UFRN – FEDERAL UNIVERSITY OF RIO GRANDE DO NORTE
PPgSC – POSTGRADUATE PROGRAM IN SYSTEMS AND COMPUTING

SAMUEL DE MEDEIROS QUEIROZ

INFRASTRUCTURE AS A SERVICE INTRA-PLATFORM INTEROPERABILITY:
An Exploratory Study with OpenStack

NATAL – RN
2018

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Setorial Prof. Ronaldo Xavier de Arruda - CCET

Queiroz, Samuel de Medeiros.
Infrastructure as a service intra-platform interoperability: an exploratory study with OpenStack / Samuel de Medeiros Queiroz. - 2018.
261f.: il.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Programa de Pós-Graduação em Sistemas e Computação. Natal, 2018.
Orientadora: Thais Vasconcelos Batista.

1. Computação - Dissertação. 2. Interoperabilidade - Dissertação. 3. Aprisionamento tecnológico - Dissertação. 4. Infraestrutura como um serviço - Dissertação. 5. Intra-plataforma - Dissertação. 6. Multinuvem - Dissertação. 7. OpenStack - Dissertação. I. Batista, Thais Vasconcelos. II. Título.

RN/UF/CCET CDU 004

Elaborado por Joseneide Ferreira Dantas - CRB-15/324

SAMUEL DE MEDEIROS QUEIROZ

INFRASTRUCTURE AS A SERVICE INTRA-PLATFORM INTEROPERABILITY:

An Exploratory Study with OpenStack

Work submitted to the Postgraduate Program in Systems and Computing (PPgSC) of the Federal University of Rio Grande do Norte (UFRN) in complete fulfillment of the requirements to obtain the title of Master of Science in Computer Science.

Research topic: Integrated and Distributed Systems

Supervisor: Prof. Thais Vasconcelos Batista

NATAL – RN

2018

SAMUEL DE MEDEIROS QUEIROZ

"Infrastructure as a Service Intra-Platform Interoperability: An Exploratory Study with OpenStack"

Esta Dissertação foi julgada adequada para a obtenção do título de mestre em Sistemas e Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.

Presidente: Dra. THAIS VASCONCELOS BATISTA (Orientador - UFRN)

Dr. BRUNO MOTTA DE CARVALHO
(Coordenador do Programa)

Banca Examinadora

Examinadora: Dr. NÉLIO ALESSANDRO AZEVEDO CACHO (UFRN)

Examinador: Dr. ANDREY ELÍSIO MONTEIRO BRITO (UFCG)

Examinador: Dr. JACQUES PHILIPPE SAUVÉ (UW)

Dezembro, 2018

To my God for the gifts of faith, life, health and curiosity.

To my wife and children, for the support and abdication allowing me to investigate the curiosity aroused.

To my parents, for guidance in good principles and for providing me with high quality education, despite the unfavorable conditions.

To my brothers, for the fellowship and unity since the childhood.

To all my professors who since the kindergarten have instigated me in the pursuit of knowledge in diverse areas, especially to the advisor of this dissertation.

Lastly, to my friends and family who contributed directly or indirectly to the realization of what is one more swimming stroke in the ocean of knowledge.

Ao meu Deus, pelos dons da fé, da vida, da saúde e da curiosidade.

A minha esposa e filhos, pelo suporte e abdicação, permitindo-me investigar a curiosidade despertada.

Aos meus pais, pela orientação nos bons princípios e por me proporcionarem educação de alta qualidade, apesar das condições não favoráveis.

Aos meus irmãos, pelo companheirismo e união desde a infância.

Aos meus professores que desde a educação infantil me instigam na busca do conhecimento nas mais diversas áreas, em especial à orientadora desta dissertação.

Por fim, a todos os meus amigos e familiares que contribuíram direta ou indiretamente para a realização de mais uma braçada no oceano do saber.

ABSTRACT

The emergence of new digital technologies comes with challenging technical and business requirements. The traditional approach to provide computational infrastructure to application workloads, which relies on in-house management of hardware, does not present technical and cost-effective attributes to deliver high-performance, reliability and scalability. As one of the biggest technologic paradigms shift in the history of humanity, cloud computing allows diverse deployment and service model alternatives, suitable to diverse requirements, such as security, latency, computational performance, availability and cost. Therefore, numerous companies distribute hundreds of clouds worldwide, creating an equitable market through competition, where players create unique features to differentiate from competitors. Consequently, in the consumer side, picking a vendor typically translates into vendor lock-in, a situation where the applications heavily depend on the vendor's approach of exposing features, making it difficult to switch between vendors whenever convenient or to support complex scenarios across multiple distributed heterogeneous clouds, such as federation. An immediate work-around for users is to pick cloud solutions that implement standards or post-facto open source platforms, such as OpenStack, which are assumed to provide native interoperability between installations. In the industry, however, OpenStack proves that the lack of interoperability is a real concern even between its deployments, due the high flexibility and complexity of supported use cases. Therefore, this investigation documents intra-platform interoperability, as in OpenStack, starting by presenting in detail the Python client library created by the community to abstract deployment differences, counting with numerous and significant contributions from the author of this dissertation. Afterwards, an extensive validation of that library was performed across one testing and five production clouds from different vendors worldwide, because despite the fact the library is extensively used by the community, it had never been validated in detail. The validation unveiled bugs, functionality and documentation gaps. Since intra-platform interoperability had never been documented in this context, a systematic literature review followed, allowing a deep comparison of the state of the art of vendor lock-in in opposition to that library, presenting its advantages, disadvantages and recommendations for users. Lastly, the suggestions for future work include support for multiple programming languages and the adoption of the client library as a standard for inter-platform interoperability.

Keywords: Interoperability, Vendor lock-in, Infrastructure as a Service, IaaS, Intra-platform, Multicloud, OpenStack, OpenStack SDK.

RESUMO

O surgimento de novas tecnologias digitais traz consigo requisitos técnicos e de negócio desafiadores. A abordagem tradicional para prover infraestrutura computacional, que depende no gerenciamento de hardware nas premissas do consumidor, não é mais tecnicamente ou financeiramente viável para entregar alta performance, confiabilidade e escalabilidade. Como uma das maiores mudanças de paradigma tecnológico da história da humanidade, computação em nuvem permite vários modelos de implantação e de serviço, adequado à diversos requisitos como segurança, latência, desempenho, disponibilidade e custo. Por consequência, empresas distribuem centenas de nuvens em todo o mundo, criando um mercado saudável através de competição, onde combinações únicas de funcionalidades diferenciam competidores. Desta forma, como consumidor, selecionar um provedor geralmente se traduz em aprisionamento de fornecedor, uma situação onde a as aplicações tem alta dependência naquela abordagem de expor funcionalidades, tornando difícil a mudança de provedor sempre que conveniente ou o suporte a cenários envolvendo múltiplas nuvens distribuídas e heterogêneas, como federação. Uma saída é a seleção de provedores que adotam padrões de interoperabilidade ou utilizam plataformas de código aberto consolidadas, tal como OpenStack, que são consideradas interoperáveis entre suas instalações. Todavia, na indústria, OpenStack prova que a falta de interoperabilidade é uma preocupação mesmo entre suas instalações, devido à alta flexibilidade e complexidade dos casos de uso suportados. Desta forma, esta investigação documenta interoperabilidade intra-plataforma no OpenStack, iniciando com o detalhamento da biblioteca Python criada pela comunidade para abstrair as diferenças das suas instalações, e que conta com numerosas e significantes contribuições do autor desta dissertação. Em seguida, uma validação foi realizada em uma nuvem de testes e seis de produção de vários fornecedores espalhados pelo mundo, dado que apesar da biblioteca ser vastamente utilizada pela comunidade, ela nunca havia sido validada em detalhes, revelando falhas e lacunas de funcionalidades e documentação. Uma vez que a interoperabilidade intra-plataforma nunca foi reportada neste contexto, uma revisão sistemática foi realizada, permitindo comparar o estado da arte com a biblioteca, descrevendo suas vantagens, desvantagens e recomendações para consumidores. Por fim, as sugestões de trabalhos futuros incluem o suporte a múltiplas linguagens de programação e a adoção da biblioteca como um padrão para interoperabilidade inter-plataforma.

Palavras-chave: Interoperabilidade, Aprisionamento Tecnológico, Infraestrutura como um Serviço, IaaS, Intra-plataforma, Multinuvem, OpenStack, OpenStack SDK.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1 INTRODUCTION

The exponential advancement in computational processing, storage and networking capacity allowed new digital technologies as Internet of Things, social, mobile, analytics, blockchain, robotics, virtual reality and artificial intelligence to emerge, affecting the global economy and changing the way the human society interacts with the world.

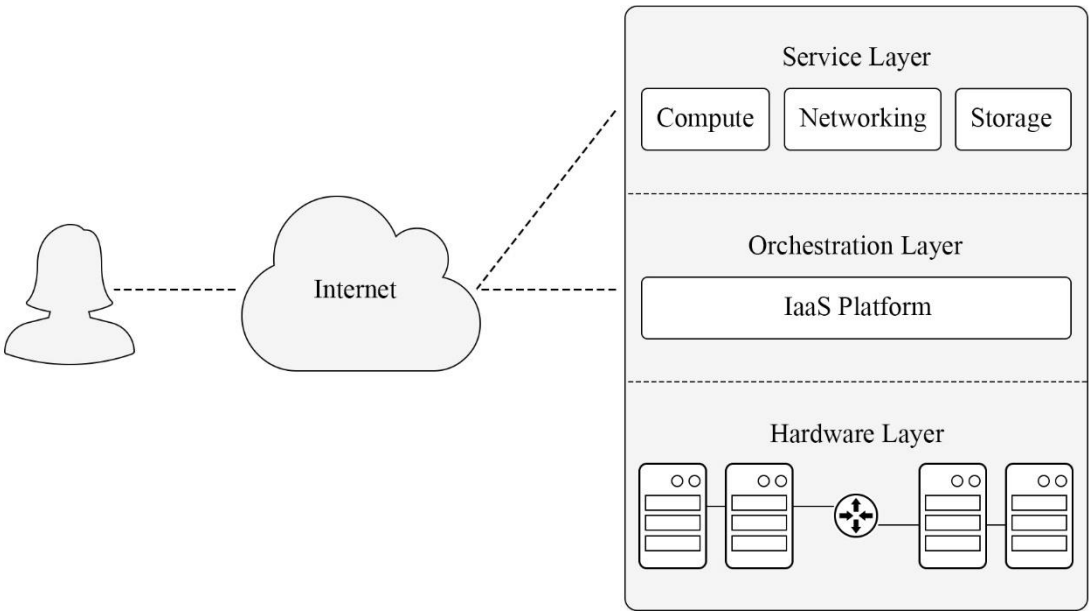
These new digital technologies require rapid provisioning of reliable, high-performant, and scalable infrastructure. The traditional Information Technology approach to provide infrastructure for applications rely on private, on-premise management of physical resources, which implies on high investments on acquiring, allocating and managing expensive hardware, while at the same time companies seek resources optimization and costs reduction. Such a scenario presents threats to new application requirements, demanding a new approach to infrastructure provisioning in one of the biggest technology paradigms shift in the history of humanity.

Cloud computing is a business model for enabling access to configurable computing resources over the network via on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service. Its main service models enable infrastructure, platform and software as a service via private, community, public or hybrid deployment models (MELL and GRANCE, 2011).

In the Infrastructure as a Service (IaaS) model, end users have the plainest access to computational resources through servers: computing, networking and storage resources are virtually controlled, allowing full flexibility for customers to set up and configure services just as in physical, on-premise, traditional servers. While providing flexibility, this model removes all the complexity of allocating and managing hardware, allowing companies to optimize resources, save investments and focus on the core business.

The most popular deployment model is the public cloud, as illustrated in Figure 1, where the physical infrastructure belongs to a third-party service provider, along with all the complexity of managing and scaling it, which includes managing physical space, cooling, electrical power, security and many others. The virtualized physical resources are allocated on-demand, in a self-service manner, without differentiation between end users, resulting in a virtually isolated, multitenant environment, where there is little or no idea where the underlying physical infrastructure is.

Figure 1 - End user communicating with IaaS platform



Source: Created by the author of this document.

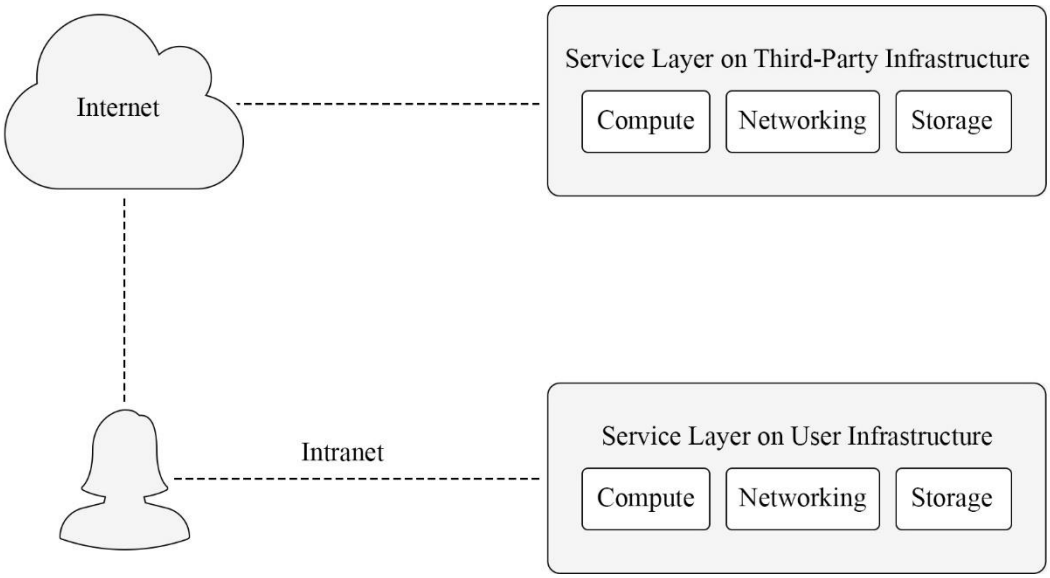
A few players, referred to as hyperscalers, dominate the public market, mostly notably represented by Amazon, Microsoft, Alibaba and Google, with the products Amazon Web Services, Microsoft Azure, Alibaba Cloud and Google Cloud Platform, respectively. Those enterprises spam physical servers around the globe, addressing a larger market by reducing network latency in many regions of the planet. Proven reliability, high availability and scalability are key aspects that differentiate the hyperscalers from other players.

For most applications, the hyperscalers' public offerings are the obvious choice of infrastructure outsourcing. However, some workloads might present characteristics that are blockers for a public model adoption:

- Compliance requirements and regulatory laws that determine strict data privacy policies, such as it cannot leave a country or an organization;
- Latency: despite having infrastructure available in multiple parts of the planet, some applications have very sensitive latency requirements, such as real-time processing;
- Performance: since the physical hardware configuration and performance is not always known, the performance of its virtualized infrastructure is not predictable;
- Cost: deploying applications requiring large pools of infrastructure is expensive;
- Multi-cloud: there are applications that require communicating to multiple clouds, such as in a scenario where processing is done at the edge of the network, in a private cloud that is close to end user devices, whereas long-living logs are stored in a cheap, slow storage pool of a public cloud.

Consequently, meeting the technical and businesses requirements of all applications is only possible with the use multiple clouds in hybrid deployments, as in Figure 2. Therefore, the immediate challenge emerging is the utilization of multiple environments without detriment of business and technical constraints, especially in dynamic businesses where constraints are constantly changing to respond to the applications' requirements.

Figure 2 - User communicating with multiple clouds in a hybrid model



Source: Created by the author of this document.

The sections in this introductory chapter are organized as follows: Section 1.1 provides an industry overview on the shift towards cloud, particularly IaaS; Section 1.2 exposes what information aroused the author's curiosity and consequently motivated this investigation; Section 1.3 enumerates the OpenStack initiatives to interoperability; Section 1.4 states the problems being addressed in this study, while Section 1.5 states what is the scope for this investigation and Section 1.6 explains how the problems were investigated; Lastly, Section 1.7 informs about the author's involvement in the OpenStack community and Section 1.8 shows the overall organization of the rest of this document.

1.1 Industry

In the industry, the global public cloud computing market is expected to generate 302.4 billion of U.S. dollars in revenue in 2020, approximately doubling the spends of 153.4 billion in 2017 (GARTNER, 2018). The revenue forecast by cloud strategy is shown in Figure 3,

highlighting the dominance of Software as a Service and the growth of IaaS in the market landscape.

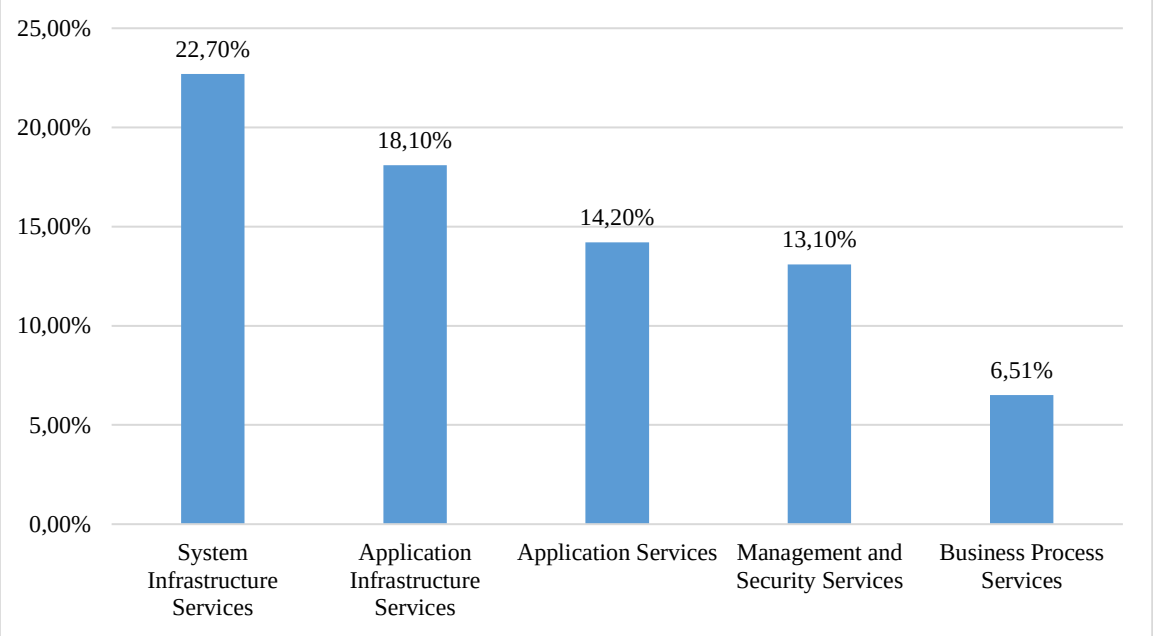
Figure 3 - Worldwide Public Cloud Service Revenue Forecast, Billions of U.S Dollars

Source: Gartner, April 2018 (GARTNER CLOUD FORECAST BLOG, 2018).

Between the cloud strategies, in a 2017 to 2021 market forecast, IaaS presents the highest Compound Annual Growth Rate (CAGR[3]), which is the mean annual growth rate of spends over that period, as in Figure 4, translating into high investments in developing this market in the upcoming years.

[3] $CAGR = \left(\frac{Ending\ value}{Beginnig\ value}\right)^{\left(\frac{1}{\#\ of\ years}\right)} - 1$

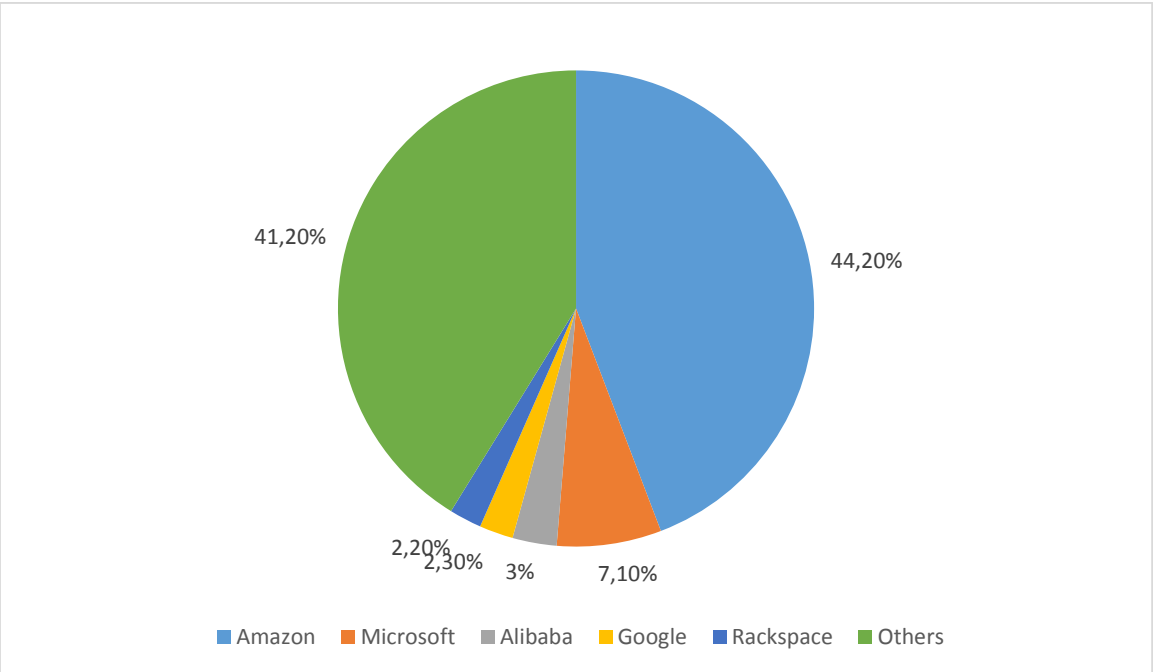
Figure 4 - CAGR by Cloud Service Category, 2016 - 2020



Source: Created by the author; CAGR calculated with data from Figure 1.

On hyperscalers, Gartner market analysts expect the top 10 providers to account for nearly 70 percent of the whole IaaS market by 2021, up from 50 percent in 2016 (GARTNER, 2018), when they had a total revenue of the 22,1 billion U.S dollars (GARTNER, 2017a), distributed as in Figure 5.

Figure 5 - IaaS Public Cloud Services Market Share, 2016



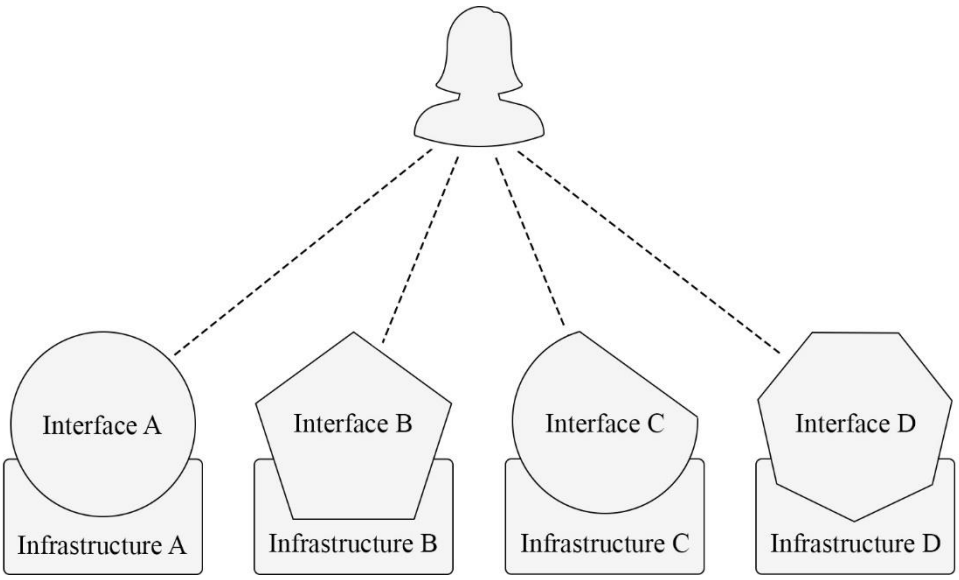
Source: Gartner, September 2017 (GARTNER, 2017a).

As mentioned in the previous section, the hyperscalers are not appropriate for every workload with only a public cloud strategy. Gartner's market analysts, who predict that by 2020, 90 percent of organizations will adopt hybrid infrastructure, confirm such a strategy (GARTNER, 2017b). Observing this market's opportunity, some hyperscalers, most notably the Amazon and Microsoft duopoly, start proposing hybrid strategies as extensions to their public cloud offerings, with the solutions VMWare Cloud on AWS (AMAZON WEB SERVICES, 2018) and Azure Stack (MICROSOFT, 2018). On the other hand, the competition between AWS and Azure will benefit sourcing executives in the short to medium term, but may be concerning in the longer term, since the lack of substantial competition could lead to an uncompetitive market, locking-in organizations into one platform by dependence on proprietary capabilities and potentially exposed to substantial price increases (GARTNER, 2017b).

1.2 Motivation

The heterogeneity of cloud deployments, with each cloud providing its capabilities via proprietary Application Programming Interfaces (APIs), data formats and varying Service Level Agreements (SLAs), presents vendor lock-in as a significant problem for end users, negatively impacting the widespread adoption of the cloud computing paradigm (OPARA-MARTINS, SAHANDI and TIAN, 2014), as illustrated in Figure 6.

Figure 6 - Multiple heterogeneous cloud description formats

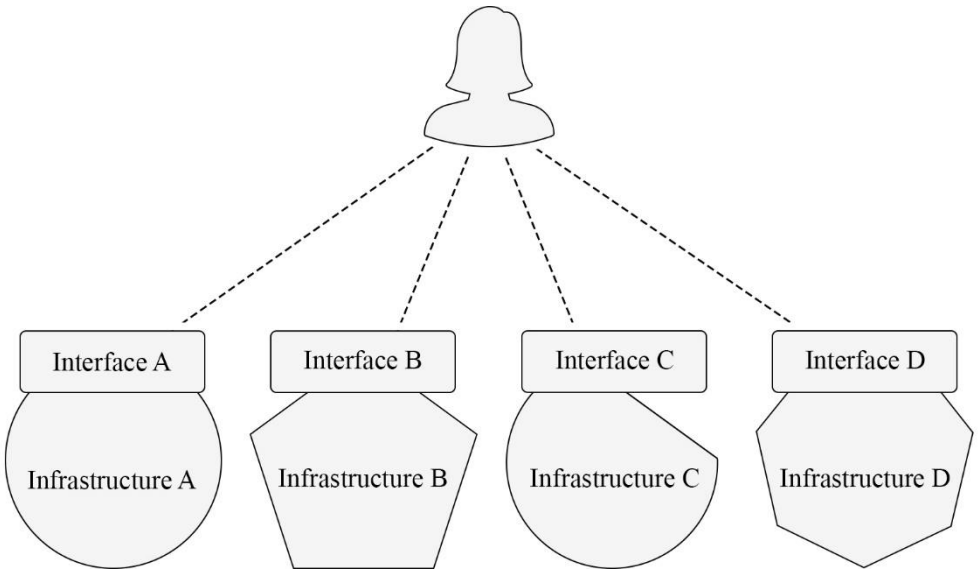


Source: Created by the author of this document.

Vendor lock-in is the most prominent challenge of adopting multiple cloud simultaneously without detriment of business and technical constraints, since it would not permit users to combine or switch providers whenever convenient in a transparent manner, making interoperability a key requirement.

In this scope, interoperability translates into the ability to run the same programs against different clouds to seamlessly set up infrastructure resources, without the need of handling conditional logic depending on what cloud the program is communicating with, ultimately avoiding vendor lock-in (PETCU, 2011), as illustrated in Figure 7.

Figure 7 - Multiple environments with homogeneous cloud description formats



Source: Created by the author of this document.

In practice, interoperability between heterogeneous systems can be achieved with the adoption of standards (CHEN and DOUMEINGTS, 2003), such as the adoption of post-facto open source software standards.

By providing cutting-edge technology that is freely available to everyone, open source software presents a great opportunity to companies of all sizes. The software is developed by a community of interested organizations and individuals, with a shared understanding that better software and standards are created with the power of collaboration. Although highly encouraged and a good practice that makes the environment sustainable, organizations or individuals are not required to contribute to its development.

OpenStack stablished itself as a post-facto standard for IaaS, being developed since 2010 and becoming the largest open source platform of its type, with a community of more than

80,000 members and hundreds of production clouds spamming the globe. The same software allows the implementation of public and private clouds, consequently allowing hybrid strategies, which helps on consistency of semantic and syntactic aspects when communicating to multiple OpenStack clouds, favoring interoperability.

For the aforementioned facts, some studies consider that post-facto solutions are able to mitigate the risk of vendor lock-in by providing interoperable cloud environments of the same platform, even explicitly citing OpenStack (TOOSI, CALHEIROS and BUYYA, 2014) (GARCÍA, CASTILLO and FERNÁNDEZ, 2016). In OpenStack, however, interoperability is a challenge even between its installations, constituting lock-in as an intra-platform issue due the high flexibility and complexity of the supported functionalities.

1.3 OpenStack Initiatives

A few version releases after its launching at the beginning of the decade, the OpenStack community realized that, by allowing such a flexibility and configurability of deployments, the software was presenting signs of lock-in. Currently, there are three community initiatives to address that: the RefStack program, the guidelines for API changes and the OpenStack SDK client library.

1.3.1 RefStack

RefStack (REFSTACK TEAM, 2018) is a standardization initiative that aims to address lock-in by defining standards for the configuration of OpenStack clouds at API level. With its certification program, it guarantees that deployments interoperate with each other when provisioning infrastructure. The project comprises a toolset for interoperability testing and a database backed website supporting collection and publication of community test results.

As a standard, it positions itself in a conservative manner, enumerating what configurations are acceptable to allow interoperability. However, since this initiative started when many deployments existed already, it presents challenges when reaching consensus on what the certification allows, because defining recommendations and expectations on deployments is a complex problem, especially after the freedom of configuration and feature flexibility were prerequisites in the design phase. Sometimes a consensus would mean some providers acknowledging to update configuration decisions to keep certified by the program,

while giving up on unique use cases. Consequently, there are lacunae in the interoperability definition by RefStack.

1.3.2 Guidelines to API Changes

In development aspects, OpenStack defines guidelines for updating its REST APIs (OPENSTACK API SPECIAL INTEREST GROUP, 2018), including but not limited to:

- The changes must not violate compatibility nor stability, i.e. the same client code works in different clouds without changes and the same client code keep working in a same cloud, upon upgrade;
- Changes in resources and request and response bodies and headers are not allowed without signaling a version boundary, i.e. the service version needs to be updated;
- If functionality is going to be removed its removal should follow standard OpenStack deprecation procedures.

As RefStack, such guidelines only help ensuring API level interoperability, since API changes within major versions are backward compatible. In practice, this means that providers within the same major version of an API will be providing common API contracts, hence allowing user to call the same operation seamlessly in multiple clouds. Similarly, it applies to a single cloud upon service upgrade. To illustrate the versioning in OpenStack with the pattern *major*.*minor*, suppose versions 2.0, 2.1 and 3.0 of a service; in this case, 2.0 and 2.1 have the same major version and therefore are compatible, while version 3.0, not necessarily.

1.3.3 OpenStack SDK

This initiative also started after many production deployments existed, but instead of aiming at standardizing the heterogeneity, it acknowledges that the flexibility and configurability of the project allows multiple valid deployments and then provide an abstraction on top of that to hide differences. In fact, it emerged as a solution for communicating with the multiple clouds providing donated resources to create servers that are used to run functional tests against the code changes proposed to the numerous OpenStack projects.

OpenStack SDK is a client library for users that acts as a broker, exposing features in an opinionated manner via its own APIs, data formats and preferred workflows to provide a

consistent and transparent user experience when communicating to multiple clouds. It supports multiple services for the same functionality, different versions of services and complex workflows. Since this approach presents the most realistic and effective approach to interoperability in OpenStack, this investigation focuses on the study and collaboration to its implementation.

1.4 Problems

OpenStack SDK has proven to work in many different clouds worldwide, especially because it is used in the Continuous Integration system that creates thousands of servers every day in donated cloud resources to run functional tests against the changes proposed to its numerous projects. However, it has never been validated in detail against arbitrary clouds. Therefore, validating the broker library is the first business problem of this investigation.

Moreover, since it has never been reported to the academic literature, there is no comparison of its implementation against the start of the art in interoperability, positioning its approach and potentially contributing to it, especially because no study documenting intra-platform interoperability in IaaS exist yet. Such comparison with the state of the art is the second business problem tackled in this study.

With the aforementioned opportunities for collaboration, the research goals of this study are enumerated in Table 1.

Table 1 - Research Goals

Goal #	Description
1	Document how OpenStack SDK implements interoperability between clouds.
2	Validate that OpenStack SDK is capable of communicating with heterogeneous arbitrary clouds.
3	Document the state of the art in terms of IaaS vendor lock-in solutions in the literature.
4	Compare OpenStack SDK to the current state of the art.

Source: Created by the author of this document.

Additionally, as a library made to work with multiple clouds by adding functionalities as needed by its users, it needs some contributions on additional features and applying consistency across provided functionalities.

1.5 General Scope

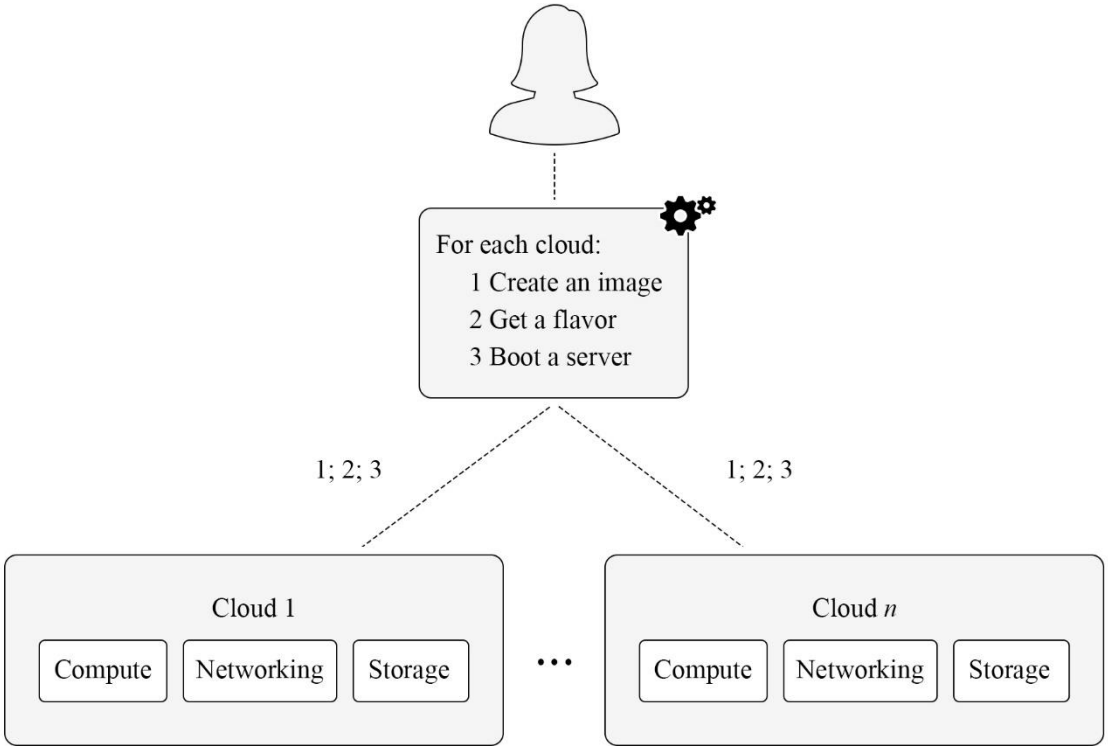
This investigation focuses on the syntactic and semantic characteristics of intra-platform interoperability, targeting semantic, technology and infrastructure management aspects (TOOSI, CALHEIROS and BUYYA, 2014) in OpenStack. Hereafter, when this document refers to the OpenStack SDK communicating to clouds, it is referring to OpenStack clouds.

Syntactic characteristics refer to the systems' capabilities of communicating and exchanging data using the same data formats and communication protocols, while semantic defines that the interpretation of the information exchanged must be meaningful and accurate within the systems.

Semantic aspects focus on message contents and functions calls and responses for consumers requests; technology refers to the agreement on encoding schemes for requests and responses; while infrastructure management is about the coordination and control of resources in multiple clouds.

In summary, it focuses on the service aspects that allow a user's orchestration script to be written once and run whenever necessary against any cloud, as illustrated in Figure 8 for booting a server.

Figure 8 - User running orchestration script on multiple clouds



Source: Created by the author of this document.

In the technology scope, as OpenStack SDK provides management for a wide set of OpenStack resources, which would result in a very extensive analysis if every resource was considered, the scope is restrict to the core resources allowing the granular management of a server and its configuration, the capability of attaching additional storage to servers, storing arbitrary objects in the cloud and creating server snapshots.

1.6 General Approach

This document comprehends the analysis, documentation and validation of OpenStack SDK, followed by a systematic literature review and analysis of OpenStack interoperability.

To document the library, for the period of the master's degree, the author dived into the source code and became part of the developer community, having numerous discussions with other developers to understand not just the technical aspects, but the history and motivations of the project.

For the validation of the library, functional tests were designed, implemented and run against several clouds. The results were collected, analyzed and discussed. All the processes and code are fully documented allowing the reproducibility of the experiments. This addresses the first business problem.

The systematic literature review had its protocol stablished and fully documented, searching for papers in multiple electronic databases, such as IEEEXplore and ACM Digital Library. The papers were filtered, analyzed and the state of the art in IaaS interoperability was documented in detail.

Lastly, a throughout comparison of the literature review outcomes with the OpenStack interoperability positions the solution in the academy, highlighting findings that had never been reported before, therefore addressing the second business problem.

1.7 Involvement in OpenStack

The author of this document is an OpenStack contributor since late 2013 who has mainly contributed and helped leading the development of Keystone, the Identity service of OpenStack. With daily efforts to create, review and maintain code, he has noticed the importance of designing code that allows interoperable deployments.

Effectively making OpenStack interoperable, the author has made significant contributions with dozens of code changes to OpenStack SDK throughout the period of this master's degree, including bug fixes, documentation improvements and development of new functionalities.

Additionally, he has been involved in community efforts to help newcomers onboarding OpenStack projects as a mentor for Outreachy (OUTREACHY, 2018) in 2016 and 2017, and as a coordinator for that program in OpenStack since late 2018. Outreachy provides paid internships for supporting diversity, focusing on people facing under-representation, systemic bias, or discrimination in the technology industry.

1.8 Document Structure

This document is organized as follows: Chapter 2 introduces concepts and explanations that are prerequisite to fully comprehend the investigation and analysis in this dissertation. The main body of this document has 4 chapters, with each addressing one research goal, as in Table 2: Chapter 3 explains in details the OpenStack SDK approach to interoperability; Chapter 4 performs a comprehensive validation of the library against one testing and five production clouds worldwide; Chapter 5 presents the results of a systematic literature review that highlights the state of the art of IaaS vendor lock-in solutions in the literature; while Chapter 6 promotes a deep discussion around the library implementation and the state of the art. Lastly, Chapter 7 reviews the work presented in this dissertation, highlighting the main findings and contributions to both industry and academy.

Table 2 - Chapters and research goals mapping

Chapter #	Research Goal Addressed
3	1 – Document how OpenStack SDK implements interoperability between clouds.
4	2 – Validate that OpenStack SDK is capable of communicating with heterogeneous arbitrary clouds.
5	3 – Document the state of the art in terms of IaaS vendor lock-in solutions in the literature.
6	4 – Compare OpenStack SDK to the current state of the art.

Source: Created by the author of this document.

2 THEORETICAL BACKGROUND

This chapter presents the concepts and definitions necessary for the full comprehension of the dissertation. Section 2.1 provides a short overview of cloud computing; Section 2.2 presents definitions and details about this concept; lastly, Section 2.3 dives into concepts of the OpenStack technology and community.

2.1 Cloud Computing

Cloud computing is a business model for enabling on-demand self-service, ubiquitous and convenient network access to a configurable pool of computing resources that can be elastically provisioned and released, with measured service and minimal management effort or service provider interaction. It presents three service models and four deployment approaches.

2.1.1 Service models

The resources managed by cloud computing are offered as a service to consumers, being infrastructure, platform or software, with varying levels of abstractions and responsibilities (MELL and GRANCE, 2011):

1. Infrastructure as a Service (IaaS): represents the plainest access to computational resources, where processing, memory, storage and networking resources are configured and managed through servers just as in any on-premise environment, allowing arbitrary software to be deployed, but without all the complexity of acquiring, maintaining and upgrading physical hardware;
2. Platform as a Service (PaaS): the environment to run applications on is entirely provided by the vendor, including the operating system, programming languages and tools. The consumer does not have any control on the underlying infrastructure, such as processing, memory, storage and networking, controlling only the applications and configuration settings in the hosting environment;
3. Software as a Service (SaaS): the applications are provided as a service to consumers, and available through the internet via thin clients, such as web browsers, or programming interfaces. The consumers do not have any control on the underlying

infrastructure environment used to deploy the application, being able to only configure potential application configuration settings.

Table 3 provides an overview of the user responsibilities in the different cloud models, in contrast with keeping on-premise infrastructure in silos.

Table 3 - End user responsibilities in on-premise and cloud models

Resource	Silo	IaaS	PaaS	SaaS
Application	•	•	•	
Data	•	•	•	
Runtime	•	•		
Middleware	•	•		
Operational System	•	•		
Virtualization	•			
Servers	•			
Storage	•			
Networking	•			
Data Center	•			

Source: Created by the author of this document.

2.1.2 Deployment models

The infrastructure providing the cloud services may be deployed in four different approaches, according to organizational technical and business requirements (MELL and GRANCE, 2011):

1. Private: the infrastructure is exclusive to a single organization's use, translating into the ability of customizing features of the service, such as custom advanced security requirements. It can be on-premise or off-premise, when the infrastructure is on the customer's site or in a third-party site, respectively. An example is a company that dedicates hardware for orchestration and consequent use by the internal developer team via the company's intranet;
2. Public: the infrastructure is provisioned for use by any consumer, be it an organization or an individual, sharing the pool of resources in a multitenant manner. An example is the Catalyst cloud (Catalyst Cloud, 2018) in New Zealand;

3. Hybrid: the total infrastructure is a composition of the public and private models, in a balance that fits the organization's needs. For example, a company may run OpenStack on-premises for application with real-time requirements, while putting the regular workflow on a public cloud;
4. Community: the cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared mission, security requirements, policy or compliance considerations. An example is the SWITCHengines cloud (SWITCHengines Cloud, 2018), which provides resources to researchers, lecturers and IT-services of universities and related institutions in Switzerland.

2.2 Interoperability

In economics, vendor lock-in is a situation where a customer becomes dependent on a vendor's product or service and cannot move to another vendor without considerable cost and technical effort. In the computer science context, it can be avoided by adopting an interoperability strategy that allows different heterogeneous systems interact with each other (TOOSI, CALHEIROS and BUYYA, 2014). In the cloud computing scenario, it is the ability of clouds from multiple vendors to operate with each other.

2.2.1 Motivation

Rises in prices, decreases in availability, and even the cloud provider's bankruptcy and consequent loss of access to data stored on the provider are drawbacks of using single provider. Furthermore, interoperability and portability are important not only for protection of the user investments but also for the realization of computing as a utility like electricity and telephony (TOOSI, CALHEIROS and BUYYA, 2014).

In addition to the technical aspects, cloud adoption impacts organizational economic, process, performance, and market flexibilities. While the benefits are clear to businesses, when decision makers perform a throughout analysis of the technology, vendor lock-in as security often appear as the main concerns impacting adoption (LAL, 2016). A problem, however, is that some customers present lack of awareness of proprietary standards when procuring services from vendors, obscuring the potential negative effects of the vendor lock-in problems on organizations (OPARA-MARTINS, SAHANDI and TIAN, 2016). Hence, the need to continue

investigating the problem and finding effective solutions to mitigate vendor lock-in grows as more companies shift towards cloud and more vendors with proprietary solutions appear in the market landscape.

2.2.2 Benefits

Besides avoiding vendor lock-in, there are other benefits for end users to design, develop and maintain interoperable systems (TOOSI, CALHEIROS and BUYYA, 2014):

- Scalability and Wider Resource Availability: independency of vendor means relying on the scalability potential of multiple deployments, promoting higher distributed resource availability when unexpected loads overburden a single, typically small cloud provider, leading to unreliable and interrupted services;
- Availability and Disaster Recovery: clouds are expected to provide high availability; however, failures are inevitable and developing a disaster recovery plan becomes crucial, such as creating deployment scripts that can easily deploy stateless applications seamlessly across multiple clouds, whereas the databases would be, by design, distributed and replicated;
- Geographic Distribution and Low-Latency Access: in a situation where applications are global and require fast response time, it is unlikely that a single vendor will offer the best cloud solution in all the regions of the globe with enough granularity;
- Legal Issues and Meeting Regulations: some applications might have specific restrictions about the legal boundaries of resources, such as geographic location. Using multiple deployments seamlessly reduces the burden of maintaining applications.

2.2.3 Types

Cloud interoperability can be classified according to several criteria (PETCU, 2011):

- **Agreement Level:** syntactic and semantic. The former refers to the systems' capabilities of communicating and exchanging data using the same data formats and communication protocols. The latter defines that the interpretation of the information exchanged must be meaningful and accurate within the interoperable systems;

- **Adoption Level:** by design or post-facto, depending on whether services adhere to common open standards or to proprietary standards due to the absolute market dominance of a particular product, respectively;
- **Deployment Level:** horizontal and vertical. The former specifies that interoperable systems are at the same delivery model, IaaS in this case. In contrast with that, vertical would mix service models, such as infrastructure and platform.

2.2.4 Targets

Orthogonally, it can be classified according to its target (PETCU, 2011):

- **Semantic Level:** function calls and responses, along with message contents;
- **Infrastructure Management:** infrastructure management is about the coordination and control of resources in multiple clouds;
- **Technology Level:** technology refers to the agreement on a particular encoding scheme for requests and responses.

Although other targets exist, such as business level that aims to align business strategies, regulations and mode of use of services, it is out of the scope of this investigation as documented in the Section 1.5.

2.2.5 Interoperability vs Portability

Interoperability and portability are closely related terms, and the former is often misused to encapsulate the latter. While the former refers to the ability of using data and services across multiple cloud providers with a unified management interface, the latter apply directly to the application data and system.

Data portability is the ability of copying data objects into and out of a cloud, allowing users to migrate application data whenever convenient between providers. System portability, in its turn, allows the migration of servers or machine images (NIST ROADMAP WORKING GROUP, 2013). As an example of standard for system portability, Open Virtualization Format (DISTRIBUTED MANAGEMENT TASK FORCE, 2018) provides a platform independent, efficient, open and extensible packaging and distribution format that facilitates the mobility of virtual machines, providing customers with platform independence.

2.3 OpenStack

OpenStack is a software that controls large pools of compute, storage, and networking resources throughout a datacenter, allowing operators to provide end users with flexible, scalable and self-service infrastructure.

Hundreds of organizations, including service providers, enterprises, government agencies and academic institutions, run on OpenStack every day, reducing costs while avoiding vendor lock-in and increasing agility on computational infrastructure provisioning, which ultimately results in moving faster.

The same software stack offers the possibility of creating public or private and, consequently, hybrid deployments. The last is specifically interesting because it necessarily requires communicating to multiple clouds, which is a process that is generally not transparent to end users, as heterogeneous clouds have different deployment configurations that affect how end users communicate with them.

As an open source software, it can be freely modified, installed and redistributed by any organization or individual, provided that the licensing terms of the Apache 2.0 license (Apache License, version 2.0, 2004) are respected. If one needs commercial support, there are multiple companies in the ecosystem offering it, just as in some Linux distributions.

The software is delivered by thousands of developers and users headquartered worldwide who use the principles of open source, open design, open development and open community to collaborate in a thriving and vibrant community, making it the largest platform of its kind.

This section provides details of OpenStack that are necessary for the comprehension of this investigation. Subsection 2.3.1 explains the history of the project, as in the OpenStack Introduction Documentation (Introduction: A Bit of OpenStack History, 2018). The mission of OpenStack is shown in the Subsection 2.3.2, highlighting its importance on providing interoperability between its deployments. Subsection 2.3.3 presents the principles surrounding the software and its developing community. The design allowing to support multiple tenants or clients in a single environment is documented in Subsection 2.3.4, while Subsection 2.3.5 provides explanation on the authentication and token validation process. Aligned to that, Subsection 2.3.6 details how the access control enforces authenticated users only have access to resources and operations intended to them, along with the upper limit of resources they can create, as explained in Subsection 2.3.7. The main architectural decision of splitting the functionalities and responsibilities in multiple services is presented by Subsection 2.3.8. The

Subsection 2.3.9 shows how user and inter service communication occurs. The Subsection 2.3.10 documents what services exist in OpenStack and what they do. Subsection 2.3.11 defines what services are in the scope of this investigation, aligned with the scope specified in the Section 1.5. The Subsections 2.3.12 to 2.3.16 further detail the resources in the OpenStack scope, while 2.3.17 explains the dependency between them. Lastly, Subsection 2.3.18 enters the development process and explains what the community does to guarantee the high quality of the software.

2.3.1 History

In 2010, Rackspace (Rackspace Website, 2018) wanted to rewrite the infrastructure code running its cloud servers offering, considering to open source the code. At the same time, NASA had Nebula, a project to provide an easily quantifiable and improved alternative to building additional expensive data centers and to provide an easier way for NASA scientists and researchers to share large, complex data sets with external partners and the public (Nebula Cloud Computing Platform, 2012). NASA published the beta code for Nova, a Python cloud computing fabric controller, allowing both efforts to converge and form OpenStack.

OpenStack then started its life, exposing its first project as Nova version 2, and between code refactoring and new functionalities, new projects emerged. For example, the Neutron service started as networking capabilities concentrated in a submodule named Nova-Network inside the Nova project, which was replaced by another project named Quantum. Afterwards, that project had its name updated to Neutron, since Quantum presented a trademark conflict with a manufacturer of tape-based data backup systems. To match the Nova-Network version at that time, Neutron had 2 as its first version. Another example is Keystone, the Identity service, which was split into a separate service and started in version 2, as did Nova itself when it became OpenStack.

With time, OpenStack had multiple organizations and individuals getting involved and proposing numerous features and new projects, generating concerns about the size and what it really was in terms of open source project. That led to new project ideas and contributors being artificially excluded from the OpenStack Community.

In late 2014, a project structure reform was introduced, known as the *big tent*. It moved the project to a community-centric definition of what OpenStack is, allowing any team that is following the OpenStack principles, using its development model and that has a scope compatible with its mission to become an OpenStack project. As of 2018, there are more than

50 official projects, providing a range of core functionalities, such as providing networking capabilities to virtual machines, to aggregated functionalities, such as integrating with containers and configuration management systems.

2.3.2 Mission

The OpenStack mission is shown in Figure 9, as defined and approved by the OpenStack community representatives.

Figure 9 - OpenStack Mission Statement

> To produce a ubiquitous Open Source Cloud Computing platform that is easy to use, simple to implement, interoperable between deployments, works well at all scales, and meets the needs of users and operators of both public and private clouds.

Source: OpenStack Wiki (OpenStack Wiki Main Page, 2018) and Mission Amendment (OpenStack Mission Amendment, 2016).

A key attribute of the mission statement that is particularly interesting for this investigation is interoperability. In OpenStack, achieving interoperability between deployments ensures end users are not locked in by proprietary infrastructure solutions, at the same time it allows excellent user experience by communicating with multiple deployments, private or public, in a consistent manner.

2.3.3 Core Values

To fulfill its mission, OpenStack define a set of core values that are known as *The Four Opens*, serving as guidance to inform and shape decisions, forming the bedrock upon which its community and software are built.

The Four Opens:

The four core values of OpenStack (The Four Opens, 2018) focus on the software code, design and development process, as well as in the development community:

- Open source: to create truly open source software that is usable and scalable. Truly open source software is not feature or performance limited, and there will be no “Enterprise Edition”;

- Open design: commitment to an open design process, where in every development cycle the community meets in events that are open to anyone to gather requirements and write specifications for the upcoming release. Therefore, the community controls the design process, allowing anyone to influence the software roadmap;
- Open development: all the source code is in public available repositories throughout the entire development process. The code reviews are public, as are the roadmaps;
- Open community: commitment to maintain a healthy and vibrant developer and user community, using a lazy consensus model. All processes are documented, open and transparent. The technical governance of the project is elected by the community. All project meetings are held in public IRC channels and recorded. Additional technical communication is through public mailing lists and is archived.

