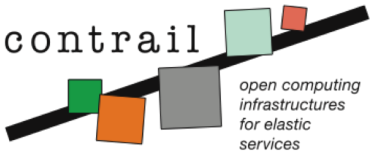


Project no. 257438

CONTRAIL

Integrated Project
OPEN COMPUTING INFRASTRUCTURES FOR ELASTIC SERVICES

First release of CONTRAIL platform
D11.3

Due date of deliverable: March 30^{th}, 2012
Actual submission date: May 10^{th}, 2012

Start date of project: October 1^{st} 2010

Type: Deliverable
WP number: WP11
Task number: T11.6

Responsible institution: EBM
Editor & and editor's address: Sébastien André
4 rue Amélie - 31000 Toulouse, France

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	12/02/03	Matej Artač	XLAB	Outline
0.2	12/02/24	Sébastien André	EBM	First draft
0.3	12/03/16	Sébastien André	EBM	Implemented reviewers comments
0.4	12/03/28	Sébastien André	EBM	Added appendix and details about the release
0.8	12/03/30	Sébastien André	EBM	Spell check for submission
0.9	12/05/03	Uroš Jovanovič	XLAB	Adding the results chapters
1.0	12/05/04	Uroš Jovanovič	XLAB	Last fixes

Reviewers:
Jan Stender (ZIB) and Philip Kershaw (STFC)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T11.6	Build CONTRAIL software releases integrated in a Linux distribution.	EBM*

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive Summary

The goal of the Contrail Project is to design, implement, evaluate and promote an open source system for Cloud Federations. In this document we describe the work that has been done to bring out the first release of the Contrail software integrated in a Linux distribution. More precisely, our main objectives were:

- to provide a coherent development environment for the developers along with best practice coding rules for high quality codes.
- to package all contributions of software in a coherent, manageable and releasable set.
- to validate the software packages and ensure they meet minimal quality criteria.

We believe that small pieces of effort, applied frequently, are more likely to improve the quality of software and to reduce the time taken to deliver it than applying quality control after completing all development as in the traditional practice. That's why our efforts were directed towards promoting agile development methods and setting up an efficient continuous integration process.

In this document we explain how we configured the services provided by the development platform to users and developers, and how we automated the validation of the software packages. We also provide a summary of how to get started with a typical deployment of Contrail, where to find documentation and how to get support. The detailed release procedure and shell scripts are included in the appendix.

Contents

List of Figures

Introduction

The goal of the Contrail Project is to design, implement, evaluate and promote an open source system for Cloud Federations. The Contrail software aims at providing a complete Cloud stack which integrates a full Infrastructure-as-a-Service and Platform-as-a-Service facilities. It will enable Cloud providers to seamlessly integrate resources from other Clouds with their own infrastructure, and break the current customer lock-in situation by allowing live application migration from one cloud to another.

We believe that small pieces of effort, applied frequently, are more likely to improve the quality of software and to reduce the time taken to deliver it than applying quality control after completing all development as in the traditional practice. That's why our efforts were directed towards promoting agile development methods and setting up an efficient continuous integration process.

This document describes the work that has been done in order to release the Contrail software integrated in a standard Linux distribution. In the first section, we describe the services provided by the development platform to users and developers. In the second section, we explain how the continuous integration brings quality to the software we produce. The third section documents the release process and how we automated the validation of software packages. In the fourth and last section, we show the typical steps for setting up a working Contrail environment and how users can get support. All shell scripts used to release Contrail as well as the detailed release and install procedures are listed in the appendix.

1 Development platform

In this section, we describe the services provided by the development platform, as well as the typical activities of users and developers. Figure 1 shows an overview of the relationships between services providers and actors of the development process.

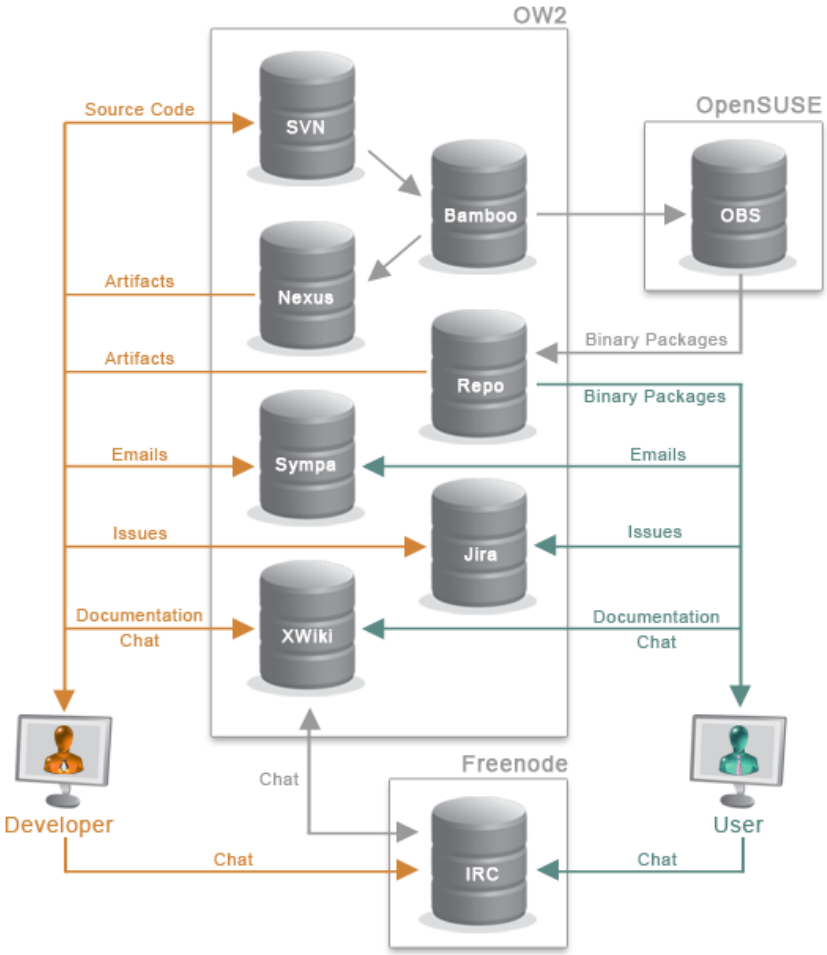


Figure 1: Overview of the Contrail development platform

1.1 Overview

1.1.1 Actors

There are mainly two kinds of actors who make use of the Contrail development platform:

Developers provide the source code and use the platform to build binary packages of the software.

Users download binary packages, read documentation and get support.

1.1.2 Services providers

The collaborative development platform provides users and developers with the services they need. These services are hosted by different partners who are all involved in the Open Source community, they are represented by grey squares on figure 1.

The OW2 consortium OW2[11] is an independent open source community committed to making available to everyone the best and most reliable enterprise computing infrastructure software, including middleware, application platforms and cloud computing technologies. This organization has been chosen at the beginning of the project to be the main host of Contrail development platform services, it provides the Contrail community with sources, artifacts and files repositories, continuous build, bug management tool, wiki and mailing lists services.

Open Build Service The Open Build Service[28] is an open and complete distribution development platform designed to encourage developers to compile packages for multiple Linux distributions including openSUSE, Red Hat, Mandriva, Ubuntu, Fedora and Debian. We use it in Contrail because it really makes packaging easy while allowing a lot of potential target Linux distributions and there was not such facilities provided by OW2. Another reason for using it is that some components of the Contrail project[1] were already using it beforehand, and it was far more convenient to merge these packages repositories with Contrail's than duplicating the packaging work.

Freenode Freenode[21] is a widely used open source software-focused IRC network. The channel #contrail is used by Contrail developers and users to discuss.

Ohloh Ohloh[6] is a website which provides a web services suite and online community platform that aims to map the landscape of open source software development. In Contrail we use it to analyse our source code base and provide some metrics and statistics. It was not represented on figure 1 because it is not directly linked to the development process.

1.2 Services

1.2.1 SVN – source code management

We use Apache Subversion[1] to manage source code revisions. It saves each modification made by developers in distinct versions and allows to revert or compare

[1] xtreemfs and scalaris

files to its previous revision. Instructions to checkout source files are available online[26]. It is possible to browse the source code repository with WebSVN[24] and also to query commit diff logs[14].

1.2.2 Bamboo – continuous build

We use Atlassian Bamboo[2] to manage the continuous build of Contrail. It consists in rebuilding the entire software automatically after each modification of the source code commited in SVN. The status of latest builds is visible online[10], a screenshot is also in figure 2 of this document.

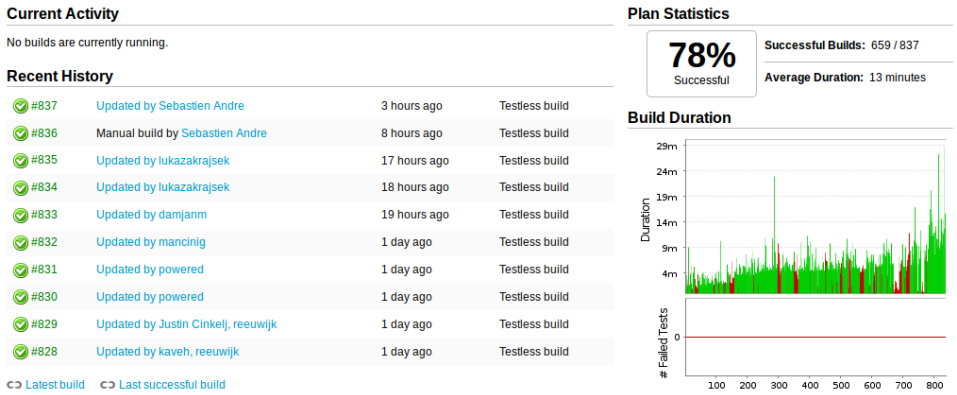


Figure 2: Screenshot of the Bamboo status page

1.2.3 Nexus – software artifacts management

We use Sonatype Nexus[29] to share software artifacts among developers. After each successful build in Bamboo, the produced artifacts are deployed in the Nexus repository and made available to developers. A search engine is provided online[12] for developers to query artifacts of the Contrail project and select the version of the dependency they want to use.

1.2.4 OBS – build binary packages

We use the Open Build Service[8] to build binary packages for every targeted Linux distribution[2] and each architecture[3]. It consists in fetching pre-build artifacts from Bamboo and applying some transformation scripts on them. The current status of packages built for testing the Contrail project is available online[28], a screenshot is also in figure 3 of this document.

[2] currently Debian and Ubuntu
[3] currently i686 and x86_64

	Debian_6.0		xUbuntu_11.04		xUbuntu_11.10	
	i586	x86_64	i586	x86_64	i586	x86_64
boost_ubuntu_1110	failed	failed	succeeded	building	succeeded	succeeded
conpaas-frontend	succeeded	succeeded	succeeded	building	succeeded	building
conpaas-mysql	succeeded	finished	succeeded	succeeded	succeeded	building
conpaas-taskfarm	succeeded	finished	succeeded	building	succeeded	building
conpaas-web	succeeded	finished	succeeded	scheduled	succeeded	scheduled
contrail-ca-server	succeeded	building	succeeded	scheduled	succeeded	scheduled
contrail-federation-api	succeeded	scheduled	succeeded	scheduled	succeeded	scheduled
contrail-federation-cli	succeeded	scheduled	succeeded	scheduled	succeeded	scheduled
contrail-federation-db	succeeded	scheduled	succeeded	scheduled	succeeded	scheduled
contrail-federation-id-prov	succeeded	scheduled	succeeded	scheduled	succeeded	scheduled
contrail-federation-web	failed	building	unresolvable	unresolvable	unresolvable	unresolvable
contrail-monitoring-hub	failed	scheduled	failed	scheduled	failed	scheduled
contrail-one-monitor	succeeded	scheduled	succeeded	scheduled	succeeded	scheduled
contrail-one-sensor	succeeded	scheduled	succeeded	scheduled	succeeded	scheduled
contrail-provider-accounting	succeeded	scheduled	succeeded	scheduled	succeeded	scheduled
contrail-provisioning-manager	failed	scheduled	failed	scheduled	failed	scheduled
contrail-rest-monitoring	succeeded	scheduled	succeeded	scheduled	succeeded	scheduled
contrail-vep-cli	succeeded	scheduled	succeeded	scheduled	succeeded	scheduled
contrail-vep-gui	succeeded	scheduled	succeeded	scheduled	succeeded	scheduled
contrail-vin	succeeded	scheduled	succeeded	scheduled	succeeded	scheduled
deps-scalarix-conpaas	succeeded	succeeded	succeeded	succeeded	succeeded	succeeded
deps-scalarix-conpaas-erl	succeeded	succeeded	succeeded	succeeded	succeeded	succeeded
deps-scalarix-conpaas-nopub	succeeded	succeeded	succeeded	succeeded	succeeded	succeeded
scalaris-svn	succeeded	succeeded	succeeded	succeeded	succeeded	succeeded
scalarix-svn-conpaas	succeeded	succeeded	succeeded	succeeded	succeeded	succeeded
xtreemfs-testing	succeeded	succeeded	succeeded	succeeded	finished	succeeded

Updated at: 2012-03-23 15:34:34+01:00

Figure 3: Screenshot of the OBS status summary page

1.2.5 Repo – publish files

Binary packages The binary packages of the Contrail software are available in an online files repository[13] hosted by OW2[23]. Once packages have been built by the Open Build Service[8] they are transfered to this repository[4].

Third party artifacts We also made our own third party repository[16] to provide developers with software dependencies that are not (yet) available in public repositories. It contributes to maintain a fast build environment by offering an alternative to uploading binary files in the source code repository, and thus reducing the average update and build durations.

[4]The download repositories feature of OBS could also have been used. However, we mirrored them instead because apt does not handle HTTP redirections (302) for mirrors[7]

1.2.6 Jira – manage tasks and issues

We use Atlassian Jira[3] to manage tasks and issues related to the development of the Contrail software. It enables users and developers to report bugs, ask questions or new features in a convenient way. A summary of these tasks and issues for the Contrail project is available online[9]. Figure 4 shows the increase of activity visible from a Jira dashboard few months before the first release.

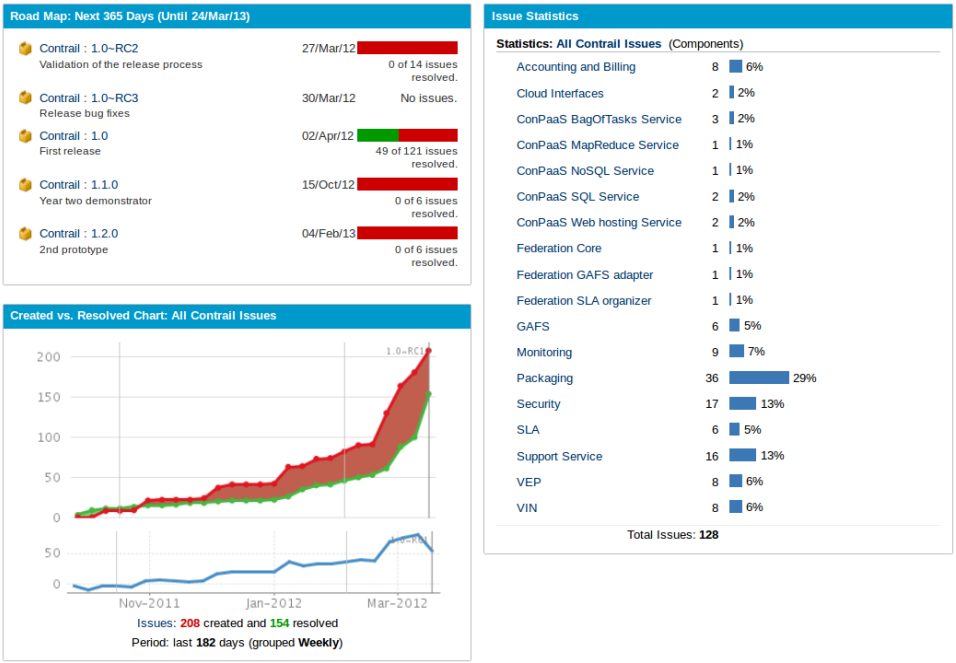


Figure 4: Screenshot of a Jira dashboard

1.2.7 XWiki – publish and share information

As shown on figure 1, nine services are involved in the development process of the Contrail software. One of them is the wiki, which aims at gathering and organizing any information about Contrail that may be useful to users or developers.

The Contrail project will end in October 2013 and in order to prepare for a smooth transition to a (potential) community managed project, we've setup a wiki for the *Contrail software* despite there was already one for the *Contrail project*. This wiki[17] is hosted by OW2, which guaranties that it will keep on running after the end of the sponsorship by the European commission. Another practical advantage is that it can be administrated by the same person who administers other development tools at OW2.

Figure 5 is a screenshot of the wiki's main page[17] where one can see it has been organized into four categories:

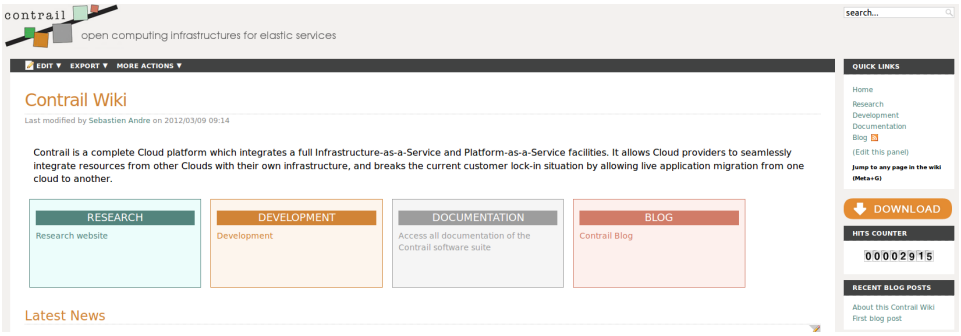


Figure 5: Screenshot of the main page in the Contrail software wiki

Research A link to the main page of the Contrail project website, which is actually about the research activity around Contrail.

