

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

UTILISATION OF A CHAT BOT APPLICATION IN CI/CD PROCESSES

VYUŽITÍ CHAT BOT APLIKACE V PROCESECH CI/CD

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. EVŽEN GASTA**
AUTOR PRÁCE

SUPERVISOR **RNDr. MAREK RYCHLÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

156413

Institut:	Department of Information Systems (UIFS)
Student:	**Gasta Evžen, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Software Engineering
Title:	**Chat Bot Applications in CI/CD Processes**
Category:	Software Engineering
Academic year:	2023/24

Assignment:

1. Familiarise yourself with the Google Workspace environment and explore the possibilities of creating Google Chat applications (chatbot applications). Also, investigate other technologies and frameworks for developing chatbot applications (e.g., from companies like OpenAI, Microsoft) and compare their capabilities.
2. Acquaint yourself with the Jenkins technology for software integration and deployment (CI/CD). Conduct an analysis of the current utilisation of Jenkins within a selected company, focusing on the communication between development and operations teams and their infrastructure components for CI/CD processes.
3. Propose a method for utilisation of a chatbot application to interact with a Jenkins server, the components, and teams within the analysed CI/CD processes. Select an appropriate technology for implementing such a chatbot, considering established communication tools for team collaboration.
4. After consultation with the supervisor, implement the proposed chatbot application. Also, implement a set of sample tasks and tests to verify the functionality of the application.
5. Integrate the developed application into communication channels for team collaboration within the selected company's CI/CD processes and evaluate its contributions.
6. Evaluate the results of your work and suggest possible extensions.

Literature:

- Smart, J. F. (2011). Jenkins: The Definitive Guide. United States: O'Reilly Media. ISBN 9781449313067.
- Batish, R. Voicebot and Chatbot Design: Flexible Conversational Interfaces with Amazon Alexa, Google Home, and Facebook Messenger. United Kingdom: Packt Publishing, 2018. ISBN 9781789136883.
- Build a Google Chat app with Google Cloud Functions [online]. Google [cit. 2023-09-14]. Available at: https://developers.google.com/chat/quickstart/gcf-app
- Jaworski, M., Ziadé, T. Expert Python Programming: Master Python by Learning the Best Coding Practices and Advanced Programming Concepts, 4th Edition. India: Packt Publishing, 2021. ISBN 9781801076197.
- Ayeva, K., Kasampalis, S. Mastering Python Design Patterns: A Guide to Creating Smart, Efficient, and Reusable Software, 2nd Edition. United Kingdom: Packt Publishing, 2018. ISBN 9781788832069.

Requirements for the semestral defence:
Items 1, 2, and 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Rychlý Marek, RNDr., Ph.D.**
Consultant:	Ing. Jindřich Luža
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2022

Submission deadline: 17.5.2023
Approval date: 30.10.2023

Abstract

This Master's thesis focuses on optimizing the precious time of Container Verification Pipeline (CVP) team members in Red Hat by integrating chatbot into their day-to-day DevOps workflows. The goal of this thesis is to lower the time required to execute various tasks and, thanks to that, save human resources. For the development of this chatbot, an AI service from Google was used, that allows the user to create an AI agent with ease. The chatbot is split into two parts (due to security and accessibility) and uses Request/Response architecture. In the end, tasks will be evaluated and compared to the current execution of tasks.

Abstrakt

Tato diplomová práce se zaměřuje na optimalizaci drahoceného času členů týmu Container Verification Pipeline (CVP) ve společnosti Red Hat, pomocí chatbot aplikace, která je integrována do jejich každodenních DevOps procesů. Cílem této práce je snížit potřebný čas k vykonání různých úkolů a díky tomu snížit lidské zdroje. Pro vývoj tohoto chatbota byla použita služba s umělou inteligencí od společnosti Google, která umožnuje jednoduše vytvořit agenta umělé inteligence. Chatbot aplikace je rozdělena na dvě části (kvůli bezpečnosti a dostupnosti) a používá architekturu Dotaz/Odpověď. Nakonec budou vyhodnoceny a porovnány úkoly s použitím chatbota a bez něj.

Keywords

Jenkins, CI/CD, DevOps, OpenShift, Amazon SQS, Red Hat AMQ, chatbot, AI, Google Dialogflow

Klíčová slova

Jenkins, CI/CD, DevOps, OpenShift, Amazon SQS, Red Hat AMQ, chatbot, AI, Google Dialogflow

Reference

GASTA, Evžen. *Utilisation of a Chat Bot Application in CI/CD Processes.* Brno, 2024. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor RNDr. Marek Rychlý, Ph.D.

Rozšířený abstrakt

Cílem této diplomové práce je vytvořit aplikaci, která pomůže v každodenních úkolech členů Container Verification Pipeline (CVP) týmu ve firmě Red Hat. To bude dosaženo vytvořením chatbot aplikace, která bude integrovaná do prostředí Google Workspace (přesněji do Google Chatu). Díky chatbot aplikaci budou mít členové CVP týmu možnost spouštět nejčastější úkoly přímo z Google Chatu a to bez přihlašování do ostatních služeb, hledání odkazů, přecházení mezi odkazy, nebo zkoumání výsledků sestavení (buildu) či chyb, které při sestavení nastaly.

Práce popisuje jaké služby a projekty bude chatbot aplikace používat, jak fungují a jak s těmito služby bude aplikace komunikovat (převážně pomocí REST API). Mezi takové služby patří například Amazon SQS, Google Dialogflow, nebo Red Hat AMQ a další. Chatbot aplikace bude rozdělena na dvě části, nasazené na Google Cloud Run a Red Hat OpenShift Container Platform, které mezi sebou budou komunikovat pomocí architektury Dotaz/Odpověď, z původní Producer/Konsumer architektury.

Část aplikace, která je nasazena na službě Google Cloud Run se chová jako server díky frameworku Flask, který tvoří základní prvek této části aplikace. Při přijetí zprávy (frameworkem Flask) poslané uživatelem z Google Chatu je zpráva přeposlána do služby Google Dialogflow, kde je detekovaný záměr ve zprávě a je vrácena odpověď (většinou se jedná o žádost pro "naplnění" povinných parametrů daného záměru). Tato odpověď z Google Dialoglow je poté přeposlána zpátky uživateli do Google Chatu. Většinou má tento cyklus několik iterací. Jakmile jsou všechny povinné parametry v Dialogflow získané od uživatele, je zaslaná zpráva do Amazon SQS fronty, kde čeká než o ni druhá aplikace zažádá. Celá aplikace je nasazena jako Docker kontejner, ve které jsou spuštěny dvě vlákna. První z nich kontroluje počet zpráv v SQS frontě a jakmile se ve frontě objeví zpráva, je zpráva přesunuta do lokální fronty. Z této lokální fronty je zpráva zpracována a přeposláná druhým vláknem do Google Chatu. Na službě Google Cloud Run je vždy dostupná minimálě jedna ze tří instancí a to z toho důvodu, aby bylo možné vždy poslat zprávu s výsledkem úkolu uživateli do Google Chatu (Google Cloud Run umožňuje mít instanci aktivní pouze v době, kdy se zpracovává příchozí požadavek).

Druhá část aplikace je nasazená na produktu Red Hat OpenShift Container Platform, kde je podobně jako na službe Google Cloud Run aplikace nasazena ve formě Docker konteinerů, pro které byly vytvořeny tzv. "BuildConfigy" a "DeploymentConfigy". Díky těmto konfiguračním souborům se vždy na základě aktivity v git repozitáři vytvoří a nasadí nový Docker kontejner, který běží ve třech různých podech (pro případ, že se v Amazon SQS frontě objeví více časově náročných úkolů). Aplikace má také dvě vlákna, které se starají o běh aplikace. První z nich má identické chování s vláknem na kontrolu a získávání zpráv z Amazon SQS fronty v Google aplikaci. Druhé vlákno se stará o vykonávání jednotlivých úloh z lokální fronty získaných zpráv z SQS fronty. Jakmile je daný úkol vykonaný, je výsledek poslán do Amazon SQS fronty přiřazené Google aplikaci.

Obě aplikace mají upravenou verzi vláken, díky čemuž se aplikace, pokud alespoň jedno vlákno selže, ukončí celá. Díky vlastnostem OpenShiftu a "HealthCheckům" ve službě Google Cloud Run se aplikace v případě pádu automaticky restartuje, díky tomu by měla mít aplikace dobrou dostupnost s minimalizací výpadků.

Chatbot má k dispozici několik úloh, které jsou pro členy CVP týmu nejčastejší a ze kterých si může uživatel vybrat, spolu s jednoduchým zobrazením URL odkazů na interní stránky CVP teamu. Mezi nejběžnější úlohy se může zařadit např. znovu spuštění daného sestavení na serveru Jenkins, získání stavu služeb na kterých je tým CVP závislý, získání stavu o nejnovějších sestaveních na serveru Jenkins, přidání tzv. "waiverů" k jednotlivým

sestavením, získání využití projektů v OpenShiftu a využití daného serveru Jenkins, nebo vypsání chyb u daného sestavení.

Díky chatbot aplikaci bude v určitých případech možné úplně odstranit prostředníky (CVP tým), např. při znovu spuštění jednotlivých sestavení, nebo alespoň snížit čas potřebný pro vykonávání úkolů o desítky minut denně a počet kroků které musí člen CVP týmu vykonat. Díky čemuž se ročně ušetří stovky hodin práce, což ušetří týmu CVP odhadem desítky tisíc dolarů (stovky tisíc korun). Členové CVP týmu vyhodnotili chatbot aplikaci jako "Bot se ukázal jako účinný při zpracování různých příkazů, integraci s Jenkinsem a dalšími službami. Jeho odezva a přesnost při rozpoznávání neplatných formátů NVR jsou chvályhodné". Chatbot aplikace se bez problémů integrovala do každodenních procesů v týmu a je denně používána.

Utilisation of a Chat Bot Application in CI/CD Processes

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Marek Rychlý. The supplementary information was provided by Mr. Jindřich Luža. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Evžen Gasta
May 14, 2024

Acknowledgements

I want to thank my family and my girlfriend for fully supporting me during my studies. I would also like to thank my consultant and my team who helped me in my problems.

Contents

1 Introduction **3**

2 DevOps **4**
2.1 CI/CD . . . 7
2.1.1 Continuous Integration . . . 8
2.1.2 Continuous Delivery . . . 8
2.1.3 Continuous Deployment . . . 8
2.2 Jenkins . . . 8

3 Cloud Service **10**
3.1 Cloud computing models . . . 10
3.1.1 Infrastructure as a Service (IaaS) . . . 11
3.1.2 Platform as a Service (PaaS) . . . 11
3.1.3 Software as a Service (SaaS) . . . 11

4 Chatbots **13**
4.1 Conversation UI . . . 13
4.2 Natural Language Processing . . . 14
4.3 Transformer neural network . . . 14
4.3.1 Attention mechanism . . . 15
4.3.2 Model Architecture . . . 15
4.3.3 Google . . . 16
4.4 Chatbot frameworks and services . . . 17
4.4.1 IBM Watsonx . . . 17
4.4.2 Microsoft Bot Framework platform and Azure AI Bot Service . . . 18
4.4.3 Meta Wit.ai . . . 18
4.4.4 Amazon Lex . . . 20

5 Architecture **21**
5.1 Building a Google Chat App . . . 21
5.1.1 Architectures of Google Chat App . . . 21
5.1.2 Google Chat messages . . . 24
5.2 Jenkins in CVP team . . . 27
5.2.1 Apache ActiveMQ . . . 27
5.2.2 CVP's Workflow . . . 28
5.2.3 Communication with CVP team in Red Hat . . . 30
5.3 Components . . . 30
5.4 Flask . . . 31

5.5 Simple Queue Service . . . 32
5.5.1 Queues . . . 32
5.5.2 Lifecycle of messages in Amazon SQS . . . 32
5.6 Google's component . . . 33
5.7 Red Hat's component . . . 34

6 Implementation **36**
6.1 Use Cases . . . 36
6.1.1 Re-triggering a failed Jenkins job . . . 37
6.1.2 Getting status from Tactical Status Page . . . 38
6.1.3 Getting status from Jenkins . . . 39
6.1.4 Getting usage from Jenkins . . . 39
6.1.5 Getting errors from team Jenkins log . . . 39
6.1.6 Adding waivers . . . 40
6.2 Difficult parts . . . 42
6.2.1 Change of the messaging . . . 42
6.2.2 Problem with messages . . . 43

7 Evaluation and Conclusion **45**
7.1 Comparison to the current execution of tasks . . . 45
7.2 Possible upgrades . . . 47

Bibliography **49**

Chapter 1

Introduction

In today's era of chatbots, AI and automation, these can be found in almost everything from mobile phones to healthcare. The main goal of all of those things is to support a human in self-repeating tasks that consume their precious time. Thanks to AI, automatization and chatbots is possible to make humans more productive in day-to-day tasks and let them focus on more relevant tasks instead of doing the same tasks over and over again. Especially in industries like healthcare and IT, every second can be expensive and sometimes even save lives.

This thesis focuses on the usage of a chatbot in processes of DevOps with the goal of improving productivity and lowering time in solving repeating tasks in the Container Verification Pipeline (CVP) team, which is part of Cloud Distribution in the company Red Hat, where I'm currently working at the time of writing this master's thesis. Currently, the CVP team provides the testing phase of Common Vulnerabilities and Exposures (CVE) related container rebuilds for automated releases, provides support for basic testing of containers, acts as a release gatekeeper for images rebuilt due to CVE's and runs for every container and operator bundle build (Kubernetes Operator is a method of packaging, deploying and managing a Kubernetes-native application. Cited from websites of Red Hat [38]).

Chapter number 2 focuses on the principles of DevOps and explains what CI/CD and project Jenkins are. In the 3rd chapter, cloud computing models are described alongside Google Workspace. Chatbots and Natural Language Processing with types of Google Chat App and types of conversation are explained in chapter number 4. The 5th chapter focuses on the architecture of the chatbot application that will be created for the CVP team and how the popular project Jenkins is used inside the team. In this chapter are also explained the technologies that are used to build this application. Implementation of various use cases (tasks) of the chatbot will be described in the 6 chapter alongside with problems and difficult parts, that were encountered when working on this thesis. The last chapter compares the current workflow processes with the updated process achieved by the created application, with possible improvements and feedback from the people who used the application.

Chapter 2

DevOps

The term DevOps combines two abbreviated words, dev(elopement) and op(eration)s. They refer to the respective organized units in an IT department, which used to be separated in traditional functional organizations and pursue different goals. In essence, DevOps may be conceived as a management concept consisting of principles, practices and tools designed to counteract typical problems in the collaboration between software development and operation units. Cited from book *Continuous innovation with DevOps* [36].

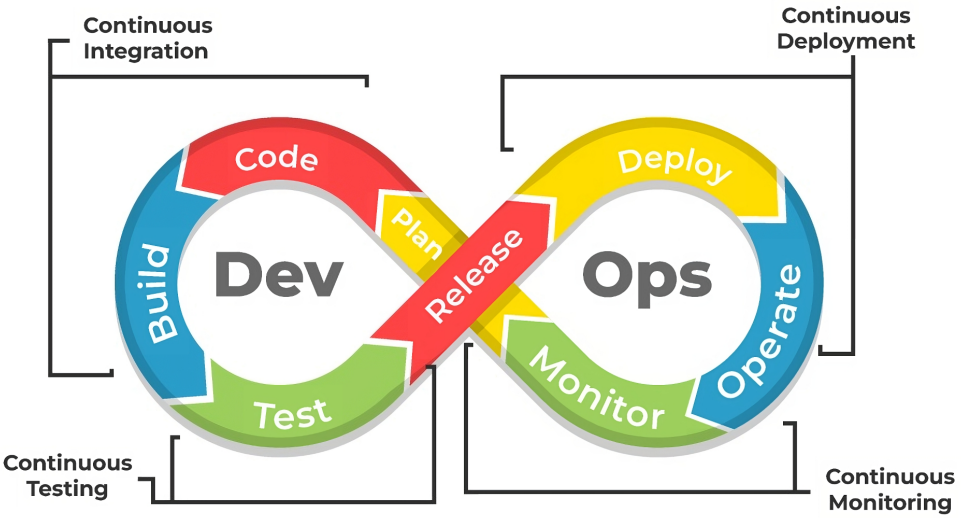


Figure 2.1: DevOps continuous lifecycle of software delivery and operations (SDO). Image available at websites of GeeksForGeeks [13].

Performance level	Deployment frequency	Change lead time	Change failure rate	Failed deployment recovery time
Elite	On demand	<1 day	5 %	<1 h
High	Between once per day and once per week	Between one day and one week	10 %	<1 day
Medium	Between once per week and once per month	Between one week and one month	15 %	1 day to 1 week
Low	Between once per week and once per month	Between one week and one month	64 %	1 month to 6 months

Table 2.1: View into how this year's survey respondents are doing with software delivery performance from *State of DevOps Report* from 2023 [21].

