



## D1.4

# SUPERCLOUD Self-Management of Security Implementation

<b>Project number:</b>	643964
<b>Project acronym:</b>	<b>SUPERCLOUD</b>
<b>Project title:</b>	User-centric management of security and dependability in clouds of clouds
<b>Project Start Date:</b>	1st February, 2015
<b>Duration:</b>	36 months
<b>Programme:</b>	H2020-ICT-2014-1
<b>Deliverable Type:</b>	Demonstrator
<b>Reference Number:</b>	ICT-643964-D1.4/ 1.0
<b>Work Package:</b>	WP 1
<b>Due Date:</b>	JUL 2017 - M30
<b>Actual Submission Date:</b>	31st July, 2017
<b>Responsible Organisation:</b>	IMT
<b>Editor:</b>	Reda Yaich, Nora Cuppens, Frédéric Cuppens
<b>Dissemination Level:</b>	PU
<b>Revision:</b>	1.0
<b>Abstract:</b>	This report describes the implementation components that constitute the Self-Management of Security. For each component, a basic description of its functioning mechanism and the interfaces needed to communicate with it are provided.
<b>Keywords:</b>	Self-Management, Security, SSLA, Negotiation, Autonomic



This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 643964.

This work was supported (in part) by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0091.

This document has gone through the consortiums internal review process and is still subject to the review of the European Commission. Updates to the content may be made at a later stage.

## **Editor**

Reda Yaich, Nora Cuppens, Frédéric Cuppens (IMT)

## **Contributors (ordered according to beneficiary numbers)**

Reda Yaich, Nora Cuppens, Frédéric Cuppens (IMT)

Marc Lacoste, Sébastien Canard (Orange)

Alysson Bessani, Fernando Ramos, Nuno Neves (FC.ID)

Krzysztof Oborzyński (PHHC)

Daniel Pletea (PEN)

Marko Vukolic (IBM)

## **Disclaimer**

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The users thereof use the information at their sole risk and liability.

This document has gone through the consortiums internal review process and is still subject to the review of the European Commission. Updates to the content may be made at a later stage.

## Executive Summary

This Deliverable describes the implementation of the SUPERCLOUD Security Self-Management Framework. We present the specification and development of security services that are needed by Cloud Service Customers (CSC) to define, control and manage the required level of protection over *compute*, *storage* and *network* planes. An important part of the Deliverable is also dedicated to the presentation of Security Service Level Specification and Negotiation Platform. In addition, we report on the integration of security services together as well as with *compute*, *data* and *network* components. Finally, we showcase the integration of Security Self-Management with SUPERCLOUD project use-cases demonstrators (i.e., Philips Imaging Platform).

# Contents

<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Motivation and Objectives . . . . .	1
1.2 Document Organization . . . . .	1
<b>Chapter 2 Security Self-Management Framework</b>	<b>2</b>
2.1 Overview of Security Self-Management . . . . .	2
2.2 Security Self-Management Approach Specification . . . . .	4
2.2.1 Micro-Services Architecture . . . . .	4
2.2.2 Docker for Micro-Services Deployment . . . . .	4
2.2.3 Security Self-Management Dockerized Architecture . . . . .	5
<b>Chapter 3 Security Service Level Management</b>	<b>6</b>
3.1 SSLA Specification Service . . . . .	6
3.1.1 Accounts Management . . . . .	6
3.1.2 Service and Protection Level Expression . . . . .	8
3.2 SSLA Negotiation Platform . . . . .	11
3.2.1 Multi-Agent SSLA Negotiation Platform . . . . .	11
3.2.1.1 Agent Creation . . . . .	12
3.2.1.2 Cloud Services Publishing and Discovery . . . . .	13
3.2.1.3 Cloud Services Negotiation . . . . .	15
<b>Chapter 4 Implementation of Security Self-Management</b>	<b>17</b>
4.1 Security Self-Management Framerwork . . . . .	18
4.1.1 Overall Security Orchestrator . . . . .	18
4.2 Security Self-Management Services . . . . .	23
4.2.1 Authorization Service . . . . .	23
4.2.2 Monitoring Service . . . . .	24
4.2.3 Software Trust Management Service . . . . .	25
4.2.4 SSLA Enforcement Service . . . . .	26
4.3 Compute-Level Security Services . . . . .	27
4.4 Data-Level Security Services . . . . .	28
4.5 Network-Level Security Services . . . . .	29
<b>Chapter 5 Integration of Security Services</b>	<b>31</b>
5.1 Integration with Compute Security Services . . . . .	32
5.2 Integration with Data Security Services . . . . .	32
5.2.1 Location-Awareness Policies for SLAs . . . . .	32
5.2.2 Monitoring of Data Access Failures . . . . .	33
5.3 Integration with Network Security Services . . . . .	34
5.4 Integration with Use-Cases Application . . . . .	34
<b>Chapter 6 Components Access and Installation</b>	<b>35</b>
6.1 SSLA Specification Platform . . . . .	35
6.2 SSLA Negotiation Platform . . . . .	35
6.3 Security Orchestrator . . . . .	35
6.4 Security Services . . . . .	36
6.4.1 Authorization Service . . . . .	36

6.4.2	Monitoring Service . . . . .	37
6.4.3	SSLA Services . . . . .	37
6.4.4	Trust Management Service . . . . .	37
<b>Chapter 7</b>	<b>Conclusion</b>	<b>38</b>
7.1	Summary . . . . .	38
7.2	Future Works . . . . .	38
	<b>List of Abbreviations</b>	<b>40</b>
	<b>Bibliography</b>	<b>40</b>

## List of Figures

2.1	Overview of Self-Management of Security . . . . .	2
2.2	Architecture of Security Self-Management components . . . . .	3
2.3	Micro Services Approach for Security Self-Management . . . . .	5
3.1	Profile Selection in the SSLA Interface . . . . .	6
3.2	SSLA Specification Authentication Page . . . . .	7
3.3	Amazon and Google Cloud Offer List . . . . .	7
3.4	Cloud Providers Management . . . . .	7
3.5	Setup of a Cloud Service Provider Offer . . . . .	8
3.6	Location Configuration in Cloud Service Offer . . . . .	8
3.7	Compute Service and Protection Level Setting . . . . .	9
3.8	Storage Service and Protection Level Settings . . . . .	10
3.9	Network Service and Protection Level Settings . . . . .	10
3.10	Request/Offer XML File Generation . . . . .	10
3.11	Overview of the SUPERCLOUD Agent-Based Market Place . . . . .	11
3.12	Introduction of a Negotiation Agent into the SUPERCLOUD Market Place . . . . .	12
3.13	Configuration of the Negotiation Agent . . . . .	12
3.14	Discovery of Cloud Services . . . . .	13
3.15	Publishing Message Content . . . . .	14
3.16	Call for Cloud Services Proposal . . . . .	15
4.1	Overview of SUPERCLOUD Security Services . . . . .	17
4.2	Overview of the Overall Orchestrator . . . . .	18
4.3	Security Orchestrator Workflow . . . . .	19
4.4	Illustration of security services deployment . . . . .	21
4.5	Deployed security services . . . . .	22
4.6	Integration of the Authorization Service with an Application or a Service . . . . .	23
4.7	Security monitoring: approach . . . . .	25
4.8	Overview of Software Trust Service . . . . .	26
4.9	SSLA Enforcement Service . . . . .	27
4.10	Captures from the SSLA Reporting Dashboard . . . . .	28
5.1	Security Services Integration Approach . . . . .	31
5.2	Integration of SSLA Management and Georeplication Services . . . . .	32
5.3	SSLA Location-Aware Janus . . . . .	33
5.4	Integration of Self-Management of Security and Janus . . . . .	33
5.5	OrBAC-based orchestration of network security policies . . . . .	34

## List of Tables

4.1 Authorization Service API . . . . .	24
---	----

# Chapter 1 Introduction

## 1.1 Motivation and Objectives

This Deliverable reports on the implementation of the SUPERCLOUD Security Self-Management Framework that was specified in Deliverable D1.2 [23]. The document presents the implementation of services that are needed by Cloud Service Customers (CSC) to define, control and manage the required level of protection over *compute*, *storage* and *network* planes.

The document also describes the implementation of a global negotiation platform for Security Service Level Agreement (SSLA). The platform relies on software agents to conduct Cloud Services Negotiation on behalf of a single customer across providers, and on behalf of a single provider across customers.

We present the integration of the Security Self-Management Framework and corresponding services with security services developed as part of Compute, Data and Network SUPERCLOUD sub-frameworks. We show also how security services of the Security Self-Management Framework are well integrated with the SUPERCLOUD project use-cases demonstrators (i.e., Philips Imaging Platform).

## 1.2 Document Organization

The rest of the document is organized as follows. We first present in Chapter 2 an overview of the Security Self-Management Architecture and the corresponding specification and deployment approaches.

Then, in Chapter 3, we present the SSLA Specification and Negotiation framework that captures Cloud Services Customers (CSCs) and Cloud Services Providers (CSPs) requirements used to negotiate cloud services among CSCs and CSPs.

In Chapter 4, we provide insight on the implementation of Security Orchestrators and Services.

In Chapter 5, we report on the integration of Security Self-Management Services with the SUPER-CLOUD Compute, Data and Network planes.

Finally, in Chapter 6, we present how the services presented in this Deliverable can be downloaded and deployed, before we conclude in Chapter 7.



## Chapter 2 Security Self-Management Framework

In this Chapter, we provide a brief introduction to the Security Self-Management Framework. In this introduction, we first highlight general design principles, and then describe the specification of the Security Self-management Architecture.

### 2.1 Overview of Security Self-Management

In Figure 2.1 depicted below, *Cloud Service Customers (CSCs)* interact with a *SUPERCLOUD Front-end* (i.e., the Frontal) to express their requirements and constraints (cf. Section 3.1). After a negotiation phase (cf. Section 3.2), a Security Service Level Agreement (SSLA) is established to define the expected quality of service and level of protection. The objective and motivation of this process are described in details in Deliverable D1.2 [23].

Based on this SSLA, a U-Cloud (User/Customer-Specific Cloud) is created over a single or multiple providers as illustrated in Figure 2.1. The hypervisor shown in the Figure refers to the SUPERCLOUD compute and network virtualization frameworks. The implementation of these SUPERCLOUD sub-frameworks have been presented in Deliverable D2.3 [10] and Deliverable D4.3 [19]. On the top of these sub-layers, automation of management and control of security services is delegated to the Security Self-Management Framework.

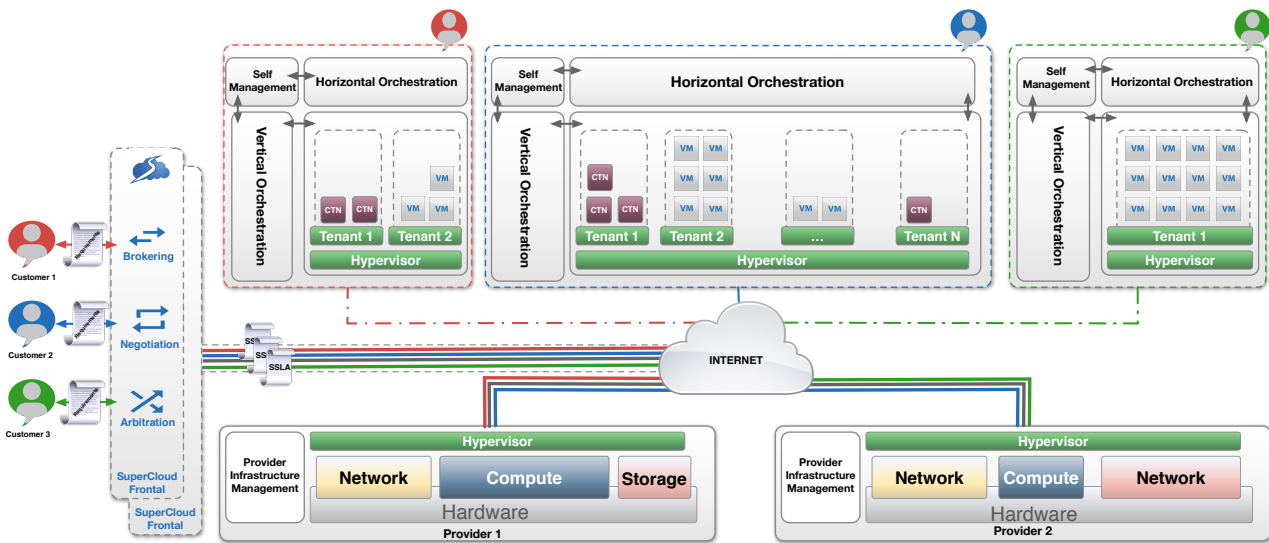


Figure 2.1: Overview of Self-Management of Security

As shown in Figure 2.2, the *Security Self-Management Framework* is structured into four sub-systems, namely (1) the *SSLA Specification and Negotiation Service*, (2) the *Orchestration Service*, (3) the *overall (i.e., Cross-Plane) Security Services*, and (4) *Plane-Specific Security Services*.

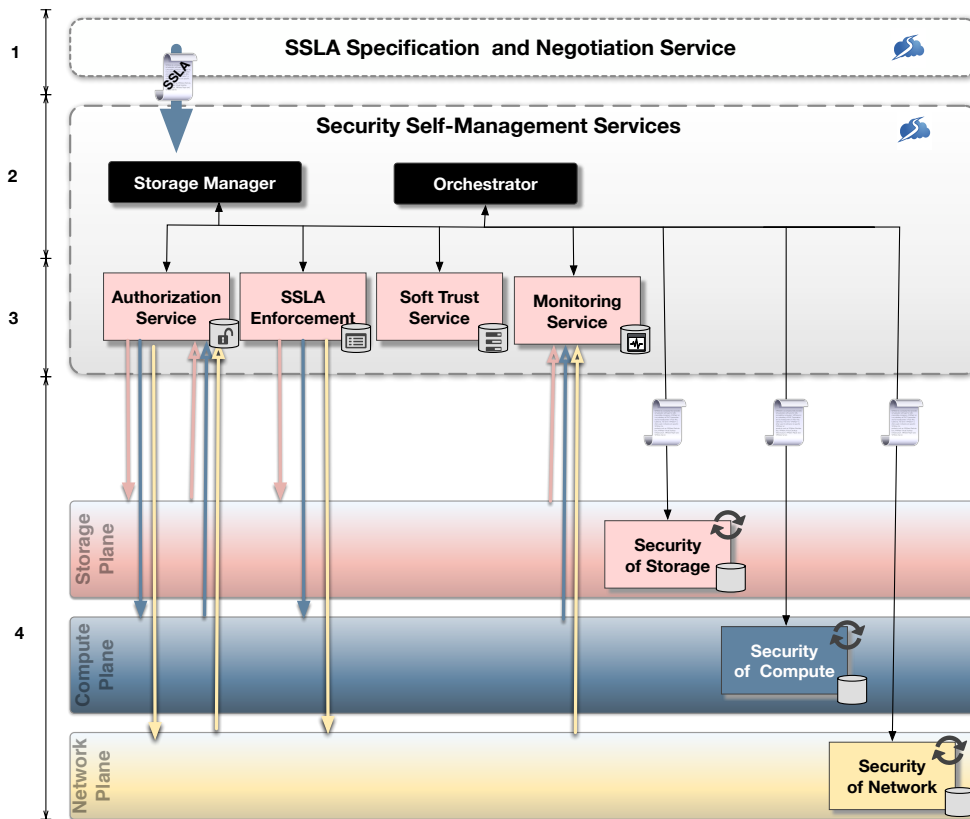


Figure 2.2: Architecture of Security Self-Management components

1. The *SSLA Specification and Negotiation Service* encapsulates the services implemented in SUPERCLOUD to capture and negotiate Protection Level and Service Level requirements and constraints expressed by Cloud Customer and Cloud Providers.
2. The *Orchestration sub-system* contains essentially the Orchestrator. This service coordinates the configuration, deployment and management of security services across planes. This is achieved through the interpretation and exploitation of Security Service Level Agreements. The Storage Service is part of the orchestration sub-system as it is in charge of the persistence of SSLA, policies and monitoring information.
3. On the top of Plane-Specific Security Services, the *Overall (Cross-Plane) Security Services* provide a unified and uniform view of security services across *compute*, *storage* and *network* planes. For instance, the Authorization Service manages access control and usage control features over the three planes. Similarly, Monitoring, SSLA Enforcement and Software Trust Services operate at the same time over the *compute*, *storage* and *network* planes.
4. At the lowest level of the architecture, we can find the *Plane-Specific Security Services*. These security services operate on a particular layer of the virtualization infrastructure (i.e., storage, compute and network planes). As illustrated in Figure 2.2, Plane-Specific Security Services are dedicated to a specific SUPERCLOUD plane. For instance, in the Network Plane, we can find services providing Deep Packet Inspection (DPI), Network Intrusion Detection (NIDS), or network DDoS mitigation – see Deliverable D4.3 [19] for more details. Similarly, both the Compute Plane and the Storage Plane possess specific security services. Detailed description of Plane-Specific Security sub-systems can be found in Deliverables D2.3 [10], D3.3 [2] and D4.3 [19]. A review of these services is also provided in Sections 5.1, 5.2 and 5.3.

