INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Atualização do Processo de Entrega de Portais

HENRIQUE ANDRÉ COUTO RIBEIRO
Junho de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Update of the Deployment Process of Portals

Henrique André Couto Ribeiro

Dissertation to obtain the master's degree in Informatics Engineering, Specialization Area in Software Engineering

Supervisor: Alexandre Bragança

Porto, July 2022

Dedicatory

To my parents, my sister, and my family, for believing in me and being comprehensive during the whole time, and to my friends, for supporting my efforts.

Abstract

The integration and deployment of an application is a regular activity in many organizations, creating value for a client and helping developers validate their work. This process, once completely manual, became automated nowadays, and it is possible to build, test and deploy an application without any human interaction. Although some organizations might still study the pros and cons, the truth is that Continuous Integration, Deployment and Delivery are a part of many organizations, helping quickly discover errors or deliver a correction or version without disrupting the work of its users.

Grasshopper SI, an organization based in Maia with 17 years, did not have these processes implemented in any of its projects. To evaluate the impact of these practices in the organization, different approaches were studied and then applied the best approach to a single project, while searching for a way to reduce the downtime between updates. This dissertation describes the whole process performed, from research to evaluation, while also evaluating the benefits and disadvantages of applying these techniques.

Keywords: continuous, integration, deployment, delivery, zero, downtime

Resumo

A integração e disponibilização de uma aplicação é um processo regular em várias organizações, ajudando as mesmas a criar valor para clientes e ajudando os desenvolvedores a validar o seu trabalho. Outrora, o processo era completamente manual, dependendo sempre de uma equipa dedicada ou dos próprios desenvolvedores; contudo, hoje em dia existem soluções de automatização, permitindo construir, testar e entregar uma aplicação sem nenhuma interação humana.

Apesar de algumas organizações estudarem ainda as vantagens e desvantagens dos processos, a verdade é que os processos de Integração, Disponibilização e Entrega Contínua são uma parte de várias organizações, ajudando-as a descobrir rapidamente erros ou a entregar correções ou novas versões sem interromper o trabalho dos seus utilizadores.

A Grasshopper SI, uma organização sediada na Maia com 17 anos, não possuía nenhum destes processos em nenhum dos seus projetos. De modo a avaliar o impacto destas práticas na organização, foram estudadas diferentes técnicas e abordagens, sendo que as melhores práticas foram aplicadas a um projeto existente na mesma. Em paralelo, foi estudada a possibilidade de reduzir o tempo de indisponibilidade durante uma atualização da aplicação.

Esta dissertação descreve todo o processo realizado, desde pesquisa até avaliação, enquanto era realizada a avaliação das vantagens e desvantagens das técnicas aplicadas.

Palavras-chave: integração, entrega, disponibilização, contínua, indisponibilidade, reduzida

Acknowledgment

Firstly, I would like to start by thanking Instituto Superior de Engenharia do Porto (ISEP) and all the docents that passed their knowledge to me during the last five years. A special thank you note to my supervisor Dr. Alexandre Bragança, for his suggestions and guidance during this project. Without him, the project would not be as complete as it is.

I would also like to thank Grasshopper SI, for taking me in two years ago while I was only looking for a project to finish my bachelor's degree and for allowing me to grow and apply my knowledge in real-life situations. A special thank you to Pedro Mendes, Jorge Rodrigues, and Nuno Madureira, for believing in all my experiments and ideas, enough to let me develop this project.

To all my colleagues in ISEP, for distracting and helping me when I need it. I hope I did the same when it was needed. I need to thank particularly Ana Rita Fernandes, Catarina Fernandes, João Tomás Rodrigues, and Rui Machado, for the countless hours working together and helping me achieve my full potential. Without their support and encouragement, it would have been impossible to finish this project.

Lastly, to my whole family, especially my parents and my sister. It is always a little difficult to consolidate work, studies, and social life, and I know that without their support and life lessons it would be impossible to finish this stage of my life. For that and everything leading to this project, you have my eternal gratitude.

Index

Figure List

Table List

Code List

Acronyms and Symbols

Acronyms

CI	Continuous Integration
CD	Continuous Deployment
DSM	Design Science Methodology
DNS	Domain Name System
FAST	Function Analysis and System Technique
FFE	Fuzzy Front End
GSI	Grasshopper SI
IaC	Infrastructure-as-code
NCD	New Concept Development
NPD	New Product Development
SVN	Subversion
UI	User Interface
URL	Uniform Resource Locator

Symbols

H_1	Alternative hypothesis
H_0	Null hypothesis

1 Introduction

This chapter contextualizes the main theme of the dissertation with the organization and presents the research questions. It also explores the value that this work will produce for the organization and identifies the main research method.

1.1 Contextualization of the problem

Grasshopper SI, or GSI, is a company from Maia that focuses on tools to help on activities of pharmaceutical companies present in Portugal. With one decade and a half of experience and a good client portfolio, including international clients, GSI is a major national player.

The main products are two web applications, both helping with the communication between pharmacies, pharmaceutical delegates, and pharmaceutical companies. The company also created a base platform for an online store with back-office management, recently used by a pharmacy to sell products directly to its customers. The projects were developed using Java and MySQL, for backend operations, and HTML 5, with CSS, JavaScript, and jQuery. Currently, each project has its source code repository, on Subversion (SVN).

1.2 Problem

For each project, the developers commit all the changes to the main branch when a functionality or correction is developed and tested locally. After that, when it is necessary to deploy the software, a developer must stop its work, compile locally the project, and deploy the solution to an Apache Tomcat server.

This process can lead to some errors while updating the application, caused by human distraction or another factor. It is very time-consuming for a single developer and there is a downtime period associated with the update process. The occurrence of an error while building,

a bug detected after the deployment or even the commit of new functionalities or corrections restarts the entire process, delaying the delivery of new functionalities or corrections.

1.3 Research Questions

The contextualization reveals that the foundation of the problems is the lack of a continuous integration and deployment process. Based on the problems found and the given contextualization, two research questions were extracted:

- How to perform an automatization of the process of source code compilation, testing, and deployment
- How to decrease the downtime period associated with the update process, preferentially using a technique that allows the deployment of an updated version while an old one is still being used and slowly transfer the traffic to the latest version

The solution for the first question will be based on the creation of a pipeline that will handle the integration of an application and its deployment. The implementation will follow the DevOps concept and principles, such as infrastructure-as-code, and the pipeline will be built using a self-hosted automation tool.

The downtime period reduction will be accomplished by containerizing the application, allowing multiple instances on different versions to be executed, while applying one technique that helps decrease the period while switching between application versions.

1.4 Contributions

It is expected a decrease in the workload and resources spent with the deployment of a new application, for current and new projects that may be created. Consequently, developers have one less demanding task to perform, which results in more time spent on other demanding tasks, thus increasing productivity.

Since the tasks are automated, the number of errors caused by a human on any process is expected to be reduced, increasing the reliability of the solution. The reduction of the downtime period is expected to increase client satisfaction.

The automation of the deployment and the decrease of downtime will also help the organization gain a competitive advantage, since it can deliver new functionalities or corrections faster and in a more reliable way, without disturbing the work of any user.

1.5 Research Methodology

The research methodology applied was the design science methodology (DSM). This methodology is characterized by focusing on designing a general solution for a set of problems and it follows three main steps: problem identification, solution design, and evaluation. It is possible to go back and forth between the steps (for example, the artifact design can unveil hypotheses and variables not considered, triggering a revision of the problem identification) (Offermann et al. 2009). Using this methodology also helps design and evaluate a solution that may be used in further works and situations, increasing the solution's range of application.

It was considered using Action Research as the main research methodology, but after some research about the differences between Action Research and DSM (Dresch, Lacerda, and Miguel 2015; Iivari and Venable 2009; Wohlin and Runeson 2021), it is possible to conclude that the Action Research methodology would decrease this work application range and it is not completely contextualized for this problem, since one of the main goals is design and implement solutions, instead of only evaluate a change that the team would do. Furthermore, it is possible to use, to some extent, this methodology combined with DSM, to evaluate the application of the solution.

1.6 Document Structure

In chapter 2, state of the art, it will be discussed other works related to automated pipeline adoption in organizations, as well as existing technologies to perform it. It also includes a contextualization and description of techniques and other terminology that is required to answer the research questions.

Chapter 3 contains a description of how the solution can generate value for the organization, including a FAST diagram and a small analysis under the New Concept Development model.

In chapter 4, it will be presented the organizational requirements for this project and the process to design a solution, including alternatives.

The implementation is described in chapter 5. It is divided into several subsections, each describing an aspect of the project, starting with a roadmap of the whole process.

In chapter 6, evaluation, it is extracted hypothesis and metrics to evaluate the presented work, containing a description of the methodologies and information sources to be used, measurements, and conclusions taken from the analysis of the results.

Lastly, chapter 7 presents the conclusions taken from the project. It includes a description of possible future work, as well as an analysis of the research questions and final considerations.

After the last chapter, the references are presented, followed by two appendixes.

2 State of the Art

This chapter explores works related to the development of pipelines of continuous integration, deployment, and development. It also contains descriptions of the technologies that are related to the work to be done.

2.1 Context

Using automated pipelines to compile, test, package and deploy software is not a new concept. Since the dawn of extreme programming, developers have been using continuous integration tools to check if their changes to a codebase do not disturb the rest of the system and if the codebase quality is maintained (Paulk 2001).

The concept of implementing a line of continuous delivery, continuous integration, or continuous deployment was studied several times since the rise of these techniques. There are challenges associated with the creation of these pipelines, cultural and technological (Claps, Berntsson Svensson, and Aurum 2015; Laukkanen, Paasivaara, and Arvonen 2015; Nath et al. 2018; Chen 2015). From a technological perspective, the architecture of the software itself can be a problem, as well as the current test infrastructure (or lack of it), the absence of time for maintenance, and the number of available tools on the market. Culturally, there are changes to adapt to the new processes, including the adoption of formal methodologies to help during the software development process (Miller 2008; Nath et al. 2018; Shahin, Ali Babar, and Zhu 2016).

Although there are challenges during the implementation, the benefits while compared to manual processes are more than enough to justify the workload, including reduced release cycle, risks and costs, and an increase in the product quality, productivity, and customer satisfaction (Chen 2015; Miller 2008; Nath et al. 2018).

As referenced before, the software architecture can be a problem. The current projects in the organization are built following a monolithic approach, which is a huge agglomerate of code,

dependencies, and layers. This architecture increases the build and testing time, affecting the performance of a pipeline (Shahin, Ali Babar, and Zhu 2016). According to a survey (Shahin et al. 2018), the goal of integrating and deploying a monolith application, even though it is harder than deploying smaller components (microservices, for example), is possible. Some reasons for the usual unsuccess include slow feedback, slow build time, complexity with dependencies, and inconsistent code styles and patterns, which can increase technical debt (Shahin et al. 2018; Naik 2016). Some techniques can help mitigate inherited problems, such as parallel testing and deployment of all artifacts through a continuous deployment pipeline (Shahin et al. 2018).

2.2 DevOps

DevOps is a set of practices used to approximate software development and software operations. Instead of each team being isolated, this technique uses a set of tools and coordination between both teams to improve certain aspects of a software lifecycle, helping to create a unique flow between conception and delivery (Virmani 2015; Artac et al. 2017). For example, by applying a pipeline with continuous integration and continuous deployment, it is possible to help the development team catch errors and obtain info about the quality of the code, while the operations team has a tool that will automatically roll out updates and can roll back on failure.

One of the main advantages, besides better monitoring and understanding of the system, is the reduction of the time between the commit of a change of any nature and the deployment of the same change to the production environment, helping organizations obtain earlier feedback about the said change and reducing the time necessary to publish changes to the client, therefore increasing efficiency and productivity.

Using DevOps usually requires changes in the organization's culture, and it is not advisable a transition from not having DevOps practices at all to full DevOps practices (Virmani 2015). There are no concrete guidelines on which concrete practices or tools to use, and it is important to firstly understand what practices could be worth adopting and for what areas or projects (Zhu, Bass, and Champlin-Scharff 2016).

2.2.1 Infrastructure-As-Code

Since one of the main goals of DevOps is to get the software developer team and operations team closer, it is desirable to have a common language to specify various operations and the overall design of the infrastructure where the software will be executed, working as a blueprint that will be executed and create said infrastructure. This practice is known as infrastructure-as-code, or IaC, and besides helping bridge the gap between the software development and operations associated with it, it is one of the main enablers of automation in DevOps operations (Artac et al. 2017; Virmani 2015).

Using an IaC blueprint and storing it in the repository allows to create, update, and duplicate a production environment in seconds. This can be helpful in test stages since it is possible to simulate the production environment and test it with fewer configuration steps than if it were to manually create the environment (Virmani 2015).

Two of the main challenges are the maintenance and testability, and the proliferation of various tools in the market, each corresponding to a different phase or task, which requires extra attention from the adopters while creating its infrastructure (Guerriero et al. 2019).

2.3 Continuous Integration

Continuous integration, or CI, is a process containing a set of tasks with the main goal of checking, usually after a commit from any developer, if the project is healthy. Normally, this includes building the project and running automated tests, notifying the developers in case of failure, and it is usually triggered by a change on the source code repository.

According to Martin Fowler (Fowler 2006), one of the advocates of this practice, there are ten key practices to properly use CI, including automatic building and testing, transparency, and fast build and test execution. Another key factor is to keep the build in a green state, which means that, upon build failure, the main priority is to understand the cause and fix it. If this principle is ignored, the source code can be changed by other developers that may or may not use the broken functionality, creating a snowball effect that will increase the correction process time and effort (Meyer 2014).

By following these principles, a CI pipeline can easily detect failures, guarantee that a project is ready to be deployed at any time and increase developers' awareness and sense of responsibility towards the code. It also creates the habit of creating code that can be committed and merged into the main branch regularly (Meyer 2014).

2.4 Continuous Delivery

Continuous delivery, or CDE, is often associated with continuous integration, and its main goal is preparing the application to be deployable at any given moment. To achieve this, after the build of the application it is expected to execute various automated tests, such as performance and acceptance tests, to check if the application is ready to be deployed. After this automated phase, a responsible must manually activate the deployment process, making the pipeline execute its final steps and deploy the code into a selected environment.

Adopting continuous delivery practices from scratch requires a lot of effort and changes. From a technological perspective, it is desirable to have fully automated tests, including acceptance tests, to make sure that the product works as expected. The organization which adopts this practice might need to perform changes to its development/deployment flow, and it is

necessary to adapt the overall company mentality to prepare its code and tests to be checked and deployed at any moment (Chen 2015; Nath et al. 2018).

2.5 Continuous Deployment

The automatization of the process that deploys an application into production, without any interaction from a developer except the one that triggers the build, is known as continuous deployment, or CD.

It is frequently associated with continuous integration and mixed with continuous delivery. The main difference between CD and CDE is that the process is fully automated, and the deployment takes place without human interaction (Shahin et al. 2017; Nath et al. 2018). This also means that every time a change is committed to an observed branch, that change will become available to the customers. With this practice, the release cycles will decrease, which allows a company to quickly adapt to new challenges and requirements. The reduction of the release cycles also helps receive faster feedback about the latest changes and can increase customer satisfaction, assuming that it is desirable to have all the new developments automatically deployed (Leppänen et al. 2015).

The same challenges that are presented on CDE are present in this practice, but with an extra: since a developer can trigger a deployment when they update the repository, it is necessary to keep the code quality above a quality gate (if it exists) and that the existing tests scenarios cover most of the flows.

It is expectable to have bugs shipped to production with this practice, although on a lower number. Since the software is deployed more frequently than on a traditional software release cycle, those bugs are found faster, sometimes by the customers (Claps, Berntsson Svensson, and Aurum 2015).

2.6 Containerization

To deploy an application and make it available for a user, it is necessary to have a host that will execute it and accept outside requests. For web applications written in Java, this can be achieved using Apache Tomcat, for example, hosted on a virtual machine or server.

However, it can be necessary, on a single host machine, to have different environments available, each with different configurations. The usage of virtual machines was quite common in these cases, each one running a configured environment, but it requires the startup and maintenance of a whole operative system inside another, which impacts the overall performance.

A more recent approach is the usage of containers. Instead of virtualizing a full operating system, smaller systems, or containers, run on a shared kernel. Each container is created and configurated only using the necessary components to perform its tasks while being kept isolated from the machine (Watada et al. 2019). For instance, it is possible to create a container that executes a Tomcat instance with a Java application, while another container only provides a database. With this, it is possible to create lightweight containers to deploy in case of failure of a previous container, for instance (Guyton 2019).

Using containers helps reduce unnecessary usage of machine resources and startup times (in comparison with complete virtual machines), while increasing isolation between different environments and providing a good startup point for scaling. Besides that, it is possible to create and manage a container through code and integrate the concept on a continuous integration/continuous deployment pipeline (Gandhi 2019; Guyton 2019).

However, it is worth noticing that, since the kernel is shared through all containers, if there is a failure, it will affect all the containers. The monitoring process is considered harder and, since the container is, by default, isolated, network connections can be harder to manage (Guyton 2019).

2.7 Zero-downtime deployment

During the deployment of an updated version of an application, there is usually a downtime associated, on which the final users cannot use the software. There are deployment techniques that drastically reduce the waiting time between versions, increasing client satisfaction, usually referred to as blue/green deployment. Besides the reduced downtime, another advantage is the instant rollback in case of failure, if both environments are online.

All the techniques share the same core idea: an updated version of an existing deployed application (green) is executed on another container, while the first (blue) is still alive and responding to requests. Then, based on the applied technique, the new application starts to (partially or completely) receive the new requests instead of the old one, while the old application is disconnected from the network.

2.7.1 DNS based

In this technique, both applications are running, and the DNS is routed to the blue environment. When the green environment is deployed and ready to use, the DNS routing is changed, and it points to the green version (Rudrabhatla 2020).

2.7.2 Load balancer based

Also known as a canary deployment (or blue/green deployment in canary style), this technique uses a load balancer to send traffic between the two versions. Initially, the load balancer will only know the blue environment, sending it all the traffic. When the green is deployed, the load balancer will start to send requests to the green environment, until something triggers the change and removes from the load balancer the references to the blue environment (Rudrabhatla 2020).

This technique is useful to test the updates on a few selected users, randomly or given a categorization (location, for example).

2.7.3 Service Discovery based

A technique implemented in a published journal article (Yang et al. 2018) uses an API gateway and a service discovery to distribute new versions. After checking if the green environment is ready, the old environment is tagged as blue, and both receive requests during a certain period. Gradually, the green environment will accumulate more requests, until the API gateway stops sending requests at all to the old environment.

2.7.4 Comparison

All the presented techniques reduce the downtime during an update, but they differ in terms of effort to implement or perform. For instance, the DNS-based technique, although it is quite simple, requires a bigger swap and roll-back time. The load-balanced technique is also easy to implement and has a reduced swapping time compared with the DNS-based technique but requires changes to the load balancer while redirecting traffic over time (Yang et al. 2018; Rudrabhatla 2020).

The Service Discovery based technique, in comparison with the other two techniques, requires the implementation of two extra components, but the new version will be automatically used, and it is only necessary to shut down the old environment (or mark it to not be used by the Service Discovery) to complete the process (Yang et al. 2018).

2.8 Automation Servers

This section describes existing automation servers on the market. This analysis was only performed on those who provide a self-hosting option, and all of them are compared in the final section.

2.8.1 Jenkins

Jenkins is an open-source automation server, written in Java, which allows developers to perform integration tasks such as building and testing their code, and even deploying software. Due to its open-source nature and openness to the community, it contains a plugin library that extends its basic functionalities even further. This fact, allied to the fact that it runs on any machine that can execute a Java Virtual Machine and its years of maturation, allowed Jenkins to endure and continue as a major player when it comes to automation servers.

A user can design a pipeline using its various plugins, allowing the definition of steps from the code checkout until the deployment, but the configuration would only live on the Jenkins server. The newest version of Jenkins allows developers to define pipelines using a proprietary domain-specific language (DSL), which improves the communication between teams inside an organization and allows its definition on a project repository (infrastructure-as-code) ("Jenkins 2 Overview" n.d.).

However, Jenkins is often categorized as one of the less user-friendly integration tools, due to its old user interface, or UI. The plugins library, although impressive, contains several plugins outdated, deprecated, or abandoned, and Jenkins itself requires some plugins for basic functionalities (for example, to connect to Git repositories). Finally, the declarative pipeline (one of the pipelines definition types) does not support all existing plugins and there is outdated documentation about Jenkins and how to operate it ("Advantages and Disadvantages of Jenkins - Tutorial And Example" 2021; Osterman 2020; "Continuous Integration (CI) Tools Comparison: TeamCity vs Jenkins vs Bamboo | AltexSoft" 2019).

2.8.2 TeamCity

TeamCity was created by JetBrains to compete in the automation branch. It allows the user to design its pipelines and automate the integration and deployment cycle, like other tools on the market. Although developed by a private company, its usage is free, even for commercial use, only requiring a license when a company requires more build configurations or build agents ("TeamCity: The Hassle-Free CI and CD Server by JetBrains" n.d.).

Each project can contain multiple configurations (up to 100 total, across all projects, on the free version) and the configurations can be managed by coding, in XML or Kotlin, and stored in the project repository.

It contains a variety of built-in functionalities and statistics boards. Some of the out-of-the-box language options to build projects are .NET, Java, Ruby, Node.js, among other languages, and the Docker integration is strong, allowing projects to be built inside containers and create images while running the pipeline, for example ("Integrating TeamCity with Docker | TeamCity On-Premises" 2021).

The user interface is modern and, as referenced before, contains a lot of options, but it can be tricky at the beginning while configuring the projects, thus increasing the learning curve. Finally, the development is performed by JetBrains, which means that the company is responsible to implement new functionalities ("Continuous Integration (CI) Tools Comparison: TeamCity vs Jenkins vs Bamboo | AltexSoft" 2019).

2.8.3 CircleCI

CircleCI is a product from a company founded in 2011 with the same name. It allows an organization to execute a continuous integration server in the cloud or on-premises. In the cloud, it has a limited number of minutes that can be expended on a free account, which can be expanded with credits bought directly to the company ("Pricing and Plan Information - CircleCI" n.d.).

Besides the built-in functionalities, including Docker support and support for popular languages, its functionalities can be enhanced by installing plugins, or Orbs as the company calls. The Orbs are YAML configurations of tasks that can be replicated by various users. Since the platform does not have built-in continuous deployment integration, third-party Orbs are a way of achieving a CI/CD pipeline on this platform (Chubb 2021).

