Grant Agreement N° FP7-318484

Title:	*Monitoring Platform Final Release*
Authors:	*Giuliano Casale, Weikun Wang (Imperial), Marco Miglierina (Polimi), Victor Ion Munteanu (IEAT)*
Editor:	*Marco Miglierina (Polimi), Weikun Wang (Imperial)*
Reviewers:	*Stepan Seycek (BOC), Craig Sheridan (Flexi)*
Identifier:	*Deliverable # D6.3.2*
Nature:	*Prototype*
Version:	*1.0*
Date:	*10/04/2014*
Status:	*Final*
Diss. level:	*Public*

Executive Summary

This prototype deliverable presents the final release of the MODAClouds monitoring platform. The platform follows the specifications given in Deliverable D6.2 *"Monitoring Platform Specification"*. A set of data collectors, deterministic data analysers, and statistical data analysers has been implemented and validated. For each of these components, the deliverable provides a high-level overview of highlights and refers to the GitHub repository for detailed information.

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 318484 (MODAClouds).

Members of the MODAClouds consortium:

Politecnico di Milano	Italy
Stiftelsen Sintef	Norway
Institutul E-Austria Timisoara	Romania
Imperial College of Science, Technology and Medicine	United Kingdom
SOFTEAM	France
Siemens Program and System Engineering	Romania
BOC Information Systems GMBH	Austria
Flexiant Limited	United Kingdom
ATOS Spain S.A.	Spain
CA Technologies Development Spain S.A.	Spain

Published MODAClouds documents

These documents are all available from the project website located at http://www.modaclouds.eu/
Software releases are available for download at http://www.modaclouds.eu/software/public-deliverables/

Contents

Summary

1 Introduction

1.1 Context and Objectives

The MODAClouds Monitoring Platform is part of the MODAClouds Runtime Environment shown in Figure 1. Its main capabilities have been described in deliverable D6.2 and can be summarised as follows:

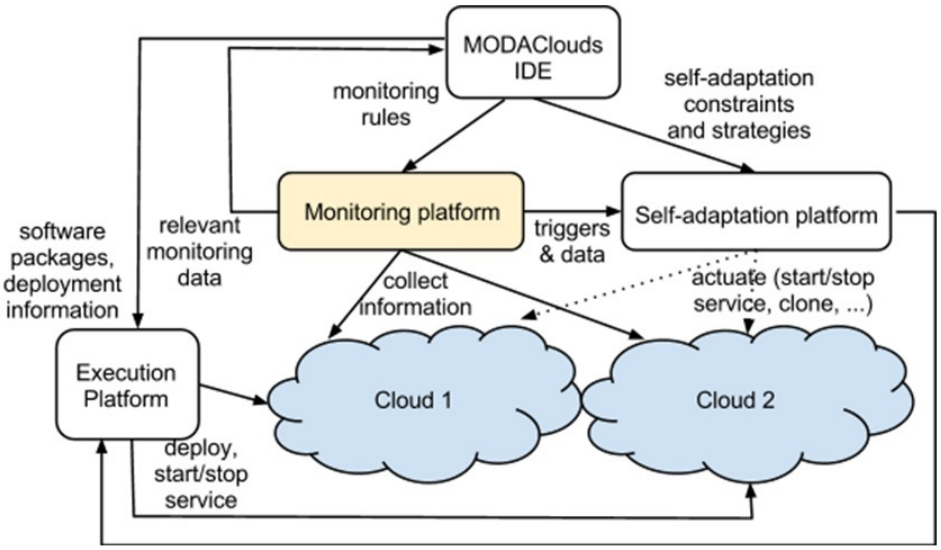

Figure 1: MODAClouds Runtime Environment

- *Monitoring*. The monitoring component gathers information at different levels to evaluate the quality-of-service (QoS) of the system. In the Monitoring Platform, both the current consumption of system resources (such as CPU and Memory) and other QoS metrics (e.g., response time of a request) are captured.
- *Analysis*. The analysis component processes the information gathered by the monitoring collectors and generates high-level statistics from it. For instance, it may analyse the correlation between different metrics to infer the cause of a given error in the system. It also performs statistical inference to estimate the parameters needed for the runtime QoS management models used in the Self-Adaptation Platform, see deliverable D3.2.2 for an integrated architectural view.
- The Monitoring Platform receives a set of monitoring rules from the MODAClouds IDE, which defines what metrics to collect and how to measure them, such as the monitoring time granularity. Then, the Monitoring Platform monitors them and provides triggers and data streams to the Self-Adaptation Platform and sends feedback to the design-time IDE.
- The Monitoring Platform deals with two main actors: the Self-Adaptation Platform and the MODAClouds IDE. The Self-Adaptation Platform is responsible for observing the monitoring data and statistics and deciding at runtime for corrective actions that can improve QoS, e.g., the deployment of new virtual machines to scale out. MODAClouds end users include developers and cloud system administrators, who will interact with the monitoring platform through graphical user interface (GUI, part of the MODAClouds IDE) or command-line interface (CLI). A typical usage scenario is a system administrator who wants to visualize the historical performance offered by a cloud application. A comprehensive description of actors and use case requirements is available in deliverable D6.1.

The main features offered in the monitoring platform are overviewed in the next subsection.

1.1.1 Overview of the Monitoring Platform

Figure 2 gives an overview of the monitoring platform components and their interaction with the rest of the platform. The main features of the components in this release are summarised below.

