

D2.4.3

Final Reference Platform and Test Case Specification

Project number:	257243
Project acronym:	TClouds
Project title:	Trustworthy Clouds - Privacy and Resilience for Internet-scale Critical Infrastructure
Start date of the project:	1 st October, 2010
Duration:	36 months
Programme:	FP7 IP

Deliverable type:	Prototype
Deliverable reference number:	ICT-257243 / D2.4.3 / 1.0
Activity and Work package contributing to deliverable:	Activity 2 / WP 2.4
Due date:	September 2013 – M36
Actual submission date:	7 th October, 2013

Responsible organisation:	POL
Editor:	Gianluca Ramunno
Dissemination level:	Public
Revision:	1.0

Abstract:	This deliverable reports the concept, architecture and testing of the final TClouds Platform as a collaborative work of the other Activity 2 workpackages.
Keywords:	platform, subsystems, prototypes, modules, trustworthy infrastructure, testing, large-scale, high-security

Editor

Gianluca Ramunno (POL)

Contributors

Tony Cutillo, Gianluca Ramunno, Roberto Sassu, Paolo Smiraglia (POL)

Alexander Buerger, Norbert Schirmer (SRX)

Alysson Bessani, Marcel Henrique dos Santos (FFCUL)

Sören Bleikertz, Zoltan Nagy (IBM)

Imad M. Abbadi, Anbang Ruad (OXFD)

Johannes Behl, Klaus Stengel (TUBS)

Mihai Bucicoiu, Sven Bugiel, Hugo Hideler, Stefan Nürnberger (TUDA)

Disclaimer

This work was partially supported by the European Commission through the FP7-ICT program under project TClouds, number 257243.

The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose.

The user thereof uses the information at its sole risk and liability. The opinions expressed in this deliverable are those of the authors. They do not necessarily represent the views of all TClouds partners.

Executive Summary

Cloud computing is an emerging technology devoted to outsource IT infrastructures, from SME needs to large-scale computing and storage. However, organizations and industries that make use of critical infrastructure are cautious to move towards cloud infrastructures since they still experience security and privacy breaches.

The TClouds project aims at facilitating the shift of computing paradigm for critical infrastructures by increasing the robustness of Infrastructure as a Service (IaaS) cloud platforms through subsystems that can be combined and used in different scenarios: private or public clouds, commodity or native TClouds clouds, or mixed scenarios. TClouds also provides building blocks as part of a secure Platform as a Service (PaaS) for application specific needs.

This deliverable is a compendium of the work done in workpackages 2.1, 2.2 and 2.3. The subsystems conceived, designed, and developed in those workpackages, have been framed into the TClouds Platform v2.1, a comprehensive secure cloud computing ecosystem comprising two different prototypes (i.e. complex groups of integrated subsystems) at the IaaS layer and several modules (i.e. single subsystems and simple aggregations of some of them) at the PaaS layer. With these two alternative infrastructure prototypes we can cover the needs of a wide range of application scenarios from private or community clouds with high security demands to large-scale public clouds. With the appropriate selection of PaaS modules, the applications can satisfy security requirements that cannot be managed at the IaaS and that have to be dealt with by the applications themselves.

The TClouds Platform described in this deliverable has the objective to satisfy the requirements set by European and national laws on data protection (WP1.1) and by two benchmark application scenarios, health-care (WP3.1) and energy related (WP3.2) applications. Such requirements and how the TClouds subsystems satisfy them are reported in the previous deliverable D2.4.2 [S⁺12a]. The successful validation of the TClouds Platform against such requirements, on a per-subsystem basis and from Activity 3 applications perspective, is reported in the deliverables D3.3.3 [Abi13] and D3.3.4 [Abi13].

This deliverable is organized in three parts. Part I describes the concept, the architecture and the instantiations of the TClouds Platform as a result of the evolution that took place throughout the project. Part II includes the documentation related to the testing of the TClouds Platform components. Finally, Part III reports the evolution of the TClouds platform concept and of the subsystems throughout the project and focuses on the programming, installation and configuration documentation for the released software APIs and to the code availability of each subsystem.

Contents

1	Introduction	1
1.1	TClouds — Trustworthy Clouds	1
1.2	Activity 2 — Trustworthy Internet-scale Computing Platform	1
1.3	Workpackage 2.4 — Architecture and Integrated Platform	2
1.4	Deliverable 2.4.3 — Final Reference Platform and Test Case Specification	4
I	TClouds Platform v2	7
2	The TClouds Platform	
	Concept, Architecture and Instantiations	8
2.1	Introduction	8
2.2	Concept of platform	9
2.3	Amazon AWS architecture	10
2.3.1	Example application: theneeds	11
2.4	The TClouds Platform Architecture	11
2.4.1	Infrastructure	12
2.4.2	Middleware	14
2.4.3	Services	15
2.5	Platform Instantiations	16
II	Testing	18
3	Final test plans for subsystems/prototypes	19
3.1	Introduction	19
3.2	TrustedInfrastructure Cloud	19
3.2.1	Test methodology/strategy	19
3.2.2	Test cases	19
3.3	Security Assurance of Virtualized Environments (<i>SAVE</i>)	24
3.3.1	Test methodology/strategy	24
3.3.2	Test cases	24
3.4	Tailored memcached service	26
3.4.1	Short subsystem intro	26
3.4.2	Test methodology/strategy	26
3.4.3	Test cases	26
3.5	Fault-tolerant Workflow Execution (FT-BPEL)	28
3.5.1	Test methodology/strategy	28
3.5.2	Test cases	29
3.6	Cryptography as a Service	33
3.6.1	Test methodology/strategy	33

3.6.2	Test cases	33
3.7	Access Control as a Service (ACaaS)	36
3.7.1	Test methodology/strategy	36
3.7.2	Test cases	36
3.8	BFT-SMaRt	38
3.8.1	Test methodology/strategy	38
3.8.2	Test cases	38
3.8.3	Demos	39
3.9	Resilient Object Storage (DepSky)	40
3.9.1	Test methodology/strategy	40
3.9.2	Test cases	40
3.10	LogService	42
3.10.1	Test methodology/strategy	42
3.10.2	Test execution	42
3.11	Remote Attestation Service	44
3.11.1	Test methodology/strategy	44
3.11.2	Test cases	45
3.12	Ontology-based Reasoner-Enforcer	50
3.12.1	Test methodology/strategy	50
3.12.2	Test cases	51
4	Test results	53
4.1	Trustworthy OpenStack Prototype	53
4.1.1	LogService	54
4.1.2	Remote Attestation Service	56
4.1.3	Cryptography as a Service	57
4.1.4	ACaaS	60
4.1.5	Ontology-based Reasoner/Enforcer	63
4.2	TrustedInfrastructure Cloud Prototype	63
4.3	BFT-SMaRt	63
4.4	Resilient Object Storage (DepSky)	64
4.5	Tailored Memcached	64
4.5.1	Service deployment	64
4.5.2	Test tailoring	65
4.6	Fault-Tolerant BPEL	65
4.6.1	Fault-free operation on standard infrastructure	65
4.6.2	Fault-free operation on FT-BPEL infrastructure	66
4.6.3	Crashed system present on standard infrastructure	66
4.6.4	Crashed system present on FT-BPEL infrastructure	67
4.7	SAVE Subsystem	68
4.7.1	Discovery	68
4.7.2	Analysis Unit Testing	68
4.7.3	Analysis System Testing	69
III	Appendices	70
A	Software details of the prototypes	71

B Subsystems' code availability	72
C Evolution of TClouds platform and integration of subsystems	73
D Low-level APIs	75
Bibliography	75

List of Figures

1.1	Graphical structure of WP2.4 and relations to other work packages.	3
2.1	Architectural and logical cloud stacks.	10
2.2	Amazon Web Services platform.	10
2.3	The TClouds platform.	12
2.4	Home healthcare scenario.	16
2.5	Smart lighting scenario.	17
3.1	libseclog dependencies installation	42
3.2	libseclog building commands	42
4.1	Nova tests results.	54
4.2	Python-novaclient tests results.	55
4.3	Quantum tests results.	55
4.4	Horizon tests results.	56
4.5	Successful JUnit Test Run.	68
4.6	Successful Groove Analysis Unit Test Run.	69

List of Tables

2.1	TClouds subsystems and classification. Some services are based on middleware ones [in brackets].	13
2.2	TClouds services and required resources.	15
B.1	List of TClouds subsystems and code availability	72
C.1	List of TClouds subsystems and their evolution in the frame of TClouds Platform	74

Chapter 1

Introduction

Chapter Author: Gianluca Ramunno (POL)

1.1 TClouds — Trustworthy Clouds

TClouds aims to develop *trustworthy* Internet-scale cloud services, providing computing, network, and storage resources over the Internet. Existing cloud computing services are today generally not trusted for running *critical infrastructure*, which may range from business-critical tasks of large companies to mission-critical tasks for the society as a whole. The latter includes water, electricity, fuel, and food supply chains. TClouds focuses on power grids and electricity management and on patient-centric health-care systems as its main applications.

The TClouds project identifies and addresses legal implications and business opportunities of using infrastructure clouds, assesses security, privacy, and resilience aspects of cloud computing and contributes to building a regulatory framework enabling resilient and privacy-enhanced cloud infrastructure.

The main body of work in TClouds defines an architecture and prototype systems for securing infrastructure clouds, by providing security enhancements that can be deployed on top of commodity infrastructure clouds (as a cloud-of-clouds) and by assessing the resilience, privacy, and security extensions of existing clouds.

Furthermore, TClouds provides resilient middleware for adaptive security using a cloud-of-clouds, which is not dependent on any single cloud provider. This feature of the TClouds platform will provide tolerance and adaptability to mitigate security incidents and unstable operating conditions for a range of applications running on a clouds-of-clouds.

1.2 Activity 2 — Trustworthy Internet-scale Computing Platform

Activity 2 carries out research and builds the actual TClouds platform, which delivers trustworthy resilient cloud-computing services. The TClouds platform contains trustworthy cloud components that operate inside the infrastructure of a cloud provider; this goal is specifically addressed by WP2.1. The purpose of the components developed for the infrastructure is to achieve higher security and better resilience than current cloud computing services may provide.

The TClouds platform also links cloud services from multiple providers together, specifically in WP2.2, in order to realize a comprehensive service that is more resilient and gains higher security than what can ever be achieved by consuming the service of an individual cloud

provider alone. The approach involves simultaneous access to resources of multiple commodity clouds, introduction of resilient cloud service mediators that act as added-value cloud providers, and client-side strategies to construct a resilient service from such a cloud-of-clouds.

WP2.3 introduces the definition of languages and models for the formalization of user- and application-level security requirements, involves the development of management operations for security-critical components, such as “trust anchors” based on trusted computing technology (e.g., TPM hardware), and it exploits automated analysis of deployed cloud infrastructures with respect to high-level security requirements.

Furthermore, Activity 2 will provide an integrated prototype implementation of the trustworthy cloud architecture that forms the basis for the application scenarios of Activity 3. Formulation and development of an integrated platform is the subject of WP2.4.

These generic objectives of A2 can be broken down to technical requirements and designs for trustworthy cloud-computing components (e.g., virtual machines, storage components, network services) and to novel security and resilience mechanisms and protocols, which realize trustworthy and privacy-aware cloud-of-clouds services. They are described in the deliverables of WP2.1–WP2.3, and WP2.4 describes the implementation of an integrated platform.

1.3 Workpackage 2.4 — Architecture and Integrated Platform

The objective of WP2.4 is the design of an overall architecture framework that serves as a basis for the combination of the research results and prototypes of work packages WP2.1, WP2.2 and WP2.3 in order to build an integrated proof of concept prototype of a resilient cloud-of-clouds infrastructure. Based on the cloud applications (WP3.1, WP3.2), and the related technical requirements (WP1.1), the resulting TClouds platform architecture and the required subsystems are defined, implemented by the corresponding work packages, and finally integrated into the proof of concept prototype of a trustworthy cloud environment, which is the major outcome of this work package.

The workpackage is split into four tasks.

- Task 2.4.1 (M01-M08): Use Case Analysis
- Task 2.4.2 (M01-M28): Architecture including public interfaces
- Task 2.4.4 (M07-M36): Initial component Integration and final Integrated Platform
- Task 2.4.5 (M07-M36): Test Methodology and Tests Cases

Task 2.4.1 took place in the first year and was devoted to select and analyze the use cases to be implemented by each subsystem, starting from the requirements formulated within Activity 1 and Activity 3 work packages. Task 2.4.2 is concerned to define an overall architecture and the interfaces; these activities were started during the first year but continued during the second year and will take part of the third year. Task 2.4.4 refers to the integration of the various subsystems into a platform; it started during the first year and will end at the end of the project. The outcome of this task for the second year (initial component integration) is the main input for the present deliverable. Task 2.4.5 is focused on defining the test methodology and the test cases and on actually performing the tests on the developed subsystems. Also this task spans from the first year to the end of the project.

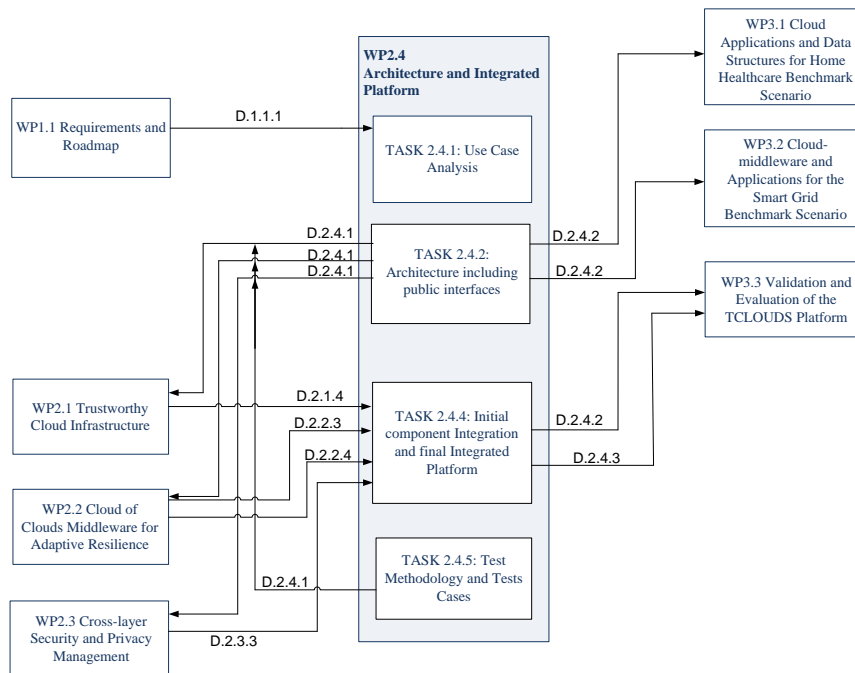


Figure 1.1: Graphical structure of WP2.4 and relations to other work packages.

During the second year the focus was the initial integration of the subsystems developed in WPs 2.1-2.3 in terms of both connecting the subsystems to cooperate and having an integrated development and testing process.

During the third year, the initial concept of the TClouds platform has been evolved into a comprehensive ecosystem, which includes the original subsystems, arranged in logical service layers that form the TClouds platform Version v2. Furthermore, the test plans for all subsystems have been refined, their final test results have been obtained, and eventually the TClouds platform has been provided to the two benchmark applications. For this reason the activities within this work package have been carried out on two parallel tracks: refining the integration among groups of subsystems to form the infrastructures at IaaS and evolving the subsystems, where necessary, to fully support the benchmark scenarios. These activities involved a close collaboration of all partners. The results of the final TClouds platform v2 concept and implementation together with the test plans and results are collected in the deliverable D2.4.3.

Figure 1.1 illustrates WP2.4 and its relations to other work packages according to the DoW/Annex I.

Requirements were collected from WP1 to define the use cases in Task 2.4.1. The architecture and the interfaces defined in Task 2.4.2 are reported back to and used by WPs 2.1-2.3 to develop their subsystems that become then the input for Task 2.4.4. The outcome of Task 2.4.2 is employed by WPs 3.1 and 3.2 to design and develop their applications. The output of Task 2.4.4 is the input for WP3.3 to perform the evaluation of the TClouds platform.

1.4 Deliverable 2.4.3 — Final Reference Platform and Test Case Specification

Overview. Cloud computing is an emerging technology devoted to outsource IT infrastructures, from SME needs to large-scale computing and storage. However, organizations and industries that make use of critical infrastructure are cautious to move towards cloud infrastructures since they still experience security and privacy breaches.

The TClouds project aims at facilitating the shift of computing paradigm for critical infrastructures by increasing the robustness of Infrastructure as a Service (IaaS) cloud platforms through subsystems that can be combined and used in different scenarios: private or public clouds, commodity or native TClouds clouds, or mixed scenarios. TClouds also provides building blocks as part of a secure Platform as a Service (PaaS) for application specific needs.

This deliverable is a compendium of the work done in workpackages 2.1, 2.2 and 2.3. The subsystems conceived, designed, and developed in those workpackages, have been framed into the TClouds Platform v2.1, a comprehensive secure cloud computing ecosystem comprising two different prototypes (i.e. complex groups of integrated subsystems) at the IaaS layer and several modules (i.e. single subsystems and simple aggregations of some of them) at the PaaS layer. With these two alternative infrastructure prototypes we can cover the needs of a wide range of application scenarios from private or community clouds with high security demands to large-scale public clouds. With the appropriate selection of PaaS modules, the applications can satisfy security requirements that cannot be managed at the IaaS and that have to be dealt with by the applications themselves.

The TClouds Platform described in this deliverable has the objective to satisfy the requirements set by European and national laws on data protection (WP1.1) and by two benchmark application scenarios, health-care (WP3.1) and energy related (WP3.2) applications. Such requirements and how the TClouds subsystems satisfy them are reported in the previous deliverable D2.4.2 [S⁺12a]. The successful validation of the TClouds Platform against such requirements, on a per-subsystem basis and from Activity 3 applications perspective, is reported in the deliverables D3.3.3 [Abi13] and D3.3.4 [Abi13].

Deviation from Workplan. This deliverable follows the workplan in the DoW/Annex I v4.

Target Audience. This deliverable aims at researchers and developers of security and management systems for cloud-computing platforms. The deliverable assumes graduate-level background knowledge in computer science technology, specifically, in virtual-machine technology, operating system concepts, security policy and models, and formal languages.

Relation to Other Deliverables. The workpackages and especially the year 3 deliverables of Activity 2 are closely related with each other, reflecting the integration efforts of Activity 2. Roughly speaking WP 2.1 provides resilience, privacy and security to individual infrastructure clouds (IaaS). WP 2.2 provides resilient middleware offering both infrastructure (IaaS) as well as platform (PaaS) services, following the cloud-of-cloud paradigm. WP 2.3 deals with the security management and finally in WP 2.4 all is integrated to the final TClouds platform. So to get the complete picture, the reader has to consider all deliverables of the different workpackages.

To help the reader to gain a clean view of the Activity 2 outcomes for the third and final year of the project, we provide here an overall view of the delivered TClouds platform and

how to map it to the actual Activity 2 deliverables (or their parts or chapters) released during the third year. This view can be understood as the logical outline of all deliverables which is broken down to the content presented in the different deliverables. The following logical view starts from high level “big picture” (i.e. the latest concept of the TClouds platform), and moves down towards an in-depth and more concrete level, covering research and technical details of the integration of subsystems, updated research and technical details of single subsystems, and the actually delivered software with instructions for installation and configuration.

