



Deliverable D4.9

Final multi-cloud application monitoring

Editor(s):	Iñaki Etxaniz
Responsible Partner:	TECNALIA
Status-Version:	Final – v1.0
Date:	27/05/2019
Distribution level (CO, PU):	CO

Project Number:	GA 726755
Project Title:	DECIDE

Title of Deliverable:	Final multi-cloud application monitoring
Due Date of Delivery to the EC:	31/05/2019

Workpackage responsible for the Deliverable:	WP4 – Continuous deployment and operation
Editor(s):	Iñaki Etxaniz (TECNALIA)
Contributor(s):	Iñaki Etxaniz, Gorka Benguria, Marisa Escalante, (TECNALIA), Javier Gavilanes (Experis)
Reviewer(s):	Lorenzo Blasi (HPE)
Approved by:	All Partners
Recommended/mandatory readers:	WP2, WP3, WP5, WP6.

Abstract:	This document accompanies the software deliverable -the final version of the engine for monitoring multi-cloud applications- and describes its technical details. It enumerates the off-shelf software products selected for the implementation, how they are integrated and the implemented interfaces.
Keyword List:	Monitoring, working conditions, micro-service, multi-cloud application.
Licensing information:	The software is released under MIT license. The document itself is delivered as a description for the European Commission about the released software, so it is not public.
Disclaimer	This deliverable reflects only the author's views and views and the Commission is not responsible for any use that may be made of the information contained therein

Document Description

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	16/03/2019	First TOC and assignments.	TECNALIA
v0.2	26/04/2019	Introduction, Abstract, Table 1	TECNALIA
v0.3	30/04/2019	Sections updated: Section 2.2, Annexes 1 and 2	TECNALIA
v0.4	17/05/2019	Sections updated: Section 3	TECNALIA, Experis
V0.5	20/05/2019	Conclusions, Exec. Summary, References. Ready for Review.	TECNALIA
V0.6	23/05/2019	Reviewer comments addressed.	TECNALIA
V1.0	27/05/2019	Ready for submission	TECNALIA

Table of Contents

Table of Contents	4
List of Figures.....	6
List of Tables.....	7
Terms and abbreviations.....	8
Executive Summary	9
1 Introduction.....	11
1.1 About this deliverable	11
1.2 Document structure	11
2 ADAPT Monitoring Manager	12
2.1 Implementation.....	12
2.1.1 Functional description.....	12
2.1.1.1 Fitting into overall DECIDE Architecture	15
2.1.2 Technical description.....	16
2.1.2.1 Prototype architecture	16
2.1.2.2 Components description	17
2.1.2.3 Technical specifications.....	18
2.2 Delivery and usage	21
2.2.1 Package information.....	21
2.2.2 Installation instructions.....	26
2.2.3 User Manual	29
2.2.3.1 ADAPT MM API.....	29
2.2.3.2 ADAPT MM UI.....	32
2.2.4 Licensing information.....	35
2.2.5 Download	35
3 Violations Handler.....	36
3.1 Implementation.....	36
3.1.1 Functional description.....	36
3.1.1.1 Fitting into overall DECIDE Architecture	38
3.1.2 Technical description.....	39
3.1.2.1 Prototype architecture	39
3.1.2.2 Components description	39
3.1.2.3 Technical specifications.....	39
3.2 Delivery and usage	41
3.2.1 Package information.....	41
3.2.2 Installation instructions.....	43
3.2.3 User Manual	43

3.2.4	Licensing information	44
3.2.5	Download	44
4	Conclusions.....	45
5	References.....	47
6	Annex1: Telegraf parser library	48
6.1	Pre-requirements	48
6.2	Downloading the Telegraf_ConfigParser library	48
6.3	Installing the library in the Maven repository.....	48
6.4	Components and plugins.....	49
6.5	Generic component structure	49
6.6	General structure of a Telegraf Plugin.	50
6.7	“HTTP Response” plugin structure	50
6.8	“Ping” plugin structure.....	51
6.9	“InfluxDB” plugin structure	51
6.10	“Mem” plugin structure	52
6.11	“Disk” plugin structure	52
6.12	“CPU” plugin structure	53
6.13	Operations on the Telegraf.conf file	53
6.14	Usage example	54
6.14.1	Capturing exceptions.....	54
6.14.2	Loading the configuration file	54
7	Annex2: iStarstopmonitoring interface specification	57
7.1	Overview.....	57
7.1.1	Version information	57
7.1.2	URI scheme.....	57
7.1.3	Tags.....	57
7.2	PathsPaths.....	57
7.2.1	createApplication	57
7.2.1.1	Description	57
7.2.1.2	Parameters	57
7.2.1.3	Responses.....	57
7.2.1.4	Consumes	57
7.2.1.5	Produces.....	57
7.2.1.6	Tags.....	57
7.2.2	getAllApplications.....	58
7.2.2.1	Description	58
7.2.2.2	Responses.....	58
7.2.2.3	Consumes	58

7.2.2.4	Produces	58
7.2.2.5	Tags.....	58
7.2.3	getApplicationstatus.....	58
7.2.3.1	Description	58
7.2.3.2	Parameters	58
7.2.3.3	Responses.....	58
7.2.3.4	Consumes	58
7.2.3.5	Produces.....	59
7.2.3.6	Tags.....	59
7.2.4	updateApplication	59
7.2.4.1	Description	59
7.2.4.2	Parameters	59
7.2.4.3	Responses.....	59
7.2.4.4	Consumes	59
7.2.4.5	Produces	59
7.2.4.6	Tags.....	59
7.2.5	deleteApplication	59
7.2.5.1	Parameters	59
7.2.5.2	Responses.....	59
7.2.5.3	Consumes	60
7.2.5.4	Produces	60
7.2.5.5	Tags.....	60
7.3	Definitions	60
7.3.1	Application.....	60
8	Annex3: Telegraf template file	60

List of Figures

FIGURE 1. ADAPT IN DECIDE ARCHITECTURE.....	15
FIGURE 2. ADAPT MONITORING M30 HIGH LEVEL ARCHITECTURE.....	16
FIGURE 3. SOURCE FOLDER STRUCTURE OF ADAPT MONITORING COMPONENT IN M30.....	22
FIGURE 4. SOURCE FOLDER STRUCTURE OF ADAPT MM CONTROL MANAGER SUB-COMPONENT	23
FIGURE 5. SOURCE FOLDER STRUCTURE OF ADAPT MM UI SUB-COMPONENT	24
FIGURE 6. SOURCE FOLDER STRUCTURE OF ADAPT MM DATA STORAGE AND AGGREGATION.....	25
FIGURE 7. SOURCE FOLDER STRUCTURE OF VAGRANT SERVER.....	25
FIGURE 8. SOURCE FOLDER STRUCTURE FOR THE SHOCK SHOP APPLICATION.	26
FIGURE 9. ADAPT MONITORING MANAGER CONFIGURATION FILE	27
FIGURE 10. DIRECTORIES OF THE SOURCE CODE FOLDER.	27
FIGURE 16. METHODS OF THE ADAPT MONITORING REST API.	30

FIGURE 11. SWAGGER EDITOR DETAIL.	30
FIGURE 18. POST METHOD OF ISTARTSTOP INTERFACE.....	31
FIGURE 19. START MONITORING RESPONSE.....	32
FIGURE 20. GRAFANA LOG IN INTERFACE.	33
FIGURE 21. DASHBOARDS TAB	33
FIGURE 22. ADAPT MONITORING AVAILABILITY DASHBOARD.	34
FIGURE 23 ADAPT MONITORING PERFORMANCE DASHBOARD	35
FIGURE 24. GENERAL ARCHITECTURE OF THE VIOLATIONS HANDLER.....	38
FIGURE 25. API INTERFACE DEFINITION FOR VIOLATION HANDLER COMPONENT.	39
FIGURE 26. MODEL OF THE VIOLATION OBJECT HANDLED BY THE API REQUESTS.	40
FIGURE 27. SOURCE CODE STRUCTURE OF THE VIOLATION HANDLER	41
FIGURE 28. STRUCTURE OF THE <i>EU.DECIDEH2020.ADAPT.VIOLATIONHANDLER.SERVER</i> MODULE	41
FIGURE 29. DATABASE <i>DB_VIOLATIONS</i> AND <i>VIOLATION</i> TABLE IN MYSQL SHELL.....	42
FIGURE 30 DOCKER COMPOSE CONFIGURATION FILE.....	43
FIGURE 31. VIOLATIONS HANDLER’S DOCKER CONTAINERS RUNNING	43
FIGURE 32. HTTP POST REQUEST BODY EXAMPLE CONTAINING THE VIOLATION INFORMATION.....	44
FIGURE 33. SWAGGER UI INTEGRATED WITHIN THE VIOLATION HANDLER.	44
FIGURE 34. FILE STRUCTURE FOR THE <i>TELEGRAF_CONFIGPARSER</i> FOLDER.....	48
FIGURE 35. <i>TELEGRAF_CONFIGPARSER</i> LIBRARY	48
FIGURE 36. <i>TELEGRAF.CONF</i> COMPONENTS STRUCTURE.	49
FIGURE 37. JAVA CLASSES WHICH IMPLEMENT THE DIFFERENT OPERATIONS TO BE PERFORMED IN A <i>TELEGRAF.CONF</i> FILE	53

List of Tables

TABLE 1. FUNCTIONALITY COVERED BY THE M30 ADAPT MONITORING PROTOTYPE.	13
TABLE 2. CONTAINERS DEPLOYED BYTHE ADAPT MONITORING MANAGER PROTOTYPE.....	26
TABLE 3. FUNCTIONALITY COVERED BY THE M30 VIOLATIONS HANDLER PROTOTYPE.	37

Terms and abbreviations

EC	European Commission
M12	Month twelve
M24	Month twenty four
M30	Month thirty
D	Deliverable
NFR	Non Functional Requirement
F	Functionality
MTBF	Mean Time Between Failures
MTTR	Mean Time To Recovery
SRC	Source Code
VH	Violation Handlers
ADAPT MM	ADAPT Monitoring Manager
MCSLA SLOs	Multi Cloud Service Level Agreement Service Level Objectives
WP	Work Package
MIT	Massachusetts Institute of Technology
REST	Representational State Transfer
API	Application Programming Interface
URL	Uniform Resource Locator
ADAPT DO	ADAPT Deployment Orchestrator
URI	Uniform Resource Identifier
JSON	JavaScript Object Notation
CRUD	Create, Read, Update, Remove
ACSml	Advance Cloud Service meta-Intermediator
UI	User Interface
TOML	Tom's Obvious, Minimal Language

Executive Summary

This document describes the two components, ADAPT Monitoring Manager (ADAPT MM) and Violations Handler (VH), forming the final version of the multi-cloud application monitoring prototype, which monitors the deployed multi-cloud application in order to verify its working conditions, and manages the actions to be performed when a violation occurs.

The document presents the missions, scopes, functional descriptions, architecture and technical approaches. Also, it contains the download and installation instructions of the source code available at the DECIDE public repository (DECIDE_components/ADAPT/Monitoring - tag M30), and the user manuals of these components comprising the prototype.

This document describes the M30 release of the ADAPT Monitoring Manager and the Violation Handler components and updates the one delivered with the M24 release [1], including the new functionalities implemented for the final version.

The final prototype (M30) of ADAPT Monitoring Manager implements new features with respect to the M24 version prototype, which are summarized in what follows, and detailed in other sections of the document:

- The ADAPT MM Control Manager sub-component: which manages the flow of actions into ADAPT monitoring, i.e: calls to the correspondent sub-components and includes now the implementation of the function that stops the monitoring of an application, in the interface with ADAPT DO.
- Automatic configuration of ADAPT MM Data Collection sub-component, through the implementation of an ad-hoc library to edit the configuration files of the telegraf tool.
- ADAPT MM Violations Detection sub-component, that now includes the integration with the DECIDE MCSLA editor tool to calculate the fulfilling of MCSLAs based on the defined metrics, and the interface with the Violations Handler.
- Automatic configuration of the new dashboards in ADAPT MM GUI Manager, based on the microservices that compose the multi-cloud application, and in the deployment specificities (virtual machines, containers, ports, etc.), all gathered from the Application description. The Application description specification has been modified to include a new structure for the *safemethods*, which now are defined as an object instead of a simple string URL. The dashboards now include a graphical view of the metrics for the availability and performance NFRs, as well as a numerical representation of them.
- Integration with the Application Description, which is the main DECIDE integration mechanism between the different tools. It is used to obtain all the information of the application needed for the monitoring, and to write on it the status of the monitorization.
- Integration with the DevOps framework, where the dashboards created by ADAP MM are to be shown. This integration is made through the Application Description, where the location of the dashboards is inserted. The DevOps framework reads this field to integrate the monitorization in its visual interface.

The final prototype (M30) of Violation Handler implements the following new features with respect to the M24 version prototype:

- Management of the restart of the workflow for redeployment. When a technologically high-risk violation is received, it will trigger OPTIMUS and inform it about the violation, so that the reason of the violation can be taken into account when simulating new deployment configurations. Furthermore, the user is notified through an email sent to him.

- Integration of the Vault tool to store and manage the secrets of the components. In this case, the user/password of the git repository, that are needed to access the application description.

1 Introduction

1.1 About this deliverable

This deliverable is produced by the Task 4.3, *Multi-cloud application monitoring*. This document is a complement to the software prototype of the same name -Final multi-cloud application monitoring- which is to be delivered at time specified at the head of the document, namely, 31/05/2019.

The document updates and extends the description of existing functionalities and appends the new developments having taken place since the previous version of the deliverable at M24 [1].

The document describes the implementation of the two components that compose the final version of the multi-cloud application monitoring prototype, which is in charge of monitoring the multi-cloud application deployed. This monitoring aims to verify that the application is working in accordance with the agreed conditions and that, in case a violation of these conditions occur, triggers the actions to be performed by the rest of the components of DECIDE (a redeployment, for example) to maintain the application working as desired.

1.2 Document structure

The first section – Introduction - describes the purpose and structure of the document. The following two sections, sections 2 and 3, describe the two components of the prototype. The architectural and implementation details of the components, their functionality and interfaces, and how to use and download them are described in detail. Section 2 is dedicated to ADAPT Monitoring Manager, and section 3 describes the Violations Handler.

The document also includes three Annexes: Annex1 describing the Telegraf parser library, implemented in the context of DECIDE project; Annex2 specifying the iStartStopmonitoring interface that constitutes the Monitoring Manager component API; and Annex3 containing a template of telegraf configuration file.

2 ADAPT Monitoring Manager

2.1 Implementation

2.1.1 Functional description

The DECIDE ADAPT MM components monitor the deployed multi-cloud based application and verify that the non-functional requirements and the SLOs are being fulfilled. If a violation of any of the NFRs or SLOs defined in the MCSLA is detected, ADAPT MM components will inform the Violations Handler component -described in section 3- which will generate the proper actions depending on each situation and context. If the violation occurs, information indicating that the working conditions are not met will be sent to the operator. Besides, if the application technology risk is low, the “adaptation” process will be launched, through the Violations Handler component.

The main functionalities of the ADAPT MM component are:

- F1. **Collect data from the deployed multi-cloud application:** Once a multi-cloud application is deployed in certain resources, ADAPT MM shall start the monitoring process. The ADAPT MM component gets the data from the deployed components to assess their service level metrics at real time. Meanwhile, the data related to the cloud services where the components of the application are deployed, will be collected by ACSmI. For this, ADAPT MM triggers ACSmI so that it can start the monitoring and assessment of the CSPs SLOs. The NFRs to be covered in the context of DECIDE project are: Performance, Availability, Security/Legal, Cost, Location, and State. From these, ADAPT monitoring will assess the SLOs of Performance Availability and Cost of the Application MCSLA.
- F2. **Store the data collected from the deployed multi-cloud application:** To assess the working conditions of the multi-cloud application, ADAPT MM will deal with considerable amounts of data at real time. This data will be stored in a time series database for further analysis (i.e. aggregation, min-max values, etc.).
- F3. **Create the aggregated data to be assessed:** from the raw metrics (e.g. http response time), the ADAPT MM component will need to create aggregated data to assess the violations. For example, the final measured NFR (e.g. Availability) calculated per component or the aggregated value for the multi-cloud application as a whole.
- F4. **Visualize the measurements:** DECIDE ADAPT MM will be in charge of providing the user with a graphical interface to visualize the monitored data of the application. DECIDE ADAPT MM will also provide monitoring information of the violations occurred.
- F5. **Detect when a violation occurs:** DECIDE ADAPT MM will detect violations on the working conditions (MCSLA SLOs) and launch the adaptation process through a call to the Violations Handler which, in turn, will send the corresponding alert to the operator. This adaptation should include the ending of the monitoring of the previous components and resources.

The existing approaches for application monitoring [2] do not tackle the monitoring of different elements of the application (components) and the fact that they are deployed into disperse cloud resources. They usually face the problem of monitoring single components, or entire software applications. In DECIDE ADAPT MM, the working conditions of a multi-cloud application with respect to specific NFRs (although the approach is extendable to others) are supported, and addressed at different levels [3] [4] (application, resources).

