jamk.fi

Automated DNS installation and configuration

Aleksi Valkeinen

Bachelor's thesis
August 2018
Technology, communication and transport
Degree Programme in Information Technology

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Valkeinen, Aleksi	Bachelor's thesis	August 2018
		Language of publication: English
	Number of pages 97	Permission for web publication: Yes

Title of publication
Automated DNS installation and configuration

Degree programme
Information Technology

Supervisor(s)
Kokkonen Tero, Häkkinen Antti

Assigned by
Qvantel Finland Oy

Abstract

The thesis was implemented as a part of Qvantel Finland Oy's development with the purpose to create an automated solution for installing and configuring services. The study focuses on Consul, a service automated by using Ansible.

The theory includes a description of Ansible, what it is for and how it works. DNS theory, Consul theory, protocols and use case are also addressed. The theory also describes Dnsmasq and other components that were used. Consul was tested by creating an environment that could be used to model Consul use cases. In this environment, different tests could be put into practice including testing name service, registering services and how the installation actually works.

The environment was built with seven different services, i.e. Docker, Consul, Dnsmasq, Mesos, Marathon, Zookeeper and Registrator. All but Dnsmasq and Mesos would be located in Docker containers. Mesos, Marathon and Zookeeper would create a distributed system kernel, that can be used to collect all resources to act as one machine. This is used to run applications on top of Marathon. Consul would be a name service, health check and key/value storage for these services. Dnsmasq would be used for port forwarding towards Consul. Registrator would act as a service registrator that would register containers automatically to Consul.

The thesis achieved the set goals, and the development continues after the finishing of the thesis. Some of the learning, troubleshooting and development took place orally during the writing of the thesis; hence, making it less visible to the reader. Testing all components was very time consuming; however, it was a good learning experience when it comes to troubleshooting and finding issues in configurations. Even though the role of Consul is ready, it will still change a great deal before going to Qvantel's own deployments.

Keywords/tags (subjects)
Automation, Distributed systems kernel, Ansible, Consul, Dnsmasq, Apache Mesos, Apache Marathon, Apache Zookeeper, Registrator, Docker

Miscellaneous (Confidential information)

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Valkeinen, Aleksi	Opinnäytetyö, AMK	Elokuu 2018
		Julkaisun kieli Englanti
	Sivumäärä 97	Verkkojulkaisulupa myönnetty: Kyllä

Työn nimi
Automated DNS installation and configuration

Tutkinto-ohjelma
Tietotekniikan koulutusohjelma

Työn ohjaaja(t)
Tero Kokkonen, Antti Häkkinen

Toimeksiantaja(t)
Qvantel Finland Oy

Tiivistelmä

Opinnäytetyö toteutettiin osana Qvantel Finland Oy:n kehittämistyötä, jonka tarkoituksena oli kehittää automatisoitu ratkaisu ohjelmien asentamiseen ja niiden konfiguroimiseen. Työssä keskityttiin Consul - nimiseen DNS - palveluun, joka automatisoitiin käyttäen Ansiblea.

Opinnäytetyön teoreettinen osuus koostuu Ansiblen tarkoituksen ja toiminnan kuvaamisesta, DNS:n teoriasta, Consulin toiminnasta, protokollista sekä sen käyttötarkoituksesta, Dnsmasqin käyttötarkoituksesta sekä muista komponenteista, joita opinnäytetyössä on käytetty. Consulin toimintaa pyrittiin mallintamaan luomalla ympäristö, jota voitaisiin käyttää todentamaan Consulin toimivuus. Tässä ympäristössä voitaisiin todentaa nimipalvelun toiminta eri ohjelmien välillä, ohjelmien rekisteröiminen sekä miten ohjelmien asennus käytännössä toimii.

Ympäristö rakentui seitsemästä eri palvelusta; Dockerista, Consulista, Dnsmasqista, Mesoksesta, Marathonista, Zookeeperista sekä Registratorista. Kaikki paitsi Dnsmasq sekä Mesos tulisivat toimimaan Docker - konteissa. Mesos, Marathon sekä Zookeeper loisivat yleisen järjestelmäkernelin, jonka resursseja voitaisiin käyttää luomaan applikaatioita Marathonin päälle. Näiden palveluiden nimipalveluna, ylläpitona sekä keskitettynä tallennuskeskuksena tulisi toimimaan Consul. Dnsmasqia käytetään porttien edelleen lähetykseen Consulille. Registrator toimii rekisteröijänä ympäristössä ja rekisteröi konteissa pyörivät palvelut automaattisesti Consuliin.

Opinnäytetyössä päästiin haluttuihin tavoitteisiin, vaikka kehitystyö jatkuu opinnäytetyön jälkeenkin. Vaikkakin Consulin rooli valmistui ja on toimiva ratkaisu, tulee sen ulkomuoto ja sisältö vielä muuttumaan, sen liikkuessa eteenpäin Qvantelin omiin asennuksiin.

Avainsanat (asiasanat)
Automaatio, Distributed systems kernel, Ansible, Consul, Dnsmasq, Apache Mesos, Apache Marathon, Apache Zookeeper, Registrator, Docker

Muut tiedot

Contents

Acronyms **7**

1 Introduction **8**

2 Research methodology and requirements **9**

2.1 Method 9

2.2 Requirements and goals set by Qvantel 9

3 Why automate **10**

3.1 Easy to install – Easy to update 10

3.2 Ansible – Simple IT Automation 11

4 Domain Name System **12**

4.1 DNS theory 12

4.2 Consul by HashiCorp 14

4.3 Dnsmasq 16

5 Other components **16**

5.1 Oracle Linux 16

5.2 Docker 16

5.3 Registrator 19

5.4 Apache Mesos 20

5.5 Apache Marathon 21

5.6 Apache Zookeeper 22

6 Development plan **23**

6.1 Overview 23

6.2 Services plan 25

6.2.1 Docker CE 25

6.2.2 Consul 25

6.2.3 Dnsmasq 26

6.2.4 Registrator 26

6.2.5 Mesos 26

6.2.6 Marathon 26

6.2.7 Zookeeper 27

6.3 Plan for environment and testing 27

7 Development 28

7.1 Environment setup 28

7.2 Docker role 32

7.3 Consul role 34

7.4 Dnsmasq configuration 43

7.5 Registrator configuration 45

7.6 Mesos configuration 46

7.7 Marathon configuration 47

7.8 Zookeeper configuration 48

7.9 QVCP role 49

7.10 Ansible setup for installation 50

7.10.1 Roles 50

7.10.2 Variables 51

7.10.3 Other settings 52

7.10.4 Playbook 53

8 Installation **54**

9 Testing **57**

10 Results **63**

11 Conclusons **63**

References **65**

Appendices **68**

Appendix 1. Master groups and users 68

Appendix 2. Agent groups and users 69

Appendix 3. Master playbook 70

Appendix 4. Agent playbook 71

Appendix 5. Docker inspect for Consul in cp00 72

Appendix 6. Docker inspect for Marathon in cp00 82

Appendix 7. Docker inspect for Zookeeper in cp00 89

Figures

Figure 1. DNS hierarchy 13
Figure 2. Consul protocols and architecture 15
Figure 3. Docker engine structure 17
Figure 4. Docker architecture 19
Figure 5. Registrator dockerfile 19
Figure 6. Mesos infrastructure 20
Figure 7. Mesos architecture 21
Figure 8. Mesos UI 22
Figure 9. Zookeeper architecture 23
Figure 10. Environment topology 24
Figure 11. KVM pool info 28
Figure 12. KVM creation commands in .sh file 29
Figure 13. Created virtual machines 29
Figure 14. SSH connection to cp00 virtual machine 30
Figure 15. Included services for virtual machines 30
Figure 16. SSH key creation 31
Figure 17. Docker role directories 32
Figure 18. Playbooks in tasks 32
Figure 19. Main.yml playbook 33
Figure 20. Docker_ce.yml playbook 34
Figure 21. Consul role directories 34
Figure 22. Consul configuration files in templates 35
Figure 23. Server configuration 36
Figure 24. Agent configuration 36
Figure 25. Firewall configuration template 37
Figure 26. Server playbook 38
Figure 27. Agent playbook 39
Figure 28. Firewalld playbook 39
Figure 29. Main playbook 41
Figure 30. Default variables 42
Figure 31. Firewall ports 43

Figure 32. Dnsmasq configuration ... 43
Figure 33. Dnsmasq playbook ... 44
Figure 34. Dnsmasq and Consul ports ... 45
Figure 35. Mesos main.yml playbook ... 46
Figure 36. Marathon configuration ... 47
Figure 37. Marathon installation playbook ... 47
Figure 38. Default settings ... 48
Figure 39. Firewall service for Marathon. ... 48
Figure 40. Zookeeper configuration ... 49
Figure 41. Zookeeper playbook ... 49
Figure 42. Repositories included in QVCP role ... 50
Figure 43. Roles in directory roles ... 50
Figure 44. Hosts file ... 51
Figure 45. Group_vars settings ... 51
Figure 46. Extra_vars ... 52
Figure 47. Package versions ... 52
Figure 48. Ansible configuration ... 53
Figure 49. Included roles ... 53
Figure 50. Playbook run ... 54
Figure 51. Playbook installation recap ... 55
Figure 52. Registered services ... 55
Figure 53. Play recap ... 56
Figure 54. Consul UI and running services ... 56
Figure 55. Mesos UI ... 57
Figure 56. Marathon using a ready image from a repository ... 58
Figure 57. Running container in Marathon UI ... 58
Figure 58. Application stdout ... 58
Figure 59. Automatically registered tweet container ... 59
Figure 60. Active tasks ... 59
Figure 61. Docker ps on cp10 ... 59
Figure 62. Exec and ping ... 60
Figure 63. Docker logs ... 60
Figure 64. Traceroute from the container ... 60

Figure 65. Nslookup from the container ... 60
Figure 66. Consul members ... 61
Figure 67. KV storage in Consul ... 61
Figure 68. Consul configuration set by Ansible ... 62
Figure 69. Mounted configuration in Consul container ... 62

Tables

Table 1. Common Docker - command options ... 18
Table 2. Names, addresses and purposes for virtual machines ... 27

Acronyms

API	Application Programming Interface
ccTLD	Country Code Top-Level Domain
CLI	Command-Line Interface
DNS	Domain Name System
gTLD	Generic Top-Level Domain
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
OL	Oracle Linux
RCP	Remote Procedure Call
SSH	Secure Shell
TCP	Transmission Control Protocol
TLD	Top-Level Domain
UDP	User Datagram Protocol
YAML	YAML Ain’t Markup Language
WAN	Wide Area Network

1 Introduction

The purpose of this bachelor's thesis is to research and develop an automated solution for installing Domain Name System (DNS) services for production environments. This bachelor's thesis was assigned by Qvantel Finland Oy.

The DNS is installed and configured by using Ansible, which is an automation software for deployment and configuration management. DNS services used are Consul by HashiCorp and Dnsmasq. Consul and Dnsmasq are to be used as a part of a larger environment that has multiple different services registered and connected through the Consul service. Other services used to demonstrate the use of Consul are Apache Mesos, Apache Marathon, Apache Zookeeper and Registrator. These can be used to run and manage applications in containers; Registrator is used as an automatic registering tool for Consul. Consul acts as a DNS for these services and also works as a simple monitoring tool with health checks and key/value storage. During the development of Consul both the Marathon and Zookeeper were also containerized and new roles were created. As Marathon and Zookeeper require java to work and by having them in containers removes the java requirement from hosts, improving security. All but Mesos and Dnsmasq will be running in Docker containers. Dnsmasq is used to forward Consul ports. Ansible playbooks and settings were built according to already existing Ansible Qvantel-made installations that the playbooks and settings of this thesis are modeled from.

The theory section discusses all the components and explains how they work. The main focus is on Consul, Ansible and Docker CE. The development plan includes a plan on how to create a working environment using the given components. The entire system should contain an automated installation and configuration. In practice, this system will be installed by using Ansible on an empty environment.

The project results in an automated Consul and Dnsmasq installation with a configuration that works and can be used in real customer deployments.

2 Research methodology and requirements

2.1 Method

This thesis was planned and created by using constructive research, which is practical and where the end result is known; however, how to implement it is not. It is common in constructive research to use existing information to build something new. With constructive research, it needs to be decided on what to build and how to build it. Constructive research was selected because the thesis is about building from existing components something new and improved. Research question is how to create an automated DNS service installation and how to implement it to the platform. (Pasian 2018, 8)

2.2 Requirements and goals set by Qvantel

This thesis is carried out as a part of Qvantel's platform development. The components and the overarching plan are guided by the company's specific needs. Addresses, names and testing are created for this thesis only, and they are not meant to be translated directly to Qvantel's use.

The requirement for the Consul is to work modularly and not be affected by a different number of servers or datacenters. The main purpose of Consul is to work as a name service for different applications and a simple monitoring tool. Consul is also meant to work as a key/value storage for applications. Dnsmasq is used as a port forwarder for Consul. All possible components should be running in Docker containers. All Internet Protocol (IP) addresses and possible data paths should be easily configurable.

3 Why automate

3.1 Easy to install – Easy to update

In year 2016, around 47 percent of the world's population used the internet. The highest percentage of users concerning population is in Europe, especially in northern Europe. In addition, Americas and the former Soviet Republics have more than fifty percent of their population using the internet. The annual growth from 2015 to 2016 had an increase of 4 percent. (Taylor 2016)

As cloud computing has become more popular, cloud computing services such as Amazon Web Services (AWS) have more than one million users worldwide and make a large profit by providing platforms for companies and private users (Amazon letter to shareowners 2015, 5).

Even if cloud computing and regular bare metal servers are competing for customers, it does not change the fact that every service still needs to be installed and configured. Even if creating a server has become as easy as pressing a button, the rest of the system still requires a heavy load of work in installing and configuring the wanted services. (Ansible – Overview n.d.)

It is important for companies to have services installed, configured and running as fast as possible. Downtime from server maintenance and updating can quickly increase if the process is not consistent. For these reasons automating deployment and configuration cannot only help employees but also reduces the cost of long downtimes. Especially in larger companies with thousands of servers it would normally take weeks to update something as normal bash scripting is no longer possible. Reducing the cost of downtimes and making updating, installing and configuring more consistent has made automation popular with all large companies. (Benson, Prevost & Rad 2016, 1)

The purpose of automation is to reduce the amount of time and effort in deployments and decrease the possibility of user failure. In practise, it means that most of the time spent on a service that is being built is front-loaded. After the development, the component is meant to be easy to install by anyone. Automation can be used for

many different purposes. In this case, Ansible can be used for cloud provisioning, configuration management, application deployment, intra-service orchestration and for many other uses. (Ansible introduction n.d.)

The other major purpose of automation is to simplify updating systems and software, as upgrading a service only needs one number change and a rerun with the installation. Of course, breaking changes in services might need some configuration changes; however, in theory updating should be quick to test, change and deploy. (Ansible introduction n.d.)

3.2 Ansible – Simple IT Automation

Ansible is a simple automation engine. It was released in 2012 by Michael DeHaan. Ansible has since become very popular. YAML files are used by Ansible to operate and also as source of information. Ansible uses Python as its language. The commands used by Ansible are run via SSH to either the local machine or distant server. (Heap 2016.)

Ansible connects to the machines defined in its inventory file by using SSH as default. This file can be used to place different machines to different groups, hereby telling Ansible which machines belong together. After the connection Ansible will send out small programs called Ansible Modules to specified hosts. Modules are resource models for the system and define the desired state of the system. After executing these modules in the node, Ansible will remove them. (Ansible introduction n.d.)

Just like the target hosts, also variables can be defined. In group_vars/ or host_vars/ user can set up variables. These variables can be then used with all the configuration under this upper level of Ansible. For example, user can define what file path to use with all the services and make it a variable data_path. Then the user can use data_path in configurations and Ansible will change it to the defined path. Variables can also be defined in an inventory file just for one specific server or directly in the role of the service. (Heap 2016, 5)

Roles include the services that are run with Ansible. There can be only one role or multiple ones. The roles can include variables for that specific role, installation

scripts, configuration and firewall rules. Installation scripts are usually called playbooks. If there is only one role in question, then its playbook can be used to install that specific role with ansible-playbook command by defining a target host in the same command. With multiple roles, a collected playbook can be created that defines what roles to use, what hosts to install and what upper level variables to include in those roles. (Heap 2016, 4)

Ansible playbooks are built upon tasks that are executed from top to bottom. The tasks can do anything from installing with package managers to insert text to a file. Playbook can be configured to do certain tasks only when conditions are met. Ways to configure playbooks depend on the one who makes it. Ansible offers a large documentation of all possible commands to use with playbooks. (Ansible documentation - Playbooks n.d.)

4 Domain Name System

4.1 DNS theory

IP addresses are used to identify machines operating in the internet. IP addresses are difficult to remember, so usually name of the network interface is used instead. Every IP address has a network interface. These are commonly known as domain names. An IP address can have multiple domain names attached to it. IP addresses and the domain names are stored in the Domain Name System (DNS) database. (Dostalek & Kabelova 2006, 1)

Internet has different domains that are logically grouped together. Domains specify where the names belong. These can be countries, companies or generic domains. Subdomains are domains inside Top Level Domains (TLD). For example

example.thesis.com

has the subdomain called thesis that is a subdomain of the domain com. TLD consists of two TLDs, Generic (gTLD) and Country Code (ccTLD). Commonly known gTLDs are

for example com and net. According to ISO 3166 there are two letter ccTLD for individual countries. For example, fi is affiliated with Finland. (Dostalek & Kabelova 2006, 1.1)

DNS is structured in a hierarchy using different areas. On the top is the root server. An uncached DNS query triggers a DNS lookup that starts from the root server. From there it will travel down to the TLD servers and forward to the specific domains or subdomains. When the query hits the name server of the correct domain, the IP is sent to the client. (Figure 1) (DNS root server n.d.)

