Lappeenranta-Lahti University of Technology LUT

School of Engineering Science

Degree Programme in Software Engineering

Roni Juntunen

OPENSHIFT FROM THE ENTERPRISE FLEET MANAGEMENT CONTEXT, COMPARISON

Inspector of the work: Associate Professor Ari Happonen

ABSTRACT

Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Degree Programme in Software Engineering
Roni Juntunen

OpenShift from the enterprise fleet management context, comparison

Bachelor's Thesis 2020

75 pages, 8 figures, 0 tables, 2 appendices
Examiner: Associate Professor Ari Happonen

Keywords: OpenShift, enterprise fleet management, comparison, edge networks, container-based virtualization, vSphere, OpenStack, SUSE CaaS Platform, Nomad, fog05, OpenVIM, Prometheus, Alertmanager

This thesis describes possibilities and compares OpenShift against other competing platforms from the virtual infrastructure manager fleet management perspective. The thesis is made for a company. Need for this work are created by phenomenon such as fast rise of the bandwidth heavy internet-based services. In addition, technological shifts such movement from hypervisor-based virtualization towards container-based virtualization and movement from centralized networking towards distributed networking have influenced creation of this work. First this thesis compares OpenShift against the suitable competing platforms and analyses whether OpenShift is suitable as a virtual infrastructure manager from the fleet management perspective. Next OpenShift application programming interfaces are described in a practical manner. Finally, OpenShift application programming interfaces are found to be sufficiently broad, and it is stated that OpenShift does not cause problems from the fleet management point of view.

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Engineering Science
Tietotekniikan koulutusohjelma
Roni Juntunen

OpenShift yritystason keskitetyn hallinnan näkökulmasta, vertailu

Kandidaatintyö 2020

75 sivua, 8 kuva, 0 taulukkoa, 2 liitettä

Työn tarkastaja:	Associate Professor Ari Happonen
Hakusanat:	OpenShift, yritystason keskitettyhallinta, vertailu, reunaverkot, konttipohjainen virtualisointi, vSphere, OpenStack, SUSE CaaS Platform, Nomad, fog05, OpenVIM, Prometheus, Alertmanager
Keywords:	OpenShift, enterprise fleet management, comparison, edge networks, container-based virtualization, cri-o, Docker, CoreOS, vSphere, OpenStack, SUSE CaaS Platform, Nomad, fog05, OpenVIM, Prometheus, Alertmanager

Tämä kandidaatintyö kuvailee mahdollisuuksia ja vertailee OpenShiftiä muihin kilpaileviin alustoihin virtuaali-infrastruktuurin hallintajärjestelmien hallinnan näkökulmasta. Työ on tehty yritykselle. Tarve työlle aiheutuu ilmiöistä, kuten nopeasta kaistanleveyttä vaativien internet palveluiden lisääntymisestä. Lisäksi teknologiset siirtymät, kuten siirtyminen hypervisor-pohjaisista virtualisointi ratkaisuista kohti konttipohjaisia virtualisointi ratkaisuja ovat vaikuttaneet työn luontiin. Aluksi tämä työ vertailee OpenShiftiä sopiviin kilpaileviin alustoihin ja analysoi OpenShiftin sopivuutta virtuaalisen infrastruktuurin hallintaan keskitetyn hallinnan näkökulmasta. Seuraavaksi OpenShiftin sovellusrajapinnat kuvataan käytännön läheisesti. Lopuksi todetaan, että OpenShiftin sovellusrajapinnat ovat riittävän laajat ja OpenShift ei tuota haasteita keskitetyn hallinnan näkökulmasta.

FOREWORD

The writing of this thesis was not easy task due to constantly expanding scope of the work and often encountered situations, where information was hard to acquire. Information acquisition problems were partly expected, because it was known that the work will include cutting-edge technologies, which are not yet that well studies. Yet, in the end information could still be found, it just took some extra effort to be discovered.

Regardless of the problems I encountered, the work is now done. One of the most important things since the beginning of the work was to create a thesis that will valuable to someone. Hopefully, I have been successful in achieving that goal and you can find something new and interesting from this work. I hope you insightful reading experience!

In the end of this foreword, I would like to thank my family for the constant background support and organizing me from time to time some other thigs to think of. In addition, I would like to thank my friends for keeping me a company through this thesis. Especially I would like to thank my thesis supervisor Associate Professor Ari Happonen and my co-workers, who gave me extremely valuable tips about improvements that could be made.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AWS	Amazon Web Service
API	Application Interface
ASIC	Application-Specific Integrated Circuit
BIOS	Basic Input-Output System
BMC	Baseboard Management Controller
CLI	Command Line Interface
CNCF	Cloud Native Computing Foundation
CRI	Container Runtime Interface
DNS	Domain Name System
EPA	Enhanced Platform Awareness
ETSI	European Communication Standards Institute
GCP	Google Cloud Platform
GSM	Groupe Spécial Mobile
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure As A Service
ICT	Information and Communication Technology
IETF	Internet Engineering Task Force
IOT	Internet Of Things
IP	Internet Protocol
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
LXC	LinuX Container
M2M	Machine To Machine
NFV	Network Function Virtualization
OCI	Open Container Initiative
OCP	Open Container Project
OSM	Open Source Mano
PaaS	Platform As A Service
POC	Proof Of Concept

QL	Query Language
RBAC	Role-Based Access Control
REST	Representational State Transfer
SDK	Software Development Kit
SDN	Software Defined Network
UEFI	Unified Extensible Firmware Interface
UID	Unique IDentifier
URI	Uniform Resource Identifier
VCF	VMware Cloud Foundation
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VR	Virtual Reality
VXLAN	Virtual Extensible Local Area Network

1 INTRODUCTION

1.1 BACKGROUND

It has been estimated that global internet usage will be approximately 4,8 zettabyte (10^{21}) by year 2022, which is 11 time more than 2012 only ten years earlier [1]. Even the mobile consumption is estimated to be aground 158 exabytes (10^{18}) by year 2022 [2]. The estimates may not be totally accurate, but one thing is still certain, and it is the fast rise of internet consumption. Usage of the data will rise in the future due to many different factors. Technologies that will increase the data usage in the future amongst other things are Internet Of Things (IOT), Machine To Machine (M2M) communication, constantly increasing resolution of the video streams and Virtual Reality (VR). [3].

To satisfy increasing bandwidth demands, new technologies and methods must be developed. Traditionally networks have been constructed using specialized networking hardware that is built on top of the Application-Specific Integrated Circuits (ASICs). Nowadays instead of AISCs Software Defined Network (SDN) and tightly related Network Function Virtualization (NFV) is increasingly used due to their flexibility and possibilities to use commodity hardware instead of special hardware. [4] Conventionally SDN and NFV has been constructed using hypervisor-based virtualization technologies, but in recent years emerging container-based virtualization technologies has been proposed to replace hypervisor based virtualization [5]. Change has been planned, because as described later more in depth in the section 2. containers can have better performance and are generally easier to manage.

At the same time as the shift from hypervisor-based virtualization to the container-based virtualization is considered, shift from centralized networking to a more distributed edge centric network is planned too. Move to a more edge centric network architecture is seemed important or even necessary due to expected massive network traffic growth and increasing minimal latency requirements set by the future technologies like self-driving cars. [6] One of the challenges with the edge centric networking is management, because more distributed approach will cause logically significantly larger of smaller installations deployed in a decentralized manner.

1.2 Goals and limitations

This thesis addresses both previously mentioned problems by introducing and analyzing Red Hat's container-based Virtual Infrastructure Manager (VIM) OpenShift from the enterprise fleet management perspective. In this thesis monitoring and management features of the OpenShift are compared against other competing platforms to verify that OpenShift's capabilities are at sufficient level compared to existing solutions or upcoming competitors. In addition to the comparison, this thesis provides comprehensive review to the OpenShift's internal technologies and describes some of the Application Programming Interfaces (APIs) that are provided by the OpenShift.

This work provides answer for the following research questions:

1. How OpenShift monitoring and management APIs perform compared to other competing platforms from the enterprise fleet management perspective?
2. How enterprise fleet management related monitoring and management information can be acquired from the OpenShift cluster in practice from a theoretical API client's perspective?

In addition to the two main research questions, also maintenance perspective of the theoretical fleet manager is considered. Maintenance section consists of observation that were made during API and comparison investigations. It should be also noted that in addition to fleet management perspective this work is also written from the NFV perspective. In addition to this introduction. NFV background mainly influences comparison where candidates are heavily influenced by the NFV needs. Basically, NFV capabilities are used as a filter in this thesis, but facts are mainly presented based on the feet management point of view.

Information for the first research question related to comparison is mainly acquired using technical documentation provided for the different platforms. Information to the second research question is based on the information that is found from the technical documentation, but it has been also heavily complemented with the information acquired using empirical methods. Empirical methods consisted mainly of practical exploration of the APIs.

During the writing of this thesis a fleet manager that can use OpenShift APIs was partly implemented. Due to confidentially reasons the fleet manager, nor integration is not described any further in this thesis. However even though the fleet manager is not described, it should be noted that construction and direction of this thesis was influenced by the fact that fleet manager implementation was ongoing at the same time.

This thesis does not include comparison of all the available VIMs, instead it concentrates on the VIMs that could be most feasible for SDN and NFV use cases. VIM comparison does not contain every possible category that could be needed by the fleet management software. Instead it concentrates on the categories that should generally be beneficial for all fleet manager software and its main goal is to set OpenShift on the map compared to other alternative VIMs. Second research question related to practical usage of the OpenShift APIs explains usage of the APIs from the hypothetical client's perspective. It does not tell about the client, nor define it, instead client is just considered to be plain party that is able to access OpenShift APIs via generic Representational State Transfer (REST) based interface. Maintainability analysis consist only of possibly problems that were observed during API exploration. It does not list every possible issue that may come up in the maintenance phase.

1.3 Thesis structure

Introduction section defines topic of the work and lays out research questions. Section also provides some information about the background of the work and sets limitations for the thesis. **Background** section provides information about the virtualization and defines two different virtualization methods currently in use. It also tells about different container technologies and provides information about container orchestrator called Kubernetes. **OpenShift overview** section describes OpenShift and tells about integral technologies related to OpenShift. Description contains technologies like Oauth and Prometheus.

Comparison with similar platforms section defines reasons for selected platforms and provides description about initially selected candidates. Further section provides information about comparison criteria and contains textual representation of the comparison results. Overall section sets OpenShift in the context compared to other VIMs. **Monitoring & management interfaces** section describes ways that were used to acquire information related to the OpenShift and describes APIs available for monitoring and management interfaces. In addition, section tells about OpenShift installation process.

Analysis section provides analysis based on the comparison information and the provided API descriptions. Section contains analysis about OpenShift feasibility, API usage and maintenance perspectives. In addition, section wraps up those analysis and provides some ideas for further research. Conclusion section wraps most important observations up and concludes this thesis.

2 BACKGROUND

This section provides general information about the virtualization in general and information about different container technologies. Section also describes the software called Kubernetes that can be used for managing the containers on the larger scale and defines CoreOS called operating system.

2.1 Virtualization background

There are two kinds of virtualization methods, traditional hypervisor-based virtualization, and newer container-based virtualization. In the hypervisor-based virtualization whole computer hardware is emulated by the software called hypervisor and a normal operating system is run on top of the virtualized hardware. The hypervisor-based virtualization is further divided in the two categories called type 1 and type 2. The difference between these two categories is that in type 1 hypervisor is run directly on top of the computer hardware and in type 2 hypervisor is run on top of the operating system. [7]

Container-based virtualization works quite differently compared to traditional hypervisor-based virtualization. Instead of emulating whole hardware of the computer, container runtimes are only isolating the processes from each other by providing features like private process space, storage, and network. Containers are like firewalls between different traditional processes and they are run directly inside the kernel that is run on top of the computer hardware. [7]

Both virtualization types have certain advantages and disadvantages. The advantages of the traditional hypervisor-based virtualization is that virtual machines can support basically any kind of hardware composition, architecture and thus operating system that works on the hardware. [7]. Hypervisor-based virtualization is more secure, because virtual machines do not have tight coupling with the hypervisor kernel as the container based virtualization do [8]. The main disadvantage of the hypervisor-based virtual machines is that, they are less efficient compared to containers at least from the application perspective due to often unnecessary overhead caused by the hardware emulation and redundant operating system [7].

The main advantage of the container-based virtualization is that containers are generally more performant and lightweight compared to virtual machines [9]. Efficiency of the containers comes from the fact that they are run directly on top of the one operating system. Due to common kernel, containers have only minimal overhead compared to programs that are run without any kind of virtualization in between. The disadvantages of the containers are same as advantages of the virtual machines, ergo no support for different computer architectures between one installation and not as tight security. [7] The security of the container is currently being worked on, and multiple software and hardware based solutions has been proposed to make container-based virtualization more secure in the future [8], [10].

2.2 Container runtime frameworks

2.2.1 Docker

Docker is an opensource container-based virtualization software that was originally developed by dotCloud and released 2013 [11]–[13]. Docker is a convenient way to package software in a package that is less environment dependent [14]. Better portability is one of the Dockers main differences and advantages compared to earlier technologies like LinuX Containers (LXC) [14], [15]. LXC was mainly centered aground idea of providing lightweight virtual machines by abstracting some details away from the application. Docker instead refined the idea of containerization a step further, and in addition to just running containers, it provides capabilities like fully portable deployments, automatic container building and versioning. [15]

Containers provided by the Docker are run on top of the Linux kernel by utilizing existing Linux technologies like namespaces and cgroups. By utilizing kernel provided features, Docker can provide isolated environments that have their own process handling, storage spacing, and networking. [16] Due to isolated environments it is possible to package everything necessary, like for example application dependencies into the singe container package and output standalone unified deployment ready package [14].

One of the main advantages of the Docker is that all the necessary libraries are readily available inside the container to the application without any manual installation beforehand. Docker based virtualization is also lighter and storage friendlier approach compared to traditional hypervisor-based based virtualization that has bigger overhead and consumes more disk space due to extra operating system. [12], [14] It has been even stated that Docker container can start almost as quick as plain application and use basically no resources if application inside container is not actively doing anything [14].

2.2.2 Cri-o

Cri-o is a lightweight Container Runtime Interface (CRI) compatible container framework, that provides a way to run Open Container Initiative OCI compatible containers [17]. OCI is an open standard made by Linux Foundation, that defines the format of the container images, and a way how those images should be run [18]. By default cri-o uses opensource runc container runtime [17]. Runc was originally part of the Docker but in 2015 it was separated and contributed to the Open Container Project (OCP) that is nowadays called OCI [19], [20]. Cri-o is developed especially for the container orchestration system Kubernetes (introduced later in the section 2.3.) and because of that it even shares the same release schedule with the Kubernetes [21].

2.2.3 Comparison of the Docker and cri-o runtime frameworks

Both Docker and cri-o have one main goal in common, which is to provide a way to run containers, but otherwise they differ significantly on the ideological level. One of the main differences is that cri-o is only trying to offer minimal feature set to satisfy Kubernetes needs, and Docker on the other hand is trying to offer some utilities or even whole ecosystem aground containers in addition to just a bare runtime [15], [21]. For example it has been stated that only five percent of the Docker’s code base is used when Docker is working just as an OCI compatible runtime [22]. This estimate does not give complete picture of the codebase sizes because cri-o needs to also comply with CRI specification, but it gives a good reference point for a comparison. The other major difference between Docker and cri-o is that Docker has its own release schedule and cri-o in sync with the Kubernetes release schedule [21].

