



Expertise
and insight
for the future

Youqin Sun

A Comparison between Java and Go for Microservice Development and Cloud Deployment

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

06 April 2021

Author Title	Youqin Sun A Comparison between Java and Go for Microservice Development and Cloud Deployment
Number of Pages Date	38 pages 06 April 2021
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Mobile Solutions
Instructors	Patrick Ausderau, Senior Lecturer Anne Pajala, Lecturer
<p>In the era of Big Data and light-speed socio-economic development, the scalability, availability, and reliability of software systems become increasingly important. As cloud technology advances and matures, a growing number of organizations are moving their operations to the cloud (Babar and Chauhan, 2011). However, the processes of developing applications with different programming languages may differ from one to another. Software projects with similar functionalities may require different resources and produce distinct performance outcomes if they are not built with the same language.</p> <p>In this paper, the theoretical knowledge of cloud computing, containerization and microservice was presented based on up-to-date resources. Two microservices with equivalent functions were successfully developed and deployed to the Google Cloud Platform using Docker, Helm and Kubernetes. One of the projects was developed with Java and the other with Go. Furthermore, the discrepancies in resource consumption and efficiency were observed and analyzed. Moreover, it was found that the application developed with Go was more efficient and cost-effective. However, the exact advantages and drawbacks of each language depends heavily on the scale, complexity and features of the software system to be developed. Each programming language offers distinct features and various supporting libraries and frameworks. Therefore, extensive research on how to maximize the advantage of those elements for the best outcome of the target application is strongly advised prior to the implementation of such projects.</p>	
Keywords	Microservice, Java, Go, Kubernetes, Docker, Cloud

Contents

List of Abbreviations

1	Introduction	1
2	Theoretical Background	3
2.1	Cloud	3
2.1.1	Cloud Computing	3
2.1.2	Cloud Services	4
2.2	Container Technology	5
2.2.1	Container and Container Image	5
2.2.2	Containerization	6
2.2.3	Container Orchestration	7
2.3	Microservices	8
3	Project Planning	10
3.1	Languages and Frameworks	10
3.2	Development and Deployment Mechanisms	12
4	Project Implementation	14
4.1	Application Development	14
4.2	Packaging for Cloud Development	15
4.3	Cloud Deployment	22
4.4	Load Testing	24
4.5	Outcomes and Analysis	25
5	Conclusion	32
	References	34

List of Abbreviations

API	Application Programming Interface. A collection of protocols defining how to interact with a software intermediary (IBM, 2021).
AWS	Amazon Web Service. A cloud-computing platform provided by Amazon.
CD	Continuous Deployment.
CI	Continuous Integration.
CPU	Central Processing Unit. The main processor of a computer system.
CLI	Command Line Interface. A program accepting text input to execute system functions.
GCP	Google Cloud Platform. A collection of computing infrastructures provided by Google as a set of cloud services (Gupta et al., 2020).
GKE	Google Kubernetes Engine. An environment to run and manage containerized applications using Google infrastructure (GKE, 2021).
IaaS	Infrastructure as a Service. Computing infrastructures provided as an online service (Tang, Ren and Pan, 2014).
JAR	Java ARchive. A file format for zipping many Java class files into one ((Oracle, 2021).
MSA	Microservice Architecture. A software development method focusing on independence for efficiency and scalability (Redhat Microservices, 2020).
PaaS	Platform as a Service. A platform provided as a service for customers to develop, deploy and manage applications (Tang, Ren and Pan, 2014).
POM	Project Object Model. An XML file containing information and configuration details for a project created with Maven (Apache Maven, 2021).

SaaS	Software as a Service. A software provided as a service based on subscription to or rental of the software (Tang, Ren and Pan, 2014).
URL	Uniform Resource Locator. A reference to the location of web resource(s) over the Internet.
VCS	Version Control System. A system that keeps track of changes made to a file or a set of files (Bieniusa, Thiemann and Wehr, 2008).

1 Introduction

The world is experiencing an explosion of information as Internet-capable devices are generating an unprecedented amount of data every day, which needs to be stored, processed and delivered constantly. As the backbone to this phenomenon, cloud technology is emerging and seeing steady and fast growth as more and more services are made available through the cloud. (Moud et al., 2020.) Nowadays nearly all the most popular applications on the market are deployed and offered via the cloud, which enables instant and seamless interactions between users and applications. (Ahmad and Kamvar, 2013.) As cloud computing gains popularity around the world, the number of businesses providing and using virtual services over the cloud increases significantly. (Babar and Chauhan, 2011.)

Despite the conveniences and advantages to the end users, cloud development and deployment face numerous challenges (Ahmad and Kamvar, 2013). The process of building applications with different programming languages may vary to a great extent. Because different programming languages have different libraries, tools and mechanisms when it comes to cloud building and deployment. Furthermore, applications with identical functionalities but made with different languages may require non-identical computing resources and infrastructures. (Li and Wan, 2018.)

After they have been deployed to the cloud successfully, one application may perform very differently from another. Each programming language offers unique benefits and produces distinctive challenges as well, even though the exact advantages and concerns depend on the purpose and use case of the application certainly.

The goal of this thesis is to find out the differences in the processes of application development with different programming languages and frameworks for cloud deployment and analyze the benefits and concerns during the implementation process. The project is also aimed to compare the cloud computing resources needed for different applications and observe the performance results after the projects have been deployed to the cloud.

Java and Go were chosen as two example programming languages to develop two microservices with similar functionalities, which provide users with lottery number services,

generating selected sets of lottery numbers or retrieving past winning numbers. Furthermore, both applications were deployed to the Google Cloud Platform (GCP) using Google Kubernetes Engine (GKE), Helm and Docker. Hence, the ultimate purpose is narrowed down to discover the answers to the above-mentioned questions based on those resources as well as the mechanisms related to them.

The content of this thesis includes an Introduction section, which introduces the project idea, scope and objectives, followed by the Theoretical Background, which explains the theoretical knowledge of technologies. Project Planning laid out the programming languages, frameworks and mechanisms used in the thesis, as well as tools, systems and platforms for cloud deployment. Project Implementation documents in detail the implementation process of the software projects. The testing against the microservices and the performance results are documented in this section as well. The observations and final outcome of this thesis can be found in the Conclusion part.

2 Theoretical Background

2.1 Cloud

2.1.1 Cloud Computing

The cloud, once seen as a buzzword, is evolving into a popular technology, which is being widely used across industries all over the world (Wang, 2019). In the information technology industry, the cloud by definition means an enormous network of servers distributed across the world but connected together over the Internet. As a result, one single harmonious ecosystem is formed, as seen in Figure 1. The whole network can be used to store, manage and deliver content for its users. With the cloud technology, data is highly available and can be accessed easily at any desired location or time through any Internet-connected device, eliminating the restrictions of storing to and retrieving from physical storage systems locally. (Microsoft Azure, 2021.) Despite all the advantages, the new technology emerged with concerns as well, including risks in security and data privacy, and doubt about its performance and reliability (Efozia et al., 2017).



Figure 1. Cloud computing (Microsoft Azure, 2021).

Cloud systems can be generally divided into three categories: private, public and hybrid. Private cloud is a system, which is designed for and can be accessed by only one organization. If the platform shares resources over the Internet and is open to the public, it is called a public cloud. Resources offered over this system might be free or charged based on usage. A hybrid cloud combines the previous two environments. Depending on the situation, resources and services can be shared between the public and private sectors in a hybrid system to achieve a more cost-effective formation. (IBM, 2021.)

Cloud computing is defined as computing infrastructures made available through the Internet, a modern and advanced alternative to the traditional way of using computing powers locally (Ray, 2018). In traditional data centers, devices and equipment need to be installed on site and maintained by its owner, which is costly, and the usage of resources is not optimized. (Shuping and Feng, 2019.)

With the emergence and advancement of cloud computing technologies, IT resources are offered on demand to customers regardless of their type, industry and size. The resources are highly available, which customers can order and use whenever it is necessary, and they pay for what they use only. Cloud technology makes the usage of computing resources more efficient and cost-effective. Over the cloud, users have access to a wide range of services, which enable businesses to build more innovative systems. Moreover, the deployment and management of applications are made easier and faster globally as cloud infrastructures are distributed across the world. The elasticity feature of cloud provides auto scaling instantly based on actual data generation, mitigating the worries of peaks in business activities. (AWS, 2020.)

2.1.2 Cloud Services

Cloud computing service types include Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The service content ranges from hardware, software and networking to databases, storage and intelligence. Through cloud computing technology resources can be accessed in a more flexible manner and at different scales. Users of cloud services are charged typically by how much they actually use. In this way, operating costs are lowered, and resources are used more efficiently. (Microsoft, 2020.)

With cloud computing services available, organizations no longer need to buy and maintain their own physical hardware and data centers all the time. Instead, they can order the infrastructures from the Internet only when it is necessary and at a scale they desire. Moreover, they can scale up or down the service freely based on the business requirements and market demands. (Amazon Web Service, 2020.) Currently major cloud service providers dominating the market include Amazon Web Service (AWS), Microsoft Azure and Google Cloud Platform, with many more fast-growing players emerging and joining the competition (Li and Wan, 2018).

