EasyCloud: a rule based toolkit for multi-platform Cloud/Edge service management

Please, cite this paper as:

Cosimo Anglano, Massimo Canonico, Marco Guazzone

"EasyCloud: a rule based toolkit for multi-platform Cloud/Edge service management,"

Proc. of the 2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC), Paris, France, April 20-23, 2020, pp. 188-195.

DOI: 10.1109/FMEC49853.2020.9144821

Publisher: https://ieeexplore.ieee.org/document/9144821

EasyCloud: a rule based toolkit for multi-platform Cloud/Edge service management

Cosimo Anglano, Massimo Canonico and Marco Guazzone
{cosimo.anglano,massimo.canonico,marco.guazzone}@uniupo.it
Computer Science Institute, DiSIT, University of Piemonte Orientale, Italy

***Abstract*—In order to satisfy the demand for collective and collaborative use of the various Cloud/Edge computing platforms available, the Cloud/Edge interoperability is necessary. Unfortunately, due to the specific solutions provided by the major Cloud/Edge providers, right now it is difficult to fully exploit different Clouds/Edges concurrently. In this paper, we aim to fill this gap by proposing EasyCloud, an easy and effective toolkit and user interface able to not only interact with multiple and different Cloud/Edge platforms at the same time but also to provide a rule-based engine where the user can specify what to do in real-time when the workload of the services running on the Clouds/Edges becomes underutilized (e.g., switch-off the service to save money) or overutilized (e.g., switch-on new computational resources to overcome the increased workload). EasyCloud is currently used by researchers, educators, and students with success and its source code is publicly available.**

***Index Terms*—Toolkit, Cloud computing, Edge computing, Intercloud, Orchestration**

I. Introduction

The NIST defines the *Cloud Computing* as "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [1]. Cloud computing providers can offer their services in three different models: (i) *Software as a Service* (SaaS) which is focused on end users of Cloud (i.e., each user just uses the service provided without considering any hardware/software details), (ii) *Platform as a Service* (PaaS), which is focused on application developers (i.e., they have tools and library ready to use to develop their application exploiting the computing resources available in the Cloud platform), and, finally, (iii) *Infrastructure as a Service* (IaaS), which provides direct access to computing hardware resources (i.e., users can access to bare metal resources or to virtualized hardware). For instance, Google Docs [2] and Gmail [3] as well as Salesforce.com CRM [4] are example of SaaS, while Google App Engine [5] and Amazon Elastic Beanstalk [6] are examples of PaaS, and finally, Amazon EC2 [7] and OpenStack [8] are examples of IaaS.

In the last years, with the increasing interest in *Edge computing* [9], [10], many Cloud providers are proposing services dedicated to this emerging paradigm. For example, Amazon CloudFront [11] provides a content delivery network service with low latency as required by edge computing applications, which seamlessly integrates with Amazon EC2. Likewise, Google Cloud [12] (with its Cloud IoT service [13]) and OpenStack (with its variant called OpenStack++ [14]) are proposing solutions targeted to edge applications. This means that the Cloud Computing providers are ready to satisfy the requirements of the Edge computing and, in the next future, probably their distinction will be gradually fuzzy.

In this paper, we focus on the IaaS model that is the bottom-most layer in a Cloud stack where virtual servers with given specification of CPUs, memory, and storage are made available over the network through *Virtual Machines* (VMs) or containers. [1] In an ideal world, a user/educator/researcher should be able to easily interact with the different Cloud providers, in order to figure out which is the best for his purposes or to use concurrently different Cloud providers to get the best from each of them. This feature is called *interoperability*, that is a user can decide when and where to go as needed. This is far from the current reality, since each Cloud provider uses different set of tools, formats, and environments, so the users/educators/researchers tend to get locked-in with a specific Cloud provider [15].

For this reason, we have designed and implemented *EasyCloud*: a toolkit able to interact with multiple Cloud platforms concurrently thanks to a modular architecture and an easy user interface. With our tool, a user can not only exploit the typical basic IaaS services (such as, start and stop VMs, manage network and storage), but also setup rules in order to guarantee the business continuity of its processes running in the Clouds.

In particular, EasyCloud is able to monitor the health of the VMs and, on the basis of user-defined rules, to prevent the occurrence of unwanted events. For example, a rule can decide to clone a VM if its CPU usage level is beyond a fixed threshold. In this way, the workload can be split to multiple servers and hopefully prevent server down event or performance degradation.

Currently, EasyCloud supports three of the most popular Cloud platforms, namely Amazon AWS, Goggle Cloud Platform and OpenStack, and thanks to its modular architecture it can be easily extended to support other Cloud platforms. Furthermore, EasyCloud is ready to support emerging Edge orchestration platforms, including OpenStack++ (where VMs running in a Cloudlet can be managed through the OpenStack API) as well as Kubernetes [16] and its variants specifically

[1] In the rest of this paper, unless otherwise specified, we refer to both traditional VMs and containers as simply VMs.

targeted to edge applications (e.g., [17]). Finally, to enable inexperienced and non developer users to easily manage their VMs with our toolkit, EasyCloud also provides an intuitive and interactive textual user interface.

