
# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
**BRNO UNIVERSITY OF TECHNOLOGY**

## FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
**FACULTY OF INFORMATION TECHNOLOGY**

## ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
**DEPARTMENT OF INTELLIGENT SYSTEMS**


# SERVER PRO CONTINUOUS INTEGRATION
**SERVER FOR CONTINUOUS INTEGRATION**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **Michal Šajdík**
**AUTHOR**

**VEDOUCÍ PRÁCE** **Ing. Ondřej Lengál, Ph.D.**
**SUPERVISOR**



**BRNO 2020**

**Brno University of Technology**
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2019/2020

# Bachelor's Thesis Specification


Student: **Šajdík Michal**
Programme: Information Technology
Title: **Server for Continuous Integration**
Category: Software analysis and testing
Assignment:

1. Get acquainted with the technologies of GitHub Webhooks, GitHub REST API, Docker containers and the principles of continuous integration.
2. Study the services of Travis, GitLab Runner, or other implementations of continuous integration.
3. Design a light-weight server for continuous integration that can be distributed in the form of a single binary executable. Design a way of how to process various requests (pushEvent from webhook, pullRequestEvent from webhook, ping, ...). The processing needs to inlude the following: execution of all scripts defined in a configuration file, logging the output of executed script, allowing users to access the logs over a web interface, setting a status for a given commit or pull request using the GitHub REST API. The designed server needs to be able to work with both public and enterprise versions of GitHub.
4. Implement the designed server so that it is executable at least on Linux and Windows 10.
5. Create tests checking basic functionality of the server.
6. Give an example that shows the functionality of the server.

Recommended literature:

- GitHub webhooks
- GitLab Runner

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

| | |
|---|---|
| Supervisor: | **Lengál Ondřej, Ing., Ph.D.** |
| Consultant: | Hartmann Tomáš, SAP |
| Head of Department: | Hanáček Petr, doc. Dr. Ing. |
| Beginning of work: | November 1, 2019 |
| Submission deadline: | May 28, 2020 |
| Approval date: | March 13, 2020 |

# Abstrakt

Tato práce obsahuje popis následujících témat: jaké technologie a principy jsou potřebné pro vytvoření kontinuálního integračního serveru, již existující řešení, proč je potřeba jeden vytvořit a jak integrovat kontinuální integrační server, který byl vytvořen během této práce, na základě informací uvedených v této práci, do pracovního prostředí. Tato práce také ukazuje efekty a některé vedlejší účinky způsobené správnou a nesprávnou konfiguraci uvedeného server pro kontinuální integraci. Uvedený server pro kontinuální integraci je také schopen běžet na MS Windows 10 a Linuxu, aniž by bylo nutné přizpůsobit konfiguraci pro konkrétní operační systém.



# Abstract

This work contains description about the following topics: what kind of technologies and principles are needed for creation of a continuous integration server, already existing solutions, why there is a need to create a new one, and how to integrate continuous integration server which was created during this work, based on the information mentioned in this work, to a working environment. This work also shows effects and some side effects of correct and incorrect configuration of the mentioned continuous integration server. Mentioned continuous integration server is also able to run on MS Windows 10 and Linux without need to adapt a configuration for a specific operating system.



# Klíčová slova

Go, Golang, Docker, ngrok, GitHub, Git, Govis-CI, Continuous, nepřetržitý vývoj, kontinuální integrace, kontinuální nasazení, kontinuální dodávání, kontinuální testování

# Keywords

Go, Golang, Docker, ngrok, GitHub, Git, Govis-CI, Continuous Development, Continuous Integration, Continuous Deployment, Continuous Delivery, Continuous Testing


# Citation

**ŠAJDÍK**, Michal. *Server for Continuous Integration*. Brno, 2020. Bachelor Thesis. Brno University of Technology, Faculty of Information Technology. Thesis supervisor Ing. Ondřej Lengál, Ph.D.

# Server for Continuous Integration

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Ondřeje Lengála, Ph.D. Další informace mi poskytl pan Tomáš Hartmann, jenž byl mým konzultantem ve firmě SAP ČR, s.r.o. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michal Šajdík
28. května 2020

## Poděkování

Rád bych poděkoval panu doktoru Lengálovi, pod jehož vedením jsem bakalářskou práci tvořil. Dále bych chtěl poděkovat panu Hartmannovi, který mi během vývoje servera pro continuous integration poskytoval odborné rady a zdroje.

# Summary

# 1 Introduction

Software development has always been a long process. This was the reason why many software developers started to think about possibilities of how to make software development faster. One of many possibilities was the automation of monotonous tasks. Automation of monotonous tasks contributes to effective management, increased productivity, and a better control over a product production. At the end of the day, all of this contributes to increased effectiveness of the software development team and less manual work. Software developers use Continuous Development principles to accomplish the above-mentioned benefits and many more.

During my internship at SAP ČR, s.r.o. I was a part of a DevOps team. I was there shortly, and a challenge already occurred. Travis announced that they will start to charge money for some of the functionalities that used to be free. This was a big problem because our development team used the Travis-CI server to incorporate Continuous Integration, which is part of Continuous Development principles. Continuous Development principles are here for development teams to help automate and manage their workload. More about Continuous Development is explained in chapter 2.

Our DevOps team needed a solution. The team collaborated and created minimal requirements for the replacement of Travis-CI. The Replacement had to be a free product or ideally open-source. At the same time there had to be an option for on-premise. Or more easily explained, to have option to run the server on your own device. There was a requirement for a subscription to GitHub webhooks events, for example push/pull request, and a possibility to make custom reactions to them, such as push event and pull request event. Because Travis-CI was usually used only on small, short-lived projects, fast set-up time is a must. After all, setting up a continuous integration server can take a lot of time, and software developers need to make a maximal profit from it. For security and isolation reasons, Travis-CI replacement needs to be easy to integrate with Docker, which is used for creating and running isolated environments.

During research for a replacement of Travis-CI, I found some software applications which would fulfill some requirements but not all of them. The search for a replacement of Travis-CI is more explained in chapter 3. At the end of the research, no software fulfilling all requirements was found. The only possible way to have a replacement for Travis-CI was to develop our own continuous integration server. After some time, a draft of a potentially new continuous integration server was created. The final design was simple and is more described in chapter 5. You can find there, what architecture was used, and which part of architecture was implemented by me, and which parts of architecture were reused from other sources. Technologies chosen to be used for the development of our new continuous integration server are described in chapter 4.

My solution does not only consist of a software for automation but also shows dependencies between different technologies and their mutual working relationships, this is better described in chapter 6.

# 2 Continuous Development

Continuous Development [1] [2] [3] represents a procedure for iterative development of software applications and consists of many other mechanisms involved, which would include continuous integration, continuous delivery, continuous deployment, and continuous testing. Continuous Development is primarily directed at automating and streamlining the process of building, testing, and deploying modern technology in a real-world setting. At the very outset of customer software, most of the software and its new versions were officially released over months of production and made available once per year to potential customers.

Additional functionality would often be extensively implemented as well as checked out over many months. A chosen and prepared version would have been published after yet another full year cycle. Customers might notice vulnerabilities, inconsistencies, and other problems in the application because they have to wait for next upgrade to be available and hopefully it contains fixes for all problems that customers might have noticed or would be able to notice. Nonetheless, development companies nowadays are able to immediately deliver new application implementations to their running applications environments. New application implementations can contain additional features, which can be added or changed innumerable times in a mere day. The only thing limiting the number of changes is the hardware.

Development companies are now using continuous development for smaller incremental improvements which could be easily moved to production and then easily removed as problems occur. In order to guarantee that a higher level of quality is preserved until the implementation is introduced, continuous development is focused on successful automated testing of functionality of the mentioned implementation. Through the automation of tests, the programmer's working time for manual quality assurance normally needed in the deployment of the new implementation is decreased.

In order to retain their profitable spot, because software industries are incredibly challenging, software developers need to keep making innovative features. Continuous development encourages software industries to progressively implement new technologies, receive information about their progress, and reduce the amount of time needed for production and deployment of new implementations. The potential advantages of continuous development are the following:

1. New features are introduced easily: Modern ideas are introduced and rapidly implemented using automated processes through continuous development. This therefore significantly improves the pace at which organizations can properly assess updated versions of the implementation. Computer-controlled testing can accurately determine compatibility problems so that bugs which were created due to incompatibility can be nearly automatically detected and resolved. Through quickly launching new applications, organizations have become capable of producing a higher average annual return than most of the other development methodologies could perhaps produce.

2. Improved product value: Continuous Development depends to a large extend on automated tests as well as recommendations from users and their behavior to continue improving the software on an iterative basis. Those who use this technique will react to feedback from users much more quickly. The implementation can be reversed instantly, or the required modifications can be easily introduced. Although software developers will presume, that approaches such as the waterfall approach are essential for users, Continuous Development uses suggestions from users and data to help decide the true significance of both the assignments and projects.

3. Risk Avoidance: Continuous Development allows developers to reduce the risk of starting massive unsuccessful projects by introducing new technologies in a lean and agile fashion.

Simple implementations can be incorporated, evaluated rapidly, and conveniently without the need for any detrimental influence when necessary.

4. Reduced Resource Requirement: Major changes may entail a substantial amount of work in programming, deployment, and evaluation. Continuous development relies on the introduction of automatic technologies to dramatically improve profitability and reduce the amount of time and effort required to successfully build and test new innovative software features.

5. Boosted Profitability: Because the response cycle from clients is accelerated, software engineers have been provided with a complete application workflow. Development team is able to accomplish this thanks to obtaining continual suggestions from different stakeholders during the process. Programmers are also given additional significant responsibilities to better refine the product.

As described above, Continuous Development involves many principles designed to ensure a smooth procedure, whether it is by refining, evaluating, or launching a product on the market.

## 2.1 Continuous Integration

Continuous Integration [3] [4] [5] [6] method collects code from the software engineers working on a specific application and places it inside an application repository. There is quite a variety of bugs that can be produced by incorporating multiple software engineers work into one application. This is where continuous integration takes effect. Continuous Integration is better described as an application development practice that requires all participants of the application engineering group to incorporate their source code into a centralized repository into which all participants have access on a regular basis. Initially, the everyday building of code has been the classic principle for continuous integration. Currently, the standard principle is for each member of the team to apply for the work as soon as it is completed, and for the building of a code to be carried out with each major change. This tends to lead to numerous implementations of software application in a centralized repository during the day, which needs to be integrated together, that is why we use the term Continuous Integration.

Based on the technology stack, the build phase requires activities like those of compiling the source code, assembling the software application, predominantly pushing the software application to such a position where it would be prepared to be deployed, and also be ready to be used. Every single integration would then be validated by automatically generated build runs, which are designed to capture any inconsistencies which could occur in the software application code base, namely absent modules, defective modules, wrong versions of modules, dependencies, or incorrect dependencies among modules.

The building phase might also involve the use of the automated testing of various types. During this process, developers can use variety of tests ranging from unit tests for source code to performance tests and acceptance tests that further significantly improve the value of the tested build. When we use this approach, then developers will know as soon as they are finished with their code, that their code must follow the minimum requirements, and then they can address issues when the tested code is still fresh in their minds. Another significant benefit of continuous integration would be that it gives developers almost instant feedback and notifications on the state of the applications they are collaborating on.

Continuous Integration is a powerful tool for a company that produces software applications. The advantages of Continuous Integration are not restricted to the development department but also influence the entire company in a good way. Continuous Integration offers greater clarity and visibility into the application creation and distribution processes. These kinds of positive effects allow the rest of the company to properly prepare and adopt business strategies.

Continuous Integration allows companies to expand in terms of a team capacity, code base size, and infrastructure. Via reducing application creation complexity and overhead interactions

for communicating. Continuous Integration helps to improve agile software development processes and enable every group representative to have a different code improvement to a different issue. Continuous Integration allows scaling by eliminating any organizational dependency between the production of individual features.

Development companies can now develop software applications in an independent isolated space and have guarantees that their code will be effortlessly incorporated with the remaining portion of the code base. Quicker feedback on management choices is yet another important effect of Continuous Integration. Software developers can test concepts and iterate product prototypes more efficiently with an optimized continuous integration architecture. Modifications can be easily moved forward and assessed for results. Whether they are successful or failures. Bugs or other problems may be easily resolved and corrected.

Continuous integration strengthens total technical coordination and transparency. By adding pull-request workflows connected to continuous integration, programmers can passively share information about their code. Pull requests allows to analyze and give feedback to changes in a software code for members of the development team. Programmers can now approach and work with many other programmers, on functionality per branch, as new functionality is being tested through the continuous integration pipeline. Continuous integration could also be used to assist with the cost of QA services. A productive continuous integration pipeline with strong assurance automated tests coverage could also help shield against regressions and ensure that additional functionality will fit the requirements. In order to merge new code, it has to complete the continuous integration test assessment checklist to avoid any new regressions.

The advantages of Continuous Integration greatly surpasses whatever difficulties exists in the process adaptation. After saying that, it is still worth to be able to recognize the difficulties of Continuous Integration. The main difficulties of Continuous Integration take shape when a development project is being moved from a non-CI environment to a continuous integration environment and the team needs to adapt to new changes in the working processes. Most commercial software projects will embrace Continuous Integration. The earlier they will the less difficulties they will face in possible future adaptation.

The difficulties of Continuous Integration are mainly related to the adaptation of a team and the fundamental technological implementation. If a team does not presently have a continuous integration solution in place, more activity might be necessary to choose one and get going. Furthermore, whenever constructing a continuous integration pipeline, attention needs to be given to the current engineering infrastructure.

When a project has constructed a continuous integration pipeline with automatic test coverage, it is the best practice to continually enhance and expand the test coverage. Every newly added function which enters the continuous integration pipeline ought to have a corresponding suite of tests to ensure that the new code handles things as anticipated.

During Continuous Integration, programmers often use Test-Driven Development. Test-Driven Development is the methodology of creating test cases before any real programming of a new functionality is accomplished. Simple Test-Driven Development will strongly engage the engineering team in the implementation of the anticipated business behavior requirement, which could then be converted into test cases. In a simple Test-Driven Development situation, programmers and the development team will come together to examine the specification or collection of requirements. The whole list of requirements will then be transformed into a code assertion checklist. Therefore, programmers should create a code that suits such assertions.

