PIACERE

Deliverable D6.1

PIACERE run-time monitoring and self-learning, self-healing platform - v1

Editor(s):	Juncal Alonso, Gorka Benguria
Responsible Partner:	TECNALIA
Status-Version:	Final - v1.0
Date:	30.11.2021
Distribution level (CO, PU):	PU

Project Number:	101000162
Project Title:	PIACERE

Title of Deliverable:	PIACERE run-time monitoring and self-learning, self-healing platform - v1
Due Date of Delivery to the EC	30.11.2021

Workpackage responsible for the Deliverable:	WP6 - Monitor plan and self-heal runtime of Infrastructure as Code
Editor(s):	Fundación Tecnalia Research & Innovation
Contributor(s):	Tecnalia, Ericsson, Polimi, Xlab, 7bulls
Reviewer(s):	HPE
Approved by:	All Partners
Recommended/mandatory readers:	Recommended WP2, WP5, WP7

Abstract:	These deliverables will contain the main outcomes from M1 to M30 of T6.1-T6.4 due to the high dependency of all the different tasks. It will include the monitoring stack coming from task 6.1 with all the time series data collected as well as the monitoring from the security policies from task 6.4, the set of machine learning algorithms (task 6.2) that comprise the self-learning mechanisms and the self-healing strategies (task 6.3) that trigger an optimized redeployment (see WP5). It will be an iterative process. Each deliverable will comprise a Technical Specification Report.
Keyword List:	Monitoring, Forecast, Healing, Security, Availability, Performance
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the authors' views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	xx.xx.202x	First TOC and sections assignment	TECNALIA
v0.2	23.03.2021	Comments and suggestions received by consortium partners	TECNALIA
V0.3	28.05.2021	Contributions to the SOTA	7BULLS, ERICSSON, POLIMI, XLAB, TECNALIA
V0.4	29.07.2021	Updates to the SOTA	7BULLS, ERICSSON, POLIMI, XLAB, TECNALIA
V0.5	17.09.2021	Implementation and delivery details on Performance monitoring, Self-learning	7BULLS, ERICSSON, POLIMI, XLAB, TECNALIA
V0.6	22.10.2021	Implementation and delivery details on Security monitoring, Self-learning	XLAB, ERICSSON
v.0.7	4.11.2021	Implementation and delivery details on Self-healing	TECNALIA, ERICSSON, POLIMI
v.0.8	15.11.2021	Updates to the SOTA	TECNALIA, XLAB, ERICSSON, POLIMI
V0.9	16.11.2021	Final Editing before internal review	TECNALIA
V0.95	26.11.2021	Addressed comments from internal reviewer	TECNALIA
V1.0	30.11.2021	Ready for submission	TECNALIA

Table of contents

List of tables

List of figures

Terms and abbreviations

AD	Anomaly Detection
AMEL	Application Modelling and Execution Language
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CAMEL	Cloud Application Modelling and Execution Language
CSLA	Cloud Service Level Agreements
CSP	Cloud Service Provider
DevOps	Development and Operation
DNN	Deep Neural Networks
DoA	Description of Action
DOML	DevOps Modelling Language
EC	European Commission
ELK	Elastic Logstasth Kibana
EMS	Event Management System
EPA	Event Processing Agents
EPM	Event Processing Manager
EPN	Event Processing Network
FP	False Positive
FT-Tree	Frequent template tree
GA	Grant Agreement to the project
HIDS	Host-based intrusion detection system
HTTPS	Secure HTTP Hyper Text Transport Protocol
HVM	Hypersphere Volume Minimization
IaC	Infrastructure as Code
IDF	Inverse Document Frequency
IDS	intrusion detection systems
IEC	Infrastructure Elements Catalogue
IEP	IaC execution platform
IOP	Infrastructure Optimizer platform
IPS	Intrusion Prevention Systems
KPI	Key Performance Indicator
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
MAPE-K	Monitor-Analyze-Plan-Execute over a shared Knowledge
MCSLAs	Multi-Cloud Service Level Agreements
MLM	Masked Language Modelling
MSE	Mean Square Error
MTBF	Mean Time Between Failures
MTTR	Mean Time To Recover
NFR	Non-functional Requirements
NSM	Network Security Monitoring
PCA	Principal Component Analysis
PRC	PIACERE Runtime Controller
QoS	Quality of Service

RCA	Root Cause Analysis
REST	REpresentational “State” Transfer
SEM	Security Event Management
SIEM	Security Information and Event Managements
SLA	Service Level Agreement
SLO	Service Level Objective
SNMP	Simple Network Management Protocol
SOTA	State of the art
SW	Software
TF	Term Frequency
URL	Uniform Resource Locator
UTM	Universal Threat Management
VAST	Visual Analytics Science and Technology
VAT	Vulnerability Assessment Tool

Executive Summary

This document is a supporting document of the PIACERE run-time monitoring and self-learning, self-healing platform. Therefore, it is one part of the D6.1. The whole D6.1 is composed by:

- The source code of the components that implement the required functionality
- The infrastructure as code specification that defines the environments in which these components are developed, tested, and integrated.
- The specification of the way in which these components should be run together
- The specification of tests over these components both individually and as an integrated set.
- The specification of the interfaces both programmatical and human oriented
- This document

The objective of this document is, on the one hand, to contain the rationale of the architecture and approaches taken in the development of the different components, and on the other hand, to provide details in how the different components have been developed and can be deployed.

To address the first objective, we have included the state-of-the-art analysis on the different aspects of this first iteration of the platform that supports the selected development approaches. We include information on the following aspects: Infrastructural elements monitoring, self-learning and self-healing. Besides, for the monitoring and self-learning we focus in performance and security dimensions.

To address the second objective, for each major component of the PIACERE run-time monitoring and self-learning, self-healing platform we include information about its implementation, deployment and usage. The implementation sections contain key information to understand the features implemented and how the component relates to other components in the architecture. The delivery and usage sections contain information that will be used during the deployment integration of the WP6 components together with other components from other work packages in the common PIACERE framework.

The current version of the PIACERE run-time monitoring and self-learning, self-healing platform, was developed with three main targets in mind. The first one is to ensure the availability of key resources among the components; the second one was to start working in the more challenging internal aspects of the components; finally, the third one was to establish the foundations for the integration with other work packages.

Next versions of this document will include updates on the approaches, the implementation and delivery and usage based on the advances and the changes introduced to move forward the integration of these components with the rest of PIACERE components.

1 Introduction

1.1 About this deliverable

This document is a supporting document of the first version (M12) of the PIACERE run-time monitoring and self-learning, self-healing platform. It is a complementary document that explains the approach, the implementation, and the way to deliver and use each one of the current components that take part in the implementation of the functionalities expected from the WP6.

This document has been developed merging contributions from all the partners of all the tasks of the WP6:

- Task 6.1 Runtime monitoring and self-healing preparation
- Task 6.2 Self-learning algorithms for failure prediction
- Task 6.3 Strategies and plans for runtime self-healing
- Task 6.4 Runtime security monitoring

The purpose of this document is twofold:

- To serve as a reference of the background of the technical decisions taken regarding the approaches followed during the development of the components
- To contain information to support future development. This includes information to understand how the components have been developed, which are their features and how can be tested.

1.2 Document structure

The document is structured into four parts. The section 2 explains the components covered by the document and their relationship. The next part addresses the state of the art that supports the technical decisions taken to develop the different components covered in this deliverable. This part is covered from section 3 to section 5. The third part addresses the implementation details of the different components.

For each component we include details about the functional description, the requirements covered, how it fits in the overall architecture and its technical description. This part is covered from the section 6 to the section 0. The requirements include project level requirements that come from the use cases, and the component internal requirements that are internally established with the upcoming integration in mind.

The final part addresses the delivery and usage of each component. This part is covered from section 12 to section 0. Section 18 reports the conclusions.

These two parts, the implementation and the delivery, have been designed with the isolated use in mind by the developers, without requiring reading the whole document. With that objective in mind some figures may be repeated to improve that isolated readability.

2 Run-time monitoring and self-learning, self-healing platform in PIACERE

The components covered by this deliverable are part of the PIACERE infrastructure advisor platform as show in Figure 1. The infrastructure advisor has the role of ensuring the optimal deployment of the application specified in the DOML (DevOps Modelling Language) along the time.

The role of the components in this architecture is on the one hand to monitor the NFR (non-functional requirements) stated in the DOML and in case there are some deviation, or a deviation is forecasted, take corrective actions. On the other hand, the components will feed data into the Infrastructure Elements Catalogue (IEC) so that the real measurements are taken into account in the following IOP (Infrastructure Optimizer platform) calculation of the optimal deployment for a given application deployment request.

These components are mainly controlled by the PIACERE Runtime Controller (PRC) that will inform the components about the new deployments that should be tracked, and the deployments that do not require to be tracked anymore.

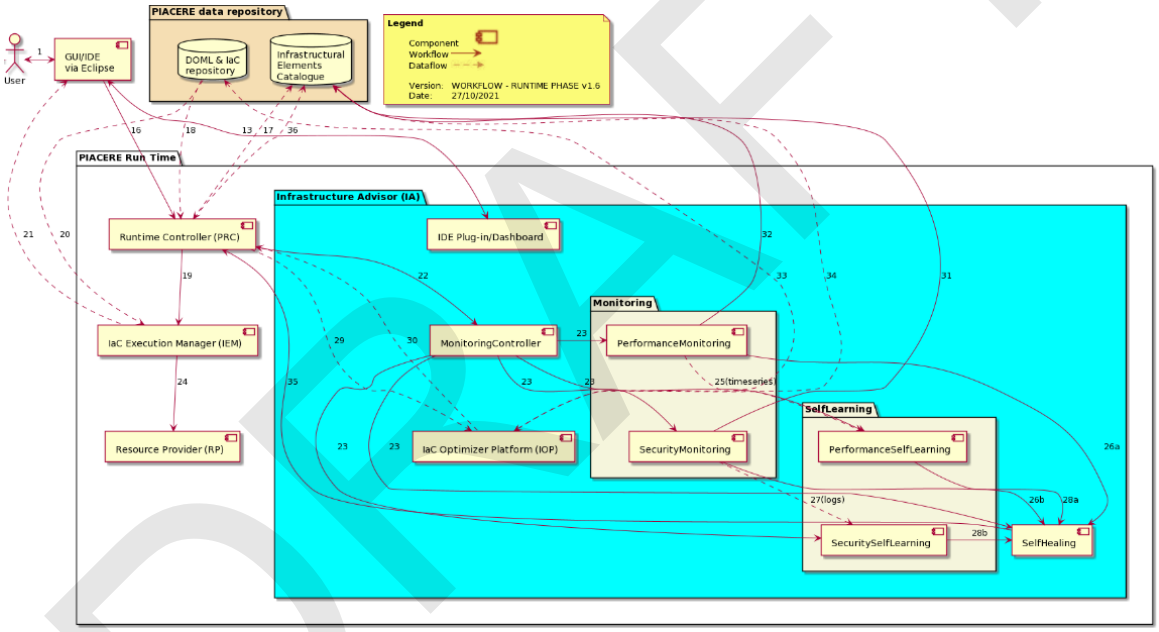

Figure 1. PIACERE Runtime Diagram on its 1.6 version

Besides the components indicated in the architecture of the PIACERE runtime platform, the monitoring components, both performance and monitoring, will deploy agents together with the deployed applications in order to gather the data required for the control of the fulfilment of the NFR (Non-functional Requirements) stated in the DOML.

WP6 will include the following components to achieve their role in the PIACERE platform:

- Monitoring Controller: in charge of centralizing the activities involved in the start and end of the monitoring of deployments, as they are created and destroyed by the PRC.
- Monitoring: This is a package that contains components that gather data from the deployments and controls the non-functional requirements continuous achievement. It includes components to control two aspects.
    - Performance (also including availability)
    - Security

- Self-learning: This is a package that contains components that perform forecasts about the future values of key measures on the infrastructure resources supporting the deployments. It includes components to provide forecasts on two aspects.
    - Performance (also including availability)
    - Security
- Self-healing: This component receives alerts from the previous components and based on the type of alert it requests the PRC to perform different actions such as redeploy, reboot, scale, etc.

NOTE: currently, the IDE Plug-in Dashboard is not included as it does not interact with the components addressed in this work package.

3 Infrastructural elements monitoring

3.1 Infrastructural elements monitoring approaches and challenges

Multi-cloud scenarios are more and more used to deploy microservices-based applications. The components of such an application are described in the literature as "*loosely coupled units of development that work on a single concern*" [1]. In the cloud, each microservice (or component) can be deployed in a different resource, even in a different cloud, attending its specific needs or non-functional requirements (NFR) such as location, cost, performance, etc., making the multi-cloud scenario especially adequate for the deployment of microservices-based applications. This new architype, where an application is deployed in distributed cloud resources, would not be possible without novel developments in governance, SLA (Service Level Agreement) management and monitoring.

In fact, one of the challenges in the area of cloud services federation, among others like the data portability or the lack of applicability of standards and legislation, is the monitoring and assessment of cloud services SLAs [2]. Precise monitoring of Quality of Service (QoS) and SLA verification of cloud services enables additional functionalities [3], as service selection or real time capacity estimation. Tools such as Nagios[1] or Ganglia[2] allow monitoring low-level metrics of computing resources in general, but automation on the configuration and calculation of complex metrics to assess CSLAs is still missing, especially when addressing multi-cloud environments. [2]

In the H2020 project DECIDE, a component that supports the brokerage of cloud services is presented, called ACSmI [4]. One of the functions of this broker is to control the fulfilment of the SLAs for each Cloud Service contracted. ACSmI monitors the SLAs (also called non-functional properties or NFRs) of the services offered by the Cloud Service Providers (CSPs) and assesses them to detect any violation. If a violation of some SLO (Service Level Objectives) is detected, an alert is raised. In ACSmI, the NFRs assessed are performance, availability, location and cost, while virtual machines are the only cloud resource used.

For each of the selected NFR, related metrics to be assessed have been defined. To be able to compare, combine and assess SLAs from different CSPs, the metrics are defined according to ISO/IEC 19086-1:2016 standard [5]. This standard seeks to establish a set of common cloud SLA building blocks (concepts, terms, definitions and contexts) that can be used to create Cloud Service Level Agreements (CSLAs).

In order to support the most standardized metrics, the guidelines defined in the mentioned ISO standard were adopted by ACSmI for the metrics selected:

- Availability. Availability is defined as *A = MTBF/(MTBF+MTTR),* $\sum_{i=1}^{n}(100\%\text{-term}_i)$ where MTBF (Mean Time Between Failures) and MTTR (Mean Time To Recover), are calculated based on other discrete metrics using different techniques.

- Performance. For the performance the usage of CPU, memory and disk is measured. Different thresholds can be configured ad-hoc through the ACSmI monitoring API.

- Location. It determines where a cloud resource is located, geo-locating its IP address from the Service registry.

[1] https://www.nagios.org/
[2] http://ganglia.sourceforge.net/

- Cost**.** Determines the current cost that a CSP is reporting on a certain resource. The actual incurred cost is calculated by monitoring the billing.

ACSmI combines push and pull monitoring (internal and external approach for monitoring VMs) for cloud resources. This implies that pre-configured agents are to be installed in the corresponding virtual machines, in an architecture described as Extended Internal Adaptive in [6]. It is composed by several components, among others:

- Metering or Data collection: Collects the data from the different cloud services where the application is deployed. Based in Telegraf[3] open source tool, Metering is automatically configured based on the information of the application to be deployed.

- SLA Assessment: in charge of the aggregation of the different raw metrics in order to assess the values of the NFRs with respect to the SLOs.

- Violations Handler: Once the assessment detects a violation of some SLA, this subcomponent registers it for future consults and informs of the violation to the CSP.

The definition of a composed SLA for a multi-cloud application is another issue that needs to be considered. This is critical for multi-cloud applications, for which the composed Multi-Cloud SLA (MCSLA) is based on the composition of the underlying Cloud services SLAs [7]. The MCSLA can act as the contract between the end-users and the developer of the multi-cloud native application and it needs to be assessed at run time. A MCSLA must act as an aggregator of all terms defined in the various SLAs.

In a related H2020 project, Melodic, a novel distributed application monitoring system was introduced – EMS: Event Management System. [8] It is able to collect, process and deliver monitoring information pertaining to a distributed, cross-cloud application, according to CAMEL model specifications, considering the defined SLOs. The aggregated monitoring data is used by Upperware (the Melodic orchestration) to trigger reactively the reasoning process and issue decisions on reconfigurations when and if needed. The big advantage of the EMS approach is its decentralized nature, which is ideal for multi-cloud applications, since it provides a hierarchical filtering of the monitoring information, avoiding bottlenecks and excessive use of network bandwidth.

EMS undertakes the task of deploying a network of agents for collecting monitoring information from the monitoring probes as events, processes them using distributed event processing methods, and forwards the results to Upperware (e.g., Metasolver – the optimisation component). A CAMEL model specifies the needed monitoring information and the kind of processing required, as these have been defined through SLOs. Both the installation of monitoring probes and the deployment of EMS agents is the responsibility of Executionware under the orchestration of the Melodic workflow. EMS is a distributed application monitoring system that comprises of a server integrated in Upperware, named Event Processing Manager (EPM), and several clients, named Event Processing Agents (EPAs). EPM and EPAs form a network of nodes for distributed event processing, called Event Processing Network (EPN). This network is orchestrated and controlled by EPM which is used to, specifically:

- Analyse the CAMEL model of a cross-cloud application in order to extract the required (by other Melodic platform components) monitoring information along with the processing needed.
- Deploy (through Executionware) EMS clients (EPAs), to each distributed application node that hosts an application component (to be monitored).

[3] https://www.influxdata.com/time-series-platform/telegraf/

- Configure each EPA to collect (from sensors) and forward the needed events, and also apply the required complex event processing rules.
- Provide the required information (specified in the CAMEL model), either by updating the application constraints model, by publishing events (any interested party may subscribe to receive them), or by requesting Melodic platform to reconfigure the distributed application (e.g., when certain SLOs are violated).

The EMS is one of the Melodic components being enhanced in another H2020 project – Morphemic. The focus is on resilience, especially in the context of edge deployments, and includes features like self-healing (i.e., automatic healing for the monitoring platform – EMS), clustering and federation.

3.2 Infrastructural elements monitoring approach in PIACERE

3.2.1 Performance Monitoring

The goal within Runtime Performance Monitoring is to continuously gather metrics from the infrastructures deployed and ensure that they continuously meet the expectations defined during the application deployment design phase.

As in the ACSmI [4] we will deploy agents in the deployed infrastructures to gather performance metrics and send them back to the PIACERE Platform. We will use Telegraf [9] open source tool to gather that information. The agents will be deployed as integral part of the application deployment in this sense the Infrastructural Code Generator (ICG) will generate the required IaC configuration not only to deploy the components of the application but also the agents required to ensure the continuous alignment of the infrastructure to the expectations reflected in the DOML.

Apart from the agents gathering the information we will need additional elements to store the data, process it, and dispose it for use by other components. For the deployment of these elements we have two options: deploy them as part of the application or deploy them as part of the PIACERE framework. We have decided to deploy them as part of the PIACERE framework for some reasons:

- It decouples the application from the monitoring and operation
- We can use the PIACERE framework to monitor several applications

For the monitoring part in the PIACERE framework we will include: time series databases to store the information, processing framework to continuously check the NFR, web user interfaces that allow to view and analyse the metrics gathered, and some configuration components to adapt the infrastructure each time an application is deployed.

For the storage of information, we will use influxdb[4] that integrates quite well with Telegraf allowing a firewall-friendly and secure communication from the agents present in the deployed infrastructure and the PIACERE framework. Influxdb is an open source time series database with a big user base. Besides, it can be deployed in different ways supporting a wide range of needs from a container to a cluster. It has a REST API to feed and query information.

For the processing framework and web user interface we will use Grafana[5]. That is also an open source platform. It provides a responsive web-based user interface with a backend that takes

[4] https://www.influxdata.com/
[5] https://grafana.com/

care of the thresholds and the notifications. It also has a REST API to manage the sources, the dashboards, the thresholds and the notifications means.

For the configuration component we will define a PIACERE-oriented REST API, and a Java based server to implement the logic for that REST API.

3.2.2 Security Monitoring

The goal within Runtime Security Monitoring is to provide a security monitoring system for the target infrastructure/application, managed by PIACERE. It complements PIACERE SAST (Statical Analysis Security Tools) technique with a dynamic perspective – using Network Security Monitoring (NSM) tools [10]. The monitoring system will be able to detect suspicious (system and/or application) log entries on the system, configuration changes of the system, file integrity issues, some types of attacks, and malware presence on the system. Network security strategies encompass protection, detection and response processes. Using the runtime security monitoring tools in PIACERE we shall be focusing primarily on the detection and secondary on the response and protection (through the self-learning and self-healing process).

Existing security monitoring solutions implement functionalities such as: sensor, parser, integrator, detector, inspector and actuator [10]. Sensor is collecting data from the target subsystem resulting into records – logs. Parser and integrator can be two components dedicated to transform (normalise) logs into a common format and aggregate the logs onto the central location. Detector is capable of detecting anomalies from the data stream (or the logs), inspector allows data inspection and actuator performs actions on the target system configuration. Network Security Monitoring tools in the market encompass from single-module solutions to a combination of the described modules. Single module solution can be network traffic sensors (sensors incorporating libpcap library such as wireshark[6], tcpdump[7], tshark[8]; traffic sessions capture, such as netflow[9]; traffic statistics using SNMP) and log and state sensors (syslog parsers, application logs parsers). Multi-module solutions have greater importance, since these allow not only collecting but also analytics and detection capabilities. These solutions can further be divided into different classes: Intrusion Detection Systems (IDSs), Intrusion Prevention Systems (IPSs), Security Event Management (SEMs), Security Information and Event Managements (SIEMs), Universal Threat Managements (UTMs). In PIACERE we are targeting existing open-source solutions providing all the modules described above (parser, integrator, detector, inspector and actuator) with the possibility of having specific sensors for the target infrastructure (or the application). OSSEC, Zeek (BRO), Wazuh and Splunk are the potential candidates to be used since all provide the needed requirements (see the list of requirements in section 8.2) and are built on top of open-source solutions/modules.

OSSEC (Open Source HIDS SECurity) [11] is a multi-platform, open source HIDS (Host-based Intrusion Detection System) that performs log analysis, integrity checking, monitoring of Windows records, and rootkit detection. It provides alerts and maintains a copy of the modified files to perform forensics tasks. It has some basic SIEM features, such as allowing the correlation of logs from several devices and formats, and mechanisms for compliance of security policies, but it has been traditionally considered to be an IDS.

Zeek (formerly known as BRO) [12] is a passive network traffic analyser. It supports a wide range of traffic analysis tasks beyond the security domain, including performance measurement and troubleshooting. Zeek has an extensive set of logs describing network activity of every

[6] https://tshark.dev
[7] https://www.tcpdump.org/
[8] https://tshark.dev
[9] https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

connection seen on the wire and also application-layer transcripts (HTTP sessions with their requested URIs, key headers, MIME types, and server responses; DNS requests with replies; SSL certificates; key content of SMTP sessions; and more). By default, Zeek writes all this information into well-structured tab-separated or JSON log files suitable for post-processing with external software.

Wazuh [13] is built on top of OSSEC – it has a robust open-source Intrusion Detection System that performs log analysis, integrating log analysis, file integrity monitoring, Windows registry monitoring, centralized policy enforcement, rootkit detection, real-time alerting, and active response from multiple devices and formats running on most operating systems. This tool has a cross-platform architecture and is centralized, allowing to target multiple systems for monitoring, managing and analysing firewalls, IDSs, web servers, and authentication logs. For each capability, Wazuh has a process defined with specific rules where it is possible to define metrics, for example:

- Compliance level with standards such as PCI DSS, HIPA, GDPR
- Occurrence of changes within system files (file integrity checks)
- Detection of rootkits installed on the infrastructure
- Number and severity of infrastructure vulnerabilities detected (e.g. CVE level of dependencies installed on the OS being monitored)
- Monitoring cloud logs (via IaaS' or PaaS' API, such as AWS' CloudMonitoring)

High level architecture of Wazuh is depicted in Figure 2. Looking at it from high-level, it consists of Wazuh Agents and Wazuh Server. The Wazuh agent (installed on endpoints) with different interfaces (modules) is able to detect different metrics on the host. Wazuh Server consists of worker nodes (Wazuh cluster), Kibana Server and ElasticSearch Cluster.

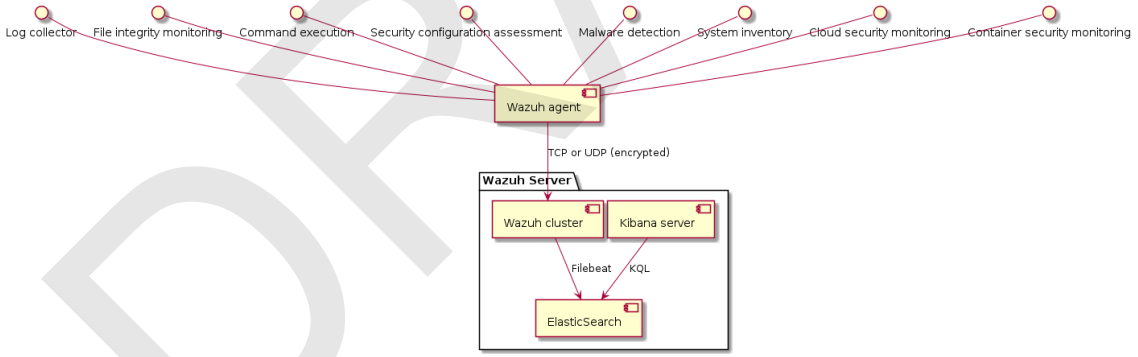

Figure 2. High-level Wazuh's architecture.

