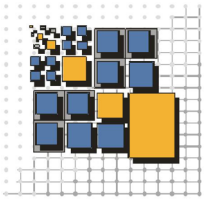

Otto-Friedrich-Universität Bamberg
Lehrstuhl für Praktische Informatik

Technical Report

Unified Deployment and Management for Platform as a Service

Contents

List of Figures

List of Tables

Listings

Abbreviations

aPaaS	application Platform as a Service
API	Application Program 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
iPaaS	integration Platform as a Service
ISV	Independent Software Vendor
JSON	JavaScript Object Notation
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
PaaSal	Platform as a Service abstraction layer
REST	Representational State Transfer
SaaS	Software as a Service
SME	Small and Medium Enterprises
SSL	Secure Sockets Layer
UML	Unified Modeling Language

1 Introduction

Cloud computing promises several advantages over classic models and has undoubtedly been one of the most hyped topics in IT-industry over the last couple of years. Nowadays, the still young Platform as a Service (PaaS) market is heavily fragmented, even though it is expected to consolidate within the next few years [NLP+11].
At least 77 different[1] PaaS offers are currently available at the market, according to the most advanced overview and comparison called PaaS Profiles[2]. Independent Software Vendors (ISVs) and Small and Medium Enterprises (SME) revealed in recent surveys why they adopted or plan to adopt the cloud in the first place. Most important, the users highlighted that they no longer have to pay for their infrastructure up front. Instead, they can consume the required resources like a utility, pay per use and therefore cut down their investment risks. In economics, this transformation is referred to as the shift from capital expenditure to operational expenditure. Further reasons for cloud adoption are the expected reduction of the total cost of ownership and the immediate access to new infrastructures, which consequently also shortens the time-to-market. [KPM13; The12; PHMH09; Bad12]
Driven by these benefits, cloud computing experienced steady growth over the recent years and is anticipated to grow even further [Pri10]. A study of Gartner predicts that cloud computing will reach the plateau phase of the Hype Cycle within only two to five years[3]. They recently also estimated a compound annual growth rate of 17 % between 2012 and 2017 on 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 is said to grow 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 cloud adoption 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. Another concern is the lack of standards between cloud providers, which hinders compatibility and fosters the chances of lock-in effects, especially the 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 your data. It bears the immediate risk of a decline in service quality, unjustified price increases and immense switching costs which can even threaten the survival of ISVs and SMEs when the vendor is changed [FK06]. Switching costs must not only be accounted for when voluntarily switching the provider, but can also arise rather unexpectedly in case of hostile takeovers, competitors leaving the market or the bankruptcy of a provider, making the provider change inevitable [MLBZG11]. Recent events highlight the market being under

[1] Services being offered as Open Source and hosted can have separate entries and are counted twice
[2] *Paasify - PaaS Profiles*. URL: `http://www.paasify.it` (Retrieved: October 31, 2014).
[3] *Gartner's 2014 Hype Cycle*. URL: `http://gartner.com/newsroom/id/2819918` (Retrieved: November 5, 2014).

consolidation, with takeovers[4] taking place and providers terminating their operations[5].

According to the providers' websites, as well as mentioned in various previous studies [PCR11; CCP14; BBSR13; CH09], the vast majority of PaaS providers offers its self-developed proprietary Application Program Interface (API) and tooling suite. Only a small number of the offers are somewhat compatible amongst each other, but mainly for the reason that they were built using the same product, e.g. an open sourced platform. The compatibility between providers 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 the granted interoperability, roll-out scripts and management applications require a specific implementation for each provider and changes must be made if the provider is changed. Only then are the scripts and applications capable of performing automated deployment, scaling and alike.

Whereas plenty of ways can prevent a vendor lock-in on the Infrastructure as a Service (IaaS) layer, e.g. by using the Open Virtualization Format or Deltacloud[6], PaaS offers are still prone to the lock-in. In order to enable a truly competitive market and unfold the full potential of cloud services, 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

[4] *Is PaaS becoming just a feature of IaaS?*. URL: `https://451research.com/report-short?entityId=79800` (Retrieved: November 5, 2014).
[5] *CloudBees Becomes the Enterprise Jenkins Company*. URL: `http://blog.cloudbees.com/2014/09/cloudbees-becomes-enterprise-jenkins.html` (Retrieved: May 3, 2015).
[6] *Apache Deltacloud*. URL: `http://deltacloud.apache.org` (Retrieved: October 28, 2014).

subscription-fee based and pay-per-use models, which is why they have an interest in preventing their customers to leave and join a competitor. [GS12]

Facing the current market situation, including the mentioned chances, obstacles and approaches, the research question is defined as follows: Is it possible to abstract the differences of vendor specific management and deployment interfaces on the PaaS layer by creating an intermediary abstraction layer?
Beyond the use of standards, abstraction layers are another feasible approach to overcome some of the relevant issues. By abstracting the differences of the proprietary APIs, only one unified interface is offered to the users. Concerning the mitigation of vendor lock-in effects and the reduction of barriers to cloud adoption, this thesis introduces the approach of identifying and utilizing a generic PaaS management and deployment API as a language independent interface. An easy to use prototype of the abstraction layer, which is supposed to enhance the interoperability and portability between PaaS cloud services regarding their deployment and management tasks, is created and evaluated.

The structure of this thesis is subdivided according to the major phases of the prototype's development. At first, all fundamentals will be explained in Chapter 2 to establish a common knowledge base among all readers. In Chapter 3, the basic approach is introduced to outline all required actions. Chapter 4 refines this basic approach and defines the detailed plan to guide the implementation phase on a theoretical basis. Important aspects of the prototype implementation and the most severe experienced issues are summarized in Chapter 5. The evaluation of the created prototype is reviewed in Chapter 6 to analyze whether the initial approach succeeded. Related works are outlined in Chapter 7 before Chapter 8 sums up the previous chapters and presents the final conclusion as well as the overall outcome.

2 Fundamentals

Essential topics for the understanding of the following chapters are going to be introduced in this chapter. After a brief definition of cloud computing in general, the terms interoperability and portability are defined. The Representational State Transfer, JavaScript Object Notation, lock-in effects and the general understanding of abstraction layers are introduced in further sections.

2.1 Cloud Computing

Cloud computing is a widely used term, despite no official definition exists. The vast majority of publications refers to the definition of the National Institute of Standards and Technology (NIST), which defines cloud computing as "[...] a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" ([MG11, p. 6]).
Marston et al. emphasized that the definition of the NIST does not cover all key characteristics and therefore defined cloud computing as follows:
"It is an information technology service model where computing services (both hardware and software) are delivered on-demand to customers over a network in a self-service fashion, independent of device and location. The resources required to provide the requisite quality-of- service levels are shared, dynamically scalable, rapidly provisioned, virtualized and released with minimal service provider interaction. Users pay for the service as an operating expense without incurring any significant initial capital expenditure, with the cloud services employing a metering system that divides the computing resource in appropriate blocks." ([MLBZG11, p. 177])

In general, the cloud model includes three distinct service models. These service models are typically referred to as layers and can be differentiated by the level of abstraction provided to the customer. [MG11, p. 6]

Infrastructure as a Service

IaaS is the most low-level layer. An IaaS provider typically only maintains the infrastructure, for instance housing, network connectivity, storage, CPU, memory and so on, whereas the customer has full control over the software stack, starting with the operating system. [Bad12; BBSR13]

Platform as a Service

Platform as a Service enhances the abstraction level even further. The offers typically provide a full development platform with hosting capabilities, which already includes frameworks, language runtimes, middleware services and other offered components. The consumer neither has control over the underlying infrastructure, nor does he have to main-

tain it. PaaS offers were formerly targeted especially at software developers, letting them focus on their products and innovation instead of having to maintain their infrastructure. Due to the lack of precise definitions, one can recognize more and more products being labeled as PaaS. According to the definition of Gartner, PaaS nowadays can be separated into further sub-categories, for instance application Platform as a Service (aPaaS) and integration Platform as a Service (iPaaS) [NLP+11]. Forrester, instead, distinguishes between Integrated Development Environments (IDEs) with cloud deployment capabilities, cloud IDEs, IDE neutral cloud runtimes and PaaS for business experts [RR11]. [Bad12; BBSR13; GS12; YBD08]

Software as a Service

Software as a Service (SaaS) is the highest of the three layers. Beyond the infrastructure, it also abstracts the platform details as well as most application details, except for some user-specific configurations. Therefore, the user does not have to take care of application updates and only consumes the software, for instance web-based email clients like Google Mail, Microsoft's Office 365 or even social networks like Facebook. [Bad12; BBSR13]

2.2 Interoperability and Portability

Even though both terms, portability and interoperability, are sometimes falsely used as substitutes in the context of cloud computing, they clearly have to be distinguished. [Lew13]
Portability is mostly referred to the capability of migrating a cloud application between different platforms without the need to rewrite or modify the application and its environment. [OF10; Int93]
Interoperability is defined as the exchange of information between two or more systems, paired with the ability to semantically process and use the exchanged information. [IEE90]

2.3 Representational State Transfer

The Representational State Transfer (REST) is an architectural style that was initially described in Roy Thomas Fielding's PhD thesis on *Architectural Styles and the Design of Network-based Software Architectures*. [Fie00] Over the last decade RESTful applications became more and more popular but the definition is still very often misunderstood and applications are falsely claimed to be RESTful[7].

The communication described by REST is stateless. All messages are to be self-descriptive and resources must be uniquely addressable. It is representation oriented, which means that actions are performed on representations of a resource, showing either the current or intended state. One of the key constraints defined in REST is Hypermedia as the Engine of Application State (HATEOAS). Amongst others, it describes that links are used to connect the data and clients can discover what to do with messages based on their content. The Hypertext Transfer Protocol (HTTP) currently is the most frequently used REST implementation. [Fie00]

[7] *REST APIs must be hypertext-driven.* URL: `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven` (Retrieved: April 30, 2015).

2.4 JavaScript Object Notation

The JavaScript Object Notation (JSON) is a language independent, lightweight and human-readable data-interchange format. Each JSON document is described using valid JavaScript syntax with a default encoding of UTF-8. More details on the JSON specification can be obtained from RFC 7159. [Bra14]
JSON schema files can be used to describe and validate JSON documents. [GZC13]

2.5 Lock-In Effects

Besides the extensive description of lock-in effects within competing markets, a lock-in effect can also be defined in a much smaller scale, for instance when customers have to choose a product or a technology [Art89]. This so called vendor lock-in is characterized by the incompatibility of products, goods or services, the switching costs and network effects that bind customers to this vendor. The product cannot be changed without inconvenience and significant costs so that the customer is dependent and incapable of responding to predictable or unpredictable changes, for instance increasing service fees. [FK06]

In the computer industry, the vendor lock-in[8] is used to refer to the intentional or unintentional incompatibility of hardware, software or file formats. Focusing on the context of software development, the lock-in effect can also describe a situation that prevents users to migrate assets to another provider. This can happen if either the migration is not possible with reasonable effort, for example due to the immense migration costs caused by the need to reimplement parts of the software, or because of technical incompatibilities as it could be the case with missing middleware services.

2.6 Abstraction Layer

"The essence of abstractions is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context." ([Gut13, p. 43])
Following this citation, the purpose of an abstraction layer in computing can be, first and foremost, defined as the reduction of complexity by hiding specifics. An example would be to hide the implementation details of the underlying layer. Usually an abstraction layer is presented as a single interface, which integrates all the underlying heterogeneous interfaces and translates the high-level call to the one or more specific low-level calls. The creation of an abstraction layer requires a common set of functionalities in the underlying layer that can be abstracted.[9,10] [Tan06]
Well-known examples of abstraction layers are hardware abstraction layers, separating the operating system from the hardware, as well as database abstraction layers that unify the communication between the software and a number of database products.

[8] *The Linux Information Project: Vendor Lock-in Definition.* URL: `http://www.linfo.org/vendor_lockin.html` (Retrieved: April 30, 2015).
[9] *Abstraction Layer.* URL: `http://en.wikipedia.org/wiki/Abstraction_layer` (Retrieved: April 30, 2015).
[10] *Abstraction Layer Definition.* URL: `http://www.pcmag.com/encyclopedia/term/37353/abstraction-layer` (Retrieved: April 30, 2015).

3 Approach

Building and deploying PaaS applications is fairly easy, especially due to the help of guidelines that are available on the Internet. Using the provider's Command-Line Interface (CLI) and API tooling, the setup of a user's complete development environment can be tailored to integrate with the PaaS platform. However, changing the provider becomes more and more complicated the further the application is integrated into the surrounding environment, for instance if continuous deployment or continuous delivery are used.
The goal of this master's thesis, the release of a publicly available abstraction layer for PaaS management and deployment APIs, shall help to simplify these migration efforts. From now on, Platform as a Service abstraction layer (PaaSal) is going to be used as codename for the prototype. The abstraction should be realizable given that the management and deployment operations of PaaS systems share the same semantics, but only use different syntax [SR10]. With PaaSal, it is planned to focus especially on the management and deployment capabilities of the providers' APIs, while neglecting the functional interfaces [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 neglected. 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 mostly resulted in complex brokering scenarios that exceed this work's scope.

CAMP, one of the most promising PaaS standards that is currently in development, could also be utilized to specify the abstraction layer's API. Nevertheless, it was decided not to use CAMP for a couple of reasons. Besides the fact that CAMP is in its early stages, which alone would require enormous customization effort to integrate it with a couple of PaaS platforms, there is no reference implementation available yet. The release of the first implementation[11] is yet to be announced, although plans originally intended to make it available before the end of 2014[12]. Despite the decision to not use CAMP, it should be feasible to integrate CAMP into PaaSal once it reached a higher maturity level and a reference implementation becomes available. More information about CAMP can be found in Chapter 7, where the related work is introduced.

Before beginning with the design of the prototype and its implementation, several aspects have to be analyzed carefully. In the following subsections, the most important aspects of the approach to build the prototype are going to be described. Section 3.1 outlines the applied criteria to select a set of PaaS vendors and presents the chosen vendors. Their APIs are evaluated and compared in Section 3.3, based on identified similarities and differences. Finally, Section 3.4 reveals the intended structure of the prototype.

3.1 Vendor Selection Criteria

Especially since a prototype is going to be created in this work, only a few selected PaaS vendors and their product can be supported. Which vendors are to be supported depends on many criteria, e.g. the supported management interfaces, deployment types, runtime

[11] *nCAMP - The CAMP Management Service, a minimal implementation of the CAMP 1.1 specification.* URL: `http://ec2-107-20-16-71.compute-1.amazonaws.com/campSrv/` (Retrieved: May 5, 2015).
[12] *CAMP - Meeting Minutes 15th October 2014.* URL: `https://www.oasis-open.org/committees/download.php/54389/Minutes%2015th%20October%20%202014.txt` (Retrieved: May 5, 2015).

languages, the vendor's popularity within the community and the plans that are offered by the providers. It is intended to cover as much heterogeneity as doable 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.

The 14 most relevant providers and their system's characteristics are evaluated in Table 1. The information were obtained from the PaaS Profiles and by evaluating the provider's documentation. Not production-ready providers are not included in the evaluation, excluding, for instance, Apache Stratos, Flynn and Nitrous.io. AppFog was not included due to the fact that it shall be replaced by V2, which is completely based on Cloud Foundry.

Provider	System	CLI	CLI Deployment	HTTP Deployment	Git Deployment	GUI Deployment	Free Plan	Runtimes	Extensible	Automatic Scaling	Horizontal Scaling	Vertical Scaling	Isolation
App42[13]	-	✓	✓	✗	✗	✓	t	Groovy, Java, Node.js, PHP, Python, Ruby	✗	✗	✓	✓	Kontena
AWS Elastic Beanstalk[14]	-	✓	✓	✓	✗	✓	t	.NET, Java, Node.js, PHP, Python, Ruby	✗	✓	✓	✓	VMs
IBM Bluemix[15]	cf	✓	✓	✓	h	✗	t	Java, Node.js, Ruby	✓	✗	✓	✓	Warden
cloudControl[16]	cc	✓	✓	✗	✓	✓	✓	Java, Node.js, PHP, Python, Ruby	✓	✗	✓	✓	LXC
Cloud Foundry[17]	-	✓	✓	✓	✗	✗	p	Go, Groovy, Java, Node.js, Ruby, Scala	✓	✗	✓	✓	Warden
Dotcloud[18]	cc	✓	✓	✗	✓	✗	✓	Java, Node.js, Perl, PHP, Python, Ruby	✓	✗	✓	✓	LXC
Exoscale[19]	cc	✓	✓	✗	✓	✗	✗	Clojure, Java, Node.js, PHP, Python, Ruby, Scala	✓	✗	✓	✓	LXC
Heroku[20]	-	✓	✗	✓	✓	✗	✓	Clojure, Groovy, Java, Node.js, PHP, Python, Ruby, Scala	✓	✗	✓	✓	LXC
HP Helion[21]	cf	✓	✓	✓	✗	✗	t	Go, Groovy, Java, Node.js, Ruby, Scala	✓	✓	✓	✓	Warden
Jelastic[22]	-	✗	✗	✓	✓	✓	t	Java, Node.js, PHP, Python, Ruby	✓	✓	✓	✓	?
Microsoft Azure[23]	-	✓	✓	✓	✓	✗	t	.NET, Java, Node.js, PHP, Python, Ruby	✓	✓	✓	✓	Hypervisor
Nodejitsu[24]	-	✓	✓	✓	h	✗	✗	Node.js	✗	✗	✓	✓	SmartOS Zone
Openshift V2[25]	-	✓	✗	✗	✓	✗	✓	Java, Node.js, Perl, PHP, Python, Ruby	✓	✓	✓	✓	Warden
Stackato[26]	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: Vendor selection criteria and the characteristics of the evaluated PaaS providers

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

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 3.2 finally presents the chosen vendors.

3.1.1 Management Interface Criteria

A basic search in the documentation of the 14 PaaS providers revealed three types of clearly distinguishable management interfaces exist. Most providers offer a web interface that allows to adjust numerous settings of the platform. The web interface is usually supported by a web based API and operating system specific command-line tools. Some vendors even provide IDE plugins and language wrappers to manage their PaaS environment, e.g. for Java or Ruby.
In contrast to web interfaces, APIs and CLIs are programmatically by far easier to use. Moreover, CLIs, language wrappers and IDE plugins all use the platform's API themselves. Consequently all other interfaces are neglected and the focus is set on using a platform's API in this prototype. As a result, management interfaces don't have any further influence on the vendor selection.

3.1.2 Deployment Type Criteria

One of the main challenges in creating PaaSal is likely to be the deployment task. Analyzing the providers, as outlined in Table 1, three distinct deployment methods that are offered by the providers could be identified: Command-line tools, deployment via Git[27], or the use of HTTP commands against the platform's API. All additional possibilities, e.g. deployment via Ant[28], Maven[29] or Gradle[30] tasks, as well as IDE plugins, utilize one of these three approaches themselves.

Command Line Interface

Most of the providers offer any kind of CLI to their users. Within 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 mostly 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. However, the invocation of these tools could be realized in

[27] *Git*. URL: `http://git-scm.com/` (Retrieved: October 29, 2014).
[28] *Apache Ant*. URL: `http://ant.apache.org` (Retrieved: October 29, 2014).
[29] *Apache Maven*. URL: `http://maven.apache.org` (Retrieved: October 29, 2014).
[30] *Gradle*. URL: `http://www.gradle.org` (Retrieved: October 29, 2014).

plenty of programming languages, for instance Groovy[31], Java[32] or Ruby[33].
Besides the providers that offer deployment via their CLI tools and are listed in the table, also AppFog V1[34] and Google's AppEngine[35] use this method. Heroku offers deployment via their CLI tool only for prepacked containers, so called slugs.

Git

The second identified deployment method relies on the use of Git repositories and locally installed Git binaries. An executed 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 commands from within an application, for instance if using Ruby[36], Groovy[37] or Java[38].
Some other platforms that allow the usage of Git commits as upload mechanism are Amazon's Elastic Beanstalk[39] and Google AppEngine.

HTTP

Even though all platforms offer HTTP management APIs, nearly none 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.
This approach is used by Cloud Foundry and also by Heroku, which partially supports the deploy operation via the RESTful API in addition to the deployment via Git. The files to be deployed via Heroku's API must already be packed into a so called slug. Slugs are not allowed to request further dependencies[40].

3.1.3 Runtime Criteria

The runtimes supported by a system are one of the most essential parts when choosing which PaaS to use. Table 1 not only lists the supported runtimes by each provider, but also highlights whether a PaaS is extensible to support additional runtimes. The most

[31] *Executing External Processes From Groovy.* URL: `http://groovy.codehaus.org/Executing+External+Processes+From+Groovy` (Retrieved: May 6, 2015).
[32] *Executing operating system commands from java.* URL: `https://blog.art-of-coding.eu/executing-operating-system-commands-from-java` (Retrieved: May 6, 2015).
[33] *Calling shell commands from Ruby.* URL: `http://stackoverflow.com/a/2400/1009436` (Retrieved: May 6, 2015).
[34] *AppFog.* URL: `http://appfog.com` (Retrieved: October 31, 2014).
[35] *Google AppEngine.* URL: `https://appengine.google.com` (Retrieved: October 29, 2014).
[36] *Ruby Git lib: ruby-git.* URL: `https://github.com/schacon/ruby-git` (Retrieved: May 6, 2015).
[37] *Groovy Git lib: grgit.* URL: `https://github.com/ajoberstar/grgit` (Retrieved: May 6, 2015).
[38] *Java Git lib: jgit.* URL: `http://eclipse.org/jgit` (Retrieved: May 6, 2015).
[39] *AWS Elastic Beanstalk.* URL: `http://aws.amazon.com/elasticbeanstalk` (Retrieved: October 29, 2014).
[40] *Creating Slugs from Scratch.* URL: `https://devcenter.heroku.com/articles/platform-api-deploying-slugs?preview=1` (Retrieved: May 5, 2015).

popular approaches to allow additional runtimes are Heroku's buildpacks and Openshift's cartridges. Both approaches have already been adopted by other vendors.
An evaluation of the natively supported runtime languages[41] lists Java, PHP, Ruby, Node.js and Python as best supported languages, with all five being offered by more than 50 % of the providers.
Based on this criteria, it was decided that the vendors to be used in the prototype must have one common language, either natively supported or by extension, to allow simple and consistent application deployment during the development.