EasyCloud is written in the Python programming language, which ensures maximum portability and fast development. To foster research and provide reproducibility, its source code is publicly available on [18].

In the rest of the paper, we will discuss in detail all the characteristics and features of EasyCloud. In particular, in Section II, we discuss the related work presented in the scientific literature, while in Section III and in Section IV, we illustrate the architecture and design of our toolkit. Section V shows the way EasyCloud can be used in real testbeds by users without any experience on Cloud computing platforms. Finally, Section VI concludes the paper and presents the future steps of our toolkit.

II. Related Work

As mentioned in Section I, many Cloud providers have developed their own platforms featuring proprietary interfaces which is not a problem as long as a single provider can fully satisfy its customers. However, the lack of standardization for interconnecting platforms makes it difficult for customers who need the combined services or resources of multiple providers. This often results in users being locked into specific providers and platform [19]. This issue has led to the idea of interconnected clouds, also known as *interclouds* or *Cloud Federation*: a cloud of clouds where the advantages (in terms of service cost [20]–[22], quality of service [23], [24], performance [25], and so on) of different cloud providers can be exploit by a single user.

The work in [26] is a survey concerning the intercloud projects proposed in the scientific literature. In this survey, the authors conclude that all the projects studied are not able to implement the interoperability between different clouds and only in the future we will be able to have a real federated cloud environments when more cloud providers will emerge with a standard interface for their services. We have tried to download/install/configure and use the 11 intercloud projects mentioned in the paper, in order to compare our toolkit with the state of the art. Unfortunately, this has not been possible due to various reasons, such as software or source code not available, web site of the project unreachable or software not working.

Conversely, a very promising project (not mentioned in the survey) is Cloudmesh [27]: a tool which enables the access to multi-cloud environments such as Amazon AWS, Microsoft Azure [28], Google Cloud [12] and OpenStack. In particular, Cloudmesh provides a command line shell allowing to boot VMs and to switch between clouds but it does not provide any feature besides the basic ones offered by the cloud platforms. It is basically a wrapper on top of the various cloud platforms, without any monitoring features or scaling mechanisms, so important in a distributed computing system [29], [30].

EasyCloud is an evolution of our previous toolkits [31], [32] used by various researchers, students and cloud users to interact easily with the popular cloud platforms (including, Amazon AWS, Google Cloud and OpenStack). With respect to our previous work, EasyCloud presents a more modular and pluggable architecture to easily integrate the support for additional cloud platforms, and its software components have been rewritten from scratch and improved. These features will be discussed in more detail in Section III.

With respect to the above works, EasyCloud, to the best of our knowledge, is the only multi-platform toolkit for cloud and edge systems able to provide both the basic functionality (such as, running a VM) and the advanced functionality (such as, preventing failures and automatically scaling in and out the computational resource provided).

In addition to the above works, different application workflow management and orchestration platforms have been proposed (e.g., Nomad [33] and Marathon [34]). For instance, Nomad is a workload orchestrator that can deploy and manage containerized, virtualized, and standalone applications in a multi-cloud environment using a single, unified workflow, and with the ability to monitor the progress of the submitted jobs and to automatically recover from machine failures. Unlike these platforms, EasyCloud works at a lower level of abstraction, by focusing on the life-cycle management of VMs rather than applications. Moreover, while these platforms may provide features that EasyCloud still does not provide (e.g., defining and running application workflows), to the best of our knowledge, they lack of features that EasyCloud already provides, like the ability to define custom rules for VM monitoring and management. Finally, these platforms often require setting up a suitable cluster of nodes where to run applications, for instance by installing and running specific client/server software agents on every node in the managed cluster. Instead, EasyCloud is very easy to install and manage as it can run as a standalone program on a single machine and it does not require the execution of any specific cluster manager (even if it can interact with some of them to manage their VMs).

III. Architecture

EasyCloud has been specifically conceived with the following goals in mind:

- *platform-independence*, to deal with the heterogeneity of the cloud/edge computing platforms;
- *interoperability*, to deal with the multi-provider peculiarity of cloud/edge systems where multiple cloud/edge providers can serve the same geographic area and when an edge/cloud interplay is necessary;
- *modularity*, to easily support and plug-in new cloud and edge computing platforms;
- *ease of use*, to allow a quick and intuitive management of the VM instances running on the underlying cloud/edge infrastructure.