(Part 1) TClouds platform v2.x: definition of platform, comparison with Amazon AWS ecosystem, big picture of TClouds ecosystem and summary presentation tailored platform instantiations into two Activity 3 benchmark scenarios (for further details on the benchmark scenarios, see respectively D3.1.5 [D⁺13] and D3.2.4 [VS13]-D3.2.5 [Per13])

- D2.4.3 (this document), Part I (Chapter 2)

(Part 2) Integrated prototypes: research/technical details of the integration of subsystems to form IaaS alternatives – Trustworthy OpenStack and TrustedInfrastructure Cloud – and PaaS modules/services – C2FS and SteelDB

- D2.1.5 [S⁺13a], Part I
- D2.2.4 [B⁺13a], Chapters 6 and 7
- D2.3.4 [B⁺13b], Chapters 3 and 5

(Part 3) Subsystems: research/technical details of single subsystems – only updates from previous deliverables

- D2.1.5 [S⁺13a], Part II
- D2.2.4 [B⁺13a], Chapters 2, 3, 4 and 5
- D2.3.4 [B⁺13b], Chapters 2, 4 and 6

(Part 4) Testing of prototypes and subsystems: test plans and results

- D2.4.3 (this document), Part II

(Part 5) Software details: instructions for installation, configuration and usage of prototypes (integrated subsystems)

- D2.1.4-D2.3.3 [BS⁺13]
- D2.4.3 (this document), Appendix A

(Part 6) Software delivery: source code and/or binary code of prototypes and subsystems

- TClouds platform v2.0 (only subsystems for single cloud): D2.1.4-D2.3.3 [BS⁺13], companion tarball(s)
- TClouds platform v2.1 (complete): D2.4.3 (this document), companion tarball(s)

Structure. Following the logical structure of the Activity 2 deliverables previously explained, this deliverable is organized in two main parts and appendices.

Part I describes the concept, the architecture and the instantiations of the TClouds Platform as a result of the evolution that took place throughout the project and only consists of Chapter 2.

Part II includes the documentation related to the testing of the TClouds Platform components. Chapter 3 reports the tests plans for most of the subsystems being part of the TClouds Platform. Chapter 4 reports the test results, grouped by prototypes (where applicable).

Part III contains the appendices. Appendix A points to the documentation for installation, configuration and usage of released software. Appendix B reports the source and/or object code availability for each subsystem and the location where the code can be accessed from. Appendix C reports the evolution of the TClouds platform concept and of the subsystems throughout the project. Appendix D points to the report R2.4.2.4 [S⁺13b] for the documentation of the low-level APIs.

Part I

TClouds Platform v2

Chapter 2

The TClouds Platform Concept, Architecture and Instantiations

Chapter¹ Authors: Alysson Bessani (FFCUL), Leucio A. Cutillo (POL), Gianluca Ramunno (POL), Norbert Schirmer (SRX), Paolo Smiraglia (POL)

2.1 Introduction

This chapter aims at presenting the TClouds platform: it wants to give the reader an overview of the platform, showing its architecture and the subsystems it is composed of without going into details. Research results related to the subsystems, including their effectiveness, and technical details can be found respectively in the referenced papers and deliverables. Our goal here is to provide an integrated view of the platform, summarizing what it offers to improve the security of cloud applications. The motivation for providing two IaaS frameworks and an initial comparison of the resource costs of the offered secure/resilient storage services are also given. The approach chosen during the third year, which does not require fully integrating all subsystems in a single software package, recognizes that the complexity of the cloud security problem could not be solved by a single solution, requiring thus a multitude of options and services that can be adapted to different application requirements with a tailored platform. Such approach is backed by the concept of platform here presented (which is the final result of an evolution occurred throughout the project), and is validated by two different instantiations for the project's benchmark scenarios. The security requirements that TClouds platform must satisfy, originated in Activity 3 for the benchmark application scenarios and in Activity 1 for the legal aspects, are collected in Chapter 2 of D2.4.2 [S⁺12a]: these requirements are mapped onto Activity 2 subsystems in Section 3.5 of the same deliverable. Such mapping was the starting point for the validation activities performed within Activity 3 during the third year: the validation of the TClouds platform against the legal and application requirements has been successfully performed on a per-subsystem basis: the validation procedure and results are collected in D3.3.4 [A⁺13]. As a result of the overall approach, the project contributed with several subsystems and fundamental results that both improved the understanding of the cloud security problem and provide new ways to think about it. Details on the evolution of the platform concept in TClouds are given in Appendix C.

¹The source material of this chapter, with the exception of the introductory section, appeared as a paper presented at the DISCCO'13 workshop, September 30th 2013, Braga, Portugal.

2.2 Concept of platform

Extending the original meaning of “flat form” as a place suitable to sustain, the concept of *platform* in a personal computing context indicates a hardware and/or software architecture that serves as a foundation on which application programs can run, that is, “*an alternative term for a computer system, including both the hardware and the software*”².

The term originally dealt with only hardware, and sometimes it is still used to refer a particular CPU model or computer family (e.g. x86, x64). However, most of the times by “platform” one means *an environment where a software can run*, ranging from an operating system which is, from the application software perspective, a platform by default, to application software, which can serve the purpose of platform from an ad-hoc extension perspective. Therefore, an application can run on a platform and serve in turn as a platform for other programs. Nevertheless, applications that do not provide any API cannot be considered as platforms.

The recent rise of cloud computing posed a paradigm shift for computation, allowing users to benefit from “everything-as-a-service” for the first time. As a matter of fact, globally deployed cost-efficient and scalable resources are made available on demand, allowing users to access them via lightweight devices and reliable Internet connection. According to NIST, cloud computing paradigm is composed by three service models [MG11]: Infrastructure as a Service (IaaS), allowing users to provision fundamental computing resources where the consumer is able to deploy and run arbitrary software; Platform as a Service (PaaS), allowing to deploy onto the cloud infrastructure applications created using tools supported by the provider; and Software as a Service (SaaS), allowing users to access providers applications running on a cloud infrastructure.

At a first glance, a parallelism between the personal computing and the cloud computing paradigm become evident. One could associate, respectively, the cloud computing IaaS to the personal computing hardware, the PaaS to a personal computing Operating System (OS) or software development frameworks, and finally the SaaS to a software application running in user space. However, such glance does not grab a main distinction among the two models: while in personal computing both software framework environments and user applications run compulsorily on top of the hardware and the hardware therefore acts as a platform for them, in cloud computing there’s no constrained dependency between IaaS, PaaS and SaaS, i.e. SaaS and PaaS provisioning does not depend on IaaS, therefore IaaS is not a mandatory platform for PaaS and SaaS. For instance, solutions like Owncloud³ propose PaaS provisioning without asking for any IaaS. Users may install Owncloud in their own (Web and RDBMS) server and install, in turn, a series of (SaaS) applications (e.g., agenda, calendar) on top of it.

Generally speaking, in the context of cloud computing, IaaS, PaaS, and SaaS may be selectively provided to the user and interact among one each other when required according to the user’s specific goals. As an additional example, a user aiming to provide a web service may (1) ask for a virtual machine, install his preferred OS and setup his own web server (IaaS); (2) use the cloud tools to build his service (PaaS); (3) use the closest web service already provided by the cloud (SaaS). In a logical point of view, when looking to IaaS, PaaS and SaaS from the cloud provider perspective, one may create a taxonomy of services and arrange them in a logical stack, as depicted in Figure 2.1. However, from an architectural point of view, IaaS, PaaS and SaaS are not arranged in any stack. They rather depend on either the cloud OS or the cloud middleware.

²http://www.ict4lt.org/en/en_glossary.htm

³<http://owncloud.org/>

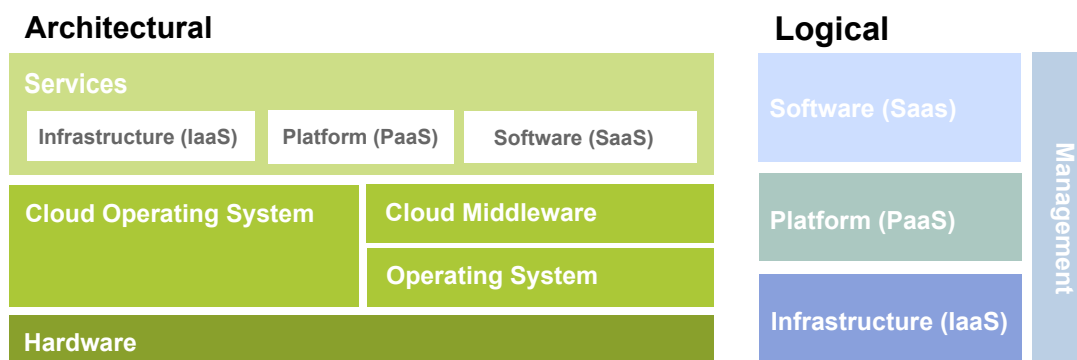


Figure 2.1: Architectural and logical cloud stacks.

2.3 Amazon AWS architecture

The aim of this section is to show that there exists commodity clouds that implement the concept of platform outlined in Section 2.2. In particular we give an overview of *Amazon Web Services* (AWS), the Amazon’s cloud platform, and we illustrate the way a real application can be built on top of it by selecting a subset of platform services/components.

Officially launched in 2006, AWS is a cloud platform providing several functionalities that allow the cloud customers to build their own cloud-based systems. Each functionality is provided *as a service* and could be adopted alone or integrated with others, in order to realize a more complex system. Figure 2.2 shows that the AWS platform is organized in layers, each one representing a class of services⁴.

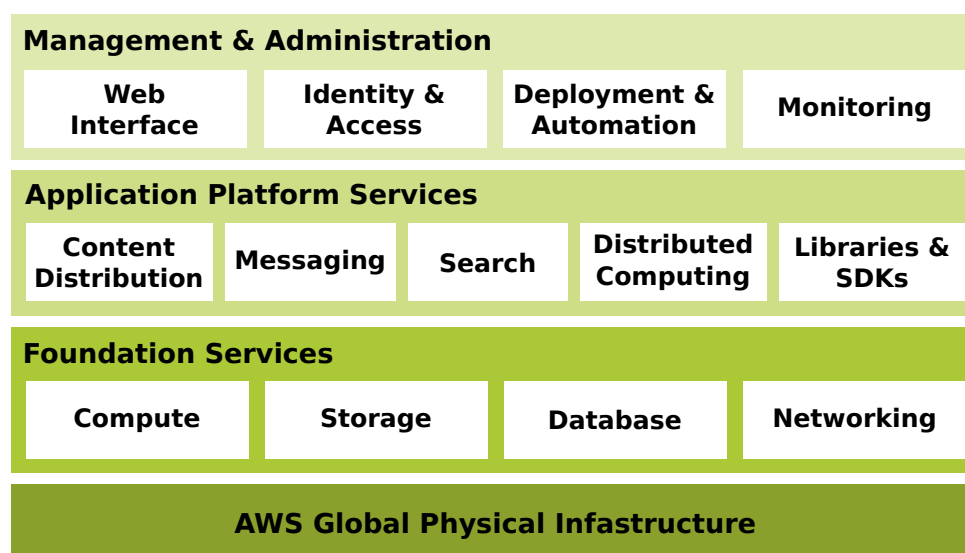


Figure 2.2: Amazon Web Services platform.

At the base of the platform, there is the global physical infrastructure of the Amazon’s cloud – spread over different geographical sites – which is shared among all the services. Above the infrastructure, there are three layers, each one representing a class of service. The first class groups the basic services (computing, storage, database and networking) while in the second,

⁴This figure is adapted from <http://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/intro.html>.

the middle-level services like searching and message queuing are provided. Finally, the third class contains some high level services providing functionality to administrate and monitor the platform or to manage the identity and the permissions of the platform users.

2.3.1 Example application: **theneeds**

A good example showing the AWS services as cloud application building blocks is **theneeds**⁵, a content curation platform that helps readers discover and share the best online content and services tailored to their specific interests. The **theneeds**^α infrastructure⁶ is deployed in two *EC2* instances both using *AutoScale*⁷. In the following, we refer to them with the names Front End (FE) and Back End (BE). The FE is interfaced to Internet through the *Elastic Load Balancer* (ELB) while static contents like images are provided via *Amazon S3*. Furthermore, the management of DNS queries is optimized thanks to the usage of the *Route53* service. For data exchange between FE and BE it is used a database back-end implemented by means of *RDS Multi-AZ*, while to manage HTTP sessions and caching, **theneeds**^α adopts a third party NoSQL back-end (Redis Cloud⁸). All the back-end tasks are dispatched by the FE on a queue implemented using *Amazon SQS* queuing service by the BE and executed on it. About the infrastructure monitoring, the FE is equipped with *CloudWatch*. Moreover, the application data are inspected via a RSyslog based system while the web traffic is analysed through Google Analytics. Finally, **theneeds**^α adopts *CloudSearch* to implement searching capabilities and *Amazon SES* for email based notification delivering. The **theneeds**^α case highlights how the integration of different “standalone” services could help cloud customers in the implementation of a complex cloud based application.

2.4 The TClouds Platform Architecture

The TClouds project defined a secure cloud vision and developed a platform – depicted in Figure 2.3 – accordingly. The platform is composed of a set of components, called subsystems and listed in Table 2.1. These components are grouped in three classes: *infrastructure*, *middleware* and *services*.

The first class of subsystems allows to make more secure the services standing at IaaS logical layer, while the second and the third classes do the same for the PaaS layer. In particular, the libraries belonging to the middleware class are the foundation for the third class of subsystems, i.e. the actual services at PaaS layer. In the first class we can further distinguish between subsystems that implement/enhance the core technology with security features, and subsystems targeted to the cloud management (marked as *mgmt* in both Figure 2.3 and Table 2.1). In the second and third classes we can further distinguish between subsystems working in a single cloud and those enabling the cloud-of-clouds paradigm (marked as *CoC* in both Figure 2.3 and Table 2.1). The TClouds platform is, therefore, a set of building blocks for applications. Within the project, such platform is being validated by two benchmark scenarios: home healthcare and smart lighting. They consist of two SaaS applications built on top of two different instantiations

⁵<http://www.theneeds.com>

⁶Details by courtesy of E. Cesena (ec@theneeds.com).

⁷All AWS services mentioned in this section are extensively documented in <http://aws.amazon.com/products/>.

⁸<http://redis-cloud.com/>

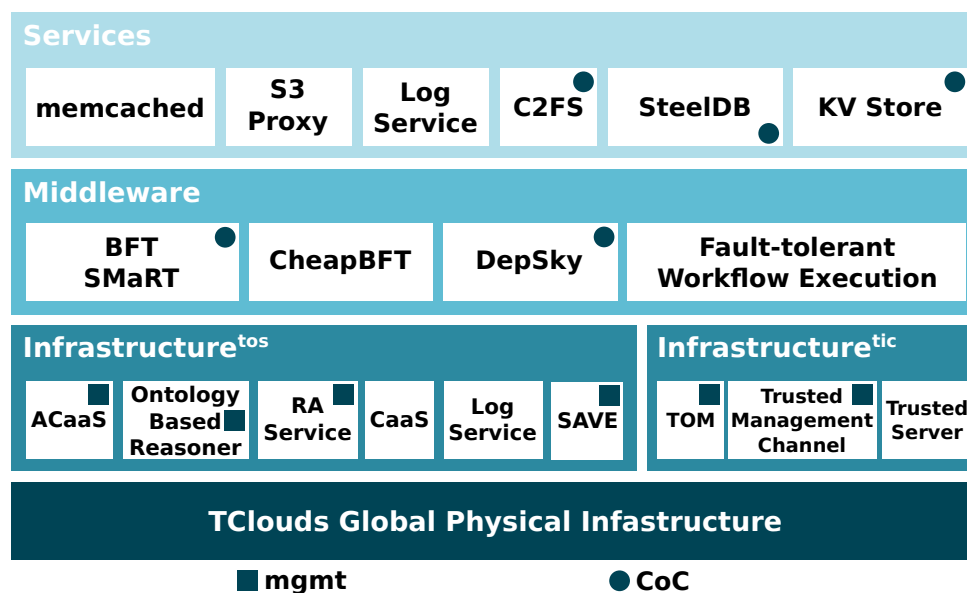


Figure 2.3: The TClouds platform.

of TClouds platform (see Section 2.5). A high level view of how the TClouds platform can be used in multi-cloud environments is given in [VBP12].

2.4.1 Infrastructure

We classify as infrastructure the set of subsystems that can be combined to create two possible secure services at IaaS layer.

The first one, called *Trustworthy OpenStack* (TOS) [S⁺12a], is based on OpenStack⁹, an open-source framework for resource management in cloud environments that integrates several subsystems (the group called infrastructure^{tos} in the Figure 2.3). *Access Control as a Service* (ACaaS) and *Remote Attestation Service* (RA Service) allow to create a trusted infrastructure in which VMs are instantiated on the cloud nodes only if (1) software installed on the nodes is validated against pre-defined sets of measurements (hash of the binaries) and (2) the instantiation respects some access control rules expressed in terms of node security properties requested by the user when launching the VM. Both subsystems are implemented as new filters for the standard OpenStack scheduler that, this way, acquires new capabilities for planning VM deployments. The RA Service is based on the OpenAttestation¹⁰ SDK and on an integrity analysis tool for Linux distributions [CRS⁺11]. *Ontology-Based Reasoner-Enforcer*¹¹ is an enhancement of libvirt library to support the Trusted Virtual Domains (TVDs) and a plugin for Quantum¹² component, thus enhancing the capabilities of OpenStack to instantiate per-customer TVDs; it can work together with *Security Assurance of Virtualized Environments* (SAVE) [BGSE11] to verify the effectiveness of the TVD isolation. The *Cryptography as a Service* (CaaS) [BBI⁺13], based on Xen hypervisor, allows the VM image and disk volumes protection while the VM is running by using transparent on-the-fly encryption/decryption. *Log Service* provides a secure log for the cloud, both as IaaS and PaaS layers: confidentiality, access control and forward

⁹<http://www.openstack.org/>

¹⁰<https://github.com/OpenAttestation/OpenAttestation>

¹¹Currently only the enforcer part is implemented.

¹²Quantum is a virtual networking service for OpenStack, available since the Folsom version: <http://docs.openstack.org/folsom/openstack-network/admin/content/index.html>

TClouds subsystem	Classification	Description
Access Control as a Service (ACaaS)	Infrastructure ^{tos} [mgmt]	It ensures that user VMs are only executed on cloud nodes matching their security requirements.
Ontology-based Reasoner-Enforcer	Infrastructure ^{tos} [mgmt]	It allows the definition of TVDs and enforces the network isolation property at infrastructure level.
Remote Attestation Service (RA Service)	Infrastructure ^{tos} [mgmt]	Web based framework performing the integrity verification of the cloud nodes via Remote Attestation.
Cryptography as a Service (CaaS)	Infrastructure ^{tos}	It enables a VM to use an encrypted storage device transparently (including the root file system) as if it were plaintext.
Log Service (*)	Infrastructure ^{tos}	Secure logging services providing log files integrity verification and confidentiality preservation.
Security Assurance of Virtualized Environments (SAVE)	Infrastructure ^{tos} [mgmt]	It extracts configuration data from multiple virtualization environments, in order to validate isolation of cloud users.
TrustedObjects Manager (TOM) [mgmt] Trusted Management Channel [mgmt] TrustedServer	Infrastructure ^{tic}	Ensures integrity by means of Trusted Computing technologies, providing TVDs (securely isolated and encrypted computing, networking and storage resources) and controlling all management aspects of the cloud, abandoning the threat of cloud administrators with elevated privileges.
State Machine Replication (BFT-SMaRt)	Middleware [CoC]	General BFT State Machine Replication middleware for Java.
Resource-efficient BFT (CheapBFT)	Middleware	Extension of BFT-SMaRt that makes use of trusted (hardware) components to ensure secure replication with less replicas.
Fault-tolerant Workflow Execution	Middleware	Robust web services orchestration engine based on BPEL.
Resilient Object Storage (DepSky)	Middleware [CoC]	Replication library for storing data in a set of S3-like cloud storage services.
Simple Key/Value Store (memcached)	Service	It efficiently stores in memory Key-Value pairs acting as a cache.
Confidentiality Proxy for S3 (**)	Service	It provides a confidentiality layer on top of the commodity S3 storage service.
Log Service (*)	Service	Secure logging services providing log files integrity verification and confidentiality preservation.
Cloud-of-Clouds File System (C2FS) [DepSky + BFT-SMaRt]	Service [CoC]	Cloud-backed file system that stores data securely in several clouds.
Fault-tolerant Relational DB (SteelDB) [CheapBFT / BFT-SMaRt]	Service [CoC if BFT-SMaRt]	SQL database replication engine implemented over BFT-SMaRt or CheapBFT
KV-Store [CheapBFT / BFT-SMaRt]	Service [CoC if BFT-SMaRt]	Key-value store implemented using BFT-SMaRt or CheapBFT

(*) The same subsystem Log Service is used with two different instantiations as infrastructure and service component.

