Are REST APIs for Cloud Computing Well-Designed? An Exploratory Study

Fabio Petrillo[1,3,4(✉)], Philippe Merle[2], Naouel Moha[1], and Yann-Gaël Guéhéneuc[3]

[1] Département d'informatique, Université du Québec à Montréal, Montreal, Canada
fabio@petrillo.com, moha.naouel@uqam.ca
[2] Equipe Spirals, Inria Lille - Nord Europe, Villeneuve d'Ascq, France
philippe.merle@inria.fr
[3] DGIGL, École Polytechnique, Montréal, Montreal, Canada
yann-gael.gueheneuc@polymtl.ca
[4] PPGC, Federal University of Rio Grande do Sul, Porto Alegre, Brazil

Abstract. Cloud computing is currently the most popular model to offer and access computational resources and services. Many cloud providers use the REST architectural style (Representational State Transfer) for offering such computational resources. However, these cloud providers face challenges when designing and exposing REST APIs that are easy to handle by end-users and/or developers. Yet, they benefit from best practices to help them design understandable and reusable REST APIs.

However, these best practices are scattered in the literature and they have not be studied systematically on real-world APIs. Consequently, we propose two contributions. In our first contribution, we survey the literature and compile a catalog of 73 best practices in the design of REST APIs making APIs more understandable and reusable. In our second contribution, we perform a study of three different and well-known REST APIs from three cloud providers to investigate how their APIs are offered and accessed. These cloud providers are Google Cloud Platform, OpenStack, and Open Cloud Computing Interface (OCCI). In particular, we evaluate the coverage of the features provided by the REST APIs of these cloud providers and their conformance with the best practices for REST APIs design.

Our results show that Google Cloud follows 66 % (48/73), OpenStack follows 62 % (45/73), and OCCI 1.2 follows 56 % (41/73) of the best practices. Second, although these numbers are not necessarily high, partly because of the strict and precise specification of best practices, we showed that cloud APIs reach an acceptable level of maturity.

1 Introduction

Cloud computing has transformed the Information Technology (IT) industry [1] by hosting applications and providing resources (e.g., CPU and storage) as services on-demand over the Internet [15]. Cloud providers, such as Google Cloud

Q.Z. Sheng et al. (Eds.): ICSOC 2016, LNCS 9936, pp. 157–170, 2016.
DOI: 10.1007/978-3-319-46295-0_10

Platform and OpenStack, usually offer these services in the form of REST (REpresentational State Transfer) [4] APIs, the *de facto* standard adopted by many software organisations for publishing their services.

However, although cloud computing offers huge opportunities for the IT industry and has gained maturity, there are still many issues that must be addressed [15]. In particular, we observe that cloud providers, such as Google Cloud Platform, present their own proprietary APIs. Other cloud APIs, although proprietary such as OpenStack, provide open implementations of cloud services. Conversely, open and standard cloud APIs have been proposed, such as the Open Cloud Computing Interface (OCCI) [7], which is a neutral-vendor cloud standard.

Consequently, there exists a wide variety of cloud APIs that might be difficult to understand and use by developers, especially within a complex and technical context as cloud computing. Moreover, well-designed REST APIs may attract client developers to use them more than poorly designed ones, particularly in the current open market, where Web services are competing against one another [6]. Indeed, client developers must understand the providers' APIs while designing and developing their systems that use these APIs. Therefore, in the design and development of REST APIs, their understandability and reusability are two major quality characteristics, which are reachable when best practices for REST APIs design [6] are followed.

Several practices were proposed or identified in the literature [2,6,8,9,12] as *CRUD function names should not be used in URIs* or *Lowercase letters should be preferred in URI paths*. In particular, a valuable contribution is the one of Massé [6], who compiles several design practices about REST APIs. Yet, despite proposing a large list of 65 practices, Massé [6] did not propose a complete list.