We achieves the above design goals as follows:

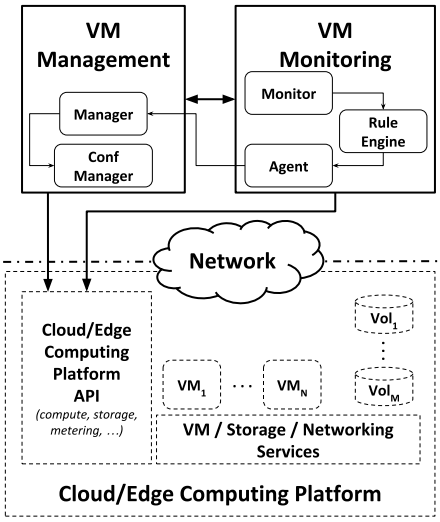

Fig. 1. The architecture of EasyCloud. Dashed boxes denote components outside the scope of this paper. The connection $C_1 \rightarrow C_2$ means that component C_1 interacts with component C_2. Symbols VM_i and Vol_j denote a particular VM instance and volume, respectively.

- platform-independence: EasyCloud provides a unified interface to interact with different cloud/edge platforms, thus enabling its clients to write platform-independent code;
- interoperability: EasyCloud is able to transparently communicate with multiple and different cloud/edge platform during the same running session;
- modularity: EasyCloud consists of one or more pluggable and extensible software components so that its functionality can be easily replaced or extended to cope with the heterogeneity of the existing and future cloud/edge platforms;
- ease of use: the ability of EasyCloud to support multiple platforms enables code reuse and frees the developer to learn all the API provided by the various cloud/edge platforms; also, its intuitive and interactive user interface allows inexperienced users to easily manage their VMs.

The architecture of EasyCloud consists of the following subsystems, shown in Figure 1 as solid boxes (while dashed boxes denote components of the cloud/edge infrastructure): the *VM Management* subsystem (which provides VM management services) and the *VM Monitoring* subsystem (which provides VM monitoring and metering functionality).

Each subsystem consists of one or more pluggable and extensible components so that its functionality can be easily replaced or extended to cope with the heterogeneity of the existing and future cloud/edge platforms. By providing a platform-agnostic interface, these subsystems hide the details about the interactions with the services provided by the cloud/edge platform. This way, EasyCloud can support multiple cloud/edge platforms by simply providing the concrete implementation of its interface for every cloud/edge platforms that is to be supported.

In the rest of this section, we discuss in detail the two subsystems mentioned before: the *VM Management* in Section III-A and the *VM Monitoring* in Section III-B.

A. VM Management

The *VM Management* subsystem provides a platform-agnostic interface for the life-cycle management of VM instances and volumes hosted by the cloud/edge infrastructure, by interacting with the virtualization services provided by the cloud/edge platform (e.g., with the OpenStack's Compute Nova component [35]). Common interactions with this subsystem comprise creating, restarting and destroying VM instances, attaching and detaching volumes to/from VM instances, and associating and disassociating network IP addresses to/from VM instances.

This subsystem comprises two main components: the *Manager*, which deals with the management of VMs (e.g., starting and stopping VM instances), and the *Conf Manager*, which is used to query and set configuration parameters of the cloud/edge platforms (e.g., the user credentials to access the platform).

To support the virtualization services of multiple cloud/edge infrastructures, EasyCloud relies on the *Apache libcloud* [36] library, which provides a unified API for interacting with many of the popular cloud service providers, including Amazon EC2 [7], Google Compute Engine [37] and OpenStack, just to name a few.

B. VM Monitoring

The *VM Monitoring* subsystem provides a platform-independent interface for gathering metric data from the monitored VM instances and triggering actions according to user-defined policies defined against those collected data, by interacting with the telemetry services provided by the cloud/edge platform (e.g., with the Amazon CloudWatch [38], for the Amazon AWS platform). Typical interactions with this subsystem include the gathering of usage data for a given running VM instance (e.g., its CPU and memory utilization).

To do so, this subsystem periodically (1) gathers metric data from the monitored VM instances, (2) checks whether the collected data activate one or more user-defined rules (see below), and then (3) triggers the actions associated with the activated rules by interacting with the *VM Management* subsystem.

This subsystem consists of three main components, namely the *Monitor*, the *Rule Engine*, and the *Agent*, which run as threads and communicate with each other by sending messages stored in synchronized queues.

The *Monitor* component periodically queries the telemetry services of the cloud/edge platform to gather metric data of the monitored VM instances (e.g., the CPU load), and sends collected data to the *Rule Engine* component for further processing. Specifically, as shown in Algorithm 1, for each VM instance v to be monitored (line 2) and for each metric m to be collected (line 3), the *Monitor* gathers a maximum of n data of metric m (with a granularity of g seconds) related to VM instance v collected by the cloud/edge platform in the

Algorithm 1 The data collection algorithm used by the *Monitor* component.