It is ready to automatically run parallel pipelines, cache project dependencies to decrease workload and network traffic, and allows debug build issues through SSH (Jethva 2021).

2.8.4 GitLab

Launched in 2014, GitLab is a tool that can be used as a source code repository (Git-based) and an integration/deployment server. It was designed to be used by all the organization members, since it contains tools to manage teams, plan activities (including interactive boards), pipeline creation and maintenance (through the platform or as code), and monitoring tools ("Maturity | GitLab" n.d.).

Although open source by nature ("GitLab.Org / GitLab · GitLab" n.d.), it contains different subscription models, even in a self-hosted version, and each version contains even more solutions to help the organization manage itself ("Self-Managed Feature Comparison | GitLab" n.d.).

2.8.5 Comparison

Each tool presented in this section has the ability, with or without external integration, to perform a full continuous cycle. The main factors for the organization and the tool supports are presented in table 1.

Table 1 – Comparison between different automation tools

Tool	Price	Allows self-hosting	Supports SVN projects	Compatible with Java applications	User-friendly configuration	CI/CD pipelines support	Job parallelism support
Jenkins	Free	Yes	Yes	Yes	No	Through plugins	Yes
TeamCity	Free with limitation	Yes	Yes	Yes	Yes	Built-in support	Up to three on free
CircleCI	Paid	Yes	Yes	Yes	Yes	Bulti-in for CI, CD requires plugins	Yes
GitLab	Free	Yes	No	Yes	Yes	Built-in support	Yes

Jenkins, one of the most mature runners, has a huge library of plugins that can enhance its performance and functionalities, and its open-source nature is attractive and allows the platform to grow. This library, however, contains deprecated, not useful, or incompatible plugins, creating an obstacle while configuring the project. The user experience and first configuration can also be a little overwhelming, and the initial plugin's installation, although better in the latest version, may difficult the installation.

CircleCI, with its automatisms and multiple integrated environments, can reduce the overall time of testing. But the payment model is not attractive, and it lacks built-in continuous delivery support.

GitLab provides several functionalities that allow a company to work on the platform, but it can overwhelm a team that only requires a CI/CD pipeline. Besides that, it works only with Git repositories, which would require extra steps for companies with, for example, Subversion repositories.

TeamCity offers, with some limitations in the configuration, a good user experience (configuring and using) and diverse options to integrate and deploy a project, after passing the initial learning curve.

3 Value Analysis

This subsection describes the value analysis performed to analyze how the work will be precepted by the organization.

3.1 New Concept Development

The New Concept Development, or NCD, is a model that tries to solve the lack of common communications terms between the Fuzzy Front End (FFE) and New Product Development (NPD), while also helping identify an opportunity and generate an idea and concept to seize it (Koen et al. 2002).

This model is composed of three parts: the engine, representing aspects controlled by the organization like leadership and culture; the influencing factors, representing outside factors just as the law and economic affairs; and, finally, the inner spoke area, formed by five controllable elements. The inner elements are graphically represented on a circle, to emphasize the idea that the process is not linear, and it is plausible to jump from one element to another, if necessary.

There are five inner elements: opportunity identification, opportunity analysis, idea generation and enrichment, idea selection, and concept definition. In this chapter, the first two are described and applied to the current work.

3.1.1 Opportunity Identification

The main goal in this element is to identify an opportunity that the organization would want to analyze and take advantage of. It can occur using formal methods, like meetings to brainstorm ideas to improve the organization, or by other methods, like forums.

As described in the previous section, each project in the organization has its source code repository. After testing locally, the performed changes are committed to the repository. When it is necessary to deploy an updated version, a developer in the company must interrupt its work, compile the project locally and only then update the server with the latest version. Since the process wastes the organization's resources and is a competitive disadvantage, it was considered obsolete and an opportunity to explore alternatives to the process arise.

It was performed a trend analysis in the field of the deployment automation process, to understand how other organizations are adjusting their old processes to the current paradigm, and most of them are adopting one or more principles of DevOps, including integration and deployment pipelines. Multinationals technological companies, like Netflix, Amazon, Facebook, and Etsy decided to follow approaches to modernize and become more competitive in their respective markets. By implementing automated pipelines and applying DevOps principles, the companies were able to develop and deliver software faster and more reliable (Null 2021).

Etsy, for instance, implemented an automated deployment pipeline and was able to deploy multiple times a day instead of twice a week (Null 2021). Sony Pictures Entertainment had a huge time gap between finishing software and releasing it to the market. After applying a continuous delivery model, it was reported the reduction of this period to mere minutes, which helped developers focus on other major tasks (Null 2021).

The Lloyds Banking Group, with the automation of repetitive tasks and usage of a third-party DevOps tool to perform CI/CD tasks, was able to cut to almost half the number of days to test and they are now able to update their products by 200% ("Implementing DevOps: 3 Surprising Success Stories | Pega" 2018).

3.1.2 Opportunity Analysis

In this element, it is important to analyze the opportunity and evaluate if the effort to pursue it will pay off in the future.

After some research, it was possible to obtain a comparison between organizations adopting or not DevOps practices. According to the research, if an organization does not use DevOps, the tasks are 41% more time-consuming in comparison with a company that uses DevOps, and 21% of the time spent is in correcting errors. By using DevOps, most of the organizations (63%) affirm that the code quality improved and can deliver new software on a more regular basis ("DevOps Stats for Doubters | UpGuard" n.d.).

Using automated tools to perform the integration and deployment of the application helps the organization become more competitive in the market, since the full process is expected to perform faster and become more reliable, while delivering to the client new functionalities on a more regular basis. It would also help increase productivity and decrease the time spent on repetitive tasks.

There are various solutions on the market that allow the creation of continuous integration and delivery pipelines and processes to update services with a decreased obligatory downtime. These tools and methods were described previously in sections 2.6 through 2.8, presenting information about the most prominent artifacts and comparisons.

3.2 Function Analysis and System Technique

The Function Analysis and System Technique, or FAST for short, is a technique that defines and represents product functions, while creating a logical relationship between them and answering why is it being done. It is useful to check if a function is missing, if there are duplicated or unnecessary functions, and what is the priority of the tasks (Dannana 2020).

Graphically, on the left side, there are cards representing why something is being performed, and on the right the functions to be implemented to fulfill requirements. When reading from the left to the right, it should be questioned how something will be done, and while reading from the right to the left, it is important to question why a certain function is being implemented. The order in which the cards appear vertically represents the implementation hierarchy, with the first cards being tasks that are more urgent and should be firstly implemented, thus representing the priorities of the project.

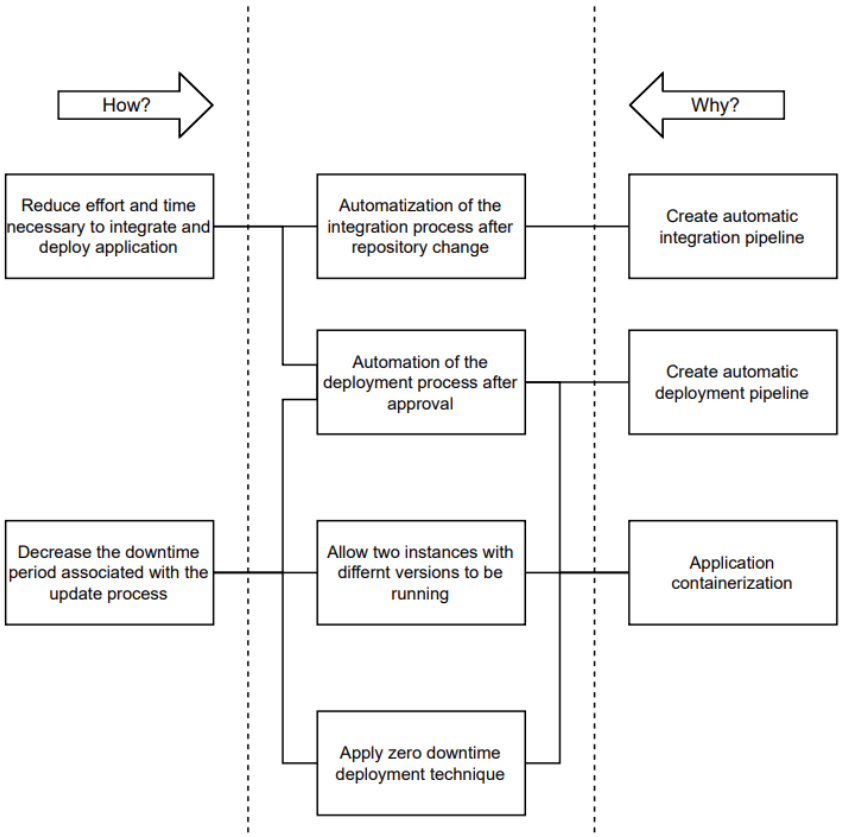

Figure 1 – FAST diagram

As demonstrated in the figure above (figure 1), the main priority is to automatize the integration and deployment process, and after that the downtime decrease. Inside the automation process, the main priority is to implement a continuous integration pipeline, and only after that, the deployment pipeline must be analyzed. Associated with the deployment process, two dependencies overlap the downtime period reduction: the automatic deployment pipeline and the application containerization.

After the implementation of the continuous integration pipeline and the deployment pipeline, the main goal is to decrease the downtime period. Two of the required tasks are already done since they were also dependencies of another requirement, which means that the last step is to study and apply a zero-downtime deployment technique.

3.3 Value for the Organization

From a utilitarian perspective to analyze the value, the organization's perception of the value of this work can be insignificant at the beginning, since the implementation has associated costs, such as the time spent on implementing the process and the additional workload on the developers.

However, in the medium to long term, the prevision is that the productivity will go up and the time-consuming tasks associated with the deployment will be drastically simplified and less time-consuming, which means that the company resources usage is improved.

The following figure (figure 2) represents the benefits taken from the project using the Osterwalder value proposition model. This model is used to represent the relationship between the customer needs (desirable gains, current pains, and tasks to be performed) and the product/service to be delivered (service/product to be presented, how it relieves pains, and what gains are in using them) (Mansfield 2019; Osterwalder et al. 2014).

Figure 2 - Osterwalder Value Proposition

4 Analysis and Design

On this section, it will be presented and discussed different approaches that can solve the research questions previously presented. The chapter starts with a small description of the current integration and deployment process, followed by a list of requirements and, finally, suggestions for the solution design.

4.1 Current integration and deployment process

Currently, the organization follows a completely manual approach to deploy an application. After receiving the request to deploy a recent version of the application, the developer must stop its work and update the current project to the latest version. Since the developer will execute this process locally, it may be necessary to stash current changes that may have been made by the developer, since there is a chance that they were not properly tested and can interfere with other functionalities.

After that, the developer will compile the project and execute previously defined automated tests, such as unit tests, culminating with the manual deployment of the project to the test environment. This deployment is performed by copying the compiled source into a server with an Apache Tomcat instance being executed, reloading the application, and making it unavailable until the reload is complete. When the reload process starts, all current user sessions are deleted.

After testing the changes, the application is manually deployed to the production environment (repeating the copy process previously explained), finishing the full process. In case of failure in any of the phases described before, the process stops, and corrective measures are executed. After the corrections are performed, the process starts over.

The activity diagram below in figure 3 represents the current manual deployment flow, on a best-case scenario.

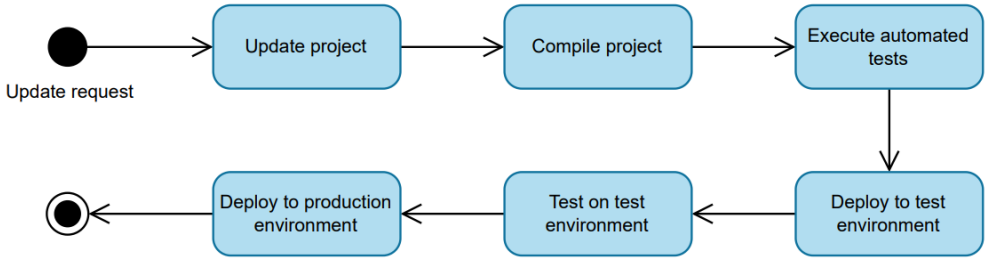

Figure 3 – Activity diagram representing the current deployment process

4.2 Requirements

To design a top-notch solution, one of the first steps to be performed is the definition of the client's requirements. Each requirement can be cataloged between two categories: functional, which describes what the system must be capable of performing, or non-functional, which describes expected behaviors while the system is performing an action (Glinz 2007). The non-functional requirements can be categorized according to their nature, being presented as subsections on this section.

4.2.1 Functional Requirements

The two main goals of the project are the full automation of the integration and deployment process and the decrease of downtime while updating an application.

When the process finishes, requires human intervention, or there is a failure, it is necessary to warn the proper users to monitor or perform corrective actions. This can be done by electronic email. The deployment to the production environment cannot be done without manual acceptance tests. The automation service must be able to deliver the recent version into the production or test environment.

Figure 4 represents the use cases extracted from the analysis of the functional requirements, as well as the actors.

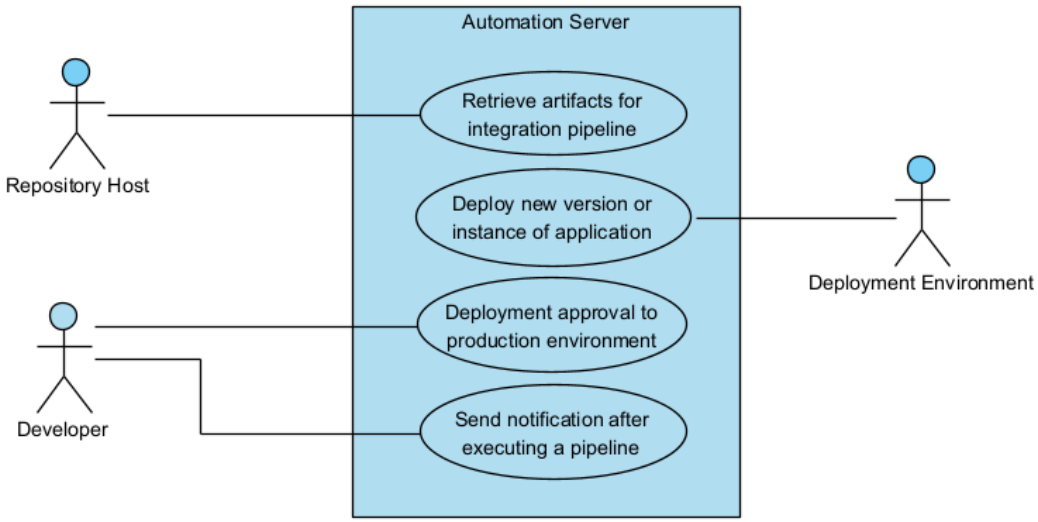

Figure 4 – Use case diagram

The automation server will be responsible for performing most of the tasks. It will retrieve the artifacts for the integration pipeline from the repository host, as well as deploy the application and send notifications after executing a pipeline to the developer. The only external intervenient is the developer, who will approve and start the deployment to the production environment.

4.2.2 Performance

The current downtime period between application updates, on this project, can take up to 20 seconds. On other projects, with a bigger codebase, this time can easily reach 5 minutes. The main goal for this metric is to reduce it by half, at least.

In terms of integration and deployment, one of the goals is to reduce the human interaction with the whole process. The process can take up to 15 minutes to integrate and deploy some projects in the organization, with the time being approximately 5 minutes for the current project.

Given the reduction of human interaction and the process previously described, the automation server must help reduce or at least keep the current process time.

4.2.3 Usability

Since it is the premiere of most of the technologies used to design/implement the solution in the organization, they should have a clear user interface and the overall configuration should be easy.

4.2.4 Security

The automation tool to be used for the automation pipelines must have basic authentication and authorization systems, to keep unauthorized users to access the system or perform any illegal actions.

4.2.5 Other requirements

Any solution must be self-hosted, helping to reduce costs and allowing expansion when necessary. Since the organization only uses SVN repositories and the projects are written in Java and built with Ant, the chosen automation tool must support these technologies, preferably out-of-the-box.

Another concern is the usage of sticky sessions on some of the applications that are being developed. Any zero-downtime deployment technique must consider that and guarantee a smooth transaction.

Since the work is experimental, it will be only used one server that should allocate the automation server and the application itself, allowing the use of various environments (production and test).

4.3 Integration and Deployment Flow

To fulfill all the requirements, there are obligatory steps to be performed:

- Checkout from the repository
- Project build
- Execution of automated tests
- Publish generated artifacts
- Deploy to test environment
- Deploy to the production environment

After the research performed in chapter 2, one of the best ways to perform it is by using an automation tool that allows the definition of an integration and deployment pipeline. In this subsection, it is first presented the pipeline design, as well as some alternatives and the definition of the tool to be used.

4.3.1 Tool

As previously discussed in section 2.8, there are several tools in the market with the ability to create and maintain a CI/CD pipeline.

After the research previously presented, and inside deliberation, the tool to be used in the process is TeamCity, by JetBrains, since it fulfills all the requirements of the organization and can execute all the required tasks without any extra configurations.

4.3.2 Using a continuous integration pipeline with a manual deployment pipeline (CI/CDE)

The process should start automatically with an external stimulus (a commit, for example). After checking out the repository, the project should be automatically built and tested, using the existing unit tests.

After the automated test phase, the artifacts should be published to the pipeline host machine. The generated artifacts are published into a test environment, to be performed manual acceptance tests. The diagram below in figure 5 represents the flow.

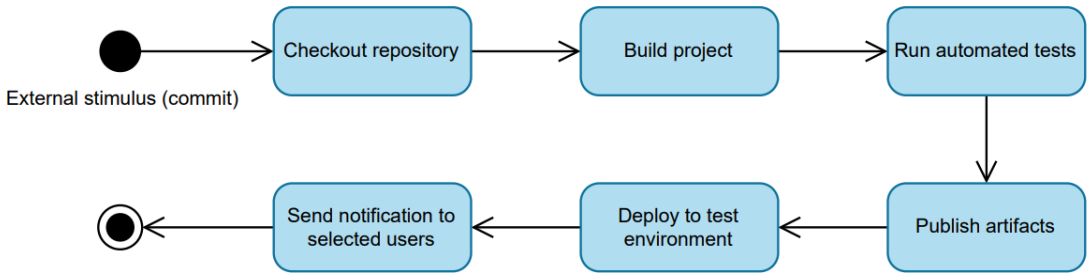

Figure 5 – Activity diagram representing the continuous integration pipeline

When the manual acceptance tests are performed, a user triggers the deployment pipeline. This pipeline's sole objective is to deploy the previously generated artifacts to the production environment, informing about its success after the operation. This flow is represented in figure 6.

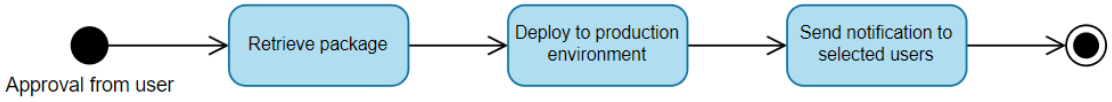

Figure 6 – Activity diagram representing the deployment to the production pipeline

It is important to notice that the above pipeline only represents the best-case scenario. If any phases fail, a list of selected users must be notified about the failure to perform corrective actions.

4.3.3 Alternative: using a single pipeline for continuous integration with manual deployment pipeline (CI/CDE)

Another flow would be joining all components into one pipeline, as presented below in figure 7. With this, the integration steps would be executed on the same pipeline as the deployment tasks, and a pipeline executing would be marked as terminated when the software is deployed.

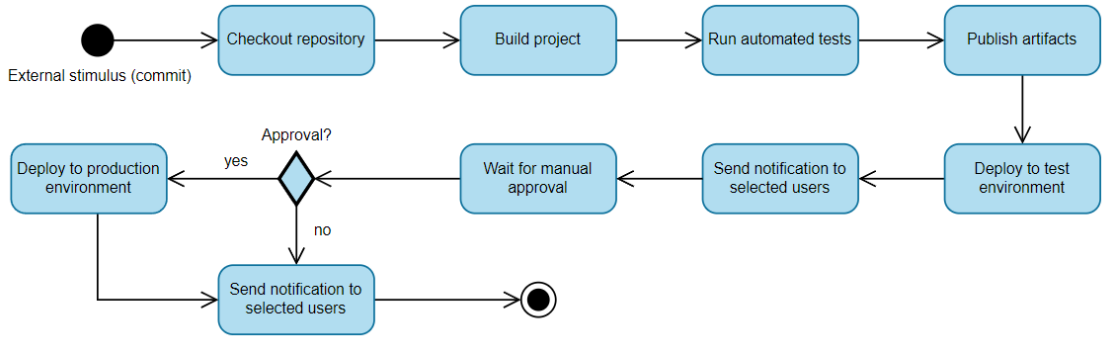

Figure 7 – Activity diagram representing a single CI/CDE pipeline

According to the project requirements, continuous deployment is not required at this point since there are manual tests to be performed. This would mean that a pipeline agent would be on hold until manual approval is given after the manual tests. A notification would be sent in case of failure or upon pipeline completion, as presented in the previous activity diagram.

If each commit triggers the pipeline, and since there is a limited number of execution agents, this will mean that, at some point, all the available agents would be occupied. This factor was enough to not consider this design proposal.

4.4 Zero-downtime deployment

One of the goals of the project is to reduce the downtime associated with the deployment, that currently affects the users. Three techniques were previously described in section 2.7 and two were considered for the project.

4.4.1 Deployment environment

Currently, the organization uses servers with an Apache Tomcat instance to provide access to the applications. Tomcat allows the execution of multiple application versions, using the parallel deployment functionality.

However, by using containers as previously described in section 2.6, it is possible to provide more instances of an application when necessary, and each application instance can be executed in personalized environments, without affecting other running applications or

environments. Considering that the same server will be responsible for hosting the automation service (TeamCity) and the application, available through Docker containers, it was designed the following deployment diagram below (figure 8), encapsulating the high-level components of the project.