2.3 Kubernetes

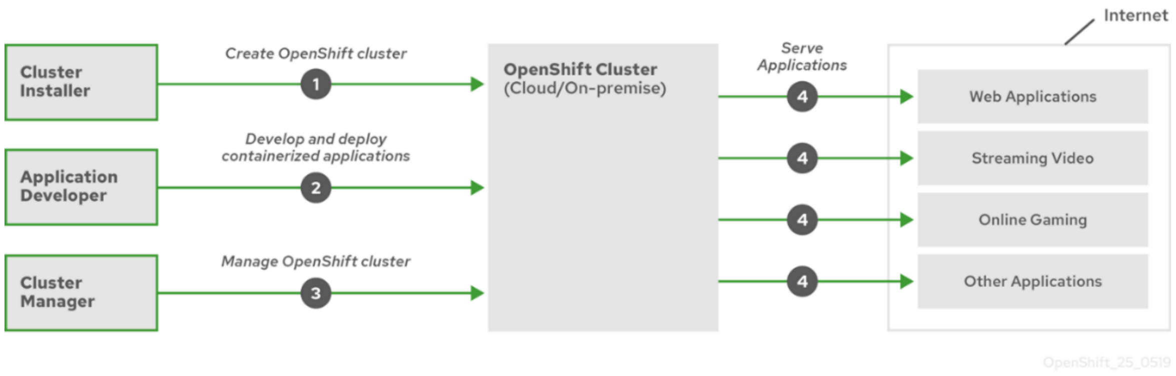
Kubernetes is an opensource container orchestrator for container deployment, scaling and management, that was originally developed by Google and was released in 2015 [11], [23]. Kubernetes is important part of the container ecosystem, because it provides a way to manage large microservice architecture applications spanning across multiple machines by utilizing its distributed configuration and networking capabilities for instance. Bare container runtime frameworks like Docker are mainly meant for managing containers just on a single machine. Kubernetes provides multiple technologies to make cluster management easier like custom network, that makes it possible for containers to communicate with each other regardless of the physical cluster machine. Other useful features that Kubernetes provides amongst other things are automated container scheduling between physical machines, automatic scaling of containers and fault tolerance by restart rules. [24]

2.4 CoreOS

CoreOS is minimal Linux based operating system, that is built to host containers efficiently and automatically [25]. CoreOS operating system was originally developed by the company named CoreOS, that was founded 2013, but was later acquired by Red Hat in 2018 [26]. CoreOS works quite differently compared to other more traditional distributions. Main differences compared to other distributions are that CoreOS root filesystem is mounted as read only (filesystem cannot be modified) and CoreOS does not have its own package manager. Instead of running traditionally packed programs, CoreOS runs all the programs inside containers. Due to containerization of the applications and read only filesystem, CoreOS can be upgraded very easily simply by changing the contents of the root filesystem. [25] This is one of the biggest strengths of the CoreOS working principles.

3 OPENSHIFT OVERVIEW

According to Red Hat’s own definition OpenShift is “a leading hybrid cloud, enterprise Kubernetes application platform” [27]. Enterprise platform is build aground platform as a service (PaaS) ideology [28]. OpenShift can be installed as a service to the public cloud or it can be installed to a user provisioned hardware with the sufficient resources. In practice OpenShift is a management software that provides tools for software developers and system administrators to manage container-based applications comprehensively. OpenShift offers for example an easy way to deploy software directly from the version control or test the application in a production grade like environment. OpenShift works on top of the Kubernetes container management system, which in conjunction manages the Cri-o-based containers. [11], [27] In the **Picture 1**. main users of the OpenShift and possible application deployments inside the OpenShift cluster can be seen.



Picture 1 High level overview of the OpenShift platform. [29]

3.1 OpenShift and OKD

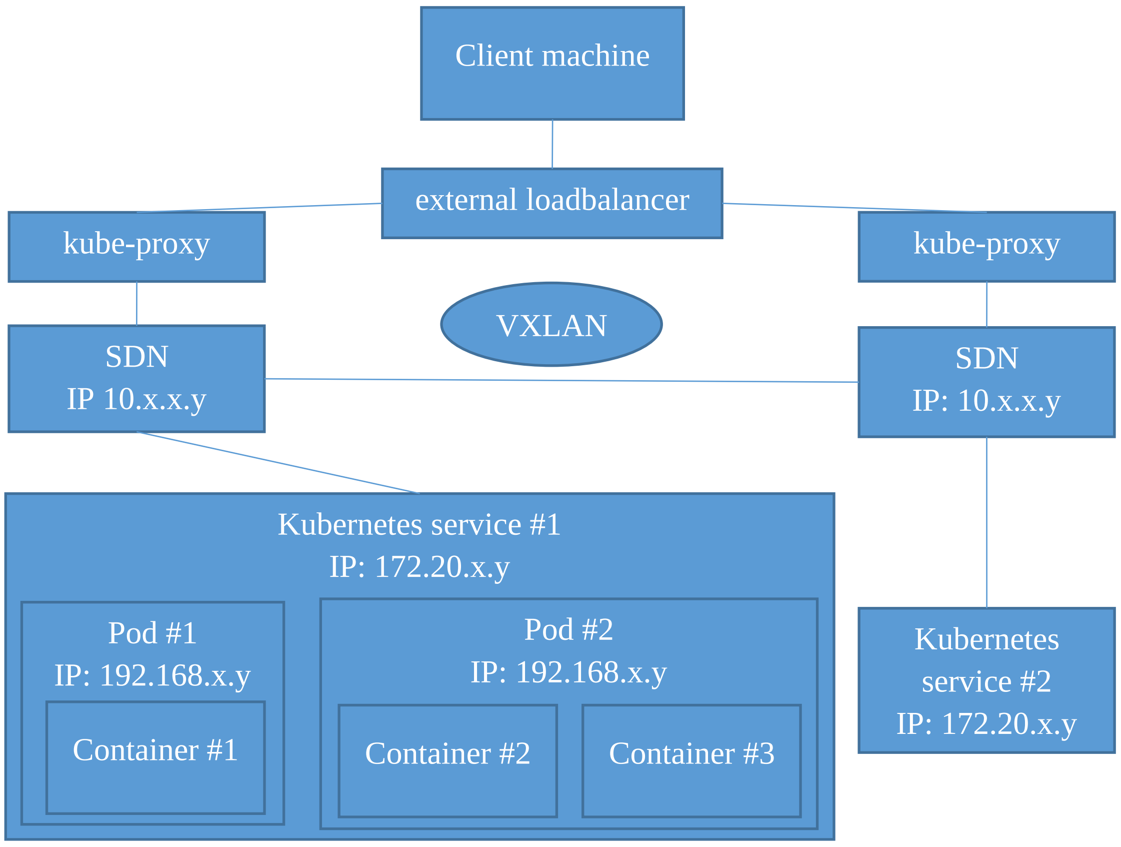
OKD is a opensource community distribution of the Kubernetes, which forms base for the OpenShift container platform [30], [31]. OKD works quite similarly compared to OpenShift, because OpenShift is built on top of the OKD, but OpenShift contains some additional features compared to OKD like Maistra operator [31]. This thesis centers specifically around OpenShift, but there are likely no reasons why OKD, would not work similarly as OpenShift from the fleet management perspective. So likely OpenShift and OKD can be kept as synonyms in the context of this thesis.

3.2 OpenShift and cri-o

Prior to OpenShift version 4.0, OpenShift used Docker as its main container runtime. Docker was changed to cri-o, because cri-o has better security and it is easier to manage between versions. Management of the cri-o is easier than Docker because cri-o share release schedule with Kubernetes. Security of the cri-o is also better than Docker in the OpenShift's use case, because cri-o has smaller codebase and thus smaller attack surface. [32]

3.3 OpenShift networking

OpenShift networking is complex and has multiple layers. **Picture 2** contains overview of the OpenShift networking. **Picture 2** does contain only overview of the OpenShift networking, because networking is not in the main scope of this thesis, but some details are still provided because of networking background of the work.



Picture 2 High level overview of OpenShift networking. [33]

At the top of the **Picture 2** there is a client machine, which represents for example an end users web browser, that is trying to reach an application service for instance on an address foo.apps.bar.com. Foremost the request goes into external transport layer (layer 4) load balancer (for example HAProxy) that evenly forwards request to one of the nodes on the cluster. After load balancing request goes into kube-proxy that resolves proper address of the application service based on the subdomain “foo”. Next the resolution request is forwarded to the SDN. [33]

The SDN takes care of the internal cluster networking and makes it possible for application services to reach any other service on the cluster regardless of the node by utilizing Virtual Extensible Local Area Network (VXLAN) protocol [33], [34]. If the service that kube-proxy is trying to reach is not on the same machine as the kube-proxy instance, cluster machine SDN will reroute the request to a correct cluster machine. The node that finally handles the request has Kubernetes service that logically contains all the resources that one application needs, by providing one common interface (resolvable name and address) for the different application parts. Inside the Kubernetes service there are pods that are used for running different part of the applications like frontend, backend, and database. Pods normally host only one container that runs single process as a part of the application. After the request has finally reached the correct container, application creates response and the request path is back tracked all the way back to the sender. [33]

Everything between Kubernetes services and containers are routed and components can normally see each other inside their own environments. It is however possible to change visibility extensively between Kubernetes services by adjusting the security rules. **Picture 2** presents only overview of the most common networking setup topology of the OpenShift cluster. There are also other networking scenarios supported in the OpenShift like direct port mapping, but they are out of scope of this thesis [33]

3.4 Other OpenShift components

OpenShift has some components that are not directly related to core structure of the OpenShift in a monolithic fashion. Instead of monolithic structure OpenShift itself also build around containers in a microarchitecture fashion [29]. This section gives details about authorization and monitoring component and describes quickly, how OpenShift installer works.

3.4.1 Authorization (Oauth 2)

Oauth 2 is a delegation protocol for user authorization originally developed by Microsoft in 2012 [35]. Oauth 2 is often misleadingly described as an authentication protocol because authentication is often part of the Oauth authorization process. Authorization and authentication are however different concepts and should not be mixed up with each other. In software engineering context, authorization is a way to delegate access to some party without specifying how the authenticity of the party should be exactly verified. Authentication on the other hand is a specific way to make sure that party is the one that it claims to be. [36] Together authorization and authentication form a secure way to get access to a resource.

Rather than providing authentication mechanism, Oauth 2 provides a way to do service (foo) to service (bar) authorization in a way that does not require user to hand over her/his access credentials directly to the service bar that is requesting access. Instead of asking credentials directly from the user, service bar that is requesting authorization for some resource provided by the service foo asks user to authenticate into the service foo. After user has authenticated, service foo asks, whether user would like to approve access for the resource(s) asked by the service bar. If the user grants access to requested services, then service bar will continue its own processes. [36]

OpenShift uses Oauth 2 based authorization scheme and provides its own username and password based authentication mechanism to provide access to the services [37]. Practical flow of the Oauth 2 based authentication is described in the section 5.2.2 in the OpenShift’s context.

3.4.2 Monitoring (Prometheus)

Prometheus is an open source metrics-based monitoring software that was originally created by SoundCloud in 2012. Prometheus is not designed for standalone usage, instead it is meant to be integrated into a pre-existing software stack. There are multiple official and unofficial clients for Prometheus called exporters, that can be used for providing data to the Prometheus. For example, there is exporter for Kubernetes that provides information about Kubernetes cluster. Prometheus metrics can be accessed via PromQL query language that provides some tools for data filtering and combination. [38] In the OpenShift, Prometheus is used as a central piece for information related to the cluster metrics [39]. Practical usage of the Prometheus is described in section 5.2.3.

3.4.3 OpenShift Installer

OpenShift installer supports multiple different kind of installation targets for example bare metal, Amazon Web Service (AWS) and Google Cloud Platform (GCP). Installer works in a declarative fashion and it tries to reach its internally set targets during installation. Installer consists of multiple parts that has different responsibilities like creation of the configuration files, booting of the system with correct parameters and installation of the cluster components. [40] Practical installation of the OpenShift on a bare metal machine is described in the section 5.1.

4 COMPARISON WITH SIMILAR PLATFORMS

4.1 Initial selection for the comparison

Six VIMs were selected as initial candidates for the comparison, because they have been either used in the practice or they have been at least considered to be proper VIMs by European Communication Standards Institute (ETSI) Open Source Mano (OSM) project [41], [42]. ETSI is an Information and Communications Technology (ICT) standardization organization that has been influential in the major networking standards like Groupe Spécial Mobile (GSM) and lately 5G [43], [44]. OSM is an ETSI-hosted initiative to develop opensource software for network management use cases [45].

In addition to other software products that were either demonstrated in practice or considered by ETSI, also openSUSE CaaS Platform was selected as an initial candidate. Exception to the selection criteria was made, because openSUSE's VIM offer is technically similar to the Red Hat's OpenShift due to its Kubernetes centric architecture and Prometheus based metric system [46], [47]. Due to significant technical similarities, SUSE CaaS Platform is interesting reference point for the OpenShift and was included as an initial candidate.

In this thesis public cloud providers are not considered to be proper alternatives, because the public cloud solutions cannot provide necessary high bandwidth NFV capabilities in the minimal latency environments like edge clouds due to natural limits caused by the speed of light. Also, it is assumed in this thesis, that VIM must be fully manageable by the installer so factors like security can be more easily verified. To the first statement there are exceptions like AWS outpost that can be installed locally to the site, but it is completely preinstalled package so it does not satisfy the second statement [48]. Also, API availability of AWS outpost compared to public cloud AWS offering is not totally unambiguous, so it would be hard to verify API availability without access to the installation.

While considering options, it was noticeable that significant chunk of the NFV market has been taken by OpenStack based VIMs. The OpenStack was even stated to be de facto standard of the industry [49]. Due to dominant status of the OpenStack specification, at the time of writing competition on the VIM market was limited, and there were not especially many options to choose from.

4.2 OpenShift alternatives

OpenShift is ideologically and technically quite different compared to other VIMs currently in use. The main difference is that OpenShift is built around PaaS principles and container technologies, instead traditional VIMs which are built originally aground Infrastructure As A Service (IaaS) principles and traditional hypervisor technologies. [28] However in the recent years traditional VIMs have also started supporting container-based virtualization in addition to hypervisor-based virtualization [50], [51]. On the other hand, OpenShift has also lately been testing more traditional hypervisor-based virtual machines via technology called: “container native virtualization”. At the time of writing there was only a technology preview (beta) version of this feature available in the OpenShift. [52]

The difference between PaaS and IaaS models is mainly related to the role of the operating system and runtime. In the PaaS model the operating system and runtime that are running the applications are largely abstracted away from the software itself often practically in a form of containers. In the IaaS model operating system and runtime must be manually installed before installation of the application. [28] See section 2.2. for more information about differences between container-based and hypervisor-based virtualization.

Because OpenShift is based on the IaaS architecture, it is more often compared to other IaaS-based offerings like the public cloud offering provided by the Amazon and Google as the quick search revealed [53]. The difference between OpenShift and traditional VIMs is significant on the ideological and technical level, but it does not cause too significant difference on the monitoring and management side as can be seen in the **APPENDIX A**. For this reason, it is possible to compare the OpenShift also to the more PaaS focused platforms.

4.2.1 OpenStack

OpenStack is an open cloud computing platform standard started as a collaboration between Rackspace and NASA [54], [55]. Implementations of the OpenStack standard has traditionally provided IaaS-based functionality with large set of supplementary services like management and monitoring [56]. There are multiple distributions of the OpenStack standard made by companies such as Red Hat and Oracle [57]. OpenStack architecture is modular, and OpenStack consists of multiple modules intended for different purposes like Nova for computing and Neutron for networking [56]. Modern version of the OpenStack standard includes also Zun and Magnum modules that are providing container management services via Kubernetes amongst other methods [51], [58]. Practically this means that recent versions of the OpenStack standard do provide support for both IaaS and PaaS style computing.

OpenStack is often used as a VIM and it has been even stated that it is “de facto standard for developing the core part of the NFV architecture” [49]. OpenStack dominant status can be also indirectly deduced from the large amount of other papers that raised OpenStack as a reference platform [42], [59].

4.2.2 VMware vSphere

vSphere is VMWare’s server virtualization platform that covers amongst other VMware’s software three products called vCloud Director, vCenter and ESXi [60], [61]. vCloud Director is a management system that is designed to manage large vSphere deployments. vCloud is built on top of the VMware’s prior products vCenter and ESXi. vCenter is a product that is used for managing multiple ESXi installation. ESXi itself is a hypervisor that takes care of the things near the hardware. [62] Compared to vCenter, vCloud works on a more abstract level and provides features like scaling between multiple vCenter installations, consumption of the virtual resources without explicit knowledge about availability of those resources and a way to provision resources to multiple customers without their explicit knowledge about the infrastructure itself [62], [63]. Connections between ESXi (vCenter) instances are build using SDN and VXLAN like in OpenShift [64].

vCenter is a centralized server management software that can manage up to 5 000 physical servers in the full installation [65]. vCenter offers web Graphical User Interface (GUI) and comprehensive set of APIs that can be used for controlling ESXi installations [62], [65]. vCenter offers also features like build in high availability and recovery [65]. vCenter is closest product in the VMware's lineup that can be qualified as a VIM and for this reason comparison mainly tries to center around vCenter provided APIs.

