BAMBERGER BEITRÄGE

ZUR WIRTSCHAFTSINFORMATIK UND ANGEWANDTEN INFORMATIK

ISSN 0937-3349

Nr. 100

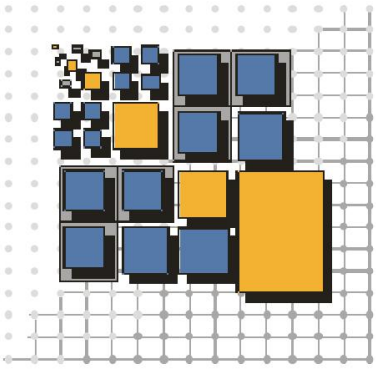
Nucleus – Unified Deployment and Management for Platform as a Service

Cedric Röck, Stefan Kolb

April 2016

FAKULTÄT WIRTSCHAFTSINFORMATIK UND ANGEWANDTE INFORMATIK

OTTO-FRIEDRICH-UNIVERSITÄT BAMBERG



Nucleus – Unified Deployment and Management for Platform as a Service

Cedric Röck, Stefan Kolb

Lehrstuhl für Praktische Informatik, Fakultät WIAI
Otto-Friedrich-Universität Bamberg
An der Weberei 5, 96047 Bamberg

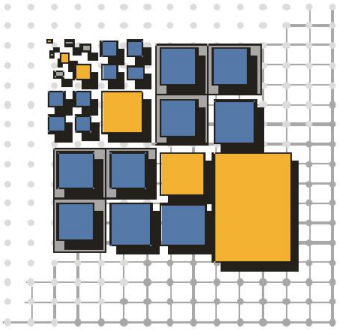
`https://github.com/stefan-kolb/nucleus`

Abstract

Cloud computing promises several advantages over classic IT models and has undoubtedly been one of the most hyped topics in the industry over the last couple of years. Besides the established delivery models Infrastructure as a Service (IaaS) and Software as a Service (SaaS), especially Platform as a Service (PaaS) has attracted significant attention these days. PaaS 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. In this study, we present the tool Nucleus, which realizes a unified interface for application deployment and management among cloud platforms. With its help, we aim to increase the portability of PaaS applications and thus help to avoid critical vendor lock-in effects.

Keywords

Cloud Computing, Platform as a Service, Portability, Interoperability, API



Distributed Systems Group

Otto-Friedrich Universität Bamberg

An der Weberei 5, 96047 Bamberg, GERMANY

Prof. Dr. rer. nat. Guido Wirtz

https://www.uni-bamberg.de/pi

Due to hardware developments, strong application needs and the overwhelming influence of the net in almost all areas, distributed systems have become one of the most important topics for nowadays software industry. Owing to their ever increasing importance for everyday business, distributed systems have high requirements with respect to dependability, robustness and performance. Unfortunately, distribution adds its share to the problems of developing complex software systems. Heterogeneity in both, hardware and software, permanent changes, concurrency, distribution of components and the need for interoperability between different systems complicate matters. Moreover, new technical aspects like resource management, load balancing and guaranteeing consistent operation in the presence of partial failures and deadlocks put an additional burden onto the developer.

The long-term common goal of our research efforts is the development, implementation and evaluation of methods helpful for the realization of robust and easy-to-use software for complex systems in general while putting a focus on the problems and issues regarding distributed systems on all levels.

Our current research activities are focused on different aspects centered around that theme:

- *Reliable and inter-operable Service-oriented Architectures*: Development of design methods, languages, tools and middle-ware to ease the development of SOAs with an emphasis on provable correct systems that allow for early design-evaluation due to rigorous development methods. Additionally, we work on approaches and standards to provide truly inter-operable platforms for SOAs.
- *A Comparative Framework for Process Runtime Environments*: Within this research project, a comparative framework for process engines in general is developed. This framework is applied to and validated with BPEL as well as BPMN engines. It focuses on the comparison of standard conformance, performance, error detection as well as automation.
- *Measuring and Improving the Quality of BPMN Process Models*: We elaborate on proposals for measuring and improving the quality of human-centric process models on different layers of abstraction. In order to improve the interoperability between modeling tools we perform an extensive analysis of the standard document to extract all rules and constraints for correct BPMN process models. Modeling tools and model instances must comply with those constraints in order to be BPMN compliant and to enable interoperability.

- *Cloud Application Portability:* We examine important aspects of portability in PaaS cloud environments and enhance the portability of cloud applications by applying common standards between heterogeneous clouds. We try to make use of a holistic view of the cloud including important aspects like cloud specific restrictions, platform configurations, the deployment and life cycle of cloud applications.
- *Visual Programming- and Design-Languages*: The goal of this long-term effort is the utilization of visual metaphors and languages as well as visualization techniques to make design- and programming languages more understandable and, hence, more easy-to-use.

More information about our work, i.e., projects, papers and software, is available at our homepage (see above). If you have any questions or suggestions regarding this report or our work in general, don't hesitate to contact me at `guido.wirtz@uni-bamberg.de`

Guido Wirtz
Bamberg, April 2016

Contents

List of Figures

List of Tables

Listings

Abbreviations

API	Application Programming Interface
CAMP	Cloud Application Management for Platforms
CI	Continuous Integration
CLI	Command-Line Interface
CRUD	Create Read Update Delete
DAO	Data Access Object
DMTF	Distributed Management Task Force
FQDN	Fully Qualified Domain Name
HAL	Hypertext Application Language
HATEOAS	Hypermedia as the Engine of Application State
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
ISV	Independent Software Vendor
JSON	JavaScript Object Notation
OVF	Open Virtualisation Format
PaaS	Platform as a Service
SaaS	Software as a Service
SME	Small and Medium Enterprises
SSL	Secure Sockets Layer

1 Introduction

Cloud computing promises several advantages over classic IT models and has undoubtedly been one of the most hyped topics in the industry over the last couple of years. Besides the established delivery models Infrastructure as a Service (IaaS) and Software as a Service (SaaS), especially Platform as a Service (PaaS) has attracted significant attention these days. PaaS facilitates the hosting of scalable applications in the cloud by providing managed and highly automated application environments.

In recent surveys, Independent Software Vendors (ISVs) and Small and Medium Enterprises (SME) revealed why they adopted or plan to adopt cloud services in the first place. Most important, the customers highlighted that they no longer have to pay for their IT infrastructures up front. Instead, they can consume the required resources like a utility, pay per use and therefore cut down their investment risks. Further reasons are the expected reduction of the total cost of ownership and the immediate access to new IT resources, which consequently also shortens the time-to-market [KPM13; The12; PHMH09; Bad12]. Driven by these benefits, cloud computing experienced steady growth and is anticipated to grow even further [Pri10]. A study of Gartner recently estimated a compound annual growth rate of 17 % between 2012 and 2017 for public cloud services [Car13], whereas KPMG forecasts a compound annual growth rate of up to 25 % until 2016 [KPM13]. The International Data Corporation tops even those estimations and expects the PaaS market to grow annually with almost 30 % and reach a market volume of more than $20 billion in 2017 [Gen14].

Even though the cloud market is said to expand massively over the next years, there are also plenty of concerns acting as market barriers and hindering further cloud adoption. ISVs and SMEs named their biggest concerns regarding the adoption of cloud services in a series of recent surveys. According to them, security and privacy concerns became more and more important over the last couple of years and nearly half of the participants fear the loss of control over their IT systems. Another major concern is the lack of standards between cloud providers, which hinders compatibility and fosters the chances of lock-in effects, especially vendor lock-in [The12; KPM13; Rig14]. In cloud computing, a vendor lock-in can be caused by multiple aspects, e.g., incompatible technologies, proprietary interfaces or even missing operations to extract application data. It bears the immediate risk of a decline in service quality, unjustified price increases and substantial migration costs which can even threaten the survival of ISVs and SMEs when the vendor is changed [FK06]. Migration costs must not only be accounted when voluntarily switching the provider, but can also arise rather unexpectedly in case of takeovers, competitors leaving the market or the bankruptcy of a provider, making the provider change inevitable [MLBZG11]. Recent events highlight the market being under consolidation, with acquisitions[1] taking place and providers terminating their operations[2].

Nowadays, the still young PaaS market is heavily fragmented, hence, a multitude of heterogeneous PaaS offerings are available at the market[3]. According to the providers' websites, as well as mentioned in various studies [PCR11; CCP14; BBSR13; CH09], the vast majority of PaaS providers offer a self-developed proprietary Application Programming Interface (API)

[1] *Is PaaS becoming just a feature of IaaS?*. URL: `https://451research.com/report-short?entityId=79800` (Retrieved: March 22, 2016).

[2] *CloudBees Becomes the Enterprise Jenkins Company*. URL: `http://blog.cloudbees.com/2014/09/cloudbees-becomes-enterprise-jenkins.html` (Retrieved: March 22, 2016).

[3] *Platform as a Service Provider Comparison*. URL: `http://www.paasify.it` (Retrieved: March 22, 2016).

and tooling suite. Only a small number of providers are somewhat compatible amongst each other, but mainly for the reason that they were built using the same product, e.g., an open source PaaS. The compatibility between PaaS systems can be separated into two categories, portability and interoperability. Portability which could theoretically allow seamless provider switches [PMPC13] is hindered for many reasons. In particular, completely different technological stacks prevent an application to be moved from one provider to another [Bad12]. A provider change most likely does not only require the application to be adapted, but also forces the developers and operators to familiarize with the new API and reimplement the surrounding of integrated applications [EK12]. However, this reimplementation need is caused by lacking interoperability. Semantic interoperability between cloud applications is an essential part of hybrid and federated cloud services, but suffers from the variety of different and incompatible APIs and tools that the providers offer [PHMRS12; ZDB+13; OF10]. Without interoperability, roll-out scripts and management applications require specific implementations for each provider and changes must be made if the provider is switched.

Whereas there are several ways to prevent lock-in on the IaaS layer, e.g., by using the Open Virtualisation Format (OVF), PaaS systems are still prone to vendor lock-in. In order to enable a truly competitive market and unfold the full potential of PaaS, which theoretically allows the immediate service adoption and abandonment, portability and interoperability must be enhanced [OF10]. Standardized management interfaces, or standards in general, are said to be an important component to realize this scenario, as they enable the consistent management of cloud applications across several providers. They support the resolution of portability and interoperability issues and proved themselves in many other markets [OF10; KLZ+13; MLBZG11]. Nevertheless, no published and widely adopted PaaS standard exists until today. A number of standards that are still in the process of making are, for instance, OASIS's Cloud Application Management for Platforms (CAMP) [OAS14], Cloud Portability and Interoperability Profiles (P2301) and Standard for Intercloud Interoperability and Federation (P2302) of the Institute of Electrical and Electronics Engineers (IEEE), as well as the results of the Distributed Management Task Force (DMTF)'s Cloud Management Working Group. However, even though some promising standards are being developed, most competitors are likely not going to adopt them on a voluntary basis. Vendors and providers develop their custom tools and interfaces not only to distinguish themselves from their competitors, but also to tie their customers. Most of their revenues are based on a mixture of subscription-fee and pay-per-use models, which is why they have an interest in preventing their customers to leave and join a competitor [GS12].

Beyond the use of standards, abstraction layers are another feasible approach to overcome several of the relevant issues. By abstracting the differences of the proprietary APIs, only one unified interface is offered to the users. Facing the current market situation, including the mentioned chances, obstacles and approaches, our research question for this report is defined as follows:

"Is it possible to abstract the differences of vendor-specific deployment and management interfaces of PaaS systems by creating an intermediary abstraction layer?"

Concerning the mitigation of vendor lock-in effects, this report introduces an approach that identifies and implements a generic PaaS management and deployment API as a language independent interface. In this study, *Nucleus*[4], a prototype of the abstraction layer, which

[4] This report ist based on version 0.2 of Nucleus. Please see `CHANGELOG.md` for changes in newer versions. Nucleus is available at `https://github.com/stefan-kolb/nucleus`

is supposed to enhance the interoperability and portability between PaaS cloud services regarding their deployment and management tasks, is defined and evaluated.

The structure of this report is subdivided according to the major phases of the prototype's development. First, in Chapter 2, the basic approach is introduced to outline all required steps for the creation of the abstraction layer. Next, Chapter 3 refines the basic approach and defines the detailed plan to guide the implementation phase. Important aspects of the implementation and noteworthy issues are summarized in Chapter 4. An evaluation of the prototype is conducted in Chapter 5 to analyze whether the initial requirements could be fulfilled. Finally, Chapter 6 states future work before Chapter 7 sums up the results of the report.

2 Approach

Today, building and deploying PaaS applications is often a well-supported task. Due to the provider's Command-Line Interface (CLI) and API tooling, the configuration of a user's complete development environment can be tailored to integrate with the PaaS platform. However, changing the provider becomes increasingly complicated the further the application is integrated into the surrounding environment, for instance if continuous delivery or deployment is used.

The goal of this report, the creation of an abstraction layer for PaaS deployment and management APIs, shall help to simplify these migration efforts. From now on, *Nucleus* is going to be used as codename for the prototype. The intended abstraction should be realizable given that the management operations of PaaS systems share the same semantics, but only use different syntax [SR10]. Our scope is to focus on the deployment and management capabilities of the providers' APIs, while neglecting the functional interfaces, i.e., the deployment artifacts [HLST11]. Moreover, all technical dependencies, e.g., the supported runtimes and database systems as well as contract-specific details such as service-level agreements, are disregarded. Handling those aspects would enhance the portability even further, but this is already part of multiple studies [ZDB+13; PMPC13; BIS+14; GCN+13], which result in brokering scenarios that exceed this work's scope.

Before starting with the design of the prototype and its implementation, several aspects have to be analyzed. In the following subsections, important foundations for building the prototype are going to be described. Section 2.1 outlines the applied criteria to select a set of PaaS vendors and presents the chosen vendors. Their APIs are evaluated and compared in Section 2.3, based on identified similarities and differences. Finally, Section 2.4 reveals the planned structure of the prototype.

2.1 Vendor Selection Criteria

Especially since a prototype is going to be created in this work, at first, only a few selected PaaS systems can be supported. The selection depends on many criteria, e.g., the management interfaces, deployment types, runtime languages, and the vendor's popularity within the community. It is intended to cover a wide variety of heterogeneity with the prototype. For instance, the vendors shall be chosen to cover all deployment methods to theoretically support new vendors in future releases without the need for major modifications.

In Table 1, a set of relevant candidates and their system's characteristics are evaluated. The information was obtained from *PaaSify.it*[5] and by evaluating the provider's documentation. Providers that were not production-ready at the time of writing were excluded from the evaluation.

[5] *Platform as a Service Provider Comparison.* URL: `http://www.paasify.it` (Retrieved: March 22, 2016).

Provider	System	CLI	CLI Deployment	HTTP Deployment	Git Deployment	GUI Deployment	Free Plan	Runtimes	Extensible	Automatic Scaling	Horizontal Scaling	Vertical Scaling	Isolation
App42[6]	-	✓	✓	✗	✗	✓	t	Groovy, Java, Node.js, PHP, Python, Ruby	✗	✗	✓	✓	Kontena
AWS Elastic Beanstalk[7]	-	✓	✓	✓	✗	✓	t	.NET, Java, Node.js, PHP, Python, Ruby	✗	✓	✓	✓	VMs
IBM Bluemix[8]	cf	✓	✓	✓	h	✗	t	Java, Node.js, Ruby	✓	✗	✓	✓	Warden
cloudControl[9]	cc	✓	✓	✗	✓	✓	✓	Java, Node.js, PHP, Python, Ruby	✓	✗	✓	✓	LXC
Cloud Foundry[10]	-	✓	✓	✓	✗	✗	p	Go, Groovy, Java, Node.js, Ruby, Scala	✓	✗	✓	✓	Warden
dotCloud[11]	cc	✓	✓	✗	✓	✗	✓	Java, Node.js, Perl, PHP, Python, Ruby	✓	✗	✓	✓	LXC
Exoscale[12]	cc	✓	✓	✗	✓	✗	✗	Clojure, Java, Node.js, PHP, Python, Ruby, Scala	✓	✗	✓	✓	LXC
Heroku[13]	-	✓	✗	✓	✓	✗	✓	Clojure, Groovy, Java, Node.js, PHP, Python, Ruby, Scala	✓	✗	✓	✓	LXC
HP Helion[14]	cf	✓	✓	✓	✗	✗	t	Go, Groovy, Java, Node.js, Ruby, Scala	✓	✓	✓	✓	Warden
Jelastic[15]	-	✗	✗	✓	✓	✓	t	Java, Node.js, PHP, Python, Ruby	✓	✓	✓	✓	?
Microsoft Azure[16]	-	✓	✓	✓	✓	✗	t	.NET, Java, Node.js, PHP, Python, Ruby	✓	✓	✓	✓	Hypervisor
Nodejitsu[17]	-	✓	✓	✓	h	✗	✗	Node.js	✗	✗	✓	✓	SmartOS Zone
OpenShift V2[18]	-	✓	✗	✗	✓	✗	✓	Java, Node.js, Perl, PHP, Python, Ruby	✓	✓	✓	✓	Warden
Stackato[19]	cf	✓	✓	✓	✗	✗	p	Clojure, Go, Groovy, Java, Node.js, Perl, PHP, Python, Ruby, Scala	✓	✓	✓	✓	Docker

LEGEND

Core System	*cc*	: cloudControl	*cf*	: Cloud Foundry
Git Deployment	*h*	: Via Git hooks		
Free Plan	*t*	: Trial period	*p*	: Private hosting

Table 1: Characteristics of the evaluated PaaS providers

In the remainder of this section, the most important criteria that were used to select the supported vendors are going to be introduced, before Section 2.2 finally presents the chosen vendors.

Management Interface Criteria

An evaluation of the documentation of the listed PaaS providers revealed three types of distinguishable management interfaces. Most providers offer a web interface that allows to adjust numerous settings of the platform. The web interface is usually supported by a web

[6] *App42 PaaS*. URL: `http://app42paas.shephertz.com` (Retrieved: March 22, 2016).
[7] *AWS Elastic Beanstalk*. URL: `http://aws.amazon.com/elasticbeanstalk` (Retrieved: March 23, 2016).
[8] *IBM Bluemix*. URL: `http://www.ibm.com/cloud-computing/bluemix/` (Retrieved: March 23, 2016).
[9] *cloudControl*. URL: `https://www.cloudcontrol.com` (Retrieved: January 20, 2016).
[10] *CloudFoundry*. URL: `http://cloudfoundry.org` (Retrieved: March 23, 2016).
[11] *dotcloud*. URL: `https://www.dotcloud.com` (Retrieved: October 31, 2015).
[12] *Exoscale*. URL: `https://www.exoscale.ch` (Retrieved: October 31, 2015).
[13] *Heroku*. URL: `https://www.heroku.com/` (Retrieved: March 23, 2016).
[14] *HP Helion Development Platform*. URL: `http://www8.hp.com/us/en/cloud/helion-devplatform-overview.html` (Retrieved: May 6, 2015).
[15] *Jelastic*. URL: `http://jelastic.com` (Retrieved: October 29, 2014).
[16] *Microsoft Azure*. URL: `http://azure.microsoft.com/en-us/overview/what-is-azure/` (Retrieved: October 31, 2014).
[17] *Nodejitsu*. URL: `https://www.nodejitsu.com/` (Retrieved: October 31, 2014).
[18] *OpenShift*. URL: `https://www.openshift.com` (Retrieved: October 29, 2014).
[19] *Stackato 3.4*. URL: `https://www.activestate.com/stackato` (Retrieved: October 31, 2014).

based API and operating system specific command-line tools. Some vendors even provide Integrated Development Environment (IDE) plugins or language wrappers to manage their PaaS environment. In contrast to web interfaces, APIs and CLIs are far easier to automate. Moreover, most of the times, CLIs, language wrappers and IDE plugins all use the platform's API themselves. Consequently, all other interfaces are neglected if possible and the focus is set on using the platform's API in this prototype.

Deployment Type Criteria

One of the main challenges when creating Nucleus is likely to be the deployment task. As outlined in Table 1, three distinct deployment methods could be identified: Command-line tools, deployment via Git[20] and the use of HTTP commands against the platform's API. All additional possibilities, e.g., deployment via Ant[21], Maven[22] or Gradle[23] tasks as well as IDE plugins, utilize one of the three basic approaches.

Command Line Interface

Most of the providers offer some kind of CLI to their users. During the deployment, these command-line tools upload the contents of a specific local directory or even binary files, usually application containers, into the cloud and deploy them directly onto the provider's servers. Some of the CLIs thereby mainly use the methods provided by the platform's API, others also invoke locally installed version control systems, for instance Git.

Offering support for providers that utilize CLI deployment might cause dependencies onto the command-line tools that are offered by the provider and hence also lead to operating system specific restrictions. Nevertheless, the invocation of these tools could be realized in plenty of programming languages, for instance Java[24] or Ruby[25]. Besides the providers that offer deployment via their CLI tools listed in the table, also AppFog V1[26] and Google AppEngine[27] use this method.

Git

The second identified deployment method relies on the use of Git repositories and locally installed Git binaries. An execution of the `git push` command uploads the files to the remote repository and triggers deployment hooks that launch the build process. This deployment method introduces a direct dependency onto a local Git installation, but there are plenty of approaches available to invoke these commands from within an application, for instance if

[20] *Git*. URL: `http://git-scm.com/` (Retrieved: October 29, 2014).
[21] *Apache Ant*. URL: `http://ant.apache.org` (Retrieved: October 29, 2014).
[22] *Apache Maven*. URL: `http://maven.apache.org` (Retrieved: October 29, 2014).
[23] *Gradle*. URL: `http://www.gradle.org` (Retrieved: October 29, 2014).
[24] *Executing operating system commands from java*. URL: `https://blog.art-of-coding.eu/executing-operating-system-commands-from-java` (Retrieved: May 6, 2015).
[25] *Calling shell commands from Ruby*. URL: `http://stackoverflow.com/a/2400/1009436` (Retrieved: May 6, 2015).
[26] *AppFog*. URL: `http://appfog.com` (Retrieved: October 31, 2014).
[27] *Google AppEngine*. URL: `https://appengine.google.com` (Retrieved: October 29, 2014).

using Ruby[28] or Java[29]. Some other platforms that allow the usage of Git commits as upload mechanism are Amazon's Elastic Beanstalk[30] and Google AppEngine.

HTTP

Even though all platforms offer HTTP management APIs, nearly none of them actively supports the application data upload and deployment via this API. HTTP deployment is favored to be used in the abstraction layer as the deployment via HTTP commands is the most independent approach that neither requires version control dependencies nor any command-line tools to be installed.

HTTP-based deployment is used by Cloud Foundry and Heroku, which partially supports the deploy operation via the RESTful API in addition to a deployment via Git. The files to be deployed via Heroku's API must already be packaged into a so called *slug*[31].

Runtime Criteria

The set of supported runtimes is one of the main criteria for customers when choosing which PaaS to use. Table 1 not only lists the supported runtimes of each provider, but also highlights whether a PaaS is extensible to support additional runtimes. The most popular approaches to allow additional runtimes are Heroku's buildpacks[32] and OpenShift's cartridges[33]. Both approaches have already been adopted by other vendors.

An evaluation of the natively supported runtime languages[34] lists Java, PHP, Ruby, Node.js, and Python as best supported languages, with all of them being offered by more than 50 % of the providers. Based on this criteria, it was decided that the vendors that should be used in the prototype must support one common language, either natively or by extension, to enable a comparable application deployment during the development of Nucleus.

2.2 Selected Vendors

Evaluating the presented facts, the following four vendors were chosen to be included in the Nucleus prototype: *cloudControl*, *Cloud Foundry*, *Heroku*, and *OpenShift*.

More details about the vendors have been collected in Table 2. The column *complete* reveals if all API methods are fully documented. The documentation of a vendor's API is *up-to-date* if it also includes the latest development state. A subjective *rating* column indicates the

[28] *Ruby Git lib: ruby-git*. URL: `https://github.com/schacon/ruby-git` (Retrieved: May 6, 2015).
[29] *Java Git lib: jgit*. URL: `http://eclipse.org/jgit` (Retrieved: May 6, 2015).
[30] *AWS Elastic Beanstalk*. URL: `http://aws.amazon.com/elasticbeanstalk` (Retrieved: March 23, 2016).
[31] *Creating Slugs from Scratch*. URL: `https://devcenter.heroku.com/articles/platform-api-deploying-slugs` (Retrieved: March 23, 2016).
[32] *Heroku Buildpacks*. URL: `https://devcenter.heroku.com/articles/buildpacks` (Retrieved: March 22, 2016).
[33] *OpenShift Origin Cartridge Developer's Guide*. URL: `https://docs.openshift.org/origin-m4/oo_cartridge_developers_guide.html` (Retrieved: March 22, 2016).
[34] *PaaS Profiles - Runtime Language Statistics*. URL: `http://www.paasify.it/statistics/languages` (Retrieved: May 6, 2015).

overall impression of the API and its documentation. In the following, the chosen vendors, their platform, and the most important aspects to be regarded during the implementation phase are introduced.

	API Authentication		API Documentation			
Provider	**Method**	**Obtain Token**	**Rating**	**Complete**	**Up-to-date**	**Examples**
cloudControl	Token	✓	− −	✗	✗	✗
Cloud Foundry V2	OAuth 2	✓	+	✓	✓	✓
Heroku	OAuth 2	✓	+ +	✓	✓	✓
OpenShift V2	HTTP Basic	✗	+	✓	✗	✓

Table 2: PaaS vendors to be supported by Nucleus

cloudControl

cloudControl[35] is the product of a German based start-up that also distributes its PaaS as white-label product. It was not only chosen as contrast to the big players, but also because of its usage of Heroku's buildpacks. Some other companies, namely Exoscale, Cloud&Heat as well as dotCloud, internally run the same PaaS. The downside of using cloudControl is likely to be the rather poor documentation of the API, being neither complete, nor up-to-date. Examples are not available, but the important options can be extracted from one of their language specific API wrappers. The API authentication uses a custom token based system.

Cloud Foundry

Cloud Foundry was chosen for a variety of reasons. First, the open source platform serves as foundation for many other PaaS providers, for instance AppFog, HP Helion, IBM Bluemix, Pivotal WS, Stackato, and many more. Second, Cloud Foundry provides a very complex, but also very well documented API. Nevertheless, it takes some time to distinguish between the operations of API version 2 and 3 in the API documentation. Third, the simple installation within a local virtual machine allows the project setup without inflicting costs and network delays. Cloud Foundry is the only selected platform that does not support the deployment via Git, instead HTTP commands must be used to upload the data. For authentication against the API, solely OAuth 2[36] can be used. In our prototype, only the operations of API version 2 shall be applied, as version 3 is still marked as experimental.

Heroku

Heroku is one of the most popular platforms for ISVs and is often mentioned for its well designed API. The API documentation is complete, up-to-date and provides extensive examples. Authentication can be achieved using OAuth 2. Besides these points, Heroku was chosen as it is the source of the buildpack technology, offers multiple deployment regions and matches all of the defined criteria.

[35] cloudControl was shutdown due to bankruptcy end of February 2016. See `https://twitter.com/cloudcontrolled/status/699530071481196544`

[36] *OAuth 2.0*. URL: `http://oauth.net/2/` (Retrieved: May 7, 2015).

OpenShift

Apart from Cloud Foundry, OpenShift is another popular open source PaaS. In Nucleus, OpenShift V2 shall be used, but V3 is expected to be released anytime soon. Most notably, OpenShift provides its own extensibility mechanism, can be installed locally, and supports automatic scaling. It also allows the application deployment to one of several deployment regions around the globe. Opposed to the other vendors, OpenShift does not use token based authentication, but requires the credentials to be submitted via HTTP Basic-Authentication. The documentation is rather complete and provides good examples, but was not updated for quite some time. Hence, the self-describing API lists several new operations that are not included in the documentation.

2.3 Vendor API Evaluation

Having agreed on the four vendors, it must also be decided which of the available deployment and management operations shall be supported in the prototype. Management features include the application lifecycle operations to create, delete, deploy, start, and stop applications. Furthermore, they usually also offer application scaling, log management, access to environment variables, and many more [KW14].

Most vendors provide their own, nonstandardized API. Consequently, there are substantial differences in terms of the supported operations and also some distinctions in the overall functionality offered to the users. cloudControl, for instance, is the only of the four vendors that supports separated deployment environments. For the reason of this API diversity, all operations that the abstraction layer shall support have to be specified, including descriptions of their purpose and if these conditions can even be realized by the platform.

Some earlier publications [DBCAZ12; SYMT13; CNS14] already defined requirements and operations that are shared between PaaS providers and should be supported by homogenized PaaS APIs. All of them defined application lifecycle operations for starting and stopping application instances as well as Create Read Update Delete (CRUD) methods for the application environment. Cunha et al. [CNS14] also included functionalities to add services, scale the application, and access the application's log files. Sellami et al. [SYMT13] focused especially on application environments and D'Andria et al. [DBCAZ12] specialized on the migration of applications between platforms. In addition to these operations, the application's domain and environment variable objects are further essential parts of PaaS offerings that are not yet covered. We also find it reasonable to provide additional methods for application scaling, log file access, and service management.

Following a detailed evaluation of cloudControl's[37], Cloud Foundry's[38], Heroku's[39] and OpenShift's[40] API documentations, Table 3 was created to list all common API resource objects and their operations.

[37] *cloudControl API doc.* URL: `https://api.cloudcontrol.com/doc/` (Retrieved: May 7, 2015).
[38] *Cloud Foundry API doc, v206.* URL: `http://apidocs.cloudfoundry.org/` (Retrieved: May 7, 2015).
[39] *Heroku Platform API Reference, state of April, 20th 2015.* URL: `https://devcenter.heroku.com/articles/platform-api-reference` (Retrieved: May 7, 2015).
[40] *OpenShift Online REST API Guide, Ed. 1.0.* URL: `https://access.redhat.com/documentation/en-US/OpenShift/2.0/html/REST_API_Guide/index.html` (Retrieved: May 7, 2015).

Belongs to	Group	Functionality	Description	Cloud Foundry v2	Heroku	cloudControl	OpenShift v2
Application operations	App	GET	Get an application entity	✓	✓	✓	✓
		DELETE	Delete the application	✓	✓	✓	✓
		UPDATE	Update the application	✓	✓	✗	✗
	Lifecycle	REBUILD	Rebuild, e.g., to use updated buildpacks	✓	✓	✓	✓
		UPLOAD	Upload the actual application data	✓	✓	✓	✓
		DOWNLOAD	Download the current application data	✓	✓	✓	✓
		START	Start the application	✓	✓	✗	✓
		STOP	Stop the application	✓	✓	✗	✓
		RESTART	Restart the application	✓	✓	✗	✓
	Scaling	AUTOSCALE	Enable / disable auto-scaling	✗	✗	✗	✓
		ADD INSTANCE	Add new instance, scale horizontally	✓	✓	✓	✓
		REMOVE INSTANCE	Remove instance, scale horizontally	✓	✓	✓	✓
		SCALE	Set instance power level, e.g. RAM	✓	✓	✓	✗
	Domains	CHECK NAME	Is the domain name available	✗	✗	✗	✗
		LIST DOMAINS	List all application domains	✓	✓	✓	✓
		GET	Get domain entity	✓	✓	✓	✓
		ADD DOMAIN	Assign domain to the application	✓	✓	✓	✓
		DELETE	Delete and remove the domain	✓	✓	✓	✓
		UPDATE	Update the domain settings	✓	✓	✓	✓
		SET CERTIFICATE	Apply the domain's SSL certificate	✗	✗	✗	✓
		REMOVE CERTIFICATE	Remove the SSL certificate	✗	✗	✗	✓
	Variables	LIST VARS	List all environment 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 application's log files	✓	✓	✓	✓
		GET SPECIFIC LOG	Get a specific log file	✓	✓	✓	✓
		DOWNLOAD LOGS	Download all logs as an archive	✓	✓	✓	✓
		GET STATISTICS	Get some application statistics	✓	✓	✗	✗
	Services	ADD SERVICE	Install and bind to the application	✓	✓	✓	✓
		UPDATE SERVICE	Update bound service settings	✓	✓	✓	✓
		REMOVE SERVICE	Remove bound service	✓	✓	✓	✓
		GET	Get bound service entity	✓	✓	✓	✓
		LIST	List all installed services	✓	✓	✓	✓
General	App	CHECK NAME	Is the application name available for use	✗	✗	✗	✓
		CREATE	Create the application	✓	✓	✓	✓
		LIST	List all applications	✓	✓	✓	✓
	Service	GET	Get available service entity	✓	✓	✓	✓
		LIST	List all available services	✓	✓	✓	✓
		GET PLAN	Get service plan entity	✓	✓	✓	✓
		LIST PLANS	List all available plans for a services	✓	✓	✓	✓

LEGEND

Values ✓ : Functionality supported ✗ : Unsupported

Table 3: Common PaaS API operations

General operations do not relate to a specific application instance and can be divided into two subgroups, being either related to the platform's application or service object. Services and their associated plans should be enumerable and retrievable, whereas the general operations for the application object include the creation and enumeration of application environments.

Operations that belong to a specific application entity make up the largest part of the proposed PaaS abstraction layer. CRUD operations are included for application, service, environment variable and domain objects. The operations of the domain group also include methods to modify domain bound Secure Sockets Layer (SSL) certificates, which are currently only supported by OpenShift's API. Besides horizontal and vertical scaling, automatic scaling of applications is also only supported by OpenShift. The logging functionality in-

cludes methods for enumerating and retrieving log files as well as the ability to download log files and the retrieval of application statistics. Statistics are currently not supported by cloudControl and OpenShift.

