Extensible Declarative Management of Cloud Resources across Providers

Oleksii Serhiienko, Panagiotis Gkikopoulos and Josef Spillner
Service Prototyping Lab (blog.zhaw.ch/splab/)
Zurich University of Applied Sciences
Winterthur, Switzerland
{serh,pang}@zhaw.ch, josef.spillner@zhaw.ch

***Abstract*—Tags and labels are annotations on resources in many commercial public cloud models. Little is known about the extent of tagging in commercially relevant settings and there is an absence of automated software to handle tags. We show that by introducing an extensible tag management middleware based on cloud functions, tags can be turned into a powerful declarative means of cloud management. Our universal connector middleware is demonstrated by a typical deployment administration scenario involving both AWS and Google Cloud Platform services.**

***Index Terms*—multi-cloud, cloud management, cloud functions, tagging**

I. Introduction and Problem Statement

Multi-cloud and cross-cloud resource management is a field of increasing interest to researchers and industry which has emerged out of general multi-cloud abstraction research. It covers the technical, but also operational and financial aspects of centralised policies on combining software, platform and infrastructure services. Its goal is to maximise the utility for the users (e.g. in companies) while minimising cost, redundant subscriptions, stray resources and unaccounted service activation. The different concepts and terminologies which vary greatly between commercial cloud providers turn multi-cloud management into a non-trivial process [1], [2].

Within this context, many companies resort to using tags to differentiate their cloud resources. Tags such as `stage:testing` or `eol:may2019` are often used or interpreted in practice as notes for administrators and DevOps teams. Eligible tag names and scopes again differ between cloud providers, leading to ad-hoc decisions on which tags to put on which resources. The non-systematic employment of tags then leads to operational mistakes and does not fully exploit the automation potential in cloud environments.

To avoid this problem, we contribute a middleware called Universal Connector with two main functions. First, it manages tags in a secure, audible and cost-effective manner. Second, it performs best practices cloud management tasks driven by regulated tags.

In this paper, we first present recent progress on multi- and cross-cloud resource management middleware to convey more background information. The concept and design of the connector are described next, followed by notes on the available implementation and the scenario-guided evaluation. We argue that future multi-cloud management products will greatly benefit from our extensible design based on cloud functions which contrasts current monolithic platforms, and we show the effectiveness of tag-driven management.

Research supported by project Amysta-SaaS/18371.1 by Innosuisse - Swiss Innovation Agency - and an AWS Cloud Credits for Research grant.

II. Background and Related Middleware

Cloud resource management is one particular angle within the general field of cloud management which focuses on allocatable resources and instances, often in combination with orchestration, data, performance, security and trust [3], [4]. The management takes place within a cloud C or from a management system $M \rightarrow C$. Multi-cloud resource management ($M \rightarrow \{C_1, C_2\}$) add challenges to centralised management due to heterogeneous definitions of resources across providers. Often, abstraction layers such as Libcloud or Deltacloud are used to harmonise different APIs and semantics where one access is routed and translated to one provider at a time [5]. Generalised cross-cloud resource management (e.g. $C_1 \rightarrow \{C_1, C_2\} \bigwedge C_2 \rightarrow C_3$) adds parallelism, scheduling, a distributed dimension and the additional need to manage the management system itself with cloud tools [6]. While many multi-cloud and cross-cloud management systems have been built (e.g. Slipstream, Cloudiator, Cloud Pier, Fog.io, Terraform, Ansible, Newrelic, ManageIQ, CloudcheckR [7]), there is no support for regulated tag management and tag-driven resource management in any of them.

For our analysis, we define cloud resources to be any resource that can be provisioned on a provider, for example VM instances, storage buckets or FaaS functions. Likewise, we define tags to be annotations or labels on these resources consisting of a key and a value. Cloud resources can be addressed by type (all), by unique identifiers (single/few) or by tags (some) in the programming models (e.g. programmable infrastructure) and interfaces offered by major commercial providers. Sometimes, the interfaces offers a uniform access with multiple attributes. For instance, the AWS EC2 API contains a `DescribeTags` method which can filter according to the tag key, tag value, tag key-value combination, resource id or resource type. The response to the method contains a list of items, each described with resource type, resource id and matching tag key-value pair.

