PIACERE

Deliverable D6.2

PIACERE run-time monitoring and self-learning, self-healing platform - v2

Editor(s):	Gorka Benguria
Responsible Partner:	TECNALIA
Status-Version:	Final v1.0
Date:	02.12.2022
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	30.11.2022

Workpackage responsible for the Deliverable:	WP6 - Monitor plan and self-heal runtime of Infrastructure as Code
Editor(s):	Gorka Benguria, Fundación Tecnalia Research & Innovation
Contributor(s):	Tecnalia (Gorka Benguria, Jesus Lopez, Iñaki Etxaniz), Ericsson (Cosimo Zotti), Polimi (Bin Xiang), XLAB (Ales Cernivec, Tomaz Martincic, Alvaro Garcia Faura), 7bulls (Radosław Piliszek, Marcin Bartmański)
Reviewer(s):	Lorenzo Blasi, HPE
Approved by:	All Partners
Recommended/mandatory readers:	Recommended WP2, WP5, WP7

Abstract:	These deliverables will contain the main outcomes from M12 to M24 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	13.10.2022	First TOC and sections assignment	TECNALIA
v0.2	19.10.2022	Comments and suggestions received by consortium partners	TECNALIA
V0.3	26.10.2022	Contributions round 1	7BULLS, ERICSSON, POLIMI, XLAB, TECNALIA
V0.4	09.11.2022	Contributions round 2	7BULLS, ERICSSON, POLIMI, XLAB, TECNALIA
V1.0	17.11.2022	Final Editing	TECNALIA
V1.0	28.11.2022	review	HPE
V1.0	02.12.2022	Final version	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
ICG	Infrastructure as Code Generator
IDE	Integrated Development Environment
IDF	Inverse Document Frequency
IDS	Intrusion Detection System
IEC	Infrastructure Elements Catalogue
IEM	IaC execution Manager
IEP	IaC execution Platform
IOP	Infrastructure Optimizer Platform
IPS	Intrusion Prevention System
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 D6.2. The whole D6.2 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 second 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 on 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.

This version of the document is a follow up version of the D6.1 – PIACERE run-time monitoring. It extends the previous version with the aspects evolved during the second year of development. Besides it includes a changelog version to understand the evolution during this second year.

Next version (D6.3) 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 second version (M24) 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. Besides, as it is a follow up version of the document it also covers the evolution with respect to the previous version.

The overall objective in this period has been to start the piloting of the features regarding the performance and security monitoring, self-learning and self-healing components.

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 threefold:

- 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.
- To describe the evolution with respect to the previous version.

1.2 Document structure

The document is structured into six parts. Section 2 explains the components covered by the document and their relationships. Section 3 summarises the evolutions introduced during the period from different perspectives: overall approach, innovation, and technical. The next part 4 addresses the state of the art that supports the technical decisions taken to develop the different components covered in this deliverable.

The fourth part 5 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 spans from section to section. The requirements include project level requirements that comes from the use cases, and the component internal requirements that are internally established with the upcoming integration in mind.

The fifth part, section 6, provides the conclusions.

Besides there is a final part Annex A that addresses the delivery and usage of each component.

The implementation and the delivery parts, have been designed to be used in isolation by the developers, without requiring them to read 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 shown 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 deviations, 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.

In this new version, we also implemented several links from IDE to the different monitoring components to provide the users with an easy way to access the different monitoring components in an easy and integrated way.

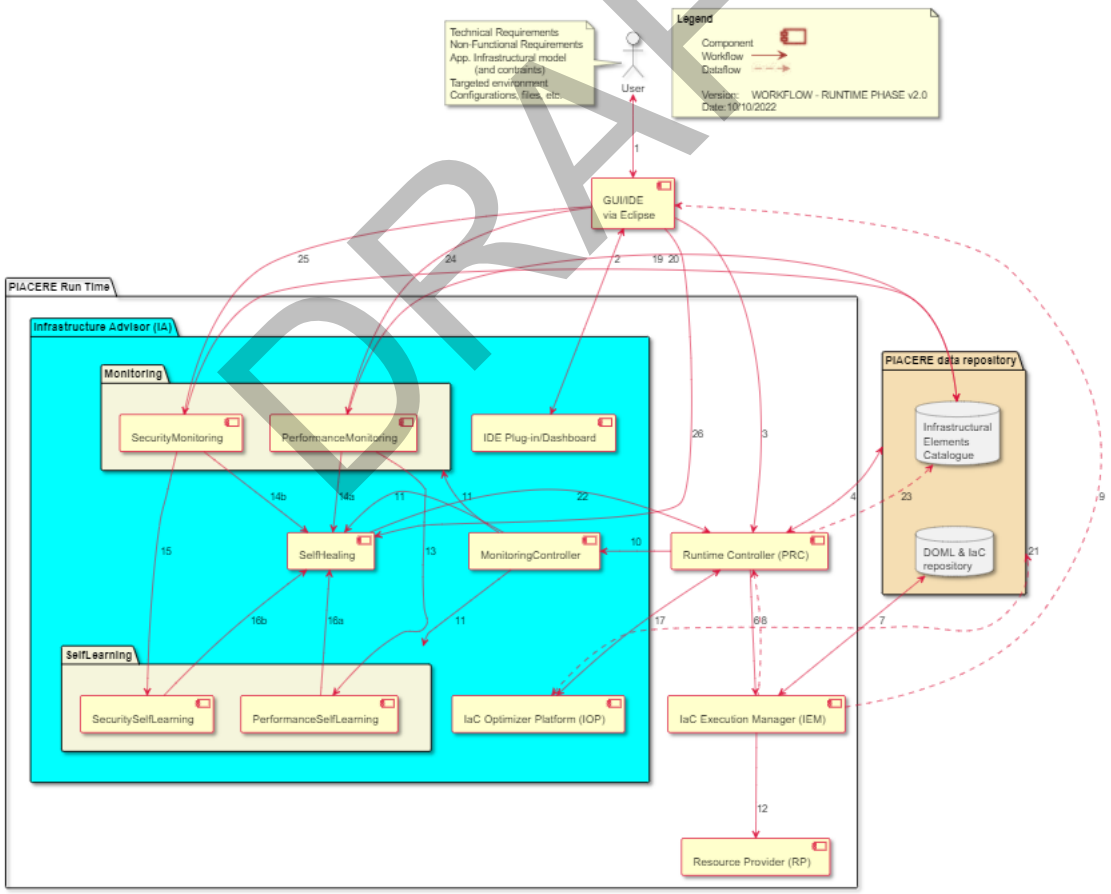

Figure 1: PIACERE Runtime Diagram on its 2.0 version.

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 controlling 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 controlling 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.

The other components shown in the Runtime Diagram (Figure 1) are documented in the following deliverables: IDE in D3.8, IaC Optimizer Platform (IOP) and Infrastructural Element Catalogue (IEC) in D5.8, Runtime Controller (PRC) and IaC Execution Manager (IEM) in D5.2.

3 Changes in v2: M12 – M24 Changelog

This section explains the evolution from the previous release M12 to the current release M24. It is structured in separate sections to describe with more detail the evolution in specific areas covered by the D6.2.

- Monitoring Controller
- Performance Monitoring
- Security monitoring
- Performance Self learning
- Security Self learning
- Self-Healing

With respect to the use case requirements coverage, we show the advance M12 to M24 in the following figures Figure 2 and Figure 3. The advance is described focusing of the different PIACERE Key Results (KR) covered in this document. KRs package the outcomes of the WP6 in exploitable results. These are:

- KR11 - PIACERE Self-learning and self-healing mechanisms
- KR12 - Runtime security monitoring

During Year 1 the advance was focused in providing the building blocks to enable the development of more complex features during the next period, the overall progress and requirements coverage is shown in Figure 2.

KR#	Progress	Requirements involved
KR 11 – PIACERE Self-learning and self-healing mechanisms		REQ11, REQ16, REQ17, REQ46, REQ47, REQ50, REQ51, REQ52, REQ72, REQ92, REQ93, REQ94, REQ97
KR12 - Runtime security monitoring		REQ16, REQ17

Figure 2: Requirements coverage at M12.

In Year 2 we advanced in the coverage of the requirements focussing on the direct usage by the targeted scenarios. The overall progress and requirements coverage is shown in Figure 3.

KR#	Progress	Requirements involved
KR 11 – PIACERE Self-learning and self-healing mechanisms		REQ11, REQ16, REQ17, REQ46, REQ47, REQ50, REQ51, REQ52, REQ72, REQ92, REQ93, REQ94, REQ97
KR12 - Runtime security monitoring		REQ16, REQ17

Figure 3: Requirements coverage M24.

3.1 Monitoring controller

This is a utility component introduced in M12 to provide a single point of control of the monitoring infrastructure to other components in the piacere framework such as the piacere runtime controller (PRC).

3.1.1 Overall evolution

The main activities performed during this period focused on the integration with other components of the PIACERE platform and the implementation of the end-to-end user scenarios.

Table 1: Evolution of the Monitoring Controller component.

M12	M24
Definition of interfaces of that will proxy the monitoring platform with the rest of elements of the piacere infrastructure, covering: activation and deactivation of monitoring for specific deployments.	Implementation and testing in pilots of the activation and deactivation functionalities towards required components in the internal monitoring infrastructure: ▪ Performance monitoring ▪ Performance self-learning ▪ Security monitoring

3.1.2 Innovation aspects

As this component is utility component that acts as proxy between the piacere platform and the monitoring component we will only address the overall evolution in the period. There are no innovation aspects directly covered by it.

3.1.3 Technical features

It includes REST clients to manage relevant component in the internal monitoring architecture: performance monitoring, performance self-learning and security monitoring.

The implementation of the clients relay in the standard definition of the API of those components using OpenAPI Specification (OAS). That enables an easier implementation of changes based on the expected evolution in those components.

3.1.4 Changes coming from use cases

No major changes from requests from the use cases have been made.

3.2 Performance monitoring

3.2.1 Overall evolution

The main activities performed during this period focused on the integration with other components of the PIACERE platform and the implementation of the end-to-end user scenarios.

Table 2: Evolution of the Performance monitoring component.

M12	M24
Deployment of base monitoring elements to enable the development of other components. Including docker based agents to continuously feed the infrastructure with data. Those agents were not suitable to be deployed on use case infrastructure elements not including docker technology.	Creation of performance monitoring agent configuration to be deployed as part of the infrastructure as code generated by the infrastructure as code generator (ICG). Deployment of multiple testing agents in a permanent way to produce the necessary information to develop other components in the PIACERE monitoring stack such as those related with the self-learning and self-healing.
Definition of interfaces of the different elements of the monitoring infrastructure. In this period the integration with IDE was not considered.	Implementation of interfaces for the integration of the monitoring platform with the integrated development environment (IDE). The integration is focussed on the facilitation of the access to the different development related dashboards from the IDE.
Prototypical performance dashboard was included. In this period only performance dashboards were envisioned.	Implementation of deployment related functionalities at Grafana side, this has required the creation of a controller services that customize the Grafana based on the deployment information. Development of different dashboards for performance and self-learning metrics.
The integration of components was mainly focused internally on the monitoring scope. ▪ Data gathering ▪ Self-learning ▪ Self-healing	Finally, we have collaborated closely with other components in the PIACERE architecture to achieve the successful end-to-end execution of a deployment including the monitoring agents and the configuration of the monitoring platform. Main components involved in this collaboration have been: ▪ PIACERE runtime controller (PRC) part of KR13 ▪ Infrastructure as code execution manager (IEM) KR10 ▪ Integrated Development environment (IDE) KR2

3.2.2 Innovation aspects

No major changes in the innovation aspects have been introduced. We continue in the achievement of the planned innovation:

- To externally monitor heterogeneous types of resources (Cloud services and IoT) and diverse metrics (performance, cost, availability, security) in a continuous and integrated manner with the defined deployment model.

3.2.3 Technical features

Performance monitoring controller has been developed. It covers two main aspects: deployment configuration and provisioning of deployment related Grafana dashboards information to the IDE. The Figure 4 shows the main features implemented in the performance monitoring controller. The API and the details of its usage can be accessed in the integration environment (https://git.code.tecnalia.com/piacere/public/the-platform/runtime-monitoring/pm/-/tree/y2/git/pmc/openapi.yaml)

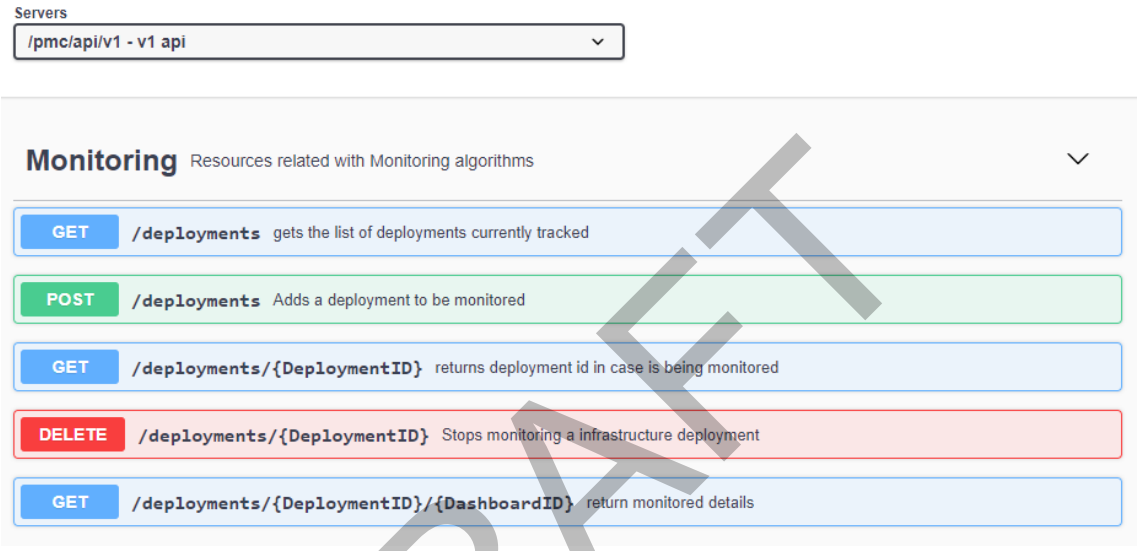

Figure 4: Performance monitoring controller.

The POST and DELETE methods are related with the creation and removal of deployment workflows and are related with the PRC integration. While the GET methods are focused on the retrieval of information about the deployment related dashboard by the IDE.

Performance monitoring agent has been developed. It covers the generation of the infrastructure as code related with the necessary collection of monitoring data at elements configured as part of the deployments created. The agent has been developed using Ansible and available open-source components when possible. The agents are placed outside from the piacere framework main deployment as they run in the infrastructure elements created by the IAC (Infrastructure as code) created by piacere.

3.2.4 Changes coming from the use cases

The experimentation with the use cases have been useful to:

- Adjust the visibility of the agent related code in order to facilitate the usage by the scenarios
- Adjust the agent variables and their management to match the stage based IaC procedure implemented in the PIACERE Project.
- Request the modification of the log information on the IEM to be able to analyze the progress of the configuration of the agents.
- Request the modification of the PRC to get the deployment id necessary to setup the performance and self-learning dashboards.

3.3 Security monitoring

3.3.1 Overall evolution

The main activities performed during the last period were focused on monitoring, self-learning components and adapting the to the security non-functional requirements.

Table 3: Evolution of the Security monitoring component.

M12	M24
We have specified a central interface for the security monitoring component. This interface takes care of the configuration of the monitoring of the deployments, similar to those developed in T6.1. We have implemented the stub of the interface, we have dockerized it and orchestrate with a docker-compose.	A new version of security controller has been developed by creating a new release of OpenAPI specification with corresponding business logics on the back-end services. Basic security mechanisms were integrated within the controllers (basic auth, communication over TLS, secure connections between internal components), thus making the security monitoring components more secure.
Security monitoring deployed it in the project CI platform.	Security monitoring facility has been integrated with the existing PIACERE CI process.
We have implemented ansible playbooks for the agent in order to facilitate their deployment and configuration at pilot scenarios. We have deployed agents to collect data related to security to be able to develop and test the security related monitoring and self-learning components.	An ansible playbook to deploy the security agent of the security monitoring following the agreed WP6 approach to deploy monitoring agents with the minimal configuration The agent has been extended with additional configuration supporting PIACERE deployments and already with the notion of multi-tenancy (or multi-project support).
	Contribution to the DOML definitions w.r.t. basic strategies for supporting security related ansible based self-healing processes have been made. These discussions are still in progress, we can expect implementation of the strategies in the next reporting period.
	Integration with the IDE (redirect to the Security Self-Learning dashboards within a project).

3.3.2 Innovation aspects

Innovations during the past period were focused on 2 main aspects:

- The dynamic notion of security monitoring integrated within the complete CI/CD. Besides the static analysis being done within WP4 we are capable to obtain dynamic

analysis with the use of PIACERE framework. This analysis evaluates infrastructure and provides a set of security issues detected (CVEs, severities detected, problems that can be resolved with hardening and also anomalies detected in the runtime).

- Machine learning (ML) based anomaly detection using Masked Language Modelling and Hypersphere Volume Minimization has been introduced and integrated. Additionally, the presented approach embedded into the whole DevOps lifecycle increases the innovation part of the ML-based anomaly detection.

3.3.3 Technical features

New version of the OpenAPI specifications. In the latest versions of the Security Monitoring Controller, resources related to the Security Self-learning have been slightly updated in order to adapt to responses from the self-learning API. Additionally, DELETE method is now supported on the Deployment resource in order to support removal of deployments (or projects).

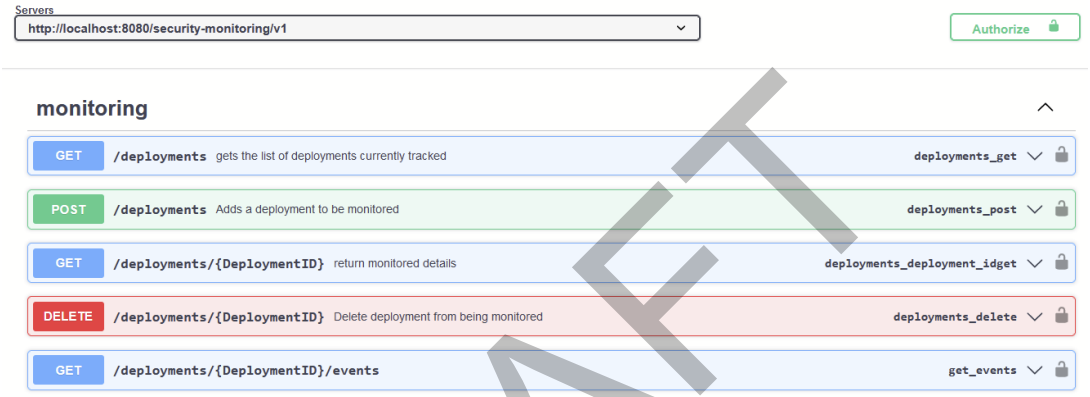

Figure 5: Security Monitoring Controller's API.

The Figure 5 above presents introduction of "deployments" and DELETE method for the resource. Complete API specification can be found on official PIACERE public repository[1].

Ansible playbook to deploy security agents and updates to the agents. In the public repositories[2] there are Ansible playbooks for deploying agents in Docker and “bare-metal” environments.

Security of the components. More configuration options are now supported with additional user names and passwords for the deployment of the core security monitoring components. Specifically, ElasticSearch can be protected by using basic authentication credentials (also for securing communication with self-learning components). Documentation on these details can be found in the README.md file of the official PIACERE public repository[3].

