Project no. 257438

CONTRAIL

Integrated Project
OPEN COMPUTING INFRASTRUCTURES FOR ELASTIC SERVICES

Architecture Design and QoS constraints matching algorithms in Federations D2.2

Due date of deliverable: July 31st, 2011
Actual submission date: September 5th, 2011

Start date of project: October 1st 2010

Type: Deliverable
WP number: WP2
Task number: T2.2

Responsible institution: HP-IIC
Editor & and editor's address: Lorenzo Blasi
Hewlett-Packard Italiana S.r.l
Innovation Center
via Grandi, 4 - 20063 Cernusco Sul Naviglio (MI)
Italy

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
0.0	11/06/11	Lorenzo Blasi	HP-IIC	Table of Contents
0.1	01/07/11	Emanuele Carlini, Massimo Coppola, Patrizio Dazzi, Giacomo Righetti	CNR	First draft of federation key objectives and architecture
0.1.1	08/07/11	Patrizio Dazzi	CNR	Minor contribution to Sec. 3.2
0.1.2	12/07/11	Philip Kershaw	STFC	First additions to 2.3
0.2	12/07/11	Matej Artac	XLAB	Federation API section
0.3	13/07/11	Lorenzo Blasi	HP	Draft content for sections 4.2, 4.3, 4.4. Version for the first internal review
0.4	24/07/11	Lorenzo Blasi	HP	Completed content for sections 4.2, 4.3, 4.4; drafted content for 4.5
0.45	25/07/11	Emanuele Carlini, Patrizio Dazzi	CNR	Edited sections 3 and 4, added Architecture module, Scenarios, Coordination
0.46	27/07/11	Emanuele Carlini, Patrizio Dazzi	CNR	Added Provisioning, Mapping
0.47	30/07/11	Giacomo Righetti, Emanuele Carlini	CNR	Added Federation Deployment, Application Model
0.48	01/08/11	Giacomo Righetti	CNR	Diagrams to pdf, Deployment revised.
0.5	02/08/11	Christian Temporale	HP	Introduction; completed section 2.2
0.55	02/08/11	Giacomo Righetti	CNR	Updated structural view of federation-support architecture, edits to 3.2, 3.2.1, 3.2.2
0.6	04/08/11	Christian Temporale	HP	Executive summary
0.65	04/08/11	Giacomo Righetti	CNR	Edits to Architecture modules, security-related sequence diagrams, changed some module names
0.67	09/08/11	Giacomo Righetti, Massimo Coppola	CNR	Image Manager, edits related to internal reviewers comments
0.7	09/08/11	Philip Kershaw	STFC	Fixes to section 5.2.7
0.8	11/08/11	Chris Kruk	STFC	Added and corrected sections 2.3.1, 2.3.2 and 3.2.4
0.9	12/08/11	Philip Kershaw	STFC	Added Security Policies - 2.7 and remaining content for 2.3
0.91	13/08/11	Massimo Coppola	CNR	Merged back changes from CNR
0.92	19/08/11	Massimo Coppola	CNR	Edits to sections 2,3,4 to apply internal reviewer comments.
0.93	25/08/11	Massimo Coppola, Patrizio Dazzi, Emanuele Carlini, Giacomo Righetti	CNR	Updates to sections 2,3,4 – roadmap, scenarios, algorithms.
0.94	31/08/11	Massimo Coppola, Patrizio Dazzi, Giacomo Righetti	CNR	Final version of sections 2,3,4.
0.95	03/09/11	Lorenzo Blasi	HP	Revised recent changes.
1.00	05/09/11	Massimo Coppola, Patrizio Dazzi, Giacomo Righetti	CNR	Final document.

Reviewers:
Guillaume Pierre (VUA) and Piyush Harsh (INRIA)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T2.2	Architecture design of the federation management support, and its revision for the second implementation	CNR*, INRIA, STFC, HP-IIC

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

Executive Summary

This deliverable analyses both functionality and algorithms for Contrail Federations and provides an architectural design which takes into account aspects of Identity management and Security.

The Federation plays a central role in Contrail: users access Contrail resources through the Federation, which is one the main selling points of Contrail. A Federation can be seen as a bridge between cloud users and cloud providers, since it mediates between users' requests and providers owning the resources.

The document starts with an overview of the key functionality offered by a Federation; then it introduces a model describing the relationships between the entities involved in a Federation. The analysis takes into account aspects of security, administration of user identities and management of cloud providers, as well as tracing those aspects and functions to the requirements gathered in the first phase of the Contrail project. In the final release, the Contrail Federation will manage multiple providers, possibly including external cloud providers such as Amazon and Azure.

The analysis of the coordination of different cloud providers by means of their SLA is not a trivial problem and it requires the utilization of optimization algorithms. For this reason, as implementing all the functionality at once is not realistic, two different scenarios – a basic one and an advanced one – have been identified; the two Federation prototypes to be released according to the Description of Work aim at implementing those two scenarios.

The Federation architecture is discussed from different perspectives: a static view showing Federation layers and internal components, a dynamic view showing the interactions between modules and a deployment view showing how software components are deployed into available hardware.

In the analysis of Federation algorithms, a big issue is the management of SLAs on different cloud providers. The difficulty comes from the fact that a Federation is not just a simple broker, but plays an active role and mediates between the aims of the final user and those of the set of providers. A Federation, in addition, may be seen itself as a provider. Once understood the real added value offered by the Federation in a cloud ecosystem, it becomes apparent that a Federation may also be a business entity earning money from its services. This implies that a Federation should also handle typical business issues, such as risk management, for being profitable. For instance, when a SLA is split between several providers, the Federation must manage prices but also penalties, both with users and providers: a SLA violation from a provider may result in an economic loss for the Federation.

Finally, flexibility and usability of the interfaces play a critical role in a Cloud, hence they also do in Contrail Federations. Three different kinds of interfaces have

been defined: a graphical web User Interface, a REST interface and a command-line interface. In CONTRAIL, the same REST interface could be used to access both providers and the Federation, thus simplifying the overall implementation, and allowing for all standard interfaces (e.g. OCCI) to be laid on top of both. A further advantage is in the opportunity to compose Federations and providers in a transparent way, allowing to exploit the Federation as a single cloud provider.

As already pointed out in D2.1, WP2 activities intrinsically overlap with other work packages, especially WP3, WP7 and WP10; Federation details on topics discussed in other deliverables are properly referenced in this document.

Contents

1 Introduction

1.1 Scope of the document

The purpose of this document is to provide the architectural design of the federation support for Contrail, starting from the key objectives of the federation, including the various forms of QoS constraints, and proceeding to analyze its implementation in greater and greater detail. After discussing the design from the functional, behavioral ans structural points of view, the document describes the main algorithms used in the federation and what are the interfaces exposed by the federation.

1.2 Document Structure

The document is made of 6 main sections and 2 appendices.

Present section 1 is the Introduction, it describes the scope of the document and its structure, including the definitions of the main terms and acronyms used hereafter.

Section 2 describes the High level functionality of the federation. After an introductory description of the key objectives pursued by a federation, it presents a federation model, showing the entities a federation is made of, including its actors and the relationships between them.

We analyze all the specific topics which are relevant to the Federation Workpackage of Contrail (WP2), as well as many cross-WP topics which link to Security (WP7) and SLA management (WP3). We describe how the federation deals with user identities and with providers, how it is managed and what are the federation goals with respect to deployment of applications, management of SLAs, and support for security policies. Finally, we present a roadmap including all the functions previously described, stating when they will be made available by Contrail within an evolutionary release cycle.

Two main scenarios are discussed, a Basic Scenario corresponding to a prototype satisfying a subset of the requirements, and an Advanced Scenario that is the reference for the foreseen final release. The technological complexity of implementing QoS constraints over a distributed federated platform clearly affect the two depicted scenarios. The time-line and foreseen roadmap includes both.

Section 3 describes the federation architecture, by providing a structural and a behavioural view. In the structural view, layers and composition of the internal modules of the federation are described in detail. In the behavioral view, interactions among the components are analyzed both in the basic and advanced scenario; as the two scenarios differ in the way the federation support is distributed across resource providers, deployment diagrams show how the federation components are deployed onto physical resources in both cases.

Section 4 is a collection of specific algorithms used throughout the implementation of the federation: Task Interaction Graphs for the Application model; translations between SLA and OVF languages; SLA-OVF compliance checks; lookup of cloud providers on the basis of SLAs; SLA splitting and SLA coordination; scheduling and mapping of resources; provisioning on multiple providers.

Section 5 describes federation user interfaces, which come in three different flavours. The web graphical user interface is intended for the various classes of users (federation administrators, cloud provider administrators, cloud federation users). A REST interface allows to interact with federation resources (SLA, appliances, networks, storage, images, virtual machines and reports) via standard HTTP verbs. A command-line interface mainly addresses the administration needs at different levels (federation, cloud provider or SLA management) but is also available to ordinary users.

Section 6 is the conclusion of this deliverable.

Appendices provide technical details about the federation interfaces. Appendix A describes the different resources available through the federation REST interfaces, while Appendix B contains the list of commands that the different classes of federation users can invoke via a command-line interface.

1.3 Terms definitions

Term	**Meaning**
API	Application Program Interface
CLI	Command Line Interface
DBMS	Data Base Management System
FRM	Federation Runtime Manager (see §3.2.2)
GAFS	Global Autonomous File System
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
OVF	Open Virtualization Format
OpenID	Open standard providing decentralization of users' authentication
OAuth	A security protocol for enabling a resource owner to delegate short terms access rights to that resource to some third party
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PaaS	Platform as a Service
QoP	Quality of Protection
QoS	Quality of Service
REST	Representational State Transfer
SAML	Security Assertion Markup Language
SLA	Service Level Agreement
SLO	Service Level Objective
SPL	SLA-based Provider Lookup (see §4.4)
TIG	Task Interaction Graph (see §4.1)
TLS	Transport Layer Security
UML	Unified Modeling Language
VEP	Virtual Execution Platform
VIN	Virtual Infrastructure Network
VM	Virtual Machine
WS	Web Services

2 High-Level Federation Functionality

In this section we first recall the key objectives of the Contrail federation model and of the underlying software architecture (section 2.1). We then present the abstract federation model in section 2.2 and provide a more in-depth analysis of the different sets of features that will be provided by Contrail federations (sections 2.3 to 2.8).

Whenever these features rely on widely accepted technology, as it mostly happens for the security-related topics of sections 2.3, 2.7 and 2.8, we directly track the requirements gathered in the requirement analysis phase of the project that motivates those features.

For modules within Contrail main focus of research and related with federation code development, the match among requirements and features is instead more analytically presented in the following section 2.10, where the development roadmap is described.

2.1 Key objectives

A federation can be considered as a bridge linking cloud users and cloud providers. As the role of the federation goes beyond mere interface adaptation, federation services act as mediators between users and providers. From a user's perspective, the components and the services supporting the federation (we will refer to them as *federation-support* in the rest of the document) act as a broker for the resources owned by providers participating in the federation. As each provider has its own, potentially different, mechanisms and policies for managing resources, the goal of the federation-support is to provide translation and mapping mechanisms for matching user needs by exploiting federated cloud providers.

The pay-per-use concept relies on the existence of a formally agreed SLA between the user and the provider(s), and the ability to monitor, enforce, and account service use and QoS. Besides the resource selection functionality across multiple providers and the consistent management of resources, a Contrail federation coordinates the SLA support of cloud providers. Indeed, as cloud providers have their own SLA management mechanisms which are useful to exploit, the role of the federation is to setup, coordinate, and enforce a global SLA, eventually negotiating SLAs with providers on behalf of the users. This leads to the possibility of a *vertical SLA management*, that is the integration of the SLA management mechanisms of the PaaS and IaaS stacked layers of a Cloud platform. This kind of vertical integration is already in the scope of SLA@SOI [19, §8.2], allowing to design SaaS and PaaS services which are provider-invariant. Our design applies the same approach to supporting the interaction between a Contrail federation and public as well as private cloud providers. Hence, after delegating the main SLA management to the providers, the federation-support will monitor the SLA agreed and will react to any violation with the aim, if possible, of correcting the application behaviour before the user-agreed SLA is affected. The rest of the section lists the key objectives of the Contrail federation.

Federation-level User ID Management A federation has to provide users with mechanisms allowing them to specify their preferences about cloud providers and resources. Federation-support manages the user identities required to access the cloud providers; it also allows users to inspect their current and historical usage of resources (accounting, billing).

The task of protecting personal user-related information, like user identities, is only the first stone when building security into a federated cloud approach. One of the problems related with federations is to save the users the burden of authenticating with resources belonging to different cloud providers, especially as many of the action on the resources have to be automated and performed 24/7. The federation should exploit proper mechanisms and credentials, in accordance with both user preferences and authentication support of the providers.

Security and protection Security plays an important role in the federation layer design as well as in the whole Contrail project, and directly affects the acceptance of the Contrail platform by potential customers. In the Contrail context, security and protection have a double scope.

- both the users data and their applications should be protected from unauthorized accesses and modifications. The federation should protect users from each other: preventing them from harming other services and from snooping on another one's job or data.
- the federation shall protect itself from service providers and administrators. The federation shall not entrust any specific provider with user information whose confidentiality exceeds the stipulated level of trust for the provider, or that causes a security threat to the security of the federation management itself.

This means that (1) the federation architecture needs its own mechanisms to evaluate and manage trust, (2) the federation's software infrastructure will often play as an independent third-party between users and providers, allowing for enhanced enforcement of properties which in other environments rely solely on institutional trust.

Application Mapping and Deployment With respect to applications, the federation has two core tasks. First, the federation has to provide a mapping between users' application and federated providers. This task is carried out according to both user requirements and user preferences, as specified in the execution constraints and in the SLA. In order to effectively set up and enact the applications mapping, deployment, and execution, the federation exploits static (related with general properties about cloud providers) as well as dynamic information (mainly based on the current provider resources). The actual mapping is a multicriteria problem, so any reduction in its size helps solving it in practical time, whether it is pruning unfit/unreliable providers or sorting them according to some aspect of their expected SLA (e.g. reliability).

The same mapping issue can resurface after the deployment phase and during the application execution, if additional resources were required or a renegotiation of the user agreement was triggered.

SLA Management The federation support has to provide mechanisms that manage the SLA at IaaS federation level possibly by employing the SLA management mechanisms available at the providers.

To begin with, the federation support will manage external systems (i.e. Contrail providers) as well as systems operating according to different protocols (i.e. external, public providers).

Following the SLA@SOI approach, a set of SLA management components are built into the federation core. A hierarchical SLA negotiation is performed between the user and the federation, which requires one or more negotiations among the federation and the providers. The easy case, i.e., a single provider is able to host the whole application, leads to simple negotiations, deployment, and well understood federation margins with respect to resource costs and SLA reliability.

Contrail also offers mechanisms for the decomposition of the application and the associated SLA to cope with the case where no single provider has resources and services matching the user SLA and to exploit the real value of the federation as well. The application is deployed onto multiple providers, each one is in charge of the SLA management for the portion of application it holds. The user-agreed global SLA will be then affected by SLA violations at the providers as well as by misalignment in the behaviour of the application parts.

In the general case, in addition to monitoring SLA violations from portions of the application (i.e. from single providers) and possibly taking corrective actions for those violations, the federation will have to coordinate the partial SLAs, as explained in Section 2.6, in order to ensure that the overall SLA is reliably satisfied without having to perform massive over-provisioning.

2.1.1 Federation Non-functional Requirements

In addition to the functional commitments, the federation-support has also to address specific non-functional requirements. They are mainly related with platform scalability, flexibility and reliability.

We shall definitely stress that the distinction between functional and non-functional requirements applied to the federation is completely different from that ordinarily applied to programs and applications.

Most of the classical non-functional goals of application execution, e.g. performance and reliability, once the application gets deployed on a Cloud (in accordance with an agreed SLA) become functional requirements for the service provider and, as such, have been mentioned in previous paragraphs[1].

What is left in the non-functional specification of the federation are key aspects of the federation which do not immediately affect the realization of any specific business case of the federation support, but they are paramount to allow a wide acceptance of the federation support developed by the Contrail consortium.

Open-source The whole core platform shall be open-source licensed. This non-functional requirement is a project choice and a DoW constraint.

Performance of the federation core is the result of the architecture design, of the algorithms used, and of the distributed cooperation mechanisms within the federation. If we look at the incoming flow or service negotiation requests and management actions[2] we can see federation performance as a combination of the *Response Time* of the interfaces, of the *Efficiency* which can be achieved in managing the requests, and of the *Availability* that the federation is able to offer.

Scalability with respect to the federation size (number of users, providers, services, active applications, . . .) is addressed by design: (1) by defining a potentially very distributed architecture whose basic components can be also used as a centralized solution (2) by a decentralized leveraging of all the management mechanisms of service providers.
Similar *Flexibility* is also aimed at when adapting existing state-of-the art software solutions to work within Contrail (e.g. security-related infrastructures).

Reliability and *Robustness* of the federation architecture are essential features underlying and granting practical value to all the adopted QoS and QoP

[1]Granting a specified QoS (or dealing with a compliance failure) is a functional commitment for Cloud providers. This applies to the federation support too. Besides performance, a major concern of the federation-support is security. Contrail introduces specific QoP terms in its SLA model, so that most security guarantees become functional requirements on the federation platform.

[2]Those are meta-actions outside the scope of formally agreed SLAs on IaaS and PaaS resources.

mechanisms.

These considerations influence the design of the federation-support, presented in Section 3.

2.2 Federation Model

This section describes an abstract model representing the entities involved at Federation level and their relations. The model shows how a Contrail Federation should be organized and which are the main objects that each Federation actor will be able to interact with. The model is the basis both for the Federation architecture (described in Section 3) and interfaces (which will be detailed in Section 5 and in the appendices A and B).

Contrail also defines another abstract model: the Provider model (described in deliverable D10.1 [15]), which represents entities involved at Provider level. As the Federation model is the basis for the Federation API, the Provider model is the basis for the Provider API. The two models contain some entities which, even if similar in their name, can represent different, linked objects, or different aspects of the same object.

As an example, the "Appliance" entity in the Federation model is an item in the Federation catalog of resources, an item which is offered to Federation users. While this Federation's "Appliance" entity represents a potentially running appliance, with all its potential constraints and parameters, in the Provider model described in D10.1 the appliance entity — named "cAppliance" — represents an actually running instance of the appliance, which is fully specified (except possibly for its yet unexploited elasticity margin). Obviously there should be a link between the two objects: each Federation level Appliance object will hold a list of references to its running instances (cAppliance objects).

Another example is the SLA entity defined as "cSLA" in the Provider model of D10.1 that only describes the infrastructure-related aspects of a SLA. A full SLA in the federation model will contain much more information, such as specification of higher level services and related guarantees, upscaling and downscaling rules, price and penalties for the offered service.

The Federation model is shown in Figure 1. Model entities can be grouped in 5 types, showed in different colors in the picture:

- *Federation entities*
  The central entity in the model is the *Federation* entity, which associates a list of *Provider* entities, all bound by a common *FederationAgreement*.

- *Provider entities*
  There are 2 types of Providers: the *ContrailProvider* and the *External-*

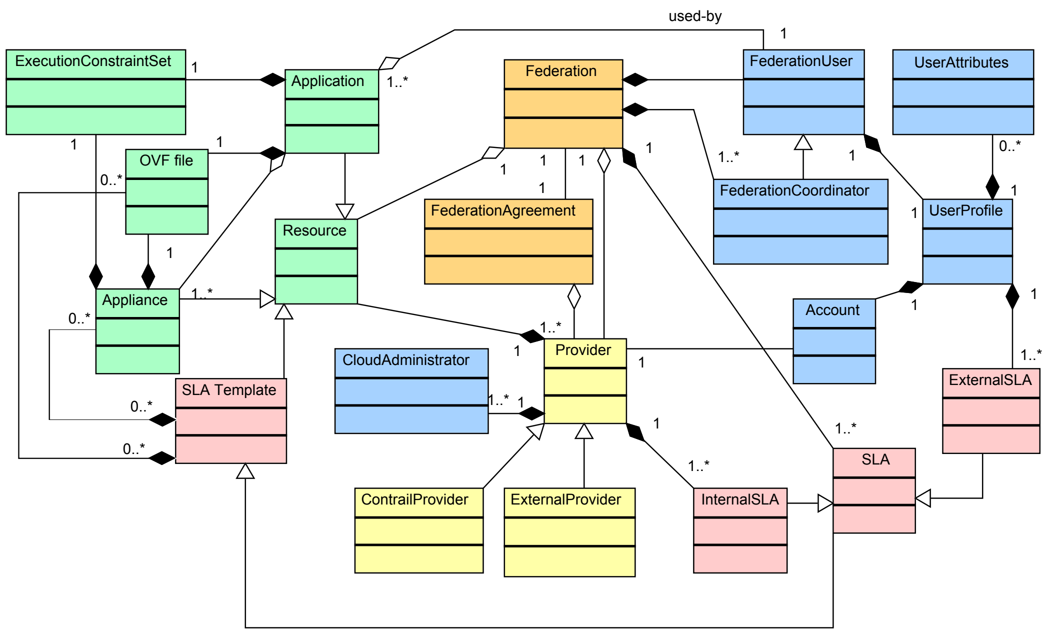


Figure 1: Federation Model

Provider; if the cloud Provider is federated inside Contrail, it is a ContrailProvider; otherwise -like Amazon, for instance- it is an ExternalProvider. The FederationAgreement does not apply in case of ExternalProvider.

- *User entities*
  Federation users belong to 3 categories: *FederationUser*, *FederationCoordinator* and *CloudAdministrator*. FederationUser is the normal final user of the Federation. A FederationCoordinator is a special user that administers the Federation; it inherits from FederationUser. Since the Federation deals with one or more cloud providers, a CloudAdministrator is a user who administers a given Provider. Each FederationUser includes an *UserProfile* entity, describing roles and credentials of a given user: in fact an UserProfile includes the *Account* entity containing the credentials to be used with a given Provider, a set of *UserAttributes* entities specifying user constraints, and one or more SLAs agreed with the Federation (*ExternalSLA*: see also *SLA entities* below).

- *SLA entities*
  A *SLATemplate* is the generic entity from which a real SLA is derived. A SLATemplate is in turn a specialization of the generic *Resource* entity. An *SLA* is a component of a Federation; in general there are more than one SLA in a Federation. Among SLA entities, it is important to distinguish

between *ExternalSLA* and *InternalSLA*. The two separate entities allow the federation to act as a smart mediator between the user and the providers.

  – An ExternalSLA is an SLA agreed between a FederationUser and Contrail. A UserProfile of a FederationUser may include one or more ExternalSLA.
  – An InternalSLA is an SLA agreed between Contrail and a Provider. a cloud Provider may include one or more InternalSLA.

While the External and Internal SLA may even coincide (the Federation acting as a pure broker), in the most general case a set of InternalSLA may be specifically negotiated by the Federation in order to complete the negotiation of an ExternalSLA. Normally an InternalSLA is negotiated after the acceptance of the ExternalSLA with the user; however, there might cases where the Federation pre-agrees InternalSLAs with one or more cloud providers, in order to anticipate future users' needs.

- *Resource entities*
  The Federation aggregates a set of *Resource* entities. In Figure 1 we do not explicitly model the full list of resources, including storage, networks and related services, in order to focus on the key entities for federation management. Apart from the aforementioned SLATemplates, other interesting examples of Resource are the *Appliance* and the *Application* entities. An Application is used by a FederationUser and is composed by a set of Appliance entities. Applications are represented by one or more OVF File (OVFFile) and a set of execution constraints (*ExecutionConstraintSet*). OVF Files and Appliances can be entities composing a SLATemplate.
  Clearly, cloud Providers themselves need Resource entities, as they offer services based on Applications and Appliances.

2.3 Identity Management

The federation manages user identities. These identities are the keys to access to individual cloud provider accounts under a given user's control. As such, the management and protection of this information is critical.

2.3.1 Single sign-on

In Contrail, there are requirements (see D2.1 [16]) stating the need for single sign-on. In the context of Contrail, there are two meanings to the term single sign-on which are orthogonal to each other: the ability to manage multiple cloud provider accounts with a single overarching federation account, and the second, the ability to devolve the authentication credentials of that federation account to an independent Identity Provider. All users have a single federation account for managing multiple cloud provider accounts, but the way in which they authenticate to that account could in itself involve some single sign-on mechanism. It is helpful to think of this from the perspective of layers. At the top layer, a user authenticates with the federation either directly, e.g. username and password, or by some means of single sign-on, e.g. Shibboleth or OpenID. This process binds their credentials to some account held at the federation layer. This account itself has a binding to the layer beneath: one or more cloud provider accounts.

Considering WP2-FIM-01, then, this applies to the abstraction of individual cloud providers beneath a federation layer. This is the primary motivation for single sign-on. It enables the management of a user's identities for each of the cloud providers that the Contrail federation supports under a single account with the Contrail federation. This raises issues of confidentiality: any given user, must entrust their individual cloud provider credentials with the federation. There is a second issue, i.e., interoperability. The federation must support the authentication mechanism used by the various cloud providers supported.

The second requirement (WP2-FAM-01) is an outward facing one in that it concerns how a Contrail federation identity can integrate with other identity management systems. This would enable a user with an existing identity from another Identity Provider to sign into the Contrail federation. For example, a user may have an existing Shibboleth identity with a home academic institution. Rather than create a new identity for the Contrail federation the user should be able to use its existing Shibboleth ID. This should also apply for other examples such as eduRoam or the IGTF PKI.

2.3.2 Federated Accounting

It should be possible for the user to obtain aggregated statistics, and accounting data from the individual cloud providers (WP2-FLA-06). This will necessitate access rights for the federated accounting service to accounting services in the individual cloud providers. To do this, it will also need to be able to access the federation to individual cloud provider account mappings for any given user.

2.3.3 User Account Auditing

Stale user accounts are a security risk to the system providing openings for an attacker to gain unauthorised access. A number of measures can be put in place to mitigate. User activity can be monitored (for example, last login time, last password change made) and any accounts where activity has elapsed for an extended period should be disabled (WP2-FLA-03). Records of federation IDs for accounts that have been disabled or deleted will need to be retained. This is needed to avoid the possibility of accounts being recycled and new users being inadvertently granted the privileges of another existing account.

2.3.4 Federation User Identity Protection

Federation user accounts need to be carefully protected since they hold access information to multiple cloud provider accounts managed by the associated user. Individual cloud account information needs to be encapsulated and stored in such a way as to be isolated to prevent unauthorised access (WP2-FLA-01). Where possible, user delegation technologies should be employed to avoid the need to store sensitive cloud provider account information. For example, the user of proxy certificates [9], a means of enabling an entity to act on behalf of a user with their access rights for a limited time and OAuth [13], a newer technology which provides a means to delegate limited access rights to some third party. In many cases, however, existing authentication mechanism supported by cloud providers will not support delegation and it will be necessary to store individual cloud provider account credentials.

Where, single sign-on is used to enable a user to authenticate to the federation, safeguards should be put in place to ensure these are executed by a secure means. For example, with Shibboleth, the enforcement of a limited set of trusted Identity Providers from a given federation. Also, with OpenID, it can be made more secure by stipulating transport layer security to enforce a white-list of acceptable Identity Providers. Each authentication mechanism should carry with it an associated assurance level as described in the following section.

2.3.5 Levels of Assurance

Levels of assurance provide a metric to quantify the degree of certainty about a given security process. This is typically a given authentication mechanism but can also include the registration process by which an individual entity obtains an account, identity tokens used in a system or mechanisms by which one party can assert to another that a user has authenticated [25]. For Contrail, the ability to record the level of assurance is important to secure sensitive information such as biomedical data (Deliverable 2.1 [16] WP2-FAM-03). Access to such data should not be permitted if a user has used processes with low levels of assurance.

Dependent on the security technologies adopted, strategies can be adopted to communicate the levels of assurance used: with OpenID, the PAPE (Provider Authentication Policy Extension) mechanism [1] and SAML provides the Authentication Context [12], With PKI (Public Key Infrastructure) based authentication, a level of assurance can be inferred from the trust roots (issuing CA [Certificate Authority] certificates) used to issue a given certificate. Different CAs are associated with different levels of trust. A metric for each authentication method needs to be agreed for the system. Immediately following authentication, the given Relying Party should associated the level of assurance value with the user's login session. This should be communicated to other consumers, so that authorisation decisions to access resources can be made taking this value into account.

