Unified Cloud Application Management

Stefan Kolb
Distributed Systems Group
University of Bamberg
Bamberg, Germany
stefan.kolb@uni-bamberg.de

Cedric Röck
Senacor Technologies AG
Nuremberg, Germany
cedric.roeck@senacor.com

***Abstract*—From its early stages, cloud computing has evolved from being a principal source for computing resources to a fully fledged alternative for rapid application deployment. Especially the service model Platform as a Service facilitates the hosting of scalable applications in the cloud by providing managed and highly automated application environments. Although most offerings are conceptually comparable to each other, the interfaces for application deployment and management vary greatly between vendors. Despite providing similar functionalities, technically different workflows and commands provoke vendor lock-in and hinder portability as well as interoperability. To that end, we present a unified interface for application deployment and management among cloud platforms. We validate our proposal with a reference implementation targeting four leading cloud platforms. The results show the feasibility of our approach and promote the possibility of portable DevOps scenarios in PaaS environments.**

Keywords*—*Cloud Computing, Platform as a Service, Portability, Interoperability, DevOps, API

I. Introduction

Even though the cloud and specially Platform as a Service (PaaS) is said to grow massively over the next years [1], [2], there are also plenty of concerns acting as market barriers or preventing further adoption. A major concern is the lack of standards among cloud providers which hinders compatibility and fosters the chances of lock-in effects caused by, e.g., incompatible technologies or proprietary interfaces. Associated application migration costs do not only occur when voluntarily switching the provider but can also arise rather unexpectedly in case of takeovers or the bankruptcy of a provider, making the provider change inevitable. Recent events show that the market is under consolidation which highlights that such circumstances are likely in the cloud market [3], [4]. To enable a truly competitive market and unfold the full potential of cloud services, portability and interoperability between offerings must be enhanced [5].

Application portability between clouds not only includes the functional portability of applications but ideally also the usage of the same service management interfaces among vendors [6], [7]. Unified management interfaces are said to be an important component to accomplish this scenario, as they enable the consistent management of applications across several providers [3], [5], [8]. Whereas the need for a unified interface to manage applications in the cloud is often mentioned in the literature [7], [9]–[14], we argue that, until now, the majority of unified interfaces target the infrastructure provisioning model [6], [15]. Due to differing value propositions and a fundamentally different set of resources and services, cloud platforms must be assessed separately [6]. In this paper, we put a special focus on the portability of the management interfaces while deferring the consideration of the application artifacts to future work. The vast majority of PaaS providers offer self-developed proprietary APIs and tooling suites [11], [15]. Hence, a provider change does not only require the application to be adapted but also urges the developers and operators from familiarizing with different tooling through to adapting existing DevOps automation to new management interfaces. DevOps is a metaphor for the collaboration of the development and IT operation units inside a company, including high task automation to streamline the software delivery process. To mitigate such vendor lock-in effects, this paper presents a unified interface for application deployment and management among cloud platforms. The interface gathers and standardizes core functionalities along the development and application life cycle supported by cloud platforms. We validate our proposal with reference to both, a study of related work and an evaluation of the state of the art. Thereby, we stress that existing approaches do not adequately model necessities for application management in cloud platforms. Additionally, we introduce *Nucleus*, a reference implementation of the presented unified interface targeting four leading cloud platforms and evaluate its utility against typical use cases. The results show the feasibility of our approach and enhance the possibility of portable DevOps scenarios in PaaS environments.

The remainder of the paper is structured as follows. In Section II, we further define our methodology and present our definition of a unified management interface for cloud platforms. Section III presents details of our reference implementation based on the introduced interface and evaluates on the feasibility of the proposed approach. In Section IV, we discuss how our approach contributes to related work. Section V discusses existing limitations and future directions. Finally, Section VI summarizes the contributions of the paper.

II. Unified Management Interface

The aim of the presented interface is to unify core management functions of cloud platforms. Rather than attempting to create a complex matchmaking and migration solution, we focus solely on the creation of a harmonized deployment and management interface. Therefore, all technical dependencies, e.g., supported runtimes as well as contract-specific details such as service-level agreements, are neglected. Those aspects are already targeted by brokering approaches such as [4], [9], [16], [17]. Due to the variety of PaaS systems and their diverging scopes, we argue that an interface covering all available offerings can hardly be defined. We see that cloud

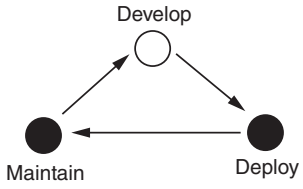

Fig. 1: Application life cycle

platforms can be classified by their proximity to SaaS and IaaS boundaries [4]. Some products are more closely related to SaaS, whereas others have evolved from a more infrastructure-based approach. The majority of offerings, which we address here, are systems that supply a classical application platform that is composed of a set of runtimes, services and other components an application can be programmed to. Especially platforms that are designed towards extending SaaS solutions or visual programming, e.g., Salesforce App Cloud[1], will have requirements for a different set of management interfaces.