In a technical report *State of DevOps* from 2021 [19], can be found a comparison of elite performers against the low performers with these results:

- 973 times more frequent code deployments
- 6570 times faster lead time from commit to deploy
- Lower change failure rate by a fraction of 3
- And 6570 times faster time to recover from incidents

The core ideas of DevOps have become visible in the widely accepted guiding principles. Those principles are hidden in the acronym CALMS, which can be found, e.g. in [36]:

- Culture - requirement to have shared responsibility of all stakeholders for the delivery of quality software as one piece
- Automation - another principle of DevOps is reducing development time, inclusion of automation testing, avoiding errors, and providing faster feedback
- Lean - focusing on quality and efficiency based on Kaizen[1] philosophy
- Measurement - having a fact-based background for improving development, setting accessible objectives with performance indicators
- Sharing - having a collective mind, shared knowledge, tools and responsibility

Continuous Delivery also emphasizes the importance of these technical practices (combined with comprehensive configuration management) as an enabler of more frequent, higher-quality, and lower-risk software releases. Cited from book *Accelerate: The Science of Lean Software and DevOps* [12]. This looks quite similar to CALMS principles described above right ...

[1] In translation this means "improving" or "change for better". More can be found e.g., on websites of Wikipedia: https://en.wikipedia.org/wiki/Kaizen

But how to implement Continuous Delivery? According to *Accelerate: The Science of Lean Software and DevOps* [12], there are three base foundations to implement Continuous Delivery:

1. Comprehensive configuration management - Which means "possible to provision our environments and build, test, and deploy our software in a fully automated fashion purely from information stored in version control"
2. Continuous Integration - More about Continuous Integration later as a part of CI/CD
3. Continuous Testing - This should be done continuously, not only when a new feature is developed, this prevents errors and speeds up the development process. Continuous Testing is usually part of Continuous Integration

AWS websites [6] provide their opinion on best DevOps Practices. Which includes Continuous Integration, Continuous Delivery, Microservices, Infrastructure as Code[2], Monitoring and Logging, with Communication and Collaboration.

DevSecOps

DevSecOps (sometimes can be found term SecDevOps, since the basic idea is referred to "security from a start" [36]) is an extension of the DevOps term. It is used when a team tries the DevOps approach, when there is no time for the additional step of searching for vulnerable places.

Although well known in IT Management, the dedicated role of a security expert is new in a DevOps team. The DevSecOps expert now brings security controls and practices into an SDO life cycle. Cited from book *Continuous innovation with DevOps* [36].

Developers now can code with security in mind, so the solution is developed from scratch with a secure infrastructure, thanks to this the security flaws can be found earlier, which reduces the cost. Also, teams could agree on using the same version of OS, using the same IDE for coding, encrypting disks, etc., but this isn't enough. This is a shift from the traditional sequential development model, where the development times were long and updates came after weeks or months and for each solution, there was a special team for security. The team should find all the vulnerable places, but this additional step could take days or weeks.

A common approach is having several smaller services instead of one big service, that's why we can see everyone start using containers or microservices. Another approach is having additional steps in the CI pipeline where some security tests could be listed. Those two approaches will help separate the solution into several smaller, more secure parts that are easier to test and could be reused afterward. In the image below can be seen the percentual usage of containers for deployment and popularity of SaaS:

[2]Configuration files are created that contain infrastructure specifications, also ensures that the same environment is provisioned every time. Information has been gained from https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac

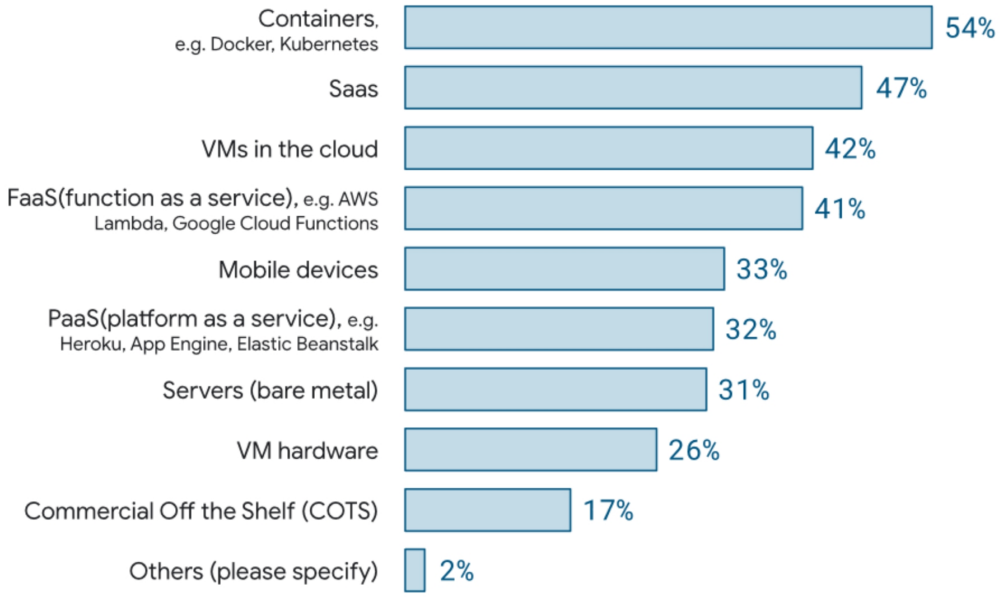


Figure 2.2: Deployment target, available in technical report *State of DevOps* from 2022 [20].

The terms DevOps and DevSecOps can sometimes be interchanged. That's because we usually mean the same thing (DevSecOps, although we substitute it for just DevOps) because no one wants insecure software.

A common technical practice in DevOps is the usage of CI/CD, which will be discussed in the upcoming section.

2.1 CI/CD

This acronym stands for continuous integration and continuous delivery. This approach uses several workflows that work as rails for building, testing, delivering, and deploying the software to the product environment. Those workflows are written declaratively, which includes information about cloud resources with dependencies and when and how the workflow should be triggered. This approach is widely used in the majority of software development processes.

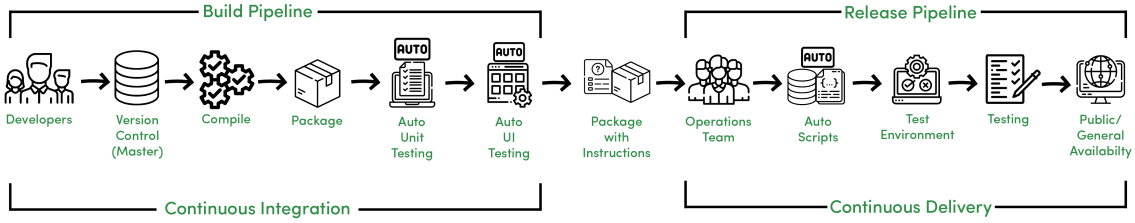


Figure 2.3: CI/CD. Image available at websites of GeeksForGeeks [24].

In an article about CI/CD from the year 2021, editor Katie Dee wrote: According to the "2021 State of Continuous Delivery Report" conducted by the Continuous Delivery Foundation, 74% of developers are currently utilizing some kind of DevOps practice. The report found that while 44% of developers use either continuous integration or deployment (CD/CD), less than 18% employ both practices to automate their delivery processes fully. The report also found that continuous integration is currently the most widely used DevOps-related activity with 32% of developers surveyed reporting that their organizations use CI in practice. Furthermore, only 29% of the respondents said that they are currently using continuous deployment as their main method of delivery automation. Cited from pages of SDTimes [31].

2.1.1 Continuous Integration

Continuous Integration, known as CI is a process where developers merge their code changes into a central repository with a Version Control System (VCS). This is usually a cloud service provider for version control using Git e.g. GitHub, GitLab, or Azure DevOps. The change is captured or webhooked by the prepared workflow that is triggered. This workflow contains steps that should be executed, which could involve installing some dependencies, building commands (that will create an artifact), and running tests (unit and integration). However, not all the steps have to be always executed, e.g. installing dependencies, building the app, and running unit tests are usually done after each commit to the branch.

2.1.2 Continuous Delivery

This step follows, if all steps from the previous (CI) step succeeded and the created artifact has passed the tests. Continuous Delivery, known under the acronym CD, is a step that is used for finishing the CI step, which could involve packaging the whole artifact and afterwards uploading the package to a non-production branch like GitHub or another artifacts registry. The package is now ready for deployment if the Continuous Delivery step has been completed successfully.

2.1.3 Continuous Deployment

When having a package uploaded to a non-production registry, there is only one step left for deploying the package to the product environment. This could be done manually by DevOps team members. Still, having multiple deployments of an application per day, sometimes even per hour, can get frustrating and very time-consuming. Another option is to use "another" or "second" CD that is usually connected with the first CD (Continuous Delivery). This step is called Continuous Deployment. Without human intervention, this step will speedup the deployment process, there will be no space for human error, and DevOps teams can have a better overview of the whole process.

In the end, Continuous Delivery and Continuous Deployment are not necessary, so it is usual for teams to use only Continuous Integration. On the other hand, the developer can't use Continuous Delivery and Continuous Deployment without Continuous Integration.

2.2 Jenkins

One of the most famous automatization tool for CI/CD is an open-source project written in Java named Jenkins (formerly Hudson). This tool connects the whole CI/CD process

together; since then, Jenkins was developed for the unique needs of developers. Thanks to this, it can be turned into a complete Continous Development Hub[3].

Additionaly, Jenkins is highly customizable thanks to extensions and plugins that can be installed from the Update Center, these plugins cover everything from version control systems, build tools, code quality metrics, build notifiers, integration with external, systems, UI customization, or even "Gamification". Another advantage is its web interface, which will help ensure the success of the given solution. Jenkins can distribute work to multiple machines with different architectures, so that the development teams can test their solution in all other architectures.

Jenkins is maintained, supported, and used by individuals and well-known organizations like AWS, Red Hat, GitHub, CloudBees, JFrog, and many others. So it's no surprise, that Jenkins's market share is, according to *Continuous Delivery Foundation* website[4] approximately 43% in the year 2023 and was even higher in the year 2010 (then still known as Hudson) with considerable market share of over 70%. Information has been gained from the book *Jenkins The Definitive Guide* [41] and websites of Jenkins[5]. But if Jenkins doesn't meet the requirements of DevOps teams, there are other options for automatization servers for CI/CD, for example, from public providers like GitHub, GitLab, and many others.

[3]Complete solution for handling both Continous Integration and Continous Development with Continous Deployment.

[4]https://cd.foundation/announcement/2023/08/29/jenkins-project-growth/

[5]https://www.jenkins.io/

Chapter 3

Cloud Service

This refers to a group of services/applications that are usually deployed in big data centers and are available to the user over the internet. For most companies, building their cloud infrastructure can be very expensive, and companies usually prefer using clouds from 3rd-end party cloud providers. The biggest of those are Amazon Web Services (AWS), Microsoft Azure, Google Cloud and IBM Cloud. Choosing to deploy a solution to a 3rd party provider data center has many advantages as well as many disadvantages.

One advantage is not needing to take care of infrastructure and paying only for those resources that are used and should be easy to scale up if needed. Also, when the development team decides to use any of the possible providers, all of them have a policy in which they promise that their infrastructure will be available at least 95% of the time, depending on a service[1].

One of the disadvantages of using a 3rd-party cloud provider infrastructure is a "risk" with security because the team has to trust the provider with their security. In addition, there is less control over the cloud and more dependence on someone else, which could also be problematic. The cloud provider could change the conditions, policies, and circumstances in which the provider provides the service. This could lead to additional problems for the company.

	Pros	Cons
On-premise	Security Control	Price Managing Backups
3rd party	Price Availability Scalability	Limited Control Trust to the provider

Table 3.1: On-premise compared to 3rd party cloud provider

3.1 Cloud computing models

In the domain of cloud computing, the concept of service - basically providing for someone else's needs by another party - is taking center stage. A new relationship called "as a

[1]Microsoft Azure Service Level Agreements (SLA) for Online Services: https://www.microsoft.com/licensing/docs/view/Service-Level-Agreements-SLA-for-Online-Services

service" has emerged. Indeed an acronym, XaaS, has been coined to stand for "anything" or "everything as a service" when offered through a cloud. Services subsumed under Xaas include: Software as a Service (SaaS), Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Storage as a Service (also SaaS), Communications as a Service (CaaS), Network as a Service (NaaS), Monitoring as a Service (MaaS), and likely many more to come. Underlying this alphabetic potage is a dramatic change in the accessibility, agility, and cost to performance ratio of information processing. In effect, the service orientation of cloud computing foretells a whole new way of doing business. Cited from book *Cloud Computing Service and Deployment Models* [9].

3.1.1 Infrastructure as a Service (IaaS)

Infrastructure as a service (IaaS) is the on-demand availability of highly scalable computing resources as services over the internet. It eliminates the need for enterprises to procure, configure, or manage infrastructure themselves, and they only pay for what they use. Cited from Google Cloud website [16].

This type of service is mostly used for web applications, test and development environments, data storage and backup/disaster recovery, machine learning, and many others. Typical examples of this service are AWS, Google Cloud, or Microsoft Azure.

3.1.2 Platform as a Service (PaaS)

Developers and especially programmers use this service, as it allows them to deploy, manage, and run their solutions with ease. This is due to the fact that there is no need to maintain the hardware, software updates, operating system, and many more layers as was necessary in the IaaS model. On the other hand, it is harder to find the right provider for the particular use case, as some providers may not have all the necessary runtime environments for a given software in their PaaS. Information has been gained from websites of Red Hat [39].

RedHat OpenShift Container Platform is one of the examples of PaaS alongside with Heruku, Google App Engine, Microsoft Azure App Service and AWS Elastic Beanstalk.

3.1.3 Software as a Service (SaaS)

SaaS is simply software delivered from a server in a remote location to a user's desktop and used online. Typically, it is operated via a browser interface, although some SaaS software uses an interface designed to provide unique characteristics for its application or its users. Initially, SaaS software existed only in real-time, within an online connection; when the connection was interrupted, the user had no data and no application. Cited from a book *Succeeding at SaaS* [43].

Since SaaS is an online version-only model, it is possible to have mobile or desktop versions of their web-based counterparts. SaaS takes care of synchronization with all the devices that are connected to it. Thanks to this feature, all data has an online version, and when working with SaaS, everything is up to date, no matter from which place the user joins.

When the user uses services like Gmail, Slack, Workday, Zoom, Meet, and thousands of others, he is using SaaS.

Google Workspace

In 2019, the COVID-19 epidemic started, which caused many problems worldwide, not only because of the risk of getting infected but also because of the impact on social life, productivity, and freedom. In addition, there was a global shortage of electronics, food, work, and many others. As a reaction to this global pandemic, many companies moved their employees to remote work. Thanks to those changes, companies like Google have started seeing increased demand for cloud applications that allow users to work from home. This was done by tweaking their current solution (G Suite) for productivity, and integrating it with other apps in the ecosystem. All those changes should help users have a seamless transition from offices to their homes. The new solution for this demand was called Google Workspace. Google has integrated, highly used tools for communication such as the Mail, Chat, Calendar, or Meet. Before this, associates and the public had to use tools from different companies. For example, BlueJeans or Microsoft Zoom for meetings, Zimbra for emails, IRC, and Slack for chats. When Google Workspace came to light, companies could use this complete solution to enable a hybrid/remote work culture. More in the book *Google Workspace User Guide* [28].

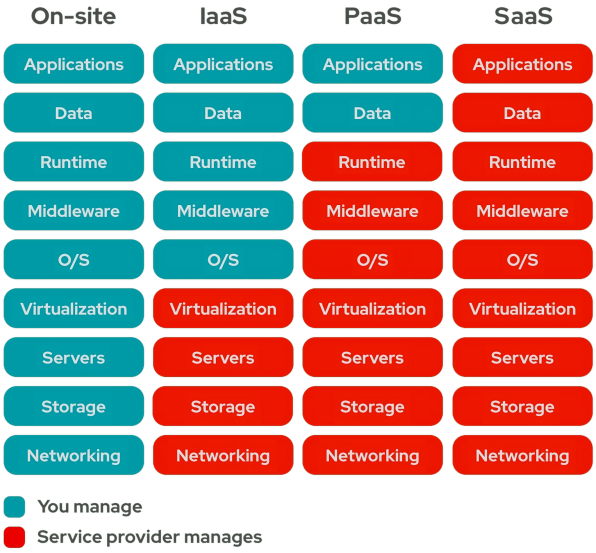


Figure 3.1: Comparison of IaaS, PaaS, and SaaS. Image available at websites of RedHat [37].

Chapter 4

Chatbots

Intelligent assistants, chatbots, and voice-enabled devices such as Amazon Alexa and Google Home, offer a new, natural, and intuitive human-machine interaction and open up a whole new world for us as humans. Chatbots and voicebots ease, speed up, and improve daily tasks. They increase our efficiency and, compared to humans, they are also very cost-effective for businesses employing them. Cited from book *Voicebot and Chatbot Design* [8].