Principles:

Deriving from *The Four Opens*, there is a comprehensive set of principles adopted by all the community (OpenStack Guiding Principles, 2018). One principle that is of interest in this investigation is *OpenStack Primarily Produces Software*, as defined in Figure 10.

Figure 10 - Principle: OpenStack Primarily Produces Software

> While the software that OpenStack produces has well defined and documented APIs, the primary output of OpenStack is software, not API definitions. We expect people who say they run "OpenStack" to run the software produced by and in the community, rather than alternative implementations of the API.

Source: OpenStack Guiding Principles (OpenStack Guiding Principles, 2018).

This principle clearly states that clouds claiming to run OpenStack are expected to use the APIs implementation by its community, not proprietary or third-party implementations.

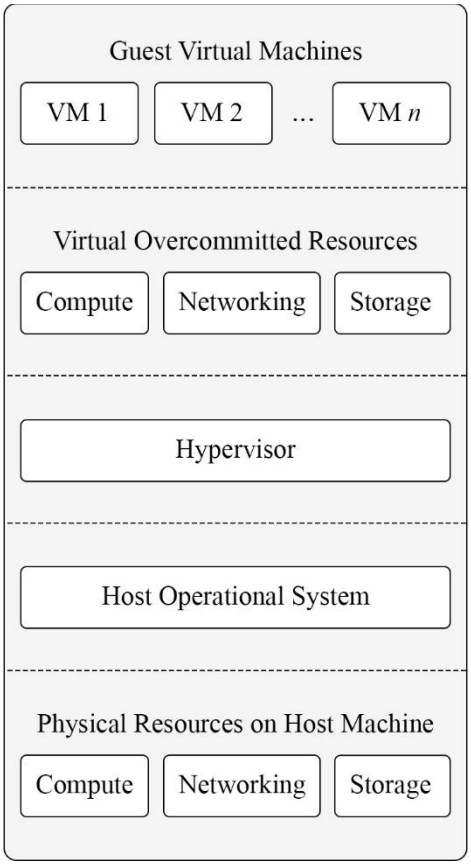
In addition to the core values and principles, all community members agree and must follow the rules described in the OpenStack Code of Conduct (The OpenStack Foundation Community Code of Conduct) in their personal interactions.

2.3.4 Multitenancy

A tenant is a group of users who share common access with specific privileges to the software. Multitenancy refers to a single software serving multiple groups of users, or tenants, in a shared, cost-efficient and secure manner.

At the physical computational infrastructure level, resources are shared, splitting the costs, and security is guaranteed with virtualization technologies. For example, when multiple virtual machines are created on a single physical host, each machine is virtually isolated, only having access to its allocated portion of CPU, memory and storage, as illustrated in Figure 11.

Figure 11 - Multiple virtual machines on a single physical host



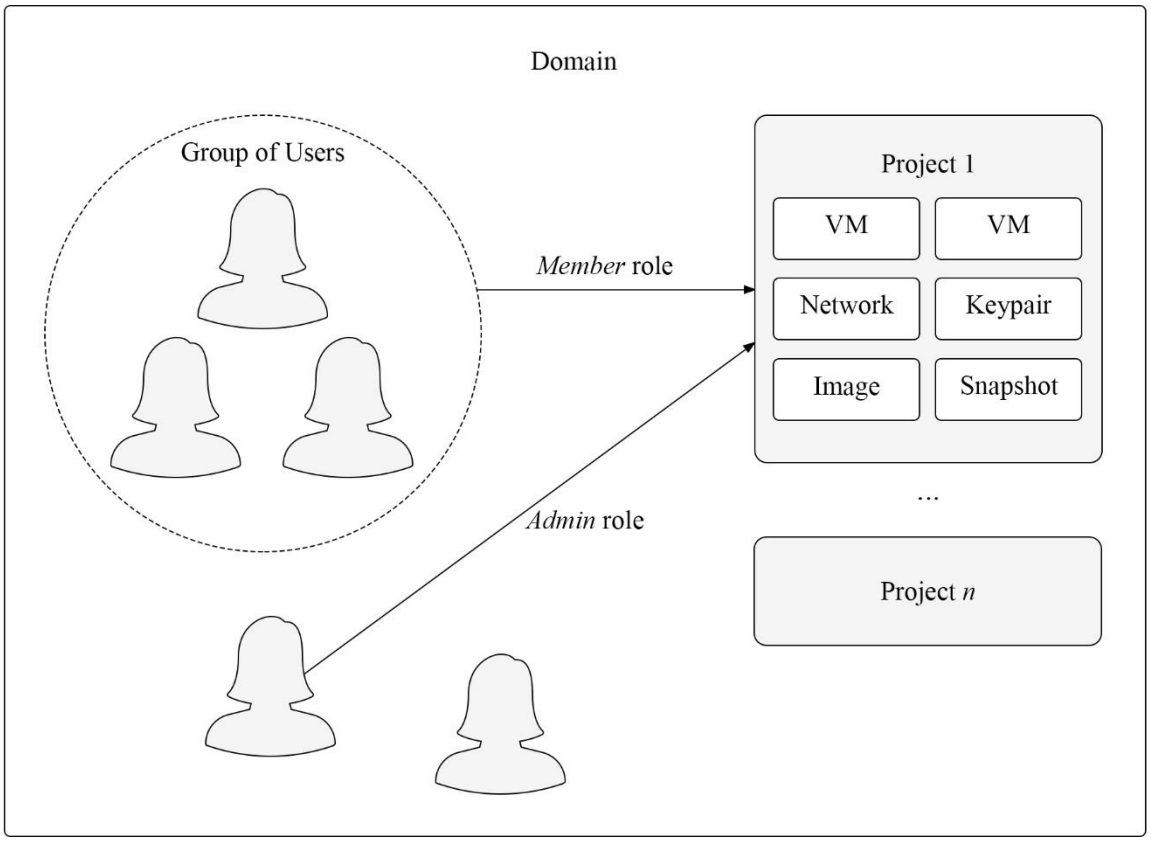
Source: Created by the author of this document.

In an OpenStack environment, however, there is more than servers sharing hardware. Users share the deployment's control plane, managing resources to orchestrate infrastructure for applications, and that process requires isolation as well. For example, users do not want their confidential objects in the Object Store service leaked to other users, allowing them to be read and potentially deleted.

To provide resource isolation, OpenStack implements projects, representing the base unit of ownership of entities. Every resource in the user scope belongs to a project and can only be accessed and managed by users with appropriate access to that project. Examples of resources in the user scope are networks and storage volumes, in contrast with flavors that are common to all users in a cloud and are managed by cloud operators.

A project, in its turn, is owned by a domain, which is a collection of projects, users and user groups that define administrative boundaries for managing resources. For example, a domain might represent a customer organization, owning all its users and projects, which in turn might be that company's departments, with users appropriately distributed and having the expected accesses, as in the example of Figure 12.

Figure 12 - Resource isolation with projects and domains



Source: Created by the author of this document.

The process of verifying appropriate access to resources starts with user authentication, where users receive access tokens containing information necessary to enforce permissions.

2.3.5 Authentication

The first step for users to communicate with the cloud is to authenticate. OpenStack implement authentication mechanisms via plugins, such as password and Time-Based One-Time Password (TOTP), allowing deployers to create custom plugins, if necessary. This

investigation is focusing on the password authentication plugin, as it is typically adopted in all the clouds.

Figure 13 presents an authentication request against Keystone, the Identity service of OpenStack, by calling its API on a bash terminal.

Figure 13 - Password Authentication Request Example

```
 1 $ curl -i \
 2   -H "Content-Type: application/json" \
 3   -d '
 4 { "auth": {
 5     "identity": {
 6       "methods": ["password"],
 7       "password": {
 8         "user": {
 9           "name": "admin",
10           "domain": { "id": "default" },
11           "password": "******"
12         }
13       }
14     },
15     "scope": {
16       "project": {
17         "name": "demo",
18         "domain": { "id": "default" }
19       }
20     }
21   }
22 }' \
23   "http://keystone.example.com:5000/v3/auth/tokens" ; echo
```

Source: Created by the author of this document.

There are two main information in the authentication request: identity and scope, in lines 5 to 14 and 15 to 20, respectively. The former defines that the method to use is *password*, and the information for the password plugin informs the user's owning *domain*, along with *name* and *password*. The latter specifies the scope in which the resources the user wants to manage are, i.e. what project owns the target resources. Tokens with which resources are managed always contain scope information because OpenStack implements the principle of least privilege, where only the minimum level of access is granted, minimizing the impact in the case an access token is compromised. An example response for this request is shown in Figure 14.

The main part of the response is in the *X-Subject-Token* header in line 2, which contains the access token that is used to further communicate with OpenStack to manage resources. Line 6 informs this token was obtained via the *password* plugin, while lines 7 to 14 present information about the authenticated user and lines 15 to 22 show what project the token is scoped to. Lines 23 to 26 list what roles the user has in that project; this information is necessary for access control enforcement, which is further detailed in the next section. Lines 27 to 37 show the service catalog with information about where the user may find the APIs to manage

resources. Lastly, lines 38 and 39 show when this token was issued and when it will expire, respectively.

Figure 14 - Password Authentication Response Example

```
 1 HTTP/1.1 201 Created
 2 X-Subject-Token: MIIFfQ...
 3
 4 {
 5   "token": {
 6     "methods": ["password"],
 7     "user": {
 8       "domain": {
 9         "id": "default",
10         "name": "Default"
11       },
12       "id": "3ec3164f750146be97f21559ee4d9c51",
13       "name": "admin"
14     },
15     "project": {
16       "domain": {
17         "id": "default",
18         "name": "Default"
19       },
20       "id": "3d4c2c82bd5948f0bcab0cf3a7c9b48c",
21       "name": "demo"
22     },
23     "roles": [{
24       "id": "c703057be878458588961ce9a0ce686b",
25       "name": "admin"
26     }],
27     "catalog": [{
28       "endpoints": [{
29         "url": "http://compute.example.com:8774/v3",
30         "region": "RegionOne",
31         "interface": "public",
32         "id": "87057e3735d4415c97ae231b4841eb1c"
33       }],
34       "type": "compute",
35       "id": "bd7397d2c0e14fb69bae8ff76e112a90",
36       "name": "nova"
37     }],
38     "issued at": "2014-06-10T20:40:14.360822Z"
39     "expires_at": "2014-06-10T21:40:14.360795Z",
40   }
41 }
```

Source: Created by the author of this document.

2.3.6 Access Control

OpenStack uses Role-Based Access Control (RBAC) to perform authorization enforcement for the operations on its resources. Roles are managed and assigned to users on projects in Keystone, the Identity service of OpenStack. These roles then go in the access tokens upon successful authentication and then are checked against the policies whenever an operation is requested, as described in the previous section.

Deployments may define as many roles as necessary to achieve enough granularity. For simplicity, consider two roles: *admin* and *member*, representing administrators and typical

project users, respectively. The rules defining what roles are required for an operation are set by the cloud operators in the cloud configuration files as exemplified in Figure 15.

Figure 15- Access Control Rules Definition Example

```
{
  "compute:create": "role:member and project_id:%(project_id)s",
  "compute:delete": "role:admin and project_id:%(project_id)s",
}
```

Source: Created by the author of this document.

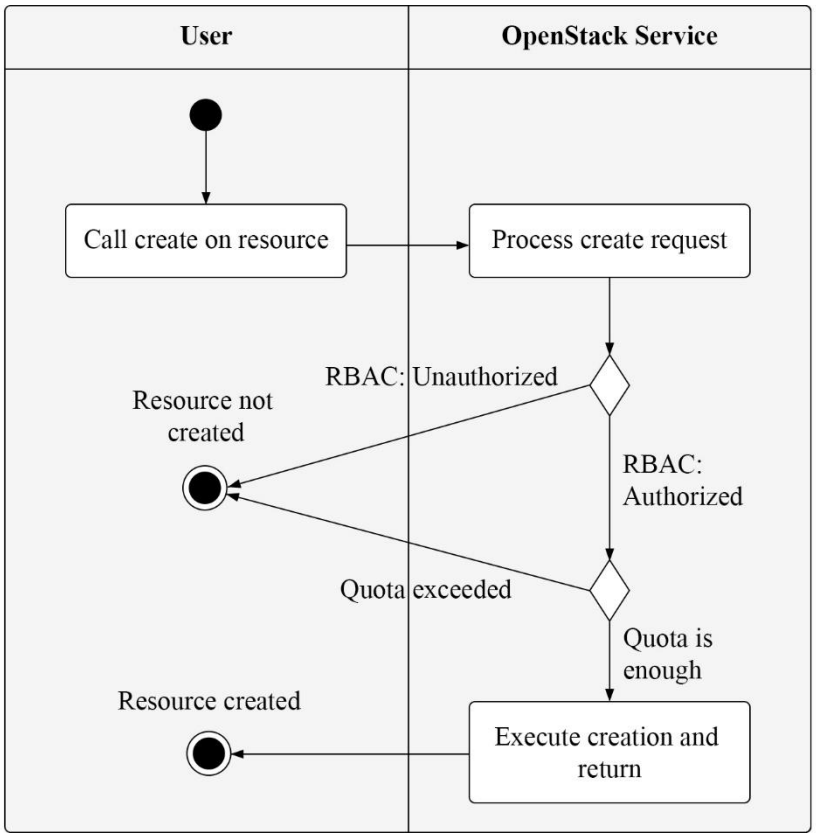
The cloud in this example requires users the *admin* role to perform a create service operation, which is managing the cloud's service catalog. For creating a server, however, it only requires the *member* role and that the server is being created in the project in which the token is authenticated to. When users try to operate on a resource and their token is invalid or they do not have the required permissions, they receive either *HTTP 401 Unauthorized* or *HTTP 403 Forbidden* error issued by the service.

2.3.7 Quotas

Quotas are resource upper limits, allowing providers and users to control and level expectations on the maximum usage of resources in a per-project basis. As an example, the maximum numbers of servers in a project may be set to 10.

When users call create operations to OpenStack, besides verifying users have a valid identity and have permission to perform operations, there is an additional check to ensure there is enough quota for that resource. If there is not, an error is returned, and the operation is not performed. If users need more quota, they typically need to contact the service provider to adjust the quotas for their projects.

Figure 16 - Typical call workflow for create operations



Source: Created by the author of this document.

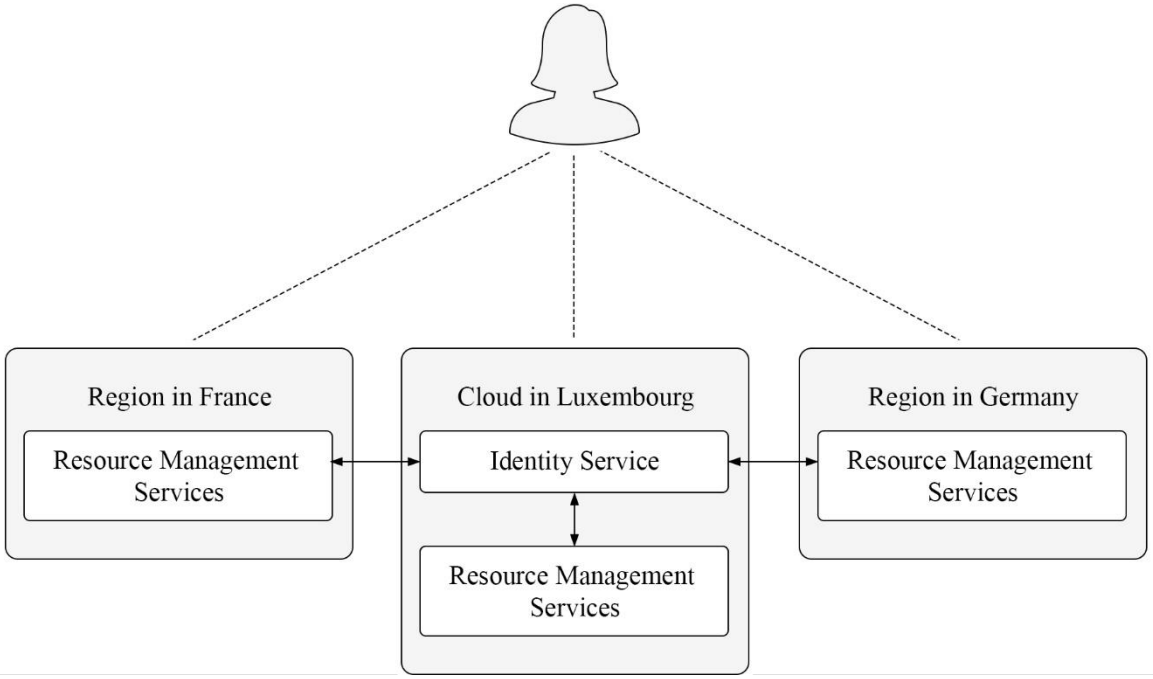
2.3.8 Architecture

OpenStack implements a micro services architecture, where services deploy and scale independently in a standalone manner, allowing cloud providers to manage only the services that fulfill their business strategy. For example, a cloud that only focuses on providing virtual machines to end users might not want to install Swift, the Object Storage service; at the same time, it might need to scale the Identity service by adding more service nodes responding to a load balancer to reduce the response time when generating tokens.

A common use case to cloud providers is to support multiple physically distributed regions with cloud services, while keeping the user and billing management in a central place. This can be accomplished by two approaches: scaling a central deployment to support all the usage; or having multiple deployments that share user information. The latter is a better solution in terms of availability and network performance because users will have the option to use the nearest deployments, reducing latency, while having other regions available for complex use cases such as disaster recovery. This approach is fully supported by OpenStack with the concept of regions, which are discrete OpenStack environments with dedicated services that typically

shares only the Identity service with other regions. Figure 17 illustrates a cloud provider with two additional regions in different countries.

Figure 17 - OpenStack cloud in Luxembourg with regions in France and Germany



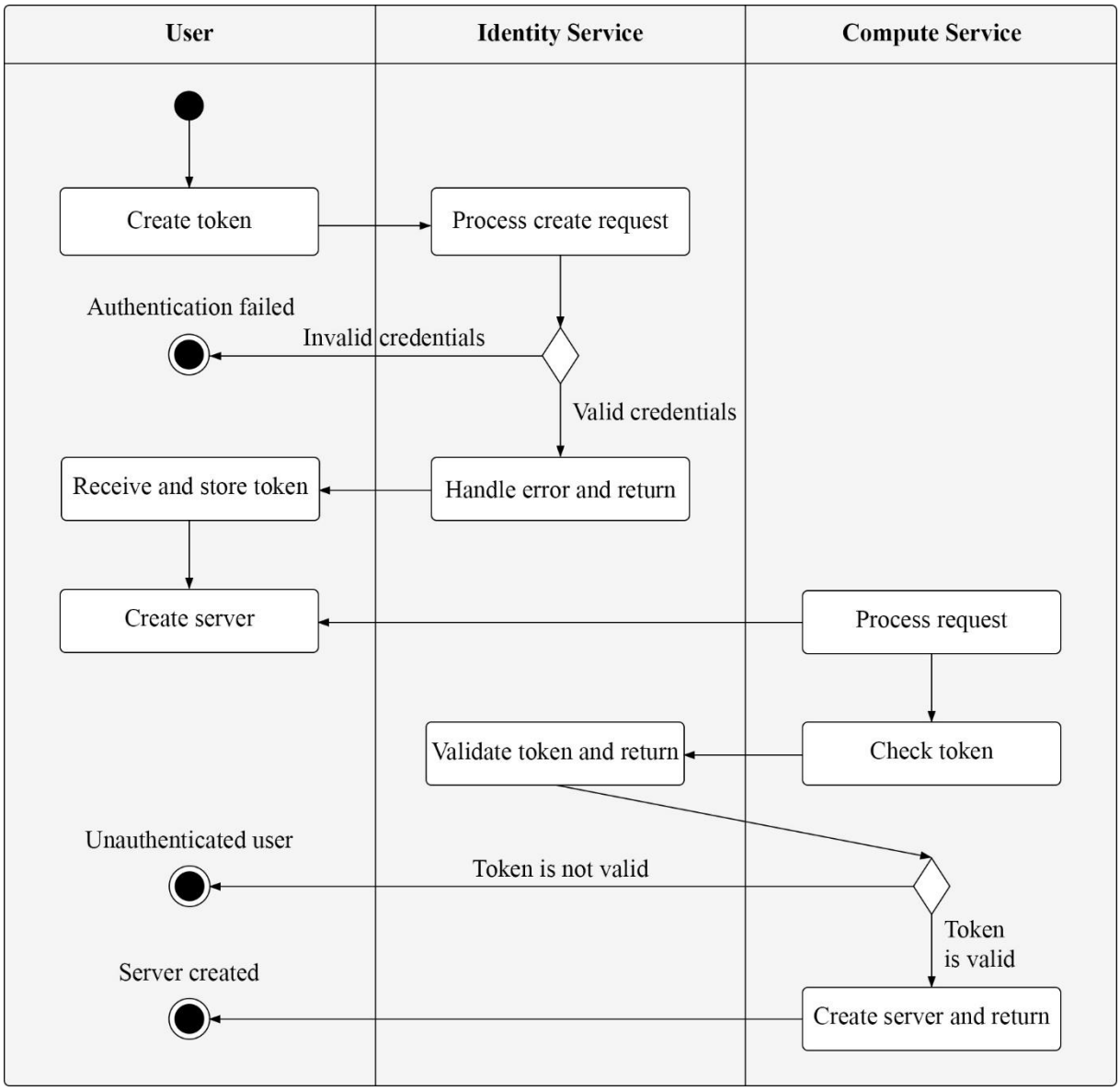
Source: Created by the author of this document.

There are several strategies on where to place the shared identity service, for example, having service nodes in both regions, communicating to a distributed database that is synchronized over a dedicated network link.

2.3.9 Communication

All OpenStack services expose resources and functionalities through Representational State Transfer (REST) APIs. Therefore, as users must authenticate prior to communicating with the appropriate services, the first call is to the Identity service's REST APIs, followed by calls to other services' APIs, as illustrated in the diagram of Figure 18.

Figure 18 - Service communication workflow via REST APIs



Source: Created by the author of this document.

While most of the services implement synchronous APIs, i.e. only return to the user a response when the operation is fully executed, there are some which adopt the asynchronous strategy, i.e. the service returns to the user acknowledging the request was received and then proceed to perform the actual operation. For example, creating servers and images return a *HTTP 202 Accepted*; these resources have a *status* attribute that is set to a temporary value, such as *building*, and switch to another, such as *error* or *active*, after the server processing is completed. In asynchronous calls, users need to keep querying the services to observe the change of the status attribute.

2.3.10 Services

With more than 50 official projects by the time of this investigation, OpenStack provides a large range of functionalities not only to manage the infrastructure itself, but its entire ecosystem, such as cloud deployment and billing management.

These projects are categorized in two classes: core projects for services implementing essential functionalities for a cloud; and additional, providing further services and tools that fit in the ecosystem depending on cloud provider needs.

Core:

There are 6 core services providing the essential functionalities for provisioning computational infrastructure in a datacenter:

- Keystone (Keystone Documentation, 2018), the Identity service, provides client authentication, service discovery, and distributed multitenant authorization through the management of access tokens, projects, users, roles and service catalog;
- Glance (Glance Documentation, 2018), the Image service, features the upload, discovery and retrieval of data assets, i.e. metadata and server image, which is a collection of files for a specific operating system;
- Neutron (Neutron Documentation, 2018), the Networking service, offers network connectivity as a service between interface devices via the management of networks, subnetworks, routers;
- Cinder (Cinder Documentation, 2018), the Block Storage service, manages volumes, that are disk-based data storage that are attached to servers;
- Nova (Nova Documentation, 2018), the Compute service, provisions compute instances. It supports the creation of virtual machines and bare metal servers and has limited support for system containers. It uses other services to perform operations, including Keystone access tokens to communicate with APIs, Glance to retrieve the operational system images to build servers with, and Neutron to determine the networking configuration of the servers;
- Swift (Swift Documentation, 2018), the Object Storage service, offers a highly available, distributed and eventually consistent blob store, allowing the storage of large pools of data in a data efficient, safe and inexpensive manner.

Additional:

The list of further projects is presented at APPENDIX A – ADDITIONAL OPENSTACK PROJECTS, including but not limited to projects implementing web frontend, workload provisioning, application lifecycle, cloud deployment and lifecycle, monitoring, billing and business logic.

2.3.11 Specific Scope

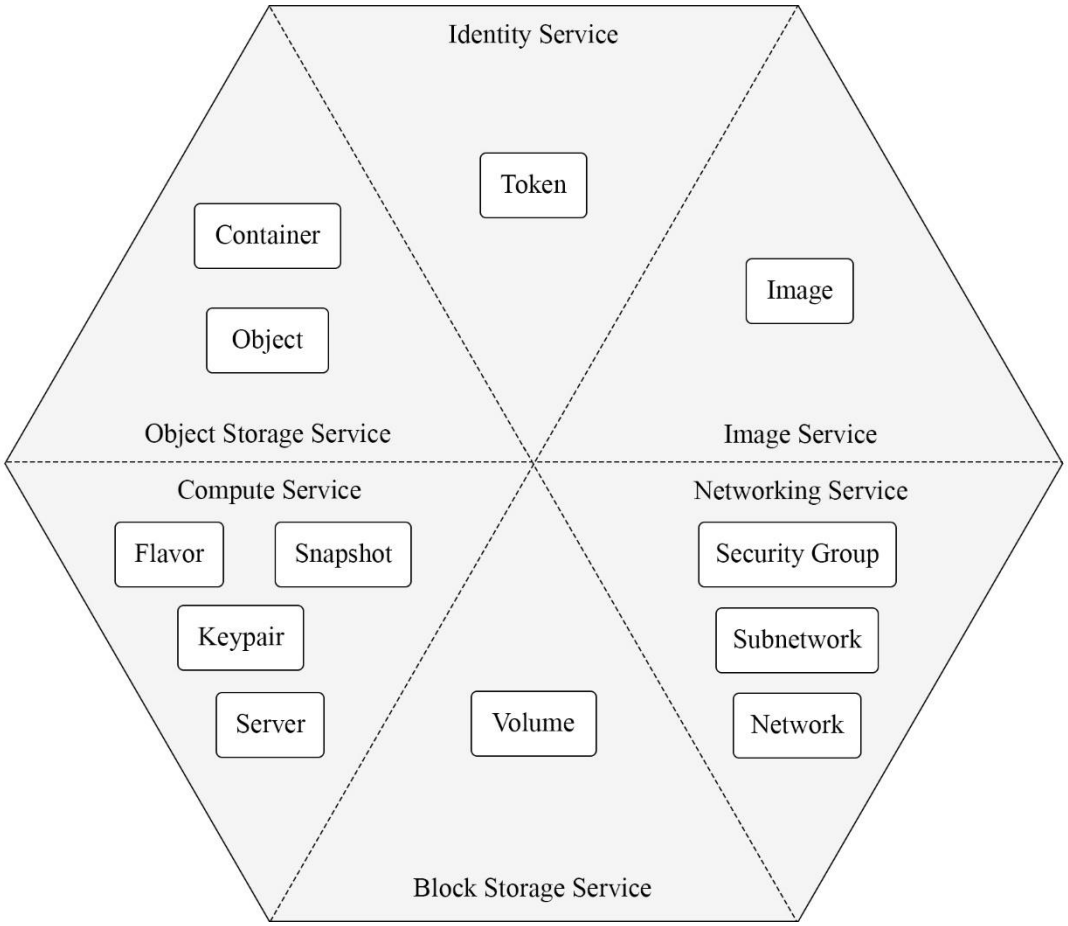
As documented in the previous sections, OpenStack is a complex solution providing a wide range of functionalities on several resources via numerous projects. With the purpose of allowing a deep analysis of the implementation and interoperability characteristics of the operations on resources, the scope needs to be restricted.

The scope of this study focuses on the core resources allowing the granular management of a server and its configuration by end users, added to the capability of attaching additional storage to servers, storing arbitrary objects in the cloud and creating server snapshots. These resources are categorized in six classes, representing six core services documented in the previous section: Identity, Image, Networking, Block Storage, Compute and Object Storage, as illustrated in Figure 19.

Tokens are the only Identity resources in the scope, since a token is required for whichever operation in the cloud, such as creating a server. The Image resources provide the operational system images for building the servers from, while the Networking resources provide servers with connectivity and control on network traffic. Block Storage resources are volumes, which are storage capacity to be plugged into servers as additional disks. Compute resources are flavors, keypairs and servers. Lastly, the Object Storage resources are containers and objects, enabling the storage of arbitrary objects in the cloud.

The next sections detail the resources of the five categories, providing definitions according to the OpenStack Glossary (OpenStack Glossary, 2018), and enumerating their main attributes and characteristics. The Identity core service is not further documented because its project, user, roles and service catalog resources are not managed by end users and are provided to services transparently via access tokens, as described in the Subsections 2.3.4, 2.3.5 and 2.3.6.

Figure 19 - Scope of this study in OpenStack



Source: Created by the author of this document.

2.3.12 Image Scope

Images are managed by Glance and are the only resources in this group, representing a requirement when creating a server, otherwise a virtual infrastructure allocation would be inaccessible without an operational system interface.

An image is a collection of files for a specific operating system to create or rebuild a server. OpenStack deployments provide pre-built images, but end users can create custom images that will belong to the project they are scoped to. For example, a deployment may come with pre-built Windows Server and Ubuntu images, while a project for the development team might want to upload a custom image that has pre-installed services and environment configurations.

2.3.13 Networking Scope

This class contains Software Defined Network (SDN) resources: network, subnetwork, subnetwork and security group, manageable via Neutron, or Nova-Network in outdated deployments, as explained in the Subsection 2.3.1. These resources are mandatory to create a server, otherwise it would be unreachable remotely. Although routers represent an important function in linking two networks, it is not a mandatory requirement for the server creation workflow, therefore it is not in the scope for this research.

Network:

Represent networks providing connectivity between entities, such as a set of server ports that share network connectivity. Networks can be private, only allowing devices in the network to reach other devices in the same network; or public, where devices have access to the network that is external to the deployment, which typically is the internet. Public networks can be a virtual network that is connected to the physical infrastructure via bridging or switching, or a completely virtual network that is connected to an existing network with external access via routing. Networks can have one or more subnetworks.

Subnetwork:

Subnetworks are logical subdivisions of IP networks, containing information such as the IP version of the packets going across it, whether it has a DHCP server automatically distributing IP addresses to connecting devices or whether it has a gateway.

Multiple subnetworks can belong to a single network. For example, a network might have one subnetwork defining IPv4 addressing while another, IPv6.

Security Group:

A set of network traffic filtering rules defining what traffic that can pass through a virtual interface port in the network. The main attributes of a rule are the IP protocol, i.e. TCP or UDP, the IP range of the origin of the request, the port range to make available and direction of the traffic, i.e. ingress or egress. The management of security groups is in the scope of this research, while the management of the granular filtering rules is not.

2.3.14 Block Storage Scope

Managed by Cinder, volumes are the only resources in this class and although they are not mandatory to create servers, it is recommended whenever additional disk storage is required.

Volume:

Disk-based persistent or ephemeral data storage that are attached to servers becoming usable generally as an Internet Small Computer System Interface (iSCSI) target with a file system. They are not hard requirements to create servers; however, they are useful for application workloads that require additional file system storage, as an alternative to creating servers with large-capacity root disks.

2.3.15 Compute Scope

This class manages the flavor, keypair, private server, public servers and snapshot resources, all managed by Nova, the Compute service.

Flavor:

Alternative term for a compute server type, flavors are resources that define how much of each computational infrastructure component is allocated to a server, i.e. its amount of CPU, RAM (in MB) and Storage (in GB). A flavor is a mandatory argument when creating a server.

Keypair:

Keypairs allow users to register and assign public keys in the platform that are later configured in the servers, allowing them to connect to servers using public-key cryptography that encrypts and decrypts the login information, not prompting the typical username and password when connecting to a server via SSH. Keypairs are not mandatory for server creation but it is extremely recommended to managing the servers via SSH connections.

Private Server:

A server is a computer that provides explicit services to the client software running on its operational system. Private servers are connected to a private network without access to an external network, which typically is the internet. In this case, all the dependencies will be explicitly provided in the create call: image, network, security group, flavor and keypair.

Public Server:

OpenStack does not distinguish servers that can connect to public or private networks only, however for the scope of this study, they are considered separate resources for allowing a better coverage of use cases such as IP address assignment.

A public server is connected to an external network and will only have its requisite resources provided on creation: network, flavor and image. Security group and keypair are not included for simplicity reasons.

Snapshot:

This refers to server snapshot, which is a point-in-time copy of a compute server, serving as back up data that permit the implementation of business continuity strategies or to simply clone servers.

There are two types of snapshots: cold or live. In the former, the server in question will be shut down before the snapshot is taken, in contrast with happen in the latter approach that keeps the server running in the process. As a user, there is no way of discovering a cloud's approach, except when it is advertised in the documentation.

2.3.16 Object Storage Scope

This group contains objects and containers, managed by the Swift service. Although these are not mandatory resources for server creation, almost every cloud provides this functionality as a convenient approach for storing and organizing arbitrary objects.

Container:

A container is a directory for the storage of objects, similar to the concept of a Linux directory but cannot be nested. It is a simple entity exposed via the API, containing basically and name that is set by the user and usage information.

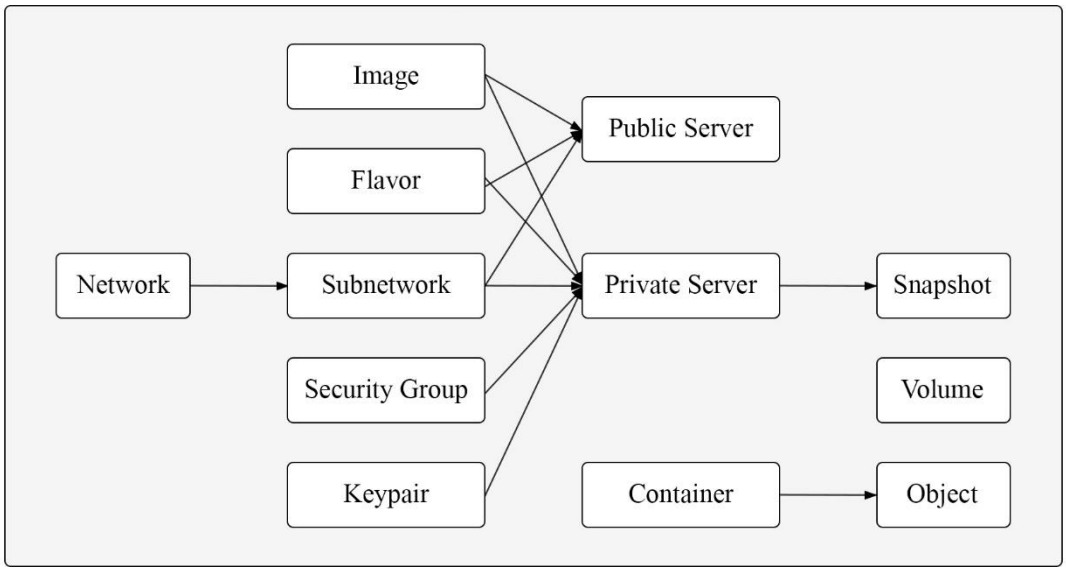
Object:

An object is a Binary Large Object (BLOB) of data that can be in any format. This allows flexibility for users to manage arbitrary documents of up to 5 GB each, by default. The service is designed to scale linearly to multiple petabytes with billions of objects.

2.3.17 Resource Dependency Graph

Figure 20 illustrates the dependency between the resources of this study, as a summary of the specific scope described in the last subsections. Resources that have dependencies in the graph have all its dependencies explicitly provided by the user when being created. For example, the scripts for the validation performed in this study will provide an image, flavor, network and subnetwork, security group and keypair when creating a private server. When creating a public server, only an image, flavor and networking information will be provided.

Figure 20 - Graph of dependency between resources

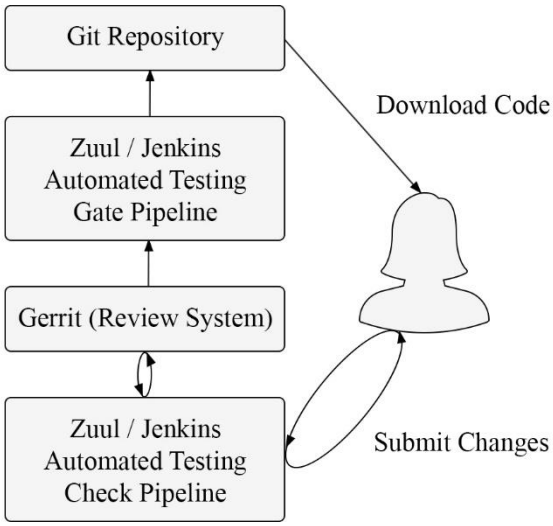


Source: Created by the author of this document.

2.3.18 Continuous Integration

In a complex software with several services communicating to each other to provision computational infrastructure in a datacenter environment, maintaining the quality of the code across services and releases with millions of lines of code is not a simple task that requires a well-designed development process that is backed by a robust Continuous Integration (CI) system, as illustrated in Figure 21.

Figure 21 - Development workflow with Continuous Integration system



Source: (Developer's Guide, 2018).

The OpenStack CI is triggered whenever a change is submitted to Gerrit (OpenStack Gerrit Code Review, 2018), the OpenStack review system. Immediately, the provisioning of testing environments starts, preparing the base for running numerous unit, functional, integration, upgrade, stress, documentation and coding style tests that are implemented in OpenStack projects, varying according to each project's needs. Once the environment is set up, the project-specific tests are run, and the results are reported back to the developers in the review system. When the changes are approved by the OpenStack community, they are submitted to another set of tests in the gate pipeline, which represent the last testing scenarios before the change is finally merged into the official Git code repository.

Environment Setup:

Each test scenario requires a specific environment configuration. For example, running unit tests written in Python 3 requires having Python 3 installed in the server where the tests will execute on. The configurations for testing environments are enumerated in the CI system code, organized per project, per test scenario.

A fresh virtual server is set up for each test scenario, ensuring the environment is clean of garbage that could potentially interfere in the tests output. For that, OpenStack uses infrastructure resources donated by supporting companies that run OpenStack clouds, creating virtual servers at scale.

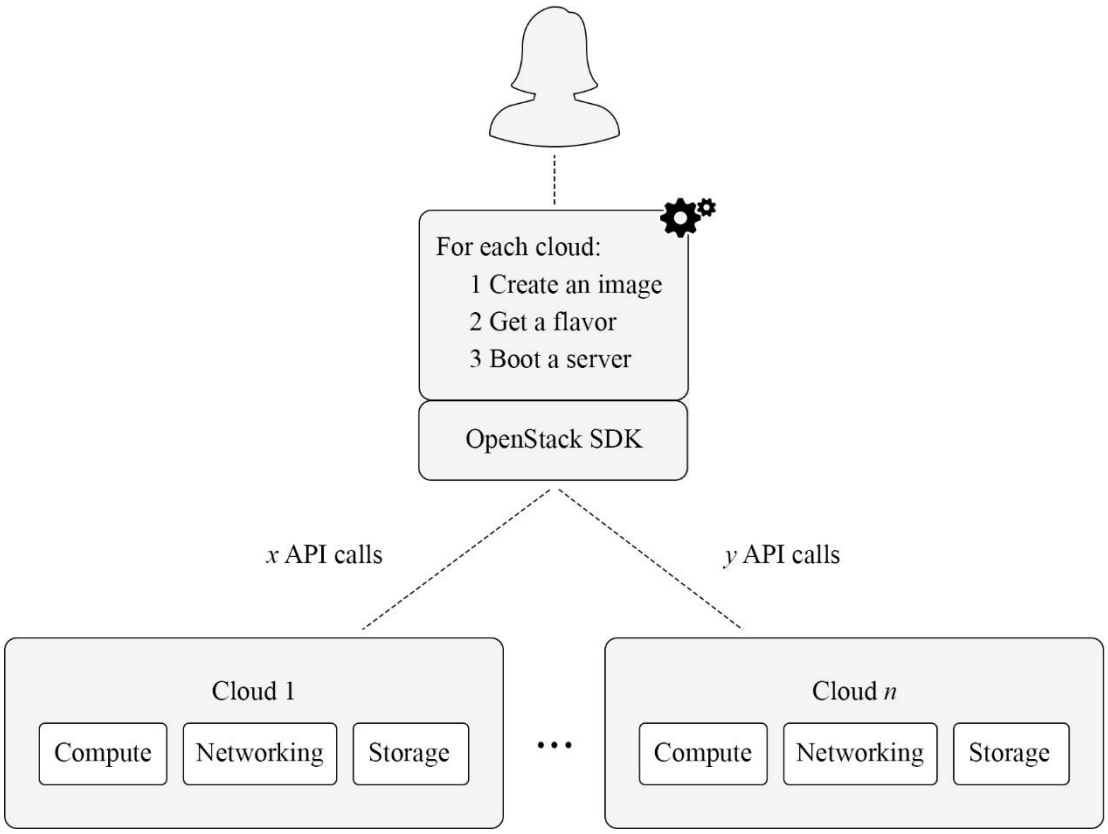
With multiple clouds comes the challenge of communicating transparently with them, easing the maintenance of scripts and, more importantly, avoiding vendor lock-in to ensure the continuity of the CI system whenever existing companies stop donating resources and new companies start doing it. To solve this problem, the OpenStack CI uses the OpenStack SDK client library to communicate with multiple clouds, which abstracts all the vendor differences, offering a transparent experience to the orchestration scripts' developers.

This workflow applies to the OpenStack SDK project testing as well, which plays a key role in the ecosystem, since the adoption of a version of that library with regressions by the CI system could potentially cause the whole CI of OpenStack, which supports dozens of projects, to stop working.

3 OPENSTACK SDK

This chapter documents OpenStack SDK , the Python library acting as a broker between users and clouds, which abstracts all the complexity of cloud-specific service versions and configuration details in a transparent manner for multicloud users, as per Figure 22.

Figure 22 - OpenStack SDK, a broker between users and clouds



Source: Created by the author of this document.

Managing resources using the abstraction layer is as user-friendly as the code shown in Figure 23, which creates servers in clouds at the seven continents of the planet with only eight lines of well-formatted code.

Figure 23 - Example code creating a server in multiple clouds

```
1 import openstack
2
3 for name in ('Asia', 'Africa', 'NorthAmerica', 'SouthAmerica',
4              'Antarctica', 'Europe', 'Australia'):
5   cloud = openstack.connect(name)
6   image = cloud.search_images(filters='[?contains(name, `ubuntu`)]')[0]
7   flavor = self.cloud.search_flavors(filters={'ram': 1024})[0]
8   cloud.create_server('masters', image=image, flavor=flavor, wait=True)
```

Source: Created by the author of this document.

The code simply connects to the each one of the clouds, one at a time, in line 5; then it retrieves an image with *ubuntu* in its name in line 6; in line 7 it retrieves a flavor with 1024 MB of RAM; and in line 8 it triggers the server creation, waiting until it is up and running.

This chapter details how OpenStack SDK gets around all the complexity of managing heterogeneous clouds to provide a transparent experience to its end users, focusing on the resources in the scope of this study, as enumerated in the Subsection 2.3.11. Although technical documentation exists in the OpenStack SDK website (OpenStack SDK Documentation, 2018), this chapter is not only a result of interpreting the existing documentation, but also a summary of numerous discussions between the author of this document with the community of developers to fill gaps where the documentation was not clear or missing, therefore representing contributions from the author.

The Section 3.1 presents the history of this library, along with what was the original motivation to communicate to multiple OpenStack environments. Section 3.2 presents the design of the library's functionalities and communication standards, while Section 3.3 provides an in-depth analysis of the operations on the resources in the scope of this study. The Section 3.4 goes through the testing process that runs whenever a code change is proposed to an OpenStack project, entering in detail for this library's project. The Section 3.5 summarizes the numerous contributions made by the author of this study to the official code repository of this library. Lastly, Section 3.6 mentions further resources that are, by the time of this investigation, supported by the client library.

3.1 History

As a complex software with several services communicating to each other to provision computational infrastructure in a datacenter environment, maintaining the quality of millions of lines of code is not a simple task that requires a Continuous Integration system, as further documented in the Subsection 2.3.18. Such system sets up virtual machines at scale in third-party donated resources that serve as environment to run the project-specific tests.

In the process of communicating to multiple OpenStack clouds to set up the donated virtual machines, the orchestration code needs to get around the heterogeneity of deployments. This abstraction code was initially in *nodepool* (Nodepool Documentation, 2018), the system for managing test node resources, and in the Ansible OpenStack Modules (Ansible OpenStack Cloud Modules, 2018).

At that time, the heterogeneity was managed in several parts of the libraries, resulting in a lot of duplicated code. The refactoring process led to the creation of a new library: Shade (Shade Documentation, 2018), a Python standalone library containing all the logic and features to communicate with multiple clouds seamlessly.

Afterwards, Shade and another project named *os-client-config* (os-client-config Documentation, 2018) were merged into the OpenStack SDK project, resulting in a complete client library for orchestrating OpenStack clouds. Before the merge, the *python-openstacksdk* project already existed, which by then was only a library that exposed the OpenStack APIs to developers in a consistent and predictable manner using Python. *os-client-config* was a library for collecting client configuration in a consistent and comprehensive approach, which introduced a file for expressing named cloud configurations.

This study focuses specifically to the Shade part of the current OpenStack SDK, along with its configuration capabilities inherited from *os-client-config*, i.e. the multicloud abstraction interface layer and the client configuration, respectively, referring to them as the OpenStack SDK library.

3.2 Design

In contrast with the service-oriented design of the services' REST APIs, where a user needs to know what service and version provide a specific resource on a cloud, OpenStack SDK is a resource-oriented client, allowing end users to consume computational infrastructure resources in a simple and transparent manner. For example, consumers can orchestrate security groups without having to know whether it is the Nova or the Neutron service that is managing that resource, as illustrated in Figure 24, where a security group is created, retrieved, updated and then deleted.

Figure 24 - User managing security groups with OpenStack SDK

```
$ python
>>> import openstack
>>> cloud = openstack.connect('devstack')
>>> cloud.create_security_group('masters', description='Custom group').name
'masters'
>>> cloud.get_security_group('masters').description
'Custom group'
>>> cloud.update security group('masters', description='My custom group').description
'My custom group'
>>> cloud.delete_security_group('masters')
True
```

Source: Created by the author of this document.

This library implements resource-oriented functionalities by communicating with the OpenStack services and representing the cloud resources in a consistent manner, as documented in the Subsections 3.2.1 and 3.2.2, respectively. Furthermore, it has its own opinions on how it communicates with the REST APIs, as per the Subsection 3.2.3.

3.2.1 Communication

OpenStack SDK consumes services by communicating to their REST APIs, which are detailed in the Subsection 2.3.9. However, before communicating, the library needs to successfully authenticate the end user and discover what services are in fact running in a cloud, since that can vary significantly between deployments.

After successful authentication, the library retrieves the service catalog and discover the appropriate service URLs to communicate with using the service discovery process, both as fully documented in the APPENDIX B – SERVICE CATALOG AND DISCOVERY. When trying to communicate with a service that is not present in a cloud, it will raise an exception stating that an endpoint URL could not be find for that specific service.

3.2.2 Resource Representation

To provide a seamless experience while using multiple clouds, it is not only necessary to abstract the cloud-specific configurations with a consistent set of functions exposing operations on the resources, such as a create network operation that is a single function in the library that accepts the same consistent set of parameters for all the clouds. In addition to that, the responses returned from the underlying service clouds need standardization, allowing end users to safely rely on the library's resource representation when orchestrating infrastructure.

Furthermore, because of its multicloud design, end users need to be able to identify easily where the resource under management is located. For that purpose, a location attribute is present in all the resources returned by the broker, containing information not only about the cloud and region the resource belongs to, but the scoping information within that environment, detailing what project owns the resource. Further explanation on projects owning resources is present at the Subsection 2.3.4.

3.2.3 Opinions

As an interface that provides abstractions to optimal user experience on multicloud usage, the abstraction layer of OpenStack SDK deviates from the published OpenStack REST APIs by adding its own opinions, adopting a simple and minimalist approach to get orchestration and resource management done. Such approach may not satisfy every consumer, such as advanced users with specific needs.

As an example, the create functions receive the arguments it considers necessary and essential to create a resource, not necessarily exposing all the arguments that are accepted by the underlying services.

If a user has very specific needs that are not satisfied by the abstraction layer in OpenStack SDK, they would need to use the specific service clients that expose the services' functionalities in a more comprehensive manner.

3.3 Implementation

OpenStack SDK is a client-side Python library, which only requires end users to install and use it as a dependency on their environment, without any intervention or action from cloud providers, since it will use the REST APIs in the same manner its consumers would do directly on the cloud services.

The main entry point for the abstraction layer is the class *OpenStackCloud* (OpenStackCloud Code, 2018), which has its implementation detailed in this section.

The Subsection 3.3.1 shows how the library authenticates and scopes in a cloud. Subsection 3.3.2 documents how OpenStack SDK communicates to the services running in the underlying cloud, detailing how it works around heterogeneous service versions in a multicloud setup. Subsection 3.3.3 documents how it deals with access control policies defined in the clouds, while Subsection 3.3.5 details the comprehensive resource data model defining the representation of the library's responses; Subsection 3.3.6 presents the mechanism of normalizing the raw resources as returned by the services into the library's returns, while Subsection 3.3.7 details how OpenStack SDK allows users to call asynchronous APIs in a synchronous strategy. The Subsection 3.3.8 presents the overall behavior of the creation, retrieval, update and delete operations available for all resources, while Subsections 3.3.9 up to 3.3.13 details the peculiarities of each operation implementation for the specific resources, if any. Subsection 3.3.14 summarizes the service versions supported by OpenStack. Lastly,

Subsection 3.3.15 shows additional functions OpenStack SDK implements to help the users manage resources that are not consistently discoverable or operations that involve multiple calls to the services.

3.3.1 Authentication

The first step to communicate with OpenStack services is to authenticate against Keystone, the Identity service, obtaining an access token that is passed along to other services, as documented in the Subsection 2.3.5.

OpenStack SDK introduces a new approach for collecting client configuration in a consistent and comprehensive manner, using the YAML format (YAML Website, 2016) in a file that is sought at */etc/openstack/clouds.yaml*, as the example in Figure 25.

Figure 25 - Example *clouds.yaml* file for a single cloud

```
clouds:
  UFRN:
    auth:
      username: samueldmq
      password: ******
      domain_name: students
      project_name: masters
      auth_url: https://ufrn.example.com
    region_name: Caico
```

Source: Created by the author of this document.

That information would allow the user *samueldmq* to connect to the region *Caico* of the cloud *UFRN*, scoping to the project *masters* of the domain *students* via authentication against the URL *https://ufrn.example.com*.

Since OpenStack SDK is a multicloud library, it is possible to define information for authenticating to multiple environments in the *clouds.yaml* configuration file by appending the cloud-specific content within the *clouds* root tag, as in Figure 26.

Figure 26 - Example *clouds.yaml* for multiple clouds

```
clouds:
  UFRN:
    auth:
      username: samueldmq
      password: ******
      domain_name: students
      project_name: masters
      auth_url: https://ufrn.example.com
    region_name: Caico
  UFCG:
    auth:
      username: samueldmq
      password: ******
      project_name: masters
      auth_url: https://ufcg.example.com
    region_name: CampinaGrande
```

Source: Created by the author of this document.

In addition to allowing the user to use the *UFRN* cloud as in Figure 25, this would allow using the region *CampinaGrande* of the cloud *UFCG*, scoping to the project *masters*. By authenticating to its entry URL: *https://ufcg.example.com*.

This library supports communicating with the versions 2 and 3 of Keystone, the Identity service. The major difference between these versions is that the latter supports multiple domains, each owning a set of projects, while the former does not. Further details about the scoping of cloud resources in projects and domains is presented in the Subsection 2.3.4.

Looking back at Figure 26, notice that the information for the cloud *UFRN* is in the version 3 format, having a *domain_name* attribute in it, whereas *UFCG* does not. As this is the root information that a user provides to the library, OpenStack SDK cannot abstract different versions at that level, therefore the user needs to gather and put all the information a cloud requires to authenticate in the *clouds.yaml* file. However, if the domain or any other required information is missing, suggestive error messages will inform the user, assisting in the one-time setup process.

Lastly, notice that the only URL that needs to be provided by end users is the authentication URL, since the library requires it to communicate to the Identity service for authenticating and consequently discovering and communicating to services on that cloud.