1: **procedure** MONITORVMS(V, M, n, g)
2: **for all** monitored VM instance v in V **do**
3: **for all** metric m to gather in M **do**
4: $t \leftarrow$ GETCURRENTTIME()
5: $\langle t_s, t_e \rangle \leftarrow$ COMPUTESTARTSTOPTIME(t, n, g)
6: $D \leftarrow$ COLLECTDATA($v, m, \langle t_s, t_e \rangle, n, g$)
7: **if not** EMPTY(D) **then**
8: SENDTORULEENGINE(v, m, D)
9: **end if**
10: **end for**
11: **end for**
12: **end procedure**

```
{
    "name": "cpu_load_gt_60%",
    "target": "cpu_load",
    "operator": ">",
    "threshold": 60.0,
    "action": "clone"
},
{
    "name": "free_mem_lt_500MB",
    "target": "memory_free",
    "operator": "<",
    "threshold": 500000000.0,
    "action": "clone"
}
```

Fig. 2. A sample rule base (with two rules) used by the *Rule Engine* component.

time window $[t_s, t_e]$ (lines 4–6), and sends them (if any) to the *Rule Engine* for applying user-defined policies (lines 7–9). The set V of the VM instances to be monitored and the metrics M to be collected can be changed at runtime. Some functions of the *Monitor* component, like gathering metric data related to a specific VM instance, require the interaction with the telemetry services provided by the cloud/edge platform where that instance is running. EasyCloud provides several built-in *Monitor* components to support the most popular cloud/edge platforms including Amazon AWS, Google Compute Platform and OpenStack.

The *Rule Engine* component receives data from one or more *Monitors* and fed them to its rule engine to decide which actions should be triggered against a given rule base. EasyCloud supports threshold-based rules as follows: "⟨*name*⟩*:* ***if*** ⟨*metric-data*⟩ ⟨*operator*⟩ ⟨*threshold*⟩ ***then*** ⟨*action*⟩." Each of these rules is specified as a JSON object [39] where the *name* field specifies a symbolic name of the rule, the *target* field specifies the metric to consider, the *threshold* field specifies the threshold value for the target metric, the *operator* field specifies the operator used for comparing the metric value with the given threshold, and the *action* field specifies which action to trigger if the condition is satisfied.

For instance, the rule base of Figure 2 contains two rules: the first one, called "cpu_load_gt_60%", triggers the cloning of a certain VM instance when its CPU utilization is greater than 60%, while the second one, named "free_mem_lt_500MB", triggers a cloning of a given VM instance when its available memory is less than 500 MB. As shown in Figure 2, the *Rule Engine*, upon receiving new metric data D for metric m, checks if some rule r in the rule base R (and related to metric m) is satisfied by at least p samples in D (lines 5–9) and, if so, it sends a message to the *Agent* component (see below) to execute the action associated with that rule against a given VM v (line 12).

Algorithm 2 The rule matching algorithm used by the *Rule Engine* component.

1: **procedure** MATCHRULES(R, v, m, D, p)
2: **for all** rule r in R for metric m **do**
3: $c \leftarrow 0$
4: **for all** data d in D **do**
5: $o \leftarrow$ OPERATOR(r)
6: $\tau \leftarrow$ THRESHOLD(r)
7: **if** ISSATISFIED(o, τ, d) **then**
8: $c \leftarrow c + 1$
9: **end if**
10: **end for**
11: **if** $c \geq p$ **then**
12: SENDTOAGENT(v, ACTION(r))
13: **end if**
14: **end for**
15: **end procedure**

Finally, the *Agent* component is in charge of executing the actions as requested by the *Rule Engine* component. First, the *Agent* component examines the received actions to filter out possible duplicates (e.g., resulting from different rules satisfied at the same time) so as to avoid executing the same action on a given VM instance multiple times. Then, it iterates the filtered set of actions and executes each one of them through the *Manager* component. For example, even if, for a given VM instance, both rules of the rule base of Figure 2 are satisfied at the same time (i.e., if both "cpu_load_gt_60%" and "free_mem_lt_500MB" are satisfied by at least p data samples in D), the VM instance will be cloned only once.

IV. DESIGN

In this section, we provide details about the internal design of the subsystems of EasyCloud described in Section III. Specifically, in Section IV-A, we describe the core classes of EasyCloud, while in Section IV-B we describe how to extend those core classes to support a given cloud/edge platform.

We graphically present these classes and their relationships through *Unified Modeling Language* (UML) class diagrams [40], where a *class* is denoted by a box whose label is the class name, and a static relationship among two classes is represented with a line whose style changes with the type of the UML relation. In those class diagrams, we also distinguish between *concrete classes*, which are represented as classes with the name in regular font, and *abstract base classes*, which

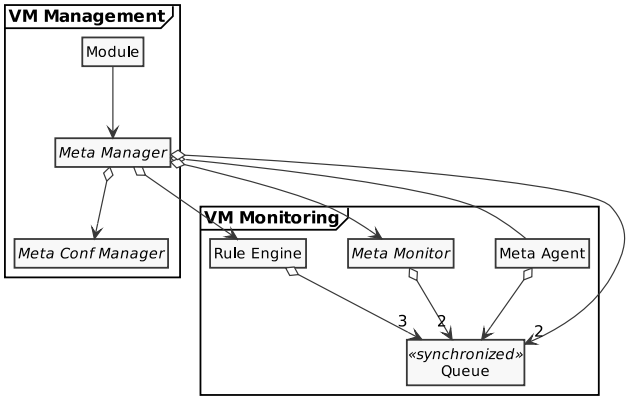

Fig. 3. The UML class diagram of the core classes of EasyCloud.