Vulnerability assessment in a context of security monitoring process is a systematic review of security weaknesses and can be performed in different ways, but the most common way is by using automated vulnerability scanning software that is usually already provided in the tools mentioned above. Due to the quick changing rate of IT environments and evolution of attacks, vulnerability scans should be performed on a regular basis (continuously collecting security metrics and categorizing these based on some predefined rules).

Devices (infrastructure) can be profiled according to their "behaviour" by exploiting system-level information in order to detect anomalous behaviours. Anomaly detection in the context of security monitoring is analysed in the section 4.2.2.

4 Self-learning

4.1 Self-learning approaches and challenges

Nowadays, many machine learning models in production are still static, i.e., they were developed and trained by data scientists or researchers on historical data, and from that point on they will not be able to incorporate new knowledge. In most real applications data arrive in the form of fast streams, and new data characteristics or trends should be incorporated into the existing models. When they remain static, these models should be retrained on a fairly regular basis (daily or even more frequently). However, this is not very efficient because:

1. It implies that an expert would have to be focused on deciding which is the best moment to train the models again,
2. nowadays data are produced in the form of fast streams, and
3. data are affected by non-stationary phenomena that occur fast, and a human cannot successfully detect changes in a real-fashion environment.

Therefore, some level of automation (*self-learning*) is crucial, and the state of the art is ready to provide us with some interesting solutions. In PIACERE we adopt some of them, taking the IaC (Infrastructure as Code) to the next level of intelligent deployment, configuration and management in the virtualization field.

We would like to start this section by highlighting a non-trivial aspect regarding *self-learning*. We can find in the literature the term *self-learning* referring to *unsupervised learning*, *self-supervised learning*, *self-labelling* or even *reinforcement learning*. In all cases, the idea is to automatically generate some kind of supervisory signal to solve some tasks, e.g., to learn data representations [14] or to automatically label a dataset. In other occasions, it refers to *autoencoders* (neural networks) [15]. However, in PIACERE we adopt the other well-known meaning [16], [17], [18], [19] that refers to the ability of a model of:

- ingesting new data as it becomes available (*incremental learning*),
- detecting by itself changes (*drifts*) in data distribution and to be automatically retrained after this occurs,
- warning the system when anomalies are detected, or
- self-optimizing and self-calibrating in case of performance issues due to *concept drift* or anomalies.

Under these circumstances, *self-learning* becomes a perfect ally in those scenarios where the change or anomaly may be present. An autonomous model allows systems to be more accurate and reliable in production for much longer periods of time. But this is hard to achieve and presents several challenges:

- these models are based on algorithms that are usually more difficult to fine-tune,
- overfitting can be a great concern,
- the stability of the model must be assured,
- false alarms (drift detections) may provoke that the retraining process is useless, even degrading the performance of the model, and
- an anomaly must not be confused with a drift.

The latter point is not trivial [20] given the relevance of it for PIACERE. One of the challenges for *concept drift* handling algorithms is not to mix the true drift with an outlier or noise which refers to a once-off random deviation or anomaly [21], [22]. No adaptivity is needed in the latter case, as Figure 3 shows.

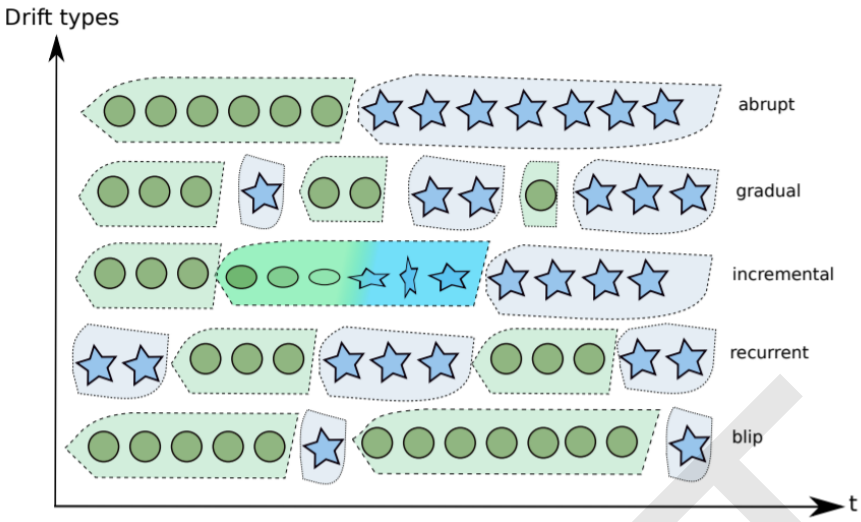

Figure 3. Types of drift according to severity and speed of changes, and noisy blips. Here the stars and circles represent the prevailing concept at every time instant [23].

4.1.1 Stream data analysis

Applications generating huge amounts of data in the form of fast streams are increasingly prevalent. These applications collect data from almost any source and analyse it to find answers that enable cost and time reductions, new product developments, optimized offerings, or smart decision making, or –as in our case– try to improve the deployment process of Infrastructure as Code (IaC). In these scenarios, instead of all training data being available from the beginning, data are often received over time in streams of samples or batches. Data streams are the basis of the real-time analysis, which is composed by sequences of items, each having a timestamp and thus a temporal order. A stream data environment shows several particularities [24] that we should consider when designing our algorithms:

- Each sample or batch is processed only once on arrival. Stream data analysis solutions should be able to process information sequentially, according to its arrival. These solutions must not put the resources (mainly memory space and processing time restrictions) at risk,
- The processing time must be small and constant, without exceeding the ratio in which new samples arrive. Otherwise, some kind of temporal storage should be considered,
- The stream data analysis solution should use only a pre-allocated amount of main memory
- The model/algorithm in which this stream data analysis solution is based, should be completely trained before next sample arrives

In Infrastructure as Code (IaC) platforms, data also arrives in the form of data streams, and thus it may suffer *anomalies* and *concept drift* phenomena, as we will see later. Finally, it deserves mentioning that data streams in IaC are usually in the form of time series, and thus the temporal dependence in data is present; and it should be considered properly. There is a surprising research in [25], where the author explains that non-change detectors can outperform change-detectors when used in a classification streaming evaluation. This may be due to the temporal dependence on the considered data, and then the evaluation of change detectors should not be done using only classifiers. PIACERE is a perfect candidate to present such particularity.

4.1.2 Anomaly detection

Data analysis nowadays faces a number of challenges. One of them has been extensively studied due to its importance on the field: Anomaly detection. When analysing real-world data, data that differs from the norm can be found, such data is called an anomaly or outlier. Anomalies can be caused by inaccurate concept, this is, data that is unexpected by the current comprehension of the phenomenon. Hawkins [26] defines an outlier as *"an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism"*. Anomalies are also referred to as abnormalities, deviants, or outliers in the data mining and statistics literature [27].

Anomaly detection provides a set of algorithms and techniques that can be used to spot out the instances dissimilar to all others. Among the most popular techniques the following algorithms can be found:

- One-Class SVMs [28]: An extension of the support vector machine standard algorithm [29].
- Local Outlier Factor [30]: Algorithm that instead of performing a binary classification estimates the probability of an instance being anomalous.
- Isolation Forest [31]: A binary tree-based search that tries to isolate anomalies. An online version called Half Space Trees [32] also exists.
- Elliptic Envelope [33]: An algorithm based on the minimum variance determinant [34] estimator that analyses in an elliptically symmetric unimodal distribution.

In recent years, an important growth of deep neural networks, a subset of the machine learning field, have been seen, with astonishing outcomes in different application areas, also when applied to anomaly detection [35] [36]. Therefore, deep learning-based anomaly detection (DAD) algorithms have obtained a privileged position and are one of the main focus areas. It is important to note that boundaries between abnormal and normal data is not precisely defined in evolving environments. This lack of boundaries represents challenges for both conventional and also deep learning techniques.

Due to the large-scale nature of the data to be analysed in the IaC platform, it becomes nearly impossible for the traditional machine learning techniques to scale and find anomalies properly. DAD techniques are able to handle these large amounts of data and are also able to learn hierarchical discriminative features solving the problem end-to-end by removing the need of developing manual features by domain experts.

Through decades of study in the field, we can find an important amount of techniques in the literature. The following list includes some of the most important set of techniques used with major achievements:

- Autoencoder (AE) [37] is an artificial neural network type that tries to learn a representation for a set of data in an unsupervised manner.
- Deep Belief Networks (DBN) [38] is a class of deep neural networks, that act as a feature detector by reconstructing probabilistically its inputs.
- Long short-term memory (LSTM) [39] [40] is an artificial recurrent neural network (RNN) developed to deal with the vanishing gradient problem, that is well suited to classifying, processing and making predictions particularly on time series data.
- Deep Neural Networks [41] are a set of artificial neural networks that use multiple layers in the network used to solve a wide set of problems in fields including audio recognition, computer vision, natural language processing and speech recognition.

- Convolutional Neural Network (CNN or ConvNet) [42] [43] [44] is a class of deep neural network, an artificial neural network with multiple hidden layers that are very successful in different fields like computer vision, natural language processing, image classification among others.
- Denoise Autoencoder, Stacked Denoise Autoencoder (DAE, SDAE) [45] [46] [47] are an alternative to the concept of regular Autoencoder, where the data is partially corrupted by noises and are trained to predict uncorrupted data.
- Recurrent Neural Network (RNN) [48] [49] is a class of artificial neural networks where directed graph is used to make connections between nodes and an internal state is used to process inputs.

However, new techniques and approaches are also being studied that offer better results, sometimes using less resources than classic techniques [38] [39].

4.1.3 "Concept drift" detection

The data generation process in these real-time applications is not always stationary because it is subject to dynamic externalities that affect the stationarity of such data streams, e.g., seasonality, errors, etc. This causes that such applications suffer from the *concept drift* phenomenon. The predictive models that are trained over these data streams may become obsolete and having problems to adapt suitably to the new conditions. Thus, in these scenarios there is a pressing need for drift detection and adaption algorithms that detect and adapt to these changes as fast as possible, in order to keep the applications updated and providing a good performance [50]. The research on concept drift is still a hot topic due to its impact on real world applications, and as we will show, PIACERE is not an exception.

Many research efforts have been dedicated to study and alleviate the effects of the *concept drift* phenomenon [51], and this has been the case for the last 3 years. The complexity in *concept drift* manifests when we try to characterize it [52]. We can find many different types of concept drifts (see Figure 3) which can be characterized by e.g., the speed or severity of change. Consequently, drift detection turns into a relevant factor for those active mechanisms that need a triggering mechanism to perform an adaptation after drift occurs [53]. A drift detector should estimate the time instant at which change occurs over the data stream, so that when the detection appears, the adaptation mechanism is applied to the base learner in order to avoid the degradation of its predictive performance. The successful design of an effective detector is not straightforward, yet it is primordial to achieve a more reliable system. The way to find the best strategy for concept drift detection still remains as an open research issue, as confirmed in [53]. This challenge to find a universal best solution becomes evident in the most recent comparative among drift detectors carried out by [54]. In light of the results achieved in this manuscript, we can realise that there is not a method with the best metrics, or even showing the best performance in most cases. We can state that the ideal goal is to develop detectors that 1) detect all existing drifts in the stream 2) with low latency, 3) with as few false alarms and, 4) as few missed detections as possible, and 5) minimizing the distance of the true positive detections, always assuring a good classification performance. Therefore, as there is not an ultimate detector, we will have to choose one depending on the characteristics of the application or scenario, giving more importance to some metrics than others (false alarms, missed detections, distance of the real drift, etc.).

Finally, the operation of a drift detector (see Figure 4) usually utilises a specific base learner (i.e., learning algorithm). The base learner is trained on the current instance of the data stream within an incremental learning process repeated for each incoming instance. The detector is analysing all the time the classification performance of the base learner (e.g., accuracy or error rate) to know whether a drift has occurred or not. Although the accuracy or error rate are often used as

inputs of the detector, others use diversity [55] or structural changes stemming from the model itself [56].

Drift detectors use different strategies to monitor the performance of the base classifier and to decide if a drift has occurred or not. A common practice is to use a lower confidence level to denote a warning, which means that a drift may have happened. If this happens, then detectors prescribe that a new base classifier is created, and it starts to be trained in parallel. Then, if a concept drift is confirmed (e.g., because the number of consecutive warnings has exceeded a threshold), the new base learner will replace the original one. However, if the warning has not been confirmed and it is a false alarm (false positive), the new base learner will be discarded.

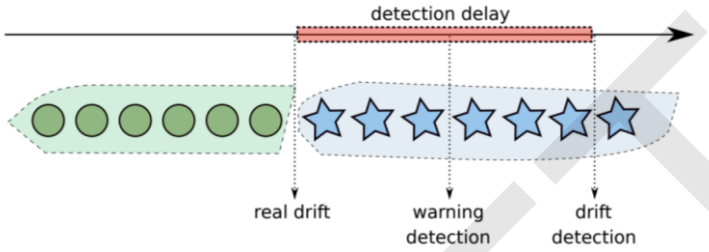

Figure 4. Drift detection example [23].

We can find a plethora of drift detection methods in the literature. Next, we delve into the details of the most well-known and used drift detectors. Some of them have recently been compared in a very remarkable study [57], so we will literally refer to some of its findings, explanations and methods. This work is crucial in the state-of-the-art, and from our view it does not make sense to reword such parts:

- DDM [58]: this detector acts as follows: when the concept changes, the base learner will incorrectly classify the arriving instances that are created based on a different data distribution. Therefore, if the error-rate increases, it is signal that a concept drift has occurred. Whereas, if the distribution remains stable (without changes, stationary), the error rate will decrease.
- EDDM [59]: similar to DDM, but instead of using the error rate, EDDM uses the distance between classification errors (number of examples between two classification errors) of the base learner to indicate if a drift has happened.
- ADWIN [60]: uses a sliding window of instances with a variable size W. When drifts are detected, W is reduced and the longer the concept the larger the size of W. Two dynamically adjusted sub-windows are stored, representing older and recent data. Drifts are detected when the difference of the means of these sub-windows is higher than a given threshold.
- STEPD [61]: it uses a statistical test of equal proportions, with continuity correction, calculated over two windows of the processed data, named recent and older. The accuracies of the base learner over these two windows are expected to be the same within each concept. Warnings and drifts are signalled when a significant difference is detected in the accuracy of the recent window.
- PHT [62]: it is a sequential analysis technique that computes the observed values (the actual accuracy of the base learner) and their average to the current time step. When a drift occurs, the classifier starts to fail to correctly classify new instances, making the current and the mean accuracy decrease.
- PL [63]: uses two learners. The stable learner (S), which uses all known instances for training, and the reactive learner (R), that only trains on the last W instances, which is a parameter. The number of instances incorrectly classified by S but correctly classified by

R is kept updated and, if its proportion of W is greater than a parametrized percentage threshold θ, a drift is detected. After concept drifts are triggered, S is replaced by R and R is reset.

- ECDD [64]: it is like EWMA [65] but to be used in data streams subject to concept drifts. EWMA detects significant changes in the mean of a sequence of random variables provided that the mean and standard deviation of the data are known in advance. However, in ECDD, the mean and standard deviation are not needed.
- HDDM [66]: It is a method based on Hoeffding's bounds with a moving average-test. There are two main versions: one uses the average as estimator, the other one uses the EWMA.
- FHDDM [67]: it uses a sliding window on the classification results and inserts a 1 into the window if the prediction result is true, otherwise it inserts 0; then the maximum (best) probability of the correct predictions observed so far is compared with the most recent probability of correct predictions: a drift is detected if the difference is greater than a threshold calculated with Hoeffding's inequality [68]. This algorithm results in less detection delay, fewer false positives and fewer false negatives.
- SEED [69]: it also draws on ADWIN and compares two sub-windows within a window W. Whenever these two sub-windows of W exhibit distinct averages higher than a chosen threshold δ, the older portion of the window (WL) is dropped. SEED uses the Hoeffding Inequality with Bonferroni correction, proposed in ADWIN, to calculate its test statistic and it also performs block compression to eliminate unnecessary cut points and merge blocks that are homogeneous in nature.
- RDDM [70]: it was proposed to alleviate a performance loss problem of DDM when concepts are very large, caused by decreased sensitivity, requiring too many instances to detect the changes. RDDM adds an explicit mechanism to discard older instances of very long concepts, periodically recalculating the RDDM statistics responsible for detecting the warning and drift levels. In addition, it forces concept drifts when the number of instances of the warning period reaches a parametrized threshold.
- WSTD [71]: it also uses two windows of data, like STEPD, but detects drifts based on an efficient implementation of the Wilcoxon rank sum statistical test [72], instead of the test of equal proportions used in STEPD.

It also deserves highlighting in this section the latest drift detection techniques, above all those which have shown a potential to impact on this field, mainly due to their citations in the last 3 years, the relevance of the journal/conference in which have been published, the relevance of the authors in the field, among others: [73], [74], [75], [76], [77], [78], [79], [80]. Despite they show less relevance than the upper ones, we consider them remarkable for being recent or being published in reputed journals or conferences.

4.1.3.1 Drift meaning in monitoring platforms

Looking for this phenomenon in monitoring platforms, we see how it is already a relevant problem to deal with.

Drift detection is important for ops teams to ensure that components are in line with the expected configuration and also to ensure compliance. For these teams, **infrastructure drift** is when there is an unwanted delta between the IaC code base and the actual state of the infrastructure. This issue becomes more and more complex as the number of environments grows. Some teams have dozens of environments that they need to keep updated. *Driftctl*[10] is an open source tool which can detect drift in Terraform managed infrastructure. It reads Terraform state files and checks that against the actual running infrastructure. The authors of

[10] https://driftctl.com/

driftcl spoke to around 200 DevOps teams to learn about infrastructure drift challenges, and they identified three main causes of drift:

- 96% of teams: a team member makes a change through the (AWS, Azure, etc) console or directly updates infrastructure resources through an application API,
- 44% of teams: a team member applies an IaC change to an environment but does not propagate it to other environments,
- 50% of teams: application and deployment induced drift.

While the first two are mostly workflow issues, the last one refers to an unintentional application and deployment induced drift, and it is completely independent from the DevOps team. Due to its unpredictable characteristic, it may cause headaches. Drift always happens, and the key challenge is being able to detect and analyse it; the faster it is detected, the easier it is to remediate drift.

AWS has the CloudFormation Drift Detection feature [81], which allows organizations who have templated their configurations and deployments, known as stacks, to detect when configuration drift occurs from out-of-band changes. These out-of-band changes have been directly applied to cloud assets, instead of leveraging a templated deployment approach. To avoid **configuration drift**, Amazon is suggesting the customers use a CloudFormation Change Set to apply changes. This way the deployment template is kept up to date and can be used to provision AWS services in a consistent manner. Drift can be detected within a few minutes from the out-of-band changes being applied so that administrators can quickly address this. Differences in configuration are detected by comparing the current stack configuration with the one specified in the template and identifying divergence. In addition, detailed information for every difference is provided.

As we see, the “concept drift” has been seen in monitoring platforms from different perspectives (configuration drift, infrastructural drift). In PIACERE we will adopt the classical view explained in section 4.1.3.

4.2 Self-learning approach in PIACERE

4.2.1 Performance Self-learning

Once the most well-known and the latest techniques have been presented in previous subsection, this one delves into the self-learning strategy for PIACERE.

The Self-learning component focuses on incrementally online learning and predicting the performance of the system to guarantee constant high-level performance. The percentage of CPU that is not actively being used (aka. *CPU usage idle*) is the metric that we use for this goal. It gives us a good idea of the "health" of the system. Once this prediction goes below a limit (i.e. 70%), the component will trigger a warning to the Self-healing component, which will decide how to consider such a warning (launching an optimization process, redeployment actions, etc.).

As we have already mentioned, the self-learning capability of this component refers to the ability of ingesting new monitoring data as it becomes available (*incremental learning*), and then make a prediction for the next time step in an online manner. The problem is that, in many occasions, real-time monitoring data may suffer from “rare” events that we need to early detect if we want to have a solid, robust and reliable system. In PIACERE these events are:

- Changes (*concept drifts*) in data distribution, and
- Anomalies.

The impact of these events on the prediction performance is different, and also the mitigation actions for each case. We will give more details of each event in the next subsections.

4.2.1.1 Monitoring data and incremental online learning

The performance monitoring data in PIACERE is in the form of time series, which means that there is a data point every 1 hour with a value of *CPU usage idle* in that moment (see Figure 5). Therefore, the *CPU usage idle* becomes our target variable to predict.

	usage_idle	moment
0	96439800000000	2021-08-17 09:00:00
1	9646530000000000	2021-08-17 10:00:00
2	964161000000000	2021-08-17 11:00:00
3	9641840000000000	2021-08-17 12:00:00
4	9648480000000000	2021-08-17 13:00:00
..	...	...
716	9595840000000000	2021-09-16 05:00:00
717	9602310000000000	2021-09-16 06:00:00
718	9597140000000000	2021-09-16 07:00:00
719	9600030000000000	2021-09-16 08:00:00
720	9607420000000000	2021-09-16 08:00:00

Figure 5. The whole toy dataset from 2021-08-17 to 2021-09-16.

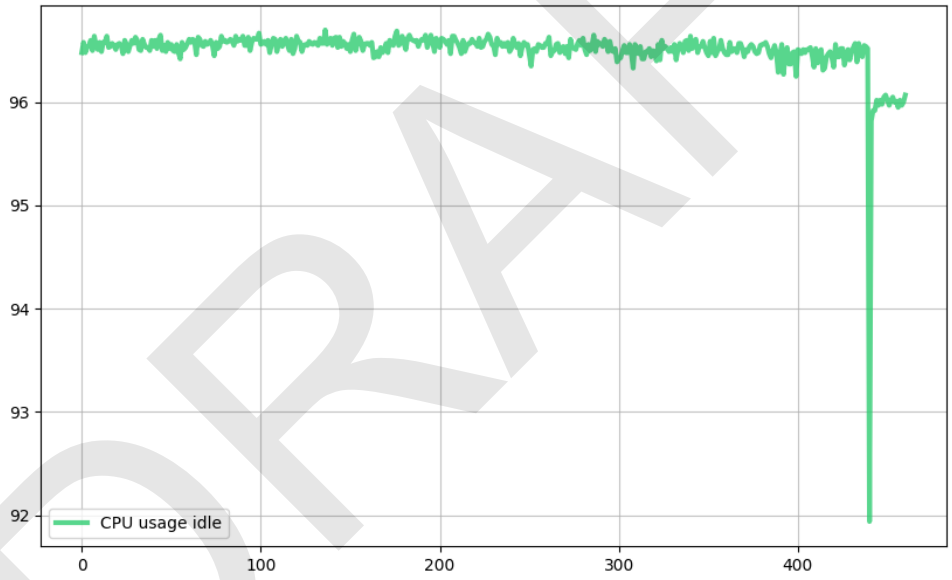

Figure 6. CPU usage idle in the testing period of the algorithm (from 2021-08-28 05:00:00 to 2021-09-16 08:00:00).

The toy dataset, which is composed by 720 data points, has been divided into two slices: the first one (the first 260 instances) to tune the parameters and train the algorithm, and the second one (the rest 460 instances) to test the performance of the algorithm. Then, we have tuned and trained a **Random Forest Regressor**[11] that is able to incrementally learn every time a new instance arrives. By following the *test-then-train* scheme used in many online learning approaches [82], we have used Mean Absolute Error (MAE) as performance metric for regression problems. Figure 7 shows how the algorithm is able to successfully predict the next *CPU usage idle* data point 1 hour ahead. The algorithm exhibits a very good predictive performance with a MAE of 0.042 and a standard deviation of 0.004.

[11] https://riverml.xyz/latest/api/ensemble/AdaptiveRandomForestRegressor/

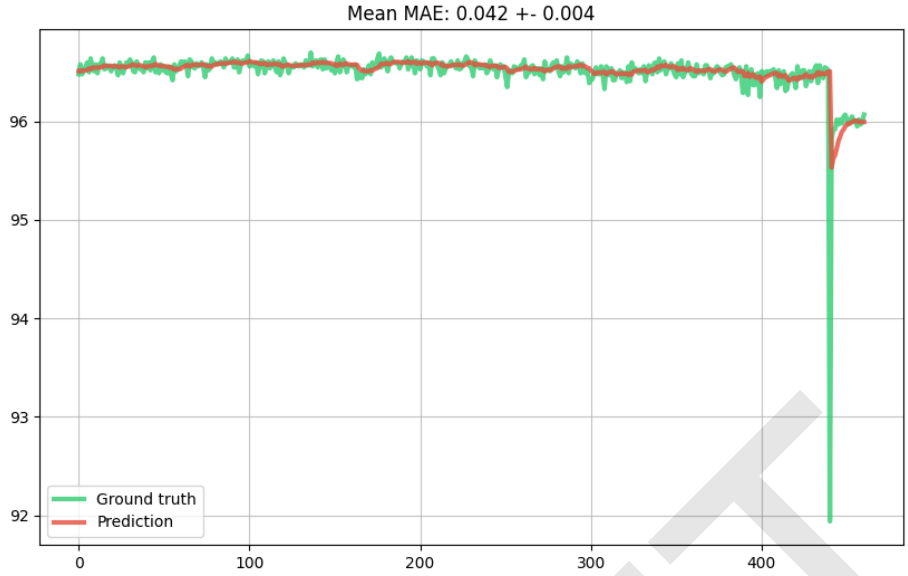

Figure 7. The incremental learning process.

