NOVA IMS Information Management School

MGI

Master Degree Program in
Information Management

Highly Efficient Software Development using DevOps and Microservices:

A Comprehensive Framework

David Alexandre do Lago Barbosa

Master Thesis

presented as partial requirement for obtaining the Master Degree Program in Information Management

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

Highly Efficient Software Development using DevOps and Microservices:

A Comprehensive Framework

by

David Alexandre do Lago Barbosa

Master Thesis presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Information Systems and Technologies Management.

Supervised by

Prof. Vítor Santos, PhD, NOVA Information Management School

July, 2024

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledge the Rules of Conduct and Code of Honor from the NOVA Information Management School.

David Barbosa

Seixal, Portugal, 12th July, 2024

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Vítor Santos for the support, patience and guidance provided throughout this thesis. It was a pleasure to work with such an enthusiastic person. The countless meetings, his constructive feedback and immense knowledge were a major help in increasing the quality of this paper.

I would also like to thank all four experts, Mr. Rodrigo Albino, Mr. Felipe Navarro Ferri, Mr. António Silva and Mr. João Sousa, for their valuable time and for the opportunity of talking to them about such a fascinating topic. Their expertise and knowledge allowed me to evaluate and further improve the proposed framework, while increasing my interest in the subject.

I want to thank my parents, for their encouragement and support. I really appreciate their love and all the valuable lessons they still give me to this day. I would also like to thank my sister, for inspiring me and for believing in me as much as I believe in her.
Finally, I want to show my gratitude towards my friends and everyone else who supported me during this very important stage of my life. It has been a blessing to have you in my life.

ABSTRACT

With the growth in popularity in DevOps culture amongst companies, and the equivalent growth in Microservices architecture development, which are both known to increase productivity and efficiency in software development, an increasingly number of organisations are aiming to incorporate them mutually. Implementing DevOps culture and good practices can be a challenging task, but increasingly important as software applications get more robust, complex and where performance is considered an obligation by the end-users. By following the Design Science Research methodology, this paper proposes an iterative framework, that closely follows the recommended DevOps practices, validated with the assistance of expert interviews, for implementing DevOps practices into Microservices architecture software development, while also offering a series of tools that serve as a base guideline for anyone following this framework, in the form of a theoretical use-case. Therefore, this thesis provides organisations a guideline for adapting DevOps and offers organisations already using this methodology a framework to potentially enhance their established practices.

KEYWORDS

DevOps; Microservices; Information Systems; Software; Efficiency; Agile; Team collaboration

Sustainable Development Goals (SGD):

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
AWS	Amazon Web Services
CI	Continuous Integration
CD	Continuous Delivery
DDD	Domain-Driver Design
DevOps	Development Operations
DSR	Design Science Research
IaC	Infrastructure as a Code
IDE	Integrated Development Environment
IS	Information Systems
MS	Microservices
PaaS	Platform as a Service
REST	Representational State Transfer
RPC	Remote Procedure Call
Https	Hypertext Transfer Protocol Secure
TLS	Transport Layer Security
IDE	Integrated Development Environment

1. INTRODUCTION

1.1. MOTIVATION AND JUSTIFICATION

DevOps offers a great set of benefits for an organization and can be briefly described as a set of cultural philosophies, practices and tools which aim to improve an organization's ability to deliver better high-quality applications and services in a reduced period. (SPM Global Technologies, 2023)

To do this, DevOps breaks down the barriers between development and IS operations. It also offers a set of tools that improves collaboration, introduces and increments automation, allows continuous integration and continuous delivery and offers useful feedback and insights. (Amazon, 2023)

Microservices, on the other hand, is a software development approach that breaks an application into smaller services that can be independently deployed, managed, and assigned to independent teams. (Jacobs et al., 2022)

It incentivises collaboration and speeds up development by micro dividing the project at hand. It also removes technology stack limitation because each component can be developed using different technology stacks. Thus, decreasing the need for retraining amongst the organization. These two topics, albeit focusing on different aspects of software development, work extremely well together, and ultimately, their end goal and vision are shared. BMC, multinational consulting company, even goes as far as saying "Microservices architecture is tailor-made for DevOps" (Wickramasinghe, 2021).

Like all topics in software development, there isn't a general rule. When it comes to the architecture used, the methodologies employed, the team composition, these are all topics open to debate and each have their own merits and demerits, as these depend on the problem at hand. For instance, building a software for marketing company differs from building a software for a renewable energy company, because a project that requires lots of business logic will have different needs, and different architecture requirements than a web app for a local business store.

Having this in mind, microservices architecture and DevOps culture are software and information systems development approaches that deliver good results for a major part of business requirements where fast and efficient implementation is prioritised. (Wayner, 2023)

1.2. RESEARCH GAP

Although the benefits of DevOps and microservices are clear, there is still a lot of confusion regarding in what way they can be both used together and how to take advantage of them and the best overall way to make good use of these two pillars of information systems solutions. The issue can be translated into the following research questions:

RQ1 - Can a set of guidelines be created to improve productivity in developing Microservices architectures using DevOps practices?

RQ2 - Would it be possible to instantiate these set of guidelines with the DevOps tools suitable for Microservices development?

RQ3 - Can this set of guidelines be understood as a way of applying the best practices of a DevOps approach to the development of microservices?

1.3. Research Objectives

In order to help answer the enunciated research questions, the goal of this research is to build a framework that delivers the best practices and approach to building a software application using DevOps philosophy and Microservices architecture.

In order to achieve the main goal, the following intermediate objectives were defined:

- Make a comprehensive study on Microservices.
- Make a comprehensive study on DevOps.
- Create a framework for the use of DevOps in Microservices based applications development.
- Create a use case.
- Validate the framework.

Thus, this thesis aims to, by experimentation and hands-on experience, elaborate a framework that illustrates the most effective way to combine DevOps and microservices.

1.4. Importance and Relevance

By exploring the different approaches to DevOps combined with Microservices, it is expected to provide information systems organisations with a comprehensive framework on the most productive way to combine this widely used culture and architecture.

DevOps has plenty of benefits, including faster and better product delivery, a reduction in error quantity, increase in resource efficiency, promoting innovation and increasing collaboration.

Being mainly a cultural transformation, focused on automation and efficiency, it is definitely not easy to implement, as it requires an effort from the entire organisation, which means a big cultural shift is needed and a big technical challenge to implement it and requires a big commitment from the entire organisation (Debois et al., 2011; Tanzil et al., 2023).

On the other hand, Microservices is an emerging software architecture. In fact, a survey conducted by Camunda concluded that 63% of enterprises surveyed are using Microservices architecture in their projects. (Camunda, 2018)

Another survey, conducted by Gajda on 650 worldwide developers found out that only roughly 8% of them have been using Microservices for over 5 years. (Gajda et al., 2020) This shows that although it is an emerging software architecture, there is still a lot of exploration and improvement to be done.

In recent software development, building solutions with a microservices architecture has become a new trend, as microservices benefits from the monitoring associated with DevOps (Giamattei et al., 2024; Waseem et al., 2020). But it is still unknown as to what extent software development efficiency

really profits from the combination of DevOps and Microservices, that is the main motivation of this study.

1.5. THESIS STRUCTURE

This thesis is organised as follows:
In the first part

Then, the state-of-the-art literature, regarding DevOps and Microservices, is summarised, including key concepts, known implementation methods, pros and cons, as well as related.

In the following phase, a framework will be proposed and a use case will be discussed, explaining the assumptions beforehand. The framework will be evaluated by experts in the field of study, through semi-structured interviews. Following this evaluation, the proposed framework and use case will be adjusted based on the advice gathered.

Finally, the conclusions on the developed work will be discussed. The analysis of the limitations of this thesis as well as propositions for future works will also be part of the conclusions chapter.

2. METHODOLOGY

This thesis will follow the Design Science Research (DSR) methodology. DSR is a problem-solving paradigm that strives to enhance scientific knowledge and solutions to real-world problems by creating innovative artifacts. This methodology is frequently used in IS research, as it facilitates innovation in organisations and has generated a lot of interest in the past 20 years (Brocke et al., 2020). The goal of this thesis is to generate a framework and evaluate it Therefore, the DSR methodology will be used achieve that goal.

2.1. DESIGN SCIENCE RESEARCH

Figure 2.1 depicts an overview of DSR framework, starting with the environment, on the left. This pillar describes the problem space in the area of interest of the subject. The environment is composed by people, organisations, and existing and/or planned technologies.

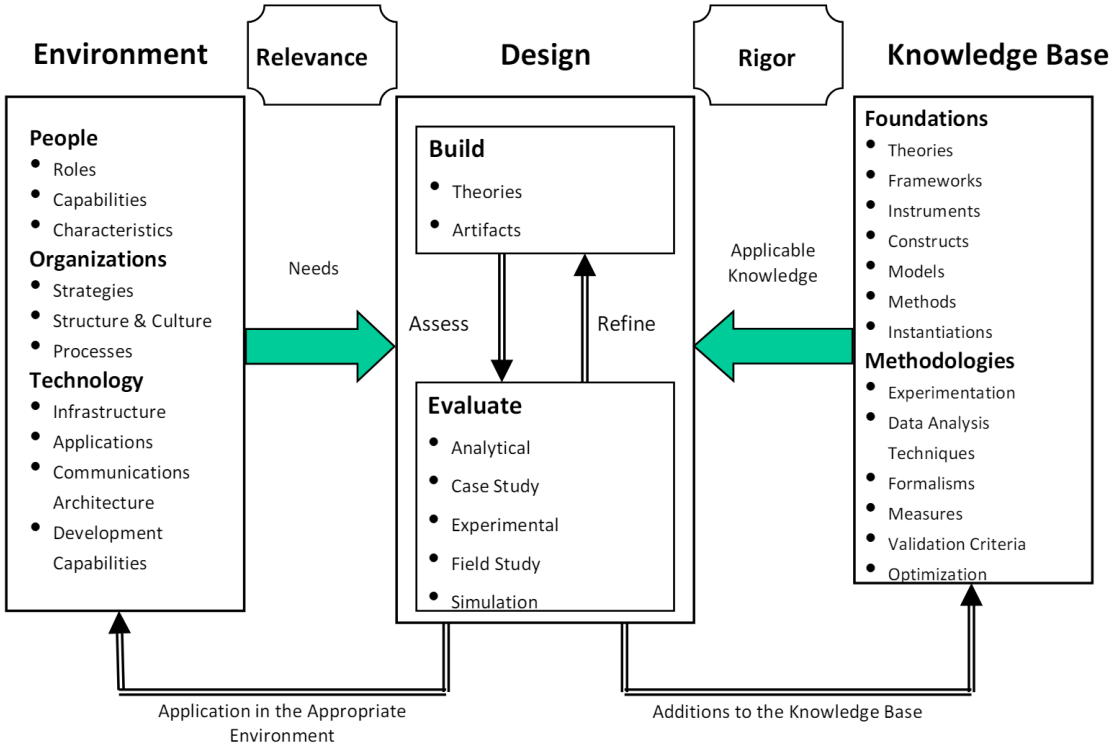

Figure 2.1 - DSR Methodology (Brocke et al., 2020, p. 4)

The needs, which are generated by the goals, tasks, problems and opportunities, are assessed against the current existing work, culture and context within organisations. These needs are placed against the existing technologies and applications, architectures, and elaboration capabilities. These notions make up the research problem for the researcher. The importance of the “research problem” to

stakeholders highlights the research problem's relevance. The knowledge base section facilitates instruments that help achieve DSR, as it is composed by "foundations" and "methodologies". Foundational theories, frameworks, instruments, constructs, models, methods and instantiations are accomplished from prior research and results from the reference disciplines. Methodologies provide guidelines for the evaluation phase of DSR, where the case study, experimentation or field study of the theory or artifact built is executed. Rigor is accomplished by appropriately applying existing foundations and methodologies.