## 2.2 Security Self-Management Approach Specification

In this Section, we present the approach specification and the technical choices that have been made during the implementation of the SUPERCLOUD Security Self-Management Framework. In the workflow we adopted to specify, develop, integrate and test the Security Self-Management service presented in Figure 2.2, adaptive and user-centric properties were given the highest consideration. We have particularly taken into account the following guidelines:

- The SUPERCLOUD front-end portal needs to be accessed through different devices (Desktop, Laptop and smartphone). Thus a web-based frontal is preferable.
- The Security Self-Management Framework needs to be developed in a modular way to enable portability, reuse and extensibility.
- Security Services are developed by different project partners, each with specific requirements in terms of programming languages and paradigms. Thus the implementation of Security Self-Management Services needs to be technology- and language-agnostic.

To reach the aforementioned objectives, we decided to adopt an agile software development approach based on *Micro-Services*. In what follows, we provide a brief presentation of the Micro-Services approach. Then, we describe the micro-services compliant SUPERCLOUD Security Self-Management Architecture.

### 2.2.1 Micro-Services Architecture

*Micro-services* is an architecture paradigm that splits down large and complex applications into multiple services. The particularity of this approach is that services can be deployed independently of one another and are loosely coupled. The other particularity of the micro-services approach is that each service is task-specific: services are focusing on completing single and different system sub-objectives in an efficient way. Additionally, micro-services are developed in any programming language and their interactions are performed using language-agnostic application programming interfaces (APIs) such as Representational State Transfer (REST).

As presented in Section 2.1, the SUPERCLOUD Security Self-Management Framework consists of numerous services called “*Security Services*” in the project. Like Micro-Services, Security Services are task-specific and system-agnostic. In other words, they do not know (and do not need to know) the system overall logic to function. They are implemented independently using different programming languages and approaches. At deployment phase, security services get composed by the Security Orchestrator in order to form the SUPERCLOUD Security Self-Management Framework. Thus, each security service can be refactored/replaced with less effort as services are loosely coupled. In addition, security services do not depend on each other to run, making their isolated deployment possible and easy to achieve. For a detailed description of security services objectives and functioning, we refer the reader to Deliverable D1.2 [23].

### 2.2.2 Docker for Micro-Services Deployment

To deploy SUPERCLOUD Security Self-Management Services, we selected Docker Tool [4] state-of-the-art technology for the deployment of the Security Micro-Services architecture presented in Section 2.2.1. The SUPERCLOUD Security Self-Management Framework is a collaborative software that include several services. Each security service is developed by a different partner having specific preferences and requirements for processing and operating environment. Thus the encapsulation of Security Services into self-contained, ready-to-deploy containers is a key issue for the successful integration of all services. All that security services’ developers need to do is to expose interaction interfaces to allow dynamic security services composition. Integration examples are provided in Chapter 5.

### 2.2.3 Security Self-Management Dockerized Architecture

In this Section, we present the SUPERCLOUD Security Self-Management architecture that corresponds to the Dockerization<sup>1</sup> of security services.

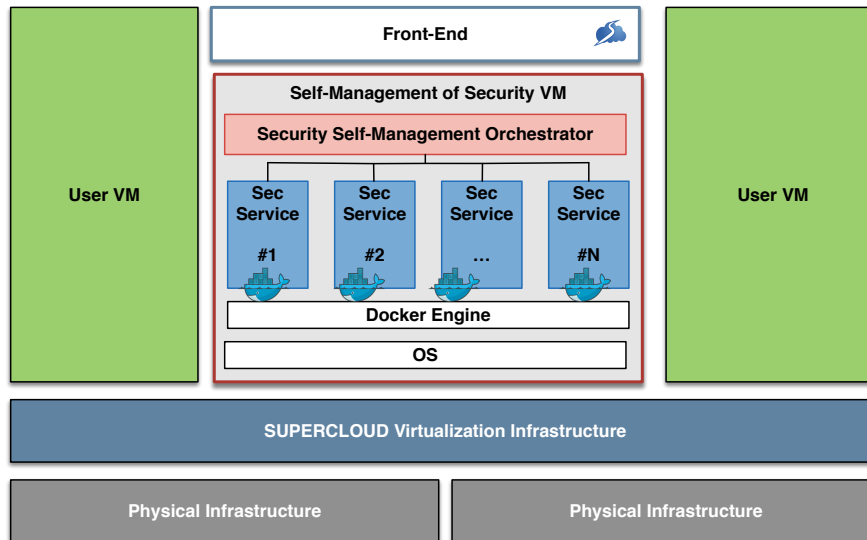


Figure 2.3: Micro Services Approach for Security Self-Management

As illustrated in Figure 2.3, the whole system is split into separate self-contained security services, each realizing an individual security objective. The objectives are directly derived from the Security Service Level Agreement specified and negotiated through the SUPERCLOUD Front-end. The translation of SSLA objectives into Security Services deployment and configuration instructions is achieved by the Security Orchestrator. Thus the interaction and communication occur primarily between the front-end and the Security Orchestrator, allowing retrieval of SSLA objectives. The Orchestrator then parses the file to extract the list of services to be deployed. The Orchestrator also coordinates the automated configuration and deployment of each security service.

To implement this architecture, the following guidelines have been followed to ensure high-compatibility and easy integration between the different parts of the architecture:

- The SUPERCLOUD frontal is hosted on a web-server Docker container.
- Each security service is developed independently and encapsulated in a Docker container.
- Orchestration and coordination between services is achieved based on Docker Compose [3].
- Services communicate over REST APIs and links between them are set by the Security Orchestrator at deployment phase.

The rest of this Deliverable provides a generic view of the development and deployment of Security Self-Management Services.

<sup>1</sup>i.e., running in a Docker container.

## Chapter 3 Security Service Level Management

In this Chapter, we will present the SUPERCLOUD Front-End service. This service encapsulates the mechanisms developed in the SUPERCLOUD project to capture and manage Security Service Level Agreements (SSLAs). These mechanisms are part of the Security Service Level Agreement Specification and Negotiation Service presented in Figure 2.2.

### 3.1 SSLA Specification Service

In this Section, we present the Security Service Level Agreement Specification Service. The objective of this service is to assist both Cloud Service Customers (CSCs) and Cloud Service Providers (CSPs) in the expression of Cloud Service Requests (for CSCs) and Offers (for CSPs).

#### 3.1.1 Accounts Management

As depicted in Figure 3.1, the SSLA specification service is used by both CSP and CSC. Hence, the first step consists in selecting the type of profile the user of the service wants to adopt. This step is subsequently followed by an authentication phase, e.g. via a classical user/password form, as illustrated in Figure 3.2.

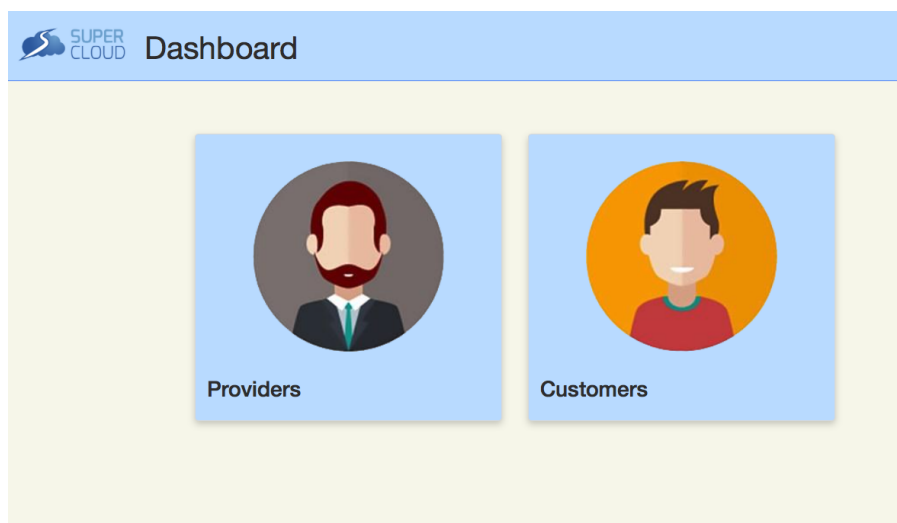


Figure 3.1: Profile Selection in the SSLA Interface

Once authenticated, both CSP and CSC will have access to their list of offers/requests. In Figure 3.3, we illustrate some examples of list of offers from Amazon and Google CSPs.

If a CSP or a CSC is not available in the system, we can create it through a specific administration section. In Figure 3.4, we can see a list of providers along with the ability of adding new ones that are not in the list.

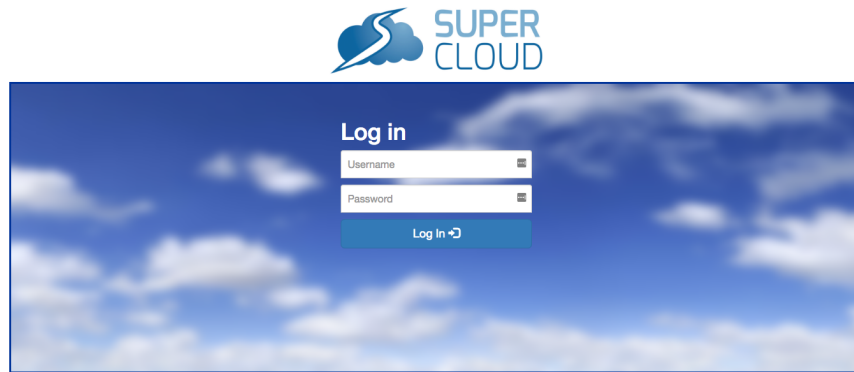


Figure 3.2: SSLA Specification Authentication Page

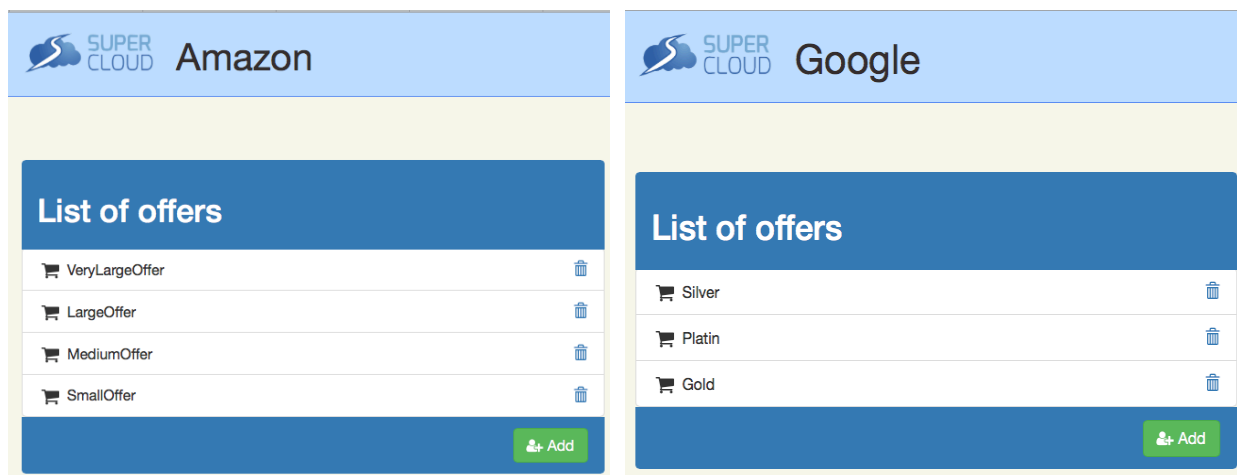


Figure 3.3: Amazon and Google Cloud Offer List

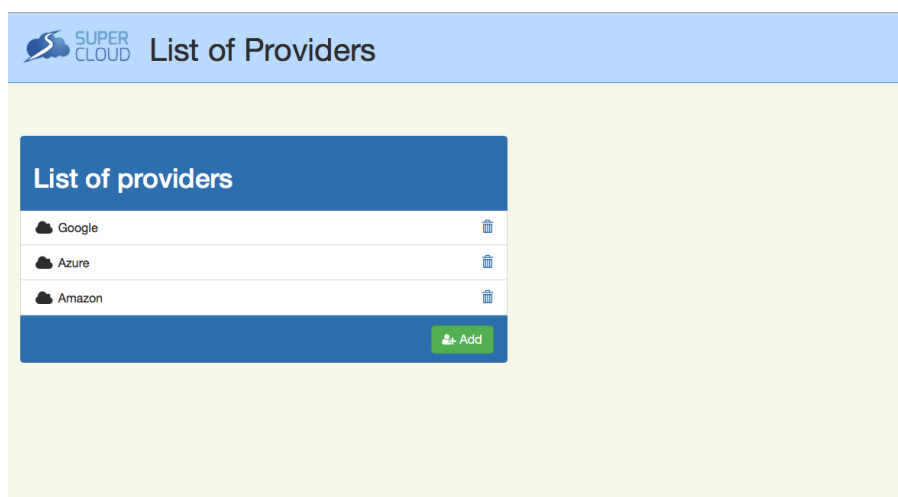


Figure 3.4: Cloud Providers Management

### 3.1.2 Service and Protection Level Expression

Once logged-in, Cloud Service Providers and Cloud Service Customers will have the ability to create, update and delete their respective requests/offers. We illustrate this Section with offers managed by CSPs, but requests expressed by CSCs are made in an analogous way.

Each offer/request is structured into three sections and a header. Sections are dedicated to each of the SUPERCLOUD planes (i.e., *Compute*, *Storage* and *Network*), while the header contains general terms that apply to all three sections. Figure 3.5 illustrates the addition of a new offer from *Amazon*. Each offer needs to have a name and a price.

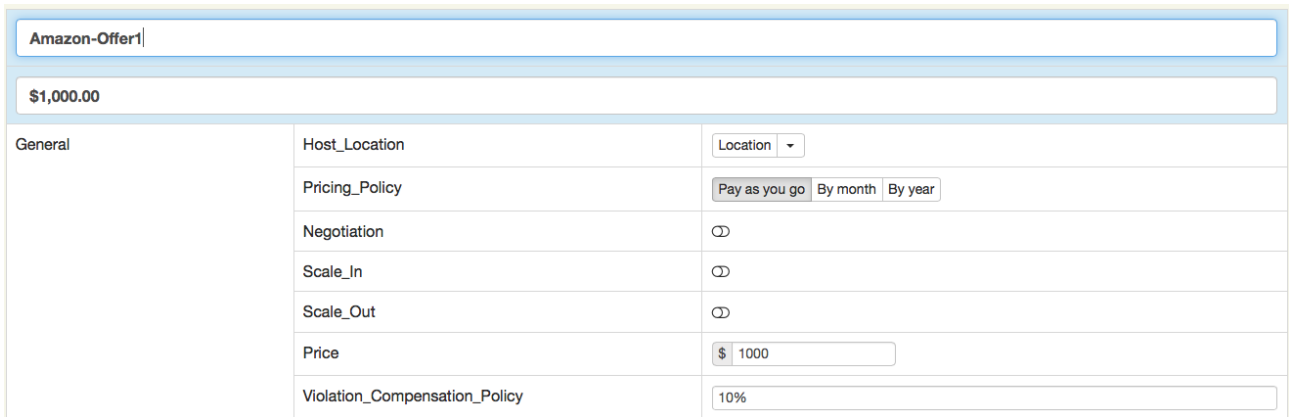


Figure 3.5: Setup of a Cloud Service Provider Offer

The price is calculated based on the price value fixed in the header in addition to the individual prices of each of the options added in the offer. Along with the price, a CSP can set:

- A pricing policy: pay-as-you-go, by month, by year, etc.
- If the offer is fixed or can be negotiated.
- If the offer can scale in and/or out.
- A default SSLA violation compensation policy.

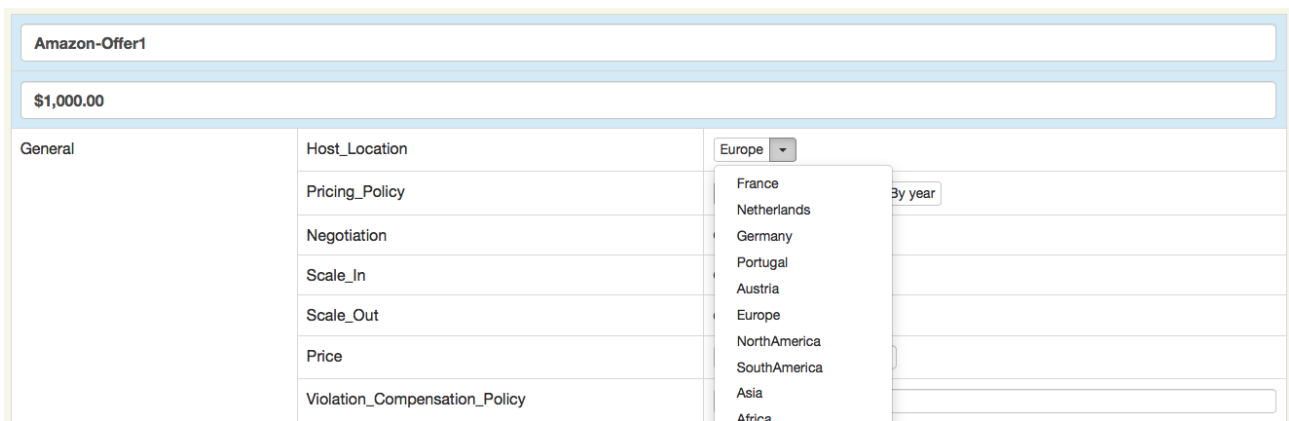


Figure 3.6: Location Configuration in Cloud Service Offer

As *Location-Awareness* and *Location-Control* have been identified in Deliverables D1.2 [23] and D5.1 [11] as key requirements, and in compliance with European Commission Regulation, Location settings are

included systematically in all offers. Thus, providers need to make explicit in their offers the geographical localization of the machines hosting the services. This location selection is illustrated in Figure 3.6.