There appears to be moderate interest in tagging in most developer communities. On StackExchange's ServerFault site, there are 18 results for the method in conjunction with the term EC2 compared to 32 for the common `DescribeInstances`. On GitHub, the ratio is 58600 code results compared to 99500. Some of these results reveal that tags have become popular in private cloud management. In Kubernetes, a popular container orchestration system, labels and selectors are central to complexity reduction in large-scale deployments [8]. Yet, the scope and utilisation of tagging have not yet appeared in the peer-reviewed literature on cloud management.

A recent survey on multi-cloud resource management proposes a taxonomy for management classification [9]. While it includes a sub-taxonomy which determines how to manage multiple cloud resources, it only considers per-application or per-task group decisions but not per-tag ones. Broker-based resource management in dynamic multi-cloud environments has been proposed but only simulated with CloudSim, omitting the challenge of how to realise the management in dominant commercial clouds [10]. A dynamic programming approach for managing multi-cloud scenarios exists on an algorithmic level but again lacks the vital discussion of how it could be realised at scale in actual cloud environments [11]. Fully implemented system proposals, albeit similarly without a discussion of tagging, include LambdaLink, a multi-cloud operation management platform [12].

III. Universal Connector Concept and Design

We define a Universal Connector in general as extensible multi-cloud/cross-cloud capable middleware which can perform both management and usage tasks related to arbitrary resources. The focus of this paper is on the management tasks, in particular those implicitly driven by appropriate tags on cloud resources. Requirements-wise, the role-based management tasks encompass the management of tags themselves, the tag-driven (de)allocation of resources (often along with (un)subscription), the aggregation of information about the current multi-cloud system state including all resources subscribed to, and the auditing of past actions through a journal.

To re-use existing implementations and for better comparison, we devise an extension architecture for the connector which builds on top of existing middleware but can also be used stand-alone with restricted functionality. Fine-grained extensibility in contemporary cloud environments can be achieved by Function-as-a-Service (FaaS) which allows for small pieces of code to be glued to existing functionality, not dissimilar to plugins and glue scripts in conventional software engineering. Using FaaS, the management overhead in compute and financial terms becomes zero in periods when no management is performed according to current commercial billing models. Consequently, the anticipated universal connector relies on a set of functions to manage cloud resources. Each function performs work on its own or by wrapping an existing middleware remote method.

A. Analysis of Tagging Capabilities across Providers

One of the biggest challenges in multi-cloud tag-based management is that each provider implements tagging in a different way. When comparing the tagging implementation of different cloud providers it is immediately apparent that there are differences that could prove problematic when managing a multi-cloud system. There are minor differences in tagging policies (e.g. number and length of tags allowed) as well as major differences in how tagging is implemented and by extension how an administrator can use the tagging interface. A universal tagging system can thus aid this process by implementing a ruleset for tags that meets the constraints of all providers simultaneously, and by providing a single interface through which to manage tagging across all providers. This way an administrator can use the tagging features without studying the internal limitations and constraints of each individual provider. In our implementation, this is done by providing a common API tagging method that abstracts the provider-specific details and provides a uniform interface, and by providing the ability to apply tags automatically, thus avoiding the procedure of individually tagging resources and significantly reducing the possibility of human error. The results of our study on different tagging mechanisms for three cloud providers – Amazon Web Services (AWS), Google Cloud Platform (GCP) and Microsoft Azure – are shown in Table I.

B. Tagging Rules and Example

One of the most important requirements of a tag-driven management system is its ability to provide consistency of tagging rules, overcoming inconsistency risks associated to manually written tags. This is especially true in complex projects involving a large amount of resources and developers, particularly in a multi-cloud system. For example, a group of developers would agree on placing the tag `aim:demo` on a number of distributed resources, including shared ones which existed before, for preparing a critical live demonstration. After the demo, all resources not also needed for other purposes are to be decommissioned. A number of conditions and criteria for different resources across providers determine this functionality. The connector can then automatically generate tags for all resources that meet the criteria, allowing them to be organised and managed by tag without error and in relative safety with regards to overlapping tags.