Figure 1 shows the management focus of cloud platforms inside a typical application life cycle. Management operations of cloud platforms are clearly focused on the operations part of the life cycle. This includes the creation of the application environment, the deployment of the application itself as well as necessary actions in the maintenance phase. Typical maintenance tasks are monitoring the application's status and initiating reactions to increasing user demand, i.e., scaling the application at runtime. Agile development methods often reiterate the depicted life cycle to update existing applications with new versions. Continuous delivery is a major enabler for reducing effort of these recurring operation tasks [18]. Cloud platforms provide a high amount of automation for these actions along the software life cycle and are, therefore, also well-suited for agile development methods which often traverse the depicted cycle.

In PaaS, not only the application life cycle operations are highly automated but also the delivery of the application environment itself. One of PaaS' key values is to supply a managed environment in which application code can be deployed, freeing the developers and operators from managing the application's runtime environment and connected services. Dependent on the application and its required runtime environment, the platform instantly provisions a container with an operating system and all necessary runtimes installed where the application package can be hosted. An application is typically not self-contained but needs additional services to store information or to outsource tasks for processing. PaaS puts the application at the very center of its focus and supplies a wide variety of pre-configured services that can be provisioned on demand for use by an application. This relieves operators from tedious tasks like setting up databases and handling their availability and scalability. Cloud platforms can handle both, user-faced applications as well as applications acting as services as part of a bigger application architecture. Due to the high automation of the environment and the deployment, cloud platforms are very well suited for microservices architectures which are composed of an array of many small isolated services [19].

[1]See http://www.salesforce.com/platform

Both of the introduced perspectives, the management of the operations during the application life cycle as well as the management of the application environment itself are targeted by the management interfaces of cloud platforms. The intended interface unification should be viable given that a set of management operations of PaaS systems share the same semantics but only use different syntax [11], [20], [21]. Currently, every vendor provides its own, nonstandardized interface with varying sets of supported operations and distinctions in the overall functionality offered to the users. For this reason, the collection of operations that should be included in the core set of the unified interface must be supported by a wide range of vendors. To define such a set of core management functions, besides analyzing the existing works (see Section IV), we also conducted a comprehensive evaluation of 70 vendors in our PaaS knowledge base[2] to be able to homogenize the currently offered capabilities. The described approach eventually led to the selection of operations presented in Table I. It depicts the proposed operations and their references in the existing literature as well as their support by current vendors. Table I also shows that a substantial amount of fundamental and well-supported operations of modern cloud application management is not adequately considered by existing approaches. Due to space limitations, we only depict the compatibility of the vendors that were also implemented in our prototype (see Section III) to exemplify real-world usage, although the overall picture can be applied to more vendors.

The operations are divided into two groups: general operations and application operations. General operations include all tasks that target the management of the platform environment, whereas application operations all relate to a specific application instance created inside the platform environment. The three resources that are administered within the general group are the list of user applications, services, and available deployment regions. The general application operations are user scoped, i.e., listing all applications will only return applications that are accessible by a particular user. Also, the creation of a new application environment is initiated inside the user's platform space. The services resource enumerates all available services that can be bound and used by an application. Although the existing literature considers application services [9], [10], these are limited to database services. However, services provisioned in cloud platforms have gone far beyond only providing database back ends for applications but nearly offer everything as a service from monitoring over messaging to payment services. This evolution was accelerated by the concept of add-on services provided by third party vendors. Typically, services are accounted separately from the normal platform fees of an application which is why an interface must also list their associated payment plans [4]. As native and add-on services cannot be distinguished at most vendors, these are managed by a single interface. Another important capability that must be retrievable are the deployment regions for applications. Often platforms do not only allow the deployment of applications to one geographical server location but offer multiple regions. This is particularly important for customers because of legal and performance reasons [4]. Table II indicates that this functionality is only supported by half of the evaluated vendors directly through their management interface, caused

[2]A comprehensive knowledge base of the state of the art of cloud platforms is available at http://PaaSify.it.

