A Cloud Monitoring Framework for Self-Configured Monitoring Slices Based on Multiple Tools

Márcio Barbosa de Carvalho, Rafael Pereira Esteves, Guilherme da Cunha Rodrigues,
Lisandro Zambenedetti Granville, Liane Margarida Rockenbach Tarouco
Institute of Informatics – Federal University of Rio Grande do Sul
Av. Bento Gonçalves, 9500 – Porto Alegre, Brazil
Email: {mbcarvalho, rpesteves, gcrodrigues, granville}@inf.ufrgs.br, liane@penta.ufrgs.br

***Abstract*—The monitoring of cloud computing environments is a key point to assure the availability of the cloud slices offered to cloud users. However, there are not any monitoring systems that satisfy all the cloud administrator requirements which imposes that cloud slices need to be monitored by a set of monitoring systems. The set of monitoring system configuration necessary to monitor a cloud slice and the corresponding set of monitored metrics we define as a *monitoring slice*. Unfortunately, the monitoring slices need to be built using solutions that are not integrated with cloud platforms. This lack of integration imposes that cloud administrators manually configure the monitoring solutions or develop scripts to automate this task. In this paper we propose a framework to address the problem of creating monitoring slices automatically independent of the monitoring solutions employed. To evaluate our proposed framework in an IaaS scenario, we develop FlexACMS, flexible automate cloud monitoring slices, which relies on a modules that flexibly handles cloud platforms and monitoring solutions.**

I. Introduction

In Infrastructure as a Service (IaaS) clouds [1], cloud users request computing resources (*e.g.*, compute, storage, and network resources) from cloud providers. The set of resources granted to a cloud user is usually referred to as a *cloud slice*. In order to increase revenue, cloud providers need to optimize the utilization of physical computing resources in each granted slice. At the same time, cloud providers need to guarantee that the offered slices present performance consistent with the cloud user's expectations; otherwise, customers may stop their cloud service subscription, which leads to revenue losses at the provider side. Cloud slices must therefore be closely monitored by the cloud provider, to avoid wasting expensive physical resources while still satisfying the cloud user's expectations.

Once a new cloud slice is created, a set of monitoring solutions need to be configured [2] in order to start monitoring the computing resources that form the new slice. We call the set of monitoring solution configurations and the corresponding monitored metrics a *monitoring slice*. Every cloud slice is coupled with a monitoring slice, whose goal is to detect cloud slice malfunctioning. However, the lack of integration between some monitoring solutions and cloud platforms imposes for cloud administrator to manually set up the solutions or develop scripts to automate the monitoring slice creation. For cloud environments with few cloud slices, triggering scripts or manually setting up monitoring slices may still be possible. However, in larger or more dynamic environments where cloud slices are created and destroyed frequently, manually handling the monitoring slices is unfeasible.

In this paper, we address the problem of automatically setting up cloud monitoring slices. In our solution, we introduce a cloud monitoring framework that enables cloud administrators to describe which monitoring solutions should be used and how these solutions must be configured, in each monitoring slice that needs to be created to monitor new granted cloud slices. When a new cloud slice is in place, its monitoring slice is then automatically set up by the framework. In addition to reduce the administrator's burden from the previously required manual configurations, our solution also facilitates the administrator's exploitation of monitoring solutions.

The main contribution of this paper is a cloud monitoring framework for cloud slices that: *(i)* enables self-configurable cloud monitoring strategies independent of the monitoring solutions employed; *(ii)* automatically creates monitoring slices with solutions that satisfy the cloud administrator's needs; and *(iii)* facilitates the reuse of scripts developed to automate the creation of monitoring slices.

The framework can be used in different cloud platforms and integrates a varying number of monitoring solutions. By collecting information from cloud environments, the framework is able to detect new cloud slices created in the cloud platform. For each new detected cloud slice, the framework triggers components that configure the monitoring solutions, building the corresponding monitoring slice for that cloud slice. Our second contribution is FlexACMS, a flexible automated cloud monitoring system that implements our framework architecture, we developed to evaluate the framework in an Infrastructure as a Service (IaaS) scenario.

The remainder of this paper is organized as follows. In Section 2, the state-of-the-art on cloud computing monitoring is reviewed. In Section 3, we define the monitoring slices, present the framework architecture, the FlexACMS implementation details, and a typical FlexACMS use case. In Section 4, we present brief comments about the evaluation results. Finally, in Section 5, we present conclusions and future work.