Once Cloud Services general terms have been specified, the SSLA Specification Service allows CSPs to tune their offers in terms of compute, storage and network capabilities.

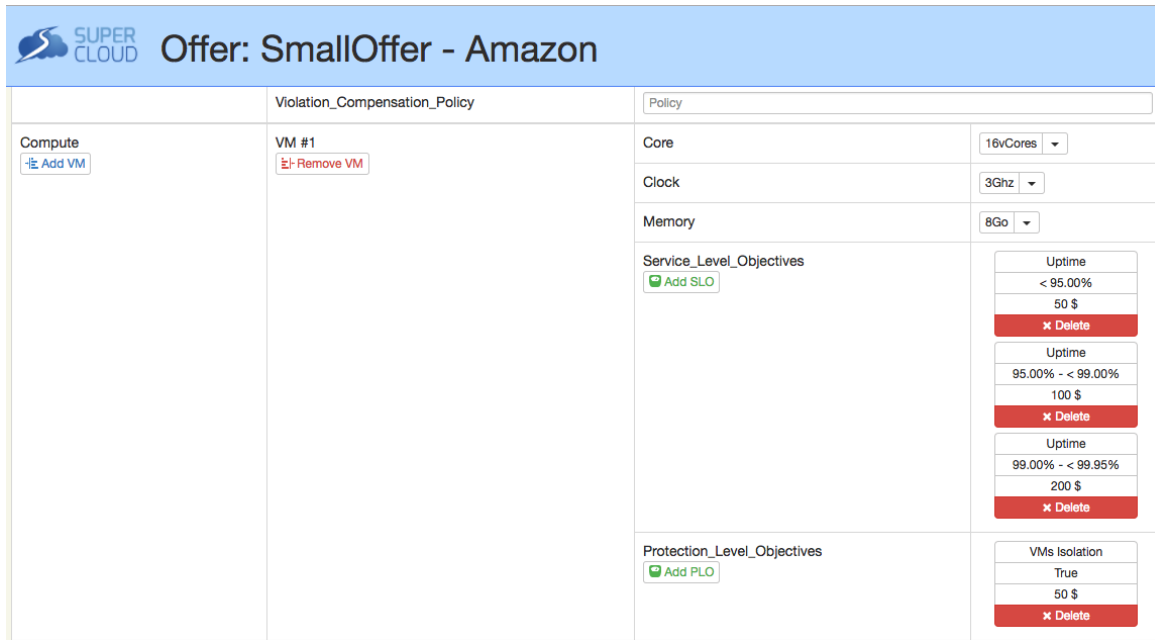


Figure 3.7: Compute Service and Protection Level Setting

In Figure 3.7, each offer is composed of one or many VMs. Each VM is defined in terms of processing capabilities: Virtual Cores, Clock Speed and RAM Memory. Most interestingly, compute-level Service Level Objectives (SLOs) and Protection Level Objectives (PLOs) are specified. In the illustrative example depicted in Figure 3.7, three *uptime* SLO options are possible, each with a specific pricing policy. Similarly, the offer specifies which Security Self-Management services are available and can be deployed at the compute level. In this example, VMs isolation is made available for CSCs.

The specification of *storage* requirements is performed similarly. In Figure 3.8, we provide an example of storage options (e.g., capacity, disk type) and associated Quality of Service (QoS) and Quality of Protection (QoP). Network offer specification is achieved in a similar manner. We provide in Figure 3.9 an example of Network offer specification.

Once the three aspects of the cloud offer have been specified, the cloud provider (and the customer as well) can either save the offer/request to be completed later or generate the XML file that will be used to perform cloud service transactions. The persistence of offers/request is achieved using the "Generate XML" button depicted in Figure 3.10.

The generated XML file will be used later by the SSLA Negotiation Platform to match CSC request with CSP offers. This process is described in the next Section.



SUPER CLOUD Offer: SmallOffer - Amazon			
		Protection_Level_Objectives <input type="button" value="Add PLO"/>	VMs Isolation True 50 \$ <input type="button" value="Delete"/>
Storage <input type="button" value="Add data store"/>	Data Store #1 <input type="button" value="Remove data store"/>	Capacity 10Go Type HD Service_Level_Objectives <input type="button" value="Add SLO"/>	Availability 99.999% 500 \$ <input type="button" value="Delete"/> Availability 99.99% 100 \$ <input type="button" value="Delete"/> Encryption True 100 \$ <input type="button" value="Delete"/>
		Protection_Level_Objectives <input type="button" value="Add PLO"/>	Encryption True 100 \$ <input type="button" value="Delete"/>

Figure 3.8: Storage Service and Protection Level Settings

SUPER CLOUD Offer: SmallOffer - Amazon			
		<input type="button" value="Add PLO"/>	True 100 \$ <input type="button" value="Delete"/>
Network <input type="button" value="Add virtual network"/>	Virtual Network #1 <input type="button" value="Remove virtual network"/>	Bandwidth 1Gbps Service_Level_Objectives <input type="button" value="Add SLO"/>	Latency 10ms 40 \$ <input type="button" value="Delete"/> IDS True 30 \$ <input type="button" value="Delete"/> Cloud Type Private 200 \$ <input type="button" value="Delete"/> Cloud Type Public 100 \$ <input type="button" value="Delete"/>
		Protection_Level_Objectives <input type="button" value="Add PLO"/>	Encryption True 100 \$ <input type="button" value="Delete"/>

Figure 3.9: Network Service and Protection Level Settings

Figure 3.10: Request/Offer XML File Generation

## 3.2 SSLA Negotiation Platform

In this Section we present the implementation of the SSLA negotiation platform used to match CSCs requests with CSPs offers. The platform has been implemented in Java using a Multi-Agent paradigm.

### 3.2.1 Multi-Agent SSLA Negotiation Platform

In the SUPERCLOUD project demonstrator, we make use of the JADE (Java Agent Development) [8] Framework for the implementation of our multi-agent based negotiation framework. JADE offers a stable middleware that complies with the FIPA<sup>1</sup> specifications and offers tools that support the debugging and deployment of agents. Agents are reactive and proactive software programs. They are able to achieve “*autonomously*”, and eventually “*collectively*”, a set of actions to fulfill a goal. Goals can be self-assigned, but generally they are set by the humans that these agents are assisting, or behaving on their behalf. Besides the agent abstraction, JADE provides a simple, yet powerful peer-to-peer agent communication based on the asynchronous message-passing paradigm. JADE offers also a yellow pages service called Directory Facilitator to support offers and requests publishing, subscription and discovery. The general overview of the Multi-Agent Negotiation Platform is depicted in Figure 3.11.

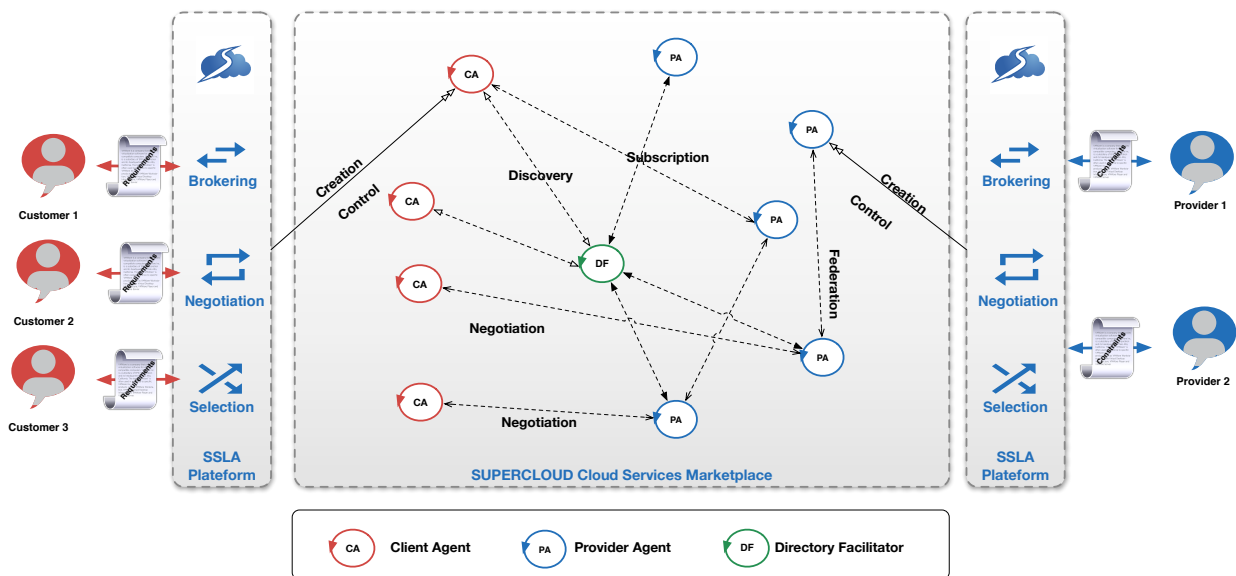


Figure 3.11: Overview of the SUPERCLOUD Agent-Based Market Place

In Figure 3.11, we identify Customers (on the left) that make use of the SSLA Specification Platform, described in Section 3.1, to formulate their requirements. Similarly, Cloud Providers (on the right) make use of the same platform to specify their constraints. The generated XML file is used by a dedicated Negotiation Agent to find the best provider/customer. Negotiation agents are split into two types : Customers Agents (CA) and Provider Agents (PA). CA are devoted to the fulfillment of Customers requests, while PA negotiate offers on the behalf of Cloud Providers. Agents interact with each other using messages that are mediated by the JADE asynchronous message passing mechanism. As illustrated in Figure 3.11, the Directory Facilitator (DF) plays a central role within the market place. It allows agents offering cloud services (i.e., Provider Agents) to subscribe to the list used by Customers Agents to find cloud service offers.

<sup>1</sup>The standards organization for agents and multi-agent systems.

In what follows, we present the negotiation workflow from the moment agents are created to the stage the agreement is reached.

### 3.2.1.1 Agent Creation

In Figure 3.12, we show the agent’s instantiation interface that allows both Cloud Customers and Cloud Provider to create and configure an agent to be run on the market place.

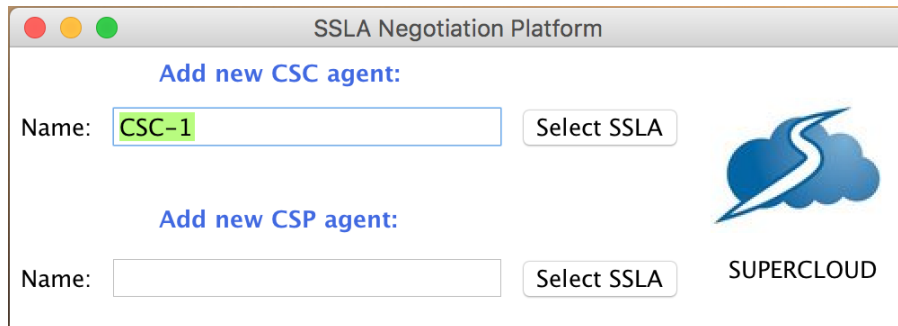


Figure 3.12: Introduction of a Negotiation Agent into the SUPERCLOUD Market Place

To create an agent, we need first to provide the agent name. This name makes the agent uniquely identified within the platform. The after pressing the "Select SSLA" button, a dialog window will pop-up letting the user select the XML file generated with the SSLA Specification Service presented in Section 3.1.

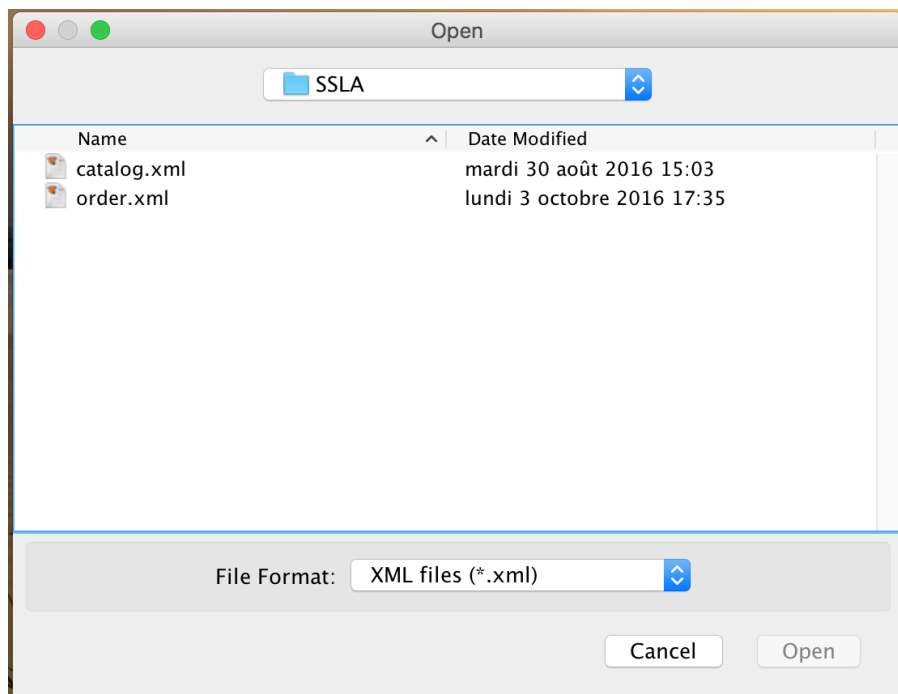


Figure 3.13: Configuration of the Negotiation Agent

Both Cloud Service Customers and Cloud Service Providers make use of the SSLA Specification Platform (cf. Section 3.1) to express their requirements and constraints for the Cloud Service.

### 3.2.1.2 Cloud Services Publishing and Discovery

Once created, both the Provider Agents (PA) and Customers Agents (CA) enter in Publishing and Discovery mode. Provider Agents promote their Cloud Service Offers to the Directory Facilitator. The DF is subsequently used by the CA to discover offers. For this purpose, we make use of the FIPA Agent Communication Language (FIPA-ACL) [6]. A FIPA ACL message contains a set of one or more message parameters (such as `performative`, `sender`, `receiver`, `content`, etc.). Precisely which parameters are needed for effective agent communication will vary according to the situation; the only parameter that is mandatory in all ACL messages is `performative`, although it is expected that most ACL messages will also contain `sender`, `receiver` and `content` parameters. The execution of offers publishing and discovery is illustrated in Figure 3.14. The Figure shows the message exchanges between CSP, CSC and the Directory Facilitator.

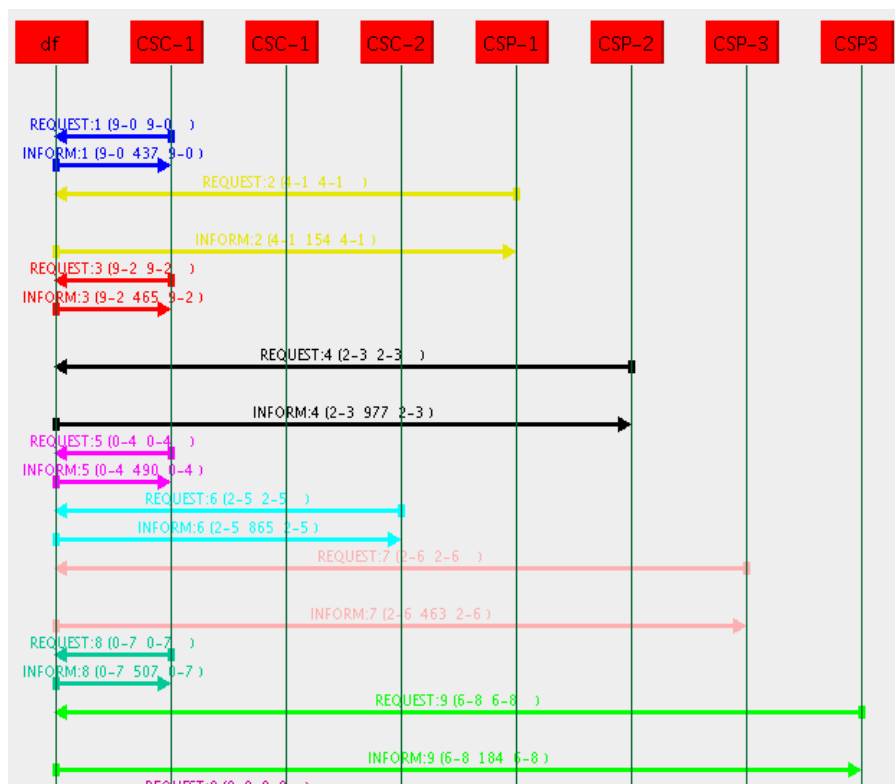


Figure 3.14: Discovery of Cloud Services

As illustrated above, each agent makes a request to the DF (left column) to inform it about the service it is offering. In Listing 3.1 below, we illustrate the code snippet used by the PA to add a new service with name `SUPERCLOUD-CSP` and type `CSP` (i.e., `CSP` in line 9) to the yellow pages service offered by the DF.