ESXi is a type 1 hypervisor made by VMware. As a type 1 hypervisor ESXi's does have quite limited set of features. On its own ESXi has only a minimal on board debugging and configuration console, network-based management API interface that has capabilities to create virtual machines. [66] ESXi is in fact so small in itself that it has been stated that installation of the ESXi could take only 32MB of storage, but officially 1GB of storage is required by VMware [60], [66].

VMware's products have been traditionally hypervisor based, but latest version of the vSphere called vSphere 7 has support also support for Kubernetes and container-based virtualization via product called VMware Cloud Foundation 4 (VCF). VCF offers hybrid infrastructure services that supports both hypervisor-based and container-based virtualization. VCF utilizes multiple earlier products like ESXi and vCenter to offer its features. [50]

4.2.3 SUSE CaaS Platform

SUSE CaaS Platform is a Kubernetes based container management solution made by SUSE since 2017, that provides capabilities to deploy, manage and scale container-based applications and services [46], [67]. The container platform has centered around three main components, orchestration, OS for microservices & containers and configuration [68]. SUSE CaaS platform also offers management features like health monitoring and non-disruptive rollout or rollback of the applications. The Cloud Native Computing Foundation (CNCF) has certified SUSE CaaS platform as an official Kubernetes distribution. SUSE CaaS platform can be installed to bare metal as well as private cloud hosted using VMware's vSphere. [46]

4.2.4 Nomad

Nomad is a minimalistic workload orchestrator made by HashiCorp that also fits ETSI MANO VIM model. Nomad architecture consist of client and server nodes. Server nodes are used for scheduling of the workloads and client nodes take care of the execution of the processing. [42] Nomad is one self-contained binary that requires only 75MB of disk space. Nomad supports both container- and hypervisor-based workloads. [69]

4.2.5 fog05

fog05 is a decentralized end-to-end management solution made by Eclipse for managing computing resources like CPU, storage, and networking. fog05 has unified plugin-based architecture that enables fog05 to support multiple operating systems, virtualization technologies and network scenarios. [70] For example, fog05 support via plugins natively both Linux and Windows based platforms and offers way to do virtualization either via container-based or hypervisor-based virtualization. [71] At the core fog05 to provide a solution that follows fog computing paradigm [70]. Fog computing refers basically to cloud computing that is done near the users at the edge [72]. Practically fog05 is developed especially edge cloud use cases in mind.

4.2.6 OpenVIM

OpenVIM is a reference VIM made by OSM for deploying virtual machines [73]. OpenVIM is designed especially Enhanced Platform Awareness (EPA) support in mind [73], [74]. EPA includes requirements like hugepages memory or CPU pinning [74].

4.3 Final selection for the comparison

For the comparison three of the six products were selected. Three products selected for the comparison were OpenStack, VMware vSphere, and SUSE CaaS Platform. OpenStack and vSphere were selected, because they are possibly the two most influential contestants on the market and because of that should be considered as reference points in VIM comparison. OpenStack specification itself was taken directly to the comparison instead of any vendor specific implementations, because many of the vendor specific implementation like Red Hat’s version did not have at the time of writing their own API documentation available [75]. SUSE CaaS Platform was chosen as a third product for comparison, because it has significant amount of similarities with OpenShift and because of that is an interesting reference point even if it is not especially designed for NFV use cases.

Three alternatives Nomad, fog05 and OpenVIM were dropped from the final comparison. Nomad was dropped from the comparison because it is not on a list curated by the OSM project and in addition it is not especially designed for the networking use cases. However dropping the Nomad was not totally unambiguous, because Nomad has been demonstrated to be used as a VIM [38]. Fog05 was dropped from the comparison, because it is still under heavy development based on the 0.2.0 version number and did not have at the time of writing almost any monitoring functionalities or even a way for authentication [76], [77]. Fog05 might be interesting platform in the future as it has even now interesting Proof of Concepts (POCs) like 360 video application on the edge installation, but platform is not ready yet for the comparison [78]. OpenVIM was dropped from the comparison because it is more of a reference design and POC than a real product and likely does not include very broad management possibilities based on the low information that is available from it.

4.4 Version selection

Latest available version of the products at the time of comparison was used in the comparison to give most recent status of the products. Specific version numbers are described in the header of the comparison table that is in the **APPENDIX A**.

4.5 Information gathering

Information about the VIMs is primarily gathered from the vendor provided API documentation. If the information could not be found from the API documentation, then other vendor provided documentation like GUI documentation is used. If the information could not still be found, then unofficial documentation and information was used. This information was mainly used to verify that the specific feature was nonexistent. Feature was considered available, if there were hints that feature is somehow exposed for programmatic remote usage.

Information was also gathered through practical API investigation in OpenShift case, because during writing of this thesis access to OpenShift cluster was accessible. Enough high-level access to other environments was not readily available so they could not be tested, nor their features be verified in practice. Some of the platform could have been possibly installed, but due to time constraints this was not done. Because of the missing access to installations OpenShift might have slight advantage or disadvantage compared to other VIMs that were measured solely based on the available documentation. As implied in the section 5. introduction it is hard to completely understand the VIM just based on the documentation.

4.6 Comparison criteria

Criteria for a comparison are chosen based on the large-scale enterprise fleet management perspective. In practice this refers to information that would be good to be visible at a single glance from a fleet of computing environments. Basically, this covers information like usage statistics, fault management, version information and centralized upgrade management. This kind of information should be available centrally because it is impractical to log into every environment manually. For example, it would waste considerable amount of time to check that there is enough disk space available, let alone install updates one by one to the machines. Criteria were also partly selected based on the ETSI NFV standard overview that describes amongst other things levels of monitoring accuracy that should be provided by the VIM [79].

4.7 Management feature comparison

This section contains textual representation of the feature comparison table accessible in the **APPENDIX A**. Section in the appendix are divided in the subsections that are named based on the table type. See tables and for full feature comparison and source information.

4.7.1 Access methods

There are three access methods that were compared, Web GUI, Command Line Interface (CLI) and REST API. OpenShift and OpenStack fully supported all the methods, but unexpectedly vSphere and SUSE CaaS Platform did not check all the marks. As stated in the table note #17 some of the vSphere APIs are REST based, but for some APIs there is only Software Development Kit (SDK) available. In SUSE CaaS Platform there is no official GUI that is fully supported, but for example Kubernetes native control panel can still be installed.

4.7.2 Language bindings

Situation with the language bindings is quite varying between VIMs. OpenShift had partial support for Kubernetes language binding due to its Kubernetes base, but those bindings did not provide support for OpenShift specific APIs. OpenStack had official Python API and unofficial bindings for other languages. vSphere had support for many different languages depending on the API, but there was no single language that would support all the available APIs. SUSE CaaS Platform had excellent support for all the languages that were listed due to its strict compliance with Kubernetes standard.

4.7.3 Authentication

There are three authentication categories in the table that are: username / password, X.509 like certificate, other. OpenShift, OpenStack and SUSE CaaS Platform supported all the ways. vSphere had some support for certificate based authentication and other authentication schemes, but situation with the authentication methods were not entirely clear based solely on the documentation and seemed to be differing based on the API and even endpoint [80].

4.7.4 System information

Unexpectedly information collection about system information and versions was not very well supported. vSphere had best support for acquiring system information and was only VIM that had any kind of support for firmware or Baseboard Management Controller (BMC) related information. vSphere was also only platform that had direct API for requesting information about system time.

OpenShift supported acquisition of the basic information like VIM version and operating system. OpenStack had support for querying VIM version. SUSE CaaS platform did not have proper support for querying any information, but there were Kubernetes APIs that provided some basic information.

4.7.5 Monitoring / statistics

Basic monitoring querying was generally well supported. Support even for specific CPU, RAM and disk usage information was provided by all the platforms. Two metrics namely overall pod/Virtual Machine (VM) count and per machine pod/VM count were only supported directly by the OpenShift while with other platforms some workarounds needed to be used. Proper information about some of the SUSE CaaS Platform metrics capabilities could not be checked, because similarly to OpenShift, exact Prometheus metrics were not documented. It is quite likely that unverified metrics are also available in the SUSE CaaS Platform based on the OpenShift Prometheus, but this cannot be verified without access to the installation.

4.7.6 Alerts

Alerts were also well supported between different platforms. All the platforms supported information like alert name, description, and severity. Only exception was alert Unique IDentifier (UID), that was only supported by the OpenStack.

4.7.7 Upgrade

Update processes varied wildly between different products. OpenShift had especially easy upgrade process that is described more thoroughly in the section 5.2.4. OpenStack upgrade process was quite manual and all the services in every node needs to be upgraded one by one [81]. vSphere had multiple upgrade methods that could be used to update the platform. One of those methods was Update Manager that can automate the upgrade process [82]. Upgrade process of the SUSE CaaS Platform was quite manual and requires nodes to be upgraded one by one [83]. Automatic firmware upgrade was only somewhat supported by VMware and even those details were not completely clear.

4.7.8 Scores sum

OpenShift got 24/35, OpenStack 23/25, vSphere 22/35, and SUSE CaaS Platform 19/35 points. In theory OpenShift took the victory, but it must be taken into consideration that five points of the SUSE CaaS Platform could not be verified due to a missing information related to monitoring statistics. So, the situation could have been different if the information would have been available. Overall points also are not very informative, because some of the features might have greater significance than another. For example, seamless live upgrade is likely more significant feature than support for one more language binding. Points do not have any weighting in the table, because it is impossible to determine exact value of the feature without any external use case prioritization.

5 MONITORING & MANAGEMENT INTERFACES

This section provides more comprehensive information about OpenShift specific monitoring and management APIs. Selection criteria for the API exploration are same as criteria for the comparison, described more thoroughly in the section 4.6. APIs described in this section are presented in a practical manner. The format was chosen, because during the writing of this thesis also real client implementation related to some of the APIs were created. As stated in the section 1.2, due to confidentiality reasons, the implementation of the actual client is not described any further in this thesis. Instead the usage of the APIs is only described strictly from the OpenShift point of view.

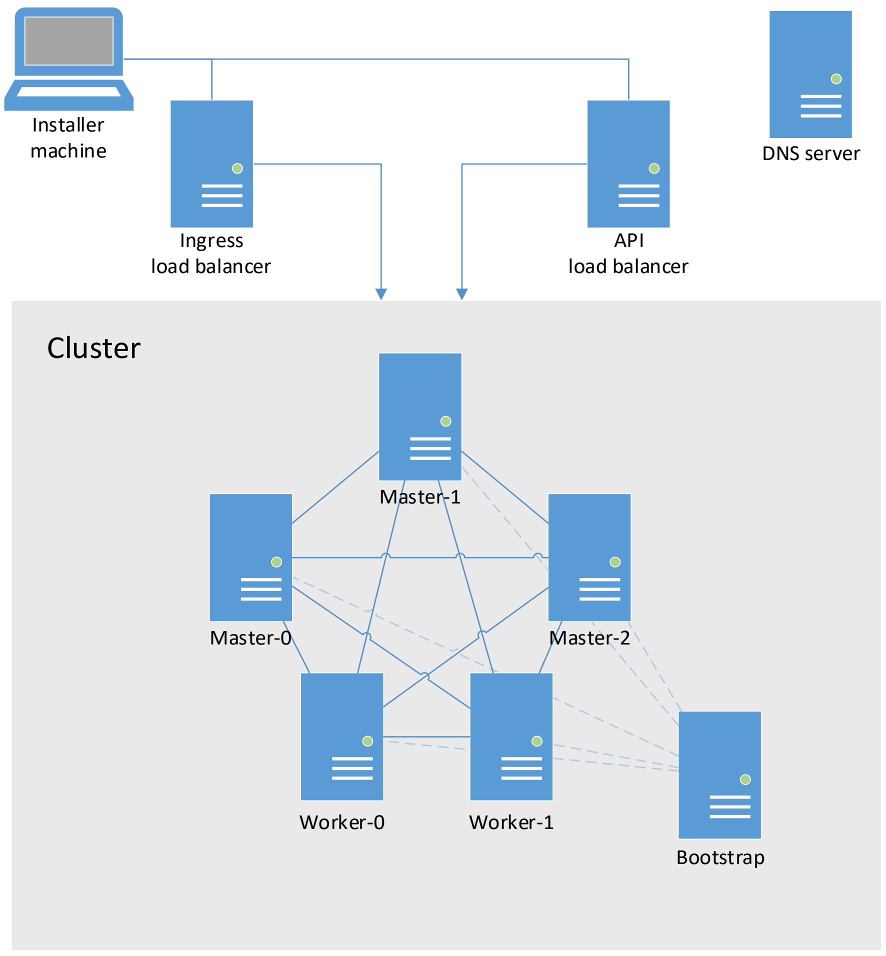
During the investigation of the APIs practical research methodologies were partly used. Practical exploration was necessary during writing of this thesis, because like larger software systems in general OpenShift is hard to comprehend at an adequate level just based on the large amount of documentation. By testing out the system in practice it was easier to understand possibilities and limitations of the system.

5.1 Installation of the OpenShift

Before any experiments with the OpenShift could be carried out, it was necessary to install the system. From the management perspective this was particularly important. At the time of the writing official OpenShift manual did not cover all the available REST APIs in its own documentation related to the monitoring, nor clearly describe exact JavaScript Object Notation (JSON) responses of those APIs in multiple cases [84].

5.1.1 Installation environment

Test cluster was installed on the machines that fulfilled OpenShift's minimum bare metal system requirements: 4 CPU cores, 16 GB RAM and 120 GB storage. Required minimum of five machines were used for installation as can be seen from the **Picture 3**. [85]



Picture 3 Overview of the OpenShift cluster [85].

As can be seen from the **Picture 3** in addition to the machines participating the cluster, also two machines were used for load balancing and one for installation management purposes. Domain Name System (DNS) machine was preconfigured externally and its configurations were modified in order to comply with 5 DNS entries as mandated by the installation manual [85], [86]. Non cluster machines were virtual machines to save resources.

Bootstrap machine seen in the **Picture 3** is not actually a physical nor virtual machine, bootstrap is one of the cluster machines that is only temporary ran in the bootstrap mode so the installation of the cluster can be initiated. After installation of the master nodes is completed bootstrap node is reinstalled as a worker node. This is possible because master nodes will act as bootstrap node for the worker nodes [40].

5.1.2 Pre installation

Before actual installation of the OpenShift cluster can be initiated, it is necessary to satisfy following preinstallation requirements as stated in the installation document [85]:

- Configuration of Hypertext Transfer Protocol (HTTP) server [87]
- Configuration of the load balancers
- Configuration of the DNS entries
- Configuration of the firewalls

HTTP configuration server was prepared as instructed in the documentation. First installation configuration file was manually written with all the necessary details like machine count and network information. Next the manually written configuration was turned into automated installation packages using provided OpenShift installation utility. Finally, Generated installation packages were moved to the Nginx based HTTP server.

In the test Cluster configuration, HTTP server was installed on the installer machine. HAProxy load balancer was configured to the two virtual machines to satisfy load balancing needs. DNS entries were configured using private DNS services. Firewall requirements were fulfilled by external firewall administrators.

In addition to satisfying requirements set by the OpenShift, the boot iso was modified to reduce manual typing during the boot process. Manual typing was required, because Internet Protocol (IP) addresses of the installation environment’s network were statically assigned. Boot parameters were altered by mounting the International Organization for Standardization (ISO)-file and copying its contents to a temporary directory [88]. After that static boot parameters like address of the configuration HTTP server were added to the boot configuration. Then boot ISO was repacked with mkisofs Linux utility.

5.1.3 Installation

The actual installation was started by manually logging into every cluster machine via BMC and using BMC provided graphical remote management interface. At the time of writing latest OpenShift installation ISO 4.4.3 was used for starting the installation process by using

BMC ISO redirection. After the machine started successfully, static IP address configurations were modified in the boot parameters and boot process was continued.

5.1.4 Finishing installation

Booting of the machines was last manual step that was necessary to get cluster into working state. OpenShift automatic installation process takes care of rest configuration activities like Kubernetes orchestrator for example. After automated installation process was completed inside the cluster machines, it was checked that operators and everything else was running smoothly as suggested in the installation manual [85].

5.2 Monitoring and management APIs

There are literary hundreds of API calls available in the OpenShift as can be seen from the API endpoints list [89]. From the general monitoring (management) point of view the following list could be considered most crucial for the fleet manager client use case:

- Authorization / authentication
- Monitoring (CPU, RAM, and disk statistics)
- Alarms (automatically detected issues)
- Version information and upgrade

At the time of writing, OpenShift’s main REST API documentation did not contain information about the APIs that are not directly part of the core system like Prometheus and Oauth2. However, documentation contained information about the discovery of those less tightly coupled APIs. For example, address of the monitoring and alarm server Prometheus can be discovered through the main API endpoint but must be accessed using discovered address and used according to Prometheus’ own documentation.