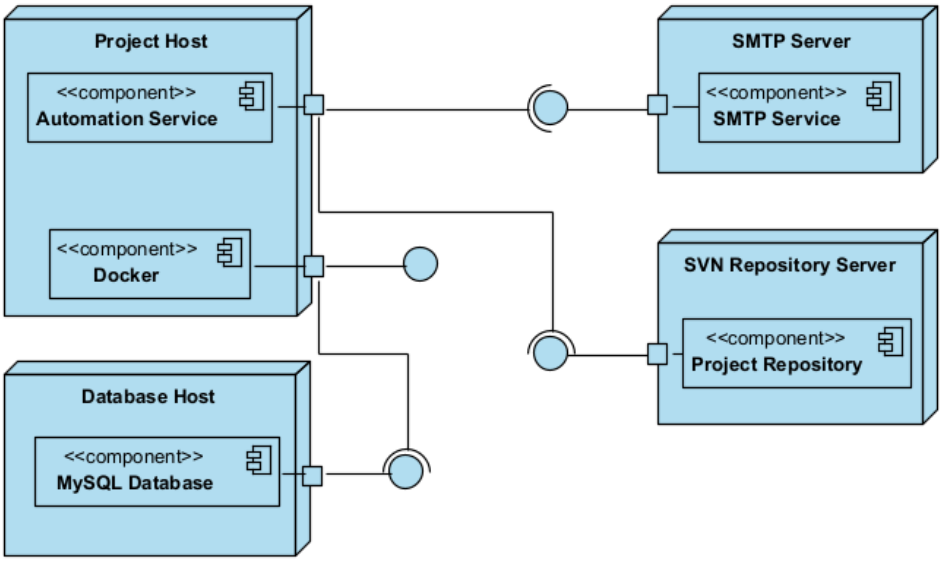

Figure 8 – Deployment diagram of the overall architecture of the project

4.4.2 Using service discovery and a proxy service with containers

Each application instance must be accessible publicly, and one way to achieve this while using containers is to add a proxy service that will manage the distribution of requests through other components, acting as an API gateway.

For this effect, it also arises the necessity of having a service discovery component, to guarantee that the proxy service always knows where it can distribute requests. This service discovery would receive requests from the proxy service, to send the request from the proxy to the respective application, and requests from the applications, to register themselves as available.

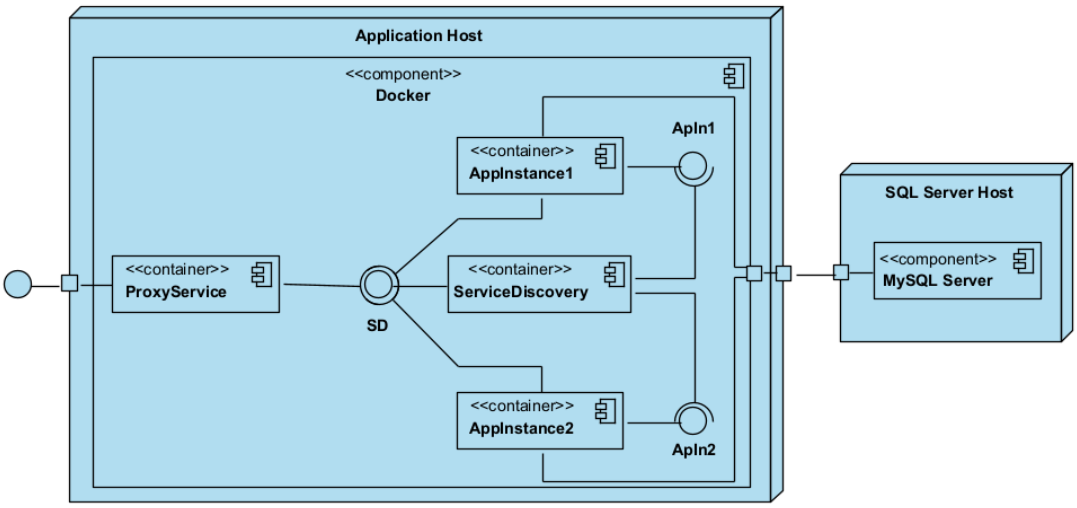

Figure 9 – Deployment diagram representing the host machine with containers, including a proxy service and a service discovery

This would allow adding, when necessary, new application instances without any extra work, since they would register themselves on the service discovery, making them available after starting up. In case of failure, a new container can be easily created and replaced with the damaged.

The main problem with this design is the existence of sticky sessions in the project. If there is not a component ready to manage this aspect, a user session can be easily invalidated if sent to a container that does not have info about it, forcing the user to log in again.

4.4.3 Alternative: using a load balancer with containers

An alternative would be using a load balancer. It would function as a bridge between the internet and the application instances, whether they are, or not, on the same machine.

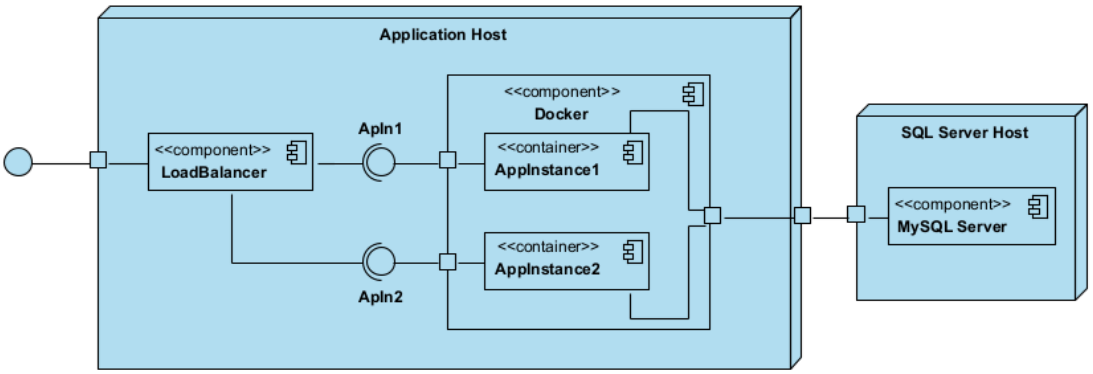

Figure 10 – Deployment diagram representing the host machine using only a load balancer and applications running on Docker

The main advantage of this technique is session persistence. Since the load balancer has the definition of all the available hosts with the application and it is static, it is easier to guarantee that a request always goes to the same server, given the session attributes.

However, the available applications are defined manually on the load balancer, which reduces the flexibility if there is a drastic change in the environment or if the organization decides to scale the application horizontally. With this method, it is possible to have lost requests if they are sent to an unavailable application, unless there is a mechanism that syncs sessions between instances. Finally, all the application instances are exposed to the system, unlike the previous alternative that only exposed the applications to an internal private network. Considering all the described factors, the current alternative was discarded.

5 Implementation

On this chapter, it is described the process to implement the project previously designed. Firstly, it is presented a roadmap, which describes the different steps that were followed during the implementation and identifies its respective subsection.

The implementation for each item identified on the roadmap is described after, according to its priority and as explained on the roadmap. The last subsection performs a global evaluation of the described process.

5.1 Roadmap

According to the FAST analysis and diagram in subsection 3.2, the main priority of the company is to implement the integration and deployment pipeline, followed by the reduction of the downtime period between updates.

Therefore, the implementation of the project was split into four different phases:

- TeamCity and CI pipeline configuration (subsection 5.2)

  TeamCity initial configuration with the integration pipeline, except the test environment deployment.

- Test environment deployment (subsection 5.3)

  Docker image creation and deployment on the current server, replacing older instances with a new version.

- Configuration of the deployment pipeline (subsection 5.4)

  Creation of a deployment pipeline using the published artifacts and deploying the new version for production.

- Zero-downtime deployment technique (subsection 5.5)

  Implementation of the approach described in section 4.4.2, to decrease the downtime between an update.

5.2 TeamCity configuration and CI pipeline

As specified before, the chosen software for the implementation of the CI/CD pipeline was TeamCity, by JetBrains. It was installed as a service on the same machine used to host the test instances of the application, along with Docker.

5.2.1 Initial TeamCity project configuration

After installing TeamCity and performing the first configuration, the page shown below in figure 11 appears and it is possible to create a project through the UI. A project acts as an aggregator for multiple build configurations connected through a common element, being this element the source code repository. A build configuration is a sequence of steps to achieve a certain goal, like the integration or deployment of a project.

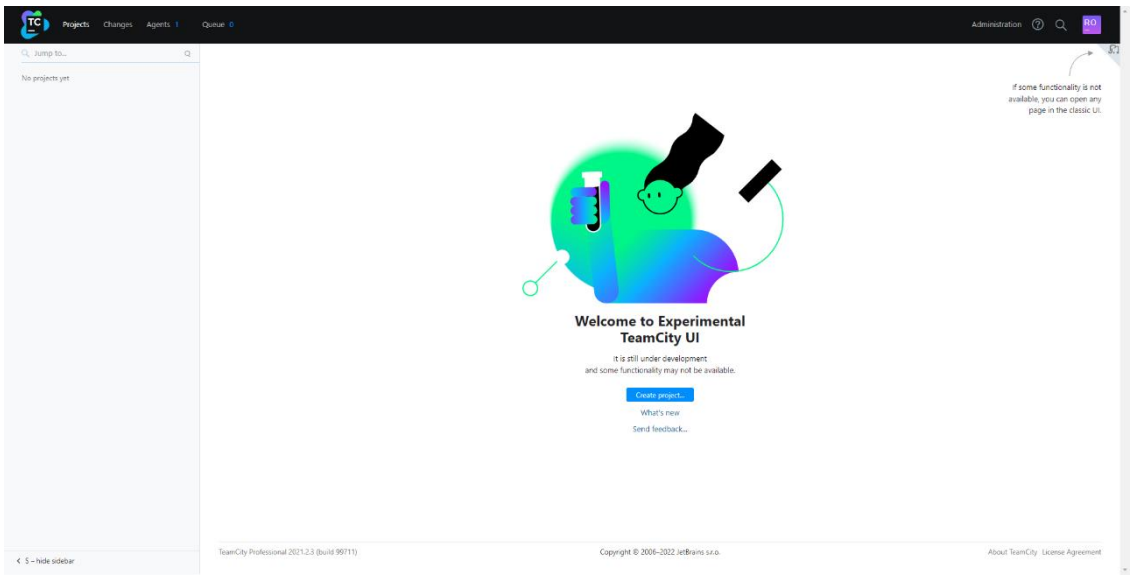

Figure 11 – Landing page after TeamCity's installation and basic configuration

Since the chosen design contemplates two pipelines, one for CI and the other for CD, it was necessary to configure two different configurations. The first step to implement both pipelines was the creation of the project, as shown in figure 12.

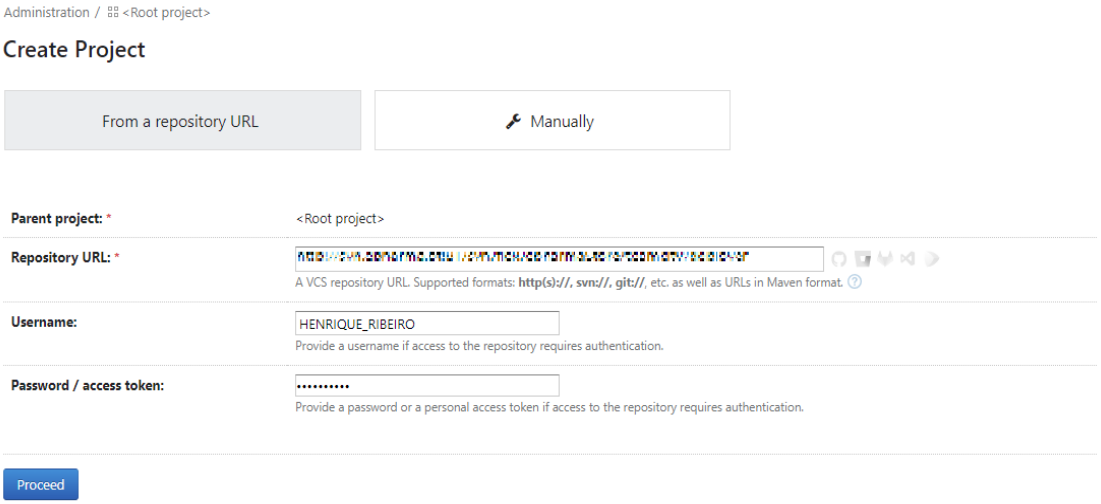

Figure 12 – First phase of project creation on TeamCity

To configure the project, it was necessary to provide the repository URL, as well as valid credentials to access it. After that, it is requested the project name and the first build configuration name, as shown below in figure 13. Since the automation process starts with the tasks associated with the integration of the project, the first build configuration was identified as the “Integration Phase”.

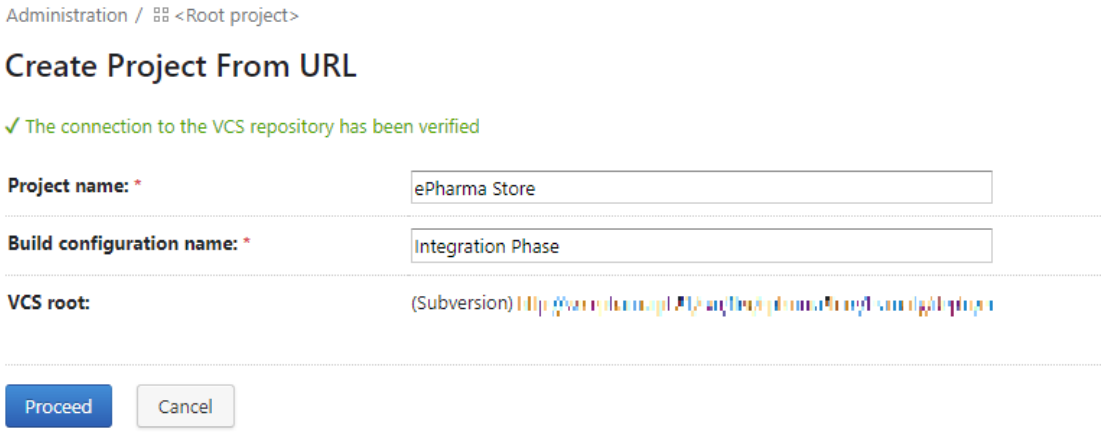

Figure 13 – Specification of the project name and first build configuration name while configuring the project

After proceeding in the previous step, TeamCity finishes the project creation and redirects automatically to the configuration of the first build configuration, “Integration Phase”. From the moment the project is created, TeamCity will automatically check the repository periodically for changes. If a change is detected, the project is automatically checked out and the pipeline, or build configuration, starts.

5.2.2 Steps configuration

Upon creating the project, TeamCity automatically analyses the repository to find a suitable build step. As shown in figure 14, TeamCity suggested a build step with the target “default” (builds the project) using the file “build.xml”.

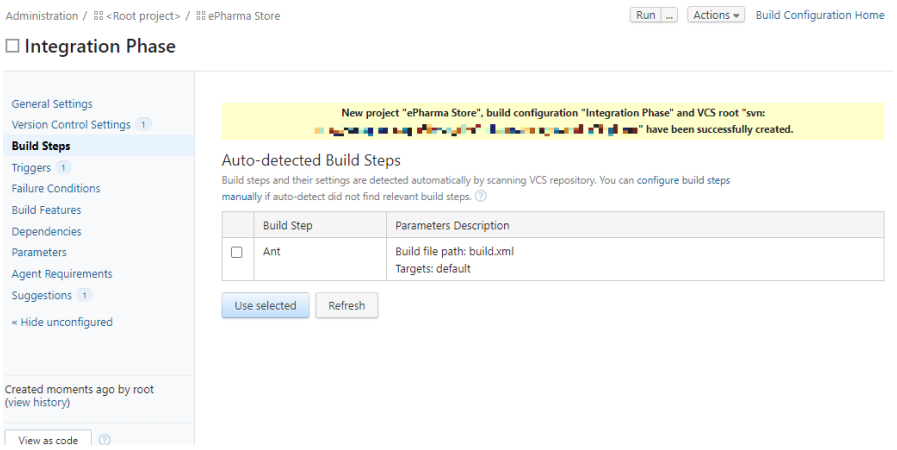

Figure 14 – Auto detection of a possible build step from the repository

Although the tool was not wrong in suggesting the target “default”, which builds the project, creates the documentation from the JAVADOCS, and runs the tests, it was necessary to adjust to the target “build-no-javadoc”. This step was created since the project does not contain JAVADOCS on all methods, resulting in an error while building.

It was also necessary to specify the Java Enterprise server home, since the project currently has dependencies associated with the server. Instead of directly defining the path into the configuration, it was defined as a variable in the build configuration that can be used in different build steps. Figure 15 shows the configuration of five variables that will be used in the project, on which one corresponds to the Java Enterprise server path and the other to be used on a future step.

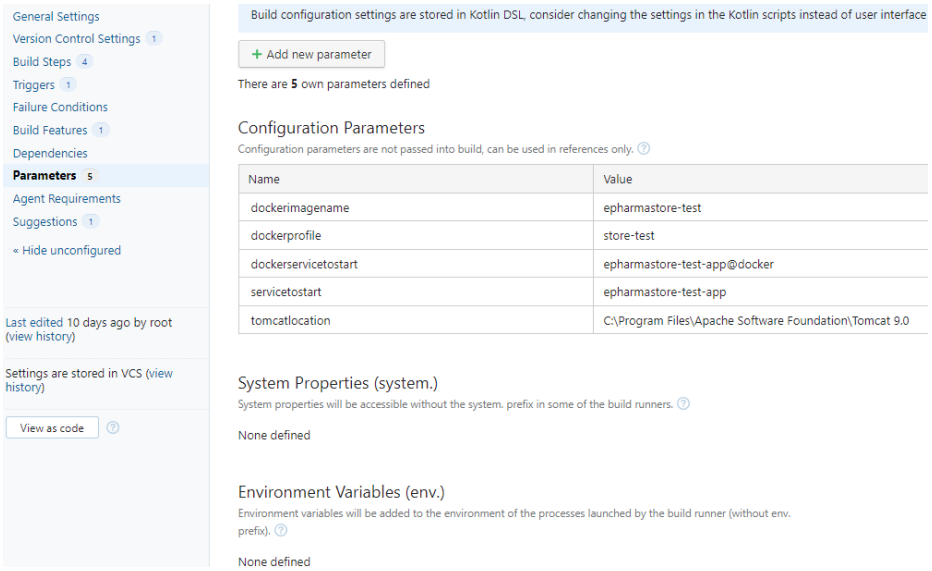

Figure 15 – Variable definitions on integration pipeline

After the configuration of a variable, it is possible to use it on different fields, as shown in figure 16, along with the specification of a different target and additional parameters for Ant.

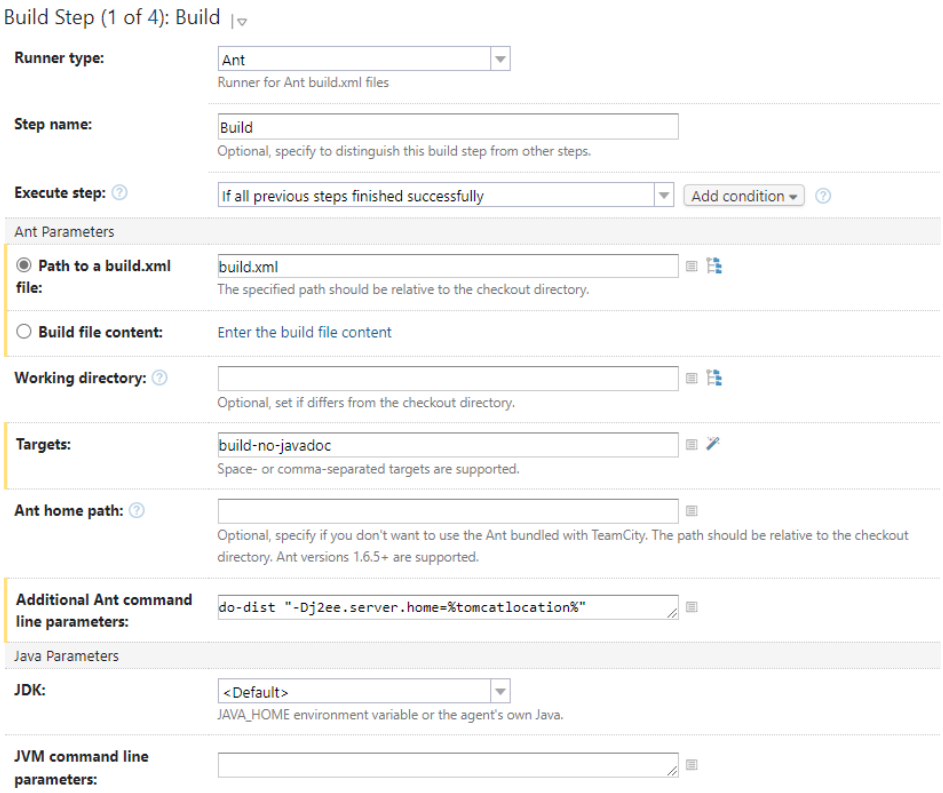

Figure 16 – Configuration of the build step on the integration pipeline

The second step is testing the project. The only change between this step and the build step is the Ant target, as shown below in figure 17.

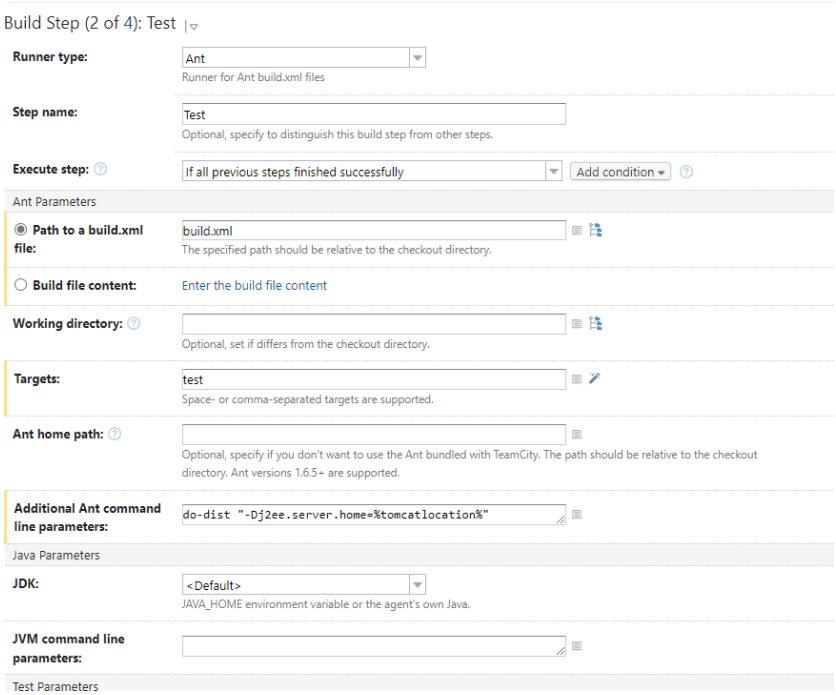

Figure 17 – Configuration of testing step on integration pipeline

As requested by the organization, it is necessary to send an email reporting a build success status, as well as possible failures. This was achieved by configuring the build feature Notifications and selecting the options shown in figure 18.