2.4 Provider Management

In order to participate in a Contrail federation, a provider has to offer a certain set of functionality. This functionality refers mainly to the features the providers make available for the federation-support in order to allow it to register users, to negotiate and coordinate SLA, to submit OVFs and to manage the lifecycle of the applications that are in execution.

The federation issues a user registration request, provider by provider, when a new user registers to the federation. To this end, each provider has to expose a proper interface allowing the federation to submit/specify a proper user name and a user password.

Each provider has also to offer an interface allowing the federation to negotiate an SLA on behalf of the user. In case of Contrail clouds it means to make available an interface supporting a well-defined set of SLA@SOI mechanisms [19], as detailed in the Deliverable 3.2 [17]. A provider shall also allow the federation-support to interact with the local provider SLA@SOI Manager to coordinate the SLA activities among different cloud providers.

A cloud provider has also to make available to the federation the mechanisms for the submission of appliances in the form of OVF [14] files.

Each provider has to allow the federation to gather information about running applications as well as the actual status of resources. Unfortunately, not all providers would like to share information about their infrastructure, neither static nor dynamic. This is true in particular for public providers like Amazon. However, even providers of Contrail-based clouds can decide to share only a limited amount of information. It is worth noting that this kind of choice can condition the business model of the Contrail federation as well as can condition the function driving the cloud selection.

2.5 Application Mapping and Deployment

In the federation, we consider an application as composed by: (i) a set of appliances and (ii) a set of *execution constraints* that provides the user requirements on a per appliance basis. An appliance identifies a set of VM images strictly cooperating to realize an application fundamental block (e.g. a pool of web servers, or a firewall and back-end database combination) and often sharing similar requirements and constraints with respect to storage and network resources.

The mapping module adopts a two-step algorithm, where the first step performs a coarse-grain filtering of the available providers, reducing the problem size for the second step.

In the first step the scheduling and mapping module collects both the appliance descriptions and the associated execution constraints. It employs the information to discard cloud providers that do not satisfy some requirements; for instance some provider may need to be ignored due to geographic constraints on resources imposed by the user, or because specific resources are unavailable temporarily or permanently unavailable there.

The second step exploits the shortlist so produced in devising and providing a mapping plan for all appliances. While in principle there is a huge number of solutions for mapping appliances to services, several hypotheses are exploited.

- Groups of similar appliances are mapped as a block, reducing the size of the problem by transforming program structure into provider locality.
- Dedicated libraries/solving engines can be used in order to solve the related optimization problems, like e.g. the GNU Linear Programming Kit [5] for large-linear and mixed integer optimization.
- Heuristics can be applied to drive the mapping along specific criteria, like the foreseen economic cost or the foreseen performance or reliability. Since these heuristics can be used to sort the available mapping options, we will investigate the option to allow the user to specify them, or to select which combination of heuristics to use for a given application, e.g. *"given the SLA is obeyed, ignore further performance gain margins and select the resource set with minimal cost"*.

2.6 SLA coordination

In the Contrail project we assume that every cloud provider belonging to the federation has proper mechanisms to deal with the SLA descriptions regarding the appliances it has to execute. Most of those mechanisms and the underlying formalism are inherited from the SLA@SOI [23] project. In particular, the SLA management yielded by Contrail cloud providers is based on three main entities: (i) SLA, (ii) SLA Template and (iii) SLA Manager.

The SLA is a structured description of user and appliance requirements, which is derived by a SLA Template and approved both by the user and the provider after a specific negotiation phase.

A SLA Template provides a customizable base that can be exploited in order to derive specific SLAs.

A SLA@SOI SLA Manager monitors a running appliance and reacts in case the appliance misbehaves with respect to its associated SLA. The actions enacted by a SLA Manager include intra-cloud appliance migration, appliance reconfiguration, and network set-up.

The federation should intervene and coordinate the involved SLA managers, in case they were unable to enforce the SLA of one or more appliances. To this end the federation extends the SLA@SOI structure, (i) by including new mechanisms to address specific needs in SLA coordination and (ii) by moving existing SLA@SOI mechanisms at federation level.

The need for active SLA coordination comes for the observation that for many applications, unless considerable widespread overprovisiong is performed via the partial SLAs, the overall SLA might not be granted with enough certainty (although satisfied on average). As an example, it is often enough that a critical part of the application is in violation to jeopardize the user SLA, and this is an issue that that the penalty mechanism of the single providers cannot generally compensate for.

The federation-support thus extends the SLA management above the level of the providers in a hierarchical way, employing provider-level actions (e.g. renegotiation, migration, Cloudbursting) to coordinate multi-provider execution.

2.7 Security Policies

In this section, we consider the security policies provided by the federation to support both clients and cloud providers. Considering clients first, user identities are managed at the federation level (requirements WP2-FLA-04 and WP2-FLA-05). Users may be registered with a username and password at the federation, (here, the federation is acting as an Identity Provider). Alternatively, they may choose to use an account from another Identity Provider using single sign-on (for example OpenID or Shibboleth). Whatever means is used, each user is represented with a unique identity in the federation layer.

2.7.1 User Policies

Various attributes will be associated with federation user identities (WP2-FS-02). For example, the authentication method(s) used by a user and the associated levels of assurance (see section 2.3.6). It may also be necessary to hold contact information such as e-mail address for notification purposes (WP2-FLA-09, WP2-FLA-11). Attribute information may be pushed over an authentication channel to a relying party or else the relying party may pull attribute information it needs from an Attribute Authority (see section 3.2.4 Security Modules). An *attribute release policy* is needed to ensure that user attribute information is protected. This would consist of a list relying parties that have the required permission to access given user attribute information.

There are number of ways this could be configured:

- Static information at account creation could set a number of accepted relying party identities;
- Users maintain profiles in their federation accounts which determine trusted relying parties and their access rights over given attributes;
- A dynamic model in which, when a relying party requests user attribute information the user is prompted to grant or deny permission;
- When a user adds an account for a given cloud provider certain permissions are granted for that provider to access given attributes for that user.

A fundamental feature of the federation layer is the ability to manage multiple cloud providers on behalf of users. A given user's account metadata will associate their federation account with multiple cloud provider accounts (WP2-FLA-12). Further, WP2-FLA-13, implies the ability for multiple federation accounts to access a single cloud provider account. To support this capability, these accounts will need to be managed with group-based access policy: to access a given cloud provider account within a given set of constraints you must be a member of this dedicated group. Careful management and validation of these policies will be critical to establish trust for both users and cloud providers. Administrators will require appropriate privileges to access these account mappings (WP2-FLA-10).

2.7.2 Cloud Provider Security Policies

Each cloud provider supported by the federation requires associated policy metadata to govern access to it. From WP2-FLA-15, this data will need to include a list of federation accounts that are registered to use it as a means to block unauthorised access requests. This metadata will also need to encapsulate usage policy and enforce recorded usage (WP2-FLA-07, WP2-FLA-08).

2.8 Federation Coordinator

The role of the federation coordinator defines the entity in charge of the set-up and maintenance of the federation. We purposely used the term coordinator instead of administrator as it best fits the responsibilities of the role, and it underlines the differences from the role of provider administrator.

The designation of a coordinator(s) can follow from an agreement among cloud providers who want to federate, or can be the choice of a single organization proposing itself as the first seed of a Cloud federation. In each case, more people can enjoy the coordinator role as 24/7 supervision may be required in a federation.

The coordinator uses a proper, dedicated, interface that is described in Section 5 and in the appendices. The coordinator is mainly involved in extraordinary maintenance of users accounts, federation policies, and in the definition of the federation structure itself via the addition/removal of providers and federation access points.

User account management is largely automated via web interfaces, but the coordinator may be involved in special cases, e.g. in account suspensions or removal due to resource misuse (requirements WP2-FLA-07, WP2-FLA-11, and WP2-FLA-14 defined in [16]).

Federation composition The coordinator is in charge of enabling new providers that join the federation, as well as of removing those that either decide to leave the federation, or critically misbehave (use cases WP2-UC01 and WP2-UC02 in [16]).

It is worth noting that the decision to accept or reject a new provider is based on a form of institutional trust that the coordinator is merely asked to certify and materially perform (e.g. by exchanging certificates or inspecting software installations). The same may hold when removing existing providers, although some kind misbehaving, e.g. security issues, can lead to immediate actions.

Role and Policy Administration The coordinator materially defines federation subsidiary roles (e.g. user groups) and federation policies (requirement WP-FS-02 in [16]). In addition to ordinary policies for authorizing users/roles to specific services and resource allocations, Contrail can also use policies to trigger federation-wide warnings and changes ([16] requirements WP2-FLA-07, WP2-FLA-08, WP2-FCM-01 to WP2-FCM-04). These *triggering policies* can exploit the federation monitoring infrastructure to indirectly affect the application of ordinary service policies. As an example, consider a policy that defines how the reputation of providers changes when certain events happen or when a certain feedback is received by a user.

2.9 High Level Scenarios

The overall goals and the whole set of functions discussed so far will be provided by the final version of the Contrail federation-support. We approach its implementation by following an evolutionary development path. A number of intermediate versions will thus be released before the final full-featured version. Each release will provide a super-set of the functionality provided by the previous one, with new functions targeting additional project requirements. As major landmarks we define two main scenarios, representing specific evolution stages of the federation support.

Before describing the scenarios, we provide early details on a preliminary version which is expected to be ready before the first Contrail release, a working prototype which is mainly aimed at validating the technical solutions.

The first real scenario refers to a basic, centralized implementation of the federation functionality. Centralized means that the federation has a single access point that can reach the different providers. The access point can receive simple applications, manage their SLAs and gather simple, static information about the cloud providers. The mapping schema is driven according to a straightforward objective function.

The second scenario, which targets the second Contrail release, will rely on a distributed structure of the federation-support, it will include research results about support for multi-criteria object functions and enhanced SLA coordination.

A quick comparison of the first release and of the two main scenarios we present, simply based on the features they support, is reported in Table 1. More detail is given in the Roadmap Section 2.10.

2.9.1 Preliminary release

Work Package 2 will provide a preliminary prototype before the first official release of the Contrail project. This version of the prototype will not be fully integrated with the other Contrail subsystems, however it will provide a reduced set of functionality whose aim is to give an initial proof of the concepts described in this deliverable.

This preliminary version will consist in a federation support able to perform the identity mapping of users but that will likely not fully deal with new user registration. The prototype will accept applications for submission, but without allowing users to negotiate an SLA with the provider, i.e., the applications are executed on federated cloud providers in either a best-effort or a fixed-SLA fashion. This version of the federation-support will not deal with all the complexities of moving the appliances to the providers, by assuming that all the appliances composing the applications of the users are already accessible by the cloud providers.

The preliminary release prototype will address the issue of integrating the federation core with the VEP and the OpenNebula systems, taking into account a preliminary release of the security subsystems (e.g. addressing static authorization instead of the full UCON model, see Deliverable 7.1 [18]). Basic integration with the VIN, the GAFS ad a subset of the ConPaaS modules will be attempted for this prototype.

2.9.2 Basic Scenario

This scenario describes the features of a very basic cloud federation support. In order to use the resources of federated providers, each user has to register to federation in order to obtain a federation-level account. Behind the scenes, the federation-support creates, on behalf of the user, an account in each federated cloud provider. In this scenario the user can neither select for which provider to create her accounts nor provide any pre-existing accounts.

After the registration, a user can submit her applications to the federation. Each application is represented by a single OVF file that contains the set of execution constraints directly related to the appliances, and by an SLA proposal derived from providers' SLA templates. The federation exploits the SLA proposals and the OVF constraints to drive the providers selection phase. A user can condition the selection phase by selecting a predefined ranking criteria (realized as an objective function), such as cost minimization and completion time. In this scenario a user can select only one objective function at a time. After the selection phase, the federation negotiates the SLA proposals with the selected provider(s). The details of the SLA negotiation process are given in Figure 5 of Section 3.3. One of the limitations of this scenario is that the SLA splitting (as well as the corresponding virtual images) is bounded to a limited number of aspects (e.g. storage and computation). As a consequence, the split can be performed only on a service basis (see Section 4.5) and consider no more than two providers.

2.9.3 Advanced Scenario

This scenario describes the features of an advanced cloud federation support. It is presented by highlighting the differences against the basic scenario.

In addition to the features provided by the basic scenario, the user after the registration can customize her account by specifying which are the cloud providers she wants to use. Moreover, the user can indicate to the federation any pre-existing accounts she already has for certain clouds. In this case the federation-support creates user accounts only for the clouds for which the user didn't indicate any pre-existing account. Each user can also specify general preferences that conversely from the basic scenario the federation shall consider for every application submitted by that user (e.g. all my application must stay in Europe).

Like in the basic scenario, after the registration, a user can submit her applications to the federation. However, now the application can be represented by multiple OVF files as well as multiple set of constraints. Even the selection phase is enhanced in this scenario: now the user can select multiple objective functions to maximize among a set of predefined ones. She can also specify a weight for each function, representing its importance. Then, in this scenario, the SLA result-

	Preliminary	Basic	Advanced
Features			
Identity management	Single Sign On	User registration	Fully customizable account mapping
Distribution	centralized	centralized	distributed
Application description	single OVF	single OVF	multiple OVF
Application deployment	forward to low-level provider	based only on static information about providers, single objective function	based on both static and dynamic information, multiple objective function, general user preferences
SLA coordination	N/A	basic SLA management	enhanced SLA management
Parameters			
Number of providers	1 – 4	≈ 10	≈ 100
Number of access points	1	1	≈ 100

Table 1: Features related to the actual releases of the federation

ing from the negotiation can be split in several parts, according to the distribution of the virtual images among the federated cloud providers. The split can be performed on a service, resource or performance basis (see Section 4.5) or even a combination of them.

In this scenario the federation can take advantage of resources belonging to external, private and/or public cloud providers.

2.10 Roadmap

According to the Contrail description of work, the WP2 has to release two federation prototypes. The first one is due at month 18. It consists in a basic federation support providing a set of functionality that allow users to interact with the federation in a way compliant with the description of the basic scenario. The second release is due at month 32. It represents the final version of the federation support. This prototype will consist in a full-featured federation support.

As already mentioned before, in addition to these releases, the Work Package 2 will provide intermediate releases starting from a preliminary release, that is expected to be presented around month 12, providing a limited set of functions. Table 2 lists how the requirements are mapped for each release.

	Preliminary	Basic	Advanced
Requirements			
Identity management	WP2-FIM-01 WP2-FIM-04	WP2-FIM-07 WP2-FAM-02 WP2-FLA-05 WP2-FLA-06 WP2-FS-02	WP2-FIM-02 WP2-FIM-05 WP2-FIM-06 WP2-FAM-03 WP2-FLA-03 WP2-FLA-04 WP2-FLA-09 WP2-FLA-12 WP2-FCM-04 WP2-FSA-05
Provider management	WP2-IAS-05	WP2-FS-05 WP2-IAS-01	*WP2-FLA-16* WCP-ARC-02 WP2-VEP-01 *WP2-PGD-06*
Security	WP2-FLA-04	WP2-FLA-05 WP2-FLA-07 WP2-FS-01* WP2-FS-02	WP2-FLA-08 WP2-FLA-09 WP2-FLA-15 WP2-IAS-02
Application mapping and provisioning	WP2-PR-01	WP2-FS-06 WP2-PGD-01 WP2-EDD-03 WP2-PR-02 WP2-MAP-01	WP2-FCM-05 WP2-FS-04 WP2-MAP-02
SLA organization		WP2-FS-06 WP2-PGD-01 WP2-PGD-05 WP2-EDD-03 WP2-SLA-01	WP2-FS-04 WCP-ARC-02 WP2-PGD-04 WP2-MPS-01 WP2-SLA-02 WP2-SLA-03
Federation coordinator		WP2-FLA-07 WP2-FLA-14 WP2-FCM-01 WP2-FS-02	WP2-FLA-08 WP2-FCM-04
Non-functional features			WP2-FSA-01 WP2-FSA-02 WP2-FSA-04 WP2-NF-01

Table 2: Mapping of requirements over the Contrail federation scenarios.

2.10.1 Features provided by the preliminary release

The preliminary release offers a limited set of functions. Users are able to access all Contrail resources with a single account mapped with multiple credentials on local providers (requirements WP2-FIM-01, WP2-FIM-04, WP2-FLA-04). Users issue application provisioning by submitting a single OVF file (WP2-PR-01). Appliances repository is local to cloud providers (WP2-IAS-05).

2.10.2 Features provided by the first release

In the first release, users (potentially without any cloud provider account) are issued with a federation account (requirement WP2-FAM-02). To this end the federation assigns to each user a unique ID (WP2-FLA-05) and a role (WP2-FS-02) within the federation. The federation-support is able to register each user to every cloud provider (WP2-FIM-07) and to keep track of the users resource usage (WP2-FLA-06). Each user has associated a threshold quota for resource consumption (WP2-FCM-04). The federation interrupts user activities if the threshold is overtaken (WP2-FLA-07). In addition, the federation coordinator is able to ban a misbehaving user (WP2-FLA-14). The federation guarantees users isolation both from external environment (i.e. user resources are not accessible from outside, requirement WP2-FS-05) and from other federation users (WP2-FS-01).

Users can select applications from a federation-level images repository (WP2-IAS-01). The first release allows user to provide SLA proposal (WP2-SLA-01) about applications on a limited number of aspects, which can be statically evaluated (WP2-MAP-01) and include geographical constraints (WP2-FS-06). Nevertheless, SLA proposals may be defined on different subsets of an application (WP-PGD-01). The SLA enforcement exploits specific interface used by an application to export monitoring information (WP2-PGD-05). This release supports a basic management of SLA violations arisen from cloud providers (WP2-SLA-03). In order to execute the application mapping, the federation takes in account user preferences (WP2-PR-02) and offers to users a set of predefined objective functions that are applied to static information gathered from cloud providers (WP2-MAP-01). The federation coordinator may influence the mapping by defining reputation policy about providers (WP2-FCM-01).

2.10.3 Features provided by the final release

The final release allows the federation of a large number of cloud providers, potentially managed with a scalable distributed structure (WP2-NF-01).

With respect to identity management, the federation provides a quick and consistent authorization service (WP2-FSA-01), logging service (WP2-FSA-02) and accounting services (WP2-FSA-04). Users can customize their accounts by defining a certain set of preferences, which include: (i) accounts linked to specific cloud providers (WP2-FIM-02, WP2-FLA-12), (ii) ranking criteria affecting cloud provider use (WP2-FIM-05) and their associated costs (WP2-FIM-06). The federation-support provides mechanisms for encouraging users to periodically check the consistence of their account data (WP2-FLA-03) as well as a virtual bill system to apply the provider virtual costs to the accounted usage data (WP2-FSA-05).

To each user a quota is associated for resource consumption. The final federation release notifies the users whenever a certain threshold is reached (a defined soft-limit) in order to prevent unexpected interruptions of user activities (WP2-FLA-08). This implies an event notification system, which raises events related to applications state modification (e.g. when a user application has been executed WP2-FLA-09). An authorization system prevents users from exploiting resources that they are not authorized to use (WP2-FLA-15).

Users can provide both SLA proposals and already negotiated SLAs (WP2-SLA-02). The SLA attributes that users can specify are extended and include also abstract terms about the application behavior (WP2-PGD-04). The federation-support deals with SLA attributes regarding the QoP enforced by the executing platform (e.g. storage and network encryption) (WP2-FS-04). In addition to the standard violation management supported by the first version (WP2-SLA-03), this prototype is able to react to violations performing activities that includes inter-cloud elasticity (VCP-ARC-02), inter-cloud migration (WP2-VEP-01), as well as cloud-bursting (WP2-MPS-01).

The mapping and deployment features are enhanced compared with the ones of the first version: advanced cost models (WP2-FCM-05) allows users to define their own functions; dynamic information about provider resources (WP2-MAP-02) can be exploited to define cost functions. This version of the federation-support is able to aggregate and exploit even resources that are not proper clouds, given that these entities provide the minimum set of functions required to be federated (WP2-PGD-06).

This software also support resource sharing among different accounts belonging to the same federation-level identity (WP2-FLA-16).

Also the security of the images executed is enhanced, indeed, the federation provides features assuring that images can not be altered (WP2-IAS-02).

3 Architecture

In this section we describe the architecture of the federation-support to be developed by the Contrail consortium.

The following Section 3.1 provides a high-level overview of the architecture and its design principles. Section 3.2 provides a structural description of the architecture, with subsection 3.2.6 summarizing the interaction and roles of the federation modules involved in SLA management.

Within Section 3.3 we provide a detailed behavioural description of the modules, i.e., how the main functions of the federation-support are achieved by means of interaction diagrams.

Finally, Section 3.4 reports how the federation-support is deployed. It presents the deployment description both of a centralized and a distributed federation access point.

3.1 Overview

The architecture we show in this and the following sections satisfies the functional and non functional requirements reported in Section 2.1.1. As it provides the federation with a user interface and essential services, we aimed for a tradeoff between the goal of achieving scalability of the federation support and the need to ease the initial development.

We designed the federation-support architecture in a way that allows to change from a centralized solution (single access point) to a distributed one (multiple access points) with minimal disruption[3].

The architecture we show is thus to be deployed at each *Federation Access Point*. Changes between the centralized case and the distributed one are confined to very few modules:

- the module encapsulating the State of the federation;
- the authentication module, that may rely on existing technology and have its own distribution/replication mechanisms.

As shown in Figure 2 (a detailed description will be given in the next section) the overall schema of our solution is composed by a top interface layer, a middle layer which contains most of the federation logic, and a bottom layer performing adaptation to different kinds of providers. A few assumption need to be stated:

- The federation maps required services/resources onto provider ones; this includes Contrail's VIN and GAFS as separate providers of connectivity and

[3]Since allocation decisions are finalized at the providers anyway, this is not a strong additional constraint on the architecture even in the centralized case.

software, whenever they are not encapsulated within a computation provider (e.g. possibly in Contrail VEP providers).

- The same holds for higher-level PaaS services and in particular for the ConPaaS services developed by Contrail. Virtual images supporting the PaaS functions will be made available by the providers to the federation users, and will be requested through the Contrail user interface[4].
- Inside the federation core, each user is handled via her identity created within the federation. Any additional identities used by providers are retrieved by-need from the User identity module (e.g. adapters can retrieve additional credentials in order to access a specific cloud or service).
- Authentication at the user interfaces is handled by the REST module, which interacts with the authentication and authorization modules.
- Policy enforcement (authentication and authorization checks) are implicit in Figure 2 and are not everywhere detailed. Instead, explicit interactions with security modules are reported where relevant, or when the subject of the interaction is triggered by an identity management operation (e.g. identity creation and management).

3.2 Structural Description

The federation acts as a bridge between users and cloud providers. The *federation-support* offers to users, in an uniform fashion, resources belonging to different cloud providers. A Contrail federation can exploit two kind providers, those based on the Contrail cloud infrastructure and the ones based on other public and commercial infrastructures. To this extent, the federation-support meets the commitments presented in Section 2, by providing ad-hoc mechanisms to adapt to both cases.

3.2.1 Layers

As shown in Figure 2 the federation architecture is composed of three layers. Every layer is in turn composed by modules, where each module addresses a well defined commitment.

The top-most layer, called *interface*, gives a view on the federation and provides proper ways to interact with the federation. The interface gathers requests from users as well as from other Contrail components that rely on the federation

[4]The ConPaaS interfaces can be integrated with the federation ones for this purpose. Note that the federation interfaces described in this deliverable do not yet explicitly support ConPaaS, but the extension will be straightforward.

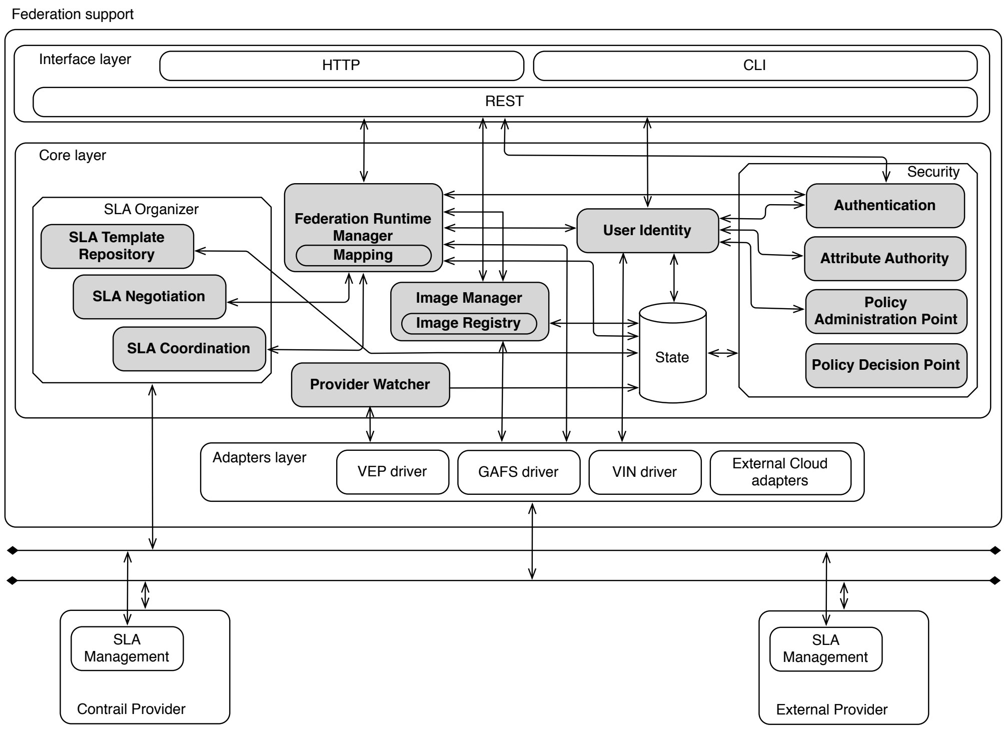


Figure 2: The structural view of the federation-support architecture

functionality and facilities. The interface layer includes a CLI and HTTP interface, from which is possible to access to REST services.

The mid layer, called *core*, contains modules that fulfill the functional (e.g. application life-cycle management) and non-functional (e.g. security) requirements of the federation. In other words, the core layer realizes the business logic of the federation-support.

The bottom layer, called *adapters*, contains the modules that retrieve information and operate on different cloud providers. This layer provides also a unified interface that possibly copes with heterogeneity of providers.

3.2.2 Core Federation Modules

The core layer contains the modules that implement the business logic of the federation. These modules solve the three main commitments demanded to the federation-support, namely identity management, application deployment and SLA coordination. These modules are in turn supported in their activities by additional auxiliary modules. In the following of this chapter we present in detail the modules that implement the business logic of the federation as well as the state module, which is in charge of the federation state management. We refer to the auxiliary modules whenever it is necessary.

State The *state* module collects, aggregates and provides information exploited by the federation-support. Items in this mass of information are subject to diverse constraints in terms of frequency, atomicity and consistency of updates, thus different architectural solutions may be needed to fulfil the federation non-functional requirements of scalability and reliability.

The involved issues become relevant when deploying the federation in a highly distributed scenario, with many federation access points. The specific purpose of the State module is to keep the core business logic of the federation unaware of the distribution aspects, only exposing the choice among different classes of data services. Each kind of information and the related constraints can be addressed by specific design patterns, whose use we will investigate further during the project.

Different federation modules ask different kinds of information to the State module.

- The User Identity module, security modules and the Federation Runtime Manager need read or write capability to access/manage user identity information and system-wide preferences;

- The Provider Watcher needs write capability to keep an up-to-date view of available resources belonging to federated and external cloud providers;

- The Provider Watcher module and the SLA Organizer gather a characterization of cloud providers, such as their geo-location, SLA templates, cost models and peculiar features;

- The Federation Runtime Manager accesses meta-data about providers (reputation, availability) and running appliances (including associated SLA).

Clearly, such an approach requires a proper distributed communication mechanism to support the flow of information among the state modules. To this end, we plan to integrate different distributed communication patterns.

