Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Networked Systems and Services

Cloud Native ChatOps Development

Bachelor of Profession's Thesis

Author
Ádám Adamek

Advisors
dr. Károly Farkas
Gergely Szabó

May 29, 2022

MŰEGYETEM 1782

SZAKDOLGOZAT FELADAT

Adamek Ádám
szigorló üzemmérnök-informatikus hallgató részére

Cloud-Native ChatOps Development

IT cloud (both public and private) became ubiquitous in the last decade. Cloud technologies made it possible to roll out application releases quickly, scale the application capacity almost endlessly while reaching unprecedented availability. The most successful branch of cloud technologies is referred to as cloud-native, where the application components are containerized, and the containers are orchestrated by a central orchestrator. Cloud native-techniques rely on automation to a large extent. The cloud-native application development and operations are often following the GitOps and ChatOps practices.

The candidate's assignment is to study and acquire cloud-native tools and architectures. Thus, in the frame of the bachelor thesis work, the following tasks have to be completed:

- Study the state-of-the-art cloud native tools and architectures;
- Develop a simple chatbot application that follows the microservices architecture and 12-factor principles, as it is stateless, containerized, etc. It shall be capable to perform basic operations;
- Implement a CI solution that automatically tests, builds container images of the application components;
- Implement a CD solution that automatically deploys the application components into the provided Kubernetes-based cloud platform;
- Integrate the application with the platform monitoring and logging solutions;
- Perform measurements to validate and demonstrate the following workflow:
    - Observe the load the application experiences with the help of monitoring dashboards;
    - Mitigate the effects of increased load by horizontally scaling the application;
    - Adopt Kubernetes-provided auto-scaling solutions and measure its behavior under different traffic conditions.
- Document the results and learnings.

Egyetemi témavezető: Dr. Farkas Károly, egyetemi docens, BME-VIK HIT tanszék

Ipari konzulens: Szabó Gergely, szoftver mérnök, Origoss Solutions Ltd.

Budapest, 2022. február 11.

Dr. Imre Sándor
egyetemi tanár
tanszékvezető

Témavezetői vélemények:

Egyetemi témavezető: □ Beadható, □ Nem beadható, dátum: aláírás:

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

1117. Budapest, Magyar tudósok körútja 2. I. Ép. B. 121.
Telefon: 463-3261 Fax: 463-3263
URL: http://www.hit.bme.hu

HALLGATÓI NYILATKOZAT

Alulírott *Adamek Ádám*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2022.05.29

Adamek Ádám
hallgató

Contents

Kivonat

A felhőalapú számítástechnika nagyléptékű fejlődése következtében mára már mindennapivá váltak azok az eszközök, amelyek lehetővé teszik a felhőalapú számítási kapacitások széles körben való felhasználását. A felhőalapú számítási kapacitások használatának növekedése többek között olyan előnyeinek köszönhető, mint az igény szerinti erőforrás biztosítása vagy a magas arányú rendelkezésre állási idő. Az ebben rejlő szinte végtelen lehetőségeket az egyik legjobban kihasználó koncepció a cloud-native. Egy cloud-native alkalmazást jellemez a konténerizáció, a magas rendelkezésre állás és a könnyű gyors skálázhatóság. A meglévő kódbázis felhő környezetbe költöztetése előtt érdemes azonban alaposan kielemezni, a szolgáltató által ajánlott feltételeket és felkészíteni a már meglévő ökoszisztémát a váltásra, hogy az minél zökkenőmentesebb legyen és ne essen áldozatul a cég a Vendor lock-in jelenségnek.

Napjainkban egyre inkább elterjedt a ChatOps koncepció, melynek lényege, hogy a különböző platformokhoz köthető munkafolyamatokat képes becsatornázni egy chat applikációba egy vagy több chatbot felhasználásával, ezzel megkönnyítve a kommunikációt és növelve a munkafolyamatok hatékonyságát. Egy chatbot pedig jól adaptálható cloud-native módszerekkel.

A szakdolgozat célja bemutatni azokat a technológiákat, illetve technikákat, amelyek lehetővé teszik valamint megkönnyítik a felhőkörnyezetben való fejlesztési, üzemeltetési feladatokat. A szakdolgozat első része felhő környezethez magához logikailag legszorosabban kapcsolódó eszközöket, különösképpen a Kubernetest, illetve a Vendor Lock-in jelenséget mutatja be. A második rész a fontosabb kód integrációs, illetve applikáció üzemeltetési technikákat és eszközöket foglalja össze. A harmadik rész a cloud-native koncepció elemeit és felépítését prezentálja. A negyedik rész pedig az előzőekben bemutatott technológiák és eszközök működését mutatja be egy olyan chatbot alkalmazáson keresztül, amely ChatOps műveleteket is végez.

Abstract

Due to the large-scale evolution of the IT cloud, many available tools enable the widespread use of cloud-based computing resources by now. An increase in the use of cloud computing resources is due to those benefits, among others, such as providing resources on-demand or high availability time. One of the concepts which successfully exploits almost endless possibilities inherent in this is the cloud-native. A cloud-native application is characterized by containerization, high availability, and hands down scalability. However, before the migration of the existing code base into the cloud environment, it is worth analyzing thoroughly the conditions recommended by the cloud provider, and preparing the existing ecosystem for change, so the company doesn't fall victim to the phenomenon of Vendor lock-in.

The ChatOps concept is becoming more prevalent nowadays. The essence of this concept is to bring different workloads, which are attached to different platforms, into one chat application, using one or more chatbots, thus facilitating communication and increasing the efficiency of daily work. And a chatbot could be well adapted with Cloud-native methods.

This thesis aims to introduce those technologies and techniques which facilitate the development and operation tasks in a cloud environment. The first part of this thesis introduces the most related tools to the cloud environment, especially with regard to the Kubernetes, and presents the phenomenon of Vendor Lock-in. The second part summarizes the key code integration and application operating techniques and tools. The third part presents the components and architecture of the cloud-native concept. The fourth section demonstrates the potential of the technologies described above in practice through a chatbot application that performs ChatOps operations.

Chapter 1

Introduction

The constantly growing popularity of IT cloud and technologies built around this platform have made cloud computing commonly used in the last few years. These technologies gave the opportunity to achieve high availability, scale application resources on demand, deliver new features to end users as quickly as possible. This thesis aims to introduce the aforementioned technologies.

In Chapter 2 besides phenomenon of Vendor lock-in, tools specific to cloud environment will be introduced. These tools are either dedicated to complete certain task for instance monitoring the application in cloud environment with Prometheus or establishes possibility for maximizing the potential of the cloud environment such as containerization. Furthermore this chapter dealing with two very popular cloud platform, and their managed Kubernetes service in only a few words. This chapter however mostly dedicated to introduce modern container orchestration tool, called Kubernetes on the most basic level.

In Chapter 3 automated tools and practices are introduced with help of which a modern CI/CD pipeline can be built. This Chapter summarizes the core of DevOps and Chatops practices and listing advantages of their usage.

Chapter 4 characterizes cloud-native application, in regard to its software architecture, used ecosystem and presents best practices of the practical side of cloud-native concept's implementation.

In Chapter 5 the tools and concepts introduced to this point of this thesis will be presented through an application.

Lastly, in Chapter 6, conclusions will be drawn based on the experiences gathered in previous chapters.

Chapter 2

Cloud environment

2.1 Cloud and phenomenon of Vendor lock-in

Although it is discussed frequently, at this chapter of this thesis presenting what cloud is, in only a few words, feels necessary. Majority of the tools introduced through upcoming sections here, are made for leveraging benefits comes from using cloud compute resources and to avoid obstacles what may arise while migrating infrastructure to cloud.

2.1.1 Cloud

Cloud computing resources and services together, which can accessed over the internet often referred as "the Cloud" [20]. Resources and services cover servers, networking, databases, software etc.. The underlying infrastructure of cloud is placed in data centers all around the world and it is owned, operated and maintained by cloud providers like Amazon Web Services(AWS) or Google Cloud Platform (GCP) [23] . Using cloud is cost effective in a way that it is not required to set up and maintain the infrastructure, including different kind of software and hardware belonging to on-site data centers. In addition to that, using cloud providers resources spares the company a more significant initial expense, which is otherwise inevitable, so it makes technologies available to smaller companies. Computing resources can be used on demand, in consequence of that flexibility grows, and it also reduces cost by using only the required amount of resources at the time.IT team can focus all their effort on the products, so using cloud infrastructure increases productivity [20].

So because of all the aforementioned benefits, the popularity of using cloud environment for various purposes has grown immensely.

2.1.2 Vendor lock-in

Although benefits of using cloud has become widely known, there are some drawbacks along the way.Vendor lock-in means that a company is "forced" to use a single cloud provider service due to the cost that switching to another cloud provider would cause substantial cost or serious technical incompatibilities. The problem is rooted in the heterogeneity of cloud providers implementation. Majority cloud Providers offer solutions that are specific to their platform, which usage leads towards deeper integration and causes difficulties when a business partner initiates migrating its code to another cloud providers platform [62].

In order to avoid Vendor lock-in, it is important to choose technologies for use consciously from the beginning. Muhammad Raza listed some actions and strategies which help to prevent Vendor lock-in; however he states there is "no magic bullet for avoiding lock-in." [3].

1. Identify complex dependencies
2. Understand the commonalities
3. Consider upgrading before migrating
4. Educate stakeholders
5. Make apps portable, aligned with open standards
6. Employ modern SDLC(Software Development Life Cycle) methodologies
7. Ensure portability once migrated
8. Develop a clear exit strategy
9. Consider a multi-cloud strategy
10. Do your due diligence

He claims IT workloads operate on legacy technologies, comes with a great limitation when the time comes to choose a vendor for migrating code to. He also states the key for making the best choice is to compare what is compatible across cloud providers, regarding the company's existing technology stack, and technical requirements, if there is not any, further inspection needed to avoid lock-in considering IT strategies etc.. Upgrading code before migrating to cloud helps to increase compatibility in order to avoid limitations in choosing new vendor. Good communication with stakeholders about technical requirements is essential in a way that bringing technical needs and business purposes under the same roof can be beneficial to avoid lock-in. An application that using open-sourced technologies instead of custom solutions is not likely ends up locked-in, due to open standards are supported by many vendors across various industry verticals. Applying design by modern SDLC methodologies, particularly DevOps and its practices like Infrastructure as Code(IaC) directing towards a state of technology-independent design. Evaluation of fees belongs to data migration between vendors is a crucial point before making the decision to use cloud compute resources. Developing an exit strategy especially determining exit's cost, involving both parties (customer, vendor) is essential; without it customers can experience too high exiting fees, which also can lead to Vendor lock-in.Developing applications that can be adjusted to a multi-cloud strategy can help to avoid situation Vendor lock-in from the very beginning by design. Under do your diligence; he listed some things which should be done in favor of a better understanding of possibilities. Understanding business and technology requirements for making the right choice, or evaluating vendor's service history etc [3].

Since the phenomenon of Vendor lock-in has been known for a long time, multiple solution or strategy has come up as a solution which, according to Gregor Hohpe, formerly a technical director with Google Cloud leads up to different kind of difficulties [2]. So the desire to avoid lock-in leads to actions where avoiding lock-in itself becomes a burden that is hard to carry. He suggests possible solutions with a detailed explanation, but first, he listed some further lock-in types to support complete understanding, which are the followings [2]:

1. Product Lock-in

   He states using open source products may help avoid Vendor lock-in, but it creates a dependency on specific product's API(Application Programming Interface)s, configurations, and features like in case of using Kubernetes or Cassandra [2].

2. Version lock-in

   Version lock-in means being locked in with a specific version of a specific product. Upgrading from that version of the product where with in case it breaks customization and extensions cost too much; therefore; as a result, they can not be carried out [2].

3. Architecture lock-in

   The architecture lock-in situation presented by Gregor Hohpe's example where a transition from small services that expose APIs and can be deployed as containers to a serverless architecture, where changing the granularity of your services closer to single functions, externalizing state management, utilizing an event-architecture and more tasks, make change too difficult [2].

4. Platform lock-in

   He describes platform lock-in as a unique flavor of product lock-in where a platform's additional services, like it is holding your user accounts and associated access rights, security policies, and different kinds of generally proprietary services, so integrated into the product that migrating from that platform to another would be "painful" [2].

5. Skills lock-in

   Skills lock-in describes a situation where developers get familiar with a certain technology and retraining to another technology or product or hiring experts in that technology would cost too much [2].

6. Legal lock-in

   In the case of legal lock-in, he describes the state when different kinds of legal terms such as licensing terms of the licensed software or other agreements with the used platform prevent company's from migrating code to another vendor [2].

7. Mental lock-in

   Mental lock-in is, according to Hohpe, one of the most dangerous. It refers to a bad practice when assumptions based on previous experiences with vendors built into the decision-making process may result in rejecting alternative options. "For example, you may reject scale-out architectures as inefficient because they do not scale linearly (you do not get twice the performance when doubling the hardware). While technically accurate, this way of thinking ignores the fact that scalability, not efficiency, is the main driver [2]."

Hohpe states that being lock-in at some degree is almost inevitable. To decide whether the commitment to a product or an application has a bad influence on the company's operation or it can be acceptable for the benefits it provides in return for commitment. Visualizing the four categories as a two-by-two matrix with the help of which decisions making can be more manageable. The matrix outlines the choices with switching costs on the y-axis and Unique Utility on the x-axis. Switching cost determines how difficult it is to switch the technology, or product to a different solution. Unique Utility defines on

what degree is the technology or product beneficial for the company. Hohpe establishes four categories for products with the help of the aforementioned matrix [2].

1. Disposable

   This category concerns products that usage do not provide much, but they do not have high switching costs either [2].

2. Accepted lock-in

   Products that provide a lot and take a lot in return belong to this category [2].

3. Caution

   Caution category covers the range of products that should be avoided if possible. Applying these products comes with only a few benefits, at a cost where after they are used the possibility of changing them to a substitutionary solution is highly unlikely. Classic lock-in situation [2].

4. Ideal

   Even on the matrix, this category is marked as rare; this includes every technology with many unique utilities, but the cost of leaving these products will not cause serious trouble or even facilitates transition by being compatible with the new solution [2].

2.2 Kubernetes

In this section, a modern and efficient container manager tool will be introduced called Kubernetes. An insight will be given in terms of components, controlling, usage, and basic operations of this prevalent tool.

2.2.1 Introduction

Kubernetes, also known as K8s, is an open-source container manager tool. The motivation to indite Kubernetes is the growing popularity of container-based microservice architectures. Previous to Kubernetes, a great amount of knowledge was gathered at Google running containerized applications, which brought to existence a tool named Borg. Borg is a kind of predecessor to Kubernetes, considering that more than one feature came from Borg are still a fundamental element of Kubernetes today, like Pods and Services. Google open-sourced Kubernetes in 2014 to benefit the broader community outside of Google [43].

2.2.2 Usage

According to Cloud Native Computing Foundation's 2021 survey, 96 % of organizations involved in this survey are either using or evaluating Kubernetes [55]. CNCF began this aforementioned survey in 2016, and this result counts as the highest number so far. It is also interesting to examine the size distribution of respondent companies; the greatest ratio of usage of 20% belongs to company size 10-49 and companies with more than 5000 employees, and company size 50-99 has the lowest usage ratio of all with 7%. As it is written in the latest Cloud Native Development Report, made for CNCF by SlashData,

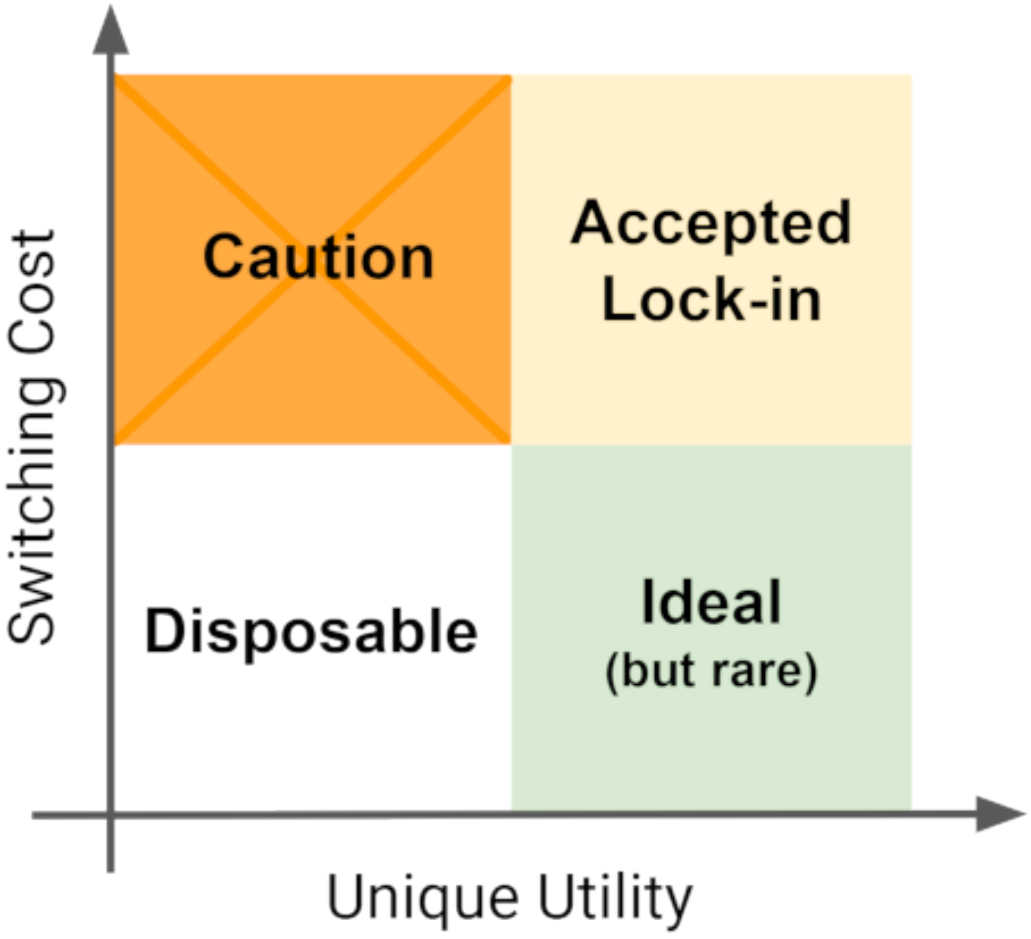

Figure 2.1: Lock-in matrix [2]

approximately 5.6 million developers using Kubernetes today [67]. Documentation and community are also a strong side of Kubernetes. According to a survey created by Aimee Ukasick and SIG Docs in 2019, 47.96% of respondents were Moderately Satisfied with the concepts section, 50.54% were Moderately Satisfied with considering tasks section, and 40.86% Very Satisfied with the reference section, and 47.25% were Moderately Satisfied with the Tutorial section [71].

2.3 Kubernetes components

Cluster is the greatest unit of Kubernetes infrastructure Figure 2.2. A Cluster usually consists of multiple Nodes but it contains at least one Node. A Node can be either a physical machine or a virtual one. When a Node holds a component, which is part of Control Plane, therefore the component is responsible for control, that Node traditionally called Master Node. The other type of Node known as Worker Node is where Pods and containerized applications inside Pods are being placed to run on [40].

