2011 Third IEEE International Conference on Coud Computing Technology and Science

Cloud Application Monitoring: the mOSAIC Approach

Massimiliano Rak, Salvatore Venticinque
Second University of Naples
Aversa, Italy
massimiliano.rak, salvatore.venticinque@unina2.it

Tamás Máhr
AITIA International, Inc.
Budapest, Hungary
tmahr@aitia.ai

Gorka Echevarria, Gorka Esnal
Tecnalia
San Sebastin, Spain
gorka.echevarria,gorka.esnal@tecnalia.com

Abstract

Cloud computing delegates the management of any kind of resources, such as the computing environment or storage systems for example, to the network. The wide-spread permeation of the Cloud paradigm implies the need of new programming models that are able to utilize such new features. Once the problem of enabling developers to manage Cloud resources in a clear and flexible way is solved, a new problem emerges: the monitoring of the quality of the acquired resources and of the services offered to final users. As the first step, the mOSAIC API and framework aim at offering a solution for the development of interoperable, portable and Cloud-provider independent Cloud applications. As the second step, this paper introduces the mOSAIC monitoring components that facilitate the building of custom monitoring systems for Cloud applications using the mOSAIC API.

1 Introduction

The Cloud Computing paradigm has been an incredible success in the last few years. The Cloud approach has largely been adopted in very different contexts and applied to an incredibly large set of technologies (which varies from virtualization to GRID computing and service oriented architectures). It is based on the idea of *delegate to the network* every kind of resources, and let the users access them in a self-service fashion (see the Cloud definition from NIST[15]). In a Cloud Environment every kind of resource is managed through a Service Oriented Architecture. Thanks to virtualization techniques, this holds even for resources that are usually managed at the physical level, e.g. computational resources or available memory. This implies incredible flexibility in terms of resource management, which is otherwise often hard to manage. As a consequence, today there is a need for new programming models that help developers in building up Cloud applications that can take advantage of such new features. At the state of art many research projects are exploring new ways to manage such flexibilities (OPTIMIS, CONTRAIL, Cloud4SOA, mOSAIC) by building Platform-as-a-Service (PaaS) solutions to help in developing Cloud applications. In this paper we focus on the solution proposed in the context of the mOSAIC project, which aims at offering a set of APIs to enable developers to build up Cloud applications in a very flexible way, being completely independent from the Cloud providers that offer the resources. The mOSAIC API offers a solution for Cloud-application developers in order to build up software systems which are portable and Cloud provider independent. This is achieved by introducing an abstraction of the way in which resources are accessed and connected to each other (see [14, 8, 10]). Accessing (virtualized) resources through high level abstraction helps in building Cloud applications in a portable way, but it has an impact on the quality of the services offered. It raises the issue whether a Cloud application is able to ensure quality features like response time or throughput. This question has two different sides: a) in which way should users and providers agree on the quality of the services offered (Service Level Agreements) and b) how can an application monitor the conformance to the agreements acquired. In [11] we propose a general architecture for management of SLA in a mOSAIC application, in this paper we focus on the monitoring sub-problem (problem b). Monitoring Cloud resources is a well known problem (see section 6). In fact it is possible to control monitoring in many different ways, depending on the granularity of the monitoring requested (which affects the monitoring overhead), on the kind of resources

978-0-7695-4622-3/11 $26.00 © 2011 IEEE
DOI 10.1109/CloudCom.2011.117

offered, on the way in which the resources are delivered, and on the services offered by Cloud providers. This paper presents the solution offered by the mOSAIC API for the problem of Cloud application monitoring, it enlists the kind of services offered, and discusses the way in which these services can be used and how (and if) they affect the behavior of the Cloud application. The remainder of this paper is organized as follows: the next section briefly summarizes the mOSAIC approach to Cloud-application development, while section 3 describes the monitoring problem in the context of mOSAIC API and Framework. Section 4 offers an overview of the components dealing with resource monitoring that are also applied in a simple example in section 5. The paper ends with a related work and a conclusion section which summarize the state of art of the monitoring problem and the results presented in this paper.