Possible approaches to explore, beside centralized ones, range from publish/ subscribe techniques to epidemic information dissemination (i.e. gossip techniques), when a large amount of low coherency / slow changing information (e.g. provider history and reputation) must be spread over large federations.

The decoupling of distributed communication within the State module is also allowed by the fact that most tasks requiring atomicity and strict synchronization (e.g resource pre-reservation and commitment) have anyway to be performed at the provider level, thus simplifying the implementation of the federation state.

User identity The federation-support provides to each user a federation-level account. By using this account the user can access to all the resources owned by the federated cloud providers. In order to interact with different providers, the federation-level user account is bound with different local providers identities. The *user identity* module is in charge of realizing the aforementioned bind. The actual connection between the module and the providers is done through the Adapter layer (discussed later).

The access to resources is managed in a seamless way, i.e., once authenticated to a Contrail federation, users should not be prompted again to access federated Cloud providers (e.g. single sign-on). The local cloud identities are stored in the state module. In order to guarantee isolation and data integrity, of the user-related data, the federation-support takes advantages of the mechanisms and policies provided by the Work Package 7 dealing with Contrail security aspects. The interface with those mechanisms and policies is the security subsystem.

Federation Runtime Manager One of the core task of the federation is application deployment. This is not a trivial task, since the user will expect the federation-support to find proper mappings between submitted appliances and cloud belonging to the federation.

In order to devise a good mapping onto the compute, storage and network services of the federation, the *federation runtime manager* (FRM) uses a set of heuristic that consider different aspects, such as to minimize economical cost and to maximize performance levels. This actual task and the heuristics are implemented by the *mapping* component, while the FRM is in charge of the orchestration between the mapping component, the SLA management system and the drivers layer. In particular, the FRM is responsible of the application life cycle management.

The FRM gathers information to cover these aspects from the State module. The information is both static and dynamic. *Static* information is mainly related with general properties about cloud providers; it includes, for instance, their geographic location, their resource- and cost-models as well as the installed software. *Dynamic* information is related to the status of cloud provider resources, as well as to cloud providers as autonomous entities forming the federation. It is the kind of information obtained by monitoring resource availability either on a per cloud-provider basis or by recording and analyzing the past history of each provider with respect to violated SLA. This information can be exploited to evaluate their reliability.

Image Manager It is necessary for the federation to take into account how the images of the actual appliances are managed during the deployment phase. Indeed, moving images is a costly operation that should be avoided whenever possible (e.g. OVA approach). From the user's perspective the images can be managed in two ways: they can be packed inside an OVF archive or referenced within the OVF files by using URI.

After the SLA negotiation step (described in Figure 5), the federation needs to submit the (possibly modified) SLA and OVF to each provider (either Contrail or external); the image links are kept, or the images are downloaded and stored somewhere, updating the links to the new place. The task of deciding what is the best storage solution is carried out by the *Image Manager*. It associates metadata to the images and decides when is necessary to copy an image or when indirection can be exploited. The actual metadata are kept inside the State module; however an *Image Registry* is introduced to decouple federation code from being modified whenever State module is modified moving from the centralized scenario to the distributed one. We keep the option open to have the OVF with the original links for the first prototypes (either ordinary links or links which actually point to the GAFS, but the user has to take care of this).

Basically this component tries to place the image in the GAFS, thus generating a new URI, and uses it with providers (avoid user bottlenecks in case of repeated data transfers. (Contrail) Providers in turn can directly access the image an transfer it to the hypervisor hosting the VMs, possibly putting it into a local cache. The image cannot be cached if the user forbids it via its QoP specification (the case of images with sensitive user data). Caching at the provider beyond the immediate needs of an application implies that there is a schema to let the federation know what is cached to check its coherency. A mechanism to notify the federation about local provider image cache updates/deletion will need to be specified. We will leave detailed specification of the caching strategies for the advanced architecture.

The hosting of images within GAFS can cause a problem with non-contrail providers but the GAFS is expected to provide CDMI access, which is standard, to negotiate access protocols and possibly convert the format. The cache content at provider is kept until the application is running, and no longer. The UID generated by provider caches are unique at the provider.

Provider Watcher This component is responsible for the State update, upon receiving monitoring information from the Adapter layer. It decouples the State from doing this task leading to a more cohesive architectural design.

The Provider Watcher can receive several information types: SLA monitoring from providers, updates from providers' SLA Template Registries, SLA violations events, resource availability updates.

Not every provider will accept to publish all information flows, e.g. resource availability may well be hidden or only partially disclosed by providers. The metadata specifying what information is made available by each provider is stored in the State module.

3.2.3 SLA Organizer

The SLA Organizer is a collection of modules related to SLA management at the Federation level, which is achieved by leveraging and coordinating SLA agreements stipulated with the federated resource providers. Actual enforcement on part of the federation is also possible for providers which support limited SLA capabilities (this is easily the case for many inter-provider network links).

SLA Coordination The *SLA Coordination* module checks that running appliances comply with the user provided and agreed SLA, and plans corrective actions as needed. An appliance that does not adhere to the agreed SLA is considered a misbehaving one. Upon being notified a violation, the SLA Coordination module logs the event, evaluates the current status of all related appliances and providers, and tries to define a reconfiguration plan for the application which compensates the violation.

The Contrail SLA management system undertakes actions that may involve either a single cloud provider, or, in more complex scenarios, multiple providers and the federation-support. The latter case happens when the SLA manager of a cloud provider is unable to address the violation by itself but a coordinate action is suitable, e.g. acquisition of new resources from a provider, SLA renegotiation with multiple providers and possibly also forms of active enforcement where a best-effort SLA term was in place.

SLA Negotiation The *SLA Negotiation* is a helper module responsible of the negotiation protocols with providers. Its main purpose is to decouple the protocols for SLA (re)negotiation from the core Business logic of the Federation Runtime manager.

SLA Template Repository SLA templates published by the providers are gathered and stored in a proper *SLA Template Repository* module.

The Federation SLA Template Repository acts primarily as a cache of the Provider's SLA Template Registries, supporting scalable SLA-based queries and user interface template selection within the federation. The repository can as well holds federation-specific SLA templates not bound to any provider.

As SLA templates are not likely to change often, caching them is feasible and cheap. The issues related to updating the SLA Template Repository are managed by the Provider Watcher and State modules.

3.2.4 Security Modules

The detailed security requirements have been already defined in D7.1 "Security Requirement, Specification and Architecture for Virtual Infrastructure". We provide here only an overview of the functions which are relevant to the federation architecture. The Security Module are grouped on the right in Figure 2.

There are five components managing security in the federation architecture: *Authentication*, *Attribute Authority*, *Policy Administration Point* (PAP), *Policy Enforcement Point* (PEP) and *Policy Decision Point* (PDP).

The PEP module is not depicted in the figure, as explained later on. The other four modules directly interact with the User Identity module and the State module. The Authentication module also interacts with Federation Runtime Manager and the Interface Layer.

Authentication User authentication is a keystone in security, a requirement in order to protects data, to provide confidentiality and to to grant appropriate level of rights/permissions. Authentication is also a requirement for accounting and billing, as the usage of resources is correlate with the user's identity.

In the first implementation user identity is created and managed within the federation. Identities recognized by other providers have be disclosed to the federation, thus allowing it to act on behalf of the user (i.e. passing the credentials to the provider adapters).

Policy Enforcement Point The Policy Enforcement Points (PEPs) intercept the invocations of security relevant operations to enforce security policy. To this aim, PEPs are integrated within the architecture components that need to be monitored, i.e., the components that implement the accesses to the system resources that are considered as security relevant. As an example, this include the modules devoted to the application deployment on the federated clouds. PEPs suspend the execution of security relevant operations, communicate the access request to the PDP, and resume or skip the execution of those operations depending on the PDP response. To implement the usage control model, PEPs should also be able to intercept the termination of security relevant operations and interrupt the execution of security relevant operations while in progress when requested by the PDP.

Policy Decision Point The Policy Decision Point performs the decision process by evaluating security policies. It is called by the Policy Enforcement Points (PEPs) when users, classes of users or federation modules (on behalf of a user) need to perform security relevant actions. This activity is important in Contrail for monitoring resource usage according to the concept of Usage Control with Access Control, where the decision factors that determine whether a user holds the rights to operate on a resource are continuously evaluated while the access is in progress. The PDP, at first, gets the security policy from a repository managed by the Policy Administration Point (PAP). The PDP exploits the data in the access request, the security policy, and some other decision factors that are managed by other components of the authorization architecture, such as attributes or environmental conditions. If the PDP decides that a user is not allowed to do the action is trying to perform, the PEP interrupts that action. Actions can be basically a translation of user level functionality (e.g submit application, federation user registration, etc.), or in a more refined approach could involve the decomposition of high level actions to low level ones. For instance, a deploy action can be defined in terms of the actual steps required (e.g. negotiate SLAs with the providers identified in the mapping, contact VIN and GAFS providers for the contextualization, access - or copy - images on the providers). In the usage control model, the PDP is always active, because when required by the security policy, it continuously evaluates a set of policy rules while an access is in progress, and it could invoke the PEP to terminate this access.

Attribute Authority This entity is devoted to store and manage user attributes paired with subjects and with resources. Examples of user attributes are their roles. Hence, when the PDP needs the value of an attribute to evaluate the security policy, it invokes the AA.

Policy Administration Point The Policy Administration Point is the component that develops and manage security policies, modifying them when certain operations do happen (e.g. new users registration). It also has the task of providing the policy to the PDP when required.

3.2.5 Adapter Modules

VIN, GAFS, VEP Drivers This layer contains the modules that enable the access to infrastructural services. These include: (i) network, which is offered by the Virtual Infrastructure Network (VIN) component, (ii) storage, which is provided by Global Autonomous File System (GAFS) and (iii) computing power, which is offered by Virtual Execution Platform (VEP). These components enrich the typical cloud infrastructural services with additional features targeting federations. These services are mostly used during the provisioning step in the lifecycle of applications, that will be discussed in Section 4.8.

The VIN should provide APIs to define a virtual network among multiple virtual machines, both intra- and inter-provider. Also the VIN should provide an API to know the QoS level of an inter-provider link, and if it is possible, the proper mechanisms to enforce a given QoS. The GAFS should provide shared data space, with the possibility for an application spanning in multiple providers to access a virtual volume. Finally, the VEP provides the proper OCCI interfaces to enable access to provider resources. Its APIs include mechanisms for reservation and configuration of resources, and starting and monitoring of machines.

External Cloud Adapters In order to extend Contrail's functionality onto external clouds and at the same time to maintain modularity, it is necessary to develop the federation logic in a way that is provider-agnostic. This means that each module of the federation-support shouldn't know in advance if it is issuing command to a Contrail provider or to an external cloud. Considering the federation commitments detailed in Section 2, and in particular the operational requirements they need, the following operations can be done on a cloud provider: authentication (login and user registration), appliance deployment, SLA negotiation, SLA enforcing, and monitoring.

Following the terminology of the Federation model presented in Section 2.2, accesses to Contrail-specific clouds can be encapsulated inside the *ContrailProvider* class, via the VEP driver. Conversely, the VIN and GAFS modules offer a way to issue inter-cloud level commands and are able to talk to a specific Contrail cloud using internal mechanisms. The different approach is explainable because we need a finer control over execution commands rather than on networks or storage commands. For the same reason, there is a dedicated component performing the SLA negotiation. Commands to a given type of (non-Contrail) external cloud can be issued via a type-specific adapter, translating requests from the federation support into requests that are understood by the provider. This task is assigned to the *ExternalProvider* module of the federation Model. This module differs from the ContrailProvider, and it does not contain any driver supporting the VIN, GAFS, or VEP. Instead, an ExternalProvider exploits the interface exposed by the public cloud. In order to address the non-functional scalability requirement, we shall take care that the adapter does not become a centralization point for the Federation access point (e.g. by using a separate instance per provider, or multithreaded adapters).

One of the differences between a Contrail cloud provider and an external one is that we cannot assume the existence of any Contrail daemon continuously running inside the external cloud.

Thus, it is also necessary to develop a mechanism to handle monitoring events from the external cloud. There are two main solutions. The first one implies that, for each deployed application, a VM is coupled with it in the public cloud and forwards monitoring information to the federation support. In this solution the user is required to spend additional money to deploy an additional machine for each submitted application; the application needs to ignore the additional resources except possibly for the sake of sending monitoring information. On the other hand, detailed application-level information can be gathered and the solution does not depend on any monitoring functionality made available by the Cloud provider.

A second class of solutions has one component inside the federation that leverages the public monitoring API of the external cloud, and routes events inside the federation support. Even in this case scalability need to be taken into account, to avoid the monitoring proxy to be a bottleneck. There is no need to modify the user application, but monitoring is restricted to those events that the provider chooses to expose to the users.

3.2.6 SLA Management

In this paragraph we recap the different modules involved in SLA Management within the Contrail architecture, also addressing the need to provide a match of the SLA@SOI functions and terminology with the definitions provided in this deliverable and in Deliverable 3.2 [17]. SLA Management in Contrail federations includes several functions:

- SLA template browsing / querying
- SLA negotiation
- SLA query / management
- SLA-OVF compliance checking
- SLA-based provider lookup
- SLA splitting
- SLA enforcement / coordination
- SLA reporting / monitoring.

Other functions which in Contrail are related to SLA management are

- OVF splitting
- Appliance/Image management
- Application/Appliance provisioning.

The basis to implement most of the SLA Management functionalities is the framework provided by SLA@SOI. Several SLA@SOI components can be reused as a starting point for both Federation-level and Provider-level SLA management components. Here only Federation-level components are considered, while Provider-level components will be considered in deliverable D3.2 [17]. The following Table 3 provides a mapping between SLA Management functionalities, Contrail architecture components (as shown in Figure 2) and SLA@SOI components.

Table 3: Correspondences between SLA Management functionality and architecture components in Contrail and SLA@SOI.

Functionality	**Contrail component**	**SLA@SOI component**
SLA template browsing / querying	SLA Template Repository	SLA Template Registry
SLA negotiation	SLA Negotiation Module	Protocol Engine
SLA query / management	State Module	SLA Registry
SLA-OVF compliance checking	SLA Coordination Module	POC (to be customized)
SLA-based provider lookup	SLA Template Repository	SLA Template Registry
SLA splitting	SLA Coordination Module, Federation Runtime Manager	POC (to be customized)
SLA enforcement coordination	SLA Coordination Module	PAC (to be customized)
SLA monitoring	Provider Watcher (partly)	Monitoring Manager
SLA reporting	Federation Runtime Manager	

Some of these SLA@SOI components require customizations in order to be usable within Contrail. Two examples are the Provisioning and Adjustment Component (PAC) and Planning and Optimization Component (POC). The custom algorithms are detailed in Section 4.

Within the core of the federation architecture, the *SLA Organizer* shown in Figure 2 implements most of the listed *SLA Management* functions, directly or within it submodules. Some core functions however are implemented by the Federation Runtime Manager.

The *SLA Negotiation* module is responsible for the SLA negotiation with providers. This component implements the SLA negotiation protocol (it embeds the Protocol Engine SLA@SOI components), collaborates with the Monitoring Manager [17] to assess the SLA monitorability and stores agreed/being-agreed SLAs in the *SLA registry* (which in Contrail federations is contained in the federation state module).

The *SLA Template Repository*, already discussed in section 3.2.3, allows the users to browse Templates offered by different cloud providers and select the SLA template that most fit their needs.

The *SLA Coordination* module (called *SLA Enforcement* component in [17]) implements SLA enforcement and coordination functions. It identifies SLA violations on all the agreed SLAs and takes actions accordingly (e.g. it can de-provision resources) as explained in 3.2.3. It also contacts the Provisioning Manager (see D3.2) to provision (or de-provision) resources.

The opportunity for SLA splitting occurs when a provider is (all providers are) only able to satisfy a part of the SLA, the remaining part needing further negotiation between the Federation and another provider. Depending on the actual algorithm used to detect the situation, the splitting can be performed within the Federation Runtime Manager, or directly inside the SLA Negotiation module (see

4.5).

The *SLA Lifecycle Manager* component (see D3.2 [17]) implements *SLA-OVF compliance checking*. An SLA-OVF compliance check is required in the provisioning step to ensure that the OVF submitted by the user in the provisioning request is compliant with the previously agreed SLA. As at the federation level the Federation Runtime Manager is in charge of the lifecycle of application, it contains the SLA Lifecycle Manager.

The *Monitoring Manager* component is responsible for collecting monitoring data on the provisioned resources; this data is needed by SLA negotiation for taking internal decisions and by the Accounting manager to trace resource usage (and then support billing). We note that the Monitoring Manager does not coincide with the Provider Watcher. The latter module has a different purpose (evaluating provider behavior instead of application behaviour) and actually relies on monitoring services provided by the former one.

It is evident that the common SLA management schema derived by SLA@SOI is declined differently at the provider level and in the federation-support, the reason being that the overall action balance changes between the two levels.

The SLA Manager at provider level deals with resources directly; resource provisioning activities usually outweigh the negotiation process activities and allow relatively easy enforcement of SLA constraints.

On the other hand the federation-level SLA Management is the middleman between a SLA (to be) agreed with the final user and potentially several SLAs (to be) agreed with as many providers. The Federation does not exercise direct control on owned resources, it only performs indirect actions through other SLA Managers, operating on rented resources from several providers.

The agreement negotiation phase is like a “wooden puzzle game”, or more formally a covering problem where the Federation SLA Manager must find the best way to combine a selection of the resources to cover the SLA requirements. Optimization problems associated to covering problems have a general formulation as integer linear programming problems. As cloud and federations deal with boolean as well as quantitative constraints, our general optimization problem will likely be formulated as a mixed linear programming problem, which can be solved with tools like the GNU Linear Programming Kit [5], or by using stochastic and heuristics methods.

It shall be noted that the actual optimization criteria is itself a matter of research: while provider will usually optimize for their own profit within the constraints imposed by the agreed SLA, the federation will have to deal with a more complex situation, where a tradeoff must be achieved among the gain margin of the providers, the profit of the federation and the user satisfaction.

At execution time the federation-level SLA enforcement phase is like an equilibrium game, where the federation SLA Manager shall not violate the user constraint and not disadvantage too much any specific party in the game, to prevent that party (a resource provider or the federation itself) from profitably changing its strategy (e.g. increasing its willingness to default agreed SLAs).

This is a form of *Nash Equilibrium* between providers and federation, where at each change in the execution conditions, the optimality of the resource allocation is potentially re-evaluated (either via mixed linear programming as reported before, or through stochastic optimization tools, which allow tuning the tradeoff between approximation and computational load, like simulated annealing, genetic algorithms or ant colony methods).

3.3 Behavioural Description

In this section we present the main federation subsystems according to their behaviour.

Figure 3 shows the steps and the federation modules involved for user registration. A user connects to the federation interface entering her personal details (i.e. federation identity and password). The federation interfaces activates the identity management module. Upon a successful registration, this module creates a federation user identity and by collaborating with the AttributeAuthority a set of specific attributes is associated with a user. Then, the PAP module is contacted to store any user policies. This process can in principle lead to the redefinition of system level policies. This may not be fully realizable in an automated way, and may require manual intervention. Then the identity management module registers the user into every federated cloud provider and the created identities are stored into the State module to realize the single sign-on support.

Assuming that some sort of authentication has been done, Figure 4 shows how the federation components interact with respect to authorization of user actions. Whenever a user issues an action, the FRM retrieves from the user identity module the information related to that particular action. As an example, let us consider a deployment action where the information is related to preferences about providers. In this case, an authorization request is sent to the PDP component which indicates if the action is authorized; the decision is taken in collaboration with the attribute authority component. This last interaction is not shown in the diagram of Figure 4 because it is part of the internal security mechanisms. The PDP module registers the information about the user, the requested action, and the authorization outcome. If the user is authorized for the action, the FRM informs the user and executes the action on her behalf. This procedure is triggered for all federation-level actions.

Once registered, the user can use the federation services to run applications into the federated clouds. Each application is associated with a set of constraints and an SLA proposal. The former of these defines the minimal requirements for

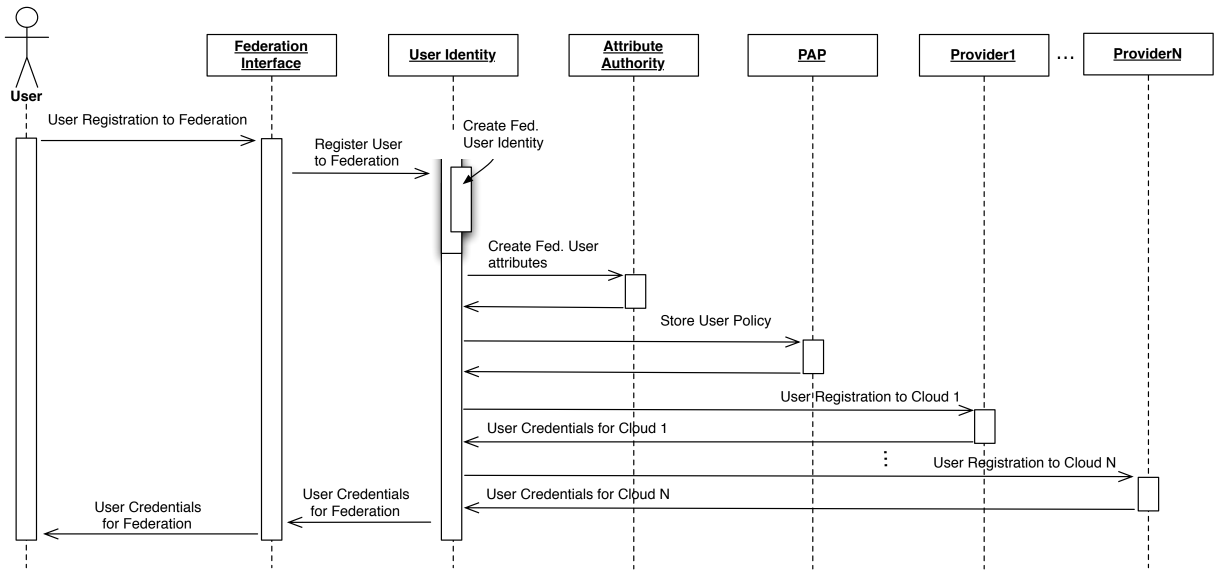

User
Federation Interface
User Identity
Attribute Authority
PAP
Provider1
...
ProviderN
User Registration to Federation
Register User to Federation
Create Fed. User Identity
Create Fed. User attributes
Store User Policy
User Registration to Cloud 1
User Credentials for Cloud 1
User Registration to Cloud N
User Credentials for Cloud N
User Credentials for Federation
User Credentials for Federation

Figure 3: Behavioral description of the User Registration process

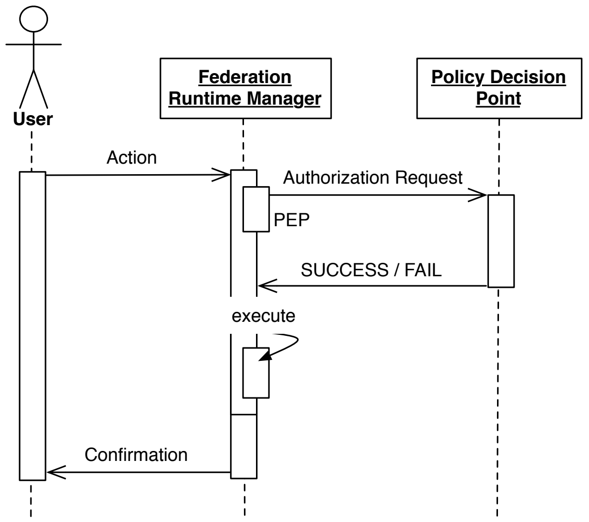


Figure 4: Interaction between the Federation Runtime Manager and the Policy Decision Point module upon federation-level action requests

the execution of the application. The latter is specified by users on the basis of her preferences and needs. The SLA proposal is created by the user from the SLA templates contained in the SLA Template Repository.

Figure 5 sketches the whole process of providers' selection and service level negotiation. The Mapping module receives from the FRM the application description and SLA proposals. According to this information the module elaborates a mapping plan, based on a predefined cost model. The plan accounts for objective functions that consider static and dynamic information about providers. The mapping plan can also exploits information coming from the Image Manager in order to discard mapping plans that imply forbidden deployment on certain providers. Subsequently, the SLA Management component uses the mapping plan and the constraints. It interacts with the federated providers in order to negotiate the necessary SLAs for the application. In this phase, the SLA can be split into multiple sub-SLAs (i.e. which target the SLA subset for a specific provider). Finally the FRM checks if SLAs adhere to the user requirements. This constitutes the necessary condition for the application decomposition.

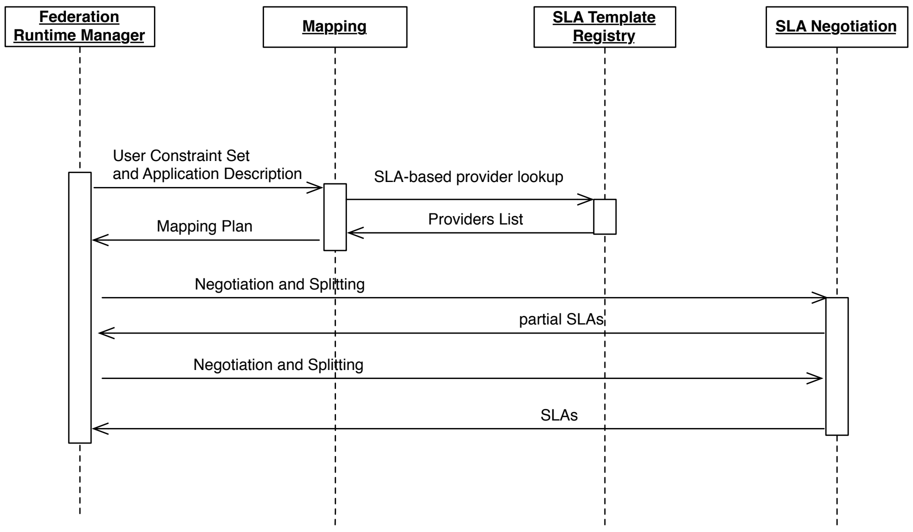

Federation Runtime Manager
Mapping
SLA Template Registry
SLA Negotiation
User Constraint Set and Application Description
SLA-based provider lookup
Providers List
Mapping Plan
Negotiation and Splitting
partial SLAs
Negotiation and Splitting
SLAs

Figure 5: Provider Selection and SLA Negotiation Process

Whenever the agreement between user and federation providers is confirmed, the negotiation phase ends and the federation starts the deployment phase. The activities performed during this phase are discussed in Section 4.8 and depicted in Figure 6 in full detail. They can be logically split into three main steps: (i) appliances transfer and contextualization, (ii) appliance execution management and (iii) appliances de-contextualization.

Transfer and contextualization In this phase the appliances are transferred to providers. Note that if the providers are aware of the appliance registry location, this is simply an indication about what appliance to use. Providers send back *deployment information*, such as the URIs of the images. This information is in turn passed to VIN and GAFS external components in order to contextualize them. Note that the order in which VIN and GAFS are contacted depends on the specific implementation choices.

Execution management In this phase, the FRM manages the execution of the appliances. When providers start the appliances, they return to the FRM an *appliance handle*, which permits to issue operations on appliances. During their execution the appliances are checked against the negotiated SLA. In case of SLA violations the SLA Organizer subsystem reacts taking proper actions. Depending on the extent of the violations these actions can regard the SLA renegotiation or the application remapping. When appliances terminate, the providers supply an *executive summary*, that the federation uses to invoke the de-contextualization.

De-contextualization After the termination of the submitted appliances, the FRM notifies GAFS and VIN in order to release the resources assigned to the execution.

3.4 Deployment Description

A UML deployment diagram describes both the distribution of a system in terms of execution nodes, and how the software components (called *software artifacts*) are placed inside the execution nodes. Here we address how different software components and modules of the Federation SW architecture are deployed on physical resources.

In fact, an execution node can also be a virtual resource in this context: nothing prevents the federation-support from running within an hypervisor. However this kind of virtual resources, if present, are statically allotted to support the federation.

