D3.1	
Version	1.0
Author	ATOS
Dissemination	PU
Date	29-06-2018
Status	FINAL

D3.1 ElasTest Platform Cloud Modules v.1

Project acronym	ELASTEST
Project title	ElasTest: an elastic platform for testing complex distributed large software systems
Project duration	01-01-2017 to 31-12-2019
Project type	H2020-ICT-2016-1. Software Technologies
Project reference	731535
Project website	http://elastest.eu/
Work package	WP3
WP leader	ATOS
Deliverable nature	Report
Lead editor	Enric Pages
Planned delivery date	30-06-2108
Actual delivery date	29-06-2108
Keywords	Open source software, cloud computing, software engineering, operating systems, computer languages, software design & development

Funded by the European Union

License

Contributors

Name	Affiliation
Enric Pages	ATOS
David Rojo	ATOS
Michael Pauls	TUB
Piyuhs Harsh	ZHAW
Andy Edmonds	ZHAW
Micael Gallego	URJC
Nick Stavros	RELATIONAL

Version history

Version	Date	Author(s)	Description of changes
0.1	02/05/2018	E.Pages	ToC
0.2	11/05/2018	ALL	Initial contributions
0.3	30/05/2018	M. Pauls	Extended EPM parts
0.3	30/05/2018	A. Edmonds	ESM contributions
0.3	30/05/2018	P. Harsh	EPM contributions
0.3	30/05/2018	D. Rojo	EIM contributions
0.4	30/05/2018	M. Gallego	TORM scope review
0.4	30/05/2018	E. Pages	Integration
0.5	01/06/2018	All	Review
0.6	01/06/2018	E. Pages	Initial version Quality Review Process
0.7	06/06/2018	M. Gallego, M. Pauls, A. Edmonds	Added content to “Challenges to Overcome” section
0.8	14/06/2018	P. Harsh	Clarified and expanded the role of EMP supporting various challenges identified within this document.
0.8	14/06/2018	N. Stavros	EDM contributions
0.9	20/06/2018	E. Pages	Integrated version addressing comments
1.0	27/06/2018	E. Pages	Final version ready for submission

Table of contents

List of Figures

List of Tables

Glossary of acronyms

Acronym	Description
CI (Continuous Integration)	This refers to the software development practice with that name.
FOSS (Free Open Source Software)	This refers to software released under open source licenses.
IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service)	This refers to different models of exposing cloud capabilities and services to third parties.
Instrumentation	This refers to extending the interface exposed by a software system for achieving enhanced controllability and observability
QoS (Quality of Service) and QoE (Quality of Experience)	In this proposal, QoS and QoE refer to nonfunctional attributes of systems. QoS is related to objective quality metrics such as latency or packet loss. QoE is related to the subjective quality perception of users. In ElasTest, QoS and QoE are particularly important for the characterization of multimedia systems and applications through custom metrics.
SiL (Systems in the Large)	A SiL is a large distributed system exposing applications and services involving complex architectures on highly interconnected and heterogeneous environments. SiLs are typically created interconnecting, scaling and orchestrating different SiS. For example, a complex microservice-architected system deployed in a cloud environment and providing a service with elastic scalability is considered a SiL.
SiS (Systems in the Small)	SiS are systems basing on monolithic (i.e. non distributed) architectures. For us, a SiS can be seen as a component that provides a specific functional capability to a larger system.
SuT (Software under Test)	This refers to the software that a test is validating. In this project, SuT typically refers to a SiL that is under validation.
TO (Test Orchestration)	The term orchestration typically refers to test orchestration understood as a technique for executing tests in coordination. This should not be confused with cloud orchestration, which is a completely different concept related to the orchestration of systems in a cloud environment.
TORM (Test Orchestration	Is an ElasTest functional set of components that abstracts

and Recommendation Manager)	and exposes to testers the capabilities of the ElasTest orchestration and recommendation engines.
TJob (Testing Job)	We define a TJob as a monolithic (i.e. single process) program devoted to validating some specific attribute of a system. Current Continuous Integration tools are designed for automating the execution of TJobs. TJobs may have different flavors such as unit tests, which validate a specific function of a SiS, or integration and system tests, which may validate properties on a SiL as a whole.
TiL (Test in the Large)	A TiL refers to a set of tests that execute in coordination and that are suitable for validating complex functional and/or non-functional properties of a SiL on realistic operational conditions. We understand that a TiL can be created by orchestrating the execution of several TJob.
ICT	Information and Communication Technology
IT	Information Technology
WP	Work Package
FMC	Fundamental Model Concept
ETM	ElasTest Test Manager
EPM	ElasTest Platform Manager
EMP	ElasTest Monitoring Platform
ESM	ElasTest Service Manager
EIM	ElasTest Instrumentation Manager
EDM	ElasTest Data Manager
TSS	Test Support Service
EUS	ElasTest User Impersonation Service
ESS	ElasTest Security Service
ECE	ElasTest Cost Engine
PoP	Point of Presence
REST	Representational State Transfer
VDU	Virtual Deployment Unit
AWS	Amazon Web Services
AAA	Authentication, Authorization, Accounting
TOSCA	Topology and Orchestration Specification for Cloud Applications
API	Application Programming Interface
SDK	Software Development Kit

SSH	Secure Shell
CPU	Central Processing Unit
R&D	Research and Development
OSBA	Open Service Broker API
SLA	Service Level Agreement
DoA	Description of Actions
UI	User Interface
GUI	Graphical User Interface
VM	Virtual Machine
KVM	Kernel-based Virtual Machine
JDK	Java Development Kit
KPI	Key Performance Indicator
R	Release
MS	Milestone

1 Executive summary

ElasTest is a cloud platform designed for helping developers to test and validate large software systems while maintaining compatibility with current continuous integration practices and tools. ElasTest enables developers to test large software systems through complex test suites created by orchestrating simple testing units (so-called TJobs).

ElasTest platform is Free Open Source Software and a community of users and contributors is being created, who can help transforming ElasTest into a worldwide reference in the area of large software systems testing and guaranteeing the long term platform sustainability.

The ElasTest platform is designed as a Service Oriented Infrastructure (SOI) where each of the modules constitutes a fine-grained SOA (micro-service). The software modules implemented within "WP3 Cloud Components" have the objective of creating all cloud components and mechanisms required by the ElasTest platform. These components are split into different categories; on one hand we can find the *cloud components* for the ElasTest platform which offers management capabilities at the level of computational resource as well as manages the lifecycle of the cloud based services deployed on top of the aforementioned resources. On the other hand ElasTest offers *Instrumentation Components* which actuates at application level offering management capabilities over the *Software under Test* (SuT).

The content of this report is focused on the specification, design and implementation of the intermediate version of the *ElasTest Cloud Components;* the work carried out in this WP is going to be reported within two iterations: "*D.3.1 ElasTest Platform cloud modules v1 [6]*" is going to be delivered in month 18 as the intermediate version of the software modules together with the accompanying documentation of this version, "D.3.2. ElasTest platform cloud modules v2 [12]" will be submitted on M36 including the final software artifacts and updated documentation of the platform modules.

2 Introduction

2.1 Overview and Objectives

This report presents the software artifacts implemented in the scope of the WP3 during the first period of the project until M18. The Platform modules covered in this report are the Platform Manager (EPM), the Service Manager (ESM), the Instrumentation Agents (EIA), and the Instrumentation Manager (EIM). The work carried out within WP3 has the objective of creating all cloud components and subsystems required by the project. These components are split into two main categories. The Cloud Components for the ElasTest platform which are executed as part of ElasTest and the Instrumentation components, these components can be executed out of ElasTest and as part of the SuT. Additionally, this report also presents the Data Manager (EDM) used by different modules across all technical work packages.

2.2 Structure of the Document

The outline of this document is as follows: First section introduces the document and its objectives. The second chapter presents the ElasTest Cloud modules describing how they are categorised and its overall roadmap. The next sections describes the enablers for managing the platform in a target cloud provider (Sec 4), the mechanism and interfaces offered for managing the on-demand cloud based services within ElasTest (Sec 5), as well as the mechanisms used to instrument the target applications under evaluation (Sec 6). In addition, the service that offers data management capabilities to the components of the platform is presented (Sec 7). Finally the last section includes the conclusion (Sec 8).

2.3 Target Audiences

The primary targets of the document are internal ElasTest technicians from WP3 to WP6 involved in the prototyping and implementation of the platform. In addition, this document is targeting technical personnel interested in testing as well as QA managers interested in adopt our solution.

3 ElasTest Cloud Modules

3.1 Rationale

New advances in ICT technology influence the way software is developed and tested, the proliferation of large scale applications targeting thousands of users that can be connected concurrently and expect real time interactions; makes the testing strategy a crucial aspect for the release management process of the applications.

Nowadays cloud technologies are creating advantages for organizations that adopt it such as: speed, agility, scalability, accessibility and flexibility; therefore ElasTest aims to extend the adoption of the aforementioned benefits offered by the cloud to testers through the creation of a cloud platform (ElasTest Platform) designed for helping to validate large software systems that require complex test suites and validation processes.

Since the irruption of the cloud computing (together with the virtualization era) as a disruptive technology, the increased use of the cloud introduced new business opportunities and challenges during the last years allowing developers to apply more easily the principles of mass production into the IT world. The current panorama reveals that a whole range of IT functions can be thought of as commodity services.

The ElasTest cloud components described within this report are in charge of the management and monitoring of the resources that the platform needs to operate; as well as of the lifecycle management associated to the on-demand testing support services catalogue which can be requested by the ElasTest Platform user dynamically. In addition to the cloud components in charge of the platform management, the report also includes other kind of cloud based component not targeting the platform itself but offering management capabilities over the software system under evaluation.

3.2 Categories

The different categories identified have a direct relationship with the tasks described within the "WP3 Cloud components". Task 3.1 implements the enablers for the platform components to be deployed in a target cloud being able as well to monitor its usage recovering in seamless way information related to the runtime execution of the platform. Task 3.2 implements the appropriate mechanism enabling the lifecycle management of the Test Support Services catalogue offered by ElasTest. Finally, Tasks 3.3 & 3.4 are devoted to the instrumentation capabilities offered over the software under evaluation.

As it has been introduced in the previous paragraph, different categories have been considered:

- Software modules for managing the computational resources of the platform.
- Software modules for managing the cloud based services offered by the platform.
- Software modules for managing the applications under test.

3.3 Roadmap

ElasTest uses an Agile Management methodology, which is suitable for innovation management. This methodology has been designed for transforming ideas into profitable products. For this, it focuses on learning and discovering how to fit a technology into the market instead on how to carry out the technological developments themselves.

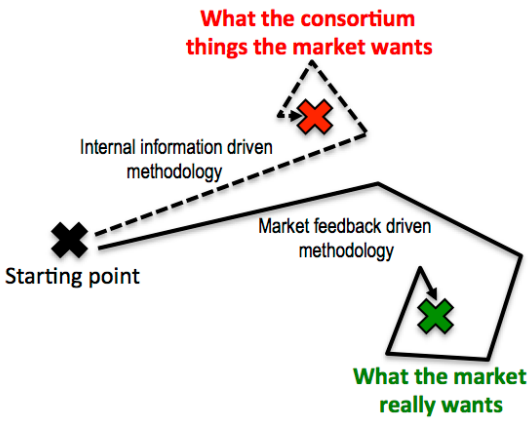

Figure 1. ElasTest Agile Management Methodology

The methodology is based on a continuous feedback loop repeated cyclically every four months aligned with the ElasTest software releases, according to ElasTest initial planning nine releases will be generated during the project duration. The content of this report covers the developments performed up to R4 where the first integrated version of the software components is delivered.

The methodology used for the specification and design phases as well as for the development/testing/release phases have been elaborated in the scope of "Task 2.2. Agile conception based on end-user feedback", further details about the methodology itself will be described in the public report "D.2.3. ElasTest requirements, use-cases and architecture v1 [5]"; where the steps followed by ElasTest technical teams are further described.

The figure below depicts the alignment between the project milestones and the software component releases.

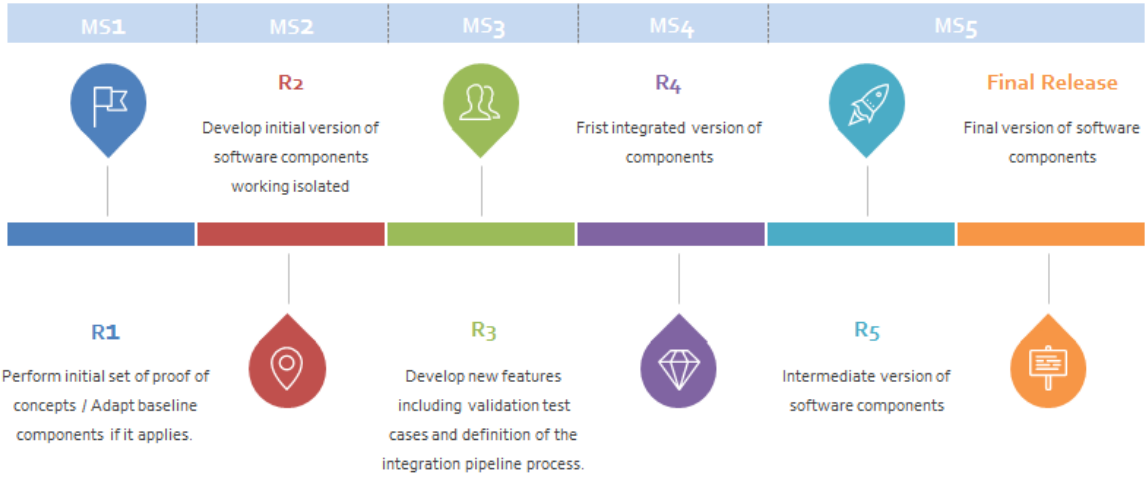

Figure 2. WP3 Cloud Components Roadmap

3.4 Challenges to Overcome

ElasTest is a platform designed to facilitate the build, execution and reporting of end-to-end tests of complex distributed applications. These types of applications present some properties like elasticity and fault tolerance that need to be tested with end-to-end tests. To execute these complex distributed applications and scalable tests, enabling resources and supporting services are needed. The primary reason that such elements must be provided is to remove the tester from the responsibility of having to manage these resources and services themselves and in doing so allow them to focus on their core business, writing complete tests that validate the SuT.

Not only resources and services should be provided for TJobs, additional cloud components must also be provided in order to allow ElasTest deploy and execute a SuT on the behalf of the tester and also deploy and execute the components required to run the ElasTest platform itself. In summary WP3's main goals are:

- Provide resources and services to execute TJobs
- Provide resources to deploy and execute SuTs
- Provide resources and components that support the complete ElasTest platform
- Provide necessary insights into the current and past state of ElasTest core components in order to facilitate stable operation of the platform itself

The key aim and contribution of WP3 to ElasTest is to provide the enabling facilities required by the Elastest Tests Manager (ETM), the main component of the TORM, to carry out its task of orchestration and executing tester supplied TJobs. As such it can be thought of as the enabling platform for the ETM.

The work in WP3 has to cover these key areas of functionality:

1. Provide the resources on-demand to allow for the execution of TJobs
2. Provide the services on-demand to allow for the support and augmentation of TJob functionality
3. Provide the means to manage all resources and services delivered to the ETM

To overcome these needs, the ElasTest architecture has the following characteristics:

- **Microservices inspired architecture:** ElasTest's architecture has been divided in several decoupled components that communicate via remote protocols. In that way, ElasTest can horizontally scale executing every component in a different computational node when necessary.
- **Decoupled test execution:** To execute a set of tests in ElasTest is necessary to configure a TJob. A TJob is defined with the following information: a) How to obtain and execute the tests; b) How to connect to a SUT already deployed or how to execute the SUT inside ElasTest and c) What support services are necessary to execute the tests. Using the same strategy as with the core components, the tests, SUT (if necessary) and support services (if necessary) are executed in decoupled components that communicate using remote protocols. In that way, every TJob can be executed in a different computational node, favouring the scalability of the platform. Hence, a TJob that needs more computational resources than available in a computational node can be split in several computational nodes.
- **No vendor dependency (lock-in):** ElasTest Platform Manager (EPM) introduces an adapter mechanism which means the adapters use a standardized northbound interface whereas the southbound interface is specific to a certain cloud infrastructure technology. In addition, to provide a standardized way of defining virtual resources, the platform manager supports native TOSCA templates. Further details are covered under the non-functional aspects of the platform.
- **No internal state persistency:** All ElasTest components can be configured to be stateless, except ElasTest Data Manager (EDM). This allows all persistency to be grouped in one specific component, while the rest of the platform is stateless. In addition, EDM via Alluxio allows the usage of external services (such as Amazon S3) for persistent data, in a way that is transparent to the rest of the ElasTest platform. Hence, by moving between different hardware/cloud platforms, the only component that needs to be ported to fit is EDM, or it can be swapped out in favor of local services that offer the same functionality.
- **Test Engines:** ElasTest can be augmented with additional components called Test engines (TE). These components are executed as decoupled components and core components can communicate with them using remote protocols. This leads to advantages mentioned to the other parts of the platform.
- **Test Support Services:** Test Support Services (TSSs) are services used by tests via TJobs. They augment the capability of a test by providing some specific features. The TSSs are not covered within this document but they are mentioned here as the ESM covered within this report is the component who manages the TSS lifecycle as well as offers them on-demand. For further information specific to the TSS, please refer to "D.5.1 ElasTest Test Support Services v1" [9].

3.4.1 ElasTest Functional Components/Services

As it can be seen, ElasTest platform is composed by several decoupled components that communicate using remote protocols. This characteristic allows the platform to be split across several physical nodes if the resources needed are not available in a

single node. Hence, we can consider that ElasTest is scalable to take advantage of cloud native design and on-demand use of resources and service to grow and shrink according to load. Also, some of the components are executed on demand and this gives elasticity to the platform. Concretely, tests engines are executed only when they are used. In the same sense, TJob's components are executed also on demand.

The core components of ElasTest are:

- ElasTest Tests Manager (ETM)
- ElasTest Services Manager (ESM)
- ElasTest Platform Manager (EPM)
- ElasTest Monitoring Platform (EMP)
- ElasTest Instrumentation Manage (EIM)
- ElasTest Data Manager (EDM)

The **ElasTest Platform Manager (EPM)** is the base component in charge of executing ElasTest components in several underlying platforms, abstracting ETM (the brain of ElasTest) of this management. Also, as several cloud resource management platforms are supported, ElasTest can be deployed in any of them without any change. To offer this abstraction of the underlying platform, EPM requires that components are packaged as docker containers. This format have been selected because is a lightweight standardized format with a standard distribution mechanism. Also, this format is widely supported in the industry. In addition, when a component is composed by several containers, docker compose descriptor file can be used to describe the component. In the current version, EPM can be executed in a single machine with docker daemon installed. This node is used to execute ElasTest core components. Other nodes can be added dynamically to EPM to execute dynamic ElasTest components like TJob components or test engines. In the future versions, Kubernetes, AWS and OpenStack platforms will be supported natively in EPM to support the real elasticity of the platform.

It is very important to monitor how computational resources are been used to avoid or adapt to overload of the system, given that TJobs are executed dynamically on-demand. If the system is above some load threshold, new TJobs can be queued until resources are available or ask to underlying platform for more nodes to execute components. The **ElasTest Monitoring Platform (EMP)** is the component in charge of monitoring ElasTest platform. This component works closely with EPM to allow the mentioned autoscaling features. Also, EMP shows system metrics it gathers from the underlying platform to the user. This is especially important for administrative tasks. However, not all platforms allow the autoscaling feature, then, monitoring information is being used to control the fixed resources available.

Through the **ElasTest Service Manager (ESM)**, ElasTest is able to provide on-demand test support services (TSSs) to testers (as defined in their TJobs) to make easier to implement complex tests and delegate non-core functionality to an internal or external service provider. For example, some of the services provided by default in ElasTest like the ElasTest User Impersonation Service (EUS), provides browsers on demand. Other TSS available is the ElasTest Security Service (ESS) that provides dynamic security tools to testers. These tools can be managed from test code using a

remote protocol. In addition to the tests included by default, ElasTest allows users to create and install new services. All of this is done without vendor lock-in by using the Open Service Broker API standard. ElasTest Service Manager (ESM) is the component that manages the register and management of TSS. It uses EPM to instantiate new services on demand when are required by ETM. ETM will ask to ESM for a new service instance if this TSS is defined in the TJob to be used by the tests. ESM also works closely with EMP which keeps tracks of health status of support services created and managed by ESM. EMP has proactive alarming capability which is the key feature of interest for ESM.

While the aforementioned components deals with the platform resources, the **ElasTest Instrumentation Manager (EIM)**, controls and orchestrates the monitoring and controllability agents which are deployed when an external SUT is tested. In that way, tester doesn't need to manually configure these agents to obtain relevant monitoring information about SUT. Using EIM, user will be able to instrument external SUT to simulate real behaviour simulating CPU load or network issues.

In addition, **ElasTest Data Manager (EDM)** provides persistence services to the platform. It is used by several components as data management service. The ElasTest Data Manager was built to separate the persistence layer of ElasTest from the rest of the platform.

3.4.2 ElasTest Non-functional Aspects

Elasticity

The ElasTest services and resources must be provided on-demand, when and only when the tester actually needs them. By having the capability, the overall cost to run a test suite against a SuT is reduced when compared to having resources and services running all the time. For example, if ElasTest is used to test the elasticity of a SUT like a video conference system, the tests should request hundreds or thousands of simultaneous browsers simulating users connecting to the platform. Then, ElasTest should execute all these browsers.

Further, by being able to request resources and services on-demand enables the capability of dynamically scaling up (or down) the set of resources and/or services assigned to a particular test suite at any point in time. In doing so, the platform is amenable to elasticity. However, before having this capability the components need to be designed in such a way to be scalable. Furthermore, monitoring and timely alerting is a key prerequisite for effective elastic control; EMP objectives already cover this element to support elastic control and management of underlying resources. EMP allows instrumented metrics to be directly sent to it via different language specific libraries. This capability can be used in conjunction with a more fine grained alert condition creation within EMP wherein the destination of the alert is the relevant application endpoint itself, it is very much possible to achieve a parallel elastic control mechanism that is self-triggered by the application and not just managed by the EPM.

Authentication, Authorization and Accounting (AAA)

ElasTest needs to provide the means for users to be identified uniquely so that specific resources, services can be associated with them and ultimately allow for the charging

of those services and resources to the specific user. ElasTest also needs to provide this from an audit and security perspective: who did what where, when and how.

To provide this an AAA (Authentication, Authorisation and Accounting) service/ component is typically used. With an AAA element as part of ElasTest and used down through the full stack (from user, through ETM and onto ESM and EPM), multi-tenancy, isolation and per user billing can be enabled. These characteristics are fundamental to cloud computing (See NIST definition[1], publication 800-145) of which ElasTest is founded on.

AAA is an ElasTest platform service to support many services and components in ElasTest. As such the proposal is to include AAA as part of WP3. No developments upon AAA topics will be carried out or upon Keystone (where necessary). In order to provide AAA within ElasTest the proposal is to use OpenStack's Keystone project[2] to enable AAA. Keystone will runs as an additional service within the ElasTest platform and so will be the responsibility of the ElasTest Toolkit to start the service.

In the basic scenario the onus is upon the User to acquire the token from the Keystone service. This can be accomplished either via API[3] or using the Keystone command line client[4] (which is now part of the main openstack command line client[5]).

With its basic usage, access to any Keystone mediated resource given by having a valid Keystone token relayed in HTTP headers. The header name used is `X-Auth-Token` and its value is the token issued by the keystone service.

It should be noted that keystone integration per component must be provided in a configurable way, in order to use Keystone a service needs 2 things:

1. **A running instance of keystone**: to do this you can use the following docker project[6] to bring up a keystone instance. You can review the README[7] for basic usage and refer to the OpenStack Keystone[8] project for further detail and information.
2. **Client code that accesses and uses keystone**: how this is accomplished is rather specific to the language and frameworks you use to implement your service/component.

Vendor Lock-in

A vendor lock-in limits the user to the usage of a certain solution or technology, and, hence, it reduces also the capabilities of the solution itself only able to cover a limited set of scenarios and use cases. As a comprehensive testing platform for highly distributed applications, one of the main goals is to avoid a vendor lock-in at the ElasTest platform level to let the freedom of choice for a certain cloud virtualization

[1] NIST Definition, https://csrc.nist.gov/publications/detail/sp/800-145/final
[2] OpenStack Keystone, https://docs.openstack.org/keystone/latest/
[3] OpenStack API, https://developer.openstack.org/api-ref/identity/v3/index.html
[4] KeyStone CLI, https://docs.openstack.org/mitaka/cli-reference/openstack.html
[5] KeyStone CLI-reference, https://docs.openstack.org/mitaka/cli-reference/openstack.html
[6] Keystone docker project, https://github.com/dizz/dock-os-keystone
[7] README file, https://github.com/dizz/dock-os-keystone/blob/master/README.md
[8] OpenStack Keystone, https://docs.openstack.org/keystone/latest/

infrastructure up to the users and their requirements. A proper approach had to be developed in order to overcome this issue which is following a plug-and-play approach.

There are two services that could be limited by vendor locking: the ElasTest Service Manager (ESM) and the ElasTest Platform Manager (EPM).

The **ESM**, avoids the issue of lock-in by adopting a widely adopted API, the Open Service Broker API[9]. Behind this API, the implementation of the ESM, is implemented such that pluggable backends are used for storage (the DB in the case of the ESM; supports simple in-memory, MongoDB and MySQL) and resource acquisition (currently local docker engines and the EPM). Should another DB or resource acquisition software be required, this is achieved by implementing the software interface for either DB[10] or resource acquisition[11] modules.

The **EPM,** introduces an adapter mechanism which means the adapters use a standardized northbound interface whereas the southbound interface is specific to a certain cloud infrastructure technology. Based on the requirements of the project consortium, the northbound interfaces was designed in a way that it allows 1) the definition of the virtual resource requirements following the internal information model of the EPM and 2) cloud infrastructure-specific templates. In this way the EPM can potentially support any type of technology assuming the corresponding adapter is in place. Currently, the focus of the adapter development is aligned with what the project consortium has seen as appropriate (Docker[12], docker-compose[13], Ansible[14], VirtualBox[15]). In the future the need of further adapters will be explored to support, for instance, OpenStack[16], OpenStack Heat[17], AWS[18], or complex orchestration solutions, such as, Aria[19] or OpenBaton[20].

In addition, to provide a standardized way of defining virtual resources, the platform manager supports native TOSCA[21] templates. TOSCA is a domain specific language and portable model for describing cloud applications. The TOSCA model is a widely recognized format and therefore would also provide an easy way for users to transition to ElasTest. The TOSCA Simple Profile for YAML 1.0 describes the way to represent the TOSCA meta-model in a simplified format using YAML[22]. The platform

[9] OpenAPI Initiative https://www.openservicebrokerapi.org
[10] ESM Store, https://github.com/elastest/elastest-service-manager/blob/master/src/adapters/store.py#L51
[11] ESM Resource, https://github.com/elastest/elastest-service-manager/blob/master/src/adapters/resources.py#L39
[12] Docker, https://www.docker.com/
[13] Docker-compose, https://docs.docker.com/compose/
[14] Ansible, https://www.ansible.com/
[15] Oracle VirtualBox, https://www.virtualbox.org/
[16] OpenStack, https://www.openstack.org/
[17] OpenStack Heat, https://wiki.openstack.org/wiki/Heat
[18] Amazon Web Services, https://aws.amazon.com/
[19] Apache Aria, http://ariatosca.incubator.apache.org/
[20] Open Baton, https://openbaton.github.io/
[21] OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA), https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
[22] YAML, http://yaml.org/

provides the option to render TOSCA templates to the internal information model following the TOSCA Simple Profile for YAML 1.0, to provide the option for users familiar with the specification to make use of its generalized model for defining cloud systems.

4 Platform Management and Monitoring

The following section introduces the core components in charge of the management and monitoring of the platform; and provides the details of the requirements, architecture, interfaces and features for each of them.

4.1 ElasTest Platform Manager (EPM)

4.1.1 Introduction

The ElasTest Platform Manager (EPM) implements the enablers for ElasTest components to be deployed in a target cloud.

DoA [1] specifies the following objectives for the EPM:

- To develop the appropriate technologies enabling ElasTest to be deployed in the target cloud environment. For this, the Platform Manager shall need to provide the required cloud orchestration services for the deployment and provisioning of all ElasTest components. This will require the Platform Manager to consume, at its southbound, the APIs exposed by the target cloud infrastructure.
- To create the appropriate technologies enabling the management of the underlying cloud resources on behalf of the TORM and of the rest of ElasTest services. These technologies shall provide the capability of instantiating computing resources, of deploying artifacts on them (e.g. TJob instances, Test Support Service instances, etc.) and of managing their lifecycle. Remark that the autoscaling of computing resources used by ElasTest shall be part of this mechanism.
- To expose, at its northbound, all these capabilities through a comprehensive and coherent API (or directly Software Development Kit) that the TORM and the rest of ElasTest testing services shall consume in runtime for implementing their logic.
- To develop a toolbox enabling the installation and management of all such capabilities in ElasTest.
- To expose a catalogue of Support Services. The Platform Manager will provide this catalogue in order to allow any developer to select the appropriate Support Services required in the experiment.

The ElasTest Platform Manager is the interface between ElasTest components (e.g. TORM, Test Support Services, etc.) and the cloud infrastructure where ElasTest is deployed. Hence, this Platform Manager must abstract the cloud services so that

ElasTest becomes fully agnostic to them and provide this abstraction via Software Development Toolkits (SDK) or REST APIs to the northbound consumers (i.e. the TORM). The ElasTest Platform Manager enabling ElasTest to be deployed and to execute seamlessly in the target cloud infrastructure that the consortium considers as appropriate (e.g. OpenStack, CloudStack[23], Mantl[24], AWS, Docker, etc.).

The EPM provides two options to describe and deploy the virtual resources:

- All-in-one Package Deployment: the package approach is designed to make use of template-dependent technologies such as docker-compose, Ansible or OpenStack Heat. The EPM gets such a template with additional metadata information which is forwarded directly to the target infrastructure to trigger the deployment as a whole.
- Step-by-Step Deployment: this step-by-step approach is designed for technologies such as Docker, OpenStack or AWS where the EPM receives the resources description which is compliant to the data model of the EPM or TOSCA. That information about virtual resources is then translated to individual commands calling the technologies' API.

Both approaches together make the EPM independent to the underlying infrastructure and give the consumer of the EPM the opportunity to use already existing templates or the data model exposed by the EPM. However, in both cases the EPM returns the information in a uniform format following the data model.

To avoid a vendor lock-in situation, the ElasTest Platform Manager introduces an adapter mechanism which means the adapters use a standardized northbound interface whereas the southbound interface is specific to a certain cloud infrastructure technology. Based on the requirements of the project consortium, the northbound interface was designed in a way that it allows 1) the definition of the virtual resource requirements following the internal information model of the EPM and 2) cloud infrastructure-specific templates. In this way the EPM can potentially support any type of technology assuming the corresponding adapter is in place. Currently, the focus of the adapter development is aligned with what the project consortium has seen as appropriate (Docker, docker-compose, Ansible, VirtualBox). In the future the need of further adapters will be explored to support, for instance, OpenStack, OpenStack Heat, AWS, or complex orchestration solutions, such as, Aria or Open Baton. A major challenge in this regard is that all adapters have to provide the same capabilities, such as, runtime management to access instances for certain operations (see *Features* table).