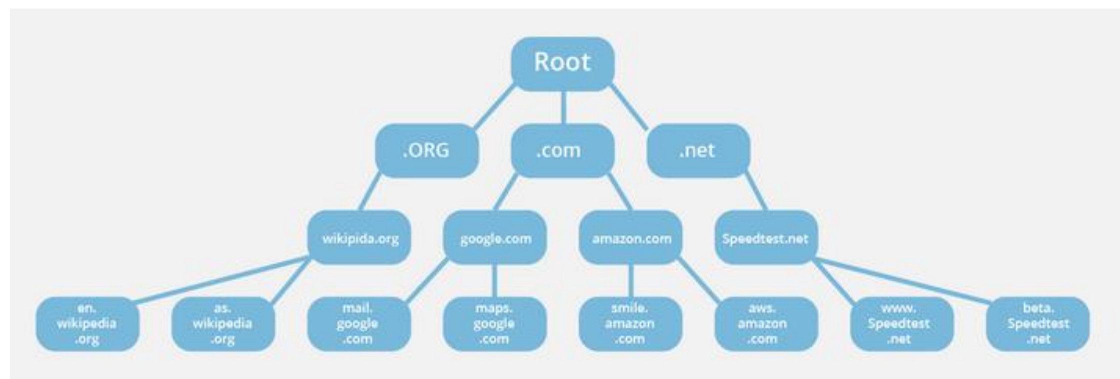

Figure 1. DNS hierarchy

DNS protocol can work with many different types of operations. The most common of them being DNS QUERY. The query makes it possible to obtain records from DNS database. New modifications have brought new operations, such as NOTIFY and UPDATE. DNS protocol works on query and answer basis, meaning that usually a client sends a query to the server and the server answers it with the needed information. DNS protocol can also use compression to make packets as small as possible. DNS protocol cannot transfer packages by itself, it being application-layer protocol. This why transferring packages is delegated to transport protocols. DNS protocol uses both User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) for sending information and default DNS ports are on 53/UDP and 53/TCP ports. (Dostalek & Kabelova 2006, 2.2)

DNS client settings on Linux operating systems are located in resolv.conf file under /etc/ directory. First line in the resolv.conf usually contains the domain or search key-

word. If domain is defined, the domain is added to unqualified domain names automatically. Search keyword is used to specify domains to be added to unqualified names. Keyword can be used to define up to six domains. Only domain or keyword can be used. Resolv.conf usually contains a list of DNS servers also. These can be defined by using name servers keyword and IP address. First server on the list is used first. (Rampling, Blair & Dalan 2003, 5.4)

4.2 Consul by HashiCorp

Consul is a service mesh solution, providing configuration, service discovery and other functionalities. All these features can be used together or separately. Consul's service discovery enables clients to register a service and other services can thereby discover providers of the given service. Consul can use DNS or HTTP to easily find services. Consul also provides health checking, key/value storage for dynamic configuration, TLS readiness and multi datacenter support. (Introduction to Consul n.d.)

Consul agents or clients are installed on every host that provides services to Consul. Agent is responsible for making health checks on the node it is installed on. Agents then talk to Consul servers where the data is stored and replicated. Servers elect a leader by themselves using bootstrapping. Services that need to discover other services can query any server or agent to discover the needed services. Consul is built on Serf that provides a full gossip protocol with membership, failure detection and event broadcast. Consul also supports multiple datacenters. (Consul Documentation – Architecture n.d.)

Consul uses Remote Procedure Call (RPC) to allow client to make requests to server. Consul uses different gossip pools for Local Area Network (LAN) and Wide Area Network (WAN). LAN contains all the nodes located in the same local area network or datacenter. WAN contains only servers, usually from all datacenters that are connected. (Consul Documentation – Architecture n.d.)

Figure 2 tells how Consul service works between two datacenters. Clients use LAN gossip between reach other and RPC for server. WAN gossip is used between different datacenters. Servers decide on the leader, that handles replication to other servers. The ports used between different tasks are different.

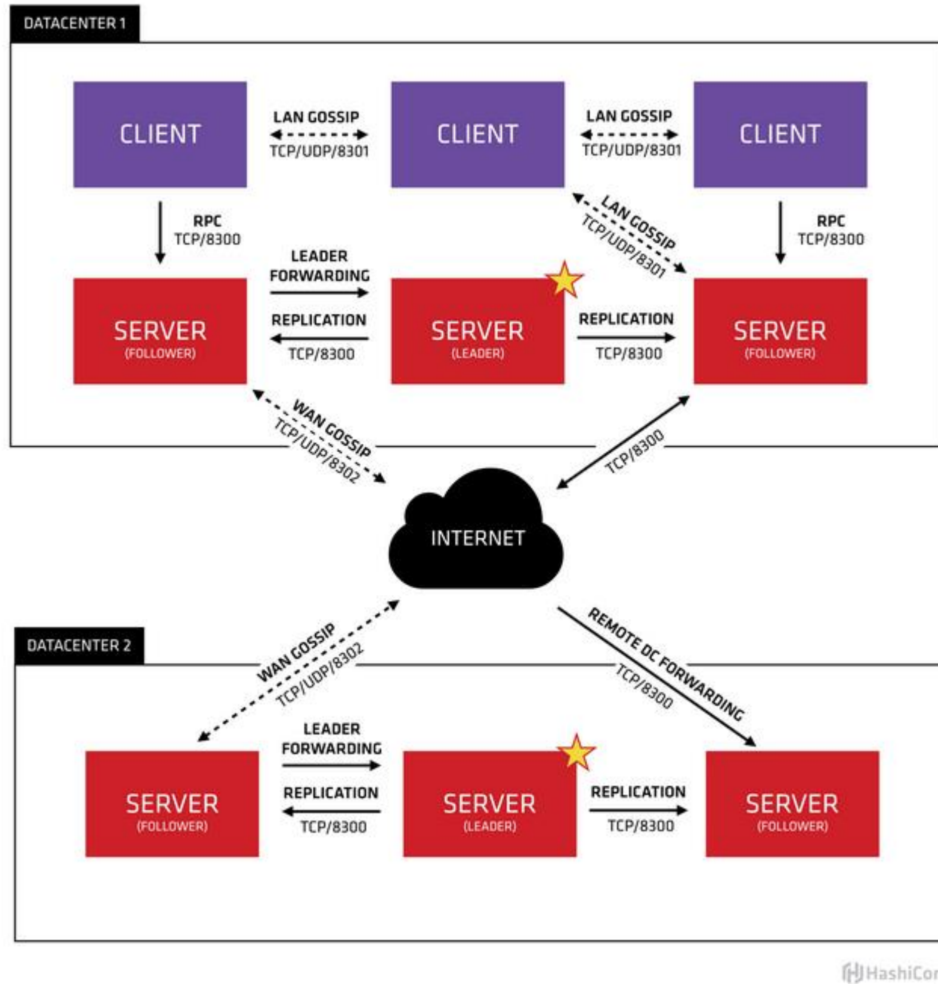

Figure 2. Consul protocols and architecture

Consul container needs to be configured with the correct cluster address or bind address. This is the address that other agents use to contact the agent. Client address is for processes on the host to contact Consul. Retry join allows Consul to try and join a cluster. Retry option allows Consul to retry joining if the first time fails. Consul can forward DNS on port 53, however, in most systems running Consul as root is not a safe choice. In this case, Consul can work with different DNS servers to forward queries to Consul. Services like BIND and Dnsmasq work well in this setup. With containerized Consul, its data directory needs to be available after restarting, which means it has to be mounted. (Consul Documentation – Consul Containers n.d.)

4.3 Dnsmasq

Dnsmasq is meant to provide infrastructure for networks. It provides a local DNS server for the network and forwards queries to upstream DNS servers. Dnsmasq is lightweight, designed not to take many resources for the system. (Dnsmasq documentation n.d.)

Dnsmasq is configured in /etc/dnsmasq.conf. By default, Dnsmasq reads /etc/hosts. Hosts can be added to the hosts file for Dnsmasq to pick them up. Dnsmasq can be also used with /etc/resolv.conf. One way is to use local address in resolv.conf and enter all upstream nameservers in dnsmasq.conf (Schroder 2018).

5 Other components

5.1 Oracle Linux

Oracle Linux (OL) is developed by Orcale Corporation that has multiple products such as databases, OpenJDK and VirtualBox. The newest version of OL is currently 7.5. It uses Yum as its package updater. Oracle Linux is compatible with Red Hat Linux and is able to run same applications. (Oracle – Linux n.d.)

5.2 Docker

Docker is meant to be a platform for developing, packaging and running applications. It is possible to build, ship and run applications on any platform as long as Docker can be installed and is working there. Docker images use the same Operating System (OS) kernel as the host machine, making it different from virtual appliances. Docker software runs in an environment called container. Each container includes their own file system and environment variables. These containers are isolated from each other and from the OS running on host. Figure 3 describes how Docker engine is structure and what it controls. (Vohra 2016, 1)

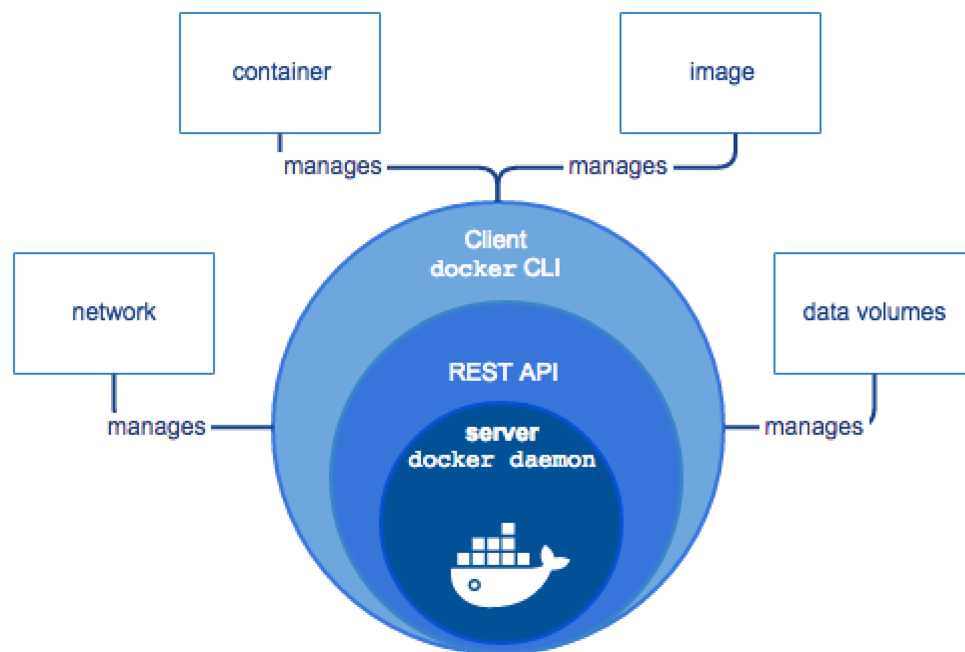

Figure 3. Docker engine structure (Docker documentation – Docker architecture n.d.)

Container contains everything a software would need to run. Files can be copied from host into the container if needed by mounting files from host to container. Containers can also be linked together so that they can communicate with each other if needed. (Vohra 2016, 1)

To build a container Docker uses dockerfile to build an image. Dockerfile consists of instructions on what to download, commands, networking, directories and variables. By running dockerfile, the base image is automatically configured depending on what the commands are in the dockerfile. Docker will then create a new image with all the configurations defined in the dockerfile. (Bernstein 2014, 83)

Many images are already made and are available in the official Docker Hub. Docker can be downloaded from the most common repositories. Command "docker" can be used to perform different tasks for Docker. Table 1 describes most common options for docker - command.

Table 1. Common Docker - command options

Docker command option	Result
ps	Prints out containers that are running.
start	Starts specified container.
stop	Stops specified container.
rm	Removes specified stopped container.
rmi	Removes specified image.
exec	Runs commands in running container.
export	Exports containers filesystem as a tar.
import	Imports containers filesystem to a im-age.
save	Saves images to a tar archive.
logs	Fetches logs generated by a container.
images	Shows images.
pull	Pulls a Docker image.
network	Commands Dockers networks.

Docker Engine is a client-server application with three major components. A server is a program called daemon process which does the building and running containers. The daemon creates and manages images, containers, networks and volumes. A REST API tells programs what interfaces to use to contact daemon. REST API also instructs daemon on what to do. A CLI is used with command docker. CLI uses REST API to control Docker daemon. REST API is used between client and daemon to communicate over UNIX sockets or by a network interface. Figure 4 describes how containers are built by client using daemon to pull images and running them. (Docker documentation – Docker architecture n.d.)

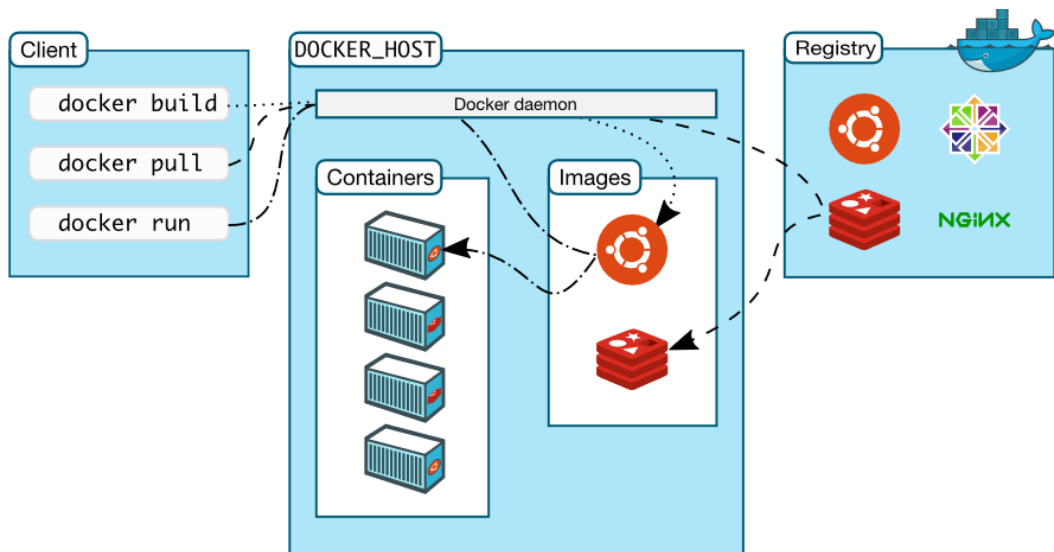

Figure 4. Docker architecture (Docker documentation – Docker architecture n.d.)

5.3 Registrator

Registrator is used to automatically register Docker containers. This happens by Registrator inspecting containers when they start. Registrator is developed by Gliderlabs. Registrator supports different service registries such as Consul. Registrator can be downloaded from Docker Hub by pulling its image with docker pull. Figure 5 describes Registrators Dockerfile. (Registrator ReadMe n.d.)

```
FROM alpine:3.7 AS builder
COPY . /go/src/github.com/gliderlabs/registrator
RUN apk --no-cache add -t build-deps build-base go git curl \
       && apk --no-cache add ca-certificates \
       && export GOPATH=/go && mkdir -p /go/bin && export PATH=$PATH:/go/bin \
       && curl https://raw.githubusercontent.com/golang/dep/master/install.sh | sh \
       && cd /go/src/github.com/gliderlabs/registrator \
       && export GOPATH=/go \
       && git config --global http.https://gopkg.in.followRedirects true \
       && dep ensure \
       && go build -ldflags "-X main.Version=$(cat VERSION)" -o /bin/registrator \
       && rm -rf /go \
       && apk del --purge build-deps

FROM alpine:3.7
COPY --from=builder /bin/registrator /bin/registrator
COPY --from=builder /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/ca-certificates.crt
ENTRYPOINT ["/bin/registrator"]
```

Figure 5. Registrator dockerfile (Registrator ReadMe – GitHub)

5.4 Apache Mesos

Apache Mesos is a system kernel that abstracts all given resources into a single pool of computing resources, which makes distributing resources such as CPU and memory easier in a larger scale. Mesos began as a research project at UC Berkeley, and its goal was to create a kernel that makes building and running applications easier. Mesos treats a datacenter like one large computer. (Mesos – Why mesos n.d.)

Mesos architecture focuses on a two-level scheduler, which makes it easy to operate, scale and extend. Mesos is used to handle infrastructure level scheduling operations, while framework handles application specific operations. This allows organizations to build their own operations' logic in their applications. Mesos can support different workloads, such as Docker containers, analytics, Big Data and even operating systems. Mesos can scale to tens of thousands of nodes while still having a simple architecture. Figure 6 describes how Mesos combines hardware resources and deploying applications with those resources. (Mesos – Why mesos n.d)

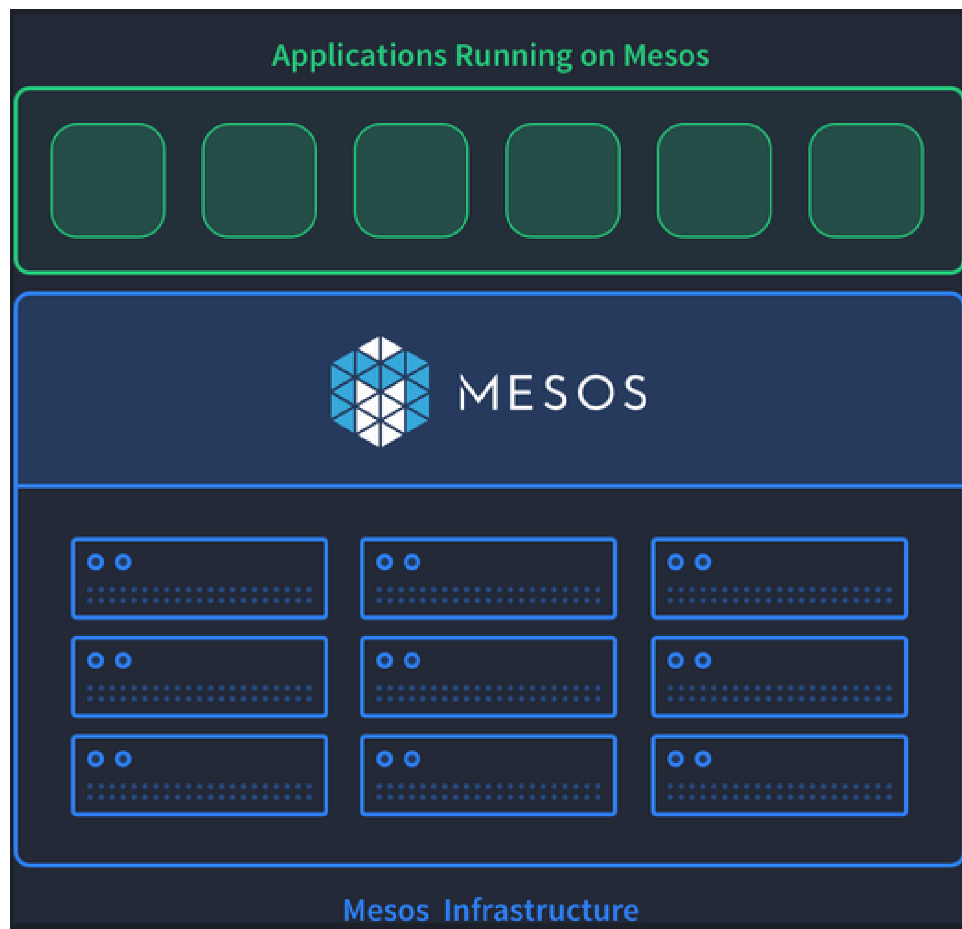

Figure 6. Mesos infrastructure (Mesos – Why mesos n.d.)

Mesos consists of masters and slaves. Masters manage slaves that run on different nodes in the cluster. Framework that is used with Mesos runs tasks on these slaves. Master enables sharing of resources by sending slaves resource offers through the framework. The framework that runs on top of Mesos consists of two different components, scheduler and executor. Scheduler registers with the Master so that it can be offered resources. Executor is connected to different slaves and it launches tasks on those nodes. Mesos uses Zookeeper for electing a leading master. Figure 7 describes how mesos architecture works. Leading master node is chosen by zookeeper quorum and agents are connected to that master. Hadoop and MPI are described here as frameworks. (Mesos Documentation – Architecture n.d.)