2.3.1 Kube-apiserver

API server is part of Control Plane through which communication with Cluster is possible. There are multiple options when a developer or a service intend to carry out operations or query the state of Kubernetes objects. Accessing Kubernetes API, is feasible via HTTP, or with usage of a command line tool like kubectl [40], [7].

2.3.2 Etcd

Etcd is also a control plane component like API server, but it performs very different tasks. Etcd is like the memory of Cluster; it holds all the essential information about the state of Cluster in the key-value store. When it comes to kubectl, get command, etcd, is in charge of answering. Furthermore, when a change occurs, for example, in Deployment's parameter, data stored in etcd serves as a point of reference; what's the next step to do [40].

2.3.3 Kube-scheduler

The kube-scheduler searches for freshly created Pods and assigns them to a Node to run on. While selecting Node for Pod kube-scheduler takes into consideration all of those parameters of a Pod has and based on those parameters and available resources inside Cluster, it places the Pod for the most suitable Node [40].

2.3.4 Kube-controller-manager

Kube-controller-manager is part of the Control Plane as well. The process that makes changes for matching the current state of the Cluster with the desired state is called Controller. Multiple Controllers are compiled in one binary and run in one process. Kube-controller-manager is responsible for running these Controllers [40].

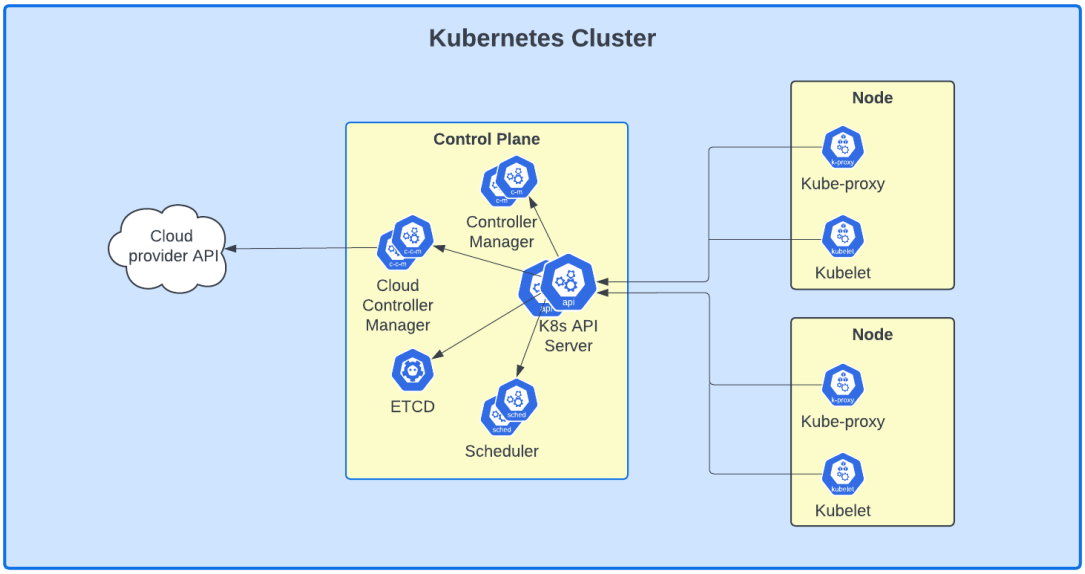

Figure 2.2: Illustration of Kubernetes components [40]

2.3.5 Cloud-controller-manager

Cloud-controller-manager, in short, is a link between Cluster and a cloud provider API. In order to do so, it embeds the cloud-specific control logic. Cloud-controller-manager is only present, in cloud environment and operating very similar to Kube-controller-manager, but targeted subjects of it's operations are controllers with cloud provider dependencies like Service controller [40].

2.3.6 Kubelet

Kubelet is, unlike previous components, not a Control Plane component. It is a Node component, and it runs on every Node in the Cluster. It supervises and ensures container state and health inside Pods according to containers specification in PodSpec. A PodSpec is a YAML or JSON object that describes a Pod [40].

2.3.7 Kube-proxy

Kube-proxy can be found on every Node in Cluster therefore, it is a Node component. It is job to maintain the described network rules, which facilitates the communication between Pods inside Cluster or to Pods from outside the Cluster [40].

2.3.8 Container runtime

The third Node component is container runtime."The container runtime is the software that is responsible for running containers [40]."

2.4 Kubernetes objects

Kubernetes objects are used to describe behaviors, resources, rules, in a word, the state, what has to be maintained inside Cluster. "A Kubernetes object is a "record of intent"–once you create the object, the Kubernetes system will constantly work to ensure that object exists. [45]" Whether implementing a new Kubernetes object or deleting or modifying one, the process takes place through communication with Kubernetes API. Describing Kubernetes objects is possible in JSON or in YAML format. These Kubernetes objects are persistent until they are modified or deleted. Whereas a great amount of optional fields can be added to a YAML file, apiVersion, kind, metadata, and spec are mandatory fields according to the documentation [45].

The following part will introduce different Kubernetes objects written in YAML format. Introducing the Kubernetes ecosystem in detail is way beyond the boundaries of this thesis because of its complexity, but the next part intends to provide an insight, which probably comes across while making a simple cloud-native application.

2.4.1 Deployment

In Deployment, all of those attributes are declared as what concerns a Pod. Before presenting those attributes, it is essential to discuss what a Pod is. Pod is the smallest unit of Kubernetes [39]. Outside of Kubernetes context, a Pod used to mean a group of whales. Building upon that analogy like a group of whales living together and sharing every aspect of their life with each other, Kubernetes calls its smallest deployable unit Pod. This appellation is very apt because containers inside the same Pod are tightly coupled in lots of ways. Container or containers that belong to one Pod share the same specification concerning, for example, network resources or storage [6].

Though it is possible to create a single individual Pod directly, it is not usual in practice because Pod's nature by design is a relatively ephemeral entity. When a Node fails where a Pod is scheduled on for various reasons, for example, lack of resource or maintenance, a new one is created. This new Pod will be almost identical, but it won't be the same Pod. There are multiple options to create and manage Pod or Pods without manually supervising them, like Jobs or Deployment. These workloads come with a Pod template on the basis of which the desired state regarding Pods can be maintained with the help of Controllers [6].

In regard to the Kubernetes object, the words desired state are always mentioned. Deployment is not an exception to that. When a Deployment is applied, all the parameters or states written in it will be maintained by Deployment Controller. A loop is constantly monitoring if there is any difference between a state in the Cluster and the state defined in Deployment. If there is any, the Controller will initiate action to adjust the state of the Cluster to expected state, referring to Pods [6].

Usage of a Deployment also provides the ability to set up rollover and rollback. As the Controller notices that changes happened in Deployment, the old ReplicaSet which is no longer matches Deployment's spec.selector field scaled-down and new ReplicaSet is brought up in a controlled way. Sometimes a rolling back is needed, for example, new Deployment is crashing constantly owing to a typo that has been made in Deployment. A rollback action can be simply carried out depending on the Deployment's rollout history and by returning to a previous version of Deployment [6].

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Listing 1: Example Deployment of Nginx containers.

The number of running Pods in the Cluster can be scaled up and down. Scaling can be carried out manually, for example, scale command applied in kubectl, or with the usage of HPA (Horizontal Pod Autoscaling). The lower and upper limits of running Pods must be compatible with the resource limitation belonging to Node or Nodes in the Cluster. With the help of HPA, Kubernetes deploys more Pod to defined the upper limit when a service or application can not serve the incoming request without users experiencing significantly increased response time. When it happens another way, so the current load does not justify the number of running Pods, their number will be decreased to the given minimum [6].

In Listing 1, a Deployment file has been declared. This example can be found in Kubernetes documentation [6]. Assuming nginx-deployment(metadata.name=nginxdeployment) has no history to the Cluster; after its specifications are applied, the following changes will be carried out. Based on the container image nginx:1.14.2(spec.template.spec.containers.image=nginx:1.14.2), three Pods under name of nginx(spec.template.spec.containers.name) are created. Pod's port 80 (spec.template.spec.containers.ports.containerPort) will be open for network traffic.

2.4.2 Service

Unlike Deployment, which is used to define every aspect of a set of Pods, Service is to define itself, so based on its attributes, a Service object will be made in the Cluster. Despite of that, a Service object will be made; the purpose of Service is the same as Deployment's, to define settings belonging to a set of Pods [46].