3.3.2 Service Communication

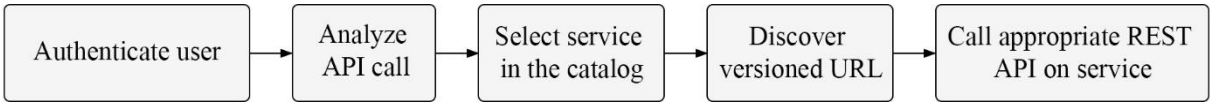
This subsection shows the process of discovering services available in a cloud based on the authentication token and how OpenStack SDK implements support for operations on services that have multiple versions.

Service Discovery:

The process of discovering the services starts with the user token obtained after successful authentication. Thereafter, the library gets access to the service catalog for that token on the cloud, which is an extensive document describing in detail all the services available. With that document, the library runs the service discovery process to find out the URLs to communicate to each service. As the service catalog is mostly static, a caching strategy guarantees this process is not run upon every operation call, avoiding time overhead.

The complete process, from user authentication to calling the service with the versioned URLs is illustrated in Figure 27. For further details, both the service catalog and discovery are documented in APPENDIX B – SERVICE CATALOG AND DISCOVERY.

Figure 27 - Authenticating user, discovering and calling services



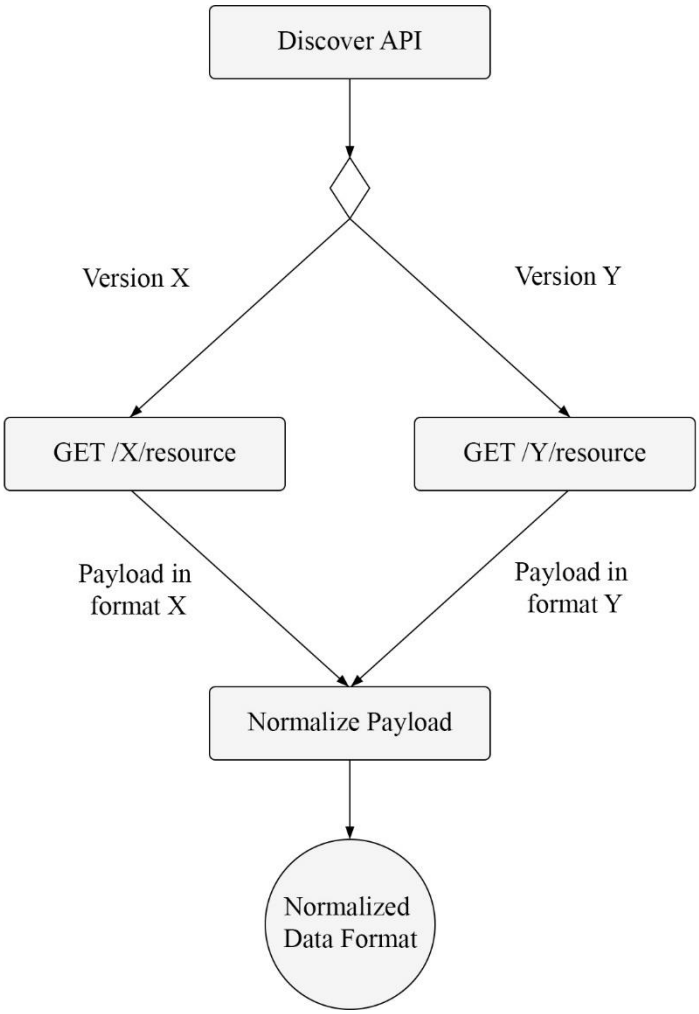
Source: Created by the author of this document.

Multiple Versions:

To support multiple versions of the services, OpenStack SDK implements different code paths depending on what is available in the cloud. As a result, for each operation supporting multiple versions, the library first checks what version is under use and then proceeds with the appropriate code path, as illustrated in Figure 28 with an example for retrieving a resource.

This results in several conditional statements in the code, which is what multicloud users would need to do by themselves in their orchestration scripts.

Figure 28 - Checking service version and proceeding with retrieval call



Source: Created by the author of this document.

3.3.3 Access Control

OpenStack SDK does not define any additional layer of access control in the client side; it respects the rules and personas defined in the cloud, expressed via roles that go in the authentication token of users, as in the section Subsection 2.3.6.

The broker library supports both cloud operator and end user main personas, implementing functions to ease the management of the deployment, such as creating flavors and cloud-wide images; and end user operations, such as creating a network and a server.

When trying to perform an operation in the underlying cloud and a *401 UNAUTHORIZED ERROR* is returned by the service, the library cannot work around and repasses the error to the end user. This is a common event when an end user tries to perform an

operator function, such as create a flavor, as illustrated in Figure 29 showing that the policy, i.e. access control rules, blocked the user from performing such operation.

Figure 29 - End user trying to create a flavor

```
1 $ python
2 >>> import openstack
3 >>> cloud = openstack.connect('devstack')
4 >>> cloud.create_flavor('custom', vcpus=4, ram=2048, disk=512)
5
6 openstack.exceptions.HttpException: HttpException: 403: Client Error for url:
https://example.com:8774/v2.1/145bfa231dae4aae97764121f018a5af/flavors, Policy doesn't allow
os_compute_api:os-flavor-manage to be performed.
```

Source: Created by the author of this document.

When there is an operation that end users are expecting to be able to perform, such as create an image, and are not allowed by the access control permissions in the cloud, the end users need to contact the cloud provider to revise their policies, as that would be considered an error while configuring the deployment.

3.3.4 Quotas

Quotas are mechanism to protect the deployment and the project administrators from overuse or misuse of computational resources. Similar to the approach for access control rules, OpenStack SDK does not define any additional layer for quotas in the client side; it respects the resource limits defined in the projects, as described in the Subsection 2.3.7.

When creating a resource in the cloud and the user quota exceeds, the library will get an exception from the service and repass it to the user, such as in the example Figure 30, where the user tries to create a server with more CPU and RAM than allowed by the quota of the project they are authenticated to.

If a user needs more quota that what has been assigned to the project they are scoped to, they need to contact the cloud administrator to request more quota, since managing quotas is a cloud operator function.

Figure 30 - End user exceeding the quota for servers

```
1 $ python
2 >>> import openstack
3 >>> cloud = openstack.connect('devstack')
4 >>> image = cloud.get_image('cirros-0.3.5-x86_64-disk')
5 >>> flavor = cloud.get_flavor('m1.small')
6 >>> cloud.create_server('masters', flavor=flavor, image=image)
7 openstack.exceptions.HttpException: HttpException: 403: Client Error for url:
https://example.com:8774/v2.1/145bfa231dae4aae97764121f018a5af/servers, Quota exceeded for
cores, ram: Requested 1, 2048, but already used 8, 16384 of 8, 16384 cores, ram
```

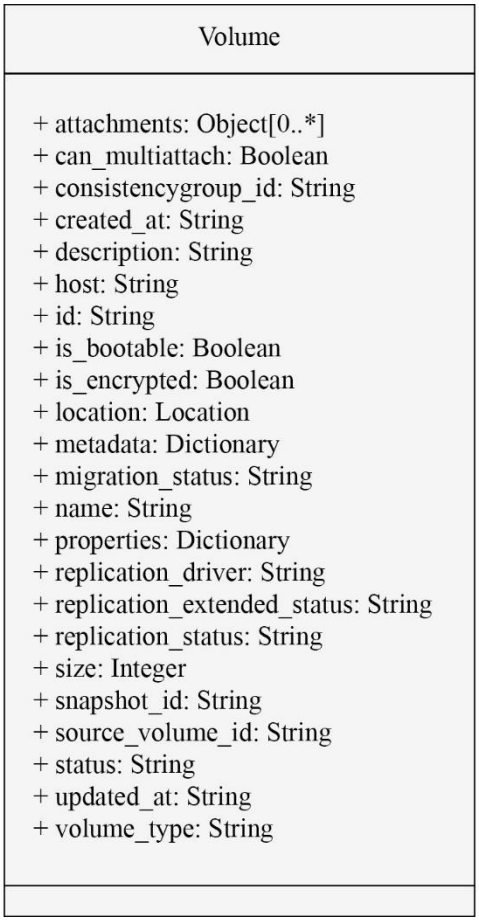
Source: Created by the author of this document.

3.3.5 Data Model

As a resource-oriented library that represents resources consistently in multicloud environments, the OpenStack SDK data model (OpenStack SDK Data Model, 2018) is a key to standardize the resources representation. As an example, all the volumes are as in Figure 31.

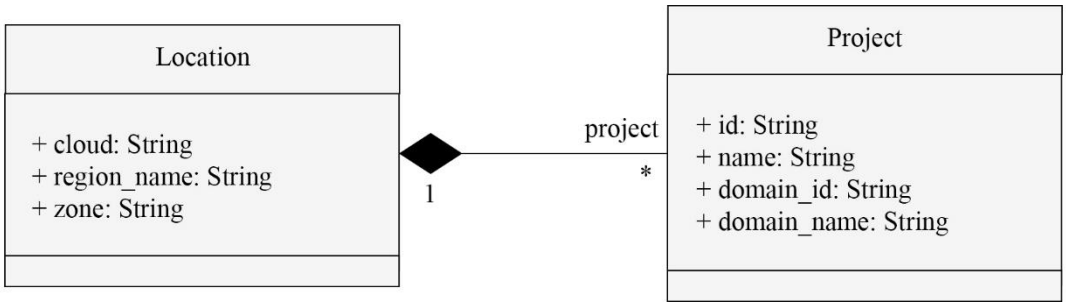
In addition to the attributes returned by the services, the client inserts a *location* attribute in every resource, as in Figure 32, allowing users to read the scope of the resource.

Figure 31 - Data model of volumes



Source: Created by the author of this document.

Figure 32 - The *location* attribute



Source: Created by the author of this document.

All the resources in the context of this investigation have their data model further detailed in the APPENDIX C – OPENSTACK SDK DATA MODEL.

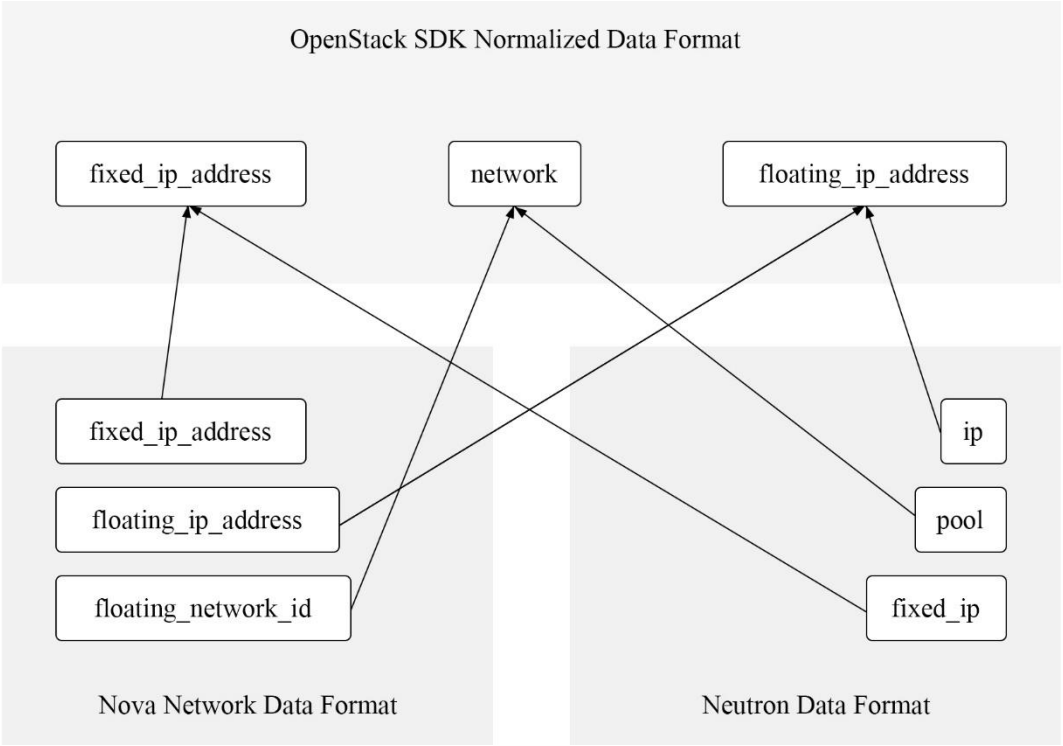
3.3.6 Resource Normalization

Different service versions might return heterogeneous representations of resources: a resource can have one of its attributes renamed or removed in a newer version, while others can be introduced. To return a consistent model, the library adapts the received responses from the services into the data model representation detailed in APPENDIX C – OPENSTACK SDK DATA MODEL, in a process named resource normalization.

Figure 33 shows an example of the normalization of a floating IP, which converges heterogeneous representations originated from different services to a unified resource.

The normalized resources are Munch (Munch Git Code Repository, 2018) objects, presenting a convenient manner to access its attributes via both object attributes and dictionary keys. Figure 34 shows an example in the Python interpreter, where a server object named *masters* is retrieved in line 4 and has its *name* attribute accessed as an attribute in line 5, and as a dictionary key in line 7.

Figure 33 - Normalizing a floating IP resource



Source: Created by the author of this document.

Figure 34 - Munch object of a server resource

```
1 $ python
2 >>> import openstack
3 >>> cloud = openstack.connect('devstack')
4 >>> server = cloud.get_server('masters')
5 >>> server.name
6 'masters'
7 >>> server['name']
8 'masters'
```

Source: Created by the author of this document.

The library's data model does not necessarily incorporate all the attributes returned by the services because it considers some as irrelevant, leaving those attributes without normalization in a dictionary named *properties*, as in Figure 35 where the *power_state* attribute is accessed, outputting the value 1 which means *RUNNING* according to the Nova API Reference documentation (Nova API Reference, 2018).

Figure 35 - Additional attributes in the properties dictionary

```
1 $ python
2 >>> import openstack
3 >>> cloud = openstack.connect('devstack')
4 >>> server = cloud.get_server('masters')
5 >>> server.properties.power_state
6 1
```

Source: Created by the author of this document.

3.3.7 Asynchronous Operations

Most of OpenStack APIs are synchronous, i.e. only respond to the end users once the operation is fully executed. However, a few create and delete APIs implement an asynchronous strategy, returning to the user the resource with a *status* attribute, which is set to a value that informs it is under creation by the service. Afterwards, the user is responsible for querying the service until the resource is, in fact, available. This strategy is adopted for resources that might take a considerable time to be created or deleted, such as servers, volumes and images, as documented in the Subsection 2.3.9.

OpenStack SDK implements a mechanism that allow users to create and delete such resources in a synchronous manner via the *wait* and *timeout* function parameters. When the user provides *wait* with the positive Boolean value, the library creates the resource and enters in a loop until it reaches either the available state or the timeout.

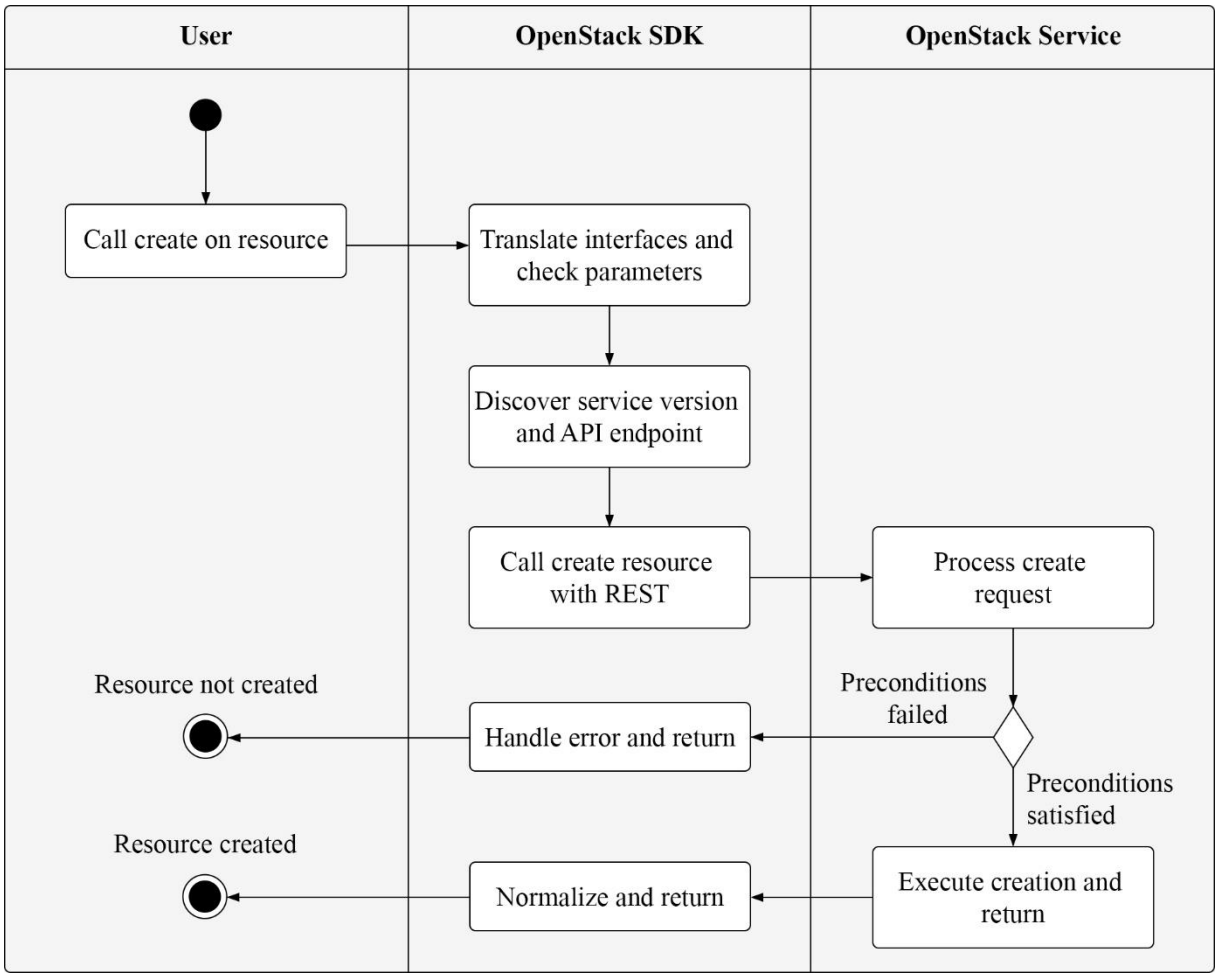
3.3.8 Core Functions

This section presents the overall behavior of the create, get, list, search, update and delete operations available for all resources (OpenStackCloud Code, 2018). The resources returned by the create, get, list, search and update operations are normalized to the data model defined in APPENDIX C – OPENSTACK SDK DATA MODEL, as per Subsection 3.3.6.

Create:

This function receives the resource's attributes as parameters, does any treatment necessary, such as matching the actual attribute names at the service, and finally calls the appropriate service via REST API. Upon getting a response, it normalizes and returns. There are multiple points where exceptions might occur. This process is illustrated in Figure 36.

Figure 36 - The create resource function

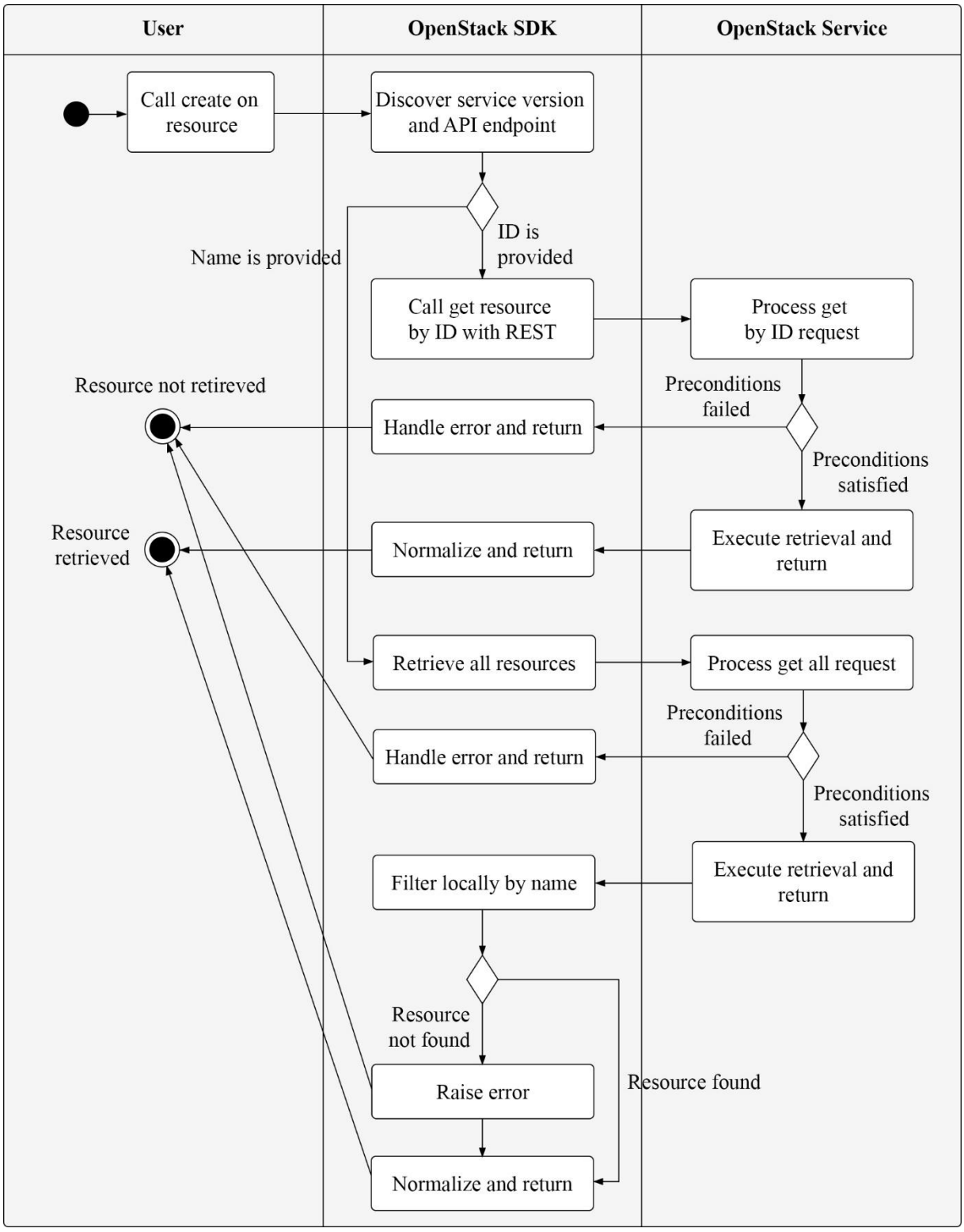


Source: Created by the author of this document.

Get:

The get function retrieves a single resource, given its name or ID, as in Figure 37. After retrieving the resource by using the service REST API, it normalizes the server response and returns to the user. There are different workflows depending on whether a name or ID is provided, and multiple points where exceptions might occur.

Figure 37 - The get resource function



Source: Created by the author of this document.

List:

This function lists all the resources of a given type by querying the service with the REST API and then normalizing each one of the returned resources, exactly and in the workflow for the get resource when a name is provided in Figure 37, except that there is no filtering after that, and all resources are normalized, one by one.

Search:

With the filters on the resources' attributes as the function parameters, the search function will filter in all resources that exact match all the attributes' values. In most of the cases, it retrieves all resources from the server as in the list operation and applies a client-side filtering on the normalized entities.

The filters can be in two formats: a dictionary or an expression using JMESPath (JMESPath Documentation, 2015), a query language for JSON, as in Figure 38.

Figure 38 - Filters for the search function

```
 1 $ python
 2 >>> import openstack
 3 >>> cloud = openstack.connect('devstack')
 4 >>> [server.name for server in cloud.list_servers()]
 5 ['abc', 'abcdef']]
 6 >>> [server.name for server in cloud.search_servers(filters={'name': 'abc'})]
 7 ['abc']
 8 >>> [server.name for server in cloud.search_servers(filters='[?name == `abc`]')]
 9 ['abc']
10 >>> [server.name for server in cloud.search_servers(filters='[?contains(name, `abc`)]')]
11 ['abc', 'abcdef']
```

Source: Created by the author of this document.

Lines 4 and 5 show there are two servers, named *abc* and *abcdef*. The filter with dictionary, as show in lines 6 and 7, does a perfect match in the provided attribute and only returns one entry. The approach with JMESPath is more flexible, allowing filtering in various manners; for example, lines 8 and 9 show a perfect match, just as in the dictionary filtering, while lines 10 and 11 show a custom filter where the name of the server must contain *abc*, causing both servers to be returned.

Update:

The attributes and respective new values are provided to the update function, which then calls the service REST API. The server responds with the updated resource, the library normalizes it and returns to the user, behaving very similarly to the create function in Figure 36. OpenStack SDK does not necessarily allow update of all the resource attributes, restricting to those in the resource data model.

Delete:

With the user provided resource name or ID, this function deletes the correspondent resource. When a name is provided, it first finds the resource with the workflow of the get function, as in Figure 37, and then proceeds with the deletion by ID. It returns a Boolean value to inform if the operation succeeded; otherwise, it returns False. Any other exception returned by the server is passed on to the user.

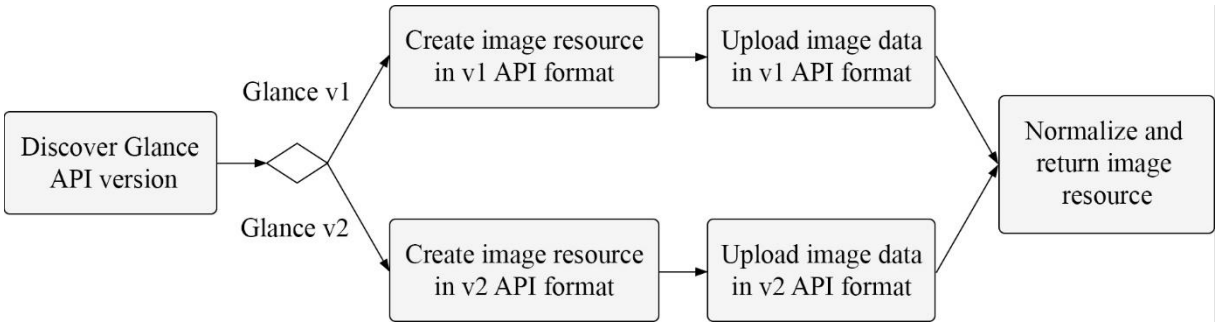
3.3.9 Image Functions

Glance, the Image service, has its resource documented in the Subsection 2.3.12, and all its major versions, 1 and 2, supported by OpenStack SDK. The core operations for images are implemented exactly as in Subsection 3.3.8, except for the create operation that has some particularities.

There are two possible workflows to create an image: uploading it directly via the PUT API of the Image service; or by uploading it to the Object Storage service first and then importing it into the Image service via its Tasks API.

The first workflow is the default and consists of two steps: create the image resource calling HTTP POST on the image API, and then uploading the image data into the resource via HTTP PUT on the resource URL. As the library supports versions 1 and 2 of Glance, it first checks what version is available and then proceed with the appropriate calls, as represented in Figure 39.

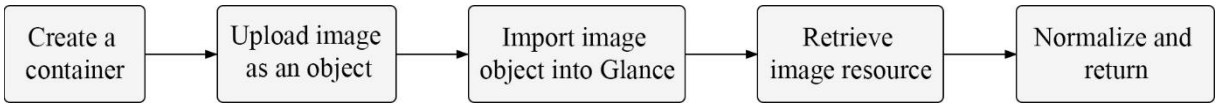
Figure 39 - Creating an image via the PUT API



Source: Created by the author of this document.

The second workflow is only available in the version 2 of the service, and comprises two major steps as well: the library creates a container and uploads the image as an object to the Object Storage service, and then imports the image into the Image service with a POST on the image tasks API, importing the image object into Glance, then formatting it to return, as in Figure 40. The first two steps in the diagram refer to communication with Swift, while the two subsequent calls are to Glance.

Figure 40 - Creating an image via the Tasks API



Source: Created by the author of this document.

The images returned by the create, get, list, search and update operations are normalized to the data model defined in the Section C.1 Image of APPENDIX C – OPENSTACK SDK DATA MODEL.

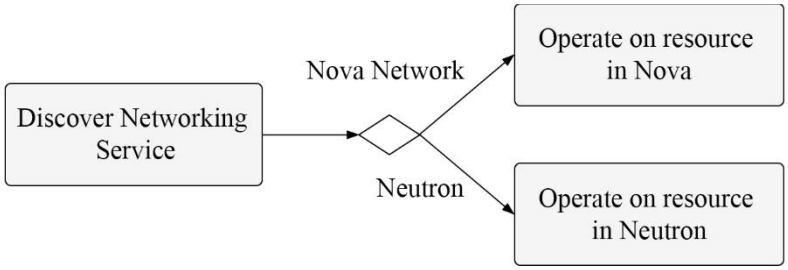
OpenStack SDK does not automatically discover what mechanism the underlying cloud accepts and run that transparently to end users. The great majority of the clouds accept image creation via the PUT API mechanism. If that is not accepted, the user needs to explicitly set in the library's configuration file that a specific cloud uses the Tasks API workflow, resulting in a rare one-time manual configuration.

3.3.10 Networking Functions

The networking resources are implemented by either Nova, the Compute service, or Neutron, the Networking service of OpenStack, as documented in the Subsection 2.3.13.

OpenStack SDK supports both Nova and Neutron in their major versions 2. The core operations for networks, subnetworks and security groups are implemented exactly as in Subsection 3.3.8. Therefore, the library only needs to discover what Networking service is running on the cloud and proceed with the appropriate call, as in Figure 41.

Figure 41 - Discovering and using the Networking service



Source: Created by the author of this document.

The networks, subnetworks and security groups returned by the create, get, list, search and update operations are normalized to the data model defined in the Section C.2 Networking of APPENDIX C – OPENSTACK SDK DATA MODEL.

3.3.11 Block Storage Functions

Cinder, the Block Storage service managing the compute resources documented in the Subsection 2.3.14, has all its major versions, 1, 2 and 3, supported by OpenStack SDK. The core operations for volumes resources are implemented exactly as defined in Subsection 3.3.8. The only particularity is in the update function that only updates descriptive attributes of volumes, such as its name, not supporting the resize of volumes, which is by itself another function exposed in the service REST APIs.

The create, get, list, search and update operations normalize the representation returned by the service REST API to the volume data model defined in the Section C.3 Block Storage of APPENDIX C – OPENSTACK SDK DATA MODEL.

3.3.12 Compute Functions

Nova, the Compute service managing the compute resources documented in the Subsection 2.3.15, has its major version 2 supported by OpenStack SDK. The core operations

for the resources in this group are implemented exactly as defined in Subsection 3.3.8, with further details on specific behavior of some operations.

The flavors, keypairs, servers and snapshots returned by the create, get, list, search and update operations are normalized versions of the representations returned by the service's REST APIs to the data model defined in the Section C.4 Compute of APPENDIX C – OPENSTACK SDK DATA MODEL.

Flavor:

Flavors are read-only resources for end users, manageable by cloud operators since the set of flavors is cloud-wide, affecting all tenants in a cloud. Therefore, the create, update and delete operations are out of the scope of this research, which focus on the user perspective.

Keypair:

Keypair resources contain the public-key secret that allows asymmetric cryptography of the user credentials for logging into the servers via SSH. Such key is an immutable entity, thus are keypairs. Consequently, all the core operations but update are implemented in this case.

Private Server:

Managing servers is generally the most important use case when orchestrating computational infrastructure, as it ultimately provides assets for users to deploy software on. A server depends on a flavor, keypair, network, subnetwork and security group, as per the Subsection 2.3.17, and all those dependencies are explicitly provided to the create operation of OpenStack SDK as function parameters during the validation process in this study.

Public Server:

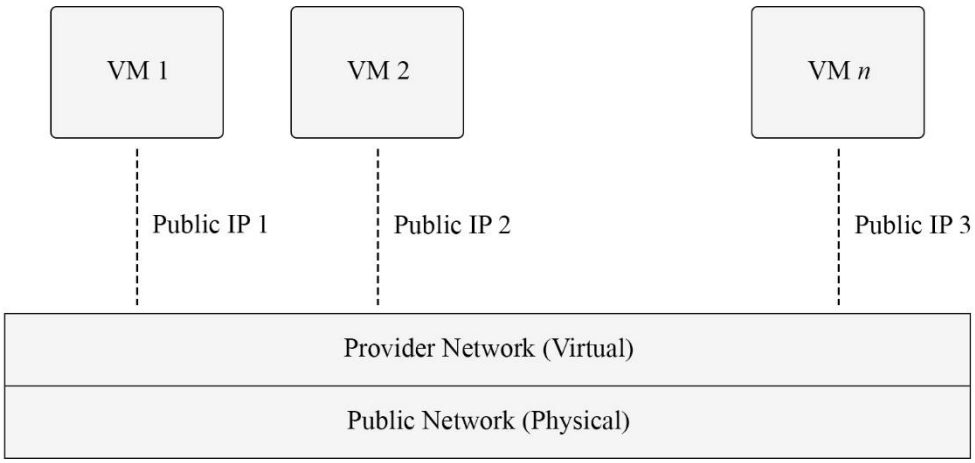
A public server is the same resource as a private server, except for the fact it has a public IP, allowing it to be accessed publicly from outside the OpenStack deployment, which generally means the internet.

When creating a public server, the library accepts the *auto_ip* Boolean parameter, enabling it to discover how the cloud functions and automatically assign a public IP to the

server. This process might happen in two distinct mechanisms: with the IP routed via DHCP or via NAT.

DHCP implements a central management of IP distribution in a network, where a new device that connects communicates to the DHCP server to have an IP address assigned to it. In OpenStack, this generally occurs when the new server is put in a virtual network that connects to a physical network infrastructure via bridging or switching, i.e. a layer 2 connection, which in turn has a DHCP server distributing public IPs, as per the architecture represented in Figure 42. Such virtual network in known as provider network, since it needs to be configured by the service provider as it requires configuration against a physical network.

Figure 42 - Architecture allowing public IP via DHCP

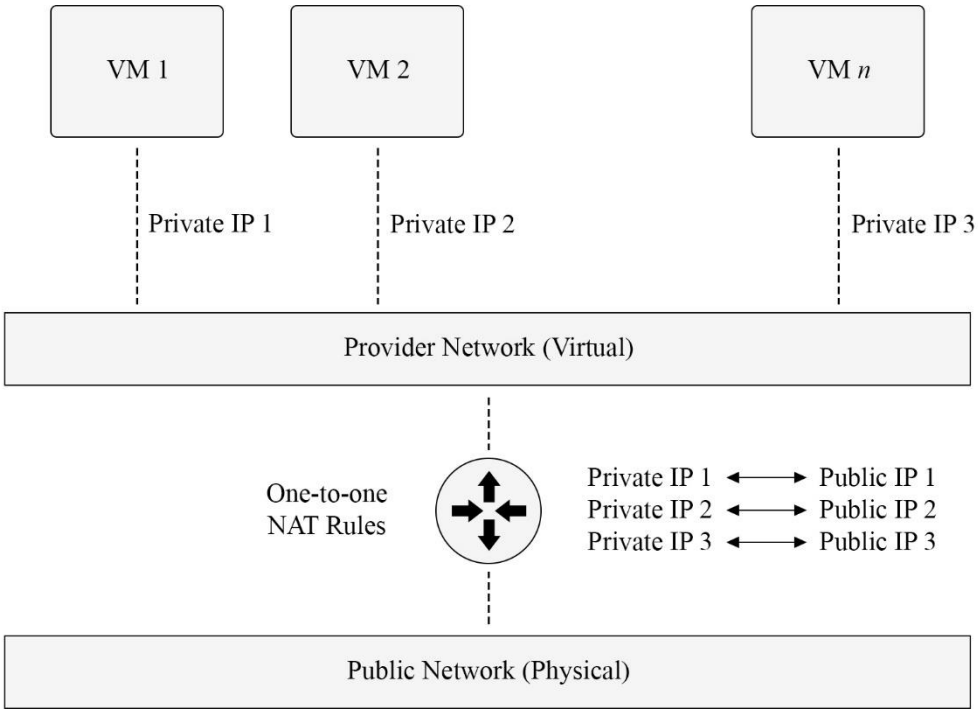


Source: Created by the author of this document.

NAT remaps an IP address space into another by modifying the headers of IP packets in transit across a NAT router. Multiple NAT strategies exist, including Source Network Address Translation (SNAT), Destination Network Address Translation (DNAT) and one-to-one NAT. OpenStack uses all those strategies but for different purposes, specifically applying the one-to-one strategy to implement the public IP addresses on servers, as in the diagram of Figure 43.

In one-to-one NAT, the NAT router maintains a one-to-one mapping between private and public IP addresses. When a packet is outbound, it rewrites the sender private IP address with its public IP address, enabling the remote device to reply the request to the sender, when appropriate. When a packet is inbound, for example in a reply from the remote device, the NAT router rewrites the destination public address with its respective private IP.

Figure 43 - Architecture allowing public IP via NAT



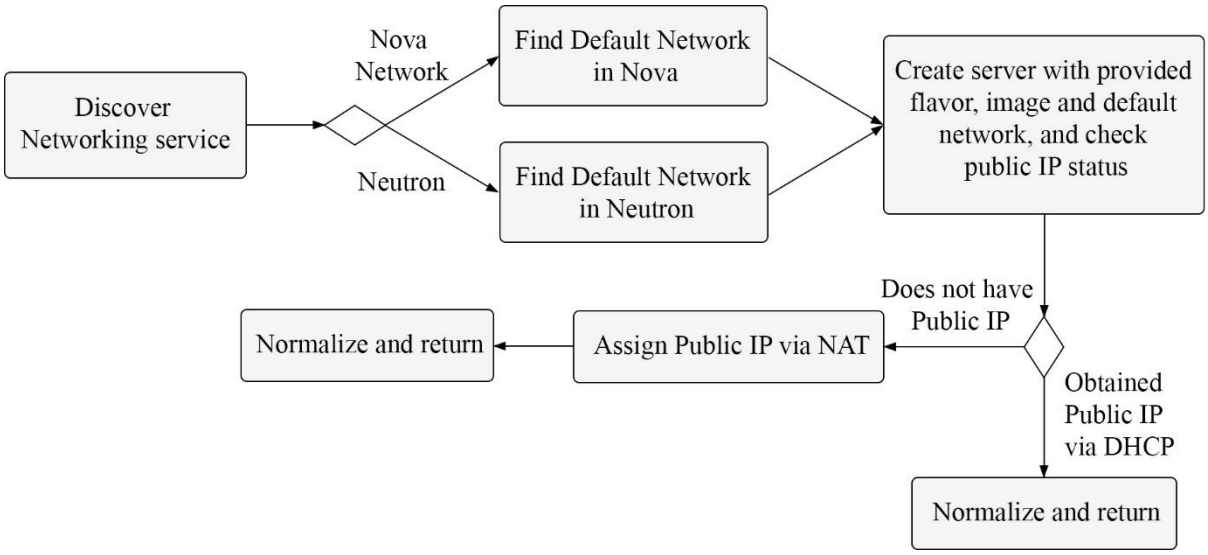
Source: Created by the author of this document.

In practice, having DHCP or NAT mechanisms in place might impact the network performance in the servers. In the NAT strategy, all the network traffic originating from and with destination to servers necessarily need to go through the OpenStack control plane, where the NAT rules are written. In the DHCP strategy, the network communication goes directly over the physical provider network, avoiding a potential virtual networking bottleneck at the control plane.

To make the process transparent to the end users, the library first discovers what is the default network if one has not been provided. Then, it creates the server and checks whether a public IP has been assigned via DHCP already. If it has not, it assigns one via NAT. In this process, the client library communicates with the networking service, which can be Neutron or Nova. The overall process is shown in Figure 44.

With that, the process of creating a public server in the validation documented in this research only requires explicit arguments for flavor and image to be provided to OpenStack SDK, as the network will be transparently handled by the library, as opposed to providing all of them as expected by the Subsection 2.3.17.

Figure 44 - Creating a server and assigning a public IP to it



Source: Created by the author of this document.

Snapshot:

Snapshots are resources manageable through the exact same operations as images, except for the create operation that receives a server as parameter, instead of an image data. Snapshots are created in a cold or live approach. In the former, the server is shut down before taking the snapshot, as opposed to the latter. Although most clouds implement the cold method, it is not discoverable as a user, therefore OpenStack SDK can only guarantee that the snapshot is taken, regardless of how the cloud is configured.

3.3.13 Object Storage Functions

Swift, the Object Storage service managing containers and objects is documented in the Subsection 2.3.16, has all its major version 1 supported by OpenStack SDK. The core operations for its resources are implemented exactly as defined in Subsection 3.3.8, except for the update operation.

Containers and objects cannot update any of its main attributes, since these resources only have names as attributes provided by the users and names are IDs, meaning that they are immutable. The content of a container is a set of objects, and an object can have its entire content replaced by another, but this is not considered as an update in the scope of this investigation, since creating another object with the same name would have the same replacement effect.

The create, get, list, search and update operations normalize the representation returned by the service REST API to the volume data model defined in the Section C.3 Block Storage of APPENDIX C – OPENSTACK SDK DATA MODEL.

3.3.14 Supported Service Versions

The service versions supported by OpenStack SDK by the time of this investigation are summarized in Table 4.

Table 4 - Supported service versions

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	2; 3	1; 2	2	1; 2; 3	2	1

Source: Created by the author of this document.

Those represent all the service versions since the conception of OpenStack as an open source project. The versions 1 for the authentication and compute services predate OpenStack, when the project was developed uniquely by Rackspace. The version 1 of the Networking service, Neutron, never existed; it was created as an evolution of the Nova Network service, which by then was in its version 2 and therefore, for convenience with the versioned URLs, Neutron started in the same version. Details about the OpenStack history and the evolution of Nova Network into Neutron, the Networking service, are documented in the Subsection 2.3.1.

3.3.15 Helper Functions

Some operations include multiple functions calls to the services or are not consistently discoverable across multiple clouds, presenting poor user experience. OpenStack SDK implements some operations to relieve the burden of querying such information.

Flavors are not consistently represented across heterogeneous clouds, which might have completely different sets of flavors available to their users. The operation *get_flavor_by_ram* returns a flavor based on its RAM requirement, expressed in MB. In contrast with the flavor search operation, this function does not perform an exact match on that attribute, finding the flavor with the least amount of RAM that is at least the specified amount. For example, calling this function with the value *512* would return a flavor with exactly 512 MB of RAM or the closest higher value to that, which could be 1024. If performed with the search flavor operation,

this would require multiple calls until a flavor is found. Another alternative would be to list flavors and manually iterate over the list.

Different clouds might have different sets of services. It can be convenient to users to check beforehand if a service is available before trying to perform operations against it and potentially receiving an exception informing the URL for that service could not be found. For that purpose, the library implements the *has_service* operation. For example, the Object Storage service can be checked by passing the value *object-store* to the *has_service* operation that will return a Boolean value.

As a last example, *get_external_ipv4_floating_networks* returns a list of network resources that use IPv4 and have communication with the environment outside the cloud, which generally is the internet. Conversely, the operation *get_internal_ipv4_networks* retrieves the IPv4 private networks. Similarly, the operations *get_external_ipv6_networks* and *get_internal_ipv6_networks* implement the same behavior for IPv6 networks, respectively. Alternatively, the user to would use the list or search operations to find networks with these specifications.

Although these operations are important to improve user experience and many other examples exist in the library's code, they are not enumerated in this document because only the core operations for creation, retrieval, update and deletion, as in the Subsection 3.3.8, are in the context of this investigation.

3.4 Continuous Integration

Being an OpenStack project, this library has its code quality guaranteed by a Continuous Integration (CI) system that runs numerous tests suites for every code change proposed to its repository, protecting the library against regressions in the code base.

Observing a sample code change (Change 60050, 2018) submitted to the review system, OpenStack SDK has its code tested in 30 different environments, as per the section *Zuul gate* in the review page, where it combines various test categories and scenarios.

The overall process of setting up the testing environments is in the Subsection 2.3.18, with the specific information for this project as follows.

Testing Environments:

Each of those 30 environments is configured in a separate virtual environment of one or more virtual servers, resulting in at least 30 virtual machines being created to test a single code change proposed to OpenStack SDK.

To create such virtual machines, the CI system uses a released version of OpenStack SDK to communicate to multiple third-party clouds, using donated computational resources. This process applies to any project in the OpenStack ecosystem, not only to this project. Consequently, if a regression is introduced in the code and goes in a released version that is adopted by the CI system, all the testing for all the code changes of all the OpenStack projects would be affected because no virtual machines would be set up to run the tests on, therefore the importance of maintaining the quality of this project.

Test Categories and Scenarios:

OpenStack SDK has tests for its code style and behavior, along with code coverage and documentation jobs.

The code style tests run checks with the Flake8 (Flake8 Documentation, 2016) library, which include tests for the Python Enhancement Proposal 8 (PEP 8 - Style Guide for Python Code, 2013) and others. PEP 8 contains the coding conventions for the code of the Python standard library.

Behavioral tests include thousands of unit, functional and upgrade test cases that run against several different scenarios. The unit tests run against the Python versions 2.7, 3.5 and 3.6. The functional tests run against a DevStack cloud that is configured in multiple manners, including some services that are not included by default, such as Trove, which provides Database as a Service. Regarding the upgrade tests, it uses the OpenStack Grenade (Grenade Documentation, 2018) project to ensure that the library maintains its behavior across OpenStack release upgrades. Lastly, there are third party unit and functional tests running against the code, ensuring the new code is not breaking any behavior in libraries that depend on it, such as the Ansible OpenStack modules, the OpenStack Client command-line library and Nodepool, which is the part of the OpenStack CI system responsible for provisioning virtual servers with the donated resources on multiple clouds.

The documentation job builds the documentation with the Python documentation generator Sphinx (Sphinx - Python Documentation Generator, 2018), ensuring that no

regression is preventing the latest documentation to be build and published. The coverage jobs generate information that is useful to reviewers to identify whether the current and new code have enough test coverage.

3.5 Author Contributions

The author made numerous contributions throughout the period of this research, including failure fixes, documentation improvements, refactoring, cleanups and new functionalities.

In total, the contributions altogether represent 71 commits, with bugs or feature requests opened in the OpenStack SDK bug tracker system (OpenStack SDK StoryBoard, 2018). Although these contributions are considered as relevant by the community, as the library is constantly improving and expanding the list of supported services and functionalities, measuring the impact and importance of the contributions, along with the discussions and technical decisions, is not trivial hence, it is not mentioned in this document. Further description of the contributions is present in APPENDIX D – AUTHOR CONTRIBUTIONS.

3.6 Further Resources

Although out of the scope of this investigation, this library supports a range of additional resources: bare metal server, container orchestration engine, database, cluster, orchestration stacks, DNS service and identity resources, managed by services further documented in APPENDIX A – ADDITIONAL OPENSTACK PROJECTS.

Ironic (Ironic Documentation, 2018) is the service providing bare metal servers, as opposed to virtual servers, managing provisioning of servers directly on hardware through remote management protocols, as documented in the Section A.8 Bare Metal.

Magnum (Magnum Documentation, 2018) and Trove (Trove Documentation, 2018) are further detailed in the Section A.3 Workload Provisioning. Magnum delivers container orchestration engine functionalities for deploying and managing containers with user applications. Databases are offered as a service with Trove, allowing rapid provision and use of features of relational databases, while handling the complex administrative tasks.

Senlin (Senlin Documentation, 2018) and Heat (Heat Documentation, 2018) are described in the Section A.5 Orchestration. Clusters are provided by the former and allow

improved management of groups of homogeneous resources provided by other OpenStack services. The latter provides orchestration stacks, offering a methodology based on templates for describing the computational infrastructure for a cloud application.

Designate (Designate Documentation, 2018) implements DNS services with the management of DNS zones and records. It is in the Section A.7 Further Networking.

Lastly, Keystone (Keystone Documentation, 2018) Identity resources comprise projects, domains, users and groups of users, roles and role assignments, service catalog and anything around authentication and authorization, being documented in the Section A.10 Shared Services.

4 OPENSTACK SDK VALIDATION

The OpenStack SDK code has its quality guaranteed by numerous tests suites, as described in Section 3.4. However, due to limitations in the testing process, that is not enough to assure it can talk to multiple clouds by abstracting numerous configuration differences.

The limitations are due to the fact its functional tests only run against a DevStack cloud, which creates a not production-ready cloud with the development versions of services. Therefore, the range of service versions is limited to the latest versions. Additionally, those clouds are always configured the same manner, meaning there is no variation in the configuration between deployments.

With the aforementioned limitations, a validation with various cloud providers with a comprehensive range of API versions and configuration choices is necessary, as presented in this chapter. The Section 4.1 details the tests designed and developed for the validation; Section 4.2 presents the cloud environments selected to run the tests against; Section 4.3 documents the results of running the tests against the target clouds; and Section 4.4 explores the test results.

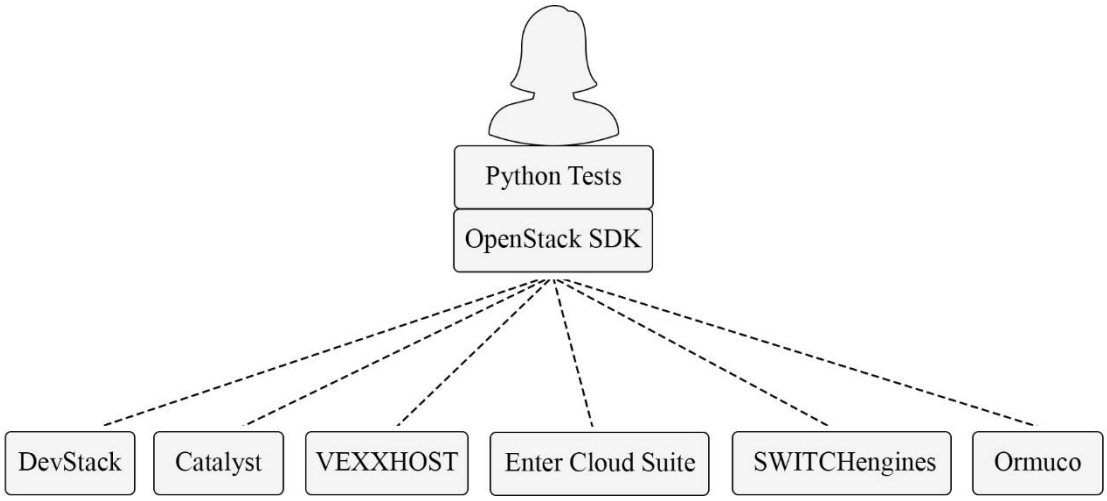
4.1 Tests

This section provides the design, implementation and limitations of the functional test cases developed to validate OpenStack SDK against real production clouds around the globe.

4.1.1 Design

The functional test cases run locally against the remote clouds by communicating to their service APIs, as illustrated in Figure 45. Regarding individual test case design, each one runs independently and does not leave any garbage after executing, regardless of its outcome, avoiding unnecessary residual allocation of resources and consequently unplanned invoices when testing against the real clouds.

Figure 45 - Local environment running tests against remote clouds



Source: Created by the author of this document.

Coverage:

Aiming to provide a comprehensive validation, the test cases will cover all the supported operations for each resource, along with the authentication functionalities that are transparently utilized by OpenStack SDK when operating on resources. As documented in the Subsection 3.3.8, there are six operations on resources: create, get, list, search, update and delete. As enumerated in the Subsection 2.3.11, there are 12 resources in this scope: flavor, keypair, private server, public server, snapshot, image, network, subnetwork, security group, volume, container and object. With that information, applying the pairwise testing combinatorial method, it results in 72 unique test cases per target environment.

Skips:

Six out of the seventy-two tests are skipped for not applying to the resource in question: create, update and delete of flavor, which are operations managed by the cloud provider; update of keypair name or content, which is an immutable resource; and update of container and object names, which represent immutable identifiers.

4.1.2 Implementation

The test cases are written in Python and use the OpenStack SDK library to communicate to the remote clouds and to execute operations on the resources. The operations responses are

submitted to assertions, assuring that both the broker and the cloud behavior are the same regardless of the underlying cloud service versions and deployment configuration choices.

Therefore, the local environment installs the OpenStack SDK library, along with its configuration file defining what clouds are available and how to authenticate to them. As additional dependencies, *unittest* (Unit testing framework, 2018) and *nose* (nose Documentation, 2018) Python testing libraries are necessary.