2.2 Container Technology

2.2.1 Container and Container Image

A container image is a software package composed of all the elements needed for an application to be deployable, including code, system tools, libraries, runtime, and settings. The package is normally a lightweight and standalone executable file. (Docker, 2020.) The binary file also provides the requirements to run the container, and the description about its necessities and capabilities. If a container is not given additional rights when created, it can access only the resources included in the container image. (RedHat OpenShift, 2020.)

As Figure 2 illustrates, a container in cloud computing is a software package containing all the components that are needed for an application to run in any computing environment easily and consistently. It represents a virtualization approach to system operation. It is formed by packaging the code and all dependencies of an application into a standardized unit of software. A container image must be created beforehand. The image transforms into a container when the binary file is executed in a computing infrastructure during runtime. (Docker, 2020.) A container uses an image file to create the run-time environment and run the application defined by the image. Therefore, container and container image represent two inseparable phases to run a container application. (phoenixNAP, 2020.)

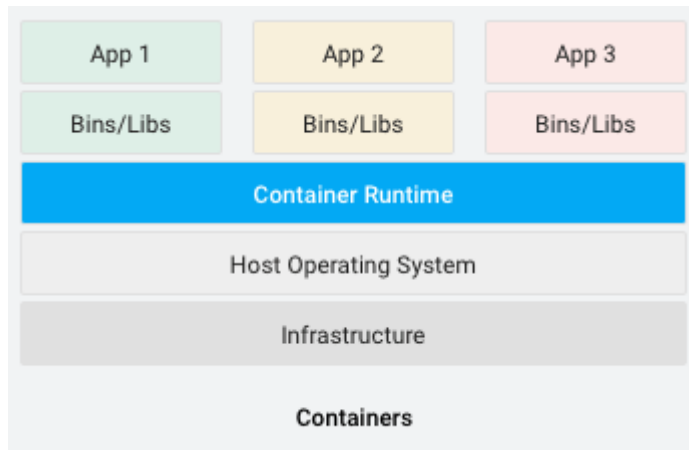


Figure 2. Containers (Google, 2020)

Containers make use of the operation system it is running on to provide services to it, thus they are very lightweight by nature (Sami and Mourad, 2018). Also, it is becoming an increasingly popular choice by organizations for long-running services, large scales of batch processing, as well as workloads of Internet of Things and Artificial Intelligence (Khan, 2017).

2.2.2 Containerization

Containerization is the process of encapsulating the code and dependencies of a software application into a portable package so that it can be run in any computing environment (IBM, 2020). This technology is an implementation of operating system virtualization, which allows applications to run in isolated containers (Citrix, 2020). With maturing containerization technology, developers no longer need to worry about bugs that might occur due to different operating systems, because all the libraries, dependencies and configuration files needed for the software to run are packaged together with the code (IBM, 2020).

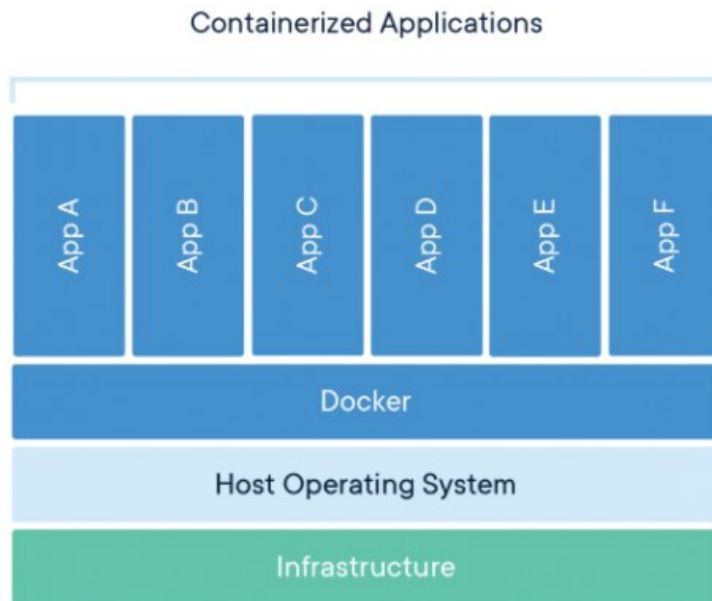


Figure 3. Containerized applications (Docker, 2020)

As illustrated by Figure 3, containerization effectively decouples the worries of software developers and operation teams, as the developing teams can concentrate their efforts on responsibilities of application development and dependencies, while the operators are relieved from the problems caused by software versions and configuration specifications. This process also enables fast, consistent, and reliable deployment of the application regardless of differences in operating platforms. (Google, 2020.)

2.2.3 Container Orchestration

As cloud computing and container technology advances, modern applications are made of dozens of or even hundreds of components, which are loosely coupled and containerized microservices. To make sure an application functions as desired, all those components must work together harmoniously and consistently. Cloud orchestration represents the process of synthesizing numerous workloads across possibly several different cloud platforms into a single workflow by automating the work of container deploying, managing, scaling and networking. (Redhat Containers, 2020.) The automated implementation processes keep all tasks synchronized and reduce manual intervention. Moreover, permission oversight and policy enforcement can be configured to ensure performance and security.

Orchestration tools are designed to configure, deploy, and manage computer and software systems in public, private, and hybrid cloud infrastructure. The technology is typically used to allocate storage capacity, manage networking, and create virtual machines. Furthermore, servers can be provisioned, deployed, and started or stopped by implementing a cohesive workflow. (Redhat Containers, 2020.)

2.3 Microservices

In the era where the world is filled with constantly dynamic business needs and exorbitant amounts of data from a wide range of client types, including web browsers and native mobile applications from various kinds of devices, such as computers, tablets and mobiles, software applications must be developed in a way that can adapt to such diversity as well as non-predictable business activities, such as sudden spikes in data handling. The software industry finds it difficult to keep up with speed with the traditional monolithic architecture in such environments, and their effort to find a new alternative facilitated the emergence and popularization of a new concept for software development, the microservice architecture (MSA). (Alshuqayran, Ali and Evans, 2018.)

Microservice is an architecture technology of momentous importance in the Internet industry (Wang Bin et al., 2017). A microservice refers to the smallest independently deployable component or process of an application. Each microservice implements a specific function with its own logic and database. Typically, a microservice defines an interface as well for other services to interact with it. (Li et al., 2018.)

As demonstrated in Figure 4, microservice architecture is an architectural framework, which breaks an application into a considerable number of small and lightweight components that can function independently from each other. Microservices each work on their own, meaning individual failure does not impact the entire software functionality. This architecture also allows easier debugging, faster development of new components and seamless continuous integration and continuous delivery. (Redhat Microservices, 2020.)

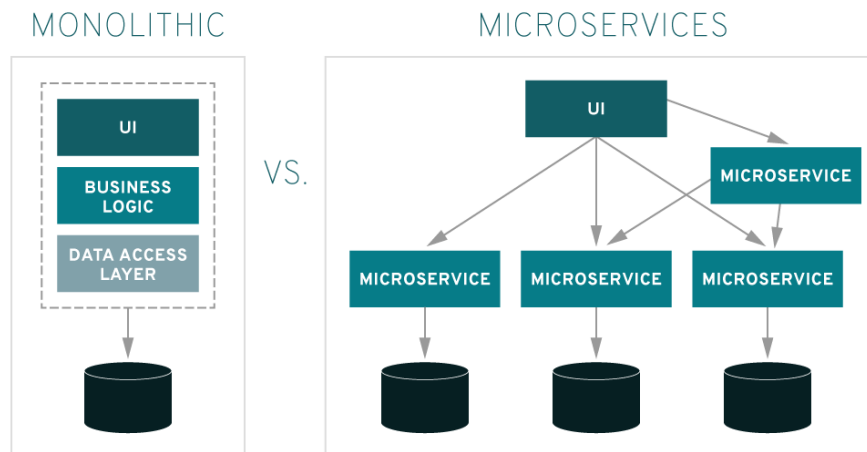


Figure 4. Monolithic VS. Microservices (RedHat Microservices, 2020).

In the traditional monolithic approach where almost all parts of an application are built into one huge-sized piece of software, everything is entangled together, thus modification of any small element may break the whole application. Compared to traditional monolithic applications, microservice architecture offers advantages in efficiency and robustness, as it breaks an entire application into the smallest units, which operate independently from each other and are managed separately by different teams typically. Applications built with microservice mechanisms are becoming increasingly popular. (Rahman and Gao, 2015.)

Microservices allow more detailed and individualistic control of applications deployed on the cloud, as each microservice functions independently and communicates with other applications on its own over an Application Programming Interface (API). It is becoming adopted more widely in the industry. Depending on the application functionality and implementation, each microservice may consume different amounts of resources. (Jindal, Podolskiy and Gerndt, 2019.) Each microservice can be maintained separately, which improves its scalability and availability. However, instead of in-process calls, this needs to be implemented via costly remote calls and the overhead for synchronization across components is higher. (Fazio et al., 2016.)

3 Project Planning

3.1 Languages and Frameworks

Java is a programming language used by millions of engineers and has more than 50 billion Java Virtual Machines worldwide (Oracle, 2021). Java was chosen as the programming language to develop the first application for this thesis project because it is one of the most popular programming languages used worldwide, as seen in Table 1, which lists the top 10 popular programming languages in 2020 based on the number of jobs on the market for each language as well as the average annual salary for a developer of that language.