In order for the auto-generation of tags to be applied, a set of consistency enforcement rules need to be employed by the system's administrator. The auto-generation method of the connector depends on an idempotent resource metadata retrieval function which provides sufficient data fields to construct unique identifiers in case the provider does not supply one. Moreover, by retrieving the entirety of the resource's metadata from its provider, it allows for rules to be created for any field in the metadata.

A ruleset defines per resource type and provider filter a rule composed of an arbitrary number of conditions plus a set of tag key-value pairs to be applied in case of a match. The tag will be applied to any of the resources that comply

TABLE I
COMPARISON OF TAGGING IMPLEMENTATION

Feature	AWS	Azure	GCP
Tags per resource	50	15	64
Length of key	127	512	63
Length of Value	256	256	63
Case sensitive	Yes	No	Lowercase only
API	Single tagging API for all supported resource types	API can tag any resource in a resource group	Separate tagging functionality in each API, group tagging at the project level
Tagging of multiple resources with one call	Yes, by providing a list of resources	Only by tagging a whole resource group	Only by tagging a whole project
Terminology	Tag	Tag	Label, a separate 'network tag' is used to apply firewall rules

with the conditions. The different conditions have a logical OR relationship, so in order for the rule to be satisfied, at least one condition must be met. The statements within a condition have a logical AND relationship, meaning all statements within a condition must be met by a resource simultaneously for it to be compliant with the rule. As such, the logic expression for a rule consisting of conditions $c_1 \dots c_n$ can be formalised along the following scheme:

$$(c_1 : k_1 = v_1 \wedge c_1 : k_2 = v_2) \vee c_2 : k_3 = v_3$$

If the expression is satisfied the method then calls the appropriate `tagResource` function and applies the tag to the resource. The two modes of tag application are complementary, only adding a tag if no other tag with the same key exists, or destructively, potentially overwriting values created manually or in previous auto-tagging invocations.

C. Tagging Algorithm

Listing 1 shows in pseudo-code notation how all resources within a project or account are tagged according to the defined rules. The inclusion of short-lived cloud functions happens in various stages of the algorithm.

Listing 1. Rule-based tagging algorithm

```
rules, providers ← rule file
resources_to_tag = []
$\forall provider \in providers$:
  resources_metadata ↔ call function to get resource metadata
  resources_list ↔ call function to get list of resources
      $\forall metadata \in resources\_metadata$:
          $\forall rule \in rules$:
              when rule applied to metadata:
                  resources_to_tag.add the resource from
                  resources list with the tags from rule
$\forall tag \in resources\_to\_tag$:
  → call function to tag list of resources with tag
```

IV. IMPLEMENTATION AND EVALUATION

We have implemented the proposed universal connector in the context of an industry innovation project which involved a company selling multi-cloud management solutions. Our three applied research interests have been the demonstration of feasibility of tag-driven management, the determination of runtime overheads and the decomposition of conventional (monolithic) cloud management functionality into a FaaS-based approach.

A. Software Architecture

We choose a fine-grained, cloud-native architecture for the universal connector to be able to offload parts of the management functionality into the cloud services next to the resources to be managed. The offloaded parts are realised as stateless and short-lived cloud functions. The set of functions along with wrapper methods and additional methods provided directly by the connector defines the connector's HTTP interface (API) which is discoverable through an OpenAPI specification. The connector's behaviour is defined through several human-readable and -editable YAML files.

The connector is linked to a time-series database acting as journal. All actions including regularly scheduled queries, e.g. for checking compliance against best practices rules, are recorded in the journal for auditing purposes. For a secure operation, a hash-linked list of records would have to be produced, whereas the focus of our architecture is to merely enable the functionality as part of the feasibility demonstration.