TABLE I: Unified interface operations

		Functionality	Description	Cloud Foundry v2	Heroku	cloudControl	OpenShift v2	Literature
Application operations	App	GET	Get an app entity	✓	✓	✓	✓	[9], [10], [12]
		DELETE	Delete the app	✓	✓	✓	✓	[9], [10], [12]
		UPDATE	Update the app	✓	✓	✗	✗	[9], [10], [12]
	Lifecycle	REBUILD	Rebuild, e.g. to use updated buildpacks	✓	✓	✓	✓	
		DEPLOY	Upload the actual app data	✓	✓	✓	✓	[9], [10], [12]
		DOWNLOAD	Download the current app data	✓	✓	✓	✓	
		START	Start the app	✓	✓	✗	✓	[9], [10], [12]
		STOP	Stop the app	✓	✓	✗	✓	[9], [10], [12]
		RESTART	Restart the app	✓	✓	✗	✓	[10], [12]
	Scaling	ADD INSTANCE	Add new instance, scale horizontally	✓	✓	✓	✓	[10], [12]
		REMOVE INSTANCE	Remove instance, scale horizontally	✓	✓	✓	✓	[10], [12]
		SCALE INSTANCE	Set instance power level	✓	✓	✓	✗	
	Domains	LIST DOMAINS	List all the app's domains	✓	✓	✓	✓	
		GET	Get domain entity	✓	✓	✓	✓	
		ADD DOMAIN	Assign domain to the app	✓	✓	✓	✓	
		DELETE	Delete and remove the domain	✓	✓	✓	✓	
		UPDATE	Update the domain settings	✓	✓	✓	✓	
	Variables	LIST VARS	List all env. variables of the app	✓	✓	✓	✓	
		CREATE VAR	Create a variable with initial value	✓	✓	✓	✓	
		UPDATE VAR	Update an existing variable's value	✓	✓	✓	✓	
		DELETE VAR	Remove a variable	✓	✓	✓	✓	
		GET VAR	Get an environment variable entity	✓	✓	✓	✓	
	Logging	LIST LOGS	Collect the app's logfiles	✓	✓	✓	✓	[10]
		GET SPECIFIC LOG	Get a specific log file	✓	✓	✓	✓	
		DOWNLOAD LOGS	Download all logs as archive	✓	✓	✓	✓	
	Services	ADD SERVICE	Install and bind to the app	✓	✓	✓	✓	[9], [10]
		UPDATE SERVICE	Update bound service settings	✓	✓	✓	✓	
		REMOVE SERVICE	Remove bound service	✓	✓	✓	✓	[9], [10]
		GET	Get bound service entity	✓	✓	✓	✓	
		LIST	List all installed services	✓	✓	✓	✓	
General	App	CREATE	Create the app	✓	✓	✓	✓	[9], [10], [12]
		LIST	List all applications	✓	✓	✓	✓	[9], [10], [12]
	Service	GET	Get available service entity	✓	✓	✓	✓	[10]
		LIST	List all available services	✓	✓	✓	✓	[9], [10]
		GET PLAN	Get service plan entity	✓	✓	✓	✓	
		LIST PLANS	List all available plans for a services	✓	✓	✓	✓	
	Region	GET	Get available region entity	✗	✓	✗	✓	
		LIST	List all available regions	✗	✓	✗	✓	
				95 %	100 %	84 %	95 %	

by a generally missing multi-region support of Cloud Foundry and cloudControl. However, a compensation for the different approach taken by the two vendors to realize this functionality will be provided by our multi-provider concept in the next paragraphs (see Figure 2). Other capabilities like runtime and framework support must be targeted by a knowledge base backed brokering solution as the vendors do not offer this data through their interfaces. Integrating this information is part of our future work (see Section V).

Operations that belong to a specific application resource are the main part of the proposed interface. We agree with the literature on typical actions of the application's life cycle. The life cycle of an application is described by deploying the actual application data, starting, stopping, and restarting the application. Also, this includes getting detailed information, e.g., the status of an application as well as updating its data, deleting the deployment or the entire application space. This life cycle is fully supported by all vendors except cloudControl which directly starts an application at deployment time and allows no further manual state changes. Additionally, the interface includes actions for rebuilding an application that has changed properties which require a redeployment of the application and the download of the current application artifacts. As an essential characteristic of cloud systems [22], elasticity is represented by both horizontal scaling (number of instances) and vertical scaling (instance power). Referring to the services category of the general capabilities, a dedicated group of operations is targeted at the management of services that are bound to an application. First, it is possible to provision or remove a service instance on demand from the services pool of the platform. Furthermore, the array of bound services or details of a dedicated service can be listed. Additionally, as mentioned before, existing literature is lacking a wide range of fundamental operations of cloud platforms. For the remaining groups logging, variables, and domains, we could not find appropriate concepts in the presented approaches in spite of their wide support in current cloud platforms. Logging plays an important role for debugging applications and monitoring application health in the maintenance phase of an application's life cycle. The variety of third party offerings in the service category that target more sophisticated logging and alerting are dependent on the logging functionalities of the main system. Therefore, the interface provides operations for listing available log files, retrieving a particular log file or the whole set of available logs. Using environment variables for configuring properties that are likely to vary between application deployments has become the de facto standard in cloud platforms[3]. This includes resource handles and credentials to services or external add-ons that the application consumes. Last, the ability to assign and manage domains is crucial to address applications and provide them to end users. Typically, cloud platforms manage plenty of customer applications on

[3]See http://12factor.net

Fig. 2: Multi-provider PaaS support

one physical host and no application retains its own dedicated public IP. Routing to application instances via domains is a suitable approach for both, inter-application communication as well as exposing applications to end users.

Even with extensive evaluation, it is challenging to define the right array of operations for a unified interface that satisfies all demands. In that regard, the NIST [6] states that one needs to define minimal standards and avoid overspecification that inhibits innovation. Therefore, we made sure that our selected set of operations is supported by the majority of vendors (see Table I). Yet, a problem often experienced in unifying approaches, caused by this rationale, is the attempt to wrap the smallest common denominator of the competing provider APIs for thorough operation support. As a consequence, an application developer is faced with a dilemma, being forced to either pick a feature-full API created by the chosen cloud provider and risk getting locked-in or favor a unified interface limited to a narrow range of functionality [6], [15]. To mitigate this problem, we decided to include the ability to still gain access to proprietary capabilities, besides the core capabilities of interface, if needed. Therefore, our interface supports a native loop-through functionality, to allow the execution of arbitrary commands against the endpoint's API. This allows the developers to use a combination of unified interactions and proprietary functionalities if necessary.

