Kirill Kubryakov

Deployment and Testing Automation in Web Applications

Implementing DevOps Practices in Production

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

October 3, 2017

Helsinki Metropolia University of Applied Sciences

Author	Kirill Kubryakov
Title	Deployment and Testing Automation in Web Applications
Number of Pages	37 pages
Date	3 October 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software
Instructor(s)	Amir Meirbekov, Software Engineer Peter Hjort, Senior Lecturer

This final year project was carried out for EasyAntiCheat Oy. The main goal of the project was to compare the available solutions for Continuous Integration (CI) and Continuous Delivery (CD), choose the tool that best suits the current project needs, and implement it in practice.

The study is based on the principles of Development Operations (DevOps) and the ideas and core values that stand behind the DevOps movement. Eight popular CI tools were chosen for comparison and their advantages and disadvantages were studied. One option was chosen based on hosting options, configuration options, prices and integration with the git repository manager already used by the case company.

Based on the results, GitLab CI was selected and integrated with the current version control system. During the case study, automation of testing, building and deployment were achieved. One GitLab Docker Runner was setup to run the pipeline, which includes the following stages: install, test, build, and deploy to production. Further development of the project may focus on creating a staging environment and closer integration with the company's messenger application.

Keywords	DevOps, Continuous Integration, Continuous Delivery

Contents

1 Introduction

Creating applications is a complex process that includes several elements: planning, prototyping, design, writing code, testing, fixing errors, continued testing, and finally shipping it into production. In this process there are three independent groups of participants: developers, testers and the operations department. Each of these groups performs its own tasks and uses different criteria to assess the efficacy of its work. For developers, this is the speed of coding and the number of functions implemented in the program code, for testers – the number of errors and bugs detected, for the operation department – the stability of the systems and the minimum failure rate. Such a model of work often leads to a conflict of interests: the first party tries to write the code quickly and send it to the testers for quality assurance, the testers check and test the code as long as necessary to reveal all the bugs, and the third party, the operations department is hesitant to make large changes to the code, because these changes have the potential to open the entire IT infrastructure up to serious risks. As a result, the entire process of creating applications stretches for an unacceptably long time. Because business need to be as flexible and client-oriented as possible, releasing new products and services in a timely manner is imperative to their survival and success.

Instead of shipping one major version of software once or twice per year, software companies are continuously shipping new versions in smaller increments. This is especially relevant in web development where services are provided online and can be updated on a nightly basis. Users want to have new features and bug fixes (which is even more important) as fast as possible. In this fast-paced development environment, continuous integration and continuous deployment need to be seamless.

Existing realities literally require software development to further reduce project implementation time: from the origin of the idea to the release of the finished product, while not degrading its quality. With an enviable periodicity, customers are asking to implement the project “yesterday”, so that someone else cannot copy their idea “today”. A shrinking development timeline coupled with a limited budget causes increased pressure on developers and the DevOps services on which they depend.

The aims of this report are to describe principles and benefits of DevOps, illustrate how DevOps techniques can improve and simplify process of development and deployment, and review which technologies can be used to implement continuous integration and continuous delivery in live development environments.

This thesis contains 5 chapters. The second chapter describes DevOps; what it is, and what are its benefits. The third chapter discusses the DevOps tech stack and the tools which are available to those wishing to apply DevOps practices in their projects. The fourth chapter contains a case study of the integration of DevOps tools at EasyAntiCheat Oy. The final chapter describes results of the case study and potential future improvements to the DevOps integration.

2 Theoretical Background

2.1 DevOps Definition

DevOps is acronym derived from development and operations. DevOps is a practice aimed at the active integration of operations and development engineers, and unification and mutual integration of their work processes during the entire service lifecycle of a software project, from design to development, to production support. In addition to a set of various tools and technologies, DevOps is also a cultural and professional movement aimed at cultural changes within companies and improving mutual understanding and cooperation between the participants of the development process. [1]

The history of DevOps started in 2008 from the talk on “Agile Infrastructure” by Andrew Clay Shafer and Patrick Debois at Agile 2008 conference in Toronto, Canada. Later, Patrick Debois organized technical conference called “Devopsdays” in Ghent, Belgium in 2009. The main topics of Devopsdays are the intersection between software development and IT infrastructure operations. Since then, Devopsdays events have multiplied and the term DevOps was popularized through them. [2]

The main problem that DevOps is aiming to solve is slow and ineffective delivery. The main purpose of DevOps is to reduce the cost of the development process by automating and integrating operations into the general flow that allow developers focus more on development. [3]

DevOps is an approach to the delivery of software based on the Lean and Agile principles [4]. In this approach, all stakeholders (business units, development units, quality assurance units and operating units) work together, organizing the delivery of software and considering feedback from real consumers. The principles of DevOps allow teams to deliver applications in a more efficient way and to base changes and improvements on feedback from real consumers [5, 27].

DevOps applies the principles of Lean development, a concept of project management, based on the desire to eliminate all types of losses. The same as Lean production, DevOps involves each employee in the process of optimizing the business and maximizing customer focus. [6]

DevOps is inspired by Agile, a flexible approach to development. The essence of the Agile can be summarized as follows: development is carried out in short cycles called iterations; at the end of each iteration, the customer receives a valuable application (or part of it) that can be used in business; the development team cooperates with the customer throughout the project; changes in the project are welcomed and quickly included in the work. Each iteration itself is like a small software project and includes all the tasks necessary to produce a minimal increase in functionality: planning, requirements analysis, design, programming, testing and documentation. Although a single iteration is usually not enough to release a new version of the product, it is understood that a flexible software project is ready for release at the end of each iteration. [7]

DevOps is very similar in ideology with Agile. But instead of focusing on the code development, in DevOps we are focused on development of the configuration. Responsibility for configurations is transferred from the team of system engineers to the development team in such way that system engineers have control and confidence that the development team will do everything right.

Usually, in DevOps projects the roles are distributed as follows:

- Developer: write project code, implement feature requests, fix bugs
- Operations: maintain services, deployment, scaling, monitoring, troubleshooting, diagnosis, incident management

Agile helps to improve and simplify development process for developers by adding short iterations and more cooperation between teams. Figure 1 represents flow of agile development process.

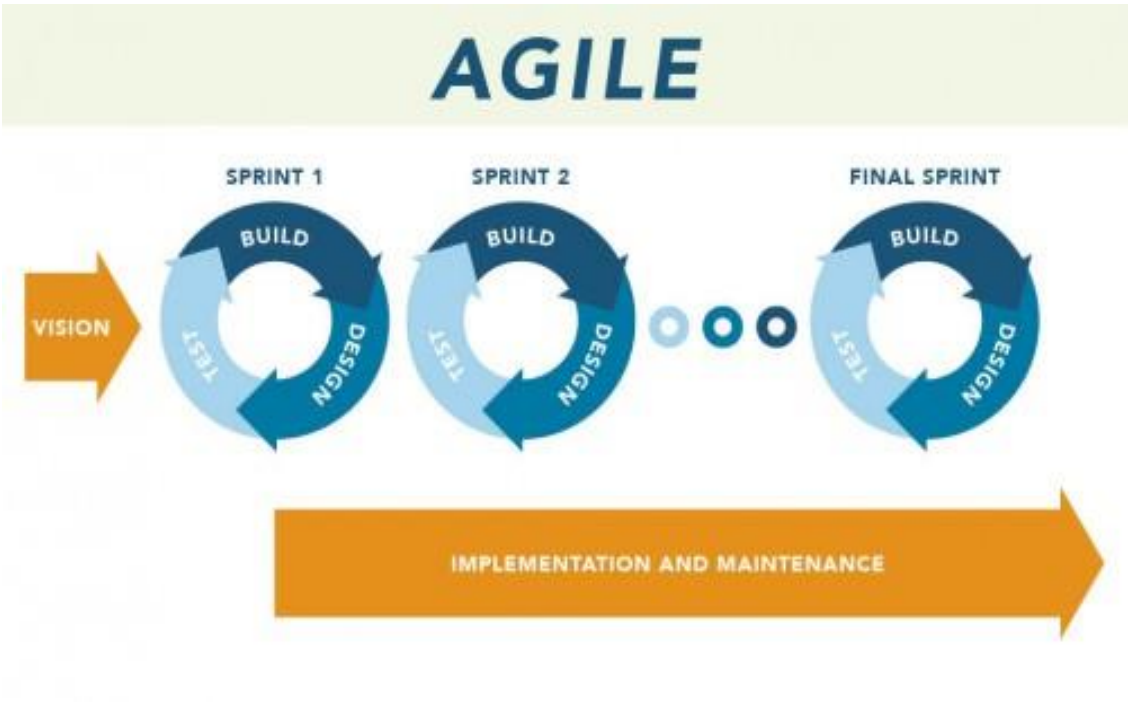

Figure 1. Agile development circle. Reprinted from M&S Consulting. [8].

DevOps' role is to take the workflow, depicted on Figure 1, as a basis, automate testing, deployment and monitoring stages. Basically, DevOps workflow is Agile workflow plus continuous integration and continuous delivery. Figure 2 shows DevOps workflow.

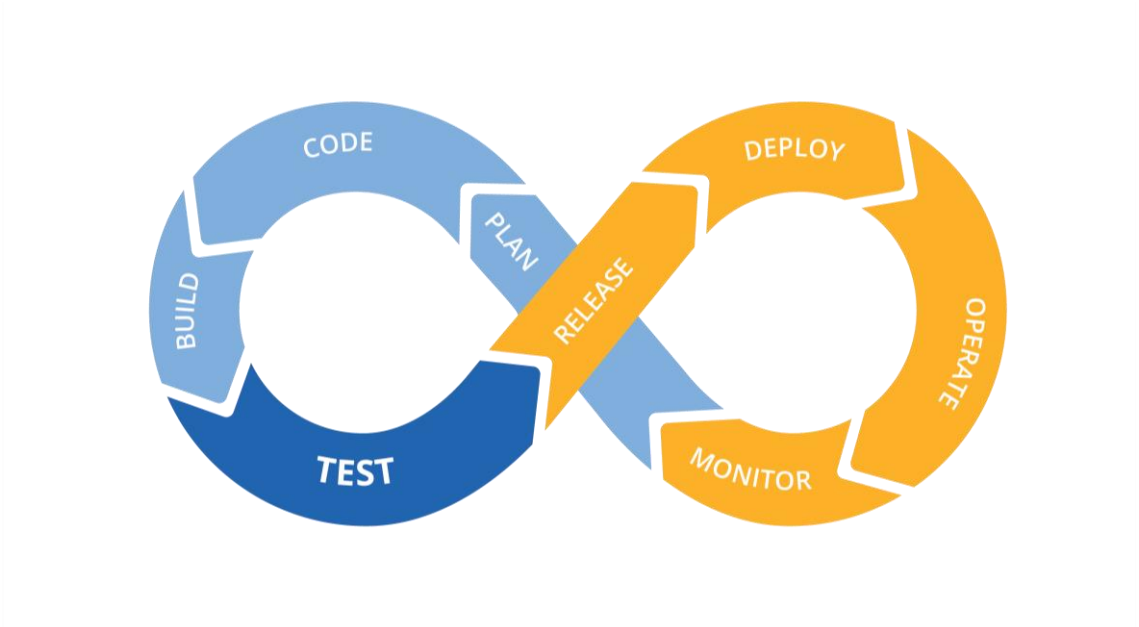

Figure 2. DevOps development flow. Reprinted from Supinfo. [9].