Many professional software development teams perform pull requests and code review processes. Code reviews are a vital activity, for a successful Continuous Integration. Code reviews are usually done on Pull requests. Once the programmer is prepared to incorporate new content into the existing source code, a pull request is established. The pull request alerts all other software programmers of the latest collection of modifications for the base code, which are prepared for

the integration. Pull requests have become a decent opportunity to start the continuous integration pipeline and run a series of automated validation procedures. An optional, manual validation procedure is typically introduced at the time of the pull request, during which a programmer completes a code review of the submitted source code, which was tested or is being tested. This encourages to study the latest code and its improvements from a different perspective. Other programmers may offer suggestions for changes. Users with the authority to merge pull requests can later accept or decline the pull request. Pull requests and code reviews are an important way of promoting indirect interaction and information sharing within the development team. It attempts to defend from technological debt in the shape of a new information isolation, where individual programmers are the only participants for some functionality of the source code.

## 2.2 Continuous Delivery

The purpose of Continuous Delivery [3] [7] is to provide software at all times which is ready for deployment. Shorter sprints make it more likely to have a shorter waiting time in identifying and getting rid of problems, which contributes to a more sustainable code base. The Continuous Delivery method uses automation to make the whole thing much more effective and successful, but at an essential stage of the process it needs a human involvement.

Rather than having to wait for additional features and functionality for weeks or even months, as it keeps happening with the standard method of the waterfall. Continuous Delivery reduces the time to days, if not hours. As well as keeping a more sustainable code base, which is a huge benefit for every developer.

Continuous Delivery is based on Continuous Integration, and as is with Continuous Integration, every code modification is evaluated immediately when it is introduced to the code base. The Continuous Delivery method involves functional testing, regression testing, and other subsequent testing methods such as pre-generated acceptance testing, in conjunction with automated unit testing and integration testing. The code modifications are submitted in a staging environment for deployment, but only if automated tests had successfully passed.

When the code modifications get to a stage of continuous delivery, a development team is able to deploy the new code to the working environment. There is a little problem for some people. The mentioned problem is the fact that we need to deploy manually. In some cases, it is the best solution for development team to check results manually and deploy if they agree with them or they can do some additional manual testing if it is required. This approach is recommended only in the case that development team does not fully believe in the results of their tests, or if they are not able to cover all use cases with automated tests. If this little absence of automation in deployment is solved, then we are talking about the continuous deployment principle.

The dependence on human interaction will create a bottleneck and that can put development teams at a serious disadvantage for software marketing in a competitive industry.

## 2.3 Continuous Deployment

Continuous Deployment [3] [8] [9] is a method in software engineering where all modifications in the code which would include new features or bug fixes are simultaneously constructed and distributed to various staging environments, this is including product distribution. It is comparable to Continuous Delivery, but it does not depend on anyone reviewing and applying updates to servers manually. There is a significant benefit of using Continuous Deployment. The benefit

is that it offers significant flexibility and more effective workflow. It is a crucial benefit when operating with a team attitude of „*move fast and break things*".

Utilizing Continuous Deployment technologies to automate activities such as assets building and reviewing, that have previously demanded much more human effort. The development team is now liberated to resume programming, saving time and money. It can happen that a release will prove to be troublesome, but there are lots of deployment technology utilities to make rolling back to a secure latest version significantly easier.

Continuous Deployment provides tremendous efficiency advantages for companies with modern solutions. It provides companies the ability to adapt to changing customer requirements, quick deployment, and testing of new concepts and features by development teams much faster. A development team might be able to respond to feedback from customers in real-time with a continuous deployment pipeline in operation. If customers send bug reports or feature requests, the development team will be able to respond and deploy solutions instantaneously. If the development team will have an idea for a new product, service, feature, or functionality, it might be in customers' hands as soon as the code is pushed through to the release repository.

The benefits of Continuous Deployment come at a price. While the return on investment is high, a continuous deployment pipeline can have an expensive initial engineering cost. In addition to the initial cost, continual engineering maintenance might be required, to ensure the pipeline continues to run fast and smooth.

Continuous deployment makes the software engineering process more automatic. Once all the automatic delivery checks have been passed, each code modification is deployed into the production environment as soon as it is possible. Since modifications are distributed rapidly and without human interference to end customers, because of this Continuous Deployment could be seen as hazardous. This needs a high degree of confidence in the current system infrastructure as well as in the engineering team and their automated test suite.

## 2.4 Continuous Testing

Continuous Testing [3] [4] [10] [11] [12] is the method of integrating automated feedback at various phases of the life cycle of software development in favor of increased performance and reliability when handling deployments. At the same time Continuous Testing is a vital force behind the success of Continuous Integration, Continuous Delivery, and Continuous Deployment methodologies and plays a key role in improving the life cycle of software development timelines by enhancing the reliability of code, removing unnecessary expensive inefficiencies, and speeding up DevOps[1] operations.

One of several fundamentals in establishing a functional DevOps methodology is to overcome the difference between quick delivery of applications and stable customer experiences. That being said, the traditional method of manually accumulating suggestions from customers at each phase of a software development (designing, coding, testing, delivery, and maintenance phases of software engineering) has contributed to inefficient and inadequate use of organizational assets, and eventually prolonged installation periods and postponed software releases. The Continuous Testing solves these inefficiencies by supporting „shift left" testing[2] for DevOps teams, offering useful suggestions in the life cycle of software development in early stages, while automating manual testing procedures and decreasing human factor which could introduce errors.

[1] More information at: https://theagileadmin.com/what-is-devops/
[2] More information at: https://devops.com/devops-shift-left-avoid-failure/

Continuous Testing represents an automated method for maintaining control over quality and inter-operation of quality throughout workflows at any point in the life cycle of software development. Through incorporating continuous feedback loops via customer and unit test modules, programmers can have measurable information from customers, that they need to strengthen their code reliability and efficiency before it is distributed. This reliability and efficiency help to avoid disconnections among various members of the DevOps team and facilitates faster delivery of software plans. The Continuous Testing operates through the use of automated tools to launch prepared quality assurance scripts at all development phases. These automated scripts minimize the necessity for daily human interaction whenever executing quality assurance scripts for testing, and consecutively confirming the efficiencies of the software code while guaranteeing that any related feedback is delivered to the correct teams instantaneously.

Should automatic tests fail, the engineering team will be informed at the particular phase of development so that they can make the required modifications to their source code before it influences the development team at numerous different software development life cycle phases. When automatic evaluations pass an examination, projects are immediately moved to the next phase of the software development life cycle, giving the development team a opportunity to build a stable production process, which maximizes efficiency and enhances internal collaboration between departments.

The advanced architectures of today‘s development are multidimensional and interwoven. Continuous Testing helps engineering teams disintegrate these challenges by introducing a responsive, automated testing approach that dramatically enhances the timelines for error detection and reparation. Eliminating or reducing the risk to the company and its expenses is another advantage of continuous testing. Specifically, in massive internally connected systems. An error in one of applications components can have many aftereffects which can trigger unnecessary delays, adversely affecting efficiency and the final result.

# 3 Existing solutions and decision

We need a continuous integration server that is easy and fast to deploy, supports the on-premise or a standalone principle, and is fully free. The requirement for easy and fast to deploy was created, because of the need to apply Continuous Integration and Continuous Delivery principles on many servers and projects independently. The requirement for an on-promise or standalone possibility was created because a company can have sensitive data that needs reliable protection. Ideally, if sensitive data does not leave the internal network. Possible alternatives for continuous integration software were Jenkins, GitLab, Travis-CI, Circle-CI. In the following sections are some points, which explains why the result of finding an already existing implementation was a failure and led us to create a new continuous integration server.

## 3.1 Jenkins

Jenkins [13] [14] [15] [16] [17] is a Java-written open source automation platform which does contain plugins developed for Continuous Integration. Jenkins is being used for Continuous Development and validation of a code, which is a part of a software project, allowing programmers to incorporate their code modifications and making it much easier for programmers to obtain a new build. It also essentially allows programmers to produce new versions of the software project continuously by combining a wide range of development, implementation, and testing techniques. Jenkins also increases the automation scale and is quickly becoming more popular in DevOps circles. One of Jenkins major benefits is its low maintenance requirement and a web-app interface for quick updates. Jenkins also provides a tailor-made solution, since there are more than 400 extensions to support almost any project. Jenkins can effectively support all of software lifetime. Including processes in design, documentation, testing, packaging, staging, delivery, static analyzes, and many more. With Jenkins, programmers can customize notifications in a number of different ways, such as receiving email notifications, pop-ups, etc., and automating them. Through incorporating the right setup for development teams, development teams receive responses almost immediately. Development teams will know if the building of a project has failed. They will learn why the job failed, and how they might be able to reverse it. Jenkins operates on a personal server or a cloud hosting option from third parties.

Theoretically, Jenkins is compatible with any kind of version control system. It also has a strong syntax of pipeline producing script, that serves to automate several quality assurance operations, which include testing. Set-up of the running environment is described in Jenkins configuration file. It can also be personalized to the tiniest detail. However, it is still among the most complex interactions, but with the support of pipeline scripts, it will be a bit less difficult. Extensions in Jenkins are called plugins. Jenkins extensions do not provide access to extra functionality that spans over the basic functions that the application requires to run. However, Development teams using Jenkins needs to use extensions to accomplish things that are sometimes extremely simple. For illustration, a development team will still have to install an extension for Docker, even if developers want to actually create a build-in Docker environment which is a very frequent scenario in modern days. Jenkins cooperates quite inconveniently with Docker via several extensions. There are currently no less than 14 separate Docker extensions for Jenkins. There are several platforms from specific providers directly linked to Docker, and even though six of them are the core platform of Docker. In many ways, Jenkins was designed in the generation of bare metal and virtual computers, just like so many other CI servers. It is a fact that Jenkins tackled on help from Docker. This is not a clever approach for a CI server to work in a constantly growing Docker-based environment.

The possible need for incorporation of all extensions for fully wanted functionality often results in prolonged set up time for Jenkins. This was the reason why Jenkins was not a satisfactory option. The information about the needed set-up time for Jenkins was obtained from company experiences and documentation of Jenkins itself.

## 3.2 GitLab-CI

GitLab Inc. [18] [19] introduced GitLab-CI, a continuous integration service shortly after GitLab was introduced. Along with testing and building projects, the GitLab-CI platform can also deploy the builds to your system, allowing you the possibility to manage various implementations understanding where each piece of code went and what use it has. The GitLab-CI platform supports all major programming languages and provides relatively easy to set-up configuration YAML file. GitLab-CI is free of charge and it can be set up quickly as a component of the GitLab. You must first add the `gitlab-ci.yml` file to your repository root directory and customize your GitLab project for using a runner to begin using the GitLab-CI. After that, each commit or push would activate the three-stage CI pipeline: build, test, and deploy. Each built can also be fragmented into multiple jobs, therefore multiple computer systems can run simultaneously. The tool provides quick and detailed feedback on building successes or failures, making clients aware when something has gone wrong or when something has broken down in the process. The GitLab-CI looked to be an ideal solution for our problem, but the GitLab-CI still had some issues with pull request events and building projects from them. The GitLab-CI generally supports push events and merge request events. Merge request events are from the GitLab and a GitHub support is not fully provided for pull request events.

There were two options: to move from the GitHub to the GitLab or to reject the GitLab as a possibility. Other technologies in the company had to be used by the GitHub. Therefore, this was a good enough reason for not choosing the GitLab-CI.

## 3.3 Travis-CI

The Travis-CI [17] [20] [21] is among the most popular services in the CI / CD community, developed for open source projects, and continued to evolve over the years toward closed source projects. It concentrates on the CI-level, continues to improve development productivity via automated testing and a notification mechanism. The Travis-CI aims to allow users to easily evaluate their code when it is being used. The Travis-CI embraces large and small code alterations and was originally meant to detect building and testing modifications. When a modification is recognized, the Travis-CI can also provide feedback as to whether or not the modification was beneficial. Programmers can also use the Travis-CI to monitor the tests while they are still running and execute a series of tests simultaneously. Slack, HipChat, Email, and so on can also be added to the control mechanism to notify a client about any problems or successful builds. The Travis-CI is good for continuous integration newcomers. Moreover, it is designed for a GitHub service by default. Once the Travis-CI for a specific repository is enabled, the GitHub will alert it as soon as new commits are pushed to this repository or a pull request is posted. It could also just be configured for certain branches or branches with titles matching a particular pattern. The Travis-CI then analyses the branch and executes the instructions in the `.travis.yml` file, which typically builds the program and performs automated testing. Upon completion of that process, the Travis-CI informs the developer in the manner in which it has been customized to do. For instance, by sending an e-mail with the test results (displaying the final outcome), or by posting information to an IRC channel. For pull requests, the pull request is annotated using a GitHub integration, with both the result and a link to the build log.

The Travis-CI can be set-up to run tests on a multitude of different devices, with different technology mounted (such as previous versions of programming language implementation, compatibility testing, etc.) and continues to support software building in many programming languages. Several pieces of the code of the Travis-CI are available on the GitHub, with some private code written in Ruby. At the same time, it provides support for all programming languages that can be compiled on Linux, macOS, and Windows. Testing that can be performed against multiple runtime environments and data stores on libraries or software without getting them mounted locally over multiple different operating systems is also part of the Travis-CI. Another advantage is well established YAML configuration file containing settings and fast project setup due to preinstalled databases and services.

Nonetheless, the Travis-CI has no free plan for a private repository or an on-premise installation for corporations. Only in the case of a project being open source, there is a possibility for a free service to be provided. If a company wanted to use the Travis-CI, it has to pay for the services provided. As a result, the Travis-CI is a no longer feasible solution, because one prerequisite for a CI solution is for it to be a free CI-server.

## 3.4 Circle-CI

The Circle-CI [17] [22] is a continuous integration and deployment cloud-based platform. The goal is to evaluate all the code changes submitted by unit tests, integration tests, and function tests. This platform is a solid choice to break the ice of continuous integration principles, particularly for containerization projects. It is able to work with all programming languages that can be compiled on Linux or iOS platforms. Furthermore, it offers both a private personal server and cloud hosting solutions. Runtime environments and automated testing on a virtual machine are some of the features that the Circle-CI can offer. Similarly to the Travis-CI and the GitLab-CI, the Circle-CI has good documentation of the configuration settings for an easy configuration of the `.yml` configuration file, which has quick set-up time for projects.