In contrast to the referenced publications [DBCAZ12; SYMT13; CNS14], environments, monitoring, and database actions were intentionally left out. Database actions are rather specific and more of a technical requirement based on the utilized data back end. More advanced monitoring is out of scope as it is not planned to evaluate the platforms' performance. Application environments are neglected as they are not widely supported at the moment. It is not planned to add artificial features for platforms, but to homogenize the currently offered capabilities.

Semantic conflicts between the different vendor interfaces [LKT11] are challenges for the abstraction layer that can be identified with the help of the compatibility results of Table 3. One semantic conflict is the special application lifecycle of cloudControl, where applications cannot be stopped or put into maintenance mode. A further semantic conflict lies in the handling of the application object. Cloud Foundry and Heroku allow to update the application settings after its creation, but cloudControl and OpenShift do not offer this option. Another constraint is that it cannot be verified if a unique name constraint is violated, e.g., for application and domain names, except for application names on OpenShift.

2.4 Initial Layout

The initial layout of the architecture for Nucleus is outlined in Figure 1. To all potential users, Nucleus shall appear as one language independent API, abstracting all logic hidden underneath.

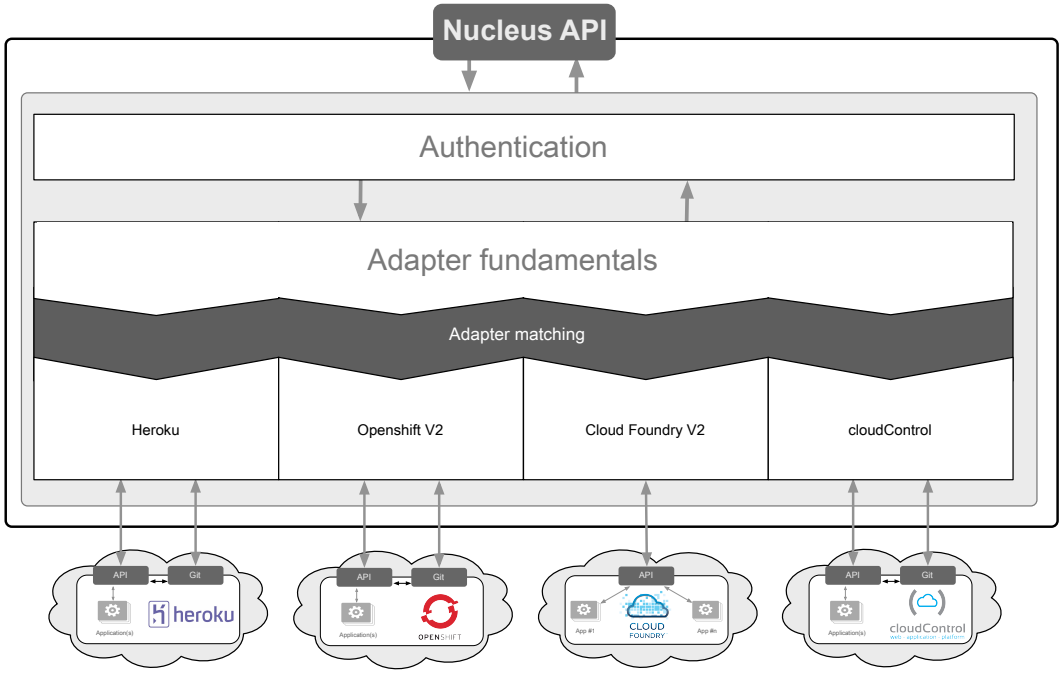


Figure 1: Initial layout of the proposed PaaS abstraction layer

Internally, it is planned to divide Nucleus into three more layers. A top-level authentication layer shall verify the presence and validity of the user's credentials. The results of the

provider evaluation revealed that it is best to request only a combination of username and password. Further authentication modes, for instance token based authentication, should be supported with these credentials as well. Below the authentication layer, it is intended to provide a set of fundamental functionalities that can be reused by all adapter modules. Shared operations are, e.g., archive file handling or Git repository actions. Each adapter is the implementation of the yet to be specified API of the abstraction layer for a specific PaaS system. The maintainability and extensibility of the prototype shall be fostered by the use of individual adapters per vendor. Below the fundamentals layer, the matching layer guarantees to automatically resolve a valid adapter for the user's PaaS. Finally, the adapters layer makes sure that the behavior of each system is equal. Adapters translate the generic operations of the Nucleus API into a sequence of native operations of the proprietary platform APIs.

3 Design

In this chapter, the detailed design of Nucleus is defined. The basic approach, which was outlined in Section 2, gets further refined so that an implementation-ready API specification is available by the end of this chapter.

One of the most important requirements for the API is to provide a programming language independent abstraction layer. In alignment with all the chosen vendors, we decided to create a RESTful API with the use of Hypertext Transfer Protocol (HTTP).

Besides the abstraction part of the API which was already introduced in the previous chapter, the need to create a public API part that provides access to the various PaaS systems emerged. In this context, from now on vendors and their platform are defined as follows:

Vendor A vendor develops and offers his PaaS, which determines the offered features to the most extend. For each supported vendor there must be an adapter that matches its unique requirements. The vendor is usually referred to by naming its platform. As an example, RedHat develops and offers the platform OpenShift.

Provider A provider uses a platform and offers it to customers, but must not necessarily have developed the platform. In this context, IBM Bluemix is an example for a provider.

Endpoint An endpoint is the API access point defined by the provider. One provider can offer multiple endpoints.

Although this distinction appears overspecified at first, it is needed to comply with all vendors and the providers in an accurate manner. Whereas most providers offer exactly one API endpoint and would not require this distinction, some providers offer multiple endpoints. As an illustration, one can take the provider IBM Bluemix which is based on the Cloud Foundry PaaS. IBM Bluemix offers two API endpoints to its customers, one referring to the Cloud Foundry service running in the United States, the other one pointing at the European counterpart. With this approach, IBM accounts for the lacking multi-region support of Cloud Foundry.

This public part of the API needs to present the vendors, providers and endpoints to the user in a way that he can analyze the available entities and navigate through the API. We decided that the entity data presented to the user shall originate from a data store. Initial data shall be loaded from adapter configuration files and populate the empty database during the startup phase. Compared to an approach that only presents static adapter configurations, this technique allows to add, update or remove providers and endpoints at runtime. This feature is especially useful when offering the abstraction layer as a hosted service or if private clouds, for instance a local Cloud Foundry deployment are used.

Figure 2 illustrates the associations between vendors, providers and endpoints in detail. For each vendor there can be an arbitrary number of providers using the platform and a provider itself can also offer any number of endpoints. Both, endpoint and provider are strictly associated with a parent provider or vendor, respectively. All three objects inherit from a common `AbstractModel` to share attributes, for instance a name, unique ID, and timestamps. The `Endpoint` object includes additional attributes, e.g., the URL at which

the API can be accessed. By using the `AdapterIndexEntry` class which associates the ID of an endpoint to a concrete adapter class, the need to traverse the complete chain upon API requests only to resolve the applicable adapter can be avoided. More detailed information on the design of this public API part can be found in sections 3.2 and 3.3.1.

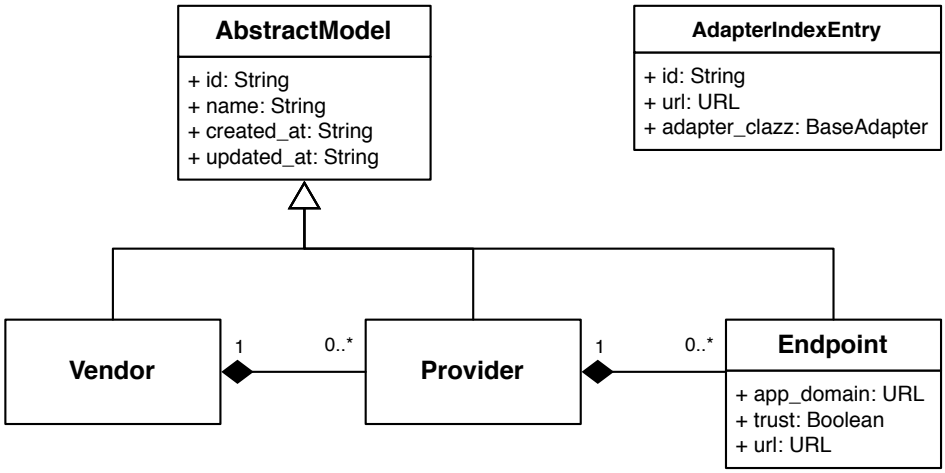


Figure 2: Class diagram showing the associated Vendor, Provider and Endpoint objects

In the remainder of this chapter, each of the following sections focuses on important subparts of the API. Section 3.1 introduces all API objects and the common application lifecycle model. In Section 3.2 the previous definitions are summarized to illustrate the big picture of the API. Thereafter, Section 3.3 describes all API operations with their pre- and post-conditions. Finally, the actual API abstractions are presented in Section 3.4. The first part of the abstractions is shown in Section 3.4.1 which describes the mapping of the vendor's API objects to Nucleus' objects. Thereafter, Section 3.4.2 presents which operations need to be called on the platforms to create the desired behavior, completing the API's definitions and the design chapter.

3.1 API Objects

Based on the identified operations of Section 2.3, this section introduces all API objects that are needed to build the foundation of the prototype. All identified objects, including their associations and relationships, are visualized in the class diagram that is shown in Figure 3. The included `PersistedEntity` and its inheriting classes follow the previous definitions to build the public API part and manage `Vendor`, `Provider`, and `Endpoint` objects.

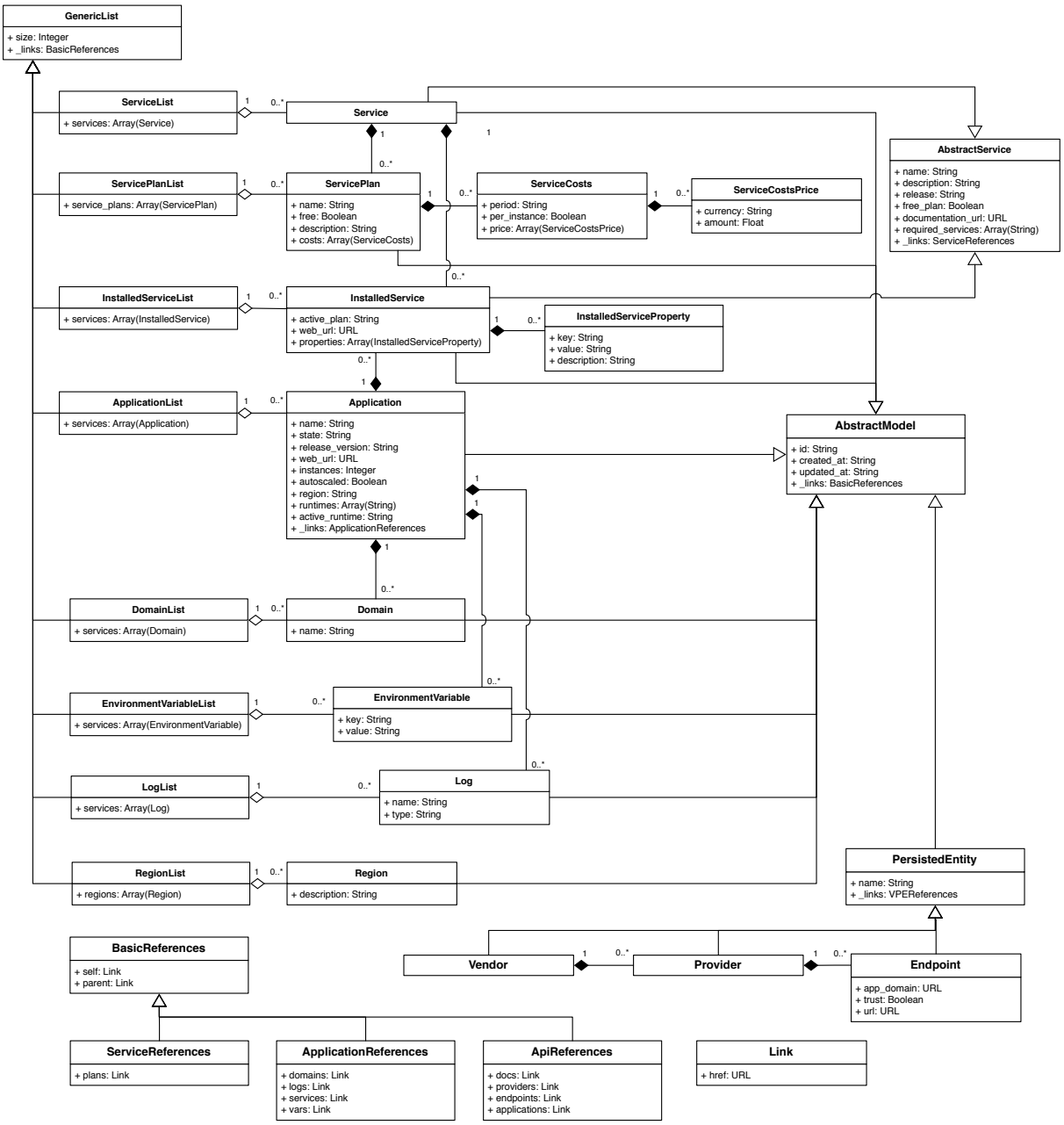


Figure 3: API objects class diagram

Following various best practices that describe how to properly build an API[41,42,43,44], the abstraction layer's API utilizes several common concepts. If applicable, all of the API's response objects shall have a unique ID and timestamps that reveal when the object was created and when it was updated for the last time. Combined with the `_links` property that shall be utilized to match the ideas behind the Hypermedia as the Engine of Application State (HATEOAS) principle of RESTful applications, those requirements lead to the

[41] *Best Practices for Designing a Pragmatic RESTful API.* URL: `http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api` (Retrieved: June 3, 2015).

[42] *HTTP API design guide extracted from work on the Heroku Platform API.* URL: `https://github.com/interagent/http-api-design` (Retrieved: November 26, 2014).

[43] *HTTP API Design.* URL: `https://github.com/interagent/http-api-design` (Retrieved: June 25, 2015).

[44] *RESTful Service Best Practices - Recommendations for Creating Web Services.* URL: `https://github.com/tfredrich/RestApiTutorial.com/raw/master/media/RESTful%20Best%20Practices-v1_2.pdf` (Retrieved: June 25, 2015).

`AbstractModel` class which possesses all those four properties and must be inherited by all other API response objects. References are required to be of the type `BasicReferences` or one of its inheriting classes. The class `BasicReference` has two links, one `self` and one `parent` link. Its self-reference is a URL that tells where this specific object can be retrieved from. The parental reference reveals to which other object the object is assigned. Both links must always be set, except for the API's root node, which does not have a parental reference. All subtypes of the `BasicReference` provide additional relations, e.g., in case of the class `ApplicationReferences` to all child object collections of an application. Further API concepts that are not directly related to the API's objects are presented in 3.2 and its subsections.

In addition to the `Application` object as core component of all evaluated APIs, Figure 3 also includes the `Region`, `Service`, `ServicePlan`, `Domain`, `EnvironmentVariable`, `Log` and `InstalledService` classes. All these objects are going to be introduced in the following subsections. Beside those, the diagram also contains dedicated `List` objects for all of the previously mentioned classes. Their purpose is to normalize the way how the API responds with object collections. Therefore, all `List` objects inherit from `GenericList` including its `size` property to indicate the number of available elements and the `_links` property for HATEOAS. Additional features, for instance pagination within the API, can easily be added to the generic class later on.

3.1.1 Region

Regions refer to a specific geographic deployment location that are offered by some providers. A provider decides which regions are offered to its customers. The initial region object is defined solely by a `description` attribute, apart from the basic ID and timestamps. Restrictions apply on some platforms, e.g., OpenShift has a dedicated location for Node.js deployments, but is missing formalizations that could be included in the object. Therefore, it was decided to use the `description` attribute for those comments and leave the validity assertions to the platform itself for now.

3.1.2 Service

Services are additions to the actual program, often dependencies, that can be installed and bound to an application. Popular examples for services are data stores, monitoring tools, logging utilities, notification services, and many more. The specification of the offered services varies between all platforms. Each `Service` object describes a service that can be installed and bound to an application. Despite the feature of some platforms to install global services that can be used with multiple applications, Nucleus' initial service object is explicitly restricted to services which are part of exactly one application. Services that are already bound to an application are described later on in Section 3.1.3 with a dedicated object.

Attributes of a service, which are shared amongst all evaluated platforms, are defined inside the `AbstractService` class, from which the `Service` class inherits. A service features common properties, e.g., the `name` and `description`, but also service specific properties like the `release`. The `release` property holds information about the software version of the service. A service's `documentation_url` refers to a web page containing more information about the

service and how it can be used. If a service can be used without any charges, the `free_plan` property must be set to TRUE. In case a service requires additional services before it can be installed, the IDs of those required services shall be shown in the `required_services` property.

In order to install and bind a service to an application, most platforms support the concept of service plans.

Service Plans

Service plans always belong to exactly one service. They describe the conditions under which the service can be used and the price that will be charged. The class diagram in Figure 3 shows the extensive model with the `ServiceCosts` and `ServiceCostsPrice` classes that are needed to achieve compatibility with all four platforms. In all response messages, the `ServiceCosts` and `ServiceCostsPrice` objects shall be embedded in the `ServicePlan` object. They should not be retrievable directly via the API. `ServiceCosts` represent costs which may be charged for the service installation or usage. The `period` property is used due to the fact that costs can either be fixed, e.g., to be paid on a monthly basis, or dependent on the number of times the service is used. Fixed periods that were encountered during the evaluation are HOURLY and MONTHLY. Some services also charge a cost per application instance, which is denoted by the Boolean `per_instance` property. The `ServiceCostsPrice` class belongs to exactly one cost object and describes the `amount` of money to be charged in the defined `currency`. Platforms can specify the charges in more than one currency. A `ServiceCosts` object must have at least one price. Costs are usually charged in the user's currency.

3.1.3 Application

The `Application` object can be seen as the core object for most parts of the abstraction layer. It has four direct child object types which can be associated with an application, namely `Domains`, `EnvironmentVariables`, `Logs` and `InstalledServices`. Standard properties are an application's `name`, the references as well as the ID and timestamps. Each application can be instantiated with one or more `runtimes`, whereby limitations of the used platform apply. The runtime that is currently used by the application shall be visible in the `active_runtime` property. If an application has been deployed, the `release_version` property indicates the version of these binary files. Within the `web_url` property, one can find the URL at which the application should be accessible by default. The geographic location of the servers at which the application is or shall be deployed is denoted by the application's `region`. If the `autoscaled` property is set to TRUE and the platform supports autoscaling, the number of application instances will be automatically adjusted to the current or expected load. The `instances` property shows the number of application instances that are created for the application.

All of the application's child objects as well as the lifecycle of a deployed application are presented in the following paragraphs.

Application Lifecycle

An application usually undergoes several stages which is manifested in the `state` property of the application object. Managing an application and its states requires a detailed knowledge of the state transitions and the overall lifecycle of the application. Each of the four platforms is based on a slightly different understanding of the application object. Nonetheless, we managed to identify a generic PaaS application lifecycle that complies with all four platforms. The lifecycle, which is illustrated in Figure 4, differentiates between a total of six states. Platforms are allowed to use only a subset of these six states, especially as not every platform supports all six states.

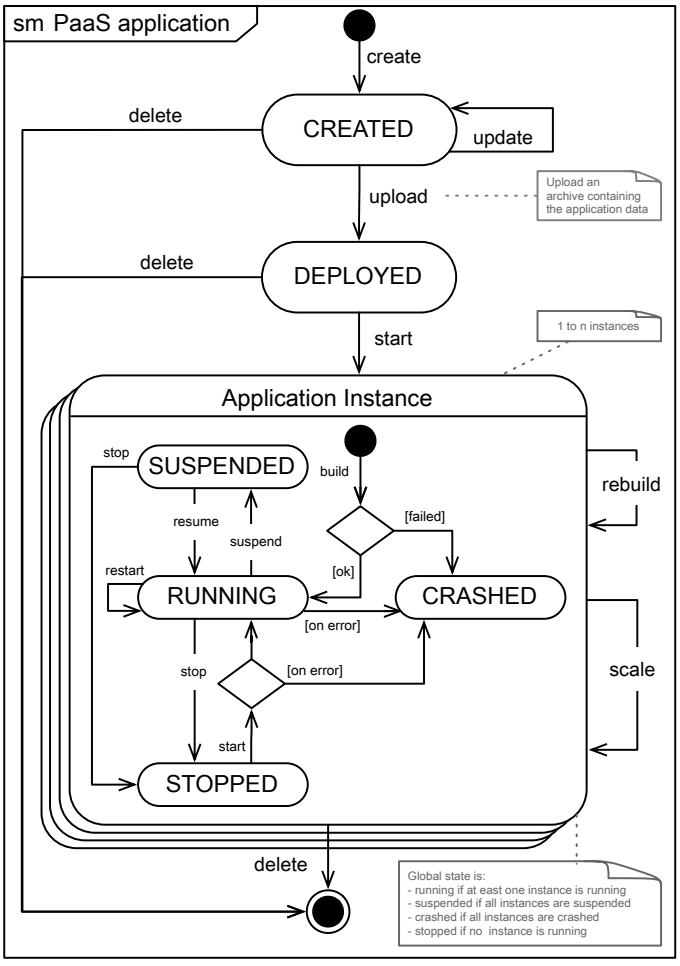


Figure 4: Generic PaaS application lifecycle

Initially, an application must be created, upon which its state shall be CREATED. In order to transition into the DEPLOYED state, the applications data must be uploaded to the platform. Even though it can be argued that the DEPLOYED state is not required and applications could directly switch into the RUNNING state, this state was retained. It provides additional flexibility, for instance persisted data can be imported or background jobs can be started before the actual application is made available to the public. From the DEPLOYED state, the instances of the application can be started. The application's build process can be executed at two positions inside the lifecycle. Cloud Foundry and cloudControl both expect an explicit build command to be called. In the abstraction layer this build is triggered before the start of the first application instance. A failed build would then cause a state transition of the application instance into the CRASHED state. In contrast, Heroku and OpenShift

start the build process within the Git deployment process. Failed builds do not cause the application instance to transition into the CRASHED state, but rather reject the data upload and retain the application's previous global state. If at least one instance of the application was successfully started, its state changes from DEPLOYED to RUNNING. After a period of inactivity, cloudControl, Heroku and OpenShift can put an application to sleep, changing the application instance's state from RUNNING to SUSPENDED. If the application instance is resumed, e.g., when new requests arrive, the state changes back to RUNNING. With the `stop` command an application can be shutdown, whereupon the new state of the application and all instances is STOPPED. The `start` command on a stopped application usually results in the state to become RUNNING, but in case of errors the application instance state can also switch to CRASHED. Apart from changes of the application's runtimes, application object modifications, triggered rebuilds and scaling do not have a direct effect on the application's state. The global state of the application can be deducted by the individual application instance states. The application is said to be running if at least one instance is RUNNING. STOPPED, CRASHED or SUSPENDED states only apply if all instances are in this state.

Domains

Domains are always bound to an application and cannot exist without a parent application. In addition to an application's default `web_url`, platforms offer to register additional domains at which an application can be made available. For the realization of the abstraction layer it is sufficient that the domain object has only one distinct property, the `name`, which must be set to the desired Fully Qualified Domain Name (FQDN). Support for more advanced features, e.g., the assignment of SSL certificates to use HTTPS connections, is not yet offered by the APIs of the evaluated platforms.

Environment Variables

Environment variables, which are often also only referred to as variables, represent key-value pairs that are available to an application at different stages of their lifecycle. On local systems, e.g., UNIX systems, those key-value pairs are loaded into the system's PATH via the .bashrc file. The variables often serve as configuration parameters, for instance credentials or destination addresses to third party systems and services. The `EnvironmentVariable` class of the abstraction layer's API has two specific properties, `key` and `value`. Applications can retrieve the `value` from the system's PATH via the `key` that must be unique per application.

Installed Services

In Section 3.1.2 the `Service` class, representing common middleware functionalities that can be added to an application, was introduced. This paragraph focuses on the `Installed-Service` class which describes a service that was already installed and bound to a parent application. Installed services share all properties of the general service, but are enriched with additional information. The optional `web_url` property of the installed service points to the web interface of the service. The identifier of the currently used service plan shall be shown as the `active_plan`. As some services also require or provide their own variables, they are made available as embedded collection named `properties`. The variables of the service

shall not be visible as `EnvironmentVariable` and only appear as properties of an installed service. Similar to the class `EnvironmentVariable` they possess `key` and `value` properties. Moreover, the `description` property contains additional information on the context of the variable.

3.2 API Structure

Progressing from the API operations and objects from the previous chapters, this section presents the structure of the Nucleus API from a global perspective. At first, the associations between objects and operations to implement all required CRUD operations are shown. Thereafter, more general API definitions are introduced, e.g., the common error schema.

Resource maps are known to be one way of visualizing RESTful APIs. A resource map of the Nucleus API is presented in Figure 5. This figure does not show all operations that are available on the objects, but tries to present the URL paths of the objects and how they can be created, resolved or updated. Object properties are also neglected, except for the associations with other API objects. As shown in the legend of the figure, API resources are depicted as rectangles. Resources with green background represent collections, whose objects can be returned as a list. Collection resources with a double border also allow that new objects can be created via `POST` requests. The content of the rectangles stands for the URL path at which the resource shall be retrievable. Curly braces need to be replaced with the ID of the object they refer to, for instance {e} needs to be replaced with the ID of the endpoint that shall be used. Resources with a white background represent a specific object instance. A double border on such an instance shows that the object can be changed via `PATCH` requests. In addition to the URL path, the lower part of the rectangles contains the references of the object. References with round borders represent an object, usually parental objects. All references with rectangular borders reference entire collection resources, usually child objects. Arrows are the connections between resources. Besides the references that link to specific resource instances, the connection from the list resource to the object resource indicates the type of the list's elements. All operations belong to one API version, as indicated by the API version node at the top of the figure. From there, the figure is separated into four dedicated API groups:

The Nucleus group is depicted in the top left of the API resource map. Below the version of the Nucleus API, the resources of all three objects are the only ones with an API top level path. All resources in this group are public and do not require authentication. The vendor collection resource can be used to show a list of all supported vendors, whereas specific vendor objects can be retrieved via the instance resource. Vendor objects cannot be created, deleted or updated at runtime as they link to the logic which communicates with the PaaS system that must be implemented by an adapter class. Similar to the vendors, providers and endpoints can also be listed or retrieved. As indicated in the illustration, they can additionally be created, updated or deleted at runtime. When creating or modifying provider or endpoint objects, the only requirement is that the names must be unique amongst the providers and endpoints of all vendors. All other resources, e.g., applications, are nested below the endpoint to which they belong.

The service group implements a dedicated group focused on providing information about the available services. Valid authentication credentials of the chosen endpoint are required. Service objects can be loaded as a collection via the list resource or as a specific instance

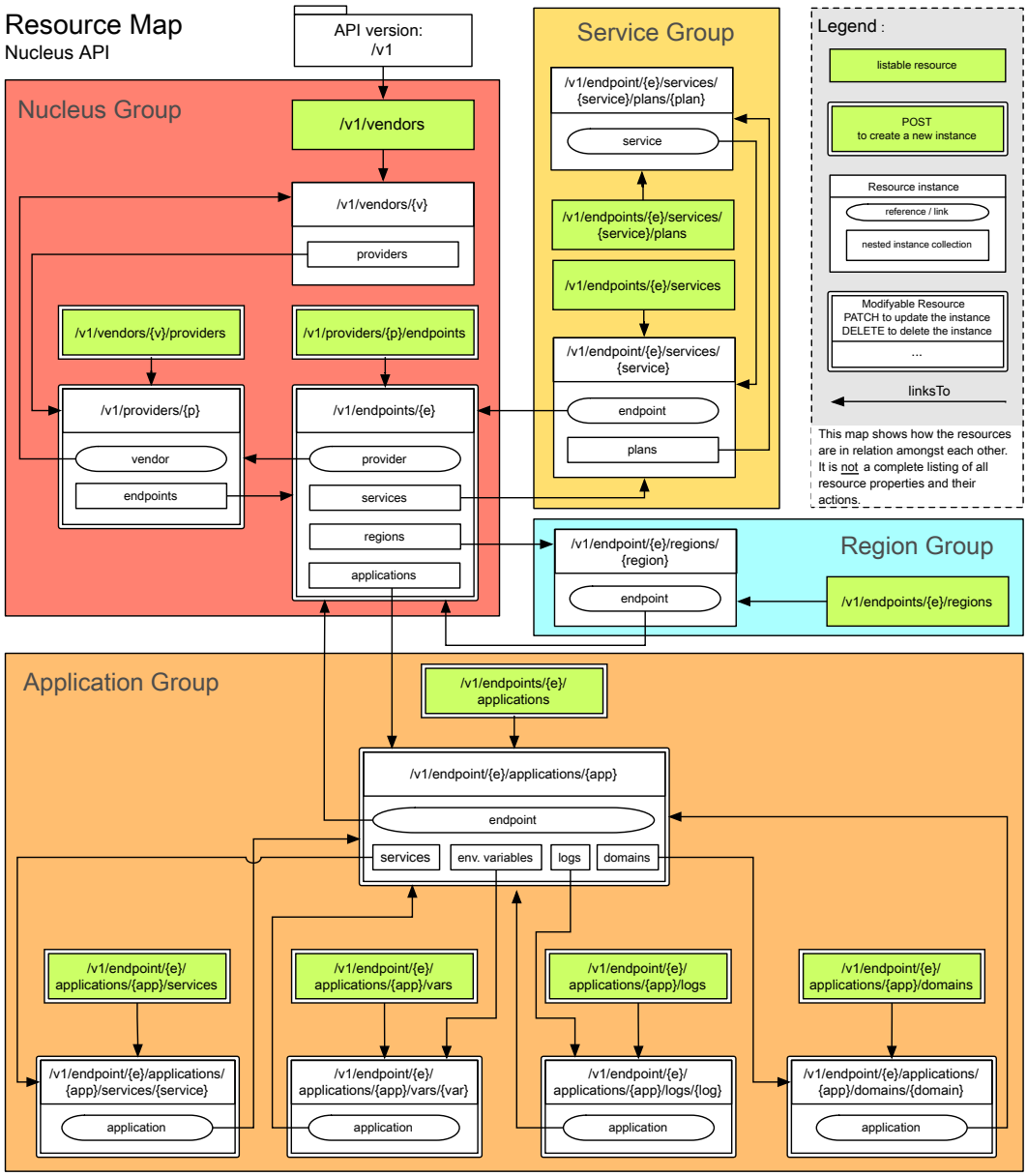


Figure 5: Nucleus API - Resource Map

via the service's ID and strictly belong to the referenced endpoint. All subordinated service plans, belonging to exactly one service, can be retrieved via the URL path that is nested below the service resource's path. As for the services, service plans can be gathered as collection or individual object. Both objects and their collections are read-only. Service objects and plans can neither be deleted, created nor updated by users. Services are maintained by the provider of the endpoint.

The retrieval of all available deployment regions is maintained by the region group. It provides two resources, the region list and specific resource instances. All region resources are read-only and belong to the currently connected endpoint. Valid authentication credentials of the chosen endpoint must be embedded in the requests against this group.

At the bottom of the figure, the application group is shown as largest of all groups. It contains the list and instance retrieval for all applications, domains, logs, variables, and installed services. The application group requires valid authentication credentials against

the chosen endpoint. The application resource is nested below the endpoint whereas the domains, logs, variables, and installed services are nested below the application to which they belong. All of the application's child objects link to their parent application and are referenced in the application object themselves.

3.2.1 Authentication

In the previous subsection the different API groups were introduced. Except for the public Nucleus group, all other groups require valid authentication credentials for any request. Credentials are a combination of a username and a password. Some providers also use the user's email address as username.

We choose to use HTTP Basic authentication for the abstraction layer's web-based API. However, this decision urges to use only HTTPS encrypted connections as the HTTP basic authentication information are transmitted in Base64 encoded cleartext and would therefore be an easy target for attackers. More secure authentication techniques, for instance HTTP digest authentication, had to be rejected as the endpoints themselves only accept credentials or API tokens, but no calculated digest values. Another approach would be to rely only on API tokens, but as of now not all providers offer this authentication method. With HTTP basic authentication the username and password are received, which can then be used for direct authentication or to obtain all needed API tokens.

In order to use HTTP Basic authentication, the credentials must be submitted via the HTTP Authorization header. The value of the key-value pair shall be the authentication method, followed by a Base64 encoded combination of the username and password which is separated by a semicolon [FHH+99; FR14a]. Listing 1 shows the Authorization header for an exemplary username MyUsername and its password MyPassword.

Listing 1: API Authentication Header example to be used with Nucleus

```
1 Authorization: Basic TXlVc2VybmFtZTpNeVBhc3N3b3Jk
```

3.2.2 Versioning and Accept Header

The Nucleus API follows the semantic versioning specification[45]. Nucleus shall allow to serve multiple versions of the API at once and provide legacy support. Each non-backward compatible change of the application, e.g., additional required parameters, must result in an increase of the major API version. If the user does not request a specific API version, the latest release version shall be used as default. To request a specific API version, the user must specify the version in the HTTP Accept header and refer to the vendor 'nucleus'. Code Listing 2 shows a valid Accept header to request API version 'v1'. In case of invalid accept headers, for instance an unknown version or vendor, the HTTP status code 406 shall be returned in accordance with the latest HTTP 1.1 standard [FR14b].