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 80
```

Listing 2: Example Service for Nginx Deployment.

Because Pod's nature is ephemeral, they are created and destroyed according to the momentary state of the Cluster, exposing an application running a set of Pods to network traffic carried out by a Service object, which is designed to be more permanent. Each Pod has its own IP address and a single DNS name assigned to a set of Pods [46]. A Cluster-aware DNS server observes the Kubernetes API for new Services and creates a set of DNS records for each one of them. After DNS has been enabled across all the Cluster then all Pods should automatically be able to resolve Services by their DNS name [46].

By staying at the example of nginx-service, the DNS name is, by default would be in the form of nginx-service. By sending a request on the address nginx-service, a Pod located within the same Cluster and within the same namespace can communicate with the application nginx. What would characterize the situation if nginx application was a microservice component can be called "backend" and if the other application would be another microservice component called "frontend". If we assume nginx application is located in the namespace called nginx-namespace, and the other application located in a different namespace, the DNS name that the other application has to send a request to would be nginx-service.nginx-namespace [46].

If Kubernetes Service can be summarized in one sentence, Service is responsible for routing inside of a Kubernetes Cluster, whether it is traffic directed from the Internet to a set of Pods or it is traffic coming from within the Cluster from a set of Pods to a set of Pods [46].

In Listing 2, Yaml file content can be seen from Kubernetes documentation [46], based on a Service object will be made that targets nginx application(spec.selector.app=nginx). The traffic is directed on its port 8080 (spec.ports.port=8080) and will be forwarded to nginx Pod's port 80 (spec.ports.TargetPort=8080) over TCP (spec.ports.protocol) protocol. Figure 2.4 introduces Cluster after ngnix-service and nginx-deployment applied.

2.4.3 Ingress

While Service is responsible for routing inside the Cluster, based on the routing rules defined in Ingress resources, network traffic from outside of the Cluster directed to Services and through Services to Pods, introduced on Figure 2.3. In order for rules defined in Ingress resource work, an ingress controller must be deployed due to ingress controllers

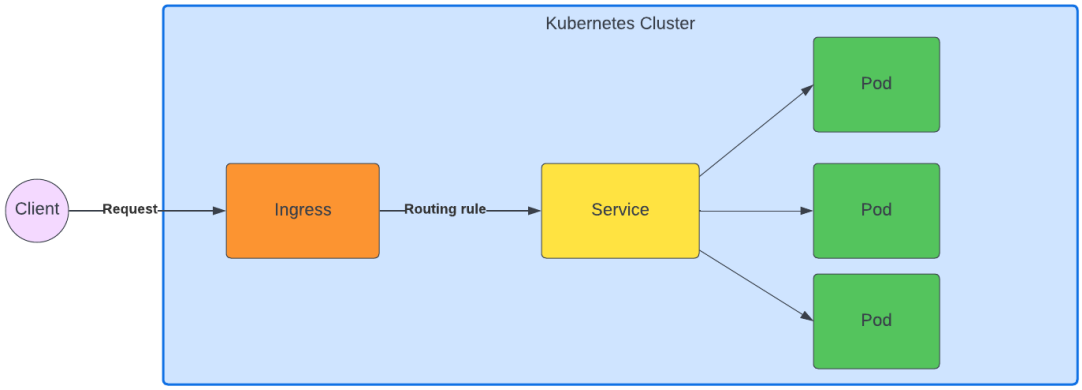

Figure 2.3: Routing by Kubernetes Ingress and Service [38]

different from other controllers,in a that manner they are not started automatically with the Cluster [38].

2.4.4 Configmap and Secret

ConfigMap and Secret existing share the same motivation, which is to store a small amount of data decoupled from the code. While ConfigMaps are used as storage for non-confidential data like environmental variables, Secrets usage aims to store data that nature is sensitive such as API tokens or passwords. Data in both cases is stored in key-value pairs, but in the case of Secrets, the value must be defined as base64-encoded strings. Content can be added to both of them explicitly, for example, URL: "www.example.com" or in case of a secret URL: "d3d3LmV4YW1wbGUuY29t", or it can be read from the file where key-value pairs are stored [41], [42].

2.5 Docker

2.5.1 Introduction

Docker is introduced in this chapter because containerizing applications is not limited to a cloud environment, but its usage counts as an industry standard for cloud-native applications. The technology of containerization made it possible to run applications regardless of the environment the application runs on. Previous to this, a great amount of work was put into reconciling dependencies and requirements between the application and the host's operating system. A missing update on one instance of hosts could cause errors, hard to find, and detection of them took time from productive work. Although concepts and ideas like chroot, kernel namespaces, or cgroups on which containerization is based on are much older than Docker, but the popularity of containerization skyrocketed since Docker presented its solution in 2013 [25].

2.5.2 What is a container?

Explaining the container in the most simple manner results in the following. A container contains all essential components and dependencies like runtime, system tools, system

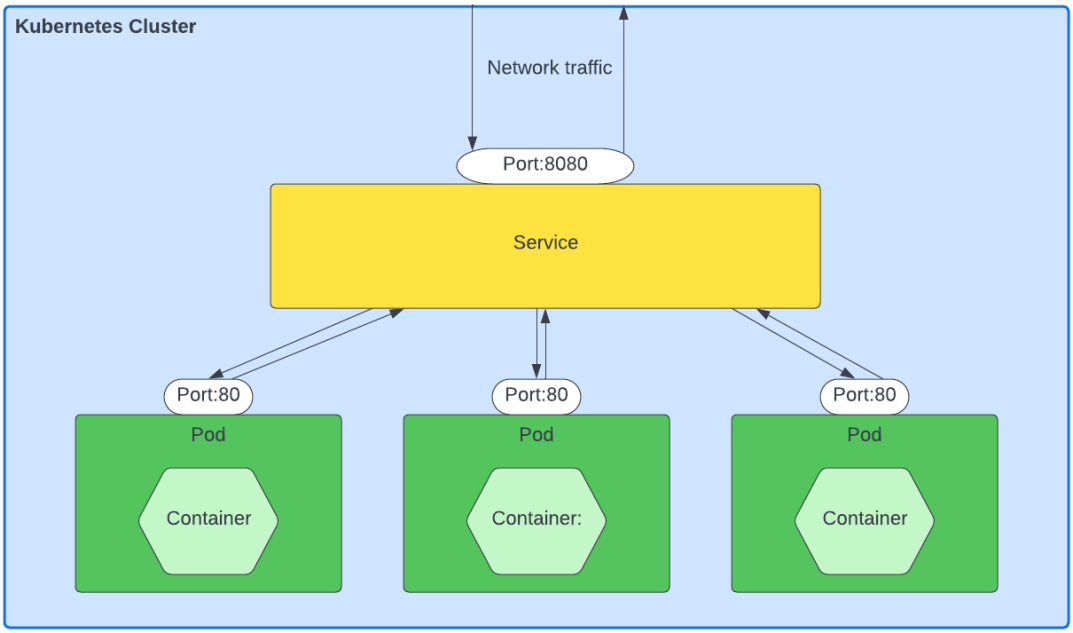

Figure 2.4: State of Kubernetes Cluster after applying nginx-deployment and nginx-service

libraries, etc., and the application's code itself, which allows an application to run in any environment. A container and container image are distinguished. A container image or Docker Image is called container at runtime, so the difference is made by interaction with the Docker engine. "Docker Image or container image is a read-only template with instructions for creating a Docker container. [26]" A Docker Image can be built manually or through automation based on a Dockerfile's content. It is also not rare that a Docker image is made based on another Docker image [26].

2.5.3 Architecture

A Docker container engine on Figure2.6 works as a client-server architecture application. Docker engines consist of three main components and provide many options in the way of usage. The core component Docker Deamon is in charge of building, running, and distributing Docker containers. By using Docker Client, which is a CLI tool, commands can be sent to Docker Deamon. The link between Docker Deamon and Docker Client is a REST API which they can communicate to each other. Besides Docker engine, there are many container engines like LDX, CRI-O, or PODMAN, however, the implementation of might be slightly different from Docker [27].

2.5.4 Virtual machine vs container

Both containers and virtual machines existence have a similar purpose, to divide a greater unit of resource into smaller parts with the help of virtualization in favor of effectiveness and speed. In order to achieve its purpose, container virtualizes the host operating system while virtual machines virtualize hardware resources. In the case of virtual machines, a fully-functioning operating system must be added to every instance; in contrast, a con-

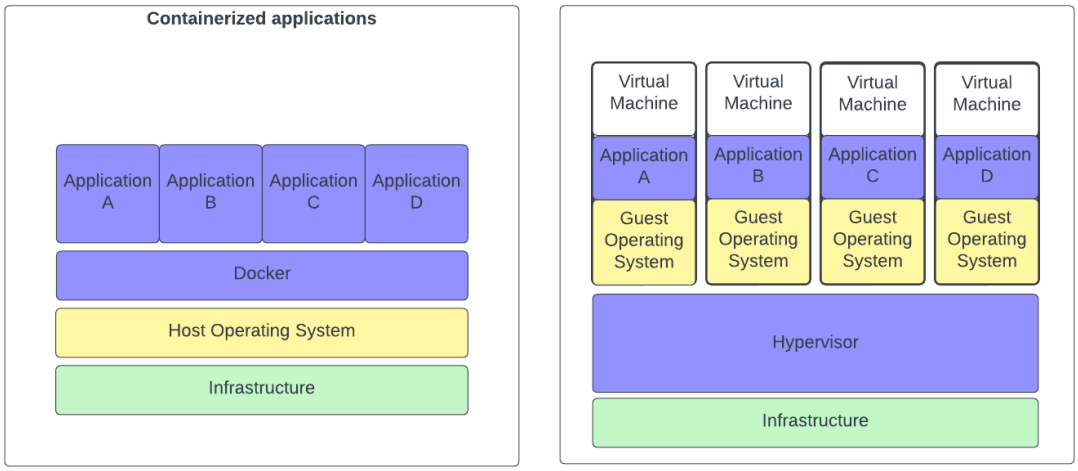

Figure 2.5: Comparison of Virtual machines and containerized applications [27]

tainer only packages dependencies what an application requires to run. Due to the different needs for resources, virtual machines can not be as efficient and portable as containers. It is important to note that virtual machines and containers perform different tasks; they are used together while deploying and managing applications, especially in cloud environments. Their comparison is justified by the avoidance of confusion that comes from their similarities [24]. Figure 2.5 structural differences.

2.6 EKS/GKE

Kubernetes infrastructure might be useful and usable apart from the cloud environment, but it works at its best in the cloud environment. Due to the popularity of the Kubernetes nurse, more and more cloud provider has a service specialized to Kubernetes, wherewith operations like maintaining Control Plane or nodes are simplified or even can be omitted. Without the using managed services, maintaining Kubernetes Cluster involves many more tasks [73]. In this chapter, two very popular Managed Kubernetes Services will be introduced briefly, along with their most useful features, usage, and reliability. The aforementioned managed services are Amazon Elastic Container Service for Kubernetes(EKS) and Google Kubernetes Engine(GKE).

2.6.1 EKS

In general cloud environment has the advantage of high availability. High availability is also important in relation to Kubernetes Cluster. EKS provides this through running and scaling Kubernetes Control Plane across multiple Availability Zone. Amazon Web Services(AWS) has three terms to define data centers around the world [11]. The Region is the greatest unit, and it indicates a physical location like North America or Asia, where data centers are clustered by AWS. A Region consists of multiple Availability Zones and one or more discrete data centers with redundant power, networking, and connectivity. Local Zones are an extension of an AWS Region placed closer to end-users in favor of latency-sensitive applications that require small latency for their proper working [12].

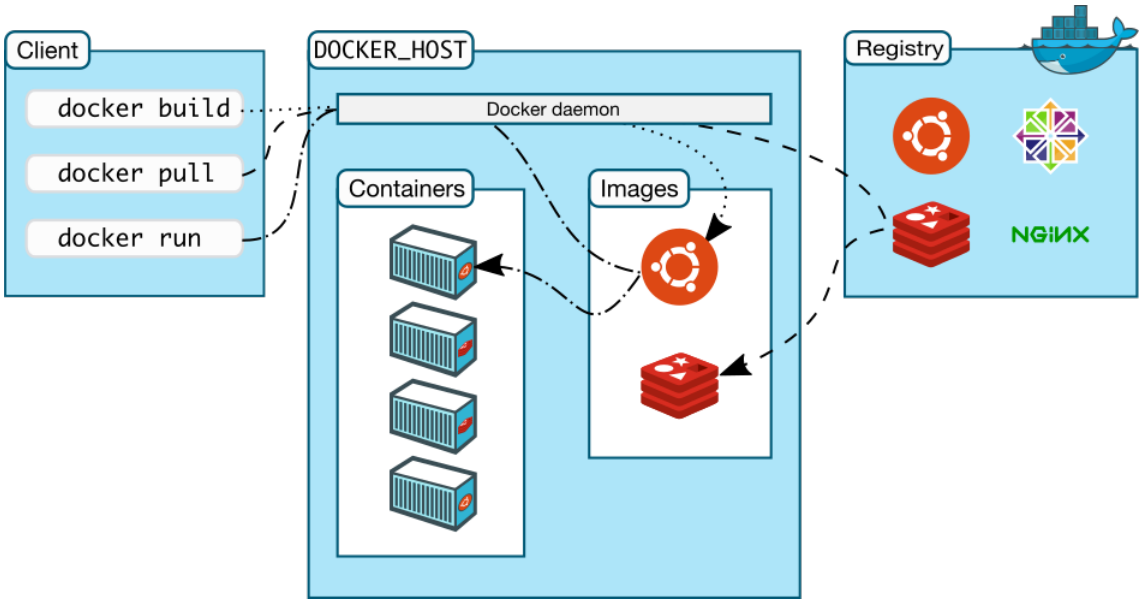

Figure 2.6: Illustration of Docker architecture [27]

EKS takes care of load balancing; it scales Master Node or Control Plane instances as the load increases or decreases. It also, has automation for detecting faulty Control Planes and replacing them with a new one. Required updates and patching are also provided. Many of AWS services are integrated into EKS to complete these tasks, for example, Amazon ECR for container images, Elastic Load Balancing for load distribution [11].

EKS takes care of running an up-to-date version of Kubernetes software; therefore, all of the latest tools and plugins are available for use. Migrating a Kubernetes application to Amazon EKS from any standard Kubernetes environment, regardless of the application was running in on-premises data centers or public clouds, is possible [11].

2.6.2 GKE

However, Amazon EKS offers great options to its managed Kubernetes service; GKE's solution might offer the same effectiveness with a bit more advanced scalability, which does not come as a surprise regarding that Kubernetes has evolved from a Google project [60]. Google launched its managed Kubernetes Service in 2015, three years earlier than Amazon launched EKS. Google Cloud's compute engine resources can be found in multiple locations around the world. Similar to AWS, Google calls its greater size units Regions, which names also indicates a geographical location. A Region consists of Zones, which cover the smaller area with services. Resources can be Regional, Or Zonal. Putting resources in different zones protects against problems that come from hardware and software failures. Putting resources in different regions helps to maintain a high availability and raises isolation from effects caused by any kind of failure on even a higher level. "All Compute Engine resources are either global, regional, or zonal" [31]. Based on the category the resource belongs to, it can be accessed by other resources from other Zones within the same Region or from different Zones or Regions. For example, Images are global resources that can be accessed by any resource in any Zone or Region within the same project." Regional resources are accessible only to resources within the same Region, regional static external IP exemplifies well this category". Zonal resources like VM instances can be accessed by resources located in the same Zone [32].

GKE also provides server-side load balancing in order to ensure Cluster and the applications running in it operate properly despite varying amounts of inbound traffic by distributing incoming traffic across multiple virtual machine instances. GKE's autoscaling mechanism can be driven by load balancing serving capacity, CPU utilization, Cloud Monitoring metrics, or schedules [30]. Based on the set method, VM instances are added or removed from the managed instance group (MIG). Managed instance group is a concept that is used to manage multiple virtual machines as one and offers further benefits in maintenance like auto-healing or automatic updating [29].

2.7 Prometheus and Grafana

Prometheus is an open-source toolkit made for services monitoring and alerting container-based environments. Prometheus server has three main components. First component collects data from monitored systems, forwards collected data to the second component, which stores the metrics as time-series data, and through the third component, what is an HTTP server provides access to the stored metrics for visualization or alerting purposes. Many Prometheus components are written in Go for a reason, it is easy to build and deploy as static binaries. For monitoring applications, Prometheus provides client libraries for different programming languages. In case of using one of these client libraries /metrics endpoint can be exposed inside the application; after that, some of the metrics are shown on that endpoint for convenience [19], [75].

Various kinds of entities can be targeted for monitoring by Prometheus, for example, a server or an application. In order to establish customized measurements on these endpoints Prometheus offers three metric types. The Counter type metrics are used to represent a value that can only increase or be reset to zero on a restart, for example, the number of errors. The using type Gauge suitable for monitoring values can increase and also decrease, for example, current CPU usage or the number of concurrent requests. The third one is a Histogram, which is used in that case when the size or duration of a value needs to be measured. Measured values are added to Configurable buckets, and it also provides an option where all observed values can be summed [75], [18].

Prometheus has a functional query language called PromQL, with the help of which more detailed queries can be carried out in order to increase accuracy or in favor of visualization or analysis. For example, when a specific time interval is applied in a PromQl expression, only data related to that time interval will be shown as a result. "The result of a PromQl expression can be viewed as tabular data in Prometheus's expression browser, or consumed by external systems like Grafana via the HTTP API [17].".

Visualizing of metrics can support better understanding of processes, and effects of events connected to the application. Using Grafana and Prometheus together is a powerful combination for monitoring purposes, Figure2.7 illustrates the communication of Prometheus and Grafana. According to this [53] website Grafana described as following "Grafana is a multi-platform open source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources." Two of the Grafana's platform is relevant to this thesis. First is called Loki where log stream of an application are monitored [5], the second is Grafana Prometheus where Prometheus metrics are visualized [16].

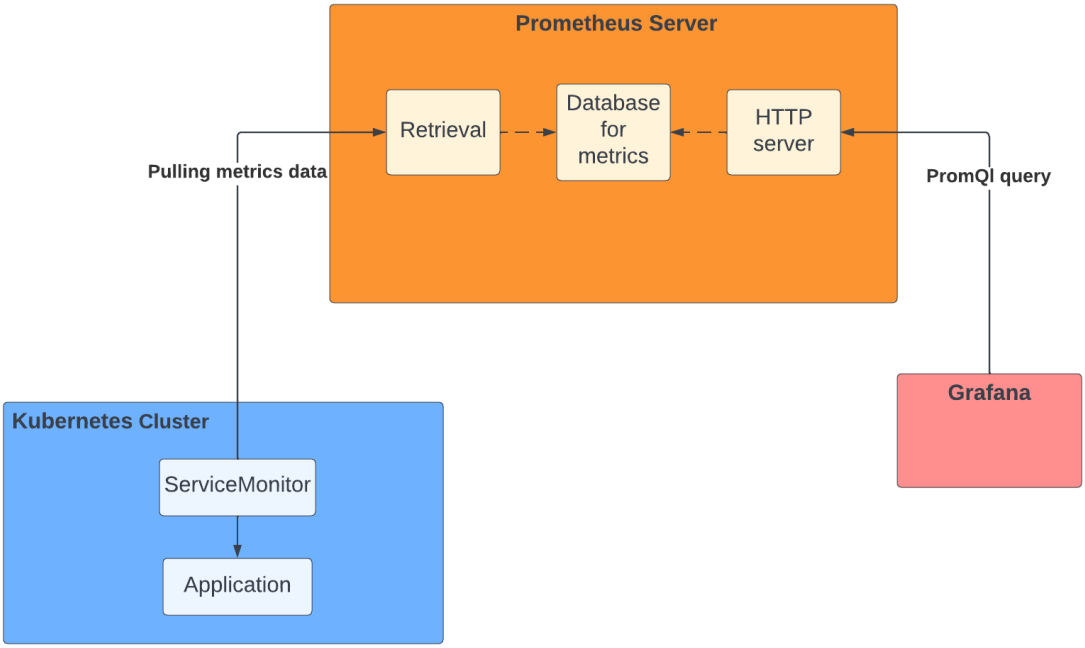

Figure 2.7: Illustration of Prometheus architecture and Grafana [75], [19]

Chapter 3

Automated processes and techniques in code integration and deployment

In this Chapter, concepts, methods, and tools will be introduced that are used for automating processes from making a code to deploying a product in a production environment. As a result of the usage of these tools the higher level of efficiency can be achieved, communication between the development and operation team, and understanding of tasks from both sides can improve.

3.1 CI/CD

As this [49] website summarizes it,"CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of application development [49]." Or put it in other words, CI/CD is a software development practice which provides a secure, reliable, and fast way to release a new feature or bugfixed version of the application by applying automated build tests and deployment. CI in CI/CD stands for continuous integration; CD either means continuous delivery or continuous deployment [49].

According to this website [1], the usage of CD reduces risk during new releases by applying patterns such as Blue-green deployment to decrease downtime while carrying out new deployments. When development team members can focus their time and effort on, for example, usability-testing or security-testing instead of lower-level fault detection like a typo in code, which is indicated by continuous delivery processes, it results in higher quality code in a cost-effective manner. Using automated processes for testing code is also advantageous in aspect of the well-being of software developers due to it frees time from monotonous mechanisms(which often leads to burnout) and supports creativity [1].

Another interesting aspect of CI/CD synthesized by Continuous delivery foundation in the state of the continuous delivery report [8], is how to measure the effectiveness of it. There are four different factors through which the effectiveness of a CI/CD pipeline can be defined. Firstly Deployment Frequency is the frequency of successful releases to the production environment, for example, whether this happens on a daily or weekly basis or if this happens less frequently. The second factor is Lead Time for Changes, which is defined by the median amount of time for a commit to be deployed into production. These factors are indicators for speed and the following two factors show reliability by numbers.

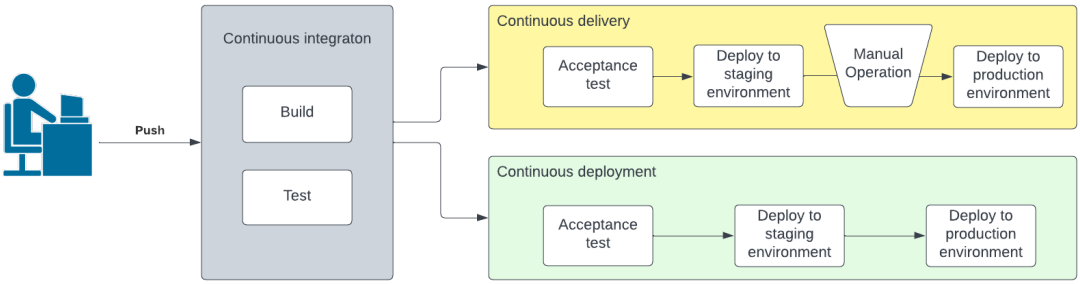

Figure 3.1: CI/CD processes [63]

Change Fail Rate is calculated by the number of failures per the number of deployments. Time to Restore Services is the median amount of time between the deployment which caused the failure and restoration [8].

State of Continuous Delivery Report [8] presented data about software delivery performance by industries and by programming languages. By plotting metrics Lead Time for Changes and Deployment Frequency(speed indicator) on the x-axis and metrics Time to Restore Services(reliability indicator) on the y-axis of the chart, we can see which industry emphasizes reliability over speed and which performs greater in speed over reliability. Data shows us that the greatest performance in reliability is in Financial Services, Government and Energy industries, and Retail extremely well at both. In the case of software languages, the first is Shell Scripting Languages(Bash, PowerShell), just like Retail among industries, perform well on both sides. Scripting languages are followed not far by Go, and finally, JavaScript [8]. Owing to the benefits that using CI/CD can provide, it is now a widely used technique in software development. Regardless of company size or the nature of task, CI/CD can be adjusted because of the extended range of tools which are available today [8].

3.1.1 Continuous integration

Continuous integration is more like an extension development side's actions due to it carries out actions that reveal problems, for example, typos connected to building a code. As part of a CI/CD pipeline, continuous integration covers the following tasks. Implementing a new feature, or fixing a problem in the code. Pushing code into a remote repository. Running integration tests. Building the code. Merging changes into the main branch [63].

3.1.2 Continuous delivery/Continuous deployment

Continuous delivery and continuous deployment are both covers the second part of CI/CD pipeline. Set of actions and steps, main purpose is to validate code and keep it in a state where deployment can be initiated at any time. A CD pipeline often contains the following steps: running acceptance tests, deploy code to different environment, for example staging environment where code is tested last time before it is deployed to production environment, and smoke tests which are according to this [10] website "Smoke tests are a subset of test cases that cover the most important functionality of a component or system, used to aid assessment of whether main functions of the software appear to work correctly" [10]. As Figure 3.1 shows continuous delivery and continuous deployment differ at one

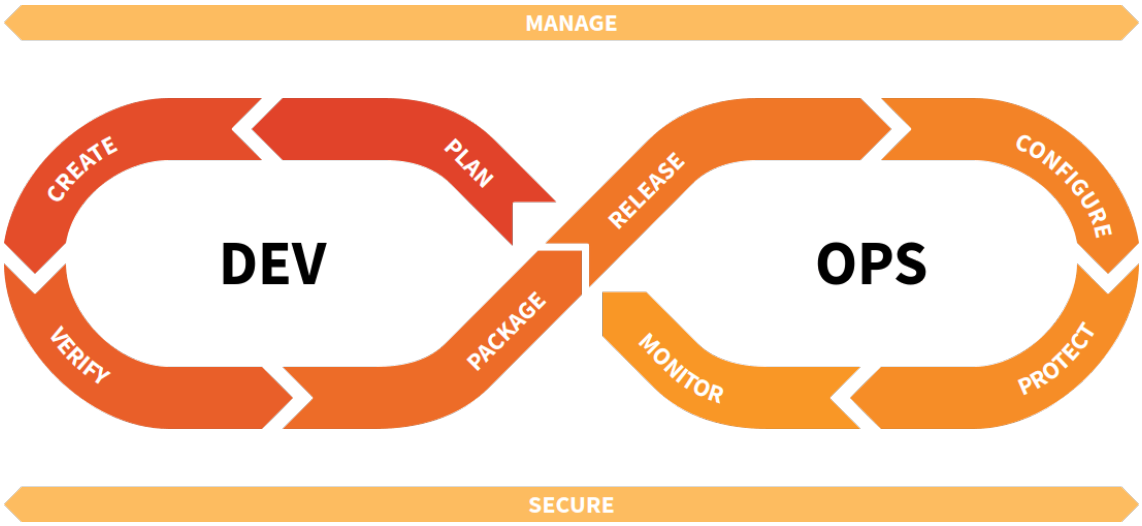

Figure 3.2: DevOps stages [14]

point. In case of continuous delivery, deployment to production environment carried out manually,while this step is completely automated in continuous deployment pipeline [49].

3.2 DevOps

"DevOps is a set of practices that combines software development (Dev) and IT operations (Ops)" [52]. DevOps regarding it's purpose similar to CI/CD practices, in a sense that by help of various tools, product or new feature of a product frequently delivered to users, but DevOps focuses more on roles and cultural philosphy side of these processes [68]. Applying DevOps practices leads to better communication and collaboration between software developers and experts responsible for deploying and operating the application in production environment. From the code quality point of view, applying DevOps practices will result less bugs and a well tested code so overall higher code quality [76].

Intends and philosophy belong to DevOps developed an actual position, called DevOps engineer. This role's main goal can be described as building a bridge between software development and operations part. Development is interested in releasing a new feature as soon as possible, while Operation wants to make sure that the application with the new feature works properly regarding speed and errors. In addition, neither of them understands completely what really are the requirements that belong to the other field have to be met, and this often leads to slower delivery. Since the DevOps engineer has good insight into the preferences also development and operation, he is capable of synchronizing actions from both side to increase efficiency in daily work [76].

As it is listed on [14] GitLab's website and introduced in Figure 3.2 in ten different stages of DevOps through an application lifecycle.

1. Manage

   At this stage feedback about how the whole infrastructure including the application functioning incorporated into the operation in order to improve the product at every iteration of delivering new feature or fixing a problem [14].

2. Plan DevOps establishes different tasks what has to be carried out, an tracks its completion [14].

3. Create

   This phase where code is written, suggestions for development discussed [14].

4. Verify

   At Verify stage automated tests run to make sure everything works error free [14].

5. Package

   In this phase software is packaged with it's dependencies [14].

6. Secure

   At this stage, DevOps searching for vulnerabilities connected to every aspect of the software [14].

7. Release

   This phase the code is released to production environment [14].

8. Configure

   At this point configuration setups to application environment, and taking care of passwords and other sensitive data are done [15].

9. Monitor

   Devops is monitoring application performance, and react in every case an error occurs [14].

10. Protect

    DevOps secures the infrastructure the software is running including container inspection [14].

3.3 ChatOps

ChatOps, just like DevOps, has come alive as a combination of two words, what is Chat and Operations(Ops).The ChatOps concept is built around improving communication by bringing operations and conversations under the same platform in order to improve effectiveness and speed. Using ChatOps in a way can be considered similar to GitOps practice' essential part, where GitHub repository considered as a single source of truth, where every change can be seen by anyone who needs to be aware [56].

ChatOps almost has infinite potential by combining different tasks in one command, Figure 3.3. For example, managing infrastructure configurations, scaling up or down running containers, starting tests, deploying an application to different environments, etc. There is always at least one chatbot integrated into a ChatOps platform. Chatbots are responsible for executing the operations triggered by the chatbot command in a chat application [56].

Besides operation started from here, using a chat application during daily work offers various advantages. When a team member initiates an operation, it is transparent for other team members, so they know in advance what changes will be carried out, and if it is connected to their work directly, they can adjust to the change in advance. If the chat application is chosen carefully, it provides many other functions independently from operations, which makes it useful for everyday work-related communication. If a development team has already been using a chat application for work-related communication, in case of integration of a chatbot to complete ChatOps tasks does not require adaptation