Ultimately, DSR seeks to create and build an innovative solution to a problem, using parts of existing solutions and combining present design knowledge.

The DSR process used in this thesis, is an iterative process, as shown in figure 2.2. It is composed of six steps: problem identification and motivation, the definition of the objectives for a solution, design and development, demonstration, evaluation and communication. It is also instantiated on four possible entry points: problem-centred initiation, objective-centred initiation, design and development-centred initiation, and client/context initiation.

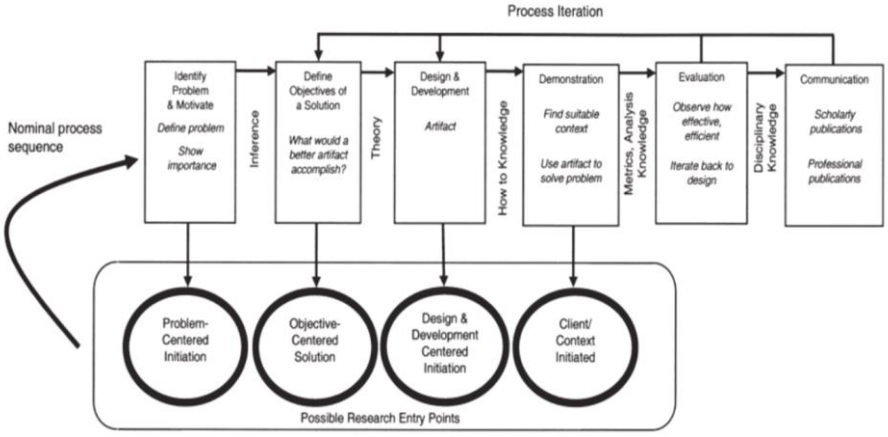

Figure 2.2 - DSR Process (Brocke et al., 2020, p. 6)

1. **Identify Problem & Motivate -** In the problem identification and motivations phase, the research problem is indicated and justified. The justification is important as it accomplishes two things: it motivates the researcher and interested parties to pursue the solution and it helps the audience appreciate the researcher's understanding of the subject. In this step, the knowledge of the state of the problem and the importance of its solution are mandatory resources.
2. **Define objectives of a Solution -** The following step is where, using the information, data or previous works gathered in the first step, the objectives and goals for the solution are defined. The objectives can be quantitative or qualitative. Quantitative objectives aim to provide a better solution than existing ones, improving on something

that already exists. Qualitative objectives have the goal of providing a new artifact that supports solutions to problems not currently addressed.

3. **Design and Development -** The design and development activity is where the artifact is created. The artifact is a product of the analysis of the previous research contribution and new acquired knowledge. This stage evolves determining the artifact's desired functionality as well as its architecture and then creating the actual artifact.
4. **Demonstration -** This activity displays the use of the artifact to solve one or more instances of the problem at hand. This can be accomplished by showing its use in experiments, simulations, case studies, proof or another form of appropriate activity.
5. **Evaluation -** The next activity is evaluation stage. In this stage, the performance of the artifact is measured. It is evaluated based on how well it provides a solution to the problem that motivates the researcher and stakeholders. In the evaluation, the solution is evaluated by experts or by observing the results in a real-world environment. At the end of this evaluation, the activity researchers can opt to iterate back to step three and improve the artifact or proceed to the communication phase and suggest future improvements for future projects.
6. **Communication -** The final stage involves the communication of all characteristics of the problem and designed artifacts to the stakeholders. The results of the research and artifacts are published and added to the existing knowledge on the subject. There are different forms of communication depending on the goals of the research and the audience, but the communication of the results allows for other researchers to analyse the results and further increment the project with new or redefined research goals.

2.2. DSR IMPLEMENTATION

The DSR implementation for this thesis will be the Hevner proposed approach, explained in chapter 2.1 and it will be performed in two cycles: the rigor cycle and the relevance cycle. The DSR process will be followed closely.

Starting with the relevance cycle, existing literature will be studied in chapter 3.0 and there will be an assessment of the current existing studies and knowledge on DevOps and Microservices, its origins and evolution, how they interact with each other and how they influence today's world and its applications.

These studies will allow to identify a research gap and the opportunities in the field of study. Helping generate the needs, and the research problem, thus, concluding the relevance cycle.

In the rigor cycle, an artifact will be created. The knowledge base that was acquired so far, will serve as a foundation for the artifact, through the study of theoretical papers, other frameworks and even other artifacts. A framework will be developed that will attempt to answer the initial research questions.

Once the artifact is created, a rigorous evaluation phase occurs, where by experimentation, the artifact's performance will be assessed by experts in the field. Due to time constraints for this thesis, a real-life experimentation will not be possible. Instead, feedback gathered from the experts will be considered and due to the cyclic nature of the DSR process, the artifact will be improved upon or proposed as future work for future projects.

This will conclude the DSR methodology approach for this thesis, as case study research will be the most efficient way of answering the initial research questions. This methodology is very well suited to Information Systems research, and it will allow to do an exploration of an artifact by experts in the field of study – software development companies.

3. LITERATURE REVIEW

The literature review chapter provides a comprehensive study on existing knowledge about the topic at hand. Therefore, it is divided into three sections. The first section addresses the fundamentals of DevOps, its motivations, what are its benefits and drawbacks, and what is known to be crucial to implement it in organisations. The second part tackles Microservices architecture, its general concepts, its comparison to Monolithic architectures and its advantages and disadvantages will be looked at. Finally, in the third section an overview of how DevOps and Microservices fit and work together will be discussed.

3.1. MICROSERVICES

3.1.1. Overview

Currently, there is no coherence on a definition for Microservices (Baumgartner, 2022, p. 31). It can be briefly described as an approach to developing a single application using a suite of interdependent and individually deployable services (Di Francesco et al., 2019).

Microservices are becoming a new standard pattern in modern software development. Some very well-known companies like Netflix, Amazon, Atlassian, Spotify and Uber are all using Microservices. Netflix, that originally began as a DVD-rental service in 1997, realised, eight years later, the power of cloud services and in 2007 started their streaming service. They began with a monolithic architecture but due to high demands and scalability issues, they migrated to microservices. (Varshneya, 2021)

This change wasn't easy by any means. But the effort paid off, as we can personally verify with their flawless and fluid platform.

3.1.2. Concepts

Containers

The use of containers is generally used to run microservices independently. Containers are standalone and lightweight, and provide a great way to compile, run and setup the needed infrastructure to run each microservice (Baumgartner, 2022, p. 19; Robitzsch et al., 2023).

They help deal with scalability issues as they allow the resources for each container to be managed independently (Waseem et al., 2020, p. 11).

Version Control

In software development, a version control system is essential. The most used version control system is GIT (Jia et al., 2023). Its adaptability, speed, compatibility and cost free nature makes it the de facto favourite by software developers. It allows developers to always have a working copy of their work, manage versions of their progress and implement new features, while being able to test them before converging them to the main repository.

Git offers functions in the form of commands, and while Git is said to be relatively easy to start using, it is hard to master. Some popular Git functions are (Git, 2024; Jia et al., 2023):

- Git init – creates a new git repository
- Git clone – clones an existing repository to work with
- Git status – returns the current status of your git repository and its files
- Git add – adds files from the working directory to the staging index
- Git commit – modifications in local repository with a message associated with the modification id
- Git push – pushes the modifications in local repository to the remote repository
- Git branch – adds/lists/deletes branches. A branch is a version of the repository, to be worked with, modified and after the feature is complete, integrated back into the main repository (branch).
- Git merge – integrates the working branch with another branch

In figure 3.1, an example workflow of a software project using Git is shown.

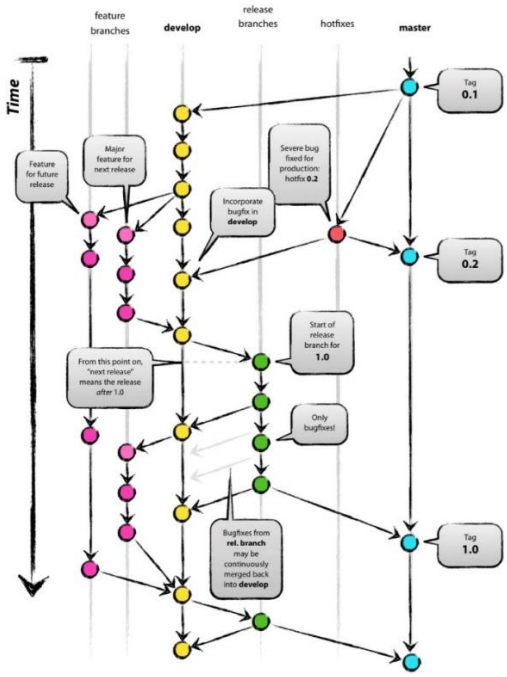

Figure 3.1 - Branching diagram describing Git workflow (Cortés Ríos et al., 2022)

Deployment and Infrastructure

One of the major aspects of Software development is releasing it to the end-users. There are many ways to achieve this; some companies still host their products on premises, but nowadays there is a variety of cloud service offers. Cloud service providers such as Microsoft Azure, Amazon Web Services (AWS) and Google Cloud offer infrastructure-as-a-service or platform-as-a-service (PaaS) solutions that allow organisations to host their products easily and securely (Pedchenko et al., 2022).