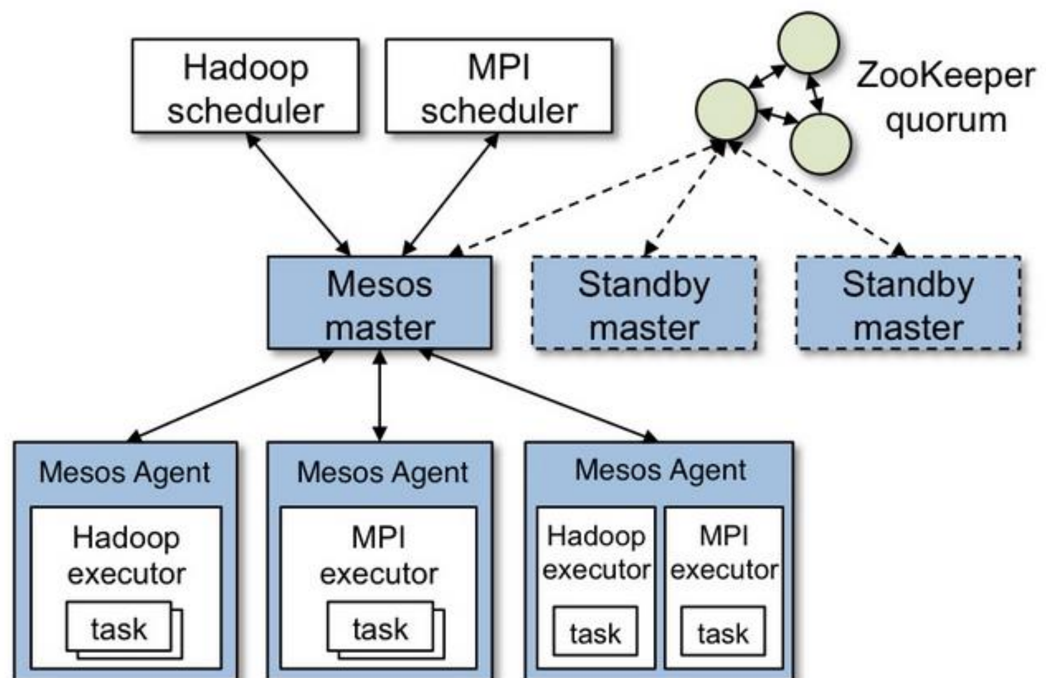

Figure 7. Mesos architecture (Mesos Documentation – Architecture n.d.)

5.5 Apache Marathon

Apache Marathon is a container orchestration platform for Mesos. Marathon orchestrates both the applications and frameworks. For Marathon to work it needs to have Mesos already installed. Marathon is also available in an official Docker container.

Marathon also uses Zookeeper to store state. Marathon provides REST API for starting, stopping and scaling applications. It is also able to run in high-availability mode by running many copies of itself. Marathon runs in HA mode by default. The state of running tasks is stored in Mesos. (Mesosphere – Marathon documentation n.d.)

Marathon can use command-line flags to set environment variables that start with MARATHON_ line. For example, MARATHON_MASTER will set an option for master node. The only flag required is the master flag. Master flag will set the URL of the Mesos master. If Zookeeper is in use, those addresses should be used. These environment variables are usually stored in /etc/default/marathon. Figure 8 describes a running application container on Marathon UI. (Mesosphere – Marathon documentation n.d.)

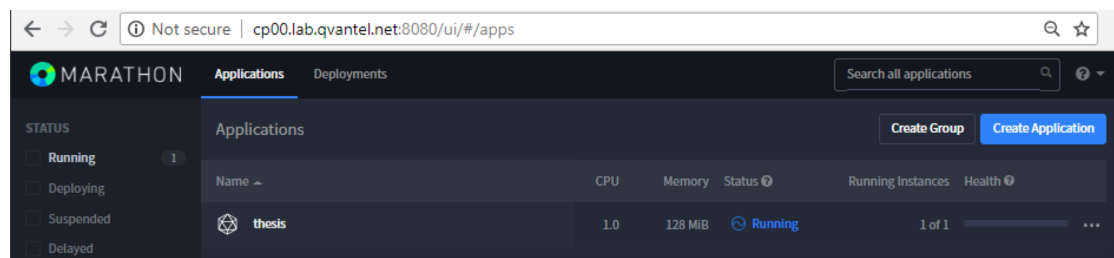

Figure 8. Mesos UI

5.6 Apache Zookeeper

Apache Zookeeper is a coordination service for applications. It uses a set of primitives that applications can build upon to implement services. Zookeeper is used for synchronization, configuration management, groups and naming in different distributed systems. Zookeeper allows processes to coordinate through namespace that is similar to standard file systems. Zookeeper data is kept in-memory, which provides high throughput and low latency. Zookeeper is meant to be replicated over different hosts to prevent failure if one node goes down. Figure 9 describes Zookeeper architecture with servers and clients connecting between each other. (Zookeeper – Overlook n.d.)

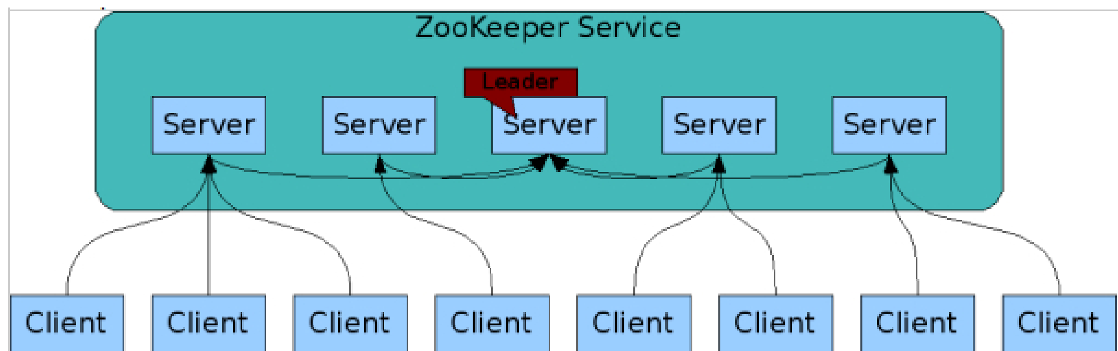

Figure 9. Zookeeper architecture

Zookeeper servers must know of each other for Zookeeper service to work. They maintain an in-memory image of state with transaction logs and snapshots in a persistent store. All Zookeeper servers maintain an image of state in in-memory, that also has logs and snapshots in a persistent store. This makes Zookeeper service available as long as majority of servers are available. Clients connect to a Zookeeper server to which it maintains TCP connection. This connection is used for sending requests, receiving responses, events and heart beats. If the connection to that specific server fails, the client will connect to a different Zookeeper server (Zookeeper – Overlook n.d.)

Zookeeper's configurations should contain a path for data directory and client port. If using replicated servers, Zookeeper servers need to be specified with their own ID and all the servers added to each one's configuration. It is also good to have initLimit and syncLimit when running replicated Zookeeper. Init will set a timeout for the length of the time the servers in quorum have to connect to a leader. Sync limits how out of date a server can be from the leader. (Zookeeper Documentation – Getting started n.d)

6 Development plan

6.1 Overview

The plan was to develop a working Consul and Dnsmasq Ansible roles that are highly modular and can scale to any environment size. As requested, all possible platform

services should be official Docker containers to ease their eventual upgrading. The rest of the services used in this thesis is also a part of the platform; however, as they are only to support the testing of Consul, their roles are explained briefly. Marathon and Zookeeper are also containerized at the same time as Consul but are given less attention. Docker, Mesos and Registrator roles are ready-made roles that have already existed before. Consul and Dnsmasq are to be included in the same role and Consul is to be containerized from the old host version. Dnsmasq will remain almost the same as before but it needs to be integrated with Consul and tested. The environment used for testing was a blade server running a total of eight virtual machines, three of which are master nodes and five agent nodes. Figure 10 describes the topology of the planned environment.

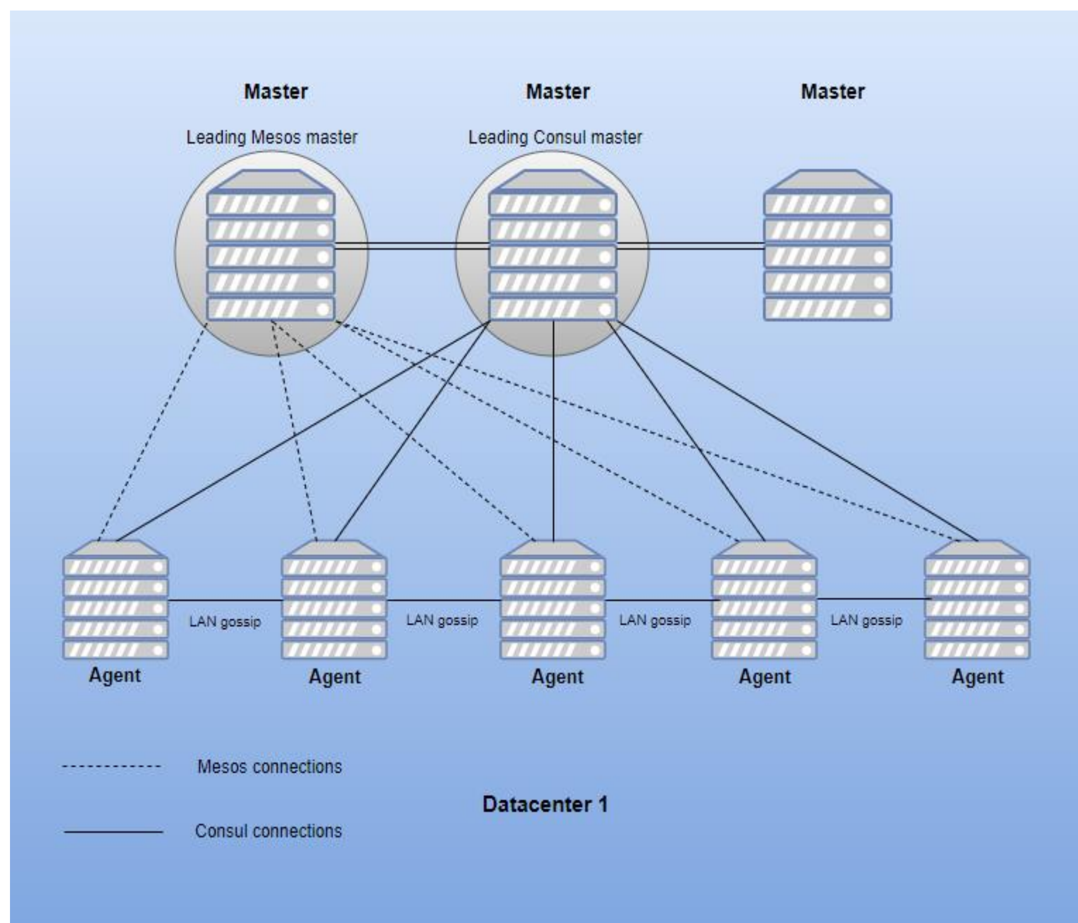

Figure 10. Environment topology

6.2 Services plan

The development started by creating each role that has a playbook to create needed directories and moves the configurations to those directories. Playbook also included Docker container installation and if needed, other setups. Roles configuration was included as a template that was moved during installation to a correct directory. In addition, all the possible variables needed to be placed inside the roles and to group_vars/ and extra_vars. The versions were to be defined on higher level, in var/versions.yml. Repositories used were Qvantel's own private repository that has all the packages needed included. This was included as a role to create a repository path to /etc/yum.repo.d/. Ansible playbooks can be built in many different ways and are also very forgiving with the placement of options.

6.2.1 Docker CE

Docker works as a platform for all the services. It was to be installed with yum, just like Dnsmasq and Mesos. Docker was already working Ansible role that did not need any configuration and could be added directly to roles. The version used was the latest stable one, 18.03.0 in this case. Docker was installed with the Ansible playbook first so that other services could run on top of it.

6.2.2 Consul

Consul was to be updated to containerized Consul. New playbook was to be created to install the container, set up directories and templates, all possible addresses and paths changed to work as variables. Firewall rules were included in the role. Consul was to be able to install from two choices, server and agent. The servers have bootstrap setting included that would change depending on how many master nodes there are. Agents were to be similar to servers without bootstrapping. They were also to connect themselves to server nodes to register services running on those nodes. Consul playbook would also add its address to resolv.conf to work as a DNS for the node. The newest stable version of Consul was to be used, which at the time of writing the thesis was 1.1.0.

6.2.3 Dnsmasq

Dnsmasq installation was to be included to Consul's playbook as it is always installed with Consul. As Dnsmasq is installed on a host, it uses yum to install the service. In this case, Dnsmasq was to be left mostly untouched from the previous versions and only have its connectivity with the container checked and changed if needed. The newest stable version of Dnsmasq was to be used.

6.2.4 Registrator

Registrator was to register containers running on top of Docker to Consul, which was to be made so that containers do not need to register by themselves. Registrator was already working Ansible role and did not need any configuration. The version used was an old one; however, still the latest official release from master branch.

6.2.5 Mesos

Mesos, like Dnsmasq, was to be installed on host machine. Newest stable version, 1.5.0, was to be used. It was possible to install Mesos as either master or slave. Masters were installed to master nodes and slaves to slave nodes. Master nodes were to work with Marathon and Zookeeper to allocate applications running on top of Marathon to different slave nodes. In slave nodes Mesos, after receiving tasks from Master, was to create a container running the application deployed through Marathon. In this case, as there are three masters, their quorum was to be 2. Should one node fail or break, the other two would continue to run as normal. Quorum setting was to be installed with Mesos masters. Mesos also needed to be registered to Consul with a separate consul.yml file.

6.2.6 Marathon

Marathon was to be containerized as the latest stable version, 1.5.8 and installed on master nodes with Mesos and Zookeeper. Marathon was to have its addresses and paths changed to variables and playbook created for installation and creating directories for configuration. The configuration had to include settings for Mesos master and Zookeeper connectivity. Firewall rules needed to be added to the role.

6.2.7 Zookeeper

Zookeeper was to be containerized as the latest stable version, 3.4.12. Zookeeper was to be installed on master nodes and it would take care of choosing a leading master for Mesos and guide Marathon to find Mesos masters. Zookeeper was to have its playbook created for installation and creating directories with configuration. Firewall rules were also to be added to the role. All the possible settings were to be changed into variables and defined on a higher level. Zookeeper ID for replication was to be described in hosts file with addresses.

6.3 Plan for environment and testing

The plan was to have an environment with eight virtual machines, all of them with similar specifications. Three of the machines were to be selected as master nodes with Consul masters, Mesos masters, Marathon, Zookeeper and Registrator. The other five were to act as slave machines with Consul agents, Mesos slaves and Registrators. It is good to have more slave nodes than masters as slaves are actually doing the hard work after Marathon would have all its applications running on top of it. In this case, it really did not matter; however, it is good to have a realistic scenario. Table 2 describes domain names, addresses and purposes of virtual machines used in testing environment.

Table 2. Names, addresses and purposes for virtual machines

Server name	Server address	Server purpose
cp00.lab.qvantel.net	192.168.81.60	Master
cp02.lab.qvantel.net	192.168.81.62	Master
cp04.lab.qvantel.net	192.168.81.64	Master
cp06.lab.qvantel.net	192.168.81.66	Agent
cp08.lab.qvantel.net	192.168.81.68	Agent
cp10.lab.qvantel.net	192.168.81.70	Agent
cp12.lab.qvantel.net	192.168.81.74	Agent
cp14.lab.qvantel.net	192.168.81.76	Agent

After running the playbooks and everything was installed properly, testing could begin. The testing took place by trying to find out if the Consul nameserver works with nslookup, ping and traceroute. Consul’s own commands to check on members were to be used inside the container. Different logs from all the services were given a look to make sure nothing was giving an error. In addition, the Consul’s UI was to be monitored and explored. Marathon was to be used to deploy a mockup service to demonstrate its use and that it works as intended. These services would then be registered to Consul after they were created on one of the slave nodes.

