Automatic, multi-grained elasticity-provisioning for the Cloud

Cloud Monitoring Tool and Cloud Information System V1

Deliverable no.: 4.2
September 26th, 2014

CELAR is funded by the European Commission DG-INFSO
Seventh Framework Programmed, Contract no.: 317790

Table of Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

EC2	Elastic Compute Cloud
DB	Database
IaaS	Infrastructure as a Service
IS	Information System
JSON	JavaScript Object Notation
MS	Monitoring System
M2M	Machine to Machine
OS	Operating System
REST	Representative State Transfer
UI	User Interface
VM	Virtual Machine
WP	Work Package
XML	eXtensible Markup Language
JAR	Java ARchive file format

Deliverable Title	**Cloud Monitoring Tool and Cloud Information System V1**
Deliverable No.	**4.2**
Filename	CELAR_D4.2_finalrelease.docx
Author(s)	Demetris Trihinas, Athanasios Foudoulis, Daniel Moldovan, Demetris Andoniades, Nicholas Loulloudes, Stalo Sofokleous, George Pallis, Hong Linh Truong, Marios D. Dikaiakos
Date	26.09.2014

Start of the project: 1.10.2012
Duration: 36 Months
Project coordinator organization: ATHENA RESEARCH AND INNOVATION CENTER IN INFORMATION COMMUNICATION & KNOWLEDGE TECHNOLOGIES (ATHENA)

Deliverable title: Cloud Monitoring Tool and Cloud Information System V1
Deliverable no.: 4.2

Due date of deliverable: 30 September 2014
Actual submission date: 26 September 2014

Dissemination Level

X	PU	Public
	PP	Restricted to other programme participants (including the Commission Services)
	RE	Restricted to a group specified by the consortium (including the Commission Services)
	CO	Confidential, only for members of the consortium (including the Commission Services)

Deliverable status version control

Version	Date	Author
0.0	31.07.2014	Demetris Trihinas
0.1	20.08.2014	Demetris Trihinas
0.2	20.08.2014	Demetris Trihinas, Athanasios Foudoulis
0.3	22.08.2014	Demetris Trihinas, Athanasios Foudoulis, Daniel Moldovan
0.4	26.08.2014	Demetris Trihinas, Athanasios Foudoulis, Daniel Moldovan

0.5	03.09.2014	Demetris Trihinas, Athanasios Foudoulis, Daniel Moldovan, Demetris Antoniades, Nicholas Loulloudes
0.6	04.09.2014	Demetris Trihinas, Athanasios Foudoulis, Daniel Moldovan, Demetris Antoniades, Nicholas Loulloudes, Stalo Sofokleous
0.7	09.09.2014	Demetris Trihinas, Athanasios Foudoulis, Daniel Moldovan, Demetris Antoniades, Nicholas Loulloudes, Stalo Sofokleous, George Pallis, Hong Linh Truong, Marios D. Dikaiakos
0.8	24.09.2014	Demetris Trihinas, Athanasios Foudoulis, Daniel Moldovan, Demetris Antoniades, Nicholas Loulloudes, Stalo Sofokleous, George Pallis, Hong Linh Truong, Marios D. Dikaiakos
1.0	26.09.2014	Demetris Trihinas, Athanasios Foudoulis, Daniel Moldovan, Demetris Antoniades, Nicholas Loulloudes, Stalo Sofokleous, George Pallis, Hong Linh Truong, Marios D. Dikaiakos

Abstract

The goal of this deliverable is to present a thorough report on the design, research and implementation status of the second version (v2.0) of the Cloud Information and Performance Monitor layer, which consists of three, integrated, components: (i) the CELAR Monitoring System; (ii) the Multi-Level Metrics Evaluation Module; and (iii) the CELAR Information System. Specifically, this document introduces the first version (v1.0) of the CELAR Information System, focusing on documenting its requirements, both functional and non-functional, an initial design and its architecture, the challenges, which it addresses, and how it interacts with other CELAR components. Additionally, it presents the second (v2.0), and final, versions of the CELAR Monitoring System and the Multi-Level Metric Evaluation Module as they shortly come to an end, by focusing on documenting their new features and improvements, in comparison to what was showcased in D4.1. Finally, it provides an evaluation study, which documents how the Monitoring System and the Multi-level Metrics Evaluation Module requirements were verified to address their assigned purposes.

Keywords

Cloud Computing, Elasticity, Cloud Monitoring, Real-Time Monitoring, Application Monitoring, Cost-Evaluation, Information System, JCatascopia, MELA

1 Introduction

CELAR aims to enhance current cloud infrastructure functionality by providing fully-automated, multi-grained elasticity control for elastically adaptive cloud services. This is accomplished by incorporating, in its core, intelligent decision-making algorithms, which evaluate elasticity at multiple levels of a cloud service and consider the impact of enforced actions in relation to *cost*, *quality* and *allocated resources*.

CELAR employs a MAPE-K control loop (Monitoring, Analyzing, Planning, and Executing using Knowledge) [Huebscher 2008] to provide elasticity support and manage cloud service resource utilization. The first step in the control loop is to gather monitoring information regarding cloud service resource utilization and performance and then, based on this information, analyze it, and decide if an elasticity control action should be enforced. However, automatic resource provisioning is challenging due to the fact that monitoring and managing elastic cloud services is not a trivial task [Trihinas 2014b]. Specifically, in order to facilitate the needs of an intelligent and complex Decision-Making module, heterogeneous types of information is required from different data sources located at multiple levels of the cloud which is then processed, structured and enriched to obtain a meaningful substance for the Decision Module to enforce appropriate elasticity actions. Such information is the following:

1. low-level system-wise monitoring information (i.e. CPU utilization, memory allocation, network traffic);
2. application-specific behavior and performance metrics (i.e. response time, latency, throughput, number of active users);
3. aggregated single instance (i.e. average CPU utilization of a server per hour) and tier-level metrics (i.e. average database backend throughput, min/max active connections for business logic tier) which are then mapped to the application structure;
4. cost-enriched monitoring information based on cloud provider price schemes;
5. elasticity behavior analysis in terms of cost, quality and resource utilization;
6. cloud provider specific information (i.e. price schemes, user quotas, available resizing actions, instance flavor types);
7. application-related information (e.g. application topology structure, elasticity requirements, past and current deployment history);

Depicted in Figure 1 is the revised CELAR System architecture that was introduced in Deliverable D1.2 [D1.2]. The layer in charge of providing the Decision-Making module with the listed information is the *Cloud Information and Performance Monitor layer* which is comprised of three major components:

- The **CELAR Monitoring System (CELAR MS)**, which is a fully-automated, multi-layer and platform independent monitoring system that runs in a non-intrusive and transparent manner to any underlying virtualized infrastructure. The MS is utilized by CELAR to collect monitoring metrics from multiple levels of the underlying cloud platform as well as performance metrics from deployed cloud services and afterwards it distributes them to subscribed users and platform operators. Subsequently, the CELAR Monitoring System is in charge of (1) and (2) of the above listing;
- The **Multi-Level Metrics Evaluation Module**, which provides CELAR with composable cost-related monitoring information, by firstly, mapping monitoring metrics collected by the underlying Monitoring System to the structure of the application and then enriching cost-wise the obtained metrics based on the cloud

provider's pricing schemes. Furthermore, the Multi-Level Metrics Evaluation Module provides cost evaluation and application elasticity analysis reports which are critical for the Decision Module to enforce elasticity actions. Therefore, the Multi-Level Metrics Evaluation Module is in charge of (4) and (5), while (3) is provided in collaboration with the Monitoring System;

- The **CELAR Information System (CELAR IS)** which collects, filters, processes and analyses application deployment information and distributes the extracted information to the end user, providing insights concerning the lifecycle and performance of deployed elastic cloud services. Specifically, the CELAR IS utilizes both collected monitoring information and cost-enriched metrics to provide a behavior analysis over a specific cloud service deployment based on application-specific information (i.e. application topology, scaling requirements) and cloud platform specific information (e.g. available resizing actions). Moreover, the CELAR IS also provides users with the ability to compare the current deployment with other (previously issued) deployments to determine which configuration parameters maximize the satisfaction of the Application User requirements (i.e. maximum performance, resource utilization, minimize cost). Subsequently, the CELAR Information System is in charge of (6) and (7).

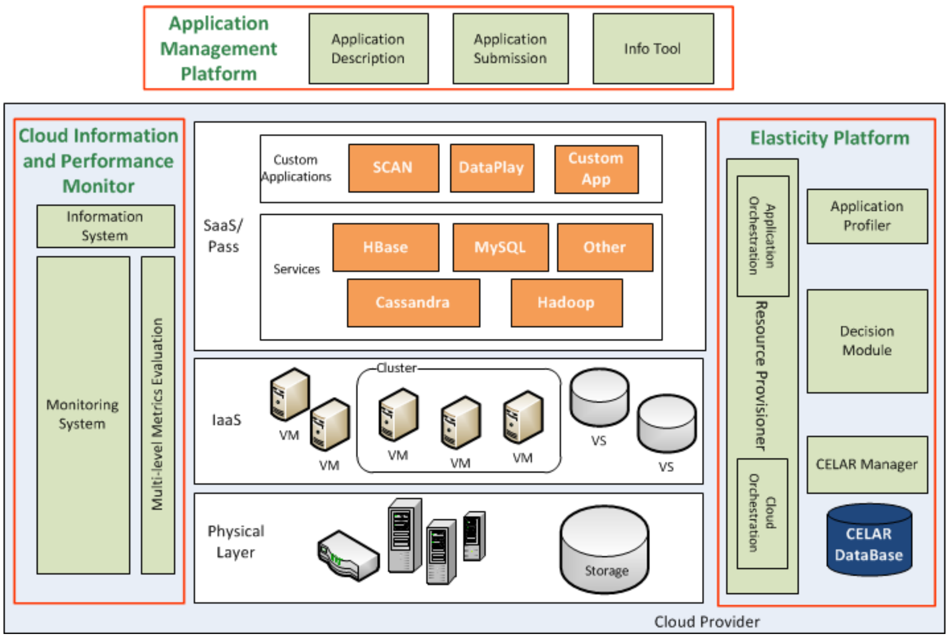


Figure 1: CELAR System Architecture

This deliverable continues the work initially described in Deliverable 4.1 [D4.1] and documents the progress made so far in relation to the three components which comprise the *Cloud Information and Performance Monitor layer*. Parts of this deliverable are based on a number of published scientific papers [Trihinas 2014a] [Trihinas 2014b] [Moldovan 2013] [Moldovan 2014], which introduce core concepts of the components part of CELAR Work Package 4 (WP4).

1.1 Purpose of this Document

This deliverable introduces the second version (v2.0) of the *Cloud Information and Performance Monitor layer* by documenting the progress made until month 24 in WP4 of the EU-funded CELAR Project. At first, we aim to introduce the first version (v1.0) of the

CELAR Information System (Task 4.3) which started on month 12 and therefore, was not part of D4.1. Specifically, for the CELAR Information System we present: (i) its requirements, both functional and non-functional; (ii) an initial design and its architecture; (iii) an analysis of the challenges which must be addressed when gathering and manipulating information from several different heterogeneous data sources; and (iv) the necessary communication and data exchange with other CELAR modules, where special emphasis is given to how the Information System interacts with the Monitoring System and the Multi-Level Metrics Evaluation Module. Secondly, we present the second (v2.0), and final, versions of the CELAR Monitoring System (Task 4.1) and the Multi-Level Metric Evaluation Module (Task 4.2) as they shortly come to an end. We focus on documenting their updated requirements, introduce new features and provide a final documentation report. Furthermore, emphasis is given to describing how composite cost estimation is addressed and how it facilitates the needs of the CELAR Project. Finally, we provide, for the Monitoring System and the Multi-Level Metrics Evaluation Module, an evaluation study, which documents how the listed requirements were verified to address their assigned purposes.

Note: As this deliverable documents the second version of the CELAR Monitoring System and Multi-Level Metrics Evaluation Module which were initially described (design and implementation) in D4.1 [D4.1], we refer readers with no previous knowledge of these components and the CELAR System in general, to D4.1. To enhance readability, a short overview of the CELAR Monitoring System and Multi-Level Metrics Evaluation Module is provided in the beginning of sections 3 & 4 respectively. Furthermore, when referring to concepts which were described in D4.1, a reference is provided to the respected section in D4.1 where it was first introduced.

1.2 Document Structure

The rest of the deliverable is structured as follows: Section 2 presents a study of the State-of-the-Art in Cloud Information Systems and an updated study of the State-of-the-Art in Cloud Monitoring and Cost-Evaluation which was provided in D4.1. Section 3, introduces the CELAR Information System by presenting its requirements, design and the current implementation status. Sections 5 and 6, focus on the second, and final, versions of the CELAR Monitoring System and Multi-Metric Evaluation Module respectively, where new features are presented alongside an evaluation study of both systems. Section 7 concludes the deliverable and outlines our future work.

2 State of the Art

This section presents an updated study of the State-of-the-Art in Cloud Monitoring and Cost-Evaluation which was initially provided in D4.1 and furthermore, a study of the State-of-the-Art in Cloud Information Systems.

2.1 Cloud Monitoring Tools

Cloud monitoring is a very active research area due to the challenges presented when monitoring elastically adaptive cloud services which may, in turn, be ported to different cloud platforms during their lifecycle or may be distributed across multiple platforms at the same time. In what follows, we update D4.1 study of the State-of-the-Art in Cloud Monitoring by presenting a number of proposed monitoring approaches, which were not mentioned in D4.1, to address cloud monitoring *elasticity support*, *portability* and *interoperability*.

OpenStack Ceilometer [Ceilometer] aims to deliver a single central contact point, via a REST API, for billing systems and other entities who wish to acquire measurements across OpenStack core components by either polling the cloud hypervisor for compute node utilization metrics or by intercepting messages in OpenStack's notification queue that contain monitoring information. Nonetheless, what makes Ceilometer an interesting project is its database interface which provides cloud administrators with the ability to implement and use their own database backend for metric storage. However, Ceilometer's metric collecting mechanisms are tightly coupled to OpenStack which limits both its portability and interoperability.

In contrast to Ceilometer, the following two, cloud specific, approaches attempt to provide a first solution to monitoring cloud platforms in a portable and interoperable manner. Rak et al. [Rak 2011] introduce the mOSAIC cloud monitoring approach, which collects low-level monitoring information from different cloud platforms, via the mOSAIC API, and application-level metrics from other monitoring solutions. However, this approach is limited to only cloud platforms supported by the EU-funded mOSAIC project [mOSAIC] and is a centralized monitoring approach suitable for only small-scale cloud service deployments. Similarly, Al-Hazmi et al. [Al-Hazmi 2012] propose a monitoring system for federated cloud environments which uses Zabbix [Zabbix] to collect low-level monitoring metrics from multi-cloud deployments and distribute them to user-deployed aggregators through message notification queues. Although an interesting approach, its feasibility, as with the mOSAIC approach, is limited to monitoring cloud platforms supported only by the EU-funded BonFIRE project [BonFIRE]. In contrast to the three aforementioned approaches, the CELAR Monitoring System was designed and implemented to be both portable and interoperable allowing system administrators and cloud users to monitor their deployed cloud services on any underlying virtual infrastructure.

To address the above challenges, a number of general purpose monitoring systems have also been proposed. Xiang et al. [Xiang 2010] introduce VMDriver where the cloud hypervisor management VM installs a monitoring driver when instantiating a new guest VM, which is then used to provide a standard interface for high-level monitoring or management tools (e.g. information systems, analytic tools) to re-construct monitoring information in an OS-independent manner. A more interesting approach is used in the EU-funded 4CaaSt project [4CaaSt] to monitor PaaS application deployments where a common interface is exposed to integrate "monitoring sources", allowing for developers to create adaptors which integrate their underlying monitoring operations or existing monitoring system (i.e. ganglia, nagios) with the 4CaaSt monitoring service. Despite the initial overhead required to create and integrate a "monitoring source" with the 4CaaSt monitoring service, it is still not clear what is the runtime overhead introduced by requiring in short monitoring intervals (e.g. range of a few seconds) to "translate" metrics collected from monitoring sources to a 4CaaSt monitoring service readable format.

In regards to elasticity support, two approaches should also be added to the study. Specifically, Quoc et al. [Quoc 2014] propose DoLen, a user-side multi-cloud application monitoring approach which can monitor elastic cloud services distributed across multiple cloud platforms but as a centralized approach it is suitable only for small-scale deployments. In contrast to DoLen, a more interesting approach is Panoptes, proposed by Uriarte et al. [Uriarte 2014], which utilizes a pub/sub communication model between agents and servers (named controllers) to enhance scalability and performance when monitoring applications deployed on a private cloud. The limitations of this approach is that, in a similar manner as with a (monitoring instance) directory service, when a new agent is instantiated it must first contact a central communication point to retrieve a list with the available monitoring servers, and then select one as well as its configurations (e.g. metrics to collect). In contrast, to the above approaches, the CELAR Monitoring System features no central, and potential single point of failure, component(s) and to provide elasticity support it enables a variation of the publish and subscribe protocol to eliminate the need of a monitoring agent first, contacting a central service to retrieve a list of available monitoring servers.

2.2 Cost-Evaluation

In what follows, we update the study over the State-of-the-Art in Cost Evaluation presented in D4.1, focusing on work dealing with *run-time* cloud application cost monitoring, an area we focus on in WP4.

The Optimis [Optimis] project approaches cloud application cost estimation from the infrastructure provider point of view, deciding based on a set of *Trust*, *Risk*, *Eco-efficiency*, and *Cost* (TREC) parameters how to organize VMs in the cloud. The introduced mechanism is completely transparent to the cloud user, providing functionality only to the cloud provider. CELAR differs as we apply the cloud infrastructure user perspective, and provide a cost estimation mechanism that can be used by a wide range of infrastructure users in estimating cost for their applications. The same infrastructure provider perspective is also adopted by 4CaaSt [4CaaSt], which provides an internal accounting service used to evaluate cost per usage and per time period for different slices of cloud applications. While such an approach could be adapted to expose cost to infrastructure users, it provides only individual cost metrics. In CELAR we provide cost evaluation at different granularities, from individual cost elements for application components, to overall cost of entire application, enabling a large array of users to retrieve the level of cost information they require. While MODAClouds [MODAClouds] adopts the same cloud user perspective in cost monitoring as CELAR, exposing to the cloud user cost of running its application, it leverages existing cloud APIs for retrieving cost information, limiting their approach to the type and format offered by cloud providers. CELAR differs as we evaluate and return cost according to the logical application structure, not only at the virtual infrastructure level supported by cloud providers. This enables CELAR users to understand the cost of their application at the application level, for application components, no matter if they run on one or multiple cloud services (e.g., VMs), and not of the individual cloud services hosting those components.

Moreover, none of the above approaches take into account the full complexity of cloud pricing schemes, as cloud providers could offer services under different configurations and possible combinations, with different costs.

2.3 Cloud Information Tools

A Cloud Information System is expected to organize, expose and transform data, making them meaningful for the application user, thus helping the user filter the provided information and to obtain insights for the application development and deployment processes. Current cloud information and visualization approaches do not provide a complete solution. To overcome this issue most cloud application developers use a mixture of services, both proprietary and third party.

All big cloud market players provide resource marketplaces to enable the application developer to search for and select desirable resources and tools for managing application deployments [Amazon AWS] [Google Compute Engine] [Microsoft Azure]. Furthermore, both Google and Azure provide tools to help analyze and visualize a deployment's performance and cost. Amazon AWS provides to its users both AWS CloudTrail [CloudTrail] and AWS Trusted Advisor [AWS Advisor]. CloudTrail is a web service that records the API calls issued to AWS from a user account. Using this API call history the user can then analyze application resource changes and take decisions to improve the application's performance. Trusted Advisor inspects the user's AWS environment and provides recommendation for saving money, improving system performance and help close security gaps.

All the above systems are bind to a single cloud provider. A number of tools and scientific works have tried to address this issue and extend analysis and recommendation functionality to multiple cloud providers. Mist.io [MistIO] is a *run everywhere* tool that allows the user to monitor and control her cloud application, enabling her to optimize spending, security, availability and performance. It provides the ability to monitor system and application metrics, get real-time analytics, get alerts and set automate responses. CloudCheckr [CloudCheckr] provides cloud deployments usage and cost monitoring and analysis, allowing the user to understand and optimize her resource usage and cost. Cloudyn [Cloudyn] provides similar functionality to the user, with hooks for additional cloud providers, public, private and hybrid. Cloudaware [Cloudaware] offers a plethora of tools for managing deployments and extracting cost and usage analytics and claims to be the 'most compressive AWS management platform'. CloudVertical [CloudVertical] also provides a mixture of performance usage and cost analytics. Cloudability [Cloudability] and teevity [teevity] focus more on cost monitoring and analysis.

A different area of interest, also addressed by a number of tools and scientific publication, is to provide advisement to the cloud application user/developer on which cloud provider better fits her application needs. Cloudscreener [Cloudscreener] and Cloudorado [Cloudorado] are two tools that help users select the best virtual resource configuration, in terms of performance and cost, among several providers, by analyzing a small number of high-level performance indicators. Kang et al., in [Kang 2011], presented Cloudle, a cloud service search engine that based on user-provided metrics of interests, searches for the best cloud provider. The authors define a number of different similarity metrics that allow them to properly compare the different cloud offerings. Cloudle is also based on a cloud ontology, introduced in [Han 2010], that represents the cloud offerings using a set of Cloud concepts, individuals of those concepts, and the relationship among those individuals. The authors claim that combining Cloudle and the cloud ontology has a better performance in finding the appropriate cloud service. Li et al. introduce CloudCmp [Li 2010], a systematic comparator of the performance and cost of cloud providers. Their evaluation showed that cloud provider offered services vary widely in performance and cost, underscoring the need for thoughtful provider selection. Garg et al. [Garg 2011], propose a framework and a mechanism to measure the quality and prioritize Cloud services, with the goal of automatically indexing cloud providers based on the user's needs.

All the above systems are agnostic to the application topology and consider each deployment as a group of virtual instances. Furthermore most of the analytic services presented are unaware of the 'deployment' terminology and handle each instantiated resource, under the same user account, as belonging to the same abstract group. The CELAR Information System, using the application topology, can reveal the multi-tier aspect of an application and the interactions between the tiers.

An additional limitation of all the above system is that they do not consider an iterative deployment procedure. Usually when developing and application there is an extended feedback loop between development and deployment that allows the application developer to optimize the application. CELAR keeps track of this loop, by allowing different application versions and deployments, thus allows the CELAR Information System to provide comparisons between the different versions of the same application and different workloads of the same version, enriching the information provided to the user and allowing for optimized cost and performance decisions.

3 CELAR Information System

An Information System is the entity which purpose is to help the user interact with the data stored in a system. Through such an Information System, the user has the ability to search, view, and navigate through the available data of the underlying system.

The CELAR Information System (CELAR IS) is the CELAR component in charge of providing, per cloud application, usage analytics by processing information regarding resource utilization and cost-enriched monitoring data, collected during the lifecycle of an elastic cloud service deployment. Furthermore, the CELAR IS enables Application Users with the ability to query, explore, compare, and visualize historic data collected from past deployments[1] and previous versions[2] of a cloud service.

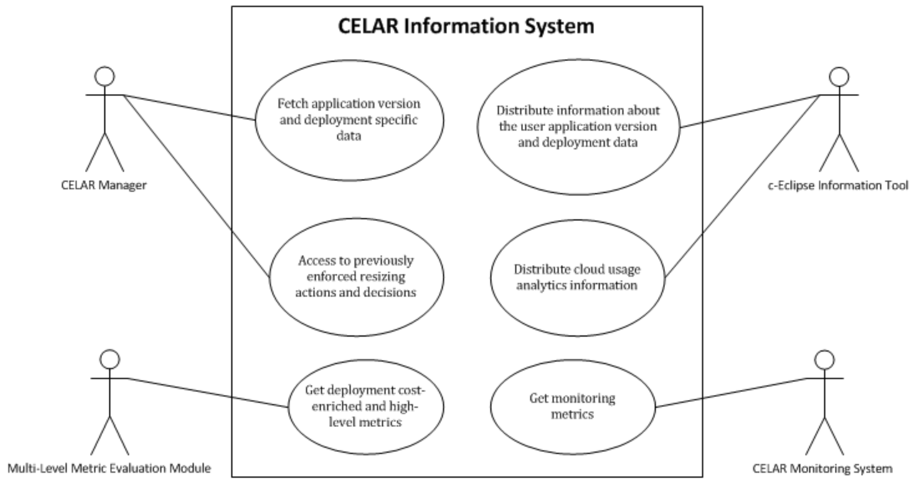
For the rest of the document we will use the term Application Version Data and Deployment Data to refer to the data stored in the CELAR DataBase, which, in turn, is accessible from the CELAR IS. Moreover, deployment data consists of: (i) deployment id (ii) start/end time of the deployment, (iii) instantiated resources along with the duration they were allocated and the resource type. The application version data, includes: (i) version and application id, (ii) submission time, (iii) application structure along with the defined resources to be utilized. The former results from the CELAR DataBase schema presented in D3.2 [D3.2]. The later will be documented thoroughly in the upcoming deliverables.

3.1 Requirements

The requirements of the CELAR IS, both functional and non-functional, were first documented in Deliverable 1.2 [D1.2] (Sections 3.3 and 3.4). In the following subsections we present an updated version of the previous requirements along with its actors and use cases.

3.1.1 Actors

Any CELAR component that interacts with the CELAR Information System and can be considered as an actor. A brief view of the actors and the various use-cases is depicted in Figure 2. These use-cases are further described in Section 3.1.2 (Table 1).



CELAR Information System
Fetch application version and deployment specific data
Distribute information about the user application version and deployment data
Access to previously enforced resizing actions and decisions
Distribute cloud usage analytics information
Get deployment cost-enriched and high-level metrics
Get monitoring metrics
CELAR Manager
c-Eclipse Information Tool
Multi-Level Metric Evaluation Module
CELAR Monitoring System

Figure 2: CELAR Information System Use Case Diagram

[1] With the term past deployment, we refer to previous deployments of the same version of a cloud service.
[2] With the term previous versions, we refer to different versions of a cloud service where changes (i.e. to the topology, elasticity requirements, scripts, artifacts, etc.) have been applied.

CELAR Manager

The CELAR Manager is the gateway and interface through which the CELAR IS obtains information from the CELAR DataBase. The CELAR IS interacts frequently with the CELAR DataBase in order to extract and present, to Application Users (via c-Eclipse) information regarding current and past cloud application deployments and versions. The aforementioned information is also utilized by the CELAR IS to process, calculate and present cloud resource usage analytics[3].

CELAR Monitoring System

The CELAR Monitoring System (CELAR MS) provides near real-time monitoring information regarding resource utilization and application behavior performance. The CELAR IS periodically compiles summaries of the current status of each of the cloud services. In a number of cases that can be identified from the collected monitoring metrics, it detects situations that will require the Application User's immediate attention. Furthermore, the CELAR IS aggregates in higher granularity, filters and stores metrics extracted from the CELAR MS to the CELAR DataBase for historical purposes since monitoring information collected via the CELAR Monitoring System is not maintained after the termination of a cloud service deployment.

Multi-Level Metric Evaluation Module

The Multi-Level Metrics Evaluation Module provides CELAR with composable cost evaluation and estimation. The CELAR IS extracts from the Multi-Level Metrics Evaluation Module cost-enriched monitoring information and presents to Application Users a summary of their cloud services current deployment costs. Moreover, the Multi-Level Metrics Evaluation Module creates according to the application structure and composition rules [D4.1] aggregated metrics creating a high-level view of how a cloud service and its components perform. The calculated cost and aggregated monitoring data is also utilized by the CELAR IS to enhance its cloud usage analytic reports.

c-Eclipse Information Tool

The Information Tool is the visualization tool part of the c-Eclipse Application Management Platform (presented in Deliverable D2.1 [D2.1] and D2.2 [D2.2]), in charge of providing Application Users with graphical access to information such as the status and cost of the current deployment as well as cloud usage analytics and behavior analysis summaries. By utilizing the Information Tool, Application Users are also immediately capable of querying and exploring cloud service past deployment and versioning historic information.