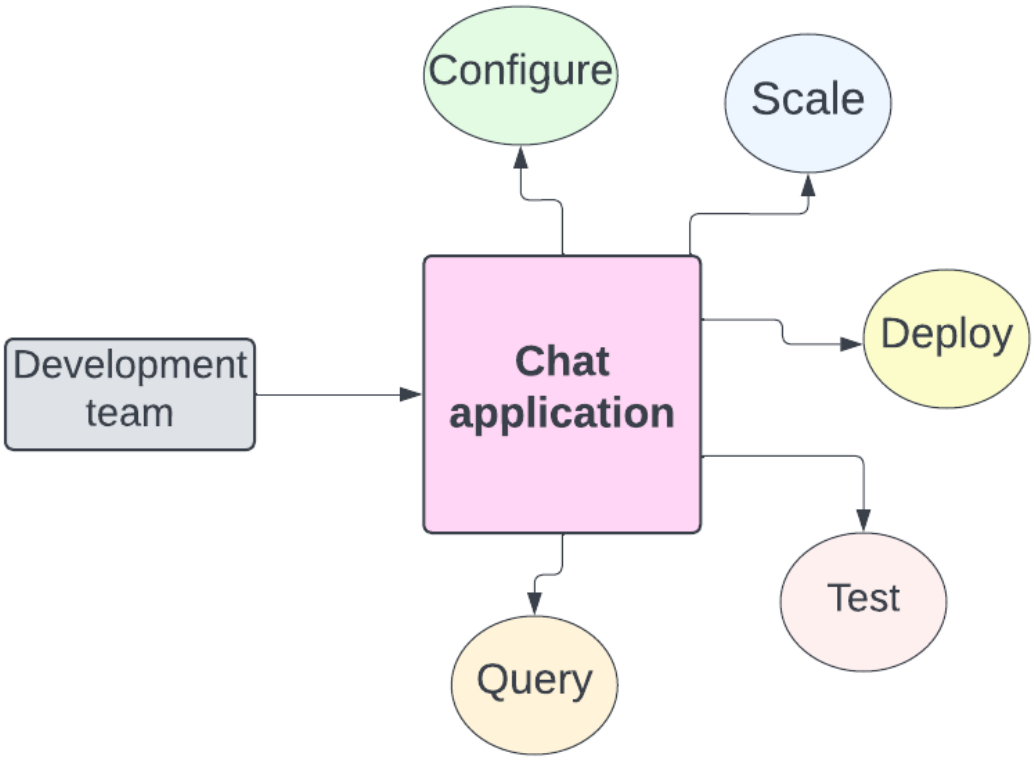

Figure 3.3: Example chat operations

to a new tool. Implementing ChatOps makes applying operations for team members out of the scope of their specific field of profession safe because an operation's effect has been limited and approved by a team member who is an expert in that field. For example, software developer member of the team needs to apply a complicated operation on the database connected to the new feature he is currently doing; he does not need to ask the Database administrator for help because a command has already been made for that action. Issuing a command in a chat also produce logs because it can be found in the chat history therefore it may could help tracing back a source of failure. Team members do not need to switch to another platform log in, navigate to a specific point of a UI, etc., for example, to query CPU utilization of a container [56].

This website [61] listed how ChatOps is not just a new tool that is integrated without any significant change happening in the attitude of the team member but also a cultural shift in the company's life. The team member who has never used ChatOps during work needs to trust a chat application as an entry to operational features that were previously only open to individual users in specific and isolated environments. Unlike before, when communication meant formal emails and meetings, there is more space in a chat for colleagues to express more from their personality or perspective on a subject with a Meme or an emoticon, so beyond the aforementioned advantages, integrating ChatOps practices on work efficiency, it also may affect beneficial on the mood in the office. As everything that happens is well documented in the history of the chatroom, it is easily traceable; it prevents members from blaming each other for missing notifications related to actions that have been made [61] .

However, ChatOps may have a good impact on multiple aspects of software development and operation; adopting it will not be straight away. Sean Regan, Head of Product Marketing, Jira and Bitbucket, identifies five stages of ChatOps adoption [64].

1. Fist stage

   At this point, the ChatOps concept is new to every team member, and sometimes they just use group chat only for messaging and sharing files, but for the most of the formal conversations, they still use email [64].

2. Second stage

   This is where everyone is getting comfortable with the concept; persistent chat rooms are formed by certain workloads and regular conversations. Chat rooms are replacing emails in formal communication. Teams begin to experience benefits from firsthand [64].

3. Third stage

   Where the whole ecosystem and legacy workloads gradually migrated to chat. Highly technical teams start to implement operations carried out by chat commands. Employees get used to searching back in chat history if they need information about a specific project [64].

4. Fourth stage

   At this stage there is no question if an email should be sent to discuss internal work related topic or a chat message. Everyone is convinced about the advantages of ChatOps technology. More and more common tasks are done by chatbots. Other tools and most of the workload are managed within chat. More unplanned work starts at chat room by instinct [64].

5. Fifth stage

   This is the last stage of adopting ChatOps, where work unlimitedly relies on a chat-based environment, and highly advanced bots complete the complex job with possible AI implementation. Teams have automated most of the work critical work [64].

These stages are not especially IT sector related; in the IT field, adoption would take place probably faster, but this only point out that on hand ChatOps practices are not only beneficial for IT companies, but for many other fields of profession and on the other hand independently from the company's profile, moving on to use a new tool or concept requires time [64].

3.4 Kustomize

Kustomize is a command-line tool with the help of which Kubernetes file's configuration management can be carried out in a template-free manner. From version 1.14, multiple Kustomize features available built-in Kubernetes command-line tool kubectl, by applying -k flag in kubectl command. Kustomize files are written in YAML language [47].

During an application development life cycle, at some point, a need may arise for a version of the same application but with a slightly different configuration to fulfill testing purposes or to run a specific Pod instance with a different resource configuration for performance

measurements. When this occurs, without using Kustomize or another configuration management tool, a file, for example, a Kubernetes Deployment, almost identical to the original, will be made. As the time goes on, another small change is needed to be carried out for the same Deployment, that does not concerns the fundamental elements, and now there is 3 version of this Deployment. As this pattern continues, the number of files will grow, and it reduces transparency and complicates maintenance [47].

Besides Kustomize there are many other great tools for an application configuration that can be managed, like the very popular Helm. Using Helm for a certain complexity of the application is like killing a fly with a sledgehammer. It will solve the same problem but at a much higher price. Kustomize uses an approach that is similar to inheritance to apply different configurations in a different environment illustrated on Figure3.4, and Helm uses templates and placeholders for configurations to achieve the same purpose. Understanding how templating works can result in a steep learning curve. If kubectl is used by a team, they do not have to learn a completely new tool, so using Kustomize besides kubectl does not require much effort to learn [70].

3.5 GitHub actions

GitHub action is a continuous integration continuous delivery platform. A GitHub action workflow has five main components, which are Workflow, Job, Event, Action, and Runner. Implementing a GitHub action workflow makes it possible to automate CI/CD pipelines or basically any other kind of software development workflow. There are some inevitably great advantages of using GitHub actions. Since Git Version control is a de facto standard in the software development industry, it is highly possible that application code will be stored at GitHub and, due to GitHub actions, is also integrated into GitHub; workflows made by using Github Action and code of the application will be stored in the same platform or even in the same repository. Besides the aforementioned advantage, further benefits can be mentioned, including flat learning curves, if someone is familiar with GitHub. Visualization of running Workflows or Jobs and transparent logging are also present features that facilitate work with GitHub actions easier. All of the components are defined in YAML language [28], [74].

3.5.1 Workflows

A Workflow is the greatest unit of GitHub actions. A Workflow or Workflows need to be placed in a certain file path inside the Github repository in order to work, which is the following repository name/.github/workflows/. A workflow contains one or more Jobs and one Job complete one task. A Workflow run is triggered by Events like Push or Pull request, or it can even be a consequence of applying new Realease tag. For example as a result of a Pull request created a Workflow's run builds tests the code what needs to merged into main branch [28].

3.5.2 Jobs

Jobs similar to Workflow consist of smaller fractions, which is a set of steps. The steps in the same Job are executed on the same Runner. A step is either a shell script or an Action. Steps always run in order. Steps are able to work together in a way that a step can use another step's output to apply it in its own execution. For example, a Job pushes

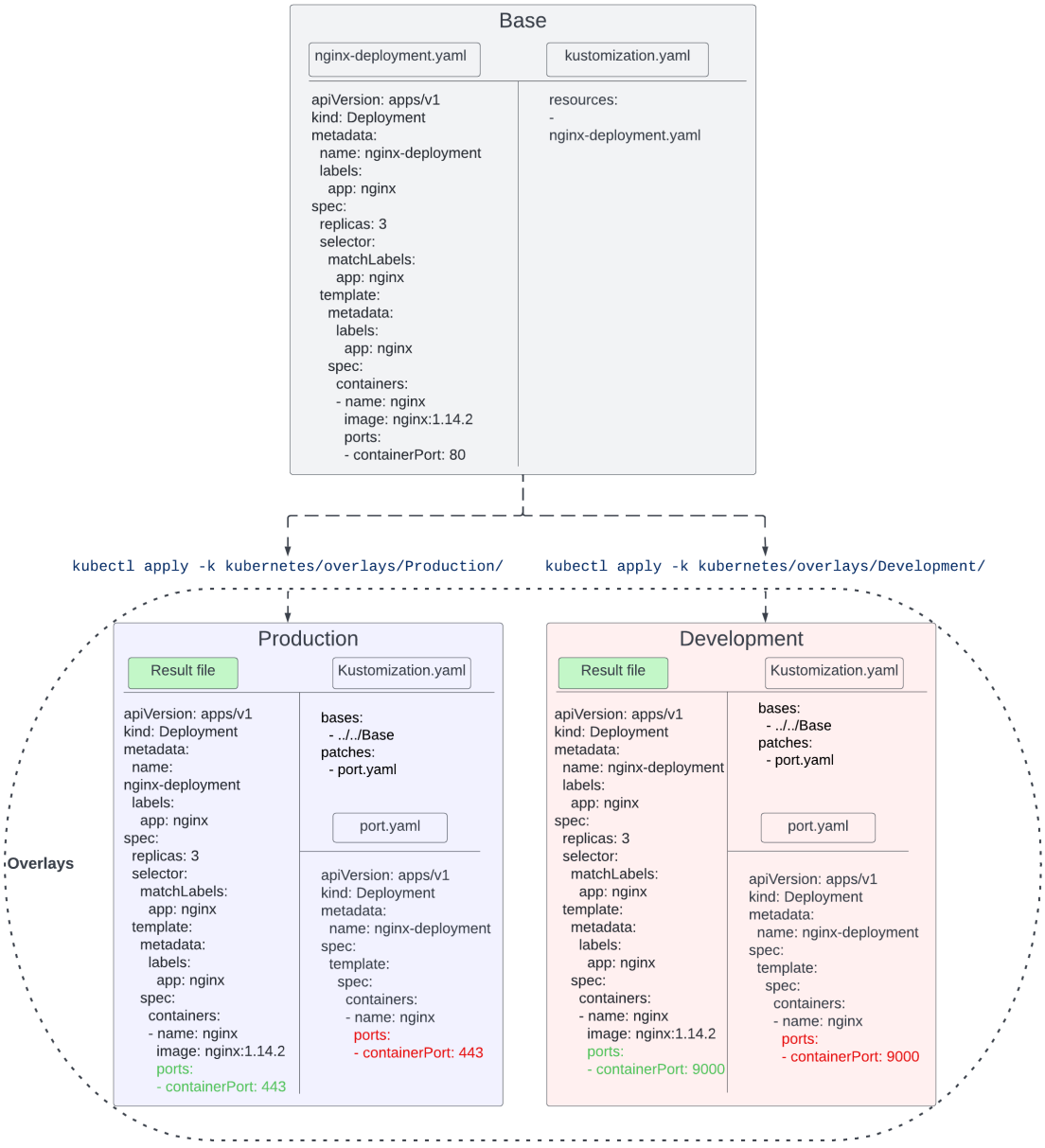

Figure 3.4: Applying different configuration in different environments using Kustomize

a container image to a registry, but before an image can be pushed, a log-in must take place, so therefore a step previous to this step has to log in to the container registry. By default, Jobs in a Workflow run parallel; however, custom execution order can be set between Jobs by developing dependencies. When for example Job B depends on Job A; Job B has to wait for its execution until Job A finishes its execution successfully. It is important to note here when a Job's execution fails, dependent Job execution won't start [28].

3.5.3 Actions

An Action is an application made for the GitHub actions platform. An Action performs tasks that are often necessary, for example, building and pushing a Docker image to a registry. Applying an Action instead of writing code to accomplish a task helps to avoid code repetition and saves time. Using an Action can be proven useful in terms of maintenance; for example, an Action, which installs Go on the Runner, might be configured to install the latest version of Go, which ensures that this part of a Workflow is always up to date [28].

3.5.4 Events

According to GitHub action's documentation, an "Event is a specific activity that triggers a Workflow to run". A wide range of options are available to choose the Event which occurs; a Workflow begins its run. From Push or Pull requests to Events like Delete which triggers a Workflow to run when a Git Reference is deleted in the Git repository where Workflow placed. In some cases, multiple Events are assigned to a Workflow; in this case, whichever Event happens Workflow runs once to each. Workflow can be run by posting HTTP requests to GitHub API, or by defining schedule, or even manually. No matter that the assignment is building CI/CD pipelines or implementing other types of software development workflow, GitHub actions platform almost covers every case to control its run by an Event [28].

3.5.5 Runners

The last component out of the five is the Runner. Runner is a server that runs a Workflow when it is triggered by an Event. For each Job's run, a new virtual machine is created and started; consequently, the results of a Job's run in a Workflow are available at the time the Workflow runs or, in other words, until every Job finishes its execution in the Workflow. By staying at the example of pushing a container image to a registry. A container image's name, what is a result of Job's run, even if it is assigned to a variable, what accessible in the scope of the Workflow, only available to the other Job, while Workflow runs. For each Job's run an operating system must be specified. GitHub actions provides Runners, that comes with Ubuntu Linux, Microsoft Windows, or macOS operating system. On Runners often, frequently used tools like kubectl, are also preinstalled [28].

Chapter 4

Application

Application itself, as it is also a significant part of the whole concept, needs to satisfy standards, patterns, rules, or implement specific architecture to leverage benefits comes from using cloud environment. In Chapter 4 the Twelve-Factor Application's principles, microservices architecture, and the concept of cloud-native will be introduced, which not only makes an application suitable for running in cloud environment, but it facilitates every future change or action like maintenance or cost-effective resource management.

4.1 Cloud-native

The definition of cloud-Native by Cloud Native Computing Foundation is the following "Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil" [13].

As it is described in the definition of a Cloud-Native application it relies on container technology for example one of the most popular is Docker associated with a modern container orchestration tool like Kubernetes. Usage of these to creates possibility to run the application with possibility to scale up and down to avoid a situation where resources paid but not used or the situation other way around when resources are needed in order to provide good service experience to users, but there is nowhere to return to for more. By adding more resource on demand whenever it is needed, and the opportunity for a company to manage application in a cost effective way, employees can focus on their full effort on development instead of configuring for example virtual machines, and since the combination of two is portable if a cloud provider raise it's fees infrastructure can be easily migrated. Robust automation in for example CI/CD pipeline or infrastructure management, reduce the number of errors caused by human factor, also achieving high degree of resilience in case of a Cloud-native application connected to automation, for example a container are automatically restarted if a containerized application runs on failure. Observable by monitoring a visualization tools Prometheus and Grafana where logs produced by an application can be seen right away, or better with better visualization performance can lead to understand deeper understanding or come up with better solution in case of anomalie, or just adjust resources alerting level on resource utilaziton in favor

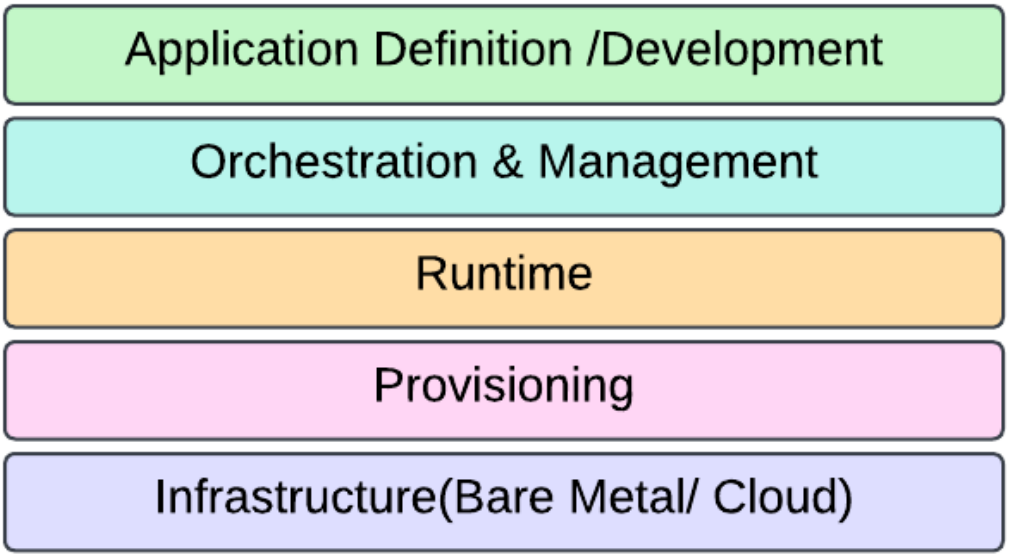

Figure 4.1: Cloud-native architecture [66]

for provide better service to users. If it is possible technology stack based on open source technologies to avoid any kind of lock-in situation [54], [66].

Although cloud-native concept's popularity grows quickly as usage of cloud environment grows, this is still count very new and only few company dived into cloud-native deeply. Cloud-native patterns can is a way of referencing certain situation and solutions connected to cloud-native concept, because adoption of this concept is not always as obvious as it seems to be. Often cloud-native applications Figure 4.1 think of as change in infrastructure and used technologies containerizing applications by someone who is new to the concept, but in reality a successful transition requires more than just a shift from using monolithic architecture to using microservices arhictecture, in order to achieve the desired result a change of point of view or change in mindset must happen. Pattern languages and design might be useful on the way to transition [21]. Pattern languages with Jamie Dobson's words is "Cloud native pattern language is a collection of design decisions about cloud-native practices and technologies and the context in which they work" [57].

On this website [22] of cloud-native patterns are grouped into four different topic Strategy&Risk reduction, Organization&Culture,Development&Design, and Infrastructure&Cloud. Patterns from these different groups can be applied at any point in a transformation design, or patterns within a single group can be used to dive deep into it and create a very detailed design for a particular stage at the beginning of transformation. Through the example provided by the website, Strategy Risk Reduction patterns might be most useful in the early planning stages of a transformation [57].

4.2 12 factor app

12 Factor App is a set of principles defined by Adam Wiggins concerns various aspect of application, and CI/CD in order to satisfy a modern software-as-a-service concept's standards. Saas describes a delivery model where application is hosted by a cloud provider,

and available for end users over the Internet. Based on 12 factor app project contributor's experiences, they gathered via the work on Heroku, they suggest to apply these principles independently from what programming language the application written in, to avoid software erosion and to improve collaboration between developers working on the same project [72].

1. Codebase

   The fist principle suggest to use one repository or "Codebase" with many "Deploys" which refers to a running instance of an application in different environments for example production or staging, with a version control [65], [72].

2. Dependencies

   A twelve-factor application comes with explicitly defined dependency declaration manifest which is isolated from code. This provides transparency, what helps developers, who are new to the project and facilitates building the code [65], [72].

3. Config

   Configuration should not be stored in the application code, in form of variable, or constants because unlike application code, configurations are not the same across different environments. The internal application configurations are out of the scope of this of restriction defined in this principle. The best practice is to store configs in environment variables what can easily replaced in different deployments [65], [72].

4. Backing services

   Backing service can mean any kind of service available via network, the application use during it's run such as databases or other API-accessible services. These services should be loosely coupled to the application, in order to the change them at any time to a substitution, if it is necessary [65], [72].

5. Build, release, run

   The three stages what code goes through before deployment. The first is build where code compiled into an executable, the second is release where configurations settings are added and code become a state, where execution can be started at any given time. Third is run stage, which runs the application in the execution environment. This principle strictly separates, the three stages from each other, as a result no code modification allowed in later stages, because propagate back changes is not possible [65], [72].

6. Processes

   A twelve-factor app consist of stateless processes, it does not share or store data, or keeps track of other processes it uses stateful backing services if those are needed. Therefore during a scaling where new application instances brought up will not produce any unexpected malfunctioning because there is no difference between the old and the new instance [65], [72].

7. Port binding

   According to this principle every twelve-factor application is completely self-contained, this mean instead of relying on runtime injection of a webserver, web app exports it's service on a specific port and listening to requests coming in on that port [65], [72].

8. Concurrency

   An application accomplishes this principle assign one process to one task, so can be scaled by its processes in a accordance with the current need for the task instead of scaling the whole application [65], [72].

9. Disposability

   The twelve-factor app's processes can be started or stopped at very short period of time, this ensures that scaling and deployment of code or apply configuration changes, carried out quickly and contributes to robustness of production deploys [65], [72].

10. Dev/prod parity

    Development and production environment should be as kept as similar as they can be including used backing services. If this goal is achieved it will result shorter period of time in between deployment from development to production environment. Also the team responsible writing code a team deploy and operate will more closely collaborate [65], [72].

11. Logs

    Instead of manage log files or store them this principle suggest,the logs produced by running processes as event stream should be displayed on standard output [65], [72].

12. Admin processes

    Run admin processes in the same environment as other processes of the application including config, codebase, and with same realase, and should be shipped with application code for synchronization [65], [72].

4.3 Microservices

Kertész Dávid very thoroughly summarized every aspect of microservices architecture in his thesis. He characterized microservices as the following "microservices are standalone services running their own processes aimed at performing a well-rounded functionality. They communicate through lightweight APIs with HTTP and are able to be deployed independently with minimal central supervision" [58].

Unlike in the case of an application with a monolithic architecture, which holds all of their services in one place as a unit, an application with microservices architecture uses a different approach by decomposing its services into smaller fractions [58]. Figure 4.2 shows cloud-native reference architecture.

The aforementioned different approach comes with many advantages and some drawbacks while operating an application that uses microservices architecture. While a monolithic architecture application has to be deployed on the same virtual machine, microservices can be distributed more freely. Taking a single microservice's specific needs for resources into consideration at the time of setting up a virtual machine allows more customization and can result in better resource management. Martin Fowler described microservices components as "independently upgradeable and independently replaceable units". He says if microservice as a component is independently upgradeable, upgrading that tiny fraction of code will not cause any unwanted side effects in the other interconnected components;however, upgrading a process of a monolithic application might cause interference between other processes that are not intended to upgrade simultaneously. The quality of Independently

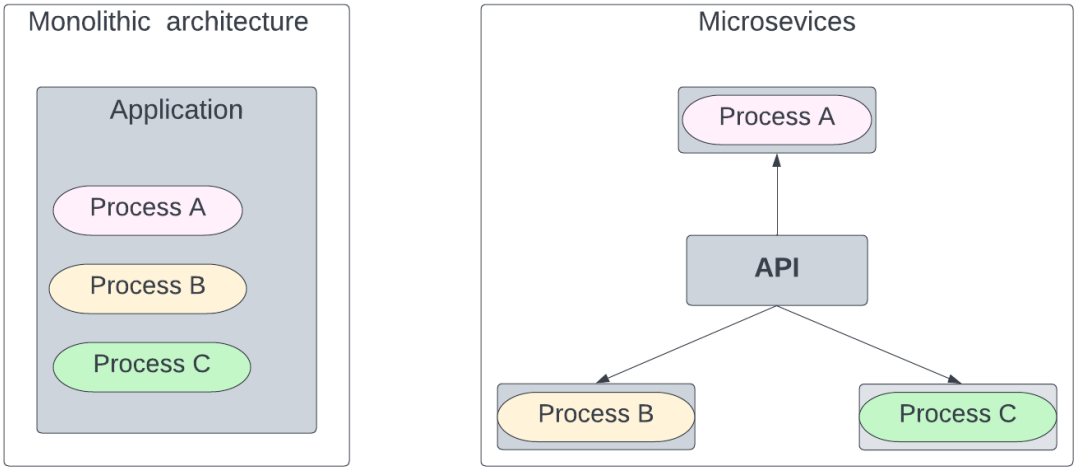

Figure 4.2: Monolithic application and Microservices architecture

replaceable components has similar advantages like carrying out upgrades; the component can be replaced without changes having to be made regarding other components [59].

Martin Fowler summarized, when to use microservices architecture over monolithic. No one should rush switching to using microservices architecture if it is not justified by the current complexity of the application or if the development team is not familiar with every aspect of the architecture requirements like more complex communication between components.As compensation for more complex communication and using distributed computing, they can be deployed partially and, as a consequence of that, more frequently and faster. Also microservices can be scaled horizontally hands down one by one [59].

Chapter 5

Tools and concepts in practice

A chatbot application has been made to demonstrate previously introduced tools, techniques, and designs in practice.

The first section will detail the processes of how a new Telegram bot can be requested and how Telegram messaging service's API works. Every aspect of implemented commands will be introduced, including the operational and non-operational types of commands.

The second section summarizes parts CI/CD pipeline, what has been built to automate tests and deployment.

The third section explains what specific Prometheus metrics have been implemented, for monitoring application's performance in Cloud environments. And introduces a load testing results concerning "PrimeGenerator" component.

5.1 The application

The application is based on Telegram's chatbot, and every component of it is written in the GO programing language. It also performs operational and non-operational types of commands. It has three main components. The first component itself is a chatbot, and the second component is called PrimeGenerator actually is a backend for GenerateBigPrime command, which has been implemented as a microservice. The third component is referred to as the Load testing tool as well as PrimeGenerator component also run as a microservice inside the Kubernetes cluster under a command called Load.

I have used the tgbotapi [9] package from Go libraries to implement the application. Also a common feature in every component, the usage of sirupsen/logrus [48] structured logger for logging. Go standard library also provides a package for logging but compared to the package mentioned above it has some limitations besides that sirupsen/logrus package is compatible with Go's standard logging package, so it is easy to use and comes with great additional features. For instance, seven different notification levels can be set for a specific log, which are the following in increasing order by severity:

1. Trace
2. Debug
3. Info
4. Warning