In addition to hosting and infrastructure capabilities, it is important for the deployment structure to be well organised. Typically, there are development, test, staging and production environments (Bakshi, 2017). The development environment is where new features are implemented and first released, while the production environment is where the final product is hosted and where the end-users interact with the product. Whereas the staging environment emulates the production environment as closely as possible and allows to conduct the final tests.

3.1.3. Comparison with Monolithic and Service Oriented Architectures

A software architecture represents the way components are organised within the project(s). These components interact with each other to form a solution that satisfies the initial requirements (Sharma et al., 2015). There are several software architectures, the most common being: Monolithic, Service Oriented Architectures.

In a monolithic architecture, there is a single code solution and a single deployable application. Everything is tightly coupled together, and it is simple to implement. This design is generally more suitable for small applications and is prone to produce challenges when or if the application evolves into a more complex solution (Richardson, 2024).

Service Oriented Architectures involved from monolithic architectures as an attempt to mitigate some of the challenges in Monolithic implementation, such as maintainability and difficulty to understand (Munaf et al., 2019, p. 1).

However, compared to microservices they still suffer from challenges that monolithic architecture does, including having to deploy the whole solution at once and the management of the solution being centralised (Amazon, 2024).

3.1.4. Advantages and Disadvantages

In the following section, we will analyse the advantages and disadvantages of using Microservices as a software architecture.

On the positive side:

Scalability stands out as a key advantage of microservices, as it is very easy for organisations to adapt to increasing workload by scaling up or down resources as needed (Baumgartner, 2022; Waseem et al., 2020).

Flexibility is another main aspect of microservices, since it provides freedom for developers with different technological backgrounds to work in the team. Notably, each microservice can be implemented in a different programming language (Baumgartner, 2022).

Due to the interdependence of microservices, if one of them fails, that failure will not affect the whole system, making microservices usage increase the resilience of the final product.

Productivity is increased with Microservices, on account of them focusing on dividing a complex solution into many small solutions, allocating each team onto one of these smaller solutions. This

also allows teams to specialize further and enables a sense of ownership within them, leading to faster development times. (Atlassian, 2024a)

Overtime, maintainability is improved with microservices, and although there is an initial overhead, adding new features becomes easier without affecting the existing solution (Munaf et al., 2019).

With small and more specialized teams, resource allocation can be optimized, causing the development of new features to be more time and resource efficient. Server and cloud resources can also be used more efficiently since they can be allocated to services with greater needs. This proves cost-efficiency to be one of the advantages in implementing microservices.

There are, however, some disadvantages to the use of microservices:

Due to its distributed nature, it has an increased complexity and developers must put in extra effort to ensure seamless communication, testing and deployment capabilities across all services. Managing those smaller subprojects also has its shares of responsibilities and additional management work (Aderaldo et al., 2017; Baumgartner, 2022).

There are also more difficulties debugging, testing is much more challenging because when an error occurs, it is more complex to track down which service is causing the issue. A lot of work is put into tracking the log files to discover the service in question.

Security is a big challenge in microservices. The entire nature of having independent services deployed independently and communicating, means a bigger portion of the solution is exposed to the network (Baumgartner, 2022; Munaf et al., 2019).

Another notable disadvantage is performance. With so many independent services interacting with each other, it is hard to keep track of performance. A set of faulty services can cause a bottleneck on the entire solution (Munaf et al., 2019).

3.1.5. Microservices Implementation

Domain-Driver Design (DDD)

Microservices matches well with the decentralisation of logic and data (Bakshi, 2017). The DDD process includes dividing a complex solution into separate sub-domains, each of them composing a microservice and these interacting amongst themselves to produce the end product. This simplifies the solution and provides teams with high knowledge about their domain (Giallorenzo et al., 2023).

Design Patterns

There are many design patterns (API Gateway, Publish/Subscribe, Proxy, Circuit breaker), but Application Programming Interface (API) Gateway seems to be the most common (Di Francesco et al., 2019; Munaf et al., 2019). As shown in figure 3.2, an API gateway is a central gateway that interacts with all services. This gateway will centralise all requests and handle authorisation, authentication and security filters (Aderaldo et al., 2017; Baumgartner, 2022; Di Francesco et al., 2019).

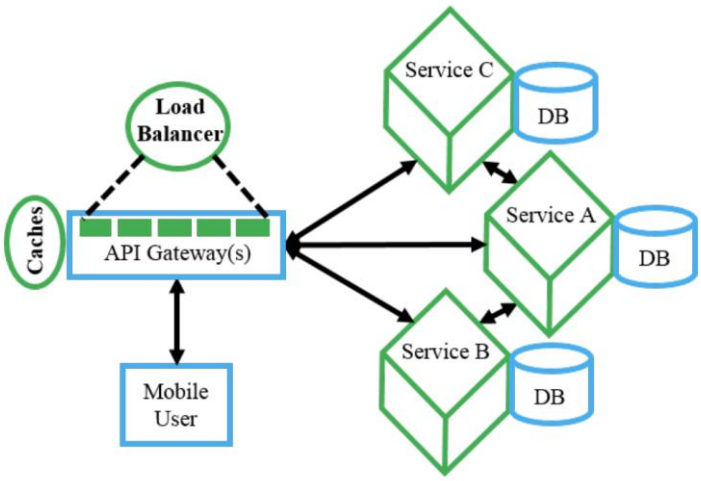

Figure 3.2 - API Gateway Design Pattern (Munaf et al., 2019, p. 86)

Communication between microservices can be done using many models, but the most common communication model for the API is Representational State Transfer (REST) and Remote Procedure Call (RPC). With each microservice being able to have its own communication model (Baumgartner, 2022; Di Francesco et al., 2019; Waseem et al., 2020).

To make sure all the microservices are only being accessed by the API Gateway, an authentication method has to be implemented. Each communication should be conducted utilising a Transport Layer Security (TLS) using protocol such as the Hypertext Transfer Protocol Secure (Https).

Monitoring and logging

With the challenges associated with a distributed architecture, setting up the logging environment is an important process. This has to be a robust logging system, which itself can be centralised or distributed, logging and tracking all microservices analytics, performance issues and errors (Baumgartner, 2022, pp. 85–87; Munaf et al., 2019, pp. 86–87).

3.2. DevOps

3.2.1. Overview

DevOps is a relatively recent movement, to better understand its impact we have to go back to the manufacturing revolution. In the 80s, before the manufacturing revolution, orders were up to six weeks, with less than 70% of those orders being shipped on time and getting far less product quality. During the revolution, by adopting lean practices, manufacturing organisations were able to improve productivity, customer satisfaction and product quality (Elrhanimi et al., 2018).

At the time, organisations that did not adapt, eventually lost market share and were surpassed by those that implemented lean practices (Kim et al., 2016).

The situation for developing and delivering technological products is very similar, the cost and time required to develop and deploy technological services has dramatically decreased in the last couple of decades (Brown et al., 2020).

While in the 2000's, with the advances of Agile principles, development time dropped to months or even weeks, deployment was still a grindy effort, and would sometimes result in failures and prejudicial outcomes.

Then two trends emerged. The first one being caused directly by Agile methodologies, which incentivises fast development, small and frequent deployments. The second one being the rise of cloud infrastructure. Operations teams had the ability to setup new environments at a pace never seen before.

With the constant pressure in delivering software at an increased pace, it is important that development and operations teams work together to deliver best results (Debois et al., 2011, pp. 3–4).

As a response to the issues, DevOps arises, sometime between 2007 and 2008, the goal of DevOps was to go beyond Agile, streamline and somewhat automate deployment methods. (Atlassian, 2024c).

3.2.2. Principles

DevOps is based on four principles (Colomo-Palacios et al., 2018; Debois et al., 2011, pp. 7–8; Fitzgerald & Stol, 2017), these being: Culture; Automation; Measurement; Sharing.

There has to be a culture of sharing responsibility and working together towards delivering high quality software to the end-user. Operations should be included in the design and building phases of IT projects, as well as retrospectives and plannings, and developers should participate in meetings with the operations teams.

On the other hand, DevOps focuses on speed, therefore it is imperative that all development and operation steps are automated whenever possible, namely the build, deployment and testing processes.

A constant measurement of the current delivery capabilities or the organisation should be conducted. Setting new goals for improvement and understanding which areas should be improved in order to meet business needs.

By sharing responsibilities, arises the necessity of sharing knowledge. Ranging from knowledge about infrastructure, development needs, and knowledge tools, it is important to share knowledge across all organisation structure.

DevOps evolves around the notions Continuous and Automated tasks (Debois et al., 2011, pp. 19–20). All phases are continuous and repeatable (Fitzgerald & Stol, 2017, pp. 181–185), and it focuses on continuous integration, continuous deployment and automated deployment, and continuous

learning. There are known tools used for each on these processes, which will be briefly described during this chapter. (Fitzgerald & Stol, 2017, p. 182; Grande et al., 2024, pp. 8–9)

The first stage is Continuous Integration (CI), in this stage, developers focus on increasing the frequency and automating the process of integrating and merging their code to shared repository and running automated tests to detect integration errors early in the development cycle. With the implementation of CI, teams induce a strong foundation for collaboration and automation, enabling quicker feedback and reducing faulty products (Aderaldo et al., 2017).

The next phase is Continuous Delivery (CD), occurring after CI, CD goes beyond CI by automating the release process. It consists of automating the deployment of validated code changes to testing, staging or production environments. By continuously delivering changes to these environments, teams can ensure that the final product is always up to date with the latest changes, once again, quickening feedback from testers and end users (Aderaldo et al., 2017).

CI/CD and its tools simplify the release process by eliminating manual steps and reducing deployment errors. They ensure consistent deployments across environments, enabling teams to deliver software updates more reliably and efficiently.

Teams should implement a deployment pipeline for developers to follow, an example is shown in figure 3.3. This process should be well matured, as it enables the two CI and CD stages described previously.