ADAPT MM has been implemented following an incremental approach, adding features in the successive releases of the tool (D4.7 was delivered in M12, D4.8 in M24, and this document, which is delivered in M30). In this final release, all the main functionalities are implemented. Furthermore, for testability and demonstration purposes, we present them using an example application (Socks Shop) as a target.

The following table details the relationship between the functional requirements -indicated in deliverable D4.3 [5] for year 2- and the implemented functionalities, giving a description of the coverage of each functionality.

Requirements covered by the prototype:

Table 1. Functionality covered by the M30 ADAPT monitoring prototype.

Functionality	Req. ID	Coverage	Status
F1	WP4-MR3, WP4-MR11, WP4-MR13	<p>ADAPT MM is able to receive the request from ADAPT DO to start the monitoring. It reads the SLOs included in the Application description to determine the NFRs to be monitored, and it can monitor the following NFRs: Performance and Availability at application level.</p> <p>In this release, the ADAPT MM includes the interface to stop the monitoring of a given application, as well as the rest of the interface: the functions to get the status or update an application and get all applications.</p> <p>Also, the Application Description has been extended to re-define the “safe methods”, that will be defined by the user to monitor the microservices that compose the application.</p> <p>Finally, the Application Description is scanned to obtain both the safe methods defined, as well as the deployment information, to be able to monitor the microservices whatever is the application topology after deployment.</p>	Implemented. Testing integration
F2	WP4-MR3, WP4-MR11, WP4-MR13	<p>The metrics collected are stored in a database.</p> <p>In this final release configures automatically the tool to (i) collect the needed metrics from the services of the application, independently of where or how it is deployed; (ii) store these metrics in the corresponding data base (time series data base) tagged to their corresponding application. For this, ADAPT MM configures the input and output plugins of the telegraf tool.</p> <p>All this configuration is done based on the information contained in the Application description.</p>	Satisfied.
F3	WP4-MR3, WP4-MR11, WP4-MR14	<p>The component gets the raw real time metrics to value (performance, availability) per component.</p> <p>In this release, the component also generates the aggregated value (performance, availability) per component-using the mcsla-core library- and is able to aggregate the values to calculate the metrics for the whole application.</p>	Satisfied.

Functionality	Req. ID	Coverage	Status
F4	WP4-MR3, WP4-MR11	In this release, the dashboards are generated automatically depending on the information found on the Application Description and configured inside the visualization tool (Grafana). DECIDE ADAPT MM also provides monitoring information of the violations occurred. Besides, the location of the dashboards is introduced in the Application description to let the DevOps Framework to include them in its GUI.	Implemented. Testing integration
F5	WP4-MR1, WP4-MR2, WP4-MR3, WP4-MR10	In this release, the real data stored in the time series database – influxdb - is queried, aggregated and compared with the SLO established in the MCSLA at application level. This is done periodically for every application deployed, and, if a violation of the SLO is found, generates a request to the VH. The parameters of the interface with the Violation Handler have been updated in this period to align the data with the changes on the rest of components.	Satisfied. Implemented. Testing integration

Detailed functionality that this prototype offers:

The delivered prototype is the Final version of the ADAPT MM component, and contains the following new features:

- The ADAPT MM Control Manager sub-component includes, in the interface with ADAPT DO, the function that stops the monitoring of an application. For this, the list of application being monitored is scanned, the data collection tool is re-configured to stop collecting the corresponding data and all the resources associated with this specific application are released.
- Automatic configuration of ADAPT MM Data Collection sub-component, through the implementation and use of an ad-hoc library (see *Annex 1: Telegraf parser* library) to edit the configuration file of the Telegraf tool. An input plugin (HTTP_RESPONSE) and an output plugin (INFUXDB) are configured (see 2.1.2.3 *Technical specifications*, and *Annex 3: Telegraf template* file for more details). After this configuration, the Telegraf tool must be re-started to read the new configuration. This is done by ADAPT MM, and it is the ultimate reason which has led us to deploy both Telegraf and ADAPT MM in the same container.
- The ADAPT MM Violations Detection sub-component includes the integration with the DECIDE MCSLA editor tool to calculate the fulfilling of MCSLAs based on the defined metrics. The metrics for Availability are: MTBF (Mean time between failures) and MTTR (Mean time to recover). To calculate these metrics, a continuous monitoring of the http_result value is done, for each microservice of the application. When the status changes from success to timeout/error, a *Down event* of the service is detected. When the status is successful again, an *Up event* is detected. Three consecutive values (Up-Down-Up) of timestamp are passed in to the mcsla-core library to calculate the metrics MTBF and MTTR (more details on the Availability metrics are shown in the section 2.1.2.3 *Technical specifications*)
- The interface with the Violations Handler has been implemented and tested, and violations are now reported. Also, the interface with the ACSml monitoring component has been implemented, so the call to start monitoring the multi-cloud resources is done.

- The Application description specification has been modified to include a new structure for the *safemethods*, which now are defined as an object instead of a simple string URL. (More details on the *safemethods* definition and how the deployment affects them in the description file can be found in the paragraph titled *Application description* below).
- Improved integration with the Application Description, which is the main DECIDE integration mechanism between the different tools. It is used to obtain all the information of the application needed for the monitoring, and to write on it the status of the monitorization. The deployment configuration is parsed (virtual machines, containers, ports, etc.) to obtain the final URL of the safe methods after each deployment of the application.
- Automatic configuration of the new dashboards in ADAPT MM GUI Manager, based on the microservices that compose the multi-cloud application, and in the deployment specificities, all gathered from the Application description. The dashboards now include a graphical view of the metrics for the availability and performance NFRs, as well as a numerical representation of them. The configuration file is read by Grafana periodically, so no restart is needed in this case.
- Integration with the DevOps framework, where the dashboards created by ADAP MM are to be shown. This integration is made through the Application Description, where the location of the dashboards is inserted. The DevOps framework reads this field to integrate the monitorization in its visual interface.

2.1.1.1 Fitting into overall DECIDE Architecture

ADAPT MM is one of the final steps in the DECIDE workflow [6]. It supports the operation phase of multi-cloud aware applications by providing means for run-time monitoring of the current deployment, with respect to selected non-functional requirements and SLOs, and re-deployment adaptation when needed. The ADAPT MM is one of the sub-components of the ADAPT key result of the DECIDE architecture, indeed it is one of the components inside this package (see figure 1). In this picture, for the sake of clarity, only the interfaces that are used/implemented by ADAPT have been depicted. The whole figure with all the interfaces between the different tools can be found in D2.5 [7].

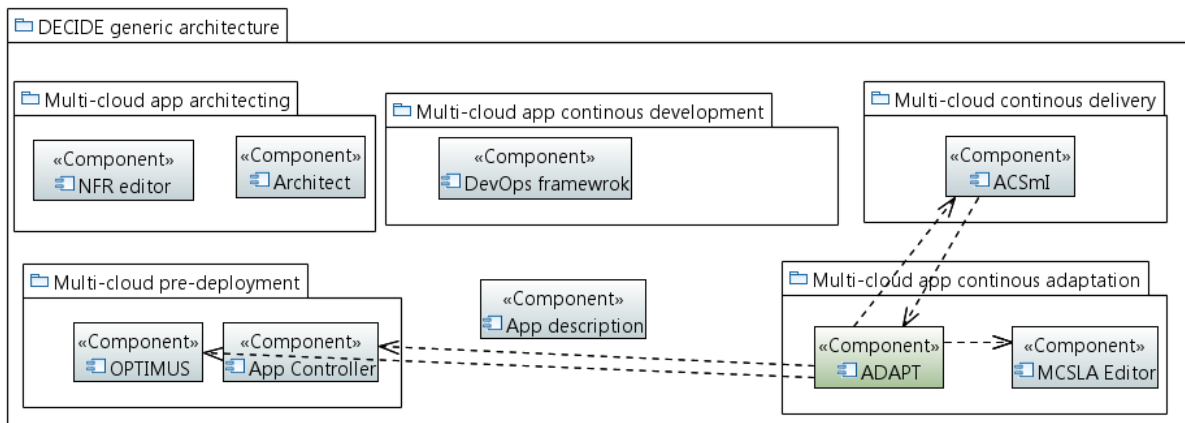


Figure 1. ADAPT in DECIDE architecture.

The ADAPT tool is formed by three components: Deployment Orchestrator, Monitoring Manager and Violations Handler. ADAPT MM interacts, on the one hand, with the two other components of ADAPT (Violations Handler and ADAPT Deployment Orchestrator) and, on the other hand, with other tools in the DECIDE framework, namely ACSmI, the MCSLA Editor, App Controller and OPTIMUS.

More specifically, ADAPT MM communicates with the MCSLA Editor/ Application Controller to gather the thresholds and provide the measured values for the different metrics to be assessed, with the

Violations Handler to provide information about any violation that could happen; and finally, with ACSml for requesting to start / stop the monitoring of the service providers SLOs. ADAPT MM also receives from the Deployment Orchestrator the requests to start/stop the monitoring.

The external interaction with the OPTIMUS component corresponds to the Violations Handler, to trigger deployment simulations.

2.1.2 Technical description

This section describes the technical details of the implemented software for the current prototype (M30) of ADAPT MM.

2.1.2.1 Prototype architecture

In the following picture, the general architecture of the ADAPT MM components is shown. It details the internal interactions among the components as well as the external interactions with other components through different interfaces. The external components are coloured differently depending if they belong to ADAPT or are part of other packages. For a detailed description, check D4.3 [5]:

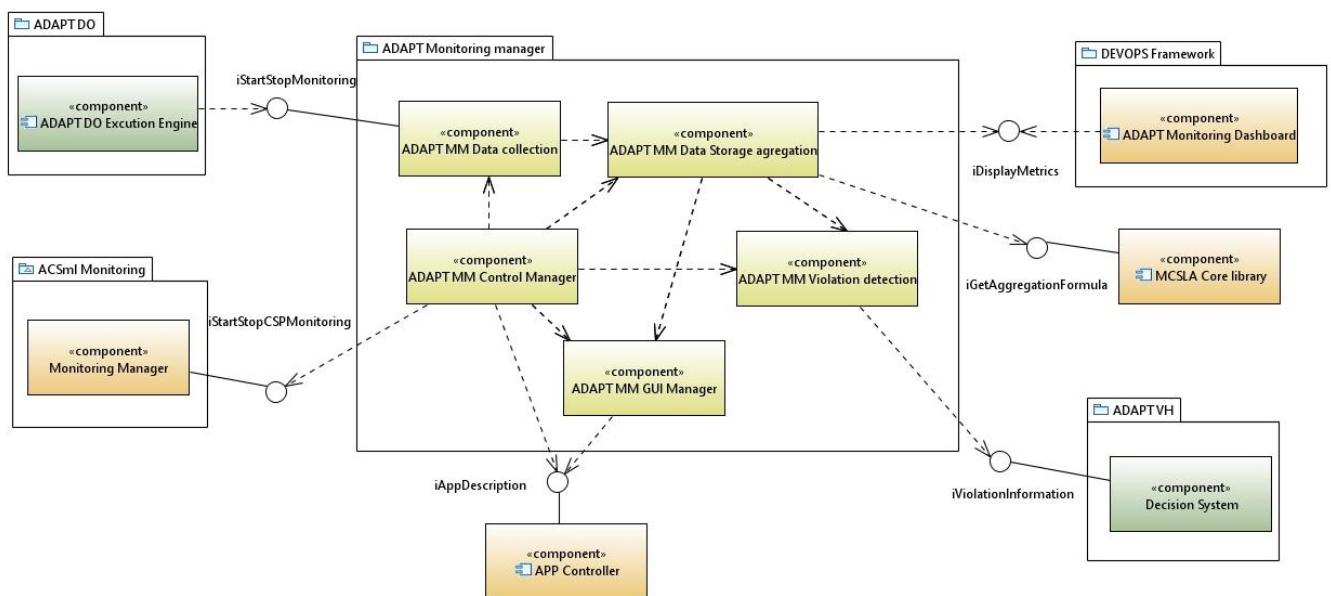


Figure 2. ADAPT Monitoring M30 High level architecture.

INTERFACES

This prototype implements or uses several interfaces, that are listed below with a brief explanation of their usage:

- **iStartStopMonitoring:** ADAPT MM Control Manager implements this REST interface that enables to start/stop of the monitoring process by other components (i.e. ADAPT DO). The specification of this interface is included in Annex 2.
- **iStartStopCSPMonitoring:** ADAPT MM Control Manager consumes this REST interface that enables to trigger the start/stop of the monitoring of the cloud resources by the ACSml components.
- **iViolationInformation:** ADAPT MM Control Manager consumes this REST interface implemented by ViolationHandler component, that enables to provide information about the violations of the MCSLAs produced during the monitoring of the application. The specification of this interface is included in Annex 2.

- **iGetAggregationFormula:** ADAPT MM Control Manager consumes this interface, implemented by MCSLA editor, to get the aggregation formula to assess the fulfillment of the MCSLA. This interface is implemented as a JAVA library [8], which is used as part of the ADAPT MM Data storage and aggregation component.
- **iAppDescription:** ADAPT MM Control Manager consumes this interface, implemented by Application Controller, to get the information of the application contained in the Application Description file. Also, as another of the tasks due to monitoring, the ADAPT MM GUI Manager consumes this interface to modify the same file introducing the URL of the dashboard to be shown in the DevOps Framework.
- **iDisplayMetrics:** this interface represents the monitoring data that goes from the ADAPT MM Data Storage component to the DevOps Framework GUI. Actually, no call is made to transfer this data, because once the Grafana dashboards have been defined and configured by the ADAPT GUI Manager component (which includes the data source, the query to obtain the data, the type of graphic representation, etc.), the data is automatically gathered from the tool and displayed without any further intervention.

2.1.2.2 Components description

The current prototype includes the 5 sub-components envisioned for ADAPT monitoring [5].

- **ADAPT MM Data Collection:** This sub-component collects the data from the multi-cloud application with respect to its working conditions. The Data collection component gets these metrics for availability and performance every 10 second in the actual configuration. It uses the *Telegraf* tool for this task. Telegraf is automatically configured to obtain the needed information of the application components and microservices, based on the information gathered from the Application description. After the configuration is made, the tool extracts the information and sends it to the selected output, in our case, a time series database.
- **ADAPT MM Data Storage and Aggregation:** This sub-component is in charge of storing the data collected from the ADAPT MM Data collection and aggregating them to create the actual measures that will be assessed by the ADAPT MM violation detection. It uses the *iGetaggregationformula* interface implemented by MCSLA Editor/Application Manager to get the aggregation formula to assess the fulfillment of the MCSLA. The storage tool used in DECIDE is the time series database *InfluxDB*. Queries to extract the needed information from the database are automatically created by the Data Storage and Aggregation and periodically issued. Also, when required, some queries to enter data are issued (for example, to store information about the violations produced).
- **ADAPT MM Control Manager:** This component manages all the activities of ADAPT MM. It receives the request to start the monitoring of the application. Then, ADAPT MM Control Manager configures the different agents in the ADAPT MM Data Collection, sets up the ADAPT MM Data Storage and Aggregation and starts the periodic assessment of the values of the different NFRs with respect to the MCSLA SLOs. To do all this, it uses the *iAppDescription* interface to read/write in the Application description file.
- **ADAPT MM GUI Manager:** This is the ADAPT MM component that configures and manages the graphical user interface for the visualization of metrics. In the final version of the prototype, this graphical interface includes the metrics of the availability and performance of the different components in real time. Also, the cost information and the alarms (violations) are displayed.
- **ADAPT MM Violations Detection:** This subcomponent assesses the aggregated metrics - previously composed based on the measured values- against their respective threshold, defined in the established MCSLA. It raises an alarm if a violation occurs, reporting it to Violation Handler via the *iViolationInformation* interface.

2.1.2.3 Technical specifications

ADAPT monitoring prototype for M30 has been implemented as a monitoring stack, covering the data collection, the data storage, data aggregation, MCSLA assessment and the data visualization. In this section, more technical details of these functionalities are explained.

Raw metrics for availability and performance NFRs

For the data collection the Telegraf open source technology is used. Telegraf is a plugin-driven server agent written in Go to collect, process, aggregate, and report metrics. It is a compiled and standalone binary that can be executed on any system with no need for external dependencies. For the purpose of ADAPT MM, the following Telegraf plugins have been configured:

- **Input plugin:** HTTP_RESPONSE plugin, which tests HTTP/HTTPS connections to a given list of URLs (for testing purposes, the Sock Shop application components above cited has been used), with a given periodicity, from a set interface. This plugin gets, among others, the following parameters:
 - http_response_code (int): the code received
 - response_time (float, seconds): Component response time.
 - result_type (string): success, timeout, response_string_mismatch, connection_failed.
 - server (string) is the URL of the service whose values are monitored.
- **Output plugin:** INFLUXDB plugin. This plugin sends the metrics gathered by the agent to an Influx DB instance. Among the parameters to be set are:
 - address: URL of the database server
 - database: name of the database
 - username, password: login parameters

These metrics are monitored per Multi-cloud app component (i.e. microservice). The compilation of these raw metrics is monitored in order to set up the final Availability [3] and Performance metric of the Multi cloud application. These compiled metrics for Availability are:

- MTBF: Mean time between failures.
- MTTR Mean time to recover.

