



DELIVERABLE D4.1

VALIDATION AND DEMONSTRATION SCENARIOS

Grant Agreement number:	786922
Project acronym:	ASTRID
Project title:	AddreSsing ThReats for virtualIseD services
Start date of the project:	01/05/2018
Duration of the project:	36 months
Type of Action:	Research & Innovation Action (RIA)
Project Coordinator:	Name: Orazio Toscano Phone: +39 010 600 2223 E-mail: orazio.toscano@ericsson.com
Due Date of Delivery:	M12 (30/04/2019)
Actual Date of Delivery:	14/06/2019
Work Package:	WP4 – Integration, Demonstration and Validation
Type of the Deliverable:	R
Dissemination level:	PU
Editors:	TUB
Version:	1.0

**List of Authors**

ETI	ERICSSON TELECOMUNICAZIONI
-----	----------------------------

Orazio Toscano

SURREY	UNIVERSITY OF SURREY
--------	----------------------

Mark Manulis

CNIT	CONSORZIO NAZIONALE INTERUNIVERSITARIO PER LE TELECOMUNICAZIONI
------	---

Matteo Repetto, Alessandro Carrega

INFO	INFOCOM S.R.L.
------	----------------

Maurizio Giribaldi, Guerino Lamanna

POLITO	POLITECNICO DI TORINO
--------	-----------------------

Fulvio Risso

TUB	TECHNISCHE UNIVERSITAET BERLIN
-----	--------------------------------

Tran Quang Thanh, Stefan Covaci

AGE	AGENTSCAPE AG
-----	---------------

Tomasz Poslada, Benjamin Ertl

UBITECH	GIOUMPITEK MELETI SCHEDIASMOS YLOPOIISI KAI POLISI ERGON PLIROFORIKIS ETAIREIA PERIORISMENIS EFTHYNIS
---------	--

Anastasios Zafeiropoulos, Thanos Xirofotos

DTU	DANMARKS TEKNISKE UNIVERSITET
-----	-------------------------------

Thanassis Giannetsos



Disclaimer

The information, documentation and figures available in this deliverable are written by the ASTRID Consortium partners under EC co-financing (Call: H2020-DS-SC7-2017, Project ID: 786922) and do not necessarily reflect the view of the European Commission.

The information in this document is provided “as is,” and no guarantee or warranty is given that the information is fit for any particular purpose. The reader uses the information at his/her sole risk and liability.

Copyright

Copyright © 2019 the ASTRID Consortium. All rights reserved.

The ASTRID Consortium consists of:

ERICSSON TELECOMUNICAZIONI (ETI)

UNIVERSITY OF SURREY (SURREY)

CONSORZIO NAZIONALE INTERUNIVERSITARIO PER LE TELECOMUNICAZIONI (CNIT)

INFOCOM S.R.L. (INFO)

POLITECNICO DI TORINO (POLITO)

TECHNISCHE UNIVERSITAET BERLIN (TUB)

AGENTSCLAPE AG (AGE)

GIOUMPI TEK MELETI SCHEDIASMOS YLOPOIISI KAI POLISI ERGON PLIROFORIKIS ETAIREIA PERIORISMENIS EFTHYNIS (UBITECH)

DANMARKS TEKNISKE UNIVERSITET (DTU)

This document may not be copied, reproduced or modified in whole or in part for any purpose without written permission from the ASTRID Consortium. In addition to such written permission to copy, reproduce or modify this document in whole or part, an acknowledgment of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.



Table of Contents

DISCLAIMER	3
COPYRIGHT	3
TABLE OF CONTENTS	4
LIST OF FIGURES	5
LIST OF TABLES	6
LIST OF ACRONYMS	7
1 EXECUTIVE SUMMARY	8
2 INTRODUCTION	9
2.1 PURPOSE AND SCOPE.....	9
2.2 ORGANIZATION.....	9
2.3 RELATION TO OTHER WPs.....	10
3 USE CASES ARCHITECTURE	11
3.1 INTEGRATION WITH ASTRID SECURITY FRAMEWORK.....	11
3.2 MAESTRO ORCHESTRATOR	14
3.3 OPENBATON ORCHESTRATOR.....	18
3.4 KUBERNETES	21
3.5 VIRTUALIZATION INFRASTRUCTURES AND MANAGEMENT SOFTWARE - OPENSTACK.....	25
3.6 BENCHMARKING TOOLS – KALI LINUX.....	26
4 USE CASES DESCRIPTION	28
4.1 SECURITY ORCHESTRATION IN CLOUD ENVIRONMENT.....	28
4.1.1 <i>Overview</i>	28
4.1.2 <i>Service Topology</i>	28
4.1.3 <i>Specific Threats</i>	31
4.1.4 <i>Implementation Plan</i>	32
4.1.5 <i>Deployment Infrastructure</i>	35
4.2 SECURITY ORCHESTRATION IN NFV ENVIRONMENT	36
4.2.1 <i>Overview</i>	36
4.2.2 <i>Service Topology</i>	36
4.2.3 <i>Specific Threats</i>	37
4.2.4 <i>Implementation Plan</i>	38
4.2.5 <i>Deployment Infrastructure</i>	39
4.3 RELATION TO THE ASTRID APPLICATION SCENARIOS.....	40
5 DEMONSTRATION AND VALIDATION	41
5.1 AUTOMATIC FIREWALLING	41
5.2 (DISTRIBUTED) DENIAL OF SERVICE.....	42
5.3 MALWARE AND INTRUSIONS.....	43
5.4 ILLEGAL ACTIVITIES	45
5.5 FORENSICS.....	46
REFERENCES	48



List of Figures

Figure 1. ASTRID Framework Architecture	12
Figure 2. Architecture for ASTRID Use Cases	13
Figure 3. MAESTRO Architecture	15
Figure 4. MAESTRO Dashboard.....	17
Figure 5. MAESTRO Application Graph Instance	17
Figure 6. OpenBaton High-level Architecture	18
Figure 7. OpenBaton Components and GitHub Project Names	20
Figure 8. OpenBaton Dashboard.....	21
Figure 9. High-level Overview of a Kubernetes Cluster	22
Figure 10. OpenStack Map.....	26
Figure 11. Kali Linux Tools Listing.....	27
Figure 12. Metasploit Framework.....	27
Figure 13. Use Case #1 Service Graph	29
Figure 14. Use Case #1 within ASTRID Framework	34
Figure 15. Use Case #2 Service Graph	36
Figure 16. Use Case #2 OpenStack Infrastructure.....	39
Figure 17. Use Case #2 within ASTRID Framework	40



List of Tables

Table 1. Summary of Main Threats for the Use Case #1	32
Table 2. Implementation Plans for Use Case #1	34
Table 3. Preliminary Deployment Plans Use Case #1	35
Table 4. Summary of Main Threats for the Use Case #2	38
Table 5. Implementation Plans for Use Case #2	39
Table 6. Mapping of Application Scenarios to Use Cases	40



List of Acronyms

Acronym	Definition
ABAC	Attribute-based Access Control
AES	Advanced Encryption Standard
API	Application Programming Interface
(D)DoS	(Distributed) Denial of Service
ECC	Elliptic-Curve Cryptography
ELK	Elasticsearch Logstash Kibana
eBPF	extended Berkeley Packet Filter
HMAC	Hash-based Message Authentication Code
IAM	Identity Access Management
IoT	Internet of Thing
LTE	Long Term Evolution
LoRaWAN	Long Range Wide Area Network
NBI	North Bound Interface
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
SeVoC	Secure VoIP Communication
SRTP	Secure Real-time Transport Protocol
SSL	Secure Socket Layer
TLS	Transport Layer Security
(v)EPC	(virtual) Evolved Packet Core
VNF	Virtual Network Function
VNFM	Virtual Network Function Manager
VPN	Virtual Private Network



1 Executive Summary

This document describes the ASTRID Use Cases, as elaborated in Task 4.1. The description extends the preliminary concepts outlined at the proposal stage and includes more concrete plans for demonstration and validation.

The two Use Cases are designed to demonstrate the feasibility of the ASTRID framework in two different domains, namely cloud applications, and network function virtualization; collectively, they cover all application scenarios identified in D1.1. The description explains how the Use Cases relate to the ASTRID architecture reported in D1.2, and which orchestration tools will be used to manage deployment and life-cycle operations in an automated way in the different domains. Based on the application design and the inventory of available software and tools, an implementation roadmap is depicted, and specific responsibilities are assigned to partners that will be involved in each Use Case. This will provide input to T4.2/4.3 for the implementation of the two Use Cases.

The methodology for demonstration and validation is based on the selection of relevant attacks and threats for each Use Case, and the identification of a set of demonstration and validation scenarios. Each scenario is matched with key performance indexes and acceptance criteria, which have been selected and set according to the expected impact listed in the proposal. This description will drive the work in T4.5/4.6, which will set up the demonstration environments, and T4.7, which will carry out performance measurements.



2 Introduction

2.1 Purpose and Scope

The main purpose of this deliverable is to describe the two ASTRID Use Cases that will be used for demonstration and validation. Each Use Case is built around a virtual application or service, which is designed to be automatically deployed and managed over a virtualization infrastructure. The ASTRID framework is explicitly conceived to remain agnostic of the different development and orchestration models, so the Use Cases are conceived to demonstrate its applicability to two different domains, i.e., cloud applications and network function virtualization. The implementation of each Use Case consists of the following main activities:

- design and development of the application/service;
- enrichment of the application/service design with ASTRID elements (see D1.2);
- development of domain-specific extensions for the ASTRID run-time (i.e., APIs to software orchestrators, see D1.2);
- setup of the experimental environments;
- functional validation and performance measurements.

This document describes the overall architecture of the two Use Cases and the methodology for demonstration and validation. The architecture shows how the ASTRID run-time environment is integrated with orchestration tools and multi-site applications. Demonstration and validation are based on a set of scenarios, which takes into account relevant features of the framework, in terms of capability to detect/react to attacks or to carry out investigation. Target KPIs and acceptance criteria have been set according to the expected impact, so to clearly assess if and how much the project has achieved its objectives. It is worth noting that, according to the project's objectives, the ASTRID framework is conceived to support more automation in the management of security aspects and better integration with emerging virtualization and orchestration technologies. In this respect, key performance aspects are mostly focused on the capability and speed to adapt to the evolving context rather than on the effectiveness of the detection algorithms (e.g., true positives, false negatives).

2.2 Organization

One of the main traits of ASTRID is more automation in management of security, through tight integration with existing orchestration frameworks. For this reason, Section 3 describes the overall architecture of the two Use Cases, by elaborating how the ASTRID run-time framework interacts with existing software orchestration tools. For the sake of completeness, a quick overview is included about related tools from the ASTRID partners and the open source community, including software orchestration and infrastructure virtualization. The overview covers all technologies that are currently used by partners for preliminary experimentation and demonstration (i.e., year-1 demos). A preliminary indication of benchmarking tools is also given at the end of this Section, which will be refined later on during the actual implementation of the Use Cases.

Section 4 describes in detail the applications that will be used for each Use Case, including required software components and service graphs. The description also includes a development plan, which covers the identification of missing components and modules, as well as the infrastructures that will be used for deployment and testing. The plan assigns the responsibilities for development to specific partners. Finally, a list of relevant threats and attacks is discussed for each Use Cases, which should summarize the main security expectations; this is intended to maximize the effectiveness of the demonstration and validation effort within each application domain.



Section 5 describes a set of demonstration and validation scenarios, which will be used to evaluate the feasibility, functional correctness and performance of the ASTRID framework. These scenarios mainly address situational awareness but also cover mitigation and reaction cases. They are defined in terms of security features and main objectives, description of the expected trials and experiment conditions, target KPIs, and acceptance criteria. While the target KPIs denotes the performance level which are expected to support a very compelling exploitation strategy, the acceptance criterium is usually a looser threshold that gives the minimum value to achieve the project objectives and to motivate further investigation/development.

It is worth mentioning that target KPIs and acceptance criteria largely depends on the complexity of the applications, so they cannot be considered as general figures that apply to all scenarios. They have been set according to the expected size of the applications at design time, so to appear compelling when compared to existing orchestration performance. They might be adjusted later on during the progress of the project, in order to address unexpected complexity of some components or any design or implementation modifications; any possible change will be properly motivated and will demonstrate the persistent improvement over other reference implementations.

2.3 Relation to other WPs

This deliverable describes the results of Task 4.1 “Definition of Validation and Demonstration Scenarios.” The definition of the ASTRID Use Cases is based on *i)* the preliminary concepts described in the proposal; *ii)* the application scenarios identified in D1.1 [1]; *iii)* the initial specification of the ASTRID architecture in D1.2 [2]. The work described in this document feeds the following activities in WP4, as described below:

- the description of the cloud application and NFV service will be used for their development in Task 4.2 and 4.3, respectively;
- the selected orchestrators will indicate which orchestration APIs are included in the ASTRID software framework (Task 4.4);
- the overall description of each Use Case (including the deployment infrastructure) and the demonstration and validation scenarios will be used to:
 - identify the storyboard for functional validation that will be used for producing videos, whitepapers, and real-time demos in Task 4.5 and 4.6;
 - carry out performance measurement and analysis in Task 4.7.



3 Use Cases Architecture

The ASTRID Use Cases are conceived to support situational awareness for specific applications that are already part of the business plans of one or more partners. Therefore, the main purpose is to *enrich* their current implementations (or their preliminary design, in case the implementation is not available yet) without affecting the main structure and without requiring large modifications. As a matter of fact, modifications to existing applications will be limited to security features that should have been integrated in any case (e.g., for compliance with the evolving regulations) or extended logging for improving the security base for detection and analysis.

According to the project's concept and objectives listed in D1.1 [1], the ASTRID framework will support more automation in the process of building situational awareness, leveraging *DevOps* paradigms and software orchestration technologies. The ambition is to avoid as much as possible human intervention in the deployment and configuration processes, in order to remove the possibility of common mistakes and carelessness, while giving more control to security experts in the design phase.

As thoroughly described in D1.2 [1], the ASTRID framework is applied in two phases of the service life-cycle: pre-deployment (which covers the necessary enhancement at design time to drive the automation at run-time) and run-time (which implements the real monitoring, detection, and reaction processes). The ASTRID Use Cases are expected to demonstrate the full workflow, including both the design and run-time phases. This will allow having direct indications about both technical performance and usability, which are key elements to support an effective communication and exploitation strategy beyond the project lifetime.

Technically speaking, the ASTRID run-time framework is partially co-located within the application/service to be protected (i.e., local monitoring and inspection hooks), whereas detection and smart reaction capabilities are kept out of the main service-business-logic and can be shared by multiple applications and even tenants. Section 3.1 describes the Use Cases from a technical perspective, i.e., how the ASTRID architecture is integrated with the application/service orchestration. The following Sections provide a quick overview of the software orchestration technologies that will be used in the Use Cases and a preliminary list of (benchmarking) tools to simulate the cyber-attacks for demonstration and validation purposes.

3.1 Integration with ASTRID Security Framework

Figure 1 shows the ASTRID architecture, as defined in D1.2¹ [2]; the purpose and internal components of the two ASTRID- subsystems (pre-deployment and run-time) have been thoroughly discussed in that document. Here, we briefly recall that the pre-deployment subsystem includes several design activities to embed security hooks in the service topology, while the run-time subsystem carries out all configurations and detection tasks on the actual service instance of the tenant. The Security Dashboard is the human interface to supervise, control, and manage all the processes involved in the ASTRID framework.

The implementation of the ASTRID framework (yellow frames in Figure 1) is covered by the activities in WP2 and WP3 (see Section 7.8 in D1.2). For the implementation of the Use Cases, it is necessary to give additional information about the greyed frames (right side of Figure 1), which concern the specific application/service and its orchestration. In particular, the implementation of the Use Cases is expected to provide the necessary API to the Orchestrator and monitoring and inspection hooks co-located with each VF.

¹ The picture is indeed an updated version, which already includes some additional outcomes from Task 1.5.

