StratusLab Contributor's Guide

StratusLab Collaboration

Version 13.12.0

Contents

Chapter 1

Preface

1.1 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.

1.2 Typographic Conventions

This guide uses several typographic conventions to improve the readability.

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

Table 1.1: Typographic Conventions

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

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

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

Chapter 2

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.

2.1 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 is 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).

2.2 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 to become a member.

2.3 Communication

The collaboration coordinates its activities primarily through a common mailing list: 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.

Chapter 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.

3.1 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.

3.2 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.

3.3 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.

3.4 Build System

The collaboration has chosen to use Maven as its build system. Consequently, all of the code is build 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.

3.5 Code Quality

3.5.1 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.

3.5.2 Code Formatting

Developers should generally follow the well-known style guides for their code. Specifically, follow the Oracle style guide for java and the PEP8 style guide for Python.

IDEs such as Eclipse and PyCharm make formatting the code to these specifications easier. StratusLab has a PyCharm licence for its developers; ask for the license key.

3.5.3 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.

3.5.4 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.

Chapter 4

Documentation

4.1 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.

4.2 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.

Chapter 5

Continuous Integration

The collaboration uses Hudson (probably Jenkins in the future) 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 Hudson server provides a list of all of the defined jobs and a dashboard of the latest status.

5.1 Supported Platforms

The project currently supports CentOS and OpenSuSE. All of the builds are automatically run on both platforms when changes are committed.

The cloud installation and functionality tests are currently only run on CentOS.

5.2 Package Repository

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

Chapter 6

Releases

6.1 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.

6.2 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.

6.3 Tagging and Packaging

All of the tagging and packaging of the releases are handled through manually-triggered jobs in Hudson. The “tagging” jobs should be executed first, with the “release” jobs executed afterwards for both CentOS and OpenSuSE.

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.

6.4 Publishing

The releases are made available through yum repositories. Again, Hudson 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.

6.5 Announcements

A release announcement and changelog must be prepared for each release. It should be published on the website and then tweeted.

Chapter 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.

Chapter 8

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.

Chapter 9

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.