```
if len(os.Args) > 1 && os.Args[1] == "-v" {
                log.SetLevel(log.DebugLevel)
                log.Debug("Set loglevel to Debug")
        } else {
                log.SetLevel(log.WarnLevel)
        }
```

Listing 3: Switch between log levels at program's start

5. Error
6. Fatal
7. Panic

Based on the the feature this sirupsen/logrus provides, I have applied "switch" in the application code. Code snippet in listing 3 examines at the start of the program if -v(v for verbose) argument was supplied with the command, what initiated program to run or, this is carried out by help of Go "os" [34] package. If "-v" argument was present every log with level debug and above will be displayed on standard output if it was not only log with warn level and above will be displayed.

This is introduced in detail because this feature will affect plenty of CD practices and the main configuration settings are built around this feature for demonstrative purposes.

The application launches a web server at the start of the application and listens for incoming requests on web server's port 8080 to receive messages from Telegram servers. This is implemented with Go "http" [33] package's ListenAndServe() function.

5.1.1 Telegram

After the registration to Telegram, it is very easy to create a chatbot. Telegram also uses a chatbot called BotFather to serve a request for new chatbots. After sending "/newbot" command, a name and a username need to be chosen for the new chatbot, then Telegram will send a token, with the help of which bot will identify itself, towards Telegram servers. There is no further action needed before starting development.

Telegram offers two different options, how chatbot application can receive updates from telegram servers about new message in chat. It is either using webhook[51], or getUpdates [50] method. In case of implementing a webhook in the application, the Telegram server will send HTTP Post request to a specified endpoint every time a new message arrives in the chat.The second option is the getUpdates method with long polling; this is different from the first solution in a way, that the endpoint has to query the telegram server frequently for getting updates. Figure 5.1shows the difference between polling and webhook. The chatbot introduced in this thesis uses a webhook because it is considered a more efficient, faster, and more reliable method than long polling [69].

5.1.2 Commands

To demonstrate ChatOps practices, as a result of some commands chatbot carries out development operations, like "Deploy_PrimeGenerator", "Deploy_PrimeGenerator_debug",

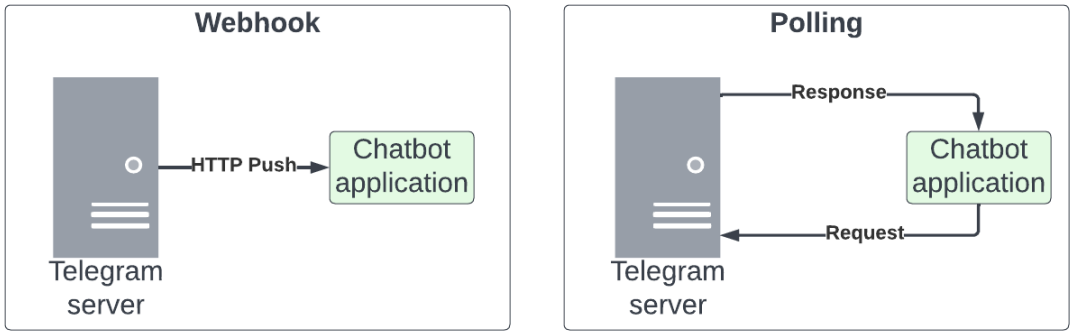

Figure 5.1: Getting updates from telegram with using webhook and with using polling

and Load while others like Ping "PrimeFactorization", "GenerateBigPrime", and "Convert" making chatbot to do more simple tasks. Non-operational type of commands on Figure 5.2

5.1.3 Ping

If a "Ping" command 5.2 is sent to chat with an answer "pong". It has zero calculation or calling function in the background only sends a text message back to chat, so this makes it suitable for testing purposes; for example, it proved its existence right at the time when I tried to reproduce the JSON object in order to load the testing component could play the role of telegram server during a load testing session. In JSON object, what made initially for trigger GenerateBigPrime command during development phase temporarily modified to trigger Ping command, and this resulted in less time to wait until it turned out at one iteration if JSON object was correctly defined or it was not.

5.1.4 PrimeFactorization

"PrimeFactorization" command 5.2 if it supplied with a positive whole number other than zero or a prime number will carry out prime decomposition and sends prime factors and a factor tree as an answer.

5.1.5 Convert

According to Hungarian language rules, the Convert command converts a positive whole number into words. The number must be less than one trillion. Figure 5.2 shows the Convert command in progress.

5.1.6 GenerateBigPrime

If a request arrives in the chatbot application to perform GenerateBigprime command, it will send a random prime number as a response. The function behind this service is to look for a random starting point between one hundred million and a billion; from that starting point backwards one by one, examines if the current number is a prime; if it is, it will

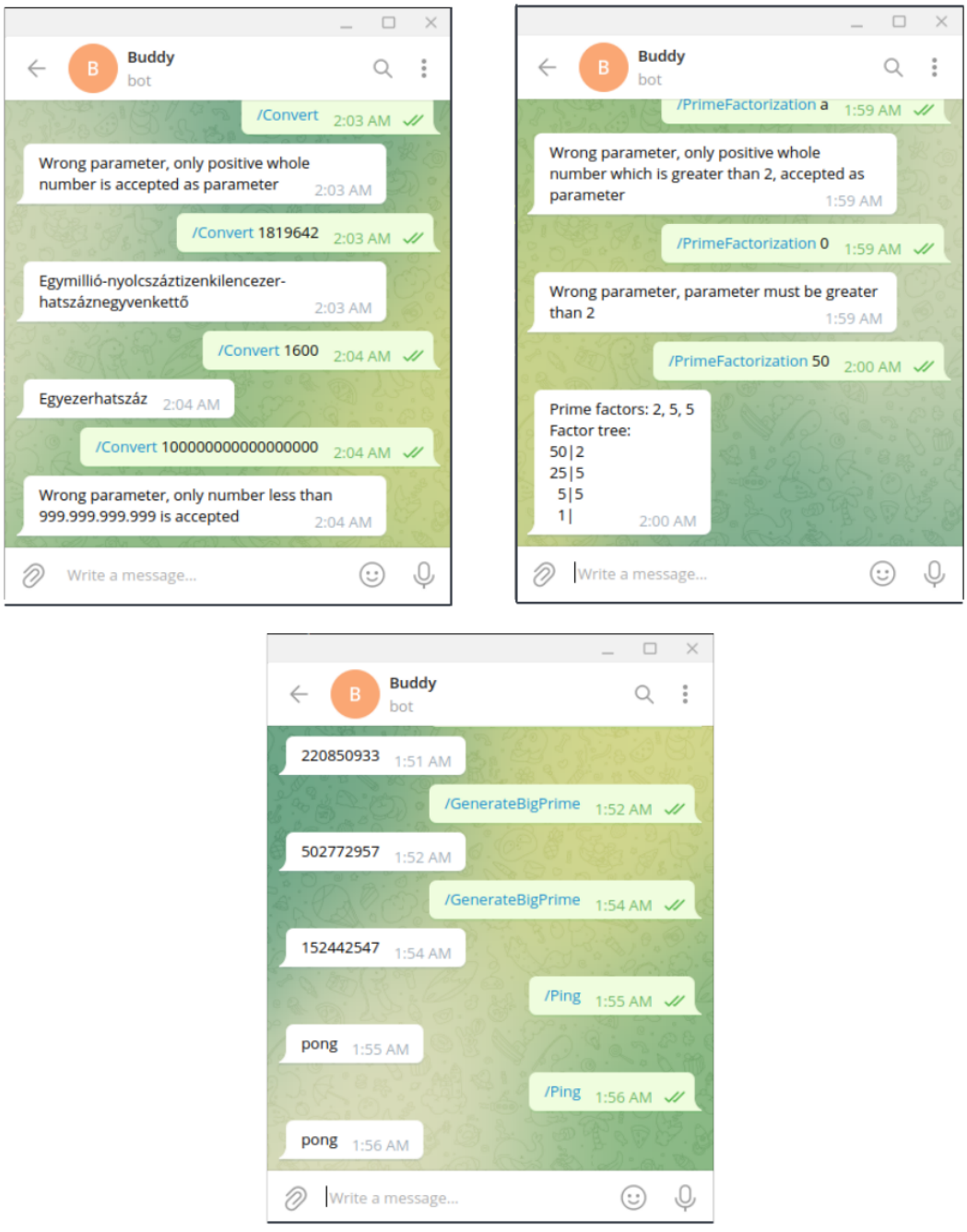

Figure 5.2: Non-operational type commands

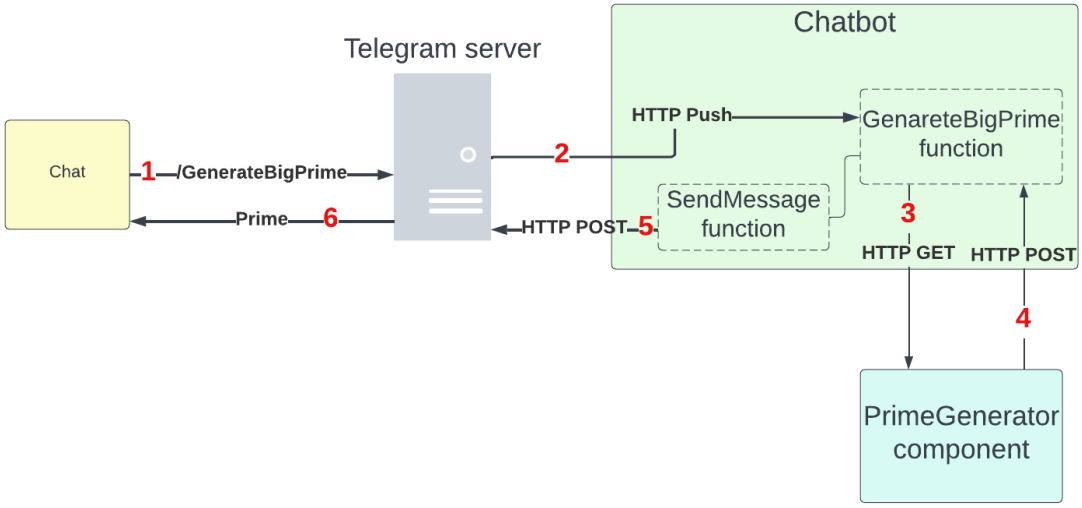

Figure 5.3: "GenerateBigPrime" command processes

return with that number. This command is implemented because it is a CPU-intensive task, making it suitable for load testing purposes.

The function what searching for prime numbers is implemented separately from the chatbot application, it is outsourced to a microservice component called "PrimeGenerator". "PrimeGenerator" component, just like a chatbot application, listens for incoming requests on it's webserver port 8080, and when a request arrives in this port, it calls the function what generates a prime number and sends a prime back as a response to the endpoint where a request came from. Figure 5.3 illustrate communication between chatbot and "PrimeGenerator" component.

5.1.7 Deploy_primeGenerator and Deploy_primeGenerator_debug

When a Deploy_primeGenerator command 5.5 is sent to the chatbot, the chatbot sends a request with HTTP POST method to GitHub API, which triggers a GitHub actions Workflow. When the Workflow finishes its run, the latest version of the PrimeGenerator application is deployed to Kubernetes Cluster. The process is the following step by step. Firstly the application extracts the chat ID from the message, which is required for the chatbot to be able to send a reply message to the same chat when the operation is over. Secondly, it serializes the JSON object that triggers the GitHub actions Workflow supplied with chat ID and sends it to GitHub API. Figure 5.4 shows the process step by step. When the Workflow starts to run, it builds a docker image based on the current version of PrimeGenerator application's code inside the GitHub repository and pushes it to GitHub container registry then sets configurations with Kustomize. When it is done, deploys PrimGenerator application to the Kubernetes cluster and pushes changes back to the repository so that everyone can see a deployment has been carried out. These processes will be introduced in detail in CI/CD section due to the CD part of the CI/CD pipeline is built from very similar elements with the same custom applications. However, the last step differs from what is implemented in CI/CD pipeline. This step, using CURL, sends JSON back to the chatbot with chat ID; this is the same chat ID that originated from chatbot. It is implemented in this way for two reasons. The first reason is introduced at the beginning of this paragraph; it is necessary for the chatbot for sending reply. The

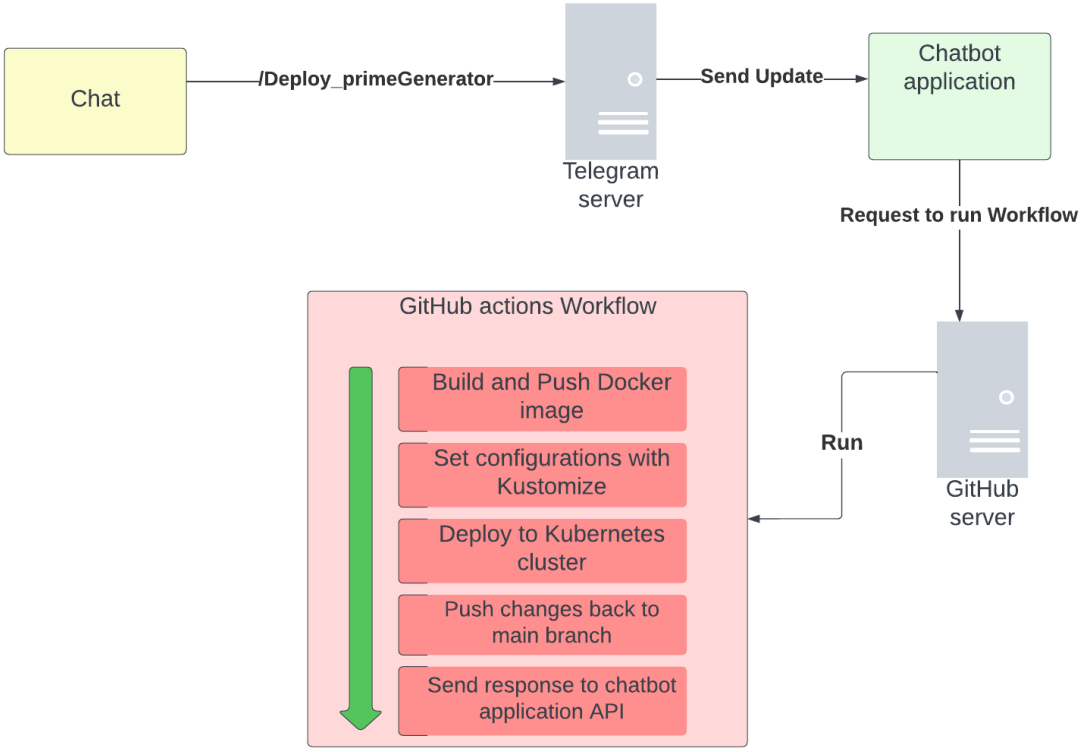

Figure 5.4: Deployment of "PrimGenerator" application from chat

second reason is, as Figure 5.6 illustrates, the Jobs inside this Workflow are dependent on each other, so in order for the sending_response_to_chat Job could start running; also, a Job called deploy_to_kubernetes_cluster and a Job called "docker" has to finish their run. This dependency ensures that the user only receives a reply message about the successful deployment when it is actually done, due to if any Job fails previously to sending_response_to_chat, the message will not be sent back to the chatbot.

The process is exactly the same in the case of Deploy_primeGenerator_debug command 5.5 except for one point. The difference is in configuration settings before deployment. This command deploys the application supplied with "-v" argument, so log messages will be displayed from debug level and above after applying this version.

5.1.8 Load

Sending a "Load" command on Figure 5.8 combined with certain parameters to the chatbot application starts a load testing process, the effect of which number of requests will be sent to the chatbot application "GenerateBigPrime" process by load testing component.

Similarly to "GenarateBigPrime" command, the process 5.7 starts in chatbot application, but a microservice performs the task called load testing component. Load command must be sent supplemented with exactly three parameters. First is the number of requests that have to be sent by load testing tool to chatbot application's "GenerateBigPrime" process. The second parameter is the frequency of requests in second; the third is the URL where the load testing tool sends requests to. After the chatbot application receives a "Load" command, it checks the limits and quality of parameters; for instance, the first parameter can not be more than 500 hundred, and it must be a number or if it is not, the

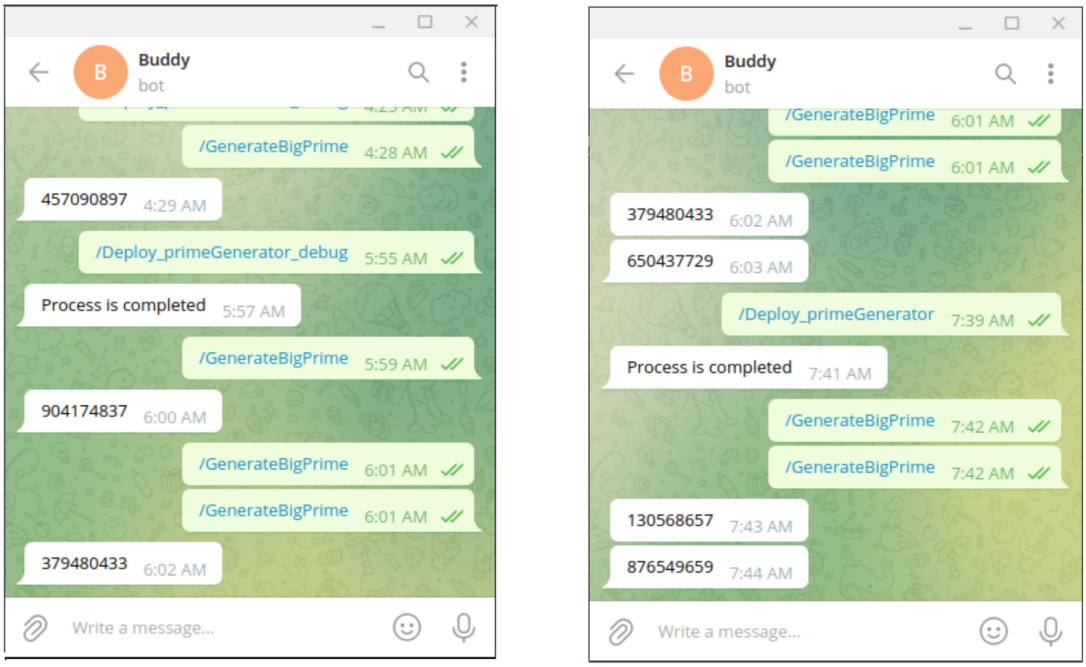

Figure 5.5: Deployment of "PrimeGenerator" component with two different configuration, initiated from chat

Figure 5.6: Dependency between GitHub actions Job's.

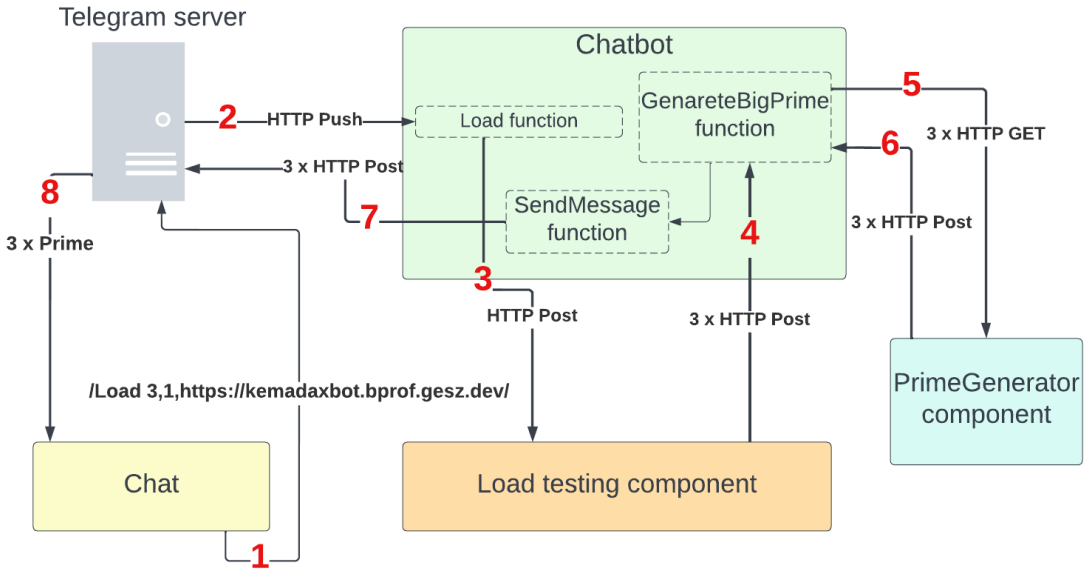

Figure 5.7: Load testing process

chatbot sends an error message back to chat. If every parameter passes checks, the chatbot application extracts the chat ID from the command, assembles the JSON object, and sends it to load testing tool. Extracting the chat ID is necessary to be handed over processes, for a reason in this manner, chatbot application can direct prime numbers resulted by load testing component's JSON object back to the same chat where "Load" command has been sent from. Since there is no need to send this request outside of the Cluster, because the chatbot application's Pod and the Pod that holds load testing tool are in the same Cluster and within the same namespace chatbot application can directly send this request to the Kubernetes Service (loadtestingtool-service) that targeting load testing tool.

After the request arrives on the load testing component's webserver port 8080, it deserializes the JSON object into a Go struct and assembles a new JSON that must be in the same form like JSON would have been sent to the chatbot application from the Telegram server in case of "GenerateBigPrime" command would have been initiated at the chat. After that, load testing microservice begins sending requests to the chatbot application with the defined parameters (number of requests, frequency). The number of requests means how many request load testing component sends, and frequency tells how much time should elapse between requests. At the end of this process, generated prime numbers are sent back to that chat from where the command had been sent.

5.2 State of the cluster

The Kubernetes Cluster, in which applications of this thesis are running in, is hosted by Amazon web services(AWS) platform. The Kubernetes Cluster was predefined for the applications at the early stage of development. The Cluster has been accessible for operations by kubeconfig file, which is used for identification towards Kubernetes previous to an operation is being initiated.

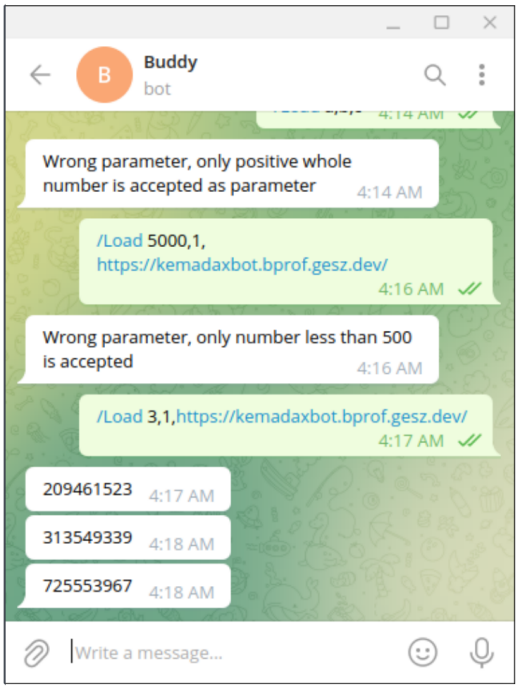

Figure 5.8: "Load" command

All Kubernetes Deployment and Service manifest's configuration based on each application deployed to the Cluster are very similar in the case of all three applications. All of the three application's Deployment relies on Kustomize configuration methods concerning the name of the Docker image, Kubernetes ConfigMap and in case of kemadaxbot-deployment(chatbot application) on Kubernetes Secrets, therefore besides Deployment manifest and Service manifest, a code snippet is illustrated from Kustomization file in listing 6.

If the chatbot application is deployed by specifications shown in Listing 4, Pod named kemadaxbot will run in the cluster in one instance; it will listen for network traffic on its' port 8080. The sensitive data like Telegram token will be injected into the Pod with the help of Kubernetes Secret and non-confidential like the URL for Telegram webhook, which is ConfigMap. Also, ConfigMap and Secret are generated by Kustomize configMapGenerator and secretGenerator based on data currently available in chatbot.env and chatbot.secret files. Both of their content is used in chatbot application as environment variables. The Pod is also supplemented with Kubernetes liveness probe. Liveness probe is constantly checking container health by sending, HTTP GET request on the endpoint defined in it's attributes. In case of kemadaxbot-deployment it will send it to container's port 8080/hello, and if HTTP response status code, Kubernetes receive from this endpoint is outside of a certain range(greater than or equal to 200 and less than 400), is identified as failure, and Kubernetes will automatically kill and restart the container [44].

The Service manifest in listing 5, what provides access to Pod for network traffic after it is applied called "kemadaxbot-service". Every request the aforementioned Service receives on it's port 80 will be forwarded on kemadaxbot Pod's port 8080 over TCP protocol.

Besides a Service what assigned to chatbot application an ingress rule in listing 7 also must be defined for chatbot application, in order to it can be requested over the Internet. In this way when a request arrives to the kemadaxbot.bprof.gesz.dev URL it is forwarded to

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kemadaxbot-deployment
  labels:
    app: kemadaxbot
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kemadaxbot
  template:
    metadata:
      labels:
        app: kemadaxbot
    spec:
      containers:
      - name: kemadaxbot
        image: imagename
        ports:
        - containerPort: 8080
        envFrom:
        - configMapRef:
            name: kemadaxbot-configmap
        - secretRef:
              name: kemadaxbot-secret
        livenessProbe:
          httpGet:
            path: /hello
            port: 8080
```