4.2.1.2 Anomaly detection

As we have previously seen, anomalies are one of the rare events that may appear in PIACERE data. They should be early detected in order to keep the prediction performance under control. Once they have been detected, the algorithm should not learn these data points in order to be robust; these outliers do not belong to the normal distribution of the monitoring data.

As there is only one outlier in the toy dataset, and in order to test our algorithm in a more real situation, we have artificially generated some of them at t=40, 90, 140, and 190. Figure 8 shows how the algorithm without an outlier detection method, learns these data, and the performance decreases after the outliers appear and are learnt. Once the algorithm continues learning the data points of the normal distribution, the performance once again becomes competitive.

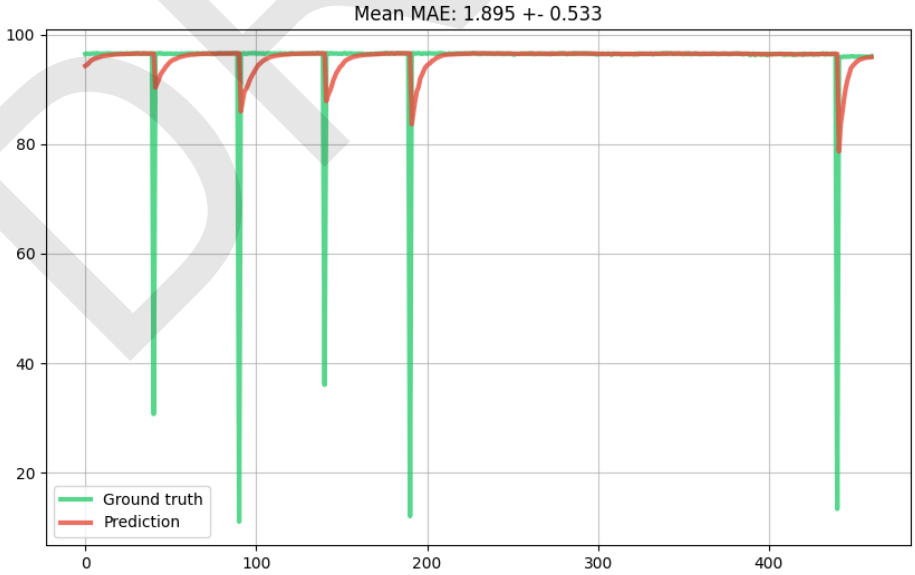

Figure 8. Learning process in the presence of outliers without detection.

In order to prevent such performance degradation in our component, we have used a **Half Space Trees**[12] (the online variant of isolation forests) to detect the outliers before the learning of the data points. Then, the learning process avoids the data points detected as outliers.

We see in Figure 9 how these outliers are not affecting the predictive performance of the algorithm. While the scheme without detection showed a MAE=1.895+-0.533, now the one with this capability gets a MAE=1.17+-0.325. Now we have achieved a self-learning component robust to outliers, and then the online prediction will be more reliable.

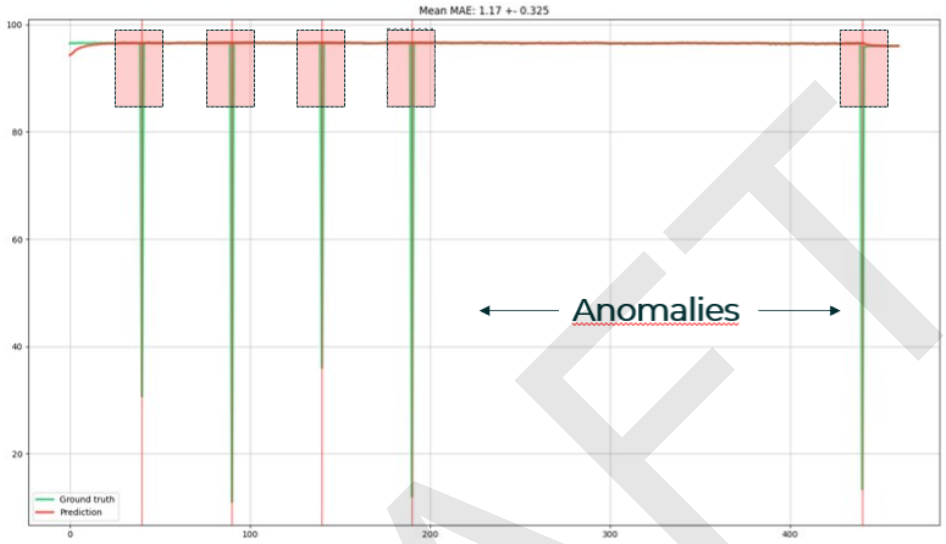

Figure 9. Learning process in the presence of outliers with detection.

4.2.1.3 Concept drift detection

The consideration of the concept drift detection in our algorithm is still pending. The toy dataset does not exhibit the concept drift phenomenon, so as we did with anomalies, we will provoke it in order to prepare our algorithm for this real possibility. In the next version of this deliverable, we will show the results for this relevant part.

4.2.1.4 Next actions

- We will prepare our algorithm for the concept drift phenomenon.
- We will try the possibility of predicting several time steps ahead, not only the next one (1h). Despite it is not crucial for the current system in PIACERE, because 1 hour is usually more than enough to perform mitigation actions (such as redeployments, optimizations, etc.), it would be a good ingredient for the project.
- We need to obtain real outliers and real drifts from our system. The real validation is always better than the artificial one.

4.2.1.5 Self-learning relevance to the use cases

As we have already mentioned in the state-of-the-art, both outliers and drifts detection are primordial to get a reliable online learning prediction in any real application based on real data. In this case, in PIACERE, the monitoring data is present in any use case, so the prediction of the health of the system will be a relevant part in it.

[12] https://riverml.xyz/latest/api/anomaly/HalfSpaceTrees/

4.2.2 Security Self Learning

Computer-generated log messages are a very valuable source of information to represent the current status of a system or application. Log messages are precisely generated to provide application developers and system operators with information that could help them, among other things, understand execution paths, find bugs or solve incidents. Generally speaking, when a problem occurs, logs are often relied upon for investigation.

In an operational phase, the automatic analysis of these logs could thus provide valuable insights regarding the current and past status of the monitored assets. Many research works have tackled this problem but there are still open research questions and further improvements lay ahead. Furthermore, in most cases, they are limited to the research sphere, and their application to real-world use cases is yet to be explored. Below, we analyse the most relevant contributions in order to understand the current state of the art.

4.2.2.1 Terminology

Most of the works follow a similar terminology with respect to log analysis. A log *message* usually refers to the text part of a log, once other fields such as timestamp, log level, component, etc. have been removed. Examples of log messages could be: *'Deletion of file file0 complete'* or *'Took 2.67 seconds to create VM'*. Log messages can be composed of both fixed and variable parts. The first is usually referred to as *log key* or *log template* and the latter as *log parameters*. Thus, log templates of the previous examples would be *'Deletion of file <*> complete'* and *'Took <*> seconds to create VM'* and their corresponding parameters (substituted by '<*>' in the templates), *file0* and *2.67*. The process of discerning which parts of a log message are fixed and which are parameters is referred to as log *parsing* and is usually the first step in a typical pipeline for log analysis.

4.2.2.2 Anomaly detection in logs

DeepLog (2017)

One of the most relevant works on the application of deep learning to log analysis is DeepLog [83]. It describes a framework composed of three different models: a log key Anomaly Detection (AD) model and a parameter value AD model, both based on stacked LSTM (Long Short Term Memory) networks, and a workflow model to help diagnose the detected anomalies (root cause analysis - RCA). Models are trained only with logs produced during normal execution of the system (i.e. with no anomalies). Spell [84] is used as the log parser. We will analyse the AD models, since they are of higher interest for our purposes.

In DeepLog, the log key model tries to predict the next log key in a sequence as a multi-class classification problem. During the inference phase, the trained model estimates which log keys are most likely to appear next, providing an ordered list of all known log keys. The N first candidates are considered to be 'normal' -since they are expected by the model- so if the true log key coming next is not within those candidates, it will be deemed anomalous. This is a common approach to decide about the abnormality of a log key depending on the parameter N, and it appears in several other works. We will refer to it as the candidate set approach. As this is a classification approach, the method is unable to deal with log keys not seen during training. DeepLog tackles this by allowing manual feedback from the user in case of detected false positives (FPs), but this solution is not scalable and requires manual intervention.

Regarding parameters, DeepLog considers each sequence of parameter value vectors -a log key may contain several parameters- for a specific log key as a separate time series. It only handles numerical parameters. During training, the validation set mean square error (MSE) is modelled

as a Gaussian distribution that is later used to determine abnormality during the inference phase.

DeepLog provided good results in several public datasets and set the state of the art at the time, but with the surge in interest towards this topic, and thus the proliferation of research works, several other novel approaches have been proposed outperforming DeepLog in those same datasets.

LogAnomaly (2019)

LogAnomaly [85] also makes use of LSTM networks in this case to detect both sequential -log order- and quantitative -log count or frequency- anomalies. The main contribution of LogAnomaly is the use of a method based on the popular word2vec [86] to capture the semantic information in log messages. This method, template2vec, converts the words in log templates into word embeddings and combines these to form template embeddings. However, it requires the use of a corpus of synonyms and antonyms, some of them manually defined to make them domain specific, which makes it very impractical.

To detect quantitative anomalies, they compose an additional representation of a log sequence by counting the appearance of every log template in the sequence. An additional LSTM model is trained to learn the quantitative pattern of the log sequence, whose output is then combined through an attention mechanism with that of the template embeddings LSTM. Only normal logs are used for training and FT-Tree (frequent template tree) [87] is used as the log parser.

LogAnomaly also uses a candidate set approach to detect anomalies during inference, but they also accept 'similar' candidates -they can measure distance between embeddings. Consequently, even if they capture the semantics of log templates to some extent, they still cannot handle previously unseen log templates during inference. Their solution is to approximate unseen log templates to the 'closest' one already included in the training set.

LogRobust (2019)

LogRobust [88] introduces an architectural change, describing a Bidirectional LSTM network in order to capture information from sequences in both directions (i.e. 'past' and 'future'). An attention mechanism is applied to the output of the Bi-LSTM to combine outputs from all time steps. Contrary to DeepLog and LogAnomaly, LogRobust tackles the problem as a binary classification method, classifying log sequences as either normal or abnormal. This is, it works in a supervised manner, so it requires labeled anomalies instead of just log data from normal operating conditions. LogRobust uses Drain [89] as the log parser.

Similarly to LogAnomaly, LogRobust leverages the semantic information within log templates. To do so, off-the-shelf word vectors pre-trained on the Common Crawl Corpus dataset are used. The vectors for the words in a log template are aggregated using TF-IDF weights, thus generating a fixed-dimension vector representing every log template, regardless of the number of words in it. This implies that they can handle any log template, also those unseen during training.

HitAnomaly (2020)

HitAnomaly [90] is the first work that leverages the Transformer [91] architecture for AD in log sequences. In order to capture the semantic information in log templates, they define a 'log encoder' architecture that takes as input the words within a template, and outputs a fixed-dimension vector representing the template. Sequences of these vectors are then fed to a 'log sequence encoder', which eventually outputs a fixed-dimension representation of the whole template sequence. Both encoders use a very similar architecture, with the only difference that

the log encoder stacks two transformer blocks, while there is only one in the log sequence encoder. HitAnomaly uses Drain [89] as the log parser.

Parameters within a log sequence are also encoded using a 'parameter encoder' with the same architecture as in the log encoder. Interestingly, the parameter representation (output of the parameter encoder) and the log template representation (output of the log encoder) are combined to capture interaction between a template and its parameters. Finally, the log template sequence representation and the log parameters sequence representation are combined through an attention mechanism and fed to a binary classifier.

HitAnomaly showed state-of-the-art results in terms of overall performance on public datasets as well as impressive results when dealing with high shares of previously unseen log templates.

NeuralLog (2021)

NeuralLog [92] does not provide any further advancement in terms of the proposed architecture, which is based on transformers. However, instead of relying on log parsing, known to be an important source of noise that severely conditions the AD results, they directly employ raw log messages, preprocess them, apply WordPiece tokenization and obtain the semantic information using a pre-trained BERT [93] model.

LogBERT (2021)

LogBERT [94] is the first work leveraging the transformer architecture working in a self-supervised fashion (i.e. training with 'normal' logs only). Drain [89] is used to parse the log messages and a unique id is assigned to each of the obtained log templates. Therefore, LogBERT does not capture the semantic content of log templates by any means. A transformer encoder architecture is trained on sequences of these ids using two different tasks: Masked Language Modelling (MLM) and Hypersphere Volume Minimization (HVM).

For MLM, template ids are randomly masked, and the model is used to predict the expected ids for the masked tokens in a classical multi-class classification approach. For HVM, an initial special token is trained to represent the whole log template sequence in terms of normality: tokens representing normal sequences are to be concentrated around the centre of the hypersphere. The distance to the centre will be used during inference to measure abnormality of a log sequence.

Harold Ott et al. (2021)

In the work by Harold Ott et al. [95], they explore the use of sentence-level embeddings obtained from pre-trained language models as log template representation. Their architecture consists of a Bi-LSTM network. The main novelty in this works lays in the comparison of two different tasks for self-supervised AD: the already mentioned candidate set approach, in which the Bi-LSTM is used for multi-class classification, and a regression approach in which the loss function is the MSE between the template embeddings.

4.2.2.3 Anomaly detection in security logs

All the reviewed approaches provided performance results in publicly available datasets. These datasets contain logs from supercomputers (BGL and Thunderbird [96]) or distributed systems (HDFS [96] and Openstack [83]). None of them was specifically validated in security data. Only DeepLog [83] provided results for the VAST Challenge 2011 [97] - MC2 data set, which is a small dataset for which detection results were satisfactory, correctly identifying log template anomalies in 5 out of the 6 suspicious activities and raising a single FP (False Positive).

Specifically, in the domain of cybersecurity, there are a few works that have proposed varied solutions to leverage AI models for security monitoring. For instance, [98] describes an active learning framework that uses unsupervised outlier detection on predefined features computed from raw data (e.g. number of successful logins) and presents rare events to an analyst. The analyst's feedback is then used to train a supervised model that would predict whether future rare events are malicious or not, complementing the unsupervised model. The framework runs on a periodical basis (e.g. daily), collecting analyst's feedback and retraining the models. The framework is validated using a proprietary credit card transactions dataset.

The use of unsupervised deep learning approaches for insider threat detection was explored in [99]. Specifically, common DNN (Deep Neural Networks) and LSTM architectures were employed for the CERT Insider Threat Dataset [100]. The input to these was composed of categorical variables (e.g. user's role, department, etc.) and engineered 'count' variables (e.g. number of logins between 12 AM and 6 AM). However, through experimentation, the categorical variables were proven unhelpful. Daily aggregation was carried out for numerical features. As future lines of work, the authors mentioned the analysis on a per-log basis to reduce or remove the feature engineering required.

The works specifically designed for security log monitoring using AI are scarce and do not yet make use of state-of-the-art architectures. Conversely, many works have been recently proposed for anomaly detection in logs, with an increasing number of research publications on the topic every year. The application of more advanced log anomaly detection methods to cybersecurity use cases remains an appealing open challenge.

5 Self-healing

5.1 Self-healing strategies and challenges

The scope of the self-healing in PIACERE is reduced to the gathering of detected anomalies anta taking corrective actions. Anomalies are gathered from other components in the PIACERE framework such as those devoted to monitoring or those performing forecasts based on the gathered trends or evidences. Corrective actions are executed by the same infrastructure that takes care of the configuration of the infrastructure and the applications. In this sense we could say that our scope is going to be mainly in the planning of the self-healing.

In literature we can find different approaches that extend the scope of the self-healing covering other phase both before and after that planning activity. The Table 1 below shows a comparison of the potentially related approaches with respect to: the covered phases of autonomic control loop "MAPE-K (Monitor-Analyse-Plan-Execute over a shared Knowledge)", technique used, whether it is a decentralized or centralized approach, and the applied context.

Table 1. MAPE-K results

Reference	MAPE-K	Technique	[De]centralized	Context
Di Nitto et al. [101]	M, A, P, E, K	Probability theory	Decentralized	Microservice
Maimó et al. [102]	M, A, P, E, K	Deep learning	Centralized	5G networks
Yang et al. [103]	M, A, P, E, K	Security theory	Decentralized	IoT-based healthcare storage
Alhosban et al. [104]	M, A, P, E	Predictive model	Centralized	Cloud service
Azaiez and Chainbi [105]	M, A, P, E	Multi-agent	Decentralized	Cloud
Gill et al. [106]	M, A, P, E	Reactive	Centralized	Cloud service
Li et al. [107]	M, A, P, E	Reactive	Centralized	Cloud computing
Magalhaes and Silva [108], [109]	M, A, P, E	Statistical theory	Centralized	Web application
Rajput and Sikka [110]	M, A, P, E	Multi-agent	Decentralized	Distributed environment
Rios et al. [111]	M, A, P, E	CAMEL based	Decentralized	Distributed environment
Mosallanejad et al. [112]	M, A, E	Reactive	Centralized	Cloud
Wang et al. [113]	M, A	Machine learning	Centralized	Cloud computing

Several approaches for self-healing/self-adaptive systems are provided in the table above. In [114], a survey of self-healing frameworks and methodologies in multi-tier architectures is provided by Schneider et al. They provide a comparative analysis of the computing environment, degree of behavioural autonomy, and organisational requirements of these approaches. Another survey of self-healing systems with the focus on approaches is provided by Psaier et al. [115]. In [116], Taherizadeh et al. provide a survey to identify the main challenges in the field of monitoring edge computing applications; to present a taxonomy of monitoring requirements for adaptive applications orchestrated upon edge computing frameworks; and to discuss and compare the use of cloud monitoring technologies. In [117], Esfahani et al. characterize the sources of uncertainty in self-adaptive software systems, and demonstrate their impact on the system's ability to satisfy its objectives. They provide an alternative notion of optimality that explicitly incorporates the uncertainty underlying the knowledge (models) used for decision making. They also discuss the state-of-the-art for dealing with uncertainty in this setting. A book chapter by Weyns [118] provides a particular perspective on the evolution of the field of self-adaptation in six waves including: i) automating tasks, ii) architecture-based adaptation, iii) models at runtime, iv) goal driven adaptation, v) guarantees under uncertainties, and vi) control-based approaches.

In Table 1, we compare several selected approaches with respect to four perspectives: the covered phases of autonomic control loop "MAPE-K" (known as: Monitor, Analyse, Plan, Execute, Knowledge) in the system, the technique used by the approach, whether it's a decentralized or centralized approach, and the applied context. The approaches in the table are ordered by their respective coverage of the autonomic control loop, from those which cover all the phases to those which cover only a few of them. The details of these methods are summarized as follows.

Di Nitto et al. [101] propose an approach named Gru based on multiagent systems that add an autonomic adaptation layer for microservice applications focusing on Docker. Gru is designed to support decentralized microservices management, and can be integrated with ease in dockerized applications, managing them with autonomic actions to satisfy application quality requirements. In [102], Maimó et al. propose a 5G-oriented cyberdefense architecture to identify cyberthreats in 5G mobile networks. Their architecture uses deep learning techniques to analyse network traffic. It allows adapting, automatically, the configuration of the architecture in order to manage traffic fluctuations, to optimize the computing resources and to tune the behaviour and the performance of analysis and detection processes. In [103], Yang et al. propose a privacy-preserving smart IoT-based healthcare big data storage system with self-adaptive access control, aiming to ensure the security of patients' healthcare data, realize access control for normal and emergency scenarios, and support smart deduplication to save the storage space in big data storage system.

Alhosban et al. [104] propose a self-healing framework for cloud-based systems, which uses the previous history to detect faults and a recovery plan to avoid future faults. Azaiez and Chainbi [105] propose a multi-agent system which interacts with the Cloud infrastructure to analyze the resources state and execute Checkpoint/Replication strategies or migration techniques to solve the problem of failed resources. Gill et al. [106] present an intelligent and autonomic resource management technique named RADAR with the focus on two properties of self-management that provide self-healing by handling unexpected failures and self-configuration of resources and applications. Li et al. [107] propose a self-healing monitoring and recovery model in cloud computing environments working in three steps: 1) monitoring the system to identify faults, 2) finding out the properties of faults, and 3) recovering from faults using an undo strategy. Magalhaes and Silva [108] propose a self-healing framework for web-based applications to fulfill the user SLA and improve resource utilization simultaneously through self-adaption of cloud

infrastructure. In [109], the same authors present a framework to provide the web-based applications with the ability to detect performance anomalies at runtime and trigger automatic recovery actions to mitigate their impact. Rajput and Sikka [110] propose an architecture which could support agent-based distributed systems to address fault recovery for achieving self-adaptiveness. Rios et al. [111] propose a Cloud Application Modelling and Execution Language (CAMEL) based model for self-healing to model the multi-cloud applications in the distributed environment.

Mosallanejad et al. [112] propose an SLA based self-healing model for the cloud environment to monitor SLA and detect SLA violation automatically. Wang et al. [113] propose a self-adaptive monitoring approach for cloud computing systems. It characterizes the running status of systems with Principal Component Analysis (PCA), estimates the anomaly degree, and predicts the possibility of faults. Based on that, it dynamically adjusts the monitoring period.

5.2 Self-healing approach in PIACERE

The goal of Self-healing is to analyse notifications coming from the monitoring and self-learning components and propose corrective actions when they are necessary. Therefore, the approach to self-healing in PIACERE will leverage three elements, see Figure 10: notification messages, notifications types and response strategies.

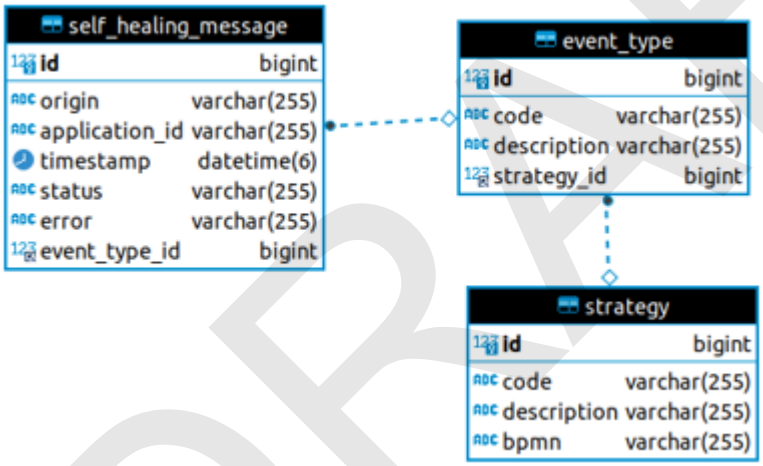

Figure 10. Self-healing elements

Self-healing messages will be sent by monitoring and self-learning components. To do so self-healing will provide a REST API, with the capability to create self_healing_messages. The self_healing_messages will include as mandatory field the event type.

Then we will create an infrastructure to define the allowed event types and the strategy to respond to each of those events. The most obvious strategy will be to redeploy the applications, but other strategies are also under evaluation such as the possibility to reboot the infrastructure, to scale, etc.

We are still in the process to evaluate the approach in the implementation of the strategies. We have the general approach in which these strategies are going to be BPMN workflows to be executed by PRC. But currently we need to do some more research and internal discussions with the PRC development team to decide if these workflows will be dynamically added or will be preconfigured in PRC.

The overall self-healing process will be to receive notifications, queue the notifications to be executed, and to proceed with the execution of the self-healing strategies in each deployment in order.

As the strategies could take some time to be executed we will also need to decide on approaches to review the notifications in the queue, because it may happen that we receive multiple notifications that could be solved with just one corrective strategy.

6 Monitoring Controller Implementation

6.1 Functional description

The PIACERE monitoring, self-learning and self-healing architecture involves several distributed and potentially scalable components that require continuous configuration as PIACERE platform creates, updates and destroys application deployments. Besides, the architecture currently covers two main non-functional aspects (performance, and security), each one involving different sets of components and technologies.

The Figure 11 describes the sequence diagram for the two main functionalities to be covered by the Monitoring Controller, this is, start and stop deployments.

