UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Software Engineering Curriculum

Gunel Ismayilova

# Adopting DevOps Practices: A Case Study

Master's Thesis (30 ECTS)

Supervisor(s): Ezequiel Scott, PhD
Madis Kapsi, CTO

Tartu 2021

## Adopting DevOps Practices: A Case Study

**Abstract:**
Nowadays, delivering on higher levels of customer satisfaction for online services is highly demanded from organisations. Furthermore, to continue supporting these services is part of the job. Delivering and supporting a higher level product from the idea to the end result requires a wide range and heavy amount of work. Time, efficiency, maintainability, security and many other factors are part of this process, i.e. the process of software development life-cycle. DevOps brings its own efficient and beneficial advantages to the field. It is a framework which integrates software development and IT operations. It is a combination of philosophies, practices and tools that can benefit an organisation to deliver applications and services at a high velocity. The goal of this thesis is to research DevOps practices and implement them in a real case scenario, in a project which serves customers. Moreover, this thesis is about tracking and measuring the results of these changes, then comparing them. The results show that DevOps, when used in a correct way, brings value to all stakeholders. Additionally, this thesis highlights the limitations when trying to adopt these practices.

**Keywords:**
DevOps, Agile Software Development, CI/CD pipeline, Cloud Environments, Continuous practices, Automation

**CERCS:**P170 - Computer Science

## Tüübituletus neljandat järku loogikavalemitele

**Lühikokkuvõte:**
Tänapäeval nõutakse organisatsioonidelt sidusteenuste puhul kõrgetasemelist kliendirahulolu. Lisaks on nende teenuste toetamine osa tööst. Kõrgetasemelise toote tootmine ja toetamine ideest lõpptulemuseni nõuab laiahaardelist ja mahukat tööd. Aeg, efektiivsus, hallatavus, turvalisus ja paljud muud faktorid on osa sellest protsessist, st tarkvaraarenduse elutsüklist. DevOps pakub efektiivseid ja kasulikke eeliseid selles vallas. See on raamistik, mis hõlmab endas tarkvaraarendust ja IT operatsioone. See on segu filosoofiatest, praktikatest ja tööriistadest, mis võivad organisatsioonil aidata rakendusi kiiresti toota. Selle uurimistöö eesmärk on uurida DevOps praktikaid ja neid praktikas rakendada projektis, mis teenindab kliente. Lisaks sisaldab see uurimus loodud muudatuste mõju jälgimist ja mõõtmist ning omavahel võrdlemist. Tulemused näitavad, et kui DevOps praktikaid õigesti kasutada, on see tulus kõikidele sidusrühmadele. Samuti on välja toodud antud praktikatega seotud piirangud.

**Võtmesõnad:**
DevOps, Agiilne tarkvaraarendus, CI/CD torujuhe, Pilvekeskkonnad, Pidevad harjutused,

Automatiseerimine

**CERCS:**P170

# Contents

# 1 Introduction

Agile is a set of principles by which a team can execute a project by splitting it into many phases and requiring constant communication with stakeholders at any point and continuous development and iteration. Agile Software Development is based on velocity of response to changes and providing competitive advantage. A dominant idea in agile development is that the team can be more effective in responding to change [4]. It does help to reduce the cost of moving information between people and the time period between making decisions and observing the changes. Agile does focus on soft skills in humans such as talent, skill and communication. Hence, skill development is also part Agile development. The more skilled and talented the more value can be delivered with shorter amount of time. However, sometimes speed, frequent changes and keeping up with the market changes at some point can create bottlenecks. Such as between Operations and Development, where delays and confusions may happen frequently. In order to solve these issues, Debois advocated a tighter integration between the Dev and Ops functions which is termed DevOps [7].

The word “DevOps” was invented in 2009 by Patrick Debois, where it was formed by combining two words “development” and “operations”. DevOps is a group of activities that combine IT operations (Ops) and software development (Dev). It increases an organisation’s ability to deliver applications and services at a high velocity. Nowadays, application development and deployment have become an important part of company operations, which is leading to a rise in the popularity of DevOps. Where DevOps aims to reduce the time spent on the software development process to bring the product to market. Although, it is possible to schedule any work in operations: releasing a system update, switching between data centres, or any system change. Still much of the organisational work is unplanned: performance fluctuations, system outages, and security compromises. Such incidents require immediate action.

On the other hand DevOps aims to reduce the time between committing a change to a system and the change being placed in to production stage, at the same time keeping a high quality. Recently more and more companies are adopting DevOps in an increasing number. DevOps practices benefit developers during the software development life cycle in various ways. Tests and deployments are possible to automate, which results narrowing down developers’ time into specific tasks. Moreover, with the help of available tools, it is possible to detect and refactor the existing or committed issues in the repository. Furthermore, DevOps accelerates Agile software development where it increases performance of IT teams which also affects cost reduction in software life-cycle at the same time ensuring high quality of a product. However, adopting modern practices has never been an easy step although the results are worth it.

With all the benefits and advantages to adopt a successful DevOps path changes are required in the organisational, personnel and technological levels, where they come at a cost. The demand in the marketplace is always increasing to be more productive, faster,

more secure, to have higher quality results and most importantly to be agile. In other terms to be flexible to any upcoming innovation.
In this thesis, I explore a case study on DevOps practices adoption. In particular, the case is on exploring a single page web-based application (SPA) project in a small-to-medium sized (SME) company located in Tartu, Estonia. The company Singularity Creations delivers services for small and medium-sized businesses. The research started in the 4th quarter of 2020 and ended in mid-2021. The examination is mainly about how DevOps practices improved the quality of software development process, which in itself includes the whole life-cycle of product from an idea to an end product. Although, there are still a lot to do but there are takeaways, benefits and advantages gained along the way. I dive into details on why the company need to adopt these practices. Furthermore, I investigate why they were beneficial and if they were, how did these benefits affected. In the end I capture the results, measures then compare them. To finalise, I discuss the results and give a conclusion on the case study.
This thesis is organised as follows. In chapter 2, I explain the fundamental concepts about this thesis. In chapter 3 I go through other related case studies and discuss briefly about them. In chapter 4 and 5 I explain the methodology and the results. Later on in chapter 6 I list the findings, lessons learned, challenges that team and I faced and lastly about future work. Finally, In the last chapter I briefly finalise the case.

# 2 Background

## 2.1 DevOps

Back in time developers worked in the day time and deployed during late hours. Where deploying was not an easy job which took several hours to build and deploy the project. There are number of challenges to old style such as: Inconsistency between Dev and Ops, slow deployment, low productivity, difficulties with managing versions and configurations, more error prone, higher costs. According to Iden, Tessem and Päiväinta [8], results to a number of problems, including IT operations not being involved in requirements specification, poor communication and information flow between the two groups, unsatisfactory test environments, lack of knowledge transfer between the two groups, systems put into production before they are complete and operational routines not established prior to deployment.
Though today this process is made easier with DevOps continuous and automated practices and tools. Automation is a key part of DevOps in that it allows the speed of delivery with quality and it extensively rely on automation tools. Hence any repeated process should be automated. As with all technological revolutions, DevOps practices impact processes, products, associated technologies, organizational structures, and business practices and opportunities [25]. Of course, with all the benefits the adoption of this process is not an easy journey. It contains both technical and cultural transition and it usually affects developers in the software development life-cycle. DevOps practices grew up in organisations providing services over the Internet with, essentially, one very complex and large system [25].
The concept of DevOps became popular in 2009 after the "DevOpsdays" conference in Belgium. Since then, such conferences have been held in many countries around the world. The popularity of DevOps has increased in recent years, leading to the creation of additional branches such as OpsDev, WinOps and BizDevOps.
There are many contradictory thoughts on it. Some say that DevOps is “a software development method that combines quality assurance mechanisms with IT operations within software engineering practices” [14], on the contrary others argue that it is “not a method”, but an approach [5]. If approached in a more technical way it can be stated like “to automate the complete deployment process from the source code in version control to the production environment” [23] or “fast feedback, small batch sizes, and independent releases” [2]. DevOps is an emerging culture in which development, testing, operations teams collaborate to deliver outcome in a continuous and effective manner DevOps represents a significant opportunity for organisations to gain market place in the comparison of their competition and build better applications; thus opening the door for increased benefits and improved customer experiences [20]. That being said according to Manish Virmani [22], DevOps just defines the set of principles but “how and using what technology” organisations adopt that approach or principle is completely to be evaluated

and decided by the organisation.
To conclude DevOps has 4 main principles [6]:

- Culture—DevOps requires a cultural change of accepting joint responsibility for delivery of high-quality software to the end-user. This means that code no longer can be “thrown over the wall” to operations.
- Automation—DevOps relies on full automation of the build, deployment and testing in order to achieve short lead times, and consequently rapid delivery and feedback from end-users.
- Measurement—Gaining an understanding of the current delivery capability and setting goals for improving it can only be done through measuring. This varies from monitoring business metrics (e.g., revenue) to test coverage and the time to deploy a new version of the software.
- Sharing—Sharing happens at different levels, from sharing knowledge (e.g. about new functionality in a release), sharing tools and infrastructure, as well as sharing in celebrating successful releases to bring development and operations teams closer together.

Table 1. DevOps capabilities and their technological enablers.

| **Capabilities** | **Technological Enablers** |
|---|---|
| Collaborative and continuous development | Build automation |
| Continuous integration and testing | Test Automation |
| Continuous release and deployment | Deployment Automation |
| Continuous infrastructure monitoring and optimization | Monitoring Automation |
| Continuous user behavior monitoring and feedback | Recovery automation |
| Service failure recovery without delay | Infrastructure automation |
| Continuous Measurement | Configuration management for code and infrastructure |

Table 1 refers to the paper [19].

### 2.1.1 Continuous Integration

According to [9], an extremely strange, but common, feature of many software projects is that for long periods of time during the development process the application is not in a working state. The reason for this is explained as, "Nobody is interested in trying to run the whole application until it is finished. Developers check in changes and might even run automated unit tests, but nobody is trying to actually start the application and use it in a production-like environment" [9]. In the long run this can end up with creating lot's problems and so many complex and confusing that it can take up to days to figure out, when trying to merge and deploy all the changes. Hence the goal is every time when there is a commit the entire application is built and the tests run against it. To make incremental changes and to get instant feedback of the implemented change.
Continuous integration is the best known of the Continuous family [6]. In Continuous Integration (CI), Developer team has been addressed during DevOps life-cycle. Routine practice for integration process of developer's code may detect the errors quickly and can avoid failure of builds [1]. In other terms, Continuous integration basically refers to integrate early, don't keep changes localised to your work-space for long, instead share your changes with team and validate how code behaves continuously [22]. However, selecting the right tools to work with is the main challenge here. Hence combining these tools is another step. Then comes the automation part in a way that as soon as the change is delivered the system detects it and triggers consecutive orders of jobs depending on the company, such as automated unit tests, end-to-end tests then building to a repository. According to [6], continuous integration may be defined as a process which is typically automatically triggered and comprises inter-connected steps such as compiling code, running unit and acceptance tests, validating code coverage, checking compliance with coding standards, and building deployment packages.
It is mostly focused on smaller code changes and smaller commits. Usually, changes are committed at least once a day. As well code from the main branch repository is pulled to the local host machine so conflicts can be minimised before merge.

### 2.1.2 Continuous Deployment

Continuous Deployment (CD): Continuous deployment states that each change that occur, move through a pipeline of tests and if it qualifies/ passes all tests, it automatically gets deployed for in the production process [1]. According to Manish Virmani [22], This is heart of DevOps and forms the critical piece of overall software delivery optimisation. It is usually followed by Continuous delivery.

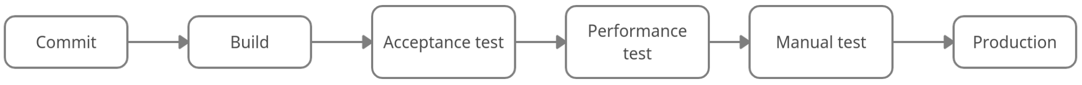
### 2.1.3 Continuous Testing

Continuous Testing (CT) considers one of the important aspect of development that ensures the product quality deploy to the end-user [1]. In continuous process, changes do happen frequently from development to deployment. Where continuous testing is about testing at the early stages, testing frequently, testing everywhere and of course automating it. This whole principle of continuous testing not only moves the testing process to early in cycle but also allows the tests to be carried out on production like system (complemented by continuous deployment) [22].

### 2.1.4 Continuous Delivery

Continuous Delivery (CD) is a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time [3]. According to [3], Continuous Delivery Pipeline can be described as in Figure 1. CD focuses to automate the process like building, testing and making it ready for production. How CD is implemented can be briefly described like this. First development team must be sure that environment variables are not stored in the application configuration. Because CI/CD allows us to hide or mask the sensitive data such as passwords and account keys, then to configure them on the fly in other terms at the time of deployment. If any failure takes place alerts and notifications will be received. The integration usually happens with version control and agile tools. The main difference between Continuous Delivery from Continuous Deployment is, in the last where deployment to production takes place it is rather done manually.

Figure 1. Continuous Delivery Pipeline.



After code commit is done the first happening thing is code compilation then if nothing fails. Then the build stage where the test coverage reports are generated. Then in build phase the artefacts are built and is uploaded to the repository. The pipelines on later stage will run upon these artefacts. Next the tests start being executed on the setup, build and ready environment.
The acceptance test stage mainly ensures that the software meets all specified user requirements [3]. The tests run in a production-like environment created by the pipelines. No failure then next step.
The performance test stage gauges how the code change will affect the software's performance, where the pipeline sets up the performance test environment, runs a suite of performance tests in this environment, and feeds the results into the tool that centrally manages software quality. [3]. The benefit of having performance tests just like any other testing phases, are it helps to get immediate feedback if the code change has negatively affected the software performance. Hence locating and fixing can be done in earlier stages which is much cheaper.