In order to satisfy the need for consistent tagging we employ the auto-tagging functionality that allows us to define a set of tagging rules in a YAML file. The rules can apply to any metadata field of each individual resource, across all providers. These rules are then used by the `autoGenerate` API method, a part of the connector that composes several cloud functions from all providers in order to retrieve all resources that satisfy the ruleset and to apply the necessary tags to them. With a well-defined set of rules, just one invocation of this function can tag every resource across all managed projects with the right tag to allow for grouping and managing by tag.

Credentials for accessing cloud providers and resources are similarly managed in YAML files and are passed to the cloud functions on demand. For a secure operation, both secure key-stores and encrypted environment variables supported by some FaaS providers would have to be used.

Multiple universal connectors can be instantiated and managed through a connector broker to map complex organisational structures to the cloud management operations. Moreover, existing cloud management platforms without tagging capabilities can be incorporated into the workflow to benefit from advanced management functionality not available from the connector itself. Fig. 1 outlines the architecture including both the functions and the resource access for two cloud providers, Amazon Web Services (AWS) and Google Cloud Platform (GCP). Optional parts (broker, multiple connectors, existing platforms) are marked in grey. The functions deployed to the various providers are added to the connector by getting registered to an event gateway that is used to populate the connector's endpoints with functions and the events to trigger them. Then the requests made to the connector reach the functions via the gateway.

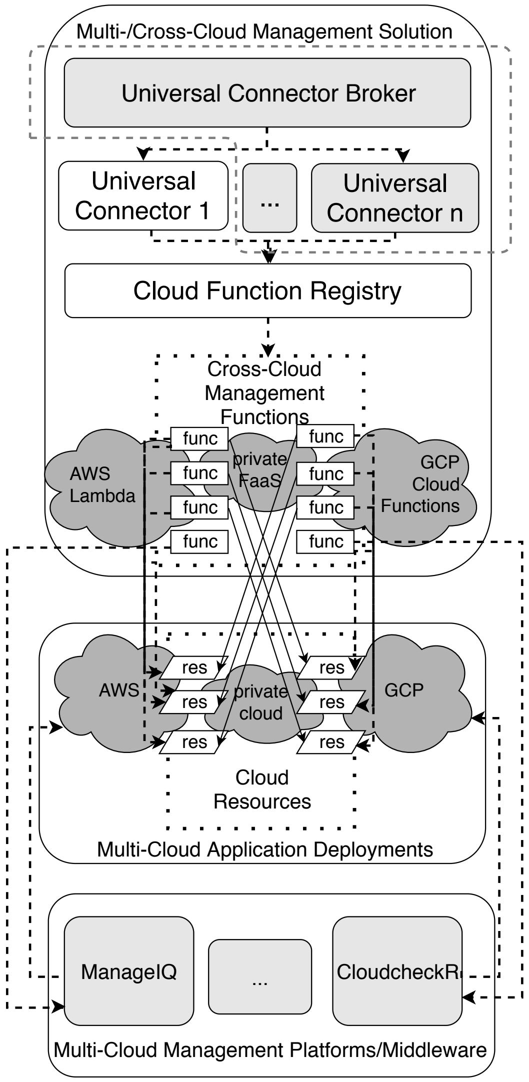
Multi-/Cross-Cloud Management Solution
Universal Connector Broker
Universal Connector 1
...
Universal Connector n
Cloud Function Registry
Cross-Cloud Management Functions
AWS Lambda
private FaaS
GCP Cloud Functions
func
res
AWS
private cloud
GCP
Cloud Resources
Multi-Cloud Application Deployments
ManageIQ
CloudcheckR
Multi-Cloud Management Platforms/Middleware

Fig. 1. Universal connector architecture outlining extensibility through FaaS and interoperability with existing management platforms

B. Implementation

The connector is implemented as a Python Flask service in conjunction with a set of functions which are implemented for both AWS Lambda and Google Cloud Functions (GCF). We note that there is an arbitrary mapping of functions to resources depending on the desired functionality and affordable implementation effort. In addition to our implementation of $AWS \rightarrow \{AWS, GCP\} \bigwedge GCP \rightarrow \{AWS, GCP\}$, the system could be extended to support $AWS \rightarrow Azure$ and other combinations.