3.1.2 Uses Cases

The following Table provides a detail overview of the CELAR Information System use cases:

[3] Cloud usage and cost analytics assist users by providing a thorough understanding regarding cloud resource utilization and cost. An analytics procedure combines both resource usage data (e.g. monitoring metrics) and extracts insights such as the trend over a time period. For more information see Section 3.2.3

	Use Case Name	Actor	Description
1	Fetch application version and deployment specific data	CELAR Manager	In order to provide analytics it is essential for the CELAR IS to have information about the cloud service topology and the resources utilized during a deployment. Further on, after getting the Application User input from the IS-UI[4], the CELAR IS will formulate the appropriate requests to the CELAR Manager, through its API, to obtain and present application version and deployment specific information
2	Access to previously enforced resizing actions and decisions	CELAR Manager	The CELAR IS requires the enforced resizing actions and decision knowledge in order to provide meaningful analytics. Moreover, the CELAR IS through its UI will present a list of the already taken decisions
3	Get monitoring metrics	CELAR Monitoring System	CELAR IS will present a summary regarding the current status of a deployment using monitoring metrics collected from CELAR MS. Metrics also need to be obtained as input to the analytics process
4	Get deployment cost-enriched and high-level metrics	Multi-Level Metrics Evaluation Module	Cost information calculated from Multi-Level Metrics Evaluation Module will help the CELAR IS expose the overall current cost of a deployment to the user. The cost information alongside with aggregated metrics information will further enhance the analytics process
5	Distribute information about the user application	c-Eclipse Information Tool	c-Eclipse Information Tool will act as a front-end to the CELAR IS enabling the end-user to access stored

[4] In terms of abstraction we are referring to the visualization tool which visualizes the CELAR IS calculated data and the Info Tool which integrates the IS presentation layer to c-Eclipse as IS-UI.

	version and deployment data		information regarding past deployment and previous versions of a cloud service
6	Distribute cloud usage analytics information	c-Eclipse Information Tool	The end-user will have the ability to request usage analytics, over past deployment data. The result calculated from the CELAR IS will be reported through the c-Eclipse Information Tool. The Tool will enable the comparisons among two or more deployments

Table 1: CELAR Information System Use Cases

3.1.3 Functional Requirements

In this section we define the functional requirements of the CELAR Information System which are derived from the previously described use cases along with the general CELAR use cases as described in the D1.1 [D1.1] and D1.2 [D1.2].

Access to Deployment Data

During a deployed application's lifecycle, many metrics are continuously collected. Plotting and interpreting the data as well as navigating throughout it in a temporal manner, can provide meaningful insights into an application's performance. These kind of post-run observations are crucial to an Application User that wants to evaluate and re-design her application. The CELAR IS needs to have access to such data and provide the necessary mechanisms that the user will use in order to visualize and explore the data concerning each of the used resources. Additionally, comparing the above data with data from the Profiler ([D3.2], section 3.3.3), which had stressed the application under different situations, is a desirable feature and an ability that CELAR IS should have. Furthermore, CELAR IS should combine this data with resource cost information, provided by the Multi-Level Metrics Evaluation Module, to provide comprehensive information about an application's overall expenses, which can result on pin-pointing costly components/instances.

Cloud Resource and Usage Analytics

The aforementioned collected data are valuable for the expert user or the user, who has enough time to go deep into the application resources usage characteristics. Most of the users will need from the system to analyze the collected data and provide meaningful reports that summarize the performance and cost of their application. Ultimately, such reports will enable the end-user to take informed decisions given the requirements and constraints of its future deployments. To this end, CELAR IS needs to encapsulate a "cloud usage analytics" engine that will take as input the data collected during a deployment and will export reports to the user with the outline of their deployments.

Access to Application Versioning Data

A core feature of CELAR is the ability to describe the application structure. The application structure defines the number and types of application components and the relationships between them [D2.1]. The CELAR IS should be aware of an application's topology and adapt its analytic procedures to it. Specifically, the CELAR IS should be aware on the different structural changes that took place among various deployments of the same application. In turn such knowledge can provide a more holistic and comparative representation of the applications' overall performance, as well as its individual components. This structural information representation helps the user to understand better the insights that the CELAR IS will provide.

Compare Different Deployment Configurations

A CELAR user can create multiple different application configurations, changing every time, either the topology, the used resources or the elasticity rules. Moreover, the Smart Deployment Module [D3.2] given an abstract application description can generate the optimal configuration for the user's needs. The user needs a comparison mechanism in order to evaluate each one of the configurations and choose the best one, according to her needs. CELAR IS should provide such a comparison mechanism that will contrast, both visually and programmatically, the collected deployment data or profiling data and the calculated analytics for each one of these version deployments and help the user decide which is "the best one".

Express Complex Condition Based Queries

As mentioned before, the CELAR IS needs to provide the user with a mechanism to access and explore the application and deployment data. Depending on the amount of data, finding the precise information may cost a lot of effort to the user. CELAR IS needs to seamlessly provide the user, through its UI, the option to apply multiple filters over the available data. For example, the user should have the ability to issue queries like "*what was the memory usage percentage of resource X, when CPU usage percentage was over 80%*". This query requires from the CELAR IS to 'run' two different queries and combine the results before responding back to the user.

Be Agnostic to Underlying Data Sources and Cloud Infrastructure

The CELAR IS should be agnostic to the underlining infrastructure. This means that CELAR IS should be implemented as a service and be able to operate over the existing, and future, cloud platforms. CELAR IS needs various data, as input, in order to operate and it must be also transparent to those. To do so, the CELAR IS should provide the needed hooks and data source interface(s) in order to be able to connect and acquire data from various systems (e.g. CELAR MS) without any intervention to its code.

3.1.4 Non-Functional Requirements

In this section we define the CELAR Information System non-functional requirements resulting from the above functional requirements.

Elasticity Awareness

Being aware of the elastic nature of an application is an essential requirement for CELAR IS. More specifically the CELAR IS must take into account and inform the user about the dynamic resource allocation. This should be done both, when CELAR IS displays deployment data and when it presents analytics reports.

Information Consistency

It is essential for an information system to provide accurate information. The CELAR IS should not keep duplicate data or data that is not consistent with the system's current state.

Low Latency

Access to accurate information in a timely manner is crucial. The Information System should return accurate information, to its users, in a small time frame.

3.2 Analysis and Design

To facilitate the requirements presented in Section 3.1, we are building a system capable of providing access to user and application data stored by the CELAR System. The system will be aware of an applications' topology and its elastic nature, which combined with the available data and through statistical analysis, will result in insights for improving the application's performance, and minimize deployment cost. Moreover, given the available data our system will allow for comparisons between past application deployments and version data.

The system's functionality can be grouped in two main categories:

- Visualization: Presenting the available data from CELAR DataBase and other CELAR components. Providing also to Application Users the ability to explore and navigate through this data.
- Analytics: Calculating and analyzing the retrieved data to provide insights. CELAR IS will offer two main categories of analytics: (i) performance and (ii) cost. CELAR IS aims to provide insights on how a deployment configuration behaves according to these two aspects and also provides a view of the tradeoff between these two. Those two categories can be further divided based on the application topology, so CELAR IS can provide analytics based on a particular component or a group of components of the same type. In terms of performance analytics the Application User can inspect (i) the usage of an entity[5], (ii) the utilization of an entity (iii) the state of the entity over time and whenever is applicable (iv) visualizations intercepting the data input rate (e.g. requests/sec) and the entity usage (CPU, memory usage). Regarding cost analytics the CELAR IS will enable the Application Users to inspect (i) the per hour/month/year cost of an entity, (ii) the change of an entity's cost over the time.

In the following sections we present the features of the CELAR IS and we conclude proposing a high-level architecture that can facilitate the system requirements.

3.2.1 Navigate Through Application Data

As stated before the CELAR IS will enable the user to query, explore and navigate through the data stored in CELAR DataBase and any other CELAR module. To tackle the need of "searching for data", the CELAR IS will provide a search engine interface, giving the ability to the Application User to browse through her data. The CELAR DataBase schema, defined in D3.2 [D3.2], is well structured and it clearly defines the fields, properties and relations between the CELAR data. Moreover, the CELAR Manager provides access to the CELAR data, and the aforementioned relationships, through a

[5] As an entity we imply one of the following: (i)a resource (ii) a component, (iii) a group of components or (iv) a deployment

REST API (D3.2). The CELAR IS functionality relies on exposing the available fields to be used for search and combining the appropriate API call(s), according to the user input. To perform such combinations the CELAR IS should be intelligent enough to perform additional calculations and two-step queries.

Using the described search functionality the Application User can express queries such as: (i) *retrieve all the application versions, which make use of the module X and they have been submitted between timestamp_T1 and timestamp_T2;* (ii) *retrieve all the application versions that are associated somehow with resource X*.

The CELAR IS will present the results in a constructive and user friendly manner, sorting them by the available fields or visually grouping them. In a scenario like the one above, the CELAR IS could sort the results by 'submission time' and visually group the versions that belong to the same application. The ways that the CELAR IS can order or group the data varies depending on the information that the CELAR IS has access too. For example when the Application User searches through past deployment data and cost or performance information for each available deployment, the CELAR IS can sort the result according to that information.

After receiving the required information, either using the search tool or navigating through the UI, the Application User will be able to explore and access additional data regarding the specific entity (e.g. application version or deployment). Since the information is present and accessible, the CELAR-IS-UI will provide all the required hooks to allow the user to tailor the data presentation to her preferences. Referring to an application version the user can view information regarding the topology of the application, how the application is described and what software and virtual infrastructure the specific application version utilizes. Moreover, referring to a deployment the user can access the performance metrics collected and resizing actions that have occurred. The user will be able to select visualization charts using the metric values and navigate through the duration of a deployment. Visualization can be defined either based on a specific time window, or zooming in the time a specific decision (resizing action event) occurred. The user can further browse through historical monitoring data of an application by expressing condition-based queries to identify if and how a metric value affects another one (e.g. ram utilization of the application server, when CPU usage was over 80%).

3.2.2 Historical Monitoring Data

Due to storage limitations, current cloud monitoring systems solutions limit the amount of raw monitoring data kept. In most cases a certain checkpoint is defined, after which the system starts deleting or filtering old data. For example, the CELAR Monitoring System, when utilizing as a database backend a relation database (e.g. MySQL), stores monitoring data for a maximum configurable timespan (e.g. two weeks), after that point the data will be purged. A similar approach is followed, as well, by commercial cloud monitoring services such as Amazon's CloudWatch [CloudWatch], which also retains historical monitoring data for two weeks. This purging, in many cases, depends on:

- How important is it to retain historical information about each instance. For example, for auditing purposes or post-analysis procedures.
- The amount of storage that is available to store collected data. This translates to the amount of money a person is willing to spend on cloud storage. So if space is an issue, we have to purge data on a more frequent basis.

Depending on the user-base and their requirements, it is needed to find the best combination of data precision and utilized storage space, to provide intuitive access to historical monitoring data; taking also into account data retrieval latency. High latency is noticeable while navigating and presenting data (see Section 3.2.1), but is highly related to the way data is stored and retrieved. For example if metrics pile up then query time is large (e.g. a query may be in need of multiple joins on time-series data such as monitoring data) and therefore latency is an issue.

The CELAR Information System is designed under the assumption that higher data resolution helps when actively addressing the *forensic analysis of an incident*. Such an incident might be the failure of an instance, where the user needs to identify, through the monitoring information the invalidation of an elasticity rule upon which the decision module [D5.1] needs to take action. In any other case, aggregated data is enough in order to provide past deployment previews and analytics.

For handling historic monitoring data, the CELAR IS needs to focus on two areas:

- Retrieve the appropriate information and present it to the user.
- Store data on CELAR DB and/or aggregate data to be used for historical purposes.

The former has been already discussed in the previous section. In regards to the second point, the CELAR IS needs to contact the underlying instantiated CELAR Monitoring System for a specific deployment at pre-specified intervals and obtain all the metrics collected during the last interval. Since the CELAR Monitoring System will, most likely, purge collected metrics every 2 weeks, we need to select a metric collection interval that is smaller than the configurable monitoring metric purge period. For example, a safe time to collect metrics can be 1 week. Therefore, every one week CELAR IS will aggregate and store in the CELAR DataBase the metric values collected during the previous week. Even if just storing the mean value of a metric might be enough for most of our calculations, to increase precision we choose to store also the min and max values of the metric measured values. Having the average value in conjunction with minimum and maximum can lead to insights about the concentration of the values. The same procedure must be also done for Multi-Level Evaluation Module in order to be available for later view and analysis. The Multi-Level Evaluation Module as previously described can feed the CELAR IS with structure and cost-enriched metrics that can further help the analytics procedure.

Another important variable that needs to be defined is the aggregation window, meaning how many metric values or better how often we will produce this summarized value. As we described before this value is a trade-off between the precision we want to achieve and the available storage, or the storage budget. In most cases, this decision is left to the user, but since the CELAR IS needs this data to provide cloud usage analytics it is essential to provide a suggested threshold depending on the maximum time spectrum of the collected data. For example, if the application is running for a week, keeping all the raw data might be a viable option. In the case the application is running for a year, we might need to use an aggregation window in the order of a few hours or even choose to have multiple aggregation windows with respect to the available storage. Having multiple aggregation windows can help retain accuracy when the available storage is limited. For example if we have a strain of 1GB maximum storage a viable approach it will be to aggregate the last six months of data with a ten minutes window, the first year data with a two hour window and second year data in a twelve hour window.

As stated in D4.1 the Multi-Level Metrics Evaluation module also aggregates and stores metric values for its own purposes. To avoid duplicating both the aggregation effort and the stored data used, the CELAR IS needs to be aware of what is previously calculated. Even if the output of the analysis of the Multi-Metric Evaluation Module is completely different, the CELAR IS needs to be aware of it, in order to enrich its own calculations.

3.2.3 Cloud Cost and Performance Analytics

The process of extracting meaningful insights from available data and their visualizations is not only time-consuming but also requires experience that may lack from the average user. Visualizations of the available data are one of the things that CELAR IS needs to provide. In order to further ease users' past deployment evaluation process, increase productivity, and aid pinpoint of interesting patterns and events CELAR IS will also generate analytics reports. Cloud Usage Analytics aim to convert the available data into easily identifiable patterns and trends that help the user review and evaluate her application lifecycle. As noted by Hosseini et al. [Hosseini], since analysis and cost management tools are *only as good as the data they collect*, the crucial information that CELAR IS needs is the application-specific information such as:

- Application version (i.e. application description);
- Deployments (status, start/end time);
- Resources instantiated for a specific deployment and the time they were 'alive';
- Resizing actions occurred (decisions);
- Monitoring data for every resource instantiated during a deployment.

Furthermore, having access to data that other CELAR modules collect, such as cost information, profiling information and smart deployment information provide added value to what will be presented to the end user.

Using the available monitoring data, for each deployed application, CELAR IS will calculate analytics for the duration of the deployment or a specific period of interest to the user. The latter gives the ability to explore a specific small period, and can be used to provide event-based analytics. For example, in a system that supports elasticity we can have as input the occurrence time of the resizing action and analyze the data for the period before and after that action.

Analytics can be, in many cases, costly procedures, requiring intensive calculations. In some of the requested analysis, there are situations where one might require to store the result, in order to reuse it for later calculations. Consider the following use case: *a user requests for a synopsis of a specific application deployment*. The backend system will calculate analytics for every used virtual instance and 'merge' them according to the application topology before answering the user request. At a later point, the user wants to drill-down through the application topology, and view a synopsis for every component. The main analysis procedure is identical, since the backend system will perform almost the same calculations. Without any memory regarding the previous analysis, the following issues arise: overhead on the data collection mechanism and on the analysis mechanism by repeating the same process as before. To tackle this issue, we plan to implement an in-memory cache to store selected results (depending on the analysis procedure) for later use.

The CELAR IS will provide simple mathematical functions such as estimation of minimum, maximum and average values of a metric for a period. These calculations, though trivial, can be used as the basis to provide more complex analysis functionality. As an example, we can extract the trend of a metric, meaning how the metric value is changing during the deployment. Additionally, the CELAR IS can calculate the time or the percentage of time that a metric was retaining a value within a predefined range (e.g. for how much time, CPU usage of a resource was over 80%).

The CELAR modules provide information beyond simple resource usage metrics. For example, the Multi-Level Metric Evaluation Module, provides cost information for each utilized resource for the duration of a deployment. CELAR IS will use this information, along with the available resource metrics, to provide further insights on the user's application. For example, CELAR IS will be able to pinpoint which resources are under-utilized, while, at the same time, their cost is skyrocketing. CELAR also provides the application topology. Using this information CELAR IS will be able to provide targeted information about a specific group of resources and output complex analytics reports. The currently available commercial services, presented in Section 2.3, do not take into account the application topology. Application topology provides information on how we can group the distinctive instances and expose high-level information. For example, using the application topology information in our analysis we can provide the following:

- Using the available monitoring metrics and the topology of the user application, the CELAR IS can expose the overall application's performance and the per-component performance. This information can be useful to identify bottlenecks or possible bottlenecks.
- Combining application topology with resource cost and performance metrics we can monitor the cost of each application component in real-time and the total cost of the application as a whole.

3.2.4 Application Data Comparisons

A common practice, in a cloud application deployment, is to go over numerous iterations, producing different applications versions, until a balance between the optimal performance and cost is achieved. Easing this process, by providing application and deployment comparisons, is the main functionality that CELAR IS plan to offer. Consider the following intuitive use case:

Through the CELAR IS visualization and analytics, an Application User observes that the allocated memory is not utilized by her application. To reduce memory costs she creates a new application version that has less allocated memory but introduces elasticity (increase memory) policies that allow the use of more memory when needed. Using CELAR IS, again, she can then compare the two versions and select the most effective, according to her requirements (e.g. maximize performance and minimize cost). Using a similar process she can then identify the optimal elasticity triggering values to use at the introduced elasticity policy. For a user who wants to re-design or slightly change her application configuration, collecting and analyzing metrics for each created version is the first step of the process. Comparing the collected data is the second step before she decides which of the versions she created fits better to her needs. For a system to provide such functionality two things are needed: (i) access to historical monitoring data and analytics; and (ii) a system or a model that indicates which of this data belongs to a specific deployment/version. The latter is missing from the most commercial services since they do not connect to an application or deployment management system. These services consider that every provisioned resource belongs to the same deployment and thus they cannot provide comparisons.

CELAR IS already has access to historical monitoring data and application data indicating which versions belong to the same application. Using this information CELAR IS will provide comparisons between different application versions and deployments' data. These comparisons can provide insights, to the application developer, such as which version topology has the best performance/cost ratio, how a different input/request pattern affects the performance of the application, and how a specific elasticity policy has affected the application performance and/or cost. Given that, the functionality of calculating Cloud Usage Analytics (Section 3.2.3) is already provided it eases the comparison process. After the Application User requests a comparison over two different versions the CELAR IS initiates an analytics procedure as described before extracting in this way a summary of each version cost and performance. The extracted information will be presented to the Application User. To satisfy expert users, the CELAR IS will enable the user to build custom comparisons over specific metrics or analytics. This can be done by plotting in the same chart the data that the Application User requests. Referring to the previously presented example, the Application User may request to *plot and inspect the differences of memory utilization for component_A (in version_1) and component_A (in version_2)*.

Taking the comparison feature out of the CELAR project scope, if a user deploys the same application topology on several providers the feedback information from these deployments can be used to provide a comparison between multiple providers for a specific application. Depending on the provider offered services (i.e. elasticity control), these comparisons can provide insights that are more useful to the user, helping her to refine her application topology and select the one that better fits her needs.

3.2.5 CELAR Information System High-Level Architecture

With all the above in mind, our goal is to provide a layered, modular and extensible system. To facilitate this, we designed a system that consists of two major components, the *Information System Service* and the *Information System Frontend*. The Information System Service is responsible for: i) retrieving the needed data from the underlying CELAR modules; ii) combining and analyzing the retrieved data; and iii) exporting the enriched data to the user via a REST API. The Information System Frontend is the presentation layer of our system that: i) helps the user interact with the Information System Service (through the exported IS API); ii) presents the information in a unified and descriptive manner, giving the user control over her data; and iii) is used by c-Eclipse through the c-Eclipse Information Tool, in order to provide the user with a unified environment to describe, manage and monitor her application. The described architecture is depicted in Figure 3 and both of the components are further described in the next section.

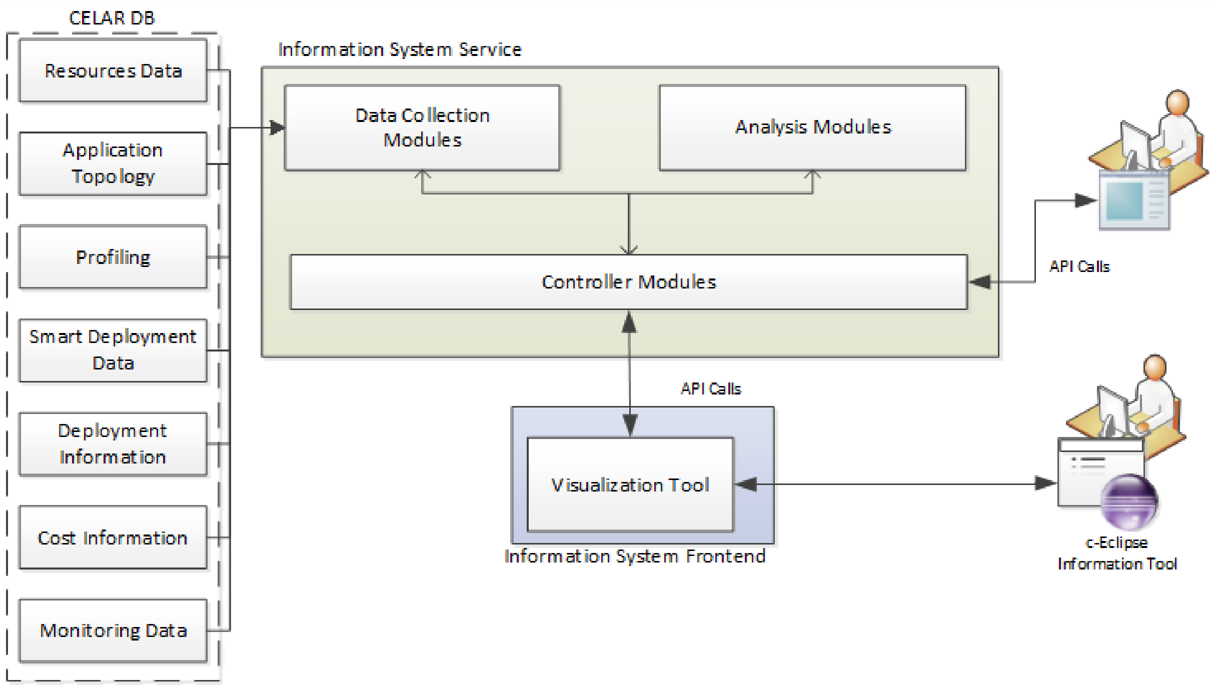


Figure 3: CELAR Information System Abstract Architecture

The "**Data Collection Modules**" are the part of the system that connects, using the exposed REST API calls, to any underlying data source (CELAR modules) and obtains the desirable information. The data is transformed, combined and/or enriched with additional information, as requested by the user, forwarded either back to the requester through the "**Controller Modules**" or to the "**Analysis Modules**", for further processing. The analysis engine is responsible for projecting analytic procedures, discussed in the previous sections (see Cloud Usage Analytics). The Visualization Tool provides the graphical interface for user to interact.

The resulting system is designed to be agnostic to the underlying services. To support this we decided to build CELAR IS as a service over the existing modules(s). The Data Collection Modules (Section 3.3.3) provide the required abstractions to avoid strictly bounding CELAR IS to any CELAR module specifically. For the CELAR IS, as a part of the CELAR ecosystem, having access to the appropriate data input from other CELAR modules running on the system, is advantageous in order to provide the aforementioned functionality. However, given the defined abstractions, if a different monitoring system provides the needed hooks for CELAR IS to get the required information, the migration to that system will be seamless.

3.3 Implementation

In the following section, we describe the implementation of the CELAR Information System. We provide a detailed overview of each of the implemented modules as well as the distinctive units that they are comprised of. The various CELAR IS modules and the interaction between them are depicted in the figure below. In the following subsection we discuss each of these modules in more detail.

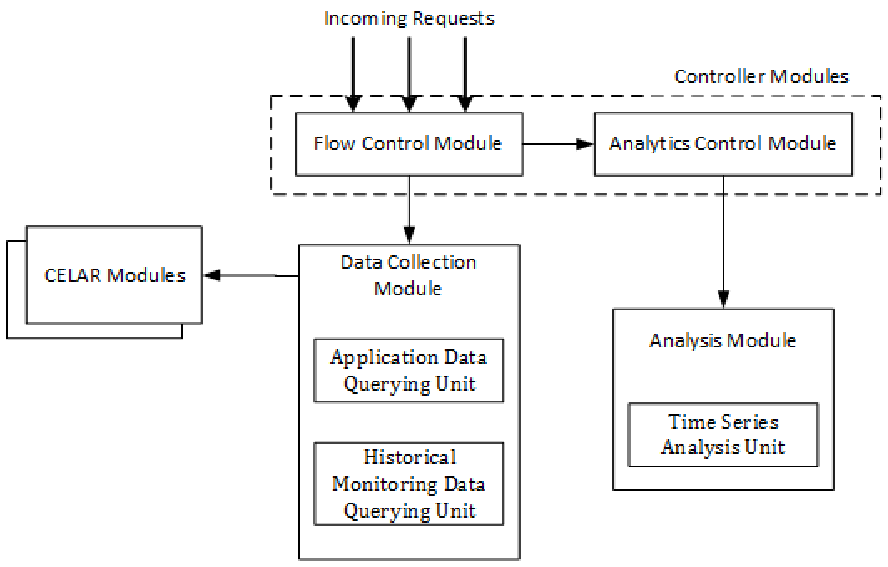


Figure 4: Inner Modules of CELAR IS

3.3.1 Flow Control

The flow control module handles the interactions between the various modules of the system and is responsible for exposing a REST API for the IS service clients to communicate. Depending on the requested procedure the flow control module will start a sequence of iterations. As incoming requests are received, the Flow Control Module, breaks them down, to self-contained routines, and forwards each routine to the sibling modules. At the end of each sequence, the flow control module constructs the REST API response and sends it back to the requester. As an example, assume that a user requests the calculation of Cloud Usage Analysis Report for a single deployment. The flow control module needs to invoke the Data Collection Module to gather information about the specific deployment, the version that this deployment belongs to and the resources that have been used during the deployment. After getting the response from these procedures, it has to invoke the Analytics Control Module to perform the analysis. The flow control module is the 'heart' of the CELAR Information System and is able to read, store and distribute any user configurations to the rest of the system.

As previously mentioned, the flow control module exposes a REST API to make the functionality of the system available to the user. The REST API is implemented utilizing the Jersey REST framework [Jersey]. Using the REST API, any interested party can have access to user application and deployment data, as well as to historical monitoring data. Interested parties can also request the calculation of analytics over that past data. In the context of CELAR, both a CELAR-IS Web Interface and a c-Eclipse Information Tool will be realized.