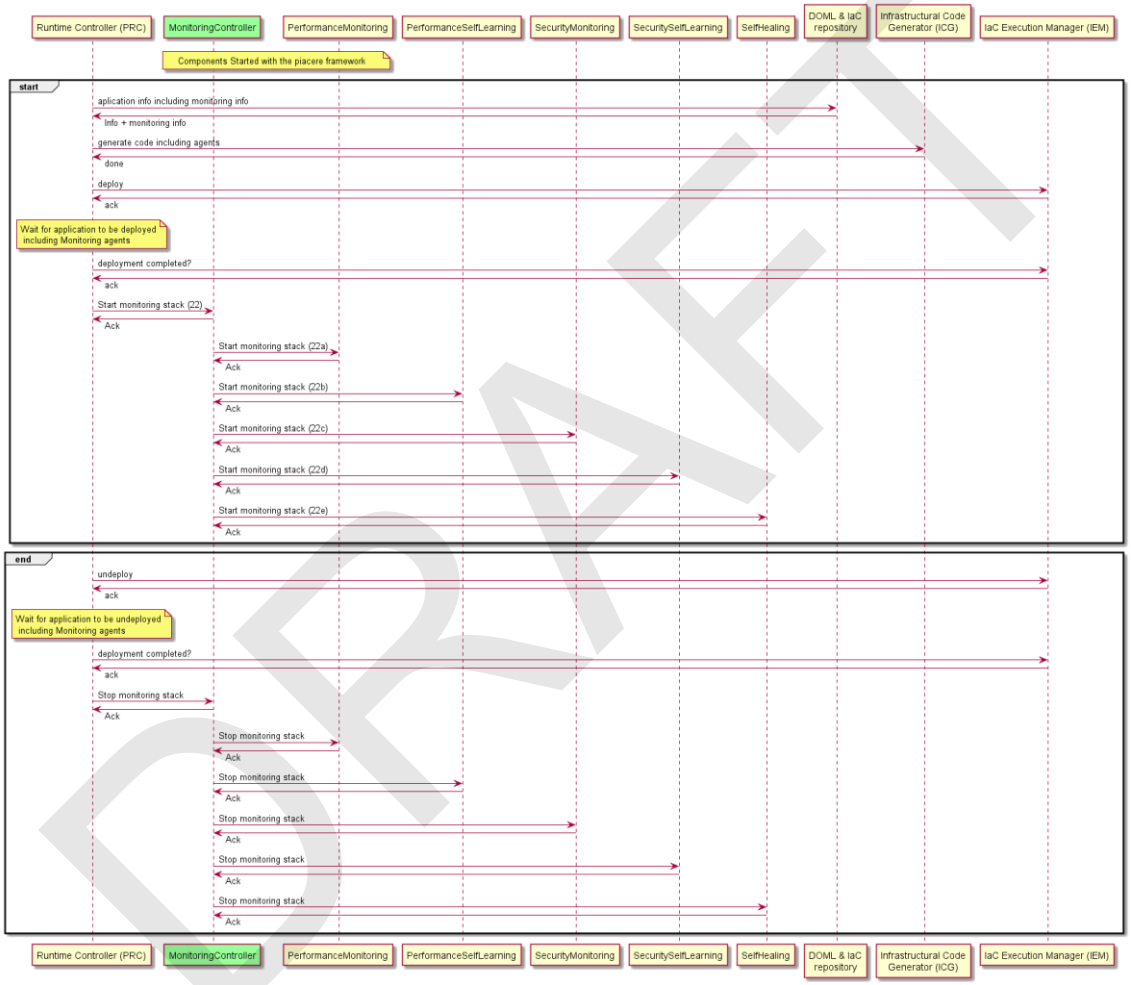

Figure 11. Monitoring Controller Sequence diagram

On one hand, the monitoring controller aims to simplify the configuration of the PIACERE monitoring, self-learning and self-healing component as PIACERE platform creates, updates and destroys application deployments. On the other hand, the monitoring controller aims to isolate the PIACERE runtime controller from changes in the PIACERE monitoring, self-learning and self-healing architecture.

The first implementation of the monitoring controller aims to provide a REST API for future use by the PIACERE Runtime Controller.

This is a proxy component that aims to provide a single point of entry to the configuration of the PIACERE monitoring, self-learning and self-healing components as PIACERE platform creates, updates and destroy application deployments, as such it does not provide innovative advances to the state of the art.

6.2 Requirements covered by this prototype

The user requirements from WP2 satisfied by this interim version are described in the Table 2. All these requirements are being polished and adapted as the project advances and we gain knowledge on the use cases and on the implemented components.

Table 2. Monitoring Controller related user requirements from WP2

Req ID	Description	Implementation Status	Requirement Coverage
REQ17	Seamless security monitoring deployment *Deployment of runtime security monitoring should happen seamlessly or with minimal effort and configuration required by the user.*	completed	A REST API has been provided and deployed to be a single point of entry for the configuration of the PIACERE monitoring, self-learning and self-healing components each time that a deployment is requested to the PIACERE runtime controller.
REQ50	Monitor performance, availability, and security *The monitoring component shall monitor the metrics associated with the defined measurable NFRs (e.g. performance, availability, and security through the runtime security monitoring).*	In progress	The REST API supports the transmission of all the necessary information for the configuration of the deployment, namely nonfunctional requirements regarding performance, availability and security. The current version of the component includes the function to call the remaining components.
REQ51	Deployment nonfunctional requirements tracking *The self-learning component shall ensure that the conditions are met (compliance with respect to SLO) and that a failure or a non-compliance of a NFRs is not likely to occur. This implies the compliance of a predefined set of non-functional requirements (e.g. performance).*	In progress	The component will forward all the necessary information to the self-learning components to be able to track the infrastructure related nonfunctional requirements.

The internal requirements satisfied by this interim version are described in the Table 3. All these requirements are as well polished and adapted as the project advances.

Table 3. Monitoring Controller related internal requirements

Title	Implementation Status	Requirement Coverage
Add code into the project source repository	Completed	The repository has been created and the code is being uploaded regularly https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/mc/mc.git
Implement REST API specification	In progress	A first version of the OpenAPI has been defined and put under configuration control https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/mc/mc/-/blob/main/mc/openapi/openapi.yaml The content of the deployment creation message requires still some discussion with other components to understand the information that can be provided and its format.
Implement specification first approach	Completed	In order to speed-up the implementation of changes derived from the expected evolution of the REST API, we have implemented a specification first approach with openapi generator. Besides, the usage of openapi generator bring additional benefits in the sense of introduction of good practices in structuring and configuring the code.
Prepare for deployment	Completed	In order to ensure that we are prepared to deploy the component in the integration environment we have integrated this component with the remaining monitoring components in a docker-compose file that includes a reverse proxy to receive all the requests using secure standard HTTPS protocol. https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-deploy
Provide fast deployment alternative for deployment, testing and evaluation	Completed	To allow a seamless infrastructure requirements free alternative to test this component we have provided a Vagrant based deployment option. This reduces the list of software requirements to two: VirtualBox[13] and Vagrant[14]. These two tools (VirtualBox and Vagrant) are available for most of the operating systems: Windows, Mac, Linux, BSD, …
Include usage documentation	Completed	We have included usage documentation at different levels: • Vagrant https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-vagrant • Docker-compose https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-deploy • Python code https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/mc/mc/-/blob/main/README.md We update continuously the documentation as we advance in the coding and pre-integration of the monitoring components

[13] https://www.virtualbox.org/
[14] https://www.vagrantup.com/

Title	Implementation Status	Requirement Coverage
Unitary test	In progress	We have included a testing framework to the code based on Tox[15]. Concrete tests are still to be developed as the component gets more mature.
Continuous integration	In progress	Continuous integration has been implemented based on Gitlab-ci [16]https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-deploy/-/blob/develop/.gitlab-ci.yml The integration approach requires to be migrated with the rest of the components of PIACERE framework.

6.3 Fitting into overall PIACERE Architecture

The Monitoring Controller is one of the components of the PIACERE architecture. It is part of the Infrastructure Advisor package in the runtime phase of PIACERE (Figure 12). The Monitoring Controller interacts with other components in the PIACERE ecosystem:

- PIACERE Runtime Controller (PRC) requests to the Monitoring Controller to start and stop the monitoring of the concrete deployments.
- The Monitoring Controller forwards the start and stop deployment monitoring requests to the PIACERE monitoring, self-learning and self-healing components. Specifically, to:
  - Performance Monitoring
  - Security Monitoring
  - Performance Self-learning
  - Security Self-learning
  - Self-healing

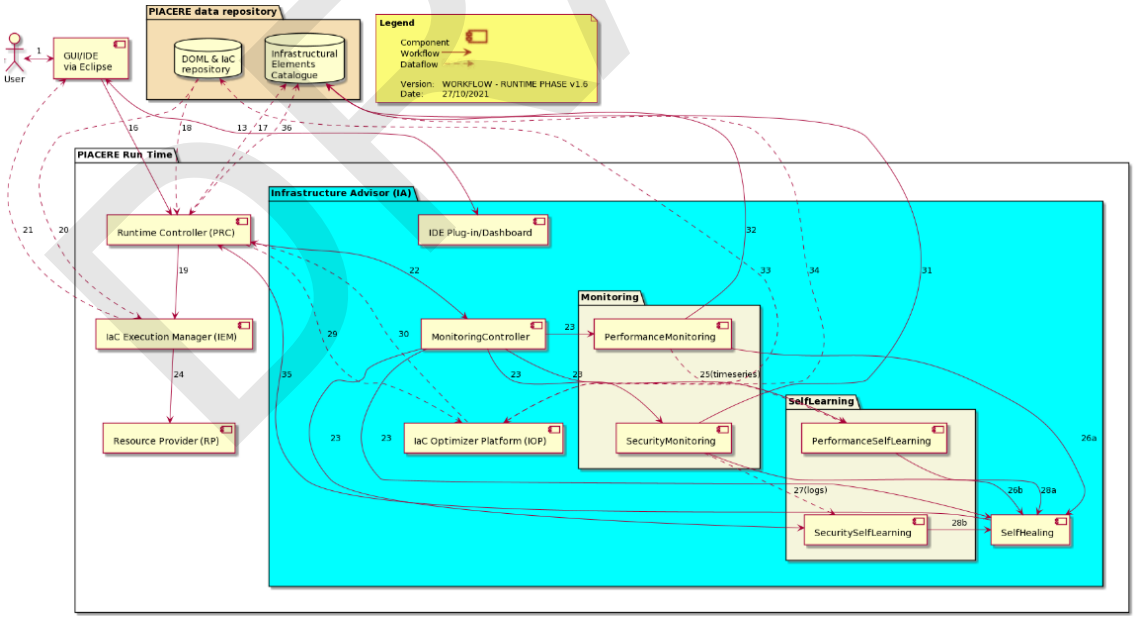

Figure 12. PIACERE Runtime Diagram on its 1.6 version

[15] https://tox.wiki/en/latest/index.html
[16] https://docs.gitlab.com/ee/ci/

6.4 Technical description

This subsection is devoted to describing the technical specification of this first prototype. First, the main architecture of the prototype and the components are shown and described in Section 6.4.1. *Prototype Architecture*. This subsection finishes with the technical specifications of the developed system in *Section 6.4.2 Technical Specifications*.

6.4.1 Prototype architecture and components description

The main architecture of this first prototype is depicted in the following Figure 13. In this architecture, seven different components can be distinguished: Connexion, Monitoring Controller, and five clients to communicate with the rest of the components of PIACERE monitoring, self-learning and self-healing components.

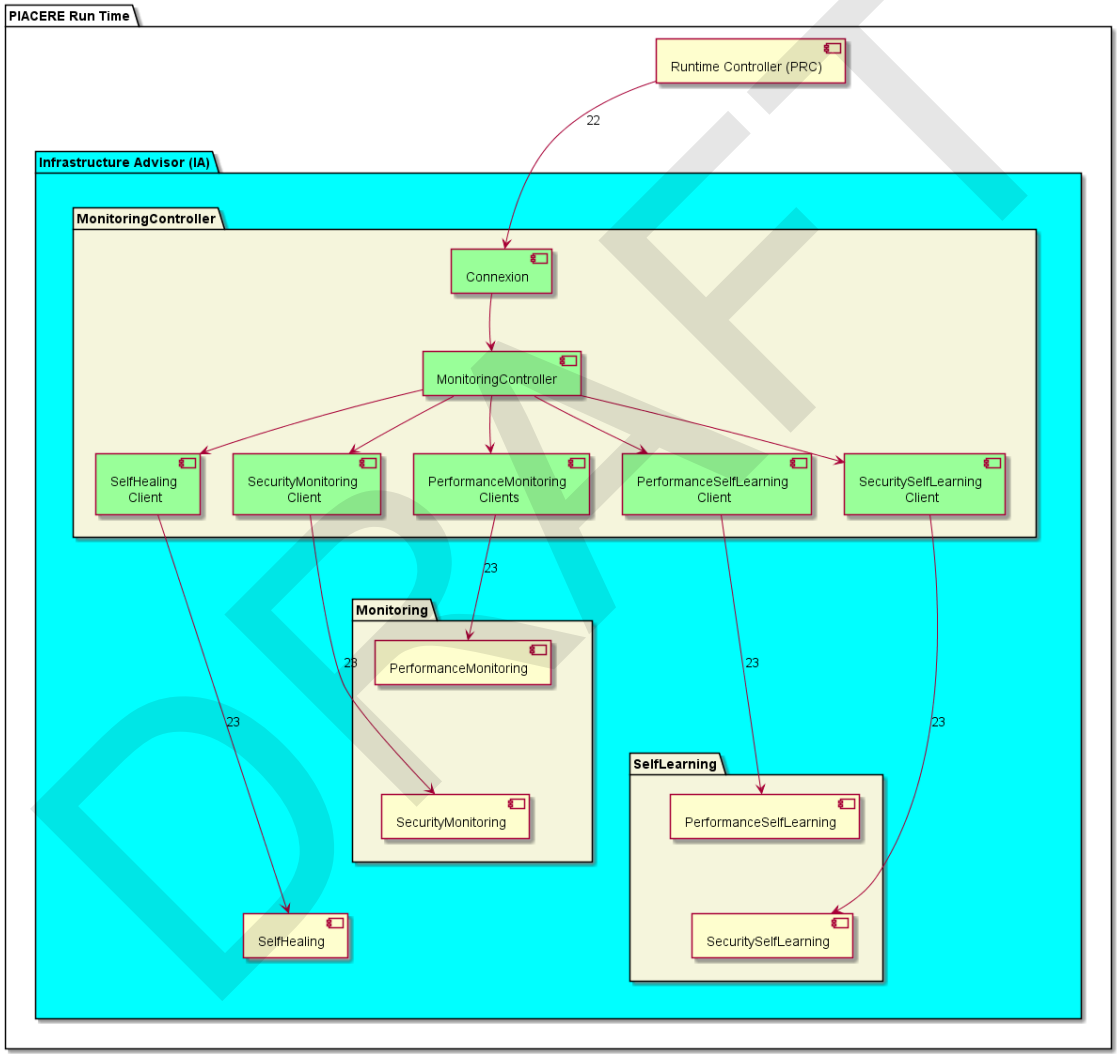

Figure 13. Monitoring Controller first prototype architecture

This first prototype of the Monitoring Controller is composed of two components and five more are planned for the next iteration with the remaining PIACERE monitoring, self-learning and self-healing components.

- Connexion: This is an open source component from Zalando https://github.com/zalando/connexion that enables the specification first approach in python.

- Monitoring Controller: This is the main component where the forwarding and the configuration are managed.

Planned components for the next iteration are:

- Performance Monitoring Client: This will be an autogenerated component from the OpenAPI of the performance monitoring with the openapi generator https://github.com/OpenAPITools/openapi-generator.
- Security Monitoring Client: This will be also an autogenerated component with the openapi generator.
- Performance Self-learning Client: This will be also an autogenerated component with the openapi generator.
- Security Self-learning Client: This will be also an autogenerated component with the openapi generator.
- Self-healing Client: This will be also an autogenerated component with the openapi generator.

6.4.2 Technical specifications

This prototype has been developed using Python, which is an interpreted class-based, high level, object-oriented and general-purpose programming language. We have chosen Python as it is easier to read, learn and write and is ideal for the fast implementation of low complexity code as the one we have to do in this component.

The component is packaged using Docker technology to simplify the Python requirements and environment management. This is also a requirement for the future integration of PIACERE components into the PIACERE framework.

7 Performance Monitoring Implementation

7.1 Functional description

The PIACERE Performance Monitoring component gathers performance and availability related information from the infrastructure resources that form part of each of the deployments managed by PIACERE. It also stores that information over the time so that it can be accessed by other components to perform more complex analysis, such as the performance self-learning component. Besides, it also monitors some metrics with respect to some thresholds in order to issue notifications to other components, in case those thresholds are exceeded. Finally, it aggregates metrics based on actual measurements and updates the characteristics of the services listed in the Infrastructure Element Catalogue.

The Figure 14 describes the sequence diagram for the three main activities that the Performance Monitoring should support apart from the user interface access and the data retrieval that are not currently included in this release, as we are going to provide them using open source components already available.

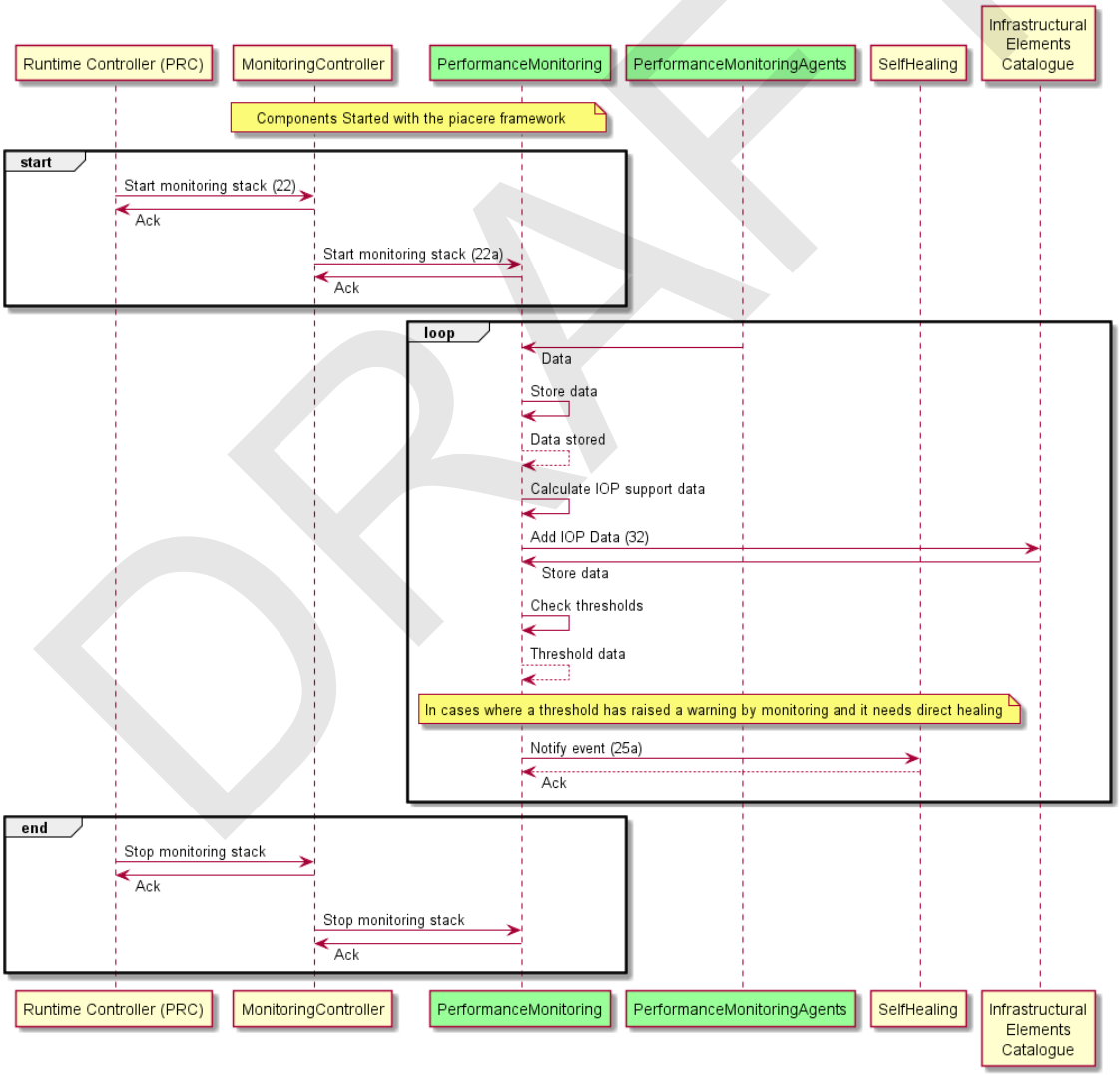

Figure 14. Performance monitoring Sequence diagram

The first implementation of the Performance Monitoring aims to:

- Identify the set of information to be retrieved from the infrastructure resources
- Start gathering and storing information from real resources, so that it is available for other components
- Expose the retrieval API so that the consuming components can start the data integration as soon as possible
- Provide the interface for the configuration of the Performance Monitoring for each of the deployments
- Expose the graphical user interface integrated with the persistence layer, for future customization based on the deployment information

7.2 Requirements covered by this prototype

The user requirements from WP2 satisfied by this interim version are described in the Table 4. All these requirements are being polished and adapted as the project advances and we gain knowledge on the use cases and on the implemented components.

Table 4. Performance Monitoring related user requirements from WP2

Req ID	Description	Implementation Status	Requirement Coverage
REQ46	IOP focused infrastructure metrics *The monitoring component shall gather metrics from the instances of the infrastructural elements at run time. These metrics need to be related to the NFR and accessible to the IOP (through the dynamic part of the infrastructural catalogue).*	planned	The feature is planned but not implemented in this M12 prototype. The feature is planned to be implemented during the next period.
REQ47	Full monitoring stack *The monitoring component shall include the needed elements in the stack to monitor the infrastructural elements.*	Completed	The Performance Monitoring includes all the elements required to monitor infrastructure elements: The agents to gather the information, the database to store the data, the analysis and presentation layer to show the metrics and follow the thresholds, and the component to configure the deployments. The elements are present, but they still require some development that should be completed in the following months: • Automate the deployment of the agents within the ICG (infrastructure code generator) and the IEM (IaC Execution Manager). • Complete missing features regarding threshold configuration and IEC (Infrastructure Element Catalogue) feed.
REQ48	Self-learning focused monitoring *The monitoring component shall transform the real time values into the correct*	Completed	Real time data is stored and the performance self-learning prototype is actually capable of consuming that information using the provided interface.

Req ID	Description	Implementation Status	Requirement Coverage
	format/type/nature for the self learning component.		
REQ50	Monitor performance, availability, and security *The monitoring component shall monitor the metrics associated with the defined measurable NFRs (e.g. performance, availability, and security through the runtime security monitoring).*	In progress	The Performance Monitor currently gathers information from infrastructure resource that can be used to compute the performance and availability metrics. In the following months, we need to associate with the defined measurable NFRs from the DOML. Besides, we also need to implement graphical user interfaces that show those measures and thresholds in real time.
REQ51	Deployment nonfunctional requirements tracking *The self-learning component shall ensure that the conditions are met (compliance with respect to SLO) and that a failure or a non-compliance of a NFRs is not likely to occur. This implies the compliance of a predefined set of non-functional requirements (e.g. performance).*	Planned	Currently we are gathering the information from the infrastructure resources, but we did not address yet the implementation of the thresholds based on the DOML information. This is planned for the next period.
REQ52	Monitored data based self-learning *Self-healing shall consume the data monitored and store it in a time-series database to create discriminative complex statistical variables and train a predictor, which will learn potential failure patterns in order to prevent the system from falling into an NFR violation situation.*	In-progress	The Performance Monitoring currently provides the time series database for the usage by the performance self learning component. This covers a part of this requirement, the other part is covered by the performance self learning component.
REQ72	monitoring user interface *The runtime monitoring component should provide an UI for the end users to see the monitored resources and the corresponding metrics/NFRs in real time.*	In progress	The current version of the Performance Monitoring includes a graphical user interface that renders the information coming from the time series database. In the future months, we will introduce a deployment-based dashboard that will include information related to the NFR thresholds coming from the DOML specification.
REQ93	Self-healing should classify the events notified	Planned	*Self-healing component shall classify the events received from the self learning and derive corrective actions.* It is planned to send notifications to the self-healing component, including the notification type information that should support the classification of the event.

The internal requirements satisfied by this interim version are described in the Table 5. All these requirements are as well polished and adapted as the project advances.

Table 5. Performance Monitoring related internal requirements

Title	Implementation Status	Requirement Coverage
Add code into the project source repository	Completed	The repository has been created and the code is being uploaded regularly https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-deploy
Implement REST API specification	In progress	A first version of the OpenAPI has been defined and put under configuration control https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pmc/-/blob/main/openapi.yaml The content of the deployment creation message requires still some discussion with other components to understand the information that can be provided and its format.
Implement specification first approach	Completed	In order to speed-up the implementation of changes derived from the expected evolution of the REST API, we have implemented a specification first approach with openapi generator. Besides, the usage of openapi generator brings additional benefits in the sense of introduction of good practices in structuring and configuring the code.
Prepare for deployment	Completed	In order to ensure that we are prepared to deploy the component in the integration environment we have integrated this component with the remaining monitoring components in a Docker-compose file that includes a reverse proxy to receive all the requests using secure standard HTTPS protocol. https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-deploy
Provide fast deployment alternative for deployment, testing and evaluation	Completed	To allow a seamless infrastructure requirements free alternative to test this component we have provide a Vagrant based deployment option. This reduces the list of software requirements to two: VirtualBox and Vagrant. These two tools (VirtualBox and Vagrant) are available for most of the operating systems: Windows, Mac, Linux, BSD, …
Include usage documentation	Completed	We have included usage documentation at different levels: • Vagrant https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-vagrant • Docker-compose https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-deploy We update continuously the documentation as we advance in the coding and pre-integration of the monitoring components
Unitary test	Planned	We plan to include a testing framework to the code JUnit in the performance monitoring controller.
Integration test	Planned	We plan to include a integration test based on Newman at docker-compose level.

Title	Implementation Status	Requirement Coverage
Continuous integration	In progress	Continuous integration has been implemented based on gitlab-ci https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-deploy/-/blob/develop/.gitlab-ci.yml The integration approach requires to be migrated with the rest of the components of PIACERE framework.

7.3 Fitting into overall PIACERE Architecture

The Performance Monitoring is one of the components of the PIACERE architecture. It is part of the Infrastructure Advisor package in the runtime phase of PIACERE ((Figure 12). The Monitoring Controller interacts with other components in the PIACERE ecosystem:

- The Monitoring Controller request to start and stop the monitoring of the concrete deployments to the Performance monitoring as well as other components.
- The Infrastructure element catalogue receive from the Performance monitoring information about the monitored infrastructure resources.
- The performance Self-learning will use the stored data by the performance monitoring to forecast events in the infrastructure resources.
- The Self-healing receive notifications from the performance monitoring about non-functional thresholds violations.