II. Related Work

A variety of monitoring solutions are available for cloud operators to gather updated status of cloud slices. Cloud monitoring solutions differ in terms of the resources that are monitored (*e.g.*, servers, storage, network, services), the ability to monitor heterogeneous environments built using different technologies, and the ease of configuration and use. In this section we review relevant cloud monitoring solutions.

Private Cloud Monitoring Systems (PCMONS) [3] is an open-source system for cloud monitoring that abstracts the heterogeneity of a cloud through a layer called Integration, which allows uniform monitoring of different cloud platforms and different virtualization technologies.

Amazon CloudWatch [4] enables scalable and flexible monitoring of Amazon cloud resources and services [5]. Amazon CloudWatch offers several monitoring functionalities, including a set of basic metrics, user-defined metrics, statistics, graphics presentation, and self-configuration.

Runtime Model for Cloud Monitoring (RMCM) [6] is a solution for monitoring all layers of a cloud environment, from the physical substrate to the hosted applications. RMCM provides customizable views of the monitored resources for different interested users (*e.g.*, cloud operators, end-users).

Global Monitoring systEm (GMonE) [2] is a cloud monitoring solution engaged in providing measures of appropriate metrics to clients and providers. On the client-side, GMonE provides metrics related to the established Service Level Agreements (SLAs). On the provider-side, GMonE provides metrics from all levels of a IaaS cloud environment.

In addition to monitoring solutions with native support for cloud computing, traditional monitoring solutions that usually do not have native support for cloud computing, such as Nagios [7], Zabbix [8], and MRTG [9], also can be employed in cloud computing monitoring. Traditional monitoring solutions have positive aspects that are not present in cloud monitoring solutions, *e.g.*, ability to deal with heterogeneous environments and the intrinsic know-how that the administrators have about traditional monitoring solutions.

Available cloud monitoring systems present one or more features that help creating cloud monitoring slices, such as self-configuration to reduce the need for human intervention (*e.g.*, Amazon CloudWatch), ability to monitor heterogeneous infrastructures (*e.g.*, Nagios), and flexibility to adapt generic monitoring solutions to the context of cloud computing (*e.g.*, PCMONS). However, there is no solution integrating all these features in a common monitoring framework. To overcome the limitations of native cloud monitoring solutions and integrating monitoring solutions without support for cloud as well, we propose a self-configurable cloud monitoring framework that allows the automatic creation of monitoring slices using multiple monitoring solutions regardless of the monitored platform.

III. Proposal

Cloud monitoring applications should be self-configurable to both adapt to changes in the cloud platform and minimize the number of error-prone human interventions. In this section, we define the concept of monitoring slices and propose a flexible cloud monitoring framework that creates monitoring slices integrating cloud-specific and non-cloud monitoring solutions that are automatically configured to reflect the creation of new cloud slices. We also provide implementation details of FlexACMS, a flexible automated cloud monitoring system, that was developed using the framework architecture to build monitoring slices as depicted on Figure 1.

A. Monitoring slices and Framework architecture

Monitoring Slices reflect all the monitoring information about a cloud slice which is composed by the collected values of the metrics monitored and the configuration of the monitoring solutions that are needed to collect these metrics. Monitoring slices are composed of monitoring solutions tackling diverse needs of cloud administrators because the monitoring solutions do not fulfill all cloud administrator needs [2]. Figure 1 depicts cloud slices coupled with their corresponding monitoring slices using native cloud monitoring solutions (*i.e.*, OpenStack Ceilometer [10]) and non-cloud monitoring solutions (*i.e.*, Nagios [7], and MRTG [9]).

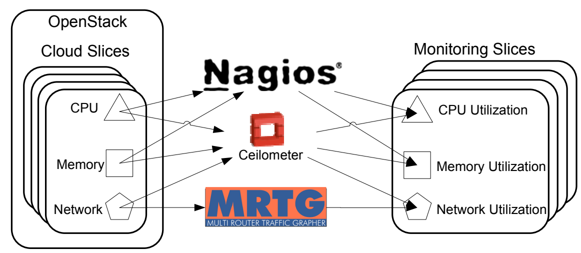

Fig. 1. Typical cloud monitoring scenario composed by cloud-specific and non-cloud monitoring solutions

The framework enables the creation of monitoring slices when a new cloud slice is created. In addition, the framework allows cloud administrators to build monitoring slices with any available monitoring solution that better fits their needs. Thus, cloud administrators do not need to manually detect the creation of new cloud slices and configure the respective monitoring slice manually or triggering scripts. The framework is based on a modular architecture as illustrated in Figure 2. The architecture of our framework is composed of three main components: gatherers, framework core, and configurators.