Lastly, the production. Just takes a click of a button. Previously, you could get bunch of errors related to different things. Now, CD has no manual deployment steps, and the deployment process and scripts have been tested many times in previous stages [3]. Previously, setting up environment and tools was so much of a hassle that these can now be done in a matter of minutes with pipelines. These all are read and executed from scripts where I will explain in further section.

### 2.1.5 Continuous Planning

The initial step for DevOps adoption in cloud is continuous planning i.e., bring all developers, operational team members, testers and business analysts at some common platform to prepare and release the plan [1]. DevOps allows you to adapt to quick changes in business environments by having a prioritised product backlog, continuous channel of feedback with customers. Product backlog should able to customised all the time depending on the priority. There is a continuous process to plan small portion - execute - get feedback - react to feedback and adjust plan if needed and the cycle continues [22].

### 2.1.6 Continuous Monitoring

One of the key factors in DevOps is continuous monitoring. Monitoring consists in collecting data from the system running in production as well as users' feedback, which can be used by Dev and QA teams for measurement and optimisation in next testing stage [17]. The goal is to make server-side data easily readable for the stakeholders in the product development.

## 2.2 Version control and branching strategies

When infrastructure is defined in code, we can do with it all the usual things that we do with application source code. These include: execute it in repeatable way, test it in an automated way and put it under version control [15]. According to [9], "In essence, the aim of a version control system is twofold: First, it retains, and provides access to, every version of every file that has ever been stored in it. Such systems also provide a way for metadata—that is, information that describes the data stored—to be attached to single files or collections of files. Second, it allows teams that may be distributed across space and time to collaborate". Additionally, configuration management is a synonym for version control. With the version control you can:

- exactly reproduce any of your environments, including the version of the operating system, its patch level, the network configuration, the software stack, the applications deployed into it, and their configuration [9].

- easily make an incremental change to any of these individual items and deploy the change to any, and all, of your environments [9].
- easily see each change that occurred to a particular environment and trace it back to see exactly what the change was, who made it, and when they made it [9].
- satisfy all of the compliance regulations that I am subject to [9].
- easily (or any team member) get the information needed, and to make the changes needs to be done [9].

Basically, Version control systems, also known as source control, source code management systems, or revision control systems, are a mechanism for keeping multiple versions of files,when any modification is made there is still a chance to access the previous versions [9]. The main idea is everything that can be changed at any time in the life cycle of product should be controlled. In other words versioning is timeline or history of the product. In conclusion, it is impossible to do continuous integration, release management, and deployment pipeline without it and it also makes a huge positive impact on collaboration within delivery teams [9].
Just like the other practices, DevOps impacts on several software engineering disciplines, including Software Configuration Management (SCM). SCM is an engineering discipline that provides methods and tools to identify and control the software throughout its development and use [16]. The important thing when storing information or versioning is to have everything at the bare minimum, which is needed to re-create the testing and production/staging/development environment. It seems that version control systems are the central way to design the deployment pipeline and to have less error-prone and risky releases.
To clear up few thoughts and to summarise I state that Continuous Delivery is not Continuous deployment. The main difference between them is about deployment part. Businesses who want to upgrade applications regularly and need a stable delivery mechanism should use CI/CD pipelines. The manufacturing process for deploying code changes includes additional effort to standardise builds, develop tests, and automate deployments. It allows teams to concentrate on the process of improving software rather than the device specifics to deliver the products.
To summarise, CI packages notifies developers if any fails happen in software builds and tests. On the other hand, CD is the automation process to deliver the changes and to run any needed extra tests.
Lastly and most importantly it should be noted that getting started with CI/CD requires development team as well as operational teams to collaborate. They need to build consensus on the right methods. Thus, they can both can be on-board with the practices when CI/CD is ready.

## 2.3 Cloud Environments for DevOps

Cloud computing is the delivery of computing services over the Internet or in other words the cloud. It provides servers, storage, database, networking, software, analytic, and intelligence. The benefits of cloud computing include lower investment on IT infrastructure, software licensing costs and high scalability. According to Tamil Nadu, cloud computing can be viewed as a new business model using existing technologies like virtualisation and distributed computing having the facilities and features to speed up IT adoption in developing economies [24]. There are three administration models of cloud computing: Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS) [12]. To bundle an application code, designs and conditions into a solitary item is possible through creating containers such as Docker, Docker Swarm and Kubernetes [12].
Companies no longer have to employ experienced personnel to answer the questions of the employees, they can simply acquire a subscription that includes support in their language (or as close to it as available) and make use of both one-on-one support options, and instructional material made available by the SaaS provider [18]. Companies do not need to purchase hardware and software, and companies also do not need to build machine room and recruit IT staff. Only you need to use the Internet information system [11]. Hence, it is responsible for pre-implementation, post-maintenance and number of services.
All these can be achieved by using five clouds: IBM Bluemix, Amazon AWS, Google Cloud Engine, Chameleon Cloud, and FutureSystems.

# 3 Related Work

In this section, I go through other studies and give a brief overview of their work, which are concerning case studies on DevOps practices and papers conducting systematic literature reviews of works in the adoption of DevOps and its practices.
The study [19] presents findings form an exploratory case study which investigates DevOps implementation on a product which is developed by a development organisation located in a New Zealand. The study aims to answer these following questions:

- What are the main drivers for adopting DevOps?
- What are the engineering capabilities and technological enablers of DevOps?
- What are the benefits and challenges of using DevOps?

The research has done six in-depth one-to-one series of interviews over a six-month period. The scope of the case is to comprehend the terms of DevOps from the perspective of employees. They verbosely describe how the teams were not in peace with each other prior to DevOps. To get an in-depth understating of the process and its effects they additionally did a post-DevOps interview with the pre-DevOps Chief Product Officer and Chief Platform Officer. Furthermore, the benefits from DevOps implementation to date, identified by interviewees. Firstly, teams are happier and more engaged, which they described that increased collaboration led to more enjoyable and motivating work. Secondly, more frequent releases. Although the benefits the company faced few challenges while adopting, where they had to recruit a new staff and up-skilling the existing staff in order to have the right technical skills. However, up-skilling was not an easy process since the IT staff was resistance to the change and uncertainty. They also noted that provisioning appropriate tools like cloud hosting platform, micro-services architecture and experimenting automated deployment and monitoring were identified as challenges. While benefits such as high autonomy, motivating collaboration, and feeling valued contributed to improved team morale and engagement, benefits such as improved code quality, natural communications, and knowledge sharing were found to contribute positively to improved deployment frequency [19]. In conclusion the process has to be gradual for both technical and non-technical reasons.
Matt Callanan and Alexandra Spillane [2] conducted a research in a group called Wotif, which is one of Australia's largest travel e-commerce platforms. The article discusses how the group used DevOps principles to recover from the downward spiral of manual release activity that nowadays many IT departments tackle with. Where their main approach is to defining the right approach to make it easy. Furthermore, article goes beyond the mechanics of establishing CD pipelines to look at the technological and cultural challenges Wotif faced and how it overcome them. Initially, like other companies Wotif also suffered from all the problems of the Deploying Software Manually release

anti-pattern in Jez Humble and David Farley's book Continuous Delivery [9]. They created their own method of releasing products and named it SLIPway (Simple Lightweight Independent Pathway). The main rules were to "keep the changes independent", "no manual testing during release", "release slots can't be reserved in the calendar", "applications must comply with the latest standards version" and "operations staff can roll back release after hours". The average release cycle has decreased from almost two weeks to around one day which motivates developers to deploy frequently. Furthermore, more releases now means for them identifying defects are easier and hotfixes can be released much faster. The results were quite outstanding with 95 percent reduction in person-hours spent releasing, 86 percent reduction in release cycle time. They did not aim to automate the deployment because of too many manual steps but the main goal was to reduce the average release cycle time from weeks to less than a day. In conclusion, their SLIPway is an alternative release path that encourages DevOps practices such as fast feedback, small batch sizes, and independent releases, emphasising increased team autonomy [2].
Lianping Chen [3] researched a rapidly growing company named Paddy Power. Company has 4000 employees and approximately €6 billion turnover, which provides services that are used in websites, mobile apps, trading and pricing systems and live-betting-data distribution systems. The paper describes how the company adopts CD, with the huge benefits and challenges involved. They describe their pre-DevOps process as "the releases were "scary" because of the too many error-prone manual actions were involved". Moreover, even setting up a testing environment took weeks of work for them. As a result, the company decided to improve and dedicate a team of eight people to implement CD. The case heavily focuses on to make the product ready for the market as soon as possible, including with reliable releases. The result CD pipeline includes performance, acceptance and manual tests and production ready deployment after code is committed and built. Although their organizational, technical and process challanges which to took them for six months to negotiate they were still able to achieve a success. Succeedingly, able to transition 20 applications to CD which gained these six main benefits: Accelerated Time to Market, Building the Right Product, Improved Productivity and Efficiency, Reliable Releases, Improved Product Quality and Improved Customer Satisfaction [3]. Nevertheless, more research is needed on understanding the challenges in more depth and developing strategies and practices to tackle them more effectively.
Steve Neely and Steve Stolt [13] describe their challenging transition to adopting Continuous Delivery. The case is on a Rally Software company where their aim was to ship the code early and often. The releases took place every eight weeks which was the main aim to improve. Likewise, other adoptions this team also had to tackle with complex challenges with their build systems, automated test suites, customer enablement, and internal communication. Since they did not want to show 5% of the features to the users they had to opt going with building togglable feature. In other words the implemented changes

are togglable which can be hidden and displayed according to desire. Additionally, They also used dark and canary deployments method, which creates a single node in a running multi-cluster system where at some point the changes can be revoked. Overall, “standard” adoption process was not followed, where automated tests are required. Although their personalised transition, they aimed to achieve throughput increased per developer over time as their efforts toward continuous delivery progressed, which is likely due to faster feedback cycles and testing frameworks. Additionally, they quantify their success with the decrease in customers detecting and calling support team. In conclusion, the paper advises to begin automating from the slowest part of the system, which are the most manual and error prone parts. Furthermore, to invest heavily in automated tests.
The goal of the paper [21] is to shed light on the interaction between CI and CD from the aspect of the general development process, managing technical debt, testing activities, technical questions about the CI infrastructure. The case happened in an organisation (ING Nederland) where 152 developers where surveyed about how they adopt a Continuous Integration and delivery pipeline during the development phase. The company has 94000 employees and 67 million customers in around 40 countries. There was a big gap between the IT and core business department, which created managerial costs and efforts. The research questions are "what are the principles of CD", "what practices are needed" and "how is CI implemented or performed". Consequently, findings are:

- better to schedule refactoring together with other development activities
- automated tools and code reviews help to identify code smells caused from "deadline pressures"
- it is not 100% accurate that whether developers follow TDD or not
- the use of a Continuous Integration infrastructure encourages developers to test their changes using private builds [21]

Apart from the studies discussed above, this case mainly focuses on the implementation of the adoption process and improving while optimising the pre-existing practices on an existing project, in a gradual and incremental way. None of the studies discussed or focused on a Software Product Line type application. Hence, diving into details from the DevOps perspective in a SPL project, implementing, measuring and discussing the process is the main aim of this thesis, where I believe a valuable contribution can be made in the end.

# 4 Methodology

In this section, the case study design and case study description are presented. The defined research questions, the details of the case study design and its description are thoroughly explained.

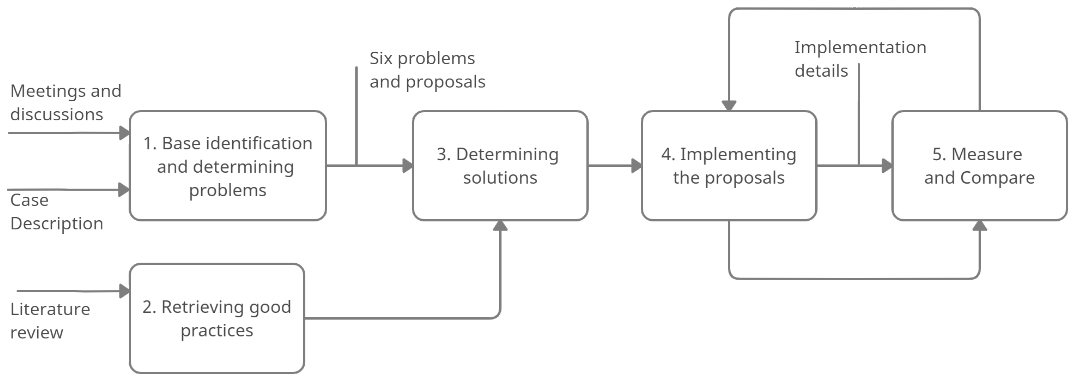
## 4.1 Research Questions

In this case study, the following questions are explored:

- RQ1: What DevOps problems does the company have?
- RQ2: How DevOps practices can improve development at Singularity Creations?