As illustrated in Figure 2, Agile workflow, which is shown in blue, is complemented by continuous integration' and continuous delivery' stages, which are shown in orange. Continuous Integration (CI) is a practice of software development, which consists of the implementation of frequent automated project deployments for the early identification and solution of integration problems. And continuous delivery (CD) is an automated process to deliver a software package to an environment. [10]

2.2 Core Values of DevOps

DevOps' core values are usually described with the CAMS abbreviation. It stands for culture, automation, measurement and sharing. [11]

DevOps breaks barriers between teams, placing focus on people and their interactions above processes and tools. Honesty, openness, and sincerity are the most valuable parts of a DevOps team. It is encouraged to ask questions and to share knowledge, lessons and discoveries. Behavior that helps to optimize processes and understand why existing processes fail, is supported and rewarded. Promoting a safe environment for innovation and productivity is a key challenge for management. [12]

Automation brings the system in order. It helps to intelligently describe the changes that are made to the environment. To understand what should be changed in the product, the team uses constantly collected logs, statistics and feedback. Analysis of the work process is constantly being done, the history of changes is reviewed to understand whether the app is getting better and whether the right direction is chosen. [12]

2.3 Principles of DevOps

The set of DevOps principles can be called "The Three Ways". This name came from Gene Kim, one of the authors of "The DevOps Handbook" and "The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win.". Those three ways are: systems thinking, culture of experimentation and learning and feedback. [13]

Organizations that follow DevOps principles think in systems. And the organization itself is a system, so all the teams in the organization work together to achieve the common goal, to release a reliable software. Developers are focused on the entire process, as

opposed to focusing only on the performance of their own team or department. It means having in mind the full lifecycle of code changes that the team has made, how it can affect on other teams in production pipeline and customers. It includes never passing known bugs down to the other teams, never allowing local optimization to create global degradation, instead focusing on bottlenecks, always seeking to increase the flow and achieve a deeper understanding of the system. [5, 15-16]

Another side of system thinking is that people make mistakes and people should have the right to make mistakes. Instead of blaming someone in case of failure, there should be investigation of the causes. The failure means that the system that allowed those code changes to go in production is not good enough. The system should be improved to catch such cases. [14]

Thinking about software projects from a system-level perspective helps to create a culture of experimentation and learning inside an organization. Experimentation, taking risks without the fear of retribution, learning from failure, gathering feedback from the all channels: between peers in one team, from other teams, from customers, allow to improve the product and provide services that are valuable to its clients.

2.4 Benefits of DevOps

Companies who adopt DevOps practices gain many technical and organizational advantages. Some of these are as follows:

- **Increased stability and quality.** When all members of the team have the same configured development environment, which mocks the actual production environment as close as possible, the integration of continuous testing is expedited. This continuous testing allows for faster and more frequent release cycles and for problems and failures to be more readily identified. [15]

- **Increased organizational effectiveness.** More time is spent to increase the value and quality of the product. [16]

- **Improved customer experience.** The ability to receive continuous feedback and more quickly introduce it into the development of the project leads to increased revenues and increased customer satisfaction. [17]

- **Faster time to market, fast reaction to changing market and customer demands.** The ability to maintain high rates of deployment is transformed into business value in two main ways: how fast can an organization move from an idea to something that can be transferred to the customer and how many experiments the organization can conduct simultaneously. High rates of deployment make it possible to conduct the experiment quickly and almost continuously. [18]

- **Automation.** Simply replacing manual procedures with automated ones makes infrastructure management more efficient. It increases your deployment rate and the ease with which you manage all your resources, either on premises, in the cloud, or in a hybrid environment. Quick response to changing business needs is essential. Automation allows you to scale up or down in response to demand. Automation ensures configuration is consistent across your network, so every developer has the same environment and new developers can easily install work environment and start working. Consistency gives you more control, and control reduces risk. The immediate benefit is that you have a standardized process for provisioning servers. [19, 6]

- **High level of communication between departments and between team members.** The business value is brought to the foreground for all departments. [20]

- **Reduced risk of change.** DevOps reduce the risk of change by making many small, incremental changes instead of fewer large ones. Small changes are easier to review and test. Since the scope of any change is small and generally isolated, it is much easier to fix possible errors or roll back to the previous version. [21]

3 DevOps Tech Stack

3.1 Flow from Development to Production

The adoption of the principles of DevOps changes the habitual project flow. Previously, developers of one team worked for a long time on their features, and all the accumulated changes were combined and sent in one large release. The complexity of the deployment process slowed the delivery of new changes and fixing of existing bugs. The use of the DevOps techniques allows to shorten the release cycle and simplify the delivery of changes. Figure 3 represents DevOps flow from development to production.

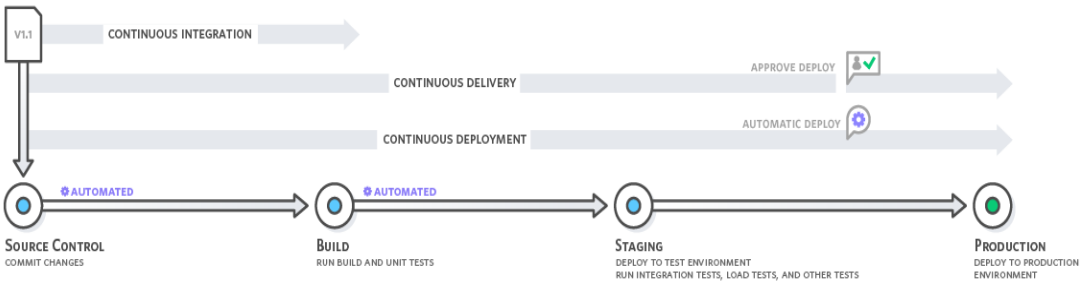

Figure 3. Flow from development to production. Reprinted from Amazon AWS. [23]

As illustrated in Figure 3, DevOps flow contains four stages: source control, build, staging, and production; and three processes: continuous integration, continuous delivery, and continuous deployment. After the developer finishes working with the feature and tests it locally, the feature branch is pushed to the repository, and the process of continuous integration is launched. The project is automatically built and tested by unit tests and any functional errors and integration errors are immediately identified. If this step was successful, the continuous delivery process starts. Continuous delivery extends the practice of continuous integration by ensuring that all code changes after the build phase are deployed in a test environment for feature branches and merge requests. After the successful testing and code review, the feature branch is merged to master branch and changes are deployed to a production environment. [22]

3.2 Continuous Delivery

Continuous Delivery (CD) is an approach to software development in which all changes, including new features, configuration changes, bug fixes and experiments - are delivered to production environment as quickly and safely as possible.

The term “continuous delivery of software” became firmly established in late 2010, after the release of the book of the same name by Jez Humble and David Farley. The book describes fairly simple ideas on how to make the process of delivering software so that it can be released after each commit. [24]

The main idea of continuous delivery is to build a pipeline (Deployment Pipeline), which allows each change in the version control system to get into the production environment in a standard and fully automated way. Hence potentially every change can be deployed into production. [25]

Continuous delivery allows developers to reduce the risks of releases by making software deployment a painless, secure event that can be performed at any time. By automating most operations, such as deployment, environment settings, testing, the time required to release a new functionality decreases.

Of course, it can never be ideal: Configuration Drift, impossibility to make a complete copy of production, different load profile in combat and development, different data in the database on different environments and other factors do not allow for a perfect or painless process. But in most of the cases it is enough to avoid typical problems and automate typical routine operations.

Another important advantage of adopting CD practices is improving the quality of the software. Spending only minutes on automatic tests, the team can spend more time on research testing, usability testing, performance testing, and security.

Any successful software products or services will change significantly during their life cycle. By investing in assembling, testing, deploying and automating the environment, a company reduces the cost of creating and delivering incremental software changes, eliminating many of the fixed costs associated with the software release process. Continuous delivery allows to deliver value to users in small but frequent batches. This allows to get

feedback from users as soon as possible, to test more hypotheses and deliver really valuable software. [26]

If changes are released in small batches, releases become an everyday event for the team. This event no longer causes panic and stress, but rather motivates, giving the opportunity to immediately see the results of their work.

All the advantages that CD gives have a beneficial effect on the development of the company, allowing it to pay more attention to the user's problems, delivering new values as often as necessary, while maintaining a high level of quality. [25]

3.3 Configuration Management

An important advantage of DevOps is the use of the “configuration as code” principle. Developers are trying to create modular applications which maximize composability: such applications are more reliable and are easier to maintain. This principle is also applicable to the infrastructures in which applications are located, encompassing both cloud and local network resources of the company. [27]

Configuration management is a set of methods aimed at systematically considering the changes made by developers in the software product during its development and maintenance. Maintaining the integrity of the system after the changes, preventing unwanted and unpredictable effects, and formalizing the process of making changes are all goals of successful configuration management.

Configuration management answers the question: “Someone has already done something, how can we reproduce this?”. There are two main approaches to answer this question: golden image configuration, or a basic image configuration with an accompanying file containing the environment settings.

Golden image configuration requires more work in the beginning. It is immutable, so it is much faster to install but slow for regeneration. Basic image configuration with an accompanying file is lighter and has lighter build process. The startup process can be really slow, since all the integration is done at startup. Basic images are easier to update because the configuration is just a text file. Those options can be combined by including some dependencies that rarely change to the golden image and placing everything else

in the configuration file. This process can be combined with caching, where only the first startup will require more time and the cache will be updated only if some of the dependencies needs to be updated. [28]

3.4 Continuous Integration

In software development, Continuous Integration means that the product will continuously go through the iteration of the assembly, testing, and demo. In practice, continuous integration systems are implemented in the form of specialized software. Usually a server is allocated to the CI, or a license is purchased for online use.