The Circle-CI has an out of the box cloud hosting solution that is also fairly easy to manage after being configured. Circle-CI interacts with an existing version control system, such as GitHub, Bitbucket, and others, and performs a series of steps each time modification in a code base is discovered. These modifications in code might be commits, opening pull requests, or any other code modification. Each modification of the code generates a build and executes the testing according to the initial configuration and specification in a clean container or a virtual machine. Every build consists of several stages involving dependencies, testing, and deployment. If the build passes the checks, it can be deployed based on your choice of an AWS, a Google Container Engine, an SSH, or any other deployment technique. The success or failure status of the builds and tests in the discussion is often sent through Slack, Internet Relay Chat, or a variety of other informing platforms so that development teams will remain informed. It is necessary to keep in mind that for a variety of languages the Circle-CI needs some adjustments and improvements, so it is better to go through the documentation for a preferred language. A kind of a special feature of the Circle-CI is ability to auto-cancel outdated GitHub builds. If a newer build on the same branch is activated, the system recognizes it and shuts down the existing builds that may be running or waiting in a queue, even though the build is not fully completed.

The Circle-CI was not chosen because Microsoft Windows is not a supported operation system and the Circle-CI is paid in case of a self-hosted solution, which breaks more than one of the company requirements for a CI-server.

# 4 Technologies used

During my research about the workings of other CI-servers and requirements needed for our CI-server, I discovered a great number of technologies. Some of these technologies were beneficial for creating a new CI-server and some of them were used for its creation. Technologies that were used are described in the following sections.

## 4.1 Go/Golang

The Go/Golang [23] [24] [25] was the best option for building a CI-server based on my research. The Go/Golang belongs to the group of compiled multi-paradigmatic programming languages. It fulfills the following paradigms: imperative, structured, and object-oriented. A code written in the Go/Golang is directly compiled to a machine code without the need for a virtual machine to be run on. One of the most important facts is that the Go/Golang is easy to use mainly because of its intuitiveness. At the same time, the Go/Golang has a lot of needed functions in base libraries. Thanks to this fact I did not need many third-party libraries. The high performance of the Go/Golang is another of its advantages. Performance of the Go/Golang is close to the C/C++ programing language. There is also a simple declaration and initialization of variables, thanks to the help of derivation based on the type of values format. The Go/Golang has a low number of keywords which helps to lower compile time. Powerful concurrency handling, even when compared to the Java programming language. Another advantage of the Go/Golang is a garbage collector, which contributes to a memory safety by taking care of the deallocation of allocated memory. The Go/Golang does not use inheritance principle but embedding. Even so, developers are able to simulate the functions of an inheritance. Overriding methods or operands are not supported. If developers want to change the type of a variable, then developers need to retype it manually. In some other programming languages like the C/C++, there is something called implicit retyping, which changes the type of a variable if it is needed and is possible based on the predefined rules and conversion of types.

High concurrency control was the main benefit of the Go/Golang, which is thanks to a CSP-style concurrency. In the Go/Golang it is possible to use something called goroutines, which allow us an easy and efficient implementation of parallel processes. Goroutines work similarly like threads. Goroutines are functions of methods that can be run at the same time with other functions or methods. We are also able to run goroutines inside goroutines. Users can consider goroutines to be threaded, but these threads are under the control of a system, and the system decides how to allocate processing time, if it is blocking or no, etc. Threads are more heavyweight. Compared with a thread the cost of creating and cost of maintaining a goroutine is much smaller. It is therefore normal for Go/Golang implementations of software to run on a multitude of goroutines at the very same moment simultaneously. It is also possible for multiple goroutines to run on one thread, but not necessary. Goroutines were designed for maximal effectivity. That concludes to the fact that if we use the Go/Golang with its goroutines we will have an efficient way to divide our workload to multiple processing cores of our server.

## 4.2 Docker

Docker [26] [27] [28] [29] is an open-source platform for containerization. Docker allows developers to assemble software programs into containers. Containers are modular executable parts that incorporate a software program source code with all libraries of the operating system and dependencies necessary to run the code in any work environment.

Although programmers are capable of constructing containers without Docker, Docker makes containerization easier, simpler, and safer to construct, deploy, and maintain. Fundamentally it is a set of tools that allows developers to create, deploy, execute, modify, and stop containers while utilizing basic instructions and time preserving automation.
This indicates that programmers should concentrate on developing software without stressing about the environment it will eventually run on. This also enables them to have a head start as a part of their software development, by using one of many software applications, which are already established, to run in a Docker container. Docker provides versatility for operations employees and theoretically decreases the number of machines required due to its smaller footprint and a reduced overhead.

Containers are facilitated by a process isolation and a virtualization of the operating system, which makes it possible for multiple application component parts to share the processing power of a single instance of an operating system in the similar manner that the machine virtualization allows multiple virtual machines to share the processing power of a sole machine. They also might be considered to require three software types: Builder tools utilized to construct a container. Engine tools utilized to operate a container. Orchestration tools utilized to supervise containers and their handling.

Containers provide all of the virtual machines advantages such as process independence, budget-effective extensibility, and disposability. But the extra abstraction layer, on operation system level, provides considerable extra benefits, for example, them being lightweight. One of attractions of using containers is their capacity to crash elegantly and re-launch automatically or upon request. Whether the destruction of a container is triggered by a fatal accident or merely when it is no longer required when server traffic is small. Containers are inexpensive to launch, and they are intended to appear and disintegrate quickly and smoothly. Since containers are intended to be short-lived and spawn new instances as much as necessary, it is presumed that they would not be supervised and handled in real-time by a person, but rather they will be automated. Another benefit of containers is that they improve developer productivity in cases where we need to use an isolated environment for development purposes. Containers are quicker and cheaper to build, distribute, and restart in comparison to virtual machines. This renders them suitable for use in continuous integration and continuous delivery pipelines and particularly suited with Agile and DevOps techniques which should be embraced by engineering teams.

## 4.3 Ngrok

Ngrok [30] [31] [32] is a cross-platform tool that allows users to use the minimum effort needed for making a local development server, which is accessible from the Internet. The ngrok tool makes the self-hosted web server pretend to be running on a ngrok.com sub-domain, which means no public IP or domain name is required on the mentioned self-hosted web server. Reverse SSH tunneling may produce comparable capabilities. However, this involves more configuration and maintenance of your own remote web server.

By establishing a TCP tunnel that is supposed to live for a lengthy time, from a seemingly randomly given sub-domain on ngrok.com to the self-hosted web server, Ngrok is also able to overcome NAT Mapping and firewall limitations. After deciding on the port on which a self-hosted web server will be listening on, the ngrok client opens a protected communication channel to the ngrok server so that anyone with the particular ngrok channel address is able to send requests to the self-hosted web server.

By design, HTTP and HTTPS endpoints are produced by ngrok, which makes it ideal for evaluating integrations with third-party services or APIs which may require creditable SSL / TLS domains. Alternate use scenarios include fast delivery of local samples to customers,

evaluating back-ends of apps, and accessing storage server resources which are running on a home computer.

One of the ngroks appreciated functionality is its capability to monitor and repeat HTTP requests via the web console interface of ngrok. The repeat option is extremely beneficial whenever re-testing or evaluating API requests or webhooks because all header information and request/response information can be conveniently analyzed in the same environment through the use of the console user interface.

## 4.4 GitHub

GitHub [33] [34] [35] is a website and a cloud-based service which lets programmers deposit and maintain their files. For the best understanding of the GitHub, there is a need to fully know what the Git is. The Git is a popular open-source version control system. Explicitly, the Git is a distributed version control system, which implies that every programmer's computer has the whole existing code base and history accessible.

Version control allows engineers to monitor and administer modifications to the code of an application project. As an application project expands its code base, it becomes more and more important to monitor the versions of the code for better management. Version control allows engineers to work securely by using cloning and merging operations. A programmer checks out the origin code source with cloning. The programmer can then modify a portion of the code securely, without affecting the rest of the application code. If the programmer has his or her portion of the code functioning correctly, then he or she can make it official, by combining the new version of the code back into the original code source. All of the above modifications can then be registered and can be reversed if needed.

GitHub supports the configuration of so-called webhooks. Webhooks are used for automatic communication purposes. Webhooks help to facilitate create or established integrations, such as GitHub Apps or OAuth Apps, that actively listen to the GitHub for chosen events. When one of those events is activated, they will dispatch a payload of an HTTP POST to the specified URL address of the webhook. Webhooks are used for updating an external issue monitoring, activating CI builds, updating a backup image, or even deploying to an application server. It is only the users' imagination that is limiting them. Webhooks may be enabled on a specific repository or organization. When enabled, the webhook will be activated each time there are one or more events that have been set to subscription. A development team can specify what events they want to obtain payloads from when they configure a webhook. They can also sign up to any current or future events, by simply listening to the chosen events that they decided on managing. It is useful for reducing the number of HTTP requests needed to get resolved on their server. They can modify the collection of events that are listened to at any time via the API or user interface.

Every event encompasses a collection of activities that may occur to the company and/or software project code base. For instance, if a user listens to a push event, the user will obtain specific payloads each time the push is executed.

# 5 Architecture

We can divide dependency of functionality for Govis-CI to 5 parts: Govis-CI server, Govis-CI client, GitHub API, and additionally it would be Docker and ngrok.

Figure 5-1 represents communication between 3 basic parts of continuous development supported by Govis-CI, namely: Govis-CI server, Govis-CI client, and GitHub API. Developer or Govis-CI client pushes commit containing changes to GitHub server or creates pull request for merging of the code, this represents step `1. PC / PR`. GitHub server receives push commit or pull request, then it views settings for webhooks, and if there is a webhook with configuration, then he extracts information from webhook configuration and acquires URL that it should send information about push commit or pull request to. This URL is supposed to be a domain address via which the Govis-CI server should be reached. Event notification also called event message is sent to the configured URL, this represents step `2. event message`. Govis-CI server parses event message and extracts needed information, for example, repository, branch of the repository, push/pull request type of message, etc. After extracting information Govis-CI server clones branch of repository corresponding to push commit or pull request, this represents step `3. git clone branch request` and `4. git clone branch response`. Every repository on the GitHub server, that wants to use Govis-CI should contain a `.govis.yaml` file. After cloning of repository is done, the Govis-CI server evaluates a `.govis.yaml` file by running scripts from it, this represents step `5. run .govis.yaml`. Govis-CI server inspects on what operating system it is running and based on the operating system Govis-CI server then runs scripts in PowerShell or Bash. From the evaluation process, the Govis-CI server has results, and based on results the status of a commit or the status of a pull request is set on the GitHub server, this represents step `6. set status`. The state status, which was set can be viewed by the developer. On the GitHub server, it is possible to configure a repository to allow code merges from pull requests, only if it was evaluated by CI-server, and the result of the evaluation is a success. On the GitHub server, it is possible to configure the option of informing developers through e-mail notifications about changes of status of pull requests, this represents step `7. e-mail`. The developer is able to view the status of a pull request or a commit by simply opening the commit or the pull request via GitHub or its API, this represents step `8. get results`.

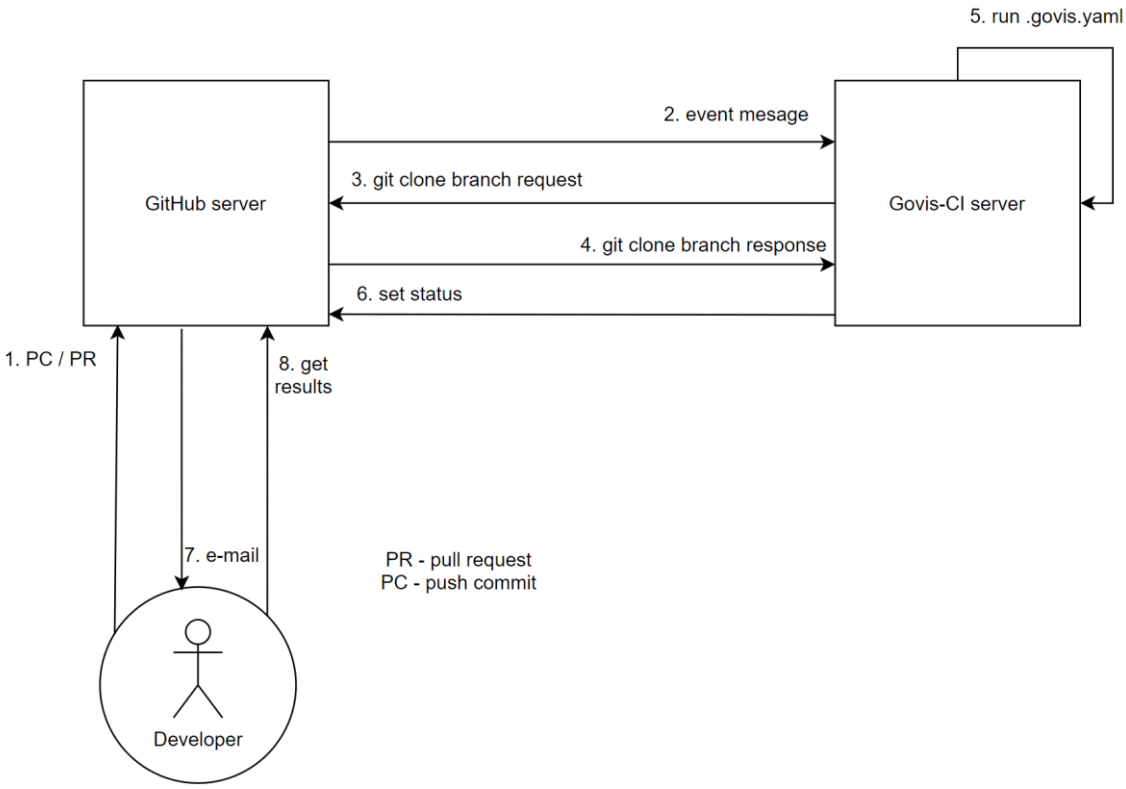


Figure 5-1: Communication between Developer, Github, and Govis-CI

For making Govis-CI accessible from the internet development team can use ngrok. To use ngrok or similar solution is only meaningful in case the Govis-CI server does not have a public IP address or a domain name that is accessible from the internet. If a team has a Govis-CI server, which is not accessible from the internet then a Govis-CI server will not receive `event message`, which is the second action in Figure 5-1. When the development team incorporates a ngrok service into their communication process flow, then the ngrok service becomes a middleman between a Govis-CI server and a GitHub server. In this case, communication between the developer, the GitHub server, and the Govis-CI server changes to communication between the developer, the GitHub server, the ngrok service, and the Govis-CI server. This situation can be seen in Figure 5-2. `2.1. event message` represents `2. event message` from Figure 5-1 and is sent from the GitHub server to the ngrok service. `2.2. event message` represents `2. event message` from Figure 5-1 and is sent from the ngrok service to the Govis-CI server. Ngrok service will provide an accessible URL, which has to be configured on the GitHub server webhook for sending `event message` to the Govis-CI server.