3.3.2 Analytics Control

The analytics control module handles every analytics report requested by the Application User. In v1.0, we consider the analytics control module as a thread pool which instantiates, configures and monitors an analytics process. The analytics control handles two types of procedures, (i) simple analytics and (ii) complex analytics. The indicator simple/complex reflects to the number of instantiations and iterations the controller has put through. An example of a simple analytics procedure is, *calculate the trend of data in a particular time interval for VM instance X*. Given that the needed data is already acquired from analytics control, the result can be calculated through a single-step process. A complex analysis procedure could be an analysis involving an applications' topology. For example, the Application User requires to *calculate analytics for the component named 'Application Server', which consists of 5 distinctive but identical VM instances*. The analytics control module will initiate an analytics procedure for each instance data and after receiving the separate results, it will synopsize them under a unified analytics report.

As described in the Section 3.2.3 a cache will be implemented to help the calculations of the cloud usage analytics. This cache unit will be encapsulated in the analytics control module. Using the cache functionality, the workflow changes slightly. The analytics control module first checks the cache for the requested result. In case of a cache hit, meaning the result was previously calculated, it retrieves the information and immediately continues to the next procedure. Only in the case of a cache miss, it will fetch monitoring data for each resource from the data collection module and initiate an analysis procedure. In the same way after the calculation of the result the analytics control module will add it in the cache.

3.3.3 Data Collection Module

The sub-modules of the Data Collection Module are responsible for collecting data from the appropriate non-IS modules, and for transforming the collected data, to (java) objects that the rest of the IS modules understand. This extra step of transforming the data keeps the rest of the system decoupled from the various data endpoints and helps us build an extensible, in terms of interoperability, system.

The implementation of the data collection module is rather simple. It consists of: (i) a set of interfaces in order to provide the needed abstraction and (ii) the implementations of those interfaces according to CELAR Modules exposed API(s). The various interface implementations are using a REST Client, which connects and obtains the requested data from the appropriate module(s). For the implementation of the REST Client, we use the Apache HTTP client [HttpClient] enhancing its functionality to fulfill our needs. Figure 5 depicts the various interfaces and their implementations. It is obvious that most of the interfaces implementations utilize the CELAR Manager REST API since the CELAR DataBase is the central database for the CELAR System

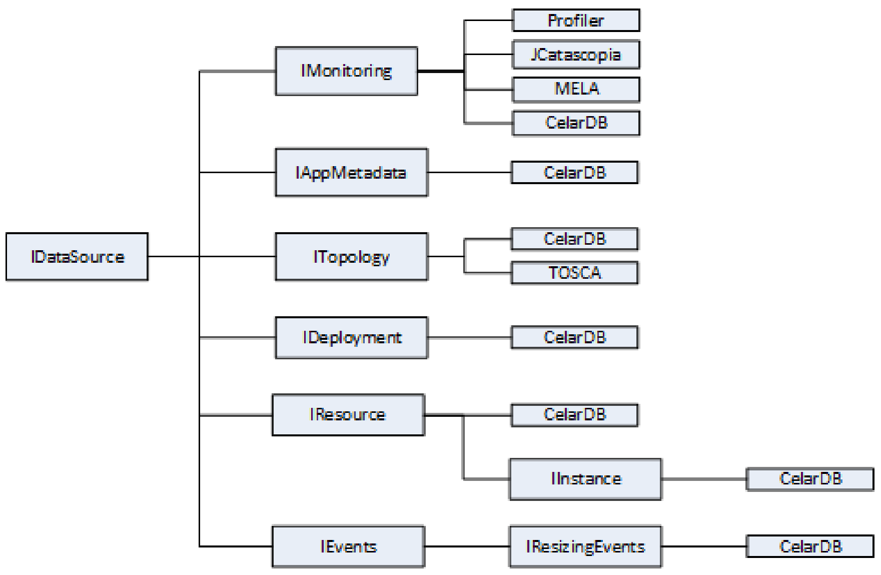


Figure 5: Data Collection Module Interfaces

For the REST Client to operate is essential to know the endpoint of the REST API Service. To do so we have implemented a set of configuration files, storing this information. This configuration information may refer to the endpoint location, i.e. the IP and the port at which the service is accessible or the path under which the REST API calls relay (e.g. /rest/apiCall_1). These configuration files will be constructed or given as input during the installation process of the CELAR IS. In the case of CELAR, the only endpoint of interest is the CELAR Manager, which should be defined before CELAR IS installation.

3.3.3.1 Query Controllers

To facilitate the need of querying the existing data we have implemented two query controllers, the application data query control and the historical monitoring data control. As we described in the previous section (Section 3.3.3) the CELAR IS will need in some cases to conduct two-step queries and enhance the information received using CELAR Manager API calls. To do so, these controllers will break down the initial query issued to the CELAR IS by the user, to the appropriate data retrieving API calls and then they will combine the returned results and forward them to the requester. Both of the controllers are specific to the underlying data sources, as is most of the data collection module, which in our case is the CELAR System.

Application Data Querying Unit

An example of "querying the application" has already been presented in the previous section and it reflects most of the functionality of this component. According to the users' requirements and the specification of the CELAR Manager API the CELAR IS will provide the user with the ability to search for applications or application versions that she owns and deployments that she launched using the following parameters:

- Time: specify a time period (start time, end time) the CELAR IS will search for submission or deployment events. When the user searches for application data, time refers to submission time. On the other hand, when the user searches for deployment data the time refers to the deployment starting time.
- Application Descriptive Information: every application field has some free text descriptive values namely description or name. The user can express a query regarding the value of this field.
- Application Structure: according to the application topology [D3.2] stored in the CELAR DataBase every application consists of a combination of modules and components, each component runs on a provided resource. The user can search for matching applications/versions using the properties of the comprising parts. For example, a user can query by 'module name' or by 'resource type' that a component will run on.

Note that both latter parameters are applicable only when the user searches for application data.

A functionality of the Application Data Querying Unit is to conduct two-step queries when it is possible. A scenario where the Application User requests a search for the keyword 'appServer' was presented in Section 3.2.1. In that case, the system can provide information that cannot be obtained with a single REST API call to the CELAR Manager by silently conducting a two-step query. The Application Data Querying unit will search CELAR DataBase using the keyword both as an application name parameter and as a module name parameter. After retrieving the results from CELAR Manager API it will join them, removing the duplicates and return the response to the user.

Historical Monitoring Data Querying Unit

To provide a visual exploration of past (historical) monitoring data, we are implementing a "Historical Monitoring Data Querying Unit". This unit in its simplest form will just formulate a call to CELAR Manager Rest API requesting for the values of a metric in a specific period for a resource. The analysis module (see Section 3.3.5) uses the Historical Monitoring Data Querying Unit to calculate analytics for only a period of a deployment.

3.3.4 Analysis Module

The analysis module is invoked occasionally from the analytics controller to conduct analytics procedures over the provided data. Regarding the input parameters, the analysis module will make the suitable calculations to transform, the data into a meaningful, for the user, output.

Our implementation of the Analysis Module includes a statistical analysis engine. We have separated the analysis control and implementation, having a more general analysis control module, as previously introduced, and let the analysis module care about implementation of the analytics procedures. The analysis module utilizes the Apache Commons Math library [Commons Math] to apply trivial mathematical functions over the existing data. So far, the supported functions are MIN, MAX, MEDIAN, MEAN, STANDARD DEVIATION and COUNT values. Wherever possible, we try to improve these calculations and reduce resource (i.e. memory) consumption. For example, calculating the mean value needs to sum up the existing data and divide it with the count. In large datasets, this summing is inefficient. To tackle this we have implemented a function that calculates the mean, in a sliding window manner, using previously calculated average values and the next value to be added.

Time Series Analysis Unit

The Time Series Analysis Unit is a library of statistical functions over time series data. For the 1st version of CELAR IS we have implemented a trend identification function, using a simple moving average function.

The simple moving average (SMA) can be calculated by defining a period window and calculating the average for the value in that window. As the window moves through the spectrum of data we are calculating the next 'average' point. Depending, on the size of the periods and the amount of total data the moving average can result to smooth irregular values. The trend is defined as the 'long term' movement in a time series without irregular effects, and is a reflection of the underlying values. Thus, using the moving average we can identify the existence or not of a trend. The methodology to identify the trend using the moving average windows is called 'moving average smoothing' [Sánchez]. To correctly identify the trend using only the moving average, we need one extra step to cross validate our assumption. Since trend depends on irregularities and the identification of irregularities depends on the size of period, we need to calculate the SMA for various periods. If the majority of those SMAs are indicating a trend, we are certain that the trend exists. In addition to that, having the moving average calculated for different periods can help understand better the sort and the long trend of our data.

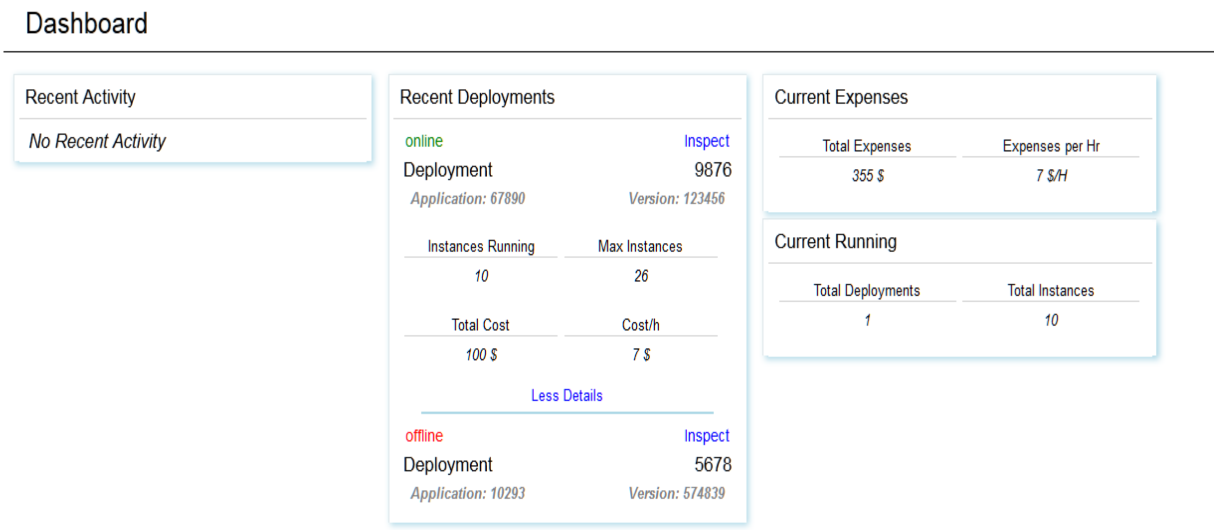
3.3.5 Information System Visualization Tool

The CELAR IS Visualization Tool, as it pointed before, is the users' graphical endpoint to interact with the CELAR IS and access the available data. The CELAR IS

Visualization Tool is implemented as a web application using Java and utilizes technologies such as HTML5, CSS3 and JavaScript. For the presentation of charts we are using the visualization API provided by Google [Charts]. According to the implemented functionality of the CELAR IS the user will have access to the following features.

Dashboard page

In dashboard page, the user can see a quick overview about her recent activity, deployments that she has launched or applications that she has submitted. Furthermore, she will have access to near-real time data for all running deployments. Clicking on an application version or deployment the user will be redirected to the appropriate page to access detailed information about it. A widget displaying cost information is also available to inform the user about her current cost expenses and the overall expenses until now. Figure 6 depicts the aforementioned functionality through a screenshot of the CELAR IS dashboard page.



Dashboard

Recent Activity

No Recent Activity

Recent Deployments

online Inspect

Deployment 9876

Application: 67890 *Version: 123456*

Instances Running	Max Instances
10	*26*

Total Cost	Cost/h
100 $	*7 $*

Less Details

offline Inspect

Deployment 5678

Application: 10293 *Version: 574839*

Current Expenses

Total Expenses	Expenses per Hr
355 $	*7 $/H*

Current Running

Total Deployments	Total Instances
1	*10*

Figure 6: CELAR IS Dashboard page

Search page

The search as its name implies, is the page where users can search according to a set of filters for application versions or deployment data. The search results will display a summary of the application versions or deployments and they will also provide a link to access more detailed information.

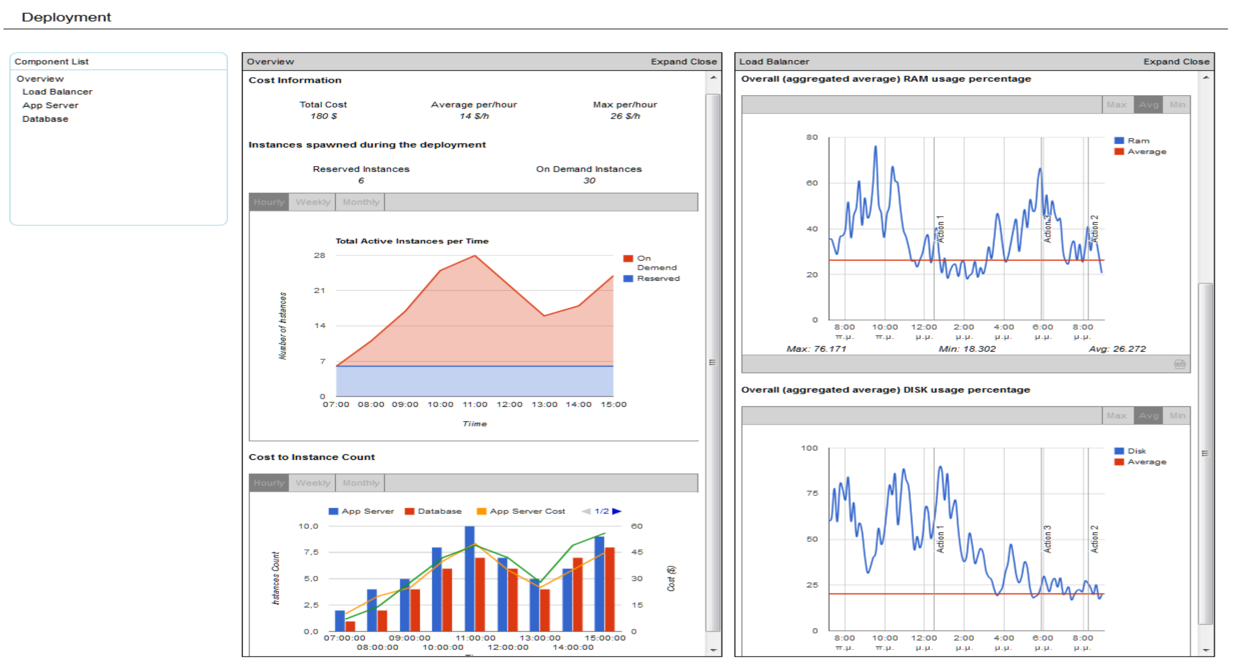
Version page

In the versions page the user will have the ability to explore through the application version data, such as the version name and the submission date. The user will also have the ability to view a graphical representation of the application topology regarding the specific version. If available, data from the smart deployment module or the profiler module will be displayed here. In the subsection 'deployments', the user can access all the deployments launched using this versions' topology.

Deployment page

The deployment page displays information about current or past deployments. Depending on whether a deployment is running or not, the user will have access to either real time information, from Monitoring System and Multi-Level evaluation

Module, or analytics based on historical data. A side menu will provide a hierarchical / tree view according to the application topology. The user will have the ability to view analytics for the whole application (as a summary) or drill down to specific components or instances. Figures 7 and 8 show the use of CELAR IS when presenting analytics for the various components of a past deployment. In the left side of both figures, we can see a navigation panel that the user can use to access analytics for the various application components based on application topology. In a later version, more 'navigation' options are to be added, such as a list of all the resizing actions enforced for each component. Selecting one of those the user can inspect information for the time before and after the resizing action.



Deployment

Component List
- Overview
  - Load Balancer
  - App Server
  - Database

Overview — Expand Close

Cost Information

Total Cost	Average per/hour	Max per/hour
180 $	14 $/h	26 $/h

Instances spawned during the deployment

Reserved Instances	On Demand Instances
6	30

Cost to Instance Count

Load Balancer — Expand Close

Overall (aggregated average) RAM usage percentage

Max: 76.171 Min: 18.302 Avg: 26.272

Overall (aggregated average) DISK usage percentage

Figure 7: Deployment Analytics

In Figure 7, the left content box named 'overview', displays information about the deployment cost and the on demand resources. More specifically at the first chart we can see how many instances (VM) where dynamically allocated and when, while at the second we can see a combination of the allocated VMs for each application component and their cost over the deployment's duration. The right content box named 'Load Balancer' shows information about a specific application component, here we can see the ram / disk usage percentage over the time. Since application level metrics are also available, in the future we plan to display them here also. Figure 8 depicts the ability of the CELAR IS to enlarge any of the above content boxes (see Figure 7) in order to provide a more detail view to the user.

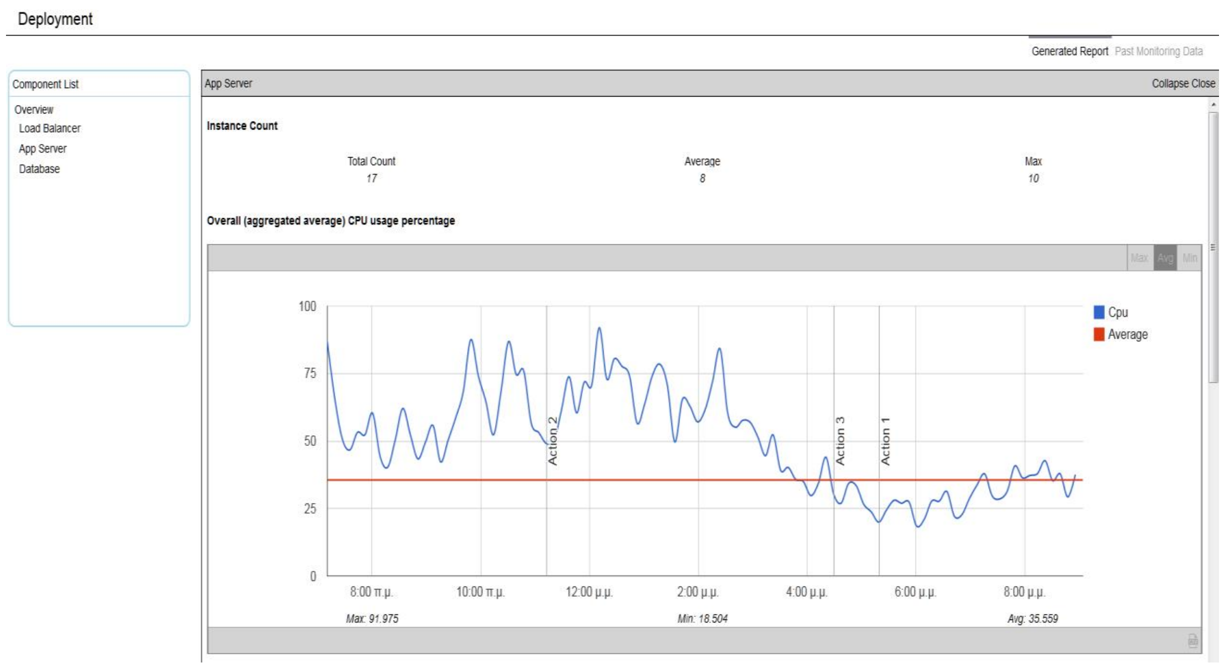


Figure 8: Deployment Analytics - Enlarged view

Comparisons page

The comparisons page will cover the functionality of comparing different deployments. The user has to select which versions she wants to compare and the system will automatically generate a report comparing the calculated analytics from each of them. Moreover, the user will have the ability to construct on demand charts for the collected metrics in order to compare the versions performance in more detail.

3.3.6 Code and Artifact Repository

The source code for the first version (v1.0) of CELAR Information System can be found at GitHub under the following URL:

https://github.com/CELAR/cloud-is

3.4 Integration with CELAR Modules

In this section we document how the CELAR Information System is integrated to the CELAR eco-system and how it interacts with other CELAR Modules. The Information System, as part of the Cloud Information and Performance Monitor layer, is tightly coupled to the Monitoring System and the Multi-Level Metrics Module. However, the Information System also interacts with the CELAR Manager, as well as with the c-Eclipse Information Tool.

Figure 9 depicts an updated version of the dataflow diagram, initially presented in D4.1 (Section 7), which showcases how the components of the Cloud Information and Performance Monitor layer (green color) interact and exchange data with other CELAR modules. For the sake of readability, we have omitted the CELAR Server container from this figure (Figure 9).

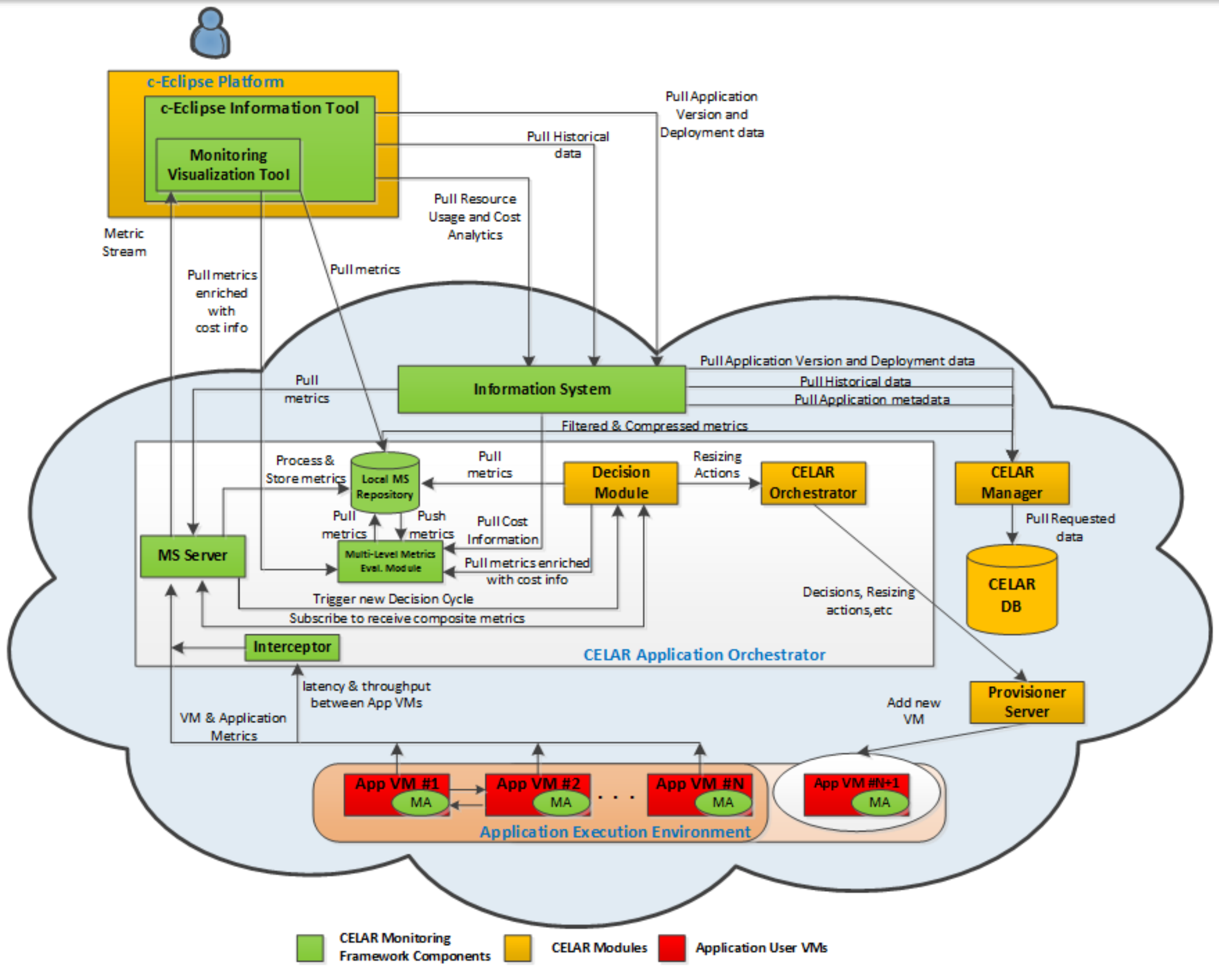

c-Eclipse Platform
c-Eclipse Information Tool
Monitoring Visualization Tool
Metric Stream
Pull metrics enriched with cost info
Pull metrics
Pull Resource Usage and Cost Analytics
Pull Historical data
Pull Application Version and Deployment data
Pull metrics
Information System
Pull Application Version and Deployment data
Pull Historical data
Pull Application metadata
Filtered & Compressed metrics
Process & Store metrics
Local MS Repository
Pull metrics
Decision Module
Resizing Actions
CELAR Orchestrator
CELAR Manager
Pull Requested data
CELAR DB
Pull metrics
Push metrics
Pull Cost Information
MS Server
Multi-Level Metrics Eval. Module
Pull metrics enriched with cost info
Trigger new Decision Cycle
Subscribe to receive composite metrics
Decisions, Resizing actions,etc
Interceptor
CELAR Application Orchestrator
latency & throughput between App VMs
VM & Application Metrics
Add new VM
Provisioner Server
App VM #1
MA
App VM #2
MA
App VM #N
MA
App VM #N+1
MA
Application Execution Environment
🟩 CELAR Monitoring Framework Components
🟨 CELAR Modules
🟥 Application User VMs

Figure 9: Cloud Information and Performance Layer Workflow Diagram

When an Application User, via c-Eclipse, describes and submits a cloud application for deployment to a cloud platform supported by CELAR, the CELAR Manager undertakes the process of instantiating a new Application Orchestration VM with the, per application, CELAR modules (Monitoring System, Multi-Level Metrics Evaluation Module, Decision Module) which are in charge of monitoring and decision-making. This process was described in D4.1 [D4.1] without including details regarding the Information System. In turn, after instantiating a new Application Orchestration VM, the CELAR Manager must provide the Information System with the appropriate metadata concerning the application in order for the IS to provide the Application User with resource usage analytics and query capabilities. The Information System, runs as a service, located on the CELAR Server (alongside the Slipstream Server and CELAR Manager) and therefore, instantiation is not required after each application deployment. The application metadata required to be provided by the CELAR Manager, for v1.0, include: (i) the endpoints for the REST APIs of the Monitoring System and Multi-Metrics Evaluation Module located on the Application Orchestration VM; (ii) the application's ID and version ID to query the CELAR Manager in order to access the CELAR DataBase for application specific information (e.g. topology). To access both monitoring metrics, as well as, cost-enriched metrics, the Information System periodically polls the pull-mechanism REST APIs, of the Monitoring System and the Multi-Level Metrics Evaluation Module respectively, and in a similar manner, to obtain updated application information, it polls the CELAR Manager.

Application Users, via the c-Eclipse Information Tool, have graphical access to the following information provided by the Information System: (i) application version and deployment information stored in CELAR DataBase; (ii) historical and near real-time monitoring metrics, as well as cost-enriched metrics; (iii) cloud resource usage analytics. The application version and deployment information is stored in the CELAR DataBase, therefore the Information System is required to fetch, process and analyze this information by querying the CELAR Manager, via its REST API. Obtaining near real-time monitoring information for the current deployment, as well as, historic information is a slightly different process for the Information System, since it must contact the Monitoring System and Multi-Level Metric Evaluation Module to have access to such information.

4 CELAR Monitoring System

This section documents the second (v2.0), and final version, of the CELAR Monitoring System as the respected WP4 task (T4.1: Monitoring Tool) ends shortly. In this section we provide a brief overview of the CELAR Monitoring System, present the new features that are incorporated in v2.0, provide a documentation report and conclude with an evaluation and verification analysis of its requirements.