(**) This subsystem is also integrated with Infrastructure^{tic} for the transparent encryption setup within a TVD.

Table 2.1: TClouds subsystems and classification. Some services are based on middleware ones [in brackets].

integrity of the log entries are guaranteed by design through different secure logging schemes (Schneier-Kelsey [SK99a] is the first one implemented). A resilient version of the Log Service uses CheapBFT middleware (see Subsection 2.4.2) to guarantee the availability of this service despite the occurrence of Byzantine faults.

Most of the mentioned subsystems are meant to increase the security of the customer's virtual resources against other customers or by guaranteeing the access to higher security cloud resources. CaaS goes further and guarantees a security property, the confidentiality of VM volumes, against a malicious cloud administrator by exploiting trusted computing technology. Most subsystems (RA Service, Ontology-Based Reasoner-Enforcer, Log Service and SAVE) can also be used independently from Trustworthy OpenStack or in conjunction with other cloud frameworks.

The second infrastructure, called *TrustedInfrastructure Cloud (TIC)* [S⁺12a] developed in the project, is based on a proprietary technology to create virtual infrastructures (network plus VMs) in which information flow constraints are enforced in a secure way through the native use

of Trusted Computing. It integrates the following subsystems (the group called infrastructure^{tic} in the Figure 2.3): *Trusted Server*, a computing node in the infrastructure, *Trusted Object Manager* (TOM), the management component, and a *Trusted Management Channel* for secure authentic communication between TOM and a Trusted Server.

The motivation for implementing two distinct infrastructures is as follows. TrustedInfrastructure Cloud is constructed from ground up with security and trustworthiness in mind, employing trusted computing technologies as a hardware anchor. With trusted boot and remote attestation we ensure that only untampered servers with our security kernel are started and that the sole way of administration is via the trusted channel from the management component TOM. Hence no administrator with elevated privileges is necessary and hence this functionality is completely disabled, abandoning the possibility for an administrator to corrupt the system. On the contrary, Trustworthy OpenStack is based on OpenStack which has a strong bias towards a scalable and decentralized architecture. We extend or embed new components into the OpenStack framework to improve its security. With these two infrastructures we can cover the needs of wider range of application scenarios. TrustedInfrastructure Cloud is especially attractive for private or community clouds with high security demands, while Trustworthy OpenStack is attractive for large-scale public clouds.

Given one of these two infrastructures to support the execution of virtual machines, one can develop applications for such secure cloud. These applications can make use of some other TClouds subsystems that fall in one of the two classes: middleware and services.

2.4.2 Middleware

Middleware components provide programming libraries and services that can be used to implement applications and services to be deployed both in a TClouds infrastructure or any other standard computing environment. The TClouds platform defines replication middleware components that are aligned with one of the main objectives of the project: avoiding single-point of failures.

The first type of middleware provided in TClouds is based on the State Machine Replication (SMR) paradigm [Sch90], where clients can invoke operations that are executed in a set of replicas in a coordinated way, i.e., all replicas execute the same sequence of operations, even if a subset of these replicas are subject to arbitrary failures that may crash or corrupt the replica state.

The platform provides two implementations of SMR. The first is BFT-SMaRt¹³, an open-source Java programming library that implements state-of-the-art replication algorithms to tolerate up to f Byzantine faults in a group of at least $3f + 1$ replicas. The second SMR middleware is CheapBFT [R. 12], an extension/redesign of BFT-SMaRt implementing a novel replication protocol that requires only $f + 1$ active replicas (plus f backup replicas) to tolerate up to f Byzantine faults.

Notice that CheapBFT requires less replicas than BFT-SMaRt, but it requires the replicas to be equipped with trusted component (e.g., a TPM) as the one provided in the TClouds infrastructural solutions. BFT-SMaRt, on the other hand, makes no assumption about the replicas, and thus can be used to implement replicated services with replicas spread around different cloud providers, enabling the cloud-of-clouds paradigm.

A second type of replication middleware provided in TClouds is the resilient object storage implemented in the DepSky cloud-of-clouds replication storage [BCQ⁺13]. DepSky is a Java

¹³<http://code.google.com/p/bft-smart>

programming library that implements a set of replication protocols that store objects (variable-size byte arrays) in a set of cloud storage services (e.g., Amazon S3, Rackspace Files, Google Storage, etc.). Notice that contrary to the SMR solutions, DepSky does not support service replication (*i.e.*, there is no server or general service), as the client-side programming library can be used only to store data securely in multiple cloud storage providers.

Finally, the third and last replication middleware is a BPEL-based orchestration engine that can be used to coordinate interactions between multiple services in a cloud environment. The key innovation of this middleware/service, when compared with similar engines, is that it is fault-tolerant [BDH⁺12].

2.4.3 Services

Services, on the other hand, are subsystems that can be used by external applications. A primary example of services would be the several storage solutions devised in the project. Some of the services developed in TClouds directly use the middleware developed within the project. In fact, most of the innovation of these services are due to this middleware.

Table 2.2 shows a comparison in terms of the resources required to run different storage services of the TClouds platform. The costs are given in terms of the number of VMs, Hardware-enhanced VMs (HVM) or storage clouds required to run these services.

Service	Resources
KV-SMR (CheapBFT)	$2f + 1$ HVMs
KV-SMR (BFT-SMaRt)	$3f + 1$ VMs
SteelDB (CheapBFT)	$2f + 1$ HVMs
SteelDB (BFT-SMaRt)	$3f + 1$ VMs
Simple Key/Value Store (memcached)	VM
Confidentiality Proxy for S3	single storage cloud
C2FS (DepSky)	$3f + 1$ storage clouds
Log Service	VM
Log Service (CheapBFT)	$2f + 1$ HVMs

Table 2.2: TClouds services and required resources.

In a similar way to AWS and other cloud providers, a TClouds-enabled provider could also offer several secure and high-available database solutions. These solutions aims to support different application needs and require different amount of resources. The KV-SMR is a highly dependable and durable storage service (that is able to tolerate even the most conspicuous failure behaviors) that can be used by applications that do not require relational semantics in their data store (in the same spirit of NoSQL databases). The implementation of such storage service is simplified by the development of a durability layer for SMR-based storage services [BSF⁺13]. Finally, the Fault-tolerant Relational DB (SteelDB) is an implementation of a Byzantine fault-tolerant relational database based on TClouds BFT middleware. The key idea here is to implement the Byzantium algorithm [GRP11] to synchronize the database replicas only when transactions are committed. Both KV-SMR and SteelDB can be deployed over BFT-SMaRt or CheapBFT. As expected, If BFT-SMaRt is used, four replicas are required to tolerate a single fault, but plain VMs could be used to run these replicas. If CheapBFT is used, only two active replicas and one backup replica is required to tolerate one Byzantine fault, however, the VMs need to be hardened by a trusted component.

The platform also offers dedicated VMs that can be used as (non-durable) main-memory cache for applications subject to low-latency requirements: the simple key/value storage.

In terms of (external) object storage, two enhanced services are provided in TClouds. The first is a confidentiality proxy for AWS' S3 storage service. The idea of this service is to use external cloud storage services ensuring the confidentiality and integrity of the data stored there. The other one is C2FS [S⁺12a], a cloud-of-clouds file system, that uses the DepSky middleware to store data in several cloud storage providers. C2FS offers confidentiality, integrity and *availability*, even if some of the cloud providers supporting the service are unavailable or offline. Despite the fact C2FS requires four clouds to tolerate a single faulty provider, its cost in terms of storage is just 50% more storage space than what is required when storing the data in a single provider (e.g., when using the confidentiality proxy). This is achieved with the use of RAID-like techniques implemented by DepSky [BCQ⁺13].

The only storage offer that is not suited for storing general application data is the log service. This is a specialized service that can be used to store secure logs for the purpose of *accountability and auditability* of TClouds-based applications. The LogService can be used with a single remote server for storing the logs (being thus subject of the same problems as in normal, non-replicated systems), or be deployed using a log storage over CheapBFT middleware, in which the availability and integrity of the logs is ensured even if some of the storage nodes are compromised.

2.5 Platform Instantiations

To demonstrate and evaluate the TClouds platform we instantiate it in the context of two application scenarios: home healthcare and smart lighting. The home healthcare scenario develops a home monitoring system for depressed patients including various stakeholders: patients, relatives, therapists, etc. The core issue is the processing of the confidential patient data. The smart lighting system controls the public lighting of a country, and the main requirement is the integrity and availability of the system. These scenarios employ different subsets of the TClouds subsystems to match their specific requirements. In the following we list the subsystems used in each scenario and provide a brief description of why they are used in the scenario.

The home healthcare scenario (Figure 2.4) [D⁺13] is hosted on top of Trustworthy Open-Stack. This platform provides some core security benefits which are the foundation for the security requirements for the application layer. The application logic itself enforces a fine grained access control on the processed patient data to ensure privacy of the data. Moreover, the application directly employs the Log Service to securely store sensitive accesses to the data within the log.

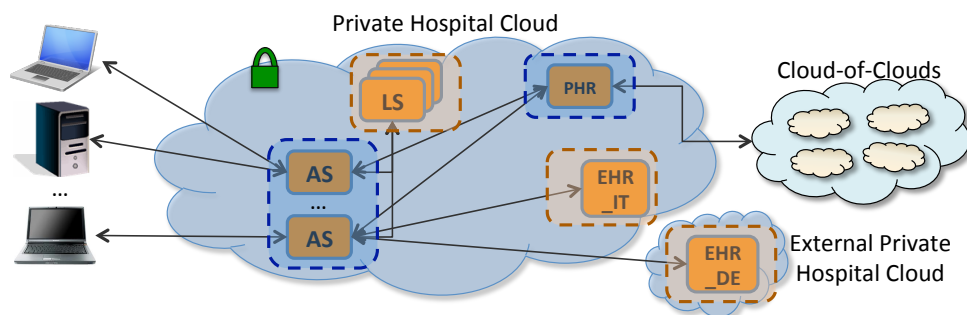


Figure 2.4: Home healthcare scenario.

In this scenario, (1) the *Cryptography as a Service* is used to provide secure key management and confidentiality for storage and VM images; (2) *Access Control as a Service* and *Remote Attestation Service* are used to control the geolocation where the VMs are deployed; (3) the Ontology-based Reasoner is used for secure separation of tenants by means of Trusted Virtual Domains; (4) the Log Service to provide tamper-proof logging of infrastructure and application events; (5) CheapBFT is used to add fault tolerance to the log service; (6) the C2FS is employed for backup important data in the cloud-of-clouds; (7) the SAVE tool is used to check the information flows and verify trusted virtual domains (TVD) isolation; finally, (8) the Simple Key / Value Store (memcached) is used for caching database data, avoiding the latency of wide-area replication.

The smart lighting scenario (Figure 2.5) [Per13] is hosted on top of the TrustedInfrastructure Cloud. The core of the application is a highly-dependable relational database which stores the smart lighting configurations. There are two main requirements for the the database: integrity and availability. These requirements are satisfied by replicating the database using the BFT State Machine Replication middleware developed in TClouds. Moreover the application provides two interfaces, one to general web browsers that can only read data from the database, and a second one, with write privileges, that can only be accessed by machines within a TVD.

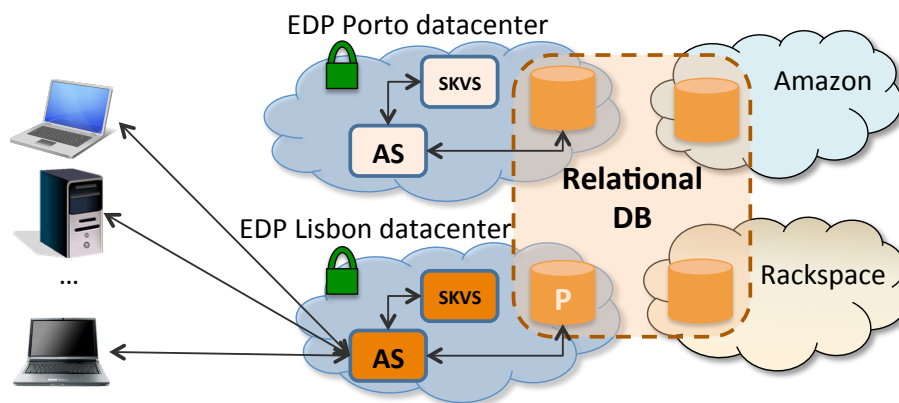


Figure 2.5: Smart lighting scenario.

This scenario uses the following TClouds subsystems: (1) Trusted Objects Manager, Trusted Server, Trusted Management Channel, which together provide the core cloud infrastructure with strong integrity and confidentiality properties to implement TVDs; and (2) State Machine Replication (BFT-SMaRt), to provide Byzantine fault tolerance to the SteelDB, with its four replicas deployed in two trusted clouds and two commercial clouds.

Part II

Testing

Chapter 3

Final test plans for subsystems/prototypes

Chapter Authors: Roberto Sassu (POL), Paolo Smiraglia (POL); Alexander Buerger, Norbert Schirmer (SRX); Alysson Bessani, Marcel Henrique dos Santos (FFCUL); Sören Bleikertz, Zoltan Nagy (IBM); Imad M. Abbadi, Anbang Ruad (OXFD); Johannes Behl, Klaus Stengel (TUBS); Mihai Bucicoiu, Sven Bugiel, Hugo Hideler, Stefan Nürnberger (TUDA).

3.1 Introduction

This chapter collects the final test plans for subsystems and prototypes. Such plans have been defined according to the testing methodology already used during the previous year and described in the Sections 4.1, 4.2, 4.3 and 4.5 of D2.4.2 [S⁺12a].

3.2 TrustedInfrastructure Cloud

3.2.1 Test methodology/strategy

The TrustedObjectsManager component will be tested in a manual way. The prerequisites for this testing are a readily setup TrustedObjectsManager, and a TrustedDesktop – as well as a TrustedServer instance connected together via network (LAN/WAN). In order to operate the tests, at least one applicable virtual machine instance (VirtualBox) is required for addressing the envisaged tests. A successful test will fulfill all test cases described in Chapter 3.2.2. This test cannot be automated since user interaction with different physical machines is required.

3.2.2 Test cases

- Type of test: manual
- Coverage: high
- Description of the procedure:
 - setup and configure TrustedObjectsManager
 - setup and configure TrustedServer
 - setup and configure TrustedDesktop
 - expected result: Virtual machine runs and TrustedServer can be used on Trusted-Desktop

TEST CASE ID	/TC 3.2.2-1/ Create compartment on TrustedObjectsManager
DESCRIPTION	Create a compartment on TrustedObjectsManager in order to be capable to start and stop it on TrustedServer and use it's provided services on TrustedDesktop from within the same TrustedVirtualDomain
TYPE	Functional test
PRECONDITIONS	The user is logged in on the TrustedObjectsManager. A virtual-disk image is available locally
STEPS	<ol style="list-style-type: none"> 1. User creates a new TrustedVirtualDomain by choosing "New TVD" and assigning a name and a color to it 2. User right-clicks on the newly created TrustedVirtualDomain and chooses "Compartments" 3. User selects "New", "Compartment Manager" and clicks "Upload" 4. User selects the unassigned virtual-disk image and presses "OK" 5. User waits for the upload to be finished 6. User waits for the calculation of the SHA1-sum 7. User clicks "Close" 8. User assigns a compartment name (without whitespaces) to the newly created compartment 9. User checks "Enable Compartment" 10. User unchecks "Enforce client update" 11. User chooses the uploaded virtual-disc image from the dropdown menu 12. User clicks "Apply" and "OK"

TEST CASE ID	/TC 3.2.2-2/ Start compartment
DESCRIPTION	Start a compartment on TrustedServer from the TrustedObjectsManager's GUI
TYPE	Functional test
PRECONDITIONS	The user is logged in on the TrustedObjectsManager. The TrustedServer is connected to the TrustedObjectsManager. A compartment is installed but not running on the TrustedServer.
STEPS	<ol style="list-style-type: none"> 1. The user selects the TrustedServer, the compartment should be started on 2. The user selects a compartment that is installed but not currently running on the TrustedServer 3. The user triggers a start of this compartment on the TrustedServer 4. The compartment should be running on the TrustedServer

TEST CASE ID	/TC 3.2.2-3/ Service usable from TrustedDesktop
DESCRIPTION	A service within a running compartment on TrustedServer can be used from the same TrustedVirtualDomain on TrustedDesktop
TYPE	Functional test
PRECONDITIONS	The TrustedServer is running and a compartment providing a service is started. The user is logged in on TrustedDesktop. The TrustedDesktop is connected to the TrustedServer. A compartment within the same TrustedVirtualDomain as the service provided by the TrustedServer, is started on TrustedDesktop.
STEPS	<ol style="list-style-type: none"> 1. User uses the service from within the compartment 2. The service answers as expected

TEST CASE ID	/TC 3.2.2-4/ Stop compartment
DESCRIPTION	Start a compartment on TrustedServer from the TrustedObjectsManager's GUI
TYPE	Functional test
PRECONDITIONS	The user is logged in on the TrustedObjectsManager. The TrustedServer is connected to the TrustedObjectsManager. A compartment is running on the TrustedServer.
STEPS	<ol style="list-style-type: none"> 1. The user selects the TrustedServer on which the running compartment should be stopped 2. The user selects the running compartment on the TrustedServer 3. The user triggers a stop of this compartment on the TrustedServer 4. The compartment should be shutdown on the TrustedServer

TEST CASE ID	/TC 3.2.2-5/ Mount Amazon S3 storage on TrustedServer
DESCRIPTION	A predefined Amazon S3 storage bucket is mounted to TrustedServer
TYPE	Functional test
PRECONDITIONS	An empty S3-bucket was already defined via the AWS console The user is logged in on the TrustedObjectsManager The TrustedServer is connected to the TrustedObjectsManager The TrustedServer is connected to the Internet
STEPS	<ol style="list-style-type: none"> 1. The user selects the S3 storage tab in the preferences of the TrustedServer 2. The user enters the S3 backend credentials 3. The user enters the name of the predefined S3 bucket 4. The user enters an encryption password for the files stored within the bucket 5. The predefined Amazon S3 bucket is mounted on the TrustedServer

TEST CASE ID	/TC 3.2.2-6/ Provide Amazon S3 storage to compartments via Trusted-Server
DESCRIPTION	The mounted Amazon S3 bucket storage can be used securely from within any TVD/VM defined in TrustedInfrastructures
TYPE	Functional test
PRECONDITIONS	The user is logged in on an TrustedDesktop attached to TrustedInfrastructures. The TrustedServer is connected to the TrustedObjectsManager. A compartment within a TVD is running on the TrustedServer. A compartment from within the same TVD is running on TrustedDesktop A S3 storage bucket is mounted to TrustedServer
STEPS	<ol style="list-style-type: none"> 1. The user of TrustedDesktop enters the compartment in a TVD 2. The user mounts the single provided smb/cifs-share to the OS, running within the compartment 3. The transparently encrypted smb/cifs-share can be used to store files
REMARKS	The encryption showcase cannot be mapped from within any TVD because of its intended transparency

TEST CASE ID	/TC 3.2.2-7/ Mount Amazon S3 storage to S3 confidentiality proxy
DESCRIPTION	A predefined Amazon S3 storage bucket is mounted to the S3 proxy appliance
TYPE	Functional test
PRECONDITIONS	The users' computer is physically connected to the S3 proxy appliance via the internal network port (eth1) The S3 proxy appliance is connected to the internet
STEPS	<ol style="list-style-type: none"> 1. The user opens a webbrowser and accesses the configuration interface of the S3 proxy appliance via https://192.168.1.2 2. The user enters the credentials to access the website 3. On the presented page, the user chooses one entry of the backend-list (here Amazon S3) 4. The user enters the S3 backend credentials 5. The user enters the name of the predefined S3 bucket 6. The user enters an encryption password for the files stored within the bucket 7. The user clicks the "Mount"-button 8. A message is shown, that the S3 bucket is mounted

TEST CASE ID	/TC 3.2.2-8/ Provide Amazon S3 storage to physical machines attached to S3 confidentiality proxy
DESCRIPTION	The mounted S3 storage bucket can be shared by machines attached to the internal network side (eth1) of the S3 proxy appliance
TYPE	Functional test
PRECONDITIONS	Amazon S3 storage is mounted to the S3 confidentiality proxy The users' computer is physically connected to the internal network of the S3 proxy appliance
STEPS	<ol style="list-style-type: none">1. The user mounts the smb/cifs-share, provided by the S3 confidentiality proxy2. The transparently encrypted smb/cifs-share can be used to store files

3.3 Security Assurance of Virtualized Environments (SAVE)

3.3.1 Test methodology/strategy

The *Discovery* component will be tested in a manual way. The requirement for this testing is an existing OpenStack infrastructure (with our OpenStack discovery extensions) that we will try to discover. A successful test will return the discovery data for this OpenStack infrastructure. This test may become automated, once the discovery and infrastructure is configured, by periodically trying to perform the discovery.