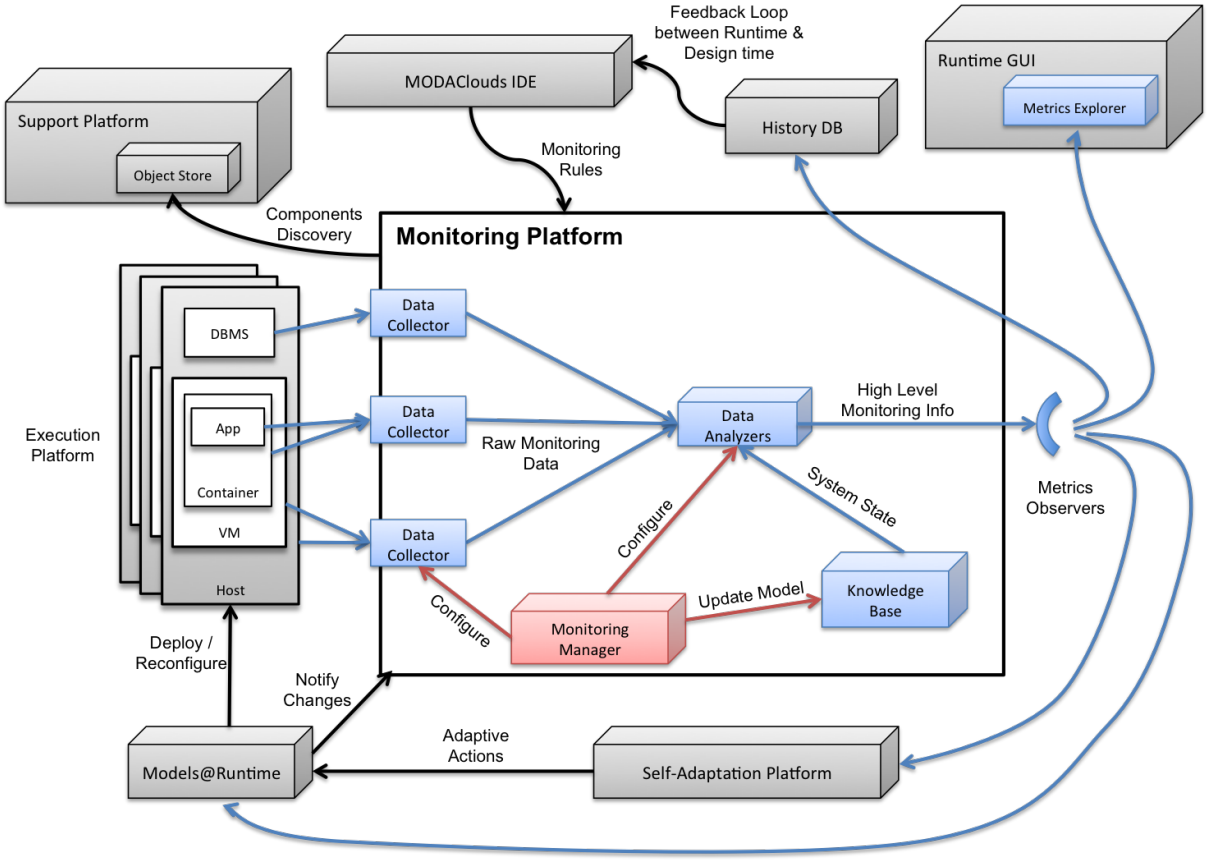

Figure 2: Monitoring Platform Architecture

The Monitoring Rule (MR). Monitoring rules are specifications that are installed in the Monitoring Platform and describe how incoming data has to be processed, what conditions have to be verified and what output should be produced. For a detailed description of Monitoring Rules, please refer to the following links:

v1.0: https://github.com/deib-polimi/modaclouds-qos-models/blob/v1.0/doc/user-manual.md
Latest version: https://github.com/deib-polimi/modaclouds-qos-models/blob/master/doc/user-manual.md

In the whole deliverable we will refer to *v1.0* and *latest version*. v1.0 is the version released for the current deliverable. In order to see any following update due to integration, bug fixing and improvements refer to the latest version.

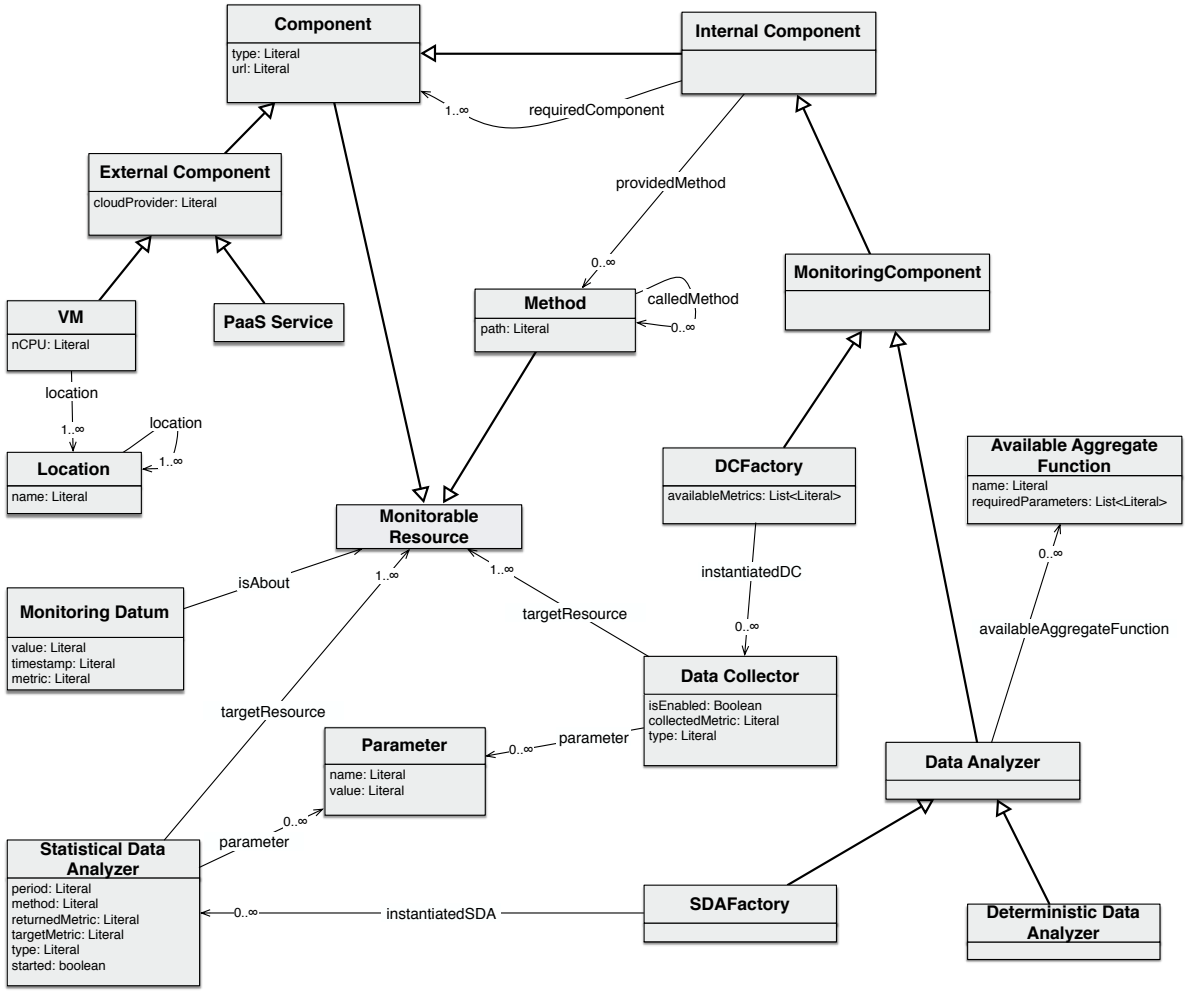

Figure 3: The Monitoring Ontology

Monitoring Manager (MM). The monitoring manager is the main coordinator of the platform. It is responsible for installing monitoring rules, configuring monitoring components, attaching external observers to requested metrics and keeping the knowledge base up to date through the interaction with Models@Runtime. The monitoring manager is the main interface towards external components.