3.2 Chosen Vendors

Beyond the previously defined requirements for eligible vendors, only vendors whose PaaS can be used without any further costs are considered for the final selection. Whereas only a few providers offer a free plan, most providers offer at least a free trial that would also be sufficient. The open-sourced Cloud Foundry and Openshift can even be installed locally.

Evaluating the summed-up facts, the following four vendors were chosen to be included in the PaaSal prototype: cloudControl, Cloud Foundry, Heroku and Openshift. More relevant details about the vendors have been collected in Table 2. The column *complete* reveals if all methods are fully documented and nothing is left blank. 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 overall impression of the documentation.

	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: Chosen PaaS vendors to be used with PaaSal

Comparing the four platforms against each other, several differences and commonalities can be noticed. All platforms share a similar understanding of applications, domains, services and variables. However, Openshift noticeably differs with the use of its cartridges in contrast to the otherwise used buildpacks. Based on the API design and the used objects, Heroku and cloudControl are more alike when compared to Cloud Foundry. This similarity is most obvious for the service object and its operations, which are nearly identical on both platforms. On the contrary, Openshift and Heroku are the only two vendors to support multiple deployment regions. In the following the chosen vendors, their platform and the most important aspects to be regarded are introduced.

cloudControl

cloudControl is the product of a German based start-up that distributes its PaaS also as white-label product. It was not only chosen as contrast to the big players, but also because of the free plan and its usage of buildpacks. Some other companies, namely

[41] *PaaS Profiles - Runtime Language Statistics*. URL: `http://www.paasify.it/statistics/languages` (Retrieved: May 6, 2015).

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, CenturyLink Cloud, HP Helion, IBM Bluemix, Pivotal IO, 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 of the chosen vendors that does not support the deployment via Git, instead HTTP commands must be used to upload the data. Authenticating against the API, solely OAuth 2[42] can be used.
Using the API, just 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 nicely designed API. The API documentation is complete, up-to-date and provides extensive examples, making it a rather pleasant experience to work with the API. Authentication can be achieved using OAuth 2.
Besides these points, Heroku was chosen as it is the source of the buildpack technology, offers a free plan[43], multiple deployment regions and matches all defined criteria.

Openshift

Besides Cloud Foundry, Openshift is another popular open-source PaaS. In PaaSal Openshift V2 shall be used, but V3 is expected to be released anytime soon. Openshift is included mainly because of the cartridge technology, which allows to add additional runtimes and services to the PaaS. Furthermore, Openshift offers a free plan, can be installed locally and supports automatic scaling. It also allows the application deployment into one of several deployment regions around the globe.
Opposed to the other three vendors, Openshift does not use token based authentication, but requires the credentials to be submitted via the HTTP Basic-Authentication headers. The documentation is complete and provides good examples, but was not updated for quite some time. The self-describing API itself lists plenty of operations that are not included in the documentation.

[42] *OAuth 2.0*. URL: `http://oauth.net/2/` (Retrieved: May 7, 2015).
[43] The free plan was significantly downgraded in May 2015, subsequent to our decision. See also: `https://blog.heroku.com/archives/2015/5/7/new-dyno-types-public-beta`

3.3 Vendor API Evaluation

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

Most vendors provide their own, non-standardized API so that there are huge 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 already defined requirements and operations that are shared between PaaS providers and should be supported by homogenized PaaS APIs.
D'Andria et al. [DBCAZ12], Sellami et al. [SYMT13] and Cunha et al. [CNS14] all defined operations that cover the application lifecycle with *start* and *stop*, as well as the application's Create Read Update Delete (CRUD) methods. Cunha et al. also covered to add services, scale the application and access the application's logfiles as shown in Figure 16. Sellami et al. focused especially on application environments and D'Andria et al., compare to Figure 13 in the related work chapter, specialized on the migration of application between platforms.

In addition to these already defined methods, the application's domain and variable objects are further essential parts of PaaS offerings that are not yet covered. We also find it reasonable to provide additional methods for the application scaling, logging access and service management.
Following a detailed evaluation of cloudControl's[44], Cloud Foundry's[45], Heroku's[46] and Openshift's[47] API documentations, Table 3 was created to list all API resource objects and their 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 listable and retrievable, whereas the general operations for the application include the listing and creation.
Operations that belong to a specific application entity are the main 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. In the scaling group, automatic scaling of applications is also supported by Openshift only. Logging includes four methods, the `list` and `get`, as well as the log download and the retrieval of application statistics. Statistics are currently not supported by cloudControl and Openshift.
In contrast to the definitions of previously cited publications, environments, monitoring and database actions were intentionally left out. Database actions are rather specific and

[44] *cloudControl API doc.* URL: `https://api.cloudcontrol.com/doc/` (Retrieved: May 7, 2015).
[45] *Cloud Foundry API doc, v206.* URL: `http://apidocs.cloudfoundry.org/` (Retrieved: May 7, 2015).
[46] *Heroku Platform API Reference, state of April, 20th 2015.* URL: `https://devcenter.heroku.com/articles/platform-api-reference` (Retrieved: May 7, 2015).
[47] *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).

a slightly technical requirement. Monitoring is out of scope as it is not planned to evaluate the platforms' performance and the operations aren't management related either. Environments are management related, but not widely supported. It is not planned to add artificial features to the platforms, but rather to homogenize the currently offered capabilities.

A well-known challenge that could arise are semantic conflicts between the different vendor interfaces [LKT11]. One semantic conflict is the special application lifecycle within cloudControl, where the applications cannot be stopped or be put into maintenance. A further semantic conflict lies in the handling of the application object. Cloud Foundry and Heroku allow to modify the application after its creation, but cloudControl and Openshift do not offer this option. In conclusion, cloudControl and Openshift do not allow to change or update the application's runtime at all. Another issue 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.

Belongs to	Group	Functionality	Description	Cloud Foundry v2	Heroku	cloudControl	Openshift v2
App. operations	App	GET	Get an app entity	✓	✓	✓	✓
		DELETE	Delete the app	✓	✓	✓	✓
		UPDATE	Update the app	✓	✓	✗	✗
	Lifecycle	REBUILD	Rebuild, e.g. to use updated buildpacks	✓	✓	✓	✓
		UPLOAD	Upload the actual app data	✓	✓	✓	✓
		DOWNLOAD	Download the current app data	✓	✓	✓	✓
		START	Start the app	✓	✓	✗	✓
		STOP	Stop the app	✓	✓	✗	✓
		RESTART	Restart the app	✓	✓	✗	✓
	Scaling	AUTOSCALE	Enable / disable auto-scaling	✗	✗	✗	✓
		ADD INSTANCE	Add new instance, scale horizontally	✓	✓	✓	✓
		REMOVE INSTANCE	Remove instance, scale horizontally	✓	✓	✓	✓
		SCALE	Set instance power level	✓	✓	✓	✗
	Domains	CHECK NAME	Is the domain name available	✗	✗	✗	✗
		LIST DOMAINS	List all the app's domains	✓	✓	✓	✓
		GET	Get domain entity	✓	✓	✓	✓
		ADD DOMAIN	Assign domain to the app	✓	✓	✓	✓
		DELETE	Delete and remove the domain	✓	✓	✓	✓
		UPDATE	Update the domain settings	✓	✓	✓	✓
		SET CERTIFICATE	Apply the domain's SSL certificate	✗	✗	✗	✓
		REMOVE CERTIFICATE	Remove the SSL certificate	✗	✗	✗	✓
	Variables	LIST VARS	List all env. variables of the app	✓	✓	✓	✓
		CREATE VAR	Create a variable with initial value	✓	✓	✓	✓
		UPDATE VAR	Update an existing variable's value	✓	✓	✓	✓
		DELETE VAR	Remove a variable	✓	✓	✓	✓
		GET VAR	Get an environment variable entity	✓	✓	✓	✓
	Logging	LIST LOGS	Collect the app's logfiles	✓	✓	✓	✓
		GET SPECIFIC LOG	Get a specific log file	✓	✓	✓	✓
		DOWNLOAD LOGS	Download all logs as archive	✓	✓	✓	✓
		GET STATISTICS	Get some app statistics	✓	✓	✗	✗
	Services	ADD SERVICE	Install and bind to the app	✓	✓	✓	✓
		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 app name available for use	✗	✗	✗	✓
		CREATE	Create the app	✓	✓	✓	✓
		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: API operations and whether they can be realized with the vendor

3.4 Initial Layout

The initial layout of the architecture for PaaSal is outlined in Figure 1. To all potential users, PaaSal shall appear as one language independent API, abstracting all logic hidden underneath. It is planned to divide PaaSal internally 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 would be best to request only a username password combination. Further authentication modes, for instance token based authentication, should be supported with these credentials as well.
Below the authentication layer, it is intend to directly place the basic operations that can be used by all so-called adapters. Each adapter is the implementation of the yet to be specified API of the abstraction layer. The maintainability and extensibility of the prototype shall be achieved with the use of individual adapters per vendor. Shared operations could be, for instance, archive file handling, Git repository actions and alike.
Below the fundamentals, the matching layer guarantees to automatically resolve and always use the valid adapter for the user's PaaS.
The final layer, the adapters, makes sure that the behavior of each system is equal. Adapters translate the generic operations to a sequence of operations being invoked on the requested platform's API and eventually the Git repositories associated with an application.

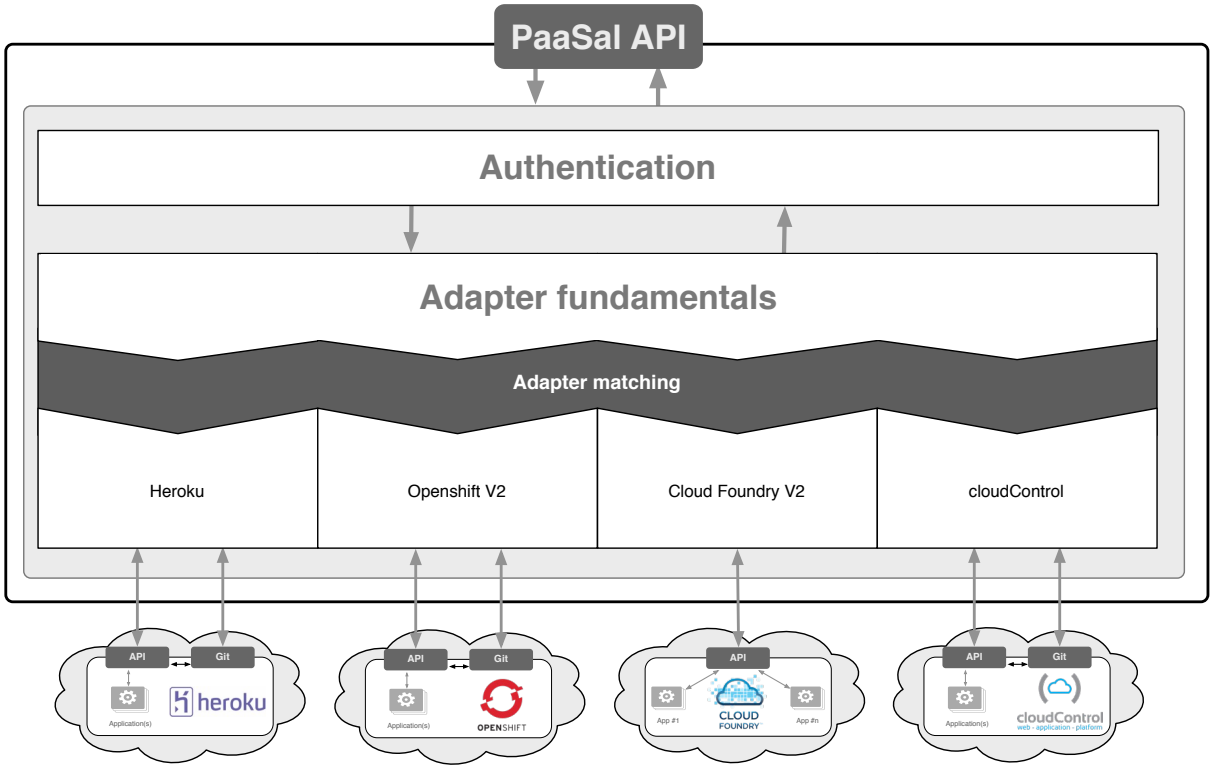

Figure 1: Initial layout of the proposed PaaS abstraction layer

4 Design

In this chapter, the detailed design of PaaSal is defined. The basic approach, which was outlined in Section 3, gets further refined so that an implementation-ready API specification is available up to the end of this chapter.

One of the most important requirements for the API is to provide a programing language independent abstraction layer. In alignment with all the chosen vendors, we decided to create a RESTful API with the use of HTTP, which assures programing language independence.

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 platforms 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 and Heroku are examples 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 urgently 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 uses the Cloud Foundry platform. 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 solved 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 the other feasible approach of presenting static adapter configurations, this technique allows to add, update or remove providers and endpoints at runtime. This feature is especially useful if offering the abstraction layer as a hosted service or if private clouds, for instance a local Cloud Foundry deployment during development, are to be used.

Figure 2 illustrates the associations between the vendors, providers and endpoints in detail, visualized as Unified Modeling Language (UML) class diagram. 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 common attributes, for instance a name, unique ID and timestamps. The `Endpoint` object includes additional attributes, especially 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 4.2 and 4.3.1.

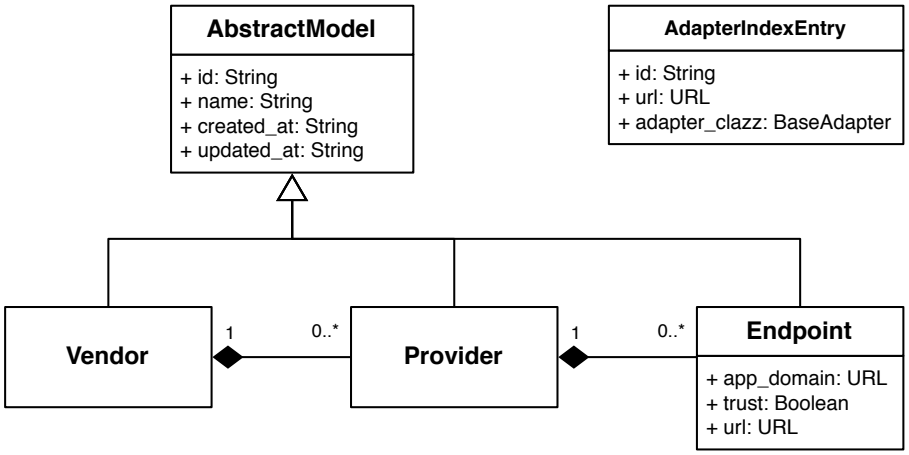

Figure 2: Class diagram showing the associated Vendor, Provider and Endpoint objects

In the remainder of this chapter, each of the following sections focuses on important parts of the API. Section 4.1 introduces the understanding of all API objects and also induces the common application lifecycle model. In Section 4.2 the previous definitions are summarized to illustrate the big picture of the planned API. Thereafter, Section 4.3 describes all API operations with their pre- and post-conditions. Finally, the actual API abstractions are presented in Section 4.4. The first part of the abstractions is made in Section 4.4.1, which describes the mapping of the vendor's API objects to PaaSal's objects. Thereafter, Section 4.4.2 presents which operations need to called on the platforms to create the desired behavior, completing the API's definitions and the design chapter.

4.1 API Objects

Emanating from the identified operations of Section 3.3, this section of the abstraction layer's design 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.

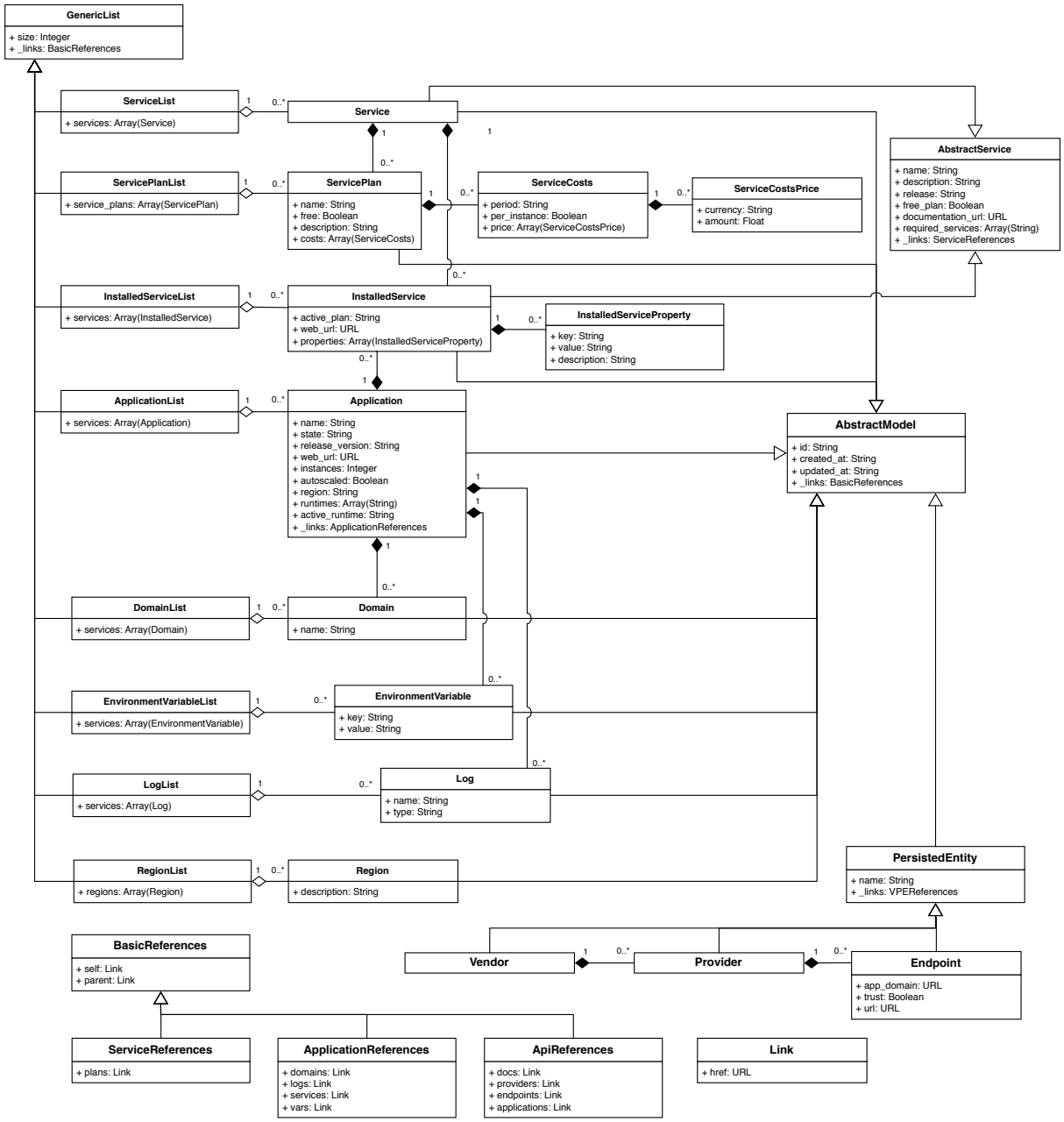

Figure 3: API objects class diagram

Following various best practices that describe how to properly build an API[48,49,50,51], 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 HATEOAS principle of RESTful applications, those requirements lead to the `AbstractModel` class, which possesses all those

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

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

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

[51] *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).

four properties and is to be inherited by all other API response objects.
References are required to be of the type `BasicReferences` or one of its inheriting classes. The `BasicReference` has two links, the `self` and the `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, for instance in case of the `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 4.2 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 the previously named classes. Their purpose is to normalize the way how the API responds with object collections. Therefore, all `List` objects inherit the `GenericList` with 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.

4.1.1 Region

Regions refer to a specific geographic deployment location that are offered by some, but not all providers. A provider decides which regions are offered to its customers. The platform only determines if the feature is supported at all.
The initial region object is defined with only a `description` attribute, apart from the basic ID and timestamps. Restrictions are applied on some platforms, for instance Openshift has a dedicated location for Node.js deployments, but are missing formalizations that could be included in the object. It was therefore decided to use the `description` attribute for those comments and leave the validity assertions to the platform itself for now.

4.1.2 Service

Services are additions to the actual program, sometimes even dependencies, that can be installed and bound to an application. Popular examples for services are data stores, monitoring tools, logging utilities, notification, security 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, PaaSal's initial service object is explicitly restricted to bound services which are part of exactly one application. Services being already bound to an application are described later on in Section 4.1.3.4 with a dedicated object.
Identified attributes of a service, which are shared amongst all evaluated platforms, are assigned to the `AbstractService`, 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.