Listing 2: API Accept Header example to be used with Nucleus

```
1 Accept: application/vnd.nucleus-v1
```

[45] *Semantic Versioning 2.0.0*. URL: `http://semver.org` (Retrieved: May 12, 2015).

3.2.3 Message Formats

The response messages of the API can be categorized into three groups: Object presentations, collections of objects, and errors. The objects as well as their commonalities that form a common message format have already been introduced in Section 3.1. Discrete child objects that can also be retrieved via dedicated API resources shall generally not be included inside an object's presentation. For instance, the application's representation shall not include any information about its variables, domains or services. However, as described in Section 3.1, all objects with child objects include references pointing to these collections that can be used to navigate through the API.

Following the definitions of the API objects and object collections, all necessary information on API errors are described in the next paragraphs.

Error Schema

All API errors that are passed to the user shall follow a common schema. The defined error schema contains five parts that present the relevant error information, brief resolution guides and optionally also a link to a more comprehensive API documentation on a web page.

status The HTTP status code of the error, for instance 404 if an object or a resource could not be found. Compare to the next paragraph for a list of all allowed error status codes.

error_code A unique error code to identify the error. The code consists of two parts, starting with three numbers that are equal to the status code, followed by three more numbers to identify the specific error.

message A basic and easily understandable description of the error. Can be shown to the user.

dev_message The developer message, containing more detailed explanations why the error might have occurred. In case of failures related to the user's requests, there are also first hints how to fix the request.

more_info Link to an online documentation at which one can explain the insights of an error and present detailed resolution approaches.

Error Status Codes

The two main groups of HTTP error codes are either user related errors or requests that failed on the server side. User related errors range from 400 to 499, whereas server related errors are indicated by error numbers between 500 and 599. Within Nucleus' API definitions, user errors are mostly thrown in case of bad request parameters, but in case of error 422 also if an action cannot be invoked as preconditions are not fulfilled.

400 - Bad Request The request is invalid. This error usually occurs on write requests, if not all required parameters are specified or contain invalid values.

401 - Unauthorized A requests is unauthorized if no authentication credentials are provided or if they are rejected by the endpoint. In accordance with RFC7235 [FR14a].

404 - Not Found The resource or the object cannot be found. This error occurs for unknown URLs, e.g., due to spelling errors or if there is no object instance with the given ID.

406 - Not Acceptable This error indicates an invalid accept header. Either the API vendor or the API version cannot be found.

422 - Unprocessable Entity The error "means the server understands the content type of the request entity [...], and the syntax of the request entity is correct (thus a 400 (Bad Request) status code is inappropriate) but was unable to process the contained instructions" [Dus07, p. 77]. This status is returned if any conditions are violated, e.g., an application cannot be started as there is no application data available yet.

Server errors can originate either directly from Nucleus or may be forwarded from the active endpoint. Especially the error codes 503 and 504 are relayed from the platform, with no possibility for the API to transparently resolve the error.

500 - Internal Server Error Internal server errors can either be caused by Nucleus or originate from the platform. The error indicates unexpected conditions or behavior and cannot be fixed by the user in most cases.

501 - Not Implemented All operations that are not (yet) implemented by an adapter shall raise this error.

503 - Service Unavailable This error is relayed by Nucleus. Usually the PaaS endpoint is available again only seconds after the failed request. Sometimes it also indicates scheduled maintenance or platform updates.

504 - Gateway Timeout This error is passed on from the platform, which raised an internal timeout error. It cannot be known to what degree the request has been processed or if it was executed at all.

Detailed information on returned status codes of valid and successful requests of each Nucleus operation can be found in Section 3.3.

3.3 API Operations

This section describes the most important API operations in detail. In general, all operations are based on four HTTP methods: `GET`, `PATCH`, `POST` and `DELETE`.

Operations using the `GET` method always retrieve certain objects or a collection of objects. They never inflict any changes on the system and shall return the same result if executed multiple times until another action changes the object. Nevertheless, responses cannot always be expected to return identical results as changes can also be triggered by internal system actions, for instance if an application transitioned into another state, e.g., SUSPENDED. Successful responses of `GET` requests are always expected to return the status code 200. The

`PATCH` method is used solely to update existing objects. Parameters shall be optional and only the provided fields will be changed inside the object. Similar to the `GET` method, the HTTP status code 200 is used for successful operations. The `POST` method does not only create new objects, but also triggers actions on existing objects, e.g., to change the application's state. In case of successful actions, the status code 200 shall be returned. However, if a new object was created, the code should be 201. Objects can be removed with the help of the `DELETE` method. If the deletion is successful, the returned status code is expected to be 204.

HTTP allows several ways to pass parameters to the server, but only a few of them are required to create the Nucleus API. One of the most commonly used approaches is the use of `query` parameters which appends key-value pairs directly to the URL. `Header` parameters are embedded in the request's headers. A third approach is the so called path-templating, whereby parameters are part of the URL's path. In the majority of cases, path-templating is used to create RESTful APIs. A `form` is usually utilized to send the user's input data to the server. The default content type for web forms is `application/x-www-form-urlencoded`. Forms must be submitted with an altering request type, for instance the `POST` method. The payload can also be added directly to the request's `body`, but this does allow only one key-value pair to be sent. However, the specification of the multipart/form-data content type also allows to provide more than one parameter within the request's body [Mas98; FR14b].

In comparison to the initially presented approach of Chapter 2, not all of the operations that were shown in Table 3 are adopted in the final API specification. The five operations mentioned below were left out, mostly for compatibility reasons:

Scaling/AUTOSCALE The operation to enable or disable autoscaling has been neglected in favor of a Boolean `autoscale` property inside the `Application` class. The functionality remains unchanged.

Domains/CHECK NAME & Application/CHECK NAME Checking if a name is already taken on a platform cannot be achieved reliably with the current API functions of the platforms.

Domains/UPDATE Updating a domain is not of use as long as the domain object has only one property, the domain name. Additionally, Heroku and cloudControl do not offer update methods themselves. The workaround to delete the existing and create a new domain that would have been used for those platforms is the same approach which can now be applied at the abstraction layer's API.

Logging/GET STATISTICS Statistics are only supported by Cloud Foundry and Heroku. There are also big differences in the format and content of the returned objects. This feature was postponed to focus on more important aspects.

In the following, the most significant operations of the specified API are introduced. Operations that are mostly identical in terms of the request's structure are not included. A complete definition of the API's operations is available in the API documentation of the released project. More information about this API documentation and how it can be accessed is described in Section 4.10.

Table 4 shows the general format that is used to present the API operations. Its first row contains the HTTP method, the group to which the operation belongs, the class of the

returned object and the expected status code of a successful response. The second row shows the URL path at which the operation can be called. Curly braces indicate variables which must be populated with dynamic values. The third row list possible operation parameters. Optional parameters are printed in italic script, whereas required parameters are in bold letters. The third column in the parameters row determines where the parameter must be made available. The object type that must be provided is shown in the fourth column. An additional description of the value is presented in the last column of the parameters row. The two remaining rows list pre- and postconditions of the API operation. If no postcondition is mentioned, the object is not supposed to be changed by this request.

HTTP method	**Operation group**	**Response object**			**Response status code**
URL path	/the/url/path/with/a/{variable}				
Description	A brief description of the operation's intentions				
Parameter(s)	**required-param**	path	String	Required path parameter	
	group/optional-param	form	String	Optional parameter in a nested group	
Precondition(s)	What must be the case that the command can be executed, e.g., a state the app must be in				
Postcondition(s)	All conditions that must be true when the operation succeeded and all triggered actions finished				

Table 4: General API operation table format

3.3.1 Vendor, Provider, and Endpoint Operations

At the beginning of Chapter 3, the idea of a public API to manage the vendor, provider, and endpoint objects was introduced. Originating from this definition, Table 5 shows selected read operations and Table 6 the most essential write operations.

All three object types can be retrieved as collection or individual instance via `GET` requests. The first entry of Table 5 shows the enumeration of all known vendors. In the second table entry, one can see the request to retrieve a specific vendor, which requires the VENDOR_ID to be set as path parameter. Entry three highlights the retrieval of a child resource, in this case all providers that are registered for a specific vendor platform. This operation requires the vendor's identifier and returns a list of all associated providers. Providers, endpoints, and child collections of theirs can be retrieved equivalently.

GET	**Vendor**	**VendorList**			**200**
URL path	/vendors				
Description	List all supported vendor platforms				

GET	**Vendor**	**Vendor**			**200**
URL path	/vendors/{vendor_id}				
Description	Get a specific vendor object				
Parameter(s)	**vendor_id**	path	String	ID of the vendor object to be retrieved	

GET	**Vendor**	**ProviderList**			**200**
URL path	/vendors/{vendor_id}/providers				
Description	List all providers associated with this vendor				
Parameter(s)	**vendor_id**	path	String	ID of the vendor object to be retrieved	

Table 5: Vendor, Provider and Endpoint object: read operations

As an illustration of the write operations concerning the vendors, providers, and endpoints, four selected operations are presented in Table 6. A `POST` request in the first entry shows how to create a new provider. In comparison, entry two shows the operation to create a new endpoint, which also allows to provide additional form parameters besides the objects

desired name. The third entry describes the update of an endpoint using the `PATCH` method whereby all form parameters are optional. Entry four concludes the exemplary operations with the deletion of an endpoint object. The delete request does not return any content as it expected to respond with status code 204. Equivalent operations, not explicitly shown here, are available to update or delete provider instances.

POST	**Vendor**	**Provider**		**201**
URL path	/vendors/{vendor_id}/providers			
Description	Create a new provider and associate with this vendor			
Parameter(s)	**vendor_id**	path	String	The vendor's ID
	provider/name	form	String	Name of the provider entity to create
Postcondition(s)	Provider created and associated with this vendor			

POST	**Provider**	**Endpoint**		**201**
URL path	/providers/{provider_id}/endpoints			
Description	Create a new endpoint and associate with this provider			
Parameter(s)	**provider_id**	path	String	The provider's ID
	endpoint/name	form	String	Name of the endpoint entity to create
	endpoint/url	form	String	Endpoint API URL
	endpoint/trust	form	Boolean	If trusted the SSL certificates won't be validated
	endpoint/app_domain	form	String	Where applications can be accessed by default
Postcondition(s)	Endpoint created and associated with this provider			

PATCH	**Endpoint**	**Endpoint**		**200**
URL path	/endpoints/{endpoint_id}			
Description	Update the endpoint object			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	endpoint/name	form	String	Name of the endpoint object
	endpoint/url	form	String	Endpoint API URL
	endpoint/trust	form	Boolean	If trusted, the SSL certificates will not be validated
	endpoint/app_domain	form	String	Where applications can be accessed by default
Postcondition(s)	Endpoint updated, provided fields replaced existing data, not specified fields remain unchanged			

DELETE	**Endpoint**	**-**		**204**
URL path	/endpoints/{endpoint_id}			
Description	Delete the provider entity			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
Postcondition(s)	Endpoint deleted			

Table 6: Vendor, Provider and Endpoint object: write operations

3.3.2 Service and Service Plan Operations

The list of additional services, e.g. data stores, which are offered to be created and bound to applications is read-only and maintained by the provider of the selected endpoint. All operations of this API group are shown in Table 7.

GET	**Service**	**ServiceList**		**200**
URL path	/endpoints/{endpoint_id}/services			
Description	List all available services of the platform			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID

GET	**Service**	**Service**		**200**
URL path	/endpoints/{endpoint_id}/services/{service_id}			
Description	Get a specific service object			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	service_id	path	String	ID of the service to retrieve

GET	ServicePlan	ServicePlanList		200
URL path	/endpoints/{endpoint_id}/services/{service_id}/plans			
Description	List all available plans of the service			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	service_id	path	String	ID of the service to retrieve

GET	ServicePlan	ServicePlan		200
URL path	/endpoints/{endpoint_id}/services/{service_id}/plans/{plan_id}			
Description	Get a specific plan object of the service			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	service_id	path	String	ID of the service to retrieve
	plan_id	path	String	ID of the service plan to retrieve

Table 7: Service and service plan operations

3.3.3 Region Operations

Similar to the service group, regions are also read-only. Table 8 contains all operations to retrieve a specific or all available deployment regions.

GET	Region	RegionList		200
URL path	/endpoints/{endpoint_id}/regions			
Description	List all available deployment regions of the platform			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID

GET	Region	Region		200
URL path	/endpoints/{endpoint_id}/regions/{region_id}			
Description	Get a specific deployment region object			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	region_id	path	String	ID of the region to retrieve

Table 8: Region operations

3.3.4 Application Object Operations

A selection of operations concerning the management of application objects is presented in Table 9. The first two entries show how to retrieve a list of all applications the user has access to, respectively a specific application. Entries three to five present the operations to create, update, and delete an application instance. In order to create an application, the name and the runtime language parameters must be provided. Optionally, the deployment region and the autoscaling option may be specified. According to the providers used in our prototype, application names must be unique amongst all applications that are registered at the endpoint.

GET	Application	ApplicationList		200
URL path	/endpoints/{endpoint_id}/applications			
Description	List all applications that are registered at the endpoint			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID

GET	Application	Application		200
URL path	/endpoints/{endpoint_id}/applications/{application_id}			
Description	Get a specific application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application

POST	Application	Application		201
URL path	/endpoints/{endpoint_id}/applications			
Description	Create a new application at the endpoint			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application/name	form	String	The application's name
	application/runtimes	form	Array(String)	Runtimes to be used for the application, e.g., 'nodejs'
	application/region	form	String	Region where to deploy the application, e.g., 'US'. Show available regions via '/regions'
	application/autoscaled	form	Boolean	Indicator if the application shall be autoscaled. Value is ignore on some platforms.
Precondition(s)	Application name is not yet used			
Postcondition(s)	Application created, state: CREATED			

PATCH	Application	Application		200
URL path	/endpoints/{endpoint_id}/applications/{application_id}			
Description	Update the application object			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
	application/name	form	String	The application's name
	application/runtimes	form	Array(String)	Runtimes to be used for the application, e.g., 'nodejs'
Postcondition(s)	Specified application fields updated with the provided values			

DELETE	Application	-		204
URL path	/endpoints/{endpoint_id}/applications/{application_id}			
Description	Delete the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
Postcondition(s)	Application deleted			

Table 9: Application object CRUD operations

Data Management

All three operations concerning the management of the application data are shown in Table 10. The first entry presents the operation to deploy application data. In comparison to other `POST` requests, this operations returns the status code 204 upon successful execution. This code is used instead of 200 or 201 to reflect potentially still ongoing deployment after the request returned, wherefore the application object may not instantly reflect the new state. The application data which should be uploaded must be included in the body of the HTTP request as compressed zip or tar.gz archive. The second entry summarizes all instructions to download previously deployed application data. The `GET` request responds with binary data in the form of the compressed application data. The desired archive format can also be specified via a query parameter. Unless the application has already been deployed, the download will fail with the error code 422. Rebuilding the application is described in the third table entry. Similar to the data download, the rebuild can only be invoked if the application has already been deployed. Except for unexpected errors due to changes in the provided runtimes, the rebuild shall not alter the application state once being completed.

POST	Application	-		204
URL path	/endpoints/{endpoint_id}/applications/{application_id}/data/deploy			
Description	Deploy application data			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
	file	body	File	Application archive (zip or tar.gz)
Postcondition(s)	Application data persisted. If state was CREATED it switches to DEPLOYED, otherwise state remains unchanged.			

GET	Application	Binary data		200
URL path	/endpoints/{endpoint_id}/applications/{application_id}/data/download			
Description	Download the deployed application data			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
	archive_format	query	String	Compression format of the downloaded archive (zip or tar.gz), default: zip
Precondition(s)	Application state must not be CREATED			

POST	Application	Application		200
URL path	/endpoints/{endpoint_id}/applications/{application_id}/data/rebuild			
Description	Rebuild the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
Precondition(s)	Application state must not be CREATED			
Postcondition(s)	Application build executed, state remains unchanged.			

Table 10: Application object data operations

Application Lifecycle Management

Managing the lifecycle of an application is supported via the three operations, start, stop, and restart, which are presented in Table 11. All three operations are nearly identical in terms of their signature and do not require any additional parameters besides the endpoint and application ID. However, all of them have different postconditions that shall apply after the operation finished. After the start or restart operation was invoked, the application shall switch to the RUNNING state. Likewise, the stop operation shall cause the application to transition into the STOPPED state. If any of the commands fails during execution, the application usually proceeds to the CRASHED state. Depending on the platform's supported states, the exact behavior may slightly vary between different providers.

POST	Application	Application		200
URL path	/endpoints/{endpoint_id}/applications/{application_id}/actions/start			
Description	Start the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
Precondition(s)	Application state must not be CREATED			
Postcondition(s)	Application start triggered. If the start is successful, state changes to RUNNING. If an error occurs the state changes to CRASHED.			

POST	Application	Application		200
URL path	/endpoints/{endpoint_id}/applications/{application_id}/actions/stop			
Description	Stop the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
Precondition(s)	Application state must not be CREATED			
Postcondition(s)	Application stop triggered, state changes to STOPPED.			

POST	Application	Application		200
URL path	/endpoints/{endpoint_id}/applications/{application_id}/actions/restart			
Description	Restart the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
Precondition(s)	Application state must not be CREATED			
Postcondition(s)	Application restart triggered. If the restart is successful, state changes to RUNNING. If an error occurs the state changes to CRASHED.			

Table 11: Application object lifecycle operations

Scaling

Evolving from the ADD INSTANCE, REMOVE INSTANCE and SCALE operations which were defined in the initial evaluation, only a single scale operation is defined inside the API to cope with all needed scaling actions (see Table 12). To add or remove application instances, the operation's `instances` parameter must be set to the desired number of application instances. If the number is greater than the currently available instances, additional instances will be created, otherwise instances will be removed. Due to restrictions on some platforms, there must always be at least one application instance. Requests for zero or negative instance numbers will fail and return an error to indicate a bad request. Vertical scaling is not yet included in this prototype. Further information about vertical scaling, the issues, and possible solutions are going to be presented as future work in Section 6.

POST	**Application**	**Application**		**200**
URL path	/endpoints/{endpoint_id}/applications/{application_id}/actions/scale			
Description	Scale the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
	instances	form	Integer	Desired number of application instances
Postcondition(s)	Available application instances must match defined number of the request's form parameters			

Table 12: Application object scale operation

3.3.5 Application Child Object Operations

In the previous chapters, the child objects of an application have already been mentioned numerous times. This section describes the most important operations to manage those objects. Table 13 contains the operations to retrieve the child objects or collections of them and Table 14 presents the methods to create and associate new child objects. All collections of child objects that are associated with an application can be retrieved in a similar manner.

The first definition in Table 13 shows that the request path has to be nested below an endpoint and application. Retrieving a specific instance of the child collections can be achieved by appending the ID of the object as additional path parameter. The second entry of the table illustrates this behavior. Both definitions can be translated to the other objects except Domains as well. All URL paths needed for those operations are visualized in the API resource map, which is shown in Figure 5.

GET	**Domain**	**DomainList**		**200**
URL path	/endpoints/{endpoint_id}/applications/{application_id}/domains			
Description	List all additional domains of the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application

GET	**Domain**	**Domain**		**200**
URL path	/endpoints/{endpoint_id}/applications/{application_id}/domains/{domain_id}			
Description	Retrieve a specific domain of the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
	domain_id	path	String	ID of the domain

Table 13: Application object operations to retrieve child object instances and collections

Except for the log objects which can only be changed by the provider of the endpoint, new domains, environment variables or services can also be added to an application by the user. Those operations are defined in Table 14. As previously stated, the response of each request

that creates new objects has the status code 201 to indicate that a new object was created. Moreover, all parameters of the methods are mandatory.

POST	**Domain**	**Domain**		**201**
URL path	/endpoints/{endpoint_id}/applications/{application_id}/domains			
Description	Add a new domain to the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
	domain/name	form	String	FQDN where the application shall be available at

POST	**Variable**	**EnvironmentVariable**		**201**
URL path	/endpoints/{endpoint_id}/applications/{application_id}/vars			
Description	Create a new environment variable for the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
	variable/key	form	String	key of the variable
	variable/value	form	String	value of the variable

POST	**Service**	**InstalledService**		**201**
URL path	/endpoints/{endpoint_id}/applications/{application_id}/services			
Description	Install a new service and bind it to the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	ID of the application
	service/id	form	String	ID of the service to install
	plan/id	form	String	ID of the plan to use with the service

Table 14: Application object operations to create and associate child objects

Operations to update the child objects follow those definitions, but as shown before, all form parameters are optional for update requests. Additionally, all objects can also be deleted by using HTTP's `delete` verb with requests against specific object instances.

3.3.6 Application Logging Operations

Log objects are currently the only application child objects that allow more than the basic CRUD operations to be executed. Table 15 contains the definitions of methods to download or tail the logs of an application. The first entry shows how to download all log files of an application, bundled as compressed archived. The archive format for the binary response can be specified via the `archive_format` query parameter. The second example also returns a binary archive, but contains only one specific log file. Next, the third entry shows the operation to load the contents of a specific log file. The response is not an object, but the actual content of the log file in plain text. Finally, the last entry defines how logs can be tailed. The response is an ongoing chunked stream, which sends new chunks with log entries as soon as they become available at the endpoint.

GET	**Log**	**Binary archive data with all log files**		**200**
URL path	/endpoints/{endpoint_id}/applications/{application_id}/logs/download			
Description	Download all log files of the application			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	The application's ID
	archive_format	query	String	Compression format of the downloaded archive (zip or tar.gz), default: zip

GET	Log	Binary archive data with the log file		200
URL path	/endpoints/{endpoint_id}/applications/{application_id}/logs/{log_id}/download			
Description	Download a specific log file			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	The application's ID
	log_id	path	String	The log's ID
	archive_format	query	String	Compression format of the downloaded archive (zip or tar.gz), default: zip

GET	Log	Log entries		200
URL path	/endpoints/{endpoint_id}/applications/{application_id}/logs/{log_id}			
Description	Show all entries of a specific log file			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	The application's ID
	log_id	path	String	The log's ID

GET	Log	Log entries, chunked response		200
URL path	/endpoints/{endpoint_id}/applications/{application_id}/logs/{log_id}/tail			
Description	Tail the log and continue to receive updates as long as the stream is open			
Parameter(s)	**endpoint_id**	path	String	The endpoint's ID
	application_id	path	String	The application's ID
	log_id	path	String	The log's ID

Table 15: Application logging operations

3.3.7 Vendor Specific Parameters

Even though the PaaS abstraction layer is designed to harmonize the actions of all supported platforms, there are still vendor specific parameters that might have to be passed to the platform from time to time. One example to illustrate the need for this feature is vertical scaling on OpenShift. In OpenShift, the performance of an application instance must be fixed when creating the application and cannot be altered afterwards. As the parameter of OpenShift's API is optional and uses a default value if not present, no parameters need to be supplied through the abstraction layer. However, a user might want to use a different instance size than the default one. To use vendor specific parameters, every input object of the API shall accept the `vendor_specific` key, into which any combination of key-value pairs can be placed, if `POST` or `PATCH` requests are invoked. Those values bypass the parameter validation of the abstraction layer's API and are added to the processed and documented parameters. If keys occur twice, the vendor specific values shall be preferred and overwrite the processed and documented parameters. Resources that accept `vendor_specific` parameters key are: Application, Domain, Variable, and InstalledService.

Listing 3 summarizes these definitions and presents a CURL command that creates a NODE.JS application on OpenShift. As mentioned above, the `gear_profile` is used to force OpenShift to instantiate a medium gear instead of the default small size gear.

Listing 3: Vendor specific parameters in a CURL request example

```
1 curl -X "POST" "{API_URL}/api/endpoints/openshift-online/applications" -H "Authorization: Basic
      {base64key}" -d "{\"application\":{\"name\":\"testapp1\",\"runtimes\":[\"nodejs\"],\"
      vendor_specific\":{\"gear_profile\":\"medium\"}}}"
```

3.3.8 Custom Endpoint API Calls

Combining the previously introduced API operations, the majority of common operations on all platforms are supported by the Nucleus API. However, in reality there can also be

situations in which those operations will not cover all requirements of the users. As an illustration, take the release management of Heroku's platform API, which is not supported by Nucleus. Releases describe a working combination of code, variables, and services that are persisted and to which one can roll-back in case of unexpected application issues. To let users access the release management or other proprietary operations, Nucleus features custom endpoint API calls. Those requests can either be called against the root URL path of the proprietary API or against a specific application. Form and body parameters that are included in the native API requests are passed to the endpoint without validation or modification. The response message and eventually arising errors are also forwarded without any modifications.

Custom calls that target the root URL path of the endpoint's API have to be called as pointed out in Listing 4. The `ENDPOINT_API_CALL_PATH` must be replaced with the complete URL path that shall be invoked on the endpoint's API.

Listing 4: Custom API call against the endpoint

```
1 /api/endpoints/{ENDPOINT_ID}/call/{ENDPOINT_API_CALL_PATH}
```

If the call shall be made on an application object, the URL is slightly different as shown in Listing 5. Here, the URL must also include the reference to the application object instance.

Listing 5: Custom API call against an endpoint's application

```
1 /api/endpoints/{ENDPOINT_ID}/applications/{APPLICATION_ID}/call/{
      ENDPOINT_API_CALL_PATH}
```

Listing 6 contrasts two API method calls against Heroku's endpoint. The first and fourth line show the call as it is described by Heroku itself, whereas the second and fifth line show the equivalent calls when made via Nucleus.

Listing 6: Heroku API call vs. Nucleus custom API call against Heroku

```
1 /account
2 /api/endpoints/heroku/call/account
3
4 /apps/{app_id_or_name}/releases/{release_id_or_version}
5 /api/endpoints/heroku/applications/{app_id_or_name}/call/releases/{release_id_or_version}
```

3.4 API Mappings

This section describes the mappings of objects and operations to achieve the abstraction layer and to harmonize the functionality as well as the object presentations of the four platforms. The section distinguishes between the object mapping, describing where the information to fill the response objects' fields can be obtained from, and the operation mapping which explains all methods and their execution plan to perform a Nucleus' operation on the platform.

3.4.1 API Object Mapping

The following sections show the mappings of the region, service, and application objects with all their associated child objects. Object mappings are a main part of the abstraction

between the platforms and a first step to solve some of the present semantic conflicts. The rules demonstrate how the API objects of the abstraction layer can be populated with data of the platforms' original API objects. Most of the objects' values can be applied without modifications, but some fields require processing of the original value or even state dependent solutions.

Region Mapping

Table 16 summarizes the region's attribute mapping and shows where the values can be taken from if not already present. By means of abstracting the differences properly, all platforms shall return a static default region object if they do not support multiple regions. The description shall mention the absence of the multi-region feature to prevent misunderstandings. On Heroku, the `name` must be mapped to become the `ID`. The remaining fields are already set. OpenShift's region object already has a description that mentions important aspects, for instance deployment restrictions. Both timestamps can be taken from a region's zone. The `ID` of Nucleus' object is the region's `name`.

Attribute \ Platform	**cloudControl**		**Cloud Foundry**		**Heroku**		**OpenShift**	
	Object	Field	Object	Field	Object	Field	Object	Field
id	-	*static*	-	*static*	region	name	region	name
created_at	-	*static*	-	*static*	region	✓	zones	min(created_at)
updated_at	-	*static*	-	*static*	region	✓	zones	max(updated_at)
description	-	*static*	-	*static*	region	✓	region	✓

Table 16: Region object attribute mapping

Service Mapping

Services, also referred to as add-ons or cartridges, are available on all four platforms. Table 17 lists the mapping for cloudControl and Cloud Foundry, whereas Table 18 contains Heroku's and OpenShift's mappings.

On cloudControl, services are known as add-ons. Timestamps and a description are not available at all. Add-ons do not have dependencies and the documentation URL can be created based on the add-on's name. An add-on has a free plan if at least one of the add-on's options has a `thirty_days_price` that is 0. Cloud Foundry has a rather complex service model, but fully supports Nucleus' object attributes. Services dependencies are properly described and require no complex mapping. The service can be free if there is at least one service-plan that is declared as free, too.

Attribute \ Platform	**cloudControl**		**Cloud Foundry**	
	Object	Field	Object	Field
id	addon	name	service/metadata	guid
created_at	✗		service/metadata	created_at
updated_at	✗		service/metadata	updated_at
name	addon	✓	service	label
description	✗		service	✓
release	addon	stage	service	version
documentation_url	addon	static + name	service	✓
required_services	-	*empty array*	service	requires
free_plan	addon/options	any(thirty_days_price == 0)	service-plans	any(free)

Table 17: Service object attribute mapping: cloudControl & Cloud Foundry

Heroku labels services as addon-service. The documentation URL is not given, but can be created analogous to cloudControl. Dependencies are not supported. A service can be free if any of its plans has a price in cents that is 0. OpenShift does not allow to specify a documentation URL for its cartridges. All remaining attributes can be retrieved. A cartridge is free if there are no usage rates specified for it.

Attribute \ Platform	Heroku		OpenShift	
	Object	Field	Object	Field
id	addon-service	✓	cartridge	✓
created_at	addon-service	✓	cartridge	creation_time
updated_at	addon-service	✓	cartridge	creation_time
name	addon-service	✓	cartridge	✓
description	addon-service	human_name	cartridge	✓
release	addon-service	state	cartridge	version
documentation_url	addon-service	static + name	✗	
required_services	-	*empty array*	cartridge	requires
free_plan	service-plans	any(price/cents == 0)	cartridge	empty(usage_rates)

Table 18: Service object attribute mapping: Heroku & OpenShift

Service Plan Mapping

On cloudControl, service plans cannot be accessed directly and have to be gathered from the service retrieval response. Service plans are thereby referred to as options of the add-on. A specific service plan is available at the position, say X, of the `addon/options` array. Timestamps and a description of the plan are not available. Costs appear only as fixed monthly charges, thus one static entry must be created for the API's response. The same applies to the price of the cost, which must be payed in the currency of the provider. The currency must be hard-coded as there is no function to determine it via the provider's API. As of now this construct is working because there is no known cloudControl provider to use more than one currency. When analyzing service plans on Cloud Foundry, one notices that there are no proper attributes describing the costs of used services. Furthermore, the solutions realized by the providers to circumvent this missing feature vary and do neither follow a common format nor a normalization. Pivotal IO specified how to access service plans via the API in their documentation[46]. IBM Bluemix does not describe the format in its documentation, but it can be deducted from response objects of their API. Altogether, it remains unclear how further providers of Cloud Foundry solved this problem. The structure of Nucleus' API objects is already capable to handle the pricing models of Pivotal IO's and IBM Bluemix's costs, nevertheless this feature has been postponed into later versions to evaluate further options in the meantime. Besides the service plan's costs, all remaining fields can be mapped without any discomfort.

Attribute \ Platform	cloudControl		Cloud Foundry	
	Object	Field	Object	Field
id	addon/options[X]	name	service_plan/metadata	guid
created_at	✗		service_plan/metadata	created_at
updated_at	✗		service_plan/metadata	updated_at
name	addon/options[X]	✓	service_plan/entity	✓
description	✗		service_plan/entity	✓
free	addon/options[X]	thirty_days_price == 0	service_plan/entity	✓
costs	-	*one array entry*	✗	
costs/period	-	*month*	✗	
costs/per_instance	addon/options[X]	price_is_per_box	✗	

[46] *PivotalIO - Catalog Metadata.* URL: `http://docs.pivotal.io/pivotalcf/services/catalog-metadata.html` (Retrieved: June 11, 2015).

Attribute \ Platform	cloudControl		Cloud Foundry	
	Object	Field	Object	Field
costs/price	-	*one array entry*	✗	
costs/price/currency	-	*determined by the provider*	✗	
costs/price/amount	addon/options[X]	thirty_days_price	✗	

Table 19: ServicePlan object attribute mapping: cloudControl & Cloud Foundry

Heroku provides only one cost per plan and one price per cost. Costs, which are always given in USD, are never based on the number of active dynos. However, the prices are given in cents and must be adapted to match the general decimal format. OpenShift V2 offers only rudimentary support for service plans. At the time of writing, all cartridges except one did not charge any costs. If there is at least one `usage_rates` specified in the cartridge, the `ID`, `period` and `amount` values can be taken from this object. The default mapping, which shall be applied when no `usage_rates` are given, creates a default plan with no costs and will also be used for private deployments of the platform where no marketplace is available.

Attribute \ Platform	Heroku		OpenShift		
	Object	Field	Object	Field	Default
id	plan	✓	cartridge/usage_rates[X]	plan_id	*default*
created_at	plan	✓	cartridge	creation_time	
updated_at	plan	✓	cartridge	creation_time	
name	plan	✓	cartridge/usage_rates[X]	plan_id	*default*
description	plan	✓	-	*static*	
free	plan/price	cents == 0	-	*false*	*true*
costs	-	*one entry*	-	*one entry*	
costs/period	plan/price	unit	cartridge/usage_rates[X]	duration	*hour*
costs/per_instance	-	*false*	-	*false*	
costs/price	-	*one entry*	-	*three entries*	
costs/price/currency	-	*USD*	-	*CAD* *EUR* *USD*	
costs/price/amount	plan/price	cents/100	cartridge/usage_rates	cad eur usd	*0.00*

Table 20: ServicePlan object attribute mapping: Heroku & OpenShift

Application Mapping

Nucleus' application object is not only a core part of the abstraction layer, but the mapping also has to consider more aspects than most other objects. All mappings are listed in tables 21 and 22.