As we intend to consolidate application management among cloud platforms with our interface, besides the unification of the operations, the need to create an interface concept that provides access to different platforms and adds multi-provider support emerged. Figure 2 illustrates the concept and the associations between vendors, providers, and endpoints. For each platform vendor, there can be an arbitrary number of providers delivering the platform and a provider itself can offer any number of endpoints. Each provider needs to operate the vendor's proprietary API so that all providers and endpoints can be served by one adapter implementation. As an example, the vendor Pivotal develops the platform Cloud Foundry (CF). A provider offers a platform to its customers but must not necessarily have developed it. In this context, IBM Bluemix is an example for a CF provider. An endpoint is the API access point defined by the provider. One provider may offer multiple endpoints. For instance, IBM Bluemix offers two API endpoints to its customers, one serving a CF instance in the United States, the other one providing the European counterpart. With this approach, IBM accounts for the lacking multi-region support of CF (see Table I).

To sum up, Figure 3 shows how the defined operations fit into the resource map of our unified API. The figure does not show all operations that are available but presents an overview of the relations between the resources and how they can be created, resolved or updated. Resource properties are also neglected, except for the associations with other API objects. The figure is separated into four dedicated API groups. The platform group on the top left is responsible for managing available vendors, providers, and endpoints. All other resources are nested below the endpoint to which they belong. The services group is responsible for providing information about the available services within a cloud platform. All subordinated service plans, which belong to exactly one service are nested below the services resource path. Equally, the available deployment regions are managed inside the region group. Both of these resources are read-only and belong to the currently connected endpoint. At the bottom of the figure, the application group includes all application operations, i.e., the list and instance retrieval for all applications, domains, logs, variables, and installed services. Applications are nested below the endpoint, whereas the domains, logs, variables and installed services are sub resources belonging to an application. The unified interface also introduces properly named and structured API objects for all of the shown resources. Additionally, a unified application lifecycle and error schema is defined. Due to space limitations, we have to omit a detailed presentation of these finer aspects and therefore refer to the technical report [23].

Next, the feasibility of the approach will be validated by providing a reference implementation of the defined interface and by evaluating the prototype against several use cases.

III. Reference Implementation

In this chapter, the design of our reference implementation Nucleus[4] is discussed. We evaluate the details of the concrete implementation as well as the challenges that must be targeted to integrate and map existing offerings to the unified management interface presented in Section II.

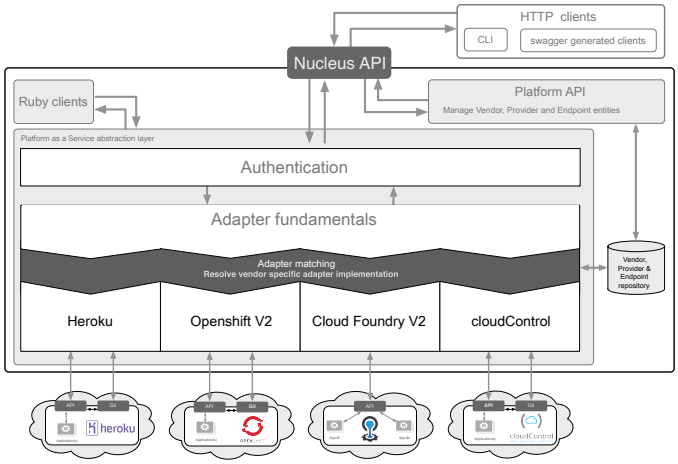

Fig. 4: Nucleus architecture

Figure 4 shows the overall architecture of the reference implementation. At the heart of the implementation, the unified Nucleus API is served to clients. One of the technical requirements for the API is to provide a platform and programming language independent abstraction layer. Here, we decided to create a RESTful API with the use of JSON/HTTP. By using these technologies, the API can be easily extended with