Chatbot is an application that is usually available online and should mimic a human being. Using a chatbot or voicebot can speed up almost every process. Usually, humans can communicate with these bots using voice or text commands. For example, in these days, there are more and more "support bots" which will be the first to respond to the user when having a problem. And if the support bot won't be able to help the user, then the user will be redirected to chat with a human who will help him. Those bots could be also used for sorting customers according to their problems.

Since one of the most important things about building a chatbot is detecting intent and remembering the context of the messages, those things will be discussed in the upcoming two sections. After this, one of "big players" in the field of AI with their chatbot solution will be introduced. This chatbot solution is described to date of 4.12.2023.

4.1 Conversation UI

It is the newest form of communication between humans and computers that has evolved from the text user interface (which is closer to Conversation UI) and graphical user interface (GUI).

In a book *Voicebot and Chatbot Design* [8], Rachel Batish made a nice demo showing what a text user interface is and how it differs from Conversation UI. In the example, she tried to search information about hotels in New York in the browser using Bing and Google search engines. The output of this is not surprising, the search engines have come up with different hotels in New York City, but now it's up to the user to decide what she wants. On the other hand, when a user chats with a chatbot, this conversation is two-way. The human wants an "one and only" answer from the chatbot. The bot could ask clarifying questions to supply the best answer possible if needed. In this context those could be questions like "How many star hotels would you like to visit?" or "In a center or on the edge of New York?", this question is counted in responses to previous questions. This information about a past conversation is called a context.

4.2 Natural Language Processing

"Natural Language Processing is the set of methods for making the human language accessible to computers" cited from book *Introduction to natural language processing* [29]. In other words, NLP is giving computers the ability to understand written and spoken words in much the same way human beings can. It also creates a better environment for communicating between humans and machines. This makes the dialog experience more humanized for humans.

Eda Kavlakoglu wrote a blog post for IBM [11] where he wrote: "Natural language processing, which evolved from computational linguistics, uses methods from various disciplines, such as computer science, artificial intelligence, linguistics, and data science, to enable computers to understand human language in both written and verbal forms", this is due to the nature of language that contains words that could be metaphors, homonyms, homographs, homophones, sarcasm, irony, and usage exceptions. All of those could lead to misunderstanding even for many adult humans, and we want to teach machines to this... Although creation of NLP is a task that takes much time, the benefits of this are huge and at this time, we couldn't live without it, because it is used by everyone multiple times per day. For example, tasks like speech recognition, assigning tags or categories, autocorrect, spam filters, text summarization, named entity recognition and many more can be listed as NLP tasks.

Natural Language Understanding (NLU) with Natural Language Generation (NLG) are both subsets of NLP. While NLU uses semantic and syntactic analysis of the text to understand the intent and meaning behind the user's words. For example, this sentence uses the same word "duck", but in each context, has totally different meanings, and the NLU has to find out which:

- You have to duck otherwise, the duck will hit you in the head.

Without the NLU would could be the machine determine the wrong meaning of the sentence, which could lead to nonsense. On the other hand, NLG is focused on generating meaningful sentences for us humans from a given dataset using hidden Markov chains, recurrent neural networks, or transformers. If this is not accomplished, the generated sentence (nor the text or speech) could lack of meaning and a reader or listener can have a problem with understanding it. Information has been gained from the blog post of IBM [11].

4.3 Transformer neural network

On 12 Jun 2017 Google's research team released a paper called *Attention Is All You Need* [42] in which a **Transformer** has been introduced, which became the foundation for all the upcoming Large Language Models (LLMs). This Transformer replaced previous models like convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Information has been gained from websites of Google Research [30].

Compared to CNNs and RNNs, the transformer is able to process the tokens faster with better results and in addition with reduced training time, this is thanks to a his encoder-decoder model. This model is able to train/process tokens in parallel, unlike the RNNs that can only process the tokens in sequential order. Another advantage of parallel token processing is better handling of long sentences and remembering the context thanks to the attention mechanism.

4.3.1 Attention mechanism

This mechanism works by paying special attention to specific tokens at each step. This can be done by using the hidden states created in the encoding phase and using them to create/decode output tokens. At first, it was used in RNNs for improving processing, but this mechanism didn't improve one of the biggest problems of the RNNs (training and processing time, forgetting the essential information learned previously). This was solved by transformer networks that upgraded this attention mechanism and is known as **self-attention**.

4.3.2 Model Architecture

The transformer model introduced in *Attention Is All You Need* [42] is created from 2 parts, an encoder and a decoder, where the encoder's output is used as input for the decoder. The encoder and decoder are present in the model six times.

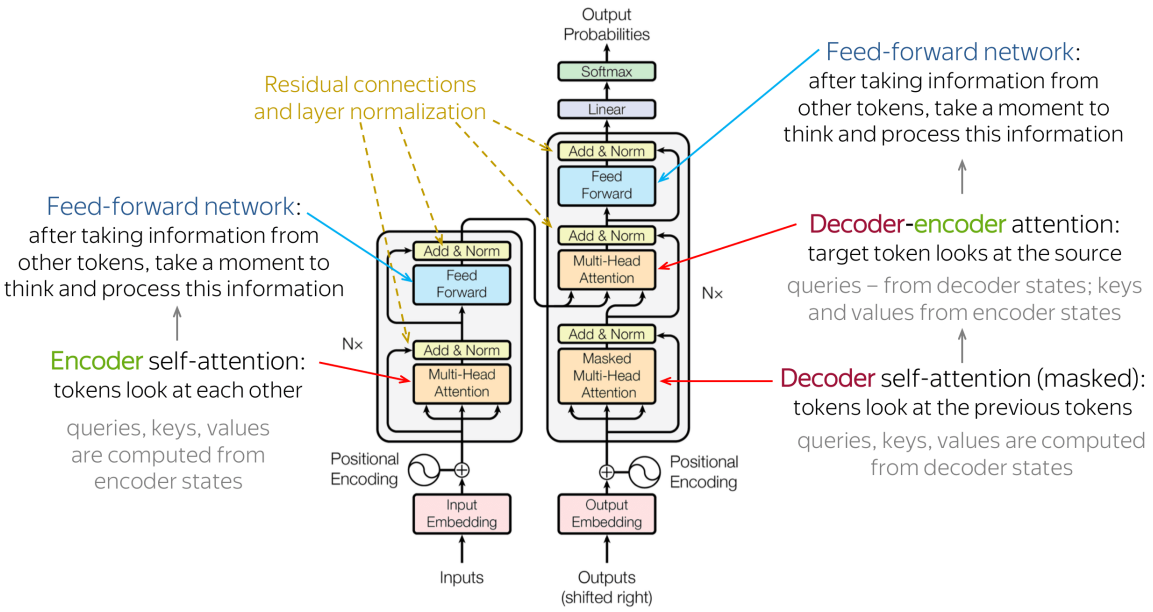


Figure 4.1: The Transformer - model architecture. Image has been gained from blog post *Sequence to Sequence (seq2seq) and Attention* that was written by Lena Voita [32].

The LLMs can have one of three types:

- "Encoder only" - means full Encoder and full Decoder (not auto-regressive ones). This type of model is used in BERT or ELECTRA. Meanwhile, after the initial explosive growth brought about by BERT, "Encoder only" models gradually began to fade away.
- "Decoder only" - means auto-regressive Encoder and auto-regressive Decoder. This is the dominating type of LLM at the time of writing this thesis, which is used in LLMs like ChatGPT, Bard, LLaMA, PaLM, and many others.
- "Encoder-Decoder" - means Encoder with auto-regressive Decoder. Those models remain promising, as this type of architecture is still being actively explored.

Cited from paper *Harnessing the Power of LLMs in Practice* [44].

Encoder

The encoding component (on the left) can be broken down into two sub-layers:

1. Self-attention layer - is taking care of keeping relations between the words. Take for example, this sentence:

   - The ceramic fell on the ground and it shattered all over the place.

   The word "it" can be associated with the word "ceramic" or with the word "ground". But the correct association would be the word "ceramic". The self-attention mechanism takes care of correctly associating one word with another. This works by seeing how similar the words are to each other in a sentence, even to itself. Then, the similarities are used to determine how each word will be encoded.

2. Feedforward layer - In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between. Cited from *Attention Is All You Need* [42].

Decoder

The decoding component (on the right) has both the self-attention and feedforward layer (introduced in a section above), but in between them is the encoder-decoder attention layer. This layer performs multi-head[1] attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent position. Cited from *Attention Is All You Need* [42].

A more detailed explanation of transformer model architecture can be found in paper *Attention Is All You Need* [42] or the blog post *The Annotated Transformer* [2] which can also be found the source code for the Transformer.

4.3.3 Google

One of the biggest player in the "game of AI" is Google with their Bard and Chat App solutions (more about Google Chat Apps later). Unlike Microsoft Bing, Google Bard uses LLM from Google called Pathways Language Model (PaLM 2), which is also a Transformer-based model, like ChatGPT trained with the Pathways system. Similarly to OpenAI's model, PaLM 2 was also learned on publicly available web documents, books, code, mathematics, and conversational data with source code datasets. This makes it a state-of-the-art[2] language model with improved multilingual, reasoning and coding capabilities. To make this model fit many use cases as well as possible, PaLM 2 comes in 4 variants:

- Gecko - A smallest and most lightweight variant of PaLM 2 model. Can be used even when offline and is mostly used on mobile devices with other resource-constrained platforms.

[1] More information can be found in the original paper

[2] Best-performing methods and cutting-edge techniques in a particular field

- Otter - Is a medium-sized variant that has the best balance between size and performance. Thanks to this it could be used for a wide range of tasks, on a variety of hardware platforms (mobile platform included), and is used as an assistant for helping users write better emails.
- Bison - This is so far the largest publicly available variant, it is used to perform a wide range of Natural Language Processing tasks with state-of-the-art accuracy. Thanks to this, it could be used as a chatbot to support in Google Cloud customers. Also, the Google Bard itself uses this variant of the PaLM 2 model.
- Unicorn - The last and biggest variant with astonishing 540B parameters, that is currently not available to the public that should be used for research and development. But generally, it should be an advanced version of Bison.

Google has made the PaLM 2 model specialized for several advanced research directions, by fine-tuning PaLM 2 model for:

- Cybersecurity (Sec-PaLM) that is now available through Google Cloud, it uses AI to help analyze and explain the behavior of potentially malicious scripts, and better detect which scripts are actually threats to people and organizations. Cited from Google blog [45].
- Healthcare (Med-PaLM 2) as a result, Med-PaLM 2 was the first LLM to perform at an "expert" test-taker level performance on the MedQA dataset of US Medical Licensing Examination (USMLE)-style questions, reaching 85%+ accuracy. Cited from Google blog [1].
- Google Workspace (Duet AI) that helps the user speed up his work, accelerate workflows, inspire, and more.

More detailed information about the PaLM 2 model can be found in its Technical Report [15].

4.4 Chatbot frameworks and services

This section will introduce frameworks and services from different companies for creating chatbots. Those services will be compared to Google Dialogflow service, described in more detail in the upcoming chapter (Architecture).

4.4.1 IBM Watsonx

It is a platform from IBM that allows users to create, manage, and build custom AI applications. The watsonx AI platform comprises many services, like watsonx Assistant, that enable the creation of chatbots without coding. Watsonx.ai is used for training, validating, tuning, and deploying models for generative AI models from foundation models from one of the open models, like from IBM itself (granite), Meta (Llama), Google (Flan-UL2 and FLAN-T5) and models from EleutherAI, BigScience, BigCode, or Mosaic, with fine-tuning their parameters. Another service watsonx.data uses AI to process the data to find hidden trends in data and watsonx.governance, which serves as an "eagle eye" for other watsonx services. Information has been gained from websites of IBM [25], [26] and [27].

Compared to Google Dialogflow, the experience was similar, but the creation of the conditions for asking for all the parameters was quite complicated compared to Dialogflow. On the other hand, watsonx offers more options for handling conditions with parameters for transitions to the following steps and a large spectrum of pre-defined response types, like date, number, confirmation, regex, currency, and others. Thanks to this, the user can select, e.g., the server type by clicking one of three options and confirm the task by clicking on thumb up/down, which is neat and comfortable for the user.

4.4.2 Microsoft Bot Framework platform and Azure AI Bot Service

Microsoft Bot Framework and Azure AI Bot Service provide tools to build, test, deploy, and manage intelligent bots. The Bot Framework includes a modular and extensible SDK for building bots (in Java, C#, Python, or JavaScript), including tools, templates, and related AI services. With this framework, developers can create bots that use speech, understand natural language, handle questions and answers, and more. Cited from websites of Microsoft [10].

Thanks to Azure AI service, the chatbot can be further improved by adding advanced capabilities on top of the basic functionality of the Bot Framework. Such as generating human-like text responses thanks to access to AI models such as GPT-3 from OpenAI or using speech-to-text and text-to-speech conversations, which would not have been possible without Azure AI.

The Bot Framework Composer is similar to other solutions for developing chatbots; it has a simple UI, so creating a given intent was straightforward, without hiccups. But the framework had the most "coding" from all of the services and is the only one with a dedicated desktop app instead of an online version, like other services. The Composer also allows switching the UI editor to "coding editor" with JSON in which the chatbot can also be written. Deploying the chatbot is, compared to other services, also more complex than it could be (mainly due to the desktop app). On the other hand, Azure allows a broad spectrum of integration with Microsoft apps and other chatting platforms.

4.4.3 Meta Wit.ai

The Wit.ai is the app that parses the message and creates the entities from a person's message. You can use the default Wit.ai for supported languages or create your own Wit.ai app for each language you want to support. When Messenger Platform receives a message, it will first find the top detected language, and use the associated Wit.ai app for extracting the entities. Cited from websites of Meta Developers [33].

This service can be used to create agents (without coding), which can be used in chatbots, make multimodal interactions, or control smart homes with their voices. Wit.ai allows the detection of intents and entities from messages. This is similar to the Google Dialogflow service, but compared to the Dialogflow, Wit.ai offers pre-defined entities with "guidelines" and units like temperature, money, and many more. The Wit.ai in its composer (similar to Dialogflow) can create almost the same model as in Dialogflow. But the equal model would be significantly larger since, when is a user asked for the required parameters, the model must have three blocks (if statement, response, and input) for each parameter. Creating a default value would take six of those blocks. Due to this, one task will have approximately the same number of blocks as Dialogflow (that has 14 building blocks for the whole model, including the start and end), and the default value can be set by checking one or two checkboxes.

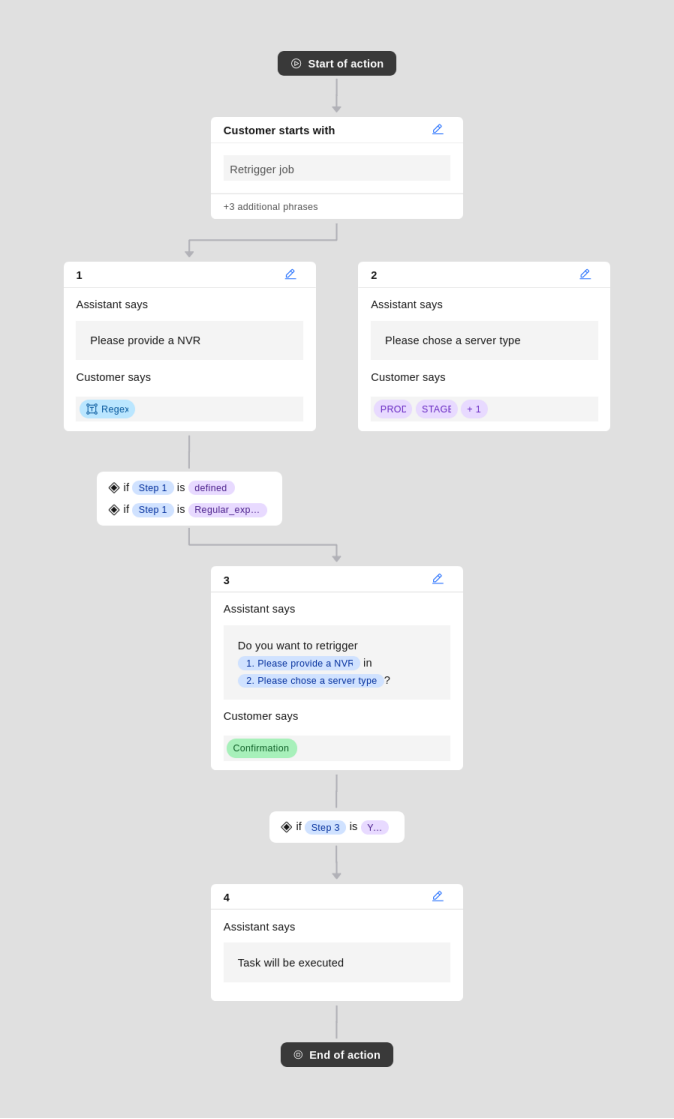


Figure 4.2: Re-trigger job example in watsonx Assistant.