The federation-support comprises two deployment packages: one for a specific provider and one for the federation access point. The federation access point package can be deployed either in a centralized way, shown in Figure 7, or in a distributed one. Only communications between the federation access point and the providers are shown in the diagrams. In relation to the description of section 3.2.5 two types of Provider are given as an example of components deployment: a ContrailProvider, that comprises the execution drivers to interface with a Contrail cloud provider, and an ExternalProvider, whose internal structure depends on the specific public cloud it interacts to. In the distributed scenario (depicted in Figure 8), it will run in possibly more than one provider. Furthermore, some Contrail clouds will have the federation access point package, while others only the provider-specific package. In our vision the only component that exchanges information among the federation access point is the *State* module.

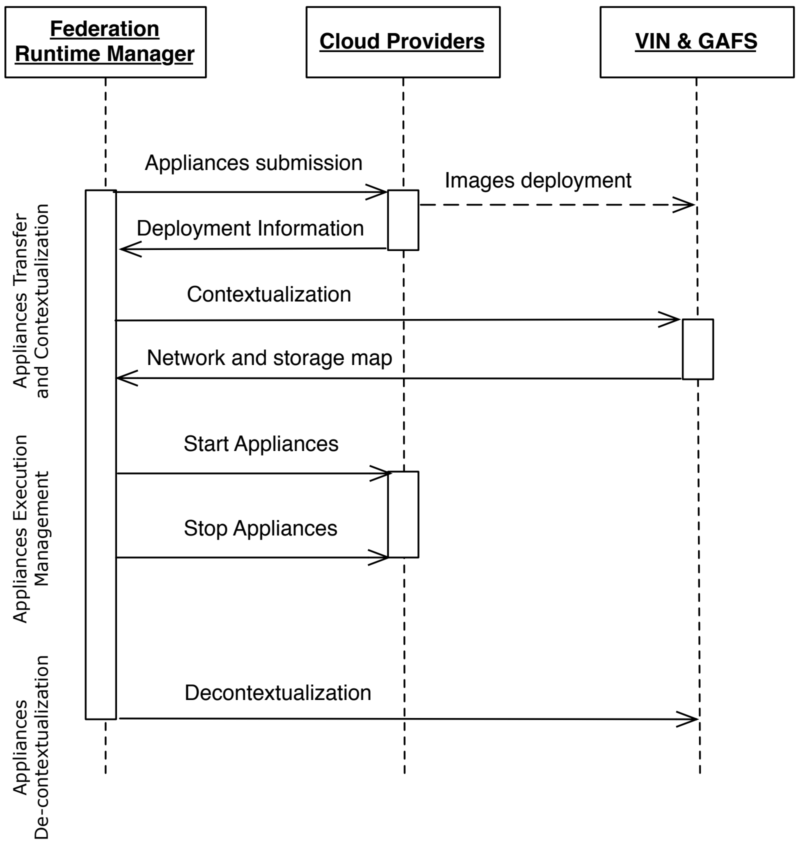


Figure 6: High-level behavioral diagram of Application Provisioning

We do not dwell here again in the functions that the state has to provide; from the point of view of the federation deployment, it suffices to know that distinct instances of the state module will provide distributed synchronization exploiting the communication patterns which best fit each specific kind of information the state can hold.

Each provider has its own adapter level. There is an ongoing integration work in order to fully describe communications (e.g specific UML stereotypes that identify communication mechanisms).

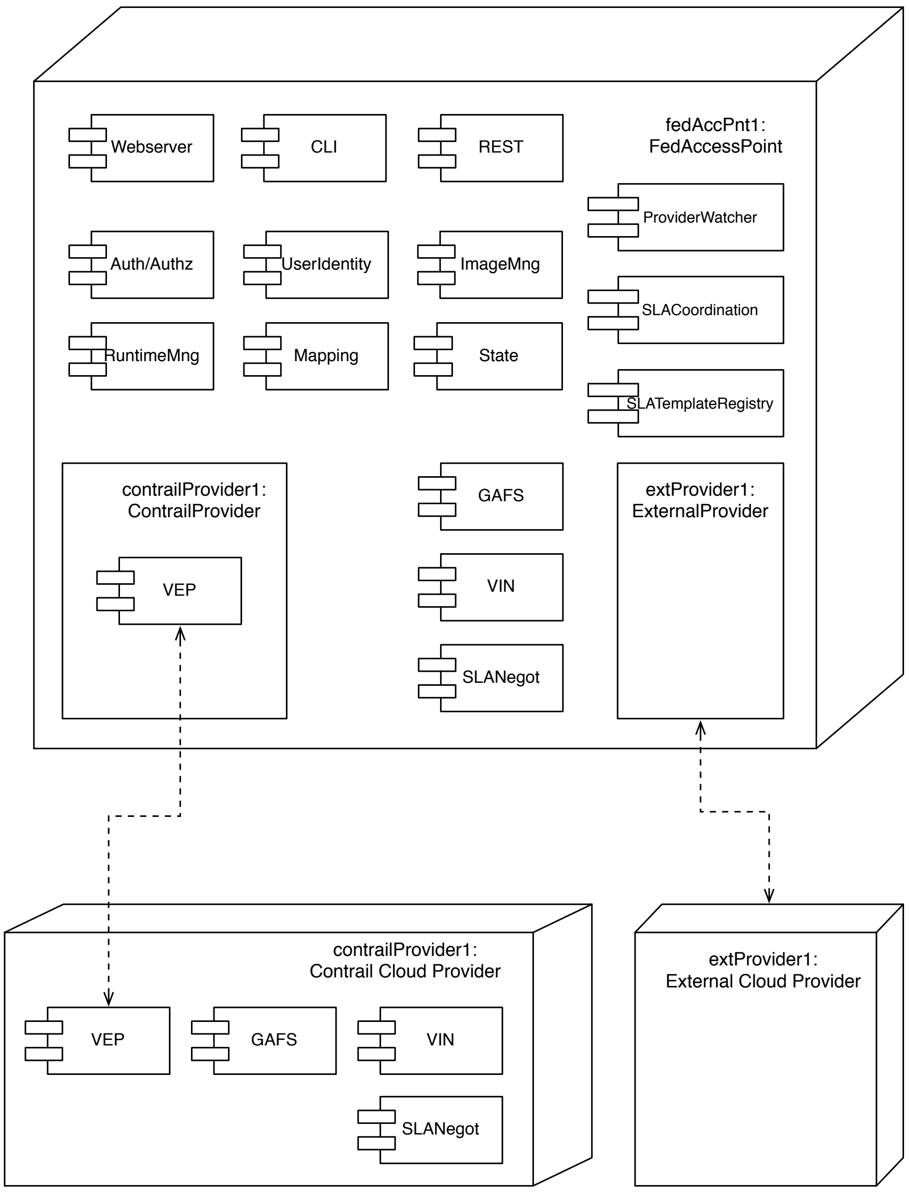


Figure 7: UML deployment diagram of the federation support in the centralized scenario

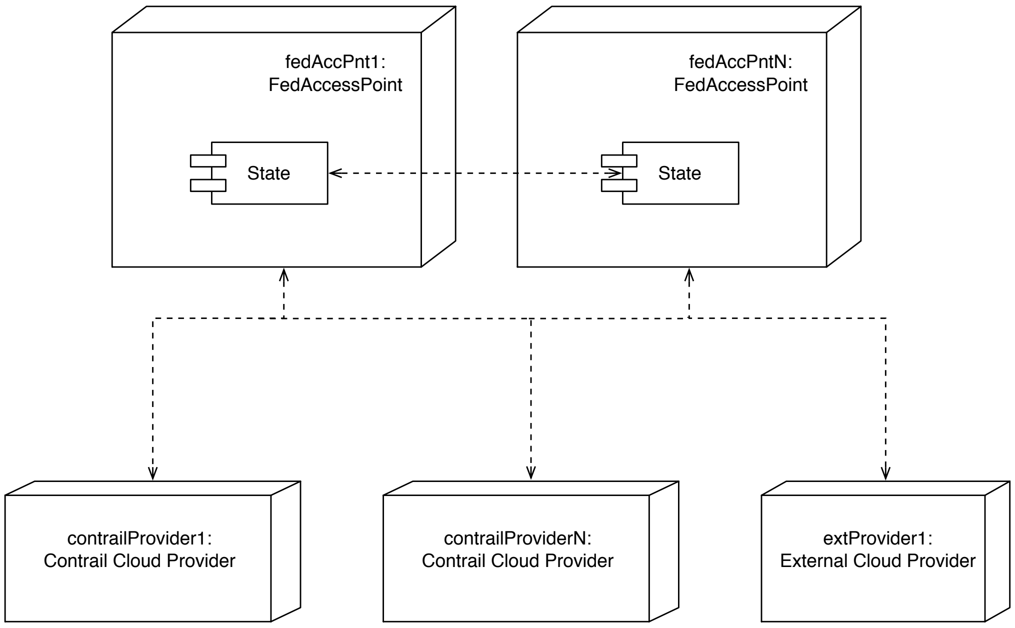


Figure 8: UML deployment diagram of the federation support in the distributed scenario

4 Algorithms

In this section we discuss what methods and what classes of algorithms we are going to exploit in the architecture modules described in Section 3. The next section 4.1 introduces the Contrail IaaS application model, which builds upon existing standards in the aim of both improving provisioning in a federated environment, and achieving portability and modularity of the application description. Section 4.2 discusses the semantic intersection between the SLA@SOI SLA language and the OVF language, and its impact on Contrail. Section 4.3 presents a simple OVF-SLA compliance checking algorithm. We then discuss how to perform SLA-based provider lookup in Contrail (Section 4.4), how exceedingly demanding applications and their SLA are split in order to leverage federated resources (Section 4.5). Then we analyze possible approaches to coordinate the resulting parts (Section 4.6). In the remaining sections we describe our approach to application mapping (Section 4.7) and the overall scheme for application provisioning (Section 4.8).

4.1 Application Model

The application model we describe in this section is the view of the application from the IaaS federation standpoint. Our purpose in explicitly defining the application model is to allow managing an application as a composition of generic *tasks* and applying mechanisms that perform the mapping between tasks and providers in a modular fashion.

While requiring the least amount of extension to accepted Cloud standards for application description, the model shall be able to express

- both requirements on each task and inter-task ones (e.g. properties of computation versus properties of interaction), and in addition
- requirements evaluated on groups of tasks versus requirements which apply the same to each single task of a group (e.g. aggregate bandwidth vs bandwidth per task).

At the IaaS level, each application *task* will be a set of one or more VMs.
Each *group* of tasks will be a set of task which share some common behaviour.

Let first recall what kind of application descriptors are submitted to a Contrail federation, before discussing how a modular application model is defined on top of them.

Open Virtualization Format (OVF) file The OVF is an XML file which describes the virtual images composing the application as well as the associated execution constraints and deployment actions. An application is composed of at least one OVF file.

In order to understand the application model, we need further details about the structure of OVF files. A single image is defined through the XML tag *VirtualSystem* and a collection of images with the tag *VirtualSystemCollection*. It is possible to nest multiple collections, thus creating a functional hierarchy in the description of the application.

SLA offers The SLA offers describe the service level constraints associated with the application. They are compiled by the user, and ideally derived from the SLA templates that the providers make available to users. Parts of an SLA refer to resources by providing their ID, and each ID can either be a full URI of a resource within a file, or a path which is relative to a file elsewhere specified.

Deployment Document (DD) is an optional document which supports non-standard features and additional functionalities when deploying over contrail providers. The DD can act as an additional link between OVF and SLA (see deliverable D10.1).

4.1.1 Abstract Task Interaction Graph

A Task Interaction Graph (TIG) [22] is a well known structure to represent requirements over networks of tasks. A TIG represents the application as an undirected graph: each node a task and edges the relationships between two tasks. A TIG is labeled on nodes and edges. In our case the labels describe the constraints and needs of the application. We do not follow the straightforward approach of building a TIG where each virtual image is a task. Such detailed representation leads to a large graph, hindering the process of mapping the application to actual resources and most likely not increasing the quality of the mapping.

As the reference model of Applications we define an *Abstract TIG* exploiting the grouping of resources provided by the OVF entity *Virtual System Collection*. In Contrail any service can be a task, e.g. access to storage is a legitimate task. From the point of view of the Abstract TIG, only the VIN is a special kind of provider, as the related constraints result in labels on edges of the TIG rather than on nodes.

We call *Mapping* of the TIG of an application the operation of identifying which sets of resources (of any kind, from one or more providers) will be used to match the application services at provisioning in order to satisfy all constraints and the SLA.

We assign to OVF collections, as well as to single images, a *Resource Class* (RC) which identifies the functional role of images within the application. The resource class is in fact the *id* attribute of the collection, so that the base case of RC is the resource itself. The user can then associate one or more SLA offers to each RC (the SLA will reference the RC within the OVF instead of the resource). This is compatible with the plain use of SLAs to specify the behaviour of each single needed resource, but allows to specify a higher-level SLA for groups of resources. We will call this group of resources *Application Components* [5] (see also section 4.6). In addition to ordinary constraints, we can place also aggregate constraints[6] on Application Components for instance addressing mutual interconnection and elasticity. As an entity id is unique within a same file, but not globally unique, the same resource class can also be reused consistently across different OVF files (see *generic SLAs* in deliverable D3.2), allowing for agreed SLAs to be reused for different OVF files sharing a common structure.

Example Figure 9 shows the OVF of a very simple application, composed by a database and two web servers (a plain HTTP server and an HTTPS one). The

[5]The term “component” is used in its more generic sense, not implying the existence of *SW components* in the application.

[6]See also Deliverable 3.2 for a list of SLA terms and for the definition of *generic*, *specific* and *hybrid* SLA offers

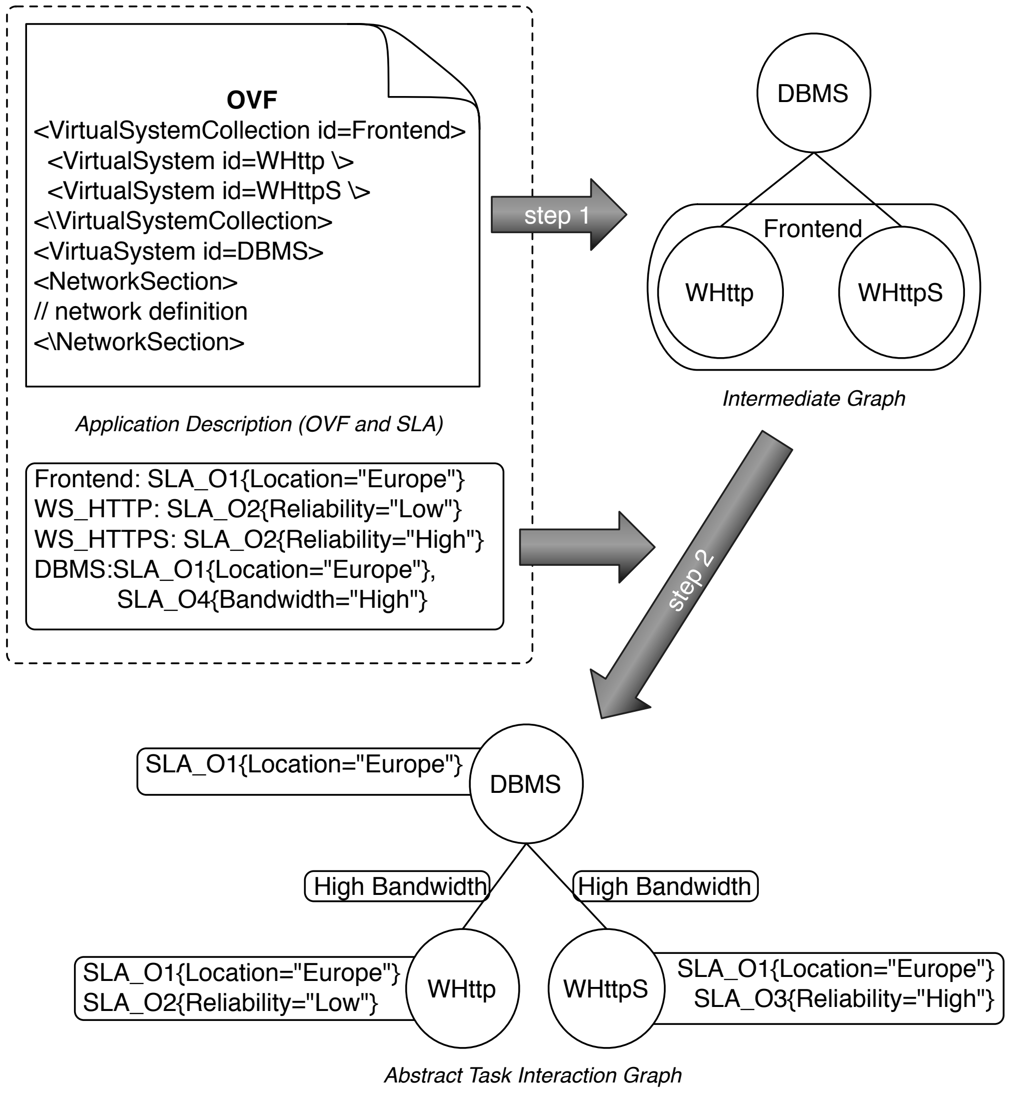

OVF
<VirtualSystemCollection id=Frontend>
<VirtualSystem id=WHttp \>
<VirtualSystem id=WHttpS \>
<\VirtualSystemCollection>
<VirtuaSystem id=DBMS>
<NetworkSection>
// network definition
<\NetworkSection>
Application Description (OVF and SLA)
Frontend: SLA_O1{Location="Europe"}
WS_HTTP: SLA_O2{Reliability="Low"}
WS_HTTPS: SLA_O2{Reliability="High"}
DBMS:SLA_O1{Location="Europe"}, SLA_O4{Bandwidth="High"}
step 1
DBMS
Frontend
WHttp
WHttpS
Intermediate Graph
step 2
SLA_O1{Location="Europe"}
DBMS
High Bandwidth
High Bandwidth
SLA_O1{Location="Europe"}
SLA_O2{Reliability="Low"}
WHttp
WHttpS
SLA_O1{Location="Europe"}
SLA_O3{Reliability="High"}
Abstract Task Interaction Graph

Figure 9: Steps to translate an OVF into a TIG

XML attribute “id” in the OVF provides each entity with a Resource Class (RC). In the example we have *Frontend*, which represents the generic set of web servers, *WS-HTTP* and *WS-HTTPS*, which differentiate the two types of web servers; *DBMS*, which represents the database server. The *NetworkSection* tag defines the network that connects the database with the web servers.

The first logical step is to materialize the hierarchy of the OVF file as an intermediate graph (step 1 in Figure 9). The intermediate graph describes the connections among tasks and maintains the hierarchical relationships from the OVF file, e.g. both web servers belong to the RC "Frontend".

The second step transforms the intermediate graph in the final application model, mapping each RC into its associated SLA offers (SLA_O1, SLA_O2 and so on, in the figure). Note that: (i) a single RC can be associated with multiple SLA offers (DBMS is associated with SLAO1 and SLAO4) and (ii) a single SLA offer can be shared among multiple RCs (DMBS and Frontend share SLAO1).

The resulting application model is a flat graph, where the same SLA offers are replicated on the associated tasks. Tasks only need to keep their basic identity in order to allow provisioning, and mapping small flat graphs onto resources is easy enough not to be a bottleneck. Independent negotiation for each task node is possible if needed, which can positively affect the mapping process. The final graph represents the input for the application mapping phase, which is described in Section 4.7.

Pragmatics Our application model exploits collections as containers which convey information provided by the application programmer. This allows to deal with large applications in terms of their (few) functional components, matching the application semantics on top of the aggregate providers' available resources. As a basic case, we have unstructured application, as well as applications which provide only trivial structure. Contrail can handle these "flat" applications and their one-node TIG, but when multi-provider mapping is needed not all available mapping heuristics may have enough information to work.

Implementation issues From the point of view of the TIG application model, the DD is just an additional indirection in the link between the SLA and the OVF. For compatibility with less expressive SLA models than the SLA@SOI one, we can exploit the DD to link the SLA offers with the resource classes.

Resource classes (and hence, SLA offers) are hierarchical: in order to successfully deploy an image, all the SLA offers belonging to the parent collections have to be satisfied. The nesting level of OVF collections is unlimited, but we only need to exploit containers as classes of resources on which to impose SLA terms. As of now, 1 or 2 additional nesting levels are needed by application as class handlers.

When dealing with application made up of multiple OVF files, we can regard the OVF files as an additional level of containers, as SLAs can obviously reference OVF files.

4.2 OVF generation from SLA / SLA generation from OVF

SLAs are the output of the negotiation phase and describe offered resources and guarantees about them. OVF (Open Virtualization Format) is a standard format used to describe multiple virtual resources, their properties and their connections. OVF is used as input to the Contrail provisioning phase (see D10.1).

A SLA should contain both a service description and guarantees about the service. In particular for IaaS services the SLA usually contains the description of the offered resources, such as VMs, RAM, number of cores. The OVF format, in particular the OVF descriptor, also describes virtual resources. As it is shown in Figure 10 there is a semantic overlap between OVF and SLA.

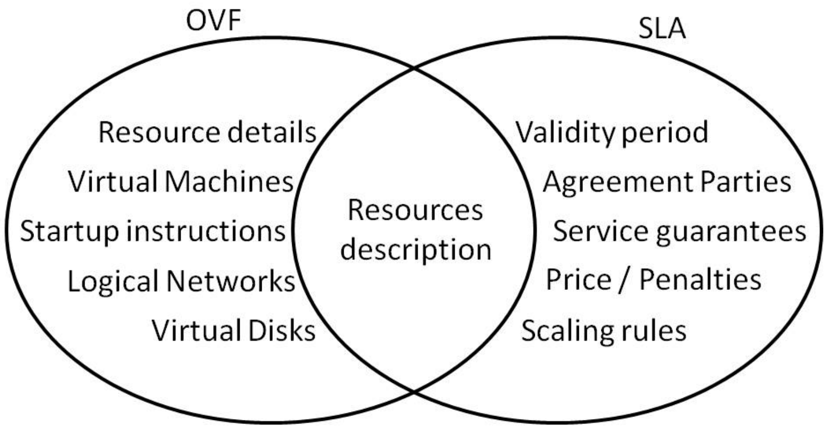


Figure 10: Semantic overlap between OVF and SLA

This overlap cannot be easily solved by only inserting resource information in the OVF and putting references to them in the SLA. This would prevent the possibility to negotiate SLAs independently from the OVFs to be provisioned. Contrail will maintain this independence between SLA and OVF at the cost of replicating some resource information in both formats.

The need arises of handling independent input file in the SLA and OVF formats, and to merge the provided information at some point.

Before provisioning, compliance shall be checked between a given OVF file (set of files) to be provisioned and the agreed SLA, especially when a generic SLA has been agreed, and the OVF file was submitted independently. The algorithm to perform this check will be described in the following section 4.3.

Another functionality which may be needed is the ability to transform part of the SLA specification into OVF and vice versa. This transformation is not possible in either direction for a complete document, it can only apply to the intersection of the semantic domains of the SLA and OVF formats (see Figure 10). The transformation between OVF and SLA is practically possible for the high-level description of attributes of virtual resources, such as number of VMs, number of cores, CPU speed or amount of memory.

In the SLA@SOI SLA model [24] the attributes of virtual resources can be expressed each with a separate `Guaranteed.State` clause. For example, to define the amount of memory for virtual machines the following SLA expression can be used:

```
MEMORY_VALUE is ( less_than_or_equals( 10240 MB ) and
                         greater_than( 1024 MB ) )  << 2048 MB >>
guaranteed_state{
        id = MEMORY_STATE
        memory( VM_X ) equals MEMORY_VALUE
        // means: 1024<memory(VM_X)<=10240, default=2048 MB
}
```

The equivalent definition in the OVF format is the following:

```
<VirtualHardwareSection>
   <Info>...</Info>
   <Item>
      <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
      <rasd:ElementName>2048 MB memory size</rasd:ElementName>
      <rasd:InstanceID>0</rasd:InstanceID>
      <rasd:ResourceType>4</rasd:ResourceType>
      <rasd:VirtualQuantity>2048</rasd:VirtualQuantity>
   </Item>
   <Item ovf:bound="min">
      <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
      <rasd:ElementName>1024 MB minimum memory size</rasd:ElementName>
      <rasd:InstanceID>0</rasd:InstanceID>
      <rasd:Reservation>1024</rasd:Reservation>
      <rasd:ResourceType>4</rasd:ResourceType>
   </Item>
   <Item ovf:bound="max">
      <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
      <rasd:ElementName>10240 MB maximum memory size</rasd:ElementName>
      <rasd:InstanceID>0</rasd:InstanceID>
      <rasd:Reservation>10240</rasd:Reservation>
      <rasd:ResourceType>4</rasd:ResourceType>
   </Item>
</VirtualHardwareSection>
```

In this case, a key OVF element is the optional `ovf:bound` attribute, which may be used to specify ranges for the `Item` elements of an OVF descriptor. In the same way the same definition of other virtual machine attributes can be expressed either in SLA or in OVF format.

The algorithm that operates this translation is very simple and can reuse the SLA@SOI code at least in the SLA-to-OVF direction. As for the OVF-to-SLA direction, the input format (OVF) is an XML file, thus standard XML parsers (such as Xerces [26]) can be used. The algorithm gets a full SLA (or OVF) file in input and produces OVF (or SLA) fragments in output describing the entities belonging to the intersection of the two semantic domains. The first step is to parse the input format, creating an object for each element, according to the abstract model that represents the input format. The abstract model of SLA@SOI SLAs is described in [24]. The OVF abstract model is represented by its schema definition (XSD) file, see [14]. The second step is to extract entities of the input model which are related to the intersection of OVF and SLA domains and map them to entities of a common semantic model which represents objects of the underlying virtual resources domain. For example a VM in this common model can be represented as shown in Figure 11.

The last step is to generate the required output syntax from the common model. Each object in the common model will have two output methods generating its representation in either OVF or SLA syntax.

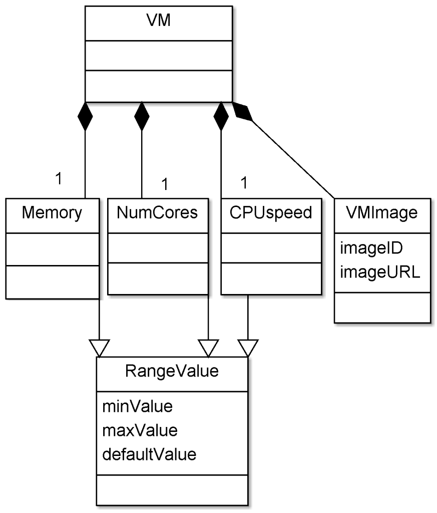


Figure 11: VM representation in the virtual resources abstract model

4.3 Checking SLA-OVF compliance

When the user submits to the Federation an application description (as an OVF file) for provisioning, he is supposed to have already negotiated the associated SLA with the Federation. As there is a semantic overlapping between OVF and SLA resource specification, a check is needed to ensure that the two descriptions do not conflict. In addition, depending on the SLA description used and on additional features, an indirection via the

Deployment Document (DD) may be needed. There are two places where the check will happen.

- The Federation will first check the OVF compliance with the agreed SLA (*external SLA*, see sec. 2.2), i.e. that (1) what is required by the OVF is covered by the agreed SLA and (2) the resource classes referenced in the SLA are found in the OVF. If the OVF requires resources that do not match those agreed in the SLA (some resource is not covered by the SLA, or the resource attributes are incompatible), then the provisioning request must fail.

  It is considered a user error to agree on some resources and then submit an OVF which does not match the agreement. The provisioning request must fail because in this case the Federation in general doesn't know about resource availability, if and how it can be monitored, and a price for the resource has not been agreed.

- A second OVF-SLA compliance check is operated by each federated Provider when it receives an OVF to be provisioned.

  The Provider will receive the OVF from the Federation on behalf of a user. It will check it for compliance with the *internal SLA* previously agreed between the federation and the provider for that user.

  As the application may have been decomposed onto several providers, the check ensures to the provider that no error happened in the splitting phase.

The algorithm to operate this OVF-SLA compliance check will first simplify both the SLA and the OVF in input eliminating all the clauses not related to the intersection of the OVF and SLA semantic domains (see Figure 10). After this simplification operation (indicated as *restrict_to_intersection()*), the algorithm will generate SLA fragments from the given OVF (see previous section) and compare those fragments with the agreed SLA.

