D1.3
SUPERCLOUD Architecture Implementation

Project number:	643964
Project acronym:	**SUPERCLOUD**
Project title:	User-centric management of security and dependability in clouds of clouds
Project Start Date:	1st February, 2015
Duration:	36 months
Programme:	H2020-ICT-2014-1
Deliverable Type:	Demonstrator
Reference Number:	ICT-643964-D1.3 / 1.0
Work Package:	WP 1
Due Date:	July 2017 - M30
Actual Submission Date:	31st July, 2017
Responsible Organisation:	TUDA
Editor:	Markus Miettinen
Dissemination Level:	PU
Revision:	1.0
Abstract:	This report describes the structure of the SUPERCLOUD architecture implementation and the roles purpose of its different components. It also describes the integration of components between the layers of the architecture and provides download and installation instructions for the software components.
Keywords:	implementation, software, architecture, computation, data storage, network, security management, virtualization

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.

Editor

Markus Miettinen (TUDA)

Contributors (ordered according to beneficiary numbers)

Mario Münzer (TEC)
Marc Lacoste, Sébastien Canard, Marie Paindavoine (ORANGE)
Marko Vukolić (IBM)
Alysson Bessani, Fernando Ramos, Nuno Neves, Eric Vial (FCiencias.ID)
Reda Yaich, Grégory Blanc (IMT)
Markus Miettinen, Ferdinand Brasser, Tommaso Frassetto (TUDA)
Daniel Pletea (PEN)

Disclaimer

Executive Summary

In this document we describe the implementation of the SUPERCLOUD architecture. The architecture provides an abstraction layer on top of which SUPERCLOUD users can realize SUPERCLOUD services encompassing secure computation workloads, secure and privacy-preserving resilient data storage and secure networking resources spanning across different cloud service providers' computation, data storage and network resources. The components of the SUPERCLOUD architecture implementation are described. Integration between the different layers of the architecture (computing security, data protection, network security) and with the facilities for security self-management is also highlighted. Finally, we provide download and installation instructions for the released software components that can be downloaded from our common SUPERCLOUD code repository.

Contents

Chapter 1 Introduction **1**
Chapter 2 Architecture implementation overview **2**
2.1 Architecture frameworks . . . 2
2.2 Architecture implementation . . . 3
Chapter 3 Computing security framework **4**
3.1 Components . . . 4
3.2 Integration with data protection framework . . . 5
3.2.1 Extending isolation to data protection . . . 5
3.2.1.1 Data access obfuscation . . . 5
3.2.1.2 Secure key management . . . 6
3.2.2 Managing SSLAs for geolocation-based data replication . . . 6
3.2.2.1 SLA management for geolocation-aware data replication . . . 6
3.2.2.2 Location-awareness policies for SLAs . . . 7
3.3 Integration with network security framework . . . 8
3.3.1 Integrating computing security orchestration with network security . . . 9
3.3.2 Integrating VM orchestration with network hypervisor . . . 9
3.3.3 Extending Cloud FPGAs to the network . . . 10
3.4 Integration with security self-management . . . 11
Chapter 4 Data protection framework **12**
4.1 Components . . . 12
4.1.1 Cryptographic libraries (attribute-based encryption and deduplication) . . . 12
4.1.1.1 Data anonymization tool . . . 13
4.1.2 Hyperledger Fabric . . . 13
4.1.2.1 Component description . . . 14
4.1.2.2 Integration with use cases . . . 14
4.2 Integration with security self-management . . . 14
4.2.1 Monitoring of data access failures . . . 14
Chapter 5 Network security framework **16**
5.1 Components . . . 16
5.2 Integration with computing security framework . . . 17
5.2.1 Integration of network hypervisor with VM orchestrator . . . 17
5.3 Integration with data protection framework . . . 18
5.3.1 Integration of SDN control with BFT SMR . . . 18
5.4 Integration with security self-management . . . 18
5.4.1 Integration of network security with security monitoring . . . 19
5.4.2 Integration of network hypervisor with network security . . . 19
Chapter 6 Installation **20**
6.1 Computing framework components . . . 20
6.1.1 Virtualization and orchestration and micro-hypervisor . . . 20
6.1.2 Isolation . . . 20
6.1.3 Security Orchestrator . . . 20
6.1.4 Authorization . . . 21

6.1.5 Monitoring . . . 21
6.1.6 Security SLA . . . 21
6.1.7 Software trust . . . 21
6.2 Data protection framework components . . . 21
6.2.1 Janus . . . 21
6.2.2 Hyperledger Fabric . . . 22
6.3 Network security framework components . . . 22
6.3.1 Multi-cloud network orchestrator and network hypervisor . . . 22
6.3.1.1 VM infrastructure . . . 22
6.3.1.1.1 SSH connection . . . 23
6.3.1.1.2 Inter-cloud tunnels . . . 23
6.3.1.1.3 Intra-cloud tunnels . . . 23
6.3.1.1.4 Docker installation . . . 23
6.3.1.2 Orchestrator . . . 23
6.3.1.3 Hypervisor . . . 23
6.3.2 Network security monitoring and appliance chaining . . . 23
6.3.3 Network security policy management . . . 24
6.4 Security self-management components . . . 24
6.4.1 Manual deployment . . . 24
6.4.2 Orchestrated deployment . . . 24

Chapter 7 Summary **25**

Chapter 8 List of Abbreviations **26**

Bibliography **28**

List of Figures

2.1 SUPERCLOUD overall component architecture 2

3.1 Overview of SUPERCLOUD computing security framework 4
3.2 Integration with data protection framework 5
3.3 SSLA and georeplication integration 7
3.4 SSLA location-aware Janus 8
3.5 Integration with network security framework 8
3.6 OrBAC-based orchestration of network security policies 9
3.7 VM virtualization and orchestration, network hypervisor, multi-cloud orchestrator [21] 10
3.8 A first interconnection scenario 11

4.1 Integrating Janus with advanced cryptographic component: upload 13
4.2 Janus and Self-Management Integration 15

5.1 SUPERCLOUD network security framework overview 16
5.2 Integration with computing security framework 17
5.3 Integration with data protection framework 18

Chapter 1 Introduction

This Deliverable describes the implementation of the SUPERCLOUD architecture that was specified in Deliverable D1.1 [11] and presented in [8]. The SUPERCLOUD architecture consists of three layers, refined into corresponding frameworks for *computing security*, *data protection* and *network security*. All frameworks are supported by an orthogonal *security self-management infrastructure*. This document provides descriptions of the components for the frameworks of each layer of the architecture. It also outlines the integration interfaces between the different layers.

In Chapter 2 we first present an overview of the overall SUPERCLOUD architecture implementation and enumerate the components comprising the frameworks for computing security, data protection and network security.

The following chapters then outline each framework separately, and highlight interconnections with other frameworks:

- Chapter 3 describes the components of the *computing security framework*. It presents the components comprising its virtualization and self-management infrastructures. It also describes integration with components of frameworks for data protection and network security and with security self-management.
- Chapter 4 describes the components of the *data protection framework*. It also presents integration with components of frameworks for computing security and for network security and with security self-management.
- Chapter 5 describes the components of the *network security framework*. It also presents integration points with components of computing security and data protection frameworks and with self-management of security.

Installation instructions including guidance on download locations for the discussed architectural components are provided in Chapter 6, before concluding with a summary in Chapter 7.

Chapter 2 Architecture implementation overview

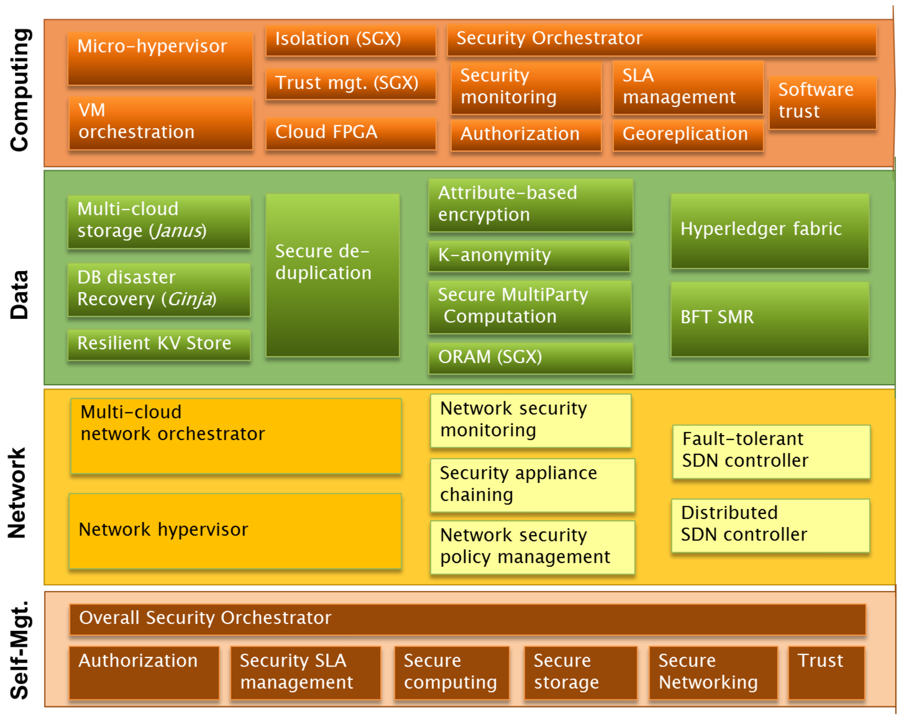

Figure 2.1: SUPERCLOUD overall component architecture

Figure 2.1 shows the overall SUPERCLOUD component architecture. As outlined in Deliverable D1.1 [11] describing the SUPERCLOUD architecture, its purpose is to provide to SUPERCLOUD users a computing, data storage and network abstraction for securely deploying SUPERCLOUD services in a way that is independent of the underlying cloud service providers.
This Deliverable describes the implementation of the overall architecture. The architecture consists of three layers, refined into *computing security*, *data protection* and *network security* frameworks. All frameworks are also supported by an orthogonal *security self-management infrastructure* providing facilities for automation of user-level control of security and privacy settings.