Development A page which provides any information a developer may need[5] to contribute or to get help.

Documentation A page where *all* the user documentation should be found.

Blogs A place for the community to share its experience with the Contrail software.

1.2.8 IRC – discussion channel

When users or developers have a question or just want to discuss with each others, they can use our IRC channel `#contrail`. This service is provided by Freenode[5] and, as shown on figure 1, can be access by both a private client or the Contrail wiki[21]. Figure 6 illustrates the connection panel in the Contrail wiki.

Figure 6: Connexion panel in the wiki to the `#contrail` IRC channel

[5] see paragraph 1.3.2 about services provided to developers

1.2.9 Sympa – mailing lists

We use the Sympa[30] mailing lists server to broadcast notification or informative emails about the development of the Contrail software. So far, we created four mailing lists for the community:

contrail For questions about using Contrail, suggestions for improvements, ideas...

contrail-dev For discussions, questions or information about the development itself.

contrail-commits Notification of all SVN commits.

contrail-notifs Notification of Bamboo build failures.

contrail-admin For users to contact administrators and for discussions concerning the development infrastructure.

A detailed description of these mailing lists and how to subscribe/unsubscribe is available online[22].

1.3 Activities

Figure 7 shown a screenshot of the "developers space" in the Contrail wiki[18]. A lot of information about the project, such as how to get support, how to contribute, or what are the source code statistics can be accessed from this page.

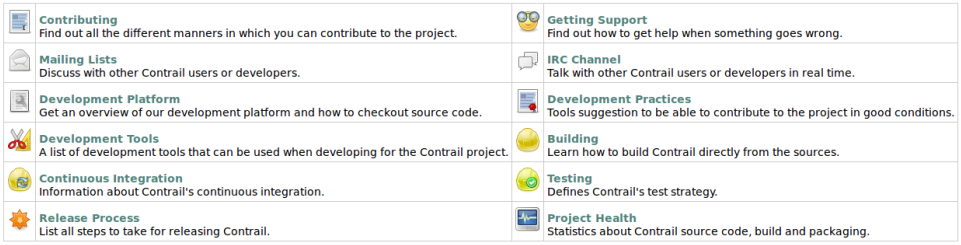


Figure 7: Screenshot of the developers space in the Contrail software wiki

1.3.1 Users

The Contrail users activity is represented by green arrows on the right side of figure 1. We've identified five kinds of activities:

- Read online documentation[15] on the wiki.
- Chat with other users or developers though the wiki IRC page.
- Post issues in the bug tracking system.

- Send emails to the community.
- Download binary packages from the files repository.

1.3.2 Developers

The Contrail developers activity is represented by orange arrows on the left side of figure 1. We've identified six kinds of activities:

- Write the documentation on the wiki.
- Chat with other users or developers though the wiki IRC page.
- Resolve issues in the bug tracking system.
- Send emails to the community.
- Download artifacts when building the software on his workstation.
- Uploading source code to save and share his work.

1.3.3 Automated

The activity on the development platform which is hidden to users and developers is represented by grey arrows on figure 1. They are of six kinds:

- Download source code from the Subversion repository to the Bamboo continous build system.
- Upload of software artifacts from Bamboo to the Nexus repository[6].
- Upload of bamboo pre-build artifacts to the Open Build Service[7].
- Download the binary packages from OBS to the OW2 files repository[8].
- Proxify the freenode "webchat" service, exposed by the wiki.

[6] see section 2.2 about dependencies management for more information.
[7] see section 2.5 about packaging for more information and the corresponding shell script in B.1.
[8] the corresponding shell script is included in B.2.

2 Continuous integration

This section describes how we configured the collaborative development tools to setup a continuous integration process. Our goal was to enable short development cycles and quick feedback to the developers in order to improve the quality of the software product.

2.1 Best coding practices

Best coding recommendations have already been documented in D11.1, a subset of the quality rules we promote is automatically enforced by our source code management system.

2.1.1 Subversion hooks

In Contrail, source code files are stored on a Subversion[1] repository, which is able to implement hooks. A hook is a program triggered by some repository event, such as the creation of a new revision or the modification of an unversioned property. Each hook is handed enough information to tell what that event is, what target(s) it's operating on, and the username of the person who triggered the event. Depending on the hook's output or return status, the hook program may continue the action, stop it, or suspend it in some way[4].

We use this mechanism to reject commits who breaks at least one of these rules:

- the log message that describes the commit contains at least one printable character
- the repository top-level structure (trunk, branches, tags) is not modified
- developer is not trying to add or update Eclipse configuration files
- developer is not trying to upload data or log files, except jar files

The shell script used to implement this hook is included in the appendix B.4.

2.1.2 Limits of rules enforcement

Storing jar libraries in the source code repository is not a good practice: it increases the time needed to checkout or update source files, slows down the build procedure which delays the feedback to the developer when a build breaks. In the long run, it also increases the maintenance cost because of the difficulty to upgrade versions of duplicated binary files.

However the rule of preventing jar files to be uploaded is not enabled in the SVN hook and as of March 2012, about 65 megabytes of binary files splitted in 200 jar files are stored in the source code repository. The main reasons are:

- we setup the thirdparty repository[9] when developers had already uploaded lot of jar files and had other priorities than rework their build procedure, it was too late.
- some developers don't want to change their work habits and prefer to use the source code repository as a simple storage.

In this case, we think that enforcing the rule of rejecting commits with jar files would be counterproductive because it would be a hindrance for developers and affect their productivity. Instead, we prefer to promote the use of the third party repository and keep an eye on the number of jar files because we expect to progressively reduce the number of these files: it's a tradeoff between quality and productivity.

2.2 Dependencies management

2.2.1 Disadvantages of embedded dependencies

When developing complex software, the source code repository must be structured with directories and the traditional way to resolve internal dependencies is to use relative paths in the directories tree. External dependencies are resolved by manually copying the artifact somewhere in the source repository and thus it becomes an internal dependency[10].

Managing this kind of project and different versions of *external dependencies stored locally* tends to be cumbersome when the size of the source code repository increases, especially when the developers team is distributed because coordination is more difficult. In a relatively complex project like Contrail, we estimate that in one year after the first release, it could lead to spend more than 50% of the maintenance effort in fixing dependencies and build issues[11]. In order to avoid this situation and keep a coherent build environment, we use a shared repository for software artifacts.

2.2.2 Advantages of using a shared repository for software artifacts

In the Contrail project, we decided to keep a rather simple structure in the source code repository and to manage dependencies externally with a shared repository. Figure 8 shows how dependencies are resolved:

1. When developer A commits the source code, the modification is stored in the Source Code Repository (SVN).
2. Few minutes later, this modification is detected by the Continuous Build process (Bamboo), which triggers a new build of the entire software and stores updated artifacts in the Software Artifacts Repository (Nexus).

[9] see "Repo" on figure 1

[10] it is exactly what developers who upload jar files in the source code repository do (see 2.1.2).

[11] based on the author's personnal experience, still looking for a better reference...

3. Developer B commits his own source code, which might depend of the source code produced by developer A, the modification is stored in the Source Code Repository (SVN).

4. Few minutes later a new build is triggered, and this time it will use the latest code provided by developer A which is now available in the Software Artifacts Repository.

s

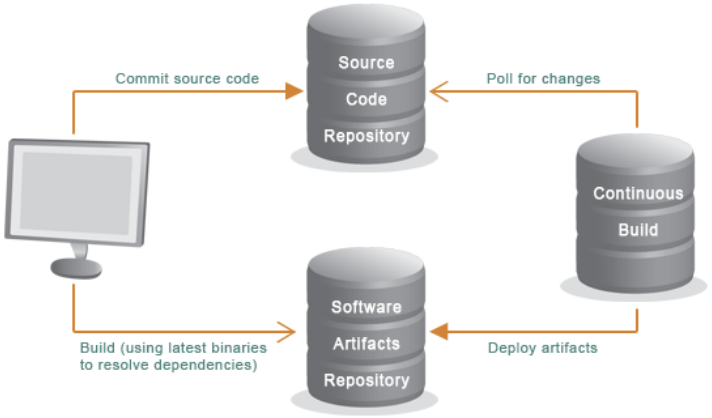


Figure 8: Management of build dependencies

Another advantage of using a shared repository for software artifacts instead of a relative link to another directory of the project is that if developer B doesn't want to use features recently added by developer A, he just has to specify the version he wants to use from the repository and doesn't need to ask developer A to differ his work.

2.3 Continuous build

2.3.1 Quick detection of build failure

What we call "continuous build" is fact a succession of builds that are triggered after each modification of the source code repository. If one of these modifications breaks the build, then it is detected only few minutes after the developers commit and will hopefully be rather simple to fix[12]. It is important to keep the build procedure fast and light in order to reduce the chances for a build to embed more than one modification, in which case debugging takes more time.

[12] it is a practice in some other projects to use SVN hooks to revert automatically every commit which would break the build so that it is impossible to have a broken build, however we do not intend to setup such practices in the Contrail project.

2.3.2 Providing quick feedback to developers

When a build fails in Bamboo, developers are instantly notified by the "contrail-notifs" mailing list and are expected to fix it as soon as possible. While the build is broken, Bamboo doesn't update the shared repository of artifacts, which might be a hindrance for some developers. Another consequence is that all subsequent commits might have introduced not yet detected compilation error that will have to be fixed later.

2.4 Integration tests

Integration tests are essential to bring quality in the produced software because unlike "validation tests" of "continuous build failures", they evaluate the functionalities implemented by developers, and expected by users. The tests strategy is documented in D11.2.

2.5 Packaging

The "packaging" of the Contrail software transforms the files stored in the source code repository into a set of standard binary packages that will be easy to install on Linux platforms.

2.5.1 Different stages of the transformation

Figure 1 shows that files from the source code repository (SVN) are not directly transformed into binary packages on the file repository (Repo), instead they go through different stages:

1. The first stage is represented by the link between SVN and Bamboo in figure 1, it corresponds to the transformation of source files into "pre-build" artifacts. We agreed with developers that every module to be packaged should produce an archive containing either its source files or needed artifacts during the build procedure, these files are stored and publicly accessible in Bamboo for each successful build. Figure 9 shows how to access to these artifacts from the Contrail Bamboo[10] website.

2. The second stage is represented by the link between Bamboo and OBS in figure 1, it corresponds to the transformation of pre-build artifacts into a standardized set of files which can be used by OBS to create binary packages. Paragraph 2.5.4 explains how to generate these files.

3. The third stage is represented by the link between OBS and Repo in figure 1, it corresponds to the upload of binary packages into the OW2 files repository. Once files have been uploaded, they are available for users to dowload.

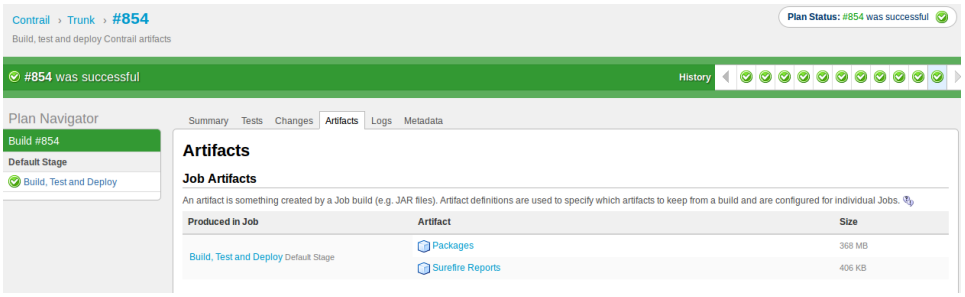


Figure 9: Artifacts produced by a build of Contrail by Bamboo

2.5.2 Reusing pre-build artifacts

In a traditional build system, we would have checked out the source code directly from the source repository instead of packaging pre-build artifacts stored in bamboo. There are two reasons for this choice:

Technical limitation of OBS It is not possible to run maven inside the OBS environment, probably because every package is created on a new empty virtual machine, and it would consume too much time and bandwidth to install all the needed dependencies to run Maven[13].

Speed of packaging Building with maven takes time, especially when starting from an empty local repository, while reusing jar files built elsewhere is not a problem since they are portable across platforms.

Only the portable softare artifacts such as jar files, php pages and shell scripts are packaged from pre-built archives. Source code that needs to be compiled on each platform such as C language or Python is packaged from source files.

2.5.3 Target platforms

Table 1 shows the platforms and architectures for which the release 1.0 of Contrail are availables. Currently the target platforms are Debian and Ubuntu, and we intend to add Redhat in few months.

2.5.4 Updating packaging configuration

Updating the packaging configuration executes the following steps for each package:

1. Download a copy of the current package configuration

[13] about 50 megabytes must be downloaded to install maven and its dependencies, and probably more than 200 megabytes of external jar dependencies would also have to be downloaded during the build to populate the local Maven cache.

	i586	x86_64
Debian 6	Yes	Yes
Ubuntu 11.04	Yes	Yes
Ubuntu 11.10	Yes	Yes
RHEL6	No	No
Fedora16	No	No

Table 1: Targets platforms and architectures of the Contrail release 1.0

2. Update source archive with the pre-build artifacts from Bamboo
3. Increment version numbers in configuration files
4. Upload modified files

The shell script used to update packages is included in appendix B.1.

2.6 Validation Tests

Validation tests are conducted to determine whether binary packages meet the minimal quality criteria to be published. Running validation tests executes the following steps for each combination of package, operating system and architecture:

1. Start a new virtual machine for a given architecture and operating system.
2. Configure the Contrail package repository to be tested[14].
3. Install the package.
4. Uninstall the package.

The shell script used to validate packages is included in appendix B.3.

2.7 Release of binary packages

Figure 10 shows how the different stages of the release process are organized into three main steps. This part only describes the main steps, the detailed procedure in available in appendix A.

2.7.1 Binary packages repositories

Binary packages are published on a server named "Repo" in figure 1 and are organized into three repositories:

release Contains stable binary packages for Contrail users.

[14] testing, staging or release

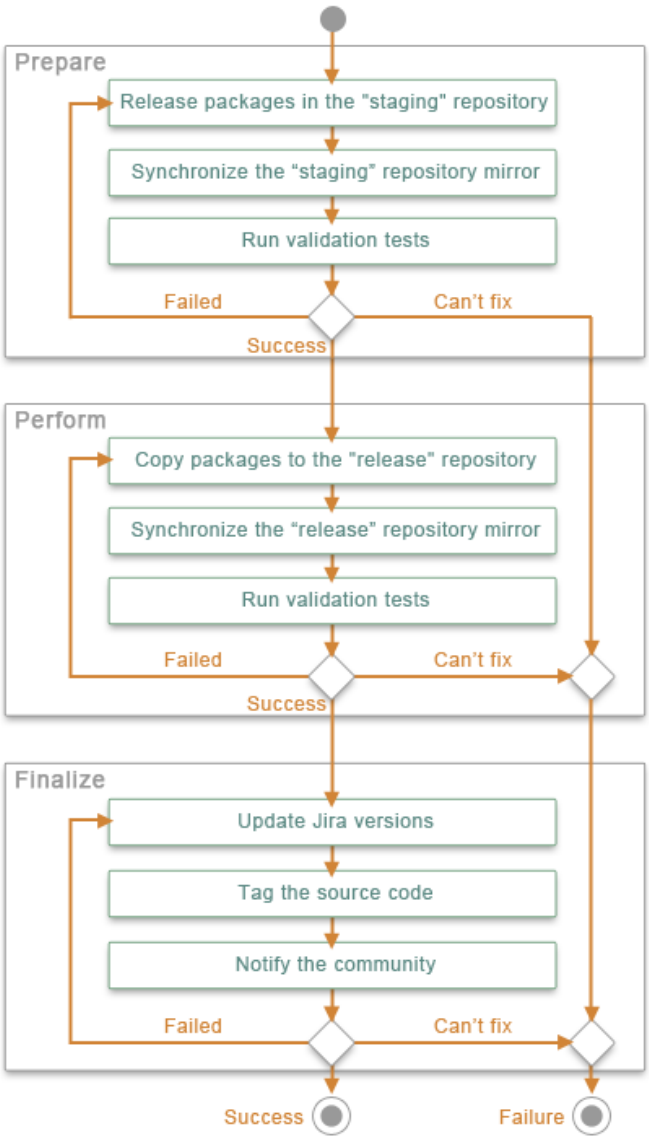


Figure 10: Release process of the Contrail software

staging Contains binary packages prepared for the next release, which are at available for integration tests and updated manually.

testing Contains latest binary packages for testing by developers and is updated automatically every day.

	Scheme	Example	Repository
Nightly Build	0+*id*	0+802	testing
Release Candidate	*x.y*~RC*z*+*id*	1.0~RC1+812	staging or release
Final Release	*x.y.z*+*id* or *x.y*+*id*	1.0+812	release

Table 2: Versioning scheme

	Unit Tests	Integration Tests	Validation Tests
Nightly Build	Yes	No	No
Release Candidate	Yes	Yes	No
Final Release	Yes	Yes	Yes

Table 3: Testing scheme

2.7.2 Main steps

Prepare After a certain amount of short development iterations, when the packages configuration in the testing repository looks fine, it is copied to the staging repository. Then the source files[15] of each package are updated with the result of the selected Bamboo build id and the configuration files are updated with the chosen version string.

Perform Packages in the staging repository are intensively tested and once they are validated, both configuration and source files are copied to the release repository without any modification.

Finalize Once all mirrors are synchronized, an OW2 news is published and an email is sent to the Contrail community so as to spread the news.

2.7.3 Types of releases

In the Contrail project, we have three types of releases:

Nightly Builds Used by developers to tests the very latest package of their code

Release Candidates Used by testers to perform integration tests

Final Releases Used by users in production, expected to be stable

Table 2 shows the versioning scheme used in Contrail for each type of release, while table 3 points out which tests are run.