Listing 3.1: Java Code of Provider Agent Publishing phase

```

1
2 // Register as a Cloud Service Provider in the DF Yellow Pages service
3
4 DFAgentDescription df = new DFAgentDescription();
5
6 dfd.setName(getAID());
7
8 ServiceDescription cloudService = new ServiceDescription();
9 cloudService.setType("CSP");
10 cloudService.setName("SUPERCLOUD-CSP");

```

```

11 fd.addServices(sd);
12
13 try {
14   DFService.register(this, df);
15 }
16 catch (FIPAException e) {
17   e.printStackTrace();
18 }

```

Figure 3.15 presents the content of a message sent by a CSP to the DF (see the `receivers` section). This message corresponds to the service created in Listing 3.1 above.

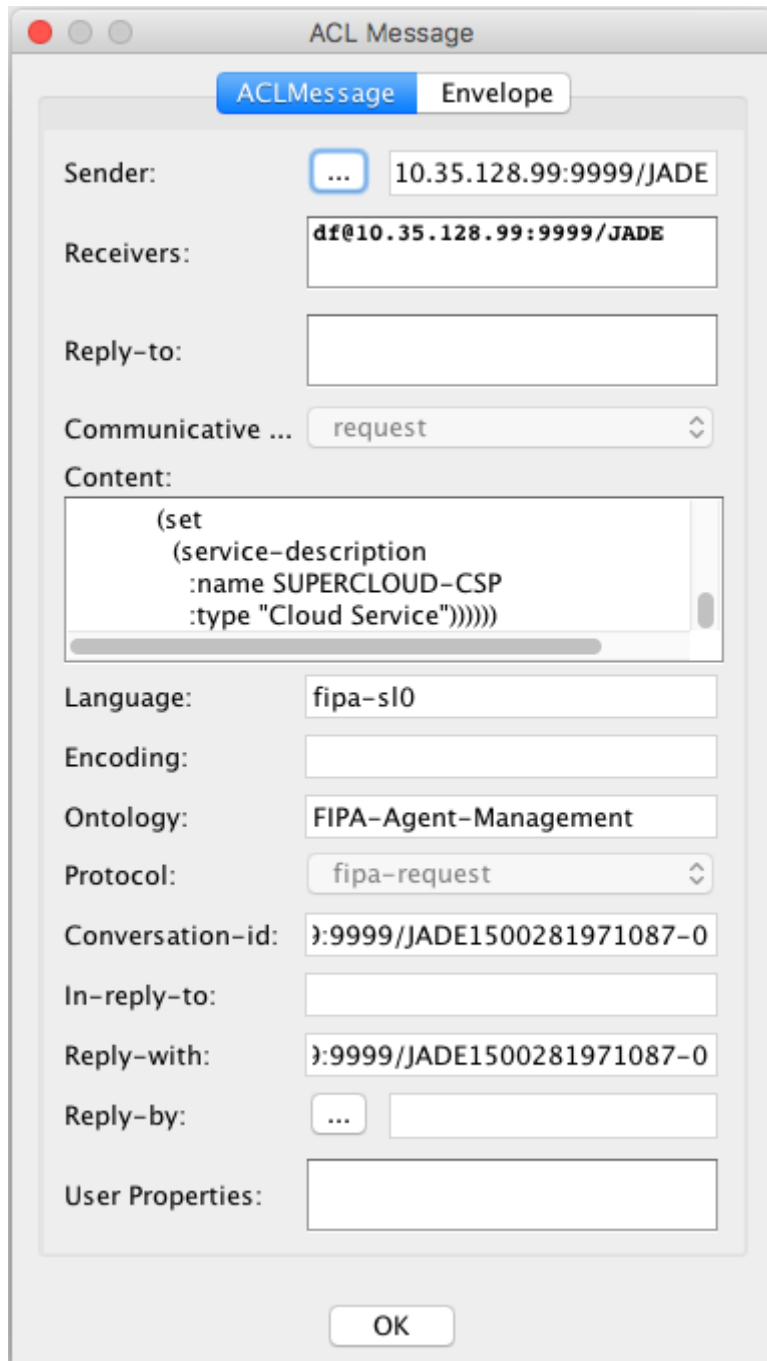


Figure 3.15: Publishing Message Content

### 3.2.1.3 Cloud Services Negotiation

When the CA is created, it receives at the same time the Cloud Customer requirements in terms of Cloud Service. This file is processed by this agent to extract Service Level and Protection Level Objectives. The CA then contacts the DF to have the list of PA offering cloud services. This list is first filtered, as only a short list of PA matching the CSC requirements is kept. The members of the filtered short list are used then as receivers to the Call For Proposal (CFP) sent by the the CA.

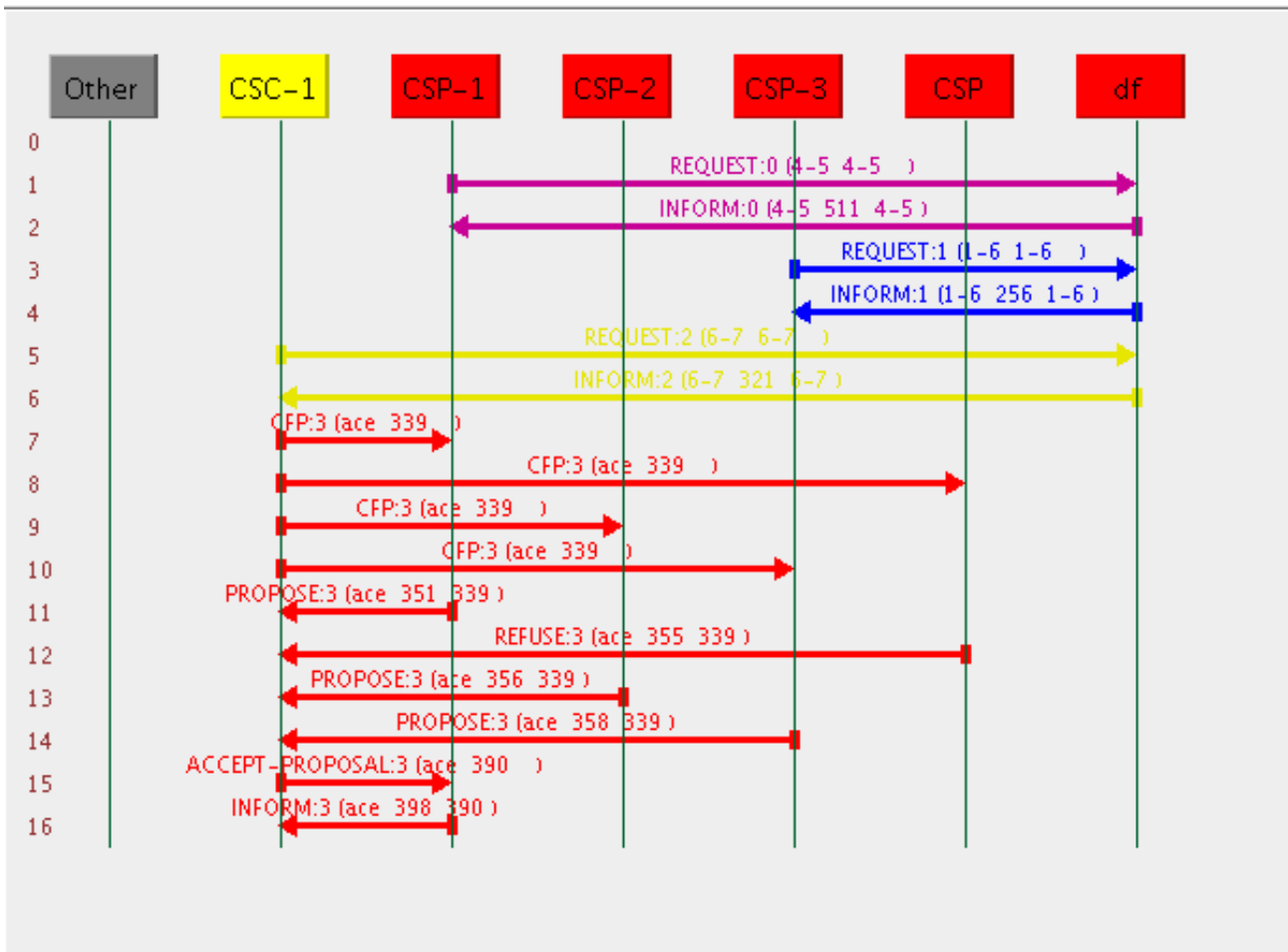


Figure 3.16: Call for Cloud Services Proposal

Figure 3.16 shows the Call for Proposal issued by CSC-1 to CSP, CSP-1, CSP-2 and CSP-3. Here we see that only three CSPs replied with offers (i.e., PROPOSE). Among the three offers, only one was accepted (i.e., offer of CSP-1). In this example, we assume that only one provider is needed to fulfill the request of the customer. However, in multi-cloud settings, multiple providers are selected by the customer to constitute the clouds federation.

Additionally, for simplicity, we showed only a one-shot selection process where the negotiation is limited to a single [proposal, accept-proposal] round. Nevertheless, the implemented negotiation framework handles more sophisticated and complex negotiation schema where CA and PA alternate offers and counter-offers until a consensus is reached. The overall negotiation process is summarized as follows:

1. The Provider Agent (PA) joins the Multi-Agent Cloud Market Place (CMP) and subscribes with the Directory Facilitator (DF) to publish cloud offers.
2. The Customer Agent joins the CMP looking for a cloud service. It retrieves the list of PA offering the service it is looking for.

3. A Call for Proposal is sent to the list of matching PAs.
4. Interested PAs replies to the CFP.
5. The CA replies to the proposals.
  - If a PA can fulfill completely the request, the CA selects the best (i.e., cheapest) offer. It negotiates the price if above the expected value.
  - If no PA can fulfill the request, the CA composes PAs offers and forms a multi-cloud based on its own needs. The CA negotiates the price to comply with the initial expected value of services.
6. The negotiation terminates if:
  - The deadline is reached.
  - No proposal have been made by PA and no multi-cloud can be formed.
  - An agreement is reached and a offer/counter-offer was mutually accepted by Customers and Providers.
7. An XML SSLA file is generated.

The SSLA file generated at the end of the negotiation represents the agreement that the CSP must enforce. Thus it is used primarily to guide the deployment of security services across *Compute*, *Data* and *Network* planes. This file will also serve as a basis for arbitration if the agreement was violated by one party. This file is converted into the standard WS-Agreement format [1]. The choice of making this conversion is motivated by interoperability needs. Indeed, we assume that Cloud Customers and Cloud Providers may need to have standard format to communicate about active agreements.

## Chapter 4 Implementation of Security Self-Management

This Chapter gives an overview of the implementation of the security services presented in Chapter 2. These services constitute the SUPERCLOUD Security Self-Management Framework. As discussed in Section 2.2, we adopted the micro-services approach to ease the automation of security services deployment and management across the different SUPERCLOUD frameworks (i.e., *compute*, *data* and *network*).

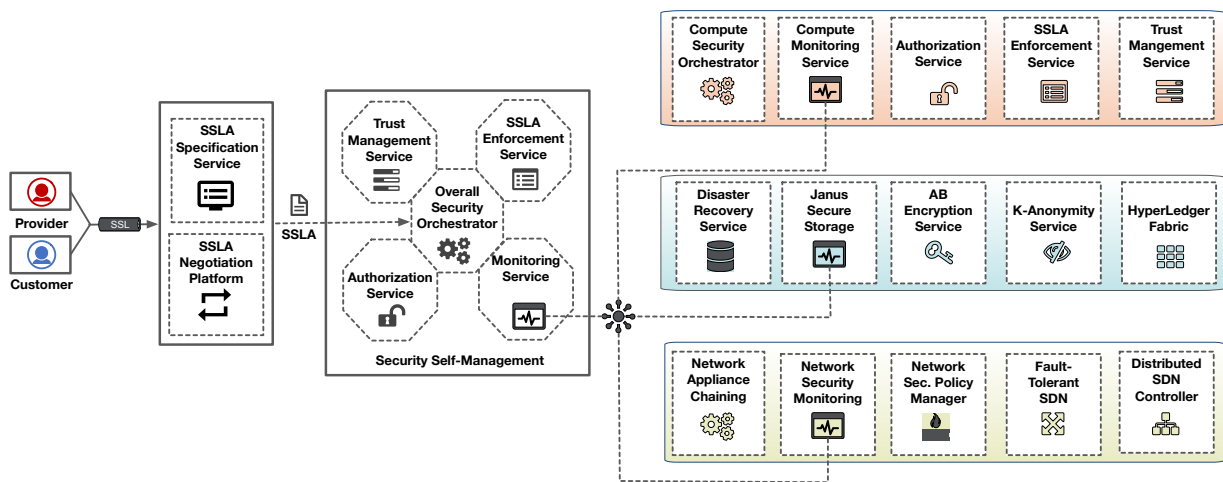


Figure 4.1: Overview of SUPERCLOUD Security Services

As illustrated in Figure 4.1, security services are split into five categories: (a) Specification and Negotiation of SSLA<sup>1</sup>, (b) Overall (and cross-planes) Security Services (cf., Sections 4.1 and 4.2), (c) Compute-Level Security Services (cf., Section 4.3), (d) Data-Level Security Services (cf., Section 4.4), and (e) Network-Level Security Services (cf., Section 4.5).

Some security services are spread across planes to allow distributed and/or decentralized operation. For instance, the Monitoring Service is provided at each level of the SUPERCLOUD framework to allow a 360° surveillance of cloud service and security service provisioning.

In what follows we first review in Sections 4.1 and 4.2 the Overall Security Orchestrator and the security services that operate across *compute*, *data* and *network* SUPERCLOUD planes. Then in Sections 4.3, 4.4 and 4.5 we provide a brief review of security services that have been implemented in, respectively, *compute*, *data* and *network* SUPERCLOUD sub-frameworks. The detailed description of these services can be found in Deliverables D2.3 [10], D3.3 [2] and D4.3 [19] respectively.

<sup>1</sup>These services have already been presented in Chapter 3.



## 4.1 Security Self-Management Framework

In this Section, we present the implementation of the Security Services that constitute the Security Self-Management Framework. The framework is mainly composed of five services, namely the *Overall Security Orchestrator* (OSO), the *SSLA Enforcement Service* (SES), the *Authorization Service* (AS), the *Trust Management Service* (TMS) and the *Monitoring Service* (MS).

### 4.1.1 Overall Security Orchestrator

In Section 2.2, we selected Docker<sup>2</sup> as a state-of-the-art technology to deploy the micro-services encapsulating the SUPERCLOUD Security Self-Management Services. Docker is known to be adequate for packaging simple and single-container applications and services. But when it comes to complex systems that consist of several dependent and independent services, a more sophisticated deployment schema is required. To that aim, we build on the top of Docker `Compose` tool [3] to develop the *Overall Security Orchestrator*. Docker `Compose` allows agile and dynamic deployment of security services. The same approach has been adopted to implement the Compute-Level Security Orchestrator<sup>3</sup>.

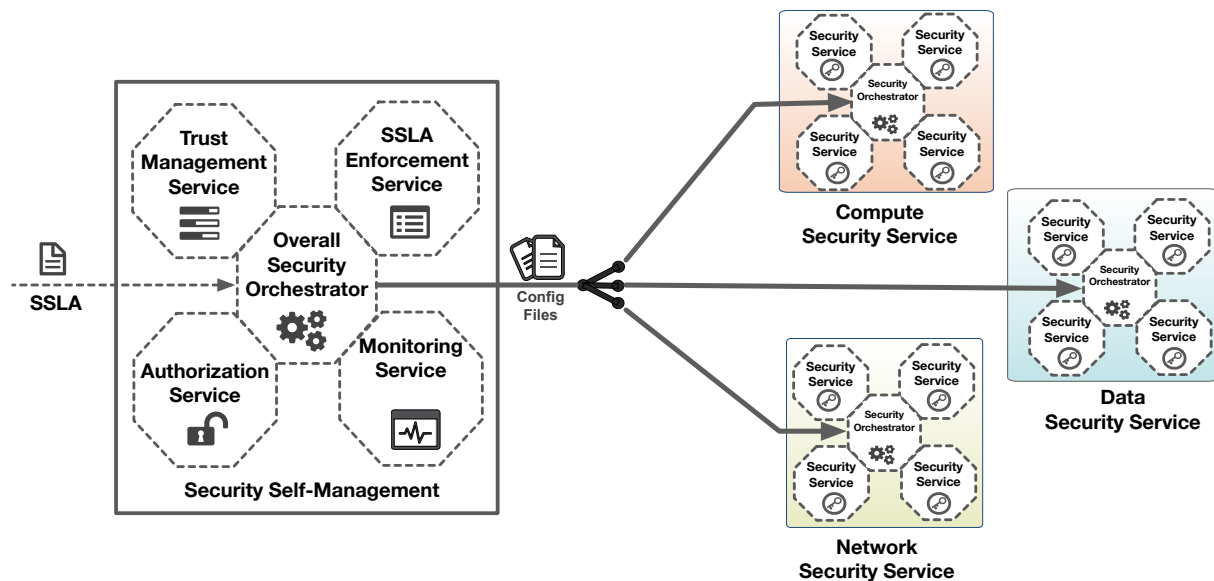


Figure 4.2: Overview of the Overall Orchestrator

Figure 4.2 represents the workflow from the moment the SSLA file is generated by the SSLA Specification and Negotiation Service, to the concrete deployment of security services. The first phase of the workflow occurs inside the Overall Security Orchestrator. During this phase, the Orchestrator processes the SSLA file and derives instructions about the security services to be deployed and the Service and Protection Level to be ensured. The Security Orchestrator, and SUPERCLOUD plane-specific security orchestrators take care as well of the configuration and deployment of security services identified in the first phase. In what follows, we provide a detailed view of the orchestration workflow illustrated in Figure 4.2.