4.1.2.1 Service Plans

Service plans always belong to exactly one service. They describe the conditions under which the service can be used and the price to 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 are not specified to be retrievable 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 in the Boolean `per_instance` property. The `ServiceCostsPrice` class belongs to exactly one cost object and describes the `amount` 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.

4.1.3 Application

The `Application` object can be seen as the core object of 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 application is to adjust its number of active instances automatically to the current or expected load. The `instances` property shows the number of application instances that are created for the application.
All the application's child objects as well as the lifecycle of a deployed binary application are presented in the following sections.

4.1.3.1 Application Lifecycle

An application usually undergoes several stages, all of which are displayed in the `state` property of the application object. Managing the application and its states requires a detailed knowledge of the state transitions and the overall lifecycle of the application. In the first place, each of the four platforms is based on a slightly different understanding of the application object.
Nonetheless, it was managed to identify a generic PaaS application lifecycle complying with all four platforms. The lifecycle, which is illustrated by 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 of the six states. On cloudControl, for instance, the presence of the CRASHED state could not be confirmed.

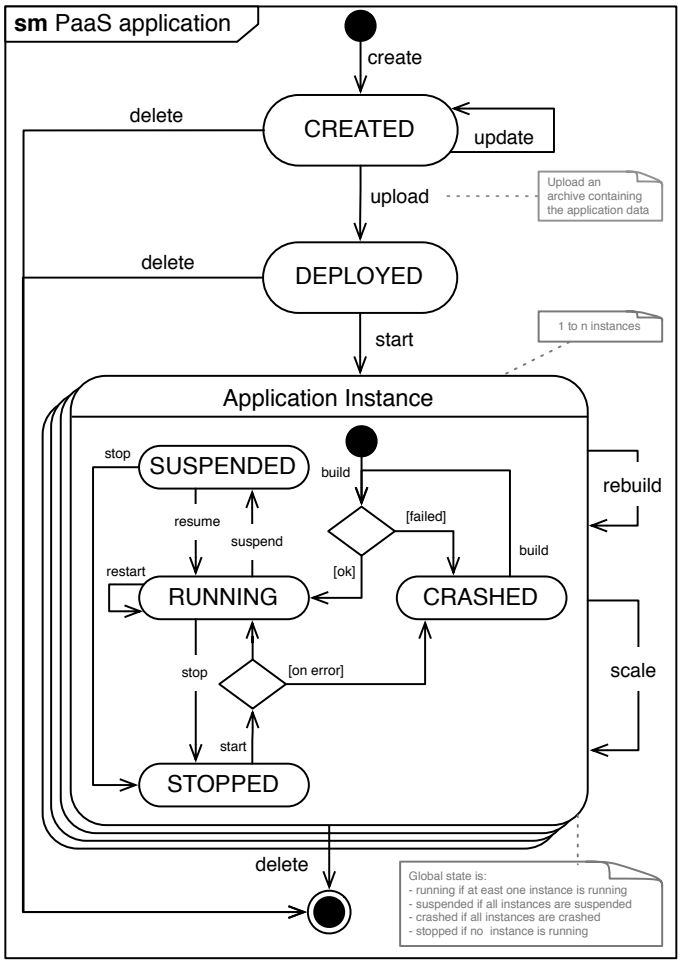

Figure 4: Generic PaaS application lifecycle as UML state diagram

Initially the application must be created, upon which its state shall be CREATED. In order to switch 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 get 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 build process can be executed at two positions in 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 keep 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 IDLE. 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 of 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 same state.

4.1.3.2 Domains

Domains are one of the objects that are bound to 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 evolved features, for instance the assignment of SSL certificates to use HTTPS connections, is not yet offered by the APIs of the evaluated platforms.

4.1.3.3 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 those key-value pairs are often loaded into the PATH via the bashrc file, if for instance using a UNIX system. The variables often serve as configuration parameters, for instance containing credentials or destination addresses of third party systems.
The `EnvironmentVariable` class of the abstraction layer's API has two specific properties, the `key` and `value`. Applications can retrieve the `value` from the system's PATH via the `key` that must be unique per application.

4.1.3.4 Installed Services

In Section 4.1.2 the `Service` class, representing common middleware functionalities that can be added to an application, was introduced. This section now focuses on the

`InstalledService`, describing a service that was already installed and bound to the 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 tells where the web interface of the service can be found. 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 `EnvironmentVariable` they possess the `key` and `value` property. Their `description` contains additional information on the usage of the variable.

4.2 API Structure

Given the planned API operations that were introduced in Chapter 3 and the objects of the previous section, the big picture of the abstraction layer slowly emerges. This section now presents the setup of the PaaSal API from a global perspective.
At first, the associations between objects and operations to make all required CRUD operations available 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 PaaSal 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 shown as rectangles. Resources with the green background indicate that a collection is represented, whose objects can be returned as 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 made available. 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 represents 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.
The arrows are the connections between the 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 on the figure is separated into four dedicated API groups.

The PaaSal group is illustrated in the top left of the API resource map in Figure 5. Beneath the chosen version of the PaaSal API, the resources of all three objects are the only ones with a top level path. All resources in this group are public and do not require authentication.

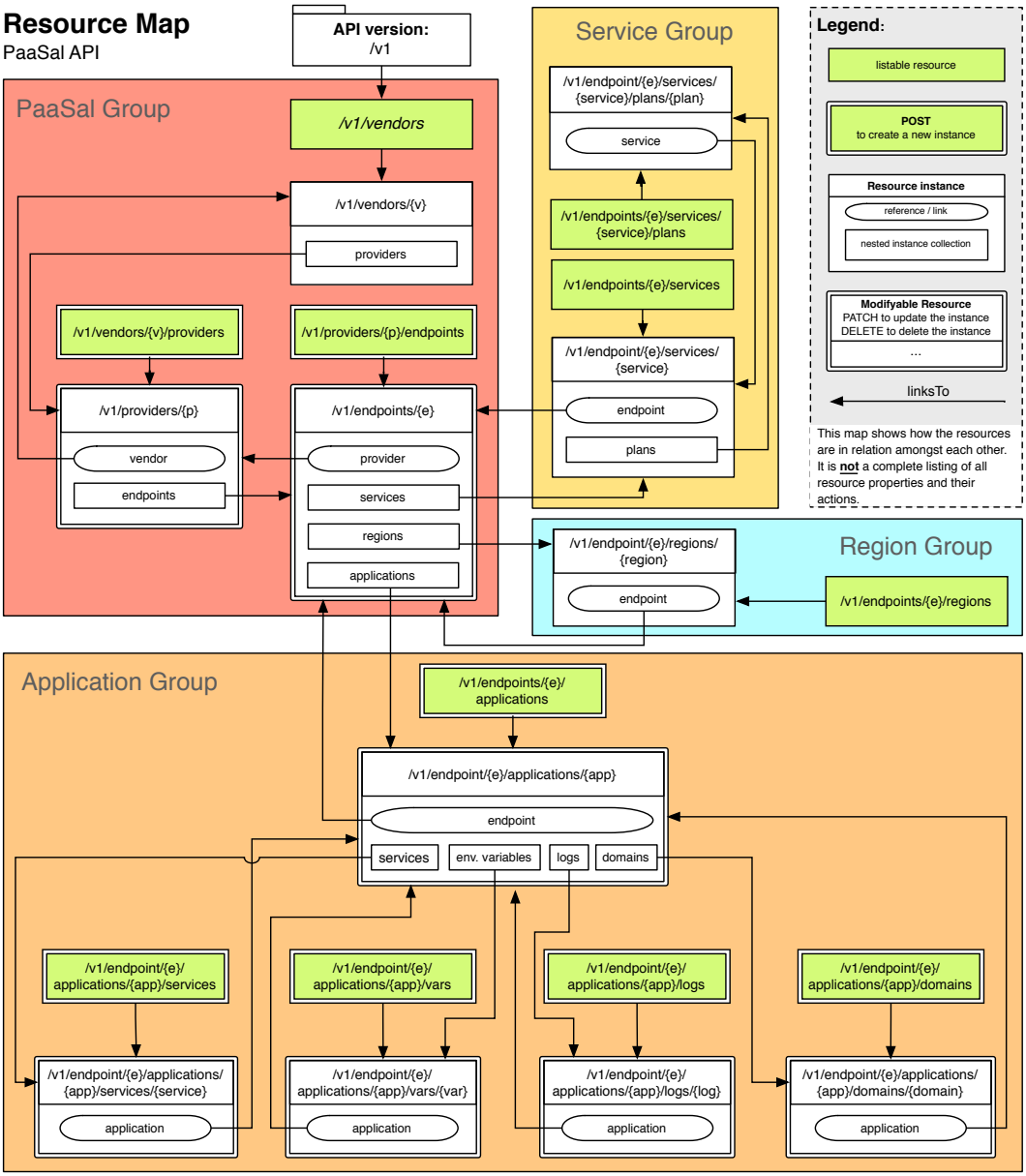

Figure 5: PaaSal API - Resource Map

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 represent the logic to communicate with their platform and must be implemented in 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.

A dedicated group being focused only on providing information about the available services was created with the API service group. Valid authentication credentials of the chosen endpoint are required. Service objects can be loaded via the list resource as collection or as specific instance via the service's ID and strictly belong to the referenced

endpoint. All subordinated service plans, which belong to exactly one service, can be retrieved via the shown 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, neither can objects be deleted or created, nor updated. Services are maintained by the provider of the endpoint.

The retrieval of all available deployment regions is maintained in the dedicated 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 ten resources, namely 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 is nested below the endpoint, whereas the domains, logs, variables and installed services are nested below the endpoint and the application to which they belong. All the application's child objects link to their parent application and are referenced in the application object themselves.

4.2.1 Authentication

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

With the abstraction layer's API being web-based, the HTTP Basic authentication was chosen to be used. 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 are defined to be placed in the 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 final Authorization header for the exemplary username MyUsername and its password MyPassword.

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

```
1 Authorization: Basic TXlVc2VybmFtZTpNeVBhc3N3b3Jk
```

4.2.2 Versioning and Accept Header

The PaaSal API follows the semantic versioning specification[52]. PaaSal shall allow to serve multiple versions of the API at once and provide legacy support. Each non-backward compatible change of the application, for instance 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 by default.
To request a specific API version, the user must specify the version in the HTTP Accept header and refer to the vendor 'paasal'. Code Listing 2 shows the 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 PaaSal

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

4.2.3 Message Formats

The outgoing messages of the API can be categorized within 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 4.1.
In an object's presentation, discrete child objects that can also be retrieved via dedicated API resources shall generally not be included. For instance, the application's representation shall not include any information about its variables, domains or services. However, as described in Section 4.1, all objects with child objects include references pointing to these collections and can be used to navigate 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 being passed on 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 narrow the error down.

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

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

dev_message The developer message, containing insight explanations why the error 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 documentary at which one can explain the insights of the error and present detailed resolution approaches.

Error Status Codes

The error codes of HTTP generally distinguish between five types, amongst others the user related errors and 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 PaaSal's 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 for writing requests if not all required parameters were specified or contained invalid values.

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

404 - Not Found The resource or the object could not be found. This error can occur with an unknown URL, 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 could not 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 within PaaSal or just being forwarded from the active endpoint. Especially the errors 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 PaaSal 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 passed on from the platform. Usually the platform 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.

Information on returned status codes of valid and successful requests can be found in Section 4.3.

4.3 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 the app is being suspended. Successful responses of `GET` requests are always expected to have the status code 200.
The `PATCH` method is used solely to update already existing objects. Parameters shall be optional and only the filled fields are to be changed on the object. Similar to the `GET` method, the HTTP status code 200 is used for successful operations.
With the `POST` method, one can not only create new objects but also trigger actions on existing objects, for instance to change the applications state. In case of successful actions, the status code 200 shall be returned. However, if a new object was created, the code has to be 201.
Objects can be deleted by the use 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 PaaSal 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 used 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 3, 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 the Boolean `autoscale` property of the `Application` class. The functionality is unchanged.

Domains/CHECK NAME & Application/CHECK NAME Checking whether 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 to 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 being 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 5.10.

Table 4 is an exemplary table, showing 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 the response.
Parameter names, being shown in the second column of the *Parameter(s)* row, are also used to indicate if they are optional or required. Optional parameters are printed in italic font, whereas required parameters are in bold letters. The third column in the parameters row determines where the parameter must be made available. Which object type must be provided is shown in the fourth column. An additional description to the value is presented in the final fifth column of the parameter.
In the remaining rows, the first column defines the content of all columns to their right. The second row, for instance, shows the URL path at which the operation can be called. Curly braces indicate variables which must be populated with dynamic values.
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: API operation table format explanation

4.3.1 Vendor, Provider and Endpoint CRUD Operations

At the beginning of Chapter 4, 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 listing 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 with the abstraction layer and utilize the vendor's platform. This operation also requires the vendor's identifier but 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 in which 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 is defined by the status code 204.
Not shown equivalent operations 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

4.3.2 Service and Service Plan Operations

Services which are offered to be installed on applications are maintained solely 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

4.3.3 Region Operations

Similar to the service group, regions are also maintained by the provider only. Table 8 contains all operations to show 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

4.3.4 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 autoscaling option can be specified. According to the providers used for our prototype, application names must be unique, not only for the user but amongst all applications that are registered at the endpoint. Once the application is created, it shall be in the initial CREATED state.

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 included in Table 10. The first entry presents the operation to deploy application data, also known as data upload. 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 sometimes still ongoing deployment after the request returned, wherefore the application object may not instantly reflect the new state. The application data which is to be uploaded shall be included in the body of the request as compressed zip or tar.gz archive.
In the second entry, all instructions to download previously deployed application data are summarized. The response upon the `GET` request is not one of PaaSal's API objects, but the compressed application data in the desired archive format, which 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 and fails otherwise. 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, archive containing the application 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

Lifecycle Management

Managing the lifecycle of an application is supported via 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 three operations have different postconditions that shall apply after the operation finished. After either the start or restart operation was invoked, the application shall switch to the RUNNING state. Likewise, the stop operation shall cause the application to change to the STOPPED state.

If any of the commands fails to be executed, the application usually proceeds to the CRASHED state. Depending on the platform's supported states, the exact behavior may vary slightly 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

Based on the ADD INSTANCE, REMOVE INSTANCE and SCALE operations which were defined in the approach, only a single scale operation is defined in Table 12 to cope with all needed scaling actions.
To add or remove application instances, the operation can be executed with the `instances` parameter being set to the desired number of instances. If the number is greater than the currently available instances, additional instances will be created. In case the number is smaller, the 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 the bad request.
Vertical scaling as a feature is not yet included in this prototype, opposed to the definition of the approach. Further information about vertical scaling, the issues and possible options are going to be presented as future work in Section 8.1.

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

4.3.5 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 and presents them in two tables, whereby Table 13 contains the operations to retrieve the child objects or collections 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 URL path has to be nested below the 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. Those operations would then return the requested object instead of a complete list. The final entry of the table illustrates this behavior. Both definitions can be translated to the other objects 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 defined, 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 in multiple groups before, all form parameters are optional for update requests. In addition to the update, all objects can also be deleted if using HTTP's `delete` method with requests against specific object instances.

4.3.6 Application Logging Operations

Log objects are currently the only application child object that allows 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 logfiles of an application, bundled as compressed archived. Which archive format shall be used for the binary response can be specified in the `archive_format` query parameter. Entry number two also returns a binary archive whose type can be specified, but contains only one specific logfile. Next, the third entry shows the operation to load the contents of a specific log. The response is not an object, but the actual content of the logfile 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 on the endpoint.

GET	**Log**	**Binary archive data with all logfiles**		**200**
URL path	/endpoints/{endpoint_id}/applications/{application_id}/logs/download			
Description	Download all logfiles 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 logfile**		**200**
URL path	/endpoints/{endpoint_id}/applications/{application_id}/logs/{log_id}/download			
Description	Download a specific logfile			
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 logfile			
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

4.3.7 Vendor Specific Parameters

Even though the Platform as a Service abstraction layer is designed to harmonize the actions of all supported platforms, there still are vendor specific parameters that might have to be passed to the platform from time to time. One example to emphasize the need of this feature is vertical scaling on Openshift. Despite the fact that one can always adjust

the number of application instances, vertical scaling is not supported and the performance of an application instance must be decided when creating the application. This behavior is unique among all four supported platforms and in favor to the fact that this parameter of Openshift's API is optional and uses a default value, no parameters had to be made available within the abstraction layer to determine this performance level.
By means of using 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.
As of now, there is no support for vendor specific parameters in other request types or operations being called with the `POST` method. Resources that accept the `vendor_specific` parameters key are: Application, Domain, Variable and InstalledService.

Code 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 in this example to force Openshift to use a different gear than the default small size.

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\"}}}"
```

4.3.8 Custom Endpoint API Calls

With all the previously introduced API operations, the majority of similar operations on all platforms is already supported. However, there can also be situations in which those operations will not cover all requirements of the abstraction layer's users.
As an illustration, one can analyze the release management of Heroku's platform API, which is not supported by PaaSal. Releases describe a working combination of code, variables and services that are persisted and to which one can always roll-back. In order to allow the release management and other operations, regardless of their support within PaaSal, the feature of custom endpoint API calls is defined. Those calls can either be directed at the API in general, or against a specific application. Form and body parameters that are included in the native API requests are passed on to the endpoint without validation or modification. The response message and eventually arising errors are also the unfiltered response of the endpoint. All custom API calls benefit from the authentication being managed by PaaSal.

Custom calls that target the endpoints API have to be called as pointed out in Listing 4. The `ENDPOINT_API_CALL_PATH` must thereby 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. The URL must then 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 that is described by Heroku itself, whereas the second and fifth line show the equivalent calls if made via PaaSal.

Listing 6: Heroku API call vs. PaaSal 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}
```

4.4 API Mappings

This section describes the mappings of objects and operations to achieve the abstraction layer and to harmonize the presentation as well as functionality 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 in order to perform one of PaaSal's actions on the platform.

4.4.1 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.

4.4.1.1 Region Mapping

Multi-region support is not provided by all platforms. Within the provider selection, it was shown that cloudControl and Cloud Foundry do not support this feature, whereas Heroku and Openshift rely on it. 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 have this feature. 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 PaaSal's 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

4.4.1.2 Service Mapping

Services in general, also referred to as addons or cartridges, are available on all four platforms. Mapping the attributes is simple, with only the `free_plan` attribute requiring more logic. 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 addons. Timestamps and a description are not available at all. Addons do not have dependencies and the documentation URL can be created based on the addon's name. An addon has a free plan if at least one of the addon's options has a `thirty_days_price` that is 0.
Cloud Foundry has a rather complex service model, but fully supports PaaSal's 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 similarly as with 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

4.4.1.3 Service Plan Mapping

On cloudControl, a service's 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 addon. 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, there is no function to determine it via the provider's API. As of now this construct is working as there is no known cloudControl provider to use more than one currency.
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 standardization. Pivotal IO specified how to access service plans via the API in their documentation[53]. 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 PaaSal's 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.

Platform / Attribute	**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	✗	
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 marktplace is available.

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

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

4.4.1.4 Application Mapping

PaaSal's 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 on cloudControl only one runtime can be applied per application, the `runtimes` field shall 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[54] 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 static configuration as there is no approach to identify the default domain at which applications shall always be available.

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 4.4.1.4			

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

Most of Heroku's application mappings can be obtained from the application object.

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

Mapping rules must be considered with 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 being 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 once created, wherefore there is also no update timestamp. Nevertheless, the creation timestamp can be taken and get mapped to this field, too. 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. Identifying the `region` can be achieved 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.

Platform / Attribute	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 4.4.1.4			

Table 22: Application object attribute mapping: Heroku & Openshift

Application State Detection Rules

In addition to the generic PaaS application lifecycle, which was initially introduced in Section 4.1.3.1, the application object mapping requires a set of rules explaining how an application's state can be derived.

Figure 6 visualizes the initial detection rules for all four platforms in the form of decision trees. The rules cover the most common states as they are described in the platform's documentation or were identified during the examination of the platform.
Openshift's application states can be detected to the most extend using only the gears. Other needed information are the number of kept deployments and the deployments themselves. Most challenging is the detection of the DEPLOYED state, precisely the initial deployment. Summing up the number of deployment activations, excluding those of the initially available deployment, reveals this state. 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.