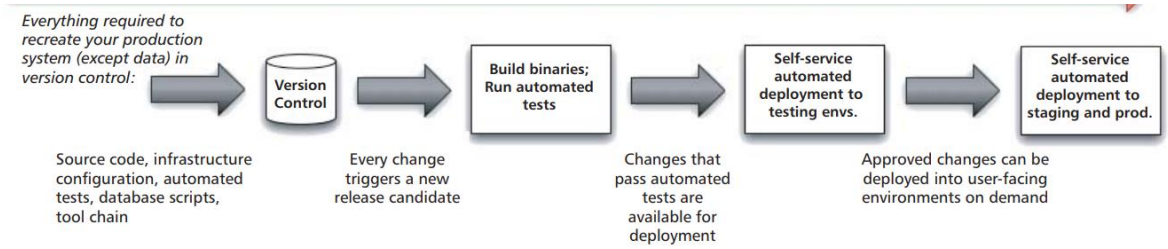

Figure 3.3 - Example of a deployment pipeline (Debois et al., 2011)

Continuous and automated Deployment (CD/AD)

When organisations achieve a high level of technological and automation maturity in the CI/CD pipeline, they are able to achieve continuous deployment. It increments upon the CI and CD established foundations by automating the deployment of every single validated change to production environments. This requires a high level of confidence in the CI/CD pipeline as well as testing practices in the development teams. Thus, contributing to deliver swift feedback to developers. The difference between CI/CD and CD/AD is illustrated in figure 3.4.

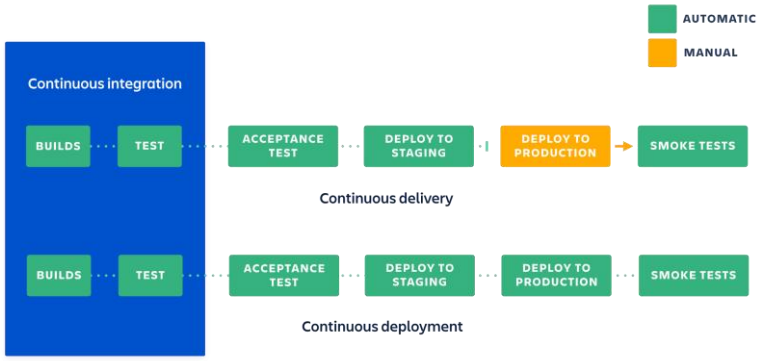

Figure 3.4 - Automated Deployment Process (Atlassian, 2024b)

Continuous Testing and Automated Testing

With continuous and automated testing new errors are detected earlier and are tracked and eliminated more efficiently.

Infrastructure as a Code (IaC)

As said by Grande (Grande et al., 2024, p. 9) *"IaC is a practice that allows the management and configuration of the infrastructure used for a concrete environment."*

It allows for quick and adaptable configurations to each solution, helping detect early errors and increase the quality of releases.

Continuous Learning

We would like to add the importance of continuous learning. DevOps inactivates innovation and experimentation (Kim et al., 2016).

3.2.3. Benefits and Drawbacks

In the following section we will debate the known benefits and drawbacks in adopting DevOps.

DevOps has several well studied benefits (Grande et al., 2024, pp. 12–13), some of which will be explained as follows. The first benefit that deserves to be emphasized is the reduction in release cycles. Due to the automation and collaboration in DevOps teams, this reduction is one of the main benefits of implementing DevOps, with teams claiming that after implementing DevOps practices the release schedule drastically increased (Debois et al., 2011, p. 35; Fitzgerald & Stol, 2017, p. 183; Grande et al., 2024, p. 12).

With the implementation of automated tests, conducted after every release, issues and errors are detected at a much earlier stage. Integration is also automated in a lot of ways, and any steps that

don’t have to be done manually reduce the chances of introducing errors into the solution, therefore, greatly improving reliability (Debois et al., 2011, p. 39; Grande et al., 2024).

Efficiency is really a designation that the DevOps culture aims for, thus efficiency is one of the major advantages of DevOps. Solutions are developed at a more efficient pace, infrastructure is managed in a more resourceful way, and teams perform more efficiently overall (Colomo-Palacios et al., 2018; Debois et al., 2011; Grande et al., 2024). There are many more advantages to using DevOps, but these are considered to be the most important ones.

Despite the immense amount of advantages to implementing DevOps in organisations, there are, however, many challenges that must be overcome. The most impactful of these disadvantages will be analysed as well.

To begin with, general lack of skills and knowledge is considered to be one of the main challenges in implementing DevOps methodologies (Khattak et al., 2023). With its definition still being subjective to interpretation, there is a high learning curve before being able to use all the required tools in implementing DevOps (Grande et al., 2024, p. 13; Tanzil et al., 2023, pp. 11–12). The development team should participate in the operations tasks, but it is still extremely important to have operation team members with high proficiency in managing infrastructure in the team (Grande et al., 2024, p. 14).

Recent studies (Tanzil et al., 2023) show that only 5% of companies interviewed admitted to having formal training in DevOps, 10% of participants were trained in online courses. This is one of the most impactful identified challenges in DevOps, being continuous learning one of the pillars in DevOps.

The potential lack of collaboration and communication is also one of the main challenges, being mentioned by a lot of DevOps teams (Khattak et al., 2023), it is fundamental that development and operations teams collaborate effectively for DevOps to work.

A shift on not only DevOps teams members but the entire organisation is needed to inactivate collaboration between involved parties. It is frequently observed that, generally, development and operation teams have different ways of working and even views on one another. This resistance to change is bound to be counterproductive in efficiently implementing DevOps mentality (Abbas & Singh, 2023; Grande et al., 2024, p. 14; Khattak et al., 2023).

Software architecture

Some DevOps teams practitioners defend that not all software architecture is suitable for DevOps; Microservices are preferred as opposed to Monolithic architectures as it is more suitable to managing complex solutions and cause less conflicts to the continuous development practices (Grande et al., 2024, p. 14).

3.2.4. Applications in Software Development

As shown in Figure 3.5, there are many software and technologies involved in implementing DevOps methodologies (Gokarna, 2021, p. 3; Grande et al., 2024, p. 15).

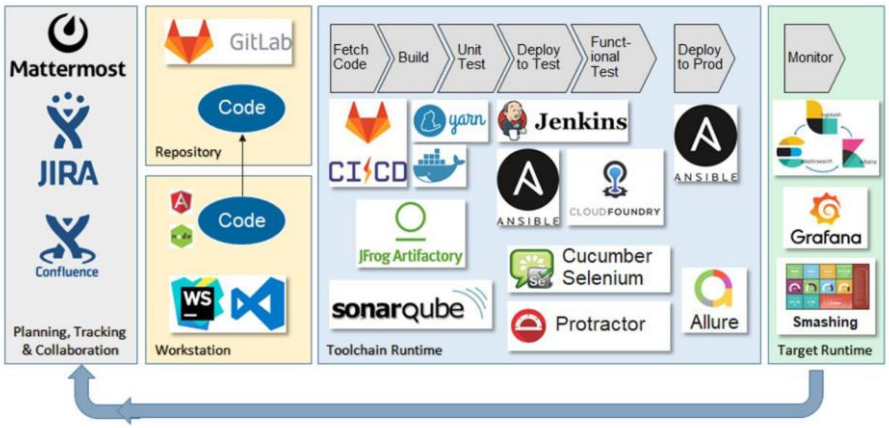

Figure 3.5 - DevOps tools at various stages (Gokarna, 2021, p. 3)

Planning, Tracking and Collaboration

Jira and Github are some of the most popular tools for sprint and task planning (Stack Overflow, 2023).

Confluence, belonging to the same company as Jira, is a popular tool for knowledge sharing that is highly used by organisations and integrates very well with Jira itself.

Regarding actual software development tools, called **integrated development environment** (IDE), there are more powerful options such as Visual Studio, Eclipse, Android Studio, IntelliJ, and more lightweight but versatile options such as Visual Studio Code, Sublime, Notepad++. With the most popular of these being Visual Studio Code, Visual Studio, and IntelliJ.

Git extensively used version control system for code sharing and enabling collaboration between different developers (Braga et al., 2023). It has a file structure to track file modifications. Git/Github, Bitbucket and Gitlab are popular options for version control systems (Beckman et al., 2021).

Toolchain Runtime

Cloud computing has seen a rise in recent years. There have always been virtualization technologies, that allowed users to deploy multiple applications in the same machine. However, these machines are large in size, unreliable and hard to maintain. The difficulties allowed for the emergence of a new process called containerisation. Containerisation allows for the virtualisation at the operating system level. The main difference between containerisation and virtualisation is that virtualisation creates virtual machines, that run its own operation system, whereas containerisation abstract the operating system and package the application in lightweight containers that include everything needed for the

application to run: code, runtime, system tools, system libraries and settings. The emerging technology to implement this containerisation is **Docker** (Morris et al., 2017; Potdar et al., 2020).

This virtualisation and the management of containers creates some overhead when managing infrastructure, therefore there is an increased need for automation of processes.

Jenkins is a continuous integration platform that automates building, testing code changes as it is pushed to a version repository in GIT and deployment. This automation helps find errors early in the development cycle and simplifies integration with the existing code base in the target application. It also participates in the continuous deployment process of DevOps, as it helps the deployment of code changes to the various environments, such as development, staging and production (Morris et al., 2017).

With the increase of containers provided by Docker, it can be hard for companies to manage them individually. That's where **Kubernetes** is applied: it is a complement to Docker and it aids with the container deployment and orchestration (Chowdary et al., 2022).

Grafana, Prometheus and Tableau are popular analytics and monitoring tools for running applications.

3.2.5. Complementarity between DevOps and Microservices

DevOps and Microservices share the same ambitions, such as providing scalable, efficient, and robust software, making them synergistic methodologies. There isn't a best architecture for all, but if one would be recommended to be used alongside DevOps culture, Microservices would very well fit that role (Grande et al., 2024). Microservices is often described as being the natural progression of embracing DevOps, as it promotes agility between teams, and studies confirm that sometimes, companies adopt microservices as an enabler for implementing DevOps culture in their company (Hasselbring, 2018; Knoche & Hasselbring, 2019). Furthermore, microservices are intricately related to the more organisational and continuous delivery aspects of DevOps (Knoche & Hasselbring, 2019, p. 10).

The benefits are clear, they range from efficiency to risk reduction, but all this comes at a price. The natural segregation aspect of microservices makes it more challenging for operations and development teams to work together and successfully deliver them. The number of interrelated components and steps that evolves releasing and application is increased, the overhead is augmented and consistency issues may arise (Sampaio Junior et al., 2017).

This further shows the needed attention regarding the complexity, team coordination and culture alignment in the teams involved. A lot of this overhead is placed at the continuous delivery aspects of DevOps and this highlights the importance of investing in continuous-deployment teams (Knoche & Hasselbring, 2019).

But, by using each of their complementarity benefits and adopting good practices, organisations can make use of the combined capabilities of DevOps and Microservices to induce innovation and produce higher quality products to their users.