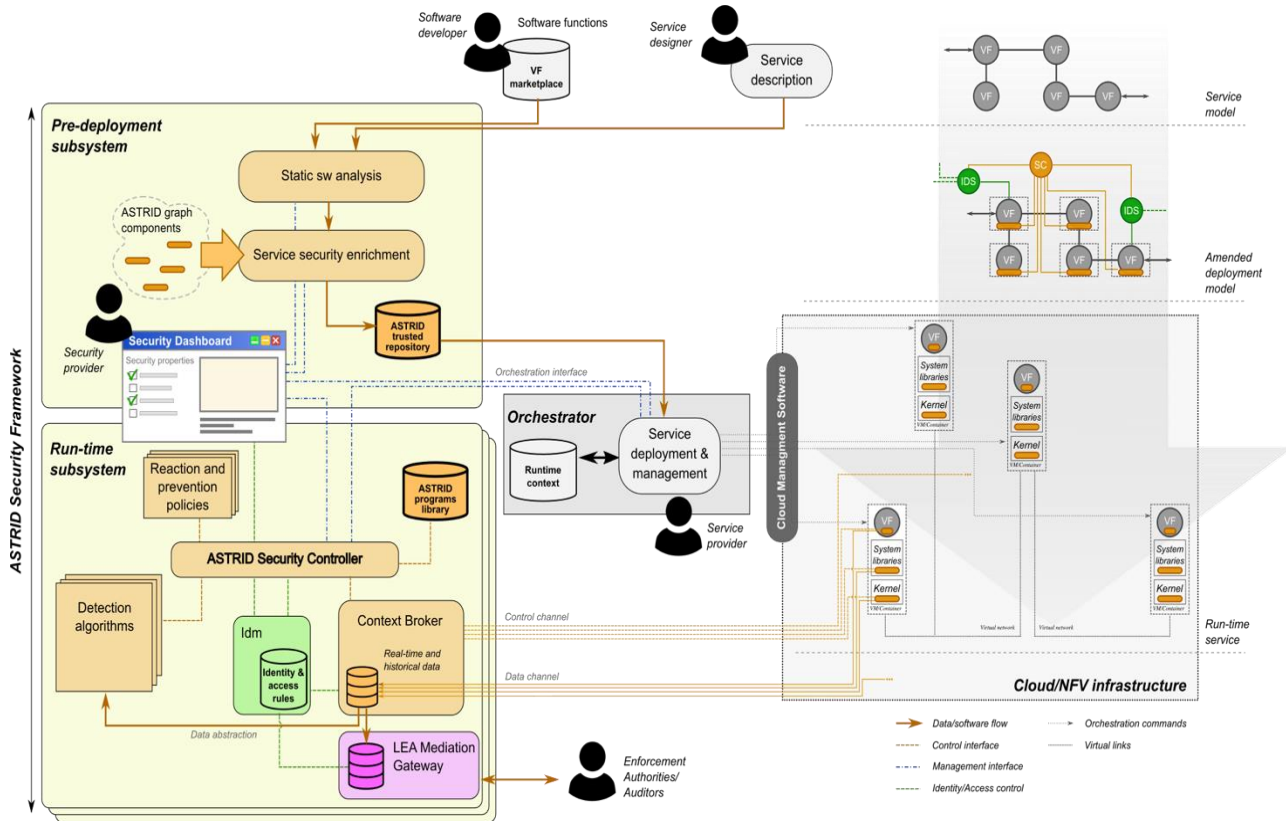


Figure 1. ASTRID Framework Architecture

Figure 2 shows the overall architecture for the ASTRID Use Cases. There are two Use Cases, covering “**Cloud applications**” and “**NFV services**”. In the first case, the reference infrastructure is the data centre, where Cloud Management Software (CMS) is used to manage the provisioning of virtual resources. In the second case, the infrastructure is provided by NFVI Point-of-Presence nodes, which can be in central offices of telco providers or at the network edge; according to the ETSI NFV-MANO framework, the management software for resource provisioning is called Virtual Infrastructure Manager (VIM). OpenStack can be used for both CMS and VIM; another very common options for the datacentres is the VMware vSphere suite. The usage of public cloud services (Amazon AWS, Google Cloud Service, Microsoft Azure) is yet another option. The CMS/VIM provision a full range of virtual resources- computing, networking, and storage being the most known services. The execution environment can be a virtual machine, a software container, or a combination of both. Virtual networking is usually based on internal segmentation made by hypervisors and VLAN/VxLAN/GRE tunnels between different servers. A common limitation of existing cloud solutions is the lack of interoperability between different installations, even when the same CMS is used. That means that a distributed application must implement its own mechanisms to provide Layer-2 connectivity between the components deployed in different virtualization infrastructures. Such mechanisms are often based on VPN technologies, which create overlays without strict control on QoS, confidentiality, and other management parameters. The usage of SDN controllers for wide-area networks (SD-WAN) is expected to create seamless end-to-end services in NFV infrastructures.

Service orchestrators, in general, deploy a virtual application services in two steps:

- a) provisioning of the necessary virtual resources to create the execution environments (i.e., VMs/containers, virtual networks, storage space), through CMS/VIM; this usually also includes booting the base OS through disk images;

- b) installation and configuration of the necessary software and libraries, through direct interaction with the base OS; this needs some management agents that must be already present in the base image (an SSH connection and an administrative account are enough to install and configure the software).

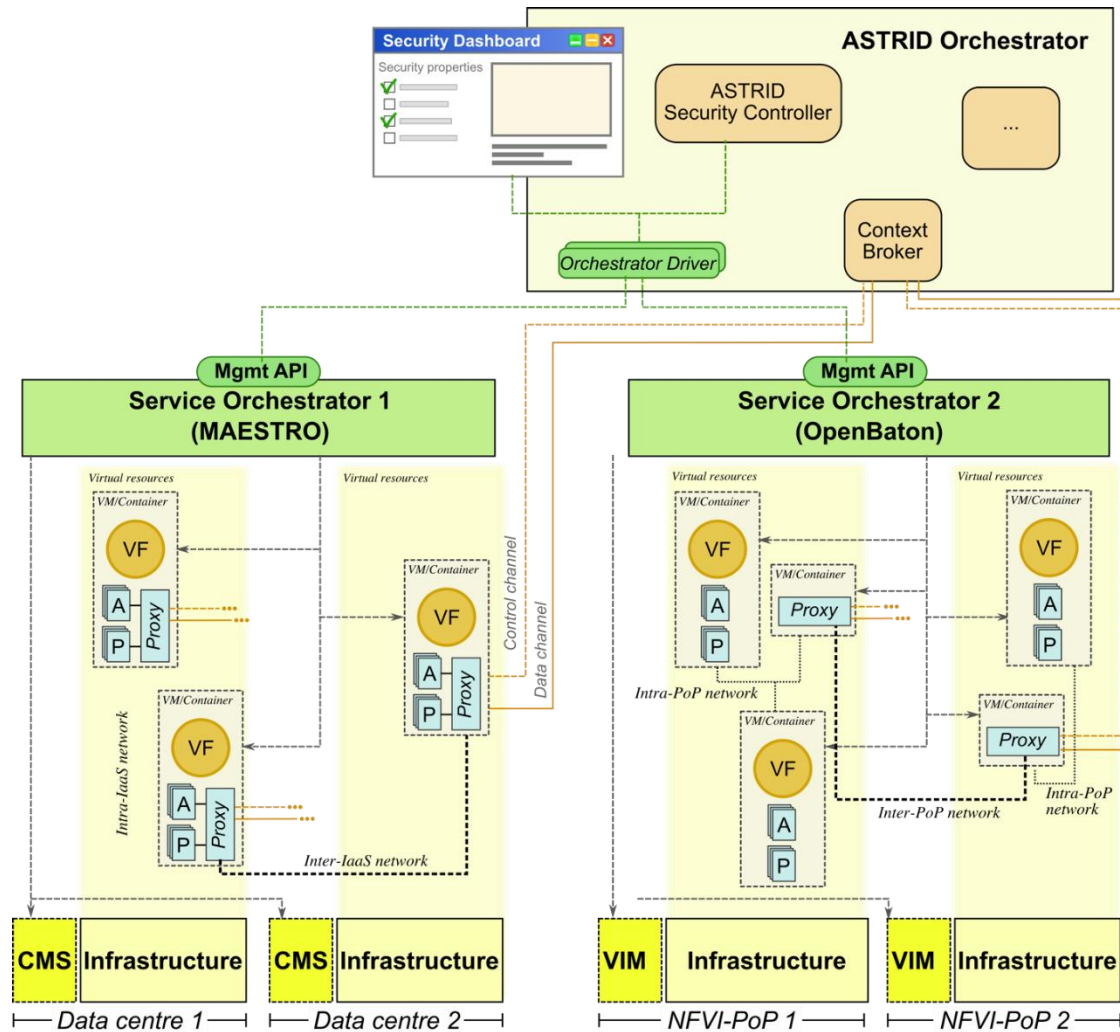


Figure 2. Architecture for ASTRID Use Cases

Service orchestrators are a relatively new technology, yet different products are already available from open-source communities (e.g., Juju, OpenBaton, Kubernetes, MAESTRO). Some of them already have their own APIs / NBIs for remote control and integration in higher-level management tools, whereas some only provide GUI/CLI interfaces for direct usage/control by humans. Remote APIs / NBIs are expected to be used to load a service graph, start/stop its deployment, modify the graph, trigger lifecycle events.



The implementation of the ASTRID Use Cases will integrate the ASTRID framework in existing orchestration frameworks, as graphically depicted in Figure 2. In the pre-deployment stage, the description of the service graph will be enriched to install the ASTRID components in the execution environment of each VF. These components will be both monitoring and inspection probes (indicated as “P” in Figure 2) and their control agents (“A”); the latter are used to change the configuration of the former, in order to make them *programmable*. Task 2.1 has already selected several existing components for probing and inspection (the ELK framework and the eBPF/Polycube suite) and will develop the necessary agents for their remote control. An additional component is necessary to create secure data and control channels between the local agents/probes and the Context Broker (“Proxy” in Figure 2). The definition of this component is not defined by the ASTRID framework, but it is left to each orchestrator; alternative architectural solutions have been already briefly discussed in D1.2. Each Use Case is therefore expected to use or extend the solutions it is already using for connecting service components deployed in different infrastructures. Examples of such technologies include VPN, OOR [3], ISTIO [4], OVS [5]. Depending on the specific solution, it may be necessary to co-locate a Proxy within each VF (as indicatively shown for the cloud use case) or a common Proxy for all components in the same infrastructure (NFV use case). The only mandatory requirement is the usage of encryption and integrity mechanisms to guarantee the reliability of the communication.

The ASTRID Security Controller and Security Dashboard will be used to request specific management actions to the service orchestrator. Internally, the ASTRID framework will use a generic abstraction of common operations that are expected to be available, but each use case shall provide a translation between the internal representation and the real API / NBI available on the orchestrator. Such translations will be implemented as *drivers* or *plugins*, according to the documentation of the ASTRID framework, and will be part of the real implementations.

3.2 MAESTRO Orchestrator

MAESTRO is a Cloud/Edge Computing Applications Orchestrator that tackles the overall lifecycle of cloud/edge computing applications’ design, development, deployment, and orchestration. A set of novel concepts are introduced, including the design and development of cloud/edge computing applications - based on cloud-native/microservices-based principles and the separation of concerns among the orchestration of the developed applications and the required network services that support them. MAESTRO follows a top-down approach where applications design and development leads to the instantiation of app required infrastructure (network/compute), over which cloud/edge computing applications can be optimally served. Different stakeholders are engaged in this process, however with clear separation of concerns among them.

- **Software developers** are developing applications following a microservices-based approach where each application component can be independently orchestratable. Based on the conceptualization of a metamodel (application graph metamodel), they declare information and requirements -in the form of an application descriptor- that can be exploited during the applications deployment and operation over programmable infrastructure. Such information and requirements may regard capabilities, envisaged functionalities and soft or hard constraints that have to be fulfilled and may be associated with an application component or a virtual link interconnecting two components within an application graph.
- **Application/Service Providers** are able to adopt the developed applications and specify policies and configuration options for their optimal deployment and operation over programmable infrastructure. Based on the provided application descriptor, application/service providers are able to design operational policies (elasticity efficiency or security policies) that have to be applied as well as declare further deployment constraints. Operational policies regard runtime adaptation of the execution mode of an application

component. Cloud/Edge computing applications orchestration is realised following a service-mesh-oriented approach.

- **Infrastructure Service Providers** are getting the application deployment request and proceeding to the appropriate reservation of resources for serving the application needs. These actions are realized in an agnostic way to application service providers. However, through a set of open APIs, requests for adaptation of the infrastructure configuration may be provided.

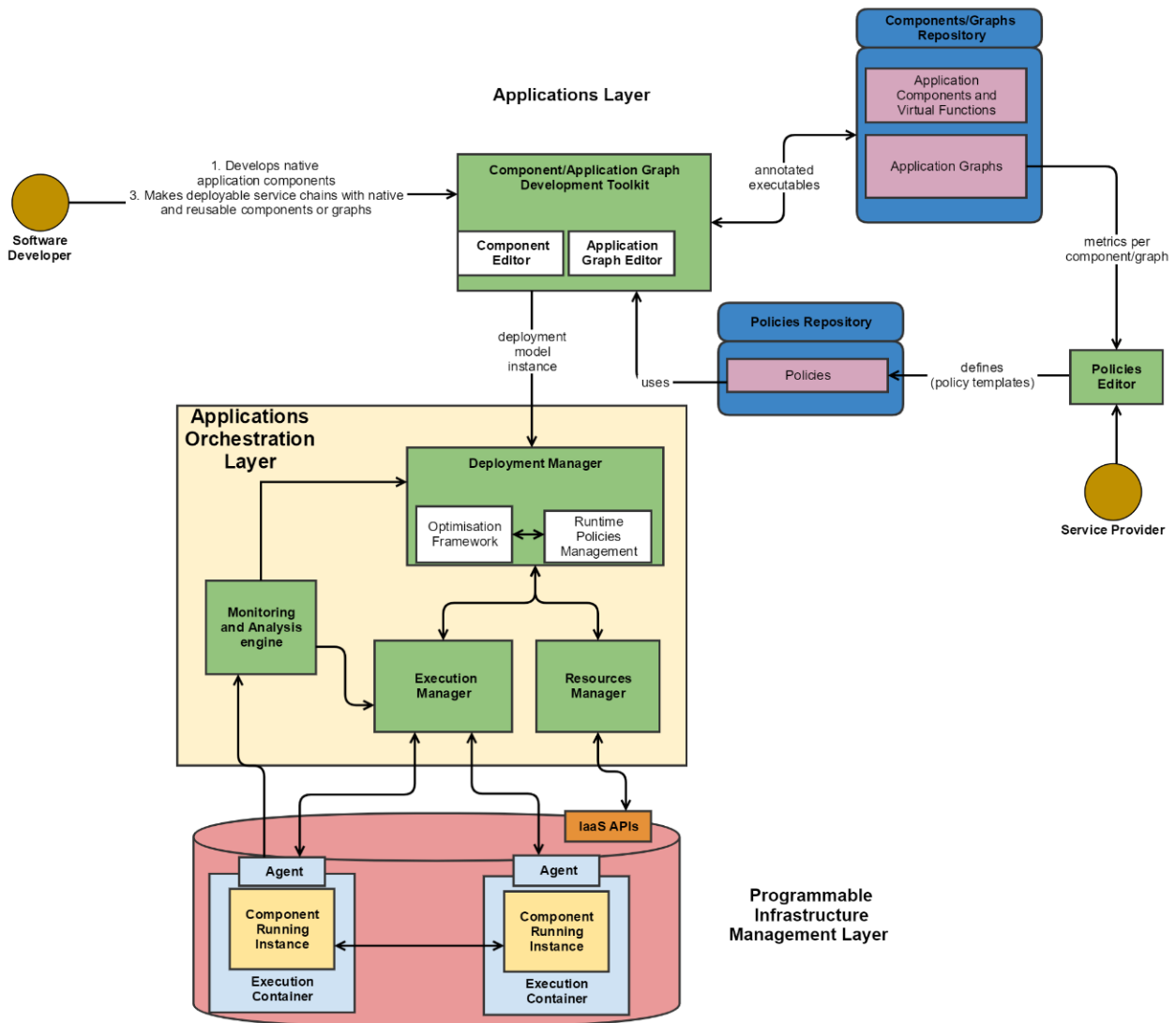


Figure 3. MAESTRO Architecture

The **MAESTRO reference architecture** is divided into three distinct layers, namely the Applications Layer, the Applications' Orchestration Layer, and the Infrastructure Management Layer. Separation of concerns per layer is a basic principle adhered towards the design of the overall architecture. The Applications Layer is oriented to software developers, the Applications Orchestration Layer is oriented to application/service providers and the Infrastructure Management Layer is oriented to infrastructure providers.