## 4.2 Case Study Design

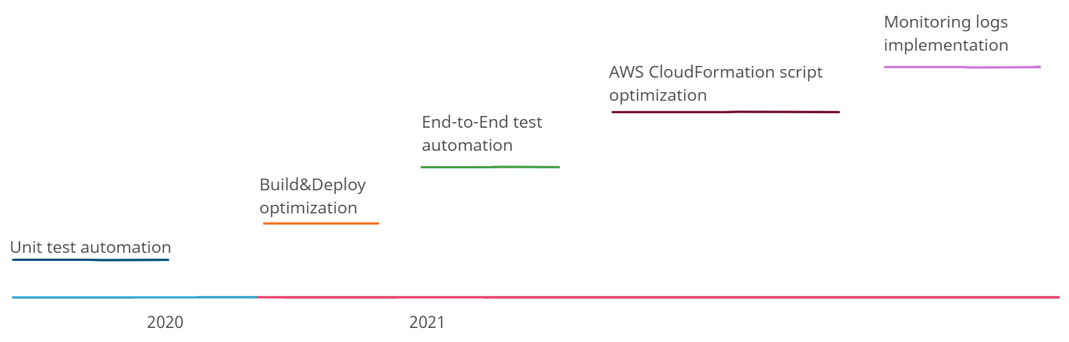
Figure 2. Case Study Design



Meetings and discussions
Case Description
Literature review
1. Base identification and determining problems
2. Retrieving good practices
Six problems and proposals
3. Determining solutions
4. Implementing the proposals
Implementation details
5. Measure and Compare

Figure 2 demonstrates the order of steps of the case. The first step, 1 Baseline identification and determining problems, is to understand the software development process in Javelin project in the first place. I have done several meetings with the company CTO who directly works on the DevOps development in the project. After several on-foot chats and 2 main meetings in which the problems and their potential solutions were discussed. The results were list of main problems and 1-2 solutions for each. The meetings happened between the CTO of the company and I took place between September and October in 2020 mostly. Figure 3 demonstrates when what problems were solved, the details are discussed in the Results section.

Figure 3. Change proposal implementation plan



The importance of prioritisation is to perform the structural changes in the project before implementing other changes. Some issues are dependent on each other. Moreover, the solutions were proposed by considering costs and efficiency since there are multiple applications (admin, client, API) deriving from single source code to multiple themes (coaching, interview and telemedicine) which then is executed in multiple environments (development, staging and production). An efficient altering in the development is required to be made, which supports deploying multiple applications without failures taking place.
The second step, Retrieving good practices, I read other case studies where they aimed to adopt DevOps practices. The case studies are described in detail in Section 2. Where, some had already existing structure but required improvement, on the other hand some started from scratch, or some started without researching and some dedicated a whole new team for the adoption process.
In the third step, Determining solutions, from the information gathered in previous two steps the CTO and I came to a conclusion list of 1-2 solutions for each problem. Eventually, by the end of October it was agreed to select one of the solutions for each of the problem.
In the fourth step, Retrieving implementations, after creating a problem-solution list and doing research I started gradually to put into practice the recommended changes. Furthermore, along the way I dedicated myself to learn DevOps CI/CD process. Additionally, in the whole change proposal process I was guided by the CTO. I also took enough amount of notes about how I made the changes depending on a timeline basis. This helped me to better understand the questions like “how it was before”, “what were the solutions”, “what was implemented”, “how was the impact”, “what were the limitations” and “what can be done for future”.
In the fifth step, Measure and Compare the process, I took the measures after imple-

menting the changes. Although, some practices where completely new which made them tough to measure, I still was able to gather enough information through different methods.
The loop line in from 4th box to 5th and from 5th box back 4th shows that I have implemented the problem, measured it, then went back to the list and focused on the next problem-solution. In may I started taking the final measures to have a better visually comparable results and measures to have a better grasp of the changes.

## 4.3 Case Study Description

Singularity Creations is a SME located in Tartu, Estonia. According to CTO, the company develops several projects where two of them are software as a service (SaaS). The software development process in the company is based on Agile methodologies. Depending on the project and team size there are different sprint weeks. However, they all include daily stand-up and zoom meetings, sprint planning and sprint review meetings as well as retrospectives. On top of that, this thesis is based on the Javelin project. Javelin has one sprint week. The users of the application are mainly in the USA although the project is not limited just to be used in USA. It has been 3 years since the company adopted DevOps where Javelin has been taking advantage of it since the beginning of its development.

### 4.3.1 The Javelin project

The Javelin project is based on Throttle Up Coaching, which is the previous version of the application. The platform provides online coaching service. In addition to this, currently it has been extended to both online Telemedicine and Interview services, where the first and original one is Coaching. Furthermore, the 4th product theme is under negotiation. These all three services have an online training concept in common, where they all are developed under one source code, which makes the product to be a Software Product Line (SPL) application. SPL is a collection of software-intensive systems that have a common and managed feature set and are built in from a common set of core assets to meet the demands of certain market segment [10]. The concept and the core assets are shared among these applications considering that the products are different and independent on their own. Moreover, to maintain a software product line application from code to organizational level optimal changes are required, if not properly setup in the first place.
**Project structure.** The project consists of three applications / source codes (repositories) admin, client and API. Each are developed in a separate sources codes or code repositories. API is written in Node.js combined with GraphQL for database management with additional libraries. Client-side user-interface is for end-users, developed in Vue.js with additional libraries. Admin-side user-interface is for managerial users (organisations,

teachers, doctors etc.), also developed by using Vue.js with additional libraries. Both client and admin is deployed with quasar framework, which is open-source Vue.js based framework. Figures 4, 5 and 6 demonstrate the user interface of the application. The left figures show client accordingly the right ones show admin panel. They share similar view and same code repository, being separate for client and admin. However as a SPL they aim to serve in different business sectors.

Figure 4. Coaching theme Client and Admin.

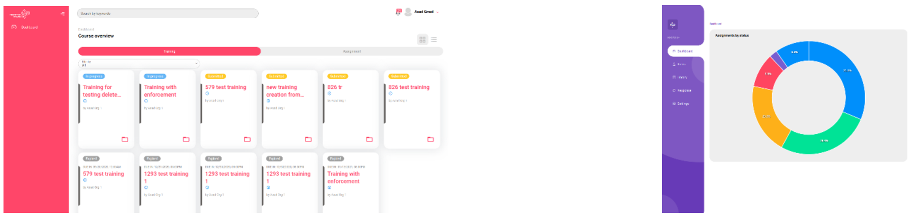


Figure 5. Interview theme Client and Admin.

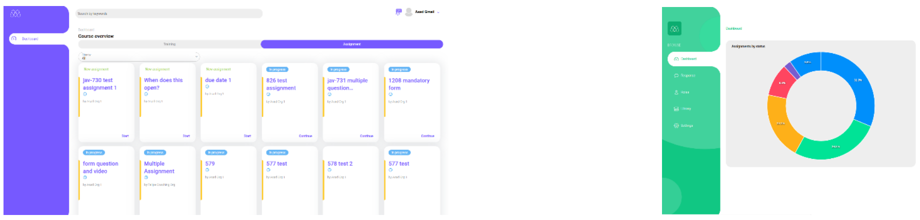
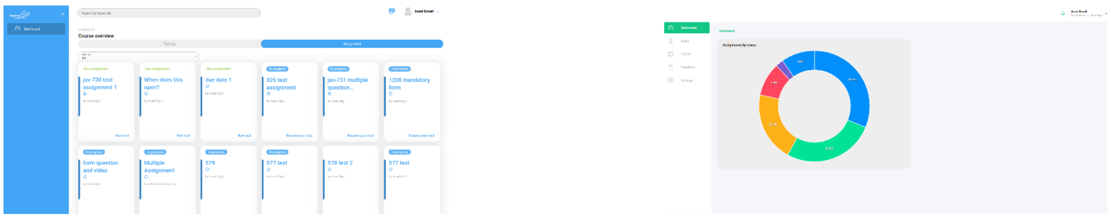
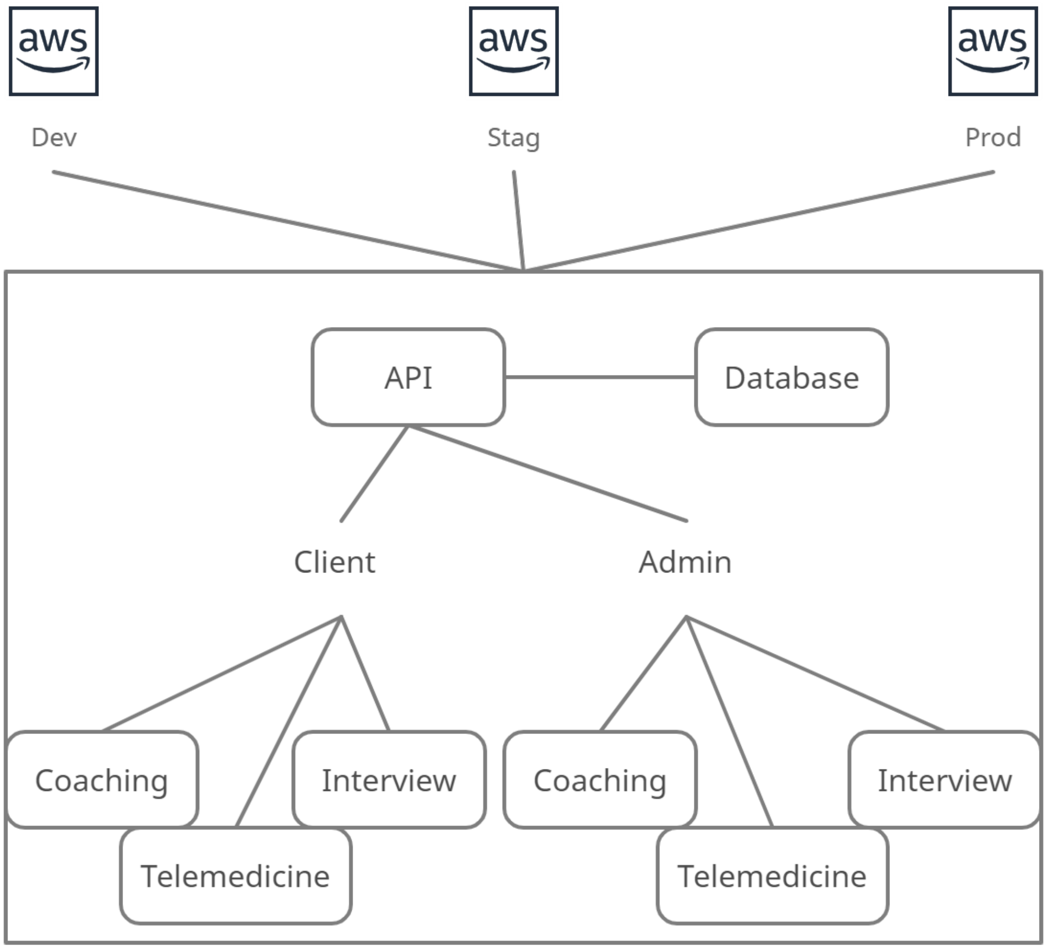


Figure 6. Telemedicine theme Client and Admin.



Database is based on PostgreSQL, which can handle can handle complex queries and massive databases. Docker is used to containerise the applications to make it easier to configure and execute on various operating systems. In more detail, configuring the application development environment is a tedious and time taking, however Docker aids this issue by making the configurable with few commands. This helps to make the development process easier and faster, since developers may use different devices with different operating systems in a company. Amazon Web Services are utilised to create and host all three applications, which also includes migrating then creating database and having database ready on the cloud environment. Figure 7 illustrates the project structure in a simpler way. Rectangle boxes represent containers. API and database are built up and running. Client and Admin texts represent that client-side applications are in 2 versions.

Figure 7. Javelin structure



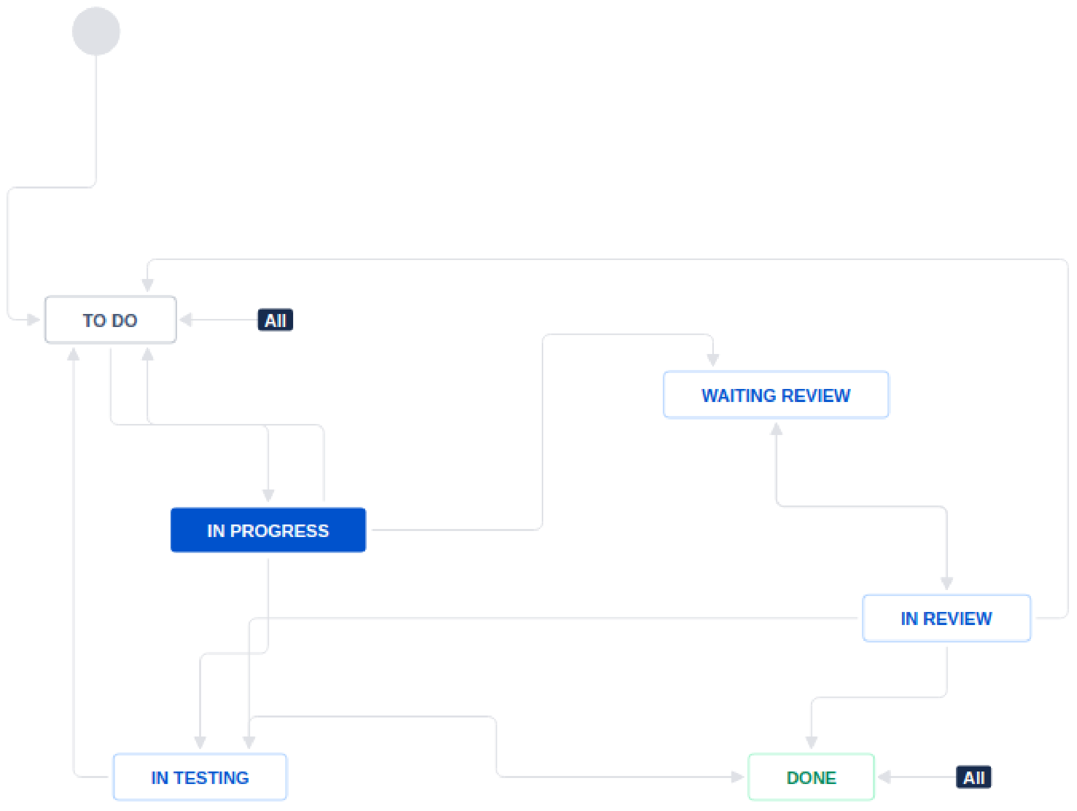
**Agile workflow.** Jira online platform is used to follow the Agile workflow. Slack platform is utilised for communication amongst all the members in the team. The sprint week is initiated on Wednesdays and the assigned tasks are listed in TO DO. The product owner creates user stories and accordingly extracts tasks from them, which then until Wednesday developers assign themselves to these tasks and add story points, where each point is equal to 4 working hours. Developer moves the task to IN PROGRESS field at the time they are working on it. After the task is implemented as thought to be and the developer feels everything is done as expected then they move it to WAITING FOR REVIEW field. One of the developers from the team voluntarily selects the task and assigns themselves as a reviewer in Jira platform and reviews the codelines in the GitLab. Subsequently, depending on the severity (High/None) of the task either it can be merged to master (None) or without merging to master (High) will be moved to TESTING field in Jira. This was decided to make the testers' workload easier since there are many tasks to test in every sprint and depending on the tasks some of them have multi-component or structural changes, where changes can affect the application from several endpoints. Testers usually test the flow of the task according to the task details defined in Jira. High severity tasks are tested first and then merged into master. On the other hand, none severity tasks are reviewed, merged then tested on master branch. Severity of the task is decided by the developer. To be more precise, developer can decide it since they know where and what level changes they have done in the code. If the changes are in structural level it is better to be tested before merging into master (high severity) or if the changes are just fixing a simple line of code, which can be chosen as none severity (by default tasks are assigned with none severity). Additionally, there are tags to make tasks more categorised. For example:

- Theme: Coaching, Telemedicine, Interview
- Environment: Development, Staging, Production
- Application: Admin, Client, API
- Component: Email, Assignment, Question...