Gatherers are responsible for collecting information about cloud platforms and for sending that information to the framework core through a REST Web service. Gatherers are developed to collect information from diverse cloud platforms using different interfaces. For example, a gatherer is developed to collect platform information using a specific interface, such as the Amazon EC2 API [11], the OGF Open Cloud Computing Interface (OCCI) [12], or the DMTF Cloud Infrastructure Management Interface (CIMI) [13]. The Amazon EC2 API is a *de facto* standard interface to cloud management and is adopted by cloud platforms such as OpenStack [14]. However, OCCI and CIMI also are proposals of standard interfaces for cloud computing management. Standard interfaces have the advantage of eliminating the need for a high number of platform-specific gatherers, *i.e.*, a small number of gatherers, one for each standard interface, can communicate with cloud platforms compliant with such standard interfaces.

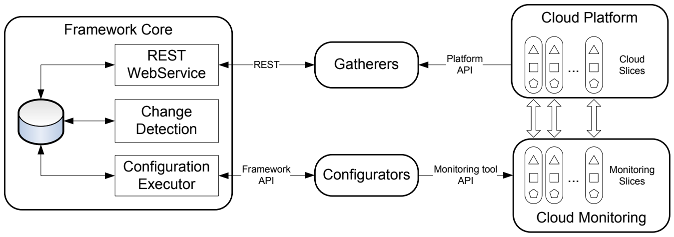

Fig. 2. Framework architecture

The *framework core* is responsible for processing the information about cloud platforms received from the gatherers, to store cloud platform information used to enable the detection of changes (*e.g.*, cloud slice creation), and to trigger configurators to actually build the corresponding monitoring slices of new detected cloud slices. The framework core receives information from gatherers through a REST Web service, which facilitates the development and deployment of gatherers.

After receiving information about cloud platforms from gatherers, the framework core tries to detect changes in the current configuration of the cloud platform, *e.g.*, if new cloud slices have been created. The *Change Detection* module looks at the identifier of each slice informed by the gatherer to compare the informed identifier to the ones currently stored in the database. If the Change Detection module finds a new slice, the module stores the new slice and inserts the detected change (new slice creation) into a change database. Of course, if the cloud platform has some facility to discover the creation of cloud slices, we do not need a change detection module. However, this facility is not available in all cloud platforms which imposes the utilization of this module.

Configurators are responsible for receiving information from framework core and for configuring monitoring solutions that form monitoring slices. Configurators are scripts that are registered in FlexACMS to be triggered to actually configure the monitoring solutions. Thus, the monitoring solutions must be able to be configured by scripts to be supported by FlexACMS. The *Configuration Executor* module triggers the configurators and passes, as arguments, the information that configurators need to setup the monitoring solution. Each configurator handles the peculiarities of the monitoring solutions deployed in the monitoring slice, *e.g.*, generating configuration files (Nagios) or running configuration scripts (MRTG). In fact, the scripts developed by the administrators to automate configuration tasks can be registered on the framework as Configurators. These scripts can benefit from the framework which retrieves the information that the scripts need and automates the execution when a new cloud slice is created.

In the framework, each configurator has an *interest* and a set of *conditions* that are verified. The interest indicates the type of change supported by the configurator (*e.g.*, *new slice*, *new resource*). The conditions are states that the object (*e.g.*, *slice* or *resource*) must satisfy to be configured by the configurator. For example, the `configure_mrtg` configurator, presented in Figure 3, is interested on *new resource* changes because it configures the MRTG solution to monitor network interfaces which are represented as resources on the framework. However, all kind of resources of a slice (*e.g.*, CPU, memory, storage) falls in this interest. Therefore, to restrict the configurator execution to the appropriate resources, `configure_mrtg` configurator must satisfy the condition that the resource identifier corresponds to *network*. Thus, interest and conditions assure that the `configure_mrtg` configurator is executed only when a new network resource is created on a cloud slice.

```
Name:           configure_mrtg
Interest:       New resource
Condition:      @resource.identifier =~ /network/
Command:        /usr/sbin/configure_mrtg.pl
Args:           --slice_name @slice.identifier
                --ip @slice.ip
                --interface_name @resource.interface
```