4. FRAMEWORK FOR THE USE OF DEVOPS IN MICROSERVICES BASED APPLICATIONS DEVELOPMENT

This chapter is composed of five sections, describing the process that originated the proposed framework for the use of DevOps in Microservices architecture applications. The first, being the assumptions, based on the literature review in chapter 3. The second chapter introduces the framework and describes it in detail. Following this, a simple use case is demonstrated, offering examples of tools for each phase of the proposed framework. In the fourth chapter, the feedback gathered from the interviews with the experts in IT software development, to whom the frame was presented, is discussed. Finally, the revised model considering the feedback gathered from the interviews with the Experts is presented.

4.1. ASSUMPTIONS

Based on what was studied in the literature review, about DevOps culture and MS architecture, it was possible to say that DevOps principles when applied to Information Systems improve collaboration between teams, all the while enhancing processes by automation and enhancing delivery capabilities. This is achieved through fostering a culture where responsibilities are shared between teams, performance and deployment capabilities are measured, there is an incentive to share knowledge and automation is encouraged when applicable, in more routinely tasks. This provides organisations with a greater agility, reliability and efficiency in managing their Information Systems, reduces the amount of occurring errors by removing the need for manual intervention in some tasks during the CI/CD pipeline.

When applied to software development, DevOps empathizes the importance of all the capabilities mentioned above, in the software development lifecycle. Breaking down the segregation between development and operations teams, automating repetitive tasks in deployment, testing and monitoring, and fostering a knowledge sharing environment between teams, it is possible to accelerate software delivery, improving quality and increasing the end user experience.

Microservices, are a software architecture that consists of breaking down applications into smaller, independently deployable services, with each service filling a specific business need. These services are loosely coupled and communicate with each other via APIs.

The interdependent nature of microservices and the breaking down into smaller projects, allows teams to focus on individual services, making it easier to develop, specialise and enabling faster development cycles, while allowing a bigger scalability and allocation of resources. This, however, induces a greater complexity to guarantee efficient communication between all the microservices, and some extra security and performance commitment.

DevOps is based on four principles: culture of sharing responsibilities and working together, automation of tasks when possible, such as deployment and testing processes, measurement of delivery capabilities and knowledge sharing.

As mentioned previously, in chapter 3.2, DevOps is highly focused on continuity, automation and iteration.

All phases of DevOps are continuous and repeatable, and they are composed by Continuous Integration, Continuous and Automated Deployment, Continuous and Automated Testing, infrastructure as a code and Continuous Learning. For each of these, there are immense set of tools and platforms available to help implement these phases. These tools will be explained from a theoretical point of view and their usage will be analysed with microservices architecture.

Continuous Integration phase requires developers to have the ability to promptly integrate and merge their code in order to obtain quick feedback from managing teams. There are two main parts in this phase. Firstly, the developers need to be able to conveniently save and/or merge their progress, so a version control system tool is needed for this. An iteration of the progress is called a commit.

Then their submitted code needs to be tested before being integrated into the current stable version. There are many ways and combination of ways to achieve this: firstly, as stated previously, automation is a very important aspect of DevOps. Some automatic testing tool or platform is crucial in this stage. There are many options regarding automatic testing, but the main idea is that a series of scripts are run against the committed code that verify that if the code were to be merged into the current development branch, it would not disrupt or introduce errors into the stable project. Secondly some sort of manual approval code review is always recommended, where other colleagues review the submitted progress and request changes or give feedback to the committed progress. Continuous and rapid integration is important so that developers are provided with feedback, by the business team, in early stages of development and thus save resources on the long run and deliver a better product.

In DevOps it is fundamental that software is continuously delivered to some form of environment. Due to the complexity nature of microservices it is important that it the delivery stages of DevOps there are tools that aid deploying software solutions. At an early stage it is not necessarily delivered to end customers, but normally testers and business owners. However, the maintenance of the environment is a hard task that requires coordination between teams and a robust infrastructure.

Once organisations achieve a high level of maturity in DevOps practices and the CI/CD pipeline, they are able to achieve continuous and automated deployment. This builds upon the CI and CD foundations by automating the deployment of every single validated change to production environments. This requires a high level of confidence in the CI/CD pipeline as well as testing practices in the development teams. However, not all organisations are willing to make this process completely automated for all environments (testing, staging, production), thus this process will be considered optional.

Testing plays a pivotal role in software development, particularly with the integration of microservices architecture and its dynamic characteristics, and the DevOps approach that incentivises introducing changes rapidly. It becomes clear that there has to be some form of automated testing.

There are code libraries that easily provide development teams with basic use cases testing on their developments, helping detect errors even before they are merged onto the development branch. But

even on the environment where the solution is hosted, there are platforms that perform more rigorous testing. Depending on the type and requirements of the product, load testing is favoured. With continuous and automated testing new errors are detected earlier and are tracked and eliminated more efficiently.

Segregated applications that make use of microservices architectures require constant communication between business, operations, and developers, not only amongst themselves but also across the different teams participating in the product. Hence the teams must have a robust communication and knowledge sharing platform. The organisation must own a centralised platform that enables seamless communication, information sharing, file sharing, and collaboration across teams and departments. It should allow for each team to be able to insert information and any findings they come across. The goal is to improve efficiency and save time throughout the organisation, avoiding redundancy and duplicated efforts so that no one needs to research a solution that has already been addressed.

But the availability of said platform is not enough, as DevOps demands a shift in culture, regular meetings, end of sprints retrospectives and overall knowledge sharing habits are needed for knowledge sharing to have any effect on the organisation. DevOps incentivises innovation and experimentation.

If these concepts are implemented successfully, any change becomes available, for a push-button ease of deployment to testing, staging and production environments (Debois et al., 2011). Significantly decreasing the release cycles, thus increasing efficiency, errors are detected at a much earlier stage in development, boosting the customer satisfaction, however challenges arise, such as the constant need to collaboration and communication between development and operation teams, a high learning curve in implementing the DevOps practices and the CI/CD pipeline, increasing the complexity of the development-release process.

Specifically, to our proposed framework and use case, in section 4.2 and 4.3, respectively, the following will also be assumed:

That organisations have access to the necessary resources such as infrastructure, tools and skilled personnel to implement DevOps culture and CI/CD practices in their microservices projects effectively.

Since there is a diverse landscape of DevOps and microservices tools available, and that the choice made by organisations depend on various factors such as technology stack across the company, budget and skills, this list will be greatly reduced in order to simplify the framework.

4.2. FRAMEWORK

The following framework is derived from the insights gathered in the literature review chapter, 3.0, being based on the assumptions list in the previous section.

Figure 4.1 presents an overview of the proposed framework, outlining the necessary steps for utilising DevOps along with Microservices, the characteristics of each phase, an overview of the tools that are considered to be the most predominant in each phase will equally be presented, tested in

the form of a use case, and subsequently validated by conducting semi-structure interviews from experts in the field of DevOps and software development.

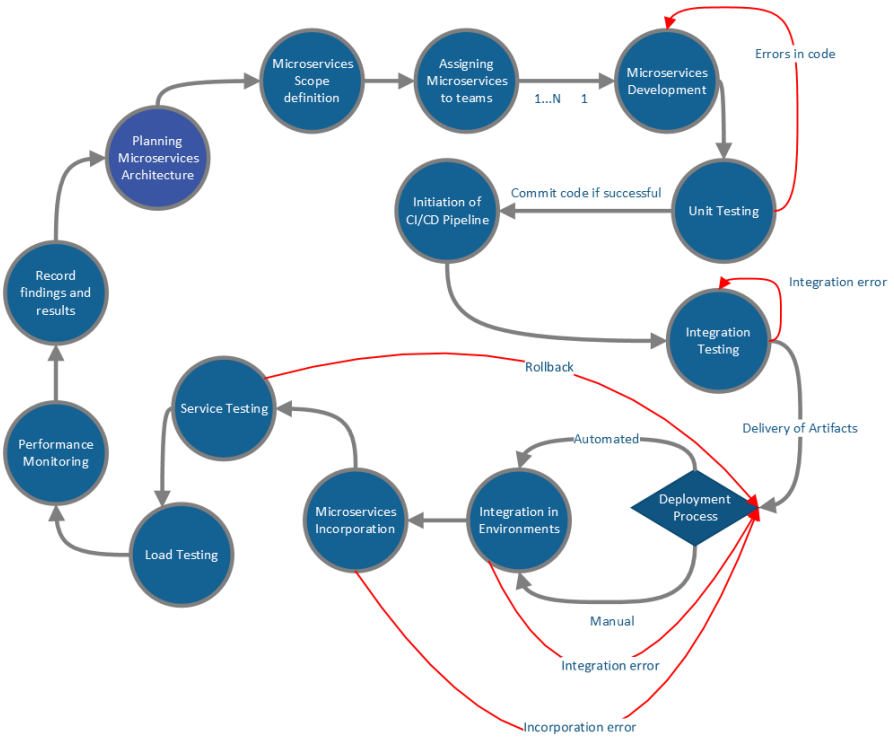

Figure 4.1 - General Overview of the Proposed Framework

To effectively implement microservices architecture alongside DevOps, in the proposed framework, organisations can adopt an iterative framework that integrates the key principles of software development within the DevOps pillars. The proposed framework is an iterative one, meaning that the cycle is repeatable and always seeking improvement and innovation and that its processes are in a state of a constant development.

The planning phase is the beginning of the cycle and one of the most important stages in software development. In the planning phase, teams define project goals, scope, and requirements, sprints definitions and change in requirements or pending errors to be addressed.

In the Microservices Scope Definition, the project as a whole must be broken down into scopes and assigned to the different teams. Each of these teams will be responsible for the development of one, sometimes more, microservices. Collaboration platforms like Confluence, Microsoft Teams should be utilised to document project specifications, plans, business logic that should be known by the entirety of the teams, as well as sprints planning and retrospectives. The progress status of each team should be available to be consulted by the project stakeholders.

During the development phase, development teams write the code iteratively to meet the requirements defined in the planning phase and correct the errors from the other phases of the framework lifecycle. In this stage, developers will interact with version control systems such as Git to enable code collaboration and management. Code review practices among developers are important to ensure code quality and maintainability, as it is important for there to be coding standards

implemented by the team and code reviews are another way of enforcing these standards. Code reviews also act as a way of knowledge sharing between developers, as it is important, specifically in microservices, that the whole team specialises in the scope at hand.

In the code development stage, it is advised and good practice that each developer becomes responsible of writing unit tests for his own code. These unit tests may include functional testing: where functional unit tests verify the behaviour of individual functions or methods. Each programming language has specific libraries, so specially in an application that adopts microservices architecture, it is important that each microservices team has a good and well-defined set of libraries available for unit testing.