7 Development

7.1 Environment setup

The environment was to be created on a single blade server that ends up having eight Oracle Linux virtual machines. These machines were to be the latest possible version of 7.5. Machines were to be created with KVM and managed with Virsh. To create the machines, a readymade script was to be used and they would be deployed with the needed information about the machines. The server needed to have a disk allocated for KVM to use. In this case, the pool used was a 930 GiB disk (Figure 11).

```
[root@c02h001h lab-tools]# virsh pool-info default
Name:           default
UUID:           77de89f2-c103-41db-b4eb-fcb8fa389176
State:          running
Persistent:     yes
Autostart:      yes
Capacity:       930.39 GiB
Allocation:     450.00 GiB
Available:      480.39 GiB
```

Figure 11. KVM pool info

Virtual machines were to be created by running the create-cpvm script. These commands would be then collected into a .sh file that can be used to install all of them at once. Figure 12 shows all the combined create-cpvm commands with addresses, hostnames and resource allocations.

```
#!/bin/bash

set -e

./create-cpvm -c 2  -m 8  -d 50  -o 'ORC'        192.168.81.60 cp00
./create-cpvm -c 2  -m 8  -d 50  -o 'ORC'        192.168.81.62 cp02
./create-cpvm -c 2  -m 8  -d 50  -o 'ORC'        192.168.81.64 cp04
./create-cpvm -c 2  -m 8  -d 50  -o 'APP'        192.168.81.66 cp06
./create-cpvm -c 2  -m 8  -d 50  -o 'APP'        192.168.81.68 cp08
./create-cpvm -c 2  -m 8  -d 50  -o 'APP'        192.168.81.70 cp10
./create-cpvm -c 2  -m 8  -d 50  -o 'APP'        192.168.81.72 cp12
./create-cpvm -c 2  -m 8  -d 50  -o 'APP'        192.168.81.74 cp14
```

Figure 12. KVM creation commands in .sh file

To install the virtual machines, the file is to be run as a command

[root@c02h001h lab-tools]# ./c02h001h.sh

This installs the machines with the correct number of cores, RAM, disk, IPs and with a hostname. After installation, the created virtual machines can be seen and managed with Virsh. Figure 13 shows the result of virsh list –all command after installing VMs.

```
[root@c02h001h lab-tools]# virsh list --all
 Id    Name                           State
----------------------------------------------------
 741   cp00                           running
 742   cp02                           running
 743   cp04                           running
 744   cp06                           running
 745   cp08                           running
 746   cp10                           running
 747   cp12                           running
 748   cp14                           running
```

Figure 13. Created virtual machines

The most basic command includes list, shutdown, reboot and start. These can be listed with

[root@c02h001h lab-tools]# virsh –help

Machines take a moment to install and update and then they automatically turn off. After that, they need to be restarted. By then, the virtual machines can be accessed with SSH (Figure 14).

```
avalkeinen@QFINW033:~$ ssh avalkeinen@cp00.lab.qvantel.net
Warning: the ECDSA host key for 'cp00.lab.qvantel.net' differs from the key for the IP address '192.168.81.60'
Offending key for IP in /home/avalkeinen/.ssh/known_hosts:41
Matching host key in /home/avalkeinen/.ssh/known_hosts:56
Are you sure you want to continue connecting (yes/no)? yes
Last login: Thu Aug  2 11:31:19 2018 from 192.168.81.67
Hostname:      cp00
OS:            OracleLinux 7.3
Environment:   lab
Role:          NONE
[avalkeinen@cp00 ~]$
```

Figure 14. SSH connection to cp00 virtual machine

Script automatically installs important packages to the machine. For example, Python, yum-utils and NetworkManager. In addition, networks are configured to eth0 so that the machines are ready out of the box. Figure 15 displays some of the packages installed to virtual machines.

```
keyboard us
skipx
timezone --utc Europe/Helsinki
selinux --enforcing

%packages --instLangs=fi
@core
-postfix
-NetworkManager
-NetworkManager-team
-NetworkManager-tui
-NetworkManager-libnm
-NetworkManager-config-server
-*-firmware
-dnsmasq
-wpa_supplicant
-alsa*
-fxload
-microcode_ctl
-libndp
-libsoup
-mysql-community-libs-compat
-ppp
-teamd
-glib-networking
-gnutls
```

Figure 15. Included services for virtual machines

The only thing missing from the host used to run Ansible is Ansible itself. Ansible can be easily downloaded with the command

root@cp00 avalkeinen]# yum install ansible

This installs Ansible to the machine, so it can be used to run playbooks. Ansible is included in the most common repositories, so for example, epel repositories already contain it. If the repository is missing, it can be easily added with the commands

> [root@cp00 avalkeinen]# wget http://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
> [root@cp00 avalkeinen]# rpm -ivh epel-release-latest-7.noarch.rpm

What is also needed is an SSH key. It can be created with ssh-keygen command. In real use the SSH key would be created for the user and user credentials would be used to create it. In this case, these are not needed for this thesis. Figure 16 shows the creation of SSH key.

```
[root@cp00 avalkeinen]#  ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:qUuMHM4kzwSCDYbmISm//MT8iRF3PmoEehSgdPOsO5A root@cp00
The key's randomart image is:
+---[RSA 2048]----+
|o+.+             |
|O*. =            |
|O.+  +           |
| oo.= . ..       |
| E.B++ oS        |
|  =XB+..o        |
|   *B=+o .       |
|    +.=.         |
|     ..          |
+----[SHA256]-----+
```

Figure 16. SSH key creation

Login to these servers works with Kerberos and is managed by Qvantel. After login in, the environment is ready for installation.

7.2 Docker role

Docker is a ready role that has been used before. It does not require any configuration for this development work. Docker role is like all the other roles, as they all will build the same way. The role consists of defaults, handlers, meta, tasks, templates and tests (Figure 17).

```
[avalkeinen@cp00 docker]$ ll
total 32
drwxr-xr-x. 2 avalkeinen staff 4096 Aug  3 13:16 defaults
-rwxr-xr-x. 1 avalkeinen staff  234 Aug  3 13:16 git.sh
drwxr-xr-x. 2 avalkeinen staff 4096 Aug  3 13:16 handlers
drwxr-xr-x. 2 avalkeinen staff 4096 Aug  3 13:16 meta
-rw-r--r--. 1 avalkeinen staff 1118 Aug  3 13:16 README.md
drwxr-xr-x. 2 avalkeinen staff 4096 Aug  3 13:16 tasks
drwxr-xr-x. 2 avalkeinen staff 4096 Aug  3 13:16 templates
drwxr-xr-x. 2 avalkeinen staff 4096 Aug  3 13:16 tests
```

Figure 17. Docker role directories

In the tasks directory one can see different playbooks all combined in the main.yml. Docker roles tasks are docker_ce.yml, docker_engine.yml, main.yml, networks.yml and overrides.yml (Figure 18).

```
[avalkeinen@cp00 tasks]$ ls
docker_ce.yml  docker_engine.yml  main.yml  networks.yml  overrides.yml
```

Figure 18. Playbooks in tasks

The main.yml includes all the playbooks that are tagged with either configure or install. Tags can be used with ansible-playbook command to only run tagged tasks from the playbook. It also handles installing docker-py and starting Docker (Figure 19).

```
---
- include: docker_ce.yml
  when: dockerce_version is defined
  tags: install

- include: docker_engine.yml
  when: docker_version is defined
  tags: install

- name: Install Docker python library
  pip:
    name: docker-py
    version: "{{ dockerpy_version }}"
  tags: install

- include: overrides.yml
  tags: configure

- name: Start Docker
  systemd:
    name: docker
    state: started
    enabled: yes
    daemon_reload: yes
  tags: start_stop_services

- include: networks.yml
  tags: configure
  when: docker_networks is defined
~
~
~
~
~
~
~
~
"main.yml" 29L, 540C
```

Figure 19. Main.yml playbook

In docker_ce.yml docker itself is installed (Figure 20).

```
---
- name: Uninstall Docker
  yum:
    name:
      - docker-engine
    state: absent

- name: Backup yum.conf
  copy:
    remote_src: yes
    src: /etc/yum.conf
    dest: /etc/yum.conf.org

- name: Turn of obsolete check (fix for docker 17.03)
  lineinfile:
    path: /etc/yum.conf
    regexp: '^obsoletes=.*$'
    line: 'obsoletes=0'

- name: Install Docker CE
  yum:
    name:
      - docker-ce{{ '-' + dockerce_version }}
    state: present
  notify: restart docker

- name: Restore yum.conf
  copy:
    remote_src: yes
    src: /etc/yum.conf.org
    dest: /etc/yum.conf
~
~
~
~
~
~
"docker ce.yml" 31L, 575C
```

Figure 20. Docker_ce.yml playbook

This role can be included in the Ansible installation as it is. It is to be included in the roles directory in Ansible's own directory.

7.3 Consul role

To start with the Consul role a directory needs to be created. The role is to be called docker-consul. In docker-consul, there will be four directories, defaults, tasks, templates and vars as well as ReadMe.md for documentation (Figure 21).

```
[avalkeinen@cp00 docker-consul]$ ll
total 20
drwxr-xr-x. 2 avalkeinen staff 4096 Aug  3 13:16 defaults
-rw-r--r--. 1 avalkeinen staff   82 Aug  3 13:16 README.md
drwxr-xr-x. 2 avalkeinen staff 4096 Aug  3 13:16 tasks
drwxr-xr-x. 2 avalkeinen staff 4096 Aug  3 13:16 templates
drwxr-xr-x. 2 avalkeinen staff 4096 Aug  3 13:16 vars
```

Figure 21. Consul role directories

The first things to create are the configurations. Configurations are needed for servers, agents, client ip and firewalls. These are created as .j2 files to templates directory (Figure 22).

```
[avalkeinen@cp00 templates]$ ll
total 16
-rw-r--r--. 1 avalkeinen staff 595 Aug  3 13:16 agent.json.j2
-rw-r--r--. 1 avalkeinen staff  40 Aug  3 13:16 client.ip.j2
-rw-r--r--. 1 avalkeinen staff 311 Aug  3 13:16 firewalld_service.j2
-rw-r--r--. 1 avalkeinen staff 959 Aug  3 13:16 server.json.j2
```

Figure 22. Consul configuration files in templates

The old server configuration was used as a template for the newer configuration. In the configuration, datacenters were defined first. This could be included with "datacenter" option while using variable "{{ consul_datacenter }}".

"datacenter": "{{ consul_datacenter }}"

In addition, the rack that Consul is in can be defined with node_meta and consul_node_rack.

"node_meta": { "Rack": " {{ consul_node_rack)}" }

Both datacenter and rack have variables that are to be defined later in the default directory. Consul will also need its bind address, client address, retry join address and recursors defined which can be set up with lines.

"bind_addr":
"client_addr":
"retry_join":
"recursors":

These are also given variables to set the address in default directory. For bind and client address, the same address can be used. Retry uses variables defined in consul_serverlist, which adds all the master servers' addresses to the retry_join. Recursors use addresses from resolv.conf and are compared to Dnsmasq's listen address, listing correct ones after recursors command. In the configuration the server is defined with:

if consul_role == 'server'
"server": true

It is to be included in the configuration if Consul is marked as server in the primary playbook. In this case, server option becomes true, making it a server and also enabling bootstrapping. Bootstrapping has its variable coming from the server list and transformed into a number; therefore, the number of Consul masters is to be placed there.

“bootstrap_expect”: {{ bootstrap_expect }}

Consul also has its data directory defined with data_dir and UI enabled. These also have variables defined to change them or turn them off. Figure 23 describes settings used in server.yml.

```
{
  "datacenter": "{{ consul_datacenter }}",
  "node_meta": {
    "Rack": "{{ consul_node_rack }}"
  "bind_addr": "{{ consul_bind_address }}",
  "client_addr": "{{ consul_bind_address }}",
  "retry_join": {{ consul_serverlist | to_json }},
  "recursors": {{ ansible_dns['nameservers'] | difference(dnsmasq_listen_address) | to_json }},
  {% if consul_role == 'server' %}
  "server": true,
  "bootstrap_expect": {{ bootstrap_expect }},
  {% endif -%}
  "data_dir": "{{ consul_data_dir }}",
  "ui": {{ consul_ui_enabled }},
}
```

Figure 23. Server configuration

The agent’s configuration is much more simplified as it has no need for many of the options. The only things that it needs to work are bind address, client address and retry join address. In addition, datacenter and rack need to be defined (Figure 24).

```
{
  "datacenter": "{{ consul_datacenter }}",
  "node_meta": {
    "Rack": "{{ consul_node_rack }}"
  },
  "bind_addr": "{{ consul_bind_address }}",
  "client_addr": "{{ consul_bind_address }}",
  "retry_join": {{ consul_serverlist | to_json }},
  "log_level": err,
```

Figure 24. Agent configuration

In client.ip.j2 an address for clients to connect to needs to be defined, which is done by having Consul’s HTTP address in the file.

http://{{ consul_client_address }}:8500

The firewall service is to be copied from already made configuration that takes the service name, its ports and protocols and adds them to firewall services as an XML file. Ports and protocols are to be defined in the roles vars directory. Figure 25 describes firewalld configuration template.

```
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>{{ item.name }}</short>
  {% if item.description is defined %}
  <description>{{ item.description }}</description>
  {% endif %}
  {% for entry in item.ports %}
  <port protocol="{{ entry.protocol }}" port="{{ entry.port }}"/>
  {% endfor %}
</service>
~
~
```

Figure 25. Firewall configuration template

Next, the tasks directory is to be created including the playbooks for server.yml, agent.yml, firewalld.yml and combining them in the main.yml. In server.yml, a task is to be created to install and create a Consul container. Container is then created by invoking docker_container – command. Under docker_container command, container variables and settings for the service can be defined. Because Consul can have many settings, it is smarter to use env file to import these. These environment settings are defined in the default directory.

```
name: start consul server
docker_container:
    env: "{{ consul_env.server }}"
```

Next are the name of the container, where the image is being installed from and what version to use.

```
image: "{{ dockerregistry_url |default(none) }}cp-consul:{{consul_version}}
```

Docker registry is defined in group_vars while the name of the image is always the same but the version is left as a variable consul_version.

Logging options and mount points to the host machine are also included. Mount points are meant to share files and directories between the host and the container.

In this case consul_data_dir is to be a directory on the host machine and it always ends in /data. Mounts can be added with the task volumes.

```
volumes:
"{{ consul_data_dir }}/data:/consul/data"
```

Files can always be found inside the container in /consul/data. In addition, ports, network mode, restart policy and state are defined. Both the environment settings, version to use, mount points and ports are placed as variables and defined in default directory. Figure 26 displays server.yml settings.

```
- name: start consul server
  docker_container:
    env: "{{ consul_env.server }}"
    name: consul-server
    image: "{{ dockerregistry_url | default(None) }}cp-consul:{{ consul_version }}"
    log_driver: json-file
    log_options:
      max-size: 25m
      max-file: "3"
    volumes:
      - "{{ consul_data_dir }}/data:/consul/data"
      - "{{ consul_data_dir }}/config:/etc/consul/config"
    ports: "{{ consul_ports }}"
    network_mode: host
    log_driver: json-file
    privileged: true
    restart_policy: always
    state: started
~
~
```

Figure 26. Server playbook

Agent playbook is similar to the server playbook. It uses the same environment file as the server but with its own settings. In agents playbook there is also the client.ip.j2 file moved to agents data directory. Files can be moved with source and destination commands.

```
template:
    scr: template/client.ip.j2
    dest: "{{ consul_data_dir }}/client.ip"
```

Figure 27 describes agent.yml settings.

```
---
- name: consul agent ip in file
  template:
    src: templates/client.ip.j2
    dest: "{{ consul_data_dir }}/client.ip"

- name: start consul agent
  docker_container:
    env: "{{ consul_env.agent }}"
    name: consul-agent
    image: "{{ dockerregistry_url | default(None) }}cp-consul:{{ consul_version }}"
    volumes:
      - "{{ consul_data_dir }}/data:/consul/data"
      - "{{ consul_data_dir }}/config:/etc/consul/config"
    ports: "{{ consul_ports }}"
    network_mode: host
    privileged: true
    restart_policy: always
    state: started
~
```

Figure 27. Agent playbook

In firewalld.yml the readymade template is moved to firewalld services directory in /etc/firewalld/services and its variables are filled with items from firewalld_services. The playbook also reloads the firewall and enable those rules. Commands can be entered in playbooks with command.

name: Reload firewalld
command: “firewall-cmd –reload”

Figure 28 describes firewalld.yml settings.

```
---
- name: Apply firewalld service configuration file
  template:
    src: templates/firewalld_service.j2
    dest: /etc/firewalld/services/{{ item.name }}.xml
  with_items: "{{ firewalld_services }}"
  tags: install
  register: firewall_services

- name: Reload firewalld
  command: "firewall-cmd --reload"
  args:
    warn: off
  when: firewall_services.changed

- name: Enable firewall rules for service
  firewalld:
    immediate: yes
    permanent: yes
    service: "{{ item.name }}"
    state: enabled
  with_items: "{{ firewalld_services }}"
  tags: configure
~
```

Figure 28. Firewalld playbook

The main.yml file includes all the required paths for Consul. With state setting the path can be set as directory.

```
file:
  path: “{{ consul_data_dir }}/config”
  state: directory
```

Reason to set paths in main.yml is that same paths would need to be specified in both server and agent playbooks. Main.yml creates both data and config paths for Consul and move configurations from templates there. Because the container has been mounted to the host machine, it replicates those folders and what is in them, so configurations can be left to the host machine and the container picks them up. The main playbook also installs python-consul. State can be used to define which version to use.

```
pip:
   name: python-consul
   state: latest
```

Main.yml also sets the container as a server or agent, depending on which one has been defined on a higher level. Playbook also adds Consul to resolv.conf by inserting text to the already existing file with blockinfile.

```
blockinfile:
   dest: /etc/resolv.conf
   block: |
       nameserver {{consul_bind_address }}
       search node.consul
   insertbefore: BOF
   marker: ‘# consul dns
```

Main.yml also includes all the playbooks created before (Figure 29).

```
---
- name: create path for consul configuration
  file:
    path: "{{ consul_data_dir }}/config"
    state: directory

- name: apply consul configuration
  template:
    src: templates/{{ consul_role }}.json.j2
    dest: "{{ consul_data_dir }}/config/{{ consul_role }}.json"

- name: create path for consul data
  file:
    dest: "{{ consul_data_dir }}/data"
    state: directory

- name: install python-consul
  pip:
    name: python-consul
    state: latest

- name: configure consul as server
  include: server.yml
  when: consul_role  == 'server'

- name: configure consul as agent
  include: agent.yml
  when: consul_role  == 'agent'

- name: add consul to resolv.conf
  blockinfile:
    dest: /etc/resolv.conf
    block: |
      nameserver {{ consul_bind_address }}
      search node.consul
    insertbefore: BOF
    marker: '# {mark} consul dns // ansible managed block'

- include: firewalld.yml
  tags: firewalld
```

Figure 29. Main playbook

Main.yml exists in the default directory including consul_env for both servers and agents, consul ports and all the variables mentioned earlier. Consul env can be created by using consul_env variable. After starting env file server and agent is defined. This way docker_container knows which to use.

```
consul_env:
  server:
     SERVICE_NAME: cp-consulserver
```

Consul_serverlist contains a list of all the servers created from higher level variables. With consul server's addresses extracted from the consul_servers group with consul_client_addresses, it gives it a list of Consul servers. Hostvars is used to include all the known hosts from hosts file. With map and extract, addresses can be chosen

from the pool. Bootstrap uses the serverlist to get the length of the list, returning a number that tells the servers how many masters to expect. Consul bind address is the same as consul client address, and client address is the default IPv4 address defined higher up. Datacenter, rack, data directory and UI are given values in this file (Figure 30).

```
---
consul_env:
  server:
    SERVICE_NAME: cp-consulserver
    SERVICE_IP: "{{ ansible_default_ipv4.address }}"
  agent:
    SERVICE_NAME: cp-consulagent
    SERVICE_IP: "{{ ansible_default_ipv4.address }}"

consul_ports:
  - 8500:8500
  - 8543:8543
  - 8300:8300
  - 53:53
  - 8301:8301
  - 8302:8302
  - 8300:8300

consul_serverlist: "{{ groups['consul_servers'] | map('extract', hostvars, 'consul_client_address') | list }}"
bootstrap_expect: "{{ consul_serverlist | length }}"
consul_bind_address: "{{ consul_client_address }}"
consul_client_address: "{{ ansible_default_ipv4.address }}"
consul_datacenter: "dc1"
consul_node_rack: "rack1"
consul_data_dir: "{{ default_path }}/consul"
consul_ui_enabled: "true"
```

Figure 30. Default variables

The only thing left for Consul is to get a definition of its firewall ports and protocols. This takes place in the vars directory that has a file main.yml created inside it. In this file, firewalld_services are defined with the name of the service, ports and protocols (Figure 31).

```
firewalld_services:
    - name: consul
      ports:
        - port:
        - protocol:
```

Ports used are standard Consul ports.

```
---
firewalld_services:
  - name: consul
    ports:
      - port: 8300
        protocol: tcp
      - port: 8301
        protocol: tcp
      - port: 8301
        protocol: udp
      - port: 8302
        protocol: tcp
      - port: 8302
        protocol: udp
      - port: 8500
        protocol: tcp
      - port: 8600
        protocol: tcp
      - port: 8600
        protocol: udp
```

Figure 31. Firewall ports

After all playbooks are done and configurations set, the Consul role is ready. It can now be added to roles with Docker.