3.3.4 Changes coming from the use cases

No major changes stemming from the use cases have been made.

[1] https://git.code.tecnalia.com/piacere/public/the-platform/runtime-security-monitoring/security-monitoring-controller/-/blob/main/swagger_server/swagger/swagger.yaml - OpenAPI specification of the Security Monitoring Controller.

[2] https://git.code.tecnalia.com/piacere/public/agents/sma-playbook – public repository of the security monitoring agents

[3] https://git.code.tecnalia.com/piacere/public/the-platform/runtime-security-monitoring/security-monitoring-deployment – repository and documentation of the Security Monitoring stack (ELK, Wazuh Manager and Security Monitoring Controller)

3.4 Performance Self-learning

3.4.1 Overall evolution

The main activities performed during this period has been focused on the coverage of additional metrics to complete the planned metrics to be adressed. We have mainly evolved the component in these lines:

Table 4: Evolution of the Self-Learning component.

M12	M24
We had developed all the metrics but only **CPU** ones were tested. Concretely: ▪ CPU metrics (usage_idle, usage_user, usage_system) were developed, integrated and tested. ▪ MEMORY (free, used_percent) and DISK (free, used_percent) metrics were only developed and integrated	Now all metrics (CPU, MEMORY and DISK ones) are developed, integrated and successfully **tested**
Concept drift detection was not developed, integrated and tested in any metrics	Now all metrics count on a concept drift detector; it has been developed, integrated and successfully tested
A monitoring data point is received every **1 hour.**	In order to be able to predict in time a decrease in the performance, and take the corresponding actions (self-healing), we have changed the frequency to have a datapoint every **10 minutes**.

3.4.2 Innovation aspects

We have not added more innovation aspects. They were included at the beginning of the project, and now it is time to take them to the reality. They continue to be the same:

- To develop improved predictive models to detect and predict the non-compliance on NFRs (runtime- and security-related) with integrated anomaly and concept drift techniques.

3.4.3 Technical features

We have mainly introduced the following technical modifications:

- Now the technique to develop the incremental learning algorithm is based on SNARIMAX[4] with a regressor based on KNN.

  The main reason for this change is that SNARIMAX is tailored to time series forecasting (it adapts to several horizons of prediction), while Random Forest is only useful is case of nowcasting (horizon=1). As we may try to make predictions for several horizons, this technique scales better in case of this need becomes a reality.

[4] https://riverml.xyz/0.14.0/api/time-series/SNARIMAX/

Previously The technique to develop the **incremental learning** algorithm was Random Forest.

- Now the technique to develop the outlier detector algorithm uses the range

*[mean-s*std : mean+s*std]*

The main reasons for this change are:

- It is not based on machine learning, but on statistics, so it is faster in processing
- It perfectly detects the outliers present in PIACERE, more than the previous technique
- It is more transparent and understandable

3.4.4 Changes coming from the use cases

The uses cases have not changed the nature of the component.

3.5 Security Self-learning

3.5.1 Overall evolution

The main activities performed during this period have been in the implementation of the training and inference process. Based on the basic security requirements, extensions to the communication layer of the security self-learning components have been made in order to make the platform more secure. Additionally, a specific model has been chosen and implemented for the training and for the inference process. Three different dashboards have been implemented and are made available for the use within PIACERE pilots.

Table 5: Evolution of the security self-Learning component.

M12	M24
We have specified a central interface for the security self-learning interactions. The interface for the self-learning functionalities have been integrated with the security monitoring controller. We have implemented the stub of the interface, we have dockerized it and orchestrate with a docker-compose.	Based on the basic security requirements, extensions to the communication layer of the security self-learning components have been made in order to make the platform more secure. Additionally, a specific model has been chosen and implemented for the training and for the inference process. Three different dashboards have been implemented and are made available for the use within PIACERE pilots.
	Integration with the IDE (redirect to the Security Self-Learning dashboards within a project).

3.5.2 Innovation aspects

LogBERT model is implemented as it is described in section 4.2.2.2.

For training process GPU-based systems are being used. Moreover, for the inference process, GPU and CPU-based systems are supported. GPU and CPU workers can be deployed on different hosts.

Different dashboards have been developed:

- Dashboard for log and anomaly scores inspection is made available.
- Dashboards for parsed log template inspection has been made available.
- Dashboards for training log parser and anomaly detection model

3.5.3 Technical features

Integration tests have been implemented. Moreover, GitLab CI pipelines for building docker images and python packages have been established.

Support for basic authentication towards external Elasticsearch instance has been made.

3.5.4 Changes coming from use cases

No major changes stemming from the use cases have been made.

3.6 Self-Healing

3.6.1 Overall evolution

The main activities performed during this period have been focused on the implementation of initial strategies, the identification of possible additional strategies and the integration of the self-healing from the user perspective.

Table 6: Evolution of the self-healing component.

M12	M24
We have implemented a first version the self-healing component where we manage different elements: notifications from the other monitoring components, types of notifications, strategies that can be applied and relationships between types of notifications and strategies. We have implemented the stub of the interface covering notification aspects.	We have implemented the integration of the self-healing with the PRC (PIACERE runtime controller) to be able to request the removal and creation of deployments. We have also collaborated in the implementation of the usage of the notification interface by other monitoring components: security monitoring, performance self-learning and security self-learning.
We have implemented a front end for the self-heling to facilitate the understanding of its configuration and the visualization of his activity.	The front end has been integrated in the IDE and it is now linked to the deployment.
We have identified an initial set of strategies: reboot, redeploy, vertical scalability and horizontal scalability.	We have refactored the vertical and horizontal scaling strategies to reduce them to one single scaling strategy. That based on the DOML will take de decision of scaling horizontal or vertically. Based on the discussions with security monitoring we have found the need to implement additional strategies: ansible, quarantine and user warning.

	Some of the identified strategies require changes in doml that are being discussed.

The main activities corresponding to the achievement of this objective have been:

- We have refactored the vertical and horizontal scaling strategies to reduce them to one single scaling strategy. Based on the DOML the self-healing component will take the decision of scaling horizontal or vertically.
- Based on the discussions with security monitoring developer partner we have found the need to implement two new strategies: quarantine and user warning.

3.6.2 Innovation aspects

No major changes in the innovation aspects have been introduced. They continue to be the same:

- To provide the means to set up self-healed IaC at runtime with minimal manual intervention

3.6.3 Technical features

New strategies have been introduced as part of the identification of potential self-healing strategies to be applied:

- Vertical scaling: this will be applied for infrastructure elements with no scale capability known that are suffering performance degradation. It will be also a possibility for those elements with scale capability.
- Horizontal Scaling: this will be applied for infrastructure elements with scale capability.
- Quarantine: this will stop the infrastructure element, when it becomes untrusty from a security perspective. This may involve the detention of the whole deployment.
- User warning: this will warn the user of some non-critical security hazards.

The self-healing user interfaces have been integrated in the IDE, now it is possible to access the self-healing logs from the IDE.

3.6.4 Changes coming from the use cases

No major changes stemming from the use cases have been made.

4 State of the Art

4.1 Infrastructural elements monitoring

4.1.1 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 or Ganglia 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.

During the current period a new initiative has gain some momentum. This initiative is Gaia-X [5] it is an ecosystem to define a next generation of data infrastructure. Among their service there is one centred in the monitoring named Continuous Automated Monitoring CAM. The purpose of CAM is to support means to transparently evaluate the compliance of the individual services offered at Gaia-X. It focusses on the compliance with some requirements and rules imposed by Gaia-X on the system.

4.1.1.1 Performance Monitoring

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.

[5] https://www.data-infrastructure.eu/GAIAX/Navigation/EN/Home/home.html

- 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.
- 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[6] 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:

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

- 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).
- 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.

4.1.1.2 Security Monitoring

Existing security monitoring solutions implement functionalities such as: sensor, parser, integrator, detector, inspector and actuator [9]. 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 , tcpdump , tshark ; traffic sessions capture, such as netflow ; 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 5.3.2) and are built on top of open-source solutions/modules.

OSSEC (Open Source HIDS SECurity) [10] 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) [11] 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 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 [12] 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 6. 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.

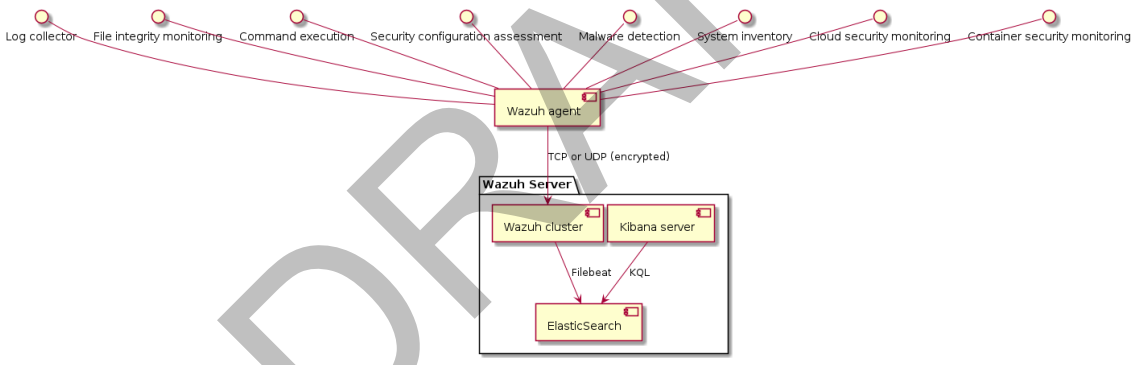

Figure 6: High-level Wazuh's architecture.

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 4.2.2.2.

4.1.2 Infrastructural elements monitoring approach in PIACERE

4.1.2.1 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 have deployed agents in the deployed infrastructures to gather performance metrics and send them back to the PIACERE Platform. We have used Telegraf [13] open-source tool to gather that information. The agents will be deployed as integral part of the application deployment in this sense the ICG generates 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 have deployed additional elements to store the data, process it, and dispose it for use by other components. For the deployment of these elements we considered 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 have included: 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 have used influxdb[7] 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 have used Grafana[8]. That is also an open-source platform. It provides a responsive web-based user interface with a backend that takes 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 have defined a PIACERE-oriented REST API, and a Java based server to implement the logic for that REST API.

4.1.2.2 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 dynamic perspective – using Network Security Monitoring (NSM) tools [9]. The monitoring system is 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

[7] https://www.influxdata.com/
[8] https://grafana.com/

monitoring tools (Wazuh, Filebeat, ELK-stack and newly developed components: anomaly detection tool based on LogBert algorithm and Security Monitoring Contoller) in PIACERE we are focusing primarily on the detection and secondary on the response and protection (through the self-learning and self-healing process). With the Wazuh (agents and a manager) and Filebeat we are capable to detect real-time security-related metrics and aggregate logs for the purpose of anomaly detection. Using dedicated ELK-stack based on open-source distribution components we are storing newly detected events and logs in Elasticsearch (open-source distribution) in different – separate indexes. Anomaly detection based on log date is capable to 1) train the model using the already provided logs from the dedicated index and 2) detect anomalies in real time based using a specific already pre-trained model. We have extended Wazuh modules to include PIACERE specific labels so that we are capable of filtering the data and events based on these labels in the data itself. Kibana is used to depict security-related events. Security Monitoring Controller is capable of monitoring the events and triggers actions based on the severity level of these detected events. These actions are, for example, webhook calls to external services (e.g., Self Healing PIACERE component). Security Monitoring Controller is equipped also with a process capable of detecting Wazuh's and Anomaly detection logs.

4.2 Self-learning

4.2.1 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. 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 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 changes or anomalies 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 7 shows.

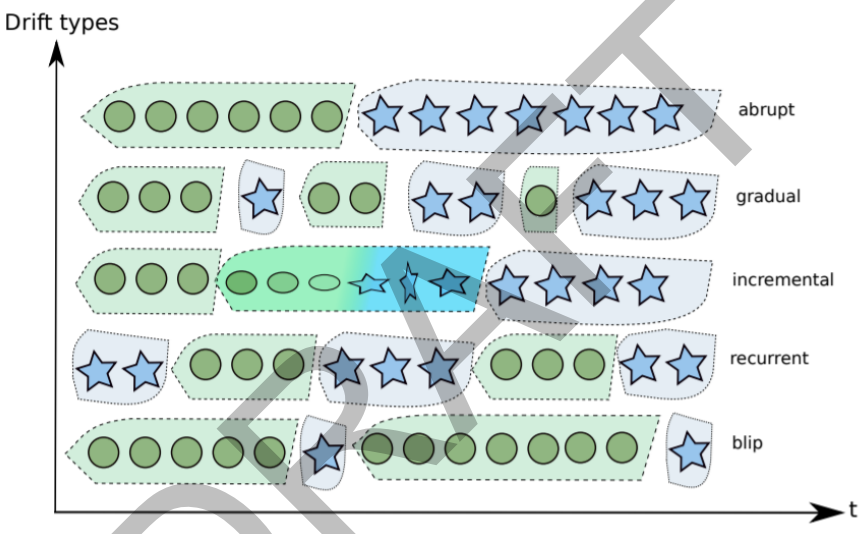

Figure 7: Types of drift according to severity and speed of changes, and noisy blips. Here the stars and circles represent the prevailing concept at every time instant [23].

4.2.1.1 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 preallocated 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. But, in the case of PIACERE, our prediction problem is closer to nowcasting (where the prediction corresponds to the next time step, and thus the problem is closer to the online learning field) than to the classic time series forecasting problem where the temporal dependence is more relevant.

4.2.1.2 Performance 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 [25] 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 [26].

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 [27]: An extension of the support vector machine standard algorithm [28].
- Local Outlier Factor [29]: Algorithm that instead of performing a binary classification estimates the probability of an instance being anomalous.
- Isolation Forest [30]: A binary tree based search that tries to isolate anomalies. An online version called Half Space Trees [31] also exists.
- Elliptic Envelope [32]: An algorithm based on the minimum variance determinant [33] 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 [34] [35]. 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) [36] 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) [37] is a class of deep neural networks, that act as a feature detector by reconstructing probabilistically its inputs.
- Long short-term memory (LSTM) [38] [39] 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 [40] 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) [41] [42] [43] 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) [44] [45] [46] 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) [47] [48] 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 [37] [38].

4.2.1.3 Security Anomaly Detection

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.

4.2.1.3.1 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.

4.2.1.3.2 Anomaly detection in logs

DeepLog (2017)

One of the most relevant works on the application of deep learning to log analysis is DeepLog [49]. 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 [50] 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 [51] 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 [52] 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) [53] 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 [54] 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 [55] 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 [56] is the first work that leverages the Transformer [57] 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 [55] 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 [58] 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 [59] model.

LogBERT (2021)

LogBERT [60] is the first work leveraging the transformer architecture working in a self-supervised fashion (i.e. training with 'normal' logs only). Drain [55] 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. [61], 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.

4.2.1.3.3 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 [62]) or distributed systems (HDFS [62] and Openstack [49]). None of them was specifically validated in security data. Only DeepLog [49] provided results for the VAST Challenge 2011 [63] - 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, [64] 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 [65]. Specifically, common DNN (Deep Neural Networks) and LSTM architectures were employed for the CERT Insider Threat Dataset [66]. 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.

4.2.1.4 "Concept drift" detection

The data generation process in 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 [67]. 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 [68], and this has been the case since the last 3 years. The complexity in *concept drift* manifests when we try to characterize it [69]. We can find many different types of concept drifts (see Figure 7) 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 [70]. 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 [70]. This challenge to find a universal best solution becomes evident in the most recent comparative among drift detectors carried out by [71]. 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 8) 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 [72] or structural changes stemming from the model itself [73].

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.

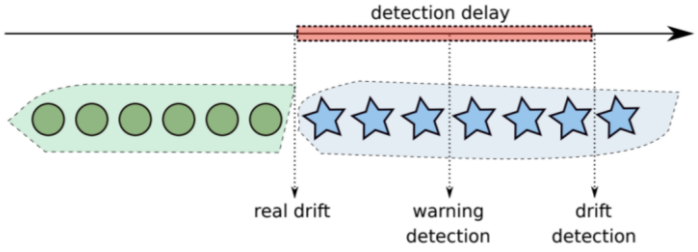

Figure 8: Drift detection example [23].

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 [74], 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 [75]: 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 [76]: 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 [77]: 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.
- PHT [78]: 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.
- HDDM [79]: 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.

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: [80], [81], [82], [83], [84], [85], [86], [87]. Despite they show less relevance than the upper ones, we consider them remarkable for being recent or being published in reputed journals or conferences.

4.2.1.4.1 Drift meaning in monitoring platforms

Looking for this phenomenon in monitoring platforms, we see how it is already a relevant problem to deal with. Despite in PIACERE we pursue the drift detection and the adaptation of our prediction algorithms for monitoring variables, it is worth mentioning the existence of this phenomenon in other scope of infrastructure.

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*[9] 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 *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.

[9] https://driftctl.com/

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 [88], 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 your 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 explored in monitoring platforms from different perspectives (configuration drift, infrastructural drift). In PIACERE we will adopt the classical view explained in section 4.2.1.4.

4.2.2 Self-learning approach in PIACERE

4.2.2.1 Performance and Availability Self-learning

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

The Self-learning component focuses on incrementally online learning and predicting the performance and the availability of the system to guarantee constant high-level performance. The performance information is provided by 7 monitoring variables:

- CPU:
  - usage_idle: % of cpu that is not being used by any program
  - usage_system: % of time the processor spends in running the operating system(i.e., kernal) functions connected to your application
  - usage_user: % of work handled by a cpu
- DISK:
  - free: % of disk that is not being used
  - used_percent: % of disk that is being used
- MEMORY:
  - free: % of memory that is not being used
  - used_percent: % of memory that is being used

At this stage of the project, the prediction for the availability is still pending, and the formula for the availability calculus itself, and it will be part of the design and implementation tasks for the next period and reported in D6.3.

These metrics give us a good idea of the "health" of the system. Once their predictions exceed a threshold, 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 on each event in the next subsections.

4.2.2.1.1 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 monitoring data point every 10 minutes with the value for *CPU, DISK and MEM* monitoring variables in that moment.

The processing has been divided into two stages: the first one counts on a reservoir of batch instances to tune the parameters of the machine learning technique and train the algorithm, and the second one to run on a "real-time" mode where the performance of the system is predicted every time new data arrive. Then, as incremental learner and predictor we have used a SNARIMAX[10] model, which stands for (S)easonal (N)on-linear (A)uto(R)egressive (I)ntegrated (M)oving-(A)verage with e(X)ogenous inputs model, and that is able to incrementally learn and predict every time a new instance arrives. By following the *test-then-train* scheme used in many online learning approaches [89], we have used Mean Absolute Error (MAE) as performance metric for regression problems. Figure 9 shows how the algorithm is able to successfully predict the next *CPU usage idle* data point 10 minutes ahead. The algorithm exhibits a very good predictive performance (see Figure 10).