Listing 4: Chatbot application's Deployment

chatbot application Service's port 80 and from there on to chatbot application webserver's port 8080. If Telegram server wants to send an updates to chatbot application it has to send it to kemadaxbot.bprof.gesz.dev URL.

5.3 CI/CD pipeline

Every component of this application is inspected and deployed by automated processes of a CI/CD pipeline. In the case of "load testing" component and "primeGenerator" component the deployment occurs by running the GitHub actions Workflow responsible for deployment manually on GitHub online platform, or the very same Workflow is triggered by an HTTP request, sent to GitHub API as a consequence of chatbot command. However deployment of the chatbot application is carried out only as an effect of the release Event. Dependently from the content of the release tag, placed on a commit, chatbot application deployed by different configurations. All the aforementioned processes are based on GitHub actions platform, and Git version control and a single GitHub repository

```yaml
apiVersion: v1
kind: Service
metadata:
  name: kemadaxbot-service
  labels:
    app: kemadaxbot
spec:
  selector:
    app: kemadaxbot
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Listing 5: Chatbot application's Service

```yaml
images:
- name: imagename
  newName: ghcr.io/bproforigoss/kemadaxbot
  newTag: sha-be27360
configMapGenerator:
- envs:
  - env/chatbot.env
  name: kemadaxbot-configmap
secretGenerator:
- envs:
  - secret/chatbot.secret
  name: kemadaxbot-secret
```

Listing 6: Kustomization file for chatbot application