Availability at software level in DECIDE.

- “Availability” is considered as a function of time, defined as the probability that system is operating correctly and is available to perform its function at the instant of time “t”.
- Metrics for availability at SW level in DECIDE:
 - $A = \text{MTBF} / (\text{MTBF} + \text{MTTR})$
- Availability is calculated per component and aggregated based on the MCSLA definition.

Performance at software level in DECIDE.

- “Performance” is an indicator of how well a software system or component meets its requirements for timeliness. Timeliness is measured in terms of response time or throughput.
- The response time is the time required to respond to a request. It may be the time required for a single transaction, or the end-to-end time for a user task.
- The throughput of a system is the number of requests that can be processed in some specified time interval.
- Metrics for performance at SW level in DECIDE:

- Response time: Response time is measured at component level and aggregated based on the MCSLA definition.
- Calculates the value of the availability and performance metrics of the multi-cloud application, based on the aggregation expression established in the MCSLA.

Data Storage and Aggregation.

For the storage of the metrics, Influx DB storage technology has been selected. InfluxDB is a data store for cases involving large amounts of timestamped data, like monitoring applications or micro-services, which is the case of ADAPT MM. Influx DB supports plugins for data ingestion protocols, like Telegraf. The measurements are injected by the Telegraf plugins in the Influx DB database automatically. Apart from that, this component also inserts other event information in the database, as is the case with the violations occurred, for example.

ADAPT MM Control Manager.

The ADAPT MM Control Manager has been developed from scratch. It is a Java program that controls the different activities to be performed and the workflow. It is able to start and programmatically configure all the elements for the monitoring, based on the information from the Application Description. It is deployed using a Jetty server¹, and receives the orders to start/stop the monitoring of a specific application from ADAPT DO.

GUI

For the generation of the user interface, where the metrics related to the current working conditions of the application are monitored, Grafana² is used. Grafana is an open platform to visualize data. It provides means to configure specific dashboards with different graphics. This component is in charge of configuring Grafana to automatically read the information from the InfluxDB and show it in the defined dashboards. For this final version, two dashboards are defined and created: one to contain the selected availability metrics per component (micro-services), and the other for metrics related to performance.

The dashboards designed in this version include the following graphics:

- Availability dashboard: The code of the response received, per component (every 10s). It includes the numeric value of availability
- Performance dashboard:
 - Mean response time per app component in the same graphic (it includes the current, maximum and minimum values too).
 - Mean response time per app component in separated graphics (it includes the current, maximum and minimum values).
 - Maximum response time in the application (considering the different components).
 - Maximum response time in the components (considering metrics in the last hour).

Violations Detection.

¹ <https://www.eclipse.org/jetty/>

² <https://grafana.com/>

The ADAPT MM Violations Detection has been developed as a module inside ADAPT MM Control manager. This module gets the aggregation formula to assess the fulfillment of the MCSLA, based on the monitored metrics. For this, it uses the library implemented by MCSLA editor [8].

Application Description

The Application Description (AD) is the main way of exchanging complex information between all DECIDE tools. This section summarizes the information exchanged by ADAPT MM through the AD.

Microservices.Safemethods:

Safemethods are a list of uris that are defined by the developer to monitor each microservice of the application, without interfering in the state of the application itself. Previously defined as string urls, finally the decision was to represent them only with the dupla {path, port} for clarity purposes, and because the rest of the information in the url (protocol, domain) is deemed as not relevant, especially after the deployment of the services.

Example

```
"safeMethods" : {
  "path" : "/customers",
  "port" : "18082"
},
```

Microservices.name, Containers.containerName:

The name of the microservice and the name of the container in which it is deployed are the same, by consensus. These fields are used to navigate from the microservices to the corresponding container.

Containers.dockerHostNameName, virtualMachines. dockerHostNameName:

These fields of the same name in containers and virtual machines are used by the ADAPT MM to know in which virtual machine is deployed each container.

virtualMachines. dockerHostPublicIP:

This field contains the IP of the virtual machine, that is needed by the ADAPT MM to form the safemethods URL after deployment (roughly is IP + port + path), and thus to know where to address the inquiries to monitor the corresponding safemethods.

Containers.portMapping:

This field contains a series of mapping of ports that each container can have defined, so that an incoming TCP request is conducted from the port used in the external call (hostPort) to another port inside the container (containerPort). This mapping is needed for the ADAPT MM to know which port it has to use to call from the outside to a safemethod located in that container.

Example

```
"containerName" : "catalogue"
...
"dockerHostNameName" : "node-2",
...
"dockerHostPublicIp" : "82.223.2.216",
...
"portMapping" : [ {
```

```

    "hostPort" : "8480",
    "containerPort" : "80"
  } ],

```

Nfrs:

This field contains a list of NFRs (Non-Functional Requirements) of the application. ADAPT MM needs to know which those requirements are, and which one it has to control the measurements against. The type field indicates the kind of nfr, being the “Availability” and “Performance” types those that are controlled by ADAPT MM .

Example

```

"nfrs" : [ {
  "type" : "Performance",
  "tags" : [ "Application" ],
  "abstractValue" : "High",
  "value" : 100.0,
  "unit" : "milliseconds"
}, {
  "type" : "Availability",
  "tags" : [ "Application" ],
  "abstractValue" : "Medium",
  "value" : 98.0,
  "unit" : "percentage"
} ],

```

Monitoring:

This part of the AD is a composed field, that contains the “status” of the monitoring (true if the application is being monitored / false if not); and a list of the Grafana dashboard urls, that is included in the “url” field. This list is used by the Framework to insert these dashboards in its GUI.

Example

```

"monitoring" : {
  "status" : true,
  "urls" : [ "http://85.91.40.245:8093/d/HbAMQcWik/performance-
metrics-for-multi-cloud-application?orgId=1" ]
}

```

•

2.2 Delivery and usage

2.2.1 Package information

Delivered package consists of the below folder structure.

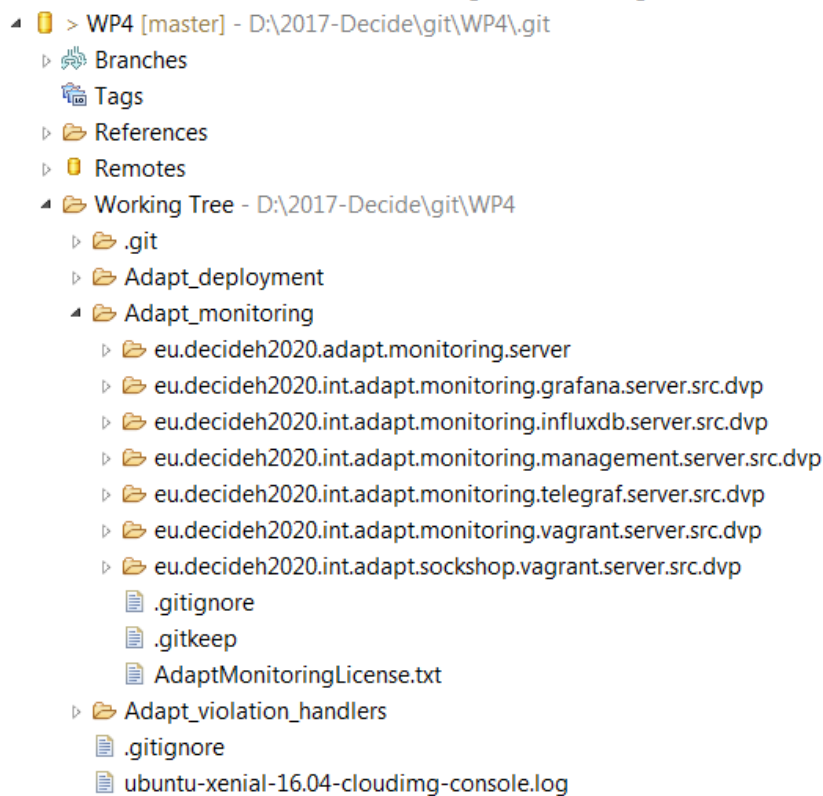


Figure 3. Source folder structure of ADAPT Monitoring component in M30.

ADAPT monitoring folder is composed of seven main sub-folders with the corresponding src folder, each one devoted to a component of ADAPT MM (see figure 2).

- **eu.decideh2020.adapt.monitoring.server:** Contains the src for the ADAPT MM Control Manager.
- **eu.decideh2020.int.adapt.monitoring.grafana.server.src.dvp:** Contains the src for the ADAPT MM GUI Manager and the corresponding Docker file to generate the container.
- **eu.decideh2020.int.adapt.monitoring.influxdb.server.src.dvp:** Contains the src for the ADAPT MM data Storage and Aggregation and the corresponding Docker file to generate the container.
- **eu.decideh2020.int.adapt.monitoring.management.server.src.dvp:** Contains the corresponding docker file to generate the container to deploy the ADAPT MM Control Manager.
- **eu.decideh2020.int.adapt.monitoring.telegraf.server.src.dvp:** Contains the src for the ADAPT MM Data Collection and the corresponding Docker file to generate the container.
- **eu.decideh2020.int.adapt.monitoring.vagrant.server.src.dvp:** Contains the src for the vagrant file, used to create the local virtualized environment (when needed) where ADAPT MM is deployed (more information about the *vagrant* tool) follows below.
- **eu.decideh2020.int.adapt.sockshop.vagrant.server.src.dvp:** Contains the src for the testing multi-cloud application (Shock Shop application) to be monitored and the corresponding Docker file to generate the container.

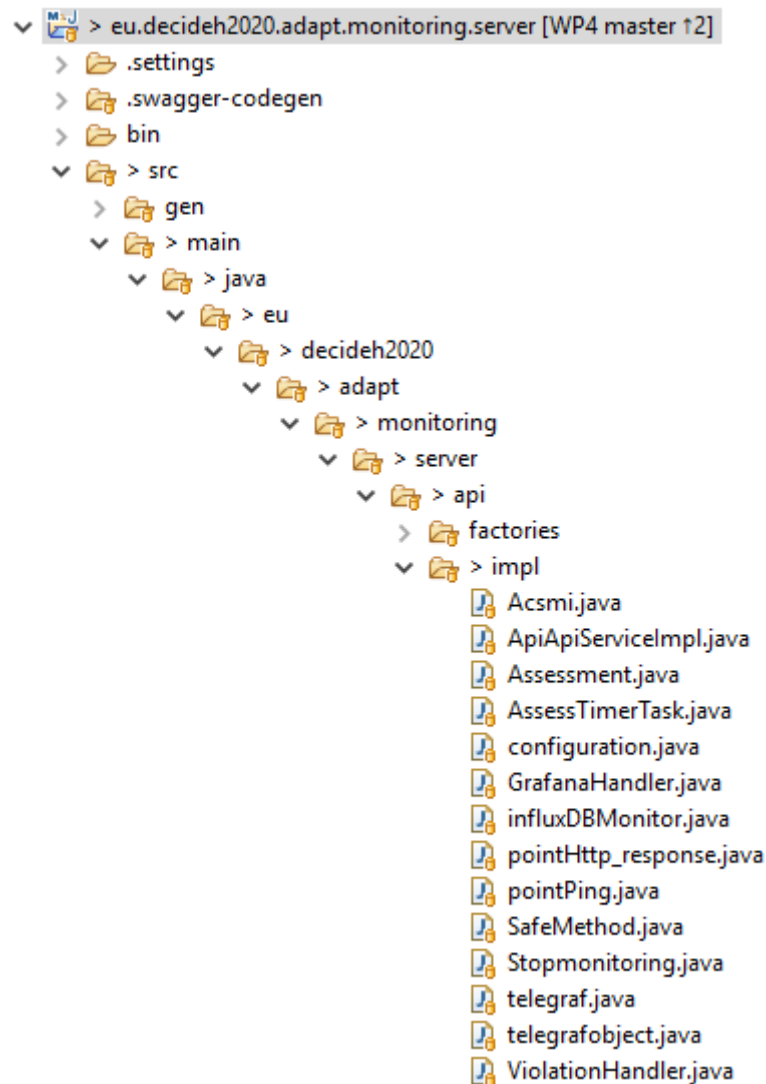


Figure 4. Source folder structure of ADAPT MM Control Manager sub-component

Inside the **eu.decideh2020.int.adapt.monitoring.server** the following relevant source files are included under the src folder (java sources with the .java extension):

- Acsmi: Java file to connect with the ACSmi monitoring and use its REST interface.
- ApiApiServiceImpl: Java file with the implementation of the iStartStopmonitoring interface.
- Assessment: Java file with all the actions to create and start the corresponding timer to assess the MCSLA of one application.
- AssessTimerTask: implements the java timer that performs the periodic assessment of the components of the multi-cloud application in real time.
- configuration: Java file with all the corresponding actions for the initial automatic configuration of the monitoring. This includes reading the Application Description and writing the relevant information on it, and launching other internal actions such as configuration of the Telegraf and Grafana, assessment of the MCSLA and others.
- GrafanaHandler: contains the needed actions to automatically configure the Grafana Dashboards based on the selected NFRs to be monitored.
- influxDBMonitor: Connects with the InfluxDB database, reading the information contained in it and writing new information when necessary.
- pointHttp Response: plain java object class to receive the HTTP_RESPONSE events information in the queries of the database.

- pointPing: plain java object class to receive the PING events information in the queries of the database.
- SafeMethod: plain java object class to represent the safeMetod object of the Application Description.
- Stopmonitoring: Java file that performs the needed actions when a stop monitoring request is received.
- telegraf: Java file that uses the implemented Telegraf parser library³ (see annex 1) to configure the telegraf.conf file based on the information of the application.
- telegrafobject: Java file that implements the object that stores the relevant information from the Application description to configure Telegraf.
- ViolationHandler: java file with the actions to connect with the ViolationHandler component and report a Violation through its API.

Under the resources folder, the following files are relevant:

- telegraf.conf: This file contains the configuration of input and output plugins to create the agents that get the metrics from the running application. This file is automatically configured by the ADAPT MM Control Manager component.

Inside the **eu.decideh2020.int.adapt.monitoring.grafana.server.src.dvp** the following relevant folders and packages are included (see figure 5):

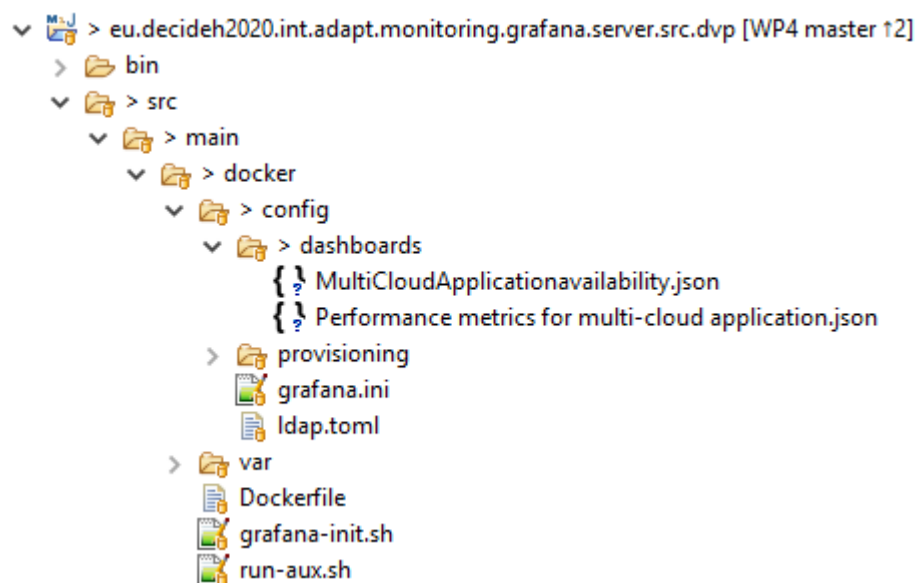


Figure 5. Source folder structure of ADAPT MM UI sub-component

- MultiCloudApplicationavailability.json: Json files with the specification of the graphical Dashboard for Availability metrics for the DECIDE MM GUI.
- Performance metrics for multi-cloud application.json: Json files with the specification of the graphical Dashboards for performance metrics for the DECIDE MM GUI.

³ This library has been implemented in the context of the DECIDE project to be able to configure the telegraf.conf file programmatically, according to the needs of DECIDE. The telegraf.conf file is written in TOML and up to DECIDE partners' knowledge there is not any open source parser that fulfils DECIDE requirements for the automatic configuration of the telegraf.conf file.

Inside the **eu.decideh2020.int.adapt.monitoring.influxdb.server.src.dvp** the following relevant folders and packages are included (figure 6):

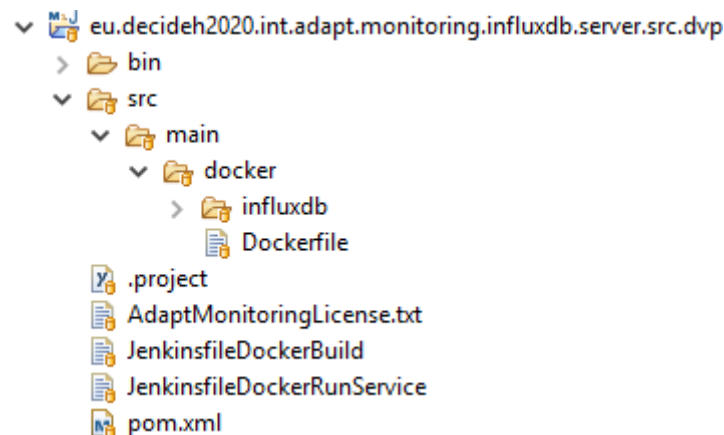


Figure 6. Source folder structure of ADAPT MM Data Storage and Aggregation

- **Dockerfile:** It is the file to generate the container of InfluxDB database.