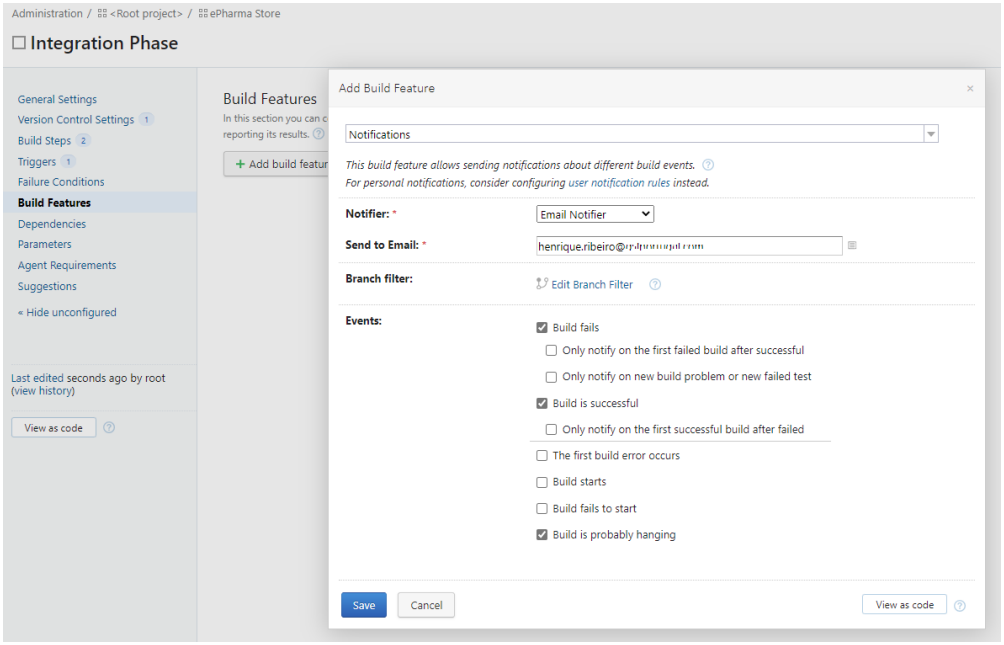

Figure 18 – Configuration of Build Feature that sends notifications via email

The last configuration on the integration pipeline was the artifacts publication. Instead of creating a new build step, it was configured in the general settings, as shown below in figure 19.

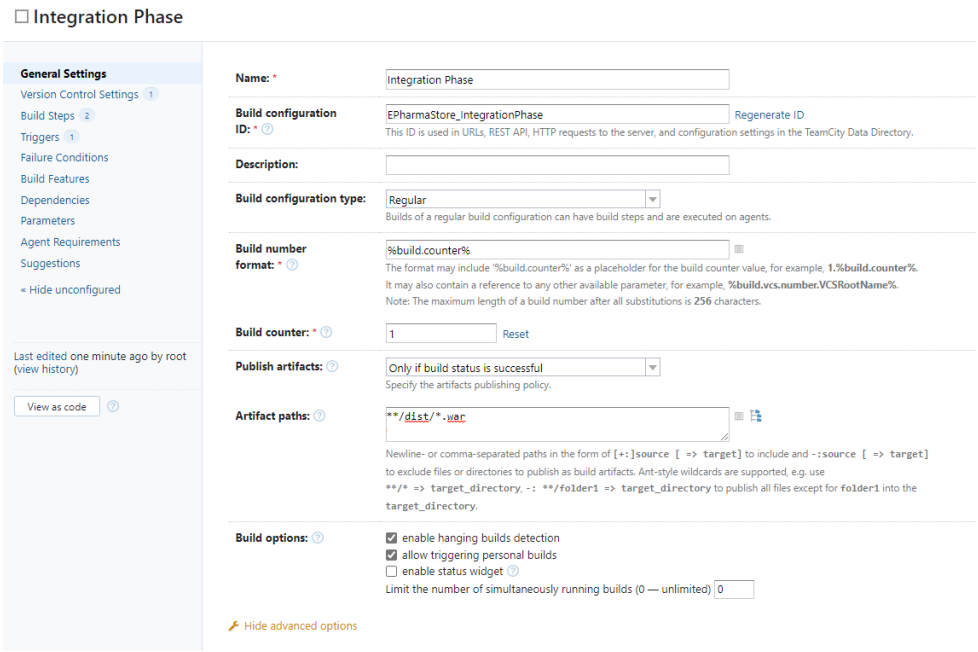

Figure 19 – Configuration of artifacts publication on integration pipeline

The publication will only occur if the pipeline execution finishes successfully. Currently, the artifact to be archived is the generated ".war" file.

The only configuration missing was the storage of all the configurations on the repository, following the DevOps principle of IaC. The information is stored on a Kotlin file present in the repository, present in appendix A, and every change performed in the UI is automatically saved on the repository. Figure 20 below shows the configuration page on the UI.

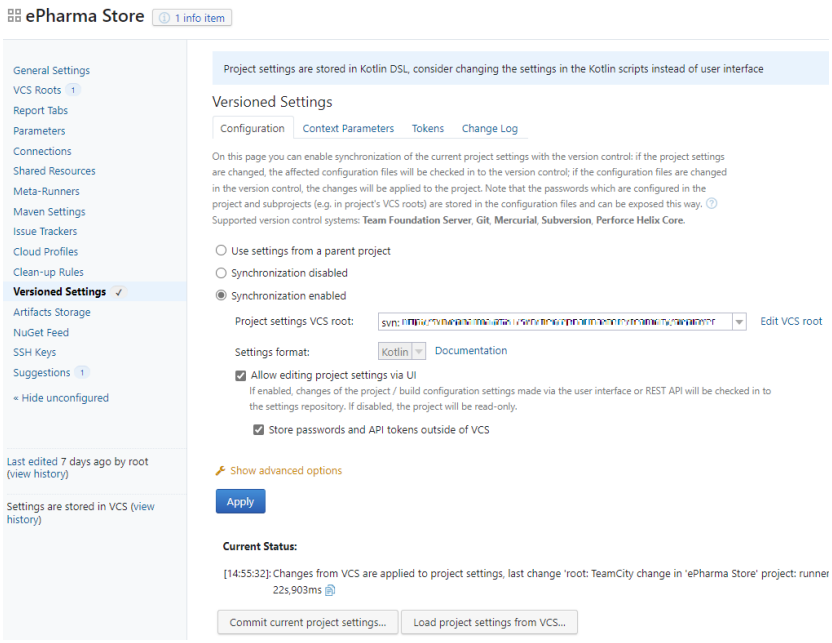

Figure 20 – Configuration of synchronization of project settings on the repository

5.3 Test Environment Deployment

This phase was included in the integration pipeline. Since one of the goals of the project was to containerize the application and provide it with a tool like Docker, the test environment deployment steps were designed to create a Docker image and create a container.

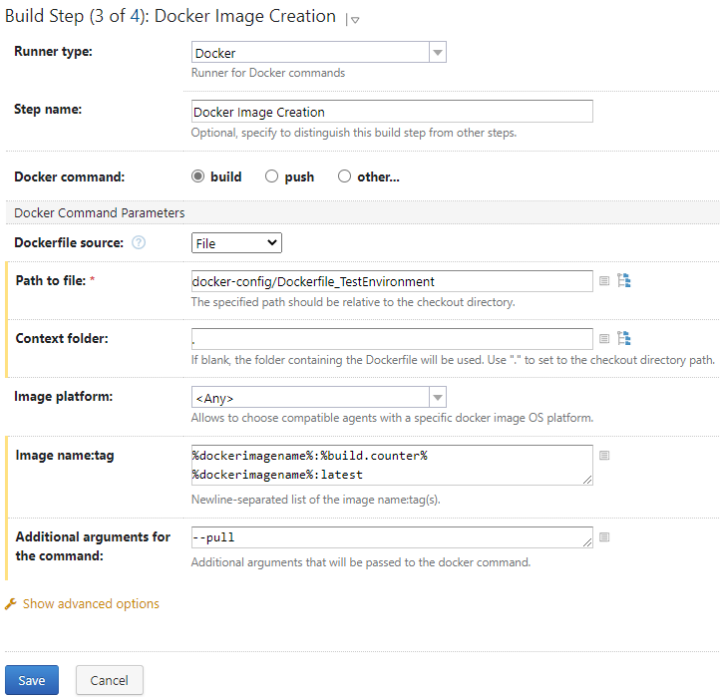

Figure 21 – Configuration of Docker image creation step on integration pipeline

As shown above, in figure 21, the configuration is straightforward: the Docker command to be executed is “build”, using the Dockerfile stored on a specific folder with the name Dockerfile_TestEnvironment (exclusive to the test environment), and the context folder was set to be the project root. The image that is built on this step will have two different tags: “latest”, to always keep the latest image marked differently, and one corresponding to the current build number. The project name is given using a variable previously defined in figure 15.

The test environment will have a different access path from the production path, but the project, when the integration pipeline is executed, only contains a configuration file with the production settings, which could result in invalid requests to the server. To dynamically change the content of the file, it was configured a step before the build that finds the configuration file and changes the content, replacing the host path with the correct one, as shown below in figure 22. Before the artifact publish step, the changes are reverted, resulting in the publishing of artifacts for the deployment pipeline with the correct configuration variable.

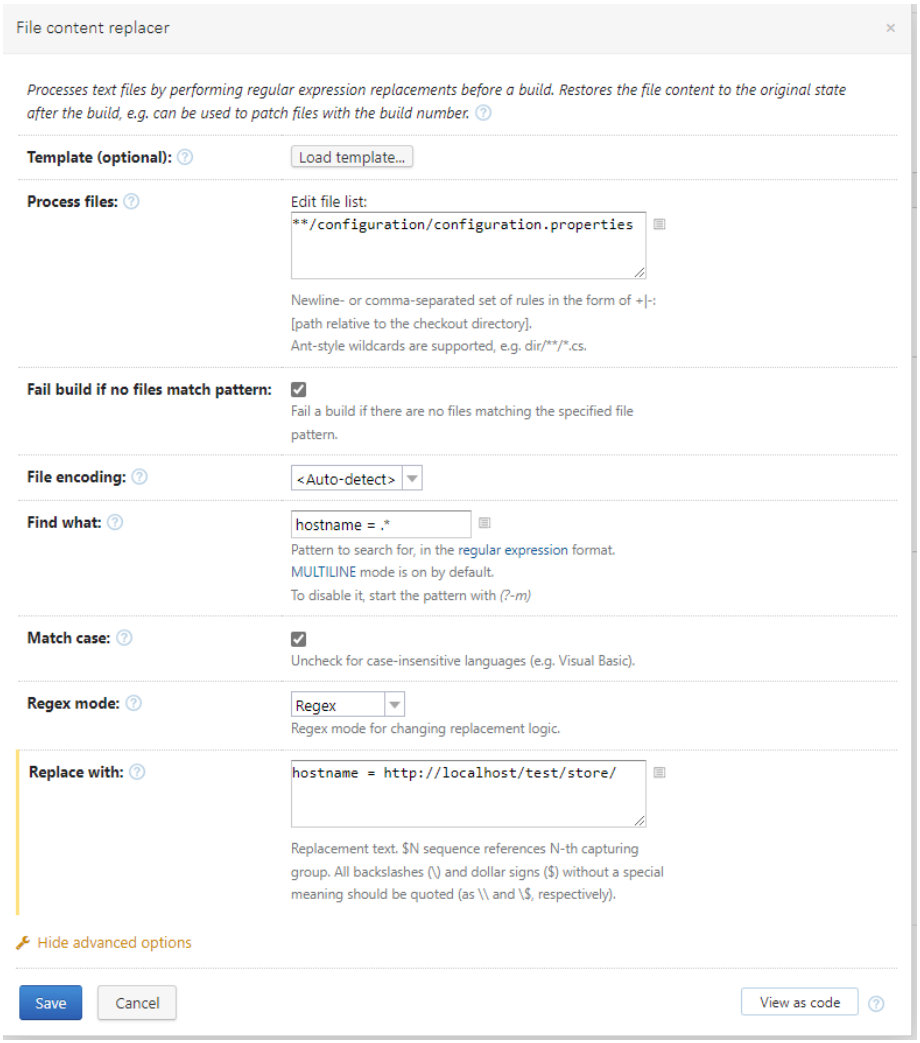

Figure 22 – Configuration of automatic text replacement on the specified file

The Dockerfile used is shown in code 1 below. It only copies the compiled application directly into a Tomcat image downloaded from the Docker Hub. The project is copied to the “test#store” folder, which results in the application being exposed on the path “test/store”. It is required to be an image with version 9 of Tomcat, since the latest images replace Java EE with Jakarta EE and the project is currently prepared to work with Java EE.

```
1  FROM tomcat:9.0.62-jdk11-corretto
2  COPY build/web/ /usr/local/tomcat/webapps/test#store
3  CMD ["catalina.sh", "run"]
```

Code 1 – First version of the Dockerfile to build application Docker image

To start the test instances, it was created another step to run a Docker Compose command on a PowerShell terminal. It executes a command that recreates the service, if already running, as shown in figure 23.

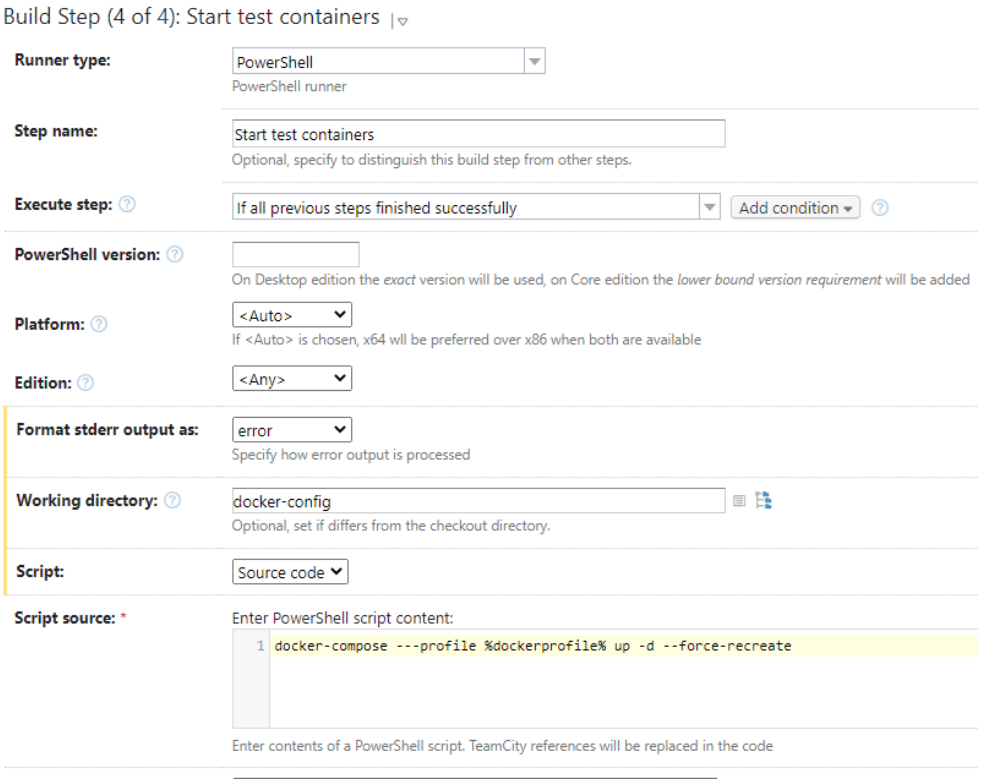

Figure 23 – Configuration of the execution of the test containers

The Docker Compose configuration file on this phase only has one service and uses the image that was previously created, and it exposes the application through the port 8080, as shown below in code 2.

Since the same Docker Compose file will be also used for production, and it is not desirable to recreate or create exceptions on the command line to only recreate a certain service, it was also defined a profile with the name “store-test”. With the profile set, it is possible to execute the Docker Compose command while specifying a profile, and only services marked with the profile or without any will be created. This is useful to have multiple environments on the same Docker Compose, such as the test and production environment.

```
1   version: '3.3'
2
3   services:
4     epharmastore-test-app:
5       image: epharmastore-test:latest
6       expose:
7         - 8080
8       ports:
9         - 8080:8080
10      profiles: ["store-test"]
```

Code 2 – Initial Docker Compose configuration file, only with the application service

The command to be executed when creating the test environment is shown in figure 23.

5.4 Configuration of the deployment pipeline

The deployment pipeline, as designed in subsection 4.3.2, only requires retrieving the generated artifacts and deploying the application.

The first step was to create a new build configuration for the deployment. However, since the artifacts were created on the integration pipeline, it is not necessary to checkout the repository. Hence, the manual configuration was chosen while creating, as shown below in figure 24.

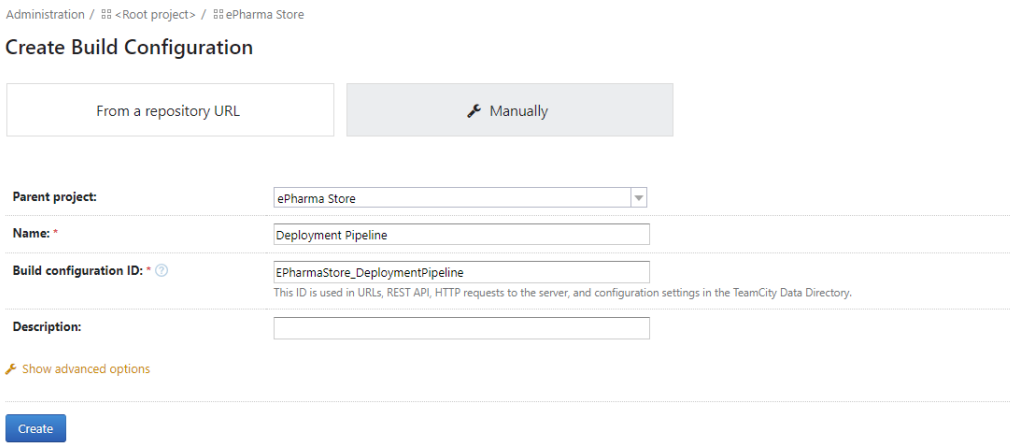

Figure 24 – Creation of build configuration for deployment on project

After the creation of the pipeline, the next step was the variable definition. The variables were set as shown in figure 25, and are practically the same as the test environment, differing in the values and the missing Tomcat location variable.

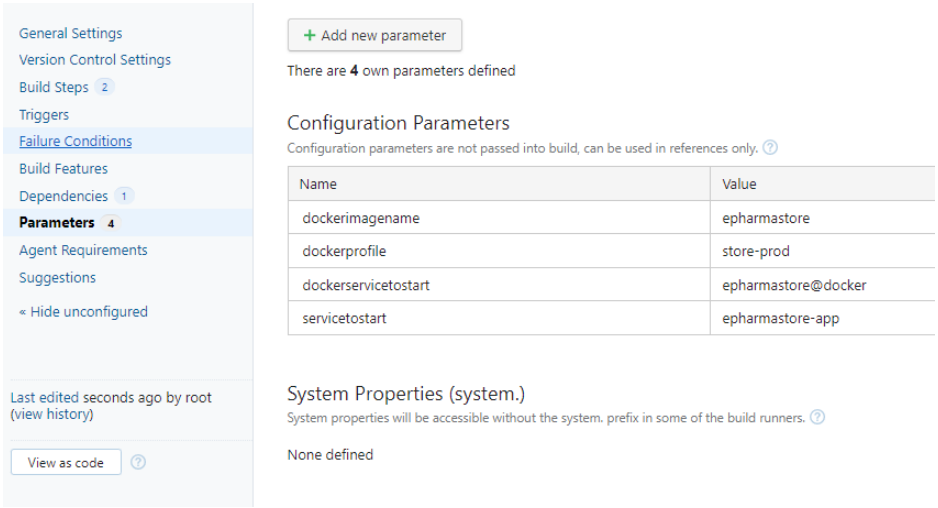

Figure 25 – Variable definition on deployment pipeline

There are only two build steps to be configured: the creation of a Docker image with the latest test image and the execution of the Docker Compose command to start the container.

The same Dockerfile can be used for the test and the production environment, but it is necessary to add more artifacts to be published by the integration pipeline. The “build\web” directory contains the compiled code that is also present in the “war” file, being necessary to build the image. The “docker-config” folder contains the necessary Dockerfile to create the image and other files that are required to create and deploy an environment.

Figure 26 shows the change performed on the integration pipeline. In comparison with the previous configuration, presented before on figure 19, it was added the “build\web” directory and the “docker-config” folder, with the content being placed in the same location as in the current workspace.

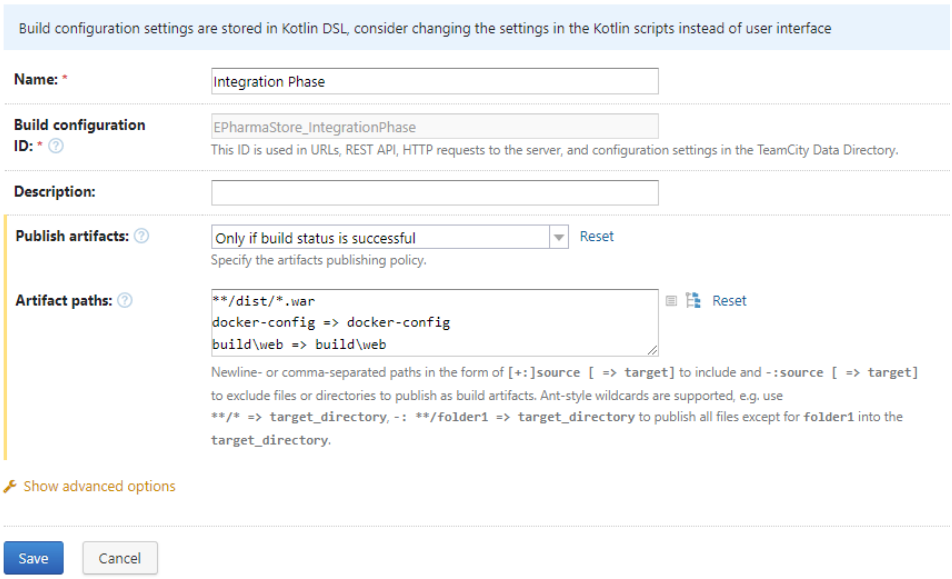

Figure 26 – Configuration of published artifacts on integration pipeline

To use the published artifacts, it is necessary to configure the pipeline to retrieve them as dependencies and copy them to the current workspace. TeamCity allows the definition of artifact dependencies, connecting directly to another pipeline and retrieving published items.