The *Analysis* component is tested using automated unit testing (JUnit) as well as overall system testing. The unit tests will verify that the translation of the discovery data into our unified graph model is performed correctly by using sample input data and reference output data. The overall system testing will perform the analysis of known good and known vulnerable infrastructures (given by its discovery data), which need to be correctly identified as such.

3.3.2 Test cases

Discovery

- type of test: manual / semi-automated
- coverage: medium
- description of the procedure:
 - setup OpenStack test infrastructure with our discovery extension
 - configure discovery with host and credentials
 - run discovery with configuration (can be done periodically afterwards for semi automation)
 - expected output: discovery data and successful termination

Analysis Unit Testing

- type of test: automated
- coverage: medium
- description of the procedure:
 - Test are aggregated in a JUnit runner

Analysis System Testing

- type of test: manual / semi-automated
- coverage: medium
- description of the procedure:
 - Run analysis against known good and known vulnerable infrastructures given by its discovery data

- For known good: analysis should return no problems
- For known vulnerable: indicate isolation problems

3.4 Tailored memcached service

3.4.1 Short subsystem intro

The main goal of this subsystem is to provide a platform for simple cloud-based services. We provide the infrastructure to discover requirements for the service based on the application and then provide a tailored service. In order to demonstrate the platform's capabilities, we implement a variant of the memcached service on top of it.

The basic building blocks are a VM image containing the generalized service and the application trying to use the service. After a short setup phase where the service is tailored to the application's usage profile, the actual service can be used.

3.4.2 Test methodology/strategy

The subsystem will be tested in several different service configurations with existing benchmark solutions for memcached. This covers both the tailoring process as well as the actual demo service.

In order to operate the test, at least one virtual machine instance is required for setting up the tailored service, and another host (physical or virtual) is needed for running the benchmark programs. Both must reside on the same physical network and should be as close together as possible in order to reduce communication latencies.

3.4.3 Test cases

TEST CASE ID	/TC 3.4.3-1/ Test service deployment
DESCRIPTION	This test will check if the service can be deployed on the OpenStack infrastructure
TYPE	Functional test
PRECONDITIONS	Trusted OpenStack platform is up and running Compiled tailored memcached image available memslap benchmark program
STEPS	<ol style="list-style-type: none"> 1. Login to OpenStack dashboard 2. Upload compiled Tailored Memcached image to OpenStack platform as type "AKI" (Amazon Kernel Image) 3. Upload small, empty disk image (1 MB) as "AMI" type 4. Configure image to require a paravirtualized Xen environment 5. Start a new instance of AMI with tailored memcached kernel (AKI) 6. Wait for memcached instance to obtain an IP address (10 seconds) 7. Run memslap benchmark with 100 test entries against the reported IP address and check results

TEST CASE ID	/TC 3.4.3-2/ Test tailoring
DESCRIPTION	This test will check if the service tailoring works as expected
TYPE	Functional test
PRECONDITIONS	Trusted OpenStack platform is up and running Precompiled tailored memcached image available with different function sets: One version with prepend/append function set enabled and one without.
STEPS	<ol style="list-style-type: none"> 1. Deploy both versions of tailored memcached on the Trusted OpenStack platform as outlined in /TC 3.4.3-1/ 2. Start two <code>telnet</code> sessions, one for each service 3. Establish a connection to both services 4. Switch <code>telnet</code> into <i>CRLF</i>-mode using the command <code>set crlf</code> on the <code>telnet</code> escape prompt 5. Create a entry for testing by issuing the command <code>set key 0 9000 3</code>, followed by <code>foo</code> as content. 6. Try to append text on both by sending <code>append key 0 9000 3</code> and then <code>bar</code> on the next line. 7. The instance with the feature enabled should confirm with the text <code>STORED</code> that the operation succeeded while the other instance does not.

3.5 Fault-tolerant Workflow Execution (FT-BPEL)

3.5.1 Test methodology/strategy

Similar to the CheapBFT subsystem, FT-BPEL has a complex setting and targets the masking of errors, which makes it very difficult to design and implement automated tests. In the case of FT-BPEL, a test configuration comprises several Apache Tomcat instances with different settings as well as an Apache ZooKeeper service with multiple replicas. Therefore, the same approach as with CheapBFT is chosen: The tests are carried out manually, but scripts and tools are provided to aid the proceeding.

In a real set-up, a minimum of ten (virtual) machines connected through a TCP/IP network would be needed. However, it is also possible to co-locate different replicas of different services lowering the number of required machines to four (three for the replicas and one for the client). It is assumed that a Linux system is running on all used machines and that the FT-BPEL software components are accessible from every machine, for instance, by means of a network file system.

3.5.2 Test cases

TEST CASE ID	/TC 3.5.2-1/ Fault-free operation (standard infrastructure)
DESCRIPTION	FT-BPEL is designed to serve as compatible replacement with enhanced fault-tolerance properties for existing standard BPEL infrastructures. Therefore, the first test should attest the compatibility and that the replication of the services does not lead to faulty service implementations. For that purpose, the same test scenario is deployed in two different settings, first in an unreplicated, standard setting and then using the FT-BPEL platform (/TC 3.5.2-2/). After the initialization of the system, a client is executed that continuously invokes a composed Web service which in turn invokes other Web services in order to process the client requests.
TYPE	Functional test
PRECONDITIONS	The test environment, which comprises at least four machines (three replica servers and one client) that are provided with all required software modules, is up and running. FT-BPEL is properly configured, especially the addresses of the machines and the location of the program files are set.
STEPS	<ol style="list-style-type: none"> 1. Set up a standard BPEL system with unreplicated BPEL process and unreplicated Web services: <pre>./rbpel.bash setup_scen -f 0 calc</pre> 2. Start the test: <pre>./rbpel.bash start \ http://services.net/calculator</pre> 3. The client should continuously issue requests which should be responded by the system. 4. Shut down and clean up: <pre>./rbpel.bash cleanup -t</pre>
NOTES	The script <code>rbpel.bash</code> is located at the directory <code>bin</code> .

TEST CASE ID	/TC 3.5.2-2/ Fault-free operation (FT-BPEL infrastructure)
DESCRIPTION	See /TC 3.5.2-1/
TYPE	Functional test
PRECONDITIONS	See /TC 3.5.2-1/
STEPS	<ol style="list-style-type: none"> 1. Set up an FT-BPEL system with replicated BPEL process and replicated Web services: <pre>./rbpel.bash setup_scen -f 1 calc</pre> 2. Start the same test: <pre>./rbpel.bash start \ http://services.net/calculator</pre> 3. Again, the client should continuously issue requests which should be responded by the system. 4. Shut down and clean up: <pre>./rbpel.bash cleanup -t</pre>
NOTES	<p>The script <code>rbpel.bash</code> is located at the directory <code>bin</code>.</p> <p>The same test client is used for the standard case (/TC 3.5.2-1/) and for the replicated FT-BPEL system. This attests that FT-BPEL can be used as a compatible replacement for standard BPEL systems. Further, the client checks the received results with the expected values in order to verify that the replication does not lead to faulty services.</p>

TEST CASE ID	/TC 3.5.2-3/ Operation in the presence of crashes (standard infrastructure)
DESCRIPTION	In order to show that composed Web services executed by means of FT-BPEL remain available even if the used cloud infrastructure is subject the crashes, the same test case as in /TC 3.5.2-1/ is conducted but during the execution instances are terminated to simulate crashes. Again, the behavior of FT-BPEL (/TC 3.5.2-4/) is compared to the behavior of a standard, unreplicated BPEL set-up.
TYPE	Functional test
PRECONDITIONS	Same as for test case /TC 3.5.2-1/ .
STEPS	<ol style="list-style-type: none"> 1. Set up a standard BPEL system with unreplicated BPEL process and unreplicated Web services <pre>./rbpel.bash setup_scen -f 0 calc</pre> 2. Start the test: <pre>./rbpel.bash start \ http://services.net/calculator</pre> 3. The client should continuously issue requests which should be responded by the system. 4. Terminate the BPEL engine or the instance hosting the Web services the composed Web service relies on: <pre>./rbpel.bash kill_host 2 1</pre> <p>or</p> <pre>./rbpel.bash kill_host 1 1</pre> 5. The system is rendered unavailable even if only a single instance is terminated. 6. Shut down and clean up: <pre>./rbpel.bash cleanup -t</pre>

TEST CASE ID	/TC 3.5.2-4/ Operation in the presence of crashes (FT-BPEL infrastructure)
DESCRIPTION	See /TC 3.5.2-3/
TYPE	Functional test
PRECONDITIONS	See /TC 3.5.2-3/
STEPS	<ol style="list-style-type: none"> 1. Set up an FT-BPEL system with replicated BPEL process and replicated Web services: <pre>./rbpel.bash setup_scen -f 1 calc</pre> 2. Start the same test: <pre>./rbpel.bash start \ http://services.net/calculator</pre> 3. Again, the client should continuously issue requests which should be responded by the system. 4. Terminate one BPEL engine replica or one replica of the Web services the composed Web service relies on: <pre>./rbpel.bash kill_host 2 1</pre> <p>or</p> <pre>./rbpel.bash kill_host 1 1</pre> 5. Contrary to the execution based on a standard BPEL set-up, using FT-BPEL ensures that the hosted services remain available and that the client receives correct replies even if a BPEL engine replica and a Web service replica are terminated. 6. Shut down and clean up: <pre>./rbpel.bash cleanup -t</pre>

3.6 Cryptography as a Service

3.6.1 Test methodology/strategy

The Cryptography as a Service will be tested manually, since there are many diverse components involved which do not lend themselves well for automatic testing. For instance, testing a hypervisor, cannot be done on the same machine where the tests are executed at, due to the fact that it needs to run on dedicated hardware because it is the most low level piece of software running on a PC. Moreover, the required trusted boot setup requires an actual hardware TPM to talk to in order to verify the correctness of the setup. This requires a non-trivial testing setup. Furthermore, the interaction between the various domains (VMs) often requires manual intervention to emulate the steps a cloud administrator or consumer would take.

In testing our Cryptography as a Service, it is essential to focus on the exact required functionality. We shall not test functionality of the Xen hypervisor which do not directly relate to the requirements of *confidentiality* and *integrity* of consumer VMs.

Our test cases reflect the customer's requirements for security objectives throughout the entire workflow of VM deployment. More precisely, *confidentiality* and *integrity* in order to protect the assets from anybody else but the customer, especially from the cloud provider or any other cloud tenants. The test cases further reflect a temporal story line from a customer's perspective starting at the moment he or she bundles the VM together with keys, then securely uploads it, to the moment it is running in the verified cloud and has secure access to those keys.

3.6.2 Test cases

TEST CASE ID	/TC 3.6.2-1/ Test domain builder functionality
DESCRIPTION	Start up the domain builder stubdom (DomT), testing whether it can communicate with the TPM (dependency for other tests)
TYPE	Functional test
PRECONDITIONS	The domain builder has ownership of the TPM
STEPS	<ol style="list-style-type: none"> 1. Machine is powered on. 2. GRUB boots with Xen, Dom0 and DomT as multiboot modules. # xl list 3. The pubkey of the TPM has been exported. # hexdump /var/domt_pubkey.bin

TEST CASE ID	/TC 3.6.2-2/ Customer makes encrypted VM available
DESCRIPTION	Customer encrypts her VM with cloud key
TYPE	Functional test
PRECONDITIONS	Domain builder initialized (test 6.2.3.6.2-1)
STEPS	<ol style="list-style-type: none"> 1. The customer takes a working VM image (disk image). <pre># file disk.img</pre> 2. The image is encrypted using a symmetric key <i>k</i>. <pre># ./deployer.py -k domt_pubkey.bin disk.img disk.enc disk_vmcb.enc</pre> 3. the key symmetric key <i>k</i> is encrypted with the asymmetric key from test 6.2.3.6.2-1.

TEST CASE ID	/TC 3.6.2-3/ Deploy and run secure image
DESCRIPTION	Deploy the customer's encrypted image to the cloud
TYPE	Functional test
PRECONDITIONS	Customer VM deployed (test 6.2.3.6.2-2)
STEPS	<ol style="list-style-type: none"> 1. Using <code>scp</code> the customer's image is copied to the Dom0 domain. <pre># scp disk.enc mihai@134.147.62.162:/home/mihai/xen/ # scp disk_vmcb.enc mihai@134.147.62.162:/home/mihai/xen/.</pre> 2. In Dom0 the cloud admin configures the domain in <code>/etc/xen/domain_X.cfg</code> for this domain to have the <code>domc = 1</code> flag. <pre># cat domain.cfg</pre> 3. The cloud admin starts the domain using <pre>xl create /etc/xen/domain_X.cfg</pre> <pre># xl create domain.cfg</pre>

TEST CASE ID	/TC 3.6.2-4/ Test Security of CaaS
DESCRIPTION	Test the avenues via which the cloud administrator can attack
TYPE	Functional test
PRECONDITIONS	Customer VM running (test 6.2.3.6.2-3)
STEPS	<ol style="list-style-type: none">1. The customer shows a successful search for a pattern on the un-encrypted VM.2. The cloud admin tries in vain search for a pattern on the customers encrypted VM.

3.7 Access Control as a Service (ACaaS)

3.7.1 Test methodology/strategy

ACaaS will be tested manually. As ACaaS prototype is serving the purpose of proof-of-concept, the major aspects to be covered through the tests will be functionality.

A working OpenStack cloud should be ready with all those OpenStack services replaced with our modified ACaaS-enabled version. This cloud will include at least two OpenStack compute node, hosting *nova-compute* and *nova-network*, and one OpenStack management node, hosting other nova services, namely *nova-scheduler*, *nova-api*, *nova-volume*, *nova-objectstore* and *glance* services. The management node can at the same time acting as the compute node. Hence at least two connected machines are needed. At least one Virtual Machine image must be pre-configured and uploaded to glance for demonstration. This image can be as simple as a Just-enough Linux system.

3.7.2 Test cases

TEST CASE ID	/TC 3.7.2-1/ User requirement management
DESCRIPTION	Test the user requirement management module. Ensure user requirement catalogues can be added, removed and queried by administrators correctly.
TYPE	Functional test
STEPS	<ol style="list-style-type: none"> 1. Create a requirement. The correct requirement is created with a valid requirement ID. 2. Remove a requirement. The correct requirement with the intended requirement ID is removed. 3. List all requirements. All existing requirements are displayed.

TEST CASE ID	/TC 3.7.2-2/ Infrastructure property management
DESCRIPTION	Ensure infrastructure properties can be specified, removed and queried correctly
TYPE	Functional test
STEPS	<ol style="list-style-type: none"> 1. Specify a property to a host and query the host's properties. The target host is specified with the correct properties. 2. Remove a property of a host and query the host's properties. The correct properties is removed from the target host.

TEST CASE ID	/TC 3.7.2-3/ ACaaS-based VM scheduling
DESCRIPTION	Ensure VMs with specified requirement can only run on hosts with appropriate properties
TYPE	Functional test
PRECONDITIONS	User requirements and infrastructure security properties have been set up (test 7.2.3.7.2-1, 7.2.3.7.2-2)
STEPS	<ol style="list-style-type: none">1. Run a VM instance with at least one host satisfying its requirements, expecting the VM scheduled to the host with satisfying properties2. Run a VM instance with no host satisfying its requirements. expecting VM not scheduled.3. Run a VM instance with at least one host not running any VM belonging to a specified user, expecting the VM scheduled to the host with no VM belonging to the specified user running on it.4. Run a VM instance with all hosts running VMs belonging to a specified user, expecting the VM not scheduled.

3.8 BFT-SMaRt

3.8.1 Test methodology/strategy

BFT-SMaRt has test cases defined using the JUnit framework. The tests are placed together with the source code and can be executed using Java or Apache Ant. The environment to run BFT-SMaRt and the JUnit tests must have JRE version 1.5 or later installed. To run the test cases using the Ant script provided with the source code it is necessary to have Apache Ant installed. JUnit can be downloaded from www.junit.org and Apache Ant can be downloaded from ant.apache.org. Together with the BFT-SMaRt source code there are a few demonstration packages to be used as examples on how to use BFT-SMaRt interfaces. These demos are not part of the test cases but can be used to see the framework running and clients using it. The demos are in the package navigators.smart.tom.demo. Instruction to run the demo packages are in the file README.txt, in the root of BFT-SMaRt source code.

3.8.2 Test cases

Tests in BFT-SMaRt are performed using code defined for the demo package. The tests tries to perform different operations in BFT-SMaRt to guaranty that the protocol responds as expected.

TEST CASE ID	/TC 3.8.2-1/ Test write and query of data in the regular case
DESCRIPTION	The test will insert data in a key value store and queries the servers to guarantee that data was correctly inserted.
TYPE	Unit test
PRECONDITIONS	JUnit framework and JRE are installed correctly.
STEPS	1. Run the test <code>BFTMapClientTest.testRegularCase()</code> .

TEST CASE ID	/TC 3.8.2-2/ Test the protocol in the presence of a faulty non leader replica
DESCRIPTION	The test will insert data in a key value store and queries the servers to guarantee that data was correctly inserted. The test will insert and verify data. After that a replica is turned off and insertion and query of data is performed to verify if the protocol still responds as expected.
TYPE	Unit test
PRECONDITIONS	JUnit framework and JRE are installed correctly.
STEPS	1. Run the test <code>BFTMapClientTest.testStopNonLeader()</code> .

TEST CASE ID	/TC 3.8.2-3/ Test the state transfer protocol
DESCRIPTION	Data is inserted and verified in a key value store. A non leader replica is turned off, data is inserted and the replica is turned on again. A different non leader replica is turned off after that. This test verifies if the first replica that was turned off and on again is capable of respond to requests using the data it received from the state transfer protocol.
TYPE	Unit test
PRECONDITIONS	JUnit framework and JRE are installed correctly.
STEPS	1. Run the test <code>BFTMapClientTest.testStopAndStartNonLeader()</code> .

TEST CASE ID	/TC 3.8.2-4/ Test the leader change protocol
DESCRIPTION	Data is inserted and verified in a key value store. The leader replica is turned off. Requests are sent after the leader removal and results tested to verify if the component still behaviors as expected.
TYPE	Unit test
PRECONDITIONS	JUnit framework and JRE are installed correctly.
STEPS	1. Run the test <code>BFTMapClientTest.testStopLeader()</code> .

TEST CASE ID	/TC 3.8.2-5/ Test the leader change protocol and state transfer protocol
DESCRIPTION	Data is inserted and verified in a key value store. All replicas, including the leader, are turned off and on again, once at a time, in a round robin fashion. After each removal and inclusion of replicas, the state of the application is verified to guarantee that no data was lost during the process.
TYPE	Unit test
PRECONDITIONS	JUnit framework and JRE are installed correctly.
STEPS	1. Run the test <code>BFTMapClientTest.testStopLeaders()</code> .

3.8.3 Demos

BFT-SMaRt source code includes a demo package with several examples to be used as a usage reference. The package is `navigators.smart.tom.demo`. Each folder inside that package is one example containing the client and server classes. To run the demos, there is a script in `runscripts` folder. Files with `.sh` and `.bat` extensions are provided. The command line with arguments to run the scripts are described in the file `README.txt`, in the source root.