7.4 Technical description

This subsection is devoted to describing the technical specification of this first prototype. First, the main architecture of the prototype and the components are shown and described in Section 7.4.1. This subsection finishes with the technical specifications of the developed system in *Section 7.4.2 Technical Specifications*.

7.4.1 Prototype architecture and components description

The main architecture of this first prototype is depicted in the following Figure 13. In this architecture, four different components can be distinguished (highlighted in green): Performance Monitoring Controller, Influxdb, Grafana and Performance Monitoring Agents. The main purpose of these components is described.

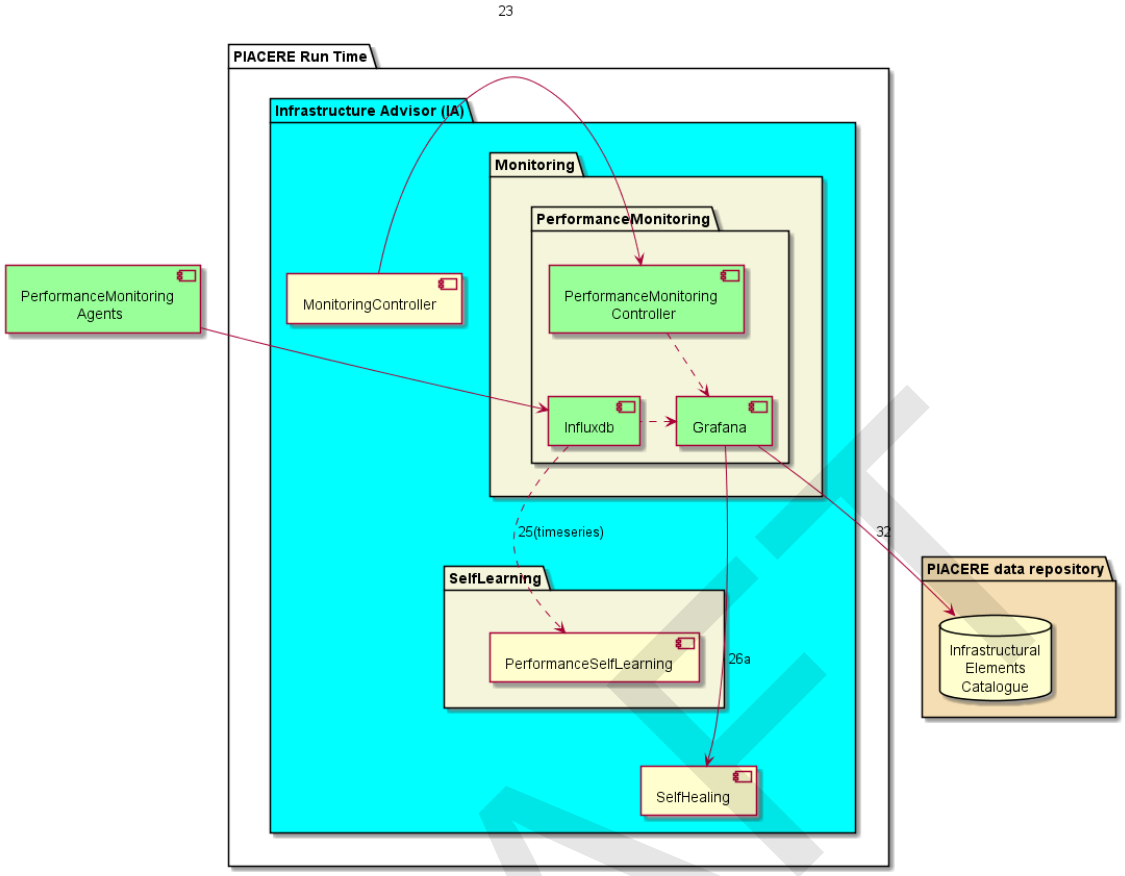

Figure 15. Performance Monitoring first prototype architecture

This first prototype of the Performance Monitoring is composed by four components, three of them will run together with the PIACERE runtime framework and the other one will run in the deployed infrastructures. The components in the PIACERE runtime framework are:

- Performance Monitoring Controller: This is the main component that receives the start and stop requests by the Monitoring controller and configures Grafana in consequence.
- Influxdb: is a time series database that will receive the information from all the Performance Monitoring agents throughout all the active deployments: This is an open source component[17] that enables the storage of time series.
- Grafana is a time series rendering web interface that includes functionalities to keep track of thresholds and sends notifications when the thresholds are exceeded. This is an open source component[18]

The component running on the deployed infrastructures is the Performance Monitoring agent. The monitoring agent gathers multiple parameters from the runtime infrastructures that run the components of the deployed applications. The Performance Monitoring agent is implemented using an open source component[19]

[17] https://www.influxdata.com/
[18] https://grafana.com/
[19] https://www.influxdata.com/time-series-platform/telegraf/

7.4.2 Technical specifications

The Performance Monitoring Controller prototype has been developed using Java, more specifically the Java Spring Boot framework[20] that is an open source, enterprise-level framework for creating standalone, production-grade applications. We have created the application using the openapi generator technology, that starting from the OpenAPI specification is capable to generate a Spring Boot architecture to implement that functionality.

The Grafana extension will be evaluated in the following month to feed the Infrastructure Element Catalogue and the self-healing components. The front-end part of Grafana is extended using plug-ins developed with flot[21], but possibly our extensions will be more in the backend side which is developed in go.

[20] https://spring.io/projects/spring-boot

[21] https://www.flotcharts.org/

8 Security Monitoring Implementation

8.1 Functional description

The Security monitoring system consists of subsystems (Wazuh deployment – manager and agents - with specific components for data transformation) collecting data in order to provide values for security metrics. As an additional option, it can provide the deployment of Vulnerability Assessment Tool (VAT) that is capable of monitoring API end-points of the specific Web Application. The system stores the (1) data aggregated by the (security) monitoring system and (2) data generated by underlying anomaly detection system using dedicated ELK stack. Finally, it aggregates metrics based on actual measurements and updates the characteristics of the services listed in the Infrastructure Element Catalogue.

The Figure 16 describes the sequence diagram of the Security Monitoring and Security Self-learning processes.

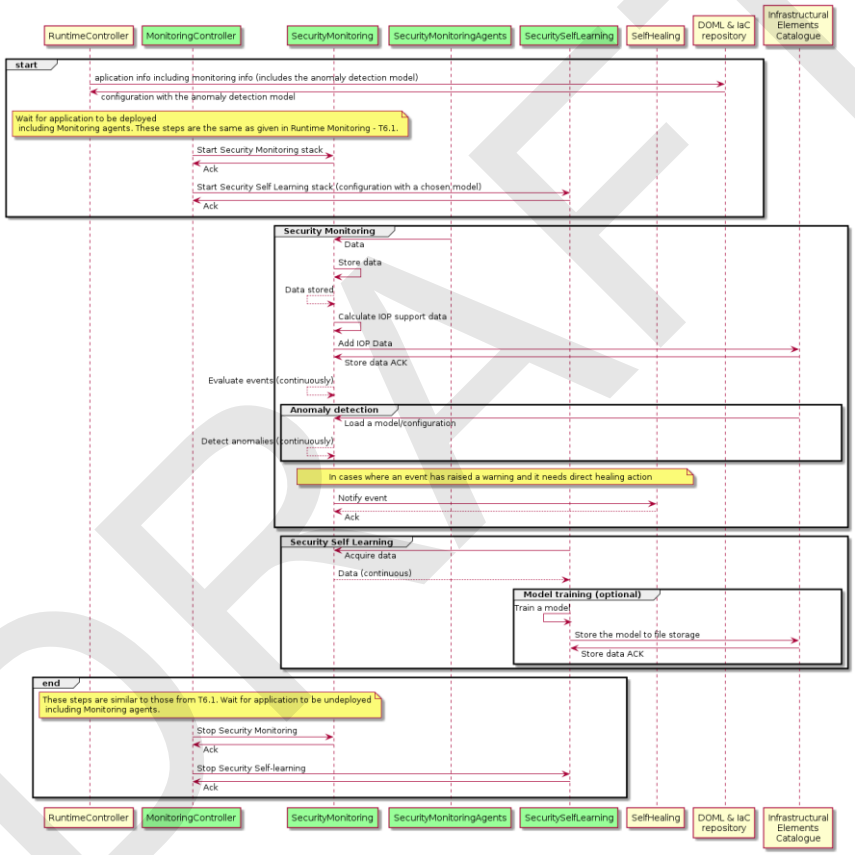

Figure 16. Security Monitoring Sequence diagram.

The first implementation of the Security Monitoring:

- Provides the Controller, exposing Security Monitoring API and orchestrating underlying services (Figure 17): Monitoring Manager and Model Trainer.
- Provides basic infrastructure based on ELK (part of Wazuh deployment) in order to aggregate relevant security metrics and provide data feed to anomaly detection components.
- Exposes integrated graphical user interface

8.2 Requirements covered by this prototype

The user requirements from WP2 satisfied by this interim version are described in Table 6.

Table 6. Security Monitoring and Security Self-learning Requirements related user requirements from WP2.

Req ID	Description	Implementation Status	Requirement Coverage
REQ14	Runtime security monitoring must provide monitoring data from the infrastructure's hosts w.r.t. security metrics	In-progress	Security Monitoring Controller provides API call in order to get the alerts/events from the stored database.
REQ15	Runtime security monitoring could provide monitoring data from the application layer (infrastructure's guest) w.r.t. security metrics	In-progress	This is planned for Y2. Anyhow, this is possible through the configuration of the Security Monitoring Manager (specifically, Wazuh configuration).
REQ16	Runtime security monitoring should contribute to mitigation actions taken when considering plans and strategies for runtime self-healing actions	In-progress	This is planned for Y2. Currently we are having conversations within WP6 how to tackle the integration with the self-healing components.
REQ17	Deployment of runtime security monitoring should happen seamlessly or with minimal effort and configuration required by the user.	In-progress	The deployment code is partially already available on the project's repository.
REQ18	Runtime security monitoring must be able to detect different types of metrics in run-time: integrity of IaC configuration, potential attacks to the infrastructure, IaC security issues (known CVEs of the environment).	In-progress	The data of these metrics are already available in the Security Monitoring infrastructure. However, this is possible through the configuration of the Security Monitoring Manager (specifically, Wazuh configuration). The configuration needs to be provided through the configuration step.
REQ19	Runtime security monitoring and alarm system (self-learning) integration must be implemented.	In-progress	This is planned for Y2. The integration of the self-healing component is foreseen.
REQ21	Runtime security monitoring and Runtime monitoring infrastructure should be integrated with minimal extensions.	In-progress	This is planned for Y2. The integration will be done through the deployment of the Security Monitoring Agents and their deployment code.
REQ50	The monitoring component shall monitor the metrics associated with the defined measurable NFRs (e.g. performance, availability, and security through the runtime security monitoring)	In-progress	This is still not implemented since there is no integration between "expressing NFRs" and configuration of the security monitoring infrastructure.
REQ51	The self-learning component shall ensure that the conditions are met (compliance with respect to SLO) and that a failure or a non-compliance of a NFRs is not likely to occur. This implies the compliance	In-progress	The self-learning component of security monitoring will build appropriate model to be used for detecting metrics with respect to anomaly detection (anomalies detected on the infrastructure). It will be possible to express these metrics and

Req ID	Description	Implementation Status	Requirement Coverage
	of a predefined set of non-functional requirements (e.g. performance)		related NFRs through the integration with DOML components and the rest of PIACERE flow (NFRs to be consumed by Security Monitoring).

The internal requirements satisfied by this interim version are described in the Table 7. All these requirements are as well polished and adapted as the project advances.

Table 7. Security Monitoring related internal requirements

Title	Implementation Status	Requirement Coverage
Add code into the project source repository	Completed	The repository has been created and the code is being uploaded regularly https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-controller and https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-deployment
Implement REST API specification	In progress	A first version of the OpenAPI has been defined and put under configuration control
Implement specification first approach	Completed	In order to speed-up the implementation of changes derived from the expected evolution of the REST API, we have implemented a specification first approach with OpenAPI generator.
Prepare for deployment	Completed	Part of the code provided on the Gitlab.
Provide fast deployment alternative for deployment, testing and evaluation	Completed	Part of the code provided on the Gitlab.
Include usage documentation	Completed	Part of the code provided on the Gitlab.
Unitary test	Planned	Not yet available
Integration test	Planned	Not yet available
Continuous integration	Planned	Not yet available

8.3 Fitting into overall PIACERE Architecture

The Security Monitoring is one of the components of the PIACERE architecture. It is part of the Infrastructure Advisor package in the runtime phase of PIACERE (Figure 12). The Monitoring Controller interacts with Security Monitoring:

- The Monitoring Controller requests to start and stop the monitoring of the concrete deployments.

- The Infrastructure Element Catalogue receives from the Security Monitoring information about the monitored infrastructure resources.
- The Security Self-learning will use the stored data by the security monitoring to build models used by the anomaly detection process in the infrastructure resources.
- The Self-healing receives notifications from the security monitoring about non-functional thresholds violations.

8.4 Technical description

8.4.1 Prototype architecture and components description

Figure 17 depicts architecture of the Security Monitoring and Security Self-learning components.

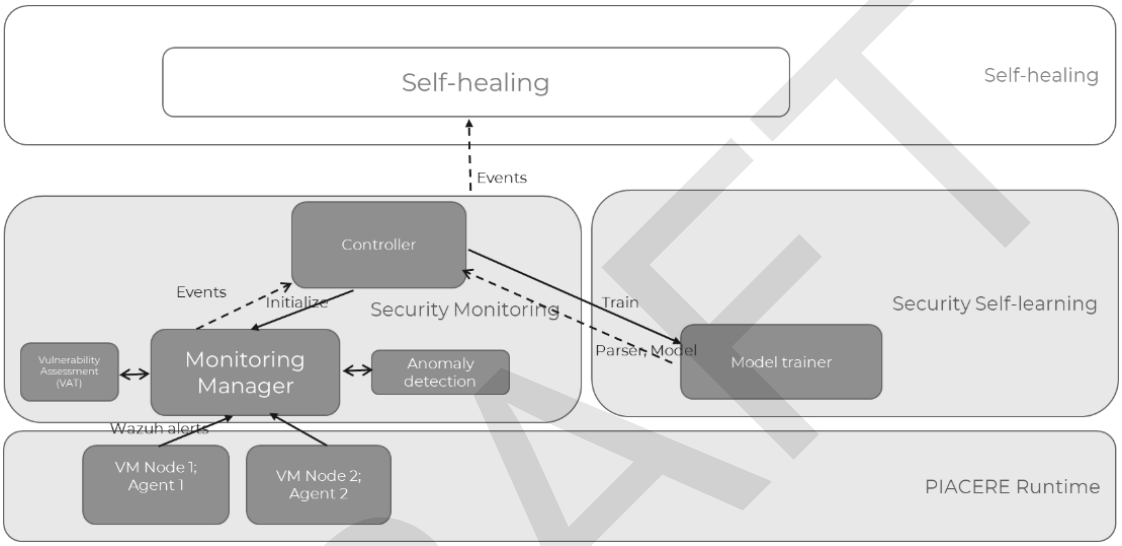

Figure 17. Architecture of Security Monitoring and Security Self-learning.

Controller exposes underlying APIs of the Monitoring Manager and Model Trainer via RESTful (OpenAPI spec) API. Anomaly Detection component uses the data feed provided by the Monitoring Manager in order to detect anomalies based on the pre-built anomaly detection model built by the Model Trainer. Monitoring Manager's Agents residing on the underlying infrastructure provide continuous data feed into the Monitoring Manager's data storage. There are possibilities to extend Monitoring Manager's Agents with other modules such as Vulnerability Assessment Tool (VAT), in order to provide different security-based metrics into the data feed, to be considered in the evaluation process.

8.4.2 Technical specifications

The Security Monitoring components are developed mainly using Python and Ansible deployment scripts. OpenAPI specification has been developed for the Controller's API (publicly accessible at https://app.swaggerhub.com/apis/alescernivec/security-monitoring/1.0.0).

The Controller uses Flask framework and its underlying Authentication/Authorization mechanisms. Through the API it provides, it exposes alerts where additional search queries are possible.

Monitoring Manager is developed using deployment of Wazuh 4.1 which already provides agents and ELK stack used for storing a plethora of different security metrics. It aggregates and stores alerts stemming from the Agents deployed on the infrastructure. Filebeat deployment is

part of these agents. Data stemming from ELK (specifically from Filebeat[22]) is being consumed by the anomaly detection mechanism within the Security Monitoring architecture. Monitoring Manager provides Kibana dashboard so that the user can review all the alerts provided by the Security Monitoring Manager.

VAT is the tool developed by XLAB. Its deployment is optional at this point. The planned use of the tool is to provide additional security metrics that could be expressed through NFRs.

[22] https://www.elastic.co/beats/filebeat

9 Performance Self-learning Implementation

9.1 Functional description

After being started by the Performance Monitoring component, the Performance Self Learning components follows an iterative process. It acquires data from the Performance Monitoring database and analyses it. The analysis consists of a concept drift algorithm and an anomaly detection algorithm, both operating at the same time. The online prediction process may send a notification to the Self-Healing when the CPU idle exceeds a threshold.

The following figure describes the sequence diagram for this iterative process.

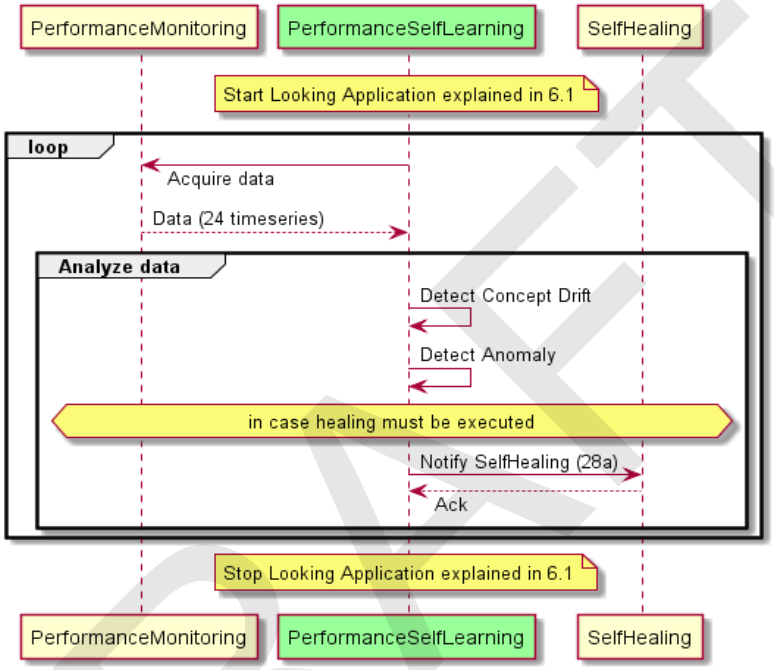

Figure 18. Self-learning sequence diagram.

9.2 Requirements covered by this prototype

The user requirements from WP2 satisfied by this interim version are described in the Table 8.

Table 8. Performance Self-learning Requirements related user requirements from WP2.

Req ID	Description	Implementation Status	Requirement Coverage
REQ11	The learning algorithm should be executed as fast as possible as it must provide an outcome before more data arrives. The anomaly detection should have fast and easy access to the monitoring data.	Completed	The whole PerformanceSelfLearning component has direct access to the data in the PerformanceMonitoring database. At the moment execution is fast but further analysis is planned for Y2/Y3.
REQ51	The self-learning shall ensure that the conditions are met (compliance with respect to SLO) and that a failure or a non-compliance of a NFRs is not likely to occur. This implies the compliance of a	Completed	The PerformanceSelfLearning component is able to run fast by filtering the data to be acquired.

Req ID	Description	Implementation Status	Requirement Coverage
	predefined set of non-functional requirements (e.g. performance)		
REQ52	Self learning shall consume the data monitored and stored in a time-series database to create discriminative complex statistical variables and train a predictor which will learn potential failure patterns in order to prevent the system from falling into an NFR violation situation	Completed	Thanks to the integration of the PerformanceMonitoring database access implementation, the PerformanceSelfLearning component is able to consume the data in an incremental way and to create the necessary variables.

The internal requirements satisfied by this interim version are described in the Table 9. All these requirements are as well polished and adapted as the project advances.

Table 9. Performance Self Learning related internal requirements

Title	Implementation Status	Requirement Coverage
Add code into the project source repository	Satisfied	The repository has been created and the code is being uploaded regularly: https://git.code.tecnalia.com/piacere/private/t62-self-learning/psl
Implement REST API specification	In progress	A first version of the OpenAPI has been defined and put under configuration control: https://git.code.tecnalia.com/piacere/private/t62-self-learning/psl/-/blob/5e234cff5aecd52537ee27f75f6f17e5f76fe481/docs/self-learning-openapi.yaml
Implement specification first approach	Satisfied	In order to speed-up the implementation of changes derived from the expected evolution of the REST API, we have implemented a specification first approach with OpenAPI generator.
Prepare for deployment	Satisfied	Part of the code provided on the Gitlab.
Provide fast deployment alternative for deployment, testing and evaluation	Satisfied	Following the Dependency specification for Python Software Packages the required file is provided: https://git.code.tecnalia.com/piacere/private/t62-self-learning/psl/-/blob/5e234cff5aecd52537ee27f75f6f17e5f76fe481/requirements.txt
Include usage documentation	Planned	Not yet available.
Unitary test	Planned	Not yet available.
Integration test	Planned	Not yet available.
Continuous integration	Planned	Not yet available.

9.3 Fitting into overall PIACERE Architecture

The Performance Self Learning is one of the components of the PIACERE architecture. It is part of the Infrastructure Advisor package in the runtime phase of PIACERE (Figure 12). The Monitoring Controller interacts with the Performance Self Learning:

- The Monitoring Controller requests to start and stop the Performance Self Learning of the concrete deployments.
- The Performance Self Learning will create different models and feed them with data from the Performance Monitoring Controlled to be able to detect Concept Drift phenomenon and detect anomalies.
- The Self-healing receives notifications from the Performance Self Learning component about warnings in the deployments behaviour.

9.4 Technical description

9.4.1 Prototype architecture and components description

The Performance Self-learning component is composed by different solutions and approaches to deal with its goal. In order to achieve its main objective, the Performance Self-learning is composed by different subcomponents portrayed in the following Figure 19.

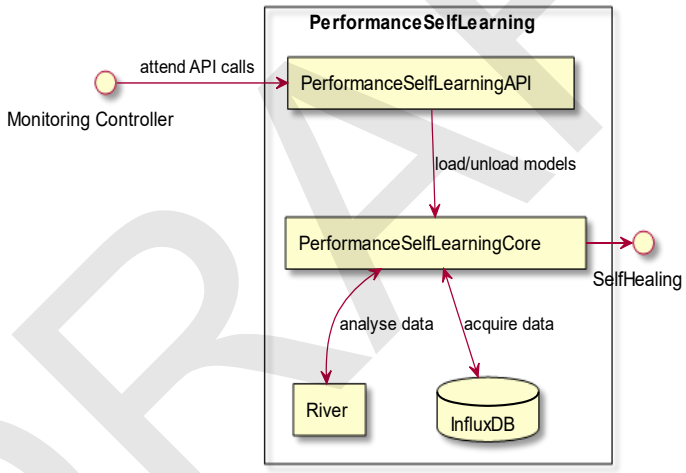

Figure 19. Architecture of the Self-learning component.

In this architecture, two main different components can be distinguished: PerformanceSelf LearningCore and PerformanceSelfLearningAPI component. The PerformanceSelfLearningCore component is also composed by two components: River and InfluxDB.

The prototype is split in two main components.

- PerformanceSelfLearningCore: This component will be in charge of loading or creating Concept Drift and Anomaly Detection learning models whenever a deployment loading has been notified or stopping the learning process on any unloading. This component will also be in charge of feeding the models with data. Finally, once data is analysed, SelfHealing component will be notified in case of exceeding the idle threshold for the "usage_idle" variable.
- PerformanceSelfLearningAPI: This component will be on charge on notifying the PerformanceSelfLearningCore component of the loading or unloading of any deployment.

The PerformanceSelfLearningCore is also split in two components:

- River: The library that implements the Concept Drift and Anomaly Detection algorithm
- InfluxDB: The time series database from which the PerformanceSelfLearningCore will receive the data to feed the models.

9.4.2 Technical specifications

River is a library for developing online machine learning solutions in Python. It was created by the combination of two of the most popular stream learning packages: scikit-multiflow and creme. Its main innovation is the use of pipelines to transform data in the process of data digestion. It also provides different learning models out of the box, specialized in jobs such as anomaly detection, classification, clustering, regression, etc. The library also offers the Half Space Trees (anomaly detection), Random Forest Regressor (incremental learning) and ADWIN (drift detection) used in the component. River has been the basis for the development of the incremental learning and anomaly detection, and it will also be the basis for the drift detection. After using River with the toy dataset, we have successfully confirmed that it is the suitable library to develop the Self-learning component in the PIACERE project.

For data provision, the official InfluxDB Python library is used, due to the use of InfluxDB as the data storage. Its use is seamlessly integrated in the current implementation, allowing data retrieval for different date ranges providing the ability to use online learning that best fits the component.

This component will also require the integration with different components with a RESTful API to be aware of new deployments to be analysed and to warn the SelfHealing component about any differing behaviour. A Flask server is used to provide the API easing the integration with different components.

Due to the use of Python programming language by the previous libraries, Python has also been selected as the main language of the prototype.