In terms of numbers, the 72 tests cases represent about 450 lines of code, organized in 26 classes using the concepts of inheritance and polymorphism of the object-oriented programming paradigm to reuse code as much as possible.

Further details about the architecture and implementation of the test cases, as well as the configuration of the local environment and guidelines to discover the services in the target clouds are documented in APPENDIX E – TESTING CODE AND ENVIRONMENT.

Test Create:

Every resource starts its life cycle by its creation. The parameters to a create API call vary largely depending on the resource fields, which in turn may vary depending on the version of the service that manages that resource. For example, a newer version of the image service renames a field in the create image operation.

This test case ensures that, by providing the appropriate parameters, it is possible to create a resource successfully. The created resource must always have the same representation, depending only on the provided parameters and independently of the underlying provider.

Test Get:

When handling resources, it is often necessary to query full information about a specific resource of which the ID or name is known. An example is a user retrieving more information about its account by providing its username.

This test case ensures that a single resource retrieved always has the same representation, independently of the provider running the service.

Test List:

Presenting a list of resources of a type is often an operation that is useful for user interfaces. An example is listing all servers that are currently running for an account. Unlike all the other operations, listing does not take any argument and returns all the resources.

This test case asserts that all the entities returned by a list operation are have the expected fields for that resource type.

Test Search:

End users do not always remember the full name of resources, nor the exact values of their fields. Additionally, cloud providers do not name default resources consistently; for example, the latest Ubuntu image will probably have different names in different providers. Therefore, being able to search resources based on part of their fields' values is a powerful feature. For example, one might want to search an image that has the string *ubuntu* in its name.

This test case ensures that the search operation always returns only the resource that is sought, and that it has the expected fields.

Test Update:

Updating fields of resources allow the system to continuously reflect the needs of end users. For example, a private network named *private* that has been connected to the internet via a router might be renamed to *public*.

This test case ensures that it is possible to update resources regardless of the underlying cloud, and that the resource in the response contains all the expected fields, including the updated field with the new value.

Test Delete:

Cleaning up resources is essential, especially in an environment where the resources are very limited, as in a small private cloud, or when using the pay-as-you-go model in a public cloud, avoiding the waste of resources and money. For example, one might want to delete a server that is shut down and thus is not going to be useful anymore.

This test case assures that a resource disappears of the user account by trying, without success, to retrieve it after its deletion.

4.1.3 Limitations

Although the tests cover all the operations for the resources in the scope of this investigation, they present some limitations due to the difficulty of testing numerous complex resources in various environments.

Firstly, not all test paths are exploited, testing only the default scenarios without featuring exceptional or error conditions. For example, there is a test for the server creation operation, but none that attempts to create a server without providing an image or with a network that does not exist.

Secondly, although all the tests assert that all the expected fields are present in a resource, they do not check all the expected values. Generally, only the values of the resource identifier and location, which describes what cloud and scope the resource belongs to, are checked.

In third place, only one resource of its type is created for each test case. For example, a single server is built when running the test for the get operation. However, for the test cases retrieving resources, i.e. get, list and search, it would be ideal to have more than one resource created to make sure the method is operating as desired with multiple elements.

An alternative to implementing the test suites was to use the existing functional tests in OpenStack SDK; however, those tests are limited to some operations per resource, do not guarantee to not leave garbage after executing and are not ready to run against an arbitrary cloud yet. Plans include, however, contributing to the project with the tests created in this investigation and setting up the functional tests to run against real cloud environments, enhancing the Continuous Integration and consequently the quality of the code delivered by the project.

4.2 Testing Environments

When seeking the remote testing environments, the objective was to select cloud providers whose deployments differ the most from each other, increasing the coverage of service versions and the code paths for creating public servers and images.

The code paths for creating a public server is documented in Subsection 3.3.12, where routing its public IP can occur via DHCP or NAT mechanisms. In the image creation case, as described in Subsection 3.3.9, it can occur via the PUT API of the image service directly, or via a two-step process where the image goes to the Object Storage service and then is imported into the image service via the Tasks API. In terms of service versions, by the time of this investigation, OpenStack SDK supports the major versions for the services enumerated in the Subsection 3.3.14.

To discover what service versions a cloud runs, as the cloud providers do generally not make this information public, it is necessary to create an account in the cloud, authenticate and then run the service discovery logic as in the Section E.3.3 Cloud Services of the APPENDIX E – TESTING CODE AND ENVIRONMENT. However, for the code path coverage, there is no way to an end user to discover what code path a deployment will run before trying it out.

For the aforementioned reasons, the service versions coverage was the main criteria when selecting the deployments, along with some punctual recommendations by the author, resulting in the following clouds: DevStack, Catalyst, VEXXHOST, Enter Cloud Suite, SWITCHengines and Ormuco. In the selection process, some clouds were not included because either: (i) the cloud was not available for sign up as an individual, only as a company; (ii) the cloud was an outlier in terms of the syntax exposed by OpenStack SDK to support it, which was the case for the Rackspace public cloud that has a specific syntax for authenticating users.

4.2.1 DevStack

DevStack (DevStack Documentation, 2018) is a series of scripts that can quickly install an OpenStack environment with the latest versions of the services from their source code repositories. It is generally used as a development environment and as the basis for most of the OpenStack projects' functional testing by having the functional tests run against it.

By the time of this investigation, DevStack installs the versions of the services presented in Table 5, in a default region with name *RegionOne*.

Table 5 - Service versions in DevStack

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	3	2	2	3	2	1

Source: Created by the author of this document.

A DevStack cloud is set up and has the OpenStack SDK functional tests run against it whenever a code change proposal is submitted to the OpenStack review system, as detailed in the Section 3.4. Despite of that, it was chosen by the author because those functional tests are limited to some operations per resource; are not guaranteed to not leave garbage after executing and are not prepared to be run against an arbitrary cloud yet, as mentioned in the Subsection 4.1.3. Additionally, this environment was useful during the development of the tests, where the scripts ran several times without affecting any production environment, avoiding garbage to be left and consequently unexpected invoices.

The details about installing DevStack and discovering its service versions are available in the Sections F.1 Deploy and F.2 Environment Setup of APPENDIX F - VALIDATION ON DEVSTACK, respectively.

4.2.2 Catalyst

Catalyst (Catalyst Cloud, 2018) is a cloud provider delivering robust and innovative cloud services in New Zealand, enabling the growth of New Zealand's digital economy with state-of-the-art cloud infrastructure and platform services delivered onshore.

Table 6 shows the versions of the services used by the validation as selected by the discovery process, except for the Block Storage, which had its default version 3 explicitly set to 1 for increasing the versions coverage, as detailed in the Section G.1.2 Services of APPENDIX G – VALIDATION ON CATALYST.

Table 6- Service versions in Catalyst

Service	**Authentication**	**Image**	**Networking**	**Block Storage**	**Compute**	**Object Storage**
Version	3	2	2	1	2	1

Source: Created by the author of this document.

The Catalyst public cloud has three regions exposed in its service catalog: *nz-por-1* representing a deployment running in Porirua, New Zealand; *nz_wlg_2*, for a deployment in Wellington, New Zealand; and lastly *nz-hlz-1*, for a deployment running in Hamilton, New Zealand. The region in Hamilton was arbitrarily chosen by the author.

4.2.3 VEXXHOST

VEXXHOST (VEXXHOST Cloud, 2018) is a Canadian company who runs OpenStack public and private clouds for its customers, providing secure OpenStack based infrastructure solutions to enterprises and developers, accelerating the ability to innovate and increase operational efficiency.

The versions of the services used by the validation were selected by the discovery process and are presented in Table 7, as further detailed in the Section H.1.2 Services of APPENDIX H – VALIDATION ON VEXXHOST.

Table 7 - Service versions in VEXXHOST

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	3	2	2	3	2	N/A

Source: Created by the author of this document.

The VEXXHOST public cloud has two regions in its service catalog: *ca-ymq-1*, representing a deployment running on Montreal, Canada; and *sjc1*, for a deployment in Santa Clara, California, in the United States of America. The region in Santa Clara was chosen by the author to have clouds spread around the globe, as Ormuco is another provider selected and only has a region in Montreal. That cloud region does not offer the Object Storage service in its service catalog.

4.2.4 Enter Cloud Suite

Enter Cloud Suite (ECS) (Enter Cloud Suite Cloud, 2015) is a European open source cloud provider delivering OpenStack public cloud services deployed in three regions, accessible via a two-way proprietary optical link to guarantee the lowest possible latency in other additional six regions.

Table 8 summarizes the versions of the services used by the validation as selected by the discovery process, except for the Image service, which had its default version 2 explicitly set to 1 to increase the coverage of service versions in this validation, as further detailed in the Section I.1.2 Services of APPENDIX I – VALIDATION ON ENTER CLOUD SUITE.

Table 8 - Service versions in Enter Cloud Suite

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	2	1	2	2	2	1

Source: Created by the author of this document.

The Enter Cloud Suite public cloud has three regions in its service catalog: *nl-ams1*, which represents a deployment in Amsterdam, Netherlands; *it-mil1*, for a deployment in Milan, Italy; and lastly *de-fra1*, for one in Frankfurt, Germany. The region in Amsterdam was arbitrarily chosen by the author.

4.2.5 SWITCHengines

SWITCHengines (SWITCHengines Cloud, 2018) provides dynamic services for research and education via compute and storage services in the form of virtual machines to researchers, lecturers and IT-services to universities and related institutions in Switzerland.

The versions of the services used by the validation are presented in Table 9, as selected in the Section J.1.2 Services of APPENDIX J – VALIDATION ON SWITCHENGINES.

Table 9 - Service versions in SWITCHengines

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	3	2	2	3	2	1

Source: Created by the author of this document.

As per its service catalog, SWITCHengines runs two regions: *LS*, representing the deployment executing in Lausanne, Switzerland; and *ZH*, for the deployment in Zurich, Switzerland. The author arbitrarily chose the region in Zurich.

4.2.6 Ormuco

Ormuco (Ormuco Cloud, 2018) is a Canadian cloud provider who has developed an innovative and intelligent software that unifies on-premises and off-premises workloads with a single platform, delivering powerful cloud orchestration and edge computing capabilities by leveraging artificial intelligence, machine learning and end-to-end automation.

Table 10 shows the versions of the services in use by the validation, as automatically selected by the service discovery process further described in the Section K.1.2 Services of APPENDIX K – VALIDATION ON ORMUCO.

Table 10 - Service versions in Ormuco

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	3	2	2	3	2	1

Source: Created by the author of this document.

By the time of this study, the Ormuco public cloud only provides a region running on Montreal, Canada, which name is *prd-ca-yul-2*. Despite not having any difference in terms of API versions when compared to DevStack, VEXXHOST and SWITCHengines, this environment is unique in terms of configuration because it has replaced the Object Storage official OpenStack service with a third-party software that documents compatibility with its API (Ceph Object Gateway Swift API, 2018).

4.2.7 Service Versions Coverage

The clouds selected cover all the service versions supported by OpenStack SDK, as shown in Table 11. All the API versions are the default as selected by the service discovery process, except for the Image service in Enter Cloud Suite deployment and the Block Storage service in Catalyst cloud.

Table 11 - Services version coverage

Service and Versions		DevStack	Catalyst	VEXXHOST	ECS	SWITCHengines	Ormuco
Authentication	2;3	3	3	3	2	3	3
Image	1;2	2	2	2	1	2	2
Networking	2	2	2	2	2	2	2
Block Storage	1;2;3	3	1	3	2	3	3
Compute	2	2	2	2	2	2	2
Object Storage	1	1	1	N/A	1	1	1

Source: Created by the author of this document.

4.3 Results

This section presents the results of running the tests against the selected deployments. In total, 432 test cases were run, of which 36 were skipped because the operation did not apply to the resource, 17 failed and 379 were successful.

4.3.1 DevStack

The test results for the DevStack cloud that was deployed are shown in Table 12, where all the tests were successful.

Table 12 - Test results for DevStack

Resource	Create	Get	List	Search	Update	Delete
Image	•	•	•	•	•	•
Network	•	•	•	•	•	•
Subnetwork	•	•	•	•	•	•
Security Group	•	•	•	•	•	•
Volume	•	•	•	•	•	•
Flavor	N/A	•	•	•	N/A	N/A
Keypair	•	•	•	•	N/A	•
Private Server	•	•	•	•	•	•
Public Server	•	•	•	•	•	•
Snapshot	•	•	•	•	•	•
Container	•	•	•	•	N/A	•
Object	•	•	•	•	N/A	•

Source: Created by the author of this document.

Regarding the specific workflow to create a public server, DevStack assigns a public IP address to it via NAT mechanism. For the image create workflow, it implements via the PUT API of the image service directly. All the details about running the tests against DevStack are in the Section F.3 Test Results of APPENDIX F - VALIDATION ON DEVSTACK.

4.3.2 Catalyst

The test results for the Catalyst cloud are summarized in Table 13, where all the tests were successful, including those running against the Image service that had its version explicitly

forced to 1. For the specific code path of creating a server and assigning a public IP to it, DevStack assigns the IP with routing via NAT. For the image create operation, it uploads the image via the PUT API of the image service directly. The tests output and further details about running the tests against Catalyst are in the Section G.2 Test Results of APPENDIX G – VALIDATION ON CATALYST.

Table 13 - Test results for Catalyst

Resource	Create	Get	List	Search	Update	Delete
Image	•	•	•	•	•	•
Network	•	•	•	•	•	•
Subnetwork	•	•	•	•	•	•
Security Group	•	•	•	•	•	•
Volume	•	•	•	•	•	•
Flavor	N/A	•	•	•	N/A	N/A
Keypair	•	•	•	•	N/A	•
Private Server	•	•	•	•	•	•
Public Server	•	•	•	•	•	•
Snapshot	•	•	•	•	•	•
Container	•	•	•	•	N/A	•
Object	•	•	•	•	N/A	•

Source: Created by the author of this document.

4.3.3 VEXXHOST

The summary of the results of running the tests against the VEXXHOST cloud is presented in Table 14, where all tests were successful, except for the tests of the Object Storage service resources, container and object.

Further debugging discovered that the Object Storage tests failed because the VEXXHOST region in Santa Clara, California, does not deploy Swift, the Object Storage service. For the code path of creating a server, VEXXHOST uses the mechanism of assigning a public IP to it with direct routing via DHCP. For the image create operation, it uploads the image via the PUT API of the image service directly. All the details about running the tests and debugging the failures are in Section H.2 Test Results of APPENDIX H – VALIDATION ON VEXXHOST.

Table 14 - Test results for VEXXHOST

Resource	Create	Get	List	Search	Update	Delete
Image	•	•	•	•	•	•
Network	•	•	•	•	•	•
Subnetwork	•	•	•	•	•	•
Security Group	•	•	•	•	•	•
Volume	•	•	•	•	•	•
Flavor	N/A	•	•	•	N/A	N/A
Keypair	•	•	•	•	N/A	•
Private Server	•	•	•	•	•	•
Public Server	•	•	•	•	•	•
Snapshot	•	•	•	•	•	•
Container	F	F	F	F	N/A	F
Object	F	F	F	F	N/A	F

Source: Created by the author of this document.

4.3.4 Enter Cloud Suite

The test results for the Enter Cloud Suite cloud are shown in Table 15, where all the tests except for the public server were successful, including those running against the Block Storage service that had its version explicitly forced to 1.

Table 15 - Test results for Enter Cloud Suite

Resource	Create	Get	List	Search	Update	Delete
Image	•	•	•	•	•	•
Network	•	•	•	•	•	•
Subnetwork	•	•	•	•	•	•
Security Group	•	•	•	•	•	•
Volume	•	•	•	•	•	•
Flavor	N/A	•	•	•	N/A	N/A
Keypair	•	•	•	•	N/A	•
Private Server	•	•	•	•	•	•
Public Server	F	F	F	F	F	F
Snapshot	•	•	•	•	•	•
Container	•	•	•	•	N/A	•
Object	•	•	•	•	N/A	•

Source: Created by the author of this document.

For the workflow of creating a server, Enter Cloud Suite assigns an IP to it with routing via NAT. For the image create operation, it uploads the image via the PUT API of the image service directly. Further details about running the tests and debugging failures against the Enter Cloud Suite cloud are in the Section I.2 Test Results of APPENDIX I – VALIDATION ON ENTER CLOUD SUITE.

4.3.5 SWITCHengines

All the tests were successful in the SWITCHengines cloud, as shown in Table 16.

Table 16 - Test results for SWITCHengines

Resource	Create	Get	List	Search	Update	Delete
Image	•	•	•	•	•	•
Network	•	•	•	•	•	•
Subnetwork	•	•	•	•	•	•
Security Group	•	•	•	•	•	•
Volume	•	•	•	•	•	•
Flavor	N/A	•	•	•	N/A	N/A
Keypair	•	•	•	•	N/A	•
Private Server	•	•	•	•	•	•
Public Server	•	•	•	•	•	•
Snapshot	•	•	•	•	•	•
Container	•	•	•	•	N/A	•
Object	•	•	•	•	N/A	•

Source: Created by the author of this document.

When creating a server, SWITCHengines assigns an IP to it with routing via NAT. For the image create operation, it uploads the image via the PUT API of the image service directly. The test outputs and further details about running the tests against this environment are in the Section J.2 Test Results of APPENDIX J – VALIDATION ON SWITCHENGINES.

4.3.6 Ormuco

The test results for the Ormuco cloud are summarized in Table 17, where all tests were successful, except for the delete test of the object resource.

Table 17 - Test results for Ormuco

Resource	Create	Get	List	Search	Update	Delete
Image	•	•	•	•	•	•
Network	•	•	•	•	•	•
Subnetwork	•	•	•	•	•	•
Security Group	•	•	•	•	•	•
Volume	•	•	•	•	•	•
Flavor	N/A	•	•	•	N/A	N/A
Keypair	•	•	•	•	N/A	•
Private Server	•	•	•	•	•	•
Public Server	•	•	•	•	•	•
Snapshot	•	•	•	•	•	•
Container	•	•	•	•	N/A	•
Object	•	•	•	•	N/A	F

Source: Created by the author of this document.

Further debugging discovered that the exact reason for the failure was because the Ceph Object Gateway (Ceph Object Gateway Swift API, 2018) is not working as designed by the OpenStack Object Storage APIs, which it documents to have support for. The failure was a consequence of that service returning a HTTP 200 OK instead of a HTTP 404 NOT FOUND when retrieving the object that had just been deleted.

In terms of the specific code paths, when creating a server, Ormuco assigns an IP to it with routing via NAT. For the image create, it uploads the image via the PUT API of the image service directly. All the details about running the tests and debugging the failure are in the Section K.2 Test Results of APPENDIX K – VALIDATION ON ORMUCO.

4.4 Discussion

This section presents a discussion on the service versions, code paths and corner cases exercised, along with the failures and information about the distribution of clouds in the validation.

Service Versions:

All the service versions supported by OpenStack SDK were exercised, being three versions for the Block Storage service; two versions for the Authentication and Image services; and the Compute, Networking and Object Storage with a single version each.

Code Paths:

Concerning the specific code paths of creating a server with a public IP documented in the Subsection 3.3.12, DevStack, Catalyst, SWITCHengines and Ormuco use the NAT mechanism. Conversely, VEXXHOST uses the DHCP mechanism. Enter Cloud Suite failed to create public servers.

For the code paths of image creation documented in the Subsection 3.3.9, the default behavior of doing the upload via the PUT API of the Image Service directly was successful for all the clouds.

Corner Cases:

Not all resources were semantically the same across heterogeneous clouds, which is the case for flavors. Flavors are provider-defined resources and are read-only for end users. Additionally, different cloud providers might expose a different set of flavors with varying CPU, RAM and storage specifications. That translates into different flavors being used in the validation process, but that ultimately does not have any functional impact.

Another corner case is with the provider-default images that are not consistent across clouds. As example, one cloud might offer the latest Ubuntu image and another, might not; and even in the case they do, such images would probably have different naming patterns. To work around that lack of standardization, a new image was created whenever necessary.

Both cases might be further analyzed by observing the interactions of the tests with the OpenStack SDK code in the Sections E.2.5 Compute Code and E.2.2 Image Code of APPENDIX E – TESTING CODE AND ENVIRONMENT.

Failures:

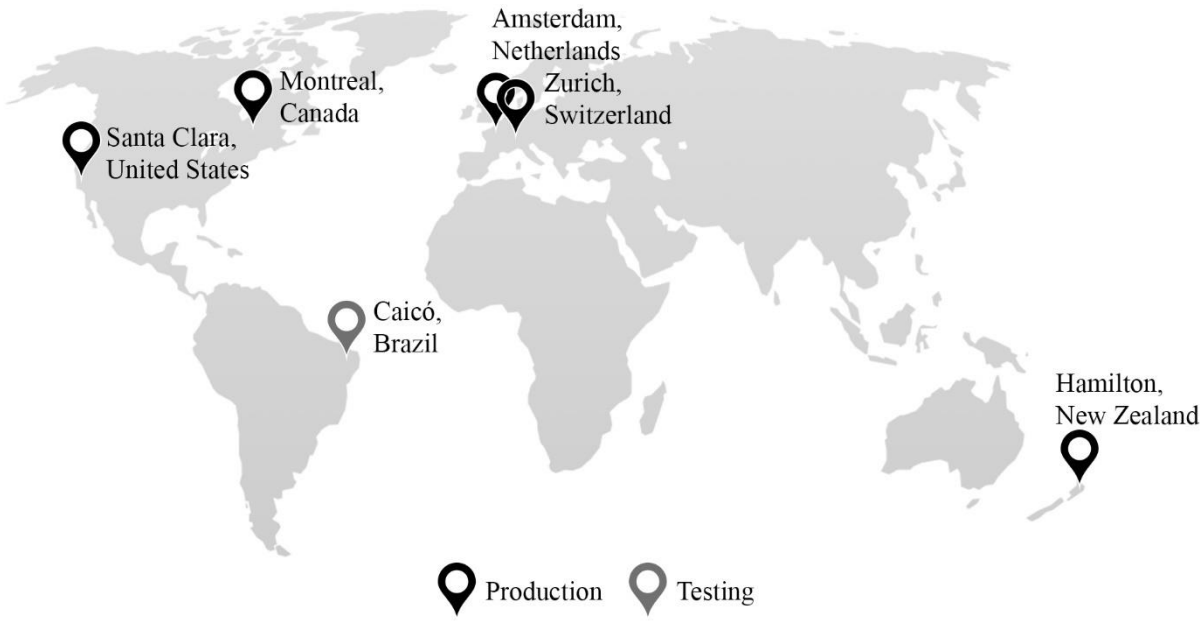
The failures that happened in the process are misbehaviors or missing services in the cloud deployments.

- VEXXHOST failed with all the Object Storage tests because it does not implement that service in the Santa Clara region;
- Enter Cloud Suite failed the public server tests because it puts servers in an active state without having their networking capabilities configured;
- Ormuco failed the object delete test because the third-party backend implementing the Object Storage APIs is not acting as designed by the OpenStack Object Storage service when retrieving an object that does not exist. This deployment choice is against the *OpenStack Primarily Produces Software* principle of OpenStack, which is further detailed in the Subsection 2.3.3.

Clouds Distribution:

Although not a functional selection criterion, it is interesting to mention that the cloud regions in this validation span the globe, as illustrated in Figure 46.

Figure 46 - Validation cloud environments spanning the globe



Source: Created by the author of this document.

Conclusion:

With the aforementioned analyses, the broker behaved as expected throughout the validation process, abstracting the deployment-specific configurations, including the heterogeneous service versions. The coverage of corner cases and service versions was comphreensive and the failures were caused by misbehaviors or missing services in the deployments. The clouds tested represent multiple different providers that span the globe, serving as the foundation infrastructure to numerous businesses.

5 INTEROPERABILITY REVIEW

The previous chapters have established the foundation to OpenStack interoperability, backgrounding what the platform is, along with the documentation and validation of the library that allows its deployments to interoperate.

As an academic investigation, it is important to develop an analysis and a discussion of what has been observed in OpenStack against what is reported in the academy. With that objective, this chapter documents the outcomes of a systematic literature review to comprehend the state of the art on IaaS interoperability, with protocol documented in APPENDIX L – SYSTEMATIC LITERATURE REVIEW PROTOCOL.

The Section 5.1 presents the techniques developed to permit heterogeneous platforms to interoperate, while Sections 5.2, 5.3 and 5.4 detail each of the approaches. Lastly, Sections 5.5 and 5.6 document the causes for vendor lock-in the consequences of having non-interoperable cloud systems, respectively.

5.1 Overview

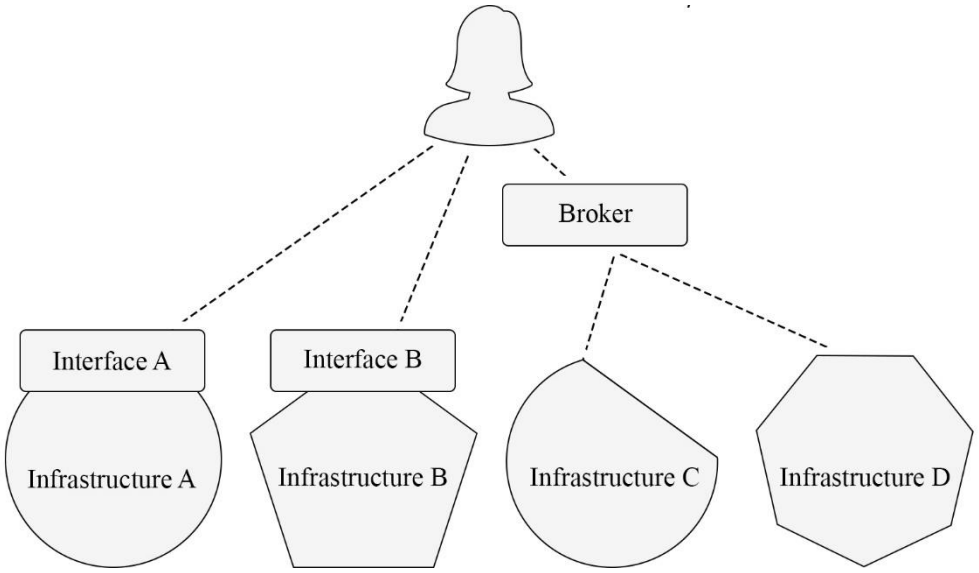
The literature documents two practical approaches to obtain systems interoperability (CHEN and DOUMEINGTS, 2003):

1. Adhesion to published interface standards; or
2. Use of brokers, or middleware, to convert one product's interface into another product's interface.

The former is an approach where end users communicate with the systems directly and, despite the heterogeneous functionality implementation internals of each system, the services provide the same interfaces. In the latter, another level of indirection is plugged into the architecture, where the middleware provides a common interface by abstracting internally the heterogeneity of multiple proprietary interfaces. These approaches are illustrated in Figure 47.

These practical approaches can be classified into provider-centric and user-centric, referring to the agent implementing interoperability; i.e. cloud providers or end users via standard interfaces or brokering, respectively (ZHANG, WU and CHEUNG, 2013). In general, open standards are the most natural means of achieving interoperability, although when competing with post-facto interoperability solutions, i.e. industry standards such as Amazon Web Services, it opens precedents for client-side solutions (PARÁK and ZDENěK, 2014).

Figure 47 - Standards vs brokers



Source: Created by the author of this document.

Each approach enables different scenarios with benefits for both cloud providers and end users (TOOSI, CALHEIROS and BUYYA, 2014), which are documented in the next two sections, along with further details, main proposals and challenges.

5.2 Provider-Centric Approaches

In the provider-centric approach, cloud providers implement services according to standards interfaces, protocols, formats and architectural components that facilitate collaboration. This enables multiple solution scenarios, with Hybrid Cloud, Cloud Federation and Inter-cloud being the most prominent.

The Hybrid Cloud scenario represents a partnership between a private and a public cloud, enabling applications running on the private datacenter to burst into the public environment when the computing utilization spikes, in a process known as cloud bursting.

Cloud Federation is a group of services from different providers aggregated in a single pool collaborating to mutually improve services by sharing resources via federation regulations It benefits both customers, who may profit from lower costs and better performance; and providers who can offer more sophisticated services (KURZE, KLEMS, *et al.*, 2011). For providers operating closely to the resource limitation in the local infrastructure, it provides a work around by outsourcing requests to other members of the federation, avoiding the rejection

of customers. In contrast with that, providers operating at low utilization can lease part of the capacity to other members, avoiding the waste of computing resources.

Inter-cloud presents the concept of global interconnection of clouds, forming a worldwide cloud federation with the adoption of open standard protocols, removing difficulties of migrating applications and supporting the dynamic scaling of applications across multiple clouds (BERNSTEIN, LUDVIGSON, *et al.*, 2009). It mimics the term *network of networks* from the Internet by producing a *cloud of clouds*, where clouds, as networks or telephones, are ubiquitously connected in a multi-provider infrastructure.

5.2.1 Standards

Standardization efforts can be classified according to the challenges being investigated (GARCÍA, CASTILLO and FERNÁNDEZ, 2016):

1. Uniform Access and Management;
2. Portability;
3. Authentication and Authorization;
4. Information Discovery;
5. Accounting and Billing.

As documented in Section 1.5, this investigation focuses on the challenges related to provisioning computational infrastructure in an interoperable manner across heterogeneous deployments, hence the challenges 1, 3 and 4 are of interest, having their main standards enumerated as follows.

Uniform Access and Management:

Three standards present the most prominent proposals in this challenge: Open Cloud Computing Interface (OCCI), Cloud Infrastructure Management Interface (CIMI) and Topology and Orchestration Specification for Cloud Applications (TOSCA).

OCCI (OCCI WORKING GROUP, 2016) was proposed by the Open Grid Forum (OPEN GRID FORUM, 2018) and is a generic protocol and API for management tasks. It was originally initiated to create a remote management API for IaaS, allowing for the development of interoperable tools for common tasks including deployment, autonomic scaling and

monitoring. It evolved into a flexible API with a strong focus on integration, portability, interoperability and innovation, being suitable to serve PaaS and SaaS, in addition to IaaS.

CIMI was proposed by the Distributed Management Task Force (CMWG, 2018) and targets the management of the lifecycle of IaaS resources through a RESTful API. It models servers, storage, and networks, providing consumers with a standard for resource management. It is registered under ISO/IEC 19831:2015 and does not go beyond IaaS to PaaS or SaaS.

TOSCA (OASIS, 2013) is a standard from the Organization for the Advancement of Structured Information Standards (OASIS, 2018) and enables the interoperable description of applications and infrastructure cloud services, the relationships between parts of the service, and the operational behavior of these services, i.e. infrastructure orchestration.

Authentication and Authorization:

The most prominent proposals into this category are X.509 Public Key Infrastructure, Security Assertion Markup Language (SAML) and OAuth.

X.509 (KAUSE and PEYLO, 2012) provides authentication via public key certificates, while authorization is possible by embedding Attribute Certificates (AC) into a certificate proxy, containing assertions about the user.

SAML (OASIS SECURITY SERVICES TECHNICAL COMMITTEE, 2005) was proposed by OASIS (OASIS, 2018) and is built on the top of X.509 certificates and define authentication and attribute assertions in XML to provide access to distributed infrastructures.

OAuth (LEIBA, 2012) proposes a protocol for establishing identity management standards across services with an alternative to sharing usernames and passwords via authorization delegation.

Information Discovery:

Grid Laboratory Uniform Environment (GLUE) was proposed by the Open Grid Forum (OPEN GRID FORUM, 2018) allows publishing concrete data model profiles that dictate how the information is generated and used in a concrete implementation of infrastructure services.

5.2.2 Challenges

There are several challenges for applying standard APIs and protocols in cloud environments (PETCU, 2011):

1. Vendors usually prefer to lock in customers as a retention strategy;
2. Providers want to have unique features to attract more customers;
3. There are numerous standards, with no agreement on which one to adopt.

Consequently, cloud providers do not easily agree on standards, requiring years to fully develop a standard and apply it globally. Additionally, it is important to notice that by creating different interoperability standards and frameworks that are not interoperable between each other will cause the lock-in problem at another level (LOUTAS, KAMATERI, *et al.*, 2011).

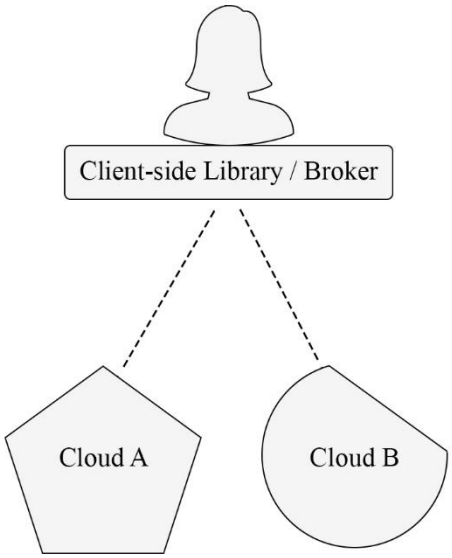
5.3 User-Centric Approaches

Persuading the whole community to agree on and adopt common models may sound unrealistic or at least hard to achieve (LOUTAS, KAMATERI, *et al.*, 2011), where industry might take years to agree upon common standards due to selfishness of the giant cloud vendors and even political obstacles. Therefore, a more flexible approach is demanded.

Instead of standardization, the user-centric approaches present practical solutions allowing users to communicate with multiple clouds seamlessly via the support of interoperability in an upper layer via the use of user-side broker libraries, regardless of the providers' interoperability (ZHANG, WU and CHEUNG, 2013).

Client-side libraries provides the flexibility to run applications on multiple clouds and reduces the complexity of migrating applications between providers. The library functions as a middleware and translates user requests into the specific cloud service description of the heterogeneous environments, as shown in Figure 48. Although technically these can implement support for multiple environments by applying existing standards to the heterogeneous cloud service definitions on the client side, developers typically implement a custom cloud service description that abstract common heterogeneous resources.

Figure 48 - Client-side library



Source: Created by the author of this document.

Although not a different approach, it is important to notice the presence of third-party service brokers (TOOSI, CALHEIROS and BUYYA, 2014), who extend the aforementioned approach to provide high-level end-user services for facilitating effective interoperability in the cloud, utilizing both cloud abstraction client libraries and open standards-based interfaces, even at the same time. Service brokers allow users to simply declare infrastructure needs, as it handles all the complexity of communicating with multiple clouds, orchestrating deployments and often even executing provided workloads and collecting results (PARÁK and ZDENěK, 2014).

5.3.1 Client-side Libraries

Apache Libcloud (APACHE SOFTWARE FOUNDATION, 2018) is a Python library for interacting with many of the popular cloud service providers using a unified API. By the time of this investigation, it supports compute, storage, load balancers, DNS, container and backup functionalities in more than 50 cloud providers. An example of launching a server with this library is presented in Figure 49, where: lines 1 and 2 import the necessary Libcloud Python modules; lines 4 and 5 instantiate the specific driver that will communicate with the Rackspace cloud, in this case; lines 10 and 11 retrieve the image and, lastly the server is created in line 13. Note that the OpenStack concepts of flavors and servers are defined as sizes and nodes, respectively.

Figure 49 - Launching a server with Apache Libcloud

```
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver

cls = get_driver(Provider.RACKSPACE)
driver = cls('username', 'api key', region='iad')

sizes = driver.list_sizes()
images = driver.list_images()

size = [s for s in sizes if s.id == 'performance1-1'][0]
image = [i for i in images if 'Ubuntu 12.04' in i.name][0]

node = driver.create_node(name='libcloud', size=size, image=image)
```

Source: Apache Libcloud Webpage (APACHE SOFTWARE FOUNDATION, 2018).

Another example was the Apache Delta Cloud (APACHE SOFTWARE FOUNDATION, 2015) project, retired in 2015. It supported the majors service providers, supporting functionalities via REST with multiple interfaces: Delta Cloud; CIMI standard; or the Amazon Web Services interfaces. It was implemented in Ruby but had client library support for multiple programming languages.

Lastly, CompatibleOne (YANGUI, MARSHALL, *et al.*, 2013) is an example of cloud service broker assisting service users with choosing providers. It is based on the OCCI standard and defines an object-based description model that serves to manage various cloud resources.

5.3.2 Challenges

Client libraries rely on the cloud's APIs to provide functionalities transparently to end users. The immediate challenge is that a proprietary API is subject to change without prior advice by the original vendor (SILVA, ROSE and CALINESCU, 2013), which would cause the broker to present failures. Therefore, it is not easy to maintain a client-side library without the supported providers' commitment.

In technical aspects, these abstraction libraries present the following disadvantages (PARÁK and ZDENěK, 2014):

1. Client-server communication is generally specific to each provider;
2. Not available for every programming language;
3. Difficulty to maintain and exponential growth;
4. Supported functionalities are either very limited to a subset that represent all supported providers, which may lead to a significant loss of functionalities for some clouds (ZHANG, WU and CHEUNG, 2013); or loss of API unification, resulting in inconsistent behavior as some capabilities might not be available in all clouds.

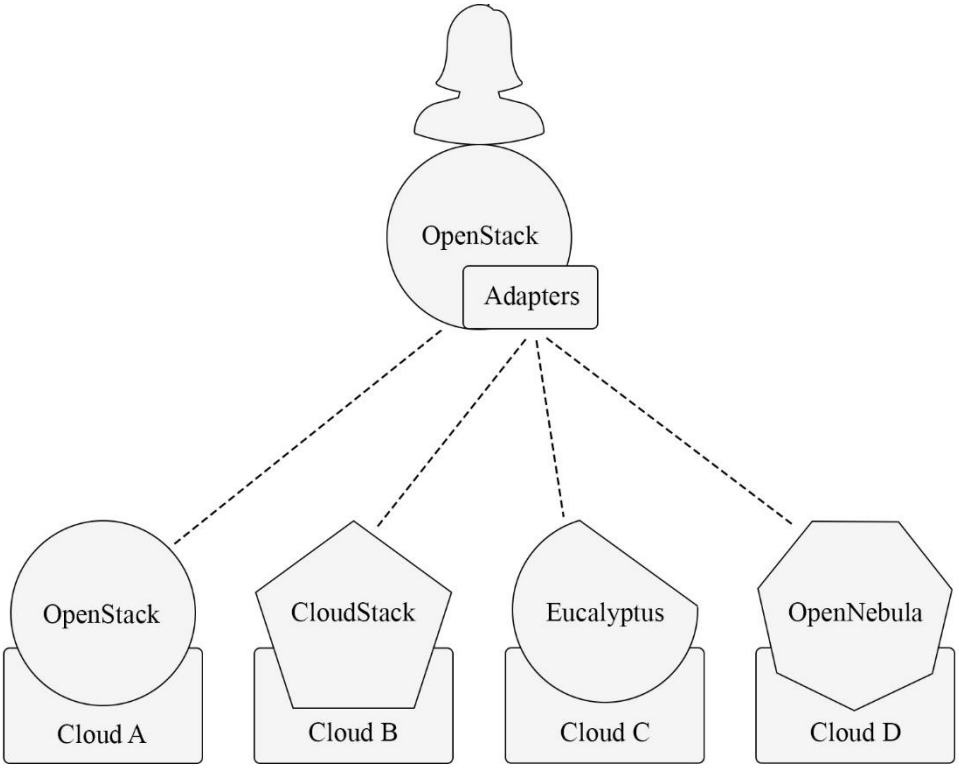
One possible conclusion is that, although libraries can provide the basic functionality, they are not as easily maintained, do not scale well and do not provide much transparency (PARÁK and ZDENěK, 2014).

In performance aspects, adding another level of indirection might present a considerable negative impact on performance and data exchange. With Delta Cloud, for example, relevant time and negligible data overheads were observed, making it suitable only when service response time is not critical (MEIRELES and MALHEIRO, 2014).

5.4 Conceptual Approaches

Some approaches are conceptual and therefore can be implemented directly into libraries or as standards. An example is an approach where each provider's API is modeled by an ontology. Equivalences between these ontologies are modeled by rules, which allow converting calls between providers' specific interfaces. This enables users to interact with different clouds by using the interfaces they are most familiar with (EJARQUE, ÁLVAREZ, *et al.*, 2011). This proposal is illustrated in Figure 50 with the OpenStack interfaces.

Figure 50 - Rule-based approach for interoperability



Source: Created by the author of this document.

5.5 Lock-in Causes

Each service provider uses different techniques to express services: languages, standards, ontologies, models, templates, or tree structure. Therefore, there is not a unified service description standard describing the technical, operational, business and semantic aspects of cloud services (GHAZOUANI and SLIMANI, 2017).

Service descriptions are exposed via cloud interfaces, which implement the technologies and semantics of a provider, therefore providing access to the service management functionalities via APIs. The heterogeneity across different interfaces is a major barrier to cloud portability and interoperability, where even API components implementing simple actions, such as creating a server, vary considerably across providers (SILVA, ROSE and CALINESCU, 2013).

5.6 Lock-in Consequences

By shifting the local infrastructure to the cloud, companies are getting locked into proprietary solutions. Acknowledging that problem, some companies are reluctant to migrate, as the eagerness to adopt the new paradigm is equally urgent as avoiding vendor lock-in. The problem, however, is that customers present lack of awareness of proprietary standards when procuring services from vendors, under-appreciating the complexity and costs of switching providers. Such lack of knowledge poses a significant barrier to obscure the potential effects of the vendor lock-in problems could have on organizations (OPARA-MARTINS, SAHANDI and TIAN, 2016).

Besides highlighting the main consequence of the loss of freedom for users to migrate or combine services whenever convenient, the lack of interoperability has not been documented as cause to limitations on cloud usability or migration of users, therefore affecting the success of providers, as in other studies in software engineering. As an example, by estimating success from user ratings, API change and fault proneness are threats to the success of Android applications, where reliability is an important criteria, since users frustrated by repeated failures abandon some applications in favor of others (LINARES-VÁSQUEZ, BAVOTA, *et al.*, 2013).

6 OPENSTACK INTEROPERABILITY DISCUSSION

The previous chapters have stablished a foundation that permits the analysis of the OpenStack interoperability by comparing it against the state of the art in the literature. Chapters 3 and 4 detailed and validated the technical broker that allows clouds to interoperate, respectively; while Chapter 5 sought and documented the literature cutting-edge investigations around vendor lock-in.

Positioning OpenStack in the literature, this chapter presents an enumeration and a classification of the vendor lock-in scenarios that exist in OpenStack in Section 6.1; subsequently, Section 6.2 provides a deep analysis of the OpenStack approach to interoperable clouds.

6.1 Classification

According to the literature, as described in the Subsection 2.2.3, the horizontal interoperability under investigation can be classified according to the agreement level, i.e. syntactic or semantic, and to the adoption level which can be by design or post-facto. While the adoption level typically refers to the solution altogether, the syntactic and semantic classification can be granular enough to discuss the vendor lock-in scenarios individually.

Therefore, the next subsections provide a definition and enumerate the syntactic and semantic vendor lock-in scenarios observed in OpenStack while documenting and validating its brokering implementation. The last subsection provides a snapshot of the interoperability landscape in OpenStack.

6.1.1 Syntactic and Semantic in OpenStack

OpenStack provides its functionalities via REST APIs, as documented in the Subsection 2.3.9, hence it defines operations over its resources using the REST syntax and semantics, hence an API call generally includes:

- HTTP method: verbs defining the requested action. The primary methods are POST, GET, PUT, PATCH, and DELETE, implementing the creation, retrieval, replacement, granular update and deletion of resources, respectively;
- URL: uniform resource locator, specifying the resource to operate on;

- Request payload: present in create and update calls, it provides resource information to the server, such as a JSON blob representing a virtual server;
- Headers: additional information, such as authentication tokens and pagination request.

When the service processes the user request, it responds using the same data formats and protocols, in which case an API response generally includes:

- Response payload: in the occasion of creation, update and retrieval of resources, the server returns its latest representation;
- Status code: the status of the request processing in the server side, according to the semantics of HTTP status codes. A *404 NOT FOUND*, for example, says the resource specified in the URL does not exist;
- Headers: the server may return additional information in the headers, such a pagination information.

According to the definition in the Subsection 2.2.3, syntactic interoperability is the capability of systems to communicate using the same data formats and communication protocols. The data formats clearly are represented by the request and response payloads components, which carries the representation of resources. The communication protocols, in its turn, comprises the HTTP method, URL and headers in the request and status code and headers in the responses.

Semantic interoperability, in its turn, is defined as the interpretation of the information exchanged as meaningful and accurate within the interoperable clouds. If a user calls multiple clouds with the same API and data format, the services must have the same interpretation of the user request and process it in a consistent manner to generate the same outputs. Correspondingly, the interpretation of the request results returned by the services must be the same by end users.

Therefore, changes in these properties will affect syntactic and semantic interoperability, generating the vendor lock-in scenarios documented in the next subsections.

6.1.2 Syntactic Lock-in Scenarios

As a brokering library, the mechanism OpenStack SDK developed to abstract vendor heterogeneous choices to users is by creating an intermediate representation, or proxy, to protocols and data formats, allowing users to safely rely on it.

In the requests, the REST API calls are mapped via various Python function calls, as explained in the Subsection 3.3.8. The resource fields required by the operation are provided via function parameters, as illustrated in Figure 51. Lines 1 to 8 show the Python code that generates the equivalent REST API call using *curl* in lines 10 to 20, assuming Neutron version 2 is the networking service.

Figure 51 - Python to REST

```
 1 def create_security_group(self, name, description):
 2   json = {
 3     'security_group': {
 4       'name': name,
 5       'description': description
 6     }
 7   }
 8   self.network.post('/security-groups.json', json=json)
 9
10 $ curl -i \
11   -H "Content-Type: application/json" \
12   -H " X-Auth-Token: ******" \
13   -d '
14   {
15     'security_group': {
16       'name': name,
17       'description': description
18     }
19   }' \
20   "http://localhost:9696/v2.0/security-groups.json "
```

Source: Created by the author.

In the response side, the service API returns are mapped to Python function returns, which raises exceptions when an unexpected behavior is signaled by a HTTP status code, such as when the quota is exceeded or the token is invalid, as described in the Subsections 3.3.3 and 3.3.4, respectively. The returned resource representations are Python dictionaries implementing the data models defined in APPENDIX C – OPENSTACK SDK DATA MODEL, via the normalization process documented in the Subsection 3.3.6.

There are two scenarios of syntactic differences that cause vendor lock-in in OpenStack: when the same functionality is in different APIs and heterogeneous pluggable backend mechanisms are in use.

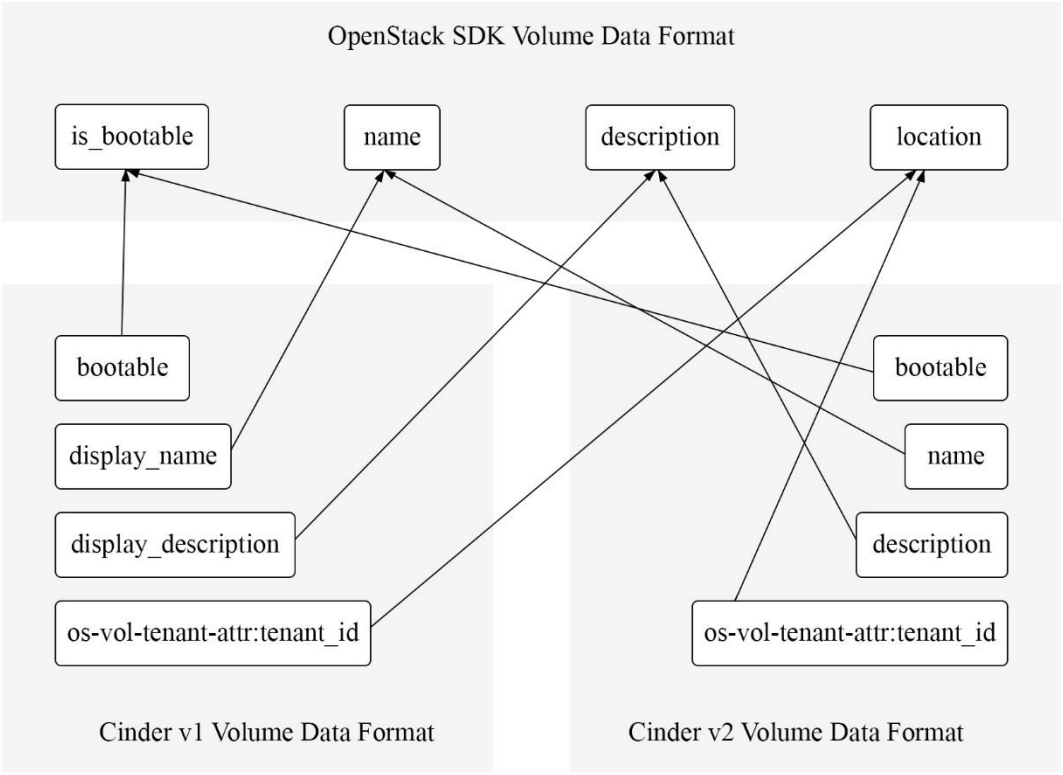
Functionality is exposed through different APIs:

Cloud A deploys the Nova Network service for networking operations. Cloud B deploys the Neutron service. As a user, how may one write an application that shows volumes in both clouds?

When the functionality is exposed through different services or by the same service but in different versions, it is implemented by different APIs, generally with the same HTTP method as its semantic is not affected, but it means multiple URL, headers and payload formats. When the URLs or headers are not equal, the request protocol is not the same. Since the input or output payloads change, the data format is affected as well.

To solve this, OpenStack SDK identifies what service and version are available in the service catalog, then proceed with the appropriate call. After getting the return from the service, it normalizes the result before returning to the user. The normalization process is illustrated in Figure 52 for some attributes of a volume in the versions 1 and 2 of Cinder, the Block Storage service.

Figure 52 - Normalizing a volume



Source: Created by the author.

Pluggable underlying mechanism:

Cloud A requires password authentication, and cloud B requires a proprietary authentication mechanism, allowing users to authenticate as in Figure 53 and Figure 54, respectively. Both return a token upon successful authentication, but each require specific REST payloads on the request. How does one get a token transparently in both clouds?

Figure 53 - Authentication via the password plugin

```
$ curl -i \
  -H "Content-Type: application/json" \
  -d '
{
  "auth": {
    "identity": {
      "methods": ["password"],
      "password": {
        "user": {
          "name": "admin",
          "domain": { "id": "default" },
          "password": "adminpwd"
        }
      }
    }
  }
}' \
  "http://localhost:5000/v3/auth/tokens" ; echo
```

Source: Created by the author.

Figure 54 - Authenticating via proprietary plugin

```
$ curl -i \
  -H "Content-Type: application/json" \
  -d '
{
  "auth": {
    "identity": {
      "methods": ["xpto"],
      "xpto": {
        "id": "0a33--"
        "sequence": "2",
        "secret": "230"
      }
    }
  }
}' \
  "http://localhost:5000/v3/auth/tokens" ; echo
```

Source: Created by the author.

OpenStack is designed to be flexible, supporting multiple vendors and technical solutions in most of its functionalities via plugins. While the semantic is preserved, different plugins may take different payloads to perform the requested operation. Since vendors are responsible for defining what plugins are available in a cloud, if the sets of plugins in different

clouds are mutually exclusive, the user would need to use multiple payload formats to communicate to multiple clouds.

If the sets of plugins are mutually exclusive, there is absolutely nothing that can be done to get around and authenticate the user seamlessly across clouds with the same arguments. However, most cloud providers support at least the password authentication plugin, which is what OpenStack SDK uses.

6.1.3 Semantic Lock-in Scenarios

The purely semantic issues found in OpenStack are related to how the cloud and the user account are configured, hence they cannot be solved by a technical workaround such as OpenStack SDK. Despite of that, there are some precautions users may consider when negotiating contracts with cloud vendors and when developing applications.

Different authorization requirements for the functionality:

Cloud A requires a user to have *member* role to upload an image, whereas cloud B requires *admin*. As a user with member role in both clouds, how may you upload an image in both A and B?

OpenStack uses Role Based Access Control (RBAC) to protect its functionalities, with the roles required to access individual APIs being configured by the cloud provider, as specified in the Subsection 2.3.6. When different providers configure access control in an incompatible manner, a user will get unauthorized errors when performing the same operations in the cloud with least privilege. In this case it is up to the user to negotiate with the cloud vendor what functionalities will be available to their account.

Cloud-wide restrictions on resources:

Cloud A sets the maximum size of an image to 512 megabytes, while cloud B sets it to 1 gigabyte. How does one upload your 700-megabyte Linux image to both clouds?

Due to nonfunctional requirements such as reducing complexity and optimizing available storage, vendors configure some options that establish upper limits for resources upon creation. When clouds have different limits for a given resource, a user may get an error when trying to create resources in the cloud with the lowest limit.