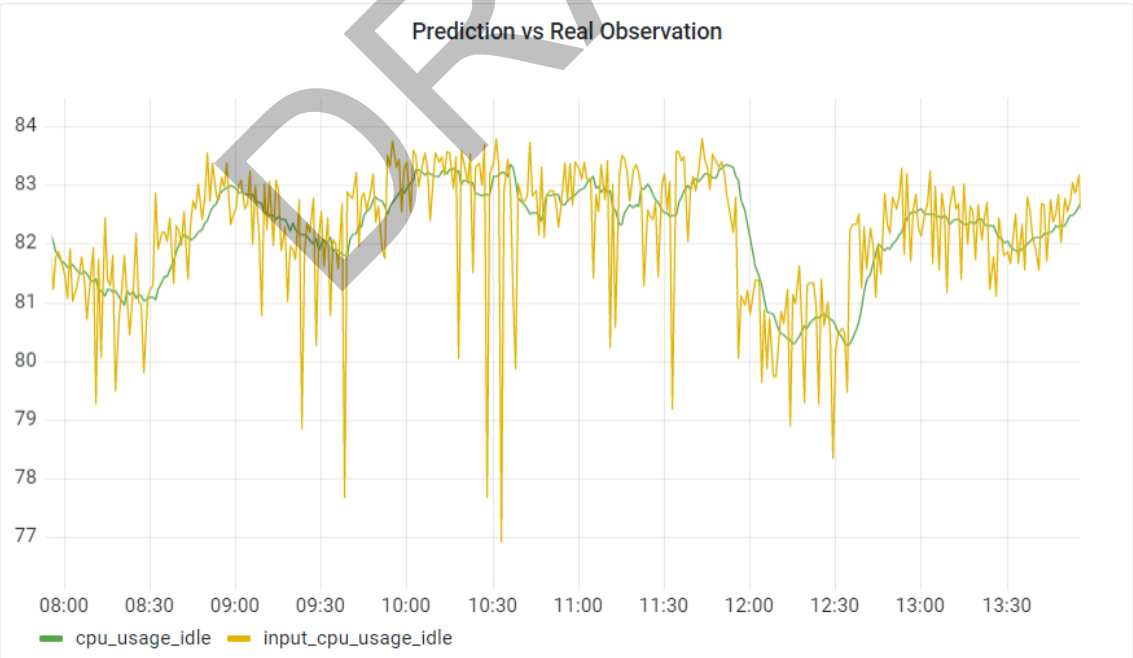

Figure 9: Incremental learning for the CPU usage_idle variable.

[10] https://riverml.xyz/0.14.0/api/time-series/SNARIMAX/

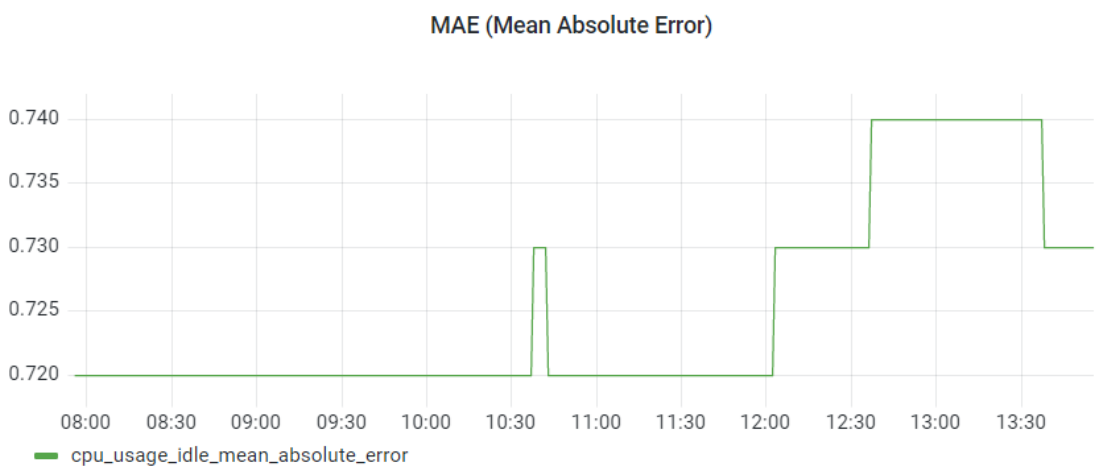

Figure 10: Evolution of the Mean Absolute Error (MAE) for the prediction of the CPU usage_idle variable.

4.2.2.1.2 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.

Figure 11 and Figure 12 show how the outlier detection algorithm is able to detect outliers, and then warns the online learning algorithm not to learn such data points.

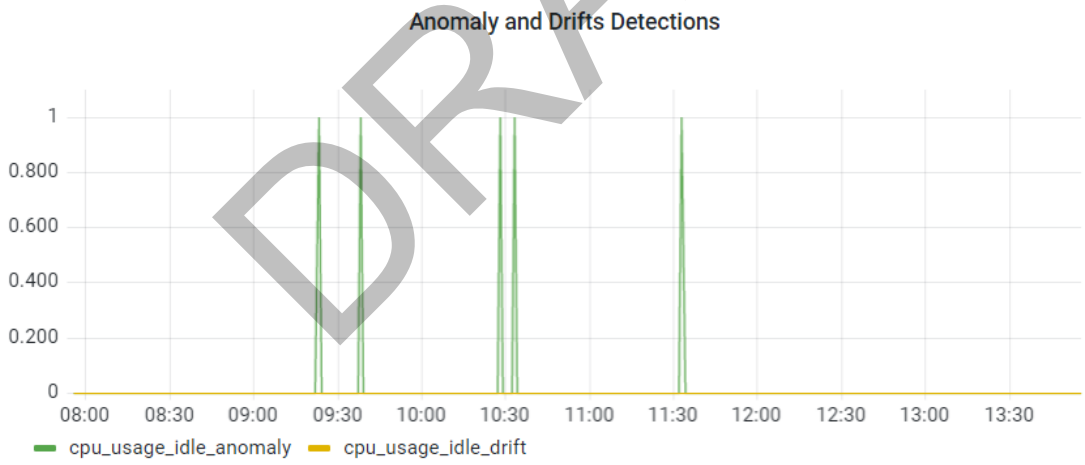

Figure 11: Anomalies detection for the CPU usage_idle.

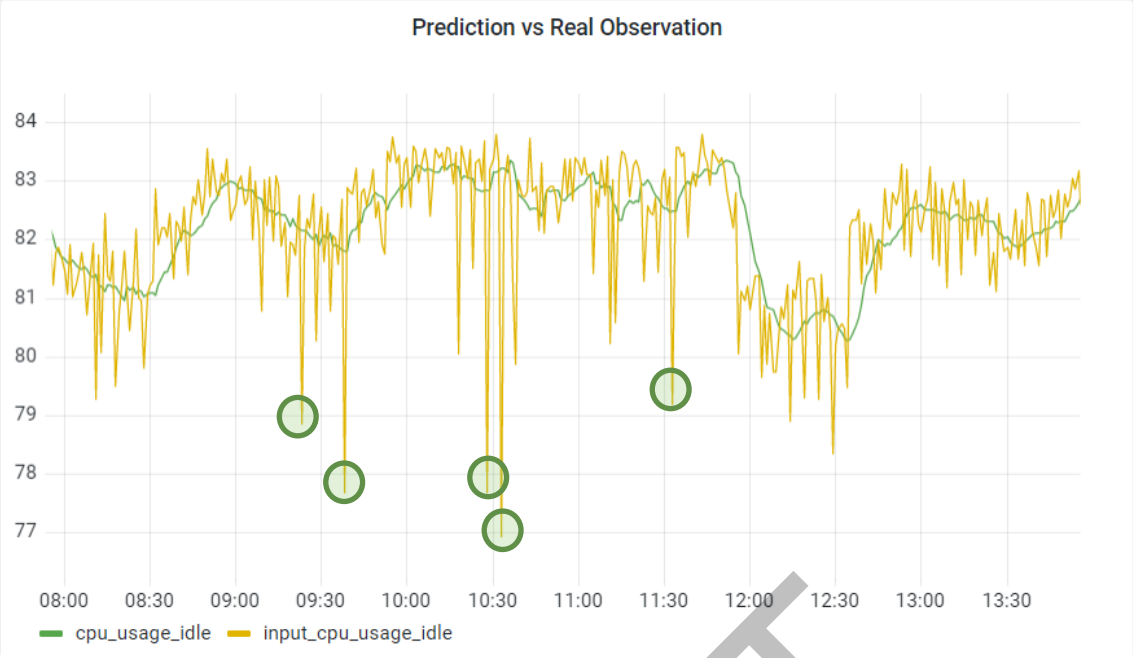

Figure 12: Anomalies detected for the CPU usage_idle variable.

In order to prevent performance degradation in our online learning algorithm, we have found that a simple systematic approach is able to detect the outliers present in our data. Concretely, the outlier detection approach uses the range [*mean - s*std : mean + s*std*] to consider any data points outside the range to be outliers; where *s* is the scalar factor that we use to adjust the sensitivity of such algorithm, and *s=4*.

We see in Figure 12 how these outliers are not affecting the predictive performance of the algorithm. Then, we have achieved a self-learning component robust to outliers, and then the online prediction will be more reliable.

4.2.2.1.3 Concept drift detection

Changes in data distribution should be early detected in order to prevent a performance degradation in the predictions of the algorithm. In order to have a good drift detector, we have prioritised the one that maximizes the true positives while keeping the number of false positives to a minimum.

The popular concept drift detector called ADWIN (*ADaptive WINdowing*) [90] efficiently keeps a dynamic-length window of recent data points, such that it states that there have not been any changes (drift) in the data distribution. Actually, this window is divided into two sub-windows (*W0,W1*), to know whether a change has occurred or not. The average of W0 and W1 are compared to confirm that they belong to the same distribution; if the distribution is different, concept drift phenomenon is detected. Then, after detecting a drift, W0 is replaced by W1, and a new W1 is initialized. We have experimentally tested ADWIN with the performance monitoring data of PIACERE, and as Figure 13 and Figure 14 show for the case of data for *CPU usage_user*, it is able to handle with such changes in data distribution.

.

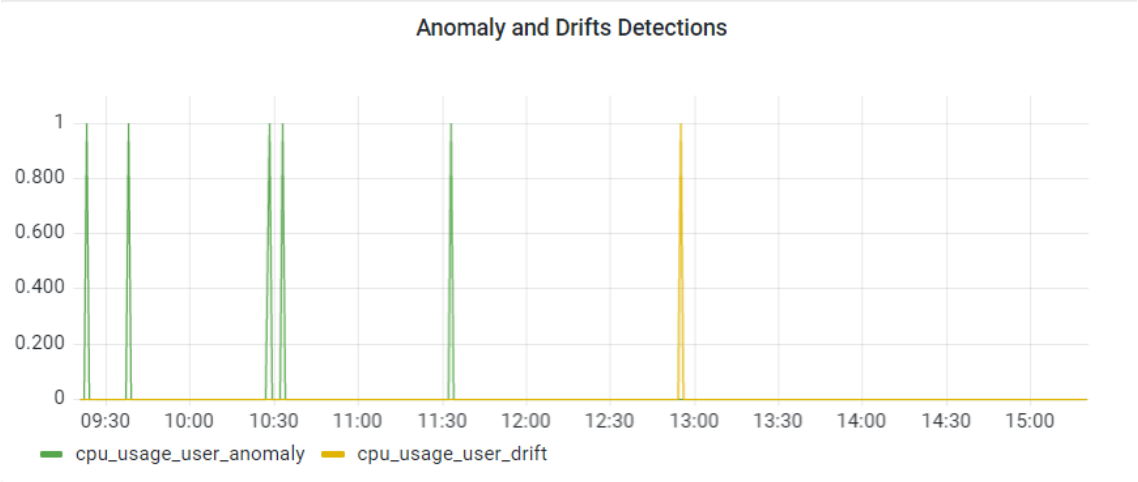

Figure 13: Drift detection for CPU usage_user.

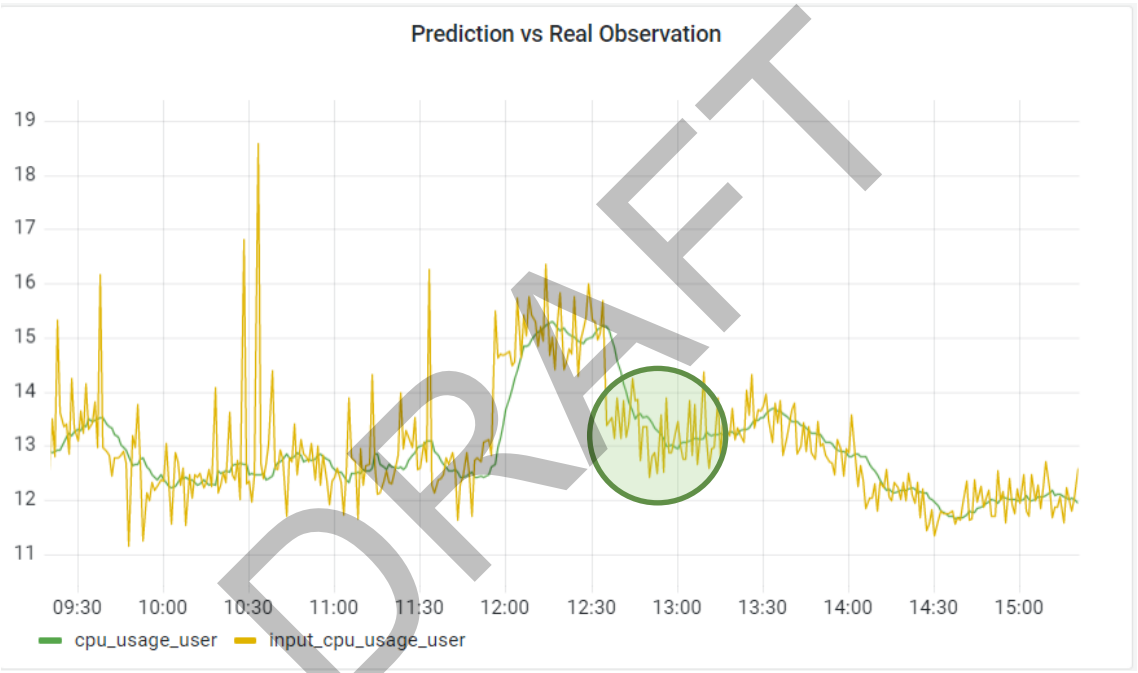

Figure 14: Drift occurrence for CPU usage_user.

4.2.2.1.4 Concept drift adaptation

Adaptation is the required phase when a change has been detected. The algorithm has been incrementally trained with data that belong to a concept (old), and from now on should be adapted to learn the new concept (one). The strategy followed in PIACERE is the recommended one in these cases:

- Reset the drift detector (ADWIN)
- Re-train the model with the new concept. To do that, we use a sliding window of past instances.

4.2.2.1.5 Next actions

- We will try the possibility of predicting several time steps ahead, not only the next one (10 mins). Despite it is not crucial for the current system in PIACERE, because 10 mins is usually more than enough to perform mitigation actions (such as redeployments, optimizations, etc.), it would be a good ingredient for the project.

- We will study the possibility of having just 1 metric for DISK and 1 metric for MEMORY
- We will also study the possibility of having just 1 metric for all the performance metrics
- We will define the availability metric
- We will develop an incremental learning (with anomaly detection, and concept drift detection and adaptation) for the AVAILABILITY metric.

4.2.2.1.6 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.

4.2.2.2 Security Self Learning

4.2.2.2.1 Approach to 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.

Anomaly detection process within PIACERE is implemented an ML-based anomaly detection solution using LogBERT implementation (better said, its variant, using similar approach). Its role is to provide a second layer of analysis of gathered data and metrics, besides code monitoring data feed provided by Security Monitoring components (agents). For the self-supervised learning process it requires only data from normal operating conditions. It transforms from unstructured logs to structured datasets (Drain method; collection of all possible types of events and covering all normal situations to avoid unseen events during the training process). Anomalies are being labelled. The process:

- Use of Masked Language Modelling (MLM) - common in self-supervised NLP

Randomly masked		Predictions
37: instance: * Terminating instance		37: instance: * Terminating instance
[MASK]	→	38: instance: * Instance destroyed successfully
39: instance: * Deleting instance files *		39: instance: * Deleting instance files *
40: instance: * Deletion of * complete		40: instance: * Deletion of * complete
[MASK]	→	41: instance: * Took * seconds to destroy the instance on the hypervisor

Figure 15: Process of masking and predictions within anomaly detection process.

- Hypersphere Volume Minimization (HVM) - Hypothesis that 'normal' samples can be mapped to close representations.

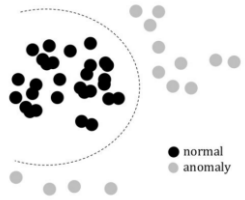

Figure 16: Example of anomaly detection.

Approach to Security self-learning has already been indicated in section 4.1.2.2: using Filebeat to aggregate the logs, then parse these logs in anomaly detection component in order to create

structure logs as an input to anomaly detector (Figure 21). Anomaly detector is capable to create and label specific events with anomaly score, and ship these newly created events back to ELK stack used by Security Monitoring.

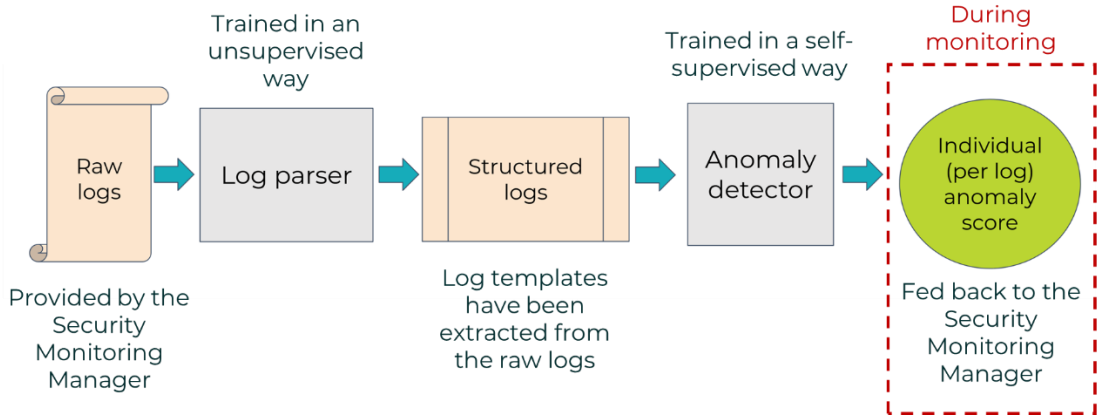

Figure 17: Approach to security self-learning in PIACERE.

4.2.2.2.2 Integration

In the following figures we show a security self-learning dashboard from one of the demo deployments. Figure 18 shows three different time-based aggregations of anomalies scores as line plots. Different granularities provide better overview about the status in the system. Figure 19 shows histogram of logs and table with details below. The table contains timestamp of a log, anomaly score, and message. The unknown column marks if a log message was recognized by log parser. All the charts can be used to zoom in a specific area for easier inspection of the logs.

Figure 18: Different aggregations of anomaly scores presented as line plots.