4.1 Overview

Figure 10 depicts an abstract view of the JCatascopia Monitoring System architecture. JCatascopia is the implemented prototype of the CELAR Monitoring System. Briefly, the architecture follows an agent-based approach. This approach provides a scalable, real-time, cloud monitoring system that can be utilized to collect monitoring metrics from multiple layers of the underlying infrastructure, as well as performance metrics from the deployed cloud services. JCatascopia is platform independent and during the metric collection process it takes into consideration the rapid changes that occur due to the enforcement of elasticity actions on the application execution environment and the cloud infrastructure.

The main components[6] which comprise the JCatascopia Monitoring System are defined as follows:

- **Monitoring Agents**: light-weight monitoring instances deployable on any cloud elements to be monitored, such as physical nodes or virtual instances. Monitoring Agents are the entities responsible for managing the metric collection process on the respective cloud element. Additional functionality of Monitoring Agents includes metric aggregation and metric distribution via a pub/sub metric delivery mechanism.
- **Monitoring Probes**: the actual metric collectors managed by Monitoring Agents. Probes are responsible for gathering system-level and application performance metrics. Metrics are forwarded to the respected Monitoring Agent in either a push or pull manner. Additionally Probes offer a filtering mechanism, which allows for metric values to be filtered, *in place*, rather than distributing metrics and later discarding them at the Monitoring Server level. Users can take advantage of the JCatascopia Java Probe API to implement their own custom Monitoring Probes and then, deploy them dynamically, at runtime, to Monitoring Agents without the need to restart the monitoring process.
- **Monitoring Servers**: entities which receive monitoring metrics from their respected Monitoring Agents. The acquired metrics are then processed and stored in the monitoring database. Monitoring Servers additionally, handle user metric and configuration requests. A hierarchy of Monitoring Servers can be utilized to enhance greater scalability.
- Finally, JCatascopia provides a **REST Web Service** that allows for external entities such as application users or the Decision Module to access monitoring information stored in the **Monitoring Database**.

[6] For a detailed analysis of the components comprising JCatascopia, readers are advised to look at D4.1 Sections 4 & 5

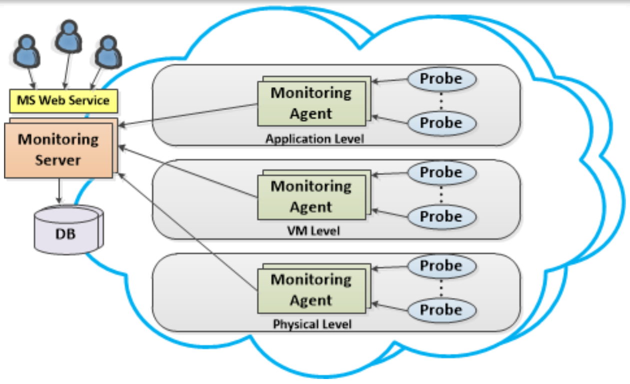


Figure 10: CELAR Monitoring System Abstract Architecture View

4.2 Requirements

The requirements listed in D4.1 (Section 3.3 and 3.4) were updated to better represent the scope and target goals of the CELAR Monitoring System and were documented in D1.2 (Requirements R4.1-R4.12). These requirements are considered as the current requirements for the CELAR MS.

4.3 New Features and Improvements

The following subsections document the latest features and improvements made available in the current version of the CELAR Monitoring System.

4.3.1 JCatascopia Pub/Sub Message Communication Protocol

An initial version (v0.9) of the JCatascopia Pub/Sub Message Communication Protocol was introduced in D4.1 (Section 5.5.1: JCatascopia Agent Message Patterns) as the protocol used to establish a connection between JCatascopia Monitoring Agents and their respected Monitoring Server. The difficulty with establishing a connection between the two entities is that in contrast to static cloud service deployments, in an elastic application execution environment, instances, and consequently Monitoring Agents, (dis-)appear dynamically due to elasticity actions. The initial version was amended to improve latency, connectivity issues and status/error messages were added to the protocol. Version v1.0 was released and described in detail in [Trihinas 2014a].

Briefly, to overcome this challenge, we vary the classic pub/sub message communication pattern as follows: (i) Monitoring Servers (depicted in Figure 11), which are the static part of the model (not affected by elasticity actions) bind to a network interface, awaiting for incoming requests; and instead (ii) Monitoring Agents, which are the metric publishers, initiate the subscription process by contacting first the Monitoring Server to inform it of their existence with a SUBSCRIBE message. Afterwards, Monitoring Agents send a METADATA message to the Monitoring Server to inform it of the metrics they are responsible to collect, their IP address, agent ID and etc. Finally, after the subscription process is over, Monitoring Agents can start publishing metrics to the established metric stream.

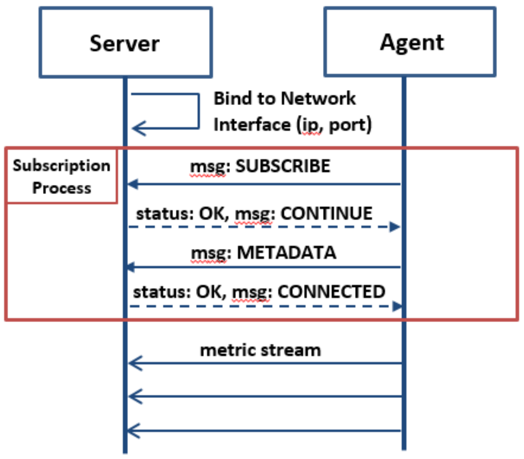

Server
Agent
Bind to Network Interface (ip, port)
Subscription Process
msg: SUBSCRIBE
status: OK, msg: CONTINUE
msg: METADATA
status: OK, msg: CONNECTED
metric stream

Figure 11: JCatascopia Pub/Sub Subscription Process

In the current version (v2.0) we take into consideration realistic special-case scenarios which may be considered as exceptions to the smooth functioning of the monitoring process, nonetheless, these situations occur from time to time.

The first scenario that we consider is the presence of network connectivity problems between a Monitoring Agent and its respective Monitoring Server. In the previous versions, if a Monitoring Agent was unavailable for a specified period of time[7], then the connection would be dropped. Any incoming messages or metric streams from the Monitoring Agent would be marked as malicious and ignored with the Monitoring Agent not having any knowledge that the metric values sent are discarded. The only way to re-establish the connection was to terminate and re-instantiate the Monitoring Agent service. However, in the current version (v2.0), if a Monitoring Agent (Figure 12) sends metric values, or issues a request, to the Monitoring Server after the connection has been dropped, the Monitoring Server will reply with a RECONNECT message to the Agent issuing the request, allowing for the Monitoring Agent to re-establish the connection without the need to restart the monitoring process.

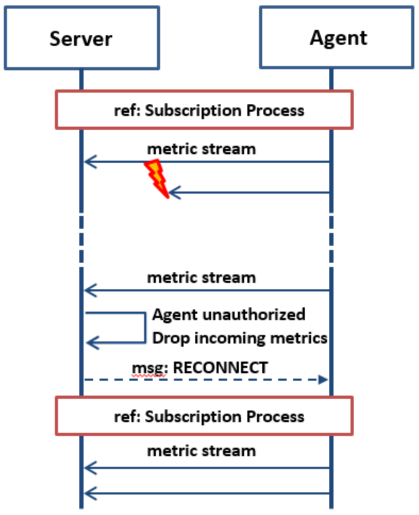


Figure 12: JCatascopia Pub/Sub Reconnect Process

[7] For more information regarding dynamic Monitoring Agent removal and heartbeat monitoring, readers are advised to look at D4.1

The second scenario that we take into consideration in v2.0, is the uncertainties imposed due to re-contextualization such as in the case where the IP address of a VM is changed at runtime. This scenario is not extreme since a number of cloud providers, such as GRNET Okeanos public cloud [Okeanos], offer the ability to attach/remove network interfaces to/from virtual instances on the fly without restarting the instance. To address this issue, JCatascopia Monitoring Agents periodically update their metadata (i.e. IP address, available metrics, etc.) and then send a METADATA message to the Monitoring Server as depicted in Figure 13. The limitation of this approach is that a request (e.g. a metric pull request) issued from a Monitoring Server to an Agent, residing on a VM that just changed IP, will fail if the Agent metadata has not yet been updated. The error space can be shortened if the service issuing the migration (e.g. CELAR Manager, Decision Module) informs the Monitoring Agent of this by triggering either the `updateAgentIP()` or `setAgentIP(ipAddress)` API calls. The same approach is followed in the situation where the IP address of a Monitoring Server is changed with the addition of one more step: After an IP address update occurs, the Monitoring Server notifies its respected Agents with a METADATA message containing its new IP address.

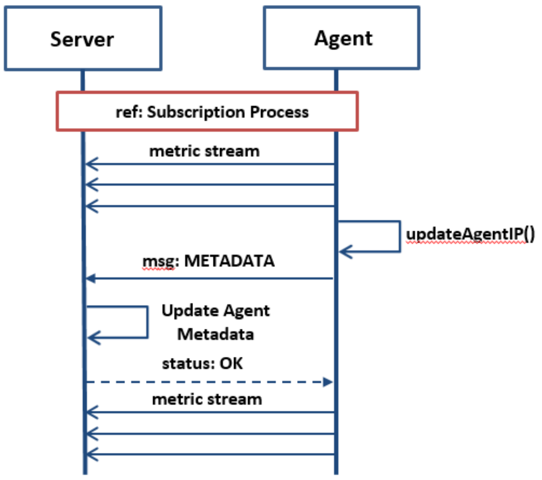


Figure 13: JCatascopia Pub/Sub Metadata Update Process

4.3.2 Monitoring Agent Probe Controller Implementation

The Probe Controller, as defined in D4.1 (Section 5.5: JCatascopia Monitoring Agent), is the internal component of the Monitoring Agent that is in charge of managing and configuring, the assigned to an Agent, Monitoring Probes. In v1.0, the Probe Controller had the ability to: (i) manage (activate, deactivate, terminate) Monitoring Probes but with the limitation that these Probes must be bundled, in the same process, with the Monitoring Agent; (ii) listen to incoming metric requests, allowing in this manner for external processes to inject monitoring metrics to the Monitoring Agent; and (iii) listen to Monitoring Probe parameter configuration requests (e.g. configure Probe sampling period).

In v2.0, the Probe Controller now has the ability to deploy external Monitoring Probes, even at runtime, thereby satisfying requirement R4.3 (Deploy Custom Monitoring Probes at Runtime). Application Users or other entities (e.g. CELAR Manager, c-Eclipse) can add Monitoring Probes at runtime by issuing to the respected Monitoring Agent an AGENT.ADD_PROBE request with parameters the *name* of the Probe and its *location*. After receiving the request, the Monitoring Agent will configure and activate the newly added Probe and will respond to the request with a status message, informing the requester if the Probe deployment was successful or not.

4.3.3 Monitoring Agent Aggregator Interface

Aggregation is a feature that was offered by Monitoring Agents in v1.0 to reduce the metric distribution and storage overhead. However, the implemented Aggregator (`StringAggregator`) was very simplistic and limited to only appending new metric values to the message, which, in turn, was withheld until an aggregation policy was satisfied (e.g. time-based, volume-based). In v2.0, JCatascopia offers application developers with the ability to either use: (i) an Aggregator available in the Aggregator library, (i.e. `MapAggregator`, distributes only the latest value per metric; `AVGAggregator`, distributes the average value of the values collected per metric) or (ii) their own Aggregator implementation by taking advantage of the JCatascopia Java Aggregator Interface. Figure 14 depicts the interface that must be implemented when creating a new Aggregator.

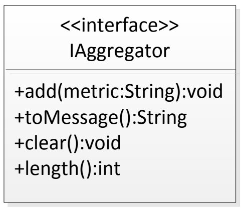


Figure 14: Monitoring Agent Aggregator Interface

4.3.4 Monitoring Agent Naming and Tagging

In v2.0, JCatascopia provides its users with the ability to *name* Monitoring Agents and to also *tag* them. Agents are identified internally by CELAR with their Agent ID (a UUID string), nonetheless, adding a name to an Agent provides users with the ability to easily identify, via the JCatascopia web interface, an instance of interest, instead of having to remember its IP address as in v1.0. However, it must be noted that if no name is provided, then the fallback utilized is to use the agent's IP address. Tags, as with agent names, provide users with the ability to group/cluster Monitoring Agents for simplicity as the number of Monitoring Agents grows very large. Users can add more than one tag to a Monitoring Agent. It must be noted that even though tags could be used to cluster Agents based on the service topology of the application they reside on, tagging does not substitute the role of the Multi-Level Metrics Evaluation Module which has specific and up-to-date knowledge of the application's topology in order to map monitoring metrics to the provided topology. Figure 15 depicts the JCatascopia web interface while monitoring one of the CELAR pilot applications (Playgen DataPlay).

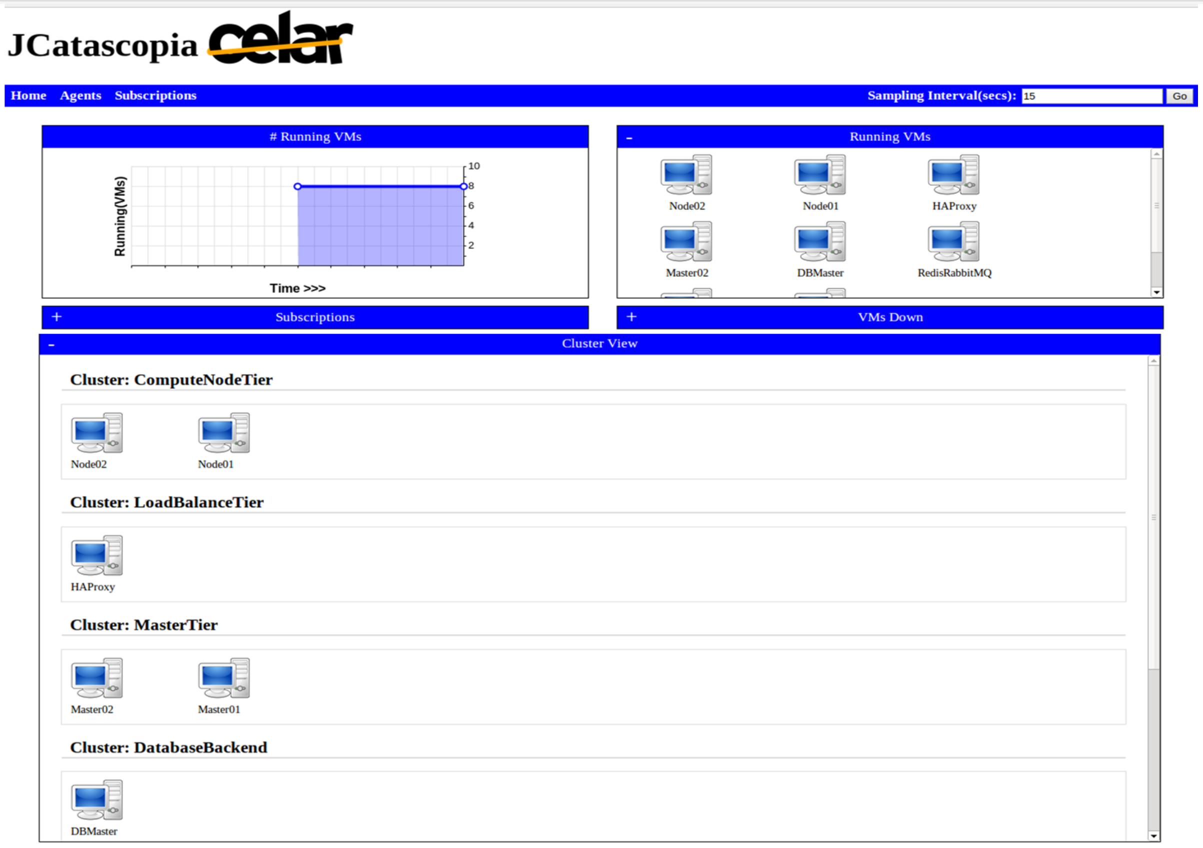
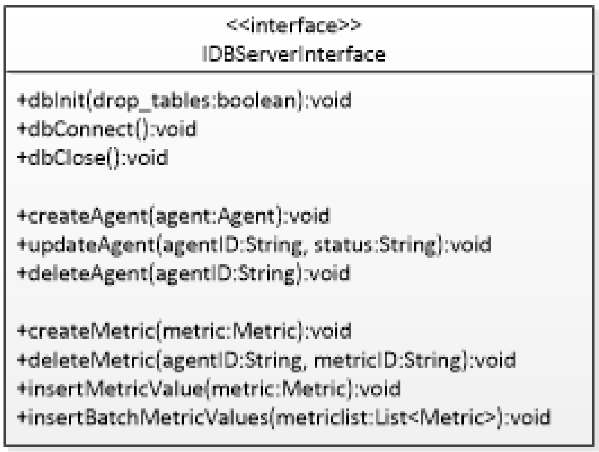


Figure 15: JCatascopia Index Page with ClusterView Tab Expanded

4.3.5 Database Interface

JCatascopia, as a modular and extensible system, offers its users with the functionality of utilizing a database solution of their own to handle metric insertion and extraction, by providing a *Database Interface Layer*, making JCatascopia flexible to which database backend is used. Specifically, users interested in implementing their own database backend must extend the `IDBServerInterface`, for metric insertion, and the `IDBWebInterface`, for metric extraction via the JCatascopia REST API. Figure 16 and Figure 17 depict the two aforementioned interfaces.



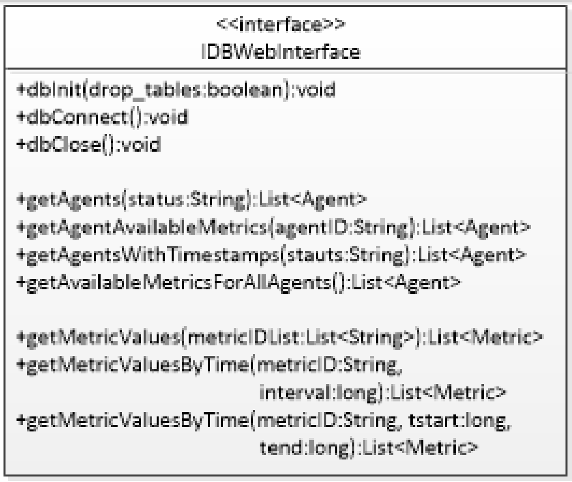
```
<<interface>>
IDBServerInterface

+dbInit(drop_tables:boolean):void
+dbConnect():void
+dbClose():void

+createAgent(agent:Agent):void
+updateAgent(agentID:String, status:String):void
+deleteAgent(agentID:String):void

+createMetric(metric:Metric):void
+deleteMetric(agentID:String, metricID:String):void
+insertMetricValue(metric:Metric):void
+insertBatchMetricValues(metriclist:List<Metric>):void
```

Figure 16: Monitoring Server Database Interface



```
<<interface>>
IDBWebInterface

+dbInit(drop_tables:boolean):void
+dbConnect():void
+dbClose():void

+getAgents(status:String):List<Agent>
+getAgentAvailableMetrics(agentID:String):List<Agent>
+getAgentsWithTimestamps(stauts:String):List<Agent>
+getAvailableMetricsForAllAgents():List<Agent>

+getMetricValues(metricIDList:List<String>):List<Metric>
+getMetricValuesByTime(metricID:String,
                       interval:long):List<Metric>
+getMetricValuesByTime(metricID:String, tstart:long,
                       tend:long):List<Metric>
```

Figure 17: JCatascopia Web API Database Interface

In v2.0, JCatascopia is bundled with two database interfaces, a MySQL [MySQL] relational database interface and a NoSQL Cassandra [Cassandra] database interface. The MySQL relational database interface was implemented to provide users and entities (e.g. Decision Module, CELAR Information System) with the ability to perform various types of complicated queries on monitoring data by allowing multiple joins on tables which cannot be facilitated by a NoSQL database. Therefore, for the scope of the project, the MySQL interface is the default database backend. However, this interface requires the support of the Cleanup Daemon (D4.1 Section 5.9: JCatascopia MS Database) to extract and process old monitoring data from the database, in order to reduce the size of the stored data and query response time. The Cassandra interface was implemented with keeping in mind the nature of monitoring data, where monitoring systems such as JCatascopia, require fast writes and, in turn, metric consumers require fast reads on relatively recent metrics. The positive aspects of selecting to create a NoSQL interface, include scalability, high-availability and the ability to add a configurable Time-To-Live expiration parameter to the inserted metrics, thus, this interface does not require the support of the Cleanup Daemon.

Figure 18 and Figure 19 depict snippets from the Monitoring Server configuration file where we define, and parameterize, as the selected database backend MySQL and Cassandra respectively.

```
##MySQL Database Properties##
db_interface=MySQL.DBHandlerWithConnPool
db_host=83.212.124.102:3306
db_user=catascopia_user
db_pass=catascopia_pass
db_database=JCatascopiaDB
```

Figure 18: MySQL Database Interface Configuration

```
##Cassandra Database Properties##
db_interface=Cassandra.DBHandler
db_host=83.212.124.102
db_user=catascopia_user
db_pass=catascopia_pass
db_database=JCatascopiaDB
```

Figure 19: Cassandra Database Interface Configuration

4.3.6 Push Metric Delivery Mechanism

Providing both a push and pull metric delivery mechanism was set as a requirement, in D4.1 and D1.2 (Requirement R4.5), for the CELAR Monitoring System. However, v1.0 of JCatascopia, only offered a pull delivery mechanism. In v2.0, JCatascopia is enhanced with a push delivery mechanism (named Noti.Me), which is implemented on top of the JCatascopia REST API, in python and it uses websockets [Websockets] to establish a bi-directional communication connection with interested entities (e.g. Multi-Metrics

Evaluation Module, Decision Module, Application Users). Specifically, interested entities are required to first, establish a connection with the mechanism, and then, create subscriptions to metric event notifications. Interesting entities will only be notified if the thresholds defined per metric are violated.

To add a monitoring metric event notification, one must provide the *metric id*, *thresholds* (as a comma-separated list) and optionally a *maximum period* (in seconds) to report a metric value even if no violation was detected. The following listing depicts an exemplary (JSON) subscription sent from a test client interested in being notified only if the average CPU utilization is under 25% or over 75%:

```
{
 "header" : "ADD", "conID" : "2847ec94a6784b85be5d3a61017b4225",
 "metricID" : "3f47fcba39244e89a114c6a0161482d1:cpuTotal",
 "action" : "NOTIFY:<25,>=75", "maxPeriod" : "45"
}
```

Listing 1: JSON Subscription to Receive Metric Event Notifications

4.3.7 JCatascopia Monitoring Probe Library

Table 2, lists the application-specific JCatascopia Monitoring Probes developed to facilitate the needs of the CELAR pilot applications. All listed Monitoring Probes were implemented in a generic manner and are not limited to the pilot applications. Thus, they can be re-used by other users and cloud services. Moreover, all developed Monitoring Probes are open-source under the Apache 2.0 License and are publically available at the JCatascopia Github repository:

https://github.com/dtrihinas/JCatascopia-Probe-Library

Probe	Metric	Type	Units	Description
HAProxy Load Balancer	activeSessions	Integer	#	Number of active connections
	requestRate	Double	req/s	Requests per second
	bytesIn	Double	bytes/s	Bytes IN per second
	bytesOut	Double	bytes/s	Bytes OUT per second
	avgResponseTime	Double	ms	Average response time in ms of all servers
	servers	Integer	#	Number of servers behind proxy
	errorRate	Integer	err/s	Errors per second
RabbitMQ[8] Message Queue Broker	memory	Integer	KB	Total amount of RAM used by this queue in KB
	consumers	Integer	#	Number of consumers this queue has
	msgs	Integer	#	Total number of messages in this queue
	dlvr_rate	Double	msgs/s	Rate at which messages are delivered
	pub_rate	Double	msgs/s	Rate at which messages are published to the queue

[8] All RabbitMQ metrics are collected per queue. For example, if the user defines in Probe configuration file multiple queues then JCatascopia will collect these metrics for each of the defined queues

Cassandra[9] Database	readLatency	Double	ms	Keyspace read latency
	writeLatency	Double	ms	Keyspace write latency
Apache Tomcat	maxThreads	Long	#	Max number of threads that tomcat can instantiate and add to thread pool
	currentThreadCount	Long	#	Number of threads created on the Apache Tomcat container
	currentThreadsBusy	Long	#	Number of busy threads on the Apache Tomcat container
	bytesReceived	Long	#	Bytes received by each request processor since last collection
	bytesSent	Long	Bytes	Bytes sent by each request processor since last collection
	requestCount	Long	Bytes	Number of requests served by each request processor since last collection
	errorCount	Long	#	Error count of each request processor since last collection
	processingTime	Long	ms	Request processing time (in milliseconds) of each global request processor
	requestThroughput	Double	req/s	Rate at which requests are processed
Playgen DataPlay	mean	Integer	#	Mean, Variance and Standard Deviation of Response Time at DataPlay Master Node layer
	standev	Integer	#	
	variation	Double	#	

Table 2: Application Specific Monitoring Probes Available in Probe Library

4.4 Documentation

4.4.1 JCatascopia Artifacts and Installation Process

All JCatascopia components and interfaces are developed in Java however components are packaged differently and contain Bash/Shell installation and service scripts for Linux.

4.4.1.1 JCatascopia Monitoring Agents

Monitoring Agents are deployed via the CELAR Provisioner to Application User VMs either while instantiating the user application for the first time or when an elasticity action requests the addition of a new virtual instance to the application execution environment. Monitoring Agents are packaged and made available in two formats: (i) as a Java runnable archive (JAR); and (ii) as a TAR archive.

