Computer Standards & Interfaces 46 (2016) 15–22

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Application level interface for a cloud monitoring service

Augusto Ciuffoletti

Department of Computer Science, University of Pisa, Pisa, Italy

ARTICLE INFO

Article history:
Received 26 January 2015
Received in revised form 8 January 2016
Accepted 8 January 2016
Available online 15 January 2016

Keywords:
Monitoring of cloud resources
Monitoring-as-a-Service
Open Cloud Computing Interface (OCCI)
RESTful service
Cloud interoperability

ABSTRACT

We introduce a new paradigm, based on an extension of the Open Cloud Computing Interface (OCCI), for the *on-demand* monitoring of the cloud resources provided to a user. We have extended the OCCI with two new subtypes of core entities: one to collect the measurements and the other to process them. The user can request instances of such entities to implement a monitoring infrastructure.
The paradigm does not target a specific cloud model, and is therefore applicable to any kind of resource provided *as a service*. The specifications include only the minimum needed to describe a monitoring infrastructure, thus making this standard extension simple and easily adoptable. Despite its simplicity, the model is able to describe complex solutions, including private/public clouds, and covers both infrastructure and application monitoring.
To highlight the impact of our proposal in practice, we have designed an engine that deploys a monitoring infrastructure using its OCCI-compliant descriptions. The design is implemented in a prototype that is available as open source.

1. Introduction

One of the most characteristic features of cloud computing is the *on-demand* delivery of IT resources. The interface between the user, who issues resource requests, and the provider, who synthesizes and delivers them, is a critical component. Aside from the existence of a graphical user interface, users describe the features of the resources they need via the Application Programming Interface (API).

From a provider's perspective, an API should ensure the control of the kinds of services offered to the user, i.e. the resources it offers, their features, and how resources communicate with each other and with the outside. This means that APIs tend to diverge from one another, following the legitimate intent of the providers to differentiate and improve their services.

However, this is a reason of concern for the user. In their paper, published in 2010 [2], Armbrust et al. stated that *business continuity* and *data lock-in* were primary obstacles for cloud adoption. In both cases, the existence of a uniform interface across providers helps, since being able to switch smoothly from one to another encourages cloud adoption. Therefore, the interests of both the user and the provider find, to some extent, a point of convergence.

Since 2009, there has been a noticeable effort to standardize an API for cloud computing. A wiki reference to these developments is at www.cloud-standards.com, where the representatives active in each organization report their current progress. However, major providers (Amazon, Microsoft, and Google) have been motivated to differentiate their products in order for them to become the *de facto* standard, also by participating in forums dedicated to designing a standard (such as the Distributed Management Task Forum, DMTF).

Somewhat in parallel with this trend is the evolution of the Open Cloud Computing Interface (OCCI) protocol, which is independent from providers and open to the contribution of academic and industrial research. OCCI provides an interoperable foundation for the cloud API, while allowing the widest range of options for the services offered. OCCI adoption alone does not guarantee interoperability, but is a fundamental step in the process.

The *Core* specification of the OCCI protocol [19] provides an abstract representation of the entities available to the user as cloud resources. It is a generic schema that defines the API of the service, and lays the foundations for further applications. For instance, the Infrastructure as a Service (IaaS) API is described in a companion document [20] that introduces specialized entities representing *Compute*, *Storage*, and *Network* resources. The term *extension* is used to indicate a document that specializes the *Core* model to a certain field of application, as for the IaaS document referenced above.

The OCCI working group is developing an extension of the core API that allows the user to describe a monitoring infrastructure in the cloud. It has been recently proposed as an OGF document and can be found in the group repository at https://redmine.ogf.org/projects/occi-wg/repository. It is intended for those users that want control over cloud resources, including being able to measure how well these resources perform [22]. This perspective is not only related to the *pay-per-use* nature of cloud services, but is also of interest in private or federated environments. The evolution of *cloud sourcing* [18] towards business-critical applications [14] emphasizes this need, so that the availability of a customized cloud monitoring service (also known as *Monitoring-as-a-Service* [23]) is a discriminating factor in the selection of a provider.

http://dx.doi.org/10.1016/j.csi.2016.01.001
0920-5489/

In this paper, we investigate the OCCI extension for *resource monitoring*. First, we show how it fits in with the existing literature and current commercial solutions. Next, we describe the *extension* and demonstrate its soundness with use cases and examples. Finally, we describe the design and the development of an engine that deploys a monitoring infrastructure *on-demand*, thus providing a practical solution that we believe opens the way to industrial-grade solutions.

1.1. From cloud monitoring to monitoring-as-a-service

Cloud monitoring has been strongly influenced by an extensive study of distributed monitoring frameworks [13,16,21,23,27]. Following a commonly adopted model, the provider is responsible for implementing the monitoring infrastructure, and the user *subscribes* to receive the data. However, a *Monitoring-as-a-Service* interface should allow the user to have control over the monitoring infrastructure, not just to subscribe to available data.

This claim is endorsed by a number of works that do not directly focus on the resources that are provided *on-demand*. In [22], the authors illustrate how monitoring a business process requires a highly adaptive monitoring framework. From a different perspective, a plant automation [26] study illustrates a monitoring framework whose requirements include the interoperability of heterogeneous systems. An investigation into self-adaptive monitoring, which adjusts itself depending on the situation, is proposed in [17]. Although such works do not refer to an implementation *in the cloud*, they support the claim that monitoring frameworks cannot be limited to a *publish-subscribe* model.