10 Security Self-learning Implementation

10.1 Functional description

The main purpose of the Security Self-learning component is to provide capabilities to train anomaly detection models. It receives data from the Security Monitoring component (see Fig. 16). As a first necessary step, a specified subset of the data has to be used to train a behavioural model. This subset of data, along with the necessary configuration files, is provided to the Model Trainer component (see Figure 17), which eventually stores every trained model in the Model Repository (part of the Infrastructural Elements Catalogue). Once a model is trained, this step is repeated only if requested to do so. A trained model is loaded from the Model Repository to carry out anomaly detection of the data collected by the Security Monitoring component. This task is thus carried out as part of the Security Monitoring workflow. Under previously specified conditions, e.g. high number of anomalies in a short time period, the Security Monitoring component will notify the Self-healing component.

Internally, the Model Trainer needs to train two different machine-learning models: the log parser and the anomaly detector. The log parser is in charge of transforming raw logs received from the Security Monitoring into structured logs. The log parser is thus trained in an unsupervised way. Then, the anomaly detector is trained on the stream of structured logs to learn patterns representing the normal behaviour. Afterwards, as already mentioned, a trained log parser and anomaly detector will be loaded by the Anomaly Detection component of Security Monitoring in order to provide an anomaly score for every incoming log message. Figure 20 illustrates this process.

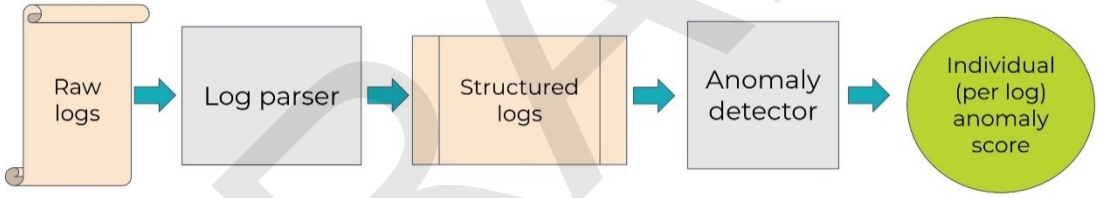

Figure 20. Internal functioning of the Model Trainer.

The sequence diagram describing the functioning of the Security Self-learning component can be seen in Figure 16, integrated with that of the Security Monitoring.

10.2 Requirements covered by this prototype

The user requirements are listed under Security Monitoring Implementation section in Table 6. Internal requirements covered by this prototype are listed in Table 10.

Table 10. Internal requirements for Security Self-learning.

Title	Implementation Status	Requirement Coverage
Add code into the project source repository	Partially satisfied	The repository has been created and the code is being uploaded regularly https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-controller
Implement REST API specification	In progress	A first version of the OpenAPI has been defined and put under configuration control

Title	Implementation Status	Requirement Coverage
Implement specification first approach	Satisfied	In order to speed-up the implementation of changes derived from the expected evolution of the REST API, we have implemented a specification first approach with OpenAPI generator.
Prepare for deployment	Satisfied	Part of the private repository (not in project's Gitlab).
Provide fast deployment alternative for deployment, testing and evaluation	Satisfied	Part of the private repository (not in project's Gitlab).
Include usage documentation	Satisfied	Part of the private repository (not in project's Gitlab).
Unitary test	Planned	Not yet available
Integration test	Planned	Not yet available
Continuous integration	Planned	Not yet available

10.3 Fitting into overall PIACERE Architecture

Security Self-learning's architecture and fitting into the overall PIACERE Architecture has depicted in section 10.4.1.

10.4 Technical description

10.4.1 Prototype architecture and Components description

The Security Self Learning component consists of a single architecture element, referred to as Model Trainer. Its architectural integration with that of the Security Monitoring is depicted in Figure 17.

The Security Monitoring controller triggers via API the model training, providing the necessary data and configuration files. As a result of the training process, a new log parser and a new anomaly detection model are created. These objects belong to the Infrastructural Elements Catalogue and are accessible via API as well.

Additionally, a dashboard is available as a submodule of the existing UI (see IDE Plug-in/Dashboard in Figure 12) to interact with the Model Trainer both for the training of the log parsers and anomaly detection models, as well as for visualization of intermediate and final results.

10.4.2 Technical specifications

Input:

- Data stemming from the Security Monitoring component. The data is already aggregated from different sources by the Security Monitoring component using ELK, which is directly accessed by the Model Trainer.

Programming languages/tools:

- Python: popular data science and machine learning libraries are used, mainly numpy, pandas, pytorch and transformers.

Dependencies:

- Grafana dashboard (deployment).
- ELK stack: storing raw log data.

11 Self-healing Implementation

11.1 Functional description

The PIACERE self-healing component receives events form the rest of the monitoring and self-learning components and based on the nature of the issue it launches different fixing strategies.

The Figure 21 presents the sequence diagram for the main activity that the self-healing should implement the notification processing workflow.

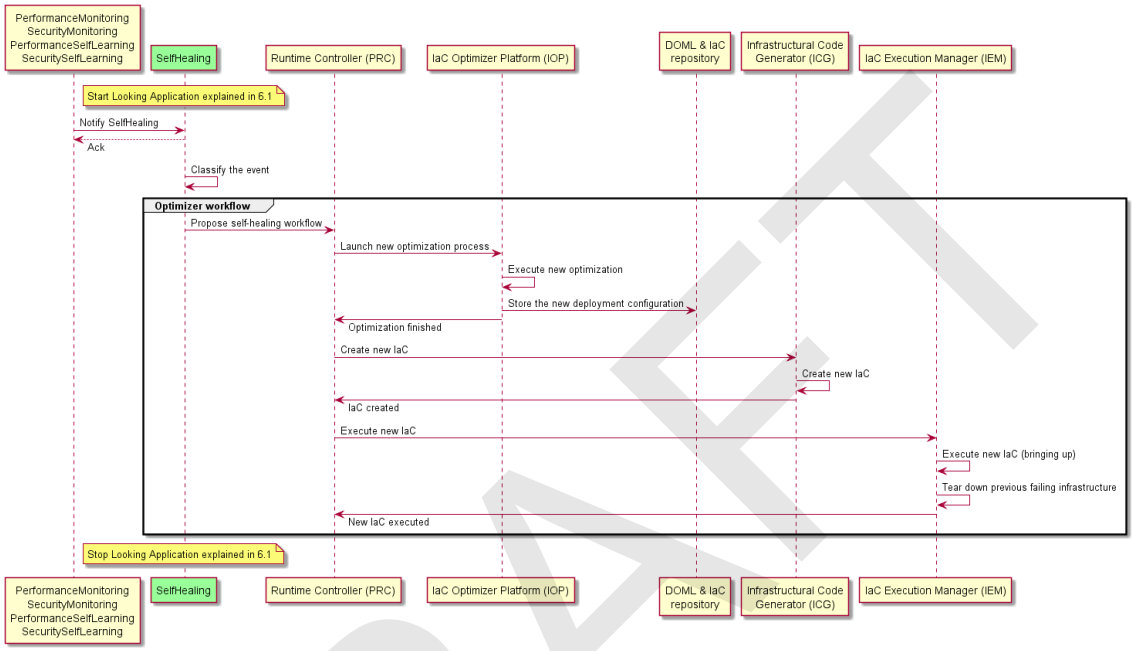

Figure 21. Self-healing sequence diagram

The first implementation of the self-healing aims to:

- Wait for event notifications form the rest of the monitoring and self-learning components in the IA (infrastructure Advisor).
- Classify the event to identify the corresponding strategy and other possible aspects in future such as the severity.
- Queues the strategies to be applied
- Request the execution of the fixing strategy to the PRC (Piacere runtime controller)

Apart from this workflow we have also implemented some supporting features to enable the monitoring of the notifications received and the management the types notifications and strategies applied to each of them.

11.2 Requirements covered by this prototype

The user requirements from WP2 satisfied by this interim version are described in the Table 11. All these requirements are being polished and adapted as the project advances and we gain knowledge on the use cases and on the implemented components.

Table 11. Self-healing related user requirements from WP2

Req ID	Description	Implementation Status	Requirement Coverage
REQ16	*Runtime security monitoring should contribute to mitigation actions taken when considering plans and strategies for runtime self-healing actions.*	In progress	The self-healing general approach, related to types of notifications and strategies has been established
REQ17	Deployment of runtime security monitoring should happen seamlessly or with minimal effort and configuration required by the user.	In progress	The self-healing will receive configuration request from the monitoring controller
REQ46	*The monitoring component shall gather metrics from the instances of the infrastructural elements at run time. These metrics need to be related to the NFR and accessible to the IOP (through the dynamic part of the infrastructural catalogue).*	In discussion	Metrics will be fed by the monitoring component about performance and security. We are evaluating the possibility of feeding the Infrastructural elements catalogue with information about self-healing actions.
REQ47	Full monitoring stack. *The monitoring component shall include the needed elements in the stack to monitor the infrastructural elements*	In progress	Self-healing components are being dockerized and deployed with container choreography tools.
REQ92	*Self-healing component shall receive notifications from the self-learning.*	In progress	Self-healing component implements a REST API to receive notifications from all the components involved in the application infrastructure monitoring.
REQ93	*Self-healing component shall classify the events received from the self-learning and derive corrective actions.*	In progress	Self-healing provides a classification field as part of the notification message.
REQ94	*Self-healing component shall inform the run time controller about the different components to orchestrate (the workflow to be executed).*	*In discussion* Planned	This will be addressed during the next period.
REQ97	*The Self-healing components provide feedback on the DOML code, without doing automatic writes. The end user can choose to accept or not the feedback received. (ex REQ56&75).*	*In discussion* Not addressed	The information is planned to be added to the Infrastructural elements catalogue instead of in the DOML

The internal requirements satisfied by this interim version are described in the Table 12. All these requirements are as well polished and adapted as the project advances.

Table 12. Self-healing related internal requirements

Title	Implementation Status	Requirement Coverage
Implement OpenAPI specification	Satisfied	A first version is implemented.

Title	Implementation Status	Requirement Coverage
Implement specification first approach	Satisfied	A preliminary catalogue of infrastructural elements is successfully loaded and read by the algorithm.

11.3 Fitting into overall PIACERE Architecture

The Self-Healing is one of the components of the PIACERE architecture. It is part of the Infrastructure Advisor package in the runtime phase of PIACERE (Figure 12). It interacts with other tools in the PIACERE ecosystem:

- Performance Monitoring, Security Monitoring, Performance Self-Learning and Security Self-Learning components send notifications to the Self-Healing component.
- Self-Healing component proposes self-healing workflow to the Runtime Controller component.

11.4 Technical description

This subsection is devoted to describing the technical specification of this first prototype. First, the main architecture and the components of the prototype are shown and described in section 11.4.1. This subsection finishes with the technical specifications of the developed system in *Section 11.4.2*.

11.4.1 Prototype architecture and components description

Self-healing architecture is based on a microservices style which splits the front-end, only for testing purposes in this stage, and the backend, so that´s it´s easier to scale and survive infrastructure issues.

In order to manage the events-oriented architecture, in this first prototype, Kafka streaming solution has been chosen.

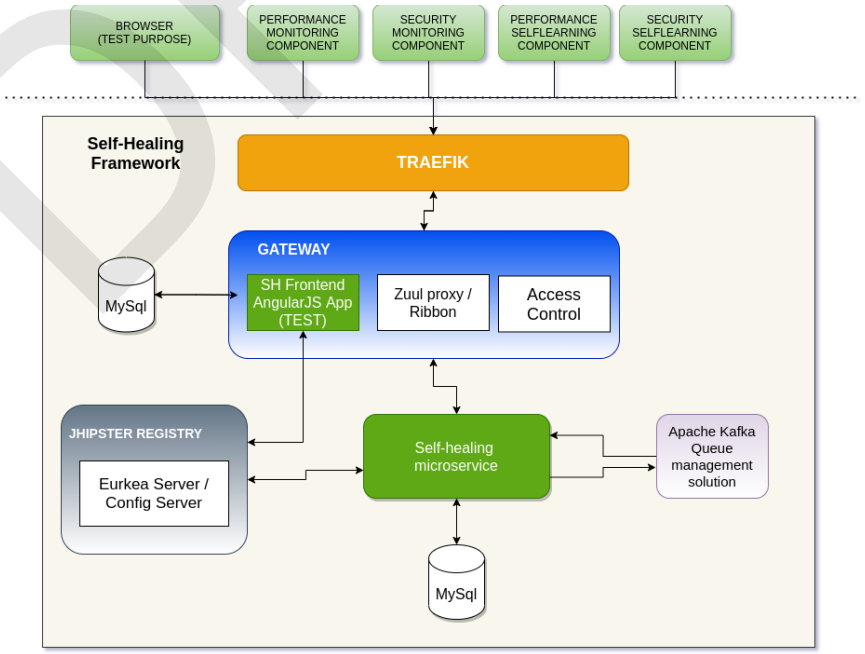

Figure 22. Self-healing internal architecture

This first prototype of the Self-healing is composed by two principal components, which main purpose are briefly described as follows. It should be noted that further detail of this components is provided in the upcoming *Section 0*.

- Self-healing service: this component manages all the logic of the self-healing. It implements the necessary logic to treat the notifications received by the components involved in the self-healing mechanism, exposing a REST service in order to simplify the interaction, and the work for communicating with the runtime controller in order to propose the self-healing mechanisms..
- Self-healing test frontend: This component has been developed to simplify the integration and test of the self-healing with the rest of the components.

In addition, some considerations about the other components of the Infrastructural Elements Catalogue:

- Access control. JSON Web Token (JWT)[23] mechanism used. A stateless security mechanism which uses a secure token that holds the user´s login name and authorities.
- Data persistence in MySQL database.
- JHipster Registry[24]. Service discovery using Netflix Eureka[25].
- Apache Kafka[26]: Event streaming solution to capture real-time data from the related components which need to send notifications to the Self-healing component.

11.4.2 Technical specifications

This prototype has been developed using JHipster Framework, which provides all the needed technologies and configuration options for a modern web application and microservice architecture.

This framework uses Spring Boot to develop, deploy and test the application.

In the client side, the test frontend gateway uses Yeoman, Webpack, Angular and Bootstrap technologies.

In the server side, the Self-healing microservice, uses Maven, Spring, Spring MVC Rest, Spring Data JPA and Netflix OSS.

The technology used in this first version of the POC to manage the events received is Kafka.

[23] https://jwt.io/introduction
[24] https://www.jhipster.tech/
[25] https://spring.io/projects/spring-cloud-netflix
[26] https://kafka.apache.org/

12 Monitoring Controller Delivery and usage

12.1 Installation instructions

There are many ways to run this component:

- Run the component in isolation
- Run with Docker
- Run with a Docker compose
- Run with Vagrant

Each approach is described into its corresponding README in the PIACERE code repository.

12.1.1 Component in isolation

The installation of the component in isolation is described at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/mc/mc

The requirement to run the component in isolation is to have Python 3.5.2+. In order to execute the component, we have to carry out three steps:

- Download the code
- Install the requirements
- Launch the Python module

To download the code we will use git:

```
git clone https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/mc/mc.git
```

To install the requirements we will use pip, so we will require to have the pip3 python tool:

```
cd mc
pip3 install -r requirements.txt
```

NOTE: the module has been developed on linux and therefore even if Python is multi-platform, we cannot ensure that the requirements are multiplatform as well. Therefore, running this step in non-linux systems may have some issues.

To launch the Python module, we require to have the port 8080 available and run:

```
python3 -m mc
```

12.1.2 Docker

The installation with Docker is also described at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/mc/mc

The requirements to run the component with Docker is to have Docker installed, we have used Docker version 20.10.10 with linux/amd64 architecture. In order to execute the component we have to carry out three steps:

- Download the code
- Build the image
- Run the image

To download the code, we will use git:

```
git clone https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/mc/mc.git
```

To build the image:

```
cd mc
docker build -t mc .
```

NOTE: the image relies on linux kernel, therefore it requires a Docker installation able to run linux based machines.

To run the image in a container we require to have the port 8080 available and run:

```
docker run -p 8080:8080 mc
```

12.1.3 Docker compose

The installation with docker-compose is described at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-deploy this docker-compose is a partial integration of components of WP6 currently we cover two components: monitoring controller and performance monitoring in the future we will cover all the remaining components: (security monitoring, performance self-learning, security self-learning and self-healing.).

The requirements to run the component with Docker is to have Docker installed, we have used Docker version 20.10.10 with linux/amd64 architecture and docker-compose version 1.29.0 . In order to execute the component, we have to carry out three steps:

- Download the code
- Setup relevant variables
- Build the images
- Run the docker-compose

To download the code, we will use git:

```
git clone –recurse-submodules https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-deploy.git
```

To setup relevant variables we need to identify the variables without values, and give value to them:

```
echo list variables to be setup
cat .env | grep -e ".*=\s*$"
```

Assign values to those variables. The current set of values are the ones show bellow, but the are subject to change as the development advance, therefore it is advisable to check the current list using the instruction above (cat …)

```
export SERVER_HOST=192.168.56.1.nip.io
export ADMIN_PASSWORD=somestrongpassword
```

NOTE: https://nip.io is a service that allows doing a mapping between any IP to a hostname.

To build the images:

```
cd pm-deploy
docker-compose build
```

NOTE: the images relies on linux kernel, therefore it requires a Docker installation able to run linux based machines.

To run the docker-compose we will need the port 443 available:

```
docker-compose up
```

12.1.4 Vagrant

The installation with Vagrant is described at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-vagrant this vagrant run the partial integration docker-compose described above.

The requirements to run the component with Vagrant is to have VirtualBox and Vagrant installed, we have used VirtualBox version 6.1.22 and Vagrant version 2.2.16. In order to execute the component, we have to carry out three steps:

- Download the code
- Start the Vagrant machine
- Build the images
- Run the docker-compose

To download the code, we will use git:

```
git clone –recurse-submodules https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-vagrant.git
```

To start the Vagrant machine

```
cd pm-vagrant
vagrant up
```

To build the images:

```
vagrant ssh
cd /vagrant-project/git/pm-deploy/
docker-compose --env-file /vagrant-project/.local/develop/.env build
```

To run the docker-compose:

```
docker-compose --env-file /vagrant-project/.local/develop/.env up -d --no-build --remove-orphans
```

12.2 User Manual

The Monitoring controller can be used through its REST API, described at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/mc/mc/-/blob/main/src/mc/openapi/openapi.yaml

In order to access that API in the running component, we need to specify the HTTP schema, the host and the port. Then it will be possible to access the REST API documentation in the same running instance where we can invoke the services:

For the component in isolation, the way to access the swagger UI, showing the REST API, will be http://localhost:8080/api/v1/ui/, in the rest of the execution options the access will depend on the server and the port specified and it will look like https://192.168.56.1.nip.io:8443/mc/api/v1/ui/

In that address we will find the standard swagger UI shown in Figure 23. The swagger UI will list the operations available and it will allow us to invoke them.

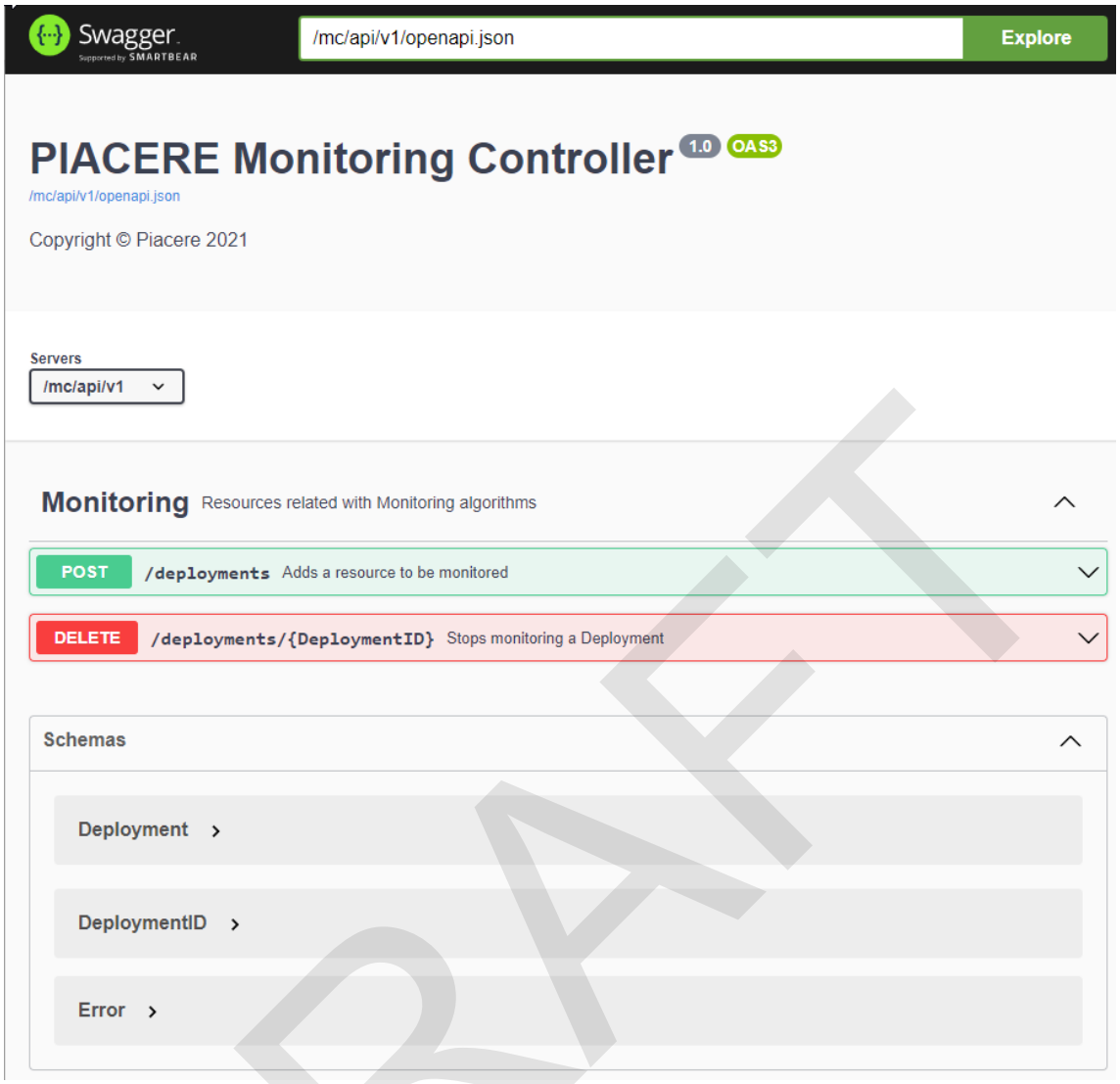

Figure 23. Monitoring Controller swagger ui

Anyway, even if the swagger UI provides a valuable resource to understand and test the API, the real way to use the component will be to integrate it with other components. To do so the best way is to get profit from the OpenAPI based client code generators such as:

- Openapi-generator: https://github.com/OpenAPITools/openapi-generator
- Swagger-codegen: https://github.com/swagger-api/swagger-codegen

12.3 Licensing information

To be defined

12.4 Download

The component code is available at PIACERE code repository at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/mc/mc

The source code will be available on the public git repository and accessible through the project's website https://www.piacere-project.eu/. At the time of writing this deliverable, the source code is provided under request through an email to the address appearing on the website (https://www.piacere-project.eu/) in the footer under "Contact Us".

13 Performance Monitoring Delivery and usage

13.1 Installation instructions

This component shares the part of the development environment with the Monitoring controller. In that sense it shares some of their ways to be executed:

- Run with a docker compose
- Run with Vagrant

Besides, as this component is composed by separate running services, it makes no sense to apply some of the execution methods available in the Monitoring controller such as: run the component in isolation or run with docker. However, focussing in the Performance monitoring controller there can be situations, such as during development, where it can make sense to run this component in isolation. For this specific case we provide specific guidelines.

Each approach is described into its corresponding README in the PIACERE code repository.

13.1.1 Performance monitoring controller in isolation

The installation of the component in isolation is described at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pmc

The requirements to run the component in isolation is to have java and maven. In order to execute the component, we have to carry out two steps:

- Download the code
- Launch the spring boot application

To download the code, we will use git:

```
git clone https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pmc.git
```

To run the spring boot application

```
mvn run
```

13.1.2 Docker compose

The installation with docker-compose is described at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-deploy. this docker-compose is a partial integration of components of WP6 currently we cover two components: monitoring controller and performance monitoring in the future we will cover all the remaining components: (security monitoring, performance self-learning, security self-learning and self-healing.). The usage details are available above 12.1.3

13.1.3 Vagrant

The installation with vagrant is described at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-vagrant this vagrant run the partial integration docker-compose described above. The usage details are available above 12.1.4

13.2 User Manual

This component has three different sub-components. In the following subsections we provide the user manual for each of them.

13.2.1 Performance Monitoring controller

The Performance Monitoring controller is used through its REST API, described at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pmc/-/blob/main/openapi.yaml

In order to access that API in the running component, we need to specify the HTTP schema, the host and the port. Then It will be possible to access the REST API documentation in the same running instance where we can invoke the services:

For the component in isolation, the way to access the swagger UI, showing the REST API, will be http://localhost:8080/pmc/api/v1/ui/, in the rest of the execution options the access will depend on the server and the port specified and it will look like https://192.168.56.1.nip.io:8443/pmc/api/v1/ui/

In that address we will find the standard swagger UI shown in Figure 24. The swagger UI will list the operations available and it will allow us to invoke them.