Consequently, we propose two contributions. For our first contribution, we review the literature extensively and compile a catalog of 73 best practices in the design of REST APIs making APIs more understandable and reusable.

For our second contribution, we evaluate and compare the design of the cloud computing REST APIs using best practices of this catalog. Compared to previous works, including some of ours [2,8,9], we study the conformance with best practices of REST APIs from the perspective of cloud providers.

After identifying and analysing 73 best practices, our results show that Google Cloud Platform follows 66 % (48/73), OpenStack follows 62 % (45/73), and OCCI 1.2 follows 56 % (41/73) of the best practices. Second, although these numbers are not necessarily high, partly because of the strict and precise specification of best practices, we showed that cloud APIs reach an acceptable level of maturity.

The reminder of the paper is organised as follows. Section 2 presents a survey about best practices on REST APIs to support our evaluation. Section 3 describes the study performed on the three cloud computing REST APIs. Section 4 presents and discusses our results and the threats to their validity. Section 5 presents some related work. Finally, Sect. 6 concludes the paper with future work.

2 Best Practices on REST API Design

REST APIs are hard to design [6] because they are not often based on precise and documented specifications but only on an architectural style [4]. Thus, we now present a catalog of REST API best practices, pertaining to understandability and reusability, extracted from the literature and organised to support our analysis on cloud REST APIs.

To build our catalog, we surveyed several studies of REST APIs elaborated by Massé [6], Rodrigues et al. [2], Palma et al. [8,9], Vinoski [13], Stowe [12], and Richardson and Ruby [11]. In particular, Massé [6] provides a concise catalog of practices organised by categories. We analysed all the cited studies to identify good practices and organise them by categories inspired from Massé's work.

Table 1. Numbers of practices by category

Category	Number of practices
URI	20
Request methods	8
Error handling	16
HTTP headers	10
Others	19
Total	73

Our literature review produced a catalog of 73 best practices to design understandable and reusable REST APIs, grouped into five categories. Table 1 lists the categories and the numbers of practices per category. Tables 2, 3, 4, 5 and 6 describe the identified practices in each category with a short description, relevant references, and the results of our analysis on the three Cloud REST APIs. The analysis of each API is further discussed in Sect. 4.

The first category, URI practices, describes how URIs are exposed by services (Table 2). The second category, Request Methods, describes how HTTP methods must be used by REST APIs (Table 3). Error Handling practices specify how HTTP messages must be used as a response of a HTTP request method (Table 4). HTTP Header practices describe how must be used HTTP headers to complete requests with metadata or complementary data (Table 5). Finally, Others is the category for grouping different and various practices as Media Types, Message Body Format, Versioning, Security, Response Representation Composition, Documentation and Hypermedia Representation (Table 6).

Table 2. URI design best practices

	Practices	References	Google	OpenStack	OCCI
1	Forward slash separator (/) must be used to indicate a hierarchical relationship	[6,9,11]	✓	✓	✓
2	A trailing forward slash (/) should not be included in URIs	[2,6,9]	✓	✓	✓
3	Hyphens (-) should be used to improve the readability of URIs	[6,9]	-	-	-
4	Underscores (_) should not be used in URIs	[2,6,9]	✓	✓	-
5	Lowercase letters should be preferred in URI paths	[2,6,9]	✓	✓	-
6	File extensions should not be included in URIs	[6]	✓	✓	-
7	Consistent subdomain names should be used for your APIs	[6,9,11]	✓	✓	✓
8	A singular noun should be used for document names	[6,9]	✓	✓	-
9	A plural noun should be used for collection names	[6,9]	✓	✓	-
10	A plural noun should be used for store names	[6,9]	✓	✓	-
11	A verb or verb phrase should be used for controller names	[6,9]	✓	✓	-
12	CRUD function names should not be used in URIs	[2,6,9]	✓	✓	-
13	Use path variables to separate elements of a hierarchy, or a path through a directed graph	[11]	✓	✓	-
14	Avoiding version number in the path	[2]	-	-	-
15	API as part of the subdomain	[2]	✓	-	-
16	The query component of a URI may be used to filter collections or stores	[6]	✓	✓	✓
17	The query component of a URI should be used to paginate collection or store results	[6]	✓	✓	✓
18	Keeping as much information as possible in the URI, and as little as possible in request metadata	[11]	✓	✓	-
19	Avoiding version number in the query params	[2]	✓	✓	✓
20	Avoiding CRUD actions in query params	[2]	✓	✓	✓