On cloudControl, the `region` and `autoscaled` fields must be hard-coded as both features are not supported by the platform. Furthermore, as cloudControl only allows one runtime per application, the `runtimes` field must be an array with one value equal to the `active_runtime`. The `active_runtime` has to be determined in two different ways. If a custom buildpack is used, which is indicated by the value '*custom*' in the `app/type/name` field, the `buildpack_url` has to be used. Otherwise, `app/type/name` describes the valid buildpack. On Cloud Foundry, only two mappings require more effort than just referencing different fields. With Stackato[47] supporting the `autoscaled` feature, the presence of the field must be checked. If the field exists, its value shall be taken. If not, automatic scaling is not supported and the value has to be set to false. Moreover, the `web_url` field has to rely on a static configuration as there is no approach to identify the default domain at which applications will be available.

[47] *Stackato 3.4*. URL: `https://www.activestate.com/stackato` (Retrieved: October 31, 2014).

Attribute \ Platform	cloudControl		Cloud Foundry	
	Object	Field	Object	Field
id	app	name	app	metadata/guid
created_at	app	date_created	app	metadata/created_at
updated_at	app	date_modified	app	metadata/updated_at
name	app	✓	app	✓
active_runtime	app	type/name	app	detected_buildpack
	app	buildpack_url		
runtimes	-	array(active_runtime)	app	buildpack
region	-	*default*	-	*default*
autoscaled	-	*false*	app	autoscale_enabled (Stackato) OR false
instances	deployment	min_boxes	app	✓
web_url	deployment	default_subdomain	app	guid + static configuration
release_version	deployment	version	app	version
state	described in Section 3.4.1			

Table 21: Application object attribute mapping: cloudControl & Cloud Foundry

Most of Heroku's application mappings can be obtained from the application object. Mapping rules must be considered for the `instances`, `release_version` and `runtimes` fields. The number of application `instances` can be determined by counting all workers of the type WEB that are listed in the `formation` object. All `runtimes` can be gathered by loading the `buildpack-installations` object and presenting an array of the individual buildpack's URLs. A region is already included in the application object of Heroku, but as only an identifier and not a nested object is wanted, the name of the region object must be remapped. To diagnose the `release_version`, there are two approaches. First, if no dyno is assigned to the application, the ID of the latest version that is contained in the `releases` object can be used. The second approach, if any dyno is assigned, is to use the ID of the latest version that is assigned to one of the dynos. OpenShift does not support updating the application object after creation, wherefore there is also no update timestamp. The `runtimes` field is supposed to be an array with one entry, the assigned `active_runtime`, as there can only be one runtime per application on OpenShift. The `region` can be identified by analyzing the `gear_group` and the individual regions of the `gears`. The `release_version` of the application is available in the `sha1` field of the active `deployment`. An active deployment can be identified by comparing a deployment's `activations` and retrieving the deployment with the latest activation timestamp.

Attribute \ Platform	Heroku		OpenShift	
	Object	Field	Object	Field
id	app	✓	app	✓
created_at	app	✓	app	creation_time
updated_at	app	✓	✗	
name	app	✓	app	✓
active_runtime	app	buildpack_provided_description	app	framework
runtimes	buildpack-installations	array(buildpack/url)	app	array(framework)
region	app/region	name	gear_groups	gears/region
autoscaled	-	*false*	app	scalable
instances	formation	quantity (type: web)	app	gear_count
web_url	app	✓	app	app_url
release_version	releases	id[max(version)]	deployments	active(deployment)/sha1
	dyno	release/id[max(release/version)]		
state	described in Section 3.4.1			

Table 22: Application object attribute mapping: Heroku & OpenShift

Application State Detection Rules

To put the generic PaaS application lifecycle (see Section 3.1.3) into practice, we require a set of rules explaining how an application's state can be derived on the different platforms. Figure 6 visualizes the initial detection rules for all four platforms as decision trees. The

rules cover the most common states as described in the platform's documentation and as identified during the examination of the platform.

OpenShift's application states can be derived for the most part by examining the state of the application's gears. Moreover, additional information on the existing deployments is needed. The most complex task is the detection of the DEPLOYED state, precisely the initial deployment. To reveal this state, we need to sum up the number of deployment activations, excluding those of the initially available deployment. If the sum is exactly one, the application has to be in the DEPLOYED state. However, the original deployment, which is provided automatically by OpenShift, must still be in the list of the kept deployments for this rule to apply. Otherwise, the application is not in the deployed state. The detection of the remaining states is self-explanatory.

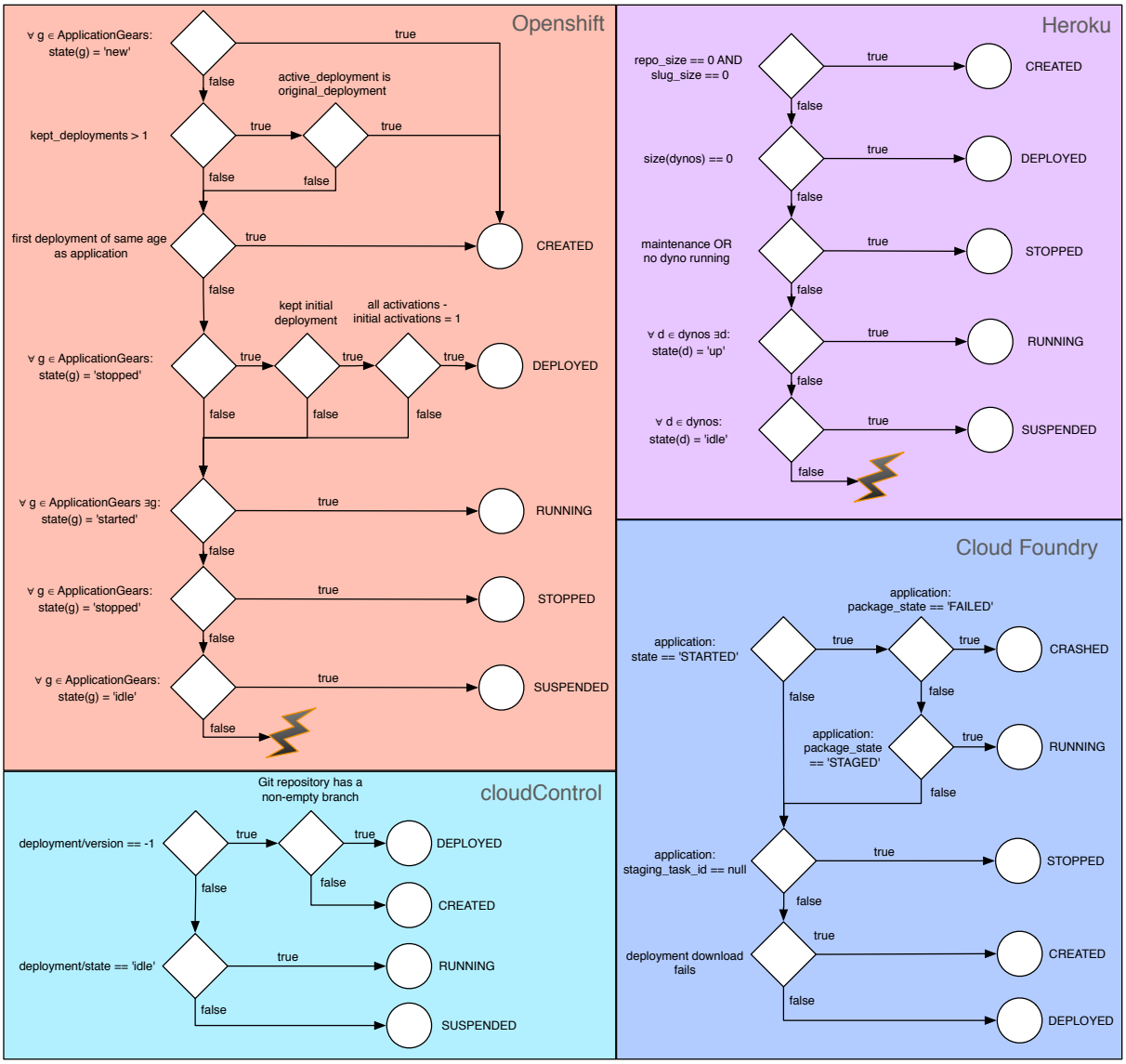


Figure 6: Application state detection rules

Heroku's detection rules first evaluate whether there has been any data uploaded to the platform. If not, the application must be in the CREATED state. In case no dyno is assigned to the application, the DEPLOYED state is most appropriate. During maintenance, or when all assigned dynos are down, the application is STOPPED. If none of the existing rules applies, the application state could not be detected, i.e., is unknown. In cloudControl's lifecycle the

application is RUNNING if a version has been deployed and the deployment's state is not idle. When there is no deployed version, the Git repository must be evaluated to detect the difference between the CREATED and DEPLOYED state. The application is in the DEPLOYED state if there exists any branch on the repository which indicates an existing deployment. When there is no branch, the application has only been CREATED. On the Cloud Foundry platform, one can use the application's state, package state and staging task properties as well as the deployment data download method to deduct the necessary set of rules. After starting an application it is either RUNNING if the package state is staged or CRASHED if it has failed. A staging task ID with a null value indicates that the application has been STOPPED. If a deployment data download returns a URL where the data could be downloaded from, the application is DEPLOYED. Receiving an error for the download request suggests that the application environment has only been CREATED.

In general, the above rules cover most scenarios. However, some of the detection rules might need further enhancements to improve the accuracy, especially for applications that are managed with varying interfaces and not solely via Nucleus.

Domain Mapping

The domain information can be mapped as described by the rules shown in Table 23. One obvious fact is the different naming of the concept domain among the platforms. Heroku uses a term identical to Nucleus', whereas cloudControl and OpenShift label domains as alias and Cloud Foundry refers to them as a combination of routes and domains. Cloud Foundry uses domains to reuse second level domains. With the use of routes, domains of a deeper level can be bound to an application. If the mapped route has a defined host, then the mapping needs to combine this host with the domain's name, otherwise only the domain's name must be used.

Platform / Attribute	**cloudControl**		**Cloud Foundry**		**Heroku**		**OpenShift**	
	Object	Field	Object	Field	Object	Field	Object	Field
id	alias	name	route	metadata/guid	domain	✓	alias	✓
created_at	alias	date_created	route	metadata/created_at	domain	✓	✗	
updated_at	alias	date_modified	route	metadata/updated_at	domain	✓	✗	
name	alias	name	route domain	host name	domain	hostname	alias	id

Table 23: Domain object attribute mapping

Environment Variable Mapping

On all four platforms, the mapping of Nucleus' environment variables lacks support for the object's timestamps. In Table 24, the mappings of cloudControl and Cloud Foundry are listed. All mapping rules for Heroku's and OpenShift's variables are shown in Table 25.

On three platforms, namely cloudControl, Cloud Foundry and Heroku, the variables are not presented as discrete objects but can only be extracted from an enumeration. To be able to use environment variables on cloudControl, the free addon `config.free` must be installed. The variables are then referred to as `config vars`, likewise to the naming on Heroku. On Cloud Foundry the variables are accessible in the environment's `environment_json` field. OpenShift is the only platform that makes variables available as dedicated API objects.

Attribute \ Platform	cloudControl		Cloud Foundry	
	Object	Field	Object	Field
id	config.free	settings/config_vars/{key}	env	environment_json/{key}
created_at	✗		✗	
updated_at	✗		✗	
key	config.free	settings/config_vars/{key}	env	environment_json/{key}
value	config.free	settings/config_vars/{value}	env	environment_json/{value}

Table 24: Environment variable object attribute mapping: cloudControl & Cloud Foundry

Attribute \ Platform	Heroku		OpenShift	
	Object	Field	Object	Field
id	config-vars	{key}	environment-variable	name
created_at	✗		✗	
updated_at	✗		✗	
key	config-vars	{key}	environment-variable	name
value	config-vars	{value}	environment-variable	✓

Table 25: Environment variable object attribute mapping: Heroku & OpenShift

Log Mapping

On all selected platforms, there exists no API operation to list the available logs. For this reason, we include all log files which are available, e.g., inside the application's `logs` directory. Furthermore, due to the variety of data structures, the log entries are also passed without modification. Additionally, the creation time is retrieved from the application and the update timestamp always uses the current time to provide at least an approximate information.

Installed Service Mapping

Most of the mappings are identical to those of the general services as described in Section 3.4.1.

For a proper mapping on cloudControl, two objects are needed: the general add-on description as well as the assignment object of the service to the application. Installed services are part of the deployment, which itself is part of the application. The Cloud Foundry mapping requires the service binding object which includes or points to all other required objects, e.g., the service instance. The ID of the binding object is also used for Nucleus' object. This is contrary to all other platforms that still operate with the original service identifier.

Attribute \ Platform	cloudControl		Cloud Foundry	
	Object	Field	Object	Field
id	see Table 17		binding	metadata/guid
created_at	see Table 17		binding	metadata/created_at
updated_at	see Table 17		binding	metadata/updated_at
active_plan	addon assignment	addon_option/name	instance	service_plan_guid
web_url	✗		instance	dashboard_url
properties	addon assignment	settings	binding	credentials

Table 26: Installed service object attribute mapping: cloudControl & Cloud Foundry

On Heroku, all config-vars have to be obtained before those key-value pairs can be selected that belong to the installed service. Whereas general services are defined as addon-service, installed services are just known to be an addon. As of now, OpenShift does not yet fully support or use plans for their cartridges.

Attribute \ Platform	Heroku		OpenShift	
	Object	Field	Object	Field
id	see Table 18			
created_at	see Table 18			
updated_at	see Table 18			
active_plan	addon	plan/id	-	*static*
web_url	addon	web_url	✗	
properties	config-vars	name is in addon/config_vars		✓

Table 27: Installed service object attribute mapping: Heroku & OpenShift

3.4.2 API Operation Mapping

In the previous sections, all API operations, their parameters as well as the API's response objects were introduced. Extending those definitions, this section discloses the operations' mapping. The mappings describe all operations that are required to be executed on the platforms to achieve one of the abstraction layer's actions. All operation mappings are defined in Table 33, which can be found in Appendix A. By means of simplifying the mapping rules, additional operations that can be executed for validation, e.g., of the given request parameters or the current application state, are not included in the mappings. All shown operations must be executed in the given order and by obeying the noted conditions.

The mapping of the available services as well as their associated service plans can be achieved on all four platforms. On Heroku and Cloud Foundry the operations can be forwarded so that only the returned objects must be processed. By adding the INLINE-RELATIONS-DEPTH=1 query to requests on Cloud Foundry, nested elements of the first level will be embedded in the response object, which saves one additional request. OpenShift and cloudControl do not allow to retrieve specific services or service plans, but they can only be extracted from the corresponding collection.

Concerning the retrieval of all or specific available deployment regions, mappings are only required for Heroku and OpenShift as Cloud Foundry and cloudControl do not offer a choice of the deployment region. The mappings are both straightforward and do not require any special operations. However, as there is no operation to retrieve a specific region object on Heroku and OpenShift, the implementation should always load the list and only process the region objects needed to build the response. On OpenShift it must also be regarded that the retrieved region has to be declared as active, which is denoted in the `allow_selection` field.

Next, all mappings concerning the application object and its operations are presented. Applications can be retrieved and listed on every of the four platforms. Still, loaded applications on cloudControl must be merged with the used deployment, which act as more or less as separate applications and shall default to *nucleus* if the application was created with the abstraction layer. Also when loading the application list, all gathered application objects must be joined with the corresponding deployment object. When creating an application on cloudControl, a runtime, the application name and 'git' as repository type must be specified. An initial deployment must always be created to guarantee the expected functionality on cloudControl, whereby its name shall default to *nucleus*. Furthermore, the environment variables feature, which is part of the core functionalities on the other three platforms, must be added explicitly via the *config.free* add-on. As the *config.free* add-on can only be added with at least one initial value, a Boolean variable named *nucleus-initialized* is provided and gets removed immediately again to achieve the desired clean state. Cloud Foundry requests the identifier of the user's space to create an application object. Therefore, this ID must

be gathered before firing the request to create the application. Next, it is also necessary to assign the default route to the application, which corresponds to the default *web_url* as it is offered by most platforms. The default route can either be identified from the list of all public routes, for instance if there is only one public route, or can be provided as a fixed value in the abstraction layer's configuration that is then assigned to the persisted *endpoint* object. For Heroku, no additional application mappings are required, only the chosen runtime must be applied in a second request if it is not natively offered by Heroku. OpenShift requires three requests to create an application object that is compatible to Nucleus. In the first request, the user's space must be identified, whereby a user always has only one space. With the ID of this space, the application can be created. The third request is needed to disable the auto deployment upon Git requests and keep at least two of the previous deployments, allowing the application to remain in the DEPLOYED state. Not doing so would cause OpenShift not to fit the generic lifecycle. Updating an application cannot be achieved on cloudControl and OpenShift, wherefore there are no mappings required. On Heroku, custom buildpacks must be regarded identically as when creating the application. No special operation mappings are required on Cloud Foundry. Deleting the application object must be concluded with the deletion of the application object itself on all four platforms. Regarding cloudControl, all deployments must be deleted before the application deletion will be accepted. If dealing with a Cloud Foundry application, the established default domain must be deleted first to prevent orphaned routes.

Application data deployment requires similar operations on cloudControl, Heroku, and OpenShift. At first the URL of the Git repository and the user's account name must be retrieved. Thereupon, the Git repository shall be cloned and the user's uploaded files get extracted into the repository's working directory. With the commit and push commands the Git interaction is finished, as this automatically executes the deployment process on Heroku. On cloudControl and OpenShift, the build process must be triggered manually. The version parameter of minus one that is used with cloudControl thereby refers to the HEAD of the Git repository that shall be used for the build. Nevertheless, the build shall only be executed if the application has already been started, otherwise the called build would immediately start the application and violate the generic lifecycle. Forcing a clean build on OpenShift prevents issues with dependencies and the chosen runtime cartridge. Opposed to the other three platforms, Cloud Foundry does not natively support the deployment via Git repositories, even if some providers, for instance Stackato and IBM Bluemix, enhanced their endpoints with this feature. The data can be deployed by embedding the zipped data archive in a multipart request. By specifying the resources parameter as empty squared brackets, no data may be reused from previous builds.

The rebuild operation requires the same HTTP requests to be executed if using cloudControl, Heroku or OpenShift. On all three platforms, the only noticeable difference compared to the deployment is that instead of extracting the uploaded data into the repository, a marker file is modified so that a new Git commit can be created and pushed. Cloud Foundry provides a native *restage* operation which can be called.

To download previously uploaded application data, Cloud Foundry also provides an operation which either returns the data embedded in the response or contains an URL redirection pointing to the location where the file can be found. If using one of the Git enabled platforms, only the repository's URL must be retrieved, whereupon the repository can be cloned, compressed, and finally returned with the response.

Lifecycle operations are generally not supported by cloudControl. However, to initially start an application, the build must be executed using the data of the HEAD repository. Horizontal scaling can be achieved by setting the number of desired instances in the *min_boxes* field of a deployment update request. Cloud Foundry applies the state and number of instance of an application if either one was requested while updating the object itself. Heroku can be started and stopped by triggering the maintenance mode of the application. However, the maintenance mode does not stop background workers, hence they must be stopped manually in order to achieve a proper stopped state. Likewise, if resuming the application, the workers must also be started again. Currently, there is no mechanism to scale the workers to the same level as before the stop. This issue could be resolved by memorizing the number of active workers inside configuration variables in future versions. The scaling of application instances works just as the worker management does, but refers to *web* instances instead. Events are used on OpenShift to handle the state transitions and even scale the application. Mapping efforts are needed mostly to calculate whether and how often the application must be scaled-up or scaled-down to achieve the desired number of application instances. Thereby, the user data is needed to analyze the maximum number of application instances the user can deploy, whereas the application object reveals the number of currently deployed application instances.

The operations for domain mappings can be used without adaptations on Heroku and OpenShift. On cloudControl not all essential information is contained in the returned alias collection, wherefore each of the list's elements must be retrieved individually, before returning the mapped response. In contrast to these three platforms, severe mapping efforts are required on Cloud Foundry to handle domains in an equivalent manner. Cloud Foundry distinguishes between private domains, which can be accessible to selected users or user groups, and shared domains that are usually open to all users of the endpoint and also reflect the default `web_url`. Domain routes, representing a subdomain of the private or shared domain, can be assigned to an application. If retrieving already assigned domains, the route must be loaded at first before combining it with the actual domain and returning the combined response. Creating the domain requires multiple steps and starts by fetching all accessible private and public domain objects. In case there is no domain matching the request's parameters, a new private domain shall be created. Next, all routes are loaded and analyzed as there is no possibility to reliably check if domain names, respectively routes, are already taken. If the route is not yet taken, it shall be created. Finally, the route, which is either the existing or a newly created route, gets assigned to the application. If a domain should be removed from a Cloud Foundry application, the route assignment is removed from the application. Thereafter, a check is performed if the route is still used in other applications. If not, it shall be removed to prevent orphaned domain and route junks. The domain is currently not tested for further usage, but could theoretically also be removed if it is of type private and not referred to anymore.

Installed services on Heroku and OpenShift require no operation mapping when fetching objects. On cloudControl, all assigned add-ons must be loaded, combined with the separately retrieved add-on assignment object and finally be enriched with the information of the general add-on description. The add-on assignment object as well as the general add-on object are necessary to populate the response object, as described in Table 26. To gather the service assignment, the currently active plan must be known as it is a path parameter of the expected URL. Therefore, all available add-ons must be loaded, too. Mapping the operations to retrieve the installed services on Cloud Foundry, one must load the service bindings, whereby the nested elements of the next level shall be included in the response.

To retrieve a specific service binding, the object's ID is needed and in consequence all the service's plans must be loaded, too. Benefiting from the usage of globally unique identifiers, the combination of the service's plans and all service bindings of the application reveals the searched binding. Except for Cloud Foundry, services can be modified with only one request on all platforms. Installing a service on Cloud Foundry requires to create a new instance of the service and bind this instance to the application. To update an installed service, the binding is needed and the same logic as when retrieving an installed service applies. When removing a service from the application, the binding has to be identified, then it must be removed from the application and finally the service instance must be deleted, too.

Environment variables on OpenShift are treated as discrete objects, just as specified in the Nucleus API. However, the remaining three platforms all regard the variables as key value pairs and therefore they cannot be retrieved individually. Instead, the list of all variables must be loaded and the desired variable must be extracted to build the response of the abstraction layer. When applying environment variables on Cloud Foundry, one must provide the complete set of the environment variables, meaning also currently applied variables must be included to make sure they do not get deleted. The `forced` parameter on cloudControl is used to always apply variable values and internally reuse the same operation to create and update variable values.

Accessing the log files is the most diverse part of the Nucleus API. cloudControl does not offer the available log files in its API, but to receive the entries of a known log one can fire an HTTP request. Likewise, the same URL path can be used to tail the log, but then requires a timestamp that specifies the earliest date of the log messages that shall be included in the response. Cloud Foundry offers an operation to access the file system of the deployed application, which sometimes contains log files. Files being retrieved via this operation must be combined with the statically defined logs that are provided by *loggregator*, the logging system of Cloud Foundry. Logs of the *loggregator* can be retrieved via normal HTTP requests, but in order to tail the logs a web socket based communication must be established. The web socket then automatically pushes new log entries once they are available on the *loggregator*. All messages received from the *loggregator* are binary files that were serialized with Google's Protocol Buffer[48] technology and must be deserialized before their content can be processed. Heroku's log list is completely static. All logs can be accessed upon requesting a valid log session, which then returns the URL at which the files can be accessed. If the session is opened with tailing enabled, the returned response is a chunked HTTP stream that remains open for as long as possible and periodically pushes new log entries. To access the log files on OpenShift, one needs to SSH into an application instance and directly retrieve the application log files from a directory inside the application artifacts. The *download* and *download_all* operations of the logging operations group do not require mappings on any of the supported platforms. The logic to provide those operations can be maintained within the API by using a combination of the previously specified log operations.

3.4.3 API Request Mapping

The operation mappings of the previous parts of this chapter already show how to translate the Nucleus API operations to the vendor's API and the way in which response objects

[48] *Google Protocol Buffers*. URL: `https://developers.google.com/protocol-buffers/` (Retrieved: June 25, 2015).

can be build properly. Nonetheless, the presentation of the HTTP request headers against the platforms is still missing to conclude the definitions. Headers are not only essential to enable the authentication against the endpoint, but also define the utilized API version of the platform and its expected response format.

In Section 2.2 of the approach, the chosen vendors and their API authentication methods were initially introduced. All requests that must be executed to obtain an API token, which is essential to execute further requests, are now presented in Table 28.

As OpenShift relies on HTTP Basic authentication and does not provide any token based access, there are no requests required in advance to executing further operations. cloudControl provides an API token once the credentials were posted with HTTP Basic authentication to the `/token` URL path. A dedicated token refresh is not offered, but a new token can be obtained by repeating the initial acquisition. Cloud Foundry based platforms do not have a common URL path at which the tokens can be obtained. However, the authenticator's URL path is included in the general endpoint information. The request that has to be posted towards the gathered URL path must define the GRANT_TYPE and include the credentials. If the token should be refreshed, the GRANT_TYPE must be set to *refresh_token*. Heroku is known to provide OAuth 2 support as recommended authentication approach. The API tokens can thereby be received when the OAuth client was manually registered with the platform. Although this would also be realizable by submitting the already known API token to the Nucleus API, it conflicts with a consistent approach that can be used on all four platforms. Therefore, the prototype is not going to use the OAuth 2 authentication but rather rely on Heroku's API token acquisition that expects the credentials to be sent to the `/login` URL. This approach is also used by the Ruby implementation of the Heroku CLI. New tokens can be requested by repeating the default authentication request.

Operation	cloudControl	Cloud Foundry	Heroku
acquire API token	POST /token *Header:* • Authorization = 'Basic {*username:password, base64 encoded*}'	GET /v2/info POST {*info/authorization_endpoint*}/oauth/token?grant_type=password&username={username}&password={password}	POST /login?username={*username*}&password={*password*}
refresh API token	***acquire API token***	GET /v2/info POST {*info/authorization_endpoint*}/oauth/token?grant_type=refresh_token&refresh_token={*access/refresh_token*}	***acquire API token***

LEGEND

Font Style	***bold + italic***	: Use other operation
	{in curly braces}	: request variable
	{*italic in curly braces*}	: variable with static or previous response related value

Table 28: Authentication operation mapping

Utilizing the acquired API token, the HTTP request headers of Table 29 must be used for every request against the determined endpoint. On all platforms, the exchanged content type is expected to equal JavaScript Object Notation (JSON). All platforms expect the authentication information to be present in the `Authorization` header. Cloud Foundry additionally requires a `Basic` property, which must be set to 'Y2Y6' in order to make the token based API authentication work. Both values, `token_type` and `access_token` are included in the response when asking for the API token. Heroku and OpenShift both require an additional `Accept` header for all requests. As in the definition of the Nucleus API, it determines which API version has to be used. Requests without the API version are either rejected or would use the latest version, which could generate errors as soon as an API change on the platform renders the abstraction layer's adapter incompatible.

Platform	Header
cloudControl	• Content-Type = 'application/json' • Authorization = 'cc_auth_token="{*api_token*}"'
Cloud Foundry	• Basic = 'Y2Y6' • Content-Type = 'application/json' • Authorization = '{*token_type*} {*access_token*}'
Heroku	• Accept = 'application/vnd.heroku+json; version=3' • Content-Type = 'application/json' • Authorization = 'Bearer {*api_token*}'
OpenShift	• Accept = 'application/json; version=1.7' • Content-Type = 'application/json' • Authorization = 'Basic {*username:password, base64 encoded*}'

Table 29: Authenticated platform requests and the essential header fields

3.5 API Design Challenges

Recalling the disclosed API mappings, most challenges that were experienced while specifying the Nucleus API could be resolved in the end. However, some fields of the API objects could not be filled with proper values of the platform's response objects. Missing timestamps could only be copied from related objects, but are expected to present similar values. As most of the problematic mappings provide only meta-data, the requests do not have to fail if any of those fields can't be mapped at all.

Cloud Foundry failed to provide standardized definitions of the service plan costs and is therefore the only object mapping which could not be harmonized in the adapters. Custom mappings would be possible if the provider is known, but for most providers this information must be obtained first. Even then this approach would remain unstable and likely to break if anything is changed or additional providers appear with their new custom schema.

One of the first challenges was the completely inconsistent definition of the deployment and build process. Whereas some differences, for instance different deployment methods, were expected, it was surprising that the deployment is sometimes only serving as mechanism to upload data, whereas others understand deployment to also act as a build process. Heroku and Cloud Foundry both automatically trigger a build process once the data is uploaded, cloudControl and OpenShift expect a separate API request to invoke the actual build process. By expecting that the deploy operation performs all necessary steps, ranging from data upload, over the build process up to deploying the finalized build, all inconsistencies could be harmonized.

Some challenges arose solely on behalf of cloudControl. Environment variables and domains are both not part of a default application on cloudControl, instead they have to be provided via services or add-ons, as cloudControl refers to them. To circumvent any conflicts, both add-ons are now added to the application immediately upon its creation. Both add-ons are free of charge, wherefore there are also no known downsides of this procedure.

Moreover, at first glance cloudControl seemed to violate the generic lifecycle of PaaS applications. Start and stop operations are not offered by the platform. Once an application has been deployed, it cannot be taken offline except by deleting the application or its data. However, a slight adaption of the lifecycle made all platforms compatible and provides more flexibility for further platforms. As depicted in the final definition of the lifecycle (see Figure 4), the build process is shown as first action after the start operation of the application instance. If builds are already part of the deployment process, as with Cloud Foundry and Heroku, the application state will not change if the build fails and the start will only fail on errors unrelated to an already succeeded build. The cloudControl adapter now somewhat

supports the start operation and thereby checks if the application has already been build and started. Nothing needs to be done if the application is started, otherwise the build will be triggered so that the application instance gets started.

Application instances are another aspect that is regarded differently by the platforms. At first, it was planned to allow the adjustment of application instances within a range from zero up to the limitation of the endpoint. This intention was however troubled by cloudControl, Cloud Foundry and OpenShift, which all require the presence of at least one application instance. Despite the limitation of Heroku that allows the removal of all instances, the scaling operation was finally designed to accept only positive integer inputs, which must be greater or equal than one. We do not expect any major side effects of this decision as the application can still be stopped to perform maintenance work or save the costs of the remaining application instance.

4 Prototype

Chapter 4 introduces the final prototype, explains its architecture, and presents important implementation details. The utilized technologies are summed up in Section 4.1 and the final structure of the project is explained in Section 4.2. All steps and files related to the initialization and loading of the code parts are demonstrated in Section 4.3. In Section 4.4, the extensible API route setup is introduced, whereupon Section 4.5 discusses the adapters and their hierarchy. Section 4.6 illuminates the handling of Git repositories with its challenges and the appropriate solutions, before Section 4.7 presents the custom errors and a general approach to exception handling. All actions that are involved in an API request are highlighted in Section 4.8, before Section 4.9 reveals the test setup of the prototype. Information about the documentation of the API are presented in Section 4.10. Finally, Section 4.11 presents instructions how to use the current prototype, adapt the default configuration and develop new adapters.

4.1 Technology

After an analysis of the API design, all related platforms and their provided libraries, it was decided that the Nucleus prototype is going to be implemented in Ruby[49]. Ruby is an open-source, dynamic, object-oriented and general-purpose programming language that is available for all major operating systems. It is quite popular in the area of PaaS applications and also many PaaS vendors utilized Ruby to implement their CLIs or other libraries that connect to their API. Another reason in favor of Ruby was the multitude of projects that are available to build, document, and test RESTful APIs.

The programming language Ruby is very often connected to the Ruby on Rails[50] web framework. One reasonable choice to build the API would have been the Rails::API[51] project, which has only recently been merged into Rails itself. However, as the projected API does not require most of Rails' core features, a more lightweight Rack[52]-based infrastructure was chosen instead. Rack is a modular and extensible interface, wrapping HTTP requests and responses into a single method call and is implemented by a variety of different application servers. After a careful consideration of all requested features, the Grape[53] project was chosen to build the API.

Grape is a micro-framework that allows to build RESTful web applications that can be run independently on top of Rack, as well as on top of existing web application frameworks as Rails or Sinatra[54]. With its extensions grape-entity[55] and grape-swagger[56], it provides a feature rich development environment to develop the Nucleus API in an object-oriented and

[49] *Ruby*. URL: `https://www.ruby-lang.org/en/` (Retrieved: June 25, 2015).
[50] *Ruby on Rails*. URL: `http://rubyonrails.org/` (Retrieved: June 25, 2015).
[51] *Rails::API*. URL: `https://github.com/rails-api/rails-api` (Retrieved: June 25, 2015).
[52] *Rack: a Ruby Webserver Interface*. URL: `http://rack.github.io/` (Retrieved: June 25, 2015).
[53] *grape: An opinionated micro-framework for creating REST-like APIs in Ruby*. URL: `https://github.com/intridea/grape` (Retrieved: June 25, 2015).
[54] *Sinatra*. URL: `http://www.sinatrarb.com/` (Retrieved: June 25, 2015).
[55] *grape-entity: A simple Facade to use with your models and API, sitting on top of an object model*. URL: `https://github.com/intridea/grape-entity` (Retrieved: June 25, 2015).
[56] *grape-swagger: Swagger compliant documentation of your grape API*. URL: `https://github.com/tim-vandecasteele/grape-swagger` (Retrieved: June 25, 2015).

properly documented manner. Grape supports multiple message formats out of the box, including XML and JSON.