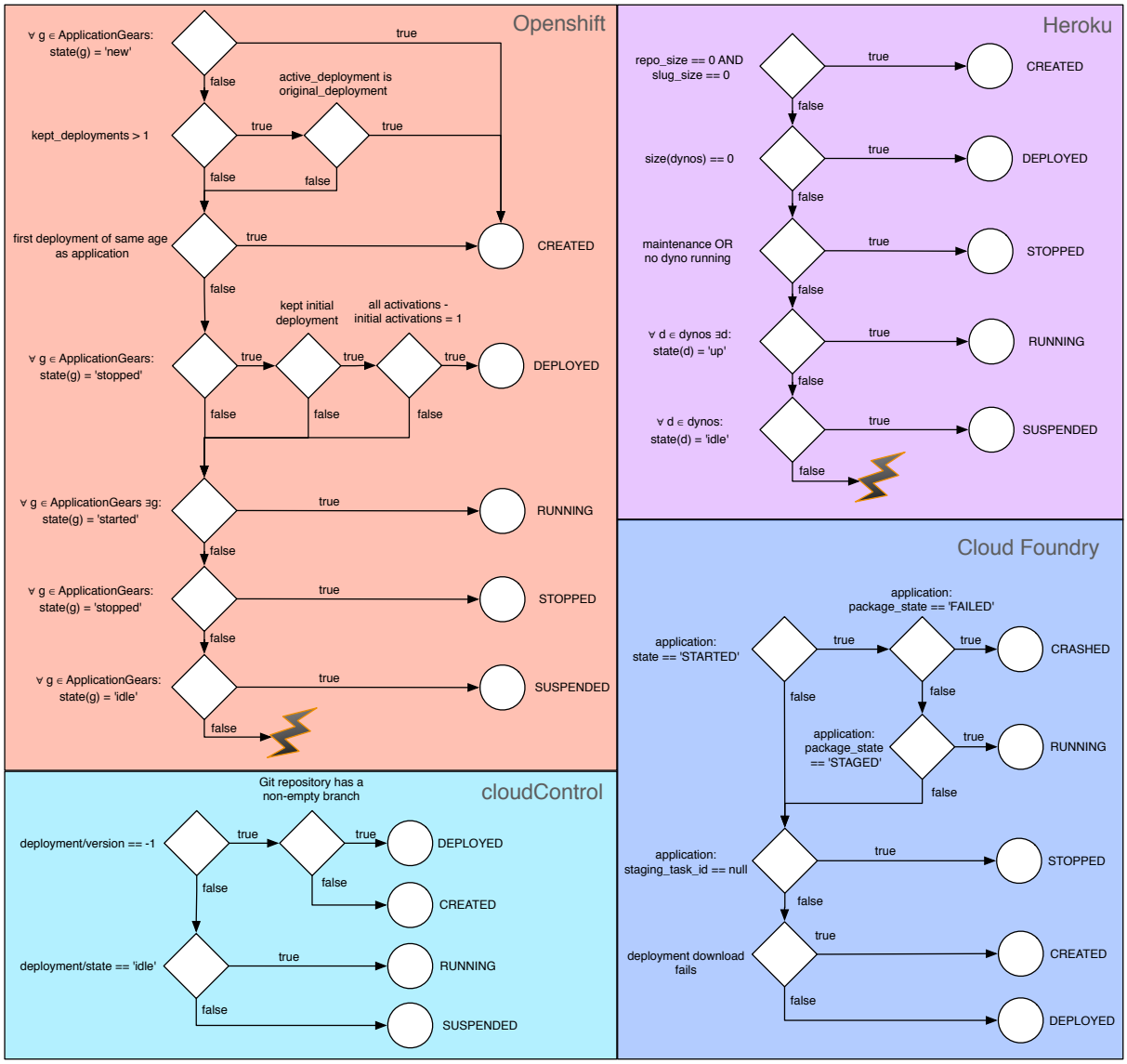

Figure 6: Application state detection rules

Heroku's detection rules first evaluate whether there has been any data upload. 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. Finally, if none of the previous rules should apply, the application state could not be detected.
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. Finding any branch on the repository points to the DEPLOYED state. The application has only been CREATED if there is no branch.
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 create the set of rules. Being started, the application is CRASHED if the package state is failed and RUNNING if the package state is staged. A staging task ID with the value null points

to the application being STOPPED. If a requested deployment data download returns a URL where the data could be downloaded from, the application is DEPLOYED. Receiving an error for the download request indicates the application being in the initial CREATED state.
All in all, 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 PaaSal.

4.4.1.5 Domain Mapping

Mapping the domains can be achieved as described with the rules being shown in Table 23. One obvious fact is the different naming. Heroku uses a naming identical to PaaSal's. On cloudControl and Openshift, the domains are labelled as an 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 their 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	host	domain	hostname	alias	id
			domain	name				

Table 23: Domain object attribute mapping

4.4.1.6 Environment Variable Mapping

On all four platforms, the mapping of PaaSal's 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 and can only be gathered from the listing. In these cases, the ID and key of PaaSal's environment variable is the key of the returned key-value pair, whereas the value is the pair's value matching the key.
By means of using environment variables with 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.

Platform / Attribute	**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

Openshift is the only platform in the selection that makes variables available as dedicated API objects. The variable's `name` is used as `ID` and `key`, the value is already applied.

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

4.4.1.7 Log Mapping

On all selected platforms, there is no API operation to list the available logs. For this reason, most of the returned objects are statically mapped. Furthermore, the log entries are passed on without modification so that there is also no mapping need.
In case of cloudControl, only the timestamps are taken from the application to provide at least an approximate information.
Similarly to the solution on cloudControl, the logs of Cloud Foundry are also mapped statically to the most extend. All log files which are written to the application's `logs` directory are included with their original filename. The creation time is being taken from the application and the update timestamp always uses the current time.
For Heroku, the creation timestamp is also using the application's timestamp, whereas the update uses the current time.
The logging functionality of Openshift is not implemented in this prototype. Not all information would have to be static as available logfiles and their timestamps can most likely be identified from the files written to the application's file system.

4.4.1.8 Installed Service Mapping

Most of the mappings are identical to those of the general services being described in Section 4.4.1.2.

For a proper mapping on cloudControl, two objects are needed: the general addon 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.
Cloud Foundry requires the service binding object to be identified, which includes or points to all other required objects, e.g. the service instance. The ID of the binding object is also used for PaaSal's 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 are 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. Therefore, a static mapping alias default plan is used. If available, properties are already included in the installed cartridge.

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

4.4.2 API Operation Mapping

In the previous section,s 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 in order to achieve one of the abstraction layer's actions. In short, the mappings describe how to get the operations output based on the given input data. All 37 operation mappings, except for those of the public Vendor/Provider/Endpoint API, 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, hence save one additional request. Openshift and cloudControl do not allow to retrieve specific services or service plans, but they can 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.

Subsequent to the region operations, 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 separate applications and shall default to *paasal* if the application was created with the abstraction layer. Also if loading the application list, all

gathered application objects must be joined with the corresponding deployment object. If creating an application on cloudControl, one runtime, the application name and 'git' as repository type must be specified with the application to be created. Similar to when loading an application, the deployment must also be taken care of if creating an application. The initial deployment must always be created in order to guarantee the expected functionality on cloudControl, whereby its name shall default to *paasal*. 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* addon. For the reason that the *config.free* addon can only be added with at least one initial value, a Boolean variable named *paasal-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. This ID must therefore be gathered before firing the request to create the application. Thereafter, 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.
Application mappings for Heroku are not 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 the application object that is compatible to PaaSal. 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 finished 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 to be called. 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 is the automatically executed 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 is allowed to 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 if 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, the data processed and finally get 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 workers 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. The user data is thereby 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.

On the subject of domain mappings, Heroku and Openshift can be used without adaptions. With cloudControl not all essential information are contained in the returned alias collection, wherefore each of the list's elements must be retrieved individually, before returning the mapped response.
In opposite to these three platforms, severe mapping efforts are required on Cloud Foundry to handle the 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 is to be deleted 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 addons must be loaded, be combined with the separately retrieved addon assignment object itself and finally be enriched with the information of the general addon description. The addon assignment object as well as the general addon object are necessary to populate the response object, as described in Table 26 of the installed service object mapping. In order to be able 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 addons 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 first create a new instance of the service and second 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 it is also specified in the PaaSal 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 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 parts of the PaaSal 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 tells up from where messages 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 com-

bined with the statically defined logs that are provided by the *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[55] 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 from. 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.
Openshift's log files cannot be accessed in this prototype. All operations that would be needed to identify or retrieve the logs are completely SSH based.
All in all, 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.

4.4.3 API Request Mapping

With the knowledge of all definitions made in the previous parts of this chapter, one already knows how to translate the PaaSal API operations to the vendor's API and the way in which response objects can be build properly. Nonetheless, one aspect is still missing to conclude the definitions, namely the headers of HTTP requests against the platforms. Headers are not only essential to enable the authentication against the endpoint, but also define the API version of the platform to be used and its expected response format.

In Section 3.2 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 is to be posted towards the gathered URL path must define the GRANT_TYPE and include the credentials. If the token is to 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 PaaSal 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

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

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 as a dedicated token refresh is not supported with this method.

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 with every request against the determined endpoint. On all platforms, the exchanged content type is expected to equal JSON. All platforms expect the authentication information, either credentials in case of Openshift or the API token for the other three platforms, to be placed in the `Authorization` header.
Cloud Foundry must also be provided with the `Basic` field, 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 the additional `Accept` header to be provided with all requests. As in the definition of the PaaSal 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

4.4.4 API Design Challenges

Reviewing the disclosed API mappings, most challenges that were experienced while specifying the PaaSal 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 be copied from related objects and are expected to present similar values. As most failed mappings provide only metadata, 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 somehow be harmonized in the adapters. Custom solutions could be mapped if the provider is known, but for most providers this information must be obtained first. Even then would this approach remain unstable and likely to break if anything is changed or additional providers appear with their new custom schema.

One of the first noticeable challenges was the completely inconsistent definition of the deployment and build process. While some differences, for instance different deployment methods, were to be 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 need the build to be invoked with a separate API request. 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 addons, as cloudControl refers to them. To circumvent any arising conflicts, both addons are now added to the application immediately upon its creation. Both addons are free of costs, wherefore there are also no known downsides to 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 the application has been deployed, it cannot be taken offline except for deleting the application or its data. However, a slight adaption of the lifecycle made all platforms compatible and provides more flexibility for further platforms to be added. As shown in the final definition of the lifecycle, compare to 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 for errors unrelated to 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 is to be done if the application is started and otherwise, the build will be triggered so that the application instance gets started.

Application instances are another aspect that is regarded very different by the platforms. At first it was planned to allow the adjustment of application instances within the range from zero up to the limitation of the endpoint. This intention was however troubled by cloudControl, Cloud Foundry and Openshift, which all require at least one application instance to always be present. Despite the limitation to 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. Major side effects are not to be expected as the application can still be stopped to perform maintenance work or save the costs of the remaining application instance.

5 Prototype

Chapter 5 introduces the final prototype, explains its architecture and presents important implementation details. Utilized technologies are summed up in Section 5.1 and the final structure of the project is explained in Section 5.2. All steps and files related to the initialization and loading of the code parts are demonstrated in Section 5.3. In Section 5.4, the extensible API route setup is introduced, whereupon Section 5.5 discusses the adapters and their hierarchy. Section 5.6 illuminates the handling of Git repositories with its challenges and the appropriate solutions, before Section 5.7 presents the custom errors and a general approach to exception handling. All actions that are involved in an API request are highlighted in Section 5.8, before Section 5.9 reveals the test setup of the prototype. Information about the documentation and the most severe issues during implementation are explained in sections 5.10 and 5.11. Finally, Section 5.12 presents the instructions how to use the current prototype, adapt the default configuration and develop new adapters.

5.1 Technology

Upon an analysis of the API design, all related platforms and their provided libraries, it was decided that the PaaSal prototype is going to be implemented in Ruby[56]. Ruby is an open-source, dynamic, object-oriented and general-purpose programming language and is available on 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 programing language Ruby is very often connected to the Ruby on Rails[57] web framework. One reasonable choice to build the API would have been the Rails::API[58] 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[59] based infrastructure was chosen instead. Rack is a modular and extensible interface, wrapping HTTP requests and responses into a single method cal,l and is implemented by a variety of different application servers. After a careful consideration of all requested features, the Grape[60] project was chosen to build the API with. Grape is 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[61]. With its extensions grape-entity[62] and grape-swagger[63], it provides a feature rich development environment to develop the PaaSal API in an object-oriented and properly documented manner. Two

[56] *Ruby.* URL: `https://www.ruby-lang.org/en/` (Retrieved: June 25, 2015).
[57] *Ruby on Rails.* URL: `http://rubyonrails.org/` (Retrieved: June 25, 2015).
[58] *Rails::API.* URL: `https://github.com/rails-api/rails-api` (Retrieved: June 25, 2015).
[59] *Rack: a Ruby Webserver Interface.* URL: `http://rack.github.io/` (Retrieved: June 25, 2015).
[60] *grape: An opinionated micro-framework for creating REST-like APIs in Ruby.* URL: `https://github.com/intridea/grape` (Retrieved: June 25, 2015).
[61] *Sinatra.* URL: `http://www.sinatrarb.com/` (Retrieved: June 25, 2015).
[62] *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).
[63] *grape-swagger: Swagger compliant documentation of your grape API.* URL: `https://github.com/tim-vandecasteele/grape-swagger` (Retrieved: June 25, 2015).

well-known projects which already rely on grape to build their API are Art.sy[64] and Blinkist[65].

Grape supports multiple message formats out of the box, for instance XML and JSON. In analogy to the supported platforms, the first version of the PaaSal API being developed in this prototype shall utilize the lightweight and readable JSON as message format for both, request and response messages.
With the use of the Hypertext Application Language (HAL)[66], 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 PaaSal's 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": "paasal",
  "active_runtime": "Ruby",
  "runtimes": [],
  "region": "eu",
  "autoscaled": false,
  "instances": 0,
  "web_url": "https://paasal.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" }
  }
}
```

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

[64] *Art.sy*. URL: `https://www.artsy.net/` (Retrieved: June 25, 2015).
[65] *Blinkist*. URL: `https://www.blinkist.com/` (Retrieved: June 25, 2015).
[66] *HAL - Hypertext Application Language*. URL: `http://stateless.co/hal_specification.html` (Retrieved: June 3, 2015).

and libraries. All supplementary Ruby gems that are used within the prototype are listed in the subsequent Table 30, including a short description why they are needed.

Gem	Version	Usage description
configatron	>= 4.5	Used as global configuration
daybreak	>= 0.3	DB store used on UNIX systems
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: Gem dependencies at runtime with their usage

Those dependencies, which are maintained by Bundler[67], are also to be found in the `paasal.gemspec` file of the prototype. The prototype of the PaaSal API 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.

5.2 Project Structure

Utilizing the technologies that were described in the previous section, the internal structure of the project is shaped mostly by the guidelines of Bundler[68]. The root directory of the deliverable has the structure as it 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 PaaSal 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

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

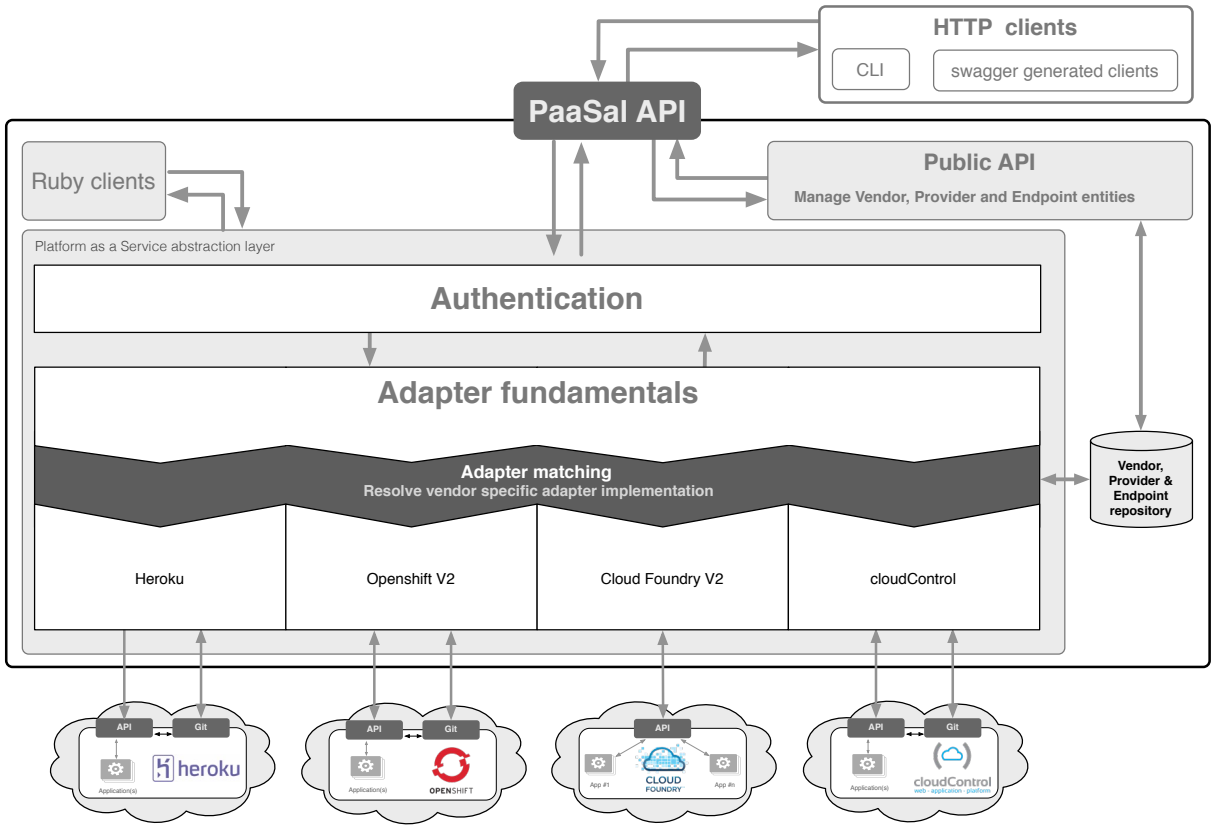

Figure 7: Final architecture of the PaaSal project

server with a variety of configuration options, e.g. to run the server as a daemon in the background.

Initial configuration files of PaaSal 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 directory contains the actual code and application logic. It is divided into two subdirectories, PAASAL to contain the actual abstraction layer and PAASAL_API to contain only the RESTful web API.

PAASAL 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 to be invoked when starting or stopping the abstraction layer.

The directories of the PaaSal API are: API to include all routes, the entities, etc., 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[69] API documentation and should therefore be moved to the API gem.

All Kwalify[70] compatible YAML schemata reside in the SCHEMAS directory. The schemata

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

[70] *Kwalify: A parser, schema validator and data binding tool for YAML and JSON.*. URL: `http://www.`

are used to validate the adapters implementing all required methods and their configuration files being in the correct format. 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. 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.

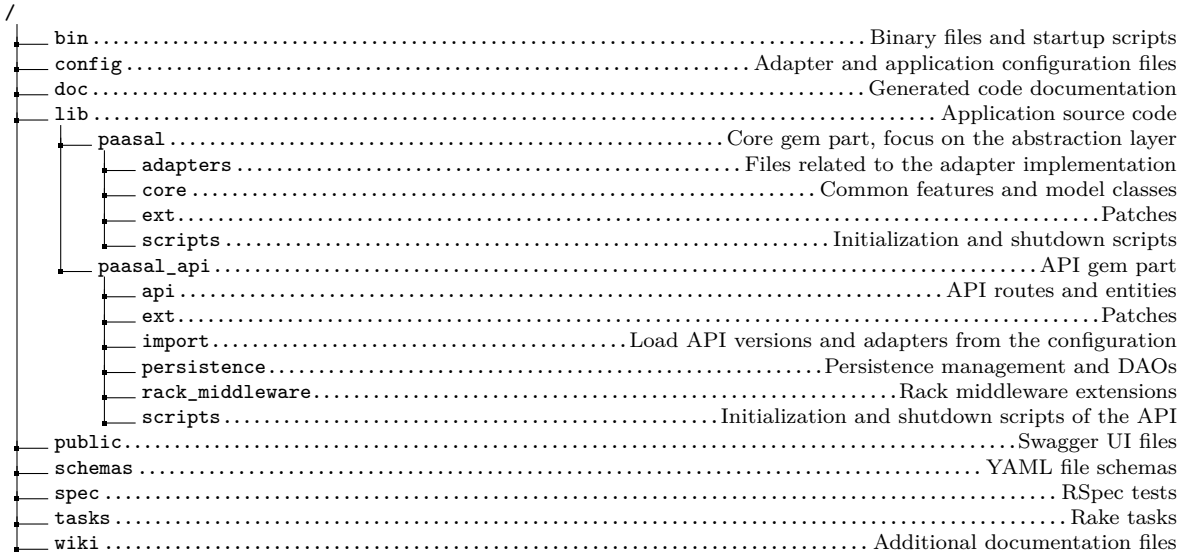

Figure 8: PaaSal's file system project structure

5.3 Initialization

The PaaSal API requires multiple actions to be executed before the web server can be made available. In general, the application is started via the provided `paasal` 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 PaaSal 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: PaaSal's 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 'paasal/scripts/setup_config'
6  # Load application
7  require 'paasal_api/scripts/load_api'
8  # Initialize the application
9  require 'paasal_api/scripts/initialize_api'
10 # Initialize the Rack environment
11 require 'paasal_api/scripts/rack_application'
12 # finally start the application
13 run Paasal::API::Rack.app
```

`kuwata-lab.com/kwalify/` (Retrieved: June 28, 2015).

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 multitude of 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 to execute the block immediately before the application is shut down.

Listing 9: PaaSal API initialization script

```
begin
  # Shutdown hook to cleanup the API
  require 'paasal_api/scripts/shutdown_api'

  # include the configuration of the project to overwrite the home dir config
  project_dir_config = '../../../config/paasal_config.rb'
  if File.exist?(File.expand_path(project_dir_config, File.dirname(__FILE__)))
    puts "Applying configuration from: #{File.expand_path(project_dir_config, File.dirname(__FILE__))}"
    require_relative project_dir_config
  end

  # now load the configuration values
  require 'paasal/scripts/initialize_config_defaults'

  require 'paasal_api/scripts/initialize_api_customizations'

  require 'paasal_api/scripts/initialize_daos'

  # finalize so that the configuration is locked
  require 'paasal/scripts/finalize'
rescue Paasal::StartupError => e
  log.error "PaaSal API startup failed (#{e.exit_code}), exit now"
  exit e.exit_code
end
```

Listing 10: PaaSal shutdown hook

```
# Implement API shutdown actions, tidy up the DB
at_exit do
  puts '-----------------------------------------------', ''
  puts 'Cleaning up the API...'

  # delete the SSHHandler generated files
  puts ' ... delete SSH files ... '
  paasal_config.ssh.handler.cleanup if paasal_config.key?(:ssh) && paasal_config.ssh.key?(:handler)

  if !paasal_config.db.key?(:delete_on_shutdown) || paasal_config.db.delete_on_shutdown
    if File.exist?(paasal_config.db.path) && File.directory?(paasal_config.db.path)
      FileUtils.rm_rf(paasal_config.db.path)
    end
    puts ' ... DB store successfully deleted' unless File.exist?(paasal_config.db.path)
  end
  puts ' ... done!'
end
```

Next, the project's configuration is applied in the `initialize_api.rb` file. As there are multiple options how to specify PaaSal's configuration, details are presented in Section 5.12.2.
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 5.5.3, which explains how to create new platform adapters. With the finalization script, the configuration is ensured to be locked and the initialization phase is completed.