- The **Applications Layer** takes into account the design and development of cloud/edge computing applications, along with the specification of the associated deployment infrastructure requirements. The associated networking/computational requirements are tightly bound together with their respective applications' graph, which defines the business functions, as well as the service qualities of the individual application.
- The **Applications' Orchestration Layer** supports the dynamic on-the-fly deployment and adaptation of the applications to its service requirements, by using a set of optimisation schemes and intelligent algorithms to provide the needed resources across the available multi-site programmable infrastructure. Deployment and runtime policies enforcement is provided through a set of optimisation mechanisms providing deployment plans based on high-level objectives and a set of mechanisms supporting runtime adaptation of the application components and/or network/security functions based on policies defined on behalf of a services provider. The **Service mesh** concept is adopted as a software management layer for controlling and monitoring internal, service-to-service traffic in microservices-based applications across multiple infrastructures. It consists of a data and a control plane. The data plane consists of a set of intelligent proxies deployed alongside the application software components supporting the provision of support/backing services (e.g. service discovery, load balancing, health checking, telemetry). The control plane manages the set of intelligent proxies based on distributed management techniques and provides policy and configuration guidance for all the running support/backing services. Policies definition for the activation and management of the set of required support/backing services is realised based on a policies editor, while policies enforcement is realised based on a rules-based management system. Advanced monitoring and analysis techniques are also applied for extracting insights that can be proven useful for application/service providers.
- The **Programmable Infrastructure Management Layer** is responsible for setting up and managing the application deployment and operation over the programmable multi-site infrastructure. Network and security mechanisms activation and orchestration, as well as monitoring streams management, are realized. Such actions are triggered based on requests provided by the Applications' Orchestration Layer through the specification of Open APIs.

Following, a set of screenshots of the MAESTRO Orchestrator are provided, showing Dashboard as well as an already deployed application graph. From the Dashboard, a Developer can upload into the MAESTRO repository the Docker images, previously available in a personal registry, which will be used for composing the Application. Furthermore, runtime security policies can be defined per application graph instance, leading to the activation of a set of monitoring rules and alerts or mitigation actions, in cases of identification of a security threat.

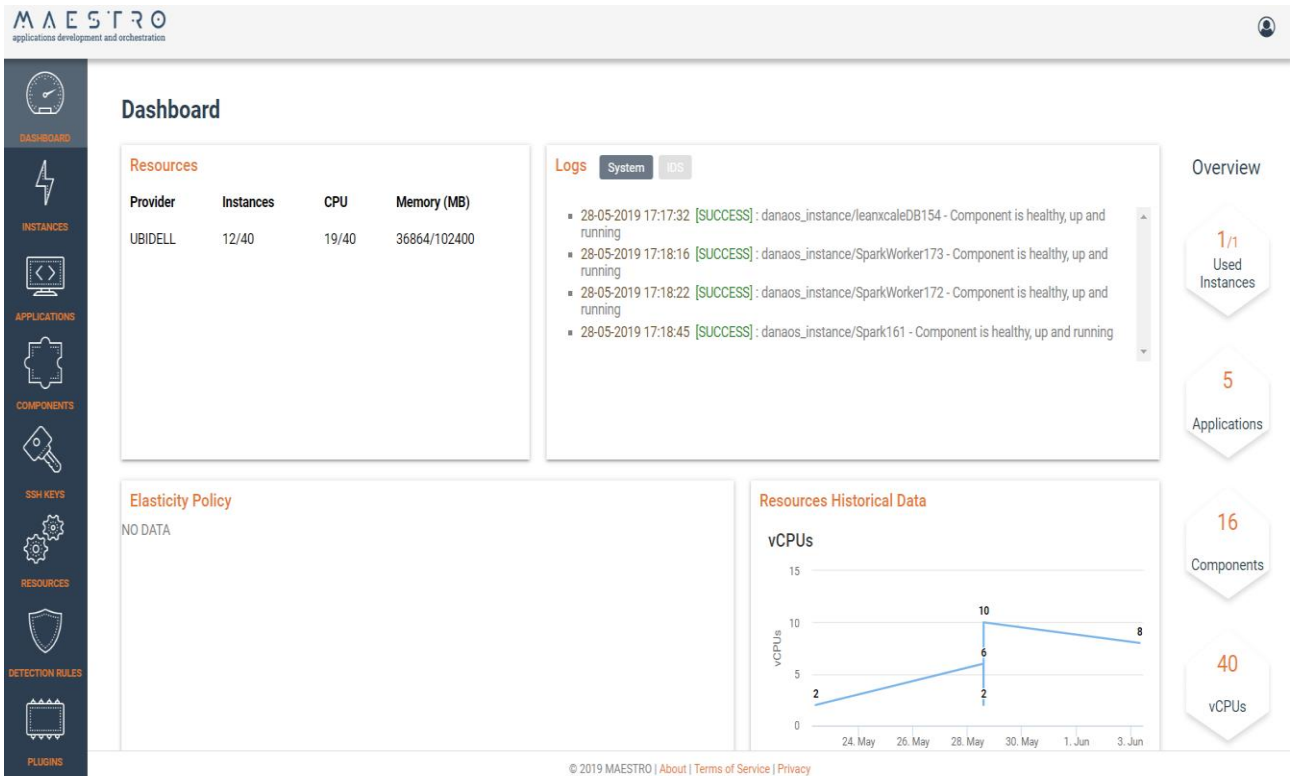


Figure 4. MAESTRO Dashboard

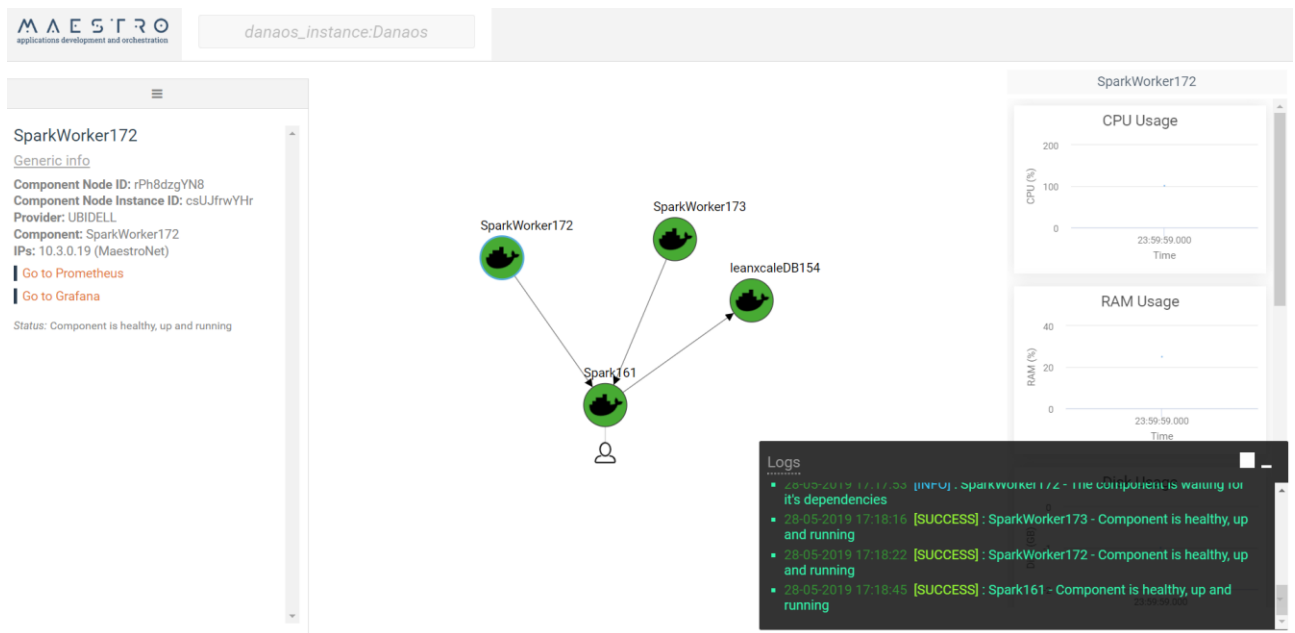


Figure 5. MAESTRO Application Graph Instance

3.3 OpenBaton Orchestrator

Open Baton² is the open source software solution developed by Technical University Berlin³ and Fraunhofer FOKUS, with the main objective of building a framework capable of orchestrating network services across heterogeneous infrastructural resources. The implementation of OpenBaton follows as much as possible the current NFV MANO architectural specification by ETSI. It provides:

- an NFVO and the message bus as central components,
- a Generic VNFM able to manage any kind of VNFs using a lightweight EMS,
- one or more specific VNFM, integrated via VNFM adapters,
- different drivers for interoperating with external VIMs and monitoring systems,
- several plug-in modules to extend the core functionalities (e.g., an FMS, an AES, and a NSE),

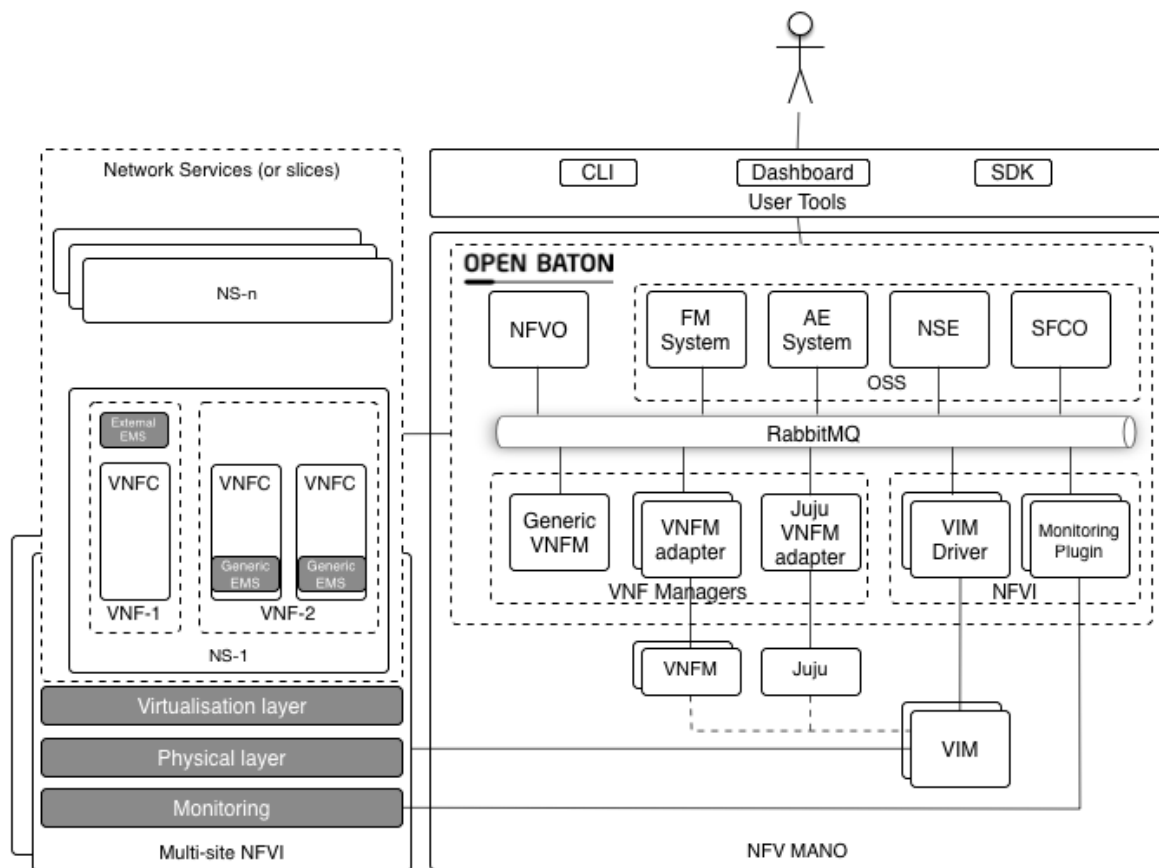


Figure 6. OpenBaton High-level Architecture

² <https://openbaton.github.io/index.html>

³ https://www.av.tu-berlin.de/research_development/tools/open_baton/



The high-level architecture of the Open Baton framework is shown in Figure 6, and a comprehensive list of features supported is provided as follows:

- Installation, deployment, and configuration of a large number of VNFs and NSs (i.e., vIMS, vEPC, etc.).
- Management of a multi-site NFVI Point of Presence, supporting heterogeneous virtualization and cloud technologies (i.e., OpenStack, Docker, etc.).
- Ensures multi-tenancy at the infrastructure level.
- Provides a Generic VNFM.
- Integrates with existing VNFMs (could be easily plugged either directly implementing the Or-Vnfm interface exposed by the NFVO, or via SDK available in different programming languages such as Java, Python, and Go).
- Supports runtime operations fulfilling the need of the FCAPS management model integrating external OSS systems, e.g. fault management (F) and auto-scaling (C).

The extreme required flexibility, led to the implementation of a loosely coupled microservice-oriented architecture, allowing each OpenBaton individual component to be implemented with the most suitable programming language (most of them have been implemented using Java). Figure 7 summarizes the different available OpenBaton components, their programming languages and project names on the GitHub⁴. The main NFVO component has been implemented as eight separate Gradle⁵ modules:

- **api**: providing the REST APIs exposed to the different consumers (being a human, or another component, like the Dashboard or the CLI).
- **cli**: Providing an implementation of a simple CLI used as part of the console of the NFVO.
- **common**: Including all the common source code across different other modules.
- **core**: Implementing the orchestration logic.
- **repository**: Including all the Java Entities which are used by other modules for persisting information in the database.
- **security**: Comprising all the classes which are dealing with authentication and authorization.
- **tosca-parser**: Including all the classes used for parsing TOSCA templates.
- **vnfm**: Providing the interfaces for communicating with available VNFMs

⁴ <https://github.com/openbaton>

⁵ <https://gradle.org/>



Project	Programming Language	GitHub Repository Project Name
NFVO	Java	NFVO
Dashboard	Javascript	dashboard
CLI	Java, Python	openbaton-client ¹
Dummy VNF Advanced Message Queuing Protocol (AMQP)	Java, Python, Go ²	dummy-vnfm-amqp ³
Dummy VNF REST	Java	dummy-vnfm-rest
Generic VNF	Java	generic-vnfm
Generic EMS	Python	ems
Juju VNF Adapter	Python	juju-vnfm
Docker VNF	Go	go-docker-vnfm
OpenStack VIM driver	Java	openstack4j-plugin
Test VIM driver	Java, Go ⁴	test-plugin ⁵
Docker VIM driver	Go	go-docker-driver
Autoscaling Engine	Java	autoscaling-engine
Fault Management System	Java	fm-system
Network Slicing Engine	Java	network-slicing-engine
Marketplace	Java	marketplace
Zabbix Plugin	Java	zabbix-plugin
Integration Tests	Java	integration-tests
OpenIMScore Packages	Bash	openimscore-packages
ClearWater Packages	Bash	clearwater-packages

Figure 7. OpenBaton Components and GitHub Project Names

The dashboard represents a comprehensive web-based GUI exposing a set of web pages allowing end users to manage infrastructure resources and (network) services. The dashboard has been implemented using Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and Javascript. The source code has been included as a git submodule inside the NFVO project so that after starting the NFVO, the dashboard is automatically available. Figure 8 shows a screenshot of the overview page of the dashboard available immediately after login.