7.4 Dnsmasq configuration

Dnsmasq is included in the docker-consul role. This is because Dnsmasq is always installed and configured with Consul. First, a configuration file is to be created with the name of 10-consul.j2. This file enables forward lookup of the Consul domain. The server is to be defined as consul and the address listed as dnsmasq_listen_address while the default is client address (Figure 32).

```
# Enable forward lookup of the 'consul' domain:
server=/consul/{{ consul_client_address }}#8600
listen-address={{ dnsmasq_listen_address | default(consul_client_address) }}
bind-interfaces
~
~
```

Figure 32. Dnsmasq configuration

In tasks, Dnsmasq has its own playbook, which installs Dnsmasq with yum, sets the Consul configuration in place, adds Consul to resolv.conf and adds node.consul to

search domains. Yum install works like other roles. Adding a line to a file works with lineinfile – command. Insertinbefore is used to insert before regular expression. Regexp is used to look in every line and replace the last line found. Line is for the line to insert.

```
-   install dnsmasq
    yum:
      name: dnsmasq
      state: latest
    tags: install
    notify: restart dnsmasq


-   name: Add node.consul to search domains
    lineinfile:
      dest: /etc/resolv.conf
      state: present
      insertbefore:
      regexp:
      line:
    tags: configure
```

With this measure, Consul can be seen as a nameservice by services. Playbook also starts and enables Dnsmasq (Figure 33).

```
---
- name: Install dnsmasq
  yum:
    name: dnsmasq
    state: latest
  tags: install
  notify: restart dnsmasq

- name: Apply Consul configuration to dnsmasq
  template:
    src: templates/10-consul.j2
    dest: /etc/dnsmasq.d/10-consul
  notify: restart dnsmasq
  tags: configure

- name: Add Consul to resolv.conf
  lineinfile:
    dest: /etc/resolv.conf
    state: present
    insertbefore: "nameserver {{ ansible_dns['nameservers'][0] | default('') }}"
    regexp: '^nameserver {{ dnsmasq_listen_address }}$'
    line: 'nameserver {{ dnsmasq_listen_address }}'
  tags: configure

- name: Add node.consul to search domains
  lineinfile:
    dest: /etc/resolv.conf
    regexp: '^search.*$'
    line: "search {{ ansible_dns['search'] | default([]) | union(['node.consul']) | unique | join(' ') }}"
    state: present
  tags: configure

- name: Start and enable Dnsmasq
  systemd:
    name: dnsmasq
    state: started
    enabled: yes
```

Figure 33. Dnsmasq playbook

Firewall rules are also added to vars/main.yml. Default ports are used for Dnsmasq (Figure 34).

```
---
firewalld_services:
  - name: dnsmasq
    ports:
      - port: 53
        protocol: tcp
      - port: 53
        protocol: udp

  - name: consul
    ports:
      - port: 8300
        protocol: tcp
      - port: 8301
        protocol: tcp
      - port: 8301
        protocol: udp
      - port: 8302
        protocol: tcp
      - port: 8302
        protocol: udp
      - port: 8500
        protocol: tcp
      - port: 8600
        protocol: tcp
      - port: 8600
        protocol: udp
```

Figure 34. Dnsmasq and Consul ports

After that dnsmasq.yml is included into main.yml in tasks so it can be installed with any Consul.

7.5 Registrator configuration

Docker-registrator is a readymade role by Qvantel that has been used with the earlier versions of Consul. As there are now newer versions of Registrator available, this role has been left untouched. The role is still built the same way as the rest of them including all the same directories and playbooks.

7.6 Mesos configuration

Mesos role is a readymade role by Qvantel. It is included because it is needed for testing and; it also works well to demonstrate how Ansible works with host installations. Mesos's role is built in the same way as most of the roles, with defaults, handlers, meta, tasks, templates, tests and vars directories in the role. Installing with yum or any other package management tool with Ansible is simple. A task is created for installing Mesos that is executed by giving it a package name and version and the state it is in. In this case present as it is wanted to be present on the machine and if not Ansible should install it. Mesos has two roles to fill: master and slave. The idea is the same as with Consul. Playbook gets the variables for both master and slave servers and chooses from them depending on the target host. Consul.yml handels registering Mesos service to Consul (Figure 35).

```
---
- include: firewalld.yml
  tags: firewalld

- name: Install Mesos
  yum:
    name: mesos{{ '-' + mesos_version }}
    state: present
  notify:
    - restart mesos master
    - restart mesos slave
  tags: install

- name: Apply Zookeeper configuration
  template:
    src: templates/mesos/zk.j2
    dest: /etc/mesos/zk
  notify:
    - restart mesos master
    - restart mesos slave
  tags: configure

- name: Apply Mesos environment variables
  template:
    src: templates/mesos/mesos.j2
    dest: /etc/default/mesos
  notify:
    - restart mesos master
    - restart mesos slave
  tags: configure

- include: mesos_{{ mesos_role }}.yml

- include: consul.yml
~
```

Figure 35. Mesos main.yml playbook

7.7 Marathon configuration

Marathon's configuration is located in the template directory. Firewalld_service.j2 is located there also and it is the same as Consul's firewall configuration. In the configuration, Marathon will have Zookeeper servers defined, so that it can connect to Mesos by them. Marathon will also use Zookeeper to choose a leading master when there are more than one Marathon.

MARATHON_MASTER=zk:// zookeeper_serverlist /2181/mesos

MARATHON_ZK=zk:// zookeeper_serverlist /2181/mesos

MARATHON_HTTP_ADDRESS= listen address

MARATHON ADDRESS= listen address

This leading master has its framework used by Mesos. In addition, addresses are defined with variables in the configuration except for libprocess port (Figure 36).

```
MARATHON_MASTER=zk://{{ zookeeper_serverlist | map('regex_replace', '$', ':2181') | join(',') }}/mesos
MARATHON_ZK=zk://{{ zookeeper_serverlist | map('regex_replace', '$', ':2181') | join(',') }}/marathon
MARATHON_LOGGING_LEVEL={{ log_level }}
LIBPROCESS_PORT=8585
LIBPROCESS_IP={{ marathon_listen_address }}
MARATHON_HTTP_ADDRESS={{ marathon_listen_address }}
```

Figure 36. Marathon configuration

The tasks have three playbooks, marathon.yml, firewalld.yml and main.yml. In marathon.yml Marathon is installed and its configuration is moved to a correct directory. The installation task is almost the same as Consul's. Figure 37 displays marathon.yml playbook and tasks in it.

```
---
- name: Apply Marathon environment variables
  template:
    src: templates/marathon/marathon.j2
    dest: "{{ marathon_conf_path }}"
  tags: configure

- name: start marathon
  docker_container:
    name: marathon
    image: "{{ dockerregistry_url | default('library/') }}cp-marathon:{{ marathon_version }}"
    log_driver: json-file
    log_options:
      max-size: 25m
      max-file: "3"
    network_mode: host
    purge_networks: true
    env_file: "{{ marathon_conf_path }}"
    restart_policy: unless-stopped
    state: started
    restart: yes
```

Figure 37. Marathon installation playbook

The main.yml includes both marathon.yml and firewalld.yml. The defaults have main.yml file to be used for defining variables. Listen address, log level and configuration path are both defined here (Figure 38).

```
---
marathon_listen_address: "{{ ansible_default_ipv4['address'] }}"
marathon_log_level: "ERROR"
marathon_conf_path: "/etc/default/marathon"
```

Figure 38. Default settings

Vars includes firewall service name, ports and protocols (Figure 39).

```
---
firewalld_services:
  - name: marathon
    ports:
      - port: 8080
        protocol: tcp
      - port: 8443
        protocol: tcp
      - port: 8585
        protocol: tcp
```

Figure 39. Firewall service for Marathon.

7.8 Zookeeper configuration

Zookeeper’ configuration is located in templates directory. The directory has both firewalld.yml and zoo.cfg.j2. Firewall configuration is the same as in other roles. In zoo.cfg.j2 Zookeeper has its client limit, tick time, init limit and sync limit specified. Data directory, client port, and port address are also defined. The configuration also has a setting for Zookeeper hosts that set the Zookeeper server id with the correct address (Figure 40).

```
server.1=192.168.81.60
server.2=192.168.81.62
server.3=192.168.81.64
```

```
maxClientCnxns=50
tickTime=2000
initLimit=30
syncLimit=10
dataDir=/data
clientPort=2181
{% endif %}
clientPortAddress={{ zookeeper_listen_address }}
{% for host in groups['zookeeper_servers'] %}
server.{{ hostvars[host]['zookeeper_id'] }}={{ hostvars[host]['zookeeper_listen_address'] }}:2888:3888
{% endfor %}
```

Figure 40. Zookeeper configuration

In tasks, zookeeper.yml playbook creates the correct data paths with zookeeper users, adds the configurations there, creates a myid file from zookeeper_id that is later specified and also creates the container. Zookeeper container is created like Consul and Marathon (Figure 41).

```
- name: Create zookeeper container
  docker_container:
    name: zookeeper
    image: "{{ dockerregistry_url | default(None) }}cp-zookeeper:{{ zookeeper_version }}"
    volumes:
      - "{{ zookeeper_data_path }}/conf/zoo.cfg:/conf/zoo.cfg"
      - "{{ zookeeper_data_path }}/data:/data"
      - "{{ zookeeper_data_path }}/datalog:/datalog"
    network_mode: host
    log_driver: json-file
    log_options:
      max-size: 25m
      max-file: "3"
    restart_policy: unless-stopped
    state: started
```

Figure 41. Zookeeper playbook

Firewalls playbook is the same as previously, using 2181 as service port and main.yml, combining those two playbooks with include.

7.9 QVCP role

QVCP is also a readymade role by Qvantel that sets all the needed repositories and keys to the host. This is always done as a pre-task when installing services to a new host. Figure 42 displays some of the used repositories.

```
---
- name: add artifactory cp repository
  yum_repository:
    name: qvcp
    description: Qvantel CP repository
    baseurl: https://artifactory.qvantel.net/artifactory/yum-cp/
    enabled: yes
    gpgcheck: no
- name: add artifactory cp stockpile repository
  yum_repository:
    name: qvcp-stockpile
    description: Qvantel CP Stockpile
    baseurl: https://artifactory.qvantel.net/artifactory/yum-cp-stockpile/
    enabled: yes
    gpgcheck: no
- name: add artifactory epel mirror
  yum_repository:
    name: epel
    description: Qvantel EPEL mirror
    baseurl: https://artifactory.qvantel.net/artifactory/epel/7/x86_64
    enabled: yes
    gpgcheck: yes
    gpgkey: https://artifactory.qvantel.net/artifactory/epel/RPM-GPG-KEY-EPEL-7
```

Figure 42. Repositories included in QVCP role

7.10 Ansible setup for installation

Ansible is meant to do all the installations and configuration which means Ansible must have playbooks for both master and slave nodes. These playbooks must install all roles and make sure the hosts have all needed repositories, packages and information needed by the roles.

7.10.1 Roles

Roles include all the roles created or discussed earlier. These roles are located inside a directory called roles (Figure 43).

```
drwxr-xr-x.  9 avalkeinen staff 4096 Aug  3 13:16 docker
drwxr-xr-x.  7 avalkeinen staff 4096 Aug  3 16:16 docker-consul
drwxr-xr-x.  7 avalkeinen staff 4096 Aug  3 22:44 docker-marathon
drwxr-xr-x.  8 avalkeinen staff 4096 Aug  3 23:34 docker-registrator
drwxr-xr-x.  7 avalkeinen staff 4096 Aug  3 23:08 docker-zookeeper
drwxr-xr-x. 10 avalkeinen staff 4096 Aug  3 13:16 mesos
drwxr-xr-x.  8 avalkeinen staff 4096 Aug  3 13:16 qvcp-common
[root@cp00 roles]#
```

Figure 43. Roles in directory roles

7.10.2 Variables

Variables are located in three different locations. Inventory/hosts, group_vars/all and extra_vars/env_variables. Hosts file has master and agent servers specified under their names. Master servers also have their zookeeper_id variables placed after their address (Figure 44).

```
[master]
cp00.lab.qvantel.net zookeeper_id=1
cp02.lab.qvantel.net zookeeper_id=2
cp04.lab.qvantel.net zookeeper_id=3
[agent]
cp06.lab.qvantel.net
cp08.lab.qvantel.net
cp10.lab.qvantel.net
cp12.lab.qvantel.net
cp14.lab.qvantel.net
~
```

Figure 44. Hosts file

Group_vars has addresses, docker registry, consul scheme and server lists specified. Docker registry points where the docker images are located, in this case, Qvantel's own registry. Listen addresses use Ansible's default address. Server lists creates a list out of Zookeeper servers and their addresses.

> Zookeeper_serverlist: "{{ groups['zookeeper_servers'] | map('extract', host-vars, 'zookeeper_listen_address') |list }}'

It uses zookeeper_servers with groups and then extracts addresses used to form hosts with hostvars and creates a list. The same is done with Mesos. Package install is a variable for offline installation but is not used in this environment. Figure 45 displays settings in group_vars.

```
dockerregistry_url: "artifactory.qvantel.net/"

zookeeper_listen_address: "{{ ansible_default_ipv4.address }}"
mesos_listen_address: "{{ ansible_default_ipv4.address }}"
marathon_listen_address: "{{ ansible_default_ipv4.address }}"
consul_scheme: "http"
zookeeper_serverlist: "{{ groups['zookeeper_servers'] | map('extract', hostvars, 'zookeeper_listen_address') | list }}"
mesos_serverlist: "{{ groups['mesos_masters'] | map('extract', hostvars, 'mesos_listen_address') | list }}"
package_install: online
```

Figure 45. Group_vars settings

Extra_vars contains extra variables, including default path for directories and logging level for mesos-master and marathon (Figure 46).

```
default_path: "/usr/share"
# this defines log level for mesos-master and marathon
log_level: "ERROR"
```

Figure 46. Extra_vars

For containers to work, users are created with playbooks in users directory. These create a group for services and add users to those groups. There are two separate playbooks, one for master and one for agent hosts. These can be found in Appendices 1 and 2.

7.10.3 Other settings

Package versions are described in vars/versions.yml (Figure 47).

```
consul_version: "1.1.0"
dockerce_version: "18.03.0.ce"
dockerpy_version: "1.10.6"
marathon_version: "v1.5.8"
mesos_version: "1.5.0"
registrator_version: "master"
zookeeper_version: "3.4.12"
```

Figure 47. Package versions

Ansible.cfg is the main Ansible configuration file. This file can be used to set different paths, user and host settings. In this case, the settings are basic. Host key checking is set to false for SSH. Callback_whitelist helps to make playbook runs more informative with date and time. Roles_path tells Ansible which roles to use and hostfile where to find host addresses. In this case, tty is not required by sudo so pipelining is true. Figure 48 displays used Ansible configuration.

```
[defaults]
host_key_checking=False
callback_whitelist = profile_tasks
roles_path = ./roles
hostfile = ./inventory/hosts

pipelining=True
~
cp12.lab.qvantel.net
cp14.lab.qvantel.net
```

Figure 48. Ansible configuration

7.10.4 Playbook

Playbooks are created for both master and agent. Both playbooks are the same except for their names and roles. Masters playbook first sets up hosts. Here it will take master nodes from inventory/hosts file and role versions from vars/versions.yml. With become: yes means it will become sudo as it is the default option. Pre-tasks include tasks done before roles. These include adding extra_vars variables to be used, adding hosts to Zookeeper servers, Consul masters and Mesos masters. This will occur if it is not currently so, defined by changed_when. Playbook ensures firewalld is installed and includes roles qvcp-common, docker and master_users.yml. This is similar in agent playbook as well. After pre-tasks, Ansible installs the specified roles. The roles include Consul, Registrator, Zookeeper, Mesos and Marathon. Consul and Mesos have additional settings of consul_role and mesos_role for what role to install, master or agent/slave (Figure 49). Their own playbooks use this setting to install the wanted version. Master playbook can be found in Appendix 3.

```
roles:
  - { role: docker-consul, consul_role: 'server' }
  - { role: docker-registrator }
  - { role: docker-zookeeper }
  - { role: mesos, mesos_role: 'master' }
  - { role: docker-marathon }
```

Figure 49. Included roles

Agents playbook is similar with only names changing and agent getting facts from master nodes. This will make it so that agent playbook if accessible to master addresses from the inventory/hosts. In addition, there are only three roles, Consul, Mesos and Registrator with Consul and Mesos roles being agent/slave.

Slave playbook can be found in Appendix 4.

As playbooks are ready and roles are set, the installation can start.

8 Installation

Installation is performed by entering command

> [root@cp00 consul-thesis]# ansible-playbook -u avalkeinen -k master.yml

in consul-thesis/ directory. Under this directory lie all the Ansible playbooks, variables and roles. Ansible asks for password because of the -k option. After entering password, the master.yml playbook runs (Figure 50).

```
PLAY [set up master nodes] ******************************************************************

TASK [Gathering Facts] **********************************************************************
Saturday 04 August 2018  23:39:14 +0300 (0:00:00.104)       0:00:00.104 *******
ok: [cp04.lab.qvantel.net]
ok: [cp02.lab.qvantel.net]
ok: [cp00.lab.qvantel.net]

TASK [include instance-specific extra variables] ********************************************
Saturday 04 August 2018  23:39:17 +0300 (0:00:02.241)       0:00:02.345 *******
ok: [cp00.lab.qvantel.net] => (item=/home/avalkeinen/consul-thesis/extra_vars/env_variables)
ok: [cp02.lab.qvantel.net] => (item=/home/avalkeinen/consul-thesis/extra_vars/env_variables)
ok: [cp04.lab.qvantel.net] => (item=/home/avalkeinen/consul-thesis/extra_vars/env_variables)

TASK [add mesos_masters, zookeeper_servers, consul_servers] *********************************
Saturday 04 August 2018  23:39:17 +0300 (0:00:00.186)       0:00:02.532 *******
ok: [cp00.lab.qvantel.net] => (item=cp00.lab.qvantel.net)
ok: [cp00.lab.qvantel.net] => (item=cp02.lab.qvantel.net)
ok: [cp00.lab.qvantel.net] => (item=cp04.lab.qvantel.net)

TASK [ensure firewalld is installed] ********************************************************
Saturday 04 August 2018  23:39:17 +0300 (0:00:00.099)       0:00:02.631 *******
ok: [cp04.lab.qvantel.net]
ok: [cp02.lab.qvantel.net]
ok: [cp00.lab.qvantel.net]

TASK [include_role : qvcp-common] ***********************************************************
Saturday 04 August 2018  23:39:18 +0300 (0:00:01.190)       0:00:03.822 *******
```

Figure 50. Playbook run

After running the master playbook, Ansible shows play recap and information on how long it took (Figure 51).

```
PLAY RECAP ********************************************************************************
cp00.lab.qvantel.net       : ok=80   changed=64   unreachable=0    failed=0
cp02.lab.qvantel.net       : ok=79   changed=64   unreachable=0    failed=0
cp04.lab.qvantel.net       : ok=79   changed=64   unreachable=0    failed=0

Saturday 04 August 2018  23:46:13 +0300 (0:00:00.405)       0:06:58.337 *******
===============================================================================
```

Figure 51. Playbook installation recap

At this point, Consul can be accessed with its hostname or IP and using its port 8500. Figure 52 displays default Consul UI with registered services.