[4]See https://github.com/stefan-kolb/nucleus

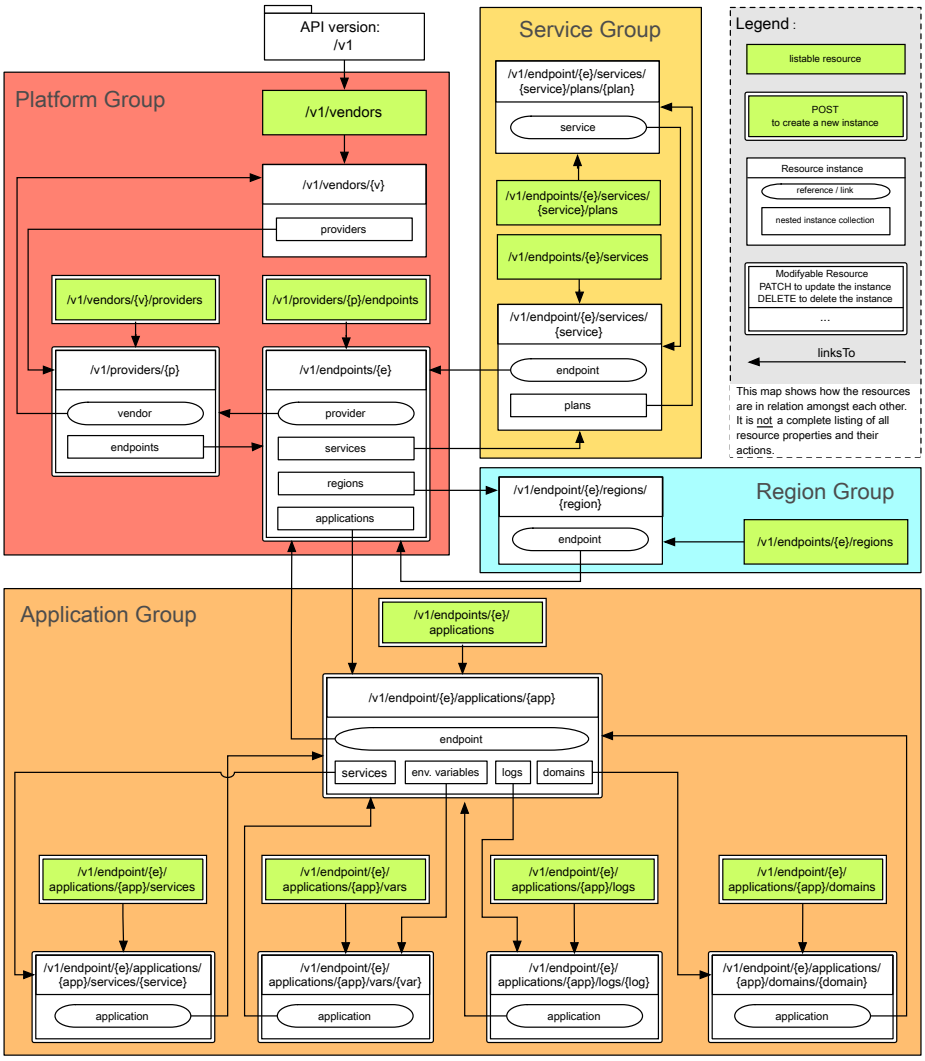

Fig. 3: Unified interface resource map

command line clients and other tools or can directly be queried by any client that is able to issue HTTP requests. Nevertheless, due to the popularity of language-specific wrappers for APIs, the Nucleus API is also available as a Ruby Gem. Nucleus' extendability is provided by a modular structure with dedicated adapters for each supported platform. A set of providers and endpoints for all adapters is available by default. This set can also be altered through the API which allows the addition of providers and endpoints at runtime. If a new provider should be added or a provider changes its API, only the adapter implementation must be adjusted. In this way, Nucleus can maintain long-term stability for tools and automation scripts and backward compatibility across different API versions for the available vendors. Consequently, also changes to the unified API must be versioned appropriately in the future. Between the API and the adapters, an independent authentication layer capable of different authentication mechanisms, e.g., OAuth or HTTP Basic, is provided that can be reused inside the adapter implementations. The entire Nucleus server implementation may be hosted as local instance as well as a public instance with multi-user support.

The current implementation includes adapters for four leading cloud platforms. These are Cloud Foundry, Heroku, Openshift, and cloudControl[5] (see Table II). Including their providers, Nucleus is able to support more than 12 public cloud platforms as well as any private deployment. The decision on possible candidates for the adapter implementations was assisted by the knowledge base and cloud brokering tool *PaaSify* [4]. Besides trying to cover a variety of heterogeneity with the prototype, in terms of technological implementations to support new vendors in future releases without the need for major modifications, the vendor's impact and popularity within the market played an important role. In fact, OpenShift, Heroku, and Cloud Foundry were the top three vendors queried on *PaaSify*. All of them are available as public and private cloud deployments[6]. Our approach does not require any additional documents, e.g., application descriptors but can be used without adaption for already existing applications. Those facts contribute to making the abstraction layer applicable for a wide

[5]cloudControl was shutdown due to bankruptcy end of February 2016.

[6]Heroku Private Spaces offers isolated cloud deployments but no hosting on private data centers.

TABLE II: Selected vendors for adapter implementations

	cloudControl	Cloud Foundry	Heroku	OpenShift
Type	Proprietary	Open Source	Proprietary	Open Source
Hosting	public, private	public, private	public, (private)	public, private
Providers	4	>5	1	2

variety of offerings, both hybrid and multi-cloud, empowering its practical utility.

One of our main points of criticism is that most of the proposed approaches are missing an evaluation through an existing and published implementation of their specification. In our opinion a unified interface can only be validated and improved by a working reference implementation. The importance of reference implementations cannot be stressed enough as conceptual problems can only be reasonably discovered in practical use cases and evaluations. Whereas, our implementation generally proves that it is possible to implement the suggested unified interface and make different existing vendors conform to the defined operations and resources, we experienced various issues while creating and evaluating the implementation.

One of the main challenges for achieving the abstraction to the unified API is the mapping of proprietary operations and resources to the unified operations and resource schemata. Although Table I shows broad support for the majority of the defined operations, these transformations are mostly not 1:1 syntactical mappings but require a series of requests on native API resources to gather required properties and initiate all necessary operation steps. Whereas we have to omit a detailed consideration of the mappings, Table III shows an aggregated view of the mapping overhead. The table points out how many HTTP requests have to be sent in order to achieve the semantically same result on all platforms. We only counted operations that are supported by the respective vendor and actually conduct API requests.