5.4 API Route Setup

Every grape API route has to inherit the `Grape::API` class. The root node of PaaSal's API is shown in Listing 11 and inherits this class. Grape helpers are a convenient way to provide commonly used functionality to all API routes. As for all other definitions being are made with grape, declarations are always available to all routes and mounted API classes that follow the declaration. The first `before` block will therefore be executed in advance to every route and the `rescue_from` block applies to all routes of the PaaSal API, handling the conversion of internal errors to the defined error response object.
With the `mount` command of line 31 the actual operations of the abstraction layer's first version are included 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, unless this is prohibited by additional configuration.
The `RootAPI's` concluding block, the `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: PaaSal API root

```
1  module Paasal
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 Paasal::API::V1::Base
32
33        route :any, '*path' do
34          if env['paasal.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 \'paasal\' 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
```

Being mounted in the `Paasal::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 4.4.3 with use of the `Paasal::Adapters::AuthClient` class, which acts similar to an interface to the individual clients, for instance the `OAuth2AuthClient`.

Listing 12: PaaSal API authentication protected routes

```
1  module Paasal
2    module API
3      module V1
4        class Auth < Grape::API
5          # ... http_basic authentication ...
6
7          ### Mount all protected routes below
8          mount Paasal::API::V1::Calls
9          mount Paasal::API::V1::Regions
10         mount Paasal::API::V1::Applications
11         mount Paasal::API::V1::ApplicationData
12         mount Paasal::API::V1::ApplicationDomains
13         mount Paasal::API::V1::ApplicationEnvVars
14         mount Paasal::API::V1::ApplicationLifecycle
15         mount Paasal::API::V1::ApplicationLogs
16         mount Paasal::API::V1::ApplicationLogsTail
17         mount Paasal::API::V1::ApplicationScaling
18         mount Paasal::API::V1::ApplicationServices
19         mount Paasal::API::V1::Services
20         mount Paasal::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 the block that dictates at which URL path all the included operations are available at, 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 suc-

ceeds 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 4.3. Whereas the `GET` can directly call the adapter implementation of the operation, the `POST` method 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 similar to those shown in this example.
All calls against the `adapter` transparently resolve the adapter that matches the targeted endpoint or use an already cached instance. More information about this process is explained in Section 5.5.1.

Listing 13: PaaSal API application routes definition excerpt

```
1  module Paasal
2    module API
3      module V1
4        class Applications < Grape::API
5          helpers SharedParamsHelper
6
7          resource 'endpoints/:endpoint_id/applications', desc: 'Endpoint\'s applications', swagger:
               { name: 'applications', nested: false } do
8            desc 'Get an applications that is registered at the endpoint' do
9              success Models::Application
10             failure [[200, 'Application retrieved', Models::Application]].concat ErrorResponses.
                   standard_responses
11           end
12           params do
13             use :application_context
14           end
15           get ':application_id' do
16             present adapter.application(params[:application_id]), with: Models::Application
17           end
18
19           desc 'Create an applications to be registered at the endpoint' do
20             success Models::Application
21             failure [[201, 'Application created', Models::Application]].concat ErrorResponses.
                   standard_responses
22           end
23           params do
24             use :endpoint_id
25             # require the keys of the application in the json object 'application'
26             requires :application, type: Hash do
27               # name is required, but we must use :all to get the presence validator. Name is
                     selected via :using
28               requires :all, using: Paasal::API::Models::Application.documentation.slice(:name)
29               requires :runtimes, Paasal::API::Models::Application.documentation[:runtimes].merge
                     (type: Array[String])
30               # everything else is optional
31               optional :all, using: Paasal::API::Models::Application.documentation.slice(:region,
                     :autoscaled)
32             end
33           end
34           post '/' do
35             application_params = declared(params, include_missing: false)[:application]
36             application_params = application_params.merge(params[:application][:vendor_specific
                   ]) if params[:application].key?(:vendor_specific)
37             present adapter.create_application(application_params), with: Models::Application
38           end
39         end
40       end
41     end
42   end
43 end
```

5.5 Adapters

Concerning the realization of the application architecture as visualized in Figure 7, it was decided to benefits from class inheritance and introduce an extensible adapter hierarchy . The resulting hierarchy is shown as class diagram 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 interface or skeleton for one specific API version. The `Stub` has to implement all methods of the API version it represents and always return the error to indicate the method has not been implemented yet. Finally, vendor specific adapters inherit from the `Stub` of the API version they are intended to be utilized for. Adapters can introduce their own private helper methods, but they have to overwrite the `Stub's` public API methods in order for the operations to 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.

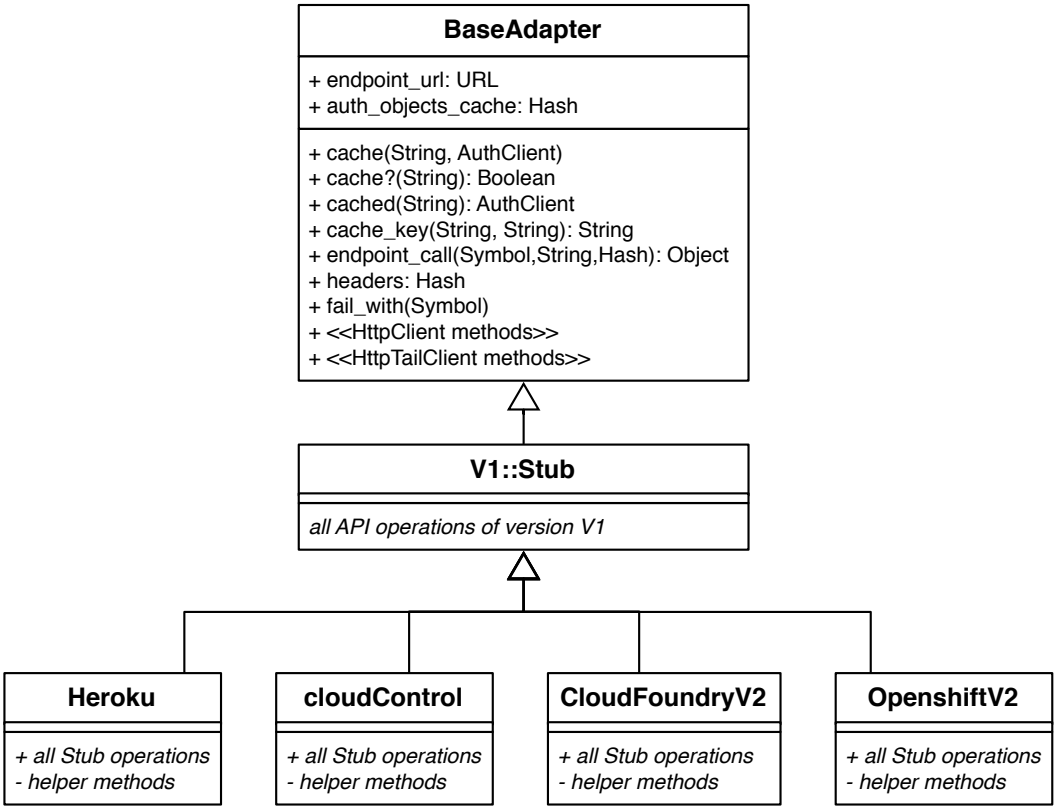

Figure 9: Adapter hierarchy class diagram

Additional classes which can be used by the adapters and are not made available by the `BaseAdapter` can be seen in Figure 10 with their method signatures. 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 all rely on these common resources.

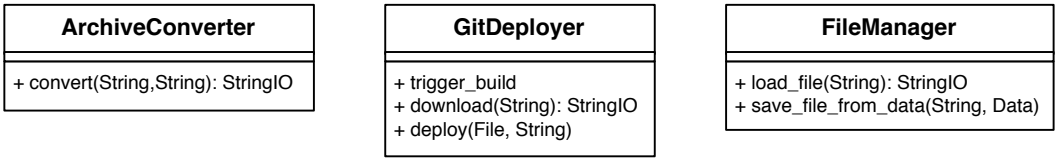

Figure 10: Archive, Git and File system helper classes to be used in adapters

5.5.1 Adapter Matching

With the previous section explaining the adapter structure, only the adapter matching that is shown in the gap between the adapters and the fundamentals of Figure 7 is left to be explained. Adapter matching describes how to resolve the proper adapter for the ongoing request, whereby the adapter to be used must always match the targeted endpoint and its parent vendor.
All needed actions to resolve the adapter can be found in Code Listing 14. The listing shows the authentication section of the PaaSal API, which is automatically invoked by all requests that targeting a protected URL path behind an endpoint. If the request contains some credentials, the first step is to load the targeted endpoint object. Using the adapter index object that was introduced in Section 4 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 PaaSal does also support re-authentication in case of expired API access tokens. Therefore, the adapter is patched with use of 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 to be executed in 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 PaaSal API. If 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: 'PaaSal API Authorization @ %{endpoint_id}',
3             realm_replace: [:endpoint_id]) do |username, password, params, env|
4    if username.nil? || username.empty? || password.nil? || password.empty?
5      # never allow empty username and / or password
6      false
7    else
8      # find a matching endpoint
9      endpoint = load_endpoint(params)
10     # resolve the required adapter
11     index_entry = adapter_dao.get params[:endpoint_id]
12     # use the already secured (https) URL of the index_entry
13     adapter = index_entry.adapter_clazz.new(index_entry.url, endpoint.app_domain, !endpoint.
           trust)
14
15     # patch the adapter so that calls are wrapped and expect valid authentication
16     AdapterAuthenticationInductor.patch(adapter, env)
17
```

```
18      # save info for the current request, no need to retrieve multiple times
19      request_cache.set("#{env['HTTP_X_REQUEST_ID']}.adapter", adapter)
20      request_cache.set("#{env['HTTP_X_REQUEST_ID']}.endpoint", endpoint)
21
22      cache_key = adapter.cache_key(username, password)
23      # THREAD HACK to work with deferred tasks (log tailing), cache auth key
24      request_cache.set("#{env['HTTP_X_REQUEST_ID']}.auth.cache.key", cache_key)
25
26      unless adapter.cache?(cache_key)
27        # no auth object available, perform authentication first
28        auth_object = adapter.auth_client
29        # throws an error if the authentication failed
30        auth_object.authenticate(username, password)
31        # cache the auth object so it does not have to be retrieved per request
32        adapter.cache(cache_key, auth_object)
33      end
34      # auth passed
35      true
36    end
37  end
```

5.5.2 Adapter Compatibility

In the first place, possible API operations to be realized with our abstraction layer were introduced in the approach of Chapter 3. Later on, the previous design chapter explained the object and operation mappings that must be obeyed in detail. Nevertheless, Table 3 of the approach also highlighted incompatibles which cannot be fixed with 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.

Heroku and Cloud Foundry both support all 37 defined operations of the adapter stub, while Openshift currently supports only 31 of the operations. However, the four logging operations could be added to this count as they can be realized but have not been implemented in this prototype. In summary, only the application update and service change are not supported, which both are not offered by Openshift at all. On Openshift, the service change is not being needed as long as plans are not actively used. 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 shall rather 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.

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	31
Supported degree	91.9 %	100 %	100 %	83.8 %

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

5.5.3 Adapter Implementation

PaaS vendors, for instance Openshift, are all represented by a unique adapter to orchestrate the communication between PaaSal 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 PaaSal project. The configuration file must be named exactly as 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 Code Listing 15.

The configuration file describes all objects to be known by the API, starting with the vendor on the first level of the YAML file. For the 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 4.3.1.

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"
```

All IDs of the objects described in the adapter configuration files are those under which the object will be available for use in the API. 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 during the API's 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 thereby 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 Paasal
  module Adapters
    module {API_VERSION}
      class {VENDOR_ID} < Stub
        # implemented methods
      end
    end
  end
end
```

In order 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.

Implementing 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 4.3.
Regarding the authentication method, PaaSal already offers four different authentication approaches which all could be used in new adapters, too. The authentication clients can be found at `lib/paasal/core/adapter_extensions/auth`.
In case 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, for instance, the domain management on Cloud Foundry or the lifecycle handling on Heroku. If the operation should not 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 test can be found in the upcoming Section 5.9.

5.6 Git Deployment and Repository Authentication

Git is used within PaaSal to maintain the 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 4.4.2, it was described which Git actions those operations need to be invoked. The Ruby gem `Git`[71] facilitates the execution of Git commands via Ruby code. All Git actions are bundled in the `Paasal::Adapters::GitDeployer` and `Paasal::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 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 = "paasal.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
```

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

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

Listing 18: Trigger rebuild method of the GitDeployer

```
def trigger_build
  push_repository_changes do |repo_dir|
    build_trigger_file = File.join(repo_dir, 'paasal-rebuild-trigger')
    current_md5 = File.exist?(build_trigger_file) ? Digest::MD5.file(build_trigger_file).hexdigest
        : nil
    data = StringIO.new("PaaSal rebuild, triggered at #{Time.now}")
    FileManager.save_file_from_data(build_trigger_file, data, false, current_md5)
  end
  nil
end
```

Despite the simplicity of the previously shown steps, the largest challenge within 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 being available on the system. Nevertheless, assuming some certificates are available on the platform, especially ones without a passphrase protection, using them would render the whole process to be quite fragile. PaaSal 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 to be used shall either be provided by the user via the application's configuration, or shall be generated during the application's startup phase. Those steps are performed within the `Paasal::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, 'paasal', '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
```