The complete processing of the SSLA XML file by security orchestrators involves four steps, each operated by a dedicated engine that we illustrate in Figure 4.3.

<sup>2</sup><https://www.docker.com/what-docker>

<sup>3</sup>Please refer to Deliverable D2.3 [10].

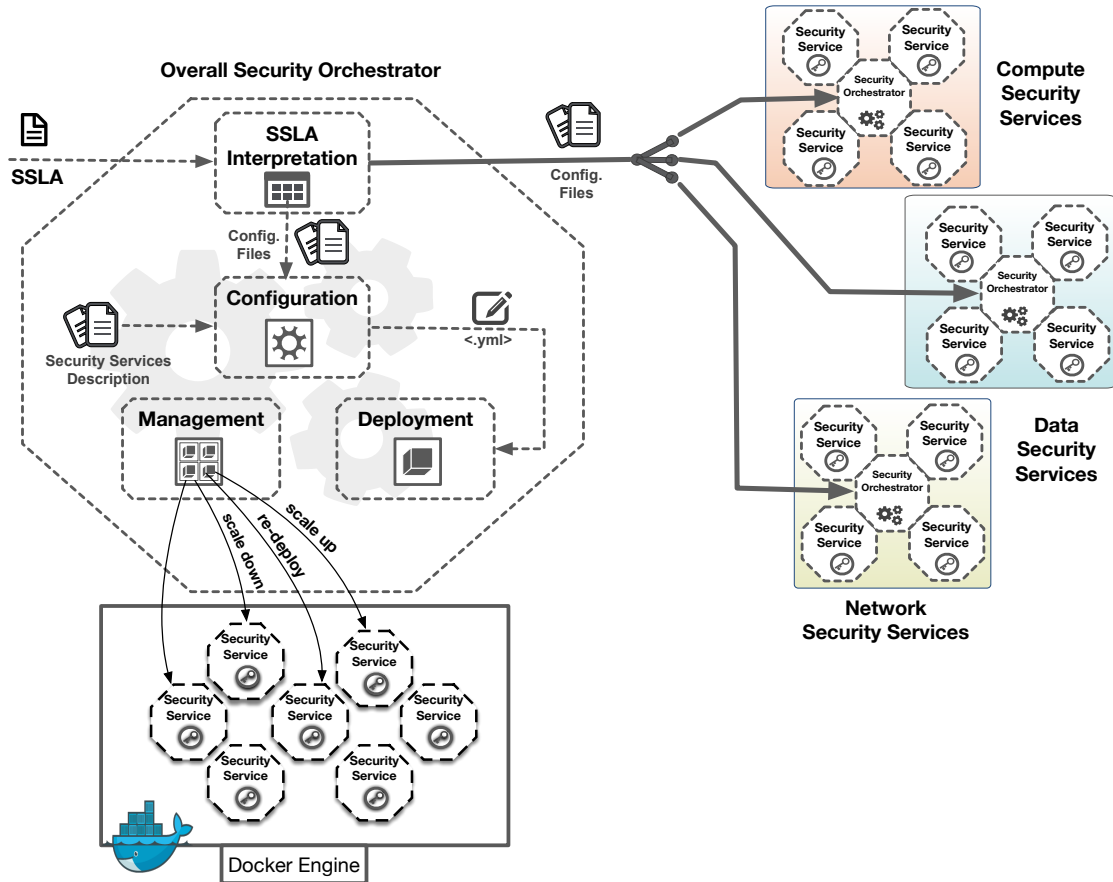


Figure 4.3: Security Orchestrator Workflow

### 1) Interpretation

The first step of the Orchestration of Security Self-Management Services is *SSLA Interpretation*. The SSLA file generated after negotiation is processed by the Overall Security Orchestrator to extract security services to be deployed. From this list of services, the Orchestrator identifies services that are part of Security Self-Management and those that are specific to one of the SUPERCLOUD planes (i.e., *compute*, *storage* and *network*). For both, the Orchestrator generates dedicated XML configuration files. Each XML file contains:

- *Security Level Objectives* such as QoS metrics to be reached.
- *Protection Level Objectives* that refer to security properties to be ensured.
- List of *Security Services* with corresponding configuration parameters.

Once generated, XML configuration files are transmitted to Security Orchestrators specific to *Compute*, *Data* and *Network* planes. The Overall Security Orchestrator configures and deploys the Security Services that encompasses the three planes (i.e., SSLA, Authorization, Monitoring and Trust) as illustrated in Figure 4.3. Finally, the Security Orchestrator of each plane configures and deploys the Security Services that are specified in the SSLA XML configuration file extracted from the SSLA. The process followed by each plane-specific Security Orchestrator to configure, deploy and manage security

services is similar to the one used by the Overall Security Orchestrator<sup>4</sup>.

## 2) Configuration

The configuration of Self-Management of Security Services is achieved through a single configuration file. This file is built by weaving Security Services Description (Left side of Figure 4.3) based on the instructions extracted by the SSLA Interpreter from SSLAs. This file specifies how services can interact with each other in a coherent and consistent way. It defines the order in which services are deployed and the conditions under which a service may be invoked or not. The file defines parameters that are necessary for the execution of each service such as network ports to be opened, volumes to be mounted or dependencies with respect to other services.

We provide in Listing 4.1 an example of what a deployment configuration file looks like.

Listing 4.1: Example of a deployment file generated by the Orchestrator

---

```

1  services:
2
3  authorization:
4  build: .
5  ports:
6  - "8080:8080"
7  volumes:
8  - /policies:/policies
9  db:
10 image: mysql
11 ports:
12 - "3306:3306"
13 environment:
14 MYSQL_ROOT_PASSWORD: 123456
15 MYSQL_USER: supercloud
16 MYSQL_PASSWORD: 123456
17 MYSQL_DATABASE: selfmanagement

```

---

In this example, the Security Orchestrator will deploy two services, an *Authorization Service* and a *Storage Service* containing persistent data such as monitoring information. The Authorization Service is built using a Docker file placed in the current directory.

The instructions specify that the Authorization Service needs to expose port 8080 and map it to port 8080 outside the container. The policies used for authorization are copied from the mounted volume `/policies`. For the Storage Service, a default MySQL image is used (cf. line 10) and port 3306 is mapped to 3306. Additional configuration information (e.g., root password, database name) is also specified (cf. lines 13-17).

## 2) Deployment

The current version of the Orchestrator relies on the Docker Compose tool for the deployment of security services (i.e., containers and links between them). Deployment is done in two steps.

First, the Docker images are built based on the instructions provided within the individual `Dockerfile` files. Listing 4.2 provides an example of the `Dockerfile` used for the Authorization Service.

Listing 4.2: Authorization Service Dockerfile

---

```

1  FROM ubuntu
2
3  MAINTAINER Reda Yaich <reda.yaich@imt-atlantique.fr>
4

```

---

<sup>4</sup>At the network plane, Security Services are deployed as applications on the north-bound interface of the Network Hypervisor by the Security Service Chaining Component. Please refer to Deliverable D4.3 [19] for more details.

```

5 # Update the base ubuntu image with dependencies needed
6 RUN apt-get update && \
7 apt-get install -y openjdk-8-jdk && \
8 apt-get install -y ant && \
9 apt-get clean;
10 RUN apt-get update && \
11 apt-get install ca-certificates-java && \
12 apt-get clean && \
13 update-ca-certificates -f;
14
15 # Setup JAVA_HOME, this is useful for docker commandline
16 ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64/
17 RUN export JAVA_HOME
18
19 # Expose the required ports
20 EXPOSE 8080:8080
21
22 #copy files to the created container volumes
23 COPY /Authorization_Service /Authorization_Service
24 COPY /policies /Authorization_Service/bin/policies
25
26 #set the working directory
27 WORKDIR /Authorization_Service
28
29 #Specifies the entrypoint to the docker
30 ENTRYPOINT ["/Authorization_Service/start.sh"]

```

The command `docker-compose build` will process the `docker-compose.yml` file present in the current folder and create the image of each Security Service.

Then, using the command `docker-compose run`, the Security Services are executed as follows:

- A self-management dedicated virtual network is created.
- Volumes are mounted for security services requiring such operation.
- Images of each service are pulled by Docker.
- Creation of services is ordered based on dependencies.

At this stage, the services are deployed as shown in Figure 4.4.

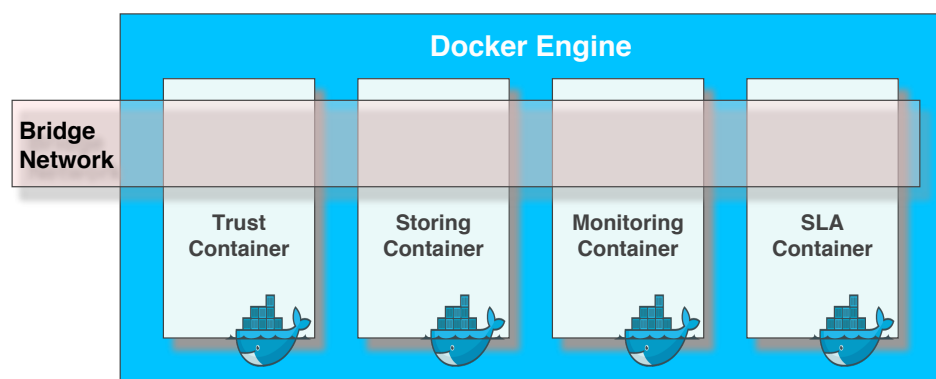


Figure 4.4: Illustration of security services deployment

We show in Figure 4.5 the corresponding process displayed by the Docker machine.

CREATED	STATUS	PORTS	NAMES
4 minutes ago	Up 4 minutes	80/tcp, 0.0.0.0:1982->8080/tcp	orchestrator_trust_1
4 minutes ago	Up 4 minutes	80/tcp, 0.0.0.0:8080->8080/tcp	orchestrator_authorization_1
4 minutes ago	Up 4 minutes	80/tcp, 0.0.0.0:1981->8080/tcp	orchestrator_ssla_1
4 minutes ago	Up 4 minutes	0.0.0.0:3306->3306/tcp	orchestrator_storage_1

Figure 4.5: Deployed security services

### 3) Management

In the deployment phase, Security Services are deployed on the host successively to avoid dependency conflicts. After that, the Orchestrator enters the *management phase* to prevent the Self-Management of Security Framework from running in an unhealthy state (e.g., due to faulty services). Two main states are handled by the Security Orchestrator:

1. **Overloaded services.** This situation can occur when the user multi-cloud scales-up in response to client demand. Consequently, some Security Services (e.g., Monitoring, Authorization) need to adapt to such change. To address this issue, the Orchestrator makes use of the `docker-compose scale SERVICE=X` command, provided by Docker-Compose and Docker Swarm, to launch X instances of the considered service.
2. **Faulty services.** For some reasons, security services may stop. As a self-managed security service, the Orchestrator needs to re-run without human intervention security services. This is achieved using command `docker-compose up --no-recreate` that allows the Orchestrator to re-launch the same service without re-building its image, hence reducing recovery time.

The automation of restarting faulty security services is important due to the critical nature of their objectives. However, in some settings, continually restarting faulty containers embodying these services may block the overall self-management process. This is mainly due to the restart process loop that will fill up the physical host disk space. This is relatively common when handling stateful services such as MySQL. This is particularly true for the Storage Service in our architecture. To address this issue, we make use of a more sophisticated management scheme that relies on the **Swarm** mode [5] of the Docker engine. Within Swarm, the Docker engine makes use of an explicit restart policy that needs to be specified within the `docker-compose.yml` configuration file.

Listing 4.3 shows an example of Docker-Swarm specific restart instructions. The `max_attempts` parameter makes the deployment process safer by fixing a limit to container restarts. The `replicas` statement enables to specify the initial number of instances for each Security Service.

Listing 4.3: Restart policy configuration in Swarm

```

1 ...
2 deploy:
3 replicas: 3
4 restart_policy:
5 condition: on-failure
6 delay: 30s
7 max_attempts: 3
8 window: 60s
9 ...

```

## 4.2 Security Self-Management Services

As illustrated in Figure 4.2, Security Self-Management is composed of several Security Services. These services are deployed based on the requirements of Cloud Service Customer to allow him/her build a U-Cloud that satisfies his/her requirements. As presented in Section 2.1, Security Services are split into two categories, Plane-Specific Services and Cross-Plane Services.

In this Section, we present the Cross-Plane Security Services. In the next Sections (i.e., Sections 4.3, 4.4 and 4.5) we provide an overview of Security Services that have been developed within *Compute*, *Data* and *Network* planes.

### 4.2.1 Authorization Service

This Section introduces the SUPERCLOUD Authorization Service. The service was presented in detail in Deliverables D1.2 [23] and D2.2 [14] and D2.3 [10]. The Authorization Service extends the OrBAC API to manage authorizations within Multi-Cloud Infrastructures. The service objective is to allow the interpretation of OrBAC policies to derive *access control* and *usage control* decisions (i.e., permission, prohibitions, obligations).

Within the SUPERCLOUD framework, other application and services can delegate authorization decisions to this service. For instance, the geo-replication service from the Compute Plane relies on this service to distinguish locations wherein replication among VMs is allowed from locations wherein it is not (cf., Section 5.1). For the same purpose, the Secure Data Storage Service Janus from the Data Plane interacts with this service to get the authorized locations for data replication (cf., Section 5.2). At the Network Plane, the Network Policy Manager delegates to the Authorization Service the management of context changes and the selection of appropriate routing paths to ensure network service availability (cf., Section 5.3). Finally, the Philips Imaging Platform relies on the Authorization Service for Access Control Decisions (cf., Section 5.4).

As illustrated in Figure 4.6, the SUPERCLOUD Authorization Service works in a classical client-server mode wherein the application (here the Philips Imaging Platform) plays the role of the client and the SUPERCLOUD authorization service the role of server.

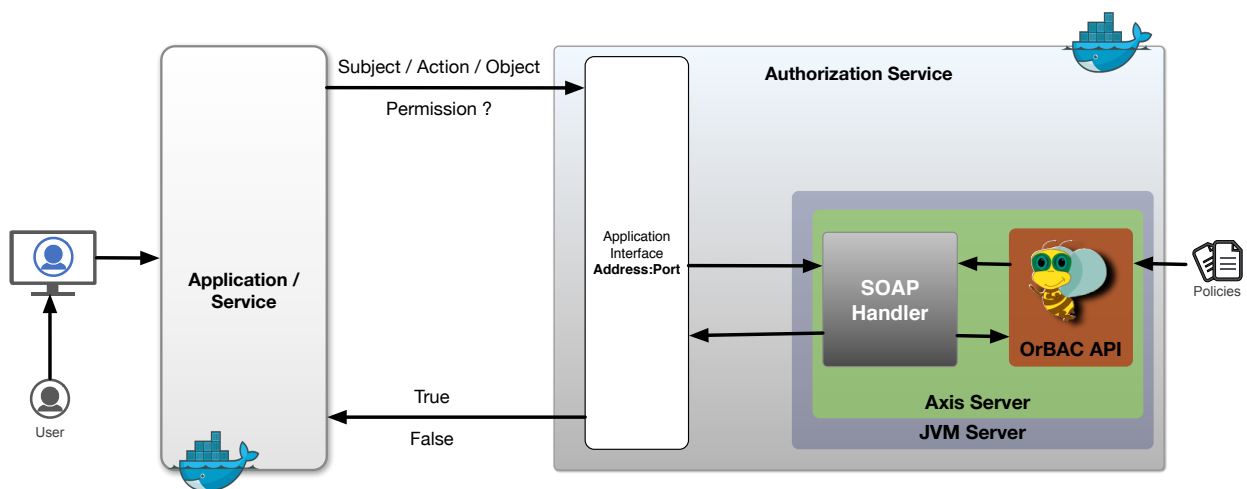


Figure 4.6: Integration of the Authorization Service with an Application or a Service

Applications that want to interact with the Authorization Service perform standard HTTP requests. Authentication is achieved by a dedicated service that needs to be specified. Hence, we assume that the SUPERCLOUD authorization service considers all identities as valid.

The SUPERCLOUD authorization service REST API<sup>5</sup> can be accessed via a standard and public REST API that only responds to GET requests and runs at:

`http://IP:PORT/services/SupercloudAuthorizationService/` endpoint.

Decisions can be of three types; *permissions*, *prohibitions* and *obligations*. A decision is made with respect to a triple (subject, object, action). For example, to verify if a subject *s* is authorized to perform an action *a* on an object *o*, the syntax of the request should be as follows:

`/IsPermitted?subject=s&action=a&object=o`

All requests represent a conjunction of conditions in which the above-mentioned triples can be presented in any order. The SUPERCLOUD authorization service standard reply to such decision requests is a Boolean value.