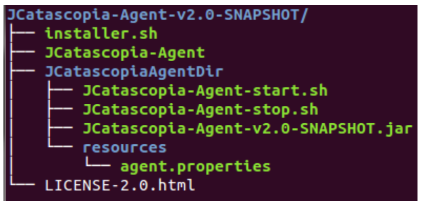
The following listing showcases how to execute the JAR packaging of the Monitoring Agent where no parameters are required to be defined upon execution, however optionally the path to a JCatascopia-Agent configuration file may be provided if this is not located in the current directory or in the default location.

```
java -jar JCatascopia-Agent-<VERSION>.jar [<PATH_TO_CONFIG_FILE>]
```

Listing 2: Executing JCatascopia-Agent via JAR Packaging

[9] All Cassandra DB metrics are collected per keyspace where the keyspaces are defined in the Probe configuration file

The default packaging used for the purposes of CELAR is the TAR archive packaging. This packaging (depicted in Figure 20) contains the Monitoring Agent as a runnable JAR, an installer, service scripts and a Monitoring Agent configuration file (`agent.properties`). To deploy the Monitoring Agent, the CELAR Provisioner downloads from the CELAR Artifact Repository the latest version of the Monitoring Agent tarball, extracts (untar) its content, and then executes the installer which copies each artifact to the required directories and sets the necessary file system permissions. Furthermore, the Monitoring Agent is registered as a Linux service (daemon) using native OS libraries (`/etc/init.d`) which allows it to run as a background system process and is automatically started/terminated when the virtual instance, in turn, is started/terminated.



```
JCatascopia-Agent-v2.0-SNAPSHOT/
├── installer.sh
├── JCatascopia-Agent
├── JCatascopiaAgentDir
│   ├── JCatascopia-Agent-start.sh
│   ├── JCatascopia-Agent-stop.sh
│   ├── JCatascopia-Agent-v2.0-SNAPSHOT.jar
│   └── resources
│       └── agent.properties
└── LICENSE-2.0.html
```

Figure 20: Monitoring Agent TAR Packaging Content

Optionally, before starting the Monitoring Agent service, the CELAR Provisioner may configure the default properties of the Monitoring Agent to meet the requirements set by either the Application User (e.g. what metrics to collect) or by the CELAR Manager (e.g. ports to use or type of Aggregator). The configuration properties of a Monitoring Agent are listed in the following table:

Property	Type	Default Value	Description
Main Settings			
logging	Required	true	When set to true, the JCatascopia Monitoring Agent will log abnormal behavior
debug_mode	Required	false	When set to true, the JCatascopia Monitoring Agent will literally print every occurring event to the console. This option should ONLY be used for testing reasons
use_server	Required	true	When set to false, the JCatascopia Monitoring Agent will function without contacting a Monitoring Server. This option should ONLY be used for testing reasons (and debug_mode=true)
name	Optional	IP Address	Monitoring Agent name
tags	Optional	NULL	Monitoring Agent tag(s)
Monitoring Probe Settings			
probes	Required	all	When set to all, ALL probes in the JCatascopia Probe Library will be activated. It is recommended for users to only activate the Probes that they require. The correct format to activate probes is: `<PROBE_1>,<PERIOD_1>;<PROBE_2>,<PERIOD_2>` e.g. `probes=CPUProbe,15;MemoryProbe,25`
probes_exclude	Optional	NULL	This option is used when the probes option is set to all (probes=all) to eliminate the need for users to have to activate each Probe when a user only wants to exclude a small number of Probes e.g. `probes=all` `probes_exclude=DiskProbe`
probes_external	Optional	NULL	This option is used when a user wants to utilize a JCatascopia Monitoring Probe that is located outside of

			the JCatascopia Probe Library. The correct format to activate probes is: `<PROBE_1>,<PATH_TO_PROBE>; <PROBE_2>,<PATH_TO_PROBE>` e.g. `probes_external=TomcatProbe, /etc/myprobes/TomcatProbe.jar`
Interface Settings			
server_ip	Required	localhost	JCatascopia Monitoring Agent uses the value(s) defined in this property to determine the IP address(es) of the Monitoring Server(s) that metrics will be distributed to
distributor_port	Required	4242	The port which JCatascopia Monitoring Agent uses to distribute metrics to Monitoring Server (should be the same as the `agent_port` of the Monitoring Server). If not required otherwise, this value should NOT be changed
distributor_interface	Required	TCP	The protocol to use to distribute metrics (either TCP or REST)
probe_controller_turnOn	Optional	true	When set to true, JCatascopia Monitoring Agent will listen to incoming requests from the Monitoring Server. It is suggested to be left turn on
probe_controller_ip	Optional	*	The network interface that the ProbeController will bind to. The default value is set to * which indicates that the Agent will bind to all network interfaces. If it must be changed then it is suggested to use the eth0 interface but users are not obligated to.
probe_controller_port	Optional	4243	The port which JCatascopia Monitoring Agent ProbeController binds to. If not required otherwise, this value should NOT be changed
Aggregator Settings			
aggregator_interval	Required	30 seconds	time-based rule to aggregate collected metrics and then distribute them to the Monitoring Server
aggregator_buffer_size	Required	2048 KB	volume-based rule to aggregate collected metrics and then distribute them to the Monitoring Server
Aggregator_interface	Required	StringAggegator	The aggregation interface (e.g. MapAggregator) to be used. Default is set to StringAggregator, the Monitoring Agent will append all collected values to the message that will be distributed.

Table 3: Monitoring Agent Configuration Properties

4.4.1.2 JCatascopia Monitoring Servers

Monitoring Servers are deployed via the CELAR Provisioner to the Application Orchestration VM while instantiating the user application deployment for the first time. Similarly with Monitoring Agents, Monitoring Servers are packaged and made available in the following two formats: (i) as a Java runnable archive (JAR); and (ii) as a TAR archive. However, Monitoring Servers are packaged as RPMs as well. The RPM packaging is the default packaging used for CELAR since all CELAR components deployed on the Application Orchestration VM follow the same packaging which uses as its Operating System CentOS [CentOS].

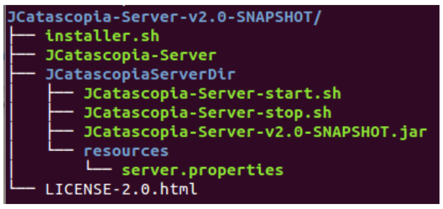
The JAR packaging follows the same logic as the JAR packaging of Monitoring Agents and the proper manner to execute the JAR packaging is depicted in the following listing:

```
java -jar JCatascopia-Server-<VERSION>.jar [<PATH_TO_CONFIG_FILE>]
```

Listing 3: Executing JCatascopia-Server via JAR Packaging

Both, the TAR archive and the RPM packaging (depicted in Figure 21) contain the Monitoring Server as a runnable JAR, an installer, service scripts and a Monitoring Server configuration file (`server.properties`). In a similar manner with Monitoring

Agents, to deploy the Monitoring Server, the CELAR Provisioner downloads from the CELAR Artifact Repository the latest version of the Monitoring Server RPM, it then executes the RPM packaging (via yum) which invokes the Monitoring Server installer, which, in turn, copies each artifact to the required directories and sets the necessary file system permissions. Furthermore, the Monitoring Server is registered as a Linux service (daemon).



```
JCatascopia-Server-v2.0-SNAPSHOT/
├── installer.sh
├── JCatascopia-Server
├── JCatascopiaServerDir
│   ├── JCatascopia-Server-start.sh
│   ├── JCatascopia-Server-stop.sh
│   ├── JCatascopia-Server-v2.0-SNAPSHOT.jar
│   └── resources
│       └── server.properties
└── LICENSE-2.0.html
```

Figure 21: Monitoring Server TAR & RPM Packaging Content

Optionally, before starting the Monitoring Server service, its default properties can be changed. The following table presents the available configurations for a Monitoring Server:

Property	Type	Default Value	Description
Main Settings			
logging	Required	true	When set to true, the JCatascopia Monitoring Server will log abnormal behavior
debug_mode	Required	false	When set to true, the JCatascopia Monitoring Server will literally print every occurring event to the console. This option should ONLY be used for testing reasons
Agent Metric Listener Settings			
agent_port	Required	4242	The port which JCatascopia Monitoring Server will bind to and listen for metric messages distributed by JCatascopia Monitoring Agents (should be the same as the `distributor_port` of each underlying Monitoring Agent). If not required otherwise, this value should NOT be changed
agent_bind_ip	Required	*	The network interface that the AgentLister will bind to. The default value is set to * which indicates that the Monitoring Server will bind to all network interfaces. If it must be changed then it is suggested to use the eth0 interface, but users are not obligated to.
Processing Settings			
num_of_processing_threads	Optional	4	The number of threads that will be used to process, in parallel, received metric messages. The default value is just an example and users are encouraged to any number of threads that meets their needs
HeartBeat Monitoring Settings			
period	Optional	60 seconds	The intensity in which the HeartBeat Monitor should check for Monitoring Agent availability
retry	Optional	3	The number of iterations that the HeartBeat Monitor will allow a Monitoring Agent to be DOWN until it is declared as DEAD
Control Listener Settings			
control_port	Required	4245	The port which JCatascopia Monitoring Server ControlListener binds to. If not required otherwise, this value should NOT be changed
control_bind_ip	Required	*	The network interface that the ControlListener will

			bind to. The default value is set to * which indicates that the Server will bind to all network interfaces. If it must be changed then it is suggested to use the eth0 interface, but users are not obligated to.
		Database Interface Settings	
db_use_database	Optional	false	When set to true, the JCatascopia Monitoring Server will store values in the defined database backend. Users may set this to false for testing purposes (e.g. trying out JCatascopia while a database is not offered)
db_drop_tables_on_startup	Optional	true	When set to true, the JCatascopia Monitoring Server, will delete all its tables upon startup. This is useful to easily clear database and also for testing purposes since starting/stoping server is often. Afterwards, users can set this to false
db_interface	Required	MySQL.DBHandlerWithConnPool	The database interface which will be used (users provide the classpath)
db_host	Required	localhost	The host (ip address) of the database backend
db_user	Required	-	The username of the user which will be used
db_pass	Required	-	The password of the user which will be used
db_database	Required	-	The database which will be used

Table 4: Monitoring Server Configuration Properties

4.4.1.3 JCatascopia Web Interface

The JCatascopia-Web Interface is deployed via the CELAR Provisioner to the Application Orchestration VM. The Web Interface is packaged and re-distributed as: (i) a WAR (Web Application Archive) file for easy and transparent deployment into an Apache Tomcat web container; and (ii) a RPM package which encapsulates the WAR file as well as the required installer to deploy and configure the WAR file in the web container of the Application Orchestration VM. The RPM packaging is the default packaging for CELAR. The following properties must be configured before instantiating the Web Interface:

Property	Type	Default Value	Description
		Main Settings	
debug_mode	Required	false	When set to true, the JCatascopia Web Interface will literally print every occurring event to the console. This option should ONLY be used for testing reasons
		Database Interface Settings	
db_interface	Required	MySQL.DBHandlerWithConnPool	The database interface which will be used to extract metrics
db_host	Required	localhost	The host (ip address) of the database backend
db_user	Required	catascopia_user	The username of the user which will be used
db_pass	Required	catascopia_pass	The password of the user which will be used
db_database	Required	jcatascopiaDB	The database which will be used

Table 5: Web Interface Configuration Properties

4.4.2 JCatascopia REST API

The following table lists the current version of the JCatascopia REST API which is used to expose information to CELAR modules such as the Multi-Level Metrics

Evaluation module, the Decision Module and Application Users via the JCatascopia-Web Interface.

BASE_URL = http://<Application_Orchestration_VM>:<Port>/jcatascopia/restAPI/

URI	Request Type	Parameters	Response Type	Description
/agents	GET	status	JSON Array	Lists Monitoring Agents in deployment along with their metadata. If status parameter is provided (e.g. UP) then only Agents with the specified status are listed
/agents/{agentID}/ availableMetrics	GET	-	JSON Array	Lists the available/offered metrics (and metadata) from the specified, via ID, Monitoring Agent
/agents/ availableMetrics	GET	-	JSON Array	Lists the available/offered metrics (and metadata) for ALL Monitoring Agents
/metrics	POST	comma separated list of metric IDs	JSON Array	Lists the current values (and timestamp) for the metrics specified, via metric IDs, in the request body
/metrics/{metricID}/	GET	interval, tstart, tend	JSON Array	Lists the current value (and metadata) of the specified metric. Optionally: (i) if interval (in seconds) parameter is provided then the latest metric values in the interval are returned; (ii) if tstart and tend are provided then the value in the specified timeframe are returned
/metrics/agent/{agentID}/	GET	-	JSON Array	Lists the current values (and timestamp) for ALL metrics collected for the specified, via agent ID, Monitoring Agent
/subscriptions	GET	-	JSON Array	Lists the created metric subscriptions (and metadata) for the deployment
/subscriptions	PUT	Subscription info [JSON]	-	Creates a new metric subscription based on the provided information (metric, aggregation type, period, etc.)
/subscriptions/{subscriptionID}	GET	-	JSON Array	Lists the metadata for the specified, via subscription ID, metric subscription
/subscriptions/{subscriptionID}	DELETE	-	-	Deletes the subscription specified via subscription ID
/subscriptions/{subscriptionID}/agent/{agentID}	POST	action	-	Based on the provided action (addAgent, removeAgent) the specified via Agent ID is either added or removed to/from the specified subscription

Table 6: JCatascopia REST API

4.4.3 JCatascopia Monitoring System Source Code

The source code for the JCatascopia Monitoring System (v2.0) can be found at GitHub under the following URL:

https://github.com/CELAR/cloud-ms

4.5 Evaluation

This section provides an evaluation report for the CELAR Monitoring System based on the system-wide requirements (when applicable) and the Monitoring System specific requirements as defined in D1.2 (R4.1-R4.12).

Req. No.	R2, R3, R4	Application Deployment, Termination and Real-Time Application Monitoring
Status	DONE	
Means of Assessment/Verification		
Both, JCatascopia Monitoring Servers and JCatascopia Monitoring Agents are deployable in a fully automated manner by the CELAR Provisioner (Slipstream). Specifically, upon instantiating a new Application Orchestration VM[10], the Provisioner downloads from the CELAR repository the latest version of the Monitoring Server and installs it. Afterwards, the Monitoring Server is configured (if required to change any of the default settings in its configuration file). Finally, when all of the CELAR modules residing on the Application Orchestration VM are ready, the Provisioner starts the Monitoring Server service and JCatscopia is up and running. In a similar manner, when a new virtual instance must be instantiated due to an elasticity action or for the initial deployment, the Provisioner will automatically deploy on it a JCatascopia Monitoring Agent, configure it (if required) and start the Monitoring Agent service. Afterwards, monitoring metrics originating from the underlying platform or the deployed cloud application(s) are collected and distributed to the Monitoring Server. Upon application termination, the allocated virtual instances are destroyed and consequently the Monitoring Agents residing on them are removed. When the Application Orchestration VM is terminated, the Monitoring Server(s) are removed without affecting the monitoring process of other applications since the monitoring process for each application is completely independent. These requirements have been successfully verified in a number of demonstrations where CELAR was showcased by deploying and monitoring different domains of cloud applications. Some of these demonstrations include: (i) the Deployment Demo which was showcased at the CELAR Year 1 Review[11]; (ii) CELAR live demo at FIA Assembly[12]; (iii) CELAR Demo at CloudScape VI[13].		
Req. No.	R4.1, R4.2	Collect Resource-Related Metrics, Collect Application-Level Metrics
Status	DONE	
Means of Assessment/Verification		
JCatascopia Monitoring Probes have been implemented to collect both resource-related utilization metrics (i.e. CPU usage, network traffic) from the underlying		

[10] A detailed description of the Application Orchestration VM can be found at D1.1 [D1.1]

[11] Video on YouTube https://www.youtube.com/watch?v=zCkwv23oIJQ&list=UUj2NkQ8fRiFxAO_tOPF8kvQ

[12] Video on YouTube https://www.youtube.com/watch?v=tOVbQ-w6bnw

[13] Video on YouTube https://www.youtube.com/watch?v=tOVbQ-w6bnw

platform and application-specific metrics (i.e. latency, throughput) from a number of cloud applications. The available Monitoring Probes along with the metrics offered by each Probe, are documented in D4.1 (Section 5.4.1), D4.2 (Section 4.3.7) and available for download from the JCatascopia Probe Library on Github at the following URL: https://github.com/dtrihinas/JCatascopia-Probe-Library. It must be noted that all the application-specific Monitoring Probes, requested to date, that are required to monitor the performance of the CELAR pilot applications are implemented and have been tested. Moreover, c-Eclipse offers its users the ability to easily develop JCatascopia Monitoring Probes with pre-defined Monitoring Probe templates (this functionality will be presented in detail in D2.3). These requirements have been successfully verified in the aforementioned demonstrations by utilizing a number of different resource-related and application-specific Monitoring Probes. Furthermore, a number of these Monitoring Probes, have been used to collect monitoring information and are exhibited in the following scientific publications [Trihinas 2014a], [Trihinas 2014b], [Sofokleous 2014], [Copil 2014].		
Req. No.	R4.3	Deploy Custom Monitoring Probes
Status	DONE	
Means of Assessment/Verification		
JCatascopia provides Application Developers with the appropriate mechanisms to deploy their own custom Monitoring Probes either when the Monitoring Agent starts, via the Agent configuration file by defining which Probes to activate; or at runtime, by notifying the Monitoring Agent, via its API, to add a new Monitoring Probe, as described in Section 4.3.2. This requirement has been successfully verified and documented in the aforementioned scientific publications. This requirement will be further assessed by presenting this functionality in the CELAR Year 2 review.		
Req. No.	R4.4, R4.9	Add/Remove Monitoring Instances at Runtime, Elasticity and Adaptability
Status	DONE	
Means of Assessment/Verification		
JCatascopia has the ability to add/remove Monitoring Agents dynamically at runtime in a fully automated manner when an elasticity action occurs without the need to restart the Monitoring Server. This is accomplished due to the JCatascopia Pub/Sub Message Communication Protocol presented in Section 4.3.1 and heartbeat monitoring to provide the appropriate elasticity support to JCatascopia. Additionally, JCatascopia is also able to identify resource-related parameter changes to a virtual instance (e.g. a new disk is attached to VM, therefore total disk utilization and allocated space must be re-calculated) This requirement has been successfully verified in the aforementioned demonstrations and the mechanism behind the JCatascopia Pub/Sub Message Communication Protocol and heartbeat monitoring is documented in D4.1, [Trihinas2014a] and Section 4.3.1 of this Deliverable.		
Req. No.	R4.5	Monitoring Metric Delivery Mechanisms
Status	DONE	
Means of Assessment/Verification		
JCatascopia v2.0 provides both a pull and push delivery mechanism, allowing for entities interested in monitoring information to either: (i) periodically poll the		

Monitoring Server via the JCatascopia REST API for metrics; or (ii) subscribe to metric event notifications, via the JCatascopia push mechanism, to be notified when specific metric thresholds are violated. The JCatascopia pull mechanism is the default metric delivery mechanism and used in all CELAR demonstrations and evaluation studies of JCatascopia. The JCatascopia push notification mechanism was introduced in v2.0 of JCatascopia and has been successfully debugged and tested for a number of elastic applications (e.g. Cassandra DB cluster) where the elasticity controller subscribes only to metric threshold violations to scale the application topology.		
Req. No.	R4.6	Infrastructure Independence
Status	DONE	
Means of Assessment/Verification		
The JCatascopia Monitoring System is platform independent, thus it is deployable and functional on any underlying IaaS cloud platform since its metric collecting mechanisms are not cloud platform specific and both Monitoring Servers and Agents are installable on any platform. This requirement has been successfully verified in the aforementioned demonstrations and scientific publications by deploying JCatascopia and monitoring different cloud applications on: (i) both consortium IaaS providers: GRNET Okeanos public cloud [Okeanos] and Flexiant FCO platform [Flexiant]; (ii) well-known public cloud platforms: Amazon EC2 [Amazon EC2] and RackSpace [RackSpace]; (iii) OpenStack Private Cloud [OpenStack]		
Req. No.	R4.7, R12	Scalability, Near Real-Time
Status	DONE	
Means of Assessment/Verification		
JCatascopia is not fragmented by the number of running Monitoring Agents in an application's topology or the number of Monitoring Probes utilized on each Monitoring Agent. To cope with an increasing number of Monitoring Agents in an application's topology monitoring metric traffic can be redirected through more than one Monitoring Servers, thus creating a hierarchy of Monitoring Servers, allowing the monitoring system to scale. This requirement has been successfully verified by conducting a scalability test which was documented in [Trihinas 2014a]. Briefly, to evaluate JCatascopia scalability, we monitored an elastic cloud service deployment which initially was comprised of two virtual instances but as the imposed workload increased, more instances where added by the elasticity controller. Archiving time[14] was used to evaluate JCatascopia scalability. At first, only one Monitoring Server was utilized and we can observe from Figure 22 that archiving time grows linearly. However, if the average archiving time rises and becomes larger than the rate monitoring metrics arrive, then metrics will either be discarded or at the very best queued at the Monitoring Server. For this reason, we then utilized a hierarchy of Monitoring Servers by adding 2 intermediate Monitoring Servers to the previous topology which were utilized to process, aggregate and distribute metrics to the root Monitoring Server. From Figure 22 we observe that archiving time is relatively stable and thus we conclude that when archiving time is		

[14] Archiving time is the average time required for the Monitoring Server to process and successfully store in the Monitoring Database a received monitoring metric

high, we can redirect monitoring metric traffic through intermediate Monitoring Servers, allowing the monitoring system to scale.

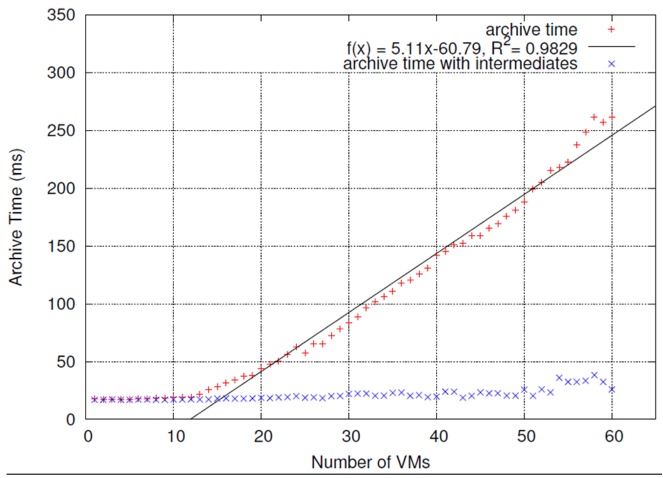


Figure 22: JCatascopia Scalability Evaluation

Req. No.	R4.8	Non-Intrusiveness
Status	DONE	
Means of Assessment/Verification		

JCatascopia Monitoring Agents are designed to have a low-runtime footprint, in order to not affect the performance (in terms of CPU, memory and network utilization) of cloud services running on user-paid virtual instances where they reside on.

This requirement has been successfully verified by conducting a runtime impact evaluation test for JCatascopia Monitoring Agents which was documented in [Trihinas 2014a]. Briefly, to evaluate the runtime footprint of Monitoring Agents, we selected to monitor a number of cloud services originating from different domains, indicatively: a Cassandra Database backend, a video streaming web service and a distributed data clustering mechanism; and compared JCatascopia runtime footprint to other similar agent-based monitoring systems: Ganglia and Lattice [Clayman 2011]. It must be noted that for the purposes of the evaluation, we used small VM flavors (2 VCPU, 2GB RAM, 10GB Disk) to enhance the difference in the impact between the under comparison systems. From the following figures, we observe that Lattice has a larger runtime footprint than the other two monitoring systems. From Figure 23 we observe that Ganglia's footprint is smaller than JCatascopia due to the fact that we collected low-level system metrics and only two application specific metrics for Cassandra (read/write query latency). However, from Figure 24 we notice that the difference between the two systems is under 0.03% and in Figure 25 both systems exhibit the same performance overhead where a total of 19 and 29 metrics were collected respectively. Additionally, we compared the network utilization (Figure 26) footprint of JCatascopia and Ganglia where we observe that JCatascopia has a smaller footprint than Ganglia and if filtering is turned on, at Probe level, then network utilization drops even more. We conclude, that when in need of application-level monitoring for large-scale distributed deployments, for a small runtime overhead, JCatascopia can reduce monitoring network traffic and consequently monitoring cost.

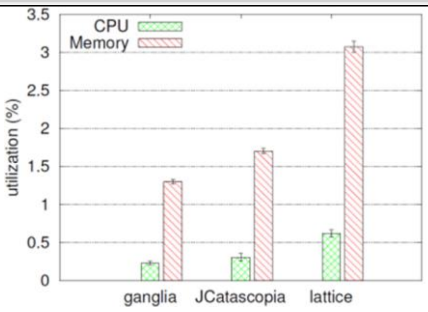


Figure 23: Agent Utilization – Cassandra Node

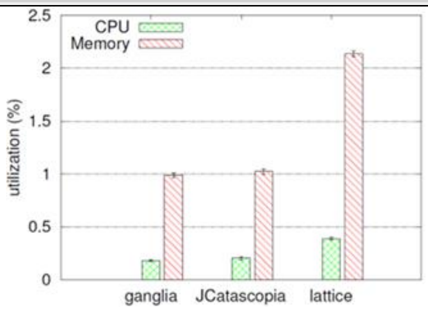


Figure 24: Agent Utilization – HASCOP Node

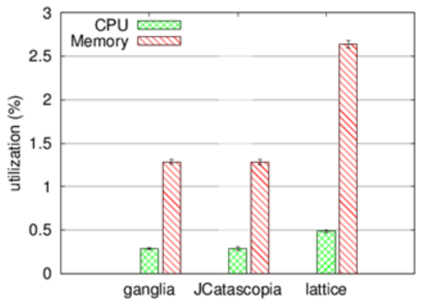


Figure 25: Agent Utilization – Video Streaming Service

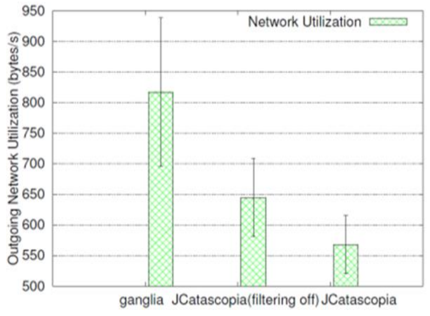


Figure 26: Agent Network Utilization

Req. No.	R4.10	Extensibility
Status	DONE	
Means of Assessment/Verification		
As previously stated, JCatascopia is both extensible and modular, thus allowing its users to: (i) utilize their own database backend or select one of from the already implemented ones; and (ii) develop and deploy (see R4.3) their own custom Monitoring Probes to collect metrics not currently available by the Monitoring System. This requirement has been successfully verified by utilizing different database backends at a number of demonstrations where CELAR was showcased. For example, at the CELAR Year 1 review and for the purposes of [Sofokleous 2014], the MySQL interface was used, whereas for [Copil 2014] [Trihinas 2014b] the Cassandra interface was selected. Regarding Monitoring Probe development and deployment see requirements R1, R2 and R3.		
Req. No.	R4.11, R7	Robustness and Fault-Tolerance, High-Availability
Status	DONE	
Means of Assessment/Verification		
In JCatascopia design we have taken into consideration the following unexpected erroneous situations which may occur and address them as follows: (i) Monitoring Agent is assumed dead due to VM failure. Due to the flexibility provided by the JCatascopia Pub/Sub Message Communication Protocol and heartbeat monitoring, a terminated/dead Monitoring Agent will be removed from the list of available Agents and it will not affect the functionality of the rest of the monitoring process. Additionally, in v2.0 (see Section 4.3.1) we also consider the situation where an Agent was only temporarily unavailable and then made available again. (ii) Monitoring Probe is not collecting metrics due to problem at metric source. Monitoring Probes run on different threads and are isolated from each other. A failure on one Probe is caught, logged and hidden from the other Monitoring Probes. (iii) Monitoring Server is assumed dead (i.e. failure of Application Orchestration VM). If a Monitoring Server is		

assumed dead, then another Monitoring Server can replace the current one and automatically re-construct its state in a few minutes without requiring for the Monitoring Agents to be restarted. This is, again, due to the JCatascopia Pub/Sub Message Communication Protocol where the Monitoring Server can replace another one by simply binding to the network interface of the previous Server without the need to subscribe to the metric stream of its respected Monitoring Agents since the subscription process is initiated by the Agents and the metric stream continues to exist. As discussed above, all of these challenges have been identified and addressed while monitoring the applications for the purpose of [Sofokleous 2014], [Trihinas 2014b] and [Copil 2014]. Moreover, JCatascopia is compliant with the CELAR High-Availability schema presented in D3.2 [D3.2]. Minor testing will still be performed in the final months of T4.2 as to verify JCatascopia availability, under these erroneous situations, while integrated in the full CELAR eco-system which is a task for the consortium in the following months.

Table 7: CELAR Monitoring System Requirement Evaluation

5 Multi-Level Metric Evaluation Module

This section documents the CELAR Multi-Level Metric Evaluation Module, which provides monitoring information mapped to the application structure for the CELAR Decision Module (WP5) to address the requirements of WP4 Task 4.2: Composable Cost Evaluation. In this section we provide a short overview of the Multi-Level Metric Evaluation Module and its implementation (MELA), present its new features, provide a documentation report and conclude with an evaluation and verification analysis of its requirements.