In analogy with the supported platforms, the Nucleus API shall utilize JSON as message format for both, request and response messages. With the use of the Hypertext Application Language (HAL)[57], a simple JSON based format to enable hyperlinks between API resources, the API matches the ideas behind the HATEOAS principle of RESTful applications. HAL facilitates not only the exploration and use of an API, but also leverages automatic processing of the resulting JSON documents. Clients can easily navigate the API by following the links of a resource, which are embedded in the `_links` property. The name of a link is described by the key of the entry within the `_links` object, whereas the actual URL of the link has to be the value of the nested `href` property. Listing 7 shows the Application object as an exemplary resource to utilize HAL in the JSON notation and to establish the resource links. The object shows six links, the `self` reference, the link to the `parent` resource, in this case the endpoint itself, as well as the links to the child resource collections.

Listing 7: HAL example of Nucleus' application object using JSON

```
{
  "id": "e0946358-8ab4-45c5-ab5a-9ee67e937318",
  "created_at": "2015-03-19T18:11:52Z",
  "updated_at": "2015-05-09T10:56:08Z",
  "name": "nucleus",
  "active_runtime": "Ruby",
  "runtimes": [],
  "region": "eu",
  "autoscaled": false,
  "instances": 0,
  "web_url": "https://nucleus.herokuapp.com/",
  "state": "deployed",
  "release_version": "08c6965f-3627-408f-9538-845d8ec79520",
  "_links": {
    "self": { "href": "http://localhost:9292/api/endpoints/heroku/applications/
        e0946358-8ab4-45c5-ab5a-9ee67e937318" },
    "parent": { "href": "http://localhost:9292/api/endpoints/heroku" },
    "logs": { "href": "http://localhost:9292/api/endpoints/heroku/applications/
        e0946358-8ab4-45c5-ab5a-9ee67e937318/logs" },
    "vars": { "href": "http://localhost:9292/api/endpoints/heroku/applications/
        e0946358-8ab4-45c5-ab5a-9ee67e937318/vars" },
    "domains": { "href": "http://localhost:9292/api/endpoints/heroku/applications/
        e0946358-8ab4-45c5-ab5a-9ee67e937318/domains" },
    "services": { "href": "http://localhost:9292/api/endpoints/heroku/applications/
        e0946358-8ab4-45c5-ab5a-9ee67e937318/services" }
  }
}
```

For building the prototype, additional features are required besides those provided by the Grape library. Ruby uses so called gems to provide and distribute self-contained programs and libraries. All supplementary Ruby gems that are used within the prototype are listed in Table 30, including a short description why they are needed.

[57] *HAL - Hypertext Application Language*. URL: `http://stateless.co/hal_specification.html` (Retrieved: June 3, 2015).

Gem	Version	Usage description
configatron	>= 4.5	Used as global configuration
em-http-request	>= 1.1	Required for log tailing against HTTP endpoints
excon	>= 0.44	Used as main HTTP / REST client
faye-websocket	>= 0.9	Required for log tailing against websockets
git	>= 1.2	Application data handling
grape	>= 0.12	Used to build the API
grape-entity	>= 0.4.5	Grape extension used to build the API in an object-oriented manner
grape-swagger	>= 0.10.1	Grape extension used to document the API
kwalify	>= 0.7	Used to import the vendor, provider and adapter setup from a configuration file with schema validation
lmdb	>= 0.4	DB store, used on Windows systems
logger	>= 1.2	Logging
mime-types	>= 2.5	Application archive handling, detect unsupported uploads
moneta	>= 0.8	Generic interface for DB store implementations
oj	>= 2.12	Used for JSON / Hash conversion and test cassette serialization. Oj is considerably faster than most JSON libs
protobuf	>= 3.4	Required for Cloud Foundry log messages
rack-ssl-enforcer	>= 0.2.8	To make sure HTTPS is used instead of HTTP
rest-client	>= 1.8	HTTP client library to use for multipart requests
rack-stream	= 0.0.5	Used to build a streaming API for the log tail action
request_store	>= 1.1	Save certain information for the current request, e.g., the already loaded adapter
require_all	>= 1.3	Application setup, require all parts of the application
rubyzip	>= 1.1	Application data handling
sshkey	>= 1.6.1	Application data handling when using git deployment
thin	>= 1.6	The ONLY supported rack server at the moment

Table 30: Nucleus' gem dependencies

The dependencies, which are maintained by Bundler[58], can also be found in the `.gemspec` file. The prototype can also be made available as Ruby gem itself, so that other developers can utilize the abstraction layer's communication features without having to setup a complete web server that runs the API. This setup allows future CLIs of the abstraction layer to use the gem, rather than having to connect to a web API. With this final change of the application architecture and its packaging, the abstraction layer's architecture that was previously defined in Figure 1 can be improved and visualized as shown in Figure 7. The visualization now considers the different clients of the abstraction layer, namely either Ruby programs or HTTP clients that connect to a deployed version of the API.

4.2 Project Structure

The internal structure of the project is shaped mostly by the guidelines of Bundler[59]. The structure of the root directory is visualized in Figure 8. Currently, the deliverable is one project that actually contains two projects, the abstraction layer and the web API, which are yet to be separated into two dedicated projects. The separation of Nucleus into a CORE and an API gem would enhance the separation of concerns so that users only have to use the project they really need.

Binary startup files in the BIN folder are part of the API and allow to start the web server with a variety of configuration options, e.g., to run the server as a daemon. Configuration files of Nucleus are placed in the CONFIG directory. It contains the adapter configurations, which would be part of the CORE feature set, as well as the general configuration that could be found in both projects, once separated. Yard's generated HTML documentation of the Ruby modules and classes is located in the DOC directory. Most important, the LIB

[58] *Bundler: managing gem dependencies.* URL: `http://bundler.io/` (Retrieved: June 25, 2015).
[59] *How to package your Ruby code in a gem.* URL: `http://guides.rubygems.org/make-your-own-gem/` (Retrieved: June 28, 2015).

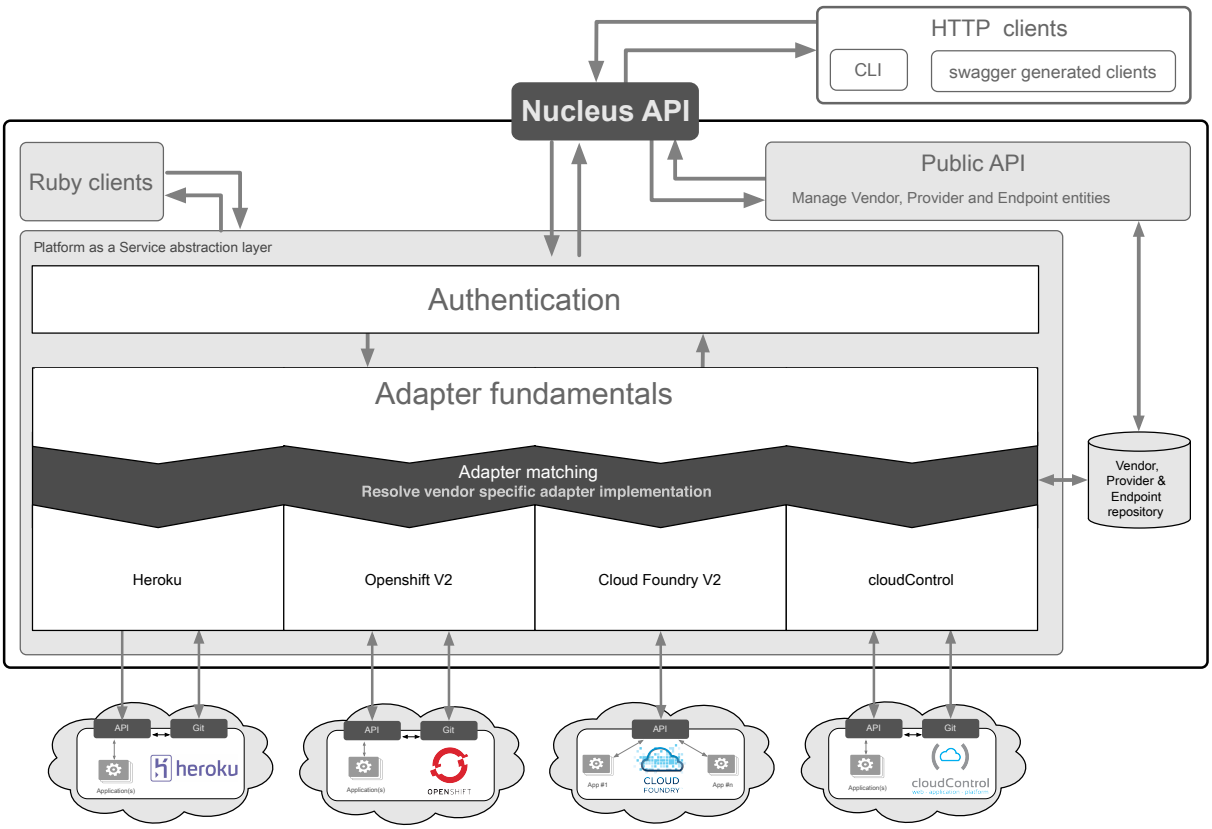


Figure 7: Final architecture of the Nucleus project

directory contains the actual code and application logic. It is divided into two subdirectories, NUCLEUS that contains the actual abstraction layer and NUCLEUS_API that includes only the RESTful web API. NUCLEUS includes further subdirectories, namely ADAPTERS for the platform adapters, CORE to include relevant features and models, EXT with all monkey patched classed and modules, as well as the SCRIPTS directory containing all actions that are invoked when starting or stopping the abstraction layer. The directories of the Nucleus API are: API to include all routes and the entities, EXT to contain the monkey patched classes and modules, IMPORT with the logic to load available adapters and to detect the available API versions, PERSISTENCE to manage the Data Access Objects (DAOs) and object models, RACK_MIDDLEWARE to cover all extensions to the rack server and finally SCRIPTS with the loading, initialization, and shutdown processes. PUBLIC contains the HTML, CSS, and JS files of the interactive swagger-ui[60] API documentation and should therefore be moved to the API gem. All Kwalify[61] compatible YAML schemata reside in the SCHEMAS directory. The schemata are used to validate the required methods inside the adapters and their configuration files. The directory would belong to the CORE after a separation. SPEC contains all files regarding the RSpec tests, as well as the sample application containers, a SSH key to be used during the tests and also the remote endpoint interaction recordings. Currently, it contains tests for both projects. Rake tasks to generate evaluation tables can be found in the TASKS directory. Further markdown files to document the project and its usage reside in the WIKI folder.

[60] *Swagger UI*. URL: `http://swagger.io/swagger-ui/` (Retrieved: June 28, 2015).
[61] *Kwalify: A parser, schema validator and data binding tool for YAML and JSON*. URL: `http://www.kuwata-lab.com/kwalify/` (Retrieved: June 28, 2015).

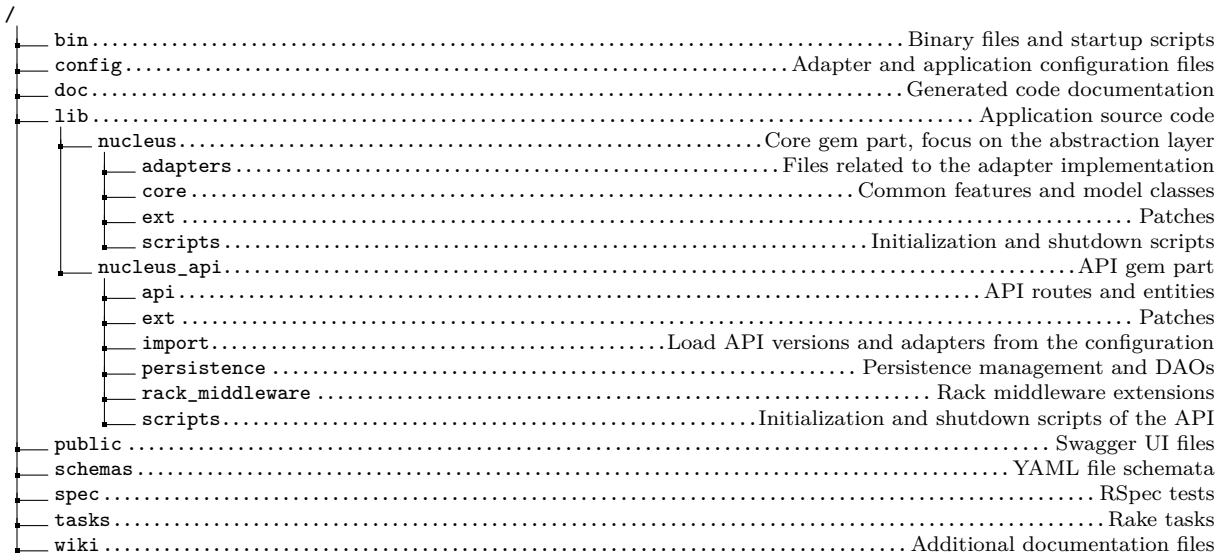


Figure 8: Nucleus' file system project structure

4.3 Initialization

The Nucleus API requires multiple actions to be executed before the web server is ready to serve user requests. In general, the application is started via the provided `nucleus` binary file or the commonly used `rackup` command, which both require the `config.ru` file of the API project as input parameter. The `config.ru` file of the Nucleus API is shown in Code Listing 8. It reveals the API's initialization order, in which at first the global configuration is made available, after which the application is loaded and initialized, before finally the rack based middleware is applied and the application can be run.

Listing 8: Nucleus' config.ru Rack server startup file

```
1  # Setup bundler compatibility, according to: http://bundler.io/v1.9/rationale.html
2  require 'rubygems'
3  require 'bundler/setup'
4  # Load configuration
5  require 'nucleus/scripts/setup_config'
6  # Load application
7  require 'nucleus_api/scripts/load_api'
8  # Initialize the application
9  require 'nucleus_api/scripts/initialize_api'
10 # Initialize the Rack environment
11 require 'nucleus_api/scripts/rack_application'
12 # finally start the application
13 run Nucleus::API::Rack.app
```

Whereas the `load_api.rb` file only loads the dependencies of the application, a closer look at the `initialize_api.rb` file, which is shown in Code Listing 9, points out that the API initialization calls a multiple other scripts. First, the shutdown hooks are registered so that the created SSH files are deleted and the database can be wiped, as illustrated in Code Listing 10. By using the `at_exit` command, it is made sure that the block is executed immediately before the application is shut down.

Listing 9: Nucleus API initialization script

```
1 begin
2   # Shutdown hook to cleanup the API
3   require 'nucleus_api/scripts/shutdown_api'
4
```

```
5   # include the configuration of the project to overwrite the home dir config
6   project_dir_config = '../../../config/nucleus_config.rb'
7   if File.exist?(File.expand_path(project_dir_config, File.dirname(__FILE__)))
8     puts "Applying configuration from: #{File.expand_path(project_dir_config, File.dirname(
          __FILE__))}"
9     require_relative project_dir_config
10  end
11
12  # now load the configuration values
13  require 'nucleus/scripts/initialize_config_defaults'
14
15  require 'nucleus_api/scripts/initialize_api_customizations'
16
17  require 'nucleus_api/scripts/initialize_daos'
18
19  # finalize so that the configuration is locked
20  require 'nucleus/scripts/finalize'
21 rescue Nucleus::StartupError => e
22  log.error "Nucleus API startup failed (#{e.exit_code}), exit now"
23  exit e.exit_code
24 end
```

Listing 10: Nucleus shutdown hook

```
1  # Implement API shutdown actions, tidy up the DB
2  at_exit do
3    puts '------------------------------------------------', ''
4    puts 'Cleaning up the API...'
5
6    # delete the SSHHandler generated files
7    puts ' ... delete SSH files ... '
8    nucleus_config.ssh.handler.cleanup if nucleus_config.key?(:ssh) && nucleus_config.ssh.key?(:handler
         )
9
10   if !nucleus_config.db.key?(:delete_on_shutdown) || nucleus_config.db.delete_on_shutdown
11     if File.exist?(nucleus_config.db.path) && File.directory?(nucleus_config.db.path)
12       FileUtils.rm_rf(nucleus_config.db.path)
13     end
14     puts ' ... DB store successfully deleted' unless File.exist?(nucleus_config.db.path)
15   end
16   puts ' ... done!'
17 end
```

Next, the project's configuration is applied in the `initialize_api.rb` file. As there are multiple options how to specify Nucleus' configuration, details are presented in Section 4.11.2. The config initialization makes sure a reasonable default value is applied to all configuration options. API customizations are adaptions of the default Grape Middleware, in this case the customized HTTP Basic authentication strategy. Initializing the DAOs populates the database with all available vendor, provider, and endpoint entities. To populate the entities, the information is taken from the adapter configuration files. More information about these configuration files can be found in Section 4.5.3, which explains how to create new platform adapters. With the finalization script, we ensure that the configuration is locked and the initialization phase is completed.

4.4 API Route Setup

Every Grape API route has to inherit the `Grape::API` class. The root node of Nucleus' API is shown in Listing 11. Grape helpers are a convenient way to provide commonly used

functionality to all API routes. All declarations inside the root class will be available to all sub routes and mounted API classes. The first `before` block will therefore be executed in advance to every route and the `rescue_from` block applies to all routes of the Nucleus API, handling the conversion of internal errors to the defined error response object. The `mount` command in line 31 includes the actual operations of the abstraction layer's first version into the API. In case another version V2 is developed and shares most of its functionality with V1, the second version could be mounted before the current version. All operations that would not be available in V2 could then automatically fallback to the definitions of V1. The `RootAPI's` concluding block, `route :any, '*path'`, catches all requests that did not match the definitions of the API's routes. In consequence, the HTTP error 404 would be returned to indicate no matching resource could be found.

Listing 11: Nucleus API root

```
1  module Nucleus
2    module API
3      class RootAPI < Grape::API
4        helpers AdapterHelper
5        helpers AuthHelper
6        helpers DaoHelper
7        helpers ErrorHelper
8        helpers FormProcessingHelper
9        helpers LinkGeneratorHelper
10       helpers LogHelper
11       helpers SharedParamsHelper
12
13       content_type :json, 'application/json'
14       default_format :json
15       default_error_formatter :json
16       format :json
17
18       before do
19         # env is not injected in every method, but the instance var can be
20         @env = env
21       end
22
23       # rescue ALL errors and comply to the error schema
24       rescue_from :all do |e|
25         # [...] HERE WOULD BE THE ERROR HANDLING LOGIC. PLEASE HAVE A LOOK AT
               THE SOURCE CODE FOR MORE INFORMATION
26
27         # send response via Rack (Grape doesn't support error! or entities via :with in rescue)
28         ::Rack::Response.new([API::Models::Error.new(entity).to_json], entity[:status], entity[:
               headers]).finish
29       end
30
31       mount Nucleus::API::V1::Base
32
33       route :any, '*path' do
34         if env['nucleus.invalid.accept.header']
35           to_error(ErrorMessages::INVALID_ACCEPT_HEADER, 'The Accept header does not
                 match to any route. Please make sure the vendor is set to \'nucleus\' and check the
                 version!')
36         else
37           to_error(ErrorMessages::NOT_FOUND, 'Please refer to the API documentation and
                 compare your call with the available resources and actions.')
38         end
39       end
40     end
41   end
42 end
```

Mounted inside the `Nucleus::API::V1::Base` class, the `Auth` API class includes all operations of the API which target foreign platforms and thus require the user's credentials. The authentication block at the top of the class, as shown in Listing 14, protects all routes that are mounted below the block so that they can only be called with valid credentials. The credentials are thereby verified as described in Section 3.4.3 by using the `Nucleus::Adapters::AuthClient` class, which acts similar to an interface to the individual clients, for instance the `OAuth2AuthClient`.

Listing 12: Nucleus API authentication protected routes

```
1  module Nucleus
2    module API
3      module V1
4        class Auth < Grape::API
5          # ... http_basic authentication   ...
6
7          ### Mount all protected routes below
8          mount Nucleus::API::V1::Calls
9          mount Nucleus::API::V1::Regions
10         mount Nucleus::API::V1::Applications
11         mount Nucleus::API::V1::ApplicationData
12         mount Nucleus::API::V1::ApplicationDomains
13         mount Nucleus::API::V1::ApplicationEnvVars
14         mount Nucleus::API::V1::ApplicationLifecycle
15         mount Nucleus::API::V1::ApplicationLogs
16         mount Nucleus::API::V1::ApplicationLogsTail
17         mount Nucleus::API::V1::ApplicationScaling
18         mount Nucleus::API::V1::ApplicationServices
19         mount Nucleus::API::V1::Services
20         mount Nucleus::API::V1::ServicePlans
21       end
22     end
23   end
24 end
```

To illustrate the detailed implementation of some API operations, Listing 13 shows an excerpt of the code blocks that build the operations to manage the application object. The initial `resource` opens a block that dictates at which URL path all the included operations will be available, which is below `endpoints/{endpoint_id}/applications` for this example. An operation's definition consists of three parts, the description to serve the API documentation, the parameters, and the actual implementation. In the documentation, the success and failure methods describe which objects shall be returned if the operation succeeds or fails. The `GET` method to retrieve an application entity, which is shown in lines 8 to 17, takes the parameters of the `application_context` helper context. All defined resources always follow the definitions of the API operations as introduced in Section 3.3. Whereas `GET` can directly call the adapter implementation of the operation, the `POST` verb that is shown in lines 19 to 38 first needs to include the `vendor_specific` parameters. Except for the more complex application logging, all definitions of the API operation routes are comparable to those shown in this example. All calls against the `adapter` transparently resolve the adapter that matches the targeted endpoint. More information about this process is explained in Section 4.5.1.

Listing 13: Nucleus API application routes definition excerpt

```
module Nucleus
  module API
    module V1
      class Applications < Grape::API
        helpers SharedParamsHelper

        resource 'endpoints/:endpoint_id/applications', desc: 'Endpoint\'s applications', swagger: {
            name: 'applications', nested: false } do
          desc 'Get an applications that is registered at the endpoint' do
            success Models::Application
            failure [[200, 'Application retrieved', Models::Application]].concat ErrorResponses.
                standard_responses
          end
          params do
            use :application_context
          end
          get ':application_id' do
            present adapter.application(params[:application_id]), with: Models::Application
          end

          desc 'Create an applications to be registered at the endpoint' do
            success Models::Application
            failure [[201, 'Application created', Models::Application]].concat ErrorResponses.
                standard_responses
          end
          params do
            use :endpoint_id
            # require the keys of the application in the json object 'application'
            requires :application, type: Hash do
              # name is required, but we must use :all to get the presence validator. Name is selected
                  via :using
              requires :all, using: Nucleus::API::Models::Application.documentation.slice(:name)
              requires :runtimes, Nucleus::API::Models::Application.documentation[:runtimes].merge(
                  type: Array[String])
              # everything else is optional
              optional :all, using: Nucleus::API::Models::Application.documentation.slice(:region, :
                  autoscaled)
            end
          end
          post '/' do
            application_params = declared(params, include_missing: false)[:application]
            application_params = application_params.merge(params[:application][:vendor_specific]) if
                params[:application].key?(:vendor_specific)
            present adapter.create_application(application_params), with: Models::Application
          end
        end
      end
    end
  end
end
```

4.5 Adapters

Concerning the realization of the application architecture as visualized in Figure 7, it was decided to benefit from class inheritance and introduce an extensible adapter hierarchy. The resulting hierarchy is shown in Figure 9 and introduces the `BaseAdapter` and `Stub` classes. All adapter fundamentals, for instance authentication caching, common error handling, native endpoint calls, a universal HTTP rest-client, as well as access to even more helper classes

are provided by the `BaseAdapter`. In contrast, a `Stub` does not contain any functionality and is rather used to serve as an interface or skeleton for one specific API version. The `Stub` has to implement all methods of the API version it represents and always return an error to indicate methods that have not been implemented yet. Finally, vendor specific adapters extend the `Stub` class and provide the actual implementations. Adapters can introduce their own private helper methods, but they have to overwrite the `Stub's` public API methods in order to make the operations work. With this setup, all unsupported or not yet implemented operations of the the adapter will automatically return the `Stub's` error and hence guarantee to match the specified error schema.

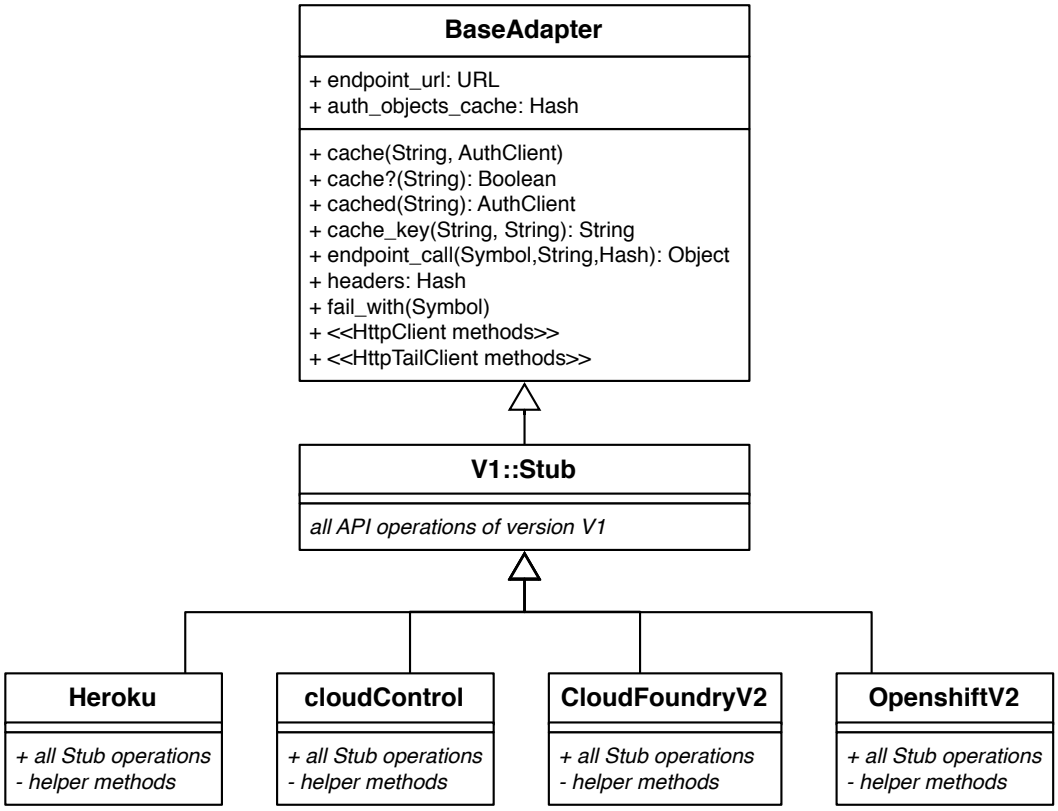


Figure 9: Adapter hierarchy class diagram

Additional classes which can be used by the adapters and are not made available by the `BaseAdapter` can be seen in Figure 10. The `ArchiveConverter` can be utilized to convert archives, for instance from zip to tar.gz, whereas the `FileManager` handles read and write access to the server's file system. Handling Git repositories is the responsibility of the `GitDeployer`, which does not only provide methods to download or deploy files, but also to trigger a new build of the application by changing a hidden marker file. With this setup, vendor specific adapters must not always come up with proprietary solutions, instead they can rely on these common resources.

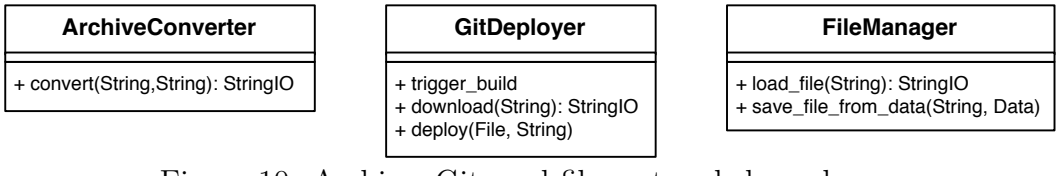


Figure 10: Archive, Git, and file system helper classes

4.5.1 Adapter Matching

Adapter matching describes how to resolve the proper adapter for a particular request. All actions that are needed to resolve the adapter can be found in Listing 14. The listing shows the authentication section of the Nucleus API, which is automatically invoked by all requests that target a protected URL path behind an endpoint. If the request contains credentials, the first step is to load the targeted endpoint object. Using the adapter index object that was introduced in Section 3 with the entities that were generated during the initialization phase, the class of the required adapter can be identified with just one additional database query. Finally, the adapter class is instantiated as shown in line 13 with all additional information given by the index entry. Moreover, the authentication layer of Nucleus does also support re-authentication in case of expired API access tokens. Therefore, the adapter is patched by the `AdapterAuthenticationInductor`, as it is also done when using an adapter via the Ruby gem. By saving the resolved and patched adapter in the `request_cache`, all further operations of this request can resolve the instantiated adapter at any time via the `request_cache`. The approach of the adapter matching is exclusive to the use of the Nucleus API. When acquiring an adapter via the `AdapterResolver` of the Ruby gem, the adapter must always be specified by the user via the vendor's name.

Listing 14: Endpoint authentication and adapter matching code sample

```
1  # defines the authentication for all subsequently mounted routes
2  http_basic(realm: 'Nucleus API Authorization @ %{endpoint_id}',
3             realm_replace: [:endpoint_id]) do |username, password, params, env|
4    # never allow empty username or password
5    return false if username.blank? || password.blank?
6
7    # find a matching endpoint
8    endpoint = load_endpoint(params)
9    # resolve the required adapter
10   index_entry = adapter_dao.get params[:endpoint_id]
11   # use the already secured (https) URL of the index_entry
12   adapter = index_entry.adapter_clazz.new(index_entry.url, endpoint.app_domain, !endpoint.trust)
13
14   # patch the adapter so that calls are wrapped and expect valid authentication
15   AdapterAuthenticationInductor.patch(adapter, env)
16
17   # save info for the current request, no need to retrieve multiple times
18   request_cache.set("#{env['HTTP_X_REQUEST_ID']}.adapter", adapter)
19   request_cache.set("#{env['HTTP_X_REQUEST_ID']}.endpoint", endpoint)
20
21   cache_key = adapter.cache_key(username, password)
22   # THREAD HACK to work with deferred tasks (log tailing), cache auth key
23   request_cache.set("#{env['HTTP_X_REQUEST_ID']}.auth.cache.key", cache_key)
24
25   unless adapter.cache?(cache_key)
26     # no auth object available, perform authentication first
27     auth_object = adapter.auth_client
28     # throws an error if the authentication failed
29     auth_object.authenticate(username, password)
30     # cache the auth object so it does not have to be retrieved per request
31     adapter.cache(cache_key, auth_object)
32   end
33   # auth passed
34   true
35 end
```

4.5.2 Adapter Compatibility

Initially, the API operations of our abstraction layer were introduced in Chapter 2. Later on, the previous design chapter explained the object and operation mappings that must be obeyed in detail. Nevertheless, Table 3 also highlighted incompatibilities which cannot be fixed by mappings as basic operations are missing on some platforms. Focusing on these issues, Table 31 visualizes the compatibility of the final vendor adapters against the definitions of the adapter stub and therefore the abstraction layer. The auto-generated table does only reveal whether an operation is somewhat supported, but does not consider if the operation is only partially compatible, as for instance the service plan mapping on Cloud Foundry. Even so, it provides a good indication where problems might occur.

Adapter compatibility	cloudControl	Cloud Foundry v2	Heroku	OpenShift v2
auth_client	✓	✓	✓	✓
regions	✓	✓	✓	✓
region	✓	✓	✓	✓
applications	✓	✓	✓	✓
application	✓	✓	✓	✓
create_application	✓	✓	✓	✓
update_application	✗	✓	✓	✗
delete_application	✓	✓	✓	✓
domains	✓	✓	✓	✓
domain	✓	✓	✓	✓
create_domain	✓	✓	✓	✓
delete_domain	✓	✓	✓	✓
env_vars	✓	✓	✓	✓
env_var	✓	✓	✓	✓
create_env_var	✓	✓	✓	✓
update_env_var	✓	✓	✓	✓
delete_env_var	✓	✓	✓	✓
start	✓	✓	✓	✓
stop	✗	✓	✓	✓
restart	✗	✓	✓	✓
deploy	✓	✓	✓	✓
rebuild	✓	✓	✓	✓
download	✓	✓	✓	✓
scale	✓	✓	✓	✓
log?	✓	✓	✓	✓
logs	✓	✓	✓	✓
log_entries	✓	✓	✓	✓
tail	✓	✓	✓	✗
services	✓	✓	✓	✓
service	✓	✓	✓	✓
service_plans	✓	✓	✓	✓
service_plan	✓	✓	✓	✓
installed_services	✓	✓	✓	✓
installed_service	✓	✓	✓	✓
add_service	✓	✓	✓	✓
change_service	✓	✓	✓	✗
remove_service	✓	✓	✓	✓
Supported methods	34	37	37	34
Supported degree	91.9 %	100 %	100 %	91.9 %

Table 31: List of API operations that are supported by Nucleus per vendor

Heroku and Cloud Foundry both support all 37 defined operations of the adapter stub, whereas OpenShift currently only supports 34 of the operations. However, the missing tail log operation could be added to this count as it can be realized, but has not been implemented in this prototype. In summary, only the application update and service change are

not supported. On OpenShift, the service change is not needed as long as service plans are not actively used by the platform. The immutability of application objects, which prevents application updates, is one core principle conflicting with Heroku and Cloud Foundry. cloudControl shares OpenShift's idea of immutable applications, on the one hand to prevent runtime changes, on the other hand as the name of an application is also used as identifier and therefore must not be changed. The remaining two unsupported operations are all caused by cloudControl's divergent lifecycle understanding. Once the application has been deployed and started, there is no way it can be stopped without undeploying the application data or deleting the services and their content.