TABLE III: Native API requests per unified operation

	cloudControl	Cloud Foundry	Heroku	OpenShift
Operations	31	35	36	30
Total API requests	47	63	49	39
Avg. API req/operation	1.52	1.8	1.36	1.3

A main source for evaluating the utility of the prototype are the designed adapter tests that do not only test all of the available operations but simulate the complete life cycle of cloud applications (see Figure 1 and 5). The use cases directly interact with the vendor APIs and record all HTTP interactions that are then matched with the expected results and unified resource structures proving the correct functioning of the implementation. For every vendor, a complete cycle is traversed. This includes the creation of an application, provisioning and configuring routes to required services, deploying the application data, and starting the application. Moreover, it covers the monitoring of application logs, scaling and recovering instances caused by load and instance failures up to an application update (see Figure 5). Furthermore, the use cases do not alone cover positive paths but also failing actions during the application life cycle. The feasibility of

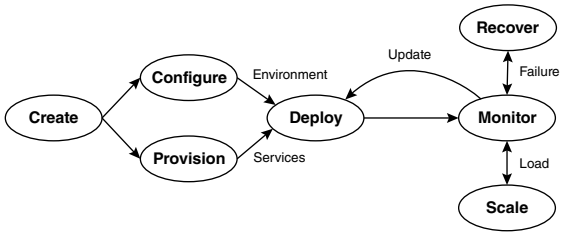

Fig. 5: Application management use cases

the operations defined by the interface are thereby not only validated in isolation but in relation with each other.

At the beginning, it was claimed that portability among PaaS can be improved with the creation of a unified interface. In summary, application portability is fostered by the common operation and object model of our unified interface that applies to all supported platforms. Diversities could be successfully harmonized amongst the four platforms. With its unified deployment and management capabilities, Nucleus allows to manage applications on different platforms. Most important, even though a fully automated vendor change is not yet viable, the effort that is needed when migrating an application to another vendor is diminished. In case of switching the provider, the effort to adapt the application's surrounding, for instance to enable continuous delivery and DevOps, can be minimized. Hybrid and multi-cloud deployments are facilitated due to the fact that the management operations can be handled consistently if all vendors support a unified interface.

IV. Related Work

In the past, several drafts for unified cloud interfaces were published as independent work or as part of a broader approach. As stated, we argue that a majority of them were focused on the infrastructure provisioning model, missing out on cloud platforms [6], [15]. Furthermore, we show that existing approaches for PaaS do not adequately consider core functionalities of modern cloud platforms. Often, the approaches are focused on supporting a unified deployment of applications but do not apply a more holistic view of the fundamental management capabilities. The following paragraphs provide an overview of related work and give distinction of how our work differs and contributes to the existing approaches.

A. Standards

A number of standards that are often still in the process of making are proposed by several standardization organizations. In many respects, the use of standards is counterproductive to the vendors' aim to achieve a strong market position. Therefore, most standard proposals suffer from the lack of acceptance and participation by industry leaders that prevents adoption.

Originally initiated to create a remote management API for cloud infrastructures, the *Open Cloud Computing Interface* (OCCI) claims to be a generic protocol and API to serve other models besides IaaS. However, extending the core specification [24], apart from a draft [25], there still only exists a normative specification for infrastructures [26].

The *Cloud Infrastructure Management Interface* (CIMI) specification [27] explicitly states to be solely focused on infrastructure management. It consists of a model for cloud infrastructure resources as well as a standardized REST over HTTP interface to manage the defined resources.

In contrast, *Cloud Application Management for Platforms* (CAMP) [28] focuses on providing a management API for cloud platforms. The recent draft of the standard proposal describes generic operations and artifacts that a PaaS cloud should ideally offer. It is currently awaiting the specification as an official OASIS standard, requiring the evidence of interoperable implementations which is still missing to date. CAMP is designed to be language, framework, and platform neutral with the goal of covering a variety of the PaaS ecosystem. The operations that CAMP specifies include building, life-cycle management, administration, and monitoring tasks. Nevertheless, we argue that the current scope and definition of operations and resources is too generic and it would require an enormous customization effort to integrate it with today's cloud platforms.

The *Topology and Orchestration Specification for Cloud Applications* (TOSCA) [29] specifies a generic metamodel for defining cloud applications. This includes both, the structure of an IT service as well as how to instantiate and manage it. However, the specification is more focused on defining the topology and packaging of services than the actual management of a hosting platform or the application itself.

B. Abstraction Layers

Abstraction layers are a solution by means of harmonizing a variety of different systems, which all share a common principle, behind one interface. In contrast to standards, this approach often comes in conjunction with adapters for mapping the abstraction layer interface to the native APIs without the need for firsthand vendor support. A variety of projects are available in the area of cloud computing which follow a concept that is related or similar to our approach.