The SUPERCLOUD authorization service REST API is shown in Table 4.1.

Request	Description	Parameters	Response
GET URL/IsPermitted?	Checks if an action is permitted on a subject	subject=s&action=a&object=o	Boolean
GET URL/IsProhibited?	Checks if a scope is prohibited	subject=s&action=a&object=o	Boolean
GET URL/IsObligated?	Checks if a scope is obliged for a user	subject=s&action=a&object=o	Boolean
GET URL/Actions?	Retrieves all active actions in the policy	None	List of actions
GET URL/Objects?	Retrieves all active objects	None	List of objects
GET URL/Subjects?	Retrieves all active subjects	None	List of subjects

Table 4.1: Authorization Service API

## 4.2.2 Monitoring Service

The main aim of the Monitoring Service is to detect threats on the SUPERCLOUD infrastructure and on U-Clouds. This service is also a building block to react to threats, either directly, or by passing threat information to other components of the SUPERCLOUD Security Self-Management Infrastructure. We focus here on threats relevant to the computing infrastructure. Another monitoring service dedicated specifically to networking threats is implemented as part of the SUPERCLOUD network security framework, as described in Deliverable D4.3 [19].

Two dimensions of monitoring are explored:

- *Vertical monitoring* aims to provide a cross-layer view of threats and of their mitigation, based on probes (and counter-measures) in the different virtualization layers or at U-Cloud level. Information from provider-controlled infrastructure monitoring systems is also taken into account.
- *Horizontal monitoring* aims to provide a multi-provider view of threats and of their mitigation. A deployment framework such as MANTUS may help towards reaching a single point of control of security [18]. This aspect of monitoring also includes composition with other security services of the SUPERCLOUD Security Self-Management Infrastructure to propose a rich security response.

We present here a preliminary version of the Monitoring Service focusing on cross-layer self-protection. A more extensive version will be described in Deliverable D2.4.

<sup>5</sup>See D2.3 [10] for more details.

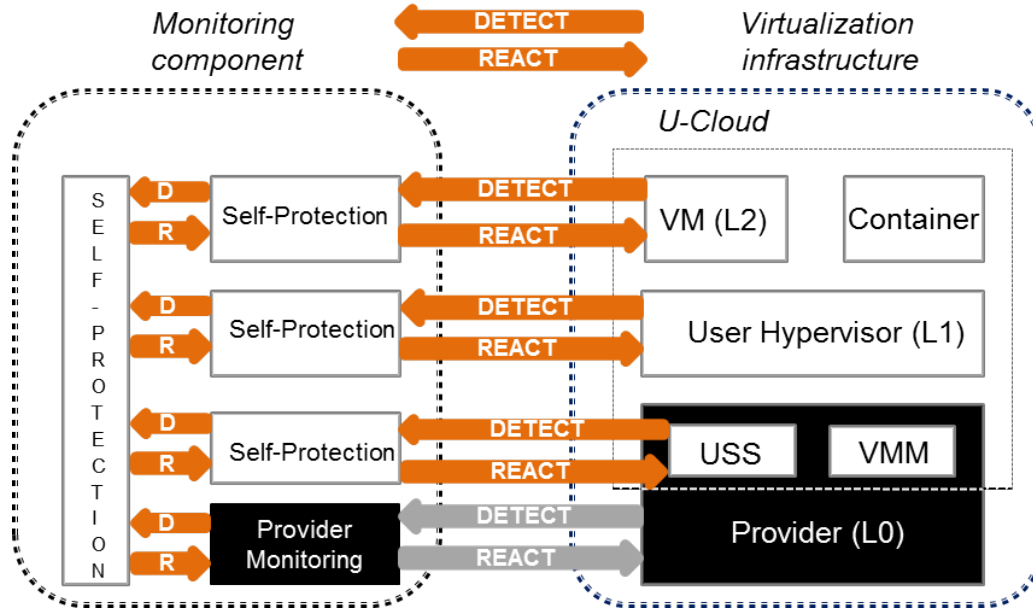


Figure 4.7: Security monitoring: approach

The security monitoring approach was presented in Deliverable D2.1 [12] and is summarized in Figure 4.7. Monitoring relies on orchestration of two hierarchical autonomic security loops.

- The first level manages *intra-layer security monitoring* in user- or provider-controlled parts of the virtualized infrastructure.
- The second level manages *cross-layer security monitoring*, also integrating monitoring information and counter-measures from the cloud provider.

The general design of the service is based on the VESPA framework [21] for the system model and on the OpenStack Watcher [17] framework for external APIs. We chose VESPA as it already implements a first two-level autonomic security monitoring model, but with a very basic API. We selected Watcher, as it is already integrated with OpenStack and provides much richer monitoring API.

### 4.2.3 Software Trust Management Service

In this Section, we present the *Software Trust Service (STS)* that processes Cloud Customers Experiences to compute trust and reputation values. These values are derived from the aggregation of monitored information. The objective of the Software Trust Service is to assist cloud customers and providers selecting the best interacting partners within the Cloud Market Place.

The trust that a cloud customer  $c$  is willing to put into a cloud provider  $p$  is derived from past experiences. The experiences constitute customers' and providers' feedback and reflect their level of satisfaction with respect to the expected quality of service and protection. This experience information is thus processed by the Trust Service for assessment of the trust that a customer can put in the candidate provider. Before making a decision<sup>6</sup>, the requesting customer  $c$  will make use of the SUPER-CLOUD Trust Management Service to derive a trust value based on past experiences. Then, during the transaction (i.e., Cloud Service Delivery), the CSC  $c$  will make use of monitoring mechanisms to observe the behavior of the provider. We make the reasonable assumption that all Service Level Objectives (SLOs) conveyed in an SLA agreement can be monitored and that monitoring information is reliable and cannot be tampered with.

Figure 4.8 illustrates the integration of the software trust model within the SUPERCLOUD computing framework, and more specifically within Self-Management of Security. The Software Trust Service

<sup>6</sup>This decision is about the provider to engage with, for a specific service  $s$ .



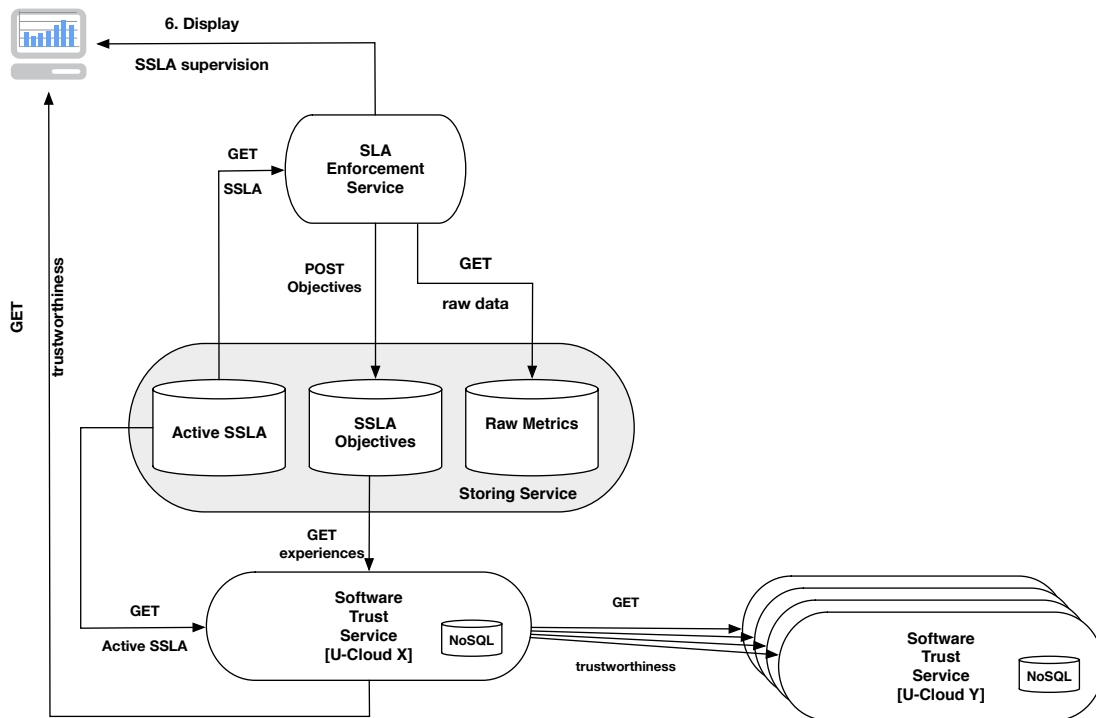


Figure 4.8: Overview of Software Trust Service

builds on the top of the SSLA enforcement and monitoring service to make trust assessments. The trust computation component mainly integrates the algorithm that processes the SSLA objectives and derives trustworthiness values as described in [22]. These values are made available via a standard REST API interface. The Software Trust Service REST API interface is split into two parts, internal and external. The *internal interface* interacts with other SUPERCLOUD services. These interactions are orchestrated by the Security Orchestrator. The *external interface* is used to communicate with other Software Trust Services to exchange experiences and trust values. This interface is mandatory when computing Cloud Marketplace-level Trust Metrics (i.e., reputation).

#### 4.2.4 SSLA Enforcement Service

The management of SSLAs involve five main phases: Specification, Negotiation, Enforcement, Monitoring and Arbitration (cf., Deliverable D1.2 [23]). Specification and Negotiation have been already described in Sections 3.1 and 3.2 of Chapter 3. In this Section, we make a focus on Enforcement and Arbitration while Monitoring will be presented in Section 4.3.

The objective of SSLA enforcement service is to provide the Cloud Service Customer (CSC) with mechanisms to supervise the execution of the active SSLA. Thus the service will process the SSLA upheld by the CSC and the CSP and extract the metrics to be monitored (Performance and Security Level Objectives).

As illustrated in Figure 4.9, the SSLA Enforcement Service (SES) translates and maps low-level raw resource metrics measured by monitoring services to high-level SSLA objectives. For instance, `upTime` and `downTime` are mapped to availability for both Compute, Storage and Network as follows:

$$Availability = 1 - \frac{upTime}{downTime} \tag{4.1}$$

Here, `downTime` refers to the time required to bring a service (Compute, Storage or Network) to

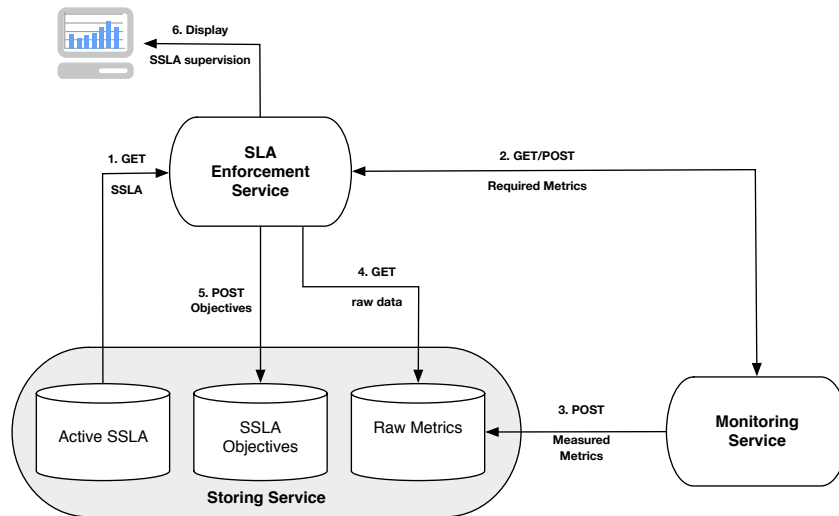


Figure 4.9: SSLA Enforcement Service

work after a failure, while `upTime` represents the sum of time without failure. Following this schema, the SSLA Enforcement Service extracts, periodically, relevant information to compute statistics that reflect the fulfillment of each Protection/Security Level Objective. This information is then displayed graphically using charts as shown in Figure 4.10.

### 4.3 Compute-Level Security Services

The Compute Sub-Framework provides a number of Security Services that are orchestrated based on Cloud Customers' requirement to ensure self-protection of U-Clouds on the top of the distributed virtualization infrastructure. The list of services include :

- **Authorization Service:** This service offers Access Control and Usage Control mechanisms to services and applications. It is built on the top of the OrBAC API. The Authorization Service derives decisions (i.e., permission, prohibitions, obligations) from OrBAC policies [9]. More details on this service can be found in Section 4.2.1 and Deliverable D2.3 [10].
- **Security Monitoring:** This service implements self-protection of U-Cloud resources, to detect and react to threats to the computing infrastructure in an autonomous manner. Two aspects of self-protection should be considered: cross-layer defense (vertical orchestration) and cross-provider defense (horizontal orchestration). Deliverable D2.3 [10] introduces a preliminary version of the monitoring component focusing on cross-layer self-protection. A more extensive version will be described in deliverable D2.4.
- **Geolocation-aware data replication:** This service aims to replicate data only in allowed locations. This objective is further complicated by relying on virtual machines (VMs) provided locally or by multiple cloud providers. Deployment of new services and requirements like availability or backup procedures cause the configuration of these VMs to be dynamic over time. The integration of the Georeplication Service with the Security Self-Management Framework is described in Section 5.1. Details on the implementation of this service and its integration with Security Self-Management can be found in Deliverable D2.3 [10].

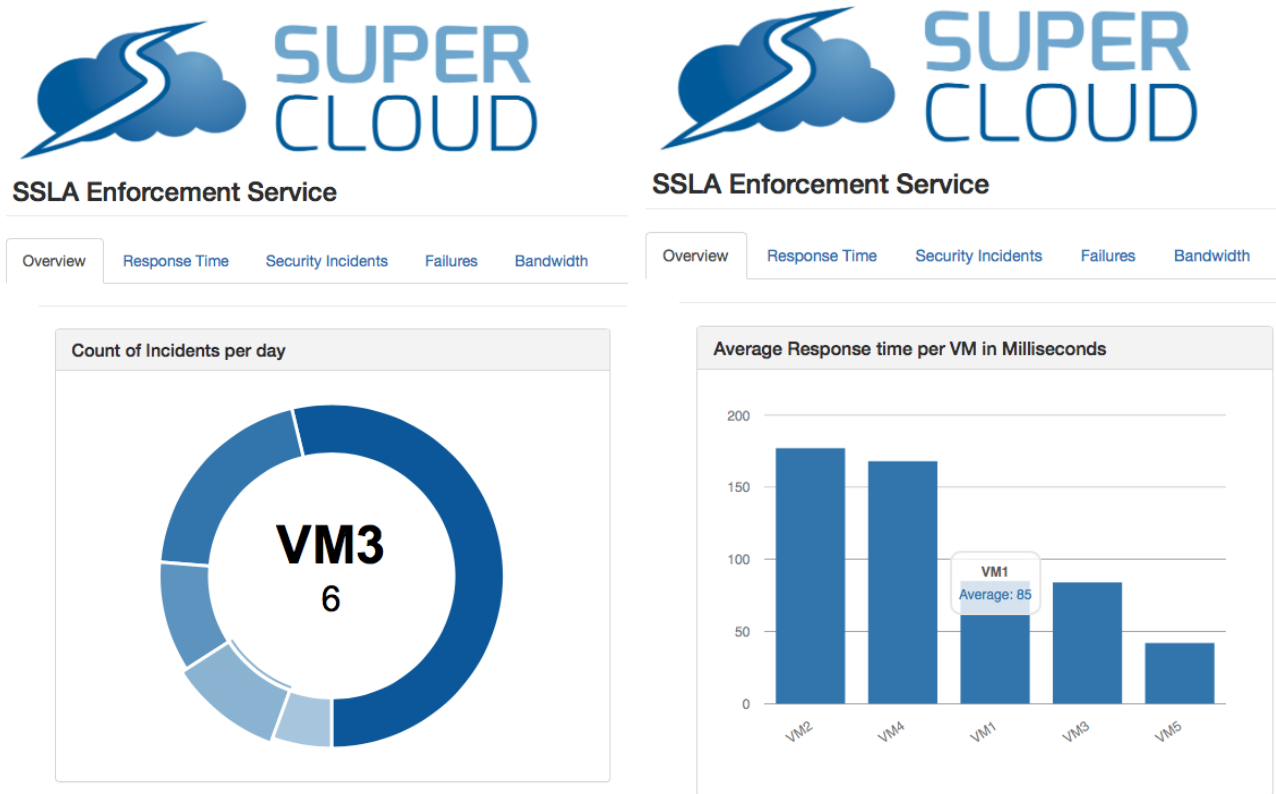


Figure 4.10: Captures from the SSLA Reporting Dashboard

- **SSLA Service:** The objective of this service is to provide the Cloud Service Customer (CSC) mechanisms to supervise the execution of the active SSLA. The SSLA Enforcement Service (SES) translates and maps low-level raw resource metrics measured by monitoring services to high-level SSLA objectives. For more details about this service, please refer to Section 4.2.4 and Deliverable D2.3 [10] as well.
- **Software Trust Service:** The objective of the Software Trust Service is to assist cloud customers and providers selecting the best interacting partners within the Cloud Market Place. The Software Trust Service builds on the top of the SSLA Enforcement and Monitoring Service to make trust assessments. The trust computation component mainly integrates the algorithm that processes the SSLA objectives and derives trustworthiness values as described in [22]. These values are made available via a standard REST API interface. Please refer to Section 4.2.3 for more details.