There are a number of scenarios for using continuous integration systems, but most of them run on a similar line [29]:

- Build a new version of the product
- Testing a new version of the product
- Processing Results

Continuous Integration gives at least one huge plus for the tester. This is automation of regression testing. With a large project, there is a need to constantly run the same scopes of tests to be sure of the quality of the new code. This kind of testing slowly begins to cause disgust due to monotony and repetition. With CI tool, those tests will be done automatically on the CI server. Continuous Integration tool can be integrated with version control service used by a company. [29] Figures 4 and 5 depict examples of output from integrated CI tool.

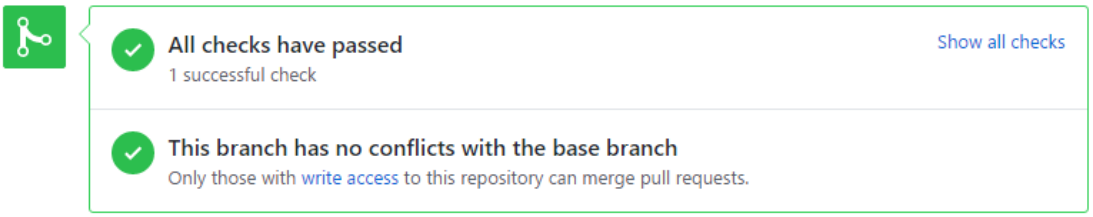

Figure 4. CI tool: example of successful message.

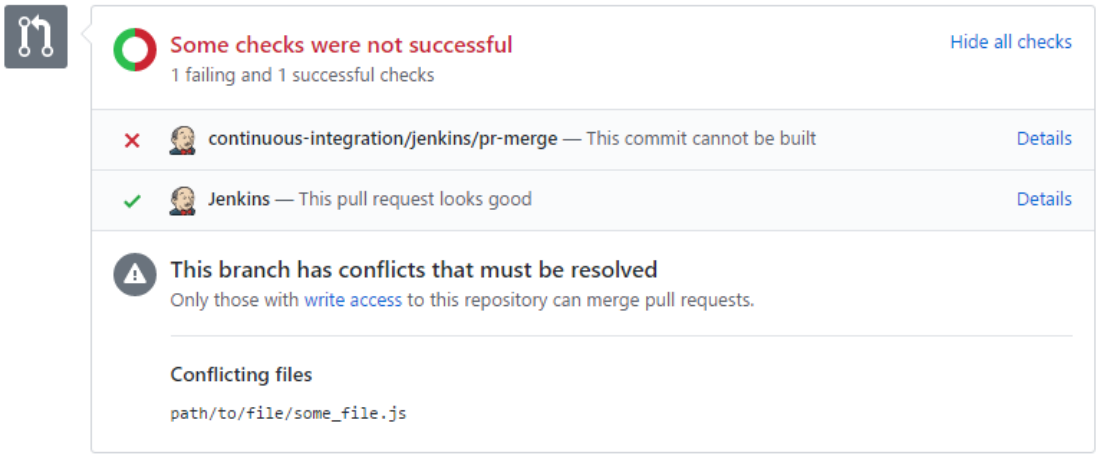

Figure 5. CI tool: example of fail message.

Figure 4 shows successful message and Figure 5 – fail message. With CI tool, it will be clearly visible in version control service interface whether all checks have passed or were not successful and need to be fixed. Various Continuous Integration tools exist in the market. The following are top 8 CI tools according to Vladimir Pecanac [30].

3.4.1 Jenkins

Jenkins is a leading open source self-hosted CI server. It was firstly developed as Hudson, but then forked and renamed to Jenkins after Oracle acquired Sun Microsystems, including the rights to the trademarked name “Hudson” in 2011 [31].

It is written in Java, so it is portable on all major platforms. It has huge community and more than a thousand plugins available from Plugins Index [32]. Jenkins can be configured from web-based graphical user interface (GUI) or through console commands. Jenkins is a powerful and flexible system for automating tasks. It is used mainly as an automation framework, and most of the logic is implemented through libraries and plugins. Everything - from listening to webhooks and viewing repositories to creating an environment and supporting different programming languages - is handled by plugins. The process of continuous integration can depend on numerous third-party plugins, which on one hand provides flexibility, on the other - makes the environment too fragile. [33]

3.4.2 TeamCity

TeamCity is CI tool distributed by JetBrains. It is commercial software, but there is a free version available which includes all the features with limitations (20 configurations and 3 build agents) [34].

TeamCity includes such features as [35]:

- Pre-testing the code before committing. This prevents the possibility of committing program code containing errors that violates the normal assembly of the project, by remotely assembling the changes before the commit.
- The ability to produce multiple project builds at the same time, testing on different platforms and in different software environments.
- Integration with systems for evaluating code coverage, code inspection and code sniffing.
- Integration with popular development environments, such as Eclipse, IntelliJ IDEA, Visual Studio.
- Instant notifications of build errors.
- Ability to run the build and test the changed code without committing to the version control system, directly from the IDE.
- Managing shared resources, allowing you to easily limit access to shared databases and test devices.
- Configurable conditions for dropping a build based on multiple metrics (the number of dropped tests, the number of uncovered classes and modules, and metrics that exclude degradation of code quality).
- Functionality to keep the server in an appropriate shape: built-in clean-up of assembly history, disk space reports and server health reports.

3.4.3 Travis CI

Travis CI is continuous integration tool for GitHub hosted projects. It has free version for open source projects and paid version for private projects. The advantage of the Travis CI is built-in support of Build matrix. Build matrix is a tool that makes it possible to perform tests using different versions of languages and dependencies. It has rich customization capabilities. If the project is wanted to be tested also with some development or alpha versions of packages, it is possible to set up build matrix in such way, that if there will be

a failure for such environments, the warning will be thrown, but the entire build will be not considered unsuccessful. [36]

3.4.4 GoCD

GoCD is a free of charge, but with paid support, continuous integration tool by Thought-Works. What distinguishes this CI tool from all others, is different implementation of pipelines. What is called pipeline in others CI tools, in GoCD is the equivalent of a single task. [37] Figure 6 represents the GoCD pipelines.

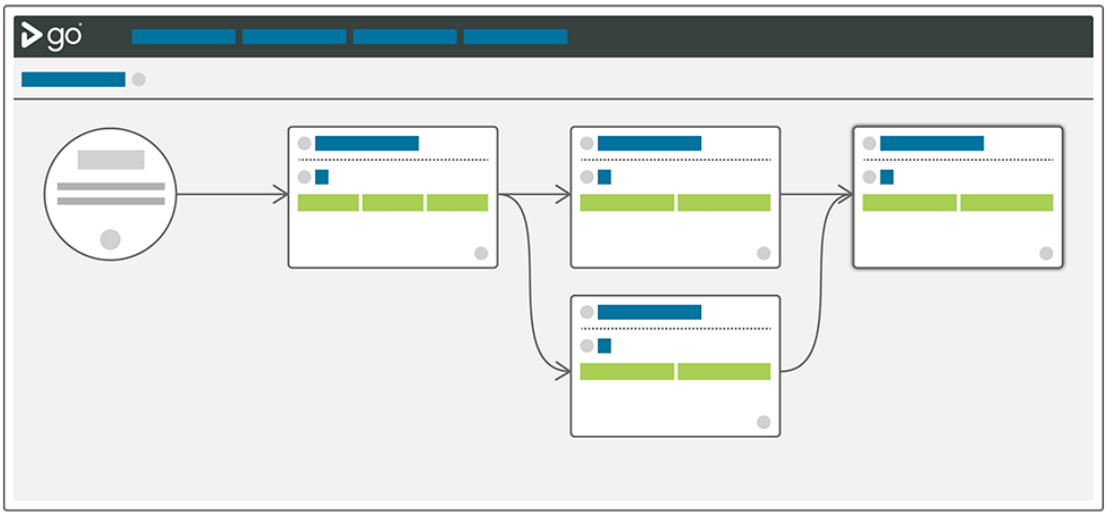

Figure 6. Go CD pipelines. Reprinted from Go. [38].

As shown in Figure 6, each box is an entire pipeline that usually is chained and run in parallel with other pipelines. Furthermore, each pipeline can contain multiple sequential stages that can contain multiple parallel jobs which also can contain multiple sequential tasks. In other words, GoCD brings more level of abstraction and simplifies configuration process for complex projects. [37]

3.4.5 Bamboo

Bamboo is a paid (with free 30 days trial), self-hosted, continuous integration tool by Atlassian. It has great native support for other Atlassian products, JIRA project manager and Bitbucket git repository manager. But it also can be integrated with other git repository managers, such as GitHub or GitLab. [39]

3.4.6 GitLab CI

GitLab CI is a tool for continuous integration built into GitLab, a platform for hosting repositories and git development tools. Initially released as an independent project, GitLab CI was integrated into the main GitLab software with the release of GitLab 8.0 in September 2015 [40]. It can be used both when using gitlab.com website or self-hosted GitLab Community Edition or Enterprise Edition.

The configuration of GitLab CI is defined inside the file in the code repository itself using the YAML, human-readable data serialization format [41]. It supports multiple stages, manual deploys, environments, and variables. Then the work is sent to machines called runners, which are easy to configure and can use different operating systems. Each build can be divided into several jobs that run in parallel on several machines. When configuring runner servers, it is possible to select Docker, shell, VirtualBox, or Kubernetes to determine how tasks are run. [42]

The close connection of GitLab CI with the GitLab platform has certain implications for the use of the software. GitLab CI is not suitable for developers who use other platforms to host repositories. On the one hand, the integrated functionality allows GitLab users to customize the CI/CD environment without installing additional tools and provide a complete solution for the organization of the workflow out of the box. Figure 7 depicts workflow with GitLab CI.