Knowledge Base (KB). The knowledge base is the component where all persistent information about the whole system deployment (cloud provider platform, MODAClouds platform and hosted application components) and its configuration is stored. The ontology describing components and their relationships is stored in the KB and it is used to interpret monitoring data. The representation of MODAClouds Monitoring ontology is shown in Figure 3. In this ontology the whole system deployment as well as components configuration is described. The ontology has been recently adapted to the ongoing development of CloudML2.0 [1] which is currently synchronizing with TOSCA [2]. Here we give a short description of the main entities:

- a *Component* is whatever running artifact that can be located with a URL. The type property is used so to describe the different types of instances (e.g.: FrontendVM, BackendVM) according to the MODACloudsML language;
- *External Components* are those components that are offered by the cloud provider;

- *Internal Components* are those components that are hosted on external components;
- internal components provide *Methods* which are identified by paths (e.g.: /login);
- *SDA* and *DC Factories* are the artifacts that are responsible of instantiating *Statistical Data Analyzers* and *Data Collectors*;
- both components and methods are *Monitorable Components*, i.e., they can be monitored by the platform.

During runtime, whenever an instance of any component is deployed by the execution platform, this event will be notified to the monitoring manager which will create a corresponding instance in the KB.

Data Collectors (DCs). DCs are responsible for collecting monitoring data from cloud resources and applications and sending it to the data analyzers. Based on the technology they use to collect data, they can be either located on the same machine where the resource they are monitoring is or on a different machine. Data Collectors configuration is stored in the knowledge base. Each collector is responsible of periodically pulling its configuration and configure accordingly.

Data Analyzers (DAs). DAs contribute the core of the monitoring platform. They are configured by the monitoring manager based on high level monitoring rules. Data analyzers are in charge of aggregating raw monitoring data into higher level monitoring information, detect violations, and output such data to observers. The monitoring platform features two different kinds of data analysers (DAs): Deterministic Data Analyser (DDA) and Statistical Data Analyser (SDA). They are executed in two different runtime containers and serve different roles.

- The DDA processes at high-speed the RDF monitoring data coming from the DCs, interpret, filter and aggregate it based on monitoring rules and on system state stored in the KB. Moreover, it detects violations based on constraints described in the monitoring rules and serialize data to be delivered to observers. However, the core of the DDA is an RDF stream processor (C-SPARQL engine) which is currently able to perform very simple aggregations such as the average, count, min, max. It has been recently extended (as a requirement of the MODAClouds project) to compute the percentile. Therefore, more complex analysis are delegated to SDAs.
- SDAs extract *hidden information* from the data using statistical methods and generate *predictions*. An example of hidden information estimated by SDAs are the resource requirements of individual requests when only aggregate measurements are available. The SDAs are classified in the following categories: *Correlation* SDAs, that calculate the correlation between monitoring data (e.g., between a web server arrival traffic and traffic consequently generated onto the database tier); *Estimation* SDAs, which estimate QoS metrics of the system that cannot be directly measured (e.g., resource consumption of individual requests corrected to ignore contention); *Forecasting* SDAs, which forecast the trends of selected metrics using statistical methods. SDAs are contained in the MATLAB Compiler Runtime (MCR). The MCR is a royalty-free container and no MATLAB license is required to execute SDAs. With minor changes, most SDAs should be able to run also inside GNU Octave.

Whenever a monitoring rule requires an aggregate function which can be computed only by an SDA, the monitoring manager configures the DDA so to deliver specific data to the SDA. The SDA will compute the aggregation and will send the aggregated data back to the DDA for further analysis on constraint violation or to be delivered to external observers.

Data Model. The monitoring data as well as the KB information is encoded in Resource Description Framework (RDF) format. RDF [3] organizes data into subject-predicate-object triples. Other data formats may be adopted, however RDF is a de-facto standard for semantically annotated data and it has been adopted in this implementation; it can be processed by RDF stream processors such as the C-SPARQL engine featured by the MODAClouds monitoring platform. A representation of a monitoring datum, encoded as RDF graph, is shown in Figure 4.

1.2 Structure of this document

The rest of the document is composed of three sections. Section 2 introduces the architecture and the interfaces of the Monitoring Platform. Section 3 gives the description of the components of the Monitoring Platform. Section 4 summarizes the monitoring metrics supported by the platform.

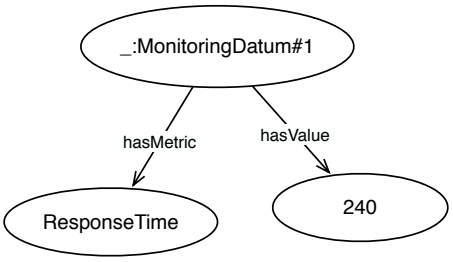

Figure 4: RDF graph representation of a monitoring datum.

2 The Architecture

In Figure 5 we describe in detail the architecture of the monitoring platform, together with the exposed and required interfaces. In this Chapter we mainly describe and give pointers to all interfaces exposed and required by the monitoring platform and internal interfaces among monitoring components, and to any library that has been implemented to access these interfaces.

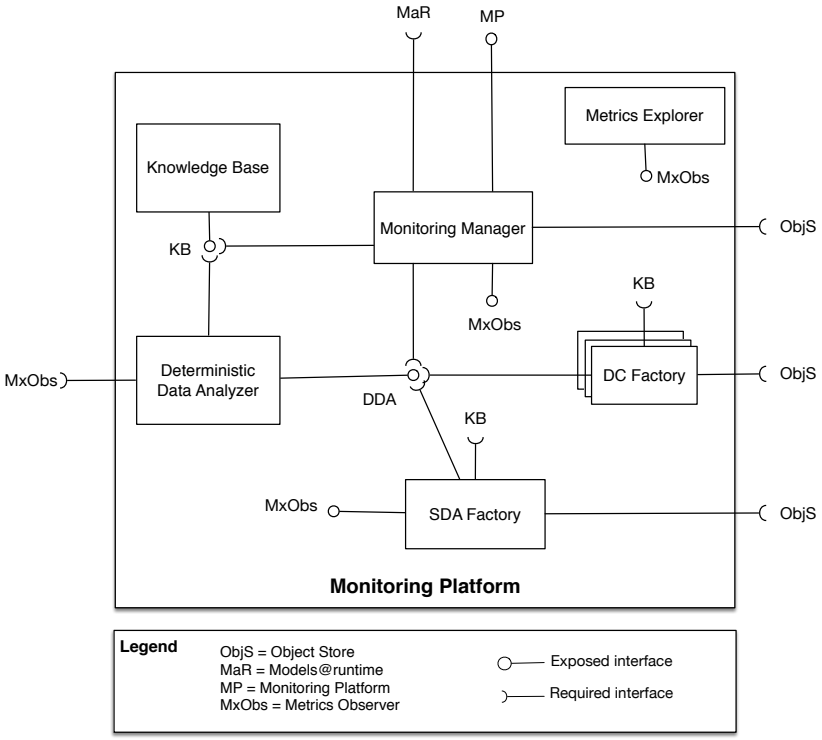

Figure 5: Monitoring Platform Architecture and Interfaces

2.1 External Interfaces

The external interface of the Monitoring Platform is the one exposed by the Monitoring Manager component, and is available at the following links:

v1.0: https://github.com/deib-polimi/modaclouds-monitoring-manager/blob/v1.0/doc/api.md#rest-api
Latest version: https://github.com/deib-polimi/modaclouds-monitoring-manager/blob/master/doc/api.md#rest-api

Only REST interfaces for managing monitoring rules and attaching observers were defined and implemented in v1.0. Interfaces for notifying the monitoring manager about new instances are part of the integration with Models@runtime due at M24 and only Java APIs were implemented for testing purposes in this deliverable.

2.2 Required Interfaces

2.2.1 Models@runtime

The required interface towards Models@runtime will be defined during the integration with Models@runtime, due at M24. It will be an interface for retrieving the current deployment configuration and will be available at the following link:

Latest version: https://github.com/deib-polimi/modaclouds-monitoring-manager/blob/master/doc/required-if.md#modelsruntime

2.2.2 Metrics Observers

A metrics observer is any component that registers to the monitoring platform in order to receive specific data about a metric. An observer can only register to metrics that are produced by monitoring rules with action *OutputMetric*. The required interface towards a Metrics Observer is a REST interface and is available at the following links:

v1.0: https://github.com/deib-polimi/modaclouds-metrics-observer/blob/v1.0/doc/api.md#rest-api
Latest version: https://github.com/deib-polimi/modaclouds-metrics-observer/blob/master/doc/api.md#rest-api

A Java Library has been created with an implementation of the above REST API, featuring the creation of a server and the result deserialization, and is available together with the documentation at the following links:

v1.0: https://github.com/deib-polimi/modaclouds-metrics-observer/tree/v1.0
Latest version: https://github.com/deib-polimi/modaclouds-metrics-observer

2.2.3 Object Store

The required interface towards the Object Store will be defined during the integration with Object Store, due at M24. It will consist of an interface that allows all the components to retrieve the url of the knowledge base and will be available at the following link:

Latest version: https://github.com/deib-polimi/modaclouds-monitoring-manager/blob/master/doc/required-if.md#object-store

A mock-up library for accessing the object store for simulating components discovery was implemented. Version 1.0 of the monitoring platform uses version 0.1-mockup of the object-store-api which is just a mock-up replacing the M24 real implementation. The release together with the documentation is available at the following links:

v0.1-mockup: https://github.com/deib-polimi/modaclouds-object-store-api/tree/v0.1-mockup
Latest version: https://github.com/deib-polimi/modaclouds-object-store-api

2.3 Internal Interfaces

2.3.1 Knowledge Base

The Knowledge Base must provide a REST interface compliant with the SPARQL protocol over HTTP[1].

For the purposes of the Monitoring Platform, a Java library was implemented to abstract from the RDF representation of entities in the KB. The qos-models library (see Section 3.1) offers a model of the ontology in terms of java classes. Every entity in the KB inherits from the KBEntity class and the *knowledge-base-api* library offers an interface for performing CRUD operations on KBEntities towards the KB. It is available together with the documentation at the following links:

v1.0: https://github.com/deib-polimi/modaclouds-knowledge-base-api/tree/v1.0
Latest version: https://github.com/deib-polimi/modaclouds-knowledge-base-api

Developers implementing non-java components, should refer to the the ontology in order to implement their API for serializing and deserializing KBEntity on the KB. The ontology is available at the following links:

v1.0: https://raw.githubusercontent.com/deib-polimi/modaclouds-qos-models/v1.0/metamodels/monitoringontology/monitoring_ontology.ttl
Latest version: https://raw.githubusercontent.com/deib-polimi/modaclouds-qos-models/master/metamodels/monitoringontology/monitoring_ontology.ttl

Figure 3 gives a graphical representation of the monitoring platform ontology.

2.3.2 Deterministic Data Analyzer

The Deterministic Data Analyzer must provide an implementation of RDF Stream Processor REST Services (rsp-services) for C-SPARQL engine. Version 1.0 of the monitoring platform requires version 0.4 of rsp-services:

0.4: https://github.com/streamreasoning/rsp-services/releases/tag/0.4
Latest version: https://github.com/streamreasoning/rsp-services

Refer also to the iswc poster *"A Restful Interface for RDF Stream Processors"*, by Balduini M. and Della Valle E. [4], for further details on the specification.

In order to access the DDA there are two java libraries provided.

rsp-services-api is a low level library that allows to access any rsp-services implementation. The release together with the documentation is available at the following links:

0.4.1: https://github.com/streamreasoning/rsp-services-api/tree/0.4.1
Latest version: https://github.com/streamreasoning/rsp-services-api

Version 1.0 of the monitoring platform uses version 0.4.1 of the rsp-services-api.

[1] http://www.w3.org/TR/2013/REC-sparql11-http-rdf-update-20130321/

modaclouds-dda-api is a java library which features an higher level interface for data collectors to access the DDA, hiding the complexity of RDF. It is developed for the purposes of the monitoring platform. It is available together with the documentation at the following links:

v1.0: https://github.com/deib-polimi/modaclouds-dda-api/tree/v1.0
Latest version: https://github.com/deib-polimi/modaclouds-dda-api

For implementing non-java data collectors please refer to documentation on how data collectors should send monitoring data to the DDA, which can be found at the following links:

v1.0: https://github.com/deib-polimi/modaclouds-dda-api/blob/v1.0/doc/dev-manual.md
Latest version: https://github.com/deib-polimi/modaclouds-dda-api/blob/master/doc/dev-manual.md

2.3.3 Metrics Observer

See Section 2.2.2.

3 Monitoring components

3.1 Monitoring Manager

Figure 6 shows the main building blocks of the monitoring manager.