Inside the **eu.decideh2020.int.adapt.monitoring.vagrant.server.src.dvp** the following relevant folders and packages are included under the src folder (figure 7):

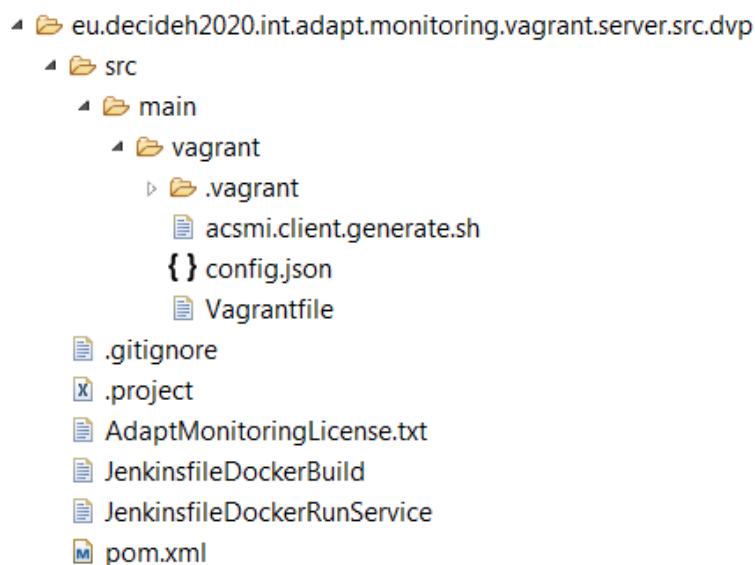


Figure 7. Source folder structure of Vagrant server.

- **Vagrantfile:** Configuration file of the vagrant tool. Vagrant is an open-source software product for building and maintaining portable virtual software development environments, that we use in DECIDE to create a local the virtual environment where ADAPT MM is deployed

Inside the **eu.decideh2020.int.adapt.shockshop.vagrant.server.src.dvp** the following relevant folders and packages are included (figure 8):

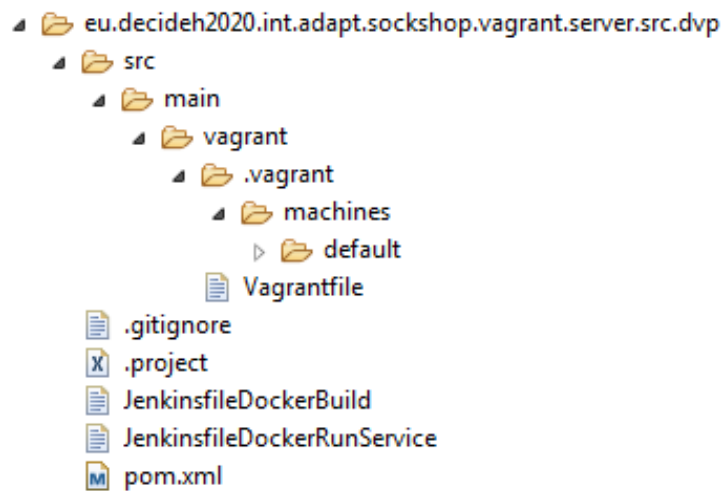


Figure 8. Source folder structure for the Shock Shop application.

Vagrantfile: Configuration file to facilitate the setting up of the Shock Shop application. It automatizes the creation of a virtual machine with the corresponding operating system and containers). This release includes the Sock Shop application as a testing application for unit tests only. The real application to be monitored will be deployed by the ADAPT Deployment Orchestrator component, which will call ADAPT Monitoring Manager to start monitoring

2.2.2 Installation instructions

The following lists the minimum requirements to get the component working:

- Linux machine (in Windows, a virtual machine will suffice) with the O.S (Ubuntu Xenial 16.04) and the Docker server.
- JRE, Java Runtime Environment.

The final prototype (M30) of ADAPT MM is composed by five main microservices: The Data collection, the Data storage and aggregation, the GUI Manager, the Control Manager and the Violations detection.

Regarding the deployment of ADAPT MM, it has been organized into three different Docker containers as follows:

- Three of these microservices (Control Manager, Data Collection, and Violation Detection) are packaged into a single Docker container and accessed through a REST API interface.
- The Data Storage and Aggregation component -formed by the *InfluxDB*, that is a time series database-, that is used by the rest of components.
- The GUI Manager component - *Grafana* visualization tool- which is configured in run time by the Control Manager, via the definition of panels and dashboards. It accesses the InfluxDB and represents graphical or numerically the metrics.

The list of containers used to deploy the ADAPT Monitoring Manager and its names are reflected in the following table:

Table 2. Containers deployed by the ADAPT Monitoring Manager prototype.

CONTAINER ID	IMAGE	NAMES
68be13b11c1d	decideh2020/adapt.monitoring.grafana	adapt.monitoring.grafana
f8af108bfac9	decideh2020/adapt.monitoring.management	adapt.monitoring.management
f279b216531f	decideh2020/adapt.influxdb	adapt.influxdb

The following steps need to be executed to get the prototype up and running:

1. Download the **source code** of the DECIDE components from the DECIDE public repository (DECIDE_components/ADAPT/Monitoring tag M30). The command line instruction is:

```
$ git clone https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components.git
```

2. Set the credentials and the path for the git repository in the **configuration file** (adaptmon.properties) that in Linux OS has to be in “/etc/decide/adaptmon.properties” folder (once ADAPT is integrated in the whole process of DECIDE, these credentials will be managed by VAULT). Other parameters to be configured in this file refer to the location of tools used by ADAPT MM and other components of DECIDE (InfluxDB, Grafana, ASCml). See an example adaptmon.properties configuration file in the Figure 9 below.

```
#-----
# Adapt Monitoring Config file
# Properties file location in Linux: "/etc/decide/adaptmon.properties" (F_PROPERTIES_LINUX)
#-----

# Appdescription in GIT (for local tests only // integration tests)
adaptMonitoring.appDescriptionGitPath=https://git.code.tecnalia.com/decide/adapt-do-demo.git
adaptMonitoring.descriptorname = AdaptMM/DECIDE.json
# GIT info: credentials
adaptMonitoring.GitUser=inaki.etxaniz@tecnalia.com
adaptMonitoring.GitToken=Gy_QiXgbFubhR8sxrUNe

#AcsmiMonitoring endpoint
adaptMonitoring.AcsmiEndPoint = http://localhost:14080/monitoringmanager/api/resourcemonitoring

# InfluxDB url (local/integration tests)
adaptMonitoring.InfluxdbURL = ["http://adapt.influxdb:8086"]

# Grafana server in Integration // Local Docker
adaptMonitoring.grafanaHost = http://adapt.monitoring.grafana:3000
```

Figure 9. ADAPT Monitoring Manager configuration file

3. The deployment of ADAPT Monitoring Manager is based on containers technology. To build the Docker containers of the different sub-components, first go to the base folder of the component source code. Among other utilities, there is a directory for every dockerized sub-component.

```
vagrant@ubuntu-bionic:/git/ADAPT/Monitoring$ ls -l
total 20
drwxr-xr-x 5 root 4096 Apr 3 16:26 eu.decideh2020.adapt.monitoring.server
drwxr-xr-x 3 root 4096 Apr 3 16:26 eu.decideh2020.int.adapt.monitoring.grafana.server.src.dvp
drwxr-xr-x 3 root 4096 Apr 3 16:26 eu.decideh2020.int.adapt.monitoring.influxdb.server.src.dvp
drwxr-xr-x 3 root 4096 Apr 3 16:26 eu.decideh2020.int.adapt.monitoring.management.server.src.dvp
drwxr-xr-x 3 root 4096 Apr 3 16:26 eu.decideh2020.int.adapt.monitoring.vagrant.server.src.dvp
drwxr-xr-x 3 root 4096 Apr 3 16:26 eu.decideh2020.int.adapt.sockshop.vagrant.server.src.dvp
drwxr-xr-x 3 root 4096 Apr 3 16:26 eu.decideh2020.int.violationhandler.client
```

Figure 10. Directories of the source code folder.

ADAPT MM DATA STORAGE AND AGGREGATION (INFLUX DB) CONTAINER:

- i. Switch to the folder where the Docker file is located.

```
$ cd
/git/ADAPT/Monitoring/eu.decideh2020.int.adapt.monitoring.influxDB.se
rver.src.dvp
```

```
$ cd src/main/docker
```

- ii. Build the docker image with the following arguments:

```
$ docker build -t decideh2020/adapt.influxdb:latest
```

- iii. Run the Docker image with the following arguments:

```
$ docker run --name adapt.influxdb -d -p 8086:8086 -e  
INFLUXDB_DB='decideh2020adapt' --network decide  
decideh2020/adapt.influxdb:latest
```

The arguments configure the following for the Influxdb container: the mapping of the ports as well as the name of the database inside the influx DB server.

ADAPT MM UI (GRAFANA) CONTAINER:

- i. Switch to the folder where the Docker file is located.

```
$ cd  
/git/ADAPT/Monitoring/eu.decideh2020.int.adapt.monitoring.grafana  
.server.src.dvp  
$ cd src/main/docker
```

- ii. Build the Docker image with the following arguments:

```
$ docker build -t decideh2020/adapt.monitoring.grafana:latest
```

- iii. Run the Docker image with the following arguments:

```
$ docker run --name adapt.monitoring.grafana -d -p 3000:3000 -e  
'GF_SERVER_ROOT_URL=http://grafana.esilab.org' -e  
'GF_SECURITY_ADMIN_PASSWORD=admin' --link  
adapt.influxdb:influxdb --network decide  
decideh2020/adapt.monitoring.grafana:latest
```

The arguments configure the following for the Grafana container: the mapping of the ports, the URL of the server, the corresponding admin password, and the related influxdb to be connected to the Grafana server with its container IP.

ADAPT MM CONTROL MANAGER & DATA COLLECTION (TELEGRAF) CONTAINER

In previous versions, the Telegraf tool was deployed in a separate container but, as the ADAPT MM needs to access and edit the configuration file of Telegraf, the decision to deploy both components in the same container was taken.

- i. Switch to the folder where the Docker file is located (ADAPT monitoring container).

```
$ cd  
/git/ADAPT/Monitoring/eu.decideh2020.int.adapt.monitoring.management.  
server.src.dvp  
$ cd src/main/docker
```

- ii. Build the Docker image with the following arguments:

```
$ docker build -t decideh2020/adapt.monitoring.management:latest
```

In the case of the ADAPT Monitoring container, in this release the GIT credentials are read from environment variables.

- iii. Run the Docker image with the following arguments:

```
$ docker run --name adapt.monitoring.management -d -p  
10080:8080 --link adapt.influxdb:influxdb --network decide  
decideh2020/adapt.monitoring.management:latest
```

The arguments configure the following for the ADAPT monitoring container: mapping of the ports and location of the Influxdb database.

Once the containers are created and running, the ADAPT monitoring is up and running.

2.2.3 User Manual

2.2.3.1 ADAPT MM API

ADAPT MM is started directly from the ADAPT DO component through the *iStartStopmonitoring* REST interface. In order to test it independently, the call to start the monitoring process needs to be done manually, by a call to the interface. This can be done using the Swagger editor⁴, loading the corresponding interface definition json file (included in the software package). To access the corresponding json to be loaded in the swagger editor, just open with the browser:

```
http://[IP where the container is  
deployed]:10080/monitoringmanager/swagger.json
```

In the swagger editor the “host” parameter needs to be changed to the IP and port where the host is deployed in each case (for local test for example, use “http://localhost:8080”).

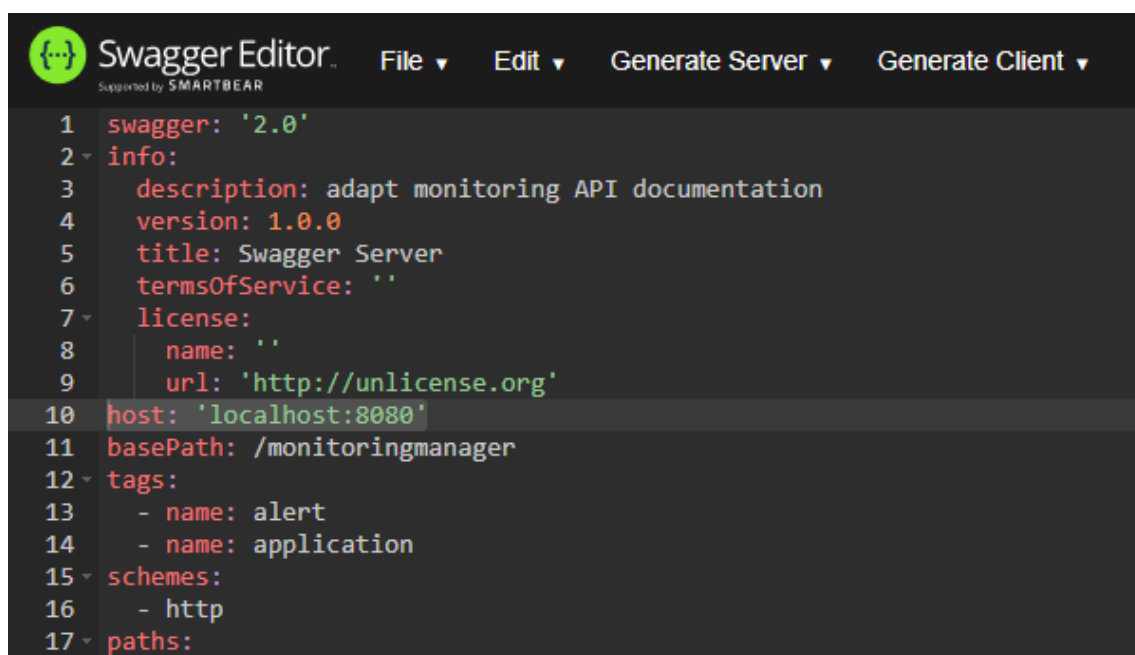
ADAPT monitoring provides the applications interface. The functions of the interface are shown at the right side of the Swagger editor. In the figure below, it can be seen that this interface defines five methods, being *createApplication* (POST) and *deleteApplication* (DELETE) used to start/stop the monitoring process.

⁴ <https://editor.swagger.io/>

application ▼

GET	/api/applications	getAllApplications
POST	/api/applications	createApplication
GET	/api/applications/{appdescuri}	getApplicationstatus
PUT	/api/applications/{appdescuri}	updateApplication
DELETE	/api/applications/{appdescuri}	deleteApplication

Figure 11. Methods of the ADAPT Monitoring REST API.



```

1  swagger: '2.0'
2  info:
3    description: adapt monitoring API documentation
4    version: 1.0.0
5    title: Swagger Server
6    termsOfService: ''
7    license:
8      name: ''
9      url: 'http://unlicense.org'
10 host: 'localhost:8080'
11 basePath: /monitoringmanager
12 tags:
13   - name: alert
14   - name: application
15 schemes:
16   - http
17 paths:

```

Figure 12. Swagger editor detail.

POST

/api/applications createApplication

Creates an application to be included in ADAPT monitoring

Parameters

Try it out

Name	Description
<div> <div>application * required</div> <div>(body)</div> </div> <div> <div>Example Value Model</div> <div> <pre>{ "monid": "string", "status": "string", "username": "string", "application": "string" }</pre> </div> <div> <div>Parameter content type</div> <div>application/json</div> </div> </div>	

Figure 13. POST method of iStartStop interface

Being the *Jetty* server of ADAPT MM started, this interface can be tested. To launch the monitoring, a POST call to *CreateApplication* function is to be performed. First, push the “Try it out” button. This allows to edit the body of the call. Once the parameters have been fulfilled, push the “Execute” button and the call is made. Once the response is received, the result is shown (see figure Figure 14). In the final version, the URI of the application description is not passed as parameter, nor the user/password of the git access; instead, they are obtained from Vault, after receiving in the call the username and application to access the Vault secrets)

The *monid* parameter is returned with the value of *applicationinstanceid* (this information is previously created by ADAPT DO and stored in the App Description). The *status* of the application has changed to “monitored”.