4.1.2 Baseline Concepts and Technologies

The EPM itself is implemented in Java making use of the Spring framework[25]. Data persistency is provided via SQL where by default it uses an in-memory database

[23] Apache CloudStack, https://cloudstack.apache.org/
[24] Mantl, https://www.mantl.com/

[25] Spring Framework, https://spring.io/

(HyperSQL[26]). Nevertheless, other SQL databases (e.g. MySQL[27]) can be easily integrated by changing the configuration inside the main properties file following the spring configuration guide.

The current version of the EPM supports the following virtual infrastructure technologies: Docker, docker-compose and Ansible. Two approaches are supported by the EPM in the meaning of the consumer can either make use of the EPM's data model or TOSCA to describe the deployment scenario or use directly templates of a certain technology. Thanks to the modular approach, other virtualization infrastructures can be easily supported by providing adapters for certain technologies. This adapter mechanism is provided via gRPC which manages the communication between the EPM itself and the corresponding adapter.

The Access, Authorization and Accounting (AAA) system can be activated for the EPM where the integrated system is OpenStack's Keystone.

In addition, the EPM makes indirectly use of several supporting services by configuring the virtual instances for the purpose of log forwarding (e.g. Logstash[28]) or monitoring (e.g. Dockbeat[29]) which are then provided indirectly to other services for further processing, such as, the ElasTest Monitoring Service, ElasTest Monitoring Platform, or the ElasTest Test Manager.

The EPM and all the available adapters are delivered as Docker containers which are available in Docker Hub. In addition, several docker-compose files are provided in the GitHub repositories to start easily the EPM with the additional components and services to ease the deployment and configuration.

4.1.3 Component Design and Architecture

This section gives an architectural overview of the ElasTest Platform Manager. The architecture (see Figure 4) is composed of several components:

[26] HyperSQL, https://spring.io/
[27] Oracle MySQL, https://www.mysql.com/
[28] Logstash, https://www.elastic.co/products/logstash
[29] DockBeat, https://www.elastic.co/blog/dockbeat-a-new-addition-to-the-beats-community

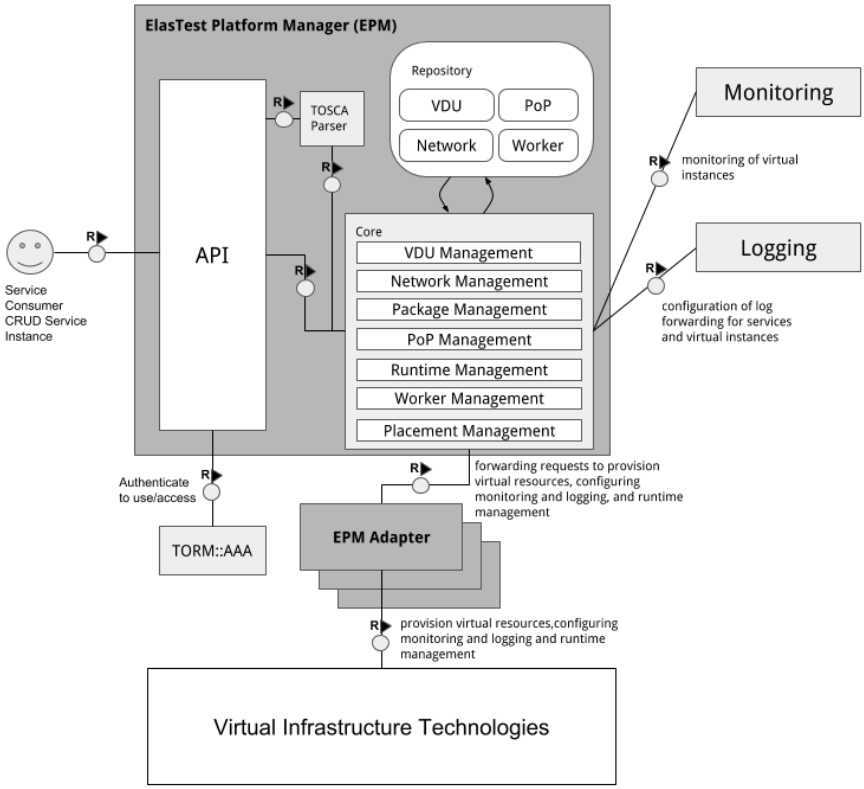

Figure 3. Architectural Overview of EPM

API: The API exposes a ReSTful API in order to allow the consumer (e.g. ETM, ESM) to manage virtual resources in a target cloud environment. It allows to allocate, terminate, update virtual resources (e.g. compute, network) and request information of those as well, execute runtime operations, and register and configure new workers. Moreover, in order to allow a programmatic usage of the EPM, a python and java client are provided that eases the usage of the EPM.

Repository: The repository persists information of managed Workers, VDUs, networks, and PoPs as well. The following gives an overview of what those entities are:

- Worker: A worker is a machine, where the EPM can set up a cloud environment and make it ready to be registered as a Point-of-Presence.
- PoP: A PoP is a Point-of-Presence that defines details of a cloud environment. This includes information about the endpoint, type and access details.
- VDU: A VDU is a Virtual Deployment Unit which reflects an abstraction of virtual compute resources. It contains information about software, network connectivity and the target cloud environment.
- Network: A network reflects the virtualized network resource which provides connectivity between VDUs.

Core: The core consists of several management units and provides the basic management functionality in order to manage PoPs, VDUs and networks. The core has access to the Repository in order to persist and request information of managed entities (PoP, VDU, and Network). In order to issue operations on different types of

cloud environments (Docker, OpenStack, Kubernetes, AWS), the Core component makes use of PoP adapters which allows the Core to interact with the PoP over well-defined interfaces.

- PoP Management: This component handles the PoPs. It is in charge of registering, unregistering and providing information of a requested PoP.
- VDU Management: This component manages virtualized resource related to the compute domain. It allocates compute resources, connects them to networks, receives details of allocated resources and releases resources in the target PoP.
- Network Management: This component manages virtualized resource related to the network domain. It creates and deletes network in the target PoP.
- Runtime Management: This component is responsible for managing runtime operations (e.g. download/upload files, execute commands, etc.) for already allocated virtual ressources.
- Package Management: This component is in charge of handling packages (e.g. docker-compose, ansible, etc.) and forwards it to the corresponding adapters. The packages contain virtualization technology-specific templates.
- Placement Management: This component is in charge of the placement of virtual resources in case no specific PoP is selected where the virtual resources have to be deployed.
- Worker Management: This component takes care of the installation and configuration of new workers added at runtime to the EPM as potential PoPs. Certain scripts are provided which will install and setup the needed artifacts so that the Worker is ready to be used as a PoP.

EPM Adapter: An EPM Adapter provides an abstracted way to interact with any kind of cloud environment. The northbound interface is exposed to the Core and abstracted in such a way, that the Core do not need to take care about the type of the target cloud environment. The southbound interface is dependent on the type of cloud environment under consideration. This allows an easy way to provide any kind of cloud environment by providing an adapter without changing anything in the core. The PoP Adapter takes also care about the configuration of logging and monitoring of the virtualized resources by receiving that information by the Core component.

4.1.3.1 Use Cases & Sequence Diagrams

This section presents three main use cases with the help of sequence diagrams. Those use cases are the Step-by-Step Deployment where the consumer describes the virtual resources to be deployed by the EPM by using the internal data model or TOSCA language, the All-in-one Package deployment where the consumer can reuse existing templates from certain cloud infrastructure technologies (e.g. docker-compose, Ansible) and the Worker registration and configuration where the consumer can add new machines at runtime which can be used for virtual deployments later.

4.1.3.1.1 #1 Step-by-Step Deployment

This scenario depicts the workflow for allocating virtual resources based on the definition using the internal information model, so called resources groups. This

approach follows the assumption that the consumer can define the requirements in a uniformed format so that it is agnostic to the actual cloud environment where virtual resource shall be allocated. This allows the user to use the same definition to be used for various cloud infrastructures. To be aligned with the adapter approach, the EPM will generate a package containing that information with an additional metadata file that will be passed to the corresponding EPM Adapter. The EPM Adapter extracts the required information from the package and initiates the step-by-step deployment starting with setting up the networking before allocating the virtual compute resources. The adapter populates the resource group with deployment information and returns it to the EPM which is then returned to the initial consumer.

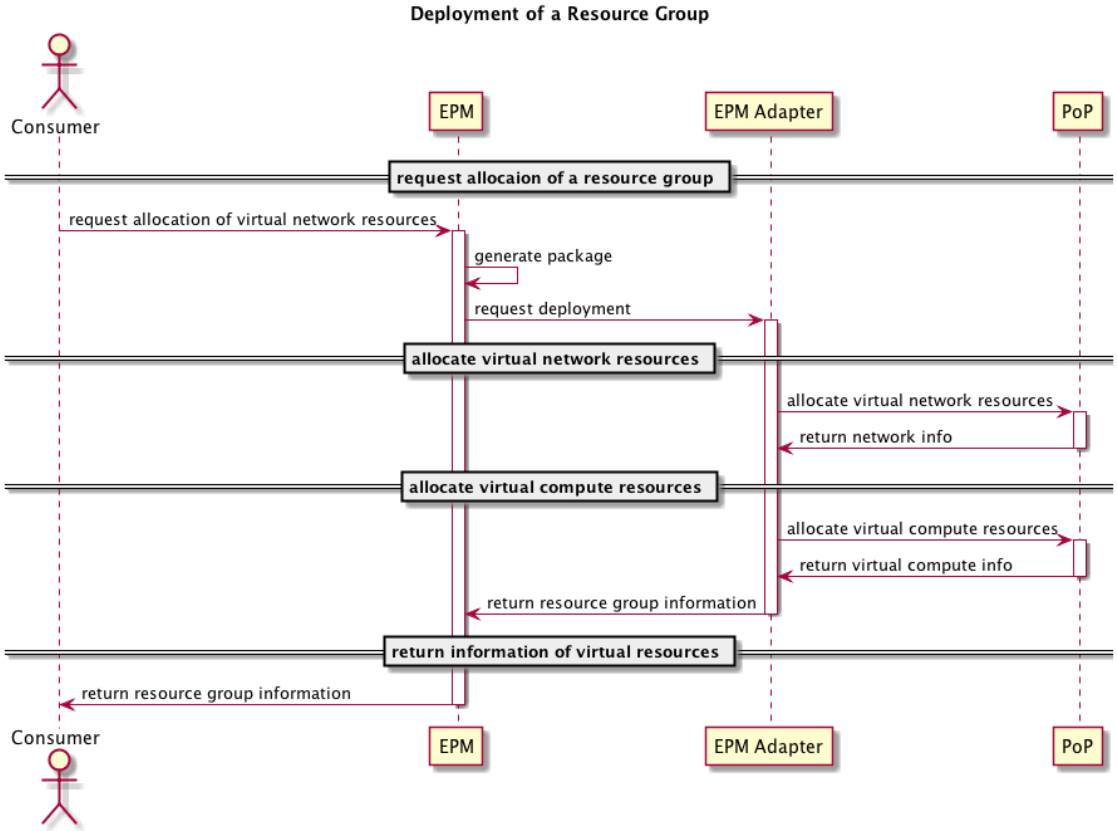

Figure 4. EPM: Deployment of a Resource Group

4.1.3.1.2 #2 All-in-one Package deployment

The sequence diagram below shows the workflow for the All-in-one Package deployment. This was designed in order to give the consumer of the EPM the freedom to use pre-existing templates without translating the requirements to the internal information model of the EPM. The consumer has to generate a package in advance which contains basically a metadata file (containing meta information, such as, the name of the service and the type of PoP) and the actual template (or several files) to be used for the deployment. Once the package is received by the EPM, the EPM will extract required information from the metadata file and forward the package to the corresponding adapter which takes care to trigger the deployment with the actual template file. Once the deployment has finished and the adapter received the

infrastructure-depended information, the EPM adapter translates those to the internal information model of the EPM and returns it to the consumer via the EPM.

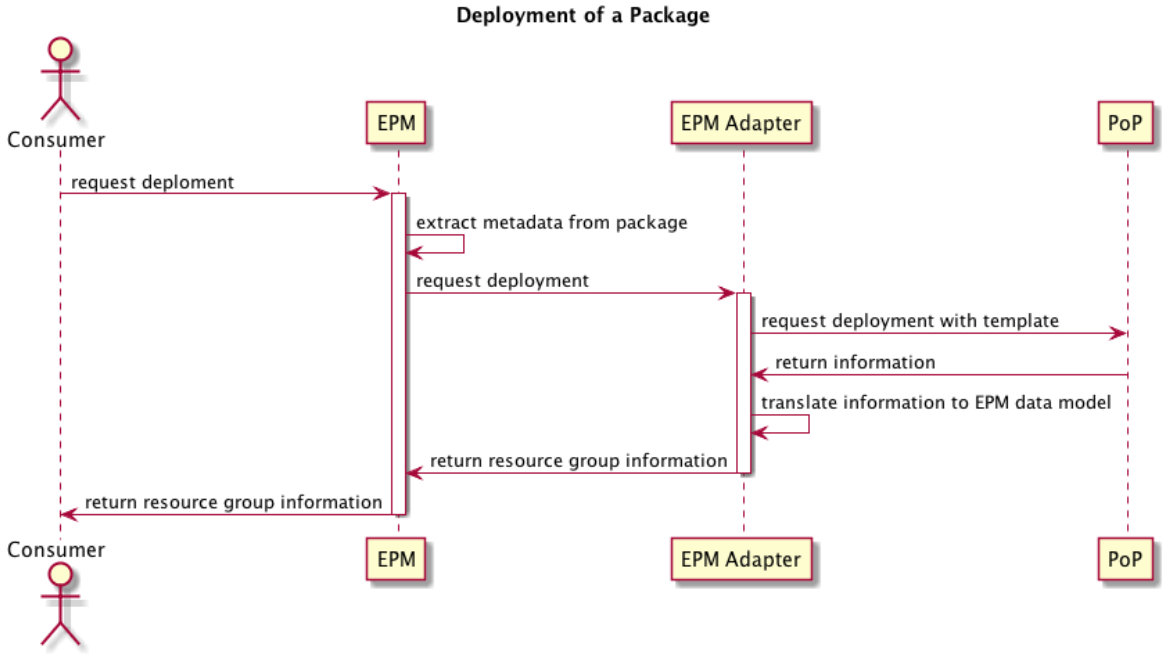

Figure 5. EPM: Deployment of a Package

4.1.3.1.3 #3 Worker registration and configuration

The sequence diagram below depicts the workflow of the worker registration and configuration. This feature has been designed in order to give the consumer the ability to register new machines (physical or virtual) on demand, basically, to provide more computation power if needed. Hence, the user needs to provide a key which allows the EPM to access those machines via SSH in order to install and configure the required artifacts. Once the key is available the user can register a new worker providing the IP so that the EPM can execute the installation and configuration steps. As shown in the sequence diagram, the EPM can optionally configure the monitoring agent to get monitored by the EMP. In the second step the EPM issues certain installation and configuration steps via ssh depending on the defined type (e.g. Docker, docker-compose, Ansible) of the worker. Once the required artifacts are installed, the EPM ensures to have this new worker ready to be used which requires the registration as a new PoP and optionally the configuration of adapters (e.g. for docker-compose). Finally, the consumer gets returned the information of this request.

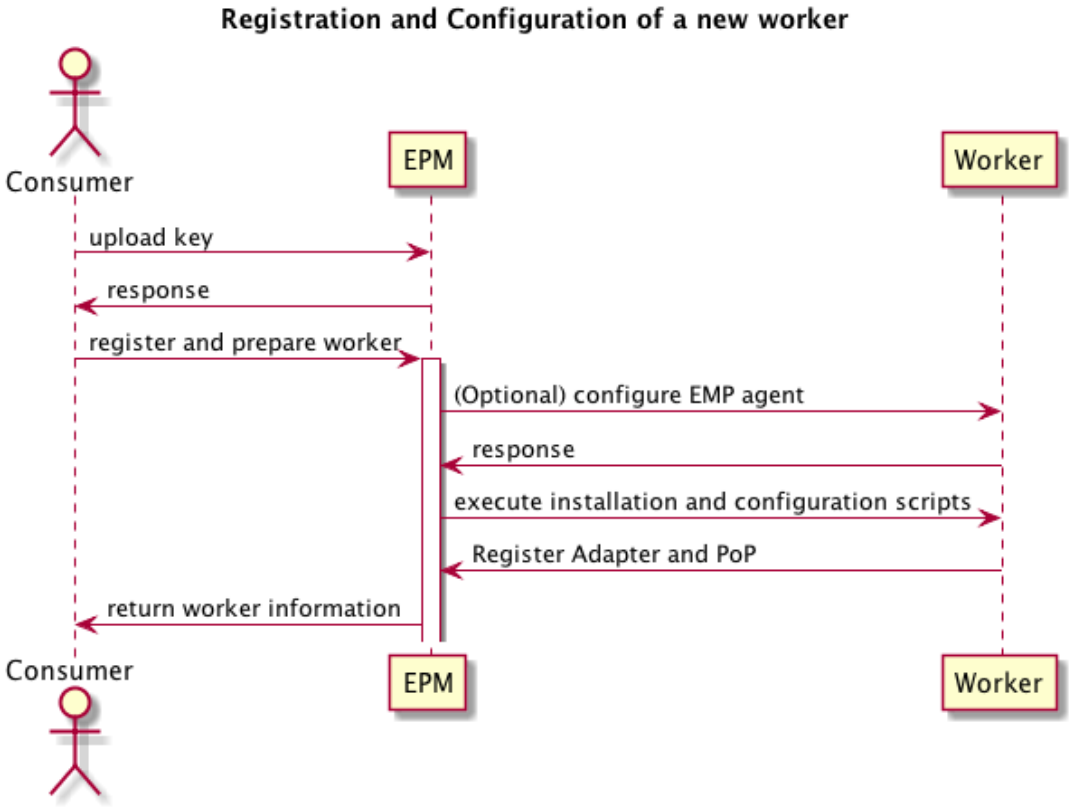

Figure 6. EPM: Registration and Configuration of a new worker

4.1.3.2 Data Model

Figure 8 shows the data model exposed to the consumer of the EPM where those entities can be retrieved and managed via the APIs that are described in the section below.

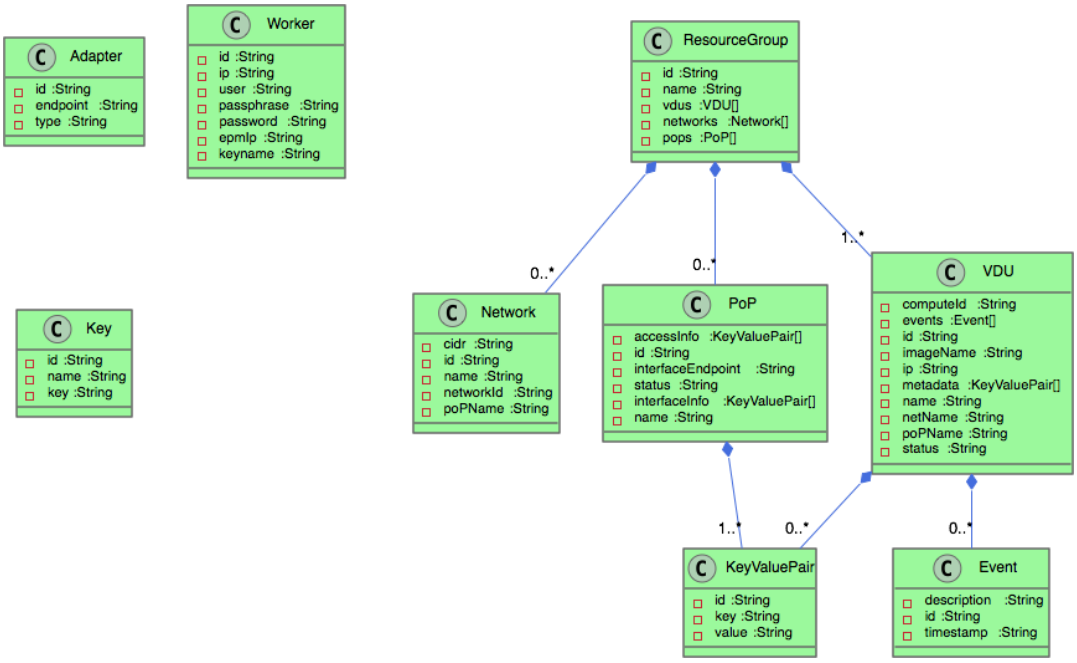

Figure 7. EPM: Data Model

4.1.3.2.1 Adapter

An adapter is the intermediate component between the EPM and the cloud infrastructure technology. Basically, the EPM forwards deployment and management request to the adapter whereas the adapter translates those requests to the cloud technology-dependent commands. It follows a plug-in approach which allows the maintainer of the EPM to plug-in new adapters at any point in time without reconfiguring or even touching the EPM itself. Developing new adapters can also be done without changing the source code of the EPM.

Table 1. EPM: Adapter Data Model

Name	Description	Schema
endpoint required	The endpoint where the Adapter is reachable. Example : `"localhost:50052"`	string
id optional	Identifier for the Adapter.	string
type required	The type of virtualization technology, that the adapter is designed to connect to. Example : `"docker-compose"`	string

4.1.3.2.2 Event

An event contains certain life cycle information of the VDU at a specific time.

Table 2. EPM: Event Data Model

Name	Description	Schema
description required	Example : `"testEvent1"`	string
id optional	Example : `"1234-abcd"`	string
timestamp required	The recorded time of the Event.	string (string)

4.1.3.2.3 Key

A private key for executing commands on a worker.

Table 3. EPM: Key Data Model

Name	Description	Schema
id optional	The identifier of the Key	string
key required	This is the key itself as String.	string

name *required*	The name of the key. This will be used for referencing the Key in a Worker. Example : `"key1"`	string

4.1.3.2.4 KeyValuePair

This entity is a Key-Value pair for storing metadata contained in other entities.

Table 4. EPM: KeyValuePair Data Model

Name	Description	Schema
id *optional*	Example : `"1234-abcd"`	string
key *required*	Example : `"testKey1"`	string
value *required*	Example : `"testValue1"`	string

4.1.3.2.5 Network

This entity defines the network connectivity and details where the VDUs are connected to.

Table 5. EPM: Network Data Model

Name	Description	Schema
cidr *required*	Example : `"192.168.1.1/24"`	string
id *optional*	The identifier of the Network in the EPM. Example : `"1234-abcd"`	string
name *required*	The name of the network, this should correspond to the name of the network in the virtualization technology. Example : `"testNetwork1"`	string
networkId *required*	The id of the Network in the virtualization technology. Example : `"1234-abcd"`	string
poPName *required*	The PoP where the Network was created.	string

4.1.3.2.6 PoP

This entity contains information about the Point-of-Presence (PoP)

Table 6. EPM: PoP Data Model

Name	Description	Schema
accessInfo *required*	Authentication credentials for accessing the PoP. Examples may include those to support different authentication schemes, e.g. OAuth, Token, etc.	<KeyValuePair> array
id *optional*	Identifier of the PoP	string
interfaceEndpoint *required*	Information about the interface endpoint. An example is a URL. Example : `"localhost"`	string
interfaceInfo *required*	Information about the interface(s) to the PoP, including PoP provider type, API version, and protocol type. Example : `"[{&quot;key&quot;:&quot;type&quot;,&quot;value&quot;:&quot;docker&quot;}]"`	<KeyValuePair> array
name *required*	Human-readable identifier of this PoP information element Example : `"testPoPName"`	string
status *optional*	Representing the status of a PoP (INACTIVE, CONFIGURE, ACTIVE)	enum (configure, active, inactive)

4.1.3.2.7 ResourceGroup

A Resource Group defines a bundle of VDUs and virtual networks which belongs together. It includes also the Point-of-Presences (PoP) where the virtual resources have to be allocated.

Table 7. EPM: ResourceGroup Data Model

Name	Description	Schema
id *optional*	The identifier of the Resource Group in the EPM.	string
name *required*	The name of the Resource Group. Example : `"testResourceGroupName1"`	string
networks *optional*	The Networks in the Resource Group.	<Network> array
vdus *required*	The VDUs of which this Resource Group consists of.	<VDU> array

4.1.3.2.8 VDU

A Virtual Deployment Unit (VDU) describes the capabilities of virtualized computing (Containers, VMs) and networking resources.

Table 8. EPM: VDU Data Model

Name	Description	Schema
computeId *required*	The identifier of the deployed VDU in the virtualization technology.	string
events *optional*	A list of events recorded for this VDU.	< Event > array
id *optional*	The identifier of the VDU in the EPM.	string
imageName *required*	The name of the image used for the VDU. Example : `"testImage1"`	string
ip *required*	The IP assigned to the VDU. Example : `"172.0.0.1"`	string
metadata *optional*	More detailed information about the VDU in a Key-Value pair format.	<KeyValuePair> array
name *required*	The name of the VDU. Example : `"testVdu1"`	string
netName *required*	The name of the network to which the VDU is associated with. Example : `"testNetworkName"`	string
poPName *required*	The name of the PoP where the VDU is deployed.	string
status *optional*	The status of the virtualized compute resource.	enum (initializing, initialized, deploying, deployed, running, undeploying, undeployed, error)

4.1.3.2.9 Worker

A worker object for registering a machine where adapters can be deployed.

Table 9. EPM: Worker Data Model

Name	Description	Schema
epmIp *required*	This is the IP where the EPM is reachable for the Worker. This is needed because the Worker has to be able to reach the EPM for registering adapters.	string
id *optional*	Identifier for the Adapter.	string
ip *required*	The IP where the Worker is reachable. The EPM will try to ssh in to the Worker at this IP.	string
keyname *required*	The name of the Key, which the EPM will use for ssh in to the Worker. This refers to the name provided when uploading the Key to the EPM. Example : `"key1"`	string
passphrase *required*	This is the Passphrase of the Key provided for connecting to the Worker.	string
password *optional*	This is the password of the user, which can be left blank if no password is needed.	string
user *required*	This is the user, which the EPM will use when trying to ssh in to the Worker. Example : `"ubuntu"`	string

4.1.4 Roadmap and Features

The following table gives an overview of the main features which shall be satisfied by the EPM.

Table 10. EPM: RoadMap & Features

Feature	Description
Allocation of compute resources	Allocate compute resources in the target cloud environment based on the requirements.
Termination of compute resources	Release compute resources in the target cloud environment
Creation/Deletion of network resources	Create/delete network resources in the target cloud environment
Forwarding logs	Compute resources/cloud environment have to be configured to forward logs of running instances to the appropriate location
Forwarding metrics	Compute resources/cloud environment have to be configured to forward measurement results of running instances to the

	appropriate location
Retrieval of resource information	External entities should be able to request information of the allocated resources
Instance management operations	EPM must be able to execute operations such as executing commands inside the instances and downloading/uploading files so that the consumer of the EPM has full flexibility of accessing and interact with the virtualized instances
Instance lifecycle operations	EPM must be able to execute lifecycle operations such as start/stop, remove instances and retrieving information of the instance at runtime so that the consumer of the EPM has full flexibility of executing lifecycle operations with the virtualized instances for a proper management at runtime
Platform Elasticity	Elasticity must be provided by the EPM so that either other ElasTest components can be scaled dynamically or the virtualized resources requested by other ElasTest components themselves
Management of external machines	EPM must be able to manage external machines which are not deployed by the EPM itself so that the EPM can manage those machines in order to integrate them as workers into the ElasTest platform

4.1.4.1 EPM adapters

Hereafter it is given an overview of available adapters.

- Docker Adapter: The Docker adapter is used to launch Docker containers. To describe the Docker instances the EPM and the Docker Adapter use an internal model called a Resource Group. The Resource group describes Docker containers and the networks connecting them. The Resource Group is packaged together with a metadata file which provides specific information relevant for the EPM and the Virtual Infrastructure which in this case is Docker. The Docker Adapter connects to Docker through the remote API, which means that it has a One-to-Many relationship with Docker. The Docker SDK also is used to execute runtime operations.
- Docker-compose Adapter: The docker-compose Adapter is used to launch docker-compose files. The docker-compose file is passed along with an additional Metadata file in a package. Due to the fact, that Docker-Compose does not expose an external API, the Adapter must be launched in the same Machine, where also docker-compose is installed. This means that the Adapter has a One-to-One relationship with the virtualization technology. The runtime operations are executed using the Docker SDK.
- Ansible Adapter: The Ansible Adapter is used to launch OpenStack instances using Ansible. An Ansible “play” file is passed along with an additional Metadata file in a package. The adapter then uses the Ansible SDK to launch

the “play”. The Ansible Adapter can connect to OpenStack instances remotely, which means that it has an One-to-Many relationship with the virtualization technology. The runtime operations are executed using SSH.

4.1.4.2 Software Development Kits (SDKs)

- Java SDK: The Java SDK makes it possible for integration with the EPM in Java. It supports all the above mentioned API calls.
- Python SDK: The Python SDK makes it possible for integration with the EPM in Python. It supports all the above mentioned API calls and is also available in the Python Package Index (pypi).

4.1.4.3 Roadmap

The overall goal in the upcoming release can be splitted in 3 areas:

- Extending platform support for other cloud infrastructure technologies: Based on the requirements of other ElasTest components, use cases and the demonstrators, the EPM is going to extend the current set of available EPM Adapters to enable deployments and runtime management for those technologies (OpenStack/Heat, AWS/CloudFormation[30], Aria, Kubernetes[31])
- Stabilize and improve the platform support for other operating systems: The EPM itself and also the support for workers shall be capable to support Windows and Mac Workers as well.
- Placement algorithms for automated orchestration: As one of the research items it is foreseen to provide placement algorithms based on several parameters. This is used to deploy the virtual resources in appropriate places which can be defined by users to allow the best allocation of resources. Automated Orchestration is already provided but uses round-robin to select the target infrastructure whereas this can be improved by designing algorithms taking into consideration the current location of PoPs, available CPU or memory, or other parameters.

4.1.4.4 Code Reports

In ElasTest, EPM has been integrated with the CI system that uses Jenkins for automated tests and builds after every commit. For calculating the code coverage the EPM is integrated with Codecov.io.

4.1.4.5 Code Repository

The EPM code repository can be found on GitHub[32] and is licensed using Apache 2.0 [3]. Within that repository, there is documentation detailing how to run, use and extend the EPM.

[30] Cloud Formation, https://aws.amazon.com/es/cloudformation/
[31] Kubernetes, https://kubernetes.io/
[32] EPM GitHub, https://github.com/elastest/elastest-platform-manager

4.1.4.6 APIs

In the figures below it can be found the APIs exposed by the EPM. Those APIs are basically consumed by the users of the EPM (e.g. TORM, ESM) which are designed for the requirements coming from the other ElasTest components. They are using the OpenAPI Specifications (OAS)[33] which is a standard, programming-agnostic interface description for REST APIs which was agreed on and is used ElasTest platform wide to. Thanks to OAS, it allows the generation of the API description and was also used to generate the SDKs for python and Java.