2.1 Architecture frameworks

The role of the **computing security framework** (cf. Chapter 3) is to provide facilities for executing computing workloads on execution environments across different cloud service providers and orchestrating their functions. The framework provides strict isolation of the execution environments of different SUPERCLOUD users that can be enhanced with hardware-based isolation and trust management. According to the requirements specified in Deliverable D1.1 [11], this framework also enables to control the deployment of computing workloads based on geographical constraints related to, e.g., legal requirements on jurisdictional borders.

The **data protection framework** (cf. Chapter 4) provides facilities for secure and resilient storage of SUPERCLOUD user data across cloud service providers. It provides advanced protection measures based on, e.g., attribute-based encryption or assurance of anonymity of data. It also features blockchain-based mechanisms for non-repudiation of records to provide reliable data storage for applications with high requirements towards data confidentiality and user privacy.

The **network security framework** (cf. Chapter 5) provides a secure network abstraction deployable over several cloud service providers' networking resources. It is based on a multi-cloud network hypervisor and a multi-cloud orchestrator to realize desired network topologies on top of the physical network resources. It also provides facilities for security self-management allowing configuration and monitoring of the security of the network and of its components.

2.2 Architecture implementation

The subsequent Chapters provide brief outlines of the implemented components of each framework. They describe the integration interfaces of individual components to other frameworks and components of the SUPERCLOUD architecture.

For implementation details of the components, we refer the reader to:

- Deliverable D2.3 "Proof-of-Concept Prototype of Secure Computation Infrastructure and SUPERCLOUD Security Services" [9] for the secure computation-related components;
- Deliverable D3.3 "Proof-of-Concept Prototype for Data Management"[3] for data protection-related components;
- and to Deliverable D4.3 "Proof-of-Concept Prototype of the Multi-Cloud Network Virtualization Infrastructure" [16] for the network security-related components.

Each Chapter also provides an overview of the integration of each framework with components for self-management of security - described in detail in Deliverable D1.4 [26].

Chapter 3 Computing security framework

This Chapter presents the SUPERCLOUD computing security framework, described more extensively in Deliverable D2.3 [9]. We give an overview of the framework's main components (Section 3.1). We then show how the framework interacts with SUPERCLOUD frameworks for data protection (Section 3.2, see Deliverable D3.3 [3]), network security (Section 3.3, see Deliverable D4.3 [16]) and self-management of security (Section 3.4, see Deliverable D1.2 [27]).

3.1 Components

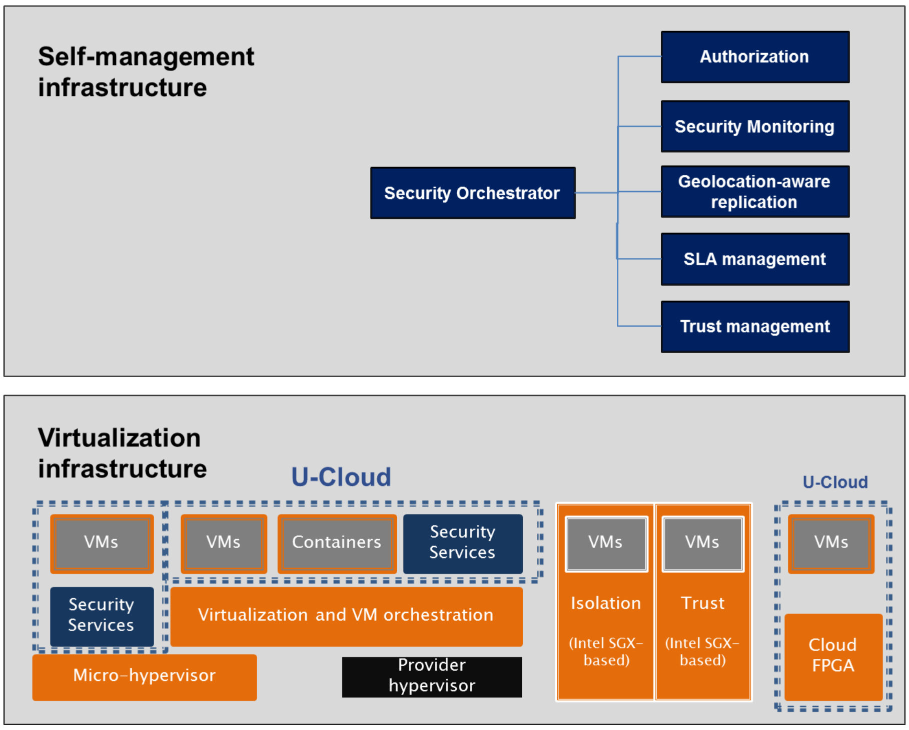

Figure 3.1: Overview of SUPERCLOUD computing security framework

Figure 3.1 shows the main components of the computing security framework. Components may be grouped into two-distinct sub-infrastructures:

- The **virtualization infrastructure** provides a distributed abstraction layer for running U-Clouds on computing resources of different cloud providers. It includes low-level infrastructure security services for: horizontal orchestration to achieve unified security management of computing elements regardless of providers; vertical orchestration, extending user-control over U-Cloud security into the lower layers of the provider infrastructure using a micro-hypervisor; and strong isolation and trust management between computing elements, relying on hardware security mechanisms (e.g., Intel SGX, FPGAs).

- The **self-management infrastructure** implements autonomic security management for the distributed cloud. This infrastructure provides a number of security services that may be orchestrated on-demand to guarantee (self-) protection of U-Clouds on top of the distributed virtualization infrastructure. Provided services include: authorization to perform resource access control at different infrastructure levels; security monitoring across infrastructure layers and providers; geolocation-aware data replication; management of security SLAs; and software-level trust management between users or providers.

3.2 Integration with data protection framework

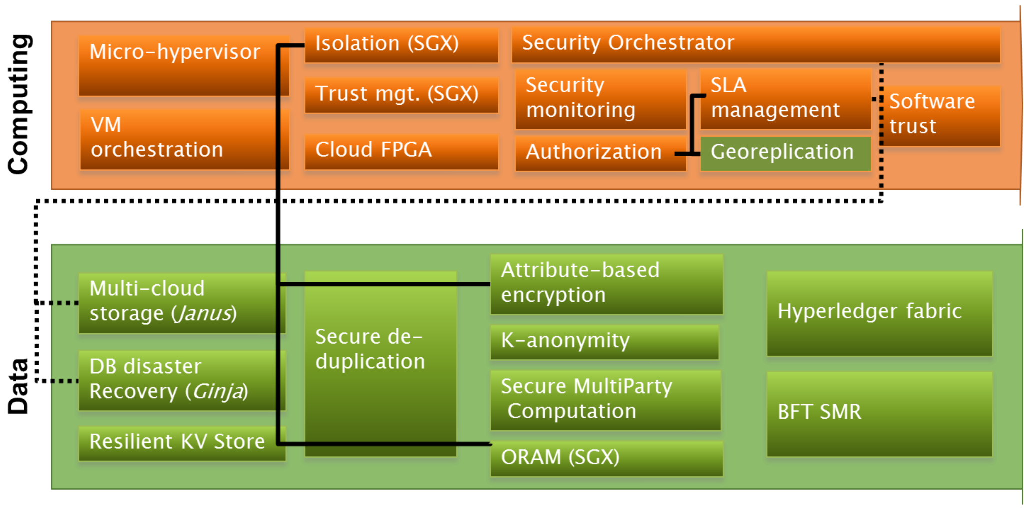

Figure 3.2: Integration with data protection framework

Figure 3.2 shows how components from the computing framework interact with those from the data protection framework. Two inter-framework connections are particularly interesting: extending hardware-based isolation to protect data (Section 3.2.1) and managing SLAs for geolocation-aware data replication (Section 3.2.2).

3.2.1 Extending isolation to data protection

This type of integration may cover both obfuscating data access (Section 3.2.1.1) and using isolation technology to achieve secure key management (Section 3.2.1.2).

3.2.1.1 Data access obfuscation

The Intel SGX execution environment (as described in more detail in Deliverable D2.3 [9], Section 3.3) provides isolation of data and code. Concretely, the memory which contains data and code cannot be accessed by any other entity except the execution environment itself. Data often processed are loaded on-demand from external resources like local storage, or over the network. Using appropriate cryptographic methods, the data cannot be inspected by an external, untrusted entity (the adversary). However, which data are accessed, at which point in time and in which order accesses happen can reveal sensitive information. For instance, if an execution environment instance is known to process medical data which are related to a certain disease and if the adversary observes access to a particular patient's file, the adversary may infer that this patient has (is suspected to have) this particular disease, which is clearly sensitive information.

To prevent such information leakage the concept of *Oblivious Random Access Memory (ORAM)* has been introduced and studied extensively [7, 20]. With ORAM, data are stored in a randomized order and the adversary cannot identify individual data sets.
In the example above, this means that the adversary would learn that *some patient's file* has been accessed. However, he would not be able to distinguish the files of different patients. Furthermore, ORAM not only randomizes data once, but on every access data are re-randomized to prevent the adversary from gradually learning about individual files.
The ORAM scheme can be integrated into the SGX execution environment as an abstraction layer for accesses to external resources, like local storage or network-attached storage. This layered integration makes the protection offered by ORAM transparent to the software executed inside the execution environment. Notably, the code interpretation layer (Python, see Deliverable D2.3 [9], Section 3.3 for details) of the SGX execution environment allows interception of any access to external resources to protect them with an ORAM-secured access.