```yaml
rules:
    - host: kemadaxbot.bprof.gesz.dev
      http:
        paths:
        - backend:
            serviceName: kemadaxbot-service
            servicePort: 80
```

Listing 7: Routing rule for chatbot application from Ingress resources

that contains all of the three components code, configurations, and dependencies that are required for the application to run.

5.3.1 CI

Every time a new feature is implemented in any of the three applications and the code is pushed to the GitHub repository, unit tests and benchmark tests are run to check some functions of the code. For instance, table tests are checking PrimeFactorization function returns with the right prime factors, etc. The same GitHub actions Workflow, that tests ran on takes care of, running latest version Go programming language, and building the code as well. Also, an advanced level code inspection tool is a part of the continuous integration processes implemented in the aforementioned Workflow. A code linter specifically made for Go programming language called golangci-lint is inspecting various aspect of the code. Golangci-lint is a collection of numerous linters made for Go; and the following linters are enabled by default [35].

1. deadcode
2. errcheck
3. gosimple
4. govet
5. ineffassign
6. staticcheck
7. structcheck
8. typecheck
9. unused
10. varcheck

Applying these linters can save developers from wasting their time and ensures that the code is clean and use every element of code is up to date; also, make a suggestion to improve code when it finds some problem in code. For instance, deadcode searching for unused code, varcheck focuses on the identification of unused global variables, and gosimple reveals if code can be simplified. If a new linter is needed to perform a particular task, it can be added easily to the list of enabled linters, or at any time, a linter can be disabled with the help of a simple command.

5.3.2 CD

As it is mentioned, the processes that lead to the deployment of chatbot application to Kubernetes Cluster, will be introduced in this section step by step. Although GitHub actions Jobs in this Workflow are very similar to other deployments (chat-based deployments), it also has some differences. In chatbot application deployment process are always two Workflows involved at one iteration in order to avoid code repetition concerning the process of building and pushing Docker image to a registry based on a new Pod is made.

The first workflow triggered by a Release Event called Build_Docker_and_Trigger in it's first Job called Build_and_Push builds a Docker image and then pushes the image to the GitHub container registry what is a storage, for images built, based on chatbot application's code. In order to achieve this pre-made Actions are implemented. The first step it logs in to GitHub container registry, the second step extracts metadata like labels and tags for Docker, and the third step places an SHA tag and the "latest" moving tag on Docker image and pushes it to the registry.

Build_Docker_and_Trigger Workflow's second and third Job are calling two different workflow. If a release tag contains the word "dev" the second Job called deploy_by_tag_dev runs or if release tag contains the word "prod" the third Job called deploy_by_tag_prod is activated. All the deploy_by_tag_dev Job does, is calling the Workflow that deploys chatbot application with "-v" OS argument to the Cloud. In case of deploy_by_tag_prod is activated another Workflow is triggered to deploy chatbot application without "-v" OS argument. This process of choosing which Job should run is controlled by an if statement at the beginning of the aforementioned Jobs. In order to trigger a Workflow from one of these Job by sending an HTTP request to GitHub API, the Workflow what is triggered has to have workflow_dispatch event in it's own list of events the Workflow controlled by. In order to send a HTTP request from within one of the Job they uses CURL. This request has a JSON object what contains the Docker image, made by the Job Build_and_Push. Figure 5.9 introduces CI/CD pipeline built for chatbot application.

In the next few steps, I will introduce the case, if a release Event with tag "dev_v1.0.0" is put on a commit in GitHub repository, and as a reslut chatbot application deployed with "-v" argument. In this case deploy_by_tag_dev Job triggers the Workflow called deploy_chatbot_dev to run. The only Job in deploy_chatbot_dev Workflow at first step does the identification towards kubernetes Cluster by using kubeconfig file which until that point was stored in a GitHub secret. Second step installs Mozilla SOPS on the Runner, with help of which in third step temporarily decrypts the file that holds sensitive data like Telegram token inside the GitHub repository. After the file content is decrypted Kustomize SecretGenerator can produce Kubernetes Secrets based on its content, so it will be available inside the pod for the application after deployment. In the next step Docker image's name what has been sent in JSON object with HTTP request from Build_Docker_and_Trigger Workflow, is placed in the chatbot's Kubernetes Deployment manifest by using Kustomize set function and deploys chatbot application with "-v" argument into Kubernetes Cluster by using Kustomize patch function. Lastly if any changes happened compared to the previous deployment it pushes changes back to repository.

5.4 Monitoring in cloud environments

According to the best practices concerning a cloud-native application, the chatbot application of this thesis continuously monitored in cloud environment. All of the three application's logs are collected by Grafana Loki, and in chatbot application various kind of Prometheus metrics has been embedded in the code in the followings it will be introduced briefly. These metrics can provide guidance for future adjustment of a cloud-native application resources based on the requests number or on a time interval a specific process lasts. Furthermore these metrics can help establishes cost-effective manner of operating an application in cloud environment, and based on the observations they can indirectly contribute to an improved user experience.

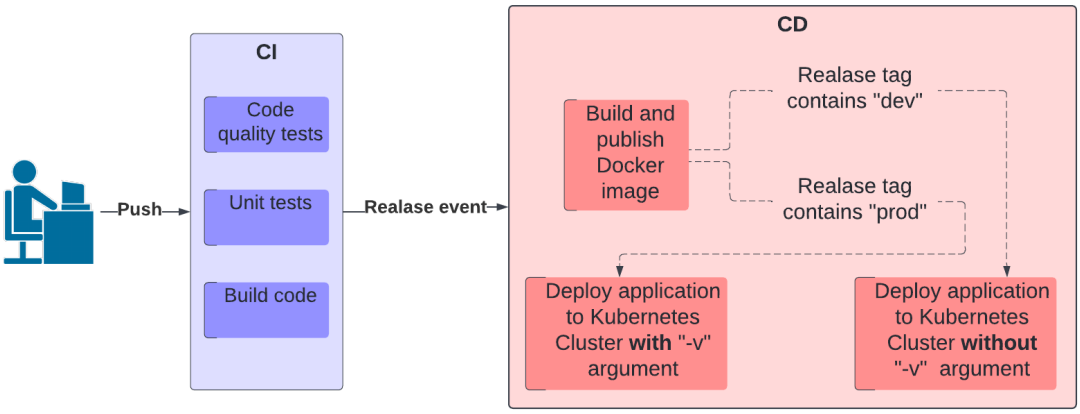

Figure 5.9: CI/CD pipeline for chatbot application

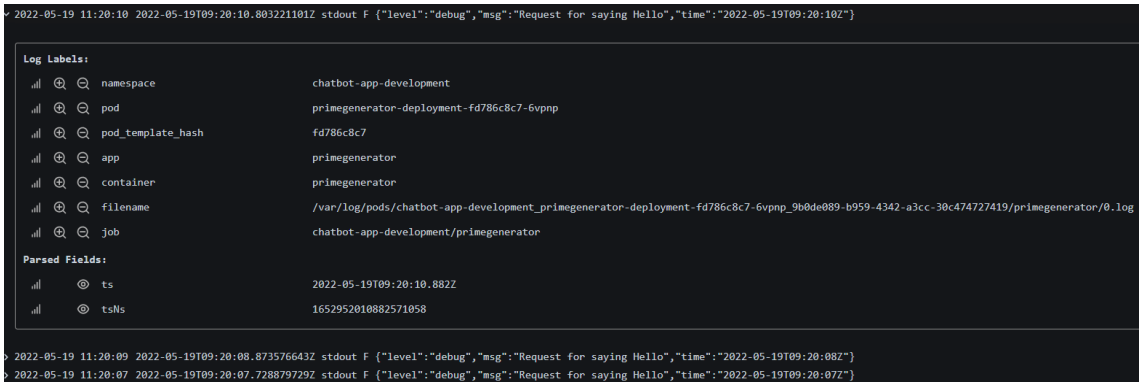

Figure 5.10: Logs aggregated by Grafana Loki

5.4.1 Metrics

To implement Prometheus metrics in code, I have used the Go client library. By default, Prometheus metrics are collected on /metrics endpoint after the aforementioned package is applied. As the methodology of how data is collected from Kubernetes Cluster by Prometheus was introduced in section Prometheus and Grafana, in this section, the exact measurements that have been applied will be presented. It is monitored and visualized in Grafana how many times a command was requested in the last one hour. The number of running pods in the Cluster is also observed in a manner as mentioned earlier. The other measurements are specifically targeting different aspects of "GenerateBigPrime" command and indirectly "PrimeGenerator" component. For example the average response duration of "PrimeGenerator" component or number of requests sent to "PrimeGenerator"component and received responses from "PrimeGenerator"component.

5.4.2 Logging

Logs produced by all of the three applications are collected by Promtail and aggregated by Grafana Loki 5.10.According to Grafana's documentation "Promtail is an agent which ships the contents of local logs to a private Grafana Loki instance or Grafana Cloud. It is usually deployed to every machine that has applications needed to be monitored" [36].

Number of running PrimeGenerator instances	2			3			4		
Number of requests sent by load testing component	5	10	15	5	10	15	5	10	15
Highest measured average response duration	2.5 min	6.9 min	12.6 min	1.6 min	7.7 min	6.7 min	3.1 min	5.2 min	6.9 min
Average response duration based on three iteration	7.3 min			5.3 min			5 min		

Figure 5.11: Results of load testing

5.4.3 Load testing

The "Primgenerator" component has been tested in three iterations by sending different loads to it with the help of the load testing component. At the time of testing, the "Primgenerator" component had the same limitation in every case: a maximum of 100 millicores of a CPU, or 0.1th fraction of a virtual CPU per Core [4]. In every iteration, 5, 10, and 15 requests were sent to the "Primgenerator" component, but the number of running instances of "Primgenerator" component was different.

In the first test case the number of running instances of "Primgenerator" component were 2 Figure 5.12 shows average response duration with 5 request, Figure 5.13 shows average response duration with 10 request and Figure 5.14 shows average response time with 15 request sent to "Primgenerator" component.

In the second test case 3 instances of "Primgenerator" were running in Kubernetes Cluster. Figure 5.15 shows average response duration with 5 request, Figure 5.16 shows average response duration with 10 request and Figure 5.17 shows average response time with 15 request sent to "Primgenerator" component.

In the third case 4 instances of "Primgenerator" components were waiting for requests. Figure 5.18 shows average response duration with 5 request, Figure 5.19 shows average response duration with 10 request and Figure 5.20 shows average response time with 15 request sent to "Primgenerator" component.

5.4.4 Summary

Based on Figure 5.11 that shows results of load testing the following conclusions can be drawn. Although the average response duration did not decrease in direct proportion to the increase in the number of Pods but the average response duration based on three

Figure 5.12: /Load 5,1,https://kemadaxbot.bprof.gesz.dev/

iteration is decreased to the increase in the number of currently running "Primgenerator" instances.

Unexpected differences in average response time, for example, in case of 3 running Pod instances average response duration is higher than in case of 2 running Pod instances by the same amount of load, can be explained with the normal functioning of PrimeGenerator component's process. The process starts to search for prime numbers in a range with a random starting point. This results in differences in some cases; finding a prime number in a short period of time sometimes finding a prime takes longer, but overall more Pod instance running in the Cluster means a shorter time to serve a request.

The decrease of average response duration in case greater number of running Pods and the very long average response duration in case of 2 running Pods justified the architectural change concerning the "Primgenerator" component because without horizontally scaling "Primgenerator", it would not have been as hand down to react the increased load.

Based on experiences established by the aforementioned load measurements, it will be easy to implement more advanced level automated scaling mechanism, for example Kubernetes Horizontal Pod Autoscaling (HPA) [37].

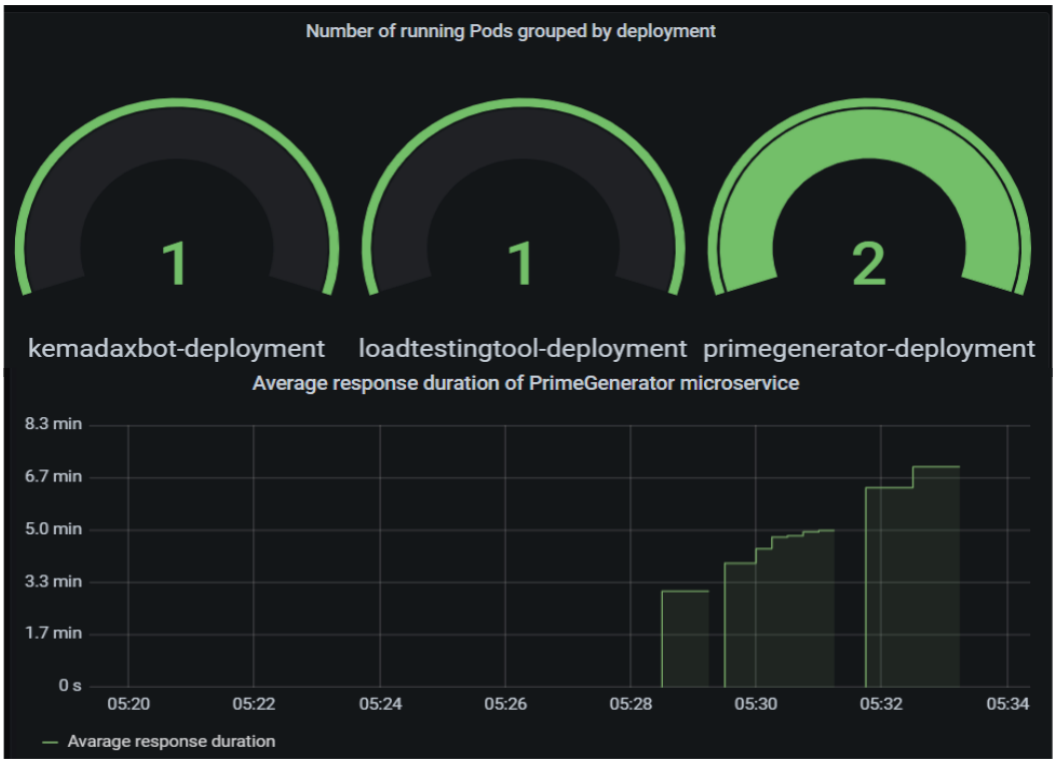

Figure 5.13: /Load 10,1,https://kemadaxbot.bprof.gesz.dev/

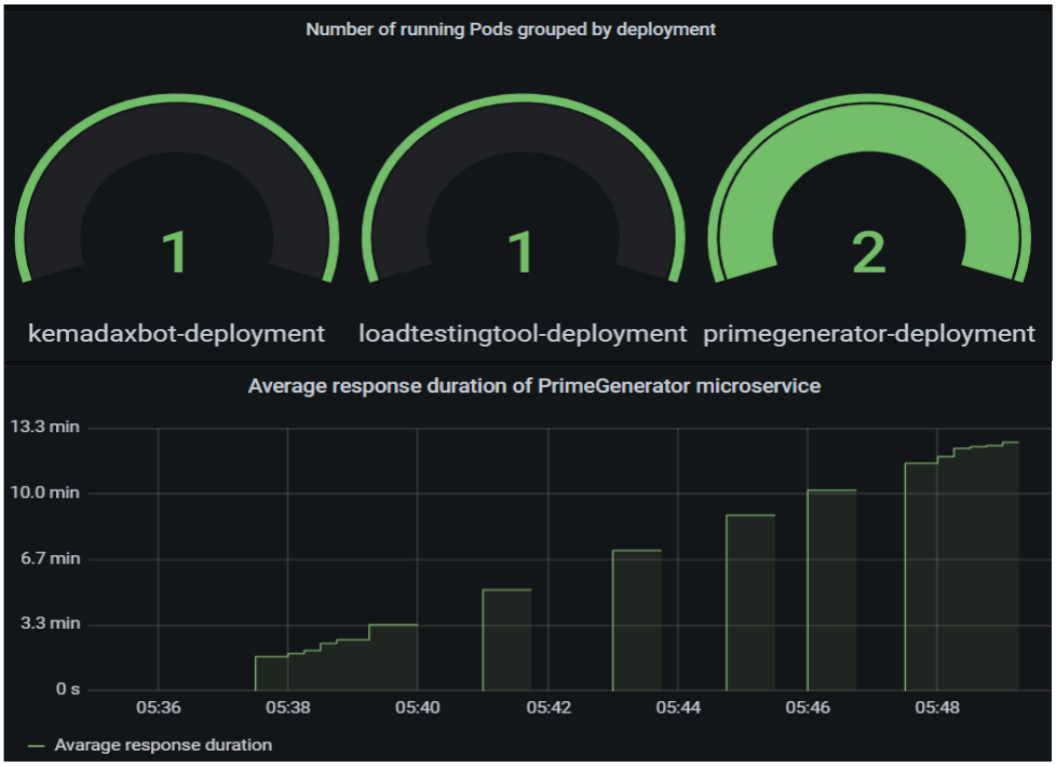

Figure 5.14: /Load 15,1,https://kemadaxbot.bprof.gesz.dev/

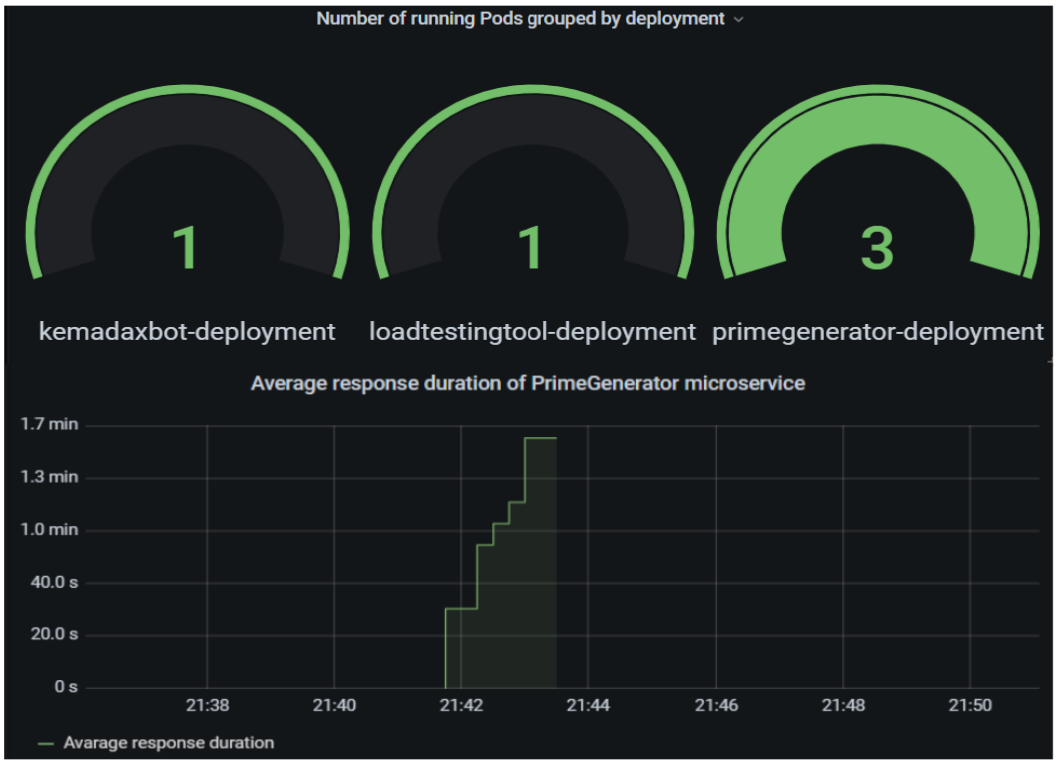

Figure 5.15: /Load 5,1,https://kemadaxbot.bprof.gesz.dev/

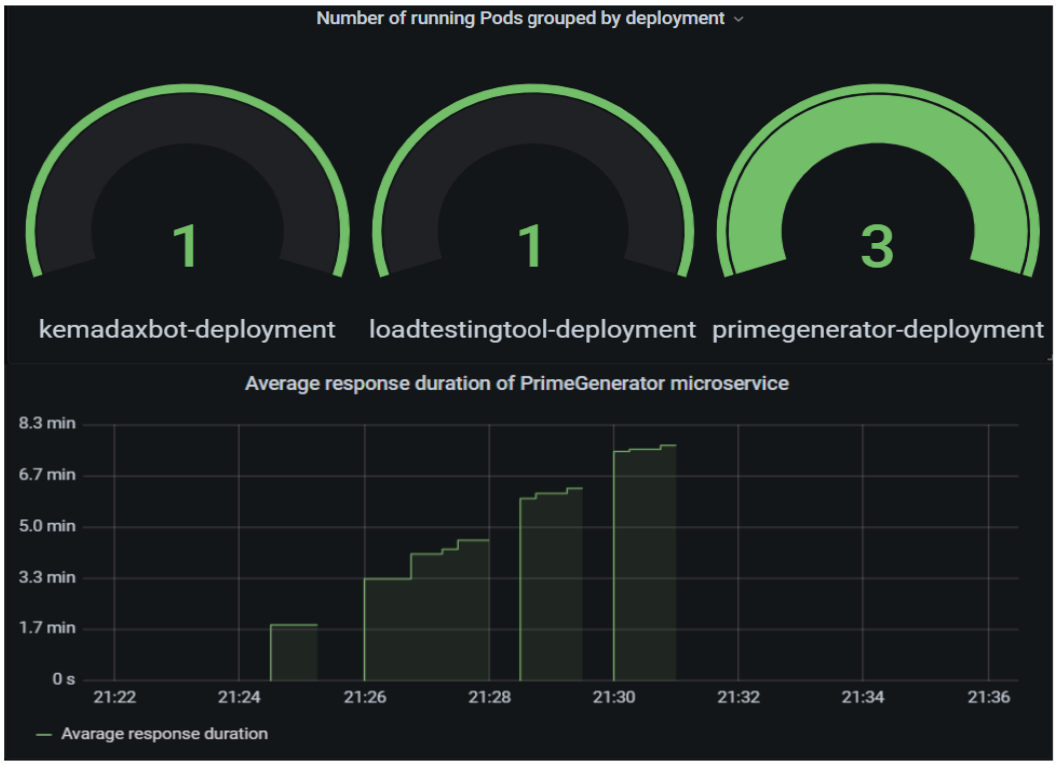

Figure 5.16: /Load 10,1,https://kemadaxbot.bprof.gesz.dev/