These tags make features easier to follow. Javelin being an SPL project which has 3 themes and
The tasks/features are created and the bugs are reported by the product owner or sometimes by the team members as well. Considering that developers are the ones who work the application the whole time and are able to detect most of the errors, bugs, inaccuracies or misinterpretations. Tasks are created with the name JAV - "task name". Here JAV represents the project name. Subsequently, branch names are created accordingly to the tasks, where they end up as a merge request. Finally, every build creates an URL with JAV - "theme code" - "task name". javelin.scr.ee. Themes and codes are as follows: Coaching – DEF, Telemedicine – MED, Interview – INT. To build applications into containers in cloud services Amazon Elastic Compute Cloud (EC2) are used. They are resizable, secure and are configurable with minimal friction. The applications are built in EC2 instances and connected with NGINX proxy withing another Docker container which is listening for a new Docker containers. When they a container is ready and have

their hostnames attached the container which is listening is registering the new containers (with application built in it) as targets for specific hostnames.
Javelin as mentioned before has three repositories/applications: Client Admin and API. Hence, the tasks do not always define which application will be changed. It may be that in a single task 3 of them are affected. Then for each application a branch is created with the name of the task. Then merged into their master branches.
Testers checks the functionalities as expected from all the required endpoints in the UI. Both tester and reviewer can move the task TO DO field if it does not meet the guidelines. The guidelines are defined in project documentation and is growing as the development is proceeding. The workflow is shown in Figure 8.

Figure 8. Jira workflow.



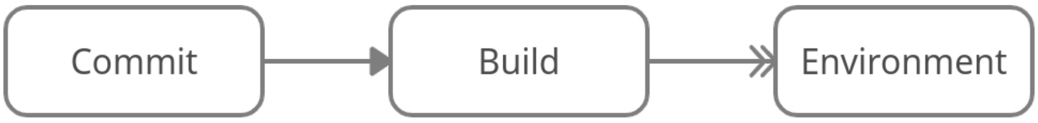
Moreover about CI/CD structure.
**Version Control.** There are several version control systems and GitLab is being adopted by the company. To prevent conflicting workflow and chaotic development an efficient branching strategy is necessary. In Javelin Task-branch development strategy is used, where eventually all the branches after being sufficiently and efficiently completed are

merged into one master branch. The aim is to make frequent small changes each time and committing them according to the task. Smaller and frequent commits clearly identify how a feature was developed, additionally easier to roll back at a specific point in time or to revert part of the change without reverting unrelated changes. This is one of the efficient ways of getting a task into done state.
GitLab provides us with number of services, from which advantages are:

- Hosting code in repositories
- Tracking proposals for new implementations, bug reports, and feedback
- Reviewing code in Merge Requests with live-preview changes per branch
- Building, testing, and deploying
- Integrating with Docker

Moreover, It is possible to integrate GitLab with third party tools. Whenever the branch, merge request that developer created or anything developer is following in GitLab, they get notified with the changes happening. For instance, if the merge request has failed with being deployed, there is a conflict with master branch, deployment has been finished. Additionally, these types of notifications are reported to the Slack bot channel, specifically to follower's and owner's private chat.
**Continuous Integration.** In development team almost everyone pushes code changes every day and maybe multiple times a day. Eventually each of these have to be built and tested automatically to save time. With the CI practice these can be implemented automatically written in a script file (gitlab-ci.yml), which helps to decrease the chances of errors that can occur in the application. Regardless of the benefits the application does not have any testing nor code cleaning automation process.
**CI/CD pipeline.** The CI pipeline for the current situation is as presented in the Figure 9. From commit to building phase the process is automatic. However from built to environment the process is more manual depending on the environment. Only in development the process is automatic. Therefore in the other two, staging and production, the process is manually done.

Figure 9. CI/CD



Each feature is not deployed into production right away. In general deploying to production in Javelin takes place almost once a year for each theme application. This is a manual process with several steps and several hours of work which is a tedious and consists of several manual steps. For instance, requesting certificates, creating SSH keys, configurations on the AWS side and many more.

### 4.3.2 Team organisation

The project team consists of 15 members each having a different role. Eight of them are developers: four mid-level, two interns (including me) and two junior level developers. The details further explained began from September 2020. A new separate team is dedicated to refactoring the whole application. Since new themes were introduced the application code based is need refactoring to better behave as a SPL project. Four developers have recently joined the team since the project is growing. Moreover, the main goal of dedicated refactoring team (including me) is to focus on refactoring the entire project, where the rest of the team will continue working on the existing bugs and new features. There are two testers and automation testing developers team. There is one front-end developer who does all the HTML & CSS coding. The CTO who solves all the DevOps related issues in mostly Javelin and few other projects in the company. Unquestionably every project especially a growing one needs a designer, where there is a dedicated designer for Javelin. Last but not least, there is one product owner who actually does most of the workload. He is the one who usually handles the client calls, distributing the tasks and workload among the team, organising sprint weeks and negotiating between development and the customer. Furthermore, coming up with solutions to any type of project related issues both technical and non-technical solutions are also part of his job. In addition to there, he likes to fix bugs whenever he has extra time.

Table 2. Third party tools used in the application.

| **Name** | **Description** |
|---|---|
| Auth0 | Each of the deployed environments has Auth0 tenant. Each tenant has a personal login URL that will be used for the application login page |
| JotForm | Is a tool which helps you to create forms as widget and integrate into your app. |
| LogRocket | LogRocket provides with log about what users do on the site, this helps to reproduce bugs and fix issues faster. |
| Ziggeo | Ziggeo is a cloud-based video technology SaaS company that provides asynchronous video APIs, mobile SDKs and tools to deliver enterprise-grade WebRTC capabilities. |
| TinyMCE | It is a rich text editor. |
| Stripe | Stripe is a company which offers payment processing software. |

The tools listed in the Table 2 above are utilised in the development of the application.

# 5 Results

In this section, there are four parts. Starting with problems, where they are defined and explained in a detailed way. Subsequently solutions, where after discussions and meetings solutions where defined for each of to be solved issues. Implementations then Measurements, explain the procedures took place to solve the problems and measure the results to demonstrate before and after versions. The steps took place in this section are according to the change proposals timeline described in Figure 3.

## 5.1 Problems

As a result from the meetings with the CTO and I a list of six problems are defined. In the following Table 3, I summarise these problems along with solutions and metrics.

Table 3. List of problems.

| **Problem** | **Solution** | **Metrics** |
|---|---|---|
| No End-to-end test automation | Develop E2E tests, add automation for E2E tests in CI pipeline | Percentage result of passed, failed and warned test suites, Coverage report (requirement: how many of the requirements from the table are covered) |
| No Unit test automation | Develop Unit tests, add automation for Unit tests in CI pipeline | Unit coverage (statement) report |
| Incomplete infrastructure setup | Update CloudFormation script to have more automated steps rather than manual developer dependant work | Frequency of errors, manual work time |
| Development environment limitations with simultaneous deployments | Alter the deploy and build process of applications (admin and client) in a way to minimally use resources and have least amount of failures | Resource consumption for deployment, ram, memory (storage memory was not considered in the measurements, there was not much of a change in terms of file sizes) |
| Slow deploy/build times | Boost the build and deploy time of applications, while reducing resource usage | Resource consumption for deployment, ram, memory |
| Low transparency due to low availability of logs for certain roles | Add a monitoring tool with roles to have logs available and readable for stakeholders | Frequency of created users logging into the system |

1. No End-to-end test automation. Before adoption, End-to-end tests were being developed based on the environment. A dedicated remote team is working to develop these tests. The tests are written with Selenium Python, which later on are executed by the quality assurance team of Javelin. The results are conducted and documented. A reasonable time is needed to be decided for the automation of end-to-end tests. In other words, in which step of feature development can the tests be executed. There are several points in Javelin development process to add testing execution. Furthermore, it is also possible to make tests automatically executed before each git push, after pushing to repository and after or before building. Testing before pushing phase is and can be time consuming for the developer. Additionally, it should be noted that testing step is required to be in both sides of the application, client and admin side separately. Moreover, on of the main problem when testing Javelin is that it is a SPL project. Multiple user interfaces need to be tested. It is desired that the tests are written and scripted to run in a way that code repetition can be prevented while testing interfaces which share the same features. Automating the step at a desirable point in the CI pipeline to test multiple themes (coaching, interview and telemedicine) in multiple environment (development, staging and production) is part of the problem.

2. No Unit test automation. Unit tests are being developed at the moment by the developers of Javelin team. However, prior to the case there were almost no unit tests. It is crucial to have unit tests since they test and approve the smallest single amount of code or a single behaviour and method in a project. The aim is that the tests are easily executable in the local environment of the developers' machine. It is a crucial step to integrate them into software development life cycle, the main reason being that Javelin is constantly growing and changing project. There is a need to take care of complexity in the code base, assure the quality of written features, detect bugs faster and get a code coverage result. Therefore, Javelin being a SPL project meaning that it has only one code repository for each application: client, admin and API. However, currently unit tests are only required for admin. The automation of these tests being added to the CI pipeline is required as well as developing the tests for the application.

3. Incomplete infrastructure setup. Important steps in setting up an environment for the applications to be up and running are done via the CloudFormation script, which is the automated part of the Continuous Delivery (CD) pipeline. However, there are some remaining manual steps that are aimed to be automated. Therefore, these manual actions are DevOps's responsibilities and to make their working life easier the script can be further automated.

   Currently for each staging and production environment a configuration file is needed, for each application: client, admin and API. Another problem is, configuration files are not validated. As an example, the configuration files can be written incorrectly, which may produce errors and waste time. In conclusion, it would be reasonable to figure out mechanism(s) to allow the team to manage the configurations in a better way to save time and can be made sure that the configuration for the particular environment contains all the required settings and in addition has a valid syntax.