Figure 27 shows the performed configuration, with the dependency pipeline being the previously defined integration pipeline. The published artifacts were stored in the workspace on the same path as in the integration workspace.

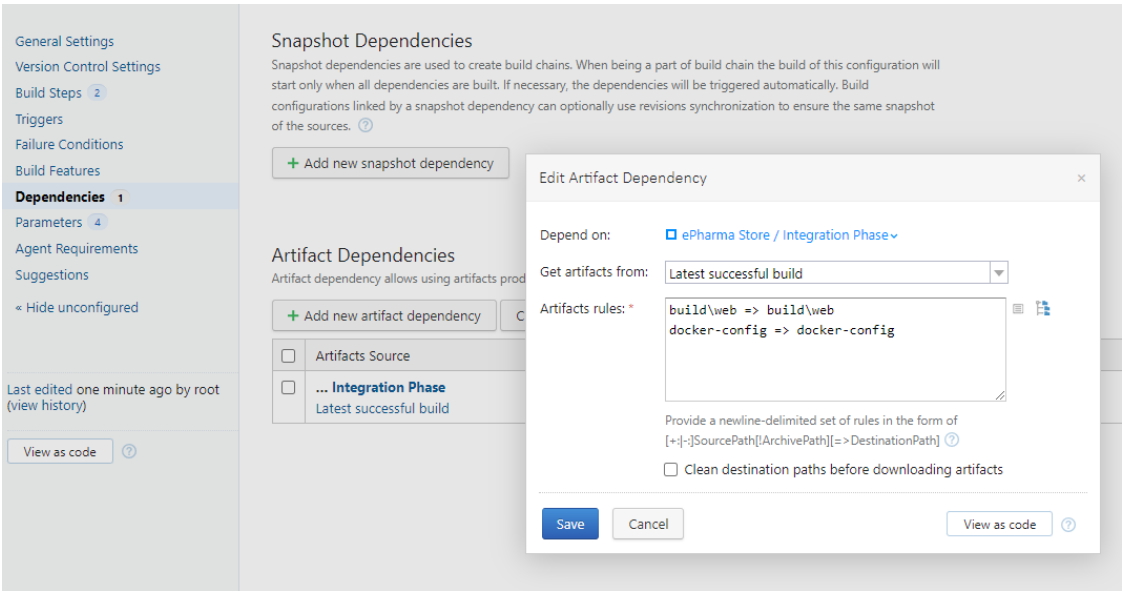

Figure 27 – Configuration of artifact dependencies on deployment pipeline

Finally, it is possible to define the steps. The first is the Docker image creation, with the configuration being the same as the one in the integration pipeline, only changing the name of the Dockerfile. The only difference between the Dockerfile used for production and test environment is the path into which the application is copied, inside the image.

The second, also configured as the same as its counterpart in the integration pipeline, is the creation of the production containers, using the same Docker Compose command as in the test environment. The performed configurations are shown below in figure 28 and figure 29.

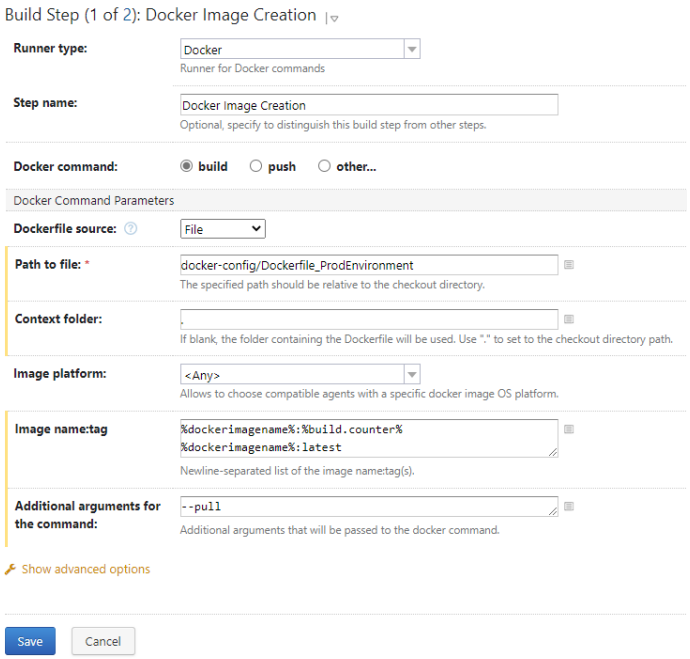

Figure 28 – Configuration of the Docker image creation on the production pipeline

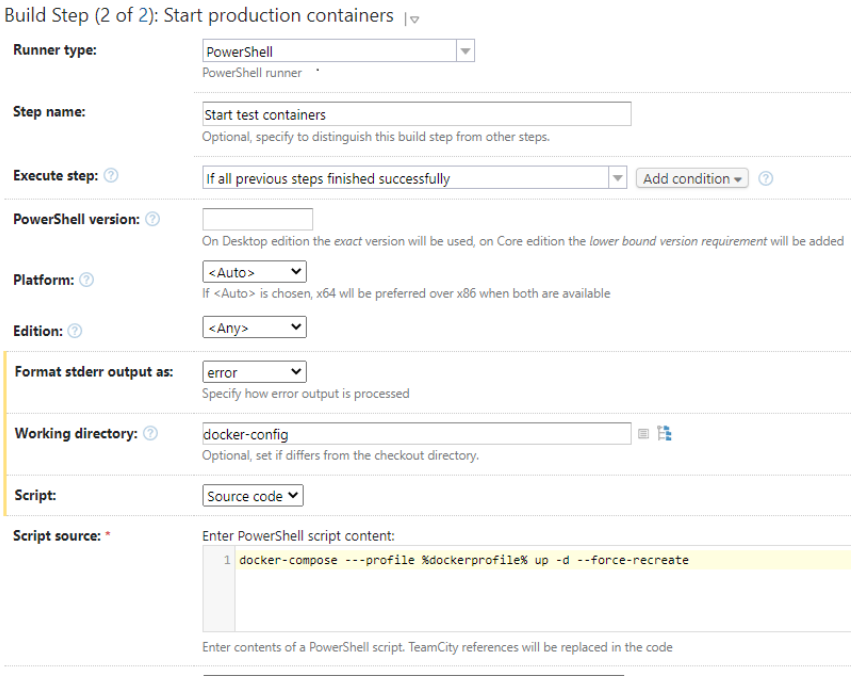

Figure 29 – Starting the production containers on production deployment

Lastly, and similarly to the integration pipeline, it was also configured the notification sender, in the same way as shown in figure 18. In the dashboard of the project, it is now possible to directly run the pipeline, as shown below in figure 30.

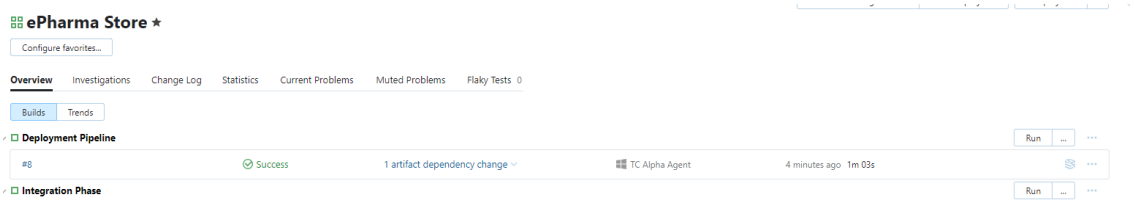

Figure 30 – Project dashboard with two configured pipelines

5.5 Zero-downtime deployment technique

As specified in subsection 4.4.2, the solution requires an API gateway, an application that acts as service discovery and maintains sessions through updates.

5.5.1 API Gateway and Service Discovery configuration

After some research, it was found a tool that is capable of handling both tasks with full Docker integration. Traefik is an open-source tool that acts as a proxy, redirecting requests to services known to it, while supporting the configuration of middleware and active service auto-discovery

(“Traefik Concepts Documentation - Traefik” n.d.; “Traefik Proxy Documentation - Traefik” n.d.). It can be executed through a Docker container.

For a container to be discovered by Traefik, it is necessary to set labels while creating it. The Docker Compose file allows the definition of all labels as specified below, on code 3.

```
1          labels:
2                - "traefik.enable=true"
3                - "traefik.http.routers.epharmastore-test-
   app.entrypoints=web"
4                - "traefik.http.routers.epharmastore-test-
   app.rule=Host(`localhost`) && PathPrefix(`/test/store`)"
5                - "traefik.http.services.epharmastore-test-
   app.loadBalancer.sticky.cookie=true"
6                - "traefik.http.services.epharmastore-test-
   app.loadBalancer.sticky.cookie.httponly=true"
7                - "traefik.http.services.epharmastore-test-
   app.loadbalancer.server.port=8080"
8                - "traefik.http.services.epharmastore-test-
   app.loadbalancer.healthcheck.path=/test/store"
```

Code 3 - Definition of labels for test environment container on Docker Compose file

The first label, presented on line 2, is self-explanatory, marking the container as to be discovered by Traefik. The third line indicates that the entrypoint for the service is through the HTTP port (80).

The third label, on line 4, specifies that a request received with the host being the local machine and with a path starting with “/test/store” must be redirected into this service. The fourth and fifth labels, on lines 5 and 6, are related to keeping sticky sessions through a cookie. By using them, a request is preferentially sent to the first server that handled the first request sent to a service.

The sixth label (line 7) sets that the port that will receive the requests on the container is the 8080. The seventh label (line 8) indicates that the root of the application will be used to check if the service is healthy and ready to receive requests. If a response to a health request is negative, the service will not be considered while redirecting requests.

Since the access to the application will only be done through Traefik, it is necessary to remove the port definition that was specified before in code 2.

Traefik is executed through a Docker instance, configured on the Docker Compose file as shown in code 4 below. The volume configuration is necessary to listen to Docker events, and the API insecure definition was necessary to check Traefik UI.

```
1     api-gateway-traefik:
2       image: traefik:v2.6
3       command:
4         - --entrypoints.web.address=:80
5         - --providers.docker
6         - --api.insecure
7       ports:
8         - "80:80"
9         - "8120:8080"
10      volumes:
11        - "/var/run/docker.sock:/var/run/docker.sock:ro"
```

Code 4 – Configuration of Traefik on Docker Compose file

After executing the Docker Compose, Traefik will be executing using port 8120 to expose its UI and port 80 (HTTP) to receive and forward requests. Figure 31 shows the active containers after executing the Compose file, while figure 32 represents the Traefik dashboard.

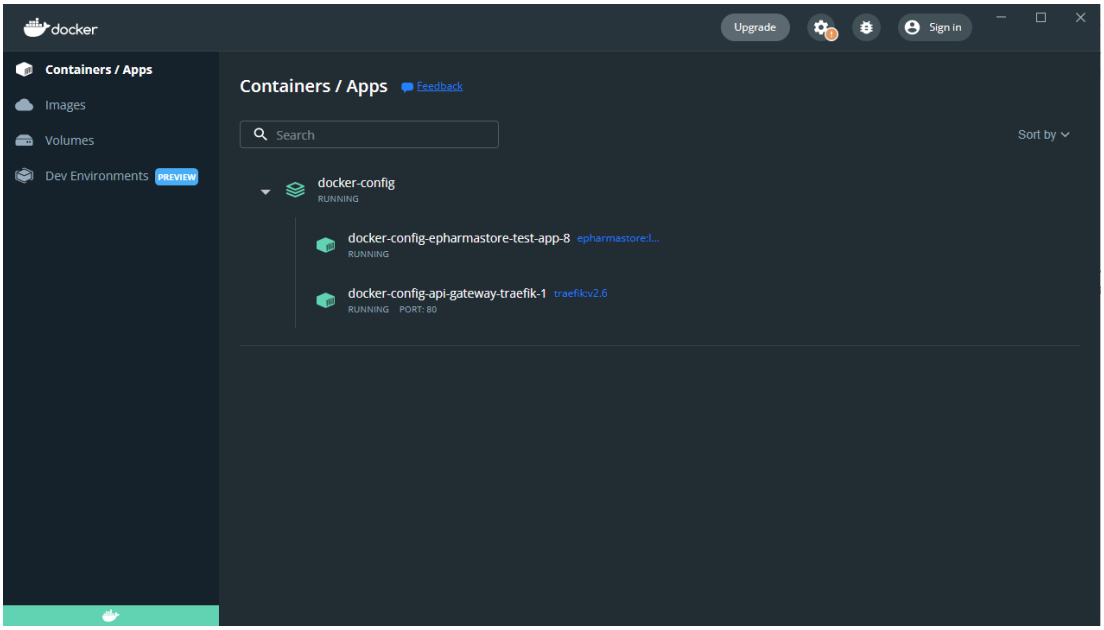

Figure 31 – Docker Desktop with containers created after Docker Compose file execution

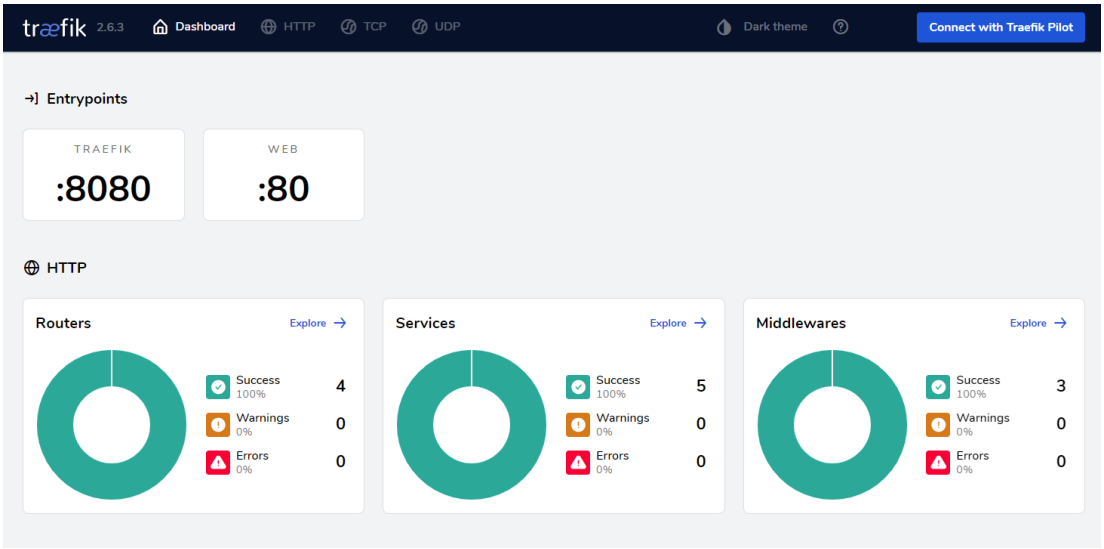

Figure 32 – Traefik dashboard

There are three services and four routers already defined. One of the routes on the list is the same configured on code 3. As shown below in figure 33, the router receives a request for the host “localhost” with the path starting with “/test/store” and sent it to the application, identified as “epharmastore-test”.

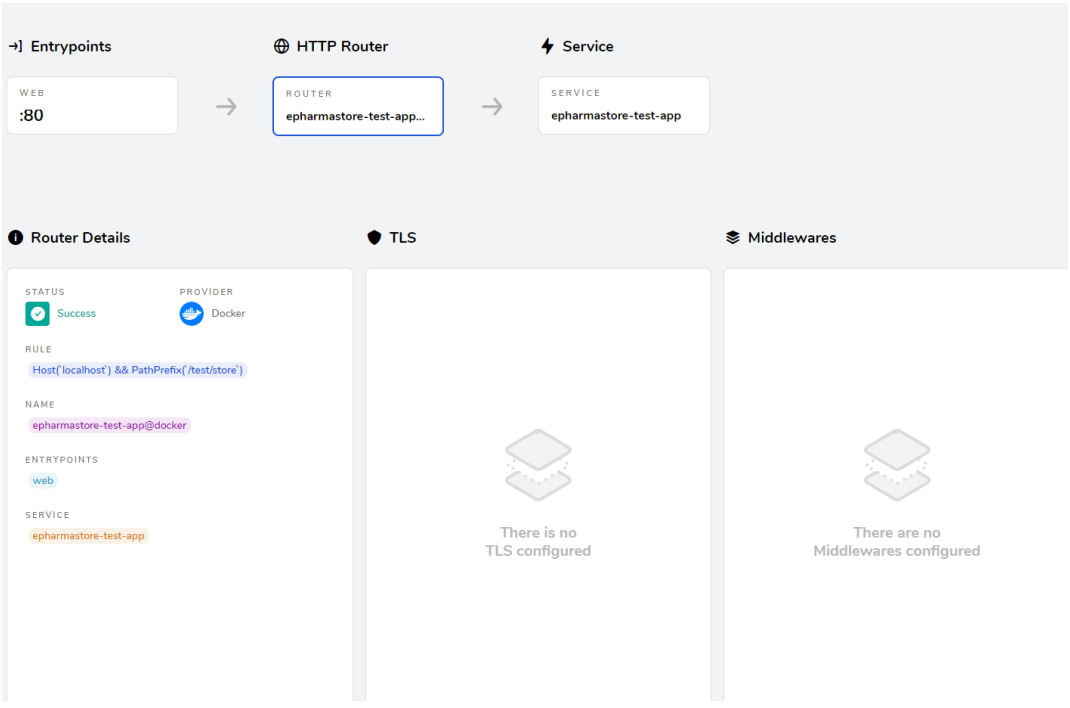

Figure 33 – Router overview for test application on Traefik

The service page for the “epharmastore-test-app” service shows information such as the provider, active servers, and what routers are using it, as shown below in figure 34.

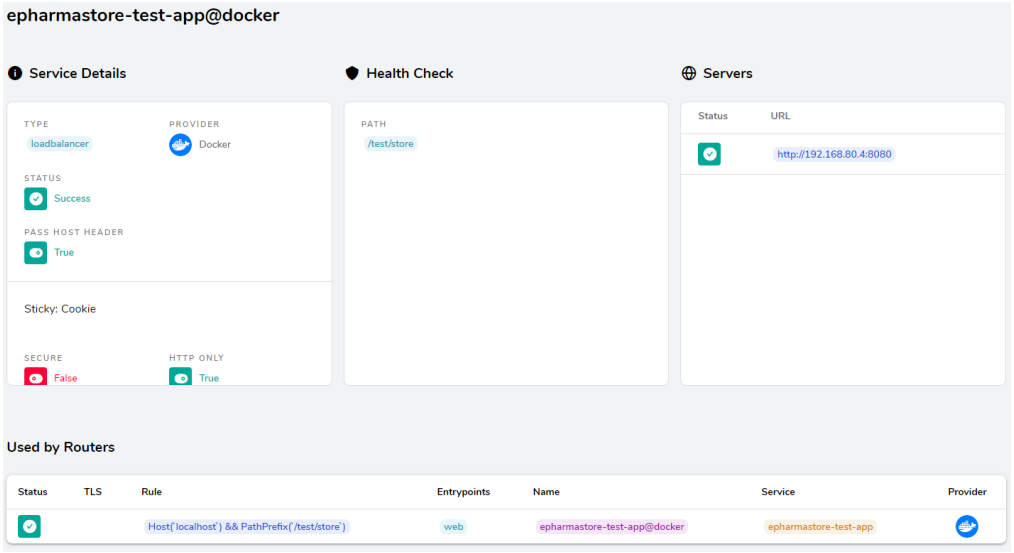

Figure 34 – Service overview for test application on Traefik

By accessing “localhost/test/store”, the main page of the project will appear.

5.5.2 Creation and removal of containers with different versions

Without using a zero-downtime technique, it would be necessary to recreate the container using a recent image, with the application being unavailable during the switch. The diagram below (figure 35) shows the necessary steps to perform the switch without downtime.

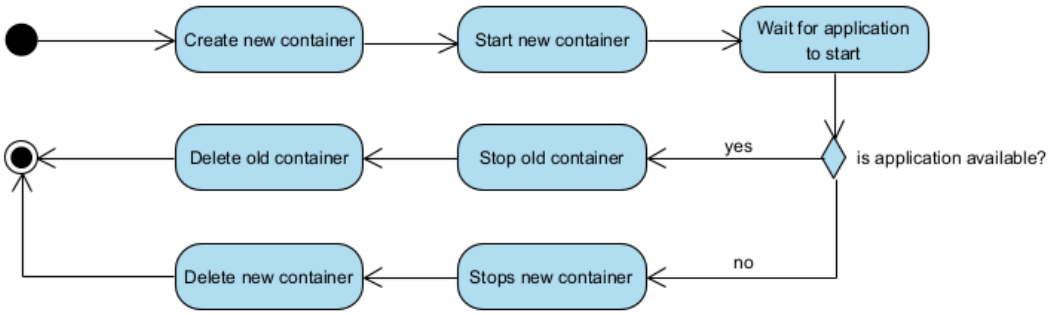

Figure 35 – Expected flow to switch instances with different versions

TeamCity cannot perform these tasks natively, so it was necessary to develop an alternative. After some research (Malviya 2021; O’Brien 2022), the best option found was to develop a script that handles all tasks previously presented, using Docker commands while taking advantage of scripting capabilities such as variable storage. There were analyzed some alternatives found during the research, but the solutions did not seem fully appropriate to the project, and it was necessary to develop a more robust solution.

The developed script, shown in code 5, starts by setting three variables on the first three lines: the service name, the API service name, and the profile to be used while executing the Docker Compose command. The values are passed by parameter when the script starts its execution.

When the script is executed on a regular scenario, there is only one container in execution for a service, and that same container needs to be stopped and deleted at the end of the script. Therefore, it is necessary to retrieve its identifier to later execute a command to perform operations. For this, it is used a Docker command to retrieve a list of container identifiers with the name containing the defined service name, retrieving the last line using the “Select-Object -Last 1” command, and storing it on a variable, as shown on line 5.