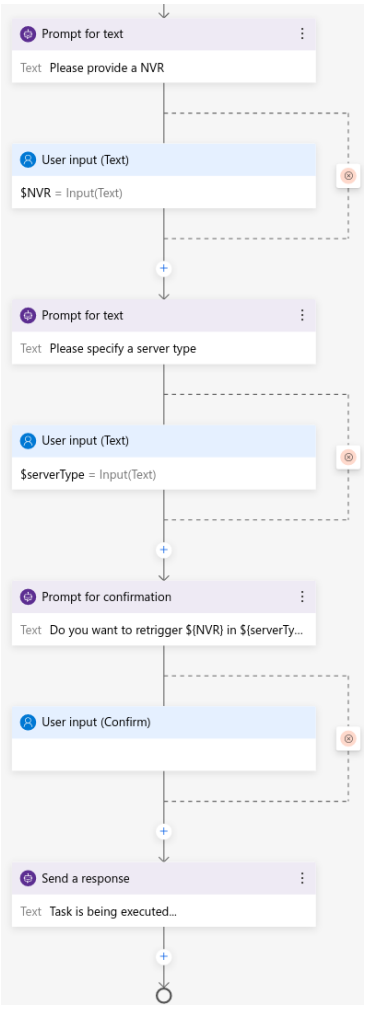


Figure 4.3: Re-trigger job example in Bot Framework Composer

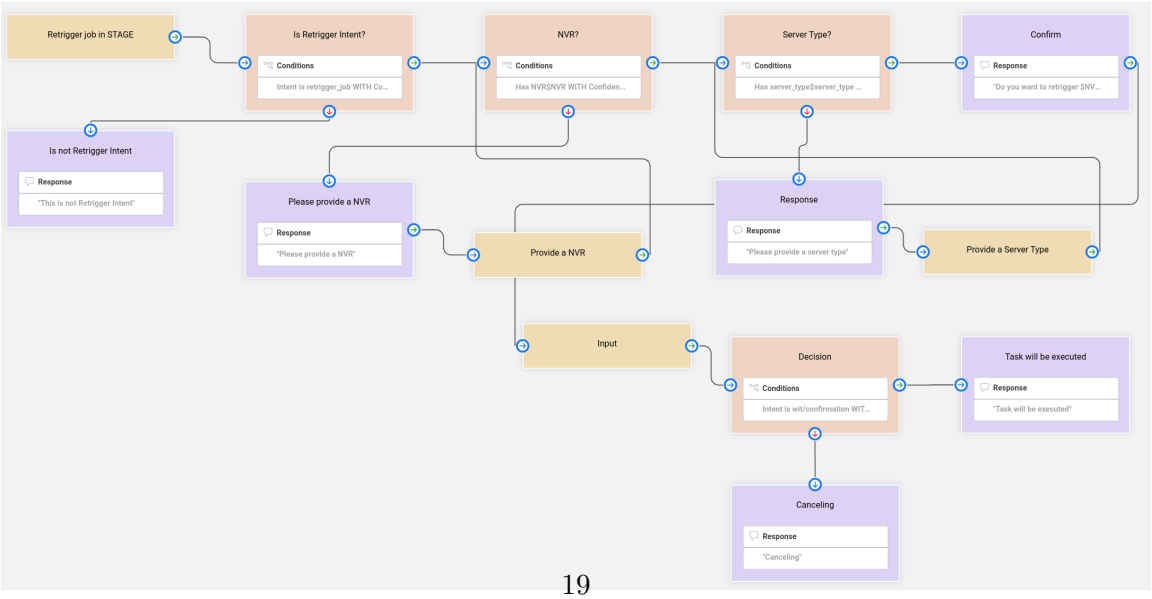


Figure 4.4: Re-trigger job example in Wit.ai.

4.4.4 Amazon Lex

Amazon Lex is an AWS service for building conversational interfaces into applications using voice and text. With Lex, the same deep learning engine that powers Amazon Alexa is now available to any developer, enabling you to build sophisticated, natural language chatbots into your new and existing applications. Amazon Lex provides the deep functionality and flexibility of natural language understanding (NLU) and automatic speech recognition (ASR). Cited from websites of AWS [7].

Lex was most similar to Dialogflow from all the other services, but offers almost every option possible, even "How long it should wait for the user to pass the required parameter for each retry attempt" and easy integration with Amazon Lambda[3]. The Visual Builder allows easy creation of a required flow of given intent and configuration without connecting the parts, which was relatively easy. On the other hand, the Editor was so overwhelming with settings, and for someone starting with building chatbots it could be pretty time-consuming. Compared to all other chatbot services, Lex offers inbuilt integration only to Facebook, Slack, and Twilio SMS, which is not much compared to other services.

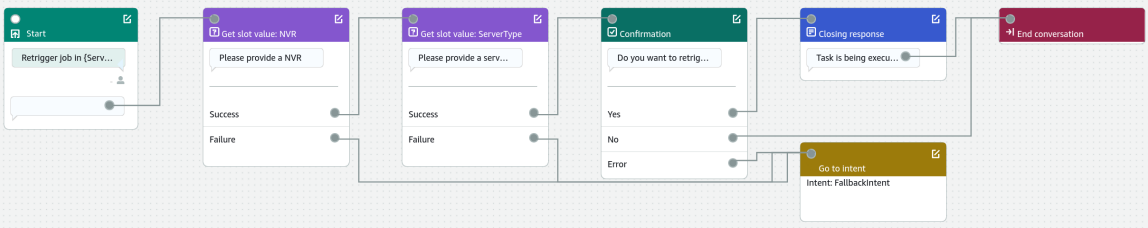


Figure 4.5: Re-trigger job example in Lex.

[3]Service that runs a code in response to events

Chapter 5

Architecture

Since the chatbot will be available on Google Chat, it was pretty easy to pick the right services for developing this chatbot, which is Google's. This decision has many advantages, such as good infrastructure for developing this chatbot and service called Google Dialogflow, and technologies used for building the chatbot (Flask and SQS messaging).

5.1 Building a Google Chat App

When building the Google Chat App, the developer will be faced with multiple choices, which are good to consider even before the development starts. He needs to make some choices, like which architecture is best for his usage if the app will be available to the public, in which spaces and how to choose the correct rights for users and many more. He should also not forget about designing UX/UI. That is probably one of the most important things to consider when creating a Google Chat App. If the UX is not user-friendly or working with the app is challenging, it would go against the purpose of creating a chatbot app (speedup workflow, integrate services, no need to go to a different page/service).

5.1.1 Architectures of Google Chat App

Choosing the right architecture is among the biggest choices when building any app. This isn't an exception when building a Google Chat app.

Web Services

This approach is mostly common, it allows to create an on-premise app that has one endpoint and uses REST API[1] for communication with Google Chat API using HTTP's methods[2]. The significant advantage of this approach is the freedom of choosing any programming language with any framework the developer likes. This approach is mostly common when creating a Google Chat app and any Web application.

Cloud Pub/Sub

Similarly to Web Services, this architecture also uses HTTP operations for communication with the app. But the difference is when the app is deployed behind a firewall (e.g. having

[1] The requests to the endpoint are delivered in several formats (JSON, HTML, XML or plain text)
[2] GET, POST, DELETE, PUT and PATCH

the app running on a cloud from AWS, IBM Cloud, or Microsoft Azure), this could be a problem because Google Chat may not be able to make requests to the app. This architecture can solve this problem; the shortcut for Pub/Sub stands for Publish and Subscribe, where Publishers send messages asynchronously to Pub/Sub service. From there, messages are broadcasted to all services (Subscribers) that have sent a subscribe message to Pub/Sub Service. After that, it is up to the service to decide what to do with the message. This is similar to an observer design pattern. More about design patterns can be found in the book *Learning Python Design Patterns* [14].

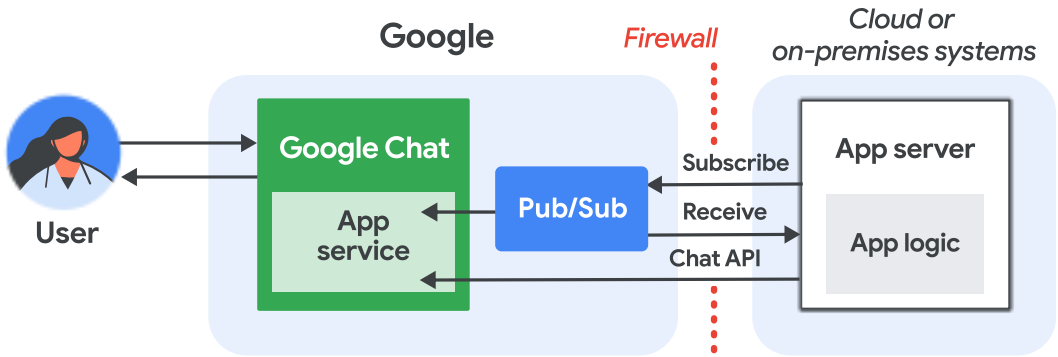


Figure 5.1: Pub/Sub model. Image has been gained from websites of Google [22]

Google App Script

Another type of architecture is Google App Script. This architecture is the easiest to create and the fastest to develop of all the available architecture options. It has great integration with the entire Google Workspace. Thanks to this, it is possible to access/create/update files in most Google Workspace services. When creating Google App Script, there is no need to install any dependencies. Everything needed is provided by Google on their cloud solution, which will be deployed afterwards. Google App Script application can be written in JavaScript with additional options for using the app from Google Workspace (special .gs files).

Google Dialogflow

Google Dialogflow is an architecture/service for building chatbots, which is probably the most interesting for most people. It uses both text user interface with the Conversation UI, in other words, it means the user can interact with the app using triggering commands (like the famous “Hey Google”, or “Hey Siri” from Apple). From this app the requests are parsed and sent to the Dialogflow endpoint, that is the Natural Language Processing service which uses NLU. From there, the response is sent back to the app, where the response is parsed to the required format and sent back to the Chat.

This could be done not only with Google Dialogflow; companies like Google[3], OpenAI[4], and Microsoft, among others, provide a service that can send requests directly to their chatbot solutions.

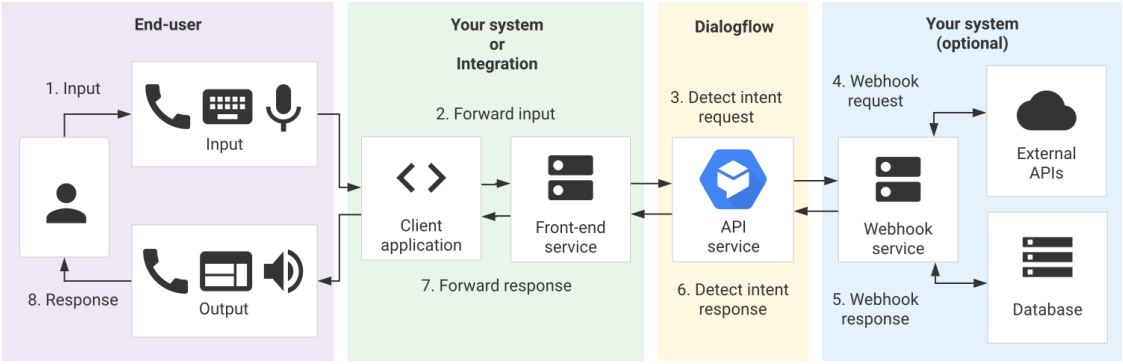


Figure 5.2: Dialogflow CX flow diagram gained from websites of Google [17].

Google is providing two Dialogflow services, ES and CX. Dialogflow CX is "taking a state machine approach to agent design. This gives you clear and explicit control over a conversation, a better end-user experience, and a better development workflow", cited from websites of Google [18].

This service is also more advanced and newer than Dialogflow ES, providing a better foundation for creating advanced ML models. The Dialogflow CX agent is composed of 4 main components:

- Pages - These are used as states and contain the logic. When entering/exiting the page, it is possible to create a fulfillment/answer.
- Flows - These are advanced mechanics in Dialogflow CX and are used for "hiding" conversation topics.
- Routes - Are used for transitions between pages (states), given transaction can be executed by fulfilling one of the conditions or detecting intent.
- Intents - A list of possible options in sentences. Each intent contains multiple examples of given intent with selected entities in a sentence.
- Entities - Are "variables" that are detected in sentences with given intent, entities can also be used as conditions for transactions between pages.

Dialogflow CX uses generative AI (at the time of writing this thesis there is an option for choosing one of two models, PaLM 2 variant Bison and Gemini Pro) to process a text message, voice, or call to determine the intent of a given input[5]. When an intent is detected in a sentence, the state machine will use this intent to determine the next page with a given route bound to the detected intent. The user can also be asked to provide some information (entities). Without that information, moving to another page would not be possible. In the end, DialogFlow CX can, e.g., send an SMS message or trigger a webhook.

[3] https://ai.google.dev/gemini-api/docs
[4] https://platform.openai.com/docs/api-reference/introduction
[5] Quotas and limits of Dialogflow agent https://cloud.google.com/dialogflow/quotas#cx-agent_3

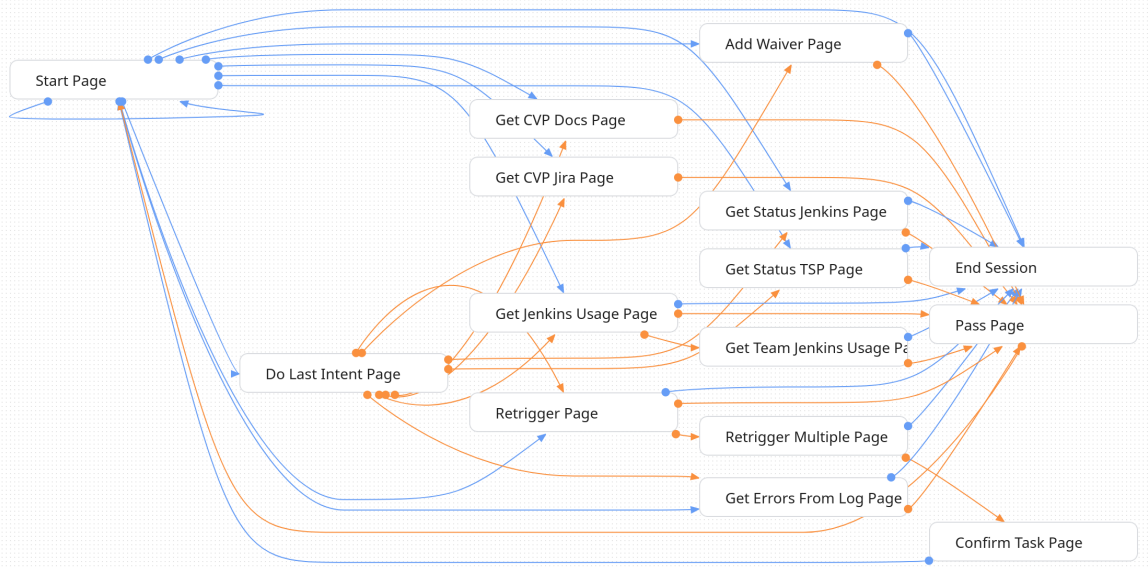


Figure 5.3: Dialogflow CX agent of the CVP chatbot.

5.1.2 Google Chat messages

Since the Google Chat API uses REST API for communication, the format of messages is usually JSON sent via HTTP's POST method. Two types of messages are available in Google Chat. First is the classic text message that allows communication with other users, mentioning them or adding bots to the given chat (in space/room or Direct Message). The other type of message is a card message. This type of message shows the app data in GUI, which is more intuitive for the human user. Since all messages are in JSON format, the text message is as simple as it can be.

```
{
   "text":"Hello World!"
}
```

Listing 5.1: Example of a simple text message sent to Google Chat.

More interesting is creating the card messages. Each card has to begin with the card message version and unique cardID. After that, some sections alongside headers and other elements like buttons, images, and more, can be added to the card.

```
{
   "cardsV2":[
      {
         "cardId":"42",
         "card":{
            "header":{
               "title":"CVP chatbot",
               "subtitle":"Helps speedup users their day-to-day tasks",
               "imageUrl":"https://imageurl.com",
               "imageType":"CIRCLE"
```

```
        },
        "sections":[
           {
              "header":"General Informations",
              "widgets":[
                 {
                    "textParagraph":{
                       "text":"The Dialogflow service will ask you..."
                    }
                 ...
```

Listing 5.2: Example of a simple card message sent to Google Chat.

More information about building a card message can be found on the websites of Google Developers[6] with their card editor[7], which can come in handy when building card messages.