#### 4.4 Data-Level Security Services

The SUPERCLOUD data protection sub-layer provides several components to implement secure and dependable data management services in a multi-cloud environment. As illustrated in Figure 4.1, the implemented architecture considers a user-centric deployment of independent data security services. The security services implemented in this sub-framework have been presented in details in Deliverable D3.3 [2]. In this Section, we provide a brief overview of each service.

The main protection and management services include:

- **Secure Data Replication Service:** JANUS is a cloud-of-clouds storage system that maintains data in a dependable and secure way using multiple cloud providers as storage backends. The

system employs several Byzantine-resilient replication and coding algorithms [16] to spread the stored data in multiple cloud storage services (Amazon S3, Google Storage, Rackspace Files, etc.) in such a way that fault tolerance and confidentiality is preserved even if a fraction of these providers is compromised. The description of the ongoing integration of Janus with Security Self-Management is described in Section 5.2 and a detailed presentation of Janus may be found in Deliverable D3.3 [2].

- **Attribute-Based Encryption Service:** This service implements an Attribute Based Encryption (ABE) scheme, in which the encryption and decryption are based on users attributes. More precisely, in such a system, each end-user possesses some characteristic attributes. When uploading a new data, the depository chooses an access control policy, based on the existing attributes, and related to the data. Then, only users having the set of attributes verifying the defined access control policy will be able to decrypt and read the stored data. More details about ABE Services can be found in Chapter 6 of Deliverable D3.3 [2].
- **Data Anonymization Service:** This service aims at allowing personal sensitive data release, while preserving the individuals privacy. The service calculates the best solution for the given data in terms of cost-efficiency. This is done by means of so-called cost metric calculation as well as the Optimal Lattice Anonymization (OLA) algorithm (cf., Deliverable D3.2 [15]). A detailed explanation of the OLA algorithm as well as of all including components of the tool can be found in the Deliverable D3.3 [2].
- **Hyperledger Fabric:** This component provides State-Machine Replication-related mechanisms as described in SUPERCLOUD Deliverable D3.2 [15]. Notably it includes: (a) a component that treats non-determinism when replicating arbitrary applications when replicas can fail in an arbitrary (i.e., Byzantine) way, (b) a component that introduces a novel model for developing reliable distributed protocols called XFT, (c) a component that empirically evaluates latency-optimization for state-machine replication in WANs and informing the design of novel state-machine replication protocols, and (d) a component that introduces a generic state-transfer tool for partitioned state-machine replication that enables elasticity.

## 4.5 Network-Level Security Services

As illustrated in Figure 4.1, the Network Security Self-management sub-framework is composed of independent security services that include: the *Network Security Monitoring Service*, the *Network Security Policy Management Service*, and the *Network Security Appliance Chaining Service*. More details about those services can be found in Deliverable D4.3 [19].

- **Network Security Monitoring Service:** This service allows the detection of security incidents in a tenant network hosted over a multi-cloud. For this purpose, the service collects statistics regarding the network state by issuing requests to switches periodically. It also provides a tunable notification service to a Context Handler for specific types of alerts (e.g., link congestion, high packet drop rate). The services processes collected information to have a complete view of the state of the network, enabling automatic response to security incidents.
- **Network Security Policy Management Service:** This service aims at managing and deploying network security policies automatically. It interacts with the Security Monitoring service which provides alerts and statistics about the SDN networks exposed by the Network Hypervisor. It is deployed as an application on top of the Network Hypervisor. It reacts to the notifications received and instructs the Network Hypervisor to deploy the changes in order to dynamically adapt the network to the context of the environment. The reaction is chosen according to security policies.

- **The Network Security Appliance Chaining Service:** This service allows end-user to easily compose his own security service chains (online or off-line) in a multi-cloud environment. The service runs on top of the Floodlight [7] controller and uses its REST API to discover the topology, get traffic statistics or install rules in the switches. REST commands used are only valid for switches compatible with OpenFlow [13] version 1.3, which must be explicitly activated in Open vSwitch.

In the next Chapter, we will provide some examples of the integration of Security Self-Management Framework.

## Chapter 5 Integration of Security Services

In this Chapter, we present an overview of the integration of Security Services. This integration has been achieved following the methodology illustrated in Figure 5.1.

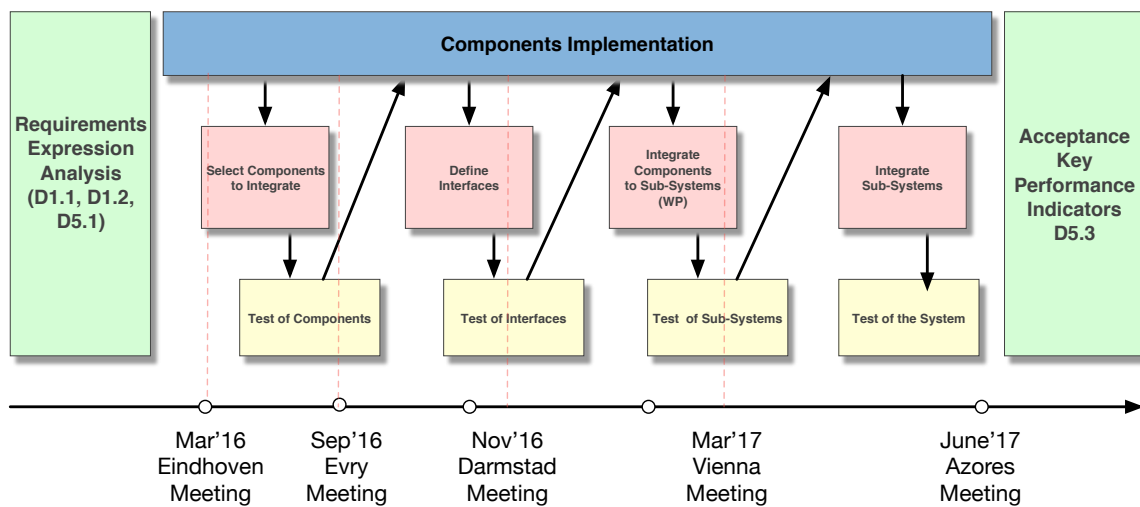


Figure 5.1: Security Services Integration Approach

The overall integration workflow has been discussed and planned throughout several SUPERCLOUD technical meetings as annotated in Figure 5.1. The process may be summarized as follows:

1. First, Security Services have been developed and tested individually.
2. Second, for each service, external interfaces that allow it to communicate with other services have been defined and developed.
3. Then, individual services were integrated into sub-systems based on common objectives and dependencies. For instance, Authorization depends on Authentication.
4. Finally, sub-systems have been integrated to use-case demonstrators in order to evaluate and validate the overall SUPERCLOUD Self-Management of Security Framework.

In the remainder of this Chapter, we will highlight examples of Security Service integration at different levels. First, we show examples of integration between services of Compute (cf., Section 5.1), Data (cf., Section 5.2) and Network (cf., Section 5.3) frameworks. Finally, in Section 5.4, we demonstrate the integration of some Security Services with applications related to project use-cases.

## 5.1 Integration with Compute Security Services

In this Section, we showcase the integration of SUPERCLOUD Security Self-Management Services with Compute-Level Security Services. In this example, we describe the integration of the Geolocation-aware Replication Service developed by Philips Research with two of the Security Self-Management Services, namely *Authorization* and *SSLA Management* services.

The main goal of the Location-aware Data Replication Service is to allow cloud users to keep control on the locations where their data could be replicated, addressing the numerous geolocation directives and regulations <sup>1</sup>. We showcase in this Section how such objective has been achieved in SUPERCLOUD.

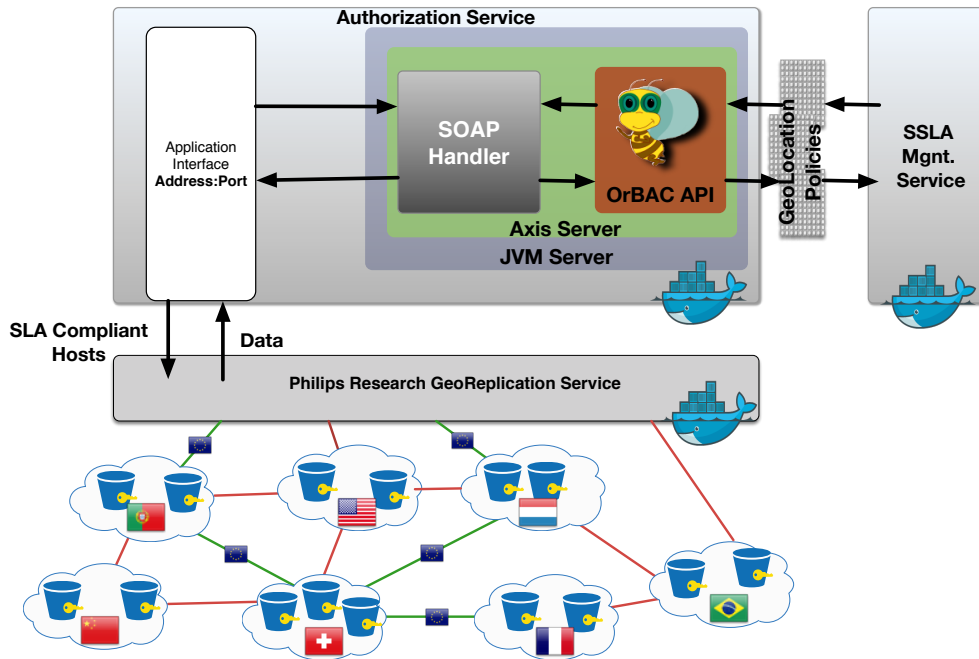


Figure 5.2: Integration of SSLA Management and Georeplication Services

As illustrated in Figure 5.2, geolocation policies are extracted from the SSLA by the SSLA Management Service and converted to OrBAC rules. These rules are subsequently processed by the Authorization Service to derive permissions and/or prohibitions. From the Georeplication Service perspective, integration with the aforementioned Authorization Service is achieved throughout a standard REST API interface. This interface allows the Georeplication Service to retrieve the list of locations/VMs that are compliant with the cloud customer requirements. This list is subsequently used by this service to achieve SSLA-compliant data replication.

## 5.2 Integration with Data Security Services

Data Security Services and Security Self-Management Services are integrated at several levels. In this Section we provide two examples of these integration actions.

### 5.2.1 Location-Awareness Policies for SLAs

This Section provides an overview on the implementation of location-aware data replication services. This feature is showcased in SUPERCLOUD through the integration of SSLA services that extract user requirements in terms of data location and the secure storage provided by Janus.

<sup>1</sup>For instance, the GDPR (General Data Protection Regulation).

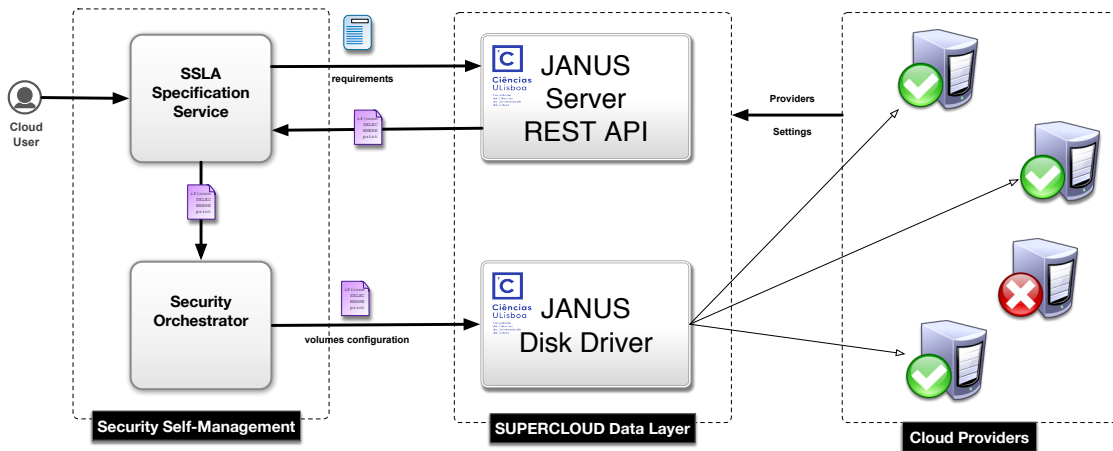


Figure 5.3: SSLA Location-Aware Janus

As illustrated in Figure 5.3, the location-aware fault-tolerance service is the result of the integration of two components of Self-Management of Security, and two components from the Janus framework. The user first specifies his data volumes requirements through the SSLA Specification Service<sup>2</sup>. Then, based on these requirements, a request is sent to the Janus service where a solver will find the best solution that matches the user’s desiderata. Once this configuration is found, the result is sent back to the SSLA Service which takes care of forwarding it to the Security Orchestrator. In the last phase, the Orchestrator will use this configuration file to deploy and configure the Janus Virtual Disk Driver. Once running, this component will do regular backups of user data only on providers that fulfill the user’s requirements in terms of location, but ALSO in terms of cost and latency.

### 5.2.2 Monitoring of Data Access Failures

In this Section, we provide an overview of the ongoing integration action between the Security Self-Management and Janus, the dependable and secure multi-cloud data storage service<sup>3</sup>.

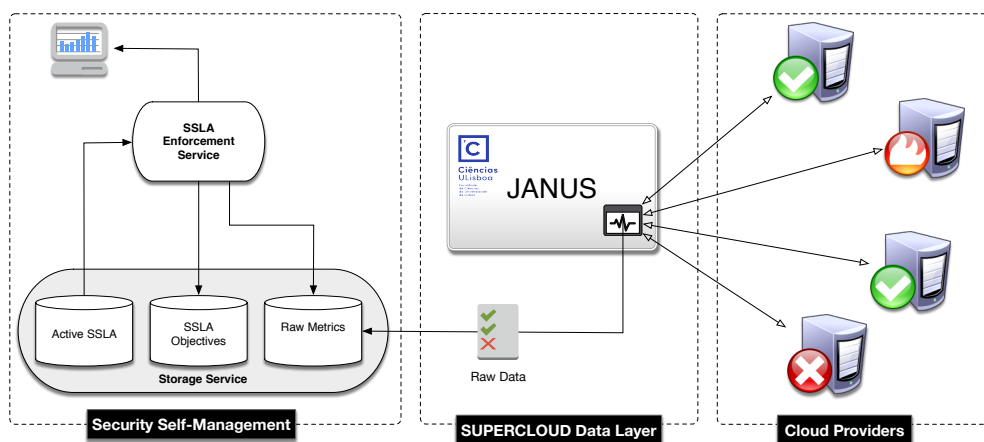


Figure 5.4: Integration of Self-Management of Security and Janus

As illustrated in Figure 5.4, the Janus secure storage service embodies monitoring components that collect information about the health of the servers wherein SUPERCLOUD user data is hosted. The

<sup>2</sup>This service is described in Chapter 3.

<sup>3</sup>We invite the reader to refer to Deliverable D3.3 [2] for more details about Janus.



collected data concern essentially data *Availability* and the observed *Latency*. The collected data are then sent to the Security Storage Service (part of the Security Self-Management Framework) to be processed by the SSLA Enforcement Service (Left part of Figure 5.4). This raw data is then processed to be converted into high level metrics to be compared with SSLA objectives defined by the SUPERCLOUD Cloud Customer. The result of this processing is then displayed to allow supervision and arbitration<sup>4</sup>.

### 5.3 Integration with Network Security Services

In this Section, we present an overview of the integration of Security Services from the Self-Management and Computing Framework with Security Services from the Networking framework. This integration showcases the implementation of user-driven and adaptive network policies for service availability.

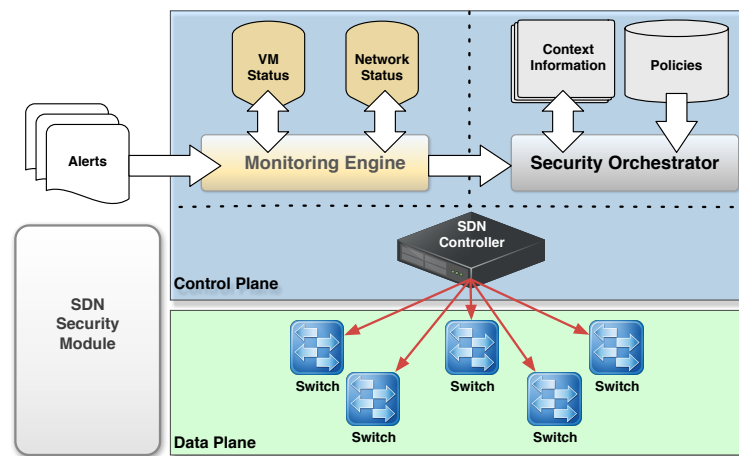


Figure 5.5: OrBAC-based orchestration of network security policies

As illustrated in Figure 5.5, user security preferences captured and negotiated by the Security Service Level Service are represented as OrBAC authorization rules (i.e., permissions and prohibitions), as shown in the top-left part of Figure 5.5. These policies are then used by the Security Orchestrator to activate and/or deactivate SDN routing paths based on context and monitoring information.