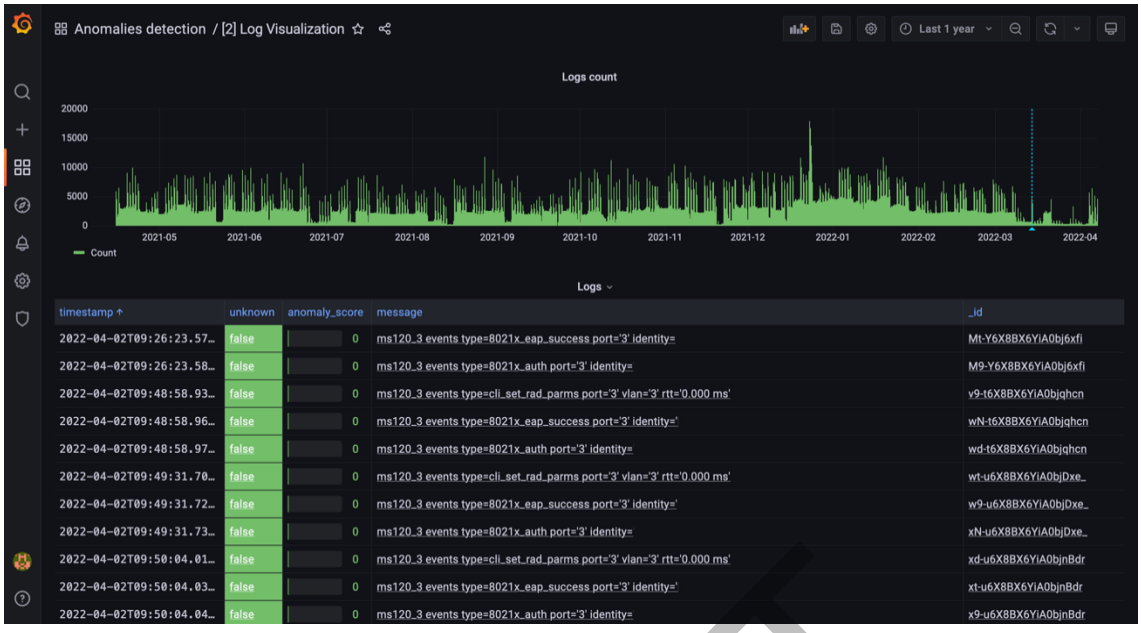

Figure 19: Histogram of logs and a table with log messages, anomaly scores, and additional information.

4.3 Self-healing

4.3.1 Self-healing strategies and challenges

The scope of the self-healing in PIACERE is focused on the gathering of detected anomalies and to take 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 the literature we can find different approaches that extend the scope of the self-healing, covering other phases both before and after that planning activity. The Table 7 below shows a comparison of the potentially related approaches with respect to: the covered phases of the classical autonomic control loop "MAPE-K (Monitor-Analyse-Plan-Execute over a shared Knowledge)" [91], technique used, whether it's a decentralized or centralized approach, and the applied context.

Table 7: MAPE-K results.

Reference	MAPE-K	Technique	[De]centralized	Context
Di Nitto et al. [92]	M, A, P, E, K	Probability theory	Decentralized	Microservice
Maimó et al. [93]	M, A, P, E, K	Deep learning	Centralized	5G networks
Yang et al. [94]	M, A, P, E, K	Security theory	Decentralized	IoT-based healthcare storage

Alhosban et al. [95]	M, A, P, E	Predictive model	Centralized	Cloud service
Azaiez and Chainbi [96]	M, A, P, E	Multi-agent	Decentralized	Cloud
Gill et al. [97]	M, A, P, E	Reactive	Centralized	Cloud service
Li et al. [98]	M, A, P, E	Reactive	Centralized	Cloud computing
Magalhaes and Silva [99], [100]	M, A, P, E	Statistical theory	Centralized	Web application
Rajput and Sikka [101]	M, A, P, E	Multi-agent	Decentralized	Distributed environment
Rios et al. [102]	M, A, P, E	CAMEL based	Decentralized	Distributed environment
Mosallanejad et al. [103]	M, A, E	Reactive	Centralized	Cloud
Wang et al. [104]	M, A	Machine learning	Centralized	Cloud computing

Several approaches for self-healing/self-adaptive systems are provided in the table above. In [105], 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. [106]. In [107], 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 [108], 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 [109] 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.

The approaches in Table 7 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. [92] 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 [93], 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 [94], 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. [95] 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 [96] 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. [97] 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. [98] 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 [99] 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 [100], 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 [101] propose an architecture which could support agent-based distributed systems to address fault recovery for achieving self-adaptiveness.

Rios et al. [102] 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. [103] propose an SLA based self-healing model for the cloud environment to monitor SLA and detect SLA violation automatically.

Wang et al. [104] 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.

4.3.2 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 leverage three elements, see Figure 20: notification messages, notifications types and response strategies.

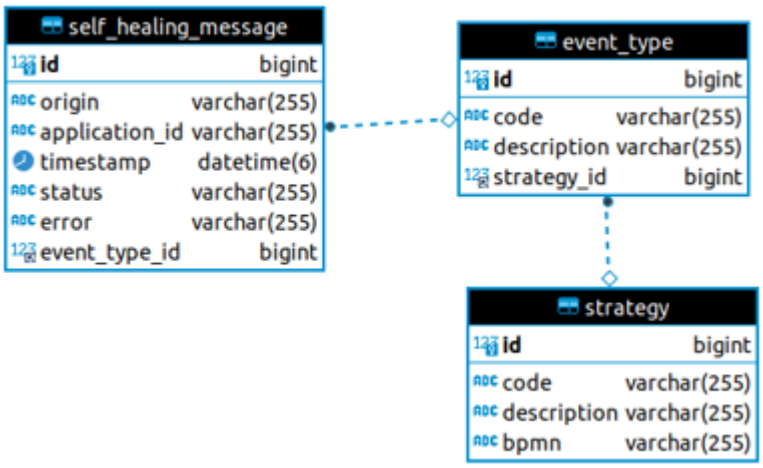

Figure 20: Self-healing elements.

Self-healing messages are sent by monitoring and self-learning components. To do so self-healing provides a REST api, with capability to create self_healing_messages. The self_healing_messages will include as mandatory field the event type. Besides it will include the severity of the event warning or critical.

We have created an infrastructure to define the allowed event types and the strategy to respond to each of those events based on their criticality. The most obvious strategy will be to redeploy the applications that raise critical situations, but other strategies are also in evaluation such as the possibility to reboot the infrastructure, to scale, to put deployment in quarantine state, to notify user, etc.

For most of the strategies the general approach in which these strategies are going implemented are BPMN workflows to be executed by PRC. During this period based on the security monitoring feedback we have also identified the ansible based strategies that will be executed in the running infrastructure element, in this case the execution will be manage by the IEM.

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.

As the strategies could take some time to be executed, we still 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. This will be done in the next period as we test the component in the pilots.

5 Implementation

5.1 Monitoring Controller

The monitoring controller is an aggregator component that serves as unique front-end to lower level specialised controllers. These are the performance monitoring controller and the security monitoring controller.

5.1.1 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 21 describes the sequence diagram for the two main functionalities to be covered by the Monitoring Controller, this is start and stop deployments.

From the previous version we have evolved the diagram to include the request dashboard workflow required by the IDE and the loop that is being implemented to feed the Infrastructure element Catalog (IEC) with the information gathered from the deployed infrastructure.

Figure 21: 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 second implementation of the Monitoring Controller still aims to provide a REST API to be used by the PIACERE Runtime Controller.

The Monitoring Controller 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 destroys application deployments, as such it does not provide innovative advances to the state of the art.

5.1.2 Requirements covered by this prototype

The user requirements from WP2 satisfied by this version is described in the Table 8. 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 8: Monitoring Controller related user requirements from WP2.

Req ID	Description	Implementation Status	Requirement Coverage at M24
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 non-functional requirements regarding performance, availability and security. The current version of the component includes the function to call the remaining components.
REQ51	Deployment non-functional requirements tracking	In progress	The component will forward all the necessary information to the self-learning components to be able to track the infrastructure

	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).		related non-functional requirements.

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: Monitoring Controller related internal requirements.

Title	Implementation Status	Requirement Coverage at M24
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	Completed	The OpenAPI has been defined and put under configuration control https://git.code.tecnalia.com/piacere/public/the-platform/runtime-monitoring/mc/-/blob/y2/src/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/public/the-platform/runtime-monitoring /pm
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[11] and Vagrant[12]. 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: • Docker-compose https://git.code.tecnalia.com/piacere/public/the-platform/runtime-monitoring/pm/pm-deploy • Python code https://git.code.tecnalia.com/piacere/public/the-platform/runtime-monitoring/mc/-/blob/y2/README.md We update continuously the documentation as we advance in the coding and pre-integration of the monitoring components
Unitary test	In progress	We have included a testing framework to the code based on Tox[13]. 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 [14] https://git.code.tecnalia.com/piacere/public/the-platform/runtime-monitoring/pm/-/blob/y2/.gitlab-ci.yml The integration approach requires to be migrated with the rest of the components of PIACERE framework.
Test in use cases	In progress	We have tested the component with use-case in the deployment feature, the un-deploy feature will be tested in the following version.

[11] https://www.virtualbox.org/
[12] https://www.vagrantup.com/
[13] https://tox.wiki/en/latest/index.html
[14] https://docs.gitlab.com/ee/ci/

5.1.3 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 22). 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

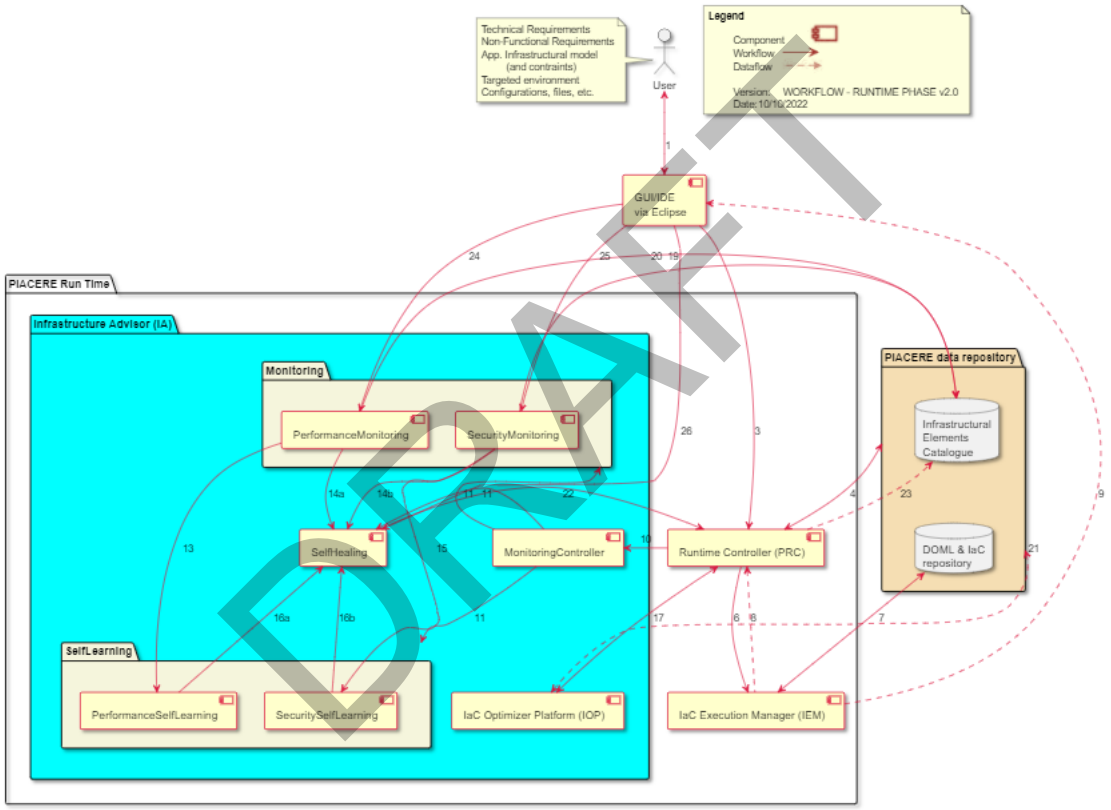

Figure 22: PIACERE Runtime Diagram on its 2.0 version focussed in monitoring components.

5.1.4 Technical description

This subsection is devoted to describing the technical specification of this second prototype. First, the main architecture of the prototype and the components are shown and described in Section 5.1.4.1. *Prototype Architecture*. This subsection finishes with the technical specifications of the developed system in *Section 5.1.4.2 Technical Specifications*.

5.1.4.1 Prototype architecture and components description

The main architecture of this second prototype is depicted in the following Figure 23. 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.

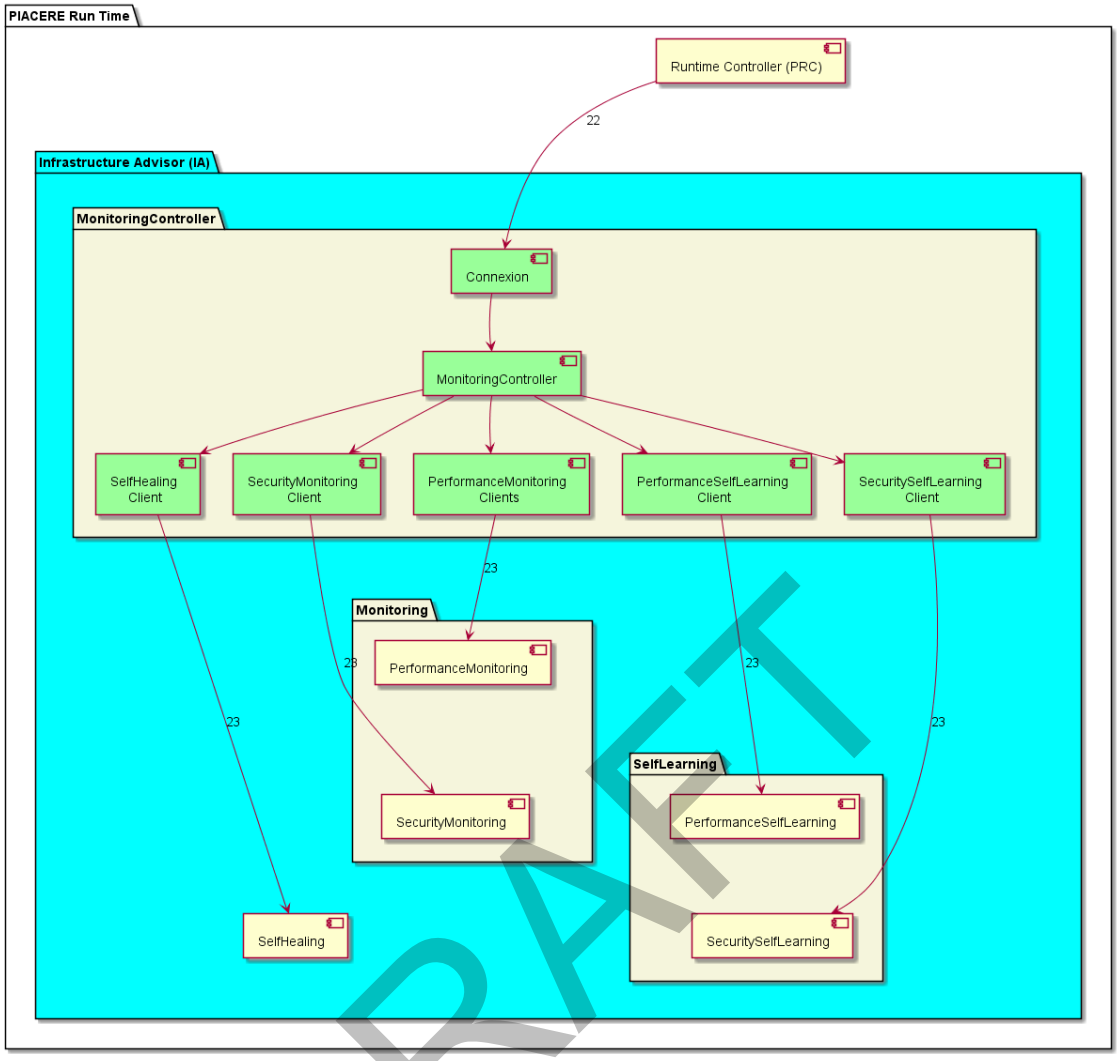

Figure 23: Monitoring Controller second prototype architecture.

This second prototype of the Monitoring Controller is composed by four components and we are still deciding if self-healing requires to know that a concrete deployment has been activated. We will integrate this fifth component if during the development of the last strategies we identify the need to setup something in the self-healing component for each active deployment.

- 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.
- 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.

Planned component for the next iteration is:

- Self-healing Client: This will be also an autogenerated component with the openapi generator.

5.1.4.2 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 ready, 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.

5.2 Performance Monitoring

5.2.1 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 24 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.

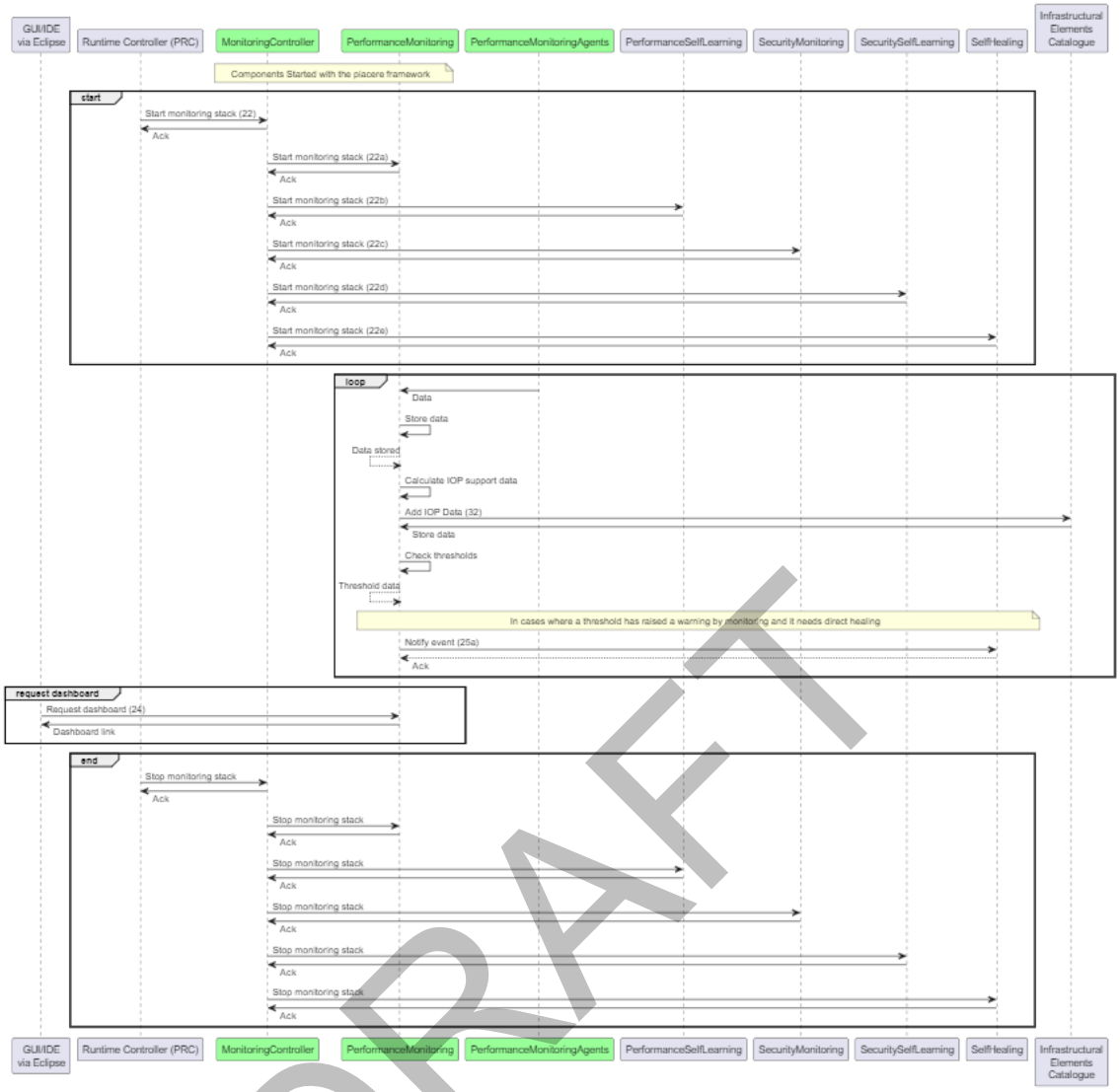

Figure 24: Performance monitoring Sequence diagram.