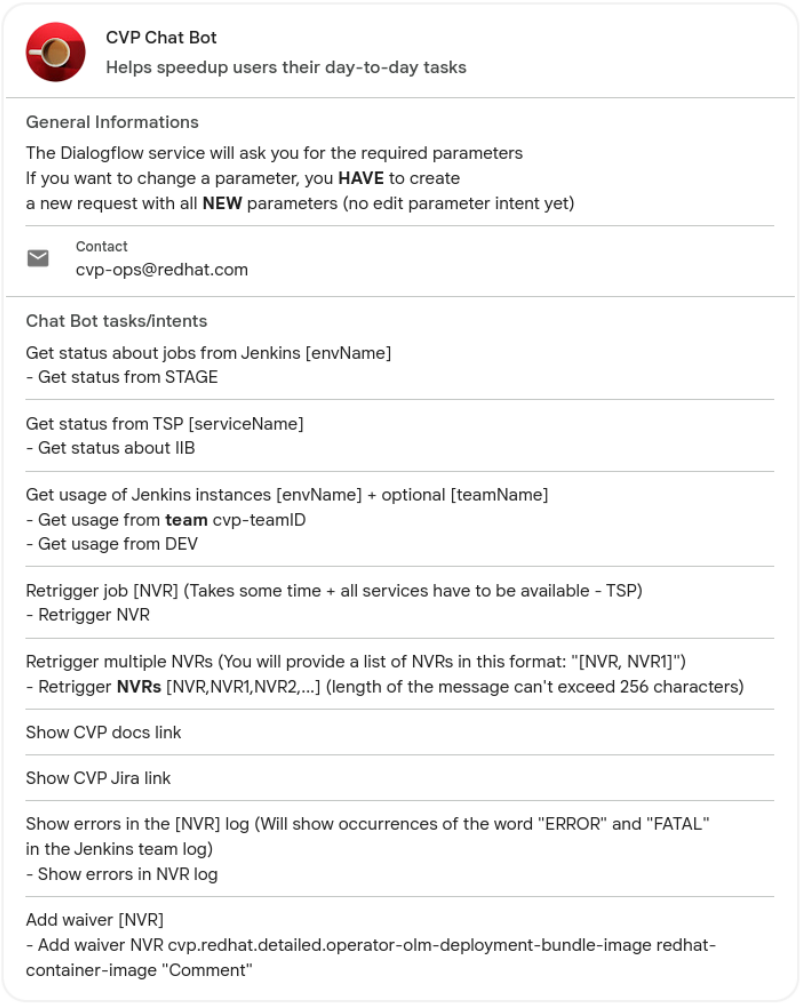


Figure 5.4: Example of card message sent from the app.

[6] https://developers.google.com/chat/api/reference/rest/v1/cards
[7] https://addons.gsuite.google.com/uikit/builder

Events in messages

In messages sent by Chat API, there can be four event types: a `ADDED_TO_SPACE`, `REMOVED_FROM_SPACE`, `MESSAGE` and `CARD_CLICKED` events. Event `ADDED_TO_SPACE` is set when the bot is added to a group chat or when a Direct Message with this chatbot is being opened. `REMOVED_FROM_SPACE` is unique because when the bot is removed from the space, it is already too late for the app to send any message to the chat because the bot is already removed from it. So this kind of message event is only for information purposes. When communicating with a bot, the most common type of event is a `MESSAGE` event sent to the app when a bot is called (e.g., using his slash command or mentioning the bot in a chat). The last option is `CARD_CLICKED` events, which are evoked when the user clicks, e.g., on the card button.

```
"type":"MESSAGE",
"eventTime":"2023-10-22T16:43:44.992879Z",
"message":{
  "name":"spaces/ABCDE/messages/ABCDE.ABCDE",
  "sender":{
     "displayName":"Evzen Gasta",
     "email":"egasta@company.com",
     "type":"HUMAN"
  },
  "text":"@CVPbot retrigger job",
  "annotations":[
     {
        "type":"USER_MENTION",
        "userMention":{
           "user":{
              "name":"users/123456789",
              "displayName":"CVPbot",
              "avatarUrl":"https://imageurl.com",
              "type":"BOT"
           }
        }
     }
  ],
  "thread":{
     "name":"spaces/AAAAGqeQb1E/threads/gry6h9C8irg"
  },
  "space":{
     "name":"spaces/AAAAGqeQb1E",
     "type":"ROOM",
     "displayName":"Test environment"
  },
  "argumentText":" retrigger job"
}
```

Listing 5.3: Shorten example message sent by Google Chat API.

In the example above, we can see a message sent by Google Chat API to the app. The first item in the message will be the `type` that indicates the type of the event. In this case, we can see that this event is `MESSAGE`, so only text is shown. Another important item in this event is a `message` item, which contains the most important data for the application, like information about the sender, the text of the message sent by the user with information about the space with thread or direct message from which the message was sent. This information (the name item) is crucial in the application since it makes it possible to address the correct thread from which the message was sent. In this thesis, those thread name items are used to create the unique ID of a session in Google Dialogflow, which means that many users can communicate in one session simultaneously instead of having multiple sessions in a space for each user. However, if the user wants to use the chatbot privately, he can always use direct messages instead of space messages.

5.2 Jenkins in CVP team

To understand how Jenkins is used in Red Hat, it is necessary to clarify a few more products and technologies that allow the use of Jenkins as it is used now. OpenShift Container Platform, Datagrepper, Waiver, and **Red Hat AMQ** (inside the company Red Hat known as a "UMB") which is an advanced version of Apache ActiveMQ adapted for needs of company Red Hat.

5.2.1 Apache ActiveMQ

Apache ActiveMQ is the most popular open-source, multi-protocol, Java-based message broker. It supports industry-standard protocols so users get the benefits of client choices across a broad range of languages and platforms. Cited from websites of Apache ActiveMQ [3].

ActiveMQ supports protocols like AMQP, MQTT, STOMP, and OpenWire and can be used in Pub/Sub or Request/Response variations. Apache provides a library for work with AMQP protocol called **Apache Qpid Proton**, which allows connections from clients written in JavaScript, C, C++, Python, .Net, and more.

AMQP 1.0 potocol

AMQP (Advanced Message Queuing Protocol) 1.0 is an efficient, reliable, wire-level messaging protocol that you can use to build robust, cross-platform messaging applications. The protocol has a simple goal: to define the mechanics of the secure, reliable, and efficient transfer of messages between two parties. Cited from websites of Microsoft [34].

This protocol also introduces the concept of virtual topics that enable more dynamic routing of messages with TLS for encryption and SASL authentication protocol. Another big advantage is that AMQP 1.0 is a binary protocol and is approved by international standard ISO and IEC as ISO/IEC 19464:2014[8].

[8] https://www.iso.org/standard/64955.html

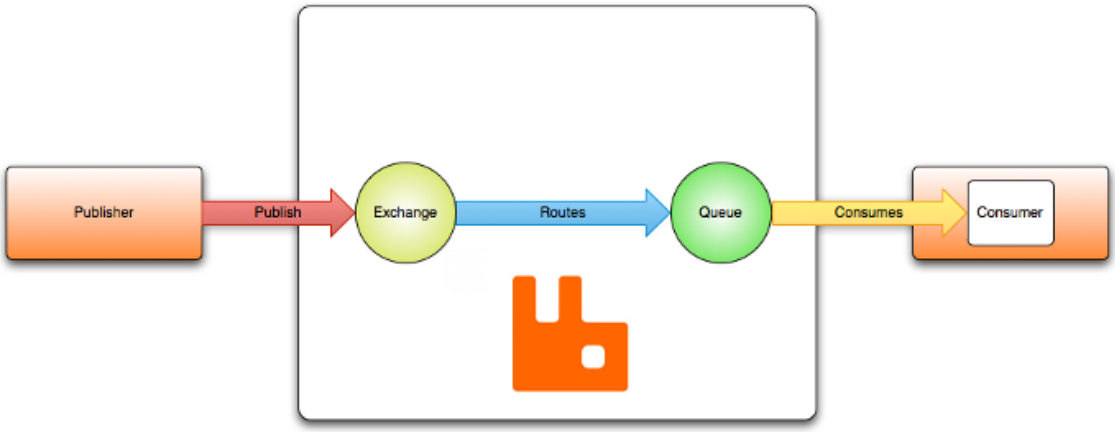


Figure 5.5: Example of AMQP 0-9-1 Model (Same as AMQP 1.0). Image has been gained from the website of RabbitMQ [35].

The model is driven by three main parts: Publisher, AMQP Brooker, and Consumer. The Publisher (producer) sends messages with one of four exchange types (Direct, Fanout, Topic, and Headers exchange) and a topic to an Exchange. AMQP Brooker delivers messages from Exchange to unique queues based on defined bindings registered in AMQP Brooker. The consumer, also known as the subscriber, fetches/pulls messages from a unique queue on demand, or the messages are sent to them by AMQP Brooker when a consumer is subscribed to a queue. The consumer also creates a binding that has to be registered in an AMQP Brooker, which specifies a route from Exchange to a unique queue. The AMQP Model can be compared to the post, where the producer is the sender, the Exchange is the mailbox or post office, the AMQP Brooker is the post, and the bindings are delivery addresses and the queues are mailboxes on the homes of the addresses (consumers). The other option is to consume the messages directly from a topic (without the queue).

Due to usage inside the Red Hat infrastructure, the Apache ActiveMQ (Red Hat AMQ) has been chosen.

5.2.2 CVP's Workflow

A development team makes a commit/PR to a VCS, and from there the code is built in the "BREW" environment. When the build finishes successfully, the Apache ActiveMQ message is sent with a topic "BREW build complete" and is subscribed by JMS Messaging Plugin[9] that is added to the "Jenkins Master". From there, messages that are not types of operator bundles or containers, are filtered out. When the message passes a filter "Jenkins Master" will forward the message to one of its two Jenkins workers (running on OpenShift), with the use of a Round-robin scheduling mechanism, where a job is triggered depending on the type of the build (operator bundle or container). Each job will create a new OpenShift worker node in a namespace of a given team.

When the job is completed, no matter the result, an email with a result is sent to the team and the members of the CVP team. The product pipelines will additionally send a UMB message to:

[9] https://github.com/jenkinsci/jms-messaging-plugin/tree/master

- ResultsDB to grab results from the Jenkins server with Greenwave[10] config (stored in dist-git[11] per container) it is used to specify what constitutes success. More about ResultDB and Greenwave later in section 6.1.6.
- DataHub for notifying about testing status with three types of messages:
    1. Pipeline status messages. These messages are sent via UMB to indicate that the pipeline has started (status „running“) or completed (status „complete“, „error“, „aborted“). DataHub uses the pipeline completed message as a trigger to grab the job artifacts from the Jenkins server (DataHub is providing log term storage of the job artifacts for CVP).
    2. Stage status messages. Those messages are identical to pipeline status messages but are used to manage statistics for pipeline failures.
    3. Test complete messages. These messages are also sent via UMB to indicate that a test has been completed and test results (xunit.xml files) are available for DataHub to grab from the Jenkins server.

Information has been gained from the internal website of the CVP team.

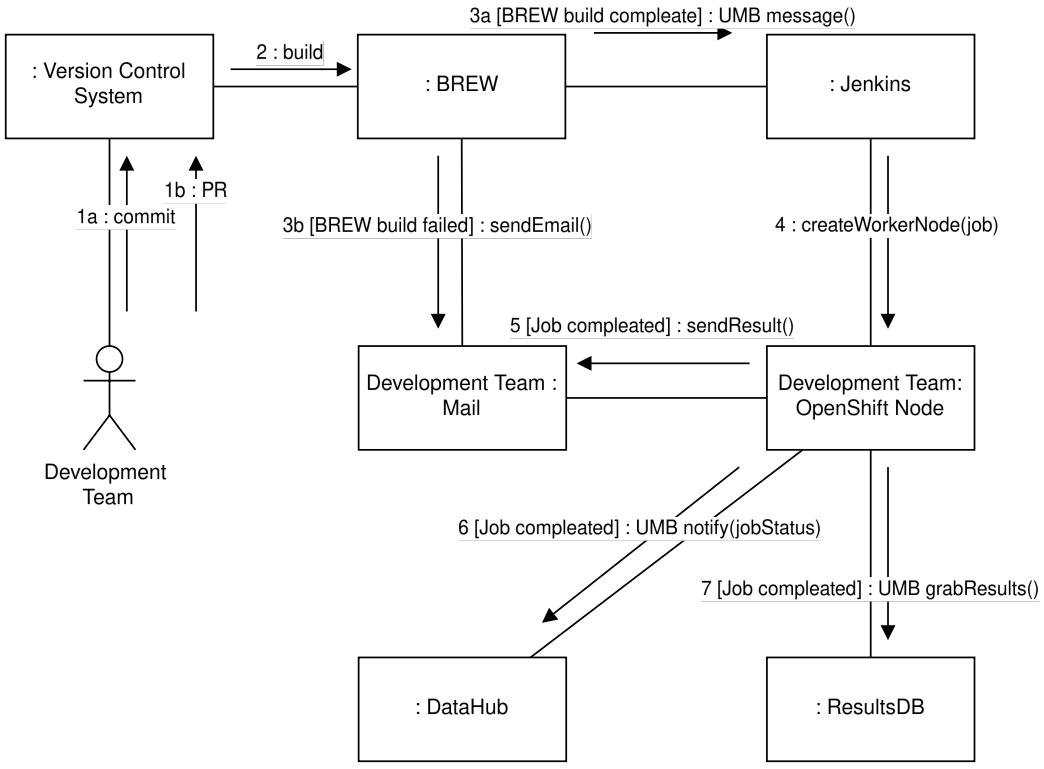


Figure 5.6: Communication diagram of the CVP's workflow.

[10] Can be imagined as a checklist of successful and non-successful tests. The Greenwave will be more described in 6.1.6

[11] Distribution Git is Git with additional data storage. It is designed to hold content of source RPMs and consists of these three main components: Git repositories, Lookaside cache to store source tarballs, and Scripts to manage both

Datagrepper

Is a web application that is used for storing all the UMB messages sent by any Red Hat application. Those messages are stored in Datagrepper for several months and can be queried anytime based on events by username, by package, by message source, by topic and more. Thanks to this, it is possible to find out historical events, e.g., some build has started, finished, or the app has got to some state.

OpenShift Container Platform

RedHat's OpenShift is a hybrid cloud foundation for developing and running containerized applications powered by Kubernetes. It is scalable up to thousands of pods running on one or multiple nodes of a Kubernetes cluster.

On top of bare metal (available from AWS, Microsoft Azure, Google Cloud, and IBM Cloud) is running Red Hat Enterprise Linux (RHEL) or CentOS Operating system that is itself running Kubernetes. OpenShift is a layer that adds features and helps the developer deploy and scale the application using an OpenShift CLI or a powerful Web console.

Waiver

The waiver serves as an "exception" for a test that has not passed in some Jenkins pipeline build. The users can request to add a waiver to the given job with a given test (this is usually done only for product engineering teams whose failing tests are blocking them in some releases). Adding a waiver could be requested by a team whose job pipeline is skipping, e.g., the operator deployment test.

5.2.3 Communication with CVP team in Red Hat

In the company Red Hat, the CVP team is using two primary communication (chatting) channels. The first one is Slack (which is also used for communication with customers), and the second one is Google Chat. In those channels are many operation and infrastructure team spaces. One of those spaces is the "CVP" space that is available on Google Chat for development teams.

As mentioned in Introduction, the CVP team provides support for the basic testing of operators. This includes providing support to those teams whose build has failed, creating waivers, or re-triggering jobs in Jenkins. All of those things are very time-consuming thanks to the need to access multiple services, like checking the status of a previous Jenkins build to prevent scenarios that can cause re-triggering an already successful build or triggering a build that is already triggered or timed out due to external test provided by the given development team.

5.3 Components

The chatbot application can be split into three main components or parts:

- Part that is running on Red Hat infrastructure.
- Part that is running on Google infrastructure.
- Part that connects those two infrastructures using AWS SQS.

The reason for splitting this chatbot application into three parts is security and trying to separate the Google infrastructure from the Red Hat infrastructure. Since the Red Hat part of the app will be accessing Red Hat internal services, it is not desirable for it to be accessible from the outside. Another reason for splitting the app into two parts is that Red Hat has a firewall that protects Red Hat's infrastructure from external access. Due to this, the app was initially intended to be implemented as a Publisher/Subscriber model; this had to be changed due to problems accessing the Red Hat infrastructure from Google infrastructure. Which had to change the messaging, due to which the Publisher/Subscriber model changed in favor of the Request/Response model (more about this change in 6.2.1).

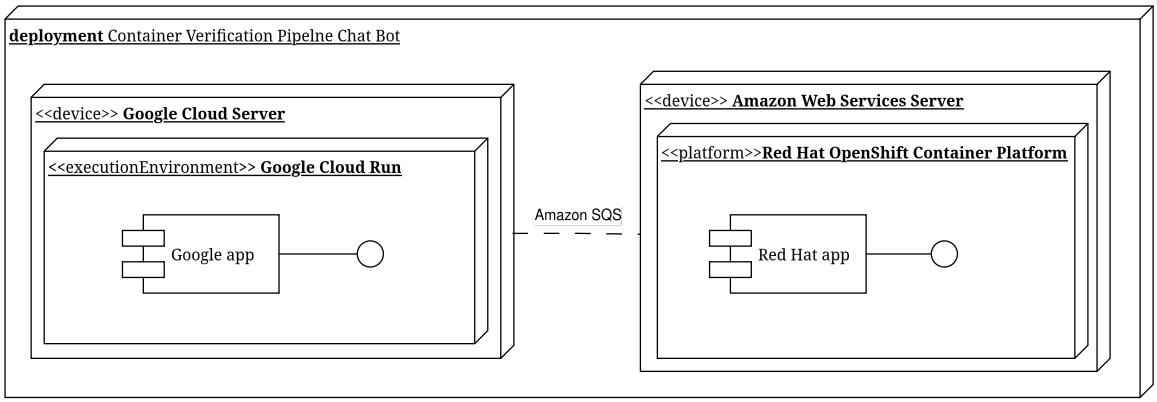


Figure 5.7: Deployment diagram of the chatbot app.