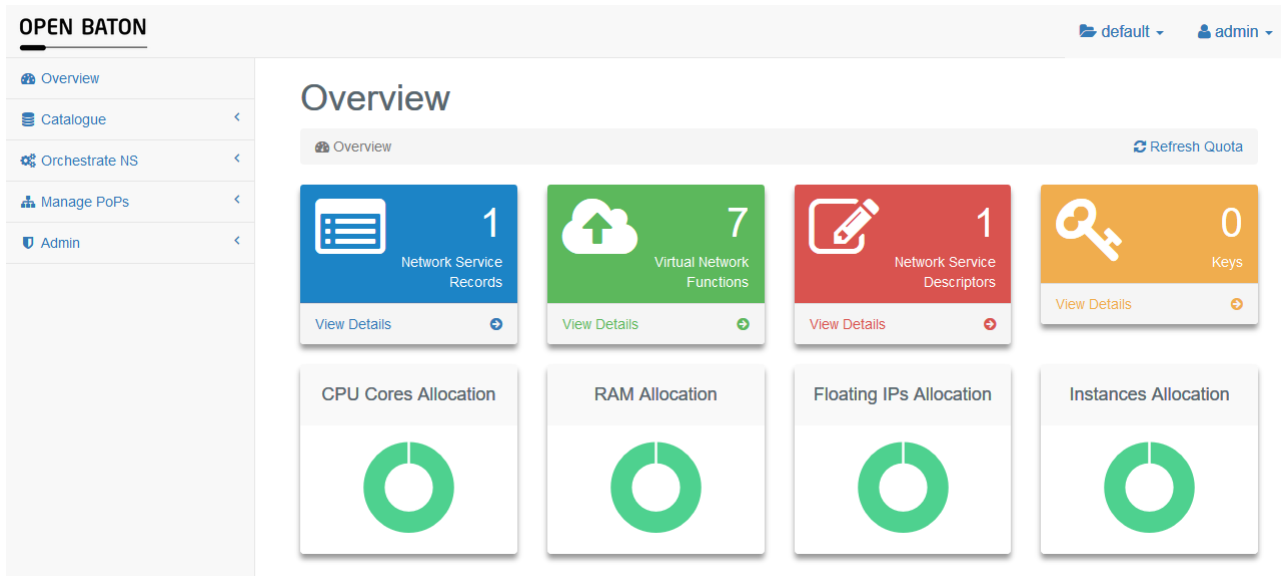


Figure 8. OpenBaton Dashboard

Last but not least, Open Baton provides several installation mechanisms including Docker (to provide an easy way to containerize services and compose more complex services in a template-based manner)

3.4 Kubernetes⁶

The Kubernetes orchestrator is the third orchestrator used in ASTRID for demonstration and validation. This section provides a high-level view of this software, originally developed by Google and later released in the open-source domain.

Kubernetes is an open source system for automating deployment, scaling, and management of containerized applications. Kubernetes enables to quickly deploy containerized applications, scaling it according to the user needs, without having to stop anything in the process. It is made to be portable, extensive and self-healing, granting an easier management from people who have to administrate the system.

General Architecture

The system is built around the following four key concepts:

- Nodes
- Pods
- Deployments
- Services

The high-level architecture of a Kubernetes cluster (which is, in Kubernetes terminology, the equivalent of a datacenter) is depicted in the figure below.

⁶ This Section has been partially inspired by <https://justanotherdevblog.com/kubernetes-an-overview-bf47b0af1865>.

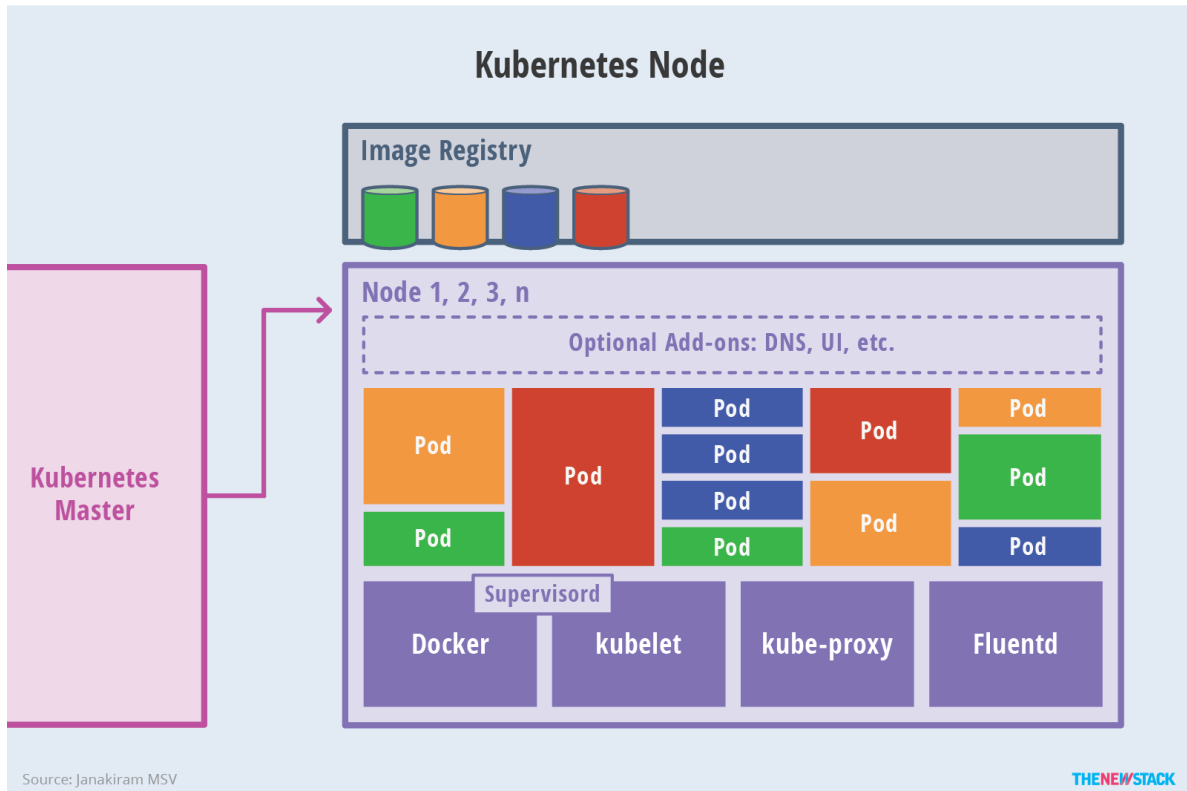


Figure 9. High-level Overview of a Kubernetes Cluster

Nodes

Nodes are the (either physical or virtual) machines on Kubernetes. When reading the statuses of a Node, the following information can be obtained:

- Addresses: hostname, internal and external IP addresses from the Node.
- Condition: there are two conditions: OutOfDisk and Ready. OutOfDisk will be true when there is insufficient space to add new pods. Ready will tell if the Node is healthy and ready to accept pods. If the Controller cannot reach the Node for 40 seconds the Ready condition is set to Unknow.
- Capacity: the resources available in the node, such as CPU, memory and the maximum number of pods allowed.
- Info: general information about the node, such as program versions and OS name.

Node Controllers are the components to control the nodes. Their responsibility is to assign a CIDR block to the node when it is registered, which will ensure the proper IP is set. They also keep track of the relation between the list of nodes available and the list of machines available: when a node is deemed unhealthy the Controller will check if the machine is unhealthy as well. If that is the case, the node is removed from the list of available nodes.

Pods

Pods are composed of a group of one or more containers, the shared storage for them and their options. Kubernetes supports many types of containers, being Docker the most common one. Every container inside a pod will have the same IP address. As expected in this case, they can found themselves by connecting to *localhost* (i.e., the loopback network interface). However, this is valid only on



containers running on the same pod: if we have two containers located in two different pods they will have different IPs and cannot find each other using localhost.

Pods should not be treated as persistent entities. In a case of some failure (e.g. node failure) they will be destroyed, which means they cannot be used as a place to hold vital information. Data that needs to outlive the pod should be stored in volumes. Since pods are intended to be ephemeral it is important to understand their lifecycle.

Each pod will have a phase, which can be described as a high-level summary of the current status of the pod in the lifecycle. A phase can assume the following values:

- Pending: the pod was already accepted by Kubernetes, but that are images still being created.
- Running: all of the containers have been created and the pod is bound to a node.
- Succeeded: the pod has terminated all containers with success.
- Failed: the pod has terminated and at least one container failed.
- Unknown: it was not possible to obtain the status of the pod.

Deployments

Deployments are used to make updates on Pods. They can be used to bring up new Pods, change the image version of a container and even recreate the previous state if something goes wrong. When creating a Deployment, users can define a desired state and Kubernetes will keep our environment in that desired state.

Let's pretend that we want to make sure we always have 3 pods running a web server. This can be achieved by creating a Deployment that defines the replicas property as 3. What Kubernetes does when it runs this Deployment is to create 3 Pods with the given configuration of the web server. If for some reason one of the pods is destroyed Kubernetes will automatically bring up a new one. This will make our desired state of 3 replicas be achieved even when some problems occur.

The status of the deployment can be monitored to see if everything is performing according to what is expected. When looking for a Deployment status, the following information is available:

- Desired: how many pods were defined in the desired state. When the deployment is finished the number of current pods should be equal to this column.
- Current: indicates the total replicas the Deployment manages.
- Up-to-date: how many pods have the latest template. For instance, the container's image version is changed and the Deployment is executed again, a pod will only be considered up-to-date when the deployment finishes.
- Available: how many pods are in the Ready status.

Services

Imagine that there are two services: ServiceA and ServiceB, the former needing to communicate with the latter. Since Pods are ephemeral we cannot use them to be the ServiceB interface. If a Pod ends up being terminated the reference to it is no longer valid and the environment will not work properly anymore. This requires something that is able to act as an interface and that will not be destroyed. Kubernetes achieves this using Services.

A Kubernetes Service consist of a set of Pods and a policy that defines the access control. Services can have label selectors, which are commonly used to invoke actions over the right subset of pods. This allows users to select a set of instances based on the given information.

When publishing Kubernetes Services, users can define how they can be exposed. For instance, a backend service usually is going to be accessible inside the local network, while a frontend service needs to be available outside the cluster. The possible types of Service we can define are listed below:



- ClusterIP: the service is going to be exposed inside the cluster, with a local IP, and will not be reachable outside the cluster. This is the default option.
- NodePort: exposes the service on the given port, using the node IP. For instance, if the Node runs on 10.0.15.5 and the NodePort is 4567, we can reach the service on 10.0.15.5:4567.
- LoadBalancer: exposes the service using a cloud provider's load balancer.
- ExternalName: the service is going to be exposed using the name configured on this property (e.g. mydomain.com).

Key Design Principles⁷

Kubernetes is designed on the principles of scalability, availability, security, and portability. It optimizes the cost of infrastructure by efficiently distributing the workload across available resources. This section will highlight some of the key attributes of Kubernetes.

Workload Scalability

Applications deployed in Kubernetes are packaged as microservices. These microservices are composed of multiple containers grouped as pods. Each container is designed to perform only one task. Pods can be composed of stateless containers or stateful containers. Stateless pods can easily be scaled on-demand or through dynamic auto-scaling. Kubernetes 1.4 supports horizontal pod auto-scaling, which automatically scales the number of pods in a replication controller based on CPU utilization. Future versions should support custom metrics for defining the auto-scale rules and thresholds.

Hosted Kubernetes running on Google Cloud also supports cluster auto-scaling. When pods are scaled across all available nodes, Kubernetes coordinates with the underlying infrastructure to add additional nodes to the cluster.

An application that is architected on microservices, packaged as containers and deployed as pods can take advantage of the extreme scaling capabilities of Kubernetes. Though this is mostly applicable to stateless pods, Kubernetes is adding support for persistent workloads, such as NoSQL databases and relational database management systems (RDBMS), through pet sets; this will enable scaling stateless applications such as Cassandra clusters and MongoDB replica sets. This capability will bring elastic, stateless web tiers and persistent, stateful databases together to run on the same infrastructure.

High Availability

Contemporary workloads demand availability at both the infrastructure and application levels. In clusters at scale, everything is prone to failure, which makes high availability for production workloads strictly necessary. While most container orchestration engines and PaaS offerings deliver application availability, Kubernetes is designed to tackle the availability of both infrastructure and applications.

On the application front, Kubernetes ensures high availability by means of replica sets, replication controllers and pet sets. Operators can declare the minimum number of pods that need to run at any given point of time. If a container or pod crashes due to an error, the declarative policy can bring back the deployment to the desired configuration. Stateful workloads can be configured for high availability through pet sets.

For infrastructure availability, Kubernetes has support for a wide range of storage backends, coming from distributed file systems such as network file system (NFS) and GlusterFS, block storage devices such as Amazon Elastic Block Store (EBS) and Google Compute Engine persistent disk, and specialized

⁷ This Section has been partially inspired by <https://thenewstack.io/kubernetes-an-overview/>.



container storage plugins such as Flocker. Adding a reliable, available storage layer to Kubernetes ensures high availability of stateful workloads.

Each component of a Kubernetes cluster — etcd, API server, nodes— can be configured for high availability. Applications can take advantage of load balancers and health checks to ensure availability.

Security

Security in Kubernetes is configured at multiple levels. The API endpoints are secured through transport layer security (TLS), which ensures the user is authenticated using the most secure mechanism available. Kubernetes clusters have two categories of users — service accounts managed directly by Kubernetes, and normal users assumed to be managed by an independent service. Service accounts managed by the Kubernetes API are created automatically by the API server. Every operation that manages a process running within the cluster must be initiated by an authenticated user; this mechanism ensures the security of the cluster.

Applications deployed within a Kubernetes cluster can leverage the concept of secrets to securely access data. A secret is a Kubernetes object that contains a small amount of sensitive data, such as a password, token or key, which reduces the risk of accidental exposure of data. Usernames and passwords are encoded in base64 before storing them within a Kubernetes cluster. Pods can access the secret at runtime through the mounted volumes or environment variables. The caveat is that the secret is available to all the users of the same cluster namespace.

To allow or restrict network traffic to pods, network policies can be applied to the deployment. A network policy in Kubernetes is a specification of how selections of pods are allowed to communicate with each other and with other network endpoints. This is useful to obscure pods in a multi-tier deployment that shouldn't be exposed to other applications.

Portability

Kubernetes is designed to offer freedom of choice when choosing operating systems, container runtimes, processor architectures, cloud platforms, and PaaS.

A Kubernetes cluster can be configured on mainstream Linux distributions, including CentOS, CoreOS, Debian, Fedora, Red Hat Linux and Ubuntu. In fact, a Kubernetes cluster can control even non-Linux nodes, such as Windows hosts. It can be deployed to run on local development machines; cloud platforms such as AWS, Azure and Google Cloud; virtualization environments based on KVM, vSphere, and libvirt; and bare metal. Users can launch containers that run on Docker or rkt runtimes, and new container runtimes can be accommodated in the future.

Through federation, it is also possible to mix and match clusters running across multiple cloud providers and on-premises. This brings the hybrid cloud capabilities to containerized workloads, hence enabling customers to seamlessly move workloads from one deployment target to the other.

3.5 Virtualization Infrastructures and Management Software - OpenStack

OpenStack⁸ is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacentre. Users can manage it through a dashboard that gives administrators control, while empowering their users to provision resources through a web interface, through command-line tools or a RESTful API.