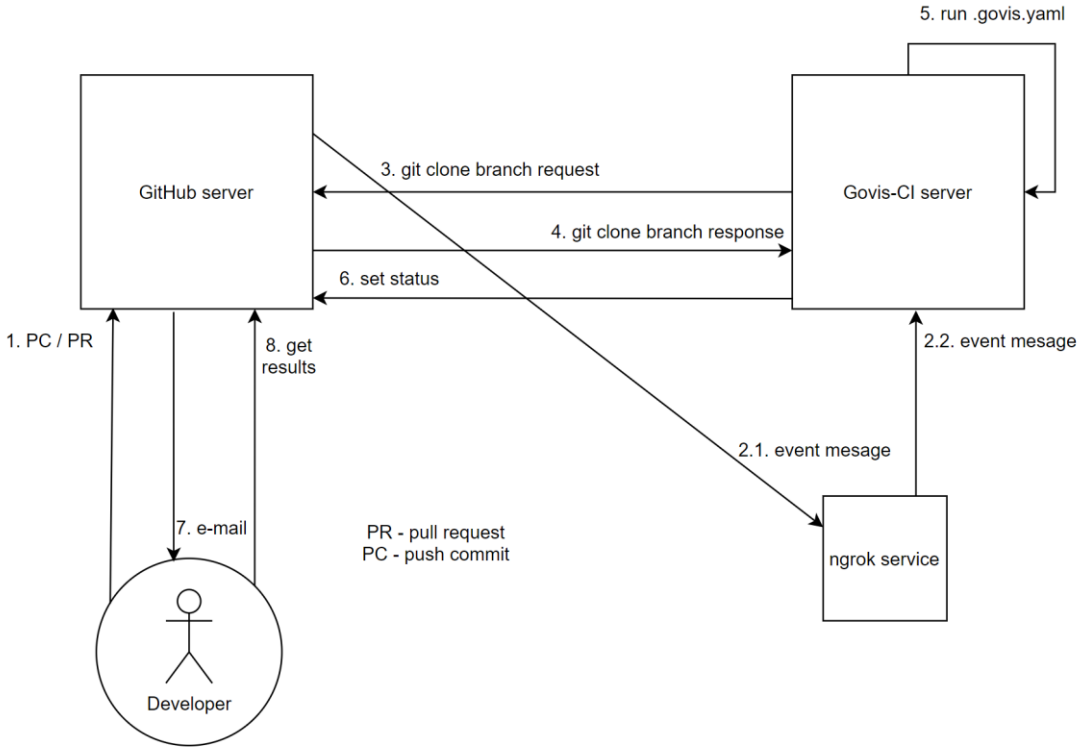


Figure 5-2: Communication between Developer, Github, ngrok, and Govis-CI

Docker can be used for virtualization and more secure handling of the Govis-CI server. A development team can incorporate Docker into the communication process flow for the Govis-CI server and other components. This often results in several advantages, which are described in Section 6.2. The development team can run several virtual instances of the Govis-CI server on one physical server. Illustration of two virtual instances of the Govis-CI server are in Figure 5-3. This means, that one physical server can be used for testing more than one GitHub credentials. In the case, that physical server has a public IP address and virtual instances of Govis-CI server does not use ngrok, then there is a need to set a different port for each virtual instance of a Govis-CI server, if it is not set, then a virtual instance of the Govis-CI server with the duplicated port will not start. In the case, that a ngrok service is used then the development team can run ngrok service in each docker container separately, resulting in every virtual instance of the Govis-CI server having different URLs and port mapping to localhost. This means that the development team has a choice if they want to configure the Govis-CI server or ngrok service. It is possible to configure the physical server, then when the Govis-CI server crashes then the docker container will be restarted, which results in the restart of the Govis-CI server and making a concrete virtual instance of the Govis-CI server available again. During problems with one virtual instance of the Govis-CI server, other virtual instances of a Govis-CI server will be not affected. This all is thanks to the docker containers, which are running completely isolated.

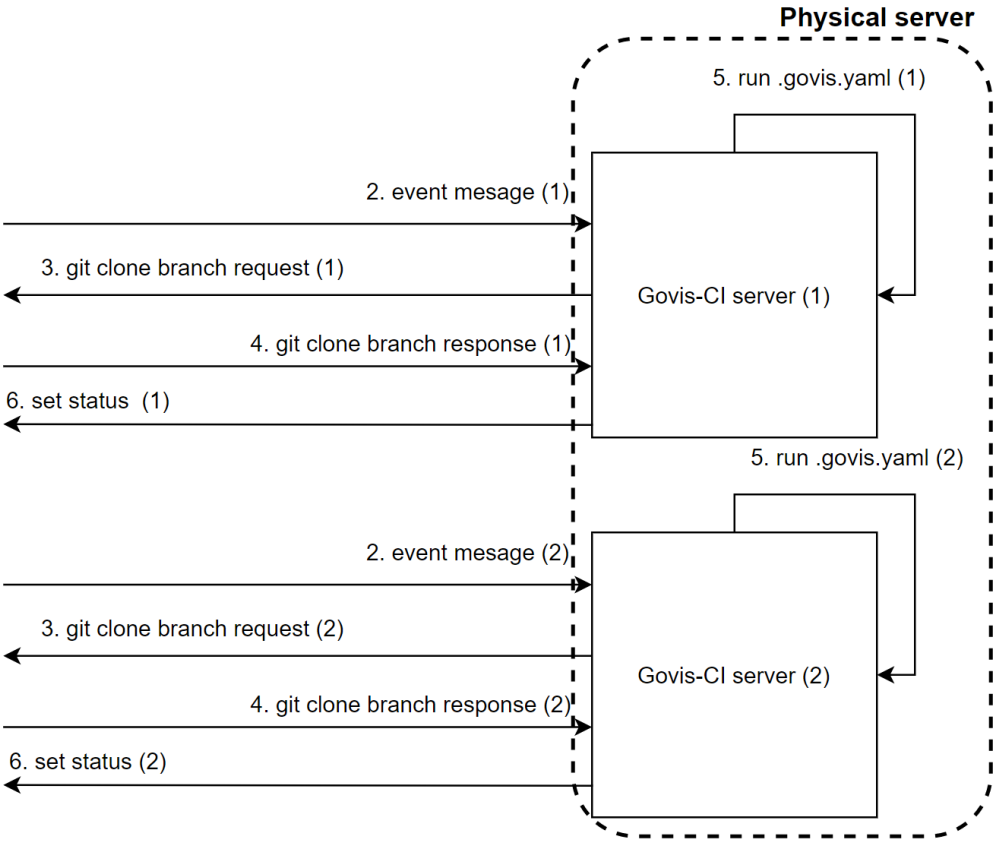


Figure 5-3: Two virtual instances of a Govis-CI server on one physical server

Govis-CI server has a very simple web user interface, which is basically a normal HTML web page. Govis-CI servers web page is used just to see some information about the Govis-CI server. For example, logs from running tests, or log of who tried to access the Govis-CI server and what kind of requests were made. Simple communication is represented in Figure 5-4. Reason for simplicity of the web page is that in some cases users use just console, and if there was an excessive amount of CSS and JavaScript, it would become unreadable from a terminal.

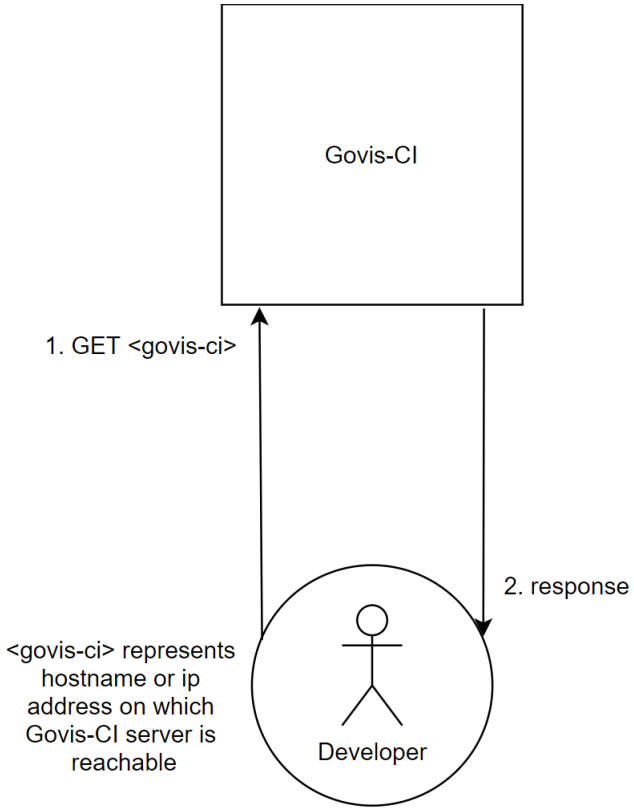


Figure 5-4: Communication between a developer and  a Govis-CI server web

# 6 Walkthrough

In the following subsections will be explained what is necessary for installation of Govis-CI server, how to configure Govis-CI server and different ways how to do it, how to configure GitHub repository, how Docker is used and what docker files are prepared for better integration of Govis-CI and Docker, how to install additional software into docker container for Govis-CI needs. How can the user configure the configuration file of the Govis-CI server and an example of usage of the Govis-CI server.

## 6.1 Installation

The first thing users need is to get binary or "`.exe`" file from one of the sources. Users can also download the whole Govis-CI project and build it from scratch. For this purpose, they need to be in the Govis-CI repository and type `go build` for the predefined build of Govis-CI. If users just want to run Govis-CI fast then type "`go run .`", but this way users will lose some possibilities of configuration, and this form or starting Govis-CI is intended only for development and testing purposes.

When users build Govis-CI they get a binary or an executable file based on their operation system. This functionality is implemented in `go.mod` and `go.sum` files.

### 6.1.1 GitHub

Users need to ensure, that they are able to authoritatively connect through the GitHub REST API to a GitHub repository for the purpose of changing the statuses of commits and pull requests. For this purpose, users need to get some information from the GitHub website.

Users need to log in to the GitHub website with an account that has the rights for setting status for GitHub commits. After being logged on GitHub then they continue by clicking on their profile in the right top corner and clicking on `Settings` as can be seen in Figure 6-1. Now users can see on the left side of the website `Personal settings` and among them is `Developer settings`. They click on `Developer settings` as can be seen in Figure 6-2 and continue with choosing `Personal access tokens` as can be seen in Figure 6-3. After this step they can see their personal access tokens if they have some. These tokens are used for verification of their identity and for determining rights, which were defined during the creation of the token. A personal access token is used for authoritative and restricted access to everything that users are able to access.

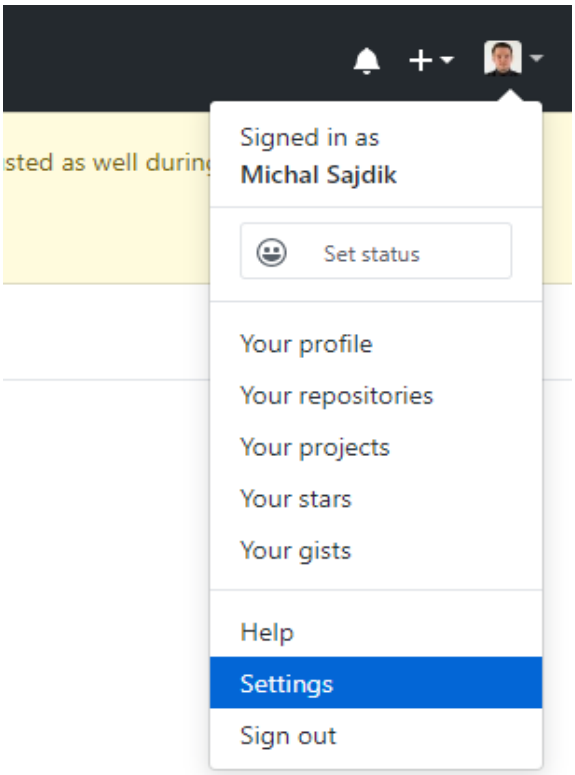


Figure 6-1: Profile menu

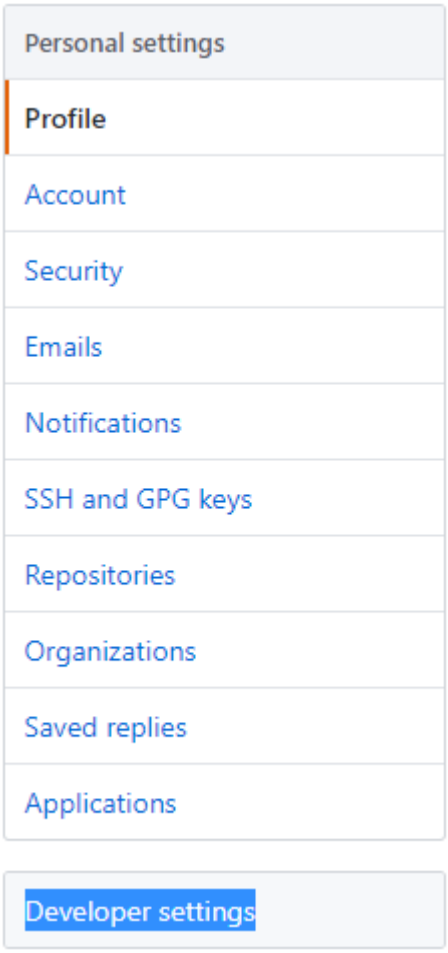


Figure 6-2: Personal settings

Users generate a new personal access token by clicking on `Generate new token` as can be seen in Figure 6-3.