2 Programming the Cloud: mOSAIC API

The full mOSAIC solution includes four main modules: the mOSAIC API, the mOSAIC framework or platform, the mOSAIC provisioning system, and the semantic engine. In this section we will focus on the relationship between the API, the framework and the provisioning system. The role of the provisioning system is to decouple Cloud applications from Cloud providers. The mOSAIC developer should never focus on any Cloud-provider specific resource, instead he should refer to resources through abstractions (named connectors), that offer uniform access independently from the provider and the technologies it supports. The provisioning system works mainly at Infrastructure as a Service (IaaS) level, managing resources like virtual machines, Cloud storage or communication systems. In mOSAIC, the functionality of the provisioning system is realized as part of the *Cloud agency* [16]. The mOSAIC framework is a collection of *Cloud components* that can run independently, and can offer a clear set of services. Predefined mOSAIC components aim at offering simple and common services which can be used by mOSAIC developers to easily build up complex applications. HTTPgw (HTTP gateway) is an example for a mOSAIC component; it is a component that offers an HTTP interface and forward messages to Cloud queues. The mOSAIC framework constitutes a Platform as a Service (PaaS) solution which enable the execution of complex services with predefined interfaces. The rationale behind the framework is to offer a simple way to reuse a large set of existing technologies and solution when building up a custom application. After introducing the mOSAIC provisioning system and the mOSAIC framework, the role of the API can well be outlined. The API offers a programming model and its implementation in a given programming language (java currently, and python in the future) to allow the building of applications that consume Cloud resources and easily interact with a large set of different technologies. It is important to note that the mOSAIC API is not a collection of wrappers that enable transparent access to Cloud providers for a well known and predefined set of resources (as opposed to other solution like jClouds [12] or Apache Delta Cloud [9]). In contrast, mOSAIC is a completely different way of thinking about Cloud applications, in which the resources are modeled in terms of the functionalities offered, and not in terms of the way in which they are accessed or in which they internally work. For example, a mOSAIC application might use a key-value store system as Cloud storage. The application, however, shouldn't care whether it is an Amazon S3 service instance or a cluster of virtual machines running Riak that provides the key-value store service. The mOSAIC API provides a new set of concepts (like the *Cloudlet*, or the *Connector*) to developers, which helps in focusing on Cloud resources and Cloud communications instead of the details of how to access the resources, or how to perform the communication. As a consequence, the application can optimally exploit the scalability, elasticity, self-adaptivity features that the Cloud paradigm offers. In the inter-Cloud environment, emerging problems related to quality management of a mOSAIC application are managed through a dedicated Service Level Agreement (SLA) based model, and separate mOSAIC components devoted to quality management, that are part of the mOSAIC Framework.

2.1 mOSAIC API Concepts

In mOSAIC, a Cloud application is developed as a composition of inter-connected *building blocks*. A Cloud "building block" is any identifiable entity inside the Cloud environment. It can be the abstraction of a Cloud service or of a software component. It is controlled by the user, it is configurable, exhibiting a well defined behavior, implementing functionalities and exposing them to other application components. Instances of Cloud "building blocks" run in a Cloud environment, and consume Cloud resources. Communication among Cloud components takes place either through Cloud resources (e.g. message queues like AMPQ, or Amazon SQS), or through non-Cloud resources (e.g. socket-based applications). Simple examples of components are: a Java application runnable in a platform as a service environment; or a virtual machine, configured with its own operating system, its web server, its application server and a configured and customized e-commerce application on it. Components can be developed following any programming language, paradigm or API. Cloudlets are the way the mOSAIC API offers to developers to create components. All the Cloudlets that make up a Cloud application run in a Cloudlet container which is managed by the mOSAIC software platform. A Cloudlet can have multiple

instances, and it is impossible to distinguish between the Cloudlet instances at runtime. When a message is directed to a Cloudlet it can be processed by any of the Cloudlet instances. The number of instances running is under control of the Cloudlet container, and it is managed in order to grant scalability (with respect to the Cloudlet workload). Furthermore, Cloudlet instances are stateless, any state is stored using Cloud resources through connectors. Connectors are an abstraction of the access model of Cloud resources (of any kind) and are technology independent. Connectors control Cloud resources through technology-dependent *drivers*. As an example, a Cloudlet can access key-value store systems through a KVstore connector, which uses an interoperability layer in order to control a Riak, or a MemBase KV driver. Further detail on the mOSAIC programming model, which is out of the scope of this paper can be found in [8, 10, 14].

