D2.4.2
Initial Component Integration, Final API Specification, and First Reference Platform

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

Deliverable type:	Deliverable
Deliverable reference number:	ICT-257243 / D2.4.2 / 1.1
Activity and Work package contributing to deliverable:	Activity 2 / WP 2.4
Due date:	September 2012 – M24
Actual submission date:	30th October, 2012

Responsible organisation:	POL
Editor:	Roberto Sassu
Dissemination level:	Public
Revision:	1.1

Abstract:	This deliverable includes and reports three prototypes, outcome of the first round of integration of subsystems developed within the other Activity 2 workpackages.

Keywords:	Legal and application requirements, subsystems, prototypes, trustworthy infrastructure, private and public clouds, TClouds native and commodity clouds

Editor

Roberto Sassu (POL)

Contributors

Roberto Sassu, Paolo Smiraglia, Gianluca Ramunno (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)

Sven Bugiel, Hugo Hideler, Stefan Nürnberger (TUDA)

Ninja Marnau (ULD)

Mina Deng, Zheyi Rong (PHI)

Miguel Areias, Nuno Emanuel Pereira (EDP)

Paulo Santos (EFACEC ENG)

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 hosting critical infrastructures internally are cautious with regards to moving them to clouds, because the latter still experience security and privacy breaches.

The TClouds project aims at facilitating the shift of computing paradigm also 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.

This deliverable is a compendium of the work done in workpackages 2.1, 2.2. and 2.3 of the TClouds project. A subset of the subsystems conceived, designed, and developed in those workpackages, has been integrated into three different prototypes. Therefore, the prototypes documented in this deliverable represent the first round of integration that took place during the second year. A more comprehensive integration will be performed during the third year of the project. However, this deliverable already gives an overall view of how the project results can be used can be by combining the presented prototypes. In particular, a mixed scenario of private-public clouds is presented as subject of the demonstration for the second year review. The private cloud is a TClouds native cloud that can be implemented either using existing cloud platforms properly enhanced for security (e.g. the prototype Trustworthy OpenStack) or a platform developed with native support for security (e.g., the prototype TrustedInfrastructure Cloud). The public clouds are commodity clouds used together to guarantee the availability and integrity of data through the Cloud-of-Clouds prototype.

Subsystems and their integration in prototypes have 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. This deliverable reports such requirements and how TClouds subsystems and prototypes satisfy them.

This deliverable is organized in three parts. Part I describes the three prototypes that will be demonstrated as the result of the second year, also including the test plans and results. Part II includes the documentation of the prototypes and subsystems being part of this deliverable. Finally, Part III describes the TClouds Infrastructure for testing and delivering the subsystems being part of Trustworthy OpenStack prototype, the code availability for all subsystems and shows some screenshots of the enhanced Dashboard of Trustworthy OpenStack.

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 2.4.2 — Initial Component Integration, Final API Specification, and First Reference Platform . . . 3

I TClouds Year 2 Demo **6**

2 TClouds Infrastructure Requirements **7**
2.1 Legal Requirements . . . 7
2.2 Application Requirements . . . 8
2.2.1 Healthcare Application . . . 8
2.2.2 Smart Lighting Application . . . 10

3 Prototypes **11**
3.1 Trustworthy OpenStack Prototype . . . 12
3.1.1 Overview . . . 14
3.1.2 Demo Storyline . . . 16
3.2 TrustedInfrastructure Cloud Prototype . . . 32
3.2.1 Architecture Overview . . . 33
3.2.2 Demo Storyline . . . 34
3.3 Cloud-of-Clouds Prototype . . . 35
3.3.1 Architecture overview . . . 36
3.3.2 Demo storyline . . . 37
3.4 Other Prototypes . . . 39
3.4.1 Security Assurance of Virtualized Environments *(SAVE)* . . . 40
3.4.2 Ontology-based reasoner: Libvirt With Trusted Virtual Domains . . . 41
3.5 Mapping legal and application requirements to subsystems and prototypes . . . 46

4 Tests Plan and Results Report **58**
4.1 Introduction . . . 58
4.2 A model for testing . . . 58
4.3 Master test plan . . . 59
4.3.1 Testing environment . . . 60
4.3.2 Testing levels . . . 60
4.3.3 Testing activities workflow . . . 61
4.3.4 Test results evaluation and exit criteria . . . 62
4.4 Test plans for subsystems/prototypes . . . 65
4.4.1 TrustedInfrastructure Cloud . . . 65

4.4.2 Security Assurance of Virtualized Environments *(SAVE)* . . . 69
4.4.3 Resource-efficient BFT (CheapBFT) . . . 70
4.4.4 Secure Block Storage . . . 74
4.4.5 Access Control as a Service (ACaaS) . . . 76
4.4.6 BFT-SMaRt . . . 78
4.4.7 Resilient Object Storage (DepSky) . . . 81
4.4.8 LogService . . . 84
4.4.9 Remote Attestation Service . . . 87
4.5 Jenkins server . . . 90
4.5.1 Subsystem setup . . . 90
4.6 Tests Results . . . 91
4.6.1 Trustworthy OpenStack Prototype . . . 91
4.6.2 TrustedInfrastructure Cloud Prototype . . . 95
4.6.3 Cloud-of-Clouds Prototype . . . 96
4.6.4 SAVE Subsystem . . . 96

II Prototypes Documentation **98**

5 Trustworthy OpenStack Prototype **99**
5.1 LogService . . . 100
5.1.1 Platform Setup . . . 100
5.1.2 LogService Subcomponents . . . 100
5.2 Remote Attestation Service . . . 102
5.2.1 Operating Environment Setup . . . 102
5.2.2 Prototype Build and Installation Instructions . . . 103
5.2.3 Prototype Execution Instructions . . . 105
5.3 Access Control as a Service . . . 106
5.3.1 Platform Setup . . . 106
5.3.2 Management Console . . . 110
5.4 Cryptography-as-a-Service (Caas) . . . 112
5.4.1 Operating Environment Setup . . . 112
5.4.2 Prototype Execution Instructions . . . 113
5.5 Resource-efficient BFT (CheapBFT) . . . 114
5.5.1 Operating Environment Setup . . . 114
5.5.2 Prototype Execution Instructions . . . 116

6 TrustedInfrastructure Cloud Prototype **117**
6.1 TrustedObjectsManager setup . . . 117
6.1.1 Using the management console . . . 117
6.1.2 Creating a company . . . 117
6.1.3 Creating a location . . . 117
6.1.4 Adding users . . . 119
6.1.5 Network configuration . . . 119
6.1.6 Creating and configuring VPNs . . . 121
6.1.7 Attaching appliances . . . 121
6.1.8 Creating TrustedVirtualDomains . . . 123
6.1.9 Adding compartments to TVDs . . . 123

6.1.10 Connecting everything together . . . 124

7 Cloud-of-Clouds Prototype **128**
7.1 BFT-SMaRt . . . 128
7.1.1 Download instructions . . . 128
7.1.2 How to install . . . 128
7.2 Resilient Object Storage (DepSky) . . . 129
7.2.1 Prototype Execution Instructions . . . 129
7.2.2 DepSky configuration . . . 129
7.2.3 Running DepSky locally . . . 129

8 Other Prototypes **130**
8.1 Security Assurance of Virtualized Environments *(SAVE)* . . . 130
8.1.1 Operating Environment Setup . . . 130
8.1.2 Prototype Build and Installation Instructions . . . 130
8.1.3 Prototype Execution Instructions . . . 130
8.2 Ontology-based reasoner: Libvirt With Trusted Virtual Domains . . . 131
8.2.1 Operating Environment Setup . . . 131
8.2.2 Prototype Build and Installation Instructions . . . 131
8.2.3 Prototype Execution Instructions . . . 131

III Appendices **132**

A TClouds Infrastructure Wiki **133**
A.1 Infrastructure Overview . . . 133
A.1.1 Code Repositories (git.tclouds-project.eu) . . . 133
A.1.2 Code Review (review.tclouds-project.eu) . . . 134
A.1.3 Testing Framework (jenkins.tclouds-project.eu) . . . 137

B Subsystems' code availability **138**

C Trustworthy OpenStack Dashboard screenshots **139**

Bibliography **139**

List of Figures

1.1 Graphical structure of WP2.4 and relations to other work packages. 3

3.1 Prototypes demo architecture and scenario 13
3.2 The Trustworthy OpenStack demo architecture 15
3.3 Remote Attestation Service demo workflow 18
3.4 ACaaS demo workflow 21
3.5 SBS Modules Overview 23
3.6 Secure Block Storage demo workflow 25
3.7 LogService demo workflow 27
3.8 Resilient Log demo workflow (CheapTiny protocol) 30
3.9 Resilient Log demo workflow (CheapSwitch protocol) 31
3.10 Resilient Log demo workflow (MinBFT protocol) 31
3.11 TrustedInfrastructure architecture 33
3.12 Start/Stop compartment demo workflow 35
3.13 C2FS basic architecture. 36
3.14 The flow of invocations in different components for directory-related operations. 39
3.15 The flow of invocations in different components when data-intensive commands are executed. 39
3.16 SAVE: Architecture Overview of Discovery Component 40
3.17 SAVE: Architecture Overview of Analysis Component 41
3.18 TVD scenario implemented using Open vSwitch bridges 42

4.1 The V-model 59
4.2 Testing activities workflow 63
4.3 TClouds Jenkins Web page 90
4.4 Jenkins Tests Results for Build#39 (OpenStack + ACaaS Scheduler) 92
4.5 Jenkins Code Style Tests Results (OpenStack + ACaaS Scheduler) 93
4.6 Successful JUnit Test Run. 96

6.1 The TrustedObjectsManager Login screen 118
6.2 The TrustedObjectsManager overview screen after login 118
6.3 Creating a “Company” 118
6.4 Creating a “Location” 119
6.5 Adding a user, step 1 119
6.6 Adding a user, step 2 120
6.7 Adding networks 120
6.8 Adding a VPN 121
6.9 Add a new appliance to the company 122
6.10 Dialog to download the configuration for the specific appliance 122
6.11 Dialog to create a new TVD 123
6.12 Adding a new compartment 124
6.13 Installing a registered compartment to TrustedServer 124

6.14 Attaching networks to compartments installed on TrustedServer 125
6.15 Editing VPN membership of TrustedServer 126

C.1 The Trustworthy OpenStack Dashboard - Login 140
C.2 Trustworthy OpenStack Dashboard - (ACaaS) Requirements and Security Properties 140
C.3 Trustworthy OpenStack Dashboard - (Remote Attestation/ACaaS) Setting Extra Specs with flavours 141
C.4 Trustworthy OpenStack Dashboard - (Remote Attestation/ACaaS) Launching an instance and setting the requirements 141
C.5 Trustworthy OpenStack Dashboard - (LogService) List of available logging sessions 142
C.6 Trustworthy OpenStack Dashboard - (LogService) Log file dump with verification results 142

List of Tables

3.1 List of TClouds subsystems and mapping to prototypes 12
3.2 List of TClouds subsystems and mapping to requirements 47

B.1 List of TClouds subsystems and code availability 138

Chapter 1

Introduction

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.

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.

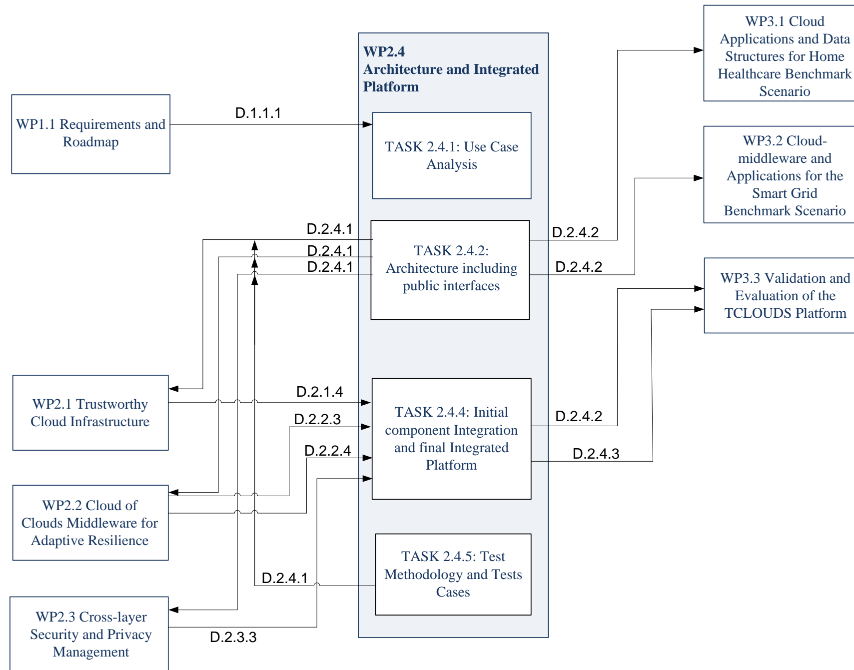

Figure 1.1: Graphical structure of WP2.4 and relations to other work packages.

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 2.4.2 — Initial Component Integration, Final API Specification, and First Reference Platform

Overview. Cloud computing is an emerging technology devoted to outsource IT infrastructures, from SME needs to large-scale computing and storage. However, organizations hosting critical infrastructures internally are cautious with regards to moving them to clouds, because the latter 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.

This deliverable is a compendium of the work done in workpackages 2.1, 2.2 and 2.3. A subset of the subsystems conceived, designed, and developed in those workpackages, has been integrated into three different prototypes. These prototypes represent the first round of integration that took place during the second year. A more comprehensive integration will be performed during the third year of the project. However, this deliverable already gives an overall

view of how the project results can be used by combining the presented prototypes. In particular a mixed scenario of private-public clouds is presented as subject of the demonstration for the second year review. The private cloud is a TClouds native cloud that can be implemented either using existing cloud platforms properly enhanced for security (e.g. the prototype Trustworthy OpenStack) or a platform developed with native support for security (e.g. the prototype TrustedInfrastructure Cloud). The public clouds are commodity clouds used together to guarantee the availability and integrity of data through the Cloud-of-Clouds prototype.

Subsystems and their integration in prototypes have 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. This deliverable reports such requirements and how TClouds subsystems and prototypes satisfy them.

Structure. This deliverable is organized in three parts.

Part I describes the prototypes that will be demonstrated as the results of the second year and consists of three chapters. Chapter 2 reports legal and application requirements from Activity 1 and Activity 3. Chapter 3 is the core of this deliverable describing the three prototypes that will be demonstrated and two more being part of this deliverable but not demonstrated. Chapter 3 also includes (at the beginning) the list of all subsystems defined in D2.4.1 [ea11c] during the first year plus a new subsystem introduced during the second year and (at the end) the mapping between the requirements stated in Chapter 2 and the subsystems and prototypes that satisfy them. Chapter 4 reports the tests plans for the subsystems being part of the prototypes described in Chapter 3 and the test results grouped by prototypes.

Part II includes the documentation of the prototypes being part of this deliverable and consists of four chapters. Chapter 5 documents the Trustworthy OpenStack Prototype, Chapter 6 documents the TrustedInfrastructure Cloud Prototype, Chapter 7 documents the Cloud-of-Clouds Prototype, and Chapter 8 documents the additional prototypes being part of this deliverable but not demonstrated.

Part III contains Appendix A describing the TClouds Infrastructure for testing and delivering the subsystems being part of Trustworthy OpenStack prototype, Appendix B reports the code availability for all subsystems, and Appendix C collects some screenshots of the enhanced Dashboard of Trustworthy OpenStack.

Deviation from Workplan. This deliverable follows the workplan in the DoW/Annex I apart from the inclusion of the final API. The latter was finalized through an internal report (R2.4.2.3, Final API specification), but it includes only few minor updates to the API already reported in D2.4.1. For this reason it has been decided not to include the API in the present deliverable.

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

The requirements were mainly collected in D1.1.1 [GHSS11], used to define the use cases reported in D2.4.1. The architecture and the interfaces defined in D2.4.1 are reported back to and used by WPs 2.1-2.3 to develop their subsystems that will be delivered as D2.1.4,

D2.2.3 [ea11a], and D2.3.3 that become then the input for this work package. The architecture and the interfaces are carried to WPs 3.1 and 3.2 to design their applications through this deliverable. The first integrated platform, output of this work package through this deliverable, is the input for WP3.3 to be evaluated and validated.

Part I

TClouds Year 2 Demo

Chapter 2

TClouds Infrastructure Requirements

The aim of the TClouds project is to provide a secure platform that addresses the security issues specific for the Cloud environments. In this chapter, we will give an overview of revised legal and application requirements that are partially addressed by the prototypes developed by Activity 2 and introduced in Chapter 3.

2.1 Legal Requirements

The nature of laws addressing data protection and data security is rather high level. There are no laws that pose requirements specifically for cloud computing or virtualisation on European level. European legislation tends to be technology neutral and unspecific. Single statutory norms require a high level objective for data processing. For instance, Article 17 of the European Data Protection Directive 95/46/EC requires the data controller in Paragraph 1 to implement

> *"appropriate technical and organizational measures to protect personal data against accidental or unlawful destruction or accidental loss, alteration, unauthorized disclosure or access, in particular where the processing involves the transmission of data over a network"*.

In Paragraph 2 the chosen processor must provide

> *"sufficient guarantees in respect of the technical security measures and organizational measures governing the processing to be carried out, and must ensure compliance with those measures"*.

These high level objectives have to be interpreted and applied to specific technical infrastructures like cloud computing. They need to be addressed at four different levels: *organisationally*, *contractually*, *technically on application level* and *technically on infrastructure level*. Only in combination those four levels of security measures can provide adequate comprehensive solutions. The opportunities and interplay of these layers of measures will be further described and analyzed in D1.2.3 and D1.2.4. The security objectives that can be derived from European legislation need to counter the specific risks that cloud computing poses regarding data protection and privacy. These risks will be identified in detail in D1.2.4 Cloud Computing - Privacy Risk Assessment. Without prejudice to national laws, there are key areas of privacy risks and the corresponding security objectives. The TClouds subsystems for a trustworthy internet-scale computing platform address these legal security objectives mainly on infrastructure level and a few on application level, so the following requirements focuses only on technical security measures. Please note that for meeting the identified security objectives there may exist several suitable measures. Often some of these have to be combined to achieve the best result.

Defined legal requirements are:

`LREQ1` - **Confidentiality of personal data**:
The Cloud Provider must prevent the breach of users' personal data by securing the infrastructure (including the internal network) and ensuring the isolation among different tenants. Further, he must avoid accesses on data by unauthorized entities through accesses management or, at least, must record relevant events through an auditable logging mechanism (that also logs actions performed by Cloud provider's employees). Confidentiality can be achieved also by encrypting data in a way that decryption would be possible only for customers.

`LREQ2` - **Availability and Integrity of personal data**:
The Cloud Provider must prevent the loss or manipulation of users' personal data through Duplication and Distribution (this poses some new risks, please refer to D1.2.3).

`LREQ3` - **Control of location (country wise) and responsible provider (cloud subcontractor)**:
The Cloud Provider must guarantee the applicability of law for processing personal data through location audit trails for the customer and safeguards that prevent data transfer to Cloud premises in other locations than those explicitly agreed with the customer.

`LREQ4` - **Unlinkability and Intervenability**:
The Cloud Provider must prevent unauthorized pooling, combining and merging of data through anonymization, pseudonymisation and splitting of data, through encryption of personal data (decryption only by customer) or isolation of tenants. The Cloud Provider must prevent the loss of control of data due to unauthorized copies through the encryption of data (with decryption by customers) or the effective and complete deletion. He must also provide to customers extensive control functions to avoid the risk of hindrance of the data subject's rights of access, rectification, erasure or blocking of data.

`LREQ5` - **Transparency for the customer**: The Cloud Provider must inform his customers about the security measures adopted to protect their personal data against loss of control due to unauthorized copies, manipulation, unauthorized pooling, combining and merging. The Cloud Provider must also prove that he did not circumvent the security measures chosen by providing customers with an auditable logging of accesses made by himself and his employees.

2.2 Application Requirements

This section provides an overview of the security and privacy requirements defined by A3 for the Healthcare and the Smart Lighting benchmark applications during the first year and revised during the development of the prototypes during the second year.

2.2.1 Healthcare Application

The TClouds healthcare application scenario focuses on developing a cloud-supported home healthcare application to provide collaborated services across different health care providers. The choice of adopting the technologies introduced by Cloud Computing was led especially by the significant costs required to provide a service accessible by remote users. These costs are arise either for building and maintaining a dedicated IT infrastructure within the hospital or for

outsourcing this service to an external organization that requires periodic payments regardless of the resources usage.

Cloud computing provides a solution to the above problem as it combines the outsourcing model with a pay-per-use model, enabling low entrance barriers and substantial cost reductions when no services are received or less resource are used. In addition, cloud computing offers scalability, because it allows to transparently add more resources to the service if there is an increase in demand, availability and resilience because a typical Cloud infrastructure is built to support a large number of customers and, finally, increased connectivity through redundant Internet connections.

However, hosting a service in a Cloud introduces security risks that do not apply to a dedicated IT infrastructure. The main objection to the adoption of Cloud Computing (65%) in the BridgeHead survey was the hospitals' concerns about the security and availability of healthcare data given the great number of threats, including privacy breaches and identity theft. Other objections include cost (26.1%) and a lack of confidence that Cloud offers greater benefits with respect to local storage media (26.1%). Current Cloud systems suffer from drawbacks and do not offer the expected Cloud infrastructure characteristics.

In the following, we present the revised security and privacy requirements that must be satisfied by the infrastructure to run the healthcare application in the Cloud environment.

`AHSECREQ1` - **Confidentiality of stored and transmitted data**:
Prevent that an attacker can retrieve and disclose data from the patient data repository or information transmitted through the communication channel between the personal front end and the management application.

`AHSECREQ2` - **Integrity of stored and transmitted data**:
Detect corruption done by an attacker of data stored in the patient data repository or exchanged through the communication channel between the personal front end and the management application.

`AHSECREQ3` - **Integrity of the application**:
Detect corruption of the management application done by an attacker to modify its functionality.

`AHSECREQ4` - **Availability of stored and transmitted data**:
Prevent Denial-of-Service attacks to the patient data repository or to the communication channel between the personal front end and the management application.

`AHSECREQ5` - **Availability of the application**:
Prevent Denial-of-Service attacks to the management application.

`AHSECREQ6` - **Non repudiation**:
Prevent that an attacker denies the fact that he/she has ever performed a specific action (e.g. he/she made the data available to unauthorized parties).

`AHSECREQ7` - **Accountability**:
Detect actions done by an attacker to provide him/her with privileges for the patient that should not be assigned to him/her.

`AHSECREQ8` - **Data source authentication**:
The attacker must not be able to run a process that appears as the legitimate management application.

AHPRIVREQ1 - Unlinkability and Anonymization of data flow:
Use data anonymization/pseudonymization techniques to anonymize/pseudonymize the documents stored in the data store and enforce process confidentiality (e.g. the state, the memory and administrative interfaces of the process) by means of strong/secure access control.

2.2.2 Smart Lighting Application

The purpose of the TClouds Energy application scenario is to develop a Public Light Management solution available online for Municipalities and the Utility operators. Traditionally, this type of solution would be hosted in the Utility Datacenter, mostly due to the required elements be already in place (D3.2.3 [SV12], Chapter 3).

Within TClouds, this solution is to be hosted initially in a commodity cloud environment, and then integrated with TClouds security components. In this way, we will investigate not only the constraints and feasibility of the migration to a cloud environment, but also the cost-benefits for adopting TClouds.

The following security requirements were specifically collected from "D3.3.3 - Validation Protocol and Schedule for the Smart Lighting and Home Health Use Cases" ([AN12]), within the Energy use case context.

ASSECREQ1 - Trustworthy Audit: Smart Lighting actions (application access, create, update, and delete data) must be fully audited, and accessible only to privileged users.

ASSECREQ2 - Trustworthy Infrastructure: The hosting infrastructure must prevent intrusions.

ASSECREQ3 - Trustworthy Persistence Engine: The persistence engine must prevent intrusions and ensure confidentiality, integrity and availability.

ASSECREQ4 - Resilient: The Smart Lighting System must be fault-tolerant at infrastructure and at persistence level.

ASSECREQ5 - Trustworthy communications: Communications between a client and the Smart Lighting System must prevent data from being altered by using adequate security mechanisms.

ASSECREQ6 - High performance & Scalable: The Smart Lighting System must have near-realtime performance, and be able to scale on increased load.

Chapter 3

Prototypes

This deliverable defines an initial integration of the TClouds subsystems defined in D2.4.1 [ea11c], Section 4.2.3, toward the final TClouds Platform v2.

We found that the best way to demonstrate progresses made during the second year of the project is to show the following three integrated prototypes:

1. Trustworthy OpenStack prototype (cf. Section 3.1)
2. TrustedInfrastructure Cloud prototype (cf. Section 3.2)
3. Cloud-of-Clouds prototype (cf. Section 3.3)

The Trustworthy OpenStack prototype as well as the TrustedInfrastructure Cloud prototype both contribute to the task of building a trusted infrastructure cloud with a different focus and different trust assumptions, as explained below. The Cloud-of-Clouds prototype is orthogonal as it is concerned with building middleware for resilience on top of existing cloud infrastructures. These existing cloud infrastructures can be today's commercial offerings as well as the Trustworthy OpenStack cloud and the TrustedInfrastructure cloud. Diversity of the cloud implementations is a desirable property for the Cloud-of-Clouds approach to achieve fault-tolerance against failures of a complete single cloud.

The Trustworthy OpenStack prototype enhances trustworthiness, security, and resilience of the open source framework OpenStack which operates on legacy operating systems. The TrustedInfrastructure cloud is rigorously building on top of Trusted Computing technologies and is operating on top of a Security Kernel. The focus of the TrustedInfrastructure cloud is to provide a managed IT infrastructure where administration is completely controlled and secured by the infrastructure and no manual administrator with elevated privileges is necessary. In this model a remote administrator of the cloud infrastructure has no longer to be trusted.

The integration effort among TClouds subsystems can be pushed further by combining the prototypes to support a mixed scenario that includes both private and public clouds. In particular, the private cloud could be a single trusted cloud (either Trustworthy OpenStack or TrustedInfrastructure Cloud), e.g., devoted to computation, while the public clouds could be commodity untrusted clouds, e.g., devoted to mass storage, pooled to form a service whose resilience is guaranteed through the Cloud-of-Clouds middleware.

Figure 3.1 shows such a high-level scenario, which is the object of the demonstration for the second year of the TClouds project: in particular, Trustworthy OpenStack has been chosen for the setup of the overall scenario, while the features of TrustedInfrastructure Cloud are demonstrated by a different setup.

This high-level scenario is only one possible option; indeed, the TClouds prototypes can be combined in similar ways to support different scenarios: more in general Trustworthy OpenStack and TrustedInfrastructure Cloud are two different implementations of a TClouds native

cloud, that can be either private or public while Cloud-of-Clouds prototype can run within and use whatever clouds, TClouds native and/or commodity clouds.

Table 3.1 lists all subsystems of D2.4.1 [ea11c], the prototype they are included in, or whether their integration is planned for the third year of the project. A new subsystem has been added to the list, the Remote Attestation Service, since being introduced during the second year.

This chapter is organized as follows: Sections 3.1-3.2-3.3 describe the prototypes that support the overall scenario and that will be demonstrated: for each prototype an architecture overview, the definition of an abstract API, and the demonstration workflow are explained. The abstract API lists the abstract functions that represent the interactions between components (shown as arrows among them), within the subsystems in the pictures of the demo workflow. Section 3.4 describes prototypes that are released in the second year but that are not part of the demonstration. Finally, Section 3.5 reports the mapping between the legal and application requirements introduced in Chapter 2 and the TClouds subsystems and prototypes.

TClouds subsystem	**TClouds prototype**
Resource-efficient BFT (CheapBFT)	Trustworthy OpenStack (Section 3.1)
Simple Key/Value Store (memcached)	[Year 3 prototype]
Secure Block Storage (SBS) (*)	Trustworthy OpenStack (Section 3.1)
Secure VM Instances (*)	Trustworthy OpenStack (Section 3.1)
TrustedServer	TrustedInfrastructure Cloud (Section 3.2)
Log Service	Trustworthy OpenStack (Section 3.1)
State Machine Replication (BFT-SMaRt)	Cloud-of-Clouds (Section 3.3)
Fault-tolerant Workflow Execution	[Year 3 prototype]
Resilient Object Storage	Cloud-of-Clouds (Section 3.3)
Confidentiality Proxy for S3	[Year 3 prototype]
Access Control as a Service (ACaaS)	Trustworthy OpenStack (Section 3.1)
TrustedObjects Manager (TOM)	TrustedInfrastructure Cloud (Section 3.2)
Trusted Management Channel	TrustedInfrastructure Cloud (Section 3.2)
Ontology-based Reasoner	libvirt: Standalone ("Other prototypes", Section 3.4); Other components: [Year 3 prototype]
Automated Validation (SAVE)	Standalone (see "Other prototypes", Section 3.4); already presented in Year 1
Remote Attestation Service **[New Year 2]**	Trustworthy OpenStack (Section 3.1)

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

Table 3.1: List of TClouds subsystems and mapping to prototypes

3.1 Trustworthy OpenStack Prototype

In this section we describe the demonstration prototype based on TClouds enhancements – i.e. a set of security extensions – to OpenStack (Trustworthy OpenStack), an overview including the architecture showing the involved subsystems, and the demonstration storyline.

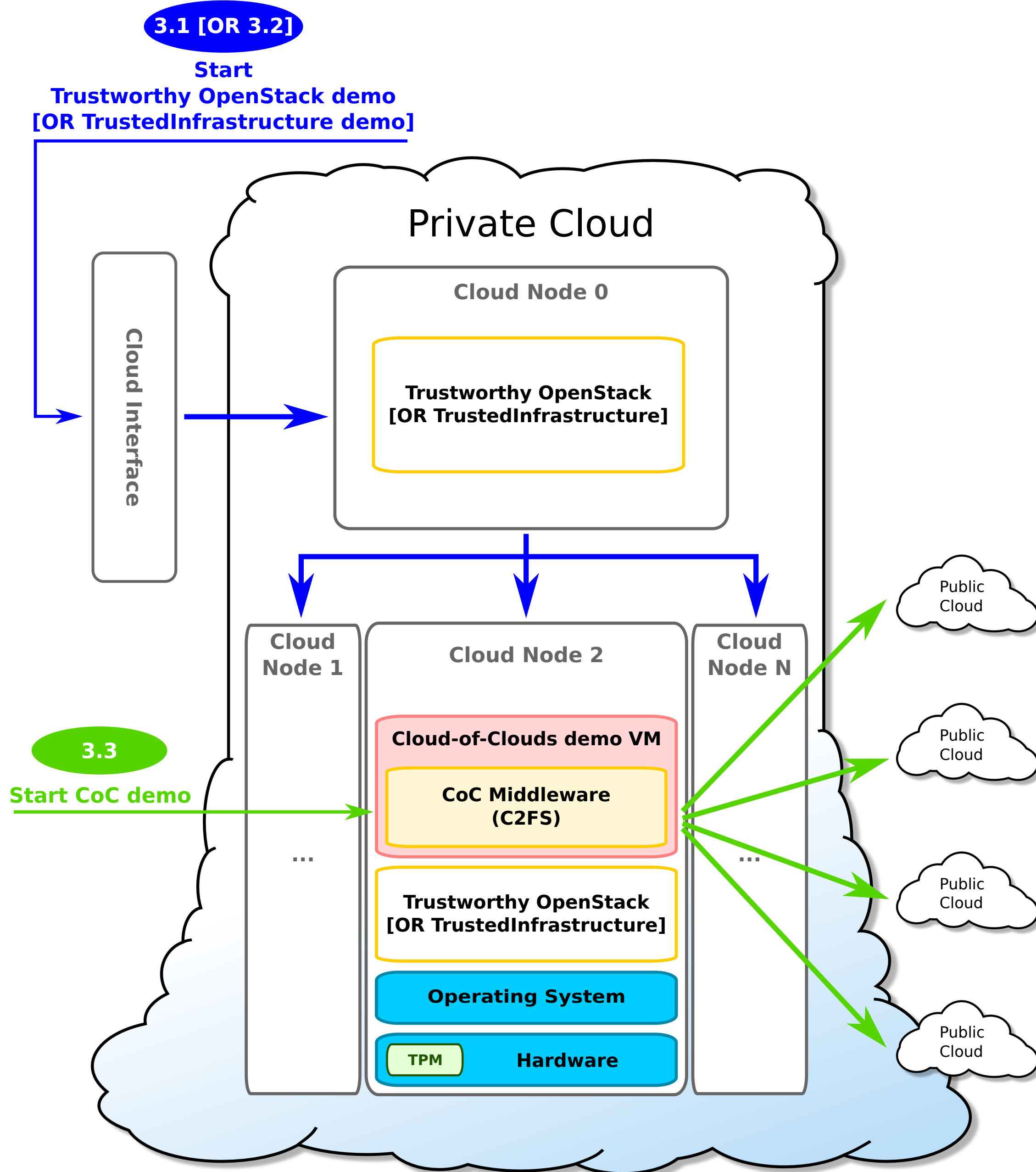

Figure 3.1: Prototypes demo architecture and scenario

3.1.1 Overview

Figure 3.2 illustrates the demo architecture showing various TClouds subsystems enhancing the security of OpenStack in various dimensions:

- Trust / Integrity:
  - The Remote Attestation Service enables users to trust that their virtual machines are actually deployed on computing nodes that satisfy their integrity requirements. Based on Trusted Computing technologies, e.g., a Trusted Platform Module, the remote attestation service verifies the configuration of the computing nodes and provides them for example to the cloud scheduler.
  - The enhanced scheduler (ACaaS) matches user requirements (e.g., location restrictions or white lists of measurements for deploying a VM) on the physical properties of computing nodes. The remote attestation service on the computing nodes is employed by the scheduler to query the computing nodes in a trustworthy way.
- Confidentiality: The Secure Block Storage and Secure VM Images subsystems provide disk encryption for volumes attached to virtual machines, as well as encryption of the VM images themselves. These subsystems offer an API to cloud users to securely provide the encryption keys without giving the cloud provider access to them. This is an important improvement over current encryption schemes where the keys are under control of the cloud provider.
- Resilience: The CheapBFT subsystem provides fault tolerance to the log service. With special FPGA hardware, this solution can tolerate byzantine (i.e. arbitrary) failure modes of a certain amount of computing nodes. While traditional solutions require $3f + 1$ replicas to tolerate f faults (i.e. 4 machines for 1 fault), the CheapBFT achieves the same with just $f + 1$ active replicas backed up by f passive ones.
- Audit: The Log Service is used to store logs of computing nodes selected by the Cloud Scheduler for VM deployment. It ensures confidentiality and integrity of the logs.

Note that the security improvements are conceived by the synergy of the careful selection and integration of TCloud subsystems, e.g.:

- The Remote Attestation Service and ACaaS together ensure that the selection of computing nodes matches the users requirements. Together with the Log Service the scheduling decisions are securely stored for audit.
- The integration of the Log Service with CheapBFT provides a fault tolerant implementation of the Log Service within the cloud infrastructure.
- The combination of Secure Block Storage with the ACaaS and Remote Attestation ensures that encrypted images will only be deployed on computing nodes that properly secure the encryption keys.

The result of the integration of such subsystems is Trustworthy OpenStack, i.e. the standard OpenStack enhanced with the following security extensions: Secure Logging, Advanced VM Scheduling, Cloud Nodes Verification/Remote Attestation, and VM Images Transparent Decryption (see Figure 3.2).

