to other system or allows better integration of the cloud with other services in the data center. In the case of VOMS proxies, this also delegates some authority to third parties to manage the users associated with a particular Virtual Organization.

## Overview

There are two aspects to the authentication configuration for a StratusLab cloud: defining the methods to be used and managing the users.

To define the authentication mechanisms for the cloud infrastructure, you must create the document “ServiceConfiguration/authn” in the Couchbase database. You can use the same web interface shown earlier to browse to the “ServiceConfiguration” collection and then to add the new document.

The document must follow the defined schema:

```
{
  "service": "authn",
  "localdb":
    {
      "password-enabled": true,
      "cert-enabled": false
    },
  "ldap":
    {
```

```

    "password-enabled": false,
    "cert-enabled": false,
    ...
  },
  "voms":
  {
    "enabled": false
  }
}

```

where the “service” and “localdb” fields are required (and the value for the “service” field must be “authn!”). The fields “ldap” and “voms” are optional. The details of the “ldap” map are given below.

If the configuration document does not exist, then the default is to only use the local user database with password credentials. It is recommended to always keep this method active along with at least one administrative account defined in the database.

**NOTE:** Enabling or disabling authentication methods requires that the CIMI service be restarted.

Adding or removing users from the system, requires changes to the local or external user databases. The details on this are given in the following sections.

**NOTE:** Adding, removing, or modifying users does **not** require restarting the CIMI service. These changes will be taken into account immediately for all future authentication actions.

You can also define “roles” for the users to either indicate group membership or rights to perform particular actions. There are three special roles “:ADMIN”, “:USER”, and “:ANON”. The first confers administrative rights to the user; users with this role will have complete control over the cloud configuration. The “:USER” role is given to anyone who has been authenticated. The “:ANON” role is used for users that have not been authenticated.

## Couchbase User Entries

You can manage users directly in the Couchbase database via the CIMI interface. There is a “User” collection and you can add, modify, and remove entries from the system with the same web browser interface you’ve used for the other resources.

The schema for user records is:

```

{
  "last-name": "required",
  "first-name": "required",
  "username": "required, must be unique",
  "password": "optional, value is bcrypt hash",
  "enabled": true,
  "roles": [ "list", "of", "roles" ],
  "altnames": {
    "x500dn": "DN of user in RFC2253 format"
  }
}

```

If the “enabled” field is not supplied, the default value is “false”. The “password” is used only for password authentication; similarly, the “x500dn” field is only used for certificate authentication.

The schema is a “loose” one, meaning that other fields may be added if you want to track additional information. One example would be an “email” field.

## Using Passwords

To use password authentication, the “localdb/password-enabled” flag must be true in the “ServiceConfiguration/authn” document. The user record for the given username must exist, must be enabled, and must have a password set.

The value of the password field is the bcrypt hash of the clear text password. See the preceeding text for generating the bcrypt hash.

## Using Certificates

To use certificate authentication, the “localdb/cert-enabled” flag must be true. The field “x500dn” must exist and have the RFC2253-formatted DN of the user’s certificate. The user can use the raw certificate or a proxy generated from that certificate. The user will be identified with the value of the “username” field in the user record.

For certificate authentication to work, the SSL configuration for the Jetty server must be done. By default, the configuration for trusting the EGI Certificate Authorities is done by the `stratus-install` command. If you want to trust different Certificate Authorities, you must adjust the SSL configuration. See the previous chapter for what needs to be done.

## LDAP User Database

To authenticate users against an LDAP database, you must have deployed and populated an LDAP server. The authentication configuration for using LDAP is quite flexible, so nearly any standard layout for the information should work.

The “ldap” field in the “ServiceConfiguration/authn” document must follow the schema:

```
"ldap": {
  "password-enabled": true,
  "cert-enabled": false,

  "connection": {
    "host": {
      "address": "localhost",
      "port": 389
    },
    "ssl?": false
  },

  "user-object-class": "inetOrgPerson",
  "user-base-dn": "ou=users,o=cloud"
  "user-id-attr": "uid",

  "role-object-class": "groupOfUniqueNames",
  "role-base-dn": "ou=groups,o=cloud",
  "role-member-attr": "uniqueMember",
  "role-name-attr": "cn",

  "skip-bind?": false,
}
```

All of the fields are required. The “connection” values provide the location of the server. The “user-” fields indicate what objects in the LDAP database are user records and the “role-” fields indicate the groups that are mapped to roles.