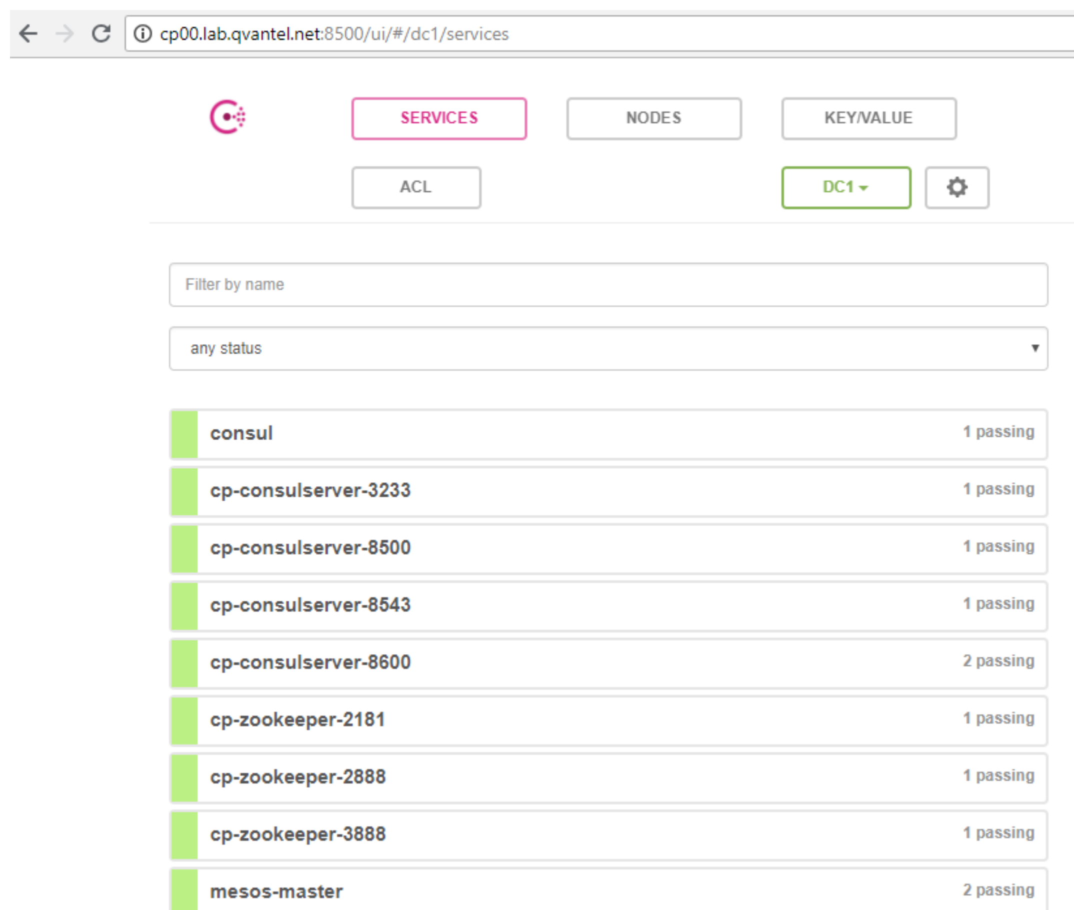

Figure 52. Registered services

With this, it can be seen that the services are healthy and installed. Running the agent playbook works with the same command.

[root@cp00 consul-thesis]# ansible-playbook -u avalkeinen agent.yml -k

After installation, it can be seen in the Ansible recap that everything was installed properly (Figure 53).

```
PLAY RECAP *********************************************************************
cp00.lab.qvantel.net       : ok=1    changed=0    unreachable=0    failed=0
cp02.lab.qvantel.net       : ok=1    changed=0    unreachable=0    failed=0
cp04.lab.qvantel.net       : ok=1    changed=0    unreachable=0    failed=0
cp06.lab.qvantel.net       : ok=63   changed=43   unreachable=0    failed=0
cp08.lab.qvantel.net       : ok=62   changed=43   unreachable=0    failed=0
cp10.lab.qvantel.net       : ok=62   changed=43   unreachable=0    failed=0
cp12.lab.qvantel.net       : ok=62   changed=43   unreachable=0    failed=0
cp14.lab.qvantel.net       : ok=62   changed=43   unreachable=0    failed=0

Saturday 04 August 2018  23:58:36 +0300 (0:00:01.003)       0:04:43.791 *******
===============================================================================
```

Figure 53. Play recap

Master nodes are included because agent.yml gets facts from them. In Consul's UI can be seen that everything was installed and is answering to the healthchecks (Figure 54).

cp00.lab.qvantel.net:8500/ui/#/dc1/services

SERVICES NODES

Filter by name | any status | EXPAND

Service	Status
consul	3 passing
cp-consulagent-3233	5 passing
cp-consulagent-8500	5 passing
cp-consulagent-8543	5 passing
cp-consulagent-8600	10 passing
cp-consulserver-3233	3 passing
cp-consulserver-8500	3 passing
cp-consulserver-8543	3 passing
cp-consulserver-8600	6 passing
cp-zookeeper-2181	1 passing
cp-zookeeper-2888	1 passing
cp-zookeeper-3888	1 passing
marathon	6 passing
marathon_exporter-9098	3 passing
marathon_exporter-9198	3 passing
mesos-master	6 passing
mesos-slave	10 passing
zookeeper	6 passing

Figure 54. Consul UI and running services

In addition, Mesos UI can be accessed from one of its server's IPs and in port 5050. Figure 55 displays Mesos UI with connected slaves.

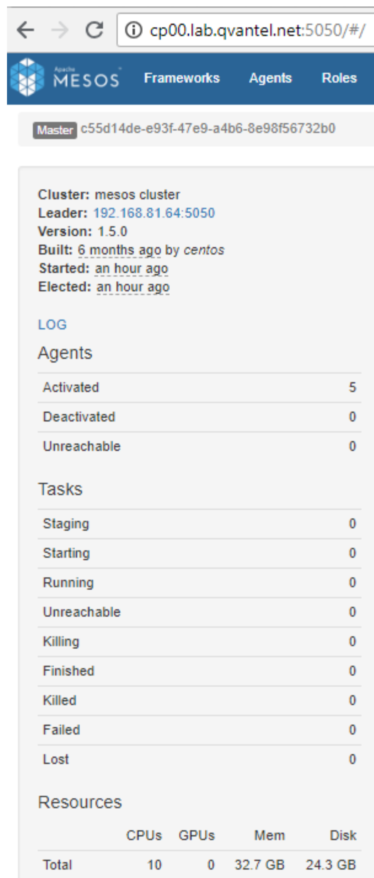

Figure 55. Mesos UI

Marathon UI can be accessed on port 8080. With this, it can be confirmed that at least UIs are working.

9 Testing

Creating a container with Marathon can be done either manually or with a ready container. In this case, a ready container was used. The container sends randomly created "tweets" from a certain President as its stdout messages. The container is created by going to Marathon's UI and selecting create application. There the container is named Thesis in general options and the image path is entered in Docker container options. Figure 56 shows the creation of a Marathon application with a ready image.

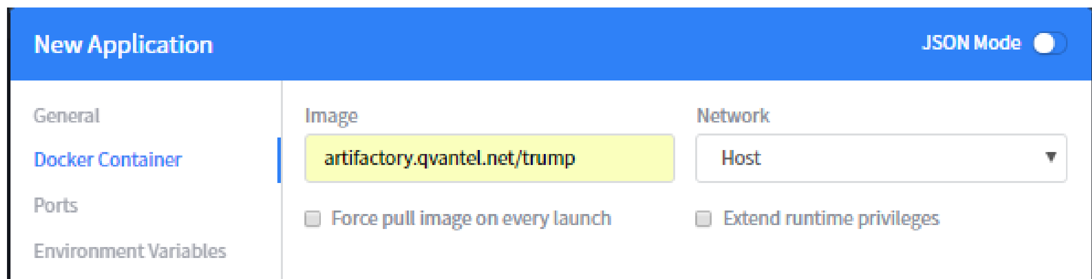

Figure 56. Marathon using a ready image from a repository

The image is pulled from Qvantel's own Docker registry. After creating the application, Consul's connectivity is tested in agent server cp10. Here consul.service.consul address is used to ping, traceroute and nslookup within the Thesis container running on top of Marathon. After deploying, it can be seen in the Marathon's application list. If status shows running, it means it is up and running (Figure 57).

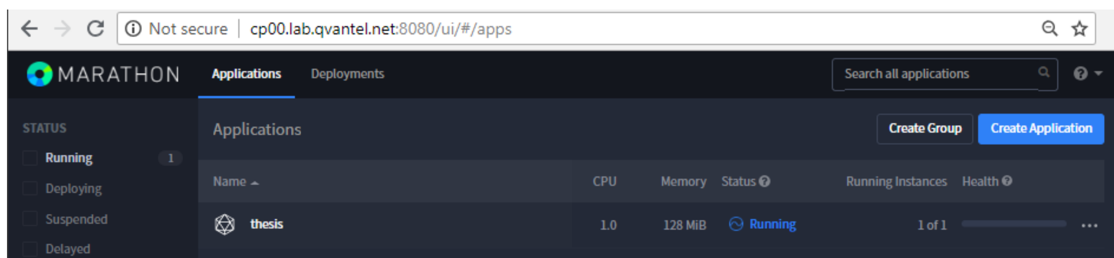

Figure 57. Running container in Marathon UI

In addition, in Mesos's UI can be seen that the stdouts is sending data (Figure 58).

{"@version":1,"@timestamp":"2018-08-04T22:27:55.783+00:00","log_level":"INFO","level_value":20000,"message":"We will begin working man for a temporary visas to open borders.","logger_name":"SAD","thread_name":"logging"}
{"@version":1,"@timestamp":"2018-08-04T22:27:56.783+00:00","log_level":"INFO","level_value":20000,"message":"We will be honest about it back to get the nuclear weapons.","logger_name":"SAD","thread_name":"logging"}
{"@version":1,"@timestamp":"2018-08-04T22:27:57.783+00:00","log_level":"INFO","level_value":20000,"message":"I ever realized.","logger_name":"SAD","thread_name":"logging"}
{"@version":1,"@timestamp":"2018-08-04T22:27:58.784+00:00","log_level":"INFO","level_value":20000,"message":"And God bless you.","logger_name":"SAD","thread_name":"logging"}
{"@version":1,"@timestamp":"2018-08-04T22:27:59.784+00:00","log_level":"INFO","level_value":20000,"message":"So important.","logger_name":"SAD","thread_name":"logging"}

Figure 58. Application stdout

In Consul's UI it can be seen that the container has been registered to Consul with Registrator (Figure 59).

mesos-master	6 passing
mesos-slave	10 passing
trump	1 passing
zookeeper	6 passing

Figure 59. Automatically registered tweet container

In Mesos it can also be seen on which machine it is running. In this case, 192.168.81.70 that is cp10 (Figure 60).

Active Tasks

Framework ID	Task ID	Task Name	Role	State	Health	Started ▼	Host	
... 8e98f56732b0-0000	thesis.72f9eb34-9835-11e8-a364-4a8de7a48f9c	thesis	*	RUNNING	-	4 minutes ago	192.168.81.70	Sandbox

Figure 60. Active tasks

With the command

[root@cp10 avalkeinen]# docker ps

it can be seen that the container is running there (Figure 61).

```
[root@cp10 avalkeinen]# docker ps
CONTAINER ID        IMAGE                                        COMMAND                  CREATED             STATUS              PORTS
    NAMES
1bffb6f70494        artifactory.qvantel.net/trump                "/bin/sh -c /bin/tru…"   4 minutes ago       Up 4 minutes
    mesos-9f8bafac-899a-48c8-9310-9bf78d4b7ffb
ccc4951c0bcf        artifactory.qvantel.net/cp-consul:1.1.0      "docker-entrypoint.s…"   30 minutes ago      Up 30 minutes
    consul-agent
9a11b96e2aab        artifactory.qvantel.net/cp-registrator:master   "/bin/registrator -i…"   2 hours ago         Up 2 hours
    registrator
```

Figure 61. Docker ps on cp10

With the command

[root@cp10 avalkeinen]# docker exec -it 1bf sh

container can be accessed. The connectivity can be tested in Consul first with ping command. Figure 62 displays both docker exec command and ping to Consul.

```
[root@cp10 avalkeinen]# docker exec -it 1bf sh
/ # ping consul.service.consul
PING consul.service.consul (192.168.81.64): 56 data bytes
64 bytes from 192.168.81.64: seq=0 ttl=64 time=1.326 ms
64 bytes from 192.168.81.64: seq=1 ttl=64 time=0.250 ms
64 bytes from 192.168.81.64: seq=2 ttl=64 time=0.376 ms
64 bytes from 192.168.81.64: seq=3 ttl=64 time=0.295 ms
^C
--- consul.service.consul ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.250/0.561/1.326 ms
```

Figure 62. Exec and ping

It can be seen that the server it connects to is 192.168.81.64, the current leading master. This can be seen in Docker's Consul logs (Figure 63).

```
    2018/08/04 21:51:32 [INFO] serf: EventMemberJoin: cp04.dc1 192.168.81.64
    2018/08/04 21:51:32 [INFO] consul: Handled member-join event for server "cp04.dc1" in area "wan"
2018/08/04 21:51:37 [DEBUG] raft-net: 192.168.81.60:8300 accepted connection from: 192.168.81.64:52682
    2018/08/04 21:51:37 [INFO] consul: New leader elected: cp04
    2018/08/04 21:51:38 [INFO] agent: Synced service "cp00:consul-server:8543"
```

Figure 63. Docker logs

After ping, traceroute command is used inside the container (Figure 64).

```
/ # traceroute consul.service.consul
traceroute to consul.service.consul (192.168.81.64), 30 hops max, 46 byte packets
 1  192.168.81.64 (192.168.81.64)  0.386 ms !C  0.384 ms !C  0.286 ms !C
```

Figure 64. Traceroute from the container

Nothing much there. With nslookup, the servers with that name can be seen (Figure 65).

```
Name:      consul.service.consul
Address 1: 192.168.81.64
Address 2: 192.168.81.62
Address 3: 192.168.81.60
```

Figure 65. Nslookup from the container

Consul has its own commands that can be used within the container. Consul container is accessed as the thesis container. From inside the container, the command

/ # consul members --http-addr=http://192.168.81.60:8500

can be used. The members command shows all Consuls connected to master servers. HTTP command makes the command to use this specific server to query the information. With the command, Consul prints its members (Figure 66).

```
/ # consul members --http-addr=http://192.168.81.60:8500
Node  Address              Status  Type    Build  Protocol  DC   Segment
cp00  192.168.81.60:8301   alive   server  1.1.0  2         dc1  <all>
cp02  192.168.81.62:8301   alive   server  1.1.0  2         dc1  <all>
cp04  192.168.81.64:8301   alive   server  1.1.0  2         dc1  <all>
cp06  192.168.81.66:8301   alive   client  1.1.0  2         dc1  <default>
cp08  192.168.81.68:8301   alive   client  1.1.0  2         dc1  <default>
cp10  192.168.81.70:8301   alive   client  1.1.0  2         dc1  <default>
cp12  192.168.81.72:8301   alive   client  1.1.0  2         dc1  <default>
cp14  192.168.81.74:8301   alive   client  1.1.0  2         dc1  <default>
/ #
```

Figure 66. Consul members

Docker containers can also be inspected with the command:

[root@cp00 consul-thesis]# docker inspect consul-server

This prints out the configuration, name, mount paths, networking and other information about the container. Consul, Marathon and Zookeeper inspects can be found from Appendices 5, 6 and 7.

Consul can also be used as a key/value store. This can hold for example dynamic configurations. When application starts it will fetch configuration from Consul's KV storage. Because configuration is copied from Consul it can be changed in Consul UI and this will be updated to the running application. This is the primary reason for Consul in addition of service discovery and health checking. Figure 67 presents Consul KV with simple configuration for Nginx.

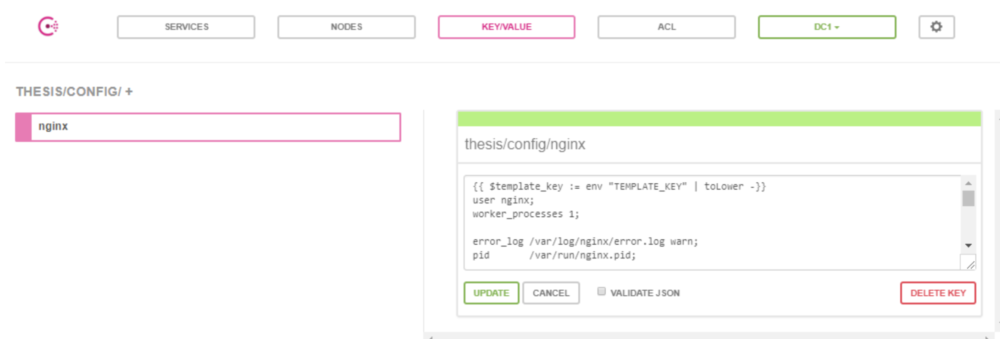

Figure 67. KV storage in Consul

Consul configuration is supposed to use /usr/share/consul/ datapath to store configurations and data. Figure 68 displays path, directories in /usr/share/consul/ and configuration set by Ansible Consul template.

```
[root@cp00 consul]# pwd
/usr/share/consul
[root@cp00 consul]# ll
total 8
drwxr-xr-x. 2 root root      4096 Aug 12 19:28 config
drwxr-xr-x. 2  100 aaa_admin 4096 Aug  5 02:18 data
[root@cp00 consul]# cat config/server.json
{
  "datacenter": "dc1",
  "node_meta": {
    "Rack": "rack1"
  },
  "bind_addr": "192.168.81.60",
  "client_addr": "192.168.81.60",
  "retry_join": ["192.168.81.60"],
  "retry_join_wan": ["192.168.81.62", "192.168.81.64"],
  "recursors": ["8.8.8.8", "8.8.4.4"],
    "server": true,
  "bootstrap_expect": 1,
  "data_dir": "/usr/share/consul",
  "ui": true,
  "log_level": err
}
[root@cp00 consul]#
```

Figure 68. Consul configuration set by Ansible

With mounting /usr/share/consul/ with the Consul container, same files can be found inside the Consul container (FIngure 69).

```
/etc/consul # cat config/server.json
{
  "datacenter": "dc1",
  "node_meta": {
    "Rack": "rack1"
  },
  "bind_addr": "192.168.81.60",
  "client_addr": "192.168.81.60",
  "retry_join": ["192.168.81.60"],
  "retry_join_wan": ["192.168.81.62", "192.168.81.64"],
  "recursors": ["8.8.8.8", "8.8.4.4"],
    "server": true,
  "bootstrap_expect": 1,
  "data_dir": "/usr/share/consul",
  "ui": true,
  "log_level": err
}
```

Figure 69. Mounted configuration in Consul container

10 Results

The goal of the thesis was to create an automated DNS service installation and implement it to the platform. This was achieved by creating new roles for Consul and including Dnsmasq to that role. Additionally, Marathon and Zookeeper were moved to containers and given new roles. A new Ansible installation was created to install new services to testing environment. The testing environment was used to see if new containers worked as intended, alone and together as a platform. Consul was tested by using common networking commands to see if the name service was working correctly. New services were registered by creating an application container on top of Marathon and seeing if Consul would register new containers with the help of Registrator, which also tests if Marathon and Zookeeper containers work properly. All tests worked as intended. Hence, it can be concluded that Consul and other roles are working and can be delivered to be as part of the Qvantel's platform.