When the SLA specifies a range for a parameter, e.g. min and max value for the amount of memory, the corresponding value (or range) requested by the OVF must be contained within the SLA-specified range.

When the SLA specifies an exact value for a parameter, the OVF value must be the same (if the parameter is optional, the OVF may left it unspecified).

When a parameter requested by the OVF is not specified in the SLA, and the parameter is about an entity belonging to the intersection of the OVF and SLA semantic domains, the compliance check will fail.

The pseudo-code of the basic algorithm follows.

```
function checkCompliance( O: OVF, S: SLA ): bool
begin
   O1 = restrict_to_intersection( O )
   S1 = generate_SLA( O1 )
   S2 = restrict_to_intersection( S )
   return compare( S1, S2 )
```

```
end

function compare( OvfSLA: SLA, AgreedSLA: SLA ): bool
begin
   foreach resource in OvfSLA
      if resource not in AgreedSLA
         then return FAILURE
      foreach parameter of resource
         if parameter.value not in range specified by AgreedSLA
            then return FAILURE
   return SUCCESS
end
```

If a DD has been provided, the *compare* function will have it as an additional parameter and use it to retrieve the resources in the OVF.

If a DD is not provided in input (the SLA directly refers to OVF resource classes), but it needs to be specified to the following provisioning phases, the compare function can write it out in its main cycle.

It should be noted that a successful compliance check only guarantees that the requests of the SLA and OVF are not conflicting, but does not ensure that the provisioning will be successful. The case may be that specific requests are in the OVF which were not negotiated in the SLA, and prevent provisioning. In general, provisioning failures after a successful compliance check will be related to entities which do not belong to the intersection of the OVF-SLA semantic domains. The only solution to the problem in the Contrail framework is to extend the SLA language to include terms that describe the useful concepts at negotiation time.

Example The OVF descriptor contains very detailed constraints on specific hardware to be used for provisioning a virtual machine, and the selected provider does not have that specific hardware in its data center. Even if the OVF is compliant with the SLA agreed with the provider, provisioning will still fail if the required resources are not available, and the OVF does not allow for any replacement.

4.4 SLA-based provider lookup

SLA-based Provider Lookup (from now on SPL) is a functionality which is exploited at Federation level during the user negotiation phase.

The Federation layer will select the set of providers with which to setup a SLA on behalf of the user, according to user preferences and applying its own heuristics. The guarantees required by the user are thus checked against the SLA templates offered by each federated Provider, in order to shortlist the candidate Providers to be considered in the SLA negotiation phase.

The functionality is also exploited when the Federation has to split a SLA to find providers that satisfy each SLA fragment, and when cloudbursting is needed and compatible providers must be found.

Each Contrail Provider maintains a SLA Template Registry, which will be based on the SLA@SOI SLA Template Registry. The component from SLA@SOI should already support "both metadata-based queries, and QoS-based queries" [19], so the simplest and more accurate algorithm for SPL will loop on all the Providers executing the desired query on each registry.

As looping on all providers from all federation access points is neither efficient nor scalable, a centralized SLA Template Repository is kept at Federation level: in this way only one query is enough, even if on a quite big registry. The Federation SLA Template registry acts as a cache, allowing to shortlist the available providers with a single query[7]. This works on the assumption that SLA templates are not likely to change often, so caching them is feasible and cheap.

The Federation SLA Template Repository can as well hold SLA templates not bound to any provider, i.e. prepared by the federation coordinators. These templates of course will be designed in order to match the capabilities of at least one provider.

In a distributed federation scenario, each federation access point keeps a copy of the SLA Template Repository, which is periodically synchronized with the other registries. Federation nodes are connected one to each other via the Status module[8], whom the synchronization is delegated to.

The current implementation of SLA@SOI for the SLA Template registry allows both metadata-based queries, where metadata for a SLA template comprises a list of key/value property pairs, and more general SLA-based queries. Even to use only metadata-based queries would not be a big limitation as many properties of an SLA can be expressed as key/value pairs.

4.5 SLA splitting

Normally the Federation will search the best provider to satisfy all user requirements (see previous section) and will negotiate a SLA on behalf of the user with that provider. Whenever a single provider cannot satisfy all the execution, quality and/or security requirements, the Contrail federation will need to split the application and deploy it as an ensemble of cooperating separate parts on different providers.

In order to still leverage the provider SLA management mechanisms, the federation will also have to correspondingly split the proposed SLA. While this will not be the most common case, it shows the federation's added value with respect to single Cloud

[7]The federation-level centralized SLA Template Repository plays the same role also in the template selection before the negotiation phase, as user interfaces also query federation templates.

[8]Depending on the federation size and on the kind of data to broadcast, the status can implement various centralized and distributed synchronization mechanisms for different classes of data, possibly even including gossip protocols. On the other hand, gossip is unlikely to be used to spread critical information like SLA templates.

providers, that is the ability to deploy applications with peculiar requirements and beyond the limits of available resources on single providers.

Splitting of SLAs can happen in different moments of the application deployment, and with different approaches. We consider two different actors in performing the SLA splitting, the federation support and any service provider, which will typically correspond to specific phases in the deployment, i.e. user negotiation and provider negotiation. In the first case we have a *federation-initiated SLA splitting*, while in the second case we speak of *provider-initiated SLA splitting*. With respect to the split technique, three main types of SLA splitting will be herein considered.

Service-based SLA splitting when the SLA is split in parts related to different kind of services or resources. An example is to split the overall SLA into aspects like computation services, permanent storage services and inter-provider network services, then to select distinct providers for each aspect.

Availability-based SLA splitting or *Resource-based SLA splitting* when the SLA is split in order to recruit the same kind of service/resource from more than one provider. The purpose can be to increase the availability of that resource beyond the possibilities of a single provider.

Performance-based SLA splitting is a more general case, where the application and SLA split is performed in order to improve performance and/or QoS (e.g. to provide a feature similar to Amazon's Availability Zones), and not because of any negotiation failures.

It is clearly a form of splitting which is only federation-initiated, that requires more complex evaluation and search heuristics. It is thus set as a separate case where a considerable research effort will be needed.

Provider-initiated Splitting The easiest case to specify is when the split of an application SLA happens during the negotiation as a provider A is not able to accept the full set of conditions in the SLA offer proposed by the federation.

The provider, as a negotiation step, can return only the part on which it can agree (see Figure 12), thus defining an SLA split.

If the split is accepted by the federation, in order to exploit those services which are available at A the Federation will then need to generate a new SLA with the remaining conditions and relevant to the remaining part of the application; then a further SLA-based lookup and negotiation will be performed trying to match the unsatisfied part of the SLA with a second provider B.

We assume that a Contrail provider will be able to perform a negotiation and propose a reduced SLA as a counteroffer. For external providers, without the capability of negotiating all parameters of an SLA, to provide equivalent functionality for deducing a counteroffer (a partial SLA) may not be possible, it may require additional support, or it may be replaced by federation-initiated splitting.

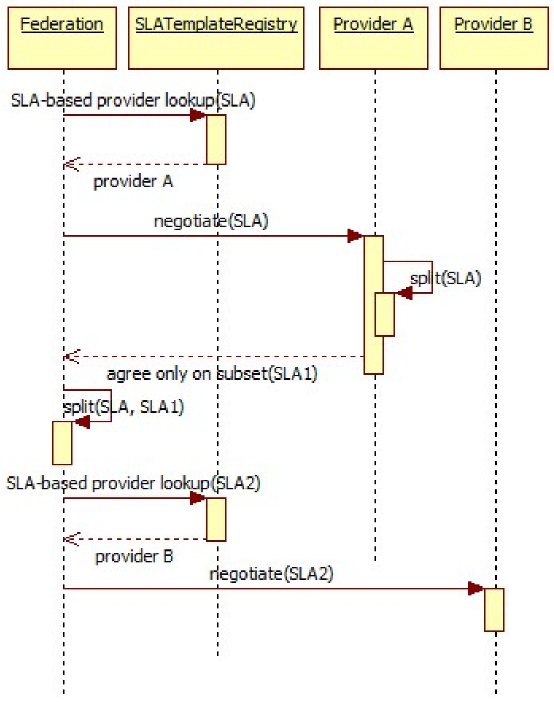


Figure 12: SLA splitting during negotiation

Splitting will usually be done in the initial negotiation phase, as seen above, but it can be done also in other phases.

SLA splitting during provisioning can happen if the provider initially selected cannot guarantee all the requested resources. Depending on the actually agreed SLA, this may either already be a violation, or just be the result of the provider not being able to renegotiate an SLA for increased resources. The federation will try to avoid defaulting the SLA with the user, and also avoid restarting the negotiation from scratch (which implies reporting to the user that the agreed SLA is no longer enforceable). The provisioning can be fixed by splitting the application and its SLA to exploit other providers. One part can be left on the same provider, and another one will be negotiated with one (or more) further provider(s), as shown in Figure 13. Note that in this case the federation may prefer renegotiating with another provider already involved in the application (e.g. provider B), as it already had the application information and could possibly deploy the needed resources quickly[9].

There are finally multiple cases when SLA splitting must occur even during execution. Examples are SLA violations, some resources moved from a provider to another, a scaling request that hits the upper resource limit of the current provider. Again, these cases present

[9]The net effect of this late SLA split can be seen as a form of work balancing by migration.

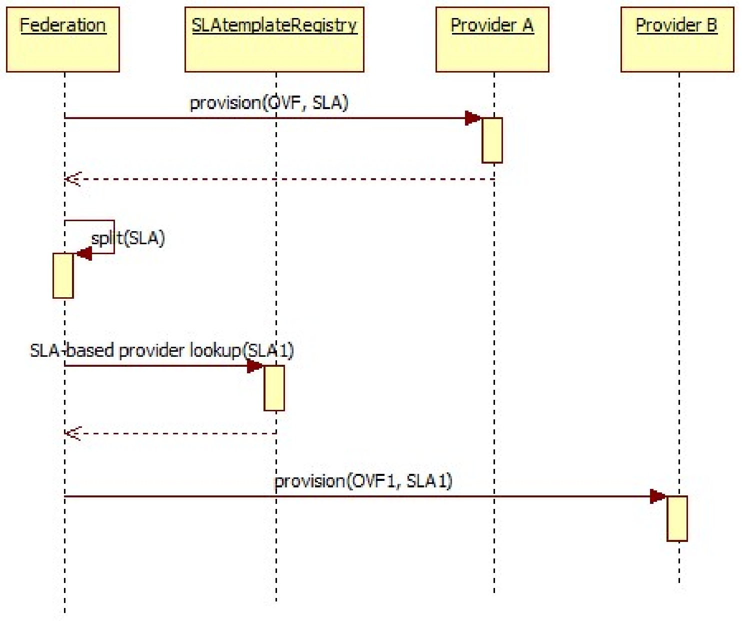


Figure 13: SLA splitting during provisioning

the same federation goal of not letting the SLA stipulated with the user fail, and can be dealt with by the technique we already discussed.

Federation-initiated Splitting As the federation maintains up-to date information about part or all its providers, the initial step of proposing the full SLA to a single provider can or must be skipped altogether, and the federation is in charge of (at least) the initial slitting of application and SLA. Concrete cases are related to the federation support having very precise knowledge of the kind of services each provider can offer (e.g. resources and appliances), up-to-date copies of the SLA Template registry of providers, and for part of the providers also having access to resource/service utilization and availability data. The subcases can be classified in three groups, to whom we can apply the same techniques as in provider-initiated splitting, except that the federation will have to consider splitting the application in terms of its components (groups of appliances).

- The *kind* of services in the SLA offer are not all available in a same provider, due to the constraint expressed in the SLA and/or to limited offer by the providers. This case leads to a service-based splitting, which is performed easily by the federation.
- The *amount* of some required resource *is known* to never be available at any provider, due to knowledge about the provider limits and SLA Templates, or to information about the provider status. Here the split is more complex, falls in the category of availability-based splitting and can require splitting groups of appliances.

- The *amount* of some required resource *is expected* as unavailable at any provider, e.g. by inference from provider negotiation histories. This is a variant of previous case, where the federation cannot rule out that a single provider is found, but most likely may save negotiation effort and time by choosing multi-provider provisioning right from the start. This case too can be dealt with with availability-based splitting.

SLA splitting and OVF splitting Splitting an application in order to deploy it on multiple providers has of course an impact also on the application representation. The method outlined in section 4.1 exploiting OVF containers allows to identify corresponding portions of the OVF and SLA. It is thus possible to apply two solutions.

- Define a separate Deployment Document (DD) for each portion of the application, specifying which parts of the OVF have to be enacted by each **Contrail provider**. This way the complexity of the SLA splitting process does not imply OVF modifications. Maximum flexibility is retained in case of changes after initial deployment (e.g. elasticity).

- The application (set of) OVF file(s) can be rewritten for each provider in their specific deployment format, only including the parts needed on the particular provider. This management overhead is anyway needed for **external providers** who do not accept a DD and possibly do not recognize the OVF format.

4.5.1 Service-based SLA splitting

Service-based SLA splitting can be done when the same SLA offer includes multiple service requests, which are not all available or accepted by a same provider. This type of splitting can generate a maximum number of SLAs which equals the number of types of service (n_{serv}) in the original SLA offer. If the number of SLA to be generated (n_{SLA}) is less than n_{serv}, there is potentially one solution for each different partitioning of the set of n_{serv} services into n_{SLA} non-empty subsets. The number of solutions is a Stirling number of the second kind [7, page 244], indicated with $\left\{ {n_{serv} \atop n_{SLA}} \right\}$ and computed as

$$\left\{ \begin{matrix} n_{serv} \\ n_{SLA} \end{matrix} \right\} = \frac{1}{n_{serv}!} \sum_{j=0}^{n_{SLA}} (-1)^j \binom{n_{SLA}}{j} (n_{SLA} - j)^{n_{serv}}$$

The actual set of potential solution will be smaller than that if not all providers support all services. Selection of one solution from the set of potential ones should be done by optimizing over indicators such as total price, QoS offered, and even the reputation of the involved providers.

Searching or pruning the set of solutions requires evaluating the potential cost of each sub-solution (either via models or by attempting a negotiation), in order to evaluate the interesting indicators, and only then combining the most promising sub-solutions.

As it is already evident from this first simple case, the selection of the best splitting for a given SLA is a multi-objective optimization problem. For a small number of alternatives a direct comparison can be done to find the optimum solution, where each single solution can be computed as outlined in section 3.2.6. For a greater number of combinations, the overall formulation as a mixed linear programming problem is likely to become too complex, and full exploration of the solution space for the splitting is not practical, so a metaheuristics approach has to be used on top of the basic problem formulation.

Several integer-problem metaheuristics from operation research can be applied, such as Ant Colony Optimization[2], Simulated Annealing[8], Tabu Search[4], or Genetic Algorithms[6]. As of today, it is not possible to specify which of the described method will suit best to the specific combination of variables and shape of the solution space presented by the problem of SLA splitting in federations.

4.5.2 Availability-Based SLA Splitting

Availability-Based SLA Splitting or *resource-based SLA splitting*, addresses the case where provider A can offer only m of all the n resources required, the remaining $(n - m)$ resources are asked to provider B.

This splitting case focuses on a single kind of service, namely a resource-based service, although it can be applied repeatedly to different groups of resources in the application (thus nesting it within Service-based splitting).

As an example, if provider A in figure 12 offers all the needed services in the SLA offer, except for one which does not get enough resources, with availability-base splitting only that service is affected by the splitting, the other ones being simply copied in SLA which is to be agreed with provider A.

Clearly, if the remaining amount of resources can not be negotiated with just one additional provider, dealing with availability based splitting can become complex, due to the need to monitor and enforce several separate SLAs in order to have the user-agreed SLA satisfied.

As such, availability-based splitting on more than two providers may be unavoidable if provider-initiated, but it is managed differently in federation-initiated splitting. Contrail federation by default should not explore solutions relying on many providers (thus also reducing the solution space) unless a specific support pattern exists for very large resource groups, with dedicated support and special mapping heuristics. As an example, ConPaas and other combinations of PaaS services may be able to exploit the composed IaaS platform by replicating their control part on each provider).

In the 2-provider case, the original SLA is reduced during negotiation to n instances of the specific resource, while the second part of the SLA just contains terms describing the missing $(n - m)$, and inherits all the other components from the first one.

If otherwise the split is iterated, then the number x_i of resources asked to the i-th provider is a variable influencing the total price and quality of the solution. A form of weighting shall be applied at mapping time to mark them as suboptimal, and a tolerance should be added to the SLA portions to take into account for the higher risk of violation due to resource dispersion.

Again, finding the best split requires solving a multi-objective optimization problem.

In addition to respecting all mandatory deployment constraint, the right value of the various x_i is evaluated in terms of the overall price of the solution, of the QoS provided, and taking into account the reputation of providers (e.g. a larger tolerance is needed for unreliable providers). In practice, the most general form of service split due to availability brings in issues which are detailed in the most general case of performance-based splitting, in the next section.

4.5.3 Performance-based SLA Splitting

The last type of splitting considered here is the *Performance-based SLA splitting*. This is conceptually the same as resource-based SLA splitting, but it uses (models of) the performance of the services in the application as criteria for splitting.

For example when service S hosted by provider A can offer only x transactions per second instead of the n required y the SLA to satisfy the expected traffic, the remaining $(n - x)$ transactions per second may be obtained by provider B.

This type of splitting is the most difficult to implement, having to cope with several issues.

- Service S must be offered by all the target providers. For this reason, SPL must handle interface-based queries, allowing the Federation to look for providers alternative to A with a template registry query constrained to the interface of S.
- If the image for service S is available for distribution, and the Federation can upload it to other providers, each new provider may be able to agree only on service hosting guarantees, and not on performance ones, as the service will be seen as foreign.
- Service entry points should be common to all target providers and should provide load balancing across several providers.
- Even more difficult to handle is the case where the service S is not stateless and needs to operate on shared data. In this case the Federation, along with splitting the original SLA and negotiating the resulting parts, shall handle the setup of all the needed shared resources, both in the storage and in the networking domain (e.g. provide connectivity guarantees that are compatible with the application synchronization needs).

4.5.4 Further issues of SLA splitting

SLA splitting has several other aspects to be analyzed, as the original SLA may not contain only guarantees about performance of services or resources, but also: price, penalties, QoP guarantees, particular actions, and scaling rules. The splitting strategy will be different for each of the listed aspects. As anticipated in section 4.1, we can place constraints and SLA terms on

1. *individual terms*, on single nodes,
2. *common terms*, on group of nodes, are common constraints still applied individually to each one,

3. *aggregate terms*, on group of nodes, are constraints with aggregate semantics which are amenable of splitting (they can be transformed when split)
4. *invariant terms*, on group of nodes, where the aggregate semantics of the SLA terms cannot change even if the group is split over provider.

For simplicity, in the following we discuss with examples related to a two-part SLA, but the definitions are general.

The semantics of groups 1 and 2 poses the least problem in splitting SLAs, as the constraints are places on application resources or directly inherited by them. In the SLA split, these term simply follow the application splitting.

Most **QoP guarantees** belong to groups 1 and 3, they are not split into either parts of the SLA but are instead replicated and included in all the resulting SLAs. Group 3 differ from group 2 as the semantics of the term cannot be applied to a single resource. Indeed, the general security rule states that the strength of a chain is the same of its weakest ring, thus, splitting any security guarantee would render the whole system less secure.

Splitting **price** constraints will naturally happen when splitting resources or performance, they belong in group 3. The Federation should balance things carefully as the unit price of resources will typically vary among providers. If splitting happens during the negotiation phase between the user and the federation, the price asked to the final customer can still be negotiated and will be determined on the basis of the total cost of the services / resources agreed with each provider, also taking into account changes related to business policies of the federation (e.g. the Federation may apply an intermediation margin or even a discount). If splitting instead happens when the SLA with the final customer has already been agreed, then there is one fewer degree of freedom, the final overall price becoming a constraint for the SLA splitting and optimization problem.

Splitting **penalties** is even more difficult than splitting price, as the likelihood of losing money increases for the Federation and must be compensated through suitable risk reduction strategies.

Violations of split SLAs In general the violation probability for the final SLA is the combination of the violation probabilities of all the SLAs resulting from a split. With reference to the symbols defined in table 4, computing the failure probability is easy if we think of isolated resources with fixed, possibly different failure probabilities,

$$F_a = 1 - \prod_i (1 - F(P_i)) \tag{1}$$

so when considering the split of the SLA we would simply need to minimize F_a.

However, if we take into account that groups of resources are under control of providers which will enact their own SLA, and try to evaluate the expected outcome of this combination of SLAs, a more complex models arises, where the failure probability at provider $F(P_i)$ does no longer represent an isolated resource, but is the outcome of provider P_i managing the $D(P_i)$ resources it is providing, according to its own SLA management policies. Note that the analysis is relevant both for

F_a	overall probability of SLA failure
$P_i, \quad i \in \{1 \dots n\}$	set of providers
A_i	agreed Service Level with provider i (the higher the better)
L_i	provided Service Level from resources on provider P_i
$V(P_i) = \min(0, L_i - A_i)/A_i$	percentage amount of violation at P_i
$F(P_i) = \{L_i > A_i\} =$ $= \{V(P_i) > 0\}$	probability of SLA failure at provider P_i
$D(P_i)$	amount of resources deployed at P_i
V_a	overall amount of SLA violation
T	tolerance, i.e. excess of QoS in the providers SLAs

Table 4: Basic parameters for QoS risk behaviour for a single attribute and a multi-provider split.

- Cloudbursting happening from inside the federation to some public resources, as the SLA on the public cloud provider is critical to ensure the federation can met the one with the user,
- Cloudbursting toward the federation, where a federation user wants to recruit services form the federation to complement those provided by its own private cloud, and the federation needs to recruit those services and resources from more than one provider.

Beside $D(P_i)$, the value of $F(P_i)$ depends on the reliability of provider P_i. So if we can compute the expected overall magnitude of violation as in equation 2, where for simplicity we can assume that the operation $\oplus$ combining the SLAs is a sum.

$$V = min\left(0, \quad \left(\bigoplus_i V(P_i)D(P_i)\right) - T\right) \tag{2}$$

Here T is the tolerance that the federation imposes between the user-agreed SLA and the composition of the SLAs agreed with the providers.

Clearly, optimal choice of the P_i and of $D(P_i)$ is much less easier. An convenient value of T must be chosen, which is a cost for the federation service but also increases the likelihood of matching the agreed QoS in spite of provider issues.

Example The difficulty involved in splitting SLAs with penalties becomes apparent with a simple example. Let say that for a given attribute α the user-agreed SLA imposes a level of performance 100 but also includes a penalty for the federation in case of miss, which is 100% of the application deploy fee.

The trivial choice of splitting this SLA in two, with two SLAs imposing on providers P_1 and P_2 a performance of 50 with a penalty of 100% on their cost (that would mean roughly 50% of the application cost) does not work. If one provider violates the local SLA, the overall one is violated and there is not enough compensation (the Federation has to pay full penalty while only collecting about half of it from providers).

As reported in equation 2, a choice that on average does not cause the federation to loose money has to take into account the expected reliability of providers, the amount of resources placed on each, and likely also has to include some degree of tolerance to reduce the likelihood of an overall violation. Since this optimality problem may be hard to solve in the general case, a basic risk reduction strategy is to minimize the number of parts in which the SLA must be split, so that a full exploration of the solution set is not combinatorially expensive.

Research Directions More advanced strategies for splitting SLAs depend on which custom actions and rules a federation can apply in case of a violation (or near-violation warning) in order to nevertheless enforce the overall SLA.

As anticipated at the beginning of this subsection (page 64) the *actions* depend on the type of QoS terms (single, group, aggregate and invariant) and on the specific QoS term.

E.g "increase backup frequency" is a action concerning reliability, which can be the result of an individual or common SLA term, is easy to tune and can be replicated and included in all the SLAs resulting from a split.

QoS terms in the *aggregate* class, such as scaling rules, should not be replicated as they are in the input. As it is clear from previous example, either (1) the triggering condition is recomputed for each part taking into account risks and costs, or (2) the whole rule is maintained at the Federation level, and the SLA with each provider is structurally different, being used as a precondition that the federation support can exploit to ensure the user SLA. As an example, a given level of performance and reliability can be obtained either with a precise SLA on a single provider, or by over-provisioning from several cheaper providers, if the overall likelihood and predicted cost of a failure can be made low enough (this can also be seen as pushing some deployment choices to the execution time, when the actual behaviour of the different providers is better known). The need to deal with additional SLA parts than those agreed with providers leads to the Contrail taxonomy of SLAs which is discussed in the following section 4.6.

4.6 SLA Coordination

This section surveys the range of techniques available for coordinating the split of an SLA. We will also discuss some of the techniques that match the different cases of SLA splitting.

We will rely on the concept of **application component** that it is defined as a part of the application (VMs and all associated resources) hosted by a single Cloud provider. Again, the term "component" is used in its more generic sense, not implying the existence of *SW components* in the application.

As an application is decomposed together with its SLA, we need to keep under control its components and ensure that the overall deployment still satisfies the initial SLA.

We thus need to address the results of the SLA decomposition into *partial SLAs*. In the following we define the main terms used in this section, introducing two distinguished partial SLAs, the GlueSLA and the CooSLA, that address the need to define QoS aspects whose fulfillment is on the infrastructure, or depends on the coordination of the recruited resources and services.

Application SLA — The initial SLA agreed between the federation and the user. In principle, whenever a single provider is able to fulfill it, no SLA splitting happens and after provisioning the Federation behaves as an ordinary Cloud.

Component SLA (CompSLA) — When SLA splitting happens, each application component needs to have a separate SLA in order to be provisioned. In the simplest cases, a Component SLA is either an aspect of the Application SLA (e.g. the storage guarantees which are enforced on a storage provider) or a trivial subdivision of the original SLA. This subdivision can involve partitioning of some SLA aspects and a large amount of replication (e.g. a set of identical, unrelated VMs are evenly scattered among different sites, all ruled by similar SLAs).

Glue SLA (GlueSLA) — In the settings of Contrail federation, communication between Cloud providers can be realized via different network resources. This can be a serious issue when running an application as two or more separate components. We call Glue SLA the part of the Application SLA which deals with guarantees concerning inter-provider resources.

Main examples of inter provider resources are wide area network connections, which are needed for instance to allow the computing nodes of provider A to access the storage system of provider B. The Glue SLA will typically be generated from a subset of the network constraints expressed by the Application SLA, i.e., those related to the inter-provider communications. If a network provider C exists that can enforce SLAs among the nodes of A and B, then the GlueSLA applies to C. From this viewpoint, the GlueSLA is a special case of the CompSLA. The rationale and the impact of GlueSLA on splitting and coordination is discussed later on.

Coordination SLA (CooSLA) — Even after splitting the Application SLAs (for instance in order to recruit a very large amount of resources) we still want to deal with all aspects of application elasticity. The Coordination SLA is the part of the SLA that needs to be addressed directly by the Federation subsystems, covering those issue that in inter-provider deployment are not handled by the providers. The CooSLA will be used whenever compliance with the original SLA requires the federation support to actively enforce actions on the providers (e.g. renegotiate elasticity with provider A if provider B is about to fail the SLA).

- terms in the original SLA that are not amenable to splitting or to straightforward replication

- additional SLA terms which express how the coordination of different components should happen and that need federation-level information to be enforced.

A notable case where CooSLA can be applied has been described in section 4.5.4, being related to Cloudbursting and to the introduction of SLA tolerances for the sake of reducing federation risks in managing split applications over multiple providers.

All the mentioned SLAs can be expressed with the SLA@SOI formalism exploiting the appropriate set of SLA terms defined within the Contrail project.

Rationale of GlueSLA and Coordination SLA We explicitly distinguish the GlueSLA (which a provider is in charge of) and CooSLA (managed directly by the federation) for two reasons.

1. While Contrail relies on the VIN component in order to provide QoS over inter-provider connections, depending on the actual network not all SLA terms can be supported with the same degree of reliability. E.g., network security can be enforced via standard techniques, while network performance is hard to enforce and sometimes even to predict when link bandwidth reservation is not available. SLA terms that cannot be negotiated within the GlueSLA need to be distinguished and placed in the CooSLA, so that the federation SLA management can monitor or enforce them.