3.2.1.2 Secure key management

Secure key management is arguably one of the most difficult challenges when it comes to implementing cryptographic protocols. In the case of SUPERCLOUD, this problem mainly arises when manipulating the Attribute-Based Encryption (ABE) library integrated into CLINIdATA, one of the SUPERCLOUD use case systems (cf. Deliverable D5.2 [19]). We propose to leverage *secure cloud enclaves*, such as Intel SGX, to help towards reliable key management and to prevent any leakage while storing and manipulating users' secret keys.
In the ABE library, each user secret key is associated with a set of *attributes*, while each ciphertext is associated with an *access policy*, which is a formula over the whole set of attributes in the system. One secret key can be used to decrypt a ciphertext if and only if its set of attributes satisfies the access policy underlying the ciphertext.
Each user secret key is only generated once and can be secretly stored in the cloud. This secure storage is achieved using the *sealing* feature of Intel SGX. This operation encrypts the cryptographic key under the enclave's secret key, before storing this encryption in a cloud service provider. When a user needs a document to be decrypted, the enclave retrieves the key, decrypts it and uses it inside the protected environment. The document is sent to the user using a secure channel, such as TLS. In this setting, the user's keys are always protected while unused, and only decrypted inside the secure enclave. This enables a user-friendly experience, where the device loss can be managed securely without losing data encrypted under the device's key.
This feature is still under implementation using OpenSGX, and will be integrated into the SUPERCLOUD testbed via OpenStack.

3.2.2 Managing SSLAs for geolocation-based data replication

This type of integration covers managing Security Service Level Agreements (SSLA) for geolocation-based data replication, replication being managed directly above the distributed storage system (Section 3.2.2.1), or integrated directly in a multi-cloud storage system such as Janus (Section 3.2.2.2).

3.2.2.1 SLA management for geolocation-aware data replication

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 [1]. We showcase in this Section how such objective has been achieved in SUPERCLOUD. As illustrated in Figure 3.3, geolocation policies are extracted from the SSLA by the SSLA management service and converted to Organization-Based Access Control (OrBAC) rules. These rules are

[1]GDPR: General Data Protection Regulation.

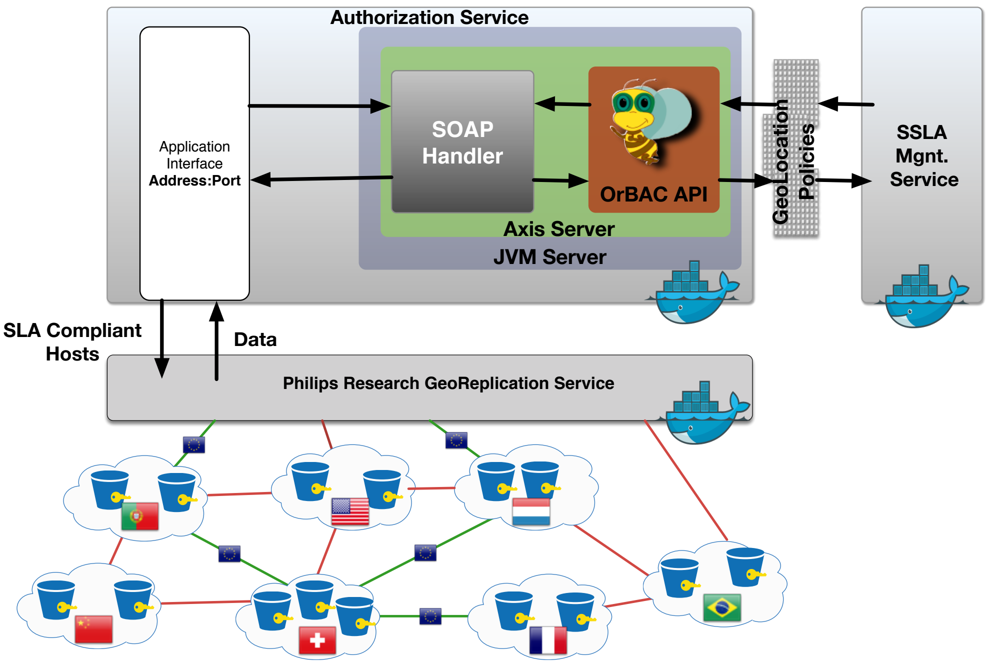

Figure 3.3: Integration of SSLA management and georeplication services

subsequently processed by the authorization service to derive permissions/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.

3.2.2.2 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 of the secure storage provided by Janus.

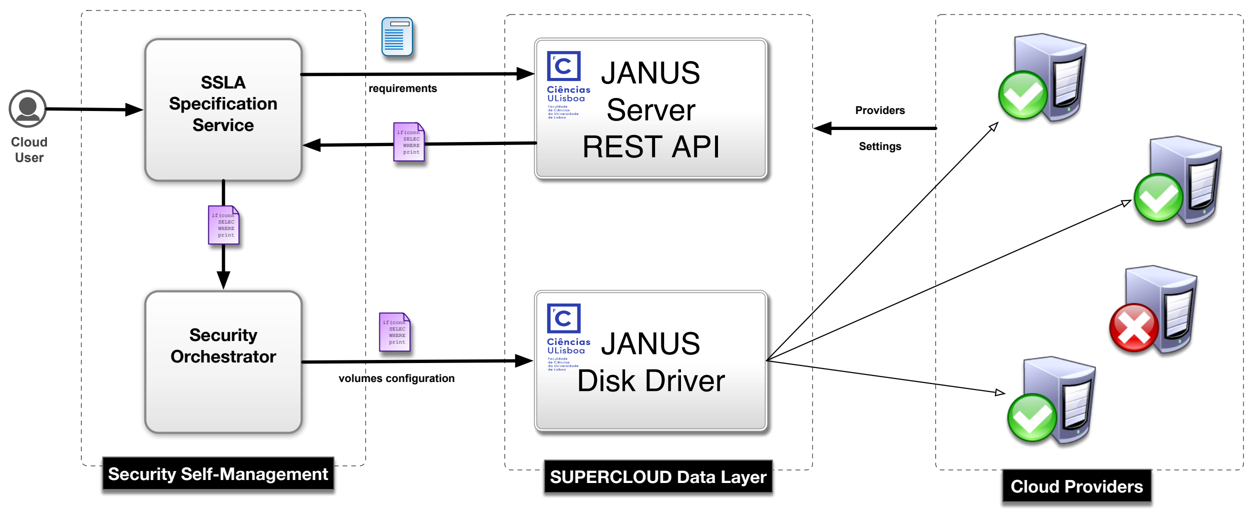

Figure 3.4: SSLA location-aware Janus

As illustrated in Figure 3.4, the location-aware fault-tolerance service is the result of the integration of two components for self-management of security, and two components from the Janus framework. The user first specifies his data volumes requirements through the SSLA specification service[2]. 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.

3.3 Integration with network security framework

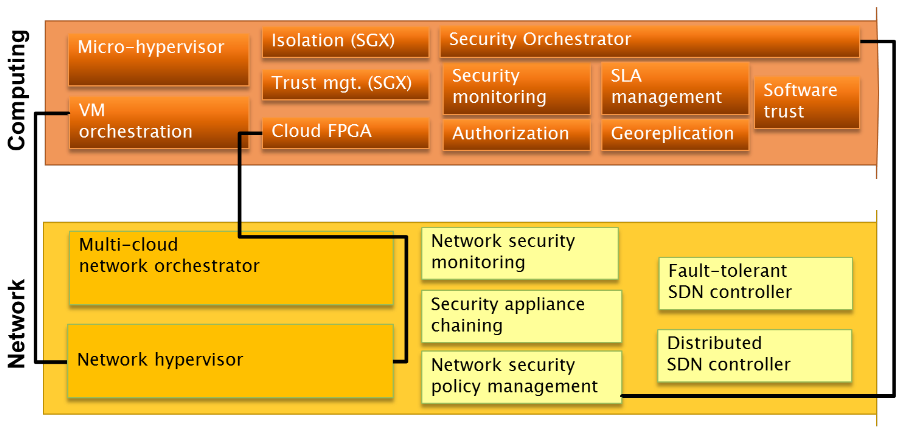

Figure 3.5: Integration with network security framework

Figure 3.5 shows how computing components may interact with network security components. Three inter-framework connections are particularly interesting : how to integrate the security orchestrator for computing resources with the network security module (Section 3.3.1), how to connect VM orches-

[2]This service is presented in detail in Deliverable D1.4 [26].

tration with the network hypervisor (Section 3.3.2), or how to apply the Cloud FPGA component in a networked setting (Section 3.3.3).

3.3.1 Integrating computing security orchestration with network security

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.

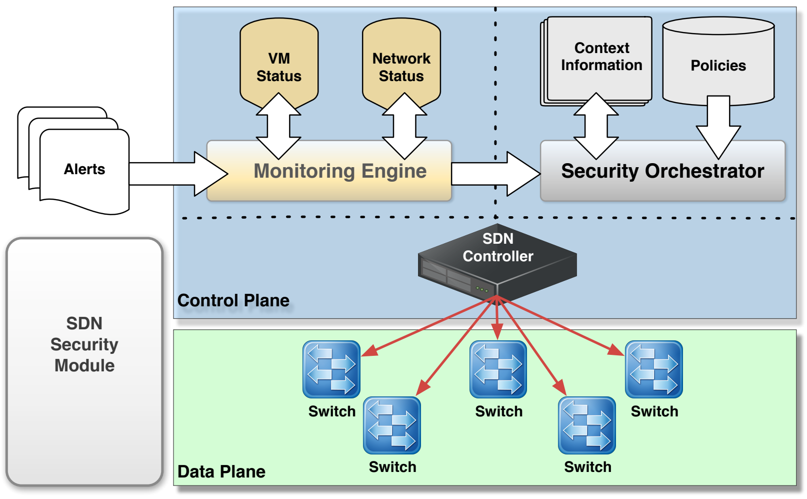

Figure 3.6: OrBAC-based orchestration of network security policies

User security preferences captured and negotiated by the SSLA service are represented as OrBAC authorization rules (i.e., permissions and prohibitions), as shown in the top-left part of Figure 3.6. These policies are then used by the Security Orchestrator to activate and/or deactivate SDN routing paths based on context and monitoring information [21].

3.3.2 Integrating VM orchestration with network hypervisor

This Section shows how the VM virtualization and orchestration computing component (implemented by the MANTUS multi-cloud builder as part of the ORBITS framework [14, 15]) could be interconnected with network hypervisor and multi-cloud orchestrator networking components (implemented by the SIRIUS multi-cloud network virtualisation platform [1, 2]). Figure 3.7 recalls the principles of these components.