Table 1. Top 10 most popular programming languages (NorthEastern University, 2021).

Top 10 Popular Programming Languages 2020	
1	Python
2	JavaScript
3	Java
4	C#
5	C
6	C++
7	GO
8	R
9	Swift
10	PHP

The Spring Framework was chosen for creating the Java application of this thesis project. Spring is an open-source application framework for the Java programming language. As a very popular choice among Java developers, it includes many modules which programmers can select freely based on their needs (Gajewski and Zabierowski, 2019). Spring offers various kinds of tools and libraries, which helped speed up the development process and ensure the security of the application (Spring, 2021).

The Go programming language, which can be referred to as GoLang, is an open-source programming language designed at Google. As shown in both Table 1 and Table 2, Go is becoming a very popular programming language worldwide. It is also the language used to develop some of the major cloud infrastructures, such as Docker and Kubernetes, which led to its prevalence especially in the cloud-computing field (GoLang, 2021). The two planned applications for this thesis project are two microservices to be developed and deployed in the cloud, therefore Go was chosen as the programming language for the second application.

Table 2. TIOBE index for March 2021 (TIOBE, 2021).

March 2021	March 2020	Programming Language	Change	Ratings
1	2	C	↑	15.33%
2	1	Java	↓	10.45%
3	3	Python		10.31%
4	4	C++		6.52%
5	5	C#		4.97%
6	6	Visual Basic		4.85%
7	7	JavaScript		2.11%
8	8	PHP		2.07%
9	12	Assembly Language	↑	1.97%
10	9	SQL	↓	1.87%
11	10	Go	↓	1.31%

Similar rankings of Java and Go among other programming languages can also be seen in Table 2, the TIOBE Index. The ratings are calculated by using data from the most dominant search engines. The number of skilled developers and training courses provided on the market are taken into account for the calculation as well. (TIOBE Index, 2021.)

3.2 Development and Deployment Mechanisms

Git was used as the Version Control System (VCS) and GitLab was chosen as the platform to host the repositories of both software projects because it offers built-in tools for Continuous Integration (CI) and Continuous Deployment (CD). As the service that holds the second largest number of project repositories, GitLab has gained higher popularity in recent years (Safari et al., 2020).

Google Cloud Platform (GCP) is a cloud platform that offers quality infrastructure and high standard services for cloud computing, data storage and machine learning (Google Cloud, 2021). GCP was selected as the target platform to deploy the two microservices in view of the fact that it is considered one of the most prominent cloud service providers with various kinds of services for enterprises and individual users all around the world (Mitchell and Zunnurhain, 2019). Based on pre-study and research carried out by the author of this thesis, GCP has very detailed and well-written documentation and step-by-step tutorials¹ for beginners. Furthermore, the platform kindly offers a free-trial package worth €248 for a period of 90 days, which serves well the purpose of this project.

Docker was decided to be the tool to create Docker images for the applications. And Docker Hub was hosting the repository for the images for cloud deployment. Docker is a set of products that offer PaaS to enable software delivery as containers by encapsulating the code and all related configuration files and dependencies into standardized packages with virtualization at operating-system level. With Docker in use, developers gain the freedom to use tools, systems and environments of their own choice. (Docker, 2020.) As cloud technology prevails, Docker has grown into one of the most widely used containerization systems, with which smooth cooperation between application developers and operators is guaranteed. With high portability and scalability as its features, Docker enables faster development and better performance. (Avino et al., 2017.)

Kubernetes, or K8s, is a platform for container orchestration. It is compatible with various container tools, including Docker. This lightweight and extensible mechanism is designed for containerized workloads management, which automates container deploying, scaling and operating jobs. (Kubernetes, 2020.) As container technology advances, the Kubernetes community grows rapidly, and its contributors have made it one of the most

¹ <https://cloud.google.com/kubernetes-engine/docs/quickstart>

active open-source projects in GitHub. (Dikaleh, Sheikh and Felix, 2017.) As a result, more services, support and tools are developed and can be accessed more easily. Additionally, Google Kubernetes Engine (GKE), whose API uses Kubernetes commands and can be managed by the platform, was selected for cloud deployment on GCP. Therefore, Kubernetes was considered a highly suitable tool to deploy the Docker images to the cluster, which consists of a number of virtual machines.

Helm is a highly reliable open-source package management tool for Kubernetes with more than thirty thousand stars in its GitHub repository. The package manager is well maintained by its big contribution community of over one thousand companies, and widely used across the world with over two million of downloads in every single month. (CNCF, 2021.) With Helm, developers can build, publish, and download Helm charts for applications through which the applications can then be defined, installed and upgraded on the Kubernetes platform (Helm, 2021). Therefore, Helm was chosen as the tool to build Helm charts for both applications and install them to the Kubernetes clusters.

4 Project Implementation

4.1 Application Development

One of the purposes of this thesis was to discover the differences in many aspects of two microservice applications with identical functionalities but developed with different programming languages. To fulfill this objective, the development of two applications was the first step in the implementation process. During the research process, the definition of microservice was well taken into consideration, and the target functionality of the applications was to be reliable, efficient, simple and independent.

The final choice on the software application was to make a backend lottery number service, which can provide users randomly generated lottery numbers for new lottery tickets and retrieve past winning numbers. Each time users need to decide on the number of lottery tickets to buy and are allowed to select one of the three lottery types: Viking Lotto, Finnish Lotto and Euro Jackpot. The default value is one set of lottery numbers for the Finnish Lotto. To check previous winning numbers, a user needs to specify a lottery type and the week and year to be checked with. When a request is received from a user, the application sends a request to the Veikkaus API² for draw results and returns the result to the user.

```
@RestController
public class RestApi {

    @Autowired
    private WinningNumberRetriever retriever;

    @GetMapping("/lmg")
    public LotteryNumberGenerationResult getLotteryNumbers(
        @RequestParam(defaultValue = "1") int howMany,
        @RequestParam(defaultValue = "Finnish Lotto") String lottoType) {

        return new LotteryNumberGenerationResult(lottoType, howMany);
    }

    @GetMapping("/result")
    public LotteryNumber getWinningNumber(
        @RequestParam(required = true) String lottoType,
        @RequestParam(required = true) int year,
        @RequestParam(required = true) int week) throws IOException {

        return retriever.getWinningNumber(year, week, lottoType);
    }
}
```

² <https://www.veikkaus.fi/api/draw-results/v1/games/>

Listing 1. Code for Java app API.

The first application was developed with Java, as the developer already had experience of working with it. The project was created with Maven, which is a tool to create Java projects and manage dependencies using Project Object Model (POM) files. Projects created with maven have a standard structure to facilitate navigation. Product code and test code are separated from each other, with each having its own source tree in parallel. (Apache Maven, 2021.) The source code of the Java application is available on a GitLab repository³.

```
import (
    "net/http"
    lnsapi "lns_golang/src/api"
)

func main() {
    http.HandleFunc("/result", lnsapi.GetPastWinningNumber)
    http.HandleFunc("/lng", lnsapi.GetLotteryNumbers)
    http.ListenAndServe(":8080", nil)
}
```

Listing 2. Code for Go app API.

The second application was planned to be written in Go. The developer did not have any previous experience with the programming language, but it took less than two weeks to learn the most basic fundamentals. Go supports type safety and built-in data types and data can be entered dynamically. Collections in Go are simplified into arrays with dynamic size. In addition, Go offers a rich standard library, which contains numerous well-designed and practical packages. The API of this application was written with the built-in `net/http` package, as shown in Listing 2. The source code of this microservice can be found on a GitLab repository⁴ as well.

4.2 Packaging for Cloud Development

The packaging process of the applications included the steps taken to prepare the applications from source code to a Java ARchive (JAR) file for the Java project and a binary for the Go application, and then a docker image, which was uploaded, to the docker repository in Docker Hub. Helm charts were created to generate template resource and

³ https://gitlab.com/youqins/lns_java

⁴ https://gitlab.com/youqins/lns_golang

config files, which were manually updated before they were packaged into a tar, file to upload to a Helm chart repository, which was hosted in Gitlab.

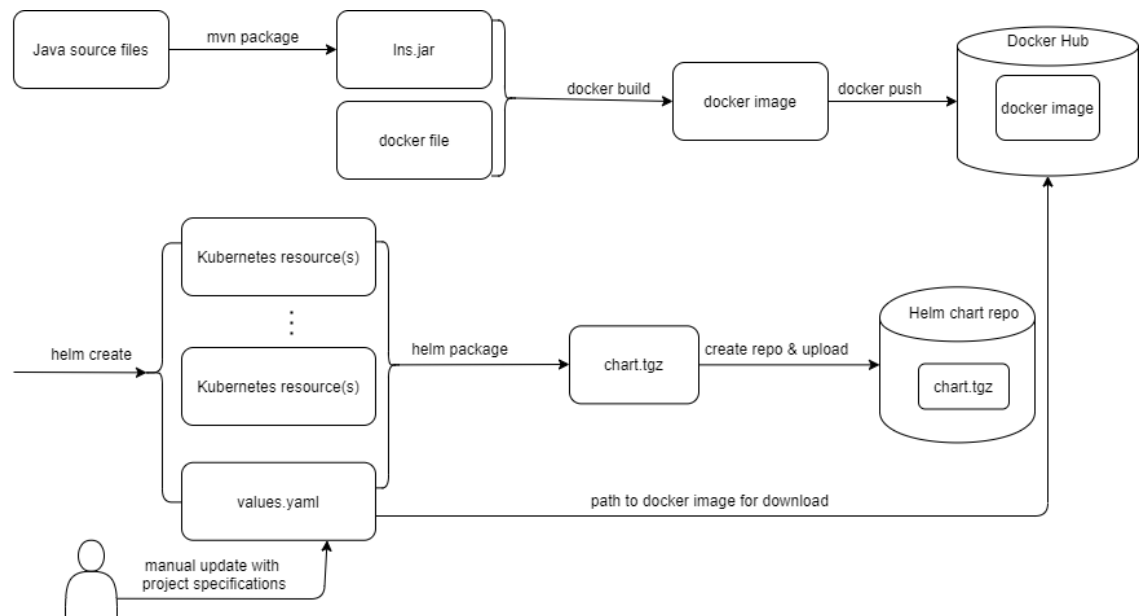


Figure 5. Packaging flowchart-Java app.