3 The Monitoring Problem in mOSAIC

In order to clarify the requirements for the monitoring API in mOSAIC, it is important to outline the kind of problems it should solve. A mOSAIC application is, as outlined above, a collection of components and resources interconnected to each other. In order to grant any quality of the services offered by such an application, it is important to offer, inside the API, a collection of functionalities that enable application developers to monitor both the application itself and the resources it uses. As a consequence, we can identify two different requirements for the mOSAIC monitoring API:

1. Resources should be monitored.
2. mOSAIC components should be monitored.

Monitoring of resources implies the monitoring of the resources offered by Cloud providers. In this case, the problem is to check that the quality promised by the Cloud provider is effectively respected. It is important to point out that Cloud providers often offer monitoring services (often for an extra charge) in order to evaluate the effective quality of the resources delivered (like Amazon CloudWatch). In addition to such Cloud-provider related solutions, it is possible to use resource related monitoring tools. For example on an IaaS service, it is possible to acquire a virtual machine (VM) and, inside the VM, monitor the effective resource usage using common operating system tools or performing periodical benchmarks. In addition, mOSAIC offers the chance of acquiring custom Cloud resources, like key-value store system built on the top of cluster virtual machines (using different technologies like Riak or MemBase). In such a case, it is possible to use mOSAIC dedicated tools to retrieve monitoring information. As a summary it is possible to identify three different ways of monitoring:

1. Monitoring by Cloud provider tools
2. Monitoring by resource-related tools
3. Monitoring by mOSAIC tools

In conclusion, we identify the following requirements for the mOSAIC API in order to support Cloud-application monitoring:

1. the mOSAIC API should offer a way to collect data directly from any of the components of a mOSAIC application (resources and other components),
2. the mOSAIC API should offer a way to collect data for any proposed techniques (accessing Cloud-provider, resource-related, and mosaic tools), and
3. the mOSAIC Cloud application should access data in a way that is oblivious about both the technology of the acquired resources, and by the way they are monitored.

In order to meet such requirements, the mOSAIC monitoring API follows the same approach adopted by the API for accessing and managing resources. The monitoring API offers a set of *connectors* which represent an abstraction of resource monitoring, and a set of *drivers* which implement different ways of acquiring monitoring data (applying the different techniques). In addition to the mOSAIC API, Cloud application monitoring is also realized by a set of tools offered in the context of the mOSAIC framework. These tools help the mOSAIC developer to build up a dedicated monitoring system in a clear and simple way. The details of such tools are briefly summarized in the following section, dedicated to the monitoring architecture of mOSAIC.

4 mOSAIC Framework Monitoring Architecture

In the context of the mOSAIC framework, the monitoring/warning (M/W) system offers a set of tools (Cloud components, resources, connectors) whose goal is to generate warnings when the target application and/or the associated resources are in conditions which may lead to a violation of one of the requirements defined in the SLA. From a mOSAIC developer point of view the M/W system can be either

- a transparent system that, thanks to the adoption of the autonomic system, grants self-adaptation features and thereby compliance to the agreement, or
- an accessible system that enables the application to monitor its own execution.

To fulfill its above outlined roles, the M/W system should be able to perform the following tasks:

- monitor Cloud resources,
- monitor application components, and
- verify the compliance of a set of rules, in order to discover warning conditions.