- *Computing:* to build the multi-cloud substrate, the computing infrastructure administrator selects the *infrastructure elements* (VMs, containers) and *security services* (mostly for computing) to deploy in the substrate, specified as templates. Both elements are then matched with provider capabilities, dispatched to the different clouds providers, weaved to inject the security services in the infrastructure, and finally instantiated on the different providers. This set of compilation operations results in a (distributed) U-Cloud composing either nested VMs or containers within VMs (depending on chosen virtualization technology) with security services located in different parts of the infrastructure. A distributed orchestration logic enables to achieve U-Cloud multi-provider awareness. The application containers are then deployed in U-Cloud execution environments using an application-level deployment framework.
- *Networking:* the networking infrastructure administrator selects the *infrastructure elements* (VM, containers) to build the substrate. The user selects the *virtual networks* to deploy, com-

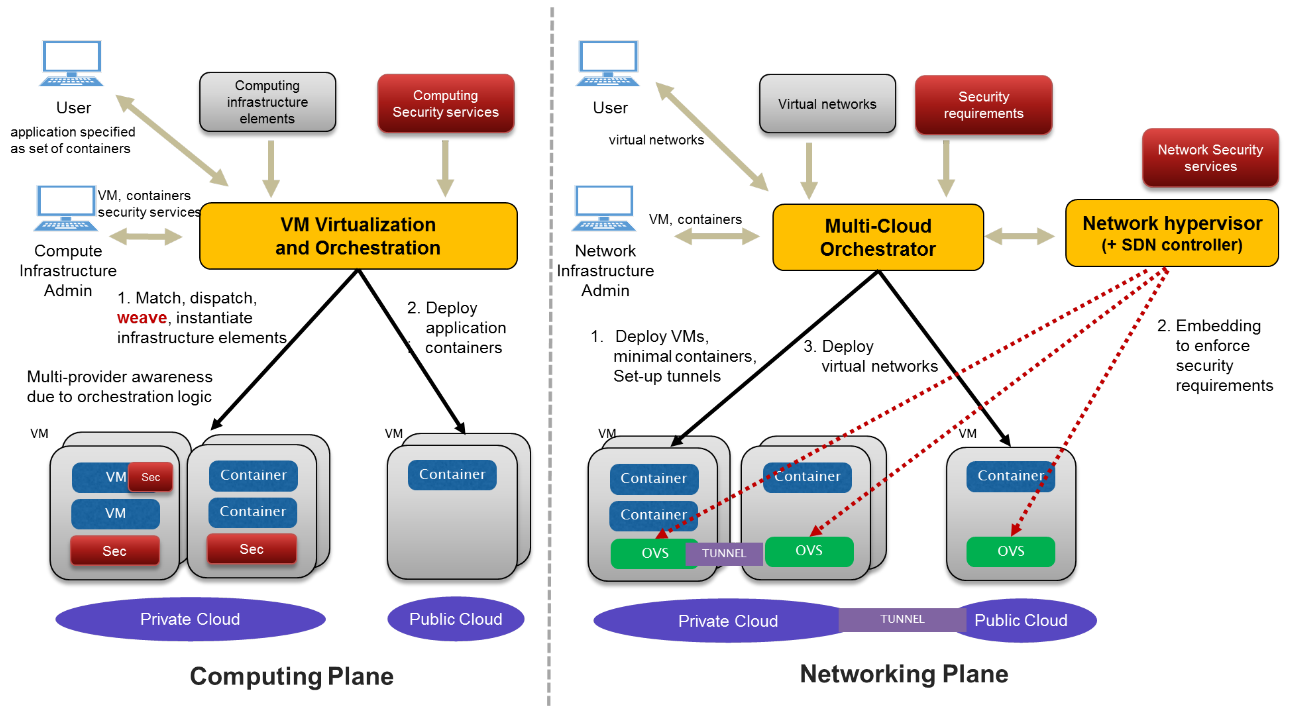
User
application specified as set of containers
Computing infrastructure elements
Computing Security services
VM, containers security services
Compute Infrastructure Admin
VM Virtualization and Orchestration
1. Match, dispatch, weave, instantiate infrastructure elements
2. Deploy application containers
Multi-provider awareness due to orchestration logic
VM
VM
Sec
VM
Sec
Container
Container
Sec
Container
Private Cloud
Public Cloud
Computing Plane
User
virtual networks
Virtual networks
Security requirements
Network Security services
VM, containers
Network Infrastructure Admin
Multi-Cloud Orchestrator
Network hypervisor (+ SDN controller)
1. Deploy VMs, minimal containers, Set-up tunnels
3. Deploy virtual networks
2. Embedding to enforce security requirements
VM
Container
Container
OVS
TUNNEL
Container
OVS
Container
OVS
Private Cloud
TUNNEL
Public Cloud
Networking Plane

Figure 3.7: VM virtualization and orchestration, network hypervisor, multi-cloud orchestrator [21]

posed of a number of containers interconnected according to an arbitrary topology, and their *security requirements*. The multi-cloud orchestrator then deploys the infrastructure elements (VMs, bootstrap containers), builds the necessary tunnels to interconnect networks, and is returned information on the state of the infrastructure such as resource availability. The network hypervisor performs the embedding operation to match the chosen virtual network topology to available resources in the substrate: it instructs the multi-cloud orchestrator to deploy the virtual network containers accordingly. The network hypervisor also updates OVS switch configurations in the infrastructure to enforce user-defined security requirements such as network isolation.

Architectures and deployment processes are thus very similar, and a first level of integration is being explored for corresponding computing and networking components. A first design is shown in Figure 3.8 to make virtual networks of VMs enriched with security services controllable remotely from the network hypervisor. Further results will be reported in Deliverable D2.4.
A possible integration scenario is the following: as previously, the user specifies the application as a set of containers and virtual networks. This knowledge is shared between the VM virtualization and orchestration and the multi-cloud orchestrator components. The infrastructure administrator also specifies the infrastructure elements to deploy: this operation will be shared by VM virtualization and orchestration component (for the private cloud) and by the multi-cloud orchestrator (for the rest of the infrastructure). Specification of security services and of network security requirements and instantiation of infrastructure elements is then performed as before. Each Open vSwitch (OVS) of the private cloud is made controllable remotely by the network hypervisor that can now perform network embedding on all OVS switches of the infrastructure to enforce security requirements. For application-level deployment, the multi-cloud orchestrator deploys virtual networks as previously in its own control perimeter. It also instructs the VM virtualization and orchestration component to deploy the other application containers in the private cloud.

3.3.3 Extending Cloud FPGAs to the network

The Cloud FPGA component of the computing framework enables to off-load tasks from CPUs of the SUPERCLOUD infrastructure such as intensive computations related to cryptography or security. Its

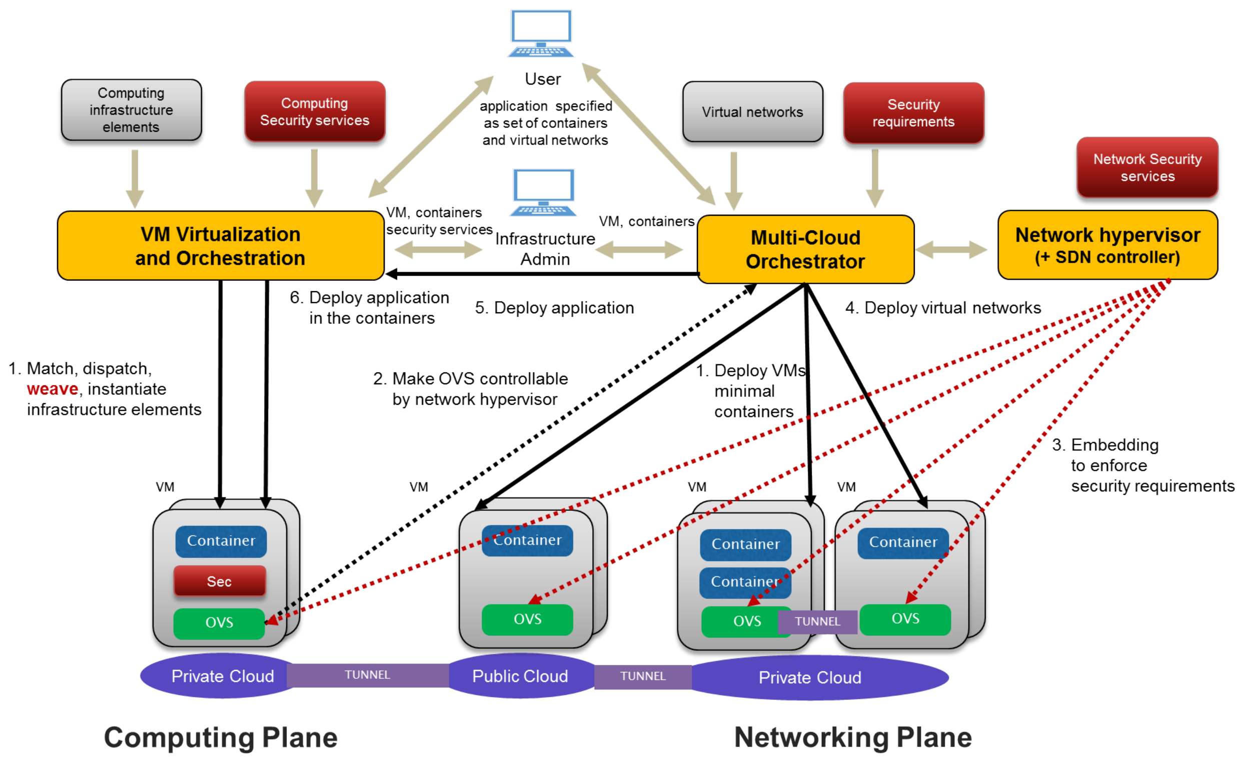
User
application specified as set of containers and virtual networks
Computing infrastructure elements
Computing Security services
Virtual networks
Security requirements
Network Security services
VM Virtualization and Orchestration
VM, containers security services
Infrastructure Admin
VM, containers
Multi-Cloud Orchestrator
Network hypervisor (+ SDN controller)
6. Deploy application in the containers
5. Deploy application
4. Deploy virtual networks
1. Match, dispatch, weave, instantiate infrastructure elements
2. Make OVS controllable by network hypervisor
1. Deploy VMs minimal containers
3. Embedding to enforce security requirements
VM
Container
Sec
OVS
TUNNEL
Private Cloud
Public Cloud
Computing Plane
Networking Plane

Figure 3.8: A first interconnection scenario

design was presented in Deliverable D2.2 [12] and its objective and internal interface in Deliverable D2.3 [9]. Moreover, this component also provides network security benefits to manage a fabric of multiple FPGAs using Software-Defined Networking, within the SUPERCLOUD network plane. More details will be provided in Deliverable D2.4.