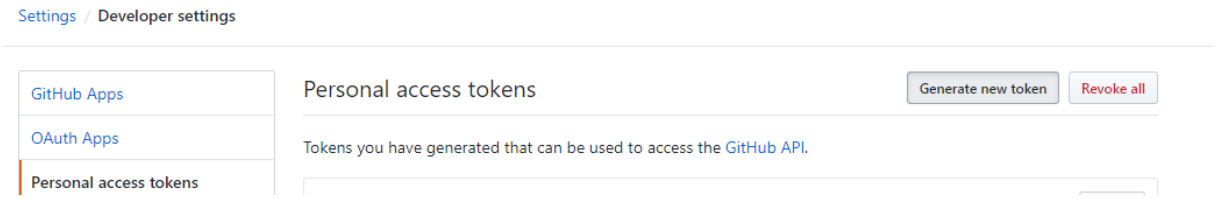


Figure 6-3: Personal access tokens list

Users need to fill the `Note` field which is mandatory. Basically, they just write a name or some identifying information for a new personal access token, this can be seen in Figure 6-4.

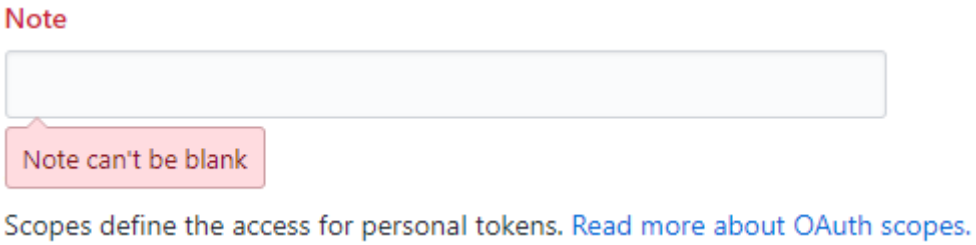


Figure 6-4: Note field for identification of a personal access token

Now they need to choose the access rights for their personal access token, it is recommended the following rights which can be seen in Figure 6-5, but users can choose more of them. In the case that they choose not to give some needed personal access rights to the created personal access token, then it is expected that some errors might arise during a run. Often resulting in one of the following:

- Govis-CI being unable to access the repository, thus being unable to test the project
- Govis-CI being able to access repository. Govis-CI will make tests and everything around. Results will be available on the Govis-CI server, but it might be unable to set a status on GitHub for the commit or the pull request, thus it will look like Govis-CI is not working. For this reason, Govis-CI has control flags, which indicates if the personal access token is valid. This can be seen on the Govis-CI web interface, which will be mentioned in section 6.5.

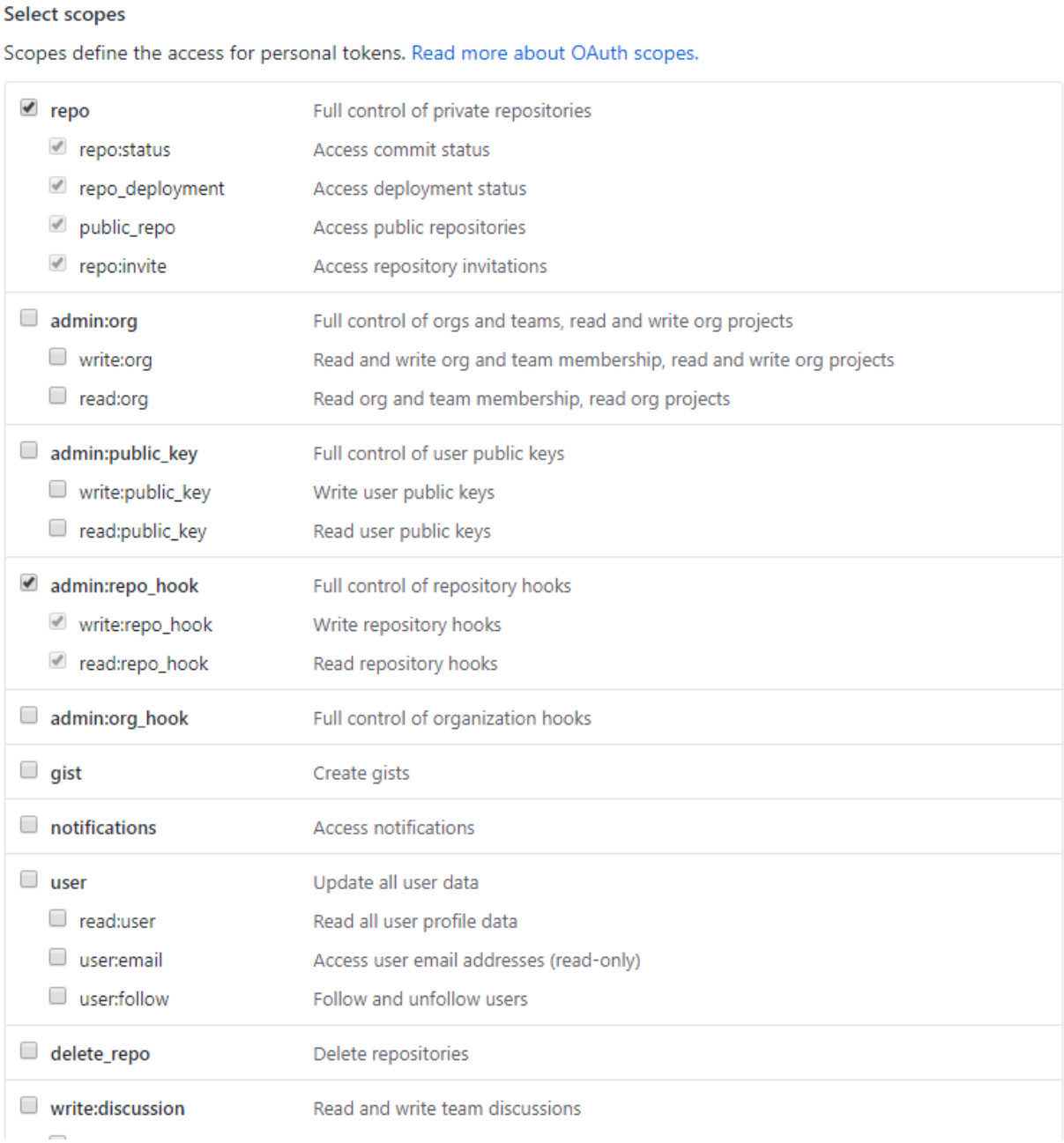


Figure 6-5: Scopes determining access rights for a personal access token

At the bottom of the page, users click at `Generate token` for generating a token with the predefined rights, as can be seen in Figure 6-6.

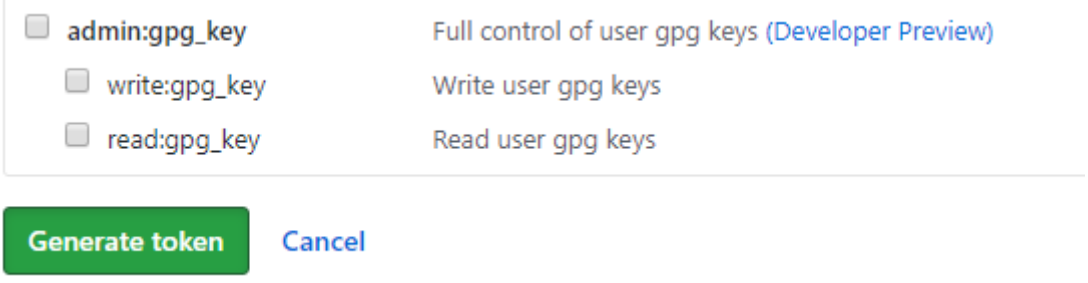


Figure 6-6: Generate token button placement

After clicking `Generate token` button, they get their personal access token, this can be seen in Figure 6-7. As it is mentioned, they need to copy this personal access token somewhere safe, because there is no way to see it again. In the case that they forget personal access token, then the only way is to generate a new token again, but if they do this, they need to change the personal access token everywhere else, because the old access token will be not valid anymore.

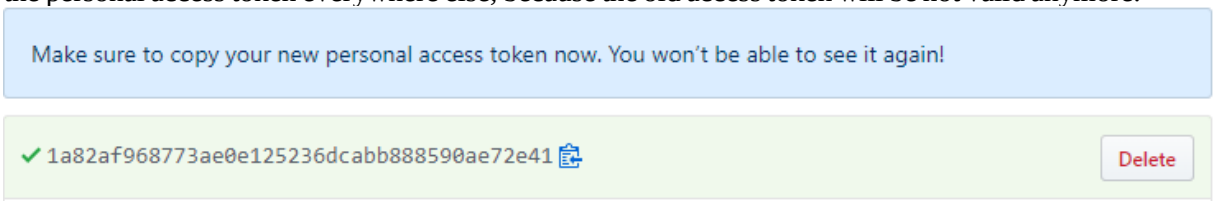


Figure 6-7: Generated personal access token

Users need to encode the newly gained personal access token with the base64 encoding, so for example from `1a82af968773ae0e125236dcabb888590ae72e41` they get `IDFhODJhZjk2ODc3M2FlMGUxMjUyMzZkY2FiYjg4ODU5MGFlNzJlNDE=` personal access token encoded in base64. This operation is done internally, so the user only needs his original personal access token.

Webhooks event notifications are important for a Govis-CI project because without them Govis-CI server would not be informed about new changes, thus being unable to react to any kind of changes in the repository, which means being unable to produce any kind of activity. In users repository they need to go to `Settings`, then choose `Hooks` from options in the left menu and continue with clicking on the `Add webhook` button, as can be seen in Figure 6-8.

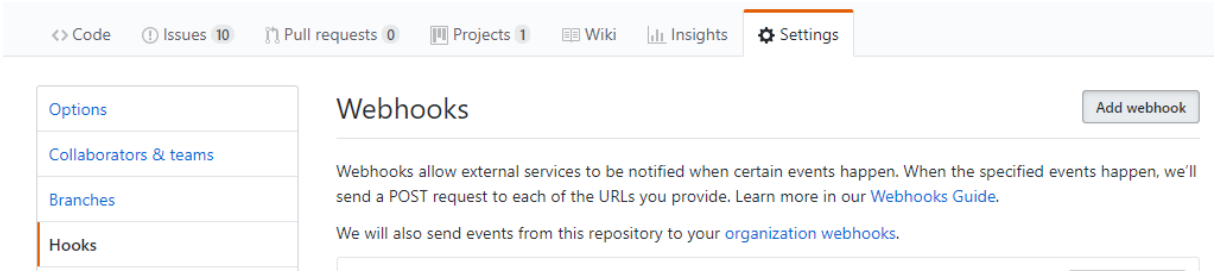


Figure 6-8: Webhooks page on GitHub

Now a webhook is set, so that it will send POST requests when some, predefined events are triggered. Users fill the form, for the `Payload URL` users enter the URL of their Govis-CI server and they need to add `/payload` to the end of URL because they want all requests to be sent there. `Content type` has to be `application/json`, this configuration can be seen in Figure 6-9. Govis-CI was built upon assumption, that content type will be in json format. If there is another format, then the behavior for the Govis-CI server is undefined, but in most cases, it throws an error into the `ServerLog.log` file. Setting `Secret` is optional, but if it is set, then users need to set it in the Govis-CI server too. Typically, it is used for better security and privacy.

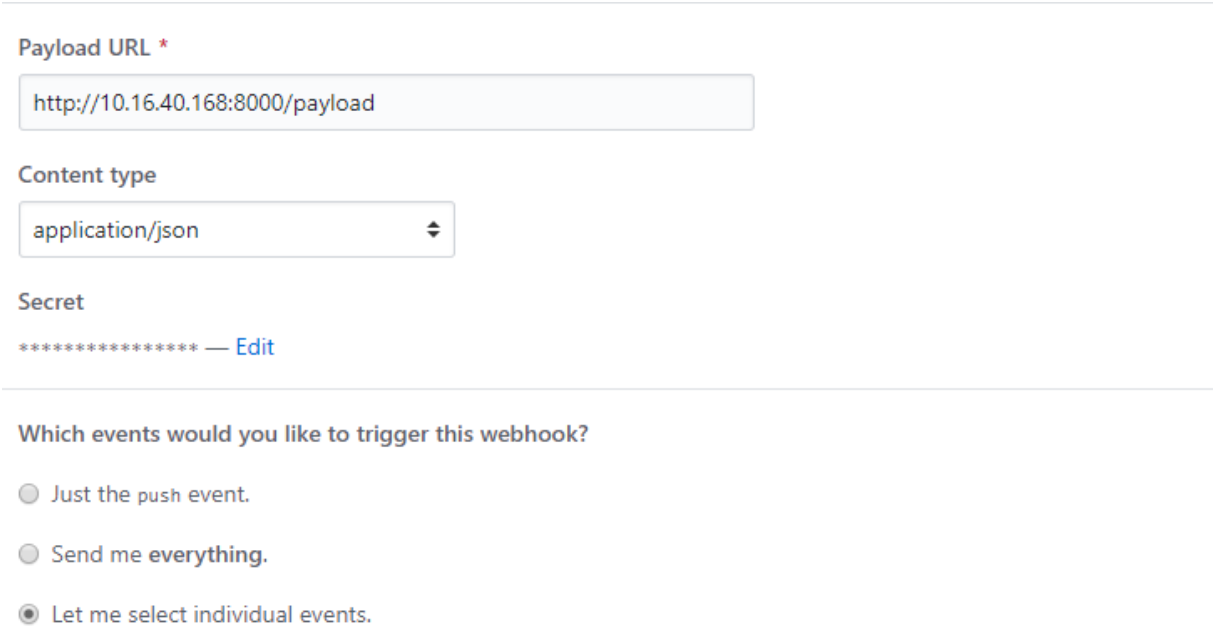


Figure 6-9: Configuration of a webhook

There are a lot of types of events, and the Govis-CI server needs or supports just a few of them. For complete functionality users need to configure webhook to send `Pull requests` and `Pushes` as they can see in Figure 6-10. Additionally, the Govis-CI server supports ping event by simply sending a positive response with status code `200 OK`. If Govis-CI gets an unknown event, then it sends back the response with status code `501 Not Implemented`.