2. The federation organization and SW architecture must work even in the case the federation does not have any available entity at all supporting a specific GlueSLA. Assuming that we need to split an application SLA over two service providers A and B, but no network provider C exists that can enforce QoS on the interconnecting links, it follows that the GlueSLA expressing that QoS must rely only on best-effort. Similarly, some feature may not be enforceable even by the federation. We are thus reversing the process, i.e. by placing constraint on the GlueSLA and CooSLA which result from the splitting, we forbid any split which would not be correctly managed at the federation level.

On top of this distinction, several strategies can be devised, which relate to different techniques to generate different contents of partial SLAs at SLA splitting time. The strategies are described in the following sections, and summarized in table 5.

4.6.1 Baseline coordination

Whatever kind of SLA and application splitting is defined, the trivial case of SLA coordination is no coordination at all.

Still, void coordination implies that SLA violations are detected, even if no reaction is planned. Any partial SLA can potentially fail, and in Baseline coordination this raises an exception for the violation of the overall SLA. Before the exception is propagated, the

Table 5: SLA decomposition and sub cases of SLA coordination. We report with respect to the content of the different partial SLAs what are the compatible coordination mechanisms, as well as the kind of splitting methods that generate the combination of partial SLAs.

CompSLA	GlueSLA	CooSLA	SLA coordination	SLA split methods
YES	empty	empty	none	no split
YES	YES	empty	none, baseline	service-, availability-based
YES	empty	YES	migration, balancing	availability-, performance-based
YES	YES	YES	migration, balancing	availability-, performance-based

federation support verifies that the overall SLA is actually violated (if redundant resources are in use, a single violation may be small enough to be undetected by the overall SLA).

The main aim of Baseline coordination is to provide definite information on the integrity state of the application (i.e., the SLA compliance state of the application is always known).

Baseline coordination algorithm When applying Baseline coordination, the CompSLA and GlueSLA can be generated with anyone of the mechanisms described in section 4.5. The CooSLA will always be empty.

As no repair action can be undertaken, the algorithm is a trivial event loop where each CompSLA violation triggers collecting the CompSLA penalty from the defaulting provider, and causes an evaluation of the Application SLA. If that is violated too, the event is logged and notified to the user. The federation is then liable to pay any penalty that was specified in the Application SLA[10].

It is optional to inform the user about local violations which did not cause an overall violation. This last functionality is useful to allow PaaS services (ConPaaS ones in particular) to more efficiently react to IaaS-related events.

4.6.2 Migration-based coordination

Migration-based coordination addresses SLA violations by moving whole Application components from one provider to a different one.

Each component in the application is to be migrated when SLA violation is detected. Potential migration cost has to be taken into account in the overall SLA. That is, in order to accommodate for the reaction delay on part of the federation, suitable tolerances shall be in place between the Application SLA and the different CompSLAs (including the GlueSLA if present), which are encoded in the CooSLA. This imposes a more restrictive

[10]Note that a violation in the GlueSLA is very likely to trigger an overall violation, while functional replication over different providers, e.g. in parameter sweep applications, can likely overcome small violations at a single computation or storage provider.

splitting and mapping procedure, where tolerances are explicitly introduced according to the federation estimated risks and costs.

At a CompSLA violation, the federation needs to find another provider to agree and sign the component's CompSLA. Specific migration HW constraints will have to be satisfied (the most basic one that the same hypervisor is used on the providers, if live migrations is employed).

If the GlueSLA is violated and other connectivity services are available[11], then the GlueSLA should be migrated (or simply renegotiated, as a special case, i.e. paying the price of moving from a best effort service to a guaranteed bandwidth mode).

Migration-based coordination suits a simple scheme where components of the application are defined in a homogeneous way (e.g. a same OVF) and the cost of moving them is relatively small with respect to the Application SLA. For instance, long-running large computational efforts, such as parameter sweep studies or farm rendering of photo-realistic video, will likely be subject to migration operations which drive whole application components where higher performance or lower costs are currently available. In these cases, the cost of migration can become secondary and be practically bounded.

Migration-based algorithm The pseudo code of the migration-based coordination algorithm in figure 14 formalizes the behaviour of the federation runtime in the different cases of violation of the partial SLAs. A few notes to the pseudo code.

- The algorithm is an event loop, where one or more SLA violations may be received.
- We deal with the GlueSLA first, as its violation will typically cause several components to default their CompSLA.
- Any failure is dealt with by remapping the whole component, then the GlueSLA is updated.
- a failure to identify a corrective migration or renegotiate a critical SLA leads to escalating the SLA failure.
- The behaviour of `signal_violation()` depends on the user SLA. Upon a non-recoverable SLA failure different applications will choose e.g. to continue executing (and benefit from the SLA violation penalty) or simply terminate (time-critical services). The flag `application.continued()` may be accordingly set in order to attempt a migration-based re-balancing after such an event.
- the `updateReputation()` function, besides influencing long-term reliability estimates of providers, can include various permanent/temporary blacklisting techniques (e.g. like in tabu search) to avoid that, several application components failing, the components are simply shuffled among providers with no actual gain.

[11] If no other connectivity provider or service is available, a migration of the other resources may theoretically enable a renegotiation of the GlueSLA, but we chose not to deal with this borderline case.

```
procedure migrationCoordination (RunApplication application, ProviderSet providers)
begin
  do loop
    event=waitNextSLAEvent(application);
    application.status.update(event);
    violation=filterViolations(event);
    if(violation.detected())
    begin
      overall_violation=application.SLA.evaluate();
      // already too late, Overall SLA violated
      if (overall_violation && not application.continued()) then
        application.user.signal_violation();
        return();
      else
      begin       //GlueSLA only violation
        glue_violation=application.GlueSLA.evaluate();
        if (glue_violation)
        begin
          newGlueSLA=mappingModule.updateGlue(application, glue_violation);
          solution=negotiationModule.renegotiateGlue(newGlueSLA);
          if (not solution)
          begin
            application.user.signal_violation();
            return();
          end
        end
        //manage components SLAs
        //step 1, update reputations and manage blacklists
        foreach group in application.componentSet;
        begin
          group_violation=group.CompSLA.evaluate();
          if (group_violation) then
            providers[group.assignedProvider].updateReputation(group_violation);
        end
        //step 2, try to reallocate components
        migrationPlanned=False;
        foreach group in application.componentSet;
        begin
          group_violation=group.CompSLA.evaluate();
          if (group_violation)
          begin                       //reallocate a component
            migrationPlanned=migrationPlanned  ||
              mappingModule.remapComponent(application, group.CompSLA);
          end
        end
        if (migrationPlanned)         //update the GlueSLA
        begin
          newGlueSLA=mappingModule.updateGlue(application, NULL_VIOLATION);
          solution=negotiationModule.renegotiateGlue(newGlueSLA);
          if (not solution)
          begin    //we can't remap the components and keep the overall SLA
            application.user.signal_violation();
            return();
          end
          else     //complete all pending negotiation of GlueSLA and CompSLA
            negotiationModule.updatenegotiations(application);
        end
      end
    end
  while not (application.status.terminated())
end
```

Figure 14: Pseudocode of migration-based coordination.

- The functionality `mappingModule.remapComponent()` of the federation mapping component is to devise a provider to map the group passed as second argument. The output is constrained to be different from the current mapping (different provider or different SLA terms) and will take into account latest provider reputation updates. It returns true if a new mapping is actually found.

4.6.3 Rebalancing coordination

The most complex approach to the coordination of different SLAs is to exploit the elastic capacity of some provider, supporting components from the application, to match the violation of other providers.

This approach allows application component splitting at execution time. That is, if an application component cannot grow on a given provider, the federation will launch a split operation as well as a mapping one in order to allocate part of the component on another provider.

In this more complex scenario we aim at taking into account not only constraints related to the user SLA, but also constraints of efficient use of providers (freeing them as they are underutilized).

This solution is suited to the cases of performance based splitting (where we not only address the user SLA but also other optimality criteria), of tolerance added by the federation on top of the partial SLAs (as we do not want the tolerance cost to run too high), and of enacting run-time renegotiation with providers.

We do not provide yet an algorithm for this case of multiple SLA coordination, as it is beyond the state of the art and several research issues have still to be settled.

Approaches that are considered in the literature, although they usually target management of resources within a Cloud provider and not inter-provider management, include in particular rule systems and the definition of different levels of compliance (e.g. high/low thresholds in the SLA term satisfaction triggering adaptation rules).

When applying rules, they may take into account both the amount of the SLA violations, and the output of optimization procedures used in the mapping phase (for the evaluation of past solution, and memoized, or related to the evaluation of potential new mappings).

4.7 Mapping

In its general formulation the mapping activity consists in finding a set of resources able to execute a given application, typically represented by a graph. As discussed in section 4.1, a Contrail federation employs an Abstract TIG where related virtual machines can be coalesced into homogeneous groups.

We focus here on the mapping activity performed by the federation support. The *Mapping* module within the Federation Runtime Manager component has to find a suitable provider (or set of providers, if group splitting is enacted) for each group of appliances in the application (the basic case being when all appliances end up on the same provider).

To this end the *Mapping* module considers different sources of information.

- Application execution constraints extracted from the OVFs, the user preferences, and the SLA proposals specified by users. This set of information is used to build a comprehensive set of constraints that have to be satisfied by a proper (set of) provider in order to successfully execute the application.
- A coherent view about providers and their actual resource availability is achieved via a provider monitoring interface. Information is kept by the federation in an aggregated form. Although the information is approximate an possibly incomplete, by early discarding unsuccesful candidates we increase the success rate of provider negotiations, and thus the scalability of the mapping process.

Information about Resource Availability We do not assume that all providers would be willing to provide detailed information about their available resources. We can assume that providers willing to federate within Contrail will accept some compromise between disclosing their data and being exploited by Contrail users (the more information is exposed, the more the provider is trusted by the federation and receives work).

When no insight is available about the provider (e.g. for public clouds) this must reflect in the reputation of that provider. The mapping procedure will have to take care in advance of the increased risk of violations, e.g. by adding extra tolerance margins to the CompSLA or by reducing the rank of that provider with respect to others.

Optimization Criteria For the federation-level mapping phase, we consider algorithms able to map the application graph on resources represented in an aggregated form. When there are many providers able to execute an application, the problem is to find the "best" one. The actual definition of best fit is not trivial. The providers will usually define their best choice as the mapping which satisfies their agreed SLA with minimal over-provisioning and resource cost. The user is normally allowed to state an SLA but not a preference criterion, i.e. a function to optimize.

In this scenario the federation has a peculiar role as intermediator. By comparing the offers of several providers, the federation can actually devise several solutions satisfying the user SLA proposal, and can afford to optimize by choosing with a specified criterion. The utility function of the federation will be a combination of

- the gain obtained by overpricing provider resources,
- the satisfaction of the providers (e.g. the amount of resources they are able to sell thanks to the federation),
- the satisfaction of the user (in term of improved achieved QoS from the federation, with respect to single providers).

These three components can be expressed as linear functions, as well as the constraints are, giving rise to mixed integer linear programming subproblems.

However, at least three classes of mapping problem are normally generated

- mapping/remapping of an application component
- mapping all the components of an application

- find the optimal split of an application component over two (a fixed number of) providers.

When the issues shown below are added to the scenario, it cannot be always granted that the overall problem is still solvable with linear programming tools (several discrete variables and weights are added, making the actual solution space complex to explore).

To solve this kind of problems usually a two layer, meta-heuristics approach is used. The "simpler" subproblems are solved directly (e.g. subproblems of the mapping which reduce to linear programming). These solvers are used as a subroutines by a higher-level search strategy that is put in charge of the remaining parameters in order to explore the whole solution space.

This kind of (possibly combinatorial) search generally exploits a set of heuristics. The existing heuristics-based tools allow to linearly combine multiple score functions with assigned weights, as well as to choose different heuristics. Some promising metaheuristics approaches have been listed in section 4.5.1.

This approach allows to configure the mapping process. For some of these algorithms we could allow the user itself to customize the weights of the metrics and heuristics. If the different heuristics provided by the federation have a definite bias for, say, higher performance, reduced cost or higher reliability, allowing the user to tune their balance is an opportunity to support extended SLAs of the form "find the resources which provide the *quickest answer* within a given *cost* constraint".

4.8 Provisioning

The goal of the provisioning algorithm is to initialize application resources (i.e. network, storage, computing and other services) in a correct order, when deploying a single application on multiple providers. Order is essential, as most of the provisioned parts of a multi-provider application rely on information produced and/or resources provided by different providers.

The interaction diagram of the provisioning algorithm of the federation is shown in Figure 15, while the pseudo code is reported in Figure 16. The algorithm is intended for Contrail cloud providers, where we assume the existence of local VIN and GAFS components controlled by the provider, and global access points to the VIN and GAFS functionality. The extended algorithm that enables provisioning also for external clouds is currently under study.

The procedure takes in input (i) a description of the application, which is composed by a list of appliances and their URI, and (ii) the mapping between appliances and providers (it is managed via *selectAppliancesForProvider* in the pseudocode).

The first step is to inform providers about the appliances to deploy as well as the related, agreed SLA, and collect their actual decisions (e.g. IP addresses of VMs and/or provider access points).

The *sendAppliances* procedure (i) forwards to the provider information on the appliances to start, causing their actual provisioning, (ii) requests the virtual machines to be deployed in a frozen state (execution does not begin) so that non-local resources are nor accessed yet, and (iii) collects information (*DeploymentInfo*), which includes the (possibly virtual) IP addresses of the appliances deployed.

In the second step, the collected information is used to drive the contextualization of storage volumes and virtual networks. This second step ensures that all the virtual machines share the same subset of storage volumes and private networks access, by forwarding the VIN and GAFS global access points information about all provisioned VMs. Here we make the Contrail-specific assumption that providers know how to contact the VIN and GAFS access points.

Finally, the appliances are unfrozen on the providers. The FRM keeps the control on appliances through a VMset handle (one per provider). The FRM also manages application termination. After the termination/freeze of all the appliances each provider returns a summary of the execution, then the FRM handles the de-contextualization of the inter-provider infrastructural resources VIN and GAFS.

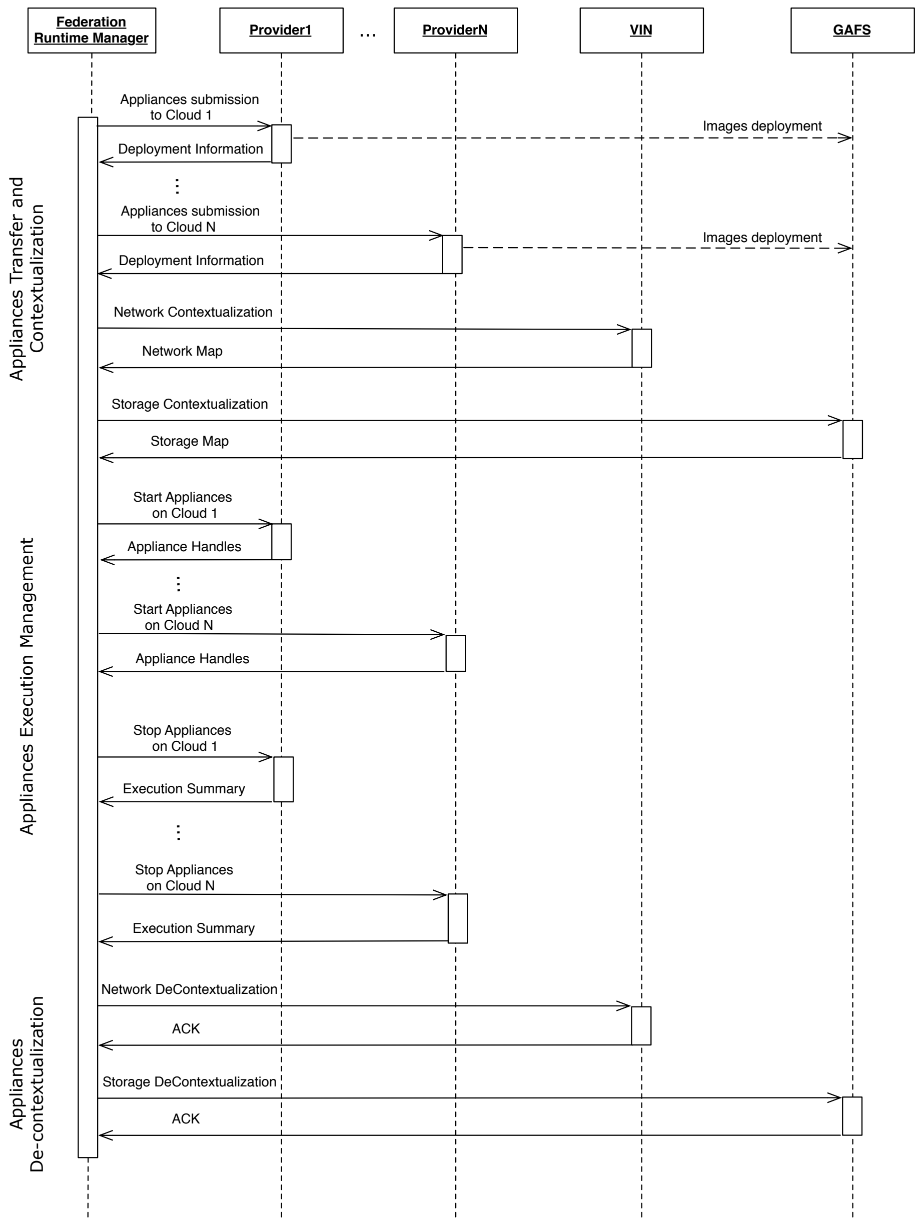

Federation Runtime Manager
Provider1
...
ProviderN
VIN
GAFS
Appliances Transfer and Contextualization
Appliances submission to Cloud 1
Images deployment
Deployment Information
Appliances submission to Cloud N
Images deployment
Deployment Information
Network Contextualization
Network Map
Storage Contextualization
Storage Map
Appliances Execution Management
Start Appliances on Cloud 1
Appliance Handles
Start Appliances on Cloud N
Appliance Handles
Stop Appliances on Cloud 1
Execution Summary
Stop Appliances on Cloud N
Execution Summary
Appliances De-contextualization
Network DeContextualization
ACK
Storage DeContextualization
ACK

Figure 15: Interaction diagram for multi-provider application provisioning.

```
procedure provisioning (Application application, Mapping map)
begin
   foreach Provider pr in map do
   begin
      // appliances are extracted (and flattened) for provider
      setOfAppliances apps =
         selectAppliancesForProvider(pr, application);
      foreach appliance a in apps do
         apps.addImageURI(a, retrieveImageForAppliance(a));
      end
      DeploymentInfo.enqueueInfo( sendAppliances(pr, apps, SLAs) );
   end

   NetworkMap = contextualizeNetworks(DeploymentInfo);
   StorageMap = contextualizeStorage(DeploymentInfo);

   foreach Provider pr in map do
   begin
      apps = selectAppliancesForProvider(pr, application);
      AppHandles.addHandle( start(pr, apps) );
   end
   // save DeploymentInfo and AppHandles for later use
end

// at application termination time
procedure deprovisioning (Application application,
   handleSet AppHandles, DInfo DeploymentInfo)
begin
   foreach handle in AppHandles do
   begin
      ExecutionSummary.enqueueSummary( stop(handle) );
   end

   decontextualizeNetworks(DeploymentInfo);
   decontextualizeStorage(DeploymentInfo);

end
```

Figure 16: Algorithm for multi-provider application provisioning.

5 Federation Interfaces

The federation is both an access point to the cloud resources as well as a way for the user to abstract from the actual details of which provider is actually serving the resources. The interface for the federation must therefore support the following categories of tasks:

- Cloud Federation management. This includes the management of the properties of the federation, the profiles of the cloud providers, and the cloud users.
- Cloud resource usage. The federation serves as a front end from the providers (Contrail VEP or non-Contrail public cloud) towards the cloud users.

5.1 Web Interface

Attracting less technically apt users to a product or a platform depends on the comprehensiveness as well as a good design of a graphical user interface (GUI). In certain cases, a good backing in a user interface can also improve the experience when having to perform maintenance tasks on a complex system.

The Contrail federation represents a system that we expect to be largely used in an automated fashion, either through scripts using the command-line interface, or directly using its API. Nevertheless we want to expose all the functionality of the federation through a user-friendly interface. Naturally, we find the web interface to be the most suitable way of making a client-platform-independent and ubiquitous GUI. Therefore, it should be the hub of the functionality that includes:

- setting up and maintaining the basic parameters of the federation operation;
- management of the properties of the cloud providers that are part of the federation;
- user management;
- appliance and application management.

The federation web interface is designed in a streamlined fashion, providing the user a top-level navigation with the web pages organised into categories. Each category represents a set of related tasks. The categories' availability for the logged in user are subject to the user's access rights. In other words, a user only sees and has access to the pages with those tasks that the user is entitled to use. In the following sections we describe the categories and the activities that the user can perform in each category.

To make a start with the activities, the user visits the federation provider's web page. She is presented with a log in page, that may be similar to the one in Figure 17. If the user does not have a valid account for the federation, there may be an option for registering as a new user. For the users with a valid account, however, the starting page will provide various schemes of authentication, such as basic username and password, OpenID, Shibboleth and similar.

The users that have more than one role at the federation will be able to select from their valid roles in order to act in the web interface with the capabilities and access rights assigned to that role alone.

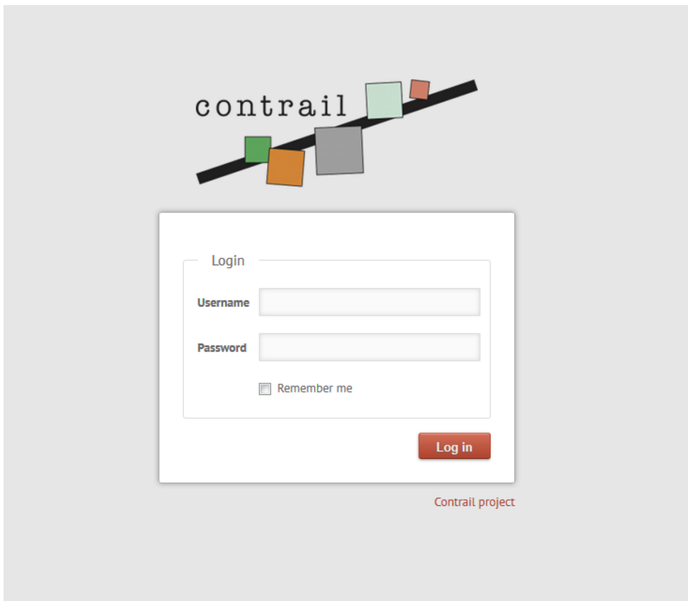


Figure 17: The log in page.

5.1.1 Federation Administration

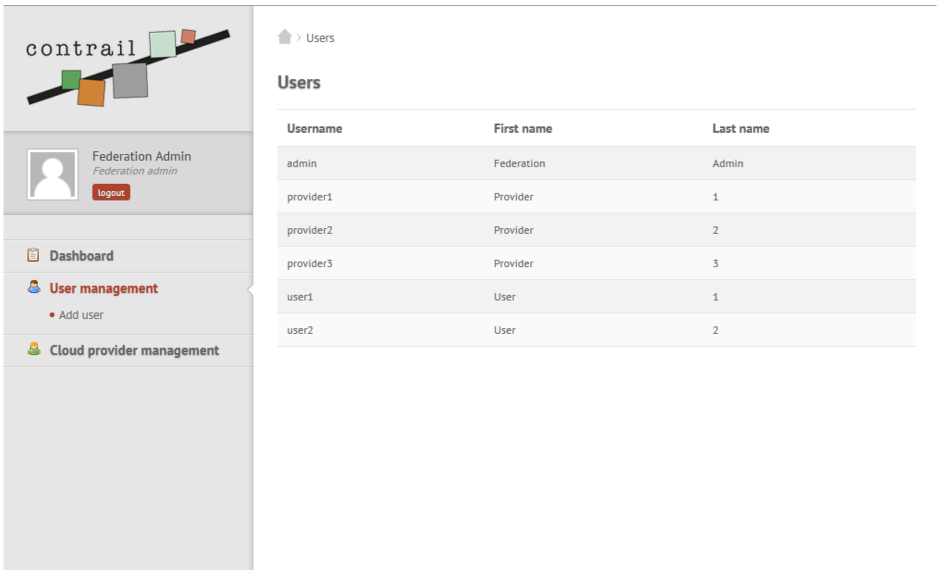
This category contains the pages with the tasks that enable the management of the federation. We expect that this functionality will only be available to few selected users with credentials of a federation coordinator role. Such a user can perform the following tasks:

- **User management.** The coordinator can create new accounts for the users, block or remove existing users, and approve (or deny) access to the newly registered users.
- **Cloud provider management.** Here, the coordinator can manage properties related to the protocols between the cloud providers and the federation service, edit any operation policies, etc. As its most basic task, the federation coordinator has to represent the Federation when establishing the mutual trust agreement between the Federation itself and any newly joining Provider.

Figures 18 and 19 show an activity where the federation coordinator can view the up-to-date list of the federation providers and an example of the form for adding a new provider respectively.

The federation coordinators can also access reports on the usage of the resources in the federation:

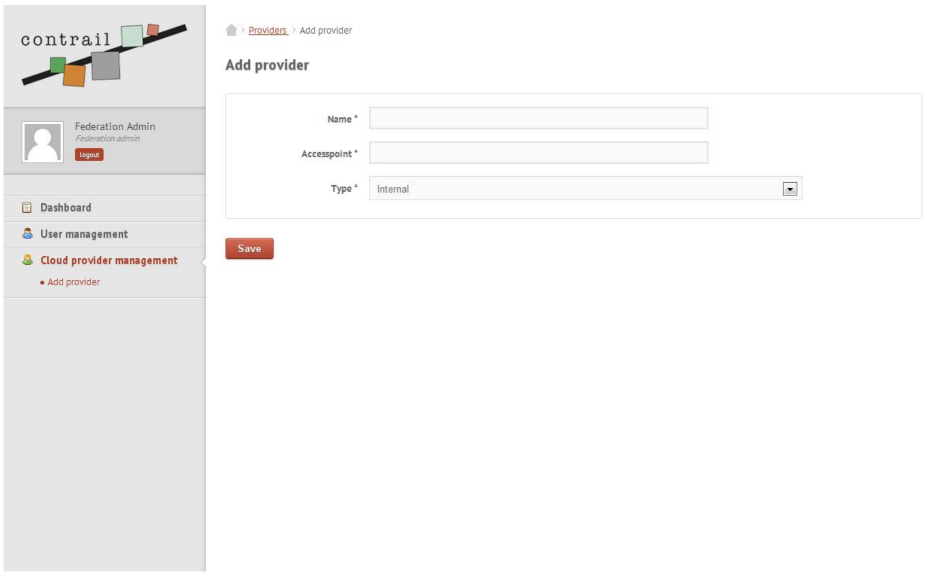
- Periodic (e.g., monthly) reports summarising the resources consumed by the users, aggregated by the cloud providers;
- Periodic reports on the overall activities per each provider.



Users

Username	First name	Last name
admin	Federation	Admin
provider1	Provider	1
provider2	Provider	2
provider3	Provider	3
user1	User	1
user2	User	2

Figure 18: List of providers in the federation.