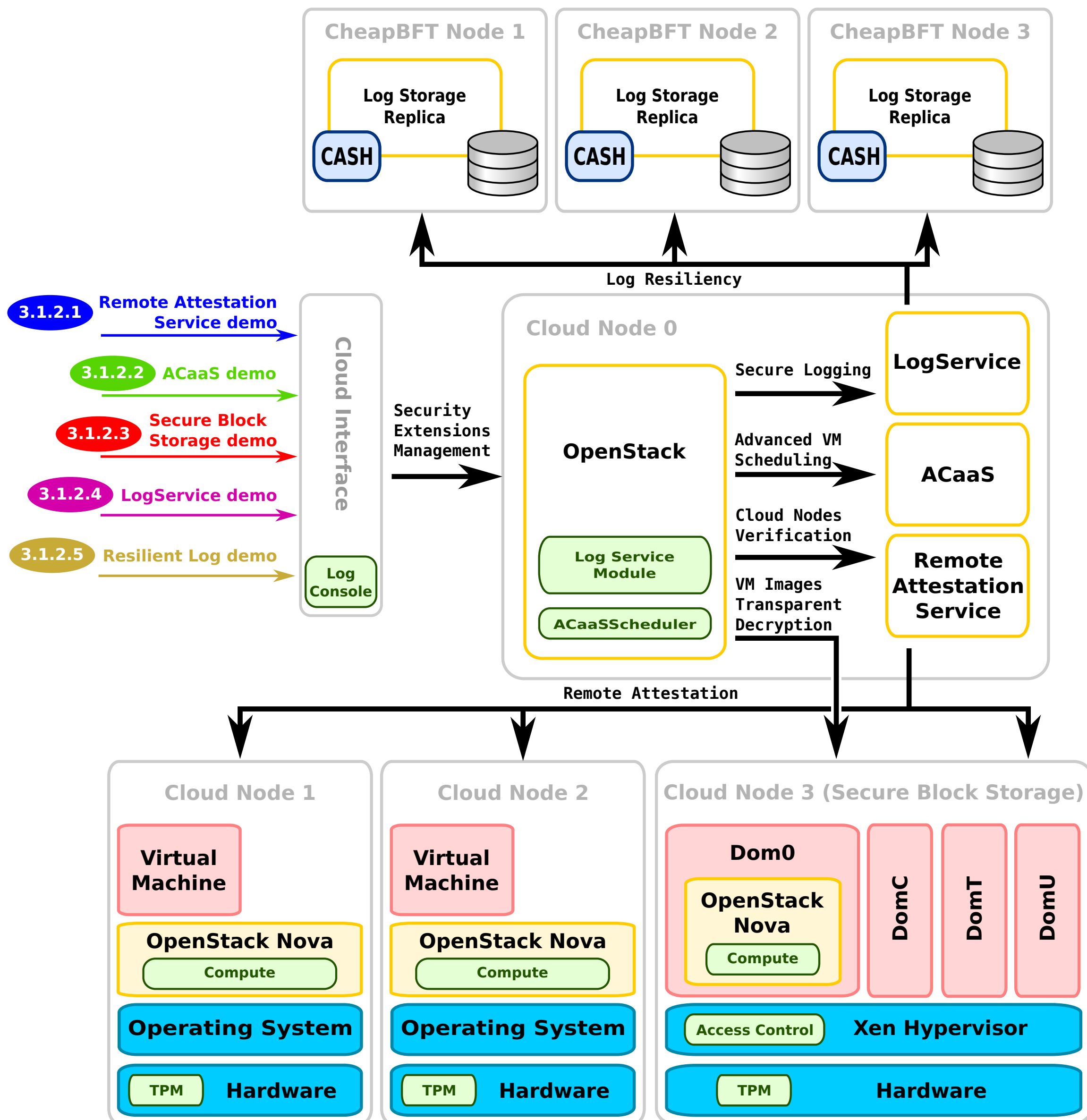

Figure 3.2: The Trustworthy OpenStack demo architecture

In the following sections, the commands to run the demo according to the storyline have to be given on command line. However the standard OpenStack dashboard has been enhanced to support the setting of for three Security Extensions (Secure Logging, Advanced VM Scheduling, Cloud Nodes Verification/Remote Attestation). Appendix C collects some screenshot of the TClouds Trustworthy OpenStack Dashboard.

3.1.2 Demo Storyline

We exercise the different subsystems of the demo architecture by a executing a number of use cases.

3.1.2.1 Remote Attestation Demo

Architecture Overview. The *Remote Attestation Service* is a Cloud subsystem responsible to assess the integrity of the nodes in the Cloud infrastructure through techniques introduced by the Trusted Computing technology.

This service gives significant advantages in the Cloud environment. First, it allows Cloud users to deploy their virtual machines in a physical host that satisfies the desired security requirements – represented by five integrity levels. Requiring a higher level will give more confidence and trust into the used physical hosts.

Secondly, this service allows Cloud Administrators to monitor the status of the nodes in an efficient way and to take appropriate countermeasures once a compromised host has been detected. For instance, administrators can isolate the host such that it can not attack other nodes of the infrastructure.

The *Remote Attestation Service* consists of two main components:

- `OpenAttestation`: this framework, developed by Intel, enables the *OpenStack Nova Scheduler* to retrieve and verify the integrity of Cloud nodes such that the former can select a host that meets the users requirements. The framework handles the Remote Attestation protocol through two submodules that act as the endpoints: *HisClient* collects the measurements done by the attesting platform, generates and sends the integrity report to the verifier; *HisAppraiser* verifies the integrity report received from a Cloud node and assigns to the latter an integrity level.
- `RA Verifier`: this component analyses the measurements performed by the Integrity Measurement Architecture (IMA), a subsystem of the Linux Kernel, running on Cloud nodes. In particular, it verifies whether the digest of binary executables and shared libraries are present in a database of known values and whether the packages these files belong to are up to date. The first check allows to detect possibly malicious software that may have been executed before verification, while the second check allows to identify loaded applications with known vulnerabilities that may be exploited by an attacker.

Abstract API

`Result` $\Leftarrow$ **`RegisterInstanceType()`**
Register a new virtual machine instance type.

`Result` $\Leftarrow$ **`StartVM()`**
Start a new virtual machine.

`HostsList` ⇐ **`GetAvailableHosts()`**
Return the list of physical hosts with enough resources to start a new virtual machine.

`IntegrityLevel` ⇐ **`GetHostIntegrityLevel()`**
Remotely attest a Cloud node and return its current integrity level. Defined levels are:

- *l0_boot_untrusted:* invalid integrity report;
- *l1_ima_digest_not_found:* unknown digests;
- *l2_ima_pkg_security_updates:* packages with security vulnerabilities;
- *l3_ima_pkg_not_security_updates:* packages with other vulnerabilities;
- *l4_ima_all_ok:* all digests recognized and packages up to date.

`IntegrityReport` ⇐ **`GetIntegrityReport()`**
Generate a new integrity report and return it to the caller.

`VerificationResult` ⇐ **`VerifyIntegrityReport()`**
Verify the IMA measurements and return the result (i.e the number of unknown digests and the number of packages with security and/or other vulnerabilities).

`Boolean` ⇐ **`VerifyIntegrityRequirements()`**
Return true if the current integrity level of the Cloud node matches the one specified in the instance type. Otherwise, return false.

`Result` ⇐ **`DeployVM()`**
Deploy a new virtual machine on the selected Cloud node.

Demo workflow. The Figure 3.3 shows the interactions that occur during the definition of a new instance type[1] and during the deployment of a new virtual machine. In the following, we refer to *Openstack Nova* as the component that performs the above tasks and we mention the submodules *Scheduler* and *Compute* only when they are relevant.

Openstack Nova will take advantage of the security logging capability, offered by *LogService*, by logging the events related to the selection of the physical host depending on user's requirements, so that the logging events can be retrieved at later time by an auditor.

During the demo, we will perform the following steps:

(A) Register a new VM instance type (integrity level: l4_ima_all_ok)
A Cloud user registers a new instance type by prompting the following commands on the Cloud interface:

```
 $ nova-manage instance_type create --name=m1.tiny_trust_lvl_4 --memory=64 --cpu=1
--flavor=6 --root_gb=0 --ephemeral_gb=0 --swap=0
$ nova-manage instance_type extra_specs_create --name=m1.tiny_trust_lvl_4
--extra_spec_key=trusted_host --extra_spec_value=l4_ima_all_ok
```

The registration is performed by the `RegisterInstanceType()` method of *OpenStack Nova*, which stores the parameters of the new instance type in a persistent database.

[1] An instance type represents the virtual hardware that an instance will be provided with, i.e. the amount of RAM, the number of CPUs and the disk size that will be allocated for an instance; many instance types (also known as flavors) can be defined and named with a label.

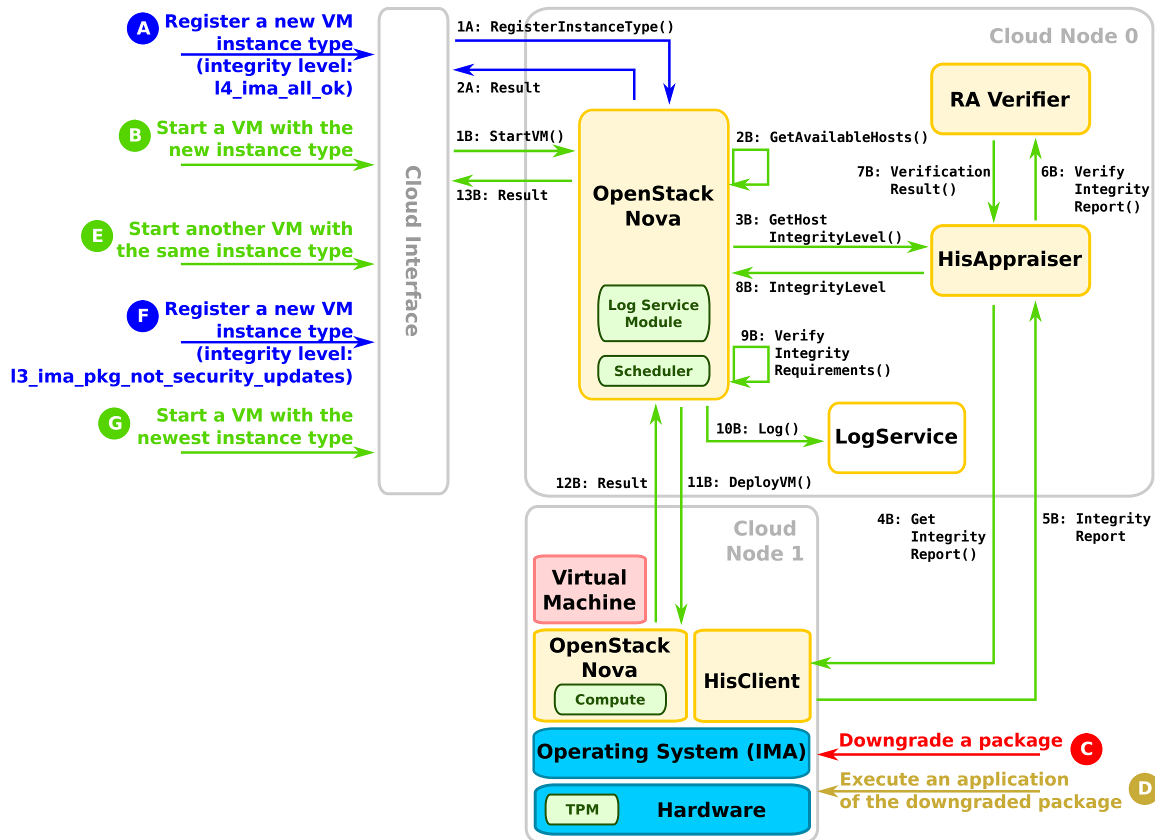

Figure 3.3: Remote Attestation Service demo workflow

(B) Start a VM with the newly created instance type

A Cloud user starts a new virtual machine by executing the command:

```
nova boot --image <image-uuid> --flavor 6 demovm
```

The `StartVM()` method of *OpenStack Nova* is implemented as follows:

- *OpenStack Nova Scheduler* builds the list of Cloud nodes with available resources by executing `GetAvailableHosts()`;
- For each node, *OpenStack Nova Scheduler* requests to *OpenAttestation* the current integrity level by invoking `GetHostIntegrityLevel()` and stops once it finds a physical host that meets the user's integrity requirements;
- *HisAppraiser* contacts the Cloud node and requests it to generate and send a new integrity report by invoking the `GetIntegrityReport()` method of *HisClient*;
- *HisAppraiser* calls the `VerifyIntegrityReport()` method of *RA Verifier* in order to verify the integrity report received. Then, it determines the current integrity level of the node depending on the verification result and returns it back to *OpenStack Nova Scheduler*;
- *OpenStack Nova Scheduler* verifies whether the attested Cloud node satisfies the user's integrity requirements by invoking `VerifyIntegrityRequirements()`. This function will return a positive result;

- *OpenStack Nova Scheduler* creates a log with the name of the physical host being analyzed and the result of the selection process and sends it to *LogService* by invoking the method `Log()`;
- *OpenStack Nova Scheduler* deploys the new virtual machine on the verified Cloud node by calling the `DeployVM()` method of *OpenStack Nova Compute*;
- *OpenStack Nova* returns the result of the requested operation to the Cloud user.

(C) Downgrade a package in the Cloud node
A Cloud operator performs a downgrade of an installed package in the Cloud node by executing:

```
$ apt-get install <package-name>=<package-version-number>
```

(D) Execute a binary file of the downgraded package
A Cloud operator executes an application from the downgraded package by prompting the command in the Cloud node:

```
$ <binary name>
```

(E) Start another VM with the same instance type
This step will fail because there are no Cloud nodes available that satisfy the user's integrity requirements.

(F) Register a new VM instance type (integrity level: l3_ima_pkg_not_security_updates)
A Cloud user repeats the step (A) and creates a new instance type with a lower integrity level (l3_ima_pkg_not_security_updates).

(G) Start a VM with the newest instance type
This operation now will succeed, because *OpenStack Nova Scheduler* finds a node that satisfies the user's integrity requirements.

3.1.2.2 Access Control as a Service (ACaaS) Demo

Architecture Overview. ACaaS is a subsystem ensuring that user VMs are only executed on hosts matching their security requirements. It is composed of three components: *AcaaSScheduler*, *Requirements Service*, and *Security Properties Service*. *Requirements Service* and *Security Properties Service* implement interfaces for users and administrators to setup scheduling requirements for VMs and security properties for compute nodes. This information is maintained by these services in the *nova-db* and is retrieved by the *AcaaSScheduler* when the latter receives a request to instantiate a new VM.

OpenStack Scheduler is enhanced by *ACaaSScheduler* to achieve ACaaS-based VM scheduling. It can deploy VMs with specific security requirements on compute nodes with targeted security properties. Every time a request for instantiating a VM is received, *ACaaSScheduler* first examines whether scheduling requirements are specified. Then, it iterates over all connected hosts and selects only the one with properties that satisfy the requirements for hosting the VM. In supporting this advanced scheduling, *OpenStack Nova* client interfaces are also modified.

To implement the requirements and security properties management facilities, the core database of OpenStack is modified. A table named `security_requirements` is added to

manage user requirements. Moreover, a new field named `security_properties` is added to the table `compute_nodes` to record the properties of the compute node. These properties are represented by the name of requirements defined in the `security_requirements` table, and a corresponding value. When a user is instantiating a VM, a requirement ID can be specified, along with the expected value. Only the hosts with the same value of the specified requirement ID can be added to the scheduling list for the VM.

An advanced scheduling criterion called `exclude-user` is also implemented by the *ACaaSScheduler*. Users can specify that their VMs can only be initiated on the hosts that are not running VMs of other particular users. In this case, a special requirement `exclude-user` (with an alphabetic ID `x`) is predefined. Upon receiving this requirement, *ACaaSScheduler* performs a search in the database through the new function `HostGetAllByUserInstance()`, introduced in the next paragraph, and fetches only the hosts without VMs from the specified user.

Further, the expected states of a host can be also defined as a scheduling criterion in the ACaaS prototype. Users can request *ACaaSScheduler* to deploy their VMs only on hosts with specific platform configurations (i.e. trusted properties). This is implemented by defining a new table, named `white_lists`, whose entries contain an ID and the location of the target white-list file. These entries can be referenced by records in the `compute_nodes` table so that it is possible to associate a compute node with a set of desired trusted properties. From this point, this node will be attested to by the management node, according to Trusted Computing specifications, against the white-list identified by the white-list ID. If the management node detects any violation during the attestation, it will re-initialize the compute node or will remove it from the scheduling pool. More details about the infrastructure that performs these tasks can be found in the D2.3.2 [ea12c] deliverable Chapter 9.

Abstract API

`Result` ⇐ **`ReqCreate()`**
Create a requirement and return the requirement ID.

`None` ⇐ **`ReqUpdate()`**
Update a requirement.

`None` ⇐ **`ReqRemove()`**
Remove a requirement.

`ReqName` ⇐ **`ReqGet()`**
Return a requirement with specified ID.

`Result` ⇐ **`ReqGetAll()`**
List all requirements.

`SecurityPropertiesList` ⇐ **`SecurityPropertiesGet()`**
Fetch all security properties associated to a host.

`None` ⇐ **`SecurityPropertiesUpdate()`**
Add or modify security properties of a host.

`None` ⇐ **`SecurityPropertiesRemove()`**
Remove security properties of a host.

`HostsList` ⇐ **`HostGetAllByUserInstance()`**
Get all the hosts deployed with VM instances of a specified user.

`Result` ⇐ **`InstantiateVMWithReqs()`**
Initiate a VM on a host satisfying specified requirements.

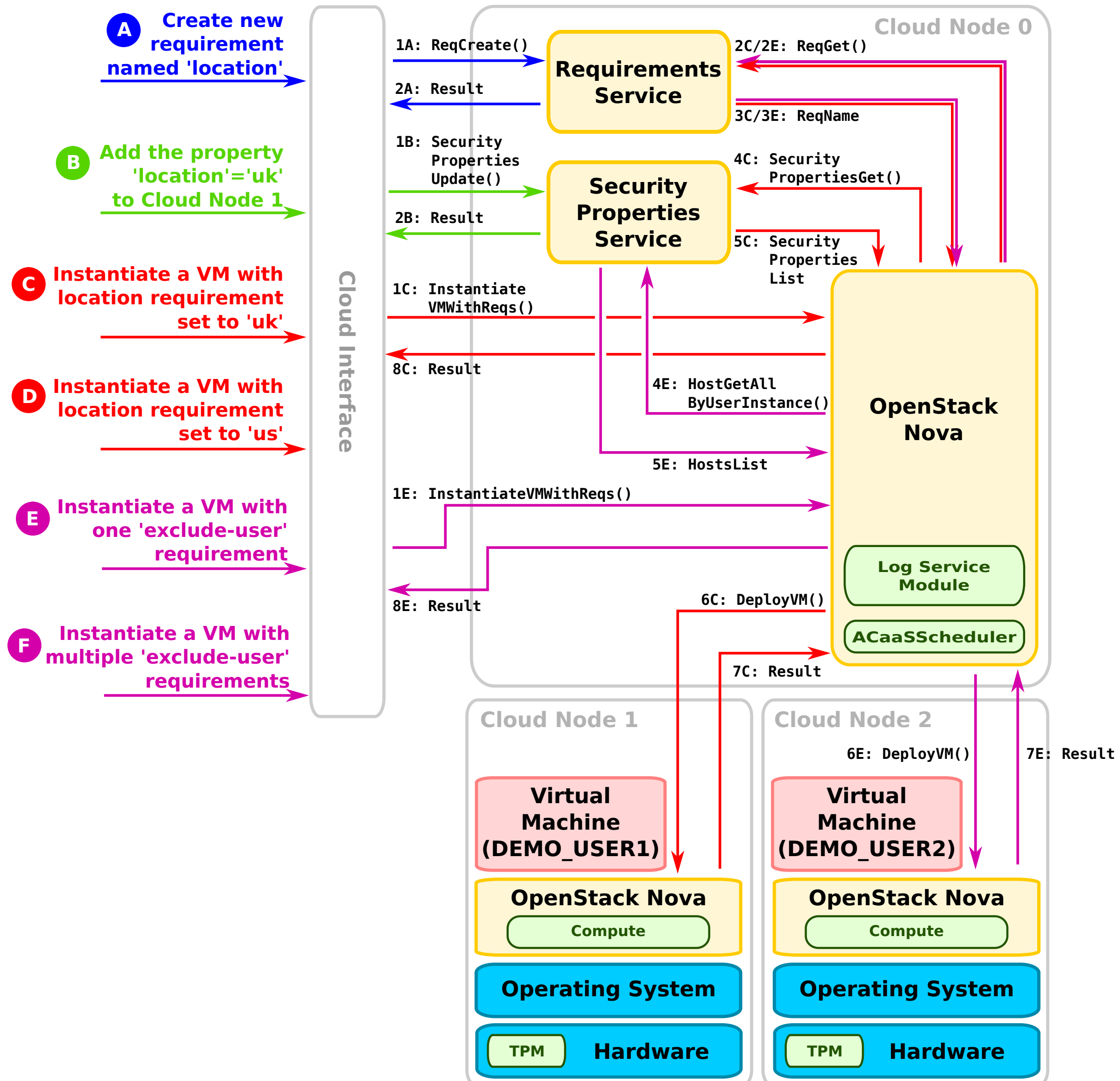

Figure 3.4: ACaaS demo workflow

Demo Workflow. At least two machines, the Cloud Node 1 (HOST1) and the Cloud Node 2 (HOST2) are needed for the demonstration. Four users have been configured, one with administration privileges (DEMO_ADMIN), and three others (DEMO_USER[1-3]).

Figure 3.4 depicts each step of the demo and the interactions among the different ACaaS entities. Steps (A) to (D) show the instantiation of a VM with specified requirements on `location`. Security requirements and platform properties should be set up correctly in advance. Steps (E) and (F) show an application of the show the '`exclude-user`' requirement. The latter can be specified using the characters 'x' or 'X' when specifying the REQ_ID. This

requirement specifies that the VM can only be deployed on the compute host with NO other VMs belonging to the users with specified USER_IDs (the DEMO_USER1 in this case).

To further increase the reliability and trustworthiness of this service, it can be combined with the secure log service.

(A) Create new requirement named 'location'

With the following command executed by the user DEMO_ADMIN, a new entry with the value `location` is created in the `security_requirements` table.

```
$ nova-manage requirement create --requirement='location'
```

(B) Add the property 'location' = 'uk' to HOST1

With the following command executed by the user DEMO_ADMIN, the `security_properties` field of HOST1 in the table `compute_nodes` is updated by setting the `location` property to `'uk'`.

```
$ nova-manage host add_properties --host=HOST1 --properties="'location': 'uk'"
```

(C) Instantiate a VM with location requirement set to 'uk'

The user DEMO_USER1 initiates a VM with the '`location`' = '`uk`' requirement and displays the log to show that the VM has been successfully instantiated on HOST1 (Assuming the requirement ID for '`location`' is 1)[2].

After prompting the following command, *ACaaSScheduler* first fetches the requirement object stored in the database with the requirement ID for '`location`'. It then fetches the list of hosts from the `compute_nodes` table with the property '`location`' set. And finally, it iterates every host in the list, and returns the most suitable one according to other scheduling criteria (in regarding only to *ACaaSScheduler*, the first one is returned).

```
$ nova boot --flavor m1.tiny --image IMAGE_UUID --req="1:'uk'" DEMO_IMAGE
```

(D) Instantiate a VM with location requirement set to 'us'

The user DEMO_USER1 initiates a VM with the '`location`' = '`us`' requirements, and displays the log to show that the VM instantiation has failed (Assuming the requirement ID for '`location`' is 1).

ACaaSScheduler iterates the same procedure as the previous step, and returns NONE as the host scheduling candidate. In this case, the VM is not deployed, and the failed-in-deployment event is recorded in the log.

```
$ nova boot --flavor m1.tiny --image IMAGE_UUID --req="1:'us'" DEMO_IMAGE
```

(E) Deploy a VM with one `exclude-user` requirement

The user DEMO_USER2 initiates a VM with the '`exclude-user`' = '`DEMO_USER1`' requirement and displays the log to show that the VM has been successfully instantiated on HOST2.

[2] Each node requirement specified using ACaaS is assigned an IDentifier that is then used to select such requirement and to set a value for it.

With this command, *ACaaSScheduler* fetches only the hosts without VMs belonging to DEMO_USER1, as discussed above, and schedules the VM to the first host of the list. In this case, as HOST1 is deployed with a VM of DEMO_USER1, the new VM will be instantiated on HOST2.

```
$ nova boot --flavor m1.tiny --image IMAGE_UUID --req="'x':['DEMO_USER1']"
DEMO_IMAGE
```

(F) Deploy a VM with multiple `exclude-user` requirements
The user DEMO_USER3 initiates a VM with the '`exclude-user`' = '`DEMO_USER1, DEMO_USER2`' requirements and displays the log to show that the VM instantiation was failed.

ACaaSScheduler proceeds the same procedure as the above step. However, as both the hosts are deployed with VMs belonging to either DEMO_USER1 or DEMO_USER2, it returns NONE as the host scheduling candidate. In this case, the VM is not deployed and the failed-in-deployment event is recorded in the log.

```
$nova boot --flavor m1.tiny --image IMAGE_UUID --req="'x':['DEMO_USER1',
'DEMO_USER2']" DEMO_IMAGE
```

3.1.2.3 Secure Block Storage (SBS) and Secure VM Images Demo (also known as Cryptography-as-a-Service)

Architecture Overview. The Secure Block Storage (SBS) component enables a VM to use an encrypted storage device transparently as if it were plaintext. Due to the modular design it builds the foundation for Secure VM Images, the ability to even boot from encrypted devices while still preserving confidentiality and integrity against the Management Domain (i.e. VM) and hence the cloud personnel and other customers. SBS consists of the following modules (see also Figure 3.5):

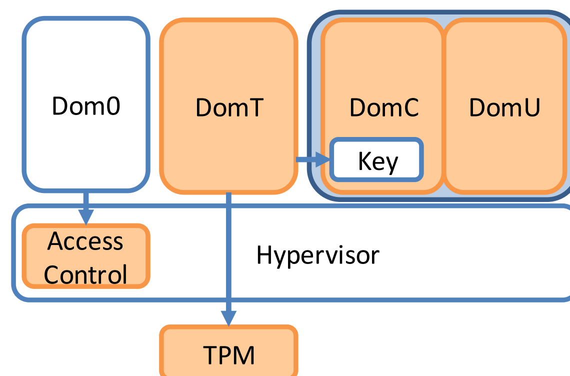

Figure 3.5: SBS Modules Overview

- The secure hypervisor
- A de-privileged Management Domain (Dom0)
- A minimal TCB VM builder and TPM management domain (DomT)
- The cryptographic Micro-VM (DomC)
- The actual customer VM (DomU)

Abstract API

`PubKey` ⇐ **`GetNodePubKey()`**
This function instructs DomT to fetch the public key of the node from the TPM and returns it to the user.

`Result` ⇐ **`StoreEncImageAndEncKey()`**
This function, that will be implemented in OpenStack, will store the encrypted image and the encrypted key in the Cloud storage.

`(EncImage, EncKey)` ⇐ **`GetEncImageAndEncKey()`**
This function, that will be implemented in OpenStack, will fetch the encrypted image and the encrypted key from the Cloud storage and will copy them to the target Cloud node.

`Key` ⇐ **`DecryptEncKey()`**
This function will decrypt the symmetric key using the TPM.

`Result` ⇐ **`PutKey()`**
This function will put the decrypted symmetric key on DomC.

`Result` ⇐ **`ScheduleDomU()`**
This function will start DomU.

`Result` ⇐ **`ScheduleDomC()`**
This function will start DomC.

`Result` ⇐ **`StartVM()`**
Start a new virtual machine.

`Result` ⇐ **`DeployVM()`**
Deploy a new virtual machine on the selected Cloud node.

Demo Workflow.

(A) Import Certified Key and Encrypt Image

The cloud consumer first imports the key of the node on which to run his VM:

```
$ deployer --import-node node_keyfile.asc
```

This commands imports a *certified binding key* (see Section 3.3.2 of deliverable D2.1.2 [ea12b]), which is a public key of the node which includes a certificate by the hardware trust anchor which vouches that the secret part of this key can only be used in a trusted configuration. The tool verifies the certificate and also verifies that the 'trusted configuration' to which the key is bound, matches a list of known software hashes which are deemed trustworthy by the consumer.

The consumer then encrypts the VM of choice for this node.

```
$ deployer --encrypt my_linux.img --outfile encr_img.enc --payload secrets.enc
```

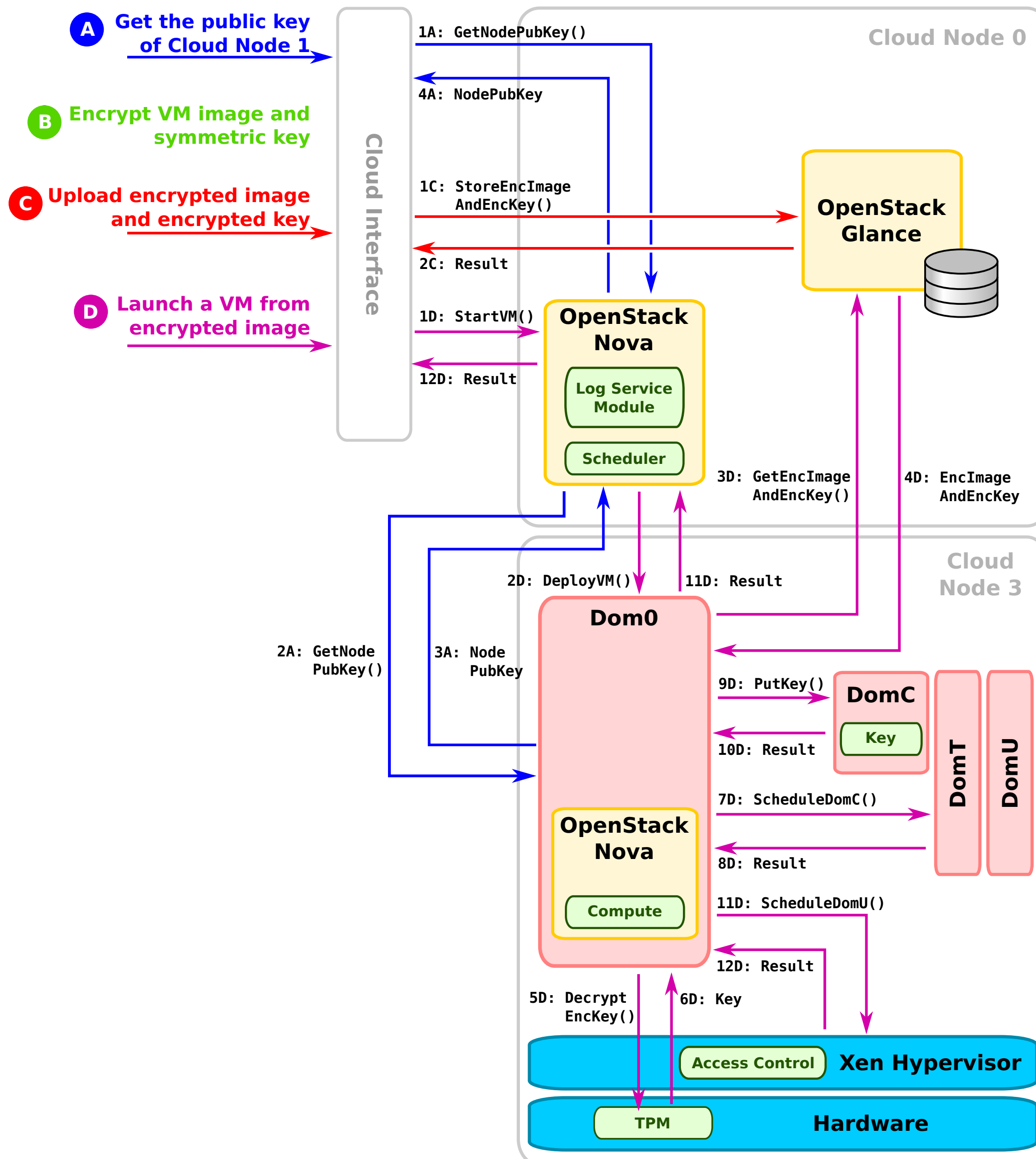

Figure 3.6: Secure Block Storage demo workflow

This operation will first encrypt the consumer's VM image with a *symmetric* key k (storing the encrypted image to `encr_img.enc`), after which k itself is encrypted *asymmetrically* with the public key of the cloud node which has been imported earlier in this step (storing it in `secrets.enc`).

(B) Uploading Encrypted VM Image
Both encrypted payloads, the key for use in *DomC* and the encrypted VM image are uploaded to the cloud and stored as usual on untrusted storage.

(C) Launch encrypted VM image
After the encrypted VM image is deployed in the cloud, the cloud administrator has to instruct Xen it wants to boot this customer's VM. In a standard Xen VM configuration file, extra information is added to tell the domain builder where the encrypted VM and payload are stored. For instance:

```
 # cat >> /etc/xen/encrypted_vm.cfg << EOF
domc_enabled = '1'
image_file = '/var/lib/xen/images/customer_1.enc'
payload_file = '/var/lib/xen/domc/payload_customer_1.enc'
EOF
```

The actual starting of the VM takes place using the following command.

```
 # xl create -c /etc/xen/encrypted_vm.cfg
```

The domain builder (DomT) takes care of the whole start VM operation in automatic fashion. DomT uses the TPM to decrypt the customer's key and puts it in his *DomC* (**C1**). Before DomU can be started, to be accessed the encrypted storage image must be decrypted by DomC on-the-fly (**C2**) using the user-provided key in DomC. After this transparent decryption/encryption is set up for the storage device, *DomU* is is put into the VCPU scheduler of Xen in order to make it runnable (**C3**).

3.1.2.4 LogService Demo

Architecture Overview. The LogService is the component that manages secure logging events in the TClouds cloud infrastructure. Within LogService, four components are defined:

- The *Log Service Module* is a software module linked to *Cloud Components* and allowing them to produce log entries using the secure logging scheme proposed by Schneier and Kelsey in [SK99] and to forward them to the *Log Storage*.
- The *Log Storage* is the component that stores the log entries produced by *Cloud Components* and groups them by logging session.
- The *Log Core* is a trusted party involved in the initialisation, closure, and verification of the logging sessions.
- The *Log Console* is a log management console which can be used by an external entity (auditor or user) to verify and retrieve logs.

The main building block of the LogService is the `libsklog` library. This library is used by the *Cloud Component* and the *Log Core* for the initialization of a new logging session, by the *Cloud Component* for the creation of the log entries using the secure scheme, and finally by the *Log Console* and the *Log Core* during the verification process.

Abstract API

`InitResult` ⇐ **`InitLog()`**
Initialize a new logging session on the *Log Core*.

`None` ⇐ **`LogEvent()`**
Log events on the *Log Storage* and bind them to the opened session.

`SessionsList` ⇐ **`RetrieveSessions()`**
Retrieve all the logging sessions opened by *Cloud Components*.

`(VerificationResult, TemporaryURL)` ⇐ **`VerifySession()`**
Verify the integrity of the selected logging session and return a temporary URL of its dump.

`LogsList` ⇐ **`RetrieveLogs()`**
Return the logs that are part of the selected logging session.

`VerificationResult` ⇐ **`VerifyLogsList()`**
Verify the integrity of the logs using the Schneier and Kelsey scheme and create a logs dump.

`VerifiedLogsDump` ⇐ **`DownloadLogsDump()`**
Download the logs dump from the temporary URL.

Demo Workflow. Figure 3.7 depicts each step of the demo and the interactions among the different LogService components. This workflow may be executed by an external entity, in this case the *Auditor*, in order to verify whether a specific operation (e.g., a user request) has been performed correctly by the cloud infrastructure. Before executing the workflow, the *Cloud Component* requests at the *Log Core* the initialization of a new logging session by calling `InitLog()` and starts saving log entries by invoking `LogEvent()`.

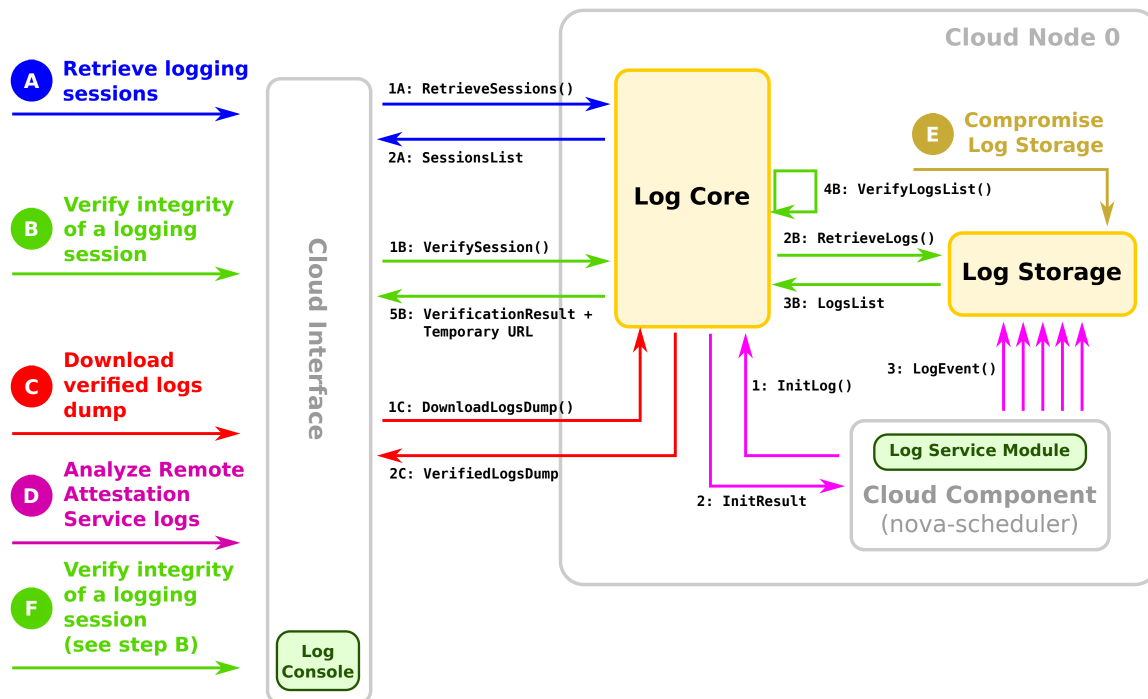

Figure 3.7: LogService demo workflow

(A) Retrieve logging sessions
Through the *Log Console* The *Auditor* requests the *Log Core* the IDs of the already initialized logging sessions by invoking `RetrieveSessions()`.