⁸ <https://docs.openstack.org/stein/>

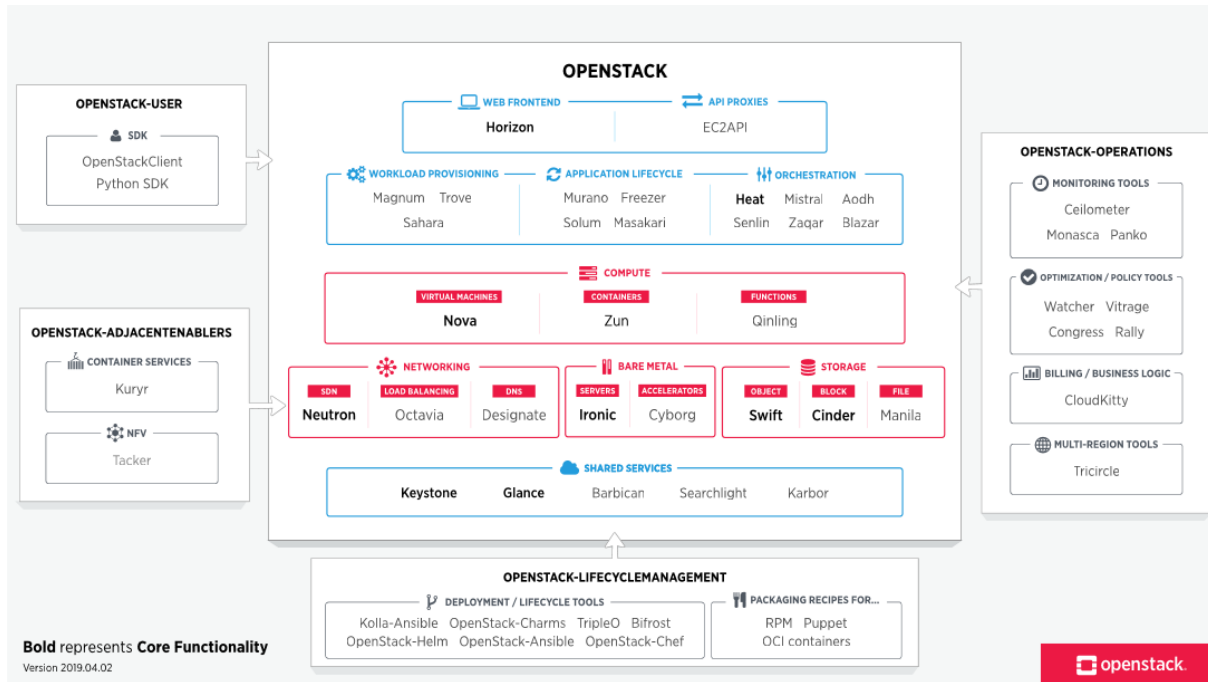


Figure 10. OpenStack Map⁹

OpenStack was founded on 21st July 2010 by Rackspace, NASA, and further 25+ partners. More than 160 companies and 3000 developers have contributed to the project so far. OpenStack has a strong ecosystem with different OpenStack-powered products/services in the market and hundreds of the world’s largest brands utilize it to run their businesses. As shown in Figure 10, OpenStack has a modular architecture with various code names for its components (e.g. Nova – Compute, Neutron – Networking, Cinder – Block Storage, Keystone - Identity). Figure 10 gives an “at a glance” view of the OpenStack landscape to see where those services fit and how they can work together.

After the introduction of Network Function Virtualization (NFV) by ETSI, OpenStack has emerged as a key virtual infrastructure platform for the management and orchestration infrastructure (NFV MANO). In most of the NFV deployments, OpenStack is used at the VIM (Virtual Infrastructure Manager) layer to give a standardized interface for controlling, monitoring and assessing all resources within NFV infrastructure. VIMs are critical to realize the business benefits enabled by NFV as they coordinate the physical and virtualised resources necessary to deliver network services. In ASTRID, OpenStack will be used in both ASTRID use cases to provide managed virtual infrastructures.

3.6 Benchmarking Tools – Kali Linux

In the field of cybersecurity, there are many security benchmark/testing/hacking tools available. Kali Linux¹⁰ is the most common solution with a large number of preinstalled tools from various different niches of the security and forensics fields. Kali Linux is created and maintained by “Offensive Security” who focuses on advancing security through tools and education.

⁹ <https://www.openstack.org/software/>

¹⁰ <https://tools.kali.org/>

As shown in Figure 11, Kali Linux can be used to support different requirements including information gathering, vulnerability analysis, stress testing, reverse engineering, and includes forensics tools as well as an abundance of attack tools (e.g. VoIP, wireless, web, password, sniffing and spoofing).

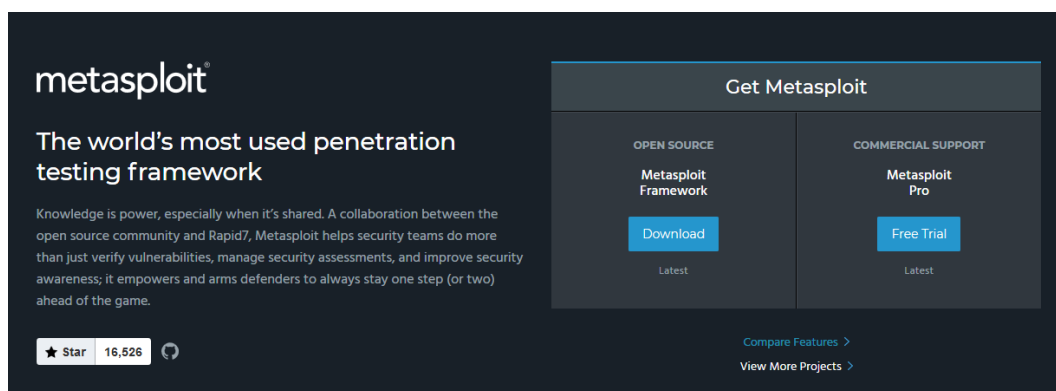


The screenshot shows the 'KALI TOOLS' website with a navigation menu (Home, Tools Listing, Metapackages) and a main heading 'Kali Linux Tools Listing'. Below the heading are four columns of tool categories:

- Information Gathering:** ace-voip, Amap, APT2, arp-scan, Automater, bing-ip2hosts, braa, CaseFile, CDPSnarf, cisco-torch, copy-router-config
- Vulnerability Analysis:** BBQSQL, BED, cisco-auditing-tool, cisco-global-exploiter, cisco-ocs, cisco-torch, copy-router-config, Doona, DotDotPwn, HexorBase, jSQL Injection
- Wireless Attacks:** Airbase-ng, Aircrack-ng, Airdecap-ng and Airdecloak-ng, Aireplay-ng, airgraph-ng, Airmon-ng, Airodump-ng, airodump-ng-oui-update, Airolib-ng, Aircserv-ng, Airtun-ng
- Web Applications:** apache-users, Arachni, BBQSQL, BlindElephant, Burp Suite, CutyCapt, DAVTest, deblaze, DIRB, DirBuster, fimap

Figure 11. Kali Linux Tools Listing

Kali Linux includes the Metasploit Framework¹¹, an open source penetration testing and development platform that provides exploits for a variety of applications, operating systems, and platforms. This Ruby-based Framework contains a suite of tools that can be used to test security vulnerabilities, enumerate networks, execute attacks, and evade detection. Users can leverage the power of the Metasploit Framework to create additional custom security tools or write their own exploit code for new vulnerabilities.



The screenshot shows the Metasploit Framework website. It features the 'metasploit' logo and the tagline 'The world's most used penetration testing framework'. Below this is a paragraph of text: 'Knowledge is power, especially when it's shared. A collaboration between the open source community and Rapid7, Metasploit helps security teams do more than just verify vulnerabilities, manage security assessments, and improve security awareness; it empowers and arms defenders to always stay one step (or two) ahead of the game.' There is a star icon with '16,526' next to it. On the right, there is a 'Get Metasploit' section with two columns: 'OPEN SOURCE Metasploit Framework' with a 'Download' button, and 'COMMERCIAL SUPPORT Metasploit Pro' with a 'Free Trial' button. At the bottom right, there are links for 'Compare Features' and 'View More Projects'.

Figure 12. Metasploit Framework

¹¹ <https://www.metasploit.com/>

4 Use Cases Description

4.1 Security Orchestration in Cloud Environment

4.1.1 Overview

The first Use Case relates to the provision of a secure voice communication service (VoIP). This scenario is representative of emerging cloud applications that process sensitive data and need proper cyber-security solutions (including privacy, confidentiality, detection of threats and compromised software) for safeguarding the interest of users. With most major telecommunications carriers currently in the process of readying Voice-over IP (VoIP) services for mass deployment, it's clear that IP telephony is finally headed for prime time. However, the promise of mass VoIP consumption also increases the risk of widespread security violations, spawning a new sense of urgency to fill in potential security gaps now before hackers wreak havoc on public and corporate voice networks.

The Secure VoIP Communication (SeVoC) application is an application that enables encrypted voice communication between application users. SeVoC integrates with the system dialer to provide a frictionless call experience but uses ZRTP to set up an encrypted VoIP channel for the actual call. It is designed specifically for mobile devices, using audio codecs and buffer algorithms tuned to the characteristics of mobile networks and using push notifications to maximally preserve your device's battery life while still remaining responsive. Even more, it provides end-to-end encryption for your calls, securing your conversations so that nobody can listen in. It's easy to use and supports functions just like the normal dialer that most people are accustomed to. The SeVoC application uses a normal mobile number for addressing, so there's no need to have yet another identifier or account name; by knowing someone's mobile number, the end user knows how to call them using the mobile application for Android. When a call is received, the phone will ring just like normal, even if it is asleep. Some of the features of the app:

- Conference Mode: Stands for encrypted conference call, allowing more than two users to speak together simultaneously.
- Message: Sending an encrypted message to your partner or a friend knowing that only the actual receiver is able to read it.
- Repository: Every user has his/her own electronic safe online. You can choose to keep any message or call you have recorded, secure and encrypted to the server. Even if your phone is stolen or broken you are able to access your data from a web browser or by installing the SeVoC on a new device.
- Some other features are: voicemail, missed call notification, delivery report of message, hide number during a call or send a private message.

Furthermore, the Secure VoIP Communication application gives the ability to record the conversation during a call or even forward a call to another SeVoC user. One major asset is that regardless of the feature(s) the client uses, communication safety is guaranteed.

4.1.2 Service Topology

In the following figure, it is depicted the SeVoC application graph and the components that it is comprised of.

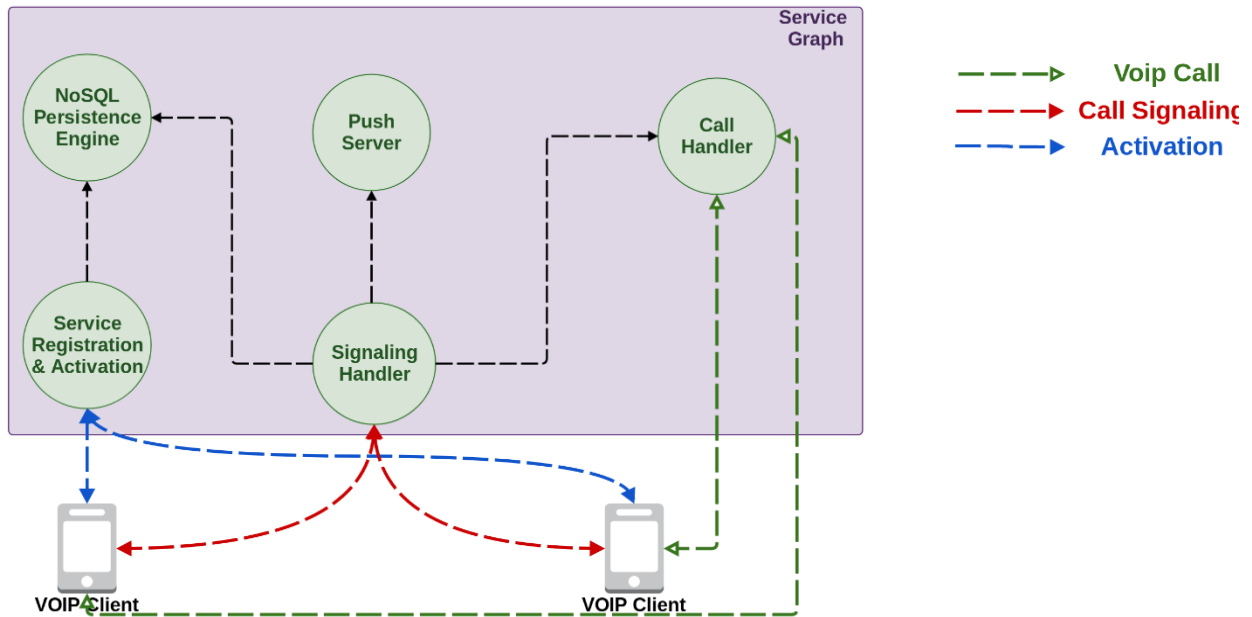


Figure 13. Use Case #1 Service Graph

The **Signalling Handler** is the component that handles all the signalling that is required during the operation of a VoIP client. This includes signalling regarding the connection status, the call establishment, the hang-up process, etc. As a component, it is dependent on the NoSQL repository and a PUSH server in order to be able to send synchronous notifications to a mobile device. When a call request is accepted by both parties, then a Call Handler undertakes the work of maintaining a valid channel.

Exposed Endpoints	Signal Management: It exposes a publicly available endpoint (sigman) that is consumed directly by the mobile clients.
Required Interfaces	<ul style="list-style-type: none"> i. NoSQLStorage: regards the scalable by design repository used to hold the details of users and is covered by a component such as Mongo (mongotcp). ii. PushServer: regards the component that notifies the clients regarding the establishment and termination of a call (push). iii. TURNServer: regards the component that undertakes the actual establishment of a UDP channel.
Configurations parameters	ManagementPort: The initial port for handling the various signals
Metrics	SignalsPerSecond: The metric provides a metric regarding the throughput of handled signals.



The **Call Handler** is responsible for establishing a UDP connection between two endpoints that may be NATed. This functionality is essential for VOIP communication use cases where two parties can belong to different network zones.

Exposed Endpoints	TURNServer: Call handler exposes a “TURN” endpoint to any component that requires connectivity. A TURN server is a NAT traversal server and gateway. It can be used as a general-purpose network traffic TURN server and gateway, too. Online management interface (over telnet or over HTTPS) for the TURN server is available. The implementation also includes some extra experimental features.
Configurations parameters	<ul style="list-style-type: none"> i. StartPort: The initial port for the UDP handshakes (minimum 1100) ii. MaxAmountOfSessions: The maximum amount of sessions that can be supported
Metrics	<ul style="list-style-type: none"> i. ActiveCalls: The main component metric used regards the ActiveCalls that refers to the number of calls that are handled. Such a metric is used during policies definition for tackling performance aspects of the component. ii. ActiveCallsPerMinute: The metric provides a metric regarding the throughput of established calls.

The **Push Server** is responsible for providing push notifications on specific clients that are behind a NAT. In the frame of call management, PUSH notifications are used for call establishment and call termination.

Exposed Endpoints	PUSHMessageToReceiver: The component exposes only this interface (push). The current implementation is used in order to send an SMS message to a VoIP client. Other Push modalities can be supported in the future (e.g. Google Cloud Platform, AWS)
Configurations parameters	PushModality: The type of PUSH modality is selected through this variable. Currently, only SMS modality is available.
Metrics	MessagesPerSeconds: The metric provides a metric regarding the throughput of delivered messages.

The **NoSQL repository** and the **Registration component** are used to persist all required information of a user.

Security Features

With regards to security, SeVoC makes use of the current state-of-the-art encryption algorithms and cryptographic protocols in order to provide users the best possible effort among security issues. In specific the following algorithms and protocols are implemented:

- Data Encryption Algorithm: Advanced Encryption Standard 256 bit (AES-256)
The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data. AES is a winner data encryption-algorithm among many Cryptography Research and Evaluation Committees including CRYPTREC, NESSIE and NSA.



- Curve25519: it is an elliptic-curve cryptography algorithm (ECC) offering 128 bits of security and designed for use with the elliptic curve Diffie–Hellman (ECDH) key agreement scheme. It is one of the fastest ECC curves and is not covered by any known patents.
- HMAC as cryptographic primitive: stands for Hashed or Hash-based Message Authentication Code. It is a MAC algorithm derived from cryptographic hash functions. HMAC is a great resistant towards cryptanalysis attacks as it uses the Hashing concept twice. HMAC consists of twin benefits of Hashing and MAC and thus is more secure than any other authentication codes.
- Cryptographic Protocol: Secure Sockets Layer (SSL) SSL are cryptographic protocols that provide communication security over the Internet. SSL uses symmetric encryption for confidentiality (AES-256 in our solution).
- Cryptographic Key-Agreement Protocol: Zimmermann Real-time Transport Protocol (ZRTP). ZRTP is a cryptographic key-agreement protocol to negotiate the keys for encryption between two endpoints in a Voice over Internet Protocol (VoIP) phone telephony call based on the Real-time Transport Protocol.
- Voice Encryption Protocol: Secure Real-time Transport Protocol (SRTP) The Secure Real-time Transport Protocol (or SRTP) defines a profile of RTP (Real-time Transport Protocol), intended to provide encryption, message authentication, and integrity, and replay protection to the RTP data in both unicast and multicast applications.