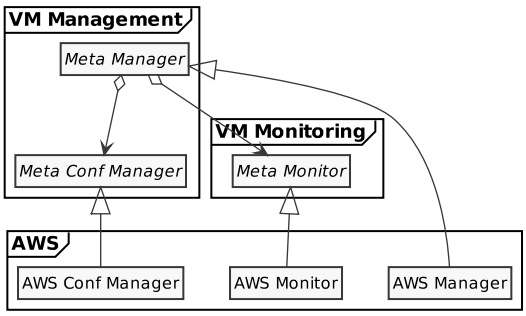

Fig. 4. The UML class diagram of the classes added to EasyCloud to support Amazon AWS.

are represented as classes with the name in italic font. In the rest of this section, unless otherwise stated, we use the terms "concrete class" and "class" interchangeably. Furthermore, in the class diagrams, we show which subsystems the various classes of EasyCloud belong to through frames (whose headings denote subsystems' names), and how those subsystems are related to each other through classes' static relationships.

A. Core Classes

The main core classes of EasyCloud are shown in the class diagram of Figure 3. As shown in the figure, the *VM Management* consists of three core classes, namely the *Meta Manager*, the *Meta Conf Manager* and the *Module*, which provide the functionality described in Section III-A. Specifically, the *Meta Manager* abstract base class models the *Manager* component of the *VM Management* subsystem and provides an interface for managing and querying VM instances and volumes like, for example, starting and stopping VM instances, attaching and detaching volumes, and listing security groups and availability zones. The *Meta Conf Manager* abstract base class models the *Conf Manager* component of the *VM Management* subsystem and is responsible of querying and changing configuration parameters, including those used internally by EasyCloud (e.g., the sampling window size n and granularity g used by Algorithm 1) and those specific to a particular cloud/edge platform (e.g., the user credentials to access the compute services of a given cloud/edge platform). Finally, the *Module* class is responsible of dynamically loading and unloading in EasyCloud the support for a given cloud/edge platform, thus providing users of EasyCloud the flexibility to plug-in a specific cloud/edge platform at runtime, only when necessary.

The remaining classes of Figure 3 provide the functionality offered by the *VM Monitoring* subsystem (as described in Section III-B). Specifically, the *Meta Monitor* abstract base class models the *Monitor* component and it is responsible of gathering metric data of VM instances by running Algorithm 1 and communicating with the telemetry services provided by the involved cloud/edge platforms. The *Rule Engine* class implements the *Rule Engine* component by providing a rule-based engine to trigger actions according to user-defined policies. The *Meta Agent* class models the *Agent* component and provides basic functionality for triggering actions requested by the *Rule Engine* component, like filtering out duplicate actions and performing the remaining actions through the *Manager* component. Finally, as described in Section III-B, most of the *VM Monitoring* components run as separate threads. These threads communicate with each other by sending messages to an instance of the *Queue* class, which models a multi-producer, multi-consumer queue and implements suitable locking semantics to enable a group of threads to safely exchange messages.

B. Supporting a Cloud/Edge Platform

EasyCloud already supports three of the most popular cloud platforms, namely *Amazon AWS* [41], *Google Cloud* [12], and *OpenStack* [8], as well as its deployment for the *Chameleon* testbed [42] (including the management of bare-bone instances).

In this section, we show how to extend EasyCloud to support a new cloud/edge platform, by using as a case study the *Amazon AWS* platform.

In EasyCloud, to integrate a new cloud/edge platform, it is necessary to implement the interfaces of the abstract base core classes described in Section IV-A, by developing subclasses that extend those core classes. In particular, for the Amazon AWS case study, this integration requires interacting with *Amazon EC2* compute services [7] and with *Amazon CloudWatch* telemetry services [38] to provide the functionality of the *VM Management* and *VM Monitoring* subsystems, respectively.

In Figure 4, we show the subclasses that have been added to EasyCloud to support Amazon AWS (see classes inside the frame *AWS*) as well their static associations with the relevant core classes of EasyCloud. Furthermore, in Figure 5, we show an outline of the more relevant methods of the interface provided by the core classes of EasyCloud that those subclasses implement, by using a simplified Python-like pseudo-code where we omit implementation details like class constructors.

As shown in Figure 4, the *AWS Conf Manager* class is a concrete subclass of the *Meta Conf Manager* abstract base class that implements its interface to query and change platform-specific configuration parameters, like the

```
class AWSConfManager(MetaConfManager):
  def read_login_data(self):
    ...
  def read_platform_options(self):
    ...
  ...

class AWSManager(MetaManager):
  def connect(self):
    ...
  def _platform_list_all_images(self):
    ...
  def _platform_create_new_instance(self,
    instance_name, image, instance_type,
    commands_queue):
    ...
  def _platform_create_volume(self, volume_name,
    volume_size):
    ...
  def _platform_attach_volume(self, volume,
    instance):
    ...
  def _platform_associate_floating_ip(self,
    floating_ip, instance):
    ...
  def _platform_get_monitor(self,
    commands_queue, measurements_queue):
    ...
  ...

class AWSMonitor(MetaMonitor):
  def connect(self):
    ...
  def _get_metric_values(self, instance, metric,
    granularity, limit):
    ...
  ...
```

Fig. 5. Implementation outline for the integration of the Amazon AWS platform.

user credentials to access Amazon EC2 resources (see method `read_login_data()` in Figure 5).