The second implementation of the Performance Monitoring aims to:

- Automate the creation of dashboard specific to the deployments under request from its API
- To provide information on those dashboards to the IDE
- To integrate dashboards for real-time and those coming from the information generated from the self-learning component

The following figure represent the internal workflow of the performance monitoring components and their internal parts. Based on the request from the user, typically done from the IDE, the PRC request the activation of the deployment to the IEM and to the monitoring stack.

The IEM deploys the IaC generated by the ICG that contains the monitoring agents. This will create somewhere the infrastructure elements required by the application, and these infrastructure elements will contain the monitoring agents.

The monitoring stack receives the deployment activation request from the PRC. It forwards this request to lower level monitoring components such as performance monitoring controller. The

performance monitoring controller configures appropriate dashboards that will take care of the visualization of metrics with different purposes.

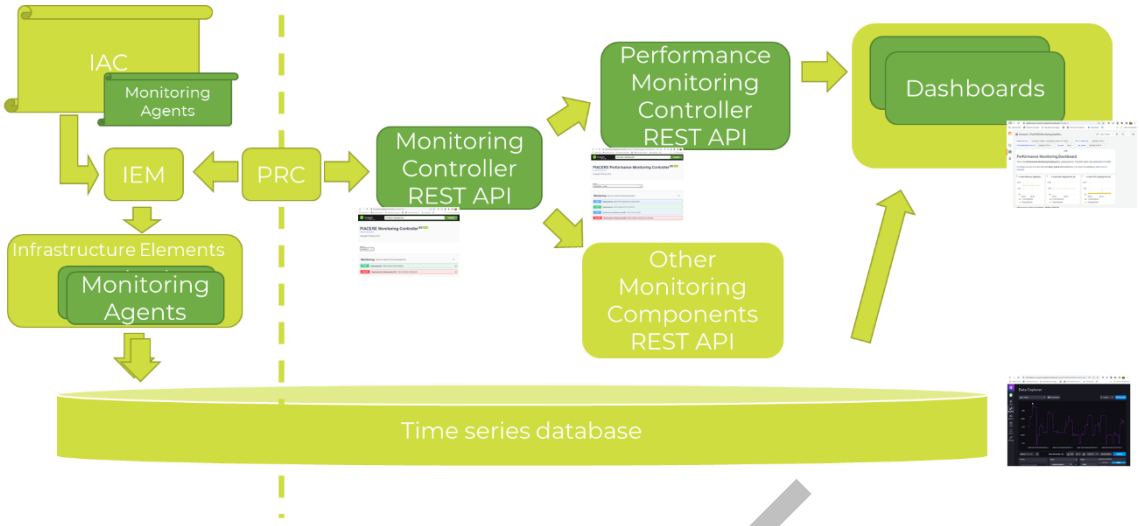

Figure 25: Performance monitoring internal workflow.

5.2.2 Requirements covered by this prototype

The user requirements from WP2 satisfied by this interim version are described in the Table 10. 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 10: Performance Monitoring related user requirements from WP2.

Req ID	Description	Implementation Status	Requirement Coverage at M24
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 was planned to be implemented in M12, we did not finalise it as we have decide to simplify the scales of the metrics to simplify the understanding by the pilots. It will be completed during the next period.
REQ47	Full monitoring stack *The monitoring component shall include the needed elements in*	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

	the stack to monitor the infrastructural elements.		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: • Complete missing features regarding notification of threshold and IEC (Infrastructure Element Catalogue) feed.
REQ48	Self-learning focused monitoring *The monitoring component shall transform the real time values into the correct format/type/nature for the self learning component.*	Completed	Real time data is stored and the performance self-learning prototype is actually capable of consuming that information using the provided interface.
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. Dashboards are in place to represent them and they are accessible from the IDE. In the following months, we plan to simplify the metrics to percentual scale. Besides, we will also try to provide an aggregated performance metric that gives high level view of the overall health of the system.
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*	In progress	We have defined a predefined set of thresholds based on the state of the practice experiences. Now we are implementing a simplification of the scales of the metrics in order to refactor the notifications base on those simplified thresholds.

	not likely to occur. This implies the compliance of a predefined set of non-functional requirements (e.g. performance).		
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.*	Completed	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. Dashboards for self-learning have been implemented and are accessible from the IDE for each deployment.
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.*	Completed	The current version of the Performance Monitoring includes a graphical user interface that renders the information coming from the time series database. We have introduced a deployment-based dashboard that includes information related to the NFR thresholds coming from the DOML specification.
REQ93	Self-healing should classify the events notified	In progress	*Self-healing component shall classify the events received from the self learning and derive corrective actions.* We are receiving notifications from some of the monitoring elements, and those elements include the typology. Currently we are extending the strategies that may head to additional or refined types from some of the monitoring components.

The internal requirements satisfied by this interim version are described in the Table 11. All these requirements are as well polished and adapted as the project advances.

Table 11: 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/public/the-platform/runtime-monitoring/pm
Implement REST API specification	Completed	The final version of the OpenAPI has been defined and put under configuration control https://git.code.tecnalia.com/piacere/public/the-platform/runtime-monitoring/pm/-/blob/y2/git/pmc/openapi.yaml The OpenAPI have been implemented and put in integration.
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/public/the-platform/runtime-monitoring/pm
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: ▪ Docker-compose https://git.code.tecnalia.com/piacere/public/the-platform/runtime-monitoring/pm

		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	Completed	We have completed the end-to-end deployment scenario for both the demo project and for the use cases.
Continuous integration	In progress	Continuous integration has been implemented based on gitlab-ci https://git.code.tecnalia.com/piacere/public/the-platform/runtime-monitoring/pm/-/blob/y2/.gitlab-ci.yml The integration approach requires to be migrated with the rest of the components of PIACERE framework.

5.2.3 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 22). 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.

5.2.4 Technical description

This subsection is devoted to describing the technical specification of this second prototype. First, the main architecture of the prototype and the components are shown and described in Section 5.2.4.1. This subsection finishes with the technical specifications of the developed system in *Section 5.2.4.2 Technical Specifications*.

5.2.4.1 Prototype architecture and components description

The main architecture of this second prototype is depicted in the following Figure 23. In this architecture, four different components can be distinguished (highlighted in green): Performance Monitoring Controller, Influxdb, Grafana and Performance Monitoring Agents. Then main purpose of these components is described.

From the previous version we have added links from the IDE to the performance monitoring dashboard to provide the users a easier access to the monitoring information.

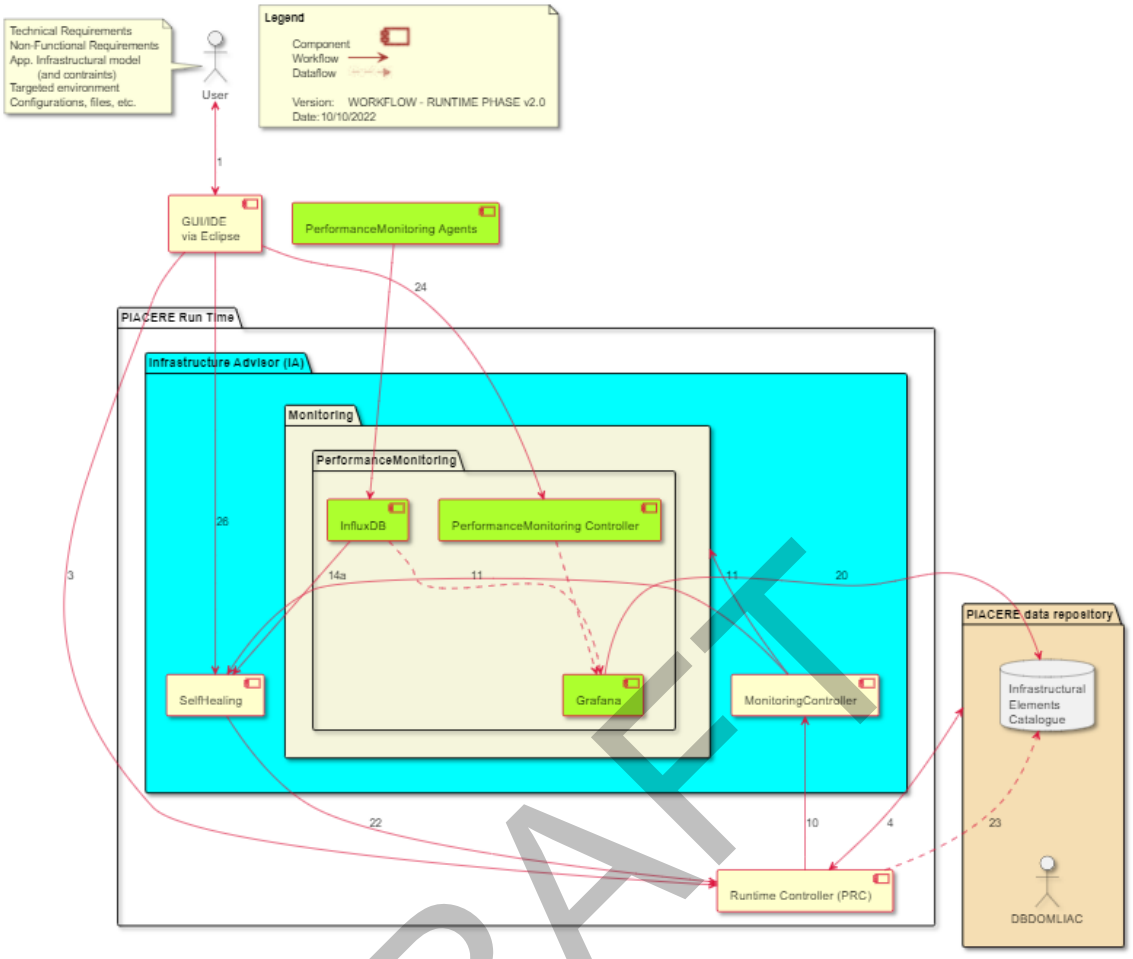

Figure 26: Performance Monitoring second prototype architecture.

This second 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[15] 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[16]

The component running in 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[17].

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

5.2.4.2 Technical specifications

The Performance Monitoring Controller prototype has been developed using Java, more specifically the Java Spring Boot framework[18] 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.

Internally we have developed an client from the recent Grafana openApi https://github.com/grafana/grafana/blob/main/public/api-spec.json this allow us to easily adapt to higher versions of Grafana in case we need to evolve to bring new features or security patches.

Additional dashboard has been integrated to show self-learning computed data, this has introduced the usage of dashboard folders to aggregate the dashboard of each deployment.

5.3 Security Monitoring

5.3.1 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 (raw logs needed for building the model and real-time detection, and events as output of the anomaly detection system. Sources of raw logs are configurable via Filebeat instance (agent), but normally it already includes these files as sources: audit.log, cron, dmesg, messages, secure and yum.log. Finally, it aggregates metrics based on actual measurements and updates the characteristics of the services listed in the Infrastructure Element Catalogue.

The Figure 27 describes the sequence diagram of the Security Monitoring and Security Self-learning processes.

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

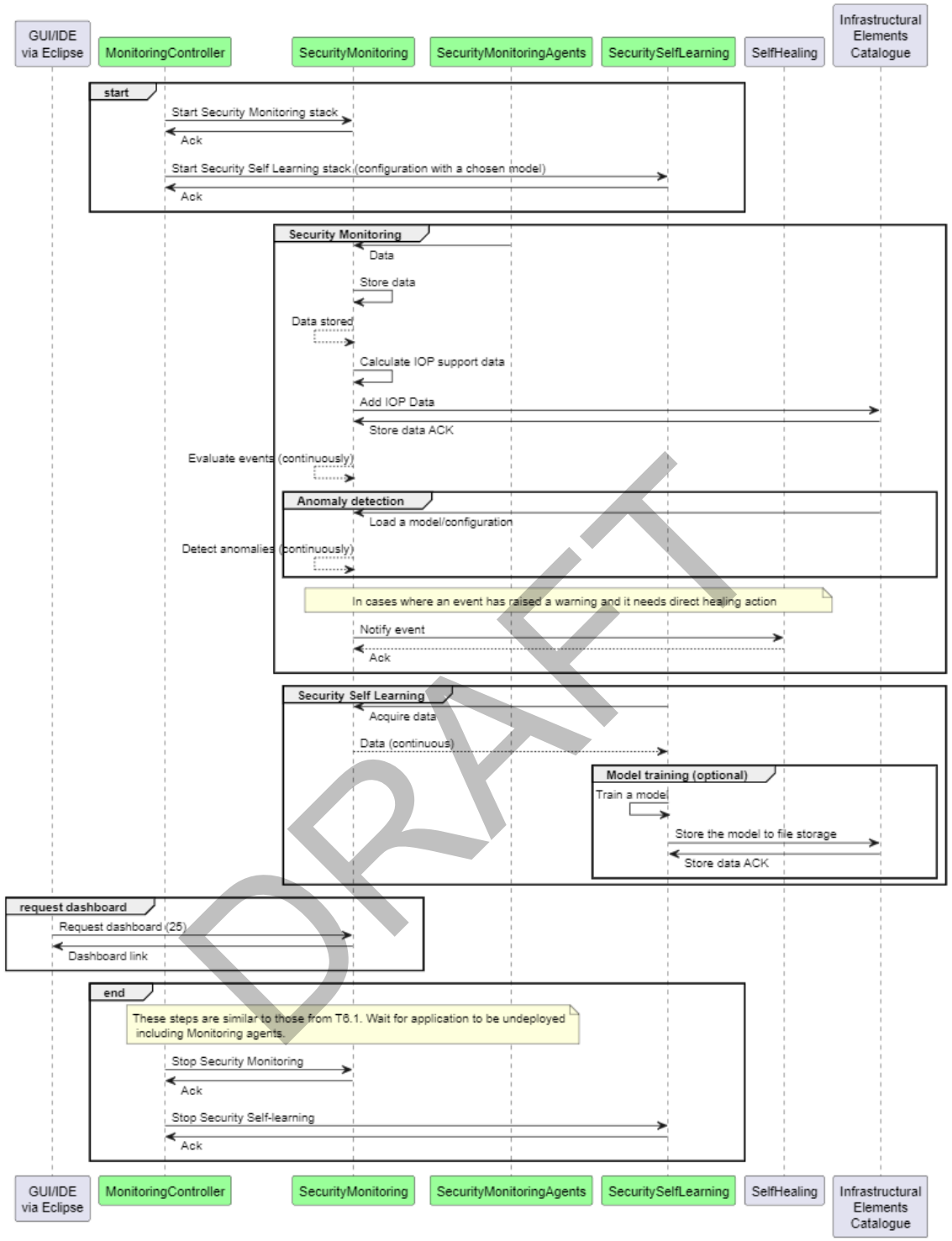

Figure 27: Security Monitoring Sequence diagram.

The second implementation of the Security Monitoring:

- Provides the Controller, exposing Security Monitoring API and orchestrating underlying services (Figure 28): 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

5.3.2 Requirements covered by this prototype

The user requirements from WP2 satisfied by this interim version are described in Table 12.

Table 12: Security Monitoring and Security Self-learning Requirements related user requirements from WP2.

Req ID	Description	Implementation Status	Requirement Coverage at M24
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 can provide monitoring data from the application layer (infrastructure's guest) w.r.t. security metrics	In-progress	This is possible through the configuration of the Security Monitoring Manager (specifically, Wazuh configuration). However, additional default configuration for the use cases needs to be included in the agent's configuration.
REQ16	Runtime security monitoring should contribute to mitigation actions taken when considering plans and strategies for runtime self-healing actions	In-progress	Basic mitigation strategies have already been defined. Currently, we are having conversations within WP6 how to advance on the integration with the self-healing components (integration from the components is already in progress).
REQ17	Deployment of runtime security monitoring should happen seamlessly or with minimal effort and configuration required by the user.	Done.	The deployment code is partially already available on the project's public 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).	Done.	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	The integration of the self-healing component is under development. Deployments of these components are available, but some configuration steps are still manual.
REQ21	Runtime security monitoring and Runtime monitoring infrastructure should be integrated with minimal extensions.	In-progress	The integration is being 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	DOML extension has already been provided in order to express basic rules on detection and triggering actions based on user input.
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 of a predefined set of non-functional requirements (e.g. performance)	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). In the proposed DOML extensions, it is possible to express new metrics and related NFRs and rules for the actions to be taken in the case of fulfilling the rules. Still, the implementation of the process has not been finished until now.

The internal requirements satisfied by this interim version are described in the Table 13. All these requirements are as well polished and adapted as the project advances.

Table 13: 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	An updated version (v1.1.1.) of the OpenAPI has been defined and put under configuration control : https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-controller/-/blob/y2/swagger_server/swagger/swagger.yaml
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

5.3.3 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 22). 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.

5.3.4 Technical description

5.3.4.1 Prototype architecture and components description

Figure 28 depicts architecture of the Security Monitoring and Security Self-learning components.

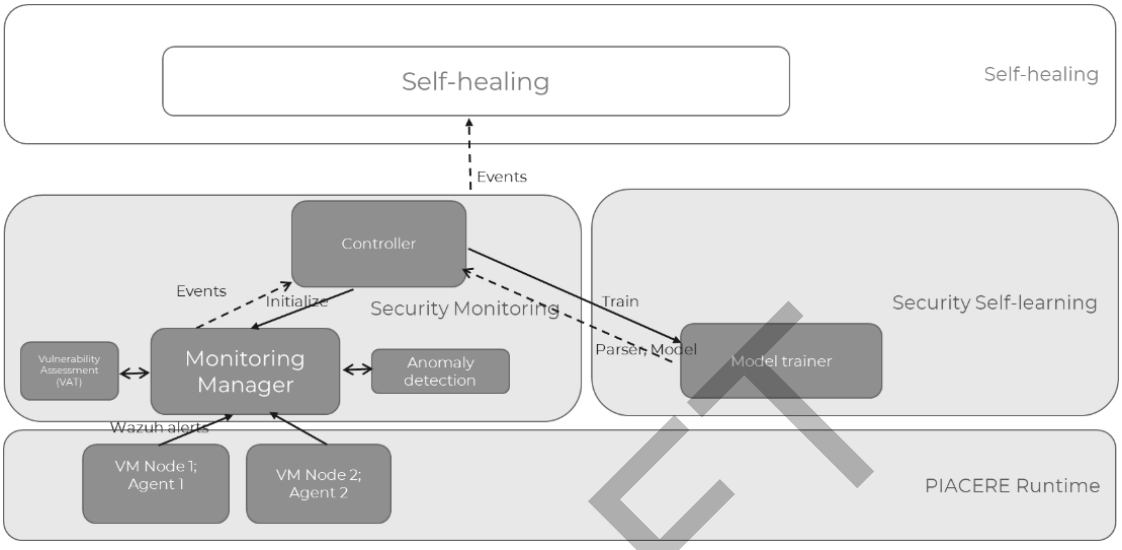

Figure 28: Architecture of Security Monitoring and Security Self-learning.

Controller exposes underlying APIs of the Monitoring Manager and Model Trainer via RESTful (OpenAPI spec) API. Model Trainer internally stores trained models in the internal Model Repository. 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 (and stored within its internal component – Model Repository). 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.

5.3.4.2 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://git.code.tecnalia.com/piacere/public/the-platform/runtime-security-monitoring/security-monitoring-controller/-/blob/main/swagger_server/swagger/swagger.yaml).