Cloud-resources monitoring is strictly linked to resource provisioning, therefore it is mainly managed by the Cloud agency. Monitoring Cloudlets and components, however, is strictly related to the application development, therefore the API should support the building of ad-hoc solutions. Moreover, application Cloudlets and components (either user developed or offered by the platform, like SLA components) need a simple way to share the monitoring information and manage the related events. In order to face the above described requirements, the proposed architecture is organized as illustrated in Figure 1, containing the following main elements:

1. monitoring event buses that collect the monitoring events from a large set of different sources,
2. connectors, dedicated to the event buses, that enable Cloudlets to intercept monitoring events,
3. connectors that generate events sent to the event buses from the application (they act like logging or tracing modules),
4. the MW component, a stand-alone Cloudlet that offers the warning features to the Cloud application.

Communication between the resource monitoring system and the mOSAIC application is based on the adoption of shared event buses. In the figure we propose mainly two different buses. The first bus is adopted in order to collect all events from Cloud resources, and the second one is oriented to collect events from the application. It is important to point out that it is up to the mOSAIC developer to choose the number of buses involved: all the events can be managed through a single bus, or be split in as many buses as the developer consider necessary. In the proposed general architecture, we split them into two, considering that there are two main sources of events: Cloud resources and application components. Resource monitoring is mainly based on the adoption of Cloud agency. The agency offers the *Archiver*, a monitoring agent that collects monitoring information from the agents distributed on all the monitored resources (VMs or VCs), and stores the messages in a storage system. In addition, the *Monitoring* agent should be able to collect monitoring information from common monitoring systems (like ganglia, nagios, SNMP-based applications, ...) and publish them on the same storage. Applications are able to interact with the Archiver through a connector which enables a Cloudlet to access Archiver information. The *Observer* is a component which accesses the storage filled by the Archiver, and generates events on the (resource) event bus. It is the duty of the Observer to generate events in order to distribute *selected information* to all the interested components. The *Warning* component acts on the basis of a policy-based approach: the developer can load a set of rules which identify the warning conditions to be verified on the events generated on the buses. It uses the connectors offered by mOSAIC to access to the event bus. Finally, application monitoring is based on the adoption of the *Application Monitoring* connector, which can be integrated in any Cloudlet, and that generates events on the connected buses.

5 An Example of Application Monitoring

In order to offer a clear view of the proposed approach and of the components we are currently developing we propose a simple example of application enriched with monitoring tools derived from mOSAIC API. Let us assume that a mOSAIC Service Developer (mSD from now on) has already developed a very simple application which receives documents from the network (HTTP interface), and stores them in a KV store after classifying them following a given algorithm. The classifier itself is out of the scope of this paper, we assume that it is developed as a mOSAIC Cloudlet, and that the application uses a mOSAIC component as a key-value store resource. Using such a simple application, we can outline many of the different monitoring needs, showing how the different components can be involved. For this, let us further assume that the mSD aims at monitoring the evolution of the application in order to know when and how to start up new virtual machines to be added to the KV store cluster, if new memory space is required, and what the global amount of resources under use is. In summary, the mSD aims at developing an application which is able to:

- monitor how many documents are classified,
- monitor the KV store memory usage, and
- monitor the CPU load of the virtual machines hosting the KV store.

In this paper, we are interested only in the monitoring of information that can be used to perform manual or automatic load balancing, increasing/decreasing the number of virtual machines, or simply in the monitoring the global cost of application execution. It is out of the scope of this paper to focus on how to identify and apply reaction to the monitored events. Figure 2 shows the architecture of such an application. The upper layer shows the classifier Cloudlet which accesses the KV storage system. Note that we adopted an application level connector, which publishes a monitoring event each time a new document has been uploaded. Moreover the CloudAgency, through the Observer, customized