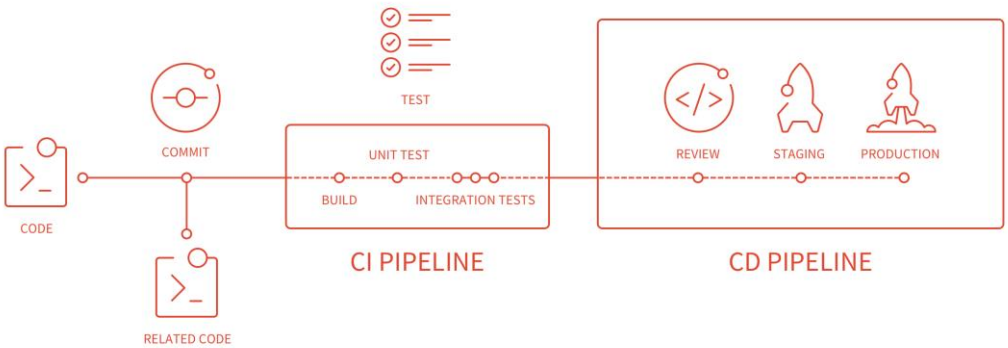

Figure 7. Workflow with GitLab CI. Reprinted from GitLab Documentation. [43]

As illustrated in Figure 7, GitLab CI is closely integrated in GitLab platform and forms a complete solution for development process. A close relationship also allows you to distribute runner-servers between your projects, automatically view the current status of the release in the repository, and store the artifacts of the builds with the code that created them. [42]

3.4.7 CircleCI

CircleCI is a fully ready continuous integration solution that requires minimal configuration. It is a cloud based tool that has free (only one container instance is available) and paid versions. CircleCI can be integrated with GitHub or BitBucket. With CircleCI, it is possible to access projects, builds, and artifacts using the REST API. CircleCI caches third-party dependencies, which helps to avoid the constant installation of necessary environments. The artifacts are stored on the CI servers and it is possible to access them via SSH (Secure Shell). [44]

3.4.8 Codeship

Codeship is another hosted continuous integration tool, similar with CircleCI and Travis CI. It supports GitHub and BitBucket repositories [44]. It has only basic setup, no encryption of secret variables and Docker support only in Pro version. There are Pro and Basic paid plans and a free plan with up to 5 private projects and up to 100 builds per month [45].

3.5 Continuous Testing

If a company in the process of software development is guided by the principles of TDD (Test Driven Development) or at least uses tests to cover the main functionality of the application, most of these tests can be automated. With the help of CI tool, it is possible to automate unit tests, component tests, system tests, functional acceptance tests. However, usability and exploratory testing remain manually implemented. [46]

Automated testing provides several big advantages to developers. CI tools will run tests in the background in a different environment, so team can focus on other tasks. When working with some feature, the developer can overlook that this feature breaks some

other part of the project and by mistake submit this toxic code. With test automation, all tests suites will be launched at the time of the push to the repository and if any of the tests fail, the developer will receive a notification and be able to fix it before it merged and deployed to production. [47]

3.6 Current Tech Stack

The case organization of this study is EasyAntiCheat Oy. The company provides a holistic anti–cheat service for computer games, using hybrid anti–cheat mechanisms to counter hacking and cheating. The author of this thesis works in the company as a web developer in the web development team. The responsibilities include development and maintenance of frontend for the internal and customer portals and dashboards.

The Docker container platform is used to run the project both in production and locally for development. The project has a main image from which the new instance can be started. The consistency of the environment for developers and for production is maintained. Therefore, it can be said that the principles of DevOps configuration management are applied.

Nevertheless, most of the tasks during the process of development and deployment are made manually. The current development workflow includes the following steps:

- First, the developer implements some new feature.
- Next, he runs the tests locally to make sure that everything is working correctly.
- Then, he pushed new code to the repository and makes a merge request to the master branch.
- Another developer does the code review, pull the code locally, run tests and checks the new feature with the local development server.
- After that he accepts the merge request and pulls updated master branch.
- Then he runs the command to build Webpack bundles.
- After the bundles are built, they are sent to the production server and copied to the production container.
- And the last step, Docker container is restarted, and new release is complete.

This whole process takes a long time to proceed and has a number of project-specific aspects. Therefore, there is a need to start using a continuous integration tool to automate the process of testing and deployment. It is planned to set up the CI and transfer all scripts there so any member of the team can easily make the release.

3.7 Chosen Architecture and Technologies

The company security policy requires a self-hosted version of any CI tool. Also, since the GitLab git repository manager is used for version control, the second requirement for the continuous integration tool is to be compatible with GitLab. Considering these two requirements, Travis CI, GoCD, CircleCI, Codeship and Bamboo can be excluded from the list. Jenkins, TeamCity and GitLab CI are remaining. Jenkins is a free, open-source and highly customizable CI tool with considerable number of plugins available. Moreover, it is already used in company by other departments. TeamCity provides helpful metrics, pre-testing before committing, shared resource management for testing and deep integration with many IDEs, but it is not entirely free, the free plan is limited and most of the team uses text editors as Atom or Visual Studio Code instead of IDEs. GitLab CI is already built-in in GitLab and does not have any additional costs. Even though it does not have all the configuration options that Jenkins has, it has all the features that will be needed for integration will the current project setup. Finally, it has a clear and understandable user interface.

GitLab CI was chosen as the most suitable tool for the project needs continuous integration tool and the next chapter covers the process of integration of GitLab CI with the project.

4 Implementation

4.1 Integration with GitLab, Test Automation

To set up GitLab CI, GitLab runner is needed. GitLab Runner is a tool build by GitLab which is used to run user jobs. It is preferable to install GitLab Runner on a separate server, not on the same one on which the GitLab itself is running. GitLab Runner can be installed and used on all most popular operating systems (GNU/Linux, macOS, FreeBSD and Windows) and also can be installed using Docker. It was decided to use Docker.

To install a runner, Docker should be installed, and gitlab-runner container should be mounted from Docker Hub. It is a multi runner, that will be used to register private runners and store their configuration. The installation was done by establishing SSH connection to the server, that will be used for continuous integration, and running Docker console commands [48]. After the necessary software is installed, the private runner can be registered. GitLab Runner utility provides a console command, that can be used to register a new runner [49]. Figure 8 depicts the process of registering a new GitLab Runner using console.

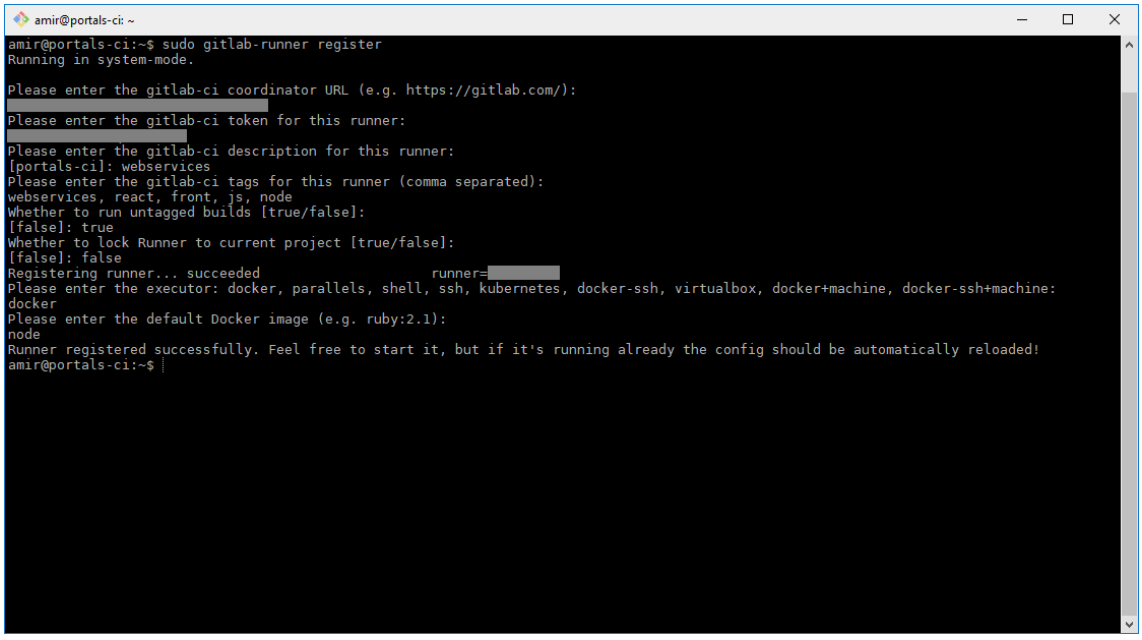

Figure 8. GitLab Runner setup.

As shown in Figure 8, after running command, user is asked for GitLab instance URL (Uniform Resource Locator), GitLab token, runner description and some other project specific parameters through command line interface to make a basic setup.

Since the company's GitLab instance is hosted locally, its IP address is needed to be added to extra hosts. It is possible to modify runner configuration and add additional parameters after registration. Configuration for all registered runners is located in `config.toml` file and can be viewed and modified. Figure 9 represents an opened GitLab runner configuration.

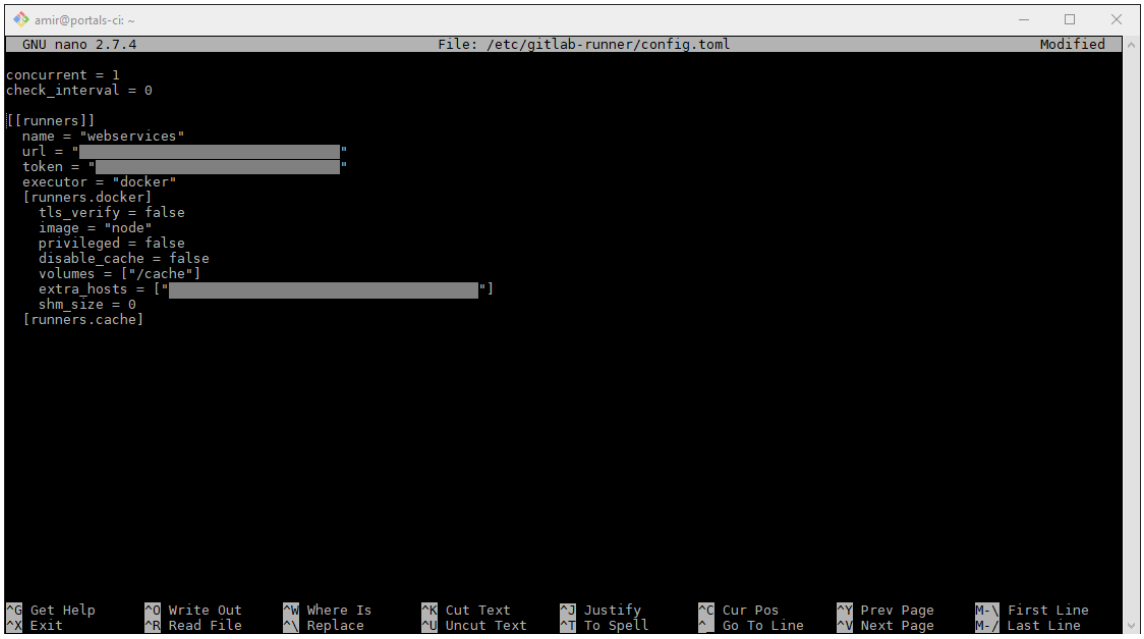

Figure 9. GitLab runner configuration.