Curl

```
curl -X POST "http://localhost:8080/monitoringmanager/api/applications" -H "accept: */*" -H "Content-Type: application/json" -d '{"monid": "string", "appdescuri": "string", "status": "string", "user": "string", "password": "string"}'
```

Request URL

```
http://localhost:8080/monitoringmanager/api/applications
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "monid": "f6229960-201c-4f56-af96-b4659a95299d", "appdescuri": "string", "status": "monitored", "user": "string", "password": "string" }</pre> <p>Response headers</p> <pre>content-type: application/json</pre>

Responses

Code	Description
200	Application monitoring Created

Figure 14. Start monitoring response

ADAPT MM Control Manager configures the monitoring of the application based on the information read from the App Description file, which is the main information sharing mechanism in DECIDE (see D4.2 [9] [5] and D2.4 [6] for details). For testing purposes, the App Description accessed by this prototype is included in the software package (see installation instructions). The access rights and user management will be managed by the DevOps Framework in DECIDE.

2.2.3.2 ADAPT MM UI

Once the application is being monitored, the ADAPT monitoring GUI can be accessed to see the metrics of the application. During testing is useful to access it directly via Grafana, using the following address:

`http://[IP of the GRAFANA container]:10080`

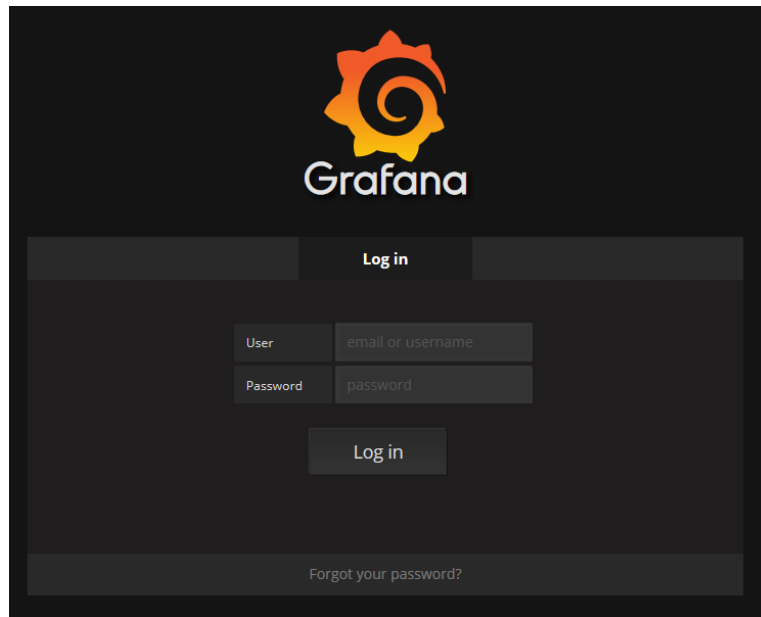


Figure 15. Grafana Log in interface.

A default user has been set up for the ADAPT MM, being the login parameters:

- User: decide
- Password: decide

Then, access to the *Dashboards* tab, and select one of the DECIDE predefined dashboards: Multi Cloud Application Availability or Performance Metrics for Multi-cloud Application:

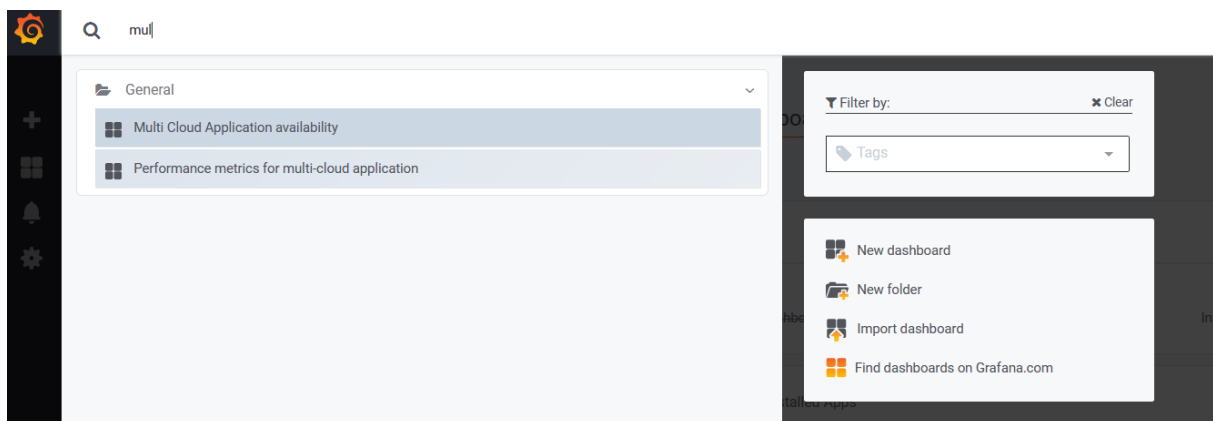


Figure 16. Dashboards tab

In the dashboard, the established metrics are monitored for the Sock Shop application.

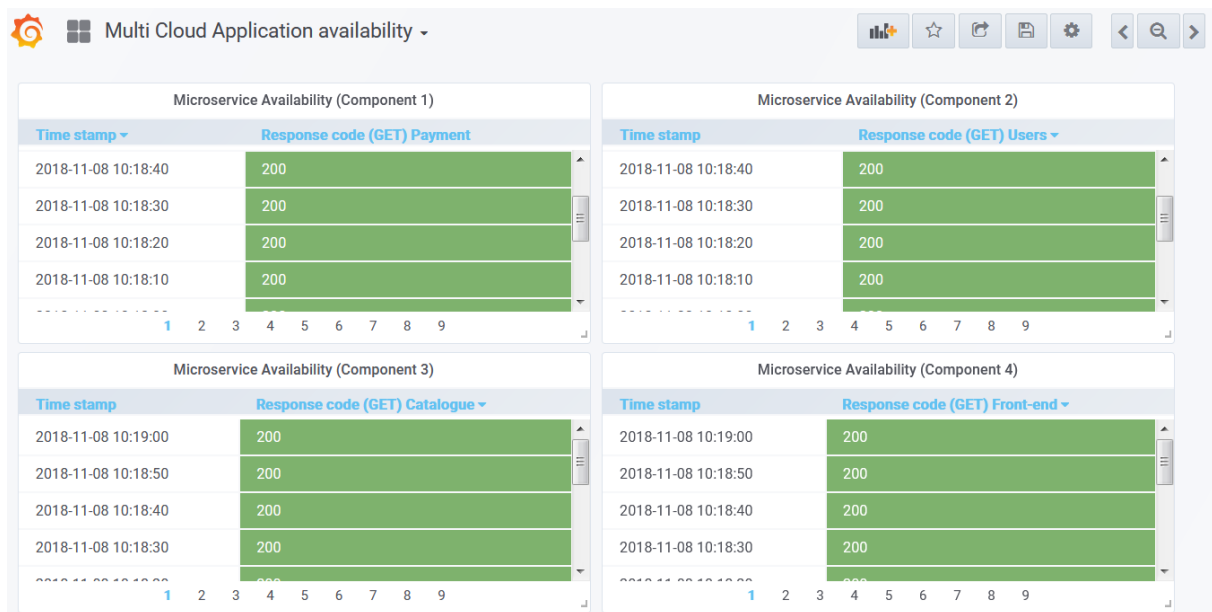
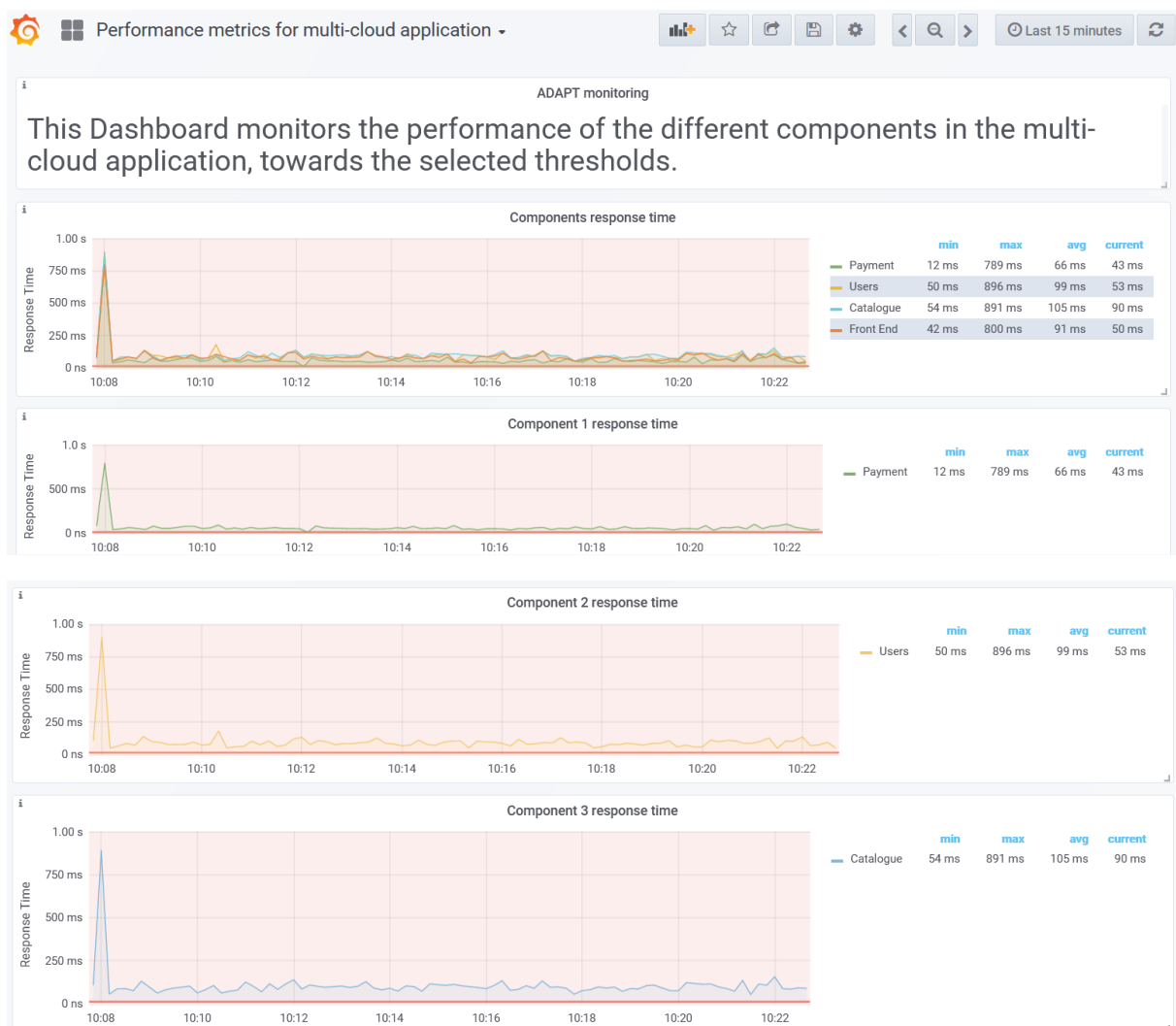


Figure 17. ADAPT Monitoring AVAILABILITY dashboard.



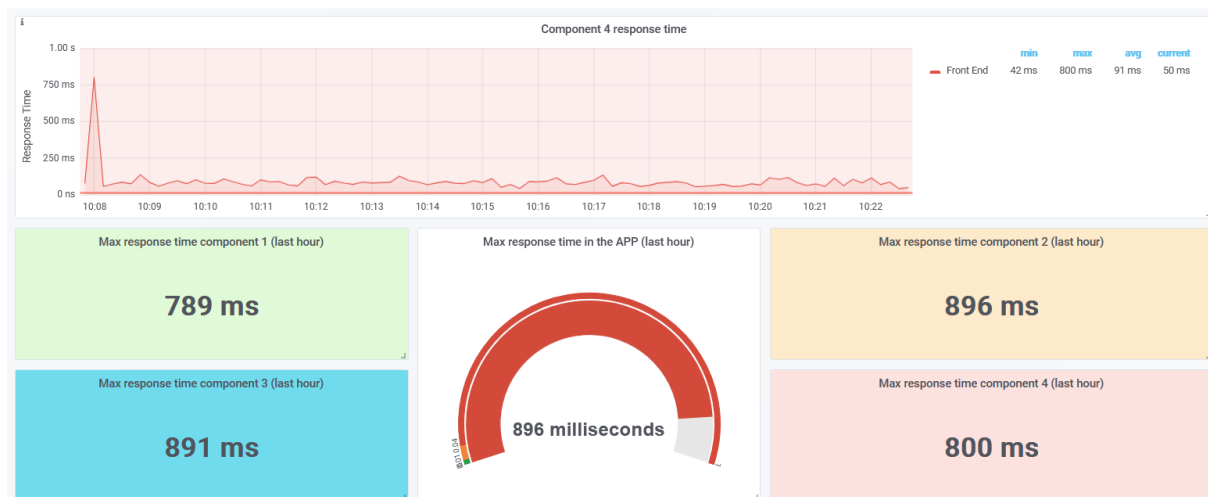


Figure 18 ADAPT monitoring PERFORMANCE dashboard

When using the integrated DECIDE environment, these dashboards are automatically integrated in the DevOps Framework (each dashboard, in the Grafana tool, has a specific URL where it can be accessed). When ADAPT MM is called, the links to these dashboards are included in the corresponding field of the application description (*monitoring* object, *url* field) which is read by the DevOps framework to load them in a tab of their own GUI.

2.2.4 Licensing information

This component is offered under MIT license.

2.2.5 Download

The source code is available in the EC portal for deliverables, included in the zip file for D4.9.

This Final release is available in the DECIDE open git repository, more precisely at the following address:

https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components/tree/master/ADAPT/Monitoring

and selecting the brand/tag M30.

3 Violations Handler

3.1 Implementation

3.1.1 Functional description

The DECIDE ADAPT Violations Handler is in charge of receiving a violation alert from ADAPT monitoring or ACSml monitoring and, depending on the application's technological risk, of notifying the operator and

- a) automatically trigger a new simulation through OPTIMUS (low technological risk) or
- b) notify the operator and give them a chance to modify settings before triggering a new simulation (in case of high technological risk).

The Violations Handler will also archive the violations to keep a history of alerts.

The main functionalities of the ADAPT Violations Handler have not changed much since the previous release of the Violations Handler, reported in D4.8 [1] on M24. These functionalities are:

F1. Receive violation alerts: when a violation occurs, the monitoring components (ADAPT monitoring and ACSml monitoring) will notify the Violations Handler, which will process the violation.

F2. Retrieve technological risk: when the Violations Handler receives a violation, it will retrieve the application's technological risk from the Application Description, in order to decide what action to carry out.

F3. Evaluate action to carry out: if the application's technological risk is low, the Violations Handler will automatically trigger a new simulation and the application will be redeployed according to the highest-ranked deployment schema. If it is high, the operator will have to confirm the deployment schema according to which the application will be redeployed. In both cases, the operator has to be notified of the violation.

F4: Notify operator: when a violation is received, the operator will receive an email containing relevant information about the alert.

F5: Send OPTIMUS a simulation order (low technological risk): if a violation has occurred, and the technological risk is low, the Violations Handler will automatically trigger a new simulation through OPTIMUS. The highest-ranked resulting deployment schema will be passed to ADAPT, which will automatically redeploy the application according to it.

F6: Redeploy the application (high technological risk): if a violation occurs but the technological risk is high, the application will be redeployed but the user has to review the deployment parameters. In this case, the VH will notify the operator that a violation took place and allow them to modify the required parameters before redeploying the application.

ADAPT Violations Handler has been implemented following an incremental approach adding features in the different releases of the tool (the first one was released on M12 and reported in D4.7 [10], and the second one reported in D4.8 on M24). In this third release, all expected functionalities are covered.

The following table details the relationship between the functional requirements -indicated in deliverable D4.3 [5] for year 3- and the implemented functionalities, with a description of the coverage for each functionality.

Requirements covered by the prototype:**Table 3.** Functionality covered by the M30 Violations Handler prototype.

Functionality	Req. ID	Coverage	Status
F1	WP4-MR1, WP4-REQ15	The prototype is able to receive an alert from ADAPT and ACSml monitoring and extract the relevant information from it. Besides, it stores the history of alerts to be visualized from the ADAPT UI.	Implemented. Testing integration.
F2	WP4-MR11	The prototype retrieves the technological risk from the Application Description.	Satisfied.
F3	WP4-MR8, WP4-MR9	Depending on the technological risk of the application, the prototype decides the action that will be carried out.	Satisfied.
F4	WP4-MR9, WP4-MR10	The prototype sends the user an email with violation information when a violation is detected.	Satisfied.
F5	WP4-MR1, WP4-MR8	The prototype sends an order to start a simulation to OPTIMUS and instructs ADAPT to automatically redeploy the application according to the highest-ranked schema.	Implemented. Testing integration.
F6	WP4-MR1, WP4-MR9	The prototype sends an order to start a simulation to OPTIMUS. Then, it presents the results of the simulation to the user, for them to choose their preferred deployment option.	Implemented. Testing integration.

Detailed functionality that this prototype offers:

This prototype corresponds to the third version of the Violations Handler. It comprises the functionalities already offered by the previous version of this component, which were:

- Receive and process alerts: the prototype receives a violation and is able to parse this violation to extract the relevant information.
- Retrieve technological risk: the prototype checks the technological risk of the component associated to the received alert. The technological risk is a variable that is stored in the Application Description.
- Store the violation: the prototype stores in a MySQL database the history of violations to be visualized from the ADAPT UI.
- Decide action to carry out: depending on the value of the technological risk, the prototype will decide what action will be carried out (automatic redeployment if the risk is low, and manual redeployment if it is high).
- Send an email: the prototype sends an email to the user when a violation is received, with relevant information about the violation.

- Trigger OPTIMUS: when a violation occurs and the technological risk is low, the prototype triggers OPTIMUS to obtain a new deployment schema that will be automatically selected as the new deployment configuration.

Besides, in this last M30 version it implements the high technology risk workflow:

- Restart the workflow for redeployment: when a violation is received, and the technological risk is high, an email will be sent for the user to be notified. Furthermore, it will trigger OPTIMUS and send it the violation, so that the reason of the violation can be taken into account when simulating new deployment configurations, to possibly avoid the problem. In this case, the user will have to select the new deployment schema.

3.1.1.1 Fitting into overall DECIDE Architecture

The Violations Handler is the component that is in charge of receiving and managing violations and orchestrating the redeployment workflows. It is part of the ADAPT Key Result, located inside the ADAPT package in Figure 1 (as it has been noted, that picture depicts only the interfaces used or implemented by ADAPT. The whole figure with all the interfaces between the different tools can be found in D2.5 [7]).

Within ADAPT, the Violations Handler interacts with ADAPT M Violation detection to receive MCSLA violations. It also interacts with other KRs: with OPTIMUS, to trigger a new simulation in case of a low technological risk, and with ACSml Monitoring to receive CSP violations. It also obtains the application's technological risk through the Application Description.

The following figure shows the general architecture of this component. For a more detailed description, check D4.3 [5]:

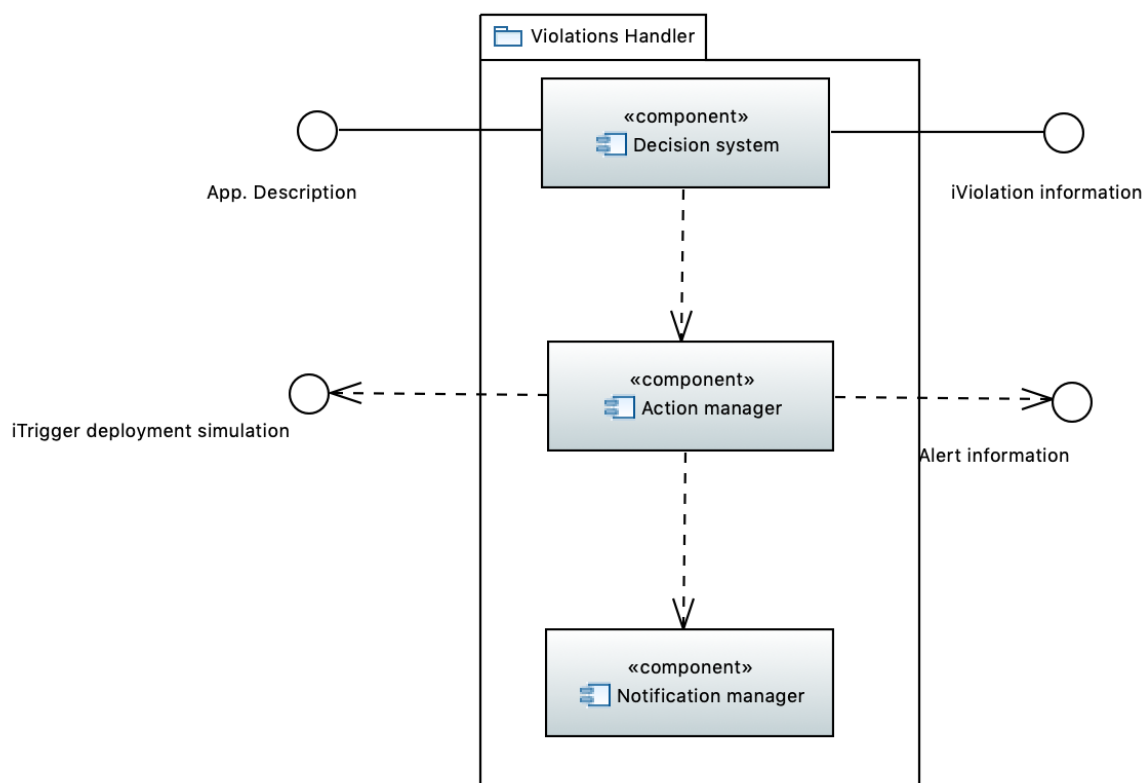


Figure 19. General architecture of the Violations Handler

3.1.2 Technical description

This section describes the technical details of ADAPT Violation handler.

3.1.2.1 Prototype architecture

From the proposed architecture that can be found in deliverable D4.1 [11] and that is shown in the above figure, the Violation Handler tool is composed by three main microservices, the **Decision system**, the **Action manager** and the **Notification manager**. All these components are packaged within the Violation Handler framework, which is deployed in a Docker container and accessed through a REST API interface. Finally, the Violation Handler also stores and retrieves information from a MySQL database.

3.1.2.2 Components description

- **Decision system:** It is responsible of receiving the violation and determine which decision should be performed. For making this decision successfully, it should check the technological risk value saved in the Application Description by the developer. In case the technological risk is high, the action will require user authorization. With all this information, it infers a result and communicates with the **Action manager** notifying which action should take place.
- **Action manager:** This component must be able to contact with the OPTIMUS microservice to trigger a new simulation process. Besides, it contacts the notification manager to send an email to the user.
- **Notification manager:** Retrieves the email from the application description and reports the violation details to the developer. Every time a new violation is received in the system, a report is sent to the user.

3.1.2.3 Technical specifications

The Violation handler component is an independent microservice mainly developed using Java programming language, implemented using Spring Boot framework. It offers a restful API interface modelled using Swagger editor that enables the communication this the submodules contained in the component. This interface provides a suitable interaction layer for reporting alerts and consulting the database that contains all the violations.

API interface definition

The definition of the API interface is presented in figure below. The controller is deployed under `/violationhandler` base URL.

violations		Show/Hide List Operations Expand Operations
POST	/violation	Report a new violation
GET	/violation/getAllViolations	Get all the violations that exist in the handler
GET	/violation/getViolationsByServiceId	Get all violations for given service ID
GET	/violation/getViolationsByUri	Get all violations for given application description URI
GET	/violation/{violationId}	Get violation from id

[BASE URL: /violationhandler , API VERSION: 1.0.0]

Figure 20. API interface definition for Violation Handler component.

For reporting a new violation in the system, both monitoring components, ACSml and ADAPT, must send a POST request with the violation object in the body. The rest of the HTTP calls enable to communicate with the violation MySQL database and to filter the violations historic list.

After reporting a new violation, the decision system extracts the relevant information from it, and infers the subsequent action. In parallel, it extracts the developer email address from the application description and contacts with the Notification manager.

The Violation model object defined in the component is composed by different attributes, as it is presented in the figure below:

```
Violation {  
  user (string): Id of the user,  
  app (string): Id of the application,  
  app_desc_uri (string, optional): Uri of the application description location,  
  id (integer, optional),  
  nfr (string): Affected NFR name,  
  service_id (string): Id of the service where violation occurs,  
  status (string): Status of the violation,  
  timestamp (integer, optional): Timestamp with the violation creation date,  
  type (string): Violation source type, SLA (0), CSP (1) or Microservice (2)  
}
```

Figure 21. Model of the Violation object handled by the API requests.

Decision system

This service is responsible of deciding the best approach for handling the violation. First of all, it pulls the application description stored in the git repository, using the *app_desc_uri* violation attribute. For this, it has integrated the *AppManager* library component, which provides an intuitive set of functions for interacting with the application description information. Afterwards, it reads the *highTechnologicalRisk* JSON field, which is a *boolean* type. In case it's tagged as true, then the redeployment of the service needs user confirmation, so the email notification will be the only output. On the other hand, if the technological risk is low, then the redeployment process must be triggered automatically without the developer consent. This redeployment process is delegated to the Actions manager. Finally, the violation is saved in the database, so it can be tracked in the future by any DECIDE tool.

Actions manager

The main action performed by this component is the orchestration of the redeployment workflow in case of a low technological risk. For that, the VH handler calls OPTIMUS' *iTriggerDeploymentSimulation* interface for generating a new deployment schema.

Violation Service - MySQL database

For managing the violation history for a certain application or microservice, every input received in the component is stored inside a MySQL database. All the violation objects can be retrieved using queries calling the API interface methods. The Violation service is responsible of contacting with the violation repository.

Notification manager

This module is responsible for sending the violation information to the developer email. The email address is retrieved from the *Application Description*. The email sender module has been implemented using the Spring Boot Mail module, contained within Spring Boot framework.

3.2 Delivery and usage

3.2.1 Package information

The Violation handler component has two modules, the server that handles the violation reception and the MySQL database where all the reports are stored.

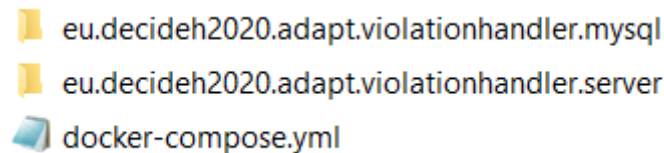


Figure 22. Source code structure of the Violation handler

In the directory structure we can distinguish two packages. The **src/gen** folder contains all the API interface definitions and model entities created in Swagger. The **src/main/java** directory contains the Violations Handler code and extends the generated Java classes for implementing the business logic.

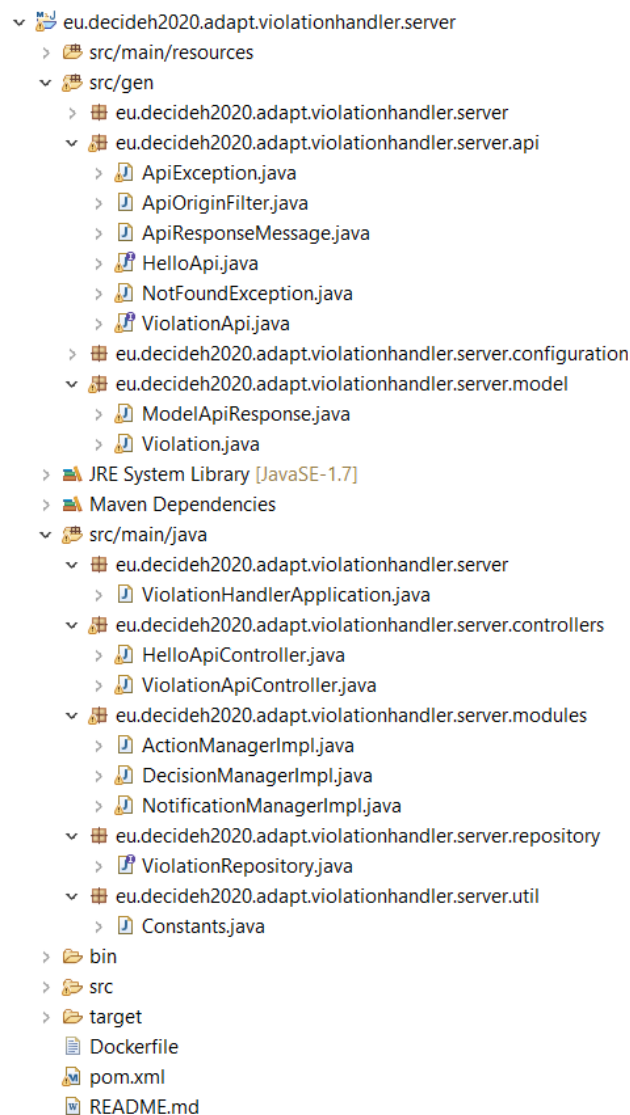


Figure 23. Structure of the *eu.decideh2020.adapt.violationhandler.server* module

The controller package contains the API implementation for the Violations Handler. This controller directly communicates with the rest of the modules. Furthermore, the modules package has all the subcomponents classes such as the *DecisionManager*, *ActionManager* and *NotificationManager*. To sum up, the *ViolationRepository* defines a CRUD (Create, Read, Update, Delete) interface for retrieving information from the MySQL database.

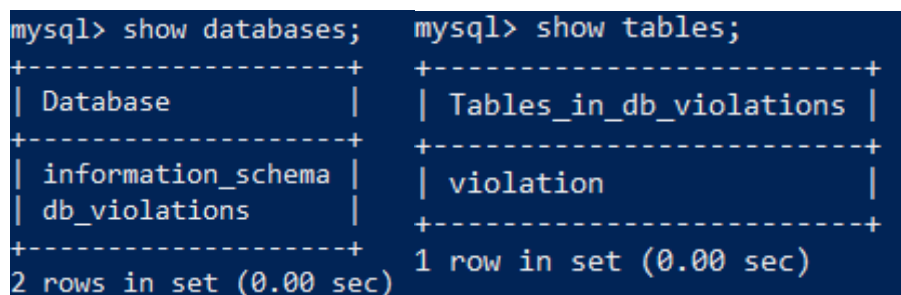
The main class is *ViolationHandlerApplication.java*, which initializes the Spring Boot application when it's invoked. The Violation entity model is used by all the API calls, and it's implemented in the *Violation.java* class, inside the generated files directory

The classes under the *util* package help to map the received JSON to functions to ease data read and handling. It also manages the constant values of the component.

Finally, the *application.properties* file contains the deployment configuration for several environments.

MySQL database

The database is composed by one table with all the violations reported to the system. The database is named *db_violations*, which contains the table called *violation* with all the rows. The database architecture is presented in Figure 24. Figure 21 details the violation attributes and the variable type for each of them.



```
mysql> show databases;      mysql> show tables;
+-----+      +-----+
| Database |      | Tables_in_db_violations |
+-----+      +-----+
| information_schema |      | violation |
| db_violations |      +-----+
+-----+      1 row in set (0.00 sec)
2 rows in set (0.00 sec)
```

Figure 24. Database *db_violations* and *violation* table in MySQL shell

The *ViolationRepository* class is responsible for querying the violations table and filter depending on the input parameter. In this version, there are three queries for accessing the data. The MySQL query for obtaining all the violations detected for a certain *service_id* is:

```
SELECT * FROM violation WHERE service_id = 'my_id'
```

The query for obtaining all the violations detected for a specific application description is:

```
SELECT * FROM violation WHERE app_desc_uri = 'my_uri'
```

The query for obtaining all the violations detected for a specific user and application is:

```
SELECT * FROM violation WHERE user = 'user_id' and app = 'app_id'
```

These queries are executed by the controller when the *getSimulationsByServiceId* and *getViolationsByUri* are requested. The API is described in Figure 28.

3.2.2 Installation instructions

The Violations Handler have been developed in Java and is supported by a MySQL database. To install it, it is first required to have a Docker environment configured with the following deployment steps:

1. First, it is necessary to compile the Violations Handler repository cloned locally. For this, you should execute:
 - `git clone https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components.git`
2. For building the image, you must navigate to the root folder and build it as follows:
 - `docker-compose build -f docker-compose-public.yml --no-cache`
3. Additionally, there is a docker-compose file which automatically deploys the Violations Handler server and the MySQL database. It also rebuilds the server if necessary:
 - `docker-compose up -d`

```
version: "3"
services:
  mysql_vh:
    image: mysql:5.5.59
    container_name: mysql_vh
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_USER: admin
      MYSQL_PASSWORD: decide
      MYSQL_DATABASE: db_violations
    ports:
      - 3306:3306
  violationhandler:
    build: ./
    container_name: violationhandler
    depends_on:
      - mysql_vh
    ports:
      - 5000:5000
networks:
  decide:
    external: true
```

Figure 25 Docker compose configuration file.

IMAGE	PORTS	NAMES
violationhandler_violationhandler	0.0.0.0:5000->5000/tcp	violationhandler
mysql:5.5.59	0.0.0.0:3306->3306/tcp	mysql_vh

Figure 26. Violations Handler's docker containers running

3.2.3 User Manual

The process starts by sending an HTTP POST request to ***violationhandler/violation***, with the following message body format describing the violation object. The API also allows to explore the violations stored, being able to query the results sending the application URI or the service id. A request body example is presented in Figure 27.