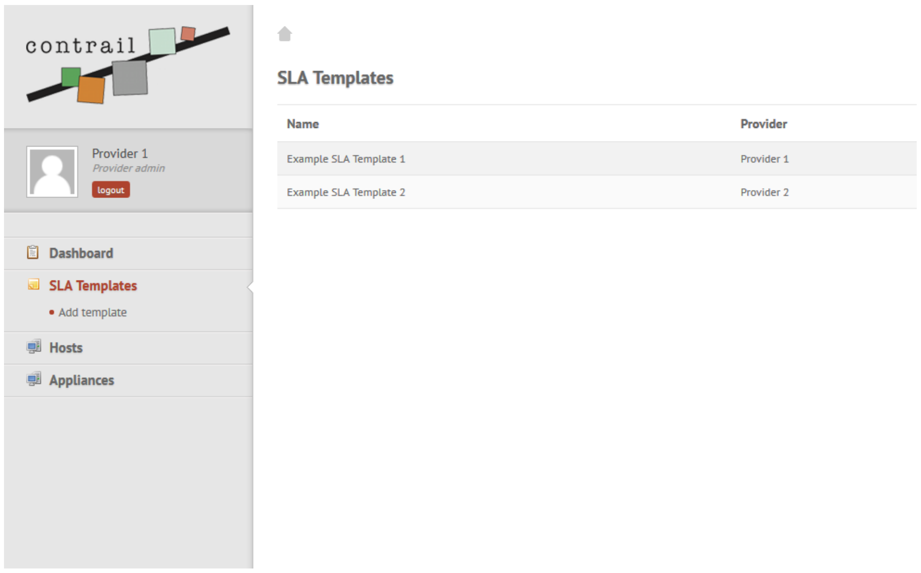
Add provider

Name *	
Accesspoint *	
Type *	Internal

Figure 19: Adding a new provider to the federation.

5.1.2 Provider Administration

This category is designed for the administrators of the individual cloud providers that want to be or are an active part of the cloud federation. Each cloud provider administrator can only access the functionality and properties of their own cloud as it is represented in the federation. The tasks include the following:



SLA Templates

Name	Provider
Example SLA Template 1	Provider 1
Example SLA Template 2	Provider 2

Figure 20: Listing the SLA templates of the current cloud provider.

- **Management of the SLA templates.** Here, the provider administrator can add or change the SLA templates that tell the federation and its users the terms that can be negotiated with this provider.
- **Management of the networks.** The federation will query for the list and details of the networks automatically, but the provider administrator may override the values if needed.
- **Storage management.** This includes uploading and registering any disk images that can be a part of the appliances ready for the users of this provider.
- **Appliance management.** This includes registering appliances that the provider is willing to offer to the users of this federation.

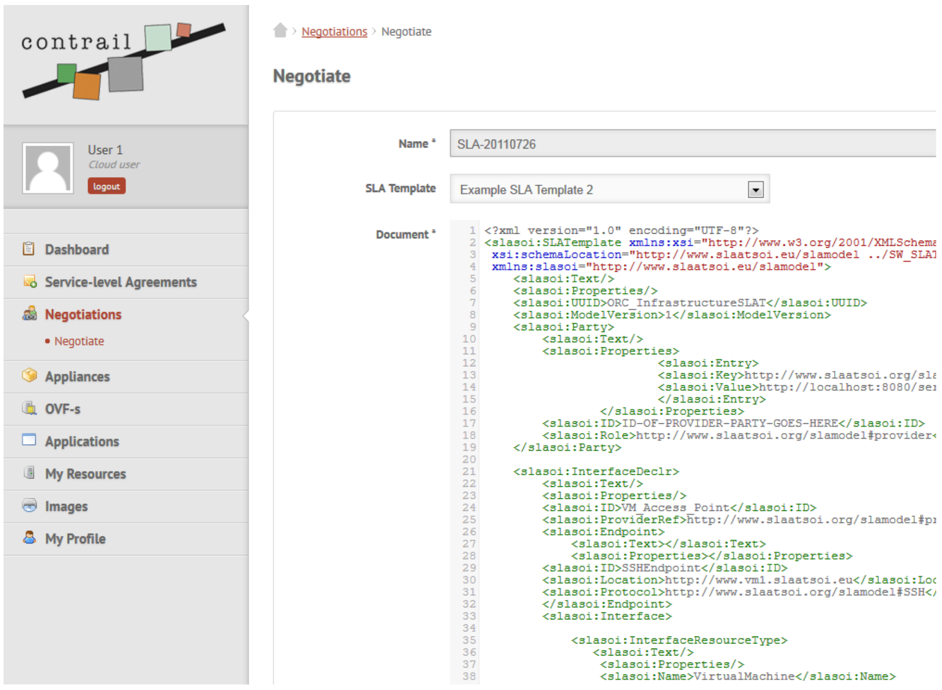
The provider administrator can also get reports on the resource usage from the federation users and monitor the activities as seen from the federation.

Figure 20 shows an example of the page where the logged in provider administrator can view the list of the SLA templates that the provider supports.

5.1.3 Cloud Federation

The Cloud Federation category is accessible to the users of the cloud providers' resources. This covers the following tasks:

- **Service-level Agreements management**, including initiating a (re)negotiation from the available SLA templates.



Negotiate

Name * SLA-20110726

SLA Template Example SLA Template 2

Document *

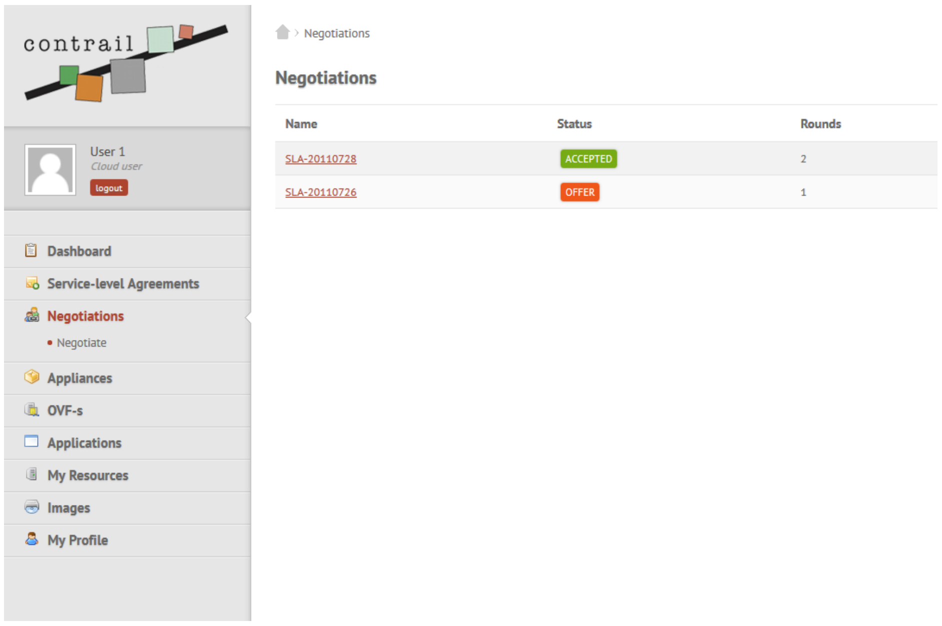
```
<?xml version="1.0" encoding="UTF-8"?>
<slasoi:SLATemplate xmlns:xsi="http://www.w3.org/2001/XMLSchema
 xsi:schemaLocation="http://www.slaatsoi.eu/slamodel ../SW_SLAT
 xmlns:slasoi="http://www.slaatsoi.eu/slamodel">
    <slasoi:Text/>
    <slasoi:Properties/>
    <slasoi:UUID>ORC_InfrastructureSLAT</slasoi:UUID>
    <slasoi:ModelVersion>1</slasoi:ModelVersion>
    <slasoi:Party>
        <slasoi:Text/>
        <slasoi:Properties>
                        <slasoi:Entry>
                        <slasoi:Key>http://www.slaatsoi.org/sla
                        <slasoi:Value>http://localhost:8080/ser
                        </slasoi:Entry>
                </slasoi:Properties>
        <slasoi:ID>ID-OF-PROVIDER-PARTY-GOES-HERE</slasoi:ID>
        <slasoi:Role>http://www.slaatsoi.org/slamodel#provider<
    </slasoi:Party>

    <slasoi:InterfaceDeclr>
        <slasoi:Text/>
        <slasoi:Properties/>
        <slasoi:ID>VM_Access_Point</slasoi:ID>
        <slasoi:ProviderRef>http://www.slaatsoi.org/slamodel#pr
        <slasoi:Endpoint>
            <slasoi:Text></slasoi:Text>
            <slasoi:Properties></slasoi:Properties>
        <slasoi:ID>SSHEndpoint</slasoi:ID>
        <slasoi:Location>http://www.vm1.slaatsoi.eu</slasoi:Loc
        <slasoi:Protocol>http://www.slaatsoi.org/slamodel#SSH</
        </slasoi:Endpoint>
        <slasoi:Interface>

            <slasoi:InterfaceResourceType>
               <slasoi:Text/>
                <slasoi:Properties/>
                <slasoi:Name>VirtualMachine</slasoi:Name>
```

Figure 21: An example form for initiating a SLA negotiation by selecting a SLA template.

- **Appliance management**, including the possibility for deploying the pre-provided appliance or uploading user's own ones.
- **Application management**, including monitoring the current application's activities, manipulating VMs, managing notifications, etc.
- **Profile management**, where the user can view or change any details of the user account, upload or request credentials, etc.

The starting point for the users is the negotiation of the SLA that will apply for all the user's applications. It takes shape of a wizard where the user first selects from a list of available SLA templates, as shown in Figure 21. Upon selection of a suitable one, the user can change specific values of the template and submit the document into the negotiation. The web interface shows an up-to-date status of the negotiation (Figure 22), which can have one of the following statuses:

- *pending*: the federation is waiting for the providers to give a response to the current user's SLA template.
- *offer*: the federation has a template that is acceptable for the providers. The user can either accept, reject, or provide a counter-offer.
- *rejected*: the federation cannot guarantee the terms based on the provided SLA template.



Negotiations

Name	Status	Rounds
SLA-20110728	ACCEPTED	2
SLA-20110726	OFFER	1

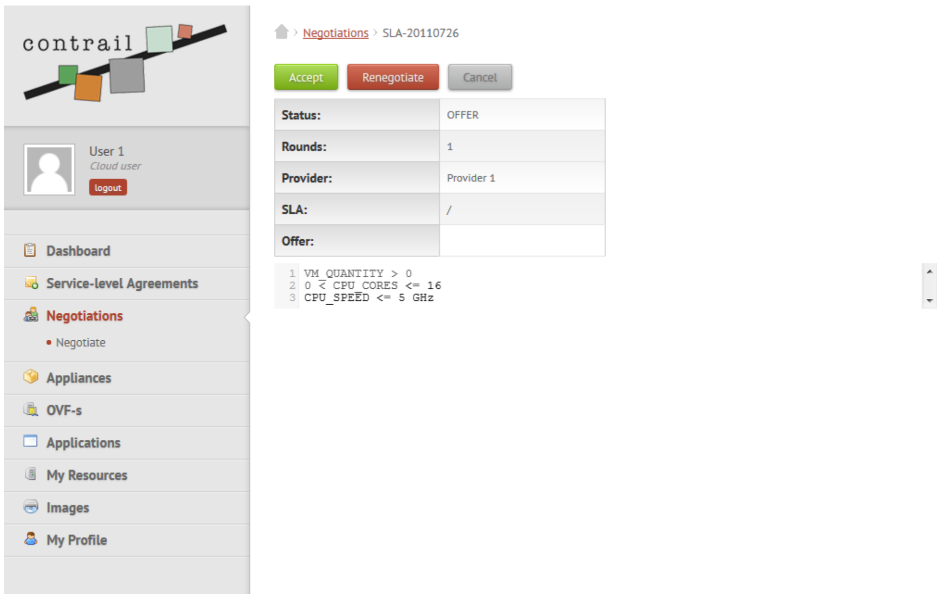
Figure 22: The status of the past and undergoing negotiations.

The user can inspect the current state of a negotiation by selecting it in the user interface, like, for example, in Figure 23. If the negotiation is in the state *offer*, the user has an option to change the offered parameters in the SLA template, sending it into another round of negotiation. Alternately, the user can also decide to stop the negotiation, by either canceling it altogether or accepting the current terms.

The web application provides the facilities for getting the reports on the past user's activity, such as the amount of resources consumed as well as up-to-date graphs showing a more detailed diagnostics of the application's performance. The user can also find in the same area the possibility for setting up instant notifications of important events that the application publishes. These notifications can appear in the web interface's windows as pop-ups, or the user can decide to have them sent to her e-mail address.

We note that the cloud federation aspects of the web interface are the ones that the majority of users will access. Therefore, we want to make the user experience for most of them as good as possible. To this end, we give the users an option to access a simplified version of the Cloud Federation web interface category, where the more advanced and complex functionalities are hidden. The ones available are the follows:

- **Applications.** The user can view the current status of the virtual machines in the application, add, or remove running virtual machines. The workflow assumes that a user is assigned a pre-negotiated SLA.
- **Profile.** The user can view and change her own profile's properties here.



Status:	OFFER
Rounds:	1
Provider:	Provider 1
SLA:	/
Offer:	

```
VM_QUANTITY > 0
0 < CPU_CORES <= 16
CPU_SPEED <= 5 GHz
```

Figure 23: The details of a SLA template under negotiation.

5.2 REST Interface

The federation is a service or a set of services, whose functionality needs to be exposed not only towards users, but also towards other software. This includes services running as a part of the Contrail stack, but our vision is also wider, covering custom applications, because including the federation in the automation schemes is very important. We feel that openness and ease adoption of the API help in a faster integration of the federation services into the Contrail, and it also grants faster inclusion into frameworks created by the community.

We feel that by using the Representational State Transfer (REST) [21] paradigm we can achieve our goals. The REST style uses the basic HTTP verbs for creating (POST), reading (HEAD, GET), updating (POST, PUT) and deleting (DELETE) resources. The state of the resources describes the state of the federation and its underlying entities.

We designed the Federation API such that it reflects the design of the Contrail VEP API as described in [15]. While the two APIs are not required to be identical, in a practical sense, the federation deals with similar concepts as VEP, even if they appear on a higher level.

5.2.1 Addressing the entities

The Contrail API bases their instances and properties on a set of fully addressable resources. Each entity has a unique URI which takes the standard HTTP URL form:

```
http://federation1.contrail.eu/api/resource/id
```

which consists of the following elements:

- `http://` is the protocol used for invoking the API. The server may also provide `https://` for a higher security.
- `federation1.contrail.eu` is a fully qualified host name of the service's (or the API front-end's) host.
- `/api/` is the name of the API to receive the requests.
- `resource/` is the name of the resource to receive the request.
- `id` is the name or the unique identifier referencing an instance of the resource.

The URI can express a hierarchy of resources, signifying ownership or customizations. For instance, the following URI represents all the SLAs owned by user `mary`:

```
http://federation1.contrail.eu/api/user/mary/sla/
```

5.2.2 HTTP Verbs

In order to obtain or change the state of the resources, effectively by using the federation, we employ a subset of the HTTP verbs invoked on the URLs. Table 6 summarises the effects on different types of entities referenced by an URI.

Type	`GET`	`POST` (create)	`POST` (action)	`PUT` (create)	`PUT` (update)	`DELETE`
Resource (path ends with `/`)	List of resource instances	Create a new instance		-	-	Delete all the instances of resource
Resource instance	Rendering of the instance (e.g., attributes)	Create a new instance		Create a new instance at given path (where applicable)	Update the resource instance	Delete the resource instance

Table 6: The effect of the HTTP verbs.

The Contrail API also responds to the HEAD and OPTIONS verbs. The HEAD verb on a resource instance retrieves the resource's metadata. The OPTIONS verb retrieves the supported verbs on the specified URI.

5.2.3 Contrail API resources

The essential part of each RESTful API is a set of resources that are uniquely addressable. The construction of the set depends on the overall state that represents the federation as well as the processes involved. The resource list does not necessarily fully reflect the model or the implementation of the federation. For example, a negotiation is a process, which involves records containing SLAs. But to properly express the process in REST, we introduce a resource for negotiation, even though a resource for SLA exists.

Here is the full list of the resources for the Contrail federation:

- user
- provider
- SLA template
- SLA
- negotiation
- image
- ovf
- deployment document
- appliance
- application
- usage policy
- notification
- report
- cloud provider membership
- network
- storage
- virtual machine

On top of the resources, we provide the following links:

- appliance provider: links appliances to the cloud providers where the appliance is supported
- cloud provider membership: links a user to a cloud provider
- appliance network: links an appliance to a specific network
- appliance storage: links an appliance to a specific storage
- storage image: links an image to a storage
- image provider: links an image to a provider where the image is hosted
- ovf deployment document: links a deployment document with an OVF document
- sla deployment document: links a deployment document with an SLA document

The list is flat, even though some of the resources appear as subordinates to another resource. The client implementation should therefore not construct the URIs, but rather follow the hyperlinks obtained in the responses.

Please find a more detailed reference of the Contrail API resources in the Appendix A.

5.2.4 HTTP rendering

In the Contrail federation REST API, we define the following format of the HTTP rendering. When the client obtains a result from a request or a query, the body of the response contains the links to the resources that are relevant for any further navigation. The rendering depends on the media type requested by the client.

For example, the following transcript is a result of querying for a user using the `text/json` media type:

```
> GET /api/user/user-001
> Accept: text/json
> [...]

< HTTP/1.1 200 OK
< Content-Type: application/json; charset=utf-8
< [...]
<
< {
<     "username": "user-001",
<     "email": "my-email@contrail.org",
<     "group": "federation-admin",
<     "status": "active",
<     "user-slas": [
<         {"name": "sla001", "link": "/api/sla/sla001/"},
<         {"name": "sla012", "link": "/api/sla/sla012/"}
<     ],
<     "resource-usage": "/api/user-001/resource-usage/"
< }
```

Similarly, the same information can be requested using `text/xml`:

```
> GET /api/user/user-001
> Accept: text/xml
> [...]

< HTTP/1.1 200 OK
< Content-Type: application/xml; charset=utf-8
< [...]
<
< <?xml version='1.0' encoding='utf-8'?>
< <user>
<     <username>user-001</username>
<     <email>my-email@contrail.org</email>
<     <group>federation-admin</group>
<     <status>active</status>
<     <user-slas>
<         <sla name="sla001" href="/api/sla/sla001/"/>
<         <sla name="sla012" href="/api/sla/sla012/"/>
<     </user-slas>
<     <resource-usage href="/api/user-001/resource-usage/"/>
< </user>
```

The following example shows the exchange when adding a new user. If successful, the result (HTTP/1.0 201 CREATED) shows the location of the newly created resource instance.

```
> POST /api/user/
> Content-Type: application/json; charset=utf-8
> [...]
> {
>     "username": "user-020",
>     "email": "user020@contrail.org",
>     "group": "cloud-user",
>     "status": "active"
> }

< HTTP/1.0 201 CREATED
< [...]
< Location: http://localhost:4567/api/user/user-020
```

5.2.5 Relation of the Contrail Federation API with the OCCI

The OCCI Infrastructure document [10] specifies the classes involved in the cloud computing infrastructure. The HTTP RESTful rendering [11] defines the standard format to be implemented by an infrastructure front-end.

The Contrail Federation acts on the behalf of the cloud provider's infrastructure, therefore, it is suitable for exposing the OCCI API. The Contrail Federation API in the first release does not yet comply with the OCCI, as in the first prototype we seek to define both the common denominator with the OCCI as well as the extensions necessary for the Contrail Federation client to be able to perform all tasks. Considering that our goal is to provide a standard interface, we also designed the Contrail Federation API in such a way that it will be upgradeable into the full standard specification with as few changes to the API as possible.

To approach the OCCI-compliant rendering, we provide the `text/contrail` and `application/contrail` media and content types. The resource attributes and their values need to be sent from the requester to the service in the headers of the requests. We use the `X-Contrail-Location` headers to express locations, and `X-Contrail-Attribute` when passing attributes and their values. As for attributes, the header should contain a `contrail.resource.attribute=value` string, where `resource` is the resource which the attribute belongs to, `attribute` is the attribute name, and `value` is the value to be set, changed or, in the response, is currently set to the resource's attribute.

5.2.6 Obtaining the resource usage

A role of the API is also the ability of the users to obtain the resource usage of the relevant cloud resources. The resource usage is subject to access control in terms of both the access to the HTTP resource and the content retrieved. Table 7 shows the resources with the available resource usage functionality and the related restrictions.

Resource	**Usage records retrieved**	**Restrictions**
user	the cloud resources consumed by the user	a cloud user is permitted to obtain own usage; the cloud provider user obtains usage filtered by provider's resources; the federation user obtains an aggregation
provider	usage of the cloud provider resources	available to provider's users only
application	usage of the virtual machines associated to the application	available only to appliance's owner

Table 7: The resource usage records retrieved from various resources, depending on the type of user doing the queries.

In order for the client to obtain the resource usage, it needs to GET an URL constructed as follows:

```
http://federation1.contrail.eu
   /api/resource-usage/?from=start_time&to=end_time
```

The `start_time` and `end_time` are the timestamps that limit the time range of the resource usage queried.

The client does not have to construct these URLs, rather it obtains them as a response when querying the resource with GET.

5.2.7 Authentication and Authorisation using REST

In using REST, we use a popular and community-endorsed technology, which uses standard HTTP verbs to manipulate resources. The Resource-Oriented architecture also advocates statelessness of the services: each request from the client should generally encompass all the needed information for the server to be able to perform the required operation, without having to have some sort of a trace of the client's previous activity. This includes authentication, i.e., "logging the client in", to establish their identity and access rights as an initial step of some sequence of requests.

In Contrail, we require in addition, single sign-on capability since we are providing a federated infrastructure to manage multiple accounts over a range of cloud providers. We also require support for access via console applications and scripts as well as browsers. This is an important consideration since many of the technologies available in this space are browser-centric in their focus.

Despite the popularity of REST interfaces, there seems to be no commonly-agreed upon standard for the authentication, the authorisation, and the transmission of any associated user attribute information required. In this section we consider options, building on the initial discussion in sections 4.1 and 4.2 of Contrail Deliverable D7.1 [18].

HTTP Basic Auth and HTTP Digest based authentication provide the most obvious means and if deployed with transport layer security would be secure as well. We can find an example of a proprietary authentication mechanism in Amazon S3, e.g. the use of a shared secret between Amazon and the user to sign specific header values of HTTP messages exchanged. Interestingly, however, some authors [21] suggest that giving to S3 the header encrypted with one's secret is a valid way of sharing a resource within S3. This is clearly an unsatisfactory way to secure assets of any value. We note the work from the HTTPsec [3] project, which provides a more sophisticated means for mutual authentication between client and server by using RSA public keys. Artifacts are passed in the HTTP header. However, this protocol appears to have had little take up.

Turning to single sign on technologies, OpenID provides a means of single sign on and has RESTful characteristics in that a URI is used to represent a resource: a token representing a user's identity and their home organisation or Identity Provider. A number of security concerns such as for example vulnerability to phishing make this technology suited for the protection of resources requiring only a low level of assurance. However, these can be mitigated to large extent by stipulating transport level security for both OpenID Provider and Relying Party endpoints. OpenID is targeted at web browsers although a profile could be established to enable use with console applications. However, this would exclude commercial providers such as Google and Yahoo. The SAML 2.0 specification has included the ECP (Enhanced Client or Proxy) Profile targeted at non-browser clients. SAML has widespread take up in the academic sector with Shibboleth.

In the Grid computing community, personal user X.509 certificates are well established and used for authentication / single sign-on. RFC3280 proxy certificates enable the creation of short-lived credentials and delegation to other entities to act on a users behalf to access resources. However, specialist SSL middleware is required for consumers to correctly verify them. The MyProxy Online CA enables short-lived credentials to be generated from username/password without the need to resort to proxy certificates. Such short-lived credentials can be used with HTTPS in the SSL-handshake to enable a server to authenticate a client.

OAuth [13] addresses a different use case, enabling the delegation of authorisation rights to another entity. It can be used together with single sign-on technologies like OpenID to facilitate access to secured resources in a federated environment. We note that some authors advise against using it for a non-browser based API [20].

Along with authentication credentials, a client may push attributes to enable the consumer to authorise the user to access resources. Alternatively, a consumer may pull attribute information out of band of the authentication channel by querying an independent attribute service. Pushing attributes has some advantages in that enables the client to determine the exact attributes released to the consumer. However, the limitations of HTTP browser clients, such as the restrictions in header size and URL length, mean that it can be difficult to pass the necessary information. SAML provides the artifact binding to enable attributes to be retrieved on a back channel independently of the main communication channel. The WS-Federation Passive Requestor Profile recommends the use of requester-side scripting to enable larger content to be returned using the POST method. OpenID Attribute Exchange uses URL query arguments but will resort to the same method if content exceeds the recommended URL length.

With short-lived credentials, it is possible to add attribute information to the certificates extension section provided in the X.509 certificate format. This could for example be in the form of a SAML attribute assertion. Considering the need for a simple protocol for command line clients we have selected this means for Contrail. Readily available HTTP clients such as *wget* and *curl* support SSL client authentication. This method requires HTTPS but this may not be desirable in all cases. To allow for this, the initial channel of communication may be HTTP. The server can respond with a redirect request to a HTTPS endpoint. This endpoint can apply the client authentication and, on success, redirect back to the original HTTP-based URL requested setting a cookie to register the authenticated state.

This solution ultimately means that the user (the command-line user, since the web interface has a back-end with more powerful credentials) will need to deal with private keys and certificates one way or another. Therefore, we need to create some tools, that are not very different from the CDA in XtreemOS or CILogon project developed for Teragrid in the US: a command-line client that creates a private key and sends a certificate request to the credential service, passing along a shared secret (username and password). This could provide for the user's environment the needed certificate and key proving the user's identity. Then the user could "log in" into the federation with another tool, which would use the identity certificate, and obtain some short-lived credentials, which could then be passed with the Contrail federation commands. Using the bash environment variables (or even some keyring) could help here.

5.3 Command-line Interface

Contrail provides the full functionality of the federation also via command-line tools. The command-line interface enables powerful and quick manipulation of the federation's state either directly or through scripts. The commands provided accept parameters according to standard conventions (e.g. returning results in the standard output in a form suitable to further script processing).

The available commands cover the same set of actions that the REST interface does (in fact, they are designed as tools to access the REST interface from a standard command shell). The command line tools can be grouped in categories, based on the aspect of the federation that they manipulate.

- **Federation users commands** — list and manage the information about the users and their profiles at the federation. Therefore these commands represent the interface with the identity management for the users.

- **Cloud provider commands** — manage the view that the federation has over the cloud providers. These commands can be used by the provider administrators in order to register the cloud access points and the related parameters that the federation needs to observe when interacting with the provider. The federation coordinators can use these commands to regulate usage policies and, overall, the cloud provider status in the federation.

- **SLA template commands** — inspect and manage SLA templates.

- **SLA commands** — inspect and manage the agreed SLAs in the federation SLA registry.

- **Appliance commands** — manage the descriptions of the appliances registered at the federation.

- **Deployment document commands** — manage the contents of the deployment document in order to submit applications.

- **Network commands** — inspect and manage the network resources on the federation provided by the cloud providers.

- **Storage commands** — inspect and manage the storage resources on the federation provided by the cloud providers.

- **Image commands** — manage the image files, i.e., the files containing the virtual file systems that can be used by the virtual machines to boot. Using these commands, users can upload images to the federation or register images available elsewhere for download (e.g. in the GAFS), allowing transparent access to the computation providers.

- **Reporting commands** — enable the users to obtain the resource usage reports from the federation.

The extended description of all commands (purpose and parameters) is reported in Appendix B. Since by ordinary security requirements not all actions are allowed to all users, the appendix also lists for each group of commands which roles they are allowed to, with respect to the three standard roles defined in a Contrail federation (federation coordinator, cloud administrator, and normal user).

6 Conclusion

This document defined what a Contrail Cloud Federation is, how it works, and proposed both an implementation architecture and several interfaces to access it. The analysis in this document introduces a major innovative contribution on the concept and mechanisms of a federation of Cloud providers. Federating Cloud Providers under a single Identity Management is an ambitious goal in itself, but Contrail adds to this the idea of coordinating the providers through SLAs, in order to offer a common view and a common SLA to the final user. SLA splitting and SLA coordination are the central research topics supporting this provider coordination.

The features of Cloud Federations are a major Contrail selling point. The analysis reported in this deliverable is just the initial step in the Contrail research path. The next step will be to actually implement a first release, that will be based on the results reported in this deliverable. Contrail Case Studies will exploit this first release and provide feedbacks which will be taken into account in planning the second relelase, along with further solutions and issues that will surface in the first implementation, as well as to research results achieved in the meantime.