The *AWS Manager* class is a concrete subclass of the *Meta Manager* abstract base class that communicates with Amazon EC2 to provide VM management services, like creating a new VM instance or volume (see methods `_platform_create_new_instance()` and `_platform_create_volume()` in Figure 5). To do so, this classes relies on the various functions provided by the *Apache libcloud* API. For instance, the `_platform_create_new_instance()` method uses the `create_node()` function of libcloud to create a new VM instance.

Finally, the *AWS Monitor* class is a subclass of the *Meta Monitor* abstract base class that interacts with Amazon CloudWatch to gather metric data related to VM instances (e.g., see method `_get_metric_values()` in Figure 5). To do so, this class relies on the various functions provided by the *Amazon AWS SDK for Python* [43] (formerly known as *Boto3*). For instance, the implementation of the `_get_metric_values` method uses Amazon Boto3's `get_metric_*()` functions (e.g., `get_metric_statistics()`) to collect at most `limit` data related to a given `metric` and VM `instance`,

```
--- Missing packages ---

The following packages are required by the Amazon Web Services (Libcloud) module:

- apache-libcloud
- boto3

If these packages are not installed, this module won't be loaded.
Do you want to install them through pip? (Yes/No)
>
```

Fig. 6. If EasyCloud detects missing libraries, it proposes to install them automatically.

```
--- Library installer ---

✔ All the packages for Google Cloud Platform (Libcloud)
were successfully installed!

- google-auth
- google-auth-oauthlib
- google-cloud-monitoring


Press a key to continue...
```

Fig. 7. EasyCloud notifies the user that all libraries required by Google Cloud Platform has been successfully installed.

and according to a given time `granularity` (in seconds).

V. Interactive User Interface

To enable inexperienced and non developer users to easily manage their VMs with our toolkit, EasyCloud also provides an intuitive and interactive Textual User Interface (TUI).

With this TUI, when a user launches our toolkit for the first time, it checks if every software requirements are satisfied. If there are missing libraries, EasyCloud asks to install them as illustrated in Figure 6. The missing libraries will be installed automatically if the user wants to and then, eventually, EasyCloud notifies the users of the outcome of the installation procedure (see Figure 7). Once all software requirements are satisfied, the user is ready to choose which cloud/edge platforms use first as we can see in the main menu illustrated in Figure 8. In Figure 9, we can see the list of all features provided by EasyCloud that has been discussed in the previous sections. The interested reader who wants to try our tool, he/she has just to obtain credentials from a cloud/edge platform (they can be obtained for free from the main platforms such as AWS, Google Cloud Platform and OpenStack), downloads the source code available from the EasyCloud website [18] and follows the detailed instructions

```
******************** EasyCloud ********************

Select a platform
--------------------------
1) Amazon Web Services
2) Google Cloud Platform
3) OpenStack
4) Close application

Please make a choice:
>
```

Fig. 8. The user can choose between three different Cloud computing platforms.

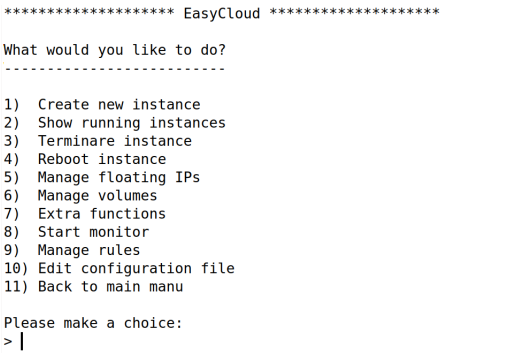

Fig. 9. List of features provided by EasyCloud for each Cloud platform.

written in the README file. As mentioned before, it is not necessary any cloud skill or experience to use EasyCloud.

VI. Conclusions

In this paper, we present EasyCloud, a toolkit able to interact with different and multiple Cloud platforms at the same time in an easy and effective manner. Thanks to its modularity, our toolkit is able to easily integrate different platforms as long as they expose their API. Furthermore, thanks to its intuitive user interface, EasyCloud is suited also to students, educators and researchers who want to approach to Cloud computing without considering the details of each implementations.