5.2.1 Investigation of the APIs

Although the API endpoint list is extensive at the time of writing it did not contain enough information to deeply analyze API integration possibilities as described in section 5.1 To acquire missing information, it was necessary to use supplementary information sources like

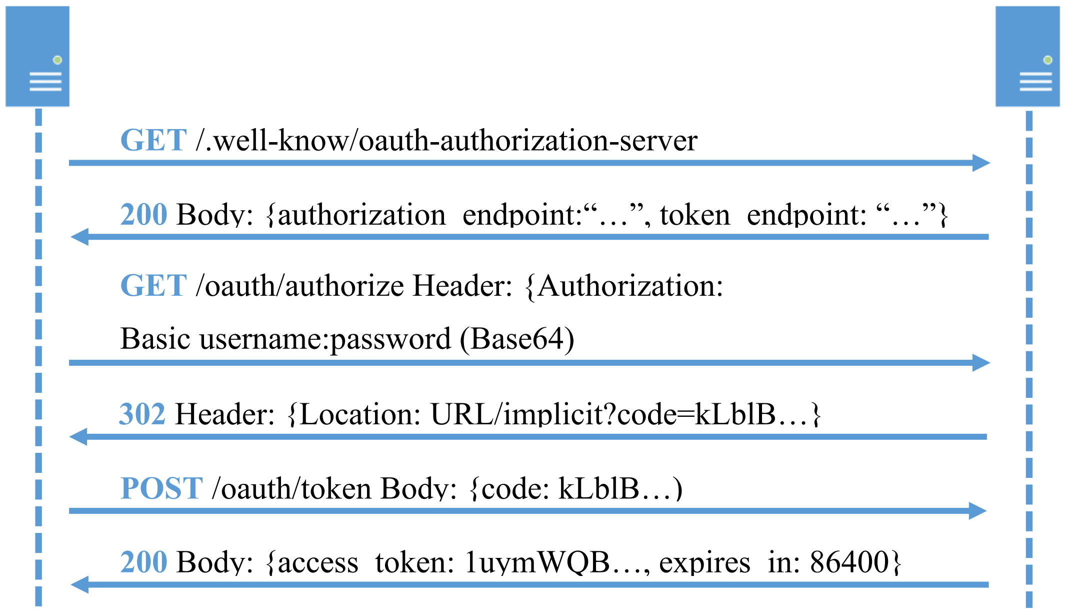
Prometheus' own documentation. Also, practical experimentation with the APIs was sometimes used.

The practical experimentation, amongst other methods, was carried out by following the communication made by the official CLI via debug possibilities exposed using the "loglevel"-parameter. The provided debug information included almost full details of the HTTP communication between CLI and the cluster except for details of the request body. Another practical information gathering method that was used, was to follow OpenShift's control panel generated API calls via debug tools included in the web browser. During authentication API exploration also source code of the CLI was studied due to missing request body information in the CLI's loglevel parameter debug prints.

5.2.2 Authorization

OpenShift authorization uses centralized Oauth 2 server to handle all authorization requests [37]. Oauth principles are explained more thoroughly in the section 3.4.1. OpenShift's Oauth server follow the Oauth standard and support normal Oauth authorization flows [35]. Based on the documentation inside authorization server there are two different authentication clients in the OpenShift oauth server available called: "openshift-browser-client" and "openshift-challenging-client". Browser client is used for interactive authentication sessions where user can interactively authenticate and make authorization decision. Challenging client is meant to be used with CLI based system for command line authentication. [37] Note that specific authentication client details are not directly part of the Oauth protocol. Instead these two authentication methods just ought to be one implementation specific way to fill the authentication needs.

This chapter introduces two explanations for Oauth authorization flow seen in the **Picture 4**. The first way describes the flow as it is used in the OpenShift challenging client. This is not a traditional Oauth style, because in this authorization flow user credentials are directly inputted from the client application instead of asking user to authenticate elsewhere and then getting delegated access. The second explanation describes authorization flow in a more traditional Oauth fashion by explaining access delegation style authorization. This style is used in the browser client [37]. **Picture 4** is drawn based on the first method, because of that some parameters are inaccurate for the second explanation.


GET /.well-know/oauth-authorization-server
200 Body: {authorization_endpoint:"…", token_endpoint: "…"}
GET /oauth/authorize Header: {Authorization: Basic username:password (Base64)
302 Header: {Location: URL/implicit?code=kLblB…}
POST /oauth/token Body: {code: kLblB…)
200 Body: {access_token: 1uymWQB…, expires_in: 86400}

Picture 4 Overview of the OpenShift authorization process.

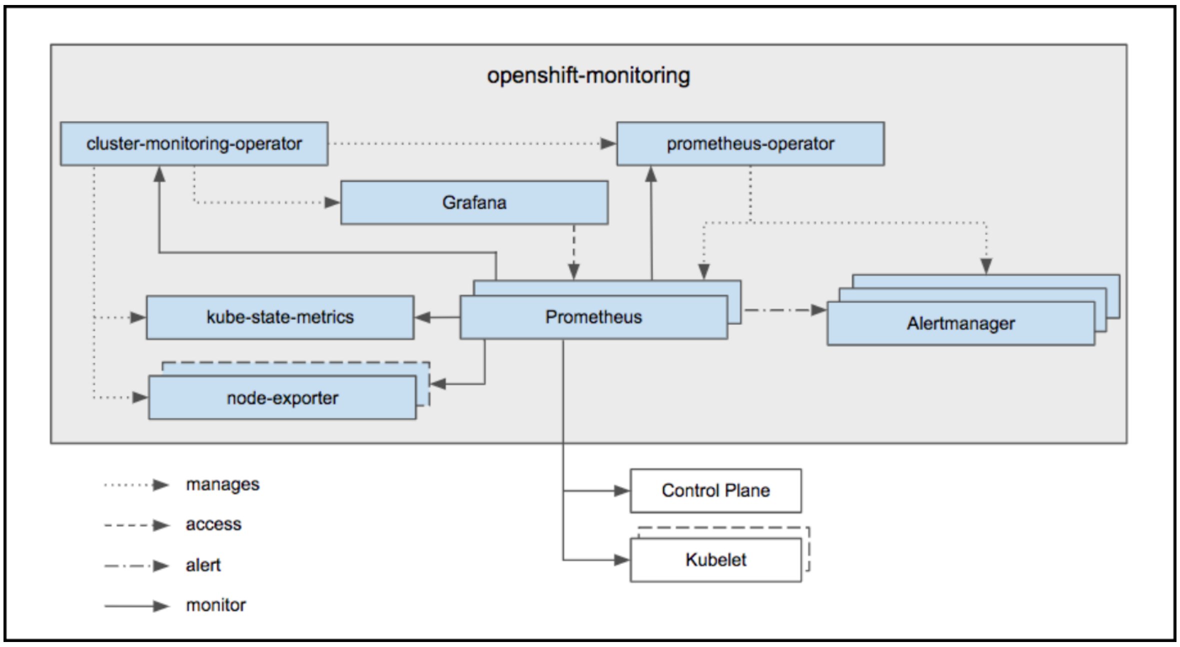
Picture 4 describes the necessary steps to acquire proper access token for a cluster via challenging client. The first step of the process is to identify location of the authorization server endpoints via provided well-known resource. After location of the server has been identified, username and password are sent to authorization endpoint. Authorization endpoint responds with the redirect response which contains temporary authentication code to continue the authentication process. This code is then used against token endpoint which issues a token with longer expiration time. The token must be used inside authorization header in order to access other API endpoints as described in Oauth 2 Internet Engineering Task Force (IETF) proposal [90].

Picture 4 can be also interpreted as a flow for acquiring authorization through a browser client. First information about endpoints are acquired via well-known resource. Authorization request is sent to authorize endpoint with client identification and redirect Uniform Resource Identifier (URI). After user has authenticated and approved client's request, client provided redirect URI will be accessed with an authorization code by user web browser. Finally, client issues the authentication token to the token endpoint and receives authorization token that can be used for issuing requests to the server. [35]

5.2.3 Metrics and alarms

Metrics and alarms in the OpenShift are based on the open-source monitoring and alerting toolkit called Prometheus [91], [92]. Prometheus collects data from the Kubernetes clusters and offers flexible PromQL query language for accessing alerts and metrics of the cluster. More information about Prometheus can be found from section 3.4.2. At the time of writing Prometheus could be accessed via: “prometheus-k8s-openshift-monitoring” subdomain.

Picture 5 provides overview of the OpenShift monitoring stack and describes dependencies of different monitoring components. **Picture 5** is made for older OpenShift 3.11 release. It is possible that there are small differences compared to current 4.4 version release. However no major differences between 3.11 and 4.4 versions has been noticed.

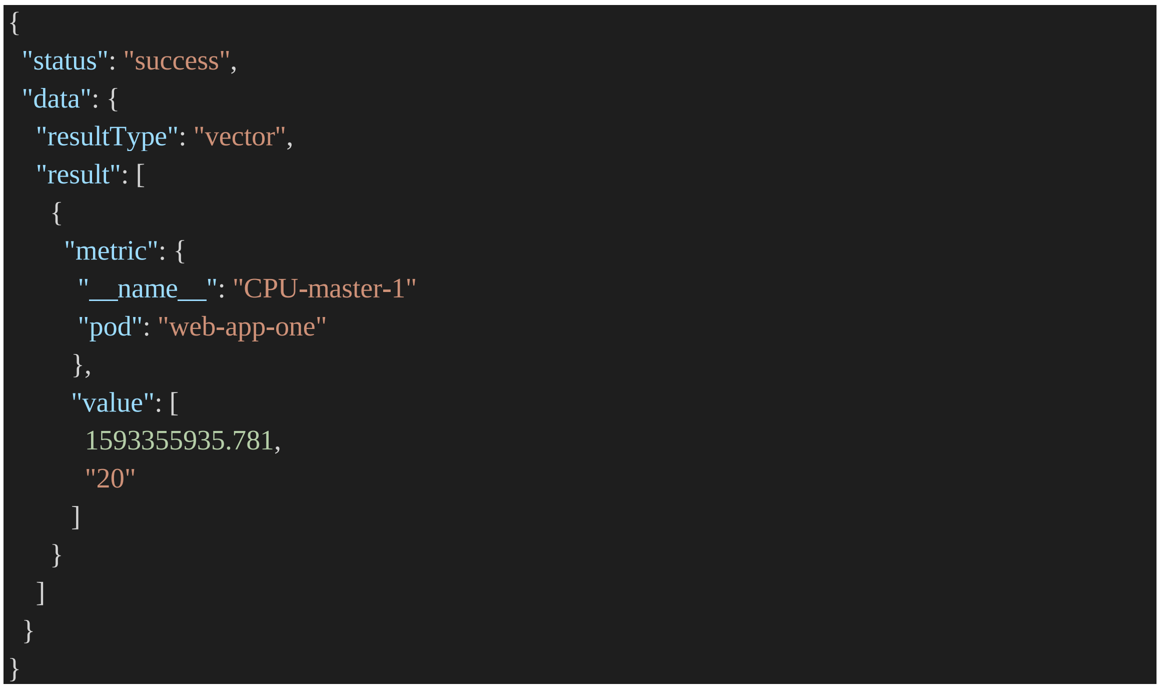


Picture 5 Overview of the OpenShift monitoring stack on OpenShift 3.11. [91]

From the integration point of view Prometheus can be used directly for the monitoring data extraction via PromQL queries. OpenShift’s Prometheus server offers very broad set of monitoring metrics that can be used for monitoring basically every aspect of the cluster like

CPU, RAM, disk, and network usage. Prometheus also offers metrics at multiple degrees of accuracy, for example at cluster, node, pod, and container level. Metrics provided by the OpenShift are not separately documented, but they are easily accessible and testable via OpenShift web GUI [93].

Picture 6 Prometheus shows example of the normal simple Prometheus response. Prometheus response consists of metadata in the form of “resultType”-field and “metric”-object and data inside “value”-array. “Metric”-object contains miscellaneous key value pairs that can be used for combining multiple different kind of metrics. For example, “metrics”-object’s filed “pod” can be used for combining CPU and RAM data together, when two queries to Prometheus are issued. Prometheus’ Value-array contains time in a UNIX-timestamp format and value as a string related to data of the metric.



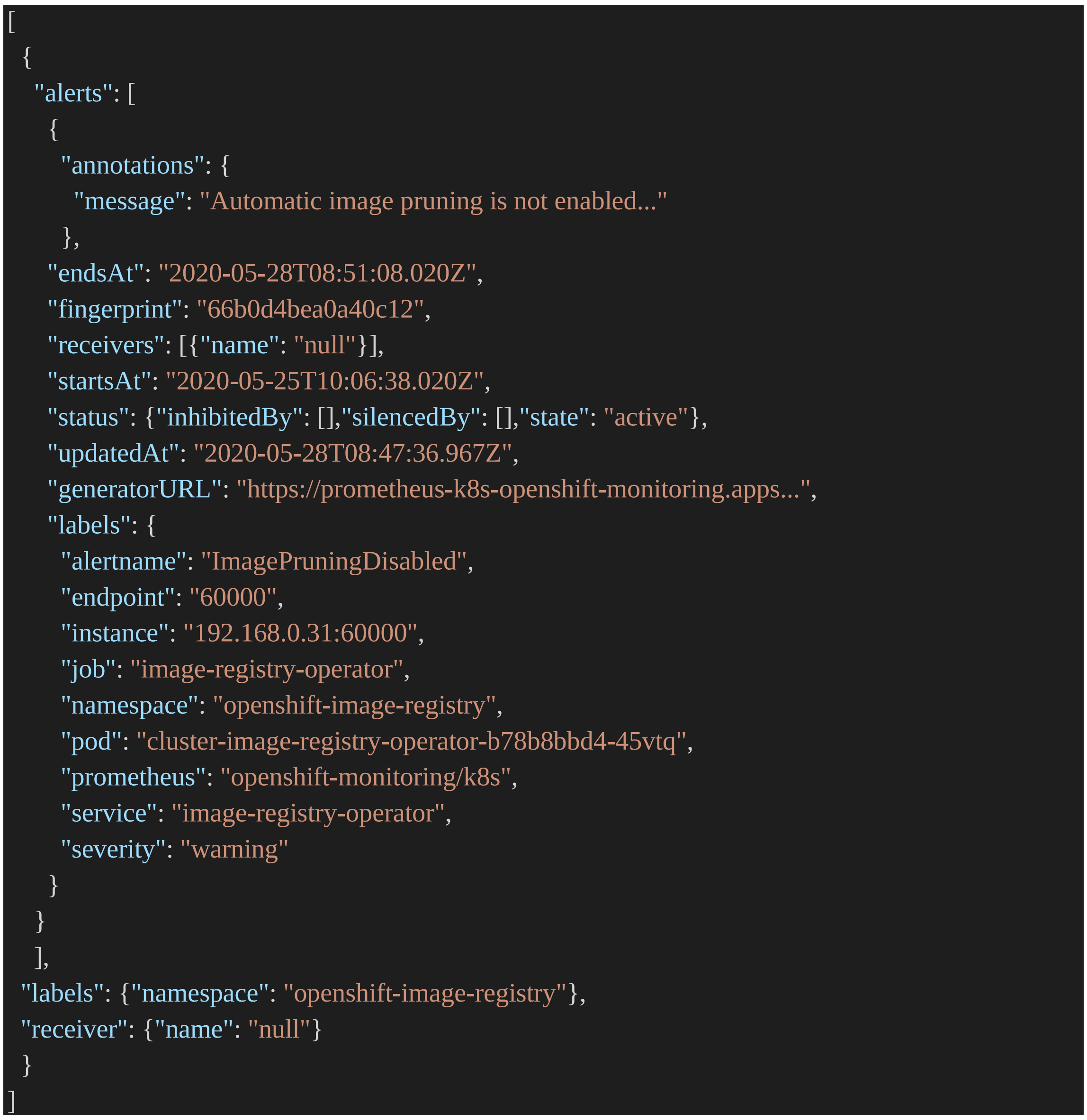
```
{
 "status": "success",
 "data": {
  "resultType": "vector",
  "result": [
   {
    "metric": {
     "__name__": "CPU-master-1"
     "pod": "web-app-one"
    },
    "value": [
     1593355935.781,
     "20"
    ]
   }
  ]
 }
}
```

Picture 6 Prometheus example response

There are also other response types called matrices and scalars, that slightly alters the form of the responses. Matrix for example has values-matrix instead of a value-array, and contains multiple values over time period for instance. [94] Main structure of the response persist similar between different types and only the amount of content inside two arrays seen in the **Picture 6**. changes.