In compliance with the scenario description, the primary features of the universal connector are:

- The automatic generation of tags, as detailed previously.
- Retrieval of resources based on their tags. This function of our API can retrieve information on resources of any type on any provider, based on the tags applied to the resource. These can include, but are not limited to, the auto-generated tags.
- Indubitably the most important feature of the multi-tag service is the capability to manage resources by tag. This function can invoke a number of resource management cloud functions from both providers. These functions in turn act on all resources with the specified tag. Managed resource actions supported currently include starting and stopping a set of VMs and deleting either VM instances or cloud functions.

Due to absence of field studies, there are no default rules; rather, specific rules need to be set in each scenario. Listing 2 exemplifies an example of a rule which can be included in the `rules.yaml` file used by the API method `/tags/autoGenerate`:

Listing 2. Auto-tagging rules

```
rule1:
 type: instance
  providers: [provider1]
   conditions:
    condition1:
     ImageId: exampleid
     InstanceType: t12.micro
    condition2:
     CpuOptions.CoreCount: 12
   tags:
    aim: demo
```

This is a rule for applying the tag key-value pair `aim:demo` to VM instances belonging to *provider1* whenever either the core count equals 12 or the image id matches for any instance of type `t12.micro`. The provider itself is described separately within the same file.

C. Initialisation and Bootstrapping

Owing to the light-weight and composite design of the connector, its startup phase is of particular interest.

On the connector side, the distributed value store *etcd*, the *Kubeless* extension to run cloud functions, the Serverless Framework's *Event Gateway* and the *InfluxDB* timeseries database are launched. Finally, the connector's Docker container is built on-demand and deployed into the Kubernetes cluster.

The multi-tag management functions (`aws_get_functions`, `aws_get_instances`, etc.) are then deployed through the Serverless Framework and its event gateway to either AWS Lambda or GCF, or even to both providers redundantly. These functions are multi-cloud capable and cross-reference all accounts subject to be managed through adequate credentials which are either passed along with the function code and configuration, or supplied on a per-request basis to lower the attack vector.

The setup procedure moreover registers all deployed functions in the connector's command registry. At any time, the registry and the rules database can be updated.

D. Scenario

We construct an application which consists of 4 VMs, two each hosted in AWS EC2 and GCP/GCE, and 4 functions, two each hosted in AWS Lambda and GCF. Our universal connector is run on a third, independent host, based on a Kubernetes environment through Minikube referred to by `uc`. The scenario encompasses two roles, application provider (administrator) and auditor.

The bootstrapping is automated with Just scripts. By invoking `just multitag-start`, the universal connector is fully initialised. The administrator first lists the resources in a multi-cloud way, finds out about missing tags, and invokes the auto-tagging based on default rules with tag `demo`. by invoking the appropriate methods (`http://uc:5000/resources/getInstances`, `http://uc:5000/resources/getFunctions`, `http://uc:5000/tags/autoGenerate`).

Then, after some time, these resources are no longer needed and the administrator conveniently stops them based on the `demo` tag. The API is queried for this purpose using either the `getByTag` or `manageByTag` methods. On the command line, this may look like the following invocation: `curl -X POST -d '"action": "stop", "key": "aim", "value": "demo"' http://uc:5000/resources/manageByTag`. Afterwards, the administrator shows the activity journal to the company's compliance officer to demonstrate that all resources have indeed been disposed by invoking `http://uc:5000/journal`. Finally, the administrator finds out that one resource is going to be re-used in production, and manually tags it with `production`, i.e. `curl -X POST -d '"ids": ["id1"], "tags":["key": "aim", "value": "production"]' http://uc:5000/tagResource`.

Listing 3 outlines the structure of the audit log, omitting some precision information for brevity. While in our prototype it is not tamper-proof, it could be implemented using a linked hash list to accommodate legal compliance requirements.

Listing 3. Audit log excerpt

```
time          endpoint                result          user
_____         _______                 _______         _____

1550516236    /tags/AutoGenerate      ['success']     auto
1550516261    /tagResource            ['fail']        admin
```

E. Evaluation

The diagram in Fig. 2 shows the time flow for the tag auto-generation process. Once the data is fetched from the YAML file which is structured as is described above, the related data for each provider is sent as parameter to an instance of the function which queries metadata and lists instances. The results are compared with conditions defined in the rules. The list of resources which matches the rules is sent jointly with related tag information as request body to the corresponding `tagResource` function call.