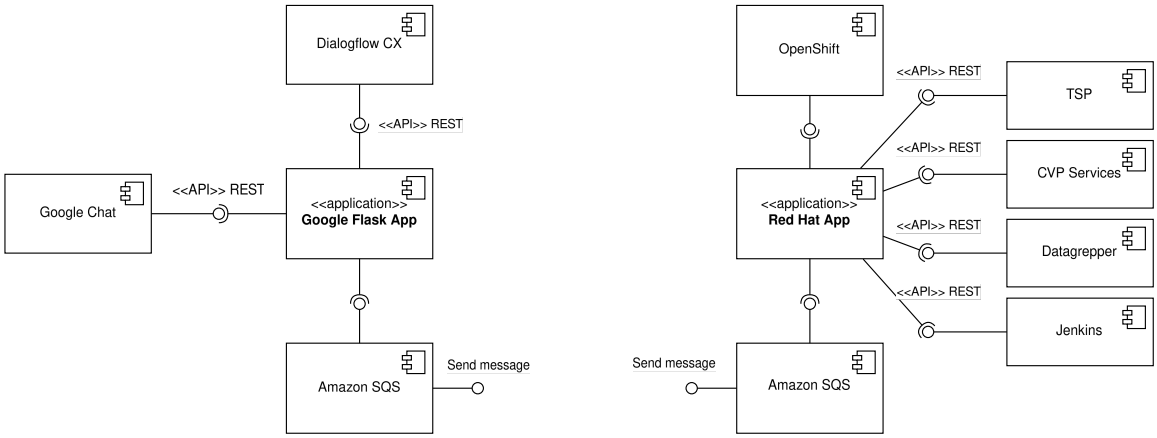


Figure 5.8: Component diagram of the chatbot app.

5.4 Flask

Flask is a small framework by most standards – small enough to be called a "micro-framework", and small enough that once you become familiar with it, you will likely be able to read and understand all of its source code. It provides a solid core with basic services, while *extensions* provide the rest.

Flask has three main dependencies. The routing, debugging and Web Server Gateway Interface (WSGI) subsystems come from Werkzeug[12]; the template support is provided by Jinja2[13]; and the command-line integration comes from Click[14]. Cited from the book *Flask Web Development* [23].

The Flask framework has been chosen since it is fast and lightweight. In addition, it isn't too complex compared, e.g., to Django, which provides a lot of unnecessary inbuilt functionality for a chatbot.

5.5 Simple Queue Service

"Amazon Simple Queue Service (Amazon SQS) offers a secure, durable, and available hosted queue that lets you integrate and decouple distributed software systems and components. Amazon SQS offers common constructs such as dead-letter queues and cost allocation tags." Cited from the website of AWS [4]. The architecture of this service is quite simple since the service contains three main parts:

- The distributed system - In this thesis, it will be the app deployed on Google Cloud and in Red Hat infrastructure.
- The queue - That is distributed on AWS servers.
- The message.

5.5.1 Queues

The Amazon SQS offers two main types of queues, the *standard* one and *FIFO*. Each one of those has a very similar behavior. Still, the main difference between those is that the FIFO queue guarantees that the messages sent will be consumed in the same order in which the messages are sent, compared to the standard one which only "tries" to preserve the order of messages. Each message is distributed to a queue with multiple duplicates in other queues (but the same ones), which creates quite a considerable redundancy, but at the same time, this principle ensures reliability. The queue also automatically deduplicates the same messages. This ensures, that the same message can't be in a queue multiple times.

5.5.2 Lifecycle of messages in Amazon SQS

The sender sends the message to a given queue with attributes specific to the given message. The message is distributed across multiple servers after being received by a specified queue where deduplication is done. Also, when receiving the message, the queue starts a timer for the message processing. If the message won't be processed, it will be automatically removed from the queue. The receiver has to receive the message in the given time specified by the message itself or globally by the queue. Multiple receivers can receive the message, but the last receiver must explicitly delete the message from a queue.

[12] https://werkzeug.palletsprojects.com/en/3.0.x/
[13] https://jinja.palletsprojects.com/en/3.1.x/
[14] https://click.palletsprojects.com/en/8.1.x/

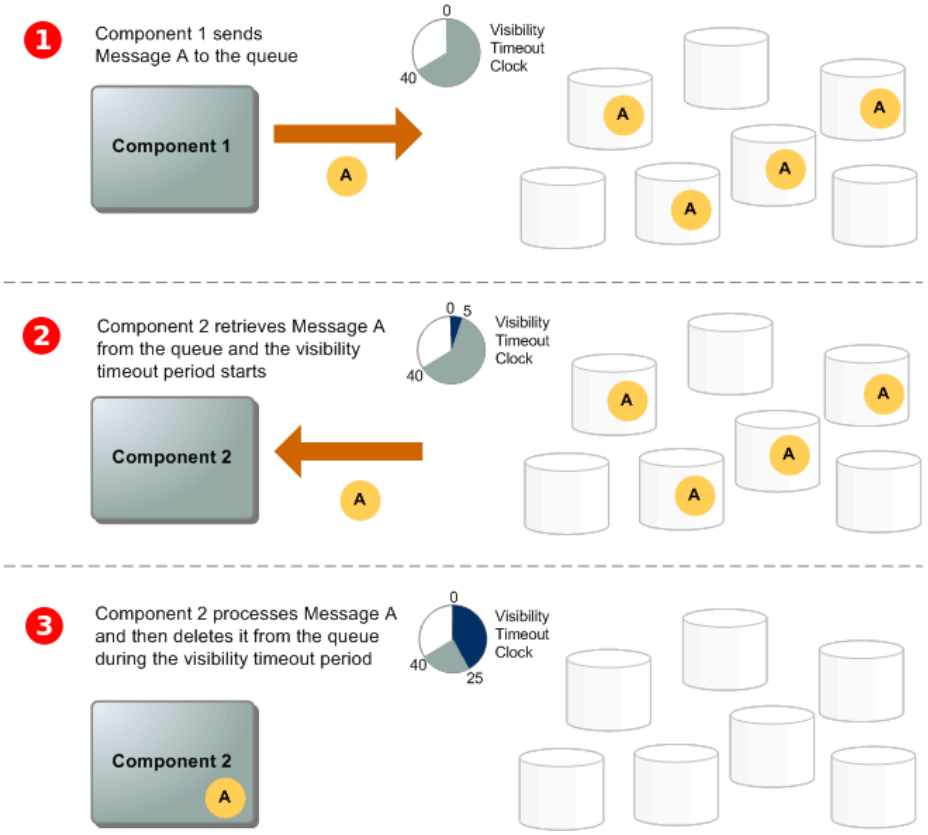


Figure 5.9: Message lifecycle of Amazon SQS messages. Image is available on website of Amazon [5].

5.6 Google's component

The Google's part of the chatbot are contained with four parts:

- Flask app.
- Google Chat.
- Google Dialogflow CX.
- AWS SQS messaging.

At the core of this component is a Flask app that takes care of communication between Google Chat and Dialogflow. When a message is sent to Google Chat, the Flask app will receive the message and forward it to Dialogflow CX using a Python module from Google. In Dialogflow, an intent will be detected and the fulfillment message will be generated and sent back to the Flask app. From there the message will be forwarded once more, but this time to Google Chat instead of Dialogflow. This loop can be executed multiple times since the ML model in Dialogflow needs multiple parameters/information to "to fulfill all the requirements" in the given intent path. When all required information is gained from a user, the confirmation message will be sent to GChat, and the SQS message will be sent to the SQS queue, from where it will be requested by the Red Hat app.

In the Google app, two threads are running that take care of requesting messages (the responses to the sent tasks) from the SQS queue and storing them in a local queue for the

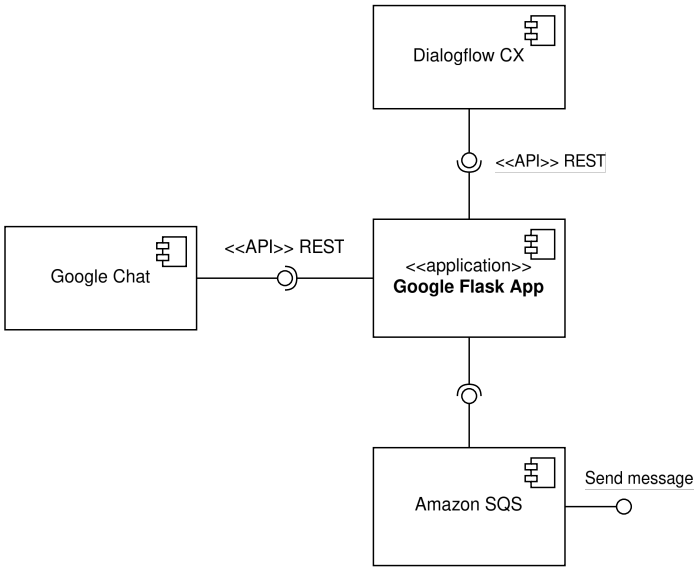


Figure 5.10: Component diagram of Google's part of the chatbot app.

other thread that is sending the messages in the local queue to GChat. The Google Cloud Run service allows the creation of health checks that make sure that if the deployment fails, it will be automatically redeployed. In addition, the chatbot instances (3 of them) are not running all the time since the chatbot is not used that frequently since most of the CVP team is located in the US. Due to this, the initial intent was to scale down the instances since CVP members won't use them. But this leads to delayed sending responses, and the response message might be sent from a Red Hat app to the Google app over the SQS, but the Google app is scaled to 0, and the response will be sent to GChat after a Google app get a new request from a user. Due to this, there is always at least one instance running at any time.

5.7 Red Hat's component

This component contains three main parts:

- Python app.
- Red Hat internal services.
- AWS SQS messaging part.

Red Hat's Python application has two threads. One of them is similar to the Google app, requesting messages (tasks) from the SQS queue and storing them in a local queue, from which the other thread processes the message and executes the script for detected intent. After executing the script, a message with a result will be sent back to the Google app via SQS messaging.

The Red Hat app is deployed as a docker image on Red Hat's product OpenShift Container Platform. Thanks to this, it will be easy to scale if needed (so far, the app runs in three pods) and the CVP team will have complete control over the Red Hat app.

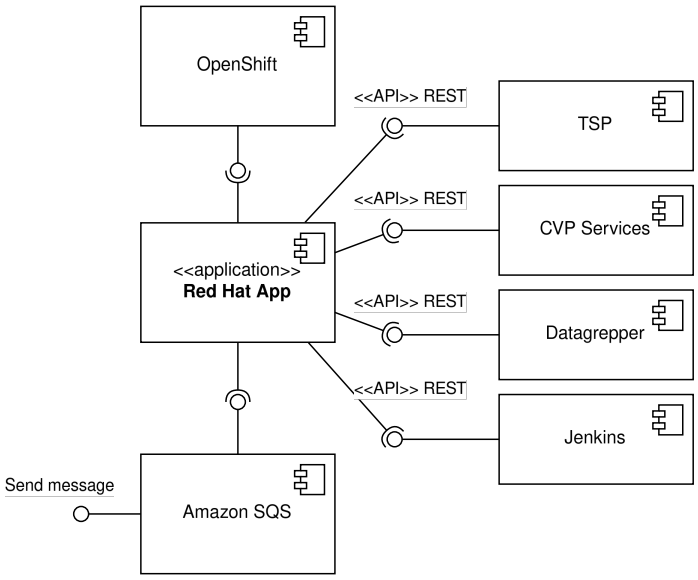


Figure 5.11: Component diagram of Red Hat's part of the chatbot app.

Chapter 6

Implementation

In this chapter use cases will be described, that have been implemented in the chatbot. After that, problems that have been encountered during the work on this master thesis will be discussed.

6.1 Use Cases

Some of the main use cases relevant for members of the CVP team can be things like:

- Re-trigger failed Jenkins job.
- Provide information about failed builds.
- Adding waivers to the NVR[1].

Into this, smaller tasks like providing links for CVP documentation tests or reporting issues can be appended. In addition, to help CVP members in their work, the chatbot can have functionality like:

- Provide information to the end users about the current load on their teams' Jenkins OpenShift project (running jobs, resources available/in-use).
- Aggregate information about jobs that may have run for a given team in terms of successes, failures, aborts, unstable jobs, along with top reasons for failures (over a time period).
- Having some mechanism for analyzing logs and providing solutions for it.
- Add options for other cloud teams, like being able to release the packages with ease.
- Get the last build for the given NVR.
- Check if the given NVR is the newest, or if the user is referring to the old one.
- Check for how long the job has been running and get the time of the last successful build of a NVR.

[1] Marking of bundles the inside the RH, with Name-Version-Release structure

6.1.1 Re-triggering a failed Jenkins job

This is the use case that CVP team members have to do multiple times per day. Before the chatbot, the team members had to complete a sequence of pre-tasks to fulfill a user's request.

Those pre-tasks can be listed, such as checking if all the services on which the CVP team is dependent are online and none of them has an outage. The next thing that the member of the CVP team has to do, is to find a reason why the previous build did not pass[2]. Check if one/many team members have already re-triggered the job and the job is already running.

After all of those things, the CVP member has to connect to Red Hat's VPN and go to the Jenkins website, re-trigger the job with the given NVR, wait until the job is completed, and then copy the link of the team instance Jenkins with a test relevant to the NVR and send it to the user.

The script for this use case is split into three parts:

1. Checking if are all required conditions passed.
2. Re-triggering the job.
3. Getting a team instance Jenkins build URL.

The first part is ensuring the user won't be able to re-trigger the job if some of the internal services have an outage. Another check is that the job won't be re-triggered unless the previous job with the same NVR is still running (since it can be successful, there is no need to re-trigger the job more times than is required) or the previous job ended with the result `SUCCESS` (Into this status is aggregated an internal status `SUCCESS_WITH_WARNINGS`[3]). The last check is that the user can't re-trigger the same job in less than 15 minutes from the previous one. This would stop the user from spamming the re-trigger request each second and doing a DoS attack.

For the re-triggering part, it is necessary to log in to the related OpenShift namespace project where are the Jenkins servers deployed; this has been done with the Python client library. It is required to get a secret login token from OpenShift, which is necessary for logging into the Jenkins Server (running on OpenShift), Jenkins environment route, and deployment type. Thanks to the deployment type, Datagrepper can be queried in the related environment where the UMB messages are stored. In the Datagrepper, a message is filtered that signals the end of the build of an image needed as an "original triggering action" for the Jenkins server. After those steps, a POST re-trigger request can be sent to the Jenkins route gained from OpenShift earlier.

Those URLs will be referred to in the upcoming sections below:

1. This is the Master Jenkins URL: `https://jenkins0-cvp.example.redhat.com/job/jobName/buildNumber` with the specified build.
2. This is the team Jenkins URL: `https://jenkins-cvp-teamID.example.redhat.com/job/jobName1/buildNumber1` with the specified build, which is referred to as "full response" in the section below and "partial response" would be the same URL, just without the buildNumber1 part.

[2]Also, there is no need to re-trigger an already successful build since it would be wasting resources

[3]This means that the job itself failed due to some external test, thus re-triggering would not affect the result itself, or there was reported some warnings

The last part sounds simple but it is a tricky one, since there can be multiple builds building at the same time, and it is not possible to query the Jenkins with `lastBuild` and get its build number. In addition, in the Jenkins Master, there is a delay that takes a few seconds to fetch the team data with the NVR (that the script is searching for). Due to this, there has to be some waiting period or checking mechanism that will make sure that the script won't be searching in the builds that haven't fetched the team data since it would be referring to the old build of the team Jenkins instance (if there exists an old one). This has been done by waiting some period (big enough that the build will appear in the build section) and checking if the `lastBuild` has finished. Thanks to this, the re-triggered build probably won't be the last build, but it will be one of the latest and will have all the required data already fetched with a complete console log from which the script will get the team Jenkins URL with a job name. The reason why mapping can't be done with the team name to the unique `teamID`, which is part of the team Jenkins URL, is because the translation can go wrong, and the `teamID` can be set to a fallback (unknown).