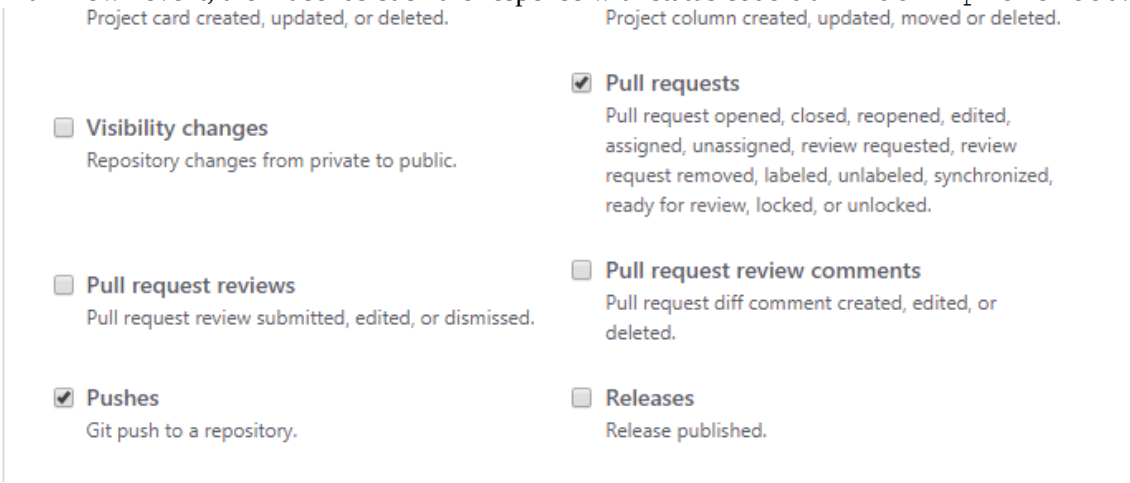


Figure 6-10: Setting of events, that will be sent

Next, users need to mark `Active` checkbox and click on `Add webhook` button, this action can be seen in Figure 6-11.

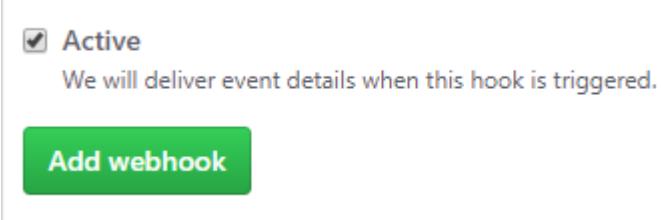


Figure 6-11: Add webhook button placement

And that is everything that users need to configure on GitHub. From now on, the Govis-CI server will get event notifications, in the form of REST API POST requests. Govis-CI server is able to work with these requests and it will get all needed information from it.

### 6.1.2 Environmental variables and CLI

For the correct functionality of the Govis-CI server, users need to specify a configuration for the Govis-CI server which will determine how it will behave. There are two ways to set the configuration for the Govis-CI server. One way is to use CLI (Command-line interface) another way is to use environmental variables. Some data are essential for the Govis-CI server and these are `github personal token`, `rights`, `status_baseurl`. Without that information, Govis-CI server will not work correctly, because:

- `github personal token` is used for verification to GitHub,
- `rights` are used for verification, that Govis-CI server is allowed to run scripts,
- `status_baseurl` is URL that is enchanted of some additional information and sends to GitHub for purpose of pointing to Govis-CI server and additional information helps to point to results of the process, that was executed in order to test code. Without `status_baseurl` user would not know about the address which points to results of the testing process.

Users can set the configuration of the Govis-CI server via CLI. For example, on Windows, they can call executable file: `.\Govis-CI.exe -h`, which will show parameters to users and what they mean. The output from command `.\Govis-CI.exe -h` can be seen in Figure 6-12. We can also use the equivalent `.\Govis-CI.exe --help`.

```
Example: ./Govis-CI
        [--api <api>]
        [--secret <github secret>]
        [--basic_auth <github personal token>]
        [--hostname <hostname/ip>]
        [--status_baseurl <hostname:port/ip:port>]
        [--script_rights <rights>]
        [--secure_mode <mode>]
        [--script_timeout <seconds>]
        [--port <port>]
        [-v]

        <api> == rest api for organization, default: https://api.github.com/
        <github secret>] == WebHook Secret
        <github personal token> == Personal_access_Api_token of user with set status access to the repository
        <hostname/ip> == hostname on which we accept webhooks (0.0.0.0 accepts all and is default)
        <hostname:port/ip:port> == hostname or ip with port number indicating where will user be redirected for results
        <rights> == if unset Govis can't run 'script' from yaml directly
        <mode> == if set to 'negative' then we don't use token for verification with github
        <port> == port to start Govis on - default : '8000'
        <seconds> == number representing time in seconds
        '-v' stands for verbose logging
```

Figure 6-12: Govis-CI help

All needed information can be set with environmental variables. Environmental variables have different names, but they are similar. In most cases is just a prefix `GOVIS_` added and the names are upper case. Environmental variables equivalents are listed below.

```
GOVIS_API -> api
GOVIS_SECRET -> secret
GOVIS_BASIC_AUTH ->basic_auth
GOVIS_HOSTNAME -> hostname
GOVIS_STATUS_BASEURL -> status_baseurl
GOVIS_SCRIPT_RIGHTS -> rights
GOVIS_SECURE_MODE -> secure_mode
GOVIS_LOG -> v
GOVIS_PORT -> port
```

```
GOVIS_SCRIPT_TIMEOUT -> script_timeout
```

In case that users use the combined approach, in other words, they use CLI parameters and environmental variables at the same time, then the Govis-CI server overwrites environmental variables with the provided CLI parameters. So, users can preserve their configuration with environmental variables, and if the need arises then they can use CLI parameters for experimental purposes because they overwrite configuration without losing its original values.

## 6.2 Docker Usage

When running process in Docker, they are run in something called Docker containers. The main source of security isolation. When users are using docker containers and when something happens in a container, it does not affect other processes on the same device, outside of the container. It affects only processes in the same docker container.

Govis-CI contains files named `Dockerfile` and `docker-compose build`, which allows using predefined docker composition. In `Dockerfile` file it is defined what should docker image contain. Essential for using `Dockerfile` file is using `build` word while building image. `Dockerfile` file can be also used by a `docker-compose.yaml` file. `Dockerfile` file was mainly used for downloading sap certificates, exposing port that the Govis-CI server will listen on, running binary or executable Govis-CI files. Of course, there are many more options and things that could be set in the `Dockerfile` file.

In `docker-compose.yaml` file users can set how is an image built, and what kind of environment will be in image after starting the mentioned image. This includes mapping ports, name of the image, name of the container, environmental variables, and other options that docker-compose tool provides. The mentioned functionality of the docker-compose tool is available only if users use `docker-compose` command running docker containers or during the build phase of the docker image. If users want to run the Govis-CI server in Docker, they have two main options, that I recommend.

- The first option is to type `docker-compose up` in the Govis-CI repository. Which will result in the immediate startup of docker on which will be run the Govis-CI server. This option is dependent on the `docker-compose.yaml` file, so users need to have all specifications set there.
- The second option is to type the `docker-compose build`, which will use `docker-compose.yaml` file and `Dockerfile` file resulting in the creation of a docker image, which users can use for deployment. If nothing is changed, then the image is named `sap/govis`. After the creation of image users need to run it. They can simply type `docker run sap/govis` and this will be almost equivalent to the first option. But in the second option users have the ability to add extra changes to the image setting that was created. They can use `-dit` as a parameter and it will result in running docker containers in the background. Users can also use `--restart unless-stopped` for automatic start of a docker container in case that docker container is for some kind of reason shut down. For example, a restart of the server. If users stop the docker container from the command line, then it is stopped and will not start itself. Users can also use `-e <parameter>` argument with the parameter for adding environmental variables to the docker container. An example of code for running a docker container that users build with `docker-compose build` command is visible in Figure 6-13. There were also used some parameters that were mentioned earlier.

```
docker-compose build
docker run -p 8000:8000 `
 -dit --restart unless-stopped `
 -e GOVIS_BASIC_AUTH='e7d90f86deeasdmklb3fc18c76e3a76e324e219a' `
 -e GOVIS_SECRET='secret1' `
 -e GOVIS_API='https://github.wdf.sap.corp/api/v3' `
 -e GOVIS_SCRIPT_RIGHTS='y' `
 -e GOVIS_SECURE_MODE='y' `
 -e GOVIS_STATUS_BASEURL=http://127.0.0.1:8000 `
 -e GOVIS_HOSTNAME=0.0.0.0 `
 -e GOVIS_PORT=8000 `
 -e GOVIS_LOG='y' `
 sap/govis
```

Figure 6-13: Configuration for the docker run command

With the command `docker ps -a` users are able to see all containers. In this case, users should see the container with image `sap/Govis` after executing the `docker-compose build` command. In Figure 6-14 is shown an example of what users should see.

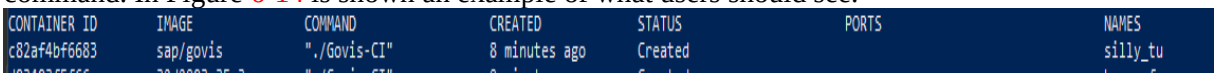


Figure 6-14: Result of `docker ps -a` command

With current settings, users should be able to access the web interface of the Govis-CI server on `localhost:8000`. If users decided to run this on a PC and they restart the PC, then they can see that after some time the docker container will be started automatically and all functionality will be available. When users are finished and they want to stop running Govis-CI server in docker then they just need to type: `docker stop <id of the container from docker ps>`, with this command, the docker container will not start after restart of the PC.

## 6.3 Installation of the additional or needed software

Govis-CI does not provide a user with an interface for choosing the software needed for his builds and automatically installing it. So, the user needs to install needed software manually to his PC or his docker container. For self-testing purposes users will usually need some of the following software: go, curl, libc-dev which can be seen in Figure 6-15. In the mentioned figure users can also see simple steps needed for installing additional software into a docker container based on its container ID. In Figure 6-15 is shown the installation of the development library for C language. Go/Golang is able to incorporate C programming language into its code, for this purpose it needs the mentioned development library. The next thing installed based on the mentioned figure is Go/Golang, and then curl application for testing purposes of Govis-CI.

```
docker exec -it <container ID> sh
apk update
apk add libc-dev
apk add go
apk add curl
```

Figure 6-15: Installation of
additional software into the docker container

During the development of Govis-CI, a need for external libraries occurred. Go/Golang language imports external libraries based on their URL, so all external code used is traceable. In Figure 6-16 are illustrated all external libraries needed for Govis-CI to compile correctly.

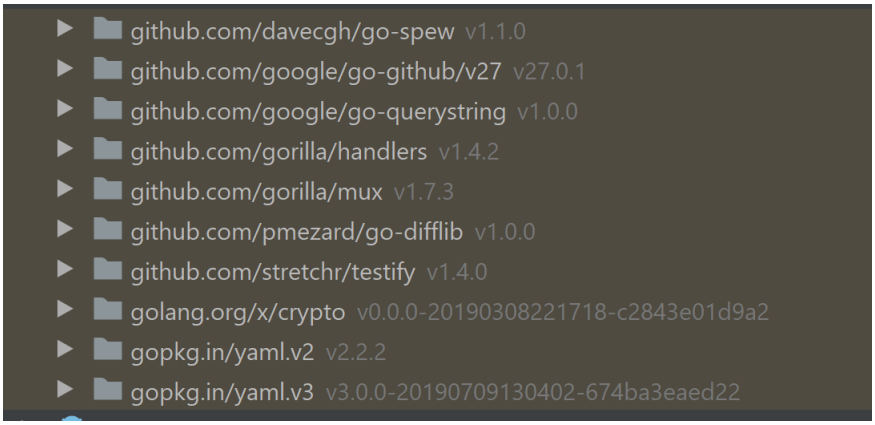


Figure 6-16: External libraries needed for Govis-CI

## 6.4 .govis.yaml file

`.govis.yaml` file is a configuration file in which users are able to describe how the Govis-CI server is supposed to start testing the new version of code. In Figure 6-17 and Figure 6-18, can seen little script parts used for self-testing purposes of Govis-CI during its development.

```
script:
  - go test Govis-CI Govis-CI -v
```

Figure 6-17: Script in testing repository

```
script:
  - go run main1.go
  - go run main2.go
  - go run main3.go
  - go run main4.go
```

Figure 6-18: Self-testing script

For self-testing purposes was used a script from Figure 6-17. From examples showed in Figure 6-17 and Figure 6-18 it can be seen that syntax of `.govis.yaml` file is not going to be hard, but rather simple. All things that users need to declare in `.govis.yaml` file is a descriptor `script:`, and `-` placed on a new line followed with a script that is going to be used in terminal to be used for testing.

The script in Figure 6-18 can be interpreted as follows:

- Govis-CI server will run four commands in a terminal.
- All four of them are used for compiling and running Golang programs.
- Four Golang programs will be run.

## 6.5 Example of Usage

After the initial set-up described in previous sections, users are ready to use the Govis-CI server for testing and automation purposes. For this reason, this section will show what users can expect from the Govis-CI server.

In Figure 6-19 can be seen the repository of testing repository for Govis-CI. This testing repository has configured webhooks as was described in previous subsections. At the same time, there exists an already running Govis-CI server with all needful configuration, which was described in the previous subsection.

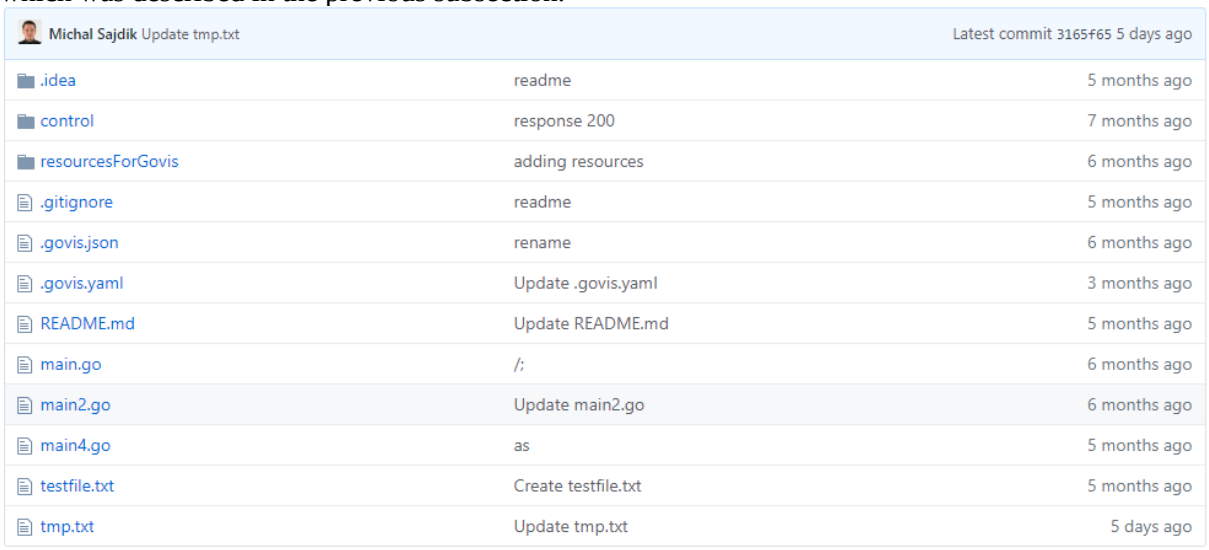


Figure 6-19: The testing repository for Govis-CI

For testing purposes, there is a need to invoke a push event by pushing some changes to it. For example, changing `main4.go` file. The contents of `main4.go` file can be seen in Figure 6-20. After changing `main4.go` file, by committing changes to the repository, by clicking on the `Commit changes` button, as can be seen in Figure 6-21, push event is invoked.