As shown in Figure 9, there are one runner registered now and using GNU nano text editor [50], which is usually built-in in Linux distributions, new parameter `extra_hosts` was added.

Then the SSH connection can be closed and all the following steps can be done using GitLab web interface. The runner is available in settings under the Pipeline tab. Figure 10 represents how the registered runner is shown in GitLab interface on GitLab pipelines configuration page.

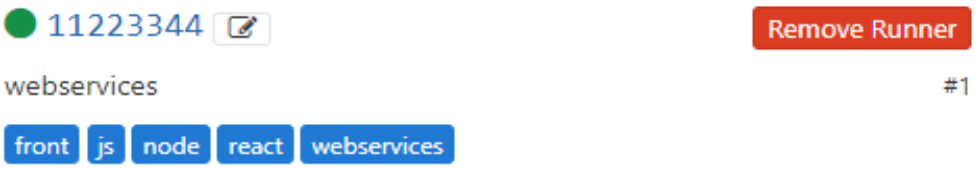

Figure 10. Registered runner on GitLab pipelines configuration page.

As can be seen in Figure 10, the status of the runner is indicated by the color circle before the runner id. Green circle means that runner is active. If for some reason the circle is red, it means that the runner is offline. In such cases it is needed to check whether the CI server is up and running and the GitLab Runner process is running. It is also possible to assign tags to the runner to make clear for which jobs this runner can be used.

When there is at least one runner and it is active, the pipeline for the project can be established. The optimal way to do it is to create a new branch from the master branch to do all the setup, commit changes, check if everything works properly and then make a merge request back to the master branch. It can be named `gitlab-ci`. One of the advantages of GitLab CI is that only one file with configuration is needed to start the pipeline and run jobs. It is possible have different configuration files in different branches and what is most important, those changes will not affect the old branches. The configuration file should be placed in the root directory of the project and named `.gitlab-ci.yml` [51]. The configuration file is in YAML. This file should include all the pipeline. Figure 11 shows GitLab file editor interface with `.gitlab-ci.yml` file opened. It has simple templates for different environments, so it is easy to pick one as starting point.

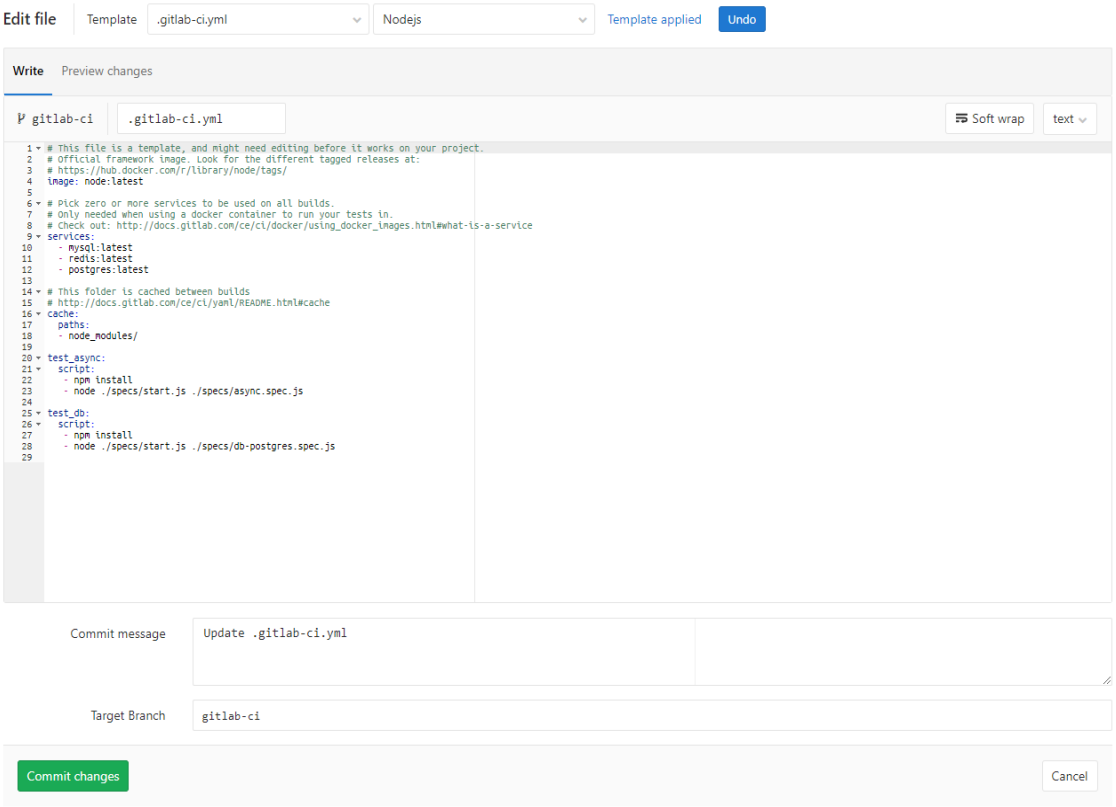

Figure 11. GitLab CI config initialization, with built-in templates for different environments. Nodejs template.

Listing 1 presents a simple pipeline configuration file that was added to the project. It has one job, which is to go to the frontend folder, install all the dependencies and run unit tests. The commands are run using Yarn package manager [52] command line interface [53] which is used in the project to manage dependencies and run predefined scripts.

```
image:
  node

stages:
  - test

test:
  stage: test
  script:
    - cd front
    - yarn install
    - yarn test
```

Listing 1. GitLab CI initial configuration.

When the changes are committed and pushed to the GitLab, the new job with “running” status will appear on Pipelines page. The runner pulled all the code for current branch and started the job. Figure 12 shows GitLab CI job with running status.

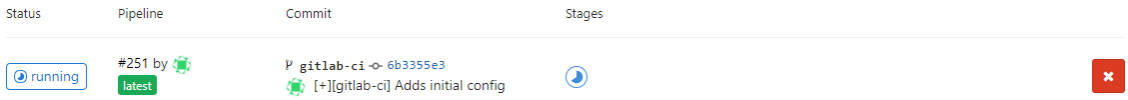

Figure 12. GitLab CI job with running status.

There is also the possibility to see console output by clicking on job status. When the runner finished, it sends a response back to GitLab and the job status changes to “passed”, which means that the job completed successfully. Figure 13 depicts GitLab CI job with passed status.

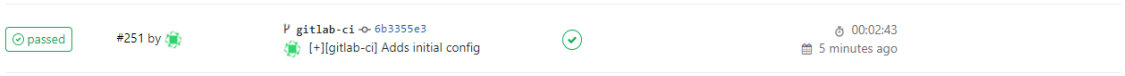

Figure 13. GitLab CI job with passed status.

GitLab Continuous Integration is fully integrated with GitLab. And if any branch or commit has pipeline configuration and requires to run any jobs, it will be indicated with a small icon every time the branch or commit is mentioned in the GitLab interface. Figure 14 shows small indication of pipeline status, that can be visible from branch page.

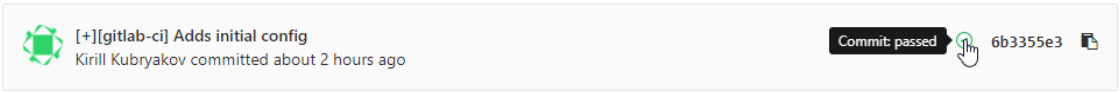

Figure 14. Indication of pipeline status on a branch page.

After every new push to the repository, the runner pulls new code and runs scripts. The next step is to configure stages and add other important parameters to the configuration. First, caching should be added. Since the project has a lot of dependencies, the process of downloading all the dependencies can be very time consuming. It is not effective to download them every time the new pipeline starts. There should be a way to download the dependencies only once and then update if necessary, the same way as in the local

environment during development. GitLab configuration has a cache option for this purpose [54]. The project dependencies are located in node_modules folder. To add this folder to cache, path to the folder should be added to cache paths. Listing 2 shows how to add node_modules folder to runner's cache.

```
cache:
  paths:
    - front/node_modules/
```

Listing 2. Adding cache to GitLab CI configuration.

If the same scripts are needed to be run before or after every job, there are `before_script` and `after_script` GitLab options [54]. The scripts that written in `before_script`, will be run before all jobs, including deploy jobs, but after the restoration of artifacts. The scripts from `after_script` will be run after every job. The project does not require any scripts to be run after the jobs, but there is one command that should be run before each job. Listing 3 shows setup of `before_script` for the project.

```
before_script:
  - cd front
```

Listing 3. Adding before_script option to GitLab CI configuration.

As presented in Listing 3, the command to go to the frontend folder were moved from test stage script to `before_script`, because all the frontend tasks are run from this directory.

4.2 Deployment Automation

The setup for automated testing is ready. The next step is to set up Continuous Delivery. This process includes `build` and `deploy` stages. Also, to make the pipeline clear, the `test` stage from the previous configuration is divided into `install` and `test` stages. The pipeline can be described as follows: install or update dependencies, then if the

dependencies were installed successfully run tests, then if all tests passed build production bundles, then ship those bundles to production server. In GitLab configuration, jobs and stages can be named differently and stage can have multiple jobs, which are run in parallel. Listing 4 shows new `install` stage, which has `yarn-install` job and new `build` stage.

```
stages:
  - install
  - test
  - build

yarn-install:
  stage: install
  script:
    - yarn install

test:
  stage: test
  script:
    - yarn test

build:
  stage: build
  script:
    - yarn build
  artifacts:
    paths:
      - dist/
```

Listing 4. New install and build stages.

As shown in Figure 4, `build` stage contains `artifacts` option. When building bundles, a new file for every bundle is generated and saved to the `dist` folder. Then those files should be transferred to the server. All the files generated during the job are accessible only inside this job. To make them accessible from outside the job, path to the files or list of file paths are specified under the GitLab `artifacts` option. The artifacts will be available to all later jobs and also for download on the Pipelines page. After pushing new changes of the configuration file to GitLab, the pipeline status of the branch updates. Figure 15 shows updated representation of the pipeline.