## Using Passwords

The “skip-bind?” field indicates whether to try to bind to the database with the given user password to authenticate the user. Normally, for password authentication this should be false and the LDAP server should be setup to allow only the user to view her user record.

## Using Certificates

This is not supported in the current version, but is planned for a future version.

## VOMS Proxy Authentication

VOMS proxies are a mechanism by which users can delegate some rights to a third party such as a service. These proxies also convey certain rights associated with a user who belongs to a Virtual Organization. The Virtual Organization itself manages its membership, defines the rights, and allocates those rights to members.

By default, the SSL configuration of the server will be setup to allow validation of VOMS proxies according to the policies of the European Grid Infrastructure. You must use this configuration (or a similar one if you are an expert) to use VOMS proxy authentication.

To enable this, set the “voms/enabled” flag to true in the authentication configuration. You must then add a pseudo-user entry in the local database for each Virtual Organization you want to support.

An example pseudo-user entry for the “vo.lal.in2p3.fr” Virtual Organization is:

```
{
  "last-name": "Virtual Organization",
  "first-name": "LAL",
  "username": "vo:vo.lal.in2p3.fr",
  "enabled": true
}
```

It is important that the username contain the “vo:” prefix to the name of the Virtual Organization that is being authorized. The entry should *not* have a password associated with it.

The “username” associated with users identified via VOMS proxies is the DN of their certificate. The FQANs (fully-qualified attribute names) of the VOMS certificate are mapped to roles in the StratusLab authentication.

## Future Authentication Methods

The authentication framework used by StratusLab is extensible allowing different mechanisms to be incorporated fairly easily. Candidates for additional mechanisms to support in the future include OAUTH2 and Shibboleth. Feedback on the desire for these (or other) mechanisms is welcome.

## 2.2.14 Using Ceph

Ceph is a set of storage technologies that allow object, block, and file storage that is reliable and scalable while maintaining high-performance.

The block storage component of Ceph can be used as a storage backend to the persistent disk service.

## Requirements

- A working Ceph cluster
- A Ceph pool for pdisk
- A manager identity for this pool

To create the pool: `ceph osd pool create ${poolname} 128 128`

To create the manager identity for the pool:

```
ceph auth get-or-create client.${identity} \
  mon 'allow r' osd \
  'allow class-read object_prefix rbd_children, allow rwx pool=${poolname}'
```

## Modifications

Some modifications are needed for the persistent disk server and the VM hosts.

- Linux kernel 3.x
- Ceph package
- Ceph configuration in `/etc/ceph/ceph.conf`
- Manager keyring in `/etc/ceph/ceph.client.${identity}.keyring`
- Parameters in `pdisk-backend.cfg` and `pdisk-host.conf`

For testing, you can use a package provided by [ELrepo repository](#) which provides a 3.x kernel for CentOS v6.4.

Currently, using a proxy server is not very useful for this backend because the persistent disk server needs a 3.x kernel and the Ceph package to attach pdisks to itself when importing a remote appliance (cf. `DiskUtils.copyUrlToVolume`).

There is not automatic configuration via the StratusLab installer yet. This is being worked on.

The `oneadmin` user must also be able to execute Ceph commands as root. Update the `/etc/sudoers` file appropriately:

```
# /etc/sudoers
oneadmin ALL= NOPASSWD: /usr/bin/rbd
```

## 2.2.15 Frequently Asked Questions

This section collects a number of frequently asked questions concerning the administration of a StratusLab cloud infrastructure. If there are other questions that you would like to see treated here, please contact StratusLab support.

### How to restart VMs after a power outage?

After the power outage the nodes hosting the virtual machines must be restarted. If this has not happened automatically, ensure that all of the nodes have rebooted.

Connect to each of the nodes to ensure that all of the iSCSI mounts are present.

```
# iscsiadm -m session
```

If the links are not correctly mounted (bad links will blink in the list), then the easiest way to recover this is simply to reboot the machine again.

Once you have verified all of the iSCSI links on all of the nodes, then you must log into the frontend machine as *oneadmin*. As this user, you can then force OpenNebula to restart all of the machines in the *unknown* state.