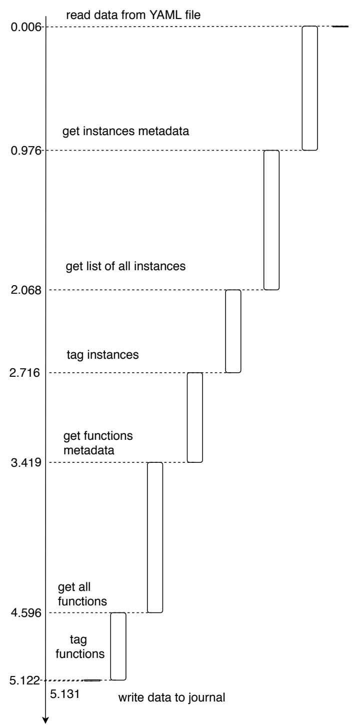

Fig. 2. Time diagram for tag auto-generation

As can be inferred from the diagram, the initial and final actions of reading configuration data and writing audit log data do not cause any delays whereas all intermediate actions contribute to a multi-second process. Possible optimisations include the introduction of multi-threading and more aggressive caching.

In order to measure the overhead of the FaaS-based designs, the resource management functions are compared to synchronous resource modification through the CloudcheckR management platform which offers functionally equivalent commands. Figures 3 and 4 demonstrate the temporal behaviour of interacting with CloudcheckR with and without Lambda functions and therefore the overhead using cloud functions to perform API calls. From the graphs, it becomes apparent that a small overhead can be introduced. In the first case in Fig. 3, the execution time is small in absolute numbers but expressed as percentage the difference looks more significant with 22% slowdown due to the involvement of functions. However as indicated by the distribution and mode in Fig. 4, the relative overhead is only 2.5% and functions may even lead to a slight speedup.

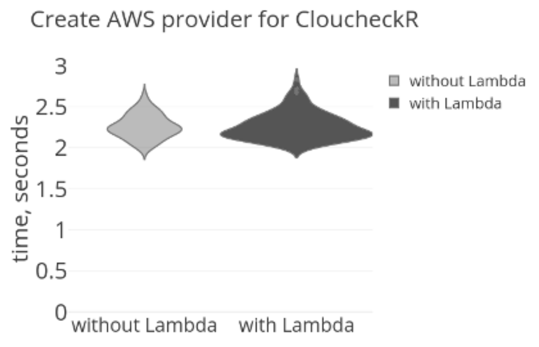

Fig. 3. Performance overhead for resource creation

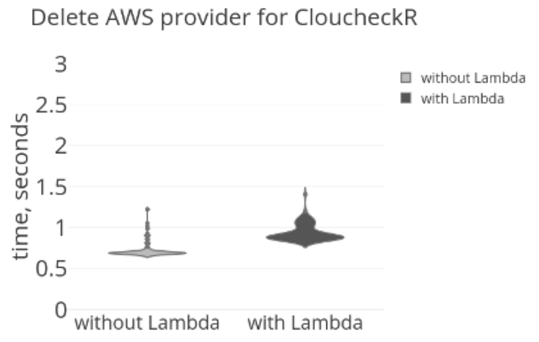

Fig. 4. Performance overhead for resource deletion

V. Summary, Resources and Future Directions

We have introduced a middleware design and system for cross-cloud management which can execute entirely within the target cloud platform through short-lived cloud functions. The main capability of the system is to manage cloud resources by tags. While the software is currently not publicly available for commercial reasons of the implementation partner company with whom we conducted the work, we expect that the conceptual descriptions given in this paper inspire future system designs to become similarly light-weight and extensible.

An explanatory screencast video is provided online in the Service Prototyping Research Videos collection[1] while a second video explains the extension of cloud management platforms with cloud functions[2].