5.7 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 4.2.3. Errors are processed within the `rescue_from :all` block of the `RootAPI`. This error handling logic is presented in Listing 5.7 and shows that all errors are also saved with the 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 to be returned and a prioritized error logging statement shall be made. Those errors of Grape, as well as errors which are expected to be raised by the adapters of PaaSal, are converted to API conform response message. Three types of exceptions have to be distinguished within PaaSal, which are all introduced in the remainder of this section.

```
1  rescue_from :all do |e|
2    env['api.endpoint'].log.debug e.to_s
3    e.backtrace.each { |line| env['api.endpoint'].log.debug line }
4
5    if e.is_a?(Errors::ApiError) || e.is_a?(Paasal::Errors::AdapterError)
6      entity = env['api.endpoint'].build_error_entity(e.ui_error, e.message)
7    elsif e.is_a?(Grape::Exceptions::ValidationErrors) || e.is_a?(Grape::Exceptions::
         InvalidMessageBody)
8      entity = env['api.endpoint'].build_error_entity(ErrorMessages::
           BAD_REQUEST_VALIDATION, e.message)
9    elsif e.is_a?(Grape::Exceptions::InvalidAcceptHeader)
10     entity = env['api.endpoint'].build_error_entity(ErrorMessages::
           INVALID_ACCEPT_HEADER, e.message, e.headers)
11     env['paasal.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

Internal errors appear during the initialization of the application and usually prevent the 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 prevent the start of the PaaSal API. For instance caused by bad configuration if the specified SSH certificates could not be found.

API Errors

An `ApiError` is caused by the user's request, for instance if a parameter's content is invalid, e.g. if it points to invalid endpoint resources. They are raised by PaaSal only. 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

The `AdapterError` is caused by errors being raised within PaaSal 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 with 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 PaaSal 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 PaaSal and the endpoints.

UnknownAdapterCallError 500, is raised by PaaSal only. Hints at implementation errors, for instance when an adapter is outdated.

AdapterMissingImplementationError 501, raised by PaaSal 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.

5.8 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 for comparison in the activity diagram of Figure 11.

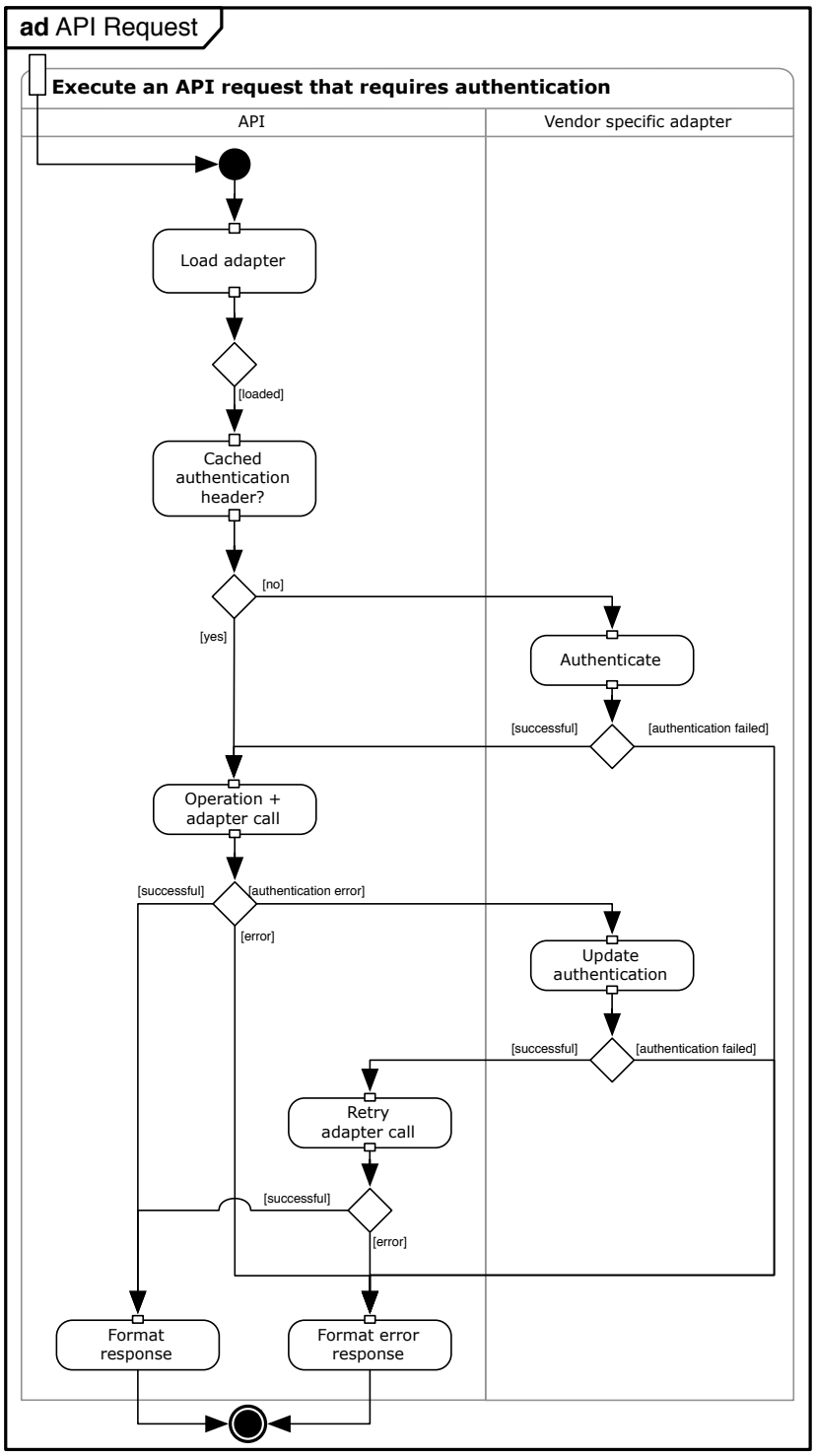

Figure 11: Activity diagram showing the involved steps of authenticated API requests

As previously explained in Section 5.5.1, the first step of the processing is to load the adapter being 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 5.5.1, especially in Code 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. On the occasion the authentication succeeds, the process is resumed as if the credentials were already available before.

If it shall be the case 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, PaaSal attempts to acquire a new token. Should this authentication update be successful, the adapter call is retried, which then is formatted according to whether the call fails or succeeds. Otherwise, the error response is formatted to indicate the expired credentials.

5.9 Automated Tests

In statically typed programing languages, for instance C++ or Java, the compiler will already find a 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 when developing applications in Ruby.
PaaSal features three distinct types of tests. In addition to the commonly used unit and integration tests, PaaSal's adapter tests can be regarded as complete system test. Adapter tests call the API of PaaSal, which then calls the 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 being placed directly in the `spec` folder are required by all tests, as for instance 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, for instance if classes are too long or complex.
`VCR`[72] is the most important gem in regards to the adapter tests. It allows to record HTTP interactions with foreign systems which can also be replayed and thus enable dramatically faster, deterministic and accurate tests. Within the test of PaaSal, 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.

[72] *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 explained

5.9.1 Adapter Tests

Adapter tests are complete system tests that include communication with third party systems, in this 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.

Code Listing 27 in Appendix B presents the basic template of a file used to describe all tests that are to 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 the platform to be tested, those tests can be marked as unsupported via the variable in the leading `before :all` block. Tests can be written for multiple endpoints, as well as for multiple API versions of the abstraction layer, which both can also be specified via the variables. 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, the `VCR` cassette is inserted. 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 being persisted so that it is available for test replays. Any error or test violation prevents the recorded cassette from getting persisted.

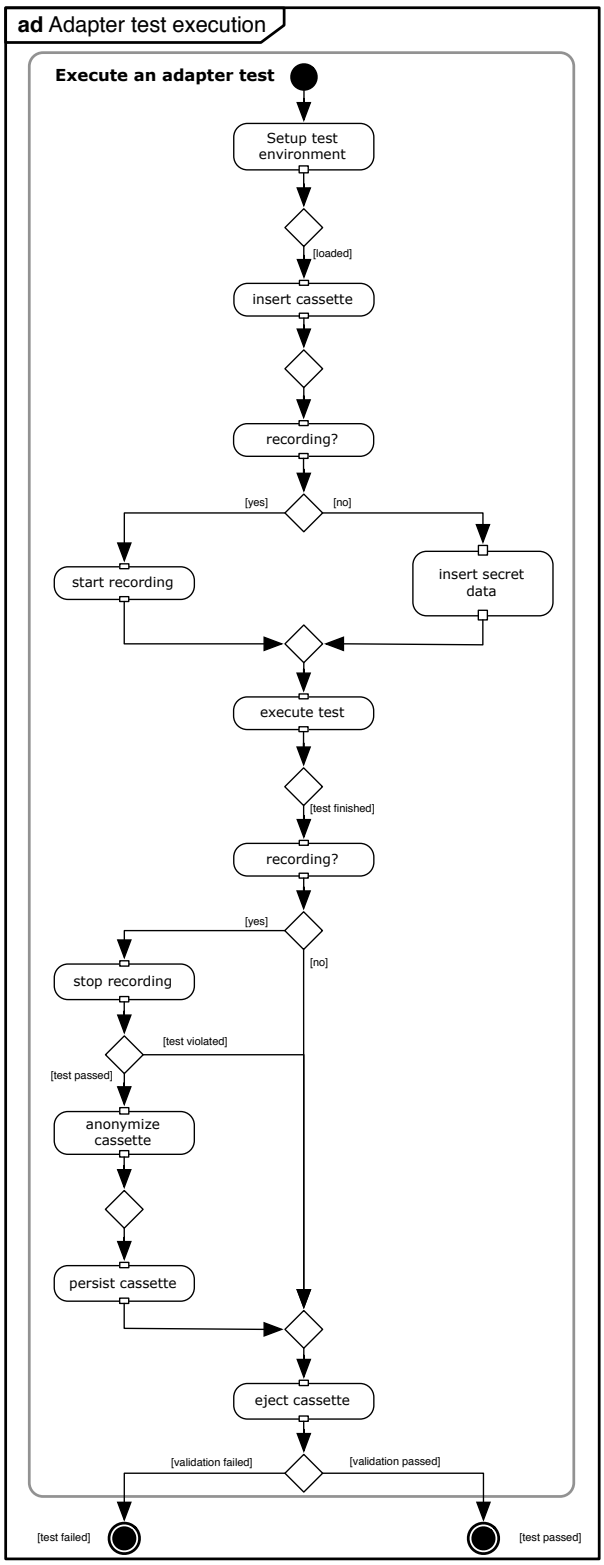

Figure 12: Sequence diagram showing the test execution process

In order to validate whether the prototype of abstraction layer is working properly and can be used for the management and deployment operations, the need to have a demo application which can be deployed on all supported platforms during the adapter tests without requiring further technical adaptions was identified.
After a brief analysis of the supported vendors and their offered runtime languages, it was decided to use a language different from PaaSal's implementation language and one

agreed on Node.js. Node.js is not only offered by all of the chosen vendors, but most PaaS vendors in general.
The application itself is the so called Word Finder[73]. Word Finder is a minimalistic open-source application and 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[74] and the source code of the Word Finder application immediately revealed the problem, which is highlighted in Code Listing 20.

Listing 20: 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 21.

Listing 21: 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 flawlessly on all four platforms during the automated adapter tests.

5.10 Documentation

PaaSal's 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 written code is based on the `YARD`[75] gem. At the time of submitting the PaaSal project, nearly 70% of the modules, classes, constants and methods were properly documented. The final version of the generated documentation files is included in the provided project. It can also be generated via the `bundle exec rake doc` command inside the project's directory. With `inch`, another gem can be used to verify the quality of the code documentation.

[73] *Word Finder - Node.js application.* URL: `https://github.com/amirrajan/word-finder/` (Retrieved: April 29, 2015).
[74] *Openshift - Node.js Overview.* URL: `https://developers.openshift.com/en/node-js-environment-variables.html` (Retrieved: April 30, 2015).
[75] *YARD: A Ruby documentation generation tool.* URL: `http://yardoc.org/` (Retrieved: July 9, 2015).

A vital part of fostering API adoption is to provide a useful and easy to understand documentation. While developing the adapter for cloudControl, it was experienced at first-hand how painful missing, incomplete or even invalid API documentations can be.
With the help of the `grape-swagger`[76] gem, an extension to the Grape API, additional API resources that describe the complete PaaSal API in version 1.2 of the standardized swagger[77] specification are made available. The specification is auto-generated with the start of the application itself and therefore only available at runtime. Swagger allows to create an interactive documentation, generate API clients for many programing languages and also enables the discoverability of the API.
The interactive documentation is a HTML and JS application known as Swagger UI[78], 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 of the deployed web application. Screenshots that hint at the capabilities of the chosen solution can be found in Appendix C. Figure 17 shows the available operation groups. All operation groups can be expanded to show the operation entries similar to Figure 18. The operations can then also be expanded, showing their full specification as presented in Figure 20 for a `GET` request and Figure 19 for a `PATCH` request. Request parameters, response objects and all allowed response status codes are explained in detail within the specification. If using the interactive possibilities of the UI, parameters that are needed to execute a request can be provided in the available input fields.

5.11 Issues

Even though the project could be created as originally intended, many obstacles had to be crossed during the implementation phase. In retrospect, some technology choices were quite unlucky but still fitting for the most part. In total, eleven changes and fixes had to be made to third party projects, most of them within the `grape` gem. Pull requests that were made to `grape` are known under the identifiers #906, #912, #913, #942 on the GitHub repository. Two fixes were made to the `grape-entity` project, namely #110 and #111. One change with the number #220 to the `grape-swagger` gem and one to the `airborne` gem known by identifier #36.
Fixes were also committed to `rack-stream` in the pull request #6 and `rack-test` in #125. However, both project maintainers did not respond to the changes, so that they had to be included as monkey patch in the PaaSal project.

Setting up the adapter tests also involved an almost uncountable number of challenges and problems. The recording of the platform interactions required a tedious anonymization of the cassettes, as otherwise personal usernames, email addresses and passwords would have been exposed. Besides all recorded HTTP interactions, it became obvious also the file system interactions, web-socket and streamed HTTP requests needed to be recorded. These recorders had to be developed from scratch as no gem could be found to provide suitable features. Once the tests were working as supposed, the overall test duration was still unbearable. A replacement of the default JSON converter with the `oj` gem, minor

[76] *grape-swagger: Swagger compliant documentation of your grape API.* URL: `https://github.com/tim-vandecasteele/grape-swagger` (Retrieved: June 25, 2015).
[77] *Swagger API representation.* URL: `http://swagger.io` (Retrieved: June 28, 2015).
[78] *Swagger UI.* URL: `http://swagger.io/swagger-ui/` (Retrieved: June 28, 2015).

changes within the tests as well as a change to only filter out private data when recording, together significantly reduced the overall test duration. On Travis CI[79], one complete test execution now takes about four minutes, opposed to approximately two minutes for a local test run.

Furthermore, there were also a variety of platform internal issues which hindered the tests, especially a reliable recording. For instance, Openshift quite often fails for long-running requests with internal timeout errors or sudden maintenance warnings.
Regarding cloudControl, names of an application must not only be globally unique, but also stay locked for a couple of days after the application has been deleted. In consequence, the adapter test file and the chosen application name must be modified if the test has to be re-recorded within this locking period. From time to time, it was also the case cloudControl rejected SSH or Git interactions, complaining about unverified server signatures. The operations always succeeded within the first repetition, so that the issue could not be reproduced. It must also be noted that it is currently not possible to run the logging tests with cloudControl, even though the functionality is basically working. This is due to an inconsistent delay of up to five minutes before messages appear in the log files. All test assertions fail after one minute at the latest and a change of the timeouts to wait even longer would only introduce different issues.

One issue which could not be resolved yet is the required domain verification on cloudControl. Each added domain must be manually verified by adding a TXT record to the DNS entries of the domain's root domain.

One of the most important features to attract developers, the unified access to all logging systems, proved to be not as simple as originally expected. Whereas Cloud Foundry only required to setup a WebSocket communication, the API operation that should allow the log tailing was the major challenge. Opposed to the advertised support to build streaming APIs, `grape` must be used with the `rack-stream` gem to enable this feature. Not only does the project enforce a lock-in into EventMachine based rack servers, as for instance Thin, but it also proved to be quite unreliable. Not before a custom modification was written, `rack-stream` did not notice lost client connections. Without this fix, the tailing processes would always remain active and waste resources before it would finally run into resource shortages.

5.12 Usage

PaaSal can be installed and used without any major system requirements. The following sections highlight the most important aspects which have to be regarded. Section 5.12.1 presents what must be done before installing PaaSal, whereas the available configuration options are explained in Section 5.12.2. The usage of the PaaSal gem is introduced in Section 5.12.3. PaaSal's language independent RESTful API is then presented in the concluding Section 5.12.4.

[79] *Travis CI*. URL: `https://travis-ci.org` (Retrieved: July 19, 2015).

5.12.1 System Requirements

Most important, PaaSal requires a Ruby version greater than 1.9.3 to be run on any operating system. It is developed and actively tested with the default MRI Ruby interpreter. Furthermore, PaaSal 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 if using one of Microsoft's Windows versions.
On Windows, both executable files should be located in the `Git/bin` installation directory of msysGit. PaaSal is verified to work with msysGit and its included version of OpenSSH. Any other alternatives, e.g. PuTTY, were not verified whether they are also compatible.

More operating system specific issues and all known fixes are listed in the project's README file.

5.12.2 Configuration

PaaSal allows a number of options to be changed within Ruby configuration files. Moreover, there are three possibilities how a configuration file can be made available to PaaSal. 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 PaaSal gem or the API. If available, the file is loaded from `paasal/paasal_config.rb` on Windows systems, and `.paasal/paasal_config.rb` on all UNIX systems, relative to the home directory.
Overwriting the system wide configuration, the PaaSal gem can be provided with its own configuration file. The `paasal_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 `paasal_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 22. 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/paasal` directory as expected by most Linux systems.
The next five options focus on the persistence of the public API entities. In theory, custom backends that are supported by moneta can be installed and used. On UNIX, the standard database backend is Daybreak, whereas on Windows LMDB is used as Daybreak does not run on Windows systems. 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 5.6, a private SSH key is needed to enable the application data deployment via Git. If wished, the location of a personal SSH key to be used can be placed inside the `paasal_config.ssh.custom_key` option. It is required to confirm the provided

key was created with OpenSSH, uses ssh-rsa and is not protected with a passphrase. By default, a new key will be created upon starting PaaSal.
All remaining options describe the API and are publicly accessible in the public part of the API. They should be filled in so that users know whom to contact and what to expect.

Listing 22: PaaSal's default configuration file

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

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

# [optional] Database backend to use. Choose one of: [:Daybreak, :LMDB]
# Defaults to: :Daybreak on UNIX, :LMDB on windows systems.
# paasal_config.db.backend = :Daybreak

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

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

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

# [optional, requires 'paasal_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
# paasal_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, PaaSal will use its own private key (config/paasal_git_key.pem) to authenticate all Git actions.
# Defaults to: nil
# paasal_config.ssh.custom_key = nil

# [optional] Specify the public API description
# paasal_config.api.title = 'PaaSal - Platform as a Service abstraction layer API'
# paasal_config.api.description = 'PaaSal allows to manage multiple PaaS providers with just one API to be used'
# paasal_config.api.contact = 'paasal@roecky.net'
# # The name of the license.
# paasal_config.api.license = 'TBD'
# # The URL of the license.
# paasal_config.api.license_url = ''
# # The URL of the API terms and conditions.
# paasal_config.api.terms_of_service_url = 'API still under development, no guarantees (!)'
```

5.12.3 Ruby Gem

PaaSal can be embedded in another application as Ruby gem. To do so, Code Listing 23 shows the essential steps. In the beginning, the dependency on the PaaSal 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 PaaSal in the application, which then automatically triggers PaaSal's internal initialization procedure.

Listing 23: Include the PaaSal gem in another Ruby application

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

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

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

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

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

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 5.12.2, are accessible via the `paasal_config` accessor.
Via the `Paasal::VersionDetector.api_versions` method call it is possible to obtain a list of all available versions of the abstraction layer. After a version has been chosen, Code Listing 24 demonstrates how the gem can be used to invoke further actions. The `Paasal::AdapterResolver` must always be instantiated with a valid version. As soon as the `AdapterResolver` is instantiated, 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 requests with. 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 to be created by PaaSal would otherwise be malformed.
The methods which can then 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 Code Listing 24 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 `Paasal::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 24: Use the PaaSal gem in another Ruby application