Table 3. Request methods best practices

	Practices	References	Google	OpenStack	OCCI
1	GET and POST must not be used to tunnel other request methods	[2,6,8,12]	✓	✓	✓
2	GET must be used to retrieve a representation of a resource	[6,11–13]	✓	✓	✓
3	HEAD should be used to retrieve response headers	[6,11–13]	✓	✓	-
4	PUT must be used to both insert and update a stored resource	[6,11–13]	-	-	✓
5	PUT must be used to update mutable resources	[6,11–13]	✓	✓	✓
6	POST must be used to create a new resource in a collection	[6,11–13]	✓	✓	✓
7	POST must be used to execute controllers	[6,12]	✓	✓	✓
8	DELETE must be used to remove a resource from its parent	[6,11–13]	✓	✓	✓

Table 4. Error handling best practices

	Practices	References	Google	OpenStack	OCCI
1	200 ("OK") should be used to indicate nonspecific success	[6, 12]	✓	✓	✓
2	200 ("OK") must not be used to communicate errors in the response body	[6, 12]	✓	✓	✓
3	201 ("Created") must be used to indicate successful resource creation	[6, 12]	-	-	✓
4	202 ("Accepted") must be used to indicate successful start of an asynchronous action	[6, 12]	-	-	-
5	204 ("No Content") should be used when the response body is intentionally empty	[6, 12]	-	-	✓
6	302 ("Found") should not be used	[6, 12]	-	-	✓
7	304 ("Not Modified") should be used to preserve bandwidth	[6, 12, 13]	-	-	-
8	400 ("Bad Request") may be used to indicate nonspecific failure	[6, 12]	✓	✓	✓
9	401 ("Unauthorized") must be used when there is a problem with the client's credentials	[6, 12]	-	✓	✓
10	403 ("Forbidden") should be used to forbid access regardless of authorization state	[6, 12]	-	-	✓
11	404 ("Not Found") must be used when a client's URI cannot be mapped to a resource	[6, 12]	✓	-	✓
12	405 ("Method Not Allowed") must be used when the HTTP method is not supported	[6, 12]	-	-	✓
13	406 ("Not Acceptable") must be used when the requested media type cannot be served	[6, 12]	-	-	✓
14	409 ("Conflict") should be used to indicate a violation of resource state	[6, 12]	-	✓	✓
15	500 ("Internal Server Error") should be used to indicate API malfunction	[6, 12]	✓	-	✓
16	Use JSON as error message response	[6, 12]	✓	✓	-

Table 5. HTTP header best practices

	Practices	Ref.	Google	OS	OCCI
1	Content-type must be used	[6, 12, 13]	✓	✓	-
2	Content-length should be used	[6]	✓	✓	-
3	Last-modified should be used in responses	[6]	-	✓	-
4	ETag should be used in responses	[6, 13]	✓	✓	-
5	Stores must support conditional PUT requests	[6]	-	-	-
6	Location must be used to specify the URI of a newly created resource	[6]	-	-	-
7	Cache-control, expires, and date response headers should be used to encourage caching	[6]	✓	✓	-
8	Cache-control, expires, and pragma response headers may be used to discourage caching	[6]	-	-	-
9	Caching should be encouraged	[6]	✓	✓	-
10	Custom HTTP headers must not be used to change the behavior of HTTP methods	[6]	✓	✓	✓