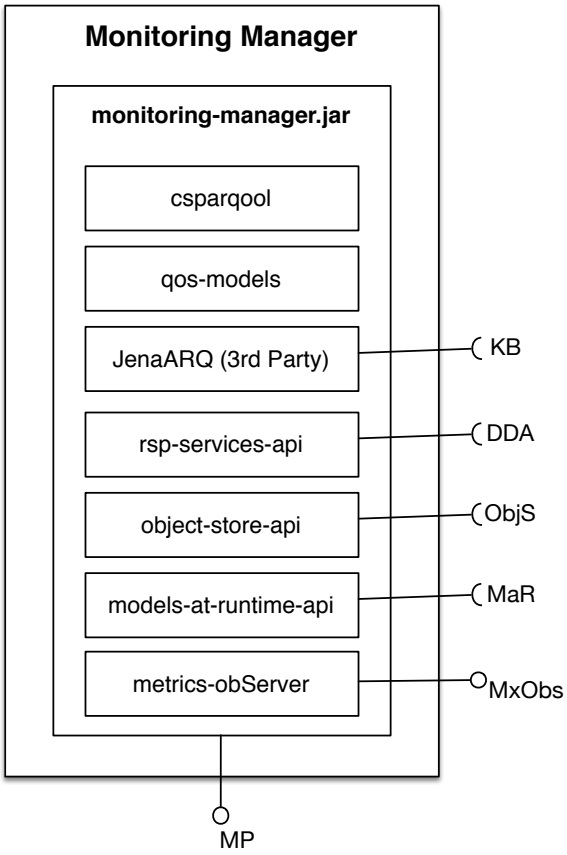

Figure 6: Monitoring Manager Components

csparqool a library for writing C-SPARQL queries in an object oriented language. This library was implemented to facilitate the translation from monitoring rules to C-SPARQL queries. It is available together with the documentation at the following links:

v1.0: https://github.com/deib-polimi/csparqool/tree/v1.0
Latest version: https://github.com/deib-polimi/csparqool

qos-models a library featuring all QoS Analysis and Monitoring Tools models, together with tools for serialization and deserialization, validation and creation of monitoring rules from qos constraints. It is available together with the documentation at the following links:

v1.0: https://github.com/deib-polimi/modaclouds-qos-models/tree/v1.0
Latest version: https://github.com/deib-polimi/modaclouds-qos-models

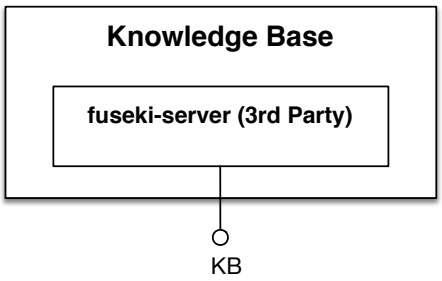

Figure 7: Knowledge Base Components

JenaARQ a third party library used for querying the Knowledge Base. Documentation and Releases are available at the following links:

Documentation: https://jena.apache.org/documentation/query/sparql-remote.html
Releases: https://jena.apache.org/download/index.cgi

Version 1.0 of the monitoring platform uses version 2.11.1 of JenaARQ.

rsp-services-api see Section 2.3.2.

object-store-api see Section 2.2.3

models-at-runtime-api a library for accessing the Models@runtime services. In this case we did not implement any mock-up since the interaction between the two platforms is still to be defined. Refer to the monitoring manager latest documentation for future developments.

metrics-obServer see Section 2.2.2.

3.2 Knowledge Base

As one can see from Figure 7 the only component that is used for implementing the knowledge base is Fuseki, a third party implementation of the KB interface defined in Section 2.3.1.

Fuseki is a SPARQL server that provides *REST-style SPARQL HTTP Update, SPARQL Query, and SPARQL Update using the SPARQL protocol over HTTP*[2]. Version 1.0 of the monitoring platform uses version 1.0.1 of Fuseki. Releases and documentation are available at the following links:

Documentation: http://jena.apache.org/documentation/serving_data/
Releases: http://jena.apache.org/download/index.cgi

3.3 Data Collectors

Figure 8 shows the main building blocks of both java and non-java data collectors factories.

All data collectors factories must be able to interface with the following three interfaces:

[2] http://jena.apache.org/documentation/serving_data/

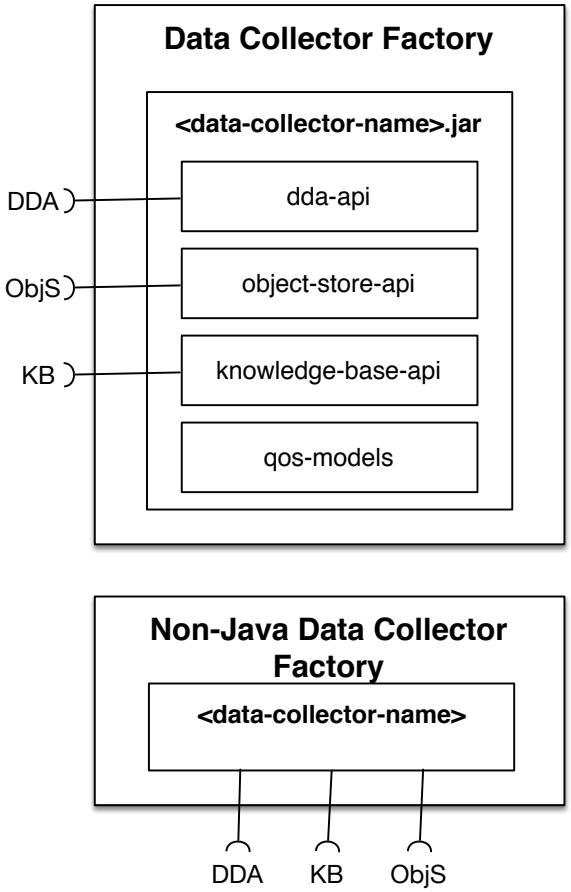

Figure 8: Data Collector Factories Components

DDA monitoring data must be sent to the DDA accessing the exposed interface (see Section 2.3.2).

ObjS the URL of the KB must be retrieved from the Object Store through the exposed interface (see Section 2.2.3).

KB for the API required to access the KB see Section 2.3.1. In the KB, data collectors factories will retrieve the configuration of their data collectors (one DataCollector object each).

3.3.1 Supported Data Collectors

According to the specifications in deliverable D6.2, 12 DCs have been developed, each capable of collecting several metrics. These DCs cover mainly the infrastructure-level, but also cover a set of application level metrics. An individual DC can acquire multiple metrics, from a few tens, to hundreds in the case of the MySQL database DC.

The data collectors include:

- JMX data collector
- Collectl data collector
- Sigar data collector
- Log file parser

- MySQL data collector
- Amazon EC2 CloudWatch collector
- Flexiant Cloud monitor
- EC2 spot price monitor
- Start-up time monitor
- Cost monitor
- Detailed cost monitor
- Availability/Reliability monitor

Here we briefly describe each data collector. The detailed information is available on the GitHub repository.