A consequent enhancement of the presented approach would be to exploit code generation from the OpenAPI specifications to further reduce the implementation effort within the cloud functions, and to implement code for business alignment.

Another enhancement concerns the economic exploitation of operating cloud brokers and connectors. The basic unit of billable activity for the FaaS offering of both providers is invocations per month. AWS provides the first 1 million requests per month for free and charges 0.2$ per million thereafter. GCP similarly provides 2 million invocation per month for free and charges 0.4$ per million thereafter. These prices are further modified by compute time and memory used per request. In contrast, CloudcheckR's pricing is 2,5% of the customer's cloud bill for their management functionality which could be achieved with an appropriate short-lived cloud function retrieving the per-tag monthly cost. According to our initial findings, this would work well with AWS but is more challenging in GCF due to the non-global tag model. A direct comparison of these two models however is not possible without an in depth analysis of a particular use case for each of them, as they demonstrate very different strategies.

[1]Video on tag-driven cloud management: https://www.youtube.com/watch?v=YAwA92LPFEE

[2]Video on management platforms: https://www.youtube.com/watch?v=VaFTDbz8mxw

References

[1] Paul Alpar and Ariana Polyviou. Management of Multi-cloud Computing. In *Global Sourcing of Digital Services: Micro and Macro Perspectives - 11th Global Sourcing Workshop 2017, La Thuile, Italy, February 22-25, 2017, Revised Selected Papers*, pages 124–137, 2017.

[2] Mathias Slawik, Christophe Blanchet, Yuri Demchenko, Fatih Turkmen, Alexy Ilyushkin, Cees de Laat, and Charles Loomis. CYCLONE: The Multi-cloud Middleware Stack for Application Deployment and Management. In *IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2017, Hong Kong, December 11-14, 2017*, pages 347–352, 2017.

[3] Sukhpal Singh Gill and Rajkumar Buyya. SECURE: Self-Protection Approach in Cloud Resource Management. *IEEE Cloud Computing*, 5(1):60–72, 2018.

[4] Merlijn Sebrechts, Gregory van Seghbroeck, Tim Wauters, Bruno Volckaert, and Filip De Turck. Orchestrator conversation: Distributed management of cloud applications. *Int. Journal of Network Management*, 28(6), 2018.

[5] Beniamino Di Martino, Giuseppina Cretella, and Antonio Esposito. Cross-Platform Cloud APIs. In *Cloud Portability and Interoperability*. Springer, 2015.

[6] Daniel Baur and Jörg Domaschka. Experiences from building a cross-cloud orchestration tool. In *Proceedings of the 3rd Workshop on CrossCloud Infrastructures & Platforms, CrossCloud@EuroSys 2016, London, United Kingdom, April 18-21, 2016*, pages 4:1–4:6, 2016.

[7] Victor Ion Munteanu, Calin Sandru, and Dana Petcu. Multi-cloud resource management: cloud service interfacing. *J. Cloud Computing*, 3:3, 2014.

[8] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes. *Commun. ACM*, 59(5):50–57, 2016.

[9] Juliana Oliveira de Carvalho, Fernando Trinta, Dario Vieira, and Omar Andrés Carmona Cortes. Evolutionary solutions for resources management in multiple clouds: State-of-the-art and future directions. *Future Generation Comp. Syst.*, 88:284–296, 2018.

[10] Naidila Sadashiv and Dilip Kumar S. M. Broker-based resource management in dynamic multi-cloud environment. *IJHPCN*, 12(1):94–109, 2018.

[11] Antonio Pietrabissa, Francesco Delli Priscoli, Alessandro Di Giorgio, Alessandro Giuseppi, Martina Panfili, and Vincenzo Suraci. An approximate dynamic programming approach to resource management in multi-cloud scenarios. *Int. J. Control*, 90(3):492–503, 2017.

[12] Kate Keahey, Pierre Riteau, and Nicholas P. Timkovich. LambdaLink: an Operation Management Platform for Multi-Cloud Environments. In *Proceedings of the 10th International Conference on Utility and Cloud Computing, UCC 2017, Austin, TX, USA, December 5-8, 2017*, pages 39–46, 2017.