Table 6. Other best practices

	Practices	Ref.	Google	OS	OCCI
1	Application-specific media types should be used	[6]	-	-	✓
2	Media type negotiation should be supported when multiple representations are available	[6]	-	-	✓
3	Media type selection using a query parameter may be supported	[6]	-	-	-
4	JSON should be supported for resource representation	[6]	✓	✓	✓
5	XML and other formats may optionally be used for resource representation	[6]	-	-	✓
6	Additional envelopes must not be created	[6]	✓	✓	✓
7	New URIs should be used to introduce new concepts	[6]	✓	✓	✓
8	Schemas should be used to manage representational form versions	[6]	✓	-	✓
9	Entity tags should be used to manage representational state versions	[6]	-	-	✓
10	OAuth may be used to protect resources	[6,12]	✓	✓	-
11	The query component of a URI should be used to support partial responses	[6]	-	-	✓
12	The query component of a URI should be used to embed linked resources	[6,11]	-	-	-
13	Consistent subdomain names should be used for your client developer portal	[6]	-	-	-
14	Accompanying human-readable documentation	[6,12]	✓	✓	✓
15	Interactive experiences to try/test API calls	[12]	✓	-	-
16	Code examples for multiple languages	[12]	✓	-	-
17	A consistent form should be used to represent links	[6,8]	✓	✓	✓
18	A consistent form should be used to advertise links	[6,8]	✓	✓	✓
19	A self link should be included in response message body representations	[6,8]	✓	✓	✓

3 Study Design

This section presents the design of our study, which aims to address the following four research questions:

RQ1 What are the main services provided by cloud REST APIs?
RQ2 How many best practices are followed by cloud REST APIs?
RQ3 What best practices are adopted by all APIs?
RQ4 What best practices are adopted by none of the APIs?

3.1 Objects

The objects of our study are three different cloud REST APIs including the proprietary cloud API of Google Cloud Platform, the open source API of OpenStack, and the standard OCCI. We specifically target these APIs because they represent the range of the different types of cloud APIs available: commercial offer, open source implementation, and open standard.

Here is a short description of each of the three studied APIs:

Google Cloud Platform is a proprietary cloud platform that consists of a set of physical assets (e.g., computers and hard disk drives) and virtual resources (e.g., virtual machines, a.k.a. VMs) hosted in Google's data centers around the globe. Google Cloud documentation is available at https://cloud.google.com/docs.

OpenStack is an open source cloud platform that controls large pools of compute, storage, and networking resources throughout a datacenter. OpenStack documentation is available at http://docs.openstack.org.

Open Cloud Computing Interface (OCCI) is a cloud computing standard that comprises a set of open community-lead specifications delivered through the Open Grid Forum. OCCI is a protocol and API for all kinds of management tasks. OCCI 1.2 documentation is available at http://occi-wg.org/about/specification.

3.2 Procedure

We investigated and analysed manually in details the documentation of each of the three APIs studied. More precisely, we identified the services provided by each API and extracted the list of URIs. Then, we compared them with each best practice as defined in our compiled catalog of best practices. The analysis of OCCI has been cross-validated by two contributors of the standard. We performed an analysis of the three APIs and reported the results of this analysis in several tables given in the next section dedicated to the results.

4 Results

We now report the results of our analysis to answer our four research questions.

4.1 RQ1 What Are the Main Services Provided by Cloud REST APIs?

We performed a manual analysis on each REST API documentation to identify the services provided. This first analysis allows us to identify 11 services as listed in Table 7. Our results show that Google Cloud Platform and OpenStack provide all identified services (11/11) while OCCI describes only four services in its specifications (4/11). We conclude that **Google and OpenStack API have a good support for several services while OCCI has yet some lacks**.

Indeed, current OCCI 1.2 official specifications only cover 4 of the 11 services listed in Table 7: VM Managing, Storage, Networking, and Tagging. However, OCCI-based services were proposed for Container Managing [10] and Monitoring [3]. Image Managing, Scaling, Access Control, Data Processing and Machine Learning would be addressed in future as there are key services for building open standard cloud platforms.