After that, the Docker Compose “up” command is executed on line 7, starting with the profile definition passed as a parameter. It contains extra arguments, such as:

- “scale”, followed by the service name and a number: the number of instances to be created of a certain service defined on the Docker Compose file
- “no-recreate”: if the number of containers for each service is the same as defined in the “scale” parameter (or one, if not specified), the containers will not be recreated
- “-d”: executes containers in the background, without printing status in the current console

For this project, the “scale” argument was used for the service being created with the number being two, which means that after the execution of the command, two containers with the same image will be available. The “no-recreate” argument guarantees that the current executing containers are not affected and running as expected, like Traefik or other environments and services.

After executing the Docker Compose command, the identifier of the new container is obtained on line 9, in a similar way as the old container identifier. It is also obtained the IP address of the new container using the “docker inspect” command, to check the container health later, as shown on line 10.

Since the services are not exposed, it is impossible to check if the Tomcat application is running except through Traefik, and Traefik will not send requests to a container that is not ready. However, it is possible to use Traefik API to obtain information about a certain service, including the status of a web service host according to its IP address. Therefore, it was implemented a finite loop on line 15 that, every ten seconds, performs an HTTP request using “Invoke-WebRequest” to get information about the service. After receiving the response, the body is parsed to JSON and the server status is extracted and stored on the variable “result”. This process is shown on line 16.

After retrieving the value, the flow continues from lines 18 through 23. It checks if it is equal to “UP”. If it is, the ready flag is marked as true, otherwise, the loop is stopped for ten seconds. This check occurs ten times or until the ready flag is true.

If the new container is ready, the “docker stop” and “docker rm” commands are executed for the older container, on lines 27 through 29, otherwise, an error message is printed, and the new container is stopped and deleted, as shown on lines 31 through 33.

```
1  $service_name=$args[0]
2  $service_name_api=$args[1]
3  $profile_to_start=$args[2]
4
5  $old_container_id = docker ps -f name=$service_name -q | Select-
   Object -Last 1
6
7  docker-compose --profile $profile_to_start up -d --scale
   $service_name=2 --no-recreate
8
9  $new_container_id = docker ps -f name=$service_name -q | Select-
   Object -First 1
10 $new_container_ip = docker inspect -f
   {{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
   $new_container_id
11
12 $new_container_ready = $false
13 $tries = 0
14 $result = "KO"
15 while (($tries -le 10) -and (!$new_container_ready)) {
16      $jsonResult = Invoke-WebRequest -Uri
   http://localhost:8120/api/http/services/$service_name_api
   -UseBasicParsing | Select-Object -Expand Content | ConvertFrom-Json
17      $result =
   $jsonResult.serverStatus."http://$($new_container_ip):8080"
18      if ($result -eq "UP") {
19          $new_container_ready = $true;
20      } else {
21          Start-Sleep -Seconds 10
22      }
23      $tries++
24 }
25
26 if ($new_container_ready) {
27     #Kills old container
28     docker stop $old_container_id
29     docker rm $old_container_id
30 } else {
31     docker stop $new_container_id
32     docker rm $new_container_id
33     Write-Error "Old container not terminated. Please check new
   container"
34 }
```

Code 5 - PowerShell script for container switching

To execute the script, it was changed the build step that was responsible for creating the containers by executing a Docker Compose command. Instead of directly running the command, as shown in figure 36, the task was changed to execute the script file on the repository, passing three parameters, both defined previously: the service name on the Docker Compose file, the Docker service name on Traefik and the profile to be executed. This change was made in the test and production environments, only changing the variable's value between the pipelines.

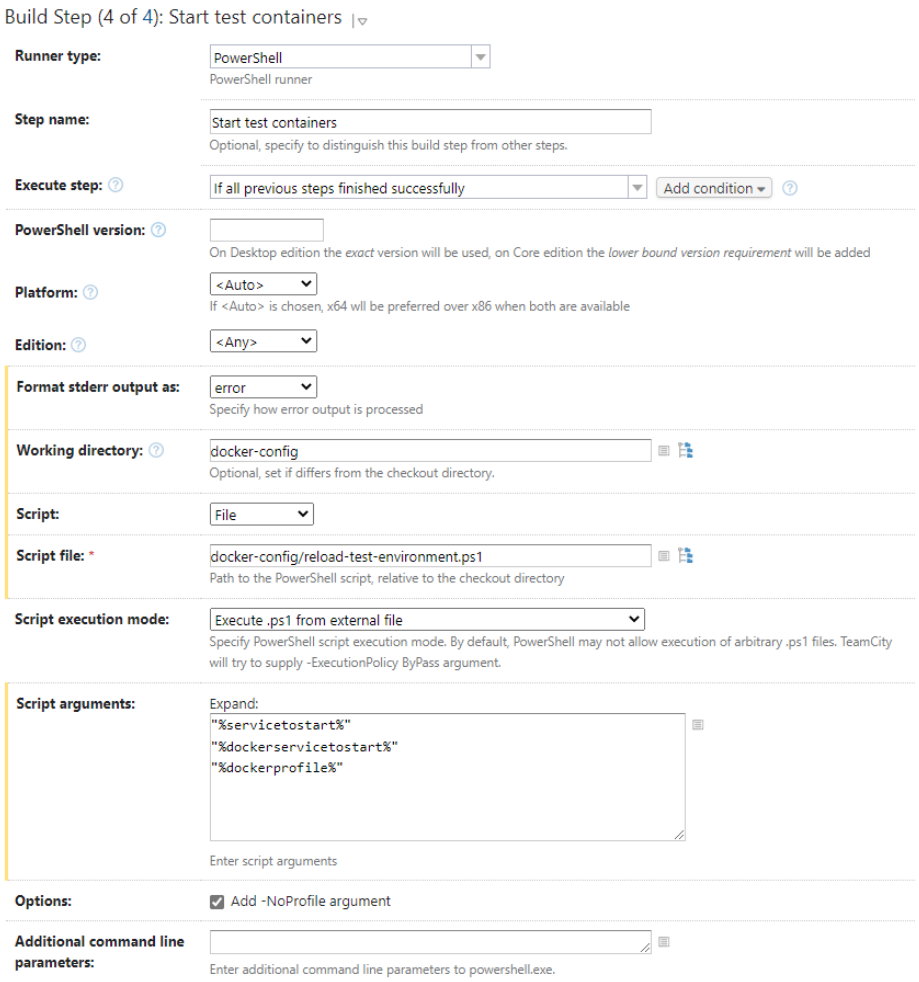

Figure 36 – Configuration of test environment configuration through PowerShell scripting

With this, the transaction is performed without human intervention. In figure 37, it is shown the application running from a certain host machine (identified by the hostname), and in figure 38 from another host machine.

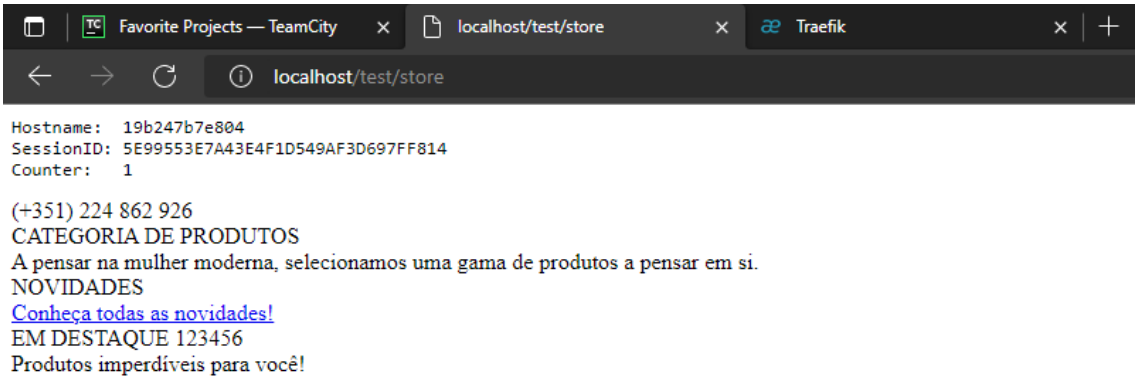

Figure 37 – Test environment executing before the execution of the switch script

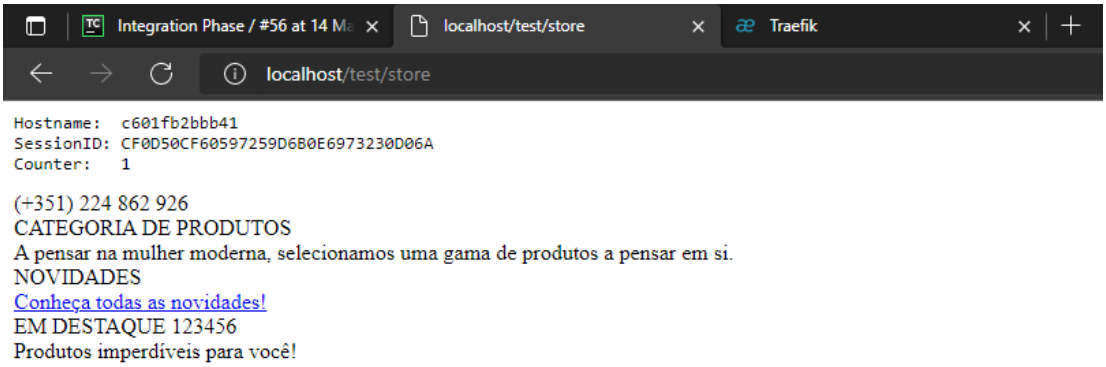

Figure 38 - Test environment executing after the execution of the switch script

5.5.3 Session replication

The user session, however, is lost between the version switching, as it happened with the old process as described in subsection 4.1. The two figures above (figure 37 and figure 38) clearly show the host machine and the session being changed, when one of the goals is to keep the current user session.

One possible solution would be sharing the user sessions between containers, before the old container is deleted. Tomcat does have a native session sharing system, internally designated as a cluster system, on which multiple Tomcat instances share sessions between them if they have the same service name. Since the containers are running on an isolated network, the chances of someone hijacking a session while being replicated is reduced. To use it, it is necessary to add the following tags on the server.xml file on Tomcat, inside the Engine tag (code 6):

```
1       <Cluster
    className="org.apache.catalina.ha.tcp.SimpleTcpCluster">
2           <Channel
    className="org.apache.catalina.tribes.group.GroupChannel">
3                 <Membership
4                       className="org.apache.catalina.tribes.memb
    ership.cloud.CloudMembershipService"
5                       membershipProviderClassName="org.apache.ca
    talina.tribes.membership.cloud.DNSMembershipProvider"/>
6           </Channel>
7       </Cluster>
```

Code 6 –Tags to configure a basic cluster on Tomcat

Since the image is created on TeamCity while using a Dockerfile, and it is necessary to inject a server.xml file with the cluster configuration, it was added to the Dockerfile a command to copy a server.xml already prepared into Tomcat before building the image, as shown below on code 7.

```
1  FROM tomcat:9.0.62-jdk11-corretto
2  COPY build/web/ /usr/local/tomcat/webapps/test#store
3  COPY docker-config/server.xml /usr/local/tomcat/conf/
4  CMD ["catalina.sh", "run"]
```

Code 7 – Dockerfile with server.xml copy to the image

The Tomcat instance also requires the specification of the service name, when starting up. This is done on the Docker Compose file, in the “environment” section, as shown below in code 8.

```
1  environment:
2        - DNS_MEMBERSHIP_SERVICE_NAME=epharmastore-test-app
```

Code 8 – Environment variables configuration for Tomcat on Docker Compose file

The application “web.xml” file requires the “<distributable/>” element for the application to know that the sessions of this application may be shared between instances. As a result, figure 39 shows the application already running with the session replication.

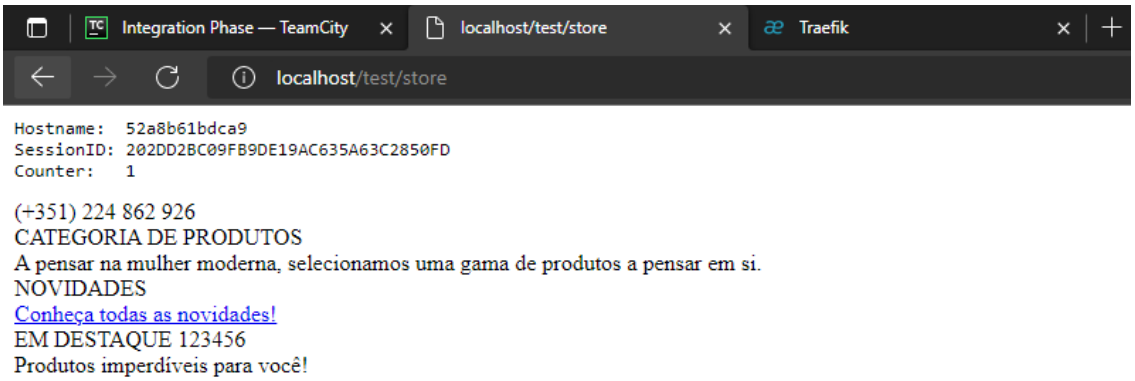

Figure 39 - Test environment executing before the execution of the switch script with session sharing

Figure 40 was captured after the execution of the integration pipeline. It shows that the host machine name has changed, but the session identifier is equal, and the access counter increased by one.

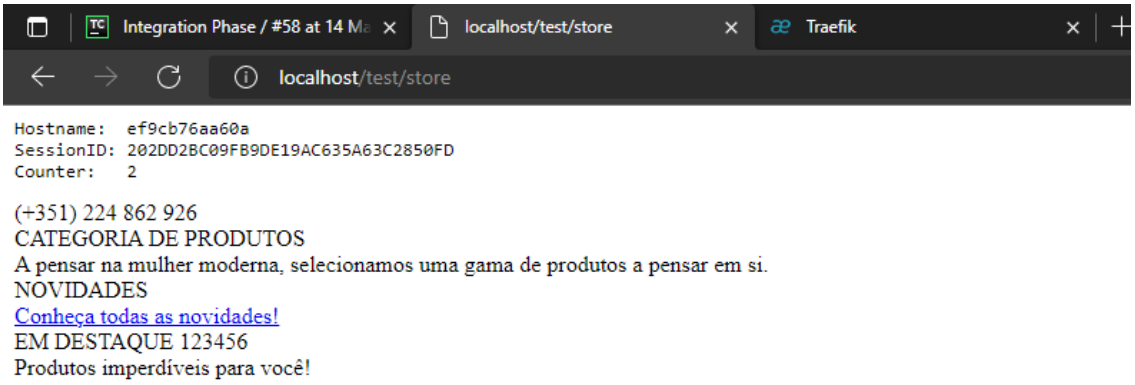

Figure 40 - Test environment executing after the execution of the switch script with session sharing

5.6 Requirements fulfillment

According to the description performed during this chapter, it is possible to conclude that the automation of the integration and deployment to multiple environments was achieved. TeamCity proved itself capable of performing all the necessary tasks without requiring extra steps such as plugin downloads, with an interaction experience that did not have a huge learning curve. In terms of security, it was possible to create various users with different permissions, so that the developers could view pipeline details and only a smaller group could grant permission to deploy a new production version.

The technique employed to diminish the downtime between versions was also implemented successfully. Using Traefik as an API gateway and service discovery while executing a script that creates a container with a new version and stops the old version was a good choice, reducing the number of different tools used and integrating directly with Docker, given the correct configuration in the Docker Compose file.

6 Evaluation

As stated before, the main goal of this work was to improve some of the organization processes, namely the integration and deployment process, resulting in an increase in the developer's productivity and a more reliable product. According to the previous chapter, the implementation was performed successfully, and now it exists an automation system for the integration and deployment tasks, as well as the application of a zero-downtime technique. In this chapter, it is specified what factors were considered to evaluate the success of the implementation and how the evaluation was conducted.

6.1 Evaluation Criteria

On section 1.3, it was specified two main research questions:

1. How to perform an automatization of the process of source code compilation, testing, and deployment

2. How to decrease the downtime period associated with the update process, preferentially using a technique that allows the deployment of an updated version while an old one is still being used and slowly transfer the traffic to the latest version

To evaluate the complete work, it is necessary to define research hypothesis. This hypothesis will help confirm if the research questions are answered correctly by validating associated metrics.

For the first question, it is possible to formulate a hypothesis: "The automation process helped the organization work better and deploy faster", with the following metrics being evaluated:

1. User satisfaction regarding the developed automation pipeline

2. Average time to deploy a new version to the test server

For the second question, the hypothesis is “The downtime period while updating an application decreased”, with a single metric being evaluated:

1. Average downtime while updating an application

All the statistical analysis that are presented were performed using RStudio. This tool is an IDE for R, a programming language mostly used to compute statistical data and graphs (“RStudio | Open Source & Professional Software for Data Science Teams - RStudio” n.d.; “R: What Is R?” n.d.).

Regarding the definition of hypothesis, H_0 always corresponds to the null hypothesis, and H_1 always corresponds to the alternative hypothesis.

6.2 User satisfaction regarding the developed automation pipeline

The main goal, while using this metric, is to understand how the affected users inside the organization feel about the usage of the automation pipeline. The group of users that will be interviewed must interact directly with the system, to understand if the automation simplified some of their tasks, or with a result of the process, for testing purposes or to check if the latest functionality was integrated correctly, for example.

On this metric, an inquiry will be created to gather information from the different users. Since it is irrelevant how the process was implemented, the questions will only focus on general transversal aspects, allowing answers from users that implemented the process and users that only use the results of the process.

Each question will correspond to an affirmation about the process (“The waiting time to test new functionalities has reduced”, for example), and the answer will correspond to a level, from zero (0) to ten (10) scale, representing the degree of agreement with the affirmation, with zero representing complete disagreement, ten representing complete agreement and five (5) representing neutrality.

For each question, the mean of the results is compared with seven (7), which is considered the turning point between neutral and positive. If the obtained value is bigger than expected, it means that the implementation had a positive impact.

6.2.1 Questionnaire structure

The questionnaire was elaborated on Microsoft Forms, an online tool that allows the creation of questionnaires. Since all the collaborators speak primarily Portuguese, the questions were elaborated in that language, although it is possible to select English to answer the questionnaire.

The following sentences were presented:

1. The reduction of the manual intervention by a user helped reduce the time spent on the whole process

2. It is possible to detect faster errors or unexpected situations that occurred on a system different from mine, including compilation or implementation of flow errors

3. The time spent on more productive or challenging tasks increased, while the time spent updating an environment decreased

4. The waiting time to test new functionalities in the test environment decreased

5. It is possible to update an environment with a reduced risk of affecting active users

6. The implementation of the integration and deployment pipelines will help shape the future of other projects in the organization

For each question, the inquired would select a value, from one to ten, representing the agreement with the sentence. The questionnaire is present on appendix B.

6.2.2 Measurements

As specified before, a questionnaire was built in Microsoft Forms, and sent to six collaborators that, as referenced before, either intervenes directly in the integration and deployment process or uses the result of the process (for testing purposes, for example). The results were registered below on table 2, where each line corresponds to a question and each column to an answer. The question number corresponds to the question identified with the same number on subsection 6.2.

Table 2 – Questionnaire answers

Question	Answer 1	Answer 2	Answer 3	Answer 4	Answer 5	Answer 6
1	8	10	10	8	10	9
2	9	10	10	6	8	8
3	5	10	10	8	10	10
4	8	10	10	8	7	10
5	8	10	10	10	10	8
6	9	10	10	10	9	9

6.2.3 Analysis

As specified at the beginning of subsection 6.2, for an aspect of the implementation be analyzed as positive, the average of the answer must be bigger than 7. To evaluate each question, it is possible to use a one-sample t-test to compare the mean of a sample with normal distribution

with the predefined value, if the data has a normal distribution. If it is not possible to execute the t-test due to the dataset not having a normal distribution, the alternative to be considered is the Wilcoxon test. This test also compares the mean of a dataset with a predefined value, but it does not require a dataset with normal distribution.

For each question, it is necessary to check if the sample has a normal distribution and only then it is possible to execute the t-test. To test the normal distribution, it was used a Shapiro-Wilk test, with the following hypothesis:

H_0 – the population follows a normal distribution

H_1 – the population does not follow a normal distribution

Since the average to be surpassed on each test is always 7, it was defined two hypotheses to be applied on both t-test and Wilcoxon test:

H_0 – the average is equal or smaller than 7

H_1 – the average is bigger than 7

6.2.3.1 Question 1 – The reduction of the manual intervention by a user helped reduce the time spent on the whole process

All the answers for this question were above or equal to 8, which means that the perceived time spent on the integration and deployment reduced after the implementation.

To check the answers average, as mentioned before, it was firstly executed the Shapiro-Wilk test to check the distribution of the data, as shown below on figure 41.

```
> shapiro.test(q1)

        Shapiro-Wilk normality test

data:  q1
W = 0.77516, p-value = 0.03473
```

Figure 41 – Shapiro-Wilk test for the data regarding the first question

Assuming a confidence level of 95%, a significance level of 0.05 and considering the hypothesis regarding the normal distribution specified at the beginning of this subsection, it is possible to discard H_0 and assert that the population does not follow a normal distribution. The obtained result does not allow the execution of the t-test, being necessary to perform a Wilcoxon test instead. The result is presented below on figure 42.

```
> wilcox.test(q1, mu = 7,  alternative = "greater")

        Wilcoxon signed rank test with continuity correction

data:  q1
V = 21, p-value = 0.01675
alternative hypothesis: true location is greater than 7
```

Figure 42 – Wilcoxon test results for the first question

Following the hypothesis defined in the beginning of this subsection regarding the average evaluation, and assuming a confidence interval of 95% and a significance level of 0.05, the *p* value is lower than the significance level, for which it is possible to discard H_0 and affirm that the response average is above 7.

6.2.3.2 Question 2 – It is possible to detect faster errors or unexpected situations that occurred on a system different from mine, including compilation or implementation of flow errors

Most of the answers were positive, with only one being below 7. This translates into 83.33% of the inquired, approximately, agreeing that the implementation helped detect failures that were not detected locally.

The Shapiro-Wilk test was executed, with the results being presented on figure 43, to verify if the dataset has a normal distribution.

```
        Shapiro-Wilk normality test

data:  q2
W = 0.90248, p-value = 0.3888
```

Figure 43 – Shapiro-Wilk test for the data regarding the second question

Following the hypothesis defined in the beginning of this subsection regarding the normal distribution test, and assuming a confidence level of 95% and a significance level of 0.05, it is not possible to discard H_0, which means that the dataset can have a normal distribution.