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 4.2 which already provides agents and ELK stack (based on OpenDistro Elasticsearch) 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[19]) 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.

5.4 Performance Self-learning

5.4.1 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 forecasted metric exceeds a threshold.

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

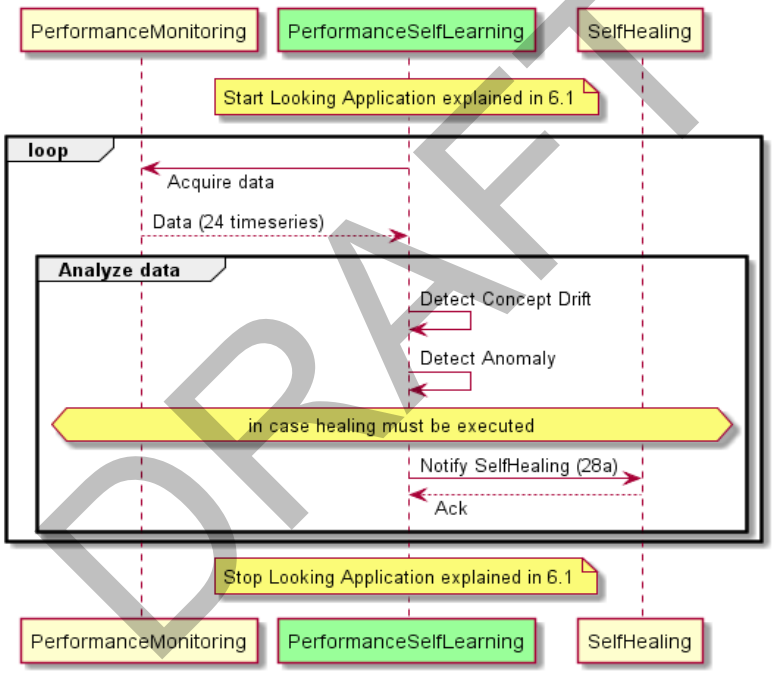

Figure 29: Self-learning sequence diagram.

The following figure describes how the sequence diagram is implemented internally in the performance self-learning. The self-learning iteratively processes metrics for each monitored deployment. The processing starts with the checking if the training has done, if the training is not done it check if enough metrics are in place to perform the initial model. Currently 200 metrics are used but that can be customised. After the initial model has been generated each follow up metric is used in the online learning and prediction algorithm that also consider anomalies and drifts. The outcome information is used to compare with the thresholds and inform the self-healing if necessary. Besides, that information is stored back in the time series database for its latter usage.

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

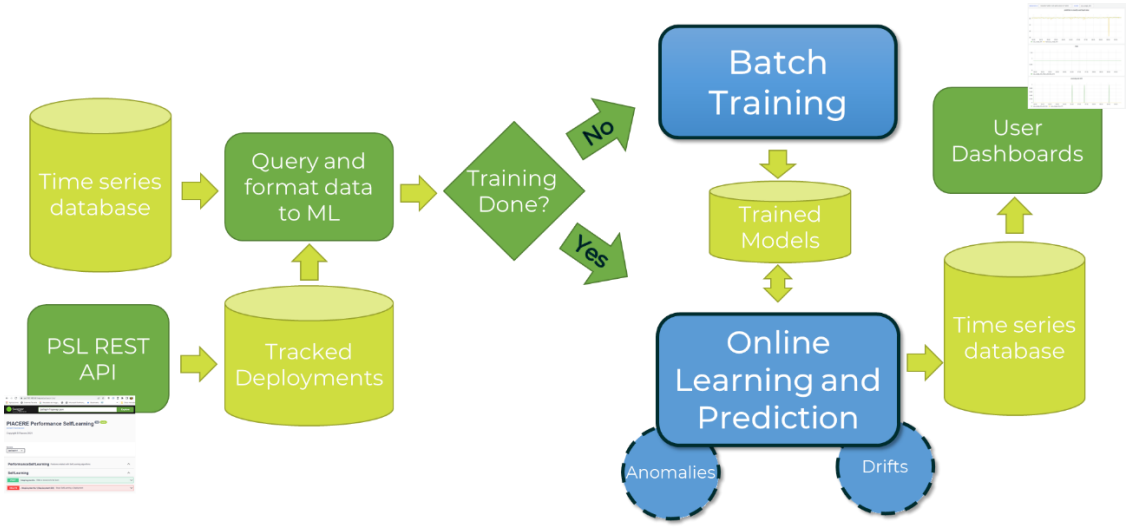

Figure 30: Self-learning workflow diagram.

5.4.2 Requirements covered by this prototype

The user requirements from WP2 satisfied by this interim version are described in the Table 14.

Table 14: 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 Performance Self-Learning component has direct access to the data in the Performance Monitoring database. At the moment execution is fast enough in the initial test of pilots but further analysis is planned for 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 predefined set of non-functional requirements (e.g. performance)	Completed	The Performance Self-Learning component is able to run fast by filtering the data to be acquired.
REQ52	Self learning shall consume the data monitored and stored in a time-series database to create	Completed	Thanks to the integration of the Performance Monitoring database access implementation, the Performance Self-Learning

	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		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 15. All these requirements are as well polished and adapted as the project advances.

Table 15: Performance Self Learning 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/public/the-platform/self-learning
Implement REST API specification	Completed	The OpenAPI has been defined and put under configuration control: https://git.code.tecnalia.com/piacere/public/the-platform/self-learning/-/blob/y2/docs/self-learning-openapi.yaml
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	Following the Dependency specification for Python Software Packages the required file is provided: https://git.code.tecnalia.com/piacere/public/the-platform/self-learning/-/blob/y2/requirements.txt
Include usage documentation	In-progress	Deployment information have been provided and usage documentation is being working out based on the feedback from the use case application.
Unitary test	In-progress	Not yet available.
Integration test	In-progress	The deployment and collection have been performed. Additional test are required to be able to exemplify the self-learning add value in a concrete scenario.

Continuous integration	Completed	The component has been deployed with the rest of the components of the PIACERE platform.

5.4.3 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 22). 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.

5.4.4 Technical description

5.4.4.1 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 31.

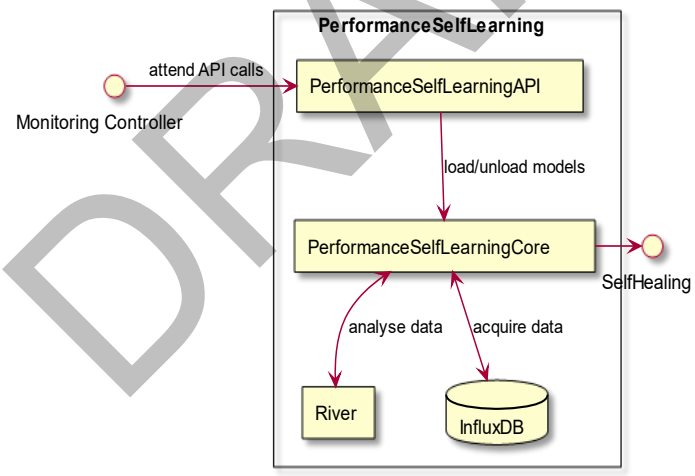

Figure 31: Architecture of the Self-learning component.

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 is notified in case of exceeding the idle threshold for the metric being monitored.

- 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.

5.4.4.2 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 requires the integration with different components with a RESTful API to be aware of new deployments to be analysed and to warn the Self-Healing 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.

5.5 Security Self-learning

5.5.1 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 Figure 27). 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 28), which eventually stores every trained model in the Model Repository (currently part of the Infrastructural Model Trainer, eventually could be part of the Elements Catalogue as initially planned). 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 32 illustrates this process.

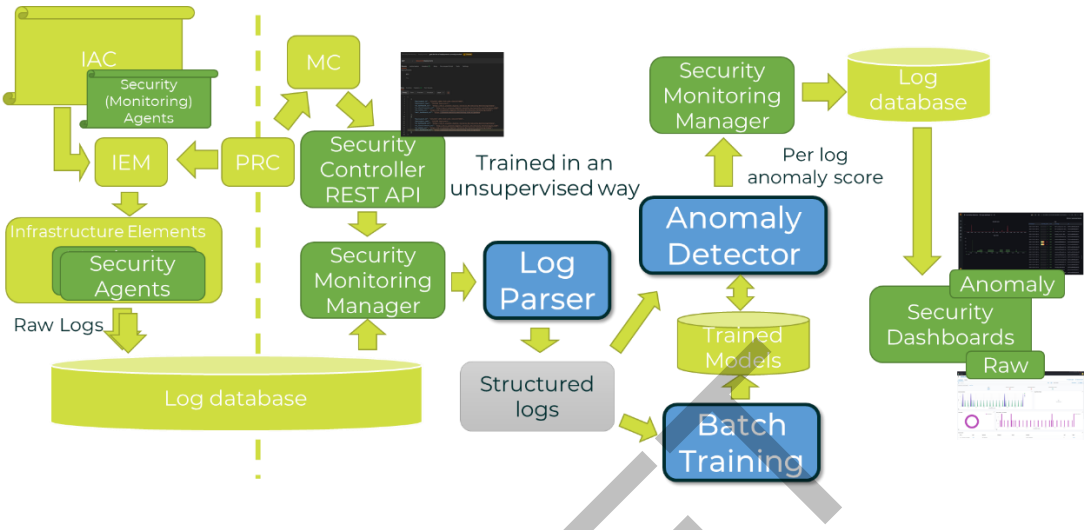

Figure 32: Internal functioning of the Model Trainer.

The sequence diagram describing the functioning of the Security Self-learning component can be seen in Figure 27, integrated with that of the Security Monitoring.

5.5.2 Requirements covered by this prototype

The user requirements are listed under Security Monitoring Implementation section in Table 12. Internal requirements covered by this prototype are listed in Table 16.

Table 16: Internal requirements for Security Self-learning.

Title	Implementation Status	Requirement Coverage at M24
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 , available also on the public repository: https://git.code.tecnalia.com/piacere/public/the-platform/runtime-security-monitoring/security-monitoring-controller/-/tree/main/
Implement REST API specification	Completed	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 private repository (not in project's Gitlab).
Provide fast deployment alternative for deployment, testing and evaluation	Completed	Part of the private repository (not in project's Gitlab).
Include usage documentation	Completed	Part of the private repository (not in project's Gitlab).
Unitary test	Completed	Part of the private repository (not in project's Gitlab). CI instance in PIACERE already links to deployment of the dashboards.
Integration test	Completed	CI instance in PIACERE already links to deployment of the dashboards.
Continuous integration	Completed	CI instance in PIACERE already links to deployment of the dashboards.

5.5.3 Fitting into overall PIACERE Architecture

Security Self-learning's architecture and fitting into the overall PIACERE Architecture is described in section 5.5.4.1.

5.5.4 Technical description

5.5.4.1 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 28.

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 based on Drain method, and a new anomaly detection model based on LogBERT are created. These objects belong to the Model Trainer component (Figure 28) 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 21, Figure 22 in section 4.2.2.2) 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.

5.5.4.2 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.

- Security Self-learning component uses dedicated Elasticsearch's indexes that are considered as input for the anomaly detection process.

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.

5.6 Self-healing

5.6.1 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 33 shows the sequence diagram for the main activity that the self-healing should implement: the notification processing workflow.

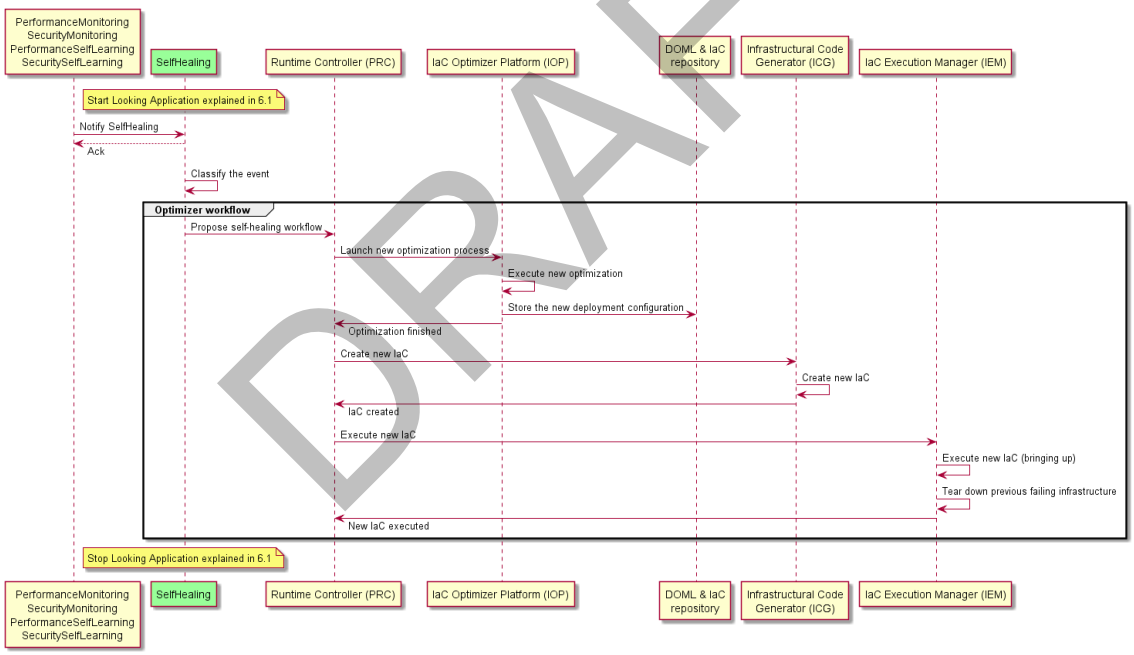

Figure 33: Self-healing sequence diagram.

The second implementation of the self-healing aims to:

- Wait for event notifications from 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 the future such as its severity.
- Queue 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 of the types of notifications and strategies applied to each of them.

The following figure represent the internal workflow of the self-healing components and their internal parts.

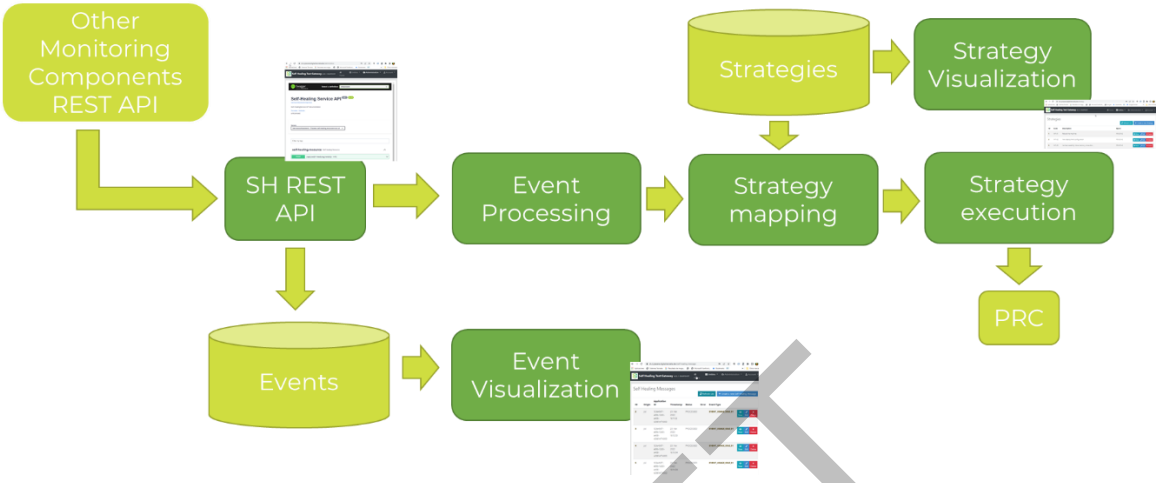

Figure 34: Self-healing internal workflow.

5.6.2 Requirements covered by this prototype

The user requirements from WP2 satisfied by this interim version are described in the Table 17. 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 17: Self-healing related user requirements from WP2.

Req ID	Description	Implementation Status	Requirement Coverage at M24
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*	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

	accessible to the IOP (through the dynamic part of the infrastructural catalogue).		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 progress	We have implemented deployment level strategies in this level. In the next period we are developing ansible strategies that will enable the specification of actions on specific infrastructure elements.
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 18. All these requirements are as well polished and adapted as the project advances.

Table 18: Self-healing related internal requirements.

Title	Implementation Status	Requirement Coverage

Implement OpenAPI specification	Completed	It is available in the public repository.
Implement specification first approach	Completed	The notification interface is implemented and it is receiving notifications from other components in the monitoring infrastructure.

5.6.3 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 22). 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.

5.6.4 Technical description

This subsection is devoted to describing the technical specification of this second prototype. First, the main architecture and the components of the prototype are shown and described in section 5.6.4.1. This subsection finishes with the technical specifications of the developed system in Section 5.6.4.2.

5.6.4.1 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 second prototype, Kafka streaming solution has been chosen.

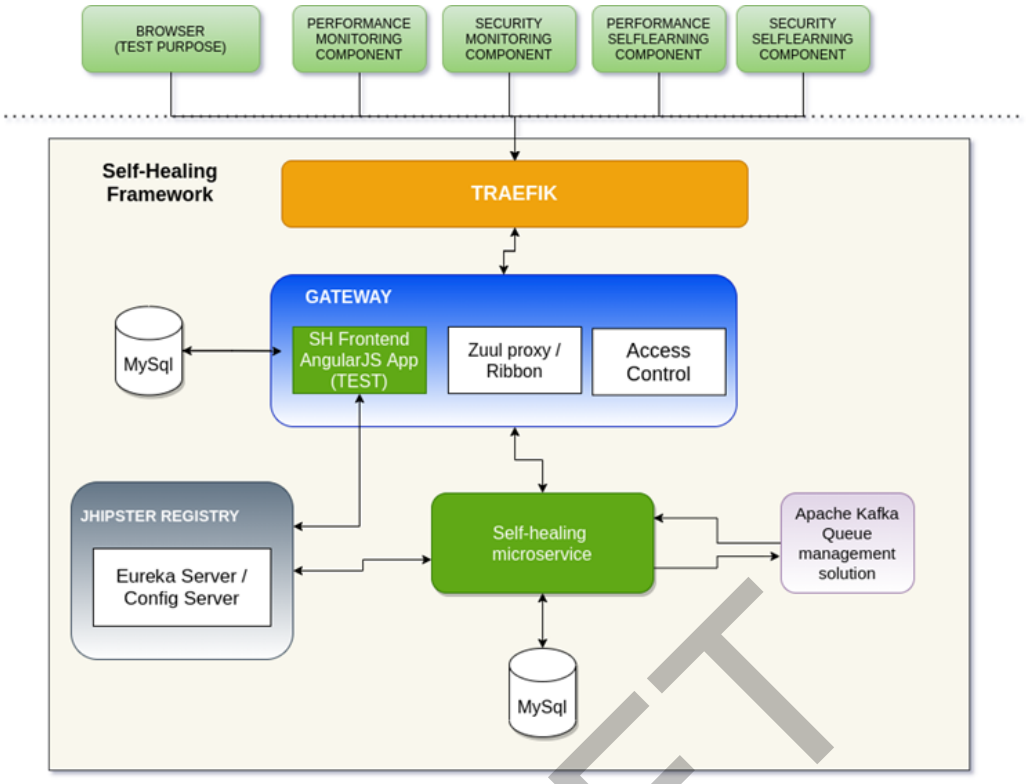

Figure 35: Self-healing internal architecture.

This second 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 on these components is provided in the upcoming Section 6.6.