- *JMX data collector* The Java Management Extensions (JMX) is a set of specifications and tools for managing and monitoring applications in the Java development and application environment. It is especially suitable for managing and monitoring applications deployed in the Java virtual machine (JVM).
- *Collectl data collector* Collectl is a lightweight monitoring tool providing a wide variety of information on the running system such as CPU, disk, network, memory and process performance and status data. It is capable of achieving find grained monitoring with low overhead.
- *Sigar data collector* Hyperics System Information Gatherer (Sigar) is a cross-platform API to collect system information. It supports Linux, FreeBSD, Windows, Mac OSX and Solaris. It provides data concerning CPU, memory, network, file system and process. This information is available in most operating systems.
- *Log file parser* Log files are quite important in today's application systems since they record the events taking place in the execution of the system and provide valuable information to track the activities of the system which helps to diagnose system errors or understand user behaviour. We implemented two log file parsers for illustration purposes: the Apache log file parser and the OFBiz log file parser.
- *MySQL data collector* Since MySQL is a very common database, we implemented a monitoring collector for MySQL. In specific, we use MySQL's command SHOW GLOBAL STATUS to collect the database information. The collector will connect to the remote database, send the query periodically and wait for the return of the statistics.
- *Amazon EC2 CloudWatch collector* Amazon Elastic Compute Cloud (EC2) is one of the most popular IaaS offerings in the world. It provides access to elastic, reliable, and secure compute resources. In order to monitor the behavior of Amazon EC2 instances, Amazon provides the CloudWatch monitoring tool, which is used by the MODAClouds monitoring platform, for both the resources and the applications.
- *Flexiant Cloud monitor* Flexiant offers a customisable method of exposing metrics to monitoring probes rather than off the shelf or existing solutions. This allows for the exposure of physical level parameters, which is the more interesting data. This is also what deployment decisions should be made on and is a reliable indicator of the health of the node. This provides useful data not usually available for a monitoring solution.

- *EC2 spot price monitor* Spot instances from Amazon Web Services allow users to bid on spare Amazon EC2 instances and run them whenever the bid price is higher than the current Spot price. The Spot price varies based on the supply and demand for the instances. The monitor will obtain the latest Spot price for the instances.
- *Start-up time monitor* The start-up time for instances on Cloud is an important concept. It is the time between starting an instance and being able to log in. This information is useful for instance at runtime resource allocation, hence the workload must be predicted in a time horizon that is beyond the start-up time of the instances so that if starting a new instance is required there will be enough time.
- *Cost monitor* The cost spent in Cloud computing is an important factor for users. It determines the strategy of the users on how to deploy their applications. The cost includes the rent of the virtual machines, network cost, storage cost, etc. This monitor provides the general cost metric "EstimatedCharges" for the user on the Amazon EC2.
- *Detailed cost monitor* The detailed cost monitor is able to provide information about the cost for each instance on EC2 with the help of the billing information from Amazon AWS.
- *Availability/Reliability monitor* The availability and reliability of an application or an instance are crucial metrics for the users and the cloud providers. For cloud providers, they are normally defined in the SLAs and if they are violated the cloud providers have to pay a fine. For the users, the availability and reliability directly affect their normal usage of the applications and instances. This monitor provides metrics like MTTF, MTTR and availability.

Through different kinds of Data collectors, it is possible to provide different levels of monitoring metrics. For instance, the Collectl and Sigar data collectors offer general infrastructure level metrics while Flexiant Cloud monitor and EC2 CloudWatch collectors provides specific cloud provider metrics. The MySQL monitor, Log file parser and the JMX data collectors collect application level and container level monitoring metrics. The other collectors such as the Start-up time monitor, Cost monitor, Detailed cost monitor and Availability/Reliability monitor provide Cloud related metrics.

v1.0: https://github.com/imperial-modaclouds/Modaclouds-DataCollectors/releases/tag/v1.0
Latest version: https://github.com/imperial-modaclouds/Modaclouds-DataCollectors
Wiki : https://github.com/imperial-modaclouds/Modaclouds-DataCollectors/wiki

The wiki contains the following information:

- Data Collectors: A detailed description of each data collector
- Rest Interface: How to use Rest service to activate data collectors
- Configuration File: Explain the configuration file for each data collector
- Structure: A brief introduction to the structure of the code

3.3.2 Application Level Monitoring

With the purpose of allowing monitoring also on applications deployed on PaaS, we developed another Data Collector type which is able to collect some metrics through code injection, exploiting aspect oriented programming and reflection. For this deliverable we implemented a library that can be added as a dependency to any java application and through simple annotations on the methods. Metrics such as *execution time* and *throughput* are collected and sent to the DDA. The library is available together with the documentation at the following links:

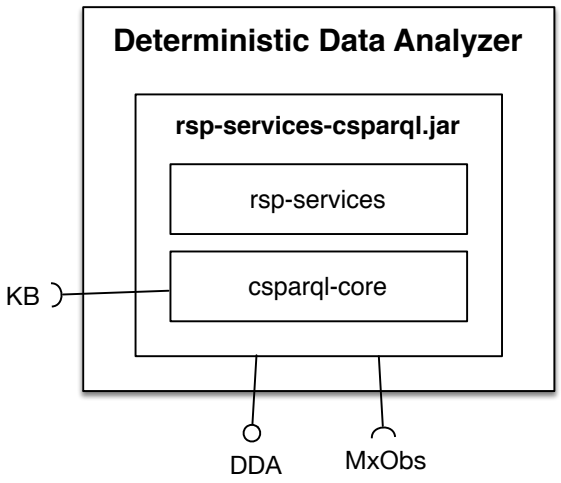

Figure 9: The Deterministic Data Analyzer Components

v1.0: https://github.com/deib-polimi/modaclouds-app-level-dc/tree/v1.0
Latest version: https://github.com/deib-polimi/modaclouds-app-level-dc

3.4 Deterministic Data Analyzer

Figure 9 shows the main building blocks of the deterministic data analyzer.

Rsp-services-csparql is an implementation of rsp-services for the C-SPARQL engine. It allows components to access the C-SPARQL engine through the REST api defined in Section 2.3.2. The DDA requires access to the knowledge base for interpreting monitoring data and executing rules. It is also responsible of sending monitoring data to metrics observers and requires observers to expose the Metrics Observers interface (see Section 2.2.2).

Version 1.0 of the monitoring platform uses version 0.4.3 of rsp-services-csparql. It is available together with the documentation at the following links:

0.4.3: https://github.com/streamreasoning/rsp-services-csparql/releases/tag/0.4.3
Latest version: https://github.com/streamreasoning/rsp-services-csparql

3.5 Statistical Data Analyzers

Figure 10 shows the main building blocks of both statistical data analyzer factories.