Getting the full URL to the currently re-triggered build is similar to getting the team Jenkins URL with a job name. There is also a delay in fetching the team data, and the CVP Jenkins is currently set so that only one build can be built in the team Jenkins instance simultaneously, compared to the Master Jenkins instance, where it could be multiple. This is quite problematic since each build can take 10 minutes to an hour to finish. Due to this, the script checks the `lastBuild` to see if the build contains the data that the script needs, if so, it will get its build number and return the "full" team Jenkins URL to the user (second item in the list above). The script will check, at most, two builds that are at the time of building marked as last builds. If the script doesn't find given data in the build, it will return only a partial response: the team Jenkins instance URL with the job name but without the build number. The user has to wait by himself or search for the re-triggered build. Thanks to this, the script won't return "full" responses all the time, but at the same time, it won't be waiting for multiple builds to finish since each of them can take up to an hour.

This task can be especially beneficial in an outage, when requests for re-triggering accumulate, and the CVP members can re-trigger jobs directly from chat by providing the NVR.

6.1.2 Getting status from Tactical Status Page

Another helpful task for the CVP team, which a chatbot can do, is to give the CVP team the ability to check if the services that the team is using are available, or not. This can be done by querying the Tactical Status Page (TSP), which provides an up-to-date picture of the delivery pipeline health and outage information of services owned by software production teams.

So far, the CVP team members had to go to the given site and search for the given name they want to know the status of, which can be time-consuming, and you can start losing precious time, e.g., when the Jenkins jobs fail one after another. This task can be done as one of the first steps to diagnose the source of the issue.

The services that the CVP team relies on are CVP pipeline, IIB (manipulates with DB with container operators), Errata (used for managing the process of preparing and shipping software updates for Red Hat Products), DPTP (creates OpenShift clusters, which are used for testing containers and containers operators), Datagrepper, or ResultsDB, whose states

are gained by querying the TSP with the ID of given service. This task is also used as a step in the re-triggering task mentioned above.

6.1.3 Getting status from Jenkins

This is one of two tasks that CVP members can use to check which jobs are currently running in Master Jenkins Instances. This task will check the state of the last build in every job in both Master Jenkins instances.

Each job can be in various states, determined mainly by colors. Some colors are commonly known, such as green, associated with success, and red, associated with error. But others can be confusing, like yellow for unstable, blue for enabled, and gray for either disabled, aborted, or not yet built. There are also some keyword statuses, such as disabled or notbuilt. Each one can have a "currently running" variant, marked as "anime" appended to a given state. This can lead to states like `ERROR_ANIME`, which means some error has occurred in the build itself, but the job is still running (anime).

This "color-ish" schema makes it possible to map a given job state color with associated text, since for users it's more friendly to see aborted instead of gray.

6.1.4 Getting usage from Jenkins

This is the second task that can be used to get usage from Jenkins (can be done from Master Jenkins or team Jenkins instance). This task can be handy when there is an outage or error that may cause the execution of jobs in Jenkins to stop. Thanks to this CVP members can quickly check how many builds are in a queue to start building and how many build executors[4] that creates the team Jenkins instances (in which the builds are built) are ready. This task also involves accessing resources in a given OpenShift project, in which the Jenkins instance is running, and checking how many pods are running, in which state they are, and their name. This can also be used as "safety check" to ensure that in the OpenShift project are no unknown/ unexpected running pods.

This use case was quite helpful when an "error" was present in the AWS, which was not removing unused nodes from the CVP cluster pool. Thanks to this outage, the CVP team could not gain any node (on which the OpenShift images are deployed) from the CVP cluster pool. Due to this, the jobs were not being built, and all the jobs were accumulating in the Jenkins queue[5].

6.1.5 Getting errors from team Jenkins log

This task can be incorporated into the tricky tasks, explaining why the build has not passed the CVP tests and ended with status `ERROR`. Thanks to this task CVP members can quickly ask the chatbot to provide errors in the log of the latest build of a given NVR.

The script will search the log for all the word errors and return them to GChat. This is easier said than done since the logs can be massive, and the script can take dozens of minutes to get all the errors from the log, which would have the opposite impact on the goal of this thesis (speed day-to-day tasks of the CVP team).

When implementing this task, multiple approaches have been tried:

[4] An executor is a slot for the execution of tasks

[5] There were almost 100 jobs in the queue

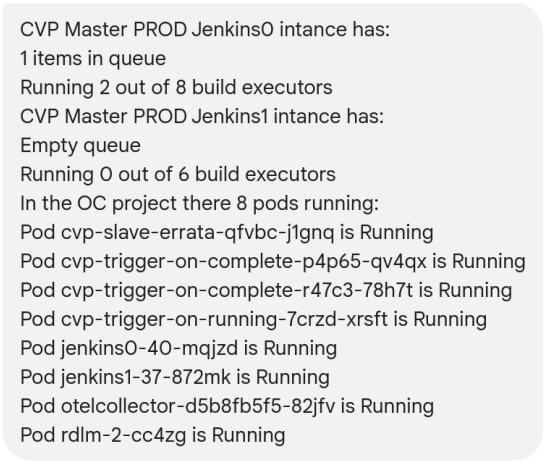


Figure 6.1: Usage from Jenkins Master 0 and 1 instance.

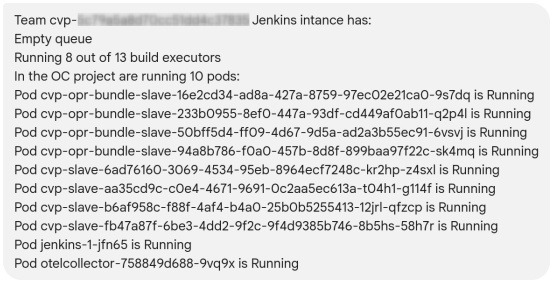


Figure 6.2: Usage of Open Shift team project, with Jenkins usage.

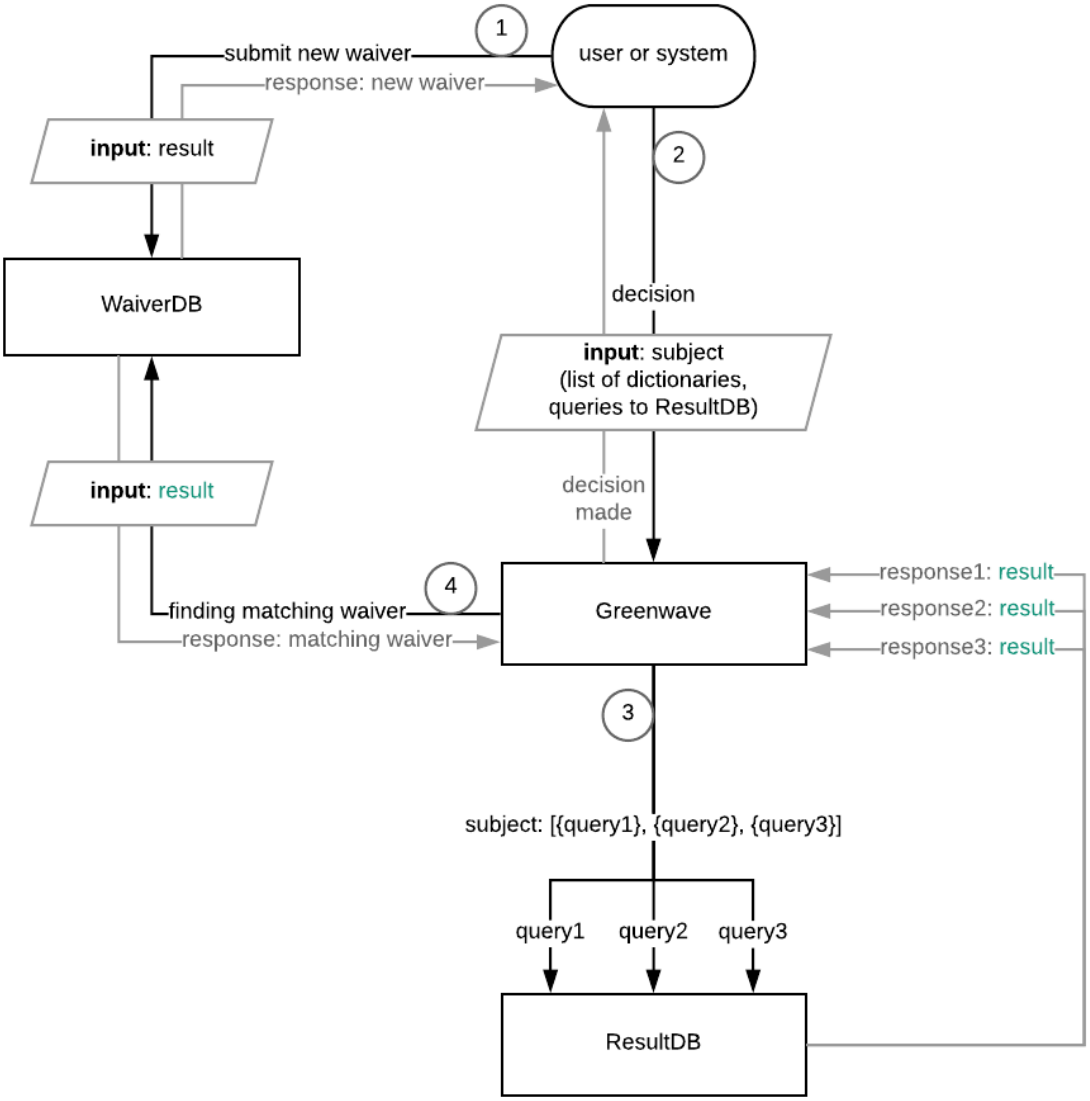
- Basic search with one CPU thread (took approximately 10 minutes), which is almost unusable.
- Trying to split the log into 4 and 8 parts and running the search with 4 and 8 threads (took 5 and 2 minutes, respectively).
- Removing unnecessary lines from the log (took approximately 10 seconds).

The approaches above show that removing unnecessary lines speeds up the search by 60 times. However, with this approach comes another question: Which data is considered redundant and which data is considered valuable? In this thesis, the "non-valid" errors have been analyzed, which are skipped in favor of the real ones that are sent to the user. For this task, it would be ideal to use some pre-trained ML model on logs and extract information about the errors or expand this idea and provide the solution for this error. However, this would not be an easy task and probably could be listed as the Master Thesis itself and be an expansion of this thesis.

6.1.6 Adding waivers

This task aims to create a Waiver for a given Jenkins build. The waivers are stored in WaiverDB, a companion service to ResultDB, for recording waivers against test results. Those two services are necessary for Greenwave, which is a service that decides whether a software artifact can pass certain gating points in a software delivery pipeline based on test results stored in ResultsDB and waivers stored in WaiverDB.

The script is making a POST request to the WaiverDB API endpoint by providing a subject identifier (this is NVR in the CVP), subject type, and test case, which specifies the "name" of the test, an additional information in a comment (can be used for linking with Jira ticket, that was created by the given team for this task). The chatbot could use the internal tool for this, the `waiverdb-cli` tool, but making a POST request has been prioritized since the WaiverDB is returning some data that are not present with the use of the tool. This data was used in the URL link, which is returned to the user afterward.



1. A user or a system submits waivers to WaiverDB (one of the tasks of a chatbot).
2. When a user (or system) queries Greenwave for a decision, it does so regarding a subject. This is a list of dicts that each represent queries to ResultsDB.
3. Greenwave takes each entry in the subject list and queries ResultsDB for all matching results. This gives Greenwave a list of results.
4. Greenwave then takes each result and queries WaiverDB for matching waivers.
    - For "green" from Greenwave has e.g. pass the result for "testABC"; or some result for "testABC" with a waiver.

Figure 6.3: Communication diagram between Greenwave, ResultDB, and WaiverDB. Image with a description has been gained from the internal websites of Red Hat.

6.2 Difficult parts

This section will describe a few parts of the thesis that were quite problematic and that have taken some time. Those parts can be listed as changed messaging alongside deploying to Google Cloud or tricky message sending to the threads.

6.2.1 Change of the messaging

This problem arose when the app decided to start deploying to the cloud. The deploying to the Red Hat's OpenShift Container Platform was without any hiccups, the app is built from the git repository where the code is stored, which meant creating BuildConfigs[6], which is started after every push to the repository. The second config created was a DeploymentConfig[7] alongside the necessary secrets needed for the app to run correctly. At this time the Apache ActiveMQ (Red Hat AMQ) worked perfectly fine, the problem occurred when the deployment of the app to the Google Cloud was tried.

From the beginning, there was a question of which service would be the best option for deployment. Since Google Cloud offers multiple products to deploy your app (Compute Engine, App Engine, and Cloud Run), each one of them has its advantages, like better performance, more freedom (Compute Engine), or better performance-cost ratio (App Engine and Cloud Run). In the end, Cloud Run has been chosen as the product that would be best for the chatbot, thanks to its easy configuration and cost and the fact that Cloud Run is a newer service. On the other hand, one significant disadvantage is the lack of an option to SSH to the revision (instance), which makes debugging the docker image significantly more challenging compared to Google Compute Engine and App Engine.

For successful connectivity, there was a need to create "Serverless Virtual Private Cloud Access connector", which should link the subnet, in which the Cloud Run instance was running with the Red Hat network and the Red Hat AMQ. From logs, it was found that the app can make a DNS lookup and "translates" the URL of the Red Hat AMQ correctly, but is not able to connect to the broker successfully. In addition, after some time, the sender/receiver client thread fails. After some days of debugging, Red Hat support, which takes care of Red Hat AMQ, was contacted for advice on what may be wrong. They said that there exists an "allow list" with external application addresses that they must explicitly specify to get access to the Red Hat AMQ. The IP address has been provided to the support team, and they updated the allow list, but it didn't work even then. Trying different approaches and specifying firewall settings also didn't work, and the connection with the broker was still unsuccessful.

Another option was to reach out to the team supporting Red Hat with the public cloud (Google Cloud, AWS, Microsoft Azure). Even according to them, all was correctly set and should work as intended, but there were some other teams which the solution was working for and some of them that were not (including me). After some time, an IT support representative reached out and said that the problem was not on my side. He also mentioned that they had opened a ticket in Google Cloud concerning the connectivity issues between the Google Cloud projects and the Red Hat network, causing the lack of connectivity. Due to this, the chatbot messaging was forced to change from the Red Hat AMQ to the Simple Queue Service, compared to the original solution, slower but with additional functionality, like de-duplication, which may help save some resources.

[6] Characterize the build strategy of one or many resources

[7] Define the template for a pod and manage to deploy new images or configuration change

6.2.2 Problem with messages

The other problem that has been faced in this thesis is that the app has started creating new sessions in Dialogflow (with each message), this behavior has caused a scenario like this:

1. User sends a message with the request to a chatbot from message thread A.
2. The chatbot will correctly respond to a message (but the message is created in thread B).
3. The user sends a confirmation message/provides additional information (from a thread B).
4. The chatbot will respond with "Unknown option" response (or detect some random intent).

So far, the first idea was that the Dialogflow itself caused the problem; due to this, the whole Dialogflow model has been reworked, but the problem was still present in GChat (in the Dialogflows emulator, the model worked as intended). That meant it had to be the threading mechanism in GChat, from which the sessions have been created. This may have also been caused by the move from spaces organized by conversation topic to spaces with in-line threading by March 31, 2024.

Gaining data from a message

This is one of two problems that were interconnected. In the Google Chat messages section, it was mentioned that each message has been sent into a unique `spaceID` and `threadID`. At first, the Google app has been getting the `spaceID` together with `threadID` from each message (DM and space) equally. But there was a catch: in the DMs, there are no threads, only the `spaceID` that is representative of the DM. Each message has a new `threadID`, similar to the `messageID`, which is also present in the message. Due to this, a new session has been created for each DM message, which has caused the problem shown in the scenario above.

The space messages have not been impacted by this problem (if the user will communicate with the bot from one thread) since the app was creating the sessions from `spaceID` and `threadID`, which was correct behavior.

Sending messages to the spaces

This problem was related to sending messages to the DMs and threading in spaces. At first, the chatbot, after getting all the required parameters from the user, started executing the task without any "confirmation" message sent to the user; this was problematic since the user didn't know whether the task was being executed or not. Due to this, a "confirmation" message has been added that informs the user that the task is being processed and can expect a response soon. The message with a response would be sent asynchronously to Google Chat after being proceed and executed by the Red Hat app.

Google has libraries for this use with a documentation[8], the only problem is that it does not work with a `threadKey` parameter sent in a request, to which they are referring to use it for sending messages to a given thread. The library does not let you send a message with

[8] https://developers.google.com/workspace/chat/create-messages#create-message-thread

"invalid" parameters, like parameter `name`, which they say is "an output-only property; do not use it in a request.".

```
{
  "text": "Hello there",
  "thread": {
    "threadKey": "DOES NOT WORK!",
    "name": "WORKS, Even though this is an output-only property!"
  }
}
```

Listing 6.1: Example of message body sent to given thread

Due to this, the Google app was forced to create response messages by using the Google Chat API endpoint, which allowed specifying the `name` instead of `threadKey` parameter.

Chapter 7

Evaluation and Conclusion

In this final chapter, the execution of picked tasks using the chatbot will be compared to "traditional" ways, with notes from members of CVP team. Finally, some ideas for future improvements to the chatbot will be proposed.