3.9 Resilient Object Storage (DepSky)

3.9.1 Test methodology/strategy

DepSky works with different cloud providers to store data in different servers. To test it, it is necessary to have accounts in cloud providers, to be able to store data and analyse the stored data. After having the accounts created it is necessary to have the user and private keys to be used to manipulate data. DepSky has drivers written for different providers, as Amazon or Nirvanix.

3.9.2 Test cases

In the tests described here, DepSky stores data units in four Amazon S3 servers distributed in different locations. To define the locations there is a file `configClouds` under the `config` directory. All tests described here contains a startup procedure which consists in: Start the DepSky client running the code:

```
./DepSky_Run.sh <container_name> <client_id> <DepSky mode>
```

The options are described in the instructions file `README.txt`. For the tests performed we used

```
./DepSky_Run.sh container1 0 1
```

as the command line. Wait for the client to connect to the servers. It is confirmed by the display of the message "All drivers started.". To verify that the data has been written to the cloud, it is necessary to open the cloud provider console. In the tests written we use Amazon S3, so, to verify the data we open the Amazon S3 console in console.aws.amazon.com/s3.

TEST CASE ID	/TC 3.9.2-1/ Test write and query of data in the regular case
DESCRIPTION	The test will run the DepSky client to write data to the cloud and then query the cloud to verify if the data was correctly inserted.
TYPE	Manual test
PRECONDITIONS	Have an Amazon S3 account created with the keys written in the <code>AWS-Credentials.properties</code> configuration file in the root of DepSky source code. Have the DepSky client code, <code>DepSky_Run.sh</code> with execution privileges in the file system.
STEPS	<ol style="list-style-type: none"> 1. First, DepSky client must be initialized, as described above. 2. Data is inserted, using the command <code>write</code>. 3. Verify in the cloud console that the buckets with the data units was created for the locations defined in the configuration file <code>configClouds</code>. 4. Verify if it is not possible to read useful data from the data units. 5. Query the data is retrieved with the command <code>read</code>.

TEST CASE ID	/TC 3.9.2-2/ Validate DepSky confidentiality and consistency against data loss or server is disconnected
DESCRIPTION	The test will run the DepSky client to write data to the cloud, remove the data written from f servers and query the data in the client.
TYPE	Manual test
PRECONDITIONS	Have an Amazon S3 account created with the keys written in the AWS-Credentials.properties configuration file in the root of DepSky source code. Have the DepSky client code, DepSky_Run.sh with execution privileges in the filesystem.
STEPS	<ol style="list-style-type: none"> 1. First, DepSky client must be initialized, as described above. 2. Data is inserted, using the command write. 3. Verify in the cloud console that the buckets with the data units was created for the locations defined in the configuration file configClouds. 4. Verify if it is not possible to read useful data from the data units. 5. Choose one server from the list of servers and remove the data units created. The data units are inside the folder with the container name defined when the DepSky client was started. 6. Query the data is retrieved with the command read.

TEST CASE ID	/TC 3.9.2-3/ Validate DepSky confidentiality and consistency for modified data
DESCRIPTION	The test will run the DepSky client to write data to the cloud and modify the data written in f servers.
TYPE	Manual test
PRECONDITIONS	Have an Amazon S3 account created with the keys written in the AWS-Credentials.properties configuration file in the root of DepSky source code. Have the DepSky client code, DepSky_Run.sh with execution privileges in the filesystem.
STEPS	<ol style="list-style-type: none"> 1. First, DepSky client must be initialized, as described above. 2. Data is inserted, using the command write. 3. Verify in the cloud console that the buckets with the data units was created for the locations defined in the configuration file configClouds. 4. Verify if it is not possible to read useful data from the data units. 5. Choose one server from the list of servers and open the folder with the container name defined. 6. Replace data units with files with the same name but different content. 7. Query the data is retrieved with the command read.

3.10 LogService

3.10.1 Test methodology/strategy

The LogService is the service within the Trustworthy OpenStack infrastructure providing secure logging functionality. The core component of the LogService is the `libseclog` library that is a refactoring of the `libsklog` library already presented in [S⁺12b] (see Section 9.3.1). To test the `libseclog` functionality, and hence those of the LogService, we defined a set of unit tests using the framework for C language CUnit [KS12].

3.10.2 Test execution

To execute the tests, it's necessary to install the library dependencies and then to compile it with the option `--enable-tests`. The LogService has been developed on Debian “Jessie”, hence the installation and execution instructions are provided considering a Debian based system.

Installation

All the `libseclog` dependencies could be installed using the Debian package manager, except the `libumberlog` library. To install them, run the following commands:

```
sudo apt-get update
sudo apt-get install make autoconf libtool libssl-dev uuid-dev libjansson-dev git pkg-config

git clone https://github.com/deirf/libumberlog.git libumberlog
cd libumberlog
mkdir m4
autoreconf -i
./configure
make
sudo make install
```

Figure 3.1: `libseclog` dependencies installation

To build the library with tests enabled, run the following comands

```
tar zxvf libseclog-<VERSION>.tar.gz
cd libseclog
./autogen.sh
./configure --enable-ceelog --with-umberlog=/urs/local --enable-python --enable-tests
make
```

Figure 3.2: `libseclog` building commands

Test cases

The `libseclog` code is organised in independent logical code units, each one containing the code that implements a specific part such as helpers functions, logging schemes related functions and high level functions. To test all the logical code units, a single testing suite has been defined.

TEST CASE ID	/TC 3.10.2-1/ Helper, logging schemes and high level functions
DESCRIPTION	Tests the helper functions (e.g. cryptographic operation and the user notifications). Moreover, it tests also the functions implementing the supported logging schemes (functions implementing the Schneier's scheme) as well as the high level functions provided by the library. The test suite could be executed with an optional parameter (-n) that specifies the number of the dummy log entries that will be generated during the run.
TYPE	Unit Test, Functional Test
PRECONDITIONS	libseclog built with the --enable-tests option (See Figures 3.1 and 3.2)
STEPS	<ol style="list-style-type: none">1. Move to the test directory <pre>cd test</pre>2. Run the command <pre>./run_tests -n 1000</pre>

3.11 Remote Attestation Service

3.11.1 Test methodology/strategy

The test cases cover only the `RA Verifier` component of the *Remote Attestation Service*, as it is the one developed by POL. Since `RA Verifier` is entirely written in Python language, we will implement them by using the *Pyunit* framework.

Our test cases check the functionality of the component, as it is very critical that the service gives the expected verification results, and accomplish this task through the black-box testing technique. First, they verify that data have been correctly inserted into the database by performing some queries and, then, compare the results obtained from the verification of sample IMA¹ measurements files with the expected ones.

Tests can be executed on a single machine and require the installation of the software specified in the deliverable D2.1.4/2.3.3 for both nodes and the following packages: `python-unittest2`, `rpmdevtools`, `yum-utils` (for Fedora only), `apt-file` (for Ubuntu only).

¹Integrity Measurement Architecture: see <http://linux-ima.sourceforge.net> for details

3.11.2 Test cases

TEST CASE ID	/TC 3.11.2-1/ Verify DB data
DESCRIPTION	This test case verifies that data have been correctly inserted into the database.
TYPE	Functional test
PRECONDITIONS	The Apache Cassandra database is up and running
STEPS	<ol style="list-style-type: none"> 1. Download the following Fedora 16 packages in a temporary directory <code>f16-pkgs</code>: <ul style="list-style-type: none"> Pkg 1: <code>http://kojipkgs.fedoraproject.org/packages/curl/7.21.7/7.fc16/x86_64/curl-7.21.7-7.fc16.x86_64.rpm</code> Pkg 2: <code>http://kojipkgs.fedoraproject.org/packages/curl/7.21.7/7.fc16/x86_64/libcurl-7.21.7-7.fc16.x86_64.rpm</code> Pkg 3: <code>http://kojipkgs.fedoraproject.org/packages/curl/7.21.7/7.fc16/x86_64/libcurl-devel-7.21.7-7.fc16.x86_64.rpm</code> 2. Execute this command to insert data into the database: <pre>\$ db/scripts/update_pkgs.sh -d \$PWD/f16-pkgs -n Fedora -q 16 \ -c x86_64 -t pyunit</pre> 3. For each digest, retrieve the record stored in the database and check the following statements: <ul style="list-style-type: none"> ● <code>976a6505edeae28ccb63b491b91bce6113e87779</code> is the digest of the file <code>/usr/bin/curl</code> which belongs to Pkg 1 ● <code>8c9f2d95d80d24332139bb33da33a1340b35e1d6</code> is the digest of the file <code>/usr/lib64/libcurl.so.4.2.0</code> which belongs to Pkg 2 ● <code>f9ca79dbbab0d2d2e190904b9fe0e451a7ce901e</code> is the digest of the file <code>/usr/include/curl/curl.h</code> which belongs to Pkg 3 4. The file <code>/usr/bin/curl</code> is of type <code>executable</code> and depends on the following shared libraries: <pre>libcurl.so.4, librt.so.1, libz.so.1, libc.so.6, libpthread.so.0</pre> 5. The file <code>/usr/lib64/libcurl.so.4.2.0</code> is of type <code>library</code> and has the following aliases: <pre>libcurl.so, , libcurl.so.4</pre>

TEST CASE ID	/TC 3.11.2-2/ Test verification of sample IMA measurements files
DESCRIPTION	This test case verifies a set of sample IMA measurements files using the <i>RA Verifier</i> component and compares results obtained with those expected.
TYPE	Functional test
PRECONDITIONS	The Apache Cassandra database is up and running
STEPS	<p>1. Download these Fedora 16 packages in a temp dir <code>f16-pkgs</code>:</p> <p>Pkg 1: <code>http://kojipkgs.fedoraproject.org//packages/coreutils/8.12/7.fc16/x86_64/coreutils-8.12-7.fc16.x86_64.rpm</code></p> <p>Pkg 2: <code>http://kojipkgs.fedoraproject.org//packages/coreutils/8.12/6.fc16/x86_64/coreutils-8.12-6.fc16.x86_64.rpm</code></p> <p>Pkg 3: <code>http://kojipkgs.fedoraproject.org//packages/glibc/2.14.90/24.fc16.9/x86_64/glibc-2.14.90-24.fc16.9.x86_64.rpm</code></p> <p>Pkg 4: <code>http://kojipkgs.fedoraproject.org//packages/glibc/2.14.90/24.fc16.7/x86_64/glibc-2.14.90-24.fc16.7.x86_64.rpm</code></p> <p>2. Execute this command to insert data into the database:</p> <pre>\$ db/scripts/update_pkgs.sh -d \$PWD/f16-pkgs -n Fedora -q 16 \ -c x86_64 -t pyunit</pre> <p>3. Manually set update type into the database:</p> <pre>coreutils-8.12-7.fc16.x86_64.rpm: bugfix coreutils-8.12-6.fc16.x86_64.rpm: bugfix glibc-2.14.90-24.fc16.9.x86_64.rpm: security glibc-2.14.90-24.fc16.7.x86_64.rpm: bugfix</pre> <p>4. Generate the sample IMA measurements files as follows:</p> <p>Sample A: boot_ aggregate + digests of files from Pkgs 1, 3</p> <p>Sample B: same as above + an unknown digest</p> <p>Sample C: same as above + a digest of file from Pkg 2</p> <p>Sample D: same as above + a digest of file from Pkg 4</p> <p>5. Verify sample IMA measurements files:</p> <pre>\$ verifier/ra_verifier.py -i <sample_ima_measurements_file></pre> <p>The script should return the following output:</p> <pre>Sample A: 614 ok, 0 unknown, 0 pkg-security, 0 pkg-not-security Sample B: 614 ok, 1 unknown, 0 pkg-security, 0 pkg-not-security Sample C: 614 ok, 1 unknown, 0 pkg-security, 1 pkg-not-security Sample D: 507 ok, 1 unknown, 109 pkg-security, 0 pkg-not-security</pre> <p>where each field indicates the num of measurements of files that:</p> <ul style="list-style-type: none"> • <i>ok</i>: have a known digest and belong to the most recent package • <i>unknown</i>: have an unknown digest • <i>pkg-security</i>: belong to a package with security updates • <i>pkg-not-security</i>: belong to a package with other updates

TEST CASE ID	/TC 3.11.2-3/ Verify DB data (Ubuntu)
DESCRIPTION	This test case verifies that data from some Ubuntu packages have been correctly inserted into the database.
TYPE	Functional test
PRECONDITIONS	The Apache Cassandra database is up and running
STEPS	<ol style="list-style-type: none"> 1. Download the following Ubuntu packages in a temporary directory <code>ubuntu-pkgs</code>: <ul style="list-style-type: none"> Pkg 1: <code>http://ubuntu.mirror.cambrium.nl/ubuntu/pool/main/c/curl/curl_7.22.0-3ubuntu4_amd64.deb</code> Pkg 2: <code>http://ubuntu.mirror.cambrium.nl/ubuntu/pool/main/c/curl/libcurl3_7.22.0-3ubuntu4_amd64.deb</code> 2. Execute this command to insert data into the database: <pre>\$ db/scripts/update_pkgs.sh -d \$PWD/ubuntu-pkgs -n Ubuntu \ -q precise -c x86_64 -t pyunit</pre> 3. For each digest, retrieve the record stored in the database and check the following statements: <ul style="list-style-type: none"> • <code>ecdb5b7ee4d0a0078007f61daf7b72c33bd3b350</code> is the digest of the file <code>/usr/bin/curl</code> which belongs to Pkg 1 • <code>ea9a8957a3240b42826cabac706de50093f35647</code> is the digest of the file <code>/usr/lib/x86_64-linux-gnu/libcurl.so.4.2.0</code> which belongs to Pkg 2 4. The file <code>/usr/bin/curl</code> is of type <code>executable</code> and depends on the following shared libraries: <pre>libcurl.so.4, librt.so.1, libz.so.1, libc.so.6</pre> 5. The file <code>/usr/lib/x86_64-linux-gnu/libcurl.so.4.2.0</code> is of type <code>library</code> and has the following aliases: <pre>libcurl.so.4, libcurl.so.3</pre>

TEST CASE ID	/TC 3.11.2-4/ Verify History of Ubuntu Packages
DESCRIPTION	This test case verifies that versions of Ubuntu packages stored into the database are ordered correctly from the oldest to the most recent.
TYPE	Functional test
PRECONDITIONS	The Apache Cassandra database is up and running
STEPS	<p>1. Download the following Ubuntu packages in a temporary directory ubuntu-pkgs:</p> <p>Pkg 1: <code>http://ubuntu.mirror.cambrium.nl/ubuntu/pool/main/v/vim/vim_7.3.429-2ubuntu2_amd64.deb'</code></p> <p>Pkg 2: <code>http://ubuntu.mirror.cambrium.nl/ubuntu/pool/main/v/vim/vim_7.3.429-2ubuntu2.1_amd64.deb'</code></p> <p>Pkg 3: <code>http://ubuntu.mirror.cambrium.nl/ubuntu/pool/main/v/vim/vim_7.3.547-4ubuntu1_amd64.deb'</code></p> <p>Pkg 4: <code>http://ubuntu.mirror.cambrium.nl/ubuntu/pool/main/v/vim/vim_7.3.547-4ubuntu1.1_amd64.deb'</code></p> <p>2. Execute this command to insert data into the database:</p> <pre>\$ db/scripts/update_pkgs.sh -d \$PWD/ubuntu-pkgs -n Ubuntu -q precise -c x86_64 -t pyunit</pre> <p>3. The list of versions stored for the vim package should contain the following items in the same order:</p> <ul style="list-style-type: none"> ● 2:7.3.429-2ubuntu2 ● 2:7.3.429-2ubuntu2.1 ● 2:7.3.547-4ubuntu1 ● 2:7.3.547-4ubuntu1.1

TEST CASE ID	/TC 3.11.2-5/ Verify Shared Libraries List for Executables
DESCRIPTION	This test case verifies that the list of shared libraries related to an executable, obtained by reading from the database, the dependencies of each library and the executable itself, is the same of that displayed by the <code>ldd</code> command.
TYPE	Functional test
PRECONDITIONS	The Apache Cassandra database is up and running
STEPS	<ol style="list-style-type: none"> 1. Create two new directories for testing, called <code>tests</code> and <code>tests/tmp</code> 2. Find the ELF binaries in the <code>/usr/sbin</code> directory 3. For each binary, obtain the list of shared libraries through the <code>ldd</code> command, that will be the expected result 4. For each binary, download the package that contains the executable itself and the list of shared libraries previously obtained through the command: <pre> \$ yumdownloader --resolve --destdir tests/tmp \ --archlist=x86_64 <binary or shared library> </pre> 5. Insert data from downloaded packages into the database by executing the command: <pre> \$ db/scripts/update_pkgs.sh -d \$PWD/tests/tmp \ -n Fedora -q 16 -c x86_64 -t pyunit </pre> 6. Obtain from the database the dependencies for each shared library extracted from the ELF header 7. For each executable, retrieve its dependencies as done in the previous step and determine from them the full list of shared libraries by recursively obtaining the dependencies of previously found elements. 8. Compare the result of the previous step with shared libraries obtained by executing the <code>ldd</code> command (the test passes if both list are identical).

3.12 Ontology-based Reasoner-Enforcer

3.12.1 Test methodology/strategy

The following test cases cover the *Enforcer* part (the only delivered) of the Ontology-based Reasoner subsystem. In particular, they allow to verify, through the black-box technique, that *Extended Libvirt* (the main part of the *Enforcer*) configures correctly the Open vSwitch component. To do so, the test cases employ two sample XML configuration files to create two virtual networks and compare, after these files have been sent to Libvirt, the current configuration of the `br-backbone` switch with what is expected.

The only requirement to run the following test cases is to install the packages of *Extended Libvirt* and Open vSwitch, distributed as part of the D2.1.4/2.3.3 deliverable, by executing the following command:

```
$ sudo dpkg -i *libvirt* openvswitch-datapath-dkms openvswitch-switch
```

3.12.2 Test cases

TEST CASE ID	/TC 3.12.2-1/ Create a <i>Backbone Network</i>
DESCRIPTION	This test case verifies that <i>Extended Libvirt</i> , a component of the Ontology-based Reasoner subsystem, configures a <i>Backbone Network</i> correctly.
TYPE	Functional test
PRECONDITIONS	<i>Extended Libvirt</i> is up and running
STEPS	<ol style="list-style-type: none"> Creates the XML configuration file, named <code>net-backbone.xml</code>, for the sample <i>Backbone Network</i>: <pre> <network> <name>net-backbone</name> <bridge name="br-backbone" type="openvswitch" stp="on" delay="0" /> <tunnel> <remoteip address="192.168.122.101"/> <device name="gre-1"/> </tunnel> </network> </pre> Create and start the <i>Backbone Network</i> in <i>Extended Libvirt</i> by executing the command: <pre> \$ sudo virsh net-create net-backbone.xml </pre> Verify that the string <code>net-backbone</code> is present in the list of started networks by executing: <pre> \$ sudo virsh net-list </pre> Verify the Open vSwitch configuration by executing: <pre> \$ sudo ovs-vsctl show </pre> Verify that the output of the command executed in the previous step contains the following text: <pre> Bridge br-backbone Port br-backbone Interface br-backbone type: internal Port br-backbone-nic Interface br-backbone-nic Port "gre-1" Interface "gre-1" type: gre options: {remote_ip="192.168.122.101"} </pre>

TEST CASE ID	/TC 3.12.2-2/ Create a <i>TVD Network</i>
DESCRIPTION	This test case verifies that <i>Extended Libvirt</i> , a component of the Ontology-based Reasoner subsystem, configures a <i>TVD Network</i> correctly.
TYPE	Functional test
PRECONDITIONS	<i>Extended Libvirt</i> is up and running and the <i>Backbone Network</i> of the previous test case has been created
STEPS	<ol style="list-style-type: none"> Creates the XML configuration file, named <code>net-tvd.xml</code>, for the sample <i>TVD Network</i>: <pre> <network> <name>net-tvd</name> <bridge name="br-tvd" type="openvswitch" sourcebridge="br-backbone" stp="on" delay="0"/> <portgroup name="pgroup-tvd" default="yes"> <vlan> <tag id="1"/> </vlan> <virtualport type="openvswitch"/> </portgroup> </network> </pre> Create and start the <i>TVD Network</i> in <i>Extended Libvirt</i> by executing the command: <pre> \$ sudo virsh net-create net-tvd.xml </pre> Verify that the string <code>net-tvd</code> is present in the list of started networks by executing: <pre> \$ sudo virsh net-list </pre> Verify the Open vSwitch configuration by executing: <pre> \$ sudo ovs-vsctl show </pre> Verify that the output of the command executed in the previous step contains the following text: <pre> Bridge br-backbone Port br-backbone Interface br-backbone type: internal Port "br-tvd" tag: 1 Interface "br-tvd" type: internal Port br-backbone-nic Interface br-backbone-nic Port "gre-1" Interface "gre-1" type: gre options: {remote_ip="192.168.122.101"} </pre>

Chapter 4

Test results

Chapter Authors: Roberto Sassu (POL), Paolo Smiraglia (POL); Alexander Buerger, Norbert Schirmer (SRX); Alysson Bessani, Marcel Henrique dos Santos (FFCUL); Sören Bleikertz, Zoltan Nagy (IBM); Imad M. Abbadi, Anbang Ruad (OXFD); Johannes Behl, Klaus Stengel (TUBS); Mihai Bucicoiu, Sven Bugiel, Hugo Hideler, Stefan Nürnberger (TUDA).