Table 7. Service analysis comparing Google Cloud, OpenStack, and OCCI

Main services	Google	OpenStack	OCCI
VM managing	✓	✓	✓
Container managing	✓	✓	✗
Image managing	✓	✓	✗
Storage	✓	✓	✓
Networking	✓	✓	✓
Scaling	✓	✓	✗
Access control	✓	✓	✗
Monitoring	✓	✓	✗
Tagging	✓	✓	✓
Data processing	✓	✓	✗
Machine learning	✓	✓	✗

4.2 RQ2 How Many Best Practices Are Followed by Cloud REST APIs?

For each practice, we analysed the documentation of the corresponding API to assess whether this provider follows or not the practice. Tables 2-6 present the detailed results of this assessment for each API and the 73 best practices by category. Table 8 presents a summary of this assessment by category of practices and shows that **on average 61 % (44/73) of the practices are followed by the three APIs.** Google Cloud Platform follows 66 % (48/73), OpenStack follows 62 % (45/73), and OCCI follows 56 % (41/73) of the best practices.

Moreover, OCCI follows only 35 % (7/20) of URI practices, while Google Cloud Platform and OpenStack follow URI practices in 90 % (18/20) and 85 % (17/20), respectively. OCCI 1.2 fails to support URI design best practices, but future OCCI releases could improve this as the OCCI REST API is automatically synthetised from a metamodel instead of designed by hand as in Google Cloud Platform and OpenStack.

All APIs strongly apply Request Methods best practices listed in Table 3 with 87 % on average. Each API requires just one improvement on PUT method for Google Cloud Platform and OpenStack, and on HEAD method for OCCI. For the latter, OCCI implementations such as erocci[1] and rOCCI[2] already support HEAD method best practice, so a consensus in the OCCI community should be easily attainable to include this best practice into OCCI specifications.

Finally, Error Handling, HTTP Headers, and the other categories are the less followed with only 52 %, 46 %, and 56 %.

[1] http://erocci.ow2.org.
[2] http://gwdg.github.io/rOCCI/.

Error handling in Google Cloud Platform and OpenStack (6/16 or 37,5 % in Table 8) requires to be strongly improved with a better documentation in the error messages.

Regarding HTTP header best practices, all the three cloud REST APIs can be improved but especially OCCI 1.2, which only supports 1 practice, i.e., 10 % in Table 5. Here the OCCI 1.2 HTTP Protocol specification must be extended to support more HTTP header best practices. For instance, this specification must explicitly state that *Content-Type must be used*, *Content-Length should be used*, *Last-Modified should be used in responses*, *ETag should be used in responses*, *Cache-Control, Expires, and Date response headers should be used to encourage caching*, *Cache-Control, Expires, and Pragma response headers may be used to discourage caching*, and *Caching should be encouraged*. Let's note that OCCI implementations such as erocci and rOCCI already implement most of these best practices, so the consensus into the OCCI community should be reasonably attainable. In contrast, both *stores must support conditional PUT requests* and *location must be used to specify the URI of a newly created resource* are best practices that no API seems to want to support.

Table 8. Followed practices by category and API

Category	Total	Google	OpenStack	OCCI	Average	Avg/Total
URI	20	18	17	7	14	70 %
Request methods	8	7	7	7	7	87 %
Error handling	16	6	6	13	8	52 %
HTTP headers	10	6	7	1	4	46 %
Others	19	11	8	13	10	56 %
Total	73	48	45	41	44	61 %

4.3 RQ3 What Best Practices Are Adopted by All APIs?

In Table 9, we identified from our results the set of practices that **all APIs follow**, forming a "consensus". We found that only 32 % (24/73) of practices were followed by all APIs. This means that the cloud API providers are not yet in agreement on the main good practices to prioritise and might be guided by technical decisions. However, it should be pointed out that the practices are strict and detailed. This explains why this number is very low. Moreover, we could identify that the majority of practices are adopted at least by one API. Overall, the APIs, even if do not follow strictly all best practices, implement relatively well all practices and are thus well-designed.