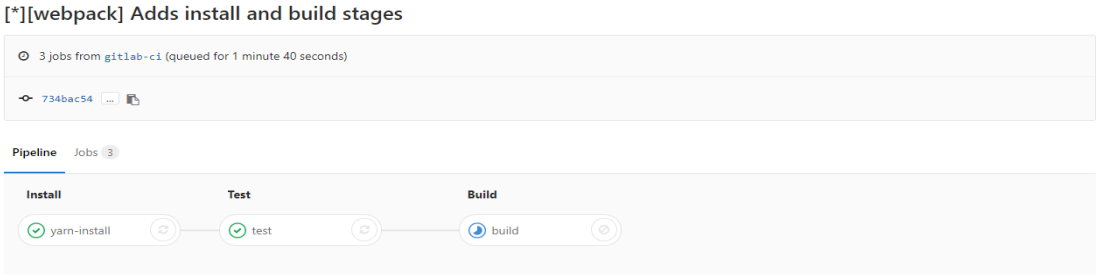

Figure 15. GitLab pipeline interface.

As presented in Figure 15, now GitLab pipeline includes three stages in order that was described previously in `stages` parameter in Listing 4.

The production servers are run using Amazon Web Services. The project backend is written in Python and it uses `awscli` Python tool to connect to the Amazon instance. The same tool is used to transfer frontend production bundles from artifacts folder to Amazon instance. For this job, a different image should be used, because Python is not available in the `node` image. It is possible to specify different image in the GitLab configuration. The `image` parameter with a new image name should be added under the job. Listing 5 shows `deploy` stage configuration.

```
stages:
  - install
  - test
  - build
  - deploy

amazon-deploy:
  image: python
  stage: deploy
  script:
    - pip install awscli
    - aws s3 cp dist/ s3://<bucket_name>/ --recursive
```

Listing 5. Deploy stage.

As presented in Listing 5, the updated configuration includes new `deploy` stage and `amazon-deploy` job, which uses `python` image. To get access to the Amazon instance, `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` should be provided. They can be added using variables section in configuration. Listing 6 shows how variables can be defined in GitLab CI configuration file.

```
variables:
  variable1: value1
  variable2: value2
  ...
```

Listing 6. Definition of variables in GitLab CI configuration.

As illustrated in Listing 6, any variables can be added using variables section. But it is not recommended to use this section for secret variables as it is not secure [54]. Secret variables should not be stored anywhere in the repository. GitLab has a special place for storing secret variables. They can be added in Pipeline settings. Figure 16 shows GitLab interface for adding secret variables.

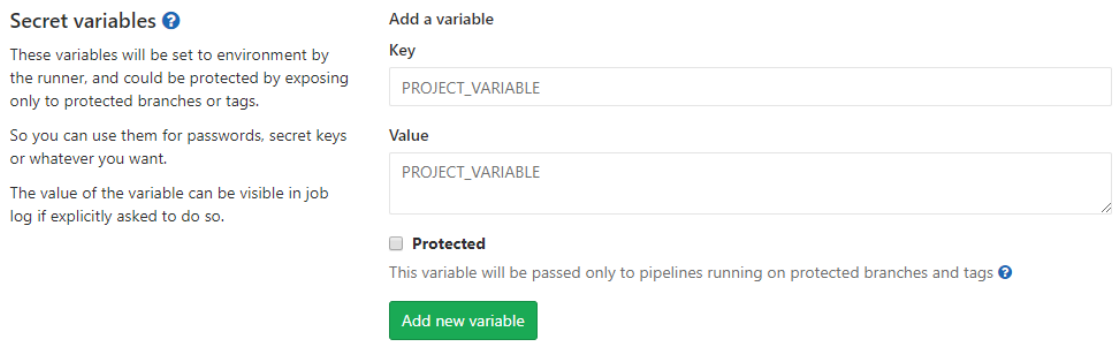

Figure 16. Interface for adding secret variables.

Before the GitLab CI configuration updates were pushed to GitLab, it should be specified that `build` and `deploy` stages should be triggered only for the master branch. GitLab has `only` and `except` parameters, that define whether to run a specific job or not. It can be defined by passing keywords, regex or just branch name to those parameters [54]. Listing 7 shows what should be added both to `build` and `deploy` stages.

```
only:
    - master
```

Listing 7. Adding only option to GitLab CI configuration.

When a merge request is merged to master branch, it not always needed to deploy immediately to production. With GitLab CI it is possible have postpone a job, so it will be waiting for pressing a play button to run. To apply this behavior, `when` GitLab parameter is used. It can specify whether to run job on success, on failure, always or manually. In the project, this parameter is set to “manual”, so when some feature branch is merged to master, it installs the dependencies, runs tests, builds production bundles and then wait for user interaction to deploy everything to production. Figure 17 depicts GitLab CI manual job.

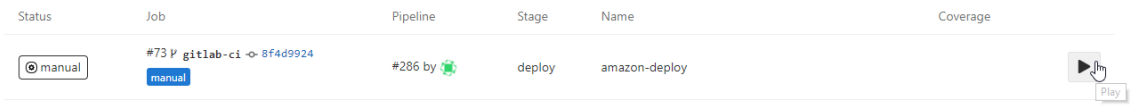

Figure 17. GitLab manual deploy.

As can be seen in Figure 17, the job is skipped and shown on Jobs page with “manual” status. It has play button from the right side, so it can be run directly from the GitLab interface.

After adding all the necessary stages and setting the parameters, the configuration can be merged to the master branch. Listing 8 represents the result version of GitLab CI configuration file.

```
image: node

before_script:
  - cd front

stages:
  - install
  - test
  - build
  - deploy

yarn-install:
  stage: install
  script:
    - yarn install

test:
  stage: test
  script:
    - yarn test

build:
  stage: build
  script:
    - yarn build
  only:
    - master
  artifacts:
    paths:
      - dist/

amazon-deploy:
  image: python
  stage: deploy
  script:
    - pip install awscli
    - aws s3 cp dist/ s3://<bucket_name>/ --recursive
  when: manual
  only:
    - master

cache:
  paths:
    - front/node_modules/
```

Listing 8. Result version of GitLab CI configuration.

As illustrated in Listing 8, the complete GitLab CI configuration for the project includes four stages: `install`, `test`, `build` and `deploy`; specifies cache paths, the order of stages and in which conditions each job should be run.

When all the changes are pushed to GitLab and merge request to maser is created, it can be noticed that the merge request page has new elements and looks slightly different in comparison with previous ones. Now the merge request page also shows the status of the pipeline. Figure 18 depict merge request page with the running pipeline.

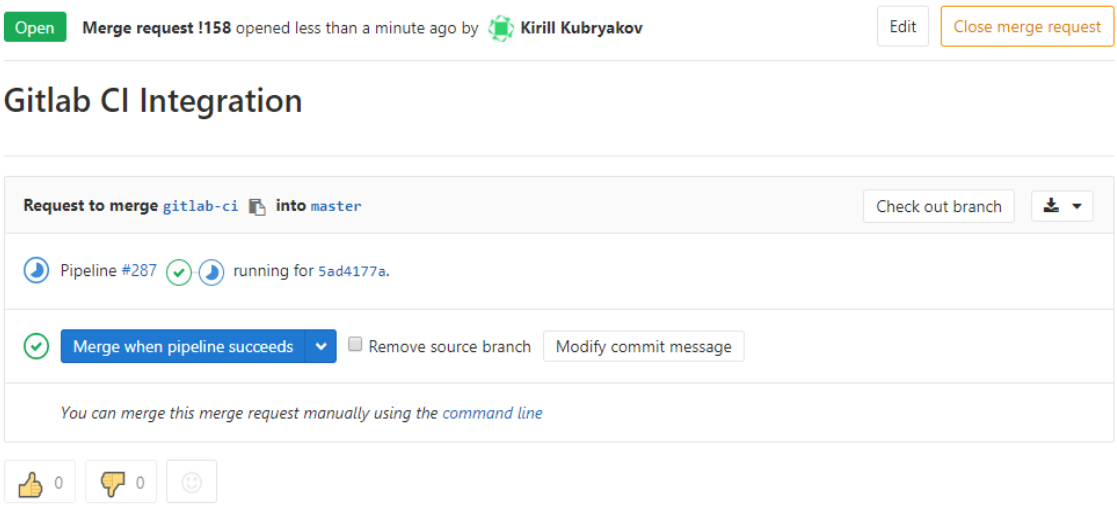

Figure 18. GitLab merge request: running status.

As shown in Figure 18, the page shows pipeline status and different submit button. Instead of asking to merge immediately, it asks to merge only when the pipeline succeeds. After the pipeline succeeds, the merge request page is updated. Figure 19 illustrates merge request page with passed status.

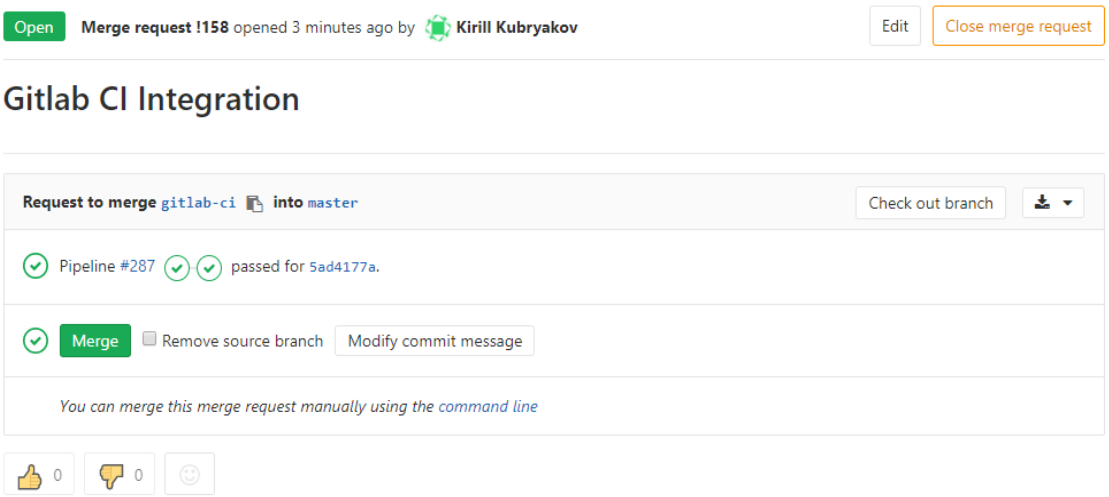

Figure 19. GitLab merge request: passed status.

As can be seen in Figure 19, after all stages are finished, it is indicated in the interface, that the pipeline is passed, and the branch can be successfully merged. But what if some breaking changes to the feature branch are pushed? Figure 20 shows merge request page with error status.