The analysis carried on in this document still leaves some open issues, such as the positioning of the Federation entity as a business entity (which may not be unique) and the overall shape of the economic interactions among the Provider, each on itself a separated business entity. Risk management strategies have to be devised to minimize the federation business risk implied by committing on the Quality of the offered Services. Strategies to maximize the Federation's profit have to be studied, as well as their relationship to users' satisfaction. These and other issues will be further investigated and taken into account in the next period.

A Federation API REST Resources

In Contrail, the federation exposed a RESTful API towards clients. Like in any proper RESTful interface, we define a set of resources that are addressable using unique URIs. In this section we provide the list, the description and the details of usage of the resources.

Resource: user	
Attributes:	*Comment*
username	
password	
group	(federation coordinator, provider admin, cloud user, ...)
role	
institution	
status	(application, active, banned)

Resources owned by user:
sla usage policy
Use cases:
Service checks authentication (password, certificate, ...). Admin adds a new user. User changes its profile (attributes). User uploads or manipulates own credentials used for cloud providers.
Description:
The resource represents a collection of users, who log into the federation, manage the state of the federation (subject to access control) and consume the cloud provider resources.

Resource: provider	
Attributes:	*Comment*
provider name/id	
provider's access point	
type: contrail, external	
public certificate / service credential	
reputation	the level of provider's reputation, modifiable by federation only
storage drivers	the list of the drivers supported for storage
networking drivers	the list of the supported networking drivers
compute drivers	the list of the supported computation drivers

Resources owned by provider:
SLA template negotiation
Use cases:
Federation admin adds a new provider. The provider becomes active in the federation. The federation admin updates attributes (access point, ...).
Description:
A cloud provider brings cloud resources to the federation. This resource provides the means for the provider to indicate important details regarding the use of its resources, and the federation can also maintain its relationship towards the provider, such as a level of the provider's reputation.

Resource: SLA template	
Attributes:	*Comment*
provider	the provider offering guarantees covered by the template (optional)
document	the document (XML, JSON, ...) describing the contents of the template

Use cases
A provider admin adds a new SLA template A federation coordinator provides an SLA template A user can obtain a list of available templates. A user can use additional parameters of a GET request to filter a list of the available templates by their metadata values. A user can obtain the template contents.
Description:
The SLA templates are the starting point of negotiations between the user and the cloud provider. It is a document containing generic or default values for the terms as well as the range of permissible values. The cloud providers need to register their SLA templates, but the federation coordinator can also provide additional SLA templates not associated with any specific provider, but which should be compatible with one or more providers' SLA templates.

Resource: SLA	
Attributes:	*Comment*
agreed at	
expires at	
owner	
document	the document (XML, JSON, ...) describing the contents of the SLA
ovf ids	

Use cases:
Created as a result of a successful negotiation (immutable for POST, PUT). A service can consult a relevant part of the agreed terms or the whole document. A user can get a list of the active SLAs. A user can use additional parameters of a GET request to filter a list of the available SLAs by their metadata values.
Description:
The SLA represents an outcome of the successful negotiation and, within its validity time range, the terms that are in effect. It provides the services the means for querying terms and constraints, as well as a reference for actions that are subject to constraints in the SLA terms.

Resource: negotiation	
Attributes:	*Comment*
initiator	the party (user, service) who started the negotiation
provider	the owner of the capabilities to be negotiated
status	at initiator, at provider, succeeded, failed
rounds	
sla template id	the id of the original SLA template (can be void)
negotiation offer	the current SLA offer

Use cases:
The user or service POSTs an SLA offer (an SLA template with parameters filled to the initial parameter values). The user or service checks for status of negotiation in progress. The provider PUTs new values as a counteroffer to the current user's offer. The user or service PUTs new values as a counteroffer to the current provider's offer. The user or service PUTs new values to trigger re-negotiation of already negotiated terms.
Description:
Negotiation is a resource that keeps the status of an on-going negotiation of SLA template (offer) between two parties.

Resource: ovf	
Attributes:	*Comment*
sla	the reference to the SLA the appliance is a part of (optional)
document	the OVF describing the appliance

Resources owned by ovf:
appliance
Use cases
User or service POSTs a new OVF to be used with an existing agreed SLA. User or service requests an OVF deployment (i.e., deployment of resources described in the OVF). User or service can obtain an OVF that can be used for an SLA
Description:
A description of one or more appliances in the form of an OVF document. The OVF can be associated with an SLA.

Resource: deployment document	
Attributes:	*Comment*
document	the document describing the deployment

Use cases
The VEP services retrieve the deployment document. The user updates the deployment document.
Description:
This resource provides access to the deployment document, which contains features and functionality which add to the standard ones appearing either in an OVF or an SLA document. The federation generates the document automatically, but the users can choose to submit their own or alter the created one. The deployment document can be linked to OVFs or SLAs.

Resource: appliance	
Attributes:	*Comment*
ovf	the reference to the OVF the appliance is a part of
id	the ID, within the OVF, of the appliance

Use cases
User or service POSTs a new OVF to be used with an existing agreed SLA. User or service requests an OVF deployment (i.e., deployment of resources described in the OVF). User or service can obtain an OVF that can be used for an SLA
Description:
A description of an appliance in the form of an OVF document. The OVF can be associated with an SLA.

Resource: image	
Attributes:	*Comment*
creator	
file name	
providers	the providers where the image is available at
file hash	
location	path, store or URL to the image

Use cases
User uploads an image to providers/federation, providing the list of providers and their respective image stores to be used. User stores an image to the GAFS volume, then references the global path for the image to be loaded into the providers' image stores. Cloud provider (a service) provides a status of an image if it is stored locally at the provider.
Description:
Represents a virtual machine image containing the OS and other data necessary for the virtual machine to run. The image may appear on different providers using different image stores, and each such instance has its own instance of the **image** resource, but they all have the same value of the `file hash`. The resource may also describe an image with no specific path or image store. In this case, another instance with the same `file hash` needs to exist on the target provider.

Resource: application	
Attributes:	*Comment*
owner	
status	
running vms	

Resources owned by application:
notification
Use cases
As VMs get deployed with appliances within an SLA, the federation obtains the information on the status of the VMs. The user or service can obtain the status of the application and links to the running appliances (VMs).
Description:
Represents the state of the application.

Resource: usage policy	
Attributes:	*Comment*
cloud provider id	
policy document	

Use cases
A user can set and modify the document defining policies of usage for each cloud provider.

Resource: notification	
Attributes:	*Comment*
event id	the id of the event
notification types	a list of active notification types (e.g., e-mail, web-interface)

Use cases
A user enables the notification of an event (e.g., a job finished) by POSTing to this resource. A user cancels notifications by DELETEing the related instance. To obtain a list of available event ids, the user can GET the notification resource subordinate to an application instance.
Description:
With the notification resources, the users can discover which events they can be notified of as they occur while running an application on the providers. The users can enable specific notifications to be sent to their e-mail or appear in the web interface.

Resource: report	
Attributes:	*Comment*
the report content	A structured output containing the resource usage report.

Use cases
A user gets the resource usage for its account in the past month. A user gets the resource usage for its account in a given time period. A resource provider administrator gets the usage of the resources it provides for a given time period.
Description:
A read-only resource providing access to the resource usage reports.

Resource: cloud provider membership	
Attributes:	*Comment*
cloud provider id	
user credentials	
membership status (active, disabled, expired)	
cost	the cost the user has for being a user of the provider

Use cases
A user can upload pre-existing credentials for the given cloud provider or have the credentials be created for the user. A user can decide whether to actively use the particular provider's resources or exclude it from being selected.

Resource: network	
Attributes:	*Comment*
name	the name of the network
driver	VIN or other network driver
type	fixed or ranged
leases	the list of MAC and IP address pairs
gateway	
dns	
multiprovider	indicates whether this network can span multiple providers (implies VPN between nodes and globally assignable IPs)
provider	the provider owning this network (optional)

Use cases
The resource provider registers the network it provides to the federation. The federation creates a new network based on the OVF created at the **ovf** resource. A user references this network in the appliance definition.
Description:
Represents a virtual network that the appliances can connect to. The model permits two types of network definitions: one is the network describing an actual resource a specific provider is capable of connecting to the virtual machine. A more general representation is not owned by any provider, but contains the information that can be matched to that of a specific provider, and can be therefore used for linking to a federation-level appliance.

Resource: virtual machine	
Attributes:	*Comment*
name	
appliance	reference of the appliance that the virtual machine has been instantiated from
state	the current state of the virtual machine
actions	the sub-resource for invoking actions on the virtual machine

Use cases
Description:
Represents the virtual machines running on a provider as a part of an application. The resource is read-only, except for the `action` attribute which provides the means for changing the state of the virtual machine.

Resource: storage	
Attributes:	*Comment*
providers	list of providers supporting this storage
state	

Use cases
Description:
Represents the availability and the state of the virtual storage (i.e., a set of images and other virtual drives).

Resource: image provider (link)	
Attributes:	*Comment*
image store	the image store on the provider hosting this image
path	the path to the image
state	the stat of the image

Use cases
A user or a service associates an existing image registered at the federation with the provider. A user, service or the provider can disassociate the image from the provider.
Description:
Details how an image registered at the federation level can be accessed at this provider.

B Federation API CLI Reference

B.1 Commands by category

In this appendix we provide a reference of the commands that make up the CLI tool set for manipulating entities in the federation. We grouped the commands into categories based on the entities they influence. At the end of the appendix, section B.2 describes how Contrail essential roles affect the different groups of commands.

B.1.1 Federation users commands

These commands list and manage the information about the users and their profiles at the federation. The commands therefore interface the identity management for the users.

- `cf-users-list`
  returns a list of all users.

- `cf-users-describe user-id`
  provides the details of the user whose user id is `user-id`.

- `cf-users-describe -u username`
  provides the details of the user whose user name is `username`.

- `cf-users-add -u username -p password -G group -r role`
  create a new entry for the user with the user name `username`, assign the password `password`, the user becomes a member of the group `group` and assumes role `role`. The `-G` and `-r` switches may appear multiple times.

- `cf-users-modify user-id [-p new-password] [-add-group new-group] [-remove-group old-group] [-add-role new-role] [-remove-role old-role]`
  modify the entry of the user whose user id is `user-id`. The command changes the attributes related to the switches provided with the command. With the `-add-group` and `-add-role`, the user is added to the `new-group` and `new-role`, respectively. With `-remove-group` and `remove-role`, the user is removed from the `old-group` and `old-role`, respectively.

- `cf-users-modify -u username [-p new-password] [-add-group new-group] [-remove-group old-group] [-add-role new-role] [-remove-role old-role]`
  modify the entry of the user whose user name is `username`. The command changes the attributes related to the switches provided with the command. With the `-add-group` and `-add-role`, the user is added to the `new-group` and `new-role`, respectively. With `-remove-group` and `remove-role`, the user is removed from the `old-group` and `old-role`, respectively.

- `cf-users-delete user-id`
  remove an entry describing the user with the user id `user-id`.

- `cf-users-delete -u username`
  remove an entry describing the user with the user name `username`.

- `cf-users-status user-id`
  returns the current status of the user with id `user-id`.

- `cf-users-status user-id new-status`
  change the status (active, blocked, ...) of the user with the user id `user-id` to status `new-status`.

B.1.2 Cloud provider commands

The following commands manage the view the federation has over the cloud providers. They can be used by the provider administrators in order to register the cloud access points and the related parameters the federation needs to observe when interacting with the provider. The federation coordinators can use the commands to regulate usage policies and overall cloud provider status in the federation.

- `cf-providers-list`
  returns a list of all the registered providers.

- `cf-providers-describe provider-id`
  lists the details of the provider with the provider id `provider-id`.

- `cf-providers-add -n provider-name -a access-point -t provider-type -c credentials-file`
  adds a new entry describing a provider to join the federation. The command requires a supplied provider name `provider-name`, the provider's access point (URL of the API) `access-point`, in `provider-type` it needs to receive whether the provider is `contrail` or `external`, and the `credentials-file` is a local path do the provider's credential that the command will upload with the request.

- `cf-providers-modify [-n new-provider-name] [-a new-access-point] [-t new-provider-type] [-c new-credentials-file]`
  replaces the value of one or more of the referenced attributes with their respective new values: a replacement for supplied provider name `new-provider-name`, the changed provider's access point (URL of the API) `new-access-point`, a change of its type `new-provider-type`, and the local path do the replacement provider's credential `new-credentials-file`.

- `cf-providers-delete provider-id`
  removes the record describing the provider with provider id `provider-id` from

the federation. After removal, the provider can no longer participate in the federation's activities.

- `cf-providers-status provider-id`
  returns the current status of the provider with id `provider-id`.

- `cf-providers-status provider-id new-status`
  changes the status of the provider in the federation to `new-status`. The status can be `active` or `disabled`.

- `cf-providers-policy provider-id`
  returns the current policy assigned to the provider with the provider id `provider-id`.

- `cf-providers-policy provider-id -f new-policy-file`
  assigns the policy contained in the local file `new-policy-file` to be used for the provider with the provider id `provider-id`.

B.1.3 SLA template commands

The following commands can be used for inspecting and managing the SLA templates.

- `cf-slats-list`
  lists the SLA templates that can be used for negotiations.

- `cf-slats-list provider-id`
  lists the SLA templates that can be used for negotiations, offered only by the provider with the id `provider-id`.

- `cf-slats-query constraint-expression`
  returns a list of the SLA templates matching the given `constraint-expr`.

- `cf-slats-describe slat-id`
  returns the contents of the document representing the SLA template with id `slat-id`.

- `cf-slats-add -f slat-file [-p]`
  adds to the list of the available SLA templates the document contained in the local file `slat-file`. If the user issuing the command is a provider's administrator, the optional switch `-p` will cause the template to be associated with this provider only.

- `cf-slats-delete slat-id`
  removes the entry containing the SLA template with id `slat-id`, causing it to no longer be available.

B.1.4 SLA commands

The following commands can be used for inspecting and managing the SLAs in the federation SLA registry.

- `cf-slas-list`
  lists the available SLAs.

- `cf-slas-describe sla-id`
  returns the document describing the SLA referred to with id `sla-id`. Depending on the current status of the SLA, the command returns an active SLA or the latest offer from the federation if the SLA is in the process of being negotiated.

- `cf-slas-delete sla-id`
  terminates and removes the SLA with the id `sla-id`.

- `cf-slas-negotiation-start -f slat-file`
  initiates negotiation with the federation using the SLA template contained in the `slat-file`. The supplied SLA template needs to be a valid template document, filled by the user to express the initial requests. The command returns the id of the created SLA.

- `cf-slas-negotiation-resubmit sla-id -f slat-file`
  the command used when the federation presents an offer in the negotiation process. The contents of the file `slat-file` should represent a template compatible with the one referred to with `sla-id`, and the values should be the next step in the negotiation.

- `cf-slas-negotiation-accept sla-id [-d dd-id]`
  if the `sla-id` refers to an SLA that is in the negotiation with the federation, but has the status accepted, this command confirms the SLA as accepted by the user. The command ends the negotiation. The optional switch `-d` can be used to indicate that the user wants to supply a custom previously submitted deployment document identified as `dd-id`.

- `cf-slas-negotiation-reject sla-id`
  if the `sla-id` refers to an SLA that is in the negotiation with the federation, but has the status accepted, the user rejects the SLA, ending the negotiation.

B.1.5 Appliance commands

The following commands manage the descriptions of the appliances registered at the federation.

- `cf-appliances-list`
  lists the available appliances.

- `cf-appliances-list provider-id`
  lists the appliances available by the provider with ID `provider-id`.

- `cf-appliances-describe appliance-id`
  returns the details of the appliance with id `appliance-id`.

- `cf-appliances-add -f ovf-file [-p]`
  submits a new set of appliances as described in the local OVF `ovf-file`. The command returns a list of IDs of the newly created appliances. If the user is a cloud provider administrator, the optional `-p` switch instructs to relate the new appliances to this provider only.

- `cf-appliances-delete appliance-id`
  removes the appliance referred to as `appliance-id`.

B.1.6 Deployment document commands

The deployment document is a special auxiliary document that can supplement the functionality or features of either OVF documents or the SLA documents. Normally, the federation constructs this document automatically after the negotiation, but the user has an option to supply a custom one. The user can use the following commands to inspect and manipulate the deployment document.

- `cf-dd-list`
  returns a list of the user's own deployment documents.

- `cf-dd-describe dd-id`
  returns the contents of the deployment document with id `dd-id`. The output includes the references (ids) of any OVF and SLA documents that the deployment document is linked to.

- `cf-dd-add -f document`
  submits the contents of the local file `document` as a new deployment document. The command returns the id of the created document.

- `cf-dd-link dd-id [-o ovf-id] [-s sla-id]`
  creates a link between the deployment document with id `dd-id` with an OVF or SLA. If the user supplies the `-o` switch, the document is linked with the OVF with id `ovf-id`. Similarly, if the user supplies the `-s` switch, the deployment document will be associated with the SLA with id `sla-id`.

- `cf-dd-unlink dd-id [-o ovf-id] [-s sla-id]`
  removes links, if they exist, between the deployment document with id `dd-id` with the OVF with id `ovf-id` or the SLA with id `sla-id`.

- `cf-dd-modify dd-id -f new-document`
  replaces the contents of the deployment document identified as `dd-id` with the contents of the local file `new-document`.

- `cf-dd-delete dd-id`
  removes the deployment document identified as `dd-id`. This command is only possible if the deployment document to be removed has been supplied by the user rather than created automatically.

B.1.7 Network commands

These commands enable inspecting and management of the network resources on the federation provided by the cloud providers.

- `cf-networks-list`
  returns a list of the available networks.

- `cf-networks-list provider-id`
  returns a list of the networks available at the provider referred to as `provider-id`.

- `cf-networks-describe network-id`
  returns a description of the network referred to as `network-id`.

- `cf-networks-add -f template-file [-p]`
  creates a record describing the network based on the submitted local network template file `template-file`. The format of the template file depends on the type of network to be offered. The command returns an id of the newly created network. If the user issuing the command is a provider administrator, an optional switch `-p` tells the federation this network is to be offered by this provider only.

- `cf-networks-delete network-id`
  removes the record describing the network referred to as `network-id`, making it unavailable.

- `cf-networks-modify network-id -f template-file`
  modifies the parameters of the network by supplying the new values to take effect in a local file `template-file`.

B.1.8 Storage commands

These commands enable inspecting and management of the storage resources on the federation provided by the cloud providers.

- `cf-storages-list`
  returns a list of the available storages.

- `cf-storages-describe storage-id`
  returns a description of the storage referred to as `storage-id`.

- `cf-storages-add -f template-file [-p]`
  creates a new storage as specified in the local storage template file `template-file`. The format of the template file depends on the type of storage to be offered. The command returns an id of the newly created storage. If the user issuing the command is a provider administrator, an optional switch `-p` tells the federation this storage is to be offered by this provider only.
- `cf-storages-delete storages-id`
  removes the record describing the storages referred to as `storages-id`, making it unavailable.
- `cf-storages-status storage-id`
  returns the current status of the storage referred to as `storage-id`.
- `cf-storages-status storage-id new-status`
  changes the status of the storage with ID `storage-id` to the new status `new-status`.
- `cf-storages-modify storage-id -f template-file` modifies the parameters of the storage `storage-id` by supplying the new values to take effect in a local file `template-file`.

B.1.9 Image commands

The image commands manage the files containing the virtual file systems that can be used by the virtual machines. Using the commands, the users can upload an image to the federation, or register images available for downloads elsewhere (e.g., in the GAFS) for the providers to transparently access.

- `cf-images-list`
  retrieves the list of the available images.
- `cf-images-list provider-id`
  retrieves the list of the images at the provider with id `provider-id` available.
- `cf-images-describe image-id`
  shows the details of the image referred to as `image-id`.
- `cf-images-add -f image-file [-n name] [-d description]`
  upload the contents of the local file `image-file` to become the new image. The command optionally takes a user-friendly name `name` for the image and a short description `description` to be assigned to the image. The command returns an id of the newly created image record.
- `cf-images-add -l image-location [-n name] [-d description]`
  create a new image from the one available at location `image-location`. The location can be an URL to an external store or a file in the GAFS volume. The command optionally takes a user-friendly name `name` for the image and a short

description `description` to be assigned to the image. The command returns an id of the newly created image record.

- `cf-images-modify image-id [-n new-name] [-d new-description]`
  modifies the information of an image with the id `image-id`. If the `-n` switch is provided, the command assigns the new name `new-name` to the image. Similarly, if the `-d` switch is provided, the `new-description` will replace the current description of the image.
- `cf-images-delete image-id`
  deletes the reference of an image with the id `image-id`, making it unavailable.
- `cf-images-status image-id`
  returns the current status of the image referred to as `image-id`.

B.1.10 Virtual machine commands

The following commands provide the means to inspect and manage at the federation level the state of the virtual machines that run on the provider.

- `cf-vms-list`
  returns a list of the VMs.
- `cf-vms-describe vm-id`
  returns the details of the VM referred to as `vm-id`.
- `cf-vms-status vm-id`
  returns the status of the VM referred to as `vm-id`.
- `cf-vms-status vm-id new-status`
  changes the status of the VM referred to as `vm-id` to become `new-status`.
- `cf-vms-deploy appliance-id sla-id`
  instantiates a new VM based on the appliance `appliance-id` and to be deployed as a part of the SLA `sla-id`.

B.1.11 Reporting commands

The following commands enable the users to obtain the resource usage reports from the federation.

- `cf-reports-user [-u user-id] [-t report-type] [-s start-date] [-e end-date]`
  returns a report containing an aggregation showing the user's activities for a given period. If no `-u` switch is provided, the command assumes the current user's id in place of `user-id`. The `report-type` can either be `monthly` or `yearly`, covering the usage in the current month or year, respectively. Instead of `-t`, the

user can provide the start date `start-date` and the end date `end-date` to limit the time span of the report.

- `cf-reports-provider [-p provider-id] [-t report-type] [-s start-date] [-e end-date]`
  returns a report containing an aggregation showing the federation users' activities on the provider's resources for a given period. The usage of the command is similar to that of `cf-report-user`.

B.2 Command usage permissions

The command-line interface enables the full functionality of the federation service, which includes a number of powerful actions. The related commands are subject to access rights check, which occurs within the federation service. However, for convenience, in Table 8 we provide the mapping between the user's role and the availability of the commands for the given role.

In the "Command" column, wherever we find that the same access rules apply for multiple commands, we provide all the commands. For brevity, however, we omit the prefix of the command in all but the first occurrence, since we assume all the commands share the same prefix. For instance, the same access control rules apply for the commands `cf-users-delete` and `cf-users-add`, thus they share a row in the table 8, but in the "Command" column, they are shown as "`cf-users-delete`, `-add`".

The main part of the table shows whether the users of a particular role (Federation coordinator, Provider administrator or Federation user) can use the command in full ("Yes"), with restrictions (defined with a comment) or not at all ("N/A").

Command	Federation coordinator	Provider administrator	Federation user
`cf-users-list`, `-describe`, `-status`, `-modify`	Yes	Own account only	Own account only
`cf-users-delete`, `-add`	Yes	N/A	N/A
`cf-providers-list`, `-describe`, `-modify`, `-status`	Yes	Own record only	N/A
`cf-providers-policy`, `-add`, `-delete`	Yes	N/A	N/A
`cf-slats-list`, `-query`, `-describe`	Yes	Yes	Yes
`cf-slats-add`, `-delete`	Yes	Provider's own SLATs	N/A
`cf-slas-list`, `-describe`	Yes	Yes	Yes
`cf-slas-delete`	N/A	N/A	Own SLAs
`cf-slas-negotiation-start`, `-resubmit`, `-accept`, `-reject`	N/A	N/A	New, own SLAs

Command	Federation coordinator	Provider administrator	Federation user
`cf-appliances-list, -describe, -add`	Yes	Yes	Yes
`cf-appliances-delete`	Yes	Provider's appliances	Own appliances
`cf-networks-list -describe`	Yes	Yes	Yes
`cf-networks-add, -delete, -modify`	Yes	Provider's networks	N/A
`cf-storages-list -describe`	Yes	Yes	Yes
`cf-storages-add, -delete, -modify`	Yes	Provider's storages	N/A
`cf-images-list, -describe, -status` (reading)	Yes	Images on provider or federation	Public and own images
`cf-images-add, -delete, -status` (changing)	Yes	Provider's images	New or own images
`cf-vms-list, cf-vms-describe, -status` (reading)	N/A	VMs running on provider	Own VMs
`cf-vms-deploy, -status` (changing)	N/A	N/A	Own VMs
`cf-reports-user`	Yes	Usage on provider	Own usage
`cf-reports-provider`	Yes	Own Usage	N/A

Table 8: Availability of the commands to the users based on their roles

References

[1] Ed. (JanRain); N. Sakimura (NRI) D. Recordon (Six Apart); M. Jones Microsoft); J. Bufu, Ed. (Independent); J. Daugherty. OpenID Provider Authentication Policy Extension 1.0, December 2009. `http://openid.net/specs/openid-provider-authentication-policy-extension-1_0.html`.

[2] Marco Dorigo and Christian Blum. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2-3):243 – 278, 2005.

[3] Stephan Fowler. HTTPsec Authentication Protocol. WWW, September 2006. `http://www.httpsec.com/1.0/` – original URL retrieved via `web.archive.org`.

[4] Fred Glover. Tabu search: A tutorial. *The Practice of Mathematical Programming*, 20(4):74–94, July-August 1990.

[5] GNU Linear Programming Kit. Web Site. `http://www.gnu.org/software/glpk/`.

[6] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1 edition, January 1989.

[7] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics – A Foundation for Computer Science*. Addison-Wesley, 3rd edition, May 1989.

[8] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[9] S. Tuecke (ANL); V. Welch (NCSA); D. Engert (ANL); L. Pearlman (USC/ISI); M. Thompson (LBNL). Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile, June 2004. `http://www.ietf.org/rfc/rfc3820.txt`.

[10] Thijs Metsch and Andy Edmonds. Open Cloud Computing Interface - Infrastructure, May 25, 2011. `http://forge.gridforum.org/sf/go/doc16265?nav=1`.

[11] Thijs Metsch and Andy Edmonds. Open Cloud Computing Interface - RESTful HTTP Rendering, May 25, 2011. `http://forge.gridforum.org/sf/go/doc16263?nav=1`.

[12] John Kemp (Nokia); Scott Cantor (Internet2); Prateek Mishra (Principal Identity); Rob Philpott (RSA Security); Eve Maler (Sun Microsystems). Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0, March 2005. `http://docs.oasis-open.org/security/saml/v2.0/saml-authn-context-2.0-os.pdf`.

[13] OAuth web site. `http://oauth.net/`.

[14] Open Virtualization Format Specification. DMTF Standard, 2009. `http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_1.0.0.pdf`.

[15] CONTRAIL Project. Deliverable D10.1 - First specification of the system architecture, August 2011.

[16] CONTRAIL Project. Deliverable D2.1 - Requirements on Federation Management, Identity and Policy Management in Federations, March 2011.

[17] CONTRAIL Project. Deliverable D3.2 - SLA Management Services Terms and Initial Architecture, August 2011.

[18] CONTRAIL Project. Deliverable D7.1 - Security Requirements, Specification and Architecture for Virtual Infrastructures, March 2011.

[19] SLA@SOI Project. Deliverable D.A5a, Foundations for SLA Management. `D.A5a-M26-SLAManagementFoundations.pdf`.

[20] George Reese. The Good, The Bad, The Ugly of REST, June 2011. `http://broadcast.oreilly.com/2011/06/the-good-the-bad-the-ugly-of-rest-apis.html`.

[21] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Series. O'Reilly, 2007.

[22] O. Sinnen and L. Sousa. A Classification of Graph Theoretic Models for Parallel Computing. Technical Report RT/005/99, Instituto Superior TÃl'cnico, Technical University of Lisbon, Lisbon, Portugal, May 1999.

[23] SLA@SOI Consortium Main Web Site. `http://sla-at-soi.eu/`.

[24] SLA@SOI SLA Model. `http://sourceforge.net/apps/trac/sla-at-soi/wiki/SLA%20Model%20%3A%20The%20Model`.

[25] W. Timothy Polk William E. Burr, Donna F. Dodson. NIST Special Publication 800-63, Electronic Authentication Guideline, April 2006. `http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63V1_0_2.pdf`.

[26] Apache Xerces web site. `http://xerces.apache.org/`.