4.4 RQ4 What Best Practices Are Adopted by None of the APIs?

An opposite analysis allows us to identify the set of practices that **none API follows**, forming a negative "consensus". We found that only ten best practices

Table 9. Practices followed by All APIs

Practices	Categories
Forward slash separator (/) must be used to indicate a hierarchical relationship	URI format
A trailing forward slash (/) should not be included in URIs	URI format
Consistent subdomain names should be used for your APIs	URI authority
The query component of a URI may be used to filter collections or stores	URI query
The query component of a URI should be used to paginate collection or store results	URI query
Avoiding version number in the query params	URI query
Avoiding CRUD actions in query params	URI query
GET and POST must not be used to tunnel other request methods	Request methods
GET must be used to retrieve a representation of a resource	Request methods
PUT must be used to update mutable resources	Request methods
POST must be used to create a new resource in a collection	Request methods
POST must be used to execute controllers	Request methods
DELETE must be used to remove a resource from its parent	Request methods
200 ("OK") should be used to indicate nonspecific success	Error handling
200 ("OK") must not be used to communicate errors in the response body	Error handling
400 ("Bad Request") may be used to indicate nonspecific failure	Error handling
Custom HTTP headers must not be used to change the behavior of HTTP methods	HTTP headers
JSON should be supported for resource representation	Message body
Additional envelopes must not be created	Message body
New URIs should be used to introduce new concepts	URI
Accompanying human-readable documentation	Documentation
A consistent form should be used to represent links	Hypermedia
A consistent form should be used to advertise links	Hypermedia
A self link should be included in response message body representations	Hypermedia

(14 %) are applied by none of the three APIs analysed. Table 10 lists the practices followed by no API. This list could be analysed to understand why these practices are not followed by more APIs. For example, as any cloud API provider performs long running actions, e.g. starting a virtual machine takes some minutes, then all cloud APIs must return *202 ("Accepted")* HTTP status to indicate successful start of an asynchronous action. Another example is that *304 ("Not Modified")* should be used by all cloud APIs to preserve network bandwidth.

4.5 Threats to Validity

As with any such empirical study, threats exist that reduce its validity, which we attempted to mitigate or had to accept. We now discuss these threats and the measures that we took with respect to them.

Threats to the construct validity of our study concern the relationship between theory and observations. We assumed (1) that good practices can be codified and

Table 10. Practices followed by No API

Practices	Categories
Hyphens (-) should be used to improve the readability of URIs	URI format
Avoiding version number in the path	URI path
202 ("Accepted") must be used to indicate successful start of an asynchronous action	Error handling
304 ("Not Modified") should be used to preserve bandwidth	Error handling
Stores must support conditional PUT requests	HTTP request
Location must be used to specify the URI of a newly created resource	HTTP request
Cache-Control, Expires, and Pragma response headers may be used to discourage caching	HTTP request
Media type selection using a query parameter may be supported	URI format
The query component of a URI should be used to embed linked resources	URI format
Consistent subdomain names should be used for your client developer portal	URI authority

shared among developers and (2) that these good practices improve the quality of the REST APIs of the cloud providers that follow them [15]. Although these assumptions are legitimate and have been withheld by many researchers and works before for example that of Zhang and Budgen [14], future work should study whether these good practices apply universally to all cloud services.

Threats to internal validity concern confounding factors that can affect our dependent variables. Although we did not carry any statistical analysis on the characteristics of the studied REST APIs, we assumed that the good practices were representative characteristics of the REST APIs. However, there may be other characteristics that describe more accurately these REST APIs, in particular their understandability and reusability. Future work include analysing and contrasting more APIs with more practices to uncover possible other characteristics. We also related the APIs and the practices manually thanks to the information provided in the literature and by the APIs documentations. Yet, other researchers should perform similar analysis to confirm/infirm ours.