In addition to metrics, Prometheus is also used for storing alerting information, as can be deduced from the **Picture 5**. Direct retrieval of the alerting information via Prometheus is unnecessarily complex, because APIs provided by the Prometheus core may contain old and duplicated alerts [95]. A more elegant way to retrieve information about the alerts is to use Prometheus component called Alertmanager. As can be seen from the **Picture 5**, Alertmanager receives alerts from the Prometheus. After receiving data Alertmanager processes it by deduplicating, grouping and silencing specific alerts [95]. By utilizing Alertmanager APIs clean stream of alerts can be easily accessed.

Responses produced by Alertmanager are a bit more complex than Prometheus metric responses as can be seen from **Picture 7**, but they are still quite clear. Alertmanager response contains fields such as “alertname” inside “labels”-object and “message” inside “annotations”-object that describes respectively a name and description of the alert. There is also field like “severity” inside “labels”-object that can be used for classification of the alert’s urgency. Other metadata fields like “pod” inside “labels”-object for example can be also beneficial and can be used for quickly determining the source of the alarms.

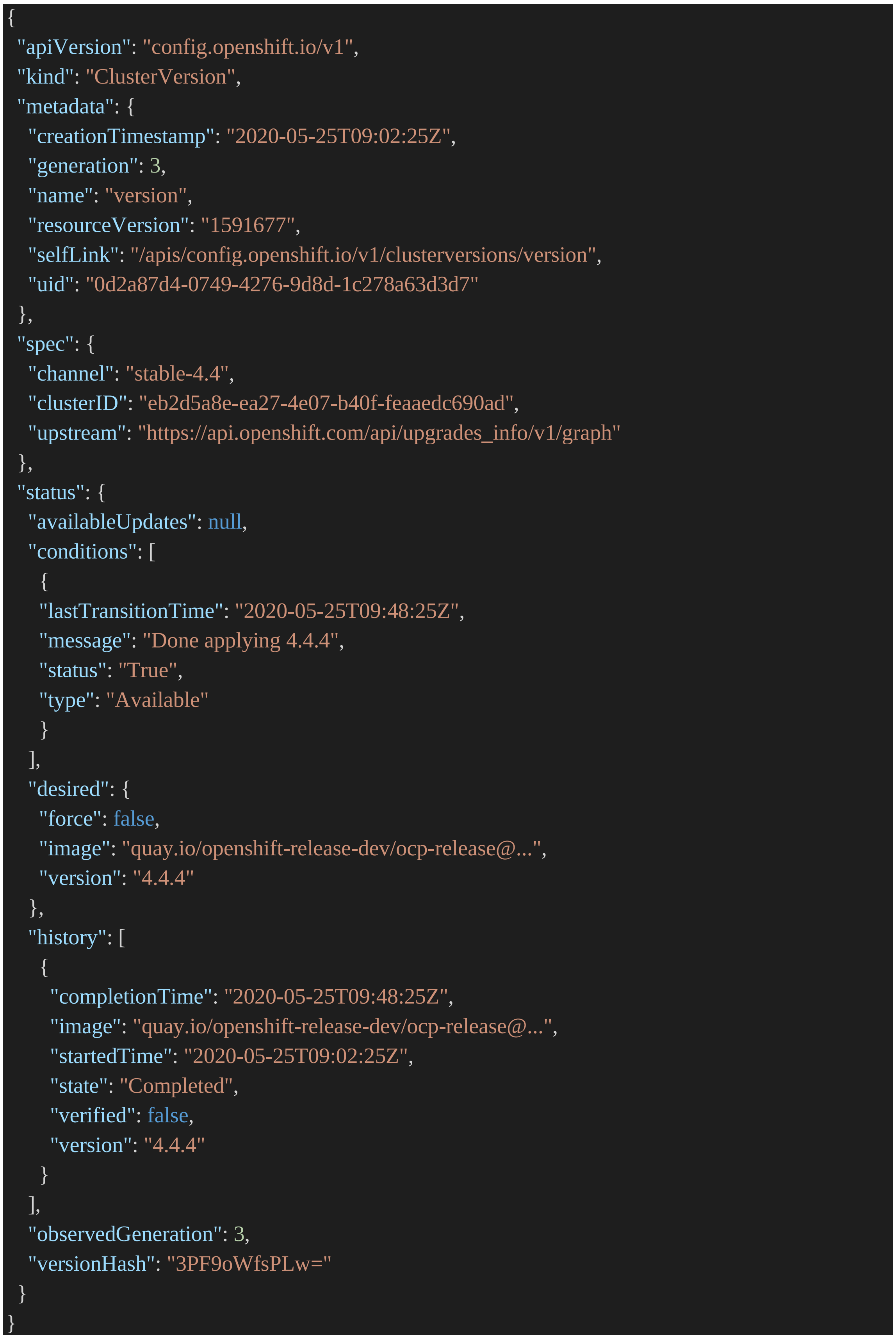


```
[
  {
    "alerts": [
      {
        "annotations": {
          "message": "Automatic image pruning is not enabled..."
        },
      "endsAt": "2020-05-28T08:51:08.020Z",
      "fingerprint": "66b0d4bea0a40c12",
      "receivers": [{"name": "null"}],
      "startsAt": "2020-05-25T10:06:38.020Z",
      "status": {"inhibitedBy": [],"silencedBy": [],"state": "active"},
      "updatedAt": "2020-05-28T08:47:36.967Z",
      "generatorURL": "https://prometheus-k8s-openshift-monitoring.apps...",
      "labels": {
        "alertname": "ImagePruningDisabled",
        "endpoint": "60000",
        "instance": "192.168.0.31:60000",
        "job": "image-registry-operator",
        "namespace": "openshift-image-registry",
        "pod": "cluster-image-registry-operator-b78b8bbd4-45vtq",
        "prometheus": "openshift-monitoring/k8s",
        "service": "image-registry-operator",
        "severity": "warning"
      }
    }
    ],
  "labels": {"namespace": "openshift-image-registry"},
  "receiver": {"name": "null"}
  }
]
```

Picture 7 Alertmanager example response

5.2.4 Version information and upgrade

Version information can be retrieved directly from the main API endpoint. Version endpoint response contains comprehensive information related to cluster version like basic version information, history and upgrade information of the cluster as can be seen from the **Picture 8**.



```
{
 "apiVersion": "config.openshift.io/v1",
 "kind": "ClusterVersion",
 "metadata": {
  "creationTimestamp": "2020-05-25T09:02:25Z",
  "generation": 3,
  "name": "version",
  "resourceVersion": "1591677",
  "selfLink": "/apis/config.openshift.io/v1/clusterversions/version",
  "uid": "0d2a87d4-0749-4276-9d8d-1c278a63d3d7"
 },
 "spec": {
  "channel": "stable-4.4",
  "clusterID": "eb2d5a8e-ea27-4e07-b40f-feaaedc690ad",
  "upstream": "https://api.openshift.com/api/upgrades_info/v1/graph"
 },
 "status": {
  "availableUpdates": null,
  "conditions": [
   {
   "lastTransitionTime": "2020-05-25T09:48:25Z",
   "message": "Done applying 4.4.4",
   "status": "True",
   "type": "Available"
   }
  ],
  "desired": {
   "force": false,
   "image": "quay.io/openshift-release-dev/ocp-release@...",
   "version": "4.4.4"
  },
  "history": [
   {
    "completionTime": "2020-05-25T09:48:25Z",
    "image": "quay.io/openshift-release-dev/ocp-release@...",
    "startedTime": "2020-05-25T09:02:25Z",
    "state": "Completed",
    "verified": false,
    "version": "4.4.4"
   }
  ],
  "observedGeneration": 3,
  "versionHash": "3PF9oWfsPLw="
 }
}
```

Picture 8 Version resource example response

In addition to broad version information, version resource can be also used for triggering update of the OpenShift installation. As stated in the OpenShift's integrated web GUI documentation, it is possible to trigger upgrade of the OpenShift cluster by adding object "desiredVersion" to the version resource's spec with a new version specification as a value. "desiredVersion"-object should contain similar structure as "desired"-object with the details about the new version. In addition to taking proper backups, this should be the only thing that must be done manually to upgrade the OpenShift cluster properly [96]. Upgrade process was tested twice and both times upgrade process went through flawlessly without any manual treatment after typing in "desiredVersion" details.

6 ANALYSIS

6.1 VIM comparison analysis

Before comparison was made, it was expected that at least some platform will support only hypervisor-based virtualization due to platform background and age. However as can be seen from the chapter 4.2. Basically, every major platform supported containerization one way or another. This indirectly implies that newer container-based virtualization is now truly mature enough for broad general use, because even the bigger players on the market have adopted the technology into their products.

Other interesting fact that can be deduced from the compared platforms is that every platform had support for Kubernetes. This implies that Kubernetes currently holds significant portion of the container orchestration market and because of that may be even considered to be dominant player on its own market.

6.1.1 OpenShift compared to other VIMs

Based on the table accessible in the **APPENDIX A** and its textual representation in the section 4.7. OpenShift performed well compared to other VIMs. Main downsides of the OpenShift were limited support for language bindings and no support for firmware information queries. OpenShift does have some support for language bindings through the Kubernetes client API as stated in the comparison table **APPENDIX A** notes #1, but it does not have its completely own language bindings for every possible functionality that can be found from the OpenShift. In practice this is not likely a problem, because Kubernetes has comprehensive list of APIs that should be enough for application, but there could be some advantages from having own OpenShift API bindings especially if OpenShift specific APIs are expanded in the future.

Support for firmware querying was generally in bad shape and OpenShift did not make exception here. Practically vSphere was only platform that had any kind of support for querying firmware information. So OpenShift does not perform especially badly here, but firmware information would be good to be easily available so checking need for a firmware upgrades would be easier.

It is hard to see directly from the comparison table, but possibly one of the greatest advantages of the OpenShift compared to other platforms is its highly automated upgrade process described in the section 5.2.4. Based on the sources referenced in the comparison table, OpenShift was only platform that offered one click style totally automated upgrade experience. Other platforms like vSphere and SUSE CaaS Platform have their own partly automated upgrade experiences, but they are still not as simple as OpenShift.

6.1.2 VIM strengths and use cases

Based on the section 4.7. and table seen in the **APPENDIX A** OpenShift is most fitting for the applications that can be containerized and should be run in an easily upgradable environment, due to its great Kubernetes support and upgrade capabilities. OpenStack is most suitable for the situations where hypervisor-based virtualization is necessary, and Python is preferred language for scripting deployments, due to its hypervisor focus and official Python SDK. vSphere is best for the situations where integration to BMCs is needed and firmware information is kept more important as its only platform to have any support for those. SUSE CaaS Platform should be used if absolutely compliance with Kubernetes is desirable and there is a need for broad language binding support, due to its strict Kubernetes compatibility.

6.2 Practical implementation analysis

Practical implementation of the client for OpenShift management APIs is quite straightforward. OpenShift management APIs do follow industry standard REST architecture, and they have sufficiently broad list of APIs available. Responses provided by the API endpoints are simple and do contain sufficiently detailed data. Although writing a client for OpenShift should be quite straight forward, there are still certain aspects that should be considered before writing the client for the provided APIs.

6.2.1 Subdomains and service discovery

As stated in the section 5.2, for example the APIs provided by the Prometheus framework are not directly available under the main domain name, instead those APIs must be accessed via subdomain. The subdomain can be discovered via main API endpoint using specialized discovery APIs. Practically, this leaves implementer of the client program two alternative ways to proceed. The first option is to hardcode subdomains into the client. The second option is to write a handler to the client that can understand responses received from the discovery APIs. These two alternatives can be converted to a question, weather it is better to make a client, that has less code, but may break because endpoint address changes, or a client that is slightly more complex, but can work even after possible subdomain name change.

In addition to simpler implementation, the hardcoded alternative does not have overhead caused by the additional discovery HTTP request. On the other hand, performance penalty of the extra HTTP request can be almost completely mitigated by the HTTP caching, but this would grow the code base slightly. In addition to possibly better longevity, the other discovery-based alternative could possibly make integration testing slightly easier. This comes from the fact, that via discovery endpoint it would be possible to inject any kind of API address to the client, which could make testing environment simpler by dropping the requirement for the manually designed subdomain handling. There is no definitive answer to this question and the correct option comes down to specific client requirements and testing environment.

6.2.2 API authorization

OpenShift has sophisticated authorization system which is based on the Role-Based Access Control (RBAC) [97]. Client must somehow take this into consideration and make sure, that it can at least work correctly when it tries to access a resource, that it does not have necessary permissions. There is at least three alternative ways to proceed:

1. Resources are accessed in a best effort manner, which means that access checking is not done beforehand, and HTTP resource is skipped, if there is no access.
2. Access is checked beforehand via provided access APIs and only resources that client is authored to access are checked.

3. Access is checked beforehand and client will not work if unauthorized resource is detected.

The first option could be fit well for polling style client. For example, if the CPU usage data can be accessed, and the alarm related information cannot be, then only CPU usage can be consumed and the information about the alarms is left blank and the collected data is added to the GUI. Possibly GUI could also include warning about the missing information.

The second option is similar to the first one. The main difference between the two is that the second option would add at least one extra HTTP request on the beginning of the information gathering, but on the other hand it would save the amount of requests later, if there are multiple resources that are not accessible. The disadvantage of the second option is that, it would possibly add some more complexity due to requirement to understand access APIs and there needs to be list of the APIs that will be accessed.

The third option would be slightly simpler than the second option. Advantage of the third option is that, after the access is fully checked there should be no problems while accessing the APIs. Disadvantage of the third option is that, it can be irritating to the user if the client cannot be used at all because there is one access that is missing. There is no single best option, because some access checking schemes may work better for some client architecture than others.

6.2.3 Overhead of the redundant requests

It is often necessary to make multiple API requests to the Prometheus, while retrieving certain information, like percentage of the CPU utilization. First request for the actual usage and second request for the total amount of the CPU cores. This will cause unnecessary load on the client application because it is wasteful to query total amount of the CPU cores every time while checking CPU usage due to a fact, that amount of CPU cores rarely changes. This problem can be mitigated by utilizing adaptive HTTP caching inside the client application. Adaptive HTTP cache should cache all the responses that will stay persistent for a long period of time and always query other requests immediately. This way amount of the server facing HTTP request should reduce significantly but data would still stay adequately up to date.

6.2.4 API documentation availability

As stated in the section 5.1. REST API documentation of the OpenShift is does not always contain every detail. The reason for the lack of the information is likely related to the fact that OpenShift heavily utilizes Kubernetes APIs which are already well documented. In addition, OpenShift uses microservice architecture as described in the section 3. that contains services like Prometheus that have their own documentation. Alternatively lack of documentation is related to the fact that CLI and GUI are main methods for accessing the cluster's configuration as can be deduced from the OpenShift manual by comparing the amount of available documentation for different access methods.

6.3 OpenShift client maintenance analysis

When developing client for an OpenShift cluster, the maintenance perspective should be taken into consideration to save the resources in the future. Software maintenance is often considered to be expensive. Because of that fact, maintenance perspective should not be forgotten. The price of the software maintenance amongst other things is caused by the ripple effect which means that if the part of the software is changed it may cause undesired changes to the other part of the software. [98]

6.3.1 Parameters or the response of the APIs are altered

There is no guarantee that APIs would not be changed in the future. Sometimes API changes are necessary to support new features, due to changes in the data and computation environment, or to make APIs easier to maintain and comprehend [99]. However large unexpected APIs changes are not likely to occur in the OpenShift platform, because every API contains clear version indicator in its path and a beta tag marker, if the API is not yet considered stable [89]. Basically, this implies that, if the API is altered, the new version will likely get its own version number which makes it possible for multiple versions of the API to coexist if necessary. It has been stated that breaking changes to the APIs should be avoided if possible and should be announced clearly to the public in case they are made. [100] Based on the version numbers in the OpenShift’s REST API, it can be seen that most of the APIs are currently on version one, which implies that OpenShift is following the good practices by making breaking changes only rarely [89].