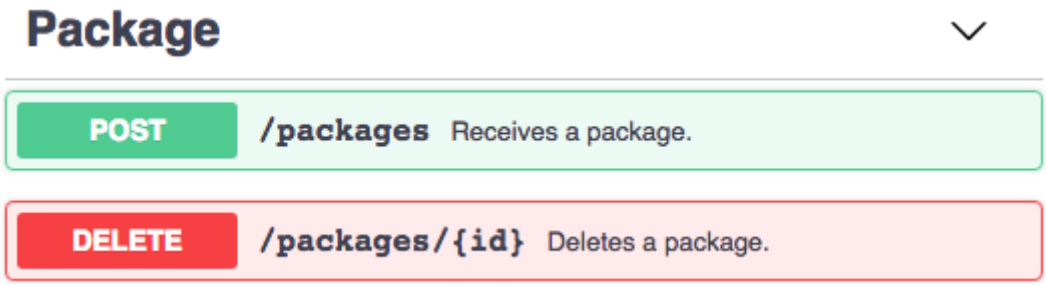

Figure 8. EPM API: Package

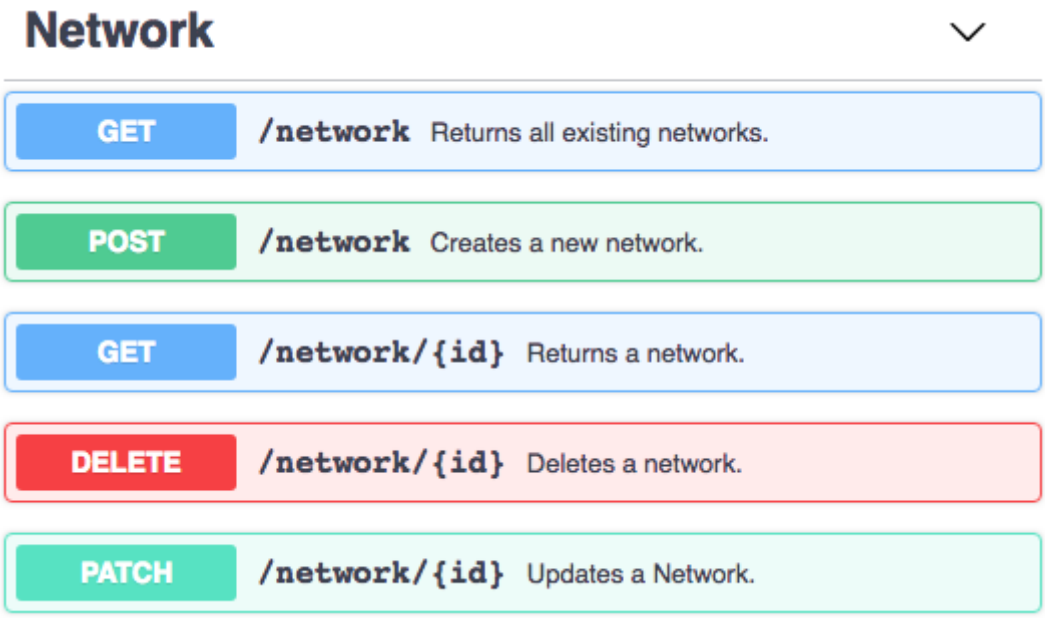

Figure 9. EPM API: Network

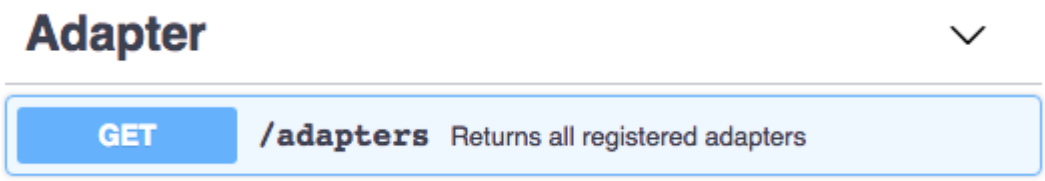

Figure 10. EPM API: Adapter

[33] OpenAPI Initiative, https://www.openapis.org/

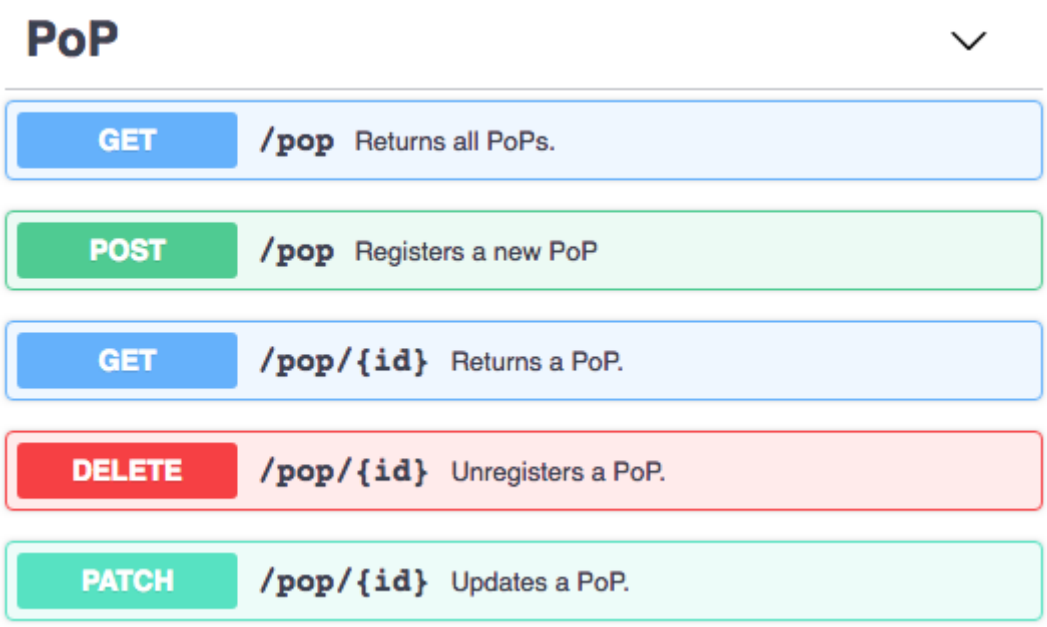

Figure 11. EPM API: PoP

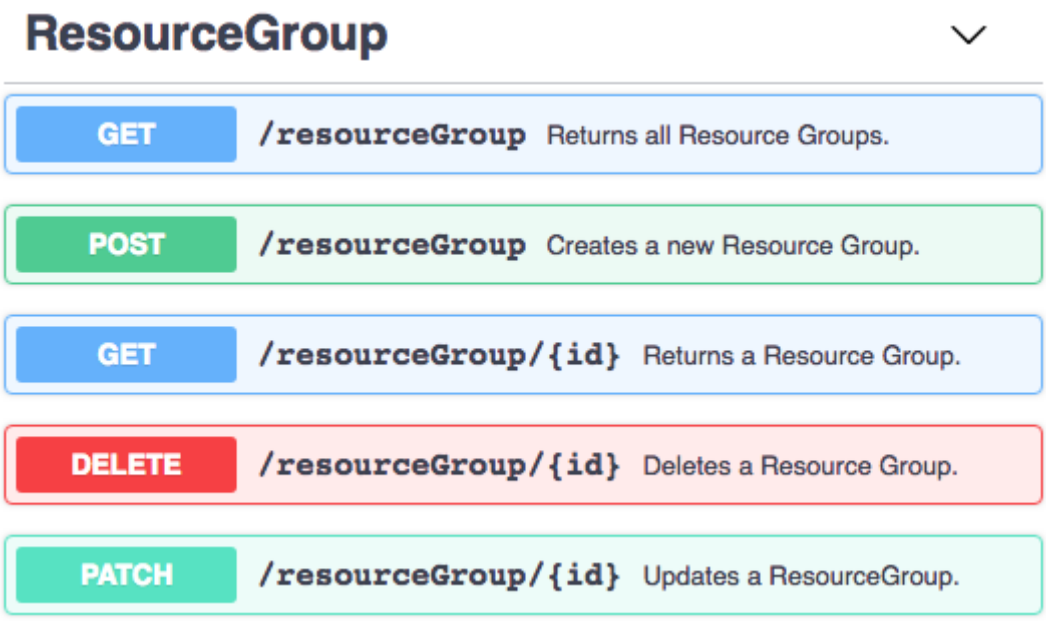

Figure 12. EPM API: ResourceGroup

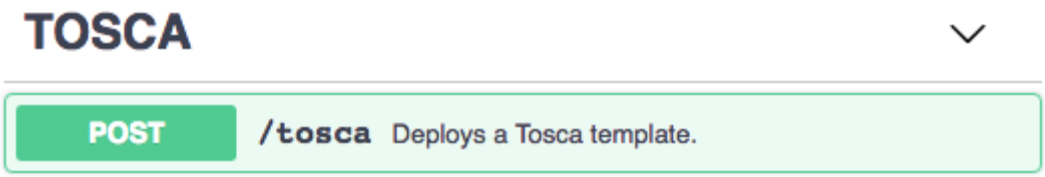

Figure 13. EPM API: TOSCA

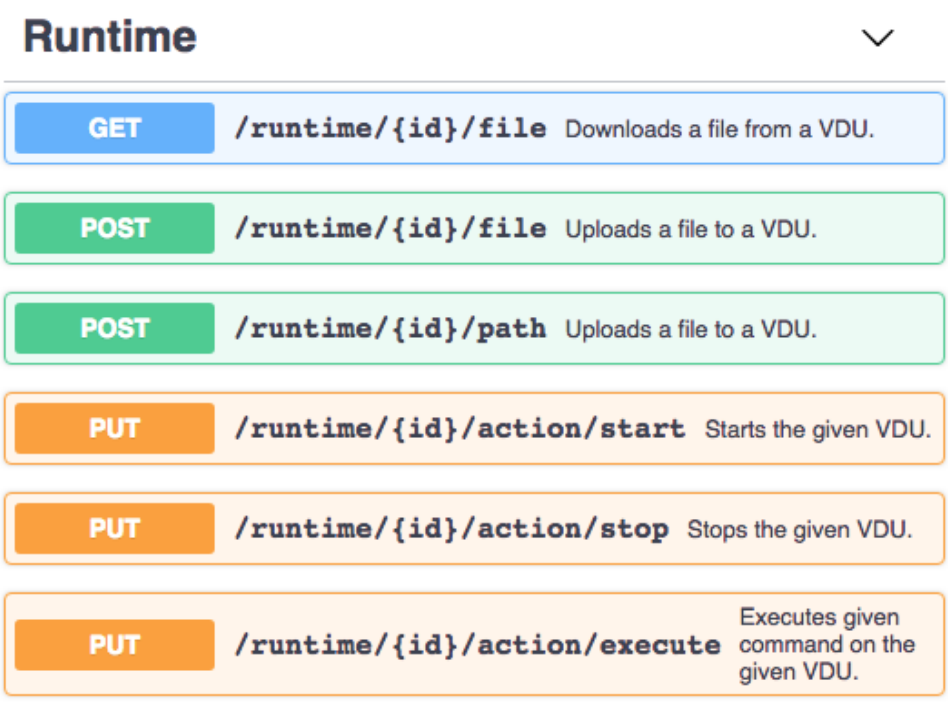

Figure 14. EPM API: Runtime

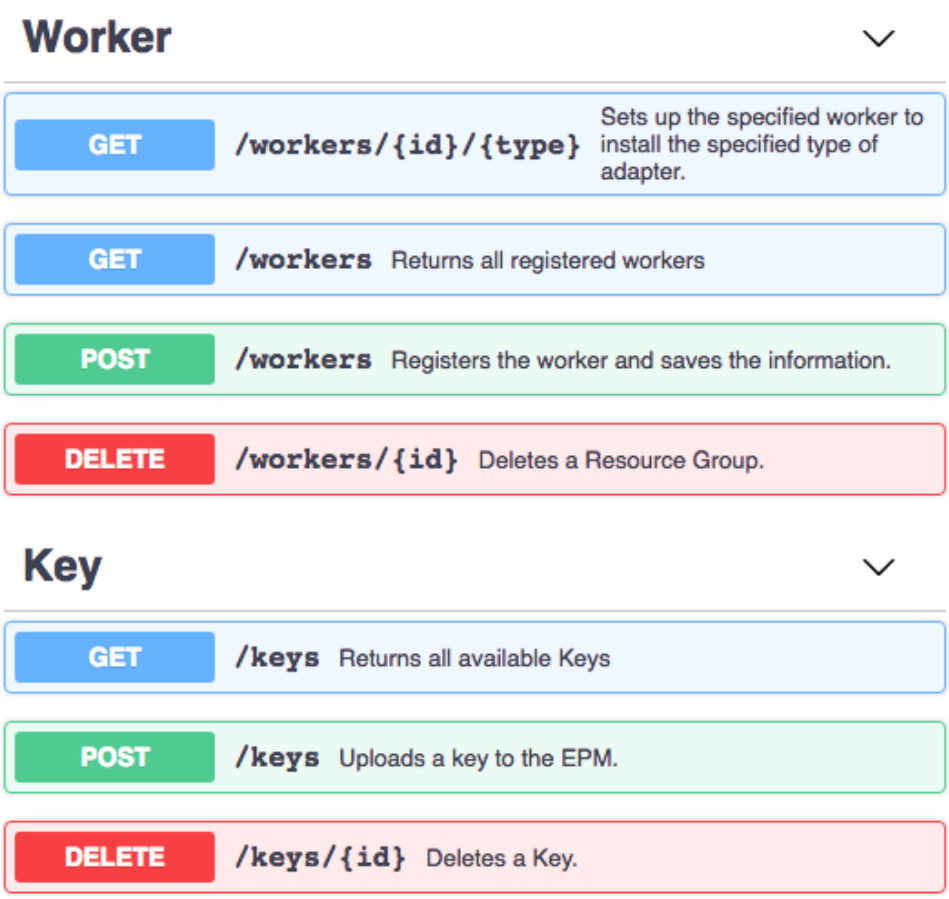

Figure 15. EPM API: Key and Worker

4.1.5 Research Results and Future Plans

- Customized orchestration solution for testing environments with advanced functionalities such as runtime operations
- Provider vendor lock-in
- Integration of several cloud environments for Multi-provider support
- Placement of virtual resources

4.2 ElasTest Monitoring Platform (EMP)

4.2.1 Introduction

From the DoA [1], the scope of ElasTest Monitoring Platform (EMP) is captured in these sentences -

“

ElasTest is a complex software itself and it needs to be monitored for different purposes including problem diagnose, resource utilization tracking, energy consumption tracking, cost tracking, etc. This subtask shall take the responsibility of creating the appropriate monitoring tools, GUIs and APIs enabling:

To recover in a seamless way information related to the runtime execution of the different ElasTest components including logs, internal status, resource utilization, etc. These capabilities shall enable the diagnosis and isolation of problems taking place inside ElasTest logic.

To collect and expose through an API the appropriate monitoring information related to resource utilization of the cloud resources consumed by the testing activities (e.g. TJobs instances, Test Support Service instances, etc.) This information shall include cost consumption, energy consumption, memory consumption, CPU consumption, etc. This information shall be made available through a northbound interface to the TORM so that the appropriate engines (see Task 4.4) can consume them.

To enable the instrumentation of the cloud resources consumed by the testing activities so that testers shall be able to inspect the status of the different TJob instances and Service instances, recover logs from them and control their lifecycle (e.g. stopping them).

To develop a toolbox enabling the installation and management of all such capabilities in ElasTest.

”

The above snippet captures the minimal set of functionalities needed for ElasTest but in a true spirit of R&D, a few additional requirements were included as part of scope of work to advance the state of the art. Deliverable D2.3 lists the requirements and high level architecture for EMP. In this section we will delve in depth into EMP, see detailed architecture, interaction diagrams, and current development status and roadmap for the remaining duration of the project.

The basis design philosophy behind EMP is quite simple. EMP supports creation of monitoring spaces. A monitoring space can be thought to be a collection of relevant metric streams belonging to either a complex system being monitored, or a set of

related microservices. Within a monitoring space, multiple metric series coexist. A series can be thought of collection of metrics stream from the single agent. An agent can be configured to handle log from a microservice, or host metrics, or a single docker container stats.

The design philosophy can be described succinctly by Figure 1. Series is marked in the following figure as *subspace*. Internally, the codeword for **EMP** implementation is **Sentinel**, therefore in the later sections; *any reference to Sentinel in the images should be interpreted as EMP*.

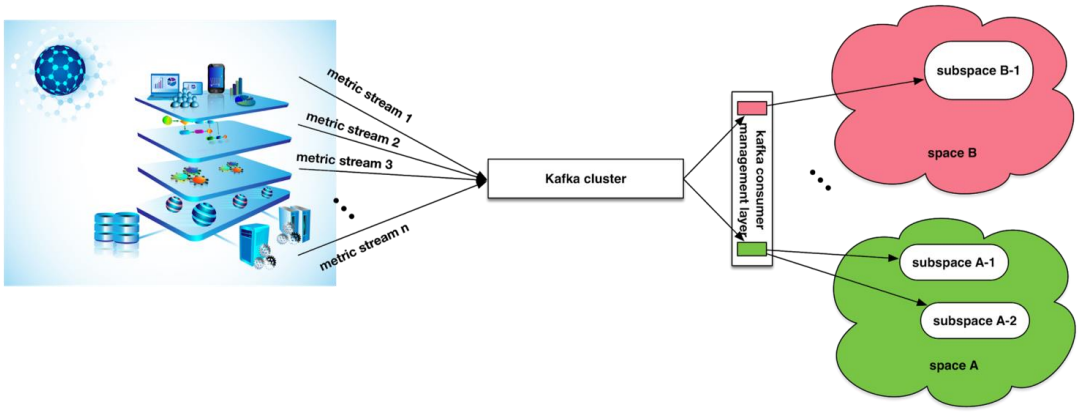

Figure 16. EMP design philoshophy, subspace is synonymous to metrics stream described in the text

4.2.2 Baseline Concepts and Technologies

EMP framework has been implemented in Java and has been packaged as Docker image which facilitates the deployment on a single machine or over a cluster of nodes in a relatively straightforward manner. The principal functions of any monitoring platform are -

- Enable metrics collection, and retention
- Allow information retrieval for analysis
- Condition based alerts and alarming functionality

In order to support high volume metrics and log streams, Apache Kafka[34] was chosen for the messaging subsystem for the following reasons:

- Fast delivery at scale
- Horizontally scalable even across multiple datacentres
- Easy programmability
- Supports multi-tenancy, geo replication
- Topic centric distribution with message containing keys is naturally aligned with EMP's notion of spaces and subspaces (series).
- Built in resilience, coordination, among other desirable qualities
- Flexibility is use as queuing, messaging system, storage or streaming platform.
- Large and active community

[34] Apache Kafka: https://kafka.apache.org/intro [accessed: 2018-05-23]

The AAA is handled internally at the moment, but in the near future, use of Keystone is anticipated as a replacement AAA system for use in EMP.

The persistence is supported by relational as well as time series optimized database. For static, account related data, file based sqlite is used as a lightweight relational database. For metric and log streams, InfluxDB[35] is used as it implements time based sharding as well as allows downsampling policies for older data. Figure 2 shows the catalogue of all relevant technologies that have been used in EMP at the time of writing of this document.

Figure 17. Technology landscap in EMP

For visualization, Grafana[36] has been used as it has a proven integration with InfluxDB and allows charting of key metrics collected in EMP a relatively straightforward task.

A few EMP agents have been developed and packaged as docker images to facilitate the metrics collection and transmission into EMP. The agents have been developed in Python3 to keep memory footprint lower and also demonstrate independence of language for development of agents. In the current release, the following agents have been developed:

- System stats collector
- Docker stats collector

[35] Influx Data, https://www.influxdata.com/
[36] Grafana, https://grafana.com/

- Log file parser, tokenizer and transmission agent (limited to log4j formatted log files from Java applications).

4.2.3 Component Design and Architecture

A high level EMP architecture is included in D.2.3. [5] along with module descriptions. Here we present a more detailed version of the same, see Figure 19.

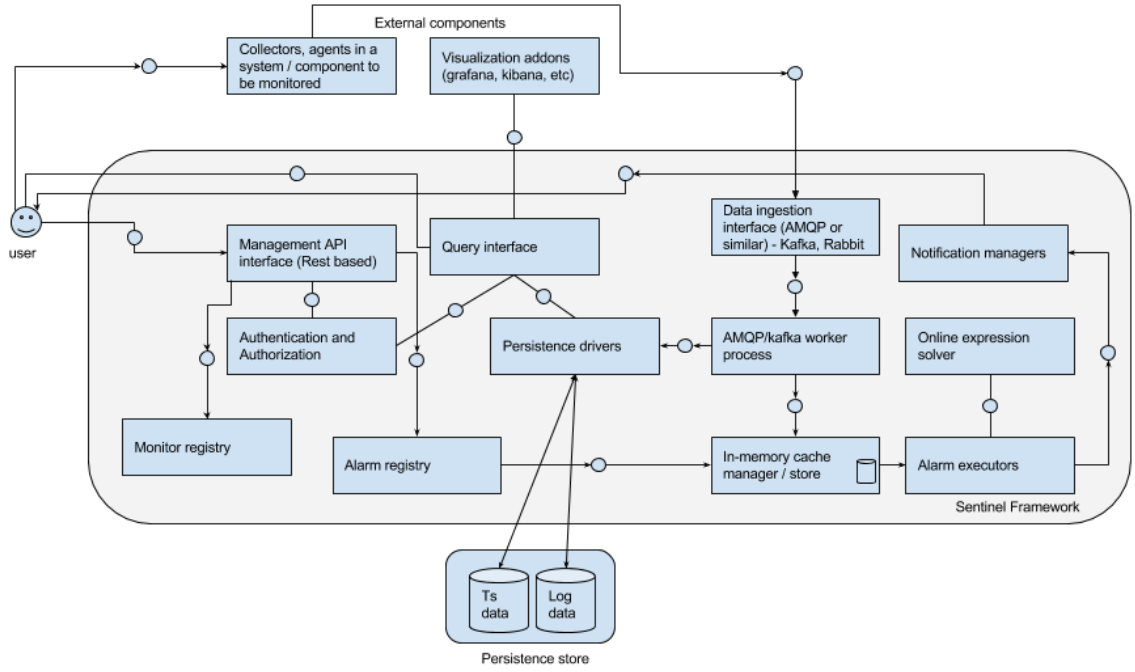

Figure 18. FMC diagram showing detailed EMP components

As can be seen in the figure, the data is gathered by agents (log parsers, system performance metrics collectors, etc) which are low profile, tiny processes running is target environment to be monitored and either periodically or on change detection gathers relevant data, pre-processes packages sending them as a stream to EMP. The user can use the Management API to create monitoring spaces and series as well as manage alert rules. The framework has authorization built in and enforces through Authentication and Authorization module and the data in motion is over industry grade TLS/SSL connection.

The alarms are stored in the Alarm Registry. The alarm definition which is a well formed mathematical expression is evaluated using recent values of corresponding metrics or series of cached recent past data values of a metric through the Online expression solver. The data ingestion interface in the initial prototype is Kafka and adding support for RabbitMQ[37] is planned. The framework is capable of using several persistence stores and the interactions are done via the Persistence drivers as shown in Figure 2 above. The query interface enables users to perform interesting analytics with the stored data which will enable easy debugging of large scale distributed

[37] RabbitMQ, https://www.rabbitmq.com/

services through data correlation studies among different series within the same monitoring space in EMP.

4.2.3.1 Use cases description

4.2.3.1.1 Use case A: simultaneous tracking of system parameters as well as log messages

Imagine a situation where a few ElasTest platform core services are not reachable due to high packet load on the n/w interfaces and not due to a bug in the software itself. A visual representation of all facets of the platform including system metrics along with log visualization will be very helpful in identifying this case. Since EMP handles both metric streams and log messages in a similar manner, it enables simultaneous visualization of both types of data streams. The ElasTest platform will have both log agents and system characterization agents sending continuous streams of metrics to EMP and using a visualization tool such as Grafana, the appropriate visualization charts can be rendered through the EMP.

4.2.3.1.2 Use case B: correlated query over multiple series in a given time window

An EMP user want to execute a correlated query in a given time window over multiple data series, s/he sends the query to EMP query interface, EMP query engine determines how to extract various data snippets from multiple series, performs appropriate filtering and aggregation as needed and responds back to user's query. Such a capability will enable users of EMP to investigate cascading effect of service degradation over the entire service ecosystem or service-chain in a distributed systems deployed at large scales. Other uses of correlated queries can be easily imagined.

4.2.3.1.3 Use case C: alert definition and triggering

Scenario: a service with overloaded CPU must be scaled out to spread the load, the user of EMP creates an alert definition for ex: if mean (last 10 CPU load readings) > 80% then send alert to elasticity manager using a call-back hook! The EMP keeps an online mean of last 10 CPU readings from the specified data series and using the online expression solver determines if the triggering conditions are satisfied or not! Is found to be *True*, the callback hook is executed.

4.2.3.1.4 Use case D: liveness detection and callback hook

Imagine, ESM needs to track which services managed by it are alive. It needs to take corrective action in case some services are unresponsive. The ESM can use EMP to achieve this. ESM process gets an user account and API credentials for EMP. Using API calls, for each instance of service it provisions, it creates a liveness-check object along with desired periodicity and the service instance endpoint in EMP. It also specifies a call-back REST call signature for such object in EMP. The EMP periodically performs the liveness test against service instance endpoint, and in case a test fails due to service unreachable error, or a timeout, EMP executes the call-back restful hook with configured message. Doing so will notify about unresponsiveness of a particular service instance to ESM. To limit the security footprint of this feature, EMP will only allow GET calls and a periodicity of no less than a few seconds.

4.2.3.2 Sequence diagrams

The following sequence diagrams illustrates interactions among internal components of EMP and external entities such as EMP users (TORM / TJob developer) while performing following tasks -

- Account creation in EMP, space and series setup, Figure 20
- Metric ingestion and persistence in EMP, Figure 21
- Alerts management, Figure 22
- Data visualization, Figure 23
- Data query, Figure 24

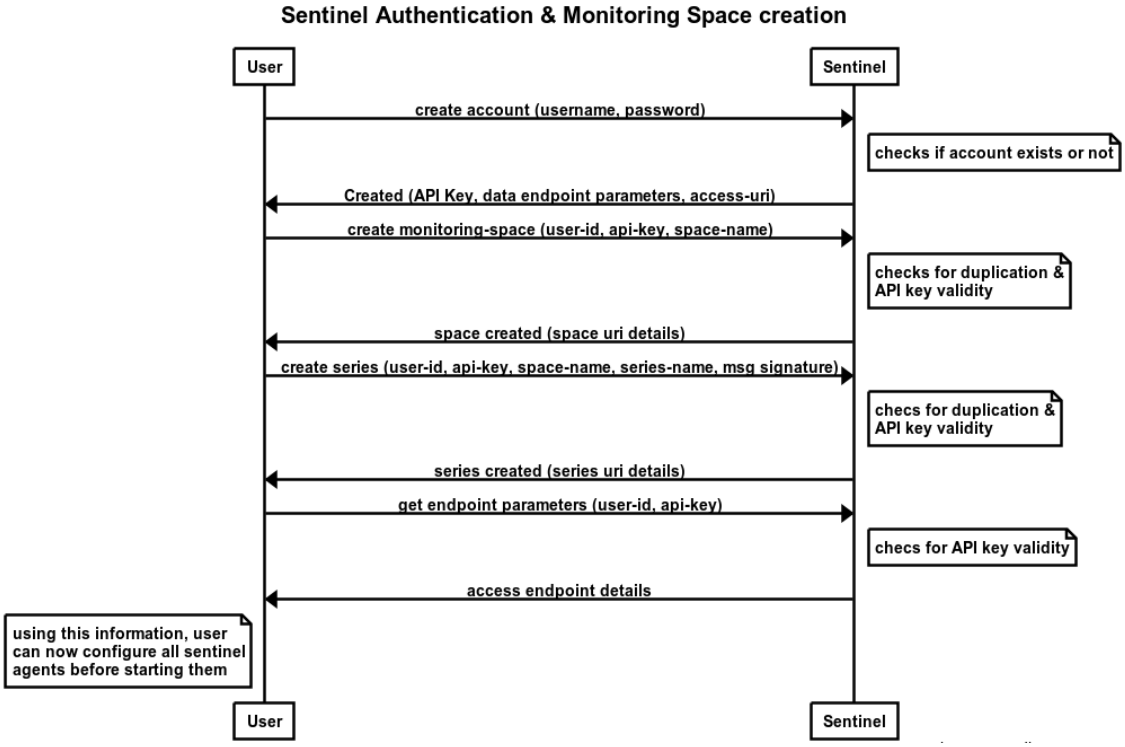

Figure 19. Sequence diagram showing user registration and monitoring space management

Figure 20 shows the messages exchanged between EMP user and the EMP AAA subsystem for registration of the account. The user account creation is only permitted to EMP special admin account. Users of EMP at this moment can't self-register. This limitation may be removed in a future release. When an account is successfully registered with EMP, a unique API-key is generated and associated with that account. Subsequent steps shown in the Figure 4 above such as creation of monitoring space, metrics series within a space, etc. must be carried out together with the API-key included in all requests.

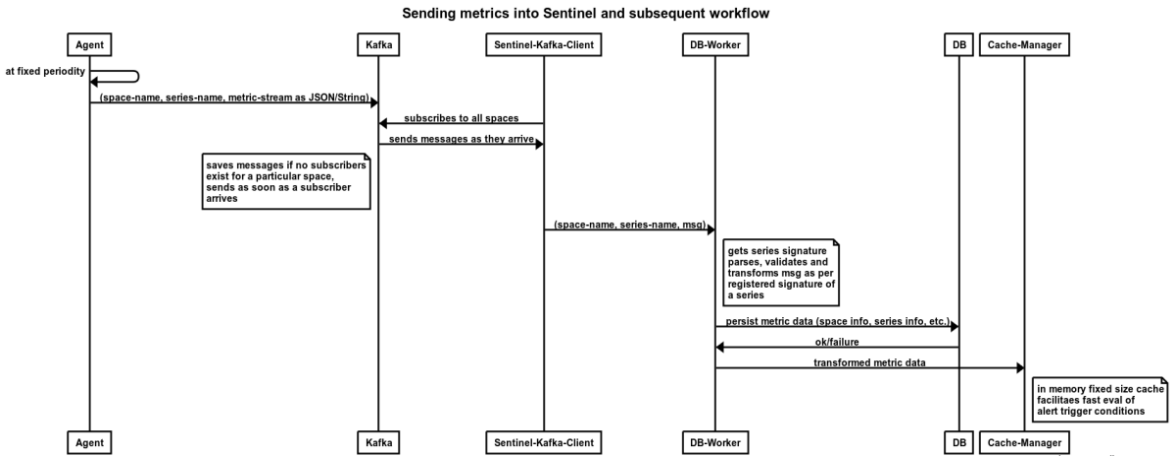

Figure 20. Sequence diagram showing metrics streams and data workflow through Sentinel

Figure 21 shows the interactions among internal components of EMP when a metric is sent by EMP agents. Kafka is the messaging technology selected and all agents send their data stream to kafka for a particular topic. The format of all messages sent to kafka cluster must be in the following style:

```
{topic, key, message}
```

Every monitoring space and metric series maps to Kafka topic with a key. This management of topics is entirely handled by EMP and the users and agents are oblivious to the process.

In EMP, Kafka worker threads handle all incoming metrics as soon as possible. Depending on the underlying physical host capability, each worker thread can be responsible for more than one topic. To keep the Kafka worker lightweight, the job of persistence is delegated to DB worker agent. Kafka workers, simply send the received messages to the DB worker agents which are sent to w common thread pool for scheduling by the underlying thread management subsystem.

There is a provision to keep most recent ‘n’ points from a metrics stream in memory to facilitate online processing and evaluation of alarms and SLA related triggers. This functionality is not available in release 0.9.0 at the time of writing of this report and in planned for the next release.