[15] archives of pre-build artifacts

Date	Developement	Integration	Packaging	Release
03/23			Finish	1.0~RC1
03/24				
03/25				
03/26	Bug Fixes		List Freeze	Prepare
03/27	Bug Fixes	Run Tests		1.0~RC2
03/28	Bug Fixes			
03/29	Bug Fixes			Prepare
03/30	Bug Fixes	Run Tests		1.0~RC3
03/31				
04/01				
04/02				Prepare
04/03			List Unfreeze	1.0

Table 4: Time Schedule of Release 1.0

2.7.4 Time schedule for the release 1.0

Table 4 shows the schedule of the week before the first release for developers, testers and maintainers. During this last week developers were asked to concentrate on fixing bugs instead of developing new features. In order to perform integration tests and also to validate the release process, we published three release candidate versions.

3 Getting started with Contrail

3.1 List of binary packages

For the release 1.0 of the Contrail software, we made 29 binary packages available for the Debian GNU/Linux distribution.

Package Name	Short Description
conpaas-frontend	ConPaaS frontend Server
conpaas-frontend-db	ConPaaS frontend database
conpaas-mysql	ConPaaS MySQL elastic server
conpaas-scalarix-one-client	Scalarix Client for ConPaaS
conpaas-scalarix-one-frontend	Scalarix Manager daemon for ConPaaS
conpaas-scalarix-one-manager	Scalarix Manager daemon for ConPaaS
conpaas-taskfarm	ConPaaS generic task scheduler
conpaas-web	ConPaaS Web
contrail-ca-server	Contrail CA Server
contrail-federation-api	Contrail Federation API
contrail-federation-cli	Contrail Federation CLI
contrail-federation-db	Contrail Federation Database
contrail-federation-id-prov	Contrail Federation ID provider
contrail-monitoring-hub	Contrail Monitoring Hub Server
contrail-one-monitor	Contrail OpenNebula Monitor
contrail-one-sensor	Contrail OpenNebula Sensor
contrail-provider-accounting	Contrail Provider Accounting
contrail-provisioning-manager	Contrail OpenNebula Sensor
contrail-rest-monitoring	Contrail REST Monitoring
contrail-safeconfig	Contrail safeconfig
contrail-vep-cli	Contrail Virtual Execution Platform CLI
contrail-vep-gui	Contrail Virtual Execution Platform GUI
contrail-vin	Contrail Virtual Infrastructure Network
scalaris	Scalable Distributed key-value store
scalaris-doc	Documentation for scalaris
xtreemfs-backend	XtreemFS server
xtreemfs-client	XtreemFS client
xtreemfs-server	XtreemFS server
xtreemfs-tools	XtreemFS administration tools

Table 5: List of binary packages

3.2 Installing a package

The Contrail software is available in standard Linux repositories and intalls very easily on Linux distributions. A detailed install procedure is included in appendix 4 and is also available online in the download page of the wiki[19].

3.3 Deploying the Contrail system

Figure 11 shows a typical installation of the Contrail System to provide users with a frontend to a cloud federation. Each site is represented with the contrail binary packages to be installed on it. So far, we identified four software stacks:

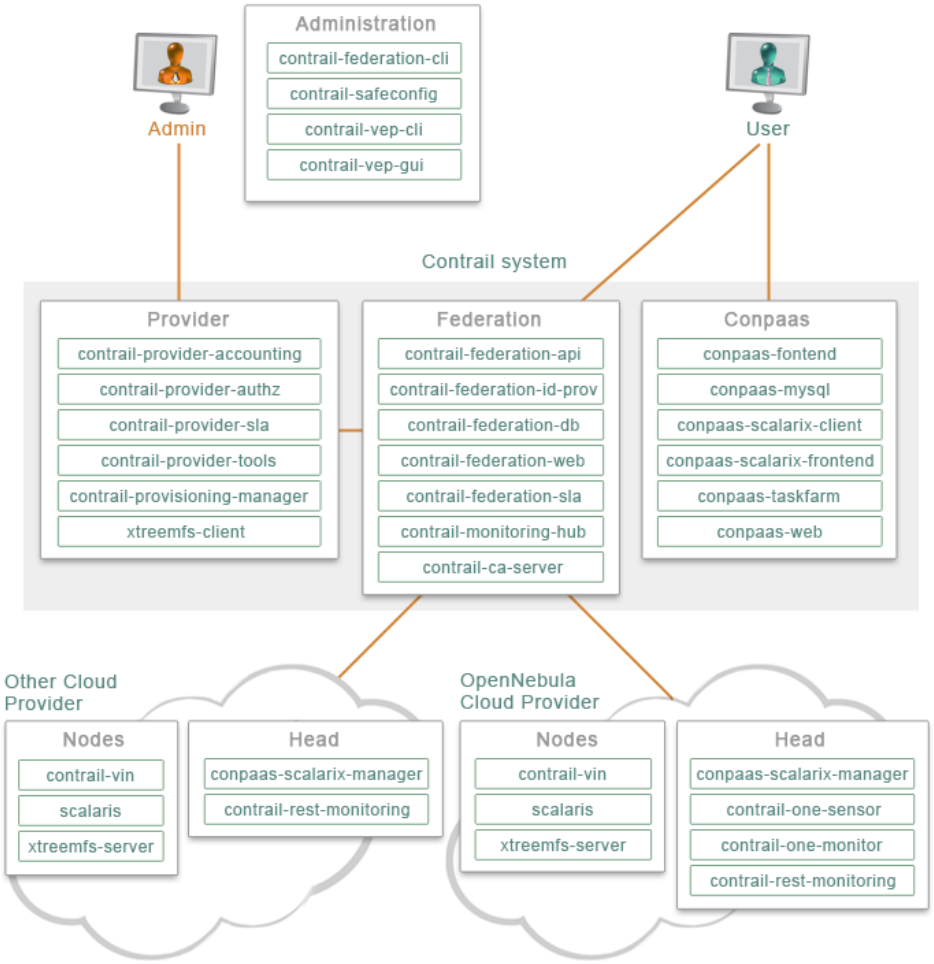


Figure 11: Overview of a typical deployment of Contrail

Contrail system A cloud providers which exposes the Contrail federation API and the ConPaaS webapp, and provides SLA and security services.

Open Nebula cloud provider A cloud provider which exposes the API of Open Nebula.

Other cloud provider Another cloud provider which exposes another API (e.g. Amazon).

Administrator The person in charge of managing the Contrail system.

Binary packages associated with each software stack are listed in figure 11, more documentation about each package should be available online[15].

3.4 Getting support

Contrail users can get support by using IRC[21], Jira[9], or one of the mailing lists[22]. The different manners to get help are also documented on the wiki[20]. The idea is to promote mutual help among the community of users by providing the tools to exchange information.

4 Install procedure for Debian GNU/Linux and Ubuntu

For the users convenience, the install procedure is documented on the download page[19] of the wiki. This section explains the installation procedure for the Debian/Ubuntu Linux distribution only.

This is step-by-step installation guide of the Contrail system. In case of any problems, please refer to the documentation of the component for detailed instructions (Contrail AdminGuide). In order to install the complete system, you need to set up three different parts:

- provider head node (for example, the head node of the ONE cluster),
- provider worker node (for example, the worker node in the ONE cluster, can be the same as head node),
- federation node (different node).

4.1 Package sources

This chapter is about getting Contrail binaries for installation and how to proceed in different installation scenarios (CD, Download repositories or Virtual Machine).

4.1.1 Setting up package repositories

In order to include Contrail repositories, select one of the following supported distributions (Debian 6.0, Ubuntu 11.04 and Ubuntu 11.10) and add one of the following lines at the end of the `/etc/apt/sources.list`. Let us note, that the current state of the Contrail stack works best on the newest versions of Ubuntu distributions, also on Debian 6.0 with unstable packaging. The issues with stable version of Debian 6.0 are described in sections 4.1.2 and 4.1.3.
Stable build:

```
deb http://contrail.ow2.org/repositories/binaries/release/Debian_6.0/ ./
deb http://contrail.ow2.org/repositories/binaries/release/xUbuntu_11.04/ ./
deb http://contrail.ow2.org/repositories/binaries/release/xUbuntu_11.10/ ./
```

Daily build:

```
deb http://contrail.ow2.org/repositories/binaries/testing/Debian_6.0/ ./
deb http://contrail.ow2.org/repositories/binaries/testing/xUbuntu_11.04/ ./
deb http://contrail.ow2.org/repositories/binaries/testing/xUbuntu_11.10/ ./
```

Once repositories are added to the list, the following command adds a key for the repositories:

```
wget -O - http://contrail.ow2.org/repositories/contrail.pub | sudo apt-key add -
```

followed by the updating of the repositories

```
sudo apt-get update
```

If `aptitude` packages are installed, you can check whether contrail packages are available by executing the following command (in order to install aptitude, use `sudo apt-get install aptitude` first):

```
aptitude search contrail
```

The command should return the list of packages. If the list is empty, either the repositories or the key are not properly set.

4.1.2 RabbitMQ

The version of RabbitMQ that is required by the Contrail is newer than the one available in the Ubuntu 11.04 and Debian 6.0 stable repositories. In order to avoid package conflicts between official old version of RabbitMQ included in the named distribution and ones required by the Contrail system, we recommend to install RabbitMQ directly from the official RabbitMQ page at `http://www.rabbitmq.com/install-debian.html`.

4.1.3 Python and Zookeeper

The current state of the Contrail stack has some missing functionality. In order for the Web interface to remember the connections between user's applications, deployed VMs, its IDs and IDs on the OpenNebula clusters, we used an additional bookkeeping framework ZooKeeper. However, there are some problems in the packaging of the stable version of Debian 6. In order to solve this problem, we suggest to install the library before installing the *contrail-federation-web* package. To install it manually, perform the following steps:

```
sudo apt-get install python-dev python-setuptools
sudo easy_install pip
sudo pip install zc-zookeeper-static
```

Please note that the *python-dev* and *python-setuptools* packages might already be installed on your system.

4.2 OpenNebula installation

In the following installation scenario, we are going to use `oneadmin` as an OpenNebula user and `oneadmin` as OpenNebula password for the user. Also, we are going to use MySQL user `root` set without password. If your installation is different, please act accordingly. The commands are used for `root` user. In case of installing the software under the normal user, use `sudo` when necessary.

4.3 OpenNebula Head Node

We start with the installation of the software stack on ONE head node. In our example ONE head is located at hostname (DNS name) `n0004`.

To install the software stack that is common for all providers, execute:

```
apt-get install contrail-provider-common
```

The package currently contains the following components:

- contrail-provisioning-manager,
- contrail-rest-monitoring,
- xtreemfs-client.

To install software stack that is specific for the ONE cluster, execute:

```
apt-get install contrail-provider-one-head
```

This package currently contains the following components:

- conpaas-scalarix-one-manager,
- contrail-one-monitor.

On some system, the following error might occur during the installation of the contrail-provider-one-head package:

```
ERROR: Could not find a valid gem 'oca' (>= 0) in any repository
ERROR: Possible alternatives: oca
1 gem installed
dpkg: error processing conpaas-scalarix-one-manager (--configure):
subprocess installed post-installation script returned error exit \
        status 2
```

To get past that error, use the following command and repeat the procedure.

```
gem install oca
apt-get install contrail-provider-one-head
```

4.3.1 Installing VEP

Let us start with the component that currently needs most configuration, VEP. There are two packages, one with GUI (`contrail-vep-gui`) and one for CLI (`contrail-vep-cli`). We've used the CLI version of the component. If you prefer to use GUI version, use the documentation that is available in the Contrail user guide. If `contrail-vep-cli` package is not part of the `contrail-provider-one-head`, yet, you need to install it manually[16].

```
apt-get install contrail-vep-cli
```

To check if vep is running use:

```
ps -x | grep vep
```

If VEP component is not available, the following error might have occurred when running:

[16] At time of writing this guide, `contrail-vep-cli` still was not part of the meta-package.

```
root@n0004:~# contrail-vep-cli -d
VEP system properties file and VEP logger properties paths were \
        not specified, using the default path for vep properties \
        (/root/.vep-cli/vep.properties) and created a new default \
        logger properties files
SLF4J: Detected both log4j-over-slf4j.jar AND slf4j-log4j12.jar \
        on the class path, preempting StackOverflowError.
SLF4J: See also http://www.slf4j.org/codes.html#log4j\
        DelegationLoop for more details.
Exception in thread "main" java.lang.ExceptionInInitializerError
    at org.apache.log4j.Logger.getLogger(Logger.java:39)
    at org.ow2.contrail.provider.vep.VEPHelperMethods.<init>\
    (VEPHelperMethods.java:32)
    at org.ow2.contrail.provider.vep.VEPStart.<init>(VEPStart.\
    java:85)
    at org.ow2.contrail.provider.vep.VEPStart.main(VEPStart.\
    java:1048)
Caused by: java.lang.IllegalStateException: Detected both \
        log4j-over-slf4j.jar AND slf4j-log4j12.jar on the \
        class path, preempting StackOverflowError. See \
        also http://www.slf4j.org/codes.html#\
        log4jDelegationLoop for more details.
    at org.apache.log4j.Log4jLoggerFactory.<clinit>\
    (Log4jLoggerFactory.java:49)
    ... 4 more
```

To correct this error, edit the script that runs the component and remove the references to `log4j-over-slf4j.jar` and `slf4j-log4j12.jar`:

```
nano /usr/bin/contrail-vep-cli
```

remove the topmost two files from the list and save the file.
We need to copy the following files for VEP to work properly:

```
mkdir /root/.vep-cli/
cp /usr/share/contrail/contrail-vep-cli/vep.properties /root/.vep-cli/
cp /usr/share/contrail/contrail-vep-cli/VEPKeyStore.jks .vep-cli/
```

Once configuration files are copied, edit the `vep.properties` file and set ONE user and password, as well as part of the ONE installation details.

```
cd /root/.vep-cli
nano vep.properties

  mysql.pass=contrail
  one.ip=
  one.port=2633
  one.user=oneadmin
  one.pass=oneadmin
  pdp.use = true/false
  contrail.cluster=1
```

VEP currently expects that ONE cluster exists. This can be made with the following command:

```
onecluster create contrail # creates cluster with ID 1
```

Try running VEP again with calling the script `contrail-vep-cli` (do not use switch `-d` now).
If output of the VEP indicates errors, such as:

```
/user/gregorb DEBUG dbHandler - Executing query: select * from \
        user where username='fedadmin'
qtp1482258114-21 DEBUG log - EOF org.eclipse.jetty.io.EofException
```

This indicates that the problem occurs when there is no `fedadmin` user in MySQL database (or no user in the table user). To get past this, the following SQL statements are currently missing when running VEP:

```
root@n0004:~# SQL="use vepdb;"
root@n0004:~# SQL+="insert into ugroup (gname,uid) values ('admin', 1);"
root@n0004:~# SQL+="insert into user (username,uid,vid,oneuser,\
        onepass,oneid,role) values ('fedadmin',1,-1,0,\
         '7bc8559a8fe509e680562b85c337f170956fcb06',-1,'admin');"
root@n0004:~# mysql -u root -e "$SQL"
```

Now, try using VEP with Telnet. As stated in the documentation, VEP REST server listens on port 10500 while telnet server listens on port 10555.

```
# use vep-cli via telnet
telnet localhost 10555
  locadmin
  loc1234
  yes
  ladmin
  l1234
```

As described in the section about the VEP component, the following commands add requested to properly set the data (datacenter, cluster, rack, host) that is needed to identify the ONE setup (in the next listing we use "/" to break the line):

```
# noninteractive telnet, check console output if command did succeed
echo -e 'ladmin\nl1234\nadd datacenter\/
        ndatac1\nSI\ndc-1-desc\ny\n\n\n' | nc localhost 10555
echo -e 'ladmin\nl1234\nadd cluster\ncl1\/
        n001\n1\ncl-1-desc\ny\n\n\n' | nc localhost 10555
echo -e 'ladmin\nl1234\nadd rack\nrc1\n1\/
        nrc-1-desc\ny\n\n\n' | nc localhost 10555
echo -e 'ladmin\nl1234\nadd host\nn0004\nim_kvm\nvmm_kvm\ntm_nfs\ny\/
        nexit\n' | nc localhost 10555
echo -e 'ladmin\nl1234\nadd fedadmin\n0\n1\n1\/
        nhost-vep-1\ny\n' | nc localhost 10555
```

To check if all is properly set call:

```
onehost list
onecluster list
```

The final step is to make sure that ONE virual networks and images which are referenced by the OVF should be public. While `oneimage` can be created on-the-fly by the VEP, `onevnet` virtual network has to be pre-created and explicitly published.

```
onevnet publish 0
```

This finalizes our installation of VEP.

4.4 OpenNebula Worker Node

4.4.1 ONE Sensors

When VEP install is finished, we can continue installing `contrail-provider-one-node` package on each ONE node. The package contains the following components:

- contrail-vin,
- scalaris,
- xtreemfs-server,
- contrail-one-sensor

Currently there is a missing dependency on `sysstat` package, therefore the installation for ONE worker node looks like:

```
apt-get install contrail-provider-one-node sysstat
```

Edit the following configuration and set the address of ONE head node for RabbitMQ host:

```
nano /etc/contrail/contrail-one-sensor/one-sensor.config
```

Example of the content of the configuration file (changed values only):

```
rabbit_mq_host = n0004
host_properties_file = /etc/contrail/contrail-one-sensor/hostConfig
```

This concludes the installation of the ONE worker node.

4.5 OpenNebula Head Node Continued

4.5.1 ONE Monitor

Once the sensors for OpenNebula are properly configure, we can continue setting up ONE head node. Next step is to set up contrail-provider-one-monitoring package by editing the following configuration file:

```
/etc/contrail/contrail-one-monitor/one-monitor.config
```

Change the host of the federation node (in our case, the federation host is n0005):

```
federation_finagle_host=n0005
```

Restart the component by calling:

```
/etc/init.d/contrail-one-monitor restart
```

If the contrail-provider-one-monitroing component is running or not can be seen by checking the /tmp/contrail-one-monitor.log file with

```
cat /tmp/contrail-one-monitor.log
```

If there are errors due to missing/pending connection to the monitoring-hub, ignore that. The connection will be restored once the monitoring-hub is installed and set up on the federation node.