Fig. 3. Configurator attributes

The *Configuration Executor* module looks to the list of registered configurators, and, for each configurator, the module performs a search on the change database looking for the corresponding changes of interest. For each change matching the configurator's interests, the Configuration Executor checks if the change was not previously configured. If the change was not configured in the past, the Configuration Executor evaluates if the change satisfies all conditions defined by the configurator. If the change satisfies all conditions, the Configuration Executor can trigger the configurator to perform the configuration of the monitoring solution. However, before triggering the configurator, Configuration Executor must evaluate the arguments needed by the configurator in order to run properly. The arguments are the way which the framework core communicates with the configurators and are signaled by the "`@`" in the configurator definition. For example, in the `configure_mrtg` presented in Figure 3, the Configuration Executor needs to retrieve `@slice.identifier`, `@slice.ip`, and `@resource.interface` from the framework database. After retrieving the arguments, the module triggers the configurator using the arguments and stores the configurator output for future analysis or debugging.

B. FlexACMS implementation and Use Case

In this subsection, the scenario illustrated on Figure 1 is reviewed to present the FlexACMS implementation and its typical use case. In the illustrated scenario, cloud slices hosted by the OpenStack platform are monitored through the native cloud monitoring solution Ceilometer [10], and through non-cloud monitoring solutions Nagios and MRTG. However, in the typical scenario, the administrator must manually configure Nagios and MRTG, since these traditional solutions do not satisfy the self-configuration property. Cloud slices are composed of CPU, memory, and network; and monitoring slices reflect CPU utilization, memory utilization, and network utilization.

To collect information about cloud slices and their resources, we develop an OpenStack gatherer that supports the

OpenStack API [15] which is based on Web Services. The OpenStack gatherer was developed using Python and use several web services from the OpenStack API to collect different information. The gatherer sends the collected information to the REST Web service in the FlexACMS core.

The FlexACMS core was developed using the Ruby on Rails framework and uses a MySQL 5.5 database to store the cloud platform information. Each time that the OpenStack gatherer sends information to the FlexACMS core, the Change Detection module evaluates the received information and updates the change database storing all changes (*e.g.*, slice creations) that were performed in the OpenStack platform.

Finally, we develop configurators for both Nagios and MRTG using Perl and Bash scripts, respectivelly. Nagios uses distinct configurations for host and for service/resource status. We then need a configurator to reflect the creation of new slices (host) in Nagios, and a configurator for each monitored resource (CPU and memory). Thus, we develop three configurators to Nagios: one for slice creation, one for CPU utilization monitoring, and one for memory utilization monitoring. On the other hand, we develop a configurator for MRTG to execute scripts that actually configure MRTG: `cfgmaker` and `indexmaker`, used to create the configuration and the index page required by MRTG graphs, respectively.

Furthermore, we need to register these configurators in FlexACMS core using attributes as shown in Table I. After registering configurators, Configuration Executor module looks at the change database and finds all changes that are of interest of the configurators, evaluates if the changed objects satisfy the configurator conditions, and triggers the corresponding configurators to actually configure the monitoring solutions.

TABLE I
CONFIGURATORS ATTRIBUTES TO THE USE CASE SCENARIO

Configurator Name	Interest	Condition
nagios	New Slice	
nagios_cpu	New Resource	@resource.identifier =˜ /CPU/
nagios_memory	New Resource	@resource.identifier =˜ /memory/
mrtg	New Resource	@resource.identifier =˜ /network/

Along this section, we proposed a framework with its described functionally. We develop FlexACMS using the framework architecture that is able to automatically create monitoring slices when cloud slices are created in the OpenStack platform. Moreover, because the framework withdraws from cloud administrators the burden of manually configuring monitoring slices and facilitates the automation scripts development, cloud administrators can explore a variety of monitoring solutions in the context of cloud computing to fulfill their needs. In addition to the description of the functional properties, we provide a brief evaluation about aspects which can influence the deployability of FlexACMS in real IaaS cloud scenarios.

IV. EVALUATION

We observed the execution of a whole cycle since the creation of cloud slices on OpenStack platform to the creation of monitoring slices using Nagios configurators. We create from 1 to 10 servers on OpenStack and observe the time to create their corresponding monitoring slices using Nagios configurators to monitor the host and a resource (CPU). This observation showed that the time to create cloud slices on OpenStack was around to 65% from the whole time of the experiment. FlexACMS uses 35% of the time which is 70% used by the Nagios configurators. The time spent by OpenStack platform and Nagios configurators are related to the scheduling strategy in the creation of cloud slices and the reload process to make new configurations available, respectively. However, both are peculiarities of the platforms and monitoring solutions which affect the correct evaluation of the whole system. Thus, we need to evaluate the FlexACMS core using strategies to do not take into account gatherers and configurators peculiarities.