(B) Verify integrity of a logging session
Through the *Log Console* the *Auditor* requests the *Log Core* the verification of a logging

session by calling `VerifySession()` and by supplying its ID. Then, the *Log Core* retrieves the logs from the *Log Storage* by calling `RetrieveLogs()`, verifies them by invoking `VerifyLogsList()` and generates a verified logs dump. Finally, *Log Core* forwards to the *Auditor* the verification result and a temporary URL pointing to the dump.

(C) Download verified logs dump
The *Auditor* downloads from the temporary URL the dump of verified logs by calling `DownloadLogsDump()`.

(D) Analyze Remote Attestation Service logs
The *Auditor* analyses the correctness of the operation previously executed during the Remote Attestation Demo (see Section **??**) at step B.

(E) Compromise Log Storage
The logs collected on the *Log Storage* are tampered in order to simulate a corruption.

(F) Verify integrity of a logging session (after the attack)
The step B is repeated in order to show that the LogService successfully detects the compromised logs.

3.1.2.5 Resilient LogService Demo

Architecture Overview. The LogService component described so far is able to detect compromised logs or, more precisely, logs corrupted in any way. Although this ensures the integrity of the log service, this is obtained at the cost of availability. If the log storage becomes faulty, due to deliberate attacks, careless use, hardware errors, or other reasons, the service will not be able to provide a correct log file.

This problem can be tackled by replicating the storage. Considering the requirement of guaranteeing integrity in the presence of arbitrary faults, the state machine replication scheme for Byzantine Fault Tolerance (BFT) would be a suitable solution. However, this scheme normally entails very high resource consumption because $3f + 1$ actively operating replicas are required to tolerate f faults. Therefore, we employ *CheapBFT* as basis for the resilient implementation of the LogService component. CheapBFT combines the active and passive replication schemes and utilizes a trusted FPGA-based submodule in order to lower the number of actively involved replicas to $f + 1$ in normal, error-free operation.

The resilient Log Service variant comprises the following main modules:

- A *CheapBFT Replica* is an instance of the CheapBFT server module that cooperates with other replicas to provide a platform for a Byzantine fault-tolerant service implementation. A CheapBFT replica can be either active or passive. Active replicas actually process and respond requests from clients whereas passive ones only update their state according to state changes determined by their active counterparts.

- CASH is a trusted FPGA-based submodule used by the CheapBFT replicas to sign and verify messages. Broadly speaking, it implements a trusted counter and is only subject to crash faults.

- *CheapBFT Proxy* is the software module which is used by clients to communicate with a CheapBFT replica group.

- The *Log Storage* provides the same functionality as described in Section 3.1.2.4, although it was adapted to run on top of the platform created by the CheapBFT replicas.
- The *Log Storage Proxy* sends requests to and receives replies from the Log Storage by utilizing the CheapBFT Proxy.

Furthermore, besides the Log Core and the Log Service module (see Section 3.1.2.4) we use one additional module for demonstration purposes:

- The *Log Generator* corresponds to the Cloud Component used in the LogService demo. Contrary to it, the Log Generator constantly produces log events which are stored by means of the (resilient) Log Storage.

Abstract API

`None` ⇐ **`StartGenLogs()`**
Instructs the Log Generator to start constantly generating log events.

`None` ⇐ **`StopGenLogs()`**
Instructs the Log Generator to stop generating of log events.

`CmdResult` ⇐ **`Invoke(<cmd>)`**
Send a command `<cmd>` to the service based on CheapBFT, that is the Log Storage. Here, `<cmd>` can be either `store` to store a log event or `retrieve` to retrieve the log file.

`CmdResult, StateChange` ⇐ **`ExecuteCommand()`**
Execute a command sent by a client and calculate the result as well as the service state modification.

`None` ⇐ **`UpdateSrvState()`**
Apply a service state modification calculated by an active replica.

`SrvState` ⇐ **`GetSrvState()`**
Return the service state.

`CmpResult` ⇐ **`CompSrvState()`**
Determine whether service states are equal.

`None` ⇐ **`ReachAgreement()`**
Reach agreement about which commands have to be executed and in which order.

`None` ⇐ **`StartCheapSwitch()`**
Start the protocol CheapSwitch, which is responible for changing the consensus protocol (CheapBFT or MinBFT).

`None` ⇐ **`ActivateReplica()`**
Activate a currently passive replica.

`MsgCert` ⇐ **`CreateMsgCert()`**
Increase the trusted counter and use it to create a certificate unique for the given message.

`VerificResult ⇐ CheckMsgCert()`
Validate a given message certificate and verify it against the expected counter value.

For the description of the functions belonging to the pure LogService component, see Section 3.1.2.4.

Demo Workflow. The purpose of this demo is twofold: First, it shows the improved resilience of the Log Service variant implemented on top of CheapBFT. Second, it demonstrates CheapBFT as a resource-efficient system for Byzantine fault-tolerant services. Figures 3.8, 3.9 and 3.10 illustrate the course of this demo.

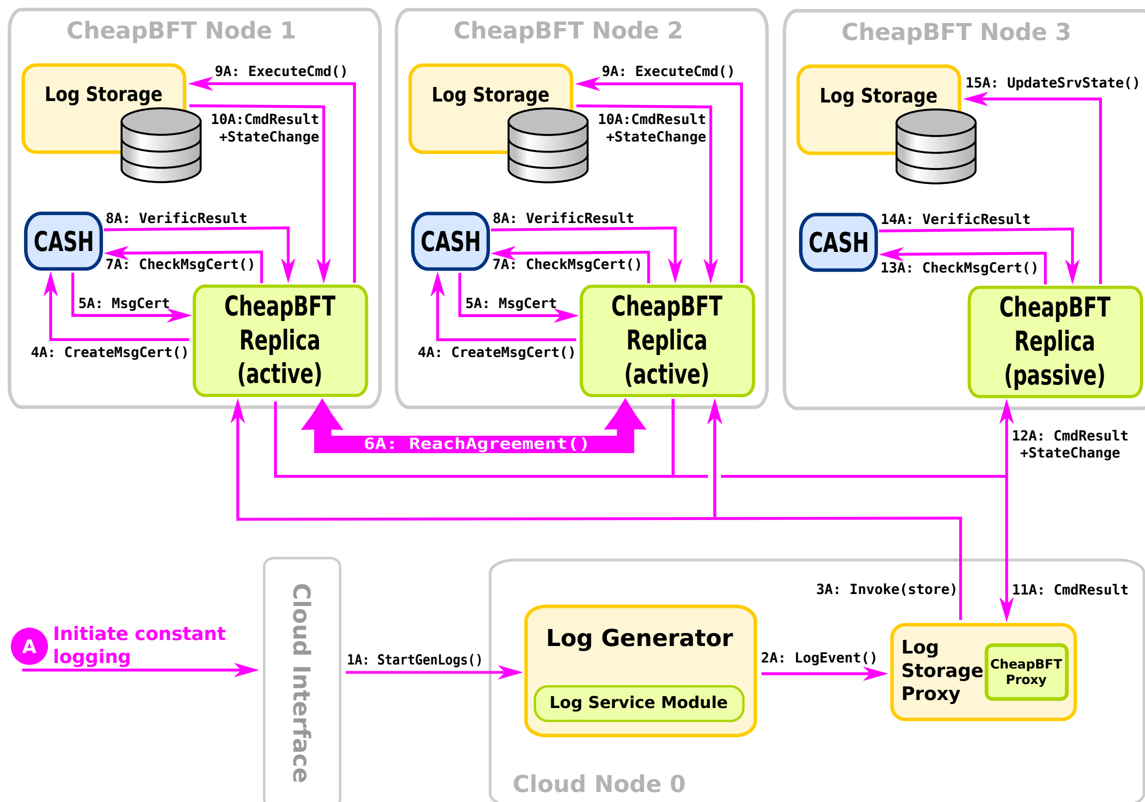

Figure 3.8: Resilient Log demo workflow (CheapTiny protocol)

The setup comprises four machines: Three machines equipped with CASH, the trusted hardware submodule, which host the replicas of the LogStorage, and one machine for the LogService core component in conjunction with the Log Generator that serves as a source of events. At the beginning of the demo, all LogService replicas are correct, that is, they are properly initialized and have all the same state. Consequently, CheapBFT is in the error-free mode, which means, that it runs a special protocol that only needs two active replicas while maintaining one passive replica as back-up. This protocol is called *CheapTiny* (Figure 3.8). The actual demo will proceed as follows:

(A) Initiate constant logging
The Log Generator is instructed to constantly log events, which causes a steady load on the Log Storage. Since the demo starts with the error-free case, log messages are only sent to the two active replicas. At this point, the active replicas have to ensure that they store the messages or, more general, that they process requests in the same order. All messages

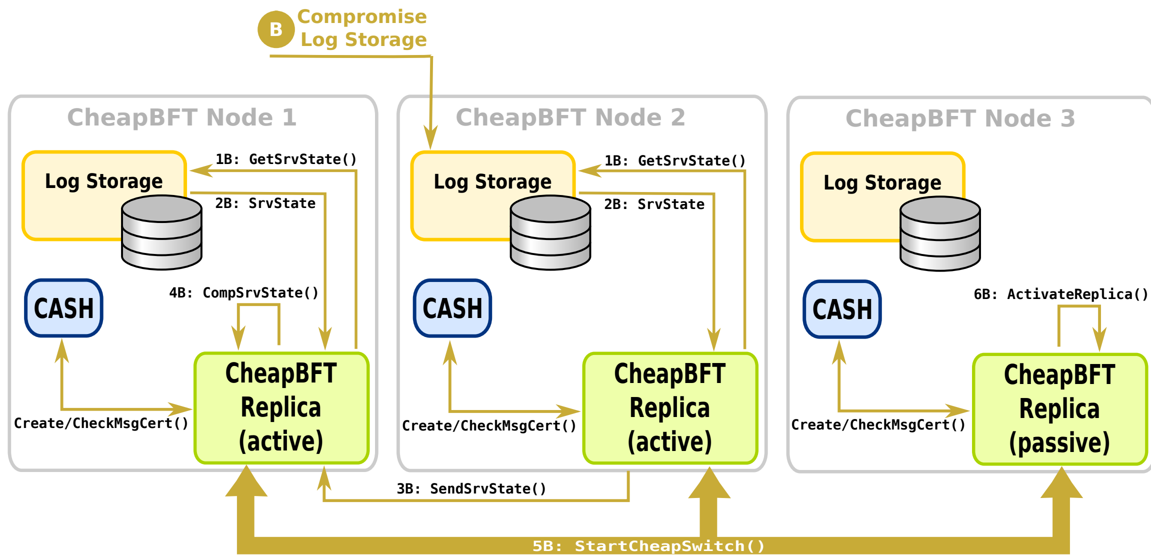

Figure 3.9: Resilient Log demo workflow (CheapSwitch protocol)

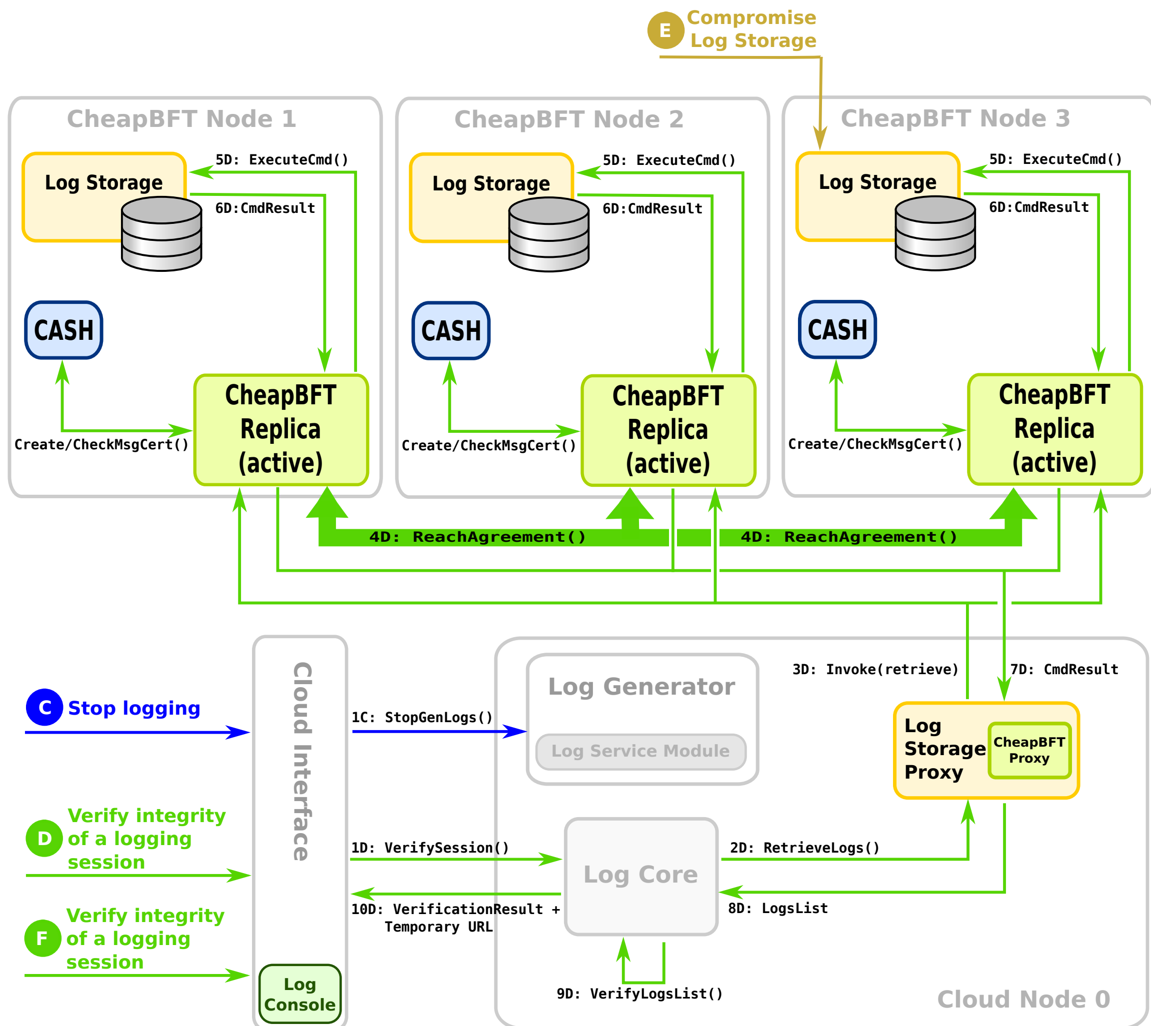

Figure 3.10: Resilient Log demo workflow (MinBFT protocol)

exchanged for the coordination necessary to reach an agreement among the replicas must contain a certificate which is created and verified by means of CASH. After the active replicas agreed on an order, they execute the command, that is, store the message and send the reply to the Log Service Proxy. Further, they inform the passive replica about the changes of the service state, which in turn updates its state accordingly.

While the demo is running, the Log Generator prints information about the generated log events every second. It also saves the generated log events in a local file to enable the tester to examine all created events. Further, the resource consumption (e. g., CPU and network) of the replicas is monitored and displayed during the whole run. Here it can be seen that the passive replica requires significantly less resources.

(B) Trying to compromise Log Storage
Eventually, an error is induced into a log file of one of the active replicas (Figure 3.9). After a short period of time, the error will be detected, because the replicas regularly exchange and compare their service states. If one of the replicas observes a difference between the states, it initiates a protocol switch, that is, CheapTiny is aborted and CheapSwitch is started. The protocol CheapSwitch is responsible for carrying out all steps required to switch to a consensus protocol that uses three active replicas, namely MinBFT (Figure 3.10). This is necessary, since two replicas are not enough to determine which one is correct and which one is faulty.

After the protocol switch, the resource consumption will increase, which can be witnessed through the resource monitoring.

(C) Stop logging
The Log Generator is stopped.

(D) Verify integrity of a logging session
Although one of the replicas is corrupted, the Log Service is still able to deliver a correct log file. To show that, a verification as described in Section 3.1.2.4 is carried out. Furthermore, the retrieved log file will be compared with the log file saved by the Log Generator.

(E) Compromise Log Storage
In this step, we inject the same error in one of the other replicas by manipulating its state. At this point, the number of compromised replicas exceeds the number of faults the system is designed to tolerate. The result is, that the CheapBFT subsystem can no longer deliver the correct results for requests from the Log Service.

(F) Verify integrity of a logging session
Again, the integrity of the logging session is tested. However, after the induction of two errors, the Log Storage returns an invalid log file and the verification fails.

3.2 TrustedInfrastructure Cloud Prototype

This section describes the demo prototype based on the subsystems TrustedDesktop, TrustedServer, TrustedObjectsManager, TrustedChannel, the architecture of the TrustedInfrastructure, and the demo storyline. TrustedDesktop is not being developed within TCloud, but it is used here for demonstration purposes.

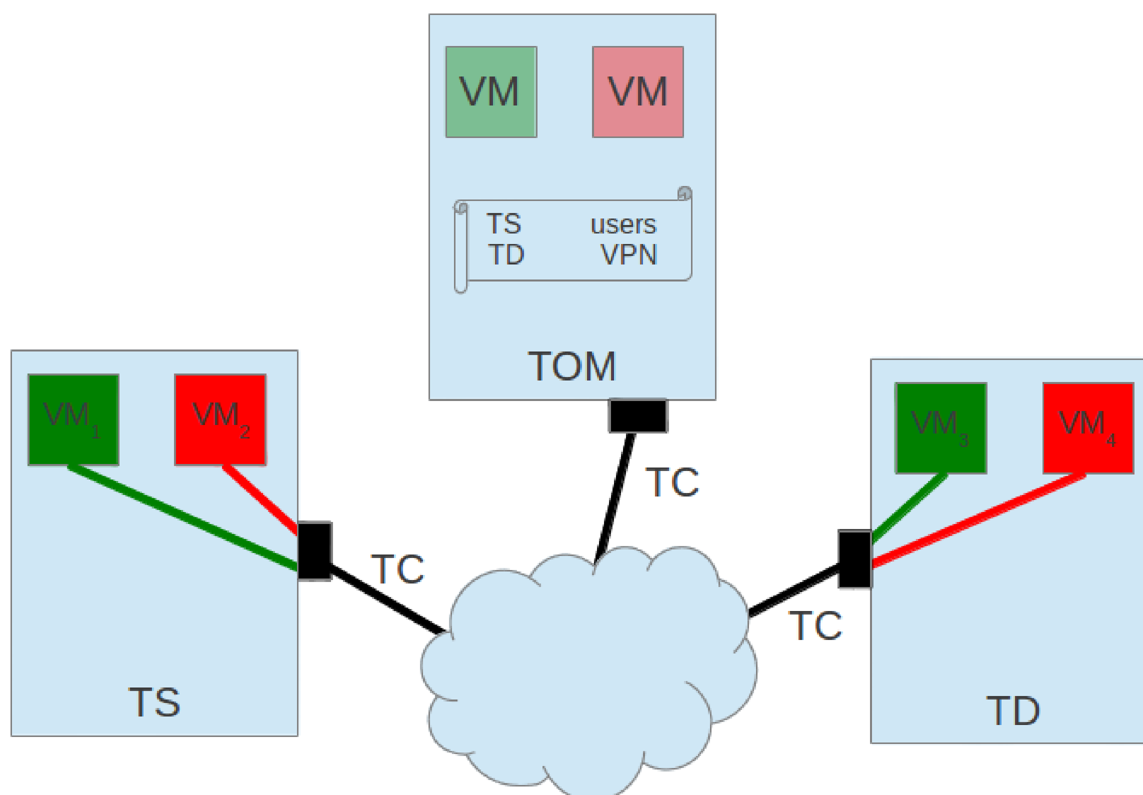

Figure 3.11: TrustedInfrastructure architecture

3.2.1 Architecture Overview

Figure 3.11 shows an overview of the involved components in the TrustedInfrastructure.

- TrustedObjectsManager (TOM): The TOM is the central management component of the trusted cloud infrastructure. The TOM manages the physical infrastructure including networks, services, and appliances (physical platforms).
  Since appliances remotely enforce a subset of the overall security policy, a permanent trusted channel between the TOM and its appliances is used for client authentication, to check their software configuration using attestation, to upload policy changes and software updates, and manage virtual machines in terms of starting, stopping, adding and removing. The TOM manages TrustedVirtualDomains (TVD) and inter-TVD information flow policies, performs key-management, and configures the managed TrustedServers (TS) and TrustedDesktops (TD) accordingly.
  As the central TVD Management component of a TVD-based infrastructure, the TOM provides the user interface (web-interface) to define TVDs and corresponding intra-TVD and inter-TVD information flow policies.

- TrustedServer (TS): The TS is based on the TURAYA SecurityKernel and provides isolation of virtual machines by linking them to TVDs. Domain-specific transparent encryption is applied by the TS, to prohibit information flows between TVDs. The TS is centrally managed by the TOM via the TrustedChannel, there is no necessity for a ”root”-account for administrators of the TS. The TS's integrity, that is, is ensured by the TPM-based Trusted Boot mechanism.

The focus of this component is to provide (together with TOM and TD) a trusted infrastructure for cloud applications.

- TrustedDesktop (TD): the TD is the client for the provided services of the TS. As the TS, the TD is centrally managed by the TOM, in terms of integrity, confidentiality and isolation. It is, as well as the TS, a trusted communication endpoint, since the applied TVD-policies are enforced end-to-end.
- TrustedChannel (TC): The TC is a TLS secured TCP/IP connection, extended by remote attestation capabilities, that allows message based communication between two endpoints.
  In particular, all security related information is exchanged via the TC, in terms of policies attached to TVDs, VPN configurations key material and certificates and virtual machine images.

3.2.2 Demo Storyline

In the envisaged demo scenario, the TOM is configured such that it provides two different virtual machines in two different TVDs ("green" and "red") for two different attached appliances (TS and TD). Any attached appliance has proven its integrity (through a TPM). These appliances are connected to the TOM via the TrustedChannel. The administration port on the TOM is implemented as a https web interface, where appliances, TVDs, virtual images, and VPNs between them, can be configured.

3.2.2.1 Abstract API

`startResult` ⇐ **`startCompartment(CompartmentID)`**
The TOM-administrator sends the command to start a dedicated compartment (identified uniquely by the CompartmentID) on TrustedServer.

`stopResult` ⇐ **`stopCompartment(CompartmentID)`**
The TOM-administrator sends the command to stop a dedicated compartment (identified uniquely by the CompartmentID) on TrustedServer.

3.2.2.2 Demonstration flow

Figure 3.12 depicts the steps of the demo and the interactions among the components involved, in order to start and stop a compartment on TrustedServer.

(A) Start Compartment on TS
The TOM-administrator initiates the boot-sequence of a stopped compartment on TrustedServer by calling `startCompartment(CompartmentID)`.

(B) Stop Compartment on TS
The TOM-administrator initiates the shutdown-sequence of a running compartment on TrustedServer by calling `stopCompartment(CompartmentID)`.

The TOM administrator logs in to the web interface and selects the connected TrustedServer appliance. The administrator then selects the first virtual machine, which is installed but not running on the TS and triggers the startup of this VM on the TS. Following the same procedure

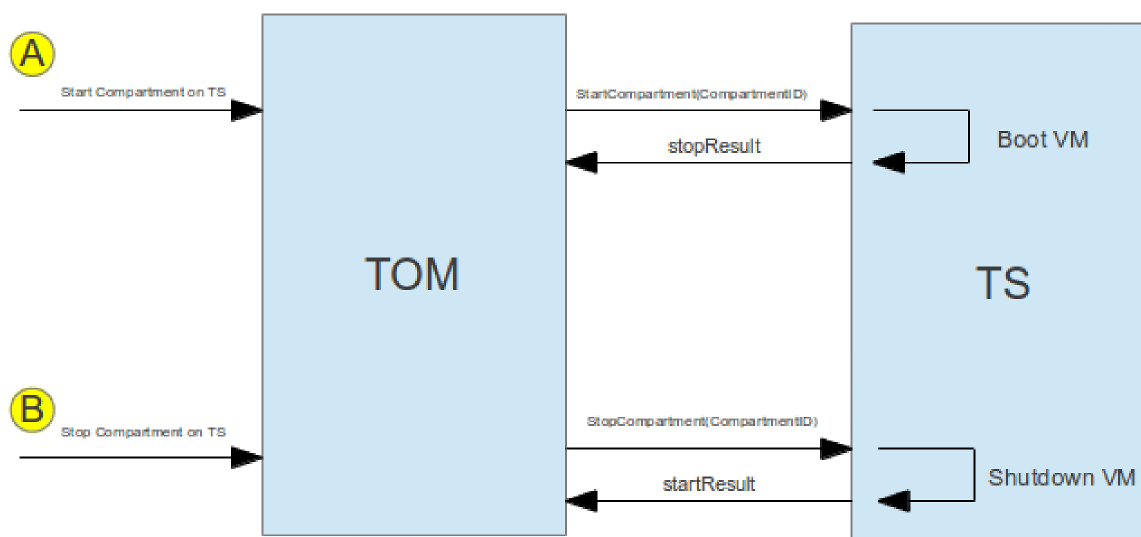

Figure 3.12: Start/Stop compartment demo workflow

for the other virtual machine, starts the second VM within another TVD on TS. The services pre-installed and configured within the VMs can now be used from within the same TVD.

Therefore, the TD-appliance is started and a pre-registered user logs in (see Figure 3.11). The TD user can now start an installed virtual machine within the same TVD one virtual machine of the TS belongs to. A VPN connection between the "green" virtual machines on TD and TS will be established automatically, such that the provided service can be used.
As an example, from within the green TVD (i.e., from the green VM on TD) an Apache-webserver can be reached, showing a static website on that server.
A SVN-server can be used from within the red TVD. This is demonstrated by a checkout.

In order to demonstrate the isolation of virtual machines, the TD-user attempts to reach the "green" Apache-webserver from the "red" virtual machine. This will fail because of the separation of TVDs.
Vice versa, trying to checkout the "red" SVN-repository from within the "green" virtual machine on TD will also fail.
In order to remotely shutdown the running virtual machines on the TS, the TOM-administrator selects the TrustedServer appliance. The administrator then selects the first virtual machine, which is running on the TS and triggers the shutdown of this VM on the TS. Following the same procedure for the other virtual machine, the administrator stops the second VM on TS.

3.3 Cloud-of-Clouds Prototype

This section describes the Cloud-of-Clouds prototype, that illustrates a subset of the results of work package 2.2. The objective of this prototype is to demonstrate how components provided in this work package, namely *BFT state machine replication* (BFT-SMaRt - initially described in D2.2.1 [ea11d] and delivered in D2.2.3 [ea11a]) and *cloud-of-clouds object storage* (DepSky - fully described and evaluated in D2.2.2 [ea12a]) to build a *cloud-backed file system* that does not require trust on individual cloud providers.

Our file system demonstrator is called C2FS (Cloud-of-Clouds File System), and it provides support for shareable file systems with near-POSIX semantics by using untrusted cloud storage

(e.g., Amazon S3, Google Store, Windows Azure Blob Storage and Rackspace Cloud Files) and computing (e.g., Amazon EC2) from unmodified public or private clouds.

3.3.1 Architecture overview

C2FS follows the same architecture of modern distributed and parallel file system (e.g., Hadoop FS or Ceph) in which metadata and access control are managed by a separated service not directly related to file block/object storage support [GVM00]. We leverage this architectural principle to use some WP2.2 TClouds components in this prototype.

More specifically, we use BFT-SMART as the replication middleware for a BFT coordination service based on the tuple spaces model [Gel85], that follows the idea FFCUL introduced in some previous work [BACF08]. This coordination service, called DEPSPACE 2 (or simply DEPSPACE), is used to store the file system namespace (i.e., the file and directory tree) and its associated metadata in a dependable way. Moreover, we store the data contained in the file in the DEPSKY, one of the cloud-of-clouds object storage systems being developed in TClouds. These two components are integrated in a Linux OS that mounts C2FS through the FUSE-J user-level file system wrapper [Lev12]. Figure 3.13 shows an overview of the basic building blocks of C2FS.

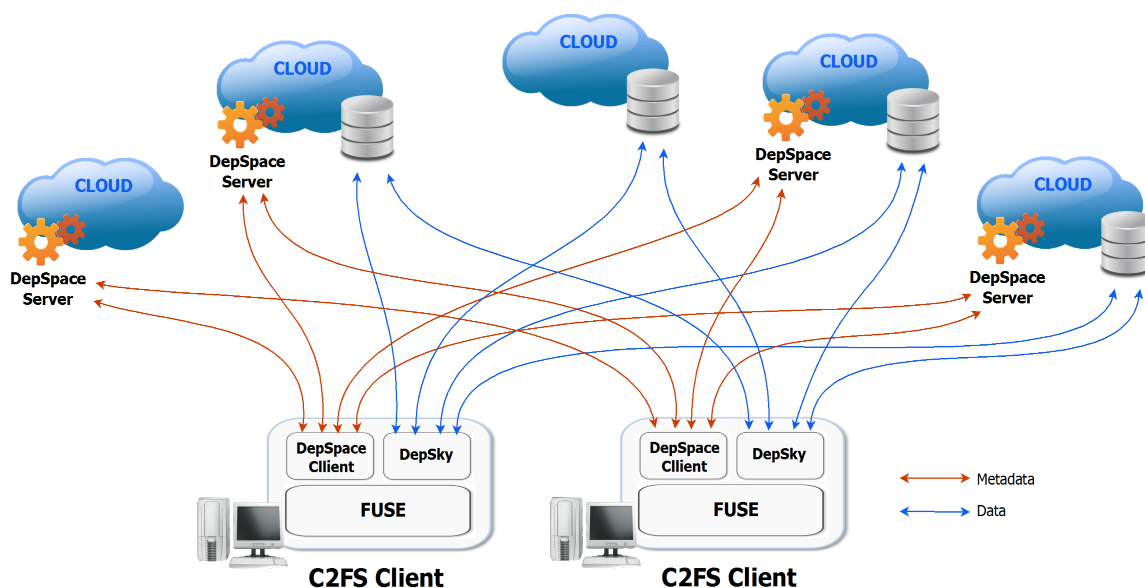

Figure 3.13: C2FS basic architecture.

There are many challenges related with the design of cloud-backed file systems such as C2FS (that we expect to fully address in the third year of the project, and describe in a future WP 2.2 deliverable), but there are two fundamental challenges that define how we engineered the system, and their knowledge is important for the understanding of how the system operates.

The first challenge is how to avoid the big latencies of accessing cloud services most of the time. More specifically, we want to avoid both (1) the latency of accessing a BFT-replicated service (DEPSPACE) on each read/update on a file metadata (100s of milliseconds) and (2) the latency of reading and writing data on the clouds (seconds), as most as possible. To do that, we extensively use both RAM and disk cache at the C2FS clients. As it will be shown in the demonstration, this feature dramatically improves the ergonomics of the system.

The second fundamental challenge is how we write and read files considering the two separated WP2.2 components of the system (i.e., DEPSPACE and DEPSKY) without endangering the consistency of the file system. We address this challenge with the following strategy. When a new file (or a new version of a file) is written, the C2FS client first write its data as an object on the Cloud-of-Clouds object store (DEPSKY) and then, when the write is concluded, the coordination service (DEPSPACE) is updated with the new file size, hash, and access key (data unit name in DEPSKY parlance – see D2.2.2 [ea12a]). For reading a file, the C2FS client first resolves the file name on the coordination service, obtaining the key for accessing the file and its hash, and then read the stored object that matches the metadata. This strategy ensures that a file (or file version) is made visible only after it is available for reading, and vice-versa.

3.3.2 Demo storyline

As already discussed, in the Cloud-of-Clouds demo we will be showing the capabilities of a file system that makes use of BFT state machine replication and a resilient object storage, two important contributions of WP 2.2. The main objective of the demo is to show the capabilities of the system, i.e., that it can be used to store and share files efficiently, and demonstrate its resiliency considering threats such as intrusion on a client, corruption and/or destruction of some information stored in a cloud provider and the failure of some replica of the coordination service used to store the file system metadata. We want to demonstrate these features with the following storyline.

(A) Show the startup of DEPSPACE coordination service replicas in one or more cloud providers[3].

(B) Show the C2FS filesystem being mounted in two laptops, from now on designated clients A and B.

(C) Both A and B create some files and directories on the system (using commands such as `cp`, `ls`, `mkdir` and `cd`), to show the usability of the cloud-backed file system.

(D) Open the web-based management console of the cloud storage providers being used to show that the files are being stored on the clouds ensuring privacy (i.e., they are encrypted) and cost-efficiency (i.e., only a portion of the file is stored on – and charged by – each provider).

(E) Client A creates a specific file F, and gives B permission to access it.

(F) Client B accesses F and updates its contents, maintaining the file open. During this period, A can read F, but cannot modify it (the file is locked by B).

(G) Kill one DEPSPACE replica to show the fault-tolerance provided by the BFT-SMART replication library.

(H) Delete some file from a cloud provider, and show that the file is still available; and/or corrupt some file from a cloud provider, and show that integrity is preserved.

(I) Assume client B compromised and show that it can corrupt/delete all files it has access (including F, created by A and shared with B), but cannot destroy any other files in the system.

[3]This depends on the connectivity we will be experiencing on the demonstration site. In the worst case we plan to deploy the replicas on a single remote cloud.

As a disclaimer, we would like to remark that this storyline is still a work in progress.

3.3.2.1 Abstract API

C2FS uses the following abstract API[4]:

`descriptor` ⇐ **`openFile(name, mode)`**
This operation is used to open a file write with a given name and mode (read, write or read-write). If the operation is successfully executed, it returns a valid descriptor for the file.

`result` ⇐ **`closeFile(descriptor)`**
This operation is used to close a previously opened file (represented by a descriptor). The result is a boolean indicating if the operation is well succeeded or not.

`result` ⇐ **`writeFile(descriptor, data)`**
This operation is used to write some data in a opened file with a given descriptor. The result is a boolean indicating if the operation is well succeeded or not.

`data` ⇐ **`readFile(descriptor)`**
This operation is used to read the contents of a opened file with a given descriptor.

`result` ⇐ **`deleteFile(name)`**
This operation is used destroy a file with a given name. The result is a boolean indicating if the operation is well succeeded or not.

`result` ⇐ **`changePermissionsFile(name, permission)`**
This operation is used to change the sharing permissions of a file with a given name. The result is a boolean indicating if the operation is well succeeded or not.

`result` ⇐ **`createDir(name)`**
This operation is used to create a directory with the given name. The result is a boolean indicating if the operation is well succeeded or not.

`result` ⇐ **`deleteDir(name)`**
This operation is used to delete a directory with the given name. The result is a boolean indicating if the operation is well succeeded or not.

`list` ⇐ **`listDir(name)`**
This operation is used for obtaining the list of files and directories on a given directory.

As may be noted, this abstract API maps directly with the steps on our demo storyline.

3.3.2.2 Demonstration flow

Given the large amounts of steps on our storyline and the large number of component interactions on each step, we opted to show some illustrative examples of operations and commands we will be executing during the demonstration and how they use the main components of the prototype, namely, FUSE-J, DEPSPACE (BFT-SMART) and DEPSKY. Figures 3.14 and 3.15

[4] The currently implemented API is a FUSE-J interface as defined in [Lev12], which includes all features and low-level details of a file system, and is not necessary to understand the C2FS demo.

shows the operation invoked in these components when some basic commands are executed by some user. These figures, together with the storyline already described can give an idea of the basic component functionalities being shown in the Cloud-of-Clouds demo.

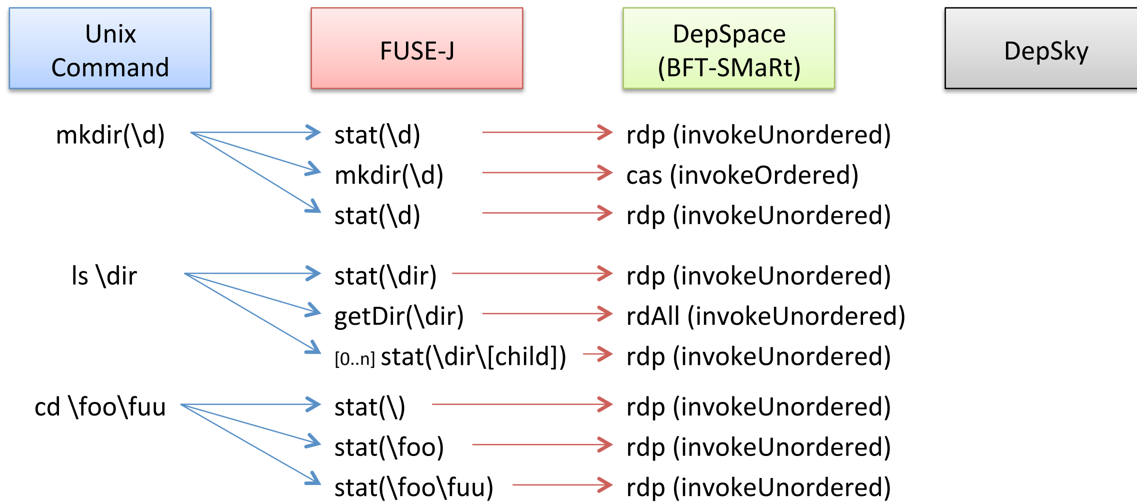

Figure 3.14: The flow of invocations in different components for directory-related operations.