5.1 Overview

In general, elastic cloud applications can re-configure individual components or the whole application at run-time, due to various elasticity requirements. Such run-time re-configuration can either focus on: (i) vertical scaling, adding/removing resources to existing components; or (ii) horizontal scaling, duplicating components/instantiating more instances of the same type. Due to individual horizontal scaling mechanisms, one application component could "burst" by a controller, creating a new instance in unused virtual machines along other components, or in new machines. This generates two main issues. First, as components can dynamically appear/disappear on virtual machines, a monitoring system for elastic applications must be able to dynamically start/stop collecting metrics on individual virtual machines. Secondly, as virtual machines themselves are created/destroyed dynamically at run-time, if monitoring information is associated only with each virtual machine, it will be lost during scale-in operations which destroy virtual machines.

While the first issue is solved by JCatascopia, which supports dynamic addition and removal of metrics collected by a Monitoring Agent, for solving the Virtual Machine (VM) volatility issue, we must associate monitoring information with the application structure. Addressing this second issue is the initial objective of the CELAR Multi-Level Metric Evaluation Module, implemented by MELA. Based on monitoring information mapped to the application structure, MELA provides both real-time and historical Composable Cost Evaluation according to different cloud pricing schemes.

5.2 Requirements

The requirements listed in D4.1 (Section 3.3 and 3.4) were updated to better represent the scope and target goals of the CELAR Multi-Level Metric Evaluation Module and were documented in D1.2 (Requirements R4.13-R4.20) and are considered as the current requirements.

5.3 New Features

The following subsections document the latest features and improvements made available in the current version of the Multi-Level Metric Evaluation Module.

5.3.1 Service Oriented Architecture

To enhance flexibility, we have re-designed MELA's architecture, presented in D4.1 Section 6, into a service-oriented architecture (Figure 27) centered around a shared data repository, enabling easy addition/removal of services which operate on the same data. As MELA also provides information used by the Decision Module, functionality belonging to WP4 (as described in D4.1) and WP5 (as described in D5.2) is implemented under separate services (as marked in the architecture). Moreover, the MELA user can choose what services to deploy and run, depending on individual requirements, thus, offering maximum usage flexibility.

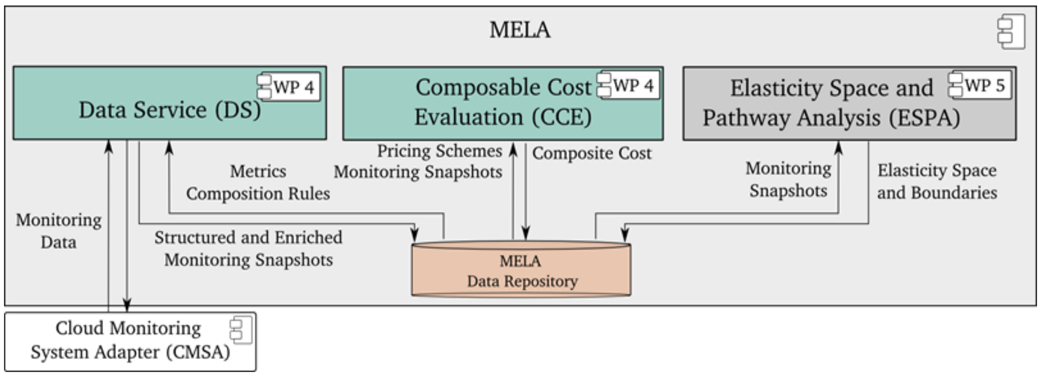

MELA
Data Service (DS)
WP 4
Composable Cost Evaluation (CCE)
WP 4
Elasticity Space and Pathway Analysis (ESPA)
WP 5
Pricing Schemes
Monitoring Snapshots
Composite Cost
Metrics
Composition Rules
Monitoring Snapshots
Elasticity Space and Boundaries
Monitoring Data
Structured and Enriched Monitoring Snapshots
MELA Data Repository
Cloud Monitoring System Adapter (CMSA)

Figure 27: MELA Architecture

For separating functionality offered to WP4 from the functionality for WP5 and to increase the usability of different MELA services, we improved the MELA structure presented in D4.1 Section 6.2.1. We have moved functionality relating to monitoring data aggregation from the previous MELA *Analysis Service* to MELA *Data Service*, which now collects monitoring data, structures, enriches and stores it in the MELA *Data Repository*. We have also moved the Graphical User Interface and REST API related to MELA configuration and run-time monitoring of elastic applications to MELA *Data Service*.

As after the refactoring, in the former MELA Analysis Service we had only functionality belonging to WP5 (as presented in D5.2 Section 4.3), we renamed the service to MELA *Elasticity Space and Pathway Analysis*, which gives a better indication of the service purpose.

Moreover, we added a new MELA *Composable Cost Evaluation Service* to implement complex cost evaluation based on complex cloud pricing schemes, based on the monitored data collected and enriched by the MELA Data Service.

In this document we will focus only on the services offering functionality for WP4: (i) the Data service; and (ii) the Composable Cost Evaluation service.

5.3.2 Multi-Service Support

The MELA version presented in D4.1 could handle one application instance or version for one MELA deployment. While such an approach conceptually provides a higher level of security, as monitoring information from different applications is not stored in the same repository, it exhibits usage limitations. This is more apparent in the case a user wants to monitor multiple applications, or different versions of the same application. Thus, we have updated MELA's services and corresponding REST APIs to handle multiple services and service versions. Details about the REST API of MELA's WP4 services are presented in Section 5.4.1.3.

5.3.3 Composable Cost Evaluation

Composable Cost Evaluation corresponds to Task 4.2 in WP4 (M4-M18), and was initially introduced in D4.1, Section 4.4. In this deliverable we present an in depth analysis over issues related to cost of cloud applications, based on which we introduce a cost model and composition mechanism. According to D4.1, Section 4.2, the model used to describe the structure of an application deployed on a cloud is the one defined in Deliverable D5.1 [D5.1 - Section 4.1.1].

To design a cost composition and evaluation mechanism, we first analyze a subset of cloud offered services which can be used for deploying elastic applications in the cloud. To ensure we understand the State-of-the-Art in offered cloud services, we focus on four main cloud providers: Amazon[15], Rackspace[16], HP Cloud[17], and Windows Azure[18]. We focus on three categories of services which can be used in deploying entire applications or individual components in cloud: (i) Infrastructure as a Service (IaaS); (ii) Platform as a Service (PaaS); and (iii) Management as a Service (MaaS). What is important from a cost perspective is that usually, different combinations of services have different pricing schemes, and different elastic applications could use at run-time different services with specific configurations, influencing the application cost.

Category	Subcategory	Cloud Offered Services			
		Amazon	Rackspace	HP Cloud	Windows Azure
IaaS	Computing	vCPU, GPGPU	vCPU	vCPU	vCPU
	Memory	RAM	RAM	RAM	RAM, Azure Caching
	Virtual Machine	m1.small, m1.large, ..., cg1.4xlarge	FG 256MB, FG 512MB, ..., NG 30GB	Extra Small, Small,..., Double Extra Large	Extra Small, Small, ..., Extra large, A6, A7
	Persistent Storage	Elastic Block Store	Block Storage	Block Storage	Data Disk
	Shared Storage	S3, Glacier, Cloud Front	Cloud Files	Object Storage, CDN	Blob Storage
	Network	Public IP, Cluster Networking, Virtual Private Cloud, Route 53, Direct Connect	Public IP, DNS	Public IP, HP Cloud DNS	Public IP, Virtual Network
PaaS	Operating System	Linux, RHEL, SLES, Windows (Standard, with SQL Standard, with SQL Web)	Linux, RHEL, Windows (Standard, with SQL Standard, with SQL Web), Brocade Vyatta vRouter	CentOS, Debian, Fedora, Ubuntu, Windows Server	Windows, Linux
	Message Queue	Simple Queue Service	-	HP Cloud Messaging	Azure Service Bus, Azure Queue
	Workflow Engine	Simple Workflow Service	-	-	Azure Workflows, SharePoint Workflows
	Communication Service	Simple Email Service, Simple Notification Service	-	-	Notification Hubs
	Data Processing	Oracle E-Business Suite, SAP Business Objects,	Microsoft SharePoint	-	HD Insight, Azure SQL

[15] http://aws.amazon.com/
[16] http://www.rackspace.com/
[17] http://www.hpcloud.com/
[18] https://azure.microsoft.com/

		Microsoft SharePoint, EMR			Reporting, Microsoft SharePoint
	Load Balancer	Elastic Load Balancing, Auto Scaling	Cloud Load Balancers	Cloud Load Balancer	Traffic Manager
	Database Services	Sql Server, Elasti Cache, RedShift, RDS, Dynamo DB} &{Cloud Databases	Cloud Databases	Cloud Relational Database	SQL Server: Web, Business, and Premium
MaaS	Monitoring	Cloud Watch	Cloud Monitoring	Cloud Monitoring	Health and availability monitoring
	Resource Reservation	On-Demand, Reserved (Light, Medium, Heavy Usage), Spot	Managed, Not Managed	On-Demand	Pay-As-You-Go, 6 or 12-Month Plan
	Backup	-	Cloud Backup	-	Azure Backup

Table 8: Cloud Services Offered by Main Cloud Providers

While the previous analysis indicates cost can be complex, involving different pricing schemes for different combinations of services, cost complexity also depends on the complexity of a cloud application's deployment structure. Moreover, depending on the level of knowledge and particular interests of the CELAR user, some users might be interested in very complex and complete cost evaluation, while others might be content with an estimation over cost, even if not completely accurate.

To this end, we decided to provide two cost composition mechanisms, one using MELA's composition rules for directly enriching monitoring information with an application's cost based on simple cost schemes; and one providing a separate complex cost estimation service, using detailed complex cloud pricing schemes.

5.3.3.1 Enriching Monitoring Information with Cost

The first introduced cost evaluation mechanism allows a user to enrich monitoring information with cost, and monitor the expected cost for the current application structure, i.e., how much the user is expected to pay considering the current structure. This feature provides run-time cost monitoring and can be used if the cloud pricing scheme is simple (e.g., cost per VM type per Hour), and thus it can be computed by applying simple arithmetical operations over monitoring information (e.g., multiplying the number of VMs with cost per VM). A user can utilize MELA's metric composition rules to enrich monitoring information with cost. This mechanism was initially introduced in D4.1, Section 4.4.2. For readability, we describe it again here, incorporating additional details and explanations.

To monitor and evaluate cost, MELA metric composition rules can be used to combine the values of the monitored metrics with simple cloud pricing schemes. The cost per used cloud services can be computed by a rule multiplying the number of instances of application components running on top of such services, with the cost of running the service over any desired time frame. By defining cross-layer metric composition, cost can be aggregated and associated to a particular level from the application structure model. Defining composition rules evaluating cost at each level provides a multi-level cost composition, describing cost for the whole Application, then for each Complex Component, for each Component, down to the cost per individual VM. Such decomposition enables the Decision Module to analyze the application cost in a top-down approach, and take appropriate adaptation actions. The conceptual view over how the cost can be computed for each individual virtual machine, and then aggregated to get cost per application component, and then overall cost is depicted in Figure 28.

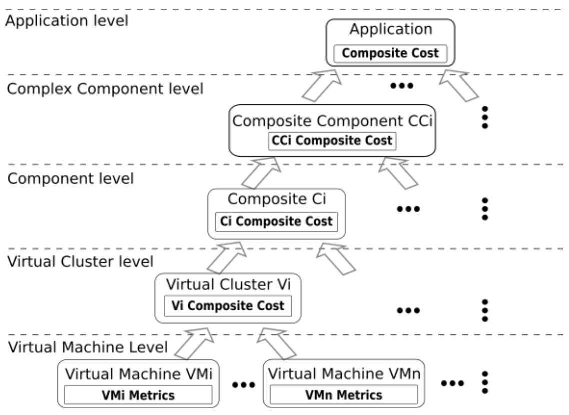


Figure 28: Conceptual Multi-level Cost Composition

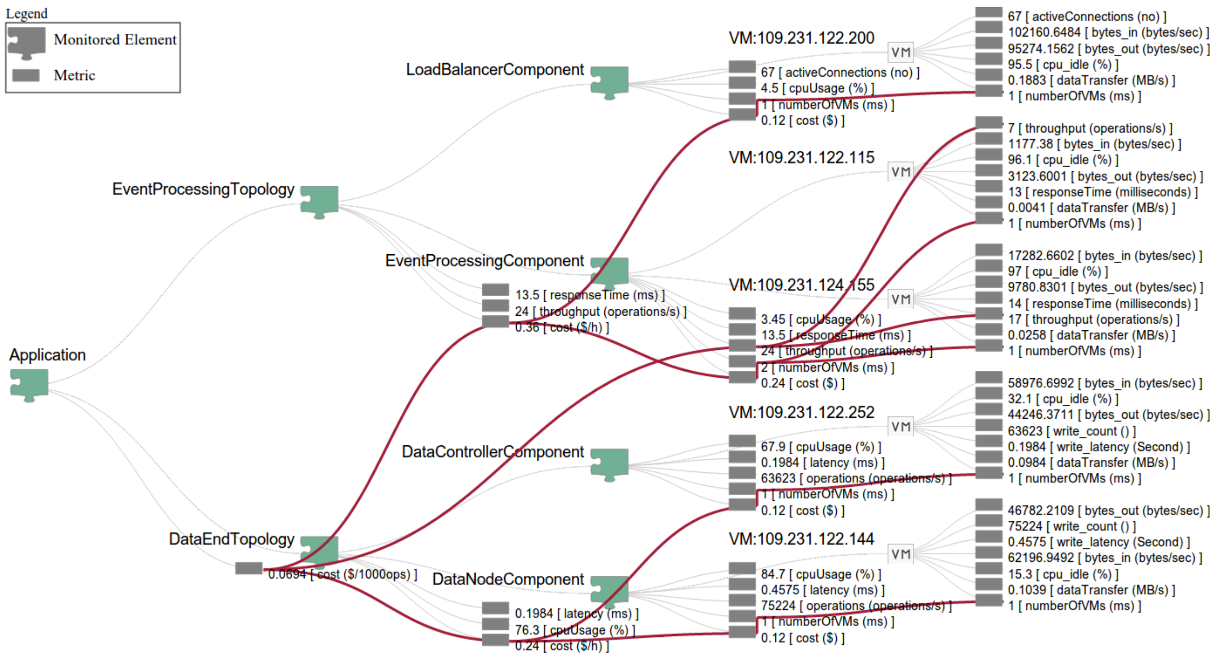
Defining metric composition rules is a joint responsibility task of the CELAR Application Expert/Developer and CELAR expert, as they involve both metrics available for the application VMs, which might be cloud specific, and metrics collected from the application components. In the following Listing we depict a potential composition rule, which first creates a new metric, `numberOfVMs`, with fixed value `1` for all virtual machines (VM) belonging to the application. Then, it computes the number of VMs used by each application component with a `SUM` operation applied for each application component over the `numberOfVMs` metric collected from the component's VMs. Finally, the cost is computed by multiplying with a `MUL` operation the assumed cost for running a VM, `0.12$/h`, with the number of virtual machines used by each component, obtaining the cost for each component.

```
<CompositionRule TargetMonitoredElementLevel="VM">
   <ResultingMetric type="RESOURCE" measurementUnit="ms" name="numberOfVMs"/>
   <Operation value="1" type="SET_VALUE"/>
</CompositionRule>

<CompositionRule TargetMonitoredElementLevel="COMPONENT">
     <ResultingMetric type="RESOURCE" measurementUnit="no." name="numberOfVMs"/>
     <Operation MetricSourceMonitoredElementLevel="VM" type="SUM">
       <ReferenceMetric type="RESOURCE" name="numberOfVMs"/>
     </Operation>
</CompositionRule>

<CompositionRule TargetMonitoredElementLevel="COMPONENT">
  <ResultingMetric type="RESOURCE" measurementUnit="$/h" name="cost"/>
  <Operation type="MUL" value="0.12"
                      MetricSourceMonitoredElementLevel=" COMPONENT " >
     <ReferenceMetric type="RESOURCE" name="numberOfVMs"/>
  </Operation>
</CompositionRule>
```
Listing 4: Computing VM Cost per Application Component

After calculating the cost of using the cloud services for the entire application or application components, the CELAR user can use simple composition rules to divide the cost to other monitored metrics, such as "throughput", obtaining cost relative to its application usage. In Figure 29 we depict an example of enriching monitoring information with simple cost evaluation using the previous cost composition rule to monitor cost on a distributed elastic application used as a pilot in [Moldovan 2014]. In this case all used cloud services (i.e., virtual machines in this case) have the same cost, and the cost composition is marked with red lines.



Legend
Monitored Element
Metric
Application
EventProcessingTopology
LoadBalancerComponent
EventProcessingComponent
13.5 [responseTime (ms)]
24 [throughput (operations/s)]
0.36 [cost ($/h)]
DataControllerComponent
DataEndTopology
0.0694 [cost ($/1000ops)]
DataNodeComponent
0.1984 [latency (ms)]
76.3 [cpuUsage (%)]
0.24 [cost ($/h)]
VM:109.231.122.200
VM
67 [activeConnections (no)]
4.5 [cpuUsage (%)]
1 [numberOfVMs (ms)]
0.12 [cost ($)]
67 [activeConnections (no)]
102160.6484 [bytes_in (bytes/sec)]
95274.1562 [bytes_out (bytes/sec)]
95.5 [cpu_idle (%)]
0.1883 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
VM:109.231.122.115
VM
7 [throughput (operations/s)]
1177.38 [bytes_in (bytes/sec)]
96.1 [cpu_idle (%)]
3123.6001 [bytes_out (bytes/sec)]
13 [responseTime (milliseconds)]
0.0041 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
VM:109.231.124.155
VM
3.45 [cpuUsage (%)]
13.5 [responseTime (ms)]
24 [throughput (operations/s)]
2 [numberOfVMs (ms)]
0.24 [cost ($)]
17282.6602 [bytes_in (bytes/sec)]
97 [cpu_idle (%)]
9780.8301 [bytes_out (bytes/sec)]
14 [responseTime (milliseconds)]
17 [throughput (operations/s)]
0.0258 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
VM:109.231.122.252
VM
67.9 [cpuUsage (%)]
0.1984 [latency (ms)]
63623 [operations (operations/s)]
1 [numberOfVMs (ms)]
0.12 [cost ($)]
58976.6992 [bytes_in (bytes/sec)]
32.1 [cpu_idle (%)]
44246.3711 [bytes_out (bytes/sec)]
63623 [write_count ()]
0.1984 [write_latency (Second)]
0.0984 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
VM:109.231.122.144
VM
84.7 [cpuUsage (%)]
0.4575 [latency (ms)]
75224 [operations (operations/s)]
1 [numberOfVMs (ms)]
0.12 [cost ($)]
46782.2109 [bytes_out (bytes/sec)]
75224 [write_count ()]
0.4575 [write_latency (Second)]
62196.9492 [bytes_in (bytes/sec)]
15.3 [cpu_idle (%)]
0.1039 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]

Figure 29: Enriching Monitoring Application with Simple Cost

5.3.3.2 *Complex Cost Evaluation Based on Cloud Pricing Schemes*

While with the previous approach a user can enrich monitoring information with simple cost schemes in order to monitor cost at run-time, cost of using cloud services to run elastic applications is, in many cases, more complex. Many existing cloud services can be configured and used in association, either mandatory or optional, with other services. For example, considering the Amazon IaaS services, a particular virtual machine (VM) service can be instantiated on different *"availability regions"*, the cost of the VM service depending on the chosen region. Many of the offered VM types can optionally use at run-time the Amazon Elastic Block Store storage service [AmazonEBS], and for additional cost, the VM can be instantiated as EBS Optimized. Moreover, some Amazon VM services, e.g., T2 series (t2.micro, t2.small, t2.medium), can be used only in conjunction with other services, in this case with an EBS storage. Thus, the cost of running T2 instances depends on the type of instance (i.e., instance resources), the availability region where the instance is deployed, and the cost of using the EBS storage. In turn, the cost of using EBS depends on the EBS storage option chosen, from General Purpose EBS at `$0.10 per GB-month`, to Amazon EBS Magnetic volumes costing `$0.05 per GB-month` of provisioned storage and `$0.05 per 1 million I/O requests`.

Evaluating cost even over a single IaaS service can be complex, due to different possible configurations of that service. For capturing this complexity, we define a model [Moldovan 2014] for associating pricing schemes to cloud offered services (Figure 30), considering cloud offered services as service units, which can have *Mandatory* and *Optional* configuration options with respect to *Cost, Quality,* and *Resources.* Each of these entities, i.e., configuration options, or the service unit, can have associated different Cost Functions with different *Cost Element* instances. A Cost Element describes one component of the entity's cost, such as cost per using EBS, or cost per particular availability regions. Cost can be *Periodic* (e.g., hourly, monthly), or per *Usage* (e.g., per million I/O), is reported per periodicity or usage *Unit* (e.g., 1 month, 1 million I/O), over a service *Metric* (e.g., VM uptime, I/O). Cost can be defined from static cost per period/usage, to cost intervals, such as first million I/O free, next million $0.05. We capture such cost intervals using a `costFunction` specifying cost over different intervals of measured *Units* of the cost *Metric*. Finally, as cost can depend on the mandatory/optional association of the entity with other entities (service units, or quality, cost, resource options), an `inAssociationWith` property describes if the described cost should be applied when the entity is associated with certain other entities.

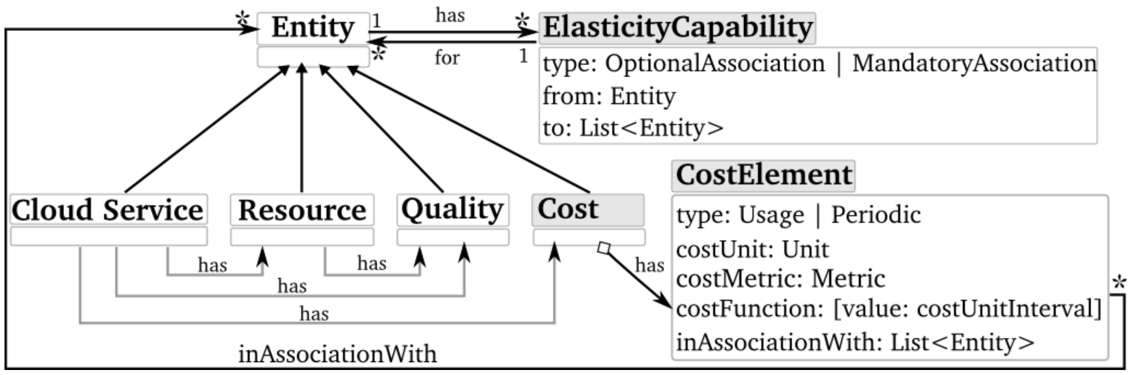


Figure 30: Composite Cost Model

Using this model for associating cost to cloud services, we can capture from simple cost functions, such as fixed cost per used cloud service, to cost per service if used in conjunction with other services, with respect to the service usage so far (usage intervals).

The description of the cloud services available for a particular cloud provider can be collected from various sources, e.g. from the CELAR Information System to manual specification by a user, depending if MELA Composable Cost Evaluation is deployed within CELAR or as a standalone service. We provide a REST API (see Section 5.4.3.2) for submitting the cloud services with associated pricing schemes using an XML-based format.

Beside the description of the cloud services' pricing schemes, to evaluate cost it is important to capture at run-time, after each elasticity action, the application structure containing the associated used cloud services (e.g., VMs, monitoring, storage services), and their particular used configuration. This is important as different configurations could have different pricing schemes, such as higher cost for higher network or storage performance. To this end, the MELA Composable Cost Evaluation uses internally an extended version of the application description model described in Deliverable D5.1 [D5.1 - Section 4.1.1]. We add to each of the Application, Complex Component and Component instances, the option to specify the used cloud services as Entity instances, according to Composite Cost Model (Figure 31). This model is used by MELA users (CELAR Components or human users) during MELA initialization and after each elasticity action, to inform MELA Data Service about the application structure and the type and configuration of used cloud services for the application.

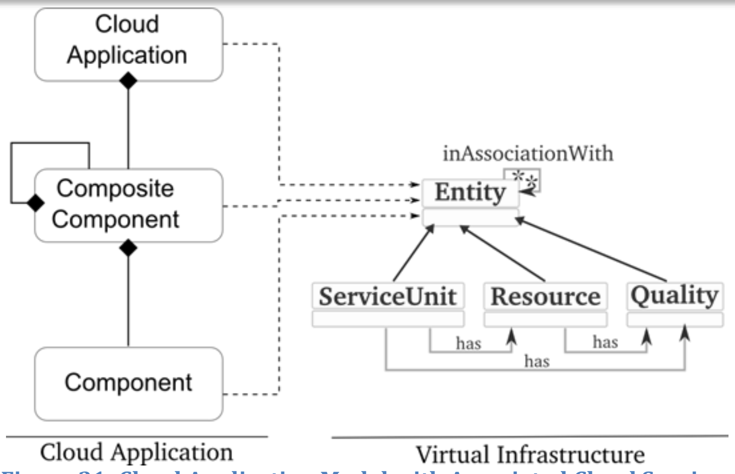


Figure 31: Cloud Application Model with Associated Cloud Services

Using this cost composition model, a MELA Composable Cost Evaluation service computes in detail the cost for the whole application or individual components, using information about the cloud services used by the application, collected from the integrated CELAR Information System, using the process depicted in Figure 32, any of the Cloud Application, Complex Component, or Component from the described cloud application model are considered by MELA as a Monitored Element. For each monitored snapshot collected and enriched by MELA Data Service, for each monitored element of the application, we retrieve the currently used cloud services and their configuration (Step 1). Then, for each used service, we search in the cloud services model if there are any defined cost functions (Step 2). Next, we analyze if the cost function is applicable (Step 3), by evaluating which cost function applies to the current Resources and Quality configuration of the service, and its current association with other services used by the application. If the cost function is applicable, for each cost element of the function (Step 5) we check if the element specifies a PERIODIC cost, which must be computed over service usage in a time period, or an USAGE cost, which must be computed directly over the usage. For example, network data transfer is usually a USAGE cost as it is paid per transferred amount, while VM cost is usually a periodic one, a user paying hourly/monthly for using the VM. Depending on the metric to which the cost element is associated (e.g., data transfer, or number of VMs), we compute the cost (step 8). Lastly, the computed cost is added as a COST metric in the application monitoring snapshot, enriching the monitoring information. In this way, other CELAR Components such as Decision Module (WP5) can use the cost information as any other monitored metric, just as the data collected and enriched by MELA Data Service was used until now, increasing the usability of the Composable Cost Evaluation service.