When requested to create a resource, OpenStack SDK simply makes the appropriate call to the underlying services. If different clouds have different limits for resource creation, there is nothing it can do to work around that. It is up to the user to understand what limits the cloud vendor sets and decide if they are acceptable. If not, try a different vendor.

User account-wide restrictions on resources:

A user needs to boot 20 servers, of which 10 go in cloud A, and 10 in cloud B. Their project quota in cloud A allows the booting of up to 6 servers, and their account in cloud B allows booting up to 12 servers. How do may one boot 10 servers in each cloud?

Restrictions on resource creation in a user account basis are called quotas, which define maximum resource boundaries a user may create, such as number of virtual machines that can be instantiated, as further documented in the Subsection 2.3.7. They are assigned by the vendors to users upon request. If the limits are not consistent across different clouds, the user may get errors in a cloud with lower limits when trying to perform the same create operation across clouds.

When OpenStack SDK tries to perform a create call and the user quota is not enough, it will simply raise the error to the user. In that case, the user would need to negotiate a consistent quota for booting servers across cloud vendors.

Inconsistent resource discovery:

How does one discover the latest version of your preferred Linux image in the heterogeneous clouds A and B?

Some resources are created and configured by the vendors and are cloud-wide, such as the public network, user roles and default images. The lack of standardization on what is available by default and how those cloud-wide resources are labeled disfavors programmatic discovery in a multi-cloud environment.

How can OpenStack SDK find the latest Ubuntu image available in all clouds? There is no standardization in image names across clouds, neither helpful metadata to make it possible. Image metadata is entirely vendor-defined, thus there is no way the library can understand it precisely in a multi-cloud environment.

Although users cannot fully understand the default cloud-provided resources, they can create and name their own resources. Thus, a possible solution for this issue is that users create

their own resources. In the example above, the user could upload the same image to all clouds in use, ensuring both the image contents and names are the same.

Pluggable underlying mechanism:

All users in cloud A are backed by an LDAP server which is read-only by OpenStack. Cloud B uses a read-write SQL backend. How does one write an application that updates user information in both clouds?

OpenStack supports multiple vendors and technical solutions via plugins. Plugins act as backends for the REST APIs, which are always available, regardless the plugin implementing the operation for that API or not. An error stating the functionality is not implemented may be raised, or the API call may be silently ignored. In that latter case, multiple clouds using different plugins might have inconsistent behaviors when requested to execute the same operation.

It is very common to organizations to maintain a central source of truth for authentication, such as an LDAP server, especially when there is the need to have a consistent user management across the whole organization, including its applications. OpenStack provides a mechanism to integrate LDAP servers for authentication purposes. Companies do not want, however, that an update or deletion of an OpenStack user propagate and affect all applications. In this scenario, OpenStack would have read-only access to the LDAP server.

When performing an update user call, OpenStack would be successful on cloud B and would fail on cloud A. There is nothing it can do about that, since the functionality is not supported by the vendor because of the manner the cloud was configured. Users need to understand what plugins the vendors support and whether those meet their needs.

6.1.4 Syntactic and Semantic Lock-in Scenarios

There are lock-in scenarios that involve syntactic and semantic challenges, which means disparities on protocols and data formats, as well as the interpretation of operations, some of which OpenStack SDK can work around in a sophisticated manner.

Multiple workflows for complex operations:

Booting a server with a floating IP attached to it is a functionality that involves many API calls and may happen in many manners, depending on how the cloud is configured, and

what services are available. How does one boot a server in two clouds, A and B, without needing to know what deployment and configuration choices were made by each?

Providing IaaS involves non-trivial operations at the service provider side. By supporting many vendors and technical solutions, there are multiple manners to perform such complex tasks, each one taking a different workflow involving multiple API calls. Even if the final semantic result is the same, executing such operations does not consistently use the same data formats neither have the same accurate interpretation throughout the process.

In the example of booting a server and assigning a public IP to it, the first step is to figure out what is the networking service in the cloud: Nova Network or Neutron. In this example, assume Neutron is available. The second step is to query Neutron to figure out if there is a public network to boot the server on. If there is, then a single API call to Nova, the Compute Service, may be performed requesting the virtual machine to be instantiated and be put directly in that public network. If there is not, the solution will be a two-step process: create a virtual machine with a private IP and then assign a floating IP later via NAT mechanism. In the IP assignment via NAT mechanism, first try to pass the port ID of the private IP of the server to the floating IP create call. If that is not possible, create a floating IP and then attach it to the server.

Another complex example is the upload image functionality, managed by Glance, the Image Service. There are two mechanisms for that: upload data directly to Glance via HTTP PUT, or upload the data to the Object Storage service, Swift, and then import it to Glance with an import task. Both alternatives are available in every Glance version 2 service. In some clouds, upload via PUT is disabled, and in other clouds the task import mechanism does not do anything and just ignores the requested action. More specifically in the task import path, the accepted payloads are all vendor or plugin specific, presenting the issues described in the pluggable underlying mechanism lock-in scenario.

Functionality is not provided:

A user writes an application that uses Object Storage to store large pools of data dynamically. How does the user deploy that application in both clouds A and B, given that only cloud A deploys the OpenStack Swift service?

Cloud vendors may opt to not deploy or to remove some of the OpenStack services for whatever reason, such as it is not part of their market strategy. Users would not be able to use the same functionality across multiple clouds. This was the case with VEXXHOST during the

validation of OpenStack SDK in this investigation, which did not expose the Object Store APIs in its cloud region in Santa Clara, California, as mentioned in the Section 4.4. Before choosing what cloud vendors to go with, users need to understand their service catalog to ensure all the expected functionalities are provided.

Third-party service implementation:

A user wants to manage objects on the cloud. Cloud A publishes the OpenStack Swift Object Store APIs that support containers and objects. Cloud B too, however it has decided to use a third-party gateway that uses a Ceph backend directly and implements REST APIs, claiming compatibility with the Swift API syntactic and semantic contracts. If the third-party gateway APIs cannot behave the same as the OpenStack official Object Storage APIs, how can a user manage objects transparently across clouds A and B?

This exact situation happened in the validation of OpenStack SDK in this investigation, where the Ormuco cloud was using such a Ceph Swift Gateway which had a misbehavior caught by our tests, as further explained in the Section 4.4.

Unlike the pluggable underlying mechanism, this is not supported by the OpenStack community hence does not have the quality assured by the OpenStack robust Continuous Integration ecosystem. Furthermore, these deployment choices are against the *OpenStack Primarily Produces Software* principle of OpenStack, which is further detailed in the Subsection 2.3.3.

6.1.5 Interoperability Landscape

The Table 18 summarizes the OpenStack vendor lock-in scenarios, showing which are addressed by the brokering library and which are not. Despite the fact it solves only a few issues in terms of quantity, those are the most impeditive for interoperability in OpenStack, because of the complexity generated to the user side. The issues that require user intervention are mostly related to understanding what functionalities are available in the clouds and negotiating their accounts configuration, along with workarounds when building their applications.

Table 18 - Lock-in scenarios addressing

Scenario #	Description	Supported
1	Syntactic – Functionality is exposed through different APIs	•
2	Syntactic – Pluggable underlying mechanism	N
3	Semantic – Different authorization requirements for the functionality	N
4	Semantic – Cloud-wide restrictions on resources	N
5	Semantic – User account-wide restrictions on resources	N
6	Semantic – Inconsistent resource discovery	N
7	Semantic – Pluggable underlying mechanism	N
8	Syntactic and Semantic – Multiple workflows for complex operations	•
9	Syntactic and Semantic – Functionality is not provided	N
10	Syntactic and Semantic – Third-party service implementation	N

Source: Created by the author of this document.

Analyzing these scenarios in the context of the validation documented in Chapter 4, the scenario 1 is the most present since there was an exhaustive variation of the service versions. Scenario 8 appeared in the workflow for creating a public server when the IP was automatically assigned via DHCP or NAT, transparently. Scenarios 3, 4, 5 and 7 were not observed in the validation, however they are known to exist due to the service and account configurability. Scenario 6 was present and was overcome by uploading a new image whenever it one was required. Scenario 9 happened for VEXXHOST that did not provide Object Storage functionalities in the selected cloud region. Lastly, scenario 10 appeared in Ormuco which used a third-party backend for exposing the OpenStack API definitions of the Object Storage service.

6.2 Literature Review Comparison

This section contains a throughout positioning of the OpenStack interoperability status in the state of the art. The following subsections will analyze the provider-centric, user-centric and conceptual approaches and solutions from the OpenStack's point of view, followed by the lock-in causes, consequences and warning to a situation in which brokering only does a makeup for interoperability.

6.2.1 Provider-centric Approaches

This subsection discusses the existing provider-centric standards, challenges and approach applicability to the OpenStack reality.

Unified Access Management Standards:

For the challenges in uniform access and management, OCCI, CIMI and TOSCA were the most prominent implementations found in the review. Since all three support the orchestration of compute, networking and storage resources, any of those could technically be adopted in OpenStack. A deeper analysis would need to be performed to figure out which would be the most appropriate.

Historically, both CIMI and OCCI appeared in 2010, as did OpenStack, which was created with the join of two existing projects by inheriting existing code, as documented in the Subsection 2.3.1. TOSCA was only created in 2014 when OpenStack was a very large project already, with millions of lines of code and hundreds of production clouds.

Adopting one of these standards by then seems unrealistic, since they were either emerging when OpenStack was created or appeared after a few years and were not industry-proven yet. Additionally, it would impact the fast progress of OpenStack in the first days, since persuading the whole community to agree on and adopt common models may sound unrealistic or at least hard to achieve (LOUTAS, KAMATERI, *et al.*, 2011), especially since it had a significant inherited code base already.

Another point of discussion is whether adopting one of those standards would allow OpenStack to be as flexible as it is, both as a software and as a development community, since the agreements would have to be taken in a broader community than just OpenStack, recalling the difficulty of agreement on standards.

Authentication and Authorization Standards:

X.509 certificates, SAML and OAuth are the most important standards in this aspect. Although only password authentication is currently supported via OpenStack SDK, which is enough for providing interoperability, OpenStack supports all of them.

SAML is one of the protocols supported for identity federation (OpenStack Federated Identity, 2018); X.509 certificates are known as tokenless authentication (Configuring

Keystone for Tokenless Authorization, 2018); lastly, the first version of OAuth is supported as an extension (OpenStack Identity API v3 Extensions, 2018).

Information Discovery Standards:

GLUE was the only standard in this category and allows publishing concrete data model to dictate how the infrastructure information is generated and used. OpenStack currently does not support any standard for resource naming, but it could be of interest to standardize the description of cloud-wide resources, such as default vendor-provided images, enabling the programmatic discovery by end users.

Challenges:

It was documented that vendors prefer to lock-in customers as a retention strategy, that providers want to have unique features to attract more customers and that there are numerous standards, without agreement on which one to adopt (PETCU, 2011).

On the first topic, when vendors have proprietary solutions, that retention strategy is applied into the products. In OpenStack, however, not just the software is open source, but the community, the design and the development processes are open too, meaning any company or individual contributor that agrees to the OpenStack code of conduct and shares its principles is welcome to contribute to the project, even influencing its roadmap, as further documented in the Subsection 2.3.3.

On the second topic, it is true that even in OpenStack service providers want to have unique features, but that generally is implemented by applying custom services on the top of the existing OpenStack base functionalities, such as granular billing and customized graphic user interfaces.

On the third and last topic, it was discussed while analyzing the Unified Access Management Standards previously in this subsection, where there is an agreement that the existence of multiple standards without an agreement of which is best obscure their widespread adoption and pose the danger of having standards that are not interoperable between themselves.

Applicability:

Disregarding the potential impacts on the development productivity and difficulty of agreement on standard implementations aforementioned, the adoption of standards in the early stages of OpenStack would have worked, although there was not a standard that was widely known and industry-proven by then, and it is still unclear which one is the best, with studies even mentioning the use of a combination of them (GARCÍA, CASTILLO and FERNÁNDEZ, 2016) (TOOSI, CALHEIROS and BUYYA, 2014).

At this stage, where OpenStack is almost 10 years since its conception, its *post-facto* standards is adopted in hundreds of productions clouds and persuading the community to change it is impracticable, since there are thousands or even millions of users relying on it, without mentioning all the corner cases supported and the implementation effort involving several million lines of code. Additionally, redesigning OpenStack on *de jure* standards would need to consider a tradeoff between feature completeness and the achievable level of interoperability, because not all features would be supported. Lastly, there is no benefit on adopting a *de jure* standard that is not widely adopted, hence the reality is that OpenStack will continue with its native *post-facto* standard.

What can be done, however, it the adoption of *de jure* standards via a higher-level brokering service that may even make use of OpenStack SDK to benefit of its native interoperability capabilities.

The example of standardization effort in OpenStack is the RefStack initiative documented in the Subsection 1.3.1, which reinforces the difficulty of agreement on standard approaches for providing functionalities even within the OpenStack community, especially after being widely adopted.

6.2.2 User-centric Approaches

This subsection discusses the existing user-centric alternative implementations to OpenStack SDK, the challenges and the applicability of this approach to the OpenStack reality.

Alternative Implementations:

Apache Libcloud and Delta Cloud were the two main broker implementations found in the review. Both provide functionalities for orchestrating resources similarly to OpenStack

SDK. With Delta Cloud discontinued, Libcloud is the most prominent alternative. However, for the same reasons why the adoption of *de jure* standards is an impracticable approach for the OpenStack APIs, updating OpenStack SDK to the Libcloud definitions is not realistically achievable because the user base and since it models resources in a more compatible manner with the OpenStack REST APIs, for example a server is called *node* in Libcloud.

What can be done is to update the Libcloud's support for OpenStack with calls to OpenStack SDK. This integration might be straightforward, especially because both brokering solutions are written in Python.

Challenges:

The immediate challenge documented was that a proprietary API is subject to change without prior advice by the original vendor (SILVA, ROSE and CALINESCU, 2013). OpenStack has open APIs, and this is not a problem at all since the development of OpenStack SDK is completely integrated with the rest of the services, and any backwards incompatible changes would near immediately be noticed by the Continuous Integration system that runs thousands of tests for OpenStack SDK.

The review also documented that client-server communication is generally specific to each provider and not available for every programming language. Both are true for OpenStack: the former applies to the reality because of the flexibility and configurability of the software, generating the lock-in scenarios documented in the Section 6.1; the latter is currently one of the major limitations of OpenStack SDK, which is only available in Python.

Moreover, this approach was documented as difficulty to maintain and presenting exponential growth, with supported functionalities either very limited to a subset, which may lead to a significant loss of functionalities for some clouds or resulting in inconsistent behavior as some capabilities might not be available in all clouds. As OpenStack SDK aims to be always backward compatible, challenges in maintaining the code base certainly exist, but the growth of cases is not exponential as the community thrives to support interoperability. About the supported functionalities, it positions itself in the second option, implementing several functionalities with inconsistent behavior across cloud, such as the Object Store service not being available in one of the clouds tested in the validation.

Furthermore, it documents the conclusion that libraries can provide basic functionality but are not easily maintained and do not scale well nor provide much transparency (PARÁK and ZDENěK, 2014). This is not true for OpenStack SDK, as it is specific for OpenStack and

is not trying to abstract multiple IaaS platforms, it benefits of disposing of a compatible resource representation and concepts that are not too abstract, sometimes providing advanced functionality such as Database as a Service. Its development is fully backed by the community, providing integrated maintenance in a transparent manner, as the code is open source. It scales well as it is used by the OpenStack functional testing workflows to set up thousands of servers every day in multiple clouds.

The last concern is in terms of performance and data exchange, as was observed with Delta Cloud. By providing such a flexibility and transparency, it is expected that OpenStack SDK does not use the minimum number of API calls in every functionality, providing considerable additions to execution time and exchange data. For this purpose, further performance evaluation needs to be designed and performed.

Applicability:

With the aforementioned comprehension about how the challenges observed in the literature are addressed, OpenStack SDK represents the most realistic approach to interoperability. Despite the fact it is considered as a complicated solution, it has proven to work and will always work as it is fully developed by the community.

It is even being adopted by third-party projects and communities, such as Ansible via the implementation of OpenStack modules (Ansible OpenStack Cloud Modules, 2018), forming another brokering level via management tools, which are considered enablers for multicloud environments in the literature (VANBRABANT and JOOSEN, 2014).

6.2.3 Conceptual Approaches

The conceptual approach found in the review was proposing an implementation where each provider's API is modeled by an ontology. Equivalences between these ontologies are modeled by rules, which allow converting calls between providers' specific APIs (EJARQUE, ÁLVAREZ, *et al.*, 2011).

Combined with OpenStack SDK, this proposal is very powerful, allowing the current support for OpenStack clouds only to be extended to other platforms, such as Amazon Web Services, while keeping the same contracts already defined, keeping backwards compatibility with the existing support.

6.2.4 Lock-in Causes

The review documents that the lock-in cause is the lack of a unified cloud description standards covering technical, operation, business and semantic aspects, since the existing descriptions utilize different languages, standards, ontologies and models (GHAZOUANI and SLIMANI, 2017). In addition, the heterogeneity between cloud interfaces, which implement the technologies and semantics of a provider, is a major barrier to cloud portability and interoperability (SILVA, ROSE and CALINESCU, 2013).

While focusing in the technical aspects, OpenStack has a consistent service description across its deployments, using the same language, standards, ontologies and models, hence the first finding does not apply to intra-platform interoperability. Regarding the second finding, that is exactly what generates the vendor lock-in scenarios documented in this investigation: the heterogeneity of cloud interfaces.

The reason for such heterogeneities being present within different installations of the same platform is not found in the review. Observing the OpenStack characteristics and its lock-in scenarios, the main causes for intra-platform lock-in is the high configurability of services, flexibility to provide a large range of use cases and API changes.

The review mentioned some studies considering that OpenStack, as a *post-facto* solution, is interoperable between installations (TOOSI, CALHEIROS and BUYYA, 2014) (GARCÍA, CASTILLO and FERNÁNDEZ, 2016). Such a misunderstanding of lock-in proneness can be justified when one looks at a proprietary product of a single provider, such as Amazon, which generally has the power and control to make its deployments interoperable. However, as discussed through all this document, it is not the case for OpenStack, as there is no control on the services, versions or configuration of the hundreds of existing clouds. As mentioned in the previous subsections, standardization efforts such as RefStack cannot reach agreement or progress in the broad community.

6.2.5 Lock-in Consequences

The review highlights the lack of acknowledgement and understanding of the vendor lock-in issues when procuring services from vendors, which poses a significant barrier to obscure the potential effects on organizations (OPARA-MARTINS, SAHANDI and TIAN, 2016). Additionally, it documents the lack of studies investigating the consequences of lock-in in the cloud, such as limitations on cloud usability or migration of users. The example presented,

that correlates API change and fault proneness to the success of Android applications, cannot be reproduced in OpenStack, as there are no criteria to measure or estimate the success of OpenStack clouds, such as public user ratings on the provided services.

6.2.6 Vendor Lock-in Makeup

The review documented by creating many standards that are not to interoperable between them, there is the danger of creating lock-in at another level when different companies decide to go with different standards (LOUTAS, KAMATERI, *et al.*, 2011). The same principle applies to brokering solutions, since they also define syntactic and semantics of the provided functionalities.

With OpenStack SDK, if the function parameters were renamed or removed between versions, or new required parameters were added, this would break users of older versions upon upgrade. Similarly, if the responses change in a way that renames or removes the resource fields, users might break. If that happens, users would be technically locked-in to specific versions of the library, requiring an effort to update their orchestration scripts. To avoid this issue, the library adopts a policy of always being backwards compatible, never allowing any of the interface changes aforementioned.

If a standard or brokering solution permits users to get lock-in to themselves, they are not solving the problem, but only adding another level of indirection by doing makeup on the interoperability issues, because even if the lock-in issues are relieved, they are not completely mitigated.

7 CONCLUSION AND FINAL REMARKS

With billions of internet users exchanging trillions of gigabytes in network traffic per year, it is clear how technology is changing the manner human society interacts with the universe. Health, news, education, communication and automation are examples of areas advancing exponentially year over year thanks to technology. Consequently, four out of the top five most valuable companies in the planet are in the technology sector: Apple, Alphabet, Microsoft and Facebook (FORTUNE 500, 2018).

Cloud computing plays a key role in this revolution, placing itself as the biggest technological paradigm shift in the history of humanity, which presents a distinguished market opportunity for numerous companies, with IaaS having the highest CAGR between the cloud delivery models. As the computational infrastructure needs grow and the market heterogenize, interoperability becomes a key requirement for enabling optimal multicloud scenarios.

In such a breathtaking scenario, this work investigated interoperability in the largest open source IaaS platform nowadays, which counts with numerous deployments worldwide. First, Chapter 1 provided the context, motivation and objectives of this research, while Chapter 2 detailed the information on cloud, interoperability and OpenStack necessary to fully comprehend this study; Chapters 3 and 4 documented and validated the brokering solution of OpenStack, while Chapters 5 and 6 performed a literature review and positioned the OpenStack interoperability in the literature.

This chapter recapitulates the findings of this investigation on OpenStack interoperability in Section 7.1, highlighting its academic and industry contributions in Sections 7.2 and 7.3, followed by suggestion for future work in Section 7.4 and recommendations for users that plan on moving workloads to the cloud in Section 7.5. Lastly, Section 7.6 acknowledges the OpenStack community and the organizations whose products were used in the validation, without them this work would not have been possible.

7.1 OpenStack Interoperability Landscape

The OpenStack mission to produce an open source cloud computing platform that is interoperable between deployments, as documented in the Subsection 2.3.2, is only fully met with OpenStack SDK, because of its interoperability attribute.

With a realistic approach, OpenStack SDK provides an abstraction layer that considers the numerous configuration options and high flexibility of clouds, working around the most

impeditive vendor lock-in scenarios documented in the Section 6.1. The scenarios it cannot work around are either details to be negotiated when contracting service providers or further improvements to the use of standards in OpenStack, especially in what refers to the discovery of resources.

The validation of the library considered many arbitrary configurations that represent various real production deployments, and despite its limitations on the numbers of resources and attributes tested, it verified the effectiveness of OpenStack SDK on communicating with multiple heterogeneous clouds.

Before proving interoperability, OpenStack was in an uncomfortable position where competitors could disfavor the benefits of adopting an open source *post-facto* standard because it would not provide an acceptable level of interoperability either. For that same reason, OpenStack needs to continue in this path to eventually achieve full interoperability in the future. Despite the fact of not being able to make existing deployments interoperable, both the RefStack and the control of API changes initiatives, as explained in the Section 1.2, will help on having future releases less prone to vendor lock-in, and consequently, the OpenStack deployments as newer versions are adopted. OpenStack SDK will always have its importance, guaranteeing interoperation despite of the native support by the underlying services, while at the same time allowing integration with other communities, such as Ansible. As an improvement, it can certainly support other programming languages than just Python, reaching a wider user community.

Going beyond orchestration interoperability, it is important for OpenStack to implement portability between its deployments, for example by supporting the Open Virtualization Format standard (DISTRIBUTED MANAGEMENT TASK FORCE, 2018) in its images, enabling customers to move server images between not just OpenStack clouds, but any cloud that supports that standard, at their convenience.

7.2 Academic Contributions

As a *post-facto* standards, some studies believe OpenStack had native interoperability support (TOOSI, CALHEIROS and BUYYA, 2014) (GARCÍA, CASTILLO and FERNÁNDEZ, 2016). As extensively discussed in this document, this is not true. The first contribution to the academy is therefore to catch the attention to the fact that intra-platform vendor lock-in exist, especially when the platform and vendor relationship is not one-to-one. When the relationship is one-to-one, such as Amazon which has a proprietary platform for its

services, not providing interoperability between different regions or deployments do not make sense, and it is perfectly achievable as a single company controls the management and upgrade of deployments. When multiple companies use the same platform, which is especially the case for open source software, they configure, maintain and upgrade the services at their own pace, which sometimes result in non-interoperable solutions.

Along with the report of the existence of intra-platform lock-in, an analysis of the brokering solution was performed, resulting in a detailed classification of the observed scenarios, which opens opportunity for future work and share experiences with other organizations or communities developing IaaS platforms. The validation of the library was performed in a very transparent manner, where all the scripts, decisions and results are documented, allowing the reproducibility of the experiment.

Another finding of this investigation is that besides the API changes, as documented in the literature, the flexibility in use cases and configurability of the platform are the root cause for intra-platform lock-in. Those are equivalent to proprietary solutions presenting unique features, but in this case as an open source project, companies share a common base implementation and customize it at the configuration level and aggregated services to differentiate in the market. In terms of consequences, no study was found correlating the lack of interoperability to the success of platforms, which opens precedents for future research.

On the power of open source software, it is important to comprehend why open source software is crucial to the development of the next generation of platforms and frameworks, with the example from OpenStack. It is a single platform that allows companies and individuals to collaborate towards a cutting-edge solution to address shared perceptions of market needs. Although there are competitors in the same ecosystem, this does not necessarily compromise conflicting market strategies, as there is enough market to multiple companies in the world, and the demand is growing year over year. A single company cannot offer optimal support in all regions of the globe. Latency, customer success, billing currency and taxes, personalized service and other criteria typically have higher success rates when customers work with local providers.

7.3 Contributions for Industry

The industry, most notably the OpenStack platform, benefited from this investigation in several aspects:

- An analysis and enumeration of lock-in scenarios allow the community to acknowledge its current interoperability level and how and where it can improve;
- The comparison with the state of the art permits the community to discover how its solution is positioned in the literature;
- The validation of its brokering solution reinforces its position in the market as an interoperable platform;
- Extensive contributions to the code base allowed the project to present more consistency across features, the report and fix of failures and documentation improvements;
- Suggestions for future work can be analyzed and potentially adopted by the community in the project roadmap.

7.4 Future Work

Supporting multiple programming languages is one of the main improvement points for OpenStack SDK. Support for gRPC APIs (GRPC, 2018) enables the library's code to be compiled into multiple languages, generating other native client libraries. This initiative was proposed to OpenStack as the Oaktree project (Oaktree Github Repository, 2016), but it never got enough contributions to evolve, given the limited number of contributors who focused on developing OpenStack SDK, which is a prerequisite and had many features in the roadmap.

It is possible to combine the gRPC layer with other proposals and broaden not just the support for programming languages, but the infrastructure platforms as well. The gRPC layer could be combined with the rule-based approach for interoperability mentioned in the Subsection 6.2.3, allowing users to use the OpenStack SDK's interfaces, on multiple programming languages, to talk to multiple heterogeneous platforms.

Although never raised as an issue by users, it is important to run a performance analysis of the library, avoiding to have time and data exchange overheads as impeditive attributes for some use cases, as happened to the Delta Cloud project (MEIRELES and MALHEIRO, 2014). Such an analysis would enable the project to profile the code and improve performance as well.

RefStack currently has a certification program that checks whether clouds are interoperable according to its definitions of valid OpenStack cloud configurations. The same idea can be extended to OpenStack SDK, where functional tests could be run against an arbitrary cloud, with having the possibility of publishing the results publicly in the OpenStack marketplace. This would ultimately allow users to easily determine if a cloud is interoperable

or not, influencing decision makers on what provider to pick, eventually pushing more and more providers to be fully compatible with OpenStack SDK.

Lastly, OpenStack SDK enables upper abstraction layers and services to safely rely on its functionalities and capacity to communicate to multiple clouds, reducing considerably the efforts to implement multicloud services, such as a proposal for a selection algorithm that exploits deployment knowledge offered via reasoning over previous application execution histories to suggest the best possible allocation decisions (KRITIKOS, MAGOUTIS and PLEXOUSAKIS, 2016).

7.5 Recommendations

Given the impacts of vendor lock-in, the overall recommendation is to select vendors who implement standards and prove interoperation with other platforms. Taking enough time to analyze providers is an investment that will save time, and possibly money, afterwards. Therefore, making well-informed decisions before selecting vendors and signing cloud contracts is crucial to avoid vendor lock-in (OPARA-MARTINS, SAHANDI and TIAN, 2016).

In the analysis phase, identifying the second preferred provider and being able to run functional interoperability tests with it is very powerful. The downside, however, is that writing a complete set of tests is not feasible for some users. In that case, a *proof of concept* might be enough to have a feeling of how complex the moving process can be. If using multiple OpenStack clouds, testing and using the cloud itself can be done with OpenStack SDK, benefiting from its vendor lock-in abstraction functionalities. While defining the test cases, users can benefit from the list of scenarios from Section 6.1 and identify what are the corner cases according to their needs. For example, when uploading images, try to upload large images on all clouds to make sure they all accept images up to that specific size, as an instance of the semantic scenario “cloud-wide restrictions on resources”. After the testing phase, moving testing applications before production workloads is always a recommended practice, since if there is any problem in the early phases, that will not affect any production system and consequently customers. The sooner users notice vendor lock-in and decide to migrate, the easier it will be.

Finally, the development process influences the user experience in the cloud as well, not only for the achievable level of automation allowing application scale out, but because of the effort to escape vendor lock-in. To illustrate that, APPENDIX M – PETS AND CATTLE IN THE HOTEL CALIFORNIA compares the cloud market with the Hotel California

(FELDER, HENLEY and FREY, 1977), where exiting providers is not as easy as checking-in. Using the *Pets vs Cattle* (BIAS, 2012) metaphor, it explains how designing and implementing applications that can scale out, like *cattle*, eases the migration process, as opposed to developing applications that can only scale up, raised with love like *pets*. In the former, the infrastructure can be cloned and remaining stateful resources such as databases can be ported, only requiring updates in the orchestration scripts to communicate with the new provider's APIs. In the case of moving between OpenStack clouds, not even the orchestration scripts would need to be updated because the platform is now interoperable with OpenStack SDK.

7.6 Acknowledgments

The author would like to thank the OpenStack community for the opportunity of collaborating to the ecosystem, especially to the OpenStack SDK team for being always available to discuss and find the best solutions for the interoperability challenges of OpenStack. Without the community neither OpenStack SDK nor this work would have been possible.

Additionally, the author would like to thank the 5 organizations that allowed the validation tests to be run and the results to be publicly published in this dissertation, benefiting not only the OpenStack community but the wider open source and cloud communities.

REFERENCES

AMAZON WEB SERVICES. Amazon Elastic Compute Cloud. **AWS Documentation**, 2018. Available at: <https://docs.aws.amazon.com/AWSEC2/latest/APIReference/Welcome.html>. Accessed on: 18 nov. 2018.

AMAZON WEB SERVICES. VMWare Cloud on AWS. **Amazon Web Services (AWS) - Cloud Computing Services**, 2018. Available at: <https://aws.amazon.com/vmware/>. Accessed on: 18 nov. 2018.

ANSIBLE. Ansible is Simple IT Automation. **Ansible**, 2018. Available at: <https://www.ansible.com/>. Accessed on: 18 nov. 2018.

ANSIBLE OpenStack Cloud Modules. **Ansible Documentation**, 2018. Available at: <https://docs.ansible.com/ansible/latest/modules/list_of_cloud_modules.html#openstack>. Accessed on: 18 nov. 2018.

AODH Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/aodh/latest/>. Accessed on: 18 nov. 2018.

APACHE License, version 2.0. **The Apache Software Foundation**, 2004. Available at: <https://www.apache.org/licenses/LICENSE-2.0>. Accessed on: 18 nov. 2018.

APACHE SOFTWARE FOUNDATION. Apache Delta Cloud. **Apache Delta Cloud Webpage**, 2015. Available at: <http://deltacloud.apache.org/>. Accessed on: 18 nov. 2018.

APACHE SOFTWARE FOUNDATION. Apache Libcloud. **Apache Libcloud Website**, 2018. Available at: <http://libcloud.apache.org/>. Accessed on: 18 nov. 2018.

BARBICAN Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/barbican/latest/>. Accessed on: 18 nov. 2018.

BERNSTEIN, D. et al. Blueprint for the Intercloud – Protocols and Formats for Cloud Computing Interoperability. **Fourth International Conference on Internet and Web Applications and Services**, p. 328-336, 2009.

BIAS, R. Architectures for open and scalable clouds. **LinkedIn SlideShare**, 2012. Available at: <https://www.slideshare.net/randybias/architectures-for-open-and-scalable-clouds>. Accessed on: 23 nov. 2018.

BIFROST Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/bifrost/latest/>. Accessed on: 18 nov. 2018.

BLAZAR Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/blazar/latest/>. Accessed on: 18 nov. 2018.

CATALYST Cloud. **Catalyst Cloud Website**, 2018. Available at: <https://catalystcloud.nz/>. Accessed on: 18 nov. 2018.

CEILOMETER Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/ceilometer/latest/>. Accessed on: 18 nov. 2018.

CEPH Object Gateway Swift API. **Ceph Documentation**, 2018. Available at: <http://docs.ceph.com/docs/master/radosgw/swift/>. Accessed on: 18 nov. 2018.

CHANGE 60050. **OpenStack Gerrit Code Review**, 2018. Available at: <https://review.openstack.org/#/c/600050/>. Accessed on: 18 nov. 2018.

CHANGE 611868. **OpenStack Gerrit Code Review**, 2018. Available at: <https://review.openstack.org/#/c/611868/>. Accessed on: 18 nov. 2018.

CHANGE 611869. **OpenStack Gerrit Code Review**, 2018. Available at: <https://review.openstack.org/#/c/611869/>. Accessed on: 18 nov. 2018.

CHEF OpenStack Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/openstack-chef/latest/>. Accessed on: 18 nov. 2018.

CHEN, D.; DOUMEINGTS, G. European initiatives to develop interoperability of enterprise applications—basic concepts, framework and roadmap. **Annual Reviews in Control 27**, p. 153-162, set. 2003.

CINDER Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/cinder/latest/>. Accessed on: 18 nov. 2018.

CIRROS. **CirrOS in Launchpad**, 2018. Available at: <https://launchpad.net/cirros>. Accessed on: 18 nov. 2018.

CIRROS Download Index. **CirrOS Download**, 2017. Available at: <http://download.cirros-cloud.net/>. Accessed on: 18 nov. 2018.

CLOUDKITTY Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/cloudkitty/latest/>. Accessed on: 18 nov. 2018.

CMWG. CMWG - Cloud Management Working Group. **CMWG Website**, 2018. Available at: <https://www.dmtf.org/standards/cmwg/>. Accessed on: 18 nov. 2018.

COHEN, R. Cloud Computing at the Hotel California,Check-in and Never Leave! **Forbes - Reuven Cohen - The Digital Provocateur**, 2013. Available at: <https://www.forbes.com/sites/reuvencohen/2013/05/02/cloud-computing-at-the-hotel-california-check-in-and-never-leave/>. Accessed on: 23 nov. 2018.

CONFIGURING Keystone for Tokenless Authorization. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/keystone/latest/advanced-topics/configure_tokenless_x509.html>. Accessed on: 18 nov. 2018.

CONGRESS Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/congress/latest/>. Accessed on: 18 nov. 2018.

CYBORG Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/cyborg/latest/>. Accessed on: 18 nov. 2018.

DESIGNATE Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/designate/latest/>. Accessed on: 18 nov. 2018.

DEVELOPER'S Guide. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/infra/manual/developers.html>. Accessed on: 18 nov. 2018.

DEVSTACK Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/devstack/latest/>. Accessed on: 18 nov. 2018.

DISTRIBUTED MANAGEMENT TASK FORCE. Open Virtualization Format. **Open Virtualization Format Website**, 2018. Available at: <https://www.dmtf.org/standards/ovf>. Accessed on: 18 nov. 2018.

EJARQUE, J. et al. A Rule-based Approach for Infrastructure Providers' Interoperability. **Third IEEE International Conference on Coud Computing Technology and Science**, p. 272-279, 2011.

ENTER Cloud Suite Cloud. **Enter Cloud Suite Website - European, Open Source Cloud Provider**, 2015. Available at: <https://www.entercloudsuite.com/en/>. Accessed on: 18 nov. 2015.

FELDER, D.; HENLEY, D.; FREY, G. **Hotel California**. [S.l.]: Bill Szymczyk, 1977.

FLAKE8 Documentation. **Flake8:** Your Tool For Style Guide Enforcement, 2016. Available at: <http://flake8.pycqa.org/en/latest/>. Accessed on: 18 nov. 2018.

FORTUNE 500. Fortune 500 10 Most Valuable Companies, 2018. Available at: <http://fortune.com/2018/05/21/fortune-500-most-valuable-companies-2018/>. Accessed on: 18 nov. 2018.

FREEZER Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/freezer/latest/>. Accessed on: 18 nov. 2018.

FUXI Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/fuxi/latest/>. Accessed on: 18 nov. 2018.

GARCÍA, A. L.; CASTILLO, E. F. D.; FERNÁNDEZ, P. O. Standards for enabling heterogeneous IaaS cloud federations. **Computer Standards & Interfaces**, 2016.

GARTNER. Gartner Says Worldwide IaaS Public Cloud Services Market Grew 31 Percent in 2016. **Gartner Website**, 2017a. Available at: <https://www.gartner.com/newsroom/id/3808563>. Accessed on: 18 nov. 2018.

GARTNER. Gartner Says a Massive Shift to Hybrid Infrastructure Services Is Underway. **Gartner Website**, 2017b. Available at: <https://www.gartner.com/newsroom/id/3666917>. Accessed on: 18 nov. 2018.

GARTNER. Gartner Forecasts Worldwide Public Cloud Revenue to Grow 21.4 Percent in 2018. **Gartner Newsroom Website**, 2018. Available at: <https://www.gartner.com/newsroom/id/3871416>. Accessed on: 18 nov. 2018.

GHAZOUANI, S.; SLIMANI, Y. A survey on cloud service description. **Journal of Network and Computer Applications** , p. 21, abr. 2017.

GLANCE Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/glance/latest/>. Accessed on: 18 nov. 2018.

GRENADE Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/grenade/latest/>. Accessed on: 18 nov. 2018.

GRPC. **GRPC Website**, 2018. Available at: <https://grpc.io>. Accessed on: 18 nov. 2018.

HEAT Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/heat/latest/>. Accessed on: 18 nov. 2018.

HELM - The Kubernetes Package Manager. **Helm**, 2018. Available at: <https://helm.sh/>. Accessed on: 18 nov. 2018.

HORIZON Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/horizon/latest/>. Accessed on: 18 nov. 2018.

INTRODUCTION: A Bit of OpenStack History. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/project-team-guide/introduction.html>. Accessed on: 18 nov. 2018.

IRONIC Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/ironic/latest/>. Accessed on: 18 nov. 2018.

JMESPATH Documentation. **JMESPath Website**, 2015. Available at: <http://jmespath.org/>. Accessed on: 18 nov. 2018.

JUJUCHARMS. **Juju**, 2018. Available at: <https://jujucharms.com/>. Accessed on: 18 nov. 2018.

KARBOR Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/karbor/latest/>. Accessed on: 18 nov. 2018.

KAUSE, T.; PEYLO, M. RFC 6712 - Internet X.509 Public Key Infrastructure -- HTTP Transfer. **RFC 6712 Website**, 2012. Available at: <https://tools.ietf.org/html/rfc6712>. Accessed on: 23 nov. 2018.

KEYSTONE Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/keystone/latest/>. Accessed on: 18 nov. 2018.

KOLLA-ANSIBLE Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/kolla-ansible/latest/>. Accessed on: 18 nov. 2018.

KRITIKOS, K.; MAGOUTIS, K.; PLEXOUSAKIS, D. Towards Knowledge-Based Assisted IaaS Selection. **8th International Conference on Cloud Computing Technology and Science**, p. 431-439, 2016.

KURYR Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/kuryr/latest/>. Accessed on: 18 nov. 2018.

KURZE, T. et al. Cloud Federation. **The Second International Conference on Cloud Computing, GRIDs, and Virtualization**, p. 32-38, 2011.

LAL, P. Understanding the impact of cloud-based services adoption on organizational flexibility: An exploratory study. **Journal of Enterprise Information Managemen**, v. 29, p. 566-588, 2016. ISSN 4.

LEIBA, B. OAuth Web Authorization Protocol. **IEEE INTERNET COMPUTING**, v. 16, n. 1, p. 74-77, 2012.

LINARES-VÁSQUEZ, M. et al. API Change and Fault Proneness: A Threat to the Success of Android Apps. **Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering**, p. 477-487, 2013.

LOUTAS, N. et al. Cloud computing interoperability: the state of play. **Third IEEE International Conference on Coud Computing Technology and Science**, p. 752-757, 2011.

MAGNUM Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/magnum/latest/>. Accessed on: 18 nov. 2018.

MANILA Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/manila/latest/>. Accessed on: 18 nov. 2018.

MASAKARI Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/masakari/latest/>. Accessed on: 18 nov. 2018.

MEIRELES, F.; MALHEIRO, B. Integrated Management of IaaS Resources. **Euro-Par 2014: Parallel Processing Workshops**, p. 73-84, 2014.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. NIST. [S.l.], p. 1-3. 2011.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. National Institute of Standards and Technology - U.S. Departmento of Commerce. [S.l.], p. 7. 2011.

MICROSOFT. What is Azure Stack. **Microsoft Azure Cloud Computing Platform & Services**, 2018. Available at: <https://azure.microsoft.com/en-us/overview/azure-stack/>. Accessed on: 18 nov. 2018.

MISCELLANEOUS operating system interfaces. **Python Documentation**, 2018. Available at: <https://docs.python.org/3/library/os.html>. Accessed on: 18 nov. 2018.

MISTRAL Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/mistral/latest/>. Accessed on: 18 nov. 2018.

MONASCA Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/monasca/latest/>. Accessed on: 18 nov. 2018.

MUNCH Git Code Repository. **Github Munch Webpage**, 2018. Available at: <https://github.com/Infinidat/munch/>. Accessed on: 18 nov. 2018.

MURANO Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/murano/latest/>. Accessed on: 18 nov. 2018.

NEBULA Cloud Computing Platform. **NASA**, 2012. Available at: <https://www.nasa.gov/open/nebula.html>. Accessed on: 18 nov. 2018.

NEUTRON Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/neutron/latest/>. Accessed on: 18 nov. 2018.

NIST ROADMAP WORKING GROUP. **NIST Cloud Computing Standards Roadmap**. National Institute of Standards and Technology - U.S. Department of Commerce. [S.l.], p. 108. 2013.

NODEPOOL Documentation. **Zuul Nodepool Webpage**, 2018. Available at: <https://zuul-ci.org/docs/nodepool/>. Accessed on: 18 nov. 2018.

NOSE Documentation. **nose Website**, 2018. Available at: <https://nose.readthedocs.io/en/latest/>. Accessed on: 18 nov. 2018.

NOSE Package. **PyPI - the Python Package Index**, 2018. Available at: <https://pypi.org/project/nose/>. Accessed on: 18 nov. 2018.

NOVA API Reference. **OpenStack Docs**, 2018. Available at: <https://developer.openstack.org/api-ref/compute/>. Accessed on: 18 nov. 2018.

NOVA Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/nova/latest/>. Accessed on: 18 nov. 2018.

OAKTREE Github Repository. **Oaktree Github Repository**, 2016. Available at: <https://github.com/openstack/oaktree>. Accessed on: 18 nov. 2018.

OASIS. TOSCA - Topology and Orchestration Specification for Cloud Applications. **TOSCA Website**, 2013. Available at: <https://www.oasis-open.org/committees/tosca/>. Accessed on: 18 nov. 2018.

OASIS. OASIS. **OASIS Website**, 2018. Available at: <https://www.oasis-open.org/>. Accessed on: 18 nov. 2018.

OASIS SECURITY SERVICES TECHNICAL COMMITTEE. SAML. **SAML Webpage**, 2005. Available at: <https://www.oasis-open.org/committees/security/>. Accessed on: 18 nov. 2018.

OCCI WORKING GROUP. Open Cloud Computing Interface. **Open Cloud Computing Interface Website**, 2016. Available at: <http://occi-wg.org/>. Accessed on: 18 nov. 2018.

OCTAVIA Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/octavia/latest/>. Accessed on: 18 nov. 2018.

OPARA-MARTINS, J.; SAHANDI, R.; TIAN, F. Critical Review of Vendor Lock-in and Its Impact on Adoption of Cloud Computing. **International Conference on Information Society**, p. 92-97, 2014.

OPARA-MARTINS, J.; SAHANDI, R.; TIAN, F. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. **Journal of Cloud Computing: Advances, Systems and Applications**, p. 18, 2016.

OPEN GRID FORUM. OGF - Open Grid Forum. **OGF Website**, 2018. Available at: <https://www.ogf.org/>. Accessed on: 18 nov. 2018.

OPENSTACK API SPECIAL INTEREST GROUP. Ensuring API Interoperability. **OpenStack Specs**, 2018. Available at: <https://specs.openstack.org/openstack/api-wg/guidelines/api_interoperability.html>. Accessed on: 18 nov. 2018.

OPENSTACK Charms Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/charm-guide/latest/>. Accessed on: 18 nov. 2018.

OPENSTACK EC2 API Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/ec2-api/latest/>. Accessed on: 18 nov. 2018.

OPENSTACK Federated Identity. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/keystone/latest/advanced-topics/federation/federated_identity.html>. Accessed on: 18 nov. 2018.

OPENSTACK Gerrit Code Review. **OpenStack Gerrit Website**, 2018. Available at: <https://review.openstack.org>. Accessed on: 18 nov. 2018.

OPENSTACK Glossary. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/glossary/common/glossary.html>. Accessed on: 18 nov. 2018.

OPENSTACK Guiding Principles. **OpenStack Governance**, 2018. Available at: <https://governance.openstack.org/tc/reference/principles.html>. Accessed on: 18 nov. 2018.

OPENSTACK Identity API v3 Extensions. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/keystone/latest/advanced-topics/configure_tokenless_x509.html>. Accessed on: 18 nov. 2018.

OPENSTACK Mission Amendment. **OpenStack Governance**, 2016. Available at: <https://governance.openstack.org/tc/resolutions/20160217-mission-amendment.html>. Accessed on: 18 nov. 2018.

OPENSTACK SDK Connection Object. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/openstacksdk/latest/user/connection.html>. Accessed on: 18 nov. 2018.

OPENSTACK SDK Data Model. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/openstacksdk/latest/user/model.html>. Accessed on: 18 nov. 2018.

OPENSTACK SDK Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/openstacksdk/latest/>. Accessed on: 18 nov. 2018.

OPENSTACK SDK package. **PyPI - the Python Package Index**, 2018. Available at: <https://pypi.org/project/openstacksdk/>. Accessed on: 18 nov. 2018.

OPENSTACK SDK StoryBoard. **OpenStack StoryBoard Website**, 2018. Available at: <https://storyboard.openstack.org/#!/project/972/>. Accessed on: 18 nov. 2018.

OPENSTACK Wiki Main Page. **OpenStack Wiki**, 2018. Available at: <https://wiki.openstack.org/wiki/Main_Page>. Accessed on: 18 nov. 2018.

OPENSTACK-ANSIBLE Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/openstack-ansible/latest/>. Accessed on: 18 nov. 2018.

OPENSTACKCLIENT Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/python-openstackclient/latest/>. Accessed on: 18 nov. 2018.

OPENSTACKCLOUD Code. **OpenStack SDK Github Repository**, 2018. Available at: <https://github.com/openstack/openstacksdk/blob/master/openstack/cloud/openstackcloud.py>. Accessed on: 18 nov. 2018.

OPENSTACK-HELM Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/openstack-helm/latest/>. Accessed on: 18 nov. 2018.

ORMUCO Cloud. **Ormuco Website - Empowering your Business with Intelligent Software**, 2018. Available at: <https://ormuco.com/>. Accessed on: 18 nov. 2018.

OS-CLIENT-CONFIG Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/os-client-config/latest/>. Accessed on: 18 nov. 2018.

OUTREACHY. Outreachy - Internships Supporting Diversity in Tech. **Outreachy Website**, 2018. Available at: <https://www.outreachy.org/>. Accessed on: 18 nov. 2018.

PANKO Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/panko/latest/>. Accessed on: 18 nov. 2018.

PARÁK, B.; ZDENěK, Š. Challenges in Achieving IaaS Cloud Interoperability across Multiple Cloud Management Frameworks. **IEEE/ACM 7th International Conference on Utility and Cloud Computing**, p. 404-411, 2014.

PEP 8 - Style Guide for Python Code. **Python.org**, 2013. Available at: <https://www.python.org/dev/peps/pep-0008/>. Accessed on: 18 nov. 2018.

PETCU, D. Portability and interoperability between clouds: challenges and case study. **Proceedings of the 4th European conference on Towards a service-based internet**, p. 62-74, 2011.

PIP Documentation. **Python Packaging Authority**, 2018. Available at: <https://pip.pypa.io/en/stable/>. Accessed on: 18 nov. 2018.

PUPPET OpenStack Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/puppet-openstack-guide/latest/>. Accessed on: 18 nov. 2018.

PYTHON 3.0 Release. **Python.org**, 2008. Available at: <https://www.python.org/download/releases/3.0/>. Accessed on: 18 nov. 2018.

QINLING Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/qinling/latest/>. Accessed on: 18 nov. 2018.

RACKSPACE Website. **Rackspace Website**, 2018. Available at: <https://www.rackspace.com/>. Accessed on: 18 nov. 2018.

RADOS REST Gateway. **Ceph Documentation**, 2018. Available at: <http://docs.ceph.com/docs/master/man/8/radosgw/>. Accessed on: 18 nov. 2018.

RALLY Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/rally/latest/>. Accessed on: 18 nov. 2018.

REFSTACK TEAM. OpenStack Interoperability. **RefStack**, 2018. Available at: <https://refstack.openstack.org/>. Accessed on: 18 nov. 2018.

SAHARA Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/sahara/latest/>. Accessed on: 18 nov. 2018.

SEARCHLIGHT Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/searchlight/latest/>. Accessed on: 18 nov. 2018.

SENLIN Documentation. **OpenSack Docs**, 2018. Available at: <https://docs.openstack.org/senlin/latest/>. Accessed on: 18 nov. 2018.

SHADE Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/shade/latest/>. Accessed on: 18 nov. 2018.

SILVA, G. C.; ROSE, L. M.; CALINESCU, R. A Systematic Review of Cloud Lock-in Solutions. **IEEE International Conference on Cloud Computing Technology and Science**, p. 363-368, 2013.

SOLUM Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/solum/latest/>. Accessed on: 18 nov. 2018.

SPHINX - Python Documentation Generator. **Sphinx Website**, 2018. Available at: <http://www.sphinx-doc.org/en/master/>. Accessed on: 18 nov. 2018.

SWIFT Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/swift/latest/>. Accessed on: 18 nov. 2018.

SWITCHENGINES Cloud. **SWITCHengines Website - Dynamic compute and storage services for research and education**, 2018. Available at: <https://www.switch.ch/engines/>. Accessed on: 18 nov. 2018.

TACKER Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/tacker/latest/>. Accessed on: 18 nov. 2018.

THE Four Opens. **OpenStack Governance**, 2018. Available at: <https://governance.openstack.org/tc/reference/opens.html>. Accessed on: 18 nov. 2018.

THE OpenStack Foundation Community Code of Conduct. **OpenStack Website**. Available at: <https://www.openstack.org/legal/community-code-of-conduct/>. Accessed on: 18 nov. 2018.

TOOSI, A. N.; CALHEIROS, R. N.; BUYYA, R. Interconnected Cloud Computing Environments: Challenges, Taxonomy, and Survey. **ACM Computing Surveys**, v. 47, n. 1, p. 47, abr. 2014. ISSN 7.

TRICIRCLE Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/tricircle/latest/>. Accessed on: 18 nov. 2018.

TRIPLEO Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/tripleo-docs/latest/>. Accessed on: 18 nov. 2018.

TROVE Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/trove/latest>. Accessed on: 18 nov. 2018.

UNIT testing framework. **Python Documentation**, 2018. Available at: <https://docs.python.org/3/library/unittest.html>. Accessed on: 18 nov. 2018.

UNITTEST.TESTCASE function signature. **Python Docs**, 2018. Available at: <https://docs.python.org/3/library/unittest.html#unittest.TestCase>. Accessed on: 18 nov. 2018.

UUID objects according to RFC 4122. **Python Documentation**, 2018. Available at: <https://docs.python.org/3/library/uuid.html>. Accessed on: 18 nov. 2018.

VANBRABANT, B.; JOOSEN, W. Configuration management as a multi-cloud enabler. **CCB '14 Proceedings of the 2nd International Workshop on CrossCloud** , p. 3, 2014.

VEXXHOST Cloud. **VEXXHOST Website - High Performance OpenStack Cloud Computing Solutions**, 2018. Available at: <https://vexxhost.com/>. Accessed on: 18 nov. 2018.

VITRAGE Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/vitrage/latest/>. Accessed on: 18 nov. 2018.

WATCHER Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/panko/latest/>. Accessed on: 18 nov. 2018.

YAML Website. **YAML Website**, 2016. Available at: <http://yaml.org/>. Accessed on: 18 nov. 2018.