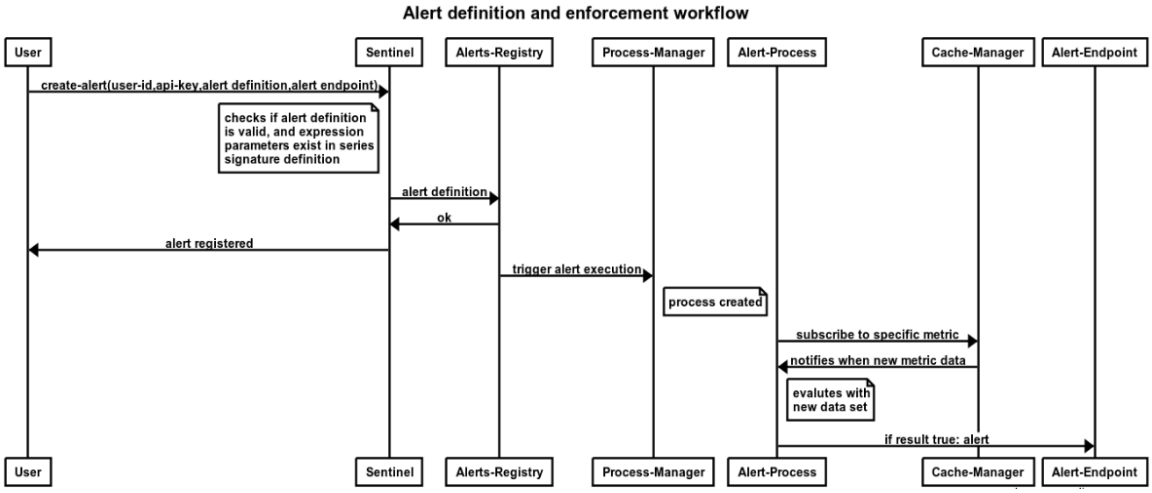

Figure 21. alert management and execution workflow

Figure 22 shows the interaction among various sub modules of EMP for alert registration and triggering mechanism. The alerts once registered are automatically scheduled and allocated to a alert process thread. The cache manager upon receiving newer metrics notifies the alert process which then re-evaluates the alert condition with newer data points and if the alert condition is met, a notification is sent to the registered alert-endpoint. A more simplistic version of alert manager is available in the current release at the time of writing of this document.

EMP uses Grafana for metric visualization. Figure 23 shows the steps necessary by the users to access the visualization engine. Grafana periodically queries directly the InfluxDB endpoint to generate live graphs.

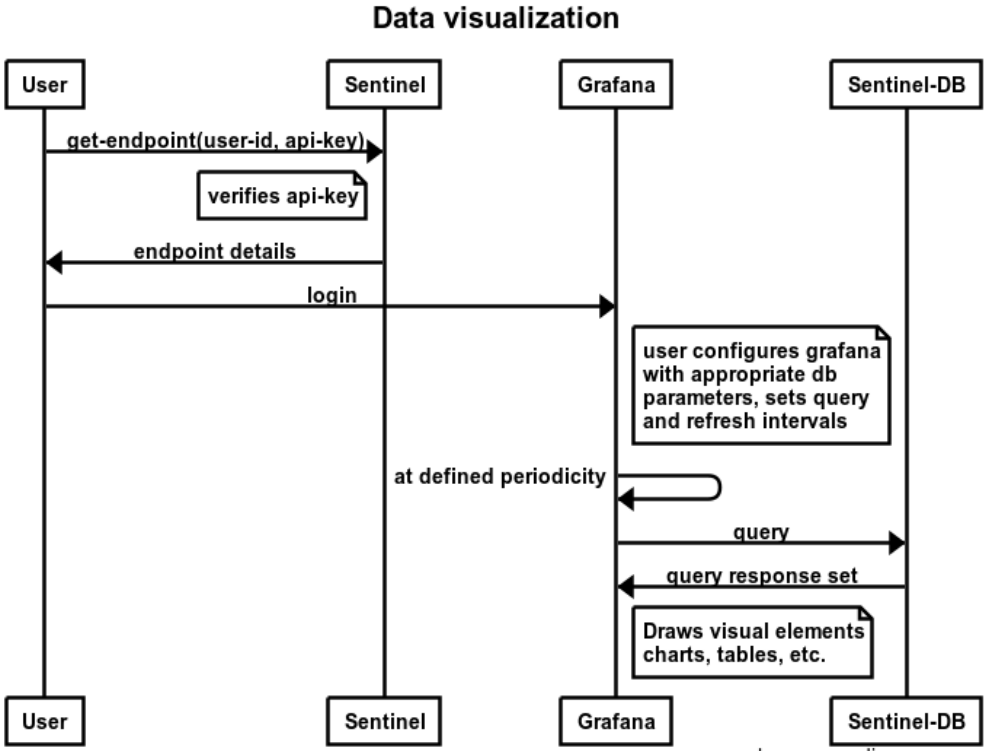

Figure 22. data visualisation sequence with Grafana and Sentinel

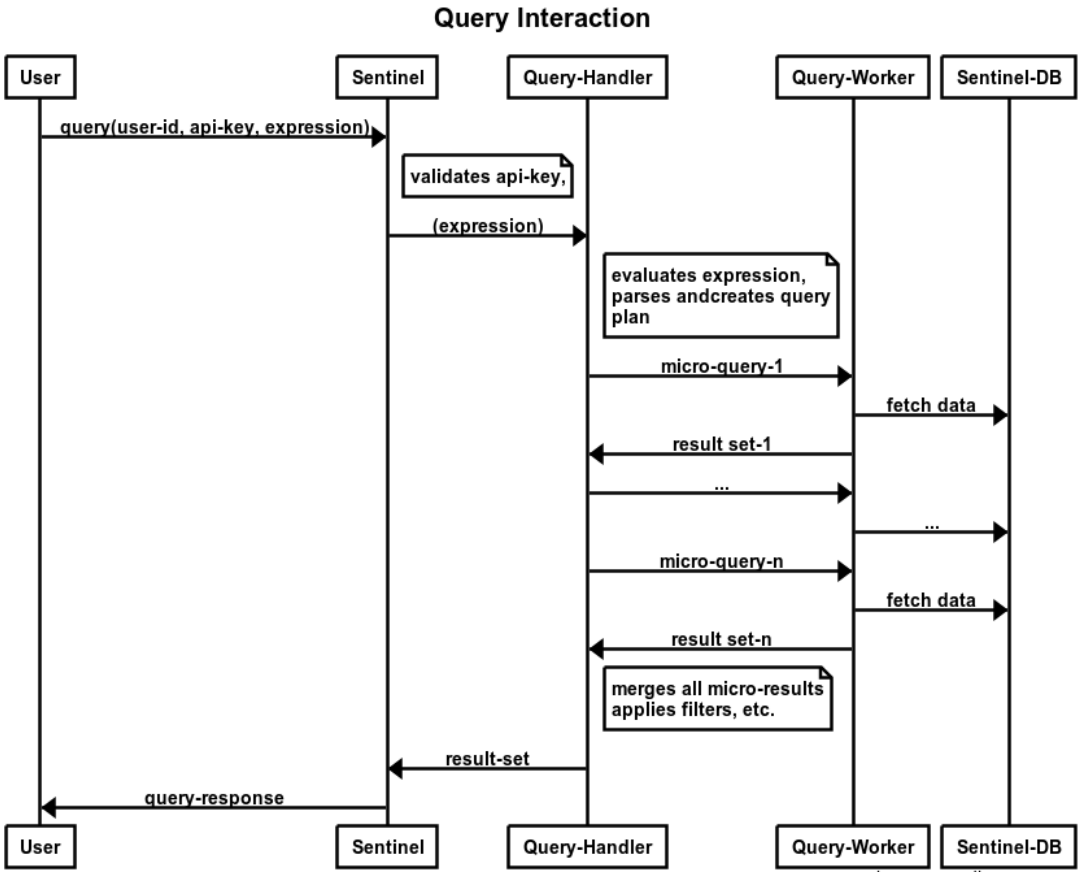

Figure 23. user query workflow

Figure 24 above shows the planned query processing by EMP. User send in a compound query using a query language to be designed for EMP. Once the query is received at EMP, the query-handler breaks down the compound query into an execution plan and sends the constituent parts to query worker processes. Each query worker independently processes the basic query and sends the result back to the handler, which then merges the responses, applies filters as necessary and returns the response back to the user.

4.2.4 Roadmap and Features

The features exposed by EMP can be summarised under these following categories:

- Uniform data ingestion interface: EMP utilizes the same common data interface to gather system stats as well as log streams
- First class status to logs and system metrics: the system metrics and log entries are treated in exactly similar manner in EMP
- Alerting capabilities: EMP will support custom alert definitions and execution call-backs, users of EMP will be able to create alert definitions based on the monitored metrics and define mathematical operations with one or more metrics as the trigger mechanism. The alerting subsystem in EMP will support working with latest live data point, or allow the computation to go back n-points in the history for execution of the trigger function.

- Correlated query management and corresponding interface: EMP will integrate with expressive query language that will enable users to perform correlation between different metric streams to investigate performance flashpoints in the ElasTest platform.
- Scheduled system liveness test support: EMP will include feature to define REST based liveness checks of remote services, and corresponding call-backs to notify dead services.

The EMP exposes a RESTful interface using which an intuitive GUI has also been developed. The APIs allows the following action:

- Creation of new user account
- Creation of monitoring spaces
- Creation of multiple data series within any space
- Fetching of agent configuration parameters to aid with agent deployment
- Registration of health checks to track liveness of monitored entities (services)
- Query of health check history

The limitation of current APIs are following:

- Missing DELETE capabilities for all managed objects: users, spaces, series, health checks, etc.
- Missing UPDATE capability for all managed objects
- Missing query interface
- Currently this is satisfied via InfluxDB native query interface

The identified limitations are planned to be overcome in the subsequent releases of EMP. A few screenshots of the EMP GUI developed over RESTful APIs are shown next. Figures 25 through 31 captures the main elements of the EMP GUI. Other aspects include retrieval of user data, common configuration parameters needed for agents' configuration, etc.

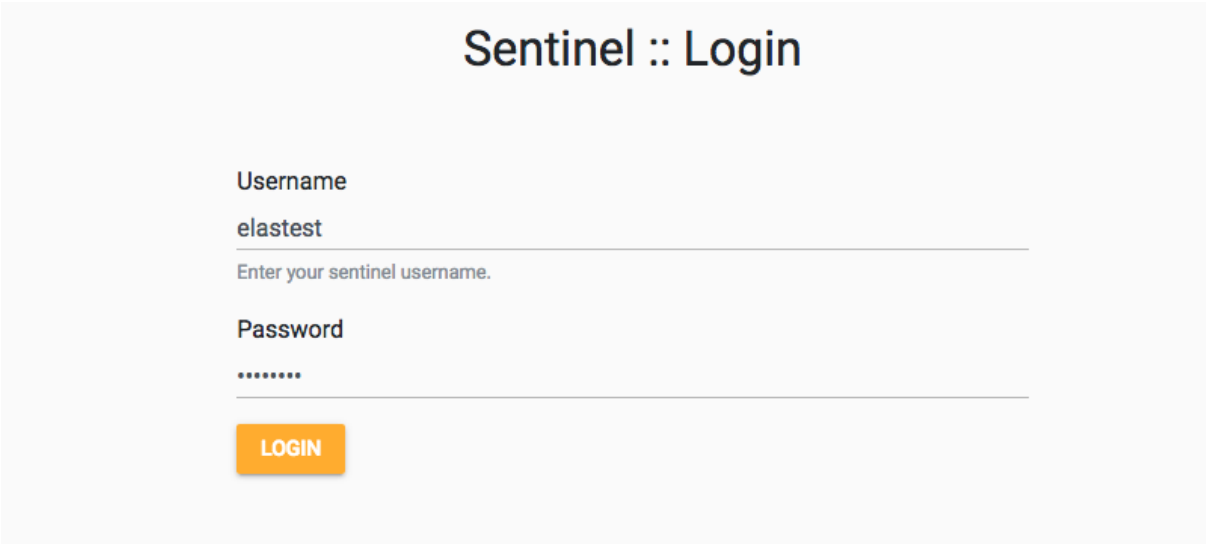

Figure 24. EMP GUI Login screen

Figure 25 above shows a simple and intuitive login screen which not only authenticates the user but also sets the API key as part of the session parameters which the web-UI uses to offer subsequent views to the user. Figures 26 and 27 below shows the overview page where the EMP user is able to get a gist of relevant information at a glance. Figure 27 shows all registered spaces and their key parameters to the user.

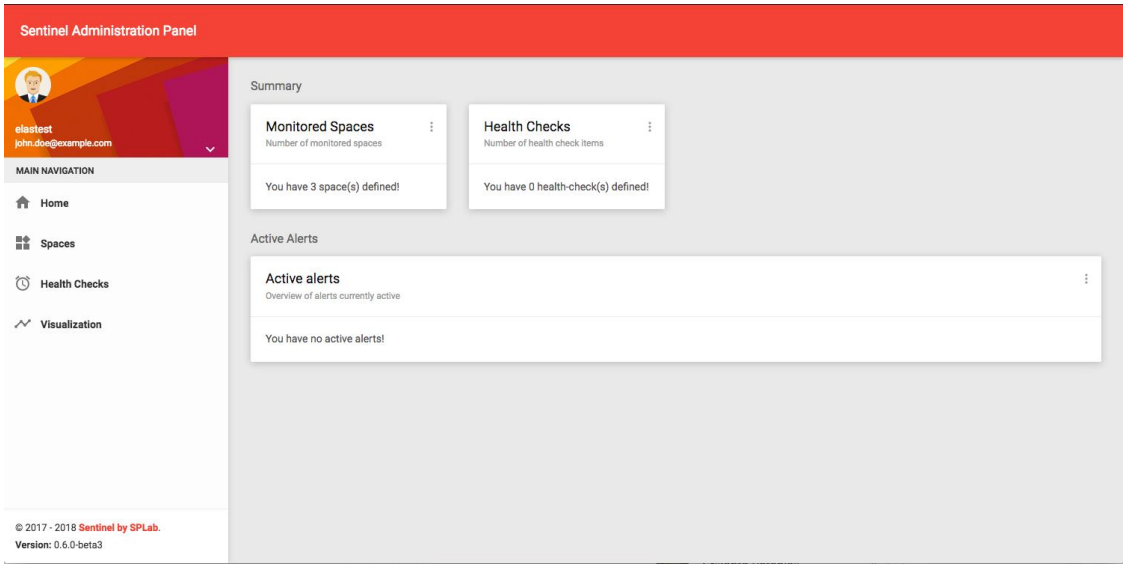

Figure 25. EMP overview page, showing spaces, health checks and any activity alerts

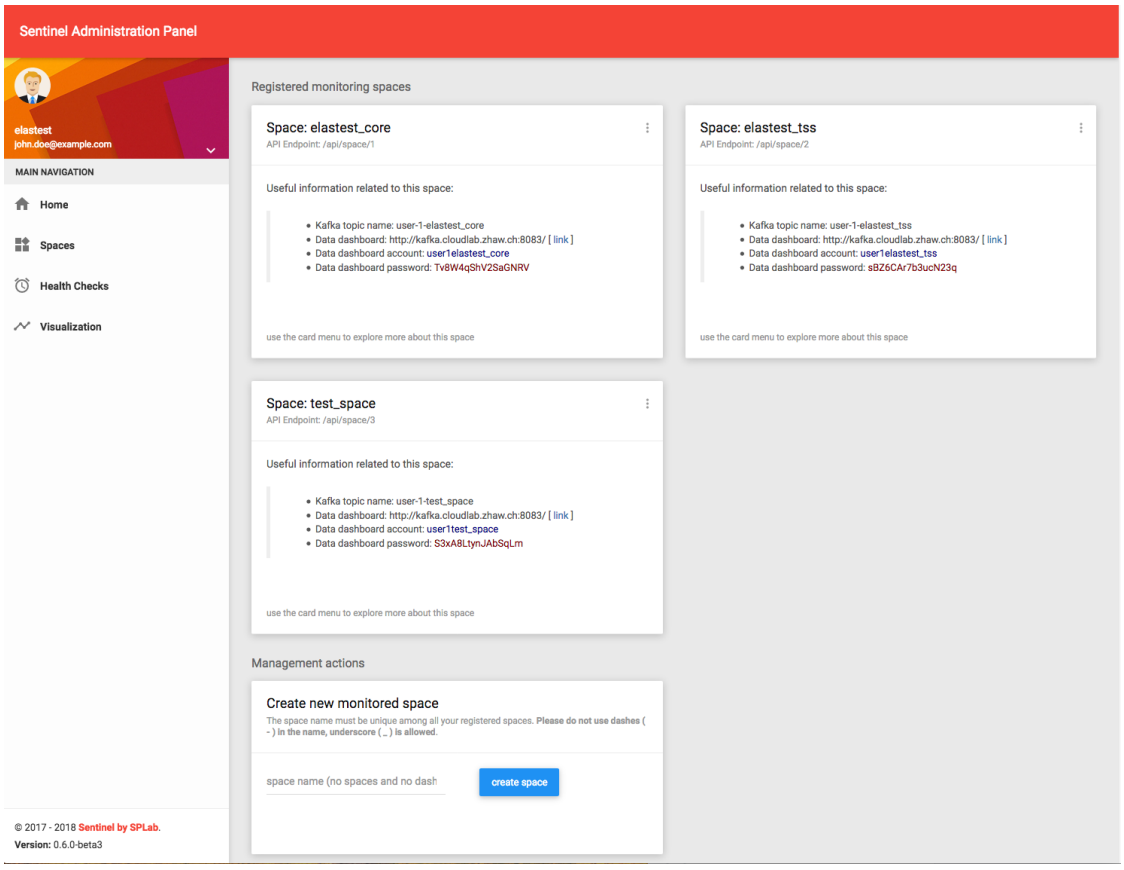

Figure 26. EMP space management page

Figure 28 shows the series management view of the EMP web-UI. It allows all functionalities to the user which they can perform using the RESTful APIs over command line.

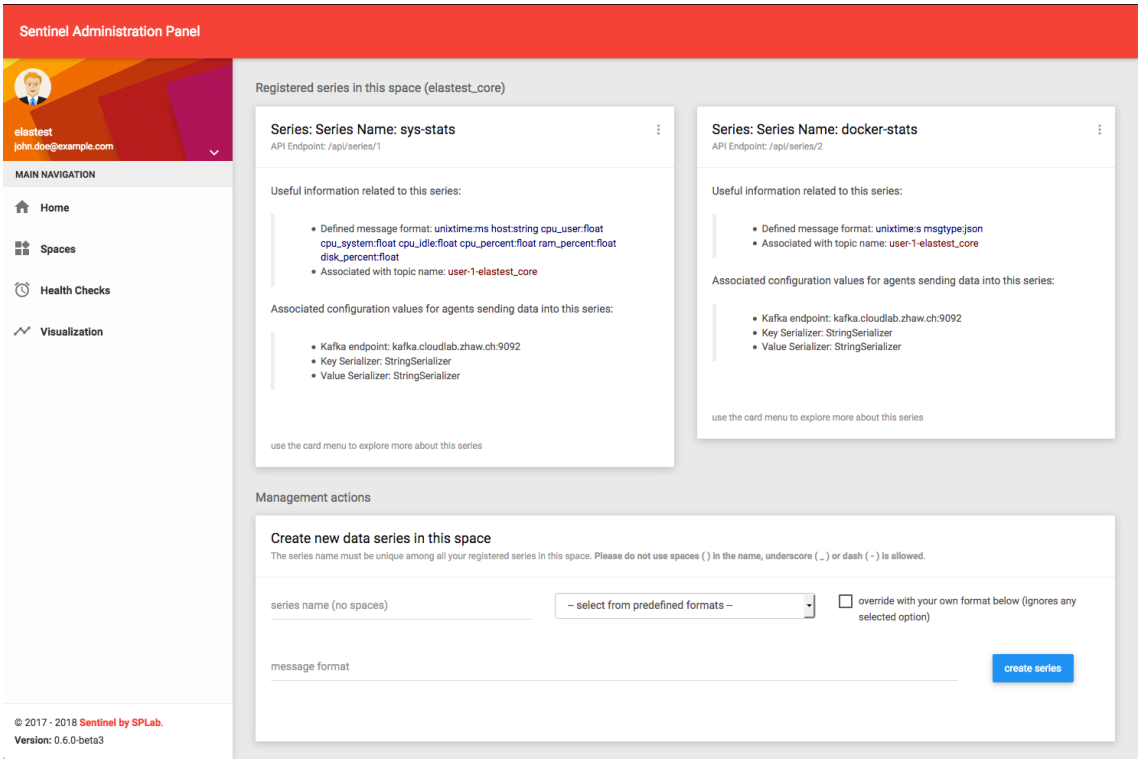

Figure 27. EMP series management (within a given space) page

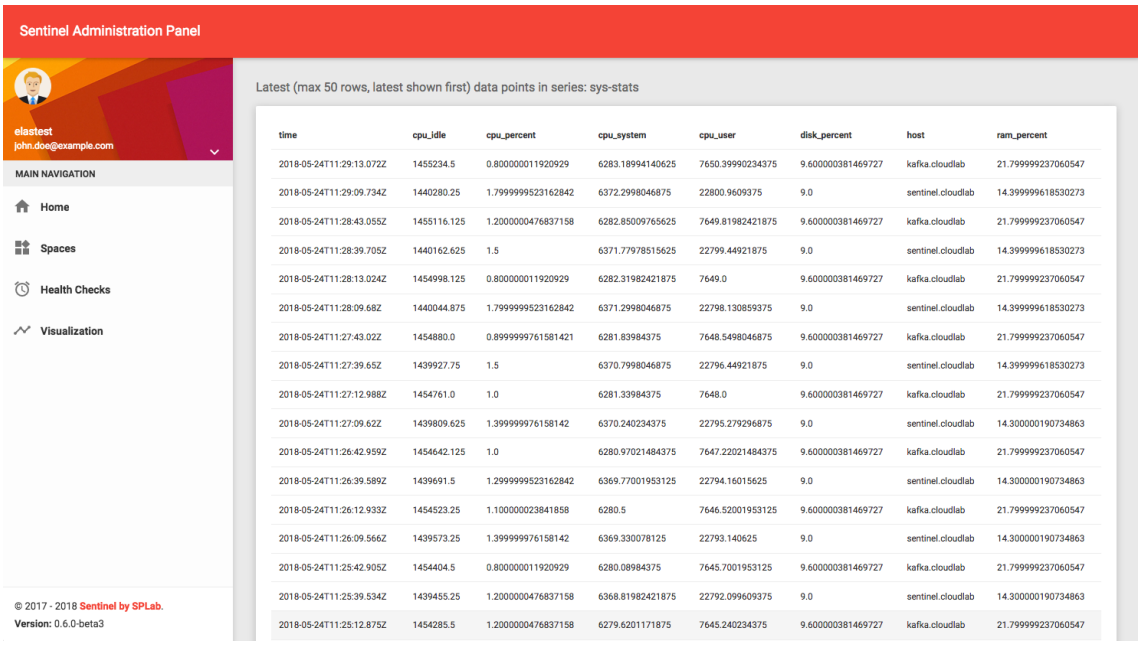

Figure 28. EMP – recent data point in a series

Figure 29 shows the page showing the user those latest 50 data values that were received by EMP as part of a series stream. In the recent releases of EMP, the data visualization is done via open source tool Grafana. Figure 30 provides a glimpse of that integration.

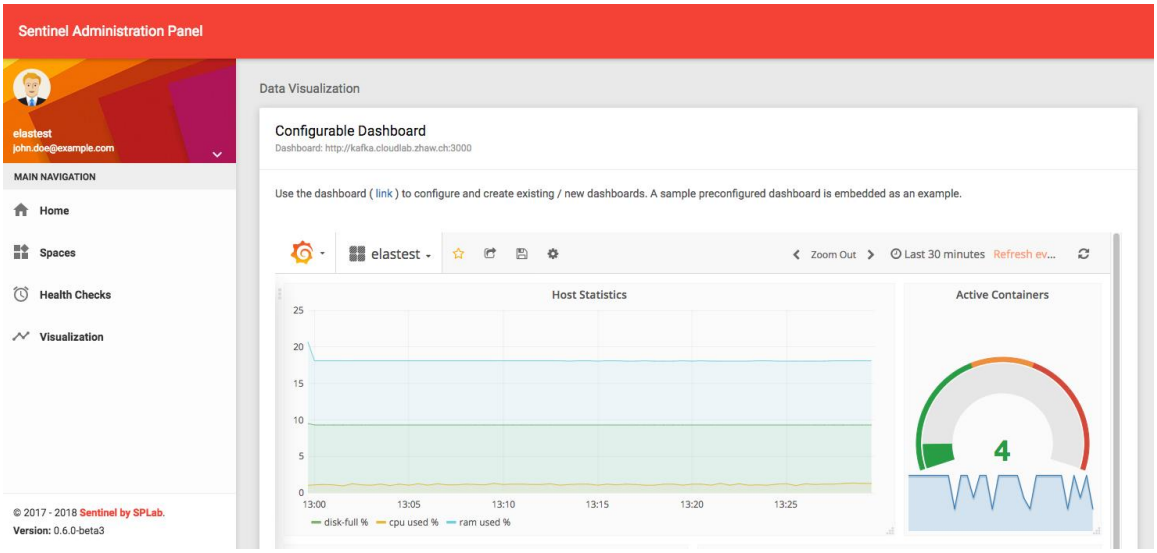

Figure 29. EMP embedded data visualisation page

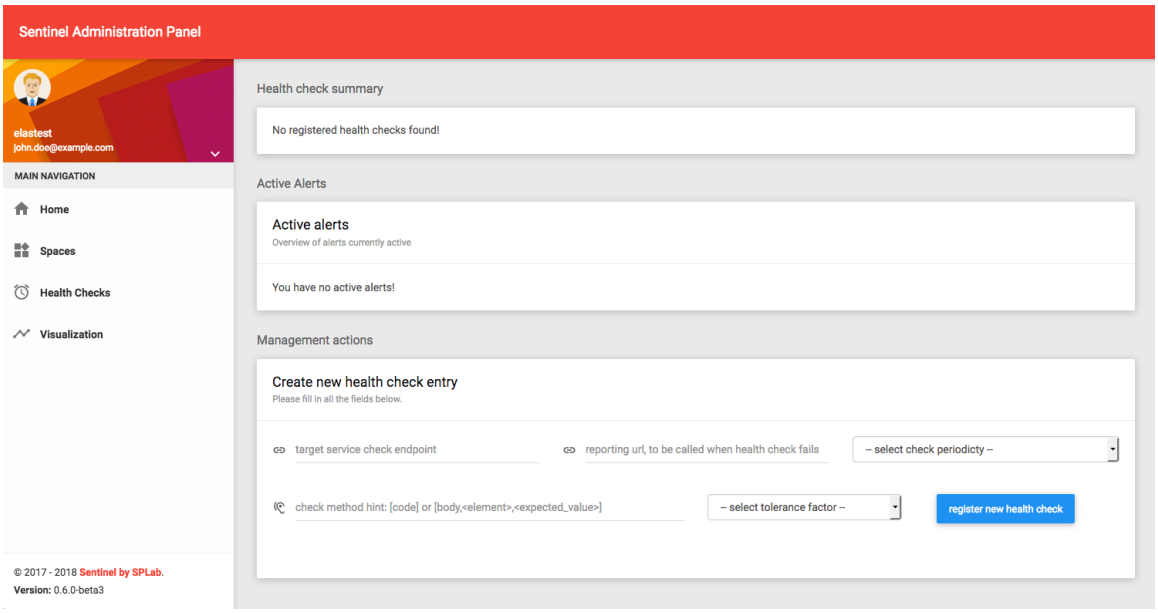

Figure 30. EMP health-check management page

Figure 31 above shows the view where user can create and monitor the state of all the health-check objects registered within EMP. If any active alert is ongoing, it is clearly shown as a prominent element in the web view to the user.

4.2.4.1 Code Reports

In ElasTest, EMP has been integrated with the CI system[38] that uses Jenkins[39]. The EMP is tested comprehensively using around 105 unit tests, and the code coverage attained has been consistently around 70%. Figure 32 shows the build pipeline within ElasTest CI system.

[38] ElasTest CI service: https://ci.elastest.io/jenkins/
[39] Jenkins, https://jenkins.io/

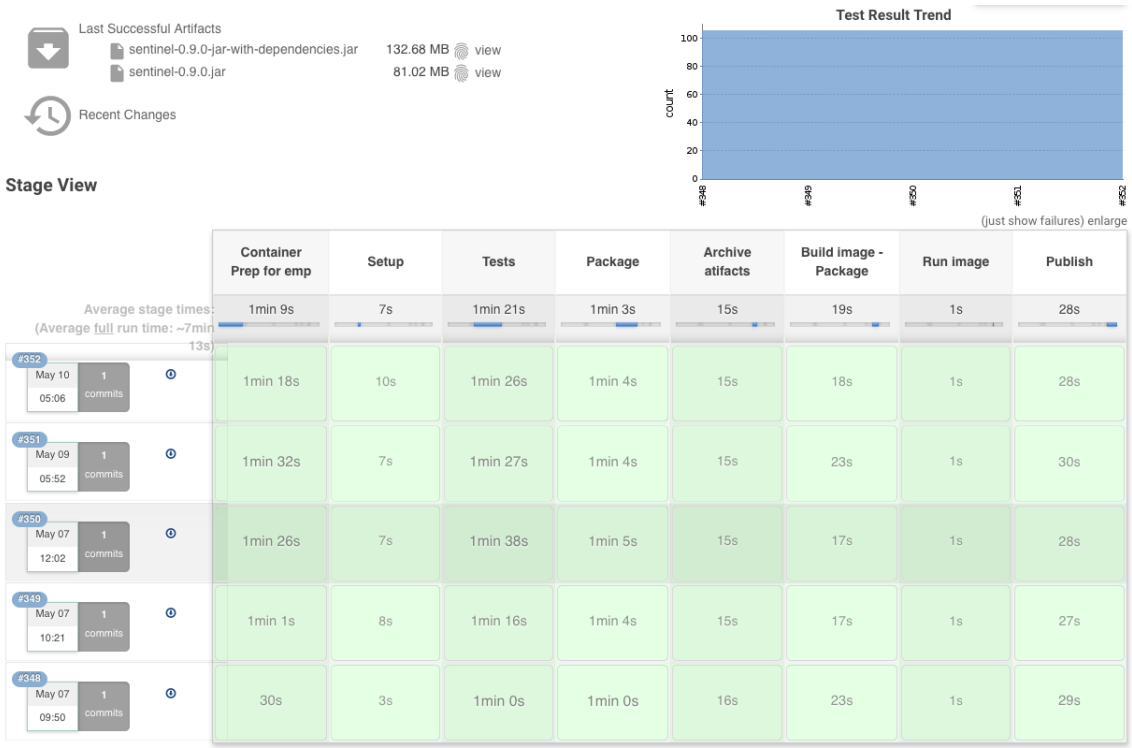

Figure 31. ElasTest CI dashboard for EMP test & build pipeline

The Java plugin Cobertura[40] is used to generate code coverage reports and then it is visualized using an external service: codecov.io[41]. Figure 33 shows the coverage chart for EMP.

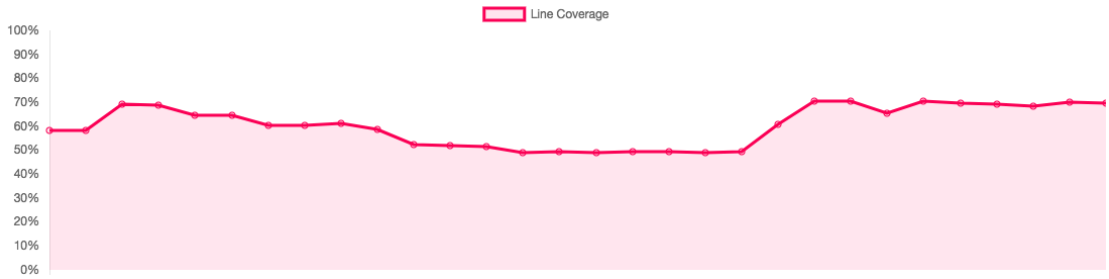

Figure 32. EMP code coverage graph [accessed: 2018-05-24]

4.2.4.2 Code Repository

The EMP codebase and all documentation, agent codes, and docker build scripts are publically available here[42]. Everything is released under Apache 2.0 software license [3].