Despite this initial observation, further evaluation in real scenarios using diverse cloud platforms and monitoring solutions still is desirable. We need to evaluate scalability issues in regards to the framework deploying. This scalability evaluation must include a varying number of metrics that can be configured by the framework which obviously can influence the framework response time. Beyond, the scalability evaluation must consider large scenarios and whether the framework performance is affected in these scenarios.

V. CONCLUSION

In this paper, we presented a cloud monitoring framework that supports the creation of monitoring slices, which are composed of a set of monitoring metrics and associated configurations used to monitor cloud slices on cloud platforms. Monitoring slices are built using diverse monitoring solutions including solutions that are not integrated to cloud platforms (*e.g.*, Nagios, MRTG). The framework presents a modular architecture that allows communication with diverse cloud platforms and monitoring solutions. In our modular architecture, gatherers are modules responsible to interact with cloud platforms to retrieve information about hosted cloud slices and send this information, through a REST Web service, to the framework core. Configurators are modules responsible to retrieve information from the framework core and configure monitoring solutions to build monitoring slices. Furthermore, gatherers and configurators communicate with the framework core through well defined interfaces.

As future work we want to extend the FlexACMS evaluation in IaaS scenarios and beyond evaluate scalability issues on the use of the framework, such as an increasing number of metrics per slice. After solve the problem of create monitoring slices automatically, we are able to address other related cloud monitoring issues, as the reconfiguration and destruction of monitoring slices and adaptive allocation strategies in creating new monitoring slices. We also plan to further observe how the cloud platform affects the time required to create new monitoring slices, and whether the CPU and memory utilization also would be impacted. Furthermore, we want to evaluate the framework in Platform as a Service (PaaS) and Software as a Service (SaaS) cloud models.

REFERENCES

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A Break in the Clouds: Towards a Cloud Definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, Dec. 2008.

[2] J. Montes, A. Sánchez, B. Memishi, M. S. Pérez, and G. Antoniu, "GMonE: A complete approach to cloud monitoring ," *Future Generation Computer Systems*, pp. –, 2013.

[3] S. De Chaves, R. Uriarte, and C. Westphall, "Toward an Architecture for Monitoring Private Clouds," *Communications Magazine, IEEE*, vol. 49, no. 12, pp. 130 –137, Dec. 2011.

[4] Amazon, "Amazon CloudWatch," 2013, available at: http://aws.amazon.com/en/cloudwatch/. access in: May 2013.

[5] ——, "Amazon Web Services," 2013, available at: http://aws.amazon.com/. accessed in: May 2013.

[6] J. Shao, H. Wei, Q. Wang, and H. Mei, "A Runtime Model Based Monitoring Approach for Cloud," in *IEEE 3rd International Conference on Cloud Computing*, Jul. 2010, pp. 313 –320.

[7] Nagios Enterprises, "Nagios," 2013, available at: http://www.nagios.org/. access in: May 2013.

[8] Zabbix, "Zabbix," 2013, available at: http://www.zabbix.com/. accessed in: May 2013.

[9] Tobias Oetiker, "MRTG," 2013, available at: http://oss.oetiker.ch/mrtg/. accessed in: May 2013.

[10] OpenStack Community, "Ceilometer," 2013, available at: https://wiki.openstack.org/wiki/Ceilometer/. accessed in: May 2013.

[11] Amazon, "Amazon EC2 (API Version 2012-07-20)," 2012, available at: http://aws.amazon.com/archives/Amazon-EC2/. accessed in: May 2013.

[12] Open Grid Forum, "GFD.183 OCCI Core (v1.1)," 2013, available at: http://ogf.org/documents/GFD.183.pdf. accessed in: May 2013.

[13] DMTF Cloud Management Working Group (CMWG), "DSP0263 - CIMI Model and REST Interface over HTTP," 2013, available at: http://dmtf.org/standards/cloud/. accessed in: May 2013.

[14] OpenStack Community, "OpenStack," 2013, available at: http://www.openstack.org/. accessed in: May 2013.

[15] ——, "OpenStack API," 2013, available at: http://api.openstack.org/. accessed in: May 2013.