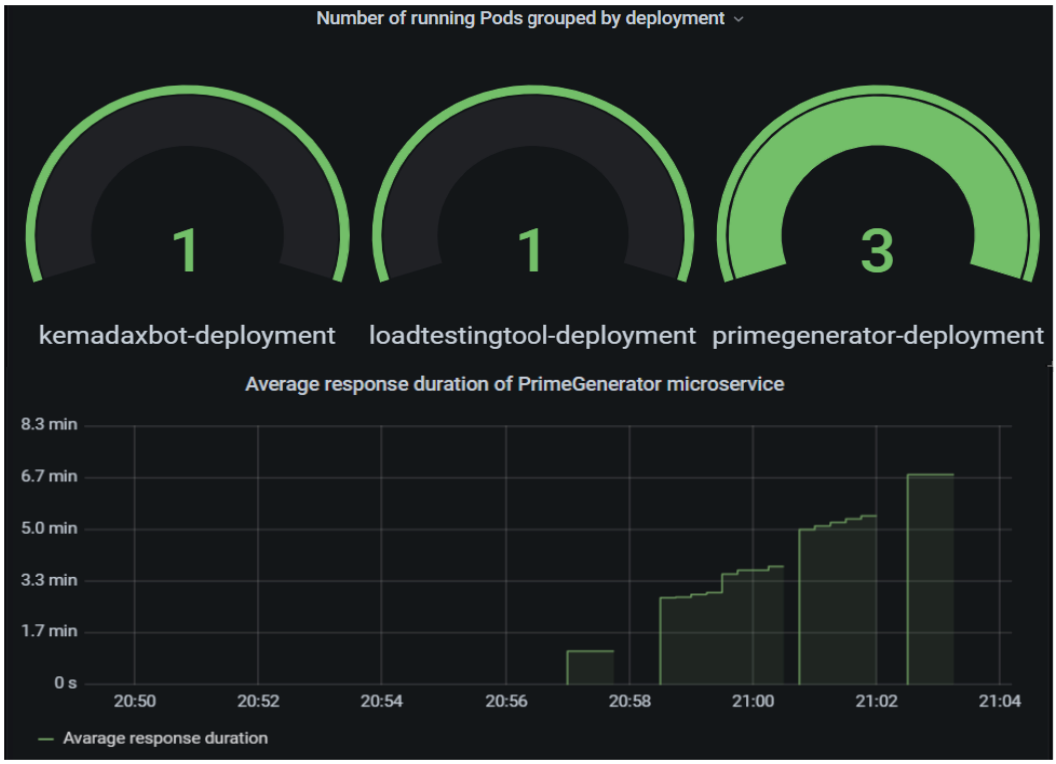

Figure 5.17: /Load 15,1,https://kemadaxbot.bprof.gesz.dev/

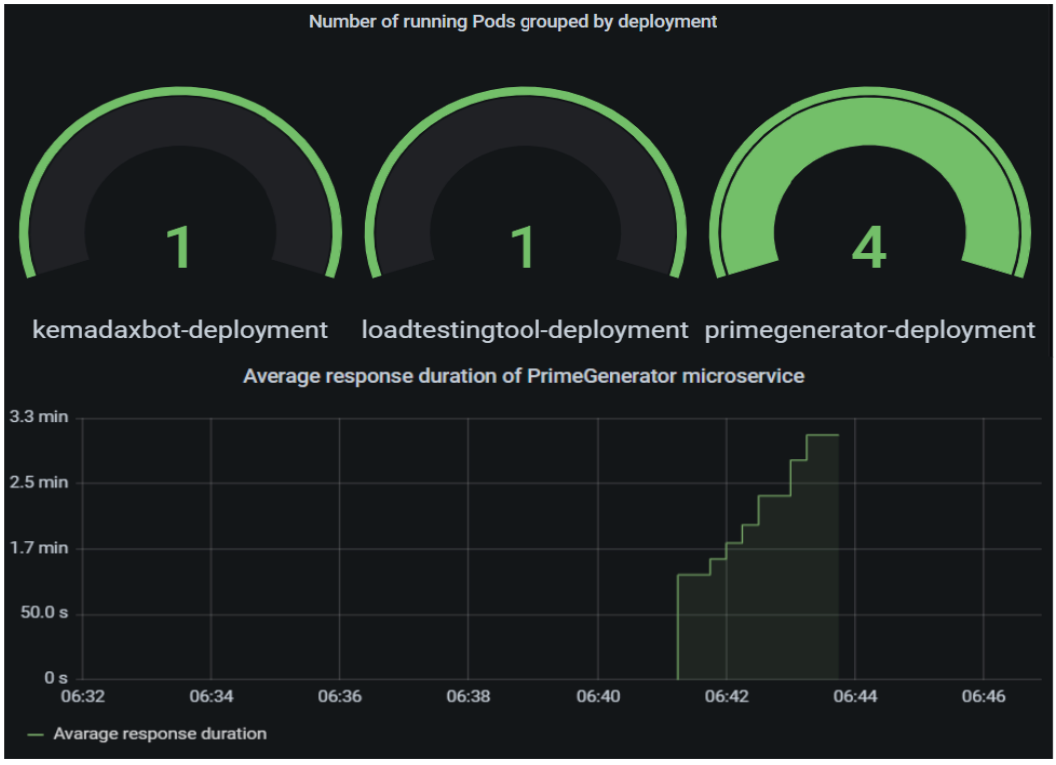

Figure 5.18: /Load 5,1,https://kemadaxbot.bprof.gesz.dev/

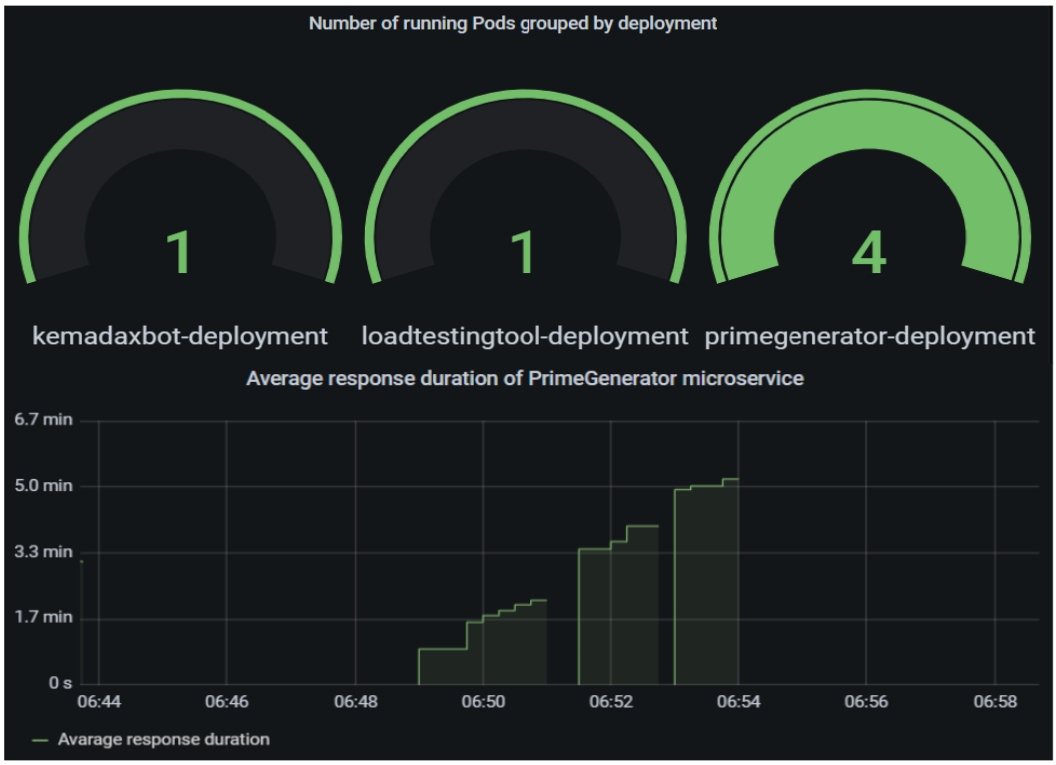

Figure 5.19: /Load 10,1,https://kemadaxbot.bprof.gesz.dev/

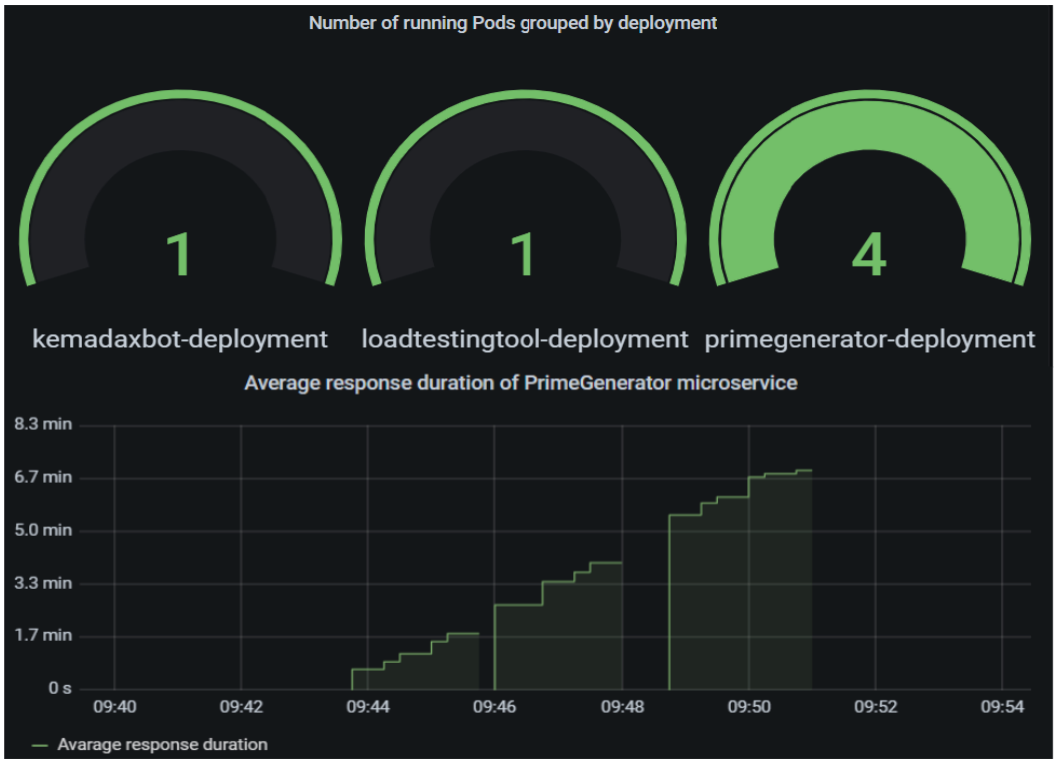

Figure 5.20: /Load 15,1,https://kemadaxbot.bprof.gesz.dev/

Chapter 6

Conclusion

The popularity of using Cloud computing resources and tools which help to leverage the benefits of the Cloud, is growing immensely due to its advantages. However, if a company decides to migrate its workload to Cloud, it should carefully choose Vendor or used ecosystem. Because lack of evaluation could lead to a Vendor lock-in or other type of lock-in situation. Suppose the company was cautious generally at the time of the decision. In that case, it should not be overcautious about making a commitment towards every technology because a certain degree of commitment is inevitable.

The chatbot application introduced in this thesis exemplifies well that a Cloud-native application that uses open source technologies supplemented with CI/CD pipeline, which is grounded by modern tools. Provides the possibility of the application's continuous integration and continuous deployment and thus makes delivering new features or bug fixes to end users fast and with minimal error.

Applying containerization makes the application lightweight and portable and ensures that the application can be run independently from the environment. In addition, since modern container orchestration tools like Kubernetes exist, it is not necessary to manage containers one by one manually, which facilitates the maintenance and operating of applications in Cloud environment. Using Kubernetes, also made applications highly available, easy to scale and robust against errors by providing automatism constantly checking the health of container or restarting them in case of an error occurs.

By promoting information to a modern tool like Prometheus, every aspect of a Cloud-native applications can be monitored continuously; it is really simple to add a new metric to the application or query existing metrics with PromQL. In this manner, easy to adjust resources on-demand or find and fix weaknesses based on data extracted from the application.

With the combination of modern tools and concepts introduced in this thesis, the benefits come with implementing chatbot in order to perform chat operations, might be useful for any company in or outside of the software development industry independently from the company size, due to it affects work efficiency positively and improves communication between team members.

Bibliography

[1] Why continuous delivery?, 2017. URL https://continuousdelivery.com/.

[2] Don't get locked up into avoiding lock-in, 2019. URL https://martinfowler.com/articles/oss-lockin.html#ShadesOfLock-in.

[3] 10 best practices to avoid cloud vendor lock-in, 2020. URL https://www.bmc.com/blogs/vendor-lock-in/.

[4] Managing resources for containers - Meaning of CPU, 2021. URL https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/.

[5] Loki documentation, 2021. URL https://grafana.com/oss/loki/.

[6] Kubernetes Deployment, 2021. URL https://kubernetes.io/docs/concepts/workloads/controllers/deployment.

[7] kubectl documentation, 2021. URL https://kubernetes.io/docs/reference/kubectl/overview/.

[8] State of continuous delivery report, 2021. URL https://cd.foundation/wp-content/uploads/sites/78/2021/06/CD-Foundation-State-of-CD-June-2021.pdf.

[9] Golang bindings for the telegram bot api, 2021. URL https://pkg.go.dev/github.com/go-telegram-bot-api/telegram-bot-api/v5.

[10] Smoke testing (software) by wikipedia, 2021. URL https://en.wikipedia.org/wiki/Smoke_testing_(software).

[11] Eks documentation, 2022. URL https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html.

[12] Aws documentation, 2022. URL https://aws.amazon.com/about-aws/global-infrastructure/regions_az/.

[13] CNCF cloud native definition v1.0, 2022. URL https://github.com/cncf/toc/blob/main/DEFINITION.md.

[14] What is devops?, 2022. URL https://about.gitlab.com/topics/devops/.

[15] Configure, 2022. URL https://about.gitlab.com/stages-devops-lifecycle/configure/.

[16] Grafana documentation, 2022. URL https://grafana.com/oss/prometheus/.

[17] Prometheus documentation, 2022. URL https://prometheus.io/docs/prometheus/latest/querying/basics/.

[18] Prometheus documentation, 2022. URL https://prometheus.io/docs/concepts/metric_types/.

[19] Prometheus documentation, 2022. URL https://prometheus.io/docs/introduction/overview/.

[20] What is cloud computing?, 2022. URL https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/#cloud-deployment-types.

[21] Cloud native transformation patterns documentation, 2022. URL https://www.cnpatterns.org/.

[22] Cloud native transformation patterns documentation, 2022. URL https://www.cnpatterns.org/patterns-library.

[23] What is the cloud?, 2022. URL https://www.cloudflare.com/learning/cloud/what-is-the-cloud/.

[24] Docker documentation, 2022. URL https://www.docker.com/resources/what-container/.

[25] Docker documentation, 2022. URL https://docs.docker.com/get-started/.

[26] Docker documentation, 2022. URL https://docs.docker.com/get-started/overview/.

[27] Docker documentation, 2022. URL https://docs.docker.com/get-started/overview/#docker-architecture.

[28] Github actins documentation, 2022. URL https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions.

[29] Gke documentation, 2022. URL https://cloud.google.com/compute/docs/instance-groups#managed_instance_groups.

[30] Gke documentation, 2022. URL https://cloud.google.com/compute/docs/load-balancing-and-autoscaling.

[31] Gke documentation, 2022. URL https://cloud.google.com/compute/docs/regions-zones.

[32] Gke documentation, 2022. URL https://cloud.google.com/compute/docs/regions-zones/global-regional-zonal-resources.

[33] Go http package, 2022. URL https://pkg.go.dev/net/http.

[34] Go os package, 2022. URL https://pkg.go.dev/os.

[35] Golangci-lint documentation, 2022. URL https://golangci-lint.run/.

[36] Grafana promtail, 2022. URL https://grafana.com/docs/loki/latest/clients/promtail/.

[37] Kubernetes documentation, 2022. URL https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/.

[38] Kubernetes documentation, 2022. URL https://kubernetes.io/docs/concepts/services-networking/ingress/.

[39] Kubernetes documentation, 2022. URL https://kubernetes.io/docs/concepts/workloads/pods/.

[40] Kubernetes documentation, 2022. URL https://kubernetes.io/docs/concepts/overview/components/.

[41] Kubernetes documentation, 2022. URL https://kubernetes.io/docs/concepts/configuration/configmap/.

[42] Kubernetes documentation, 2022. URL https://kubernetes.io/docs/concepts/configuration/secret/.

[43] Kubernetes documentation, 2022. URL https://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes/.

[44] Kubernetes documentation, 2022. URL https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/.

[45] Understanding Kubernetes objects, 2022. URL https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/.

[46] Kubernetes documentation, 2022. URL https://kubernetes.io/docs/concepts/services-networking/service/.

[47] Kustomize documentation, 2022. URL https://kustomize.io/.

[48] Logrus is a structured logger documentation, 2022. URL https://github.com/sirupsen/logrus.

[49] What is ci/cd?, 2022. URL https://www.redhat.com/en/topics/devops/what-is-ci-cd.

[50] Telegram documentation, 2022. URL https://core.telegram.org/bots/api#getupdates.

[51] Telegram documentation, 2022. URL https://core.telegram.org/bots/api#setwebhook.

[52] Devops by wikipedia, 2022. URL https://en.wikipedia.org/wiki/DevOps.

[53] Grafana's introduction by wikipedia, 2022. URL https://en.wikipedia.org/wiki/Grafana.

[54] CNCF (Cloud Native Computing Foundation) YouTube channel Alexis Richardson. Webinar: What is cloud native and why should i care?, 2017. URL https://www.youtube.com/watch?v=d_8Vly4_ofo.

[55] CNCF. Annual survey 2021, 2021. URL https://www.cncf.io/wp-content/uploads/2022/02/CNCF-AR_FINAL-edits-15.2.21.pdf.

[56] Rackspace Developers YouTube channel James Fryman. Chatops: Technology and philosophy, 2015. URL https://www.youtube.com/watch?v=IhzxnY7FIvg&list=PLUdHjbIG5WMsTUukNcmlZPZFfFcQGy664.

[57] CNCF (Cloud Native Computing Foundation) YouTube channel Jamie Dobson. Webinar: What is cloud native and why should i care?, 2021. URL https://www.youtube.com/watch?v=mSShgnBDn0s.

[58] Dávid Richárd Kertész. Best practices of cloud native application development. Bachelor of profession's thesis, Budapest University of Technology and Economics, Budapest, 2021.

[59] GOTO Conferences YouTube channel Martin Fowler. Microservices • martin fowler • goto 2014, 2014. URL https://www.youtube.com/watch?v=wgdBVIX9ifA.

[60] Adit Modi. Comparing managed kubernetes services: Eks vs. aks vs. gke, 2021. URL https://dev.to/cloudtech/comparing-managed-kubernetes-services-eks-vs-aks-vs-gke-4l8d.

[61] Jonathan Johnson Muhammad Raza. Chatops explained: How chatops supports collaboration, 2020. URL https://www.bmc.com/blogs/chatops/.

[62] Sahandi R. Tian F. Opara-Martins, J. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. *Journal of Cloud Computing*, 5, 4 2016. https://doi.org/10.1186/s13677-016-0054-z.

[63] Sten Pittet. Continuous integration vs. delivery vs. deployment, 2022. URL https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

[64] Sean Regan. What is chatops? a guide to its evolution, adoption, and significance, 2016. URL https://www.atlassian.com/blog/software-teams/what-is-chatops-adoption-guide.

[65] Bob Reselman. An illustrated guide to 12 factor apps, 2021. URL https://www.redhat.com/architect/12-factor-app.

[66] Alexis Richardson. What is cloud native and why should i care?, 2017. URL https://www.cncf.io/wp-content/uploads/2020/08/What-is-Cloud-Native-CNCF-Webinar-23-Feb-2017-1.pdf.

[67] SlashData. The state of cloud native development, 2021. URL https://www.cncf.io/wp-content/uploads/2021/12/Q1-2021-State-of-Cloud-Native-development-FINAL.pdf.

[68] John Steven. What's the difference between agile, ci/cd, and devops?, 2018. URL https://www.synopsys.com/blogs/software-security/agile-cicd-devops-difference/.

[69] SendGrid Team. What's a webhook?, 2014. URL https://sendgrid.com/blog/whats-webhook/.

[70] Seigo Uchida. Introduction to kustomize, 2022. URL https://speakerdeck.com/spesnova/introduction-to-kustomize.

[71] Aimee Ukasick and SIG Docs. Kubernetes documentation survey, 2019. URL https://kubernetes.io/blog/2019/10/29/kubernetes-documentation-end-user-survey/.

[72] A. Wiggins. The Twelve-Factor App, 2017. URL https://12factor.net/.

[73] TechWorld with Nana YouTube channel. Containers on aws overview: Ecs | eks | fargate | ecr, 2020. URL https://www.youtube.com/watch?v=AYAh6YDXuho&t=1234s.

[74] TechWorld with Nana YouTube channel. Github actions tutorial - basic concepts and ci/cd pipeline with docker, 2020. URL https://www.youtube.com/watch?v=R8_veQiYBjI.

[75] TechWorld with Nana YouTube channel. How prometheus monitoring works | prometheus architecture explained, 2020. URL https://www.youtube.com/watch?v=h4Sl21AKiDg.

[76] TechWorld with Nana YouTube channel. What is devops? really understand it | devops vs sre, 2022. URL https://www.youtube.com/watch?v=0yWAtQ6wYNM.

List of Figures

List of Source Codes