YANGUI, S. et al. CompatibleOne: The Open Source Cloud Broker. **J Grid Computing**, p. 17, out. 2013.

ZAQAR Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/zaqar/latest/>. Accessed on: 18 nov. 2018.

ZHANG, Z.; WU, C.; CHEUNG, D. W. L. A Survey on Cloud Interoperability: Taxonomies, Standards, and Practice. **Performance Evaluation Review**, v. 40, n. 4, p. 22, mar. 2013.

ZUN Documentation. **OpenStack Docs**, 2018. Available at: <https://docs.openstack.org/zun/latest/>. Accessed on: 18 nov. 2018.

APPENDIX A – ADDITIONAL OPENSTACK PROJECTS

In addition to the Authentication, Image, Networking, Block Storage, Compute and Object Storage services providing the core functionalities around the provisioning of computational resources, OpenStack implements 49 additional services and tools for various functions in its ecosystem. This appendix provides short descriptions of what role each of the additional services play in the OpenStack ecosystem.

A.1 Web Frontend

While services expose high granularity via HTTP APIs, those can present unnecessary complexity to users, who might need to interact with a user interface (UI) with a subset of the most common functionalities allowing for optimized user experience, as most services on the web, leaving the APIs for advanced service usage.

Horizon (Horizon Documentation, 2018) is the canonical implementation of the OpenStack dashboard, which provides a web-based user interface for the management of cloud resources exposed by the services.

A.2 API Proxies

API Proxies are middleware implementing third-party APIs by mapping incoming requests into the appropriate OpenStack operations and afterward translating responses into the format defined by the external APIs.

The EC2 API (OpenStack EC2 API Documentation, 2018) provides a compatibility layer with EC2, or Amazon Elastic Compute Cloud (AMAZON WEB SERVICES, 2018), which is the core part of the Amazon Web Services providing compute capacity in the cloud.

A.3 Workload Provisioning

Some services are specialized in provisioning workloads that are utilized by user applications, including container orchestration engine (COE), databases and data processing infrastructure.

Magnum (Magnum Documentation, 2018) offers the functionalities of COEs as a service to users, allowing the deployment and management of containers via a consistent API that can communicate with different orchestration backend engines in the market. It currently supports Docker Swarm, Kubernetes and Apache Mesos.

Trove (Trove Documentation, 2018) manages databases with isolation and high performance, automating the complex administrative of deploying, configuring, patching, backing up, restoring and monitoring the databases.

Sahara (Sahara Documentation, 2018) provisions data processing infrastructure as a service, providing Apache Hadoop, Apache Spark and Apache Storm without the complexity of installing these frameworks.

A.4 Application Lifecycle

This set of services allow better management of application lifecycle, improved business continuity and developer productivity, along with high availability of the cloud deployments.

Murano (Murano Documentation, 2018) provides application lifecycle and catalog management as a service, allowing developers to publish and maintain applications that are offered to end users.

Freezer (Freezer Documentation, 2018) implements disaster recovery and backup and restore functionalities that users can leverage into business continuity strategies.

Solum (Solum Documentation, 2018) offers a platform for building, testing and deploying applications on OpenStack clouds via Continuous Intregration and Continuous Delivery, improving developer productivity and application portability.

Masakari (Masakari Documentation, 2018) gives OpenStack deployments high availability to virtual servers running on the platform with automatic recovery in the case of failure.

A.5 Orchestration

Orchestration of computational resources and applications is a key requirement for scalability and automated management. The services in this section implement orchestration, resource management and messaging mechanisms in OpenStack.

Heat (Heat Documentation, 2018) reads user-defined orchestration templates in text files to manage the entire lifecycle of infrastructure and applications in OpenStack clouds.

Mistral (Mistral Documentation, 2018) provides a mechanism to define tasks and workflows, which represent business processes. Users describe workflows as a set of tasks and their transitions. These functionalities are offered via a visualization tool that helps debugging workflow executions, identifying execution paths, exploring task run times, and more.

Aodh (Aodh Documentation, 2018) is the alarming service, enabling the ability to trigger actions depending on metrics or events collected in the cloud.

Senlin (Senlin Documentation, 2018) is a service to create and manage clusters of any OpenStack resources, such as servers. It offers the ability to scale and resize clusters.

Zaqar (Zaqar Documentation, 2018) is a multi-tenant cloud messaging and notification service for use by mobile and web developers, who can use this service to send messages between various software applications in a scalable and secure manner.

Blazar (Blazar Documentation, 2018) controls resource reservations in the cloud environment, where users lease resources for a specific amount of time, immediately or in the future.

A.6 Further Compute

In addition to Nova (Nova Documentation, 2018), the service providing virtual servers which is in the scope of this investigation and further detailed in Compute Scope, OpenStack has the Zun (Zun Documentation, 2018) and Qinling (Qinling Documentation, 2018) projects to provide compute to users' applications.

Zun is the container service, allowing users to create manage workloads on containers without the need to manage servers or clusters. It implements APIs to control containers directly, such as creating, starting, pausing and restarting a container.

Qinling offers functions as a service, supporting serverless computing, which means zero management of servers or containers. Users simply upload their code and the service runs and scales it with high availability.

A.7 Further Networking

Besides Neutron (Neutron Documentation, 2018), the service providing SDN which is in the scope of this research, which is further documented in the section Networking Scope, OpenStack exposes the Octavia (Octavia Documentation, 2018) and Designate (Designate Documentation, 2018) projects in the networking domain.

Octavia is a load balancing as a service solution that is essential to enable simple or automatic delivery scaling and availability for applications. It is useful for cloud operators deploying services in the OpenStack control plane as well as for end users to distribute the workload of applications between multiple servers.

Designate offers DNS as a service, managing DNS zones and records. The DNS servers are therefore set in the servers' network configuration and will translate the domain names when HTTP requests are generated.

A.8 Bare Metal

Not all resources in a cloud are virtualized, the provision of bare metal infrastructure might be in need for several use cases, such as intensive CPU or Graphics Processing Unit (GPU) processing requirements, which makes the use of virtualization unfeasible.

Ironic (Ironic Documentation, 2018) provisions bare metal servers directly on hardware through common remote management protocols such as Preboot Execution Environment (PXE) or Intelligent Platform Management Interface (IPMI) or vendor-specific protocols that are supported via plugins. It integrates with other services, allowing users to interact with bare metal servers the same as they do with virtual machines.

Cyborg (Cyborg Documentation, 2018) is a general-purpose framework for the management of accelerators, such as and Field Programmable Gate Arrays (FPGAs).

A.9 Further Storage

In addition to Swift (Swift Documentation, 2018), which provides Object Storage and is documented in the section Object Storage Scope, and Cinder (Cinder Documentation, 2018) providing Block Storage as documented in the section Block Storage Scope, OpenStack implements the Manila (Manila Documentation, 2018) project.

Manila offers shared or distributed filesystems as a service, providing extensibility to multiple backends and accommodating a variety of shared or distributed file systems types.

A.10 Shared Services

Some services provide functionalities that are useful for a large range of use cases, which can be combined with other services. In addition to Keystone (Keystone Documentation, 2018) and Glance (Glance Documentation, 2018), which are in the scope of this investigation, OpenStack has Barbican (Barbican Documentation, 2018), Searchlight (Searchlight Documentation, 2018) and Karbor (Karbor Documentation, 2018). Glance is documented in the Subsection 2.3.12 and Keystone in the Subsection 2.3.5.

Barbican is the key manager service, providing secure storage, provisioning and management of secret data, including asymmetric and symmetric keys, certificates and raw binary data.

Searchlight provides indexing and search capabilities on resources, with high performance and flexible querying along with near real-time indexing of attributes.

Karbor protects and restores application data as a service. It implements a standard framework that allows vendors to provide plugins through a unified interface to protect data and metadata against loss and damage using mechanisms such as backup and replication.

A.11 Further SDK

Along with OpenStack SDK (OpenStack SDK Documentation, 2018), the library object of this investigation, the community develops the OpenStack Client (OpenStackClient Documentation, 2018) library.

OpenStack Client is a command-line client that presents a uniform command structure exposing a command set for OpenStack resources. The example in Figure 55 shows the bash command to create a server named *masters* with the image *ubuntu* and the flavor *m1.medium*.

Figure 55 - Creating a server with OpenStack Client

```
1 $ openstack server create --image ubuntu --flavor m1.medium --wait masters
```

Source: Created by the author of this document.

A.12 Container Services

Additional container networking and storage services exist to connect the container technologies with the existing services in OpenStack, leveraging functionalities and stability.

Kuryr (Kuryr Documentation, 2018) exposes networking capabilities to containers by leveraging Neutron resources and services to Docker, acting as a proxy.

Fuxi (Fuxi Documentation, 2018) connects Docker containers with volumes from Cinder and shared filesystems from Manila, leveraging the advanced features and numerous vendor-specific drivers already implemented in these services.

A.13 Network Function Virtualization (NFV)

NFV is a networking architecture that virtualizes network nodes' functions to create communication services. It is particularly interesting for the telecommunication business, which requires agility to provision and maintain network connectivity.

Tacker (Tacker Documentation, 2018) has a general-purpose Virtual Network Functions Manager (VNFM) that deploys and operates Network Function Virtualization (NFV). An example use case is to utilize the VNFM to create VNFs in service provider network to deliver agile network services for remote customer networks.

A.14 Cloud Deployment and Lifecycle

Deploying and managing OpenStack clouds is a complex task, given the amount of potential connecting microservices and flexibility supporting an enormous range of use cases. There are scripts created by the community to support these tasks by communicating with existing OpenStack clouds, Ansible (ANSIBLE, 2018), Juju Charms (Jujucharms, 2018) and Helm (Helm - The Kubernetes Package Manager, 2018).

Kolla-Ansible (Kolla-Ansible Documentation, 2018) offers Ansible playbooks to meet Kolla's mission, which is to provide production-ready containers to deploy and operate clouds. Kolla is highly opinionated but is customizable, permitting operators with minimal experience to deploy OpenStack rapidly and as experience grows modify the configuration to better suit the operator's exact requirements.

OpenStack-Charms (OpenStack Charms Documentation, 2018) implement Juju Charms that contain all the instructions to deploy and configure OpenStack services, allowing operators to deploy services and scale with minimal effort in a fast, reliable and repeatable manner.

OpenStack-Helm (OpenStack-Helm Documentation, 2018) provides means to deploy an OpenStack cloud using Helm, leveraging all its features and ecosystem benefits to the deploy and management of clouds.

TripleO (TripleO Documentation, 2018), or OpenStack On OpenStack, permits installing, upgrading and operating OpenStack clouds on the top of another OpenStack cloud. The base cloud is called *undercloud*, while the TripleO deployed clouds are *overclouds*.

Bifrost (Bifrost Documentation, 2018) is a set of Ansible playbooks that deploy Ironic, the bare metal service, in a standalone manner, facilitating the deployment of infrastructure.

A.15 Packaging Recipes

There are some projects offering scripts and packaging to enable building and operating OpenStack clouds in an automated manner. The services are packaged both in RPM and Open Container Initiative (OCI) containers. Additionally, services may be provisioned with Ansible playbooks (OpenStack-Ansible Documentation, 2018), Puppet modules (Puppet OpenStack Documentation, 2018) and Chef cookbooks (Chef OpenStack Documentation, 2018).

A.16 Monitoring

Monitoring systems and resources is essential to detect, analyze and fix faults or misbehaviors. The services in this section provide functionalities to collect, store and process monitoring information on servers and resources.

Ceilometer (Ceilometer Documentation, 2018) is a service that collects and distributes event and metering data from services. The collected information can be used to provide customer billing, resource tracking and alarming capabilities.

Monasca (Monasca Documentation, 2018) provides a multi-tenant, highly scalable, performant and fault-tolerant monitoring as a service solution. It allows monitoring systems, which can be user servers or the actual hosts running the OpenStack control plane.

Panko (Panko Documentation, 2018) stores and allows querying event data that enable users to capture the state of resources at a given time, enabling a scalable mechanism of storing data for use cases such as system debugging and auditing.

A.17 Optimization and Policy

There are additional services focused on cloud optimization, root cause analysis of problems, policy management and non-functional attributes testing.

Watcher (Watcher Documentation, 2018) is a resource optimization service, providing a complete workflow from receiving metrics to process and profile complex events and apply action plans, resulting in a robust framework for cloud optimization, covering use cases such as the reduction of data center operating costs, increased energy efficiency and system performance via intelligent server migration.

Vitrage (Vitrage Documentation, 2018) offers a root cause analysis service for organizing, analyzing, and expanding alarms and events, ultimately generating insights regarding the root cause of problems and deducing their existence before they are directly detected

Congress (Congress Documentation, 2018) is an open policy for clouds, allowing operators to declare, monitor, enforce, and audit policies. It fetches information about resources from other services and verifies that the cloud's actual state abides by the operator's policies. It works with any policy on any cloud resource.

Rally (Rally Documentation, 2018) is a testing tool that checks how OpenStack work at scale by implementing cloud verification, testing and profiling. It automates and unifies multi-node deployment and can be used by OpenStack distributions to continuously improve SLAs, performance and stability.

A.18 Billing and Business Logic

CloudKitty (CloudKitty Documentation, 2018) offers rating as a service, translating usage metrics to prices. It allows operator to configure the price of cloud resources with a hash map, where each resource gets a price for a certain period of use, or by uploading custom plugins defining pricing strategies that address specific requirements of the business model.

A.19 Multi-region

It is very common that cloud providers define business strategies that span multiple geographic regions, and this decision reflects into the OpenStack deployments, where a single cloud may expose services running on different continents, for example. By default, this does not permit, however, a consistent networking management across the different locations, which is a functionality provided by the Tricircle (Tricircle Documentation, 2018) service.

Tricircle implements a distributed setup of the Neutron service in a multi-region deployment. There is a central node that coordinates the operations with the local nodes, which run in each of the regions. With this centralization mechanism, it can ensure IP address pools, IP and MAC address and network segments allocations are managed globally without conflict. Additionally, it can handle the data link and the network layers across local Neutron servers, allowing servers in a single tenant to communicate with each other, no matter what region they are in fact running on.

APPENDIX B – SERVICE CATALOG AND DISCOVERY

In the Sections B.1 Service Catalog and B.2 Service Discovery, this appendix details what the service catalog and the service discovery are, which ultimately allows consumers to programmatically communicate with OpenStack services via versioned URLs.

B.1 Service Catalog

The service catalog enumerates the service endpoints an authenticated user may communicate with, categorized by cloud region. It is an extensive document containing all endpoints of all services, for all the cloud regions available. A provider might have multiple regions in a cloud, such as one running on a datacenter in Brazil and another running on Australia, where both are available for users after authenticating against the cloud.

Service Endpoints

A service endpoint is mainly composed of a service URL, a region name, and an interface name, which defines whom the endpoint URL is intended to be consumed by, and is one of *public*, *internal* or *admin*.

The URLs of *public* interfaces are visible from the internet, thus intended for consumption by end users; *internal* interfaces might be restricted to the hosts that contain OpenStack services, thus intended for consumption by other services; and *admin* interfaces might be restricted to administrators who operate the cloud infrastructure, generally on a secure network interface. An example of service endpoints for an arbitrary service is in Figure 56.

Figure 56 - Service endpoints example

```
 1 [ {
 2     "interface": "public",
 3     "region": "BR",
 4     "url": "https://br.example.com:5000/"
 5   },
 6   {
 7     "interface": "internal",
 8     "region": "BR",
 9     "url": "https://10.0.0.21:5000/"
10   },
11   {
12     "interface": "admin",
13     "region": "BR",
14     "url": "https://192.168.0.6:5000/"
15   } ]
```

Source: Created by the author of this document.

For this study, only *public* endpoints will be considered, as the testing scripts in the validation of the OpenStack SDK may consume them.

Example

Figure 57 is an excerpt of a service catalog, showing the public endpoint for the identity service on the region BR, referring to the cloud region deployed in Brazil.

Figure 57 - Service catalog example

```
[
  {
    'endpoints':[
      {
        'url': 'https://br.example.com:5000/',
        'interface': 'public',
        'region': 'BR',
        'id': 'cf7e04819c924891a848c16ea796e43e'
      }
    ],
    'type': 'identity',
    'id': '8f9c8ca448a246ca976b118b8324f3b8',
    'name': 'keystone'
  }
]
```

Source: Created by the author of this document.

As stated earlier, a service catalog is an extensive document. For example, a cloud with 4 regions, each with 6 services running the 3 interface types would have 72 service endpoints in its service catalog.

B.2 Service Discovery

The URLs in the service catalog are generally unversioned, i.e. do not have version identification in it. However, the URLs that one uses to communicate with the services need to be versioned.

Service discovery is the mechanism for discovering the appropriate versioned URL from an unversioned root URL, which is a two-step process of discovering what versions are available and defining what is the most appropriate. This process runs whenever an operation on a resource is triggered, e.g., the discovery for the identity service would be triggered when a call to retrieve a user information was requested by an OpenStack SDK user.

To discover what versions are available for a service, the mechanism first needs to query the unversioned URL, which will return a document listing the versions available and their respective versioned URLs. Figure 58 exemplifies this step for the service catalog in Figure 57.

Figure 58 - Version discovery example

```
$ curl https://br.example.com:5000
{
   "versions":{
      "values":[
         {
            "status":"stable",
            "updated":"2016-04-04T00:00:00Z",
            "id":"v3.6",
            "links":[
               {
                  "href":"https://br.example.com:5000/v3/",
                  "rel":"self"
               }
            ]
         },
         {
            "status":"stable",
            "updated":"2014-04-17T00:00:00Z",
            "id":"v2.0",
            "links":[
               {
                  "href":"https://br.example.com:5000/v2.0/",
                  "rel":"self"
               }
         }
      ]
   }
}
```

Source: Created by the author of this document.

The discovery mechanism then enters the second step with a list of candidate versioned URLs containing *https://br.example.com:5000/v3/* and *https://br.example.com:5000/v2.0/*, representing the versions 3.6 and 2.0, respectively. The mechanism gives precedence to newer versions, since they are expected to provide the latest features and bug fixes. In this case, the latest version is v3.6, represented by the URL *https://br.example.com:5000/v3/*, which is subsequently used to perform whatever operation was requested by the end user.

APPENDIX C – OPENSTACK SDK DATA MODEL

Because of communicating to multiple API versions of services, OpenStack SDK receives heterogeneous responses that are normalized to a comprehensive data model, allowing users to safely rely on its responses regardless of the underlying cloud configuration. This appendix presents the data models (OpenStack SDK Data Model, 2018) for the Image, Networking, Block Storage, Compute and Object Storage resources in the scope of this investigation.

C.1 Image

Figure 59 - Image data model

Source: Created by the author of this document.

C.2 Networking

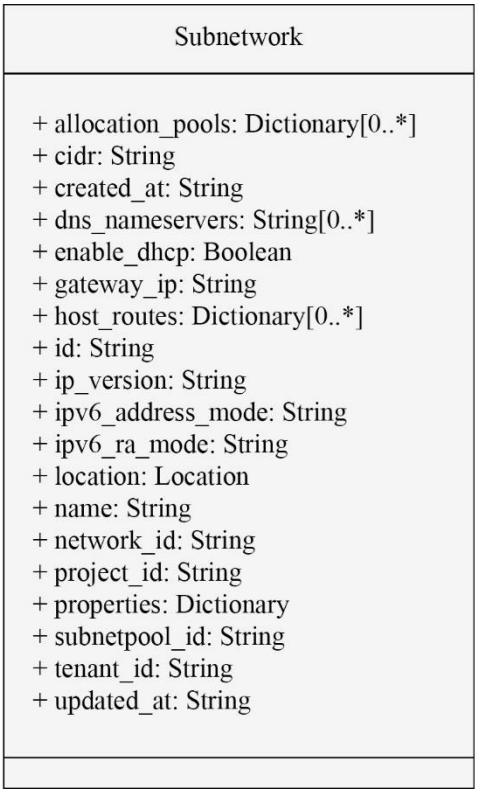
Network:

Figure 60 - Network data model

Source: Created by the author of this document.

Subnetwork:

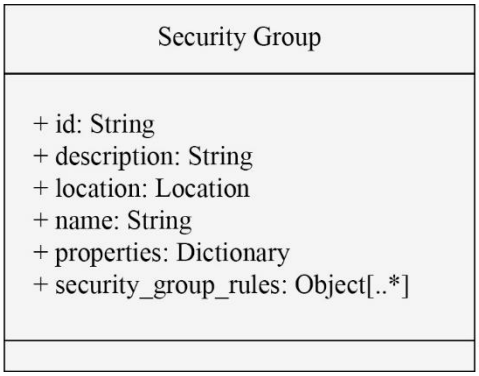
Figure 61 - Subnetwork data model



Source: Created by the author of this document.

Security Group:

Figure 62 - Security group data model

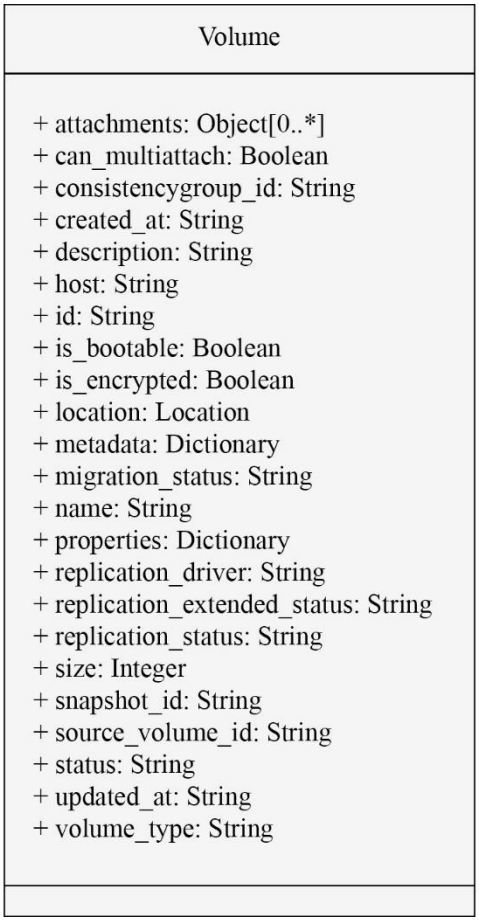


Source: Created by the author of this document.

C.3 Block Storage

Volume:

Figure 63 - Volume data model

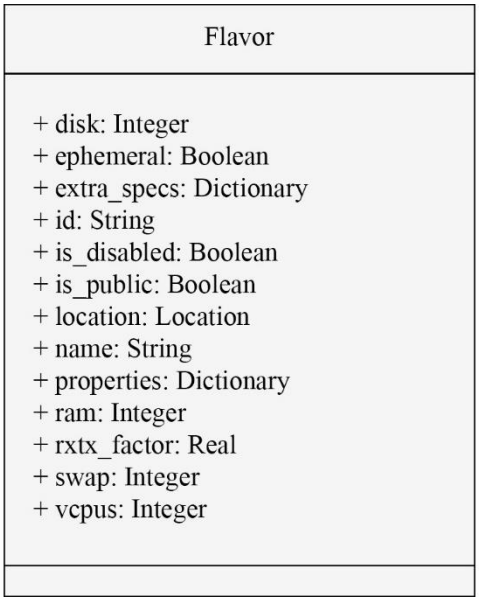


Source: Created by the author of this document.

C.4 Compute

Flavor:

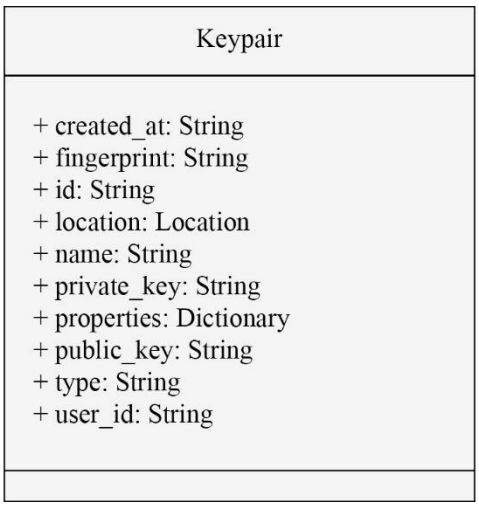
Figure 64 - Flavor data model



Source: Created by the author of this document.

Keypair:

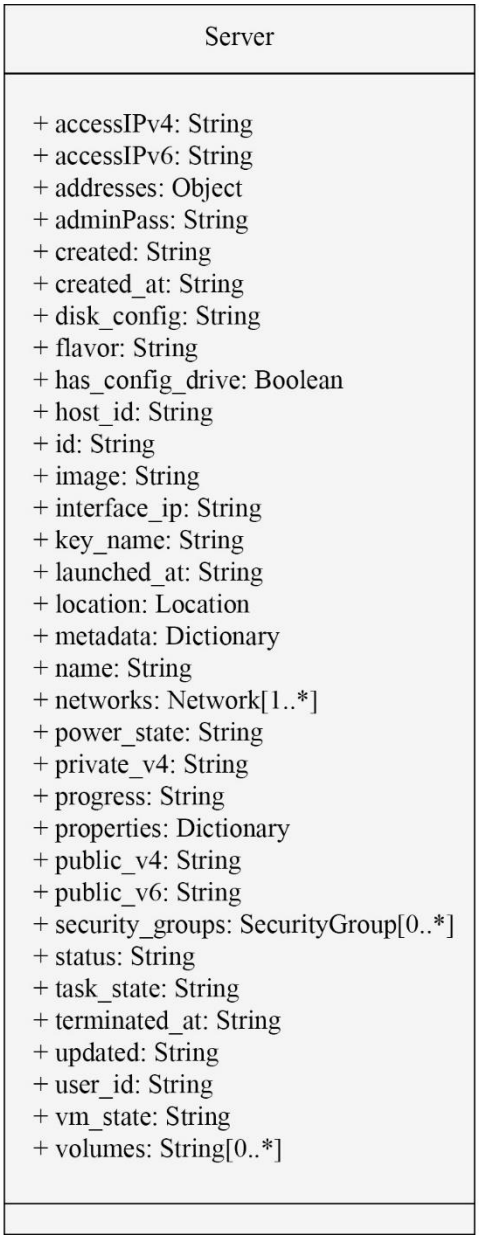
Figure 65 - Keypair data model



Source: Created by the author of this document.

Server:

Figure 66 - Server data model



Source: Created by the author of this document.

Snapshot:

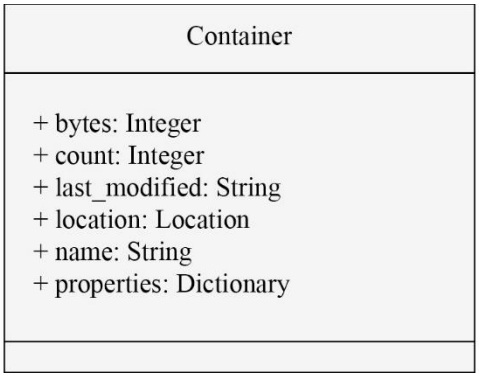
Figure 67 - Snapshot data model

Source: Created by the author of this document.

C.5 Object Storage

Container:

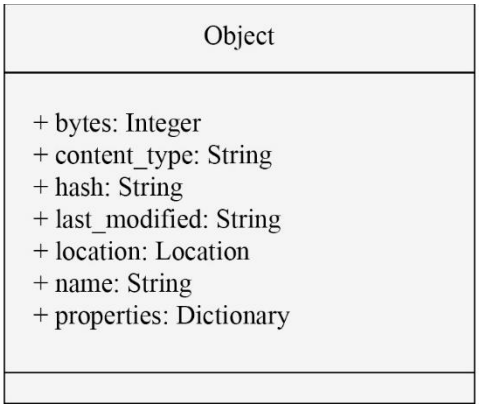
Figure 68 - Container data model



Source: Created by the author of this document.

Object:

Figure 69 - Object data model



Source: Created by the author of this document.

APPENDIX D – AUTHOR CONTRIBUTIONS

This appendix enumerates the author's contributions to OpenStack SDK during the period of this investigation.

D.1 De-client-ify Identity Resources Management

Contributions on this section implement the use of Keystone's REST APIs directly, as opposed to the Keystone Python client. The objective was to help ensuring backward compatibility at the API level, as opposed to the Python client level. Table 19 enumerates the commits for general resources, while Table 20 for the core identity resources: users and groups.

Table 19 - Contributions for REST APIs on resources management

Commit	URL
De-client-ify Endpoint Create	https://review.openstack.org/#/c/502346/
De-client-ify Endpoint Update	https://review.openstack.org/#/c/501352/
De-client-ify List Role Assignments	https://review.openstack.org/#/c/501268/
De-client-ify Endpoint List	https://review.openstack.org/#/c/501200/
De-client-ify List Roles for User in v2.0	https://review.openstack.org/#/c/501186/
De-client-ify Role Grant and Revoke	https://review.openstack.org/#/c/501039/
De-client-ify Endpoint Delete	https://review.openstack.org/#/c/500943/
De-client-ify Service Delete	https://review.openstack.org/#/c/482197/
De-client-ify Service Update	https://review.openstack.org/#/c/482196/
De-client-ify Service List	https://review.openstack.org/#/c/482195/
De-client-ify Role Delete	https://review.openstack.org/#/c/485145/
De-client-ify Role List	https://review.openstack.org/#/c/482246/
De-client-ify Role Create	https://review.openstack.org/#/c/482245/
De-client-ify Service Create	https://review.openstack.org/#/c/482194/
De-client-ify Domain Search	https://review.openstack.org/#/c/478324/
De-client-ify Domain Get	https://review.openstack.org/#/c/478323/
De-client-ify Domain List	https://review.openstack.org/#/c/478322/
De-client-ify Domain Update and Delete	https://review.openstack.org/#/c/478321/
De-client-ify Domain Create	https://review.openstack.org/#/c/478225/
De-client-ify Project Update	https://review.openstack.org/#/c/473629/
De-client-ify Project Create	https://review.openstack.org/#/c/473609/
De-client-ify Project Delete	https://review.openstack.org/#/c/473628/
De-client-ify Project List	https://review.openstack.org/#/c/473581/

Source: Created by the author of this document.

Table 20 - Contributions for REST APIs on identity management

Commit	**URL**
De-client-ify Group Delete	https://review.openstack.org/#/c/485122/
De-client-ify Group Update	https://review.openstack.org/#/c/485118/
De-client-ify Group List	https://review.openstack.org/#/c/485108/
De-client-ify Group Create	https://review.openstack.org/#/c/485102/
De-client-ify User Password Update	https://review.openstack.org/#/c/500930/
De-client-ify Remove User from Group	https://review.openstack.org/#/c/499360/
De-client-ify Check User in Group	https://review.openstack.org/#/c/499357/
De-client-ify Add User to Group	https://review.openstack.org/#/c/499345/
De-client-ify User Update	https://review.openstack.org/#/c/499284/
De-client-ify User Create	https://review.openstack.org/#/c/474397/
Remove keystoneclient dependency	https://review.openstack.org/#/c/502518/

Source: Created by the author of this document.

D.2 Refactoring and Cleanups

This section enumerates refactorings and cleanups in Table 21. Refactorings are improvements without functional impacts, such as reorganizing code for better readability or maintainability. Code cleanups remove unnecessary code.

Table 21 - Code contributions for refactoring and cleanups

Commit	URL
Refactor the create endpoint code	https://review.openstack.org/#/c/502345/
Reorganize endpoint create code	https://review.openstack.org/#/c/502344/
Switch to _is_client_version in list_services	https://review.openstack.org/#/c/497171/
Consolidate client version checks in an utility method	https://review.openstack.org/#/c/493598/
Consolidate the use of self._get_and_munchify	https://review.openstack.org/#/c/485523/
_discover_latest_version is private and not used	https://review.openstack.org/#/c/457787/
Remove improper exc handling in is_user_in_group	https://review.openstack.org/#/c/500560/
Consolidate cloud/base.py into functional/base.py	https://review.openstack.org/#/c/591031/

Source: Created by the author of this document.

D.3 Documentation

Improvements in the OpenStack SDK's documentation are enumerated in Table 22. The necessity of improving the documentation was noticed when doing the validation of the library.

Table 22 - Documentation contributions

Commit	URL
Improve grant docs on when and how use domain arg	https://review.openstack.org/#/c/473589/
Rename ClusterTemplate in OpenStackCloud docs	https://review.openstack.org/#/c/439779/
Fix OpenStack and ID misspellings	https://review.openstack.org/#/c/439744/
Remove service names in OpenStackCloud docs	https://review.openstack.org/#/c/439709/
Add compute API info and fix provider names	https://review.openstack.org/#/c/605253/
Update vendor support info for vexxhost	https://review.openstack.org/#/c/605252/
Update vendor support info for switchengines	https://review.openstack.org/#/c/605249/
Update vendor support info for ecs	https://review.openstack.org/#/c/605248/
Update vendor support info for catalyst	https://review.openstack.org/#/c/605245/
Fix location region field in docs	https://review.openstack.org/#/c/605081/

Source: Created by the author of this document.

D.4 Failure Fixes

During the validation of OpenStack, some failures were observed and reported as bugs to the project, and subsequently fixed in the commits listed in Table 23.

Table 23 - Code contributions for failure fixes

Commit	URL
Project update to change enabled only when provided	https://review.openstack.org/#/c/476914/
Invalidate cache upon container deletion	https://review.openstack.org/#/c/596545/
Listing containers to return Munch objects	https://review.openstack.org/#/c/596539/
Listing objects to return Munch objects	https://review.openstack.org/#/c/596780/
Format URL when updating image props in Glance v1	https://review.openstack.org/#/c/605534/
Send image fields in headers for PUT in Glance v1 [4]	https://review.openstack.org/#/c/605535/

Source: Created by the author of this document.

[4] Commit under review by the OpenStack community by November 18th.

D.5 New Functionalities

Some features were not consistent across the management of different resources, such as normalizing entities. Additionally, some functionalities were not implemented yet, as they are added to OpenStack SDK as users need them. Table 24 enumerates the code contributions for new functionalities.

Table 24 - Code contributions for new functionalities

Commit	URL
Implement volume update	https://review.openstack.org/#/c/600057/
Implement network update	https://review.openstack.org/#/c/600050/
Allow search on objects	https://review.openstack.org/#/c/600683/
Allow JMESPath on searching networking resources	https://review.openstack.org/#/c/599078/
Allow search on containers	https://review.openstack.org/#/c/600680/
Allow returning munch on create and get object [5]	https://review.openstack.org/#/c/597198/
Allow returning munch on create and get container [5]	https://review.openstack.org/#/c/596546/
Normalize security groups when using Neutron	https://review.openstack.org/#/c/602147/
Normalize image when using PUT on Glance v2	https://review.openstack.org/#/c/602031/
Normalize object resources [5]	https://review.openstack.org/#/c/602234/
Normalize container resources [5]	https://review.openstack.org/#/c/602233/
Normalize subnet resources [5]	https://review.openstack.org/#/c/602228/
Normalize network resources [5]	https://review.openstack.org/#/c/602218/

Source: Created by the author of this document.

[5] Commit under review by the OpenStack community by November 18th.

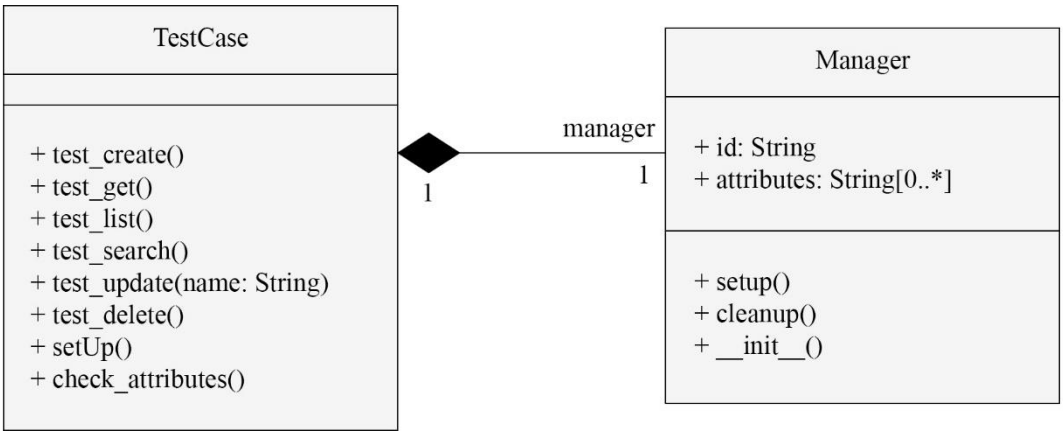
APPENDIX E – TESTING CODE AND ENVIRONMENT

This appendix describes the elements necessary to validate OpenStack SDK against OpenStack clouds using automated test cases. The Section E.1 Testing Architecture explains the architecture of the tests. Section E.2 Testing Code presents the code developed to implement the tests. Section E.3 Testing Environment details the configuration and additional tooling necessary to prepare the local environment to run the automated tests and to understand the remote cloud. Section E.4 Running Tests shows how the tests are run, and Section E.5 Authorization details the explicit authorization granted from the cloud providers to mention their names in this document, preventing any potential legal issues.

E.1 Testing Architecture

The tests are designed to cover the basic operations for handling cloud resources: create, get, list, search, update and delete. As shown in Figure 70, each test scenario is a composition of two objects: a resource manager, that is responsible for performing the operations on a resource; and a test case suite, which implements the test cases by calling the resource manager and making assertions on its responses.

Figure 70 - Test scenario with test suite and resource manager



Source: Created by the author of this document.

Both managers and test case classes define a base class that groups the common behavior, reusing code.

E.1.1 Manager Classes

The manager classes handle cloud resources by making appropriate calls to OpenStack SDK. As the calls to OpenStack SDK really vary depending on the resource, there is not much logic that can be grouped in a base class, leaving it with the shared logic to set up and tear down dependencies. For example, the function to create a keypair is *create_keypair* and only takes the parameters *name* and *public_key*, whereas the function to create an image takes numerous parameters, both as shown in the code excerpts of Figure 71.

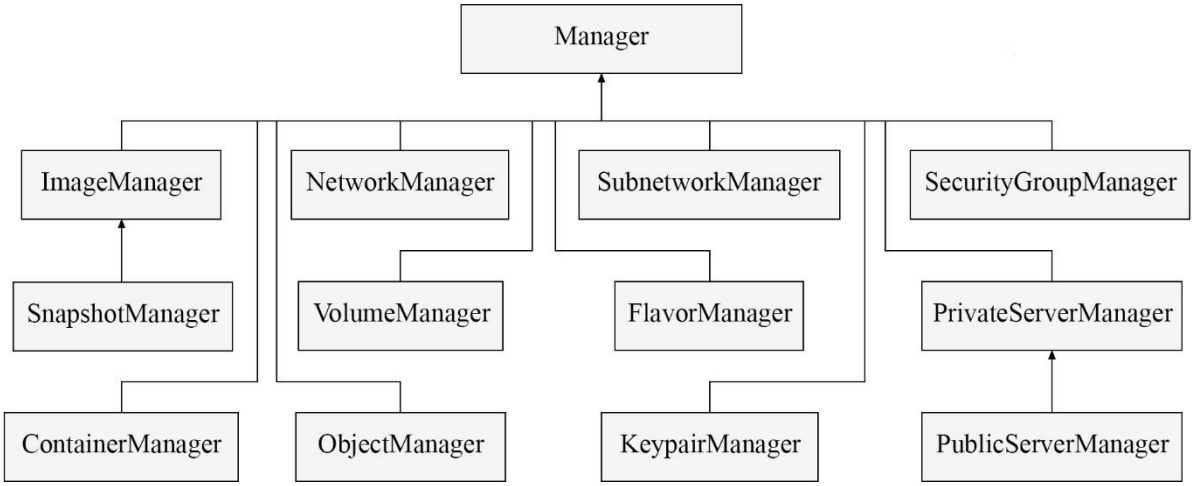
Figure 71 - OpenStack SDK functions to create keypair and image

```
1   def create_keypair(self, name, public_key=None):
2     ...
3
4   def create_image(
5       self, name, filename=None, container=None, md5=None, sha256=None,
6       disk_format=None, container_format=None, disable_vendor_agent=True, wait=False,
7       timeout=3600, allow_duplicates=False, meta=None, volume=None, **kwargs):
8     ...
```

Source: Created by the author of this document.

Figure 72 shows the base manager class with its common methods and attributes, being inherited by all the specific managers handling resources in the scope of this study.

Figure 72 - Resource manager and its subclasses



Source: Created by the author of this document.

E.1.2 Test Case Classes

As the test case classes own resource managers, they define the tests on a given resource by making assertions on the calls to their respective managers. As the code to test the basic operations on a resource does not vary depending the resource, most of the logic is kept in a base class. For example, the logic of testing a get operation on a server is no different from the logic for a get on an image, where in both cases it calls the manager and the return should not be null and should contain an ID and a location field, along with specific attributes of each resource, as illustrated in Figure 73.

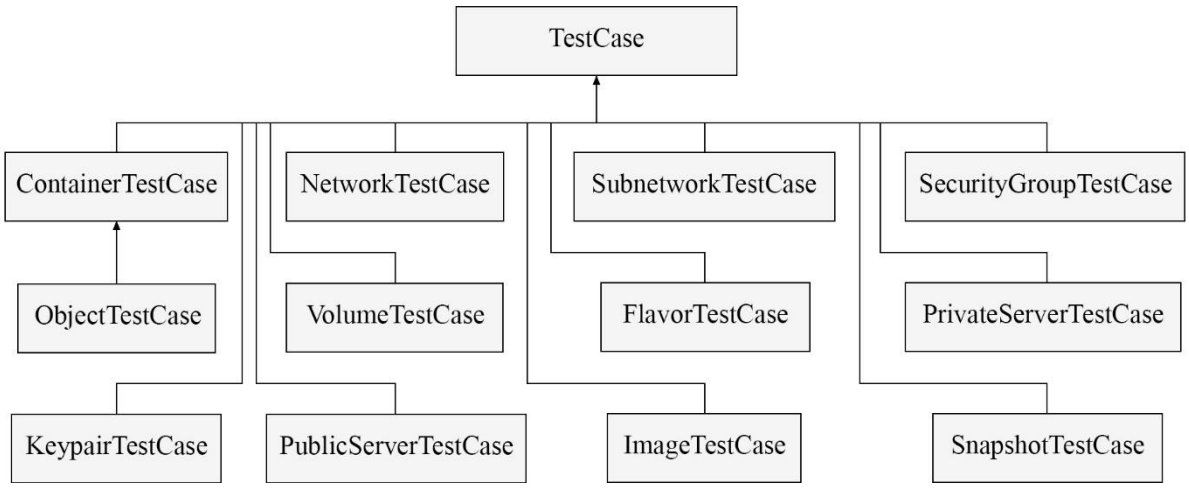
Figure 73 - Logic for testing the get operation on resources

```
 1  def test_get(self):
 2    obj = self.manager.get()
 3    self.check_attributes(obj)
 4
 5  def check_attributes(self, obj):
 6    self.assertIsNotNone(obj)
 7    for attr in self.manager.attributes:
 8      self.assertIn(attr, obj)
 9    self.assertIsNotNone(getattr(obj, self.manager.id))
10    for attr in ('cloud', 'region_name', 'zone', 'project'):
11      self.assertIn(attr, obj.location)
12    self.assertEqual(obj.location.cloud, os.environ['CLOUD'])
```

Source: Created by the author of this document.

Figure 74 shows the base test case suite class with its common methods and attributes, being inherited by all the specific test case classes for resources in the scope of this study.

Figure 74 - Test case suite and its subclasses



Source: Created by the author of this document.

E.1.3 Test Skips

As an end user, not all operations may be performed on some resources. The first case is when the operation requires administrative privileges, which is the case for creating, updating and deleting flavors, which are intended for cloud operators; the second case is the update operation on immutable resources, which is not reasonable and is the case for keypair name and content, along with container and object names. The managers of these resources do not implement such operations, and their respective test cases are marked as to be skipped.

E.2 Testing Code

This section presents the Python code implemented to validate OpenStack SDK against clouds, respecting the architecture established in the previous section E.1 Testing Architecture. All the code below belongs to a single file named *test_sdk.py*, appended in the order the code excerpts are presented.

E.2.1 Base Code

The code presented in this section serves as foundation for the code that is specific to each resource, grouping all the common behavior of both manager and test cases by applying the concepts of inheritance and polymorphism of the object-oriented programming paradigm.

Dependencies:

The code excerpts in this section may depend on some of the Python libraries imported in Figure 75.

Figure 75 - Test dependencies

```
1 import openstack
2 import os
3 import unittest
4 import uuid
```

Source: Created by the author of this document.

openstack (OpenStack SDK Connection Object, 2018) represents the connection module of OpenStack SDK, which allows an *OpenStackCloud* (OpenStackCloud Code, 2018)

object to be instantiated upon connecting to a cloud. *os* (Miscellaneous operating system interfaces, 2018) provides interface with operational system functions, allowing to read system environment variables. *unittest* (Unit testing framework, 2018) provides base functions allowing test cases to write assertions to be checked at runtime. Lastly, *uuid* (UUID objects according to RFC 4122, 2018) is used to name resources when running the test cases with unique strings.

Base Manager:

The base manager class is *Manager* and it groups all the shared logic between all the specific managers, i.e. its subclasses, as per Figure 76.

Figure 76 - Base resource manager class

```
 1 class Manager(object):
 2
 3   def __init__(self):
 4     try:
 5       self.setup()
 6       self.cloud = openstack.connect(cloud=os.environ['CLOUD'])
 7       self.obj = self.create() if hasattr(self, 'create') else self.get()
 8     except:
 9       self.cleanup()
10       raise
11
12   @property
13   def id(self):
14     return 'id'
15
16   def setup(self):
17     pass
18
19   def cleanup(self):
20     pass
```

Source: Created by the author of this document.

The attributes of a manager are *cloud*, *obj* and *id*, as shown in lines 6, 7 and 12 to 14, respectively. *cloud* is the instance of *OpenStackCloud* (OpenStackCloud Code, 2018), a connection to the cloud which name is set in the system environment variable *CLOUD*; *obj* is an instance of the resource under testing; while *id* represents the name of the field that is the main identifier of the resource.

In addition to that, the manager implements the functions *setup*, *cleanup* and the constructor *__init__*. *setup*, in lines 16 and 17, instantiates all the dependencies to other managers that will be used when operating over the resource under testing, e.g. when creating a server, an image will be required, meaning *ServerManager* will depend on *ImageManager*. The manager dependencies respect the resource dependencies defined in the Subsection 2.3.17.

The function *cleanup*, in lines 19 and 20, deletes all the resources created by dependent managers to ensure no garbage is left. Both *setup* and *cleanup* are abstract, with the behavior implemented in the subclasses since it depends on the resource under testing. Lastly, *__init__*, in lines 3 to 10, is the constructor that calls *setup* to instantiate all the dependencies, creates the *cloud* connection and *obj*; if anything goes wrong, it calls *cleanup* and raise an exception.

Base Test Case:

The base test class is *TestCase* and groups all the testing logic, as in Figure 77.

Figure 77 - Code of the base test case suite

```
 1 class TestCase(object):
 2
 3   def setUp(self):
 4     self.manager = self.manager_class()
 5     self.addCleanup(self.manager.cleanup)
 6     if hasattr(self.manager, 'delete'):
 7       self.addCleanup(self.manager.delete)
 8
 9   def check_attributes(self, obj):
10     self.assertIsNotNone(obj)
11     for attr in self.manager.attributes:
12       self.assertIn(attr, obj)
13     self.assertIsNotNone(getattr(obj, self.manager.id))
14     for attr in ('cloud', 'region_name', 'zone', 'project'):
15       self.assertIn(attr, obj.location)
16     self.assertEqual(obj.location.cloud, os.environ['CLOUD'])
17
18   def test create(self):
19     self.check_attributes(self.manager.obj)
20
21   def test_get(self):
22     obj = self.manager.get()
23     self.check_attributes(obj)
24
25   def test_list(self):
26     objs = self.manager.list()
27     for obj in objs:
28       self.check_attributes(obj)
29     self.assertIn(getattr(self.manager.obj, self.manager.id),
30                   [getattr(obj, self.manager.id) for obj in objs])
31
32   def test_search(self):
33     objs = self.manager.search()
34     for obj in objs:
35       self.check attributes(obj)
36     self.assertEqual([getattr(self.manager.obj, self.manager.id)],
37                      [getattr(obj, self.manager.id) for obj in objs])
38
39   def test_update(self):
40     new_name = uuid.uuid4().hex
41     obj = self.manager.update(name=new_name)
42     self.check_attributes(obj)
43     self.assertEqual(obj.name, new_name)
44
45   def test_delete(self):
46     self.assertIsNotNone(self.manager.get())
47     self.manager.delete()
48     self.assertIsNone(self.manager.get())
```

Source: Created by the author of this document.

This class owns a single attribute, named *manager*, which is an instance of the manager class that handles the resource under testing. Additionally, it implements the *setUp* and *check_attributes* functions to support the test cases, in the lines 3 to 7 and 9 to 16, respectively. *setUp* instantiates the *manager* attribute based on the *manager_class* attribute that is set in the subclasses; while *check_attributes* checks that a given resource object has the expected fields based on the *manager*'s *attributes* attribute.

Furthermore, it implements six test cases that will run for every resource in this validation: *test_create*, *test_get*, *test_list*, *test_search*, *test_update* and *test_delete*.

For the creation workflow, *test_create*, implemented in lines 18 and 19, calls *check_attributes* with the manager's *obj* attribute that was created when instantiating the manager.

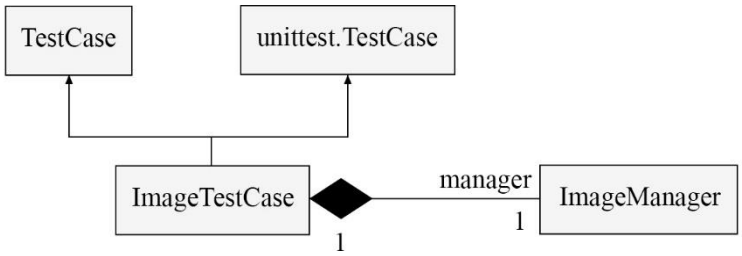
Regarding the retrieval, *test_get*, in lines 21 to 23, calls the get method of the manager and checks its result with *check_attributes*. *test_list*, implemented in lines 25 to 30, calls the manager's list method and checks the result by calling *check_attributes* for each resource of the retrieved list and that the manager's *obj* is in that list. *test_search*, in lines 32 to 37, calls the manager's search function and checks the result by calling *check_attributes* for each resource of the retrieved list. Additionally, it checks that the list only contains exactly the manager's *obj*.

In the update workflow, *test_update*, in lines 39 to 43, calls the update function of the manager to change the resource's *name* attribute, then checks the returned resource is valid with *check_attributes* and that its *name* has in fact been updated.

Lastly, in the delete workflow, *test_delete*, implemented in lines 45 to 48, retrieves the resource under testing by calling the manager's get method and asserts it is not null, thus it calls delete in the manager and then ensure the get function will now return null, as the object has been deleted.

For all those test cases, the Python testing framework *unittest* (Unit testing framework, 2018) is used to develop and run the tests. Although the base class does not inherit from *unittest.TestCase* (unittest.TestCase function signature, 2018) directly, all its subclasses do, making use of the multiple inheritance mechanism of Python to allow the tests to be discovered and run by the testing engine, as in the diagram in Figure 78.

Figure 78 - Test case inheritance with unittest



Source: Created by the author of this document.

The subsections that follow show the code of the specific managers that map the calls to OpenStack SDK and test case suites, which inherit from the base class and skip inappropriate tests, if any, from Figure 79 up to Figure 102.

E.2.2 Image Code

This subsection presents the code for validating the image operations. The manager implements the *attributes* property that lists all the fields for the image data model, as documented in the Section C.1 Image of APPENDIX C – OPENSTACK SDK DATA MODEL.

Figure 79 - Code of the image manager

```
class ImageManager(Manager):

  @property
  def attributes(self):
    return ['location', 'size', 'min_ram', 'min_disk', 'created_at', 'updated_at',
            'tags', 'status', 'is_protected', 'locations', 'properties', 'is_public',
            'visibility', 'checksum', 'container_format', 'direct_url', 'disk_format',
            'file', 'id', 'name', 'owner', 'virtual_size']

  def create(self):
    return self.cloud.create_image(name=uuid.uuid4().hex + 'sdk',
                                   filename='cirros-0.4.0-x86_64-disk.img')

  def get(self):
    return self.cloud.get_image(self.obj.id)

  def list(self):
    return self.cloud.list_images()

  def search(self):
    return self.cloud.search_images(filters='[?contains(name, `sdk`)]')

  def update(self, name):
    self.cloud.update_image_properties(name_or_id=self.obj.id, name=name)
    return self.get()

  def delete(self):
    self.cloud.delete_image(self.obj.id, wait=True)
```

Source: Created by the author of this document.