4.1.3 Specific Threats

Voice communications are largely used for quick and often informal interaction, without the overhead of creating written documents or other media files. The digitalization and transmission over packet networks represent a more cost-effective technology with respect to traditional circuit-switching, but also introduce additional security concerns due to the usage of public, multi-service, and multi-tenancy infrastructures.

The transmission of voice packets over public networks makes VoIP vulnerable to several attacks that typically do not affect private infrastructures. The first security matter is confidentiality because packets travel across a number of different domains, some of which may be untrusted or even hostile. Encryption is commonly used by all VoIP services today, this Use Case not being an exception, so eavesdropping is a minor concern in this case. However, the persistent vulnerabilities in common routing and switching protocols for the Internet make spoofing, man-in-the-middle, and similar attacks real threats for this kind of applications, both in the core and access networks. In addition, availability and quality of service can be easily undermined by vicious types of (D)DoS attacks, which jeopardize the robustness and the stability of the app. The ASTRID framework performs statistical measures and packet inspection locally, hence collecting a pervasive security context that allows detecting missing packets and other forms of anomalies in the traffic flow. This provides a very effective solution which also covers the case when the discrete components of the application are deployed in different infrastructures.

The usage of the public cloud allows to scale the application according to the actual workload but also brings a number of threats due to multi-tenancy and lack of a strong security perimeter. Indeed, ensuring trustworthiness and integrity of a VoIP call is one of the most crucial aspects of the application, at least to guarantee confidentiality and privacy to its users. In that sense, there has to be a mechanism that not only guarantees the trustworthiness of the application's components per se but also verification and validation of their integrity and the graph behaviour upon instantiation. This functionality will be based on the capability to analyse system calls, so to provide static analysis and validation of the network functions as well as dynamic real-time behavioral analysis of the complete network service.



One very simple pre-requisite for integrity and availability of the application is strict control over access from the network. Firewalling is a basic yet effective security mechanism, also prone to vulnerabilities due to wrong or missing configurations. With dynamic services, ensuring the consistency of the rules across the service is a challenging task when managed by humans. The ASTRID framework envisions both the ability to enforce filtering rules locally and to compute the correct set of rules to be applied at each virtual function.

Table 1. Summary of Main Threats for the VoIP Use Case

Source	Threats	ASTRID improvements
Public Internet	DoS, spoofing, redirection	Custom network traffic statistics and analytics
Public cloud	Unauthorized access	Automatic firewall configuration
Public cloud	Service integrity	Remote attestation
End users	Illegal activities	Collection of logs and records for forensic analysis Interception and redirection of traffic flows

VoIP applications are likely to provide public communication services and should not represent a way to escape security controls already present in legacy telco infrastructures. As a matter of fact, voice communications are largely used for illegal activities, based on the fact that they are more difficult to detect than direct meetings, without violating the privacy of individuals. Illegal activities must be detectable, investigated, and prosecuted according to existing regulations and practice. Service providers are therefore expected to comply with local regulations that demand evidence of calls and interception capabilities upon authorization of relevant law enforcement agencies. ASTRID will support forensics and legal investigation by collecting logs from the VoIP components and keeping them in secure repositories. In addition, the elasticity of the cloud model will easily allow the re-direction of traffic flows to external monitoring agents. Table 1 briefly summarizes the main covered threats.

4.1.4 Implementation Plan

The implementation of this Use Case will follow the general workflow devised by the ASTRID architecture (D1.2) [2], including pre-deployment enrichment and run-time demonstration and validation.

Pre-deployment enrichment (May 2019 – November 2020)

As soon as the Secure VoIP Communication application gets into the Pre-deployment subsystem, an enhanced version of the original service graph will be created. This phase will include the following activities:

- i. Modification of the Call Handler so to enable duplication of call traffic towards an external entity. Redirection will be a management action triggered by the service orchestrator, upon request from the ASTRID Security Controller (Task 4.2).
- ii. Implementation of an additional component for legal interception. It acts as a gateway, which receives duplicated traffic and stores it for offline access and evidence.
- iii. The static software analysis will evaluate the source code of each component of SeVoC (mainly Signalling Handler, Call Handler, Push Server) against vulnerable outdated and un-patched code (Task 4.5).



- iv. Registration of call records (time, duration, caller identity and location, callee identity and location) and performance measures that can be used to detect anomalies (Task 4.2), which will be integrated with ASTRID monitoring hooks (Task 4.5), especially for the Signalling Handler which acts as the brain of the application.
- v. Development of eBPF programs to collect network statistics, to trace system calls, to detect traffic patterns, to filter packets. These will cover the range of attacks envisioned by the demonstration and evaluation plans (see Section 5, Task 4.5).
- vi. Development of the analytics toolkit for network threats envisioned by the demonstration and evaluation plans (see Section 5, Task 4.5).

Run-time demonstration and validation (December 2019 – April 2021)

The VoIP application will be integrated with the ASTRID framework developed in WP2 and WP3, and integrated into Task 4.4. Specific activities towards this objective include:

- i. Implementation of the REST APIs on the MAESTRO orchestrator and the ASTRID adapters to use the APIs, according to the model that will be defined by Task 2.1
- ii. Set-up of the demonstration and validation environment, including the mobile terminals and the tools to simulate the attacks described in the demonstration and evaluation plans (Section 5, Task 4.5).

The identification of volumetric DoS attacks is especially challenging in this context because it may concern the provider's infrastructure or other tenants, so it does not necessarily correspond to large volume of packets within the service. However, the overall network congestion caused by a volumetric DoS degrades the transmission of all packets, therefore affecting the service performance. Monitoring the metrics by which the VoIP service is measured is necessary to prevent the degree of deviation from the norm that will cause service deterioration. These measurements include latency, jitter, lost packets, MOS, and R-value.

- i. **Latency** — the measure of time delay in moving packets from the transmitting UE to the receiving one; the maximum duration of latency that a VoIP system can sustain without deterioration of service is 150 ms in any one direction. In SeVoC the target value for latency will be ≤ 30 ms.
- ii. **Jitter** — a variation in packet transit delay caused by queuing, congestion, timing drifts, route changes and serialization effects on the path through the network; the maximum allowable duration of jitter is 40 ms before deterioration occurs. In SeVoC the target value for jitter will be ≤ 20 ms.
- iii. **Lost packets** — the failure of one or more packets to reach their destination across the network; the maximum allowable packet loss is less than 1% for WANs and less than 0.05% for LANs. Target value is 0.03%.
- iv. **MOS** — mean opinion score is a subjective measure of voice quality that gives a numerical indication of the perceived quality of the media received; MOS is expressed as a number from 1 to 5, with 1 (bad) being the worst and 5 (excellent) being the best. A tolerated voice call is around 2.5 MOS. SeVoC's target for MOS is 3.5-4.0.
- v. **R-value** — a quantitative expression of the subjective quality of speech in communication systems for digital networks that carry VoIP or for which VoIP is under consideration; R-values range from 1 (worst) to 100 (best), and the metric is often used in conjunction with MOS, although the R-value is considered a more accurate portrayal of the effects of packet loss and latency. Targeted R Value for SeVoC is ≤ 90 .

All the protection and mitigation actions (based on rules on runtime policies) that are going to be applied in the SeVoC app will start from the Security Dashboard where the security provider can select detection algorithms, as well as define tailored reaction and mitigation strategies. In the Figure below, it is depicted a security rule that is triggered from the security analyst through ASTRID platform and enforces the blockage of traffic from the Signalling Handler to one of the three UEs.

Table 2 lists the required activities and responsible partner to implement the Use Case, together with a target deadline. These activities will be carried out in Task 4.2 and Task 4.5.

Table 2. Implementation Plans for Use Case #1

Component / Feature	Partner	WP / Task
Traffic duplication (legal interception)	UBITECH	T4.2
Static software analysis (remote attestation)	DTU/SURREY	WP3
Extended call records	UBITECH	T4.2
Logstash beat (log collection)	CNIT	WP2
eBPF programs (local inspection and monitoring)	UBITECH/POLITO	WP2
Analytics toolkit (detection of network threats)	ETI/UBITECH	WP3
Firewalling policy (automatic configuration of rules)	POLITO/UBITECH	WP2
Use activity and forensics data collection (automatic collection of logs)	INFO	WP2
Legal Interception (automatic collection of traces)	INFO	WP2
MAESTRO APIs (Northbound interface of service orchestrator)	UBITECH	T4.4

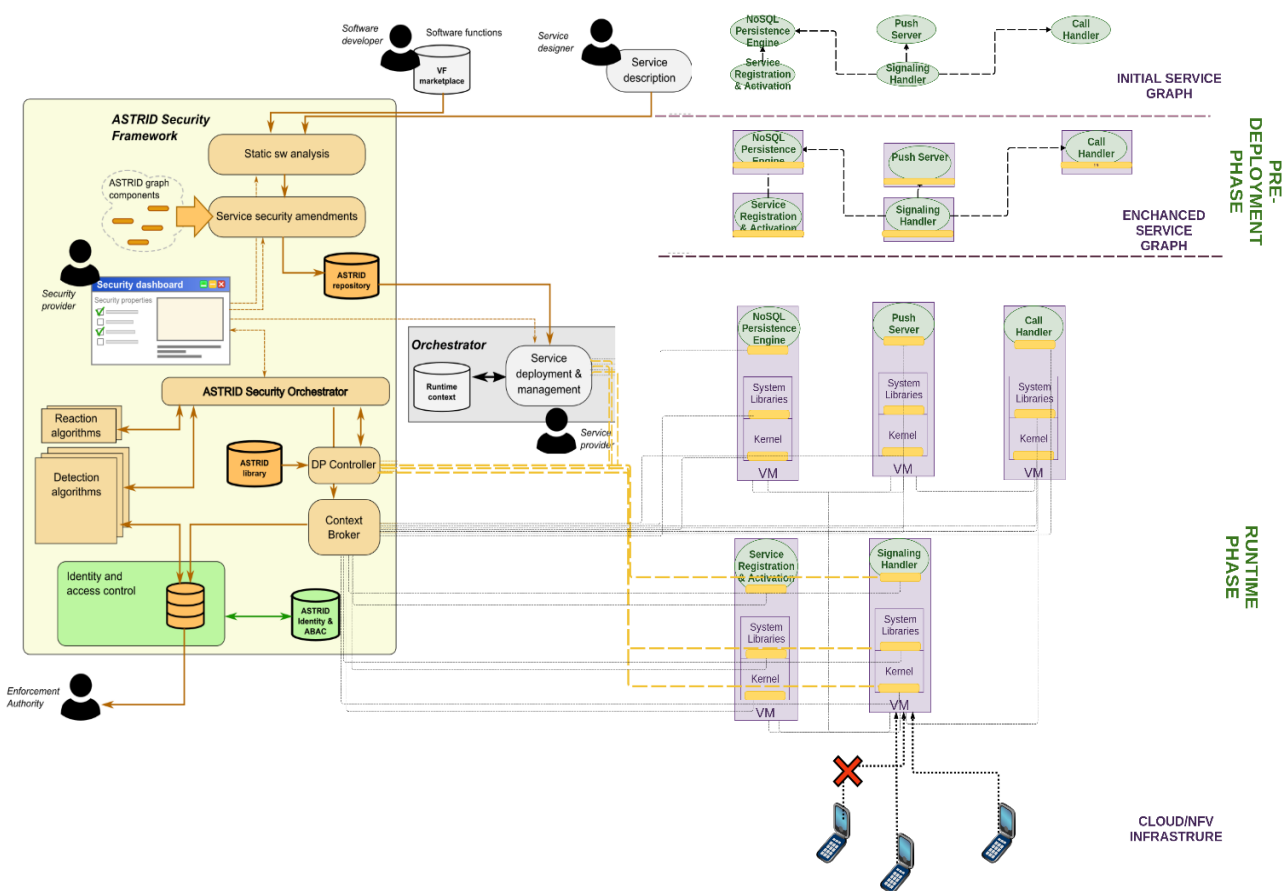


Figure 14. Use Case #1 within ASTRID Framework



4.1.5 Deployment Infrastructure

The SeVoC application will be deployed over multiple cloud installations, so to demonstrate the applicability of the ASTRID framework to multi- and cross-cloud environments. The cloud installations will be provided by UBITECH and CNIT. UBITECH will host the demonstrator for demonstration and functional validation, whereas CNIT will host part of demonstrator to carry out validation over more realistic scenarios.

The deployment and life-cycle of the enriched service graph will be managed by MAESTRO, which will also perform mitigation and reaction operations when triggered by the ASTRID Security Controller or Security Dashboard. An indicative deployment plan is shown in Table 3.

Table 3. Preliminary Deployment Plans

Component	Infrastructure
Signaling Handler	UBITECH
NoSQL Persistence Engine	UBITECH
Push Server	UBITECH
Service Registration & Activation	UBITECH
Call Handler	UBITECH/CNIT
VoIP clients	UBITECH
MAESTRO orchestrator	UBITECH
ASTRID orchestrator (run-time)	UBITECH/CNIT
ASTRID Security Dashboard	UBITECH

UBITECH's Cloud infrastructure includes more than 100 cores (about 400 vCPUs) at 2.4GHz with virtualization capabilities. Currently, KVM-based and ESXI-based hypervisors are used, OpenStack is used as Infrastructure-as-a-Service (IaaS) API. It should be noted that every 6 months the head version of OpenStack is adopted. The computational infrastructure is complemented by a Network Attached Storage (NAS) with an effective capacity of 25 Tb of data (in RAID 6 mode). Furthermore, a secondary NAS with an effective capacity of 20 Tb of data on a RAID 5 mode that can extend, on demand, the storage capacity of the aforementioned experimental Cloud Infrastructure to a total of 45 Tb of effective data storage. On top of the storage a Ceph-based API is used in order to offer storage-as-a-service functionality. Such functionality is exposed in various modalities such as S3-based object storage, block storage, and POSIX-based filesystem. Also, UBITECH possesses plenty of UEs with different Android versions (6.0, 7.0, 8.0).

CNIT's cloud infrastructure is built of 4 Intel KP-S2600KPR boards with 2x Intel E5-2660v4 @ 2.00 GHz and 128 GB of RAM, for a total of 112 physical cores and 512 GB RAM available to VMs. A dedicated controller node is run on a Intel KP-2600KPR board with 2x E5-2630v4 @ 2.20 GHz and 64 GB RAM. OpenStack Rocky is the current CMS for IaaS, using KVM/QEMU hypervisors. Block storage is provided on an iSCSI network, currently hosting 8 TB of data in hardware RAID 1 mode (mirroring).

4.2 Security Orchestration in NFV Environment

4.2.1 Overview

Mobile network architecture evolution is continuing toward the fifth generation (5G). The 5G networks are expected to have a high-speed data transfer, extremely low latency and ubiquitous connectivity support. Additional key driving factors will enter the scene including the convergence of mobile network and Cloud Computing or the support of **Internet of Things (IoT)** applications from **vertical industries** (e.g., automotive, healthcare, energy). In addition to that, to support new requirements, mobile operators are currently considering “network softwarization” for their network infrastructures by investigating several enabling software-related technologies such as Network Functions Virtualization (NFV) and Software Defined Network (SDN). With NFV, applications that were previously coupled to proprietary hardware can now be virtually instantiated and deployed on generic commercial off-the-shelf (COTS) computing hardware.

This use case is designed to showcase how ASTRID integrates gracefully with a NFV-based telecom system, which includes a set of multi-access edge cloud (MEC) nodes able to connect to a set of central cloud (CC) nodes across multiple available access/backhaul networks. As main use case to illustrate the effectiveness and efficiency of the ASTRID solution, the distributed networking platform implemented in demonstrator #2 will be comprehensively applied to a (mobile) IoT use case, named hereafter **Campus IoT-Vertical as a Service**. The demand for private campus networks [6] offers telco operators an opportunity for value generation with an estimated market size of 60-70 billion euro by 2025. The Network Service Provider (NSP), e.g., a Telco Operator or a Virtual Mobile Network Operator, is delivering to a customer campus-location a dedicated (and possibly private) managed network-service for connecting a number of (mobile) IoT- and end-user equipment-devices to one or more IoT-Applications.