[40] Cobertura: http://cobertura.github.io/cobertura/ [accessed: 2018-05-24]

[41] Codecov.io: https://codecov.io/gh/elastest/elastest-monitoring-platform [accessed: 2018-05-24]

[42] ElasTest Monitoring Platform (EMP): https://github.com/elastest/elastest-monitoring-platform/

4.2.4.3 APIs

The API for EMP were developed using OpenAPI specification. The functionality is already listed earlier. Here we show the Swagger rendering of the EMP APIs (Figure 34).

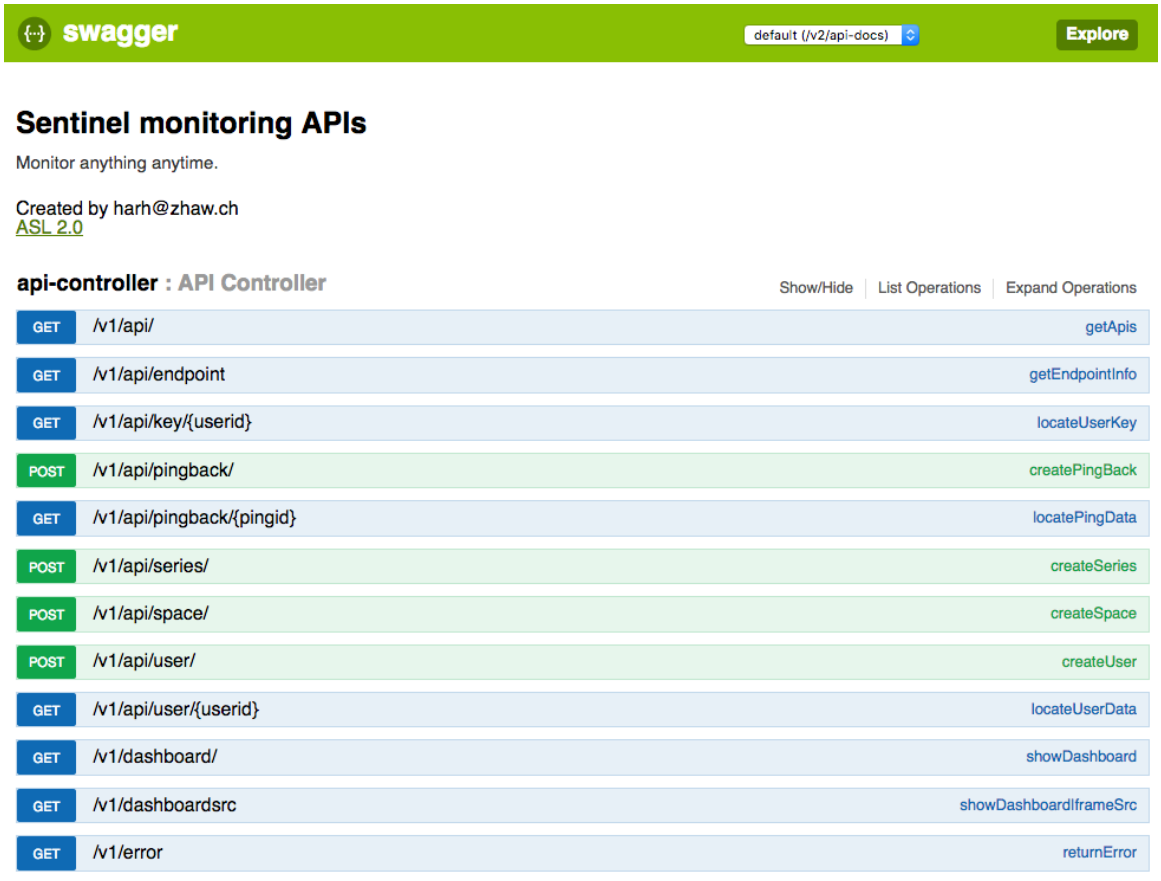

Figure 33. OpenAPI specification of EMP REST APIs, Swagger rendering

API endpoint	Verb	Return codes	Comments
/v1/api/	GET	200	ok
		500	service down
/v1/api/user/	POST	201	created
		400	check data
		401	valid admin token needed
		409	user account already exists
		500	system error
/v1/api/user/{id}	GET	200	ok
		401	unauthorized
		400	check data
/v1/api/space/	POST	201	created
		400	check data
		401	invalid api key
		409	space already exists for user
		500	system error
/v1/api/series/	POST	201	created
		400	check data
		401	invalid api key
		409	series already exists for user
		500	system error
/v1/api/key/{id}	GET	200	ok
		400	no such user exist
		401	invalid password
/v1/api/endpoint	GET	200	ok
		401	invalid api key
/v1/dashboard/	GET	200	ok
		500	system error
/v1/api/pingback/	POST	201	created
		400	check data
		401	invalid api key
		500	system error
/v1/api/pingback/{id}	GET	200	ok
		401	invalid api key

Figure 34. Expanded descriptions, methods, status codes for EMP APIs

Figure 34 shows at a glance overview of current set of REST APIs exposed by EMP together with the supported methods, status codes and interpretation of the returned status codes.

4.2.5 Research Results and Future Plans

One of the targeted R&D activities in EMP is correlated queries and analysis to enable fault tracing among cooperating distributed services. Below are itemized list of capabilities desired in EMP which forms the basis of continued R&D activity for the remainder of the project duration.

- Core feature: Comprehensive query language and query interface design
  - Allowing expression of intent to involve multiple spaces
  - Allowing temporal filtering
  - Allowing spatial & temporal aggregation
  - Allowing one or more data sets as response elements
- Core feature: Ability to support QoS/SLA equations in alerting subsystem
- UI enhancement: At a glance overview page showing all critical data elements in an easy to digest form.
- Detailed benchmarking of ingestion and query performance markers
- Ongoing SoTA comparative analysis of EMP with alternatives in open source

4.2.6 ElasTest Monitoring Platform Integration within ElasTest

EMP has been integrated with the ElasTest GUI and delivers a metric visualization that tracks resource consumption of ElasTest Platform key modules. EMP also tracks host metrics tracking CPU, Disk and Memory parameters identifying resource limitations of the node(s) where ElasTest is deployed. Figure 36 below shows a sample screenshot taken from the ElasTest nightly[43] deployment that depicts the metric visualization using Grafana embedded frame that uses the data collected by EMP from the two agents deployed on the node -

- EMP system stats agent
- EMP Docker stats agent

The EMP system stats agent was configured to send metrics to EMP at 30 seconds interval, and the EMP Docker stats agent was configured to send the collected metrics every 60 seconds.

[43] ElasTest Nightly: http://nightly.elastest.io:37000/ [accessed: 2018-05-24]

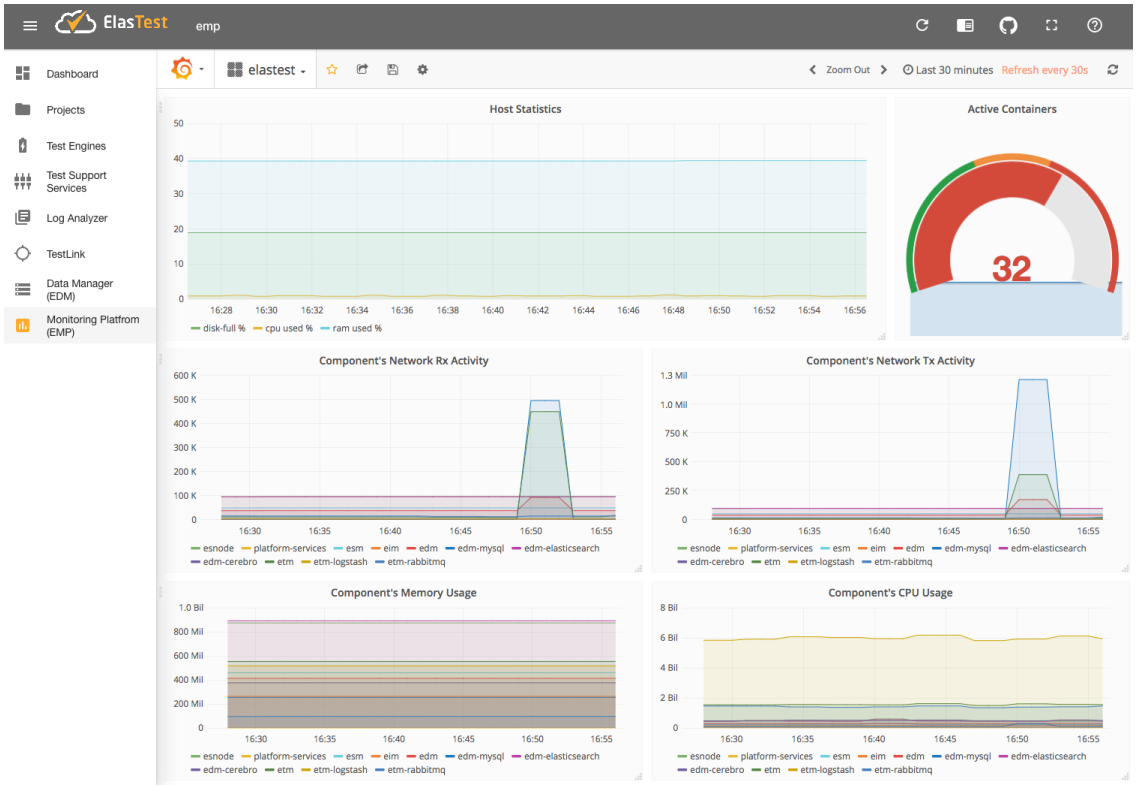

Figure 35. EMP visualisation pane tracking ElasTest Platform core modules

5 Service Lifecycle Management

5.1 ElasTest Service Manager (ESM)

5.1.1 Introduction

The service manager is based around the idea of delivering service instances on-demand to end-user consumers in an efficient and easy way to present their software to the service manager. For this, the ElasTest Service Manager (ESM) supports deployment using Docker-compose descriptions and will soon support Kubernetes-based descriptions. In order to use the facility of the ElasTest Service Manager one must use its API which is based upon the 2.12 version of the Open Service Broker API[44], and from which there are some specific ElasTest extensions added.

For the specifics of how a TSS should be presented to the ESM, D5.1. [9] has this information.

[44] Open Service Broker API, https://www.openservicebrokerapi.org/

5.1.2 Baseline Concepts and Technologies

Currently the ESM is implemented in Python and delivered as a Docker[45] container. This container does not persist data and so eases scaling the logic provided by the ESM.

The persistency of data models and structures are by default (unless configured otherwise) are persisted in-memory and will lose this information if the ESM process is destroyed or rebooted. In order to avoid this, the ESM provides support through the MongoDB[46] or MySQL[47] driver.

The Access, Authorization and Accounting (AAA) system that can be used with the ESM is OpenStack's Keystone[48]. No other AAA systems are seen to be supported as AAA is seen to be something that can be taken "off the shelve" and so a pragmatic decision was made to adopt Keystone, a widely deployed system.

The monitoring system is partially based on a combination of the ESM's own implementation and the ElasTest Monitoring Platform (EMP). In order to provide service-level measurements, the ESM carries this out. To provide resource-level metrics, the ESM uses the EMP, whose responsibility is to do this.

The API used to managing services offered by the ESM is the Open Service Broker API[49].

5.1.2.1 Open Service Broker API

The OSBA project defines an abstract API (using OpenAPI[50]) that allows developers, independent software vendors and SaaS vendors to deliver services and applications upon any platform that supports the on-demand creation of virtualized resources. Examples of such platforms include CloudFoundry[51], OpenShift[52] and Docker. The OSBA originates from the CloudFoundry platform, which allowed internal and external service providers integrate their service offering within the CloudFoundry platform. The services offered were typically those that an application/service developer needed but did not want to maintain or implement themselves. By not carrying out these tasks, developers received great productivity gains and reduced technical debt. The effort around the OSBA began on December 13th 2016 and was based on the work already used within CloudFoundry. The initial list of contributors to the effort included Fujitsu, Google, IBM, Pivotal, Red Hat and SAP. This gave a large amount of weight to

[45] Docker, http://docker.com
[46] MongoDB, http://mongodb.com
[47] MySQL, http://mysql.com
[48] OpenStack Keystone, https://docs.openstack.org/keystone/latest/
[49] Open Service Broker API, http://openservicebrokerapi.org
[50] Open API Initiative, https://www.openapis.org
[51] Cloud Foundry, https://www.cloudfoundry.org/
[52] Open Shift, http://openshift.com

the effort and along with support from Kubernetes[53] in the Cloud Native Computing Foundation[54], it is currently and arguably the *defacto* standard API in delivering services.

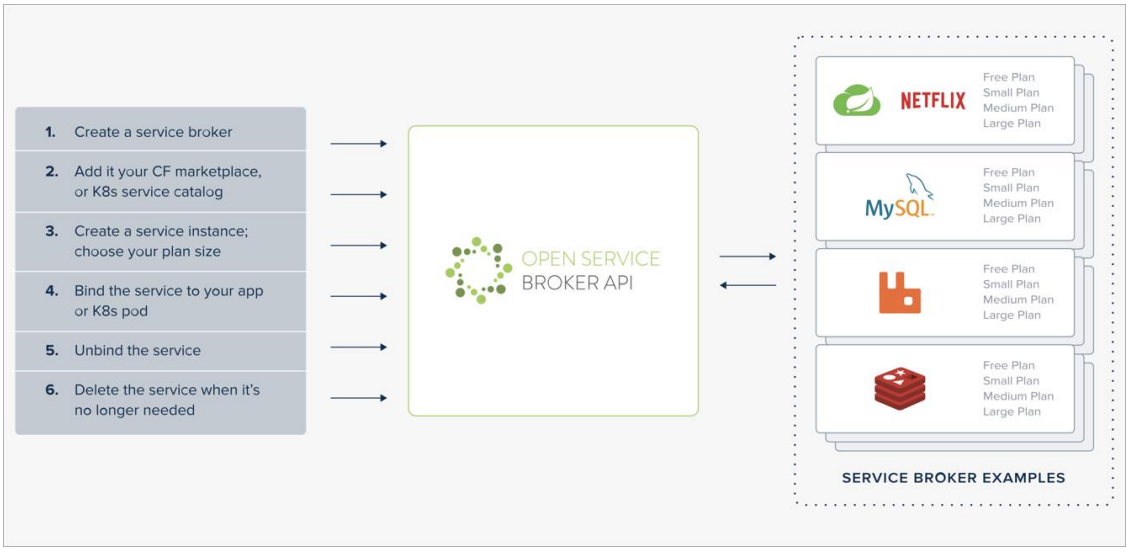

Figure 36. Open Service Broker API (OSBA) overview

Within the API there are two key entities that are operated upon over the REST-based API; Service Type and Service Instance. Service Types are registered with OSBA implementations in their catalog and from there they are queryable by consumers. Each Service Type has one or more Plans associated with it, enabling charging per unit-item (typically time-based, but others can be used). Once a consumer has identified what Service Type it would like to use in its application a provisioning request (creation) is issued against the API. The assumption with the OSBA is that once the instantiation request is complete the Service Instance is ready for use. If the provisioning time exceeds a service developer time period (this is not specified in the specification and is left to the developer to set. Obviously the faster a service instance can be created the better it is.), then it is expected that the client issue an asynchronous request by supplying the 'accepts_incomplete' query parameter set to 'true' to the call. Once the Service Instance has been provisioned it can be then used by the details available from querying the service instance. Some Service Instances will require that a final step be carried out to use the Service Instance. This step is known as Service Binding and typically involves the on-demand generation of credentials to access the Service Instance's functionality. Once a Service Instance is no longer required it can be unbound (unbind, rebinding is also possible) and then deprovisioned (destroy). Naturally, there are further fine grained details on the OSBA which can be found in the specification[55].

[53] Kubernates, http://kubernetes.io

[54] Cloud Native Computing Foundation, https://www.cncf.io

[55] OSBA specification, https://github.com/openservicebrokerapi/servicebroker/blob/v2.13/spec.md

5.1.2.2 OSBA & Billing

Each service type that is registered in the catalog has one or more plans associated with it. Each of these plans per specification contains information on what business level aspects are provided with the associated Service Type. In order to enable the ElasTest Cost Estimation Engine (ECE), the ECE billing model is integrated with the Plan entity (see below in ESM Data Model). This is done so that an initial estimation of costs for the whole set of TSSs in a TJob can be calculated. This calculation is a static one and with further integration of a billing engine with the ESM will enable dynamic billing and actual cost reporting per service instance per TJob.

5.1.2.3 ESM Data Model

Below is the data model used in the OSBA and the OpenAPI definition of the specification can be found at the ESM repository[56].

[56] ESM Repository, https://github.com/elastest/elastest-service-manager/blob/master/docs/images/datamodel/sm-datamodel.png

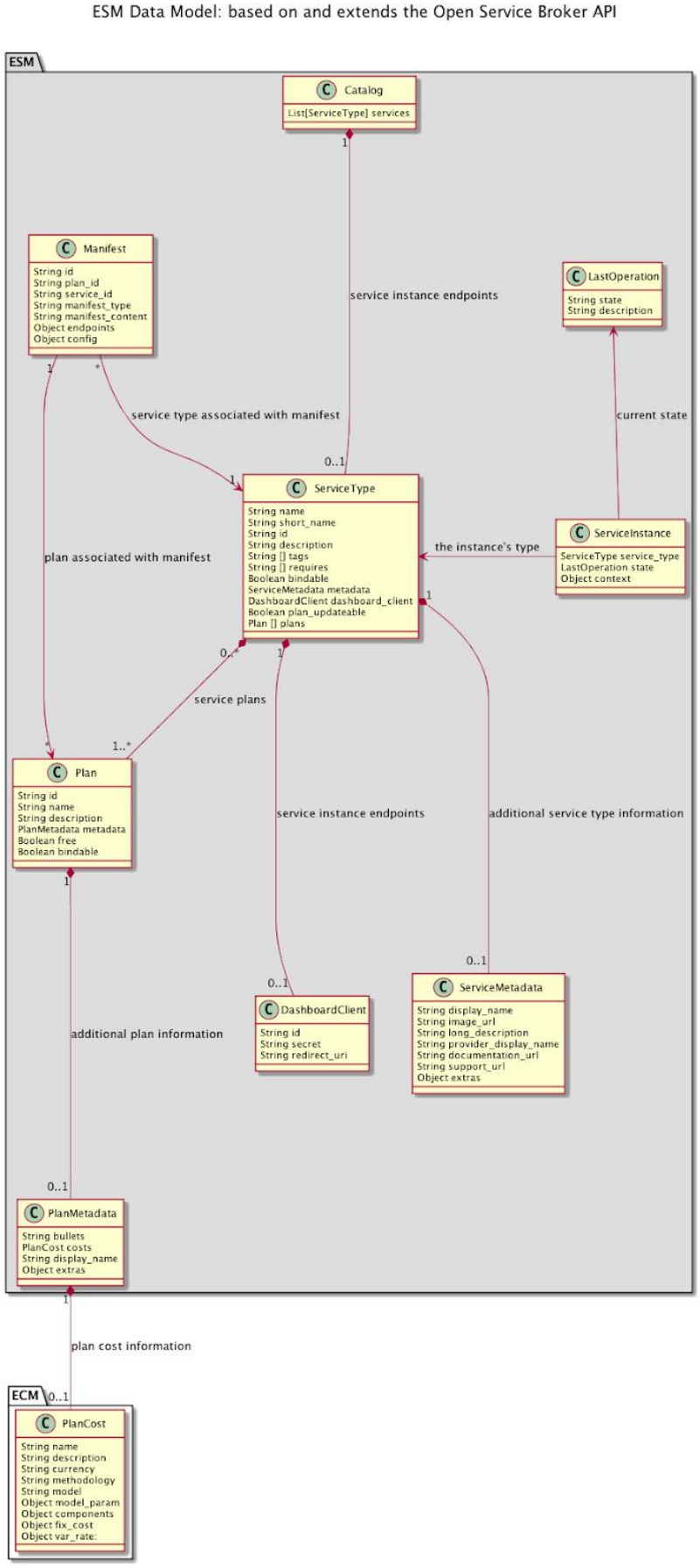

Figure 37. ESM Data Model

5.1.3 Component Design and Architecture

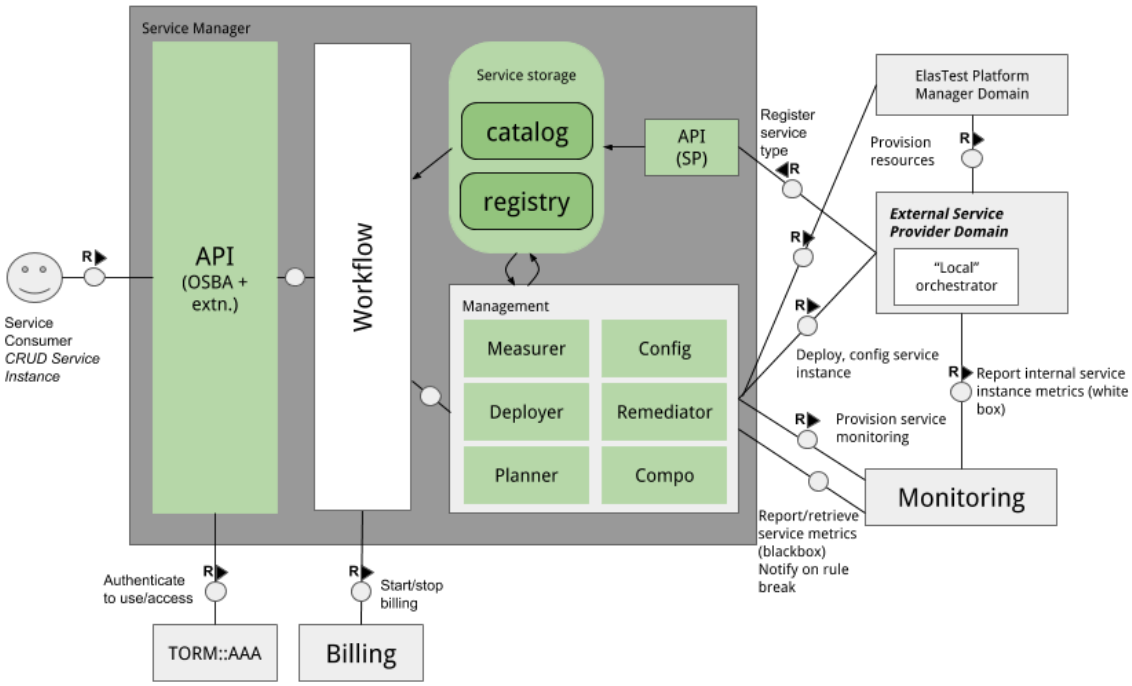

Figure 38. ESM FMC Diagram

There are a number of components within the ESM and here we provide a brief explanation of each. The internal components of the ESM are briefly described here

- **API (OSBA, extensions) & API (SP)**: This is the main service consumer entry point. The API allows the creation and management of service instances. The second API is one that is merged with the OSBA API. This is the API that allows service providers to register and update their service offerings within the SM. This is the representation of extensions. **[Implemented]**
- **Workflow**: manages the interactions between the various functional components of the service manager. **[Currently Not implemented]**
- **Catalog**: query available services, services offered by the SM. The catalog is a collection of service types that can be instantiated on demand by the ESM. **[Implemented]**
- **Registry**: query service instances currently managed by the SM. **[Implemented]**
- **Measurer**: measures service metrics and reports them to the EMP **[Basic Implementation]**
- **Deployer**: requests creation of service, adaptors to "local orchestrator" are provided by the ESM, including Docker and EPM. **[Implemented]**
- **Config**: configures a created instance.
    - **[Implemented]**. Note that configuration of the service instance can currently happen in two ways:
        - overriding configuration parameters on provisioning
        - retrieval of service instance API and configuration supplied, if

and only if the API allows it

Updating a service instance (configuration or plan) **[Currently Not implemented]**

- **Remediator**: if notified of plan/SLA breach, the remediator takes necessary actions to resolve the problem. **[Currently Not implemented]**
- **Compo**: responsible for composition of services and resolution of dependencies a service has. **[Currently Not implemented]**
- **Planner**: plans deployment, Can also select the best service provider if there's none specified or the best location based on technical requirements (latency, geo-location, etc.). Also ensures that if there are dependencies that they are selected for deployment. **[Currently Not implemented]**

The ESM has a number of interactions with external entities. A brief description of these is provided here:

- **ElasTest Platform Manager (EPM)**: If service provider uses own resources then this is optional. For the case of ElasTest it is mandatory to use the EPM.
  - Status: **Integrated**.
- **AAA**: not provided by ElasTest. Currently uses OpenStack Keystone[57].
  - Status: **Integrated**.
- **Billing**: not provided by ElasTest. Currently provided by Cyclops[58].
  - Status: **Not integrated**.
- **Monitoring**: monitors service from external perspective e.g. response time, latency, RTT etc.
  - Status: **Integrated**.

Note: The "Local" Orchestrator can be provided by service owner. It should be the EPM however; other orchestrators can be supported by the ESM architecture. The main goal is the orchestration of services and dependencies. This abstraction allows the service provider either use its own system to provide the service's resources (for example using the local docker engine driver) or to use those by the EPM.

5.1.3.1 Service Lifecycle

A service that is offered through the ESM uses the common life cycle as defined by the ESM. It is a simple lifecycle and only accounts for the technical realisation of the service. The lifecycle includes all phases from the design of the service through to the disposal of a service instance. The lifecycle is one that has already been used in the Hurtle orchestrator. The phases of the life cycle are as follows:

[57] OpenStack Keystone, https://docs.openstack.org/keystone/latest/

[58] Cyclops, https://icclab.github.io/cyclops/

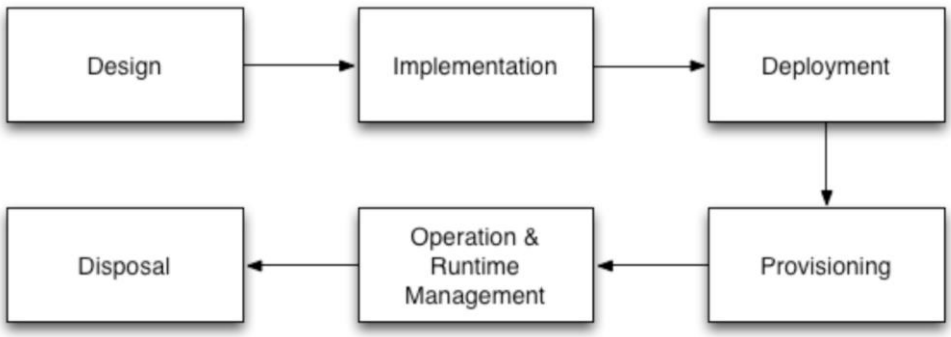

Figure 39. ESM Lifecycle

- **Design**: Design of the architecture, implementation, deployment, provisioning and operation solutions. This allows the owner of the service "design" their service. This generally a human-oriented activity.
- **Implementation**: of the designed architecture, functions, interfaces, controllers, APIs, etc. It should be noted here that all TSS APIs are specified using the OpenAPI interface description language.
- **Deployment**: Deployment of the implemented elements, e.g. networks, volumes and containers, etc. Anything such that the service can be used, but don't provide access to the service. With respect to the OSBA, this phase is associated with the provisioning phase.
- **Provisioning**: Provisioning of the service environment. Activation of the service such that the user can actually use it. This in the specific case of ElasTest this means configuring the service with the required parameters. These parameters are supplied by passing environment variables into the container.
    - Once the service is appropriately configured the owner of the service instance can then request access to the service instance through “binding”. This part of the provisioning phase is specific to the Open Service Broker API.

  With respect to the OSBA, this phase is associated with binding as well as the specific case of service configuration.
- **Operation and Run-Time Management**: in this stage the service instance is ready and running. Activities such as scaling, reconfiguration of Sub-Services are carried out here. This phase is currently a service-specific task.
- **Disposal**: Release of all SSs, the service instance itself and virtual resources is carried out here. With respect to the OSBA, this phase is associated with both unbinding and deprovisioning.

5.1.3.2 Use case scenarios

Please see D2.3.[5] for the use case scenarios that the ESM is designed and implemented to satisfy. However, for reference in the following sequence diagrams the following use cases are considered for the ESM Actors:

1. ETM and *ServiceManagerUI* are all types of *ServiceConsumer*. A ServiceConsumer is the client side of the business relationship and the user of the functionality delivered by the ESM.
2. *ServiceProvider* is a subclass of *ServiceConsumer* as it has not only access the functionality of a consumer but also adds basic functionality required to operate a service. The provider is the entity that designs, implements and operates a service type upon a particular platform. The provider offers its service type (implementation) to consumers, who can create instances that are billed.

Table 11. Set of ESM Use Cases and their Implementation Status

#	ServiceConsumer	ServiceProvider
1	list available service types	list available service types
2	create a service instance of a specific service type	register service type and endpoint information
3	get/poll service instance status	register service type manifest, **implemented via ID #2**.
4	bind service instance	update service type business information: plan and description (this is the same technical implementation as **ID #5**)
5	configure service instance	update service type technical information: endpoint/API (this is the same technical implementation as **ID #4**)
6	get service instance details, **implemented via ID #3**	report service instance metrics
7	get service instance metrics	
8	update service instance	
9	unbind service instance	
10	delete service instance	

In the table above, the key use cases are listed and those that are already implemented are coloured in green. Those that are coloured in orange are either in progress or planned to be implemented in upcoming releases of the ESM and as such these items are not included in the set of ESM sequence diagrams.

5.1.3.3 Sequence Diagrams

In D.2.3.[5], the sequence diagrams related to the ESM only showed the interactions with other external components and services. As such it provides a black box view. There are further details that are illustrated in sequence diagrams that show the internal details of the ESM or in other words the "white box" view. The sequence diagrams are grouped by base actor (*ServiceConsumer* and *ServiceProvider*) and are included in the appendix of this document (see section 10.1).

5.1.4 Roadmap and Features

From the ElasTest DoA [1] the following features were expected to be delivered by the ESM:

- *Service Manager components and architectural models, interfaces*
  - These are described in this document.
- *Seamless installation and onboarding of services*
  - This has been achieved by firstly deciding on what the requirements for service owners were (this is done through WP5 and are reported in D5.1). The ESM has support for onboarding service through the API or through the user interface should a more user friendly means be required.
- *Common life cycle for all services*
  - A common life cycle has been defined by the ESM and followed by all services in WP5. Currently, this lifecycle is a simple one and does not cover the case of online service upgrades. *done: explain the Mobile Cloud Networking (MCN) project[59] lifecycle and the required extensions to it (future work)*
- *Monitoring, observation and debugging capabilities*
  - Currently, the ESM has Integration with EMP. Service-level metrics are measured per service instance and then reported to the EMP. The EMP then provides the time-series storage required for the submitted metrics and through the EMP's interface those are retrieved on request made available to the ESM (e.g. showing service instance metrics in the ESM UI)

Here we list the set of features currently available in the current version of the (0.9 as of writing) ESM.

5.1.4.1 User Interface

The key aim of the user interface has been to expose the ESM's API in a visual fashion and also provide metrics per service instance. Those metrics are retrieved from the EMP.

- All API functionality is exposed through the user interface.
- On-boarding of services enabled through UI
- Visualisation of service and resource metrics

The UI itself is an independent code base and is deployed in a separate container. This allows those that operate an ESM to select to use the user interface or not.