4.5.2 REST Monitoring

To properly configure contrail-rest-monitoring package, edit the following config file:

```
/usr/share/contrail/common/rest-monitoring/config
```

And change the location of the directory with ONE images. The location of the ONE images depends on the ONE deployed. In our case, the changed part of the configuration file looks like:

```
image_cache_dir = /srv/one-images-2
```

4.6 Federation

Once providers are set, we need to set the federation node. Our federation node is located on host n0005.

Components in the contrail-federation:

- contrail-federation-api,
- contrail-federation-db,
- contrail-federation-id-prov,
- contrail-federation-web,
- contrail-monitoring-hub.

4.6.1 Monitoring Hub

For basic setup, we need to set the monitoring-hub and the federation-web components. The monitoring-hub must know where to locate the federation RabbitMQ. To set this up, we need to configure the following file:

```
/etc/contrail/contrail-monitoring-hub/config.json
```

And change the values:

```
rabbit,enabled=true
rabbit,host=n0005
```

When configuration is complete, restart the service by calling:

```
/etc/init.d/contrail-monitoring-hub restart
```

The following output is an example of a properly set monitoring-hub.

```
688731 [New I/O server worker #1-7] INFO org.ow2.contrail.\
        monitoring.hub.HubServer$ - PUT /metrics/route/host.\
        n0004-xc2-xlab-lan.disk
688731 [New I/O server worker #1-7] INFO org.ow2.contrail.\
        monitoring.hub.HubServer$ - New metric for route host.\
        n0004-xc2-xlab-lan.disk for provider 1
```

The easiest way to check if the monitoring-hub is properly set is to use the stdout for messages. This is obtained by using the following commands:

```
# To view log file - it is stdout actually, so
/etc/init.d/contrail-monitoring-hub stop
contrail-monitoring-hub
```

4.6.2 Federation Web

The final component that needs to be configured is contrail-federation-web. Currently, contrail-federation-cli and zookeeper (only a temporary solution) should be installed manually:

```
apt-get install contrail-federation-cli zookeeperd
```

Continue the configuration by setting up the configuration file which is located at:

```
/etc/contrail/contrail-federation-web/federation-web.conf.
```

We need to set up the locations of the federation and provider head nodes. In our case, the changed values are:

```
FEDERATION_API_URL = http://n0005:8080/federation-api
SLA_EXTRACTOR_BASE = http://n0004:8080/rest-monitoring/sla/slaextractor
MONITORING_BASE = http://n0004:8080/rest-monitoring/monitoring
```

Some parts of the configuration (user data, not install) are not included in the GUI, yet. The script, which is listed in the following lines, is accessible on-line[17]. The following commands add provider, server, and SLAT (SLA Template) via `contrail-federation-cli` (provider can be added via GUI; actually only `PROVIDER_ID` is required for add-server and add-slats commands):

```
export FEDERATION_IP=n0005

export IP=n0004

export HOSTNAME=n0004

export FEDERATION_CLI_URL="http://n0005:8080/federation-api"

PROVIDER_ADD_DATA=$(contrail-federation-cli add-provider -data "{'name':\
'CloudProvider2','email':'cloudprovider2.com','country':'UK','typeId':42,\
'providerUri':'http://$IP:10500'}")

export PROVIDER_ID=$(echo $PROVIDER_ADD_DATA | python -c "import sys; import \
json; print json.loads(sys.stdin.read())['headers']['Location']\
.split('/')[-1]")

contrail-federation-cli add-server -providerId $PROVIDER_ID -data "\
{'name': '$HOSTNAME', 'ram_total': '3915', 'ram_used': '1152', \
'ram_free': '2763', 'cpu_cores': '4', 'cpu_speed': '2494.276', \
'cpu_load_one': '0.09', 'cpu_load_five': '0.04'}"

contrail-federation-cli add-slat -providerId $PROVIDER_ID -data "\
{'name': 'XLAB SLAT', 'url': 'http://contrail.xlab.si/test-\
files/ubuntu-test-xlab-SLA.xml'}"
```

Web GUI should now be available at `http://n0005/`.

The CloudCoordinator can login as `coordinator:password`. He creates new users with role `FederationUser`. New users are able to use provider SLATs and to deploy their applications (more on this in the User's Guide).

You can also check if the `contrail-federation-api` is working properly by calling (and obtaining non-empty JSON file in response):

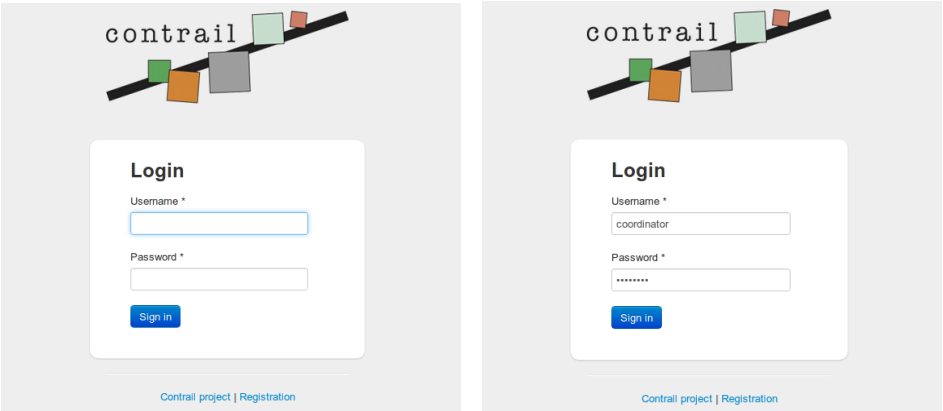
[17] `http://contrail.xlab.si/test-files/init_fed.sh`

```
apt-get install curl        # In case you do not have 'curl' \
        installed yet
curl http://localhost:8080/federation-api/users # check, by \
        querying federation-api rest interface
```

This conclude the installation steps of the Contrail components. The usage of the system is described in the User Manual.

5 Using Federation Web

This package provides web interface for Contrail Federation API. In this chapter we describe three different user views based on the role of the user. Since **federation coordinator** is the one who is able to manage the federation, we first provide coordinator's view. **Cloud administrator** is able to manage a cloud provider. Last but not least, we describe the view of a **Federation user**.



(a) First page (b) Coordinator must log in on first page

Figure 12: Figure (a) depicts first log-in page for Contrail users, figure (b) is an example for user *coordinator*

5.1 Federation coordinator

Federation coordinator has privileges to manage the federation. Default installation of Contrail provides a default account for federation coordinator. Credentials are

- username: *coordinator*
- password: *password*

The management of the federation consists of: *user management* and *cloud provider management*. After logging into the federation console (figure 13), coordinator can manage the list of users (depicted in figures 14, 15).

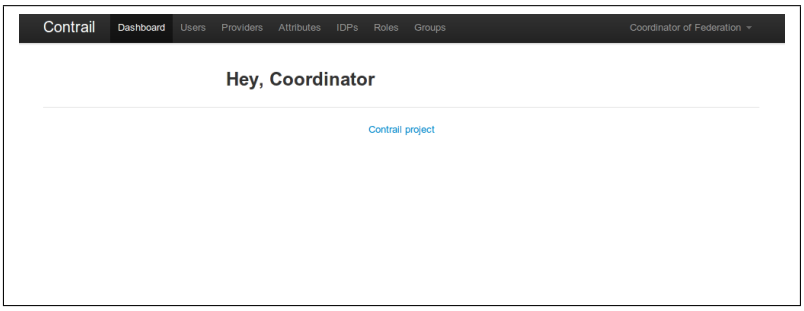


Figure 13: Coordinator's home page – figure depicts coordinator's home page

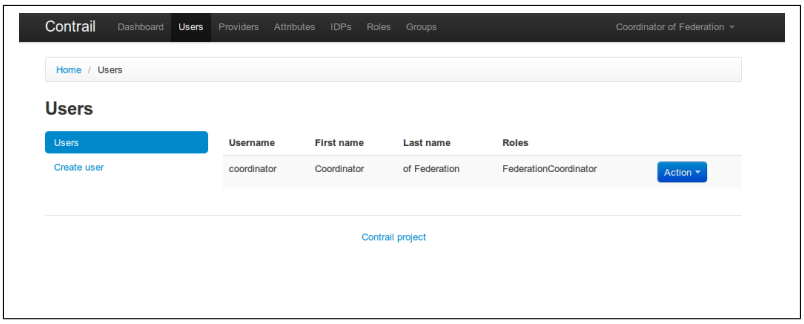


Figure 14: List of users. Figure shows a list of users managed by coordinator

User management User management consists of *creating a new user*, *defining user's role*, *group* and *attributes*. User creation is depicted in figure 15. User role and group management is shown in figures 16 and 17, respectively. Attribute management is shown in figure 18.

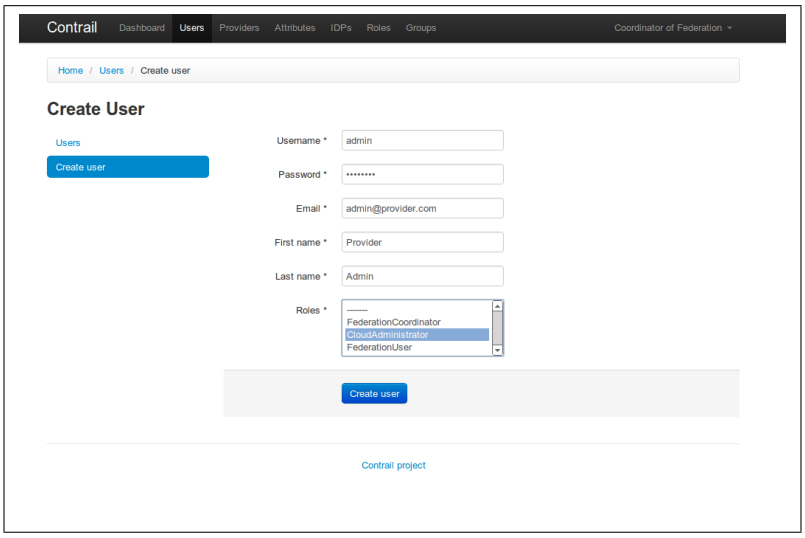


Figure 15: Coordinator can create new users. Figure depicts creation of a new CloudAdministrator user.

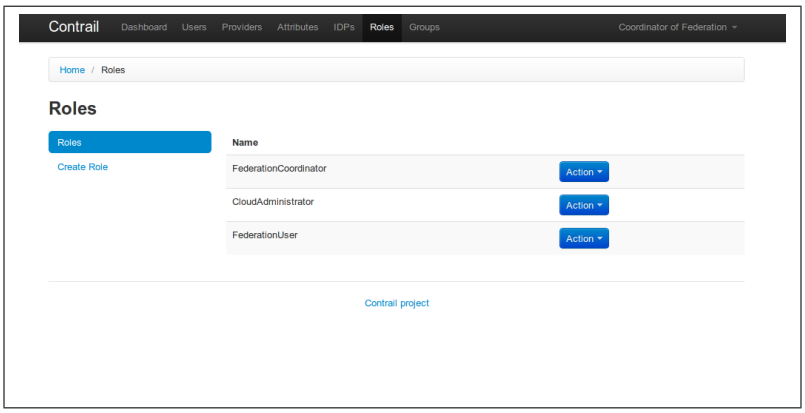


Figure 16: Coordinator can manage list of user roles.

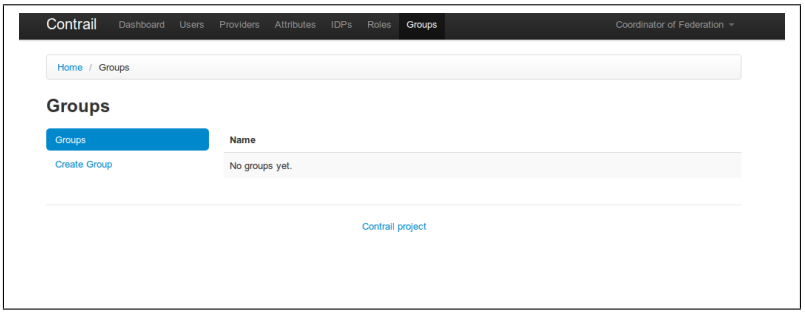


Figure 17: Coordinator can manage list of user groups.

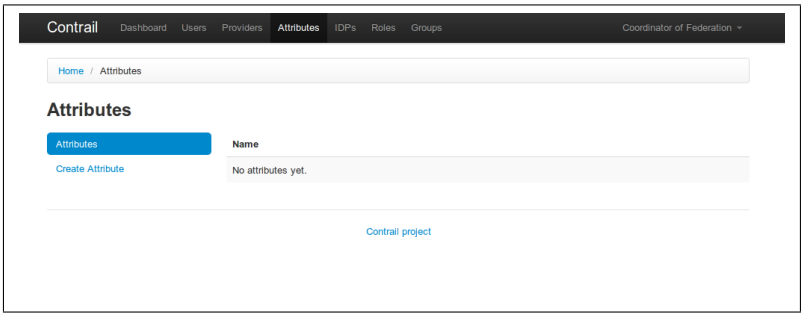


Figure 18: Coordinator can manage list of possible user attributes.

Figure 19 shows management of *Identity Providers*. Since Contrail currently supports only internal identity provider, this feature is not yet supported.

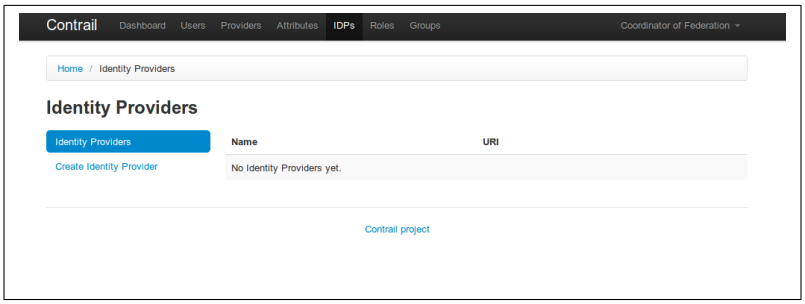


Figure 19: Coordinator can manage the list of identity providers.

Cloud provider management Coordinator can manage basic properties related to cloud providers. Basic task of a federation coordinator is to manage the list of the providers. Figure 20 depicts a form for creating/removing providers in the federation. Input for **Name** and **Provider URI** is expected. Provider's URI is an access point to provider's VEP instance (for detail on VEP component, see Contrail Administration Guide for more detailed description and Contrail Installation guide for quick set-up).

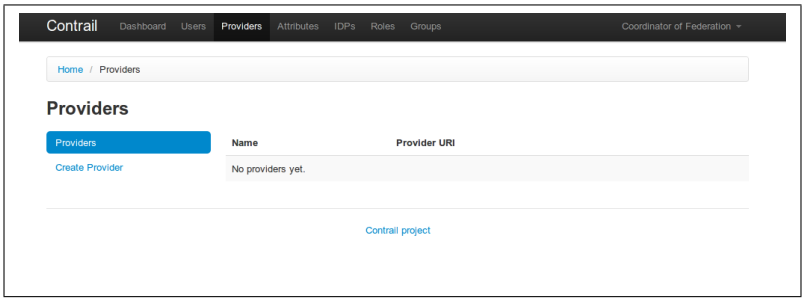


Figure 20: Coordinator can manage list of providers.

5.2 Cloud administrator – administering providers

The Cloud administrator administers individual cloud provider that wishes to be a part of the cloud federation. Default installation of Contrail provides a default account for cloud administrator. Credentials are

- username: *admin*
- password: *password*

The Cloud provider must be register on the federation using `contrail-federation-cli` before users can negotiate SLAs, register and submit applications. Following lines can be used for registration of a new cloud provider (details are given after the listing bellow):

```
1  FEDERATION_IP=10.0.2.2
2  IP=$(ifconfig eth0 | grep inet | grep -v inet6 | awk '{print $2}'|sed \
3    's/addr://')
4  HOSTNAME=$(hostname)
5  export FEDERATION_CLI_URL="http://$FEDERATION_IP:8080/federation-api"
6
7  PROVIDER_ADD_DATA=$(contrail-federation-cli add-provider -data "{'name':\
8    'CloudProvider', 'typeId': 42, 'providerUri': 'http://$IP:10500'}")
9  PROVIDER_ID=$(echo $PROVIDER_ADD_DATA | python -c "import sys; import \
10   json; print json.loads(sys.stdin.read())['headers']['Location']\
11   .split('/')[-1]")
12
13 contrail-federation-cli add-server -providerId $PROVIDER_ID -data "\
14   {'name': '$HOSTNAME', 'ram_total': '3915', 'ram_used': '1152', \
15   'ram_free': '2763', 'cpu_cores': '4', 'cpu_speed': '2494.276', \
16   'cpu_load_one': '0.09', 'cpu_load_five': '0.04'}"
17 contrail-federation-cli add-slat -providerId $PROVIDER_ID -data "\
18   {'name': 'XLAB SLAT', 'url': 'http://contrail.xlab.si/test-\
19   files/ubuntu-test-xlab-SLA.xml'}"
```

Environmental variable `FEDERATION_IP` is the IP of the machine running Federation API (line 1). Variable `HOSTNAME` (line 4) presents hostname of the provider's server. In line 5 admin sets URL of the Federation API, which is used later. With the command in line 7 admin creates a provider with the federation. Returned value from line 7 is passed to a command depicted in line 9. From the returned value administrator extracts `PROVIDER_ID` and uses it in line 13. With the command in line 13 admin adds a new server to the cloud provider. Variables `PROVIDER_ID` (from line 7) and `HOSTNAME` (from line 4) are used in this command. This way the administrator adds a new SLA template to the provider.

Now, the administrator needs to login into the federation using provided form. Cloud administrator enters credentials as depicted in figure 21.

Cloud administrator sees the list of clouds she can administer (figure 22). The one depicted in the figure was created with the commands from the listing in the beginning of the section 5.2.