4. Development environment limitations with simultaneous deployments. Considering that almost everyday developers push features/tasks and according to the agile flow each branch is deployed as a separate application with it's own domain at the same time connected to the existing, up and running API (which runs database). Therefore, as the applications started growing to multiple themes, utilising more resources, failures began to take place. The problem here is the client and admin has to be built and deployed separately in multiple themes (coaching, interview and telemedicine), as Javelin is SPL. Depending on the feature, relatively how many themes being affected those themes must be built, tested, assured then qualified to be merged into master branch of the application. Hence the problem can be described as follows. The gitlab-ci.yml script file (detected by GitLab runner and executed as jobs) has build and deploy jobs. First, the docker container is built then the application defined (for example, admin coaching) in feature/task is deployed inside it the container. However, it may be that there is something incorrect in the application, hence is throwing an error in the container. Therefore, the container is already up and running with a faulty application, which the application was not even built. Thus, the failures were interrupting the development process in a negative way, slowing down the development life cycle. The discussions took place between CTO and I for several times in several meetings, where the problem details were thoroughly discussed. In a nutshell, the issue is about if a larger number of branches have been deployed to development servers, at a point the server starts failing due to lack of resources. Additionally, there is an additional effort of maintaining EC2 instances that host the Docker containers. Finalising the faced problems:

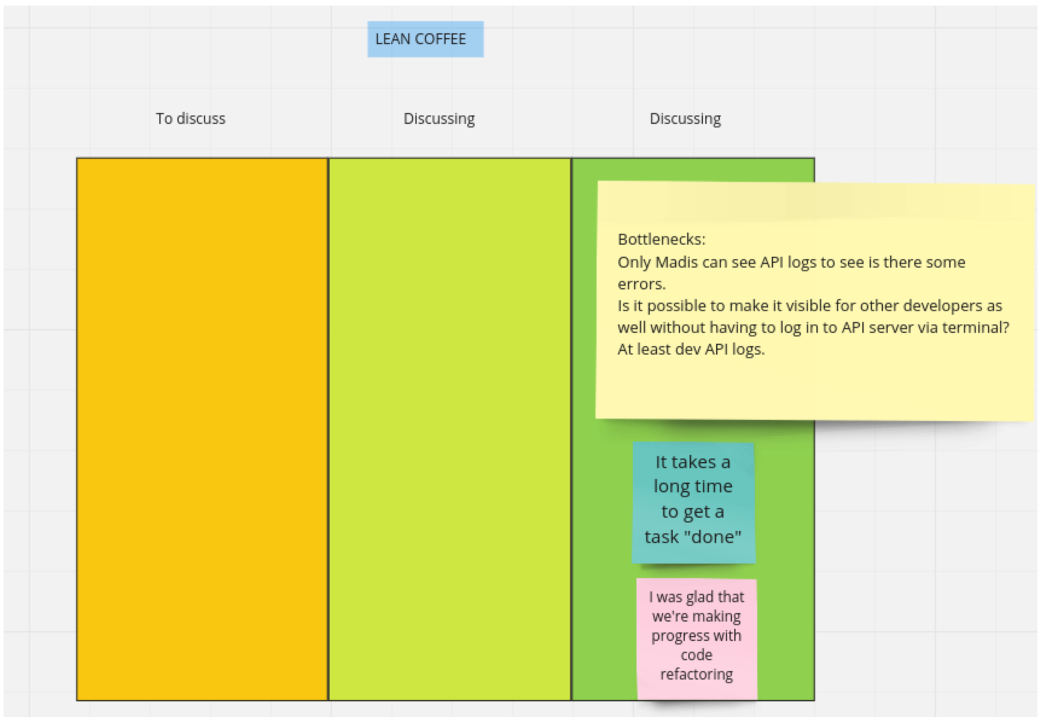
   - According to the CTO of the company the development servers have been manually fixed up to 6 times in a scope of 3 days as compared to before. Major reason of it comes from failures within the development branches.
   - The build is deployed using up more than expected resources (storage, RAM), main reason causing the failures, which these resources are expensive.

5. Slow deploy/build times. Growing project with themes and environments needs to be optimised to have an efficient build/deploy time. The Javelin build and deploy time also began lagging. In development environment everyday multiple

deployments happen by every developer. It is important to have these two steps faster and efficient, to save both time of the developer and available resources. The responsibility, in terms of the development team includes libraries and application structure to be optimised. However, from the DevOps point of the view they need to make sure to implement every possible method to reduce build times. The root cause of the problem is discovered to be deriving from the same cause with the problem 4.

6. Low transparency due to low availability of logs for certain roles. It is necessary for certain roles to have access to logs from the server for several reasons. For instance, debugging and testing. There is no available monitoring in the system for the developers or testers to see the result of required actions, these require immediate glance to logs in server. In addition to these, to gain a system-wide visibility into resource utilisation, application performance, and operational health a monitoring of the application is nice to have. Server monitoring makes it easy for to keep the website activities running up to date. The downtime of the server can also be reduced. Hence, hosting activities are also secured and they can be scaled up effortlessly. Therefore, Javelin is missing this essential practice in its servers. Eventually, one day developers of the team complained about the issue in a lean coffee day, Figure 10, held by the Product owner. Correspondingly, the problem was addressed with a solution.

Figure 10. Lean Coffee Meeting day


LEAN COFFEE
To discuss
Discussing
Discussing
It takes a long time to get a task "done"
I was glad that we're making progress with code refactoring

Bottlenecks:
Only Madis can see API logs to see is there some errors.
Is it possible to make it visible for other developers as well without having to log in to API server via terminal? At least dev API logs.

A simple method on how can the problem affect the whole process of development can be explained as follows. Developer has to ask the DevOps person, what are the result of his interactions. Following, DevOps has to login to the server from his computer using SSH in his machine terminal. After accessing into the servers, he prints out the logs then finds the important message in a humongous list of logs. Finally, send it to the developer. This indeed is both time consuming for everyone sides. Additionally it can be that the DevOps is on their day off. In conclusion, to access or view logs, messages, system failures, errors, resource usages and many others developers are heavily dependant on DevOps.

## 5.2 Solutions

1. No End-to-End test automation. Firstly, decent amount of tests for each theme of the application need to be developed. Where the tests are now developed by a outsource team testing team. Proposed solution:

    - Add jobs to CI/CD pipelines that can run end-to-end tests for both admin and client applications.

However a question is: when is the right time period to execute the testing and what are the required scenarios to be tested? Correspondingly to answer this question a better method of executing E2E tests is to be done manually in GitLab jobs. These tests take considerable amount of time to execute. Considering that Javelin is a SPL where it has 3 themes for 2 applications which makes 6 in total, hence can make development process tougher to run the whole tests in each feature development process. In conclusion, it is decided that the tests are added to CI pipeline however to be executed manually.

2. No Unit test automation. To begin with automation tests need to be written. Then tests need to be executed automatically by the GitLab runner in the CI pipeline. It was decided to write component based unit tests by the development team. It is better to have unit tests run the first thing after push in the pipeline to prevent any upcoming failures. Secondly, it is additional advantage to be able to run the tests in local environment easily. Hence, developers can run them even before pushing to the repository, which can be an extra time save and resource save. Instead running with the runner and occupying it, is better to run firstly in local environment.

3. Incomplete infrastructure setup. The approaches solving this issue can be as follows:

   - Revise the setup flow and figure out the items that can still be included in environment setup script (CloudFormation)
   - Implement the possible CloudFormation items.

   The first step is to create a completely separate repository for configurations. It would contain a set of configuration file templates for each application (client, admin and API). Afterwards, it would be possible to run pipelines to build configurations for each environment with known proper attributes set within the template. If configuration file structure changes for admin, then only the template of that configuration file in the repository is changed then re-build the configuration files for all the environments that are deployed past that point may be needed. At the moment the only issue with this point is that the GitLab instalment does not support cross-project pipelines and artefact sharing. In other words, either GitLab can be upgraded to a paid tier or a way could be found that work-around with reasonable effort. This can allow to achieve similar results without incurring a cost related to user-based license fees.

   Second is to add a testing mechanism for configurations before an environment is deployed. That means before a build/deploy process is initialised, the process needs to have a process in the pipeline, which makes sure a proper configuration file is added. In other words, where all the required variables are set and syntax of the file is properly written. This part is important for preventing faulty configurations from being deployed and does not make the management of the configurations any less cumbersome. Possible solutions extracted from discussions above:

- A step for testing CI/CD configurations in the pipelines against “gold standard” configuration files.
- Building configuration files based on templates, possibly in an external repository, not the application repository.

In addition, a "gold standard" could be explained as company based configuration for network devices in other words a checklist of how all of their devices should be configured.

4. Development environment limitations with simultaneous deployments. From the discussions two solutions were proposed:

   - In the build step first the app is built using *quasar build*, the contents of *dist* folder is retrieved. The retrieved content contains the built application, in details, HTML, CSS and JavaScript files as an artefact (which is a way to retrieve a data or a result from a completed job in the pipeline). Following, in the deploy step, the artefacts' content is uploaded to the development server. However, along this *dist* folder content a new docker-compose file that contains reference to a NGINX (or Apache) image that would serve this *dist* folder content is also required.
   - In the build step, in addition to running *quasar build*, actually a docker image that contains the *dist* folder of the build is built. In simple words, each branch (branches are created for features/tasks) is having its own docker image.

     The second solutions was chose by reason of it does not generate many images that needs to be handled compared to the first solution, this can also take up lot's of resources. More or less this approach is similar to the existing flow.

     The goal is to make smother building and deploying process of client and admin applications, using the caching mechanism to achieve a faster build for each subsequent one. Lastly, the script should work for all the three environments (development, staging and production).

5. Slow deploy/build times. To improve the slow build and deploy time caching mechanism of GitLab can be utilised. As discussed in the problem the main responsibility here is more on DevOps rather than developers. Optimising the gitlab-ci.yml script (responsible for build/deploy process in development environment) to have a smoother process with a decent research is needed.

6. Low transparency due to low availability of logs for certain roles. Amazon Web Services already provides its users with CloudWatch service, which is monitoring and observable service. Hence, the service can be used to integrate into the application development as a solution to the problem.

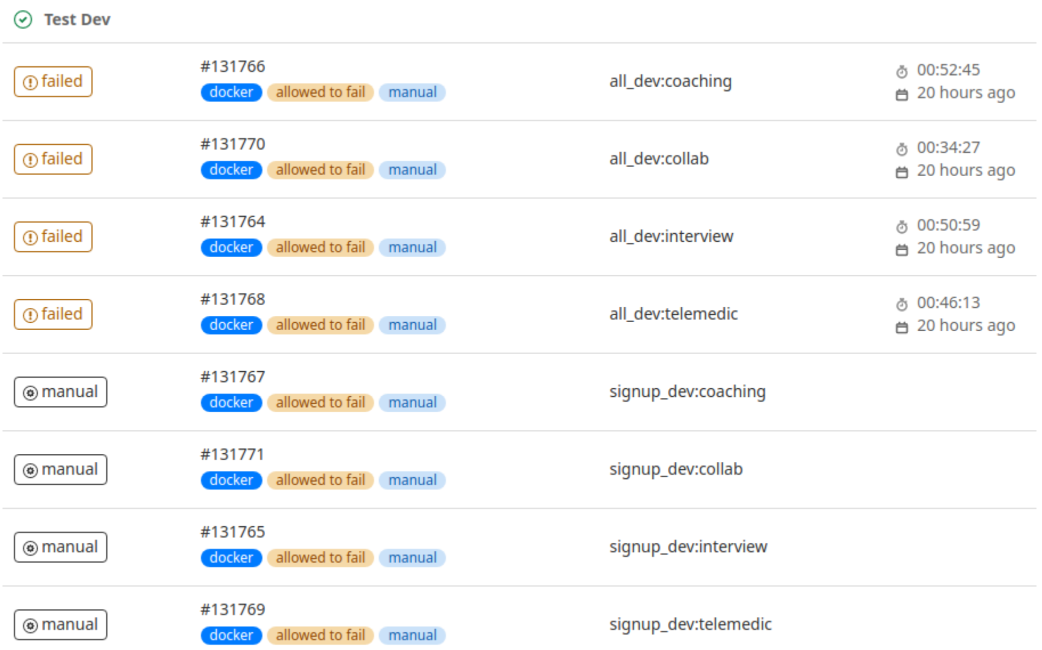
## 5.3 Implementations

1. No End-to-End test automation.

   E2E tests are written by a separate professional development team which works remotely. They follow a TDD approach where Javelin team develops the project and remote team develops tests. I implemented the automation of these tests, adding automation to CI pipeline. In details, adding lines of codes to gitlab-ci.yml file to make the tests able to be executed by the GitLab runner. The tests are in a separate GitLab project, jobs are listed in Figure 11. Hence, the execution process requires jobs to run tests against separately deployed applications (admin) for environments (development, staging and production) and within each environment for each of the themes (coaching, interview and telemedicine).

   The Figure 11 shows the list of jobs in a pipeline that can be executed manually against defined theme applications on both staging and development environments.

Figure 11. End-to-End testing pipeline jobs, GitLab Testing project



Test Dev

| | | | |
|---|---|---|---|
| failed | #131766 docker allowed to fail manual | all_dev:coaching | 00:52:45 20 hours ago |
| failed | #131770 docker allowed to fail manual | all_dev:collab | 00:34:27 20 hours ago |
| failed | #131764 docker allowed to fail manual | all_dev:interview | 00:50:59 20 hours ago |
| failed | #131768 docker allowed to fail manual | all_dev:telemedic | 00:46:13 20 hours ago |
| manual | #131767 docker allowed to fail manual | signup_dev:coaching | |
| manual | #131771 docker allowed to fail manual | signup_dev:collab | |
| manual | #131765 docker allowed to fail manual | signup_dev:interview | |
| manual | #131769 docker allowed to fail manual | signup_dev:telemedic | |

Tests are manually executed, since they take long time to run it is better to run them when there is a need. Yellow failed is a warning sign, which does not mean that tests have failed, mainly related to runner. The second column is GitLab runner related tags. The third column represents the tags explained in Figure 11. The fourth column is the results of how long and when tests ran. Hence, it is clearly visible tests do take considerable amount of time when executed. The amount of time is also growing as the application is developing and more tests are being written.

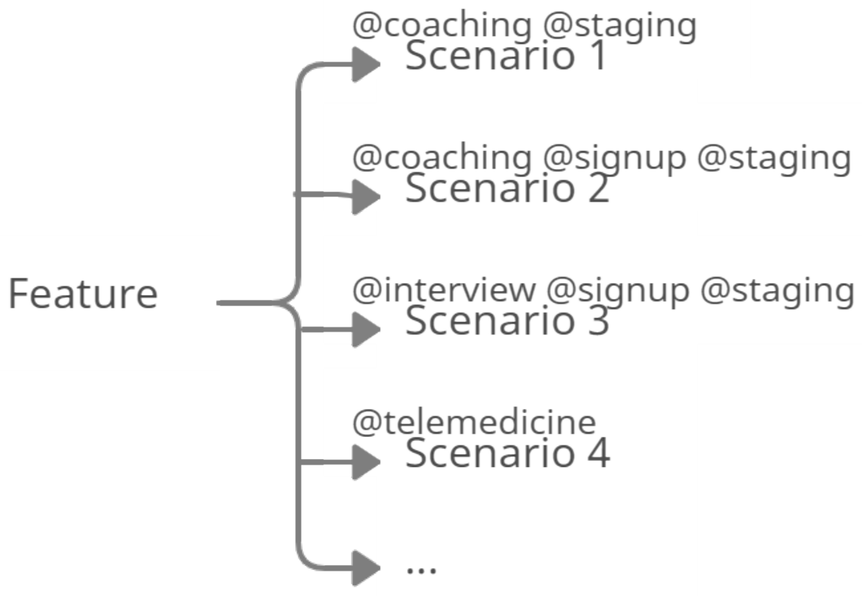
Additionally, to make the tests run faster a separate repository is created in which includes a container image with the necessary tools. Each time when tests run a ready image including tools is pulled from the repository and container is built on top of it. Subsequently, the GitLab runner will run the tests against defined hosts. Hosts are built applications in various themes, for example, Admin application with Interview theme in development environment. The hosts are defined in the script. The jobs are decided to be executed manually by anyone from the team, where I explained the reason in solution 1 belonging to problem 1.