As in Year 2, Activity 2 partners conducted two type of testing activities: testing of each individual subsystem and integration testing. Regarding the first, each partner revised his test plans, by modifying or adding new test cases to reflect modifications that were eventually done in the subsystems released this year. The final test plans are provided in Chapter 3.

Regarding the second activity, the efforts were concentrated on ensuring that developed subsystems were integrated correctly in the respective prototype (Trustworthy OpenStack, and TrustedInfrastructure Cloud). To achieve this goal, Activity 2 partners performed regression testing on the Trustworthy OpenStack by executing OpenStack test suites through our Jenkins infrastructure (described in Appendix A of the D2.4.2 [S⁺12a]) deliverable, so that they can ensure that the prototype is ready from the technical point of view. For TrustedInfrastructure Cloud the tests were performed manually. Then, Activity 3 partners performed validation activities on the prototype, whose specifications and results are provided respectively in the D3.3.3 [Abi13] and D3.3.4 [A⁺13] deliverables, to verify that the prototypes are working correctly from the applications developers perspective.

In the following, we provide these contributions: the results of regression testing for the Trustworthy OpenStack prototype and the results for the test cases defined for each subsystem.


4.1 Trustworthy OpenStack Prototype

One main development activity done during the third year consisted in porting the TClouds patches released for OpenStack Essex to Folsom. With small tweaking, we adapted our infrastructure to build packages and test the code of the newer OpenStack version. Further, we took advantage of the work done by OpenStack developers to verify whether porting our patches to Folsom did not introduce errors in the original code used for the Year 3 prototype.

We recall that submitting of a patch to our Code Review site (<https://review.tclouds-project.eu>) causes the execution of specific tests by Jenkins:

- **merge**: verify whether the submitted patch applies on top of the GIT branch specified;
- **pep8**: verify whether there are code style issues by using PEP8;
- **python27**: execute unit tests by using python 2.7 as a script interpreter;
- **docs**: verify whether the documentation is generated correctly;
- **selenium**: execute user interface tests with Selenium (for the Horizon service only).

In the following, we are providing the results of the tests executed for the following Trustworthy OpenStack components: Nova, Python-novaclient, Quantum, Horizon. Figures 4.1, 4.2, 4.3 and 4.4 show that all tests for the four components were completed successfully.



The screenshot displays the Jenkins CI interface for an OpenStack build. The main heading is "Build #211 (Sep 5, 2013 8:14:32 PM)". The build status is "SUCCESS". The pipeline is "check". The build was triggered by change 170,1 on the branch "tclouds-stable-folsom". The reported result is "SUCCESS". The test result shows "Test Result (no failures)".

Figure 4.1: Nova tests results.

4.1.1 LogService

The Listing 4.1 shows the output produced by the testing framework used to perform the unit tests for the LogService described in the Test Case /TC 3.10.2-1/. As reported in the Listing 4.1, all the unit tests produced a positive result.

```
./run_tests -n 1000

CUnit - A unit testing framework for C - Version 2.1-2
http://cunit.sourceforge.net/

Suite: helper test suite
Test: timeval2ascii() ...passed
Test: timeval2usec() ...passed
Test: X509_fingerprint() ...passed
Test: X509_vrfy() ...passed
Test: digest() ...passed
Test: encrypt() ...passed
Test: decrypt() ...passed
Test: hmac() ...passed
Test: b64_enc() ...passed
Test: b64_dec() ...passed
Test: pke_encrypt() ...passed
Test: pke_decrypt() ...passed
Test: sign() ...passed
Test: sign_verify() ...passed
Test: tlv_create() ...passed
Test: tlv_parse() ...passed
Test: SENTINEL ...passed
Suite: sk test suite
Test: sk_new_ctx() ...passed
```


The screenshot shows the Jenkins CI interface for an OpenStack project. The main heading is "Build #25 (Sep 6, 2013 8:14:26 PM)". The build status is "SUCCESS". The pipeline is "check". The build was triggered by change 172,1 on the branch "tclouds-2.9.0". The reported result is "SUCCESS". The build log shows four successful tests: "gate-python-novaclient-docs #25: SUCCESS", "gate-python-novaclient-merge #27: SUCCESS", "gate-python-novaclient-pep8 #25: SUCCESS", and "gate-python-novaclient-python27 #25: SUCCESS". The build was started by an anonymous user and has no failures. The page footer indicates it was generated on Sep 6, 2013 8:15:37 PM.

Figure 4.2: Python-novaclient tests results.

The screenshot shows the Jenkins CI interface for an OpenStack project. The main heading is "Build #7 (Sep 5, 2013 7:31:53 PM)". The build status is "SUCCESS". The pipeline is "check". The build was triggered by change 169,3 on the branch "tclouds-stable-folsom". The reported result is "SUCCESS". The build log shows four successful tests: "gate-quantum-docs #7: SUCCESS", "gate-quantum-merge #8: SUCCESS", "gate-quantum-pep8 #7: SUCCESS", and "gate-quantum-python27 #7: SUCCESS". The build was started by an anonymous user and has no failures. The page footer indicates it was generated on Sep 6, 2013 8:09:01 PM.

Figure 4.3: Quantum tests results.



openstack Jenkins CI

Jenkins > Horizon > gate-horizon-python27 > #116

ENABLE AUTO REFRESH

Back to Project

Status **Build #116 (Sep 5, 2013 8:24:31 PM)** [Keep this build forever](#)

Started 23 hr ago
Took 8 min 2 sec on jenkins-slave

Changes

Console Output

Edit Build Information

Delete Build

Parameters

Injected Environment Variables

Test Result

Previous Build

Triggered by change: 171,1
Branch: **tclouds-stable-folsom**
Pipeline: **check**

All builds for this change set:

- gate-horizon-docs #118: SUCCESS
- gate-horizon-merge #119: SUCCESS
- gate-horizon-pep8 #116: SUCCESS
- gate-horizon-python27 #116: SUCCESS
- gate-horizon-selenium #79: SUCCESS

Reported result: **SUCCESS**

[edit description](#)

No changes.

Started by anonymous user

Test Result (no failures)

Help us localize this page

Page generated: Sep 6, 2013 8:09:35 PM REST API Jenkins ver. 1.529

Figure 4.4: Horizon tests results.

```

Test: sk_init_ctx() ...passed
Test: sk_open_step_1() ...passed
Test: sk_open_step_2() ...passed
Test: sk_log() ...passed
Test: sk_retrieve_sessions() ...passed
Test: sk_vrfy_ctx_new() ...passed
Test: sk_verify() ...passed
Test: sk_vrfy_ctx_free() ...passed
Test: sk_close() ...passed
Test: sk_free_ctx() ...passed
Test: SENTINEL ...passed
Suite: seclog test suite
Test: seclog_new_ctx() ...passed
Test: seclog_init() ...passed
Test: seclog_open() ...passed
Test: seclog_log() ...(elapsed time: 1.343607) passed
Test: seclog_retrieve_sessions() ...passed
Test: seclog_verify() ...(elapsed time_1: 1.309146) (elapsed time_2: 0.006265) passed
Test: seclog_close() ...passed
Test: seclog_free_ctx() ...passed
Test: SENTINEL ...passed

Run Summary:
  Type      Total    Ran Passed Failed Inactive
  suites     3         3     n/a     0       0
  tests    38         38     38     0       0
  asserts 3672      3672  3672     0     n/a

Elapsed time = 28.250 seconds

```

Listing 4.1: Unit test results for the core library of the LogService

4.1.2 Remote Attestation Service

The *RA Verifier* module of the *Remote Attestation Service* subsystem has been tested according to the test plan in Chapter 3. Test cases have been implemented in a script, called `test_cases.py`, so that they can be run in an automatic way from the console. In the following, there is the output of the script,

executed at 2013-09-09.

```
runTest (__main__.IMAMeasurementsVerificationTestCase) ... ok
runTest (__main__.VerifyDBDataTestCase) ... ok
runTest (__main__.VerifyDBDataUbuntuTestCase) ... ok
runTest (__main__.VerifyDBLibDataTestCase) ... ok
runTest (__main__.VerifyHistoryUbuntuPackagesTestCase) ... ok

-----
Ran 5 tests in 139.860s

OK
```

Listing 4.2: Test results for the RA Service

As depicted in the listing above, all tests were completed successfully.

4.1.3 Cryptography as a Service

We successfully executed the tests from the test plan in Chapter 3 at 2013-09-05. The tests were ran in a manual, consecutive fashion as described in the test plan. The tests were run in our Xen development environment.

Test Case	Date	Result
/TC 3.6.2-1/	2013-09-05	passed
/TC 3.6.2-2/	2013-09-05	passed
/TC 3.6.2-3/	2013-09-05	passed
/TC 3.6.2-4/	2013-09-05	passed

In the following the output of the various steps of the performed tests is reported.

```
# xl list
Name          ID    Mem VCPUs    State    Time(s)
Domain-0      0    1023    8    r-----    46.8
Domain-T      1    1024    1    -b-----    0.1
```

Listing 4.3: CaaS test case [/TC 3.6.2-1/](#) – step #2

```
# hexdump /var/domt_pubkey.bin
00000000 0000 0100 0300 0100 0000 0c00 0000 0008
00000010 0000 0200 0000 0000 0000 0001 7688 26fa
00000020 a134 7aa5 9865 a22f 607c ec9e b4ab a30d
00000030 871a 0b54 e58a 7855 b0f5 6863 b237 269a
00000040 0743 7cda 05c1 18e4 e7ed 7a51 ed66 c08c
00000050 841d ea95 dd2d 0526 5340 fbd1 5922 52e8
00000060 b99c 319d a3fd 7e4d 6a16 140e 3838 71bd
00000070 3003 a6d8 c313 36b9 9e56 c708 e8c0 0e40
00000080 eddd 015f f223 270f 0827 367a a842 bbce
00000090 0919 e673 b72f fff2 f8cc 3fa8 b628 9e12
000000a0 38cc f67b 4b9c d83a d597 efc7 271c f21f
000000b0 fbdf 0d15 3764 ea9b 63b7 a58d 7102 45be
000000c0 6448 c66e e3cb 2ca8 37bb 290c 1b19 4c42
000000d0 8125 0f91 69df 1e9e 9235 ec27 1048 c3b2
000000e0 d954 7128 7421 5c3d afd8 544d ce73 765d
000000f0 56ad 70d9 8034 b5f7 5e74 df35 738d f983
00001000 ce98 124e ee30 8d62 ca9c d004 2139 7500
00001100 85a7 8798 803b 64c0 9db6 cbf9
000011c0
```

Listing 4.4: CaaS test case [/TC 3.6.2-1/](#) – step #3

```
# file disk.img
disk.img: Linux rev 1.0 ext2 filesystem data, UUID=77291821-bceb-4eae-9ab1-3f8a7adf1b48 (large
files)
```

Listing 4.5: CaaS test case /TC 3.6.2-2/ – step #1

```
# ./deployer.py -k domt_pubkey.bin disk.img disk.enc disk_vmcb.enc
*** TU-Darmstadt CaaS/SBS cloud deployer ***

Python: 3.2.3 (default, Oct 19 2012, 20:10:41)
[GCC 4.6.3]

STAGE 1: encrypting VM
-----
Encrypting file of 568.0MB: 0%...25%...50%...75%...100% in 29s (19.3MB/s)
=> 1163264 ESSIV-encrypted sectors written from disk.img to disk.enc

STAGE 2: encrypting symmetric key
-----
read 284 bytes from domt_pubkey.bin
(TSPI) Attempting: creating object which holds RSA pubkey...
(TSPI) Attempting: loading the RSA pubkey...
(TSPI) Attempting: creating object which holds plaintext...
(TSPI) Attempting: binding plaintext to the pubkey...
(TSPI) Attempting: extracting the ciphertext...
(TSPI) success: bind() finished. Length of ciphertext: 256 bytes
256 bytes written with RSA ciphertext to disk_vmcb.enc
graceful exit
```

Listing 4.6: CaaS test case /TC 3.6.2-2/ – steps #2 and #3

```
# scp disk.enc mihai(at)134.147.62.162:/home/mihai/xen/.
# scp disk_vmcb.enc mihai(at)134.147.62.162:/home/mihai/xen/.
```

Listing 4.7: CaaS test case /TC 3.6.2-3/ – step #1

```
# cat domain.cfg
# Kernel image file.
kernel = "/home/mihai/xen/vmlinuz"
ramdisk = "/home/mihai/xen/initramfs"

# Cryptoproxy additions.
domc = 1
domc_vmcb = "/home/mihai/xen/vmcb.enc"
memory = 128
name = "guest"
#disk = [ 'file:/home/mihai/xen/disk.img,xvda1,w' ]
#disk = [ 'file:/home/mihai/xen/disk.img,xvda1,w' ]
domc_disk = [ 'file:/home/mihai/xen/disk.enc,xvda1,w' ]
# kernel cmdline.
root = "/dev/xvda1 ro debug"
extra = "5"
```

Listing 4.8: CaaS test case /TC 3.6.2-3/ – step #2

```
# xl create domain.cfg
Parsing config file domain.cfg
do_domain_create We are creating via domT.

==== printing domain config ====
```

```
(domain
  (domid -1)
  (create_info)
  (domc 1)
  (hvm 0)
  (hap 1)
  (oos 1)
  (ssidref 0)
  (name guest)
  (cpupool Pool-0)
  (xsdata (null))
  (platformdata (null))
  (build_info)
  (max_vcpus 1)
  (tsc_mode 0)
  (max_memkb 131072)
  (target_memkb 131072)
  (nomigrate 0)
  (image
    (linux 0)
    (kernel /home/mihai/xen/vmlinuz)
    (cmdline root=/dev/xvda1 ro debug 5)
    (ramdisk /home/mihai/xen/initramfs)
  )
)
(domc image info
  (domc_vmcb /home/mihai/xen/vmcb.enc)
)
(DC device
  (DC tap
    (backend_domid 0)
    (frontend_domid 0)
    (physpath /home/mihai/xen/disk.enc)
    (phystype 2)
    (virtpath xvda1)
    (unpluggable 0)
    (readwrite 1)
    (is_cdrom 0)
  )
)
)
)
==== domain config ====

creating device model for domid 2
creating device model for domid 3
[1] Exit libxl_domain_create_new.
Daemon running with PID 3114

# xl list
Name          ID   Mem VCPUs   State   Time(s)
Domain-0      0  1023    8   r-----   75.1
Domain-T      1  1024    1   -b-----    0.4
guest         2   128    1   -b-----    2.0
guest-domc    3    64    1   -b-----    0.4

# xl console guest
[... output missing ...]
Debian GNU/Linux 7.0 CaaS hvc0

CaaS login: root (automatic login)
Last login: Tue Apr 30 14:29:44 UTC 2013 on hvc0
Linux CaaS 3.2.0-4-amd64 #1 SMP Debian 3.2.41-2 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
-bash: /setup.sh: No such file or directory
root(at)CaaS:~#
```

Listing 4.9: CaaS test case /TC 3.6.2-3/ – step #3

```
# hexdump -v -C disk.img | grep passwd
002a6040 67 70 61 73 73 77 64 2e 31 2e 67 7a 00 00 00 00 |gpasswd.1.gz....|
002a6050 20 00 15 01 67 70 61 73 73 77 64 2e 31 2e 67 7a | ...gpasswd.1.gz|
002a6070 94 0f 15 01 67 70 61 73 73 77 64 2e 31 2e 67 7a | ...gpasswd.1.gz|
002aa2a0 77 05 00 00 14 00 0c 01 67 70 61 73 73 77 64 2e |w.....gpasswd.|
002aa3b0 67 70 61 73 73 77 64 2e 31 2e 67 7a 2e 64 70 6b |gpasswd.1.gz.dpk|
002aa3d0 67 70 61 73 73 77 64 2e 31 2e 67 7a 2e 64 70 6b |gpasswd.1.gz.dpk|
002aa3f0 70 61 73 73 77 64 2e 31 2e 67 7a 2e 64 70 6b 67 |passwd.1.gz.dpkg|
002b92a0 90 05 00 00 14 00 0c 01 67 70 61 73 73 77 64 2e |.....gpasswd.|
002b93b0 67 70 61 73 73 77 64 2e 31 2e 67 7a 2e 64 70 6b |gpasswd.1.gz.dpk|
002b93d0 67 70 61 73 73 77 64 2e 31 2e 67 7a 2e 64 70 6b |gpasswd.1.gz.dpk|
002b93f0 70 61 73 73 77 64 2e 31 2e 67 7a 2e 64 70 6b 67 |passwd.1.gz.dpkg|
002d9180 cf 05 00 00 14 00 0c 01 67 70 61 73 73 77 64 2e |.....gpasswd.|
002d92a0 20 00 15 01 67 70 61 73 73 77 64 2e 31 2e 67 7a | ...gpasswd.1.gz|
002d92c0 3c 00 15 01 67 70 61 73 73 77 64 2e 31 2e 67 7a |<...gpasswd.1.gz|
002d92e0 1c 00 14 01 70 61 73 73 77 64 2e 31 2e 67 7a 2e |....passwd.1.gz.|
002d9300 70 61 73 73 77 64 2e 31 2e 67 7a 2e 64 70 6b 67 |passwd.1.gz.dpkg|
010030d0 65 63 75 72 69 74 79 2f 6f 70 61 73 73 77 64 20 |ecurity/opasswd |
020347d0 2d 70 72 6f 78 79 2d 70 61 73 73 77 64 3d 50 41 |-proxy-passwd=PA|
02080ae0 72 6f 78 79 2d 70 61 73 73 77 64 3d 48 41 53 a3 |roxy-passwd=HAS.|
020961e0 70 61 73 73 77 64 3d 50 41 53 53 20 20 20 20 20 |passwd=PASS |
020a6fd0 2d 66 74 70 2d 70 61 73 73 77 64 3d 47 45 53 4c |-ftp-passwd=GESL|
020bf7c0 62 20 67 65 74 70 61 73 73 77 64 2d 67 6e 75 20 |b getpasswd-gnu |
0210ccf0 68 74 74 70 70 61 73 73 77 64 00 68 74 74 70 70 |httppasswd.http|
0210ce80 72 6f 78 79 70 61 73 73 77 64 00 70 72 6f 78 79 |roxypasswd.proxy|
0210fdf0 74 70 2d 70 61 73 73 77 64 00 68 74 74 70 2d 70 |tp-passwd.http-p|
0210ff80 78 79 2d 70 61 73 73 77 64 00 70 72 6f 78 79 2d |xy-passwd.proxy-
```

Listing 4.10: CaaS test case /TC 3.6.2-4/ – step #1

```
# hexdump -v -C disk.enc | grep passwd
#
```

Listing 4.11: CaaS test case /TC 3.6.2-4/ – step #2

4.1.4 ACaaS

We tested *ACaaS Scheduler* with the steps described in the test plan Chapter 3 at 2013-09-06. These tests were successfully executed manually in a consecutive fashion.