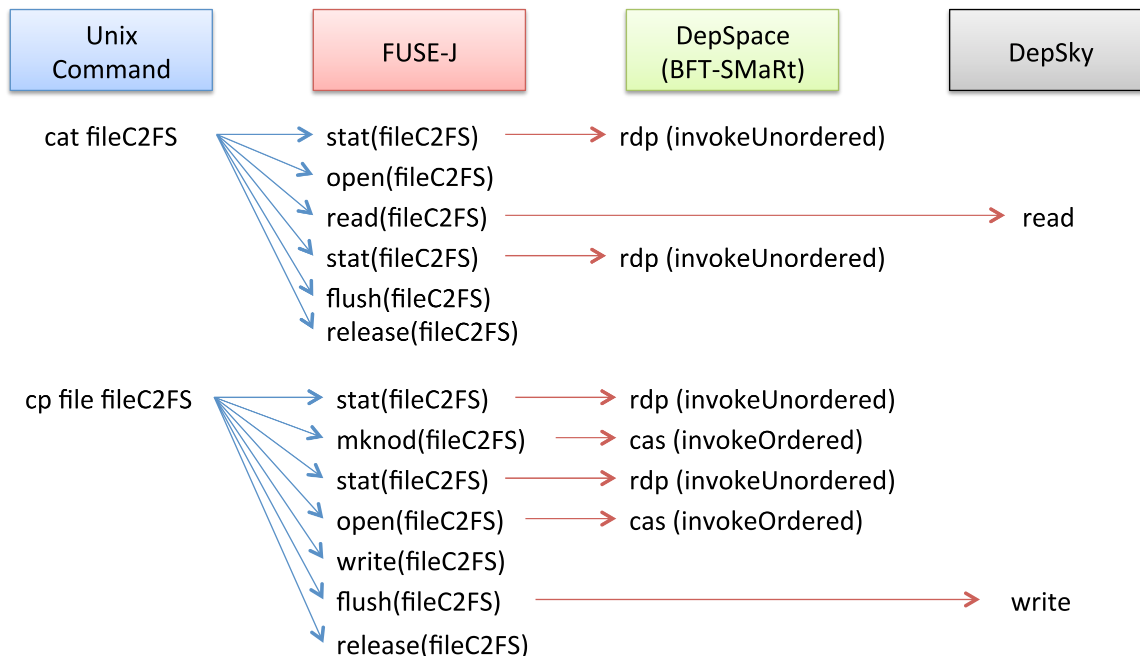

Figure 3.15: The flow of invocations in different components when data-intensive commands are executed.

3.4 Other Prototypes

In this chapter, we will describe the prototypes that will not be presented as part of the TClouds second year's demonstration.

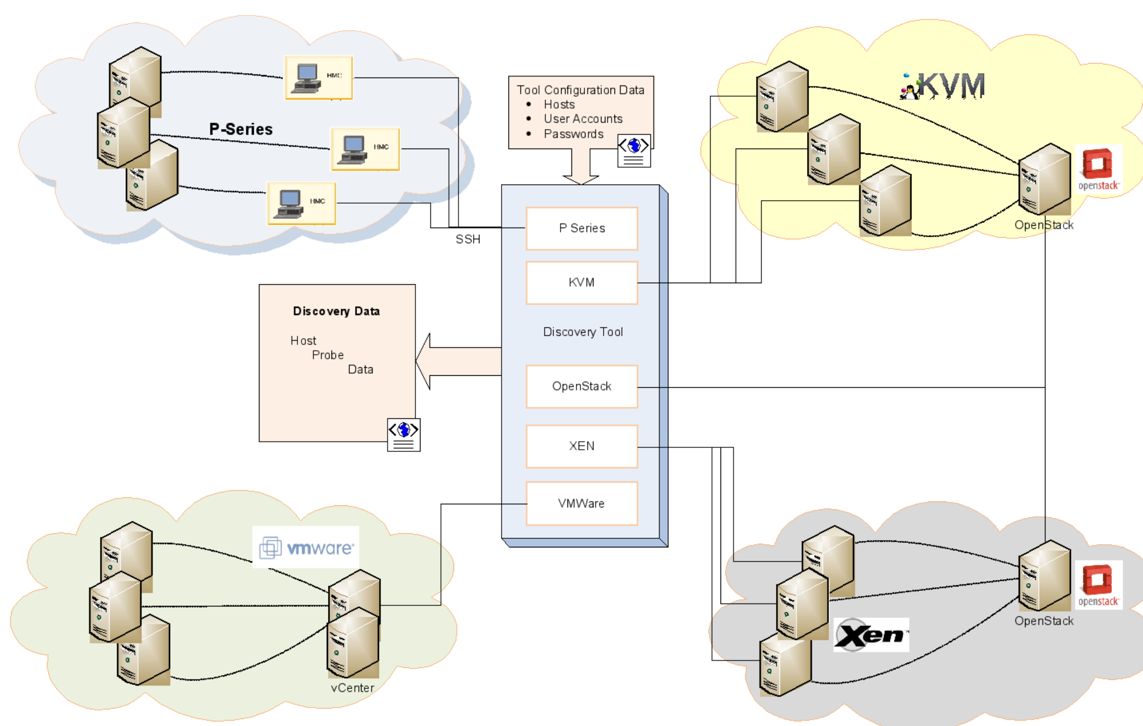

Figure 3.16: SAVE: Architecture Overview of Discovery Component

3.4.1 Security Assurance of Virtualized Environments *(SAVE)*

3.4.1.1 Architecture Overview

SAVE is a tool developed at IBM research for extracting configuration data from multiple virtualization environments, transforming the data into a normalized graph representation, and subsequent analysis of its security properties. IBM will integrate and adapt this technology for the demonstrator based on OpenStack, in order to validate isolation of cloud users.

SAVE is structured into two components: *Discovery* and *Analysis*.

Discovery. The data discovery phase is used to collect virtualization data from a number of heterogeneous environments (cf. Figure 3.16). It is configured with the set of hosts to query and basic authentication information required to access the data. The tool uses simple heuristics to identify the environment in which each host is situated. Based on the environment (HMC, VMware, XEN, libvirt) the appropriate probe is selected. The tool outputs a single XML file containing all virtualization information that was discovered. This XML file is used as input into the data analysis components.

Analysis. The analysis component takes the discovery XML format as input, in addition to a specification of traversal rules and a security policy (cf. Figure 3.17). The traversal rules are formulated in XML and specify the information flow and trust assumptions about elements of the virtualized infrastructure in general. The security policy is specified in a logical term language called VALID, which expresses attack states that violate the high-level security goals, in a nutshell. VALID is language developed with a formal methods background and based on the AVISPA Intermediate Format (IF) and ASlan, two languages widely used in model checking.

For the validation of the discovered infrastructure against the security policy, SAVE will

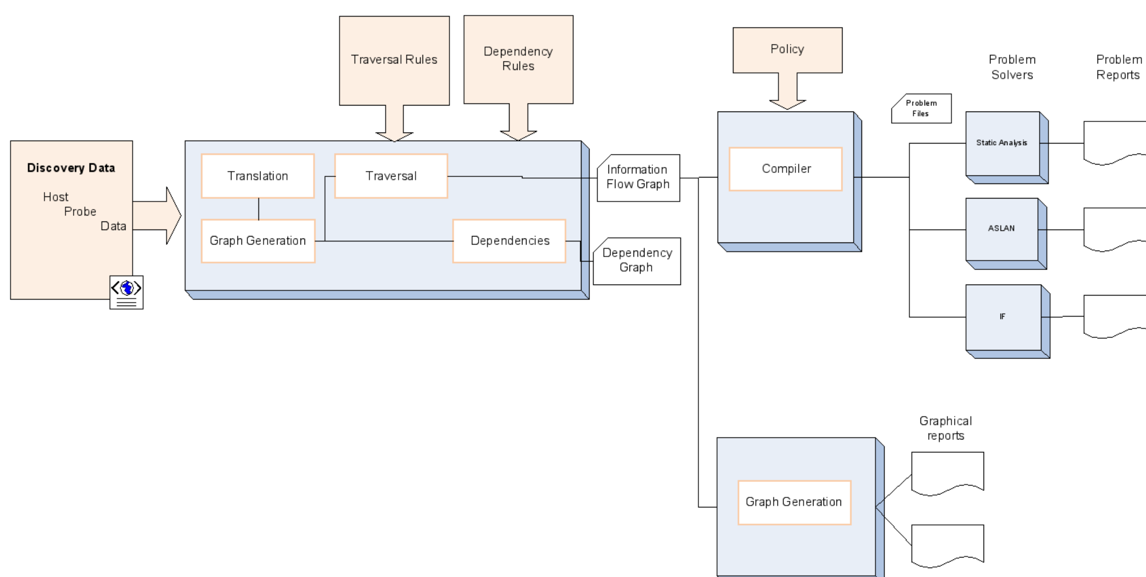

Figure 3.17: SAVE: Architecture Overview of Analysis Component

compile problem statements for model checkers in their respective language, such as IF, ASlan or First-Order Logic (FOL). It also takes proprietary output format of the model checkers as feedback and evaluates their output with respect to the realization model to find alarm states.

The general persistent output of the SAVE analysis may be textual, as fault logs, or a standard graph format (GEXF), in order to render big-picture views on the topology and fine-grained views on problem areas for diagnosis.

3.4.2 Ontology-based reasoner: Libvirt With Trusted Virtual Domains

3.4.2.1 Introduction

This prototype is an enhancement of Libvirt that allows to define and configure a *Trusted Virtual Domain* (TVD) as the aggregation of virtual machines. It includes an extension of the XML configuration language to model the network and completes the support for the management of Open vSwitch bridges. The implementation of the TVD concept is based on the proposal made by Bussani *et al.* in [BGJ$^+$05] (Table 2, Case 1) that foresees the usage of VLAN[5] + IPSec[6] for the communication channels.

Our subsystem anticipates a feature that will be presented in the Year 3 prototype. We decided to present this work in advance so that all partners start thinking about how to take advantage of it to solve existing security issues. For this reason, we provide in this document a brief overview of the architecture and some details about the implementation, and we will deliver the missing parts next year.

3.4.2.2 Architecture

Our enhancement allows to configure the scenario depicted in Figure 3.18, where two TVDs are defined (*TVD-Red* and *TVD-Green*) and TVD members, the virtual machines, are spanned in two physical hosts (*Host 1* and *Host 2*).

[5] http://standards.ieee.org/findstds/standard/802.1Q-2005.html
[6] http://tools.ietf.org/html/rfc4301

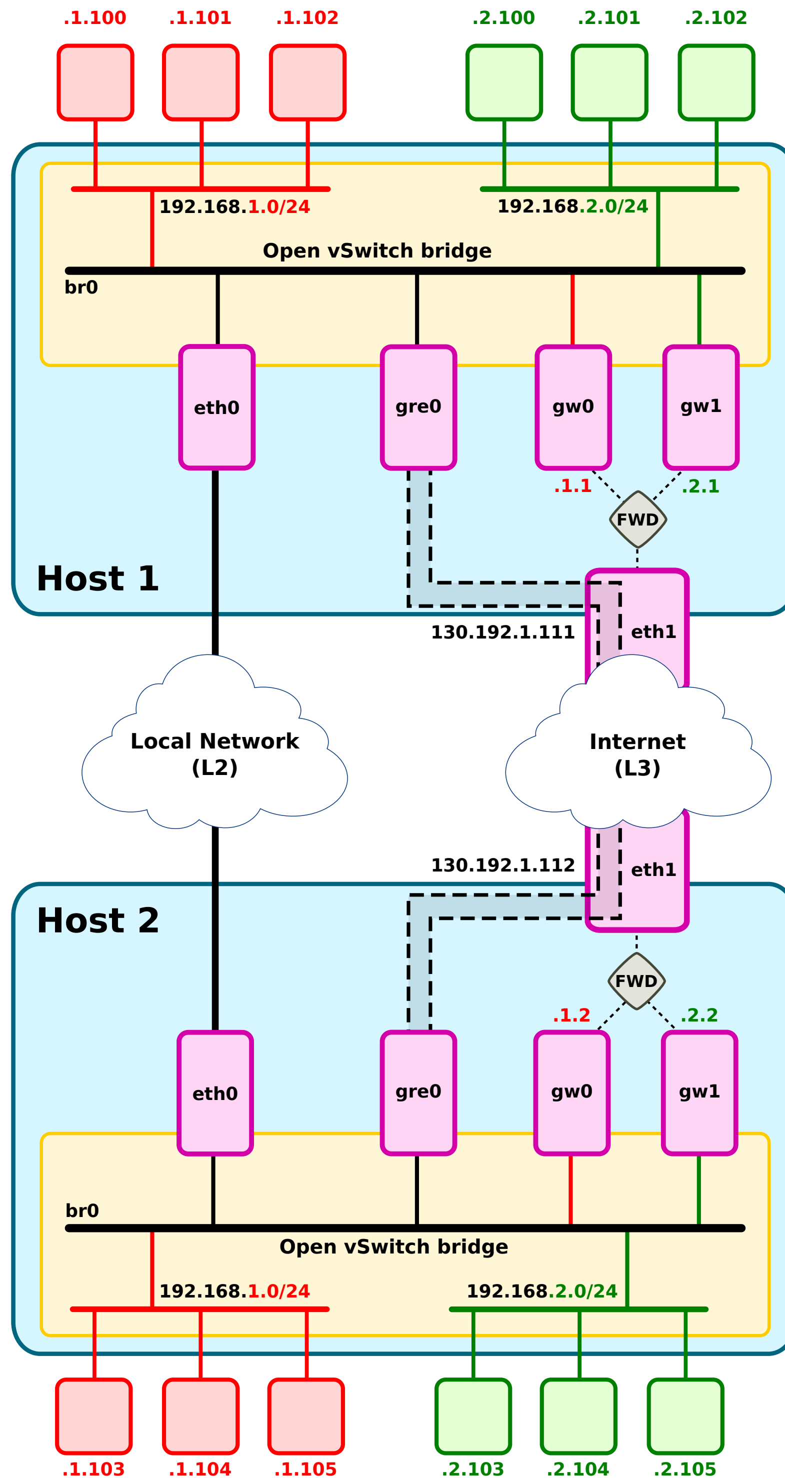

Figure 3.18: TVD scenario implemented using Open vSwitch bridges

In order to enforce the isolation property of TVDs, each physical host is responsible to separate the local communication between virtual machines so that data can be exchanged only among members of the same TVD. Further, physical hosts must collaborate to prevent that a virtual machine of a TVD in the local host can communicate with a virtual machine of a different TVD in the remote host.

For the implementation of these features, we decided to use Open vSwitch [OvT12] as the bridge that connects together the back-end interfaces of virtual machines. This software allows to associate a VLAN tag to each interface (also called *access port*), so that Ethernet frames coming from the virtual machine are encapsulated with this tag. Then, these frames can be received, in the local host, only by the virtual machines whose interface has been assigned the same tag. With this separation, virtual machines of the same TVD believe to be connected through an isolated network, even if they share the same bridge with virtual machines of different TVDs.

Moreover, when a virtual machine needs to contact a member in the remote host, the local host must transmit Ethernet frames to their destination over the physical network. This can be done alternatively by adding to the bridges the interfaces connected to the physical network (e.g. *eth0*), if hosts are L2 adjacent, or by creating an IP tunnel (e.g with GRE) and adding to them the endpoints (e.g. *gre0*), if hosts are L3 adjacent. In both cases, these interfaces are configured as *trunk ports* in order to preserve the tag encapsulated in the packets, so that the isolation between TVDs can be uniformly enforced by all hosts.

Further, we guarantee that the tag of a packet has not been tampered with by assuming that, when physical hosts are L2 adjacent, the physical network is trusted, i.e., an attacker can not modify the tag applied to Ethernet frames. Instead, if physical hosts are L3 adjacent, we protect IP packets through an IPSec connection that guarantees confidentiality and integrity on their payload.

From the Layer 3 perspective, each TVD has assigned a class C network of the private address space: `192.168.1.0/24` for the *TVD-Red* and `192.168.2.0/24` for the *TVD-Green*. Each host assigns an IP address to the virtual machines through a management interface, unique for the TVD, (*gw0* for *TVD-Red* and *gw1* for *TVD-Green*) and an instance of the *Dnsmasq*[7] daemon (executed by Libvirt) that listens to it. In order to avoid collisions, each host must have assigned its own range of IP addresses for each TVD.

Our prototype also allows to connect a TVD to the Internet. When a specific configuration is provided, a virtual machine can send packets through the management interface, that acts as a gateway, and then the local host forwards them to the destination through the public interface (e.g. *eth1*) by using the IP masquerading technique. Since all the management interfaces of a TVD are visible by members of that TVD, a virtual machine can send packets to Internet even if there is no connectivity in the local host, by simply specifying as a gateway in the routing table the IP address of the management interface in a host with the forwarding enabled.

Our prototype allows to describe the above network configuration by extending the Libvirt XML language and allows to enforce it by executing Open vSwitch commands. In the following, we will provide more details about the implementation.

3.4.2.3 Implementation

In this section, we explain how to specify the configuration for the scenario depicted in Figure 3.18 and how Libvirt enforces the isolation among TVDs by translating this configuration in low-level commands to Open vSwitch.

[7] http://www.thekelleys.org.uk/dnsmasq/doc.html

XML language extension. The current format of the XML language (Libvirt version 0.9.11) does not allow to correctly specify the configuration of TVDs. While there are some elements that can be used to isolate the network in groups of virtual machines (the *portgroup* and *virtualport* elements), they appear to be not adequate to describe the configuration at Layer 3. Indeed, if multiple portgroups are defined in the same network object, they should have assigned their own gateway and range of IP addresses but, in the version that we evaluated, Libvirt allows to define these settings only once for each network. For this reason we decided to split this configuration in three main steps:

1 Definition of the *TVD backbone network*

2 Definition of *TVD specific networks* and assignment to the TVD backbone network

3 Assignment of a virtual machine network interface to a TVD specific network.

The *TVD backbone network*, represented in Figure 3.18 with black lines, connects together, through the Open vSwitch bridge `br0`, the interfaces defined in the TVD specific network, the physical host interfaces and, eventually, one or more tunnels to other physical hosts.

The Listing 3.1 shows the XML description of the TVD backbone network in our scenario for *Host 1*. We want to focus the attention on the attribute `type` of the element `bridge`. Such attribute has been introduced by our prototype in order to instruct Libvirt to create an Open vSwitch bridge (attribute value `openvswitch`) instead of a standard Linux bridge (attribute value `linuxbridge`). To maintain the compatibility with previous versions, this attribute has been defined as optional: if not specified, Libvirt creates a bridge of the latter type. Further, the Listing 3.1 shows the new element `tunnel`, introduced by our prototype, that allows to configure an IP tunnel between L3-adjacent physical hosts. Within this element it is possible to specify the IP address of the remote host, the name of the interface that acts as tunnel endpoint and the list of allowed VLAN tags.

```
<network>
  <name>tvd_backbone</name>
  <bridge name="br0"  type="openvswitch"    />
  <tunnel>
    <remoteip address="130.192.1.112" />
    <device name="gre0" />
    <allow vlans="1,2" />
  </tunnel>
</network>
```

Listing 3.1: XML configuration file of the *TVD backbone network* in *Host 1*

TVD specific networks, represented with red and green lines respectively for the *TVD-Red* and *TVD-Green*, consist of the back-end interfaces of virtual machines and a management interface, which are plugged in the Open vSwitch bridge of the TVD backbone network.

The Listing 3.2 illustrates the XML definition of the TVD specific network `tvd_red` for the *TVD-Red* TVD in *Host 1*. Also in this case, there are some configuration elements introduced by our prototype. First, in the `bridge` element we introduced the attributes `type` and `sourcebridge`. The former has been already described for the TVD backbone network while the latter allows to specify the name of the Open vSwitch bridge where interfaces of the TVD specific network will be plugged in. When a network of this type is activated, Libvirt creates the management interface `gw0` and plugs it to `br0`. Secondly, the element `virtualport` now supports the attribute `vlantag` that allows to specify the VLAN tag for the interfaces defined

in the TVD specific network. It is also worth to mention that tvd_red is allowed to connect to Internet through the *eth1* public interface, as stated in the unmodified element forward.

```
<network>
  <name>tvd_red</name>
  <forward mode="nat" dev="eth1">
  <bridge name="gw0"  type="openvswitch"   sourcebridge="br0"   />
  <ip address="192.168.1.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.1.100" end="192.168.1.199" />
    </dhcp>
  </ip>
  <portgroup name="red" default="yes">
    <virtualport type="openvswitch">
      <parameters  vlantag="1"   />
    </virtualport>
  </portgroup>
</network>
```

Listing 3.2: XML configuration file of the *TVD specific network* for *TVD-Red* in *Host 1*

The assignment of a virtual machine network interface to a TVD specific network is the most simple task, as our prototype does not require any modification on the domain XML file. The Listing 3.3 shows that in order to assign a network interface to the *TVD-Red* TVD it is sufficient to specify the name of the TVD specific network tvd_red in the source subelement of interface.

```
<domain>
  ...
  <devices>
    ...
    <interface type="network">
      <source network="tvd_red">
    </interface>
    ...
  </devices>
  ...
</domain>
```

Listing 3.3: XML configuration file of a virtual machine that is member of *TVD-Red*

Open vSwitch administrative commands. After Libvirt parses the configuration files before a virtual network is activated, it performs the commands to Open vSwitch in order to enforce the isolation among TVDs. When the TVD backbone network is started, Libvirt creates the Open vSwitch bridge by executing the command:

```
ovs-vsctl add-br br0
```

This bridge can be connected alternatively to the Local Network (L2) or to Internet (L3) respectively through the eth0 (not currently supported) or the gre0 interface. The latter, which is an endpoint of a tunnel whose parameters are specified in the tunnel element of configuration file for this network, is created and added to the bridge br0 by Libvirt through the command (for *Host 1*):

```
ovs-vsctl -- --may-exist add-port br0 gre0 trunks=1,2 -- set Interface gre0
type=gre options:remote_ip=130.192.1.112
```

The list of VLAN tags specified in the command option `trunks=1,2` and also in Listing 3.1 allows to filter the Ethernet frames coming from the remote tunnel endpoint. If, for example, a compromised host starts sending packets with an arbitrary tag, benign hosts can prevent that this host communicates with TVDs for which it is not allowed to start virtual machines, by discarding Ethernet frames whose tag is not in the list.

When a TVD specific network is started, Libvirt creates the management interface by using the "fake bridge" feature of Open vSwitch. In the following, there is the command executed to create the fake bridge `gw0` for `tvd_red` with VLAN tag 1 in *Host 1*:

```
ovs-vsctl add-br gw0 br0 1
```

Regarding the assignment of virtual machine network interfaces, Libvirt plugs them to the fake bridge of a TVD specific network (e.g. `gw0` for the `tvd_red` network) by executing:

```
ovs-vsctl -- --may-exist add-port gw0 vnet0 tag=1 [...]
```

As the fake bridge feature of Open vSwitch is only an abstraction introduced for compatibility reasons to allow software to deal with standard Linux bridges, the command above, with non-relevant options omitted, causes the interface `vnet0` to be attached to `br0` of the TVD backbone network.

3.5 Mapping legal and application requirements to subsystems and prototypes

Table 3.2 maps the legal and application requirements stated in Chapter 2 to TClouds subsystems listed in Table 3.1. An 'X' in a cell means that the subsystem in that column *contributes* to the satisfaction of the requirement in that row. Multiple subsystems can contribute to the satisfaction of a single requirement. Table 3.2 does not show if the requirements are fully satisfied or not.

In the following the mapping is presented by prototype: for each on them which subsystems form it and for each subsystem, which requirements it satisfies.

Requirements	Resource-efficient BFT (CheapBFT)	Simple Key/Value Store (memcached)	Secure Block Storage (SBS)	Secure VM Instances	TrustedServer	Log Service	State Machine Replication (BFT-SMaRt)	Fault-tolerant Workflow Execution	Resilient Object Storage	Confidentiality Proxy for S3	Access Control as a Service (ACaaS)	TrustedObjects Manager (TOM)	Trusted Management Channel	Ontology-based Reasoner	Automated Validation (SAVE)	Remote Attestation Service
LREQ1 - Confidentiality of personal data			X	X	X	X			X	X					X	
LREQ2 - Availability and Integrity of personal data	X		X		X		X		X					X		
LREQ3 - Control of location and responsible provider						X					X	X				
LREQ4 - Unlinkability and Intervenability						X			X							
LREQ5 - Transparency for the customer						X									X	X
AHSECREQ1 - Confidentiality of stored and transmitted data			X	X	X				X	X				X	X	
AHSECREQ2 - Integrity of stored and transmitted data	X		X		X		X		X							
AHSECREQ3 - Integrity of the application	X	X					X	X								
AHSECREQ4 - Availability of stored and transmitted data	X						X		X			X				
AHSECREQ5 - Availability of the application	X						X	X				X		X		
AHSECREQ6 - Non repudiation						X										X
AHSECREQ7 - Accountability						X										X
AHSECREQ8 - Data source authentication	X						X							X		
AHPRIVREQ1 - Unlinkability and Anonymization of data flow																
ASSECREQ1 - Trustworthy Audit						X										X
ASSECREQ2 - Trustworthy Infrastructure		X	X	X	X							X	X	X		X
ASSECREQ3 - Trustworthy Persistence Engine	X						X		X							
ASSECREQ4 - Resilient	X						X	X	X							
ASSECREQ5 - Trustworthy communications	X						X						X	X		
ASSECREQ6 - High performance & Scalable	X	X														

Table 3.2: List of TClouds subsystems and mapping to requirements

PROTOTYPE	Trustworthy OpenStack
RESOURCE-EFFICIENT BFT (CHEAPBFT)	REQUIREMENTS: LREQ2 - **Availability and Integrity of personal data** – The CheapBFT subsystem improves the availability by providing fault-tolerance, i.e. it can mask arbitrary faults in the cloud infrastructure. This also includes arbitrary alterations in the transmitted data, ensuring the integrity of the exchanged information. AHSECREQ2 - **Integrity of stored and transmitted data** – Integrity of application state is checked against other replicas when externalised, i.e. transmitted to the client. Modifications can be detected and masked, thus ensuring integrity. AHSECREQ3 - **Integrity of the application** – The integrity of the application instance is ensured by state machine replication. If the application misbehaves on one of the replicas in a way that is externally visible, the CheapBFT subsystem can detect and mask this fault. AHSECREQ4 - **Availability of stored and transmitted data** – As the CheapBFT subsystem is based on replication, it does not only tolerate manipulation of data, but also the complete outages of single replicas. AHSECREQ5 - **Availability of the application** – Since the replicas can run any application that can be modelled as a deterministic state machine, most software used in practice could be ported to use CheapBFT for improved availability. Data storage is actually just one of the simpler applications that can be built on top of the CheapBFT protocol. AHSECREQ8 - **Data source authentication** – CheapBFT is based on the assumption that the clients and servers mutually authenticate each other so that the source of each transmitted message is verifiable. ASSECREQ3 - **Trustworthy Persistence Engine** – If the persistence engine was built as a replicated application on top of CheapBFT, the integrity and availability requirements can be addressed by the properties the communication protocol provides. ASSECREQ4 - **Resilient** – CheapBFT can be used mask singular failures in the infrastructure and also in instances of the persistence engine, thus improving the overall resilience of the system. ASSECREQ5 - **Trustworthy communications** – All messages exchange using the CheapBFT subsystem are cryptographically signed, thus any manipulation can be detected and prevented from having an impact. ASSECREQ6 - **High performance & Scalable** – The CheapBFT protocol uses a trusted and fast hardware component for secure message signing. Compared to other BFT systems, a lower number of replicas are required to provide the same level of fault-tolerance and thus also the amount of messages that have to be exchanged is decreased. Therefore it also becomes easier to build fast and scalable systems based on CheapBFT.

PROTOTYPE	Trustworthy OpenStack
SECURE BLOCK STORAGE (SBS)	REQUIREMENTS: **LREQ1 - Confidentiality of personal data** – SBS builds the foundation for transparent encryption and authentication of data for legacy VMs that are not aware of cryptography. SBS transparently encrypts storage used by customer's VM so that data that is processed in the cloud management layer or stored on cloud storage is always encrypted and hence provides confidentiality since no plain text data leaves the cryptographically protected VMs (*DomC-DomU* couple, as explained in 3.1.2.3). **LREQ2 - Availability and Integrity of personal data** – In the same way as for LREQ1 -, integrity of personal data is satisfied: data is protected so that tampering becomes evident and integrity can be verified. This currently is only supported for (block) storage in the Y2 prototype but the very same key stored in the crypto-VM (*DomC*) can easily be used to wrap network traffic in an SSL tunnel. However, this is already possible from within the user-VM (*DomU*) by establishing an SSL channel and letting the separate DomC do the encryption and decryption of the encrypted and authenticated connection. Availability cannot be satisfied, because the cloud administrator is in control of the encrypted data. **AHSECREQ1 - Confidentiality of stored and transmitted data** – Cf. AHSECREQ1 -. **AHSECREQ2 - Integrity of stored and transmitted data** – Cf. LREQ2 -. **ASSECREQ2 - Trustworthy Infrastructure** – The keys are bound to a particular software configuration of SBS and the hypervisor. This *implicit attestation* assures that valuable key material can only be unwrapped (revealed) by a known-good software configuration that is trusted by and known to the customer.

PROTOTYPE	Trustworthy OpenStack
LOGSERVICE	REQUIREMENTS: LREQ1 - **Confidentiality of personal data** – This subsystem allows to detect attackers intrusions that caused the leakage of users' personal data. LREQ3 - **Control of location and responsible provider** – This subsystem can be used to log transfers of data between different Cloud datacenter sites. LREQ4 - **Unlinkability and Intervenability** – This subsystem implements techniques to prevent logs be associated to a real user. LREQ5 - **Transparency for the customer** – This subsystem can be used, together with the Remote Attestation Service, to reliably track operations performed by the Cloud Provider on the infrastructure and logs produced by users' applications. AHSECREQ6 - **Non repudiation** – This subsystem ensures that an attacker can not deny to have performed a specific action by guaranteeing, with the Remote Attestation Service, integrity of logs and the proof that the application is trusted to properly record the actions. AHSECREQ7 - **Accountability** – This subsystem ensures that an attacker can not deny to have granted users' privileges without permission by guaranteeing, with the Remote Attestation Service, integrity of logs and the proof that the application is trusted to properly record these actions. ASSECREQ1 - **Trustworthy Audit** – See AHSECREQ6 - and AHSECREQ7 -.
ACCESS CONTROL AS A SERVICE (ACAAS)	REQUIREMENTS: LREQ3 - **Control of location and responsible provider** – ACaaS associated with each Cloud computing node a set of physical properties. The properties identifies the capabilities of the computing node. One of the capabilities is related to stating the physical location of the node. A user, later on, when requesting the creation of a VM defines any restrictions of hosting location. ACaaS proves the assurance user requirements are met at the computing node level.

PROTOTYPE	Trustworthy OpenStack
REMOTE ATTESTATION SERVICE	REQUIREMENTS: LREQ5 - **Transparency for the customer** – This subsystem guarantees transparency to customers because the Cloud Provider can give the proof of the integrity of the nodes. AHSECREQ6 - **Non repudiation** – This subsystem allows to prove that the application can be trusted to take logs (e.g. through the LogService). If also the integrity of the logs is guaranteed, an attacker can not deny to have performed a specific action. AHSECREQ7 - **Accountability** – This subsystem allows to prove that the application can be trusted to take logs (e.g. through the LogService). If also the integrity of the logs is guaranteed, an attacker can not deny to have performed an action to take user privileges. ASSECREQ1 - **Trustworthy Audit** – See AHSECREQ6 - and AHSECREQ7 -. ASSECREQ2 - **Trustworthy Infrastructure** – This subsystem allows to prove that the infrastructure was trustworthy to perform a specific action (e.g. launch a virtual machine in a node with specified security requirements).

PROTOTYPE	TrustedInfrastructure Cloud
TRUSTED-SERVER	REQUIREMENTS: LREQ1 - **Confidentiality of personal data** – Confidentiality is ensured by the TVD policy. Data is always encrypted according to the TVD. LREQ2 - **Availability and Integrity of personal data** – Integrity of the server platform is ensured by Trusted Computing technologies for secure and integer system boot. AHSECREQ1 - **Confidentiality of stored and transmitted data** – Confidentiality is ensured by the TVD policy. Data is always encrypted according to the TVD. AHSECREQ2 - **Integrity of stored and transmitted data** – Integrity of the server platform is ensured by Trusted Computing technologies for secure and integer system boot. The integrity of the system is a prerequisite for data integrity of the stored and processed data on the system. Additional means for data integrity with TVDs are necessary and are future work.
TRUSTED-OBJECTS MANAGER (TOM)	REQUIREMENTS: LREQ3 - **Control of location and responsible provider** – Information cannot cross the boundaries of the TVD. The TOM is in control of which TrustedServer is allowed to join the TVD. ASSECREQ2 - **Trustworthy Infrastructure** – The TOM is a trustworthy management component for TrustedServers and builds on Trusted Computing technology to ensure by remote attestation that only integer TrustedServer enter the infrastructure. AHSECREQ4 - **Availability of stored and transmitted data** – The TOM is in control of which entities are allowed to join a TVD. If only trusted entities are allowed this minimizes the risk for denial of service attacks. AHSECREQ5 - **Availability of the application** – The TOM is in control of which entities are allowed to join a TVD. If only trusted entities are allowed this minimizes the risk for denial of service attacks.
TRUSTED MANAGEMENT CHANNEL	REQUIREMENTS: ASSECREQ2 - **Trustworthy Infrastructure** – The Trusted Management Channel ensures authenticity, integrity and confidentiality for the communication between the TOM and the TrustedServers.

PROTOTYPE	Cloud-of-Clouds
STATE MACHINE REPLICATION (BFT-SMART)	REQUIREMENTS: LREQ2 - **Availability and Integrity of personal data** – This component ensures this due to the use of Byzantine fault-tolerant replication of a data store implemented using it, i.e., the system remains correct and available even if a fraction of its replicas, say at most f (fault threshold), are compromised. AHSECREQ2 - **Integrity of stored and transmitted data** – By the same reason as in LREQ2 -, if some replica corrupt the data the system will mask and possible detect the fact that some replica is sending wrong results. AHSECREQ3 - **Integrity of the application** – By the same reason as in LREQ2 -, if some replica of a service executes the operation incorrectly (due to a failure or intrusion) the system will mask and possible detect the fact that some replica is faulty. AHSECREQ4 - **Availability of stored and transmitted data** – The fact the middleware provides Byzantine fault tolerance means that it support both data/service corruption and possible replica unavailability periods. AHSECREQ5 - **Availability of the application** – Same as AHSECREQ4 -. AHSECREQ8 - **Data source authentication** – As long as the attacker do not compromise more replicas than the fault threshold tolerated by BFT-SMaRt, it cannot impersonate the service. ASSECREQ3 - **Trustworthy Persistence Engine** – See AHSECREQ2 -, AHSECREQ3 -, AHSECREQ4 - and AHSECREQ5 -. Notice that confidentiality cannot be ensured by this component alone. ASSECREQ4 - **Resilient** – See AHSECREQ2 -, AHSECREQ3 -, AHSECREQ4 - and AHSECREQ5 -. ASSECREQ5 - **Trustworthy communications** – See AHSECREQ8 -.

PROTOTYPE	Cloud-of-Clouds
RESILIENT OBJECT STORAGE	REQUIREMENTS: LREQ1 - **Confidentiality of personal data** – This is ensured by storing only encrypted data on cloud providers. The keys used for encryption are either stored with the client or spread in several providers using secret sharing, ensuring no provider alone has access to the key. LREQ2 - **Availability and Integrity of personal data** – This is ensured by replicating the encrypted stored data in more than one cloud provider and by using novel Byzantine fault-tolerant protocols for reading and writing this data. LREQ4 - **Unlinkability and Intervenability** – See LREQ1 -. AHSECREQ1 - **Confidentiality of stored and transmitted data** – See LREQ1 -. AHSECREQ2 - **Integrity of stored and transmitted data** – See LREQ2 -. AHSECREQ4 - **Availability of stored and transmitted data** – See LREQ2 -. ASSECREQ3 - **Trustworthy Persistence Engine** – See LREQ1 - and LREQ2 -. ASSECREQ4 - **Resilient** – This applies for the persistence level, as described in ASSECREQ3 -.