The commands for this (assuming that you’re logged into the frontend as *root*), are:

```
# su - oneadmin
$ for vm in `onevm list all |grep unk | awk '{print $1}'`; \
do; \
    onevm restart $vm; \
done
```

After this, all of the VMs should return to the *running* state and again be accessible from the network. Any machines that don’t restart are probably lost and will have to be killed and a new instance created.

## How to support advanced CPU flags in virtual machines?

By default, libvirt and KVM provide a generic CPU processor to virtual machines. For computing intensive applications, access to more advanced CPU capabilities (CPU “flags”) can make an enormous difference in the efficiency of a calculation. Consequently, the generic CPU profile provided by libvirt and KVM is inadequate for these applications.

Most modern processors have the advanced CPU capabilities that numerical applications need. The only problem is how to get libvirt and KVM to expose these capabilities to the hosted virtual machines?

Unfortunately, libvirt does not provide a mechanism for passing arbitrary options to the underlying hypervisor. However, the call to the hypervisor can be wrapped to provide the additional options.

Create a wrapper script for the call to *qemu* that will add the *-cpu* option:

```
#!/bin/bash
/usr/libexec/qemu-kvm -cpu host $@
```

This additional option will allow the host CPU specification to be passed directly to virtual machines, allowing all of the advanced capabilities of the host processor to be used.

**Warning:** The wrapper script must be installed on all of the nodes of the StratusLab cloud infrastructure. This script is named */var/lib/one/bin/qemu-start.sh* below; change this value as necessary.

Once the wrapper script is installed, then the OpenNebula configuration must be changed to force this script to be used to start virtual machines. Modify the configuration files */etc/one/vmm\_exec/vmm\_exec\_kvm\*.conf* setting the value of the *EMULATOR* parameters:

```
EMULATOR = /var/lib/one/bin/qemu-start.sh
```

Change the value to match the wrapper script that you have created. Once this is done, restart OpenNebula. Execute:

```
$ service oned restart
```

on the frontend node of your StratusLab cloud infrastructure.

Once this is done, any new virtual machines will use the wrapper script and will have the full CPU capabilities of the host processor. You can verify the CPU capabilities from the virtual machine with the command:

```
$ cat /proc/cpuinfo
```

The relevant capabilities are in the “flags” part of the response.

## How to fix the keyboard mapping for virt-manager?

Occasionally when debugging problems with virtual machines it is useful to use *virt-manager* via X11 to bring up the machine console. The default configuration unfortunately makes the keyboard mapping completely useless. To fix this, you must use the *-k* option when starting virtual machines with QEMU.

To do this, follow the same procedure as for exposing the CPU flags above and add the *-k en-us* option to the qemu command line. This will cause the keyboard mapping to always be a US English. You may use other language values, although the US English is the safest default.

## 2.3 Contributor Guide

This guide provides information for people who contribute to the development and documentation of the StratusLab cloud distribution. It describes how to become a member of the StratusLab development community and provides information about languages and coding standards.

### 2.3.1 Preface

#### Target Audience

This guide provides information for people who contribute to the development and documentation of the StratusLab cloud distribution. It describes how to become a member of the StratusLab development community and provides guidelines on contributions.

Those wanting to use a StratusLab cloud infrastructure should consult the StratusLab User's Guide.

System administrators wanting to install a StratusLab cloud on their own resources should start with the StratusLab Administrator's Guide.

#### Typographic Conventions

This guide uses several typographic conventions to improve the readability.

links	<a href="#">some link</a>
filenames	<code>\$HOME/.stratuslab/stratuslab-user.cfg</code>
commands	<code>stratus-run-instance</code>
options	<code>--version</code>

Table: 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.

### 2.3.2 StratusLab Collaboration

The StratusLab Collaboration aims to provide a complete, open-source cloud distribution that allows resource centers to deploy their own public or private “Infrastructure as a Service” (IaaS) cloud infrastructure.



The collaboration's motto is "Darn Simple Cloud". We aim to provide a cloud solution that is both simple to install and simple to use. Where possible, we build on existing software and technologies to minimize and to simplify our own code base.

At the same time, we strongly believe in providing high quality, well tested code; we use agile software development processes and tools to achieve this.

## History

StratusLab started as an informal, academic collaboration between several partners involved in the EGEE series of grid projects in 2008. The initial aim of this collaboration was to investigate if the new cloud technologies (primarily Amazon Web Services) could be used as a platform for running grid services. The collaboration concluded that it was indeed possible and further, was convinced that cloud technologies would be an important platform for scientific computing.