The packaging process as displayed in Figure 5 illustrates how the Java application was packaged. The flowchart included packaging steps from source code to docker image. It explained how the Helm chart was prepared for cloud installation. The project was created with maven, which offers the possibility for packaging the source code into a JAR file.

```

FROM openjdk:11-jre
COPY target/*.jar /
WORKDIR /
CMD ["java", "-jar", "lns_java-0.0.1-SNAPSHOT.jar"]
  
```

Listing 3. DockerFile for Java app.

With Docker installed locally, a docker command line interface (CLI) was available, with which the docker image was created by issuing the command `docker build` based on the instructions in the Dockerfile which was created manually with project specifications, as illustrated in Listing 3. A docker repository, as shown in Figure 6, was necessary in Docker Hub for the docker image to be uploaded to and later pulled from it and Docker images were pushed with tags, which were used later to identify which image to pull for creating Helm charts.

dockerhub Search for great content (e.g., mysql)

Repositories > Insth / lottery-number-service

General Tags Builds Collaborators Webhooks Settings

Insth / lottery-number-service

This repository does not have a description

Last pushed: 14 days ago

Tags and Scans VULNERABILITY SCANNING - DISABLED [Enable](#)

This repository contains 18 tag(s).

TAG	OS	PULLED	PUSHED
go-latest		10 days ago	14 days ago
go-fix-docker-image		14 days ago	14 days ago
go-fix-docker-tag		14 days ago	14 days ago
latest		13 days ago	14 days ago

Figure 6. Docker repo for both projects in Docker Hub.

In order to prepare for the installation to Kubernetes, Helm, which is the Kubernetes package manager, was installed locally so that the Helm CLI could be used to create Helm charts. When the command `helm create` was issued via the CLI inside the target project, default template files were generated to allow manual modification with project-specific values. A Helm chart includes a Kubernetes deployment, specifying the pod to be deployed for lottery-service application, and a Kubernetes service, providing the access point to the lottery-service pod.

```
apiVersion: v1
kind: Service
metadata:
  name: {{ include "lottery-service.fullname" . }}
  labels:
    {{- include "lottery-service.labels" . | nindent 4 }}
spec:
  type: ClusterIP
  ports:
```

```

- port: 80
  targetPort: 8080
  protocol: TCP
  name: http
selector:
  {{- include "lottery-service.selectorLabels" . | nindent 4 }}

```

Listing 4. service.yaml for Java app.

When a Helm chart is used for installation, it creates a deployment resource and a service resource in the Kubernetes cluster. As illustrated in Listing 4, the service.yaml file provided access to the pod via the port 80. The IP address was a ClusterIp, which was internal to the cluster where the pod was installed.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "lottery-service.fullname" . }}
  labels:
    {{- include "lottery-service.labels" . | nindent 4 }}
spec:
  replicas: 1
  selector:
    matchLabels:
      {{- include "lottery-service.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      labels:
        {{- include "lottery-service.selectorLabels" . | nindent 8 }}
    spec:
      {{- with .Values.imagePullSecrets }}
      imagePullSecrets:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - name: http
              containerPort: 8080
              protocol: TCP
          livenessProbe:
            httpGet:
              path: /lng
              port: 8080
          readinessProbe:
            httpGet:
              path: /lng
              port: 8080
          resources:
            {{- toYaml .Values.resources | nindent 12 }}

```

Listing 5. deployment.yaml for Java app.

A Kubernetes deployment describes how the pod instances should be created or modified. Also, the deployment generated from a Helm chart creates a pod running the container. The deployment.yaml file in Listing 5 defined the container image by specifying the image repository and tag.

```
image:
  repository: lnsth/lottery-number-service
  pullPolicy: IfNotPresent

  tag: "latest"

imagePullSecrets:
  - name: lottery-svc-registry-credentials
```

Listing 6. Values.yaml for Java app.

The Values.yaml file Listing 6 describes the image repository, pullPolicy and tag needed to pull the docker image from Docker Hub. It also included the imagePullSecrets, which was manually configured in Kubernetes. The Secrets were created to hide credentials for accessing a private repository.

```
apiVersion: v2
name: lottery-service
description: A Helm chart for Kubernetes
type: application
version: 0.1.0
appVersion: 1.16.0
```

Listing 7. Chart.yaml for Java app.

The Chart.yaml file was used to hold the meta data about the Helm chart, as displayed in Listing 7. Once all the values related to the application had been updated, the command `helm package` was issued to create the Helm chart. Helm charts were uploaded to the Helm chart repository in GitLab via CI and later downloaded for installation on Kubernetes clusters.

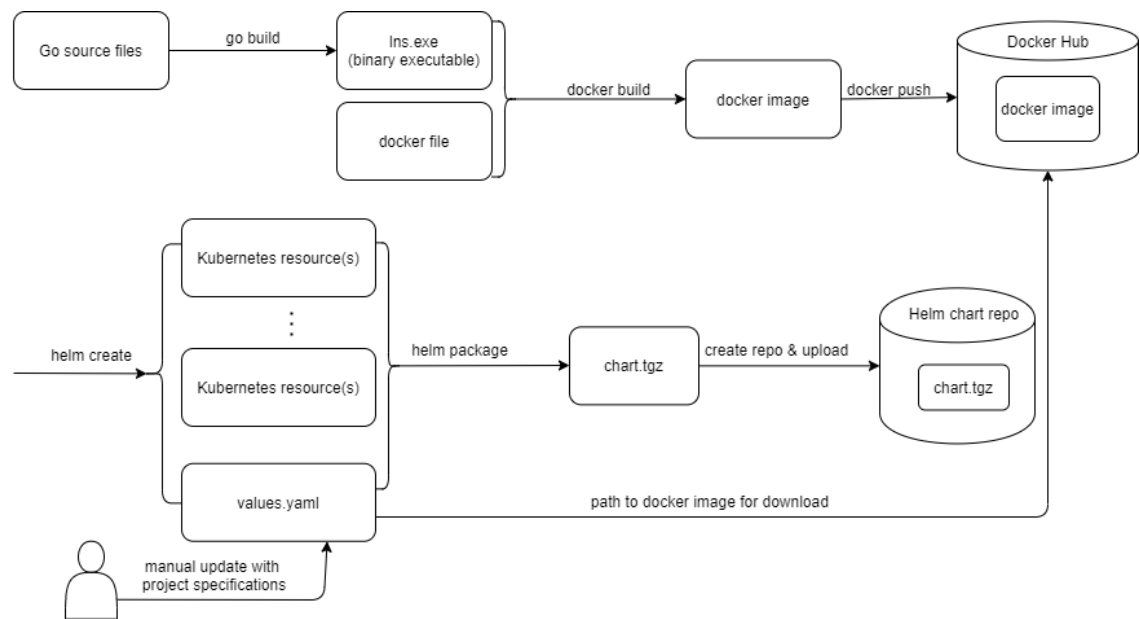


Figure 7. Packaging flowchart-Go app.

The packaging process of the Go application was very similar to that of the Java project, as demonstrated in Figure 7. One difference was that the built-in `go build` tool was used to create the binary executable file for building the docker image. The process of preparing the Helm chart followed the same steps as the Java application.

```

image: maven:latest

stages:
  - build
  - test
  - package
  - docker
  - helm
  - deploy

build:
  stage: build
  script:
    - mvn compile

test:
  stage: test
  script:
    - mvn test

jar:
  stage: package
  script:
    - mvn package
  artifacts:
    paths:
      - target/*.jar
    expire_in: 1 week

buildImage:
  
```

```

stage: docker
image: docker:19.03.12
services:
  - docker:19.03.12-dind
variables:
  IMAGE_NAME_WITH_TAG: ${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_SLUG}
script:
  #default - echo ${CI_REGISTRY_USER} ${CI_REGISTRY} ${CI_REGISTRY_PASSWORD}
  - |
    if [ "${CI_COMMIT_REF_NAME}" == "master" ]; then
      echo "we are on master"
      IMAGE_NAME_WITH_TAG=${CI_REGISTRY_IMAGE}:latest"
    fi
  - echo 'preparing to build and push image $IMAGE_NAME_WITH_TAG'
  - docker login -u ${CI_REGISTRY_USER} -p ${CI_REGISTRY_PASSWORD} ${CI_REGISTRY}
  - docker build -t $IMAGE_NAME_WITH_TAG .
  - docker push $IMAGE_NAME_WITH_TAG

helm:
stage: helm
image: dtzar/helm-kubect1:3.4.1
script:
  - cd helm
  - helm package lottery-service
artifacts:
paths:
  - helm/lottery-service-*.tgz

```

Listing 8. CI file for Java app (Part 1).

All the above-mentioned packaging steps were included in the CI process and automated through preconfigured stages in the GitLab CI file, as shown above in Listing 8. During the package stage, `mvn package` was used to create a JAR file. At the docker stage, a docker image was created and pushed to the specified repository in Docker Hub. A Helm chart was prepared for cloud installation during the last stage.