PROTOTYPE	Other prototypes
SIMPLE KEY/-VALUE STORE (MEMCACHED)	REQUIREMENTS: **AHSECREQ3 - Integrity of the application** – By developing the subsystem in a type-safe language even at the operating system level and minimising the running code base, the entire service becomes very hard to attack. This increases confidence in the integrity of the system. **ASSECREQ6 - High performance & Scalable** – As the the amount of program code is reduced and the operating system becomes part of the application, many time consuming intermediate steps can be skipped. Also taking special considerations for cloud computing environments into account, this should result in better performance than traditional software stacks can achieve. **ASSECREQ2 - Trustworthy Infrastructure** – The improved type-safety and compile-time checks reduce the attack surface of the memcached storage. This prevents intrusions into cloud infrastructures providing a key/value service using our implementation, thus improving the trustworthiness of the infrastructure.
SECURE VM INSTANCES	REQUIREMENTS: Since SBS builds the foundation for Secure VM Instances and already satisfies three requirements, they are identical to the SBS case due the fact that they are provided transitively by SBS.
FAULT-TOLERANT WORKFLOW EXECUTION	REQUIREMENTS: **AHSECREQ3 - Integrity of the application** – Instead of running a given business process only on a single machine, it is executed on multiple machines in parallel. If one of the replicas produces a wrong result it can be detected by comparing the result with the other replicas. While this doesn't prevent that the application can enter an erroneous state in the first place, it is possible to detect and remove the faulty replica in most cases. **AHSECREQ5 - Availability of the application** – The replicated execution of the business process also improves the availability of the service provided by the process. If one of the replicas crashed there are still other working instances available that can be used to continue without any interruptions. **ASSECREQ4 - Resilient** – Parts of the application that are implemented using BPEL could use the replication mechanism to improve the resilience against faults at the infrastructure level; It can tolerate crashes of single replica instances.

PROTOTYPE	Other prototypes
CONFIDENTIALITY PROXY FOR S3	REQUIREMENTS: LREQ1 - **Confidentiality of personal data** – Confidentiality is ensured by the TVD policy. Data is always encrypted according to the TVD before it is stored on public S3 storage. AHSECREQ1 - **Confidentiality of stored and transmitted data** – Confidentiality is ensured by the TVD policy. Data is always encrypted according to the TVD. ASSECREQ2 - **Trustworthy Infrastructure** – The TrustedServer is rigorously built on top of Trusted Computing technology, ensuring integrity of the system and supporting remote attestation.
ONTOLOGY-BASED REASONER	REQUIREMENTS: LREQ2 - **Availability and Integrity of personal data** – Through the Trusted Virtual Domain concept, this subsystem helps satisfy these requirements by isolating the network connecting the virtual machines that run the user application. This avoids that an attacker intercepts the communication channel among TVD members or mounts a denial-of-service attack. AHSECREQ1 - **Confidentiality of stored and transmitted data** – See LREQ2 -. AHSECREQ5 - **Availability of the application** – See LREQ2 -. AHSECREQ8 - **Data source authentication** – This requirement is satisfied because a virtual machine can communicate only with members of the same TVD. ASSECREQ2 - **Trustworthy Infrastructure** – This subsystem helps prevent attacks from compromised hosts in the infrastructure. ASSECREQ5 - **Trustworthy communications** – Data exchanged among virtual machines members of the same TVD can not be tampered with by an attacker.

PROTOTYPE	Other prototypes
SECURITY ASSURANCE OF VIRTUALIZED ENVIRON-MENTS (SAVE)	REQUIREMENTS: LREQ1 - **Confidentiality of personal data** – Confidentiality is achieved by verifying that the isolation of certain security zones exists in the current configuration and topology of the cloud infrastructure. A security zone may be based on tenants, in order to ensure that tenants in a multi-tenant cloud infrastructure are isolated, or it may be based on a tenant's definition of zones (e.g. to isolate testing from production systems). LREQ5 - **Transparency for the customer** – The cloud provider can show the verification results of the isolation of the security zones to the customers. However, it does not prevent against a malicious cloud provider that manipulates the verification results, but it protects against misconfigurations of the provider. AHSECREQ1 - **Confidentiality of stored and transmitted data** – Cf. LREQ1 - with a security zone for the patient data storage and processing.

Chapter 4

Tests Plan and Results Report

4.1 Introduction

Testing is one of the main activities that is performed during the software development. It allows to detect faults that may occur from the definition of its specifications to the implementation phase. Each step of the development process must be carefully checked because, for example, a misunderstanding of the software specifications may lead to implement features that are not intended in the original design.

It is also important to perform the testing as early as possible because, first, detecting errors on the whole software is more harder than checking the code just after it has been written and, secondly, the costs for fixing that errors are greater when the product has already been implemented.

In the TClouds project performing testing becomes very critical because the product it aims to deliver will be composed by a collection of software subsystems developed by different partners, that must work together to provide the intended security services. Further, since this project is focused on releasing a prototype to demonstrate the feasibility of the solutions proposed, testing allows to monitor progresses in the development process and to ensure that the prototype will be delivered in time.

In this document, we define a master test plan for the TClouds project by indicating the activities that each partner should perform and by providing details about how tests will be performed and how the documentation with test results should be written. Finally, we provide specific test plans from partners involved in the development of subsystems in Activity 2.

4.2 A model for testing

Since the main TClouds project objective is to release a concrete prototype integrated within open source framework for Cloud Computing, it is necessary to adopt a model for doing the tests that covers all aspects of the development process, from the definition of the requirements to the software implementation. A model that can be used for this purpose is the V-model.

This model is particularly suitable for our purposes as it accommodates the structure of TClouds that is split in: Activity 1 is responsible for the definition of legal and business requirements; Activity 2 will build the TClouds Platform with the security functionality required to satisfy these requirements; Activity 3 will develop two applications, one for a medical use case and another for the energy management on a public infrastructure, that will run on top of the TClouds Platform.

In the V-model, depicted in Figure 4.1, a specific level of testing is planned for each phase of the development. In particular, the requirements definition is tested through the *acceptance testing* performed by customers, the functional system design is tested using *system testing*, the

technical system design is verified through *integration testing*. Finally, the component specification phase is tested through *unit testing*.

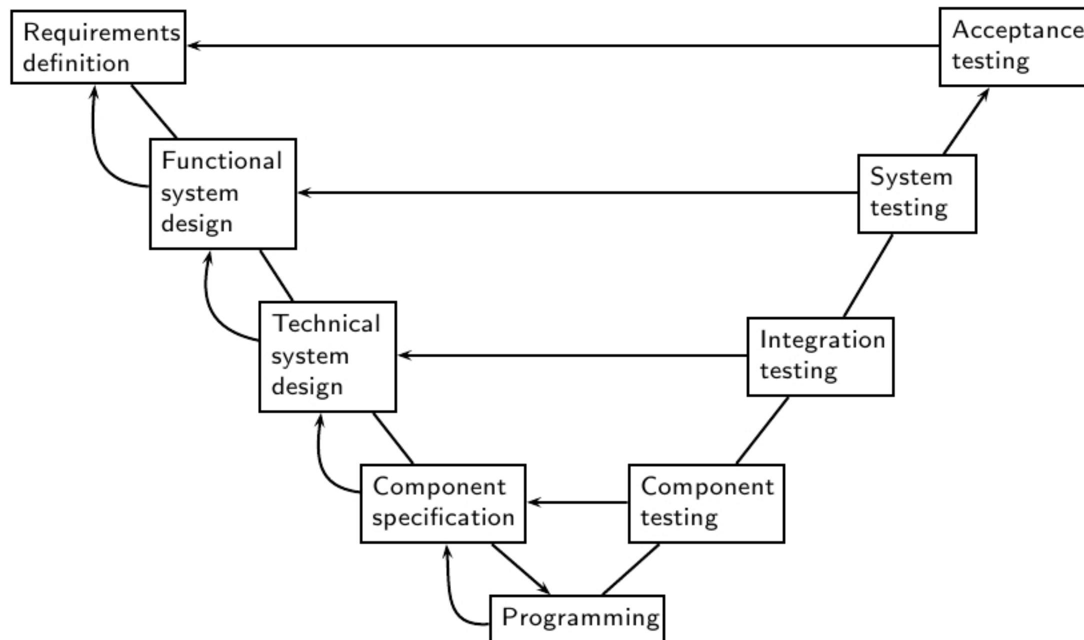

Figure 4.1: The V-model

Acceptance testing would allow a customer to verify whether legal and business requirements defined in Activity 1 are met either by the TClouds Platform from Activity 2 and the applications developed by Activity 3.

System testing can be used to check whether the functions exposed through the *TClouds API* behave as defined in the functional design. These functions can be tested using the applications from Activity 3.

Integration testing should be performed on the subsystems developed in Activity 2 and, in particular, can be used to test the communication between them through the *Internal API*. This type of testing would allow to verify whether a subsystem collaborates with other subsystems in the expected way in order to perform complex functions.

Finally, *unit testing* can be used by each partner to test his subsystems and to verify whether the implementation has been done according to the specifications. We refer the reader to Section 4.3.2 for details about the testing levels that are covered in the TClouds master test plan.

4.3 Master test plan

In this section, we define how tests should be performed, given the fact that the above Activities are performed in parallel and that the development process has already been scheduled in the Description of Work (DoW).

While in the Year 1 partners have been involved in the definition of their subsystems, the overall architecture of the TClouds Platform and the specifications of the applications that will run on top of it, in the Year 2 and 3 they will concentrate on the development of code that will be tested according to levels introduced in the Section 4.2.

Since the development tasks on the TClouds Platform and the applications are overlapped, Activities 2 and 3 need to perform tests separately until the end of the Year 2 when the first mockup integration (TClouds Platform v1) will be available. Meanwhile, partners can perform unit and integration testing on their software and, then, they can perform system testing together by executing the applications test cases on top of the TClouds Platform.

However, we expect some test cases to fail because only a subset of the security functionality will be available in the first version of the TClouds Platform. The complete set will be available at the end of the Year 3, when all subsystems introduced in the Year 1 deliverables will be integrated in the final platform (TClouds Platform v2).

During the Year 3 another run of unit and integration tests will be executed for the new software to ensure that it works together with the existing subsystems of the TClouds Platform and, then, the system testing will be executed on the new version of the platform. Finally, the acceptance testing can be performed by customers of the whole TClouds product in order to verify that the latter satisfies the legal and business requirements from Activity 1.

4.3.1 Testing environment

Partners will start the testing activities during the Year 2, when they begin to implement the software as defined in the deliverables. Before related tasks will take place it is necessary to define the environment in which tests will be executed.

Since the software will be under heavy development especially in the first months, it is necessary to setup two separate environments: one for running tests on software that is being developed, so that it would be possible to detect and fix specific errors; another for testing the interactions between subsystems that have been tested for a while, in order to find whether they communicate in the correct way.

The first environment will be created on a dedicated machine that periodically builds the code of the configured subsystems by fetching it from the TClouds code repository, launches the tests defined and finally collects and stores the results obtained in a report. Since these tests allow to evaluate the stability of the software, the environment must ensure that a subsystem is isolated from each other during the test execution.

Once a subsystem becomes sufficiently stable, for instance the percentage of passed tests is higher than the defined threshold, it could be integrated in the second environment which consists of a Cloud built only for testing purposes. This environment will be used to test interactions between subsystems developed from Activity 2 or between these subsystems and the applications from Activity 3. For those subsystems that need to be deployed in a Cloud of Clouds, the testing environment will be extended using other Cloud solutions, like Amazon AWS or Rackspace.

4.3.2 Testing levels

During the project TClouds partners involved in the development of software will perform testing activities at the following levels:

- **Unit testing:** each partner is responsible to perform unit tests on his subsystems. These can be either black-box testing and white-box testing and will allow to determine whether the software can be integrated in the Cloud environment. When a defect is encountered on a subsystem the unit tests must be run after the software has been fixed and must show

that the issue has been resolved. Only after the tests are passed, the updated version of the subsystem can be integrated in the Cloud environment.

- **Integration testing:** the integration tests allow to verify the interactions between subsystems that collaborate together to perform complex functions. The partners that own these subsystems are responsible to develop the test cases that will run on the Cloud environment. If a subsystem exposes a function through the TClouds API, the partner can test this type of interaction using a *test driver*, a simple program that contains a call to that function. Instead, applications developers can perform integration tests by developing a *stub*, a software that exposes the same function of the TClouds API, but without its implementation. These additional pieces of software are needed because the applications and the TClouds Platform will be available only at the end of the Year 2.

- **System testing:** systems tests will be defined by the applications developers and will consist in performing operations that require the interaction with the TClouds Platform through the TClouds API. At the end of the Year 2 the test drivers and the stubs will be replaced with the software developed. However all application test cases are expected to pass only at the end of Year 3 when all the security functionality of the TClouds Platform are implemented.

4.3.3 Testing activities workflow

A detailed workflow of the testing activities that should be performed is depicted in the Figure 4.2. The testing activities will be performed by four actors: the *Partners*, the *Jenkins server*, the *Cloud Administrator* and the *Tester*.

1 **Upload the code:** the Partners are responsible to upload the code of their subsystems to the TClouds code repository. A version control mechanism (e.g. Subversion) should be used to keep track of the modifications occurred on the code over the time.

2 **Configure Jenkins:** the Partners can configure the Jenkins server running on the dedicated machine (introduced in the Section 4.3.1) in order to execute automatic unit tests. This machine will have installed all required software dependencies required to build the code and to launch the tests. The Partners will find the instructions to setup Jenkins in the Section 4.5.

3 **Fetch latest code:** the Jenkins server will fetch the latest code from a repository owned by the Partners or from the TClouds code repository configured by TEC.

4 **Build the code:** the Jenkins server will build the fetched code and stores the logs in its internal database. The instructions for the building phase must be specified by the Partners during the setup of Jenkins.

5 **Execute unit tests:** the Jenkins server will launch the unit tests defined by the Partners. These tests should produce an output compatible with the JUnit XML format (`http://www.junit.org`), so that Jenkins can parse the results obtained and can display them in a Web page.

6 **Store unit tests results:** the Jenkins server stores the results of the executed unit tests in the TClouds reports repository.

7 **Download and install stable code:** the Cloud Administrator is responsible to build the infrastructure to be used for tests in the Cloud environment. This infrastructure will be built using the OpenStack framework (`www.openstack.org`) as it is the platform chosen by the TClouds project. The Cloud Administrator will integrate subsystems developed by the Partners by downloading the stable version from the TClouds code repository and installing them in the Cloud nodes. When a faulty software is updated, he is responsible for replacing the old version with the new one.

8 **Launch integration/system tests**

a) **Automatic:** the Jenkins server will coordinate the execution of automatic integration and system tests. It can execute the commands directly on the Cloud environment or can launch the tests locally on the dedicated machine if the subsystem is reachable over the network.

b) **Manual:** the Tester will execute manual integration and system tests on the Cloud environment through the Management interface. He will perform the steps described in the test cases defined by the Partners.

9 **Collect integration/system tests results**

a) **Automatic:** the Jenkins server collects the outputs generated during the execution of the tests. These outputs must be compatible with the JUnit XML format, otherwise ad hoc parsers must be developed to understand the custom format.

b) **Manual:** the Tester must observe the behavior of the software being tested during its execution and must fill the report based on a template, reporting the results obtained.

10 **Store integration/system tests results**

a) **Automatic:** the Jenkins server stores the reports obtained from the parsed tests output in the TClouds reports repository.

b) **Manual:** the Tester stores the compiled reports in the TClouds reports repository through the Management interface of the Cloud environment.

4.3.4 Test results evaluation and exit criteria

The testing activities will allow partners to improve the development process and to detect in advance defects that may cause the whole system not to work as expected. During the tests definition, it is possible to achieve different goals depending on the role a subsystem plays in the Cloud environment. For example, a subsystem could be tested for its performances because other subsystems may require that data is delivered by the former in a certain time frame. Depending on the goal, a test may return completely different results that must be interpreted by the test writer.

However, there are common goals that should be achieved when defining test cases. In particular, for the TClouds project, test cases are needed to evaluate:

- **Stability:** a significant part of test cases should be defined to verify that the software can handle different combinations of inputs without crashing. The number of tests passed and their coverage will allow to determine when a software is ready for integration in the Cloud environment.

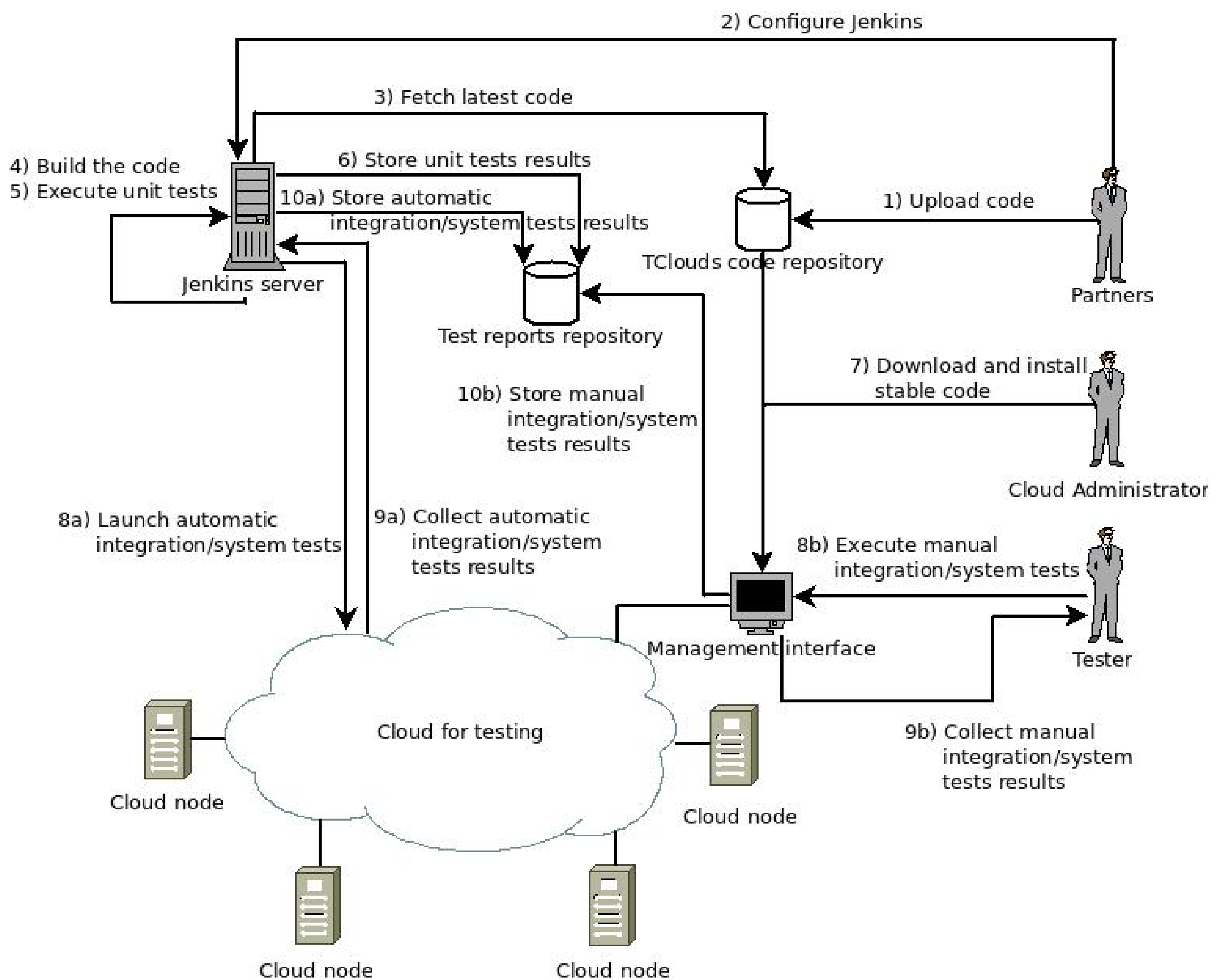

Figure 4.2: Testing activities workflow

- **Functionality:** another set of test cases should be defined to verify whether the features described in the design document are currently implemented or not. These tests are useful to keep track of the software development and to determine whether it can be delivered in time.

Once the tests are executed and their results are collected, it is needed to define criteria to determine when a software can be considered of sufficient quality for release as part of the final TClouds prototype. Since it is not aim of the TClouds project to develop a commercial product that can be used in production environments but, instead, to deliver a prototype to demonstrate research results from partners in the area of Cloud Computing, the tests can be considered passed when the whole system works in the correct way under normal conditions. For example, one aspect that will not be considered in this document is the misbehavior of the underlying hardware or software that could lead to unexpected results.

4.4 Test plans for subsystems/prototypes

4.4.1 TrustedInfrastructure Cloud

4.4.1.1 Test methodology/strategy

The TrustedObjectsManager component will be tested in a manual way. The prerequisites for this testing are a readily setup TrustedObjectsManager, 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 4.4.1.2. This test cannot be automated since user interaction with different physical machines is required.

4.4.1.2 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 and can be used on TrustedDesktop

TEST CASE ID	**/TC 4.4.1.2-1/** 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	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”
RESULT	Passed ☐ Failed ☐

TEST CASE ID	**/TC 4.4.1.2-2/** 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	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
REMARKS	
RESULT	Passed ☐ Failed ☐

TEST CASE ID	**/TC 4.4.1.2-3/** 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	1 User uses the service from within the compartment 2 The service answers as expected
REMARKS	
RESULT	Passed ☐ Failed ☐

TEST CASE ID	**/TC 4.4.1.2-4/** 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	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
REMARKS	
RESULT	Passed ☐ Failed ☐

4.4.2 Security Assurance of Virtualized Environments *(SAVE)*

4.4.2.1 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.

4.4.2.2 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

4.4.3 Resource-efficient BFT (CheapBFT)

4.4.3.1 Test methodology/strategy

CheapBFT is a system with complex setting and configuration whose main purpose is to mask occurring errors. That makes it relatively difficult to test the system in an automated manner. Therefore, CheapBFT is tested manually. However, scripts and tools are provided to enable an easy set-up, conducting, and analyzing of test runs. Since CheapBFT is not a Byzantine fault tolerant service itself but a basis for the implementation of such services, test applications are required. Here, we employ an artificial benchmark and, as a more realistic application, the secure log service whose test plan is describe in Section 4.4.8.

Before the tests can be carried out, three machines have to be equipped with FPGA cards which in turn have to be initialized with the firmware of the trusted hardware module of Cheap-BFT, named Cash. Further, a Linux system and all CheapBFT software components must have been installed on the provided machines. The machines must be connected over an TCP/IP network. Moreover, one additional machine that hosts the clients for the benchmark and the log service is required. (A more detailed description of the set-up and its installation can be found at Section **??**.)

The system (comprising CheapBFT and the log service hosted on it) and all tests can be started by means Bash of scripts. The outcome of test runs are basically log messages and other feedback provided to a tester in form of files and as console or graphical output. The given feedback enables a tester to assess the proper functioning of the system and to evaluate the performance in comparison with the resource usage.

4.4.3.2 Test cases

TEST CASE ID	**/TC 4.4.3.2-1/** Fault-free operation
DESCRIPTION	As a first simple function test, an artificial benchmark application is executed on top of CheapBFT. A client sends requests of configurable size to the system. All requests are handled through the active replicas by generating replies and updates for the passive replicas (also with configurable size). This is done without any additional (simulated) computation.
TYPE	Functional test
PRECONDITIONS	The test environment, which comprises four machines (three servers and one client) that are provided with all required hardware and software modules, is up and running. CheapBFT is properly configured, especially the addresses of the machines and the location of the program files are set.
STEPS	1 Select CheapTiny as consensus protocol: ./run_micro.bash setup_prot cheap 2 Start the test: ./run_micro.bash start [--warmup=<warm-up time in sec>] [--run=<test run time in sec>] [--micro.reqsize=<request size>] [--micro.replysize=<reply size>] [--micro.updsize=<update size>] 3 The client should continuously issue requests and the replicas should respond them. 4 Shut down and clean up: ./run_micro.bash cleanup -t
NOTES	The script run_micro.bash is located at the directory cheap/bin. Please note, that the syntax of the commands might change in future versions.
REMARKS	
RESULT	Passed ☐ Failed ☐

<table>
<tr><td>TEST CASE ID</td><td>/TC 4.4.3.2-2/ Comparison of different consensus protocols</td></tr>
<tr><td>DESCRIPTION</td><td>In this test, the behavior and resource usage of CheapTiny is compared to MinBFT. The basis for the comparison is the same benchmark application as used in text case /TC 4.4.3.2-1/ .</td></tr>
<tr><td>TYPE</td><td>Benchmark test</td></tr>
<tr><td>PRECONDITIONS</td><td>Same as for test case /TC 4.4.3.2-1/ .</td></tr>
<tr><td>STEPS</td><td>1 Select CheapTiny or MinBFT as consensus protocol:

<code>./run_micro.bash setup_prot cheap</code>
<code>./run_micro.bash setup_prot min</code>

2 Start the benchmark (see test case /TC 4.4.3.2-1/) for each protocol with different arguments for the request, reply, and update size.
3 Shut down and clean up the system after each test run.
4 Depending on the settings for sizes of requests, replies, and updates, the resource monitors should show different resource usage between the the leader of the replica group, the second active replica and the passive one in the case of CheapTiny. In the case of MinBFT, there should be only a noticeable difference between the leader and the other two active replicas.</td></tr>
<tr><td>NOTES</td><td>See test case /TC 4.4.3.2-1/ .

Contrary to the simple benchmark employed here, a real application would carry out some kind of computation or other activities to fulfill the requests, which would be more beneficially for CheapBFT regarding the resource usage because of the lower count of replicas actively executing requests. However, this test is mainly meant to illustrate the different characteristics in resource usage. Alternatively, the Log Service application from test case /TC 4.4.3.2-3/ can be employed.</td></tr>
<tr><td>REMARKS</td><td></td></tr>
<tr><td>RESULT</td><td>Passed ☐
Failed ☐</td></tr>
</table>

<table>
<tr><td>TEST CASE ID</td><td>/TC 4.4.3.2-3/ Operation in the presence of errors</td></tr>
<tr><td>DESCRIPTION</td><td>CheapBFT is meant for the provisioning of fault-tolerant services. This test is to examine the proper functioning of the system in the presence of errors. Here, the more realistic demo application, the secure log service, is used.</td></tr>
<tr><td>TYPE</td><td>Functional test</td></tr>
<tr><td>PRECONDITIONS</td><td>Same as for test case /TC 4.4.3.2-1/ .</td></tr>
<tr><td>STEPS</td><td>1 Select CheapTiny as consensus protocol:
<code>./run_logsrv.bash setup_prot cheap</code>
2 Start the test:
<code>./run_logsrv.bash start</code>
3 The demo client should continuously invoke store and retrieve operations of the log service.
4 Induce an error into the log file of one replica:
<code>./run_logsrv.bash inderr 1</code>
5 CheapBFT should eventually detect the error and should switch from CheapTiny to MinBFT. This can be verified by the tester through corresponding log messages and status information.
6 The client should not be affected (except of changed timings) by the induced errors. It should still receive correct answers from the the log service.</td></tr>
<tr><td>NOTES</td><td>See test case /TC 4.4.3.2-1/ .
The script <code>run_logsrv.bash</code> is located at the directory <code>logservice/bin</code>.</td></tr>
<tr><td>REMARKS</td><td></td></tr>
<tr><td>RESULT</td><td>Passed ☐
Failed ☐</td></tr>
</table>

4.4.4 Secure Block Storage

4.4.4.1 Test methodology/strategy

The Secure Block Storage will be tested manually, since there are many diverse components involved which do net 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 Secure Block Storage, 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, this is *confidentiality* and *integrity* in order to protect the assests 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.

4.4.4.2 Test cases

TEST CASE ID	**/TC 4.4.4.2-1/** 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	1 Machine is powered on. 2 GRUB boots Tboot with Xen, Dom0 and DomT as multiboot modules. 3 In Dom0 the admin uses the standard Xen tool which has been patched: `xl domt --out-file=~/wrapkey.enc getkey.`
RESULT	Passed ☐ Failed ☐

TEST CASE ID	**/TC 4.4.4.2-2/** Customer makes encrypted VM available
DESCRIPTION	Customer encrypts her VM with cloud key
TYPE	Functional test
PRECONDITIONS	Domain builder initialized (test **4.4.4.4.4.2-1**)
STEPS	1 The customer takes a working VM image (disk image). 2 The image is encrypted using a symmetric key k. 3 the key symmetric key k is encrypted with the asymmetric key from test 4.4.4.4.4.2-1.
RESULT	Passed ☐ Failed ☐

TEST CASE ID	**/TC 4.4.4.2-3/** Deploy and run secure image
DESCRIPTION	Deploy the customer's encrypted image to the cloud
TYPE	Functional test
PRECONDITIONS	Customer VM deployed (test **4.4.4.4.4.2-2**)
STEPS	1 Using `scp` the customer's image is copied to the Dom0 domain. 2 In Dom0 the cloud admin configures the domain config in `/etc/xen/domain_X.cfg` for this domain to have the `domc = 1` flag. 3 The cloud admin starts the domain using `xl create /etc/xen/domain_X.cfg`.
RESULT	Passed ☐ Failed ☐

TEST CASE ID	**/TC 4.4.4.2-4/** Test Security of SBS
DESCRIPTION	Test the avenues via which the cloud administrator can attack
TYPE	Functional test
PRECONDITIONS	Customer VM running (test **4.4.4.4.4.2-3**)
STEPS	1 The cloud admin tries in vain a hexdump on the customers encrypted VM. 2 The cloud admin tries in vain to use the `xc_map_foreign_range()` hypercall to do introspection (for instance, with the help of the LibVMI[1] library which abstracts from the bare hypercall).
RESULT	Passed ☐ Failed ☐

[1] http://code.google.com/p/vmitools/.

4.4.5 Access Control as a Service (ACaaS)

4.4.5.1 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.

4.4.5.2 Test cases

TEST CASE ID	**/TC 4.4.5.2-1/** User requirement management
DESCRIPTION	Test the user requirment management module. Ensure user requirement catalogues can be added, removed and queried by administrators correctly.
TYPE	Functional test
STEPS	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.
RESULT	Passed ☐ Failed ☐

TEST CASE ID	**/TC 4.4.5.2-2/** Infrastructure property management
DESCRIPTION	Ensure infrastructure properties can be specified, removed and queried correctly
TYPE	Functional test
STEPS	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.
RESULT	Passed ☐ Failed ☐

TEST CASE ID	**/TC 4.4.5.2-3/** 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 **4.5.4.4.5.2-1, 4.5.4.4.5.2-2**)
STEPS	1 Run a VM instance with at least one host satisfying its requirements, expecting the VM scheduled to the host with satisfying properties 2 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.
RESULT	Passed ☐ Failed ☐

4.4.6 BFT-SMaRt

4.4.6.1 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 verion 1.5 or later installed. To run the test cases using the Ant script provided with the source code it is necessary to have Apach 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 demo is 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.

4.4.6.2 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 4.4.6.2-1/** 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 BFTMapClientTest.testRegularCase().
RESULT	Passed ☐ Failed ☐

TEST CASE ID	**/TC 4.4.6.2-2/** 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 BFTMapClientTest.testStopNonLeader().
RESULT	Passed ☐ Failed ☐

Test Case ID	**/TC 4.4.6.2-3/** 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 BFTMapClientTest.testStopAndStartNonLeader().
Result	Passed ☐ Failed ☐

Test Case ID	**/TC 4.4.6.2-4/** 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 BFTMapClientTest.testStopLeader().
Result	Passed ☐ Failed ☐

Test Case ID	**/TC 4.4.6.2-5/** 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 BFTMapClientTest.testStopLeaders().
Result	Passed ☐ Failed ☐

4.4.6.3 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.

4.4.7 Resilient Object Storage (DepSky)

4.4.7.1 Test methodology/strategy

DepSky works with different clouds 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 have the accounts created it is necessary to have the user and private keys to be used to manipulate data. DepSky have drivers written for different providers as Amazon, Nirvanix.

4.4.7.2 Test cases

In the tests described here, DepSky stored data units in four Amazon S3 servers distributed in different locations. To define the locations there is the file configClouds under the config directory. All tests described here contains a startup procedure described here: 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 was written to the cloud, it is necessary to open the cloud provider console. In the tests written we used Amazon S3, so, to verify the data we opened the Amazon S3 console in console.aws.amazon/s3.

TEST CASE ID	**/TC 4.4.7.2-1/** 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 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 file system.
STEPS	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 Query the data is retrieved with the command read.
RESULT	Passed ☐ Failed ☐

TEST CASE ID	**/TC 4.4.7.2-2/** 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	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.
RESULT	Passed ☐ Failed ☐

TEST CASE ID	**/TC 4.4.7.2-3/** 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	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.
RESULT	Passed ☐ Failed ☐

4.4.8 LogService

4.4.8.1 Test methodology/strategy

Log Service is the component that provides secure logging capabilities in the TClouds architecture. The core element of the Log Service is the `libsklog` library. Such a library is mainly written in C, hence is possible to implement unit tests using frameworks like CUnit [KS12] or Cmockery [Mil12]. We select the framework CUnit because it is capable to export test results in a format which is readable and importable by Jenkins. Despite `libsklog` is designed to operate on a distributed environment is possible to install it on a single node in order to execute the tests without losing test case. We base the development of the `libsklog` library on Debian *Wheezy*. To execute the tests the following packages need to be installed:

```
# available as Debian packages
libtool
autoconf
build-essentials
OpenSSL >= 1.0.0
SQLite3
libuuid
libconfig >= 1.4.8
libjansson >= 2.3.1
libreadline
python-sphinx
python-bottle
python-dev
libmysqlclient-dev

# available from sources
libumberlog (https://github.com/algernon/libumberlog)
```

4.4.8.2 Test cases

Tests cover the main functionality of Log Service. In detail, we test the initialisation of a new logging session, the logging of a set of dummy events, the retrieval of already opened logging session and finally, the verification of a logging session. The tests are provided within the `libsklog` source code. In order to run them, it is necessary to build the library with the option `--enable-tests` as shown in the Listing 4.1:

```
cd /temp
tar zxvf libsklog-<version>.tar.gz
cd libsklog-<version>
./autogen.sh
./configure --enable-tests --with-storage=rest
make
cd test
```

Listing 4.1: Commands to build the `libsklog` library in testing mode

TEST CASE ID	**/TC 4.4.8.2-1/** Checking for malformed input
DESCRIPTION	Tests the behaviour of the main functions in case of malformed input. In detail, is checked the behaviour of the function when a NULL pointer is passed as argument. Moreover, is checked how the functions reacts when a data that non respect the specifications is passed as input (for instance when the accepted values are α, β but γ is supplied).
TYPE	Functional test
PRECONDITIONS	`libsklog` built with `--enable-test` option (Listing 4.1).
STEPS	1 Exec the command: `./run_tests --malformed-input`

TEST CASE ID	**/TC 4.4.8.2-2/** Logging session initialisation
DESCRIPTION	Tests the initialisation of a new logging session. The test simulates the initialisation message exchanging among the *Cloud Component* and the *Log Core*. The test is considered passed if a new log file is created and if such a file contains the initialisation log entries.
TYPE	Functional test
PRECONDITIONS	`libsklog` built with `--enable-test` option (Listing 4.1).
STEPS	1 Exec the command: `./run_tests --init-session`

TEST CASE ID	**/TC 4.4.8.2-3/** Log dummy events
DESCRIPTION	Tests the execution of the logging operation. The test logs a set of 1000 dummy events that are bound to the logging session initialised in the testcase **/TC 4.4.8.2-2/** and checks if the generated log file contains 1000 log entries plus 3 control entries (2 initialisation entries + 1 closure entry).
TYPE	Functional test
PRECONDITIONS	`libsklog` built with `--enable-test` option (Listing 4.1). Execution of testcase **/TC 4.4.8.2-2/** .
STEPS	1 Exec the command: `./run_tests --init-session --log-events`

TEST CASE ID	**/TC 4.4.8.2-4/** Logging sessions retrieval
DESCRIPTION	Checks the retrieval of already initialised logging sessions. The test checks if the list size is ≥ 1.
TYPE	Functional test
PRECONDITIONS	`libsklog` built with `--enable-test` option (Listing 4.1). Execution of testcase **/TC 4.4.8.2-2/** .
STEPS	1 Exec the command: `./run_tests --retrieve-sessions`

TEST CASE ID	**/TC 4.4.8.2-5/** Logging session verification
DESCRIPTION	Tests the verification process in case of undamaged log and in case of damaged log. In the first case the test pass if the verification return a positive result, in the second case, the test pass if the verification result is negative.
TYPE	Functional test
PRECONDITIONS	`libsklog` built with `--enable-test` option (Listing 4.1). Execution of testcase **/TC 4.4.8.2-3/** .
STEPS	1 Exec the command: `./run_tests --init-session --log-events \` `--verify-session`

4.4.9 Remote Attestation Service

4.4.9.1 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 IMA[1] measurements files with the expected ones.