Concerning the future work, we plan to integrate in EasyCloud other Cloud platforms such as *Microsoft Azure* [28] and *IBM Cloud* [44]. Moreover, we want to add new features like resource reservation as proposed by some Cloud provider. Also, we plan to extend our Rule Engine component by integrating in EasyCloud the *Intellect* package [45] to support a richer set of rule types. Thanks to the modularity of our toolkit, we also plan to extend the providers to be fully complaint with native Edge computing platforms such as *MicroK8s* [46] and *EdgeX* [47], just to name a few. Furthermore, we plan to integrate EasyCloud in our previous work dealing with cloud/edge infrastructure management [48]–[50] and experimentation [51]. In order to involve new users, we plan to propose EasyCloud to the most popular cloud communities such as *Chameleon users group* [52] and *Google Cloud Faculty Community group* [53], just to name a few. Last but not least, we are planning to design and implement a web interface able to make the usage of EasyCloud even more easy without any software installation.

Acknowledgment

This research has a financial support of the Università del Piemonte Orientale and of the INdAM – GNCS Project 2020.

References

[1] P. M. Mell and T. Grance, "Sp 800-145. the nist definition of cloud computing," Gaithersburg, MD, USA, Tech. Rep., 2011.

[2] Google, "Docs," available: https://www.google.com/docs. Accessed: Jan 25, 2020.

[3] ——, "Gmail," available: https://www.google.com/gmail. Accessed: Jan 25, 2020.

[4] Salesforce.com, "CRM," available: https://www.salesforce.com/crm/. Accessed: Jan 25, 2020.

[5] Google, "App Engine," available: https://cloud.google.com/appengine. Accessed: Jan 25, 2020.

[6] Amazon AWS, "Amazon Elastic Beanstalk," available: https://aws.amazon.com/elasticbeanstalk/. Accessed: Jan 25, 2020.

[7] ——, "Amazon EC2," available: https://aws.amazon.com/ec2/. Accessed: Jan 25, 2020.

[8] OpenStack Foundation, "OpenStack," available: https://www.openstack.org/. Accessed: Jan 25, 2020.

[9] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, 2015.

[10] C. Anglano, M. Canonico, and M. Guazzone, "Profit-aware resource management for edge computing systems," in *Proc. of the 1st International Workshop on Edge Systems, Analytics and Networking (EdgeSys)*. Munich, Germany: ACM, 2018, pp. 25–30.

[11] Amazon AWS, "Amazon CloudFront," available: https://aws.amazon.com/cloudfront/. Accessed: Jan 25, 2020.

[12] Google, "Cloud," available: https://cloud.google.com/. Accessed: Jan 25, 2020.

[13] ——, "Cloud IoT," available: https://cloud.google.com/solutions/iot. Accessed: Jan 25, 2020.

[14] K. Ha and M. Satyanarayanan, "Openstack++ for cloudlet deployment," Carnegie Mellon University, Tech. Rep. CMU-CS-15-123, Aug. 2015.

[15] N. K. Sehgal, P. C. P. Bhatt, and J. M. Acken, *Cloud Computing with Security: Concepts and Practices*. Springer, 2020.

[16] Kubernetes, "Kubernetes: A production-grade container orchestration," available: https://kubernetes.io/. Accessed: Jan 25, 2020.

[17] KubeEdge, "KubeEdge: An open platform to enable Edge computing," available: https://kubeedge.io/. Accessed: Jan 25, 2020.

[18] C. Anglano, M. Canonico, and M. Guazzone, "EasyCloud repository," https://gitlab.di.unipmn.it/DCS/easycloud/, 2020, [Online; accessed 25-Jan-2020].

[19] D. G. Kogias, M. G. Xevgenis, and C. Z. Patrikakis, "Cloud federation and the evolution of cloud computing," *Computer*, vol. 49, no. 11, pp. 96–99, nov 2016.

[20] M. Guazzone, C. Anglano, and M. Canonico, "Energy-efficient resource management for cloud computing infrastructures," in *Proc. of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Athens, Greece: IEEE Computer Society, Nov 2011, pp. 424–431.

[21] M. Guazzone, C. Anglano, and M. Sereno, "A game-theoretic approach to coalition formation in green cloud federations," in *Proc. of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. Chicago, IL, USA: IEEE Computer Society, May 2014, pp. 618–625.

[22] C. Anglano, M. Guazzone, and M. Sereno, "Maximizing profit in green cellular networks through collaborative games," *Computer Networks*, vol. 75, Part A, pp. 260–275, 2014.

[23] C. Anglano, M. Canonico, P. Castagno, M. Guazzone, and M. Sereno, "A game-theoretic approach to coalition formation in fog provider federations," in *Proc. of the 3rd International Conference on Fog and Mobile Edge Computing (FMEC)*. Barcelona, Spain: IEEE, April 2018, pp. 123–130.

[24] ——, "Profit-aware coalition formation in fog computing providers: A game-theoretic approach," *Concurrency and Computation: Practice and Experience*, 2019, in press.

[25] M. Guazzone, C. Anglano, and M. Canonico, "Exploiting VM migration for the automated power and performance management of green cloud computing systems," in *Proc. of the 1st International Workshop on Energy-Efficient Data Centres (E2DC)*, ser. Lecture Notes in Computer Science, vol. 7396. Madrid, Spain: Springer Berlin Heidelberg, May 2012, pp. 81–92.