Exception to the OpenShift’s normal versioning rule is included Oauth authorization API that does not have any versioning in its path. Oauth API is not at the time of writing very well documented in OpenShift’s own documentation, which may at first glance cause some suspicion towards the future of the API [37]. However Oauth 2 requests and responses are well defined and documented in its own specification, which indicates that API can be considered to be stable [35].

Some of the requests like Prometheus Metric queries use URL encoded parameters as can be seen from the **APPENDIX B.** Prometheus query format is documented and mandated by the Prometheus team, but all of it values that can be issued as a parameter are not, like for example the names of the metrics which are mandated by the Red Hat [94]. At the time of writing due to lack of the available online documentation Prometheus metric names needed to be manually investigated either using OpenShift’s GUI or CLI [91]. Regardless of the online documentation’s availability, all the available metrics were still easily available in the OpenShift’s web GUI.

Even if the information of the current metrics was quite easily available, there was no indication about the longevity of the different available metrics. This creates a risk that metric could disappear or change name in any future version. The risk should be taken into consideration, while implementing the client. Possible problems caused by the changes can be partly mitigated by designing a client that utilizes metric information beforehand in a flexible fashion. Flexible construction should make it easier to support different versions of the metrics in the future if needed.

Main construction of the metrics responses is unlikely to change significantly in the future for the same reasons as stated in the first paragraph. However small changes are possible to the sections like Prometheus’ “metric”-section seen in the **Picture 6** that may have different amount of fields in the future due to variable nature of the metric-object described in the specification [94]. To avoid breakages due to small changes, usage of the flexible JSON de serializer should be used during implementation, instead of a static one. As described in the section 5.2.3. Prometheus “metric”-section’s fields can be used for combining data, but as stated above this can be troublesome if some of the fields are changed or removed in the future. This problem cannot be mitigated completely, because the metainformation is crucial in some cases, however usage of the meta information can possibly be reduced by using for example more appropriate metrics that do not need any manual combining.

6.3.2 Subdomain of the API endpoint changes

Changes to the subdomains of the Prometheus and Alertmanager are certainly possible based on the fact that OpenShift offers customized APIs for discovering those services [89]. Change of the subdomain names should not be a big deal if the discovery APIs are properly used because then names are updated automatically. Similar discovery rules applies to Oauth server which can be discovered via “.well-known” host path [101].

6.3.3 OpenShift components might be changed

Prometheus and Oauth are not in the core of the OpenShift and thus can be replaced in theory. However at least replacement of the Prometheus is unlikely in the near future, because it was just introduced 4 major versions ago [91], [102]. The change was likely made, because the maintenance for the old monitoring software called Heapster was ended [103]. Alertmanager is part of the Prometheus so it has same advantage as its parent software.

Oauth authentication stack has been used quite a long time now in the OpenShift, at least from the version 3.0 [104]. There are no significant sources that would neither approve nor deny that Oauth would be changed in the future to something else. Even if the Oauth itself would not be ditched from the OpenShift anytime soon, there is a possibility that version of the Oauth protocol will be changed. Oauth 2 protocol has been around since year 2012 and currently new version of the protocol is in the works [35], [105]. There is no information available yet about when the Oauth 3 will be released [105]. However, considering the age of the Oauth 2 protocol, Oauth 3 protocol might be released in the next couple of years.

6.4 OpenShift analysis conclusions

Based on the prior analysis there are no reasons to believe that OpenShift would not be suitable as a VIM managed by a fleet manager. OpenShift has similar level of functionality compared to other VIMs candidates and OpenShift provided APIs are implementable in practice. In addition, there are no especially bad problems in sight considering the future compatibility.

6.5 Future research

This thesis was focused on the fleet management context of the OpenShift and gave some information about, how the OpenShift performed compared to other VIMs from the management perspective. However more research should be made before conclusion about OpenShift fit for network infrastructure purposes should be made. Especially following subject should be researched.

One important subject could be to check OpenShift's network performance. For example, how well does OpenShift perform on the concrete networking infrastructure applications compared to other VIM? More specifically it should be researched, what kind of network characteristics OpenShift can produce. Amongst other things, what is maximum throughput of the OpenShift, how small is the OpenShift latency and overall, how reliable OpenShift network is?

Another interesting subject could be best installation setup for the OpenShift cluster. More specifically what is the most efficient cluster size and how different nodes should be physically organized? This question is interesting, because in theory small cluster would have less networking overhead in the management network, but on the other hand larger installation could have more efficiently specialized nodes.

7 CONCLUSION

Container-based virtualization is now fully ready for the general use due to its current high adoption rates. Container-based OpenShift shares similar levels of monitoring and management features compared to similar competing platforms. Different platforms have different advantages and OpenShift is especially good dealing with upgrades. There are no reasons why OpenShift could not be used as a fleet manager managed VIM. However further research is required concerning OpenShift’s real world application performance compared to other platforms to verify NFV performance.

OpenShift installation preparation has multiple steps, but the installation process itself is highly automated. OpenShift has many REST based APIs that can be used for monitoring and management. Monitoring metrics is provided by Prometheus server, which provides plenty of metrics that can be used for monitoring different parts of the cluster. From the API perspective OpenShift should be integrable to the VIM fleet manager. OpenShift APIs can considered to be generally stable and should not cause significant problems on the maintenance phase.

REFERENCES

[1] T. Barnett, S. Jain, U. Andra, and T. Khurana, ‘Cisco Visual Networking Index (VNI)’. Dec. 2018.

[2] ITU, ‘IMT traffic estimates for years 2020 to 2030’. Electronic Publication Geneva, Jul. 2015, [Online]. Available: https://www.itu.int/pub/R-REP-M.2370-2015.

[3] Cisco, ‘Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper’. Cisco, Mar. 09, 2020, Accessed: Mar. 25, 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[4] H. Farhady, H. Lee, and A. Nakao, ‘Software-Defined Networking: A survey’, *Computer Networks*, vol. 81, pp. 79–95, Apr. 2015, doi: 10.1016/j.comnet.2015.02.014.

[5] D. Gedia and L. Perigo, ‘Performance Evaluation of SDN-VNF in Virtual Machine and Container’, in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2018, pp. 1–7, doi: 10.1109/NFV-SDN.2018.8725805.

[6] J. Pan and J. McElhannon, ‘Future Edge Cloud and Edge Computing for Internet of Things Applications’, *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 1–2, Feb. 2018, doi: 10.1109/JIOT.2017.2767608.

[7] M. Eder, ‘Hypervisor-vs . Container-based Virtualization’, presented at the Seminar Future Internet WS2015/16, Jul. 2016, doi: 10.2313/NET-2016-07-1_01.

[8] A. R. Manu, J. K. Patel, S. Akhtar, V. K. Agrawal, and K. N. B. Subramanya Murthy, ‘A study, analysis and deep dive on cloud PAAS security in terms of Docker container security’, in *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, Mar. 2016, pp. 1–13, doi: 10.1109/ICCPCT.2016.7530284.

[9] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, ‘Performance Overhead Comparison between Hypervisor and Container Based Virtualization’, in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, Mar. 2017, pp. 955–962, doi: 10.1109/AINA.2017.79.

[10] S. Sultan, I. Ahmad, and T. Dimitriou, ‘Container Security: Issues, Challenges, and the Road Ahead’, *IEEE Access*, vol. 7, pp. 52976–52996, May 2019, doi: 10.1109/ACCESS.2019.2911732.

[11] G. Shipley and G. Dumpleton, *OpenShift for Developers: A Guide for Impatient Beginners*. O’Reilly Media, Inc., 2016.

[12] Opensource.com, ‘What is Docker?’, *Opensource.com*. https://opensource.com/resources/what-docker (accessed Mar. 26, 2020).

[13] Docker, ‘Why Docker? | Docker’. https://www.docker.com/why-docker (accessed Mar. 26, 2020).

[14] D. Merkel, ‘Docker: lightweight linux containers for consistent development and deployment’, *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[15] Docker, ‘Docker frequently asked questions (FAQ)’, *Docker Documentation*, Jul. 03, 2020. https://docs.docker.com/engine/faq/ (accessed Jul. 03, 2020).

[16] C. Anderson, ‘Docker [Software engineering]’, *IEEE Software*, vol. 32, no. 3, pp. 102–105, May 2015, doi: 10.1109/MS.2015.62.

[17] Cloud Native Computing Foundation, ‘cri-o’. https://cri-o.io/ (accessed Jun. 10, 2020).

[18] Linux Foundation, ‘Open Container Initiative - Open Container Initiative’, 2020. https://opencontainers.org/ (accessed Jun. 10, 2020).

[19] S. Hykes, ‘Introducing runC: a lightweight universal container runtime’, *Docker Blog*, Jun. 22, 2015. https://www.docker.com/blog/runc/ (accessed Jun. 10, 2020).

[20] Docker, ‘Industry Leaders Unite to Create Project for Open Container Standards | Docker’, Jun. 22, 2015. https://www.docker.com/docker-news-and-press/industry-leaders-unite-create-project-open-container-standards (accessed Jun. 10, 2020).

[21] cri-o contributors, ‘cri-o/cri-o’, *GitHub*. https://github.com/cri-o/cri-o (accessed Jun. 10, 2020).

[22] S. Walli, ‘Demystifying the Open Container Initiative (OCI) Specifications’, *Docker Blog*, Jul. 19, 2017. https://www.docker.com/blog/demystifying-open-container-initiative-oci-specifications/ (accessed Jun. 10, 2020).

[23] The Kubernetes Authors, ‘Production-Grade Container Orchestration’. https://kubernetes.io/ (accessed Mar. 26, 2020).

[24] D. Vohra, ‘Hello Kubernetes’, in *Kubernetes Microservices with Docker*, D. Vohra, Ed. Berkeley, CA: Apress, 2016, pp. 41–42.

[25] R. Mocevicius, *CoreOS Essentials*, 1st ed., 132 vols. Livery Place, Birmingham: Packt Publishing Ltd, 2015.

[26] P. Cormier, ‘Red Hat to Acquire CoreOS, Expanding its Kubernetes and Containers Leadership’, Jan. 30, 2018. https://www.redhat.com/en/about/press-releases/red-hat-acquire-coreos-expanding-its-kubernetes-and-containers-leadership (accessed Jun. 18, 2020).

[27] Red Hat, ‘What is OpenShift - Red Hat OpenShift’. https://www.openshift.com/learn/what-is-openshift (accessed Mar. 26, 2020).

[28] Red Hat, ‘What is PaaS?’ https://www.redhat.com/en/topics/cloud-computing/what-is-paas (accessed Jun. 21, 2020).

[29] Red Hat, ‘Product architecture, OpenShift Container Platform 4.3’, 2020. https://docs.openshift.com/container-platform/4.3/architecture/architecture.html (accessed Mar. 26, 2020).

[30] Red Hat, ‘OKD - The Community Distribution of Kubernetes that powers Red Hat OpenShift.’ https://www.okd.io/#v3 (accessed Aug. 09, 2020).

[31] B. Harrington, ‘OpenShift and Kubernetes: What’s the difference?’, *Red Hat Blog*, Feb. 05, 2019. https://www.redhat.com/en/blog/openshift-and-kubernetes-whats-difference (accessed Aug. 09, 2020).

[32] S. McCarty, ‘Red Hat OpenShift Container Platform 4 now defaults to CRI-O as underlying container engine’, *Red Hat Blog*, Jun. 04, 2019. https://www.redhat.com/en/blog/red-hat-openshift-container-platform-4-now-defaults-cri-o-underlying-container-engine (accessed Jun. 10, 2020).

[33] W. Caban, *Architecting and Operating OpenShift Clusters - OpenShift for Infrastructure and Operations Teams | William Caban | Apress*, 1st ed., vol. 2019, XVII, 286 vols. Barkley. CA: Apress.

[34] S. Herrod, ‘VMware: The Console Blog: Towards Virtualized Networking for the Cloud’, *The Console Blog*, Aug. 30, 2011. https://web.archive.org/web/20111216135432/blogs.vmware.com/console/2011/08/towards-virtualized-networking-for-the-cloud.html (accessed Jun. 18, 2020).

[35] D. Hardt, ‘The OAuth 2.0 Authorization Framework’, Oct. 2012. https://tools.ietf.org/html/rfc6749#section-4.1.1 (accessed Jun. 30, 2020).

[36] J. Richer and A. Sanso, *OAuth 2 in Action*, 1st edition. Shelter Island, NY: Manning Publications, 2017.

[37] Red Hat, 'Understanding authentication OpenShift 4.4 authentication'. https://docs.openshift.com/container-platform/4.4/authentication/understanding-authentication.html (accessed Jun. 28, 2020).

[38] B. Brazil, *Prometheus: Up & Running: Infrastructure and Application Performance Monitoring*, 1 edition. Sebastopol, CA: O'Reilly Media, 2018.

[39] Red Hat, 'OpenShift Prometheus Cluster Monitoring 4.4'. https://docs.openshift.com/container-platform/4.4/monitoring/cluster_monitoring/about-cluster-monitoring.html (accessed Jul. 04, 2020).

[40] Red Hat, 'Installation and update, Architecture'. https://docs.openshift.com/container-platform/4.4/architecture/architecture-installation.html#architecture-installation (accessed Jun. 28, 2020).

[41] ETSI ISM, 'Setup of Virtual Infrastructure Managers (VIMs)', 2020. https://osm.etsi.org/docs/user-guide/04-vim-setup.html (accessed Jun. 22, 2020).

[42] P. L. Ventre *et al.*, 'Performance evaluation and tuning of Virtual Infrastructure Managers for (Micro) Virtual Network Functions', in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2016, pp. 141–147, doi: 10.1109/NFV-SDN.2016.7919489.

[43] S. Dahmen-Lhuissier, 'ETSI - Standards, mission, vision, direct member participation', *ETSI*. https://www.etsi.org/about (accessed Jul. 05, 2020).

[44] J. Pelkmans, 'The GSM standard: explaining a success story', *Journal of European Public Policy*, vol. 8, no. 3, pp. 432–453, Jan. 2001, doi: 10.1080/13501760110056059.

[45] S. Dahmen-Lhuissier, 'ETSI - Open Source Mano | Open Source Solutions | Mano NFV', *ETSI*. https://www.etsi.org/technologies/open-source-mano (accessed Jul. 05, 2020).

[46] SUSE, 'CaaS Platform: Container Management & Kubernetes Technology | SUSE'. https://www.suse.com/products/caas-platform/ (accessed Jul. 22, 2020).

[47] SUSE, 'SUSE CaaS Platform 4.2.1: Chapter 6. Monitoring (Administration Guide)'. https://documentation.suse.com/suse-caasp/4.2/html/caasp-admin/_monitoring.html (accessed Jul. 22, 2020).

[48] Amazon, ‘AWS Outposts overview’, *Amazon Web Services, Inc.* https://aws.amazon.com/outposts/ (accessed Jul. 18, 2020).

[49] M. Di Mauro *et al.*, ‘Availability evaluation of the virtualized infrastructure manager in Network Function Virtualization environments’, *Risk, Reliability and Safety: Innovating Theory and Practice*, pp. 2591–2596, 2016.

[50] K. Colbert, ‘How to Get vSphere with Kubernetes’, *VMware vSphere Blog*, Apr. 02, 2020. https://blogs.vmware.com/vsphere/2020/04/how-to-get-vsphere-with-kubernetes.html (accessed Jul. 05, 2020).

[51] OpenStack, ‘What is OpenStack Zun?’, *OpenStack*. https://www.openstack.org/software/ (accessed Jul. 04, 2020).

[52] Red Hat, ‘About container-native virtualization 4.4’. https://docs.openshift.com/container-platform/4.4/cnv/cnv-about-cnv.html (accessed Jul. 04, 2020).

[53] ‘List of Best OpenShift Alternatives & Competitors 2020’, *TrustRadius*. https://www.trustradius.com/products/openshift/competitors (accessed Jun. 21, 2020).

[54] ‘OpenStack’, *OpenStack*. https://www.openstack.org/ (accessed Jun. 21, 2020).

[55] K. Jackson, *Openstack Cloud Computing Cookbook*. Packt Publishing Ltd, 2012.

[56] OpenStack, ‘What is OpenStack?’, *OpenStack*. https://www.openstack.org/software/ (accessed Jun. 21, 2020).

[57] OpenStack, ‘Commercial Distributions and Hardware Appliances of OpenStack Private Cloud’, *OpenStack*. https://www.openstack.org/marketplace/distros/ (accessed Jul. 23, 2020).