3.4 Integration with security self-management

Computing and security self-management components are already tightly integrated by the very design of the computing framework. They can then integrate further with other SUPERCLOUD planes such as storage to introduce location-awareness policies for SLAs (Section 3.2.2). Integration of security orchestration, SLA management, and network security policy management was already described in Section 3.3.1, and will be described in more detail in Deliverable D1.4 [26].

Chapter 4 Data protection framework

This Chapter presents the SUPERCLOUD data management and data protection framework, described more extensively in Deliverable D3.3 [3]. We give an overview of the framework main components in Section 4.1. Interactions of the data management and protection framework with self-management of security (itself detailed in Deliverable D1.2 [27]) are described in Section 4.2.
For information about interaction of the data management framework with the computing framework, we refer the reader to Section 3.2 of this Deliverable.

4.1 Components

The SUPERCLOUD data protection and management components implement secure and dependable data management services in a cloud-of-clouds environment. The overall architecture envisioned for the project data management, and the main contributions in this context were described in previous Deliverables [24, 25, 3].
In a nutshell, this architecture considers many different instantiations of data management services. This includes the use of existing public cloud services like Amazon S3 and Rackspace Files for storing files and support cheap disaster recovery, the design of custom blockchain-like multi-cloud services, and new tools and programming libraries for implementing advanced security features in existing applications.

The main data protection and management components include:

- *Janus*, a cloud-backed storage service that can be configured on the web, by specifying workloads and requirements for data storage, and using a proxy locally to transfer this data to/from a set of cloud storage services selected by the system.
- *Cryptographic libraries* which span *k-anonymity* (for removing privacy-sensitive datasets), *attribute-based encryption (ABE)* (that allows encryption of data records such that only parties satisfying certain attributes can decrypt such records), and *deduplication* (which removes redundant information stored on clouds). These cryptographic libraries integrate with the Janus cloud storage service as described in Section 4.1.1.
- *Hyperledger Fabric*, described briefly in Section 4.1.2, a permissioned blockchain framework that employs Byzantine-resilient consensus on its core. Although Hyperledger Fabric is not 100% supported by SUPERCLOUD, we are contributing with several alternatives for implementing the agreement on the ordering of blocks.

4.1.1 Cryptographic libraries (attribute-based encryption and deduplication)

We present here how advanced cryptographic components can be integrated with Janus as a further avenue of research. We focus our study on both attribute-based encryption (ABE), which allows fine-grained access control, and convergent encryption (CE), which allows secure deduplication.
In a nutshell, the cryptographic component is in charge of generating session keys, that are next used to encrypt the data itself. In order to do so, the component needs to obtain different types of

information, such as an access control policy for ABE or compressed data for convergent encryption. The compressed data has to come from the Janus component to ensure secure deduplication. The session key thus generated is used within the Janus component to encrypt compressed data and proceed with storage into cloud service providers. These interactions in the upload phase are summed up in Figure 4.1.

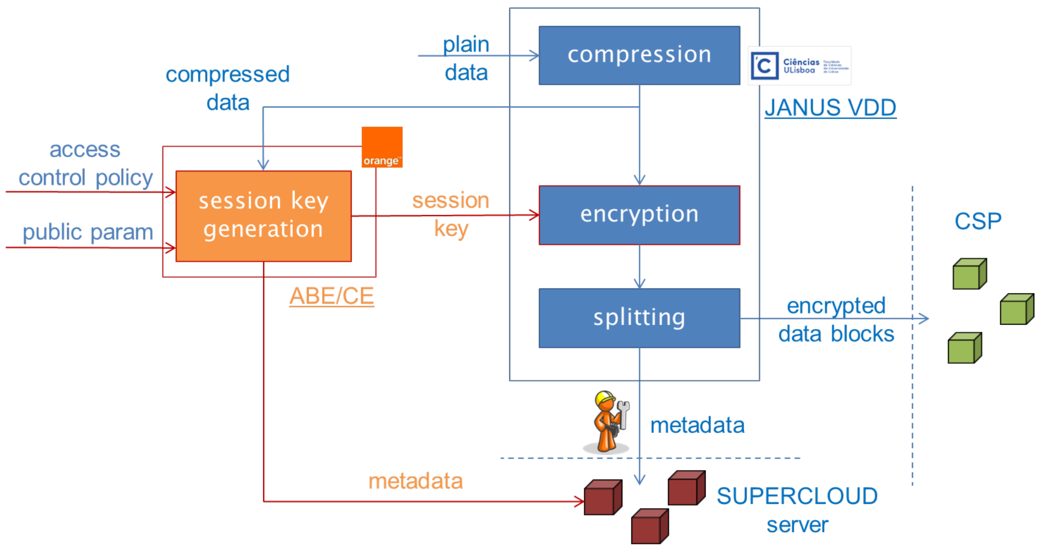

Figure 4.1: Integrating Janus with advanced cryptographic component: upload

For downloading data, the cryptographic component has first to recompute the session key from the user's secret key (or to retrieve it if securely stored via SGX, as described in Section 3.2.1). Then, Janus uses this key to decrypt files before decompressing.

4.1.1.1 Data anonymization tool

Data anonymization techniques open the possibility of releasing personal and sensitive data, while preserving an individual's privacy. The data anonymization tool in this context is among others based on *k-anonymity*, whereby the focus is put on the irreversibility of the released data. The tool aims to calculate 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. A detailed explanation of the OLA algorithm as well as of all including components of the tool can be found in the Deliverable D3.2 [25].

4.1.2 Hyperledger Fabric

In general, *blockchains* are distributed ledgers (typically immutable and totally ordered) of transactions pertaining to distributed applications. Distributed applications may be cryptocurrencies (such as Bitcoin) but also general applications (i.e., state machines, sometimes called *smart contracts*). *Permissioned blockchains* [22, 23] are those blockchains in which the membership of the nodes that hold copies of the ledger is restricted and managed in some way.
Permissioned blockchains work across multiple administrative domains and are a good match for SUPERCLOUD project goals and its data management requirements. SUPERCLOUD project contributes to the Hyperledger Fabric open-source blockchain project and benefits from it. Hyperledger Fabric (HLF)[1] is an open-source project within the Hyperledger umbrella project under the auspices of the Linux Foundation. HLF is a modular general-purpose permissioned blockchain system which can be also seen as a distributed operating system for permissioned blockchains.

[1] `https://github.com/hyperledger/fabric`

The SUPERCLOUD project contributes to the HLF project by influencing its overall architecture, including approach to handling non-determinism in the system [5]. It also serves as an integration vector for State-machine replication related components of SUPERCLOUD as described in Deliverable D3.2 [25]. Notably these include:

- A component that treats non-determinism when replicating arbitrary applications when replicas can fail in an arbitrary (i.e., Byzantine) way [5].
- A component that introduces a novel model for developing reliable distributed protocols called XFT [10].
- A component that empirically evaluates latency-optimization for state-machine replication in WANs and informing the design of novel state-machine replication protocols [18].
- A component that introduces a generic state-transfer tool for partitioned state-machine replication that enables elasticity [13].

The integration of the last two components is done via integration of HLF and the BFT-SMaRt library [4] in which those components were implemented.

4.1.2.1 Component description

The architecture of the component is described in the main architecture document `https://github.com/hyperledger/fabric/blob/master/proposals/r1/Next-Consensus-Architecture-Proposal.md`. This overall architecture has been developed in part in the SUPERCLOUD project.
Other lower-level architecture design documents are available at `https://wiki.hyperledger.org/community/fabric-design-docs`. With the exception of the Simplified Byzantine Fault Tolerance (SBFT) consensus, which was developed in the context of the SUPERCLOUD project, other components have been developed outside SUPERCLOUD, by the community.

4.1.2.2 Integration with use cases

Hyperledger Fabric will serve as a secure globally replicated log for the Maxdata use case. The distributed application that will be BFT-replicated using Hyperledger Fabric will be in this case an append-only immutable log that will be used for auditing access to healthcare records.

4.2 Integration with security self-management

In this Section, we provide an overview of the ongoing integration action between security self-management and Janus, the dependable and secure multi-cloud data storage service[2].

4.2.1 Monitoring of data access failures

As illustrated in Figure 4.2, the *Janus* secure storage service embodies monitoring components that collect information about the health of the servers wherein SUPERCLOUD users' data is hosted.
The collected information concern essentially data *Availability* and the *Latency* witnessed. The collected data are then sent to the Security Storage Service (part of Security Self-Management) to be processed by the SSLA enforcement service (Left part of Figure 4.2). 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 results of this processing is then displayed to allow supervision and arbitration. [3]

[2]We invite the reader to refer to Deliverable D3.3 [3] for more details about Janus.
[3]A more detailed description of the SSLA enforcement service can be found in Deliverable D2.3 [9].