Since Javelin is a SPL project the challenging part is to test all the applications in all environments. For this reason a tag system has been proposed and implemented. These tags are categorised as follows:

- Environment tags: staging, development
- Theme tags: coaching, interview, telemedicine
- Application tags: client
- Other tags: cleanup

The tags are passed through the GitLab script, this is where it is decided on which environment which application will be tested in which theme. Tags rescue process from duplication. Figure 12 illustrates how the tag structure would look like. The passed tags through jobs then are matched to tags/keys defined on scenarios and are able to execute only them. With this a test scenario is executed against multiple environments and themes in a cleaner way.

Figure 12. Tag system in hierarchy.



To note that client application tests are still under development. Additionally, there is no need for API E2E tests since it does not have an user interface.

Allure services are added to see report results of tests. Allure is an open-source framework which generates test execution reports that are clear to everyone in the team. After the execution of the GitLab Testing project's jobs there are reports generated during the execution time then are transformed into a HTML report. There are set of URLs to access and see the results. With this a person with any background can have a look at the reports. The results of Allure are listed in measurements of this problems in measurements section.

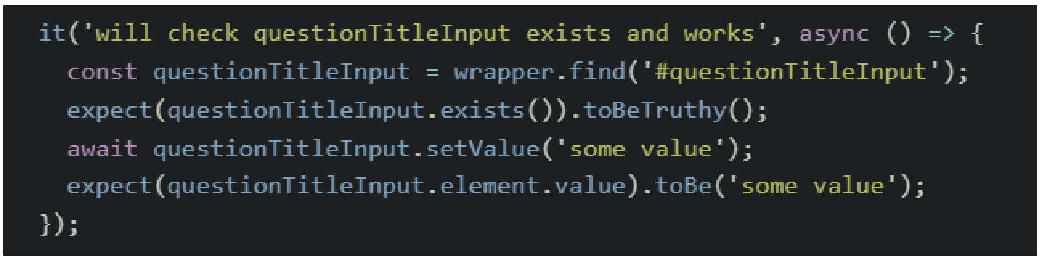
2. No Unit test automation. Firstly, to make this implementation I updated the gitlab-ci.yml scripting file to have the automated tests in GitLab made possible to be executed by the runner. Secondly, before pushing to repository it is made available for the developers to run and get a report of their tests. More about how the tests are written. By using Jest framework a component-based method was applied. Jest is written in JavaScript language. For Vue.js based application it is suitable to utilise. Currently, after each push to repository automatic unit tests are running.

Moreover, if any failure happens pipeline will not proceed to the next step. The next step includes the build and deploy of the application for specific feature or task. By allowing this step happen only after successful result of tests it is possible to save on resources of both GitLab and AWS usage.

Unit tests are part of developers' responsibilities in Javelin. Hence, all the developers in team write unit tests when there is a need for it. Where it mostly depends on the task to be received. Currently, a testing approach is not followed. The reason being that the testing was integrated later in to the project development.

Below in Figure 13 an example of single unit test from Javelin Admin is shown:s

Figure 13. Unit test example



```
it('will check questionTitleInput exists and works', async () => {
  const questionTitleInput = wrapper.find('#questionTitleInput');
  expect(questionTitleInput.exists()).toBeTruthy();
  await questionTitleInput.setValue('some value');
  expect(questionTitleInput.element.value).toBe('some value');
});
```

The Javelin team follows the correct methods of writing unit tests in order to overcome test smells. There are common rules and guidelines on how to write and not write unit tests. The guidelines in Javelin documentation are updated almost every sprint week. If they are followed properly tests smells can be avoided. With code review team members make sure to motivate each other avoiding these smells.

3. Incomplete infrastructure setup. The implementation is done by the CTO. He set up the whole Javelin project AWS configurations, infrastructure and GitLab pipelines. However, my contribution here was to learn, track and analyse the process. The script contains automated steps to be able to deliver product in production/staging. Still there are many manual steps left and some are now added to the script.

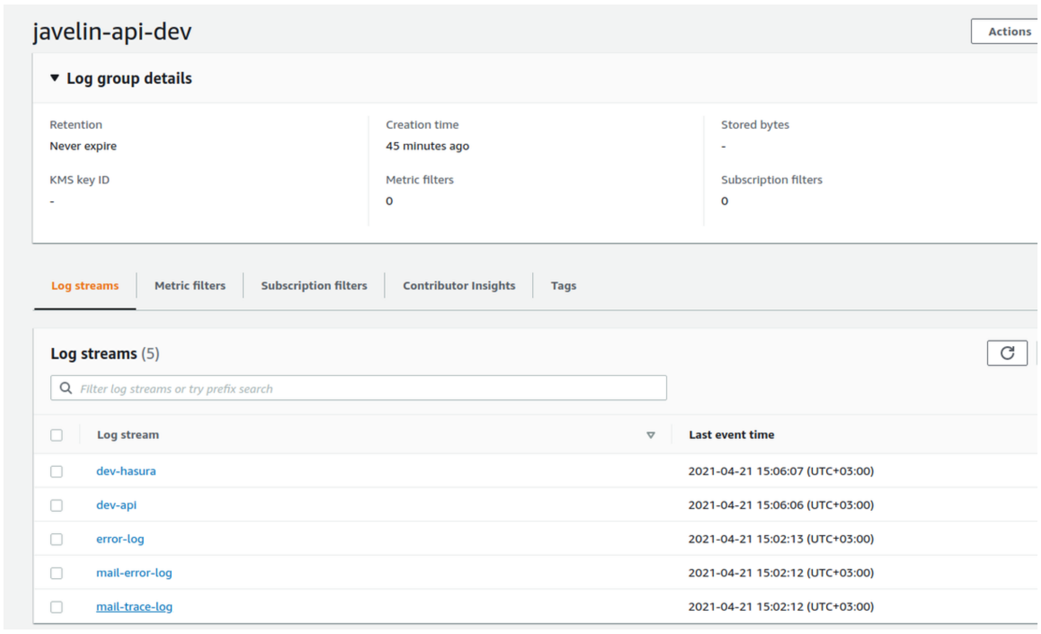
4. Development environment limitations. To implement the solution explained firstly an existing script has to be revised. CTO and I together have revised and discussed the issue in detail. With his guidance I have altered the script to have a smoother, less error-prone and low failure build and deploy process for the applications admin and client. Additionally, caching mechanism is used from previously successfully built files. As a result, every next time when building happens files are retrieved from the cache. This script is for all the three environments (development, staging and production). Despite the fact that firstly the build and deploy processes were

in a separate job, now it was decided to merge the two jobs and have a build and a deploy in the same job for convenience to make it happen with one click. The job is manual. In more details, there are multiple themes, it is best to have it be done manually, otherwise redundant deployments can happen if they were automatically built. Not always all themes are need to be built at the same time. Furthermore, it is not an efficient to have all themes built from the resources wise as well. As a result, a user-friendly (readable) hostname is generated and attached to each development branch, which is deployed and this makes them easily accessible and testable. The generation process of hostnames is explained in details in the Methodology section, in Agile workflow section.

5. Slow deploy/build times. Improving the script in previous problem affected this problem to be fixed in the same process.

6. Low transparency due to low availability of logs for certain roles. The implementation for this problem is quite simple and smooth. I explain the process in simple and in a detailed way. Amazon provides its users with numerous services, CloudWatch being one of them, as well as extensive documentation to follow. Firstly, to implement a policy attached to a role in AWS console is created with the required permissions. In details, which access can this role have in the servers where it will be used. Later the CloudWatch service is installed into container instance, where it is needed. Each container are given a name in the docker configuration file for the CloudWatch. The names are listed in Figure 14 under the Log Streams. The user is attached to the docker container, related files in the container are updated and restarted to allow installed AWS service to gather and transfer data to its console. As a result full list of logs are streamed in the Amazon console. Additionally, these logs can be filtered, alerted and many more functionalities can be added. In Figure 14 each name is clickable and redirects to logs from containers.

Figure 14. Real-time Monitoring Dashboard AWS console


javelin-api-dev

Actions

▼ Log group details

| | | |
|---|---|---|
| Retention | Creation time | Stored bytes |
| Never expire | 45 minutes ago | - |
| KMS key ID | Metric filters | Subscription filters |
| - | 0 | 0 |

Log streams | Metric filters | Subscription filters | Contributor Insights | Tags

Log streams (5)

Filter log streams or try prefix search

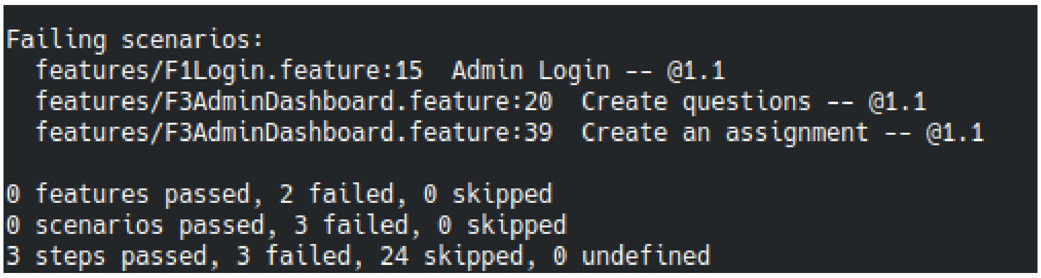
| Log stream | Last event time |
|---|---|
| dev-hasura | 2021-04-21 15:06:07 (UTC+03:00) |
| dev-api | 2021-04-21 15:06:06 (UTC+03:00) |
| error-log | 2021-04-21 15:02:13 (UTC+03:00) |
| mail-error-log | 2021-04-21 15:02:12 (UTC+03:00) |
| mail-trace-log | 2021-04-21 15:02:12 (UTC+03:00) |

## 5.4 Measurements

1. No End-to-End test automation.

   Prior to the development of E2E tests and automation of their execution there were no tests nor automation. However, very simple and three features (Admin Interview) existed prior to implementation, results are in Figure 15.

Figure 15. Coverage report


```
Failing scenarios:
  features/F1Login.feature:15  Admin Login -- @1.1
  features/F3AdminDashboard.feature:20  Create questions -- @1.1
  features/F3AdminDashboard.feature:39  Create an assignment -- @1.1

0 features passed, 2 failed, 0 skipped
0 scenarios passed, 3 failed, 0 skipped
3 steps passed, 3 failed, 24 skipped, 0 undefined
```

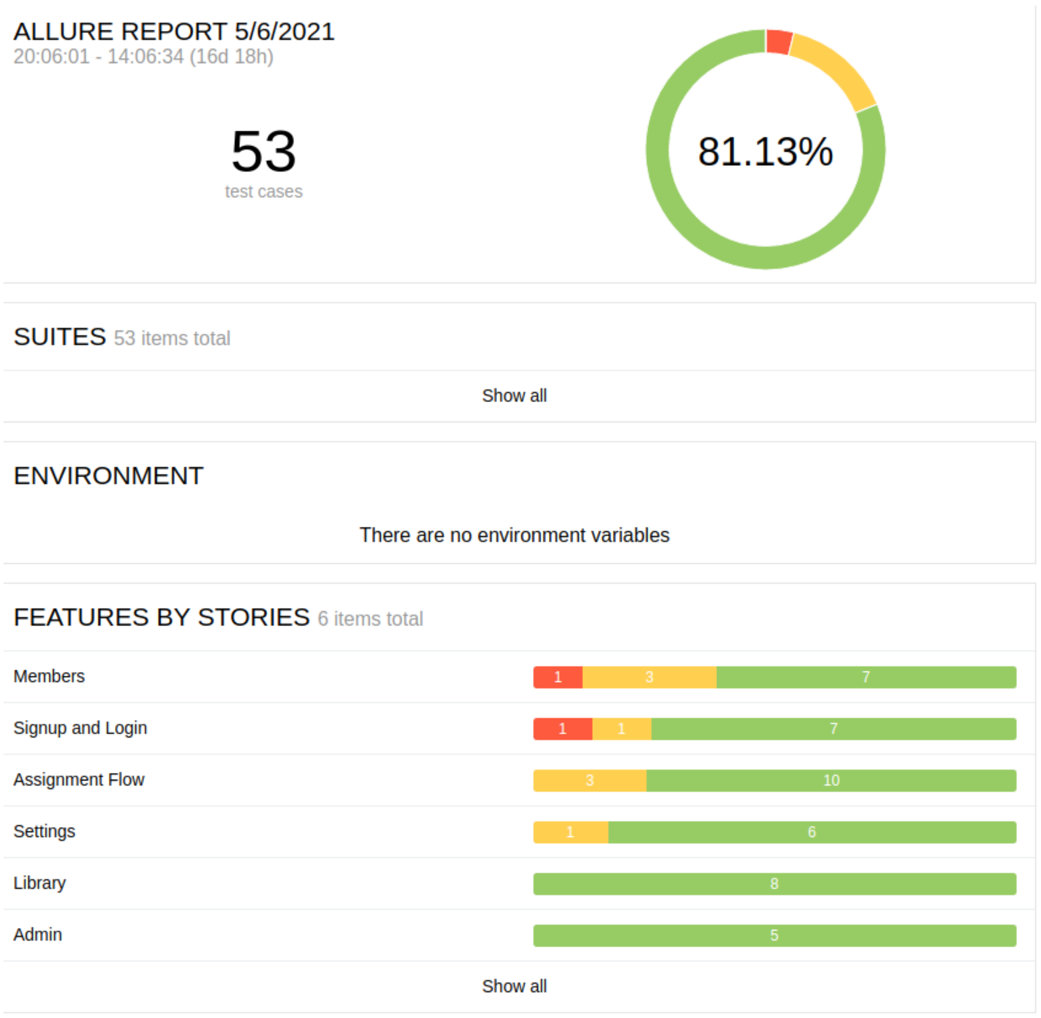
The code coverage report printed and visualised by percentage in Table 4.