4.2.2 Service Topology

The following Figure presents a high-level overview of the use-case service graph with its main components.

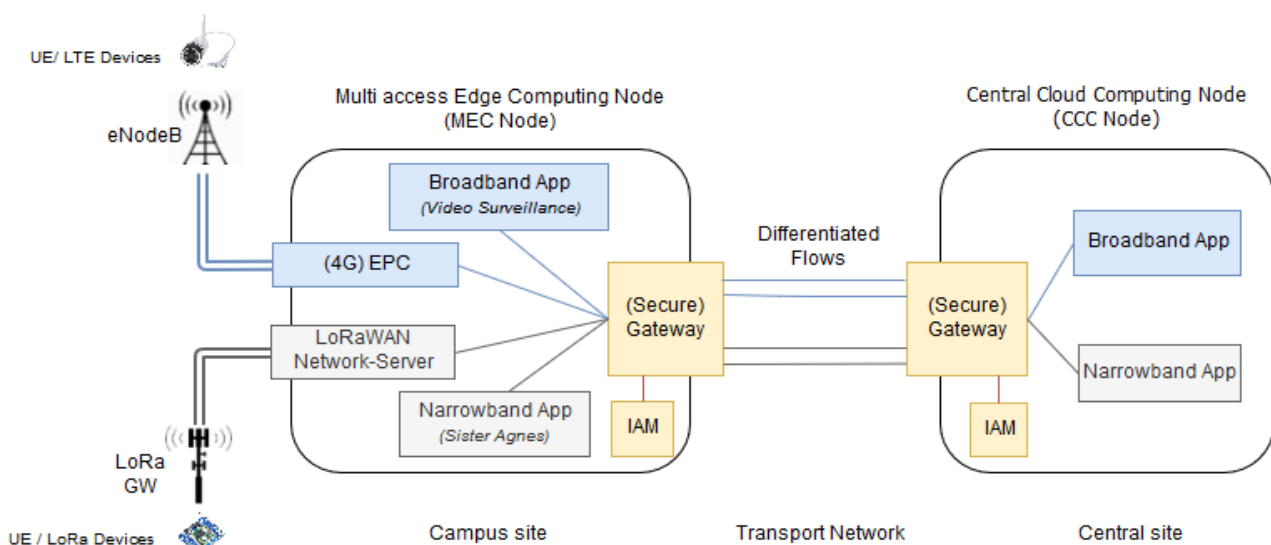


Figure 15. Use Case #2 Service Graph



Two different IoT applications will be implemented and tested, leveraging two common communication technologies: Cellular (LTE) and low power Long Range Wide Area Network (LoRaWAN). LTE is a great fit for applications that need high data throughput and large coverage. LoRaWAN is suitable for applications and devices that need to send small amounts of data over long distances with multi-year battery lifetime. In the former case, a video surveillance application will be developed allowing access to live video/image and recorded content from IP Cameras through an LTE/EPC network. In the latter case, an application that collects and displays different types of data from sensors (e.g. Temperature and Humidity¹²) over a LoRaWAN network will be developed.

To support the different access networks (LTE and LoRa), appropriate open source solutions will be investigated. As an illustration, a software deployment of the current LTE/EPC mobile network will be realized based on existing open source solutions (e.g., openair-cn¹³, nextEPC¹⁴, srsLTE¹⁵). For the LoRaWAN part, LoRaServer¹⁶ is selected. Two security-related components will be developed including the "Identity Access Management" (IAM) and the "Secure Gateway". The former implements the required authentication and attribute-based access control (ABAC) functionalities for the deployed applications. The later implements the "Proxy" component in Figure 2 to provide secure use-data- and control-data-channels between the application components as well as the local agents/probes.

4.2.3 Specific Threats

The global IoT market will reach \$4.3 trillion by 2024 according to Machina Research [7]. The number of new deployed devices/things are growing year by year. Such devices connect to different networks to provide collected information from environments to be later processed, analysed and stored on the cloud in order to make/take decisions. The convergence of Cloud computing and IoT brings business opportunities but it also raises new risks and challenges, especially due to the **poor security on the IoT devices**. Because of their poor attack- resistance, IoT devices are becoming the new targets of the fraudsters/hackers and many incidents were reported in the past. Hundreds of stolen SIM cards of smart light devices in Johannesburg were discovered too late [8] (the damage was not only losses caused by using these SIM for making malicious calls but also the traffic jam and the accidents). Botnets are growing larger and smarter than ever by exploiting poorly-secured IoT devices (e.g., CCTV cameras) to launch a DDoS attack [9]. Security-enabled IoT services are getting a lot of interest from the community [10] [11]. Table 4 briefly summarizes the main threats covered by this Use Case.

¹² <https://www.thethingsnetwork.org/marketplace/product/ls-113>

¹³ <https://github.com/OPENAIRINTERFACE/openair-cn>

¹⁴ <https://nextepc.org/configuration/03-lte/>

¹⁵ <https://github.com/srsLTE/srsLTE>

¹⁶ <https://www.loraserver.io/>



Table 4. Summary of Main Threats for the Use Case #2

Source	Threats	ASTRID improvements
Public Internet / Edge Access Network	DoS, spoofing, redirection	Custom network traffic statistics and analytics Mitigation from the Edge (close to the source of attack)
Devices/End Users	Illegal activities (DoS by Botnets, malicious requests)	Collection of logs and records for forensic analysis Mitigation from the Edge (close to the source of attack)
Insider	Attacks from compromised service components	Automatic firewall configuration

4.2.4 Implementation Plan

The use case implementation and validation will be conducted in two phases, aligned with the related WP4-Tasks, and follow the general workflow devised by the ASTRID architecture [2]

Pre-deployment Enrichment (May 2019 – November 2020)

This phase aims to develop the ASTRID use case components. The results will be provided in D4.2, D4.3, and D4.4. This phase will include the following activities:

- Develop all components/VNFs identified in Figure 15. Some components (EPC/LoRaWAN) will be developed by reusing existing open source solutions. The selection of suitable solutions will be based on a further investigation related to their capability set, flexibility and performance/resource consumption.
- Develop eBPF programs to collect network statistics, to trace system calls, to detect traffic patterns and to filter packets.
- Develop Logging/Logstash beat to collect logs from use case components

Run-time demonstration and validation (December 2019 – April 2021)

This phase aims to integrate and validate the use case components integrated into Task 4.4. using the ASTRID framework developed in WP2 and WP3. The results of this phase will be provided in D4.5, D4.6, and D4.7 The main tasks of this phase include:

- Develop an adapter/plugin to integrate the ASTRID framework with the OpenBaton Orchestrator and that translates ASTRID operations to the OpenBaton APIs
- Prepare all required component VNF-descriptors to be deployed and managed with OpenBaton
- Set-up of the demonstration and validation environment, including the ASTRID framework and benchmarking/analytics toolkits.
- Validate the ASTRID framework as further described in the demonstration and evaluation plans.

Table 5 lists the required activities and responsible partner to implement the Use Case # 2.



Table 5. Implementation Plans for Use Case #2

Component/Feature	Partner	WP/ Task
Use Case Components/VNFs implementations	TUB/AGE	T4.3
Build VNF packages	TUB/AGE	T4.4
OpenBaton Adapter implementation	TUB/AGE	T4.4
VNF Logging/Logstash beat (log collection)	CNIT	WP2
eBPF programs (local inspection and monitoring)	POLITO	WP2
Analytics toolkit (detection of network threats)	ETI	WP3
Firewalling policy (automatic configuration of rules)	POLITO	WP2

4.2.5 Deployment Infrastructure

To support the use case deployment on both the Edge and Central site nodes, two OpenStack (Queens version) virtual infrastructure environments (NFVI) have been set up in the TUB-IT cloud infrastructure (see Fig. 16 and Fig. 17).

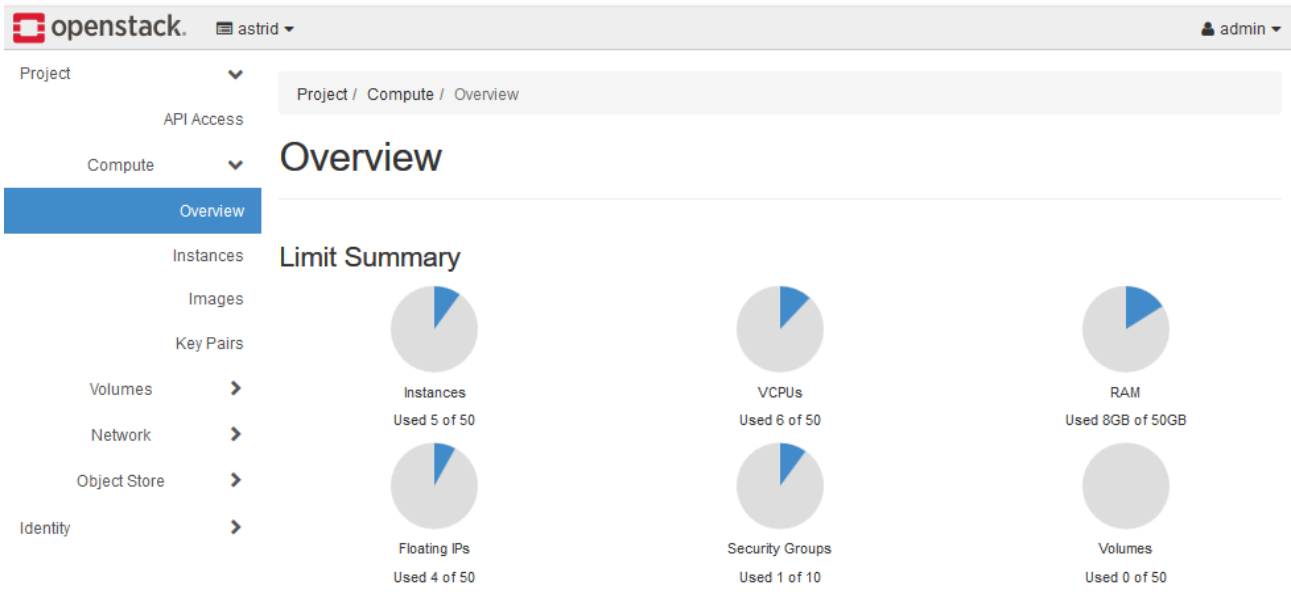


Figure 16. Use Case #2 OpenStack Infrastructure

Two other servers have been allocated for the deployment of the ASTRID framework and of the OpenBaton orchestrator. We will utilize the latest version of OpenBaton (Version 6). The current version supports the deployment of ASTRID use case components in different types of NFVI infrastructure including OpenStack and Docker. Other servers can be added to the ASTRID use case #2 deployment if needed.

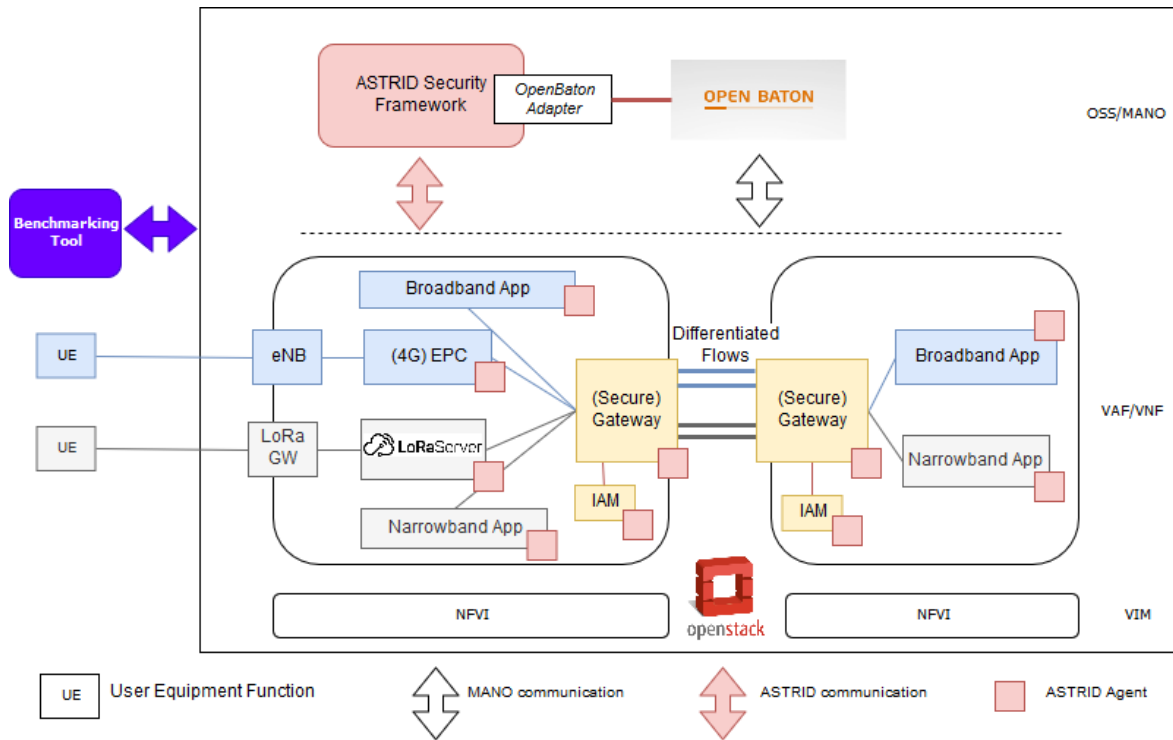


Figure 17. Use Case #2 within ASTRID Framework

4.3 Relation to the ASTRID Application Scenarios

In the early project phases, a number of reference application scenarios have been identified, to describe the main improvements of the ASTRID framework with respect to the current practice, which are described in D1.1 [1]. Table 6 lists the application scenarios. For each scenario, a reference to the descriptive Section in D1.1 is documented, together with an indication of the Use Case that includes such scenario.

Table 6. Mapping of Application Scenarios to Use Cases

Name	Use Case	Section in D1.1
Situational awareness for virtualized services	Security orchestration in cloud environment Security orchestration in NFV environment	3.7.1
Distributed firewall for cross-cloud applications and cyber-physical systems	Security orchestration in cloud environment Security orchestration in NFV environment	3.7.2
Programmable network traffic monitoring for DoS protection	Security orchestration in cloud environment Security orchestration in NFV environment	3.7.3
Trusted software and safe execution at run-time	Security orchestration in cloud environment	3.7.4
Response to Attacks and Threat Enabling Forensic Investigation	Security orchestration in cloud environment	3.7.5
Lawful interception	Security orchestration in cloud environment	3.7.6



As seen in Table 6, the Use Case #1 “Security Orchestration in Cloud Environment” is conceived to cover as much as possible all these scenarios, so to fully demonstrate the improvements brought by the ASTRID framework. On the other hand, there are several important contributions from Use Case #2 to the ASTRID objectives regarding the requirements to reduce the overhead of security processing (by providing protection at the edge, close to the source of attack) as well as to create a new potential market targeting telco operators. Telco operators have more opportunity and are better positioned to exploit the IoT applications markets by providing “context-aware” security-as-a-service solutions to vertical industry stakeholders. Regulated industries such as finance and healthcare need data to be stored and analysed on-premise of the enterprises. Attacks and data breaches need to be prevented/mitigated more quickly, close to the source, at the network-edge, instead of in central data-centre/commercial clouds.

5 Demonstration and Validation

Demonstration activities will prove the feasibility and functional validation of the ASTRID framework for the two selected Use Cases. Validation activities will evaluate the achievement of the technical objectives and expected impacts listed in the proposal. Demonstration and performance evaluation will be driven by the following validation objectives:

- **improved visibility** over cloud applications and NFV services deployed in multi- and cross-cloud environments, without relying on Third-Party- services and - infrastructures;
- **programmatic access to the security context**, made-up of data, events, and measures from a heterogeneous set of sources, balancing the depth and processing-overhead of inspection and monitoring processes;
- **faster and more effective / better reaction to attacks**, leveraging automation and orchestration tools to carry out mitigation and recovery actions in the execution environments and on the service graph.