```

pages:
image:
  name: dtzar/helm-kubect1:3.4.1
stage: deploy
variables:
  DOCKER_IMAGE_NAME_WITH_TAG: ${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_SLUG}
script:
  - mkdir -p ./public
  - "echo \"User-Agent: *\nDisallow: /\n\" > ./public/robots.txt"
  - cd helm/lottery-service
  - wget https://github.com/mikefarah/yq/releases/download/v4.2.0/yq_linux_amd64 -O /usr/bin/yq && chmod +x /usr/bin/yq
  - yq eval '.image.tag="latest"' -i values.yaml
  - cat values.yaml
  - cd ..
  - helm package lottery-service --destination ../public
  - cd ../public
  - helm repo index --url https://${CI_PROJECT_NAMESPACE}.gitlab.io/${CI_PROJECT_NAME} .
artifacts:
paths:
  - public
only:
  - master

```

Listing 9. CI file for Java app (Part 2).

With the CI step `pages` in Listing 9, which was executed on master branch only, a Helm chart was created to the `/public` destination directory. This was accessed later to download the chart for helm installation in Kubernetes. A Uniform Resource Locator (URL) was generated with `helm repo index`, through which the file could be downloaded and viewed.

Status	Pipeline	Triggerer	Commit	Stages	Duration	Actions
passed	#262784193 latest		master → ddf689a Merge branch 'fix-docker-tag-v2...		00:06:01 2 weeks ago	
passed	#262782791		fix-docker-t... → c9b207db store docker image version into ...		00:05:38 2 weeks ago	
failed	#262779556		master → 17cd1677 Merge branch 'fix-docker-tag' in...		00:04:48 2 weeks ago	
passed	#262778614		fix-docker-t... → 6b7f718d store docker image version into ...		00:05:38 2 weeks ago	
canceled	#262774031		master → ba02e005 Merge branch 'store-docker-ver...		00:04:04 2 weeks ago	
passed	#262772717		store-docker... → c5caa2b0 store docker image version into ...		00:05:38 2 weeks ago	

Figure 8. Screenshot of Java app CI/CD Pipelines.

Each time when a new commit was pushed to the project repo in GitLab, a new Pipeline was triggered to run through all the stages defined in the CI file, as shown by the screenshot of part of the Java application's CI Pipelines in Figure 8. New packages with the latest master were built and pushed to DockerHub and Helm Chart Repo automatically.

4.3 Cloud Deployment

The precondition for any operation on GCP is to create an account on the platform. The first step was to select a project or create a new one if no existing projects were available. After the project was all set up, which might take as long as half an hour, the GKE page could be navigated to through the left-side menu. From there, the user can create a new cluster, which will present him/her with the options to set up the details.

In Kubernetes, a node is a server carrying and processing applications. It represents the smallest hardware unit, which can be a physical or virtual machine. A group of nodes form a cluster. (Google Cloud. 2021.) For the purpose of this thesis, a standard cluster

with five nodes was created. One of them was used to deploy an application and the rest for load testing which needed one master and three workers. Once the cluster was created, the command `kubectl get nodes` was used to list all the nodes, as shown below in Listing 10. The Cloud Shell Editor offered by GCP was used to connect to it for further operations.

```
xiaouuvivian_it@cloudshell:~ (metropolia-thesis)$ kubectl get nodes
NAME                                STATUS    ROLES    AGE      VERSION
gke-load-testing-cluster-default... Ready    <none>   5m25s   v1.19.7-gke.2503
gke-load-testing-cluster-default... Ready    <none>   5m24s   v1.19.7-gke.2503
gke-load-testing-cluster-default... Ready    <none>   5m24s   v1.19.7-gke.2503
gke-load-testing-cluster-default... Ready    <none>   5m23s   v1.19.7-gke.2503
gke-load-testing-cluster-default... Ready    <none>   5m25s   v1.19.7-gke.2503
```

Listing 10. Output of `kubectl get nodes` to show created cluster nodes.

Since the Helm charts with the latest code commit were uploaded to the Helm Chart Repo in GitLab, the installation of the applications was done simply with a few commands:

- `helm repo add helm-repo-golang https://youqins.gitlab.io/lms_golang/`
- `helm repo add helm-repo-java https://youqins.gitlab.io/lms_java/`
- `helm install lottery-service-golang helm-repo-golang/lottery-service-golang`
- `helm install lottery-service-java helm-repo-java/lottery-service`

The helm repo for each application was created inside its project's repository in GitLab. The first two commands were used to add the paths to the helm repos for the two micro-services. Once configured, Kubernetes knew where to download the Helm charts for cloud installation. The applications were installed with the third and fourth command separately.

```
xiaouuvivian_it@cloudshell:~ (metropolia-thesis)$ kubectl get pod
NAME                                READY    STATUS    RESTARTS   AGE
lottery-service-java-5f6d97d6f5-175xb 1/1      Running   0           48s
```

Listing 11. Output of `kubectl get pod` to show all running pods.

After the installation had been completed, the status of the service was checked with the command `kubectl get pod`. The output lists all the running pods and their detailed information. As shown in Listing 11, one pod of `lotter-service-java` has been ready and running for 48 seconds. In addition, the number for restart of the pod was 0, which meant the application had not been restarted so far.

4.4 Load Testing

Load testing is defined as generating large waves of concurrent user requests towards a target application to simulate real-life usage as a way to assess its performance (Wescott, 2013). Load testing, which is a standard way to measure the behavior of a software project under load in the industry, is considered a crucial part in the evaluation of large-scale software systems because various kinds of potential issues related to load can be discovered when big scale requests are received by the system synchronously. (Chen et al., 2017.)

Locust⁵ is an open-source tool designed to create realistic loads of user requests against a dynamic website. In most cases, it is used to test web services, but that does not exclude its ability to work with other systems or protocols. (Locust, 2021.) The tool allows testers to decide the number of users to generate and the spawn rate through a simple and user-friendly web interface, on which the testing progress and real-time data are displayed in separate graphs.

```
xiaouuvivian_it@cloudshell:~ (metropolia-thesis)$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
locust-master-5675988db9-572n4      1/1     Running   0           109s
locust-worker-699c9bbd9-55b8z       1/1     Running   0           109s
locust-worker-699c9bbd9-8xk9p       1/1     Running   0           108s
locust-worker-699c9bbd9-h7j7x       1/1     Running   0           108s
lottery-service-golang-788f69ddc     1/1     Running   0           2m51s
```

Listing 12. Installation of Locust and target app on different nodes.

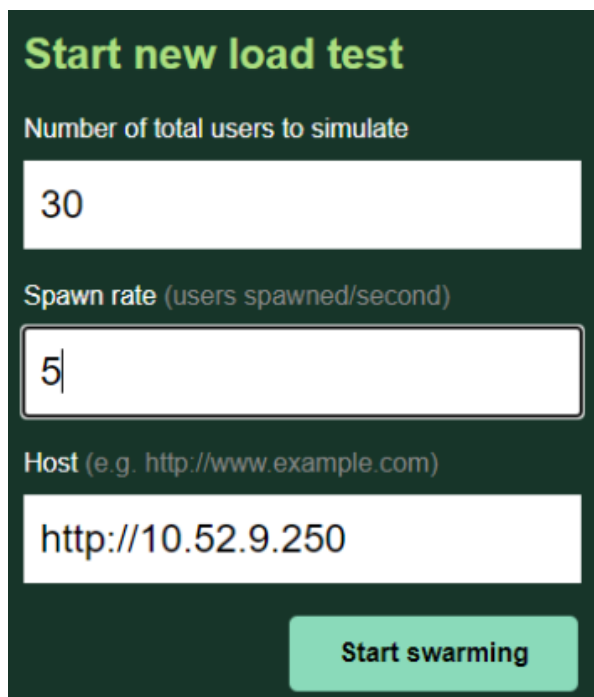
In this project, load testing was used to measure the availability, efficiency, resource consumption and performance of the two applications with identical functionalities but developed with different programming languages. Locust was the tool installed to carry out such testing. To ensure the accuracy of results, the testing was performed on one application at a time with the application installed on a separate node from Locust, as shown in Listing 12.