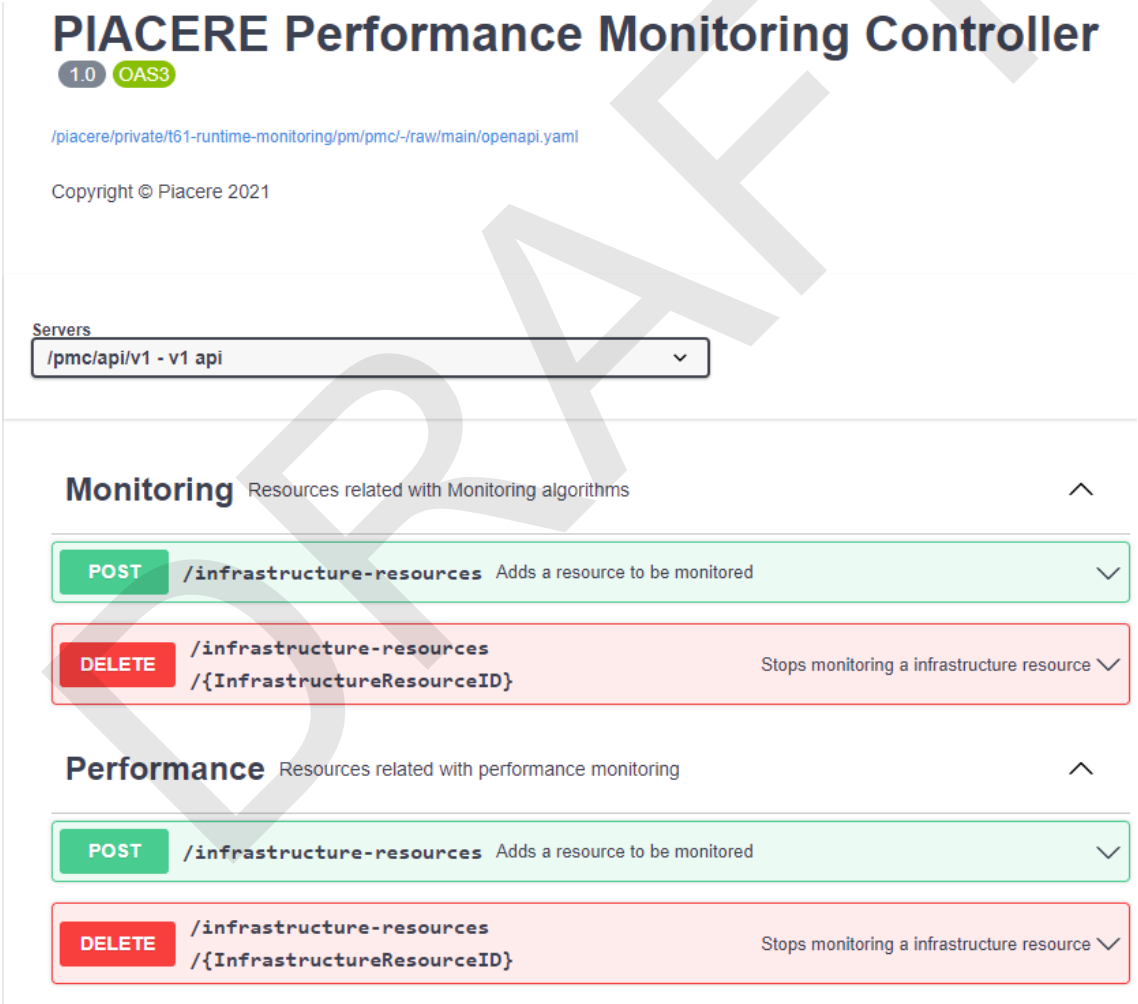

Figure 24. Performance Monitoring Controller swagger ui

Anyway, even if the swagger UI provides a valuable resource to understand and test the API, the real way to use the component will be to integrate it with other components. To do so the best way is to get profit from the OpenAPI based client code generators such as:

- Openapi-generator: https://github.com/OpenAPITools/openapi-generator

- Swagger-codegen: https://github.com/swagger-api/swagger-codegen

13.2.2 Influxdb

Influxdb will be used following the standard user guideline https://docs.influxdata.com/influxdb/v2.0/. The instance will be available in different URLs depending on the execution method selected. For example, if we use the Vagrant method, it will be accessible at https://influxdb.192.168.56.1.nip.io:8443/. As another example, as shown in the Figure 25, in our internal continuous integration framework the component is accessible at https://influxdb.piacere.esilab.org:8443/

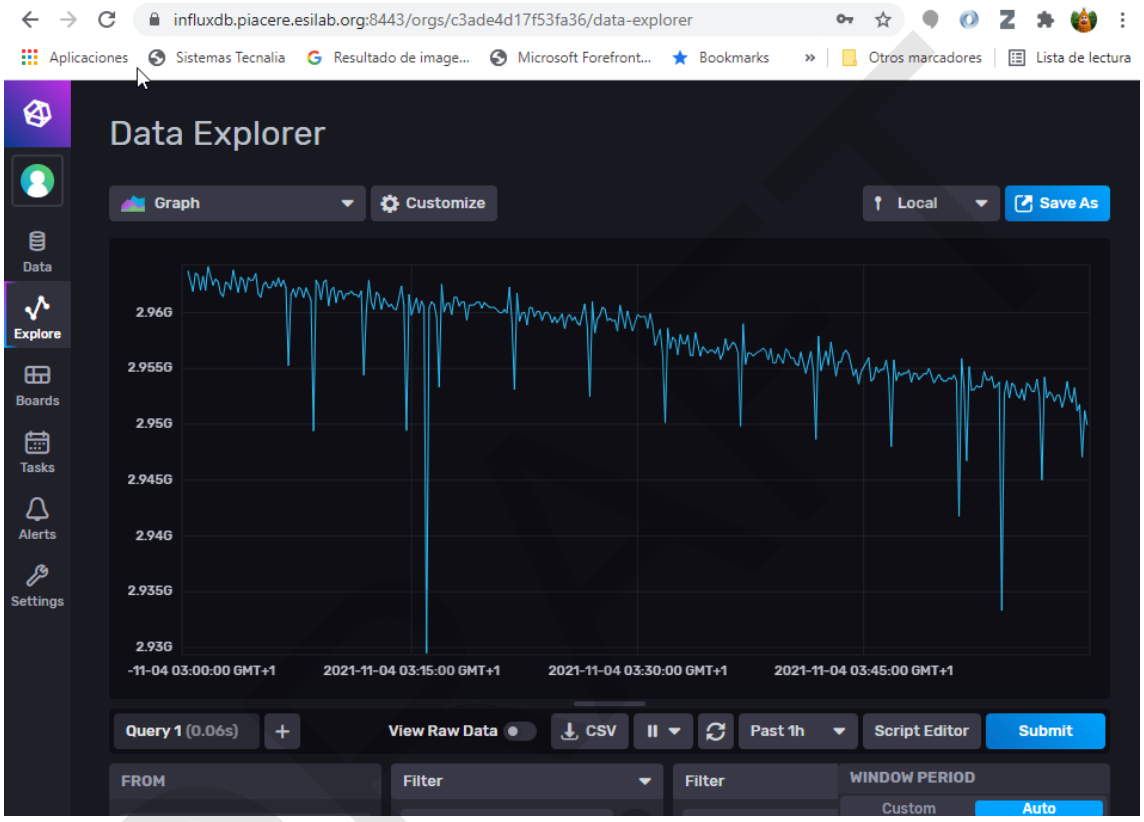

Figure 25. Influxdb

13.2.3 Grafana

Grafana will be accessible at https://192.168.56.1.nip.io:8443/grafana/

Grafana will be used following the standard user guideline https://grafana.com/docs/grafana/latest/getting-started/getting-started/. The instance will be available in different URLs depending on the execution method selected. For example, if we use the Vagrant method, it will be accessible at https://192.168.56.1.nip.io:8443/grafana, as shown in the Figure 26. As another example, in our internal continuous integration framework, the component is accessible at https://piacere.esilab.org:8443/grafana

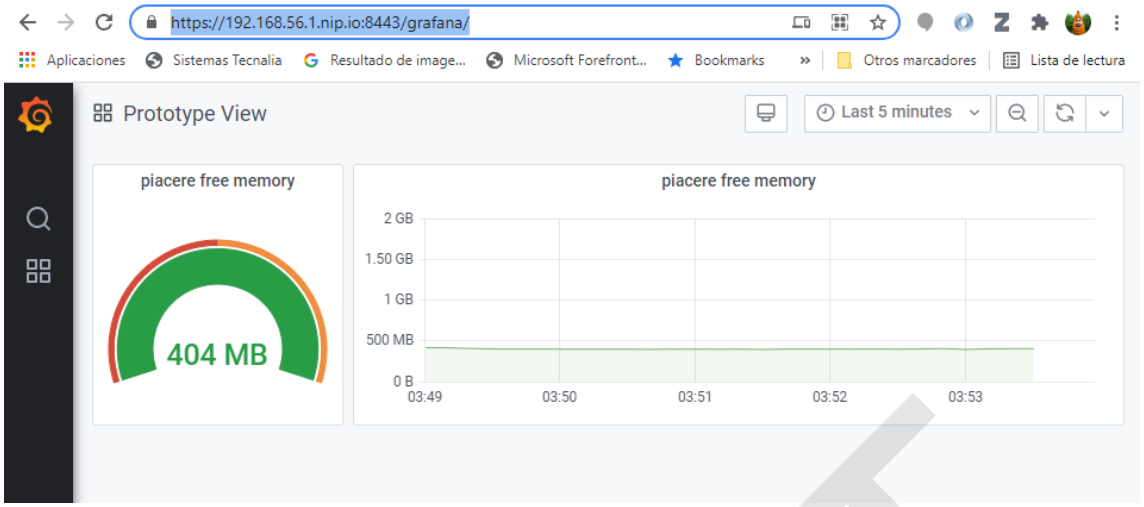

Figure 26. Grafana

13.3 Licensing information

To be defined

13.4 Download

The component code is available in the PIACERE code repository at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-deploy.

Besides, a testing oriented agent infrastructure can be deployed using the code available at https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pma-deploy.

The source code will be available on the public git repository and accessible through the project's website https://www.piacere-project.eu/. At the time of writing this deliverable, the source code is provided under request through an email to the address appearing on the website (https://www.piacere-project.eu/) in the footer under "Contact Us".

14 Security Monitoring Delivery and usage

14.1 Security Monitoring Service

The Security Monitoring Service is expected to be deployed eventually as a set of containers. However, currently the deployment is available using specific Ansible playbook on top of an environment (i.e., inventory built either manually or using the Vagrant tool). The deployment consists of (Figure 17):

- **Security Monitoring Controller**: API entry point for underlying components (also the Model trainer of the Security Self-learning). It is also in charge of regularly pushing events towards external services such as Self-healing components.
- **Security Monitoring Manager** (includes Kibana dashboard): collects all the necessary events from the Security Manager Agents from the infrastructures and provides data feed to the Model Trainer service (of the Security Self-learning)
- **Security Manager Agents**: these are in charge of collecting monitoring data and forwarding this towards the Manager for analytics and storage.

14.2 Installation Instructions

Installation of the Security Monitoring components consists of the Controller and the Monitoring Manager.

14.2.1 Installing Controller

The code resides on the project's repository: https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-controller

```
$ git clone git@git.code.tecnalia.com:piacere/private/t64-runtime-security-
monitoring/security-monitoring-controller.git
```

To run the server, please execute the following from the root directory:

```
pip3 install -r requirements.txt
python3 -m swagger_server
```

To run the server on a Docker container, please execute the following from the root directory:

```
# building the image
docker build -t swagger_server .

# starting up a container
docker run -p 8080:8080 swagger_server
```

14.2.2 Installing Monitoring Manager

These are summary of the code available on the repository:

https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-deployment

First, checkout Wazuh's tag `v4.1.5` into the current directory:

```
$ git clone https://github.com/wazuh/wazuh-ansible.git
$ git checkout tags/v4.1.5
```

You need to update 2 files:

- wazuh-ansible/playbooks/wazuh-agent.yml
- wazuh-ansible/playbooks/wazuh-odfe-single.yml

And provide IPs of the manager and the agents. IPs can be found in the inventory file of the Ansible script: https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-deployment/-/blob/develop/security-monitoring-ansible/environments/vagrant-1manager-2agents/inventory.txt

Provision Wazuh server and Wazuh agents:

```
$ cd security-monitoring-ansible
$ ENVIRONMENT=vagrant-1manager-2agents make create provision
```

14.3 User Manual

14.3.1 Controller

Figure 27 depicts Security Monitoring's API as it is served by the Security Monitoring Controller after it is made available (deployed).

Open your browser to here:

http://localhost:8080/security-monitoring/v1/ui/

Your Swagger definition lives here:

http://localhost:8080/security-monitoring/v1/swagger.json

Figure 27. Security Monitoring part of the Security Monitoring Controller API.

14.3.2 Monitoring Manager

Check the running instances:

- Navigate browser to: https://192.168.33.10:5601 (the IP from the inventory file), login with default credentials admin:changeme. Navigate to wazuh section on the left hand-side.
- You should see 2 agents registered and running with Wazuh.

List of indices:

```
curl -X GET https://192.168.33.10:9200/_cat/indices?v -u admin:changeme -k
```

List all entries in the index wazuh-alerts:

```
$ curl -X GET https://192.168.33.10:9200/wazuh-alerts-4.x-2021.11.03/_search -u
admin:changeme -k
```

14.4 Licensing information

Currently the code is owned by XLAB, and the license still to be defined.

14.5 Download

The Security Monitoring Controller's code is available on the project repository: https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-controller

The Security Monitoring Deployment code is available here:

https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-deployment/-/tree/develop

15 Performance Self-learning Delivery and usage

15.1 Performance Self-learning Service

The structure of the Performance Self-learning is split in two main sections. The first section, the libraries (folder "libs"), contains functional code used for dataset manipulation, input/output and learning.

- dataset.py: It contains the utility functions used to manipulate datasets and transform them to be ready to be feed to learning algorithms.
- Io.py: Input/output code used for data retrieval from different sources like the filesystem or a given database.
- learning.py: Everything related to learning algorithms to predict different outcomes from the data.

The second main section is the Performance Self-learning engine (folder "src") used to receive notifications of different deployments, acquire and feed data to learning algorithms and send notifications to the SelfHealing component.

An essential part of the code is also the generated swagger client used to notify the SelfHealing component.

15.2 Installation Instructions

The Performance Self-learning is run as a standalone component at the moment. The code must be downloaded from a git repository, setup properly and then it can be executed.

To download the code, we have to clone the repository:

```
git clone https://git.code.tecnalia.com/piacere/private/t62-self-learning/psl.git
```

Once the repository has been cloned, the library requirements must be installed. Move to the cloned repository directory and install requirements:

```
cd psl
pip3 install -r requirements.txt
```

In the last step, the connection parameters must be setup. Rename the `connection.ini.sample` in the src directory as `connection.ini` and set the correct values in the file.

Finally, we can execute the component with the following command:

```
python3 -mc src/main.py
```

15.3 User Manual

The Performance Self-learning component behaviour expects notifications through the RESTful API. The specification of the API can be found at:

```
https://git.code.tecnalia.com/piacere/private/t62-self-learning/psl/-
/blob/main/docs/self-learning-openapi.yaml
```

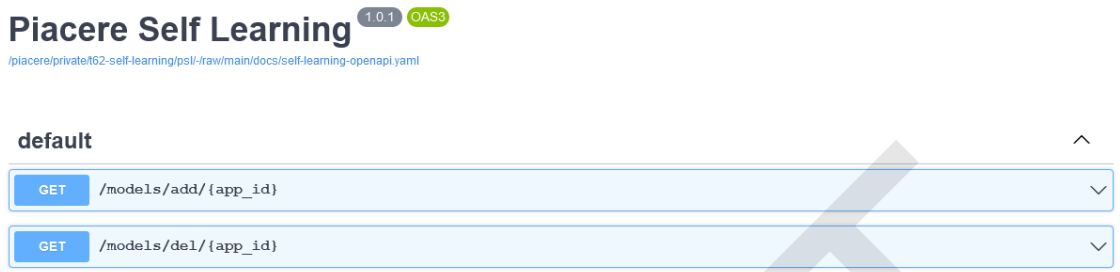

Figure 28. Performance Self-learning OpenAPI

Once the component is running, no more interactions than the call to those two endpoints are necessary. The following example commands show how to call those endpoints.

Add a model:

```
curl -X 'POST' 'http://localhost:8080/psl/api/v1/models/add/1' -H 'accept:
application/json' -d ''
```

Delete a model:

```
curl -X 'DELETE' 'http://localhost:8080/psl/api/v1/models/del/1' -H 'accept:
application/json'
```

15.4 Licensing information

Currently the code is owned by Tecnalia, and the license still to be defined.

15.5 Download

The Performance Self Learning's code is available on the project repository:

https://git.code.tecnalia.com/piacere/private/t62-self-learning/psl.git

16 Security Self-learning Delivery and usage

16.1 Security Self-Learning Service

The Security Self-Learning service is expected to be deployed as a single microservice that will be exposed to the Security Monitoring Controller and will coordinate the whole training process, from the connection to the data source containing raw logs, to the training of the log parser and AD model and their storage in the Model Repository

16.2 Installation Instructions

All the different services composing the Security Self-Learning are currently running as standalone services which have to be manually executed. In all cases, conda[27] environments are used to handle dependencies in an isolated manner. The codebase is closed and hosted on private repositories.

16.3 User Manual

As describer earlier in this document, the interaction with the Security Self-Learning service will be done exclusively by the Security Monitoring controller, which exposes an API (included below in Figure 29) for such purposes.

Example of reading all available ad_models trained by the self_learning instance that are available to be used by the Security Monitoring Anomaly Detector:

```
curl -X 'GET' 'https://localhost:8080/security-monitoring/v1/ad_models' -H 'accept:
application/json'
```

Result (returning object with parent train_id reference and other details of the model:

```
[
  {
    "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "train_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "name": "string",
    "description": "string",
    "configuration": "string"
  }
]
```

[27] https://docs.conda.io/en/latest/

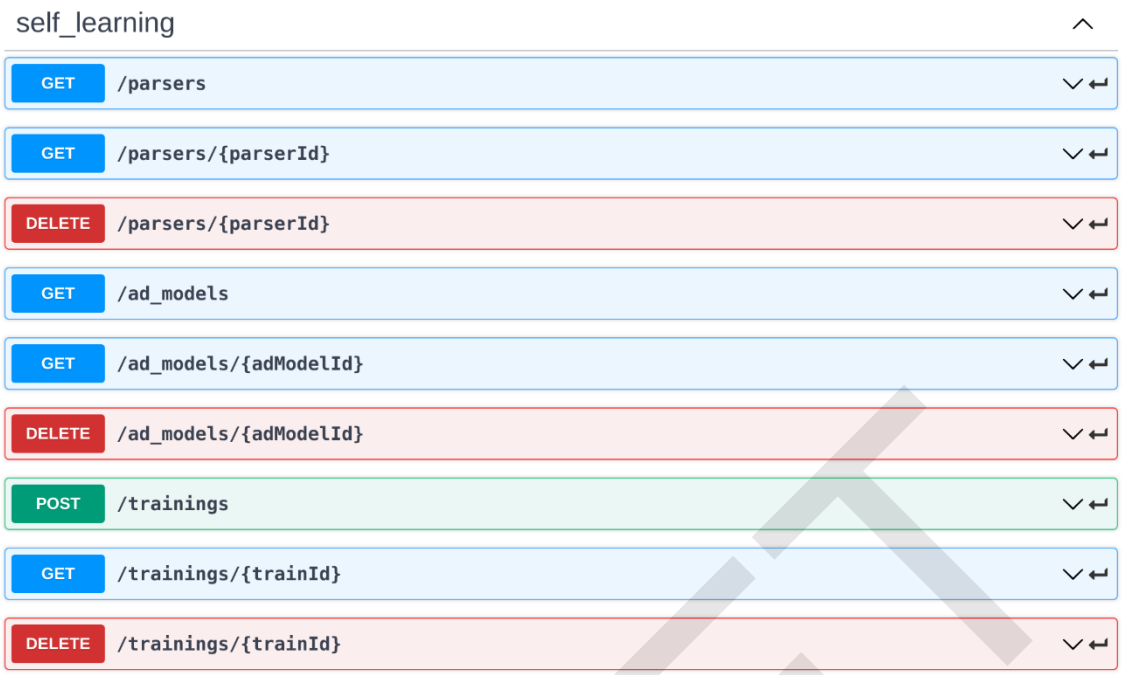

Figure 29. Self-learning API provided by Security Controller.

The dashboard is based on Grafana and is provided as an informative tool that would provide insights on the intermediate steps and results of the training process. The design of the exact functionalities that it will include is an on-going process.

16.4 Licensing information

Currently is closed source, owned by XLAB.

16.5 Download

The Security Monitoring Controller's code is available on the project repository. The Controller provides API endpoints to the Security Self-learning component: https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-controller

17 Self-healing Delivery and usage

17.1.1 Self-healing service

The main structure of the prototype developed in this first stage of the project is composed by the packages shown in the following Figure 30.

Figure 30. Self-healing project structure

Each of these packages has its own objective and its context within the whole prototype. Furthermore, these packages are also comprised by several JAVA classes. With all this, the main purpose and composition of each component is as follows:

- *com.piacere.selfhealing.service.aop.logging*: this package is composed by LoggingAspect.java, which defines the aspect for logging execution of service and repository Spring components.
- *com.piacere.selfhealing.service.client*: Composed by UserFeignClientInterceptor.java which implements RequestInterceptor.java. This class checks and add JWT token to the request header.
- *com.piacere.selfhealing.service.config*: this package contains all classes related to configuration purposes.
- *com.piacere.selfhealing.service.consumer*: this package contains classes to consume messages from the queue configured.
- *com.piacere.selfhealing.service.domain*: this package contains data model classes.
- *com.piacere.selfhealing.service.domain.enumeration*: this package contains enum objects.
- *com.piacere.selfhealing.service.producer*: this package contains classes to produce messages to the queue configured.
- *com.piacere.selfhealing.service.repository*: this package contains Spring Data SQL repository classes.
- *com.piacere.selfhealing.service.security*: this package contains Spring Security related classes for security management.
- *com.piacere.selfhealing.service.security.jwt*: this package contains Java Web Token security configuration related classes.

- *com.piacere.selfhealing.service.serde*: this package contains classes to serialize/deserialize queue messages received.
- *com.piacere.selfhealing.service.service*: this package contains self healing services for CRUD operations and other requirements needed.
- *com.piacere.selfhealing.service.service.dto*: this package contains self healing data transfer objects.
- *com.piacere.selfhealing.service.service.mapper*: this package contains mapping classes to map data transfer objects.
- *com.piacere.selfhealing.service.web.rest*: this package contains classes to expose Self-healing rest end points.
- *com.piacere.selfhealing.service.web.rest.errors*: this package contains error classes used in the rest end points.

17.1.2 Kafka streaming solution

In the context of Self-healing component, it has been implemented the necessary logic to manage the notifications received with the Kafka streaming solution.

The configuration needed:

```
kafka:
  bootstrap.servers: kafka:9092
  polling.timeout: 10000
  consumer:
    selfHealingMessage:
      enabled: true
      '[key.deserializer]': org.apache.kafka.common.serialization.StringDeserializer
      '[value.deserializer]': com.piacere.selfhealing.service.serde.SelfHealingMessageDeserializer
      '[group.id]': iec-self-healing
      '[auto.offset.reset]': earliest
  producer:
    selfHealingMessage:
      enabled: true
      '[key.serializer]': org.apache.kafka.common.serialization.StringSerializer
      '[value.serializer]': com.piacere.selfhealing.service.serde.SelfHealingMessageSerializer
  topic:
    selfHealingMessage: queuing.iec_self_healing.self_healing_message
```

Figure 31. Self-healing configuration

- Topic: Topic *queuing.iec_self_healing.self_healing_message* has been defined to associate all the events related to the Self-healing logic.
- In the context of Kafka, we need one producer and one subscriber to manage the events received
  - Producer: In charge of sending messages to the topic defined.

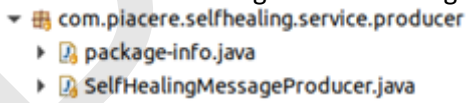

Figure 32. Self-healing producer

  - Consumer: In charge of processing the messages asynchronously.

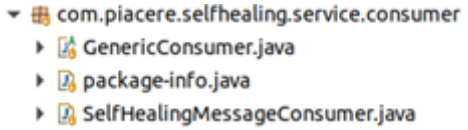

Figure 33. Self-healing consumer

17.2 Installation instructions

This project is executed in a Docker container.

There are docker compose files for each environment development/production.

To execute this project in the production environment, the next docker-compose files are used by the gitlab-ci continuous integration configuration:

- *docker-compose.yaml*, main file with all the services needed by the self-healing component.
- *docker-compose-dev.yaml*, traefik and portainer services configuration.
- *docker-compose-traefik-tecnalia-selfsigned.yaml*, traefik configuration for Tecnalia internal server.
- *docker-compose-expose.yaml*, traefik configuration to expose ports.

To execute this project in a development environment the *docker-compose-local-dev.yaml* file is needed:

- *git clone https://git.code.tecnalia.com/piacere/private/t63-self-healing/sh-deploy.git*
- *docker-compose -f docker-compose-local-dev.yaml up --build -d*
- *cd ./git/selfHealingService*
- *./mvnw -Pdev,api-docs -Dskip-tests*
- *cd ../../git/selfHealingGateway*
- *./mvnw -Pdev,webapp,api-docs -Dskip-tests*

Frontends Available services after initialization:

- JHipster registry: http://localhost:8761
- Self-healing test web app: http://localhost:8080
- Self-healing Api Documentation:
  http://localhost:8080/services/selfhealingservice/v3/api-docs

17.3 User manual

To test the self-healing functionalities:

- Login to the web app with user/password: admin/admin
- In the administration menu, access openApi.
- Choose SelfHealingService.
- Post a message through the Self-healing notify rest service.
- In this web app, entities menu, can be seen the message received and its status.