All statistical data analyzer factories must be able to interface with the following interfaces:

DDA monitoring data processed by any SDA must be sent back to the DDA exactly as any data collector does. So refer to Section 2.3.2.

metrics-obServer in order to receive raw monitoring data to be processed, SDA factories must be metrics observers. see Section 2.2.2.

ObjS the url of the KB must be retrieved from the Object Store. See Section 2.2.3.

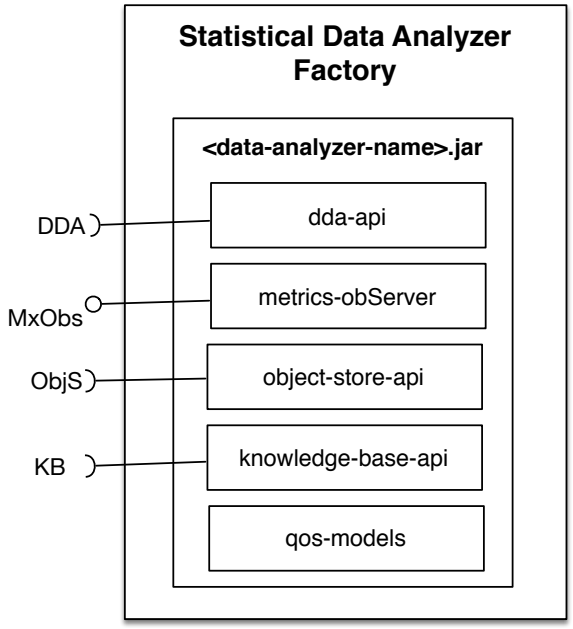

Figure 10: Statistical Data Analyzer Factory Components

KB the access to the KB for SDA factories is the same of DC factories except for the fact that SDA factories will retrieve from the KB StatisticalDataAnalzyzer objects for their configuration instead of DataCollector objects. Refer to 2.3.1.

3.5.1 MCR runtime environment

The SDAs run on the MATLAB Compiler Runtime (MCR). MCR is a container that allows the execution of MATLAB scripts as standalone applications or shared libraries on machines without a MATLAB installation. MCR can run in any major operating system. The SDAs are thus developed in MATLAB, but they can be run on any machine through the MCR. In the current release, a set of precompiled SDAs is provided, so no MATLAB programming skills or compiler availability are assumed. The data acquisition from the DCs relies on the ability of the MCR to execute Java code. The installation of the MCR runtime environment has however to be customised for execution with the MODAClouds monitoring platform, details are given in the installation instructions section.

3.5.2 Estimation SDAs

Estimation SDAs are a key component of the MODAClouds Monitoring Platform, and their results are useful for other components of the MODAClouds Monitoring Runtime Environment. The Self-Adaptation Platform relies on performance models to evaluate the effect of the possible adaptation actions that can be taken under the varying conditions of a cloud deployment. The success of these models highly depends on two main elements: their ability to adequately capture the main features of the application and its deployment; and the quality of the model parameter values, such that they reflect the actual conditions of the application and its deployment.

The quality of the model parameters is achieved by the Estimation SDAs, which must provide tools to accurately estimate the value of these parameters from the available monitoring metrics. While some parameters can be easily estimated from server logs, such as request arrival rates, others can be very challenging to obtain, such as the resource demands posed by the requests. For instance, a key parameter in many performance models is the effective CPU time consumed while processing each request

type. Monitoring this metric directly is possible, but the overhead necessary can be too large to be a feasible alternative for production systems, where continuous monitoring would be required. It is therefore necessary to infer the resource demand from the available monitoring metrics, which depend on the deployment environment.

We have developed a number of estimation methods, and implemented these, as well other methods recently developed by other researchers, as Estimation SDAs. The main difference among these methods is that they require different input metrics and can offer different tradeoffs between computational costs and accuracy. We offer four different algorithms: Complete Information, Utilization-based optimization, Utilization-based regression and FCFS regression. The details are in the GitHub wiki which is described in the last subsubsection.

3.5.3 Forecasting SDAs

Forecasting SDAs are another important component of the MODAClouds Monitoring Platform. At runtime, the system will face varying workload like burstiness and the system should react automatically to deal with it such as starting a new virtual machine to balance the requests. Therefore it is necessary for the system to predict the incoming workload so that there is enough time for it to actuate.

In addition to forecast the workload, other metrics could also be predicted. For instance, if there is a seasonal pattern in the CPU utilization the future usage could be predicted based on the pattern. This applies to all the metrics we have. As long as a certain value is needed, the time series forecasting could be employed.

We have implemented a number of forecasting methods including both time series forecasting and machine learning based forecasting algorithms. For the time series forecasting, we include the methods such as Autoregressive model (AR) and Autoregressive integrated moving average model (ARIMA). For machine learning algorithms, there are Linear Regression, Gaussian Process and SMO regression methods. The details are in the GitHub wiki which is described in the last subsubsection.

3.5.4 Correlation SDAs

Correlation SDAs are a non negligible component of the Monitoring Platform. It is mainly used to obtain estimations for metrics that haven't been monitored. For instance, to obtain the response time at run time requires looking at both the arrival and departure timestamps of the request, which poses overhead to the system. Instead, a machine learning model could be trained offline with a benchmark to correlate metrics like CPU utilization, throughput and the response time to extract the potential patterns between them. Besides the value of the response time, the Correlation SDA also supports estimating classes. From the last example, we could classify the response time as "violation" or "success" based on a predefined threshold. Then the training model will report if the response time has violated or not at run time with CPU utilization and throughput.

We use WEKA library (http://www.cs.waikato.ac.nz/ml/weka/) to implement the machine learning methods. For different functionalities, different algorithms are provided.

- *Correlate classes*: Naive Bayes and SMO methods are provided.
- *Correlate values*: Linear Regression and SMO regression are provided.

v1.0: https://github.com/imperial-modaclouds/Modaclouds-SDA/releases/tag/v1.0
Latest version: https://github.com/imperial-modaclouds/Modaclouds-SDA
Wiki : https://github.com/imperial-modaclouds/Modaclouds-SDA/wiki

The wiki contains the following information:

- Data Analyzers: A detailed description of each data analyzer
- SDA Retriever: Introduction of the retriever to receive data from DDA
- Configuration File: Explain the configuration file for each data analyzer
- Installation: The installation steps to use SDAs
- Compilation: Describe the procedures of compiling the MATLAB scripts

3.6 Metric Explorer

The Metric Explorer, as described in D6.3.1, is a support tool for the storage and visualization of metrics. It provides support for visual inspection of metrics through a configurable dashboard web services as part of T5.4 Filling the gap, its full architecture will be described in D5.3.2.