Figure 80 - Code of the image test case suite

```
1 class ImageTestCase(TestCase, unittest.TestCase):
2
3   manager_class = ImageManager
```

Source: Created by the author of this document.

E.2.3 Networking Code

The code for validating the operations of networks, subnetworks and security groups is presented in this section. The managers implement the *attributes* property, which lists all the fields of the data model for the resources they manage, as in the Section C.2 Networking of APPENDIX C – OPENSTACK SDK DATA MODEL.

Network:

Figure 81 - Code of the network manager

```
 1 class NetworkManager(Manager):
 2
 3   @property
 4   def attributes(self):
 5     return ['availability_zone_hints', 'availability_zones', 'admin_state_up',
 6             'created_at', 'external', 'id', 'location', 'mtu', 'name',
 7             'port_security_enabled', 'project_id', 'properties', 'shared', 'status',
 8             'subnets', 'tenant_id', 'updated_at']
 9
10   def create(self):
11     return self.cloud.create_network(name=uuid.uuid4().hex + 'sdk')
12
13   def get(self):
14     return self.cloud.get_network(self.obj.id)
15
16   def list(self):
17     return self.cloud.list_networks()
18
19   def search(self):
20     return self.cloud.search_networks(filters='[?contains(name, `sdk`)]')
21
22   def update(self, name):
23     return self.cloud.update_network(self.obj.id, name=name)
24
25   def delete(self):
26     self.cloud.delete_network(self.obj.id)
```

Source: Created by the author of this document.

Figure 82 - Code of the network test case suite

```
1 class NetworkTestCase(TestCase, unittest.TestCase):
2
3   manager_class = NetworkManager
```

Source: Created by the author of this document.

Subnetwork:

Figure 83 - Code of the subnetwork manager

```
class SubnetManager(Manager):

  @property
  def attributes(self):
    return ['allocation_pools', 'cidr', 'created_at', 'dns_nameservers', 'enable_dhcp',
            'gateway ip', 'host routes', 'id', 'ip version', 'ipv6 ra mode',
            'ipv6_address_mode', 'location', 'name', 'network_id', 'project_id',
            'properties', 'subnetpool_id', 'tenant_id', 'updated_at']

  def create(self):
    return self.cloud.create_subnet(network_name_or_id=self.network.obj,
                                    subnet_name=uuid.uuid4().hex + 'sdk',
                                    cidr='10.0.0.0/24', gateway_ip='10.0.0.1',
                                    enable_dhcp=True)

  def get(self):
    return self.cloud.get_subnet(self.obj.id)

  def list(self):
    return self.cloud.list_subnets()

  def search(self):
    return self.cloud.search_subnets(filters='[?contains(name, `sdk`)]')

  def update(self, name):
    return self.cloud.update_subnet(self.obj.id, subnet_name=name)

  def delete(self):
    self.cloud.delete_subnet(self.obj.id)

  def setup(self):
    self.network = NetworkManager()

  def cleanup(self):
    self.network.delete()
    self.network.cleanup()
```

Source: Created by the author of this document.

Figure 84 - Code of the subnetwork test case suite

```
class SubnetTestCase(TestCase, unittest.TestCase):

  manager_class = SubnetManager
```

Source: Created by the author of this document.

Security Group:

Figure 85 - Code of the security group manager

```
class SecurityGroupManager(Manager):

  @property
  def attributes(self):
    return ['location', 'id', 'name', 'security_group_rules', 'description',
            'properties']

  def create(self):
    return self.cloud.create_security_group(name=uuid.uuid4().hex + 'sdk',
                                            description=uuid.uuid4().hex)

  def get(self):
    return self.cloud.get_security_group(self.obj.id)

  def list(self):
    return self.cloud.list_security_groups()

  def search(self):
    return self.cloud.search_security_groups(filters='[?contains(name, `sdk`)]')

  def update(self, name):
    return self.cloud.update_security_group(self.obj.id, name=name)

  def delete(self):
    self.cloud.delete_security_group(self.obj.id)
```

Source: Created by the author of this document.

Figure 86 - Code of the security group test case suite

```
class SecurityGroupTestCase(TestCase, unittest.TestCase):

  manager_class = SecurityGroupManager
```

Source: Created by the author of this document.

E.2.4 Block Storage Code

The code for validating the operations of volumes is showed in this section. The manager implements the *attributes* property, which lists all the fields of the volume data model, as in the Section C.3 Block Storage of APPENDIX C – OPENSTACK SDK DATA MODEL.

Volume:

Figure 87 - Code of the volume manager

```
class VolumeManager(Manager):

  @property
  def attributes(self):
    return ['location', 'id', 'name', 'description', 'size', 'attachments', 'status',
            'migration status', 'host', 'replication driver', 'replication status',
            'replication_extended_status', 'snapshot_id', 'created_at', 'updated_at',
            'source_volume_id', 'consistencygroup_id', 'volume_type', 'metadata',
            'is_bootable', 'is_encrypted', 'can_multiattach', 'properties']

  def create(self):
    return self.cloud.create_volume(size=16, display_name=uuid.uuid4().hex + 'sdk',
                                    wait=True)

  def get(self):
    return self.cloud.get_volume(self.obj.id)

  def list(self):
    return self.cloud.list_volumes()

  def search(self):
    return self.cloud.search_volumes(filters='[?contains(display_name, `sdk`)]')

  def update(self, name):
    return self.cloud.update_volume(self.obj.id, name=name)

  def delete(self):
    self.cloud.delete_volume(self.obj.id, wait=True)
```

Source: Created by the author of this document.

Figure 88 - Code of the volume test case suite

```
class VolumeTestCase(TestCase, unittest.TestCase):

  manager_class = VolumeManager
```

Source: Created by the author of this document.

E.2.5 Compute Code

This subsection presents the code for validating the operations of flavor, keypair, private server, public server and snapshot resources. Each manager implements the *attributes* property that lists all the fields of the data model for the resources they manage, as defined in the Section C.4 Compute of APPENDIX C – OPENSTACK SDK DATA MODEL.

Flavor:

Figure 89 - Code of the flavor manager

```
class FlavorManager(Manager):

  @property
  def attributes(self):
    return ['location', 'id', 'name', 'is_public', 'is_disabled', 'ram', 'vcpus', 'disk',
            'ephemeral', 'swap', 'rxtx factor', 'properties', 'extra specs']

  def get(self):
    for ram in (256, 512, 1024, 2048):
      try:
        return self.cloud.search_flavors(filters={'ram': ram})[0]
      except IndexError:
        continue

  def list(self):
    return self.cloud.list_flavors()

  def search(self):
    return self.cloud.search_flavors(filters='[?name==`' + str(self.obj.name) + '`]')
```

Source: Created by the author of this document.

Figure 90 - Code of the flavor test case suite

```
class FlavorTestCase(TestCase, unittest.TestCase):

  manager_class = FlavorManager

  @unittest.skip('Flavors cannot be created')
  def test_create(self):
    pass

  @unittest.skip('Flavors cannot be updated')
  def test update(self):
    pass

  @unittest.skip('Flavors cannot be deleted')
  def test_delete(self):
    pass
```

Source: Created by the author of this document.

Keypair:

Figure 91 - Code of the keypair manager

```
class KeypairManager(Manager):

  @property
  def attributes(self):
    return ['location', 'type', 'created_at', 'id', 'properties', 'fingerprint',
            'name', 'private key', 'public key', 'user id']

  def create(self):
    return self.cloud.create_keypair(name=uuid.uuid4().hex + 'sdk')

  def get(self):
    return self.cloud.get_keypair(self.obj.id)

  def list(self):
    return self.cloud.list_keypairs()

  def search(self):
    return self.cloud.search_keypairs(filters='[?contains(name, `sdk`)]')

  def delete(self):
    self.cloud.delete_keypair(self.obj.id)
```

Source: Created by the author of this document.

Figure 92 - Code of the keypair test case suite

```
class KeypairTestCase(TestCase, unittest.TestCase):

  manager_class = KeypairManager

  @unittest.skip('Keypairs cannot be updated')
  def test_update(self):
    pass
```

Source: Created by the author of this document.

Private Server:

Figure 93 shows the code for the private server resource and, as documented in the Subsection 2.3.17, it depends on the flavor, image, keypair, security group, subnetwork and network resources created by their respective managers.

Figure 93 - Code of the private server manager

```
class PrivateServerManager(Manager):

  @property
  def attributes(self):
    return ['id', 'name', 'flavor', 'image', 'location', 'volumes', 'has_config_drive',
            'host_id', 'progress', 'disk_config', 'power_state', 'task_state',
            'vm_state', 'launched_at', 'terminated_at', 'security_groups', 'created_at',
            'interface_ip', 'properties', 'accessIPv4', 'accessIPv6', 'addresses',
            'adminPass', 'created', 'key_name', 'metadata', 'networks', 'private_v4',
            'public_v4', 'public_v6', 'status', 'updated', 'user_id']

  def create(self):
    return self.cloud.create_server(
        name=uuid.uuid4().hex + 'sdk', image=self.image.obj, flavor=self.flavor.obj,
        network=self.subnet.network.obj, key_name=self.keypair.obj.name,
        security_groups=[self.security_group.obj.name], auto_ip=False, wait=True)

  def get(self):
    return self.cloud.get_server(self.obj.id)

  def list(self):
    return self.cloud.list_servers()

  def search(self):
    return self.cloud.search_servers(filters='[?contains(name, `sdk`)]')

  def update(self, name):
    return self.cloud.update_server(self.obj.id, name=name)

  def delete(self):
    self.cloud.delete_server(self.obj.id, wait=True)

  def setup(self):
    self.flavor = FlavorManager()
    self.image = ImageManager()
    self.keypair = KeypairManager()
    self.security_group = SecurityGroupManager()
    self.subnet = SubnetManager()

  def cleanup(self):
    for manager in (self.image, self.keypair, self.security_group, self.subnet):
      manager.delete()
      manager.cleanup()
```

Source: Created by the author of this document.

Figure 94 - Code of the private server test case suite

```
class PrivateServerTestCase(TestCase, unittest.TestCase):

  manager_class = PrivateServerManager
```

Source: Created by the author of this document.

Public Server:

Figure 95 shows the code for the public server resource and, as documented in the Section 2.3.17, it only depends on the flavor and image resources created by their respective managers. The rest of the resources necessary, such as the network, are discovered by OpenStack SDK, which selects default resources.

Figure 95 - Code of the public server manager

```
 1 class PublicServerManager(PrivateServerManager):
 2
 3   def create(self):
 4     return self.cloud.create_server(name=uuid.uuid4().hex + 'sdk',image=self.image.obj,
 5                                     flavor=self.flavor.obj, auto_ip=True, wait=True)
 6
 7   def setup(self):
 8     self.flavor = FlavorManager()
 9     self.image = ImageManager()
10
11   def cleanup(self):
12     self.image.delete()
```

Source: Created by the author of this document.

Figure 96 - Code of the public server test case suite

```
1 class PublicServerTestCase(TestCase, unittest.TestCase):
2
3   manager_class = PublicServerManager
```

Source: Created by the author of this document.

Snapshot:

Figure 97 - Code of the snapshot manager

```
 1 class SnapshotManager(ImageManager):
 2
 3   def create(self):
 4     return self.cloud.create_image_snapshot(name=uuid.uuid4().hex + 'sdk-snapshot',
 5                                             server=self.server.obj.id, wait=True)
 6
 7   def search(self):
 8     return self.cloud.search_images(filters='[?contains(name, `sdk-snapshot`)]')
 9
10   def setup(self):
11     self.server = PrivateServerManager()
12
13   def cleanup(self):
14     self.server.delete()
15     self.server.cleanup()
```

Source: Created by the author of this document.

Figure 98 - Code of the snapshot test case suite

```
1 class SnapshotTestCase(TestCase, unittest.TestCase):
2
3   manager_class = SnapshotManager
```

Source: Created by the author of this document.

E.2.6 Object Storage Code

This section presents the code for validating the operations of containers and objects. The managers implement the *attributes* property, which lists all the fields of the data model for

the resources they manage, as in the Section C.5 Object Storage of APPENDIX C – OPENSTACK SDK DATA MODEL.

Container:

Figure 99 - Code of the container manager

```
class ContainerManager(Manager):

  @property
  def id(self):
    return 'name'

  @property
  def attributes(self):
    return ['bytes', 'count', 'last_modified', 'location', 'name', 'properties']

  def create(self):
    return self.cloud.create_container(name=uuid.uuid4().hex + 'sdk',
                                       return metadata=False)

  def get(self):
    return self.cloud.get_container(self.obj.name, return_metadata=False)

  def list(self):
    return self.cloud.list containers()

  def search(self):
    return self.cloud.search_containers(filters='[?contains(name, `sdk`)]')

  def delete(self):
    self.cloud.delete_container(self.obj.name)
```

Source: Created by the author of this document.

Figure 100 - Code of the container test case suite

```
class ContainerTestCase(TestCase, unittest.TestCase):

  manager_class = ContainerManager

  @unittest.skip('Object store cannot be updated')
  def test_update(self):
    pass
```

Source: Created by the author of this document.

Object:

Figure 101 - Code of the object manager

```
class ObjectManager(Manager):

  @property
  def id(self):
    return 'name'

  @property
  def attributes(self):
    return ['bytes', 'content_type', 'hash', 'last_modified', 'location', 'name',
            'properties']

  def create(self):
    return self.cloud.create_object(container=self.container.obj.name,
                                    name=uuid.uuid4().hex + 'sdk',
                                    filename='test_sdk.py', return_metadata=False)

  def get(self):
    response = self.cloud.get_object(container=self.container.obj.name,
                                     obj=self.obj.name, return_metadata=False)
    return response[0] if response else response

  def list(self):
    return self.cloud.list_objects(self.container.obj.name)

  def search(self):
    return self.cloud.search_objects(self.container.obj.name,
                                     filters='[?contains(name, `sdk`)]')

  def delete(self):
    self.cloud.delete_object(container=self.container.obj.name,
                             name=self.obj.name)

  def setup(self):
    self.container = ContainerManager()

  def cleanup(self):
    self.container.delete()
```

Source: Created by the author of this document.

Figure 102 - Code of the object test case suite

```
class ObjectTestCase(ContainerTestCase, unittest.TestCase):

  manager_class = ObjectManager
```

Source: Created by the author of this document.

E.3 Testing Environment

Since the OpenStack SDK is a client-side library, the automated tests will run in a local machine and then communicate with remote clouds, making assertions on the results of the HTTP API calls performed by the library. This section details all the steps necessary to prepare a local environment to run the developed tests against OpenStack clouds. The local machine was running *Ubuntu 16.04.4 LTS* as operating system and had the dependencies downloaded, configuration files and scripts to discover cloud services created, and further logging enabled.

E.3.1 Download Dependencies

Assuming Python 3.0 (Python 3.0 Release, 2008) or later is available and the tool for installing Python packages *pip* (pip Documentation, 2018) is installed, the first step is to install the latest version of *openstacksdk* (Openstack SDK package, 2018) SDK and *nose* (nose Package, 2018) packages, as in Figure 103. *nose* (nose Documentation, 2018) is a Python testing library that extends the default *unittest* (Unit testing framework, 2018) Python testing framework.

Figure 103 - Installing openstacksdk and nose

```
1 $ pip install openstacksdk
2 $ pip install nose
```

Source: Created by the author of this document.

With all the Python dependencies installed, the CirrOS (CirrOS, 2018) image, which is a minimal Linux distribution designed for use as test image on clouds, was downloaded and put in the same directory as *test_sdk.py*, which contains the testing code that references this image. In this study, the image on version *4.0.0* was downloaded from the CirrOS image directory (CirrOS Download Index, 2017) and saved locally as *cirros-0.4.0-x86_64-disk.img*. Any image of an operational system would, but CirrOS was chosen because it is lightweight and is open source.

E.3.2 Configuration File

OpenStack SDK searches for a file named */etc/openstack/clouds.yaml* to find authentication information that allow it to connect to clouds, as detailed in the Section 3.3.1. The example in Figure 104 would allow a user to connect to a cloud named *openstack* on the region *Brazil* as the user *samueldmq*, scoped to a project named *masters*, by authenticating against the specified URL.

Figure 104 - Example clouds.yaml file

```
clouds:
  openstack:
    region_name: Brazil
    auth:
      username: samueldmq
      password: ******
      project_name: masters
      auth_url: https://identity.example.com
```

Source: Created by the author of this document.

E.3.3 Cloud Services

For the validation process, once the local environment configuration is complete, it is important to understand what service versions are available in the remote cloud environments by exploring their service catalog via the service discovery process, both as detailed in APPENDIX E – TESTING CODE AND ENVIRONMENT.

To facilitate this process, the Python script in Figure 105 will allow printing the full service catalog, as exemplified in Figure 106.

Figure 105 - Script to print the service catalog

```
import openstack

cloud = openstack.connect('devstack')
print(cloud.service_catalog)
```

Source: Created by the author of this document.

Figure 106 - Example output of the service catalog script

```
[
  {
    'endpoints':[
      {
        'url': 'https://br.example.com:5000/',
        'interface': 'public',
        'region': 'BR',
        'id': 'cf7e04819c924891a848c16ea796e43e'
      }
    ],
    'type': 'identity',
    'id': '8f9c8ca448a246ca976b118b8324f3b8',
    'name': 'keystone'
  }
]
```

Source: Created by the author of this document.

Similarly, the script in Figure 107 discovers and prints the versioned URLs, as exemplified in Figure 108.

Figure 107 - Script to print the services versioned URLs

```
1 import openstack
2
3 cloud = openstack.connect('openstack')
4 for service in ('identity', 'compute', 'image', 'network',
5                 'block-storage', 'object-store'):
6   print(cloud.session.auth.get_endpoint_data(cloud.session, service).url)
```

Source: Created by the author of this document.

Figure 108 - Example output of the versioned URLs script

```
1 http://192.168.3.7/identity/v3/
2 http://192.168.3.7/compute/v2.1
3 http://192.168.3.7/image/v2/
4 http://192.168.3.7:9696/v2.0/
5 http://192.168.3.7/volume/v3/ba41486d1b694f069bb079ae66523af9
6 http://192.168.3.7:8080/v1/AUTH_ba41486d1b694f069bb079ae66523af9
```

Source: Created by the author of this document.

E.3.4 Logging

As detailed in the Subsection 3.3.12, a server may have a public IP assigned directly via DHCP or via NAT mechanism. To discover what mechanism was used when testing the server creation, a patch (Change 611869, 2018) was applied to the local OpenStack SDK code.

Similarly, as detailed in the Subsection 3.3.9, images may be created via the PUT API of the image service directly or via the Object Storage service and then imported into the image service via an import task. To discover what mechanism was used when testing the image creation, another patch (Change 611868, 2018) was applied to the local OpenStack SDK code.

E.4 Running Tests

When in the same directory of *test_sdk.py*, which contains all the testing code and the image file, the command in Figure 109 allows all the test cases to be discovered and run.

Figure 109 - Running all the test cases

```
1 $ CLOUD=devstack nosetests --with-timer -q -e TestCase
```

Source: Created by the author of this document.

A specific set of tests can be run by specifying the test case suite class with the *--tests* parameter of the nose framework, as exemplified in Figure 110 for the container test cases.

Figure 110 - Running a specific set of test cases

```
1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test_sdk.ContainerTestCase
 2 test_update (test_sdk.ContainerTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ContainerTestCase) ... ok
 4 test_get (test_sdk.ContainerTestCase) ... ok
 5 test_list (test_sdk.ContainerTestCase) ... ok
 6 test_search (test_sdk.ContainerTestCase) ... ok
 7 test_delete (test_sdk.ContainerTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 2.284s
11
12 OK (SKIP=1)
```

Source: Created by the author of this document.

Specifically, for public server and image tests, it is necessary to add the flag *--debug* with the logger name *test_sdk*, defined in the previous Section E.3.4 Logging. The example for the public server is shown in Figure 111, where the output of the logging is in line 9. That output is, in fact, repeated the number of times the logging code is reached, which means one per test case, but the repeated output is removed to make the reporting cleaner.

Figure 111 - Running public server test cases with debug enabled

```
1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test_sdk.PublicServerTestCase --
debug=test_sdk
 2 test_create (test_sdk.PublicServerTestCase) ... test_sdk: ok
 3 test_get (test_sdk.PublicServerTestCase) ... test_sdk: ok
 4 test_list (test_sdk.PublicServerTestCase) ... test_sdk: ok
 5 test_search (test_sdk.PublicServerTestCase) ... test_sdk: ok
 6 test_update (test_sdk.PublicServerTestCase) ... test_sdk: ok
 7 test_delete (test_sdk.PublicServerTestCase) ... test_sdk: ok
 8
 9 test_sdk: DEBUG: IP via NAT
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 221.810s
13
14 OK
```

Source: Created by the author of this document.

E.5 Authorization

The author asked for explicit authorization to mention the names of the cloud providers used in the validation. The communication was via e-mail with the providers Catalyst, VEXXHOST, Enter Cloud Suite, SWITCHengines and Ormuco.

APPENDIX F - VALIDATION ON DEVSTACK

This appendix describes the validation of OpenStack SDK against a DevStack (DevStack Documentation, 2018) cloud by running the tests from APPENDIX E – TESTING CODE AND ENVIRONMENT. The Section F.1 Deploy shows how a DevStack cloud was deployed on a remote virtual machine; Section F.2 Environment Setup details how the testing environment was configured; whereas the Section F.3 Test Results reports the tests execution and output.

F.1 Deploy

In this experiment, DevStack was installed on a remote virtual machine of 4 vCPUs, 32 GB of RAM and 10 GB of disk, running Ubuntu 16.04.2 LTS as operational system. Thanks to the hard work of the OpenStack community, deploying a DevStack cloud was a straightforward process performed and reproducible with the steps in Figure 112.

Figure 112 - Deploying a DevStack cloud

```
 1 $ sudo useradd -s /bin/bash -d /opt/stack -m stack
 2 $ echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
 3 $ sudo su - stack
 4 $ git clone https://git.openstack.org/openstack-dev/devstack
 5 $ cd devstack
 6 $ cat > local.conf <<- "EOF"
 7 [[local|localrc]]
 8 ADMIN_PASSWORD=******
 9 DATABASE_PASSWORD=$ADMIN_PASSWORD
10 RABBIT_PASSWORD=$ADMIN_PASSWORD
11 SERVICE_PASSWORD=$ADMIN_PASSWORD
12
13 enable_service s-proxy s-object s-container s-account
14 SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5
15 SWIFT_REPLICAS=1
16 SWIFT_DATA_DIR=$DEST/data
17 EOF
18 $ ./stack.sh
```

Source: DevStack upstream documentation (DevStack Documentation, 2018)

Lines 1 and 2 create a new user and assign super user privileges. Lines 3 to 5 log in as the new user, download the latest code and enter its directory. Lines 6 to 17 create the DevStack configuration file named *local.conf*, containing the passwords to be set when configuring services and explicitly enabling the Object Storage service, since it was disabled by default. Lastly, line 18 starts the deploy.

F.2 Environment Setup

The local environment was configured and the DevStack cloud services were discovered by following the guidelines defined in the Section E.3 Testing Environment. The local tests will use OpenStack SDK to communicate and test the DevStack cloud remotely. The next subsections detail the user authentication information and cloud services versions.

F.2.1 Authentication

The configuration details allowing OpenStack SDK to connect to the remote cloud are specified in its configuration file, located at */etc/openstack/clouds.yaml*, as in Figure 113. Lines 3 to 8 define how to authenticate to the cloud; line 9 specifies that the region to use is named *RegionOne*, the only one created by default in the DevStack setup.

Figure 113 - OpenStack SDK configuration for DevStack

```
1 clouds:
2   devstack:
3     auth:
4       auth_url: http://192.168.3.7/identity
5       project_id: ba41486d1b694f069bb079ae66523af9
6       username: samueldmq@gmail.com
7       user_domain_name: Default
8       password: ******
9     region_name: RegionOne
```

Source: Created by the author of this document.

F.2.2 Services

The cloud services are exposed in the service catalog, where each service may support multiple API versions. When using OpenStack SDK, the versions are automatically selected by the service discovery process, which outputs the versioned URLs. Both service catalog and discovery are described in APPENDIX B – SERVICE CATALOG AND DISCOVERY. For DevStack, the service discovery selects the versioned URLs as in Figure 114.

Figure 114 - Services versioned URLs discovery for DevStack

```
 1 $ python
 2 >>> import openstack
 3 >>> cloud = openstack.connect('devstack')
 4 >>> for service in ('identity', 'compute', 'image', 'network',
 5 ...                 'block-storage', 'object-store'):
 6 ...   print(cloud.session.auth.get_endpoint_data(cloud.session, service).url)
 7 ...
 8 http://192.168.3.7/identity/v3/
 9 http://192.168.3.7/compute/v2.1
10 http://192.168.3.7/image/v2/
11 http://192.168.3.7:9696/v2.0/
12 http://192.168.3.7/volume/v3/ba41486d1b694f069bb079ae66523af9
13 http://192.168.3.7:8080/v1/AUTH_ba41486d1b694f069bb079ae66523af9
```

Source: Created by the author of this document.

Therefore, the validation process will use those URLs, resulting in the final list of service versions for DevStack summarized in Table 25.

Table 25 - Service versions discovered for DevStack

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	3	2	2	3	2	1

Source: Created by the author of this document.

F.3 Test Results

This section reports the output of running the testing scripts from the Section E.2 Testing Code for Compute, Image, Networking, Block Storage and Object Storage resources against the DevStack cloud environment as specified in Section E.4 Running Tests. All tests ran successfully in this environment.

F.3.1 Image

The Section 3.3.9 detailed that there are two workflows to create an image: directly via the PUT API of the image service or by uploading the image to the Object Storage service and then importing it to the image service via the Tasks API. The DevStack cloud runs the former, as shown in line 9 of Figure 115, which outputs the custom logger print along with the successful execution of the image tests.

Figure 115 - Image tests output for DevStack

```
 1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test_sdk.ImageTestCase --debug=test_sdk
 2 test_create (test_sdk.ImageTestCase) ... ok
 3 test_get (test_sdk.ImageTestCase) ... ok
 4 test_list (test_sdk.ImageTestCase) ... ok
 5 test_search (test_sdk.ImageTestCase) ... ok
 6 test_update (test_sdk.ImageTestCase) ... ok
 7 test_delete (test_sdk.ImageTestCase) ... ok
 8
 9 test_sdk: DEBUG: Image via PUT
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 15.115s
13
14 OK
```

Source: Created by the author of this document.

F.3.2 Networking

As shown in Figure 116, Figure 117 and Figure 118, all the tests for network, subnetwork and security group, respectively, have run successfully.

Network:

Figure 116 - Network tests output for DevStack

```
 1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test_sdk.NetworkTestCase
 2 test_create (test_sdk.NetworkTestCase) ... ok
 3 test_get (test_sdk.NetworkTestCase) ... ok
 4 test_list (test_sdk.NetworkTestCase) ... ok
 5 test_search (test_sdk.NetworkTestCase) ... ok
 6 test_update (test_sdk.NetworkTestCase) ... ok
 7 test_delete (test_sdk.NetworkTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 18.705s
11
12 OK
```

Source: Created by the author of this document.

Subnetwork:

Figure 117 - Subnetwork tests output for DevStack

```
 1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test sdk.SubnetTestCase
 2 test_create (test_sdk.SubnetTestCase) ... ok
 3 test_get (test_sdk.SubnetTestCase) ... ok
 4 test_list (test_sdk.SubnetTestCase) ... ok
 5 test_search (test_sdk.SubnetTestCase) ... ok
 6 test update (test sdk.SubnetTestCase) ... ok
 7 test_delete (test_sdk.SubnetTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 45.188s
11
12 OK
```

Source: Created by the author of this document.

Security Group:

Figure 118 - Security group tests output for DevStack

```
 1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test_sdk.SecurityGroupTestCase
 2 test create (test sdk.SecurityGroupTestCase) ... ok
 3 test_get (test_sdk.SecurityGroupTestCase) ... ok
 4 test_list (test_sdk.SecurityGroupTestCase) ... ok
 5 test_search (test_sdk.SecurityGroupTestCase) ... ok
 6 test_update (test_sdk.SecurityGroupTestCase) ... ok
 7 test_delete (test_sdk.SecurityGroupTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 7.230s
11
12 OK
```

Source: Created by the author of this document.

F.3.3 Block Storage

Volume:

Figure 119 - Volume tests output for DevStack

```
 1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test_sdk.VolumeTestCase
 2 test_create (test_sdk.VolumeTestCase) ... ok
 3 test_get (test_sdk.VolumeTestCase) ... ok
 4 test_list (test_sdk.VolumeTestCase) ... ok
 5 test_search (test_sdk.VolumeTestCase) ... ok
 6 test_update (test_sdk.VolumeTestCase) ... ok
 7 test_delete (test_sdk.VolumeTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 40.861s
11
12 OK
```

Source: Created by the author of this document.

F.3.4 Compute

As shown in Figure 120, Figure 121, Figure 122, Figure 123 and Figure 124, all the tests for flavor, keypair, private server, public server and snapshot, respectively, have run successfully.

Flavor:

Figure 120 - Flavor tests output for DevStack

```
$ CLOUD=devstack nosetests -v -e TestCase --tests=test_sdk.FlavorTestCase
test_create (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be created
test_delete (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be deleted
test_update (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be updated
test_get (test_sdk.FlavorTestCase) ... ok
test_list (test_sdk.FlavorTestCase) ... ok
test_search (test_sdk.FlavorTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 3.952s

OK (SKIP=3)
```

Source: Created by the author of this document.

Keypair:

Figure 121 - Keypair tests output for DevStack

```
$ CLOUD=devstack nosetests -v -e TestCase --tests=test_sdk.KeypairTestCase
test_update (test_sdk.KeypairTestCase) ... SKIP: Keypairs cannot be updated
test_create (test_sdk.KeypairTestCase) ... ok
test_get (test_sdk.KeypairTestCase) ... ok
test_list (test_sdk.KeypairTestCase) ... ok
test_search (test_sdk.KeypairTestCase) ... ok
test_delete (test_sdk.KeypairTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 6.349s

OK (SKIP=1)
```

Source: Created by the author of this document.

Private Server:

Figure 122 - Private server tests output for DevStack

```
 1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test sdk.PrivateServerTestCase
 2 test_create (test_sdk.PrivateServerTestCase) ... ok
 3 test_get (test_sdk.PrivateServerTestCase) ... ok
 4 test_list (test_sdk.PrivateServerTestCase) ... ok
 5 test_search (test_sdk.PrivateServerTestCase) ... ok
 6 test update (test sdk.PrivateServerTestCase) ... ok
 7 test_delete (test_sdk.PrivateServerTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 245.279s
11
12 OK
```

Source: Created by the author of this document.

Public Server:

The section Compute Functions detailed that there are two workflows to create a server and get a public IP address assigned to it, the first is when the IP is directly routable via DHCP and the second is via NAT mechanism. The DevStack cloud runs the latter, as shown in line 9, which outputs the custom logger print.

Figure 123 - Public server tests output for DevStack

```
1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test sdk.PublicServerTestCase --
debug=test_sdk
 2 test_create (test_sdk.PublicServerTestCase) ... test_sdk: ok
 3 test_get (test_sdk.PublicServerTestCase) ... test_sdk: ok
 4 test_list (test_sdk.PublicServerTestCase) ... test_sdk: ok
 5 test_search (test_sdk.PublicServerTestCase) ... test_sdk: ok
 6 test update (test sdk.PublicServerTestCase) ... test sdk: ok
 7 test_delete (test_sdk.PublicServerTestCase) ... test_sdk: ok
 8
 9 test_sdk: DEBUG: IP via NAT
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 221.810s
13
14 OK
```

Source: Created by the author of this document.

Snapshot:

Figure 124 - Snapshot tests output for DevStack

```
 1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test sdk.SnapshotTestCase
 2 test_create (test_sdk.SnapshotTestCase) ... ok
 3 test_get (test_sdk.SnapshotTestCase) ... ok
 4 test_list (test_sdk.SnapshotTestCase) ... ok
 5 test_search (test_sdk.SnapshotTestCase) ... ok
 6 test_update (test_sdk.SnapshotTestCase) ... ok
 7 test_delete (test_sdk.SnapshotTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 288.442s
11
12 OK
```

Source: Created by the author of this document.

F.3.5 Object Storage

As shown in Figure 125 and Figure 126, all the tests for container and object, respectively, have run successfully.

Container:

Figure 125 - Container tests output for DevStack

```
 1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test_sdk.ContainerTestCase
 2 test_update (test_sdk.ContainerTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ContainerTestCase) ... ok
 4 test_get (test_sdk.ContainerTestCase) ... ok
 5 test_list (test_sdk.ContainerTestCase) ... ok
 6 test_search (test_sdk.ContainerTestCase) ... ok
 7 test_delete (test_sdk.ContainerTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 2.284s
11
12 OK (SKIP=1)
```

Source: Created by the author of this document.

Object:

Figure 126 - Object tests output for DevStack

```
 1 $ CLOUD=devstack nosetests -v -e TestCase --tests=test sdk.ObjectTestCase
 2 test_update (test_sdk.ObjectTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ObjectTestCase) ... ok
 4 test_get (test_sdk.ObjectTestCase) ... ok
 5 test_list (test_sdk.ObjectTestCase) ... ok
 6 test search (test sdk.ObjectTestCase) ... ok
 7 test_delete (test_sdk.ObjectTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 4.655s
11
12 OK (SKIP=1)
```

Source: Created by the author of this document.

APPENDIX G – VALIDATION ON CATALYST

This appendix describes the validation of OpenStack SDK against the Catalyst cloud by running the tests from APPENDIX E – TESTING CODE AND ENVIRONMENT. The Section G.1 Environment Setup details how the testing environment was configured; whereas G.2 Test Results reports the tests execution and output.

G.1 Environment Setup

The local environment was configured, and the Catalyst cloud services were discovered by following the guidelines defined in the Section E.3 Testing Environment. The local tests will use OpenStack SDK to communicate and test the Catalyst cloud remotely. The next subsections detail the user authentication information and cloud services versions.

G.1.1 Authentication

The configuration details allowing OpenStack SDK to connect to the remote cloud are specified in its configuration file, located at */etc/openstack/clouds.yaml*, as in Figure 127. Lines 3 to 8 define how to authenticate to the cloud; line 9 specifies that the region to use is the one in Hamilton, New Zealand; while line 10 sets the URL to communicate to the Block Storage service, forcing the use of the version 1 to increase version coverage in the validation.

Figure 127 - OpenStack SDK configuration for Catalyst

```
 1 clouds:
 2   catalyst:
 3     auth:
 4       auth_url: https://api.nz-hlz-1.catalystcloud.io:5000/v3
 5       username: samueldmq@gmail.com
 6       project_id: 16ccee5e96ed466d86b23a5b7d95d377
 7       user_domain_name: Default
 8       password: ******
 9     region_name: nz-hlz-1
10     volume_endpoint: https://api.cloud.catalyst.net.nz:8776/v1/16cce***5d377
```

Source: Created by the author of this document.

G.1.2 Services

The cloud services are exposed in the service catalog, where each service may support multiple API versions. When using OpenStack SDK, the versions are automatically selected by the

service discovery process, which outputs the versioned URLs. Both service catalog and discovery are described in APPENDIX B – SERVICE CATALOG AND DISCOVERY. For Catalyst, the service discovery selects the versioned URLs as in Figure 128.

Figure 128 - Services versioned URLs discovery for Catalyst

```
 1 $ python
 2 >>> import openstack
 3 >>> cloud = openstack.connect('catalyst')
 4 >>> for service in ('identity', 'compute', 'image', 'network',
 5 ...                 'block-storage', 'object-store'):
 6 ...   print(cloud.session.auth.get_endpoint_data(cloud.session, service).url)
 7 ...
 8 https://api.nz-hlz-1.catalystcloud.io:5000/v3/
 9 https://api.nz-hlz-1.catalystcloud.io:8774/v2/16cce***5d377
10 https://api.nz-por-1.catalystcloud.io:9292/v2/
11 https://api.nz-hlz-1.catalystcloud.io:9696/v2.0
12 https://api.nz-hlz-1.catalystcloud.io:8776/v3/16cce***5d377
13 https://object-storage.nz-hlz-1.catalystcloud.io:443/v1/AUTH_16cce***5d377
```

Source: Created by the author of this document.

Therefore, the validation process will use those URLs, except for the Block Storage service that had its versioned URL explicitly set in the configuration file, resulting in the final list of service versions for Catalyst summarized in Table 26.

Table 26 - Service versions discovered for Catalyst

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	3	2	2	1	2	1

Source: Created by the author of this document.

G.2 Test Results

This section reports the output of running the testing scripts from the Section E.2 Testing Code for Compute, Image, Networking, Block Storage and Object Storage resources against the Catalyst cloud environment as specified in Section E.4 Running Tests. All tests ran successfully in this environment.

G.2.1 Image

The Section 3.3.9 detailed that there are two workflows to create an image: directly via the PUT API of the image service or by uploading the image to the Object Storage service and then importing it to the image service via the Tasks API. The Catalyst cloud runs the former,

as shown in line 9 of Figure 129, which outputs the custom logger print along with the successful execution of the image tests.

Figure 129 - Image tests output for Catalyst

```
 1 $ CLOUD=catalyst nosetests -v -e TestCase --tests=test_sdk.ImageTestCase --debug=test_sdk
 2 test_create (test_sdk.ImageTestCase) ... ok
 3 test_get (test_sdk.ImageTestCase) ... ok
 4 test_list (test_sdk.ImageTestCase) ... ok
 5 test_search (test_sdk.ImageTestCase) ... ok
 6 test_update (test_sdk.ImageTestCase) ... ok
 7 test_delete (test_sdk.ImageTestCase) ... ok
 8
 9 test_sdk: DEBUG: Image via PUT
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 57.184s
13
14 OK
```

Source: Created by the author of this document.

G.2.2 Networking

As shown in Figure 130, Figure 131 and Figure 132, all the tests for network, subnetwork and security group, respectively, have run successfully.

Network:

Figure 130 - Network tests output for Catalyst

```
 1 $ CLOUD=catalyst nosetests -v -e TestCase --tests=test_sdk.NetworkTestCase
 2 test_create (test_sdk.NetworkTestCase) ... ok
 3 test_get (test_sdk.NetworkTestCase) ... ok
 4 test_list (test_sdk.NetworkTestCase) ... ok
 5 test_search (test_sdk.NetworkTestCase) ... ok
 6 test_update (test_sdk.NetworkTestCase) ... ok
 7 test_delete (test_sdk.NetworkTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 8.695s
11
12 OK
```

Source: Created by the author of this document.

Subnetwork:

Figure 131 - Subnetwork tests output for Catalyst

```
 1 $ CLOUD=catalyst nosetests -v -e TestCase --tests=test sdk.SubnetTestCase
 2 test_create (test_sdk.SubnetTestCase) ... ok
 3 test_get (test_sdk.SubnetTestCase) ... ok
 4 test_list (test_sdk.SubnetTestCase) ... ok
 5 test_search (test_sdk.SubnetTestCase) ... ok
 6 test update (test sdk.SubnetTestCase) ... ok
 7 test_delete (test_sdk.SubnetTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 11.678s
11
12 OK
```

Source: Created by the author of this document.

Security Group:

Figure 132 - Security group tests output for Catalyst

```
 1 $ CLOUD=catalyst nosetests -v -e TestCase --tests=test_sdk.SecurityGroupTestCase
 2 test create (test sdk.SecurityGroupTestCase) ... ok
 3 test_get (test_sdk.SecurityGroupTestCase) ... ok
 4 test_list (test_sdk.SecurityGroupTestCase) ... ok
 5 test_search (test_sdk.SecurityGroupTestCase) ... ok
 6 test_update (test_sdk.SecurityGroupTestCase) ... ok
 7 test_delete (test_sdk.SecurityGroupTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 4.506s
11
12 OK
```

Source: Created by the author of this document.

G.2.3 Block Storage

Volume:

Figure 133 - Volume tests output for Catalyst

```
 1 $ CLOUD=catalyst nosetests -v -e TestCase --tests=test_sdk.VolumeTestCase
 2 test_create (test_sdk.VolumeTestCase) ... ok
 3 test_get (test_sdk.VolumeTestCase) ... ok
 4 test_list (test_sdk.VolumeTestCase) ... ok
 5 test_search (test_sdk.VolumeTestCase) ... ok
 6 test_update (test_sdk.VolumeTestCase) ... ok
 7 test_delete (test_sdk.VolumeTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 59.039s
11
12 OK
```

Source: Created by the author of this document.

G.2.4 Compute

As shown in Figure 134, Figure 135, Figure 136, Figure 137 and Figure 138, all the tests for flavor, keypair, private server, public server and snapshot, respectively, have run successfully.

Flavor:

Figure 134 - Flavor tests output for Catalyst

```
$ CLOUD=catalyst nosetests -v -e TestCase --tests=test_sdk.FlavorTestCase
test_create (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be created
test_delete (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be deleted
test_update (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be updated
test_get (test_sdk.FlavorTestCase) ... ok
test_list (test_sdk.FlavorTestCase) ... ok
test_search (test_sdk.FlavorTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 18.432s

OK (SKIP=3)
```

Source: Created by the author of this document.

Keypair:

Figure 135 - Keypair tests output for Catalyst

```
$ CLOUD=catalyst nosetests -v -e TestCase --tests=test_sdk.KeypairTestCase
test_update (test_sdk.KeypairTestCase) ... SKIP: Keypairs cannot be updated
test_create (test_sdk.KeypairTestCase) ... ok
test_get (test_sdk.KeypairTestCase) ... ok
test_list (test_sdk.KeypairTestCase) ... ok
test_search (test_sdk.KeypairTestCase) ... ok
test_delete (test_sdk.KeypairTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 11.499s

OK (SKIP=1)
```

Source: Created by the author of this document.

Private Server:

Figure 136 - Private server tests output for Catalyst

```
 1 $ CLOUD=catalyst nosetests -v -e TestCase --tests=test sdk.PrivateServerTestCase
 2 test_create (test_sdk.PrivateServerTestCase) ... ok
 3 test_get (test_sdk.PrivateServerTestCase) ... ok
 4 test_list (test_sdk.PrivateServerTestCase) ... ok
 5 test_search (test_sdk.PrivateServerTestCase) ... ok
 6 test update (test sdk.PrivateServerTestCase) ... ok
 7 test_delete (test_sdk.PrivateServerTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 240.201s
11
12 OK
```

Source: Created by the author of this document.

Public Server:

The Section 3.3.12 detailed that there are two workflows to create a server and get a public IP address assigned to it, the first is when the IP is directly routable via DHCP and the second is via NAT mechanism. The Catalyst cloud runs the latter, as shown in line 9, which outputs the custom logger print.

Figure 137 - Public server tests output for Catalyst

```
 1 $ CLOUD=catalyst nosetests -v -e TestCase --tests=test sdk.PublicServerTestCase --
debug=test_sdk
 2 test_create (test_sdk.PublicServerTestCase) ... ok
 3 test_get (test_sdk.PublicServerTestCase) ... ok
 4 test_list (test_sdk.PublicServerTestCase) ... ok
 5 test_search (test_sdk.PublicServerTestCase) ... ok
 6 test update (test sdk.PublicServerTestCase) ... ok
 7 test_delete (test_sdk.PublicServerTestCase) ... ok
 8
 9 test_sdk: DEBUG: IP via NAT
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 212.492s
13
14 OK
```

Source: Created by the author of this document.

Snapshot:

Figure 138 - Snapshot tests output for Catalyst

```
 1 $ CLOUD=catalyst nosetests -v -e TestCase --tests=test sdk.SnapshotTestCase
 2 test_create (test_sdk.SnapshotTestCase) ... ok
 3 test_get (test_sdk.SnapshotTestCase) ... ok
 4 test_list (test_sdk.SnapshotTestCase) ... ok
 5 test_search (test_sdk.SnapshotTestCase) ... ok
 6 test update (test sdk.SnapshotTestCase) ... ok
 7 test_delete (test_sdk.SnapshotTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 1090.233s
11
12 OK
```

Source: Created by the author of this document.

G.2.5 Object Storage

As shown in Figure 139 and Figure 140, all the tests for container and object, respectively, have run successfully.

Container:

Figure 139 - Container tests output for Catalyst

```
 1 $ CLOUD=catalyst nosetests -v -e TestCase --tests=test_sdk.ContainerTestCase
 2 test_update (test_sdk.ContainerTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ContainerTestCase) ... ok
 4 test_get (test_sdk.ContainerTestCase) ... ok
 5 test_list (test_sdk.ContainerTestCase) ... ok
 6 test_search (test_sdk.ContainerTestCase) ... ok
 7 test_delete (test_sdk.ContainerTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 3.263s
11
12 OK (SKIP=1)
```

Source: Created by the author of this document.

Object:

Figure 140 - Object tests output for Catalyst

```
 1 $ CLOUD=catalyst nosetests -v -e TestCase --tests=test sdk.ObjectTestCase
 2 test_update (test_sdk.ObjectTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ObjectTestCase) ... ok
 4 test_get (test_sdk.ObjectTestCase) ... ok
 5 test_list (test_sdk.ObjectTestCase) ... ok
 6 test search (test sdk.ObjectTestCase) ... ok
 7 test_delete (test_sdk.ObjectTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 7.804s
11
12 OK (SKIP=1)
```

Source: Created by the author of this document.

APPENDIX H – VALIDATION ON VEXXHOST

This appendix describes the validation of OpenStack SDK against the VEXXHOST cloud by running the tests from APPENDIX E – TESTING CODE AND ENVIRONMENT. The Section H.1 Environment Setup details how the testing environment was configured; whereas H.2 Test Results reports the tests execution and output.

H.1 Environment Setup

The local environment was configured and the VEXXHOST cloud services were discovered by following the guidelines defined in the Section E.3 Testing Environment. The local tests will use OpenStack SDK to communicate and test the VEXXHOST cloud remotely. The next subsections detail the user authentication information and cloud services versions.

H.1.1 Authentication

The configuration details allowing OpenStack SDK to connect to the remote cloud are specified in its configuration file, located at */etc/openstack/clouds.yaml*, as in Figure 141. Lines 3 to 8 define how to authenticate to the cloud; line 9 specifies that the region to use is the one in Santa Clara, California.

Figure 141 - OpenStack SDK configuration for VEXXHOST

```
1 clouds:
2   vexxhost:
3     auth:
4       auth_url: https://auth.vexxhost.net
5       project_id: 70dadafa31184691b8f1a97c95bcfa1b
6       username: samueldmq@gmail.com
7       user_domain_name: Default
8       password: ******
9     region_name: sjc1
```

Source: Created by the author of this document.

H.1.2 Services

The cloud services are exposed in the service catalog, where each service may support multiple API versions. When using OpenStack SDK, the versions are automatically selected by the service discovery process, which outputs the versioned URLs. Both service catalog and

discovery are described in APPENDIX B – SERVICE CATALOG AND DISCOVERY. For VEXXHOST, the service discovery selects the versioned URLs as in Figure 142.

Figure 142 - Services versioned URLs discovery for VEXXHOST

```
$ python
>>> import openstack
>>> cloud = openstack.connect('vexxhost')
>>> for service in ('identity', 'compute', 'image', 'network',
...                 'block-storage', 'object-store'):
...   print(cloud.session.auth.get_endpoint_data(cloud.session, service).url)
...
https://auth.vexxhost.net/v3/
https://compute-ca-ymq-1.vexxhost.net/v2.1
https://image-ca-ymq-1.vexxhost.net/v2/
https://network-ca-ymq-1.vexxhost.net/v2.0/
https://block-storage-ca-ymq-1.vexxhost.us/v3/70dadafa31184691b8f1a97c95bcfa1b
https://object-storage-ca-ymq-1.vexxhost.net/v1/70dadafa31184691b8f1a97c95bcfa1b
```

Source: Created by the author of this document.

Therefore, the final list of service versions for VEXXHOST shown in Table 27. The script has discovered the URLs for the cloud region in Montreal, while the one in Santa Clara is used in this validation. The versions will be the same, although the Object Storage service does not exist in that region, as it will be further discussed in the next section in the results for container and object resources.

Table 27 - Service versions discovered for VEXXHOST

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	3	2	2	3	2	N/A

Source: Created by the author of this document.

H.2 Test Results

This section reports the output of running the testing scripts from E.2 Testing Code for Compute, Image, Networking, Block Storage and Object Storage resources against the VEXXHOST cloud environment as specified in the Section E.4 Running Tests. All tests but the Object Storage test cases ran successfully in this environment.

H.2.1 Image

The Section 3.3.9 detailed that there are two workflows to create an image: directly via the PUT API of the image service or by uploading the image to the Object Storage service and

then importing it to the image service via the Tasks API. The VEXXHOST cloud runs the former, as shown in line 9 of Figure 143, which outputs the custom logger print along with the successful execution of the image tests.

Figure 143 - Image tests output for VEXXHOST

```
 1 $ CLOUD=vexxhost nosetests -v -e TestCase --tests=test_sdk.ImageTestCase
 2 test_create (test_sdk.ImageTestCase) ... ok
 3 test_get (test_sdk.ImageTestCase) ... ok
 4 test_list (test_sdk.ImageTestCase) ... ok
 5 test_search (test_sdk.ImageTestCase) ... ok
 6 test_update (test_sdk.ImageTestCase) ... ok
 7 test_delete (test_sdk.ImageTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 26.231s
11
12 OK
```

Source: Created by the author of this document.

H.2.2 Networking

As shown in Figure 144, Figure 145 and Figure 146, all the tests for network, subnetwork and security group, respectively, have run successfully.

Network:

Figure 144 - Network tests output for VEXXHOST

```
 1 $ CLOUD=vexxhost nosetests -v -e TestCase --tests=test_sdk.NetworkTestCase
 2 test_create (test_sdk.NetworkTestCase) ... ok
 3 test_get (test_sdk.NetworkTestCase) ... ok
 4 test_list (test_sdk.NetworkTestCase) ... ok
 5 test_search (test_sdk.NetworkTestCase) ... ok
 6 test_update (test_sdk.NetworkTestCase) ... ok
 7 test_delete (test_sdk.NetworkTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 20.881s
11
12 OK
```

Source: Created by the author of this document.

Subnetwork:

Figure 145 - Subnetwork tests output for VEXXHOST

```
 1 $ CLOUD=vexxhost nosetests -v -e TestCase --tests=test sdk.SubnetTestCase
 2 test_create (test_sdk.SubnetTestCase) ... ok
 3 test_get (test_sdk.SubnetTestCase) ... ok
 4 test_list (test_sdk.SubnetTestCase) ... ok
 5 test_search (test_sdk.SubnetTestCase) ... ok
 6 test update (test sdk.SubnetTestCase) ... ok
 7 test_delete (test_sdk.SubnetTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 47.674s
11
12 OK
```

Source: Created by the author of this document.

Security Group:

Figure 146 - Security group tests output for VEXXHOST

```
 1 $ CLOUD=vexxhost nosetests -v -e TestCase --tests=test_sdk.SecurityGroupTestCase
 2 test create (test sdk.SecurityGroupTestCase) ... ok
 3 test_get (test_sdk.SecurityGroupTestCase) ... ok
 4 test_list (test_sdk.SecurityGroupTestCase) ... ok
 5 test_search (test_sdk.SecurityGroupTestCase) ... ok
 6 test_update (test_sdk.SecurityGroupTestCase) ... ok
 7 test_delete (test_sdk.SecurityGroupTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 14.451s
11
12 OK
```

Source: Created by the author of this document.

H.2.3 Block Storage

Volume:

Figure 147 - Volume tests output for VEXXHOST

```
 1 $ CLOUD=vexxhost nosetests -v -e TestCase --tests=test_sdk.VolumeTestCase
 2 test_create (test_sdk.VolumeTestCase) ... ok
 3 test_get (test_sdk.VolumeTestCase) ... ok
 4 test_list (test_sdk.VolumeTestCase) ... ok
 5 test_search (test_sdk.VolumeTestCase) ... ok
 6 test_update (test_sdk.VolumeTestCase) ... ok
 7 test_delete (test_sdk.VolumeTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 52.971s
11
12 OK
```

Source: Created by the author of this document.

H.2.4 Compute

As shown in Figure 148, Figure 149, Figure 150, Figure 151 and Figure 152, all the tests for flavor, keypair, private server, public server and snapshot, respectively, have run successfully.