Threats to conclusion validity deal with the relation between the treatment and the outcome. Again, we did not carry any statistical analysis between the REST APIs and the identified good practices and the characteristics of understandability and reusability. Yet, we argued that comparing these REST APIs according to their use of the practices is sensible because they belong to the same design space. We also accepted that external characteristics could influence their design and, hence, their quality. We accepted this threat and future work could uncover more novel characteristics and measures of REST APIs.

Threats to external validity concern the generalisability of our results. Although we presented, to the best of our knowledge, the largest study on the design of REST APIs based on an catalog of good practices from the literature, we cannot generalise our results to all REST APIs. Future work is necessary to

analyse more REST APIs, from other cloud providers, to confirm and–or infirm our observations on their design quality characteristics.

5 Related Work

To the best of our knowledge, few work studied the evaluation of REST APIs.

Rodríguez et al. [2] evaluated the conformance of good and bad practices in REST APIs from the perspective of mobile applications. They analysed large data logs of HTTP calls collected from the Internet traffic of mobile applications, identified usage patterns from logs, and compared these patterns with design best practices. Zhou et al. [16] showed how to fix design problems related to the use of REST services in existing Northbound networking APIs in a Software Defined Network and how to design a REST Northbound API in the context of OpenStack. Both of these two previous work made contributions to the design evaluation of REST APIs for two specific domains, mobile and networking, while we consider the domain of cloud services.

Maleshkova et al. [5] analysed a set of 220 publicly-available Web APIs, including RPC, REST, and hybrid (a mix of RPC and REST) styles. They investigated six characteristics of the APIs: general information, types of Web APIs, input parameters, output formats, invocation details, and complementary documentation. This work provides a view on how Web APIs are developed and exposed. In particular, it shows that Web APIs are not necessarily REST and that they suffer from under-specification because important information such as data-type and HTTP methods are missing. This work supports our paper on the need to study the design of REST APIs.

In our previous works [8,9], we evaluated the design of several REST APIs based on REST patterns and antipatterns, which correspond to good and bad practices in the design of REST services. However, the APIs evaluated were selected from different and general domains. They included Facebook, Twitter, Dropbox, and Bestbuy. So, it was not possible to compare and discuss the results among the APIs. Moreover, the list of patterns and antipatterns was really limited compared to the catalog of best practices presented in this paper.

6 Conclusion and Future Work

In this paper, we claimed that well designing REST APIs is difficult for cloud providers although there exist best practices pertaining to understandability and reusability. We supported our claim by performing, to the best of our knowledge, the first study evaluating and comparing the designs of the REST APIS of several cloud providers. We included in our study the REST APIs provided by Google Cloud Platform, OpenStack, and OCCI. We presented thus two contributions, a catalog of best practices from the literature and an evaluation of the use of these practices in three sets of APIs.

For our first contribution, we reviewed the literature extensively and compiled a catalog of 73 best practices in the design of REST APIs making APIs more

understandable and reusable. We believe that our catalog is exhaustive at the time of writing. We hope that this catalog can be used by other researchers to study some quality characteristics of REST APIs as well as practitioners when designing, implementing, and assessing their own REST APIs.

For our second contribution, we evaluated and compared the design of the REST APIs using best practices from the literature and showed that Google Cloud follows 66 % (48/73), OpenStack follows 62 % (45/73), and OCCI 1.2 follows 56 % (41/73) of the best practices.

Thus, we showed that best practices can help evaluate REST APIs and design better REST APIs in terms of understandability and reusability. Moreover, in opposition of a recent assessment [2], we also showed that cloud APIs reach an acceptable level of maturity when considering good practices pertaining to understandability and reusability.