In order to interact with the monitoring platform as well as support additional monitoring environments, the Metric Explorer features a series of metric adapters whose main purpose is to interface with various collectors and retrieve and convert the metrics in an internal format that is later stored for visualization.

As such, the C-SPARQL Adapter will register as a metrics observer in order to retrieve the results produced by monitoring rules and forward them to the Messaging Middleware. As such, it will implement the REST interface presented in 2.2.2 by using the provided Java library.

4 Monitoring metrics

In this section, the default supported monitoring metrics are listed. For some metrics, several Tools are able to collect it. For instance, both collectl and Sigar are able to collect CPU utilization values. However, Sigar also works on Windows while collectl only runs on Linux systems.

Table 1: Infrastructure level monitoring metrics

Metric	Resources	Definition	Tools
CPUUtil	Host, VM	Total CPU utilization. Normalized in 0-1. At host level the metric refers to the application VM.	collectl, Sigar
DiskReadBytes	Host, VM	Number of bytes read from disk since last measurement. At host level the metric refers to the application VM.	collectl
DiskReadOps	Host, VM	Number of read operations to disk since last measurement. At host level the metric refers to the application VM.	collectl
DiskWriteBytes	Host, VM	Number of bytes written to disk since last measurement. At host level the metric refers to the application VM.	collectl
DiskWriteOps	Host, VM	Number of write operations to disk since last measurement. At host level the metric refers to the application VM.	collectl
NetworkInBytes	Host, VM	Number of bytes received over the network interfaces since last measurement. At host level the metric refers to the application VM.	collectl
NetworkOutBytes	Host, VM	Number of bytes sent over the network interfaces since last measurement. At host level the metric refers to the application VM.	collectl
ContextSwitch	VM	Total number of CPU context switches since last measurement.	collectl
CPUUtilStolen	VM	Time spent for a virtual CPU in wait state while the hypervisor is servicing another virtual processor.	collectl, Sigar
Interrupts	VM	Total number of CPU interrupts since last measurement.	collectl
MaxProcs	VM	Maximum number of processes running simultaneously since the last measurement	collectl
MemUsed	VM	Number of bytes in memory at the measurement instant.	collectl, Sigar
MemSwapSpaceUsed	VM	Used memory swap space at the measurement instant.	collectl

Table 2: JVM Container-Level Metrics

Metric	Definition	Tools
PeakThreadCount	Peak live thread count since the JVM started or peak was reset.	JMX
HeapMemoryUsed	Current memory usage of the heap that is used for object allocation.	JMX
NonHeapMemoryUsed	Current memory usage of non-heap memory that is used by the Java virtual machine.	JMX
MemoryPoolPeakUsage	Peak memory usage of a memory pool since the JVM started or since the peak was reset.	JMX
MemoryPoolUsage	Estimate of the memory usage of a memory pool.	JMX
Uptime	Uptime of the Java virtual machine in milliseconds	JMX

Table 3: Infrastructure level monitoring metrics (Amazon EC2)

Metric	Definition
CPUUtilization	The CPU utilization in percentage.
DiskReadOps	Number of read operations to disk since last measurement.
DiskWriteOps	Number of write operations to disk since last measurement.
DiskReadBytes	Number of bytes read from disk since last measurement.
DiskWriteBytes	Number of bytes written to disk since last measurement.
NetworkIn	Number of bytes received over the network interfaces since last measurement.
NetworkOut	Number of bytes sent over the network interfaces since last measurement.

Table 4: Infrastructure level monitoring metrics (Flexi cloud)

Metric	Definition
NodeID	The ID of the node.
NodeRAM	The number of available of RAM in the node.
CPUcores	The number of CPU cores in the node.
State	The system state.
Error	Indication of if error occurs
Load	The system load.
Servers	The number of servers on this node.
Configgroup	The number of configuration group.
Nodetype	The type of this node.
Routergroup	The number of router group.
MACaddresses	The MAC address of the machine

Table 5: Application-Level Metrics

Metric	Definition	Tools
RequestTimestamps	The timestamps of the current record	OFBiz log file parser
RequestClientIP	The IP address of the client for current request	OFBiz log file parser
RequestAction	The request type, such as login, logout, etc.	OFBiz log file parser
RequestFlag	The flag of the current request, i.e. begin or done	OFBiz log file parser
ClientIP	The client IP address	Apache log file parser
UserIdentifier	The identifier of the user	Apache log file parser
UserID	The user ID of the client	Apache log file parser
Timestamps	The timestamps of the request	Apache log file parser
RequestContent	The content of the request	Apache log file parser
HTTPStatusCode	The http status code from the server	Apache log file parser
ObjectSize	The size of the object returned to the client	Apache log file parser
Referer	The site that the client reports having been referred from	Apache log file parser
UserAgent	The information of the user agent, like the client browser	Apache log file parser

Table 6: Application-Level Metrics (MySQL)

Metric	Definition
Uptime	The time that the server has been up in second
Uptime_since_flush_status	The time since last flush
Threads_running	The number of threads running currently
Threads_cached	The number of threads in cache
Threads_connected	The number of threads connected currently
Threads_created	The number of threads created
Queries	The number of queries executed by the server
Bytes_received	The number of bytes received from all clients
Bytes_sent	The number of bytes sent to all clients
Connections	The number of all connection attempts
Aborted_connects	The number of failed connections to the server
Aborted_clients	The number of connections aborted due to the client died without closing the connection properly
Table_locks_immediate	The number of times that a request for a table lock could be granted immediately
Table_locks_waited	The number of times that a request for a table lock could not be granted immediately
Com_insert	The number of times an insert query has been executed
Com_delete	The number of times a delete query has been executed
Com_update	The number of times an update query has been executed
Com_select	The number of times a select query has been executed
Qcache_hits	The number of query cache hits

Table 7: Cloud Specific Metrics

Metric	Definition	Tools
Availability	The ratio of time the application or VM is functional to the total time	Availability monitor
MTTF	Mean time to failure	Availability monitor
MTTR	Mean time to repair	Availability monitor
GeneralCost	Total cost on the Amazon AWS	Cost monitor
Cost	The cost on Amazon EC2 per instance	Detailed cost monitor
SpotPrice	The current price for a particular type of Amazon spot instance	EC2 spot price monitor
StartupTime	The startup time for a particular VM or application	Startup time monitor

References

[1] http://cloudml.org/.

[2] "Topology and orchestration specification for cloud applications," OASIS, 2011.

[3] http://www.w3.org/RDF/.

[4] M. Balduini and E. D. Valle, "A restful interface for rdf stream processors," http://www.iswc2013.semanticweb.org/sites/default/files/iswc_poster_8.pdf.