Tests can be executed on a single machine and require the installation of the software specified in Section 5.2 for both nodes and the following Fedora packages: `python-unittest2` and `rpmdevtools`.

4.4.9.2 Test cases

[1]Integrity Measurement Architecture: see `http://linux-ima.sourceforge.net` for details

TEST CASE ID	**/TC 4.4.9.2-1/** 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	1 Download the following Fedora 16 packages in a temporary directory `f16-pkgs`: `Pkg 1`: http://kojipkgs.fedoraproject.org//packages/curl/7.21.7/7.fc16/x86_64/curl-7.21.7-7.fc16.x86_64.rpm `Pkg 2`: http://kojipkgs.fedoraproject.org//packages/curl/7.21.7/7.fc16/x86_64/libcurl-7.21.7-7.fc16.x86_64.rpm `Pkg 3`: http://kojipkgs.fedoraproject.org//packages/curl/7.21.7/7.fc16/x86_64/libcurl-devel-7.21.7-7.fc16.x86_64.rpm 2 Execute this command to insert data into the database: `$ update_pkgs.sh -d f16-pkgs -n F16 -c x86_64 -t testing` 3 For each digest, retrieve the record stored in the database and check the following statements: • `976a6505edeae28ccb63b491b91bce6113e87779` is the digest of the file *usr/bin/curl* which belongs to `Pkg 1` • `8c9f2d95d80d24332139bb33da33a1340b35e1d6` is the digest of the file *usr/lib64/libcurl.so.4.2.0* which belongs to `Pkg 2` • `f9ca79dbbab0d2d2e190904b9fe0e451a7ce901e` is the digest of the file *usr/include/curl/curl.h* which belongs to `Pkg 3` 4 The file `usr/bin/curl` is of type `executable` and depends on the following shared libraries: `libcurl.so.4, librt.so.1, libz.so.1, libpthread.so.0, libc.so.6, libidn.so.11, liblber-2.4.so.2, libldap-2.4.so.2, libgssapi_krb5.so.2, libkrb5.so.3, libk5crypto.so.3, libcom_err.so.2, libssl3.so, libsmime3.so, libnss3.so, libnssutil3.so, libplds4.so, libplc4.so, libnspr4.so, libdl.so.2, libssh2.so.1, ld-linux-x86-64.so.2, libresolv.so.2, libsasl2.so.2, libkrb5support.so.0, libkeyutils.so.1, libssl.so.10, libcrypto.so.10, libcrypt.so.1, libselinux.so.1, libfreebl3.so` 5 The file `usr/lib64/libcurl.so.4.2.0` is of type `library` and has the following aliases: `libcurl.so, libcurl.so.4`

TEST CASE ID	**/TC 4.4.9.2-2/** Test verification of sample IMA measurements files
DESCRIPTION	This test case verifies a set of sample IMA measurements files using the *RA Verifier* component and compares results obtained with those expected.
TYPE	Functional test
PRECONDITIONS	The Apache Cassandra database is up and running

STEPS	1 Download the following Fedora 16 packages in a temporary directory `f16-pkgs`:
	`Pkg 1:` http://kojipkgs.fedoraproject.org//packages/coreutils/8.12/7.fc16/x86_64/coreutils-8.12-7.fc16.x86_64.rpm
	`Pkg 2:` http://kojipkgs.fedoraproject.org//packages/coreutils/8.12/6.fc16/x86_64/coreutils-8.12-6.fc16.x86_64.rpm
	`Pkg 3:` http://kojipkgs.fedoraproject.org//packages/glibc/2.14.90/24.fc16.9/x86_64/glibc-2.14.90-24.fc16.9.x86_64.rpm
	`Pkg 4:` http://kojipkgs.fedoraproject.org//packages/glibc/2.14.90/24.fc16.7/x86_64/glibc-2.14.90-24.fc16.7.x86_64.rpm
	2 Execute this command to insert data into the database:
	`$ update_pkgs.sh -d f16-pkgs -n F16 -c x86_64 -t updates`
	3 Generate the sample IMA measurements files as follows:
	`Sample A:` all digests of files from `Pkgs 1,3`
	`Sample B:` same as above + an unknown digest
	`Sample C:` same as above + a digest of file from `Pkg 2`
	`Sample D:` same as above + a digest of file from `Pkg 4`
	4 Run the verification of sample IMA measurements files by prompting the command:
	$ ra_verifier.py -i <sample_ima_measurements_file>
	The script should return the following output:
	`Sample A:` 613 ok, 0 unknown, 0 pkg-security, 0 pkg-not-security
	`Sample B:` 613 ok, 1 unknown, 0 pkg-security, 0 pkg-not-security
	`Sample C:` 613 ok, 1 unknown, 0 pkg-security, 1 pkg-not-security
	`Sample D:` 506 ok, 1 unknown, 109 pkg-security, 0 pkg-not-security
	where each field indicates the number of measurements of files that:
	• *ok*: have a known digest and belong to the most recent package
	• *unknown*: have an unknown digest
	• *pkg-security*: belong to a package with security updates
	• *pkg-not-security*: belong to a package with other updates

4.5 Jenkins server

According to its Web site (http://jenkins-ci.org) *"Jenkins is an award-winning application that monitors executions of repeated jobs, such as building a software project or jobs run by cron. Among those things, current Jenkins focuses on the following two jobs: [...] building/testing software projects continuously and [...] monitoring executions of externally-run jobs"*.

We chose this software because it will simplify for partners the process of building the code and launching automatic tests that have already been defined in the Section 4.4. Among others, it is currently used by Apache (https://builds.apache.org) and OpenStack (http://jenkins.openstack.org - requires registration).

Our Jenkins installation is running on a dedicated machine hosted by TEC and can be contacted at the URL: http://jenkins.tclouds-project.eu. The main Web page is shown in the Figure 4.3 where actually only the POL-LogService subsystem has been configured. We expect that other partners will progressively register their subsystems during the Year 2.

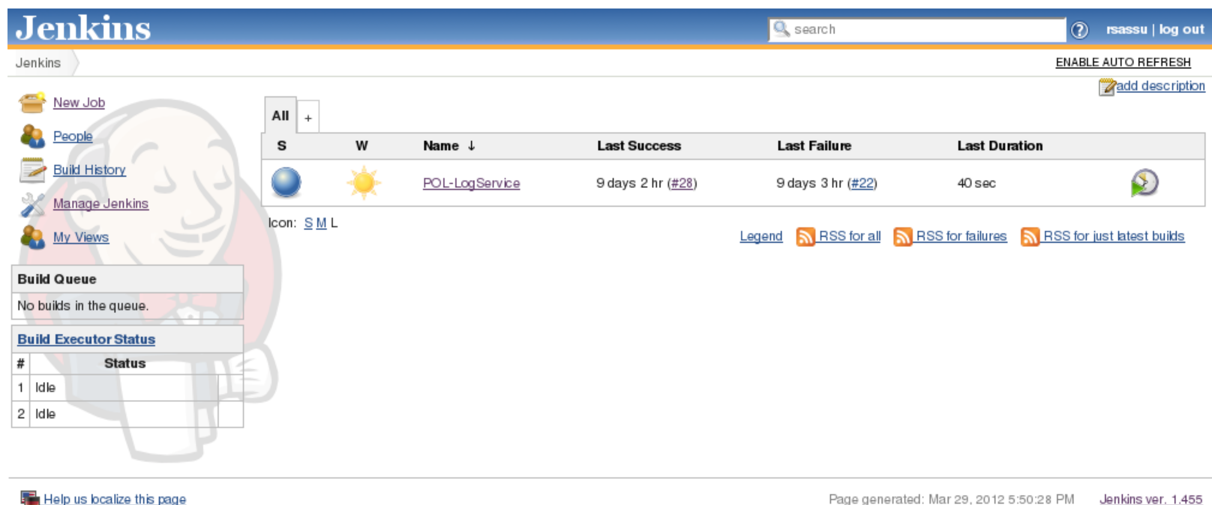

Figure 4.3: TClouds Jenkins Web page

4.5.1 Subsystem setup

The procedure required for configuring Jenkins to build and test a subsystem is very simple and consists of a few steps. The complete documentation that describes all configuration parameters can be found on the Jenkins Web site. In this section we provide a brief tutorial to build a subsystem written in C language using Jenkins.

1 In the main TClouds Jenkins Web page click the mouse over the icon named "New Job" on the top right corner.

2 Prompt the "Job name" in the text field, select the checkbox "Build a free-style software project" and, then, click the OK button.

3 In the loaded Web page there are a lot of configuration parameters used by Jenkins to determine the operations that will be performed during the building and testing phases.

In order to configure the former phase, it is needed to select the correct version control system used to manage the code (CVS, git and Subversion are currently supported) and to provide a valid URL of the repository.

4 In the section "Build" of the same Web page, it is necessary to provide the instructions for building the subsystem. These may consist of standard Linux commands (e.g. make), or custom build scripts. Finally, at the end of the Web page, click the "Save" button to save the configuration.

5 In the main Web page, click on the new job added and try to build the code by clicking the mouse over the icon named "Build Now" on the top right corner.

It could be possible that the build fails due to some missing software dependencies. In this case, it is necessary to install them manually by accessing the dedicated machine through SSH.

4.6 Tests Results

During Year 2, partners spent most of the effort in the development of their subsystems and in the integration of them in the respective prototypes. This section will provide a brief summary of results obtained from integration tests executed on the three prototypes delivered this year and results of unit tests performed on each subsystem, as described in the Section 4.4.

4.6.1 Trustworthy OpenStack Prototype

This prototype is being tested using a clone of the infrastructure built by OpenStack developers for testing their software. This choice was led by the fact that this testing environment is particularly suitable for the TClouds project, where patches developed by partners should be merged in a central repository. Another motivation was that this environment allows to run tests already defined for OpenStack, so that it is possible to verify whether new patches developed by partners break the unmodified version. Last motivation was that replicating the testing infrastructure is a relative easy task, as OpenStack developers publish the scripts used for the configuration of their platforms (`https://github.com/openstack/openstack-ci-puppet`).

As a result, we built our infrastructure by configuring three servers:

- `https://review.tclouds-project.eu`: Code Review Web site
- `https://jenkins.tclouds-project.eu`: Jenkins Web site
- `https://git.tclouds-project.eu`: TClouds Code Repository

The Code Review site allows code maintainers to review and test patches submitted by contributors before merging them in the code repository. The Jenkins Web site automatically determines if a patch can be applied on top of the current version and then, executes the test cases defined by OpenStack developers in a virtual environment. Lastly, patches that successfully pass the tests and are approved by maintainers will be merged on top of the current version in the TClouds Code Repository. More details about the instructions that partners should follow in order to submit a new patch can be found in the Appendix A.

Currently, we tested the integration of the following contributions:

- Project: `openstack/nova`

- *Modified .gitreview and disabled test_authors_up_to_date test*
- *Add scheduler filter for trustedness of a host*
- *log: added support for secure logging*
- *nova-manage: added subcommands to manage extra specs for an instance type*
- *Added ssl_verify option for the TrustedFilter scheduler filter*
- *ACaaS Scheduler*

- Project: `openstack/python-novaclient`
  - *Modified host in .gitreview*
  - *ACaaS Scheduler CLI*

In the following, we will provide some tests results obtained from the Jenkins Web site. Figure 4.4 is a snapshot of a Jenkins Web page after the latest version of the patch that introduces the *AcaaS Scheduler* in *OpenStack Nova* was submitted by OXFD to the Code Review site.

Figure 4.4: Jenkins Tests Results for Build#39 (OpenStack + ACaaS Scheduler)

This figure shows that the build was triggered by a change in the Code Review site and that the patch set will be merged into the `OXFD/ACaaS` branch of *OpenStack Nova* by Jenkins if it is approved by the maintainers of the repository and tests succeed. The figure also shows that four tests, defined by OpenStack developers, were executed: `gate-nova-docs`, `gate-nova-merge`, `gate-nova-pep8` and `gate-nova-python27`. Since the reported result was `SUCCESS`, this means that the patch set was ready for merge into the TClouds Code Repository.

The Figure 4.5 provides another example of tests results executed in a previous build, where the test `gate-nova-pep8` failed due to code style errors. This test is also important, because if the code is well written, it is more easy to maintain and to inspect when a bug is discovered.

Type	Violations	Files in violation
pep8	107	14

pep8

filename	l	m	h	number ↑
bin/nova-manage	6	14	0	20
nova/scheduler/acaas_scheduler.py	2	14	0	16
nova/scheduler/manager_integrity.py	3	13	0	16
nova/db/api.py	1	11	0	12
nova/db/sqlalchemy/api.py	0	12	0	12
nova/compute/manager.py	1	6	0	7
nova/db/sqlalchemy/migrate_repo/versions/084_security_requirements.py	1	5	0	6
nova/db/sqlalchemy/migrate_repo/versions/086_white_lists.py	1	5	0	6
nova/db/sqlalchemy/migrate_repo/versions/085_add_requirement_reference.py	1	3	0	4
nova/db/sqlalchemy/migrate_repo/versions/083_add_security_properties_to_compute_nodes.py	1	2	0	3
nova/compute/api.py	0	2	0	2
nova/db/sqlalchemy/migration.py	0	1	0	1
nova/exception.py	0	1	0	1
nova/scheduler/chance.py	1	0	0	1

Figure 4.5: Jenkins Code Style Tests Results (OpenStack + ACaaS Scheduler)

With the exception of the Resilient Log, each subsystem of the Trustworthy OpenStack prototype has been tested using a set of unit tests. For the Resilient Log, we conducted some first resource measures. The obtained results, grouped per subsystem, will be shown in the following.

LogService. The Listing 4.2 shows the output produced by the testing framework used to perform the unit tests for the LogService. As reported in the Listing, all tests produce a positive result. The test suite **`libsklog_test_suite_1`** implements the test case **/TC 4.4.8.2-1/** and includes all the tests used to check the behaviour of all functions in case of malformed input. The test suite **`libsklog_test_suite_2`** includes all others tests. In detail, the test function `test_SKLOG_initialization()` implements the test case **/TC 4.4.8.2-2/** , the function

`test_SKLOG_log()` the test case **/TC 4.4.8.2-3/** , the function `test_SKLOG_retrieve()` the test case **/TC 4.4.8.2-4/** and finally, the function `test_SKLOG_verify()` the test case **/TC 4.4.8.2-5/** .

```
     CUnit - A Unit testing framework for C - Version 2.1-0
     http://cunit.sourceforge.net/


Suite: libsklog_test_suite_1
  Test: test_SKLOG_T_NewCtx() ... passed
  Test: test_SKLOG_T_InitCtx() ... passed
  Test: test_SKLOG_T_ManageLogfileRetrieve() ... passed
  Test: test_SKLOG_T_ManageLogfileUpload() ... passed
  Test: test_SKLOG_T_ManageLogfileVerify() ... passed
  Test: test_SKLOG_T_ManageLoggingSessionInit() ... passed
  Test: test_SKLOG_T_FreeCtx() ... passed
  Test: test_SKLOG_U_NewCtx() ... passed
  Test: test_SKLOG_U_InitCtx() ... passed
  Test: test_SKLOG_U_Open_M0() ... passed
  Test: test_SKLOG_U_Open_M1() ... passed
  Test: test_SKLOG_U_Open() ... passed
  Test: test_SKLOG_U_LogEvent() ... passed
  Test: test_SKLOG_U_Close() ... passed
  Test: test_SKLOG_U_DumpLogfile() ... passed
  Test: test_SKLOG_U_FlushLogfile() ... passed
  Test: test_SKLOG_U_FreeCtx() ... passed
  Test: test_SKLOG_V_NewCtx() ... passed
  Test: test_SKLOG_V_InitCtx() ... passed
  Test: test_SKLOG_V_RetrieveLogFiles() ... passed
  Test: test_SKLOG_V_RetrieveLogFiles_v2() ... passed
  Test: test_SKLOG_V_VerifyLogFile() ... passed
  Test: test_SKLOG_V_VerifyLogFile_uuid() ... passed
  Test: test_SKLOG_V_VerifyLogFile_v2() ... passed
  Test: test_SKLOG_V_FreeCtx() ... passed
Suite: libsklog_test_suite_2
  Test: test_SKLOG_initialization() ... passed
  Test: test_SKLOG_log() ... passed
  Test: test_SKLOG_retrieve() ... passed
  Test: test_SKLOG_verify() ... passed

--Run Summary: Type      Total     Ran  Passed  Failed
               suites        2       2     n/a       0
               tests        29      29      29       0
               asserts      73      73      73       0
```

Listing 4.2: Unit test results for the core library of the LogService

Remote Attestation Service. The *RA Verifier* module of the *Remote Attestation Service* subsystem has been tested according to the test plan in Section 4.4. Test cases have been implemented in a new script, called `test_cases.py`, so that they can be run in a automatic way from the console. The following table summarizes the results of an execution of the script.

Test Case	Date	Result
/TC 4.4.9.2-1/	21/09/12	passed
/TC 4.4.9.2-2/	21/09/12	passed

The overall time required to complete both tests was about 84 seconds.

Secure Block Storage. We successfully executed the tests from the test plan Section 4.4 at 14 Sept. 2012. The tests were ran in the manual, consecutive fashion as described in the test plan. The tests were run in our Xen development environment.

Test Case	Date	Result
/TC 4.4.4.2-1/	14/09/12	passed
/TC 4.4.4.2-2/	14/09/12	passed
/TC 4.4.4.2-3/	14/09/12	passed[2]
/TC 4.4.4.2-4/	14/09/12	passed

ACaaS. We tested *ACaaS Scheduler* with the steps described in the test plan Section 4.4 at 19 Sept. 2012. These tests were successfuly executed manually in a consecutive fashion.

Test Case	Date	Result
/TC 4.4.5.2-1/	19/09/12	passed
/TC 4.4.5.2-2/	19/09/12	passed
/TC 4.4.5.2-3/	19/09/12	passed

Resilient Log. In order to give a first evaluation of the resource usage of CheapTiny compared to MinBFT, we did a few test runs with our demonstration cluster for the Resilient Log in conjunction with the Log Generator. The following table presents the average values of the runs for each protocol. We measured the number of stored events per second (Reqs/s) and the combined average CPU load of all replicas (CPU load). The cluster consists of multi-core machines, thus an average CPU load of 100% indicates the full utilization of one core. Each test run lasted 150 seconds including a warm-up time of 90 seconds.

Protocol	Reqs/s	CPU load	CPU load/Reqs/s
CheapTiny	392	80%	0.204%
MinBFT	385	120%	0.312%

The values show that under the given scenario CheapTiny incurs about 35% less CPU load per request than MinBFT. This means the comparison test passed successfully.

Test Case	Date	Result
/TC 4.4.3.2-2/	15/09/12	passed

4.6.2 TrustedInfrastructure Cloud Prototype

We successfully tested all functional tests described in Section 4.4 within a prototypical but automatically reproducible environment.

Test Case	Date	Result
/TC 4.4.1.2-1/	10/09/12	passed
/TC 4.4.1.2-2/	10/09/12	passed
/TC 4.4.1.2-3/	10/09/12	passed
/TC 4.4.1.2-4/	10/09/12	passed

[2]For debug reasons the verification of the user key is not done in the TPM.

4.6.3 Cloud-of-Clouds Prototype

We successfully executed the tests described in the Section 4.4. The results are listed below

Test Case	Date	Result
/TC 4.4.6.2-1/	10/09/12	passed
/TC 4.4.6.2-2/	10/09/12	passed
/TC 4.4.6.2-3/	10/09/12	passed
/TC 4.4.6.2-4/	10/09/12	passed
/TC 4.4.6.2-5/	10/09/12	passed
/TC 4.4.7.2-1/	21/09/12	passed
/TC 4.4.7.2-2/	21/09/12	passed
/TC 4.4.7.2-3/	21/09/12	passed

In the test **/TC 4.4.7.2-1/** the query returned the expected value. Data read from the cloud using the cloud provided console wasn't useful.

In the test **/TC 4.4.7.2-2/** 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 begining.

In the test **/TC 4.4.7.2-3/** 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.

4.6.4 SAVE Subsystem

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 [ea11b], Section 8.7).

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 4.6 shows the successful test-run.

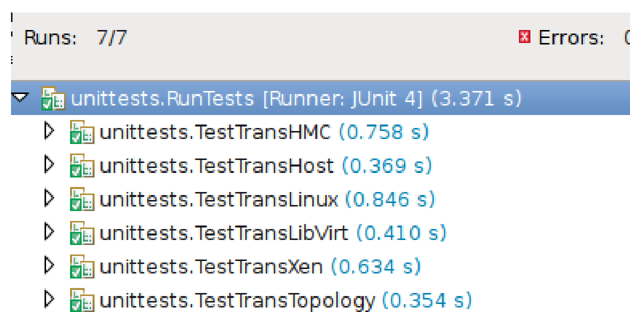

Figure 4.6: Successful JUnit Test Run.

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 a analysis of both a known-good and known-bad infrastructure that *SAVE* identified as such (D2.3.2, Section 4.7).

Part II

Prototypes Documentation

Chapter 5

Trustworthy OpenStack Prototype

The Trustworthy OpenStack prototype code is available either as a tarball, which can be downloaded from https://jenkins.tclouds-project.eu/tarballs or as a package which URL is https://jenkins.tclouds-project.eu/packages. These files are generated by our Jenkins infrastructure, described in detail in Appendix A, whenever a patch submitted by partners successfully passes the tests and is merged by Jenkins into the tclouds branch of the TClouds GIT Repository. It is also possible to set up an APT[1] repository in order to automatically install the Trustworthy OpenStack packages in the Ubuntu 12.04 LTS distribution by executing the following instructions.

Copy your SSH public key into the Jenkins server (credentials are stored in the TClouds SVN) by executing:

```
$ ssh-copy-id -i $HOME/.ssh/id\_rsa.pub tcloudsuserjenkins.tclouds-project.eu
```

Create the file /etc/apt/sources.list.d/tclouds.list to include the TClouds packages repository in the APT sources:

```
deb ssh://jenkinsjenkins.tclouds-project.eu/home/jenkins/packages precise main
```

Create the file /etc/apt/preferences.d/00-tclouds to set the highest priority to the above repository

```
Package: nova-*
Pin: version 2012.1.3+git*
Pin-Priority: 1001

Package: python-novaclient
Pin: version 2012.1+git*
Pin-Priority: 1001
```

Then, you can install Trustworthy OpenStack normally by following steps described in the documentation of the OpenStack Web site at the URL http://docs.openstack.org/essex/openstack-compute/install/apt/content. In order to use the Security Extensions of this prototype, you can refer to the documentation of specific subsystems in the remaining of this chapter. Just one remark, for dependency issues, the LogService subsystem should be installed before Trustworthy OpenStack to avoid APT errors.

[1] https://help.ubuntu.com/12.04/serverguide/package-management.html

5.1 LogService

In this section we will present the procedure to install the LogService. The core library used by the LogService is developed on Debian *Wheezy*, therefore the installation steps will be described considering as operating system a Debian-based Linux distribution.

5.1.1 Platform Setup

To install the `libsklog`, the core library of the LogService, several dependencies need to be resolved. While the major part of them can be installed using the package manager, there is a library that must be installed manually. The Listing 5.1 shows the command to install the required packages.

```
apt-get install libtool autoconf build-essential libssl1.0.0 libssl-dev uuid-dev \
  libconfig8-dev libjansson-dev libreadline-dev python-sphinx python-bottle python-devel \
  libcurl3
```

Listing 5.1: Packaged dependencies

The library `libumberlog` [NF12] needs to be installed manually. To do that it's necessary to run the commands in the Listing 5.2.

```
apt-get install pkg-config
cd /temp
git clone https://github.com/deirf/libumberlog.git libumberlog
cd libumberlog
git checkout -b libumberlog-0.2.1 libumberlog-0.2.1
./autogen
./configure
make
make install (as root)
```

Listing 5.2: `libumberlog` installation

At this point all the dependencies have been resolved. To install the `libsklog` library it's necessary to run the commands depicted in the Listing 5.3.

```
cd /temp
git clone https://github.com/psmiraglia/Libsklog.git libsklog
cd libsklog
./autogen
./configure --with-storage=rest --enable-apps
make
make install (as root)
```

Listing 5.3: `libsklog` installation

5.1.2 LogService Subcomponents

As already mentioned in the Section 3.1.2.4, the LogService is the composition of four subcomponents (Log Core, Log Storage, Log Console, Log Service Module). In this section will be illustrated how to configure and use each subcomponent.

5.1.2.1 Log Core

Within the `libsklog` sources, some sample applications are provided including one called `RESTserver.py`. Such application is a simplified version of the Log Core that implements all the main capabilities (logging session initialisation, verification and retrieval). Before starting the application it's necessary to set some values in the configuration file. In the Listing 5.4 is depicted an example of configuration.

```
#
# FILE: /usr/local/etc/libsklog/libsklog-t.conf
#

...

# Information about log storage
log_storage_host = "localhost"
log_storage_resource = "logservice/logstorage"
log_storage_port = 8080
```

Listing 5.4: Log Core configuration file

To start the execution of the Log Core run the command

```
userlocalhost:  RESTserver.py
```

Listing 5.5: Execution of simplified Log Core

5.1.2.2 Log Storage

Since the Log Storage is a subcomponent that is integrated by design with CheapBFT, to use the Log Service separately it's possible to run a dummy version of the Log Storage implemented in Python that exposes the same functionality.

```
userlocalhost:  dummy-logstorage.py
```

Listing 5.6: Execution of the dummy Log Storage

5.1.2.3 Log Console

Like the Log Core, the `libsklog` library provide a sample application also for the Log Console. Such application is called `RESTverifier` and make possible the interaction with the Log Core.

```
userlocalhost:  RESTverifier.py
```

Listing 5.7: Execution of simplified Log Console

5.1.2.4 Log Service Module

The Log Service Module is a set of functions for C languages provided by the library `libsklog`. At the moment, each function is bound also for the Python language. To use the Log Service

Module in Python it's necessary to install a Python module called `pysklog` provided with the `libsklog` sources. The installation commands are depicted in the Listing 5.8.

```
cd /temp/libsklog
pip install PySklog-0.0.1.tar.gz
```

Listing 5.8: `pysklog` installation commands

To use the module it's necessary set some values in a configuration file

```
#
# FILE: /usr/local/etc/libsklog/libsklog-u.conf
#

...

# Information about log storage
log_storage_host = "localhost"
log_storage_resource = "logservice/logstorage"
log_storage_port = 8080
```

Listing 5.9: Log Service Module configuration file

5.2 Remote Attestation Service

This section provides the documentation for the *RA Verifier* component. Installation instructions for *OpenAttestation* can be found in the `docs` folder of the official code repository (URL: `https://github.com/OpenAttestation/OpenAttestation`). The following documentation refers to the Fedora 16 Linux distribution but we are adapting it for Ubuntu 12.04 LTS.

5.2.1 Operating Environment Setup

A typical installation of the *Remote Attestation Service* requires the setup of two hosts or virtual machines. One platform, called *Database Node*, is dedicated to running the Apache Cassandra database and the other, named *OpenAttestation Node*, to *OpenAttestation* and *RA Verifier*. The former platform should satisfy at least the following hardware requirements: CPU 2.0 Ghz, RAM 2 GB or more, Hard Disk 500 GB. Instead, for the latter platform, there are no particular needs.

Regarding the software requirements, it is necessary to install the following software:

- *Database Node*
  - The `python-pip` python library (Fedora package)
  - The `pycassa` python library (install it by executing `pip-python install pycassa`)
  - The `python-fedora` python library (Fedora package)
  - The `java-1.6.0-openjdk-devel` OpenJDK JAVA headers (Fedora package)
  - The `gcc` and `make` development tools (Fedora packages)
- *OpenAttestation Node*
  - The `python-pip` python library (Fedora package)

- The `pycassa` python library (install it by executing `pip-python install pycassa`)
- The `python-matplotlib` python library (Fedora package)
- The `python-networkx` python library (Fedora package)

5.2.2 Prototype Build and Installation Instructions

5.2.2.1 Database Node

In order to use *RA Verifier* it is necessary to setup the database. First, download Apache Cassandra from the URL: http://cassandra.apache.org. From the download options, chose the latest version of the 1.0 branch. For example, the Cassandra database can be downloaded by executing the command:

```
$ wget http://it.apache.contactlab.it/cassandra/1.0.11/
apache-cassandra-1.0.11-bin.tar.gz
```

Extract files from this package in the target directory (e.g. `/srv`):

```
$ tar zxf apache-cassandra-1.0.11-bin.tar.gz -C /srv
```

Extract the files from the tarball of the *ratools* software in the target directory (e.g. `/srv`):

```
$ tar zxf ratools-1.0.0.tar.gz -C /srv
```

Install the custom libraries for Apache Cassandra by calling the `install_cassandra_libs.sh` script and by providing the directory where Cassandra was extracted. For example, the command executed could be:

```
$ ./install_cassandra_libs.sh /srv/apache-cassandra-1.0.11
```

Edit the Apache Cassandra init script `db/cassandra/init.d/cassandra` and set the BASH variable `PROGDIR` to the directory where Cassandra has been installed (e.g. `/srv/apache-cassandra`
Then, install the script and execute the service:

```
$ cp db/cassandra/init.d/cassandra /etc/init.d
$ chkconfig --add cassandra
$ chkconfig cassandra on
$ service cassandra start
```

Install the database schema by executing `cassandra-cli` in the `bin` directory of Apache Cassandra:

```
$ /srv/apache-cassandra-1.0.11/bin/cassandra-cli -h localhost -B -f
db/cassandra/schema/cassandra-schema.txt
```

Create the `/etc/ra` directory, copy the configuration files in the `db/conf` directory of the *ratools* software to `/etc/ra` and remove the suffix `.sample`:

```
$ mkdir /etc/ra
$ cp db/conf/ra.conf.sampe /etc/ra.conf
$ cp db/conf/pkgs_download_list.conf.sample /etc/ra/pkgs_download_list.conf
```

Edit the file `/etc/ra/ra.conf` and set the following BASH variables:

- `RABASEDIR` to the directory where the *ratools* tarball was extracted (e.g. `/srv/ratools-1.0.0`)
- `TARGETBASEDIR` to a temporary directory in a large partition (at least 100 GB) where packages will be downloaded (e.g. `/srv/ratools-1.0.0/Packages`).
- `CASSANDRAURL` to the <IP:port> of the Apache Cassandra database

Create the temporary directory specified in the `TARGETBASEDIR` variable:

```
$ mkdir /srv/ratools-1.0.0/Packages
```

Edit the file `/etc/ra/pkgs_download_list.conf` and configure the repository of the Linux distributions from which the `update_pkgs.sh` script will download the packages and insert the information from extracted files into the database. The file has the following format:

```
<distro ID> <distro Arch> <repo dir> <repo URL> <subdir of TARGETBASEDIR>
```

5.2.2.2 OpenAttestation Node

Install the OpenAttestation RPM package with POL patches through the `yum` command:

```
# yum install OAT-Appraiser-Base-OATapp-1.0.0-2.fc16.x86_64.rpm
```

Then, install the *RA Verifier* component by following the instructions for the *Database Node*. For the configuration, it is necessary to set only the `RABASEDIR` variable in the configuration file `/etc/ra/ra.conf` to the directory where *ratools* was extracted.

5.2.2.3 Cloud Nodes

In order to perform the measurements of software executed in Cloud nodes it is necessary to perform the following steps.

Install the provided custom Linux kernel (with IMA enabled) and the patched *systemd*:

```
# yum install kernel-3.4.2-1.ima.fc16.x86_64.rpm
systemd-37-25.torsec.fc16.x86_64.rpm
systemd-sysv-37-25.torsec.fc16.x86_64.rpm
systemd-units-37-25.torsec.fc16.x86_64.rpm
```

Create the `/etc/ima` directory and put into it the IMA policy file named `ima-policy` which content should be:

```
measure func=BPRM_CHECK mask=MAY_EXEC
measure func=FILE_MMAP mask=MAY_EXEC
```

5.2.3 Prototype Execution Instructions

5.2.3.1 Database Node

Insert data from packages of the configured Linux distribution into the database by executing the `update_pkgs.sh` script from the *ratools* software directory:

```
$ db/scripts/fedora/update_pkgs.sh
```

In addition, the script `update_pkgs.sh` allows to insert data from packages in a given directory by executing:

```
$ db/scripts/fedora/update_pkgs.sh -d <packages directory> -n <distro ID> -c
<distro arch> -t <update type>
```

where allowed values for the options are:

- `<distro ID>`: F16, F17, ...
- `<distro arch>`: i686, x86_64
- `<update type>`: newpackage, updates, testing

5.2.3.2 OpenAttestation Node

Before using the OpenAttestation Node with OpenStack, it is recommended to try the *RA Verifier* component alone by testing the verification of an IMA measurements file. Obtain it by booting a Cloud node with the updated configuration and executing:

```
# cat /sys/kernel/security/ima/ascii_runtime_measurements >
ima_measurements.txt
```

The file `ima_measurements.txt` should be similar to:

```
10 000...000 ima 000...000 boot_aggregate
10 1f3...c2d ima 0fb...278 /lib/systemd/system-generators/systemd-getty-generator
10 7c8...874 ima ef0...20b ld-2.14.90.so
10 084...3a2 ima ab1...3e2 libselinux.so.1
10 25d...36f ima 624...f06 libcap.so.2.22
10 8c4...cae ima 4c9...51c librt-2.14.90.so
10 ecc...b70 ima c33...456 /lib/systemd/system-generators/systemd-cryptsetup-...
10 a7a...917 ima 0d3...eb3 /lib/systemd/system-generators/systemd-rc-local-...
10 819...978 ima db2...15b libc-2.14.90.so
10 ae1...275 ima e77...b27 libdl-2.14.90.so
10 3ff...306 ima 893...008 libattr.so.1.1.0
10 7ec...75f ima e5c...3f6 libpthread-2.14.90.so
10 e3c...8bf ima 967...726 /lib/systemd/systemd-readahead-collect
10 c79...cfc ima 870...2d9 libsystemd-daemon.so.0.0.0
10 07e...416 ima 57d...d3d libudev.so.0.12.0
10 d85...7c3 ima a64...89d libgcc_s-4.6.3-20120306.so.1
10 7ac...4a1 ima f70...23a /lib/systemd/systemd-cgroups-agent
10 933...095 ima f5b...3a1 libdbus-1.so.3.5.6
```

Then, copy this file to the *OpenAttestation Node* and verify the IMA measurements by executing the `ra_verifier.py` script from the *ratools* software directory:

```
$ verifier/ra_verifier.py -i ima_measurements.txt -t openattestation
```

It should return an output like:

```
ok:  212, unknown:  0, pkg_security:  0, pkg_not_security:  0
```

5.2.3.3 Trustworthy OpenStack Management Node

Edit the configuration file `/etc/nova/nova.conf` and add the following lines at the end of the `[DEFAULT]` group so that *Nova Scheduler* filters the Cloud nodes that can run a VM depending on the integrity level specified by users:

```
scheduler_available_filters=nova.scheduler.filters.trusted_filter.TrustedFilter
scheduler_default_filters=TrustedFilter
```

Then, add these lines at the end of the same file to tell *Nova Scheduler* that it should contact the *OpenAttestation Node* at the specified address in order to obtain the current integrity level of a Cloud node:

```
[trusted_computing]
server=<IP of OpenAttestation Node>
port=8443
```

5.3 Access Control as a Service

5.3.1 Platform Setup

Ubuntu 12.04 is deployed as the base system on every node, installed and configured with default packages . The prototype relies on a complete deployment of OpenStack with ACaaS patches, and the Trusted Computing Infrastructure. A typical deployment includes deploying one node as the management nodes, and several nodes as the compute nodes.

5.3.1.1 ACaaS Setup

ACaaS prototype is implemented on OpenStack Essex release. Its building and installation are as simple as applying ACaaS patch to Essex source codes, and then following the general python software building procedure. As described above, two major components are modified: the OpenStack Nova Compute, and the python-novaclient.

1 OpenStack Nova

- Fetching nova source code

```
$apt-get source nova-compute
```

- Applying ACaaS patches

```
$cd nova-2012.1
$patch -p1 < nova-acaas.patch
```

- Build and install

```
$python setup.py build
$sudo python setup.py install
```

- Restart the services (For management node)

```
$sudo restart nova-api
$sudo restart nova-scheduler
```

(For compute node)

```
$sudo restart nova-compute
```

2 Python-novaclient

- Fetching python-novaclient source code

```
$apt-get source python-novaclient
```

- Applying ACaaS patches

```
$cd python-novaclient
$patch -p1 < novaclient-acaas.patch
```

- Build and install

```
$python setup.py build
$sudo python setup.py install
```

5.3.1.2 Trusted Computing Infrastructure Setup

The management node is deployed with the nova-scheduler, nova-api, central database, rabbitmq. A compute node is deployed with nova-compute, and optionally nova-volume and nova-network. These deployments and configurations follow the general OpenStack settings, which are well documented and can be found online, e.g. the ”OpenStack Install and Deploy Manual” (http://docs.openstack.org/essex/openstack-compute/install/apt/content/index.html)

To enable ACaaS, the ACaaSScheduler should be specified as the computer scheduler driver, which can be achieved by modifying the /etc/nova/nova.conf to have the corresponding field set as follows:

```
compute_scheduler_driver = nova.scheduler.acaas_scheduler.ACaaSScheduler
```

Trusted Computing Infrastructure is deployed for enabling the Trusted-based scheduling, providing by ACaaS prototype. It is composed of the integrity measurement and reporting service on the compute nodes, and the remote attestation service on the management node.