Microservices will most likely be based on APIs, so API testing is a major part of ensuring the product is robust. Fortunately, there are very good and easy to use tools for testing an API, like Postman, Swagger, Soap UI. This enables the user to send API requests, validate responses, and ensure the correctness of API behaviour.

CI automates the process of integrating code changes into a shared repository and running automated tests to detect integration errors early. Some of the most popular examples of these tools include Jenkins, Travis CI, CircleCI, GitLab CI/CD.

CI tools interact with version control systems, such as Git, to monitor for code changes. They trigger automated builds and run tests on code commits, notifying developers of build status and the test results. Such tools significantly improve efficiency by reducing manual effort in code integration and testing. They detect errors earlier in the development cycle, enabling quicker feedback and leading to quicker detection and resolution of integration errors.

Tools like JUnit and TestNG are integrated into the CI pipeline and useful for preforming integration testing. Integration testing consists of evaluating the interactions between different modules of an application, making sure they interact as intended together and report results back to the CI system.

If the integration tests have a negative outcome, the developers are notified of the errors that occurred, otherwise the process proceeds to the deployment stage. Subsequently, in the deployment phase, organisations can opt for manual or automated deployment.

Continuous Deployment and **Automated Deployment** occurs when companies have matured in their DevOps culture, after they've integrated continuous integration and continuous delivery, they can advance to continuous deployment. Examples of tools commonly used in CD are Kubernetes, Docker Swarm, AWS Elastic Beanstalk, Google Cloud Run.

Continuous and automated deployment tools automate the deployment of every validated change to production environments without the need for manual intervention. This requires the organisation to have a very solid DevOps culture and continuous integration & continuous delivery practices.

These tools work very closely with the CI/CD pipelines to allow organisations to achieve rapid and reliable software releases with almost no human intervention. They receive validated artifacts from the CI/CD pipelines and automatically deploy them to production environments. This is enabled by receiving validated docker images or application packages, and automatically deploying them to production environments.

Continuous and automated deployment tools manage container orchestration, by scheduling, scaling and load balancing, they also allocate resources to ensure high availability and reliability of the deployed applications.

This process, when done correctly, allows organisations to achieve rapid and reliable software releases in the shortest time possible, greatly reducing deployment times, minimizing human error while maintaining the ability to perform seamless rollbacks in case of issues. With teams being able to deliver changes to production quicker, organisations are able to accelerate time-to-market delivery of their product and increase customer satisfaction. Whether organisations choose to do this process manually or automatically is up to the business owners, but generally the deployment to testing and staging environments are automated while the deployment to production environments are released manually, as organisations prefer to have a better control regarding what features are being deployed to the end-users and when. This process composes the CD stage of DevOps cycle.

The following stages are focused on testing and monitoring.

Service testing is the validation of deployed microservices and ensures its good functioning isolated and along with other microservices. These tests consist of testing the expected outcome and that it is interacting well with the other microservices that compose the final product.

Testing tools are fundamental in validating deployments and functionalities of the application in its hosting environment. Continuous testing and monitoring consist of implementing testing and monitoring tools of the product to ensure its reliability, performance and security. While some of these tools can be integrated on the CI/CD pipeline, there is a need for a focused stage for testing and monitoring. These tools perform duties in the hosting environments, such as Docker, Kubernetes clusters, virtual machines, as well as cloud services like Azure, Amazon Web Services.

Load testing is important to find potential bottlenecks in the application, through the simulation of requests to measure how the service performs under stress. Since not all services will have the same number of requests, more or less resources can be allocated to a high or low demanding web service, according to needs.

As said by (Grande et al., 2024, p. 9) “IaC is a practice that allows the management and configuration of the infrastructure used for a concrete environment.”

It allows for quick and adaptable configurations to each solution, helping detect early errors and increase the quality of releases.

Performance monitoring involves the supervision of web services behaviour and performance in real time. Useful tools for the monitoring of applications include Azure monitoring, Amazon Web Services, Grafana and Prometheus. These tools provide useful feedback in regards to the behaviour of applications, its response times and high access periods, error rates. This allows the early detection of unexpected behaviour and allows for teams to proactively correct arising issues and increase performance, leading to a higher customer satisfaction.

Finally, the sharing of results and findings is important to foster a learning environment within the organisations. In this stage, teams are encouraged to share their findings and insights from their developed work, study other teams’ results and share knowledge between themselves.

4.3. USE CASE

In order to product a use case for the previously proposed framework in chapter 4.2, two criteria were used: the chosen tools enable as many phases of the framework as possible and they are usable in multiple programming languages. Table 4.1 translates the proposed use-case. Of course, each technology is based on the programming language and environment used.

For this particular use case, having in mind that one of the benefits of microservices is that it allows each microservice team to use the technological stack they are most comfortable with, we will consider a product composed of teams using: C# .Net and TypeScript for our API microservices teams and Angular framework for our frontend team.

Table 4.1 – Proposed Framework Use Case

Phase	Description	Recommended Software/Tool
Planning Microservices Architecture	Define project goals, scope, and requirements. Design the Microservices architecture.	GitHub for project management and issue tracking. Microsoft Teams and Confluence for general scopes and definitions.
Microservices Scope Definition	Define the scope of the microservices.	Confluence, Microsoft Teams for project scope definition.
	APIs documentation	Confluence, OpenAPI (Swagger) for API documentation.
Assigning Microservices to teams	Distribute the proposed Microservices domain to specific teams.	Confluence
Microservices Development	IDEs for code writing	Visual Studio, Visual Studio Code, Android Studio.
	Development Environments	Docker
Unit Testing	Unit Testing	JUnit, Jest, Jasmine for unit testing
Initiation of CI/CD Pipeline	Package applications and their dependencies into containers	Docker
Integration Testing	Automate integration and testing of code changes	GitHub Actions for CI/CD pipelines

	Image Registry	Docker Hub, GitHub Container Registry
Deployment Process	Deployment of validated code changes to testing, staging, or production environments (Automated or Manual, depending on environment)	GitHub Actions
Integration in Environments	Integration of artifacts in testing, staging or production environment.	Kubernetes, Azure, Amazon Web Services
Microservices Incorporation	Incorporation of microservice in project.	Docker, Kubernetes
Service Testing	Testing that the microservice is working and produces expected results.	Postman
Load Testing	Load and stress testing of microservice.	Apache Jmeter
Performance Monitoring	Monitorisation of performance from microservice.	Prometheus, Azure Monitor
	Automate testing processes to ensure code quality and reliability	Selenium, Puppeteer for end-to-end testing
	Monitor and log application and infrastructure metrics	ELK Stack (Elasticsearch, Logstash, Kibana) for log analysis
Record findings and results	Gather insights and learnings	Confluence for knowledge sharing across the organisation, Microsoft Teams and Slack for communication between teams.

With the growth of DevOps practices and the need for implementation of CI/CD pipelines, there are various options available for each stage.

When a code change is pushed to a Git repository, a GitHub actions will trigger a CI workflow defined in the repository, and this workflow will run steps to build the docker image for the application based on the latest code changes and specified tests are run to ensure code quality. Following the build of a successful docker image, it is pushed to a Docker image registry, such as Docker hub.

The Docker platform, for instance, facilitates robust testing within the hosting environment, offering a comprehensive testing framework. Docker offers developers the ability to run and ship applications in containers, by encapsulating its application and dependencies and allowing it to run consistently in different environments, such as testing and production environments.

Depending on the nature and requirements of the software product, load testing may also be prioritized to assess performance under various conditions. By integrating continuous and automated testing into the development pipeline, teams can swiftly identify and address potential issues, resulting in more efficient error resolution and enhanced software quality. Docker participates in the development, CI, and partially in testing phases of DevOps.

This transitions to the CD phase of our framework cycle. Kubernetes is a container orchestration and participates in the CD stage of the lifecycle, and it facilitates the deployment of containerised applications to production environments.

Kubernetes interacts with Docker through a container runtime interface, as it can manage container runtimes from different providers with a defined API that is used for creating, starting, stopping, and deleting containers.

By monitoring changes to the docker image registry, once a new docker image is identified, Kubernetes triggers a new deployment process. The new image is pulled and is replaced as the new version of the application within the Kubernetes cluster in the desired environment (development, staging or production). Kubernetes ensures the proper configuration of the application, load balancing and scaling. This segregation is especially useful for microservices as it allows more resources to be put into microservices that need so. This concludes the deployment process in our framework.

After the application is available to the end-users, testing and monitoring is a necessary. There are various monitoring tools that are useful to check the availability of microservices and collected metrics. There are also examples of hosting environments, such as Azure, that operate with Kubernetes and offer a useful set of monitoring tools, performance metrics and logs on the deployed application.

4.4. Framework Evaluation

To validate the proposed framework, three interviews were conducted with experts in the IT industry. The first one was Developer and a DevOps engineer in a national IT company that is a filial of a car manufacturer, the second was another DevOps engineer with three years of experience working specifically with DevOps at a multinational company and overall, twenty years of experience working in IT. Finally, two very experienced engineers were interviewed, a Research and Development Director and a Product Development Manager. Both these engineers have a very high knowledge in product development, DevOps culture and in making the software development process as efficient as possible for all involved parties.

Table 4.2 gives an overview of the background and years of experience of each participant.

Table 4.2 – Interviews Overview

Participant	Background	Approximate years of experience in IT
Expert 1	Developer and DevOps engineer	3
Expert 2	DevOps Engineer	20
Expert 3	Research & Development Director	27
Expert 4	Product Developer Manager	21

All interviews were conducted during the month of June 2024, via zoom meetings and all participants agreed to be recorded and allowed for a transcript of the interview. Each interview is transcribed as part of the master thesis (Appendix A- Appendix C).

The three interviews were semi-structured, and they were preceded by a short presentation giving a brief overview of the thesis, as well as the initial problem, objectives and goals, followed by an explanation of the framework and a use case of the framework by proposing tools for each phase of the framework.

At the end of the presentation, the four following questions were asked, and every participant was given time to add any extra feedback.

Q1: Do you consider the proposed framework useful and why? If not, why do you believe it is not?

Q2: Do you have any criticism or recommendations towards the proposed framework? Please explain.

Q3: Would you consider implementing the proposed framework? Please clarify why/why not.

Q4: Does every member of the team participate in every stage of the lifecycle?

The questions were structured to incentivise the participants to critically evaluate and consider the use of the framework in a real-life environment, but the interview was also flexible in order to allow for any additional ideas or criticisms to be given by the experts.

All four experts found the framework to be useful and precise with what is currently implemented in their companies. Some even said that in previous companies they worked in, such framework would be extremely useful.