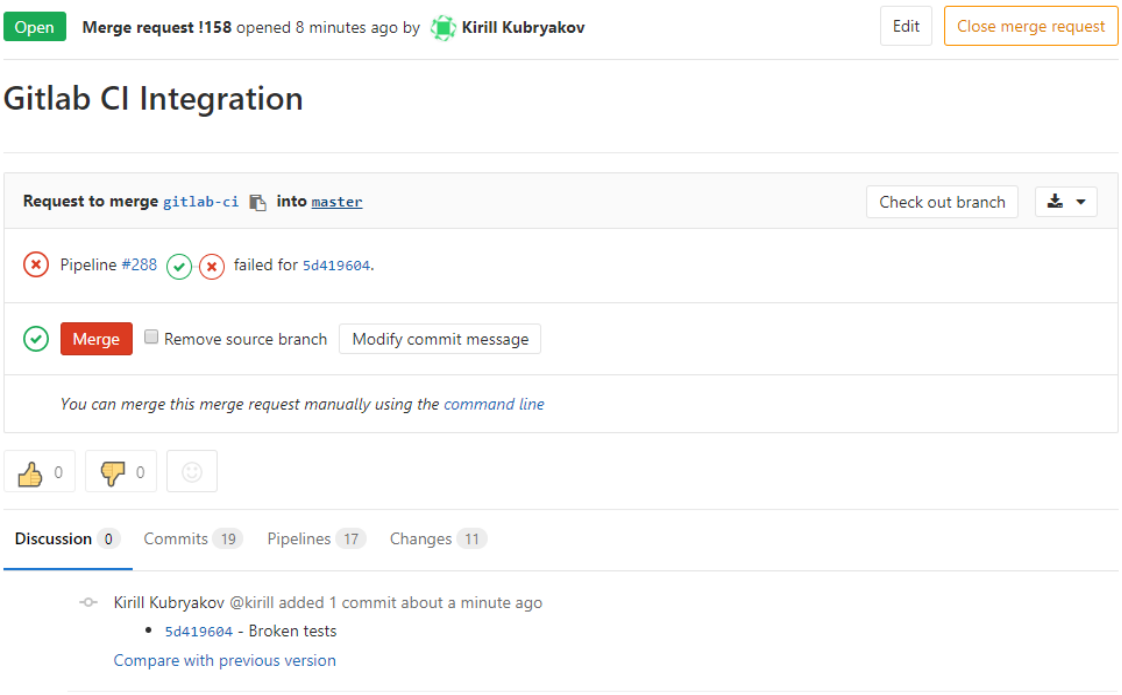

Figure 20. GitLab merge request: error status.

As illustrated in Figure 20, if there are some breaking changes, the status of the pipeline is changed to “failed” and submit button becomes red. It is still possible to accept such merge request, but the developer and the code reviewer are warned that this merge is undesirable, and it will be better to fix the errors first.

4.3 Integration with Other Services

GitLab CI can be integrated with different services. There is some built-in support for popular messengers, issue trackers, other CI tools, and webhooks for other services. GitLab webhooks allow to send a POST request with data to the webhook URL if there is any specific event happens. It supports GitLab events such as pushes, merge requests, as well as GitLab CI events such as pipeline status updates. [55]

Mattermost messenger [56] is used in the case company for internal communication. To integrate GitLab with Mattermost, an incoming webhook integration should be added in Mattermost settings. After that, on GitLab Integrations page, Mattermost pipeline notifications can be enabled by activating Pipeline Trigger. Figures 21 and 22 shows possible Mattermost messages received from GitLab CI.

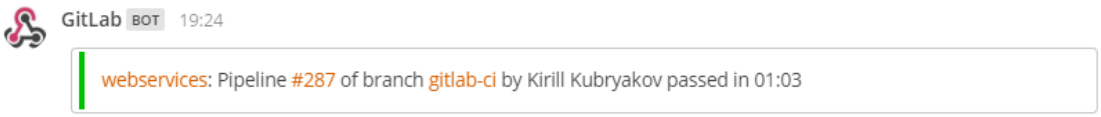

Figure 21. Mattermost success message.

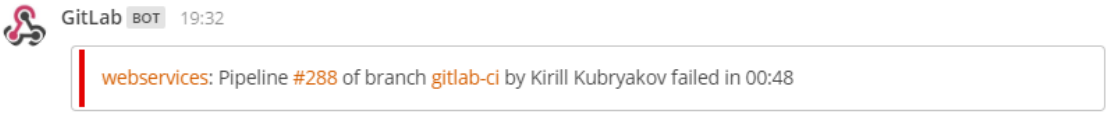

Figure 22. Mattermost fail message.

As can be seen from Figure 21 and 22, the message includes the pipeline status and the name of the branch. Additionally, the pipeline status is indicated by color: green for passed pipelines and red for failed pipelines. The message also includes the name of the user, who created the branch or made the merge request, so he gets a notification immediately after any pipeline status change. It can be helpful especially in the case when a pipeline fails, then the user can quickly start working on fixing bugs and update the branch with fixes.

5 Results

As a result, automation of testing, building and deployment were achieved. The company's web-services GitLab repository is now integrated with GitLab CI. The continuous integration pipeline was setup for the project, which allows to improve and simplify the flow of development and shipping changes to production.

5.1 Description of New Workflow

In comparison with the earlier workflow, the delivery process is simplified by removing manual tasks and describing the entire procedure with all the necessary stages and commands in the configuration. The configuration is stored in one file in master branch. Now, when a developer creates a new feature branch, he will have a configuration file included. During the development, on every push to the repository, GitLab CI will start pipeline and run tests. The pipeline will be also started on merge request to master branch when the feature is ready. If the pipeline is passed, another developer will make a code review and accept the merge request. After merging the feature branch to the master branch, on Jobs page, there will be a play button available to run deploy stage and ship the new feature to production.

5.2 Possible Improvements

The main improvement would be to add a staging environment. It will require a separate server. GitLab allows creating different environments [57]. Staging can be configured two ways. One way is to have one staging branch that will be automatically deployed to the staging server. Thus, every feature branch will be merged to the staging branch and only after undergoing tests staging branch will be merged to master branch. And the second way is to have staging for every feature branch. The second way is preferable, but it requires more resources.

Another improvement can be integration with Mattermost slash commands [58]. It will add additional interface to perform common tasks, and it will allow to respond faster if a problem is detected and rollback is needed.

During this project, a new major version of GitLab was released. This version includes a new product called Auto DevOps [59]. Auto DevOps is built on top of GitLab CI and includes auto build, auto test, auto code quality, auto review app, auto deploy and auto monitoring functionality out of the box. If Auto DevOps is enabled in project settings, then GitLab CI will automatically detect which language is used for application and which tools are used. For example, if there is a Dockerfile in the repository, then it will use `docker build` command to create a Docker image. Currently, this feature is in Beta and not recommended for production use. But it might be an interesting solution and worth switching after it gets stable, because it uses best practices of CI/CD configuration.

6 Discussion

The current study describes the DevOps principles and workflow. The results of the study prove that the implementation of the DevOps practices makes the development process more understandable and effective. The principles of DevOps can be applied to any team, since the principles do not depend on the language, and the tools used for continuous integration and continuous delivery, support various languages and technologies.

Eight CI tools were described and compared in the study. The main contenders were Jenkins and GitLab CI. Even though Jenkins has more functionality and it is more customizable, GitLab CI was chosen because it has better integration with the company GitLab, a set of required options, a simpler setup, and user-friendly interface.

After completing the practical part, the changes were evaluated. It was noticed by the team that less time is spent for testing and it becomes easier to do code reviews, because GitLab CI runs all the tests in background and indicates their status on merge request page. Also, the CI tool helps to keep track of current changes. After some amount of work is done and several commits are pushed to the repository, it notifies in the messenger whether all the tests have passed successfully. Another benefit is that the CI tool helps to avoid mistakes, a person can miss something, but the machine works automatically.

Before the continuous integration tool was added to the project, Docker and Vagrant were already in use. All the developers had the same development environment on their working machines, but the testing and deploy processes were done manually. By using GitLab CI, it was possible to achieve automation and this gap is filled.

The integration of the continuous integration tool and the configuration of the pipeline made a significant impact on the development process of the company's web services. Earlier, the deployment process was complex and required multiple manual tasks. Now all the settings and scripts are stored directly in the project’s GitLab repository and any member of the team can make a release. This is especially important in connection with the fact that the company is growing, and new developers are joining the team.

7 Conclusions

The goal of the final year project was to integrate continuous integration system into the ongoing project to provide continuous testing and continuous delivery. The objective of the project was met successfully. CI tool was integrated with the version control system and a continuous delivery pipeline was configured. Although there are still opportunities for further enhancements of the pipeline, such as adding a staging server and improving the test coverage.

This study shows how the use of the DevOps principles can change the workflow in the company. Even though the release process itself takes the same amount of time, this time is not taken from developers, but the process is running on background on a CI server and the team get a notification when the release is ready. Thereby developers can spend more time on implementing new features. It also affects product stability because the new version will never be released if there are any test failures. All this together allows the company to optimize the development process, and react more quickly to changing markets and customer demands.

References

1 Mike Loukides. Revisiting “What is DevOps” [online]. O’Reilly Radar; 2014.
URL: http://radar.oreilly.com/2014/06/revisiting-what-is-devops.html
Accessed 10 May 2017.

2 Richard Rapaport. A Short History of DevOps [online]. Rewrite Digital Magazine; 2014.
URL: https://www.ca.com/us/rewrite/articles/devops/a-short-history-of-devops.html
Accessed 10 May 2017.

3 6 Problems DevOps Helps to Solve [online]. Datical; 2015.
URL: https://www.datical.com/blog/6-problems-devops-helps-solve/
Accessed 28 May 2017.

4 Ernest Mueller. What Is DevOps? [online]. The agile admin; 2017.
URL: https://theagileadmin.com/what-is-devops/
Accessed 12 May 2017.

5 Gene Kim, Jez Humble, Patrick Debois, John Willis. The DevOps Handbook. Portland, OR: IT Revolution Press; 2015.

6 Scott Ambler. The Principles of Lean Software Development [online]. The Disciplined Agile (DA) Framework; 2016.
URL: http://www.disciplinedagiledelivery.com/lean-principles/
Accessed 12 May 2017.

7 Scott Ambler. Examining the Agile Manifesto [online]. Ambysoft; 2014.
URL: http://www.ambysoft.com/essays/agileManifesto.html
Accessed 12 May 2017.