1 Measurement Services

The measurement services build the chain-of-trust on every compute node from its Core-Root-of-Trust-for-Measurement, a specific piece of code in it BIOS up to every software component running on top of it. The chain-of-trust is built in an iterative method during the bootstrapping procedure of the node, i.e. every component participating in the booting procedure measure the next component before loading and giving control to it. The term ’measure’ in TCG terminology stands for taking the hash value of the target software component and store the value into the TPM.

Consequently, every software component responsible for booting a platform should be modified to support the measurement services, namely the BIOS, bootloader, and the OS kernel. First of all, the integrity measurement service should be turned on in the BIOS to enable the first-step measurement for measuring the bootloader. Secondly, the trusted BIOS should be installed to implement the trusted boot for measuring the kernel, e.g. the TrustedGrub (http://sourceforge .net/projects/trustedgrub/). Finally the Linux kernel on our base system should have its IMA component built and enabled, for implementing the measurement of every software component loaded on the platform. To implement this, the kernel of the base system (Ubuntu 12.04) should be re-configured and compiled with the IBM IMA configuration turned on. The kernel-arg entry in the grub configuration should also be added with the ”ima_tcb” argument.

2 Remote Attestation Service

Remote attestation service is implemented by the OpenPTS system in our prototype. The deployment of OpenPTS includes the measurement reporting sub-system on the compute nodes, and attestation sub-system on the management node:

(a) Setting up the compute nodes

- Installing OpenPTS components

```
 $sudo apt-get install trousers tpm-tools libtspi-dev libtspi1
$sudo dpkg -i openpts-0.2.6-2.x86_64.deb
```

- Setting up TPM

```
 $tpm_take_ownership -y -z
```

- Configure ptsc
Adjust the configuration le /etc/ptsc.conf, choosing/configuring the appropriate reference models (rm). The detailed rm configurations are specific to each platform and are out of scope of this documents. Detailed information can be found at
http://sourceforge.jp/projects/openpts/. An exemplar is as follows:

```
 irm.num=2
rm.model.1.pcr.4=grubpcr4hdd.uml
rm.model.1.pcr.5=grubpcr5 uml
rm.model.1.pcr8=grubpcr8.uml
rm.model.1.pcr.10=f12imapcr10.uml
```

- Initialize Collector ptsc

```
 $/usr/sbin/ptsc -i
```

- Selftest the target platform

```
 $/usr/sbin/ptsc -s
```

- Startup tcsd and ptsc

```
 $service trousers start
$service ptsc start
```

- Set whether ptsc should run on startup

```
 $chkconfig --add ptsc
```

(b) Setting up the management nodes

- Installing OpenPTS components

```
 $sudo apt-get install trousers tpm-tools libtspi-dev libtspi1
$sudo dpkg -i openpts-0.2.4-1.x86_64.deb
```

- Setup SSH public key authentication between compute nodes and management node

```
 $ssh-keygen
$ssh-copy-id ptsc@compute_node_N
```

- Enrollment with trust collector

```
$openpts -if compute_node_N
```

- Testing attestation with trust collector

```
$openpts -l ptsc -v compute_node_N
```

(c) Setting up the nova periodically attestation

To enable the periodically attestation supported by ACaaS Scheduler, ACaaS Scheduler Manager should be enabled, which can be achieved by modifying the /etc/nova/nova.conf to have the corresponding field set as follows:

```
scheduler_manager=nova.scheduler.manager_integrity.SchedulerManager
```

SchedulerManager is a modification to the OpenStach TrustedComputingPool. Instead of using the RESTful API, the SchedulerManager in ACaaS performs attestation directly. In our prototype this is achieved by invoking the OpenPTS facilities as described above. The following parameters can be specified in /etc/nova/nova.conf for controlling the openpts operation, with the default values given:

i. Enabling openpts When openpts is disabled, attestation in ACaaS is in demo mode: simply output text indicating the operation to invoke. Without a well-configured openpts infrastructure, enabling it will cause ACaaS to hang.

```
openpts_enabled = False
```

ii. Path of openpts command

```
openpts_bin = '/usr/bin/openpts'
```

iii. Path for storing aide db for openpts The aide db is used as the white-list, representing the expected trusted properties of a target node

```
openpts_aide_path = '/var/lib/aide/aide.gz'
```

iv. ssh username required by openpts

```
openpts_username ='ptsc'
```

v. ssh port number required by openpts

```
openpts_port = ''
```

vi. ssh key file required by openpts

```
openpts_key = ''
```

By modifying the configuration files, and if necessary, a minor portion of the scheduler manager source codes, other attestation facilities can easily be switched to.

5.3.2 Management Console

5.3.2.1 Requirement Management

Requirement management facilities provide functionalities for creating, removing and querying specifies user requirements for VM scheduling. The management interfaces are listed as follows:

- List all requirements

```
$nova-manage requirement list
```

- Create new requirement, with a string as the name of the requirement (e.g. location)

```
$nova-manage requirement create --requirement=NAME
```

- Remove the requirement with specific ID (e.g. 9)

```
$nova-manage requirement remove --id=ID
```

5.3.2.2 Trusted Requirement Management

Trusted Requirement management facilities provide functionalities for adding, removing, querying trusted properties (the while list database). These properties represent the expected configuration (or state) of a node. They can be trustworthy examined (attested to) by the trusted computing remote attestation as described above. Any violation of the attestation result to the white list (the expected configuration) indicates the untrusted state of the node, and will result in the node to be examined and re-initiated (in current prototype, the state-changed host with be removed with the white list property).

- List all white-lists

```
$nova-manage white-lists list
```

- Create new white-lists With a string as the name of the white-lists and the location for storing the white-list database as value for the name

```
$nova-manage white-lists create --white-list=white-list-file-path
```

- Remove the white-lists with specific WL_ID

```
$nova-manage white-lists remove --id=WL_ID
```

5.3.2.3 Security Properties Management

Security Properties management facilities provide functionalities for adding, removing, querying properties of a compute node. The management interfaces are listed as follows:

- Query a HOST for its security properties

```
$nova-manage host get_properties --host=HOST
```

- Add a security properties to a HOST Specified by NAME and VALUE. When NAME equals to ”trusted-host”, it represented the trusted properties and its VALUE should identifies one of the TP_IDs

```
$nova-manage host add_properties --host=HOST --properties="'NAME': 'VALUE',
..."
```

- Specify white-list White-list is specified to a host as a requirement to provide a same management interface. In current implementation, the name for the white-list requirement should be: whitelist.

```
$nova-manage host add_properties --host=HOST --properties="'whitelist':
'WL_ID'"
```

- Remove a security properties from a HOST, specified by NAME

```
$nova-manage host remove_properties --host=HOST --properties="['NAME', ...]"
```

5.3.2.4 Requirement-based VM Instantiation

- Initiating a VM with with general requirement matching The REQ_ID represents the requirement id for each requirement defined by the requirement management component, and the REQ_VALUE will be the expected value matching the specific requirement.

```
$nova boot --flavor XXX --image XXX --key_name XXX --security_group XXX
--req="REQ_ID:'REQ_VALUE'" IMAGE_NAME
```

- Staring VMs with exclude-user requirements A predefined alphabic REQ_ID is specified for these type of requirement matchings. The ’exclude-user’ is represented by ’x’ or ’X’ as REQ_ID to specified the VM can only be launch on the compute host with NO other VMs belonging to the users identified by USER_IDs.

```
$nova boot --flavor XXX --image XXX --key_name XXX --security_group XXX
--req="'x':['USER_ID1', 'USER_ID2', ...]" IMAGE_NAME
```

- Staring VMs with trusted-hosts requirements A predefined REQ_ID is specified for these type of requirement matching. The ’trusted-host’ is represented with ’t’ or ’T’ as REQ_ID to specified the VM can only be launch on the compute host with a target attestation white-list (representing a set of trusted properties) identified by TP_ID.

```
$nova boot --flavor XXX --image XXX --key_name XXX --security_group XXX
--req="'w':TP_ID" IMAGE_NAME
```

5.4 Cryptography-as-a-Service (Caas)

5.4.1 Operating Environment Setup

5.4.1.1 Hardware Prerequisites

Cryptography-as-a-Service requires (in a minimal configuration) two platforms: a computing platform, on which the user's VMs are deployed, and verifying platform (short *Cloud Verifier*), which delegates management tasks between user and computing platform and which verifies the trustworthiness of the computing platform. The computing platform requires certain capabilities from the underlying platform:

Trusted Platform Module v1.2 A TCG TPM in version 1.2 must be present on the platform. Moreover, the module must be activated and owned. The owner and Storage Root Key (SRK) authentication values must be made available to the CaaS software.

Hardware Virtualization Support The CPU and chipset must support hardware virtualization, e.g., Intel VT-d or AMD-V. In the current version of CaaS only Intel VT-d is supported.

5.4.1.2 Download and Installation

CaaS builds on the Xen hypervisor in version 4.1.2, which can be retrieved from the Xen project webpage[2], the MiniOS (from the Xen 4.1 source tree), and Intel tboot[3] for a measured launch of the software stack. The corresponding patches (`libxl` and `libxc`) for Xen 4.1.2 and MiniOS to enable CaaS support and install a basic configuration/templates for a domain builder and crypto-service VM can currently only be retrieved upon request from partner TUDA[4]. After applying the patches to the Xen source code and MiniOS, this code is compiled and installed according to the default Xen documentation.

Our current prototype has the following minimal software requirements for building and executing Xen:

- GCC v3.4 or later
- GNU Make
- GNU Binutils
- zlib-dev
- Python v2.3
- libncurses-dev
- openssl-dev
- xorg-x11-dev
- uuid-dev

[2] http://www.xen.org/
[3] http://sourceforge.net/projects/tboot/
[4] *stefan.nuernberger@trust.cased.de* and *sven.bugiel@trust.cased.de*

- bridge-utils package (/sbin/brctl)
- iproute package (/sbin/ip)
- hotplug or udev
- GNU bison and GNU flex
- GNU gettext
- 16-bit x86 assembler, loader and compiler (dev86 rpm or bin86 & bcc debs).

On the verifier platform, Python is required for executing the verification and delegation service.

5.4.1.3 Configuration

Before first boot, the CaaS software has to be configured to be able to use the platform's TPM. This is achieved by entering the SRK and owner authentication values into the *TPM_Auth.cfg* configuration file.

For the first boot, the default steps for booting a vanilla Xen with Intel tboot should be followed. During boot, the domain builder and management domain will automatically be launched. On first boot, the domain builder VM will automatically create a new TPM binding key, which is bound to the measurement of the platform and provided at a world-wide readable location. This key has to be provided to the user to enable encryption of VM images and secrets deployed in the cloud by the user.

5.4.2 Prototype Execution Instructions

After the system has booted and the configuration was completed, the workflow looks as follows:

- Cloud Verifier attests the Xen-based computing machine, and receives the public key of the certified binding key for this configuration.
- The user uploads his VM image to the cloud, and sends his user secret encrypted under the public TPM binding key to the CV.
- The computing platform asks the CV for the user secret in order to run the VM image.
- The CV sends the encrypted secret to the computing platform.
- The domain building code on the computing platform uses the TPM to decrypt the user secret, the VM image by using the secret, and starts building the domain.

For a detailed description of the inner workings during user VM launch, we refer to deliverable D2.1.2 [ea12b].

5.5 Resource-efficient BFT (CheapBFT)

5.5.1 Operating Environment Setup

5.5.1.1 Hardware Prerequisites

A system setup of CheapBFT comprises at least four machines. Three server systems are required with the following specification:

- x86 (better x86-64) machines with identical or at least similar hardware specs; at least Intel Pentium 4 class processor
- 4 GB RAM
- PCI bus (necessary for the FPGA-cards, see below)
- 1 Gigabit/s or better IP-network connectivity
- Enterpoint ”Raggedstone 1” FPGA card

In order to programm the FPGA cards, a Xilinx JTAG programmer with USB interface is necessary.

Further, one or two additional computer with similar specification as the server system (except the FPGA cards) are required for running the benchmark tools.

5.5.1.2 Software Environment

Following software should be installed on all test machines:

- Linux distribution with kernel 2.6.32 or better (e.g. Ubuntu 10.04), 32- or (preferably) 64-bits.
- `xz` data compression tools
- Eclipse 3.6 (Indigo) or better
- Java Runtime environment 6.0 or better
- Secure Shell Server (OpenSSH) with key-based authentication to enable scripted remote-logins
- `screen` terminal multiplexer
- Xilinx ISE WebPack 13.x for Linux
- Header files and build environment for the running Linux kernel

5.5.1.3 Installation

The source code is provided as a GNU tarball, packed with the xz data compression utility. In order to unpack it on a GNU/Linux system, use the following command: `xzcat cheapbft.tar.xz | tar xf -` All files are extracted into the subdirectory `cheapbft/`. The following subsections will describe the necessary steps to configure and translate the program code to get it into a working state. All specified paths are relative to the cheapbft directory, unless beginning with a forward slash (/), or mentioned otherwise.

FPGA Hardware The source code for the FPGA firmware and associated Linux kernel driver is located in the directory `cash/pci_base`. In order to build the firmware image, the Xilinx ISE tools must be configured properly and present in the search path of the Linux shell. This can be usually accomplished running the command `source settings32.sh` in the installation directory of the Xilinx suite.

Each of the three FPGA boards must be assigned an unique identifier so that to they generate different signatures. This ID can be adjusted in the file `fpga/pci_base/source/adrdec.vhd` in the line starting with `constant SYS_ID`. The value is at the end of this line, enclosed in double quotes and must be a 16-bit hexadecimal number.

To compile the firmware image, change into the directory `fpga/pci_base` and simply call make. The shipped Makefile will create a `.bit` file suitable for the FPGA chip of the "Raggedstone 1" board. This build process can take up to an hour, even on fast machines, as the logic is quite complex and requires expensive optimization processes to fit onto the FPGA.

When the build process has finished, the next step is to load the firmware onto the card. Make sure the USB programmer is connected to the Raggedstone board. There are two possible modes to store the firmware: The command `make load` will just reconfigure the FPGA and keep it until the host PC is turned off. The command `make i-really-want-to-flash` can be used to additionally save the image in persistent flash memory, so that it will survive when the machine is powered down. Just loading the firmware is the preferred mode when working on the FPGA code, as the flash memory has a limit on the number it can be erased. The host PC has to be rebooted before.

As each card requires an unique ID embedded in its firmware, the image has to be rebuilt and loaded for each of the three FPGA boards individually. After programming the FPGA, the PC hosting the card should be rebooted as soon as possible, otherwise the card won't be detected properly and may interfere with the operation of the machine in unpredictable ways.

Kernel Driver In order to use the cards from any Linux application software, it is necessary to install a kernel driver that can interface with the hardware registers on the FPGA board. The driver provides a device node called `/dev/counter` for the protocol implementation. To compile the driver, change into the directory `fpga/pci_base/kernel` and edit the Makefile so that the `KDIR` variable points to the directory containing the header files for your current kernel version.

Type make (using your regular user account) in order to build the driver. When the module `counter.ko` was built successfully, switch to the root account (e.g using `su` or `sudo`). Now the driver can be loaded using `make device`. This command will also create the necessary device node and allows everyone to use it. For security reasons it is advisable to change permissions for `/dev/counter`, so that access is restricted to the user account(s) that will run the application protocol later.

Userland applications The userland software is entirely written in the Java language (except for a few programs for testing purposes) and ships in form of multiple Eclipse projects. To compile and install them, it is necessary to import two projects into the Eclipse workspace:

- Modular-SMaRt from the folder `modular-smart/`
- CheapBFT from the folder `cheap/`

Do not copy the project directories into your workspace upon import, as they contain relative references to other directories in the source tree. If everything is set up, the projects can be built

with the normal `Build project` function in the menu. As the CheapBFT project depends on class files in the Modular-SMaRt, it is necessary to compile the Modular-SMaRt tree first.

5.5.2 Prototype Execution Instructions

The benchmark suite is implemented as a set of bash scripts that connect to the participating machines via SSH and set up the required services. The progress can be monitored by switching between terminals in the `screen` software that is used to manage the parallel execution of aforementioned scripts. The required files are all stored in the directory `cheap/bin/`.

There is a script for each test scenario. For instance, `run_micro.bash` can be used to start micro benchmarks. However, the main functionality is implemented in `system.bash`, which is imported in every script for a scenario in order to provide consistent use.

The behavior of the running system is configured via three configuration files which are stored in `cheap/bin/config`: `system.config` is the main configuration file, which contains, for example, settings concerning the used consensus protocol. `hosts.config` contains a list of all replica endpoints and `logging.properties` can be used to configure the logging output generated during a run. Instead of editing these configuration files directly, the scenario scripts (or `system.bash`) should be used in conjunction with configuration templates (also contained in `cheap/bin/config`) to set up the system. For instance, `system.config.cheap` is a template for a configuration with CheapBFT as consensus protocol. The same holds for `system.config.cheap.sof`, with the exception that a software module is employed for the message verification instead of an FPGA.

Before a test run can be started, the addresses of the hosts executing servers and clients have to be configured. The addresses are usually stored in files named according to the pattern `hosts.config.*`. These files contain two variables: `ENDPOINT_SERVERS` and `HOSTS_CLIENT`. The former is a list of machines equipped with the FPGA boards together with a base port. The specified machines act as servers and server sockets are opened with port numbers starting from the given base port as required. (For that purpose, at least five ports starting from the base port should be available.) The latter of the two variables is a list of host addresses only. Here, the clients for the test run are executed. Please ensure that your current user account can login to all specified server and client hosts via SSH, without being asked for a password. This can be accomplished by either deploying host-based authentication or (probably easier) just supplying proper SSH keys for the user.

After the host addresses have been provided, the test configuration has to be set up. For example, the command `./system.bash setup_prot cheap` sets `system.config.cheap` as protocol configuration, hence selects CheapBFT as consensus protocol. `./system.bash setup_hosts <yourhosts>` initializes the server and client hosts according to the file `hosts.config.<yourhosts>`. With `./system.bash setup_logging con` logging output is redirected to the console.

Next, a screen environment can be started by running `./system.bash screen` This should start the `screen` program with a basic configuration from which the individual client and server programs will be launched.

The actual benchmark can be started, for example, using `./run_micro.bash start` This will connect to the previously configured host machines and run a minimal performance benchmark. Statistics are printed in regular intervals to the screens.

All available commands and their options can be obtained by `./run_micro.bash help` (or `./system.bash help`).

Chapter 6

TrustedInfrastructure Cloud Prototype

6.1 TrustedObjectsManager setup

Essentially the TrustedObjectsManager (TOM) is a webserver that provides the necessary management and administrative functions via a web interface.
The managment console holds the complete set of configuration-files for all TrustedDesktops and TrustedServers, which are attached to the TOM. These appliances establish a permanent secure tunnel to the management console and get their dedicated initial configuration, as well as configuration changes via this tunnel automatically (TrustedChannel).
The mentioned appliances derive the settings (firewall-, router-, security-, user-settings, etc.) which have to be applied, from the centrally downloaded configuration and apply them autonomously.

This chapter describes the steps to be done in order to attach a TrustedServer and a TrustedDesktop to the TOM in order to start, stop install and remove a compartment (an virtual machine (VM) associated with a trusted virtual domain (TVD)) and to use its provided services.

6.1.1 Using the management console

The web interface of a newly installed TOM is reachable via the https-adress 192.168.1.11 from any client's webbrowser within the same network, showing the login screen as in Figure 6.1. The TOM's IP-address can be changed later on. After logging in with the predefined credentials the screen appears as shown in Figure 6.2.

6.1.2 Creating a company

At first a "company"-object has to be created, to store the essential structures like VPNs, (local) networks, users, TVDs, compartments and their relationships within a company or a project. In Figure 6.3, the company "TCLOUDS" is created via a right-click on "Management Console" and selecting "New".

6.1.3 Creating a location

Within this newly created company one has to create a new location in order to logically sort different company branches. Here the location "Bochum" is created by expanding the sub-tree menu, selecting "Locations" and a right-click choosing "New..." and "Location".

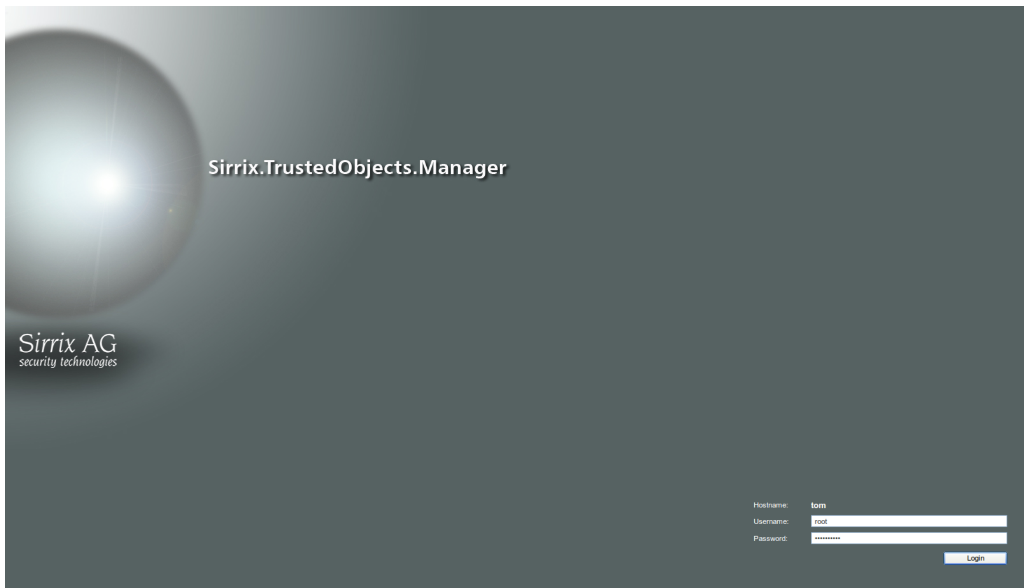

Figure 6.1: The TrustedObjectsManager Login screen

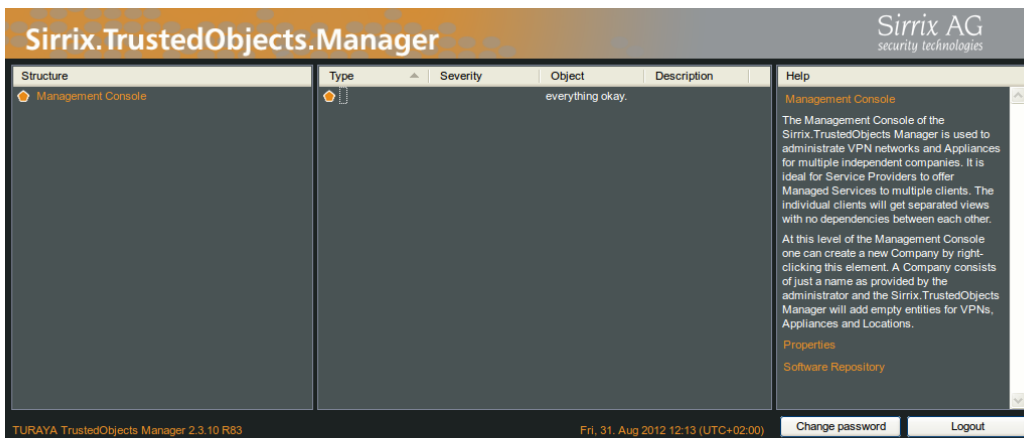

Figure 6.2: The TrustedObjectsManager overview screen after login

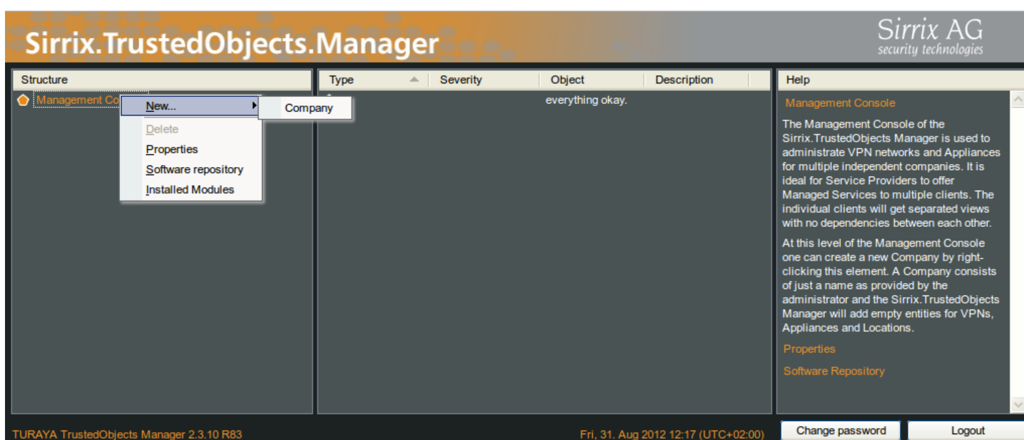

Figure 6.3: Creating a “Company”

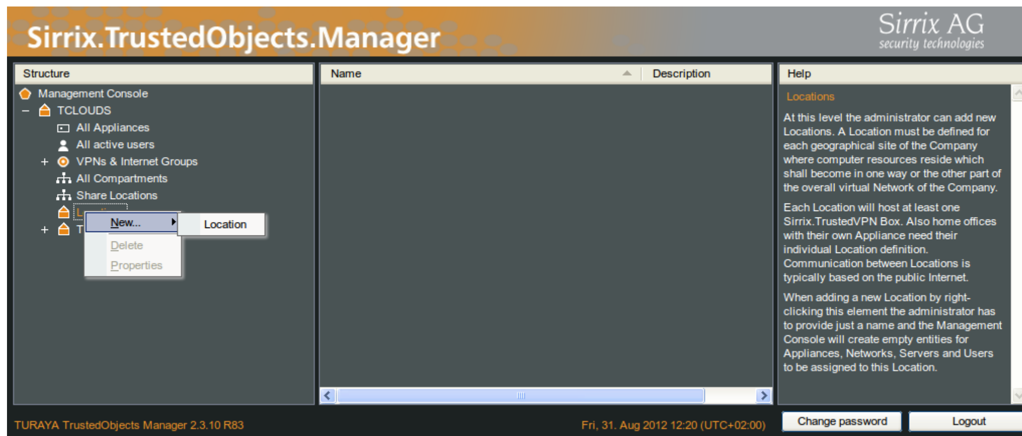

Figure 6.4: Creating a “Location”

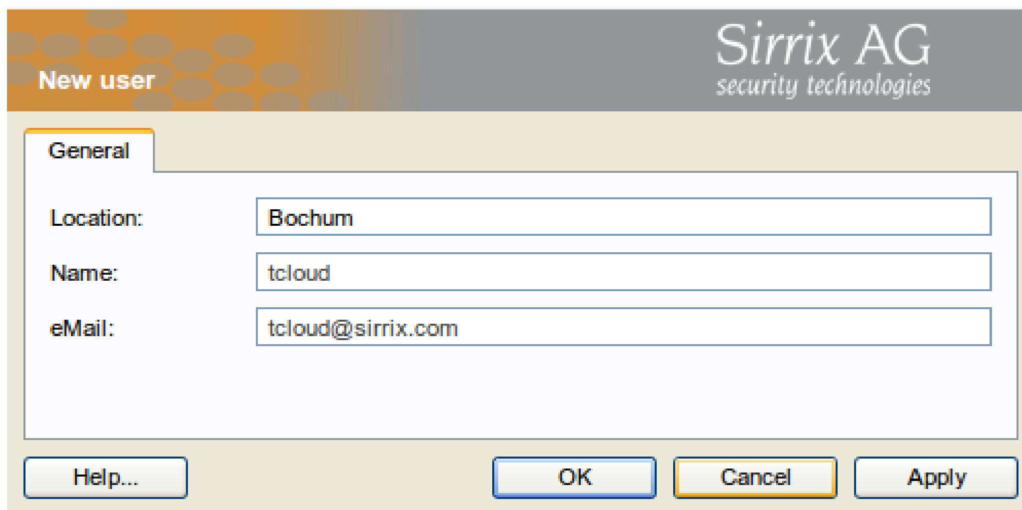

Figure 6.5: Adding a user, step 1

6.1.4 Adding users

Choose the newly created location “Bochum” and open the sub-tree by a left-click. The appearing “Users”-entry allows to add users to the infrastructure via a right-click selecting “New”, “User”. The form has to be filled out with a real-name and an existing email-address (Figure 6.5). After clicking “Apply”, three additional tabs appear, whereat the “Login Name”, a “Password” which has to be confirmed, and a “Hold-back time” has to be entered. The “Expiry Date” is calculated based on the value entered in the “Hold-back time” field (Figure 6.6).
Only those users entered in the “Users” sub-tree are granted access to the whole infrastructure.

6.1.5 Network configuration

Within the newly created location, the local networks have to be defined, choosing “Locations”, “Bochum” and a right-click on ”Networks (Figure 6.7). Here, the 4 networks “TD-ConfidentialNet”, “TD-PublicNet”, “TS-ConfidentialNet” and “TS-PublicNet” are created with different IP-ranges. These networks i.e. IP-adress-ranges, will be attached to the virtual machines’ net-

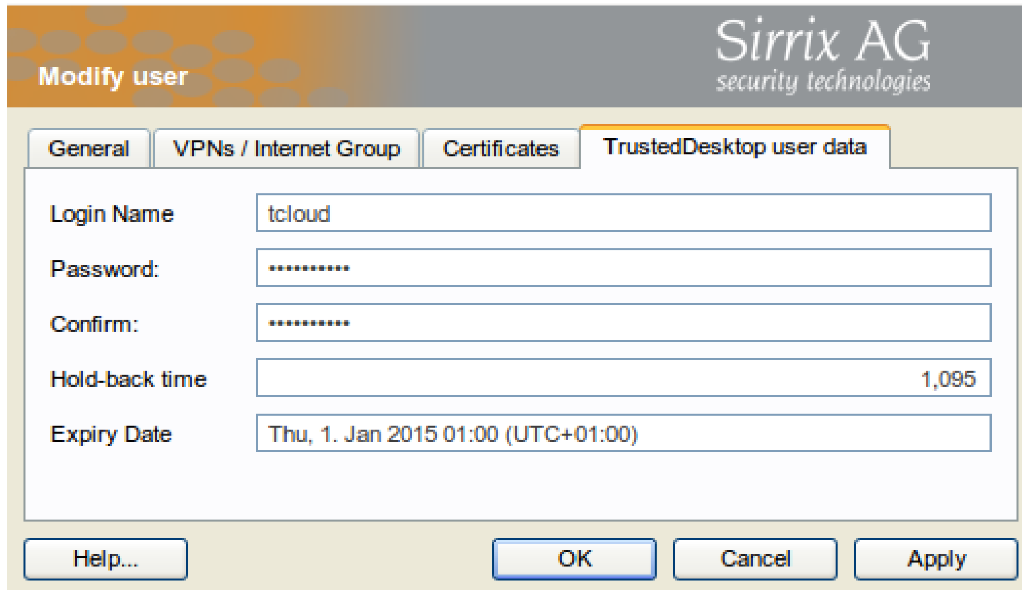

Figure 6.6: Adding a user, step 2

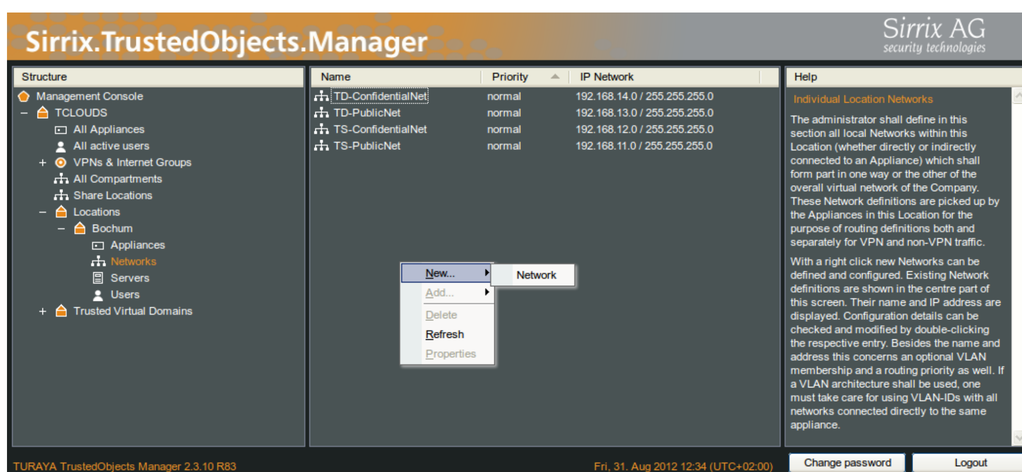

Figure 6.7: Adding networks

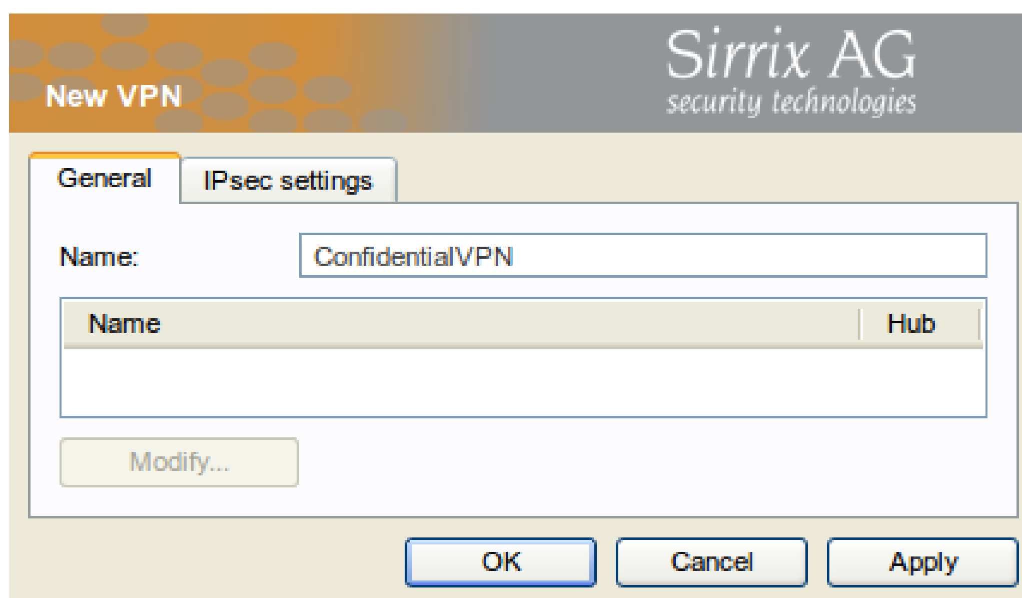

Figure 6.8: Adding a VPN

work interfaces running on the TrustedServer (TS) and on the TrustedDesktop (TD). Furthermore the "Confidential"-networks, as well as the "Public"-networks will reside within the "Confidential"- or "Public"-VPN respectively, which themselves will be part of the "Confidential" or "Public"-TrustedVirtualDomain (TVD). The assignment of networks to VPNs and TVDs ensures a separation of information-flow on the network level.

6.1.6 Creating and configuring VPNs

Logical VPNs are created by choosing "VPNs & Internet Groups", "New", "VPN". The appearing dialog (Figure 6.8) requires in this step just a name, here "ConfidentialVPN". The assignment of networks to VPNs (see: Section 6.1.5), takes place in a later step (see: Section 6.1.10).