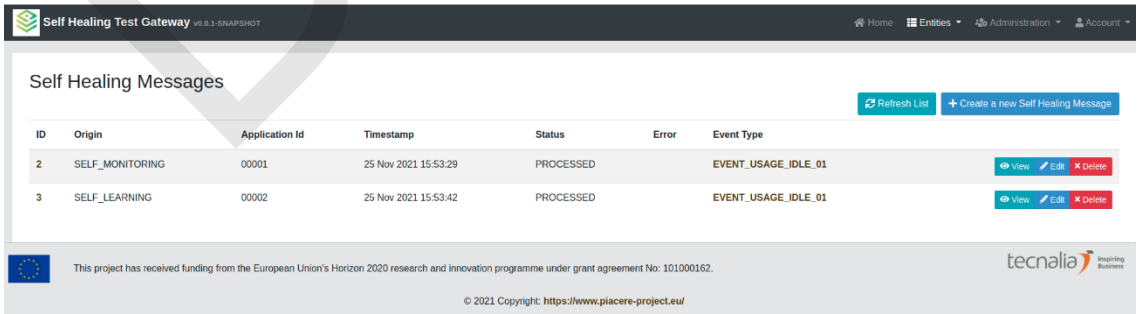

Figure 34. Messages received in the Self-Healing component

17.4 Licensing information

Information about license not included yet

17.5 Download

The code is available in Tecnalia GitLab repository:

https://git.code.tecnalia.com/piacere/private/t63-self-healing/sh-deploy

The source code will be available on the public git repository and accessible through the project's website https://www.piacere-project.eu/. At the time of writing this deliverable, the source code is provided under request through an email to the address appearing on the website (https://www.piacere-project.eu/) in the footer under "Contact Us".

18 Conclusions

Along this document, we have presented the current state of the development of the PIACERE run-time monitoring and self-learning, self-healing platform, together with the rationale that supports the decisions taken in this period.

As we have stated in the executive summary, the objective in this period has been to ensure the availability of key components, start working in the more challenging aspects, and finally establish the foundations for the integration. Regarding the availability of key components, we have deployed agents and databases in the monitoring area to have data from the very beginning. Besides, we have also provided means to recover the information gathered and stored by the monitoring components by the self-learning components. Regarding the challenging aspects, we have started working in the self-learning prediction algorithms and the self-healing approaches. Finally, regarding the establishments of baselines for the integration, we have worked in the formal definition of the interfaces and in the specification of the infrastructure required to run each component. That specification has been provided following the infrastructure as code principles using Dockerfile language. Dockerfile language allows to specify the characteristics of the container infrastructure required by the components developed in the project, including among other elements: environment variables, persistence, networking, and software packages.

During the next months, we will advance in the integration and the implementation of the core workflows of the PIACERE framework. This will include:

- The full deployment of an application including the agents and the configuration of the monitoring components.
- The simulation of problems in the infrastructure so that we can test the performance and security monitoring capabilities.

The progress will be reported in subsequent deliverables D6.2 (M24) and D6.3 (M30).

19 References

[1] S. Peyrott, “An Introduction to Microservices, Part 1,” 4 09 2015. [Online]. Available: https://auth0.com/blog/an-introduction-to-microservices-part-1/. [Accessed 15 11 2021].

[2] J. O.-E. L. E. M. Alonso, «Contribution to the uptake of Cloud Computing solutions: Design of a cloud services intermediator to foster an ecosystem of trusted, interoperable and legal compliant cloud services.,» Proceedings of the 15th International Conference on Web Information Systems and Technologies (WEBIST), Vienna, 2019.

[3] M. H. L. O.-E. Juncal Alonso, «ACSmI: A solution to address the challenges of Cloud services federation and monitoring towards the Cloud Continuum,» *International Journal of Computational Science and Engineering,* 2021.

[4] DECIDE Consortium;, “D5.4 Final Advanced Cloud Service meta-Intermediator,” 2019.

[5] I. O. f. Standardization, «ISO/IEC 19086:1-2016 Clod computing -Service level agreement (SLA) framework,» 2016.

[6] J. G. A. J. Alcaraz Calero, «Comparative analysis of architectures for monitoring cloud computing infrastructures,» *Future Generation Computer Systems ,* vol. 47, pp. pp.16-30, 2015.

[7] DECIDE Consortium, “DECIDE D3.15 Final multi-cloud native application composite CSLA definition,” 2019.

[8] Y. Verginadis, «D3.4: Workload optimisation recommendation and adaptation enactment,» H2020 Melodic, 2019.

[9] Telegraf, "Telegraf is the Agent for Collecting & Reporting Metrics & Data," [Online]. Available: https://www.influxdata.com/time-series-platform/telegraf/. [Accessed November 2017].

[10] N. M. Fuentes-García, J. Camacho y G. Maciá-Fernández, «Present and Future of Network Security Monitoring,» *IEEE Access,* vol. 1, nº 1, p. 99, 2021.

[11] “ossec (Open Source HIDS SECurity),” [Online]. Available: https://www.ossec.net/. [Accessed 2021].

[12] “Zeek (formerly Bro),” [Online]. Available: https://zeek.org/. [Accessed 2021].

[13] “wazuh,” [Online]. Available: https://wazuh.com/. [Accessed 2021].

[14] C. a. Z. A. Doersch, “Multi-task self-supervised visual learning,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[15] A. a. M. A. Gogna, “Semi supervised autoencoder,” in *International Conference on Neural Information Processing*, 2016.

[16] K. a. K. J. Pathak, «Reinforcement evolutionary learning method for self-learning,» *arXiv preprint,* nº arXiv:1810.03198, 2018.

[17] T. a. P. S. a. V. F. a. A. D. a. B. E. Cerquitelli, "Automating concept-drift detection by self-evaluating predictive model degradation," *arXiv preprint,* no. arXiv:1907.08120, 2019.

[18] J. a. L. A. a. S. Y. a. Z. G. Lu, «Data-driven decision support under concept drift in streamed big data,» *Complex and Intelligent Systems,* vol. 6, nº 1, pp. 157--163, 2020.

[19] A. a. P. D. a. B. Y. kishore Ramakrishnan, "Enabling self-learning in dynamic and open IoT environments," *Procedia Computer Science,* vol. 32, pp. 207--214, 2014.

[20] A. a. I. I. a. L. J. A. Carreño, «Analyzing rare event, anomaly, novelty and outlier detection terms under the supervised classification framework,» *Artificial Intelligence Review,* pp. 1--20, 2019.

[21] V. a. B. A. a. K. V. Chandola, "Anomaly detection: A survey," *ACM computing surveys (CSUR),* vol. 41, no. 3, pp. 1--58, 2009.

[22] H. M. a. R. J. a. B. A. a. B. J. P. a. G. J. Gomes, «Machine learning for streaming data: state of the art, challenges, and opportunities,» *ACM SIGKDD Explorations Newsletter,* vol. 21, nº 2, pp. 6-22, 2019.

[23] J. L. Lobo, "New perspectives and methods for stream learning in the presence of concept drift," 2018.

[24] P. a. H. G. Domingos, «A general framework for mining massive data streams,» *Journal of Computational and Graphical Statistics,* vol. 12, nº 4, pp. 945--949, 2003.

[25] A. Bifet, «Classifier concept drift detection and the illusion of progress,» de *International Conference on Artificial Intelligence and Soft Computing*, 2017.

[26] Hawkins y Douglas M., Identification of Outliers, Springer Netherlands, 1980.

[27] C. Aggarwal, Outlier Analysis, Springer International Publishing, 2017.

[28] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola y R. C. Williamson, «Estimating the Support of a High-Dimensional Distribution,» *Neural Comput.,* vol. 13, nº 7, p. 1443–1471, 2001-07.

[29] C. Cortes y V. Vapnik, «Support Vector Networks,» *Machine Learning,* vol. 20, pp. 273-297, 1995.

[30] M. M. Breunig, H.-P. Kriegel, R. T. Ng y J. Sander, «LOF: identifying density-based local outliers,» de *ACM sigmod record*, 2000.

[31] F. T. Liu, K. M. Ting y Z.-H. Zhou, «Isolation Forest,» de *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, 2008.

[32] S. C. Tan, K. M. Ting y T. F. Liu, «Fast Anomaly Detection for Streaming Data,» de *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, 2011.

[33] P. J. Rousseeuw y K. V. Driessen, «A Fast Algorithm for the Minimum Covariance Determinant Estimator,» *Technometrics,* vol. 41, nº 3, pp. 212-223, 1999.

[34] P. J. Rousseeuw, «Least Median of Squares Regression,» *Journal of the American Statistical Association,* vol. 79, nº 388, pp. 871-880, 1984.

[35] H.-K. Peng, «Multi-scale compositionality,» *PLoS One,* 2015.

[36] W. Sun, A. Javaid, Q. Niyaz y M. Alam, «Deep Learning Approach for Network Intrusion Detection System,» de *Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, Brussels, 2015.

[37] T. Luo y S. Nagarajan, «Distributed Anomaly Detection Using Autoencoder Neural Networks in WSN for IoT,» de *2018 IEEE International Conference on Communications (ICC)*, 2018.

[38] I. Kakanakova y S. stoyanov, «Outlier Detection via Deep Learning Architecture,» de *18th International Conference on Computer Systems and Technologies*, New York, 2017.

[39] W. Zhang, W. Guo, X. Liu, Y. Liu, J. Zhou, B. Li, Q. Lu y S. Yang, «LSTM-Based Analysis of Industrial IoT Equipment,» *IEEE Access,* 2018.

[40] B. A. Mudassar, J. H. Ko y S. Mukhopadhyay, «An Unsupervised Anomalous Event Detection Framework with Class Aware Source Separation,» de *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[41] F. Awwadl, «Power profiling of microcontroller's instruction set for runtime hardware Trojans detection without golden circuit models,» de *Design, Automation Test in Europe Conference Exhibition*, 2017.

[42] S. Faghih-Roohi, S. Hajizadeh, A. Núñez, R. Babuska y B. D. Schutter, «Deep convolutional neural networks for detection of rail surface defects,» de *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016 .

[43] L. N. Nielsen, K. A. Steen, R. N. Jørgensen y H. Karstoft, «DeepAnomaly: Combining Background Subtraction and Deep Learning for Detecting Obstacles and Anomalies in an Agricultural Field,» *Sensors,* 2016.

[44] A. Fuentes, S. Yoon, S. C. Kim y D. S. Park, «A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition,» *Sensors,* 2017.

[45] W. Yan y L. Yu, «On Accurate and Reliable Anomaly Detection for Gas Turbine Combustors: A Deep Learning Approach,» *arXiv:1908.09238 [cs, stat],* 2019.

[46] J. Dai, H. Song, G. Sheng y X. Jiang, «Cleaning Method for Status Monitoring Data of Power Equipment Based on Stacked Denoising Autoencoders,» *IEEE Access,* 2017.

[47] H. Luo y S. Zhong, «Gas turbine engine gas path anomaly detection using deep learning with Gaussian distribution,» de *2017 Prognostics and System Health Management Conference (PHM-Harbin)*, 2017.

[48] L. Banjanovic-Mehmedovic, A. Hajdarevic, M. Kantardzic, F. Mehmedovic y I. Dzananovic, «Neural network-based data-driven modelling of anomaly detection in thermal power plant,» *Automatika,* 2017.

[49] N. Nguyen Thi, V. L. Cao y N.-A. Le-Khac, «One-Class Collective Anomaly Detection Based on LSTM-RNNs,» *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXVI: Special Issue on Data and Security Engineering,* 2017.

[50] M. a. B. A. a. G. J. a. G. H. M. a. M. S. Bahri, "Data stream analysis: Foundations, major tasks and tools," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* p. e1405.

[51] J. a. L. A. a. D. F. a. G. F. a. G. J. a. Z. G. Lu, «Learning under concept drift: A review,» *IEEE Transactions on Knowledge and Data Engineering,* vol. 31, nº 12, pp. 2346--2363, 2018.

[52] G. I. a. H. R. a. C. H. a. N. H. L. a. P. F. Webb, «Characterizing concept drift,» *Data Mining and Knowledge Discovery,* vol. 30, nº 4, pp. 964--994, 2016.

[53] H. a. K. M. a. S. T. S. Hu, «No Free Lunch Theorem for concept drift detection in streaming data classification: A review,» *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* vol. 10, nº 2, p. e1327, 2020.

[54] R. S. M. a. S. S. G. T. C. Barros, «A large-scale comparison of concept drift detectors,» *Information Sciences,* vol. 451, pp. 348--370, 2018.

[55] L. L. a. Y. X. Minku, «DDD: A new ensemble approach for dealing with concept drift,» *IEEE transactions on knowledge and data engineering,* vol. 24, nº 4, pp. 619--633, 2011.

[56] J. L. a. D. S. J. a. L. I. a. B. M. N. a. K. N. Lobo, «Drift detection over non-stationary data streams using evolving spiking neural networks,» de *International symposium on intelligent and distributed computing*, 2018.

[57] R. S. M. a. S. S. G. T. C. Barros, «A large-scale comparison of concept drift detectors,» *Information Sciences,* vol. 451, pp. 348--370, 2018.

[58] J. a. M. P. a. C. G. a. R. P. Gama, «Learning with drift detection,» de *Brazilian symposium on artificial intelligence*, 2004.

[59] M. a. d. C.-Á. J. a. F. R. a. B. A. a. G. R. a. M.-B. R. Baena-García, «Early drift detection method,» de *Fourth international workshop on knowledge discovery from data streams*, 2006.

[60] A. a. G. R. Bifet, «Learning from time-changing data with adaptive windowing,» de *Proceedings of the 2007 SIAM international conference on data mining*, 2007.

[61] K. a. Y. K. Nishida, «Detecting concept drift using statistical testing,» de *International conference on discovery science*, 2007.

[62] E. S. Page, «Continuous inspection schemes,» *Biometrika,* vol. 41, nº 1/2, pp. 100--115, 1954.

[63] S. H. a. M. M. A. Bach, «Paired learners for concept drift,» de *2008 Eighth IEEE International Conference on Data Mining*, 2008.

[64] G. J. a. A. N. M. a. T. D. K. a. H. D. J. Ross, «Exponentially weighted moving average charts for detecting concept drift,» *Pattern recognition letters,* vol. 33, nº 2, pp. 191--198, 2012.

[65] S. Roberts, «Control chart tests based on geometric moving averages,» *Technometrics,* vol. 42, nº 1, pp. 97--101, 2000.

[66] I. a. d. C.-Á. J. a. R.-J. G. a. M.-B. R. a. O.-D. A. a. C.-M. Y. Frias-Blanco, «Online and non-parametric drift detection methods based on Hoeffding's bounds,» *IEEE Transactions on Knowledge and Data Engineering,* vol. 27, nº 3, pp. 810--823, 2014.

[67] A. a. V. H. L. Pesaranghader, «Fast hoeffding drift detection method for evolving data streams,» de *Joint European conference on machine learning and knowledge discovery in databases*, 2016.

[68] W. Hoeffding, «Probability inequalities for sums of bounded random variables,» *Journal of the American Statistical Association,* vol. 58, nº 301, pp. 13--30, 1963.

[69] D. T. J. a. K. Y. S. a. D. G. a. P. R. Huang, «Detecting volatility shift in data streams,» de *2014 IEEE International Conference on Data Mining*, 2014.

[70] R. S. a. C. D. R. a. G. J. P. M. a. S. S. G. Barros, «RDDM: Reactive drift detection method,» *Expert Systems with Applications,* vol. 90, pp. 344--355, 2017.

[71] R. S. M. a. H. J. I. G. a. d. L. C. D. R. de Barros, «Wilcoxon rank sum test drift detector,» *Neurocomputing,* vol. 275, pp. 1954--1963, 2018.

[72] F. Wilcoxon, «Individual comparisons by ranking methods,» de *Breakthroughs in statistics*, Springer, 1992, pp. 196--202.

[73] F. a. d. S. E. M. a. G. J. Pinagè, «A drift detection method based on dynamic classifier selection,» *Data Mining and Knowledge Discovery,* vol. 34, nº 1, pp. 50--74, 2020.

[74] A. a. L. J. a. Z. G. Liu, «Concept Drift Detection via Equal Intensity k-means Space Partitioning,» *IEEE transactions on cybernetics,* 2020.

[75] A. a. V. H. a. P. E. Pesaranghader, «Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams,» *Machine Learning,* vol. 107, nº 11, pp. 1711--1743, 2018.

[76] T. a. K. A. a. d. C. A. A. a. V. M. Escovedo, «DetectA: abrupt concept drift detection in non-stationary environments,» *Applied Soft Computing,* vol. 62, pp. 119--133, 2018.

[77] Q.-H. a. R. H. a. N. K. a. F.-V. P. a. D. T.-L. Duong, «High utility drift detection in quantitative data streams,» *Knowledge-Based Systems,* vol. 157, pp. 34--51, 2018.

[78] S. a. A. A. a. S. S. Micevska, «SDDM: an interpretable statistical concept drift detection method for data streams,» *Journal of Intelligent Information Systems,* pp. 1--26, 2021.

[79] R. F. a. V. Y. a. G. C. H. a. B. A. de Mello, «On learning guarantees to unsupervised concept drift detection on data streams,» *Expert Systems with Applications,* vol. 117, pp. 90--102, 2019.

[80] O. A. a. P. E. a. A. N. a. C. J. Mahdi, «Fast reaction to sudden concept drift in the absence of class labels,» *Applied Sciences,* vol. 10, nº 2, p. 606, 2020.

[81] J. Barr, "New – CloudFormation Drift Detection," 2018. [Online]. Available: https://aws.amazon.com/es/blogs/aws/new-cloudformation-drift-detection/. [Accessed 2021].

[82] J. Ž. I. B. A. P. M. &. B. A. Gama, «A survey on concept drift adaptation,» *ACM computing surveys (CSUR),* vol. 46, nº 4, pp. 1--37, 2014.

[83] M. e. a. Du, «Deeplog: Anomaly detection and diagnosis from system logs through deep learning.,» de *2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[84] M. Du y F. Li., «Spell: Streaming parsing of system event logs,» de *2016 IEEE 16th International Conference on Data Mining (ICDM).* , 2016.

[85] Meng, Weibin y e. al, «LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs,» *IJCAI,* 2019.

[86] Mikolov, Tomas y e. al, «Efficient estimation of word representations in vector space,» *arXiv,* 2013.

[87] Zhang, Shenglin y e. al., «Syslog processing for switch failure diagnosis and prediction in datacenter networks,» de *IEEE/ACM 25th International Symposium on Quality of Service (IWQoS).* , 2017.

[88] Zhang, Xu y e. al, «Robust log-based anomaly detection on unstable log data,» de *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.

[89] P. He y e. al., «Drain: An online log parsing approach with fixed depth tree,» de *2017 IEEE international conference on web services (ICWS).* , 2017 .

[90] S. Huang y e. al., «HitAnomaly: Hierarchical Transformers for Anomaly Detection in System Log,» *IEEE Transactions on Network and Service Management 17.4 (2020): ,* 2020.

[91] A. Vaswani y e. al., «Attention is all you need,» *Advances in neural information processing systems,* 2017.

[92] V.-H. Le y H. Zhang, «Log-based Anomaly Detection Without Log Parsing,» *arXiv,* 2021.

[93] Devlin, Jacob y e. al., «Bert: Pre-training of deep bidirectional transformers for language understanding,» *arXiv 1810.04805,* 2018.

[94] Guo, Haixuan, S. Yuan y X. Wu, «"LogBERT: Log Anomaly Detection via BERT,» *arXiv ,* 2021.

[95] H. Ott y e. al., «Robust and Transferable Anomaly Detection in Log Data using Pre-Trained Language Models,» *arXiv ,* 2021.

[96] W. Xu y e. al, «Detecting large-scale system problems by mining console logs,» de *ACM SIGOPS 22nd symposium on Operating systems principles*, 2009.

[97] "VAST Challenge 2011. 2011. MC2 - Computer Networking Operations," 2011. [Online]. Available:

https://www.cs.umd.edu/hcil/varepository/VAST%20Challenge%202011/challenges/MC2%20-%20Computer%20Networking%20Operations/. [Accessed 2021].

[98] K. Veeramachaneni y e. al, «AI^ 2: training a big data machine to defend,» de *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and Security (IDS).* , 2016 .

[99] A. Tuor y e. al, «Deep learning for unsupervised insider threat detection in structured cybersecurity data streams.,» de *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence. 2017.*, 2017.

[100] J. Glasser y B. Lindauer, «Bridging the gap: A pragmatic approach to generating insider threat data,» de *2013 IEEE Security and Privacy Workshops. IEEE*, 2013.

[101] L. F. a. D. A. T. E. Di Nitto, «Autonomic Decentralized Microservices: The Gru Approach and Its Evaluation,» *Microservices: Science and Engineering,* 2020.

[102] L. F. Maimó, Á. L. P. Gómez, F. J. G. Clemente, M. G. Pérez y G. M. Pérez, «A Self-Adaptive Deep Learning-Based System for Anomaly Detection in 5G Networks,» *IEEE Access,* 2018.

[103] Y. Yang, X. Zheng, W. Guo, X. Liu y V. Chang, «Privacy-preserving smart IoT-based healthcare big data storage and self-adaptive access control system,» *Information Sciences,* 2019.

[104] A. Alhosban, K. Hashmi, Z. Malik y B. Medjahed, «Self-healing framework for Cloud-based services,» de *ACS International Conference on Computer Systems and Applications (AICCSA)*, 2013.

[105] M. Azaiez y W. Chainbi, «A Multi-agent System Architecture for Self-Healing Cloud Infrastructure,» de *International Conference on Internet of things and Cloud Computing*, New York, 2016.

[106] S. S. Gill, I. Chana, M. Singh y R. Buyya, «RADAR: Self-configuring and self-healing in resource management for enhancing quality of cloud services,» *Concurrency and Computation: Practice and Experience,* 2019.

[107] W. Li, P. Zhang y Z. Yang, «A Framework for Self-Healing Service Compositions in Cloud Computing Environments,» de *IEEE 19th International Conference on Web Services*, 2012.

[108] J. P. Magalhães y L. M. Silva, «A Framework for Self-Healing and Self-Adaptation of Cloud-Hosted Web-Based Applications,» de *IEEE 5th International Conference on Cloud Computing Technology and Science*, 2013.

[109] J. P. Magalhães y L. M. Silva, «SHõWA: A Self-Healing Framework for Web-Based Applications,» de *ACM Transactions on Autonomous and Adaptive Systems*, 2015.

[110] P. K. Rajput y G. Sikka, «Multi-agent architecture for fault recovery in self-healing systems,» *J Ambient Intell Human Compu,* 2021.

[111] E. Rios, E. Iturbe y M. C. Palacios, «Self-healing Multi-Cloud Application Modelling,» de *12th International Conference on Availability, Reliability and Security*, New York, 2017.

[112] A. Mosallanejad, R. Atan, M. A. Murad y R. Abdullah, «A hierarchical self-healing SLA for cloud computing,» de *International Journal of Digital Information and Wireless Communications (IJDIWC)*, 2014.

[113] T. Wang, J. Xu, W. Zhang, Z. Gu y H. Zhong, «Self-adaptive cloud monitoring with online anomaly detection,» de *Future Generation Computer Systems*, 2018.

[114] C. Schneider, A. Barker y S. Dobson, «A survey of self-healing systems frameworks,» *Software: Practice and Experience,* vol. 45, 2015.

[115] H. Psaier y S. Dustdar, «A survey on self-healing systems: approaches and systems,» *Computing,* vol. 91, 2011.

[116] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao y V. Stankovski, «Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review,» *Journal of Systems and Software,* 2018.

[117] N. Esfahani y S. Malek, «Uncertainty in Self-Adaptive Software Systems,» *Software Engineering for Self-Adaptive Systems II,* 2013.

[118] D. Weyns, «Software Engineering of Self-Adaptive Systems: An Organised Tour and Future Challenges,» *Handbook of Software Engineering,* 2019.

[119] A. Bifet, «Classifier Concept Drift Detection and the Illusion of Progress,» de *International Conference on Artificial Intelligence and Soft Computing*, 2017.

[120] I. a. B. A. a. R. J. a. P. B. a. H. G. Zliobaite, «Evaluation methods and decision theory for classification of streaming data with temporal dependence,» *Machine Learning,* vol. 98, nº 3, pp. 455--482, 2015.

[121] M. a. N. I. V. a. o. Basseville, Detection of abrupt changes: theory and application, vol. 104, Prentice hall Englewood Cliffs, 1993.

[122] J.-i. a. Y. K. Takeuchi, «A unifying framework for detecting outliers and change points from time series,» *IEEE transactions on Knowledge and Data Engineering,* vol. 18, nº 4, pp. 482--492, 2006.

[123] F. a. G. F. Gustafsson, Adaptive Filtering and Change Detection, vol. 1, Citeseer, 2000.

[124] A. a. R. J. a. Z. I. a. P. B. a. H. G. Bifet, «Pitfalls in benchmarking data stream classification and how to avoid them,» de *Joint European conference on machine learning and knowledge discovery in databases*, 2013.

[125] I. Zliobaite, «How good is the electricity benchmark for evaluating concept drift adaptation,» *arXiv preprint,* nº arXiv:1301.3524, 2013.