8 Key features of Waterfall and Agile development methods [online]. M&S Consulting; 2017.
URL: http://www.mandsconsulting.com/key-features-of-waterfall-and-agile-development-methods
Accessed 21 May 2017.

9 Ashton Scheshan Gangadeen. What is DevOps? [online]. Supinfo; 2016.
URL: https://www.supinfo.com/articles/single/3652-what-is-devops
Accessed 21 May 2017.

10 Nick M Loghmani. Continuous Integration vs Continuous Delivery vs Continuous Deployment [online]. LinkedIn; 2015.
URL: https://www.linkedin.com/pulse/continuous-integration-vs-delivery-deployment-nick-m-loghmani/
Accessed 21 May 2017.

11 CAMS [online]. DevOps Dictionary; 2015.
URL: http://devopsdictionary.com/wiki/CAMS
Accessed 25 May 2017.

12 John Willis. DevOps Culture (Part 1) [online]. IT Revolution; 2012.
URL: http://itrevolution.com/devops-culture-part-1/
Accessed 25 May 2017.

13 Gene Kim. The Three Ways: The Principles Underpinning DevOps [online]. IT Revolution; 2012.
URL: http://itrevolution.com/the-three-ways-principles-underpinning-devops/
Accessed 27 May 2017.

14 Adam Bertram. The Three Ways: Core principles of DevOps [online]. InfoWorld; 2016.
URL: https://www.infoworld.com/article/3090137/devops/the-three-ways-core-principles-of-devops.html
Accessed 27 May 2017.

15 Sven Malvik. Top 3 DevOps Practices for Operational Stability [online]. DevOps.com; 2015.
URL: https://devops.com/top-3-devops-practices-operational-stability/
Accessed 28 May 2017.

16 Pete Goldin. 25 Advantages of DevOps - Part 4 [online]. DEVOPSdigest; 2016.
URL: http://www.devopsdigest.com/devops-advantages-4
Accessed 28 May 2017.

17 Pete Goldin. 25 Advantages of DevOps - Part 2 [online]. DEVOPSdigest; 2016.
URL: http://www.devopsdigest.com/devops-advantages-2
Accessed 28 May 2017.

18 Scott Johnston. DevOps Improves Time to Market — and Revenue [online]. Puppet blog; 2013.
URL: https://puppet.com/blog/devops-improves-time-to-market-%E2%80%94-and-revenue
Accessed 28 May 2017.

19 Automation for DevOps [online]. Chef; 2016.
URL: https://pages.chef.io/rs/255-VFB-268/images/automation-for-devops.pdf
Accessed 2 June 2017.

20 Patrick W Roach. Dice Breakers: using DevOps principles and nerdery to reimagine Team building [online]. DevOps.com; 2015.
URL: https://devops.com/dice-breakers-using-devops-principles-nerdery-reimagine-team-building/
Accessed 2 June 2017.

21 Jim Bird. Change control in DevOps [online]. O'Reilly; 2016.
URL: https://www.oreilly.com/ideas/change-control-in-devops
Accessed 2 June 2017

22 Mirco Hering. Continuous everything in DevOps…what is the difference between CI, CD, CD…? [online]. Accenture Technology; 2015.
URL: https://www.accenture.com/us-en/blogs/blogs-continuous-everything-devops
Accessed 10 August 2017

23 What is Continuous Integration? [online]. Amazon AWS; 2017.
URL: https://aws.amazon.com/devops/continuous-integration/
Accessed 10 August 2017

24 Srini Penchikala. Interview and Book Review: Continuous Delivery [online]. InfoQ; 2011.
URL: https://www.infoq.com/articles/humble-farley-continuous-delivery
Accessed 10 August 2017

25 Martin Fowler. ContinuousDelivery [online]. Martin Fowler blog; 2013.
URL: https://martinfowler.com/bliki/ContinuousDelivery.html
Accessed 14 August 2017

26 Lianping Chen. Continuous Delivery: Huge Benefits, but Challenges Too [online]. InfoQ; 2015.
URL: https://www.infoq.com/articles/cd-benefits-challenges
Accessed 14 August 2017

27 Christopher Null. Infrastructure as code: The engine at the heart of DevOps [online]. TechBeacon; 2015.
URL: https://techbeacon.com/infrastructure-code-engine-heart-devops
Accessed 17 August 2017

28 Alex Pott. Principles of Configuration Management - Part One [online]. CHAPTERTHREE Blog; 2014.
URL: https://www.chapterthree.com/blog/principles-of-configuration-management-part-one
Accessed 17 August 2017

29 Dan Radigan. Continuous integration, explained [online]. Atlassian; 2017.
URL: https://www.atlassian.com/continuous-delivery/continuous-integration-intro
Accessed 18 August 2017

30 Vladimir Pecanac. Top 8 Continuous Integration Tools (2017 edition) [online]. Code Project; 2017.
URL: https://www.codeproject.com/Articles/1080526/Top-Continuous-Integration-Tools
Accessed 18 August 2017

31 Alex Blewitt. Hudson Renames to Jenkins [online]. InfoQ; 2011.
URL: https://www.infoq.com/news/2011/01/jenkins
Accessed 18 August 2017

32 Plugins Index [online]. Jenkins; 2017.
URL: https://plugins.jenkins.io/
Accessed 18 August 2017

33 What is Jenkins and Use of it? [online]. Selenium Easy; 2016.
URL: http://www.seleniumeasy.com/jenkins-tutorials/what-is-jenkins-and-advantages-of-jenkins-continuous-integration-tool
Accessed 18 August 2017

34 Buy TeamCity [online]. TeamCity; 2017.
URL: https://www.jetbrains.com/teamcity/buy/
Accessed 23 August 2017

35 Features [online]. TeamCity; 2017.
URL: https://www.jetbrains.com/teamcity/features/
Accessed 23 August 2017

36 Kevin Deisz. Best Practices and Common Mistakes with Travis CI [online]. LocalyticsEng; 2017.
URL: http://eng.localytics.com/best-practices-and-common-mistakes-with-travis-ci/
Accessed 27 August 2017

37 ABS. Continuous Delivery Pipelines: GoCD vs Jenkins [online]. HighOps; 2014.
URL: https://highops.com/insights/continuous-delivery-pipelines-gocd-vs-jenkins
Accessed 27 August 2017

38 Ken Mugrage. Part 2 - It's Not CD if You Can't Deploy Right Now: Add Security Testing to Your Deployment Pipelines [online]. Go; 2016.
URL: https://www.gocd.org/2016/02/08/not-done-unless-its-done-security.html
Accessed 27 August 2017

39 Kishanthan Thangarajah. Bamboo and Jenkins – Continuous Integration [online]. Kishanthan's Blog; 2011.
URL: https://kishanthan.wordpress.com/2011/12/18/bamboo-and-jenkins-continuous-integration/
Accessed 27 August 2017

40 Job van der Voort. GitLab 8.0 released with new looks and integrated CI! [online]. GitLab Blog; 2015.
URL: https://about.gitlab.com/2015/09/22/gitlab-8-0-released/
Accessed 27 August 2017

41 Margaret Rouse. YAML (YAML Ain't Markup Language) [online]. TechTarget; 2017.
URL: http://searchitoperations.techtarget.com/definition/YAML-YAML-Aint-Markup-Language
Accessed 27 August 2017

42 Pierre de La Morinerie. Building our web-app on GitLab CI [online]. GitLab Blog; 2016.
URL: https://about.gitlab.com/2016/07/22/building-our-web-app-on-gitlab-ci/
Accessed 27 August 2017

43 GitLab Continuous Integration (GitLab CI) [online]. GitLab Documentation; 2017.
URL: https://docs.gitlab.com/ce/ci/
Accessed 27 August 2017

44 Ben Dougherty. Bamboo Vs. Travis CI Vs. Circle CI Vs. Codeship [online]. IT Central Station; 2016.
URL: https://www.itcentralstation.com/product_reviews/travis-ci-review-32073-by-ben-dougherty
Accessed 27 August 2017

45 Alex Gorbatchev. Continuous Integration in the Cloud: Comparing Travis, Circle and Codeship [online]. StrongLoop; 2015.
URL: https://strongloop.com/strongblog/node-js-travis-circle-codeship-compare/
Accessed 27 August 2017

46 Jez Humble. Continuous Testing [online]. Continuous Delivery; 2016.
URL: https://continuousdelivery.com/foundations/test-automation/
Accessed 3 September 2017

47 Ely Hechtel. Top 10 Benefits of Automated Testing [online]. SauseLabs; 2015.
URL: https://saucelabs.com/blog/top-10-benefits-of-automated-testing
Accessed 3 September 2017

48 Run GitLab Runner in a container [online]. GitLab Documentation; 2017.
URL: https://docs.gitlab.com/runner/install/docker.html
Accessed 11 September 2017

49 Registering Runners [online]. GitLab Documentation; 2017.
URL: https://docs.gitlab.com/runner/register/index.html
Accessed 11 September 2017

50 The GNU nano homepage [online]. The GNU nano; 2017.
URL: https://www.nano-editor.org/
Accessed 12 September 2017

51 Getting started with GitLab CI [online]. GitLab Documentation; 2017.
URL: https://docs.gitlab.com/ce/ci/quick_start/README.html
Accessed 12 September 2017

52 Yarn package manager [online]. Yarn; 2017.
URL: https://yarnpkg.com/lang/en/
Accessed 12 September 2017

53 CLI Introduction [online]. Yarn; 2017.
URL: https://yarnpkg.com/en/docs/cli/
Accessed 12 September 2017

54 Configuration of your jobs with .gitlab-ci.yml [online]. GitLab Documentation; 2017.
URL: https://docs.gitlab.com/ee/ci/yaml/
Accessed 12 September 2017

55 Project integrations [online]. GitLab Documentation; 2017.
URL: https://docs.gitlab.com/ee/user/project/integrations/
Accessed 14 September 2017

56 Mattermost [online]. Mattermost; 2017.
URL: https://about.mattermost.com/
Accessed 14 September 2017

57 Introduction to environments and deployments [oneline]. GitLab Documentation; 2017.
URL: https://docs.gitlab.com/ce/ci/environments.html
Accessed 1 October 2017

58 Mattermost slash commands [oneline]. GitLab Documentation; 2017.
URL: https://docs.gitlab.com/ee/user/project/integrations/mattermost_slash_commands.html
Accessed 1 October 2017

59 Mike Bartlett. GitLab 10.0 Released with Auto DevOps and Group Issue Boards [online]. GitLab Blog; 2017.
URL: https://about.gitlab.com/2017/09/22/gitlab-10-0-released/
Accessed 1 October 2017