A technique applied by some projects is to duplicate popular APIs from leading vendors and supply a different implementation. Two examples for such interface clones are Eucalyptus [30] that is based on Amazon's EC2 API and AppScale [31] that mirrors parts of the Google App Engine API. This concept is typically reasonable for Open Source clones of proprietary offerings that try to supply the exact same functionalities as the base vendor. However, it is less feasible to unify a variety of technically differing offerings.

Apache Deltacloud[7] is an abandoned attempt to improve the interoperability amongst various infrastructure providers. It provides a set of RESTful APIs that allow to interface with a multitude of providers. The supported APIs include a custom Deltacloud API, the standardized CIMI API, and a clone of Amazon's EC2 API. Besides Deltacloud, there are also several language-specific abstraction layers for IaaS, e.g., jclouds[8], Fog[9], pkgcloud[10] or Libcloud[11]. Our approach combines both of the described styles, but for PaaS, as it serves a programming language independent RESTful API and a Ruby wrapper library.

Cunha et al. [10] propose PaaS Manager, an approach similar to ours, defining a set of common operations abstracting the differences of application deployment and life cycle management of multiple providers. PaaS manager declares to support adapters to three platforms, i.e., CloudBees, Cloud Foundry, and Heroku. However, to our knowledge, a proof-of-concept implementation has never been publicly released. From what can be deducted from [10], [32], their API definition is lacking important environment configuration operations, e.g., domains and multi-region support.

Similarly, Sellami et al. [12] introduced the *Compatible One Application and Platform Service* (COAPS) API. Their implementation includes connectors for Cloud Foundry, Openshift, and Google App Engine [12], [33]. Compared to our findings, COAPS is missing various essential operations of typical cloud platforms, like scaling, monitoring of applications or the management of services and deployment regions [12], [34].

Another EU funded project from which a unified managing approach for PaaS evolved was Cloud4SOA [9]. The functionality of Cloud4SOA was partially transferred in subsequent projects CloudPier and most recently SeaClouds [16]. Cloud4SOA provides four core capabilities, of which one is the unified management and deployment of applications to cloud platforms. According to the publications, adapters were offered for AWS Elastic Beanstalk, Cloud Foundry, Openshift, and CloudBees [8], [9], later extended with Heroku and cloudControl. Akin to the previously described approaches, the presented unified interface does not consider important standard functionalities. For Cloud4SOA this especially concerns application management, e.g. domains, logging, scalability, deployment regions, and also a dedicated ability to manage application services.

V. Limitations and Future Work

Even though the presented unified interface already copes with the core aspects of today's cloud platforms, there are still open challenges remaining which can be worked on in future projects.

Conceptually, PaaS is still evolving and so do the management interfaces. Whereas we integrated flexibility to cover this case via the native loop-through functionality, also the unified interface will need revisions and upgrades over time to provide an appropriate abstraction of the state of the art. Two examples of upcoming features are application environments and multi-application services. Whereas services in general are already supported, some platforms allow the creation of services that must not be bound but can be used across application boundaries. Also, environments to manage different versions of an application for, e.g., staging and production, could be one of the major changes in a next revision. With agile development, multiple environments are becoming a first class feature for developers that they also want to use in the cloud. As these features are not supported by a wide range of platforms, we could not reasonably include them in the current revision of the interface.

[7] See http://deltacloud.apache.org
[8] See http://jclouds.apache.org
[9] See http://fog.io
[10] See https://github.com/pkgcloud/pkgcloud
[11] See https://libcloud.apache.org

Although our contributions allow for the deployment of applications through a unified interface, we also have to take into account that the implementation artifacts needed to deploy onto the PaaS may be different [11]. Possible solutions for this have to be investigated independently.

Extending our reference implementation, several starting points can be identified. Naturally, the range of applicability of the tool could be enhanced by additional adapters. Furthermore, we could serve multiple APIs in front of our implementation. Especially, if standard efforts like CAMP mature further, we could integrate these to into our prototype and provide a standard-compliant interface to multiple providers straightaway. Another beneficial project would be to create a console client for our unified API. This would considerably enhance the value for end users, as it is the preferred way for users and developers to interact with cloud platforms.

Moving away from the previously described implementation tasks to a more general perspective, Nucleus could be integrated with the PaaS knowledge base [4] to gain additional semantic insights on cloud platforms and their capabilities. Nucleus could then prevent semantic errors before they appear, for instance by warning that a certain runtime is not supported by the chosen provider or by serving additional information about available platform capabilities. Combining both projects, the emerging system would evolve to become a brokering solution that can not only identify the right platform for the user's needs but also enhances usage without having to fear the effects of vendor lock-in.

VI. Conclusion

The inherent need for a unified interface to manage applications in the cloud is stressed in multiple research papers [7], [9]–[14]. Whereas standards and approaches for the infrastructure provisioning model have already gained traction, proposals for cloud platforms are still premature. The focus on a high level of DevOps automation in Platform as a Service further stresses the need for a homogeneous approach among vendors. Therefore, we presented a unified interface to manage applications in cloud platforms. The results both build upon extending former approaches and an extensive evaluation of the state of the art. We validated our proposal with a reference implementation mediating to four leading platform vendors making the abstraction layer applicable for a wide variety of offerings, both hybrid and multi-cloud. The presented unified management interface for cloud platforms together with our reference implementation *Nucleus* increases the portability and interoperability of PaaS applications and thus helps to avoid critical vendor lock-in effects.