```
xiaouuvivian_it@cloudshell:~ (metropolia-thesis)$ kubectl get svc
NAME            TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(s)          AGE
Kubernetes     ClusterIP      10.52.0.1    <none>        443/TCP          31m
locust          LoadBalancer   10.52.13.122 34.67.196.254 5557:30685/TCP   54s
lottery-service... ClusterIP      10.52.8.52   <none>        80/TCP           2m56s
```

⁵ <https://locust.io/>

Listing 13. Output of `kubectl get svc` to show information of all running services.

As displayed in Listing 13, after the application was successfully installed on the cluster, the `kubectl get svc` command was used to expose its Cluster-IP. This IP was used as the target for Locust to send user requests to. In addition, Locust was configured to have an External-IP, which was used to open in a web browser locally to set the number of users and spawn rate against the application.



The image shows a dark-themed web interface for starting a new load test. At the top, it says "Start new load test" in green. Below that, there are three input fields: "Number of total users to simulate" with the value "30", "Spawn rate (users spawned/second)" with the value "5", and "Host (e.g. http://www.example.com)" with the value "http://10.52.9.250". At the bottom right, there is a green button labeled "Start swarming".

Figure 9. Number of users and spawn rate for loading testing.

The number of total users to simulate was set to 30 and the spawn rate to 5 per second for the testing of both applications in this project, as displayed in Figure 9. The request was sent to `/lmg` without any parameters, and the default response of one set of numbers for the Finnish Lotto was returned. The testing for both applications lasted 10 minutes. The test progress and the outcome data were accessible from the same user interface of Locust.

4.5 Outcomes and Analysis

Based on the experience of the projects implemented for this thesis, the programming process with Java was slightly smoother partly due to the prior experience the developer

had with the language. Nevertheless, Go was not a difficult language to learn by following the tutorials offered by golang.org⁶. The coding process took actually shorter time compared to the Java project since there was no change in the application logic and the number of code lines was similar. The exact statistics of the projects are listed in Table 3 for comparison. The number of lines of code for the two projects were quite close. However, a relatively big difference in the sizes of the Docker images was observed, with the Java application over 43 times bigger. In addition, its binary file was more than twice the size of the Go app.

Table 3. Statistics about the two applications.

Item	Java App	Go App
Lines of code	403	351
Binary file size	16.55M	7.2M
Docker image size	318MB	7.38MB

During load testing, the Total Requests per Second and the Response Times over time from the applications were captured. The overall Request Statistics and Response Time Statistics were presented in the Locust Test Report as well. The resource usage metrics were available in GKE Workloads on GCP. The data from the above-mentioned resources were examined for both the Java and the Go applications.

⁶ <https://tour.golang.org/list>

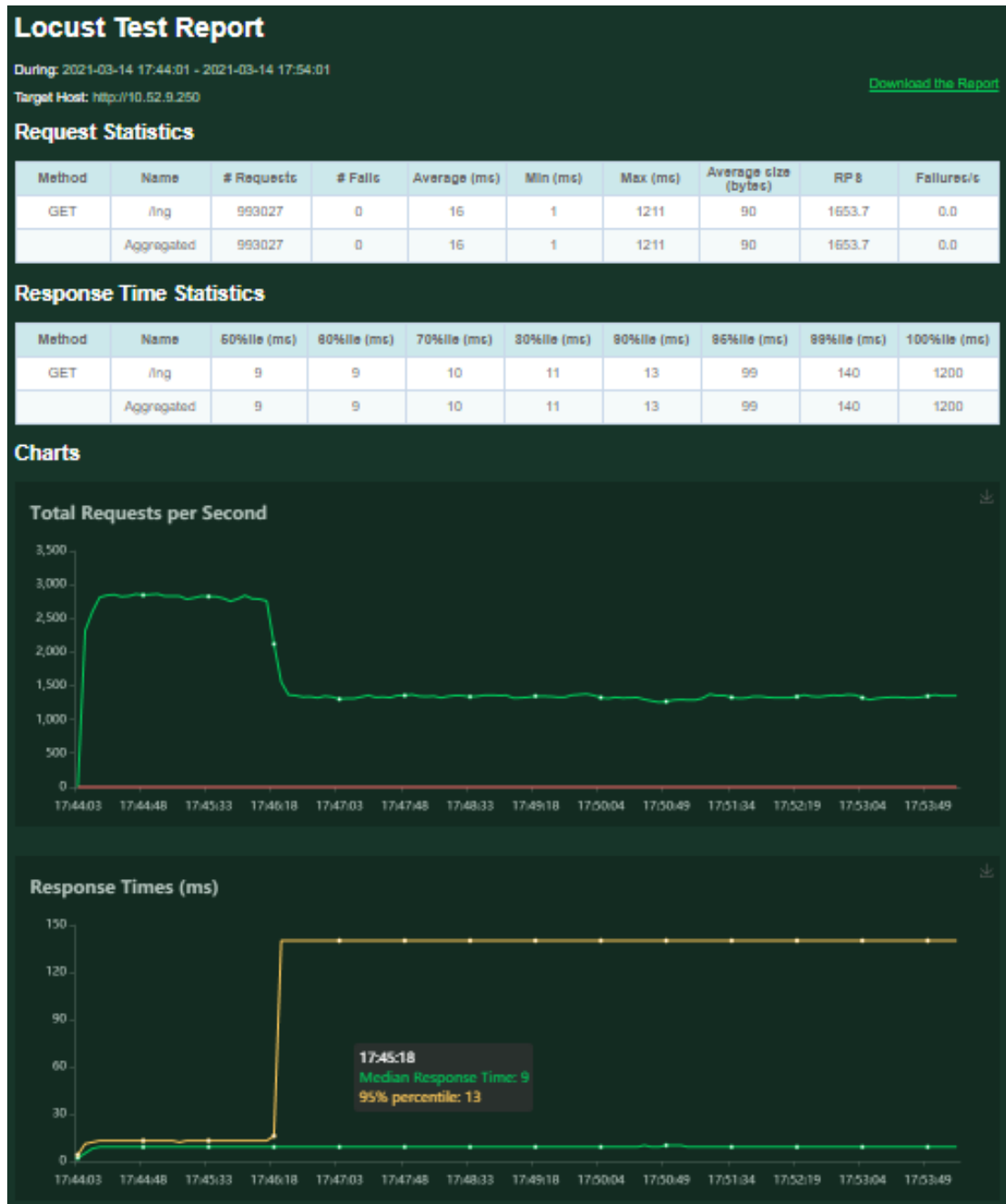


Figure 10. Locust test report of Go app.

As displayed on the Locust Test Report of the Go application in Figure 10, the total number of requests processed reached almost one million during the 10-minute testing, out of which 0 failure was recorded. It took less than 20 seconds for the service to be able to handle almost 3000 requests per second. However, it experienced a sudden drop after two minutes to around 1300 until the end of the process. For each second, the system was able to process more than 1600 requests on average. The response time started at

17ms, but the number increased to 140 after two minutes and remained at this level for the rest 8 minutes.

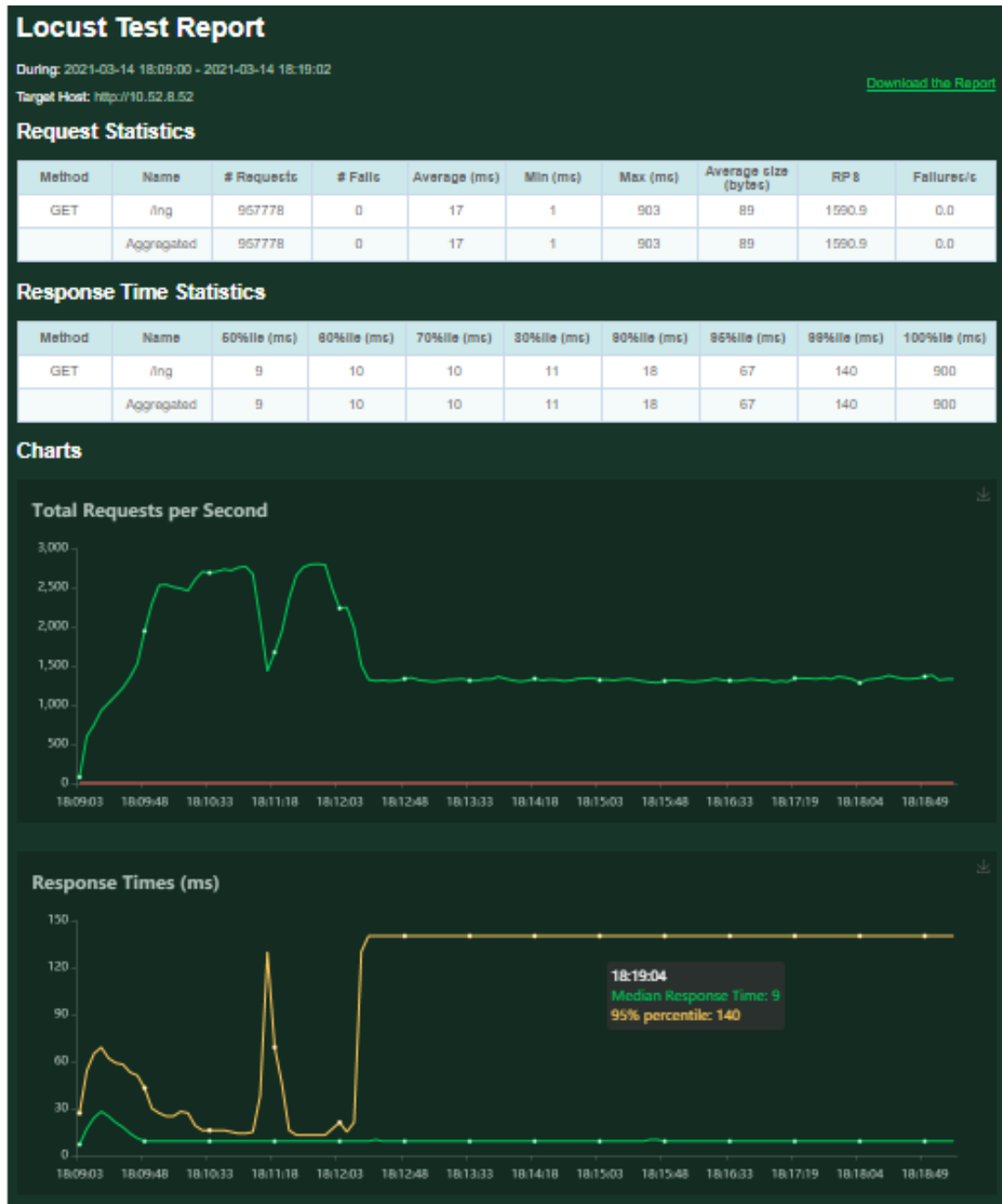


Figure 11. Locust test report of Java app.

The Test Report of the Java application is displayed in Figure 11. On the report, the total number of requests was 957,778 and all requests received successful responses. The

average value for request per second reached almost 1600. At the beginning of the testing, the graphs of request and response demonstrated sharp rises and drops before they reached a stable level.



Figure 12. CPU usage of Go app (left) and Java app (right).

The central processing unit (CPU) usage of both applications is displayed in Figure 12. The metrics on the left represents the data of the Go app and the one on the right side is for the Java app. CPU usage of the Go application never went over 0.25 and the average value was around 0.15. While the CPU usage of the Java software system went up to almost 1 at its peak and the average was between 0.4 and 0.5.

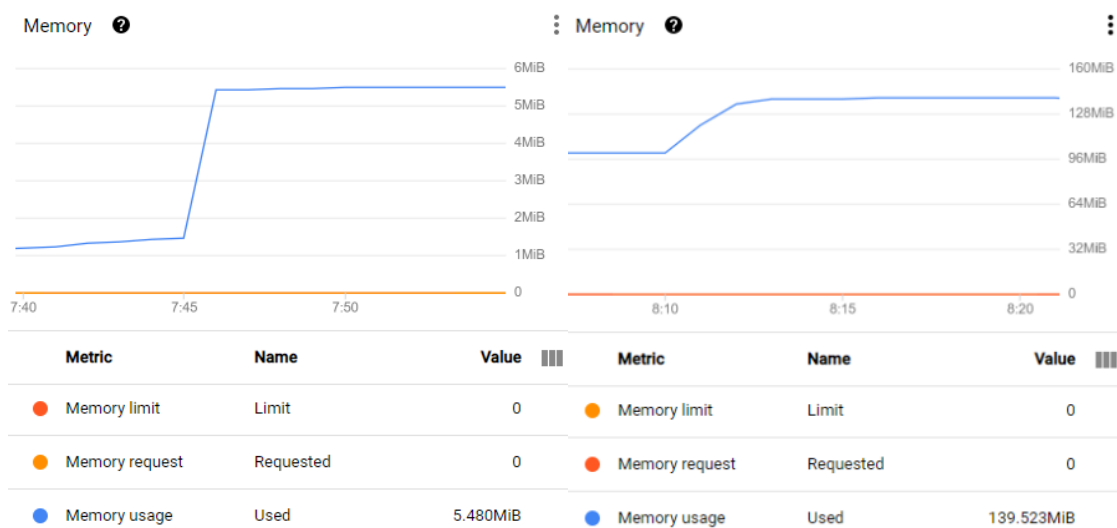


Figure 13. Memory usage of Go app (left) and Java app (right).

The memory usage metrics of the two systems are shown in Figure 13. The memory used by the Go project was lower than 2 Mib during the first 5 minutes. Despite the sudden rise afterwards, the number was lower than 6MiB at the highest level. Whilst the memory usage of the Java application started at number higher than 96MiB. Also, it went over 128MiB during most of the time of the testing.



Figure 14. Disk usage of Go app (left) and Java app (right).

The disk usage metrics of both microservices are demonstrated in Figure 14. The values remained at a constant level. No change was observed in the numbers during the testing. The limit for volume capacity was 843MiB. In addition, 12.000KiB was used by each application. There was no difference in this regard between the two software systems.

From the statistics displayed in the above Locust Test reports in Figure 10 and Figure 11, the overall performance outcomes of the two microservices were both satisfactory. They were both considered reliable since no crash was observed during the testing cycle. However, the Go application was able to handle over 35,000 more requests during the 10-minute test and all the numbers about response time were lower than those of the Java microservice. It was also observed that the Go app took less time to gain the ability to handle a large number of user requests. However, the developer was not able to understand the sudden drop in the number of requests displayed on the Total Request per Second graph. More investigation and testing were needed to discover the possible root causes.

The exact average values of resource consumption for the whole testing cycle were not available from the GCP service. However, the data displayed on the graphs on the metrics in Figure 12 and Figure 13 revealed quite big differences in CPU and memory usage by the two microservices. The numbers on the graphs showed that the CPU consumed by Java the application was nearly two times higher. In addition, it used nearly 20 times more memory resources. Further investigation was needed to decide how much this is related to the Java programming language and the Spring framework used for the development.