Despite minor conflicts, the compatibility table shows that the most relevant operations could be brought in line. All remaining issues are inflicted by differences in the general understanding of an application's lifecycle and the chosen system architecture.

4.5.3 Adapter Implementation

PaaS vendors are all represented by a unique adapter to orchestrate the communication between Nucleus and the endpoints of the vendor's platform. The implementation of those adapters can be separated into a few steps, which are going to be introduced in the later of this section.

Adapter Configuration File

Each adapter has its own `.yml` configuration file that is located in the `config/adapters` directory of the Nucleus project. The configuration file must be named exactly like the vendor's identifier, so that the file for the current implementation of OpenShift would be called `config/adapters/openshift_v2.yml`. Its default content can be seen in Listing 15. The configuration file describes all known objects, starting with the vendor on the first level of the YAML file. For a vendor, there can then be an arbitrary number of providers, which themselves can possess any number of endpoints. Endpoints must be provided with the `url` of the endpoint's API and can also take some optional attributes, such as the `app_domain` and `trust`, which were already introduced along with the API operations in Chapter 3.3.1.

Listing 15: OpenShift V2 adapter configuration file in YAML syntax

```
---
 name: "OpenShift 2"
 id: "openshift_v2"
 providers:
   -
     name: "OpenShift Online"
     id: "openshift-online"
     endpoints:
       -
         name: "OpenShift Online"
         id: "openshift-online"
         url: "openshift.redhat.com/broker/rest"
```

The objects will be made available at the API via the IDs given in the adapter configuration files. Different platform versions shall therefore be distinguished by appending `_v{version}`

to the ID, as shown with OpenShift version 2 in this example. Provider and endpoint objects can also be modified at runtime, whereas the vendor objects that represent the adapter implementation are write-protected and can only be changed via the configuration files.

Adapter Implementation Files

Before starting the implementation of an adapter, it must be decided for which version of the abstraction layer the platform's adapter shall be developed. All adapter files then have to be created in a directory matching the following path:
`app/adapters/{API_VERSION}/{vendor_id}/{vendor_id}.rb`
It is very important to make sure the `vendor_id` is equal to the identifier that was assigned to the vendor in the adapter configuration file, as otherwise the adapter cannot be resolved at runtime. The module and class naming, as it is shown in Listing 16 must also be regarded carefully. Each adapter must inherit from the `Stub` adapter matching the chosen API version.

Listing 16: Adapter class and module namespace

```
module Nucleus
  module Adapters
    module {API_VERSION}
      class {VENDOR_ID} < Stub
        # implemented methods
      end
    end
  end
end
```

To enhance the project's code quality and modularity, as well as to strengthen the separation of concerns, the adapter should be divided into multiple smaller modules. For instance, Cloud Foundry's adapter is distributed to 14 files. As soon as the adapter is defined with its configuration file, inherits the `Stub` and is placed in the correct location, it should already be available in the API. However, all calls would fail and return an error with status code 501 to indicate the missing implementation.

To implement the adapter's logic, one requires the expected behavior of the adapter methods, including information about the method parameters and the expected response object. This documentation can not only be taken from the `Stub` adapter, but is in most cases even identical to the API operation definitions of Section 3.3. Regarding the authentication method, Nucleus already offers four different authentication approaches which all could be used in new adapters. If the targeted platform does not directly support a certain operation, it is legit to apply minor workarounds by means of achieving the goal. Workarounds that were applied in the included version of the prototype are, e.g., the domain management on Cloud Foundry or the lifecycle handling on Heroku. If the operation cannot be supported at all, as for instance cloudControl does not support to STOP applications, the method shall not be present in the adapter so that the `Stub` class can provide a common error response. More aspects regarding the implementation of adapter tests can be found in the upcoming Section 4.9.

4.6 Git Deployment and Repository Authentication

Git is used within Nucleus to manage application data on cloudControl, Cloud Foundry, and Heroku. Especially the `deploy`, `download` and even the `rebuild` methods of those platforms depend on Git interactions.

In the operations mapping Section 3.4.2, it was described which Git actions need to be invoked by the operations. The Ruby gem `Git`[62] facilitates the execution of Git commands inside Ruby code. All Git actions are bundled in the `Nucleus::Adapters::GitDeployer` and `Nucleus::Adapters::GitRepoAnalyzer` classes, which can be called directly from within the platform adapters. To illustrate this procedure, Listing 17 contains the `download` method of Heroku's adapter. Whereas the first lines only perform assertions and fetch the required information, the task itself starts at line seven with registering the SSH key which is required to allow the access to the Git repository. Thereafter, the `GitDeployer` gets instantiated and the download is invoked.

Listing 17: Heroku's download adapter method

```
1  def download(application_id, compression_format)
2    app = get("/apps/#{application_id}").body
3    if application_state(app) == Enums::ApplicationStates::CREATED
4      fail Errors::SemanticAdapterRequestError, 'Application must be deployed before data can be
           downloaded'
5    end
6    repo_name = "nucleus.app.repo.heroku.download.#{application_id}.#{SecureRandom.uuid}"
7    with_ssh_key do
8      GitDeployer.new(repo_name, app[:git_url], nil).download(compression_format, true)
9    end
10 end
```

Triggering a `rebuild` via Git is achieved by manipulating a custom marker file called `nucleus-rebuild-trigger`. Code Listing 18 visualizes this part of the `GitDeployer`.

Listing 18: Trigger rebuild method of the GitDeployer

```
1  def trigger_build
2    push_repository_changes do |repo_dir|
3      build_trigger_file = File.join(repo_dir, 'nucleus-rebuild-trigger')
4      current_md5 = File.exist?(build_trigger_file) ? Digest::MD5.file(build_trigger_file).hexdigest :
           nil
5      data = StringIO.new("Nucleus rebuild, triggered at #{Time.now}")
6      FileManager.save_file_from_data(build_trigger_file, data, false, current_md5)
7    end
8    nil
9  end
```

Despite the simplicity of the previously shown steps, the largest challenge for all Git actions is to gain access to the application's Git repository. Usually, the Git commands utilize a SSH certificate based authentication and attempt to gain access with all private keys available on the system. Nevertheless, assuming that some certificates are available on the platform, especially ones without a passphrase protection, using them would make the whole process quite fragile. Nucleus would always have to validate the certificates in terms of their presence, their type, and passphrase protection. Only then, they could be used at all. In consequence, it was decided that the SSH key that will be used shall either be provided by the user via the application's configuration, or shall be generated during the application's startup phase.

[62] *Git Library for Ruby.* URL: `https://github.com/schacon/ruby-git` (Retrieved: July 2, 2015).

Those steps are performed within the `Nucleus::SSHHandler`. Most important, Git must also be told to use the chosen private key for its authentication attempts. This can be achieved via SSH agents wrapping the actual Git commands and are applied by the `Git` gem. However, the commands must be available as script file on the local disk, wherefore the code that is shown in Listing 19 writes the file to a temporary location of the active file system. Both scripts, a UNIX and Windows version, point to the private key, accept all foreign hosts, disable the host check which would require manual confirmation and finally execute the original Git commands with all its parameters.

Listing 19: SSH agent creation of the SSHHandler to use a custom private key for Git

```
def create_agent
  # use uuid so that more than one instance can run at the same time
  @agent_file = File.expand_path(File.join(Dir.tmpdir, 'nucleus', 'ssh', 'agent', SecureRandom.uuid
      ))
  # windows requires the extension, otherwise git complains that it can't spawn such a file
  @agent_file = "#{@agent_file}.bat" if OS.windows?
  FileUtils.mkdir_p(File.dirname(@agent_file))

  if OS.unix?
    File.write(@agent_file, "ssh -i #{@key_file} -o UserKnownHostsFile=/dev/null -o
        StrictHostKeyChecking=no $*")
    FileUtils.chmod(0700, @agent_file)
  else
    File.write(@agent_file, "@echo off\r\nssh -i #{@key_file} -o UserKnownHostsFile=NUL "\
      '-o StrictHostKeyChecking=no %*')
  end
end
```

4.7 Exception Handling

All exceptions which could not be resolved internally are finally returned to the user with the error message format as it was described in Section 3.2.3. Errors are processed within the `rescue_from :all` block of the `RootAPI` class. This error handling logic is presented in Listing 20 and shows that all errors are also passed to Nucleus' logging system so that issues can be detected and analyzed. Grape itself is also eligible to raise errors, for instance if the parameter or header validation failed. If an exception shall be rescued that was not expected, the HTTP 500 error is returned and a prioritized error logging statement shall be made. Errors of Grape, as well as errors which are expected to be raised by the adapters of Nucleus, are converted to API conform response message. Three types of exceptions have to be distinguished within Nucleus, which are all introduced in the remainder of this section.

Listing 20: Nucleus error handling

```
rescue_from :all do |e|
  env['api.endpoint'].log.debug e.to_s
  e.backtrace.each { |line| env['api.endpoint'].log.debug line }

  if e.is_a?(Errors::ApiError) || e.is_a?(Nucleus::Errors::AdapterError)
    entity = env['api.endpoint'].build_error_entity(e.ui_error, e.message)
  elsif e.is_a?(Grape::Exceptions::ValidationErrors) || e.is_a?(Grape::Exceptions::
      InvalidMessageBody)
    entity = env['api.endpoint'].build_error_entity(ErrorMessages::BAD_REQUEST_VALIDATION,
        e.message)
  elsif e.is_a?(Grape::Exceptions::InvalidAcceptHeader)
    entity = env['api.endpoint'].build_error_entity(ErrorMessages::INVALID_ACCEPT_HEADER, e
        .message, e.headers)
```

```
11      env['nucleus.invalid.accept.header'] = true
12    else
13      entity = env['api.endpoint'].build_error_entity(ErrorMessages::RESCUED, "Rescued from #{e.
            class.name}. Could you please report this bug?")
14      env['api.endpoint'].log.error("API error via Rack: #{entity[:status]} - #{e.message} (#{e.class
            }) in #{e.backtrace.first}:")
15    end
16
17    # send response via Rack, since Grape does not support error! or entities via :with in the rescue
          block
18    ::Rack::Response.new([Models::Error.new(entity).to_json], entity[:status], entity[:headers]).finish
19  end
```

Startup Errors

Startup errors are internal errors that appear during the initialization of the application and usually prevent a successful start. Currently, there are two specific startup errors, but sometimes the Ruby `StandardError` is raised, too.

AmbiguousAdapterError raised if more than one adapter file was found for an adapter configuration. File naming must be checked!

StartupError prevents the start of the Nucleus API. For instance caused by bad configuration, e.g., if the specified SSH certificates could not be found.

API Errors

An `ApiError` is caused by the user's request, e.g., if a parameter's content is invalid. API errors are only raised by Nucleus. An instantiated error contains all information that is required to build the JSON response message within the `rescue_from` block of the `RootAPI`.

ApplicationArchiveError HTTP status code 400, raised if the provided data could not be extracted, e.g., if it is corrupted.

ResourceNotFoundError 404, is raised if a vendor, provider or endpoint resource could not be found in the database.

Adapter Errors

An `AdapterError` is caused by errors raised within Nucleus or the endpoint's response. Likewise to an API error, it contains all required information to build the formatted error response message.

AdapterRequestError HTTP status code 400, raised if the request is invalid, usually due to missing or semantically invalid parameters that could not be processed by the syntactical validation of Grape.

EndpointAuthenticationError 401, originally raised by the endpoint if the authentication attempt failed, e.g., due to bad credentials.

AdapterResourceNotFoundError 404, is raised if one of the referenced objects could not be found on the endpoint.

SemanticAdapterRequestError 422, is raised if not all conditions are met to invoke the operation. Can originate from Nucleus or the endpoints.

PlatformSpecificSemanticError 422, is raised if platform specific conditions are violated, for instance if cloudControl was not yet provided with a billing address that is required for some operations. Can originate from Nucleus and the endpoints.

UnknownAdapterCallError 500, is raised by Nucleus only. Hints at implementation errors, e.g., when an adapter is outdated.

AdapterMissingImplementationError 501, raised by Nucleus if an operation has not (yet) been implemented for the current adapter.

PlatformUnavailableError 503, raised by endpoints if they are temporarily not available, for instance during maintenance.

PlatformTimeoutError 504, raised by endpoints if they experienced an internal timeout on asynchronous operations.

4.8 Authenticated API Requests

Summarizing the previous sections, this section demonstrates the internal workflow that is taken when authenticated endpoint requests are executed. The complete process is visualized in the activity diagram of Figure 11.

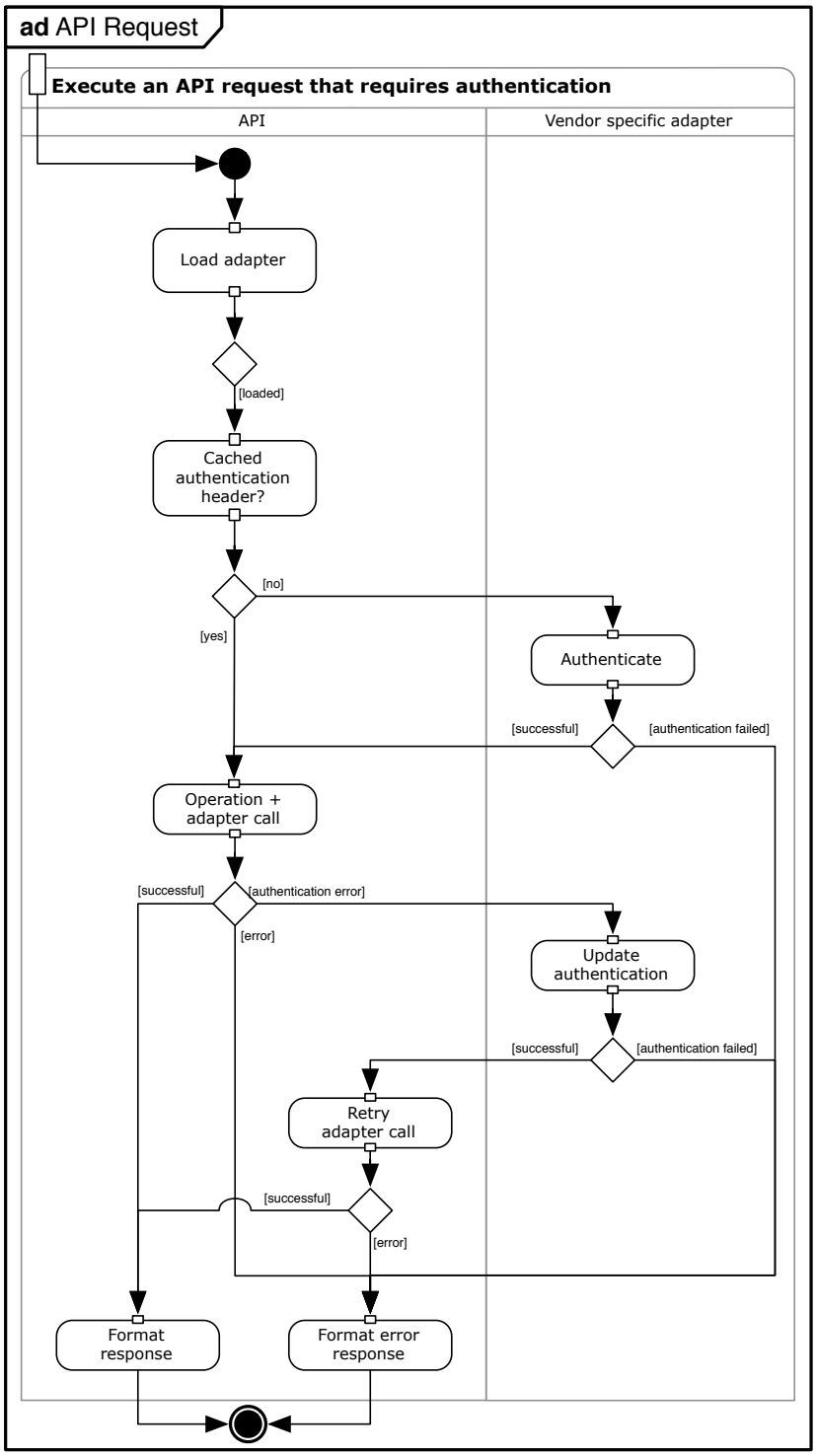


Figure 11: Activity diagram showing the steps of authenticated API requests

As previously explained in Section 4.5.1, the first step of the processing is to load the adapter that is capable of handling the targeted endpoint. Thereafter, it is checked if an API token

has already been retrieved and did not expire yet. This process is also shown in Section 4.5.1, especially in Listing 14. If the cache does not contain a valid token, the authentication is attempted by using the adapter's authentication client. Assuming that the authentication fails, a formatted error response is returned. When the authentication succeeds, the process is resumed as if the credentials were available before. If a token is already stored in the cache or a prior authentication succeeded, the next step is to execute the API operation with its adapter call. Unless it fails, the formatted response is returned and the request finished. API and Adapter errors are formatted as an error response and finish the request as well. However, if an authentication error is returned by the adapter call, for instance if the token has been revoked, Nucleus attempts to acquire a new token. When this authentication update is successful, the adapter call is retried and formatted according to whether the call fails or succeeds. Otherwise, the error response is formatted to indicate the expired credentials.

4.9 Automated Tests

In statically typed programming languages, for instance C++ or Java, the compiler will already find a good share of errors as it acts as first evaluation instance. However, in dynamic languages as Ruby, there is no compiler to perform these checks. Under these circumstances, a high test coverage with unit and integration tests is even more important.

Nucleus features three distinct types of tests. In addition to the commonly used unit and integration tests, Nucleus' adapter tests can be regarded as complete system test. Adapter tests call the API of Nucleus, which then calls the PaaS endpoint to perform the necessary tasks, before the response is generated and can be evaluated against the tests expectations. Within the `spec` folder of the project, all RSpec files, helper classes and required binary files of the application's tests can be found. All files of the three test types are available in dedicated subdirectories. Files directly residing inside the `spec` folder are required by all tests, e.g., the general spec helper which initializes the application and includes dependencies to calculate the test coverage.

A list of all additional dependencies needed to make the tests work is presented in Listing 32. `Airborne` is used to facilitate the calls against the running API, which therefore simulates the functionality of a rack server. `Rubocop` performs static checks and validates the written code against a rule set of best practices. It prevents changes from being committed if any of the checked classes violates these rules. `VCR`[63] is the most important gem in regards to the adapter tests. It allows to record HTTP interactions of external systems which can also be replayed and thus enable dramatically faster, deterministic, and accurate tests. Within the test of Nucleus, VCR records and replays all interactions with the platforms' APIs, so that the test duration is reduced by more than an hour, but even more important, it allows to run the adapter tests in Continuous Integration (CI) systems without having to provide the endpoint user credentials publicly.

[63] *VCR documentation*. URL: `http://www.relishapp.com/vcr/vcr/docs` (Retrieved: July 6, 2015).

Gem	Usage description
airborne	RSpec driven API testing framework
factory_girl	Factory to generate test objects
faker	Generate fake data
memfs	Fake file system
rspec-wait	Wait for conditions in RSpec
rubocop	Ruby code style checking tool to enforce the community-driven Ruby Style Guide
vcr	Record and replay your test suite's HTTP interactions
webmock	Stubbing HTTP requests and expectations on HTTP requests

Table 32: Gem dependencies for development and tests

4.9.1 Adapter Tests

Adapter tests are complete system tests that include communication with third party systems, in our case requests for the execution of operations on the targeted endpoint. Within the adapter tests, most of the abstraction layer's behavior is tested, so that the received feedback is a good measure to indicate the compatibility of the platforms and evaluate which parts are not yet properly implemented.

Listing 28 in Appendix B presents the basic template of a file used to describe all tests that should be run for an adapter. This adapter test definition should be placed beneath `spec/adapter/v1/{VENDOR_ID}/{VENDOR_ID}_spec.rb` to be included in the automated test runs. If a certain functionality should not be provided by a platform, those tests can be marked as unsupported in the leading `before :all` block. The `before do |example|` block takes care of skipping unsupported features and loading a new adapter instance for each test. Thereafter, the actual tests are included. All tests are written and included as shared examples, a feature of RSpec which allows to not only place the tests into dedicated files, but also enables their reuse. The most extend of the adapter verifications is included via the `compliant adapter with valid credentials` shared example. It simulates the lifecycle of two applications, creates, modifies, collects, and deletes various entities to cover all API operations at least once.

The sequence diagram in Figure 12 illustrates the test execution process of a single adapter test. Commencing the test, the environment is initialized via all included test helpers, which takes care of loading the dependencies and their configuration. In the next step, a `VCR` cassette is inserted that will include the set of HTTP interactions of a test run. If the test execution shall be recorded, the recording is started, otherwise the secret data of the previous recording is made available to the test execution. Once the recording is setup, the actual RSpec test is executed together with its validations. Regardless of the recording mode and the test result, the cassette gets ejected at the end of the test, before RSpec returns the test result to indicate if the validations passed or failed. However, if the recording was enabled, three more steps are required before ejecting the cassette. The first step is to stop the recorder. If and only if there were no errors and failed validations during the test execution, the recorded data is anonymized to strip private data, e.g., credentials, before finally persisting the data so that it is available for test replays. Any error or test violation prevents the recorded cassette from getting persisted.

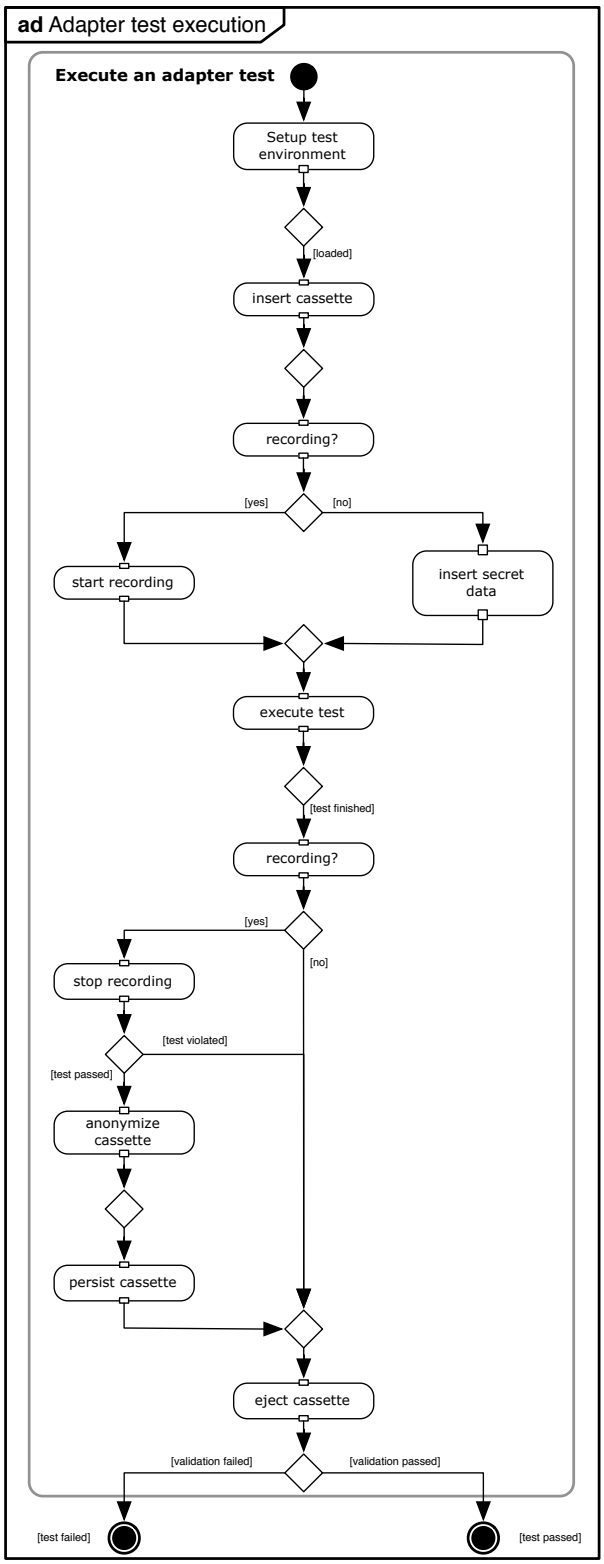


Figure 12: Sequence diagram showing the test execution process

In order to validate whether Nucleus is working properly and can be used for the management and deployment operations, we identified a need for a demo application which can be deployed on all supported platforms during the adapter tests without requiring further technical adaptations. After a brief analysis of the supported vendors and their runtime languages, we decided to use Node.js as runtime language. Node.js is not only offered by all of the chosen vendors, but most PaaS vendors in general. The application itself is called Word

Finder[64]. Word Finder is a minimalistic open-source application that allows to find words containing specific characters. It has only minimal technical requirements and does not require a database connection or any other third party service. Manual attempts to deploy the application on the chosen platforms succeeded without any issues in most instances and only the deployment on OpenShift failed. A comparison of OpenShift's Node.js cartridge documentation[65] and the source code of the Word Finder application immediately revealed the problem, which is highlighted in Code Listing 21.

Listing 21: Wordfinder sample application, original IP and port configuration

```
1 app.listen(process.env.PORT || config.port);
```

During the startup process, the application will not be bound to a specific IP address, but only to a port that can be specified via the `PORT` environment variable or the application's configuration. Based on this information, the startup process was modified to take advantage of the OpenShift specific environment variables and fallback to commonly used environment variables, the application's configuration or a default value. This solution can be seen in Code Listing 22.

Listing 22: Wordfinder sample application, fixed IP and port configuration

```
1 port = process.env.OPENSHIFT_NODEJS_PORT || process.env.VCAP_APP_PORT ||
      process.env.PORT || config.port;
2 ip = process.env.OPENSHIFT_NODEJS_IP || "0.0.0.0";
3 app.listen(port, ip);
```

With these modifications the application can be deployed on all four platforms during the automated adapter tests.

4.10 Documentation

Nucleus' extensive documentation is divided into three major groups: basic comments in the code, the API documentation, and the generated code documentation. Besides those documents, additional and rather informal information regarding the general application setup and usage can also be found in the markdown files of the developed project.

The generated HTML documentation of the implementation code is based on the `YARD`[66] gem. Currently, nearly 70% of the modules, classes, constants and methods are properly documented. The documentation can be generated via the `bundle exec rake doc` command.

A vital part of fostering API adoption is to provide a useful and easy to understand documentation. With the help of the `grape-swagger`[67] gem, an extension to the Grape API, additional API resources that describe the complete Nucleus API in version 1.2 of the standardized swagger[68] specification are made available. The specification is auto-generated at

[64] *Word Finder - Node.js application.* URL: `https://github.com/amirrajan/word-finder/` (Retrieved: April 29, 2015).
[65] *OpenShift - Node.js Overview.* URL: `https://developers.openshift.com/en/node-js-environment-variables.html` (Retrieved: April 30, 2015).
[66] *YARD: A Ruby documentation generation tool.* URL: `http://yardoc.org/` (Retrieved: July 9, 2015).
[67] *grape-swagger: Swagger compliant documentation of your grape API.* URL: `https://github.com/tim-vandecasteele/grape-swagger` (Retrieved: June 25, 2015).
[68] *Swagger API representation.* URL: `http://swagger.io` (Retrieved: June 28, 2015).

the start of the application and therefore only available at runtime. Swagger allows to create an interactive documentation, generate API clients for many programming languages and also enables the discoverability of the API. The interactive documentation is a HTML and JS application known as Swagger UI[69], which consumes the JSON resources that describe the actual API. As the Swagger UI project is embedded in the project and gets automatically hosted by the rack server, it can be opened in any browser at the */docs* URL path. Screenshots that hint at the capabilities of the chosen solution can be found in Appendix C. Figure 13 shows the available operation groups. All operation groups can be expanded to show the operation entries similar to Figure 14. The operations can also be expanded, showing their full specification as presented in Figure 16 for a `GET` request and Figure 15 for a `PATCH` request. Request parameters, response objects and all allowed response status codes are explained in detail within the specification. When using the interactive possibilities of the UI, parameters that are needed to execute a request can be provided in the available input fields.

4.11 Usage

Nucleus can be installed and used without any major system requirements. The following sections highlight the most important aspects which have to be regarded. Section 4.11.1 presents prerequisites before installing Nucleus, whereas the available configuration options are explained in Section 4.11.2. The usage of the Nucleus gem is introduced in Section 4.11.3. Nucleus' language independent RESTful API is presented in the concluding Section 4.11.4.

4.11.1 System Requirements

Most important, Nucleus requires a minimum Ruby version of 1.9.3. Nucleus is developed and actively tested with the default MRI Ruby interpreter. Furthermore, Nucleus requires a few programs to be available on the system's PATH, namely `git` and `ssh`. Whereas it is sufficient for programs to be installed on UNIX via the system's package manager, for instance apt-get, additional steps must be considered when using Microsoft Windows. On Windows, both executable files should be located in the `Git/bin` installation directory of msysGit. Nucleus is verified to work with msysGit and its included version of OpenSSH. Any other alternatives, e.g., PuTTY, are untested. More operating system specific issues and all known fixes are listed in the project's README file.

4.11.2 Configuration

Nucleus allows a number of options to be changed inside configuration files. Moreover, there are three possibilities how a configuration file can be made available to Nucleus. All three options can also contain only partial information.

The first and least significant option is to provide a system wide configuration file in the home directory of the user account that invokes the Nucleus gem or the API. If available, the file is loaded from `nucleus/nucleus_config.rb` on Windows systems, and

[69] *Swagger UI.* URL: `http://swagger.io/swagger-ui/` (Retrieved: June 28, 2015).

`.nucleus/nucleus_config.rb` on all UNIX systems, relative to the home directory. Overwriting the system wide configuration, the Nucleus gem can be provided with its own configuration file. The `nucleus_config.rb` file must be placed in the `config` directory of the gem. If the API is used, the third configuration option is to place the `nucleus_config.rb` file in the `config` directory of the API, which then overwrites both previously introduced configuration locations.

The default configuration file, in which all options are commented out, can be seen in the Listing 23. Options one and two allow to change the logging severity level as well as the directory into which the files shall be written. This allows, for instance, that the API can be installed as system service and log to the `/var/log/nucleus` directory as expected by most Linux systems. The next four options focus on the persistence of the public API entities. The `path` option allows to specify the location where files of the storage backend can be written to. Combined with a `delete_on_shutdown` option which is set to FALSE, the API can easily be made persistent to make sure custom objects are retained even beyond restarts. As discussed in Section 4.6, a private SSH key is needed to enable the application data deployment via Git. The location of a personal SSH key can be configured by the `nucleus_config.ssh.custom_key` property. The provided key must be created with OpenSSH, use ssh-rsa and must not be protected with a passphrase. By default, a new key will be created upon starting Nucleus.

Listing 23: Nucleus' default configuration file

```
# [optional] The available levels are: FATAL, ERROR, WARN, INFO, DEBUG
# Defaults to: Logger::Severity::WARN
# nucleus_config.logging.level = Logger::Severity::WARN

# [optional] Logging directory
# Defaults to: File.expand_path(File.join(File.dirname(__FILE__), '..', 'log'))
# nucleus_config.logging.path = File.expand_path(File.join(File.dirname(__FILE__), '..', 'log'))

# [optional] Options to start the backend.
# See http://www.rubydoc.info/gems/moneta/Moneta/Adapters for valid options on the chosen adapter.
# Defaults to: {}
# nucleus_config.db.backend_options = {}

# [optional] Please specify the DB directory if you plan to use a file storage.
# Defaults to: a temporary directory.
# nucleus_config.db.path = '/Users/cmr/Documents/workspace-rubymine/nucleus/store/'

# [optional] If true, the DB will be deleted when the server is being closed.
# Defaults to: true
# nucleus_config.db.delete_on_shutdown = false

# [optional, requires 'nucleus_config.db.path'] If true, the DB will be initialized with default values,
# which may partially override previously persisted entities.
# False keeps the changes that were applied during runtime.
# Defaults to: false
# nucleus_config.db.override = false

# [optional] Specify the location of a private key (ssh-rsa, OpenSSH) that shall be used for Git actions.
# E.g. /home/myusername/.ssh/id_rsa
# If set to false, Nucleus will use its own private key (config/nucleus_git_key.pem) to authenticate all Git actions.
# Defaults to: nil
# nucleus_config.ssh.custom_key = nil
```

4.11.3 Ruby Gem

Nucleus can also be embedded in another application as Ruby gem. Listing 24 shows the necessary steps. First, the dependency on the Nucleus gem should be added in the gemspec or Gemfile of the project. It must then be installed, either via Bundler or the `gem install` command. Step three is to require Nucleus in the application, which then automatically triggers Nucleus' internal initialization procedure.