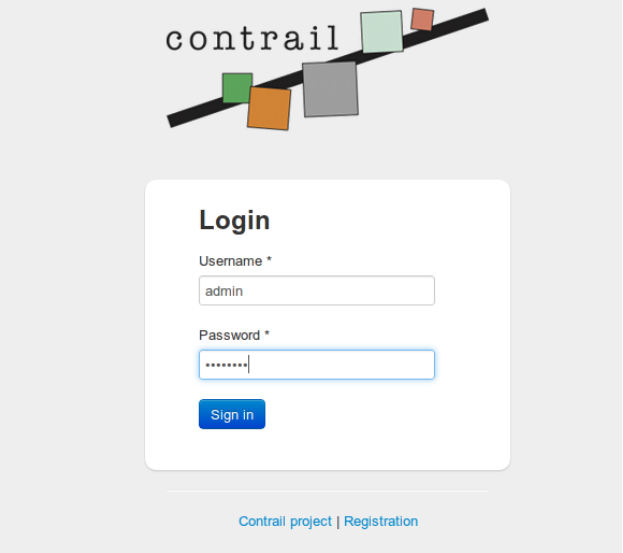


Figure 21: Login page for the Cloud administrator.

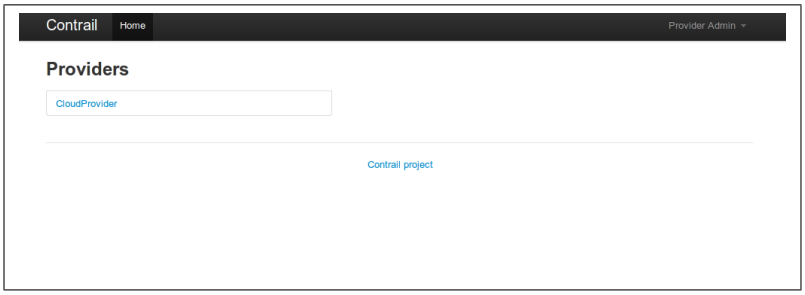


Figure 22: First cloud administrator's page. First, she must chose a provider.

Cloud administrator's dashboard, which provides basic tasks for administering the cloud, is depicted in figure 23. Administrator can take on next tasks:

- management of OVFs
- management of SLA templates
- management of virtual machines (VMs)
- management of cloud servers
- management of cloud's clusters
- management of virtual organizations

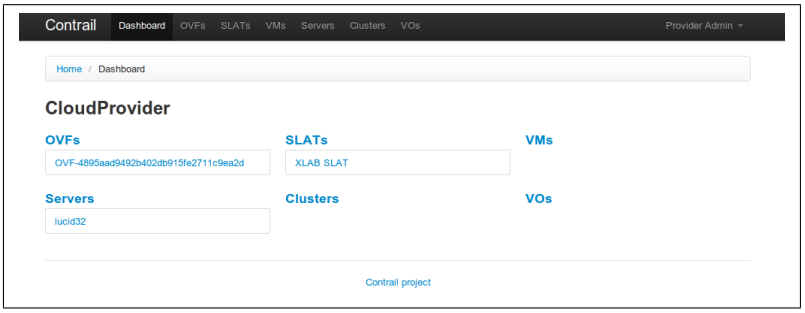


Figure 23: Dashboard

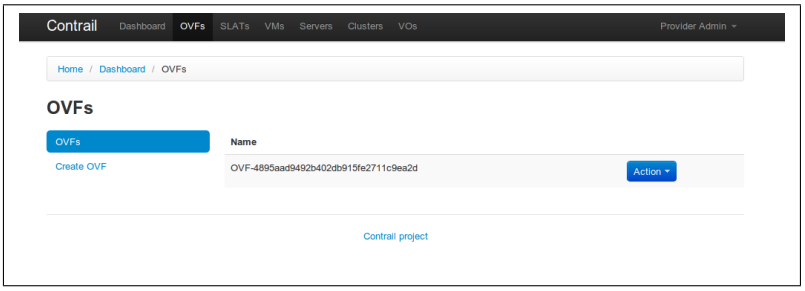


Figure 24: List of OVFs

In the figure 25 administrator can choose between available SLA templates. She can also provide a new SLA template by clicking on **Create SLAT**. In the form, which is shown, she must provide a SLAt's **Name** and **URL**. Both attributes are mandatory. The later has point to a SLA template file accessible through HTTP.

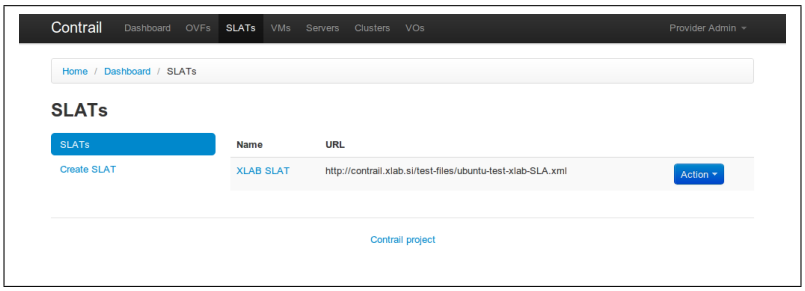


Figure 25: List of available SLA templates.

In the figure 26 the list of running VMs is depicted. When new user applications are created the list is automatically updated.

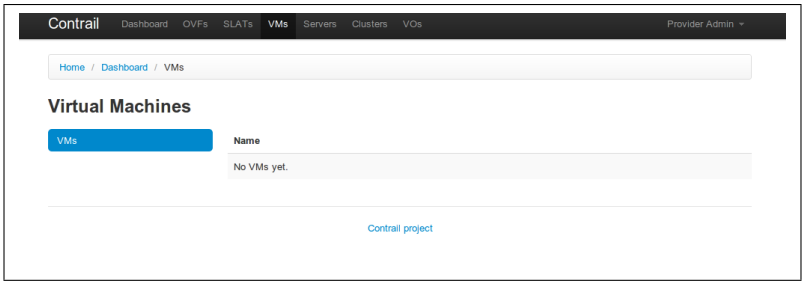


Figure 26: List of virtual machines.

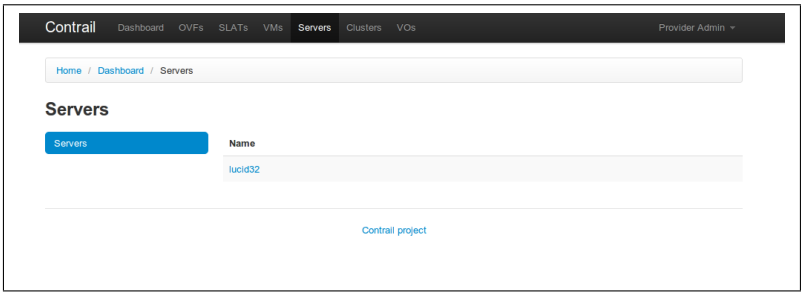


Figure 27: List of servers.

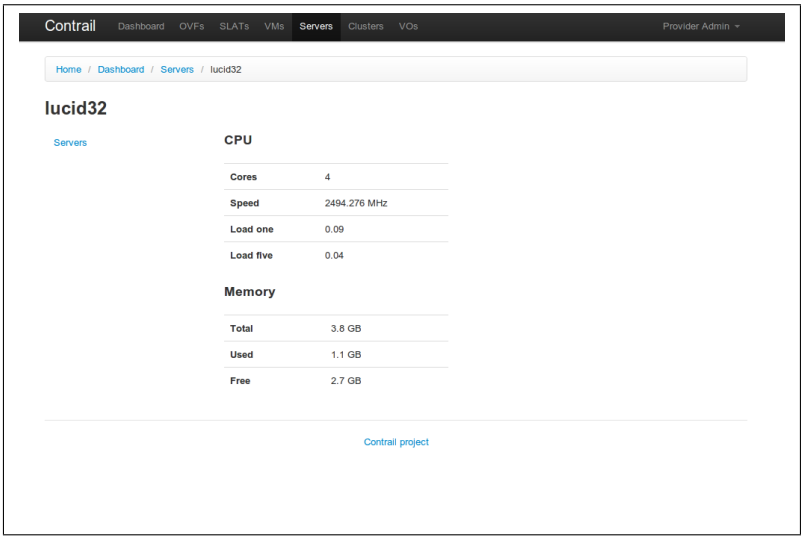


Figure 28: Server details.

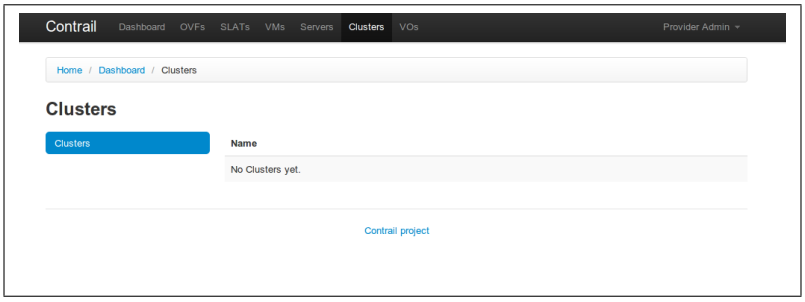


Figure 29: List of clusters.

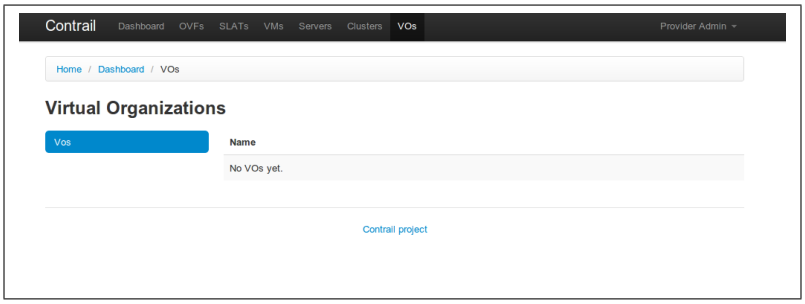


Figure 30: List of virtual organizations.

5.3 User

User has to register with the federation if she wishes to use it (see figure 31). Default Contrail installation does not provide any default user account. Instead, either federation coordinator has to provide one or user registers with the federation web interface. The coordinator has the ability to see a list of users and there is no need to confirm a registration with the federation coordinator.

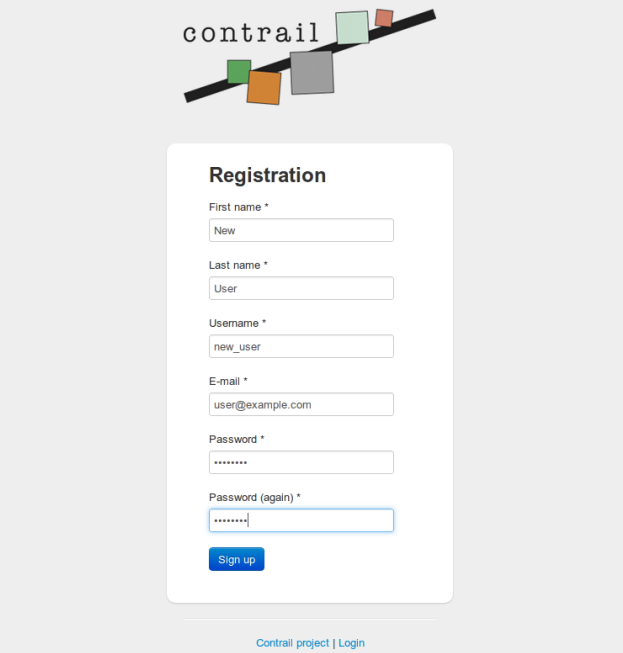


Figure 31: User must first registers with the federation.

In case of successful registration user can log-in via the federation web gui (figure 32).

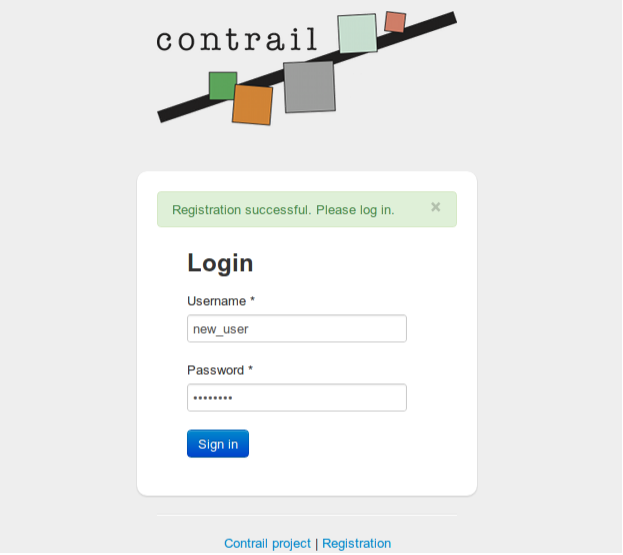


Figure 32: If registration is successful, the user can log in.

When user logs in, she sees two buttons, namely **SLA Negotiation** and **Register application** (figure 33). In order to create a new SLA (she wishes to negotiate an SLA), she should click on SLA Negotiation, where she can set limits to metrics on the provider (RAM total, RAM free, CPU cores, CPU speed, CPU load one, CPU load five).

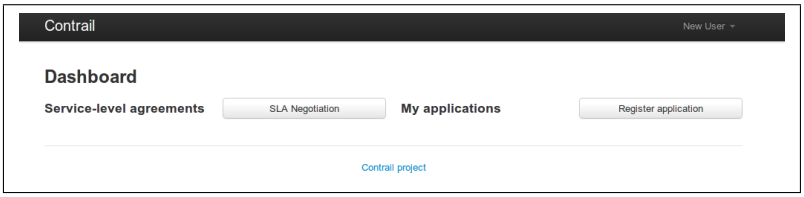


Figure 33: Dashboard contains the list of SLAs and applications.

In case the limits match with the provider, available match is listed just below the SLA Negotiation scroll bars, under section **Matching providers** (figure 34). Clicking on the link provided by matching providers list takes user to the form similar to the one on figure 35.

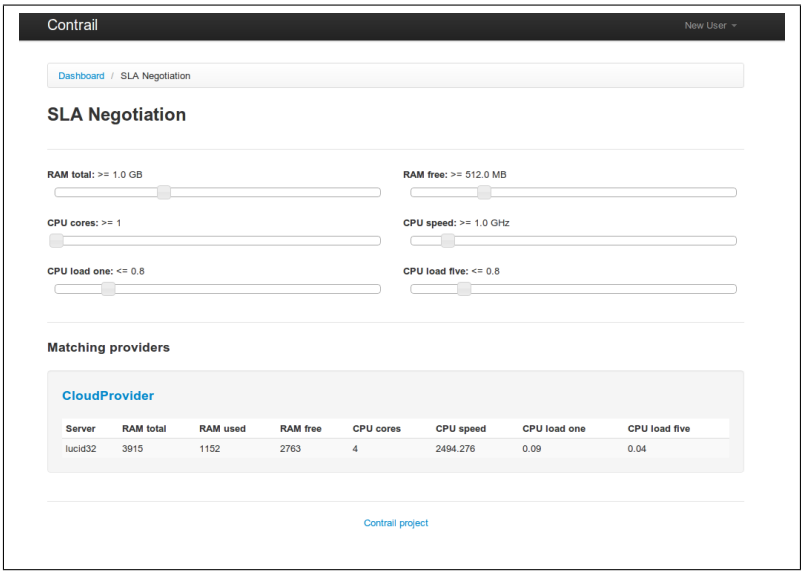


Figure 34: User must first negotiate a SLA. To do that, he must find a provider, that qualifies.

On the form **SLA Negotiation** user can choose among SLA templates that are registered on the chosen provider. When user click on one of the SLA Template, the SLA textbox is filled with the SLA content. When she clicks **Negotiate** SLA is sent to negotiation. In order to create an application based on this new SLA, user clicks on the **Register application** button on the form from figure 33.

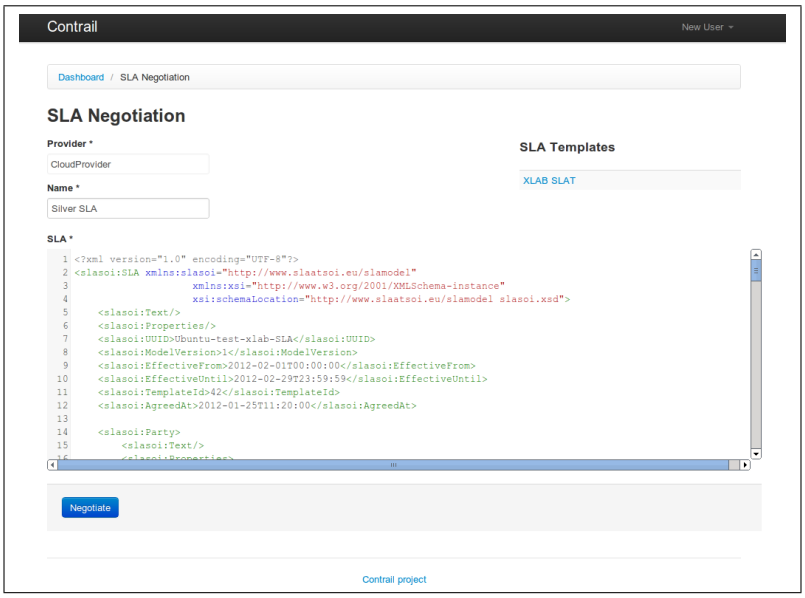


Figure 35: When the user chooses provider, he must enter the SLA name and SLA content. The user can use an existing SLA template if the provider adds one.