[59] Mobile Cloud Networking, http://mobile-cloud-networking.eu/, EU FP7 grant agreement #318109

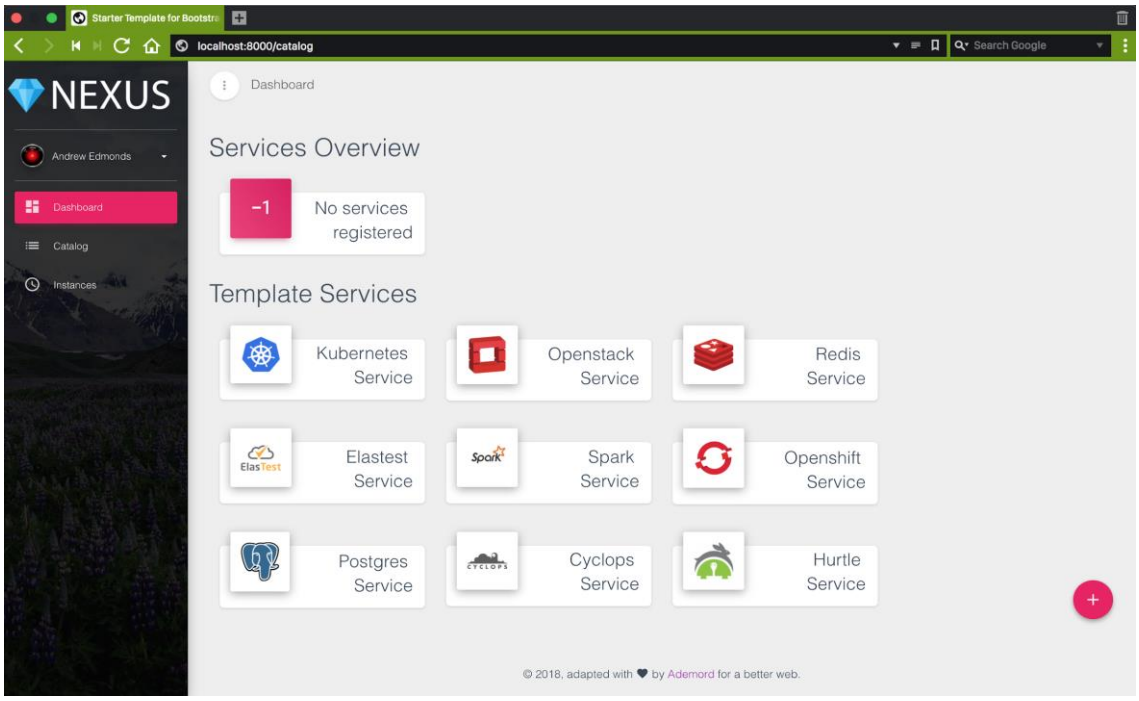

Figure 40. ESM: A listing of services available in the Service Catalog

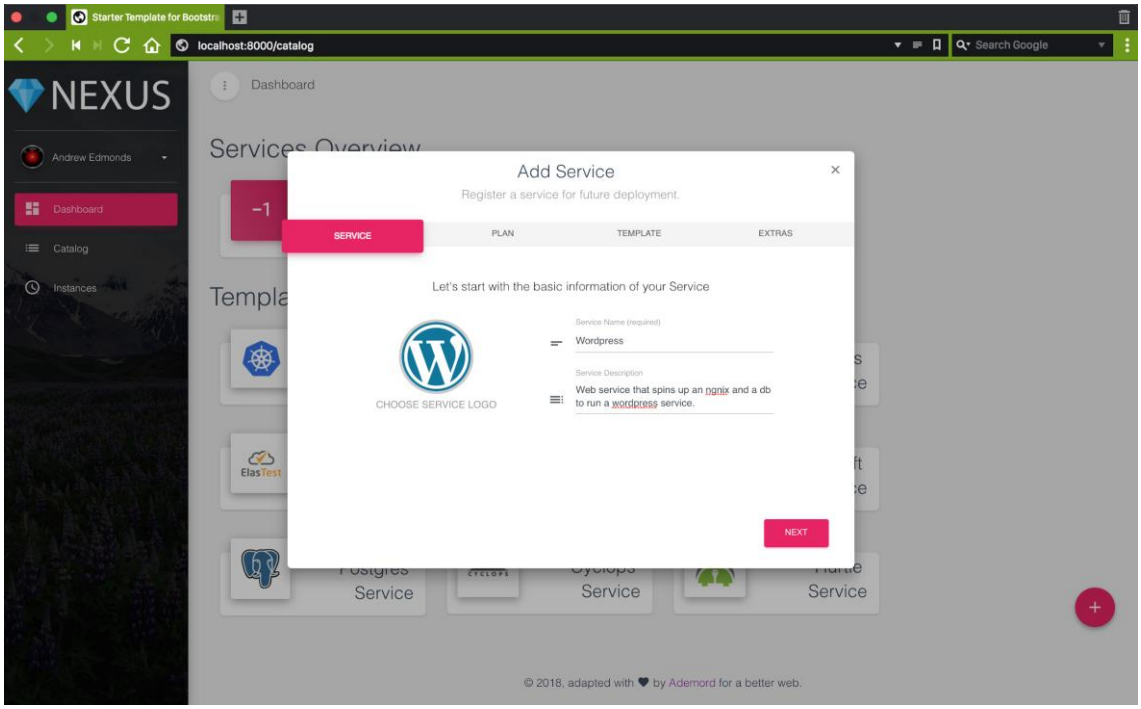

Figure 41. ESM: Add Service

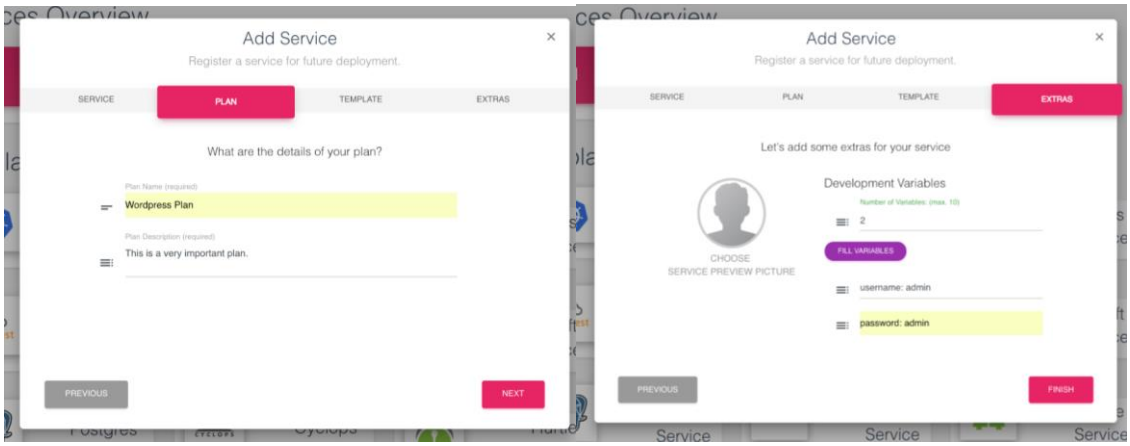

Figure 42. ESM: Onboarding a new Service Type

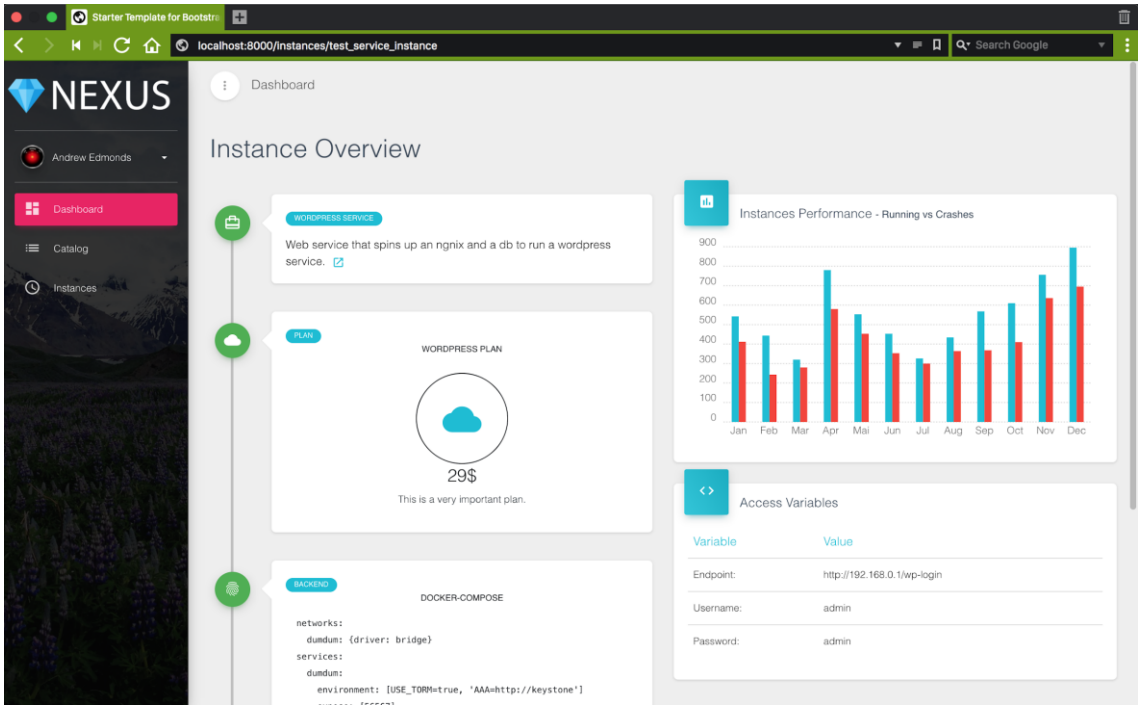

Figure 43. ESM: Viewing a Serice Intance Details

5.1.4.2 Road Map

Below are a set of features that should be implemented in the ESM. Which feature has more priority is subject to the ElasTest release planning meetings.

- Currently the provisioning of services is somewhat synchronous. Complete asynchronous handling of these requests will be provided in upcoming releases.
- The monitoring of services is currently basic (it simply gathers metrics based on each service instance's health endpoint). These metrics will be extended to provide further enhanced service monitoring.
- Updating a service instance (configuration or plan) is yet to be implemented and will be done to provide a type of "static" scalability. In this case, the service

consumer/owner can change the plan (e.g. resources assigned to a service) and so increase the processing capacity of the service instance. This change needs to be transparent to the service owner and not result in any data loss.

- One request by service owners is to have a means to debug service instances. This will be investigated to provide a solution that is useful to WP5 service owners.
- At the moment there are only two backends implemented. In order to provide further tenant isolation and control from the ESM perspective other resource backends (e.g. OpenShift/Kubernetes) should be provided
- Currently there are a number of components in the logical architecture that have no implementation (including the Planner, Composition, Remediator and Workflow). Depending on requirements and needs of the overall platform these will be implemented either as core or investigated in the research areas.
- Billing integration to compliment the cost estimation engine (ECE).
- Update to OSBA API version 2.13, or if available v2.14.

5.1.4.3 Code Reports

The ESM is implemented using python and this code is tested using the Python unittest module. There are currently 51 tests that are run against all configurations of the ESM. These tests consist of both unit and integration tests. Currently the code coverage is at approximately 87%, which is calculated using Codecov.io[60] every time a new build of the ESM is done. As shown, the code coverage for the ESM for the last 6 months has remained stable. Note that currently code complexity is not calculated.

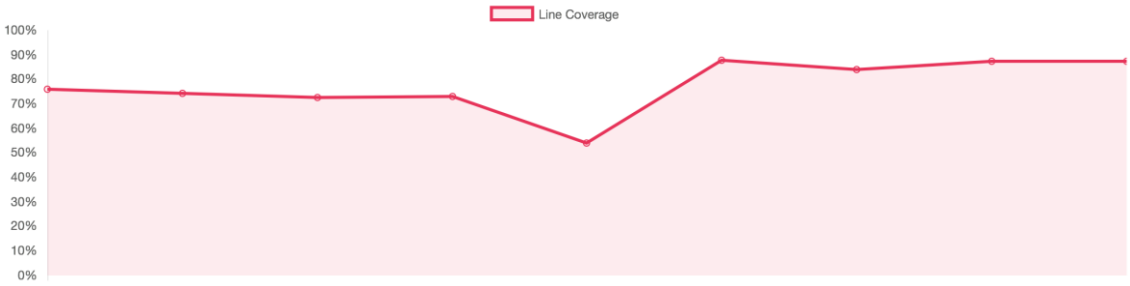

Figure 44. ESM Code Coverage over Time

The status and history of all the ESM builds can be viewed at this page[61], which is the public view of the ElasTest continuous integration and delivery system (based on Jenkins).

5.1.4.4 Code Repository

- The ESM code repository can be found on GitHub[62] and is licensed using

[60] Code Coverage, https://codecov.io

[61] ESM status build, https://ci.elastest.io/jenkins/view/Components%20build%20dashboard/job/elastest-service-manager/job/esm/

Apache 2.0.

- Within that repository, there is documentation[63] detailing how to run, use and extend the ESM.
- The API of the ESM[64] can be viewed online here[65].

5.1.4.5 API

The API of the currently versioned ESM is based on the v2.12 OSBA specification. The functionality that is currently available is:

- OSBA: list the contents of the service catalog.
- OSBA: provision (create) a service instance.
- OSBA: bind (configure) a service instance.
- OSBA: unbind (unconfigure) a service instance.
- OSBA: deprovision (delete) a service instance.

In order to integrate the OSBA API with ElasTest and fit its needs the following extensions were provided (uses the REST path namespace of `/v2/et`):

- ElasTest Extension: get details on a service instance.
- ElasTest Extension: register a service in the service catalog.
- ElasTest Extension: register a service manifest associated with a service description.

Below is a summary of the functionality exposed by the ESM's API. Extensions to the OSBA are kept separate from standard API and are placed under the `/v2/et` namespace:

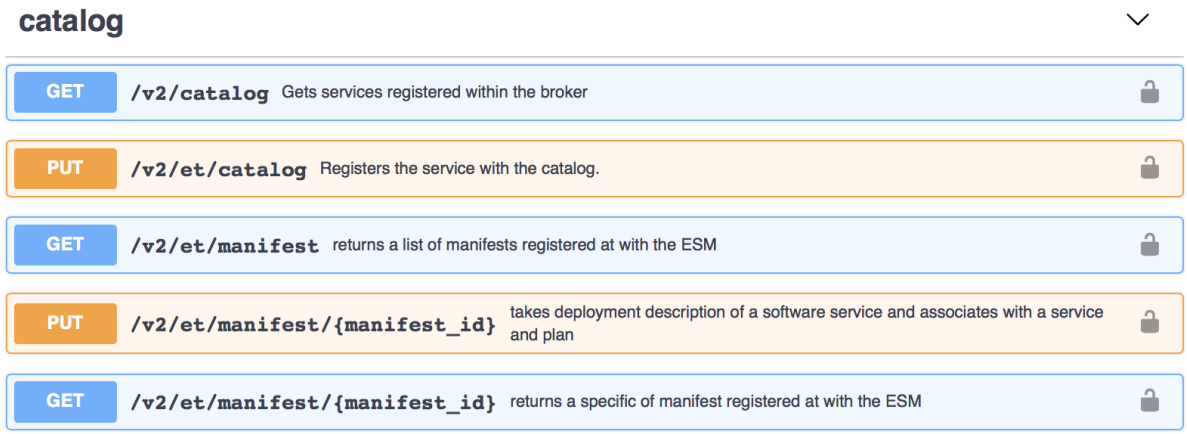

Figure 45. ESM: The Catalog API

[62] ESM GitHub repository, https://github.com/elastest/elastest-service-manager
[63] ESM documentation, https://github.com/elastest/elastest-service-manager/tree/master/docs
[64] ESM API YAML, https://github.com/elastest/elastest-service-manager/blob/master/api.yaml
[65] ESM API, https://elastest.io/docs/api/esm/

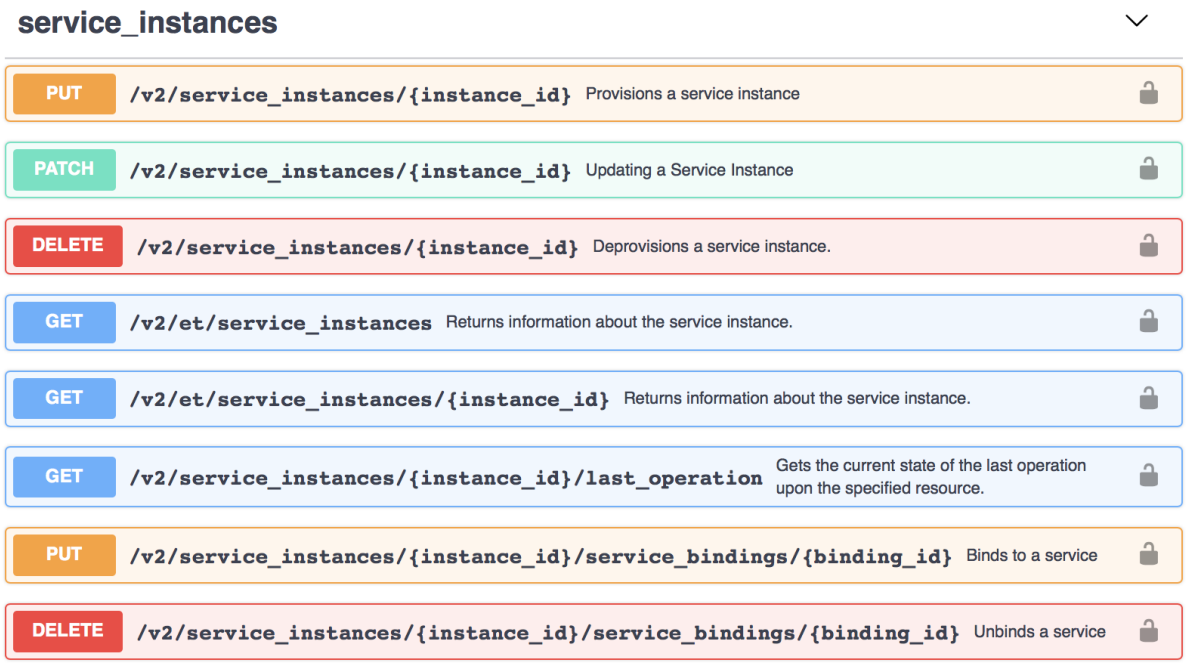

Figure 46. ESM: API Related to Service Instances

5.1.4.6 Internal

- Support for in-memory, MongoDB and MySQL persistency through the Store abstraction
- Support for local docker and EPM resource providers through the Resource Backend abstraction.
- Support for service level metrics and reported to the EMP, through the basic Measurer.
- Support for providing ESM logging direct to the EMP using the standard python logging interface[66].

5.1.5 Research Results and Future Plans

Currently the research work of the ESM has been submitted to ICDCS18[67], USENIX HotEdge18[68] and also to QUATIC18[69]. The work in the former two is around the idea of providing near real-time provisioning of services using an approach that combines technologies of *Unikernels* and can be applied to *Edge computing*. The latter provides an overview of how the ESM supports the TSSs of WP5.

Currently, the ESM is dependent upon docker-compose description documents and can support others, however a common model that can accommodate different technologies may be researched in order to provide a single document that describes a

[66] Python logging interface, https://docs.python.org/3/library/logging.html
[67] ICDCS18, http://icdcs2018.ocg.at
[68] Hot Edge, https://www.usenix.org/conference/hotedge18/
[69] QUATIC, https://sites.google.com/view/quatic2018/

service. Currently, the only standard available that can provide this is the OASIS TOSCA[70] specification, so attention this this shall be given.

Having a service description format that allows specification of location is becoming more important with the advent of Edge and Fog Computing, however it is arguable that the platform should or could (without the explicit specification of the programmer) manage the placement of service components. Another area of interest and worthy of research is how one can integrate other service delivery technologies, such as serverless and Function-as-a-Service technologies. This also has a relationship to service composition, which can also be investigated.

Finally, given the Plan-based means to bill a service instance, work upon how the integration of the EMP and SLAs (essentially the Plan of a service) can be achieved such to maintain the level of service.

6 SuT Management

6.1 ElasTest Instrumentation Manager (EIM) & Instrumentation Agents

6.1.1 Introduction

Instrumentation is a collective term used for measuring instruments, which is the activity of obtaining and comparing established standard objects and events (used as units), the process of measurement gives a number relating the item under study and the referenced unit of measurement.

The term has been used since long time ago in different contexts depending on the sectors where it applies, from the pre-industrial period to nowadays where we can find large integrated computer-based systems instrumented. In the context of computer programming, instrumentation refers to extending the interface exposed by a software system for achieving enhanced controllability (i.e. the ability to modify behaviour and runtime status) and observability (i.e. the ability to infer information about the runtime internal state of the system). Programmers implement instrumentation in the form of code instructions that monitor/control specific components in a system.

Tasks 3.3 & 3.4 are in charge of designing and implementing Instrumentation Agents, as described in Section 1.3.2 on PartB of the DoA [1]. These Agents instrument the operating system of the SuT host instances. Thanks to it, the agent is capable of exposing two types of capabilities: (1) controllability, through which the agent can force custom behaviours on the host's network, CPU utilization, memory consumption, process lifecycle management or system shutdown, etc.; and (2) observability, through which the Agent collects all information relevant for testing or monitoring purposes (e.g. energy consumption, resources utilization, etc.) For this, these tasks shall:

• Develop the Instrumentation Agent, that shall work at least, on the Linux kernel (other operating systems might also be considered).

[70] OASIS TOSCA specification, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

• Implement the mechanism for the registration of the Agent into the Instrumentation Manager. This enables the Manager to control the Agent and the Agent to provide status information into the Manager.

• Develop the appropriate technologies enabling the installation and control of the Agent on different cloud infrastructures including VM (e.g. KVM) or Containers (e.g. Docker). These technologies should guarantee availability of the tools on all public or private clouds, distributions and kernel versions that the partners of the project or the community of users demand.

6.1.2 Baseline Concepts and Technologies

Instrumentation within ElasTest refers to extending the interface exposed by a software system for achieving enhanced **controllability** (i.e. the ability to modify behaviour and runtime status) and **observability** (i.e. the ability to infer information about the runtime internal state of the system).

Previous works have been carried out by ATOS team on the field of "instrumentation observability". In the context of the FP7 EU funded project CloudWave[71], where a fundamental topic is the 'execution analytics' which deals with the gathering, processing, and analysis of the vast amounts of monitoring data available in the Cloud, the ATOS team worked with a toolbox of technologies developed in the project with the objective of instrumenting cloud applications using Aspects. (i.e.: using AspectJ[72] a seamless aspect-oriented extension to the Java programming language).

Table 12. EIM: Baseline technology.

Input technology	Input TRL	Description
Cloud Interceptor (ATOS-CI)	4	Provided by ATOS. It is as toolbox of technologies developed in CloudWave project with the objective of instrumenting cloud applications using Aspects

However, after the first user feedback loop at the design phase, ElasTest pilots have been arguing that they prefer not to modify their applications source code adding code annotations in order to be able to observe applications behaviour at runtime, for this reason the manager and agents actuates at OS level consuming well established operating system interfaces to guarantee interoperability across distributions and versions.

ElasTest Instrumentation Manager software module has been implemented in Java and has been packed as Docker image which facilitates the deployment over different resources topologies. The aforementioned module is part of the ElasTest Platform core-components, so it is deployed together with the platform through the platform docker-compose file.

The development environment used for this component looks as follows:

[71] CloudWave – Agile Service Engineering for the Future Internet, http://www.cloudwave-fp7.eu/
[72] AspectJ project, https://www.eclipse.org/aspectj/

- JAVA JDK >= 7
- Maven >=3
- Eclipse IDE or similar JAVA development environment.

If the EIM is used outside the ElasTest platform additional systems are deployed to expose the features it offers:

- MySQL-like DB[73] or MongoDB[74]**:** The Database System that uses the EIM to store the persistent data, necessary to manage the Agents. Within ElasTest, the EIM is using the EDM component to persist data.
- Elasticsearch[75]: As indicated on its website "is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases. As the heart of the Elastic Stack, it centrally stores your data ".
- Logstash[76]: As indicated on its website "It is a server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to your favorite stash". EIM uses it to gather and parse logs and metrics produced in the SuT where Beats are deployed. The logs and metrics are sent to Elasticsearch.
- Kibana [77] : As indicated on its website "Kibana lets you visualize your Elasticsearch data and navigate the Elastic Stack". EIM uses it to visualize the data collected from Beats from SuT.
- SuT: In order to provide something to test with, an Ubuntu14.04 container is provided to interact with EIM Server Application.

The EIM software module is licensed under Apache License, Version 2.0 [3].

6.1.3 Component Design and Architecture

The ***Instrumentation Agent*** consists of a software agent that instruments the operating system of the computing nodes (i.e. virtual machines, containers, etc.) where the SuT is deployed. This agent makes it possible to customize all the resources under the control of the node's operating system kernel (e.g. network stack behaviour, CPU utilization, node shutdown, etc.) In addition, the agent collects information on node behaviour including metrics for performance, resource consumption, energy, etc. This is compatible with all types of cloud technologies as it only requires installing and launching the agents on the nodes where the SuT is deployed.

The ***Instrumentation Manager*** consists of a service that can be launched in the same cloud where a SuT is deployed or together with the platform. The Instrumentation Manager acts as a registrar of the different Instrumentation Agents in the sense that all agents in the SuT register into the manager as they start up. After that, the Manager exposes to testers the ability of controlling agent's behaviour through a REST API. The

[73] MySQL-like DB, https://www.mysql.com/
[74] MongoDB, https://www.mongodb.com/
[75] ElasticSearch, https://www.elastic.co/products/elasticsearch
[76] Logstash, https://www.elastic.co/products/logstash
[77] Kibana, https://www.elastic.co/products/kibana

ElasTest **Instrumentation Manager (EIM)** component controls and orchestrates the Instrumentation Agents that are deployed in ElasTest[78] platform. These agents will instrument the operating system of the SuT (Software under test) host instances.

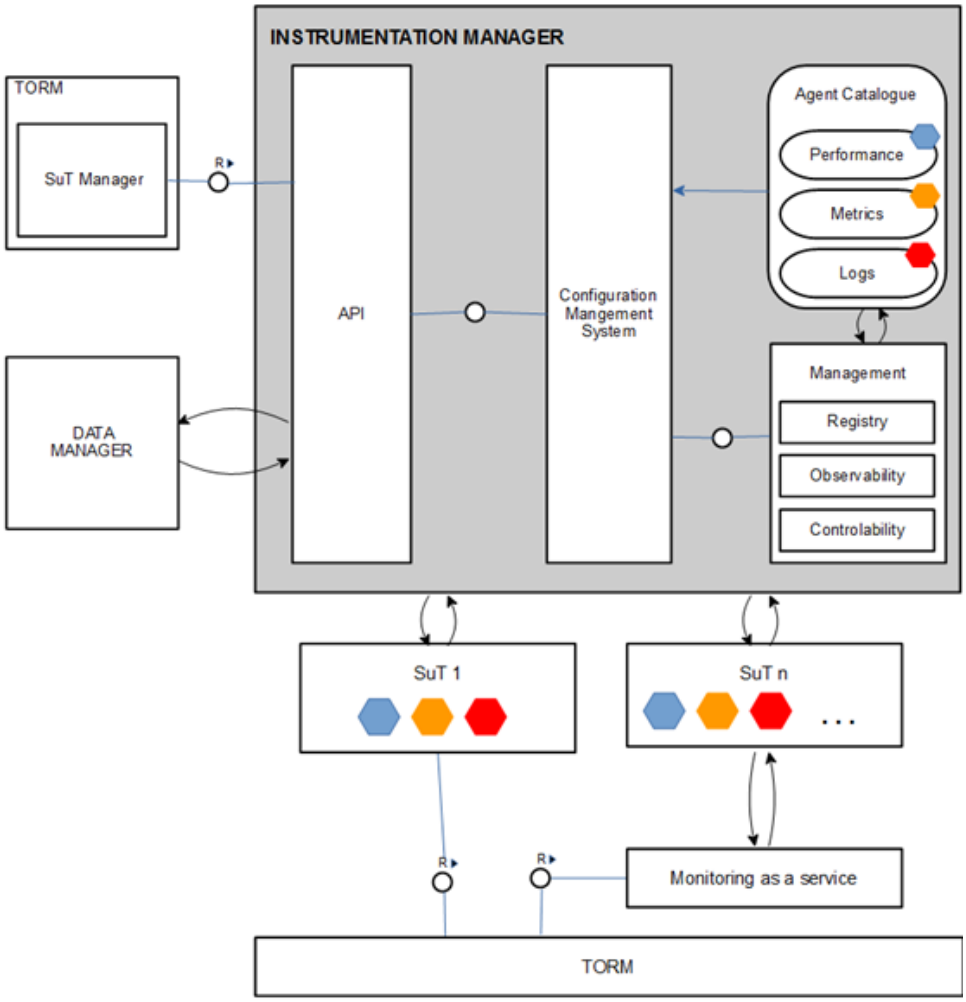

Figure 47. EIM FMC Diagram

Here we provide a brief explanation of each internal module of the EIM:

- **API**: This is the main SuT manager entry point (northbound interfaces exposed to TORM module). The component exposes a REST API that allows the registration, configuration and deployment of the agents on top of the resources to monitor or control.
- **Conf. Management System:** The configuration management system is able after the registration of a new target host to download, install and configure the agents selected by the users/testers.
- **Agent Catalogue:** This catalogue contains the list of actions supported by the EIM, the current version of the component includes mainly observability actions, and the controllability ones will be part of the second report.

[78] ElasTest Plattform, http://elastest.io/

- **Management:** Implements the instrumentation capabilities offered by the software module.

6.1.3.1 Data Model

6.1.3.1.1 PublicKey

Table 13. EIM: PublicKey Data Model

Name	Description	Schema
publickey	This entity defines the public Key of EIM Example : `"ssh-rsa AAAAB…"`	string

6.1.3.1.2 Agent

Table 14. EIM: Agent Data Model

Name	Description	Schema
Agent ID	This entity defines the ID of the agent	string
host	Target host	String

6.1.3.1.3 Host

Table 15. EIM: Host Data Model

Name	Description	Schema
address	Host endpoint Example : `"127.0.0.1"`	string
User	Username Example : `"root"`	String
private_key	This is the key itself as String.	String
logstash_ip	Logstash endpoint Example : `"127.0.0.1"`	String
logstash_port	Logstash port Example : `"5044"`	String

6.1.3.1.4 AgentConfiguration

Table 16. EIM: AgentConfiguration Data Model

Name	Description	Schema
exec	This entity defines the ID of the agent	string
Component	Target host	string
network_agent	Stream and path	string

logging_agent	Stream	string
performance_agent	Stream	string

The figure bellow depicts the technology map used in the reference implementation of the Instrumentation Manager (EIM) working in isolation of the ElasTest platform:

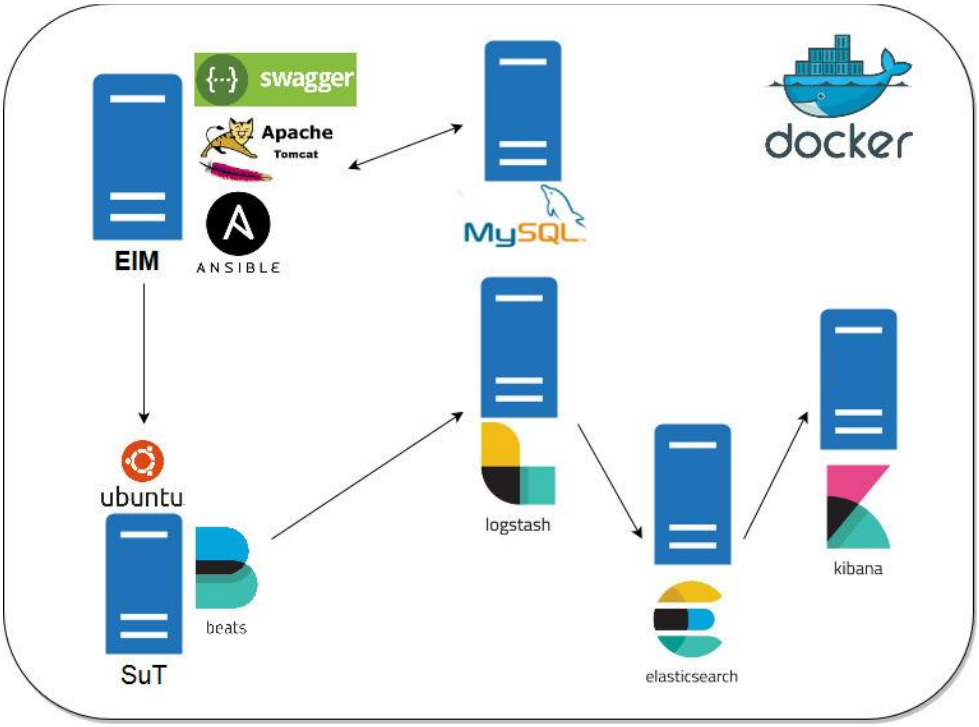

Figure 48. Instrumentation Manager & Agents technology map.