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

5.12.4 Server

A rack server to run PaaSal can be started in multiple ways. The most convenient and preferred solution is to use the provided startup script, which is located at `bin/paasal` of the project. The script is provided in two versions, one running on UNIX systems and a .bat file for Windows. It comes with a variety of options, which can be seen in Listing 25 showing the help output of the file. 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 25: PaaSal's startup script

```
bin/paasal --help

Usage:
paasal [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, PaaSal will automatically redirect all incoming plain HTTP connections to HTTPS connections if it is running in the production environment.
However, the API can also be started by using the typical rackup command as shown in Listing 26.

Listing 26: Rackup of the PaaSal API

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

By reason of technical dependencies on the event based rack servers, PaaSal can currently run only on the Thin[80] web server. This dependency was introduced by the log tailing operations and the therefore required `rack-stream` gem.

When running PaaSal as a web-service without a permanent database, all changes made to the vendor, provider and endpoint objects at runtime 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 only the location where to store the data to be specified.

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

6 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 literature.
With the help of Travis CI and the developed test suites, PaaSal 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 2.1.3 and MRI 2.2.1. 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 on 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, for instance the application data management and the lifecycle handling.

Further aspects of cloud applications were also taken into account with the 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 a use case to change the current vendor is named, as for instance by Dana Petcu [Pet11] and Loutas et. al. [LKT11]. 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 PaaSal and most of the task's needed work to be achieved is already done. It would also be required that the application container is compatible to the technical requirements of the chosen platform, which is however not of further concern for this work. The user must also have an account which is legitimated to create and deploy applications on both platforms. Nevertheless, PaaSal does not yet integrate support to manage data being kept within database and thus neither can the data be moved to the new platform nor can the use case be passed.
According to Loutas et. al, additional use cases referring to the management interfaces of PaaS platforms are the application deployment on a PaaS offering and PaaS systems interoperation [LKT11]. 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, PaaSal passes this use case as it harmonizes different operation namings with the same functionality over all supported platforms. Hybrid clouds and the requested common PaaS offering model are not offered by PaaSal. Nevertheless, the provided information, for instance 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, being either a developer or a company, 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. PaaSal can fulfill the first three, namely 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 PaaSal are the configuration of application health monitoring and the notification about a service condition or event. Depending on the proper definition of both use cases, they can be questioned to be realizable with any of the chosen platforms due to missing features and operations.

At the beginning of this thesis, it was claimed 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 for 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.
PaaSal also enhances the interoperability of PaaS platforms and applications. The common interface and 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 which would be required for this interaction are the scaling and lifecycle operations. PaaSal's common object model thereby allows applications on different platforms to be capable of semantically processing the information.

Concerning the aspect of the demanded programing 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[81]. 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.
PaaSal's extendability is provided by the modular structure with the dedicated adapters for each supported platform and the possibility to host multiple versions in parallel.

Performance was never a crucial part of the abstraction layer's design or implementation. In the adapter tests, it was noted that the API's performance mostly depends on the targeted platform, its virtualization technology and the fact whether operations are executed synchronously or asynchronously. Another important aspect to compare the performance is shown in table 34 of Appendix D. The table points out how many HTTP requests had to be made in order to achieve the same result on all platforms. All table rows with a red background cannot be used for comparison as the requests in tests are repeated until a timeout is violated or the assertion is matched. An overall comparison is also not very meaningful due to the different operations that are unsupported on the platforms. However, even though a sound comparison is questionable, cloudControl seems to require approximately one HTTP requests less to achieve the abstraction.
Future evaluations regarding the performance of the abstraction layer or even the platforms themselves could be based on the tests' recording durations. Currently, the duration varies between a few minutes for a local Cloud Foundry installation and more than half an hour when recording the tests on Openshift Online.

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

7 Related Work

Abstraction layers are a well-known and accepted solution by means of harmonizing a variety of different systems, which all share a common principle, behind one interface. In our prototype, the moneta[82] gem is used to interact with multiple key/value stores, for instance Daybreak[83] and LMDB[84] that are used with PaaSal on UNIX, respectively Windows systems.
Looking for references in the area of cloud computing, which introduced or followed an approach that is related or similar to the idea behind PaaSal, we found a variety of projects.

Apache Deltacloud[85] is a successful attempt to improve the interoperability amongst various IaaS providers. It provides a generic and language independent RESTful API that allows to talk to a multitude of providers without dealing with the provider's API itself. If a provider changes its API or a new provider enters the market, only the Deltacloud implementation must be adapted. Therefore, all applications utilizing the Deltacloud API can remain unchanged, which is one of the main advantages of this abstraction layer.
Besides Deltacloud, there are also several language specific abstraction layers on the IaaS level, for instance jclouds[86] (Java), libcloud[87] (Python) and fog[88] (Ruby).

Strongly related to their OW2 FraSCAti project[89], Paraiso et al. describe a multi-cloud PaaS system. They deployed their custom framework onto 13 different PaaS platforms. It adds runtime support for the service component architecture and allows SaaS applications to be installed on top. This procedure was demonstrated with three applications to validate the claimed improvements in portability, interoperability, heterogeneity and geo-diversity. Dedicated infrastructure services of theirs are used to manage the federation and take care of node provisioning and application deployment.
However, no detailed information is available how they managed the deployment onto all 13 diverse PaaS providers. The project appears to be discontinued. [PHMRS12]

The current draft of the Cloud Application Management for Platforms standard proposal describes an API and the required artifacts that a PaaS cloud shall ideally offer. CAMP was initially developed as a collaboration between CloudBees, Cloudsoft Corporation, Huawei, Oracle, Rackspace, Red Hat and the Software AG. It is currently awaiting the specification as an official OASIS standard, which requires the evidence of interoperable implementations. CAMP is designed to be language, framework and platform neutral with the goal of enabling a set of common frameworks, libraries, plugins and tools for the PaaS ecosystem. The operations that CAMP specifies include building, life-cycle management, administration and monitoring tasks. In the area of PaaS standards and

[82] *Moneta: A unified interface for key/value stores.* URL: `https://github.com/minad/moneta` (Retrieved: June 28, 2015).
[83] *Daybreak: A simple and very fast key value store for ruby.* URL: `http://propublica.github.io/daybreak/` (Retrieved: June 28, 2015).
[84] *Symas Lightning Memory-Mapped Database (LMDB).* URL: `http://symas.com/mdb/` (Retrieved: June 28, 2015).
[85] *Apache Deltacloud.* URL: `http://deltacloud.apache.org` (Retrieved: October 28, 2014).
[86] *Apache jclouds.* URL: `https://jclouds.apache.org` (Retrieved: November 9, 2014).
[87] *libcloud.* URL: `https://libcloud.apache.org` (Retrieved: November 9, 2014).
[88] *fog.* URL: `http://fog.io` (Retrieved: November 9, 2014).
[89] *FraSCAti.* URL: `http://frascati.ow2.org` (Retrieved: November 9, 2014).

especially standards for PaaS management, CAMP is by far the most advanced concept, but has not found wide adoption yet. [OAS14]
A first minimal proof of concept implementation of the standard called nCAMP[90] was expected to be released before the end of 2014[91], but was not published as of July 2015.

SeaClouds[92], an EU research project having started in 2013, is funded with nearly three million Euros and expected to run until early 2016. Once finished, the project is expected to be open-sourced and shall be "providing seamless adaptive multi-cloud management of complex applications by supporting the distribution, monitoring and migration of application modules over multiple heterogeneous PaaS platforms" [BIS+14, p. 1].

The mOASIC[93] project aims at personalizing the IaaS layer to establish and fulfill the user's requirements on the PaaS layer. Its main objective is to provide a solution for multi-cloud applications. In order to communicate with multiple PaaS platforms, mOASIC uses the concepts of drivers, connectors and cloudlets. However, mOASIC only supports applications being created from scratch and is not applicable to already existing applications. [Pet11]

In 2013, Sellami et al. introduced their Compatible One Application and Platform Service (COAPS) API, which was successfully integrated in *The Compatible One* project as proof of concept realization. The COAPS API is a provider independent approach which allows the application management and provisioning in the context of PaaS. Reference implementations of the API are provided for Pivotal's Cloud Foundry and RedHat's Openshift. Figure 13 visualizes all operations that the COAPS API supports. [SYMT13; Tel13]

Application management operations	
Operation	**Command**
Create Application	POST /app
Update Application	POST /app/{appId}/update
Find Applications	GET /app
Start Application	POST /app/{appId}/start
Stop Application	POST /app/{appId}/stop
Restart Application	POST /app/{appId}/restart
Describe Application	GET /app/{appId}
Destroy Application	DELETE /app/{appId}
Destroy Applications	DELETE /app/delete
Deploy Application	POST /app/{appId}/action/deploy/env/{envId}
Undeploy Application	POST /app/{appId}/action/undeploy/env/{envId}
Environment management operations	
Operation	**Command**
Create Environment	POST /environment
Update Environment	POST /environment/{envId}/update
Destroy Environment	DELETE /environment/{envId}
Find Environments	GET /environment
Describe Environment	GET /environment/{envId}
Get Deployed Applications	GET /environment/{envId}/app
Get information	GET /environment/info

Figure 13: Operations of the COAPS API defined by Sellami et al. [SYMT13]

[90] *nCAMP - The CAMP Management Service, a minimal implementation of the CAMP 1.1 specification.* URL: `http://ec2-107-20-16-71.compute-1.amazonaws.com/campSrv/` (Retrieved: May 5, 2015).
[91] *CAMP - Meeting Minutes 15th October 2014.* URL: `https://www.oasis-open.org/committees/download.php/54389/Minutes%2015th%20October%20%202014.txt` (Retrieved: May 5, 2015).
[92] *SeaClouds - Seamless adaptive multi-cloud management of service applications.* URL: `https://github.com/cloudpier` (Retrieved: July 11, 2015).
[93] *mOSAIC.* URL: `http://www.mosaic-cloud.eu/` (Retrieved: October 28, 2014).

Cloud Pier[94] is described by its developers as an open source multi-cloud application manager for PaaS. Cloud Pier can either be used as a (hosted) SaaS application or as local installation. It originated from the European Commission funded project Cloud4SOA[95] and is available publicly on GitHub[96]. Cloud4SOA provides four core capabilities, of which one is the management and deployment of applications to one of the supported PaaS providers. According to the original publications, adapters were offered for AWS Elastic Beanstalk, Cloud Foundry, Openshift and CloudBees. [KLZ+13; DBCAZ12]
Since CloudBees announced to terminate their PaaS offering and the AWS Elastic Beanstalk API is undergoing major changes, only two of the four adapters are likely to work today. However, more recent information[97] were published on their website and state that there are also adapters available for Heroku and cloudControl. The general capabilities of the Cloud4SOA adapters are shown in Figure 14. All operations with a brief description are presented in Figure 15.

Cloud4SOA Capabilities	AWS Beanstalk	CloudBees RUN@cloud	CloudControl	Heroku	CloudFoundry	OpenShift
Matchmaking:						
The criteria and information of the PaaS within Cloud4SOA's matchmaking search.	✓	✓	✓	✓	✓	✓
Governance						
Deploy	✓	✓	×	×	✓	×
Deploy through GIT	×	×	✓	✓	×	✓
Start	✓	✓	✓	✓	✓	✓
Stop	✓	✓	✓	✓	✓	✓
Delete	✓	✓	✓	✓	✓	✓
Create Database	✓	✓	✓	✓	✓	✓
Delete Database	✓	✓	✓	✓	✓	✓
Import Data to Database*	✓	✓	✓	✓	✓	✓
Export Data from Database*	✓	✓	✓	✓	✓	✓
Monitoring						
get Status	✓	✓	✓	✓	✓	✓
get HttpResponse Time	✓	✓	✓	✓	✓	✓
get ICMP Ping Response Time	✓	✓	✓	✓	✓	✓
Migration						
Migrate**	⇆(1)	⇆(1)	⇆(2)	⇆(2)	⇆(1)	×
*	Importing and Exporting Data to a database is done by using the database configuration parameters, and is supported to a limited set of database systems					
**	Migration of application and its data between different platforms is supported, but it is only possible between specific platforms that are mentioned with the same number in the table					

Figure 14: Cloud4SOA adapter features[97] as of March 2014

[94] *Cloud Pier*. URL: `http://www.opencloudpier.org` (Retrieved: December 2, 2014).
[95] *Cloud4SOA repository*. URL: `https://github.com/Cloud4SOA/Cloud4SOA` (Retrieved: October 28, 2014).
[96] *Cloud Pier repository*. URL: `https://github.com/cloudpier` (Retrieved: December 2, 2014).
[97] *Concertation Meeting - E2 Software & Services, Cloud Computing*. URL: `http://www.cloudwatchhub.eu/sites/default/files/Cloud4SOA_DAndria_Concertation-Meeting_12-13Mar2014.pdf` (Retrieved: May 9, 2015).

	API method	Description
Development Life-Cycle Supportive Methods	**create Application**	It creates an empty application. If the specified application cannot be created, then a NULL object is returned. Otherwise, a potentially modified Application object is returned, as dictated by the underlying platform.
	delete Application	It deletes an application. If the application cannot be deleted or does not exist, then return false. Otherwise, true.
	update Application	It updates application elements. If the specified application cannot be updated, then a NULL object is returned. Otherwise, a potentially modified Application object is returned, as dictated by the underlying platform.
	createApp Version	It creates an application version which refers to the given application. If no application with the specified ID can be found or the application version cannot be created, then a NULL object is returned.
	deleteApp Version	It deletes Versioned Application of the application with the given ID and version string. If no such object refers to the aforementioned pair of data then a NULL object is returned.
	deploy	It deploys the specified application version. If no application version can be found, or no environment is associated to it, then a NULL object is returned.
	un-deploy	It UN-DEPLOY the specified application version. If no application version can be found, or it is not already deployed, then a NULL object is returned.
	deployToEnv	It deploys the specified application version into the specified environment. If neither object can be located, then return a NULL object. Otherwise, return the corresponding Application Space.
	uploadAnd DeployToEnv	It creates a new application (if needed), then a new application version, upload the respective executable file, associate and deploy it to the specified environment.
Monitoring Methods	**getAppStatus**	Get status of the specified application version. If no application version exists for the given pair of application ID and version, then a NULL object is returned. Otherwise, the corresponding status object is returned.
	getApp Statistics	Get statistics of the specified application versions. A NULL object is returned if no application version is found or if no statistics can be retrieved from the underlying infrastructure, then a NULL object is returned.
	getRunning Status	Get the status of the running instance that corresponds to the deployed application version. Instance Status is an enumeration with four indicative values, namely {UNRESPONSIVE, RUNNING, STOPPING, STOPPED}.
	getSummary Statistics	Get summary statistics of all running application versions of the given application in the given environment. If the specified objects cannot be found or no statistics can be retrieved by the underlying infrastructure, then a NULL object is returned, otherwise, a Statistics instance.
Instance Handling Methods	**getApp Versions**	Returns a list of application versions found in the PaaS for a specific application.
	getApp Environments	Get all environment information about all versions of the application with the given ID. If the specified application does not exist, then a NULL object is returned. Otherwise, return an array of Environment objects.
	getRunning AppVersion	Get all currently running application versions of the specified application in the specified environment. If the given application or environment cannot be found, or no application version refers to the combination of these, then a NULL object is returned.
	start	Start a non-started instance of the specified application version. If no application version can be found, or it is cannot be started, then a NULL object is returned.
	stop	Stop an already started instance of the specified application version. If no application version can be found, or it is cannot be stopped, then a NULL object is returned.
Resource Handling Methods	**check App Availability**	Checks whether the specified application name is available. If so, then true is returned; otherwise, false.
	listApplications	It is used in order to return a list of all applications registered for a specific user a specific PaaS provider. Information returned may vary between PaaS providers.
	create Environment	Create a working environment, given the appropriate information set. If such an environment cannot be created, then a NULL object is returned.
	delete Environment	Delete the environment that corresponds to the given ID. If no environment can be found, or the environment cannot be deleted, then false is returned. Otherwise, true is returned
	attach Environment	Associates the specified application version with the given environment. If no application version or no environment has the given IDs or the association fails, then false is returned. Otherwise, true is returned.
	update Environment	Update environment details. If the given ID does not correspond to an existing environment or no update can be accomplished, then a NULL object is returned.
	createDB	It creates an empty database. If the specified database cannot be created, then a NULL object is returned. Otherwise, a potentially modified database object is returned, as dictated by the underlying platform.
	getDBList	It is used in order to return a list of all databases registered for a specific user under a specific PaaS provider. Information returned may vary between PaaS providers.
	deleteDB	It deletes a database. If the database cannot be deleted or does not exist, then return false. Otherwise, true. This method can be fully implemented for all the indicative providers
Handling Methods	**exportApp Version**	It exports application version executable file plus any associated external data, which was defined in the Environment object. Possible data export types include database dumps, file system copies or raw data exports.
	downloadDB	Saves database data and schema to a specified target. If the database cannot be downloaded or does not exist, then return false. Otherwise, true
	restoreDB	It restores data and schema to a database from a specified source. If the database cannot be restored or does not exist, then return false. Otherwise, true.

Figure 15: Cloud4SOA Harmonized API [DBCAZ12]

Similar to the proposed approach of PaaSal, the PaaS Manager (or PaaSManager) defines a set of common operations with the goal of improving application portability on the PaaS level. It abstracts the differences of multiple providers and supports the lifecycle management as well as deployment. [CNS13; CNS14]
To our knowledge, the PaaS Manager application has neither been released, nor adopted by a significant number of PaaS users. It was included in the Cloud Service Broker framework [GCN+13], which apparently was not released either.
Figure 16 presents the operations of the PaaS Manager and the actions that are needed to succeed with the three platforms, CloudBees, Cloud Foundry and Heroku. The operations are grouped as management and information services.

Management Services	*Description*	*CloudBees*	*Cloud Foundry Iron Foundry*	*Heroku*
createApp	create an application in a specific PaaS	-	createApplication	createApp
deployApp	deploy the application source code	application DeployArchive	uploadApplication	Git
migrateApp	migrate an application to another PaaS	application DeployArchive	uploadApplication	Git
startApp	start an application	application Start	startApplication Application	set Maintenance Mode = 0
stopApp	stop an application	application Stop	stopApplication	set Maintenance Mode = 1
restartApp	restart an application	application Restart	restartApplication	restart
deleteApp	delete an application	application Delete	deleteApplication	destroy Application
scaleApp	manual scaling	application Scale	updateApplication Instances	scaleProcesses
updateApp	update the application source code	application DeployArchive	uploadApplication	Git
createService	bind a database to an application	databaseCreate	createService bindService	addAddon
deleteDatabase	remove a database	databaseDelete	deleteService unbindService	removeAddon
Information Services	*Description*	*CloudBees*	*Cloud Foundry Iron Foundry*	*Heroku*
getAppStatus	application health	applicationInfo getCrashes	getApplication	listProcesses
getAppStatistics	application real-time statistics	New Relic API	getApplicationStats	New Relic API
getAppInfo	application basic information	applicationInfo	getApplication getCrashes	listApps listProcesses
getAppListInfo	list of applications	applicationInfo	getApplication getCrashes	listApps listProcesses
getServiceInfo	database basic information	databaseInfo	getService	listAppsAddons
getServiceApp ListInfo	list of databases	databaseInfo	getService	listAppsAddons
getAppLogs	application logs	tailLog	-	getLogs
getPaaSOffering	PaaS supported technologies	-	-	-

Figure 16: PaaS Manager operations and required actions on the platforms [CNS14]

Beyond creating only an up-to-date solution and in contrast to some of the introduced projects, the approach of PaaSal allows to deploy any application regardless of its implementation language. Moreover, the API of the abstraction layer allows to be used within any programing language. The extensive and interactive API documentation enables the fast and simple access for developers to use the abstraction layer, which is not given by any of the introduced projects. PaaSal does not require any documents, e.g. application descriptors to be created, but can be used without adaption for already existing applications.
Compared to the PaaS Manager, the COAPS API and Cloud4SOA, PaaSal has a strong focus on the management and deployment interfaces. It supports not only very rudimentary features but also allows to use the provider's scaling and logging features. Even though the horizontal scaling capabilities are one of the biggest advantages of cloud applications compared to traditional software, they appear to not be directly supported in the COAPS API and Cloud4SOA. PaaSal introduces properly named and structured API objects, which can all be accessed and modified separately. To enhance the supported functionality of the prototype even further, it allows the execution of arbitrary commands against the endpoint's API while transparently handling all authentication mechanics.
In comparison to Cloud4SOA, PaaSal allows a much more fine grained control of the application and its associated objects. Another strength of PaaSal is the unified deployment, which is not supported by Cloud4SOA. Rather than attempting to create a complex matchmaking, management and monitoring solution, PaaSal focuses solely on the creation of an extensively harmonized management and deployment API.

8 Conclusion

The research question of this thesis was defined to be whether it is possible to abstract the differences of vendor specific management and deployment interfaces on the PaaS layer by creating an intermediary abstraction layer. Having finished all major stages of the project, the question can undoubtedly be affirmed.
Dozens of issues were experienced despite the extensive evaluation of the platform's APIs and technologies upfront to the start of the prototype's implementation. Hidden complexities, for instance the management of the application lifecycle on Openshift, or the access to Cloud Foundry's logging system, challenged the designed abstraction layer once in a while. Whereas the missing API documentation of cloudControl caused frustration during the design phase, the unstable API of Openshift quite frequently troubled the test system. The majority of the remaining issues was related solely to the prototype's implementation and especially the chosen technologies. Grape and its extensions form a perfect ecosystem to build sophisticated RESTful APIs, but did fall short on delivering up to their promises, especially concerning the level of maturity. One unresolved drawback of the API is the limitation to use only Thin as web server. Recording the adapter tests remains a cumbersome process that takes time and is error-prone due to the instability of the platforms' APIs. All in all, the challenges could be resolved properly and without sacrificing the goals or the quality of the abstraction layer.
With the generic PaaS lifecycle model, access to the logging systems of the platforms, a smoothly working unified deployment process and even support for platform specific services, the prototype achieves very good abstraction and covers the core parts of all PaaS systems. The simulation of an application's lifecycle within the adapter tests proved that PaaSal can be used to manage and deploy PaaS applications on cloudControl, Cloud Foundry V2, Heroku and Openshift V2.
Even though the abstraction layer already copes with the core aspects of PaaS platforms, there are still many open challenges remaining. Data management features are not included at all. Application statistics, vertical scaling and the support for global services are not supported on all platforms, but could be of high importance to quite a few developers. Further improvements are therefore also to be influenced by the feedback this first prototype receives upon its public release.
In conclusion, the project was a complete success and the prototype of the Platform as a Service abstraction layer performs exactly as it was originally intended. The abstraction layer's design proved itself as very robust and conformed to all special cases that the platforms disclosed during the implementation. Diversities concerning the runtimes, models and operations could be successfully harmonized amongst the four platforms. With its unified deployment and management capabilities, the prototype allows to manage applications on different platforms. In case of switching the provider, the effort to adapt the application's surrounding, for instance to enable continuous delivery, can be minimized. PaaSal increases the portability and interoperability of PaaS applications and thus helps to avoid critical vendor lock-in effects.

8.1 Future Work

Based on the current state of the defined abstraction layer and its prototype, there still are 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 Apprenda[98], Flynn[99], the upcoming release V3 of Openshift[100] 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 thereby 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 the small, medium and large gears on Openshift, represent fixed hardware specifications. One approach could thereby 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, for instance Cloud Foundry, allow the creation of general services that must not be bound and can be used by multiple applications.
In order to reduce the need of using multiple tools, PaaSal could integrate basic data management features so that the contents of databases could be imported and extracted. Besides the still relevant technical challenges, this feature would remove one of the remaining barriers to enable an automated migration of PaaS application, or at least help to further reduce the existing migration efforts [KLW15].

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 from the above-named implementation tasks to a more general perspective, PaaSal could be integrated with 3rd party systems to gain additional insights on PaaS platforms and their capabilities. If PaaSal would be integrated with the PaaS profiles project, both projects could improve their quality. PaaSal 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 PaaSal API. Combining both projects, the emerging system would evolve to become a basic 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.
With the findings of the vendor's platform evaluation, it would be plausible to update and extend the PaaS Taxonomy [KW14]. New categories could be the utilized application technology, referring to the use of virtual machines or container based solutions, as well as the API authentication mechanisms. Within the service category, it could be noted whether the platform allows services to be used by multiple applications. Further important aspects are the used runtime format, for instance buildpacks or cartridges, and if SSL certificates can be uploaded to be used with custom domains.

[98] *Apprenda*. URL: `http://apprenda.com/` (Retrieved: June 28, 2015).
[99] *Flynn*. URL: `https://flynn.io/` (Retrieved: June 28, 2015).
[100] *Openshift Origin 3 Repository*. URL: `https://github.com/openshift/origin` (Retrieved: May 3, 2015).

References

[Art89] W. Brian Arthur. Competing technologies, increasing returns, and lock-in by historical events. *The economic journal*:116–131, 1989. JSTOR: 2234208.

[Bad12] Lee Badger. *Cloud Computing Synopsis and Recommendations: Recommendations of the National Institute of Standards and Technology*. Createspace, 2012. ISBN: 978-1-4776-2105-9.

[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, pages 5–14.

[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):1–4, 2014.

[Bra14] T. Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159 (Proposed Standard). Internet Engineering Task Force, March 2014.

[Car13] Darryl Carlton. Cloud Computing 2014: ready for real business? Gartner, Inc., October 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 (SeaClouds2014). September 2, 2014, Manchester (UK)*, 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), November 2009.

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

[CNS13] David Cunha, Pedro Neves, and Pedro Sousa. A Platform-as-a-Service API Aggregator. In Álvaro Rocha, Ana Maria Correia, Tom Wilson, and Karl A. Stroetmann, editors, *Advances in Information Systems and Technologies*. Volume 206, pages 807–818. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN: 978-3-642-36981-0.

[CNS14] David Cunha, Pedro Neves, and Pedro Sousa. PaaS manager: A platform-as-a-service aggregation framework. *Computer Science and Information Systems*, (00):28–28, 2014. ISSN: 1820-0214. DOI: 10.2298/CSIS130828028C.

[DBCAZ12] Francesco D'Andria, Stefano Bocconi, Jesus Gorronogoitia Cruz, James Ahtes, and Dimitris Zeginis. Cloud4soa: Multi-cloud Application Management Across PaaS Offerings. In. IEEE, September 2012, pages 407–414. ISBN: 978-1-4673-5026-6. DOI: 10.1109/SYNASC.2012.65.

[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.

[Fie00] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. PhD thesis. University of California, Irvine, 2000.

[FK06] Joseph Farrell and Paul Klemperer. Coordination and Lock-In: Competition with Switching Costs and Network Effects. *SSRN Electronic Journal*, 2006. ISSN: 1556-5068. DOI: 10.2139/ssrn.917785.

[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 MetaCloud. In *CRC 2012: 12a Conferência sobre Redes de Computadores*, 2013, pages 7–13.

[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 (JITTA)*, 12(3):1, 2012.

[Gut13] John Guttag. *Introduction to computation and programming using Python*. The MIT Press, Cambridge, Massachusetts, spring 2013 edition edition, 2013. 268 pages. ISBN: 0-262-51963-1 978-0-262-51963-2.

[GZC13] Francis Galiegue, Kris Zyp, and Gary Court. JSON Schema: core definitions and terminology. Internet Engineering Task Force (IETF), January 2013.

[HLST11] Michael Hogan, Fang Liu, Annie Sokol, and Jin Tong. Nist cloud computing standards roadmap. *NIST Special Publication*, 35, 2011.

[IEE90] IEEE Computer Society and Standards Coordinating Committee and Institute of Electrical and Electronics Engineers and IEEE Standards Board and American National Standards Institute. *IEEE standard glossary of software engineering terminology*. Institute of Electrical and Electronics Engineers, New York, N.Y., USA, 1990. ISBN: 155937067X 9781559370677.

[Int93] International Organization for Standardization. ISO/IEC 2382-1. 1993.

[KLW15] Stefan Kolb, Jörg Lenhard, and Guido Wirtz. Application Migration Effort in the Cloud – The Case of Cloud Platforms. In *Proceedings of the 8th IEEE International Conference on Cloud Computing (CLOUD)*. International Conference on Cloud Computing (CLOUD). IEEE, New York, NY, USA, July 2015.

[KLZ+13] Eleni Kamateri, Nikolaos Loutas, Dimitris Zeginis, James Ahtes, Francesco D'Andria, Stefano Bocconi, Panagiotis Gouvas, Giannis Ledakis, Franco Ravagli, Oleksandr Lobunets, et al. Cloud4soa: A Semantic-Interoperability PaaS Solution for Multi-cloud Platform Management and Portability. In, *Service-Oriented and Cloud Computing*, pages 64–78. Springer, 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 (SOSE). IEEE, Oxford, United Kingdom*. IEEE, April 2014.

[Lew13] Grace A. Lewis. Role of Standards in Cloud-Computing Interoperability. In. IEEE, January 2013, pages 1652–1661. ISBN: 978-0-7695-4892-0. DOI: 10.1109/HICSS.2013.470.

[LKT11] Nikolaos Loutas, Eleni Kamateri, and Konstantinos Tarabanis. A Semantic Interoperability Framework for Cloud Platform as a Service. In. IEEE, November 2011, pages 280–287. DOI: 10.1109/CloudCom.2011.45.

[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)*:993–995, 2012. ISSN: 2159-6182. DOI: 10.1109/CLOUD.2012.111.

[MG11] Peter Mell and Tim Grance. The NIST definition of cloud computing, 2011.

[MLBZG11] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing — The business perspective. *Decision Support Systems*, 51(1):176–189, April 2011. ISSN: 01679236. DOI: 10.1016/j.dss.2010.12.006.

[NLP+11] Yefin V. Natis, Benoit J. Lheureux, Massimo Pezzini, David W. Cearly, Eric Knipp, and Daryl C. Plummer. PaaS Road Map: A Continent Emerging. Gartner, Inc., 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*, pages 105–115. Springer, 2010.

[PCR11] Dana Petcu, Ciprian Craciun, and Massimiliano Rak. Towards a cross platform Cloud API. *Components for Cloud Federation, Procs. CLOSER*:166–169, 2011.

[Pet11] Dana Petcu. Portability and interoperability between clouds: challenges and case study. In, *Towards a Service-Based Internet*, pages 62–74. Springer, 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 *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on Cloud Computing*. IEEE, 2012, pages 392–399.

[PMPC13] Dana Petcu, Georgiana Macariu, Silviu Panica, and Ciprian Crăciun. Portable Cloud applications—From theory to practice. *Future Generation Computer Systems*, 29(6):1417–1430, August 2013. ISSN: 0167739X. DOI: 10.1016/j.future.2012.01.009.

[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.

[RR11] Stefan Ried and John R. Rymer. The Forrester Wavetm: Platform-As-A-Service For App Dev And Delivery Professionals, Q2 2011. Forrester, May 19, 2011.

[SR10] Amit Sheth and Ajith Ranabahu. Semantic modeling for cloud computing, part 2. *Internet Computing, IEEE*, 14(4):81–84, 2010.

[SYMT13] Mohamed Sellami, Sami Yangui, Mohamed Mohamed, and Samir Tata. PaaS-Independent Provisioning and Management of Applications in the Cloud. In. IEEE, June 2013, pages 693–700. ISBN: 978-0-7695-5028-2. DOI: 10.1109/CLOUD.2013.105.

[Tan06] Andrew S. Tanenbaum. *Structured computer organization*. Pearson Prentice Hall, Upper Saddle River, N.J, 5th ed edition, 2006. 777 pages. ISBN: 0-13-148521-0.

[Tel13] Telecom SudParis, Computer Science Department. The Compatible One Application and Platform Service (COAPS) API specification (draft). Version 1.5.3. May 15, 2013.

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

[YBD08] Lamia Youseff, Maria Butrico, and Dilma Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE'08*. IEEE, 2008.

[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 16, 2013. ISSN: 1895-1767. DOI: 10.12694/scpe.v14i1.824.

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/paasal	GET /v2/apps/{aid}	GET /apps/{aid}	GET /application/{aid}
GET Application List	GET /app/ **For each loaded application:** • GET /app/{aid}/deployment/paasal	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 = 'paasal' POST /app/{application/name}/deployment/{dplid}/addon *Body:* • addon = 'config.free' • options = '{"paasal-initialized": "true"}' PUT /app/{application/name}/deployment/{dplid}/addon/config.free *Body:* • addon = 'config.free' • settings = '{"paasal-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/paasal GET /user *Git: Clone repo, extract file, commit, push* **If the application was already started:** PUT /app/{aid}/deployment/paasal *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/paasal GET /user *Git: Clone repo, modify marker, commit, push* PUT /app/{aid}/deployment/paasal *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/paasal *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/paasal PUT /app/{aid}/deployment/paasal *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/paasal *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 PaaSal API to vendor specific operations

Appendix B - Adapter test spec template

Listing 27: Adapter test spec template