Based on these objectives, a set of demonstration and validation scenarios has been selected, based on the threats model for each Use Case (Section 4) and target usage scenarios (D1.1, Section 3.7). Each demonstration and validation scenario describes the main objective, trial conditions, target KPIs and acceptance criteria. Additional validation scenarios, as well as extensions to those proposed in this document, might be identified by WP2/3, to better validate specific tools and algorithms.

As a general remark to performance evaluation, it is worth underlining that KPIs and acceptance criteria might be adjusted later on in the Project in case the testing conditions (components of the application, virtualization infrastructures, behaviour and complexity of attacks) should differ substantially from what was envisioned at this stage.

5.1 Automatic Firewalling

The correct configuration of firewalling rules is of paramount importance to avoid malicious network connections to applications and to permit network connections only for application- and platform-internal use (microservices, application-services) or debugging and testing purposes. In a distributed and dynamic cloud environment, the definition of firewall rules is challenging, because of the many components (and their versions) that may be present and the evolving topology. In this respect, ASTRID automates the generation and application of firewall rules, based on the service description, which defines what communication patterns are allowed within the service and with external entities (users and other applications).



Description. This scenario will demonstrate the correct behaviour in deriving and applying the firewall rules. A list of communication patterns will be derived automatically from the service description (e.g., by logical links between clients and servers); in addition, the Security Provider will insert additional constraints in the Security Dashboard (e.g., set of IP addresses allowed to use the service). The service will be deployed by the service orchestrator, while firewall rules will be set by the Security Controller at initialization time. After deployment, the actual rules will be retrieved through the Context Broker and checked for correctness.

Conditions. The validation will consider small/medium services (less than 10 discrete virtual functions). This is rather representative of common services which are often replicated for different service providers or users.

KPIs and acceptance criteria. The automatic computation of firewall rules must not delay indefinitely the initialization of the service. Based on typical deployment times, target KPIs and acceptance criteria are defined as follows:

Parameter	Target KPI	Acceptance criteria
Computation of firewall rules	2 min	5 min
Instantiation of firewall rules	30 sec	1 min

5.2 (Distributed) Denial of Service

Availability is an important (Service Level Agreement) feature for any kind of service provider, for safety, economic, or reputation reasons. Denial of Service (DoS) attacks make a service unavailable by consuming computing/storage/networking resources or by affecting its correct behaviour (e.g., by dropping or redirecting network packets). They are difficult to prevent and mitigate because it is difficult to distinguish between legitimate and malicious activities, especially in case of distributed DoS (DDoS) and spoofing. The availability of a rich security context from multiple sources (file logs, system calls, network packets) is expected to improve the filtering capability, while service orchestration can migrate the service to a different location when no mitigation actions are possible (for example, in case a network link or the provider's infrastructure is saturated).

Description. This scenario will demonstrate and validate the ability to mitigate (D)DoS attacks. The Security Provider will activate the "DDoS protection" feature (both prior to the service deployment and at runtime). Two kinds of DoS attack will be initiated, a directed SYN flood (with source spoofing) targeting the cloud/NFV application and an indirect volumetric attack targeting the infrastructure (NFVI) that hosts the application. The attacks will be performed with growing volumes, in order to understand if and when the attack is detected. The mitigation strategies will be based on filtering of malicious packets in case of direct attack and migration of the service to a different infrastructure in case of indirect attack.

Conditions. The two attacks will be generated with growing traffic volumes, proportional to a reference load for the application (to be defined based on the actual set up of the demonstrators) in the case of direct attacks and proportional to the link bandwidth of the infrastructure for indirect attacks. The volume will range from very small attacks (with limited impact but difficult to identify) to large attacks (easy to detect, difficult to mitigate selectively), according to the following plan:



Type	Target	Volume
Direct	Application component (signalling handler)	5%
		25%
		50%
		100%
		150%
Indirect	Network link (data centre)	5%
		25%
		50%
		75%
		100%

KPIs and acceptance criteria. The effectiveness of the ASTRID framework depends both on the capability of the detection-algorithms and on the reaction time. The two aspects will be evaluated separately, to better identify which component may need further improvement and development before exploitation.

Parameter	Target KPI	Acceptance criteria
Time to detect direct attacks	2 min	10 min
Time to detect indirect attacks	30 min	1 h
Time to trigger reaction	30 s	2 min
Time to migrate a single virtual function	1 min	2 min
Time to migrate an entire service	5 min	5 min

The above figures fit the more generic targets set in the proposal for detecting network anomalies (< 8 h), responding to attacks (< 2 min), replacing a single function (< 2 min), and re-deploying a service graph (< 5 min).

5.3 Malware and Intrusions

Cloud applications are usually booted from trusted images, but there are no guarantees they cannot be modified or get compromised at a later time. Intruders may guess valid credentials to log in, find backdoors to bypass security controls or extract secret information (i.e., keys, passwords, etc.) in an attempt to breach the confidentiality of the underlying communications among system entities and components. The introduction of malware is less likely than with desktop computers and laptops, because of the different usage patterns, but some forms of code injection through cross-site scripting or request-forgery are still possible for dynamic web applications (which are expected to be present in many virtual services). ASTRID collects a broad range of security events and security-context information including application logs, system logs, and system calls that can help identify anomalies and unexpected behaviours.



Description. This scenario will demonstrate and validate the ability of the ASTRID framework to provide strong guarantees for device and component integrity, during run-time, through continuously monitoring the execution of the deployed services in order to detect any deviations from the normal (expected) software execution flow. The goal here is to detect any attempts for breaching the security of any valuable data in transit (i.e., encrypted data reflecting VoIP calls) that rely on various infrastructural components in the service graph. However, the infrastructure will not behave as expected if a component is compromised. A tampered component could, for example, leak the data to a location chosen by the attacker, expose it for brute-forcing or compromise the encryption keys that have been established. Although the communication in the use cases will be protected (i.e., through the establishment of an end-to-end secure channel), components could still leak metadata about the traffic or can expose it for subsequent attacks on the secure protocols and/or cryptographic algorithms. To reduce such risks, the trusted management of these infrastructural components will take into account the integrity of the deployed components/services by means of the advanced remote attestation techniques provided by the ASTRID framework. Two concrete instances will be investigated: First, before the establishment of a secure communication channel between users, the Security Provider will initiate the process for attesting the integrity of the infrastructure entities that host the deployed functions in the service graph (i.e., Signalling Handler, Call Handler, etc.). This will be achieved through security policies, managed by the Security Provider, describing whitelists of the software executables that are allowed to be installed in an execution environment (VM, Container) so as to have strong guarantees on the integrity of the host entities. The attestation toolkit of the host machine will take secure measurements of the device's memory map (through the internal Trusted Component, i.e., Trusted Platform Module (TPM)) to be sent to the Context Broker for verification/attestation. Second, after the establishment of the communication session, continuous monitoring of the execution flow of the specific system and of low-level properties will be performed to detect any deviations from the expected control-flow graph that is indicative of the presence of potential malware. Particular focus will be given on attesting the execution flow of the functions (e.g., Call Handler) that handle the security and confidentiality of the communication session between users, to identify any malevolent attempt to compromise the session keys used for the cryptographic primitives (i.e., AES, HMAC keys, etc.). In this context, attack vectors will include the deployment of a service/function, during run-time, that is not part of the initial established whitelist (first case) and the deployment of targeted backdoors in specific execution environments running the Call Handler so as to then leverage advanced techniques such as Return-Oriented-Programming (ROP) [6] for exploiting the execution flow of such handlers (second case) without however altering the service's overall status (as this will be detected from the component integrity checker).

The mitigation strategies will be based on re-routing traffic to other components in the infrastructure that have not been deemed as compromised (immediate action) and on the initiation of the process for re-deploying the services whose control-flow graphs have failed to be attested from the ASTRID trusted repository (second-level action). Furthermore, since the output of the attestation is binary and can only reflect on whether there is a deviation in the execution flow, the Security Handler and Context Broker will also send system commands to activate additional monitoring hooks for investigating specific system calls, memory data structures, etc., in order to be sent for further offline investigation to concretely identify the specific vulnerabilities that led to such an attack.

Conditions. As already described in the ASTRID overall architecture [2], the presence of a Trusted Component is needed for performing the secure operations of the novel remote attestation toolkit. In the context of the experiments to be conducted, a virtual- or software-based TPM will be leveraged and will be installed in all infrastructural entities that can act as the prover and verifier in the attestation process (i.e., Security Handler, Context Broder, deployed VMs / Containers). Furthermore, as the goal of the project is not to showcase the execution of sophisticated attack vectors but their efficient detection and mitigation, targeted backdoors will be installed in specific entities that will act as the intrusion point

and allow further, more advanced tampering of the installed executables. Such executables may have been tampered even before their deployment in the service graph, although this is not the case in the normal execution of the ASTRID framework (as services and functions will have been statically analysed before being added in the Trusted Repository). In the context of the experiments, altered control-flow paths will be activated in specific time slots in order to better emulate the execution of advanced ROP attacks.

KPIs and acceptance criteria. The effectiveness of the ASTRID framework depends on detecting a wide range of attack vectors (targeting device and software integrity) in all deployed entities, in the service graph, and on the detection and reaction time. These aspects will be evaluated separately, in order to better identify possible improvements.

Parameter	Target KPI	Acceptance criteria
Number of components in the service graph whose integrity is monitored by the Security Handler	100%	100%
Amount of integrity attacks on components in the service graph	80% (with integrity models and whitelists) 60% (advanced control-flow integrity mechanism)	>60% >40%
Amount of traffic diverted to alternative paths when a component/entity is compromised	80%	>60%
Time to detect compromised software	10 s (with integrity models) 1 min (with advanced control-flow attestation)	< 1 min < 5 min
Time to trigger reaction (based on the result of the attestation process)	30 s	< 2 min
Time to replace a single virtual function	1 min	< 3 min
Time to re-deploy an entire service	5 min	< 10 min

5.4 Illegal Activities

Privacy is an undeniable right for citizens, but this must not be an excuse to hide illegal activities. End-to-end encryption of data is necessary for confidentiality, especially when crossing public and untrusted infrastructures, but this must not become a resource for bypassing investigations from public authorities. Interception of communications and access to private data need to be strictly regulated, but should anyway be possible, conditioned to lawful authorization. The integration with the orchestration process in the ASTRID framework enables the modification of the service to support legal interception or inspection of data at rest, only for an authorized and limited time period, so to not create vulnerabilities and weakness in the overall service.

Description. An authorized user that belongs to the service provider's organization activates a "legal interception" feature from the ASTRID Security Dashboard, which installs a specific policy in the run-time environment. When a communication session is established between two users, the event is



notified to the Security Controller that, if the session meets the conditions in the interception policies (e.g., identity of one or both peers), duplicates the traffic exchanged between the peers and sends it to a specific mediation gateway that connects the service provider to the relevant Law Enforcement Agency. To save resources, interception gateway, i.e. the virtual device that forwards intercepted traffic towards the mediation gateway is automatically deployed before the first interception begins and removed just after the termination of the last active session. Interception gateways tunnel and encrypt traffic to preserve confidentiality. Mediation and interception gateways authenticate each other to ensure the correct delivery of intercepted traffic. ASTRID generates a notification towards the service provider to notify the establishment of an intercepted communication.

Conditions. No specific conditions have been currently identified for this validation scenario.

KPIs and acceptance criteria. The performance of this scenario is mostly related to the time to activate the legal interception. On-the-fly activation avoids the need to run additional resources for a long time, hence making this security service cost-effective.

Parameter	Target KPI	Acceptance criteria
Time to deploy the interception gateway	1 min	1 min
Time to change the forwarding rules in the service	20 s	1 min
Time to trigger interception	3 s	10 s

5.5 Forensics

Many network attacks are carried out by botnets and compromised devices, so it is usually difficult to identify the culprit and prosecute him. In some cases, there may have been guilt, negligence or carelessness from internal users, which are more likely to identify. Moreover, even if the culprit cannot be identified, it may be useful to demonstrate that all best practices and reasonable countermeasures were taken. During an attack, the main concern is to stop or mitigate it, so there is usually a lack of time to deeply investigate the matter and identify the cause and responsible. Keeping records of logs, packets, events, system calls, memory dumps, and other relevant information, helps perform offline analysis. However, the larger the amount of data the larger the storage requirements and the overhead of the collection process. The ASTRID framework aims to provide a good balance between the opposite needs of fine granularity and small resource consumption, by adapting the verbosity of security information to the actual context. For example, aggregated statistics on network flows may be collected every 30 s/1 m, whereas more detailed information (source address, packet's content) may be collected as soon as an alarm is triggered by any detection algorithm. In the same way, memory dumps might only be performed occasionally, and with higher frequency in case of anomalies. It will support offline investigation of the attack, in order to find the origin and identification patterns. An additional consideration concerns the format of the data, where they are going to be used as evidence with legal validity. The usage of timestamps, the identification of the origin, the storage in dedicated infrastructures, and other technical and/or procedural requirements may be necessary to fulfill the regulation.

Description. The Service Provider sets the per-service policies for forensics data collection by exploiting Security Provider's facilities. A policy defines the minimal set of data and information to be registered, the time this information is kept when no anomalies occur (1 week, 1 month, 6 months, roughly depending on the threat landscape the service is addressing), the additional measurements and data to be collected in case of specific attacks and generic anomalies. As soon as an alert is triggered, the forensic policy increases the verbosity, by re-programming local security hooks through the Context



Broker; this also extends the persistence of data in the historical database. When the attack has been stopped or the investigation has terminated, the Security Provider resets the policy, so that it reverts to its basic behaviour.

Conditions. This scenario is applied in combination with one of the demonstration scenarios described in Sections 5.2-5.4.

KPIs and acceptance criteria. This scenario is mostly conceived for demonstration rather than performance evaluation. It will extend one of the previous scenarios to show that the collected information can be used to investigate the threat/attack and that it is compliant with existing regulations. The collected information and the knowledge of attack will be used to assess how effectively the attack was detected and mitigated. The only meaningful performance measure will be the time to reprogram the local monitoring hooks:

Parameter	Target KPI	Acceptance criteria
Time to change/update monitoring and inspection rules	30 s	2 min



References

- [1] ASTRID Project. Deliverable D1.1 – State of the Art, Project Concept and Requirements, v. 1.0, February 2019.
[Online] Available: <https://private.astrid-project.eu/Documents/PublicDownload/17>
- [2] ASTRID Project. Deliverable D1.2 – ASTRID Architecture, v. 1.0, March 2019.
[Online] Available: <https://private.astrid-project.eu/Documents/PublicDownload/31>
- [3] OOR – Open Overlay Router. Website: <https://openoverlayrouter.org>
- [4] Istio - Connect, secure, control, and observe services. Website: <https://istio.io>
- [5] OVS – Open vSwitch, Production Quality, Multilayer Open Virtual Switch.
Website: <https://www.openvswitch.org/>
- [6] Private Campus Networks, February 2019.
Website: <https://www.adlittle.de/en/insights/viewpoints/private-campus-networks>
- [7] The global IoT market opportunity will reach USD4.3 trillion by 2024.
Website: <https://machinaresearch.com/news/the-global-iot-market-opportunity-will-reach-usd43-trillion-by-2024/>
- [8] Smith, D., “No stopping Johannesburg's traffic light thieves”, January 2011,
Website: <http://www.guardian.co.uk/world/2011/jan/06/johannesburg-traffic-light-thieves-sim>
- [9] Fox-Brewster, T., “How Hacked Cameras Are Helping Launch The Biggest Attacks The Internet Has Ever Seen”, September 2016.
Website: <https://www.forbes.com/sites/thomasbrewster/2016/09/25/brian-krebs-overwatch-ovh-smashed-by-largest-ddos-attacks-ever/>
- [10] Securing Internet of Things (IoT) with AWS, April 2019.
[Online] Available: https://d1.awsstatic.com/whitepapers/Security/Securing_IoT_with_AWS.pdf
- [11] ENISA. Towards secure convergence of Cloud and IoT, September 2018.
[Online] Available: <https://www.enisa.europa.eu/publications/towards-secure-convergence-of-cloud-and-iot>