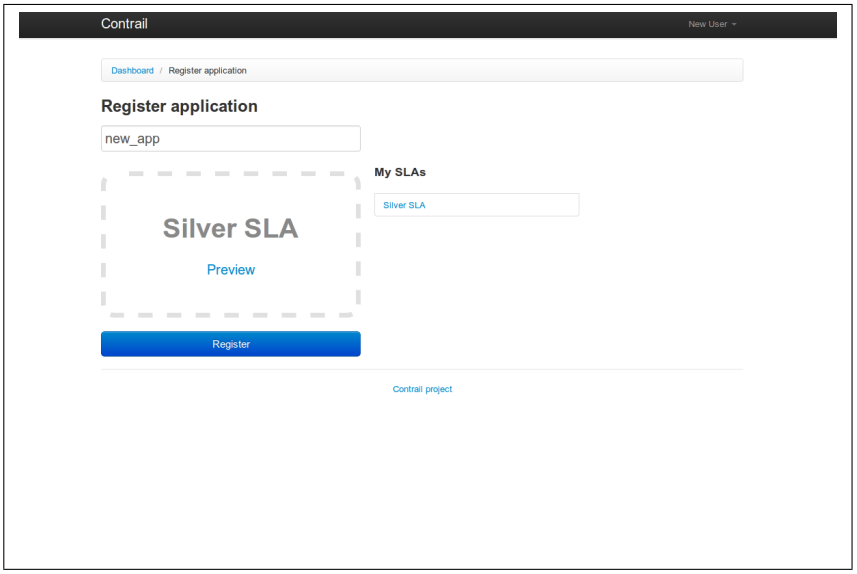


Figure 36: When the SLA negotiation is done, the user can register an application. The application must have a name and a SLA. User must drag an existing SLA to target

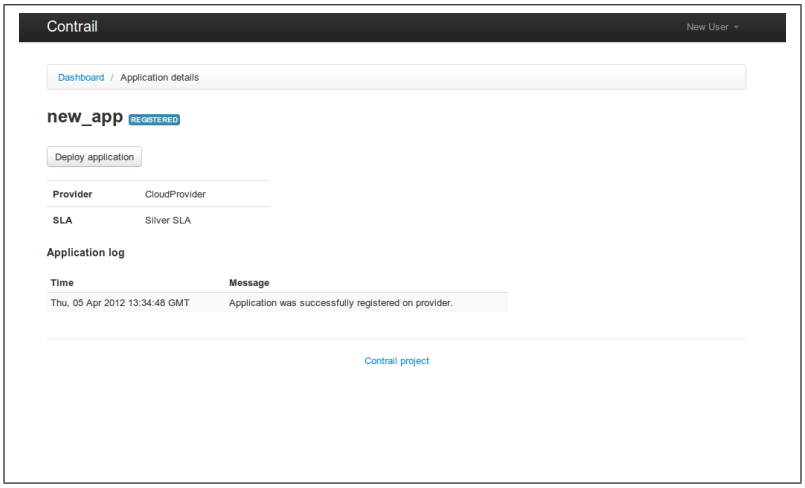


Figure 37: The user can see application's details when the application is registered. User can deploy the application using "Deploy application" button

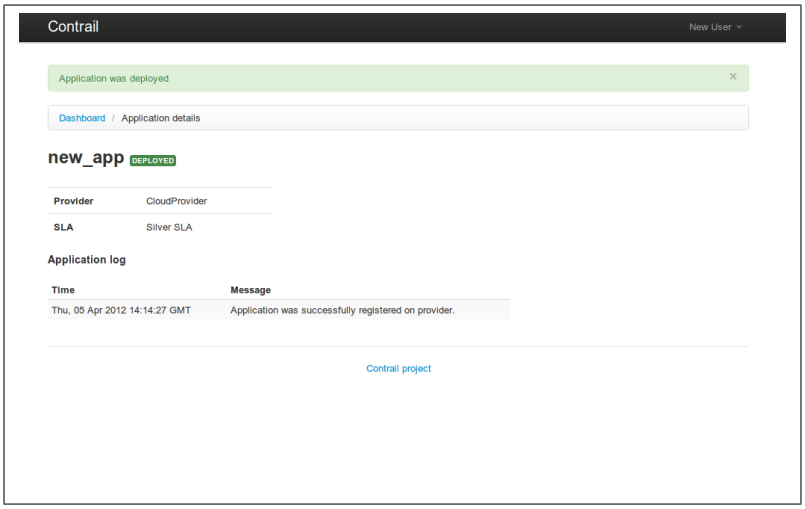


Figure 38: User will be notified when the application is deployed

Conclusion

In this document, we've given an overview of the Contrail release process, including release cycles, package management, as well as a summary of how to get started with Contrail, where to find documentation and how to get support.

The Contrail development platform we set up allows developers to share software artifacts efficiently and enforces some quality rules on source files. It also provides developers with a quick feedback about the quality of their contribution by rebuilding the entire software frequently and sending notifications when the build has been broken. We've also made the release procedure simple enough so that it is possible to release often, with short development cycles.

At the time this document is written, we already successfully published two release candidate versions[18] of Contrail and we are confident that the official release 1.0 will be delivered on time and that binary packages will meet minimal quality criteria.

Next upcoming challenges are to promote a better use of agile tools among developers, to provide Contrail integrated with more Linux distributions, and to maintain a fast and coherent build environment while the source code base is growing.

[18] 1.0~RC1 and 1.0~RC2

References

[1] Apache. Subversion.
http://subversion.apache.org/.

[2] Atlassian. Bamboo.
http://www.atlassian.com/software/bamboo/.

[3] Atlassian. Jira.
http://www.atlassian.com/software/jira/.

[4] SVN Book. Hook scripts.
http://svnbook.red-bean.com/en/1.0/svn-book.html#svn-ch-5-sect-2.1.

[5] Freenode. Discussion facilities for the free and open source software.
http://freenode.net/.

[6] Ohloh. The contrail project.
https://www.ohloh.net/p/contrail.

[7] OpenSUSE. Configuring debian with open build service.
http://en.opensuse.org/openSUSE:Build_Service_Debian_builds#Configuring_sources.list.

[8] OpenSUSE. Open build service.
https://build.opensuse.org/.

[9] OW2. Atlassian jira for the contrail project.
https://jira.ow2.org/browse/CONTRAIL.

[10] OW2. Bamboo page of the contrail project.
http://bamboo.ow2.org/browse/CONTRAIL-TRUNK.

[11] OW2. Contrail.
http://www.ow2.org/view/ActivitiesDashboard/Contrail.

[12] OW2. Contrail artifacts in nexus.
http://repository.ow2.org/nexus/index.html#nexus-search;quick~contrail.

[13] OW2. Contrail binaries repository.
http://contrail.ow2.org/repositories/binaries/.

[14] OW2. Contrail commit diffs (remove in url).
http://websvn.ow2.org/log.php?repname=contrail&path=\%2F&isdir=1&max=40&.

[15] OW2. Contrail documentation on the wiki.
http://contrail.projects.ow2.org/xwiki/bin/view/Documentation/WebHome.

[16] OW2. Contrail third party repository.
http://contrail.ow2.org/repositories/thirdparty/.

[17] OW2. Contrail wiki.
http://contrail.ow2.org/.

[18] OW2. Contrail wiki / development space.
http://contrail.projects.ow2.org/xwiki/bin/view/Development/WebHome.

[19] OW2. Download page of the contrail project.
http://contrail.projects.ow2.org/xwiki/bin/view/Main/Download.

[20] OW2. Getting support in the contrail project.
http://contrail.projects.ow2.org/xwiki/bin/view/Development/Community+Support.

[21] OW2. IRC channel of the contrail project.
http://contrail.projects.ow2.org/xwiki/bin/view/Development/IRC\#HIRC.

[22] OW2. Mailing lists of the contrail project.
http://contrail.projects.ow2.org/xwiki/bin/view/Development/Mailing+Lists.

[23] OW2. The open source community for infrastructure software.
http://www.ow2.org/.

[24] OW2. WebSVN for the contrail project.
http://websvn.ow2.org/listing.php?repname=contrail.

[25] OW2 GForge. Forge of the contrail project.
https://forge.ow2.org/projects/contrail/.

[26] OW2 GForge. Svn repository for the contrail project.
http://forge.ow2.org/plugins/scmsvn/index.php?group_id=396.

[27] Open Build Service. Staging repository of the contrail project.
https://build.opensuse.org/project/monitor?project=home\%3Acontrail\%3Astaging.

[28] Open Build Service. Testing repository of the contrail project.
https://build.opensuse.org/project/monitor?project=home\%3Acontrail\%3Atesting.

[29] Sonatype. Nexus.
http://www.sonatype.org/nexus/.

[30] Sympa. Sympa mailing list server.
http://www.sympa.org/index.

A Release procedure

This section provides the detailed procedure to release Contrail, a graphical representation of the process available in figure 10.

A.1 Prerequisites

Before starting to release, you will need to retrieve some parameters as well as release scripts.

A.1.1 Release parameters

The following parameters must be prepared before starting the release procedure:

Build id It is the number of a successful bamboo build (e.g. 802), which can be retrieved from the Bamboo webpage of the Contrail project[10].

Version string It is the name of the version to release (e.g. 1.0~RC3 or 1.0).

News message The message that will be sent in the projects RSS feed and also in the OW2 newsletter.

Figure 39 shows an example of news submitted with OW2 GForge for the release 1.0 RC1.

Figure 39: Example of news submitted with OW2 GForge

A.1.2 Release scripts

You will also need the administration shell scripts, which are included in B.1 and B.2. The following listing shows how to prepare the environment:

```
$ cd /tmp
$ svn checkout svn://svn.forge.objectweb.org/svnroot/contrail/
    trunk/common/ow2-admin-scripts
```

```
A    ow2-admin-scripts/validation_rsa
A    ow2-admin-scripts/contrail-packages.sh
A    ow2-admin-scripts/validation_rsa.pub
A    ow2-admin-scripts/contrail-repositories.sh
A    ow2-admin-scripts/pom.xml
A    ow2-admin-scripts/contrail-validation.sh
A    ow2-admin-scripts/Makefile
Checked out revision 2436.
$ cd ow2-admin-scripts/
$ make install DESTDIR=~/bin
bash -n contrail-packages.sh
cp contrail-packages.sh contrail-packages
bash -n contrail-repositories.sh
cp contrail-repositories.sh contrail-repositories
bash -n contrail-validation.sh
cp contrail-validation.sh contrail-validation
mkdir -p /home/sandre/bin/
install -m 0755 contrail-packages contrail-repositories contrail-
    validation /home/sandre/bin/
$ export PATH="$HOME/bin:$PATH"
```

A.2 Prepare

This part of the release process aims at preparing the release in the "staging" repository so that it can be intensively tested. In the following exemple we use bamboo build id 802 and version 1.0~RC1, you should change these values for your particular case.

A.2.1 Release packages in the "staging" repository

```
$ contrail-packages copy testing staging
...
$ contrail-packages --build-id 802 --version 1.0~RC1 update
    staging
...
```

Then you'll have to periodically check out the monitoring webpage of the staging repository[27] until *all* packages are built and available in the OBS repositories. An example of this monitoring page is shown on figure 3.

A.2.2 Synchronize the "staging" repository mirror

For this step, you need to connect to the OW2 shell server with admin rights on the Contrail project:

```
$ ssh svn.forge.objectweb.org
$
```

Then repeat procedure A.1.2 to install shell scripts if needed, and mirror the "staging" repository:

```
$ contrail-repositories mirror staging
...
```

Connect to OW2 GForge[25] with admin rights on the Contrail project and click on the "Push" button in teh "Admin" panel as shown on figure 40.

Figure 40: Button to publish files in OW2 GForge

A.2.3 Run validation tests

At this point, packages are ready for integration tests and also for validation tests. You can run validation tests in the "staging" repository like this:

```
$ contrail-validation -r staging run
...
```

A.3 Perform

This part of the release process aims at publishing the tested packages of the staging repository.

A.3.1 Copy packages to the "release" repository

Execute the following command to copy packages from the staging repository to the release repository:

```
$ contrail-packages copy staging release
...
```

A.3.2 Synchronize the "release" repository mirror

For this step, you need to connect to the OW2 shell server with admin rights on the Contrail project:

```
$ ssh svn.forge.objectweb.org
$
```

Then repeat procedure A.1.2 to install shell scripts if needed, and mirror the "release" repository:

```
$ contrail-repositories mirror release
...
```

Connect to OW2 GForge[25] with admin rights on the Contrail project and click on the "Push" button in teh "Admin" panel as shown on figure 40.

A.3.3 Run validation tests

At this point, packages are ready for users and also for validation tests. You can run validation tests in the "release" repository like this:

```
$ contrail-validation -r release run
...
```

A.4 Finalize

A.4.1 Update Jira versions

Connect to jira[9] and mark the version you just released as "released". Figure 41 shows a screenshot of this webpage.

A.4.2 Tag the source code

This step saves a copy of the source code at the time of the release in the "tags" directory of the source code repository. In the following example, we tag source code revision 2535 which corresponds to release version 1.0~RC2.

```
$ contrail-repositories tag 2535 1.0~RC2
...
```

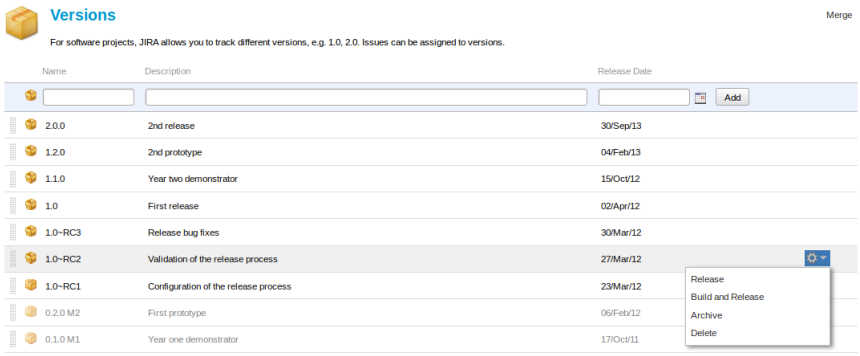


Figure 41: Managing Contrail versions in Jira

A.4.3 Notify the community

Send the news message Connect to OW2 GForge[25] with admin rights on the Contrail project and submit the news message you prepared. Figure 39 shows an example of the news submitted with OW2 GForge for the release 1.0 RC1.

Email the community Send an email to `contrail-dev@ow2.org` and `contrail@ow2.org` with the same message you just submitted in the news.