Future work includes studying whether these good practices apply universally to all cloud APIs. In particular, we planned to analyse and contrast more APIs, especially other commercial offers like Amazon Web Services and other open source cloud stacks like Apache's CloudStack, with more practices to uncover possible other characteristics. Finally according to the results of this study, we will contribute to the improvement of OCCI specifications in order to make them more visible as OCCI is the only open standard addressing the management of any cloud computing resource.

Acknowledgment. Thank to Boris Parak from CESNET and Jean Parpaillon from Inria for their helps in evaluating the support by OCCI of our catalog of best practices. This work is partially supported by the OCCIware research and development project (http://www.occiware.org) funded by French Programme d'Investissements d'Avenir (PIA).

References

1. Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A.: A view of cloud computing. Commun. ACM **53**(4), 50 (2010). http://portal.acm.org/citation.cfm?doid=1721654.1721672
2. Rodríguez, C., Baez, M., Daniel, F., Casati, F., Trabucco, J.C., Canali, L., Percannella, G.: REST APIs: a large-scale analysis of compliance with principles and best practices. In: Bozzon, A., Cudré-Mauroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 21–39. Springer, Heidelberg (2016). doi:10.1007/978-3-319-38791-8_2
3. Ciuffoletti, A.: Application level interface for a cloud monitoring service. Comput. Stand. Interfaces **46**, 15–22 (2016). http://www.sciencedirect.com/science/article/pii/S0920548916000027
4. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000). http://www.ics.uci.edu/fielding/pubs/dissertation/top.htm

5. Maleshkova, M., Pedrinaci, C., Domingue, J.: Investigating web APIs on the world wide web. In: 2010 Eighth IEEE European Conference on Web Services, pp. 107–114. IEEE (2010). http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5693251
6. Masse, M.: REST API Design Rulebook, vol. 53. O'Reilly Media, Sebastopol (2011)
7. Merle, P., Barais, O., Parpaillon, J., Plouzeau, N., Tata, S.: A precise metamodel for open cloud computing interface. In: 2015 IEEE 8th International Conference on Cloud Computing, pp. 852–859, June 2015
8. Palma, F., Dubois, J., Moha, N.: Service-Oriented Computing. Lecture Notes in Computer Science, vol. 8831. Springer, Berlin, Heidelberg (2014). http://link.springer.com/10.1007/978-3-662-45391-9
9. Palma, F., Gonzalez-Huerta, J., Moha, N., Guéhéneuc, Y.G., Tremblay, G.: Are RESTful APIs well-designed? Detection of their linguistic (anti)patterns. In: Toumani, F., et al. (eds.) ICSOC 2014. LNCS, vol. 8954. Springer, Heidelberg (2015). http://link.springer.com/10.1007/978-3-319-22885-3, http://link.springer.com/10.1007/978-3-662-48616-0_11
10. Paraiso, F., Challita, S., Al-Dhuraibi, Y., Merle, P.: Model-driven management of docker containers. In: Proceedings of 9th IEEE International Conference on Cloud Computing (CLOUD) (2016, to appear)
11. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media Inc., Sebastopol (2007)
12. Stowe, M.: Undisturbed REST: A Guide to Designing the Perfect API. MuleSoft, San Francisco (2015)
13. Vinoski, S.: RESTful web services development checklist. IEEE Internet Comput. **12**(6), 95–96 (2008). http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4670126
14. Zhang, C., Budgen, D.: What do we know about the effectiveness of software design patterns? IEEE Trans. Softw. Eng. **38**(5), 1213–1231 (2012)
15. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. J. Internet Serv. Appl. **1**(1), 7–18 (2010). http://www.springerlink.com/index/10.1007/s13174-010-0007-6
16. Zhou, W., Li, L., Luo, M., Chou, W.: REST API design patterns for SDN northbound API. In: 2014 28th International Conference on Advanced Information Networking and Applications Workshops, pp. 358–365. IEEE (2014). http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6844664