Both expert 1 and expert 2 stated that all the processes were correct and in line with the best practices followed within their organisations. Expert 2 further mentioned that some of the tools shown in my use case were used by him and his colleagues implementing the CI/CD pipeline. While

expert 1 suggested for Jira to be considered for the first three stages of the framework (Planning Microservices Architecture, Microservices Scope definition, Assigning Microservices to teams).

Expert 1 considered that “Load testing” and “performance monitoring” occurred in parallel and might not be stages. But expert 2 disagreed with such statement and considered them to be valid steps with a need for attention by the DevOps lifecycle.

Experts 3 and 4 considered the framework to be in accordance with the practices followed in their company except for the “Integration Testing” which in their case, occurred specifically at the Development environment, but mentioned that some companies go as far as doing the “Integration Testing” by replicating the Development environment, helping maintain the Development environment always stable.

Furthermore, they added that the first phase of the proposed framework could have its name improved, as “Planning Microservices Architecture” makes sense in the first cycle of the framework but not so much in the following cycles.

Regarding the usability of the framework in a company looking to implement DevOps, all the interviewed specialists considered the framework to be very precise but simple, in a way that any company looking to adopt a DevOps culture in a MS architecture could use it as a starting point. Adding that each step is shown accurately and provides examples of tools each one.

In a nutshell, the “Planning Microservices Architecture” is considered for a name revision; “Integration Testing” should be considered to occur after “Microservices Integration”. Overall, no major recommendations were made regarding the presented framework.

4.5. Revised Model

The framework structure itself didn’t receive any significant recommendations by the interviewed experts, except for a proposed change of designation of the first stage of the framework, recommending replacing “Planning Microservices Architecture” with “Microservices Architecture and Sprint Planning”. This proposal was accepted as it makes more sense when iterating the framework after the first cycle, as well as making the significance of this phase clearer to the end-user.
Taking into account the feedback of Expert 3 and Expert 4, it was concluded that the “Integration Testing” is often carried out by organisations within the Development environment. So the “Integration Testing” was made optional after the ”Initiation of CI/CD Pipeline” and another optional “Integration Testing in Dev Environment” was added after “Microservices Incorporation”, as suggested. This provides the organisation following this framework with a simpler option of running the integration tests directly on a development environment, or via a specific platform, emulate the environment and run the tests on the generated environment before committing to the actual development environment and thus offering organisations a more stable development environment.

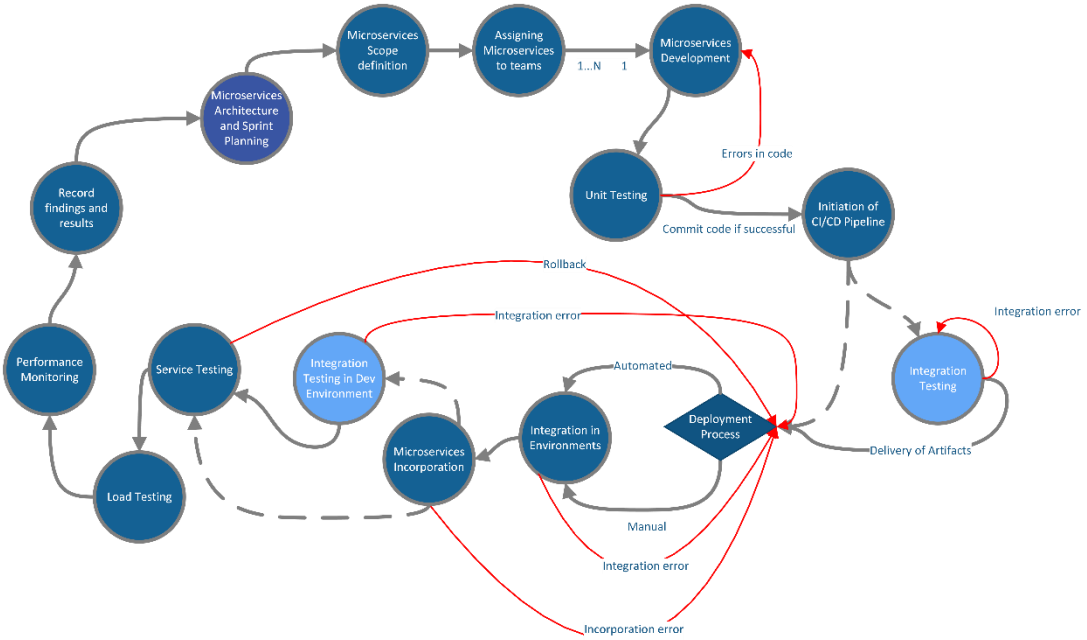

Figure 4.2 – Revised Overview of the Proposed Framework

Jira and Aspire recommended by the interviewed experts and were therefore, included in the use case. The updated version of the use case is displayed in table 4.3 below.

Table 4.3 – Proposed Framework Revised Use Case

Phase	Description	Recommended Software/Tool
Microservices Architecture and Sprint Planning	Define project goals, scope, and requirements. Design the Microservices architecture. Evaluate backlogs and designate sprints.	GitHub, Jira for project management and issue tracking. Microsoft Teams and Confluence for general scopes and definitions.
Microservices Scope Definition	Define the scope of the microservices.	Jira, Confluence, Microsoft Teams for project scope definition.
	APIs documentation	Confluence, OpenAPI (Swagger) for API documentation.
Assigning Microservices to teams	Distribute the proposed Microservices domain to specific teams.	Jira, Confluence

Microservices Development	IDEs for code writing	Visual Studio, Visual Studio Code, Android Studio.
	Development Environments	Docker
Unit Testing	Unit Testing	JUnit, Jest, Jasmine for unit testing
Initiation of CI/CD Pipeline	Package applications and their dependencies into containers	Docker, .Net Aspire
Integration Testing (opt between this and in Dev Environment)	Automate integration and testing of code changes in modules. Conducted on a platform.	GitHub Actions for CI/CD pipelines, .Net Aspire
	Image Registry	Docker Hub, GitHub Container Registry
Deployment Process	Deployment of validated code changes to testing, staging, or production environments (Automated or Manual, depending on environment)	GitHub Actions, .Net Aspire
Integration in Environments	Integration of artifacts in testing, staging or production environment.	Kubernetes, Azure, Amazon Web Services, .Net Aspire
Microservices Incorporation	Incorporation of microservice in project.	Docker, Kubernetes, .Net Aspire
Integration Testing in Development Environment	Automate integration and testing of code changes in modules. Conducted in a dedicated Development environment.	
Service Testing	Testing that the microservice is working and produces expected results.	Postman
Load Testing	Load and stress testing of microservice.	Apache Jmeter, .Net Aspire
Performance Monitoring	Monitorisation of performance from microservice.	Prometheus, Azure Monitor, .Net Aspire

	Automate testing processes to ensure code quality and reliability	Selenium, Puppeteer for end-to-end testing
	Monitor and log application and infrastructure metrics	ELK Stack (Elasticsearch, Logstash, Kibana) for log analysis
Record findings and results	Gather insights and learnings	Confluence for knowledge sharing across the organisation, Microsoft Teams and Slack for communication between teams.

Aspire by .Net was strongly recommended by one of the interviewed experts because if follows the proposed framework very closely.

However, it is confined to the .Net ecosystem, thus goes against the aim of this thesis, which is providing organisations with a framework that isn't technology restricted. Nonetheless it should be considered as a useful tool for software development companies.

This framework should still be considered valuable, even for .Net confined companies, as Microsoft does not consider Aspire to be a replacement for a robust CI/CD systems and tools (Pine et al., 2024).

5. CONCLUSIONS

Although there is a clear consensus that implementing a DevOps culture and CI/CD pipelines across the entire organisation is a demanding task, its benefits are clear.

Indeed, we sometimes forget that the reason we do IT is for the business, and if the business doesn't make a profit, we can be out of a job before we know it. As Phifer emphasizes, this relation to the business is an important aspect in DevOps: some people get stuck on the word "DevOps," thinking that it's just about development and operations working together. Although this feedback loop is important, it must be seen as part of the complete system. Therefore, DevOps must apply to the whole organization, not only between development and operations (Debois et al., 2011).

There is no proven hypothesis that states that microservices are the best overall strategy, neither one that states that DevOps increases productivity as well as efficiency, much less one that recognizes microservices and DevOps combined will deliver the best results. But the truth is that microservices and DevOps go hand in hand in software development and DevOps practices and tools are also mentioned when developing software in microservices architecture. (Arcsona, 2023).

The careful perusal of the current literature on the DevOps culture and Microservices architecture allowed the establishment of a framework that promotes best practices for software development, the studied culture and architecture. This framework was analysed, in the form of a semi-structured interview, by three experts in the software development area, with vast experience in DevOps and Microservices. All interviewed experts considered the framework to be useful and precise, as well as a good starting point for organisations looking to start implementing DevOps in their work culture. Having the feedback into consideration, the framework can be considered valid in the context of microservices development.

5.1. SYNTHESIS OF THE DEVELOPED WORK

The developed framework, that applies DevOps best practices onto microservices architecture, was created following a design science research process. Firstly, the comprehensive literature review was conducted to gather knowledge on DevOps best practices and microservices architecture characterisations. Using this knowledge, research was conducted to determine how DevOps pillars and Microservices practices could potentially best interact with each other. The results and findings from the systematic literature were used to elaborate a framework that aims to help companies start following best practices on developing Microservices architecture with DevOps culture in a way that the productivity and efficiency is enhanced. A use-case was also developed, where some practical tools were suggested. These tools were chosen based on two criteria: having a wide range of application on coding languages; facilitating as many phases of the proposed framework as possible. The proposed framework was then validated by gathering feedback from four DevOps experts that work with software developers, namely developers that utilise microservices architecture.

5.2. LIMITATIONS AND FUTURE WORK

The main limitation of this paper is the fact that it is theory based and validated. This was mainly due to the time limitations inherent to this curricular course. It would be interesting to explore this framework on a more practical point of view, for instance exploring how the proposed tools and libraries correspond to the theoretical expectations. Additionally, it would be useful to validate how this framework performs in a real-world environment, using a real-life company as a use-case.

Furthermore, although the proposal was validated by four experts and the recommended changes were applied directly to the framework, it was not revalidated, once more due to time constraints.