Table 4. Coverage report results

| | Statements | Missing | Excluded | Coverage % |
|---|---|---|---|---|
| Total | 498 | 260 | 0 | 48 |

The used Allure service provides with readable and understandable reports. For Javelin the results can be found in Figure 16. It is clearly visible how many test suites are written hence passed (green), failed (red) and are in warning (yellow) conditions. As seen in the picture there are six total features being developed. From 53 test cases 81.13% have successfully passed.

Figure 16. Allure results, Admin application, Coaching theme, Development environment



ALLURE REPORT 5/6/2021
20:06:01 - 14:06:34 (16d 18h)

53
test cases

81.13%

SUITES 53 items total

Show all

ENVIRONMENT

There are no environment variables

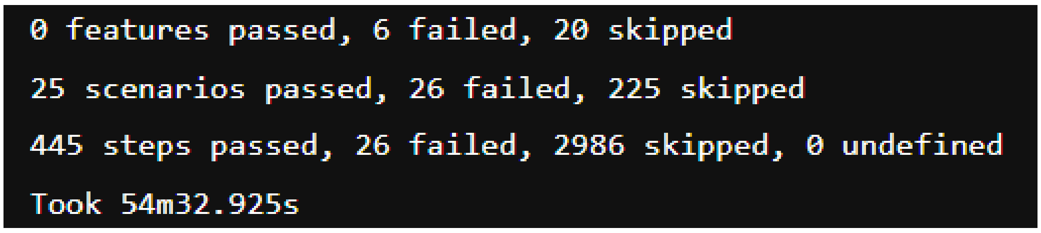
FEATURES BY STORIES 6 items total

| | | | |
|---|---|---|---|
| Members | 1 | 3 | 7 |
| Signup and Login | 1 | 1 | 7 |
| Assignment Flow | | 3 | 10 |
| Settings | | 1 | 6 |
| Library | | | 8 |
| Admin | | | 5 |

Show all

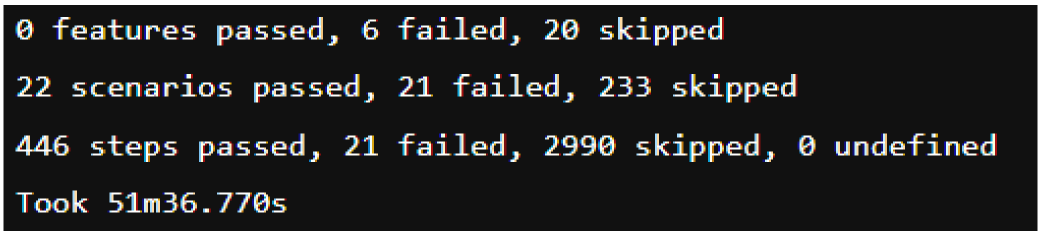
To simply demonstrate the results for other environments and themes I present the end results after tests being executed. How many passed, failed, skipped and how long it took are clearly visible from the Figure 17 and 18.

Figure 17. Admin application, Interview theme, Development environment



```
0 features passed, 6 failed, 20 skipped
25 scenarios passed, 26 failed, 225 skipped
445 steps passed, 26 failed, 2986 skipped, 0 undefined
Took 54m32.925s
```

Figure 18. Admin application, Telemedicine theme, Development environment



```
0 features passed, 6 failed, 20 skipped
22 scenarios passed, 21 failed, 233 skipped
446 steps passed, 21 failed, 2990 skipped, 0 undefined
Took 51m36.770s
```

From Allure reports I list below the results in percentage in Table 5.

Table 5. Allure report results for E2E

| **Theme** | **Test Cases** | **Passed %** |
|---|---|---|
| Coaching | 53 | 81.13 |
| Interview | 48 | 45.83 |
| Telemedicine | 42 | 50 |

I list the results of coverage report in a table for more simpler view for each theme and their relevant results in Table 6.

Table 6. Coverage report results for E2E

| **Theme** | **Statements** | **Missing** | **Excluded** | **Coverage %** |
|---|---|---|---|---|
| Coaching | 6811 | 3850 | 0 | 43 |
| Interview | 4384 | 2988 | 0 | 32 |
| Telemedicine | 6811 | 4233 | 0 | 38 |

2. No Unit test automation. Jest framework is used for developing tests in Javelin, it also provides coverage results by just running the command *npm run test:unit* on the basis of NPM package. Additionally, it prints a HTML report where each component and their coverage report by percentage can be seen.

   Since Javelin project is a SPL type of application it only has one source of code regardless of different themes in both client and admin applications. To add, tests are developed solely for the admin side of the project. Hence, there are reports for only one theme or one source code. I print the list of results in a table below, Table 7. To capture a better before and after result I took measures right after implementation was done with automation and one after several months. This gives comparable results, since tests are being developed gradually.

   Table 7. Coverage report results for Unit tests

   | **Date** | 08.12.2020 | 14.04.2021 |
   |---|---|---|
   | **Statements (%)** | 3.85 | 6.19 |
   | **Branches (%)** | 3.67 | 5.21 |
   | **Functions (%)** | 4.51 | 8.37 |
   | **Lines (%)** | 3.88 | 6.28 |
   | **Test Suites** | 22 | 43 |
   | **Tests** | 159 | 361 |
   | **Time (sec)** | 27.553 | 30.787 |

   The time row in the table shows total time spent on running all the unit tests. The first date column shows right after the automation was added to the pipeline, which means very beginning of unit tests being written. The second date is after considerable amount of time to show the development in unit tests and results.

3. Incomplete infrastructure setup.

   Time and amount of manual work are the main measurements for this practice.

   In old or previous version of setting up the environment the time spent manually was 45 minutes at least for AWS configuration (before and after running script). The script does the following steps:

   - Create S3 buckets
   - CloudFront delivery
   - EC2 instance to host API
   - RDS database for API
   - Firewall rules for auto-deployment (pipelines)

- EC2 to DB communication firewall rules

The script execution time is noticeably less:

Start: 2020-10-21 00:26:23 UTC+0300

End: 2020-10-21 00:28:45 UTC+0300

After the changes are applied more automation to the steps are added and less manual work is required. then the script execution time has increased due to the fact that now it has to do more steps in order to have less manual work that require developer's attention, which is around 30 minutes. The listed items above were not automated as mentioned. The Table 8 below shows which steps have been automated and is not running manually by the developer. Additionally some steps are in the process of being automated.

Table 8. CloudFormation automated steps

| **Name** | **Automated** | **To be automated** |
|---|---|---|
| Request certificates | | + |
| Create SSH key | | + |
| Adding load balancer | + | |
| Listeners and listener rules | + | |
| Target groups | + | |
| Some of the firewall configurations | + | |
| DNS records for the apps | +* | |
| Extract resource info (Database identifier and IP of EC2 instance) | | +** |

* Currently for the production environment servers which belong to Client are used. Hence for DNS records this cannot be automated in production case, there is a manual step that has to be emailed to the client to ask the records. However, for staging this is automated.

** When CloudFormation script is done it can reproduce an output which can be useful in further steps. This can be automated in the future by using this service.

The script execution time:

Start: 2021-02-25 18:52:13 UTC+0200

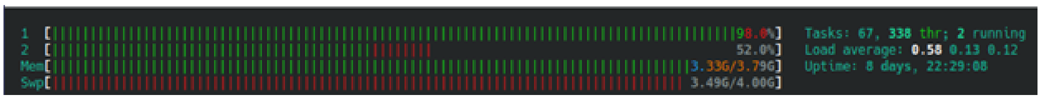
End: 2021-02-25 19:00:17 UTC+0200

Now the time of the script has increased to 8 minutes and the manual time has decreased from 45 minutes to almost 30 minutes.

4. Development environment limitations.

   The build is deployed using up more resources than expected. It is visible from Figure 18 how much resources are being wasted to a maximum usage of Ram and even SWAP. In other words the main cause of failures in the deployments. To explain it simply, when many merge requests take place, they hold up the sources and consequently deployments start failing. Figure 19 displays how much resources are being consumed in a normal day with multiple branches being deployed and queuing to be deployed in development environment.

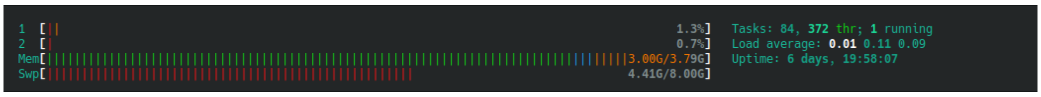
   Figure 19

```
  1  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||98.0%]   Tasks: 67, 338 thr; 2 running
  2  [|||||||||||||||||||||||||||||||||||||||||||||||                             52.0%]   Load average: 0.58 0.13 0.12
  Mem[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||3.33G/3.79G]   Uptime: 8 days, 22:29:08
  Swp[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 3.49G/4.00G]
```

   After successful changes it is visible from Figure 19 that resource consumption has drastically dropped. RAM and SWAP are used and distributed efficiently and are not spiked up to their maximum usage.

   Figure 20

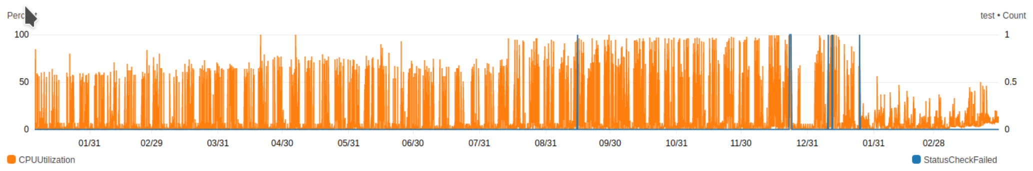
   

```
  1  [||                                                                         1.3%]   Tasks: 84, 372 thr; 1 running
  2  [|                                                                          0.7%]   Load average: 0.01 0.11 0.09
  Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||3.00G/3.79G]   Uptime: 6 days, 19:58:07
  Swp[|||||||||||||||||||||||||||||||||||||||||||                          4.41G/8.00G]
```

   No failure has been reported since. In this problem solution the fifth problem has also been solved indirectly.

   Figure 21 captures the CPU utilization of the development server over 15 months from AWS. In more details, the usage is being dropped drastically after the changes were implemented is clearly visible. The blue lines represent 0/running and 1/failure states. Although, there were more downtimes of the server they were not caught or captured by the system. The reason for this can be explained as the fewer the server is in downtime state it is not caught by the system and logged.

Figure 21

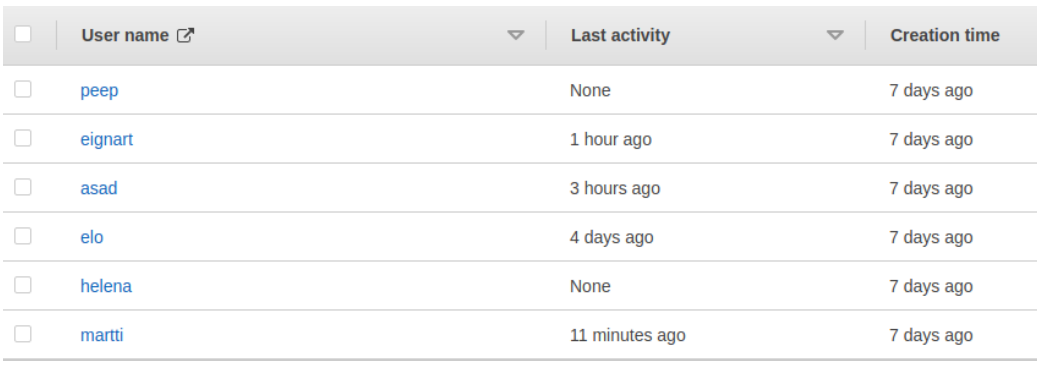


5. Slow deploy/build times. This problem was indirectly fixed in Problem 4. Previously the build and deploy were in separate jobs The script runtime for building the admin application for a single theme took up to 2 minutes. Later deploying to development or staging environment could take up to 3 minutes. These are taken from GitLab runner execution results.

   Although the first build&deploy runtime can take up to 3 minutes for every branch, every other rebuild and redeploy will runtime is decreased to 1 minute 54 seconds. This happens due to caching mechanism used in the script execution process.

6. Low transparency due to low availability of logs for certain roles. The metrics used in this practice is to see if it the practice is being used by the stakeholders and is useful. The results are printed about how many times the console page was logged in by the users who have access. Figure 22, taken on 30th of April, clearly visualises the list of users who have access to the dashboard. The users are Developers (4), tester (1) and product owner (1). Prior to the practice there was no access or tool to see the logs.