```
require 'spec/adapter/adapter_spec_helper'

describe Paasal::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: 'paasaltestappminproperties', updated_name: '
        paasaltestappminproperties', region: 'default' }
    @app_all = { original_name: 'paasaltestappallupdated', updated_name: '
        paasaltestappallupdated', 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

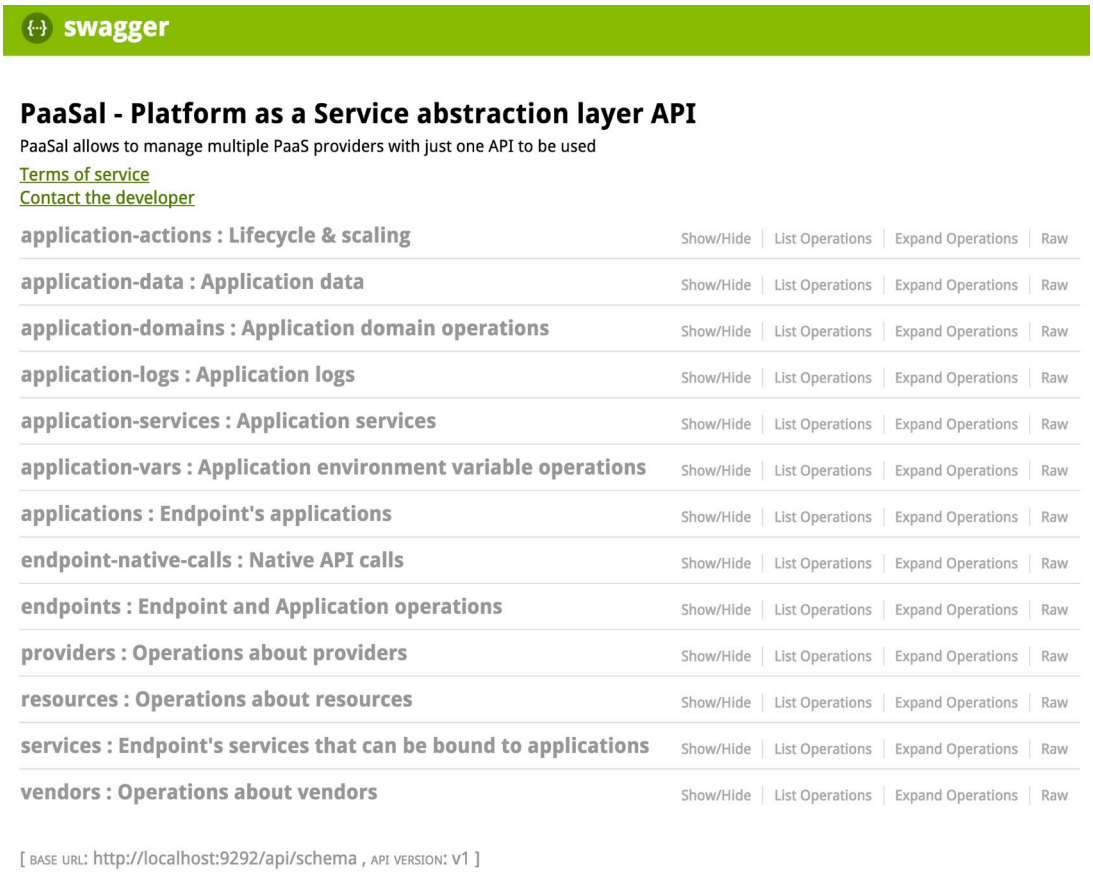

Figure 17: Swagger UI Overview, showing the grouped API objects and operations

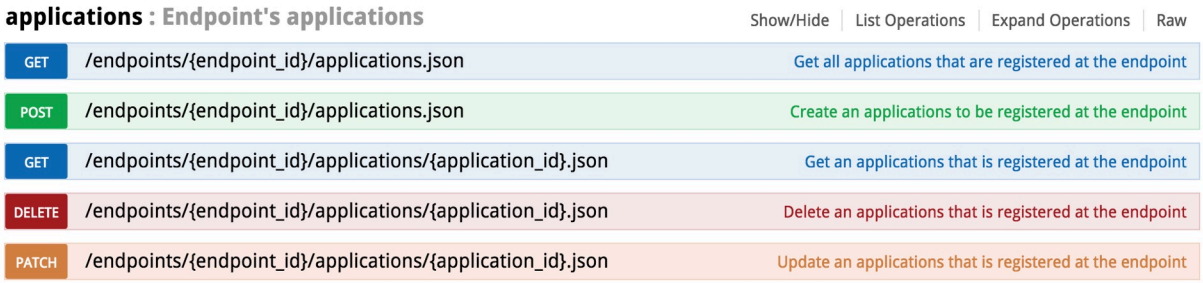

Figure 18: Swagger UI showing all methods available in an operation group

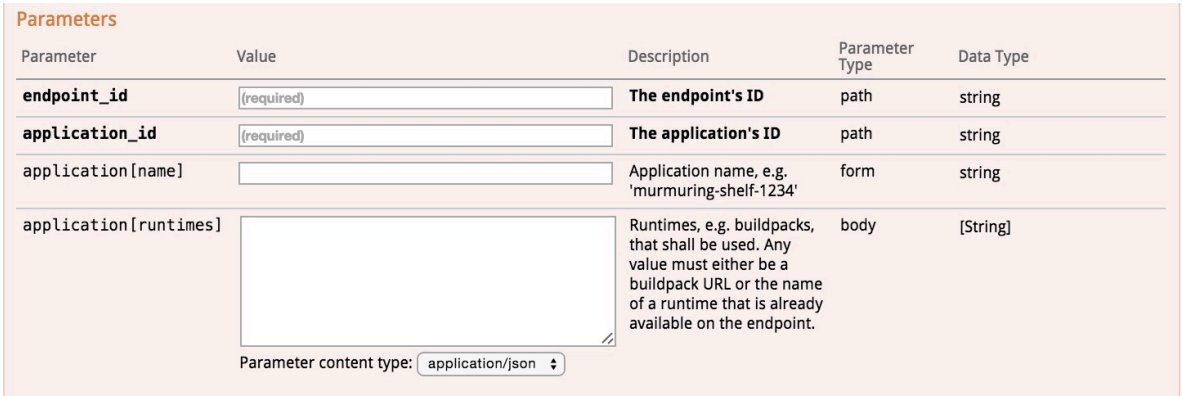

Figure 19: Swagger UI presentation of the PATCH request to update an application

Figure 20: Swagger UI presentation of the GET request to list all applications

Appendix D - Test Evaluation Request Count

recorded test cassette	cloudControl	Cloud Foundry v2	Heroku	Openshift v2
_auth_client.json		1		
app-actions/lifecycle/fail/restart/fails.json		4	6	6
app-actions/lifecycle/fail/restart/subsq_req_shows_no_state_changes.json		7	6	6
app-actions/lifecycle/fail/start/fails.json	5	4	6	6
app-actions/lifecycle/fail/start/subsq_req_shows_no_state_changes.json	6	7	6	6
app-actions/lifecycle/fail/stop/fails.json		4	6	6
app-actions/lifecycle/fail/stop/subsq_req_shows_no_state_changes.json		7	6	6
app-actions/lifecycle/restart/succeeds_for_app_all_if_currently_running.json		16	23	14
app-actions/lifecycle/restart/succeeds_for_app_all_if_currently_stopped.json		16	23	14
app-actions/lifecycle/restart/succeeds_for_app_min_if_currently_running.json		16	23	14
app-actions/lifecycle/restart/succeeds_for_app_min_if_currently_stopped.json		16	23	14
app-actions/lifecycle/start/succeeds_for_app_all_if_already_running.json	9	11	14	14
app-actions/lifecycle/start/succeeds_for_app_all_if_currently_stopped.json	10	231	16	16
app-actions/lifecycle/start/succeeds_for_app_min_if_already_running.json	9	11	14	14
app-actions/lifecycle/start/succeeds_for_app_min_if_currently_stopped.json	10	223	24	16
app-actions/lifecycle/stop/succeeds_for_app_all_if_already_stopped.json		11	14	14
app-actions/lifecycle/stop/succeeds_for_app_all_if_currently_running.json		11	14	14
app-actions/lifecycle/stop/succeeds_for_app_min_if_already_stopped.json		11	14	14
app-actions/lifecycle/stop/succeeds_for_app_min_if_currently_running.json		11	14	14
app-actions/scaling/fails/with_0_instances_value.json	1	2	1	1
app-actions/scaling/fails/with_invalid_instances_property_type.json	1	2	1	1
app-actions/scaling/fails/with_missing_instances_property.json	1	2	1	1
app-actions/scaling/fails/with_negative_instances_value.json	1	2	1	1
app-actions/scaling/succeeds/with_scale-in_and_removes_an_application_instance.json	3	6	6	9
app-actions/scaling/succeeds/with_scale-out_and_adds_an_application_instance.json		6		9
app-data/deploy/fails_for/corrupted_tar_gz_archive.json	9	3	6	9
app-data/deploy/fails_for/corrupted_zip_archive.json	9	3	6	9
app-data/deploy/fails_for/unsupported_archive_tbz2.json	1	2	1	1
app-data/deploy/fails_for/unsupported_archive_tbz2_but_with_supported_mime_type.js	9	3	6	9
app-data/deploy/succeeds/and_app_with_all_properties.json	6	7	6	6
app-data/deploy/succeeds/and_app_with_min_properties.json	6	7	6	6
app-data/deploy/succeeds/with_valid_tar_gz_application_archive.json	9	4	7	13
app-data/deploy/succeeds/with_valid_zip_application_archive.json	9	4	7	13
app-data/download/fails/for_non-existing_application.json	3	3	2	4
app-data/download/fails/with_invalid_archive_format_rar.json	1	2	1	1
app-data/download/of_type_tar_gz_fails_when_there_is_no_deployment.json	5	4	2	6
app-data/download/succeeds/for_archive_format_tar_gz.json	5	5	6	8
app-data/download/succeeds/for_default_archive_format_zip.json	5	5	6	8
app-data/download/with_default_type_fails_when_there_is_no_deployment.json	5	4	2	6
app-data/rebuild/changes_the_release_version_property.json	13	15	19	25
app-data/rebuild/fails_when_there_is_no_deployment.json	5	4	2	6
app-data/rebuild/succeeds.json	9	7	11	15
app-domains/create/fails/with_invalid_name/email_as_name.json	1	2	1	1
app-domains/create/fails/with_invalid_name/malformed_name_with_multiple_dots.json	1	2	1	1
app-domains/create/fails/with_invalid_name/malformed_name_with_trailing_dot.json	1	2	1	1
app-domains/create/fails/with_invalid_name/name_without_TLD.json	1	2	1	1
app-domains/create/fails/with_missing_name.json	1	2	1	1
app-domains/create/if_the_name_is_already_used.json	2	7	2	4
app-domains/create/succeeds/with_hostname_in_the_name.json	3	12	2	4
app-domains/create/succeeds/without_hostname_in_the_name.json	3	13	2	4
app-domains/delete/fails_for/non-existing_application.json	2	3	2	4
app-domains/delete/fails_for/non-existing_domain_id.json	2	4	2	4
app-domains/delete/succeeds/for_previously_created_entity_with_hostname.json	5	11	3	7
app-domains/delete/succeeds/for_previously_created_entity_without_hostname.json	4	10	3	7
app-domains/get/fails_for_non-existent_domain.json	2	3	2	4
app-domains/get/succeeds/with_hostname.json	5	10	3	7
app-domains/get/succeeds/without_hostname.json	5	10	3	7
app-domains/list.json	2	5	2	4
app-domains/list/fails_for_non-existing_application.json	2	3	2	4
app-domains/list/succeeds.json	4	7	2	4

recorded test cassette	cloudControl	Cloud Foundry v2	Heroku	Openshift v2
app-logs/download-all/fails/with_invalid_archive_format_log.json	1	2	1	
app-logs/download-all/fails/with_invalid_archive_format_rar.json	1	2	1	
app-logs/download-all/fails/with_non-existing_application.json	2	3	2	
app-logs/download-all/succeeds/as_tar_gz.json	29	36	27	
app-logs/download-all/succeeds/as_zip.json	29	36	27	
app-logs/download/fails/with_invalid_file_format_rar.json	1	2	1	
app-logs/download/fails/with_non-existing_application.json	2	3	2	
app-logs/download/fails/with_non-existing_log.json	2	3	2	
app-logs/download/succeeds/for_type_request_as_log.json	5	6	7	
app-logs/download/succeeds/for_type_request_log_as_tar_gz.json	5	6	7	
app-logs/download/succeeds/for_type_request_log_as_zip.json	5	6	7	
app-logs/get/fails/with_non-existent_log_id.json	2	5	2	
app-logs/get/fails/with_non-existing_application.json	2	3	2	
app-logs/get/of_type_request.json	3	4	4	
app-logs/get/with_empty_results_for_type_request.json	3	4	4	
app-logs/list/fails_for_non-existing_application.json	2	3	2	
app-logs/list/succeeds.json	2	6	2	
app-logs/tail/fails/with_non-existent_log_id.json	2	6	2	
app-logs/tail/fails/with_non-existing_application.json	2	3	2	
app-logs/tail/request.json		13	11	
app-services/add/fails/if_the_service_is_already_assigned.json	2	8	3	6
app-services/add/fails/with_invalid_plan.json	2	6	2	5
app-services/add/fails/with_invalid_service.json	2	4	2	4
app-services/add/fails/with_missing_plan.json	1	2	1	1
app-services/add/fails/with_missing_service.json	1	2	1	1
app-services/add/succeeds/with_1st_invocation.json	3	10	6	6
app-services/change/fails/with_non-existent_application.json	2	3	3	
app-services/change/fails/with_non-existent_plan.json	3	5	5	
app-services/change/fails/with_non-existent_service.json	2	4	3	
app-services/get/fails/with_non-existent_application.json	2	3	3	4
app-services/get/fails/with_non-existent_service.json	2	4	3	4
app-services/get/succeeds.json	4	8	7	4
app-services/list.json	2	4	2	4
app-services/list/fails_for_non-existing_application.json	2	3	2	4
app-services/list/succeeds.json	3	6	5	4
app-services/remove/fails/with_non-existent_application.json	2	3	3	4
app-services/remove/fails/with_non-existent_service.json	2	4	3	4
app-services/remove/succeeds.json	3	8	5	4
app-vars/create/fails/if_the_key_is_already_used.json	2	4	2	4
app-vars/create/fails/with_missing_key_property.json	1	2	1	1
app-vars/create/fails/with_missing_value_property.json	1	2	1	1
app-vars/create/succeeds/no_2_using_app_all/did_not_alter_other_vars.json	2	4	2	4
app-vars/create/succeeds/no_2_using_app_all/succeeds.json	3	6	3	4
app-vars/create/succeeds/using_app_all.json	3	6	3	4
app-vars/create/succeeds/using_app_with_min_properties.json	3	6	3	4
app-vars/delete/did_not_alter_other_vars.json	2	4	2	4
app-vars/delete/fails_for/non-existing_application.json	2	3	2	4
app-vars/delete/fails_for/non-existing_key.json	2	4	2	4
app-vars/delete/succeeds.json	3	6	3	5
app-vars/get/fails_for_non-existent_key.json	2	3	2	4
app-vars/get/succeeds.json	2	4	2	4
app-vars/list/fails_for_non-existing_application.json	2	3	2	4
app-vars/list/succeeds.json	2	4	2	4
app-vars/list/with_empty_result_list.json	2	4	2	4
app-vars/update/did_not_alter_other_vars.json	2	4	2	4
app-vars/update/fails/with_missing_value.json	1	2	1	1
app-vars/update/fails/with_non-existing_key.json	2	4	2	4
app-vars/update/succeeds.json	3	6	3	4
app/create/fails/in_2nd_attempt_with_duplicate_name.json	2	4	2	4
app/create/fails/with_invalid/region.json	1	2	2	3
app/create/fails/with_invalid/runtimes/by_bad_URL_and_unknown_name.json	1	4	1	2
app/create/fails/with_missing/name_property.json	1	2	1	1
app/create/fails/with_missing/runtimes_property.json	1	2	1	1
app/create/succeeds/of_type_nodejs_with_all_properties.json	9	14	8	8

recorded test cassette	cloudControl	Cloud Foundry v2	Heroku	Openshift v2
app/create/succeeds/of_type_nodejs_with_minimal_properties.json	9	14	7	7
app/delete/fails_for_non-existing_application.json	2	3	2	4
app/delete/for_created_application/with_all_properties/makes_GET_of_that_app_retur	2	3	2	4
app/delete/for_created_application/with_all_properties/succeeds.json	4	7	2	4
app/delete/for_created_application/with_min_properties/makes_GET_of_that_app_retu	2	3	2	4
app/delete/for_created_application/with_min_properties/succeeds.json	4	7	2	4
app/get/fails_for_non-existent_application.json	2	3	2	4
app/get/succeeds.json	6	7	6	6
app/list.json	14	12	14	6
app/web_access/for_app_with_all_properties.json	4	7	6	7
app/web_access/for_app_with_min_properties.json	5	13	7	8
application_update/app/update/fails.json		3	2	
application_update/app/update/succeeds/changing_name_and_runtimes.json		7	7	
application_update/app/update/succeeds/changing_only_the_name.json		7	6	
application_update/app/update/succeeds/changing_only_the_runtimes.json		7	7	
application_update/app/update/succeeds/reverting_runtime_change_for_app_all.json		7	7	
application_update/app/update/succeeds/reverting_runtime_change_for_app_min.json		7	7	
regions/get.json	1	2	2	2
regions/list.json	1	2	2	2
service-plans/get/fails_for/get_of_service_plan_with_non-existent_service.json	2	3	2	2
service-plans/get/fails_for/get_of_service_plan_with_non-existent_service_plan.json	2	4	2	2
service-plans/get/succeeds.json	2	5	2	2
service-plans/list/fails_for_get_of_service_plan_with_non-existent_service.json	2	3	2	2
service-plans/list/succeeds.json	2	4	2	2
services/get/fails_for_get_of_non-existent_service.json	2	3	2	2
services/get/succeeds.json	2	4	3	2
services/list.json	2	3	152	2
Tested methods	130	151	149	121
Total vendor API requests	480	1326	883	672
Avg. vendor API requests per tested method	3.69	8.78	5.93	5.55
Tested methods without repeated requests	117	138	136	108
Total vendor API requests without repeated requests	426	737	679	504
Avg. vendor API requests per tested method without repeated requests	3.64	5.34	4.99	4.67

Table 34: Requests per method test case for all vendors