Listing 24: Include the Nucleus gem in another Ruby application

```
# 1a) Add dependency in Gemfile
gem 'nucleus'

# 1b) Add dependency in gemspec
 specification .add_runtime_dependency 'nucleus'

# 2a) Install the dependency via Bundler
$ bundle install

# 2b) Manually install the dependency
$ gem install nucleus

# 3) Require the gem in your application
require 'nucleus'
```

Once the gem is installed and required, its configuration can also be adapted from within the new application. All values, which are explicitly described in Section 4.11.2, are accessible via the global `nucleus_config` accessor. The `Nucleus::VersionDetector.api_versions` method call makes it possible to obtain a list of all available versions of the abstraction layer. After a version has been chosen, Listing 25 demonstrates how the gem can be used to invoke further actions. The `Nucleus::AdapterResolver` must always be instantiated with a valid version. As soon as the `AdapterResolver` is initialized, it provides access to all adapters that are compatible with this version of the abstraction layer. Adapters can then be loaded via its `load` method. The `load` method requires the name of the vendor, as well as the URL of the API endpoint and the credentials that shall be used to authorize the API requests. Optionally, it can also be specified to skip the assertion of SSL certificates or provide a custom application domain. By default, the adapter will be populated with the default configuration options that are defined in the vendor's configuration for the selected endpoint. However, if a custom installation is used, e.g., of OpenShift or Cloud Foundry, the `app_domain` option should always be specified as the `web_url` links created by Nucleus would otherwise be malformed. The methods which can be called on the adapter equal the API's operations to the most extend. Solely the log download methods are not available in the adapters. All adapters offer a method to receive the application's log entries instead. The example in Listing 25 shows three sample method calls, which retrieve all available regions of the endpoint, create and finally delete an application.

A complete list of the available methods can be obtained via the documentation of the `Nucleus::Adapters::V1::Stub` adapter. Detailed information about the method's parameters, including the declarations, which fields are required and those that are only optional, can also be taken from the documentation of the RESTful API.

Listing 25: Use the Nucleus gem in another Ruby application

```
available_versions = Nucleus::VersionDetector.api_versions
resolver = Nucleus::AdapterResolver.new('v1')
available_adapters = resolver.adapters
# --> {"cloudcontrol"=>Nucleus::Adapters::V1::CloudControl, "cloud_foundry_v2"=>Nucleus::
    Adapters::V1::CloudFoundryV2, "heroku"=>Nucleus::Adapters::V1::Heroku, "openshift_v2"=>
    Nucleus::Adapters::V1::OpenShiftV2}

adapter = resolver.load('cloudcontrol', 'api.cloudcontrol.com', 'your_username', 'your_password')
# adapter = resolver.load('cloud_foundry_v2', 'api.example.org', 'your_username', 'your_password',
    app_domain: 'apps.example.org', check_ssl: false)

# Show available regions
regions = adapter.regions
# Create our first application
app = adapter.create_application(region: regions[0].id, name: 'myusersfirstapplication', runtimes: ['
    nodejs'])
# And delete the application again ;-)
adapter.delete_application(app[:id])
```

4.11.4 Server

The server mode of Nucleus can be started in multiple ways. The most convenient and preferred solution is to use the provided startup script, which is located at `bin/nucleus` of the project. It comes with a variety of options, which can be seen in Listing 26. All options can be categorized into three groups. The first group concerns the general web server bindings, followed by options to use the API as daemon or background service. Group three allows to enable SSL protection of the API and specify the key and certificate files.

Listing 26: Nucleus' startup script

```
bin/nucleus --help

Usage:
nucleus [options]

Options:
    -r, --hostname HOSTNAME          Bind to HOST address (default: localhost)
    -p, --port PORT                  Use PORT (default: 9292)
    -e, --env ENV                    Environment (default: "development")
    -t, --timeout TIMEOUT            Timeout for single request (default: 30)
    -h, --help

Daemon options:
    -d, --daemon                     Run the server as daemon in the background (default: false)
    -u, --user USER                  User to run daemon as. Use with -d (default: "nobody")
    -g, --group GROUP                Group to run daemon as. Use with -d (default: "nobody")
    -b, --pid PID                    File to store PID (default: tmp/pids/thin.pid)
    -l, --logdir LOGDIR              Directory for log files if run as daemon, defaults to {
        current_directory}/log

SSL options:
    -s, --ssl                        Enable SSL (default: false, use with: ssl-key and ssl-cert)
    -k, --ssl-key KEY                SSL key file to use (use with: ssl and ssl-cert)
    -c, --ssl-cert CERT              SSL certificate file to use (use with: ssl and ssl-key)
```

All users are highly encouraged to only use HTTPS connections if the application is running in production or can be accessed from outside of the local computer. This recommendation is based on the fact that all passwords are passed via the HTTP basic authentication, which

does only encode, but not encrypt the data, wherefore any third party could log and identify the transmitted credentials. To enforce this policy, Nucleus will automatically redirect all incoming plain HTTP connections to HTTPS connections if it is running in the production environment. Additionally, the API can also be started by using the typical rackup command as shown in Listing 27.

Listing 27: Rackup of the Nucleus API

```
1 rackup -s thin config.ru
```

By reason of technical dependencies on event based rack servers, Nucleus can currently only run on the Thin[70] web server. This dependency was introduced by the log tailing operations and the `rack-stream` gem.

When running Nucleus as a Web service without a permanent database, all changes made to the vendor, provider, and endpoint objects will be discarded once the application is terminated. To prevent the data disposal and persist the data permanently, the functionality can be enabled in the configuration, requiring the location where to store the data to be specified.

[70] *Thin: A fast and very simple Ruby web server*. URL: `http://code.macournoyer.com/thin/` (Retrieved: June 29, 2015).

5 Evaluation

In this chapter, the created Platform as a Service abstraction layer is briefly evaluated against its initial requirements and some selected use cases which are described in the literature.

With the help of Travis CI and the developed test suites, Nucleus is known to work with the most recent Ruby versions. All tests were successfully executed against the Ruby interpreters MRI 1.9.3, MRI 2.0.0, MRI 2.1 and MRI 2.2. Not only do the tests show the compatibility with the major Ruby versions, the included adapter tests also simulate the complete lifecycle of cloud applications. All deployed applications of the tests were accessible at the anticipated URLs and generated valid logging entries. Some of the abstraction layer's operations can be seen as minor use cases on their own, e.g., the application data management and the lifecycle handling.

Further aspects of cloud applications were also taken into account with a use case based evaluation of the abstraction layer and its capabilities. In previous publications, some authors already worked out sophisticated use cases to match the characteristics of PaaS applications. Quite often the need to change the current vendor is mentioned. The user, being either a developer or a company, wants to move an application from one platform to another. Loutas et. al, while referring to the findings of the Cloud Computing Use Case Discussion Group [Clo10], states that such a "semantic conflict can be resolved by the means of a standardized management interface" [LKT11]. With the common object description and the identical API for all supported vendors, a unique management interface is provided by Nucleus. Additionally, it would be necessary that the application container is compatible to the technical requirements of the chosen platform, which is however out of scope in this work. Moreover, Nucleus does not yet offer operations for managing and transferring application data, i.e., from databases, which shows that further work is required to pass this holistic use case. According to Loutas et. al [LKT11], additional use cases referring to the management interfaces of PaaS platforms are the application deployment on a PaaS offering and PaaS systems interoperability. Even though the definition of the application deployment use case illustrates the problem of differences in the management interfaces, it mostly refers to technical issues that are related to the chosen database of the application in its explanation. Assuming the use case is pointing at the management interface of the platforms, Nucleus passes this use case as it harmonizes different operation naming with the same functionality over all supported platforms. Hybrid clouds and the requested common PaaS offering model are not offered by Nucleus. Nevertheless, the provided information, e.g., the available runtimes and their versions, are already processed to present a common object model and therefore partially help to pass the use case's requirements. The DMTF specified that it must be possible for the user to increase or decrease the resources that are available to an application so that it can serve more, respectively less requests [Dis10]. Whereas it is currently not possible to increase the capacity of an application instance, the application scaling operation allows to add or remove instances and therefore theoretically allows to adjust the amount of requests that can be handled, too. Maiya et al. [MDYSM12] refer to the foundations of the DMTF and name five use cases. Nucleus can fulfill the first three: the deployment of a new application to the cloud, application scaling during peak demands, and the deployment of a new version of the application. Use cases which cannot be achieved with Nucleus are the configuration of application health monitoring and the notification about a service condition or event. Generally, both use cases can be questioned to be realizable with any of the chosen platforms due to missing features and operations.

At the beginning of this study, it was claimed that interoperability and portability of PaaS can be improved with the creation of an abstraction layer. In summary, application portability is enabled mostly with the common object model that applies to all supported platforms. Hybrid Clouds, describing the deployment of an application to multiple platforms around the globe, are facilitated by the fact that the application description does not have to be changed if all vendors support the technical requirements. Most important, even though an automated vendor change is not yet realizable, the effort that is needed when migrating an application to another vendor is diminished. Nucleus also enhances the interoperability of PaaS platforms and applications. The common interface and the application lifecycle model that apply to all supported platforms allow that applications can easily interact and be adjusted, for instance to handle shifting load scenarios. Tasks that are required for this interaction are the scaling and lifecycle operations. Nucleus' common object model thereby allows applications on different platforms to be capable of semantically processing the information.

Concerning the aspect of the demanded programming language independence, the created JSON interface fully satisfies this requirement. One the one side, Swagger and its tools allow the generation of clients in many languages, for instance Java, PHP, Python, Ruby, Swift and many more[71]. On the other side, the independence is already demonstrated with the interactive Swagger UI interface, from which the API can be used. The Swagger UI and the extensive documentation contribute to a pleasant experience for developers that adopt the abstraction layer. Nucleus' extendability is provided by the modular structure with the dedicated adapters for each supported platform and the possibility to host multiple API versions in parallel.

[71] *Swagger Code Generator*. URL: `https://github.com/swagger-api/swagger-codegen` (Retrieved: July 10, 2015).

6 Future Work

Based on the current state of the defined abstraction layer and its prototype, there are still plenty of tasks which can be worked on in future projects.

Besides minor optimizations and fixes, one improvement that targets mainly at the prototype itself is the definition of additional adapters, e.g., to include Flynn[72], the upcoming release of OpenShift 3[73] or any other platform. Another feature, which did not make it into this first release but was already evaluated in the initial approach, is the support of vertical scaling. Vertical scaling probably is not as important as horizontal scaling, but belongs to the core features which are available on most platforms. The major challenge that must be solved is the combination of precise scaling systems which expect concrete numbers for each adjustable property, as for instance on cloudControl and Cloud Foundry, with the custom scaling levels of Heroku and OpenShift. The custom levels, for instance small, medium, and large gears on OpenShift, represent fixed hardware specifications. One approach could be to translate precise figures to custom levels, but an issue could be to find levels that match with all providers. An alternative would be to translate the abstract levels to precise numbers and apply the levels to meet the minimal requirements. Services are already supported, but there are multiple aspects that could not yet be regarded. Some platforms, e.g., Cloud Foundry, allow the creation of general services that must not be bound and can be used by multiple applications. Aiming at a better standardization of PaaS in general, the implementation of CAMP via a dedicated API version is one of the more ambitious projects that could be realized in the future. The implementation of a CAMP adapter would allow to communicate with additional platforms that support the standard without having to implement a new tailored adapter.

Moving away from the previously mentioned implementation tasks to a more general perspective, Nucleus could be integrated with third party systems to gain additional insights on PaaS platforms and their capabilities. If Nucleus would be integrated with the PaaS profiles project, both projects could improve their quality. Nucleus could prevent semantic errors before they appear, for instance by warning that a certain runtime is not supported by the chosen provider, whereas some sections of the PaaS profiles, e.g., the available services and their plans, could be updated automatically via the use of the Nucleus API. When 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 the PaaS usage without having to fear the effects of a vendor lock-in.

[72] *Flynn*. URL: `https://flynn.io/` (Retrieved: June 28, 2015).
[73] *OpenShift Origin 3 Repository*. URL: `https://github.com/openshift/origin` (Retrieved: May 3, 2015).

7 Conclusion

The research question of this study was whether it is possible to abstract the differences of vendor specific deployment and management interfaces on the PaaS layer by creating an intermediary abstraction layer. After completing all major stages of the project, the question can be affirmed. With its unified deployment and management capabilities, the prototype allows to manage applications on different cloud platforms. Diversities among the platforms could be successfully harmonized and the simulation of an application's lifecycle within the adapter tests proved that Nucleus can be used to deploy and manage applications on cloudControl, Cloud Foundry, Heroku, and OpenShift. When switching the provider, the effort to adapt existing DevOps automation, e.g., the deployment and management processes inside continuous delivery, can be minimized. Nucleus increases the portability and interoperability of PaaS applications and thus helps to avoid critical vendor lock-in effects.

References

[Bad12] Lee Badger. *Cloud Computing Synopsis and Recommendations: Recommendations of the National Institute of Standards and Technology*, 2012.

[BBSR13] Alexandre Beslic, Reda Bendraou, Julien Sopenal, and Jean-Yves Rigolet. Towards a solution avoiding Vendor Lock-in to enable Migration Between Cloud Platforms. In *MDHPCL@ MoDELS*, 2013.

[BIS+14] Antonio Brogi, Ahmad Ibrahim, Jacopo Soldani, José Carrasco, Javier Cubo, Ernesto Pimentel, and Francesco D'Andria. SeaClouds: a European project on seamless management of multi-cloud applications. *ACM SIGSOFT Software Engineering Notes*, 39(1), 2014.

[Car13] Darryl Carlton. Cloud Computing 2014: ready for real business? Gartner, Inc., 2013.

[CCP14] Jose Carrasco, Javier Cubo, and Ernesto Pimentel. Towards a flexible deployment of multi-cloud applications based on TOSCA and CAMP. In *Proceedings of the 1st SeaClouds Workshop*, 2014.

[CH09] Daniele Catteddu and Giles Hogben. Cloud Computing - Benefits, risks and recommendations for information security. European Union Agency for Network and Information Security (ENISA), 2009.

[Clo10] Cloud Computing Use Case Discussion Group. Cloud Computing Use Cases White Paper - Version 4.0. 2010.

[CNS14] David Cunha, Pedro Neves, and Pedro Sousa. PaaS manager: A platform-as-a-service aggregation framework. *Computer Science and Information Systems*, 2014.

[DBCAZ12] Francesco D'Andria, Stefano Bocconi, Jesus Gorronogoitia Cruz, James Ahtes, and Dimitris Zeginis. Cloud4Soa: Multi-cloud Application Management Across PaaS Offerings. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2012.

[Dis10] Distributed Management Task Force (DMTF). Use cases and interactions for managing clouds - A White Paper from the Open Cloud Standards Incubator. June 18, 2010.

[Dus07] L. Dusseault. HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). RFC 4918 (Proposed Standard). Updated by RFC 5689. Internet Engineering Task Force, June 2007.

[EK12] Evren Eren and Christian Karnath. Knackpunkt API. Standardisierte IaaS-Cloud-Schnittstellen. *NET*, (11/2012), November 2012.

[FHH+99] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard). Updated by RFC 7235. Internet Engineering Task Force, June 1999.

[FK06] Joseph Farrell and Paul Klemperer. Coordination and Lock-In: Competition with Switching Costs and Network Effects. *SSRN Electronic Journal*, 2006.

[FR14a] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Authentication. RFC 7235 (Proposed Standard). Internet Engineering Task Force, June 2014.

[FR14b] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231 (Proposed Standard). Internet Engineering Task Force, June 2014.

[GCN+13] Carlos Gonçalves, David Cunha, Pedro Neves, Pedro Sousa, Joao Paulo Barraca, and Diogo Gomes. Towards a cloud service broker for the Meta-Cloud. In *12a Conferência sobre Redes de Computadores*, 2013.

[Gen14] Frank Gens. Worldwide and Regional Public IT Cloud Services 2014–2018 Forecast. IDC, October 2014.

[GS12] Andrea Giessmann and Katarina Stanoevska-Slabeva. Business Models of Platform as a Service (PaaS) Providers: Current State and Future Directions. *Journal of Information Technology Theory and Application*, 12(3), 2012.

[HLST11] Michael Hogan, Fang Liu, Annie Sokol, and Jin Tong. NIST cloud computing standards roadmap. *NIST Special Publication*, 2011.

[KLZ+13] Eleni Kamateri, Nikolaos Loutas, Dimitris Zeginis, et al. Cloud4SOA: A Semantic-Interoperability PaaS Solution for Multi-cloud Platform Management and Portability. In, *Service-Oriented and Cloud Computing*, 2013.

[KPM13] KPMG International. Breaking through the cloud adoption barriers. February 13, 2013.

[KW14] Stefan Kolb and Guido Wirtz. Towards Application Portability in Platform as a Service. In *Proceedings of the 8th IEEE International Symposium on Service-Oriented System Engineering*, April 2014.

[LKT11] Nikolaos Loutas, Eleni Kamateri, and Konstantinos Tarabanis. A Semantic Interoperability Framework for Cloud Platform as a Service. In, November 2011.

[Mas98] L. Masinter. Returning Values from Forms: multipart/form-data. RFC 2388 (Proposed Standard). Internet Engineering Task Force, August 1998.

[MDYSM12] Madhavi Maiya, Sai Dasari, Ravi Yadav, Sandhya Shivaprasad, and Dejan Milojicic. Quantifying Manageability of Cloud Platforms. *2013 IEEE 6th International Conference on Cloud Computing (CLOUD)*, 2012.

[MLBZG11] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing – The business perspective. *Decision Support Systems*, 51(1), April 2011.

[OAS14] OASIS. Cloud Application Management for Platforms Version 1.1. August 2014.

[OF10] Karsten Oberle and Mike Fisher. ETSI CLOUD–initial standardization requirements for cloud services. In, *Economics of Grids, Clouds, Systems, and Services*, 2010.

[PCR11] Dana Petcu, Ciprian Craciun, and Massimiliano Rak. Towards a cross platform Cloud API. *1st International Conference on Cloud Computing and Services Science*, 2011.

[PHMH09] Randy Perry, Eric Hatcher, Robert P. Mahowald, and Stephen D. Hendrick. Force. com cloud platform drives huge time to market and cost savings. *IDC-Whitepaper*, 2009.

[PHMRS12] Fawaz Paraiso, Nicolas Haderer, Philippe Merle, Romain Rouvoy, and Lionel Seinturier. A federated multi-cloud PaaS infrastructure. In *5th International Conference on Cloud Computing*, 2012.

[PMPC13] Dana Petcu, Georgiana Macariu, Silviu Panica, and Ciprian Crăciun. Portable Cloud applications—From theory to practice. *Future Generation Computer Systems*, 29(6), August 2013.

[Pri10] Ben Pring. Cloud Computing: The Next Generation of Outsourcing. Gartner, Inc., November 1, 2010.

[Rig14] RightScale. RightScale 2014: State of the cloud report. Rightscale, Inc., April 2014.

[SR10] Amit Sheth and Ajith Ranabahu. Semantic modeling for cloud computing, part 2. *Internet Computing, IEEE*, 14(4), 2010.

[SYMT13] Mohamed Sellami, Sami Yangui, Mohamed Mohamed, and Samir Tata. PaaS-Independent Provisioning and Management of Applications in the Cloud. In *Proceedings of the 6th International Conference on Cloud Computing*, June 2013.

[The12] The Open Group. Cloud ROI Survey Results Comparison 2011 & 2012. December 19, 2012.

[ZDB+13] Dimitris Zeginis, Francesco D'Andria, Stefano Bocconi, Jesus Gorronogoitia Cruz, Oriol Collell Martin, Panagiotis Gouvas, Giannis Ledakis, and Konstantinos A. Tarabanis. A user-centric multi-PaaS application management solution for hybrid multi-Cloud scenarios. *Scalable Computing: Practice and Experience*, 14(1), April 2013.

Appendix A - API Operation Mappings

Operation	cloudControl	Cloud Foundry	Heroku	OpenShift
GET Service	GET /addon/{sid}	GET /v2/services/{sid}?inline-relations-depth=1	GET /addon-services/{sid}	GET /cartridge/{sid}
GET Service List	GET /addon	GET /v2/services?inline-relations-depth=1	GET /addon-services	GET /cartridges
GET ServicePlan	GET /addon/{sid}	GET /v2/services/{sid}/service_plans/{pid}	GET /addon-services/{sid}/plans/{pid}	GET /cartridge/{sid}
GET ServicePlan List	GET /addon/{sid}	GET /v2/services/{sid}/service_plans	GET /addon-services/{sid}/plans	GET /cartridge/{sid}
GET Region	-	-	GET /regions	GET /regions
GET Region List	-	-	GET /regions	GET /regions
GET Application	GET /app/{aid} GET /app/{aid}/deployment/nucleus	GET /v2/apps/{aid}	GET /apps/{aid}	GET /application/{aid}
GET Application List	GET /app/ **For each loaded application:** • GET /app/{aid}/deployment/nucleus	GET /v2/apps	GET /apps	GET /applications
POST Application	POST /app *Body:* • repository_type = 'git' • type = {application/runtimes[0]} • name = {application/name} POST /app/{application/name}/deployment *Body:* • name = 'nucleus' POST /app/{application/name}/deployment/{dplid}/addon *Body:* • addon = 'config.free' • options = '{"nucleus-initialized": "true"}' PUT /app/{application/name}/deployment/{dplid}/addon/config.free *Body:* • addon = 'config.free' • settings = '{"nucleus-initialized":null}' • force = true	GET /v2/spaces POST /v2/apps *Body:* • space_guid = {*user_space_guid*} • buildpack = {application/runtimes[0]} GET /v2/routes PUT /v2/apps/{aid}/routes/{*def_route_id*}	POST /apps *Body:* • {application} **If a custom buildpack is applied:** PUT /apps/{aid}/buildpack-installations *Body:* • updates = {application/runtimes}	GET /domains POST /domains/{domains[0]/name}/applications *Body:* • {application} PUT /application/{aid} *Body:* • keep_deployments = 2 • auto_deploy = false
PATCH Application	-	PUT /v2/apps/{aid}	PATCH /apps/{aid} *Body:* • {application} **If a custom buildpack is applied:** PUT /apps/{aid}/buildpack-installations *Body:* • updates = {application/runtimes}	-
DELETE Application	GET /app/{aid}/deployment **For each deployment:** • DELETE /app/{aid}/deployment/{dplid} DELETE /app/{aid}	GET /v2/apps/{aid}/routes?q=host:{aid}&inline-relations-depth=1 **With default route:** • DELETE /v2/routes/{*route_id}* DELETE /v2/apps/{aid}	DELETE /apps/{aid}	DELETE /applications/{aid}

Operation	cloudControl	Cloud Foundry	Heroku	OpenShift
POST Application/-data/deploy	GET /app/{aid}/deployment/nucleus GET /user *Git: Clone repo, extract file, commit, push* **If the application was already started:** PUT /app/{aid}/deployment/nucleus *Body:* • version = '-1'	PUT /v2/apps/{aid}/bits *Body:* • multipart = true • application = {file as .zip} • async = false • resources = '[]'	GET /apps/{aid} GET /account *Git: Clone repo, extract file, commit, push*	GET /application/{aid} GET /user *Git: Clone repo, extract file, commit, push* POST /application/{aid}/deployments *Body:* • force_clean_build = true
POST Application/-data/rebuild	GET /app/{aid}/deployment/nucleus GET /user *Git: Clone repo, modify marker, commit, push* PUT /app/{aid}/deployment/nucleus *Body:* • version = '-1'	POST /v2/apps/{aid}/restage	GET /apps/{aid} GET /account *Git: Clone repo, modify marker, commit, push*	GET /application/{aid} GET /user *Git: Clone repo, modify marker, commit, push* POST /application/{aid}/deployments *Body:* • force_clean_build = true
GET Application/-data/download	GET /app/{aid}/deployment/nucleus *Git: Clone repo*	GET /v2/apps/{aid}/download	GET /apps/{aid} *Git: Clone repo*	GET /application/{aid} *Git: Clone repo*
POST Application/actions/start	GET /app/{aid}/deployment/nucleus PUT /app/{aid}/deployment/nucleus *Body:* • version = '-1'	PUT /v2/apps/{aid} *Body:* • state = 'STARTED'	PATCH /apps/{aid} *Body:* • maintenance = false PATCH /apps/{aid}/formation *Body:* • updates = [{process:'worker',quantity:1}]	POST /application/{aid}/events *Body:* • event = 'start'
POST Application/actions/stop	-	PUT /v2/apps/{aid} *Body:* • state = 'STOPPED'	PATCH /apps/{aid} *Body:* • maintenance = true PATCH /apps/{aid}/formation *Body:* • updates = [{process:'worker',quantity:0}]	POST /application/{aid}/events *Body:* • event = 'stop'
POST Application/actions/restart	-	***stop*** ***start***	***stop*** ***start***	POST /application/{aid}/events *Body:* • event = 'restart'
POST Application/actions/scale	PUT /app/{aid}/deployment/nucleus *Body:* • min_boxes = {instances}	PUT /v2/apps/{aid} *Body:* • instances = {instances}	PATCH /apps/{aid}/formation *Body:* • updates = [{process:'web',quantity:{instances}}]	GET /application/{aid} GET /user **a) If {instances} > {current_instances}, repeat {instances} - {current_instances} times:** POST /application/{aid}/events *Body:* • event = 'scale-up' **b) If {instances} < {current_instances}, repeat {current_instances} - {instances} times:** POST /application/{aid}/events *Body:* • event = 'scale-down'
GET Domain	GET /app/{aid}/deployment/{dplid}/alias/{did}	GET /v2/apps/{app_id}/routes GET {route/domain_url}	GET /apps/{aid}/domains/{did}	GET /application/{aid}/aliases/{did}
GET Domain List	GET /app/{aid}/deployment/{dplid}/alias **For each alias in the response:** GET /app/{aid}/deployment/{dplid}/alias/{did}	GET /v2/apps/{app_id}/routes **Load for each route:** GET {route/domain_url}	GET /apps/{aid}/domains	GET /application/{aid}/aliases

Operation	cloudControl	Cloud Foundry	Heroku	OpenShift
POST Domain	POST /app/{aid}/deployment/{dplid}/alias *Body:* • name = {domain/name}	GET /v2/private_domains GET /v2/shared_domains **If such a domain does not exist:** GET /v2/spaces POST /v2/private_domains *Body:* • name: {*domain_name*} • owning_organization_guid: {user_org_guid} GET /v2/routes **If the route does not exist:** GET /v2/spaces POST /v2/routes *Body:* • domain_guid = {*domain/guid*} • host = {*domain_host*} • space_guid = {*user_space_guid*} PUT /v2/apps/{aid}/routes/{*route/guid*}	POST /apps/{aid}/domains *Body:* • hostname = {domain/name}	POST /application/{aid}/aliases *Body:* • id = {domain/name}
DELETE Domain	DELETE /app/{aid}/deployment/{dplid}/alias/{did}	DELETE /v2/apps/{aid}/routes/{did} GET /v2/routes/{did}/apps **If route_in_apps/total_results == 0:** DELETE /v2/routes/{did}	DELETE /apps/{aid}/domains/{did}	DELETE /application/{aid}/aliases/{did}
GET Installed Service	GET /app/{aid}/deployment/{dplid}/addon GET /app/{aid}/deployment/{dplid}/addon/{*assigned_addon_id*} GET /addon/{sid}	GET /v2/services/{sid}/service_plans GET /v2/apps/{aid}/service_bindings?inline-relations-depth=1	GET /addon-services/{sid}	GET /application/{aid}/cartridge/{sid}
GET Installed Service List	GET /app/{aid}/deployment/{dplid}/addon **For each addon in the list:** GET /addon/{*assignment/addon_option/name*}	GET /v2/apps/{aid}/service_bindings?inline-relations-depth=1	GET /addon-services	GET /application/{aid}/cartridges
POST Installed Service	POST /app/{aid}/deployment/{dplid}/addon *Body:* • addon = '{service/id}.{plan/id}'	GET /v2/services/{sid} POST /v2/service_instances *Body:* • space_guid = {*user_space_guid*} • service_plan_guid = {plan/id} • name = {*dynamically generated name*} POST /v2/service_bindings *Body:* • service_instance_guid = {*instance/guid*} • app_guid = {aid}	POST /apps/{aid}/addons *Body:* • plan = '{service/id}:{plan/id}'	GET /cartridge/{sid} POST /application/{aid}/cartridges *Body:* • cartridge = {service/id}
PATCH Installed Service	PUT /app/{aid}/deployment/{dplid}/addon/{pid} *Body:* • addon = {plan/id}	GET /v2/services/{sid}/service_plans GET /v2/apps/{aid}/service_bindings?inline-relations-depth=1 PUT /v2/service_instances/{service_instance_guid} *Body:* • service_plan_guid = {plan/id}	PATCH /apps/{aid}/addons/{sid} *Body:* • plan = {plan/id}	-
DELETE Installed Service	DELETE /app/{aid}/deployment/{dplid}/addon/{pid}	GET /v2/services/{sid}/service_plans GET /v2/apps/{aid}/service_bindings?inline-relations-depth=1 DELETE /v2/apps/{aid}/service_bindings/{binding_guid} DELETE /v2/service_instances/{service_instance_guid}	DELETE /apps/{aid}/addons/{assid}	DELETE /application/{aid}/cartridge/{sid}
GET Environment Variable	GET /app/{aid}/deployment/{dplid}/addon/config.free	GET /v2/apps/{aid}/env	GET /apps/{aid}/config-vars	GET /application/{aid}/environment-variable/{vid}
GET Environment Variable List	GET /app/{aid}/deployment/{dplid}/addon/config.free	GET /v2/apps/{aid}/env	GET /apps/{aid}/config-vars	GET /application/{aid}/environment-variables

Operation	cloudControl	Cloud Foundry	Heroku	OpenShift
POST Environment Variable	PUT /app/{aid}/deployment/{dplid}/addon/config.free *Body:* • addon = 'config.free' • force = true • settings = '{"{variable/key}":"{variable/value}"}'	GET /v2/apps/{aid}/env PUT /v2/apps/{aid} *Body:* • environment_json = {*current environment_json*} + {variable/key}:{variable/value}	PATCH /apps/{aid}/config-vars *Body:* • {variable}	POST /application/{aid}/environment-variables *Body:* • name = {variable/key} • value = {variable/value}
PATCH Environment Variable	Platform does not offer an update operation. Values can be applied using the operations that are described in the above CREATE Environment Variable mapping.			PUT /application/{aid}/environment-variable/{vid} *Body:* • value = {variable/value}
DELETE Environment Variable	PUT /app/{aid}/deployment/{dplid}/addon/config.free *Body:* • addon = 'config.free' • force = true • settings = '{"{variable/key}":null}'	GET /v2/apps/{aid}/env PUT /v2/apps/{aid} *Body:* • environment_json = {*current environment_json*} - {var_id}	PATCH /apps/{aid}/config-vars *Body:* • {var_id} = null	DELETE /application/{aid}/environment-variable/{vid}
GET Log List	-	GET /v2/apps/{aid}/instances/0/files/logs	-	*via SSH interaction*
GET Log	GET /app/{aid}/deployment/{dplid}/log/{lid}	**a) If log is a stream:** GET *{loggregator_endpoint}*:443/recent?app={aid} **b) If log is a file:** GET /v2/apps/{aid}/instances/0/files/logs/{lid}	POST /apps/{aid}/log-sessions *Body:* • source = {*'heroku' or 'app'*} • tail = false • *(optional)* dyno = {*'api' or 'router'*} GET {logplex_url}	*via SSH interaction*
GET Log download	**Load the log with the GET Log operation, then bundle it to the response archive**			
GET Log download all	**Load all logs that are included in the list with the GET Log operation, then bundle them to the response archive**			
GET Log/tail	GET /app/{aid}/deployment/{dplid}/log/{lid}?timestamp={latest_msg_time}	**a) If log is a stream:** wss://*{loggregator_endpoint}*:443/tail/?app={aid} **b) If log is a file, call in loop:** GET /v2/apps/{aid}/instances/0/files/logs/{lid}	POST /apps/{aid}/log-sessions *Body:* • source = {*'heroku' or 'app'*} • tail = true • *(optional)* dyno = {*'api' or 'router'*} **Connect to stream:** GET {logplex_url}	*via SSH interaction*

LEGEND

Font Style	**bold**	: Conditional instructions	***bold + italic***	: Use other operation *italic* : comments, configuration variables, non-HTTP interactions
	{in curly braces}	: request variable	{*italic in curly braces*}	: variable with static or previous response related value
Variable Abbreviations	*aid*	: application_id	*sid*	: service_id
	pid	: plan_id	*did*	: domain_id
	vid	: variable_id	*lid*	: log_id
[only used with cloudControl]	*dplid*	: deployment_id	*assid*	: assignment_id

Table 33: Operations mapping overview, from Nucleus API to vendor specific operations