- 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 self-healing:

- Access control. JSON Web Token (JWT)[20] mechanism is 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[21]. Service discovery using Netflix Eureka[22].
- Apache Kafka[23]: Event streaming solution to capture real-time data from the related components which need to send notifications to the Self-healing component.

[20] https://jwt.io/introduction
[21] https://www.jhipster.tech/
[22] https://spring.io/projects/spring-cloud-netflix
[23] https://kafka.apache.org/

5.6.4.2 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 to manage the events received is Kafka.

6 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 start the piloting of the features regarding the performance and security monitoring, self-learning and self-healing components.

During this period, we have deployed all components in the integration platform, and we have fully covered de IaC deployment from the IDE including the agent deployment and the configuration of the monitoring components to track that deployment. We have achieve the testing of the deployment in the context of one of the pilots gathering valuable feedback for the next iteration.

During the next months, we will complete in the integration and the implementation of the remaining workflows of the PIACERE frameworks and the new self-healing strategies.

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. Alonso, L. Orue-Echevarria and M. Escalante, “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] J. Alonso, M. Huarte and L. Orue-Echevarria, “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] International Organization for Standardization, “ISO/IEC 19086:1-2016 Clod computing -Service level agreement (SLA) framework,” 2016.

[6] J. Alcaraz Calero and J. Gutiérrez Aguado, “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] N. M. Fuentes-García, J. Camacho and G. Maciá-Fernández, “Present and Future of Network Security Monitoring,” *IEEE Access,* vol. 1, no. 1, p. 99, 2021.

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

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

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

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

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

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

[16] K. Pathak and J. Kapila, “Reinforcement evolutionary learning method for self-learning,” *arXiv preprint,* no. arXiv:1810.03198, 2018.

[17] T. Cerquitelli, S. Proto, F. Ventura, D. Apiletti and E. Baralis, "Automating concept-drift detection by self-evaluating predictive model degradation," *arXiv preprint,* no. arXiv:1907.08120, 2019.

[18] J. Lu, A. Liu, Y. Song and G. Zhang, "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. kishore Ramakrishnan, D. Preuveneers and Y. Berbers, "Enabling self-learning in dynamic and open IoT environments," *Procedia Computer Science,* vol. 32, pp. 207--214, 2014.

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

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

[22] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal and J. Gama, "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. Domingos and G. Hulten, "A general framework for mining massive data streams," *Journal of Computational and Graphical Statistics,* vol. 12, no. 4, pp. 945--949, 2003.

[25] Hawkins and Douglas M., Identification of Outliers, Springer Netherlands, 1980.

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

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

[28] C. Cortes and V. Vapnik, "Support Vector Networks," *Machine Learning,* vol. 20, pp. 273-297, 1995.

[29] M. M. Breunig, H.-P. Kriegel, R. T. Ng and J. Sander, "LOF: identifying density-based local outliers," in *ACM sigmod record*, 2000.

[30] 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.

[31] S. C. Tan, K. M. Ting and T. F. Liu, "Fast Anomaly Detection for Streaming Data," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, 2011.

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

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

[34] H.-K. Peng, "Multi-scale compositionality," *PLoS One,* 2015.

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

[36] T. Luo and S. Nagarajan, "Distributed Anomaly Detection Using Autoencoder Neural Networks in WSN for IoT," in *2018 IEEE International Conference on Communications (ICC)*, 2018.

[37] I. Kakanakova and S. stoyanov, "Outlier Detection via Deep Learning Architecture," in *18th International Conference on Computer Systems and Technologies*, New York, 2017.

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

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

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

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

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

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

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

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

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

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

[48] N. Nguyen Thi, V. L. Cao and 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.

[49] M. Du, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning.," in *2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[50] M. Du and F. Li., "Spell: Streaming parsing of system event logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM).*, 2016.

[51] Meng, Weibin and e. al, "LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs," *IJCAI,* 2019.

[52] Mikolov, Tomas and e. al, "Efficient estimation of word representations in vector space," *arXiv,* 2013.

[53] Zhang and Shenglin, "Syslog processing for switch failure diagnosis and prediction in datacenter networks," in *IEEE/ACM 25th International Symposium on Quality of Service (IWQoS).*, 2017.

[54] Zhang and Xu, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.

[55] P. He and e. al., "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS).*, 2017.

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

[57] A. Vaswani, "Attention is all you need," *Advances in neural information processing systems,* 2017.

[58] V.-H. Le and H. Zhang, "Log-based Anomaly Detection Without Log Parsing," *arXiv,* 2021.

[59] Devlin, Jacob and e. al., "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv 1810.04805,* 2018.

[60] Guo, Haixuan, S. Yuan and X. Wu, ""LogBERT: Log Anomaly Detection via BERT," *arXiv,* 2021.

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

[62] W. Xu and e. al, "Detecting large-scale system problems by mining console logs," in *ACM SIGOPS 22nd symposium on Operating systems principles*, 2009.

[63] "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].

[64] K. Veeramachaneni, "AI training a big data machine to defend," in *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.

[65] A. Tuor, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams.," in *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence. 2017.*, 2017.

[66] J. Glasser and B. Lindauer, "Bridging the gap: A pragmatic approach to generating insider threat data," in *2013 IEEE Security and Privacy Workshops. IEEE*, 2013.

[67] M. Bahri, A. Bifet, J. Gama, H. M. Gomes and S. Maniu, "Data stream analysis: Foundations, major tasks and tools," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* p. e1405.

[68] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering,* vol. 31, no. 12, pp. 2346--2363, 2018.

[69] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery,* vol. 30, no. 4, pp. 964--994, 2016.

[70] H. Hu, M. Kantardzic and T. S. Sethi, "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.

[71] R. S. M. Barros and S. G. T. C. Santos, "A large-scale comparison of concept drift detectors," *Information Sciences,* vol. 451, pp. 348--370, 2018.

[72] L. L. Minku and X. Yao, "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.

[73] J. L. Lobo, J. Del Ser, I. Laña, M. N. Bilbao and N. Kasabov, "Drift detection over non-stationary data streams using evolving spiking neural networks," in *International symposium on intelligent and distributed computing*, 2018.

[74] R. S. M. Barros and S. G. T. C. Santos, "A large-scale comparison of concept drift detectors," *Information Sciences,* vol. 451, pp. 348--370, 2018.

[75] J. Gama, P. Medas, G. Castillo and P. Rodrigues, "Learning with drift detection," in *Brazilian symposium on artificial intelligence*, 2004.

[76] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda and R. Morales-Bueno, "Early drift detection method," in *Fourth international workshop on knowledge discovery from data streams*, 2006.

[77] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*, 2007.

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

[79] I. Frias-Blanco, J. del Campo-Ávila, G. Ramos-Jimenez, R. Morales-Bueno, A. Ortiz-Diaz and Y. Caballero-Mota, "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.

[80] F. Pinagè, E. M. dos Santos and J. Gama, "A drift detection method based on dynamic classifier selection," *Data Mining and Knowledge Discovery,* vol. 34, no. 1, pp. 50--74, 2020.

[81] A. Liu, J. Lu and G. Zhang, "Concept Drift Detection via Equal Intensity k-means Space Partitioning," *IEEE transactions on cybernetics,* 2020.

[82] A. Pesaranghader, H. Viktor and E. Paquet, "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.

[83] T. Escovedo, A. Koshiyama, A. A. da Cruz and M. Vellasco, "DetectA: abrupt concept drift detection in non-stationary environments," *Applied Soft Computing,* vol. 62, pp. 119--133, 2018.

[84] Q.-H. Duong, H. Ramampiaro, K. Norvaag, P. Fournier-Viger and T.-L. Dam, "High utility drift detection in quantitative data streams," *Knowledge-Based Systems,* vol. 157, pp. 34--51, 2018.

[85] S. Micevska, A. Awad and S. Sakr, "SDDM: an interpretable statistical concept drift detection method for data streams," *Journal of Intelligent Information Systems,* pp. 1--26, 2021.

[86] R. F. de Mello, Y. a. G. C. H. Vaz and A. Bifet, "On learning guarantees to unsupervised concept drift detection on data streams," *Expert Systems with Applications,* vol. 117, pp. 90--102, 2019.

[87] O. A. Mahdi, E. Pardede, N. Ali and J. Cao, "Fast reaction to sudden concept drift in the absence of class labels," *Applied Sciences,* vol. 10, no. 2, p. 606, 2020.

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

[89] 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.

[90] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*, Minneapolis, 2007.

[91] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer,* vol. 36, no. 1, pp. 41--50, 2003.

[92] E. D. Nitto, L. Florio and D. A. Tamburri, "Autonomic Decentralized Microservices: The Gru Approach and Its Evaluation," *Microservices: Science and Engineering,* 2020.

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

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

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

[96] M. Azaiez and W. Chainbi, "A Multi-agent System Architecture for Self-Healing Cloud Infrastructure," in *International Conference on Internet of things and Cloud Computing*, New York, 2016.

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

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

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

[100] J. P. Magalhães and L. M. Silva, "SHõWA: A Self-Healing Framework for Web-Based Applications," in *ACM Transactions on Autonomous and Adaptive Systems*, 2015.

[101] P. K. Rajput and G. Sikka, "Multi-agent architecture for fault recovery in self-healing systems," *J Ambient Intell Human Compu,* 2021.

[102] E. Rios, E. Iturbe and M. C. Palacios, "Self-healing Multi-Cloud Application Modelling," in *12th International Conference on Availability, Reliability and Security*, New York, 2017.

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

[104] T. Wang, J. Xu, W. Zhang, Z. Gu and H. Zhong, "Self-adaptive cloud monitoring with online anomaly detection," in *Future Generation Computer Systems*, 2018.

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

[106] H. Psaier and S. Dustdar, "A survey on self-healing systems: approaches and systems," *Computing,* vol. 91, 2011.

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

[108] N. Esfahani and S. Malek, "Uncertainty in Self-Adaptive Software Systems," *Software Engineering for Self-Adaptive Systems II,* 2013.

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

[110] A. Bifet, J. Read, I. Zliobaite, B. Pfahringer and G. Holmes, "Pitfalls in benchmarking data stream classification and how to avoid them," in *Joint European conference on machine learning and knowledge discovery in databases*, 2013.

[111] S. H. Bach and M. A. Maloof, "Paired learners for concept drift," in *2008 Eighth IEEE International Conference on Data Mining*, 2008.

[112] R. S. Barros, D. R. Cabral, P. M. Goncalves Jr and S. G. Santos, "RDDM: Reactive drift detection method," *Expert Systems with Applications,* vol. 90, pp. 344--355, 2017.

[113] M. Basseville, I. V. Nikiforov and others, Detection of abrupt changes: theory and application, vol. 104, Prentice hall Englewood Cliffs, 1993.

[114] A. Bifet, "Classifier concept drift detection and the illusion of progress," in *International Conference on Artificial Intelligence and Soft Computing*, 2017.

[115] A. Bifet, "Classifier Concept Drift Detection and the Illusion of Progress," in *International Conference on Artificial Intelligence and Soft Computing*, 2017.

[116] R. S. M. de Barros, J. I. G. Hidalgo and D. R. de Lima Cabral, "Wilcoxon rank sum test drift detector," *Neurocomputing,* vol. 275, pp. 1954--1963, 2018.

[117] F. Gustafsson and F. Gustafsson, Adaptive Filtering and Change Detection, vol. 1, Citeseer, 2000.

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

[119] D. T. J. Huang, Y. S. Koh, G. Dobbie and R. Pears, "Detecting volatility shift in data streams," in *2014 IEEE International Conference on Data Mining*, 2014.

[120] K. Nishida and K. Yamauchi, "Detecting concept drift using statistical testing," in *International conference on discovery science*, 2007.

[121] A. Pesaranghader and H. L. Viktor, "Fast hoeffding drift detection method for evolving data streams," in *Joint European conference on machine learning and knowledge discovery in databases*, 2016.

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

[123] G. J. Ross, N. M. Adams, D. K. Tasoulis and D. J. Hand, "Exponentially weighted moving average charts for detecting concept drift," *Pattern recognition letters,* vol. 33, no. 2, pp. 191--198, 2012.

[124] J.-i. Takeuchi and K. Yamanishi, "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.

[125] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*, Springer, 1992, pp. 196--202.

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

[127] I. Zliobaite, A. Bifet, J. Read, B. Pfahringer and G. Holmes, “Evaluation methods and decision theory for classification of streaming data with temporal dependence,” *Machine Learning,* vol. 98, no. 3, pp. 455--482, 2015.

Annex A. Delivery and Usage

6.1 Monitoring Controller

6.1.1 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.

6.1.1.1 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
```

6.1.1.2 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
```

6.1.1.3 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
```

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 image 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
```

6.1.1.4 Vagrant

The installation with Vagrant is described at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-vagrant. This vagrant runs 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
```

6.1.2 User Manual

The Monitoring controller can be used through its REST API, described at: https://git.code.tecnalia.com/piacere/public/the-platform/runtime-monitoring/mc/-/blob/y2/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 36. The swagger UI will list the operations available and it will allow us to invoke them.

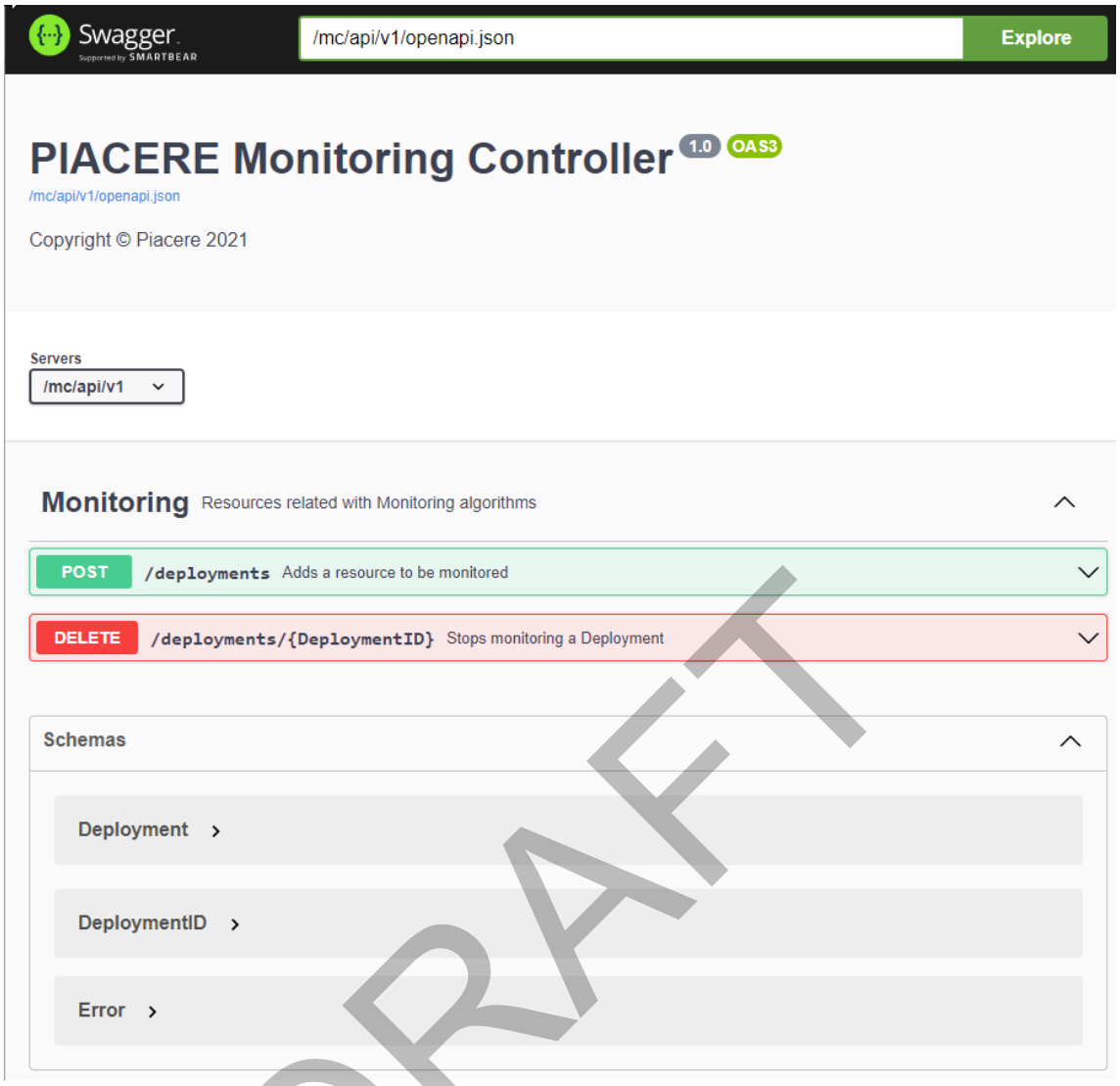

Figure 36: Monitoring Controller swagger ui.

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

6.1.3 Licensing information

To be defined

6.1.4 Download

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

6.2 Performance Monitoring

6.2.1 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.

6.2.1.1 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
```

6.2.1.2 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 WP6 components. 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 in section 6.1.1.3.

6.2.1.3 Vagrant

The installation with vagrant is described at: https://git.code.tecnalia.com/piacere/private/t61-runtime-monitoring/pm/pm-vagrant. This vagrant runs the partial integration docker-compose described above. The usage details are available above in section 6.1.1.4.

6.2.2 User Manual

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

6.2.2.1 Performance Monitoring controller

The Performance Monitoring controller is used through its REST API, described at: https://git.code.tecnalia.com/piacere/public/the-platform/runtime-monitoring/pm/-/tree/y2/git/pmc/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 37. The swagger UI will list the operations available and it will allow us to invoke them.

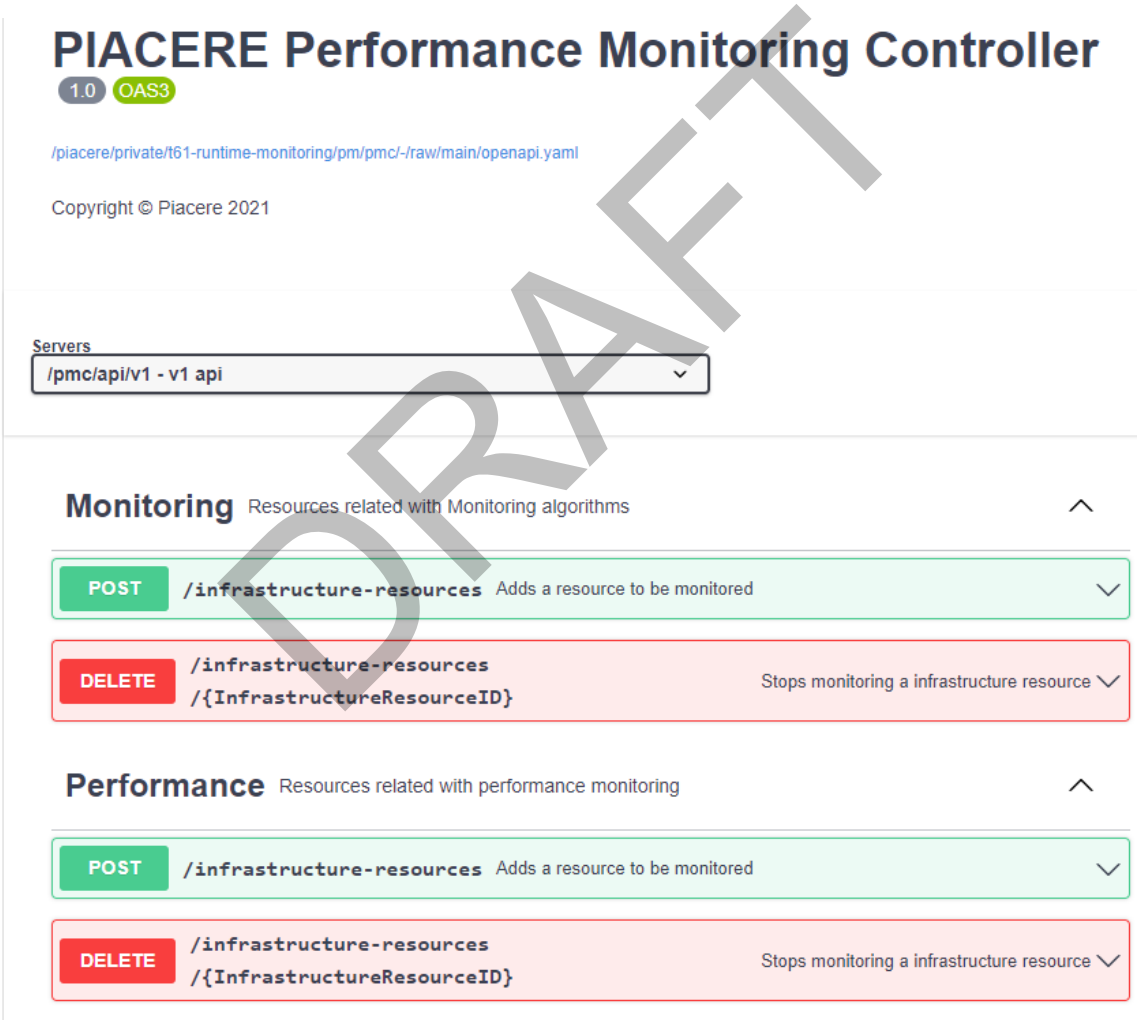

Figure 37: Performance Monitoring Controller swagger ui.