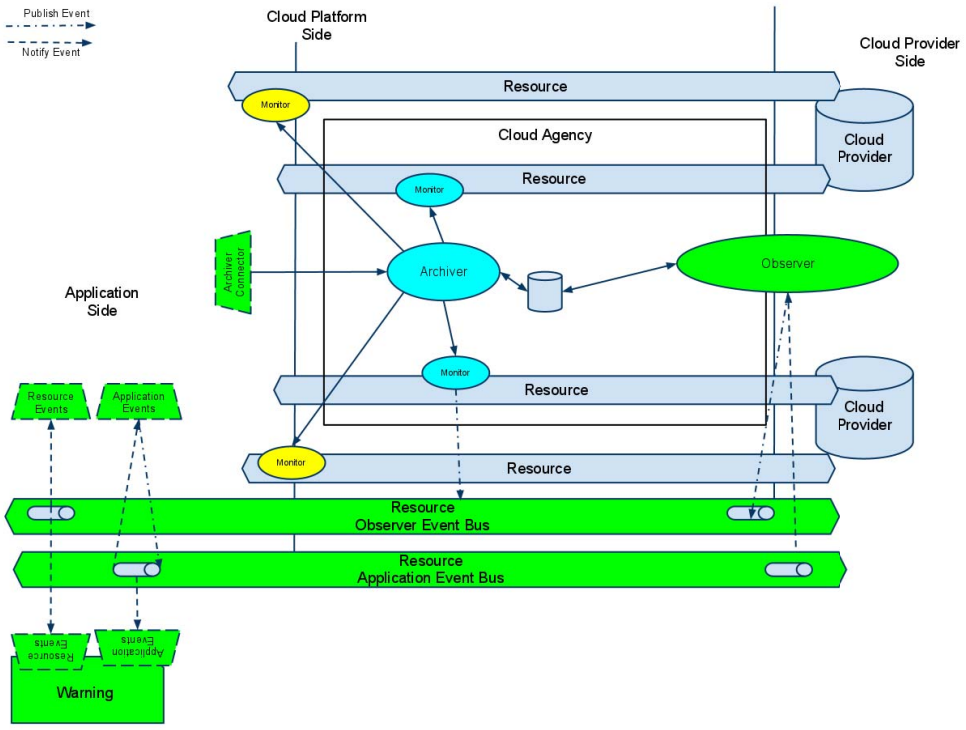

Figure 1: mOSAIC Monitoring Components Architecture

by the user, publish resource-related information, such as the CPU usage, at given time intervals. The mOSAIC developer has the role of developing the monitoring Cloudlet, which, through the connectors to the Event bus is able to retrieve all the monitored events. Moreover it is able to access to the Cloud Agency in order to retrieve more general purpose information.

6 Related Work

As discussed in previous sections, in monitoring of Cloud-based applications, we can distinguish two separate levels: the infrastructure level and the application-level monitoring. Infrastructure-level resource monitoring [7] aims at the measurement and reporting of system parameters related to real- or virtual infrastructure services offered to the user (e.g. CPU, RAM, or data storage parameters). At this level, some of the better advertised Cloud-monitoring solutions include the Nimsoft Monitor [4], Monitis [3], or the Tap in Cloud Management Service [5]. These services cover different subsets of Cloud-service providers, and they support different measurements based on the actual Cloud provider that is being monitored. On the application level, the nature of the monitored parameters, and the way their values should be retrieved depend on the actual software being monitored, and not on the Cloud infrastructure it is running on. VMware vFabric Hyperic [6], e.g., specializes in web-application monitoring, and is capable of monitoring applications that utilize any of the app. 75 supported web technologies. A more general approach is to utilize the JMX Java framework [1], that is employed by most Java application containers, and is capable of providing information on the status of the application running in the container. This, however, requires that the application is written in Java, and that it is prepared to publish information through the JMX subsytem. Regardless of the level of parameters being monitored, a general monitoring infrastructure is required to collect and process the information provided by the monitored components. Such an infrastructure is provided by the Lattice framework [2] (also utilized in the RESERVOIR project [13]) that has a minimal run-time footprint and is not intrusive, so as not to adversely affect the performance of the system itself or any running applications. The framework defines a system of data sources, data consumers, and control strategies that influence the collection of monitoring data. The monitoring data can be transported over IP multicast solutions, Event Service Bus, or a publish/subscribe mechanism. This is a very flexible framework, which is tailored towards distributed applications, but not Cloud applications. In contrast, in the mOSAIC monitoring subsystem, we can utilize such Cloud-oriented services as reliable messaging and flexible storage options for the measurement data.