It was performed a t-test for the dataset of the question, assuming the hypothesis presented at the beginning of the subsection.

```
> t.test(q2, mu = 7, alternative = "greater")

        One Sample t-test

data:  q2
t = 2.4227, df = 5, p-value = 0.02996
alternative hypothesis: true mean is greater than 7
95 percent confidence interval:
 7.252405      Inf
sample estimates:
mean of x
      8.5
```

Figure 44 – T-test results for the second question

According to the results obtained on the t-test, shown above on figure 44, the *p* value is 0.02996. Considering the hypothesis defined in the beginning of this subsection regarding the average evaluation, a confidence level of 95% and a significance level of 0.05, since the *p* value is lower than the significance level, it is possible to discard H_0 and assume that the dataset average for this question is above 7.

6.2.3.3 Question 3 – The time spent on more productive or challenging tasks increased, while the time spent updating an environment decreased

Only one answer out of six was below 7, which indicates that approximately 83.33% of the inquired agreed that the time spent on tasks other than integration and deployment decreased, being replaced by other tasks.

Regarding statistical tests, firstly it was executed the Shapiro-Wilk test to check if the dataset has a normal distribution. The result is registered below on figure 45.

```
        Shapiro-Wilk normality test

data:  q3
W = 0.68449, p-value = 0.004226
```

Figure 45 - Shapiro-Wilk test for the data regarding the third question

Assuming a confidence interval of 95%, a significance level of 0.05, and the hypothesis defined in the beginning of this subsection regarding the normal distribution, it is possible to discard H_0 and assume that the data does not have a normal distribution.

Since the data does not have a normal distribution, a Wilcoxon test was performed, with the results being presented below on figure 46.

```
        Wilcoxon signed rank test with continuity correction

data:  q3
V = 19, p-value = 0.04223
alternative hypothesis: true location is greater than 7
```

Figure 46 - Wilcoxon test results for the third question

Assuming a confidence interval of 95%, a significance level of 0.05, and the hypothesis defined in the beginning of this subsection regarding the average evaluation, it is possible to discard H_0 and assume that the average is above 7.

6.2.3.4 Question 4 – The waiting time to test new functionalities in the test environment decreased

Only one answer is equal to 7, with the remaining dataset being bigger than the reference value. For a majority of the inquired it was perceived a decreased in the time to test new functionalities in the test environment.

In terms of answers average, firstly it was executed the Shapiro-Wilk test to check if the answer dataset has a normal distribution, with the result being presented below on figure 47.

```
        Shapiro-Wilk normality test

data:  q4
W = 0.80514, p-value = 0.06534
```

Figure 47 - Shapiro-Wilk test for the data regarding the fourth question

Assuming a confidence interval of 95%, a significance level of 0.05, and the hypothesis defined in the beginning of this subsection regarding the normal distribution, it is not possible to discard H_0. Since it is impossible to discard the hypothesis that assumes that the dataset has a normal distribution, it is possible to execute a t-test, with its results being presented below on figure 48.

```
        One Sample t-test

data:  q4
t = 3.3786, df = 5, p-value = 0.009852
alternative hypothesis: true mean is greater than 7
95 percent confidence interval:
 7.739913      Inf
sample estimates:
mean of x
 8.833333
```

Figure 48 – T-test results for the fourth question

Considering a confidence interval of 95% and the hypothesis regarding the average presented at the beginning of the subsection, the *p* value obtained is lower than the significance level (0.05), which allows the discard of H_0 and affirm that the mean for this dataset is above 7.

6.2.3.5 Question 5 – It is possible to update an environment with a reduced risk of affecting active users

All the answers for this question are above 7, with 66.67% (approximately) of the inquired affirming that completely agrees that it is possible to perform updates without disrupting active users. Regarding the average metric, it was executed the Shapiro-Wilk test to check if the dataset has a normal distribution.

```
        Shapiro-Wilk normality test

data:  q5
W = 0.63989, p-value = 0.001351
```

Figure 49 - Shapiro-Wilk test for the data regarding the fifth question

According to figure 49, presented above, the *p* value is 0.001351. Considering a confidence interval of 95%, a significance level of 0.05 and the hypothesis presented at the beginning of this subsection, it is possible to discard the null hypothesis and assume that the dataset does not have a normal distribution.

Since it does not have a normal distribution, the Wilcoxon test was executed for this dataset, with the results being presented below on figure 50.

```
        Wilcoxon signed rank test with continuity correction

data:  q5
V = 21, p-value = 0.01527
alternative hypothesis: true location is greater than 7
```

Figure 50 - Wilcoxon test results for the fifth question

Considering a confidence interval of 95% and the hypothesis regarding the average presented at the beginning of the subsection, the *p* value obtained is lower than the significance level (0.05), which discards H_0 and allows to affirm that the mean for this dataset is above 7.

6.2.3.6 Question 6 – The implementation of the integration and deployment pipelines will help shape the future of other projects in the organization

All the inquired answer to this question either was a 9 or a 10, with the distribution being 50% for each result. This is the only question on which the answers were all above or equal to 9.

In terms of average evaluation, it was executed the Shapiro-Wilk test to check if the dataset has a normal distribution. The results are presented below on figure 51.

```
        Shapiro-Wilk normality test

data:  q6
W = 0.68268, p-value = 0.004039
```

Figure 51 - Shapiro-Wilk test for the data regarding the sixth question

Assuming the hypothesis defined at the beginning of the subsection and a confidence interval of 95%, it is possible to discard H_0, which implies that the dataset does not have a normal distribution. The Wilcoxon test was executed for this dataset, with the result obtained being presented below on figure 52.

```
        Wilcoxon signed rank test with continuity correction

data:  q6
V = 21, p-value = 0.01601
alternative hypothesis: true location is greater than 7
```

Figure 52 - Wilcoxon test results for the sixth question

Considering a confidence interval of 95% and the hypothesis regarding the average presented at the beginning of the subsection, the *p* value obtained is lower than the significance level (0.05), which discards H_0 and allows to affirm that the mean for this dataset is above 7.

6.2.4 Conclusions

Overall, the answers were very positive, with very answer surpassing the turning point specified at the beginning of subsection 6.2 (7), meaning that all goals of the implementation that were described on the questionnaire were well perceived by the inquired.

The question with the higher average corresponds to future application of the project inside the organization. This could indicate that the inquired would like to set up on new projects inside the organization the same processes that were previously described.

6.3 Average time to deploy an updated version to the test server

On this metric, the objective is to understand if the time to deploy a new version, after being integrated, changed compared to the manual process. To achieve this goal, it is necessary to obtain two samples of data, each corresponding to each process (manual and automated), in

equal quantity. Each measurement must correspond to the execution of the same tasks, which includes the integration of the application (build and automated tests).

After obtaining the data, it is possible to calculate the mean and compare the two datasets, to conclude if the average time suffered any change since the implementation of the automated pipeline and if it decreased, as initially expected.

6.3.1 Testing approach

As stated before, the previous integration and deployment process was manual, and there are no formal registries of time spent while performing said tasks. Therefore, it was necessary to retrieve the values in this phase. The manual process was previously described in subsection 4.1, with the activity flow shown in figure 3.

The second dataset, corresponding to the results of the automated pipeline, will be obtained directly from the automation tool.

Between each measurement, it was performed some changes on the source code repository, to trigger the automatic integration process and guarantee that the build phase could not be skipped.

6.3.2 Measurements

As stated before, it was performed five changes on the source code, and for each change the project was subjected to the manual and automatic integration and deployment to the test environment process.

The table below shows the time spent on each measurement for each process type, in seconds.

Table 3 – Integration and deployment time for automatic and manual processes

Measurement	Manual process time, in seconds	Automatic process time, in seconds
1	380	338
2	390	384
3	340	372
4	353	335
5	374	333

6.3.3 Analysis

To analyze the difference between the average of the two datasets, it can be performed a paired t-test. This test requires a normal distribution in the population composed by the difference between each pair of measurements for small samples and the samples must be

paired. Since each measurement corresponds to the same sample on different processes, the data is paired.

To test the normal distribution, it was used a Shapiro-Wilk test. It was considered two hypotheses, H_0 being that the population described before follows a normal distribution and H_1 being that the population described before does not follow a normal distribution.

According to the figure below (figure 53), the *p* value of the difference between measurements dataset is 0.3918. It was considered a confidence interval of 95 and a significance level of 0.05. With *p* being bigger than the significance level, it is not discarded the normal distribution of the population.

```
> process_time_difference <- with(process_time_df, process_time_df$manual_process_time -
 process_time_df$automatic_process_time)
> shapiro.test(process_time_difference)

        Shapiro-Wilk normality test

data:  process_time_difference
W = 0.89669, p-value = 0.3918
```

Figure 53 – Shapiro-Wilk test for the difference between the manual process and automatic process measurements

It is now possible to execute a paired t-test. Assuming a confidence interval of 95%, with a significance level of 0.05, it was written two hypotheses:

H_0 – the process time average increased or stayed the same

H_1 – the process time average decreased

Using RStudio to compute the result, it was possible to obtain the values as shown below on figure 54.

```
> process_time_ttest_result <- t.test(manual_process_time, automatic_process_time, pai
red = TRUE)
> process_time_ttest_result

        Paired t-test

data:  manual_process_time and automatic_process_time
t = 1.1022, df = 4, p-value = 0.3322
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -22.78414  52.78414
sample estimates:
mean difference
             15
```

Figure 54 – Paired t-test execution result for comparison between manual and automatic process time

The *p* value is bigger than the significance level, which does not allow to discard H_0, meaning that it is not possible to conclude that the average time decreased with the new automatic process.

6.3.4 Conclusions

According to the statistical analysis performed in the previous subsection, it was not possible to conclude that the average time suffered a decrease.

Regarding the specified requirements, it was stated in subsection 4.2.2 that the average time could only be maintained or decreased. Although it is not possible to assume that the average time of the process suffered a reduction, it is more important to check if the developer's time spent on this process decreased, since the time registered on the manual process corresponds to a single developer executing all the actions and stopping his/her tasks, while the automatic process time does not have human intervention.

The opinion of the developers was registered on the previous metric, presented in subsection 6.2, and it shows that most of the inquired agrees that the time spent on integration and deployment tasks has decreased, increasing the time spent on more productive tasks.

6.4 Average downtime while updating an application

This metric will be used to determine if the downtime caused by an update is reduced with the new method, in comparison with the old manual process.

To obtain data for this metric, it is necessary to measure in equal quantity the downtime after an update for both methods. This downtime period corresponds from the moment that the application starts until the main page is presented to the end-user.

After both sets of times are retrieved, it is possible to calculate the mean and, by comparing them, it is possible to conclude if the average downtime while updating suffers a decrease, as expected, or did not decrease at all.

6.4.1 Testing approach

For this metric, it was required to measure the downtime when updating with both processes, in equal quantity. To measure the downtime, it was used Apache JMeter. This tool can be used to perform load and performance tests on web applications, while compiling all the collected information on the user interface or in exportable reports (Apache JMeter - Apache JMeterTM, no date).

To configure a test suite in JMeter, it is necessary to create a thread group. A thread group contains one or more threads, that will execute test actions such as HTTP requests and compilation of results. A thread can be seen as a user and it is possible to set the number of threads in a group, the number of times to execute the test plan inside the group, and the time interval between loops.

The test consists of letting JMeter execute requests, one per second, and register the time between the first HTTP response failure with the HTTP status code 503 (service unavailable) and the first positive result after the error (HTTP status code 200). The configuration of the HTTP requests on JMeter, aside from the basic (protocol, host, and path), also included a timeout period. This configuration is important to stop a request that might hang on while the server is finishing its startup, disturbing the remaining results.

Since JMeter can automatically calculate the average response time of a set of requests and its deviation, the first step stipulated was the execution of two sets of 10000 HTTP requests for the home page of the application, one for each deployment approach (Tomcat on machine or containerized application). With the average response time and the deviation, the timeout was defined as the double of the average plus the deviation time.

6.4.2 Measurements

The first environment to be tested was the containerized one. As described before, the first time was the calculation of an average response time to set the timeout. The average time was 7 milliseconds, with a deviation of 2 milliseconds. With this information, the calculated timeout was 16 milliseconds.

The measurements for the automatic process were registered below in table 4.

Table 4 – Downtime period for the automatic process

Measure	First HTTP response failure timestamp	First HTTP success response after failure timestamp	Elapsed time (seconds)
1	12:30:28.235	12:30:32.101	3.87
2	14:35:25.864	14:35:27.491	1.63
3	14:39:17.085	14:39:19.263	2.18
4	14:45:23.363	14:45:25.852	2.49
5	14:48:42.812	14:48:45.645	2.83

Lastly, the Tomcat has an average response time of 1 millisecond, with a deviation of 10 milliseconds. Using the method previously presented, the timeout was calculated and defined as 12 milliseconds. After configuring the thread group, the following measures were taken as shown in table 5.

Table 5 – Downtime period for the manual process

Measure	First HTTP response failure timestamp	First HTTP success response after failure timestamp	Elapsed time (seconds)
1	16:07:33.558	16:07:43.471	9.91
2	16:13:49.208	16:13:58.541	9.33
3	16:16:03.521	16:16:12.786	9.26
4	16:24:46.317	16:24:55.595	9.27
5	16:27:09.602	16:27:19:330	9.73

6.4.3 Analysis

To check the average difference, it can be used an unpaired two-samples t-test. This test compares the mean of two independent groups, being adequate for this case. It is necessary, however, to check if both groups are normally distributed and the variance of the two groups are equal.

Using RStudio, it is possible to perform all tests previously mentioned. To test the normal distribution, it was used a Shapiro-Wilk's test. For both datasets, it was considered two hypotheses, H_0 being that the population described before follows a normal distribution, and H_1 being that the population described before does not follow a normal distribution.

After loading the data in RStudio, the Shapiro-Wilk's test was executed, with the results being presented below on figure 55.

```
> shapiro.test(new_process)

        Shapiro-Wilk normality test

data:  new_process
W = 0.96889, p-value = 0.8681

>
> shapiro.test(old_process)

        Shapiro-Wilk normality test

data:  old_process
W = 0.81824, p-value = 0.1132
```

Figure 55 – Shapiro-Wilk test results for the downtime datasets

Assuming a confidence interval of 95% and a significance level of 0.05, it is not possible to discard H_0 since both *p* values are above the significance level.

To test the variance of the populations, and since both populations can have a normal distribution as checked before, it was used the F-test to verify the second requirement for the t-test. This test compares the variance on two populations. It was considered two hypotheses: H_0 being that the variance of the two datasets is equal, and H_1 being that the variance of the two datasets is different. Figure 56 below shows the result of the test execution:

```
> ftest_result <- var.test(old_process, new_process)
> ftest_result

        F test to compare two variances

data:  old_process and new_process
F = 0.12894, num df = 4, denom df = 4, p-value = 0.0723
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.01342443 1.23836311
sample estimates:
ratio of variances
         0.1289353
```

Figure 56 – F-test results for the downtime datasets

Assuming a confidence interval of 95% and a significance level of 0.05, it is not possible to discard H_0, therefore validating the last requirement for the unpaired t-test.

For the t-test, it was considered two hypotheses:

H_0 – the downtime period average increased or stayed the same

H_1 – the downtime period average decreased

By executing the test in RStudio, it was obtained the results presented in the following figure (figure 57):

```
> downtime_test_ttest_result

        Two Sample t-test

data:  old_process and new_process
t = 17.371, df = 8, p-value = 1.229e-07
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 5.98402 7.81598
sample estimates:
mean of x mean of y
      9.5       2.6

> |
```

Figure 57 – Unpaired two sample t-test results for the downtime datasets

Considering a confidence interval of 95% and a significance level of 0.05, the *p* value is smaller than the significance level. Therefore, it is possible to discard H_0 and affirm that the downtime average decreased.

The test also shows that the downtime average of the old process is 9.5 seconds, and the new process is 2.6 seconds. The reduction percentage between both averages is 72.63%, approximately.

6.4.4 Conclusions

Although a zero-downtime technique was used to achieve these results, it still exists a small period during which the application is not responsive. However, the data analysis performed indicates that it is possible to assume that the downtime period decreased.

Using both downtime averages obtained during the test, it was registered a reduction of 72.63%, approximately. According to the criteria defined in subsection 4.2.2, this reduction exceeds the expected value of 50%.

7 Conclusion

This last section is used to obtain an overall panorama of the project, if the research questions were promptly answered, as well as discussed future works that can be done regarding the project.

7.1 Research questions and results

At the beginning of the project, it was stated two research questions:

- How to perform an automatization of the process of source code compilation, testing, and deployment

- How to decrease the downtime period associated with the update process, preferentially using a technique that allows the deployment of an updated version while an old one is still being used and slowly transfer the traffic to the latest version

Regarding the first question, two metrics were used to determine the answer. The first metric, stated in subsection 6.2, reveals that the process was well received by the organization members directly affected by the process, with all the affirmations regarding the implementation being agreed by the inquired. It is possible to conclude, according to the questionnaire data, that the collaborators spent less time on integration and deployment tasks, it existed an increment of the productivity, and it is desirable to apply the described processes on new projects that may arise. Furthermore, the question with bigger results refers to the application of the implementation on other projects in the organization.

The other metric related to the first research question, described in subsection 6.3, revealed that it is not possible to determine a reduction in the integration and deployment process to the test environment, according to the statistical tests performed. However, and as revealed by the questionnaire, the time that was previously occupied by integration and deployment tasks

is used for other tasks. Therefore, although it was not measured a drastic reduction between processes, the time that a user interacts directly with the process decreased, compensating the non-reduction of the overall time.

The last research question, evaluated by the metric described in subsection 6.4, shows that the average downtime period decreased, with the average downtime period from the data obtained registering a decrease of 72.63% between the manual and the automatic process. The implementation also removed the session loss that was present in the old method, which means that an update to any environment will not result in a user losing its session, assuring that an update during working hours is possible with fewer concerns.

7.2 Future work

Now that there is empirical evidence that the project implementation helped the organization in the integration and deployment of a single project, one of the next steps that can be conducted is the application of the process for other applications. Although the technological basis is the same, some key aspects differ such as the configuration file format, and it requires extra configuration on TeamCity.

Another concern is the differentiation between different clients for a single product. This would require, for example, a robust branching system and different configurations on TeamCity for each branch, that would represent a client. There are also social implications while implementation a system like this, as stated in the contextualization in the state of the art (subsection 2.1), and each case must be thoroughly analyzed and planned before advancing with the implementation.

In terms of implementation, the containerization of the database, instead of being hosted remotely, can help decrease the response time of the application, and reduce the workload on a single node to access data. The database update process should be also revised, since currently it is a manual task, requiring a developer to perform changes to the database schema prior or after an update. Although now it does not have a high impact, since it is expected to only have one instance of an application running in a certain version, it is an improvement that should be studied and applied, if adequate.

It was not possible to fully eliminate the downtime period between version switching, although the user sessions are not lost with the new process. Tinkering the switch script could help achieve a downtime of zero seconds.

7.3 Final considerations

The automation of the integration and the deployment system helped different actors of the process, starting from the developers, that are more productive and have less repetitive and

error-prone tasks, to the final consumer, that can receive updates faster. The implementation of a zero-downtime technique also benefited both sides, allowing the execution of maintenance and feature updates with minimal disruption to the end-user.

The research questions previously presented were answered on a positive note, which further confirms that the implementation impact positively the organization integration and deployment process for this project. Although there are required some adjustments to apply to other projects in the organization, as stated in future work (subsection 7.2), it is plausible to assume that the implementations described on this document could benefit other projects in the organization.

References

"Advantages and Disadvantages of Jenkins - Tutorial And Example". 2021. January 22, 2021 <https://www.tutorialandexample.com/advantages-and-disadvantages-of-jenkins/> [accessed 5 January 2022].

"Apache JMeter - Apache JMeter™". n.d. accessed May 30, 2022 <https://jmeter.apache.org/> [accessed 30 May 2022].

Artac, Matej, Tadej Borovssak, Elisabetta di Nitto, Michele Guerriero and Damian Andrew Tamburri. 2017. "DevOps: Introducing Infrastructure-as-Code". *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017*, June, 497–498. <https://doi.org/10.1109/ICSE-C.2017.162> [accessed 2 November 2021].

Chen, Lianping. 2015. "Continuous Delivery: Huge Benefits, but Challenges Too". *IEEE Software* 32: 50–54. <https://doi.org/10.1109/MS.2015.27> [accessed 18 December 2021].

Chubb, Iris. 2021. "16 Jenkins Alternatives for Continuous Integration in 2021". June 1, 2021 <https://blog.inedo.com/jenkins/alternatives-for-continuous-integration> [accessed 5 January 2022].

Claps, Gerry Gerard, Richard Berntsson Svensson and Aybüke Aurum. 2015. "On the Journey to Continuous Deployment: Technical and Social Challenges along the Way". *Information and Software Technology* 57: 21–31. <https://doi.org/10.1016/J.INFSOF.2014.07.009> [accessed 18 December 2021].

"Continuous Integration (CI) Tools Comparison: TeamCity vs Jenkins vs Bamboo | AltexSoft". 2019. February 19, 2019 <https://www.altexsoft.com/blog/engineering/comparison-of-most-popular-continuous-integration-tools-jenkins-teamcity-bamboo-travis-ci-and-more/> [accessed 5 January 2022].

Dannana, Sundar. 2020. "Function Analysis and System Technique - FAST Diagram - ExtruDesign". February 19, 2020 <https://extrudesign.com/function-analysis-and-system-technique-fast-diagram/> [accessed 3 February 2022].

"DevOps Stats for Doubters | UpGuard". n.d. accessed February 6, 2022 <https://www.upguard.com/blog/devops-success-stats> [accessed 6 February 2022].

Dresch, Aline, Daniel Pacheco Lacerda and Paulo Augusto Cauchick Miguel. 2015. "A Distinctive Analysis of Case Study, Action Research and Design Science Research". *Review of Business Management* 17: 1116–1133. <https://doi.org/10.7819/RBGN.V17I56.2069> [accessed 15 January 2022].

Fowler, Martin. 2006. “Continuous Integration”. May 1, 2006 <https://www.martinfowler.com/articles/continuousIntegration.html> [accessed 24 December 2021].

Gandhi, Rajeev. 2019. “The Benefits of Containerization and What It Means for You | IBM”. 2019 <https://www.ibm.com/cloud/blog/the-benefits-of-containerization-and-what-it-means-for-you> [accessed 3 February 2022].