In fact, all major cloud providers offer an API to configure resource monitoring and there is a clear trend towards giving the user the tools to customize this service. Windows Azure enables users to configure auto-scaling and alerts based on defined metrics. With Google Cloud, Amazon AWS, and Rackspace users can define custom metrics, which can be used to produce alerts. In particular, Rackspace allows a server to be polled from various geographical regions. Stackdriver offers a service that simplifies and improves the monitoring of AWS resources. With all these providers, the service is limited to the production of measurements, with a limited range of asynchronous user-defined triggered actions. One step ahead is Openstack which has an ongoing *Monitoring as a Service* project named MONaaS. MONaaS is based on a preexisting service used for billing, and offers many ways to process data, like a cloud service that polls cloud resources and synthesizes aggregated metrics. CompatibleOne is a research project that follows a similar approach. It aims to provide the widest possible compatibility with existing cloud computing platforms, thus making it a unique tool to interact with all major providers. The interface embeds tools to describe a monitoring framework, including a specialized *monitoring agent* that oversees the monitoring *probes* [16].

However, there is a significant interoperability problem. If we consider the EGI project http://www.egi.euhttp://www.egi.eu, which gathers the European scientific grids into a unique infrastructure, the adoption of a common interface for cloud services is mandatory when monitoring a heterogeneous system. With the adoption of OCCI as the interface for all the IaaS offered by all partners in the project, EGI provided the way for a uniform test of all infrastructures in the SAM system. Although the EGI case regards cloud management, it reflects the same problem experienced by users who want to deploy a complex infrastructure in a multi-tenant environment, or in the case of hybrid public/private clouds.

Providers are thus investing considerable effort in implementing a flexible resource monitoring service. However, none of these systems is sufficiently flexible to do any more than trigger an action based on measurements, and each of them provides a distinct interface. The purpose of this paper is to go beyond this point.

An extension for monitoring infrastructures can be found in the road-map of the OCCI working group. One important purpose addresses those users that obtain resources from several providers, since they would benefit from a uniform interface for cloud monitoring. A proposed extension has been discussed within the OCCI working group [8], and is summarized in [5]. In [15], other authors extend OCCI monitoring by adding the capacity to recover from a deviation from a defined Service Level Agreement. The CompatibleOne project mentioned above includes the implementation of an API following the OCCI proposed extension. In [25], the authors suggest another extension of the OCCI core model which supports the whole life cycle, from resource delivery to reconfiguration, including monitoring.

This paper aims to demonstrate the soundness of the proposal in [5]. This is done by comparing it against realistic cases, and by designing and implementing a prototype for a specific case.

The design tackles the challenge of synthesizing the monitoring infrastructure using its OCCI-compliant description. We believe that this result is original: among the commercial providers listed above, only OpenStack is figuring out a project for on-demand monitoring, while the others offer a limited range of triggered actions. It supports the proposed scheme, and gives a practical indication for a real scale implementation.

The prototype implementation has many limits: among others, it is not designed for performance, and there are no security measures in place. The rationale behind these options is to keep the design as transparent as possible, trading off code optimization for clarity. But, in combination, the design and the prototype are evidence that there is a real implementation of the abstract paradigm that is behind the schema proposed in [5].

The next section describes the model: first with an overview of the OCCI protocol, followed by a description of the monitoring extension, highlighting how it improves the state of the art.

2. A standard API for on-demand monitoring

The OCCI working group of the Open Grid Forum (OGF) has produced the specifications of an API for the communication between the user and the provider of a cloud computing service: the Open Cloud Computing Interface (OCCI). The aim of the OCCI working group is to foster the convergence towards a widely adopted standard.

A distinguishing feature of the OCCI proposal, which differentiates it from other similar standardization initiatives (such as TOSCA from the OASIS [3], or the CIMI from DMTF [1]), can be found in this bottom-up approach: the OCCI core model defines the foundations for an extensible family of standards.

The interface follows the Representational State Transfer (REST) paradigm [12]: the client and the server exchange *request* and *response* messages which operate HTTP verbs on *entities* that describe cloud resources. Each *entity* instance is associated with a *kind*. *Kind*s are arranged in a tree structure to form a sub-typing hierarchy.

The OCCI working group has defined two core *kinds*: the OCCI *resource*—standing for a *cloud* resource—and the OCCI *link*, which represents a relationship between two OCCI *resources*.

An *entity* can be associated with one or more *mixin*s in order to be further characterized by additional *attributes*, other than those defined by its *kind*. The provider is expected to define *mixin*s to give access to specific terms of the service: for instance, the provider might define a *mixin* for a given software package that may be associated with a *compute entity*.

Each *attribute* is defined by a name in a name-space, a type, the indication of its dynamic mutability, and whether its definition is mandatory or not.

Fig. 1 shows a UML class diagram that describes the OCCI model, and a full explanation can be found in [19].

Besides the core standard which is designed to be long-lasting as valuable standards need to be, there are satellite documents that customize the core protocol to specific applications. For instance, the *IaaS Extension* defines the entities used to describe a cloud infrastructure,