5 Conclusion

This thesis aimed to discover the advantages and challenges of different programming languages in terms of microservice development and cloud deployment. Two applications were planned to be developed for identical functions but with different programming languages, one with Java and the other with Go, for cloud deployment to the GKE on the GCP with Docker, Helm and Kubernetes. To achieve these objectives, two applications were successfully developed and deployed to the target cloud platform. Moreover, the advantages and challenges of each language in this regard were analyzed thoroughly from different perspectives. The measurements included the programming experience with both languages, the packaging procedures as well as the cloud deployment processes. Most importantly, the performance outcomes and resource consumption of both projects were tested and analyzed after they were deployed to the cloud.

In terms programming languages, as one of the most popular and well-paid programming languages among developers (North Eastern University, 2021), Java has a relatively longer history and bigger community. The language features a comparatively bigger number of libraries, frameworks and system tools available for project development. Thus, Java represents a good choice to build large-scale applications with complicated features (Chang et al., 2019). Furthermore, Java can define business logic in a way that cannot be handled by simple operators (Lee, Kang and Hur, 2012).

Go on the other hand offers a way to write simpler code due to its feature of encapsulation with structs and built-in packages. In addition, Go is a compiled language, which makes it a very fast language ideal for building software systems that are demanding on efficiency and reliability. (Yasir et al., 2019.) Some of the most popular cloud platforms are written in Go, including Docker and Kubernetes (GoLang, 2021), which makes it increasingly popular for cloud computing.

However, the most important findings of this thesis concluded that the sizes of both the binary file and the Docker image of the application written in Go were much smaller than the Java project, despite the fact that the number of lines of code for the two projects were very close. After being deployed to the GCP, the Go software system required much less computing resource when running in the cloud. Furthermore, it took less time to pick up a relatively large number of user requests and the charts for user request handling during the load testing process were less volatile. More importantly, the system

was able to handle more user requests within the same time duration and the response time was shorter compared to the application written in Java. With all those discoveries in mind, it is safe to say the application developed with Go for this thesis was more efficient and cost effective.

Nevertheless, the exact strengths and weaknesses of different programming languages in this regard may vary largely due to differences in purpose, use case, complexity, scale and many other aspects of the application itself. Each programming language features different advantages and offers a different number of supporting libraries and frameworks. Therefore, extensive research and pre-testing on how to magnify the advantages and mitigate the drawbacks of those aspects for the overall benefits of the target application is strongly advised prior to the implementation of such projects.

References

A. Bieniusa, P. Thiemann and S. Wehr, "The Relation of Version Control to Concurrent Programming," 2008 International Conference on Computer Science and Software Engineering, Wuhan, China, 2008, pp. 461-464.

A. Gupta, P. Goswami, N. Chaudhary and R. Bansal, "Deploying an Application using Google Cloud Platform," 2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bangalore, India, 2020, pp. 236-239.

Apache Maven. 2021. What is maven? Online. <<https://maven.apache.org/what-is-maven.html>>. Accessed: 5 January 2021.

Amazon Web Service. 2020. What is cloud computing? Online. <<https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/>>. Accessed: 3 October 2020.

Anshul Jindal, Vladimir Podolskiy, and Michael Gerndt. 2019. Performance Modeling for Cloud Microservice Applications. In <i>Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering</i> (<i>ICPE '19</i>). Association for Computing Machinery, New York, NY, USA, 25–32.

AWS. 2020. What is cloud computing? Online. <<https://aws.amazon.com/what-is-cloud-computing/>>. Accessed: 10 December 2020.

Citrix. 2020. What is containerization and how does it work? Online. <<https://www.citrix.com/glossary/what-is-containerization.html>>. Accessed: 5 October 2020.

CNCF. 2021. Helm Project Journey. Online. <<https://www.cncf.io/cncf-helm-project-journey-report/>>. Accessed: 5 January 2021.

Docker. 2020. What is a Container? Online. <<https://www.docker.com/resources/what-container/>>. Accessed: 27 September 2020.

Fei Li, Joachim Fröhlich, Daniel Schall, Markus Lachenmayr, Christoph Stückjürgen, Sebastian Meixner, and Frank Buschmann. 2018. Microservice Patterns for the Life Cycle of Industrial Edge Software. In <i>Proceedings of the 23rd European Conference on Pattern Languages of Programs</i> (<i>EuroPLoP '18</i>). Association for Computing Machinery, New York, NY, USA, Article 4, 1–11.

Feilong Wang. 2019. The Journey of Cloud Computing with Open Source. In <i>Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion</i> (<i>UCC '19 Companion</i>). Association for Computing Machinery, New York, NY, USA, 167.

G. Avino, M. Malinverno, F. Malandrino, C. Casetti, and C. F. Chiasserini. 2017. Characterizing Docker Overhead in Mobile Edge Computing Scenarios. In <i>Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems</i> (<i>HotConNet '17</i>). Association for Computing Machinery, New York, NY, USA, 30–35.

GKE. 2021. GKE overview. Online. <<https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview>>. Accessed: 1 April 2021.

GoLang. 2021. Go Documentation. Online. <<https://golang.org/doc/>>. Accessed: 12 March 2021.

Google. 2020. containers at Google. Online. <<https://cloud.google.com/containers>>. Accessed: 27 September 2020.

Google Cloud. 2021. Google Cloud Products. Online. <cloud.google.com>. Accessed: 19 February 2021.

Helm. 2021. What is Helm? Online. <<https://helm.sh/>>. Accessed: 6 January 2021.

H. I. Moud, J. Hariharan, H. Hakim, C. Kibert and I. flood, "Sustainability Assessment of Data Centers Beyond LEED," 2020 IEEE Green Technologies Conference (GreenTech), Oklahoma City, OK, USA, 2020, pp. 62-64.

H. Safari, N. Sabri, F. Shahsavan and B. Bahrak, "An Analysis of GitLab's Users and Projects Networks," 2020 10th International Symposium on Telecommunications (IST), Tehran, Iran, 2020, pp. 194-200.