```
{
  "user": "experis",
  "app": "ShockShop",
  "id": 0,
  "nfr": "availability",
  "service_id": "a1b2c3d4e5f6",
  "status": "detected",
  "timestamp": 11549684646546,
  "type": "1"
}
```

Figure 27. HTTP POST request body example containing the violation information

The Violations Handler has integrated the Swagger UI that enables to explore and consume the API interface directly from the browser. It can be accessed under <http://localhost:5000/violationhandler>.

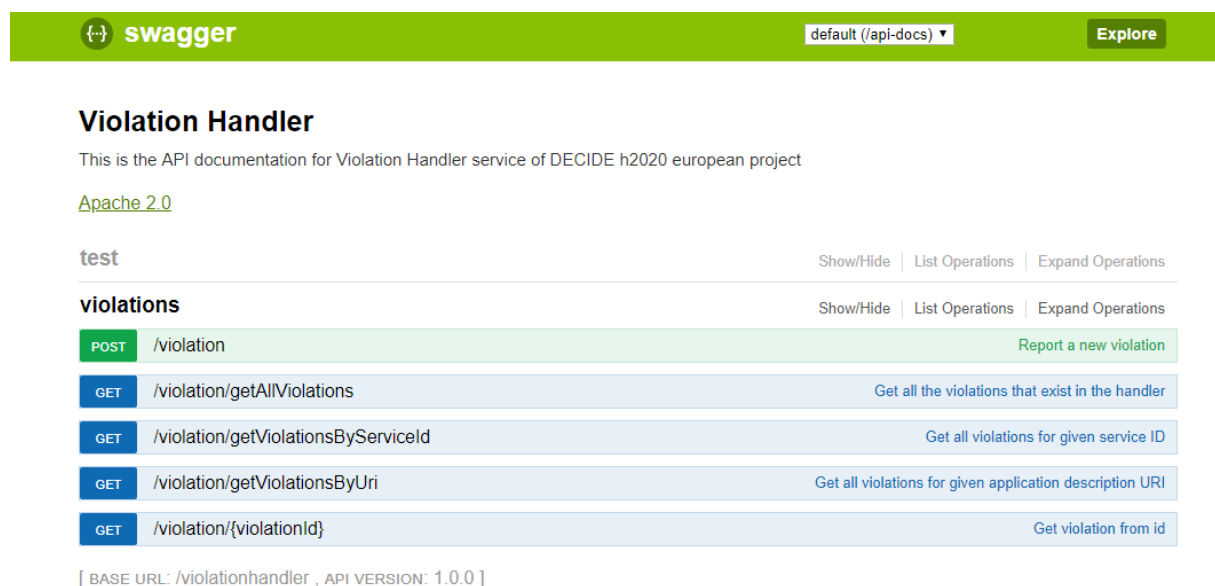


Figure 28. Swagger UI integrated within the Violation Handler.

To sum up, the component has been deployed in the DECIDE integration environment, and the microservice is located in <http://85.91.40.245:8095/violationhandler>

3.2.4 Licensing information

This component is offered under the MIT license.

3.2.5 Download

The source code is available in the EC portal for deliverables, included in the zip file for D4.9.

This release is available in the DECIDE open git repository, more precisely at the following address:

https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components.git

DECIDE_Components/ADAPT/ViolationsHandler (tag M30).

4 Conclusions

In this document, the Final multi-cloud application monitoring is presented. The functional and the technical details of the components comprising this version: **ADAPT Monitoring Manager** and **Violations Handler**, are described. Their installation processes and user manuals are detailed as well. The development is almost finished. Still, some final adjustment (tests, errors fixing) and integrations (mainly the Vault security tool) are pending, which will be undertaken in the few months period that remains until the end of the project.

Both components released provide new functionalities with respect to the previous version, as has been detailed in previous chapters.

ADAPT Monitoring Manager incorporates in this release the following new features:

- The implementation of the stopping of the monitoring of an application, as well as the functions to get the status, update an application and get all applications.
- The Application Description has been extended to re-define the “safe methods” -that are defined by the user to monitor the microservices that compose the application-.
- The Application Description is scanned to obtain both the safe methods and the deployment information, which allows to monitor the microservices whatever is the application topology after deployment.
- Automatic configuration of the input and output plugins of the *telegraf* tool, that collects the metrics from the services of the application, and stores them in the corresponding data base.
- Generation of the aggregated value (performance, availability) per component -using the *mcsla-core* library-, and for the whole application.
- The dashboards are generated automatically depending on the information found on the Application Description and configured inside the visualization tool (Grafana). Besides, the locations of the dashboards are introduced in the Application description to let the DevOps Framework to include them in its GUI.
- The data stored in the time series database -*influxdb*- is queried, aggregated and compared with the SLO established in the MCSLA at application level. This is done periodically for every application deployed.
- Integration with ACSmI Monitoring -calls to start resources monitoring- and with Violations Handler -call if a violation of the SLO is found-.
- **ADAPT Violation Handler** includes in this final release a new feature: Management of the restart of the workflow for redeployment. When a technologically high-risk violation is received, it will trigger OPTIMUS and inform it about the violation, so that the reason of the violation can be taken into account when simulating new deployment configurations. Furthermore, the user is notified through an email sent to him.

This final release of the multi-cloud application monitoring, with the new functionalities added, and the improvements included, assure that the current component satisfies the requirements defined in D4.3 [5].

During the next months until the end of the project, the pending work is the integration of Vault, as a tool to manage the identity and access in the complete workflow of DECIDE. This will imply changing the security parameters received in the interface calls, and the implementation of the access to Vault to get the secrets stored there. Also, the monitoring of the Cost of the deployed application in the dashboards is foreseen, which will be possible for the ADAPT MM by calling an API defined by ACSmI to get the cost of deployment in the different cloud resources available.

Besides this, some possible improvements that have been identified for future developments are the inclusion of new metrics of performance (throughput), the extension mechanism of the application

NFRs to be integrated and tested. and the design of new more complete panels and graphics in the dashboards.

5 References

- [1] DECIDE Consortium, D4.8 Intermediate multi-cloud application monitoring, 2018.
- [2] DECIDE consortium, “DECIDE DoA (Description of action),” 2016.
- [3] C. McLuckie, “Perspective on multi-cloud (part 3 of 3) — Availability and multi-cloud,” [Online]. Available: <https://blog.heptio.com/perspective-on-multi-cloud-part-3-of-3-availability-and-multi-cloud-5018762d2702>. [Accessed 22 11 2018].
- [4] ARTIST Consortium, “D7.1 Definition and extension of performance,” 2014.
- [5] DECIDE Consortium, “D4.3-Final DECIDE ADAPT Architecture,” 2018.
- [6] DECIDE consortium, “D2.4 Detailed architecture v1,” 2017.
- [7] DECIDE consortium, “D2.5-DECIDE detailed architecture v2,” 2018.
- [8] DECIDE consortium, “D3.14-Intermediate multi-cloud native application composite CSLA definition,” 2018.
- [9] DECIDE Consortium, “D4.2-Intermediate DECIDE ADAPT Architecture,” 2018.
- [10] DECIDE Consortium, “D4.7-Initial multi-cloud application monitoring,” 2017.
- [11] DECIDE Consortium, “D4.1 Initial DECIDE ADAPT Architecture,” 2017.
- [12] Weaveworks, “Sock Shop: A microservices demo application,” [Online]. Available: <https://microservices-demo.github.io/>. [Accessed 25 10 2017].
- [13] WeaveWorks, “Sock Shop - Docker compose,” [Online]. Available: <https://github.com/microservices-demo/microservices-demo/tree/master/deploy/docker-compose>. [Accessed 22 11 2018].
- [14] “Sock Shop, A Microservices Demo Application,” [Online]. Available: <https://microservices-demo.github.io/>.
- [15] DECIDE, “Deliverable D2.8 Final DECIDE DevOps Framework Integration”.

6 Annex 1: Telegraf parser library

This library has been implemented in the context of the DECIDE project to be able to configure the telegraf.conf file programmatically, according to the needs of DECIDE. The telegraf.conf file is written in TOML⁵ format and up to DECIDE partners' knowledge there is not any open source parser that fulfils DECIDE requirements for the automatic configuration of the telegraf.conf file.

6.1 Pre-requirements

The only pre-requirement is to have Telegraf_ConfigParser library (JAR file) installed in the Maven repository.

6.2 Downloading the Telegraf_ConfigParser library

Download the library from the DECIDE public repository (https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components.git):

The screenshot shows the DECIDE_Components repository on GitHub. It displays a commit titled "Added WP5 tools and mcsla-core for ADAPT MO" by Escalante Martinez, Marisa, made 12 minutes ago. Below the commit, a table lists the files and folders in the repository:

Name	Last commit	Last update
ACSml	Added WP5 tools and mcsla-core for ADAPT MO	12 minutes ago
ADAPT	Added WP5 tools and mcsla-core for ADAPT MO	12 minutes ago
ARCHITECT	includes components	1 year ago
AppController	APPController for ADAPT VH	58 minutes ago
DevOpsFramework	Adds updated DevOpsFramework	9 months ago
MCSLA	Added WP5 tools and mcsla-core for ADAPT MO	12 minutes ago
OPTIMUS	APPController for ADAPT, ACSml and OPTImus	1 hour ago
README.md	Update README.md	9 months ago

Figure 29. File structure for the Telegraf_ConfigParser folder.

The JAR file is accessible in “ACSml/Telegraf_ConfigParser/dist”:

Name	Last commit	Last update
..		
Telegraf_ConfigParser.jar	Added WP5 tools and mcsla-core for ADAPT MO	15 minutes ago

Figure 30. Telegraf_ConfigParser library

Download the JAR file and store temporarily in a system folder.

6.3 Installing the library in the Maven repository

Execute the following command in cmd:

⁵ <https://en.wikipedia.org/wiki/TOML>


```
mvn install:install-file -Dfile=C:\D\Telegraf_ConfigParser.jar -
DgroupId=eu.decideh2020 -DartifactId=
eu.decideh2020.telegraf.configparser -Dversion=0.0.1-SNAPSHOT -
Dpackaging=jar -DgeneratePom=true
```

Note: this command corresponds to a single line. Update the pom.xml in the corresponding project:

```
<dependency>
  <groupId>eu.decideh2020</groupId>
  <artifactId>eu.decideh2020.telegraf.configparser</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
```

6.4 Components and plugins

6.5 Generic component structure

Telegraf_ConfigParser collects the basic structure of the Telegraf configuration file (“telegraf.conf”) with the following components structure:

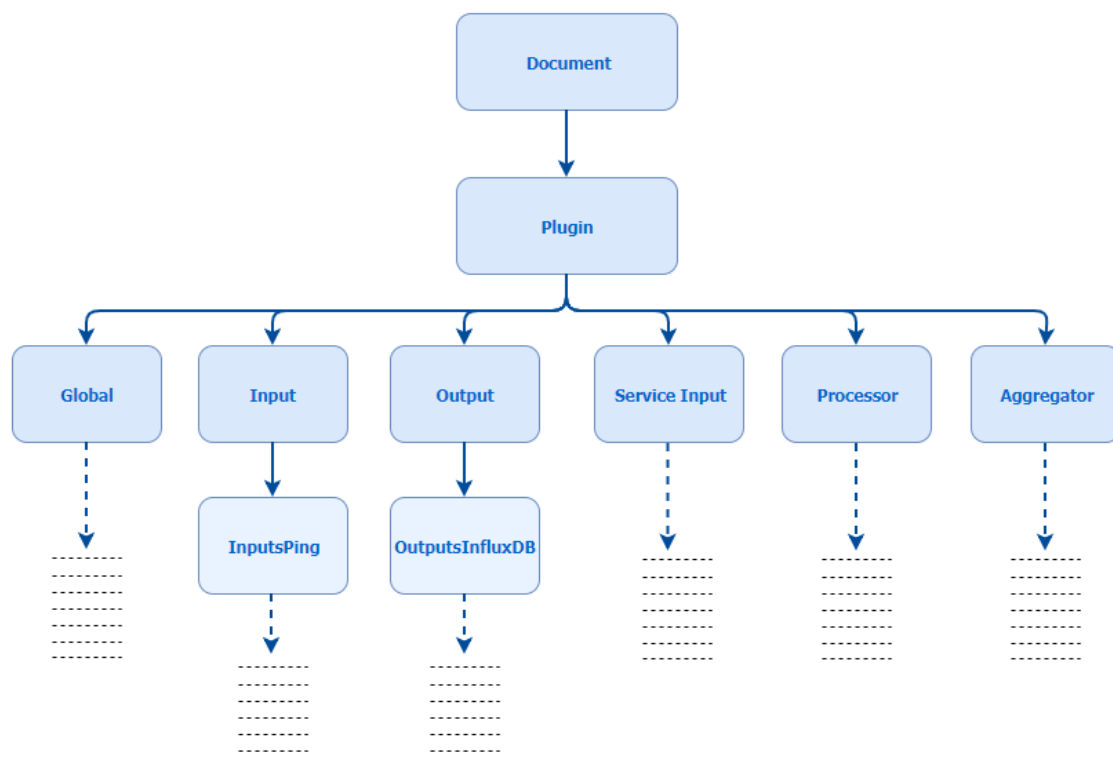


Figure 31. Telegraf.conf components structure.

The Telegraf configuration file is collected into a “Document” object, with a list of “Plugin” objects. This “Plugin” object contains the following information:

- Global.
- Input.
- Output.

- Service Input.
- Processor.
- Aggregator.

6.6 General structure of a Telegraf Plugin.

The plugins inside the Telegraf configuration file are described using TOML⁶ format. This library is focused into six of these plugins which are used by ADAPT monitoring and ACSml monitoring.

- **HTTP Response:** Input plugin type. Corresponds to the “InputsHTTPResponse” component/class.
- **Ping:** Input plugin type. Corresponds to the “InputsPing” component/class.
- **InfluxDB:** Output plugin type. Corresponds to the “OutputsInfluxDB” component/class.
- **Mem:** Input plugin type. Corresponds to the “InputsMem” component/class.
- **Disk:** Input plugin type. Corresponds to the “InputsDisk” component/class.
- **CPU:** Input plugin type. Corresponds to the “InputsCPU” component/class.

6.7 “HTTP Response” plugin structure

Telegraf.conf file contains the following information for this plugin:

```
# HTTP/HTTPS request given an address a method and a timeout
[[inputs.http_response]]
  ## Server address (default http://localhost)
  address = "http://localhost"
  ## Set http_proxy (telegraf uses the system wide proxy settings if
  it's is not set)
  http_proxy = "http://localhost:8888"
  ## Set response_timeout (default 5 seconds)
  response_timeout = "5s"
  ## HTTP Request Method
  method = "GET"
  ## Whether to follow redirects from the server (defaults to false)
  follow_redirects = false
  ## Optional HTTP Request Body
  body = '''
  {'fake':'data'}
  '''
  ## Optional substring or regex match in body of the response
  response_string_match = "\"service_status\": \"up\""
  response_string_match = "ok"
  response_string_match = "\".*_status\".\"?:.?\"up\""
  ## Optional TLS Config
  tls_ca = "/etc/telegraf/ca.pem"
  tls_cert = "/etc/telegraf/cert.pem"
  tls_key = "/etc/telegraf/key.pem"
  ## Use TLS but skip chain & host verification
  insecure_skip_verify = false
  ## HTTP Request Headers (all values must be strings)
  [inputs.http_response.headers]
    Host = "github.com"
```

⁶ <https://en.wikipedia.org/wiki/TOML>

6.8 “Ping” plugin structure

Telegraf.conf file contains the following information for this plugin:

```
# Ping given url(s) and return statistics
[[inputs.ping]]
  ## List of urls to ping
  urls = ["example.org"]
  ## Number of pings to send per collection (ping -c <COUNT>)
  count = 1
  ## Interval, in s, at which to ping. 0 == default (ping -i
  <PING_INTERVAL>)
  ## Not available in Windows.
  ping_interval = 1.0
  ## Per-ping timeout, in s. 0 == no timeout (ping -W <TIMEOUT>)
  timeout = 1.0
  ## Total-ping deadline, in s. 0 == no deadline (ping -w <DEADLINE>)
  deadline = 10
  ## Interface or source address to send ping from (ping -I
  <INTERFACE/SRC_ADDR>)
  ## on Darwin and FreeBSD only source address possible: (ping -S
  <SRC_ADDR>)
  interface = ""
  ## Specify the ping executable binary, default is "ping"
  binary = "ping"
  ## Arguments for ping command
  ## when arguments is not empty, other options (ping_interval,
  timeout, etc) will be ignored
  arguments = ["-c", "3"]
```

6.9 “InfluxDB” plugin structure

Telegraf.conf file contains the following information for this plugin:

```
# Configuration for sending metrics to InfluxDB
[[outputs.influxdb]]
  ## The full HTTP or UDP URL for your InfluxDB instance.
  ## Multiple URLs can be specified for a single cluster, only ONE of
  the
  ## urls will be written to each interval.
  urls = ["http://acsmi.influxdb:8086"] # required
  ## The target database for metrics; will be created as needed.
  ## For UDP url endpoint database needs to be configured on server
  side.
  database = "decideh2020acsmi" # required
  ## The value of this tag will be used to determine the database. If
  this
  ## tag is not set the 'database' option is used as the default.
  database_tag = ""
  ## If true, no CREATE DATABASE queries will be sent. Set to true
  when using
  ## Telegraf with a user without permissions to create databases or
  when the
  ## database already exists.
  skip_database_creation = false
  ## Name of existing retention policy to write to. Empty string
  writes to
  ## the default retention policy. Only takes effect when using HTTP.
  retention_policy = ""
```