[58] OpenStack, ‘What is OpenStack Magnum?’, *OpenStack*. https://www.openstack.org/software/ (accessed Jul. 23, 2020).

[59] T. Sechkova, M. Paolino, and D. Raho, ‘Virtualized Infrastructure Managers for Edge Computing: OpenVIM and OpenStack Comparison’, in *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, Jun. 2018, pp. 1–6, doi: 10.1109/BMSB.2018.8436858.

[60] A. Mayer, ‘VMware ESXi vs vSphere vs vCenter: A Comparison’, *Official NAKIVO Blog*, Feb. 19, 2020. https://www.nakivo.com/blog/vmware-esxi-vs-vsphere-vs-vcenter-key-differences/ (accessed Jul. 04, 2020).

[61] 'What is vSphere 7 Server Virtualization Software VMware', *VMware*, 2020. https://www.vmware.com/products/vsphere.html (accessed Aug. 09, 2020).

[62] O. Krieger, P. McGachey, and A. Kanevsky, 'Enabling a marketplace of clouds: VMware's vCloud director', *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 4, pp. 103–114, Dec. 2010, doi: 10.1145/1899928.1899942.

[63] D. Davis, 'VMware vCenter vs. vCloud Director - VMware Cloud Provider Blog', *VMware Cloud Provider Blog*, Feb. 14, 2013. https://blogs.vmware.com/cloudprovider/2013/02/vmware-vcenter-vs-vcloud-director.html (accessed Jul. 05, 2020).

[64] S. Gallagher, *VMware Private Cloud Computing with vCloud Director*. John Wiley & Sons, 2013.

[65] VMware, 'vCenter Server: Centralized visibility, proactive management and extensibility for VMware vSphere from a single console', *VMware*. https://www.vmware.com/products/vcenter-server.html (accessed Jul. 05, 2020).

[66] M. Brown and H. Cartwright, *VMware vSphere 6.7 Data Center Design Cookbook: Over 100 practical recipes to help you design a powerful virtual infrastructure based on vSphere 6.7, 3rd Edition*, 3 edition. Packt Publishing, 2019.

[67] SUSE, 'Release Notes | SUSE CaaS Platform 1.0', Sep. 22, 2017. https://www.suse.com/releasenotes/x86_64/SUSE-CAASP/1.0/ (accessed Jul. 22, 2020).

[68] SUSE, 'What is SUSE CaaS Platform? What's all the buzz about?', *SUSE Blog*, May 02, 2017. https://www.suse.com/c/suse-caas-platform-whats-buzz/ (accessed Jul. 22, 2020).

[69] *hashicorp/nomad*. HashiCorp, 2020.

[70] Eclipse, 'What is Eclipse fog05? · Eclipse fog05'. https://fog05.io/docs/overview/ (accessed Jul. 05, 2020).

[71] Eclipse, 'eclipse fog05 repositories', *GitHub*. https://github.com/eclipse-fog05 (accessed Jul. 05, 2020).

[72] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, 'Fog computing and its role in the internet of things', in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, Helsinki, Finland, Aug. 2012, pp. 13–16, doi: 10.1145/2342509.2342513.

[73] OSM, ‘OpenVIM README.rst’. https://osm.etsi.org/gitweb/?p=osm/openvim.git;a=blob;f=README.rst;h=00ea01aaaadc4ab46ebd40178347dff3b3f8ce30;hb=HEAD (accessed Jul. 05, 2020).

[74] OSM, ‘EPA and SDN assist - OSM Public Wiki’. https://osm.etsi.org/wikipub/index.php/EPA_and_SDN_assist (accessed Jul. 05, 2020).

[75] Red Hat, ‘Is there a OpenStack API documentation from Red Hat ?’, *Red Hat Customer Portal*, Mar. 08, 2019. https://access.redhat.com/solutions/2680301 (accessed Jul. 22, 2020).

[76] fog05, ‘Releases · eclipse-fog05/fog05’, *GitHub*, Jun. 22, 2020. /eclipse-fog05/fog05/releases (accessed Jul. 22, 2020).

[77] fog05, ‘fog05 - GoDoc’. https://godoc.org/github.com/eclipse-fog05/api-go/fog05 (accessed Jul. 22, 2020).

[78] G. Rigazzi, J.-P. Kainulainen, C. Turyagyenda, A. Mourad, and J. Ahn, ‘An Edge and Fog Computing Platform for Effective Deployment of 360 Video Applications’, in *2019 IEEE Wireless Communications and Networking Conference Workshop (WCNCW)*, Apr. 2019, pp. 1–6, doi: 10.1109/WCNCW.2019.8902860.

[79] ETSI, ‘NFV Release 2 Description’. Feb. 2020, [Online]. Available: https://docbox.etsi.org/ISG/NFV/Open/Other/ReleaseDocumentation/NFV(20)000015_NFV_Release_2_Description_v1_10_0.pdf.

[80] VMware, ‘Authentication and Authorization for ESXi and vCenter Server - vSphere Web Services SDK Programming Guide - VMware’, *VMware {code}*. https://code.vmware.com/docs/1682/vsphere-web-services-sdk-programming-guide/doc/PG_Authenticate_Authorize.8.3.html (accessed Jul. 22, 2020).

[81] OpenStack, ‘OpenStack Docs: Upgrades’, Aug. 23, 2019. https://docs.openstack.org/operations-guide/ops-upgrades.html (accessed Jul. 22, 2020).

[82] VMware, ‘Understanding Update Manager’, May 31, 2019. https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.update_manager.doc/GUID-EF6BEE4C-4583-4A8C-81B9-5B074CA2E272.html (accessed Jul. 22, 2020).

[83] SUSE, ‘SUSE CaaS Platform 4.2.1: Administration Guide, Update’, Jun. 10, 2020. https://documentation.suse.com/suse-caasp/4.2/single-html/caasp-admin/index.html (accessed Jul. 22, 2020).

[84] C. Severance, ‘Discovering JavaScript Object Notation’, *Computer*, vol. 45, no. 4, pp. 6–8, Apr. 2012, doi: 10.1109/MC.2012.132.

[85] Red Hat, ‘Installing OpenShift 4.4 on a bare metal’. https://docs.openshift.com/container-platform/4.4/installing/installing_bare_metal/installing-bare-metal.html#installing-bare-metal (accessed May 30, 2020).

[86] P. V. Mockapetris, ‘Domain names - concepts and facilities’. https://tools.ietf.org/html/rfc1034 (accessed Jun. 28, 2020).

[87] T. Barnes-Lee, ‘RFC 1945: Hypertext Transfer Protocol -- HTTP/1.0’, May 1996. https://www.hjp.at/doc/rfc/rfc1945.html (accessed Jun. 28, 2020).

[88] International Organization for Standardization, ‘ISO 9660:1988(en), Information processing — Volume and file structure of CD-ROM for information interchange’, Apr. 1998. https://www.iso.org/obp/ui/#iso:std:iso:9660:ed-1:v1:en (accessed Jul. 23, 2020).

[89] Red Hat, ‘OpenShift REST API endpoints’. https://docs.openshift.com/online/starter/rest_api/index.html (accessed May 31, 2020).

[90] D. Hardt and M. Jones, ‘The OAuth 2.0 Authorization Framework: Bearer Token Usage’. https://tools.ietf.org/html/rfc6750#section-2 (accessed Jun. 28, 2020).

[91] Red Hat, ‘OpenShift Prometheus Cluster Monitoring 3.11’. https://docs.openshift.com/container-platform/3.11/install_config/prometheus_cluster_monitoring.html (accessed May 31, 2020).

[92] Prometheus Authors, ‘Prometheus overview’. https://prometheus.io/docs/introduction/overview/ (accessed May 31, 2020).

[93] Red Hat, ‘Examining cluster metrics OpenShift 4.4’. https://docs.openshift.com/container-platform/4.4/monitoring/cluster_monitoring/examining-cluster-metrics.html (accessed Jun. 28, 2020).

[94] Prometheus Authors, ‘HTTP API | Prometheus’, 2020. https://prometheus.io/docs/prometheus/latest/querying/api/#instant-queries (accessed Jun. 28, 2020).

[95] Prometheus Authors, *Prometheus alermanager*. .

[96] Red Hat, ‘Updating a cluster within a minor version from the web console’. https://docs.openshift.com/container-platform/4.4/updating/updating-cluster.html (accessed Jul. 05, 2020).

[97] Red Hat, ‘Using RBAC to define and apply permissions 4.4’. https://docs.openshift.com/container-platform/4.4/authentication/using-rbac.html (accessed Jul. 08, 2020).

[98] S. S. Yau, J. S. Collofello, and T. MacGregor, ‘Ripple effect analysis of software maintenance’, in *The IEEE Computer Society’s Second International Computer Software and Applications Conference, 1978. COMPSAC ’78.*, Nov. 1978, pp. 60–65, doi: 10.1109/CMPSAC.1978.810308.

[99] N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil, and W.-G. Tan, ‘Types of software evolution and software maintenance’, *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 13, no. 1, pp. 3–30, 2001, doi: 10.1002/smr.220.

[100] B. De, ‘API Version Management’, in *API Management: An Architect’s Guide to Developing and Managing APIs for Your Organization*, B. De, Ed. Berkeley, CA: Apress, 2017, pp. 105–110.

[101] M. Nottingham and E. Hammer-Lahav, ‘Defining Well-Known Uniform Resource Identifiers (URIs)’. https://tools.ietf.org/html/rfc5785 (accessed Jul. 01, 2020).

[102] Red Hat, ‘Enabling Cluster Metrics 3.10’. https://docs.openshift.com/container-platform/3.10/install_config/cluster_metrics.html (accessed Jun. 30, 2020).

[103] *kubernetes-retired/heapster*. Kubernetes Retired, 2020.

[104] Red Hat, ‘Authentication - Additional Concepts OpenShift 3.0’. https://docs.openshift.com/enterprise/3.0/architecture/additional_concepts/authentication.html (accessed Jul. 01, 2020).

[105] A. Parecki, ‘OAuth 3’. https://oauth.net/3/ (accessed Jul. 01, 2020).

APPENDICES

APPENDIX A. VIM comparison table

Marker description table:

Support level and type:	**Marked as score**	**Marker:**
Support is said in the documentation or in the other sources	yes	✓^{1}
Not directly supported by the core but tightly integrated	yes	✓^{2}
Not officially supported but is still fully accessible or API is supported but provides only partial integration or information.	no	✗^{3}
Not officially supported but is still partly accessible	no	✗^{4}
Not supported (no information available and could not be verified empirically)	no	✗^{5}
Inaccessible documentation	no	?^{6}

VIM feature comparison table:

	OpenShift 4.4	OpenStack 17.0 (Victoria)	vSphere 7.0	SUSE CaaS Platform 4.2.1
Access methods:	3/3	3/3	2/3	2/3
-Web GUI	✓^{1} [1]	✓^{1} [2]	✓^{1} [3]	✗^{3} [4] [5]
-CLI	✓^{1} [6]	✓^{1} [7]	✓^{1} [#17]	✓^{1} [#28]
-REST API	✓^{1} [8]	✓^{1} [9]	✗^{3} [#18]	✓^{1} [#29]
Language bindings:	0/5	1/5	0/5	5/5
-Golang	✗^{4} [#1]	✗^{3} [10]	✗^{3} [#19]	✓^{1} [#30]
-Java	✗^{4} [#1]	✗^{3} [10]	✗^{3} [#19]	✓^{1} [#30]
-JavaScript	✗^{4} [#1]	✗^{3} [10]	✗^{3} [#19]	✓^{1} [#30]
-Python	✗^{4} [#1]	✓^{1} [10]	✗^{3} [#19]	✓^{1} [#30]
-Other	✗^{4} [#1]	✗^{3} [10]	✗^{3} [#19]	✓^{1} [#30]
Authentication:	3/3	3/3	1/3	3/3

(continues)

APPENDIX A. (continuation)

-Username / password	✓^{1} [11]	✓^{1} [12]	✓^{1} [#20]	✓^{1} [13]
-X.509 like Certificate	✓^{1} [11]	✓^{1} [12]	✗^{3} [#20]	✓^{1} [13]
-Other	✓^{1} [#2]	✓^{1} [12]	✗^{3} [#20]	✓^{1} [13]
System information:	2/5	1/5	4/5	0/5
-VIM version	✓^{1} [#3]	✓^{2} [#9]	✓^{1} [14]	✗^{4} [#31]
-Operating system version	✓^{1} [#3]	✗^{5} [#10]	✓^{1} [#21]	✗^{4} [#31]
-Unified Extensible Firmware (UEFI) / Basic Input Output (BIOS) version	✗^{5}	✗^{5} [#11]	✗^{3} [#22]	✗^{5}
-Baseboard Management Controller (BMC) information	✗^{5}	✗^{5} [#12]	✓^{1} [15]	✗^{5}
-System time information	✗^{4} [#4]	✗^{5}	✓^{1} [16]	✗^{5}
Monitoring / statistics:	11/12	10/12	10/12	5/12
-CPU core, overall usage	✓^{2} [#5]	✓^{1} [#13]	✓^{1} [#23]	?^{6} [#32]
-CPU core, per node usage	✓^{2} [#5]	✓^{1} [#13]	✓^{1} [#23]	✓^{1} [17]
-CPU core, per pod/VM usage	✓^{2} [#5]	✓^{1} [#13]	✓^{1} [#23]	✓^{1} [17]
-RAM, overall usage	✓^{2} [#5]	✓^{1} [#13]	✓^{1} [#23]	?^{6} [#32]
-RAM, per node usage	✓^{2} [#5]	✓^{1} [#13]	✓^{1} [#23]	✓^{1} [17]
-RAM, per pod/VM usage	✓^{2} [#5]	✓^{1} [#13]	✓^{1} [#23]	✓^{1} [17]
-Disk space, overall usage	✓^{2} [#5]	✓^{1} [#13]	✓^{1} [#23]	?^{6} [#32]
-Disk space, per node usage	✓^{2} [#5]	✓^{1} [#13]	✓^{1} [#23]	?^{6} [#32]
-Disk space, per pod/VM usage	✗^{3} [#6]	✓^{1} [#13]	✓^{1} [#23]	?^{6} [#32]
-Overall pod/VM count	✓^{2} [#5]	✗^{4} [#14]	✗^{4} [#24]	✗^{4} [#33]
-Per machine pod/VM count	✓^{2} [#5]	✗^{4} [#14]	✗^{4} [#24]	✗^{4} [#33]
-Historical data available	✓^{2} [#7]	✓^{2} [#15]	✓^{1} [#25]	✓^{2} [#34]
Alerts:	4/5	5/5	4/5	4/5
-Alert UID	✗^{5} [#8]	✓^{1} [#16]	✗^{5} [18]	✗^{5} [#34]
-Alert name	✓^{2} [#8]	✓^{1} [#16]	✓^{1} [18]	✓^{2} [#34]
-Alert description	✓^{2} [#8]	✓^{1} [#16]	✓^{1} [18]	✓^{2} [#34]
-Alert timestamp	✓^{2} [#8]	✓^{1} [#16]	✓^{1} [18]	✓^{2} [#34]
-Alert severity	✓^{2} [#8]	✓^{1} [#16]	✓^{1} [18]	✓^{2} [#34]
Upgrade:	1/2	0/2	1/2	0/2
-Seamless live update available	✓^{1} [19]	✗^{5} [20]	✓^{1} [#26]	✗^{4} [#35]
-Automatic Firmware (UEFI / BIOS) upgrade available	✗^{5}	✗^{5}	✗^{3} [#27]	✗^{5}
Sum	24/35	23/35	22/35	19/35

(continues)

APPENDIX A. (continuation)

Comparison notes:

Index:	**Description:**
#1	OpenShift does not have own support for language bindings but Kubernetes does. Because OpenShift is built on top of the Kubernetes these language bindings can be partly used. In addition to mentioned languages, Kubernetes supports officially also dotnet and Haskell. [21]
#2	Access token can be retrieved via web condole and then used in the requests. [11]
#3	Empirically verified that API returns correct information. [22]
#4	Pretty good time estimate can be received by issuing Prometheus query. There is no separate API endpoint for getting time and time zone information.
#5	Empirically verified to be working. Prometheus queries checked via through OpenShift web interface. [23]
#6	There is only query for pods that compresses replica set of the pods to single value.
#7	Prometheus has range selectors which give historical information and has been empirically tested to work correctly. [24]
#8	Empirically tested to work. Alerts were gathered using APIs provided by the alert manager. [23]
#9	OpenStack does not have one consistent version number. Instead OpenStack is build using many services that have their own version numbers as can be seen from upgrade page and APIs own documentation. [25] [20]
#10	At the time of writing there is no official API in OpenStack that would tell OS information. There are APIs that are near like “Compute services” and “Hypervisors”, but they don’t provide information about operating system. Some implementations of the standard have own APIs for the version information and there has been proposal to create new official API that would provide this information [25] [26]