References

[1] F. Gens, "Worldwide and Regional Public IT Cloud Services 2014–2018 Forecast," IDC, Tech. Rep., 2014.

[2] F. Biscotti *et al.*, "Market Trends: Platform as a Service, Worldwide, 2013-2018, 2Q14 Update," Gartner, Tech. Rep., 2014.

[3] S. Marston *et al.*, "Cloud computing – The business perspective," *Decision Support Systems*, vol. 51, no. 1, 2011.

[4] S. Kolb and G. Wirtz, "Towards Application Portability in Platform as a Service," in *Proc. Symp. Service-Oriented System Engineering*, 2014.

[5] K. Oberle and M. Fisher, "ETSI CLOUD – Initial Standardization Requirements for Cloud Services," in *Economics of Grids, Clouds, Systems, and Services*. Springer, 2010.

[6] M. Hogan *et al.*, "NIST Cloud Computing Standards Roadmap," *NIST Special Publication 500-291*, 2011.

[7] D. Petcu, "Portability and Interoperability between Clouds: Challenges and Case Study," in *Towards a Service-Based Internet*. Springer, 2011.

[8] E. Kamateri *et al.*, "Cloud4SOA: A Semantic-Interoperability PaaS Solution for Multi-cloud Platform Management and Portability," in *Service-Oriented and Cloud Computing*. Springer, 2013.

[9] F. D'Andria *et al.*, "Cloud4SOA: Multi-cloud Application Management Across PaaS Offerings," in *Proc. Symp. Symbolic and Numeric Algorithms for Scientific Computing*, 2012.

[10] D. Cunha *et al.*, "PaaS Manager: A Platform-as-a-service Aggregation Framework," *Computer Science and Information Systems*, vol. 11, no. 4, 2014.

[11] S. Kolb *et al.*, "Application Migration Effort in the Cloud - The Case of Cloud Platforms," in *Proc. Conf. Cloud Computing*, 2015.

[12] M. Sellami *et al.*, "PaaS-Independent Provisioning and Management of Applications in the Cloud," in *Proc. Conf. Cloud Computing*, 2013.

[13] Cloud Computing Use Case Discussion Group, "Cloud Computing Use Cases White Paper – Version 4.0," 2010.

[14] N. Loutas *et al.*, "A Semantic Interoperability Framework for Cloud Platform as a Service," in *Proc. Conf. Cloud Computing Technology and Science*, 2011.

[15] D. Petcu *et al.*, "Towards a cross platform cloud API," in *Proc. Conf. Cloud Computing and Services Science*, 2011.

[16] A. Brogi *et al.*, "EU Project SeaClouds - Adaptive Management of Service-based Applications Across Multiple Clouds," in *Proc. Conf. Cloud Computing and Services Science*, 2014.

[17] B. Surajbali and A. Juan-Verdejo, "A Marketplace Broker for Platform-as-a-Service Portability," in *Advances in Service-Oriented and Cloud Computing*. Springer, 2015.

[18] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, 2010.

[19] J. Lewis and M. Fowler, "Microservices," 2014, http://martinfowler.com/articles/microservices.html.

[20] A. Sheth and A. Ranabahu, "Semantic Modeling for Cloud Computing, Part 2," *IEEE Internet Computing*, vol. 14, no. 4, 2010.

[21] N. Loutas *et al.*, "Towards a Reference Architecture for Semantically Interoperable Clouds," in *Proc. Conf. Cloud Computing Technology and Science*, 2010.

[22] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *NIST Special Publication 800-145*, September 2011.

[23] C. Röck and S. Kolb, "Nucleus – Unified Deployment and Management for Platform as a Service," University of Bamberg, Tech. Rep., 2016.

[24] Open Grid Forum, "Open Cloud Computing Interface - Core," 2011.

[25] ——, "Open Cloud Computing Interface - Platform," 2016.

[26] ——, "Open Cloud Computing Interface - Infrastructure," 2011.

[27] DMTF, "Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol – An Interface for Managing Cloud Infrastructure," 2012.

[28] OASIS, "Cloud Application Management for Platforms Version 1.1," 2014.

[29] ——, "Topology and Orchestration Specification for Cloud Applications Version 1.0," 2013.

[30] D. Nurmi *et al.*, "The Eucalyptus Open-Source Cloud-Computing System," in *Proc. Symp. Cluster Computing and the Grid*, 2009.

[31] N. Chohan *et al.*, "Appscale: Scalable and Open AppEngine Application Development and Deployment," in *Cloud Computing*. Springer, 2010.

[32] D. Cunha *et al.*, "A Platform-as-a-Service API Aggregator," in *Advances in Information Systems and Technologies*. Springer, 2013.

[33] E. Hossny *et al.*, "A Case Study for Deploying Applications on Heterogeneous PaaS Platforms," in *Proc. Conf. Cloud Computing and Big Data*, 2013.

[34] Telecom SudParis, Computer Science Department, "The Compatible One Application and Platform Service (COAPS) API specification – Version 1.5.3," 2013.