The collaboration subsequently grew into a more formal collaboration that proposed a European-level project to build an open-source cloud distribution suitable for running grid services. This proposal was accepted and StratusLab, as a project co-funded by the European Commission, ran from June 2010 to May 2012. During this period, the foundations of the current StratusLab cloud distribution were built.

The StratusLab collaboration continues to maintain and to develop further this cloud distribution as an open collaboration of institutes and individuals. The four principal institutes currently participating are CNRS (France), SixSq (Switzerland), GRNET (Greece), and Trinity College Dublin (Ireland).

## Joining the Collaboration

The collaboration welcomes new collaborators, both institutions and individuals. All it takes is a willingness to participate constructively in the collaboration and to provide contributions consistent with the licenses used by the collaboration. Contact us at [contact@stratuslab.eu](mailto:contact@stratuslab.eu) to become a member.

## Communication

The collaboration coordinates its activities primarily through a common mailing list: [stratuslab-devel@googlegroups.com](mailto:stratuslab-devel@googlegroups.com). Developers are expected to solicit feedback and to announce significant changes on that list. There is also a weekly teleconference on Thursdays at 11:30 (Paris) to allow more interactive discussion of progress and problems.

### 2.3.3 Code Management

As a software development collaboration, the management of the common code base is of primary concern. This chapter describes the best practices for the collaboration.

## Licenses

Apache 2 is the primary code license of the collaboration. Although, exceptions can be made for specific cases (e.g. plugins to libraries under a different license), all core software must be under the Apache 2 license to ensure that it can be legally integrated into a common distribution and to ensure the widest possible reuse of the software.

High-level documentation (e.g. User's Guide) is released under the Creative Commons Attribution license to ensure its widest possible distribution. Documentation integrated with the code (e.g. README) are released under the same license as the code itself.

All non-trivial source code files, must have a comment block at the beginning of the file declaring the license for the code. This block must also contain the copyright statement for the file. The copyright is held by the original author (or author's institute) and must also appear in this initial comment block.

### Preferred Languages

To maximize the number of people who can contribute to the maintenance and development of the StratusLab software, the collaboration attempts to minimize the number of development languages used within the project.

On the server side, the collaboration prefers the use of Clojure (on the JVM) and Java. Historically, all of the services developed by the project have been written in Java; more recently this has been shifting towards Clojure.

On the client side, Python is the preferred language. The StratusLab client is nearly entirely written in Python and many of the utility scripts (e.g. binding to Libvirt) are migrating from shell to Python.

Exceptions are made for specific cases. For example, server startup scripts are written in bourne shell. If in doubt, ask for feedback via the developers' mailing list.

### Repositories

All of the code for the collaboration resides in [GitHub](#) under the "StratusLab" organization. Generally, each major service resides in its own repository.

*All developers have write access to all repositories.* This allows everyone to contribute to any of the services and to correct bugs as they are found. Nonetheless, major changes in a repository should be discussed on the developers' mailing list and with the primary developer of the service.

### Build System

The collaboration has chosen to use Maven as its build system. Consequently, all of the code is built using the standard Maven workflows. The collaboration maintains a common top-level `pom.xml` file to manage plug-ins and dependencies. New dependencies should be discussed on the developers' mailing list and then added to the top-level `pom`.

### Code Quality

#### Issues

Significant changes to the code and specific bug fixes should be tracked via GitHub issues. These issues should also be tied to milestones so that release announcements can be prepared. Reference the Issue ID number in the commit message, so that GitHub will automatically cross-reference the commit to the issue.

### Code Formatting

Developers should generally follow the well-known style guides for their code. Specifically,

- Clojure: [Clojure Style Guide](#)
- Java: [Oracle style guide](#)
- Python: [PEP8 style guide](#)

Spaces are preferred over tabs with indentation depending on the language (2 for Clojure; 4 for others).

IDEs such as Eclipse, IntelliJ, and PyCharm make formatting the code to these specifications easier. Their use is recommended.

### **Static Analysis**

Where possible, static analysis of the code should be integrated with the build process.

For Java, FindBugs is an excellent tool for finding common errors through the static analysis of the generated bytecode. All Java-based project, must run FindBugs as part of the build process. Any “critical” problems identified by FindBugs must be corrected before committing the code.