H. Sami and A. Mourad, "Towards Dynamic On-Demand Fog Computing Formation Based On Containerization Technology", 2018 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2018, pp. 960-965.

IBM. 2021. What is Cloud Computing? Online. <<https://www.ibm.com/cloud/learn/cloud-computing>>. Accessed: 11 March 2021.

IBM. 2020. What is containerization? Online. <<https://www.ibm.com/cloud/learn/containerization>>. Accessed: 5 October 2020.

IBM. 2021. Application Programming Interface (API). Online. <<https://www.ibm.com/cloud/learn/api>>. Accessed: 2 April 2021.

Jihyun Lee, Sungju Kang and Sung Jin Hur, "Web-based development framework for customizing Java-based business logic of SaaS application", 2012 14th International Conference on Advanced Communication Technology (ICACT), PyeongChang, 2012, pp. 1310-1313.

Khan, A. 2017, "Key Characteristics of a Container Orchestration Platform to Enable a Modern Application", IEEE Cloud Computing, vol. 4, no. 5, pp. 42-48.

Kubernetes. 2020. What is Kubernetes? Online. <<https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>>. Accessed: 7 October 2020.

Linlin Tang, Pingfei Ren and Jengshyang Pan, "An improved k-subset algorithm for load balance problems in Cloud Computing," 2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems, Shenzhen, China, 2014, pp. 175-179.

Locust. 2021. What is Locust? Online. <<https://docs.locust.io/en/stable/what-is-locust.html>>. Accessed: 19 February 2021.

M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen and M. Villari, "Open Issues in Scheduling Microservices in the Cloud," in IEEE Cloud Computing, vol. 3, no. 5, pp. 81-88, Sept.-Oct. 2016.

M. Gajewski and W. Zabierowski, "Analysis and Comparison of the Spring Framework and Play Framework Performance, Used to Create Web Applications in Java," 2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH), Polyana, Ukraine, 2019, pp. 170-173.

M. Rahman and J. Gao, "A reusable automated acceptance testing architecture for microservices in behavior-driven development", Service-Oriented System Engineering (SOSE) 2015 IEEE Symposium, pp. 321-325, 2015.

Microsoft. 2020. What is cloud computing? Online. <<https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/>>. Accessed: 4 October 2020.

Microsoft Azure. 2021. What is the cloud? Online. <<https://azure.microsoft.com/en-us/overview/what-is-the-cloud/>>. Accessed: 11 March 2021.

Muhammad Ali Babar and Muhammad Aufeef Chauhan. 2011. A tale of migration to cloud computing for sharing experiences and observations. In <i>Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing</i> (<i>SE-CLOUD '11</i>). Association for Computing Machinery, New York, NY, USA, 50–56.

N. Alshuqayran, N. Ali and R. Evans, "Towards Micro Service Architecture Recovery: An Empirical Study," 2018 IEEE International Conference on Software Architecture (ICSA), Seattle, WA, USA, 2018, pp. 47-4709.

N. F. Efozia, E. Ariwa, D. C. Asogwa, O. Awonusi and S. O. Anigbogu, "A review of threats and vulnerabilities to cloud computing existence," 2017 Seventh International Conference on Innovative Computing Technology (INTECH), Luton, UK, 2017, pp. 197-204.

N. Shuping and W. Feng, "The Network Architecture Design of Distributed Dual Live Data Center," 2019 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS), Shenyang, China, 2019, pp. 638-642.

Nicholas J. Mitchell and Kazi Zunnurhain. 2019. Google cloud platform security. In <i>Proceedings of the 4th ACM/IEEE Symposium on Edge Computing</i> (<i>SEC '19</i>). Association for Computing Machinery, New York, NY, USA, 319–322.

NorthEastern University. 2021. The 10 Most Popular Programming Languages to Learn in 2021. Online. <<https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/>>. Accessed: 10 March 2021.

Oracle. 2021. Java Programming Language. Online. <<https://www.oracle.com/java/>>. Accessed: 4 March 2021

phoenixNAP. 2020. Docker Images vs Containers. Online. <<https://phoenixnap.com/kb/docker-image-vs-container>>. Accessed: 27 September 2020.

P. P. Ray, "An Introduction to Dew Computing: Definition, Concept and Implications," in IEEE Access, vol. 6, pp. 723-737, 2018.

RedHat OpenShift. 2020. Docker Images. Online. <https://docs.openshift.com/enterprise/3.0/architecture/core_concepts/containers_and_images.html#docker-images>. Accessed: 27 September 2020.

Redhat Containers. 2020. What is container orchestration? Online. <<https://www.redhat.com/en/topics/containers/what-is-container-orchestration>>. Accessed: 6 October 2020.

Redhat Microservices. 2020. What are microservices? Online. <<https://www.redhat.com/en/topics/microservices/what-are-microservices>>. Accessed: 6 October 2020.

R. M. Yasir, M. Asad, A. H. Galib, K. K. Ganguly and M. S. Siddik, "GodExpo: An Automated God Structure Detection Tool for Golang," 2019 IEEE/ACM 3rd International Workshop on Refactoring (IWor), Montreal, QC, Canada, 2019, pp. 47-50.

Salman Ahmad and Sepandar Kamvar. 2013. The dog programming language. In <i>Proceedings of the 26th annual ACM symposium on User interface software and technology</i> (<i>UIST '13</i>). Association for Computing Machinery, New York, NY, USA, 463–472.

Serjik Dikaleh, Ozair Sheikh, and Chris Felix. 2017. Introduction to kubernetes. In <i>Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering</i> (<i>CASCON '17</i>). IBM Corp., USA, 310.

Spring. Why Spring? Online. <<https://spring.io/why-spring>>. Accessed: 3 March 2021.

S. S. Brimzhanova, S. K. Atanov, Moldamurat Khuralay, K. S. Kobelev, and L. G. Gagarina. 2019. Cross-platform compilation of programming language Golang for raspberry pi. In <i>Proceedings of the 5th International Conference on Engineering and MIS</i> (<i>ICEMIS '19</i>). Association for Computing Machinery, New York, NY, USA, Article 10, 1–5.

TIOBE Index, TIOBE Index for March 2021. 2021. Online. <<https://www.tiobe.com/tiobe-index/>>. Accessed: 12 March 2021.

Tse-Hsun Chen, Mark D. Syer, Weiyi Shang, Zhen Ming Jiang, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. 2017. Analytics-driven load testing: an industrial experience report on load testing of large-scale systems. In <i>Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track</i> (<i>ICSE-SEIP '17</i>). IEEE Press, 243–252.

W. Li and X. Wan, "An Analysis and Comparison for Public Cloud Technology and Market Development Trend in China," 2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), Shanghai, China, 2018, pp. 200-205.

Wang Bin, Yang Shulin, Ren Xuelei, and Wang Guyang. 2017. Research on Digital Publishing Application System Based on Micro-Service Architecture. In <i>Proceedings of the 2017 VI International Conference on Network, Communication and Computing</i> (<i>ICNCC 2017</i>). Association for Computing Machinery, New York, NY, USA, 140–144.

Wescott, Bob. 2013. The Every Computer Performance Book, Chapter 6: Load Testing. CreateSpace.

Yee-Kang Chang, Patrick Tiu, Eric Lau, Leo Christy Jesuraj, Alvin So, and Gilbert Kwan. 2019. Hands-on workshop on fast, efficient & seriously open cloud-native Java. In

Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering (*CASCON '19*). IBM Corp., USA, 373–375.