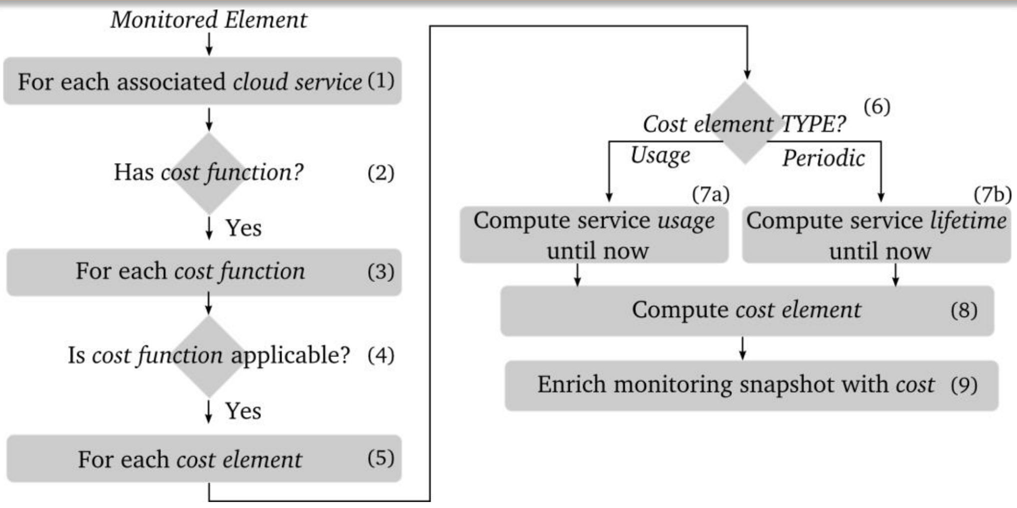
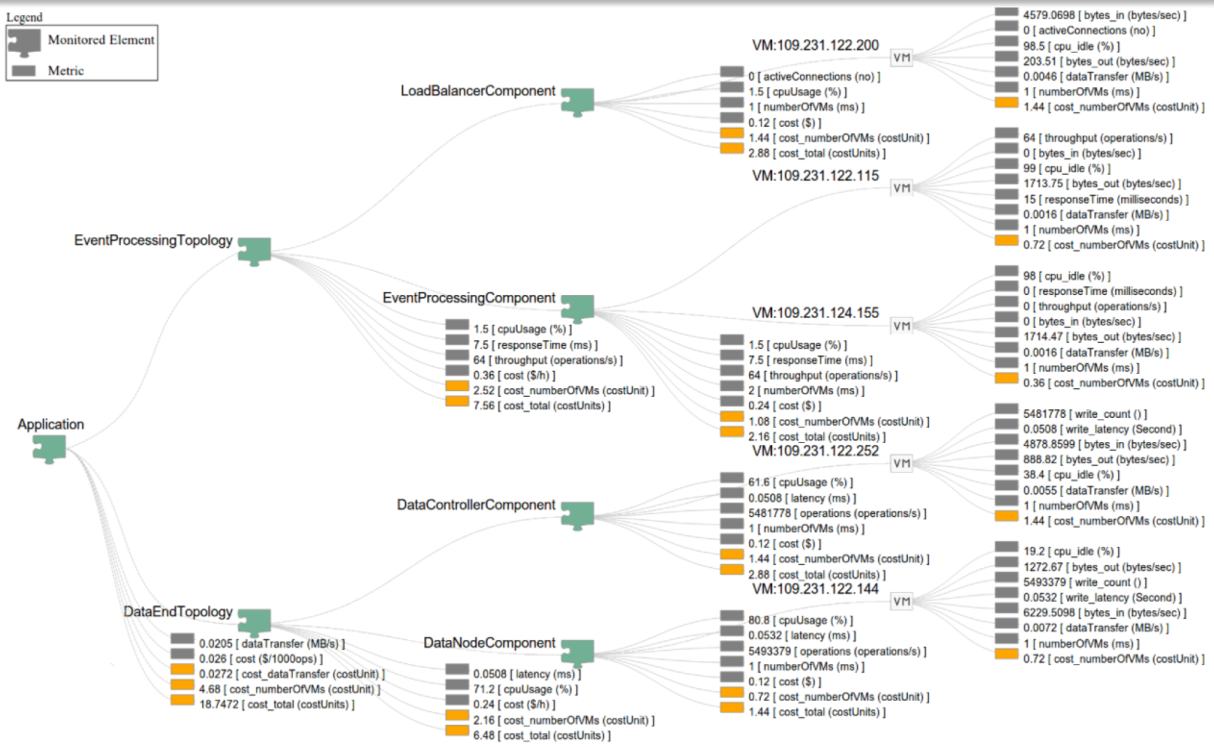

Monitored Element
For each associated cloud service (1)
Has cost function? (2)
Yes
For each cost function (3)
Is cost function applicable? (4)
Yes
For each cost element (5)
Cost element TYPE? (6)
Usage
Periodic
(7a)
Compute service usage until now
(7b)
Compute service lifetime until now
Compute cost element (8)
Enrich monitoring snapshot with cost (9)

Figure 32: Complex Cost Evaluation Process

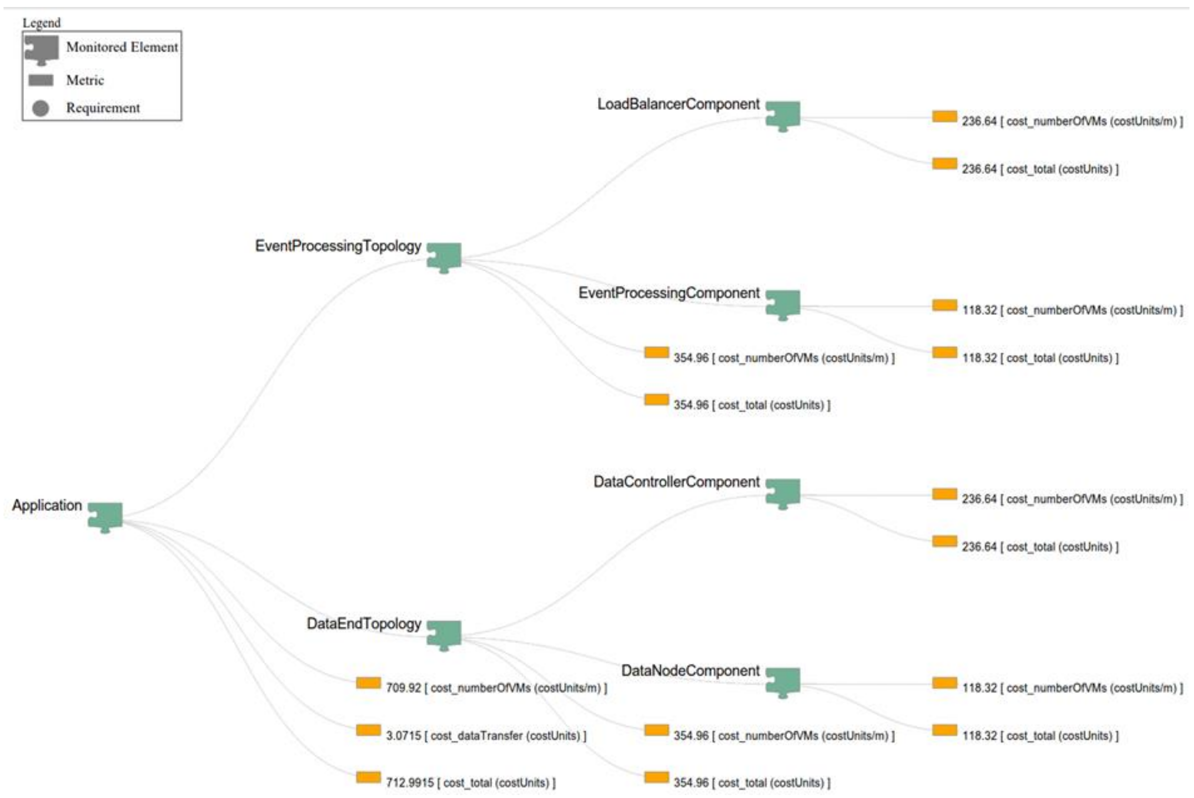
Based on the computed cost for each monitoring snapshot, the total cost of the service is computed by summing up the cost of each monitoring snapshot, with respect to the type of cost elements, i.e., PERIODIC or USAGE.

In Figure 33 we depict an example of enriching monitoring information snapshots with complex cost evaluation metrics (marked with orange) on a distributed elastic application used as a pilot in [Moldovan 2014]. In this, used virtual machines have different cost. Moreover, using the enhanced cloud application model, we specified the whole application uses a "Network" service, for which it has a cost function with a USAGE type cost element indicating first GB of transferred data is free, then the next 10 have different price. In this case, using our complex cost evaluation approach, we obtain cost for each cost element, here for VM usage and data transfer, and total cost for each application component, complex component (depicted as Topology), and whole application. Moreover, the total cost so far of the application is depicted in Figure 34, containing a summation in time with respect to the cost's payment period (e.g., hourly), of the cost rate computed for each monitored snapshot.



Legend
Monitored Element
Metric
Application
EventProcessingTopology
LoadBalancerComponent
VM:109.231.122.200
0 [activeConnections (no)]
1.5 [cpuUsage (%)]
1 [numberOfVMs (ms)]
0.12 [cost ($)]
1.44 [cost_numberOfVMs (costUnit)]
2.88 [cost_total (costUnits)]
4579.0698 [bytes_in (bytes/sec)]
0 [activeConnections (no)]
98.5 [cpu_idle (%)]
203.51 [bytes_out (bytes/sec)]
0.0046 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
1.44 [cost_numberOfVMs (costUnit)]
VM:109.231.122.115
64 [throughput (operations/s)]
0 [bytes_in (bytes/sec)]
99 [cpu_idle (%)]
1713.75 [bytes_out (bytes/sec)]
15 [responseTime (milliseconds)]
0.0016 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
0.72 [cost_numberOfVMs (costUnit)]
EventProcessingComponent
1.5 [cpuUsage (%)]
7.5 [responseTime (ms)]
64 [throughput (operations/s)]
0.36 [cost ($/h)]
2.52 [cost_numberOfVMs (costUnit)]
7.56 [cost_total (costUnits)]
VM:109.231.124.155
1.5 [cpuUsage (%)]
7.5 [responseTime (ms)]
64 [throughput (operations/s)]
2 [numberOfVMs (ms)]
0.24 [cost ($)]
1.08 [cost_numberOfVMs (costUnit)]
2.16 [cost_total (costUnits)]
98 [cpu_idle (%)]
0 [responseTime (milliseconds)]
0 [throughput (operations/s)]
0 [bytes_in (bytes/sec)]
1714.47 [bytes_out (bytes/sec)]
0.0016 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
0.36 [cost_numberOfVMs (costUnit)]
DataEndTopology
0.0205 [dataTransfer (MB/s)]
0.026 [cost ($/1000ops)]
0.0272 [cost_dataTransfer (costUnit)]
4.68 [cost_numberOfVMs (costUnit)]
18.7472 [cost_total (costUnits)]
DataControllerComponent
VM:109.231.122.252
61.6 [cpuUsage (%)]
0.0508 [latency (ms)]
5481778 [operations (operations/s)]
1 [numberOfVMs (ms)]
0.12 [cost ($)]
1.44 [cost_numberOfVMs (costUnit)]
2.88 [cost_total (costUnits)]
5481778 [write_count ()]
0.0508 [write_latency (Second)]
4878.8599 [bytes_in (bytes/sec)]
888.82 [bytes_out (bytes/sec)]
38.4 [cpu_idle (%)]
0.0055 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
1.44 [cost_numberOfVMs (costUnit)]
DataNodeComponent
0.0508 [latency (ms)]
71.2 [cpuUsage (%)]
0.24 [cost ($/h)]
2.16 [cost_numberOfVMs (costUnit)]
6.48 [cost_total (costUnits)]
VM:109.231.122.144
80.8 [cpuUsage (%)]
0.0532 [latency (ms)]
5493379 [operations (operations/s)]
1 [numberOfVMs (ms)]
0.12 [cost ($)]
0.72 [cost_numberOfVMs (costUnit)]
1.44 [cost_total (costUnits)]
19.2 [cpu_idle (%)]
1272.67 [bytes_out (bytes/sec)]
5493379 [write_count ()]
0.0532 [write_latency (Second)]
6229.5098 [bytes_in (bytes/sec)]
0.0072 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
0.72 [cost_numberOfVMs (costUnit)]
VM

Figure 33: Monitoring Snapshot with Complex Cost Evaluation



Legend
Monitored Element
Metric
Requirement
LoadBalancerComponent
236.64 [cost_numberOfVMs (costUnits/m)]
236.64 [cost_total (costUnits)]
EventProcessingTopology
EventProcessingComponent
118.32 [cost_numberOfVMs (costUnits/m)]
118.32 [cost_total (costUnits)]
354.96 [cost_numberOfVMs (costUnits/m)]
354.96 [cost_total (costUnits)]
Application
DataControllerComponent
236.64 [cost_numberOfVMs (costUnits/m)]
236.64 [cost_total (costUnits)]
DataEndTopology
709.92 [cost_numberOfVMs (costUnits/m)]
3.0715 [cost_dataTransfer (costUnits)]
712.9915 [cost_total (costUnits)]
DataNodeComponent
354.96 [cost_numberOfVMs (costUnits/m)]
354.96 [cost_total (costUnits)]
118.32 [cost_numberOfVMs (costUnits/m)]
118.32 [cost_total (costUnits)]

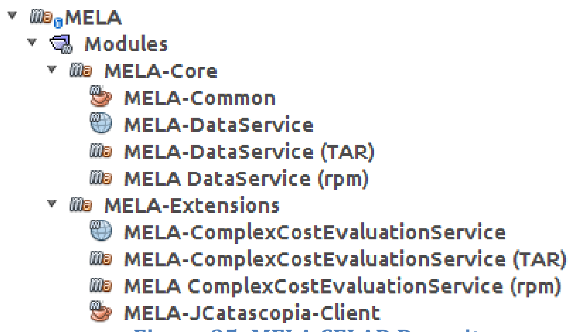
Figure 34: Complex Cost Evaluation of Total Cost

Using our generic cost composition mechanism, a user can monitor from coarse- to fine- grained individual cost components, both in terms of the current cost rate (i.e., speed at which money are spent), and of the total cost.

5.4 Documentation

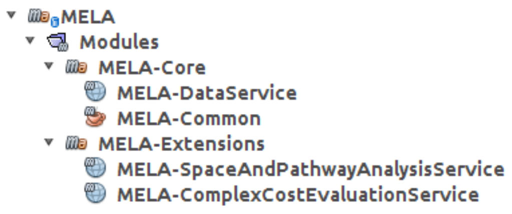
As we promised *flexibility* as a non-functional requirement of the Multi-level Metrics Evaluation Module, we have created for its implementation, MELA, two Github repositories. The first repository, https://github.com/CELAR/multilevel-metrics-evaluation, contains MELA configured to be used within CELAR, with JCatascopia as data source, and at compilation-time it depends on the CELAR distribution project (https://github.com/CELAR/celar-distribution). This repository contains the MELA-JCatascopia-Client implementing the necessary connectors to use the pull and push mechanisms exposed by JCatascopia to retrieve monitoring information (Figure 35). Furthermore, the repository contains corresponding projects to distribute MELA modules as "tarballs" or "rpm" packages.

In the second repository, https://github.com/tuwiendsg/MELA, MELA is configured to run standalone, and does not contain any dependencies towards CELAR modules (Figure 36). By splitting MELA in these two repositories, we make sure a user can easily choose the version s/he desires, either to run MELA in the context of CELAR, or standalone outside CELAR, maximizing MELA's usability.



```
▾ MELA
  ▾ Modules
    ▾ MELA-Core
        MELA-Common
        MELA-DataService
        MELA-DataService (TAR)
        MELA DataService (rpm)
    ▾ MELA-Extensions
        MELA-ComplexCostEvaluationService
        MELA-ComplexCostEvaluationService (TAR)
        MELA ComplexCostEvaluationService (rpm)
        MELA-JCatascopia-Client
```

Figure 35: MELA CELAR Repository



```
▾ MELA
  ▾ Modules
    ▾ MELA-Core
        MELA-DataService
        MELA-Common
    ▾ MELA-Extensions
        MELA-SpaceAndPathwayAnalysisService
        MELA-ComplexCostEvaluationService
```

Figure 36: MELA Standalone Repository

To understand how the two WP4 MELA services, Data Service and Composable Cost Evaluation are used by a MELA user, such as another CELAR Decision Module, or human user, we depict the interaction between the services and users in Figure 37. First, the data sources used to collect monitoring information by the Data Service must be specified (See Section Configure MELA data sources for more details). Then, when starting the Data Service, the MELA user creates a Data Repository, where monitoring information collected and enriched from the data sources is stored. This information is used by the Composable Cost Evaluation service in evaluating cost, together with the cloud services pricing scheme supplied by the MELA user.

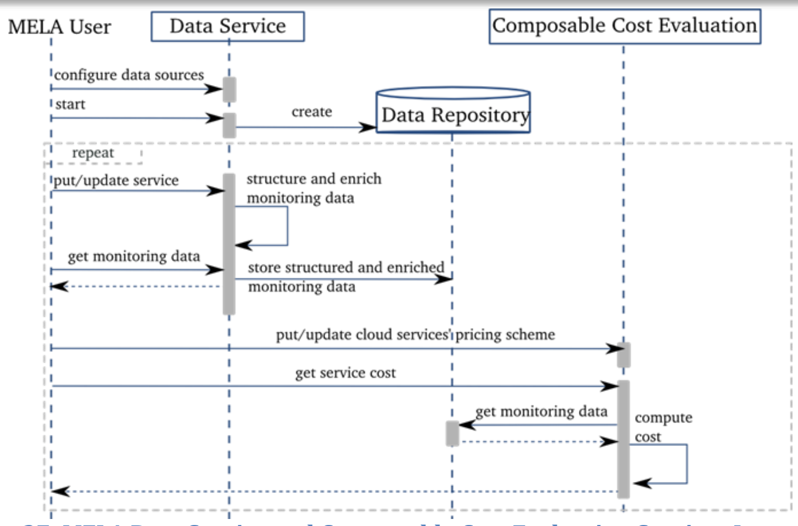


Figure 37: MELA Data Service and Composable Cost Evaluation Services Interaction

5.4.1 MELA-DataService

MELA Data Service is a JAVA Web application which exposes its API using RESTful services. For ease of use, it contains an embedded Tomcat 7 application server, and is compiled in an executable Jar, rather than a WAR file, which can be executed standalone, without the need for a separate application server.

5.4.1.1 Configure MELA data sources

To run MELA Data Service, several configuration steps are needed. First, the MELA data sources must be defined in `./config/dataSources.xml`. Depending on the MELA data source, i.e., JCatascopia pull/push mechanism, or Ganglia pull mechanism, one or more data sources should be added in the configuration file as follows.

For retrieving monitoring data using the push-based mechanism JCatascopia exposed, a data source configuration contains the following parameters: a unique string ID of the data source, the URL and port where the JCatascopia REST API is exposed, and the interval in milliseconds at which monitoring information should be collected:

```
<mela:jcatascopia-poll-datasource id="ID" polling-interval-ms="5000"
 url="http://10.0.0.1/JCatascopia-Web/restAPI" port=”JCatascopiaPort”/>
```

Listing 5: Adding JCatascopia Pull Data Source

In a similar manner, MELA can be used to retrieve monitoring data using the pull-based mechanism Ganglia exposed:

```
<mela:ganglia-datasource id="ID" host="IP" port="GANGLIA_PORT" polling-
interval-ms="milliseconds"/>
```

Listing 6: Adding Ganglia Pull Data Source

Finally, for retrieving monitoring data using the push-based mechanism JCatascopia exposed, a data source configuration contains the following parameters: a unique string ID of the data source, the URL and port where the JCatascopia Noti.Me Push Mechanism web socket listener is exposed:

```
<mela:jcatascopia-push-datasource id="ID"
 url="ws://10.0.0.1/JCatascopia/NotiMe " port=”JCatascopiaNotiMePort”/>
```

Listing 7: Adding JCatascopia Noti.Me Push Data Source

5.4.1.2 *MELA Metrics Composition Rule Language*

In the Composition Rules, MELA represents applications as a service, complex components as topologies, and application components as service units.

Element	**Description**
Resulting Metric	The type (RESOURCE, COST, QUALITY), name and measurement unit of the Metric resulting from the composition rule. *Example*: `<ResultingMetric type="RESOURCE" measurementUnit="ms" name="NAME"/>`
Target Monitored Element Level	At which level in the application structure (SERVICE, SERVICE_TOPOLOGY, SERVICE_UNIT, VIRTUAL_CLUSTER, and VM) the composition rule is applied. *Example*: `<CompositionRule TargetMonitoredElementLevel="SERVICE_UNIT">`
Target Monitored Element ID	If a list of "TargetMonitoredElementID" are specified, this indicates particular elements from the application structure on which the composition rule will be applied.
Composition Operation	A Composition Operation applies an operation over one or more operands, which can be values or metrics.

Operand	a. A “value”: any statically defined value
	Example: `<Operation value="100" type="ADD">`
	
	b. A “ReferenceMetric”, which indicates that the operation will be applied on the value of the ReferenceMetric.
	Example:
	`<Operation MetricSourceMonitoredElementLevel="VM" type="AVG">`
	`<ReferenceMetric type="RESOURCE" name="cpuIdle"/>`

Operation Type	**SUM**: Intended to sum the values retrieved from a “ReferenceMetric” **AVG**: Applied similar to SUM, extracts the average. **MAX / MIN:** Applied similar to SUM, extracts the maximum or minimum value. **DIV/MUL/ADD/SUB:** Divide, Multiply, Subtract From, Add To, the value of the first operand the value of the second operand **SET_VALUE:** Used in creating metric values **CONCAT:** Used to concatenate a series of values **UNION:** Performs set union of values **KEEP/ KEEP_LAST/ KEEP_FIRST**: Extracts one value from a set of values

Sub-Operations	An operation can contain sub-operations, and the values returned by those sub-operations are used by the current operation, in a hierarchical manner.
	
	Example:
	`<Operation value="100" type="ADD">`
	`  <Operation MetricSourceMonitoredElementLevel="VM" type="AVG">`
	`    <ReferenceMetric type="RESOURCE" name="cpuIdle"/>`

Table 9: MELA Composition Rules Language Elements

For ease of use, MELA provides a set of default metric composition rules over standard monitored metrics, which can be applied to any service. The operations are applied only if the source metric is found in the collected monitoring information at the specified level. The following Table presents the default metric composition rules with respect to JCatascopia Monitoring Probes. The same rules are also available for Ganglia, adapted for Ganglia metric names, such as `cpu_idle` instead of `cpuIdle`. The rules below are also applied for each application component having as source VMs, and each complex component, with metric source the components.

Resulting Metric	Source Metric	Operation	Description
cpuUsage	cpuIdle from VMs	100 - AVG(cpuIdle)	Computes average CPU Usage for a Service Unit from CPU Idle collected from the VMs hosting unit instances.
activeConnections	activeConnections	SUM	Computes total number of connections for an HAProxy load balancer
activeSessions	activeSessions	SUM	Similar as above.
requestRate	requestRate	SUM	Similar as above.
responseTime	avgResponseTime	AVG	Computes average response time from the unit's VMs. Applicable to web servers exposing such information.
throughput	throughput	SUM	Similar as above.
avgThroughput	throughput	AVG	Similar as above.
numberOfVMs	-	SET VALUE 1	Enriches the collected monitoring information, for each VM, with a new metric called numberOfVMs, with value 1
numberOfVMs	numberOfVMs	SUM	Computes the number of VMs hosting instances of the service unit. This metric is used as a base for enriching monitoring data with cost.
MB_in	bytesIn bytesReceived	DIV(SUM,1024)	Sums the bytes-in collected from all VMs of a specific unit, and converts the value to Mega Bytes (MB)
MB_out	bytesOut bytesSent	DIV(SUM,1024)	Similar as above.
avgReadLatency	readLatency	AVG	Computes average read latency for each VM hosting the same unit instance.
avgWriteLatency	writeLatency	AVG	Similar as above.
avgLatency	avgReadLatency avgWriteLatency	AVG	Average between average read latency and average write latency

Table 10: MELA Default Composition Rules for JCatascopia Metrics

5.4.1.3 MELA-DataService RESTful API

MELA-DataService exposes a RESTful API (Table 11), in which the application ID or version is marked with the "{serviceID}" parameter. The API enables a user to submit or remove an application (1, 2), add metrics composition rules (3), application requirements (4), and update the application structure after scaling actions (5). The updated structure of the application can be retrieved at each given time using call 6, and the available monitored metrics for the application's VMs using call 7. The application structured monitoring data can be retrieved either as JSON (8), or as XML (9). If monitoring data is required only for a certain application component, it can be retrieved using call (10), by issuing a POST containing as argument the monitored element for which data should be retrieved. Historical monitoring data can also be retrieved using calls 11-13, while 14 and 15 provide methods for accessing the metrics composition rules in JSON or XML for the target application. Methods 16-17 let a user add/remove executing actions on the application, used for display purposes. Method 18 allow a user to test if all metrics coming from a service have values >=0, indicating whether the application is operating properly, useful for determining after a scale action if the application is stable. Finally, call 19 allows a user to list all applications monitored by MELA-DataService.

No.	Method	URL	Input	Output
1.	PUT	/service	application/xml	-
2.	DELETE	/{serviceID}	-	-
3.	PUT	/{serviceID}/metricscompositionrules	application/xml	-
4.	PUT	/{serviceID}/requirements	application/xml	-
5.	PUT	/{serviceID}/structure	application/xml	-
6.	GET	/{serviceID}/structure	-	application/xml
7.	GET	/{serviceID}/metrics	-	application/xml
8.	GET	/{serviceID}/monitoringdata/json	-	application/json
9.	GET	/{serviceID}/monitoringdata/xml		application/xml
10.	POST	/{serviceID}/monitoringdata/xml	application/xml	application/xml
11.	GET	/{serviceID}/historicalmonitoringdata/all/xml	-	application/xml
12.	GET	/{serviceID}/historicalmonitoringdata/ininterval /xml	-	application/xml
13.	GET	/{serviceID}/historicalmonitoringdata/lastX/xml	-	application/xml
14.	GET	/{serviceID}/metriccompositionrules/json	-	application/ json
15.	GET	/{serviceID}/metriccompositionrules/xml	-	application/xml
16.	PUT	/{serviceID}/{targetEntityID}/executingaction/{a ction}	-	application/xml
17.	DELETE	/{serviceID}/{targetEntityID}/executingaction/{a ction}	-	application/xml
18.	GET	/{serviceID}/metricsGreaterEqualThanZero	-	Boolean
19.	GET	/services	-	application/json

Table 11: MELA-DataService REST API

5.4.2 MELA-ComposableCostEvaluationService

MELA ComposableCostEvaluationService is a JAVA Web application which exposes its API using RESTful services. For ease of use, it contains an embedded Tomcat 7 application server, and is compiled in an executable JAR, rather than a WAR file, which can be executed standalone, without the need for a separate application server.

5.4.2.1 MELA-ComposableCostEvaluation RESTful API

MELA- ComposableCostEvaluationService exposes a RESTful API (Table 12), in which the application ID or version is marked with the “{serviceID}” parameter. The API enables a user to submit and update the cloud services’ pricing scheme used in cost evaluation (1-4). The evaluated total cost of the application can be retrieved in JSON, CSV (comma separated values file) and XML, for maximum flexibility, using calls 5-7. Cost computed over a specified time interval is provided by calls 8-9, while calls 10-15 retrieve only cost for specified individual application components. For retrieving the current application monitored snapshot enriched with evaluated cost rate information a user uses calls 16-17, while the total cost overlapped over the current application monitored snapshot is retrieved with calls 18-19.