### 5.4 Integration with Use-Cases Application

In this Section, we demonstrate the integration of the Philips Imaging Platform with the Security Self-Management Framework. This integration is achieved at two levels. First, the Philips Imaging Platform is integrated with the SSLA Service wherein user requirements are captured and used to derive requirements for Compute, Data and Network but also Customer's desiderata about security services and their configuration. For instance, the Philips Imaging Platform is integrated with Janus, the Secure Data Storage Service developed in the Data Plane. And as illustrated in Section 5.2.1, Janus deployment and configuration is orchestrated by the Security Self-Management Framework. Thus, the Philips Imaging Platform is also implicitly integrated through Janus. In addition, the Philips Imaging Platform delegates Access Control Decisions to the Authorization Framework (cf., Deliverable 5.2 [20] for more details). This integration is illustrated in Figure 4.6. Finally, the Philips Imaging Platform also automatically benefits from the SSLA Enforcement Service as well as from the Trust Management Service. This illustrates the complete integration of Security Self-Management with SUPERCLOUD project use-case scenarios.

<sup>4</sup>A more detailed description of the SSLA Enforcement Service can be found in Deliverable D2.3 [10].

## Chapter 6 Components Access and Installation

This Chapter presents how to download and deploy the security services presented in this Deliverable.

### 6.1 SSLA Specification Platform

The SSLA Specification Platform is encapsulated in a Web-Server Docker. The component can be downloaded from the SUPERCLOUD repository<sup>1</sup>. Once the image downloaded, the `start.sh` script shell needs to be executed. The script takes care of the deployment of the image that contains the web-server hosting the SUPERCLOUD frontal.

The successful deployment of the SSLA Specification Platform can be verified by accessing the url : `http://Docker-Machine-IP:8080/SSLA/`. The user should be prompted the welcome page depicted in Figure 3.1.

### 6.2 SSLA Negotiation Platform

The multi-agent based SSLA Negotiation Platform presented in Section 3.2 is provided as a JAVA jar application. The application can be downloaded from the SUPERCLOUD repository<sup>2</sup>. Once downloaded, the following steps are needed to deploy the market place.

- The command `java -jar RunMarketPlace.jar` will start the multi-agent platform representing the cloud market place.
- Once the platform started, the user will be prompted the GUI depicted in Figure 3.12. This interface will allow the user to create an agent by specifying its name and selecting the XML file containing the cloud service requirements/constraints. The procedure is described in Section 3.2.1.1.
- If the negotiation assistant agent succeeded to find appropriate offers/clients, an agreement is reached and an SSLA file is generated. It is this file that is used by Security Orchestrator to configure and deploy security services as described in Chapter 4.

### 6.3 Security Orchestrator

The Orchestrator is a core component in SUPERCLOUD Self-Management of Security. It is split into three sub-modules, each responsible of a specific phase of the process presented earlier: interpretation, configuration, deployment and management (cf., Section 4.1.1). Next, we describe how these modules can be downloaded and tested. Each module is currently provided as a Java JAR application. In what follows, we list the necessary steps to download and run them.

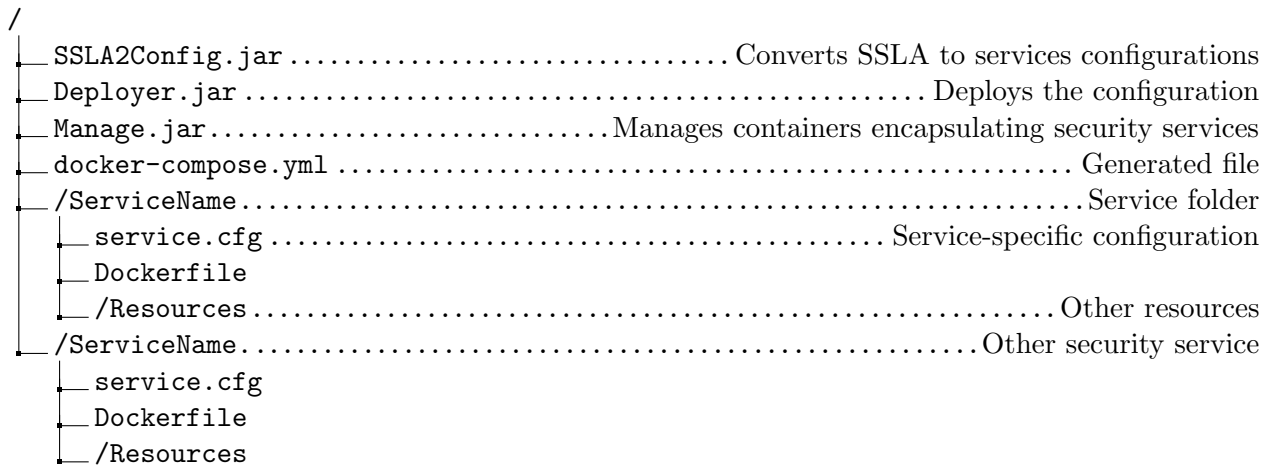
- Download the security service deployment module from the SUPERCLOUD repository<sup>3</sup>.

<sup>1</sup><https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP1/selfmanagement/specification>

<sup>2</sup><https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP1/selfmanagement/negotiation>

<sup>3</sup><https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP1/selfmanagement/>

- The component needs to be placed in the root directory containing security services resources as follows:



- Execute the `SSLA2Config.jar` to launch the processing of SSLA files and the generation of configuration files.
- Execute `java -jar Deployer.jar` to deploy security services on the local machine.
- An execution trace will be displayed to verify that services are running.

## 6.4 Security Services

This Section describes the procedure to download and deploy SUPERCLOUD Self-Management Security Services presented in Chapter 4. The procedures explaining how to access and deploy Security Services that have been implemented in each SUPERCLOUD sub-framework (i.e., Compute, Data and Network) are described in dedicated Deliverables (i.e., Deliverable D2.3 [10], D3.3 [2] and D4.3 [19]).

### 6.4.1 Authorization Service

The Authorization Service is provided as a deploy-ready Docker image encapsulating the OrBAC REST API. We present hereafter the procedure to follow to run and test the service. The following deployment process has been tested in Linux and Mac OS and Windows.

- Download the tar compressed file containing the service Docker image<sup>4</sup>.
- The archive contains :
  - The Authorization Service repository that wraps the OrBAC REST API service running using Apache Axis 2.
  - A Docker file that specifies the way the image should be built.
  - A `start.sh` shell script that runs the service.
- Launch the `./start.sh` script to start the Authorization Service.
- Test if the service is running.
  - Go to the URL address:  
`http://localhost:8080/axis2/services/SupercloudAuthorizationService?wsdl`
  - Type the command-line: `curl -get` with the same URL above.

<sup>4</sup><https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP1/services/authorization>

### 6.4.2 Monitoring Service

A first version of the Monitoring Service is accessible at <https://github.com/Orange-OpenSource/vespa-core>.

### 6.4.3 SSLA Services

As presented in Section 2.2, the SSLA Enforcement Service is provided as a deploy-ready Docker image. It contains the web application responsible for displaying the information collected by the Monitoring Service and aggregated by the SSLA engine. We present hereafter the procedure to download, deploy and test the services on the user's machine<sup>5</sup>.

- Download the tar compressed file containing the service Docker image<sup>6</sup>.
- Launch the `./start.sh` script to start the SSLA Enforcement Service.
- To test if the service is running, go to the URL address : `http://localhost:80`
- This component displays the fulfillment of SSLA based on metrics from the Storage Service.

### 6.4.4 Trust Management Service

We present here after the procedure to retrieve and deploy the Software Trust Service<sup>7</sup>. Like other Self-Management of Security services, the Trust Service is available as a deploy-ready Docker image.

- Download the tar compressed file containing the service Docker image<sup>8</sup>.
- Launch the `./startSTS.sh` script to start the Software Trust Service.
- Once the trust service starts, it can be accessed through standard REST calls. For instance, `curl -XGET ://Docker-Machine-IP:8000/trust` will retrieve the trust values from the service.

---

<sup>5</sup>The following deployment process has been tested in Linux and Mac OS and Windows 10.

<sup>6</sup><https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP1/services/ssl>

<sup>7</sup>The following deployment process has been tested on Mac OS, but tests have been conducted to verify that the services would run similarly on any other operating system.

<sup>8</sup><https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP1/services/trust/>

## Chapter 7 Conclusion

### 7.1 Summary

This Deliverable reports on the implementation of the Security Self-Management Framework. It also describes the demonstrators of security services developed within SUPERCLOUD at the compute, data and network levels. To proceed, we first presented in Chapter 2 an overview of the Security Self-Management Architecture and the corresponding specification and deployment approaches.

Then, we presented in Chapter 3 the SLA Specification and Negotiation framework that captures Cloud Services Customers (CSCs) and Cloud Services Providers (CSPs) requirements that are used to negotiate cloud services among CSCs and CSPs. In Chapter 4, we provided insight on the implementation of Security Orchestrators and Services. In Chapter 5, we reported on the integration of Security Self-Management Services with SUPERCLOUD *Compute*, *Data* and *Network* planes. Finally, in Chapter 6, we presented how the services presented in this Deliverable can be downloaded and deployed.

### 7.2 Future Works

The last period of the project will be dedicated to the full integration of all SUPERCLOUD security services into a unified and unique Security Self-Management Framework. The integration action includes also the complete integration of the Framework with SUPERCLOUD *compute*, *data* and *network* sub-frameworks. For some services, such as monitoring (cf., Section 4.2.2), there is already a planned evolution. Hence, an additional integration process might be needed. Finally, we will also devote some efforts towards deployment of SUPERCLOUD Security Self-Management in the project testbed.

## List of Abbreviations

API	Application Programming Interface
EC	European Commission
CA	Customer Agent
CFP	Call For Proposal
CMP	Cloud Market Place
CSC	Cloud Services Customer
CSP	Cloud Services Provider
DDoS	Distributed Denial of Service
DF	Directory Facilitator
DPI	Deep Packet Inspection
FIPA	Foundation for Intelligent Physical Agents
FIPA-ACL	FIPA Agent Communication Language
GDPR	General Data Protection Regulation
JADE	Java Agent Development
JSON	JavaScript Object Notation
NIDS	Network Intrusion Detection
OrBAC	Organization-Based Access Control
OS	Operating System
PA	Provider Agent
PLO	Protection Level Objective
QoP	Quality of Protection
QoS	Quality of Service
REST	Representational State Transfer
SES	SSLA Enforcement Service
SLO	Service-Level Objective
SMS	Self-Management of Security
SDN	Software-Defined Networking
SSLA	Security Service Level Agreement
STS	Software Trust Service
PEP	Policy Enforcement Point
U-Cloud	User Cloud
URL	Uniform Resource Locator
VM	Virtual Machine
VESPA	Virtual Environments Self-Protecting Architecture
XML	eXtensible Markup Language

## Bibliography

- [1] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification. 2002. Accessed: 2016-06-02. URL: <https://www.ogf.org/documents/GFD.107.pdf>.
- [2] Alysson Bessani, Mario Münzer, Sébastien Canard, Nicolas Desmoulins, Marie Paindavoine, Marko Vukolic, and Daniel Pletea. D3.3 - Proof-of-Concept Prototype for Data Management. *SUPERCLOUD*, 2017.
- [3] Docker Compose. Docker compose tool. 2016. URL: <https://docs.docker.com/compose/>.
- [4] Docker. Docker tool, 2017. URL: <https://docs.docker.com/>.
- [5] Docker Swarm, 2017. Accessed: 2017-07-13. URL: <https://docs.docker.com/swarm/>.
- [6] FIPA. FIPA SL Content Language Specification. 2002. URL: <http://www.fipa.org/specs/fipa00008/SC00008I.html>.
- [7] Floodlight-Project. Floodlight Controller, 2016. Accessed: 2016-10-13. URL: <http://http://www.projectfloodlight.org/floodlight/>.
- [8] Telecom Italia. JADE - Java Agent DEvelopment Framework, 2010 (accessed July 10, 2017). URL: <http://jade.tilab.com>.
- [9] Anas Abou El Kalam, Salem Benferhat, Alexandre Miège, Rania El Baida, Frédéric Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, and Gilles Trouessin. Organization Based Access Control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, 2003.
- [10] Marc Lacoste, Mario Münzer, Felix Stornig, Alex Palesandro, Denis Bourge, Charles Henrotte, Housseem Kanzari, Marko Vukolic, Jagath Weerasinghe, Reda Yaich, Nora Cuppens, Frédéric Cuppens, Markus Miettinen, Ferdinand Brassler, Tommaso Frassetto, and Daniel Pletea. D2.3 - Proof-of-Concept Prototype of Secure Computation Infrastructure and SUPERCLOUD Security Services. *SUPERCLOUD*, 2017.
- [11] Marc Lacoste, Yvan Rafflé, Fano Ramparany, Gregory Blanc, Fabien Charmet, Gitesh Vernekar, Krzysztof Oborzyński, and Paulo Sousa. D5.1 - use case requirements, specification, and evaluation plan. *SUPERCLOUD*, 2016. URL: [supercloud-project-archive/SC-D5.1-Use-case-requirements-specification-evaluation-plan-C0-M18.pdf](https://supercloud-project-archive/SC-D5.1-Use-case-requirements-specification-evaluation-plan-C0-M18.pdf).
- [12] Marc Lacoste, Benjamin Walterscheid, Alex Palesandro, Aurélien Wailly, Ruan He, Yvan Rafflé, Jean-Philippe Wary, Yanhuang Li, Sören Bleikertz, Alysson Bessani, Reda Yaich, Sabir Idrees, Nora Cuppens, Frédéric Cuppens, Ferdinand Brassler, Jialin Huang, Majid Sobhani, Krzysztof Oborzynski, Gitesh Vernekar, Meilof Veenigen, and Paulo Sousa. D2.1 - Architecture for Secure Computation Infrastructure and Self-Management of VM Security. *SUPERCLOUD*, 2015. URL: <https://supercloud-project.eu/downloads/SC-D2.1-Secure-Computation-Infrastructure-PU-M09.pdf>.

- [13] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008. URL: <http://doi.acm.org/10.1145/1355734.1355746>, doi:10.1145/1355734.1355746.
- [14] Markus Miettinen, Mario Münzer, Felix Stornig, Marc Lacoste, Alex Palesandro, Denis Bourge, Charles Henrotte, Housseem Kanzari, Ruan He, Marko Vucolic, Jagath Weerasinghe, Sabir Idrees, Reda Yaich, Nora Cuppens, Frédéric Cuppens, Ferdinand Brassler, Raad Bahmani, Tommaso Frassetto, David Gens, Daniel Pletea, and Peter van Liesdonk. D2.2 - Secure Computation Infrastructure and Self-Management of VM Security. *SUPERCLOUD*, 2016.
- [15] Mario Münzer, Sébastien Canard, Marie Paindavoine, Andre Nogueira, Antonio Casimiro, João Sousa, Joel Alcântara, Tiago Oliveira, Ricardo Mendes, Alysson Bessani, Christian Cachin, Simon Schubert, Caroline Fontaine, Daniel Pletea, Meilof Veeningen, and Jialin Huang. D3.2 - Specification of Security Enablers for Data Management. *SUPERCLOUD*, 2016.
- [16] Tiago Oliveira, Ricardo Mendes, and Alysson Bessani. Exploring key-value stores in multi-writer byzantine-resilient register emulations. In *Proc. of the 20th International Conference On Principles Of Distributed Systems – OPODIS’16*, December 2016.
- [17] OpenStack Watcher. URL: <https://github.com/openstack/watcher>.
- [18] Alex Palesandro, Marc Lacoste, Nadia Bennani, Chirine Ghedira Guegan, and Denis Bourge. Putting Aspects to Work for Flexible Multi-Cloud Deployment. In *IEEE International Conference on Cloud Computing (CLOUD)*, 2017.
- [19] Fernando M. V. Ramos, Nuno Neves, Ruan He, Pascal Legouge, Marc Lacoste, Nizar Kheir, Redouane Chekaoui, Medhi Boutaka, Eric Vial, Max Alaluna, Khalifa Toumi, Rishikesh Sahay, and Gregory Blanc. D4.3 - Proof-of-concept Prototype of the Multi-Cloud Network Virtualization Infrastructure. *SUPERCLOUD*, 2017.
- [20] Paulo Sousa, Gregory Blanc, Krzysztof Oborzyński, Bruno Ferreira, and Joana Cruz. D5.2 - use-case demonstrators. *SUPERCLOUD*, 2017.
- [21] VESPA: Virtual Environment Self-Protecting Architecture. URL: <https://github.com/Orange-OpenSource/vespa-core>.
- [22] R. Yaich, N. Cuppens, and F. Cuppens. Enabling Trust Assessment In Clouds-of-Clouds: A Similarity-Based Approach. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*, 2017. doi:10.1145/3098954.3098970.
- [23] Reda Yaich, Sabir Idrees, Nora Cuppens, Frédéric Cuppens, Marc Lacoste, Nizar Kheir, Ruan He, Khalifa Toumi, Krzysztof Oborzynski, Meilof Veeningen, and Paulo Sousa. D1.2 - SUPERCLOUD Self-Management of Security Specification. *SUPERCLOUD*, 2015. URL: [https://supercloud-project.eu/downloads/SC-D1.2-Self-Management\\_Security\\_Specification-PU-M09.pdf](https://supercloud-project.eu/downloads/SC-D1.2-Self-Management_Security_Specification-PU-M09.pdf).