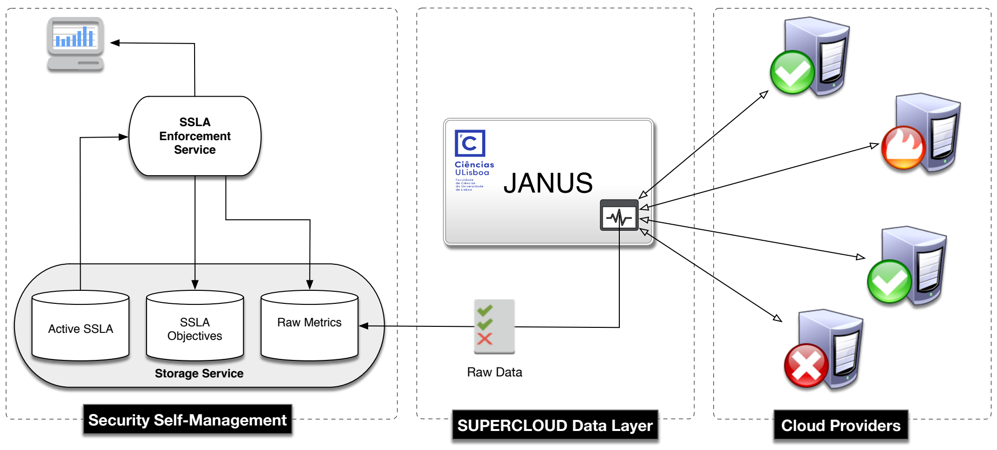

Figure 4.2: Integration of self-management of security and Janus

Chapter 5 Network security framework

This chapter presents a short overview of the network framework and of the interactions with the other SUPERCLOUD components. We start with a short overview of the main components of the framework, in Section 5.1. Then, in Sections 5.2 and 5.3 we explain how the network framework interacts with the computing and data protection frameworks, respectively. Finally, in Section 5.4, we focus on the integration of the network framework with the security self-management modules.

5.1 Components

In this Section we present a short description on the network architecture. For more detail we invite the reader to Deliverable D4.3 [16]. Our solution leverages a substrate infrastructure that entails both public clouds and private datacenters. In our platform tenants can define virtual networks with arbitrary topologies, while making use of the full address space. In addition, we allow them to specify security and dependability requirements for all virtual resources.
The main components of the network framework are shown in Figure 5.1: the multi-cloud network hypervisor, the multi-cloud orchestrator, and the self-management security services.

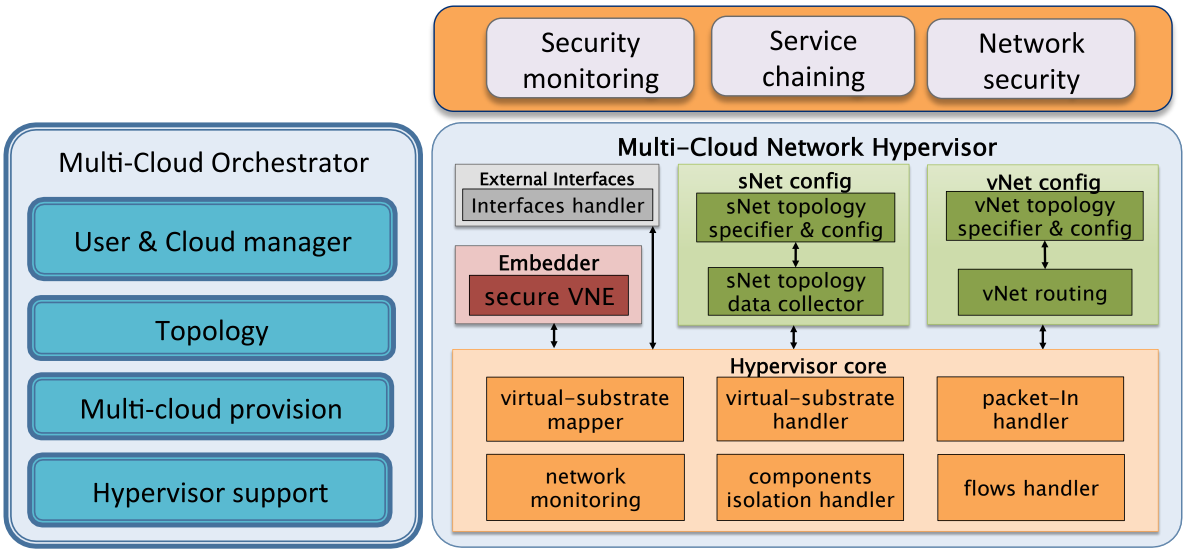

Figure 5.1: SUPERCLOUD network security framework overview

The *cloud orchestrator* is responsible creating the substrate infrastructure by deploying the necessary VMs and containers. This component takes care of the configuration of secure tunnels between participating clouds, normally building a fully connected topology. These tunnels are established in a particular VM, the *gateway*, that acts like an edge router, receiving local packets whose destination is in another cloud and then forwarding them to its peer gateways using the secure tunnels set up. Intra-cloud communications between tenant containers is performed with the use of GRE (Generic Routing Encapsulation) tunnels that are setup between the local VMs, to ensure isolation.

The *network hypervisor* runs as an application on top of a Software-Defined Networking (SDN) controller. This component forms the core of our solution, dealing with the placement of the virtual networks, setting up the necessary network paths, and intercepting all control messages between the substrate infrastructure and the users' virtual networks to enable full network virtualisation.
Finally, the *self-management security services* run on top of the network hypervisor. They include three modules: a security monitor to detect incidents related to security, a network security service with the main function of responding to these incidents, and a service chaining component that allows tenants to compose security service chains.

5.2 Integration with computing security framework

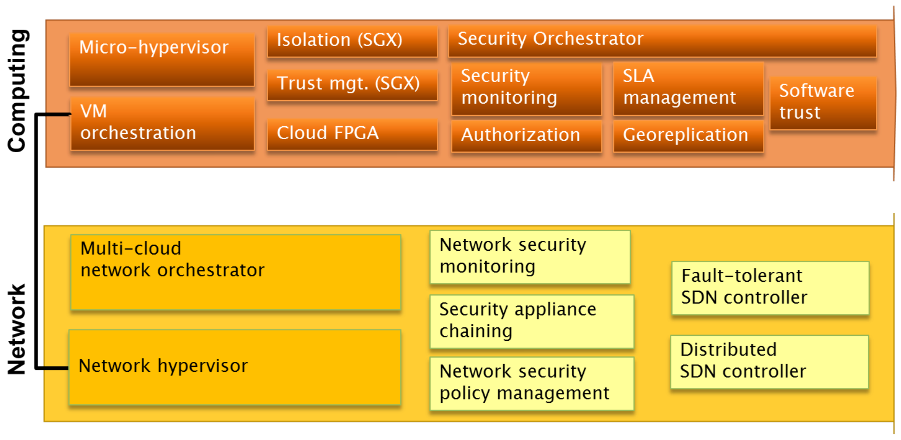

Figure 5.2: Integration with computing security framework

Figure 5.2 shows how networking components may interact with the computing components. An interesting inter-framework connection is notably how to integrate the networking multi-cloud security orchestrator and network hypervisor with the VM orchestration component for computing resources (Section 5.2.1).

5.2.1 Integration of network hypervisor with VM orchestrator

A first possible integration scenario was described in Section 3.3.2. Further results will be reported in Deliverable D2.4.

5.3 Integration with data protection framework

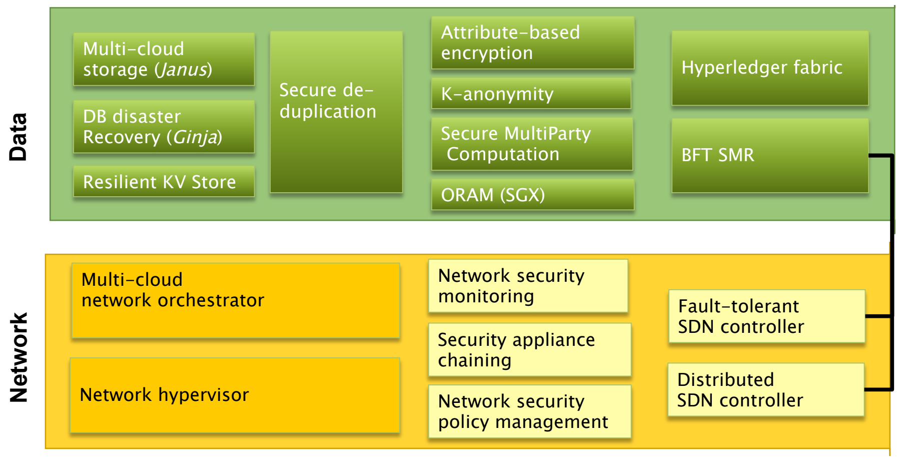
Data
Multi-cloud storage (*Janus*)
DB disaster Recovery (*Ginja*)
Resilient KV Store
Secure de-duplication
Attribute-based encryption
K-anonymity
Secure MultiParty Computation
ORAM (SGX)
Hyperledger fabric
BFT SMR
Network
Multi-cloud network orchestrator
Network hypervisor
Network security monitoring
Security appliance chaining
Network security policy management
Fault-tolerant SDN controller
Distributed SDN controller

Figure 5.3: Integration with data protection framework

Figure 5.3 shows how networking components interact with data protection components. The main inter-framework connection includes integration of the fault-tolerant and distributed SDN controller with the Byzantine fault-tolerant state-machine replication component.

5.3.1 Integration of SDN control with BFT SMR

The SUPERCLOUD network hypervisor has resilience built-in by design into the control plane (the SDN controller). The goal is to ensure that correct operation can be maintained under the most relevant failure scenarios. The design of scalable and fault-tolerant SDN controllers traditionally gives up strong consistency for the network state, adopting instead the more efficient eventually consistent storage model, for performance reasons. This lack of consistency can lead to network anomalies that may result in security breaches, an undesirable consequence, particularly for the SUPERCLOUD context.

In the project, we are developing fault-tolerant and distributed SDN control planes to address this issue. The solutions proposed are supported by a fault-tolerant data store that provides the necessary strong consistency properties. We use replicated state machines to build the data store. This technique considers a set of replicas implementing the data store being accessed through a total order multicast protocol that ensures all replicas process the same sequence of requests. The core of the total order multicast protocol we use is the consensus algorithm BFT SMR. In order to deal with the fundamental concern of such design, we apply several techniques, tailored to SDN, for optimizing the data store performance.