No.	Method	URL	Input	Output
1.	PUT	*/cloudofferedservice/pricingscheme*	application/xml	-
2.	*GET*	*/cloudofferedservice/pricingscheme*	-	application/xml
3.	*DELETE*	*/{cloudofferedserviceID}/pricingscheme*	-	-
4.	*GET*	*/{cloudofferedserviceID}/pricingscheme*	-	application/xml
5.	*GET*	*/{serviceID}/cost/total/xml*	-	application/xml
6.	*GET*	*/{serviceID}/cost/total/json*	-	application/json
7.	*GET*	*/{serviceID}/cost/total/ csv*	-	application/csv
8.	*GET*	*/{serviceID}/cost/ininterval/{start}/{end}/xml*	-	application/xml
9.	*GET*	*/{serviceID}/cost/ininterval/{start}/{end}/json*	-	application/json
10.	*POST*	*/{serviceID}/cost/total/xml*	application/xml	application/xml

11.	*POST*	*/{serviceID}/cost/total/json*	application/xml	application/ *json*
12.	*POST*	*/{serviceID}/cost/total/csv*	application/xml	application/csv
13.	*POST*	*/{serviceID}/cost/ininterval/{start}/{end}/xml*	application/xml	application/xml
14.	*POST*	*/{serviceID}/cost/ininterval/{start}/{end}/json*	application/xml	application/ *json*
15.	*POST*	*/{serviceID}/cost/ininterval/{start}/{end}/csv*	application/xml	application/csv
16.	*GET*	*/{serviceID}/monitoringdata/cost/rate/json*	-	application/json
17.	*GET*	*/{serviceID}/monitoringdata/cost/rate/xml*	-	application/xml
18.	*GET*	*/{serviceID}/monitoringdata/cost/total/json*	-	application/json
19.	*GET*	*/{serviceID}/monitoringdata/cost/ total /xml*	-	application/xml

Table 12: MELA-ComposableCostEvaluation API

5.5 Evaluation

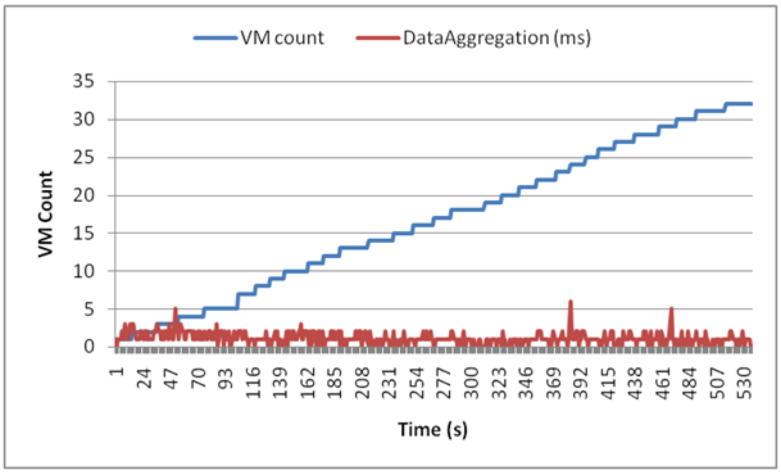
This section provides an evaluation report for the CELAR Multi-Level Metrics Evaluation Module based on the requirements defined in D1.2 (R4.13-R4.19).

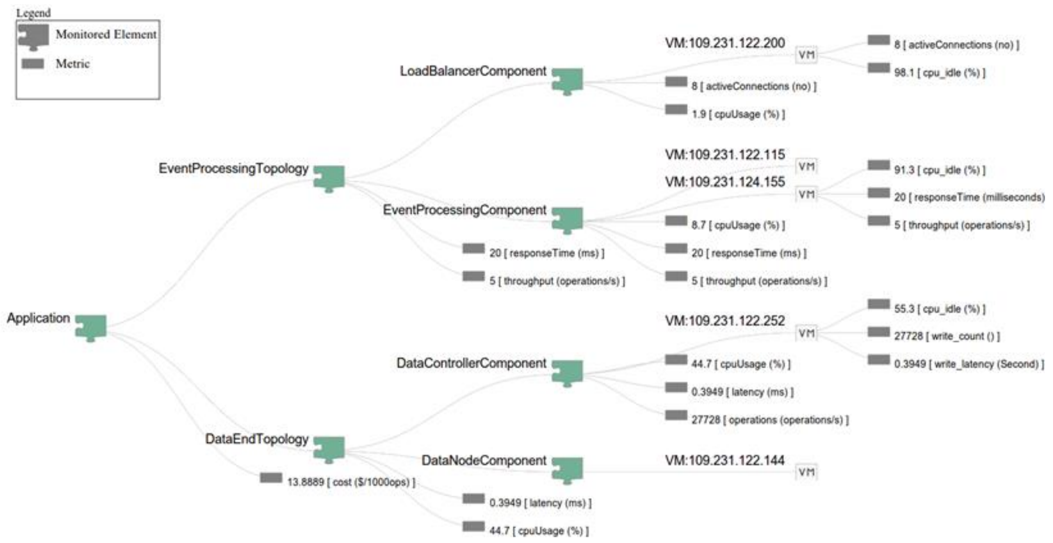
Req. No.	R14	Convert Monitoring Data
Status	DONE	
Means of Assessment/Verification		
Using *metric composition rules*, collected metrics can be converted to desired measurement units. We have defined an XML-based domain specific language for describing metric composition rules, which can be used at different ends, including converting monitoring information. For example, using the composition rule below a user converts a throughput monitored in operations per second (ops/s) to thousands of operations per second (thousand/ops/s) by dividing the monitored value with 1000.		

```
<CompositionRule TargetMonitoredElementLevel="COMPONENT">
    <ResultingMetric type="RESOURCE" measurementUnit="thousand/ops/s"
                                                         name="numberOfVMs"/>
        <Operation MetricSourceMonitoredElementLevel="VM" type="DIV"
                                                         VALUE=”1000”>
            <ReferenceMetric type="RESOURCE" name="throughput"
                                                   measurementUnit=”ops/s” />
        </Operation>
</CompositionRule>
```

Req. No.	R15	Aggregate Monitoring Data Based Upon the Application Structure Model
Status	DONE	
Means of Assessment/Verification		
The monitoring information collected by the CELAR Monitoring System is logically organized after the application structure. The developed language for specifying metric composition rules allows the definition of any metric at any of the application levels (component, complex component, application), based on metrics from the same or different levels, and even insert new metrics. This was evaluated by deploying MELA-DataService on different cloud infrastructures (TUW private cloud, and Flexiant), to aggregate information retrieved from JCatascopia for different service structures, on CELAR pilot applications, and on an elastic service for M2M platform[19] developed by TUW.		

[19] https://github.com/tuwiendsg/DaaSM2M

Req. No.	R13	Perform Composable Cost-Evaluation
Status	DONE	
Means of Assessment/Verification		
Using *metric composition rules*, cost for the entire application or individual components can be obtained by combining cloud pricing schemes with virtual infrastructure usage information collected by CELAR Monitoring System. Moreover, we have developed a mechanism for evaluating more complex cost based on cloud pricing schemes, described in this deliverable in Section Cost Evaluation Based on Complex Cloud Pricing Schemes.		
Req. No.	R16	Scalability
Status	DONE	
Means of Assessment/Verification		
As elastic applications can scale out “infinitely”, the module must be able to handle various amounts of incoming data. For testing scalability we have used MELA Data Service to aggregate monitoring information for an elastic service for the aforementioned M2M platform developed by TUW, and deployed on Flexiant. To assess also the impact of application structure complexity on MELA’s performance we have evaluated first a simple structure, the application having all VMs hosting instances of the same component. In the second evaluation we used a more complex structure, the application having two complex components, each in turn containing 2 components. We have scaled the application from 0 to 32 VMs (1 to 4 VMs per component in the last scenario), each VM reporting 39 metrics. We used a total of 20 metric composition rules at application, complex component, and component levels, sufficient to aggregate information about application resource performance (e.g., CPU usage), data transfer, and application performance (throughput, response time). MELA was deployed on a machine having an I5 Intel CPU processor at 2.8 GHz, with 6GB of RAM and SSD storage. Focusing on the results, we observed that in both evaluated scenarios, the processing time for a monitoring snapshot did not exceed 5 milliseconds, as depicted in Figure 38 for the second scenario. These results indicate that MELA can handle increasing amounts of monitoring data without any performance degradation. Moreover, given the low data processing time, it could handle even larger cloud applications.  Figure 38: MELA Data Aggregation Scalability		
Req. No.	R17	Robustness

Status	DONE	
Means of Assessment/Verification		
During the enforcement of scaling actions, incoming monitoring data might not be accurate, or data sources might be missing. This was demonstrated by using MELA for the aforementioned M2M elastic service and selectively deactivating the CELAR pilot applications' monitoring system (i.e. JCatascopia). Below we show a snapshot of the MELA DataService GUI depicting the instant application monitoring information. In this case there is no monitoring information retrieved for two VMs, as their monitoring system was disabled. Thus, MELA can handle situations in which for an application component one VM reports monitoring information, while another does not.  Legend Monitored Element Metric Application EventProcessingTopology LoadBalancerComponent VM:109.231.122.200 VM 8 [activeConnections (no)] 98.1 [cpu_idle (%)] 8 [activeConnections (no)] 1.9 [cpuUsage (%)] EventProcessingComponent VM:109.231.122.115 VM VM:109.231.124.155 VM 91.3 [cpu_idle (%)] 20 [responseTime (milliseconds) 5 [throughput (operations/s)] 8.7 [cpuUsage (%)] 20 [responseTime (ms)] 5 [throughput (operations/s)] 20 [responseTime (ms)] 5 [throughput (operations/s)] DataEndTopology 13.8889 [cost ($/1000ops)] DataControllerComponent VM:109.231.122.252 VM 55.3 [cpu_idle (%)] 27728 [write_count ()] 0.3949 [write_latency (Second)] 44.7 [cpuUsage (%)] 0.3949 [latency (ms)] 27728 [operations (operations/s)] DataNodeComponent VM:109.231.122.144 VM 0.3949 [latency (ms)] 44.7 [cpuUsage (%)] Figure 39: MELA Data Service with Incomplete Monitoring Information		
Req. No.	R18	Reusability
Status	DONE	
Means of Assessment/Verification		
The module must support any cloud pricing scheme format and any cloud application structure format. This was demonstrated by submitting multiple application structures to be applied both over the same virtual infrastructure topology (e.g. VMs), out of which we present in the following two structures. In the first structure the application consists of a set of components, each having associated VMs (Figure 40). The second structure used to aggregate monitoring information over the same virtual infrastructure is more complex (Figure 41), containing components grouped in more complex components denoted as topologies. By giving the MELA user the possibility to impose any logical structure on top of the virtual infrastructure used to run its applications, MELA provide a high degree of reusability, as it can support different users with different views over their applications.		

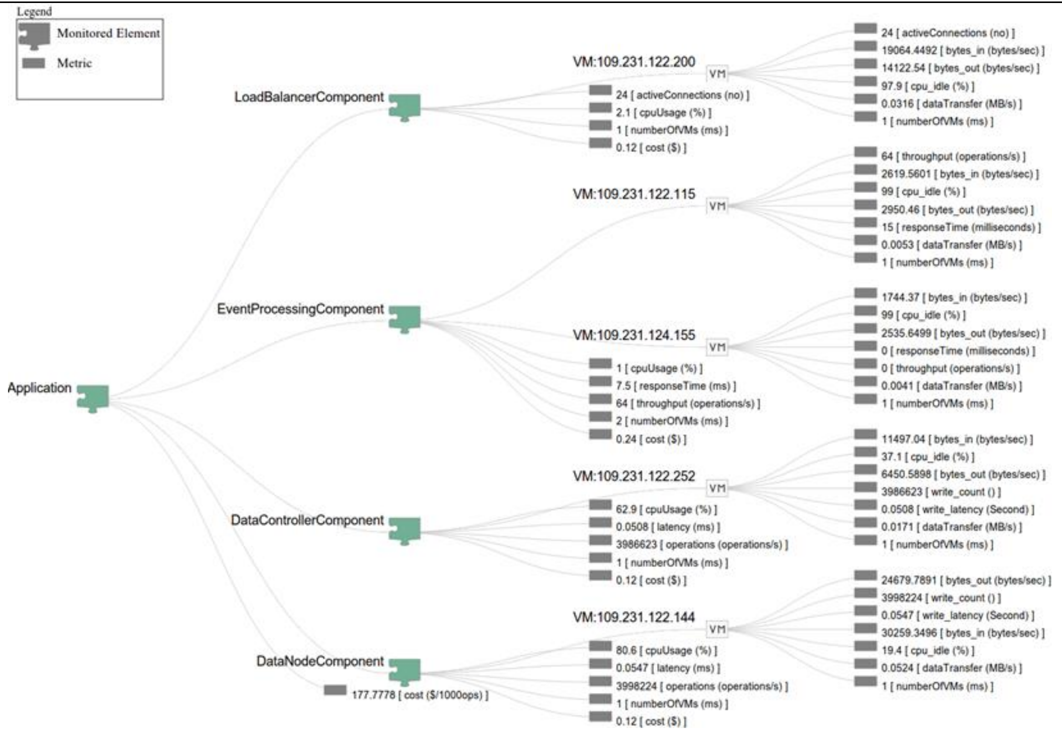

Legend
Monitored Element
Metric
Application
LoadBalancerComponent
EventProcessingComponent
DataControllerComponent
DataNodeComponent
177.7778 [cost ($/1000ops)]
VM:109.231.122.200
24 [activeConnections (no)]
2.1 [cpuUsage (%)]
1 [numberOfVMs (ms)]
0.12 [cost ($)]
24 [activeConnections (no)]
19064.4492 [bytes_in (bytes/sec)]
14122.54 [bytes_out (bytes/sec)]
97.9 [cpu_idle (%)]
0.0316 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
VM:109.231.122.115
64 [throughput (operations/s)]
2619.5601 [bytes_in (bytes/sec)]
99 [cpu_idle (%)]
2950.46 [bytes_out (bytes/sec)]
15 [responseTime (milliseconds)]
0.0053 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
VM:109.231.124.155
1 [cpuUsage (%)]
7.5 [responseTime (ms)]
64 [throughput (operations/s)]
2 [numberOfVMs (ms)]
0.24 [cost ($)]
1744.37 [bytes_in (bytes/sec)]
99 [cpu_idle (%)]
2535.6499 [bytes_out (bytes/sec)]
0 [responseTime (milliseconds)]
0 [throughput (operations/s)]
0.0041 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
VM:109.231.122.252
62.9 [cpuUsage (%)]
0.0508 [latency (ms)]
3986623 [operations (operations/s)]
1 [numberOfVMs (ms)]
0.12 [cost ($)]
11497.04 [bytes_in (bytes/sec)]
37.1 [cpu_idle (%)]
6450.5898 [bytes_out (bytes/sec)]
3986623 [write_count ()]
0.0508 [write_latency (Second)]
0.0171 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]
VM:109.231.122.144
80.6 [cpuUsage (%)]
0.0547 [latency (ms)]
3998224 [operations (operations/s)]
1 [numberOfVMs (ms)]
0.12 [cost ($)]
24679.7891 [bytes_out (bytes/sec)]
3998224 [write_count ()]
0.0547 [write_latency (Second)]
30259.3496 [bytes_in (bytes/sec)]
19.4 [cpu_idle (%)]
0.0524 [dataTransfer (MB/s)]
1 [numberOfVMs (ms)]

Figure 40: Application Structure Containing only Components

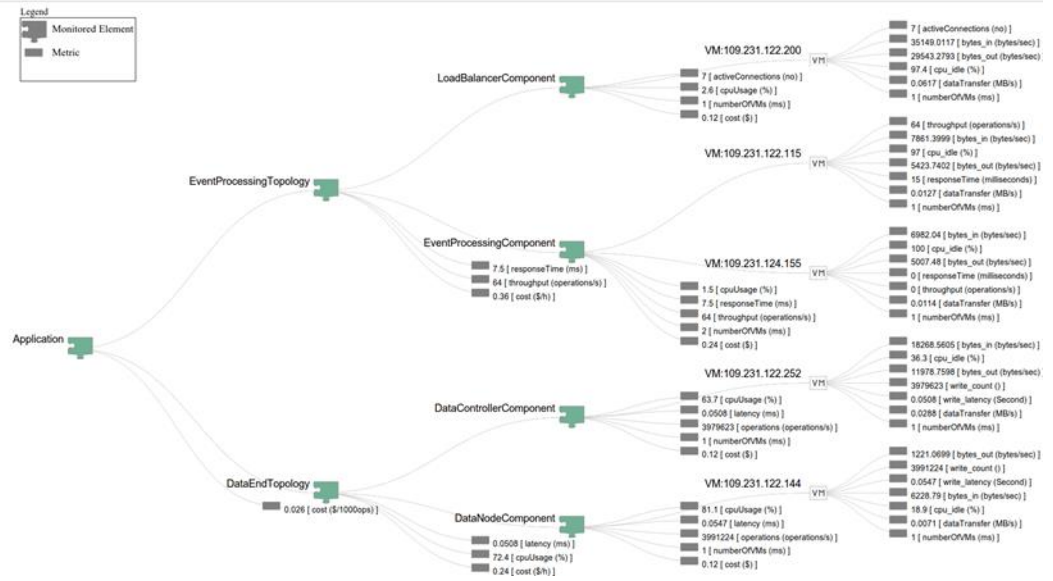


Figure 41: Application Structure Containing Complex Components and Components

Req. No.	R19	Flexibility
Status	DONE	
Means of Assessment/Verification		
The module must be configurable and usable outside CELAR, as to maximize exposure of CELAR modules. A plug-in for collecting monitoring information from Ganglia was developed, and tests were carried with Ganglia as monitoring data source, enabling MELA to run outside CELAR Platform. Moreover, as JCatascopia can be used standalone outside CELAR Platform, MELA-Data Service can also operate in conjunction with JCatascopia outside CELAR, further increasing the flexibility of using MELA.		

Table 13: Multi-Level Metric Evaluation Module Requirement Evaluation

6 Conclusions and Future Work

In this deliverable, we have provided a detailed description of the second version of the Cloud Information and Performance Monitor layer by documenting the current design and implementation status of the three components which comprise this layer: (i) the CELAR Monitoring System; (ii) the Multi-Level Metrics Evaluation Module; and (iii) the CELAR Information System. Specifically, we have provided a study of the State-of-the-Art in Cloud Information Systems and an updated study of the State-of-the-Art in Cloud Monitoring and Cost-Evaluation Systems. Furthermore, we have introduced the first version of the CELAR Information System, where we have presented its requirements, the challenge which it must address and we have provided a detailed description of its design and implementation specifics of its internal modules. Moreover, we have presented the second and final versions of the CELAR Monitoring System and the Multi-Level Metrics Evaluation Module where emphasis was given in introducing new features as well as improvements made since v1.0 which was presented in D4.1.

Finally, special focus was also given in presenting an evaluation report on how the requirements for the two aforementioned components have been verified to see if they facilitate their purposes.

As Task 4.1 (Monitoring Tool) shortly concludes (Month 26), we will devote our time on improving the internals modules of the CELAR Monitoring System by focusing on fixing bugs as well as enhancing fault-tolerance, scalability and availability. Furthermore, a number of application-specific Monitoring Probes will be developed to facilitate the needs of not only the CELAR pilot applications but also other interested cloud application stakeholders interested in utilizing our monitoring system. Finally, a website will be developed to host documentation, tutorials, artifacts and support of the JCatascopia Monitoring System prototype.

Furthermore, as Task 4.2 (Composable Cost Evaluation) has recently been completed (Month 18), with respect to Multi-Level Metric Evaluation from now on we will focus on integration with other CELAR Modules in the context of WP6. As the Composable Cost Evaluation service is a new addition to WP4, we will focus on its integration with the CELAR Decision Module (WP5). Moreover, given that Task 4.3 (Cloud Information System) will continue until the end of the project, effort will be spent on integration with Cloud Information System (WP4). Additionally, we aim to remove bugs and improve performance of the Multi-Level Metrics Evaluation Module where required, based on experience gathered from using it in conjunction with the other CELAR modules, namely CELAR Decision Module (WP5) and CELAR Information System (WP4).

In the first months of the lifespan of Task 4.3 (Cloud Information System) we defined its requirements and designed the first version of our system. The v1.0 of CELAR IS includes most of the functionality, but in a very early stage. For the upcoming year we are going to work on the analytics engine of CELAR IS in order to be able to provide more insightful cloud usage analytics for the user. In that point of view, we will enrich the calculation of the analytics with more cost and topology specific information. We will also work on the 'comparisons' feature presented earlier, we will define what is useful for the user to see under this section and we will improve the UI for viewing this information. Finally, we will further evaluate the performance and the efficiency of our system.

7 References

[4CaaSt] EU-funded 4CaaSt Project. http://www.4caast.eu/

[Al-Hazmi 2012] Yahya Al-Hazmi, Konrad Campowsky, and Thomas Magedanz. A monitoring system for federated clouds. In Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on, pages 68–74, Nov 2012.

[BonFIRE] EU-funded BonFIRE Project. http://www.bonfire-project.eu/

[Amazon AWS] Amazon Web Services http://aws.amazon.com/

[AmazonEBS] http://aws.amazon.com/ebs/

[AmazonEC2] http://aws.amazon.com/ec2/

[AWS Advisor] AWS Trusted Advisor https://aws.amazon.com/premiumsupport/trustedadvisor/

[cache2k] http://cache2k.org/

[Cassandra] Apache Cassandra Project, http://cassandra.apache.org/

[Ceilometer] OpenStack Ceilometer. https://wiki.openstack.org/wiki/Ceilometer

[CentOS] http://www.centos.org/

[Charts] https://developers.google.com/chart/

[Clayman 2011] Clayman, S.; Galis, A.; Mamatas, L., "Monitoring virtual networks with Lattice". Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP, vol., no., pp.239,246, 19-23 April 2010

[CloudTrail] AWS CloudTrail http://aws.amazon.com/cloudtrail/

[CloudWatch] http://aws.amazon.com/cloudwatch/

[CloudCheckr] CloudCheckr http://cloudcheckr.com/

[Cloudyn] Cloudyn http://www.cloudyn.com/

[CloudAware] CloudAware http://www.cloudaware.com/

[Cloudvertical] Cloudvertical https://cloudvertical.com/

[Cloudability] Cloudability https://cloudability.com/

[CloudScreener] CloudScreener http://www.cloudscreener.com/

[Cloudorado] Cloudorado http://www.cloudorado.com/

[Commons Math] http://commons.apache.org/proper/commons-math/

[Cooper 2010] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears. “Benchmarking Cloud serving systems with YCSB” *In the proceedings of the 1st ACM symposium on Cloud computing (SoCC 2010)*. ACM, New York, NY, USA, pages 143-154.

[Copil 2014] G. Copil and D. Trihinas and H.L Truong and D. Moldovan and G. Pallis and S. Dustdar and M. D. Dikaiakos, "ADVISE -- a Framework for Evaluating Cloud Service Elasticity Behavior" in *12th International Conference on Service Oriented Computing*, ICSOC 2014, Paris, France 2014.

[D1.2] D1.2: Updated User Requirements and System Architecture, CELAR project, FP7-317790

[D2.1] D2.1: Application Description Tool V1, CELAR project, FP7-317790

[D2.2] D2.2: Application Description Tool and Application Submission Tool V1, CELAR project, FP7-317790

[D3.2] D3.2: Elasticity Provisioning Platform V1, FP7-317790

[D4.1] D4.1: Cloud Monitoring Tool V1, CELAR project, FP7-317790

[Ehcache] http://ehcache.org/

[Google Compute Engine] Google Compute Engine https://cloud.google.com/products/compute-engine/

[Garg 2011] Garg, Saurabh Kumar, Steven Versteeg, and Rajkumar Buyya. "Smicloud: A framework for comparing and ranking cloud services." Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on. IEEE, 2011.

[Guava] https://code.google.com/p/guava-libraries/

[Flexiant] http://www.flexiant.com/flexiant-cloud-orchestrator/

[Han 2010] Han, Taekgyeong, and Kwang Mong Sim. "An ontology-enhanced cloud service discovery system." Proceedings of the International MultiConference of Engineers and Computer Scientists. Vol. 1. 2010.

[Huebscher 2008] Markus C. Huebscher, Julie A. McCann. “A survey of autonomic computing – degrees, models, and applications”, ACM Comput. Surv., 40(3):7:1–7:28, August 2008

[Jersey] https://jersey.java.net/

[Kang 2011] Kang, Jaeyong, and Kwang Mong Sim. "Cloudle: an ontology-enhanced cloud service search engine." Web Information Systems Engineering–WISE 2010 Workshops. Springer Berlin Heidelberg, 2011.

[Li 2010] Li, Ang, et al. "CloudCmp: comparing public cloud providers." Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, 2010.

[Microsoft Azure] Microsoft Azure https://azure.microsoft.com/en-us/

[MistIO] Mist.io https://mist.io/

[Moldovan 2013] D. Moldovan, G. Copil, H.-L. Truong, S. Dustdar, "MELA: Monitoring and Analyzing Elasticity of Cloud Services", in *5th International Conference on Cloud Computing*, (CloudCom 2013), Bristol, UK, 2013

[HttpClient] http://hc.apache.org/httpclient-3.x/

[Moldovan 2014] D. Moldovan, G. Copil, H.-L. Truong, S. Dustdar, "QUELLE -- a Framework for Accelerating the Development of Elastic Systems", in Third European Conference on Service-Oriented and Cloud Computing (ESOCC 2014,) 2-4 September, Manchester, UK, 2014

[mOSAIC] EU-funded mOSAIC Project. http://www.mosaic-cloud.eu/

[MODAClouds] EU-funded MODAClouds Project. http://www.modaclouds.eu/

[MySQL] http://www.mysql.com/

[Okeanos] GRNET Okeanos Public Cloud, https://okeanos.grnet.gr/home/

[OpenStack] http://www.openstack.org/

[Optimis] EU-funded Optimis Project. http://www.optimis-project.eu/

[Papadopoulos 2013] Andreas Papadopoulos, George Pallis, Marios D. Dikaiakos, "Identifying Clusters with Attribute Homogeneity and Similar Connectivity in Information Networks", *2013 IEEE/WIC International Conference on Web Intelligence and Intelligent Agent Technology,* Atlanta, GA, USA, 2013

[Quoc 2014] Do Le Quoc, Lenar Yazdanov, and Christof Fetzer. Dolen: User-side multi-cloud application monitoring. In Future Internet of Things and Cloud (to appear). IEEE, 2014.

[RackSpace] http://www.rackspace.com/

[Rak 2011] M. Rak, S. Venticinque, T. Mahr, G. Echevarria, and G. Esnal. "Cloud Application Monitoring: The mOSAIC Approach." In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pages 758–763, Nov 2011.

[Sánchez] Germán Sánchez, Albert Samà, Francisco J. Ruiz, and Núria Agell. 2010. Moving Intervals for Nonlinear Time Series Forecasting. In Proceedings of the 2010 conference on Artificial Intelligence Research and Development: Proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence, René Alquézar, Antonio Moreno, and Josep Aguilar (Eds.). IOS Press, Amsterdam, The Netherlands, The Netherlands, 217-225.

[Sofokleous 2014] C. Sofokleous and N. Loulloudes and D. Trihinas and G. Pallis and M. Dikaiakos, "c-Eclipse: An Open-Source Management Framework for Cloud Applications", (EuroPar 2014), Porto, Portugal 2014.

[Teevity] Teevity http://www.teevity.com/

[Trihinas, 2014a] D. Trihinas, G. Pallis, M. D. Dikaiakos, "JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud", in *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing,* (CCGRID 2014), Chicago, IL, USA, 2014

[Trihinas 2014b] D. Trihinas and C. Sofokleous and N. Loulloudes and A. Foudoulis and G. Pallis and M. D. Dikaiakos, “Managing and Monitoring Elastic Cloud Applications” in *14th International Conference on Web Engineering,* (ICWE 2014), Toulouse, France, 2014

[Uriarte 2014] Rafael Brundo Uriarte and Carlos Becker Westphall. Panoptes: A monitoring architecture and framework for supporting autonomic clouds. In Network Operations and Management Symposium (NOMS), 2014 IEEE, pages 1–5, May 2014.

[Websockets] W3C Sockets API, http://www.w3.org/TR/2009/WD-websockets-20091222/

[Xiang 2010] Guofu Xiang, Hai Jin, Deqing Zou, Xinwen Zhang, Sha Wen, and Feng Zhao. Vmdriver: A driver-based monitoring mechanism for virtualization. In Reliable Distributed Systems, 2010 29th IEEE Symposium on, pages 72–81, Oct 2010.

[Zabbix] http://www.zabbix.com/