6.1.4 Roadmap and Features

6.1.4.1 Component Usage from the ElasTest dashboard:

Instrument SuT from ElasTest dashboard: (Register agent step)

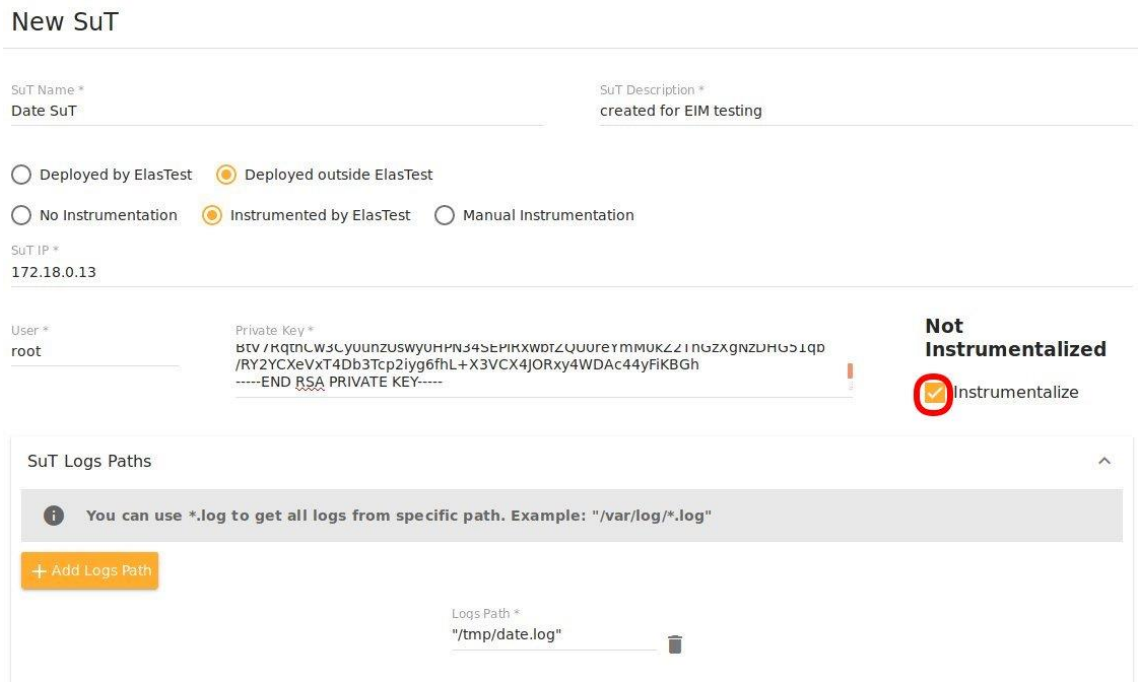

Figure 49. Instrumenting SuT from ElasTest dashboard

Install and configure agents on remote SuT:

```
2018-04-24 13:59:12 INFO  TemplateUtils:315 -
2018-04-24 13:59:12 INFO  TemplateUtils:315 - TASK [METRICBEAT - Configure Logstash and file output] ************************
2018-04-24 13:59:12 INFO  TemplateUtils:315 - changed: [172.18.0.13]
2018-04-24 13:59:12 INFO  TemplateUtils:315 -
2018-04-24 13:59:12 INFO  TemplateUtils:315 - TASK [FILEBEAT - Restart filebeat] ***************************************
2018-04-24 13:59:13 INFO  TemplateUtils:315 - changed: [172.18.0.13]
2018-04-24 13:59:13 INFO  TemplateUtils:315 -
2018-04-24 13:59:13 INFO  TemplateUtils:315 - TASK [PACKETBEAT - Restart packetbeat] *************************************
2018-04-24 13:59:14 INFO  TemplateUtils:315 - changed: [172.18.0.13]
2018-04-24 13:59:14 INFO  TemplateUtils:315 -
2018-04-24 13:59:14 INFO  TemplateUtils:315 - TASK [METRICBEAT - Restart metricbeat] *************************************
2018-04-24 13:59:15 INFO  TemplateUtils:315 - changed: [172.18.0.13]
2018-04-24 13:59:15 INFO  TemplateUtils:315 -
2018-04-24 13:59:15 INFO  TemplateUtils:315 - PLAY RECAP *********************************************************************
2018-04-24 13:59:15 INFO  TemplateUtils:315 - 172.18.0.13               : ok=19   changed=18   unreachable=0    failed=0
2018-04-24 13:59:15 INFO  TemplateUtils:315 -
2018-04-24 13:59:15 INFO  TemplateUtils:320 - exit: 0
2018-04-24 13:59:15 INFO  AgentApiServiceImpl:223 - Successful execution for the beats script generated to agent iagent1
2018-04-24 13:59:15 INFO  EimDbAgentCfgManager:48 - Connecting to EIM database...
2018-04-24 13:59:15 INFO  EimDbAgentCfgManager:51 - Connected to EIM database successfully...
2018-04-24 13:59:15 INFO  EimDbAgentCfgManager:120 - Adding new agent configuration to DB for agent with agentId = iagent1
2018-04-24 13:59:15 INFO  EimDbAgentCfgManager:134 - Agent Configuration inserted in database wiht agentId = iagent1
2018-04-24 13:59:15 INFO  EimDbAgentCfgManager:152 - Searching agent configuration for agent with agentId = iagent1
2018-04-24 13:59:15 INFO  EimDbAgentCfgManager:164 - Agent configuration finded in DB for agent with agentId = iagent1 with ID iagent1
2018-04-24 13:59:15 INFO  EimDbAgentManager:47 - Connecting to EIM database...
2018-04-24 13:59:15 INFO  EimDbAgentManager:50 - Connected to EIM database successfully...
2018-04-24 13:59:15 INFO  EimDbAgentManager:437 - Setting agent iagent1 monitored = true
2018-04-24 13:59:15 INFO  EimDbAgentManager:446 - Agent modified in database wiht agentId = iagent1
2018-04-24 13:59:15 INFO  EimDbAgentManager:216 - Searching host in DB with agentId = iagent1
2018-04-24 13:59:15 INFO  EimDbAgentManager:233 - Host finded in DB with agentId = iagent1 with ID iagent1
2018-04-24 13:59:15 INFO  AgentApiServiceImpl:228 - iAgent iagent1 monitored succesfully
2018-04-24 13:59:15 INFO  AgentApiServiceImpl:230 - postAction method with monitor param for agent iagent1 OK response: class AgentFull {
    agentId: iagent1
    host: 172.18.0.13
    monitored: true
    logstash_ip: 172.18.0.8
    logstash_port: 5044
}
```

Figure 50. Install and configure agents on remote SuT

Select relevant KPIs from the metrics catalogue:

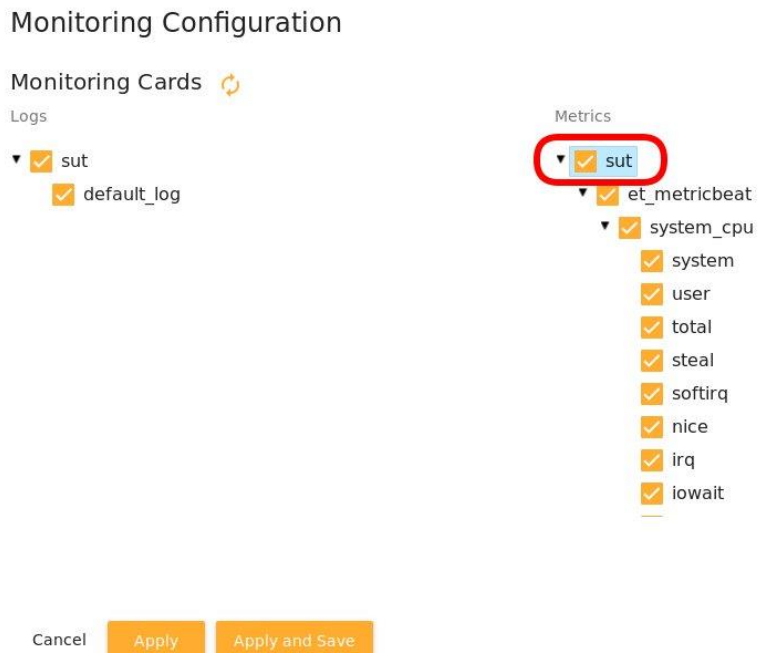

Figure 51. Select KPIs to monitor

Metrics and logs visualized from the platform dashboard:

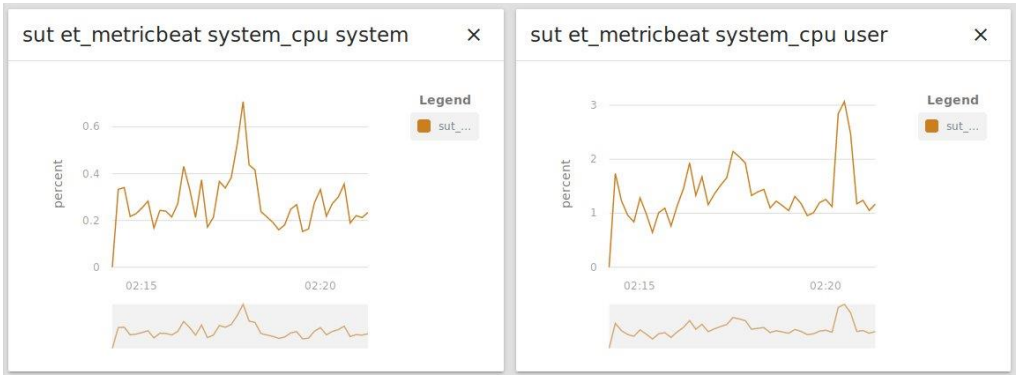

Figure 52. Metrics visualisation within the ElasTest dashbaord

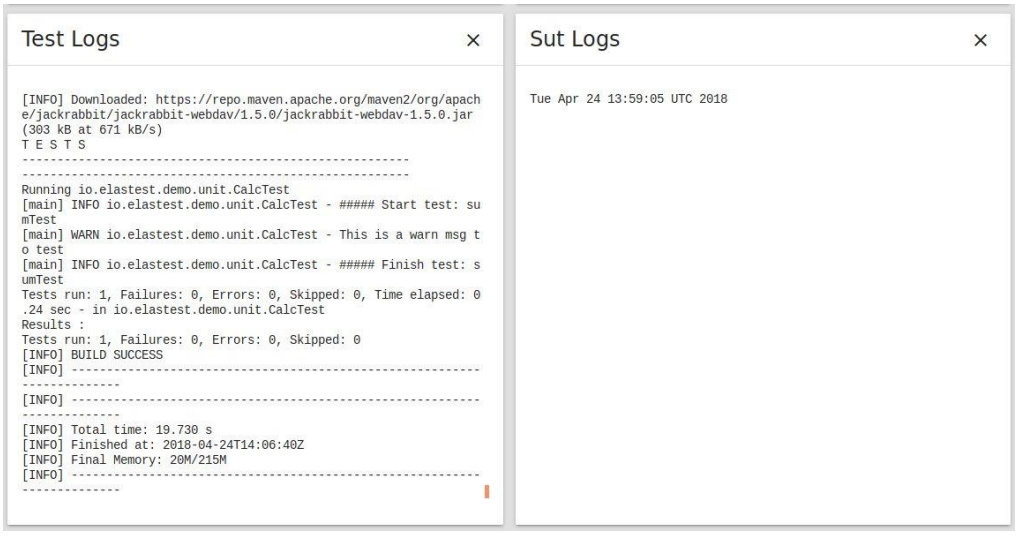

Figure 53. Logs visualisation within the ElasTest dashbaord

For further details check ElasTest documentation located in our public repository.

6.1.4.2 Road Map

Release 1: Initial Proof of Concepts includes;

- Set of monitoring agents less intrusive as possible in order to produce low overhead during the instrumentation.
- Lightweight agents as they are going to be deployed within the SuT.
- Evaluate different configuration management systems.

Release2: Initial version of the component working isolated.

- Initial version of EIM web service available
- Docker files allowing to easily deploy external software systems needed by our software module.

Release3: Adding new features

- Agent registration procedure implemented
- Support for relational and non-relational databases implemented.
- Deploy automatically instrumentation agents in the target cloud environments
- Adding new actions within the catalogue allowing to monitor
    - Performance and availability metrics
    - Logs and files
    - Network indicators

Release 4: First integrated version

- Fixings bugs reported in previous version
- Northbound interfaces integrated with TORM
- Integration with the EDM component as a persistent storage of data.
- Adapting component interfaces to be able to configure the output against TORM or against the EMS support service if it exists.
- Lifecycle of agents fully covered from its deployment/configuration to its termination.

Release 5: Adding new observability features + adding controllability feature.

- Use set of capabilities offered by the EPM module
- Extend controllability capabilities
- Test robustness and scalability across the different type of cloud infrastructure required by the project.
- Instrument other types of systems (i.e: AWS, Kubernetes)
- Extend the configurability of the component.

6.1.4.3 Code Reports

In ElasTest, EIM has been integrated with the CI system that uses Jenkins for automated tests and builds after every commit. For calculating the code coverage the EIM is integrated with Codecov.io.

Further details can be seen as part of the WP6 reports.

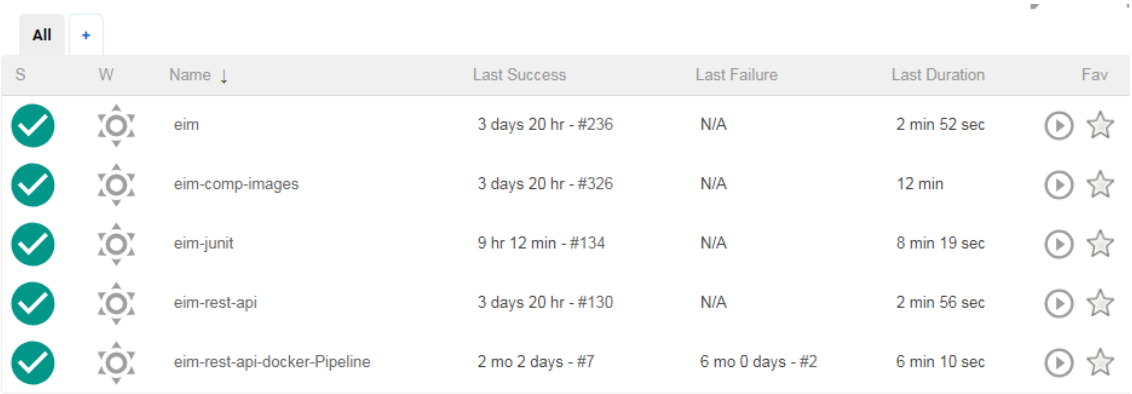

Figure 54. EIM Jenkins buil report

6.1.4.4 Code Repository

- The EIM code repository can be found on GitHub[79] and is licensed using Apache 2.0 [3].
- Within that repository, there is documentation[80] detailing how to run, use and extend the EIM.
- The API of the EIM can be viewed online here[81].

6.1.4.5 API

Figure 55. EIM: Publickey API

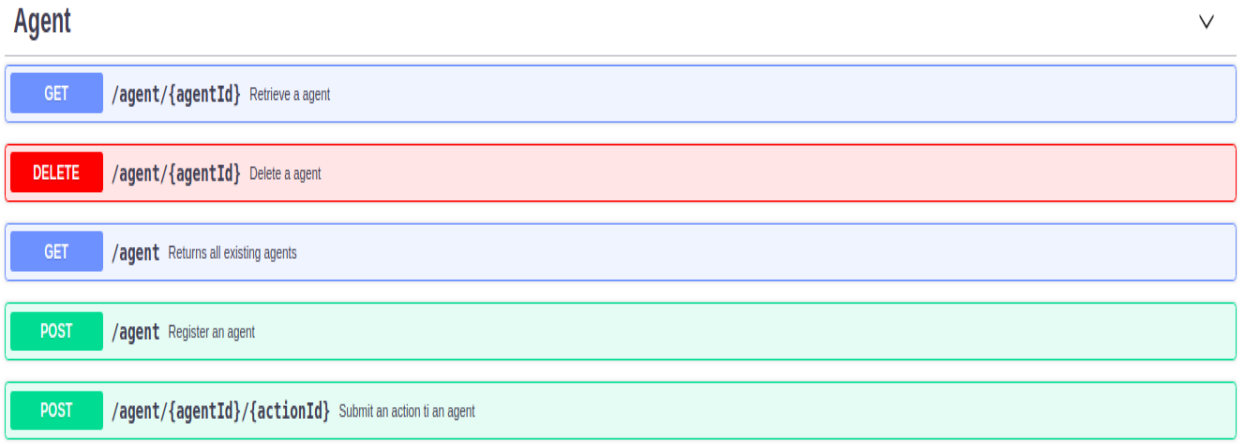

Figure 56. EIM: Agent API

[79] ElasTest Intrumentation Manager, https://github.com/elastest/elastest-instrumentation-manager
[80] EIM Documentation, https://github.com/elastest/elastest-instrumentation-manager/tree/master/docs
[81] EIM API, https://elastest.io/docs/api/eim/

Figure 57. EIM: AgentConfiguration API

6.1.5 Research Results and Future Plans

In order to archive the projects goals, ElasTest was designed to work always-on Instrumentation, Measurement, and Control (IMC) basis in a fully distributed way; we need to be able to dynamically provision resources in a unified manner within a short time period. To this aim, gathering metrics become a key aspect to consider during the execution of the tests against the software under evaluation.

Nowadays, there are many types of providers that can accommodate the provisioning of the computational resources and its configuration. From private providers to public ones, each of them exposing monitoring information through their own libraries or APIs, in that sense it becomes very complex to find a single way to gather information from an ecosystem of different Cloud Service Providers available out there. ElasTest project proposes to capture runtime information through monitoring agents instantiated one top of the computational nodes where the application is deployed.

The first consideration to take into account is that the aforementioned agents should be designed as less intrusive as possible considering that we do not want to affect the behaviour of the applications. To this end, the agents need to be as lightweight as possible, as they need to be deployed within the SuT, in addition the overhead produced by the agents needs to be minimal. We are avoiding vendor Cloud provider lock-in situation not relaying on a certain provider or API but at the same time we need to ensure that the agents are designed to consume well established operating system interfaces to guarantee interoperability across different OS distributions. In the scope of the project at least Linux systems are targeted.

The current version of the component supports the configuration and deployment of observability agents that are able to capture availability, performance and networks metrics from the application, as well as gather logs from different filesystem paths.

The future plans for the component considers extending its capabilities not only to allow the collection of metrics, but giving as well the possibility to inject custom behaviours to the application at runtime. The controllability capabilities expected in the new releases of the component will be able to emulate real world conditions while the application is running. The set of actions that are considered covers behaviours such as: overload the resource (CPU, mem, etc.), network packet loses, connectivity issues, fault tolerance after the failure of certain resource, among other actions that can be considered based on the vertical demonstrators needs.

7 Data Persistence Management

7.1 ElasTest Data Manager (EDM)

7.1.1 Introduction

The ElasTest Data Manager was built to separate the persistence layer of ElasTest from the rest of the platform. It is provided as a docker-compose deployment that is able to be scaled by changing compose number of images, so that scalability could be provided and tested in order to facilitate the future Kubernetes migration more easily.

The concept was to provide all mutable entities of ElasTest in a single component, in order to ease management, testing and expansion of them. This is generally a good practice in containerized environments, as the practice to handle separately (or even totally avoid to containerize) the data stores is quite common in large production environments.

7.1.2 Baseline Concepts and Technologies

Currently EDM provides several different data services as a set of Docker containers, all orchestrated by docker-compose and monitored by an API written in Java. Each service is provided as one or more containers, forming a minimal required cluster size. In addition to the persistence services, EDM also contains a set of supporting services, in order to provide data caching and data visualization.

EDM provides to ElasTest components the following persistence services:

- **MySQL[82]:** One of the most used open source RDBMS service. It is used by ETM to store structured information like TJobs, Suts, executions, etc. It is also used by other services like EIM, EPM and ESM.
- **Elasticsearch[83]:** Provides a distributed data-store for semi-structured data. It is the solution of choice used by ETM to store logs and metrics generated during the execution of TJobs.
- **Hadoop[84]:** Provides a data-lake store, in order to provide the most possibly flexible solution in data management. It provides the capability to store raw data in its original format and process it as-is, as well as extend with other technologies (e.g. Spark, Hive) and create a data platform fit for every purpose. It is used in ElasTest by ElasTest Recommendation Engine (ERE) and for ElasTest BigData Service (EBS).
- **Alluxio[85]:** Is a project that provides data abstraction and caching from multiple file and blob storage backends. This eases interactions with actual data, as the developer doesn't need to change anything to switch between data sources, and also provides a caching mechanism for faster read access.

[82] https://www.mysql.com/
[83] https://www.elastic.co/products/elasticsearch
[84] http://hadoop.apache.org/
[85] http://www.alluxio.org/

In addition, EDM also provides some services tailored to the management and administration of these persistence services:

- **Cerebro**[86]**:** Provides a management UI for Elasticsearch.
- **Kibana**[87]**:** Provides a data-visualization layer for Elasticsearch.
- **CloudCommander**[88] Provides a web-based file browser, that is able to surf HDFS files. Its purpose is to provide a user-friendly way of interacting with ElasTest data stores, e.g. upload a large data file or download a test output.

7.1.3 Component Design and Architecture

As described before, EDM is basically a set of persistence and related management services. It also provides a custom developed service, called Management API, to perform some administrative tasks over the persistence services. Figure 58 shows how EDM sub-components are associated and how are used by other ElasTest components.

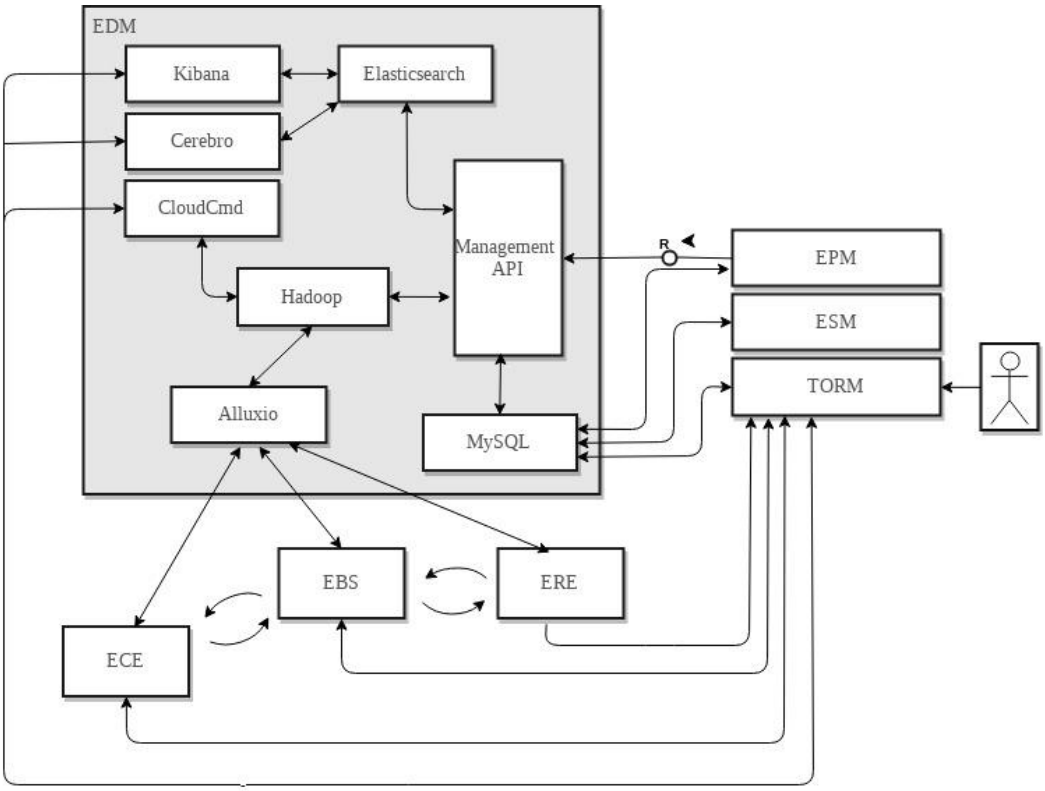

Figure 58. EDM FMC Diagram

[86] Cerebro, https://github.com/lmenezes/cerebro
[87] Kibana, https://www.elastic.co/products/kibana
[88] CloudCommander, http://cloudcmd.io/

7.1.4 Roadmap and Features

Currently, EDM provides an extensive set of features and a single point of management for the whole set of services. The idea behind this is that the central ElasTest management system will have fine-grained control to its data, by only using an abstract API. Generally, all services provided by EDM fall under these three categories:

- Data persistence: These services actually manage data storage.
- Data visualization: These services provide an insight on the actual data stored, and allow for basic querying and visualization.
- Data management: These services allow for data and cluster management.

Extending the service groups with new services or extending service capabilities is an ongoing process that evolves in parallel with ElasTest. Although the project's roadmap is generally clear, pivoting also affects EDM decisions.

7.1.4.1 API

EDM Management API have been developed by Relational project member. It provides a RESTful API to perform data management operations on all persistence services. The following operations are supported:

- Backup all data from EDM persistence services
- Restore all data for the platform based on a specific backup ID
- Retrieve a list with all the SuT backups on EDM
- Backup all data for a specific SuT
- Restore all data for a specific SuT
- Delete all data for a specific SuT

Figure 59 shows the Swagger documentation endpoint for all those operations.

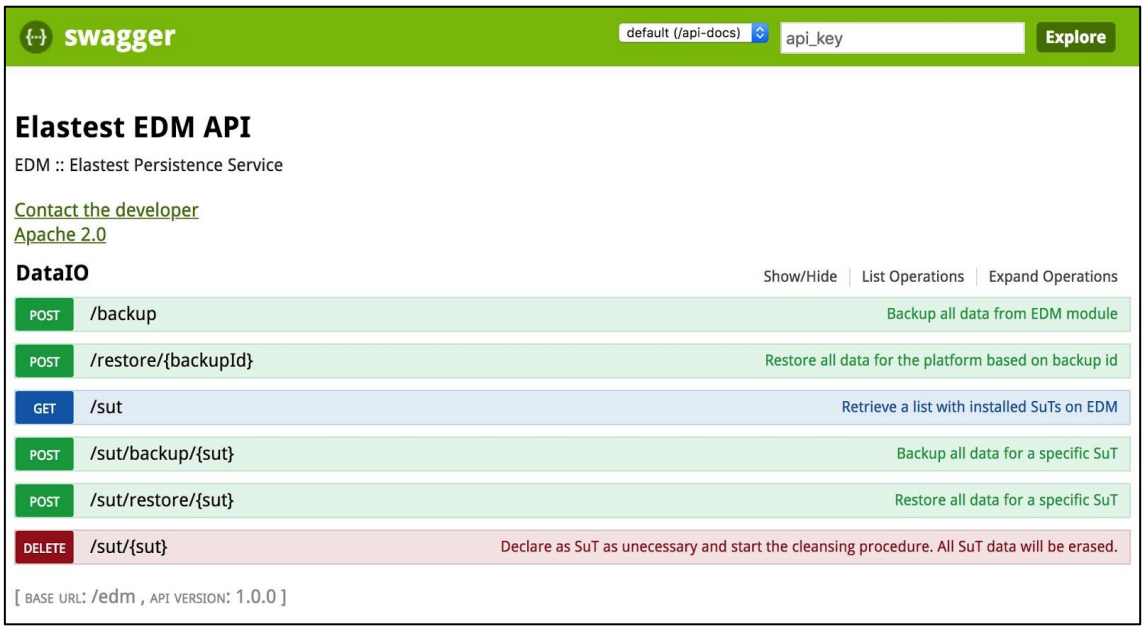

Figure 59. EDM API Documentation

Limitations of this API include:

- Multiple historical backups.
- Backup list, with timestamps and comments
- Service scaling management (currently available only via docker-compose).

These limitations are subject for resolution in future EDM releases in order to provide a more polished, end-user friendly product. Service scaling management specifically, is one very interesting feature that is still missing specifications on the underlying infrastructure.

7.1.4.2 Code Reports

In the scope of ElasTest, EDM is integrated in Jenkins automated build system. The pipeline contains a number of stages that build the several different services, as well as unit tests for the Management API service (Figure 60).

Figure 60. EDM Jenkins pipeline

EDM Management API service is tested via a set of 44 unit tests. Test reports are created with Cobertura (since the service is written in Java), and they are uploaded to codecov.io for visualization. Since EDM development is frozen for the last six months, the longest histogram provided by codecov.io is now a flat line. Coverage is currently at 90% as it can be seen in Figure 61.

Figure 61. EDM Coverage Report

7.1.4.3 Code Repository

The EDM code can be found on GitHub[89] and is licensed under Apache Licence 2.0. Within the repository there is documentation[90] on how to use EDM. The API of EDM can be viewed online here[91].

8 Conclusions

The intermediate version of the ElasTest Cloud Components extends the benefits offered by the cloud to testers that have to face the challenges of validate large scale software systems. The set of components presented individually within this report has been integrated with the ElasTest Platform; each of them reaches its fourth release (R4) where the first integrated version of the all platform is delivered

The document provides an overview of the components developed in this period; the intermediate version of the Cloud modules covers the specification, design and their implementation until M18. The resultant modules described here are aligned with the initial requirements and specifications gathered in "D.2.3 Requirements, use-cases and architecture v1" [5].

The next steps towards the final release will be focused on the completion of the features scheduled on each of the components RoadMap, the priority of each of the features is subject to the ElasTest release planning meetings, and specially within the next period those priorities are going to be highly influenced by two factors: a) the research objectives that the components need to achieve and b) the feedback/support requested by the Vertical demonstrators that are going to validate the platform.

[89] EDM GitHub, https://github.com/elastest/elastest-data-manager
[90] EDM Documentation, https://github.com/elastest/elastest-data-manager/blob/master/docs/
[91] EDM API, https://elastest.io/docs/api/edm/

9 References

[1] ElasTest project Description of Action (DoA) – part B. Amendment 1. Reference Ares (2017)343382. 23 January 2017.