B Shell scripts

B.1 Contrail packages manager

```
#!/bin/bash
#
#    Manage Contrail binary packages
#
# Usage
# -----
#    contrail-packages [options] <command> [<args>...]
#
# Versions
# --------
#    2012-03-27 Sebastien Andre <sebastien.andre@petalslink.com>
#        * Copy only project packages, not links or aggregates
#        * Improved usage message with updated examples
#    2012-03-23 Sebastien Andre <sebastien.andre@petalslink.com>
#        * Implemented the copy command
#        * Simplified command line interface
#    2012-03-22 Sebastien Andre <sebastien.andre@petalslink.com>
#        * Skip linked and aggregated packages when updating
#        * Changed interactive shell prompt
#    2012-03-06 Sebastien Andre <sebastien.andre@petalslink.com>
#        * Added DATADIR in variables set in debian.rules
#    2012-03-05 Sebastien Andre <sebastien.andre@petalslink.com>
#        * Now set variables PACKAGE, LIBDIR, BINDIR in debian.
    rules
#        * Improved the interactive shell interface
#    2012-03-01 Sebastien Andre <sebastien.andre@petalslink.com>
#        * Added notice message when starting the interactive
    subshell
#        * Fixed problem processing non versioned osc packages
#    2012-02-29 Sebastien Andre <sebastien.andre@petalslink.com>
#        * Replaced option --revision (SVN) by --build-id (Bamboo)
#        * Replaced option --user by --maintainer
#        * Changed packages version incrementation method
#    2012-02-28 Sebastien Andre <sebastien.andre@petalslink.com>
#        * Added --user, --email and --interactive options
#        * Implemented automatic update of debian configuration
    files
#    2012-02-27 Sebastien Andre <sebastien.andre@petalslink.com>
#        * Initial version of update-contrail-packages
#
set -o nounset -o pipefail -o errexit


declare -r NAME='contrail-packages'
declare -r VERSION='2012-03-22'


declare -r SVN_URL='svn://svn.forge.objectweb.org/svnroot/contrail
    '
```

```
46 declare -r BAMBOO_URL='http://bamboo.ow2.org/browse/CONTRAIL-TRUNK
     -%ID%/artifact/JOB1/Packages'
   declare -r BAMBOO_RSS='http://bamboo.ow2.org/rss/
     createAllBuildsRssFeed.action?feedType=rssAll&buildKey=
     CONTRAIL-TRUNK&os_authType=basic'
48 declare -r OBS_PRJ='home:contrail'
   declare -r CACHE_DIR="$(echo ~)/.cache/$NAME"
50
   # TODO: convert to local variables
52 declare MAINTAINER='Contrail Administrators' EMAIL='contrail-
     admin@ow2.org'

54 declare -r USAGE="Usage: $NAME [options] <command> [<args>...]

56 Options:

58    -v, --version STR
          Set the version string of the packages
60    -b, --build-id NUMBER
          Select Bamboo build NUMBER
62    -c, --cleanup
          Empty the local cache directory before starting.
64    -m, --maintainer NAME
          Set the maintainer's name (default ${MAINTAINER})
66    -e, --email EMAIL
          Set the maintainer's email (default ${EMAIL})
68    -i, --interactive
          Allow manual modification of files before commit
70

72 Commands:

74    update <subproject> [<package>...]
          Rebuild every <package> (default all) from
76        <subproject> for the latest successful Bamboo build.

78    copy <source> <target> [<package>...]
          Copy <package> (default all) from <source> subproject
80        to <target>.


82
   Examples:
84
      # running nightly builds
86    $NAME update testing

88    # staging release 1.0~RC2
      $NAME copy testing staging
90    $NAME -b 840 -v 1.0~RC2 update staging

92    # publishing a staged release
      $NAME copy staging release
94
      # staging only bug fixes of package conpaas-web 1.0.0
```

```
    $NAME -v 1.0.1 -b 844 update staging conpaas-web
"



# Print an info message
function info () {
  echo ":: $(date '+%F %T') [$NAME] $*" >&2
}

# Print a warning message
function warn () {
  info "WARNING: $*" >&2
}

# Terminate the script with an error message
function fail () {
  info "ERROR: ${1:-failed}" >&2
  exit ${2:-1}
}


# Overwrite a file with content read from STDIN for inplace
    updates
function ow () {
  local f="${1?}"
  info "$f: rewrite file"
  cat > "$f.tmp" && mv -f "$f.tmp" "$f"
}


# Print the Bamboo build id of the latest successful build if
# $1 empty, print $1 otherwise
function print_build_id () {
  info "retrieve bamboo build id"
  local bid="${1:-$((
    wget -nv -O - --post-data "${BAMBOO_RSS#*\?}" "${BAMBOO_RSS
        %%\?*}" \
      | tr '>' '\n' \
      | awk '
        sub(/^CONTRAIL-TRUNK-/, "") && sub(/ was SUCCESSFUL.*/, ""
            ) {
        print
        exit
      }'
    # TODO: fix broken pipe error
  ) 2>/dev/null)}"
  echo "$bid"
  test ! -z "$bid"
}

# Print and check the Bamboo URL of artifacts for build id $1
function print_build_url () {
  local bid="${1?}"
```

```
  local url="$(sed -e "s/%ID%/$bid/g" <<<"$BAMBOO_URL")"
  if wget --spider "$url" >&2
  then
    info "build id #$bid has been found on Bamboo"
    echo "$url"
  else
    info "build id #$bid was NOT found on Bamboo"
    return 1
  fi
}


# Print cache directory for package $2 in subproject $1
function print_cache_dir () {
  local dir="$CACHE_DIR/$OBS_PRJ${1:+:$1}${2:+/$2}"
  echo "$dir"
}

# Update or checkout OBS files for subproject $1
# and print the resulting directory path
function update_cache () {
  local sub="$OBS_PRJ:${1?}"
  local dir="$CACHE_DIR/$sub"

  mkdir -vp "$CACHE_DIR"
  if [ -d "$dir" ]
  then
    info "update cache for repository \`$sub'"
    ( cd "$dir" && osc update -u )
  else
    info "checkout repository \`$sub' in local cache"
    ( cd "$CACHE_DIR" && osc checkout -u "$sub" )
  fi
}

# List packages in cache for subproject $1
function list_cache () {
  local sub="${1?}"
  ( cd "$CACHE_DIR/$OBS_PRJ:$sub" && echo * )
}


# Tests whether package $2 from subproject $1 is a project package
# and not a link, an aggregate or a simple directory
function is_project_package () {
  local dir="$(print_cache_dir "${1?}" "${2?}")" pkg="$2"

  if [ ! -d "$dir" ]
  then
    warn "$pkg: package not found in cache"

  elif [ ! -d "$dir/.osc" ]
  then
    warn "$pkg: package is not under version control"
```

```
202   elif [ -f "$dir/_link" ]
      then
204     info "$pkg: $(sed '1!d' < "$dir/_link")"

206   elif [ -f "$dir/_aggregate" ]
      then
208     info "$pkg: $(sed '2!d' "$dir/_aggregate")"

210   else
        return 0
212   fi

214   return 1
    }
216


218


220
    # Run an interactive shell in directory $1
222 function run_interactive_shell () {
      local dir="${1:-.}"
224
      echo '
226 :::::::::::::::::: Interactive Shell ::::::::::::::::::::
    :: Type "exit" when all modifications are done.       ::
228 :: To discard all changes, exit with a non-zero code. ::
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::'
230   ( cd "$dir" && PS1="${PWD##*/}> " sh )
    }
232


234 # Ask confirmation for message $1
    # Return 0 for 'y', 1 for 'n' and loop otherwise
236 function ask_configuration () {
      local prompt="$1"
238
      while true
240   do
        read -p "$prompt? [Y/n] "
242     case "${REPLY:-y}" in
          y|Y) return 0 ;;
244       n|N) return 1 ;;
          *) echo 'Please answer "y" for yes or "n" for no'
246     esac
      done
248 }

250 # Update debian dsc for package $1 and version $2
    function update_debian_dsc () {
252   local pkg="${1?}" ver="${2?}"

254   info "$pkg: update debian descriptor"
```

```
  echo "Format: 1.0
Source: $pkg
Version: $ver
Binary: $pkg
Maintainer: $MAINTAINER <$EMAIL>
Architecture: any
Build-Depends: $(awk '
  /^Files:/ { exit }
  sw { print }
  sub(/^Build-Depends: */, "") { if ($0) print; sw=1 }
' $pkg.dsc)
Files:
 $(
  md5sum $pkg.tar.gz | awk '{ print $1 }'
) $(
  stat -c %s $pkg.tar.gz
) ${pkg}_${ver}.orig.tar.gz" \
    | ow "$pkg.dsc"
}

# Print the next version string for build id $1 and version $2
function print_next_version () {
  local bid="${1?}" ver="$2"
  echo "${ver:-0}+$bid"
}


# Update debian rules for package $1 and build url $2
function update_debian_rules () {
  local pkg="${1?}" url="${2?}"

  info "$pkg: update debian rules file"
  sed -i "
    s|^\(UPSTREAM_URL=\).*|\1$url|
    s|^\(PACKAGE=\).*|\1$pkg|
    s|^\(LIBDIR=\).*|\1/usr/lib/contrail/$pkg|
    s|^\(DATADIR=\).*|\1/usr/share/contrail/$pkg|
    s|^\(ETCDIR=\).*|\1/etc/contrail/$pkg|
    s|^\(BINDIR=\).*|\1/usr/bin|
  " "debian.rules"
}

# Update debian changelog for package $1, version $2 and build id
    $3
function update_debian_changelog () {
  local pkg="${1?}" ver="${2?}" bid="${3?}"

  info "$pkg: update debian changelog file"
  echo "$pkg ($ver) unstable; urgency=low

  * updated with artifacts from bamboo build #$bid

 -- $MAINTAINER <$EMAIL>  $(date -R)
" \
```

```
308     | ow "debian.changelog"
}

# Update debian control file for package $1 and version $2
function update_debian_control () {
  local pkg="${1?}" ver="${2?}"

  info "$pkg: update debian control file"
  sed -i -e "
    s/^Source:.*/Source: $pkg/
    s/^Maintainer:.*/Maintainer: $MAINTAINER <$EMAIL>/
    s/^Version:.*/Version: $ver/
  " "debian.control"
}


# Update debian package $2 in subproject $1 for build id $3, new
    version $4
# and build url $5. All files are expected to be in the current
    directory
function update_debian_package () {
  local sub="${1?}" pkg="${2?}" bid="${3?}" ver="$4" url="${5?}"

  if is_project_package "$sub" "$pkg"
  then
    update_debian_changelog "$pkg" "$ver" "$bid"
    update_debian_rules "$pkg" "$url"

    info "$pkg: get upstream source"
    make -f debian.rules get-orig-source

    update_debian_dsc "$pkg" "$ver"
    update_debian_control "$pkg" "$ver"
  else
    info "$pkg: skipped"
  fi
}




#
    ################################################################################




if [ $# -eq 0 ]
then
  echo "$USAGE" >&2
  exit 2
fi
```

```
358
declare command= opts= build_id= version=
360 declare -i interactive=0 clean=0

362 opts="$(POSIXLY_CORRECT='' getopt --options "b:v:cm:e:i" \
    --longoptions "build-id:,version:,clean,maintainer:,email:,
        interactive" \
364     --name "$NAME" --shell 'bash' -- "$@")" \
  || exit 2
366
eval set -- "$opts"
368
while true
370 do
  case "$1" in
372   -b|--build-id)
      egrep -q '^[0-9]+$' <<<"$2" \
374       || fail "$2: invalid bamboo build id"
      build_id="$2"
376     shift 2
    ;;
378   -v|--version)
      version="$2"
380     egrep -q '^[0-9A-Z\.\+\~\-]+' <<<"$2" \
        || fail "$2: invalid version number"
382     shift 2
    ;;
384   -c|--clean)
      clean=1
386     shift
    ;;
388   -i|--interactive)
      interactive=1
390     shift
    ;;
392   -m|--maintainer)
      MAINTAINER="$2"
394     shift 2
    ;;
396   -e|--email)
      email="$2"
398     shift 2
    ;;
400   --)
      shift
402     break
    ;;
404   *)
      fail "internal error" 127
406   esac
done
408

410 declare command="$1"
```

```
shift

info "start $command"

case "$command" in
update)
  declare pkg subprj build_url
  declare -a pkgs

  if [ $# -eq 0 ]
  then
    fail "$#: bad number of arguments"

  elif ! osc list "$OBS_PRJ:$1" &> /dev/null
  then
    fail "$1: subproject not found"

  else
    subprj="$1"
    shift
  fi

  if ! build_id="$(print_build_id "$build_id")"
  then
    fail "failed to retrieve bamboo build id"

  elif ! build_url="$(print_build_url "$build_id")"
  then
    fail "failed to reach build artifacts on Bamboo"
  fi

  if (( clean ))
  then
    info "$(print_cache_dir "$subprj"): clean cache directory"
    rm -fr "$(print_cache_dir "$subprj")"
  fi

  update_cache "$subprj" \
    || fail "$subprj: failed to update local cache"

  if [ $# -eq 0 ]
  then
    pkgs=( $(osc list "$OBS_PRJ:$subprj") )
  else
    pkgs=( "$@" )
  fi

  info "print Open Build Service packages list"
  tr ' ' '\n' <<<"${pkgs[*]}" | nl

  version="${version:-0}+$build_id"
  info "update to version $version"

  for pkg in "${pkgs[@]}"
```

```
  do
    pushd .
    cd "$(print_cache_dir "$subprj" "$pkg")"

    update_debian_package "$subprj" "$pkg" "$build_id" "$version"
        "$build_url"
    osc status

    if (( ! interactive ))
    then
      osc commit -m "updated by $NAME for version $version"

    elif run_interactive_shell . \
      || ask_confirmation "Discard these changes"
    then
      osc commit

    else
      warn "$pkg: changes in cache have NOT been saved"
    fi

    popd
  done
;;
copy)
  declare source target pkg
  declare -a pkgs

  if [ $# -lt 2 ]
  then
    fail "$#: bad number of arguments"

  elif ! osc list "$OBS_PRJ:$1" &> /dev/null
  then
    fail "$1: source subproject not found"

  elif ! osc list "$OBS_PRJ:$2" &> /dev/null
  then
    fail "$2: target subproject not found"

  else
    source="$1"
    target="$2"
    shift 2
  fi

  if (( clean ))
  then
    info "clean cache directories"
    rm -fr "$(print_cache_dir "$source")" "$(print_cache_dir "
        $target")"
  fi

  if ! update_cache "$source" || ! update_cache "$target"
```

```
  then
    fail "failed to update local cache"
  fi

  if [ $# -eq 0 ]
  then
    pkgs=( $(osc list "$OBS_PRJ:$source") )
  else
    pkgs=( "$@" )
  fi

  for pkg in "${pkgs[@]}"
  do
    if is_project_package "$source" "$pkg"
    then
      info "$pkg: copy package from $source to $target"
      osc copypac "$OBS_PRJ:$source" "$pkg" "$OBS_PRJ:$target"
    else
      info "$pkg: skipped"
    fi
  done
;;
*)
  fail "$command: unknown command"
esac

info 'terminate'
exit 0
```

Listing 1: Script contrail-packages.sh

B.2 Contrail repositories manager

```
#!/bin/bash
#
#    Manage artifacts repositories for the Contrail project
#
# Usage
# -----
#    contrail-repositories <source_tar> <target_dir>
#
# Versions
# --------
#    2012-03-28 Sebastien Andre <sebastien.andre@gmx.fr>
#        * Added the tag command
#    2012-03-27 Sebastien Andre <sebastien.andre@gmx.fr>
#        * Fixed a bug with extra / in the URL
#    2012-03-26 Sebastien Andre <sebastien.andre@gmx.fr>
#        * Mirror only one repository instead of everything
#        * Do not use /tmp anymore to store downloaded files
#    2012-03-22 Sebastien Andre <sebastien.andre@gmx.fr>
#        * Simplified command line usage
```

```
#     2012-02-27 Sebastien Andre <sebastien.andre@gmx.fr>
#         * Now print started and finished time
#         * Added usage message
#     2012-02-21 Sebastien Andre <sebastien.andre@gmx.fr>
#         * Added code to mirror repositories
#     2012-02-08 Sebastien Andre <sebastien.andre@gmx.fr>
#         * Initial version
#
set -o nounset -o errexit

declare -r NAME='contrail-repositories'
declare -r VERSION='2012-03-26'

declare -r OW2_DIR='/var/lib/gforge/chroot/home/groups/contrail/
    htdocs/repositories/binaries'
declare -r OBS_URL='http://download.opensuse.org/repositories/home
    :/contrail:'
# TODO: fix double slashes // in url who break the --cut-dirs
    option in wget


declare -r USAGE="Usage: $NAME <command> [args...]

    create <source> <target>
        Convert a <source> directory with Maven 3rd party files to
            a Maven
        repository in <target> directory. All files in <source>
            must match
        '<groupId>/<ArtifactId>-<version>.<packaging>'.

    mirror <repo> [<max rate>]
        Create a mirror copy of the OBS subproject <repo>.
        The transfert rate can be limited by <max rate>.

    tag <revision> <version> [<login>]
        Create a tag copy of source files at <revision> for <
            version>.
        If <login> is provided, it is used to connect to SVN.

"


# Print error message $1 and exit with status $2
function fail () {
  echo "ERROR: ${1:-failed}" >&2
  exit ${2:-1}
}

function info () {
  echo ":: $(date '+%F %T') [$NAME] $*"
}

# Convert files directory $1 to maven repository $2
function create_maven_repo () {
```

```
68    local src="${1?}" dst="${2?}"
      local groupId artifactId version packaging dir file base
70
      for dir in "$src"/*
72    do
        groupId="${dir##*/}"
74      for file in "$dir"/*
        do
76        base="${file##*/}"
          packaging="${base##*.}"
78        artifactId="${base%-*}"
          version="${base#$artifactId-}"
80        version="${version%.$packaging}"

82        info "install $groupId:$artifactId:$version:$packaging"
          mvn install:install-file -DlocalRepositoryPath="$Target" \
84          -Dfile="$dir/$base" -DgroupId="$groupId" \
            -DartifactId="$artifactId" -Dversion="$version" \
86          -Dpackaging="$packaging" \
            -DgeneratePom=true -DcreateChecksum=true
88      done
    done
90  }

92  # Create a mirror copy of URL $1 in directory $2 with optional
        rate limit $3
    function mirror_binaries_repo () {
94    local url="${1?}" rate="${3:-800k}"
      local newdir="${2?}" bkpdir="${2%%/}.bkp" tmpdir="${2%%/}.tmp"
96
      info "$url: testing if URL is online"
98    wget --spider --quiet "$url"

100   info "$tmpdir: create temporary directory"
      rm -fr "$tmpdir"
102   mkdir -p "$tmpdir"

104   info "$url: download files from repository to backup dir"
      wget --no-verbose --recursive --no-timestamp --no-cache --
          execute 'robots=off' \
106     "--limit-rate=$rate" --no-parent --no-host-directories --cut-
            dirs=4 \
              --reject '.html,.mirrorlist,.torrent,.btih,.meta4,.
                  metalink' \
108           --directory-prefix "$tmpdir" -- "$url"

110   info "remove unwanted files"
      find "$tmpdir" -type f -name 'index.html*' -delete
112   rm -f "$tmpdir/$(basename "$newdir")"

114   info "fix group and permission of files"
      chgrp -R contrail "$tmpdir"
116   find "$tmpdir" -type d -exec chmod 2775 {} +
      find "$tmpdir" -type f -exec chmod 2664 {} +
```

```
118
    info "$newdir: setup new directory"
120   if [ -e "$newdir" ]
    then
122     rm -fr "$bkpdir"
      mv "$newdir" "$bkpdir"
124     mv "$tmpdir" "$newdir"
      rm -fr "$bkpdir"
126   else
      mv "$tmpdir" "$newdir"
128   fi
}
130


132


134 if [ $# -eq 0 ]
then
136   echo "$USAGE" >&2
    exit 2
138 fi

140 info "$@"

142 declare command="$1"
shift
144
case "$command" in
146 create)
    if [ $# -ne 2 ]
148   then
      fail "$#: bad number of arguments"
150
    elif [ ! -d "$1" ]
152   then
      fail "$1: source directory not found"
154
    elif [ -e "$2" ]
156   then
      fail "$2: target directory already exists"
158
    elif ! create_maven_repo "$1" "$2"
160   then
      fail
162   fi
;;
164 mirror)
    declare url dir rate
166
    if [ $# -lt 1 -o $# -gt 2 ]
168   then
      fail "$#: bad number of arguments"
170
    elif [ $# -ge 2 ] && egrep -vqw '[0-9\.]+[km]?' <<<"$2"
```

```
172   then
        fail "$2: invalid download rate limit"
174
      elif [ ! -d "$OW2_DIR" -o ! -w "$OW2_DIR" ]
176   then
        fail "$OW2_DIR: unable to write in directory"
178
      else
180     url="$OBS_URL/$1"
        dir="$OW2_DIR/$1"
182     rate="${2:-1m}"
      fi
184
      mirror_binaries_repo "$url" "$dir" "$rate"
186 ;;
    tag)
188   declare rev ver dev

190   if [ $# -lt 2 -o $# -gt 3 ]
      then
192     fail "$#: bad number of arguments"

194   elif ! egrep -qw '[0-9]+' <<<"$1"
      then
196     fail "$1: invalid SVN revision number"

198   elif [ -z "$2" ]
      then
200     fail "version name cannot be empty"

202   else
        rev="$1"
204     ver="$2"
        dev="${3:-$USER}"
206   fi

208   # TODO: deep copy of externals content
      if svn copy --ignore-externals --revision "$rev" \
210     --message "[contrail-repositories] tag revision $rev for
            version $ver" \
        "svn+ssh://${dev}@svn.forge.objectweb.org/svnroot/contrail/
            trunk" \
212     "svn+ssh://${dev}@svn.forge.objectweb.org/svnroot/contrail/
            tags/contrail-trunk-${ver}"
      then
214     info "created tag contrail-trunk-$ver for revision $rev"
      else
216     fail "failed to tag SVN revision $rev"
      fi
218 ;;
    *)
220   fail "$command: invalid command"
    esac
222
```

```
224 info 'terminate'
    exit 0
```