Appendix B - Adapter test spec template

Listing 28: Adapter test spec template

```
require 'spec/adapter/adapter_spec_helper'

describe Nucleus::Adapters::{API_VERSION}::{VENDOR_CLASS} do
  before :all do
    # skip these example groups / tests for this adapter. E.g.:
    # @unsupported = ['with valid credentials is compliant and application update']
    @unsupported = []
    @endpoint = '{ENDPOINT_ID}'
    @api_version = '{API_VERSION}'
    @app_min = { original_name: 'nucleustestappminproperties', updated_name: '
        nucleustestappminproperties', region: 'default' }
    @app_all = { original_name: 'nucleustestappallupdated', updated_name: '
        nucleustestappallupdated', region: 'default' }
  end
  before do |example|
    if skip_example?(described_class, example.metadata[:full_description], @unsupported)
      skip('This feature is currently not supported by CloudControl - 501')
    end
    # reload adapter for each test
    @adapter = load_adapter(@endpoint, @api_version)
  end

  context 'with invalid credentials' do
    let!(:request_headers) { credentials(@endpoint, false) }
    include_examples 'compliant adapter with invalid credentials'
  end

  describe 'with missing credentials' do
    let!(:request_headers) { {} }
    include_examples 'compliant adapter with invalid credentials'
  end

  context 'with valid credentials' do
    let!(:request_headers) { credentials(@endpoint) }
    include_examples 'compliant adapter with valid credentials'

    describe 'native adapter call' do
      describe 'against endpoint' do
        describe 'does fetch all available addons' do
          before do
            get "/endpoints/#{@endpoint}/call/addon", request_headers
          end
          include_examples 'a valid GET request'
          it 'with the specified structure' do
            expect(json_body[0].keys).to include(:name, :stage, :options)
          end
          it 'with the matching content declaration' do
            expect_json_types(:array)
          end
        end
      end
    end
  end
end
```

Appendix C - Swagger UI - API documentation

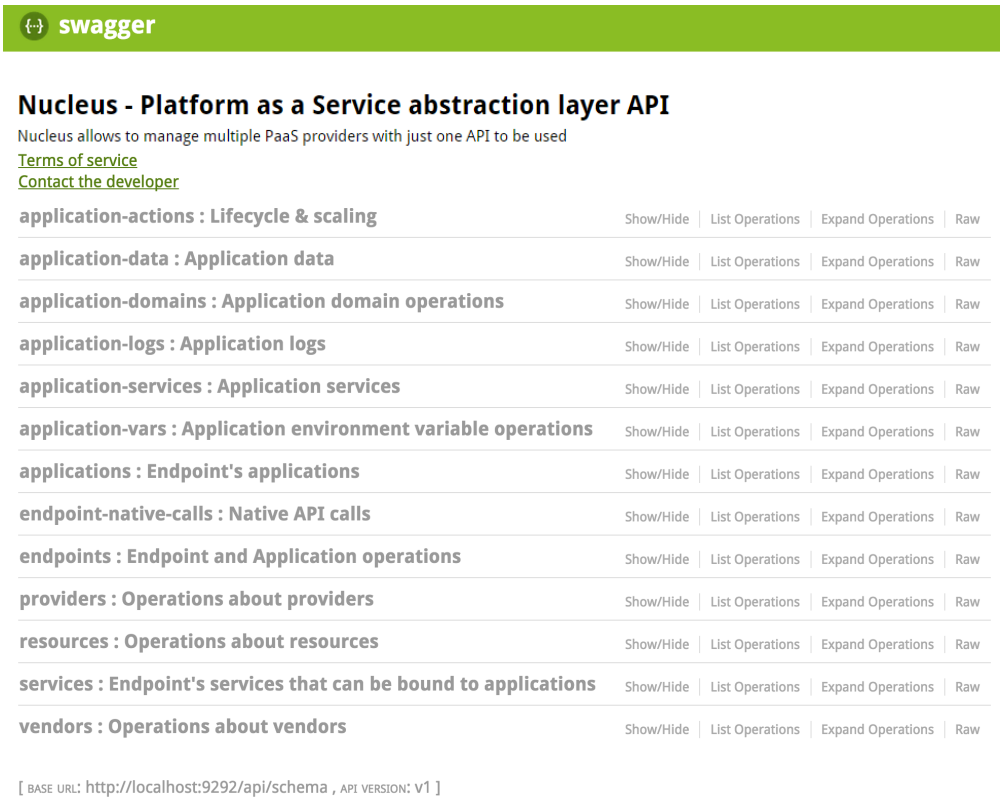


Figure 13: Swagger UI Overview, showing the grouped API objects and operations

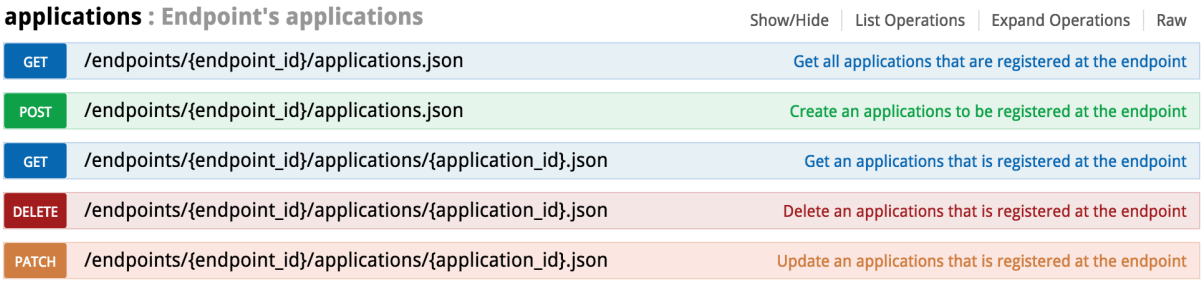


Figure 14: Swagger UI showing all methods available in an operation group

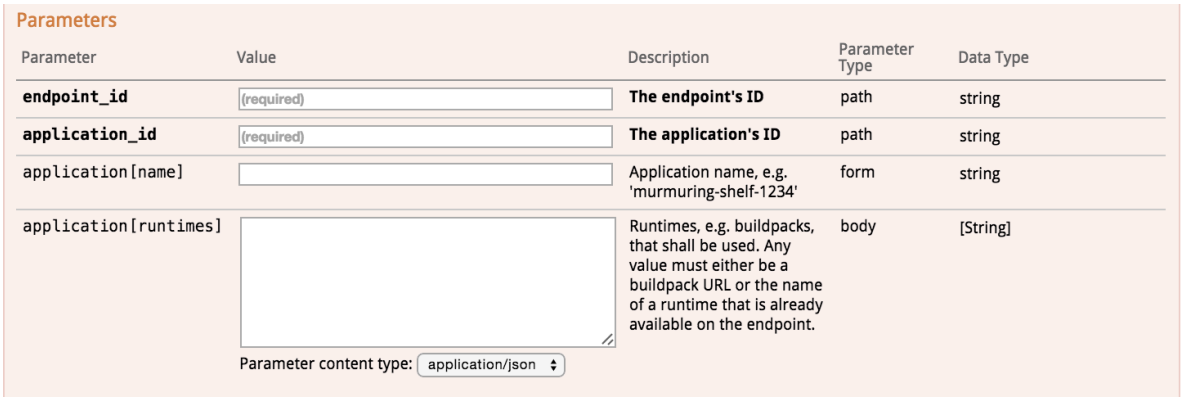
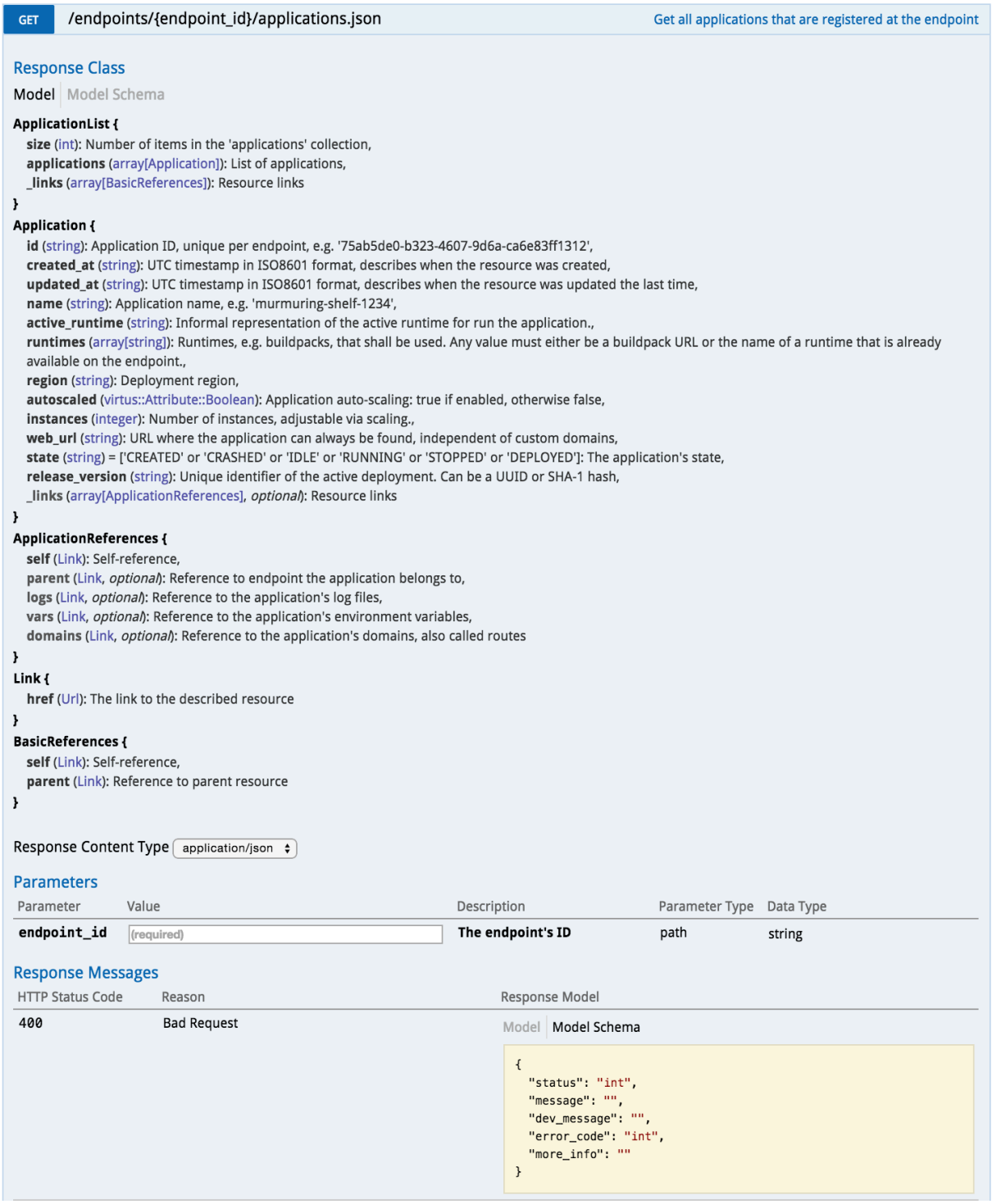


Figure 15: Swagger UI presentation of the PATCH request to update an application



GET /endpoints/{endpoint_id}/applications.json — Get all applications that are registered at the endpoint

Response Class

Model | Model Schema

ApplicationList {
size (int): Number of items in the 'applications' collection,
applications (array[Application]): List of applications,
_links (array[BasicReferences]): Resource links
}
Application {
id (string): Application ID, unique per endpoint, e.g. '75ab5de0-b323-4607-9d6a-ca6e83ff1312',
created_at (string): UTC timestamp in ISO8601 format, describes when the resource was created,
updated_at (string): UTC timestamp in ISO8601 format, describes when the resource was updated the last time,
name (string): Application name, e.g. 'murmuring-shelf-1234',
active_runtime (string): Informal representation of the active runtime for run the application.,
runtimes (array[string]): Runtimes, e.g. buildpacks, that shall be used. Any value must either be a buildpack URL or the name of a runtime that is already available on the endpoint.,
region (string): Deployment region,
autoscaled (virtus::Attribute::Boolean): Application auto-scaling: true if enabled, otherwise false,
instances (integer): Number of instances, adjustable via scaling.,
web_url (string): URL where the application can always be found, independent of custom domains,
state (string) = ['CREATED' or 'CRASHED' or 'IDLE' or 'RUNNING' or 'STOPPED' or 'DEPLOYED']: The application's state,
release_version (string): Unique identifier of the active deployment. Can be a UUID or SHA-1 hash,
_links (array[ApplicationReferences], *optional*): Resource links
}
ApplicationReferences {
self (Link): Self-reference,
parent (Link, *optional*): Reference to endpoint the application belongs to,
logs (Link, *optional*): Reference to the application's log files,
vars (Link, *optional*): Reference to the application's environment variables,
domains (Link, *optional*): Reference to the application's domains, also called routes
}
Link {
href (Url): The link to the described resource
}
BasicReferences {
self (Link): Self-reference,
parent (Link): Reference to parent resource
}

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
endpoint_id	(required)	**The endpoint's ID**	path	string

Response Messages

HTTP Status Code	Reason	Response Model
400	Bad Request	Model | Model Schema

```
{
  "status": "int",
  "message": "",
  "dev_message": "",
  "error_code": "int",
  "more_info": ""
}
```

Figure 16: Swagger UI presentation of the GET request to list all applications

List of previous University of Bamberg reports

Bamberger Beiträge zur Wirtschaftsinformatik

Nr. 1 (1989) Augsburger W., Bartmann D., Sinz E.J.: Das Bamberger Modell: Der Diplom-Studiengang Wirtschaftsinformatik an der Universität Bamberg (Nachdruck Dez. 1990)

Nr. 2 (1990) Esswein W.: Definition, Implementierung und Einsatz einer kompatiblen Datenbankschnittstelle für PROLOG

Nr. 3 (1990) Augsburger W., Rieder H., Schwab J.: Endbenutzerorientierte Informationsgewinnung aus numerischen Daten am Beispiel von Unternehmenskennzahlen

Nr. 4 (1990) Ferstl O.K., Sinz E.J.: Objektmodellierung betrieblicher Informationsmodelle im Semantischen Objektmodell (SOM) (Nachdruck Nov. 1990)

Nr. 5 (1990) Ferstl O.K., Sinz E.J.: Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM)

Nr. 6 (1991) Augsburger W., Rieder H., Schwab J.: Systemtheoretische Repräsentation von Strukturen und Bewertungsfunktionen über zeitabhängigen betrieblichen numerischen Daten

Nr. 7 (1991) Augsburger W., Rieder H., Schwab J.: Wissensbasiertes, inhaltsorientiertes Retrieval statistischer Daten mit EISREVU / Ein Verarbeitungsmodell für eine modulare Bewertung von Kennzahlenwerten für den Endanwender

Nr. 8 (1991) Schwab J.: Ein computergestütztes Modellierungssystem zur Kennzahlenbewertung

Nr. 9 (1992) Gross H.-P.: Eine semantiktreue Transformation vom Entity-Relationship-Modell in das Strukturierte Entity-Relationship-Modell

Nr. 10 (1992) Sinz E.J.: Datenmodellierung im Strukturierten Entity-Relationship-Modell (SERM)

Nr. 11 (1992) Ferstl O.K., Sinz E. J.: Glossar zum Begriffsystem des Semantischen Objektmodells

Nr. 12 (1992) Sinz E. J., Popp K.M.: Zur Ableitung der Grobstruktur des konzeptuellen Schemas aus dem Modell der betrieblichen Diskurswelt

Nr. 13 (1992) Esswein W., Locarek H.: Objektorientierte Programmierung mit dem Objekt-Rollenmodell

Nr. 14 (1992) Esswein W.: Das Rollenmodell der Organsiation: Die Berücksichtigung aufbauorganisatorische Regelungen in Unternehmensmodellen

Nr. 15 (1992) Schwab H. J.: EISREVU-Modellierungssystem. Benutzerhandbuch

Nr. 16 (1992) Schwab K.: Die Implementierung eines relationalen DBMS nach dem Client/Server-Prinzip

Nr. 17 (1993) Schwab K.: Konzeption, Entwicklung und Implementierung eines computergestützten Bürovorgangssystems zur Modellierung von Vorgangsklassen und Abwicklung und Überwachung von Vorgängen. Dissertation

Nr. 18 (1993) Ferstl O.K., Sinz E.J.: Der Modellierungsansatz des Semantischen Objektmodells

Nr. 19 (1994) Ferstl O.K., Sinz E.J., Amberg M., Hagemann U., Malischewski C.: Tool-Based Business Process Modeling Using the SOM Approach

Nr. 20 (1994) Ferstl O.K., Sinz E.J.: From Business Process Modeling to the Specification of Distributed Business Application Systems - An Object-Oriented Approach -. 1st edition, June 1994

Ferstl O.K., Sinz E.J. : Multi-Layered Development of Business Process Models and Distributed Business Application Systems - An Object-Oriented Approach -. 2nd edition, November 1994

Nr. 21 (1994) Ferstl O.K., Sinz E.J.: Der Ansatz des Semantischen Objektmodells zur Modellierung von Geschäftsprozessen

Nr. 22 (1994) Augsburger W., Schwab K.: Using Formalism and Semi-Formal Constructs for Modeling Information Systems

Nr. 23 (1994) Ferstl O.K., Hagemann U.: Simulation hierarischer objekt- und transaktionsorientierter Modelle

Nr. 24 (1994) Sinz E.J.: Das Informationssystem der Universität als Instrument zur zielgerichteten Lenkung von Universitätsprozessen

Nr. 25 (1994) Wittke M., Mekinic, G.: Kooperierende Informationsräume. Ein Ansatz für verteilte Führungsinformationssysteme

Nr. 26 (1995) Ferstl O.K., Sinz E.J.: Re-Engineering von Geschäftsprozessen auf der Grundlage des SOM-Ansatzes

Nr. 27 (1995) Ferstl, O.K., Mannmeusel, Th.: Dezentrale Produktionslenkung. Erscheint in CIM-Management 3/1995

Nr. 28 (1995) Ludwig, H., Schwab, K.: Integrating cooperation systems: an event-based approach

Nr. 30 (1995) Augsburger W., Ludwig H., Schwab K.: Koordinationsmethoden und -werkzeuge bei der computergestützten kooperativen Arbeit

Nr. 31 (1995) Ferstl O.K., Mannmeusel T.: Gestaltung industrieller Geschäftsprozesse

Nr. 32 (1995) Gunzenhäuser R., Duske A., Ferstl O.K., Ludwig H., Mekinic G., Rieder H., Schwab H.-J., Schwab K., Sinz E.J., Wittke M: Festschrift zum 60. Geburtstag von Walter Augsburger

Nr. 33 (1995) Sinz, E.J.: Kann das Geschäftsprozeßmodell der Unternehmung das unternehmensweite Datenschema ablösen?

Nr. 34 (1995) Sinz E.J.: Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme - Entwicklung, aktueller Stand und Trends -

Nr. 35 (1995) Sinz E.J.: Serviceorientierung der Hochschulverwaltung und ihre Unterstützung durch workflow-orientierte Anwendungssysteme

Nr. 36 (1996) Ferstl O.K., Sinz, E.J., Amberg M.: Stichwörter zum Fachgebiet Wirtschaftsinformatik. Erscheint in: Broy M., Spaniol O. (Hrsg.): Lexikon Informatik und Kommunikationstechnik, 2. Auflage, VDI-Verlag, Düsseldorf 1996

Nr. 37 (1996) Ferstl O.K., Sinz E.J.: Flexible Organizations Through Object-oriented and Transaction-oriented Information Systems, July 1996

Nr. 38 (1996) Ferstl O.K., Schäfer R.: Eine Lernumgebung für die betriebliche Aus- und Weiterbildung on demand, Juli 1996

Nr. 39 (1996) Hazebrouck J.-P.: Einsatzpotentiale von Fuzzy-Logic im Strategischen Management dargestellt an Fuzzy-System-Konzepten für Portfolio-Ansätze

Nr. 40 (1997) Sinz E.J.: Architektur betrieblicher Informationssysteme. In: Rechenberg P., Pomberger G. (Hrsg.): Handbuch der Informatik, Hanser-Verlag, München 1997

Nr. 41 (1997) Sinz E.J.: Analyse und Gestaltung universitärer Geschäftsprozesse und Anwendungssysteme. Angenommen für: Informatik '97. Informatik als Innovationsmotor. 27. Jahrestagung der Gesellschaft für Informatik, Aachen 24.-26.9.1997

Nr. 42 (1997) Ferstl O.K., Sinz E.J., Hammel C., Schlitt M., Wolf S.: Application Objects – fachliche Bausteine für die Entwicklung komponentenbasierter Anwendungssysteme. Angenommen für: HMD – Theorie und Praxis der Wirtschaftsinformatik. Schwerpunkheft ComponentWare, 1997

Nr. 43 (1997): Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using the Semantic Object Model (SOM) – A Methodological Framework - . Accepted for: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1997

Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using (SOM), 2nd Edition. Appears in: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1998

Nr. 44 (1997) Ferstl O.K., Schmitz K.: Zur Nutzung von Hypertextkonzepten in Lernumgebungen. In: Conradi H., Kreutz R., Spitzer K. (Hrsg.): CBT in der Medizin – Methoden, Techniken, Anwendungen -. Proceedings zum Workshop in Aachen 6. – 7. Juni 1997. 1. Auflage Aachen: Verlag der Augustinus Buchhandlung

Nr. 45 (1998) Ferstl O.K.: Datenkommunikation. In. Schulte Ch. (Hrsg.): Lexikon der Logistik, Oldenbourg-Verlag, München 1998

Nr. 46 (1998) Sinz E.J.: Prozeßgestaltung und Prozeßunterstützung im Prüfungswesen. Erschienen in: Proceedings Workshop „Informationssysteme für das Hochschulmanagement". Aachen, September 1997

Nr. 47 (1998) Sinz, E.J.:, Wismans B.: Das „Elektronische Prüfungsamt". Erscheint in: Wirtschaftswissenschaftliches Studium WiSt, 1998

Nr. 48 (1998) Haase, O., Henrich, A.: A Hybrid Respresentation of Vague Collections for Distributed Object Management Systems. Erscheint in: IEEE Transactions on Knowledge and Data Engineering

Nr. 49 (1998) Henrich, A.: Applying Document Retrieval Techniques in Software Engineering Environments. In: Proc. International Conference on Database and Expert Systems

Applications. (DEXA 98), Vienna, Austria, Aug. 98, pp. 240-249, Springer, Lecture Notes in Computer Sciences, No. 1460

Nr. 50 (1999) Henrich, A., Jamin, S.: On the Optimization of Queries containing Regular Path Expressions. Erscheint in: Proceedings of the Fourth Workshop on Next Generation Information Technologies and Systems (NGITS'99), Zikhron-Yaakov, Israel, July, 1999 (Springer, Lecture Notes)

Nr. 51 (1999) Haase O., Henrich, A.: A Closed Approach to Vague Collections in Partly Inaccessible Distributed Databases. Erscheint in: Proceedings of the Third East-European Conference on Advances in Databases and Information Systems – ADBIS'99, Maribor, Slovenia, September 1999 (Springer, Lecture Notes in Computer Science)

Nr. 52 (1999) Sinz E.J., Böhnlein M., Ulbrich-vom Ende A.: Konzeption eines Data Warehouse-Systems für Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule" im Rahmen der 29. Jahrestagung der Gesellschaft für Informatik, Paderborn, 6. Oktober 1999

Nr. 53 (1999) Sinz E.J.: Konstruktion von Informationssystemen. Der Beitrag wurde in geringfügig modifizierter Fassung angenommen für: Rechenberg P., Pomberger G. (Hrsg.): Informatik-Handbuch. 2., aktualisierte und erweiterte Auflage, Hanser, München 1999

Nr. 54 (1999) Herda N., Janson A., Reif M., Schindler T., Augsburger W.: Entwicklung des Intranets SPICE: Erfahrungsbericht einer Praxiskooperation.

Nr. 55 (2000) Böhnlein M., Ulbrich-vom Ende A.: Grundlagen des Data Warehousing. Modellierung und Architektur

Nr. 56 (2000) Freitag B, Sinz E.J., Wismans B.: Die informationstechnische Infrastruktur der Virtuellen Hochschule Bayern (vhb). Angenommen für Workshop "Unternehmen Hochschule 2000" im Rahmen der Jahrestagung der Gesellschaft f. Informatik, Berlin 19. - 22. September 2000

Nr. 57 (2000) Böhnlein M., Ulbrich-vom Ende A.: Developing Data Warehouse Structures from Business Process Models.

Nr. 58 (2000) Knobloch B.: Der Data-Mining-Ansatz zur Analyse betriebswirtschaftlicher Daten.

Nr. 59 (2001) Sinz E.J., Böhnlein M., Plaha M., Ulbrich-vom Ende A.: Architekturkonzept eines verteilten Data-Warehouse-Systems für das Hochschulwesen. Angenommen für: WI-IF 2001, Augsburg, 19.-21. September 2001

Nr. 60 (2001) Sinz E.J., Wismans B.: Anforderungen an die IV-Infrastruktur von Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule 2001" im Rahmen der Jahrestagung der Gesellschaft für Informatik, Wien 25. – 28. September 2001

Änderung des Titels der Schriftenreihe *Bamberger Beiträge zur Wirtschaftsinformatik* in *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* ab Nr. 61

Note: The title of our technical report series has been changed from *Bamberger Beiträge zur Wirtschaftsinformatik* to *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* starting with TR No. 61

Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik

Nr. 61 (2002) Goré R., Mendler M., de Paiva V. (Hrsg.): Proceedings of the International Workshop on Intuitionistic Modal Logic and Applications (IMLA 2002), Copenhagen, July 2002.

Nr. 62 (2002) Sinz E.J., Plaha M., Ulbrich-vom Ende A.: Datenschutz und Datensicherheit in einem landesweiten Data-Warehouse-System für das Hochschulwesen. Erscheint in: Beiträge zur Hochschulforschung, Heft 4-2002, Bayerisches Staatsinstitut für Hochschulforschung und Hochschulplanung, München 2002

Nr. 63 (2005) Aguado, J., Mendler, M.: Constructive Semantics for Instantaneous Reactions

Nr. 64 (2005) Ferstl, O.K.: Lebenslanges Lernen und virtuelle Lehre: globale und lokale Verbesserungspotenziale. Erschienen in: Kerres, Michael; Keil-Slawik, Reinhard (Hrsg.); Hochschulen im digitalen Zeitalter: Innovationspotenziale und Strukturwandel, S. 247 – 263; Reihe education quality forum, herausgegeben durch das Centrum für eCompetence in Hochschulen NRW, Band 2, Münster/New York/München/Berlin: Waxmann 2005

Nr. 65 (2006) Schönberger, Andreas: Modelling and Validating Business Collaborations: A Case Study on RosettaNet

Nr. 66 (2006) Markus Dorsch, Martin Grote, Knut Hildebrandt, Maximilian Röglinger, Matthias Sehr, Christian Wilms, Karsten Loesing, and Guido Wirtz: Concealing Presence Information in Instant Messaging Systems, April 2006

Nr. 67 (2006) Marco Fischer, Andreas Grünert, Sebastian Hudert, Stefan König, Kira Lenskaya, Gregor Scheithauer, Sven Kaffille, and Guido Wirtz: Decentralized Reputation Management for Cooperating Software Agents in Open Multi-Agent Systems, April 2006

Nr. 68 (2006) Michael Mendler, Thomas R. Shiple, Gérard Berry: Constructive Circuits and the Exactness of Ternary Simulation

Nr. 69 (2007) Sebastian Hudert: A Proposal for a Web Services Agreement Negotiation Protocol Framework . February 2007

Nr. 70 (2007) Thomas Meins: Integration eines allgemeinen Service-Centers für PC-und Medientechnik an der Universität Bamberg – Analyse und Realisierungs-Szenarien. February 2007 (out of print)

Nr. 71 (2007) Andreas Grünert: Life-cycle assistance capabilities of cooperating Software Agents for Virtual Enterprises. März 2007

Nr. 72 (2007) Michael Mendler, Gerald Lüttgen: Is Observational Congruence on μ-Expressions Axiomatisable in Equational Horn Logic?

Nr. 73 (2007) Martin Schissler: out of print

Nr. 74 (2007) Sven Kaffille, Karsten Loesing: Open chord version 1.0.4 User's Manual. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 74, Bamberg University, October 2007. ISSN 0937-3349.

Nr. 75 (2008) Karsten Loesing (Hrsg.): Extended Abstracts of the Second *Privacy Enhancing Technologies Convention* (PET-CON 2008.1). Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 75, Bamberg University, April 2008. ISSN 0937-3349.

Nr. 76 (2008) Gregor Scheithauer, Guido Wirtz: Applying Business Process Management Systems – A Case Study. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 76, Bamberg University, May 2008. ISSN 0937-3349.

Nr. 77 (2008) Michael Mendler, Stephan Scheele: Towards Constructive Description Logics for Abstraction and Refinement. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 77, Bamberg University, September 2008. ISSN 0937-3349.

Nr. 78 (2008) Gregor Scheithauer, Matthias Winkler: A Service Description Framework for Service Ecosystems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 78, Bamberg University, October 2008. ISSN 0937-3349.

Nr. 79 (2008) Christian Wilms: Improving the Tor Hidden Service Protocol Aiming at Better Performances. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 79, Bamberg University, November 2008. ISSN 0937-3349.

Nr. 80 (2009) Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb, Andreas Schönberger, Guido Wirtz: QoS Enabled B2B Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 80, Bamberg University, May 2009. ISSN 0937-3349.

Nr. 81 (2009) Ute Schmid, Emanuel Kitzelmann, Rinus Plasmeijer (Eds.): Proceedings of the ACM SIGPLAN Workshop on *Approaches and Applications of Inductive Programming* (AAIP'09), affiliated with ICFP 2009, Edinburgh, Scotland, September 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 81, Bamberg University, September 2009. ISSN 0937-3349.

Nr. 82 (2009) Ute Schmid, Marco Ragni, Markus Knauff (Eds.): Proceedings of the KI 2009 Workshop *Complex Cognition*, Paderborn, Germany, September 15, 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 82, Bamberg University, October 2009. ISSN 0937-3349.

Nr. 83 (2009) Andreas Schönberger, Christian Wilms and Guido Wirtz: A Requirements Analysis of Business-to-Business Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 83, Bamberg University, December 2009. ISSN 0937-3349.

Nr. 84 (2010) Werner Zirkel, Guido Wirtz: A Process for Identifying Predictive Correlation Patterns in Service Management Systems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 84, Bamberg University, February 2010. ISSN 0937-3349.

Nr. 85 (2010) Jan Tobias Mühlberg und Gerald Lüttgen: Symbolic Object Code Analysis. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 85, Bamberg University, February 2010. ISSN 0937-3349.

Nr. 86 (2010) Werner Zirkel, Guido Wirtz: Proaktives Problem Management durch Eventkorrelation – ein *Best Practice* Ansatz. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 86, Bamberg University, August 2010. ISSN 0937-3349.

Nr. 87 (2010) Johannes Schwalb, Andreas Schönberger: Analyzing the Interoperability of WS-Security and WS-ReliableMessaging Implementations. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 87, Bamberg University, September 2010. ISSN 0937-3349.

Nr. 88 (2011) Jörg Lenhard: A Pattern-based Analysis of WS-BPEL and Windows Workflow. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 88, Bamberg University, March 2011. ISSN 0937-3349.

Nr. 89 (2011) Andreas Henrich, Christoph Schlieder, Ute Schmid [eds.]: Visibility in Information Spaces and in Geographic Environments – Post-Proceedings of the KI'11 Workshop. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 89, Bamberg University, December 2011. ISSN 0937-3349.

Nr. 90 (2012) Simon Harrer, Jörg Lenhard: Betsy - A BPEL Engine Test System. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 90, Bamberg University, July 2012. ISSN 0937-3349.

Nr. 91 (2013) Michael Mendler, Stephan Scheele: On the Computational Interpretation of CKn for Contextual Information Processing - Ancillary Material. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 91, Bamberg University, May 2013. ISSN 0937-3349.

Nr. 92 (2013) Matthias Geiger: BPMN 2.0 Process Model Serialization Constraints. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 92, Bamberg University, May 2013. ISSN 0937-3349.

Nr. 93 (2014) Cedric Röck, Simon Harrer: Literature Survey of Performance Benchmarking Approaches of BPEL Engines. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 93, Bamberg University, May 2014. ISSN 0937-3349.

Nr. 94 (2014) Joaquin Aguado, Michael Mendler, Reinhard von Hanxleden, Insa Fuhrmann: Grounding Synchronous Deterministic Concurrency in Sequential Programming. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 94, Bamberg University, August 2014. ISSN 0937-3349.

Nr. 95 (2014) Michael Mendler, Bruno Bodin, Partha S Roop, Jia Jie Wang: WCRT for Synchronous Programs: Studying the Tick Alignment Problem. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 95, Bamberg University, August 2014. ISSN 0937-3349.

Nr. 96 (2015) Joaquin Aguado, Michael Mendler, Reinhard von Hanxleden, Insa Fuhrmann: Denotational Fixed-Point Semantics for Constructive Scheduling of Synchronous Concurrency. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 96, Bamberg University, April 2015. ISSN 0937-3349.

Nr. 97 (2015) Thomas Benker: Konzeption einer Komponentenarchitektur für prozessorientierte OLTP- & OLAP-Anwendungssysteme. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 97, Bamberg University, Oktober 2015. ISSN 0937-3349.

Nr. 98 (2016) Sascha Fendrich, Gerald Lüttgen: A Generalised Theory of Interface Automata, Component Compatibility and Error. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 98, Bamberg University, March 2016. ISSN 0937-3349.

Nr. 99 (2014) Christian Preißinger, Simon Harrer: Static Analysis Rules of the BPEL Specification: Tagging, Formalization and Tests. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 99, Bamberg University, August 2014. ISSN 0937-3349.

Nr. 100 (2016) Cedric Röck, Stefan Kolb: Nucleus - Unified Deployment and Management for Platform as a Service. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 100, Bamberg University, March 2016. ISSN 0937-3349.