[2] Bertolino, A., 2007, May. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering* (pp. 85-103). IEEE Computer Society.

[3] Apache 2.0 license terms. https://www.apache.org/licenses/LICENSE-2.0. Accessed on 07 March 2017.

[4] Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION. Communications Networks, Content and Technology. 11 November 2016.

[5] D.2.3. ElasTest requirements, use-cases and architecture v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.

[6] D.3.1. ElasTest Platform cloud modules v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.

[7] D.4.1. Test Orchestration basic toolbox v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.

[8] D.4.2. Test recommendation engines v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.

[9] D.5.1. ElasTest Test Support Services v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.

[10] D.6.1. ElasTest Continuous Integration and Validation System v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.

[11] D.6.2. ElasTest platform toolbox and integrations v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.

[12] D.3.2. ElasTest Platform cloud modules v2. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.

10 Appendix

10.1 ElasTest Service Manager Sequence Diagrams

10.1.1 ServiceConsumer Sequence Diagrams

10.1.1.1 #1 List available service types

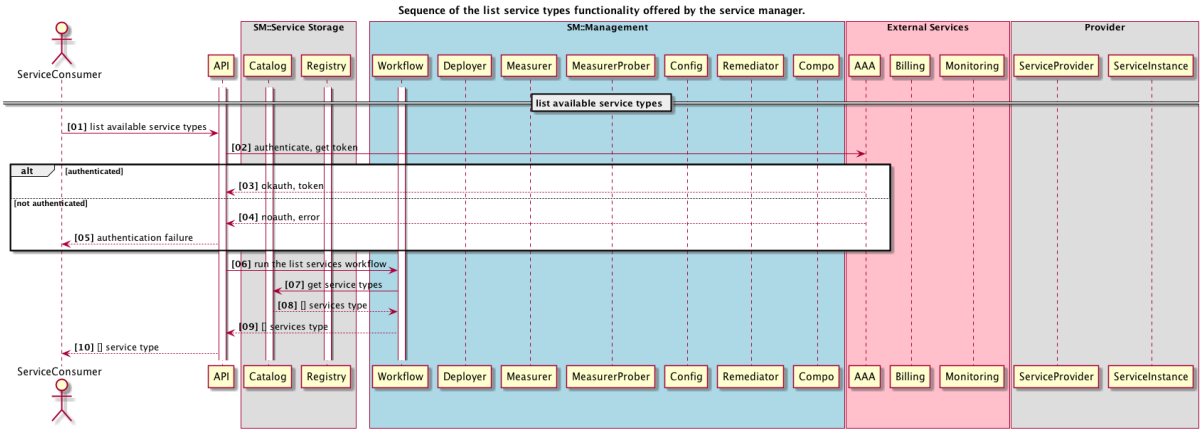

Figure 62. Consumer: List of available service types

10.1.1.2 #2 Create a service instance of a specific service type

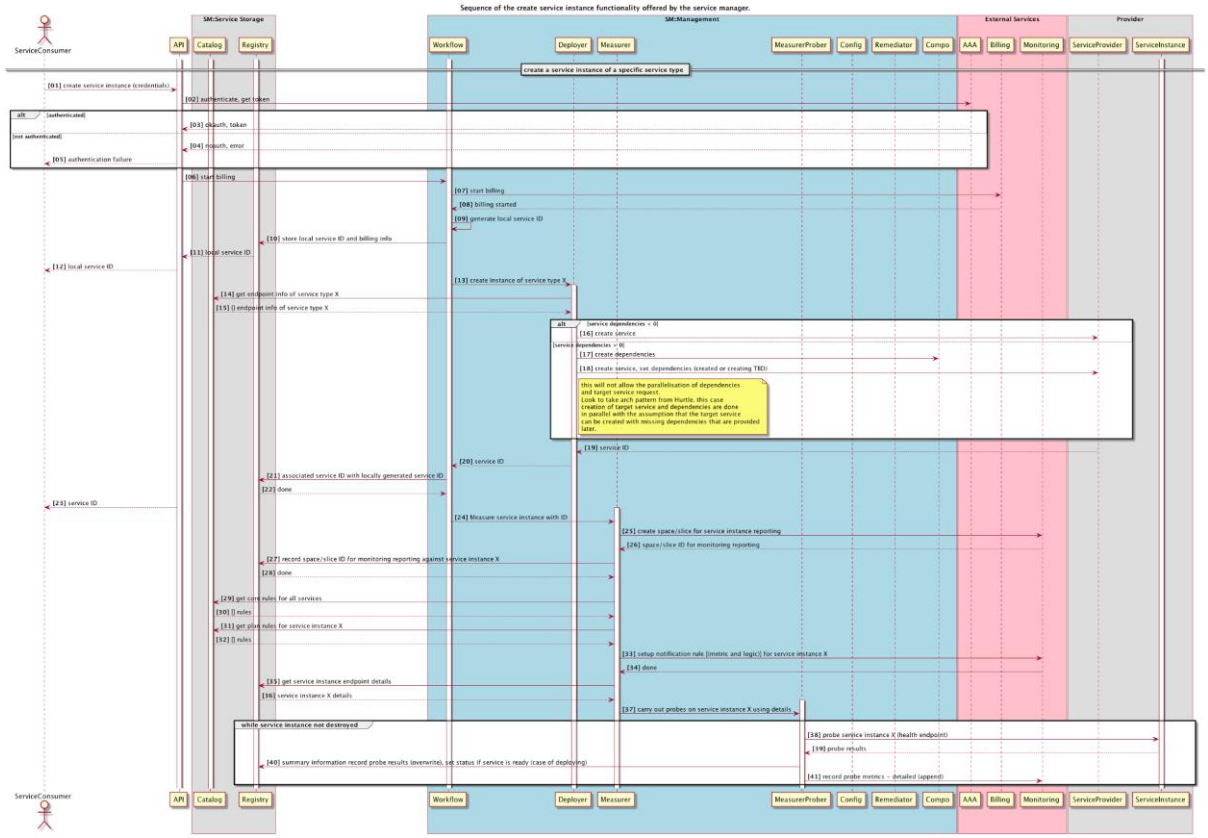

Figure 63. Consumer: Create a service instance

10.1.1.3 #3 Get/poll service instance status

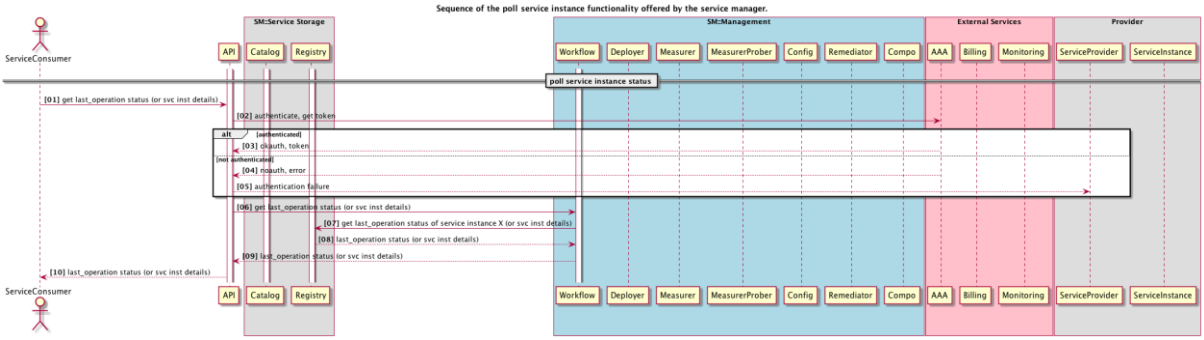

Figure 64. Consumer: Get/poll serice status

10.1.1.4 #4 Bind service instance

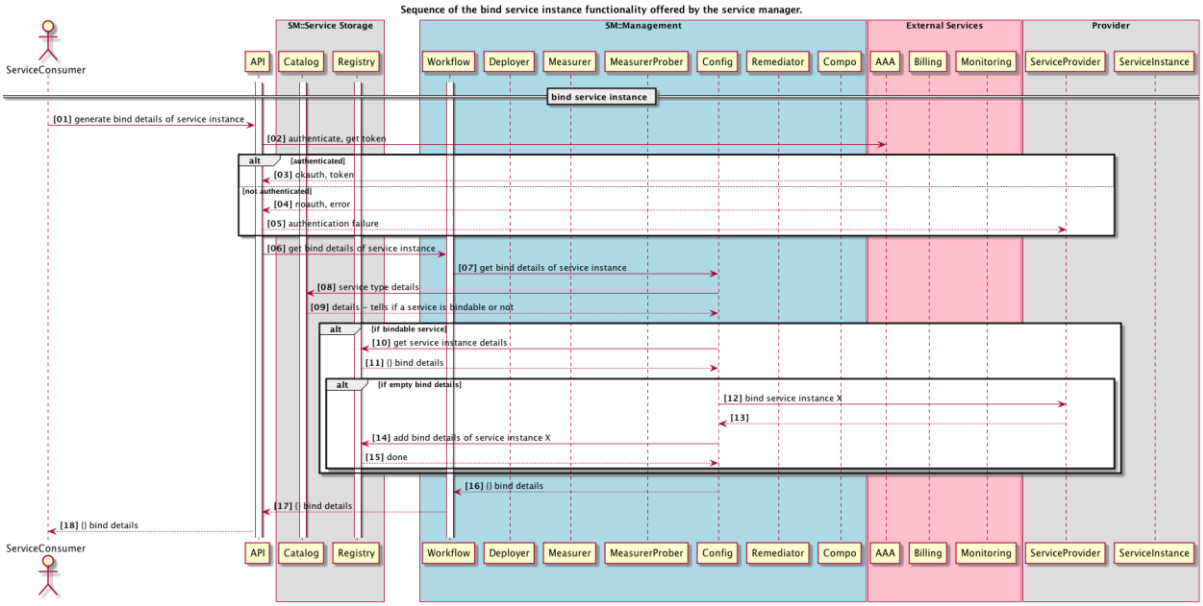

Figure 65. Consumer: Bind service

10.1.1.5 #5 Configure service instance

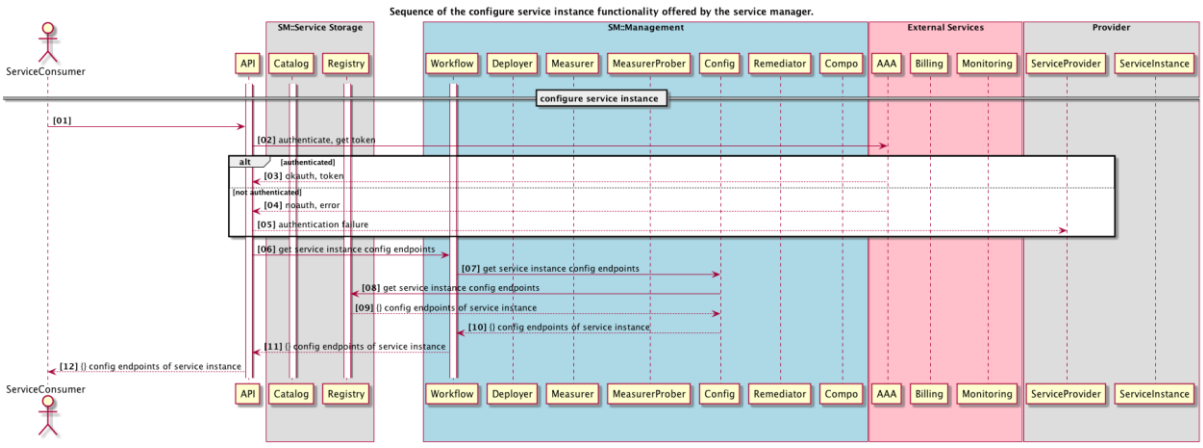

Figure 66. Consumer: Configure service instance

10.1.1.6 #6 Get service instance details

This is implemented via use case and shown in sequence diagram ID #3

10.1.1.7 # 7 Get service instance metrics

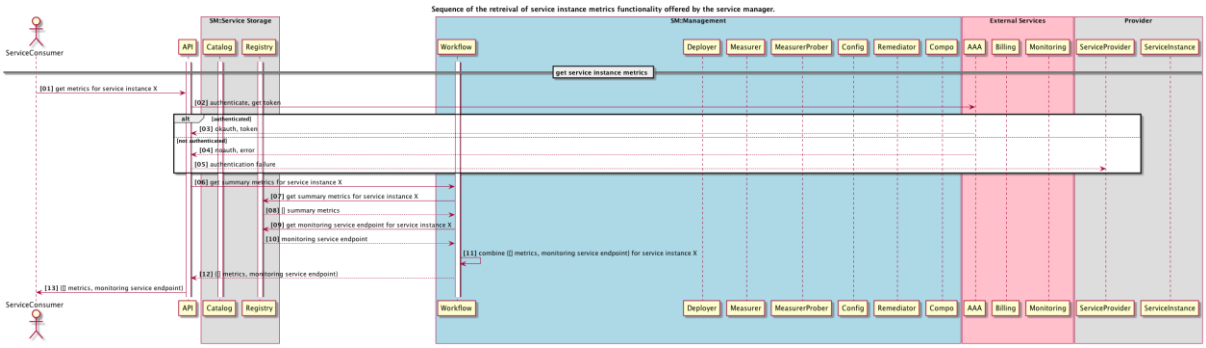

Figure 67. Consumer: Get service metrics

10.1.1.8 # 8 Update service instance

This is currently not implemented; therefore the sequence diagram is omitted.

10.1.1.9 # 9 Unbind service instance

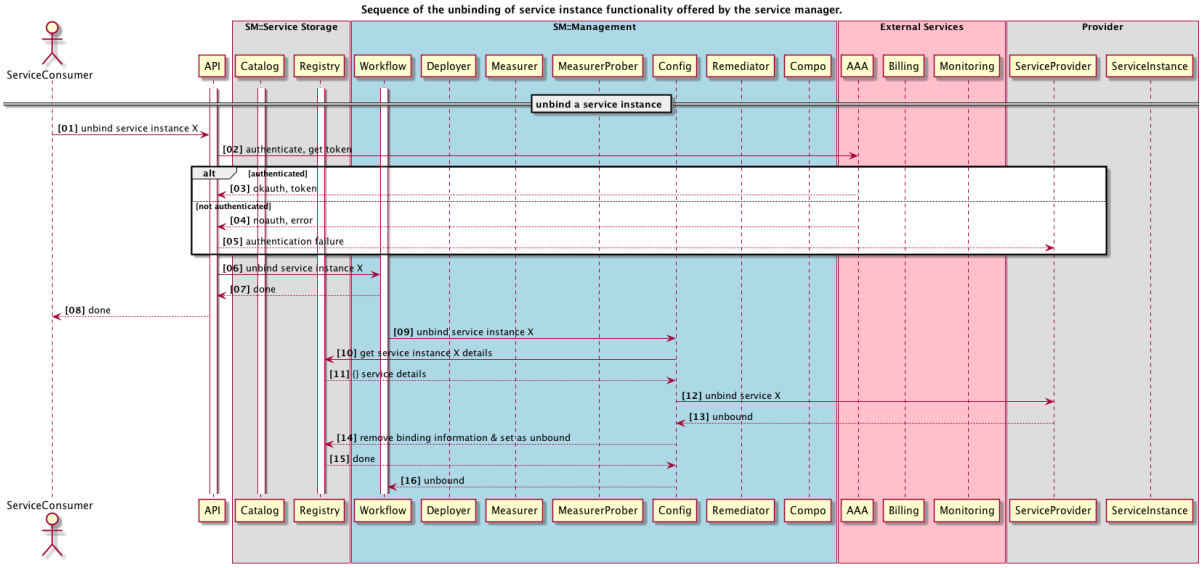

Figure 68. Consumer: Unbind service

10.1.1.10 # 10 Delete service instance

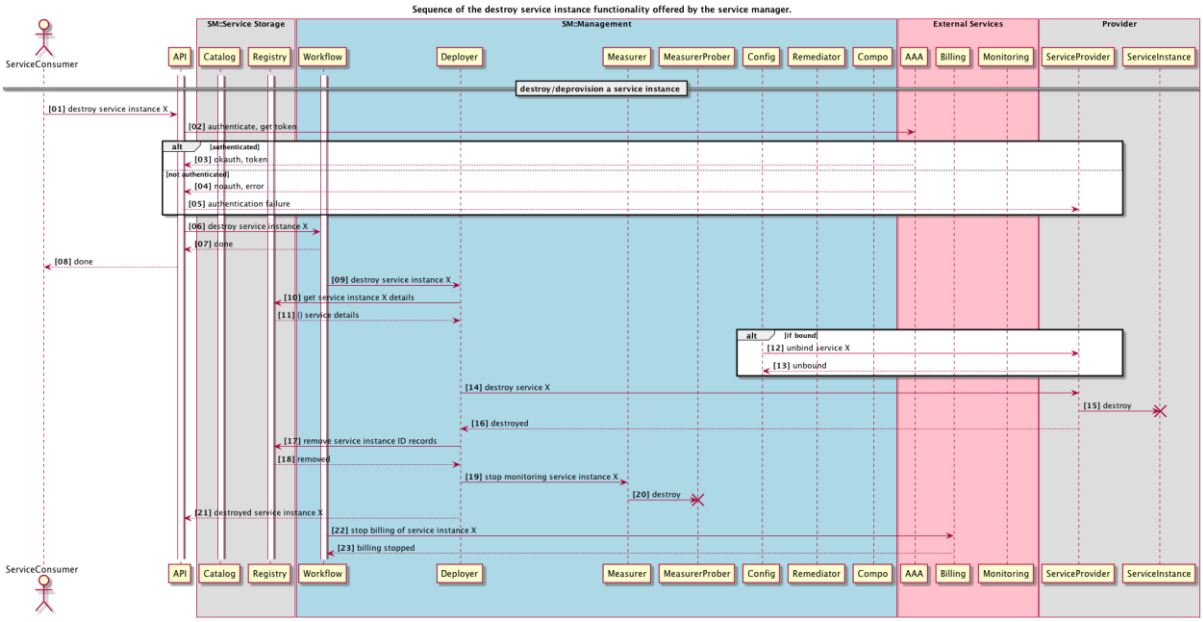

Figure 69. Consumer: Delete service

10.1.2 ServiceProvider Sequence Diagrams

10.1.2.1 #1 List available service types

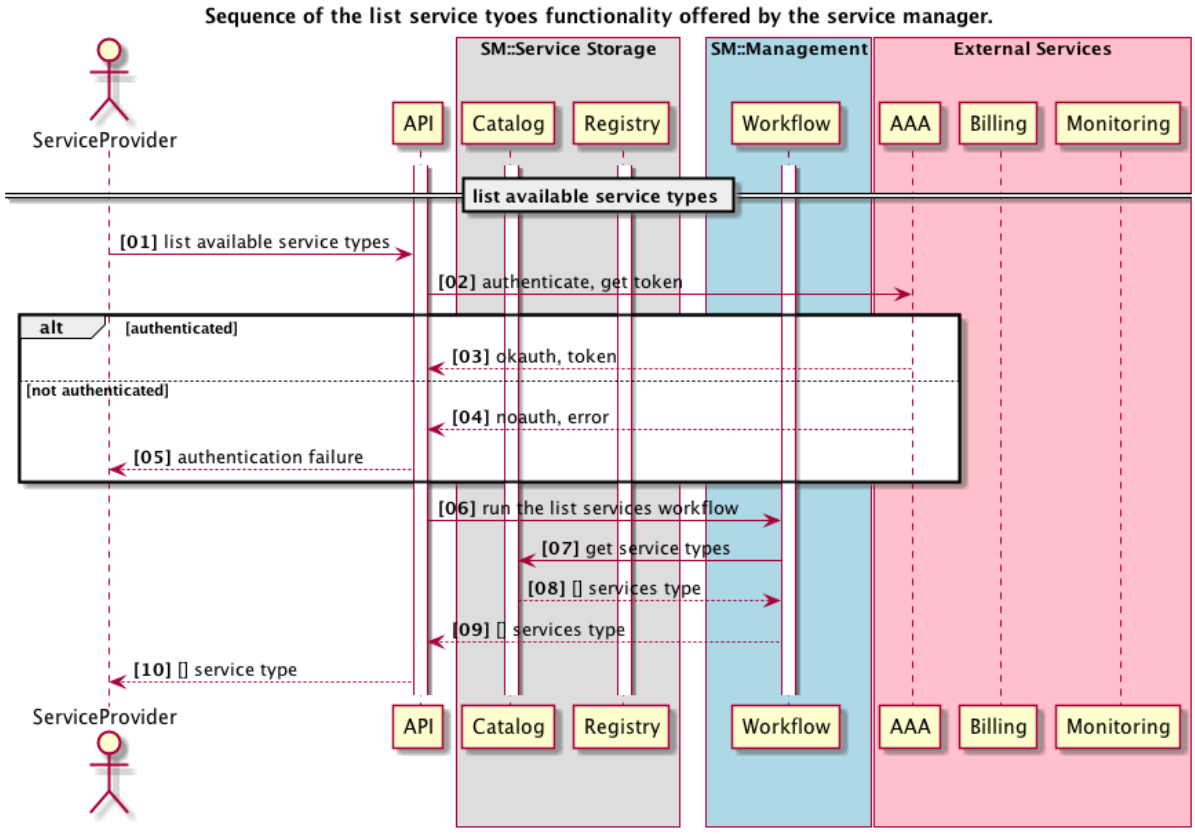

Figure 70. Provider: List service

10.1.2.2 #2 Register service type and endpoint information

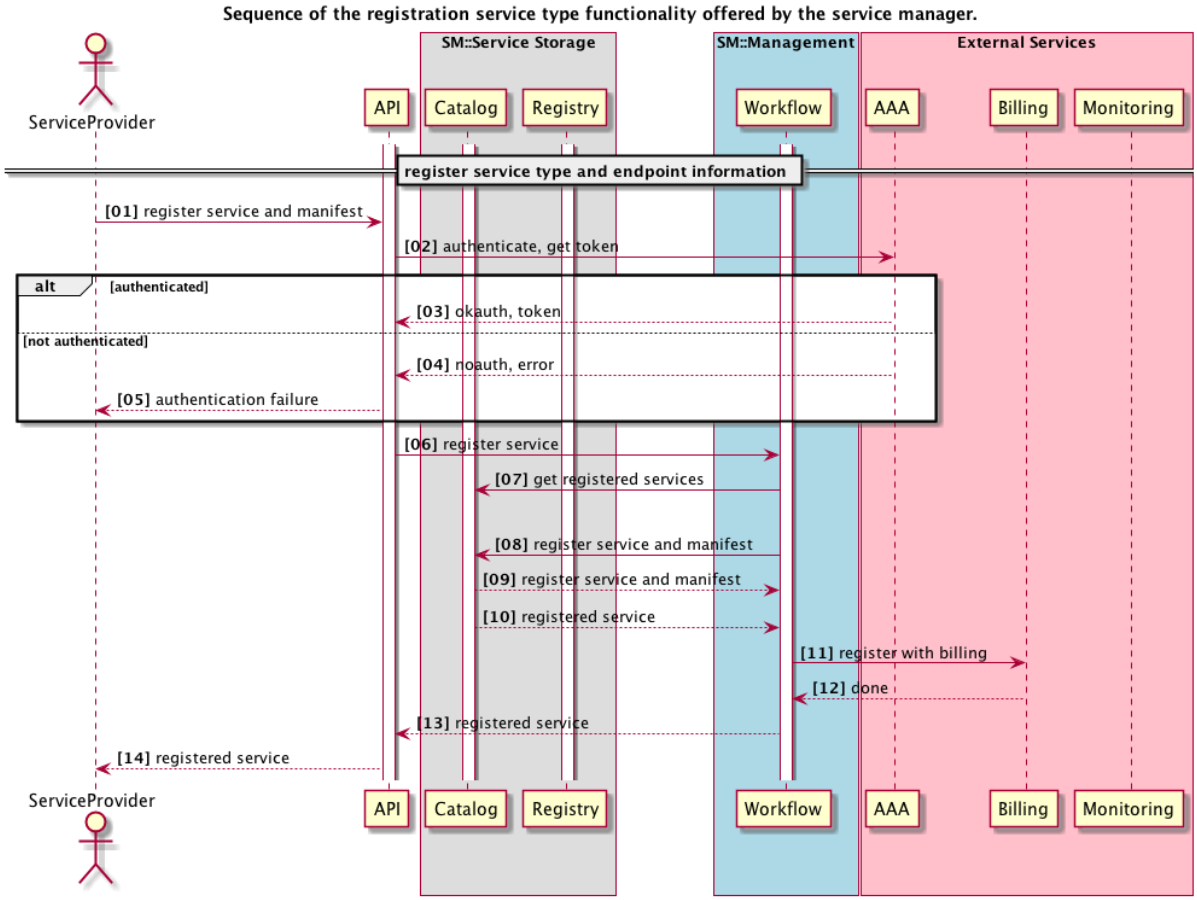

Figure 71. Provider: Register service

10.1.2.3 #3 Register service type manifest

This is implemented via sequence diagram ID #2.

10.1.2.4 #4 Update service type business information: plan and description

Note that this is the same technical implementation as ID #5

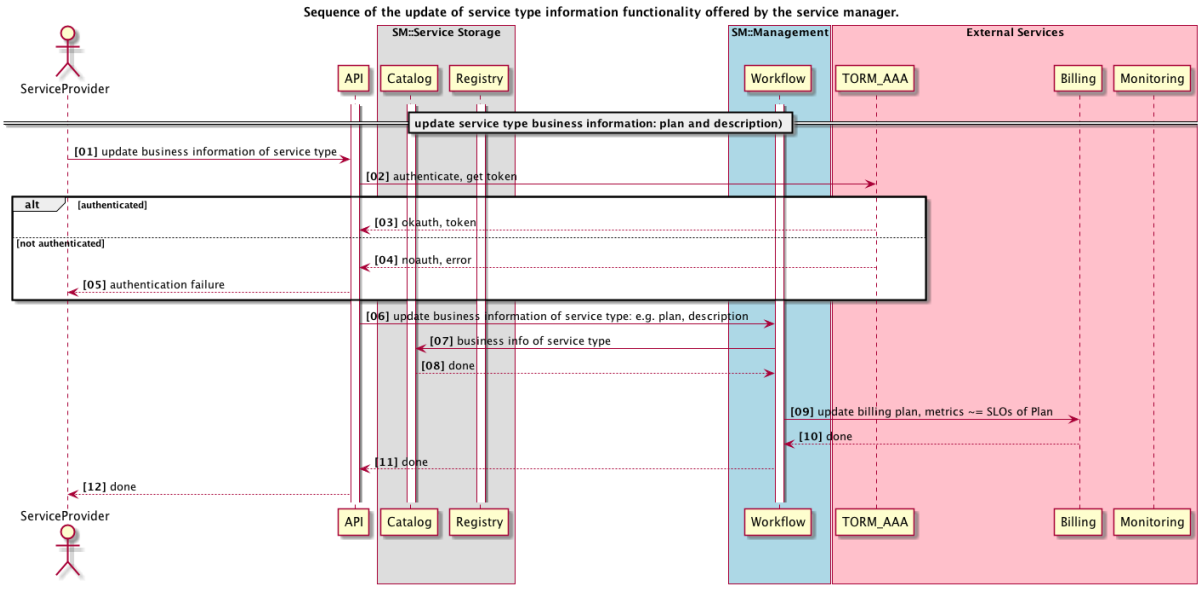

Figure 72. Provider: Update plan and description

10.1.2.5 #5 Update service type technical information: endpoint/API

Note this is the same technical implementation as ID #4

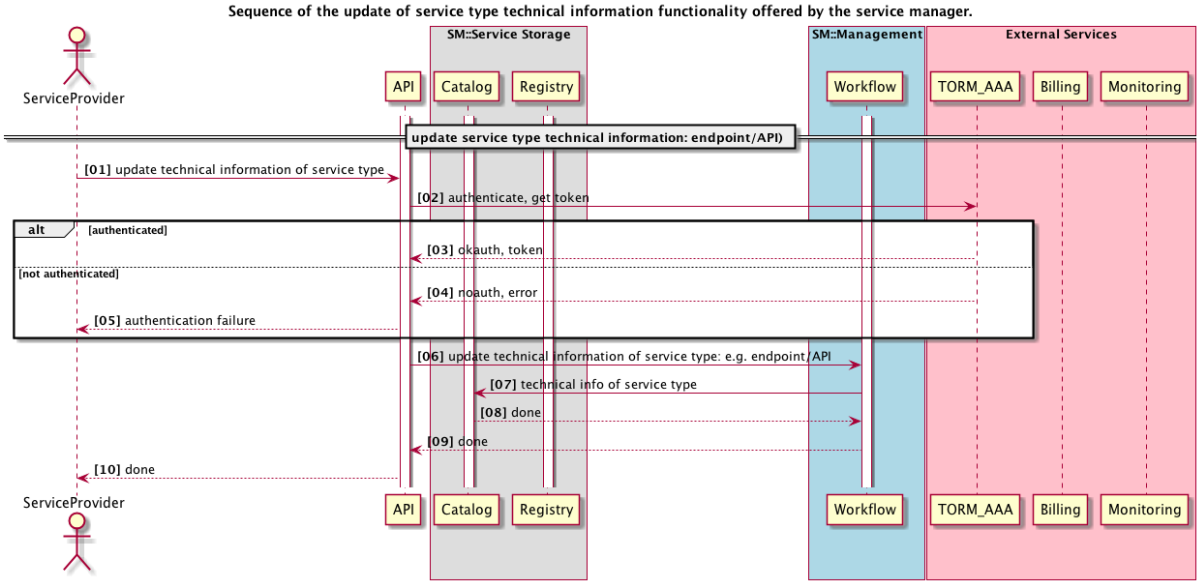

Figure 73. Provider: Update endpoint

10.1.2.6 #6 Report service instance metrics

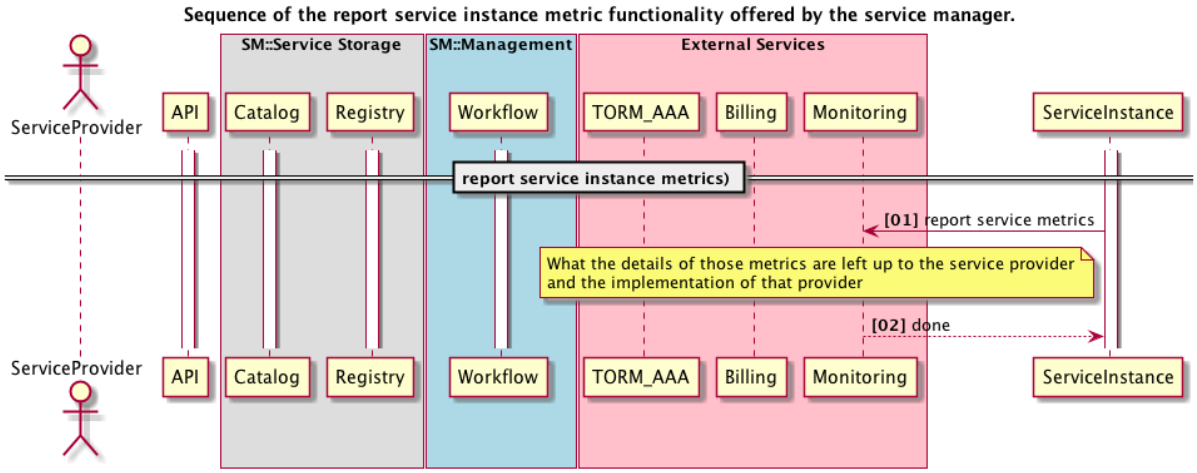

Figure 74. Provider: Report service metrics