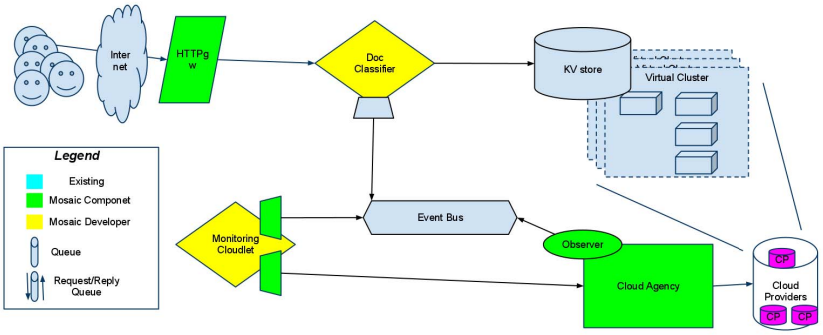

Figure 2: Example of a Monitoring Application

7 Conclusions and Future Works

This paper presents the approach that the mOSAIC project is following in supporting Cloud-application monitoring. The paper offers a brief overview of the mOSAIC API and illustrates the main components, offered in the API and the framework, that can be used to build up a custom monitoring system for a given Cloud application. An example is proposed in order to clarify how the components can be applied in a monitoring system. The approach proposed founds its originality and motivation in the mOSAIC API programming model, which enables development of portable Cloud applications, and applies the same principles to resource monitoring. Through the multiple mOSAIC API layers, it is possible to easily build up custom monitoring systems which collect data at many different granularities. Further steps include the definition of a single format for monitoring-event representation, and the development of different drivers to enable the collection of data from different resources. Such drivers should focus both on monitoring services offered by Cloud providers, and on resources accessed through the mOSAIC components. The proposed solution is intended to be applied in the project case studies.

Acknowledgment

This research is partially supported by the grant FP7-ICT- 2009-5-256910 (mOSAIC).

References

[1] Jmx, java management framework. `http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html`, 2011.

[2] Lattice framework. `http://clayfour.ee.ucl.ac.uk/lattice/`, 2011.

[3] Monitis. `http://portal.monitis.com/`, 2011.

[4] Nimsoft monitor. `http://www.nimsoft.com/solutions/nimsoft-monitor`, 2011.

[5] Tap in cloud management service. `http://www.tapinsystems.com/cloud-services/tap-in-cloud-management-service-overview/`, 2011.

[6] Vmware vfabric hyperic. `http://www.vmware.com/products/vfabric-hyperic/`, 2011.

[7] S. Clayman, A. Galis, C. Chapman, and G. Toffetti. Monitoring service clouds in the future internet. *Framework*, pages 115–126, 2010.

[8] M. v. S. F. Leymann, I. Ivanov, B. S. S. . Science, and T. Publications, editors. *Towards a cross platform Cloud API. Components for Cloud Federation*, 2011.

[9] A. Group. Delta cloud: Many clouds. one api. no problem. `http://incubator.apache.org/deltacloud/index.html`, 2010.

[10] IEEE, editor. *Building an Interoperability API for Sky Computing*, 2011.

[11] IEEE, editor. *User Centric Service Level Management in mOSAIC Applications*, 2011.

[12] Jclouds. Jclouds: multi-cloud library. `http://code.google.com/p/jclouds/`, 2010.

[13] M. Lindner, F. Marquez, C. Chapman, S. Clayman, D. Henriksson, and E. Elmroth. The cloud supply chain: A framework for information, monitoring, accounting and billing. In *2nd International ICST Conference on Cloud Computing (CloudComp 2010)*. Springer Verlag, 2010.

[14] mOSAIC. mosaic: Open source api and platform for multiple clouds. `http://www.mosaic-cloud.eu`, 2010.

[15] Peter Mell and Tim Grance. The nist definition of cloud computing. `http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc`, 2009.

[16] B. D. M. S. Venticinque, R. Aversa and D. Petcu. Agent based cloud provisioning and management, design and prototypal implementation. In M. v. S. Frank Leymann, Ivan Ivanov and B. Shishkov, editors, *1st International Conference on Cloud Computing and Services Science (CLOSER2011)*, pages 184–191. ScitePress, 2011.