These solutions were described in detail in Deliverable D4.2 [17]. Further enhancements that are currently work in progress will appear in Deliverable D4.4.

5.4 Integration with security self-management

As threats on the cloud services and infrastructures are varied and constantly changing, it appears necessary to provide an autonomous security management system. Self-management of security is enabled through a combination of interacting components, as described in this Section. At the core of this system sits the network security module which features a context-aware dynamic security policy decision point which reacts on both the security alerts and the network status changes, and enforces security policies upon the network infrastructure. In order to realize the former, the network security module needs to integrate with the security monitoring component, while to perform the latter, it interfaces with the network hypervisor.

5.4.1 Integration of network security with security monitoring

As described in Deliverable D4.3 [16], the security monitoring component collects information from the network infrastructure and outputs alerts in the IDMEF format. Such format is parsable by the network security module, which exposes a REST API to process the alerts. The alerts enable the network security module to define a context on which to fine-tune the security policy decision.
Further results will be reported in Deliverable D4.4.

5.4.2 Integration of network hypervisor with network security

As described in Deliverable D4.3 [16], the network hypervisor exposes a REST API that allows the network security module to enforce the decided security policy. Upon deciding on the appropriate security policy to enforce, the network security module outputs a high-level policy action and some parameters concerning the suspected flow that the network hypervisor will use to instantiate the low-level instructions that will enforce the reaction.
Further results will be reported in Deliverable D4.4.

Chapter 6 Installation

The code of components of the SUPERCLOUD architecture can be found in a central code repository at `https://github.com/H2020-SUPERCLOUD`. Before the code repository is made publicly available, please contact `marc.lacoste@orange.com` for obtaining access to the code.
For source code of used open-source software integrated in the SUPERCLOUD architecture implementation, we provide links to the respective code repositories from which the required software can be downloaded.

6.1 Computing framework components

Instructions to obtain and install components of the computing framework are provided in Deliverable D2.3 [9]. The code of most components can be obtained from the SUPERCLOUD repository (WP2 subtree)[1] or from open source repositories. Detailed instructions for installation and further documentation about the software are distributed together with the code release. We provide simply here some component-specific information.

6.1.1 Virtualization and orchestration and micro-hypervisor

These components can be obtained from the SUPERCLOUD repository[2], with sub-folders for virtualization and orchestration (ORBITS and MANTUS frameworks) and for the micro-hypervisor.
ORBITS and MANTUS are built for OpenStack environments, with also support for Amazon Web Services. They notably rely on the `tosca-parser` library[3] to manipulate TOSCA ServiceTemplates. The micro-hypervisor component extends the Genode OS framework[4]. Instructions are provided to build and run U-Cloud nodes with cross-layer system support.

6.1.2 Isolation

The isolation framework requires the Intel SGX SDK, which can be obtained on the project's website[5], and common build tools. The code for the framework can be obtained on the SUPERCLOUD repository[6]. The script `build.sh` downloads all other required dependencies, builds the framework, and starts an interactive session.

6.1.3 Security Orchestrator

The security service deployment component may be downloaded from the SUPERCLOUD repository [7]. Execution of `SSLA2Config.jar` and `Deploy.jar` allow to configure the Orchestrator and to deploy security services locally.

[1] `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP2/`
[2] `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP2/virtualization/virtualization`
[3] `https://github.com/openstack/tosca-parser`
[4] The Genode toolchain and code are available at `https://github.com/genodelabs/genode.git` and `https://sourceforge.net/projects/genode/files/genode-toolchain/15.05/genode-toolchain-15.05-x86_64.tar.bz2`.
[5] `https://01.org/intel-software-guard-extensions/downloads/intel-sgx-linux-1.5-beta-release-0`
[6] `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/WP2-isolation`
[7] `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP2/selfmanagement/sorchestrator`

6.1.4 Authorization

The authorization component is provided as a deploy-ready Docker image that may be downloaded from the SUPERCLOUD repository [8]. Script `./start.sh` starts the component.

6.1.5 Monitoring

The monitoring component is accessible at `https://github.com/Orange-OpenSource/vespa-core`, with the corresponding documentation[9]. A tutorial is also provided to set up a first monitoring environment.

6.1.6 Security SLA

The SSLA enforcement component is provided as a deploy-ready Docker image that may be downloaded from the SUPERCLOUD repository [10]. Script `./start.sh` starts the component.

6.1.7 Software trust

The Software Trust Service is available as a deploy-ready Docker image that may be downloaded from the SUPERCLOUD repository [11]. Script `./startSTS.sh` starts the service, accessible through its REST API.

6.2 Data protection framework components

6.2.1 Janus

Code/component access. In this Section, we describe how JANUS can be installed as a NFS server inside a Docker container. The server is implemented using Java, so it can be deployed in any operating system that supports Docker. More specifically, we make available a deploy-ready Janus NFS Docker image, that can be easily integrated with other SUPERCLOUD components, in particular with the network virtualization element being developed in WP4. The commands below are for Windows deployment, but they are almost the same for the other OSs. For instance, for a Linux-based OS, `.sh` shell scripts should be used. We describe instead the `.bat` commands. Moreover, the commands may need to be run with sudo capabilities.
In the following, we present a short overview of the steps required for running the system. Nevertheless, before going into the steps, the user must first request a JANUS account by sending an email to `anbessani@ciencias.ulisboa.pt`.

1. Login into the JANUS platform at `http://janus.lasige.di.fc.ul.pt`;
2. Download the Janus NFS Docker image on the "Download" tab on the left side;
3. Uncompress `janus-nfs.zip`;
4. Open the terminal, and go to the uncompressed `janus-nfs` folder;
5. Load the Janus NFS image by running `docker load -i janus-nfs.docker`;
6. Run the container by running `run.bat`;[12]

[8] `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP2/selfmanagement/sservices/authorization`
[9] `http://vespa-core.readthedocs.io/en/latest/`
[10] `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP2/selfmanagement/sservices/sla`
[11] `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP2/selfmanagement/sservices/trust/`
[12] This container will have a default volume configured to spread the data across the globe and in different storage providers.

7. Wait until the output of running status.bat be “RUNNING”;
8. (On Windows) Install the “Services for NFS” package, which is part of “Windows Features”;
9. Mount the Janus NFS share by running `mount-janus-nfs.bat`;
10. Open “Explorer”, go to “This PC” and there one will find a “Janus” network location which is the Janus NFS share.[13]

Note. All the steps and scripts provided were tested only in Ubuntu 16.04 and Windows 10 (Build 14393). We are working on the Docker container for MacOS 10.12.4. However, the Janus NFS server (undockerized) works in all of these platforms.
A more complete description of the system can be found in Chapter 10 of SUPERCLOUD D3.2 [25].

6.2.2 Hyperledger Fabric

As Hyperledger Fabric installation instructions are not stable yet, please refer to the URL `https://hyperledger-fabric.readthedocs.io/en/latest/getting_started.html` for the latest instructions.
The Hyperledger Fabric v1 code is accessible at: `https://github.com/hyperledger/fabric/`. The integration between Hyperledger Fabric v1 and BFT-SMaRt is still under work[14]. Meanwhile, the code for BFT-SMaRt is available on `http://bft-smart.github.io/library/`.
The documentation is available at: `https://hyperledger-fabric.readthedocs.io/en/latest/`.

6.3 Network security framework components

6.3.1 Multi-cloud network orchestrator and network hypervisor

Detailed instructions to install and run the Sirius components can be found in the `README` file at `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP4/sirius`. Hereafter, we will only deal with some of the main requirements and steps to run the code. We will discuss the three parts of the Sirius architecture:

- The **VMs** are deployed in private or public clouds and run the *Docker* containers mapped from the virtual infrastructure.
- The **Hypervisor** is responsible for maintaining the substrate topology information and performing the embedding process to create the virtual topology.
- The **Orchestrator** allows to design the substrate topology and deploy the required containers through a web-based graphical interface.

6.3.1.1 VM infrastructure

The first version of the Orchestrator has been designed to interact with Linux VMs. As the Docker platform also supports Windows systems, Windows-based VMs could be used in theory but will require modifications in the current code.

[13] Note that it is impossible to write to the root folder of the Janus NFS folders. It is only possible to work in volumes' folders.

[14] Recall that HLF is not only a SUPERCLOUD component, but a large and complex project with many stakeholders. Therefore, adding a new component in its codebase is a complex process which requires the development of several performance and integration tests.

6.3.1.1.1 SSH connection

The SSH protocol is used to connect the VMs and execute remote commands. Default port 22 TCP should be open but other ports can be configured. SSH sessions on VMs are established based on public/private key authentication. Key files must be available in the `"sirius/keys"` folder as described in the `README` file.

6.3.1.1.2 Inter-cloud tunnels

Gateway VMs from different clouds are connected through OpenVPN tunnels. Port 1194 UDP must be open on those machines. As the Orchestrator does not handle yet the creation of new gateway VMs, OpenVPN tunnels have to be set up and running before the Orchestrator starts.

Note that the Orchestration can be run with a set of local VMs configured in a single cloud. In this case, no OpenVPN tunnels are required.

6.3.1.1.3 Intra-cloud tunnels

According the substrate topology, *GRE* tunnels are automatically created by the Orchestration between VMs located in the same cloud. No predefined configuration is required.

6.3.1.1.4 Docker installation

The Docker platform as well as the OpenVSwitch components have the installed in all VMs. The `‘‘installDocker’’` script from `‘‘sirius/script’’` can be used to automatically install the required libraries in *Ubuntu* environments.

6.3.1.2 Orchestrator

The Orchestrator runs on Windows or Linux platforms and only requires a HTTP/Servlet server. Instructions to compile and install the code are detailed in the `README` file. Assuming that the Apache Tomcat server is installed on Linux, the script `‘‘/opt/tomcat/startup.sh’’` starts the server.