(continues)

APPENDIX A. (continuation)

#11	Configuration of BIOS parameters is supported in the ironic API, but retrieval of the BIOS version is not. Also, Bare Metal API works only on the nodes that act as a slave. [27]
#12	Ironic API uses BMC interfaces to provision machines but does not provide any version information. [28]
#13	Metrics provided at least by compute (nova) and metrics (ceilometer) services [29] [30]. There are multiple third-party services that can provide metrics from OpenStack installation [31].
#14	VM count is not easily available but can be extracted by manually going through the instance listing. Calculating count this way is not trivial task because due to pagination in the API. [32] [33]
#15	There are at least two projects that provide historical metrics to some degree. [34] [35]
#16	Alerting is not internal part of the OpenStack specification. However, there is at least service called aodh that provides alarms. [36] [37] [38]
#17	Common vSphere CLI package was deprecated as of latest 7.0 release. ESXCLI is now released as separate package. [39] [40]
#18	Only subset of the APIs is provided in a REST manner [41] [42]
#19	Availability depends on the API that is used. Some API SDK's have wider language support. [43]
#20	Depends on the API. vSphere Web Service has wider authentication support than REST based vSphere Automation API [44] [42]
#21	Build version (operating system version) can be determined via web GUI, so there must be an API that exposes this information. However public documentation about the API was not found [45]
#22	There are some hints that the feature would be available, but no proper documentation was found. [46]
#23	Metric information can be fetched via vSphere Web Services API Performance Manager object [44]. Also, some information may be available via vSphere

(continues)

APPENDIX A. (continuation)

	automation API [47]. In addition, web GUI provides comprehensive information listing. [48]
#24	It seems there is no official way to get virtual machine count, but there is a way to get the count using other APIs. [49]
#25	VMware provides historical data mainly via vCenter, but some historical data is also directly accessible via ESXI host. [44] [50]
#26	VMware has update manager that can do seamless live upgrades [51] [52].
#27	Based on the documentation it seems that there are some vendors that support automatic upgrades of the firmware [53].
#28	SUSE CaaS Platform uses upstream tools like kubectl and kubeadm as a CLI. In addition, SUSE provides CLI called skuba that is meant to simplify usage of the kubeadm by providing simplified wrapper. [54] [55] [56]
#29	SUSE CaaS Platform does not have any specific REST API, but because CLI is based on the kubectl, Kubernetes API should be available fully available. [57]
#30	Because SUSE CaaS Platform uses standard Kubernetes APIs, all the programming language-based clients should be supported. [21]
#31	Kubernetes does not natively provide specific version information for the whole system, but has some information like OS type and Kubernetes version itself [58].
#32	May be available via Prometheus server. No documentation available, needs empirical verification.
#33	No direct API available, but number of pods can be calculated from the list of all pods.
#34	Prometheus has some historical information and Alertmanager. In theory system should work similarly to OpenShift's monitoring stack.
#35	Cluster upgrade process is somewhat automatic, but not very seamless, because pods needs manually be scaled to 0 and distribution to the service is likely to occur on a node under upgrade [59].

(continues)

APPENDIX A. (continuation)

Feature table references

[1] Red Hat, "Accessing the web console," [Online]. Available: https://docs.openshift.com/container-platform/4.4/web_console/web-console.html#web-console-overview_web-console. [Accessed 22 7 2020].

[2] OpenStack, "OpenStack componenets Horizon," [Online]. Available: https://www.openstack.org/software/releases/ussuri/components/horizon. [Accessed 22 7 2020].

[3] VMware, "GUI Deployment of the vCenter Server Appliance," [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vcenter.install.doc/GUID-477E5C38-2AA8-4319-B2C6-41671570AD27.html. [Accessed 22 7 2020].

[4] SUSE, "Administration Guide, Stratos Web Console," [Online]. Available: https://documentation.suse.com/suse-caasp/4.2/single-html/caasp-admin/#_stratos_web_console. [Accessed 22 7 2020].

[5] SUSE, "Administration Guide, Deploy Kubernetes Dashboard as an example," [Online]. Available: https://documentation.suse.com/suse-caasp/4.2/single-html/caasp-admin/#_deploy_kubernetes_dashboard_as_an_example. [Accessed 22 7 2020].

[6] Red Hat, "Getting started with the CLI," [Online]. Available: https://docs.openshift.com/container-platform/4.4/cli_reference/openshift_cli/getting-started-cli.html. [Accessed 22 7 2020].

[7] OpenStack, "Install the OpenStack command-line clients," [Online]. Available: https://docs.openstack.org/newton/user-guide/common/cli-install-openstack-command-line-clients.html. [Accessed 22 7 2020].

[8] Red Hat, "OpenShift Container Platform 4.4 REST APIs," [Online]. Available: https://docs.openshift.com/container-platform/4.4/rest_api/index.html. [Accessed 22 7 2020].

(continues)

APPENDIX A. (continuation)

[9] OpenStack, "OpenStack API Documentation," [Online]. Available: https://docs.openstack.org/api-quick-start/. [Accessed 22 7 2020].

[10 OpenStack, "OpenStack Software Development Kits," [Online]. Available: https://wiki.openstack.org/wiki/SDKs. [Accessed 22 7 2020].

[11 Red Hat, "Understanding authentication," [Online]. Available: https://docs.openshift.com/container-platform/4.4/authentication/understanding-authentication.html. [Accessed 22 7 2020].

[12 OpenStack, "Authentication Mechanisms," [Online]. Available: https://docs.openstack.org/keystone/latest/admin/authentication-mechanisms.html. [Accessed 22 7 2020].

[13 The Kubernetes Authors, "Authenticating," [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/authentication/#x509-client-certs. [Accessed 22 7 2020].

[14 VMware, "vSphere Automation API, Get Version," [Online]. Available: https://developer.vmware.com/docs/vsphere-automation/latest/appliance/rest/appliance/system/version/get/. [Accessed 22 7 2020].

[15 VMware, "Configure IPMI or iLO Settings for vSphere DPM," [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.resmgmt.doc/GUID-D247EC2C-92C5-4B9B-9305-39099F30D3B5.html. [Accessed 22 7 2020].

[16 VMware, "vSphere Automation API, Get Time," [Online]. Available: https://developer.vmware.com/docs/vsphere-automation/latest/appliance/rest/appliance/system/time/get/. [Accessed 22 7 2020].

[17 SUSE, "Administration Guide, Resource Usage," [Online]. Available: https://documentation.suse.com/suse-caasp/4.2/single-html/caasp-admin/#_usage. [Accessed 22 7 2020].

(continues)

APPENDIX A. (continuation)

[18 VMware, "Monitoring Events, Alarms, and Automated Actions," [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.monitoring.doc/GUID-9272E3B2-6A7F-427B-994C-B15FF8CADC25.html. [Accessed 22 7 2020].

[19 Red Hat, "Updating a cluster within a minor version from the web console," [Online]. Available: https://docs.openshift.com/container-platform/4.4/updating/updating-cluster.html. [Accessed 22 7 2020].

[20 OpenStack, "OpenStack upgrades," [Online]. Available: https://docs.openstack.org/operations-guide/ops-upgrades.html. [Accessed 22 7 2020].

[21 The Kubernetes Authors, "Kubernetes Client Libraries," [Online]. Available: https://kubernetes.io/docs/reference/using-api/client-libraries/. [Accessed 22 7 2020].

[22 Red Hat, "OpenShift Container Platform 4.4 REST APIs, version," [Online]. Available: https://docs.openshift.com/container-platform/4.4/rest_api/index.html#clusterversion-v1-config-openshift-io. [Accessed 22 7 2020].

[23 Red Hat, "About cluster monitoring," [Online]. Available: https://docs.openshift.com/container-platform/4.4/monitoring/cluster_monitoring/about-cluster-monitoring.html. [Accessed 22 7 2020].

[24 P. Authors, "Querying Prometheus," [Online]. Available: https://prometheus.io/docs/prometheus/latest/querying/basics/. [Accessed 22 7 2020].

[25 OpenStack, "Compute API," [Online]. Available: https://docs.openstack.org/api-ref/compute/. [Accessed 22 7 2020].

[26 OpenStack, "SystemInfoAPI proposal," [Online]. Available: https://wiki.openstack.org/wiki/SystemInfoAPI. [Accessed 22 7 2020].

(continues)

APPENDIX A. (continuation)

[27 OpenStack, "Bare Metal API, BIOS," [Online]. Available: https://docs.openstack.org/api-ref/baremetal/?expanded=show-node-details-detail,list-all-bios-settings-by-node-detail#node-bios-nodes. [Accessed 22 7 2020].

[28 OpenStack, "Bare Metal API, drivers," [Online]. Available: https://docs.openstack.org/api-ref/baremetal/?expanded=show-node-details-detail,list-all-bios-settings-by-node-detail#drivers-drivers. [Accessed 22 7 2020].

[29 OpenStack, "Show usage statistics for hosts and instances," [Online]. Available: https://docs.openstack.org/nova/latest/admin/common/nova-show-usage-statistics-for-hosts-instances.html. [Accessed 22 7 2020].

[30 OpenStack, "Ceilometer measurements," [Online]. Available: https://docs.openstack.org/ceilometer/latest/admin/telemetry-measurements.html. [Accessed 22 7 2020].

[31 OpenStack, "Operations/Tools," [Online]. Available: https://wiki.openstack.org/wiki/Operations/Tools. [Accessed 22 7 2020].

[32 OpenStack, "Nova Server Count API Extension," [Online]. Available: https://specs.openstack.org/openstack/nova-specs/specs/juno/approved/server-count-api.html. [Accessed 22 7 2020].

[33 OpenStack, "Nova Server Count API Parameter change abandoned," [Online]. Available: https://review.opendev.org/#/c/134279/. [Accessed 22 7 2020].

[34 OpenStack, "OpenStack wiki Gnocchi," [Online]. Available: https://wiki.openstack.org/wiki/Gnocchi. [Accessed 22 7 2020].

[35 OpenStack, "OpenStack wiki Monasca," [Online]. Available: https://wiki.openstack.org/wiki/Monasca. [Accessed 22 7 2020].

[36 OpenStack, "OpenStack monitoring, resource alerting," [Online]. Available: https://docs.openstack.org/operations-guide/ops-monitoring.html#resource-alerting. [Accessed 22 7 2020].

(continues)

APPENDIX A. (continuation)

[37 OpenStack, "Aodh alarms," [Online]. Available: https://docs.openstack.org/aodh/latest/admin/telemetry-alarms.html. [Accessed 22 7 2020].

[38 OpenStack, "Aodh REST api," [Online]. Available: https://docs.openstack.org/aodh/ocata/webapi/v2.html#Alarm. [Accessed 22 7 2020].

[39 VMware, "vSphere CLI," [Online]. Available: https://code.vmware.com/web/tool/6.7/vsphere-cli. [Accessed 22 7 2020].

[40 VMware, "ESXCLI," [Online]. Available: https://code.vmware.com/web/tool/7.0/esxcli. [Accessed 22 7 2020].

[41 VMware, "VMware API references," [Online]. Available: https://developer.vmware.com/. [Accessed 22 7 2020].

[42 VMware, "vSphere Automation API," [Online]. Available: https://developer.vmware.com/docs/vsphere-automation/latest/appliance/. [Accessed 22 7 2020].

[43 VMware, "SDKs: vSphere SDK, vCenter SDK, vCloud SDK," [Online]. Available: https://code.vmware.com/sdks. [Accessed 22 7 2020].

[44 VMware, "vSphere Web Services API," [Online]. Available: https://code.vmware.com/apis/968. [Accessed 22 7 2020].

[45 VMware, "Determining the build number of VMware ESX/ESXi and VMware vCenter Server," [Online]. Available: https://kb.vmware.com/s/article/1022196. [Accessed 22 7 2020].

[46 VMware, "VMware ESX/ESXi Servers - How to Attain BIOS Version on ESX and ESXi," [Online]. Available: https://support.hpe.com/hpesc/public/docDisplay?docId=mmr_kc-0101067. [Accessed 22 7 2020].

[47 VMware, "vSphere Automation API, Get Monitoring," [Online]. Available: https://developer.vmware.com/docs/vsphere-

(continues)

automation/latest/appliance/rest/appliance/monitoring/stat_id/get/. [Accessed 22 7 2020].

[48 VMware, "vSphere Hosts monitoring," [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.monitoring.doc/GUID-C8279E94-633F-4802-B08F-4FE67E71EFB1.html. [Accessed 22 7 2020].

[49 LucD, "VMware forums, Get VM Count by Cluster," [Online]. Available: https://communities.vmware.com/thread/295657. [Accessed 22 7 2020].

[50 VMware, "vSphere Performance Data Collection," [Online]. Available: https://pubs.vmware.com/vsphere-50/index.jsp?topic=%2Fcom.vmware.wssdk.pg.doc_50%2FPG_Ch16_Performance.18.2.html. [Accessed 22 7 2020].

[51 VMware, "Understanding Update Manager," [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.update_manager.doc/GUID-EF6BEE4C-4583-4A8C-81B9-5B074CA2E272.html. [Accessed 22 7 2020].

[52 VMware, "Overview of the vSphere Upgrade Process," [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/6.5/com.vmware.vsphere.upgrade.doc/GUID-7AFB6672-0B0B-4902-B254-EE6AE81993B2.html. [Accessed 22 7 2020].

[53 VMware, "Firmware updates," [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere-lifecycle-manager.doc/GUID-34AF5B19-FC80-4915-8358-D5FCC8A8E69E.html. [Accessed 22 7 2020].

[54 SUSE, "Administration Guide, Access Control," [Online]. Available: https://documentation.suse.com/suse-caasp/4.2/single-html/caasp-admin/#_access_control. [Accessed 22 7 2020].

(continues)

APPENDIX A. (continuation)

[55 SUSE, "Administration Guide," [Online]. Available: https://documentation.suse.com/suse-caasp/4.2/single-html/caasp-admin/. [Accessed 22 7 2020].

[56 SUSE, "Skuba on SUSE CaaS Platform 4," [Online]. Available: https://www.suse.com/c/skuba-on-suse-caas-platform-4/. [Accessed 22 7 2020].

[57 The Kubernetes Authors, "Access Clusters Using the Kubernetes API," [Online]. Available: https://kubernetes.io/docs/tasks/administer-cluster/access-cluster-api/. [Accessed 22 7 2020].

[58 The Kubernetes Authors, "kubectl Cluster Info," [Online]. Available: https://kubectl.docs.kubernetes.io/pages/resource_printing/cluster_information.html. [Accessed 22 7 2020].

[59 SUSE, "Administration Guide, Cluster Updates," [Online]. Available: https://documentation.suse.com/suse-caasp/4.2/single-html/caasp-admin/#_cluster_updates. [Accessed 22 7 2020].

[60 Red Hat, "OpenShift, Understanding authentication," [Online]. Available: https://docs.openshift.com/container-platform/4.4/authentication/understanding-authentication.html. [Accessed 23 7 2020].

[61 Red Hat, "How to use kubectl and a custom X509 certificate to log into a remote OpenShift cluster," [Online]. Available: https://access.redhat.com/solutions/3681611. [Accessed 22 7 2020].

APPENDIX B. Request examples for API endpoints

Header parameter for return the type and authorization is assumed to be set in every request: *curl -H "Authorization: Bearer xxx" -H "Accept: application/json" ...*

Name:	Example curl query:
Alermanager	*curl “https:// alertmanager-main-openshift-monitoring.apps. cluster-name.basedomain:6443/api/v2/alerts/groups ?silenced=false&inhibited=false&active=true”*
Prometheus metrics	*curl “https://prometheus-k8s-openshift-monitoring.apps.cluster-name.basedomain:6443/api/v1/query?query=cluster%3Acpu_usage_cores%3Asum%5B2h%3A60m%5D”*
Version	*curl “https://api.cluster-name.basedomain:6443 /apis/config.openshift.io/v1/clusterversions/version”*