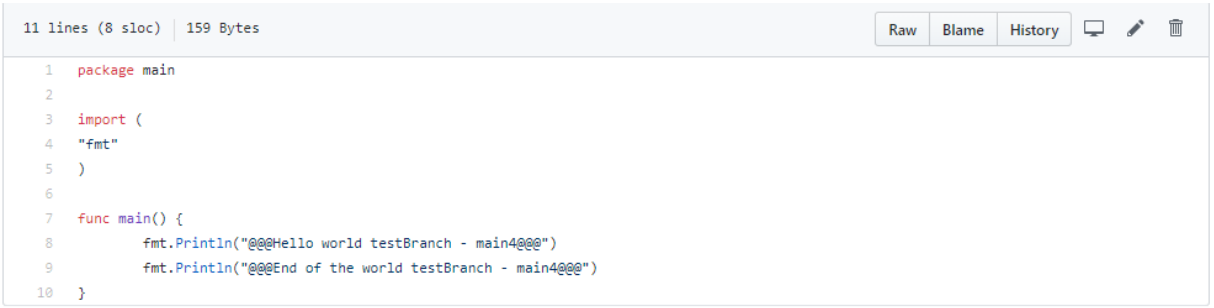


Figure 6-20: `main4.go` file

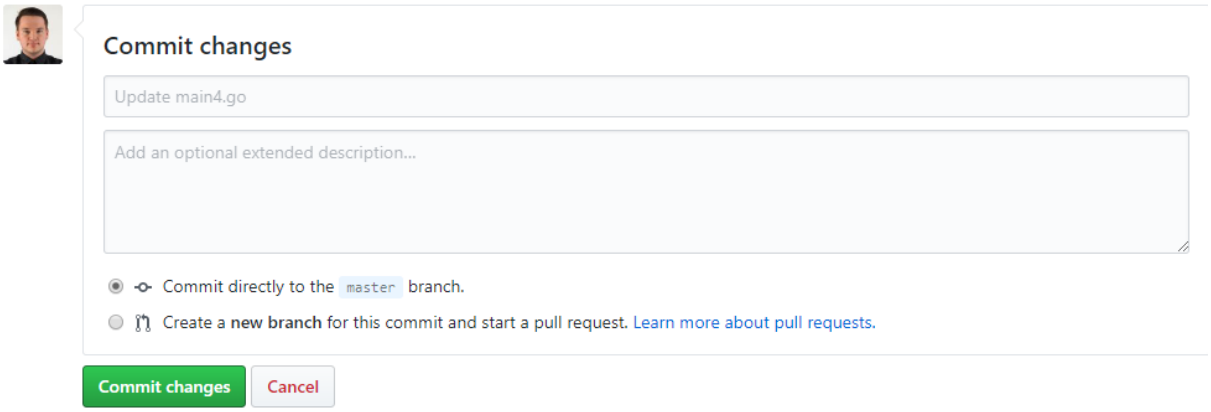


Figure 6-21: Commit changes GUI

After committing new changes two things happen. The first thing is that webhooks gets activated. The other is that the status of the commit is updated in the testing repository, after finishing scripts. When webhooks are activated they send a push event request to the server specified in the field `Payload URL` mentioned in earlier subsections. The request that is sent can be seen in Figure 6-22. it can also be seen that the request has a header and a body, which is named payload. From Govis-CI server-side information that are needed are `content-type`, which is used, and what kind of `X-GitHub-Event` is used. Configuration of the `content-type` was showed in earlier subsections. Govis-CI supports only `application/json` as a content-type value. Govis-CI server checks `X-GitHub-Event` because there is a need to filter only events that are expected. In the case of `X-GitHub-Event` being a pull request event, the Govis-CI server needs to use a different structure for parsing the request payload. In case of an unknown event, the Govis-CI server will send error message, and log it appropriately. In the payload, users can find a lot of useful information. The information that Govis-CI server needs to extract from a webhook request body is the owner of the commit, the name of the repository, the id of the commit, the URL for statuses to be updated on, the URL to the committing repository and to which branch was push commit applied to. Govis-CI server uses this information for determining what repository and what branch it needs to download, so it can run scripts on the correct code from the correct repository. After making sure that all information that it got is right, the Govis-CI server will send a response to the request from GitHub that GitHub receives. GitHub's response can be seen in Figure 6-23. From the mentioned figure it can be seen how long the body is, and its type and time (date) on which was response send. If there is a need to, it is possible to add more information to it. In the body of the response, it is possible to see a report from the Govis-CI server. The report from the Govis-CI server is usually empty in case that everything went right.

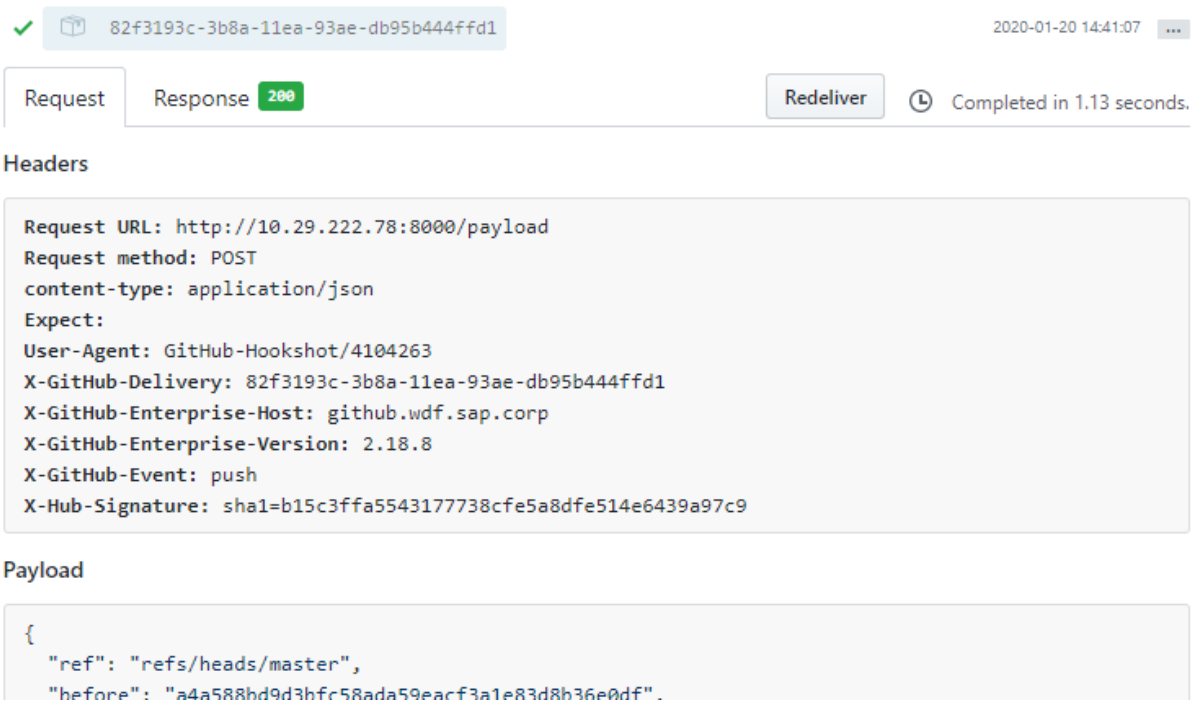


Figure 6-22: Push event request body from GitHub


Figure 6-23: Push event response body from Govis-CI server

After confirmation of information, but before responding to GitHub's request, the Govis-CI server runs a parallel process for the purpose of handling confirmed information. At the start, the Govis-CI server sets the status of the commit to the `pending` state. Govis-CI server continues with parsing of `.govis.yaml` file from which are extracted scripts. If no `.govis.yaml` is found, then the status of the commit is set to `failed`. Extracted scripts are run and logged on the server. After scripts finish running, the Govis-CI server has all the information needed for determining if the status will be set to the `failed` state or the `success` state. Govis-CI server logs results and sends a request to GitHub to change the status of the commit.

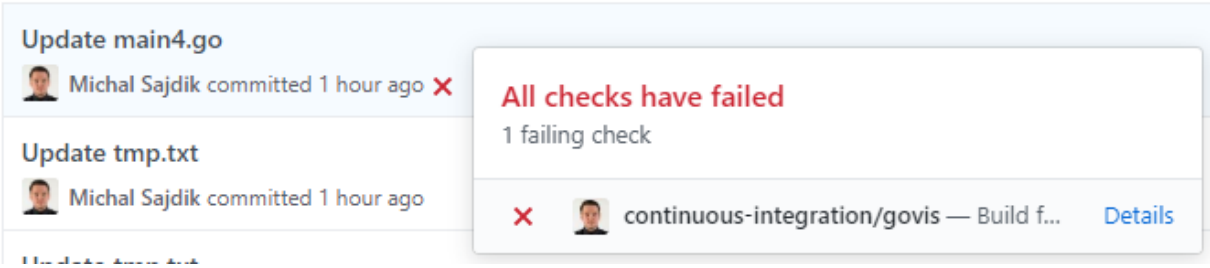


Figure 6-24: Testing failure

In Figure 6-24 it is possible to see how a commit looks when tests fail. By clicking on the `Details` link users are transported to the Govis-CI server website. On the website, they will see logs that belong to the run belonging to the commit, on which `Details` users clicked. An example of logs is shown in Figure 6-25.

```
INFO: 2020/01/20 13:41:58 Repository was cloned >>

INFO: 2020/01/20 13:41:58 Location >> /app/results/testRepositoryForGovis_2020-01-20_13_41_57_+0000_UTC_a51fe18/testRepositoryForGovis
INFO: 2020/01/20 13:41:58 master was checkout-ed >> Your branch is up to date with 'origin/master'.
 >> with command >> git checkout master@at=/app/results/testRepositoryForGovis_2020-01-20_13_41_57_+0000_UTC_a51fe18/testRepositoryForGovis
INFO: 2020/01/20 13:41:58 Govis parsed from .govis.yaml
INFO: 2020/01/20 13:41:58 GovisParse >>  {[go run main4.go] []}
INFO: 2020/01/20 13:41:58 Starting run for >> go run main4.go
INFO: 2020/01/20 13:41:58 App: /bin/sh; Script:'-c'(go run main4.go)'

INFO: 2020/01/20 13:41:58 exit status 127: /bin/sh: go: not found

INFO: 2020/01/20 13:41:58 Result:
INFO: 2020/01/20 13:41:58 #########
INFO: 2020/01/20 13:41:58 All scripts in array finished
```

Figure 6-25: Logs from running scripts from `.govis.yaml` indicating a failure

As users can see, the status is set to the `failed` state. That is why logs are important, because actually in testing repository everything is ok. The problem is in the Govis-CI server, which does not have `go` installed. After installing Golang on the running Govis-CI server, users need to trigger a webhook push event again. In Figure 6-23, it is possible to see a button with a text `Redeliver`. After clicking on the mentioned button, a webhook event is sent again, and the whole process is repeated. The Govis-CI server saves data for all runs. Users can watch the status of a failed commit as it quickly gets evaluated to the `pending` state, as users can observe in Figure 6-26. After running scripts are finished, commit will get the `success` state because everything went without problems. Changes can be seen in Figure 6-27.

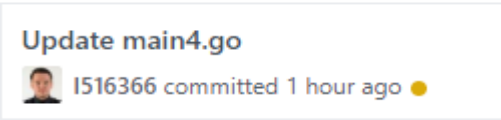


Figure 6-26: Pending state of commit

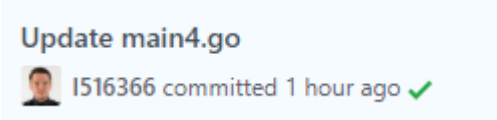


Figure 6-27: Success state of commit

Now, users can check logs from the `success` state, and users can compare how are result logs different. When users compare Figure 6-25 and Figure 6-28, then they can see that in the first picture they got the error, but in the second figure, they got successful output from scripts.

```
INFO: 2020/01/20 14:54:30 Repository was cloned >>

INFO: 2020/01/20 14:54:31 Location >> /app/results/testRepositoryForGovis_2020-01-20_14_54_30_+0000_UTC_a51fe18/testRepositoryForGovis
INFO: 2020/01/20 14:54:31 master was checkout-ed >> Your branch is up to date with 'origin/master'.
 >> with command >> git checkout master@at=/app/results/testRepositoryForGovis_2020-01-20_14_54_30_+0000_UTC_a51fe18/testRepositoryForGovis
INFO: 2020/01/20 14:54:31 Govis parsed from .govis.yaml
INFO: 2020/01/20 14:54:31 GovisParse >>  {[go run main4.go] []}
INFO: 2020/01/20 14:54:31 Starting run for >> go run main4.go
INFO: 2020/01/20 14:54:31 App: /bin/sh; Script:'-c'(go run main4.go)'

INFO: 2020/01/20 14:54:31 <nil>: # command-line-arguments
loadinternal: cannot find runtime/cgo

INFO: 2020/01/20 14:54:31 Result: @@@Hello world testBranch - main4@@@
@@@End of the world testBranch - main4@@@

INFO: 2020/01/20 14:54:31 #########
INFO: 2020/01/20 14:54:31 All scripts in array finished
```

Figure 6-28: Logs from running scripts from `.govis.yaml` indicating success

In Figure 6-29 is shown the Govis-CI server homepage. The webpage contains the following: the time when the webpage was loaded, five indicators for determining if the environment and the basic settings are compliant, the hostname determining on what hostname Govis-CI server listens (if 0.0.0.0, it listens on all), the address for whole address showing on what port Govis-CI server accepts requests (port 8000 by default), API which shows what API is set. In Figure 6-29 can also be seen the GitHub API for SAP (every corporate version of GitHub has its own API form), `Results folder`, `AccessLog.log`, and `ServerLog.log` hypertext links. `Results folder`, `AccessLog.log`, and `ServerLog.log` hypertext links will be mentioned later.