11 Conclusons

The research question was to create an automated DNS service installation and implement it to the platform. This was achieved by successfully creating a new Ansible role for Consul that could be integrated to Qvantel's platform services. Consul's role was first theorized, planned and then implemented in practice. During the creation of Consul role, also Marathon and Zookeeper were developed; however, the thesis did not focus on them. Dnsmasq was integrated into the same role as Consul as it is always installed on same host as Consul.

Constructive research was chosen as the method of research for this thesis. Constructive research was chosen because the required results were known; however, the practice was not. Research methodology was chosen correctly, as the thesis focused on developing something new from something that was already known and existed in a different form.

The thesis turned out to be fairly comprehensive look into Consul, Ansible, Docker and Mesos. The study could have been completed more straightforwardly, skipping extra components and focusing solely on Consul and Ansible. Nevertheless, as Consul

is wanted as a part of a larger system, this was still a good way to see how it would work with multiple components and in a real environment. Multiple datacenter connectivity was left out of Consul as there was no time to test it properly. This was one feature that was left out and will be added at a later date, when Consul is integrated into Qvantel's systems. For now, Consul datacenters can be connected manually with one command.

It was easy to find theory on different components; however, the planning and development was mostly carried out using official documentation for each component. As books usually focus on a specific way of implementing a service, they do not provide much help. Every company has their own way of doing things, and something written can never be implemented easily. While making different roles, the support on how to do them, how they work and what is needed was mostly gained orally from colleagues. This was not documented as there are no interviews or physical documentation of this knowledge.

Creating simple Ansible roles teaches not only how to configure and use Ansible but how to configure different services and how to troubleshoot them. As installation is trivialized, most issues come after installing or upgrading those services. This thesis and the rest of the development work done outside of this research has increased my personal knowledge of open source software. Even if Consul role will go through some changes after being integrated with rest of the roles, this thesis has still helped Qvantel to reach their goal of creating an automated platform installation.

References

Amazon letter to shareholders. 2015. Accessed on 11.8.2018. Retrieved from http://phx.corporate-ir.net/phoenix.zhtml?c=97664&p=irol-SECText&TEXT=aHR0cDovL2FwaS50ZW5rd2l6YXJkLmNvbS9maWxpbmcueG1sP2lwYWdlPTEwODYwMjA1JkRTRVE9MCZTRVE9MCZTUURFU0M9U0VDVElPTl9FTlRJUkUmc3Vic2lkPTU3)

Ansible documentation - Playbooks. N.d. Ansible documentation. Accessed on 11.8.2018. Retrieved from https://docs.ansible.com/ansible/2.3/playbooks.html

Ansible introduction. N.d. Ansible webpage. Accessed on 11.8.2018. Retrieved from https://www.ansible.com/overview/how-ansible-works

Ansible overview. N.d. Ansible webpage. Accessed on 12.8.2018. Retrieved from https://www.ansible.com/overview/it-automation/

Consul Documentation – Architecture. N.d. Consul documentation. Accessed on 11.8.2018. Retrieved from https://www.consul.io/docs/internals/architecture.html

Consul Documentation – Consul Containers. N.d. Consul documentation. Accessed on 11.8.2018. Retrieved from https://www.consul.io/docs/guides/consul-containers.html

Consul introduction. N.d. Consul webpage. Accessed on 11.8.2018. Retrieved from https://www.consul.io/intro/index.html

D. Bernstein. 2014. "Containers and Cloud: From LXC to Docker to Kubernetes". *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81-84.

Dnsmasq documentation. N.d. Dnsmasq documentation. Accessed on 11.8.2018. Retrieved from http://www.thekelleys.org.uk/dnsmasq/doc.html/

DNS root servers. N.d. Cloudflare webpage. Accessed on 11.8.2018. Retrieved from https://www.cloudflare.com/learning/dns/glossary/dns-root-server/

Dostalek, Libor, and Alena Kabelova. 2006. "Chapter 1 - Domain Name System". DNS in Action: A Detailed and Practical Guide to DNS Implementation, Configuration, and Administration. Packt Publishing.

Dostalek, Libor, and Alena Kabelova. 2006. "Chapter 2 - DNS Protocol". DNS in Action: A Detailed and Practical Guide to DNS Implementation, Configuration, and Administration. Packt Publishing.

Docker documentation – Docker architecture. N.d. Docker overview. Accessed on 11.8.2018. Retrieved from https://docs.docker.com/engine/docker-overview/#docker-architecture

Docker commands. N.d. Docker documentation. Accessed on 11.8.2018. Retrieved from https://docs.docker.com/engine/reference/commandline/docker/#child-commands

Docker images and containers. N.d. Docker get started. Accessed on 11.8.2018. Retrieved from
https://docs.docker.com/get-started/#images-and-containers

Heap, Michael. 2016. “Chapter 1 - Getting Started”. Ansible: From Beginner to Pro. Apress.

Heap, Michael. 2016. “Chapter 4 - Ansible Roles”. Ansible: From Beginner to Pro. Apress.

Heap, Michael. 2016. “Chapter 5 - Parameterizing Playbooks”. Ansible: From Beginner to Pro. Apress.

J. O. Benson, J. J. Prevost and P. Rad. 2016. "Survey of automated software deployment for computational and engineering research". *2016 Annual IEEE Systems Conference (SysCon)*, Orlando, FL, pp. 1-6.

Schroder, Carla. “Advanced Dnsmasq Tips and Tricks”. 2018. Linux webpage. Accessed on 12.8.2018. Retrieved from
https://www.linux.com/learn/intro-to-linux/2018/2/advanced-dnsmasq-tips-and-tricks/

Mesosphere – Marathon documentation. N.d. Marathon documentation. Accessed on 11.8.2018. Retrieved from
https://mesosphere.github.io/marathon/docs/

Mesos – Why mesos, N.d. Mesos webpage. Accessed on 11.8.2018. Retrieved from
https://mesosphere.com/why-mesos/

Mesos Documentation – Architecture. N.d. Mesos documentation. Accessed on 11.8.2018. Retrieved from
http://mesos.apache.org/documentation/latest/architecture/

Oracle Linux. N.d. Oracle Linux webpage. Accessed on 11.8.2018. Retrieved from
https://www.oracle.com/linux/

Pasian, Beverly. 2018. Desingns, Methods and Practices for Research of Project Managements. Gower.

Rampling, Blair, and David Dalan. 2003. "Chapter 5 - Configuring a DNS Client". DNS for Dummies. John Wiley & Sons.

Registrator ReadMe. N.d. Registrator GitHub readme. Accessed on 11.8.2018. Retrieved from
http://gliderlabs.github.io/registrator/latest

Taylor, Adam. 2016. 47 percent of the world’s population njow use the Internet, study says. The Washington Post. Accessed on 11.8.2018. Retrieved from
https://www.washingtonpost.com/news/worldviews/wp/2016/11/22/47-percent-of-the-worlds-population-now-use-the-internet-users-study-says

Vohra, Deepak. 2016. "Chapter 1 - Hello Docker". Pro Docker. Apress.

Zookeeper Documentation – Getting started. N.d. Zookeeper documentation. Accessed on 11.8.2018. Retrieved from
https://zookeeper.apache.org/doc/current/zookeeperStarted.html

Zookeeper – Overlook. N.d. Zookeeper documentation. Accessed on 11.8.2018. Retrieved from
https://zookeeper.apache.org/doc/current/zookeeperOver.html

Appendices

Appendix 1. Master groups and users

```
---
- name: create groups for host
  group:
    name: zookeeper
    state: present

- name: Add user "zookeeper"
  user:
    name: zookeeper
    groups: zookeeper
    shell: /sbin/nologin
    append: yes
    comment: "zookeeper nologin User"
    state: present
  become: true

- name: create groups for host
  group:
    name: consul
    state: present

- name: Add user "consul"
  user:
    name: consul
    groups: consul
    shell: /sbin/nologin
    append: yes
    comment: "consul nologin User"
    state: present
  become: true

- name: create groups for host
  group:
    name: mesos
    state: present

- name: Add user "mesos"
  user:
    name: mesos
    groups: mesos
    shell: /sbin/nologin
    append: yes
    comment: "mesos nologin User"
    state: present
  become: true
```

Appendix 2. Agent groups and users

```
- name: create groups for host
  group:
    name: consul
    state: present

- name: Add user "consul"
  user:
    name: consul
    groups: consul
    shell: /sbin/nologin
    append: yes
    comment: "consul nologin User"
    state: present
  become: true

- name: create groups for host
  group:
    name: mesos
    state: present

- name: Add user "mesos"
  user:
    name: mesos
    groups: mesos
    shell: /sbin/nologin
    append: yes
    comment: "mesos nologin User"
    state: present
  become: true
```

Appendix 3. Master playbook

```
---
- name: set up master nodes
  hosts: master
  vars_files:
    - vars/versions.yml
  become: yes

  pre_tasks:
    - name: include instance-specific extra variables
      include_vars: "{{ item }}"
      with_fileglob: extra_vars/*
      tags: configure

    - name: add mesos_masters, zookeeper_servers, consul_servers
      add_host:
        groups: zookeeper_servers,consul_servers,mesos_masters
        name: "{{ item }}"
      with_items:
        - "{{ groups['master'] }}"
      changed_when: false
      tags: configure

    - name: ensure firewalld is installed
      yum:
        name: firewalld
        state: present

    - include_role:
        name: qvcp-common
      when: package_install == 'online'

    - include: users/master_users.yml

    - include_role:
        name: docker

  roles:
    - { role: docker-consul, consul_role: 'server' }
    - { role: docker-registrator }
    - { role: docker-zookeeper }
    - { role: mesos, mesos_role: 'master' }
    - { role: docker-marathon }
```

Appendix 4. Agent playbook

```
---
- name: get facts from orchestration nodes
  hosts: orchestration
  tags: configure

- name: set up application nodes
  hosts: application
  vars_files:
    - vars/versions.yml
  become: yes

  pre_tasks:
    - name: include instance-specific extra variables
      include_vars: "{{ item }}"
      with_fileglob: extra_vars/*
      tags: configure

    - name: add mesos_masters, zookeeper_servers, consul_server
      add_host:
        groups: zookeeper_servers,consul_servers,mesos_masters
        name: "{{ item }}"
      with_items:
        - "{{ groups['orchestration'] }}"
      changed_when: false
      tags: configure

    - name: ensure firewalld is installed
      yum:
        name: firewalld
        state: present

    - include_role:
        name: qvcp-common
      when: package_install == 'online'

    - include: users/agent_users.yml

    - include_role:
        name: docker

    - include: container_images.yml
      when: package_install == 'offline'

  roles:
    - { role: docker-consul, consul_role: 'agent' }
    - { role: mesos, mesos_role: 'slave' }
    - { role: docker-registrator }
```

Appendix 5. Docker inspect for Consul in cp00

[
  {
    "Id":
"068ee6ff8b9f1d26df668e4ca0e999ebf69f9210db0443a782bf40141d4aad4a",
    "Created": "2018-08-04T23:21:12.948452844Z",
    "Path": "docker-entrypoint.sh",
    "Args": [
      "agent",
      "-dev",
      "-client",
      "0.0.0.0"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 4756,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2018-08-05T04:06:13.812145588Z",
      "FinishedAt": "2018-08-05T04:02:31.322212458Z"
    },
    "Image":
"sha256:e5193fe01bbc3f497319898abdb121be690b403beef8e388a0fc43616e1b418
9",
    "ResolvConfPath": "/var/lib/docker/contain-
ers/068ee6ff8b9f1d26df668e4ca0e999ebf69f9210db0443a782bf40141d4aad4a/re-
solv.conf",

"HostnamePath": "/var/lib/docker/contain-
ers/068ee6ff8b9f1d26df668e4ca0e999ebf69f9210db0443a782bf40141d4aad4a/host
name",

"HostsPath": "/var/lib/docker/contain-
ers/068ee6ff8b9f1d26df668e4ca0e999ebf69f9210db0443a782bf40141d4aad4a/host
s",

"LogPath": "/var/lib/docker/contain-
ers/068ee6ff8b9f1d26df668e4ca0e999ebf69f9210db0443a782bf40141d4aad4a/068
ee6ff8b9f1d26df668e4ca0e999ebf69f9210db0443a782bf40141d4aad4a-json.log",

"Name": "/consul-server",

"RestartCount": 0,

"Driver": "overlay2",

"Platform": "linux",

"MountLabel": "",

"ProcessLabel": "",

"AppArmorProfile": "",

"ExecIDs": null,

"HostConfig": {

"Binds": [

"/usr/share/consul/config:/etc/consul/config:rw",

"/usr/share/consul/data:/consul/data:rw"

],

"ContainerIDFile": "",

"LogConfig": {

"Type": "json-file",

"Config": {

"max-file": "3",

"max-size": "25m"

}

},

"NetworkMode": "host",

"PortBindings": {

"3233/tcp": [

{

```
        "HostIp": "0.0.0.0",
        "HostPort": ""
    }
],
"8300/tcp": [
    {
        "HostIp": "0.0.0.0",
        "HostPort": "8300"
    },
    {
        "HostIp": "0.0.0.0",
        "HostPort": "8300"
    }
],
"8301/tcp": [
    {
        "HostIp": "0.0.0.0",
        "HostPort": "8301"
    }
],
"8302/tcp": [
    {
        "HostIp": "0.0.0.0",
        "HostPort": "8302"
    }
],
"8500/tcp": [
    {
        "HostIp": "0.0.0.0",
        "HostPort": "8500"
    }
],
```

```
    "8543/tcp": [
        {
            "HostIp": "0.0.0.0",
            "HostPort": "8543"
        }
    ]
},
"RestartPolicy": {
    "Name": "always",
    "MaximumRetryCount": 0
},
"AutoRemove": false,
"VolumeDriver": "",
"VolumesFrom": null,
"CapAdd": null,
"CapDrop": null,
"Dns": [],
"DnsOptions": [],
"DnsSearch": [],
"ExtraHosts": null,
"GroupAdd": null,
"IpcMode": "shareable",
"Cgroup": "",
"Links": null,
"OomScoreAdj": 0,
"PidMode": "",
"Privileged": true,
"PublishAllPorts": false,
"ReadonlyRootfs": false,
"SecurityOpt": [
    "label=disable"
],
```

"UTSMode": "",
"UsernsMode": "",
"ShmSize": 67108864,
"Runtime": "runc",
"ConsoleSize": [
  0,
  0
],
"Isolation": "",
"CpuShares": 0,
"Memory": 0,
"NanoCpus": 0,
"CgroupParent": "",
"BlkioWeight": 0,
"BlkioWeightDevice": null,
"BlkioDeviceReadBps": null,
"BlkioDeviceWriteBps": null,
"BlkioDeviceReadIOps": null,
"BlkioDeviceWriteIOps": null,
"CpuPeriod": 0,
"CpuQuota": 0,
"CpuRealtimePeriod": 0,
"CpuRealtimeRuntime": 0,
"CpusetCpus": "",
"CpusetMems": "",
"Devices": null,
"DeviceCgroupRules": null,
"DiskQuota": 0,
"KernelMemory": 0,
"MemoryReservation": 0,
"MemorySwap": 0,
"MemorySwappiness": null,

"OomKillDisable": false,

"PidsLimit": 0,

"Ulimits": null,

"CpuCount": 0,

"CpuPercent": 0,

"IOMaximumIOps": 0,

"IOMaximumBandwidth": 0

},

"GraphDriver": {

"Data": {

"LowerDir": "/var/lib/docker/over-
lay2/a9bfb369817e1dad252be9b5b2abc317138f0f1a38e847845f684d2077a76865-
init/diff:/var/lib/docker/over-
lay2/5c4b4eef54e5220a1f9f9f9755abff35c6280379c21ed3af4bbc784f982bc28c/diff:
/var/lib/docker/over-
lay2/fff2015282b6dc7b204628bfd5a60c3206c547265d3d47921e285fff785a0d53/diff
:/var/lib/docker/over-
lay2/4b1786398ec70e53622b6f798079224b0c7b38b8a537c60e75839bab3cf4acac/di
ff:/var/lib/docker/over-
lay2/fe1fd078f730207e266ec6f0c14b68ba5e90af76fbed85d2b0d9d974c3416ff1/diff:
/var/lib/docker/over-
lay2/2a8815a10074804d9dbbf00d58632f464e0d8b673f15b6df5d9f0edfd943f4f9/dif
f",

"MergedDir": "/var/lib/docker/over-
lay2/a9bfb369817e1dad252be9b5b2abc317138f0f1a38e847845f684d2077a76865/
merged",

"UpperDir": "/var/lib/docker/over-
lay2/a9bfb369817e1dad252be9b5b2abc317138f0f1a38e847845f684d2077a76865/d
iff",

"WorkDir": "/var/lib/docker/over-
lay2/a9bfb369817e1dad252be9b5b2abc317138f0f1a38e847845f684d2077a76865/w
ork"

},

"Name": "overlay2"