For other languages, you may wish to integrate similar tools in the build process. For these other languages, we do not have specific recommendations.

### **Unit Testing**

All of the StratusLab code should use unit testing to ensure that the code behaves as expected. These unit tests are especially critical when refactoring code (something that happens fairly frequently!).

The code should use the appropriate xUnit framework for the programming language being used. Ideally, the unit test reports should be formatted in the JUnit XML format to allow better reporting from the continuous integration tools.

## **2.3.4 Documentation**

### **Repository Documentation**

Each code repository must have a README (in markdown format). It must contain the following:

- A description describing the purpose of the code
- A statement about the license
- Any acknowledgements (e.g. for the European Commission)

It can also describe:

- Manual installation instructions for the service
- Firewall and port requirements
- Dependencies

or any other relevant information.

### **Guides**

The collaboration maintains three guides: one for users, one for administrators, and one for contributors. These should be reviewed and updated when significant changes are made to the code. These *must* be reviewed before each release.

### 2.3.5 Continuous Integration

The collaboration uses [Jenkins](#) for continuous integration. Each time a change is committed to a StratusLab repository, the Hudson server launches a job to rebuild the software, executing the defined unit tests and repackaging the software.

Assuming that the unit tests pass, jobs to reinstall a test cloud and to verify the high-level functionality are also run.

The [continuous integration server](#) provides a list of all of the defined jobs and a dashboard of the latest status.

### Supported Platforms

The project currently supports CentOS. All of the builds run on this platform along with the cloud installation and functionality tests.

All of the code is highly portable and should run on any Linux platform. Additional operating systems can be supported if there is enough effort to support the packaging and testing of the code on the operating system.

### Package Repository

The artifacts (packages) from all of the builds are uploaded to a Maven repository. The project runs a [Nexus server](#) for this.

### 2.3.6 Releases

#### Frequency

The project follows a timed-release policy where a new release is prepared every quarter. The release numbering follows the Ubuntu-style format (e.g. 13.05.0 for the May 2013 release). Patch releases may also be prepared for individual components between the quarterly releases as necessary.

#### Milestones

GitHub issues should be tied to a milestone. Where the milestones are named after the releases (e.g. 13.05 for the May 2013 release). This allows the changelog for the release to be prepared easily.

#### Tagging and Packaging

All of the tagging and packaging of the releases are handled through manually-triggered jobs in Jenkins.

Note that these must be done in the correct order to ensure that the proper dependencies are picked up. The proper dependencies will need to be updated by hand in the pom files.

#### Publishing

The releases are made available through yum repositories. Again, Jenkins is used to create the content of these repositories. They must be copied by hand to the StratusLab yum repository machine.

There are also a couple of python modules (stratuslab-client and stratuslab-libcloud-drivers) that must be pushed into PyPi. Once these are tagged, they should be built by hand and then pushed into PyPi.

## Announcements

A release announcement and changelog must be prepared for each release. It should be published on the website and then tweeted.

### 2.3.7 Service Configuration

All StratusLab services must follow a common configuration scheme where the core of the configuration is stored in the Couchbase database. Only the Couchbase configuration parameters are read from a file on the local filesystem.

This scheme allows the overall system to scale with multiple service instances while maintaining consistent configurations between them.

#### Couchbase Connection Parameters

The Couchbase connection parameters must reside in a file on the physical machine to bootstrap the configuration mechanism. This file `/etc/stratuslab/couchbase.cfg` is in the standard “ini” format:

```
[DEFAULT]
hosts=host1,host2,host3
bucket=bucket1
password=bucket1_password

[service]
hosts=host4
bucket=other_bucket
password=other_password
cfg_docid=docid
```

Each section corresponds to the parameters for a particular service. The name of the section is defined by the service. If parameters are not defined in a specific section, then the ones in the `DEFAULT` section will be used. Services may have a default value for the `cfg_docid` value.

#### Service Instance Configuration

After reading the Couchbase connection parameters, the service should read its configuration from a document in Couchbase. The document identifier containing the configuration is defined by the service, but may be overridden by the `cfg_docid` parameter in the local configuration.

To use Couchbase efficiently, these configuration files should be in JSON format. This allows them to be pulled into native data structures easily for our preferred implementation languages.



---

## Search

---

- search