Test Case	Date	Result
/TC 3.7.2-1/	2013-09-06	Passed
/TC 3.7.2-2/	2013-09-06	Passed
/TC 3.7.2-3/	2013-09-06	Passed

```
# nova-manage requirement
/usr/local/bin/nova-manage category action [<args>]
Available actions for requirement category:
    create
    list
    remove

# nova-manage requirement list
ID Reqs
1 location
```

```
# nova-manage requirement create --requirements='ids'

# nova-manage requirement list
ID Reqs
1 location
2 ids

# nova-manage requirement remove --id=2

# nova-manage requirement list
ID Reqs
1 location
```

Listing 4.12: ACaaS test case /TC 3.7.2-1/

```
# nova-manage host get_properties --host=TCloud1
Properties of host TCloud1 are:
location: us

# nova-manage host add_properties --host=TCloud1 --properties='{ids : yes}'

# nova-manage host get_properties --host=TCloud1
Properties of host TCloud1 are:
location: us
ids: yes

# nova-manage host remove_properties --host=TCloud1 --properties='ids'

# nova-manage host get_properties --host=TCloud1
Properties of host TCloud1 are:
location: us
```

Listing 4.13: ACaaS test case /TC 3.7.2-2/

```
# nova-manage requirement list
ID Reqs
1 location

# nova-manage host get_properties --host=TCloud1
Properties of host TCloud1 are:
location: us

# nova-manage host get_properties --host=TCloud2
Properties of host TCloud2 are:
location: uk

# nova boot --flavor 1 --image test_acaas --req='{1:us}' vm-us

# cat /var/log/nova/nova-scheduler.log | grep ACaaS
ACaaS: got req_list: {'location': 'us'}
ACaaS: hosts for scheduling ['TCloud1', 'TCloud2']
ACaaS: get security_properties of TCloud1: {'location': 'us'}
ACaaS: satisfy_requirements: req = {'location': 'us'}
ACaaS: adding host TCloud1 for scheduling
ACaaS: get security_properties of TCloud2: {'location': 'uk'}
ACaaS: satisfy_requirements: req = {'location': 'us'}
ACaaS: Requirement value 'us' not matched
ACaaS: filtered_hosts: ['TCloud1']
```

Listing 4.14: ACaaS test case /TC 3.7.2-3/ – step #1

```
# nova-manage requirement list
ID Reqs
1 location

# nova-manage host get_properties --host=TCloud1
Properties of host TCloud1 are:
location: us

# nova-manage host get_properties --host=TCloud2
Properties of host TCloud2 are:
location: uk

# nova boot --flavor 1 --image test_acaas --req='{1:'de'}' vm-de

# cat /var/log/nova/nova-scheduler.log | grep ACaaS
ACaaS: got req_list: {'location': 'de'}
ACaaS: hosts for scheduling ['TCloud1', 'TCloud2']
ACaaS: get security_properties of TCloud1: {u'location': u'us'}
ACaaS: satisfy_requirements: req = {'location': 'de'}
ACaaS: Requirement value 'de' not matched
ACaaS: get security_properties of TCloud2: {u'location': u'uk'}
ACaaS: satisfy_requirements: req = {'location': 'de'}
ACaaS: Requirement value 'de' not matched
ACaaS: filtered_hosts: []
```

Listing 4.15: ACaaS test case /TC 3.7.2-3/ – step #2

```
# nova boot --flavor 1 --image test_acaas --req='{x:['user_1']}' vm_excl_u1

# cat /var/log/nova/nova-scheduler.log | grep ACaaS
ACaaS: got req_list: {'_exclude-user': 'user_1'}
ACaaS: hosts for scheduling ['TCloud1', 'TCloud2']
ACaaS: get security_properties of TCloud1: {u'location': u'us'}
ACaaS: getting user name list: user_1, user_2
ACaaS: Found excluded users: user_1
ACaaS: get security_properties of TCloud2: {u'location': u'uk'}
ACaaS: getting user name list: user_2, user_3
ACaaS: adding host TCloud2 for scheduling
ACaaS: filtered_hosts: ['TCloud2']
```

Listing 4.16: ACaaS test case /TC 3.7.2-3/ – step #3

```
# nova boot --flavor 1 --image test_acaas --req='{x:['user_2']}' vm_excl_u2

# cat /var/log/nova/nova-scheduler.log | grep ACaaS
ACaaS: got req_list: {'_exclude-user': 'user_2'}
ACaaS: hosts for scheduling ['TCloud1', 'TCloud2']
ACaaS: get security_properties of TCloud1: {u'location': u'us'}
ACaaS: getting user name list: user_1, user_2
ACaaS: Found excluded users: user_2
ACaaS: get security_properties of TCloud2: {u'location': u'uk'}
ACaaS: getting user name list: user_2, user_3
ACaaS: Found excluded users: user_2
ACaaS: filtered_hosts: []
```

Listing 4.17: ACaaS test case /TC 3.7.2-3/ – step #4

4.1.5 Ontology-based Reasoner/Enforcer

The Ontology-based Reasoner/Enforcer subsystem has been tested accordingly to the test plan provided in Chapter 3. The test was done in a virtual machine with Ubuntu Precise 12.04.3 LTS with the latest version of *Extended Libvirt* and *Open vSwitch* installed from the Jenkins Server repository. The following table reports the results of the test executed on this subsystem.

Test Case	Date	Result
/TC 3.12.2-1/	2013-09-09	passed
/TC 3.12.2-2/	2013-09-09	passed

In addition, the Ontology-based Reasoner/Enforcer subsystem has been tested by executing the *Libvirt* test suite to check whether POL patches break existing functionalities. The relevant output of the `make check` command (executed at 2013-09-09) is provided below:

```

...
=====
All 188 tests passed
(23 tests were not run)
=====
...
=====
All 71 tests passed
(3 tests were not run)
=====
...

```

Listing 4.18: Output of `make check` (Extended Libvirt)

4.2 TrustedInfrastructure Cloud Prototype

We successfully tested all functional tests described in Chapter 3 within a prototypical but automatically reproducible environment.

Test Case	Date	Result
/TC 3.2.2-1/	2012-06-20	passed
/TC 3.2.2-2/	2012-06-20	passed
/TC 3.2.2-3/	2012-06-20	passed
/TC 3.2.2-4/	2012-06-20	passed
/TC 3.2.2-5/	2013-02-22	passed
/TC 3.2.2-6/	2013-02-22	passed
/TC 3.2.2-7/	2013-02-22	passed
/TC 3.2.2-8/	2013-02-22	passed

4.3 BFT-SMaRt

We successfully executed the tests described in the Chapter 3. The results are listed below

Test Case	Date	Result
/TC 3.8.2-1/	2013-09-09	passed
/TC 3.8.2-2/	2013-09-09	passed
/TC 3.8.2-3/	2013-09-09	passed
/TC 3.8.2-4/	2013-09-09	passed
/TC 3.8.2-5/	2013-09-09	passed

4.4 Resilient Object Storage (DepSky)

We successfully executed the tests described in the Chapter 3. The results are listed below

Test Case	Date	Result
/TC 3.9.2-1/	2013-09-09	passed
/TC 3.9.2-2/	2013-09-09	passed
/TC 3.9.2-3/	2013-09-09	passed

In the test /TC 3.9.2-1/ the query returned the expected value. Data read from the cloud using the cloud provided console wasn't useful.

In the test /TC 3.9.2-2/ the query returned the expected value. Data read from the cloud using the cloud provided console wasn't useful. After removing data from one of the servers the data read from the DepSky client was still the same written in the beginning.

In the test /TC 3.9.2-3/ the query returned the expected value. Data read from the cloud using the cloud provided console wasn't useful. After corrupting data from one of the servers the data read from the DepSky client was still the same written in the beginning.

4.5 Tailored Memcached

The Tailored Memcached was tested using two different scenarios. In the first one we check the deployment and general usability of the service, while in the second one we verify that the reconfiguration actually enables or disables the corresponding features as advertised.

4.5.1 Service deployment

For the first test, described in /TC 3.4.3-1/ , we deployed the Tailored Memcached on a Xen node and checked whether the service works using the standard memslap test program. A transcript with the DHCP configuration on top and the results of the memslap program on the bottom is shown in Listing 4.19.

```
00:16:3e:28:07:cc -> ff:ff:ff:ff:ff:ff
Network stack initialized.
00:16:3e:28:07:cc -> ff:ff:ff:ff:ff:ff
Bound to: 169.254.0.147
169.254.0.147

$ memslap --initial-load=100 --execute-number=100 --test=get \
--servers=169.254.0.147
  Threads connecting to servers 1
  Took 0.141 seconds to read data
```

Listing 4.19: Transcript of memslap test

The test produced the expected result.

4.5.2 Test tailoring

The test **/TC 3.4.3-1/** verifies that the tailoring process works by comparing two different variations of the service. The Listing 4.20 shows the console transcript of the test. The instance with IP 169.254.0.148 was compiled with support for the append operation, while the instance with IP 169.254.0.149 does not.

```
telnet> open 169.254.0.148 11211
Trying 169.254.0.148...
Connected to 169.254.0.148.
Escape character is '^]'.
^]

telnet> set crlf
Will send carriage returns as telnet <CR><LF>.
set key 0 9000 3
foo
STORED
append key 0 9000 3
bar
STORED
quit
Connection closed by foreign host.

telnet> open 169.254.0.149 11211
Trying 169.254.0.149...
Connected to 169.254.0.149.
Escape character is '^]'.
^]

telnet> set crlf
Will send carriage returns as telnet <CR><LF>.
set key 0 9000 3
foo
STORED
append key 0 9000 3
bar
SERVER_ERROR Command not found
quit
Connection closed by foreign host.
```

Listing 4.20: Transcript of telnet feature test

As expected, the stripped-down version does no longer support the append command.

Test Case	Date	Result
/TC 3.4.3-1/	2013-09-05	passed
/TC 3.4.3-2/	2013-09-05	passed

4.6 Fault-Tolerant BPEL

The Fault-Tolerant BPEL subsystem improves the reliability of business processes. In order to test our system, we compare the behaviour of our system with a standard implementation in different settings.

4.6.1 Fault-free operation on standard infrastructure

In the first test **/TC 3.5.2-1/**, we checked that our test setup and all the services work using a standard, unreplicated BPEL execution engine. For that purpose, a composed Web service implementing a business process based on two other Web services was continuously invoked. As shown in Listing 4.21, the system responded at a steady rate.

1	cnt	0	time	0	avg	0	min	0	max	0
2	cnt	2	time	1809005	avg	904502	min	524850	max	1284154

3 cnt	2	time	1103189	avg	551594	min	550580	max	552608
4 cnt	2	time	918232	avg	459116	min	347719	max	570512
5 cnt	3	time	1099606	avg	366535	min	349491	max	376530
6 cnt	2	time	691408	avg	345704	min	333803	max	357604
7 cnt	3	time	1124240	avg	374746	min	313103	max	492569
8 cnt	3	time	1043065	avg	347688	min	331817	max	377833
9 cnt	2	time	988903	avg	494451	min	426454	max	562448
10 cnt	3	time	954588	avg	318196	min	279979	max	353793
11 cnt	3	time	880915	avg	293638	min	276972	max	310319

Listing 4.21: Excerpt from the log of the client invoking a standard BPEL process

The test produced the expected output.

4.6.2 Fault-free operation on FT-BPEL infrastructure

In test /TC 3.5.2-2/, we ran the same business process on our replicated platform. The BPEL process should work as on the standard system, which is confirmed by the output of our test scripts, as shown in Listing 4.22.

1 cnt	0	time	0	avg	0	min	0	max	0
2 cnt	3	time	1730409	avg	576803	min	234435	max	1180931
3 cnt	2	time	627441	avg	313720	min	299943	max	327498
4 cnt	3	time	1610625	avg	536875	min	238131	max	978639
5 cnt	3	time	877224	avg	292408	min	272166	max	313658
6 cnt	4	time	1000240	avg	250060	min	224127	max	260122
7 cnt	4	time	1155592	avg	288898	min	256016	max	323967
8 cnt	3	time	907777	avg	302592	min	226311	max	389587
9 cnt	4	time	980416	avg	245104	min	224392	max	266277
10 cnt	3	time	879553	avg	293184	min	232451	max	343598
11 cnt	4	time	1107933	avg	276983	min	251387	max	308607

Listing 4.22: Excerpt from the log of the client invoking a replicated BPEL process

4.6.3 Crashed system present on standard infrastructure

Test /TC 3.5.2-3/ verifies that the standard business process is unable to operate in the presence of a crashed node. We simulated this by killing the BPEL engine executing the BPEL process after some time. The result can be seen in Listing 4.23.

30 cnt	4	time	1463083	avg	365770	min	247543	max	609679
31 cnt	3	time	980096	avg	326698	min	254185	max	466932
32 cnt	4	time	1024862	avg	256215	min	242936	max	276280
javax.xml.ws.WebServiceException: java.net.ConnectException: Connection refused									
at com.sun.xml.internal.ws.transport.http.client.HttpClientTransport.readResponseCodeAndMessage(HttpClientTransport.java:201)									
at com.sun.xml.internal.ws.transport.http.client.HttpTransportPipe.process(HttpTransportPipe.java:151)									
at com.sun.xml.internal.ws.transport.http.client.HttpTransportPipe.processRequest(HttpTransportPipe.java:83)									
at com.sun.xml.internal.ws.transport.DeferredTransportPipe.processRequest(DeferredTransportPipe.java:78)									
at com.sun.xml.internal.ws.api.pipe.Fiber.__doRun(Fiber.java:587)									
at com.sun.xml.internal.ws.api.pipe.Fiber._doRun(Fiber.java:546)									
at com.sun.xml.internal.ws.api.pipe.Fiber.doRun(Fiber.java:531)									
at com.sun.xml.internal.ws.api.pipe.Fiber.runSync(Fiber.java:428)									
at com.sun.xml.internal.ws.client.Stub.process(Stub.java:211)									
at com.sun.xml.internal.ws.client.sei.SEIStub.doProcess(SEIStub.java:124)									
at com.sun.xml.internal.ws.client.sei.SyncMethodHandler.invoke(SyncMethodHandler.java:98)									
at com.sun.xml.internal.ws.client.sei.SyncMethodHandler.invoke(SyncMethodHandler.java:78)									
at com.sun.xml.internal.ws.client.sei.SEIStub.invoke(SEIStub.java:107)									
at sun.proxy.\$Proxy20.multiplyadd(Unknown Source)									
at eu.tclouds.rbpel.demo.calculator.CalculatorProxy.invoke(CalculatorProxy.java:25)									
at eu.tclouds.rbpel.demo.ServiceProxy.invoke(ServiceProxy.java:41)									

```

    at eu.tclouds.rbpel.demo.ThroughputClientBase$Worker.invokeService(
        ThroughputClientBase.java:268)
    at eu.tclouds.rbpel.demo.ThroughputClientBase$Worker.run(ThroughputClientBase.java
        :219)
Caused by: java.net.ConnectException: Connection refused
    at java.net.PlainSocketImpl.socketConnect(Native Method)
    at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:327)
    at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:193)
    at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:180)
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:384)
    at java.net.Socket.connect(Socket.java:546)
    at java.net.Socket.connect(Socket.java:495)
    at sun.net.NetworkClient.doConnect(NetworkClient.java:178)
    at sun.net.www.http.HttpClient.openServer(HttpClient.java:409)
    at sun.net.www.http.HttpClient.openServer(HttpClient.java:530)
    at sun.net.www.http.HttpClient.parseHTTPHeader(HttpClient.java:761)
    at sun.net.www.http.HttpClient.parseHTTP(HttpClient.java:633)
    at sun.net.www.protocol.http.HttpURLConnection.getInputStream(HttpURLConnection.java
        :1162)
    at java.net.HttpURLConnection.getResponseCode(HttpURLConnection.java:397)
    at com.sun.xml.internal.ws.transport.http.client.HttpClientTransport.
        readResponseCodeAndMessage(HttpClientTransport.java:198)
    ... 17 more
33 cnt      0 time      0 avg      0 min      0 max      0
34 cnt      0 time      0 avg      0 min      0 max      0
33 cnt      0 time      0 avg      0 min      0 max      0
34 cnt      0 time      0 avg      0 min      0 max      0
35 cnt      0 time      0 avg      0 min      0 max      0
36 cnt      0 time      0 avg      0 min      0 max      0
37 cnt      0 time      0 avg      0 min      0 max      0
38 cnt      0 time      0 avg      0 min      0 max      0
39 cnt      0 time      0 avg      0 min      0 max      0
40 cnt      0 time      0 avg      0 min      0 max      0
41 cnt      0 time      0 avg      0 min      0 max      0
42 cnt      0 time      0 avg      0 min      0 max      0
43 cnt      0 time      0 avg      0 min      0 max      0
44 cnt      0 time      0 avg      0 min      0 max      0

```

Listing 4.23: Excerpt from the log of the client invoking a standard BPEL process and an induced crash at about $t = 30$

Once the BPEL Engine had become unavailable, the client was no longer able to invoke the BPEL process successfully. This is the expected result for the standard BPEL configuration.

4.6.4 Crashed system present on FT-BPEL infrastructure

Our replicated BPEL subsystem should be able to work despite a crashed node, so we checked with the test case /TC 3.5.2-4/. The console output is shown in Listing 4.24.

```

29 cnt      4 time    1024179 avg    256044 min    231907 max    276157
30 cnt      5 time    1135744 avg    227148 min    195802 max    251975
31 cnt      4 time    975627  avg    243906 min    195701 max    280056
32 cnt      3 time    808294  avg    269431 min    235970 max    287943
33 cnt      4 time    1031234 avg    257808 min    236093 max    280000
34 cnt      5 time    1196258 avg    239251 min    199497 max    276057
35 cnt      3 time    860422  avg    286807 min    275966 max    300469
36 cnt      4 time    980019  avg    245004 min    179567 max    328733
~ 2013-09-05 16:14:21.241 OP-f4fff92e-6954-43dc-9ebe-453805fe656b:
  Error while invoking proxy b842d7e7-6832-4383-813e-2346a930e9bc for
  request 717b3c7c75084489_82be096542380310/000136
  javax.xml.ws.WebServiceException:
  java.util.concurrent.ExecutionException:
  javax.xml.ws.WebServiceException:
  java.net.ConnectException: Connection refused
~ 2013-09-05 16:14:21.364 OP-f4fff92e-6954-43dc-9ebe-453805fe656b:
  Error while invoking proxy b842d7e7-6832-4383-813e-2346a930e9bc for
  request 717b3c7c75084489_82be096542380310/000137
  javax.xml.ws.WebServiceException:
  java.util.concurrent.ExecutionException:

```

```

com.sun.xml.internal.ws.client.ClientTransportException:
HTTP transport error: java.net.ConnectException: Connection refused
37 cnt      2 time    544170 avg    272085 min    232272 max    311897
38 cnt      0 time      0 avg      0 min      0 max      0
39 cnt      0 time      0 avg      0 min      0 max      0
40 cnt      0 time      0 avg      0 min      0 max      0
41 cnt      0 time      0 avg      0 min      0 max      0
42 cnt      3 time    5479356 avg    1826452 min    252600 max    4939554
43 cnt      4 time    1064764 avg    266191 min    234435 max    281887
44 cnt      4 time    1067688 avg    266922 min    239352 max    282607
45 cnt      4 time    1003485 avg    250871 min    236491 max    275720
46 cnt      3 time     904613 avg    301537 min    275201 max    325262
47 cnt      3 time     855932 avg    285310 min    272891 max    307952

```

Listing 4.24: Excerpt from the log of the client invoking a replicated BPEL process and an induced crash at about $t = 35$

As expected, the service was able to resume normal operation shortly after detecting the crashed node.