Figure 22. Logged users list AWS console



| | User name | Last activity | Creation time |
|---|---|---|---|
| ☐ | peep | None | 7 days ago |
| ☐ | eignart | 1 hour ago | 7 days ago |
| ☐ | asad | 3 hours ago | 7 days ago |
| ☐ | elo | 4 days ago | 7 days ago |
| ☐ | helena | None | 7 days ago |
| ☐ | martti | 11 minutes ago | 7 days ago |

# 6 Discussions

This section discusses the research questions of how Singularity improved DevOps in Javelin project. The results indicate that the software development process can be improved by using DevOps in its recommended way and adopting more practices if needed.

## 6.1 Findings

End-to-end and No Unit test automation. As explained earlier that testing is important part of the development process further automating and configuring pipelines for smoother execution brings more benefits. Test automation is the main automation expertise that may further promote continuous practices [1].

1. No End-to-end test automation. End-to-end testing and automation of it are both essential to reduction of risk. The success of a test is dependent on all the smaller pieces of code that make up an application and ensure that it is working properly [22]. Comparing the before (not having any automated tests) and now (having GitLab pipeline running for automated tests against the admin side application) from the coverage report it is visible how much part of the application is being tested. By tracking the test case statuses, test progress, coverage reports, Allure results, defect details and the availability of the environment it is visible that there is a success.

2. No Unit test automation. Test automation is vital for maintaining software quality in Agile software development, including unit tests. It is visible that there were no tests and no automation prior to the implementation of the practice. Later on the first measurement is done to show the results of very basic tests for components. After some time, developers became experienced with unit tests and consequently more tests and more measures are captured. Tests take only 30 seconds to run, which can motivate developers to run it locally before pushing to repository. Later on when it is executed by the GitLab runner it does not take much resources. Executing these test makes the application more reliable. Additionally, it results in how well the components were developed and much developers followed the guidelines.

3. Incomplete infrastructure setup. The script now allows debugging in less amount of time. For example, if a developer accidentally changed the configuration of the client application to **private** state or **not accessible**, then the responsible person can look at the script or debug and detect what is the misconfiguration to be solved.

   This adds value to make the process of deploying more reliable and less error-prone. Having fewer manual steps allows to have fewer manual actions which leads to fewer occurring errors. However, this script is important for having a smooth and error prone environment setup. The metric for this implementation is quiet thought to visualise. The crucial part is every setup to have fewer manual work. Which prevents many consequences. If there was to be a missing character in any of the copied keys it can cause the DevOps person to spend hours of debugging in order to find out the problem, then restart the whole set up process.

4. Development environment limitations. The positive results from the results of this problems solution show that implementation is quite helpful. The important take is no failures or downtimes are happening. These changes in addition led to using fewer resources, which explains how the Problem number 5 was fixed within this solution. Prior to changes there was a Database, an API, a GraphQL engine and Admin application running on one instance. Where it was thought that the client running on a separate instance can save the application from failures and downtimes, apparently it did not. At most 5 branch deployments were enough to collapse the server. Nevertheless, after changes were applied the usage of resources are downsized to a single Instance. Considering that Client is additionally running in the same instance with others listed services and applications. To add more, now Allure Docker container is also running in this same instance. Allure is discussed in more details in the End-to-End test integration section.

5. Slow deploy/build times. Currently there is no downtime in between build and deploy processes. Deploying before was resulting in service to be failing (down), it was not scalable, build and deploy method were used at the same time. Still the build time is not minimal because when adding more tools, the build time increases. However now more sustainable, less error prone process is taking place the reason being that the build is happening before deployment and if build is not complete with success then the process will not continue to further steps. Previously CTO (DevOps) was logging in and looking at the problem in the server to see if failure was happening while Docker was started via GitLab pipeline. In simple words, it would more sound like "let's push and hope it works", though now the process goes like build happens then if it is verified/correct then Docker can start.

6. Low transparency due to low availability of logs for certain roles. I have collected data from key areas throughout the application to be visualised in the real-time dashboard. At the moment, the most important concern is that the data is available, accessible, shared, and is used to guide decisions and saves everyone's/stakeholder's time. As a result, DevOps, developers, product owners and testers are not dependent on each other for diving into the logs. Errors in the application are easily accessible, filtered and readable for every background person.

In order to answer *RQ1: What DevOps problems does the company have?*, from śeveral discussions and meetings a list of problems was elicited. The list is depicted in Table 3. Subsequently, after researching and deeply understanding them possible solutions and ways of measuring the success were also added to the list. The table containing these problems and their respective solutions was finalised after numerous back and forth meetings conducted with the CTO and I. The table is described in the Results and Discussions' Findings sections.
In order to answer *RQ2 How DevOps practices can improve development at Singularity Creations?*, metrics were defined to measure the process of improvement. These metrics include resources usage and human interactions. They more thoroughly explained in the Results' Measurements section. From the results of each problem it can be noted that the adoption and improvement processes have been successfully completed, although there are still more work to be done.

## 6.2 Lessons Learned

I have worked on improving the development process in Javelin along with CTO. Accordingly, the changes include the work conducted before and after the software's development. The measures and the changes implemented for the whole product development process were based on qualitative and quantitative methods. It is possible to locate, document and address the main problems or weaknesses and change them accordingly. However, there is always space for improvement. More work can be and will be done. Further scaling and improving DevOps processes require more research, study in other words higher level of skill-set. More work can be done includes, for instance, adding integration tests to the API [1], having automated key value pair checking job, developing unit tests for the end-user side of the application as well.
In conclusion, comparing from where I started on this project and how it is now, the amount of improvement is a success. Therefore, there is always space for improving DevOps from both the agile and DevOps perspectives [7].

## 6.3 Limitations

The goal was to examine how DevOps practices were improved in the agile software development process in Javelin and compare the results to get better understanding. On the other hand, not all results were accessible and measurable. To measure problem 6 tracking users data is needed. To achieve such a data AWS Cloud trail can be used. However, it is a paid service and the problem 6 was limited to get a sufficient data. Another limitation was that the project is constantly growing. Almost every sprint week new features are requested by the customer. Thus, it was certainly challenging process to document and capture every change in details of this case study. An experience was required to handle the changes and be able to track the development.

## 6.4 Future Work

In this chapter I present the remaining and more work that can be done in the future. Due to lack of time implementing everything in the best desired way was not achieved. On the other hand there is always space to improve DevOps further more. Future work concerns deeper analysis of particular problems and practices, new proposals to try different methods. The IT field is constantly changing, growing and moving forward, hence better technologies are released and are ready to use. Time, resources and dedication is required to learn and apply upcoming practices into organisations and their project development architectures.
There are some potential proposals for future work that can be done in the Javelin project from the DevOps perspective:

1. Creating a step in the process that checks the key and value pair in client and admin applications.

2. Developing Integration tests for the API application. As I discussed the importance of integration tests in the application in earlier chapters having them in an application is crucial. It helps to easily debug and find the source of a problem when it occurs. Additionally, the client application in the project does not have any unit or end-to-end tests currently.

3. Adding automated monitoring to get performance and availability in every environment, in other words collect metrics across all environments in real time. This is helpful in order to get immediate response from the cloud servers if any uploaded changes broke anything. Furthermore, receiving an early warning is key in understanding the overall health and performance digital assets.

4. Making CloudFormation script to have more automated steps in order to have smoother and less error-prone deployment, as I discussed in The Results section. Because, as applications become increasingly distributed and complex, more automated practices are in need to maintain application availability at the same time reduce the time and effort spent on detecting, debugging, and resolving upcoming issues in operations.

# 7 Conclusion

I finalise the research and present the conclusions of it described in the thesis. The aim of the research was to develop and improve the DevOps practices in a web-based SPL software. The main focus was to figure out what was missing and how to solve them, what could have been improved and what new practices could be adopted. After several meetings and discussions coming to a conclusion a list of objectives was sorted. Furthermore, research has been done and implementation was applied to the project with guidance. Nowadays developing faster, testing regularly and more frequent releases are required. Moreover, this study contains the popular DevOps practices in details such as test automation, improve CloudFormation script for more automation in the CI/CD process, measuring the process of development and infrastructure monitoring.
The results show that the CI/CD process in the project has improved and comparisons reveal how actually they have affected. Although not many practices were adopted in the case study, still the results show that the gradual changes have brought benefits to the project development life-cycle. Furthermore, there are still future works will or can be done gradually which can bring significant impacts for the project or any upcoming project in the Singularity company. For future endeavours, it will be valuable to teams who will work on SPL project to be supported with a research that has brought detailed information about how to improve a such a project or how to adopt DevOps practices. It is helpful in terms of when to actually begin applying practices, when they are needed, how does the automation help the process and what values does DevOps bring. Finally, the more research that could be done the more applies, improvements and benefits can be brought.

## References

[1] Prashant Agrawal and Neelam Rawat. Devops, a new approach to cloud development amp; testing. In *2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, volume 1, pages 1–4, 2019.

[2] Matt Callanan and Alexandra Spillane. Devops: Making it easy to do the right thing. *IEEE Software*, 33(3):53–59, 2016.

[3] Lianping Chen. Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32(2):50–54, 2015.

[4] A. Cockburn and J. Highsmith. Agile software development, the people factor. *Computer*, 34(11):131–133, 2001.

[5] Floris Erich, Chintan Amrit, and Maya Daneva. Cooperation between information system development and operations: A literature review. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, New York, NY, USA, 2014. Association for Computing Machinery.

[6] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 2017.

[7] Aymeric Hemon-Hildgen, Barbara Lyonnet, Frantz Rowe, and Brian Fitzgerald. From agile to devops: Smart skills and collaborations. *Information Systems Frontiers*, 22, 08 2020.

[8] Jon Iden, Bjørnar Tessem, and Tero Päivärinta. Problems in the interplay of development and it operations in system development projects: A delphi study of norwegian it experts. *Information and Software Technology*, 53(4):394–406, 2011. Special section: Software Engineering track of the 24th Annual Symposium on Applied Computing.

[9] Muhammad Khan, Awais Jumani, Farhan Mahar, Waqas Siddique, and Asad Shaikh. Fast delivery, continuously build, testing and deployment with devops pipeline techniques on cloud. *Indian Journal of Science and Technology*, 13:552–575, 02 2020.

[10] C.W. Krueger. New methods in software product line development. In *10th International Software Product Line Conference (SPLC'06)*, pages 95–99, 2006.

[11] Shan Liu, Keming Yue, Hua Yang, Lu Liu, Xiaorong Duan, and Ting Guo. The research on saas model based on cloud computing. In *2018 2nd IEEE Advanced*

*Information Management,Communicates,Electronic and Automation Control Conference (IMCEC)*, pages 1959–1962, 2018.

[12] Nikhil Marathe, Ankita Gandhi, and Jaimeel M Shah. Docker swarm and kubernetes in cloud computing environment. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 179–184, 2019.

[13] Steve Neely and Steve Stolt. Continuous delivery? easy! just change everything (well, maybe it is not that easy). In *2013 Agile Conference*, pages 121–128, 2013.

[14] Marta Olszewska and Marina Waldén. Devops meets formal modelling in high-criticality complex systems. In *empty*, QUDOS 2015, page 7–12, New York, NY, USA, 2015. Association for Computing Machinery.

[15] Nicolás Paez. Versioning strategy for devops implementations. In *2018 Congreso Argentino de Ciencias de la Informática y Desarrollos de Investigación (CACIDI)*, pages 1–6, 2018.

[16] Nicolás Paez. Versioning strategy for devops implementations. In *2018 Congreso Argentino de Ciencias de la Informática y Desarrollos de Investigación (CACIDI)*, pages 1–6, 2018.

[17] Roberto Pietrantuono, Antonia Bertolino, Guglielmo De Angelis, Breno Miranda, and Stefano Russo. Towards continuous software reliability testing in devops. In *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, pages 21–27, 2019.

[18] Ionuţ Criştian Resceanu, Cristina Floriana Reşceanu, and Sabin Mihai Simionescu. Saas solutions for small-medium businesses: Developer's perspective on creating new saas products. In *2014 18th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 140–144, 2014.

[19] Mali Senapathi, Jim Buchan, and Hady Osman. Devops capabilities, practices, and challenges: Insights from a case study. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, EASE'18, page 57–67, New York, NY, USA, 2018. Association for Computing Machinery.

[20] Mitesh Soni. End to end automation on cloud with build pipeline: The case for devops in insurance industry, continuous integration, continuous testing, and continuous delivery. In *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 85–89, 2015.

[21] Carmine Vassallo, Fiorella Zampetti, Daniele Romano, Moritz Beller, Annibale Panichella, Massimiliano Di Penta, and Andy Zaidman. Continuous delivery

practices in a large financial organization. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 519–528, 2016.

[22] Manish Virmani. Understanding devops amp; bridging the gap from continuous integration to continuous delivery. In *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, pages 78–82, 2015.

[23] Johannes Wettinger, Uwe Breitenbücher, and Frank Leymann. Compensation-based vs. convergent deployment automation for services operated in the cloud. In Xavier Franch, Aditya K. Ghose, Grace A. Lewis, and Sami Bhiri, editors, *Service-Oriented Computing*, pages 336–350, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[24] Berlin Mano Robert Wilson, Babak Khazaei, and Laurence Hirsch. Enablers and barriers of cloud adoption among small and medium enterprises in tamil nadu. In *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 140–145, 2015.

[25] Liming Zhu, Len Bass, and George Champlin-Scharff. Devops and its practices. *IEEE Software*, 33(3):32–34, 2016.

# II. Licence

## Non-exclusive licence to reproduce thesis and make thesis public