Listing 2: Script contrail-repositories.sh

B.3 Contrail validation tests

```
    #!/bin/bash
2   #
    #    Manage validation tests for the Contrail project
4   #
    #    This script uses libvirt along with a set of preconfigured
6   #    virtual machines templates to install/uninstall all binary
    #    packages automatically and independently. It aims at
        improving
8   #    the softwares quality by detecting packaging and
        configuration
    #    issues before publication.
10  #
    # Usage
12  # -----
    #    contrail-validation [options] <command> [<args>...]
14  #
    # Versions
16  # ----------
    #    2012-03-30 Sebastien Andre <sebastien.andre@gmx.fr>
18  #        * Improved the os and domains filter
    #        * Added custom command when starting/stopping VMs
20  #    2012-03-28 Sebastien Andre <sebastien.andre@gmx.fr>
    #        * Added new domains and IP of the ESX server
22  #        * Implemented the --oses option
    #    2012-03-26 Sebastien Andre <sebastien.andre@gmx.fr>
24  #        * Changed default directory for key files
    #        * Added staging repository
26  #    2012-03-22 Sebastien Andre <sebastien.andre@gmx.fr>
    #        * Added interactive terminal to virtual hosts
28  #    2012-03-20 Sebastien Andre <sebastien.andre@gmx.fr>
    #        * Added start/stop/copy-id commands
30  #        * Added create/revert snapshots during tests
    #        * Added test script to install/remove/install/purge
        packages
32  #    2012-03-19 Sebastien Andre <sebastien.andre@gmx.fr>
    #        * Initial version
34  #
    set -o pipefail +o errexit
36
    declare -r NAME='contrail-validation'
38  declare -r VERSION='2012-03-28'


40
    # URL of the binaries repository
```

```
declare REPO_URL='http://contrail.ow2.org/repositories'

# Max time in seconds for a VM to start
declare -i STARTUP_TIMEOUT=50

# Identity File to connect cluster and VMs
declare KEY_FILE="./validation_rsa"

# IP address of domain 0
declare DOM0_IP='192.168.1.23'

# Repositories per OBS subproject
declare -A REPO_OS=(
  [testing]='Debian_6.0 xUbuntu_11.04 xUbuntu_11.10'
  [staging]='Debian_6.0 xUbuntu_11.04 xUbuntu_11.10'
  [release]='Debian_6.0 xUbuntu_11.04 xUbuntu_11.10'
)

# Compatible domains per os
declare -A OS_DOMS=(
  [Debian_6.0]='debian-6-64 debian-6-32'
  [xUbuntu_11.04]='ubuntu-11-04-32 ubuntu-11-04-64'
  [xUbuntu_11.10]='ubuntu-11-10-32'
)

# Ip address of domains
declare -A DOM_IPS=(
  [debian-6-32]='192.168.1.200'
  [debian-6-64]='192.168.1.201'
  [ubuntu-11-04-32]='192.168.1.202'
  [ubuntu-11-04-64]='192.168.1.203'
  [ubuntu-11-10-32]='192.168.1.204'
)


# Default list of repositories
declare -a REPOS=(
  "${!REPO_OS[@]}"
)

# Default list of OS
declare -a OSES=(
  "${!OS_DOMS[@]}"
)

# Default list of domains
declare -a DOMAINS=(
  "${!DOM_IPS[@]}"
)

# Default list of packages
declare -a PACKAGES=(
conpaas-frontend
conpaas-frontend-db
```

```
conpaas-mysql
conpaas-scalarix-one-client
conpaas-scalarix-one-frontend
conpaas-scalarix-one-manager
conpaas-taskfarm
conpaas-web
contrail-ca-server
contrail-federation-api
contrail-federation-cli
contrail-federation-db
contrail-federation-id-prov
contrail-monitoring-hub
contrail-one-monitor
contrail-one-sensor
contrail-provider-accounting
contrail-provisioning-manager
contrail-rest-monitoring
contrail-safeconfig
contrail-vep-cli
contrail-vep-gui
contrail-vin
scalaris
scalaris-doc
xtreemfs-backend
xtreemfs-client
xtreemfs-server
xtreemfs-tools
)


declare -r USAGE="$NAME [options] <command>

Options:
   -r, --repositories LIST
            Names of repositories of binary packages to be tested
                .
   -o, --oses LIST
            Names of the operating systems to be tested.
   -d, --domains LIST
            Names of libvirt domains (Virtual Machines).
   -p, --packages LIST
            Names of packages to be tests.

   -k, --key-file PATH
            File from which the user's DSA, ECDSA or DSA
            authentication identity is read. (default $KEY_FILE)

   -i, --interactive
            Run an intercative shell once tests have finished

Commands:
   copy-id    Copy Key ID file on every machine
   run        Install and uninstall every package in separated
       machines.
```

```
148     list        List selected domains and tasks.
        errors      Print errors from log files
150
        start       Start domains
152     stop        Stop domains

154 Manage validation tests for the Contrail project.

156 Default behavior is to test all packages from each repository on
        every
    domain. Using options can restrict tests to specific packages or
        repositories.
158
    Elements in LIST are all comma separated.
160


162 Examples:
        # Test install/uninstall of the contrail-vin nightly build on
            debian
164     $NAME -r testing -p contrail-vin -d debian6-64,debian6-32 run

166 "



168


170 # Terminate the script with an error message
    function fail () {
172   echo "ERROR: ${1:-failed}" >&2
      exit ${2:-1}
174 }

176 function warn () {
      echo "WARNING: $*" >&2
178 }

180 # Overwrite a file with content read from STDIN allowing inplace
        updates
    function ow () {
182   cat > "${1?}.tmp" && mv -f "$1.tmp" "$1"
    }
184
    # Print the intersection of $@ -- $@
186 function inter () {
      local -a stack=( ) result=( )
188
      while (( $# ))
190   do
        case "$1" in
192     --)
          break ;;
194     *)
          stack+=( "$1" )
196       shift
        esac
```

```
198   done
200   shift
202   while (( $# ))
      do
204     for item in "${stack[@]}"
        do
206       case "$1" in
          $item)
208         result+=( "$1" )
            shift
210         continue 2
          esac
212     done
        shift
214   done
216   echo "${result[*]}"
    }
218
220 # Print all selected combinations of (repository, os, domain,
        package)
    function print_tasks () {
222   local repo os dom pkg
      local -a repos oses doms
224
      for repo in "${REPOS[@]}"
226   do
        oses=( ${REPO_OS[$repo]} )
228
        if [ ${#oses[@]} -eq 0 ]
230     then
          warn "$repo: no OS compatible with this repository"
232       continue
        fi
234
        for os in $(inter "${oses[@]}" -- "${OSES[@]}")
236     do
          doms=( ${OS_DOMS[$os]} )
238
          if [ ${#doms[@]} -eq 0 ]
240                       then
                                warn "$os: no valid domain found
                                    for this operating system" >&2
242                             continue
                          fi
244
          for dom in $(inter "${doms[@]}" -- "${DOMAINS[@]}")
246       do
            if [ -z "${DOM_IPS[$dom]}" ]
248         then
              warn "$dom: no ip address configured for domain"
```

```
250             continue
            fi

            for pkg in "${PACKAGES[@]}"
            do
              echo "$repo|$os|$dom|$pkg"
            done
          done
        done
      done
}

# Print the test script for package $2 from repository $1
function print_test_script() {
  local url="${1?}" pkg="${2?}"
  echo "
echo "deb $url ./" >> /etc/apt/sources.list
wget -O - $REPO_URL/contrail.pub | apt-key add -
aptitude update
aptitude search contrail conpaas scalaris xtreemfs
aptitude -y install $pkg
aptitude -y remove $pkg
aptitude -y install $pkg
aptitude -y purge $pkg
"
}


# Run command $@ on dom0
function exec_dom0 () {
  ssh -i "$KEY_FILE" -l root "$DOM0_IP" "$@"
}

# Run command $@ on domain $1
function exec_dom () {
  local dom="${1?}"
  shift
  ssh -i "$KEY_FILE" -l root "${DOM_IPS[$dom]}" "$@"
}


# Wait until port $2 on machine $1 is in $3 state ("up" or "down")
# within $4 seconds (default 30)
function wait_updown () {
  local ip="${1?}" port="${2?}" state="${3?}"
  local -i threshold=$(( $(date +%s) + ${4:-${STARTUP_TIMEOUT}} ))
        ret=0

  case "$state" in
  up|down)
    while true
    do
      ( exec <> "/dev/tcp/$ip/$port" ) &> /dev/null
      ret=$?
```

```
304       if [ $ret -eq 0 -a "$state" = "up" ] \
            || [ $ret -ne 0 -a "$state" = "down" ]
306       then
            return 0
308
          elif (( $(date +%s) > threshold ))
310       then
            return 1
312
          else
314         sleep 1
          fi
316     done
    ;;
318 *)
      fail "$state: invalid state"
320 esac
}
322


324
#
    ##############################################################################

326 declare opts prog pkg name

328 opts="$(POSIXLY_CORRECT="" getopt --options "r:p:o:d:k:i" \
  --longoptions "key-file:,packages:,oses:,domains:,repositories:,
      interactive" \
330       --name "$NAME" --shell 'bash' -- "$@")" \
    || exit 2
332 eval set -- "$opts"

334 while true
do
336   case "$1" in
  -k|--key-file)
338     KEY_FILE="$2"
      shift 2
340   ;;
  -o|--oses)
342     OSES=( ${2//,/ } )
      shift 2
344   ;;
  -p|--packages)
346     PACKAGES=( ${2//,/ } )
      shift 2
348   ;;
  -d|--domains)
350     DOMAINS=( ${2//,/ } )
          shift 2
352   ;;
  -r|--repositories)
```

```
354       REPOS=( ${2//,/ } )
                shift 2
356     ;;
    -i|--interactive)
358       INTERACTIVE='yes'
          shift
360     ;;
    --)
362       shift
          break
364     ;;
    *)
366       fail "internal error" 127
    esac
368 done


370
    declare cmd repo os dom pkg
372
    if [ $# -eq 0 ]
374 then
    echo "$USAGE" >&2
376   exit 2

378 elif [ ! -f "$KEY_FILE" ]
    then
380   fail "$KEY_FILE: key file not found"

382 else
    cmd="$1"
384   shift
    fi
386



388
    case "$cmd" in
390 copy-id)
    for ip in "$DOM0_IP" "${DOM_IPS[@]}"
392 do
      ssh-copy-id -i "$KEY_FILE.pub" "root@$ip"
394 done
    ;;
396 list)
    exec_dom0 virsh list --all
398 echo "
    Repository OS Domain Package
400 ---------- -- ------ -------
    $(print_tasks | tr '|' ' ')
402 " \
      | column -t
404 ;;
    start)
406 for dom in $(print_tasks | cut -d '|' -f 3 | sort -u)
    do
```

```
408     echo ":: Start domain $dom"
        if exec_dom0 virsh list | grep -qwF "$dom"
410     then
          echo "Domain $dom is already running"
412     else
          exec_dom0 virsh start "$dom"
414       echo "Boot in progress..."
          wait_updown "${DOM_IPS[$dom]}" 22 up
416     fi
    done
418 exec_dom0 virsh list --all
    if [ $# -gt 0 ]
420 then
      echo ":: Execute custom command - $*"
422   ssh -i "$KEY_FILE" -l root "${DOM_IPS[$dom]}" "$@"
      echo ":: Returned $?"
424 fi
;;
426 stop)
    if [ $# -gt 0 ]
428 then
      echo ":: Execute custom command - $*"
430   ssh -i "$KEY_FILE" -l root "${DOM_IPS[$dom]}" "$@"
      echo ":: Returned $?"
432 fi
    for dom in $(print_tasks | cut -d '|' -f 3 | sort -u)
434 do
      echo ":: Stop domain $dom"
436   if exec_dom0 virsh list | grep -qwF "$dom"
      then
438     #exec_dom "$dom" bash -c 'poweroff &'
        #wait_updown "${DOM_IPS[$dom]}" 22 down
440     exec_dom0 virsh destroy "$dom"
        sleep 2
442   else
        echo "Domain $dom was already shut off"
444   fi
    done
446 exec_dom0 virsh list --all
;;
448 run)
    for v in $(print_tasks)
450 do
      repo="$(cut -d '|' -f 1 <<<"$v")"
452   os="$(cut -d '|' -f 2 <<<"$v")"
      dom="$(cut -d '|' -f 3 <<<"$v")"
454   pkg="$(cut -d '|' -f 4 <<<"$v")"

456   "$0" -k "$KEY_FILE" -d "$dom" start

458   echo ":: Create snapshot for domain $dom"
      exec_dom0 virsh snapshot-create "$dom"
460   exec_dom0 virsh snapshot-list "$dom"
```

```
    echo ":: Start of validation tests - $(date -R)"
    print_test_script "$REPO_URL/binaries/$repo/$os" "$pkg" \
      | exec_dom "$dom" bash -l \
      | tee "$(date +%F.%T).$repo.$os.$dom.$pkg.log"
    echo ":: End of validation tests - $(date -R)"

    if [ "$INTERACTIVE" = 'yes' ]
    then
      echo ":: Run interactive shell"
      ssh -i "$KEY_FILE" -l root "${DOM_IPS[$dom]}"
      echo ":: Returned $?"
    fi

    echo ":: Restore snapshot $snapshot for domain $dom"
    snapshot=$(exec_dom0 virsh snapshot-list "$dom" \
        | awk 'NF > 3 { t=$1 } END { print t }')
    exec_dom0 virsh snapshot-revert "$dom" "$snapshot"
    exec_dom0 virsh snapshot-delete "$dom" "$snapshot"

    "$0" -k "$KEY_FILE" -d "$dom" stop
  done
;;
errors)
  for v in $(print_tasks)
  do
    for f in *."${v//|/.}.log"
    do
      if [ -f "$f" ]
      then
        echo "$f"
      fi
    done
  done \
    | xargs -r grep --color=auto -C 2 -iwHE '(E:|error|fatal|fail)
        '
;;
*)
  fail "$1: unknown command"
esac

exit 0
```

Listing 3: Script contrail-validation.sh

B.4 Subversion hook script

```
#!/bin/bash
#
# Name
#    pre-commit: SVN pre commit hook for the Contrail project
#
# Versions:
```

```
#     2012-01-25
#         send rejected commits infos to contrail-notifs mailing list
#         added .DS_Store in the list of forbidden files
#         updated error messages
#     2011
#         first versions
#
# Usage:
#
#   [1] REPOS-PATH   (the path to this repository)
#   [2] TXN-NAME     (the name of the txn about to be committed)
#
#   [STDIN] LOCK-TOKENS ** the lock tokens are passed via STDIN.
#
#   If STDIN contains the line "LOCK-TOKENS:\n" (the "\n" denotes a
#   single newline), the lines following it are the lock tokens for
#   this commit.  The end of the list is marked by a line containing
#   only a newline character.
#
#   Each lock token line consists of a URI-escaped path, followed
#   by the separator character '|', followed by the lock token string,
#   followed by a newline.
#
# The default working directory for the invocation is undefined, so
# the program should set one explicitly if it cares.
#
# If the hook program exits with success, the txn is committed; but
# if it exits with failure (non-zero), the txn is aborted, no commit
# takes place, and STDERR is returned to the client.   The hook
# program can use the 'svnlook' utility to help it examine the txn.
#


# Path to the SVN repository
REPOS="$1"

# Id of the transaction
TXN="$2"

# List of users for which rules do not apply (admins)
ADMINS="sandre"

# Freeze the code if value is "yes"
FREEZE="no"
```

```
52
# Execute the svnlook command $1
54 svnlook() {
  /usr/bin/svnlook "${1?}" -t "$TXN" "$REPOS" \
56   || reject "server or connexion error"
}
58

60 # Reject the commit with error message $1
reject () {
62   mail -s "Contrail: svnhook usage: pre-commit" contrail-admin@ow2
       .org <<<"
A commit has been rejected by the Contrail SVN pre-commit hook.
64
Committer: $(svnlook author)
66 Date: $(svnlook date)

68 Message:
${1:-"<empty>"}
70
Changed:
72 $(svnlook changed)
"
74
   echo "
76 ERROR: ${1:-'your commit transaction has been rejected'}.

78 For further explanations about this error message,
please refer to our wiki page http://contrail.projects.ow2.org/
    xwiki/bin/view/Development/Development+Practices#HCommitHooks
80
If you think your commit shouldn't have been rejected, please
    contact
82 us at <contrail-admin@ow2.org>." >&2
   exit 1
84 }

86 # Allow the commit
allow() {
88   echo "commit ${TXN}@${REPOS} allowed"
   exit 0
90 }

92



94
#
    ##############################################################################


96
# commit allowed if author is one of the admins
98 echo "$ADMINS" \
   | grep -qwF "`svnlook 'author'`" \
100   && allow
```

```
102
# commit not allowed if code frozen
104 [ "$FREEZE" = "yes" ] \
  && reject 'source code is currently frozen'
106

108 # commit rejected if log message does not contain any printable
    character
svnlook 'log' \
110   | grep -qE '[^[:space:]]' \
   || reject 'commit message is blank or empty'
112

114 # commit rejected if attempt to modify repo structure
svnlook 'dirs-changed' \
116   | grep -qxF '/' \
  && reject 'trying to modify the repository structure'
118

120 # commit rejected if trying to add or update eclipse config files
svnlook 'changed' \
122   | grep -qE '^(A|U|_U|UU)[[:space:]]+.*/\.(classpath|project|
      DS_Store|settings/)' \
  && reject 'trying to commit configuration files'
124

126 # commit rejected if trying to add or update a data file
svnlook 'changed' \
128   | grep -qE '^(A|U|_U|UU)[[:space:]]+.*\.(log|class)$' \
  && reject 'trying to commit data files'
130

132 # All checks passed, so allow the commit.
allow
```

Listing 4: Subversion pre-commit hook