```

## Write consistency (clusters only), can be: "any", "one", "quorum",
"all".
## Only takes effect when using HTTP.
write_consistency = "any"
## Timeout for HTTP messages.
timeout = "5s"
## HTTP Basic Auth
username = "telegraf"
password = "metricsmetricsmetricsmetrics"
## HTTP User-Agent
user_agent = "telegraf"
## UDP payload size is the maximum packet size to send.
udp_payload = "512B"
## Optional TLS Config for use on HTTP connections.
tls_ca = "/etc/telegraf/ca.pem"
tls_cert = "/etc/telegraf/cert.pem"
tls_key = "/etc/telegraf/key.pem"
## Use TLS but skip chain & host verification
insecure_skip_verify = false
## HTTP Proxy override, if unset values the standard proxy
environment
## variables are consulted to determine which proxy, if any, should
be used.
http_proxy = "http://corporate.proxy:3128"
## Additional HTTP headers
http_headers = {"X-Special-Header" = "Special-Value"}
## HTTP Content-Encoding for write request body, can be set to "gzip"
to
## compress body or "identity" to apply no encoding.
content_encoding = "identity"
## When true, Telegraf will output unsigned integers as unsigned
values,
## i.e.: "42u". You will need a version of InfluxDB supporting
unsigned
## integer values. Enabling this option will result in field type
errors if
## existing data has been written.
influx_uint_support = false

```

6.10 “Mem” plugin structure

Telegraf.conf file contains the following information for this plugin:

```

# Read metrics about memory usage
[[inputs.mem]]
# no configuration

```

6.11 “Disk” plugin structure

Telegraf.conf file contains the following information for this plugin:

```

# Read metrics about disk usage by mount point
[[inputs.disk]]
## By default stats will be gathered for all mount points.
## Set mount_points will restrict the stats to only the specified
mount points.
mount_points = ["/"]
## Ignore mount points by filesystem type.

```

```
ignore_fs = ["tmpfs", "devtmpfs", "devfs", "overlay", "aufs", "squashfs"]
```

6.12 “CPU” plugin structure

Telegraf.conf file contains the following information for this plugin:

```
# Read metrics about cpu usage
[[inputs.cpu]]
  ## Whether to report per-cpu stats or not
  percpu = true
  ## Whether to report total system cpu stats or not
  totalcpu = true
  ## If true, collect raw CPU time metrics.
  collect_cpu_time = false
  ## If true, compute and report the sum of all non-idle CPU states.
  report_active = false
```

6.13 Operations on the Telegraf.conf file

Telegraf_ConfigParser implements a package to manage the different operations which can be performed on the Telegraf.conf file:

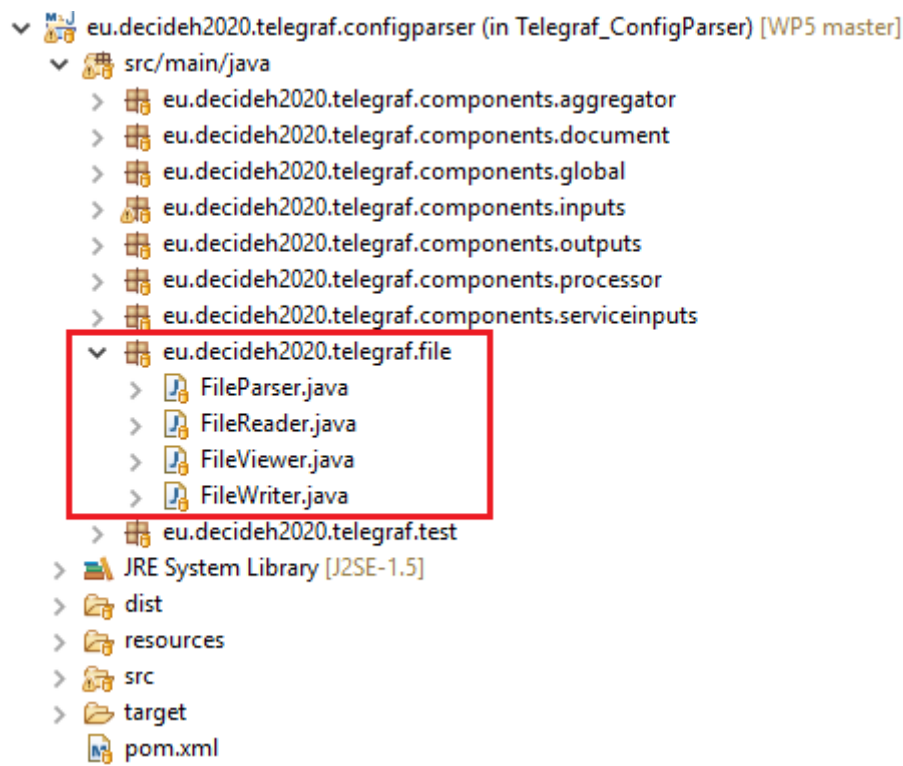


Figure 32. Java classes which implement the different operations to be performed in a Telegraf.conf file

Following these operations are described:

- FileReader: Manages the reading operation in the configuration file.
- FileWriter: Manages the writing operations in the configuration file: Conversion of a plugin object to the file format, conversion of a document object into the file format, writing the file located in an absolute path, writing the file located in an absolute path or relative path (to be selected).

- **FileParser:** This class allows to operate in the “Documents” object: Commenting/uncommenting one/all the plugin properties, Commenting/uncommenting one/all the plugins, modify a property with one/several values, duplicate a plugin, delete a plugin, count the number of plugins in the file with the same name.
- **FileViewer:** This class allows to perform the operations to view in the console the configuration file loaded: visualize the list inside the configuration file, visualize the document object.

6.14 Usage example

6.14.1 Capturing exceptions

The first step is to define a TRY - CATCH block to contain the code that is used:

```
try {
    . . . . .
} catch (IOException ioe) {
    ioe.printStackTrace();
} catch (ClassNotFoundException cnfe) {
    cnfe.printStackTrace();
} catch (IllegalAccessException iae) {
    iae.printStackTrace();
} catch (InstantiationException ie) {
    ie.printStackTrace();
} catch (CloneNotSupportedException cnse) {
    cnse.printStackTrace();
}
```

Exceptions can be captured one by one, or all at once by capturing an Exception object:

```
try {
    . . . . .
} catch (Exception) {
    e.printStackTrace();
}
```

If the library is to be used within another TRY - CATCH block that already existed, this step can be skipped.

6.14.2 Loading the configuration file

The next step is to declare a "FileParser" object and load the configuration file into a "Document" object. This is done in a single line:

```
try {
    FileParser parser = new FileParser("etc/telegraf/telegraf.conf");
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

```

} catch (ClassNotFoundException cnfe) {
    cnfe.printStackTrace();
} catch (IllegalAccessException iae) {
    iae.printStackTrace();
} catch (InstantiationException ie) {
    ie.printStackTrace();
} catch (CloneNotSupportedException cnse) {
    cnse.printStackTrace();
}
}

```

In this example, it is loaded from an absolute path.

The "FileParser" class has a "Document" object as an attribute. The constructor method will be responsible for using the "FileReader" class to load the contents of the configuration file in the "Document" object.

From this point, the operations described can begin to be used. For example:

- 1) Make two duplicates of the "inputs.ping" plugin.
- 2) Modify the "count" property of the original "inputs.ping" plugin (index = 1).
- 3) Modify the property "timeout" of the first plugin "inputs.ping" duplicated (index = 2).
- 4) Modify the property "urls" of the second plugin "inputs.ping" duplicated (index = 3) with a list of values.
- 5) Remove the original "inputs.ping" plugin (index = 1).
- 6) Remove the second duplicate "inputs.ping" plugin (index = 3). As in step 5) a duplicate has been eliminated, this second duplicate happens to have index = 2.
- 7) Show the result in console.

```

try {
    FileParser parser = new FileParser("etc/telegraf/telegraf.conf");
    // Step 1)
    parser duplicatePlugin("inputs.ping");
    parser duplicatePlugin("inputs.ping");
    // Step 2)
    parser.changePropertyValue("inputs.ping", "count", "3");
    // Step 3)
    parser.changePropertyValue("inputs.ping", 2, "timeout", "1.5");
    // Step 4)
    final String[] values = {"www.google.com",
        "www.google.es", "www.google.ie"};
    parser.changePropertyValues("inputs.ping", 3,
        "urls", Arrays.asList(values));
    // Step 5)
    parser.deletePlugin("inputs.ping");
    // Step 6)
    parser.deletePlugin("inputs.ping", 2);
    // Step 7)
}

```

```
    FileViewer.viewDocumentConsole(parser.getDocument());  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

The last step is to dump the modified "Document" object in the configuration file:

```
try {  
    FileParser parser = new FileParser("etc/telegraf/telegraf.conf");  
    // Step 1)  
    parser.duplicatePlugin("inputs.ping");  
    parser.duplicatePlugin("inputs.ping");  
    // Step 2)  
    parser.changePropertyValue("inputs.ping", "count", "3");  
    // Step 3)  
    parser.changePropertyValue("inputs.ping", 2, "timeout", "1.5");  
    // Step 4)  
    final String[] values = {"www.google.com",  
                             "www.google.es", "www.google.ie"};  
    parser.changePropertyValues("inputs.ping", 3,  
                                "urls", Arrays.asList(values));  
    // Step 5)  
    parser.deletePlugin("inputs.ping");  
    // Step 6)  
    parser.deletePlugin("inputs.ping", 2);  
    // Step 7)  
    FileViewer.viewDocumentConsole(parser.getDocument());  
    // Step 8)  
    FileWriter.writeFile(  
        "etc/telegraf/telegraf.conf", parser.getDocument());  
} catch (Exception e) {  
    e.printStackTrace();  
}
```


7 Annex 2: iStarstopmonitoring interface specification

7.1 Overview

This API contains the methods to access the ADAPT MM functionalities.

7.1.1 Version information

Version : 0.0.1

7.1.2 URI scheme

Host : localhost:8080

BasePath : /monitoringmanager

Schemes : HTTP

7.1.3 Tags

application : adapt application tag

7.2 Paths

7.2.1 createApplication

POST /api/applications

7.2.1.1 Description

Creates an application to be included in ADAPT monitoring

7.2.1.2 Parameters

Type	Name	Schema
Body	application <i>required</i>	Application

7.2.1.3 Responses

HTTP Code	Description	Schema
200	Application monitoring Created	Application
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

7.2.1.4 Consumes

- application/json

7.2.1.5 Produces

- */*

7.2.1.6 Tags

- application

7.2.2 getAllApplications

GET /api/applications

7.2.2.1 Description

Gets the list of all the applications in ADAPT monitoring, including their id and the status

7.2.2.2 Responses

HTTP

Code	Description	Schema
200	The list of all the monitoring elements in ADAPT monitoring is being returned	< Application > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

7.2.2.3 Consumes

- application/json

7.2.2.4 Produces

- */*

7.2.2.5 Tags

- application

7.2.3 getApplicationstatus

GET /api/applications/{appdescuri}

7.2.3.1 Description

Gets the status wrt monitoring of a given application

7.2.3.2 Parameters

Type	Name	Description	Schema
Path	appdescuri <i>required</i>	application description uri	string

7.2.3.3 Responses

HTTP Code	Description	Schema
200	Specific app monitoring is returned	Application
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

7.2.3.4 Consumes

- application/json

7.2.3.5 Produces

- */*

7.2.3.6 Tags

- application

7.2.4 updateApplication

PUT /api/applications/{appdescuri}

7.2.4.1 Description

Updates the monitoring status of a given application

7.2.4.2 Parameters

Type	Name	Description	Schema
Path	appdescuri	<i>required</i> application description uri	string

7.2.4.3 Responses

HTTP Code	Description	Schema
200	Status updated	Application
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

7.2.4.4 Consumes

- application/json

7.2.4.5 Produces

- */*

7.2.4.6 Tags

- application

7.2.5 deleteApplication

DELETE /api/applications/{appdescuri}

7.2.5.1 Parameters

Type	Name	Description	Schema
Path	appdescuri	<i>required</i> application description uri	string

7.2.5.2 Responses

HTTP Code	Description	Schema
200	App deleted form the monitoring list	No Content
204	No Content	No Content
401	Unauthorized	No Content

403

Forbidden

No Content

7.2.5.3 Consumes

- application/json

7.2.5.4 Produces

- $\backslash^*/^*$

7.2.5.5 Tags

- application

7.3 Definitions

7.3.1 Application

Name	Schema
monid <i>required</i>	String
status <i>required</i>	String
username <i>required</i>	String
application <i>required</i>	String

8 Annex 3: Telegraf template file

Excerpt of the `Telegraf.conf` file, where is detailed the template for the `HTTP_RESPONSE` input plugin, and below, two real examples of configuration of this plugin for two microservices. Below, the `INFLUXDB` output plugin are shown. These are input/output plugins that have been configured by ADAPT MM and during the unitary tests.

```
#####  
#                               INPUT PLUGINS                               #  
#####  
  
# # HTTP/HTTPS request given an address a method and a timeout  
# [[inputs.http_response]]  
#   ## Server address (default http://localhost) Sock Shop Front End  
#   address = "www.google.com"  
#   ## Set response_timeout (default 5 seconds)  
#   response_timeout = "5s"  
#   ## HTTP Request Method  
#   method = "GET"  
#   ## Whether to follow redirects from the server (defaults to false)  
#   follow_redirects = true  
#   ## HTTP Request Headers (all values must be strings)  
#   [inputs.http_response.headers]  
#     Host = "github.com"  
#  
#   ## Optional HTTP Request Body  
#   body = ''  
#     { 'fake': 'data' }  
#   ''
```

```
# ## Optional substring or regex match in body of the response
# ## response_string_match = "\"service_status\": \"up\""
# ## response_string_match = "ok"
# ## response_string_match = "\".*_status\".\"?:.?\"up\""
#
# ## Optional SSL Config
# # ssl_ca = "/etc/telegraf/ca.pem"
# # ssl_cert = "/etc/telegraf/cert.pem"
# # ssl_key = "/etc/telegraf/key.pem"
# ## Use SSL but skip chain & host verification
# # insecure_skip_verify = false
#
# HTTP/HTTPS request given an address a method and a timeout
[[inputs.http_response]]
    ## Server address (default http://localhost) Sock Shop Front End
    address = "http://172.26.252.81:18082/customers"
    ## Set response_timeout (default 5 seconds)
    response_timeout = "5s"
    ## HTTP Request Method
    method = "GET"
    ## Whether to follow redirects from the server (defaults to false)
    follow_redirects = true
    ## HTTP Request Headers (all values must be strings)
    # [inputs.http_response.headers]
    #   Host = "github.com"
#
## Optional HTTP Request Body
# body = ''
# # # # # # # # # # # # # # # # # # # # {'fake':'data'}
# ''
#
## Optional substring or regex match in body of the response
## response_string_match = "\"service_status\": \"up\""
## response_string_match = "ok"
## response_string_match = "\".*_status\".\"?:.?\"up\""
#
## Optional SSL Config
# ssl_ca = "/etc/telegraf/ca.pem"
# ssl_cert = "/etc/telegraf/cert.pem"
# ssl_key = "/etc/telegraf/key.pem"
## Use SSL but skip chain & host verification
# insecure_skip_verify = false
#
# HTTP/HTTPS request given an address a method and a timeout
[[inputs.http_response]]
    ## Server address (default http://localhost) Sock Shop Front End
    address = "http://172.26.252.81:18083/catalogue"
    ## Set response_timeout (default 5 seconds)
    response_timeout = "5s"
    ## HTTP Request Method
    method = "GET"
    ## Whether to follow redirects from the server (defaults to false)
    follow_redirects = true
    ## HTTP Request Headers (all values must be strings)
    # [inputs.http_response.headers]
    #   Host = "github.com"
#
## Optional HTTP Request Body
# body = ''
```

```
# # # # # # # # # # # # # # # # {'fake':'data'}
# ''

## Optional substring or regex match in body of the response
## response_string_match = "\"service_status\": \"up\""
## response_string_match = "ok"
## response_string_match = "\".*_status\".\"?:?\\.\"up\""

## Optional SSL Config
# ssl_ca = "/etc/telegraf/ca.pem"
# ssl_cert = "/etc/telegraf/cert.pem"
# ssl_key = "/etc/telegraf/key.pem"
## Use SSL but skip chain & host verification
# insecure_skip_verify = false

#####
#                               OUTPUT PLUGINS                               #
#####

# Configuration for influxdb server to send metrics to
[[outputs.influxdb]]
## The HTTP or UDP URL for your InfluxDB instance. Each item should be
## of the form:
##   scheme "://" host [ ":" port]
##
## Multiple urls can be specified as part of the same cluster,
## this means that only ONE of the urls will be written to each interval.
# urls = ["http://adapt.influxdb:8086"] # required
## The target database for metrics (telegraf will create it if not exists).
database = "decideh2020adapt" #
required

## Name of existing retention policy to write to. Empty string writes to
## the default retention policy.
retention_policy = ""
## Write consistency (clusters only), can be: "any", "one", "quorum", "all"
write_consistency = "any"

## Write timeout (for the InfluxDB client), formatted as a string.
## If not provided, will default to 5s. 0s means no timeout (not
recommended).
timeout = "5s"
# username = "telegraf"
# password = "metricsmetricsmetricsmetrics"
## Set the user agent for HTTP POSTs (can be useful for log differentiation)
# user_agent = "telegraf"
## Set UDP payload size, defaults to InfluxDB UDP Client default (512 bytes)
# udp_payload = 512

## Optional SSL Config
# ssl_ca = "/etc/telegraf/ca.pem"
# ssl_cert = "/etc/telegraf/cert.pem"
# ssl_key = "/etc/telegraf/key.pem"
## Use SSL but skip chain & host verification
# insecure_skip_verify = false
```