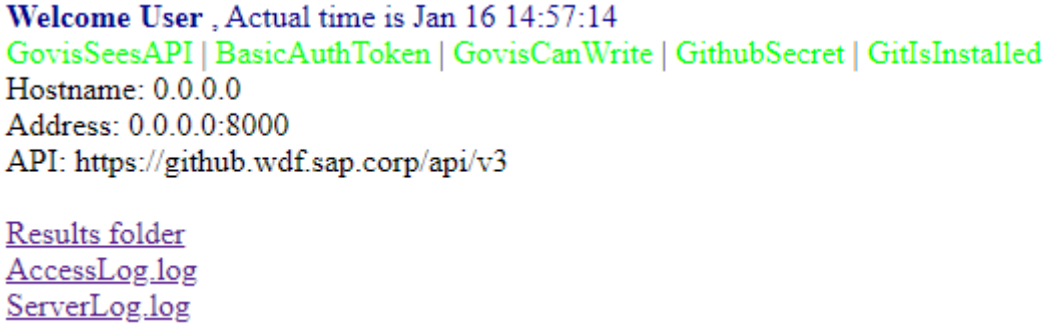


Figure 6-29: Web interface of Govis-CI server

When the five indicators showed in Figure 6-29 are green, then it means that the provided value is valid. Every one of the five indicators is tested separately, which is why the Govis-CI server is able to determine if only one of them is wrongly set. By default, the values of all indicators are red, but after they are verified, they are turned to green one after another. In Figure 6-30 it is possible to see how `GovisSeesAPI` is turned to red when the referred API is not valid. This can easily happen with a simple mistake during everyday work.

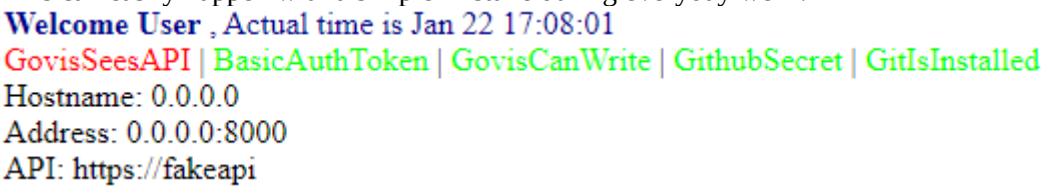


Figure 6-30: Web interface of Govis-CI server with incorrect configuration

When users click on the `Results folder`, they are redirected to results saved on the Govis-CI server. It is possible to see an example view of the results list from the results folder in Figure 6-31. To differentiate between results of different runs, during development the name of each result was divided into three parts: name of a repository, time of execution, determinable starting part of commit ID. From here users can choose a specific result,

that they want. When they click on a chosen result, they are redirected to its content consisting of `result.log` and another redirection to repository files. It is possible to see in Figure 6-32 `result.log`. The `result.log` is a file, where users are redirected from the GitHub page when they want to see more details about status given for a specific commit. The `testRepositoryForGovis/`, which can be seen in Figure 6-32, is a directory for repository files when users click on it, they will be redirected to a directory which contains all files that were downloaded from GitHub. These files were used for testing.

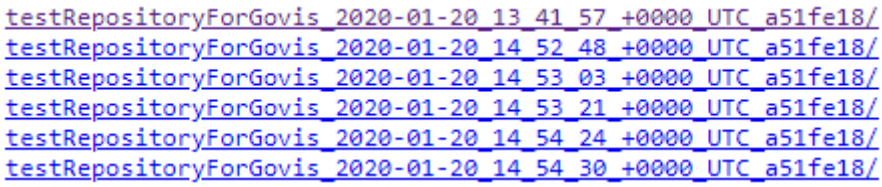


Figure 6-31: Content of `Results folder`

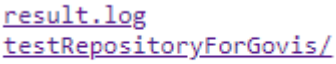


Figure 6-32: Content generated from one run

When users go back to the main page and choose to click on `AccessLog.log`. In this log file, users are able to see all attempts of connection to the Govis-CI server. This function is for security and debug reasons, it does have no effect on the automation of processes. An example of a connection attempt can be seen in Figure 6-33.

```
172.17.0.1 - - [16/Jan/2020:14:57:14 +0000] "GET / HTTP/1.1" 200 2079 "" "Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/79.0.3945.117 Safari/537.36"
```

Figure 6-33: One log from `AccessLog.log` file

When users return to the main page again and click on the `ServerLog.log`, then users are shown contents of the file containing all server logs. These logs mainly consist of information messages used for debugging in the case of some error on the Govis-CI server side. Users can see a simple example in Figure 6-34. Govis-CI server has three types of logs, namely: INFO, WARN, and ERROR. INFO is used for tracking progress and verifying values during a run. WARN is supposed to warn users if some values during a run do look suspicious or if users set some values incorrectly, but the Govis-CI server is still able to somehow work with these little difficulties. Even though there is no guarantee that the Govis-CI server will work as expected. The third type is ERROR, this type of error is printed to console and to the `ServerLog.log` file simultaneously. So, users are able to better notice when such a log appears. The appearance of any ERROR log in almost all cases means, that some fatal mistake was made. In most cases, there is some problem with the initial configuration.

```
INFO: 2020/01/16 14:57:10 @@@Hello world!!!@@@
INFO: 2020/01/16 14:57:10 Listening...
INFO: 2020/01/16 14:57:14 Request for >> /
INFO: 2020/01/16 14:57:14 FileName of web is >> index.html
```

Figure 6-34: Part of `ServerLog.log` file

# 7 Conclusion

In this bachelor thesis were explained Continuous Development principles and its various parts were explained. After the explanation, various possible implementations were described for the mentioned principles. In each section about existing solutions, I also explained why they were not used, and why I needed to create a new implementation. There are also mentioned technologies used for making a Govis-CI server and why these technologies were used. One chapter is dedicated to how Govis-CI was created, and how to integrate a Govis-CI server to a development environment. Integration of the Govis-CI server was described in a form of a partial walkthrough, with an additional explanation, why it was done that way if there was a need to.

During the development of Govis-CI, it was used for testing itself. At the same time, it was the first project at SAP that a Govis-CI server was used on. The Govis-CI server was used on other projects, but it was effective only on small projects. Mainly because small projects did not need automation for setting up a testing environment. This is the main problem, that is planned to be solved in the future development of Govis-CI. As a part of the development process of Govis-CI, with the approval in the process from SAP, and suggestions from colleagues at work, I decided that it would be a good idea for the Govis-CI to become an open-source project. For this purpose section named „How to contribute“ was created in the README.md file of the Govis-CI. This section contains steps for other developers to be able to contribute to Govis-CI development without lowering its quality.

# References


[1] Luetgebrune, J. *Continuous Development: How Iterative Processes Can Improve Your Code* [online]. 2018 [cit. 2020-02-03]. Available: https://deploybot.com/blog/continuous-development.

[2] Synopsys, Inc. *Continuous Development* [online]. [cit. 2020-02-04]. Available: https://www.synopsys.com/glossary/what-is-continuous-development.html.

[3] Pittet, S. *Continuous integration vs. continuous delivery vs. continuous deployment* [online]. [cit. 2020-01-05]. Available: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

[4] TestingNews. *Continuous Testing | Definition, Benefits & How to Perform* [online]. 2019 [cit. 2020-01-06]. Available: https://dev.to/testingnews1/continuous-testing-definition-benefits-how-to-perform-1kio.

[5] Rehkopf, M. *What is Continuous Integration* [online]. [cit. 2019-11-05]. Available: https://www.atlassian.com/continuous-delivery/continuous-integration.

[6] CodeShip. *CONTINUOUS INTEGRATION ESSENTIALS* [online]. [cit. 2019-11-21]. Available: https://codeship.com/continuous-integration-essentials.

[7] Atlassian. *Continuous delivery* [online]. [cit. 2019-12-23]. Available: https://www.atlassian.com/continuous-delivery.

[8] Rouse, M. *continuous deployment* [online]. 2018. [cit. 2019-12-28]. Available: https://searchitoperations.techtarget.com/definition/continuous-deployment.

[9] Scaled Agile, Inc. *Continuous Deployment* [online]. 2019. [cit. 2020-01-04]. Available: https://www.scaledagileframework.com/continuous-deployment/.

[10] Lönn, R. *Continuous Testing: What exactly is it?* [online]. 2015. [cit. 2019-12-22]. Available: https://devops.com/continuous-testing-what-exactly-is-it/.

[11] Guru99. *What is Continuous Testing in DevOps? Definition, Benefits, Tools* [online]. [cit. 2020-01-08]. Available: https://www.guru99.com/continuous-testing.html.

[12] IBM. *Continuous Testing* [online]. [cit. 2020-02-21]. Available: https://www.ibm.com/cloud/learn/continuous-testing.

[13] Jenkins. *Jenkins Handbook* [online]. [cit. 2020-01-18]. Available: https://www.jenkins.io/doc/book/.

[14] Jenkins. *Installing Jenkins* [online]. [cit. 2020-01-18]. Available: https://www.jenkins.io/doc/book/installing/.

[15] Saurabh. *What is Jenkins? | Jenkins For Continuous Integration | Edureka* [online]. 2019. [cit. 2020-01-18]. Available: https://www.edureka.co/blog/what-is-jenkins/.

[16] Leuffen, H.-M. *The Many Problems with Jenkins and Continuous Delivery* [online]. 2017. [cit. 2020-01-19]. Available: https://thenewstack.io/many-problems-jenkins-continuous-delivery/.

[17] Django Stars. *Continuous Integration: CircleCI vs Travis CI vs Jenkins vs Alternatives* [online]. 2017. [cit. 2020-01-25]. Available: https://djangostars.com/blog/continuous-integration-circleci-vs-travisci-vs-jenkins/.

[18] GitLab. *GitLab CI/CD* [online]. [cit. 2020-01-19]. Available: https://docs.gitlab.com/ee/ci/.

[19] GitLab. *GitLab Continuous Integration (CI) & Continuous Delivery (CD)* [online]. [cit. 2020-01-20]. Available: https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/.

[20] Knapsack Pro. *Travis CI vs Gitlab CI* [online]. [cit. 2020-01-22]. Available: https://knapsackpro.com/ci_comparisons/travis-ci/vs/gitlab-ci.

[21] Travis. *Core Concepts for Beginners* [online]. [cit. 2020-01-23]. Available: https://docs.travis-ci.com/user/for-beginners/.

[22] Simper, K. *Review of CircleCI newly launched 2.0* [online]. 2017. [cit. 2020-01-26]. Available: https://medium.com/@kevinsimper/review-of-circleci-newly-launched-2-0-79afe1cf22fd.

[23] Galarza, C. *5 reasons to use Golang* [online]. 2019. [cit. 2019-11-11]. Available: https://dev.to/carloslfu/5-reasons-to-use-golang-3d3h.

[24] Osadchiy, V. *Why Use the Go Language for Your Project?* [online]. [cit. 2019-11-11]. Available: https://yalantis.com/blog/why-use-go/.

[25] Bai, Y. *Best Practices: Why Use Golang For Your Project* [online]. [cit. 2019-11-11]. Available: https://uptech.team/blog/why-use-golang-for-your-project.

[26] RedHat. *What is Docker?*,“ [online]. [cit. 2019-11-12]. Available: https://www.redhat.com/en/topics/containers/what-is-docker.

[27] IBM Cloud Education. *Docker* [online]. 2020. [cit. 2020-02-21]. Available: https://www.ibm.com/cloud/learn/docker.

[28] Red Hat, Inc. *What is Docker?* [online]. [cit. 2019-11-12]. Available: https://opensource.com/resources/what-docker.

[29] Vaughan-Nichols, J.-S. *What is Docker and why is it so darn popular?* [online]. [cit. 2019-11-12]. Available: https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/.

[30] PubNub Inc. *What is ngrok?* [online]. [cit. 2019-11-13]. Available: https://www.pubnub.com/learn/glossary/what-is-ngrok/.

[31] Nash, P. *6 awesome reasons to use ngrok when testing webhooks* [online]. 2015. [cit. 2019-11-13]. Available: https://www.twilio.com/blog/2015/09/6-awesome-reasons-to-use-ngrok-when-testing-webhooks.html.

[32] DrVoIP.com. *WTF is Ngrok?* [online]. 2017. [cit. 2019-11-13]. Available: https://www.drvoip.com/blog/voip-service-solutions/wtf-is-ngrok/.

[33] Kinsta Inc. *What Is GitHub? A Beginner’s Introduction to GitHub* [online]. 2020. [cit. 2020-03-04]. Available: https://kinsta.com/knowledgebase/what-is-github/.

[34] Lee, M. *Opinion: GitHub vs GitLab* [online]. 2018. [cit. 2020-03-05]. Available: https://www.linuxjournal.com/content/opinion-github-vs-gitlab.

[35] GitHub Inc. *Webhooks* [online]. [cit. 2020-03-05]. Available: https://developer.github.com/webhooks/.

# Attachment A:

# Content of attached DVD

Attached DVD contains the following files and folders:

- Govis-CI/ – source code of this bachelor thesis
- Govis-CI.exe – server for continuous integration in the single executable file
- xsajdi00_bp.pdf – an electronic version of this bachelor thesis
- xsajdi00_bp.docx – a source file for electronic version of this bachelor thesis
- pkg.zip – compressed folder of third-party packages needed for Govis-CI compilation

# Attachment B:

# First compilation

For compiling of Govis-CI is needed Go/Golang. Installation steps:

1. Step: This can be installed with the following command from PowerShell (Run as Administrator):

```
Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```
[1]

2. Step: Then you can install Go/Golang by `Chocolatey` with command:

```
choco install -y golang
```

3. Step: Restart PowerShell

4. Step: Type `go version` output from this command should be installed version of Go/Golang on device

5. Step: packages
    a. Add files from pkg.zip to packages used by Govis-CI
    b. `go run .` command in Govis-CI repository will run Govis-CI server and install all needed packages from internet, if they are still available

6. Step: Type `go build` for building executable Govis-CI file

7. Step: Run executable file from 6. Step or optionally add configuration explained in chapter 6.

[1] Source: https://chocolatey.org/courses/installation/installing?method=installing-chocolatey#powershell