Flavor:

Figure 148 - Flavor tests output for VEXXHOST

```
$ CLOUD=vexxhost nosetests -v -e TestCase --tests=test_sdk.FlavorTestCase
test_create (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be created
test_delete (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be deleted
test_update (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be updated
test_get (test_sdk.FlavorTestCase) ... ok
test_list (test_sdk.FlavorTestCase) ... ok
test_search (test_sdk.FlavorTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 21.014s

OK (SKIP=3)
```

Source: Created by the author of this document.

Keypair:

Figure 149 - Keypair tests output for VEXXHOST

```
$ CLOUD=vexxhost nosetests -v -e TestCase --tests=test_sdk.KeypairTestCase
test_update (test_sdk.KeypairTestCase) ... SKIP: Keypairs cannot be updated
test_create (test_sdk.KeypairTestCase) ... ok
test_get (test_sdk.KeypairTestCase) ... ok
test_list (test_sdk.KeypairTestCase) ... ok
test_search (test_sdk.KeypairTestCase) ... ok
test_delete (test_sdk.KeypairTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 9.494s

OK (SKIP=1)
```

Source: Created by the author of this document.

Private Server:

Figure 150 - Private server tests output for VEXXHOST

```
 1 $ CLOUD=vexxhost nosetests -v -e TestCase --tests=test sdk.PrivateServerTestCase
 2 test_create (test_sdk.PrivateServerTestCase) ... ok
 3 test_get (test_sdk.PrivateServerTestCase) ... ok
 4 test_list (test_sdk.PrivateServerTestCase) ... ok
 5 test_search (test_sdk.PrivateServerTestCase) ... ok
 6 test update (test sdk.PrivateServerTestCase) ... ok
 7 test_delete (test_sdk.PrivateServerTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 278.711s
11
12 OK
```

Source: Created by the author of this document.

Public Server:

The Section 3.3.12 detailed that there are two workflows to create a server and get a public IP address assigned to it, the first is when the IP is directly routable via DHCP and the second is via NAT mechanism. The VEXXHOST cloud runs the former, as shown in line 9, which outputs the custom logger print.

Figure 151 - Public server tests output for VEXXHOST

```
 1 $ CLOUD=vexxhost nosetests -v -e TestCase --tests=test sdk.PublicServerTestCase --
debug=test_sdk
 2 test_create (test_sdk.PublicServerTestCase) ... ok
 3 test_get (test_sdk.PublicServerTestCase) ... ok
 4 test_list (test_sdk.PublicServerTestCase) ... ok
 5 test_search (test_sdk.PublicServerTestCase) ... ok
 6 test update (test sdk.PublicServerTestCase) ... ok
 7 test_delete (test_sdk.PublicServerTestCase) ... ok
 8
 9 test_sdk: DEBUG: IP via DHCP
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 212.689s
13
14 OK
```

Source: Created by the author of this document.

Snapshot:

Figure 152 - Snapshot tests output for VEXXHOST

```
 1 $ CLOUD=vexxhost nosetests -v -e TestCase --tests=test sdk.SnapshotTestCase
 2 test_create (test_sdk.SnapshotTestCase) ... ok
 3 test_get (test_sdk.SnapshotTestCase) ... ok
 4 test_list (test_sdk.SnapshotTestCase) ... ok
 5 test_search (test_sdk.SnapshotTestCase) ... ok
 6 test update (test sdk.SnapshotTestCase) ... ok
 7 test_delete (test_sdk.SnapshotTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 328.088s
11
12 OK
```

Source: Created by the author of this document.

H.2.5 Object Storage

As shown in Figure 153 and Figure 154, all the tests for container and object, respectively, have failed. The failure output explains that no public endpoint for the object storage service was found in the Santa Clara region, which means this service is not available in this cloud deployment.

Container:

Figure 153 - Container tests output for VEXXHOST

```
 1 $ CLOUD=vexxhost nosetests -v -e TestCase --tests=test_sdk.ContainerTestCase
 2 test_update (test_sdk.ContainerTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ContainerTestCase) ... ERROR
 4 test_get (test_sdk.ContainerTestCase) ... ERROR
 5 test_list (test_sdk.ContainerTestCase) ... ERROR
 6 test_search (test_sdk.ContainerTestCase) ... ERROR
 7 test_delete (test_sdk.ContainerTestCase) ... ERROR
 8
 9 keystoneauth1.exceptions.catalog.EndpointNotFound: public endpoint for object-store
service in sjc1 region not found
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 3.377s
13
14 FAILED (SKIP=1, errors=5)
```

Source: Created by the author of this document.

Object:

Figure 154 - Object tests output for VEXXHOST

```
 1 $ CLOUD=vexxhost nosetests -v -e TestCase --tests=test sdk.ObjectTestCase
 2 test_update (test_sdk.ObjectTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ObjectTestCase) ... ERROR
 4 test_get (test_sdk.ObjectTestCase) ... ERROR
 5 test_list (test_sdk.ObjectTestCase) ... ERROR
 6 test search (test sdk.ObjectTestCase) ... ERROR
 7 test_delete (test_sdk.ObjectTestCase) ... ERROR
 8
 9 keystoneauth1.exceptions.catalog.EndpointNotFound: public endpoint for object-store
service in sjc1 region not found
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 3.228s
13
14 FAILED (SKIP=1, errors=5)
```

Source: Created by the author of this document.

APPENDIX I – VALIDATION ON ENTER CLOUD SUITE

This appendix describes the validation of OpenStack SDK against the Enter Cloud Suite cloud by running the tests from APPENDIX E – TESTING CODE AND ENVIRONMENT. The Section I.1 Environment Setup details how the testing environment was configured; whereas I.2 Test Results reports the tests execution and output.

I.1 Environment Setup

The local environment was configured and the Enter Cloud Suite cloud services were discovered by following the guidelines defined in the Section E.3 Testing Environment. The local tests will use OpenStack SDK to communicate and test the Enter Cloud Suite cloud remotely. The next subsections detail the user authentication information and cloud services versions.

I.1.1 Authentication

The configuration details allowing OpenStack SDK to connect to the remote cloud are specified in its configuration file, located at */etc/openstack/clouds.yaml*, as in Figure 155. Lines 3 to 7 define how to authenticate to the cloud; line 8 specifies that the region to use is the one in Amsterdam, Netherlands; while line 9 sets the URL to communicate to the Image service, forcing the use of the version 1 to increase version coverage in the validation.

Figure 155 - OpenStack SDK configuration for Enter Cloud Suite

```
1 clouds:
2   entercloudsuite:
3     auth:
4       auth_url: https://api.nl-ams1.entercloudsuite.com
5       project_id: cb7088c9306742858e8a38b97744fba8
6       username: samueldmq@gmail.com
7       password: OpenStack!2018
8     region_name: nl-ams1
9     image_endpoint: http://glance.nl-ams1.entercloudsuite.com/v1/
```

Source: Created by the author of this document.

I.1.2 Services

The cloud services are exposed in the service catalog, where each service may support multiple API versions. When using OpenStack SDK, the versions are automatically selected by the service discovery process, which outputs the versioned URLs. Both service catalog and discovery are described in APPENDIX B – SERVICE CATALOG AND DISCOVERY. For Enter Cloud Suite, the service discovery selects the versioned URLs as in Figure 156.

Figure 156 - Services versioned URLs discovery for Enter Cloud Suite

```
 1 $ python
 2 >>> import openstack
 3 >>> cloud = openstack.connect('entercloudsuite')
 4 >>> for service in ('identity', 'compute', 'image', 'network',
 5 ...                 'block-storage', 'object-store'):
 6 ...   print(cloud.session.auth.get_endpoint_data(cloud.session, service).url)
 7 ...
 8 https://api.it-mil1.entercloudsuite.com/v2.0/
 9 https://nova.it-mil1.entercloudsuite.com/v2/cb7088c9306742858e8a38b97744fba8
10 https://glance.it-mil1.entercloudsuite.com/v2/
11 https://neutron.it-mil1.entercloudsuite.com/v2.0
12 https://cinder.it-mil1.entercloudsuite.com/v2/cb7088c9306742858e8a38b97744fba8
13 https://swift.it-mil1.entercloudsuite.com/v1/KEY_cb7088c9306742858e8a38b97744fba8
```

Source: Created by the author of this document.

Therefore, the validation process will use those URLs, except for the Image service that had its versioned URL explicitly set in the configuration file, resulting in the final list of service versions for Enter Cloud Suite presented in Table 28. The script has discovered the URLs for the cloud region in Milan, Italy, while the one in Amsterdan, Netherlands is used in this validation. The versions will be the same, but the URLs will change to include *nl-ams1* instead of *it-mil1*.

Table 28 - Service versions discovered for Enter Cloud Suite

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	2	1	2	2	2	1

Source: Created by the author of this document.

I.2 Test Results

This section reports the output of running the testing scripts from E.2 Testing Code for Compute, Image, Networking, Block Storage and Object Storage resources against the Enter

Cloud Suite cloud environment as specified in the Section E.4 Running Tests. All tests, except for the public server test cases, ran successfully in this environment.

I.2.1 Image

The Section 3.3.9 detailed that there are two workflows to create an image: directly via the PUT API of the image service or by uploading the image to the Object Storage service and then importing it to the image service via the Tasks API. The Enter Cloud Suite cloud runs the former, as shown in line 9 of Figure 157, which outputs the custom logger print along with the successful execution of the image tests.

Figure 157 - Image tests output for Enter Cloud Suite

```
 1 $ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test_sdk.ImageTestCase --
debug=test_sdk
 2 test_create (test_sdk.ImageTestCase) ... ok
 3 test_get (test_sdk.ImageTestCase) ... ok
 4 test_list (test_sdk.ImageTestCase) ... ok
 5 test_search (test_sdk.ImageTestCase) ... ok
 6 test_update (test_sdk.ImageTestCase) ... ok
 7 test_delete (test_sdk.ImageTestCase) ... ok
 8
 9 test_sdk: DEBUG: Image via PUT
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 40.152s
13
14 OK
```

Source: Created by the author of this document.

I.2.2 Networking

As shown in Figure 158, Figure 159 and Figure 160, all the tests for network, subnetwork and security group, respectively, have run successfully.

Network:

Figure 158 - Network tests output for Enter Cloud Suite

```
 1 $ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test sdk.NetworkTestCase
 2 test_create (test_sdk.NetworkTestCase) ... ok
 3 test_get (test_sdk.NetworkTestCase) ... ok
 4 test_list (test_sdk.NetworkTestCase) ... ok
 5 test_search (test_sdk.NetworkTestCase) ... ok
 6 test update (test sdk.NetworkTestCase) ... ok
 7 test_delete (test_sdk.NetworkTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 12.221s
11
12 OK
```

Source: Created by the author of this document.

Subnetwork:

Figure 159 - Subnetwork tests output for Enter Cloud Suite

```
 1 $ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test_sdk.SubnetTestCase
 2 test create (test sdk.SubnetTestCase) ... ok
 3 test_get (test_sdk.SubnetTestCase) ... ok
 4 test_list (test_sdk.SubnetTestCase) ... ok
 5 test_search (test_sdk.SubnetTestCase) ... ok
 6 test_update (test_sdk.SubnetTestCase) ... ok
 7 test_delete (test_sdk.SubnetTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 27.606s
11
12 OK
```

Source: Created by the author of this document.

Security Group:

Figure 160 - Security group tests output for Enter Cloud Suite

```
 1 $ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test_sdk.SecurityGroupTestCase
 2 test_create (test_sdk.SecurityGroupTestCase) ... ok
 3 test_get (test_sdk.SecurityGroupTestCase) ... ok
 4 test_list (test_sdk.SecurityGroupTestCase) ... ok
 5 test_search (test_sdk.SecurityGroupTestCase) ... ok
 6 test_update (test_sdk.SecurityGroupTestCase) ... ok
 7 test_delete (test_sdk.SecurityGroupTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 10.596s
11
12 OK
```

Source: Created by the author of this document.

I.2.3 Block Storage

As shown in Figure 161, all tests for volume have run successfully.

Volume:

Figure 161 - Volume tests output for Enter Cloud Suite

```
$ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test_sdk.VolumeTestCase
test_create (test_sdk.VolumeTestCase) ... ok
test_get (test_sdk.VolumeTestCase) ... ok
test_list (test_sdk.VolumeTestCase) ... ok
test_search (test_sdk.VolumeTestCase) ... ok
test_update (test_sdk.VolumeTestCase) ... ok
test_delete (test_sdk.VolumeTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 41.295s

OK
```

Source: Created by the author of this document.

I.2.4 Compute

As shown in Figure 162, Figure 163, Figure 164 and Figure 166, all the tests for flavor, keypair, private server and snapshot, respectively, have run successfully. As per Figure 165, the tests for public server have failed with an unexpected state reached by the server.

Flavor:

Figure 162 - Flavor tests output for Enter Cloud Suite

```
$ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test_sdk.FlavorTestCase
test_create (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be created
test_delete (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be deleted
test_update (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be updated
test_get (test_sdk.FlavorTestCase) ... ok
test_list (test_sdk.FlavorTestCase) ... ok
test_search (test_sdk.FlavorTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 33.896s

OK (SKIP=3)
```

Source: Created by the author of this document.

Keypair:

Figure 163 - Keypair tests output for Enter Cloud Suite

```
$ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test sdk.KeypairTestCase
test_update (test_sdk.KeypairTestCase) ... SKIP: Keypairs cannot be updated
test_create (test_sdk.KeypairTestCase) ... ok
test_get (test_sdk.KeypairTestCase) ... ok
test_list (test_sdk.KeypairTestCase) ... ok
test search (test sdk.KeypairTestCase) ... ok
test_delete (test_sdk.KeypairTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 8.464s

OK (SKIP=1)
```

Source: Created by the author of this document.

Private Server:

Figure 164 - Private server tests output for Enter Cloud Suite

```
$ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test_sdk.PrivateServerTestCase
test create (test sdk.PrivateServerTestCase) ... ok
test_get (test_sdk.PrivateServerTestCase) ... ok
test_list (test_sdk.PrivateServerTestCase) ... ok
test_search (test_sdk.PrivateServerTestCase) ... ok
test_update (test_sdk.PrivateServerTestCase) ... ok
test_delete (test_sdk.PrivateServerTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 223.615s

OK
```

Source: Created by the author of this document.

Public Server:

Figure 165 shows the output for the public server tests, where all test cases failed. The reason of failure is that the server created reached an active state, which means it is ready for use, without having any IP address assigned to it. A server without an IP address is useless in the context of cloud computing, where end users are meant to have remote access to servers.

Figure 165 - Public server tests output for Enter Cloud Suite

```
 1 $ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test_sdk.PublicServerTestCase --
debug=test_sdk
 2 test_create (test_sdk.PublicServerTestCase) ... ERROR
 3 test_get (test_sdk.PublicServerTestCase) ... ERROR
 4 test_list (test_sdk.PublicServerTestCase) ... ERROR
 5 test_search (test_sdk.PublicServerTestCase) ... ERROR
 6 test_update (test_sdk.PublicServerTestCase) ... ERROR
 7 test_delete (test_sdk.PublicServerTestCase) ... ERROR
 8
 9 openstack.exceptions.SDKException: Server reached ACTIVE state without being allocated an
IP address.
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 147.918s
13
14 FAILED (errors=6)
```

Source: Created by the author of this document.

Snapshot:

Figure 166 - Snapshot tests output for Enter Cloud Suite

```
 1 $ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test_sdk.SnapshotTestCase
 2 test_create (test_sdk.SnapshotTestCase) ... ok
 3 test_get (test_sdk.SnapshotTestCase) ... ok
 4 test_list (test_sdk.SnapshotTestCase) ... ok
 5 test_search (test_sdk.SnapshotTestCase) ... ok
 6 test_update (test_sdk.SnapshotTestCase) ... ok
 7 test_delete (test_sdk.SnapshotTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 286.732s
11
12 OK
```

Source: Created by the author of this document.

I.2.5 Object Storage

As shown in Figure 167 and Figure 168, all the tests for container and object, respectively, have run successfully.

Container:

Figure 167 - Container tests output for Enter Cloud Suite

```
 1 $ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test sdk.ContainerTestCase
 2 test_update (test_sdk.ContainerTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ContainerTestCase) ... ok
 4 test_get (test_sdk.ContainerTestCase) ... ok
 5 test_list (test_sdk.ContainerTestCase) ... ok
 6 test search (test sdk.ContainerTestCase) ... ok
 7 test_delete (test_sdk.ContainerTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 17.872s
11
12 OK (SKIP=1)
```

Source: Created by the author of this document.

Object:

Figure 168 - Object tests output for Enter Cloud Suite

```
 1 $ CLOUD=entercloudsuite nosetests -v -e TestCase --tests=test_sdk.ObjectTestCase
 2 test update (test sdk.ObjectTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ObjectTestCase) ... ok
 4 test_get (test_sdk.ObjectTestCase) ... ok
 5 test_list (test_sdk.ObjectTestCase) ... ok
 6 test_search (test_sdk.ObjectTestCase) ... ok
 7 test_delete (test_sdk.ObjectTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 39.555s
11
12 OK (SKIP=1)
```

Source: Created by the author of this document.

APPENDIX J – VALIDATION ON SWITCHENGINES

This appendix describes the validation of OpenStack SDK against the SWITCHengines cloud by running the tests from APPENDIX E – TESTING CODE AND ENVIRONMENT. The Section J.1 Environment Setup details how the testing environment was configured; whereas J.2 Test Results reports the tests execution and output.

J.1 Environment Setup

The local environment was configured and the SWITCHengines cloud services were discovered by following the guidelines defined in the Section E.3 Testing Environment. The local tests will use OpenStack SDK to communicate and test the SWITCHengines cloud remotely. The next subsections detail the user authentication information and cloud services versions.

J.1.1 Authentication

The configuration details allowing OpenStack SDK to connect to the remote cloud are specified in its configuration file, located at */etc/openstack/clouds.yaml*, as in Figure 169. Lines 3 to 8 define how to authenticate to the cloud; line 9 specifies that the region to use is the one in Zurich, Switzerland.

Figure 169 - OpenStack SDK configuration for SWITCHengines

```
1 clouds:
2   switchengines:
3     auth:
4       auth_url: https://keystone.cloud.switch.ch:5000
5       project_id: 7d04eab08a194304a8942217a8292be3
6       username: samueldmq@gmail.com
7       user_domain_name: Default
8       password: ******
9     region_name: ZH
```

Source: Created by the author of this document.

J.1.2 Services

The cloud services are exposed in the service catalog, where each service may support multiple API versions. When using OpenStack SDK, the versions are automatically selected by

the service discovery process, which outputs the versioned URLs. Both service catalog and discovery are described in APPENDIX B – SERVICE CATALOG AND DISCOVERY. For SWITCHengines, the service discovery selects the versioned URLs as in Figure 170.

Figure 170 - Services versioned URLs discovery for SWITCHengines

```
 1 $ python
 2 >>> import openstack
 3 >>> cloud = openstack.connect('switchengines')
 4 >>> for service in ('identity', 'compute', 'image', 'network',
 5                    'block-storage', 'object-store'):
 6 ...   print(cloud.session.auth.get_endpoint_data(cloud.session, service).url)
 7 ...
 8 https://keystone.cloud.switch.ch:5000/v3/
 9 https://api.zhdk.cloud.switch.ch:8774/v2/7d04eab08a194304a8942217a8292be3
10 https://api.unil.cloud.switch.ch:9292/v2/
11 https://api.zhdk.cloud.switch.ch:9696/v2.0
12 https://api.zhdk.cloud.switch.ch:8776/v3/7d04eab08a194304a8942217a8292be3
13 https://os.zhdk.cloud.switch.ch/swift/v1
```

Source: Created by the author of this document.

Therefore, the list of service versions for SWITCHengines summarized in Table 29.

Table 29 - Service versions discovered for SWITCHengines

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	3	2	2	3	2	1

Source: Created by the author of this document.

J.2 Test Results

This section reports the output of running the testing scripts from E.2 Testing Code for Compute, Image, Networking, Block Storage and Object Storage resources against the SWITCHengines cloud environment as specified in the Section E.4 Running Tests. All tests ran successfully in this environment.

J.2.1 Image

The Section 3.3.9 detailed that there are two workflows to create an image: directly via the PUT API of the image service or by uploading the image to the Object Storage service and then importing it to the image service via the Tasks API. The SWITCHengines cloud runs the former, as shown in line 9 of Figure 171, which outputs the custom logger print along with the successful execution of the image tests.

Figure 171 - Image tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test_sdk.ImageTestCase --
debug=test_sdk
 2 test_create (test_sdk.ImageTestCase) ... ok
 3 test_get (test_sdk.ImageTestCase) ... ok
 4 test_list (test_sdk.ImageTestCase) ... ok
 5 test_search (test_sdk.ImageTestCase) ... ok
 6 test_update (test_sdk.ImageTestCase) ... ok
 7 test_delete (test_sdk.ImageTestCase) ... Ok
 8
 9 test_sdk: DEBUG: Image via PUT
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 136.571s
13
14 OK
```

Source: Created by the author of this document.

J.2.2 Networking

As shown in Figure 172, Figure 173 and Figure 174, all the tests for network, subnetwork and security group, respectively, have run successfully.

Network:

Figure 172 - Network tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test_sdk.NetworkTestCase
 2 test_create (test_sdk.NetworkTestCase) ... ok
 3 test_get (test_sdk.NetworkTestCase) ... ok
 4 test_list (test_sdk.NetworkTestCase) ... ok
 5 test_search (test_sdk.NetworkTestCase) ... ok
 6 test_update (test_sdk.NetworkTestCase) ... ok
 7 test_delete (test_sdk.NetworkTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 24.188s
11
12 OK
```

Source: Created by the author of this document.

Subnetwork:

Figure 173 - Subnetwork tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test sdk.SubnetTestCase
 2 test_create (test_sdk.SubnetTestCase) ... ok
 3 test_get (test_sdk.SubnetTestCase) ... ok
 4 test_list (test_sdk.SubnetTestCase) ... ok
 5 test_search (test_sdk.SubnetTestCase) ... ok
 6 test update (test sdk.SubnetTestCase) ... ok
 7 test_delete (test_sdk.SubnetTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 71.245s
11
12 OK
```

Source: Created by the author of this document.

Security Group:

Figure 174 - Security group tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test_sdk.SecurityGroupTestCase
 2 test create (test sdk.SecurityGroupTestCase) ... ok
 3 test_get (test_sdk.SecurityGroupTestCase) ... ok
 4 test_list (test_sdk.SecurityGroupTestCase) ... ok
 5 test_search (test_sdk.SecurityGroupTestCase) ... ok
 6 test_update (test_sdk.SecurityGroupTestCase) ... ok
 7 test_delete (test_sdk.SecurityGroupTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 15.930s
11
12 OK
```

Source: Created by the author of this document.

J.2.3 Block Storage

Volume:

Figure 175 - Volume tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test_sdk.VolumeTestCase
 2 test_create (test_sdk.VolumeTestCase) ... ok
 3 test_get (test_sdk.VolumeTestCase) ... ok
 4 test_list (test_sdk.VolumeTestCase) ... ok
 5 test_search (test_sdk.VolumeTestCase) ... ok
 6 test_update (test_sdk.VolumeTestCase) ... ok
 7 test_delete (test_sdk.VolumeTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 56.994s
11
12 OK
```

Source: Created by the author of this document.

J.2.4 Compute

As shown in Figure 176, Figure 177, Figure 178, Figure 179 and Figure 180, all the tests for flavor, keypair, private server, public server and snapshot, respectively, have run successfully.

Flavor:

Figure 176 - Flavor tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test_sdk.FlavorTestCase
 2 test_create (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be created
 3 test_delete (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be deleted
 4 test_update (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be updated
 5 test_get (test_sdk.FlavorTestCase) ... ok
 6 test_list (test_sdk.FlavorTestCase) ... ok
 7 test_search (test_sdk.FlavorTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 37.948s
11
12 OK (SKIP=3)
```

Source: Created by the author of this document.

Keypair:

Figure 177 - Keypair tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test_sdk.KeypairTestCase
 2 test_update (test_sdk.KeypairTestCase) ... SKIP: Keypairs cannot be updated
 3 test_create (test_sdk.KeypairTestCase) ... ok
 4 test_get (test_sdk.KeypairTestCase) ... ok
 5 test_list (test_sdk.KeypairTestCase) ... ok
 6 test_search (test_sdk.KeypairTestCase) ... ok
 7 test_delete (test_sdk.KeypairTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 12.181s
11
12 OK (SKIP=1)
```

Source: Created by the author of this document.

Private Server:

Figure 178 - Private server tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test sdk.PrivateServerTestCase
 2 test_create (test_sdk.PrivateServerTestCase) ... ok
 3 test_get (test_sdk.PrivateServerTestCase) ... ok
 4 test_list (test_sdk.PrivateServerTestCase) ... ok
 5 test_search (test_sdk.PrivateServerTestCase) ... ok
 6 test update (test sdk.PrivateServerTestCase) ... ok
 7 test_delete (test_sdk.PrivateServerTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 407.694s
11
12 OK
```

Source: Created by the author of this document.

Public Server:

The Section 3.3.12 detailed that there are two workflows to create a server and get a public IP address assigned to it, the first is when the IP is directly routable via DHCP and the second is via NAT mechanism. The SWITCHengines cloud runs the latter, as shown in line 9, which outputs the custom logger print.

Figure 179 - Public server tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test sdk.PublicServerTestCase --
debug=test_sdk
 2 test_create (test_sdk.PublicServerTestCase) ... ok
 3 test_get (test_sdk.PublicServerTestCase) ... ok
 4 test_list (test_sdk.PublicServerTestCase) ... ok
 5 test_search (test_sdk.PublicServerTestCase) ... ok
 6 test update (test sdk.PublicServerTestCase) ... ok
 7 test_delete (test_sdk.PublicServerTestCase) ... ok
 8
 9 test_sdk: DEBUG: IP via NAT
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 340.585s
13
14 OK
```

Source: Created by the author of this document.

Snapshot:

Figure 180 - Snapshot tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test sdk.SnapshotTestCase
 2 test_create (test_sdk.SnapshotTestCase) ... ok
 3 test_get (test_sdk.SnapshotTestCase) ... ok
 4 test_list (test_sdk.SnapshotTestCase) ... ok
 5 test_search (test_sdk.SnapshotTestCase) ... ok
 6 test update (test sdk.SnapshotTestCase) ... ok
 7 test_delete (test_sdk.SnapshotTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 471.279s
11
12 OK
```

Source: Created by the author of this document.

J.2.5 Object Storage

As shown in Figure 181 and Figure 182, all the tests for container and object, respectively, have run successfully.

Container:

Figure 181 - Container tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test_sdk.ContainerTestCase
 2 test_update (test_sdk.ContainerTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ContainerTestCase) ... ok
 4 test_get (test_sdk.ContainerTestCase) ... ok
 5 test_list (test_sdk.ContainerTestCase) ... ok
 6 test_search (test_sdk.ContainerTestCase) ... ok
 7 test_delete (test_sdk.ContainerTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 13.993s
11
12 OK (SKIP=1)
```

Source: Created by the author of this document.

Object:

Figure 182 - Object tests output for SWITCHengines

```
 1 $ CLOUD=switchengines nosetests -v -e TestCase --tests=test sdk.ObjectTestCase
 2 test_update (test_sdk.ObjectTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ObjectTestCase) ... ok
 4 test_get (test_sdk.ObjectTestCase) ... ok
 5 test_list (test_sdk.ObjectTestCase) ... ok
 6 test_search (test_sdk.ObjectTestCase) ... ok
 7 test_delete (test_sdk.ObjectTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 26.614s
11
12 OK (SKIP=1)
```

Source: Created by the author of this document.

APPENDIX K – VALIDATION ON ORMUCO

This appendix describes the validation of OpenStack SDK against the Ormuco cloud by running the tests from APPENDIX E – TESTING CODE AND ENVIRONMENT. The Section K.1 Environment Setup details how the testing environment was configured; whereas K.2 Test Results reports the tests execution and output.

K.1 Environment Setup

The local environment was configured and the Ormuco cloud services were discovered by following the guidelines defined in the Section E.3 Testing Environment. The local tests will use OpenStack SDK to communicate and test the Ormuco cloud remotely. The next subsections detail the user authentication information and cloud services versions.

K.1.1 Authentication

The configuration details allowing OpenStack SDK to connect to the remote cloud are specified in its configuration file, located at */etc/openstack/clouds.yaml*, as in Figure 183. Lines 3 to 8 define how to authenticate to the cloud; line 9 specifies that the region to use is the one in Montreal, Canada.

Figure 183 - OpenStack SDK configuration for Ormuco

```
1 clouds:
2   ormuco:
3     auth:
4       auth_url: https://api.ormuco.com:5000
5       project_id: a840642f24a14bf18be54c1b68227ebb
6       username: samueldmq@gmail.com
7       user_domain_name: Default
8       password: ******
9     region_name: prd-ca-yul-2
```

Source: Created by the author of this document.

K.1.2 Services

The cloud services are exposed in the service catalog, where each service may support multiple API versions. When using OpenStack SDK, the versions are automatically selected by the service discovery process, which outputs the versioned URLs. Both service catalog and

discovery are described in APPENDIX B – SERVICE CATALOG AND DISCOVERY. For Ormuco, the service discovery selects the versioned URLs as in Figure 184.

Figure 184 - Services versioned URLs discovery for Ormuco

```
 1 $ python
 2 >>> import openstack
 3 >>> cloud = openstack.connect('ormuco')
 4 >>> for service in ('identity', 'compute', 'image', 'network',
 5 ...                 'block-storage', 'object-store'):
 6 ...   print(cloud.session.auth.get_endpoint_data(cloud.session, service).url)
 7 ...
 8 https://api.ormuco.com:5000/v3/
 9 https://api.ormuco.com:8774/v2.1/a840642f24a14bf18be54c1b68227ebb
10 https://api.ormuco.com:9292/v2/
11 https://api.ormuco.com:9696/v2.0/
12 https://api.ormuco.com:8776/v3/a840642f24a14bf18be54c1b68227ebb
13 https://api.ormuco.com:6780/swift/v1/a840642f24a14bf18be54c1b68227ebb
```

Source: Created by the author of this document.

Therefore, the validation process will use those URLs, resulting in the final list of service versions for Ormuco shown in Table 30.

Table 30 - Service versions discovered for Ormuco

Service	Authentication	Image	Networking	Block Storage	Compute	Object Storage
Version	3	2	2	3	2	1

Source: Created by the author of this document.

K.2 Test Results

This section reports the output of running the testing scripts from E.2 Testing Code for Compute, Image, Networking, Block Storage and Object Storage resources against the Ormuco cloud environment as specified in the Section E.4 Running Tests. All tests, except for the delete test case of object, ran successfully in this environment.

K.2.1 Image

The Section 3.3.9 detailed that there are two workflows to create an image: directly via the PUT API of the image service or by uploading the image to the Object Storage service and then importing it to the image service via the Tasks API. The Ormuco cloud runs the former, as shown in line 9 of Figure 185, which outputs the custom logger print along with the successful execution of the image tests.

Figure 185 - Image tests output for Ormuco

```
 1 $ CLOUD=ormuco nosetests -v -e TestCase --tests=test_sdk.ImageTestCase --debug=test_sdk
 2 test_create (test_sdk.ImageTestCase) ... ok
 3 test_get (test_sdk.ImageTestCase) ... ok
 4 test_list (test_sdk.ImageTestCase) ... ok
 5 test_search (test_sdk.ImageTestCase) ... ok
 6 test_update (test_sdk.ImageTestCase) ... ok
 7 test_delete (test_sdk.ImageTestCase) ... ok
 8
 9 test_sdk: DEBUG: Image via PUT
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 83.132s
13
14 OK
```

Source: Created by the author of this document.

K.2.2 Networking

As shown in Figure 186, Figure 187 and Figure 188, all the tests for network, subnetwork and security group, respectively, have run successfully.

Network:

Figure 186 - Network tests output for Ormuco

```
 1 $ CLOUD=ormuco nosetests -v -e TestCase --tests=test_sdk.NetworkTestCase
 2 test_create (test_sdk.NetworkTestCase) ... ok
 3 test_get (test_sdk.NetworkTestCase) ... ok
 4 test_list (test_sdk.NetworkTestCase) ... ok
 5 test_search (test_sdk.NetworkTestCase) ... ok
 6 test_update (test_sdk.NetworkTestCase) ... ok
 7 test_delete (test_sdk.NetworkTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 34.874s
11
12 OK
```

Source: Created by the author of this document.

Subnetwork:

Figure 187 - Subnetwork tests output for Ormuco

```
 1 $ CLOUD=ormuco nosetests -v -e TestCase --tests=test sdk.SubnetTestCase
 2 test_create (test_sdk.SubnetTestCase) ... ok
 3 test_get (test_sdk.SubnetTestCase) ... ok
 4 test_list (test_sdk.SubnetTestCase) ... ok
 5 test_search (test_sdk.SubnetTestCase) ... ok
 6 test update (test sdk.SubnetTestCase) ... ok
 7 test_delete (test_sdk.SubnetTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 52.570s
11
12 OK
```

Source: Created by the author of this document.

Security Group:

Figure 188 - Security group tests output for Ormuco

```
 1 $ CLOUD=ormuco nosetests -v -e TestCase --tests=test_sdk.SecurityGroupTestCase
 2 test create (test sdk.SecurityGroupTestCase) ... ok
 3 test_get (test_sdk.SecurityGroupTestCase) ... ok
 4 test_list (test_sdk.SecurityGroupTestCase) ... ok
 5 test_search (test_sdk.SecurityGroupTestCase) ... ok
 6 test_update (test_sdk.SecurityGroupTestCase) ... ok
 7 test_delete (test_sdk.SecurityGroupTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 42.034s
11
12 OK
```

Source: Created by the author of this document.

K.2.3 Block Storage

Volume:

Figure 189 - Volume tests output for Ormuco

```
 1 $ CLOUD=ormuco nosetests -v -e TestCase --tests=test_sdk.VolumeTestCase
 2 test_create (test_sdk.VolumeTestCase) ... ok
 3 test_get (test_sdk.VolumeTestCase) ... ok
 4 test_list (test_sdk.VolumeTestCase) ... ok
 5 test_search (test_sdk.VolumeTestCase) ... ok
 6 test_update (test_sdk.VolumeTestCase) ... ok
 7 test_delete (test_sdk.VolumeTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 49.866s
11
12 OK
```

Source: Created by the author of this document.

K.2.4 Compute

As shown in Figure 190, Figure 191, Figure 192, Figure 193 and Figure 194, all the tests for flavor, keypair, private server, public server and snapshot, respectively, have run successfully.

Flavor:

Figure 190 - Flavor tests output for Ormuco

```
$ CLOUD=ormuco nosetests -v -e TestCase --tests=test_sdk.FlavorTestCase
test_create (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be created
test_delete (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be deleted
test_update (test_sdk.FlavorTestCase) ... SKIP: Flavors cannot be updated
test_get (test_sdk.FlavorTestCase) ... ok
test_list (test_sdk.FlavorTestCase) ... ok
test_search (test_sdk.FlavorTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 21.810s

OK (SKIP=3)
```

Source: Created by the author of this document.

Keypair:

Figure 191 - Keypair tests output for Ormuco

```
$ CLOUD=ormuco nosetests -v -e TestCase --tests=test_sdk.KeypairTestCase
test_update (test_sdk.KeypairTestCase) ... SKIP: Keypairs cannot be updated
test_create (test_sdk.KeypairTestCase) ... ok
test_get (test_sdk.KeypairTestCase) ... ok
test_list (test_sdk.KeypairTestCase) ... ok
test_search (test_sdk.KeypairTestCase) ... ok
test_delete (test_sdk.KeypairTestCase) ... ok

----------------------------------------------------------------------
Ran 6 tests in 7.763s

OK (SKIP=1)
```

Source: Created by the author of this document.

Private Server:

Figure 192 - Private server tests output for Ormuco

```
 1 $ CLOUD=ormuco nosetests -v -e TestCase --tests=test sdk.PrivateServerTestCase
 2 test_create (test_sdk.PrivateServerTestCase) ... ok
 3 test_get (test_sdk.PrivateServerTestCase) ... ok
 4 test_list (test_sdk.PrivateServerTestCase) ... ok
 5 test_search (test_sdk.PrivateServerTestCase) ... ok
 6 test update (test sdk.PrivateServerTestCase) ... ok
 7 test_delete (test_sdk.PrivateServerTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 945.712s
11
12 OK
```

Source: Created by the author of this document.

Public Server:

The Section 3.3.12 detailed that there are two workflows to create a server and get a public IP address assigned to it, the first is when the IP is directly routable via DHCP and the second is via NAT mechanism. The Ormuco cloud runs the latter, as shown in line 9, which outputs the custom logger print.

Figure 193 - Public server tests output for Ormuco

```
 1 $ CLOUD=ormuco nosetests -v -e TestCase --tests=test sdk.PublicServerTestCase --
debug=test_sdk
 2 test_create (test_sdk.PublicServerTestCase) ... ok
 3 test_get (test_sdk.PublicServerTestCase) ... ok
 4 test_list (test_sdk.PublicServerTestCase) ... ok
 5 test_search (test_sdk.PublicServerTestCase) ... ok
 6 test update (test sdk.PublicServerTestCase) ... ok
 7 test_delete (test_sdk.PublicServerTestCase) ... ok
 8
 9 test_sdk: DEBUG: IP via NAT
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 714.946s
13
14 OK
```

Source: Created by the author of this document.

Snapshot:

Figure 194 - Snapshot tests output for Ormuco

```
 1 $ CLOUD=ormuco nosetests -v -e TestCase --tests=test sdk.SnapshotTestCase
 2 test_create (test_sdk.SnapshotTestCase) ... ok
 3 test_get (test_sdk.SnapshotTestCase) ... ok
 4 test_list (test_sdk.SnapshotTestCase) ... ok
 5 test_search (test_sdk.SnapshotTestCase) ... ok
 6 test update (test sdk.SnapshotTestCase) ... ok
 7 test_delete (test_sdk.SnapshotTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 618.291s
11
12 OK
```

Source: Created by the author of this document.

K.2.5 Object Storage

As shown in Figure 195, all the tests for container have run successfully. However, as per Figure 196, the tests for object failed in the delete test case.

Container:

Figure 195 - Container tests output for Ormuco

```
 1 $ CLOUD=ormuco nosetests -v -e TestCase --tests=test_sdk.ContainerTestCase
 2 test_update (test_sdk.ContainerTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ContainerTestCase) ... ok
 4 test_get (test_sdk.ContainerTestCase) ... ok
 5 test_list (test_sdk.ContainerTestCase) ... ok
 6 test_search (test_sdk.ContainerTestCase) ... ok
 7 test_delete (test_sdk.ContainerTestCase) ... ok
 8
 9 ----------------------------------------------------------------------
10 Ran 6 tests in 6.552s
11
12 OK (SKIP=1)
```

Source: Created by the author of this document.

Object:

Figure 196 shows that the delete test case failed with a list index out of range, which means the code tried to access a location that does not exist in a list.

Figure 196 - Object tests output for Ormuco

```
 1 $ CLOUD=ormuco nosetests -v -e TestCase --tests=test_sdk.ObjectTestCase
 2 test_update (test_sdk.ObjectTestCase) ... SKIP: Object store cannot be updated
 3 test_create (test_sdk.ObjectTestCase) ... ok
 4 test_get (test_sdk.ObjectTestCase) ... ok
 5 test_list (test_sdk.ObjectTestCase) ... ok
 6 test_search (test_sdk.ObjectTestCase) ... ok
 7 test_delete (test_sdk.ObjectTestCase) ... ERROR
 8
 9 IndexError: list index out of range
10
11 ----------------------------------------------------------------------
12 Ran 6 tests in 12.857s
13
14 FAILED (SKIP=1, errors=1)
```

Source: Created by the author of this document.

Further debugging was done to discover the exact reason that test failed, finding that what was running behind the Object Storage API in the OpenStack deploy was not an OpenStack service, but the Ceph Object Gateway (RADOS REST Gateway, 2018), which declares it supports the OpenStack Object Storage API (Ceph Object Gateway Swift API, 2018). The failure was a consequence of that service returning a HTTP 200 OK instead of a HTTP 404 NOT FOUND when asserting that the object that had just been del eted did not exist anymore in the cloud.

APPENDIX L – SYSTEMATIC LITERATURE REVIEW PROTOCOL

This appendix provides details about the systematic literature review protocol and subsequent selected papers on IaaS interoperability. The Section L.1 Protocol defines the formal protocol that led to the retrieval of the papers in L.2 Papers.

L.1 Protocol

The systematic review protocol definition, population, search and filtering methods are described in this section.

Definition:

The general goal of this literature review is to supply a comprehension from the existing literature of what are the interoperability solutions, causes and consequences of the lack of it in the scope of IaaS. There are three specific goals, enumerated in Table 31.

Table 31 - Systematic literature review goals

Goal #	Description
1	Identify general solutions for the lack of interoperability in IaaS
2	Identify the causes for IaaS systems not to interoperate
3	Identify the consequences of non-interoperable IaaS systems

Source: Created by the author of this document.

To meet these goals, the research questions in Table 32 need to be answered by the findings of the selected papers.

Table 32 - Systematic literature review research questions

Research Question #	Description
1	What are the general solutions for the lack of interoperability in IaaS?
2	What are the causes for IaaS systems not to interoperate?
3	What are the consequences of non-interoperable IaaS systems?

Source: Created by the author of this document.

Population:

The interest in this review is on studies discussing IaaS interoperability problems, causes and consequences, along with approaches for achieving it and use cases. Only studies that are published in full text in English are considered for further filtering.

Search:

Papers will be sought via automated search in the electronic databases enumerated in Table 33, whose represent relevant paper databases in cloud computing.

Table 33 - Automated search sources

Search Source	URL
IEEEXplore	http://ieeexplore.ieee.org
ACM Digital Library	http://dl.acm.org
ScienceDirect.com	http://www.sciencedirect.com
Scopus	http://www.scopus.com
Web of Science	http://www.webofknowledge.com

Source: Created by the author of this document.

Search on papers focus on the presence or absence of certain keywords defining the context in which the information is sought. The keywords for this review are enumerated and described in Table 34.

Table 34 - Systematic literature review keywords

Keyword	Description
Interoperability	Determines the subject to this investigation.
Vendor lock-in	The opposite of interoperability, i.e. the absence of interoperability in a multi-vendor context.
Infrastructure as a Service	Represents the scope in which interoperability is analyzed.
IaaS	Acronym for Infrastructure as a Service.

Source: Created by the author of this document.

These keywords generate the base search string in Figure 197, necessary as input by the automated search electronic databases.

Figure 197 - Base search string

(("Interoperability" **OR** "Vendor lock-in") **AND** ("Infrastructure as a Service" **OR** "IaaS"))

Source: Created by the author of this document.

Each paper database defines a different language syntax to querying papers, requiring the base search string to be adapted, as enumerated in Table 35. Note that, for this study, the search strings will search for specific words in the title, abstract and keywords attributes of papers.

Table 35 - Adapted search strings

Search Source	Adapted Search String
ACM Digital Library	acmdlTitle:(("interoperability" OR "vendor lock-in") AND ("infrastructure as a service" OR "iaas")) OR recordAbstract:(("interoperability" OR "vendor lock-in") AND ("infrastructure as a service" OR "iaas")) OR keywords.author.keyword:(("interoperability" OR "vendor lock-in") AND ("infrastructure as a service" OR "iaas"))
IEEEXplore	(("Document Title":"interoperability" OR "Document Title":"vendor lock-in") AND ("Document Title":"infrastructure as a service" OR "Document Title":"iaas")) OR (("Abstract":"interoperability" OR "Abstract":"vendor lock-in") AND ("Abstract":"infrastructure as a service" OR "Abstract":"iaas")) OR (("Author Keywords":"interoperability" OR "Author Keywords":"vendor lock-in") AND ("Author Keywords":"infrastructure as a service" OR "Author Keywords":"iaas"))
ScienceDirect.com	Title-Abstr-Key(("interoperability" OR "vendor lock-in") AND ("infrastructure as a service" OR "iaas"))
Scopus	TITLE-ABS-KEY(("interoperability" OR "vendor lock-in") AND ("infrastructure as a service" OR "iaas"))
Web of Science	TS=(("interoperability" OR "vendor lock-in") AND ("infrastructure as a service" OR "iaas"))

Source: Created by the author of this document.

One particularity of the search on the ACM Digital Library database was that instead of performing the query on the default database *Full-Text Collection* of the search engine, *The ACM Guide to Computing Literature* was explicitly set. The former contains all ACM publications, including journals, conference proceedings, technical magazines, newsletters and books; while the latter is a comprehensive bibliographic database focused exclusively on the field of computing.

Filtering:

Once the studies' references were retrieved from the electronic databases, an analysis is necessary. Each study has its relevancy individually analyzed, where a study is considered relevant if it does not meet any exclusion criterion and meet at least one inclusion criterion, enumerated in Table 36 and Table 37, respectively.

Table 36 - Inclusion criteria

Criterion #	Description
1	It discusses issues and challenges related to interoperability.
2	It reviews existing approaches for addressing interoperability.
3	It proposes a novel approach for addressing interoperability.
4	It presents a case study analyzing interoperability.

Source: created by the author of this document.

Note that the inclusion criteria do not clearly specify the scope of interoperability, which is IaaS. Such restriction is introduced as an exclusion criterion, causing a study that is not in this scope to be removed from the analysis.

Table 37 - Exclusion criteria

Criterion #	Description
1	It is not available in its complete form (full-access).
2	It is not directly related to interoperability.
3	It is not directly related to IaaS platforms.
4	It is a previous version of a more complete paper about the same research.
5	It is not written in English, the most common language in scientific papers.

Source: created by the author of this document.

Checking if papers are relevant is a two-phased filtering process:

1. Preliminary selection with reading of title, abstract and keywords;
2. Further selection with the complete reading of the study.

Besides selecting the papers retrieved in the search, it is possible to select relevant papers that appear in the references of those, in a process named snowballing. Papers suggested by specialists, such as the author or his supervisor, might be individually added to the list as well.

L.2 Papers

After performing the search on the electronic databases with the appropriate adapted search strings, the number of gross entries for each base was as in Table 38, with a total of 274. The search was last performed on October 30th, 2018.

Table 38 - Retrieved entries

Electronic Database	# Entries
ACM Digital Library	49
IEEEXplore	35
ScienceDirect.com	10
Scopus	117
Web of Science	63
Total	274

Source: Created by the author of this document.

Firstly, all the irrelevant entries were removed, which include overview of conferences' proceedings and chapters of books, resulting in a total of 253 entries. In second place, all the duplicates were removed, lasting 121 papers to be analyzed. During the removal of duplicates by the paper name and authors, newer versions in latest years were prioritized. The results of this process are documented in Table 39.

Table 39 - Analyzed papers

Electronic Database	# Papers
ACM Digital Library	43
IEEEXplore	34
ScienceDirect.com	9
Scopus	105
Web of Science	62
Total (Gross)	253
Total (Unique)	121

Source: Created by the author of this document.

The 121 papers were analyzed, and the selected papers created the base for answering the research questions of this literature review, as in Table 32, allowing the goals in Table 31

to be met. Some papers were systematic reviews or surveys (TOOSI, CALHEIROS and BUYYA, 2014) (ZHANG, WU and CHEUNG, 2013) (SILVA, ROSE and CALINESCU, 2013) (PARÁK and ZDENěK, 2014) (GARCÍA, CASTILLO and FERNÁNDEZ, 2016), shortcutting the literature review by avoiding a complete re-analysis of the state of the art. Therefore, such simplification led the review process to not follow the best practices of systematic literature reviews of having multiple analysts that feed their analysis into a data sheet and then discuss to find a consensus about the analyzed papers. Lastly, only two papers were added by the author, as recommendation from supervisor and past literature reviews (OPARA-MARTINS, SAHANDI and TIAN, 2016) (LINARES-VÁSQUEZ, BAVOTA, *et al.*, 2013).

Most of the output of this literature review is documented in the Chapter 5, while a few papers ended up throughout the document, such as in the conclusion and introduction.

APPENDIX M – PETS AND CATTLE IN THE HOTEL CALIFORNIA

This section presents an analogy of the vendor lock-in problem in cloud computing with the music Hotel California (FELDER, HENLEY and FREY, 1977), based on an existing metaphor in the industry (COHEN, 2013), criticizing the fact that leaving cloud providers is not as easy as checking-in. Subsequently, the metaphor *Pets vs Cattle* (BIAS, 2012) is introduced to the discussion, emphasizing how cloud-native application architecture can ease the process of migrating workloads, hence avoiding vendor lock-in.

Cloud Computing at the Hotel California:

The analogy for the need of cloud computing and the vendor choice is presented in the verses represented in Figure 198. The first verse illustrates users acknowledging the existence of a problem: in cloud, they notice that the conventional approach of providing infrastructure does not scale and is not enough anymore; in the music, the character is travelling, tired and needs some rest. A solution appears in the second verse, and they accept it without hesitation: users go to cloud without thinking about the vendor lock-in problem, sometimes not even knowing it exists; in the music, the character enters the hotel for the night. In the third verse, the solution seems perfect for the circumstances: in the cloud, getting started is very easy and convenient, and there is plenty of room at any time of the year, just as in the hotel mentioned in the music.

Figure 198 - Hotel California's first verses

On a dark desert highway, cool wind in my hair
Warm smell of colitas, rising up through the air
Up ahead in the distance, I saw a shimmering light
My head grew heavy and my sight grew dim
I had to stop for the night.

There she stood in the doorway;
I heard the mission bell
And I was thinking to myself
"This could be heaven or this could be Hell"
Then she lit up a candle and she showed me the way
There were voices down the corridor,
I thought I heard them say

Welcome to the Hotel California
Such a lovely place (such a lovely place)
Such a lovely face.
Plenty of room at the Hotel California

Any time of year (any time of year) you can find it here

Source: (FELDER, HENLEY and FREY, 1977)

Cloud computing is like a hotel, with ease and convenience to scale and become a temporary home for users' applications. Both provide the comforts of home without capital expenditure (CAPEX) or the need of doing any of the physical setup and maintenance, presenting the ideal computing model. And it is, until users notice problems as circumstances change and they try to leave.

Figure 199 contains verses that present an analogy to the customers' realization that the adopted choice is not the best anymore and moving applications is not an easy task. The first verse focuses on the character realization that they became a prisoner of the hotel; in the cloud, customers figure out they are locked-in. In the last verse, the character, who is running for the door, tries to get out of the hotel, but the responsible for the establishment says the hotel is only programmed, or designed, to receive, hence they can show intentions to, but they can never leave; in the cloud the situation can be the same, where customers have intentions to migrate their workloads, but they might be locked-in to proprietary solutions in a way that is near impracticable to leave, or at least it is not worth the hassle.

Figure 199 - Hotel California's last verses

Mirrors on the ceiling,
The pink champagne on ice
And she said, "we are all just prisoners here, of our own device"
And in the master's chambers,
They gathered for the feast
They stab it with their steely knives,
But they just can't kill the beast

Last thing I remember, I was
Running for the door
I had to find the passage back to the place I was before
"Relax" said the night man,
"We are programmed to receive.
You can check out any time you like,
But you can never leave!"

Source: (FELDER, HENLEY and FREY, 1977)

The original analogy (COHEN, 2013) states that even within OpenStack there is a degree of lock-in, because one can easily switch only between products of the same platform. While this is technically true, the platform lock-in is drastically different from vendor lock-in, for example OpenStack is a platform that has many vendors, as previously explained in this

chapter. Platform, framework or standard lock-in will always exist in the sense that different solutions implement different interfaces and users adopt their definitions; this is probably what the original author tried to say. This danger is also mentioned in the literature, highlighting that many standards exist and that they might not be interoperable between them (LOUTAS, KAMATERI, *et al.*, 2011).

However, if applications are designed in a cloud-native architecture, the moving will be much simpler to migrate services, as they would not require special customization, being easily reconfigured in a different provider.

Pets vs Cattle:

Pets receive names and when they get sick, one nurses them back to health. This is a metaphor for application architectures that scale up, where servers are like pets: unique, hand raised with love and cared for, with several hard to repeat customizations.

Cattle, in its turn, are numbered, and when they get sick, they get replaced, just as servers in application architecture that scale out, as opposed to scale up, where servers are mostly identical to each other.

Final Remarks:

In the Hotel California landscape, scaling applications up translates into trouble when migrating, since repeating the configuration process is not straightforward, hence leaving users with the alternative of porting their applications to another provider. Portability has its own challenges, such as having a compatible format between the two providers and network reconfiguration upon migration. In contrast with that, scaling applications out means that by updating the orchestration scripts, users can clone the same infrastructure architecture in another cloud, only needing to port stateful resources such as databases. In the case of moving between OpenStack clouds, not even the orchestration scripts would need to be updated because the platform is now interoperable with OpenStack SDK.