[26] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Comput. Surv.*, vol. 47, no. 1, May 2014. [Online]. Available: https://doi.org/10.1145/2593512

[27] G. von Laszewski, B. Abdul-Wahid, F. Wang, H. Lee, G. C. Fox, and W. Chang, "Cloudmesh in support of the nist big data architecture framework," Technical report, Indiana University, Bloomingtion IN 47408, USA, Tech. Rep., 2017.

[28] Microsoft, "Azure," available: https://azure.microsoft.com/. Accessed: Jan 25, 2020.
[29] C. Anglano, M. Canonico, and M. Guazzone, "FC2Q: Exploiting fuzzy control in server consolidation for cloud applications with SLA constraints," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 4491–4514, 2015.
[30] ——, "FCMS: A fuzzy controller for CPU and memory consolidation under SLA constraints," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 5, p. e3968, 2017.
[31] M. Canonico, A. Lombardo, and I. Lovotti, "Cloudtui: A multi cloud platform text user interface," in *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, ser. ValueTools '13. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, p. 294–297. [Online]. Available: https://doi.org/10.4108/icst.valuetools.2013.254413
[32] M. Canonico and D. Monfrecola, "Cloudtui-fts: A user-friendly and powerful tool to manage cloud computing platforms," in *Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools*, ser. VALUETOOLS'15. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016, p. 220–223. [Online]. Available: https://doi.org/10.4108/eai.14-12-2015.2262718
[33] HashiCorp, "Nomad," available: https://nomadproject.io/. Accessed: Jan 25, 2020.
[34] D2iQ, Inc., "Marthon," available: https://mesosphere.github.io/marathon/. Accessed: Jan 25, 2020.
[35] OpenStack, "OpenStack Compute (nova)," available: https://docs.openstack.org/nova/latest/. Accessed: Jan 25, 2020.
[36] Apache Software Foundation, "Apache Libcloud," available: https://libcloud.apache.org. Accessed: Jan 25, 2020.
[37] Google, "Google Compute Engine," available: https://cloud.google.com/compute/. Accessed: Jan 25, 2020.
[38] Amazon AWS, "Amazon CloudWatch," available: https://aws.amazon.com/cloudwatch/. Accessed: Jan 25, 2020.
[39] JSON, "Introducing JSON," available: https://www.json.org. Accessed: Jan 25, 2020.
[40] OMG et al., "OMG Unified Modeling Language (OMG UML), version 2.5," Object Management Group, Inc., Specification formal/2015-03-01, Jun 2015, available from: http://www.omg.org/spec/UML/2.5. Accessed: Jan 25, 2020.
[41] Amazon, "Amazon Web Services," available: https://aws.amazon.com. Accessed: Jan 25, 2020.
[42] K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, J. Mambretti, P. Rad, and P. Ruth, *Chameleon: A Scalable Production Testbed for Computer Science Research*. CRC Press, 2019, ch. 5.
[43] Amazon AWS, "AWS SDK for Python (Boto3)," available: https://aws.amazon.com/sdk-for-python/. Accessed: Jan 25, 2020.
[44] IBM, "Cloud," available: https://cloud.ibm.com/. Accessed: Jan 25, 2020.
[45] M. J. Walsh, "Intellect: A Domain-specific language and Rules Engine for Python," available: https://github.com/nemonik/Intellect. Accessed: Jan 25, 2020.
[46] Canonical, "MicroK8s," available: https://microk8s.io/. Accessed: Jan 25, 2020.
[47] EdgeX Foundry, "EdgeX," available: https://www.edgexfoundry.org/. Accessed: Jan 25, 2020.
[48] L. Albano, C. Anglano, M. Canonico, and M. Guazzone, "Fuzzy-Q&E: Achieving QoS guarantees and energy savings for cloud applications with fuzzy control," in *Proc. of the* 3^{rd} *International Cloud and Green Computing Conference (CGC)*. Karlsruhe, Germany: IEEE Computer Society, Sept 2013, pp. 159–166.
[49] C. Anglano, M. Canonico, and M. Guazzone, "Online user-driven task scheduling for FemtoClouds," in *Proc. of the* 4^{th} *International Conference on Fog and Mobile Edge Computing (FMEC)*. Rome, Italy: IEEE, June 2019, pp. 5–12.
[50] ——, "WQR-UD: An online scheduling algorithm for FemtoClouds," in *Proc. of the* 12^{th} *EAI International Conference on Performance Evaluation Methodologies and Tools*, ser. VALUETOOLS 2019, 2019, pp. 179–182.
[51] ——, "Prometheus: A flexible toolkit for the experimentation with virtualized infrastructures," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 11, p. e4400, 2018.
[52] Chameleon, "Users," available: https://lists.chameleoncloud.org/mailman/listinfo/users. Accessed: Jan 25, 2020.
[53] Google, "Cloud Faculty Community," available: https://groups.google.com/forum/\#\!forum/googlecloudfaculty. Accessed: Jan 25, 2020.