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

6.2.2.2 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 38, in our internal continuous integration framework the component is accessible at https://influxdb.piacere.esilab.org:8443/

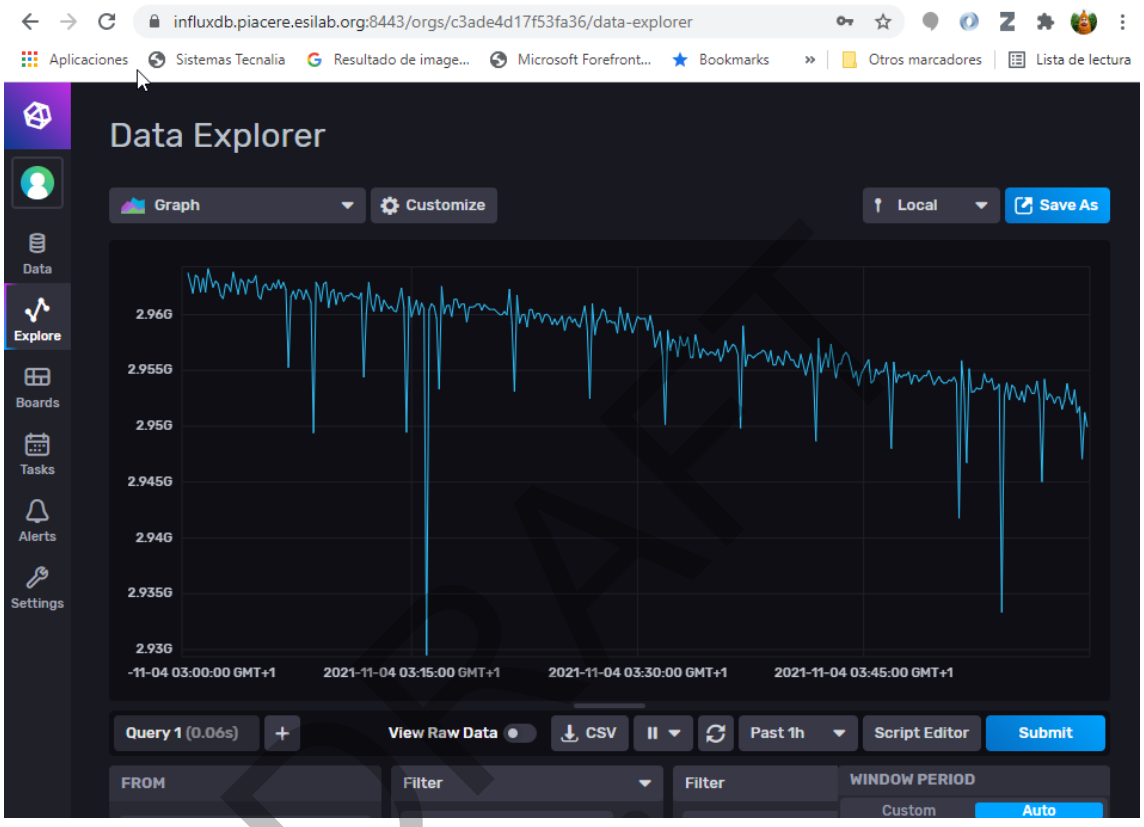

Figure 38: Influxdb.

6.2.2.3 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 39. As another example, in our internal continuous integration framework, the component is accessible at https://piacere.esilab.org:8443/grafana

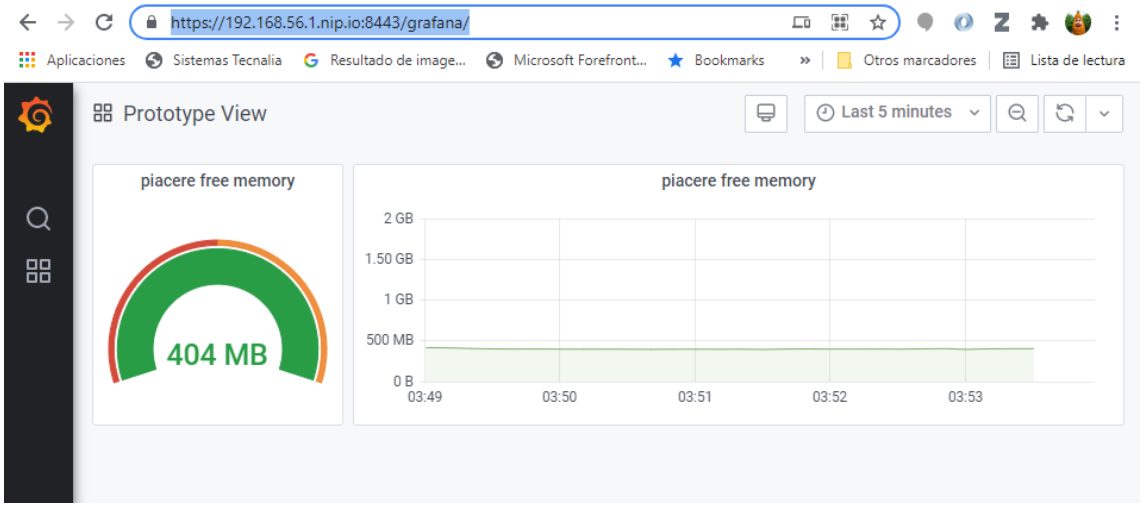

Figure 39: Grafana

6.2.3 Licensing information

Currently the code is owned by Tecnalia, and the license is still to be defined in D6.3.

6.2.4 Download

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

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. This enables the feeding of data into the piacere monitoring platform without requiring to perform real deployments that may involve some costs.

6.3 Security Monitoring

6.3.1 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 28):

- **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.
- **Security Self-learning component.** It is not part of the basic deployment package. Currently it is being offered only as SaaS model and is loosely-coupled integrated with other monitoring components.

6.3.2 Installation Instructions

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

6.3.2.1 Installing Controller

The code resides on the project's repository (also on the public counterpart https://git.code.tecnalia.com/piacere/public):

- Private: https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-controller
- Public- counterpart: https://git.code.tecnalia.com/piacere/public/the-platform/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
```

6.3.2.2 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 `v4.2.7` into the current directory:

```
$ git clone https://github.com/wazuh/wazuh-ansible.git
$ git checkout tags/v4.2.7
```

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
```

6.3.3 User Manual

6.3.3.1 Controller

Figure 40 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

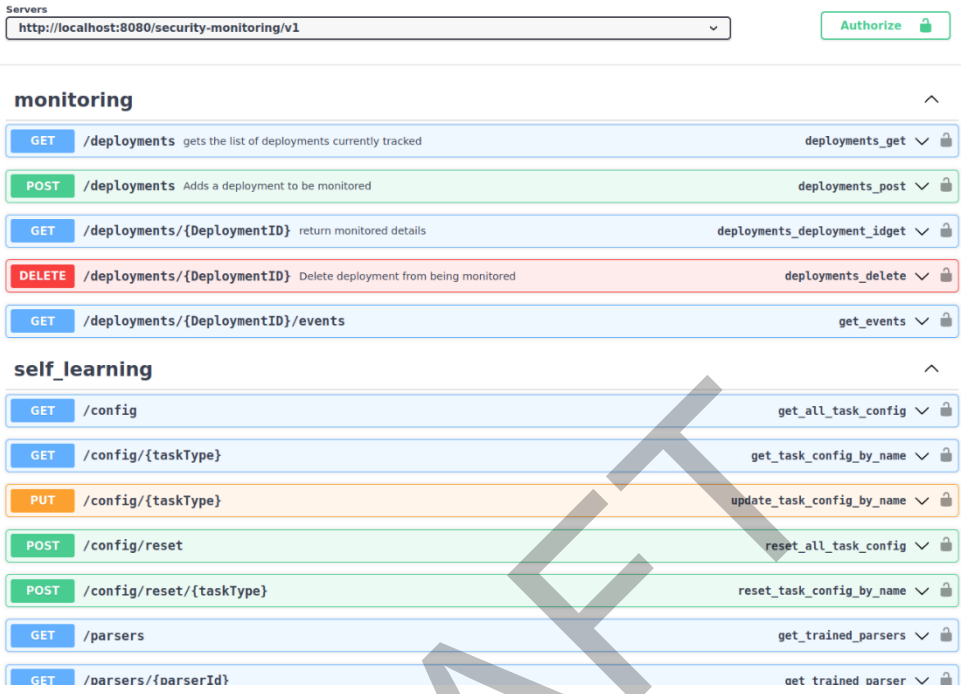

Figure 40: Security Monitoring part of the Security Monitoring Controller API.

6.3.3.2 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
```

6.3.4 Licensing information

The code is Apache 2.0 licensed.

6.3.5 Download

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

- Public repository: https://git.code.tecnalia.com/piacere/public/the-platform/runtime-security-monitoring/security-monitoring-controller

The Security Monitoring Deployment code is available here:

- Private repository: https://git.code.tecnalia.com/piacere/private/t64-runtime-security-monitoring/security-monitoring-deployment/-/tree/develop
- Public repository: https://git.code.tecnalia.com/piacere/public/the-platform/runtime-security-monitoring/security-monitoring-controller

6.4 Performance Self-learning

6.4.1 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.

6.4.2 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
```

6.4.3 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

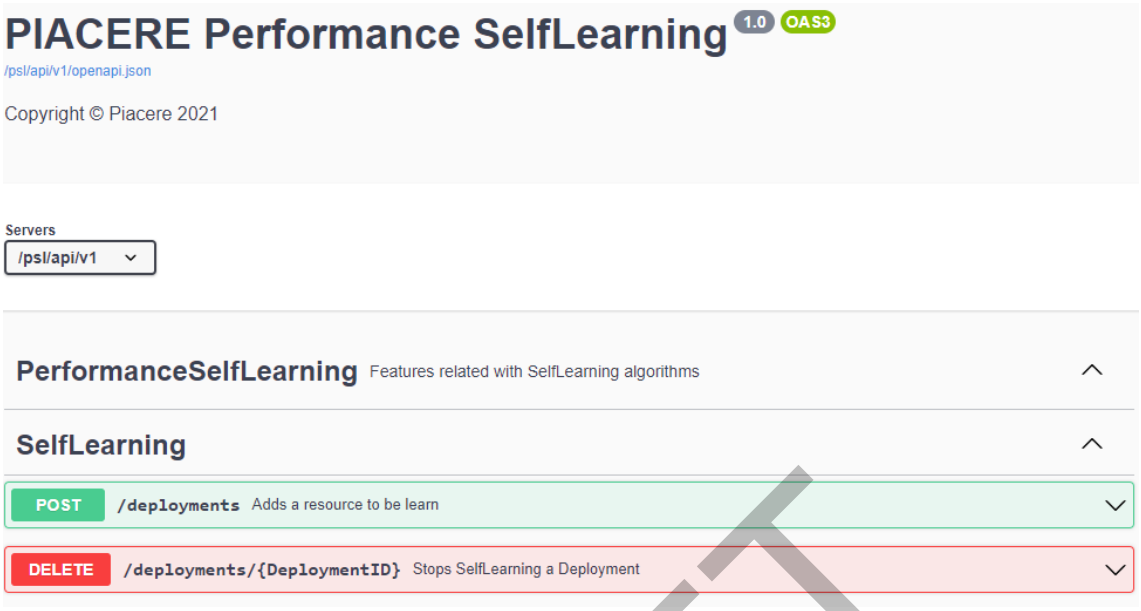

Figure 41: Performance Self-learning OpenAPI.

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'
```

6.4.4 Licensing information

Currently the code is owned by Tecnalia, and the license is still to be defined in D6.3.

6.4.5 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

6.5 Security Self-learning

6.5.1 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

6.5.2 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[24] environments are used to handle dependencies in an isolated manner. The codebase is closed and hosted on private repositories.

6.5.3 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 42) 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"
  }
]
```


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

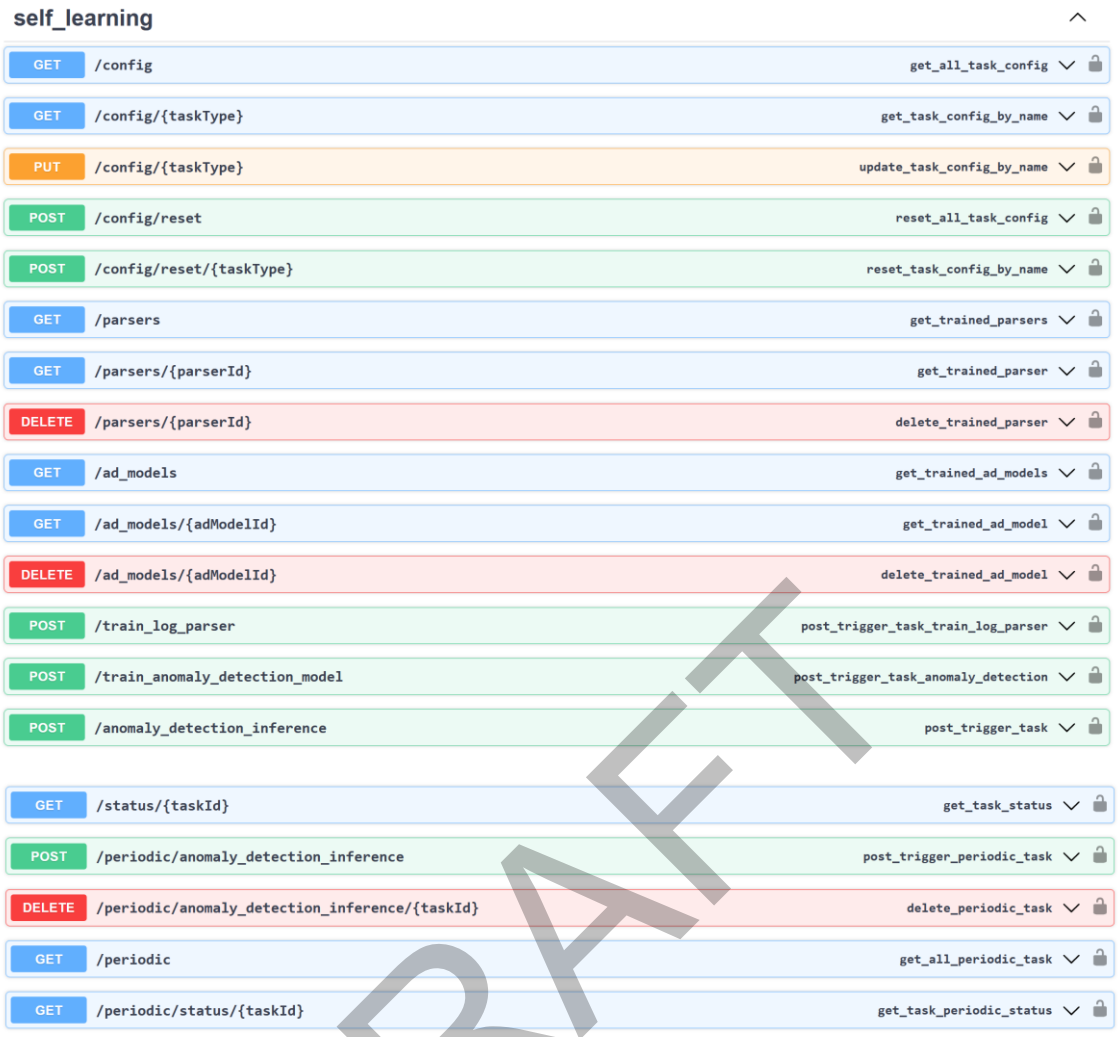

Figure 42: Self-learning API provided by Security Controller.

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.

6.5.4 Licensing information

Currently is closed source, owned by XLAB.

6.5.5 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

6.6 Self-healing

6.6.1.1 Self-healing service

The main structure of the prototype developed in this second stage of the project is composed by the packages shown in the following Figure 43.

Figure 43: 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.

6.6.1.2 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:

Figure 44: 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.

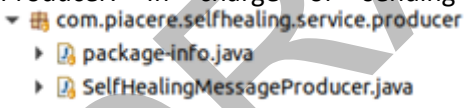

Figure 45: Self-healing producer.

  - Consumer: In charge of processing the messages asynchronously.

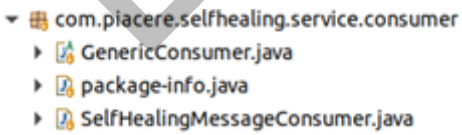

Figure 46: Self-healing consumer.

6.6.2 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

6.6.3 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.

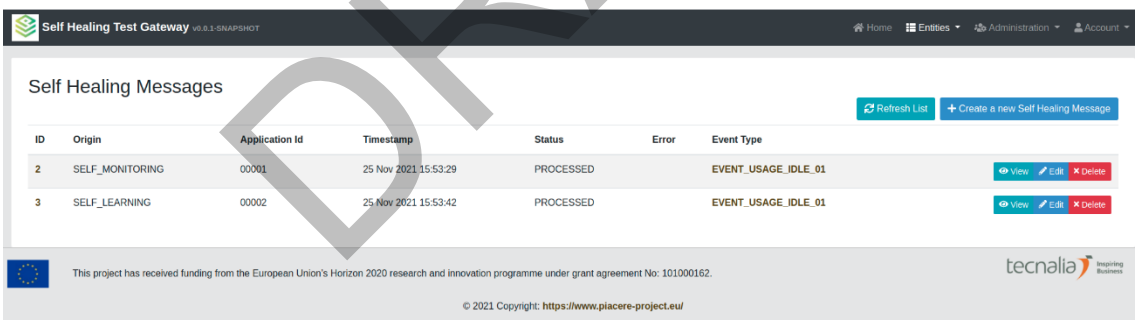

Figure 47: Messages received in the Self-Healing component.

6.6.4 Licensing information

Information about license not included yet

6.6.5 Download

The code is available in Tecnalia GitLab repository:

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