},

"Mounts": [

{

"Type": "bind",

```
            "Source": "/usr/share/consul/data",
            "Destination": "/consul/data",
            "Mode": "rw",
            "RW": true,
            "Propagation": "rprivate"
        },
        {
            "Type": "bind",
            "Source": "/usr/share/consul/config",
            "Destination": "/etc/consul/config",
            "Mode": "rw",
            "RW": true,
            "Propagation": "rprivate"
        }
    ],
    "Config": {
        "Hostname": "cp00",
        "Domainname": "",
        "User": "",
        "AttachStdin": false,
        "AttachStdout": false,
        "AttachStderr": false,
        "ExposedPorts": {
            "3233/tcp": {},
            "8300/tcp": {},
            "8301/tcp": {},
            "8301/udp": {},
            "8302/tcp": {},
            "8302/udp": {},
            "8500/tcp": {},
            "8543/tcp": {},
            "8600/tcp": {},
```

```
    "8600/udp": {}
  },
  "Tty": false,
  "OpenStdin": false,
  "StdinOnce": false,
  "Env": [
    "SERVICE_53_IGNORE=True",
    "SERVICE_NAME=cp-consulserver",
    "SERVICE_8300_IGNORE=True",
    "SERVICE_8302_IGNORE=True",
    "CONSUL_ALLOW_PRIVILEGED_PORTS=True",
    "SERVICE_IP=192.168.81.60",
    "SERVICE_8301_IGNORE=True",
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "CONSUL_VERSION=1.1.0",
    "HASHICORP_RELEASES=https://releases.hashicorp.com"
  ],
  "Cmd": [
    "agent",
    "-dev",
    "-client",
    "0.0.0.0"
  ],
  "ArgsEscaped": true,
  "Image": "artifactory.qvantel.net/cp-consul:1.1.0",
  "Volumes": {
    "/consul/data": {},
    "/etc/consul/config": {}
  },
  "WorkingDir": "",
  "Entrypoint": [
    "docker-entrypoint.sh"
```

```
        ],
        "OnBuild": null,
        "Labels": {}
    },
    "NetworkSettings": {
        "Bridge": "",
        "SandboxID":
"330f98b6565b8928fd77da1cea5398f0278b0bcf08a5cdd9b8def2f46e3e7cc5",
        "HairpinMode": false,
        "LinkLocalIPv6Address": "",
        "LinkLocalIPv6PrefixLen": 0,
        "Ports": {},
        "SandboxKey": "/var/run/docker/netns/default",
        "SecondaryIPAddresses": null,
        "SecondaryIPv6Addresses": null,
        "EndpointID": "",
        "Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "IPAddress": "",
        "IPPrefixLen": 0,
        "IPv6Gateway": "",
        "MacAddress": "",
        "Networks": {
            "host": {
                "IPAMConfig": null,
                "Links": null,
                "Aliases": null,
                "NetworkID":
"2ef70a9501d223bd8bc11226a8b9138066ba844738c25fbe419702a88c829a57",
                "EndpointID":
"10f263b1e01c505549061502f255cdc71b1d94ee370d8a639149d6269be77346",
                "Gateway": "",
```

```
                "IPAddress": "",
                "IPPrefixLen": 0,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "MacAddress": "",
                "DriverOpts": null
            }
        }
    }
}
]
```

Appendix 6. Docker inspect for Marathon in cp00

[
{
"Id":
"49e22cb2ccf337bc7341f489fa3f2fc45f56cd7dfab95cc6481d9092b078040c",
"Created": "2018-08-04T23:23:46.6371916Z",
"Path": "bin/marathon",
"Args": [],
"State": {
"Status": "running",
"Running": true,
"Paused": false,
"Restarting": false,
"OOMKilled": false,
"Dead": false,
"Pid": 4725,
"ExitCode": 0,
"Error": "",
"StartedAt": "2018-08-05T04:06:13.844340799Z",
"FinishedAt": "2018-08-05T04:02:32.171518339Z"
},
"Image":
"sha256:3673003f7e29b9cf50aa8ddecad6bf76bb28f37ba800e90309e783c7a337d2c
2",
"ResolvConfPath": "/var/lib/docker/contain-
ers/49e22cb2ccf337bc7341f489fa3f2fc45f56cd7dfab95cc6481d9092b078040c/re-
solv.conf",
"HostnamePath": "/var/lib/docker/contain-
ers/49e22cb2ccf337bc7341f489fa3f2fc45f56cd7dfab95cc6481d9092b078040c/host-
name",
"HostsPath": "/var/lib/docker/contain-
ers/49e22cb2ccf337bc7341f489fa3f2fc45f56cd7dfab95cc6481d9092b078040c/hosts
",
"LogPath": "/var/lib/docker/contain-
ers/49e22cb2ccf337bc7341f489fa3f2fc45f56cd7dfab95cc6481d9092b078040c/49e2
2cb2ccf337bc7341f489fa3f2fc45f56cd7dfab95cc6481d9092b078040c-json.log",

```
"Name": "/marathon",
"RestartCount": 0,
"Driver": "overlay2",
"Platform": "linux",
"MountLabel": "",
"ProcessLabel": "",
"AppArmorProfile": "",
"ExecIDs": null,
"HostConfig": {
  "Binds": [],
  "ContainerIDFile": "",
  "LogConfig": {
    "Type": "json-file",
    "Config": {
      "max-file": "3",
      "max-size": "25m"
    }
  },
  "NetworkMode": "host",
  "PortBindings": null,
  "RestartPolicy": {
    "Name": "unless-stopped",
    "MaximumRetryCount": 0
  },
  "AutoRemove": false,
  "VolumeDriver": "",
  "VolumesFrom": null,
  "CapAdd": null,
  "CapDrop": null,
  "Dns": [],
  "DnsOptions": [],
  "DnsSearch": [],
```

"ExtraHosts": null,
"GroupAdd": null,
"IpcMode": "shareable",
"Cgroup": "",
"Links": null,
"OomScoreAdj": 0,
"PidMode": "",
"Privileged": false,
"PublishAllPorts": false,
"ReadonlyRootfs": false,
"SecurityOpt": null,
"UTSMode": "",
"UsernsMode": "",
"ShmSize": 67108864,
"Runtime": "runc",
"ConsoleSize": [
  0,
  0
],
"Isolation": "",
"CpuShares": 0,
"Memory": 0,
"NanoCpus": 0,
"CgroupParent": "",
"BlkioWeight": 0,
"BlkioWeightDevice": null,
"BlkioDeviceReadBps": null,
"BlkioDeviceWriteBps": null,
"BlkioDeviceReadIOps": null,
"BlkioDeviceWriteIOps": null,
"CpuPeriod": 0,
"CpuQuota": 0,

"CpuRealtimePeriod": 0,

"CpuRealtimeRuntime": 0,

"CpusetCpus": "",

"CpusetMems": "",

"Devices": null,

"DeviceCgroupRules": null,

"DiskQuota": 0,

"KernelMemory": 0,

"MemoryReservation": 0,

"MemorySwap": 0,

"MemorySwappiness": null,

"OomKillDisable": false,

"PidsLimit": 0,

"Ulimits": null,

"CpuCount": 0,

"CpuPercent": 0,

"IOMaximumIOps": 0,

"IOMaximumBandwidth": 0

},

"GraphDriver": {

"Data": {

"LowerDir": "/var/lib/docker/over-
lay2/cb0f3c5063ea169c98230066246eae83e71edaeee31c8dd03e9cac502ae8ad39-
init/diff:/var/lib/docker/over-
lay2/881fcda0a468001c6841a5a23911ee93248309750e69afc8027f64f42f59fff6/diff:
/var/lib/docker/over-
lay2/26cb9f498201f1d11a4324034a65413e764e9e5631e0c8ae03b2ce781d60ddd5/
diff:/var/lib/docker/over-
lay2/268483e59b8455feb42ea7f6dfd46257b2003bca7e244e06fc622335e911d13c/di
ff:/var/lib/docker/over-
lay2/5d9fc7094e143a1de1e0caa2fed72c947c60f9f0ea17da0a53f73b0660bd9ae1/dif
f:/var/lib/docker/overlay2/e7141b08ab-
abfbc7b82545f5daa1afb285721fa25c9d949c9a86d2580047aa6d/diff:/var/lib/docker
/over-
lay2/da7186b827106fb5bf008438e6288800dc0bca22be2d7885a4093abdd0174fdc/d
iff",

"MergedDir": "/var/lib/docker/over-
lay2/cb0f3c5063ea169c98230066246eae83e71edaeee31c8dd03e9cac502ae8ad39/
merged",

"UpperDir": "/var/lib/docker/over-
lay2/cb0f3c5063ea169c98230066246eae83e71edaeee31c8dd03e9cac502ae8ad39/d
iff",

"WorkDir": "/var/lib/docker/over-
lay2/cb0f3c5063ea169c98230066246eae83e71edaeee31c8dd03e9cac502ae8ad39/w
ork"

},

"Name": "overlay2"

},

"Mounts": [],

"Config": {

"Hostname": "cp00",

"Domainname": "",

"User": "root",

"AttachStdin": false,

"AttachStdout": false,

"AttachStderr": false,

"Tty": false,

"OpenStdin": false,

"StdinOnce": false,

"Env": [

"LIBPROCESS_PORT=8585",

"MARATHON_ZK=zk://192.168.81.60:2181/marathon",

"MARATHON_LOGGING_LEVEL=ERROR",

"MARATHON_HTTP_ADDRESS=192.168.81.60",

"MARATHON_MASTER=zk://192.168.81.60:2181/mesos",

"LIBPROCESS_IP=192.168.81.60",

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",

"JAVA_HOME=/docker-java-home"

],

"Cmd": [],

"ArgsEscaped": true,
"Image": "artifactory.qvantel.net/cp-marathon:v1.5.8",
"Volumes": null,
"WorkingDir": "/marathon",
"Entrypoint": [
"bin/marathon"
],
"OnBuild": null,
"Labels": {}
},
"NetworkSettings": {
"Bridge": "",
"SandboxID":
"96c93c01ce1953fabfd544bff0037aecad5ccc856009e90e5b83f5497e97ad2c",
"HairpinMode": false,
"LinkLocalIPv6Address": "",
"LinkLocalIPv6PrefixLen": 0,
"Ports": {},
"SandboxKey": "/var/run/docker/netns/default",
"SecondaryIPAddresses": null,
"SecondaryIPv6Addresses": null,
"EndpointID": "",
"Gateway": "",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"IPAddress": "",
"IPPrefixLen": 0,
"IPv6Gateway": "",
"MacAddress": "",
"Networks": {}
}
}

]

Appendix 7. Docker inspect for Zookeeper in cp00

[
  {
    "Id":
"b1196ac82b1e1d97b9d8681e041fb5d266cf1bac349c32407703d5a9c363b094",
    "Created": "2018-08-04T23:22:07.067227185Z",
    "Path": "/docker-entrypoint.sh",
    "Args": [
      "zkServer.sh",
      "start-foreground"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 4706,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2018-08-05T04:06:13.824277976Z",
      "FinishedAt": "2018-08-05T04:02:31.650611049Z"
    },
    "Image":
"sha256:bf5cbc9d5cac93b5688523961f994897e1e51d37804b50390f0247ea3537e2f
b",
    "ResolvConfPath": "/var/lib/docker/contain-
ers/b1196ac82b1e1d97b9d8681e041fb5d266cf1bac349c32407703d5a9c363b094/re
solv.conf",
    "HostnamePath": "/var/lib/docker/contain-
ers/b1196ac82b1e1d97b9d8681e041fb5d266cf1bac349c32407703d5a9c363b094/h
ostname",

"HostsPath": "/var/lib/docker/containers/b1196ac82b1e1d97b9d8681e041fb5d266cf1bac349c32407703d5a9c363b094/hosts",

"LogPath": "/var/lib/docker/containers/b1196ac82b1e1d97b9d8681e041fb5d266cf1bac349c32407703d5a9c363b094/b1196ac82b1e1d97b9d8681e041fb5d266cf1bac349c32407703d5a9c363b094-json.log",

"Name": "/zookeeper",

"RestartCount": 0,

"Driver": "overlay2",

"Platform": "linux",

"MountLabel": "",

"ProcessLabel": "",

"AppArmorProfile": "",

"ExecIDs": null,

"HostConfig": {

"Binds": [

"/usr/share/zookeeper/conf/zoo.cfg:/conf/zoo.cfg:rw",

"/usr/share/zookeeper/data:/data:rw",

"/usr/share/zookeeper/datalog:/datalog:rw"

],

"ContainerIDFile": "",

"LogConfig": {

"Type": "json-file",

"Config": {

"max-file": "3",

"max-size": "25m"

}

},

"NetworkMode": "host",

"PortBindings": null,

"RestartPolicy": {

"Name": "unless-stopped",

"MaximumRetryCount": 0
},
"AutoRemove": false,
"VolumeDriver": "",
"VolumesFrom": null,
"CapAdd": null,
"CapDrop": null,
"Dns": [],
"DnsOptions": [],
"DnsSearch": [],
"ExtraHosts": null,
"GroupAdd": null,
"IpcMode": "shareable",
"Cgroup": "",
"Links": null,
"OomScoreAdj": 0,
"PidMode": "",
"Privileged": false,
"PublishAllPorts": false,
"ReadonlyRootfs": false,
"SecurityOpt": null,
"UTSMode": "",
"UsernsMode": "",
"ShmSize": 67108864,
"Runtime": "runc",
"ConsoleSize": [
0,
0
],
"Isolation": "",
"CpuShares": 0,
"Memory": 0,

"NanoCpus": 0,
"CgroupParent": "",
"BlkioWeight": 0,
"BlkioWeightDevice": null,
"BlkioDeviceReadBps": null,
"BlkioDeviceWriteBps": null,
"BlkioDeviceReadIOps": null,
"BlkioDeviceWriteIOps": null,
"CpuPeriod": 0,
"CpuQuota": 0,
"CpuRealtimePeriod": 0,
"CpuRealtimeRuntime": 0,
"CpusetCpus": "",
"CpusetMems": "",
"Devices": null,
"DeviceCgroupRules": null,
"DiskQuota": 0,
"KernelMemory": 0,
"MemoryReservation": 0,
"MemorySwap": 0,
"MemorySwappiness": null,
"OomKillDisable": false,
"PidsLimit": 0,
"Ulimits": null,
"CpuCount": 0,
"CpuPercent": 0,
"IOMaximumIOps": 0,
"IOMaximumBandwidth": 0
},
"GraphDriver": {
"Data": {

"LowerDir": "/var/lib/docker/over-
lay2/3f852bb6e9d51ded1abd4040bef4a7ef0d4753bcc2674a16bdd1ae80812152b6-
init/diff:/var/lib/docker/over-
lay2/a3ce44b72b508cf7b141442d45d660fc63e6958f5d4153f6b6fded40f8325163/dif
f:/var/lib/docker/over-
lay2/55a82f18e897e952e34687f502de769f20548e0bfa187c84edfcb1597efee1dd/dif
f:/var/lib/docker/over-
lay2/ca1785fdfe5cc9ced12aa6c9c254c02562d2110e4ccf60147a39014b0a2b836f/diff
:/var/lib/docker/over-
lay2/a17f7707e0802a14af12543e3bfe59ce568c4fdcfeb43be478ff0b154d34fb8c/diff:
/var/lib/docker/over-
lay2/3cffda207948dbfd5b2e855079d3c468f7c9af9cde069a98b92dda9217ff7a88/diff
:/var/lib/docker/over-
lay2/e5abfe0371c68dd7cb39b46f7e852a554ffd5210f05e96b73cd5ec312d370204/dif
f:/var/lib/docker/over-
lay2/2a8815a10074804d9dbbf00d58632f464e0d8b673f15b6df5d9f0edfd943f4f9/dif
f",

"MergedDir": "/var/lib/docker/over-
lay2/3f852bb6e9d51ded1abd4040bef4a7ef0d4753bcc2674a16bdd1ae80812152b6/
merged",

"UpperDir": "/var/lib/docker/over-
lay2/3f852bb6e9d51ded1abd4040bef4a7ef0d4753bcc2674a16bdd1ae80812152b6/
diff",

"WorkDir": "/var/lib/docker/over-
lay2/3f852bb6e9d51ded1abd4040bef4a7ef0d4753bcc2674a16bdd1ae80812152b6/
work"

},

"Name": "overlay2"

},

"Mounts": [

{

"Type": "bind",

"Source": "/usr/share/zookeeper/conf/zoo.cfg",

"Destination": "/conf/zoo.cfg",

"Mode": "rw",

"RW": true,

"Propagation": "rprivate"

},

{

```
            "Type": "bind",
            "Source": "/usr/share/zookeeper/data",
            "Destination": "/data",
            "Mode": "rw",
            "RW": true,
            "Propagation": "rprivate"
        },
        {
            "Type": "bind",
            "Source": "/usr/share/zookeeper/datalog",
            "Destination": "/datalog",
            "Mode": "rw",
            "RW": true,
            "Propagation": "rprivate"
        }
    ],
    "Config": {
        "Hostname": "cp00",
        "Domainname": "",
        "User": "",
        "AttachStdin": false,
        "AttachStdout": false,
        "AttachStderr": false,
        "ExposedPorts": {
            "2181/tcp": {},
            "2888/tcp": {},
            "3888/tcp": {}
        },
        "Tty": false,
        "OpenStdin": false,
        "StdinOnce": false,
        "Env": [
```

```
"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/jvm/java-1.8-openjdk/jre/bin:/usr/lib/jvm/java-1.8-openjdk/bin:/zookeeper-3.4.12/bin",
    "LANG=C.UTF-8",
    "JAVA_HOME=/usr/lib/jvm/java-1.8-openjdk/jre",
    "JAVA_VERSION=8u151",
    "JAVA_ALPINE_VERSION=8.151.12-r0",
    "ZOO_USER=zookeeper",
    "ZOO_CONF_DIR=/conf",
    "ZOO_DATA_DIR=/data",
    "ZOO_DATA_LOG_DIR=/datalog",
    "ZOO_PORT=2181",
    "ZOO_TICK_TIME=2000",
    "ZOO_INIT_LIMIT=5",
    "ZOO_SYNC_LIMIT=2",
    "ZOO_MAX_CLIENT_CNXNS=60",
    "ZOOCFGDIR=/conf"
  ],
  "Cmd": [
    "zkServer.sh",
    "start-foreground"
  ],
  "ArgsEscaped": true,
  "Image": "artifactory.qvantel.net/cp-zookeeper:3.4.12",
  "Volumes": {
    "/conf/zoo.cfg": {},
    "/data": {},
    "/datalog": {}
  },
  "WorkingDir": "/zookeeper-3.4.12",
  "Entrypoint": [
    "/docker-entrypoint.sh"
```

```
    ],
    "OnBuild": null,
    "Labels": {}
  },
  "NetworkSettings": {
    "Bridge": "",
    "SandboxID":
"3d9f56a151b6964ab2f9708672a76c6e4f40a4dfa2f4ff478d462674cb3e2aac",
    "HairpinMode": false,
    "LinkLocalIPv6Address": "",
    "LinkLocalIPv6PrefixLen": 0,
    "Ports": {},
    "SandboxKey": "/var/run/docker/netns/default",
    "SecondaryIPAddresses": null,
    "SecondaryIPv6Addresses": null,
    "EndpointID": "",
    "Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "IPAddress": "",
    "IPPrefixLen": 0,
    "IPv6Gateway": "",
    "MacAddress": "",
    "Networks": {
      "host": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID":
"2ef70a9501d223bd8bc11226a8b9138066ba844738c25fbe419702a88c829a57",
        "EndpointID":
"67a90d1dfc276a203c7918d713610c222b19d69e5453dc0f968dcc1ef6ac8439",
        "Gateway": "",
```

"IPAddress": "",
"IPPrefixLen": 0,
"IPv6Gateway": "",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"MacAddress": "",
"DriverOpts": null
}
}
}
}
]