“GitLab.Org / GitLab · GitLab”. n.d. accessed January 6, 2022 <https://gitlab.com/gitlab-org/gitlab> [accessed 6 January 2022].

Glinz, Martin. 2007. “On Non-Functional Requirements”. [accessed 27 January 2022].

Guerriero, Michele, Martin Garriga, Damian A. Tamburri and Fabio Palomba. 2019. “Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry”. *Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019*, September, 580–589. <https://doi.org/10.1109/ICSME.2019.00092> [accessed 2 November 2021].

Guyton, Stephen. 2019. “Containers & Containerization – The Pros and Cons”. May 24, 2019 <https://spin.atomicobject.com/2019/05/24/containerization-pros-cons/> [accessed 3 February 2022].

Iivari, Juhani and John R Venable. 2009. “Action Research and Design Science Research-Seemingly Similar but Decisively Dissimilar Recommended Citation”, 2009. <http://aisel.aisnet.org/ecis2009/73> [accessed 15 January 2022].

“Implementing DevOps: 3 Surprising Success Stories | Pega”. 2018. 2018 <https://www.pega.com/insights/resources/implementing-devops-3-surprising-success-stories> [accessed 5 February 2022].

“Integrating TeamCity with Docker | TeamCity On-Premises”. 2021. November 2, 2021 <https://www.jetbrains.com/help/teamcity/integrating-teamcity-with-docker.html> [accessed 5 January 2022].

“Jenkins 2 Overview”. n.d. accessed January 5, 2022 <https://www.jenkins.io/2.0/> [accessed 5 January 2022].

Jethva, Hitesh. 2021. “Jenkins vs CircleCI – What’s the Difference? (Pros and Cons)”. December 3, 2021 <https://cloudinfrastructureservices.co.uk/jenkins-vs-circleci/> [accessed 5 January 2022].

Koen, P., Greg Ajamian, S. Boyce, A. Clamen, E. Fisher, S. Fountoulakis, Albert Johnson, P. Puri and Rebecca Seibert. 2002. “1 Fuzzy Front End : Effective Methods , Tools , and Techniques”. [accessed 3 February 2022].

Laukkanen, Eero, Maria Paasivaara and Teemu Arvonen. 2015. “Stakeholder Perceptions of the Adoption of Continuous Integration-A Case Study”. *Proceedings - 2015 Agile Conference, Agile 2015*, September, 11–20. <https://doi.org/10.1109/AGILE.2015.15> [accessed 18 December 2021].

Leppänen, Marko, Simo Mäkinen, Max Pagels, Veli Pekka Eloranta, Juha Itkonen, Mika v. Mäntylä and Tomi Männistö. 2015. “The Highways and Country Roads to Continuous Deployment”. *IEEE Software* 32: 64–72. <https://doi.org/10.1109/MS.2015.50> [accessed 3 February 2022].

Malviya, Rishabh. 2021. “Create a Zero-Downtime Deployment of Your Machine Learning API | by Rishabh Malviya | Better Programming”. January 11, 2021 <https://betterprogramming.pub/create-a-zero-downtime-deployment-of-your-machine-learning-api-6486cb6394c3> [accessed 22 April 2022].

Mansfield, Timothy. 2019. “Value Proposition Canvas Explained: How to Match Your Services to Customer Needs”. August 7, 2019 <https://interaction.net.au/articles/value-proposition-canvas-explained/> [accessed 3 February 2022].

“Maturity | GitLab”. n.d. accessed January 6, 2022 <https://about.gitlab.com/direction/maturity/> [accessed 6 January 2022].

Meyer, Mathias. 2014. “Continuous Integration and Its Tools”. *IEEE Software* 31: 14–16. <https://doi.org/10.1109/MS.2014.58> [accessed 18 December 2021].

Miller, Ade. 2008. “A Hundred Days of Continuous Integration”, August, 289–293. <https://doi.org/10.1109/AGILE.2008.8> [accessed 18 December 2021].

Naik, Vishal. 2016. “Architecting for Continuous Delivery | Thoughtworks”. January 11, 2016 <https://www.thoughtworks.com/insights/blog/architecting-continuous-delivery> [accessed 18 December 2021].

Nath, Mahendra Prasad, Mahendra Nath, Jayashree Muralikrishnan, Kuzhanthaiyan Sundarrajan and Madhu Varadarajanna. 2018. “Continuous Integration, Delivery, and Deployment: A Revolutionary Approach in Soware Development”. *International Journal of Research and Scientific Innovation (IJRSI)* | V. <www.rsisinternational.org> [accessed 3 November 2021].

Null, Christopher. 2021. “10 Companies Killing It at DevOps”. May 19, 2021 <https://techbeacon.com/app-dev-testing/10-companies-killing-it-devops> [accessed 5 February 2022].

O’Brien, Stephen. 2022. “Simple, Zero-Downtime Deploys with Nginx and Docker-Compose | Tines”. January 5, 2022 <https://www.tines.com/blog/simple-zero-downtime-deploys-with-nginx-and-docker-compose> [accessed 22 April 2022].

Offermann, Philipp, Olga Levina, Marten Schönherr and Udo Bub. 2009. “Outline of a Design Science Research Process”. *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST ’09*. <https://doi.org/10.1145/1555619.1555629> [accessed 15 January 2022].

Osterman, Erik. 2020. “Jenkins Pros & Cons (2020) | Cloud Posse”. January 27, 2020 <https://cloudposse.com/devops/jenkins-pros-cons-2020/> [accessed 5 January 2022].

Osterwalder, Alexander, Yves Pigneur, Gregory Bernarda and Alan Smith. 2014. *Value Proposition Design: How to Create Products and Services Customers Want*. Vol. 2. John Wiley & Sons.

Paulk, Mark C. 2001. “Extreme Programming from a CMM Perspective”. *IEEE Software* 18: 19–26. <https://doi.org/10.1109/52.965798> [accessed 18 December 2021].

“Pricing and Plan Information - CircleCI”. n.d. accessed January 5, 2022 <https://circleci.com/pricing/#comparison-table> [accessed 5 January 2022].

“R: What Is R?” n.d. accessed June 18, 2022 <https://www.r-project.org/about.html> [accessed 18 June 2022].

“RStudio | Open Source & Professional Software for Data Science Teams - RStudio”. n.d. accessed June 18, 2022 <https://www.rstudio.com/> [accessed 18 June 2022].

Rudrabhatla, Chaitanya K. 2020. “Comparison of Zero Downtime Based Deployment Techniques in Public Cloud Infrastructure”. *Proceedings of the 4th International Conference on IoT in Social, Mobile, Analytics and Cloud, ISMAC 2020*, October, 1082–1086. <https://doi.org/10.1109/I-SMAC49090.2020.9243605> [accessed 16 December 2021].

“Self-Managed Feature Comparison | GitLab”. n.d. accessed January 6, 2022 <https://about.gitlab.com/pricing/self-managed/feature-comparison/> [accessed 6 January 2022].

Shahin, Mojtaba, Muhammad Ali Babar and Liming Zhu. 2016. “The Intersection of Continuous Deployment and Architecting Process: Practitioners’ Perspectives”. <https://doi.org/10.1145/2961111.2962587> [accessed 18 December 2021].

Shahin, Mojtaba, Muhammad Ali Babar, Mansooreh Zahedi and Liming Zhu. 2017. “Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges”. *International Symposium on Empirical Software Engineering and Measurement* 2017-November: 111–120. <https://doi.org/10.1109/ESEM.2017.18> [accessed 24 December 2021].

Shahin, Mojtaba, Mansooreh Zahedi, Muhammad Ali Babar and Liming Zhu. 2018. “An Empirical Study of Architecting for Continuous Delivery and Deployment”, September. <https://doi.org/10.1007/s10664-018-9651-4> [accessed 20 November 2021].

"TeamCity: The Hassle-Free CI and CD Server by JetBrains". n.d. accessed January 5, 2022 <https://www.jetbrains.com/teamcity/> [accessed 5 January 2022].

"Traefik Concepts Documentation - Traefik". n.d. accessed May 24, 2022 <https://doc.traefik.io/traefik/getting-started/concepts/> [accessed 24 May 2022].

"Traefik Proxy Documentation - Traefik". n.d. accessed April 25, 2022 <https://doc.traefik.io/traefik/> [accessed 25 April 2022].

Virmani, Manish. 2015. "Understanding DevOps & Bridging the Gap from Continuous Integration to Continuous Delivery". *5th International Conference on Innovative Computing Technology, INTECH 2015*, July, 78–82. <https://doi.org/10.1109/INTECH.2015.7173368> [accessed 3 November 2021].

Watada, Junzo, Arunava Roy, Ruturaj Kadikar, Hoang Pham and Bing Xu. 2019. "Emerging Trends, Techniques and Open Issues of Containerization: A Review". *IEEE Access* 7: 152443–152472. <https://doi.org/10.1109/ACCESS.2019.2945930> [accessed 3 February 2022].

Wohlin, Claes and Per Runeson. 2021. "Guiding the Selection of Research Methodology in Industry–Academia Collaboration in Software Engineering". *Information and Software Technology* 140: 106678. <https://doi.org/10.1016/J.INFSOF.2021.106678> [accessed 13 January 2022].

Yang, Bo, Anca Saile, Siddharth Jain, Angel E. Tomala-Reyes, Manu Singh and Anirudh Ramnath. 2018. "Service Discovery Based Blue-Green Deployment Technique in Cloud Native Environments". *Proceedings - 2018 IEEE International Conference on Services Computing, SCC 2018 - Part of the 2018 IEEE World Congress on Services*, September, 185–192. <https://doi.org/10.1109/SCC.2018.00031> [accessed 16 December 2021].

Zhu, Liming, Len Bass and George Champlin-Scharff. 2016. "DevOps and Its Practices". *IEEE Software* 33: 32–34. <https://doi.org/10.1109/MS.2016.81> [accessed 2 November 2021].

Appendix A – Kotlin project configuration code generated by TeamCity

```
import jetbrains.buildServer.configs.kotlin.v2019_2.*
import jetbrains.buildServer.configs.kotlin.v2019_2.buildFeatures.notifications
import jetbrains.buildServer.configs.kotlin.v2019_2.buildFeatures.replaceContent
import jetbrains.buildServer.configs.kotlin.v2019_2.buildSteps.ant
import jetbrains.buildServer.configs.kotlin.v2019_2.buildSteps.dockerCommand
import jetbrains.buildServer.configs.kotlin.v2019_2.buildSteps.powerShell
import jetbrains.buildServer.configs.kotlin.v2019_2.triggers.vcs

version = "2021.2"

project {

    buildType(DeploymentPipeline)
    buildType(IntegrationPhase)
}

object DeploymentPipeline : BuildType({
    name = "Deployment Pipeline"

    publishArtifacts = PublishMode.SUCCESSFUL

    params {
        param("dockerimagename", "epharmastore")
        param("dockerservicetostart", "epharmastore-app@docker")
        param("dockerprofile", "store-prod")
        param("servicetostart", "epharmastore-app")
    }

    steps {
        dockerCommand {
            name = "Docker Image Creation"
            commandType = build {
                source = file {
                    path = "docker-config/Dockerfile_ProdEnvironment"
                }
                contextDir = "."
                namesAndTags = """
                    %dockerimagename%:%build.counter%
                    %dockerimagename%:latest
                """.trimIndent()
                commandArgs = "--pull"
            }
        }
```

```
44          powerShell {
45              name = "Start production containers"
46              formatStderrAsError = true
47              workingDir = "docker-config"
48              scriptMode = file {
49                  path = "docker-config/reload-test-environment.ps1"
50              }
51              param("jetbrains_powershell_scriptArguments", """
52                  "%servicetostart%"
53                  "%dockerservicetostart%"
54                  "%dockerprofile%"
55              """.trimIndent())
56          }
57      }
58
59      features {
60          notifications {
61              notifierSettings = emailNotifier {
62                  email = "henrique.ribeiro@gsiportugal.com"
63              }
64              buildFailed = true
65              buildFinishedSuccessfully = true
66          }
67      }
68
69      dependencies {
70          artifacts(IntegrationPhase) {
71              buildRule = lastSuccessful()
72              artifactRules = """
73                  build\web => build\web
74                  docker-config => docker-config
75              """.trimIndent()
76          }
77      }
78  })
79
80  object IntegrationPhase : BuildType({
81      name = "Integration Phase"
82
83      artifactRules = """
84          **/dist/*.war
85          docker-config => docker-config
86          build\web => build\web
87      """.trimIndent()
88      publishArtifacts = PublishMode.SUCCESSFUL
89
90      params {
91          param("dockerimagename", "epharmastore-test")
92          param("tomcatlocation", """C:\Program Files\Apache Software
    Foundation\Tomcat 9.0""")
93          param("dockerservicetostart", "epharmastore-test-
    app@docker")
94          param("dockerprofile", "store-test")
95          param("servicetostart", "epharmastore-test-app")
96      }
97
98      vcs {
99          root(DslContext.settingsRoot)
100     }
101
```

```
102     steps {
103         ant {
104             name = "Build"
105             mode = antFile {
106             }
107             targets = "build-no-javadoc"
108             antArguments = """do-dist "-
    Dj2ee.server.home=%tomcatlocation%""""
109         }
110         ant {
111             name = "Test"
112             mode = antFile {
113             }
114             targets = "test"
115             antArguments = """do-dist "-
    Dj2ee.server.home=%tomcatlocation%""""
116         }
117         dockerCommand {
118             name = "Docker Image Creation"
119             commandType = build {
120                 source = file {
121                     path = "docker-
    config/Dockerfile_TestEnvironment"
122                 }
123                 contextDir = "."
124                 namesAndTags = """
125                     %dockerimagename%:%build.counter%
126                     %dockerimagename%:latest
127                 """.trimIndent()
128                 commandArgs = "--pull"
129             }
130         }
131         powerShell {
132             name = "Start test containers"
133             formatStderrAsError = true
134             workingDir = "docker-config"
135             scriptMode = file {
136                 path = "docker-config/reload-test-environment.ps1"
137             }
138             param("jetbrains_powershell_scriptArguments", """
139                 "%servicetostart%"
140                 "%dockerservicetostart%"
141                 "%dockerprofile%"
142             """.trimIndent())
143         }
144     }
145
146     triggers {
147         vcs {
148         }
149     }
150
151     features {
152         notifications {
153             notifierSettings = emailNotifier {
154                 email = "henrique.ribeiro@gsiportugal.com"
155             }
156             buildFailed = true
157             buildFinishedSuccessfully = true
158             buildProbablyHanging = true
```

```
159         }
160         replaceContent {
161             fileRules = "**/configuration/configuration.properties"
162             pattern = "hostname = .*"
163             replacement = "hostname = http://localhost/test/store/"
164         }
165     }
166 })
```

Appendix B – Questionnaire

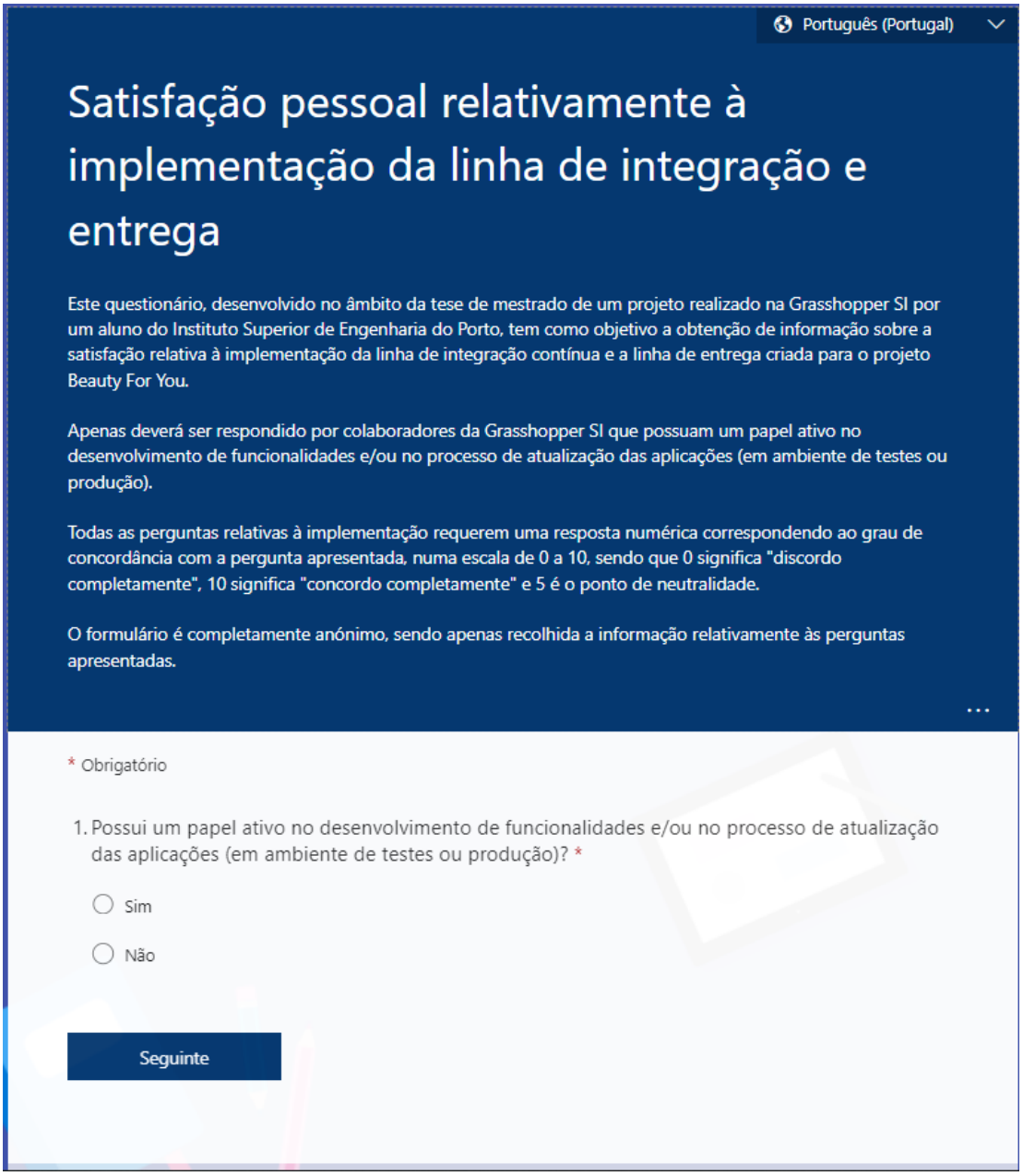

Figure 58 – First page of questionnaire, with instructions and verification about inquired role in the project

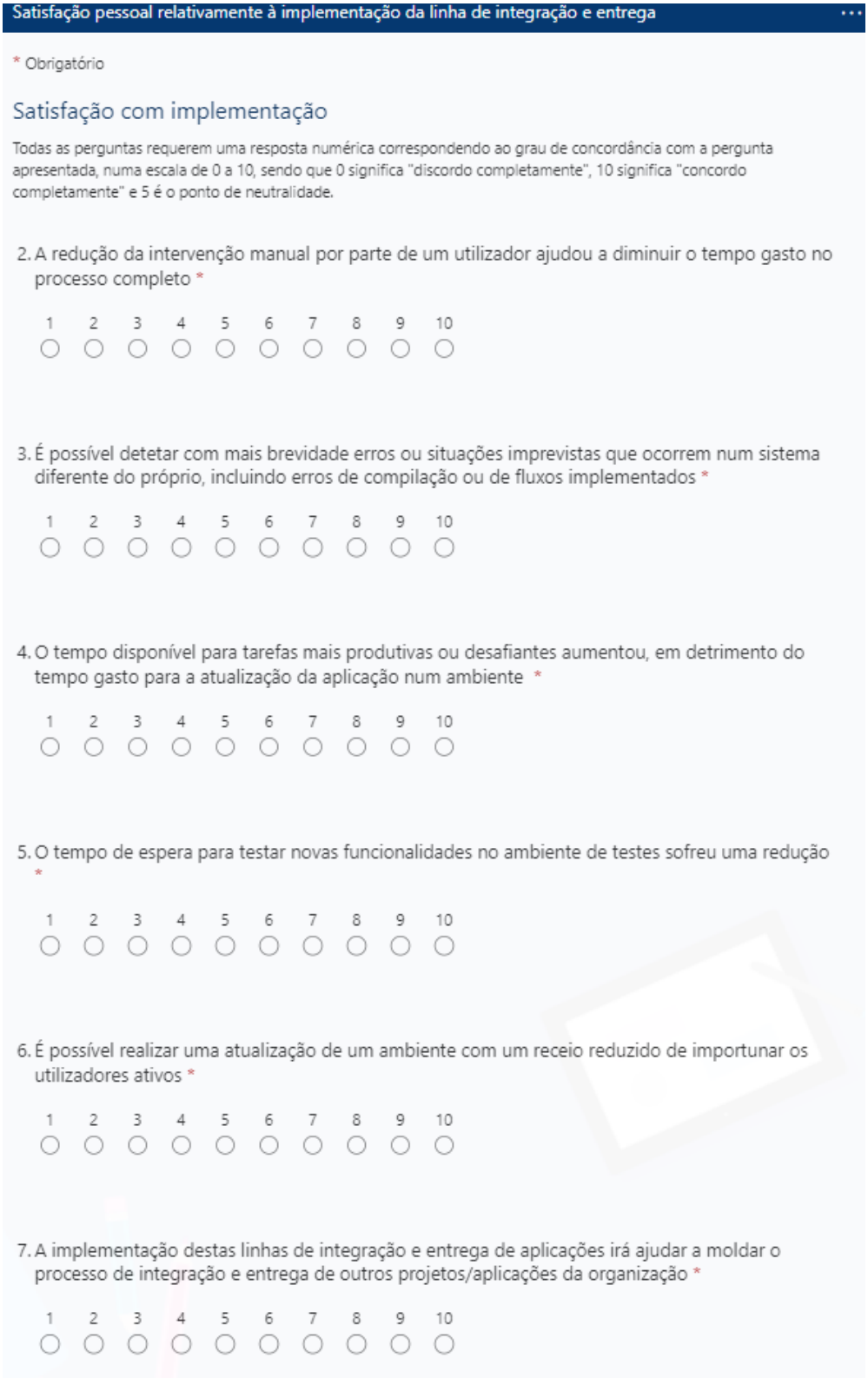

Figure 59 – Questionnaire main structure, with the affirmations and the scale to answer