Test Case	Date	Result
/TC 3.5.2-1/	2013-09-05	passed
/TC 3.5.2-2/	2013-09-05	passed
/TC 3.5.2-3/	2013-09-05	passed
/TC 3.5.2-4/	2013-09-05	passed

4.7 SAVE Subsystem

4.7.1 Discovery

We successfully tested the discovery with a small test infrastructure running OpenStack. The underlying virtualization management is based on *libvirt*, which was already supported and tested previously in *SAVE*. Furthermore, we successfully tested and operated the discovery with a mid-sized virtualized infrastructure based on VMware that was part of a *SAVE* case-study (cf. D2.3.1 [C⁺11], Section 8.7).

4.7.2 Analysis Unit Testing

The automated unit-testing verified the successful translation of discovery data samples of different virtualization management technologies into our unified graph-based model. Figure 4.5 shows the successful test-run.

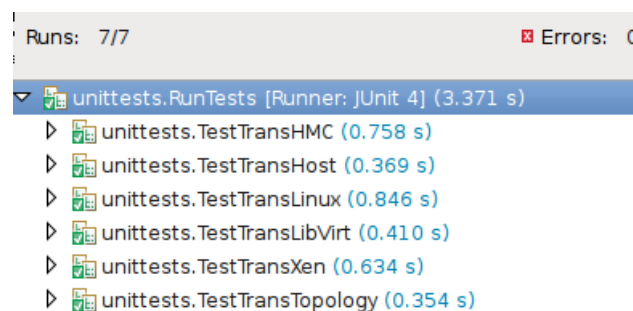


Figure 4.5: Successful JUnit Test Run.

Furthermore, we successfully tested our analysis backend that employs the Groove graph transformation tool (cf. D2.3.4, Chapter 2) as shown in Fig 4.6.

```
[info] GrooveSpec:  
[info] Groove  
[info] - should load a grammar from files  
[info] - should partially load a grammar from files and provide a host graph from memory  
[info] - should create rules programmatically  
[info] - should output the correct number of results for linear explorations on a given grammar  
[info] - should output the correct number of results for BFS explorations on a given grammar  
[info] - should match the correct subgraph upon exploring the effect of a given rule  
[info] - should convert a given model into a Groove hostgraph
```

Figure 4.6: Successful Groove Analysis Unit Test Run.

4.7.3 Analysis System Testing

We successfully tested the overall system in a case-study of a mid-sized production infrastructure (D2.3.1, Section 8.7), as well as an analysis of both a known-good and known-bad infrastructure that *SAVE* identified as such (D2.3.2, Section 4.7).

Part III

Appendices

Appendix A

Software details of the prototypes

Chapter Author: Gianluca Ramunno (POL)

The subsystems for single cloud of TClouds Platform v2 have been delivered as companion tarball(s) of D2.1.4-D2.3.3 [BS⁺13]. The document part of such deliverable collects the instructions for installing/-configuring/using the related subsystems. The present deliverable, instead, collects all subsystems developed in Activity 2 throughout the project, forming the TClouds Platform v2.1. The related documentation is either the one included in D2.1.4-D2.3.3 [BS⁺13] or, if not included or not up-to-date anymore, is included in the corresponding tarballs.

Appendix B

Subsystems' code availability

Chapter Author: Gianluca Ramunno (POL)

Table B.1 reports the code availability for each subsystem.

TClouds subsystem	Code availability
Resource-efficient BFT (CheapBFT)	Source code in D2.4.3 tarball (*)
Simple Key/Value Store (Tailored memcached)	Source code in D2.4.3 tarball (*) (**)
Cryptography as a Service (CaaS)	Object code in D2.4.3 tarball (*)
TrustedServer	Confidential
Log Service	Source code in D2.4.3 tarball (*) (***)
State Machine Replication (BFT-SMaRt)	Source code in D2.4.3 tarball (*) (****)
Fault-tolerant Workflow Execution (FT-BPEL)	Source code in D2.4.3 tarball (*) (**)
Resilient Object Storage (DepSky)	Source code in D2.4.3 tarball (*) (*****)
Confidentiality Proxy for S3	Confidential
Access Control as a Service (ACaaS)	Source code in D2.4.3 tarball (*)
TrustedObjects Manager (TOM)	Confidential
Trusted Management Channel	Confidential
Ontology-based Reasoner/Enforcer (Enforcer)	Source code in D2.4.3 tarball (*) (***)
Automated Validation (SAVE)	Confidential
Remote Attestation Service	Source code in D2.4.3 tarball (*) (***)
Cloud-of-Clouds File System (C2FS)	Source code in D2.4.3 tarball (*)
Fault-tolerant Relational DB (SteelDB)	Source code in D2.4.3 tarball (*)
Key-Value Store (KV Store)	Source code in D2.4.3 tarball (*)

(*) D2.4.3 tarball also includes binary packages for Ubuntu 12.04.3 LTS Linux distribution generated from source code and also source code of the original OpenStack and Open Attestation, existing external software. They have been included for ease of installation (and for rebuilding packages from source code, if wanted). The packages for OpenStack (also including TClouds patches) have been automatically generated by the Jenkins platform (see Section 4.5 and Appendix A of D2.4.2 [S⁺12a]). For details and installation instructions see the deliverable D2.1.4-D2.3.3 [BS⁺13] and the README file included in the root folder of the tarball.

(**) Also available from

<http://www.ibr.cs.tu-bs.de/projects/tclouds/download/>

(***) Also available from <http://security.polito.it/tclouds/>

(****) Also available from <http://code.google.com/p/bft-smart>

(*****) Also available from <http://code.google.com/p/depsky/>

Table B.1: List of TClouds subsystems and code availability

Appendix C

Evolution of TClouds platform and integration of subsystems

Chapter Author: Gianluca Ramunno (POL)

The concept of *platform* in TClouds evolved throughout the project towards the result described in this document, particularly in Chapter 2. There has been a parallel evolution of the pieces of the platform developed by the partners and called *subsystems*: the set originally conceived in Year 1 evolved from 15 to 18 subsystems in Year 3. They were developed independently during Year 1 and mostly integrated in Year 2 into three *prototypes* (Trustworthy OpenStack, TrustedInfrastructure Cloud and Cloud-of-Clouds) framed in a single scenario called TClouds platform TClouds v1.

During Year 3 the concept of TClouds platform has been improved from the original plan written in Annex I – a fully integrated platform with a unified API – to an overall richer framework, the TClouds platform v2.1, offering both Infrastructure (IaaS) as well as Platform (PaaS) services. As in commercial cloud-service ecosystems (e.g. Amazon Web Services, Windows Azure, . . .), each component is a building block that can be selected according to the applications' needs. In particular, all TClouds subsystems in the platform v2.1 are arranged in logical layers. The evolution of the two *prototypes* presented in Year 2 (Trustworthy OpenStack and TrustedInfrastructure Cloud) consists now of two different secure alternatives (with respect to the commodity clouds) for the IaaS layer. With these two alternative infrastructure prototypes we can cover the needs of a wide range of application scenarios from private or community clouds with high security demands to large-scale public clouds.

The evolution of the third prototype presented in Year 2 (Cloud-of-Clouds) is the new Cloud-of-Clouds File System (C2FS) subsystem standing at the Services layer. All other subsystems are arranged partly in the Middleware layer, mostly acting as building blocks for the remaining subsystems standing at the upper Services layer. The Middleware and Services layers together resemble to the PaaS layer as defined by NIST [MG11]: the subsystems standing on these layers are platform *components* that can be used by the applications. Therefore the TClouds Platform v2.1 offers to the applications a set of *prototypes* and *components* to be tailored according to their needs: an application can be built on top of one selected IaaS infrastructure and can use a selection of the subsystems at the PaaS layer according to its security requirements. The validity of this approach is actually demonstrated through the two Activity 3 benchmark scenarios (Workpackage 3.1 Home Healthcare and Workpackage 3.2 Smart Lighting System) that will be presented at the final review running on two tailored instantiations of the TClouds platform v2.1.

The evolution of the subsystems throughout the project, including their positioning in the TClouds Platform and their usage within the two benchmark scenarios, is thoroughly described in Table C.1.

TClouds subsystem	Year 2	Year 3	
	TClouds Platform v1	TClouds Platform v2.x	
	TClouds prototype	TClouds prototype	TClouds Platform layer [1]
Resource-efficient BFT (CheapBFT)	Trustworthy OpenStack	[platform component]	Middleware (PaaS)
Simple Key/Value Store (tailored memcached)	N/A (but planned)	[platform component]	Services (PaaS)
Secure Block Storage (SBS) [2]			
Secure VM Instances [2]			
Cryptography as a Service (CaaS) [2]	Trustworthy OpenStack	Trustworthy OpenStack	Infrastructure (IaaS)
TrustedServer	TrustedInfrastructure Cloud	TrustedInfrastructure Cloud	Infrastructure (IaaS)
Log Service	Trustworthy OpenStack	Trustworthy OpenStack	Infrastructure (IaaS) Service (PaaS) [8]
State Machine Replication (BFT-SMaRt)	Cloud-of-Clouds [3]	[platform component]	Middleware (PaaS)
Fault-tolerant Workflow Execution (FT-BPEL)	N/A (but planned)	[platform component]	Middleware (PaaS)
Resilient Object Storage (DepSky)	Cloud-of-Clouds [3]	[platform component]	Middleware (PaaS)
Confidentiality Proxy for S3	N/A (but planned)	[platform component] [7] TrustedInfrastructure Cloud	Services (PaaS)
Access Control as a Service (ACaaS)	Trustworthy OpenStack	Trustworthy OpenStack	Infrastructure (IaaS)
TrustedObjects Manager (TOM)	TrustedInfrastructure Cloud	TrustedInfrastructure Cloud	Infrastructure (IaaS)
Trusted Management Channel	TrustedInfrastructure Cloud	TrustedInfrastructure Cloud	Infrastructure (IaaS)
Ontology-based Reasoner/Enforcer	[standalone]	Trustworthy OpenStack	Infrastructure (IaaS)
Automated Validation (SAVE)	[standalone]	Trustworthy OpenStack	Infrastructure (IaaS)
Remote Attestation Service	Trustworthy OpenStack (not originally planned) [6]	Trustworthy OpenStack	Infrastructure (IaaS)
Cloud-of-Clouds File System (C2FS) [3]	N/A (not originally planned)	[platform component]	Services (PaaS)
Fault-tolerant Relational DB (SteelDB) [4]	N/A (not originally planned)	[platform component]	Services (PaaS)
Key-Value Store (KV Store) [5]	N/A (not originally planned)	[platform component]	Services (PaaS)

[1] “Layer” to be intended according to the TClouds Platform architecture described in Section 2.4 and depicted in Figure 2.3.

[2] Secure Block Storage (SBS) and Secure VM Instances during the second year have been combined to form Cryptography as a Service.

[3] Year 2 Cloud-of-Clouds prototype evolved into the Year 3 new C2FS subsystem, built on top of BFT-SMaRt and DepSky subsystems.

[4] Built on top of BFT-SMaRt.

[5] Built on top of BFT-SMaRt.

[6] It was added in substitution of the “reasoner” part of the Ontology-based Reasoner/Enforcer

[7] Confidentiality Proxy for S3 is strictly speaking one Service, but it is integrated in TrustedInfrastructure Cloud as for it concerns the transparent encryption setup within a TVD.

[8] Log Service is used at the Infrastructure layer in Trustworthy OpenStack but it can be used as Service for applications.

Table C.1: List of TClouds subsystems and their evolution in the frame of TClouds Platform

Appendix D

Low-level APIs

Companion document of this deliverable is the report R2.4.2.4 [S⁺13b] that includes the low-level APIs of the Activity 2 subsystems; because of the document size, such report is delivered as a separate zip archive containing PDF files to be printed only if necessary.

Bibliography

- [A⁺13] Marco Abitabile et al. TClouds – D3.3.4 Final Report On Evaluation Activities. Deliverable D3.3.4, TClouds Consortium, October 2013.
- [Abi13] Marco Abitabile. TClouds – D3.3.3 Validation Protocol and Schedule for the Smart Power Grid and Home Health Use Cases. Deliverable D3.3.3-v2, TClouds Consortium, May 2013.
- [B⁺13a] Alysson Bessani et al. TClouds – D2.2.4 Adaptive Cloud-of-Clouds Architecture, Services and Protocols. Deliverable D2.2.4, TClouds Consortium, September 2013.
- [B⁺13b] Sören Bleikertz et al. TClouds – D2.3.4 Automation and Evaluation of Security Configuration and Privacy Management. Deliverable D2.3.4, TClouds Consortium, September 2013.
- [BAB⁺12] Alysson Bessani, Imad M. Abbadi, Sven Bugiel, Emanuele Cesena, Mina Deng, Michael Grone, Ninja Marnau, Stefan Nurnberger, Marcelo Pasin, and Norbert Schirmer. Privacy and Resilience for Internet-scale Critical Infrastructures, 2012.
- [BACF08] Alysson N. Bessani, Eduardo P. Alchieri, Miguel Correia, and Joni S. Fraga. DepSpace: a Byzantine fault-tolerant coordination service. In *Proc. of the 3rd ACM European Systems Conference – EuroSys’08*, pages 163–176, April 2008.
- [BBI⁺13] Sören Bleikertz, Sven Bugiel, Hugo Ideler, Stefan Nürnberger, and Ahmad-Reza Sadeghi. Client-controlled Cryptography-as-a-Service in the Cloud. In *11th International Conference on Applied Cryptography and Network Security (ACNS 2013)*, June 2013.
- [BCQ⁺13] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando Andre, and Paulo Sousa. DepSky: Dependable and secure storage in cloud-of-clouds. *ACM Transactions on Storage*, 2013.
- [BDH⁺12] Johannes Behl, Tobias Distler, Florian Heisig, Rüdiger Kapitza, and Matthias Schunter. Providing Fault-tolerant Execution of Web-service-based Workflows within Clouds. In *Proceedings of the 2nd International Workshop on Cloud Computing Platforms (CloudCP ’12)*, 2012.
- [BGJ⁺05] Anthony Bussani, John Linwood Griffin, Bernhard Jansen, Klaus Julisch, Genter Karjoth, Hiroshi Maruyama, Megumi Nakamura, Ronald Perez, Matthias Schunter, Axel Tanner, and et al. Trusted virtual domains: Secure foundations for business and it services. *Science*, 23792, 2005.
- [BGSE11] Sören Bleikertz, Thomas Groß, Matthias Schunter, and Konrad Eriksson. Automated information flow analysis of virtualized infrastructures. In *Proc. of ESORICS’11*, 2011.
- [BPN⁺11] Sven Bugiel, Thomas Pöppelmann, Stefan Nürnberger, Ahmad-Reza Sadeghi, and Thomas Schneider. Amazonia: When elasticity snaps back. In *18th ACM Conference on Computer and Communications Security*. ACM, Oct 2011.
- [BS⁺13] Sören Bleikertz, Norbert Schirmer, et al. TClouds – D2.1.4/D2.3.3 Proof of Concept Infrastructure / Implementation of Security Configuration and Policy Management. Deliverable D2.1.4/D2.3.3, TClouds Consortium, April 2013.

- [BSF⁺13] Alysson Bessani, Marcel Santos, Joao Felix, Nuno Neves, and Miguel Correia. On the efficiency of durable state machine replication. In *Proc. of USENIX ATC'13*, 2013.
- [C⁺11] Christian Cachin et al. D2.3.1 - Requirements, Analysis, and Design of Security Management. Technical report, TClouds Consortium, October 2011. TClouds deliverable.
- [CRS⁺11] Emanuele Cesena, Gianluca Ramunno, Roberto Sassu, Davide Vernizzi, and Antonio Lioy. On scalability of remote attestation. In *Proc of the ACM STC '11*. ACM, Dec 2011.
- [CS11] C. Cachin and M. Schunter. A cloud you can trust. *IEEE Spectrum*, 48, 2011.
- [D⁺13] Mina Deng et al. TClouds – D3.1.5 Proof of concept for home healthcare. Deliverable D3.1.5, TClouds Consortium, October 2013.
- [FR11] Miguel Correia Francisco Rocha, Salvador Abreu. The final frontier: Confidentiality and privacy in the cloud. *IEEE Computer*, 44(9), 2011.
- [Gel85] David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.
- [GRP11] Rui Garcia, Rodrigo Rodrigues, and Nuno Preguiça. Efficient middleware for Byzantine fault tolerant database replication. In *EuroSys'11*, 2011.
- [GVM00] Garth A. Gibson and Rodney Van Meter. Network attached storage architecture. *Communications of the ACM*, 43(11):37–45, November 2000.
- [KS12] Anil Kumar and Jerry St.Clair. A Unit Testing Framework for C. Retrieved from: <http://cunit.sourceforge.net/>, September 2012.
- [Lev12] Peter Levert. FUSE-J: A Java binding for FUSE. Retrieved from: <http://sourceforge.net/projects/fuse-j/>, 2012.
- [MG11] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. *NIST Special Publication 800-145*, September 2011.
- [Mil12] Stewart Miles. A lightweight library to simplify and generalize the process of writing unit tests for C applications. Retrieved from: <http://code.google.com/p/cmocker/>, September 2012.
- [MMHJ11] M. Mahjoub, A. Mdhaffar, R.B. Halima, and M. Jmaiel. A comparative study of the current cloud computing technologies and offers. In *Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on*, pages 131–134, 2011.
- [MT09] Di Ma and Gene Tsudik. A new approach to secure logging. *Trans. Storage*, 5:2:1–2:21, March 2009.
- [NF12] Gergely Nagy and Zoltán Fried. CEE-enhanced syslog() API. Retrieved form: <https://github.com/deirf/libumberlog>, September 2012.
- [OvT12] Open vSwitch Team. Open vSwitch. Retrieved form: <http://openvswitch.org/>, September 2012.
- [Per13] Nuno Pereira. TClouds – D3.2.5 Smart Lighting System Final Report. Deliverable D3.2.5, TClouds Consortium, October 2013.
- [PVP12] Dana Petcu and Jos Luis Vzquez-Poletti. *European Research Activities in Cloud Computing*. Cambridge Scholars Publishing, United Kingdom, 2012.

- [R. 12] R. Kapitza et. al. CheapBFT: resource-efficient Byzantine fault tolerance. In *EuroSys'12*, 2012.
- [R⁺13] Gianluca Ramunno et al. TClouds – D2.4.3 Final Reference Platform and Test Case Specification. Deliverable D2.4.3, TClouds Consortium, October 2013.
- [S⁺12a] Roberto Sassu et al. TClouds – D2.4.2 Initial Component Integration, Final API Specification, and First Reference Platform. Deliverable D2.4.2, TClouds Consortium, October 2012.
- [S⁺12b] Norbert Schirmer et al. TClouds – D2.1.2 Preliminary Description of Mechanisms and Components for Single Trusted Clouds. Deliverable D2.1.2, TClouds Consortium, September 2012.
- [S⁺13a] Norbert Schirmer et al. TClouds – D2.1.5 Final Report on Requirements, Architecture, and Components for Single Trusted Clouds. Deliverable D2.1.5, TClouds Consortium, September 2013.
- [S⁺13b] Norbert Schirmer et al. TClouds – R2.4.2.4 Final low-level architecture and private interfaces specification. Internal Report R2.4.2.4, TClouds Consortium, July 2013.
- [Sch90] Fred B. Schneider. Implementing fault-tolerant service using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.
- [SK99a] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Information Systems Security*, 2(2):159–176, 1999.
- [SK99b] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2:159–176, May 1999.
- [SK13] Matthias Schunter and Klaus-Michael Koch. TClouds – Privacy and Resilience for Internet-scale Critical Infrastructures. Technical Annex I - "Description of Work" - v4 Grant Agreement No. 257243, Technikon GmbH, Villach, AT, March 2013.
- [VBP12] Paulo Verissimo, Alysson Bessani, and Marcelo Pasin. The TClouds architecture: Open and resilient cloud-of-clouds computing. In *Proc. 2nd Int. Workshop on Dependability of Clouds, Data Centers and Virtual Computing Environments (DCDV'12)*, 2012.
- [VS13] Paulo Viegas and Paulo Santos. TClouds – D3.2.4 Smart Lighting System Final Prototype. Deliverable D3.2.4, TClouds Consortium, September 2013.