6.3.1.3 Hypervisor

The Hypervisor is available in the form of a JAR file to be downloaded and installed from the GIT repository. The Hypervisor uses the same configuration file (`‘‘console.properties’’`) as the Orchestration. Both components communicate through a TCP socket and that way can operate in different machines. IP addresses and TCP ports can be set up in the properties file. The script `‘‘sirius/script/floodlight.sh’’` starts the Hypervisor which relies on the Floodlight SDN controller.

6.3.2 Network security monitoring and appliance chaining

The network security monitoring and appliance chaining components are being put in open source. The components will then be accessible at `https://github.com/Orange-OpenSource`, also being published on the SUPERCLOUD private repository[15]. More detailed instructions are provided to build and run a Floodlight SDN controller enhanced with security monitoring features, or extended to run the appliance chaining application.

[15] `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP4/monitoring` and `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP4/service-chaining`

6.3.3 Network security policy management

The network security module may be downloaded from the SUPERCLOUD repository [16]. Instructions for installation are detailed in the `README` file. The module runs as a Python program, so a Python runtime environment (version 2.6 or later) is all that is necessary to execute it. It is also recommended to allocate a dedicated storage space to host network security policies.

6.4 Security self-management components

Components developed as part of the security self-management are made available as ready-to-deploy Docker images. The detailed description of each components as well as the installation and deployment procedure are provided in Deliverable D1.4 [26]. The images of the components can be found in the SUPERCLOUD repository [17].
In what follows we present two approaches for the deployment of the security self-management components, a manual one in Section 6.4.1 and an automatic one in Section 6.4.2.

6.4.1 Manual deployment

The procedure to follow in order to deploy each component individually is as follows:

- Download the `.tar` file containing the component binaries;
- Download the `launch.sh` script containing commands for the deployment of the image;
- Place the `.tar` file and the `.sh` file in the same repository;
- Execute the `.sh` script.

6.4.2 Orchestrated deployment

In the orchestrated deployment, security self-management components are deployed by the Orchestrator in an automatic way. The deployment schema relies on Docker-Compose [6]. To proceed, all is needed is to run the `start.sh` script. The Orchestrator will take care of downloading the the components from the project repository and deploy them individually. The Orchestrator will also take care of the creation of a virtual network to bind the components between them.

[16] `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP4/network-security`
[17] `https://github.com/H2020-SUPERCLOUD/SUPERCLOUD-FW/tree/master/WP1/`

Chapter 7 Summary

In this Deliverable, we have provided an overview of the implementation of the SUPERCLOUD architecture. It is composed of a number of software components organized into frameworks for each of the three layers of this architecture, respectively providing secure computing, data storage and networking functionalities that can be deployed across infrastructures of different cloud service providers. Each framework is supported by an orthogonal security self-management infrastructure providing facilities for user-level control of security and privacy settings. We have provided descriptions of the implemented architectural components and described their integration across the different frameworks. The code of the released software components is accessible on a common SUPERCLOUD code repository at `https://github.com/H2020-SUPERCLOUD`. The installation instructions for these components have also been provided in this Deliverable.

Chapter 8 List of Abbreviations

ABE	Attribute-Based Encryption
API	Application Programming Interface
BFT	Byzantine Fault Tolerance
CE	Convergent Encryption
CPU	Central Processing Unit
DB	Database
EC	European Commission
FPGA	Field-Programmable Gate Array
GDPR	General Data Protection Regulation
GRE	Generic Routing Encapsulation
HLF	Hyperledger Fabric
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
KV	Key-Value
NFS	Network File System
OLA	Optimal Lattice Anonymization
ORAM	Oblivious Random Access Memory
OrBAC	Organization Based Access Control
ORBITS	ORchestration for Beyond InTer-cloud Security
OS	Operating System
OVS	Open vSwitch
PC	Personal Computer
REST	Representational State Transfer
SBFT	Simplified Byzantine Fault Tolerance
SDK	Software Development Kit
SDN	Software-Defined Networking
SGX	Software Guard eXtensions
SLA	Service Level Agreement
SMR	State Machine Replication
SSH	Secure Shell
SSLA	Security Service Level Agreement

TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOSCA	Topology and Orchestration Specification for Cloud Applications
U-Cloud	User Cloud
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VM	Virtual Machine
VNE	Virtual Network Environment

Bibliography

[1] M. Alaluna, F. Ramos, and N. Neves. (Literally) Above the Clouds: Virtualizing the Network over Multiple Clouds. In *IEEE NetSoft*, 2016.

[2] Max Alaluna, Eric Vial, Nuno Neves, and Fernando Ramos. Secure and Dependable Multi-Cloud Network Virtualization. In *EuroSys 1st International Workshop on Security and Dependability of Multi-Domain Infrastructures (XDOM0)*, 2017.

[3] Alysson Bessani, Mario Münzer, Sébastien Canard, Nicolas Desmoulins, Marie Paindavoine, Marko Vukolić, and Daniel Pletea. D3.3 - Proof-of-Concept Prototype for Data Management. *SUPERCLOUD*, 2017.

[4] Alysson Bessani, Joao Sousa, and Eduardo Alchieri. State machine replication for the masses with BFT-SMaRt. In *Proc. of the IEEE/IFIP International Conference on Dependable Systems and Networks – DSN 2014*, June 2014.

[5] Christian Cachin, Simon Schubert, and Marko Vukolić. Non-determinism in byzantine fault-tolerant replication. In *20th International Conference on Principles of Distributed Systems, OPODIS 2016, December 13-16, 2016, Madrid, Spain*, pages 24:1–24:16, 2016.

[6] Docker Compose. Docker compose tool. 2016.

[7] O. Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 182–194, New York, NY, USA, 1987. ACM.

[8] Marc Lacoste, Markus Miettinen, Nuno Neves, Fernando M. V. Ramos, Marko Vukolic, Fabien Charmet, Reda Yaich, Krzysztof Oborzynski, Gitesh Vernekar, and Paulo Sousa. User-Centric Security and Dependability in the Clouds-of-Clouds. *IEEE Cloud Computing*, 3(5):64–75, 2016.

[9] Marc Lacoste, Mario Münzer, Felix Stornig, Alex Palesandro, Denis Bourge, Charles Henrotte, Houssem Kanzari, Marko Vukolić, Jagath Weerasinghe, Reda Yaich, Nora Cuppens, Frédéric Cuppens, Markus Miettinen, Ferdinand Brasser, Tommaso Frassetto, and Daniel Pletea. D2.3 - Proof-of-Concept Prototype of Secure Computation Infrastructure and SUPERCLOUD Security Services. *SUPERCLOUD*, 2017.

[10] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolić. XFT: practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 485–500, 2016.

[11] Markus Miettinen, Ferdinand Brasser, Ahmad-Reza Sadeghi, Marc Lacoste, Nizar Kheir, Marko Vukolić, Alysson Bessani, Fernando Ramos, Nuno Neves, and Majid Sobhani. D1.1 - SUPERCLOUD Architecture Specification. *SUPERCLOUD*, 2015.

[12] Markus Miettinen, Mario Münzer, Felix Stornig, Marc Lacoste, Alex Palesandro, Denis Bourge, Charles Henrotte, Houssem Kanzari, Ruan He, Marko Vukolić, Jagath Weerasinghe, Sabir Idrees, Reda Yaich, Nora Cuppens, Frédéric Cuppens, Ferdinand Brasser, 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.

[13] Andre Nogueira, Antonio Casimiro, and Alysson Bessani. Elastic state machine replication. *IEEE Transactions on Parallel and Distributed Systems*, March 2017. Accepted for publication.

[14] Alex Palesandro, Chirine Ghedira Guegan, Marc Lacoste, and Nadia Bennani. Overcoming barriers for ubiquitous user-centric healthcare services. *IEEE Cloud Computing*, 3(6):64–74, 2016.

[15] 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.

[16] 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.

[17] Fernando M. V. Ramos, Nuno Neves, Marc Lacoste, Nizar Kheir, Max Alaluna, André Mantas, Luis Ferrolho, José Soares, Grégory Blanc, Fabien Charmet, and Khalifa Toumi. D4.2 - Specification of Self-Management of Network Security and Resilience. *SUPERCLOUD*, 2016.

[18] Joao Sousa and Alysson Bessani. Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines. In *Proc. of the 34th International Symposium on Reliable Distributed Systems – SRDS'15*, September 2015.

[19] Paulo Sousa, Gregory Blanc, Krzysztof Oborzyński, Bruno Ferreira, and Joana Cruz. D5.2 - Use-Case Demonstrators. *SUPERCLOUD*, 2017.

[20] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. In *Proceedings ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 299–310, New York, NY, USA, 2013. ACM.

[21] K. Toumi, M. S. Idrees, F. Charmet, R. Yaich, and G. Blanc. Usage control policy enforcement in sdn-based clouds: A dynamic availability service use case. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 578–585, Dec 2016.

[22] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*, pages 112–125, 2015.

[23] Marko Vukolić. Rethinking permissioned blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, BCC '17, pages 3–7, 2017.

[24] Marko Vukolić, Mario Münzer, Sébastian Canard, Marie Paindavoine, Alysson Bessani, Caroline Fontaine, Krzysztof Oborzyński, Meilof Veeningen, and Paulo Sousa. D3.1 - Architecture for Data Management. *SUPERCLOUD*, 2015.

[25] Marko Vukolić, Mario Münzer, Sébastian 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.

[26] Reda Yaich, Nora Cuppens, Frédéric Cuppens, Marc Lacoste, Sébastien Canard, Alysson Bessani, Fernando Ramos, Nuno Neves, Markus Miettinen, Krzysztof Oborzyński, Daniel Pletea, and Marko Vukolić. D1.4 - SUPERCLOUD Self-Management of Security Implementation. *SUPERCLOUD*, 2017.

[27] Reda Yaich, Sabir Idrees, Nora Cuppens, Frédéric Cuppens, Marc Lacoste, Nizar Kheir, Ruan He, Khalifa Toumi, Krzysztof Oborzyński, Meilof Veeningen, and Paulo Sousa. D1.2 - SUPERCLOUD Self-Management of Security Specification. *SUPERCLOUD*, 2015.