6.1.7 Attaching appliances

Adding appliances like TrustedServer or TrustedDesktop to the company takes place within the sub-entry "Appliances" of a "Location". Right-clicking "Appliance" and choosing "New", "Appliance" opens a window like in Figure 6.9. The appliance has to be named, and the "Serial number" has to be entered. Here the TrustedServer is created. This 5x5 alphanumeric number has to gathered from the console of a newly installed TrustedDesktop or TrustedServer. The "Interface" field is greyed out and set statically to "eth0 - external network interface". This cannot be changed currently. If the appliance to attach is reachable via a static IP-address the "Static" radio-button has to be chosen and the appropriate fields "IP / Mask", "Gateway" and "DNS 1 / DNS 2" have to be filled. In case, the appliance gets its IP-address from a DHCP-server, the "Dynamically using DHCP"-option has to be chosen.
After clicking the "Apply" button, the "Download configurations / software update"-button will become active. By clicking this, a dialog opens, where the "configurations"-file can be downloaded. This file has to be stored on the root-folder of an USB-drive.
Attaching the USB-drive to the newly installed TrustedServer or TrustedDesktop, the configu-

Figure 6.9: Add a new appliance to the company

Figure 6.10: Dialog to download the configuration for the specific appliance

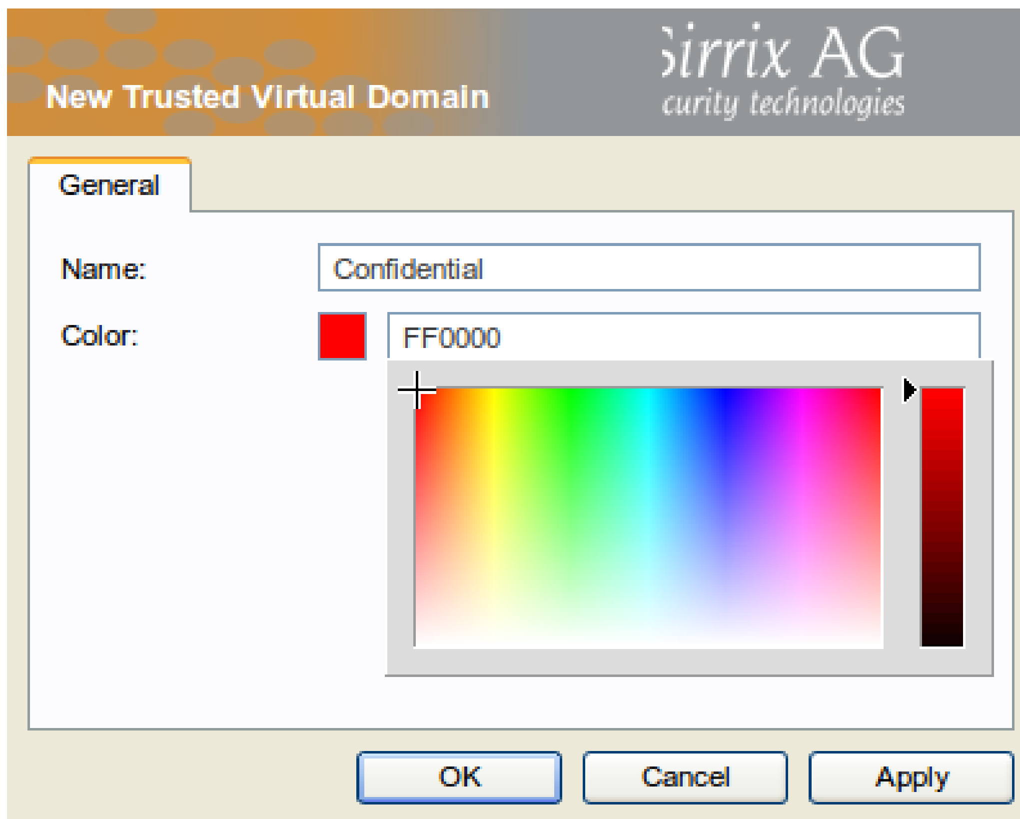

Figure 6.11: Dialog to create a new TVD

ration will be found and applied to this dedicated machine. Not till then, the appliance is able to connect to the TrustedObjectsManager, getting additional configuration settings after that the integration of the appliance to the infrastructure will be finished.

6.1.8 Creating TrustedVirtualDomains

To create a new TVD, “TrustedVirtualDomains” can be chosen from the context menu, right-clicking and choosing “New”, “TrustedVirtualDomain”. The TVD has to be named and a color has to be chosen from the color palette (Figure 6.11). The chosen color will appear as an visual border around the running compartment on TrustedDesktop. As shown in the Figure 6.11, in this scenario the TVDs “Confidential” and “Public” are created.

6.1.9 Adding compartments to TVDs

In order to add a virtual machine disk image to a TVD, one chooses for example “TrustedVirtualDomains”, “Confidential”, “Compartments” and right-clicks on “New”. The new compartment screen (Figure 6.12) asks for the name, and a description of the compartment. The “Availability” checkbox has to be checked, in order to allow clients to download and use the compartment. Furthermore the virtual disc image template has to uploaded. If there are already files registered at the underlying database, one of these can be selected. Otherwise a new virtual disc image can be uploaded via the “Manage”-button. Press “Add local file”, if the *.file is already on the TOM’s harddisk or “Upload” in order to upload a *.vdi-file from remote.
An explaining comment of the image’s content is optional. The “Last change” field in the

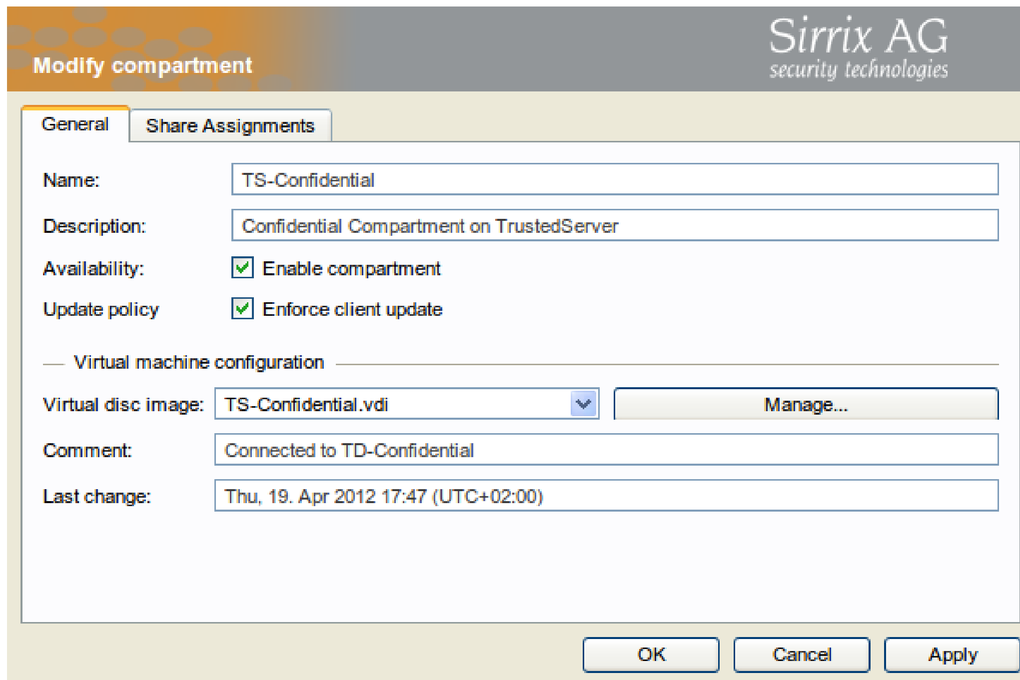

Figure 6.12: Adding a new compartment

"New compartment" window is automatically filled with the derived virtual disc image's timestamp.

6.1.10 Connecting everything together

The available compartments, uploaded to the TOM in Section 6.1.9 can now be installed to the TrustedServer by choosing the "Properties" of the appliance (Figure 6.13). Switching to the "Compartments"-tab allows the user to choose "Install new..." and select the desired compartment. The remote installation of the compartment starts immediately after a click on "OK" in case the TrustedServer is online.
Now, the corresponding network, defined in Section 6.1.5 has to be attached to the installed compartment. For that purpose, the registered appliance has to be modified via "Properties".

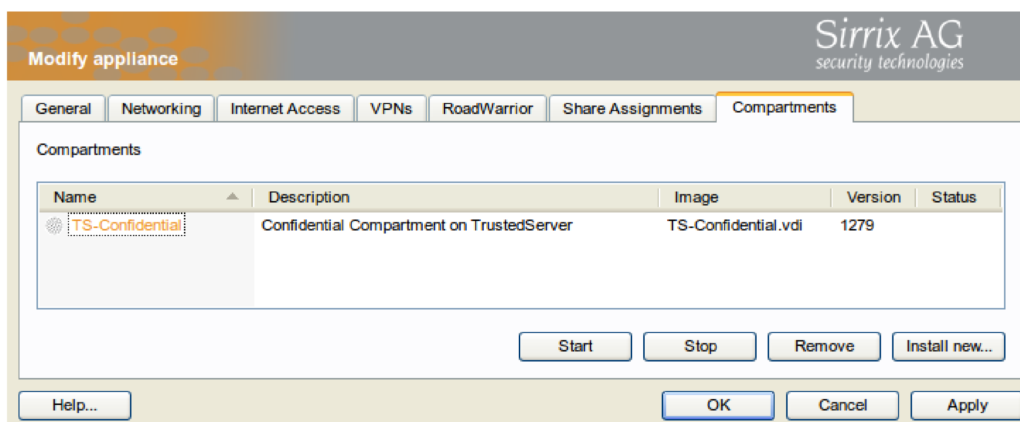

Figure 6.13: Installing a registered compartment to TrustedServer

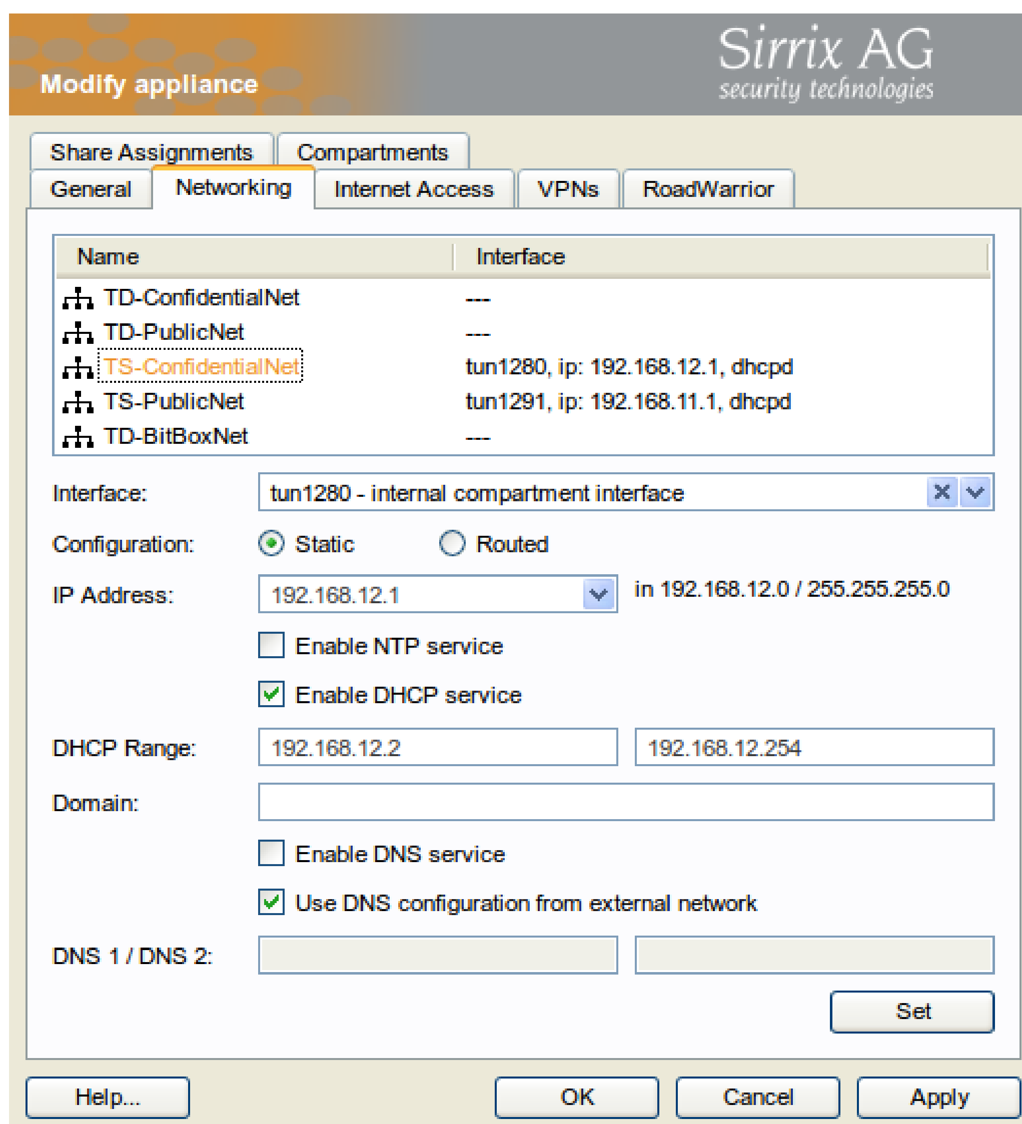

Figure 6.14: Attaching networks to compartments installed on TrustedServer

The networking-tab, lists all installed compartments of the chosen appliance. Choosing one of them ("TS-ConfidentialNet" in Figure 6.14) leads to the selection of an "Interface", which is an entry like "tun1280 - internal compartment interface". All upcoming fields and check-boxes are automatically set to the values already defined during the step of creating networks. See Section 6.1.5. Pressing "Set" and "OK" applies all changes made to this tab. The installed compartment on the TrustedServer will now get a network connection after startup.
In order to separate the information-flow between different TVDs on different machines the VPNs are now set up finally. Dragging the TrustedServer-appliance from the list to the previously created "ConfidentialVPN"-entry in the "VPNs & Internet Groups" opens a window like in Figure 6.15. Checking "This appliance connects to all other appliances" is mandatory. Furthermore the "Policy" for the "Share"-networks have to be choosen, which is "IP forwarding (bidirectional)" in this case. Clicking "OK" closes the dialog and the setup is complete.

The procedure described in this section has to be repeated for the "Public"-TVD and the containing VPNs, networks and compartments.

Figure 6.15: Editing VPN membership of TrustedServer

The compartments can now be started, stopped or removed remotely by clicking the appropriate button in the "Compartments"-tab of an appliance (see Figure 6.13).

Chapter 7

Cloud-of-Clouds Prototype

7.1 BFT-SMaRt

7.1.1 Download instructions

The BFT-SMaRt source repository is in http://code.google.com/p/bft-smart. In the repository there is the latest stable and work versions of BFT-SMaRt, wiki and API documention.

7.1.2 How to install

BFT-SMaRt is a middleware to perform state machine replication. Altough it is not an application, the source code provides a demo package to be used as example on how to extend and use BFT-SMaRt.

Environment requirements for BFT-SMaRt:

- 3f+1 computers, where f is the number of faults that can be tolerated
- Java Runtime Environment 6 or later

To extend and use BFT-SMaRt has to implement two interfaces, one in the replica side and another for the client side. The replica interface will define how the application data is replicated. The client side will define how to send data to servers. Instructions on how to extend client and server interfaces are provided in the BFT-SMaRt web page wiki. To run existing demo packages provided with BFT-SMaRt, the user has to:

- Download the code from the BFT-SMaRt repository
- Extract the code and copy to the desired location on each server
- Edit the /config/hosts.config and set the ip and port number of each server
- Start BFT-SMaRt server on each replica by calling the executable /runscripts/smartrun.sh or smartrun.bat, depending on the operating system.

The instructions above are also listed in the instructions file README.txt in the root of BFT-SMaRt source code.

7.2 Resilient Object Storage (DepSky)

7.2.1 Prototype Execution Instructions

DepSky is a library to store data in multiple clouds and ensure confidentiality of the data. It does not require software to be installed in the clouds. It is necessary to configure DepSky to access the user accounts in the different clouds.

7.2.2 DepSky configuration

The current version of DepSky has driver and instructions implemented for different servers in the Amazon cloud. The AwsCredentials.properties configuration file has to be modified with the user account information. Instalation instructions are also available in the file README.txt.

7.2.3 Running DepSky locally

To run and test DepSky locally, there is an additional package to simulate a cloud locally. It is the package ServerClouds. To start the server locally the ServerThread class has to be ran by the command java ServerThread. After the local cloud is started, in another terminal window a DepSky client can be started by the shell script DepSky_Run.sh. The command line to start the application is: ./DepSky_Run.sh <container_name> <client_id> <DepSky mode>. The container name can be any name defined by the user. The client id can be any client that has keys in the configuration folder config. DepSky mode can be 0 for DepSky-A, 1 for DepSky-CA, 2 to use only erasure codes and 3 to use only secret sharing.

Chapter 8

Other Prototypes

8.1 Security Assurance of Virtualized Environments *(SAVE)*

8.1.1 Operating Environment Setup

In order to enable SAVE to discover OpenStack, OpenStack needs to be modified.

Hardware requirements:

- Hardware virtualization (VT-x) capable physical machine

Software requirements:

- OpenStack's cactus release patched with the TClouds API patch

After patching OpenStack, once it's up and running, it can export data to SAVE. Ideally more than one virtual machine should be deployed, but the discovery works with even one.

8.1.2 Prototype Build and Installation Instructions

- Unzip the cactus release of OpenStack
- patch -p1 < openstack-save.patch
- start up OpenStack as normally
- start some virtual machines

8.1.3 Prototype Execution Instructions

SAVE relies on using probes to communicate with target systems. Such a probe must be configured. An example configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<DiscoverConfig>
  <Host hostname="openstack1" address="127.0.0.1" enabled="true">
    <Credential type="API" username="openstack-admin" password="secretpassword" port="8775"/>
  </Host>
</DiscoverConfig>
```

Upon starting up SAVE, one can select the project to execute. This should be a directory containing the above XML in a "discovery.xml" file. Once that is loaded, executing the probe and showing the discovery result in a visual way can be done from the GUI.

8.2 Ontology-based reasoner: Libvirt With Trusted Virtual Domains

8.2.1 Operating Environment Setup

In order to use Libvirt with Trusted Virtual Domains, it is necessary to install the Open vSwitch software. It can be downloaded from http://openvswitch.org/download and can be installed by following the installation instructions in the Documentation section (http://openvswitch.org/support).

Other software dependencies are automatically installed by the package manager of the Fedora 16 distribution (YUM).

8.2.2 Prototype Build and Installation Instructions

Libvirt with Trusted Virtual Domains can be installed by executing the command:

```
# yum install <libvirt pkgs>
```

where `<libvirt pkgs>` must be replaced with packages contained in the `Ontology-Libvirt-TVD` directory of the tarball for the D2.4.2 deliverable.

8.2.3 Prototype Execution Instructions

In order to configure Libvirt to create a Trusted Virtual Domain, it is necessary to create XML files for the virtual network and virtual machines as described in Section 3.4.2.3.

Part III

Appendices

Appendix A

TClouds Infrastructure Wiki

This section contains the instructions to use the TCLOUDS infrastructure for OpenStack patches.

A.1 Infrastructure Overview

Our infrastructure consists of three sites:

- `https://git.tclouds-project.eu`: code repositories
- `https://review.tclouds-project.eu`: code review with gerrit
- `https://jenkins.tclouds-project.eu`: testing framework for automatic tests

A.1.1 Code Repositories (git.tclouds-project.eu)

The code repositories that use GIT as version control system are managed by git.tclouds-project.eu. Actually, this server contains the following repositories:

- openstack/nova.git (the Nova module of OpenStack)
- openstack/openstack-ci-puppet (scripts for automating the installation of the infrastructure)

Each partner can create his own repositories by using Gerrit (below there are the instructions to create a new repository). New repositories should follow the naming convention:

- **<PARTNER_ID>/<Project_Name>**

A.1.1.1 Layout of the Nova repository (openstack/nova.git)

Actually there are the following branches (created by OpenStack developpers):

- *master* (development branch);
- *milestone-proposed* (branch that contains patches to be released as part of the next milestone);
- *stable/diablo* (OpenStack *diablo* release + fixes);
- *stable/essex* (OpenStack *essex* release + fixes).

and the following branch (created by TClouds):

- *tclouds* (this branch will contain all the patches developed by partners);

Partners can create temporary branches for development purposes using the following naming convention:

- **<PARTNER_ID>-<Branch_Name>**

A.1.1.2 Authentication

Steps required to create a new set of credentials are described in the *Code Review* section.

A.1.2 Code Review (review.tclouds-project.eu)

Gerrit is intended to provide a light weight framework for reviewing every commit before it is accepted into the code base. Changes are uploaded to Gerrit but dont actually become a part of the project until theyve been reviewed and accepted.

A good description of this software and the steps that should be performed in order to submit code can be found at http://wiki.openstack.org/GerritWorkflow.

A.1.2.1 Authentication

In order to distinguish between different users, each partner (it is possible to create an account for each member) should register a new set of credentials in Apache. In the following, there are the steps to setup Apache:

1 Log in the *jenkins.tclouds-project.eu:1027* server through SSH as *tcloudsuser* (the password was decided in the Darmstadt integration meeting);

```
$ ssh -p29418 <your_user_id>@jenkins.tclouds-project.eu gerrit create-project
--name <PARTNER_ID>/<Project_Name>
```

2 Set the new user credentials by executing the command:

```
$ ssh -p29418 mrossi@jenkins.tclouds-project.eu gerrit create-project --name
POL/LogService
```

3 Access the Code Review site by giving the newly set of credentials;

4 Click on the *Sign Out* link on the right upper corner to refresh credentials;

5 Enter the full name, register an email and a SSH public key (required to submit code changes with git review);

6 Click the *Continue* link at the bottom of the page;

7 Close the browser, reopen it and go to the Code Review site;

8 Access the Code Review site by giving the credentials of *tcloudsuser*;

9 Click on the *Sign Out* link on the right upper corner to refresh credentials;

10 Go to Admin -> Groups -> Project Bootstrappers;

11 In the *Members* field enter your full name and click the *Add* button;

12 Close the browser, reopen it and go to the Code Review site;

13 Access the Code Review site by giving your credentials;

14 Click on the *Sign Out* link on the right upper corner to refresh credentials;

A.1.2.2 Creation of New Code Repositories

Each partner can create a new code repository by executing the following command:

```
$ ssh -p29418 <your_user_id>@jenkins.tclouds-project.eu gerrit create-project
--name <PARTNER_ID>/<Project_Name>
```

example

```
$ ssh -p29418 mrossi@jenkins.tclouds-project.eu gerrit create-project --name
POL/LogService
```

Then, in the working copy directory, the partner should first create and commit the file *.gitreview* which content is:

```
[gerrit] host=review.tclouds-project.eu port=29418
project=<PARTNER_ID>/<Project_Name>
```

and execute the command:

```
$ git review -s
```

Finally, the partner should push the master branch to *gerrit*:

```
$ git push gerrit HEAD:refs/heads/master
```

A.1.2.3 Creation of Additional Branches in Existent Code Repositories

Additional branches can also be created through the web UI, assuming at least one commit already exists in the project repository. A project owner can create additional branches under Admin > Projects > Branches. Enter the new branch name, and the starting Git revision.

A.1.2.4 Creating a New Patch Set and Submitting It for Review

1 Configure git:

```
$ git config --global user.name "Your Name"
```

```
$ git config --global user.email "your@email.com"
```

2 Clone the remote repository (e.g. Nova):

```
$ git clone https://git.tclouds-project.eu/openstack/nova.git
```

3 Create a new branch (assign it a names that summarize the goal of the patch set):

```
$ git checkout -b <new branch name> origin/tclouds
```

4 Modify the code;

5 Save the changes:

```
$ git commit -a -s -m"Your Comment"
```

6 Submit the patch set to Gerrit:

```
$ git review tclouds
```

NOTE: alternatively you can work on a different branch than ”tclouds”. For instance, you can submit the patches (for testing purposes) on a branch that you created before. More details are below in the section *OpenStack Patches Submission Workflow*.

A.1.2.5 Review of Patches

With Gerrit it is possible to review submitted patches before they are merged in the code repository. The submission process relies on a voting system to determine when a patch will be effectively merged and requires that three conditions are satisfied:

1 the patch must receive a +2 vote for the *Code-Review* requirement;

2 the patch must receive a +1 vote for the *Approved* requirement;

3 the patch must receive a +2 vote for the *Verified* requirement.

All partners can partecipate in the review process and are allowed to merge code. However, while they can directly do the merge operation, it is useful to wait that Jenkins completes all tests and does the merge by itself (this will happen after submitted patches receive a +2 vote for the *Code-Review* and +1 for the *Approved* requirements).

A.1.2.6 OpenStack Patches Submission Workflow

In this section, it is described the workflow that each partner should follow in order to submit patches for OpenStack.

Each partner can submit patches in two ways:

1 if a partner wants to submit minor changes that do no need to be tested separately, he can submit the patch to Gerrit by executing the command:

```
$ git review tclouds
```

2 if a partner wants to develop his patches in a separate branch, he should:

(a) follow the instructions contained in the subsection *Creation of Additional Branches in Existent Code Repositories*

(b) submit the patch to Gerrit by executing the command:

```
$ git review <partner branch>
```

(c) when a patch is ready, submit the patches to Gerrit by referring to the *tclouds* branch:

```
$ git review tclouds
```

A.1.3 Testing Framework (jenkins.tclouds-project.eu)

The Jenkins framework will be used to test OpenStack patches and the subsystems developed by partners.

A.1.3.1 Authentication

In order to access the Jenkins Web site, it is sufficient to supply the same credentials provided for the Code Review site.

A.1.3.2 Testing OpenStack

This operation will be done automatically each time a new patch is submitted to Gerrit and before the code is merged in the repository. This site behaves like the OpenStack Jenkins site, which is used by OpenStack community to perform automatic tests on this software. While actually only the Nova module has been configured, it is possible to test also other OpenStack modules like Glance or Swift.

A.1.3.3 Testing Partners' Subsystems

The report R2.4.5.2 contains the workflow that partners should follow in order to perform automatic and manual tests and the steps required to configure partners subsystems in Jenkins.

Appendix B

Subsystems' code availability

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

TClouds subsystem	Code availability
Resource-efficient BFT (CheapBFT)	Source code in D2.4.2 tarball (**)
Simple Key/Value Store (memcached)	It will delivered in Year 3
Secure Block Storage (SBS) (*)	Object code in D2.4.2 tarball (**)
Secure VM Instances (*)	Object code in D2.4.2 tarball (**)
TrustedServer	Confidential
Log Service	Source code in D2.4.2 tarball (**)
State Machine Replication (BFT-SMaRt)	Source code delivered as D2.2.3, available at http://code.google.com/p/bft-smart
Fault-tolerant Workflow Execution	It will delivered in Year 3
Resilient Object Storage (DepSky)	Source code in D2.4.2 tarball (**)
Confidentiality Proxy for S3	It will delivered in Year 3
Access Control as a Service (ACaaS)	Source code in D2.4.2 tarball (**)
TrustedObjects Manager (TOM)	Confidential
Trusted Management Channel	Confidential
Ontology-based Reasoner (libvirt)	Source code in D2.4.2 tarball (**)
Automated Validation (SAVE)	Confidential
Remote Attestation Service **[New Y2]**	Source code in D2.4.2 tarball (**)

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

(**) D2.4.2 tarball also includes binary packages for Ubuntu 12.04 LTS and Fedora 16 Linux distributions generated for 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 Sections 4.5 and A.1.3). For details and installation instructions see the README files included in the root folder of the tarball.

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

Appendix C

Trustworthy OpenStack Dashboard screenshots

The standard OpenStack Dashboard (i.e., the Graphical User Interface for cloud management) has been enhanced to set up some of the Security Extensions forming Trustworthy OpenStack (Secure Logging, Advanced VM Scheduling, and Cloud Nodes Verification/Remote Attestation). In the following, some sample screenshots (without the whole workflow) will be shown to give an overall view of the enhancements applied to the standard Dashboard.

Figure C.1 shows the login page of TClouds Trustworthy OpenStack Dashboard. Figure C.2 shows the page to define the cloud-wide Requirements and the Security Properties of the cloud nodes (for the ACaaS subsystem, see Section 3.1.2.2). Figure C.3 shows the page to set the TClouds newly added requirements for an instance type (for the RemoteAttestation and ACaaS subsystems, see Sections 3.1.2.1 and 3.1.2.2). Figure C.4 shows the page to start an instance and to directly set up TClouds newly added requirements for the instance being started (for the RemoteAttestation and ACaaS subsystems, see Sections3.1.2.1 and 3.1.2.2). Figure C.5 shows the list of opened secure logging sessions that can be selected to be verified for integrity and shown (for the LogService subsystem, see Section 3.1.2.4). Finally Figure C.6 shows a successfully verified secure logging session, with all log entries (for the LogService subsystem, see Section 3.1.2.4).

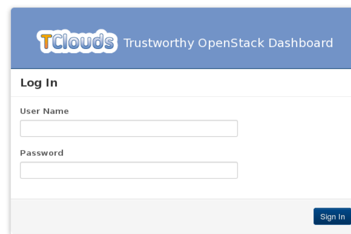

Figure C.1: The Trustworthy OpenStack Dashboard - Login

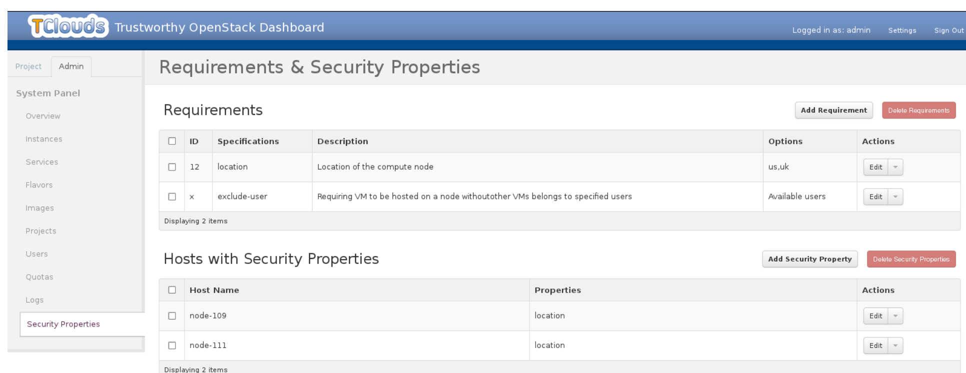

Figure C.2: Trustworthy OpenStack Dashboard - (ACaaS) Requirements and Security Properties

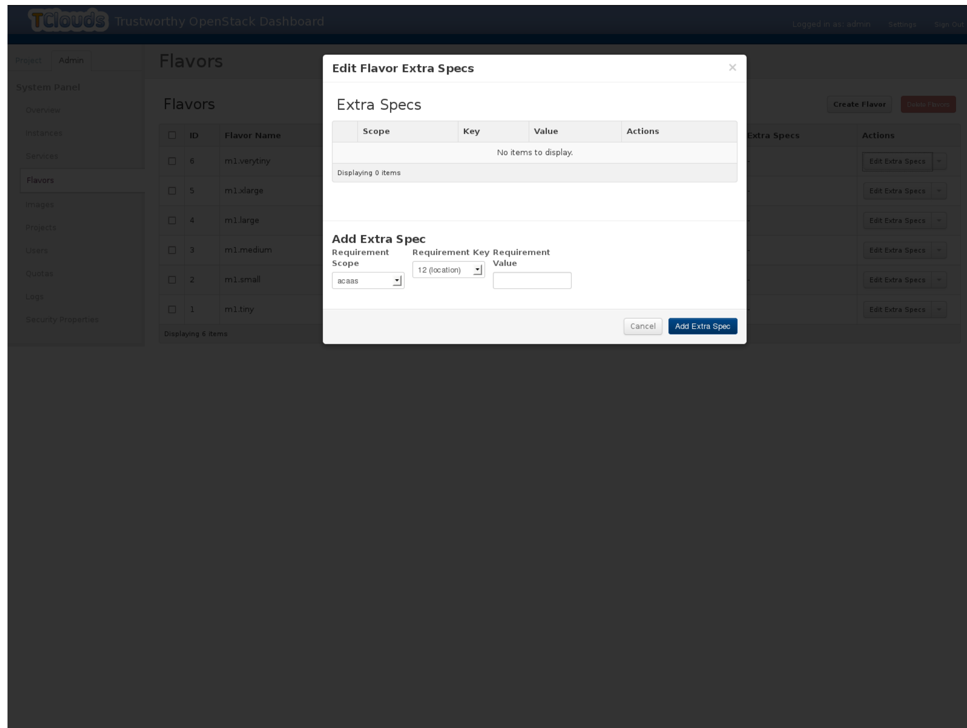

Figure C.3: Trustworthy OpenStack Dashboard - (Remote Attestation/ACaaS) Setting Extra Specs with flavours

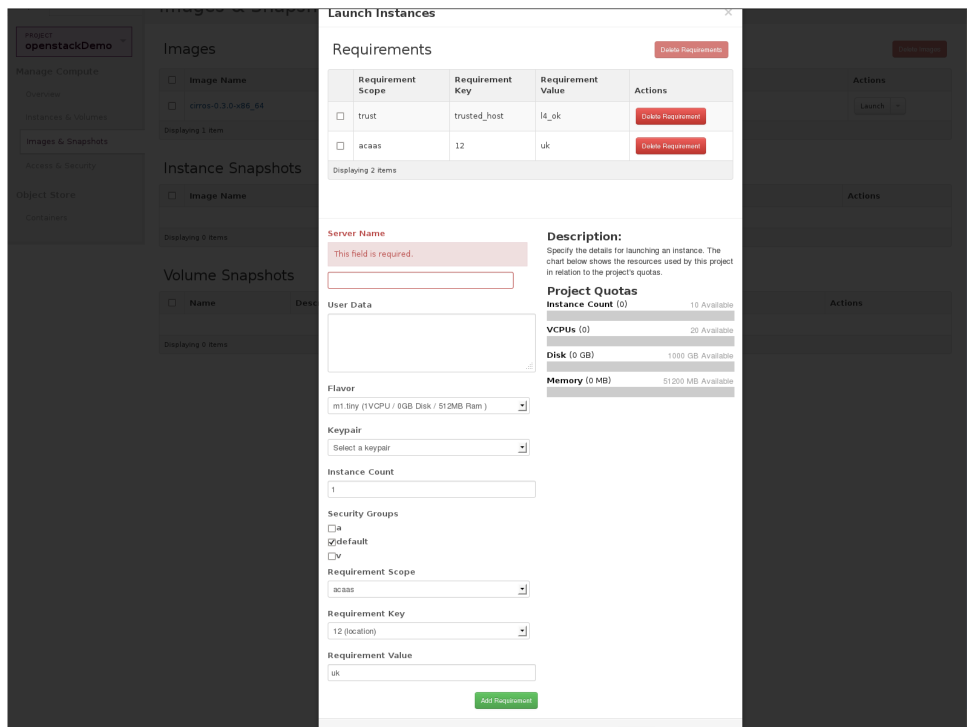

Figure C.4: Trustworthy OpenStack Dashboard - (Remote Attestation/ACaaS) Launching an instance and setting the requirements

Figure C.5: Trustworthy OpenStack Dashboard - (LogService) List of available logging sessions

Figure C.6: Trustworthy OpenStack Dashboard - (LogService) Log file dump with verification results

Bibliography

[AN12] Marco Abitabile and Marco Nalin. D3.3.3 - Validation Protocol and Schedule for the Smart Power Grid and Home Health Use Cases. Technical report, FSR, September 2012. TClouds deliverable.

[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.

[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.

[ea11a] Alysson Bessani et al. D2.2.3 Proof-of-concept of Middleware for Adaptive Resilience. Technical report, FFCUL et al., September 2011. TClouds deliverable.

[ea11b] Christian Cachin et al. D2.3.1 - Requirements, Analysis, and Design of Security Management. Technical report, IBM et al., October 2011. TClouds deliverable.

[ea11c] Emanuele Cesena et al. D2.4.1 - Clouds Prototype Architecture, Quality Assurance Guidelines, Test Methodology and Draft API. Technical report, Politecnico di Torino et al., September 2011. TClouds deliverable.

[ea11d] Marcelo Pasin et al. D2.2.1 - Preliminary Architecture of Middleware for Adaptive Resilience. Technical report, FFCUL et al., October 2011. TClouds deliverable.

[ea12a] Alysson Bessani et al. D2.2.2 - Preliminary Specification of Services and Protocols of Middleware for Adaptive Resilience. Technical report, FFCUL et al., September 2012. TClouds deliverable.

[ea12b] Norbert Schirmer et al. D2.1.2 - Preliminary Description of Mechanisms and Components for Single Trusted Clouds. Technical report, SRX et al., September 2012. TClouds deliverable.

[ea12c] Sor̈en Bleikertz et al. D2.3.2 - Components and Architecture of Security Configuration and Privacy Management. Technical report, IBM et al., September 2012. TClouds deliverable.

[Gel85] David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[GHSS11] Ruediger Glott, Elmar Husmann, Matthias Schunter, and Ahmad-Reza Sadeghi. D1.1.1 - Draft Scenario and Requirements Report. Technical report, UMM et al., April 2011. TClouds deliverable.

[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 Levart. FUSE-J: A Java binding for FUSE. Retrieved from: http://sourceforge.net/projects/fuse-j/, 2012.

[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/cmockery/, September 2012.

[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.

[SK99] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2:159–176, May 1999.

[SV12] Paulo Santos and Paulo Viegas. D3.2.3 - Smart Lighting System Draft Prototype. Technical report, EFACEC ENG, September 2012. TClouds deliverable.