This are some feedback highlights from one of the members of CVP team, alongside the list below: "The bot has proven effective in handling various commands and integrating with Jenkins and other services. Its responsiveness and accuracy in recognizing invalid NVR formats are commendable."

- Responsive and Reliable - The bot's responsiveness to commands, such as "@CVPbot Get status from TSP [serviceName]", has been excellent.
- Error Handling - The detection and reporting of invalid NVR formats in the command "@CVPbot Retrigger job invalid NVR" are handled well, clearly notifying the user of issues.
- Seamless Integration - This seamless integration has improved our team's efficiency and the bot's usability.
- User-Oriented Design - The bot's design focuses on user needs, which is evident from its ease of use and quick adaptation by the team. Your work ensures that the bot is not only functional but also intuitive.

7.1 Comparison to the current execution of tasks

The first one of the compared tasks will be Getting status from Tactical Status Page; in this task using the chatbot took approximately 30 seconds (from sending a request message to receiving a response message), which was slightly faster than the manual way (40 seconds).

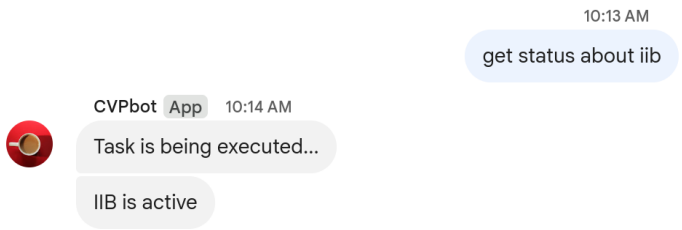


Figure 7.1: Getting status from TSP about IIB service.

An even more significant time difference would be if the user wants to get status from all of the services, which CVP relies on. Due to this, an `ALL` option has been added, which prevents the user from sending multiple requests with a different service and gets status from all services. This would take almost 2 minutes to manually get the status from all the dependent services.

The other task that will be compared is Getting usage from Jenkins. In addition to "Getting status from TSP" task, this task would require logging into OpenShift Console using a SSO token[1], which keeps adding additional time. From there the user would have to select all pods running in the given OpenShift project and query the given Jenkins server for information about executing the build.

The Getting errors from team Jenkins log and Getting status from Jenkins are tasks, that both do not take much time to do manually and can probably be done even faster (compared to the total time of sending a message and receiving the response) if the user was logged in to the VPN already due to a communication delay between applications. In addition, the CVP members can contact different teams for information. Also, stakeholders sometimes provide information about what may be wrong or which test is not passing. The feedback that has been received for a task with getting errors was to add additional filters for removing not relevant errors, like "ERROR: Warning: did not find results in ResultsDB...", which are quite common in the logs and can obscure the real-ones. Returning to the chat a few more lines around the error and returning errors, that are marked as a "FATAL" was also part of the feedback to the Getting errors from team Jenkins log task. The feedback to the Getting status from Jenkins was related to the unfortunate naming schema related to jobs.

Re-triggering a failed Jenkins job saves one "checking" step and adds one additional one to CVP members compared to manual execution. The re-triggering using a chatbot takes three and a half minutes (compared to the total time of sending a message and receiving the response), which is compared to the "manual" re-triggering slightly faster, which takes approximately 5 minutes. The re-triggering task can be combined with a pre-task Getting errors from team Jenkins log from which it can be said (depending on the type of failed test) that if the build is re-triggered, the result would be the same, in this case, this would be the re-triggering of wasting resources. Combining those two tasks is comparable with checks that the CVP members usually do before "manual" re-triggering in this case the manual execution would take around 10-15 minutes.

Adding waivers is one of the more straightforward tasks and one of the more frequent ones. The task replaces a process that requires using *waiverdb* command line interface tool or *curl* command, which requires a special format of providing parameters and switching between windows, which is not ideal. Chatbot removes specifying parameters in the given order and, after successful execution, gives a user URL of the added waiver.

The interaction with a chatbot is generally designed to be quick and intuitive (thanks to the examples provided with each parameter requested by Dialogflow). The execution time can vary due to the current utilization of other services and servers. Due to this, the "raw" time for executing tasks is sometimes longer than the manual/traditional approach. However, a chatbot adds additional functionality to those tasks (like checking the status from TSP or the status of a previous build-in re-trigger task), which will add a slight delay. The most significant advantage is saving precious time since the user will only write four messages to the chatbot at most, which could take at most 20s ($\tilde{5}$s for each message). For

[1] https://auth0.com/blog/what-is-and-how-does-single-sign-on-work/

a re-triggering job, with getting errors from Jenkins (and going through the errors), both of which can be done from the chatbot in 5 seconds, compared to traditional, which can take dozens of minutes waiting for the user to get the team Jenkins build URL.

On the other hand, in almost every task, the user is not getting an SSO token, creating a Kerberos ticket, or connecting to a VPN for each task once more. This would reduce the time executing things manually, but even so, using a chatbot is faster and easier than the manual approach.

7.2 Possible upgrades

First of the upgrades could be the improvement of the task for getting errors from logs introduced in 6.1.5, or even providing error-solving responses. This would be pretty tricky since the chatbot would need some external DB with errors and fixes for these errors. The disadvantage of this would be filling the DB with unique problems with solutions since this process could take a while, along with searching in this DB. AI could also solve this problem by using CVP's knowledge base. This could be done, for example, by watsonx from IBM or Vertex AI from Google[2], which both allow the training of the custom model/ use of the current model while getting information from the knowledge base.

The second upgrade could be changing the messaging from SQS from Amazon back to UMB, which is used internally in Red Hat. This could speed up sending messages between both chatbot apps and provide additional information for the CVP team. This could also be done by creating a basic DB for storing requests to a chatbot, with details like the user, transaction IDs, and metrics (although the Google Cloud Run provides monitoring of requests, instance count, usage of resources, and more).

Another upgrade could be adding more use cases for the chatbot; since it is split into two parts, the upgrade could be done by creating a new script for a given use case. By committing the changes to the git repository, OpenShift will automatically build the Red Hat app and deploy it into OpenShift.

Adding "multi-re-trigger" option, this upgrade has already been implemented on request of members of the CVP team. Thanks to this upgrade, the user can specify the list of NVRs that he wants to re-trigger, and the chatbot will go through them and try to re-trigger all of them using the Re-triggering a failed Jenkins job script. Compared to other use cases, this option has one additional step, which is "confirmation/summary" message, which is sent to the user with the list of specified NVRs. The user has to confirm this summary to start processing and executing this task in the Red Hat app. This was removed from other tasks since the user occasionally would say "no" to an, e.g., "Do you want to get status from TSP about IIB?" question.

The next option could be implementing tasks to list all team Jenkins and their statuses; check the Prowjobs (Prow is a Kubernetes-based CI/CD system. Jobs can be triggered by various types of events and report their status to many different services. In addition to job execution, Prow provides GitHub automation in the form of policy enforcement, chat-ops via /foo style commands, and automatic PR merging. Cited from documentation of Prow [40].) and their failure rates, and monitor active cluster pools and AWS instances. These would provide a holistic view of the environment and help in proactive management.

Integrating the chatbot for other communication channels (like Slack) would be the highest priority of all the updates mentioned above. This would be the highest priority

[2] https://cloud.google.com/vertex-ai?hl=en#train-custom-ml-models

since the CVP channel (from which CVP members get the requests from stakeholders) will eventually be migrated to Slack.

Bibliography

[1] Aashima Gupta and Amy Waldron. *A responsible path to generative AI in healthcare* [online]. Google Cloud, april 2023 [cit. 7.11.2023]. Available at: https://cloud.google.com/blog/topics/healthcare-life-sciences/sharing-google-med-palm-2-medical-large-language-model.

[2] Alexander Rush. *The Annotated Transformer* [online]. harvardnlp, april 2018 [cit. 19.2.2024]. Available at: https://nlp.seas.harvard.edu/2018/04/03/attention.html.

[3] Apache. *Flexible & Powerful Open Source Multi-Protocol Messaging* [online]. December 2023 [cit. 9.2.2024]. Available at: https://activemq.apache.org/.

[4] AWS. *What is Amazon Simple Queue Service?* [online]. April 2018 [cit. 2.4.2024]. Available at: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html.

[5] AWS. *Message lifecycle* [online]. April 2023 [cit. 8.4.2024]. Available at: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-basic-architecture.html#sqs-message-lifecycle.

[6] AWS. *What is DevOps?* [online]. December 2023 [cit. 20.12.2023]. Available at: https://aws.amazon.com/devops/what-is-devops/.

[7] AWS. *Amazon Lex Documentation* [online]. April 2024 [cit. 24.4.2024]. Available at: https://docs.aws.amazon.com/lex/.

[8] Batish Rachel. *Voicebot and Chatbot Design: Flexible conversational interfaces with Amazon Alexa, Google Home, and Facebook Messenger*. 1st ed. Packt Publishing, 2018. ISBN 9781789136883. Available at: https://books.google.cz/books?id=XSxxDwAAQBAJ.

[9] Bento AI. *Cloud Computing Service and Deployment Models: Layers and Management*. 1st ed. Business Science Reference, 2012. Premier reference source. ISBN 9781466621886. Available at: https://books.google.cz/books?id=IZ_V4uK49RMC.

[10] D'Souza Wiltshire, I., Urban, E., Fingold, J., Shemirani, M. L. and Lumb, I. *Choose the right chatbot solution for your use case* [online]. Microsoft, may 2023 [cit. 30.10.2023]. Available at: https://learn.microsoft.com/en-us/azure/bot-service/bot-overview?view=azure-bot-service-4.0.

[11] Eda Kavlakoglu. *NLP vs. NLU vs. NLG: the differences between three natural language processing concepts* [online]. IBM, november 2020 [cit. 16.10.2023]. Available

at: https://www.ibm.com/blog/nlp-vs-nlu-vs-nlg-the-differences-between-three-natural-language-processing-concepts/.

[12] Forsgren Nicole and Humble Jez and Kim Gene. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. 1st ed. IT Revolution Press, 2018. ISBN 9781942788355. Available at: https://books.google.cz/books?id=Kax-DwAAQBAJ.

[13] GeeksforGeeks. *DevOps Tutorial* [online]. May 2023 [cit. 3.10.2023]. Available at: https://www.geeksforgeeks.org/devops-tutorial/.

[14] Giridhar Chetan. *Learning Python Design Patterns*. 2nd ed. Packt Publishing, 2016. Community experience distilled. ISBN 9781785887376. Available at: https://books.google.cz/books?id=161KDAAAQBAJ.

[15] Google. *PaLM 2 Technical Report* [online]. Google, 2023. Available at: https://ai.google/static/documents/palm2techreport.pdf.

[16] Google Cloud. *IaaS defined* [online]. September 2023 [cit. 28.9.2023]. Available at: https://cloud.google.com/learn/what-is-iaas#section-2.

[17] Google Cloud. *Dialogflow CX basics* [online]. January 2024 [cit. 23.1.2024]. Available at: https://cloud.google.com/dialogflow/cx/docs/basics.

[18] Google Cloud. *Dialogflow CX documentation* [online]. January 2024 [cit. 23.1.2024]. Available at: https://cloud.google.com/dialogflow/cx/docs.

[19] Google Cloud's DevOps Research and Assessment (DORA) team. *Accelerate State of DevOps Report 2021* [online]. DORA and Google Cloud, 2021. Available at: https://services.google.com/fh/files/misc/report_2021_accelerate_state_of_devops.pdf.

[20] Google Cloud's DevOps Research and Assessment (DORA) team. *Accelerate State of DevOps Report 2022* [online]. DORA and Google Cloud, 2022. Available at: https://services.google.com/fh/files/misc/2022_state_of_devops_report.pdf.

[21] Google Cloud's DevOps Research and Assessment (DORA) team. *Accelerate State of DevOps Report 2023* [online]. DORA and Google Cloud, 2023. Available at: https://services.google.com/fh/files/misc/2023_final_report_sodr.pdf.

[22] Google Developers. *Build a Google Chat app behind a firewall with Pub/Sub* [online]. December 2023 [cit. 23.1.2024]. Available at: https://developers.google.com/chat/api/guides/firewall/pub-sub.

[23] Grinberg Miguel. *Flask Web Development: Developing Web Applications with Python*. 2nd ed. O'Reilly Media, 2018. ISBN 9781491991695. Available at: https://books.google.cz/books?id=cVlPDwAAQBAJ.

[24] harstack. *CI/CD: Continuous Integration and Continuous Delivery* [online]. GeeksforGeeks, january 2021 [cit. 4.12.2023]. Available at: https://www.geeksforgeeks.org/ci-cd-continuous-integration-and-continuous-delivery/.

[25] IBM. *Watsonx.ai* [online]. April 2024 [cit. 24.4.2024]. Available at: https://www.ibm.com/products/watsonx-ai.

[26] IBM. *Watsonx.data* [online]. April 2024 [cit. 24.4.2024]. Available at: https://www.ibm.com/products/watsonx-data.

[27] IBM. *Watsonx.governance* [online]. April 2024 [cit. 24.4.2024]. Available at: https://www.ibm.com/products/watsonx-governance.

[28] Iyer Balaji and Jeevaganambi Abhi. *Google Workspace User Guide: A practical guide to using Google Workspace apps efficiently while integrating them with your data*. 1st ed. Packt Publishing, 2022. ISBN 9781801070683. Available at: https://books.google.cz/books?id=8r9gEAAAQBAJ.

[29] Jacob Eisenstein. *Introduction to natural language proceessing*. 1st ed. The MIT press, 2019. ISBN 9780262042840. Available at: https://books.google.cz/books?id=72yuDwAAQBAJ.

[30] Jakob Uszkoreit. *Transformer: A Novel Neural Network Architecture for Language Understanding* [online]. Google Research, august 2017 [cit. 15.2.2024]. Available at: https://blog.research.google/2017/08/transformer-novel-neural-network.html.

[31] Katie Dee. *CI/CD still in early phase for much of software development community* [online]. SDTimes, december 2021 [cit. 10.10.2023]. Available at: https://sdtimes.com/softwaredev/report-ci-cd-still-in-early-phase-for-much-of-software-development-community/.

[32] Lena Voita. *Sequence to Sequence (seq2seq) and Attention* [online]. November 2023 [cit. 21.4.2024]. Available at: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html.

[33] Meta. *Natural Language Processing for Messenger Platform* [online]. April 2024 [cit. 24.4.2024]. Available at: https://developers.facebook.com/docs/messenger-platform/built-in-nlp.

[34] Pelluru, S., Troyer, S., Coulter, D., Parente, J., Potapenko, M. et al. *Advanced Message Queueing Protocol (AMQP) 1.0 support in Service Bus* [online]. Microsoft, august 2023 [cit. 11.2.2024]. Available at: https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-amqp-overview.

[35] RabbitMQ. *AMQP 0-9-1 Model Explained* [online]. January 2024 [cit. 9.1.2024]. Available at: https://www.rabbitmq.com/tutorials/amqp-concepts.html.

[36] Rainer Alt and Gunnar Auth and Christoph Kögler. *Continuous innovation with DevOps : IT management in the age of digitalization and software-defined business*. 1st ed. Cham: Springer Nature, 2021. SpringerBriefs in information systems. ISBN 978-3-030-72704-8.

[37] Red Hat. *IaaS vs. PaaS vs. SaaS* [online]. August 2022 [cit. 28.9.2023]. Available at: https://www.redhat.com/en/topics/cloud-computing/iaas-vs-paas-vs-saas.

[38] Red Hat. *What are Red Hat OpenShift Operators?* [online]. June 2022 [cit. 7.5.2024]. Available at: https://www.redhat.com/en/technologies/cloud-computing/openshift/what-are-openshift-operators.

[39] Red Hat. *What is PaaS?* [online]. August 2022 [cit. 29.4.2024]. Available at: https://www.redhat.com/en/topics/cloud-computing/what-is-paas.

[40] Seo, J., Ali, M., Wagner, C., Ebert, T. and wuhuizuo. *Prow Overview* [online]. May 2024 [cit. 9.5.2024]. Available at: https://docs.prow.k8s.io/docs/overview/.

[41] Smart John Ferguson. *Jenkins: The Definitive Guide: Continuous Integration for the Masses*. 1st ed. O'Reilly Media, 2011. ISBN 9781449313067. Available at: https://books.google.cz/books?id=pQ-DG7aT0IgC.

[42] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. *Attention Is All You Need*. 2023. Available at: https://arxiv.org/abs/1706.03762.

[43] Wohl Amy. *Succeeding at SaaS: Computing in the Cloud*. 1st ed. Wohl Associates, 2008. ISBN 9780615256368. Available at: https://books.google.cz/books?id=0NiBAgAAQBAJ.

[44] Yang, J., Jin, H., Tang, R., Han, X., Feng, Q. et al. *Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond*. 2023. Available at: https://arxiv.org/abs/2304.13712.

[45] Zoubin Ghahramani. *Introducing PaLM 2* [online]. Google, september 2023 [cit. 7.11.2023]. Available at: https://blog.google/technology/ai/google-palm-2-ai-large-language-model/.