REFERENCES

Abbas, S. imran, & Singh, M. (2023). DevOps for Edge Computing: Challenges and Solutions. *2023 3rd International Conference on Pervasive Computing and Social Networking (ICPCSN)*, 1267–1273. https://doi.org/10.1109/ICPCSN58827.2023.00213

Aderaldo, C., Mendonça, N., Pahl, C., & Jamshidi, P. (2017). *Benchmark Requirements for Microservices Architecture Research*. https://doi.org/10.1109/ECASE.2017.4

Amazon. (2023). *What is DevOps? - DevOps Models Explained - Amazon Web Services (AWS)*. Amazon Web Services, Inc. https://aws.amazon.com/devops/what-is-devops/

Amazon. (2024). *SOA vs Microservices—Difference Between Architectural Styles—AWS*. Amazon Web Services, Inc. https://aws.amazon.com/compare/the-difference-between-soa-microservices/

Arcsona. (2023). *The Benefits of Using a Microservices Architecture in DevOps | LinkedIn*. https://www.linkedin.com/pulse/benefits-using-microservices-architecture-devops-arcsona-inc/

Atlassian. (2024a). *5 Advantages of Microservices [+ Disadvantages]*. Atlassian. https://www.atlassian.com/microservices/cloud-computing/advantages-of-microservices

Atlassian. (2024b). *Continuous integration vs. Delivery vs. Deployment*. Atlassian. https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

Atlassian. (2024c). *History of DevOps*. Atlassian. https://www.atlassian.com/devops/what-is-devops/history-of-devops

Bakshi, K. (2017). Microservices-based software architecture and approaches. *2017 IEEE Aerospace Conference*, 1–8. https://doi.org/10.1109/AERO.2017.7943959

Baumgartner, J. K. (2022). *From Monolith to Microservices*. https://run.unl.pt/bitstream/10362/150901/1/TGI1548.pdf

Beckman, M. D., Çetinkaya-Rundel, M., Horton, N. J., Rundel, C. W., Sullivan, A. J., & Tackett, M. (2021). Implementing Version Control With Git and GitHub as a Learning Objective in

Statistics and Data Science Courses. *Journal of Statistics and Data Science Education*, *29*(sup1), S132–S144. https://doi.org/10.1080/10691898.2020.1848485

Braga, P. H. P., Hébert, K., Hudgins, E. J., Scott, E. R., Edwards, B. P. M., Sánchez Reyes, L. L., Grainger, M. J., Foroughirad, V., Hillemann, F., Binley, A. D., Brookson, C. B., Gaynor, K. M., Shafiei Sabet, S., Güncan, A., Weierbach, H., Gomes, D. G. E., & Crystal-Ornelas, R. (2023). Not just for programmers: How GitHub can accelerate collaborative and reproducible research in ecology and evolution. *Methods in Ecology and Evolution*, *14*(6), 1364–1380. https://doi.org/10.1111/2041-210X.14108

Brocke, J. vom, Hevner, A., & Maedche, A. (2020). Introduction to Design Science Research. In *Design Science Research. Cases* (pp. 1–13). Springer International Publishing. https://doi.org/10.1007/978-3-030-46781-4_1

Brown, M., Dikshit, Harrysson, Srivastava, & Thanki. (2020). *A new management science for banking technology delivery | McKinsey*. https://www.mckinsey.com/industries/financial-services/our-insights/a-new-management-science-for-technology-delivery

Camunda. (2018). *New Research Shows 63 Percent of Enterprises Are Adopting Microservices Architectures Yet 50 Percent Are Unaware of the Impact on Revenue-Generating Business Processes*. Camunda. https://camunda.com/press_release/new-research-shows-63-percent-of-enterprises-are-adopting-microservices/

Chowdary, M. N., Bussa, S., Chowdary, C. K., & Gupta, M. (2022). Automated Pipeline for the Deployment using OpenShift. *Procedia Computer Science*, *215*, 220–229. https://doi.org/10.1016/j.procs.2022.12.025

Colomo-Palacios, R., Fernandes, E., Soto-Acosta, P., & Larrucea, X. (2018). A case analysis of enabling continuous software deployment through knowledge management. *International Journal of Information Management*, *40*, 186–189. https://doi.org/10.1016/j.ijinfomgt.2017.11.005

Cortés Ríos, J. C., Embury, S. M., & Eraslan, S. (2022). A unifying framework for the systematic analysis of Git workflows. *Information and Software Technology*, *145*, 106811. https://doi.org/10.1016/j.infsof.2021.106811

Debois, P., Humble, J., Molesky, J., Shamow, E., Fitzpatrick, L., Dillon, M., Phifer, B., & DeGrandis, D. (2011). *DevOps: A Software Revolution in the Making? | Cutter Consortium*. https://www.cutter.com/offer/devops-software-revolution-making-0

Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. *Journal of Systems and Software*, *150*, 77–97. https://doi.org/10.1016/j.jss.2019.01.001

Elrhanimi, S., el Abbadi, L., & Abouabdellah, A. (2018). Lean manufacturing: From the craft production to the global emergence. *International Journal of Engineering and Technology(UAE)*, *7*, 54–59.

Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, *123*, 176–189. https://doi.org/10.1016/j.jss.2015.06.063

Gajda, M., Cooper, P., Mezzalira, L., Rodger, R., & Banskota, S. (2020). *State of Microservices 2020 Report | TSH.io*. https://tsh.io/state-of-microservices/

Giallorenzo, S., Montesi, F., Peressotti, M., & Rademacher, F. (2023). LEMMA2Jolie: A tool to generate microservice APIs from domain modelsImage 1. *Science of Computer Programming*, *228*, 102956. https://doi.org/10.1016/j.scico.2023.102956

Giamattei, L., Guerriero, A., Pietrantuono, R., Russo, S., Malavolta, I., Islam, T., Dînga, M., Koziolek, A., Singh, S., Armbruster, M., Gutierrez-Martinez, J. M., Caro-Alvaro, S., Rodriguez, D., Weber, S., Henss, J., Vogelin, E. F., & Panojo, F. S. (2024). Monitoring tools for DevOps and microservices: A systematic grey literature review. *Journal of Systems and Software*, *208*, 111906. https://doi.org/10.1016/j.jss.2023.111906

Git. (2024). *Git—Git Documentation*. https://git-scm.com/docs/git

Gokarna, M. (2021). *DevOps phases across Software Development Lifecycle* [Preprint]. https://doi.org/10.36227/techrxiv.13207796.v2

Grande, R., Vizcaíno, A., & García, F. O. (2024). Is it worth adopting DevOps practices in Global Software Engineering? Possible challenges and benefits. *Computer Standards & Interfaces*, *87*, 103767. https://doi.org/10.1016/j.csi.2023.103767

Hasselbring, W. (2018). Software Architecture: Past, Present, Future. In V. Gruhn & R. Striemer (Eds.), *The Essence of Software Engineering* (pp. 169–184). Springer International Publishing. https://doi.org/10.1007/978-3-319-73897-0_10

Jacobs, M., Casey, C., & Kaim, E. (2022). *What are Microservices? - Azure DevOps*. https://learn.microsoft.com/en-us/devops/deliver/what-are-microservices

Jia, H., Yang, W., Shen, C., Pan, M., & Zhou, Y. (2023). Git command recommendations using crowd-sourced knowledge. *Information and Software Technology*, *159*, 107199. https://doi.org/10.1016/j.infsof.2023.107199

Khattak, K.-N., Qayyum, F., Naqvi, S. S. A., Mehmood, A., & Kim, J. (2023). A Systematic Framework for Addressing Critical Challenges in Adopting DevOps Culture in Software Development: A PLS-SEM Perspective. *IEEE Access*, *11*, 120137–120156. IEEE Access. https://doi.org/10.1109/ACCESS.2023.3325325

Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations* (Illustrated edition). IT Revolution Press.

Knoche, H., & Hasselbring, W. (2019). *Drivers and Barriers for Microservice Adoption—A Survey among Professionals in Germany*. *14*, 1–35. https://doi.org/10.18417/emisa.14.1

Morris, D., Voutsinas, S., Hambly, N. C., & Mann, R. G. (2017). Use of Docker for deployment and testing of astronomy software. *Astronomy and Computing*, *20*, 105–119. https://doi.org/10.1016/j.ascom.2017.07.004

Munaf, R. M., Ahmed, J., Khakwani, F., & Rana, T. (2019). Microservices Architecture: Challenges and Proposed Conceptual Design. *2019 International Conference on Communication Technologies (ComTech)*, 82–87. https://doi.org/10.1109/COMTECH.2019.8737831

Pedchenko, Y., Ivanchenko, Y., Ivanchenko, I., Lozova, I., Jancarczyk, D., & Sawicki, P. (2022). Analysis of modern cloud services to ensure cybersecurity. *Procedia Computer Science*, *207*, 110–117. https://doi.org/10.1016/j.procs.2022.09.043

Pine, D., Montemagno, J., Hazell, L., Moseley, D., Erhardt, E., Matthews, A., & Fowler, D. (2024, May 23). *.NET Aspire overview—.NET Aspire*. https://learn.microsoft.com/en-us/dotnet/aspire/get-started/aspire-overview

Potdar, A. M., D g, N., Kengond, S., & Mulla, M. M. (2020). Performance Evaluation of Docker Container and Virtual Machine. *Procedia Computer Science*, *171*, 1419–1428. https://doi.org/10.1016/j.procs.2020.04.152

Richardson, C. (2024). *Microservices Pattern: Monolithic Architecture pattern*. Microservices.Io. http://microservices.io/patterns/monolithic.html

Robitzsch, S., Centenaro, M., di Pietro, N., Cordeiro, L., Gomes, A. S., Sanders, P., & Ishaq, A. (2023). Prospects on the adoption of a microservice-based architecture in 5G systems and beyond. *Computer Networks*, *237*, 110058. https://doi.org/10.1016/j.comnet.2023.110058

Sampaio Junior, A., Kadiyalax, H., Hux, B., Steinbachery, J., Erwinz, T., Rosa, N., Beschastnikhx, I., & Rubin, J. (2017). *Supporting Microservice Evolution*. https://doi.org/10.1109/ICSME.2017.63

Sharma, A., Kumar, M., & Agarwal, S. (2015). A Complete Survey on Software Architectural Styles and Patterns. *Procedia Computer Science*, *70*, 16–28. https://doi.org/10.1016/j.procs.2015.10.019

SPM Global Technologies. (2023). *DevOps Solutions: Streamlining Software Development and IT Operations | LinkedIn*. https://www.linkedin.com/pulse/devops-solutions-streamlining-software-development-operations/

Stack Overflow. (2023). *Stack Overflow Developer Survey 2023*. Stack Overflow. https://survey.stackoverflow.co/2023/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2023

Tanzil, M. H., Sarker, M., Uddin, G., & Iqbal, A. (2023). A mixed method study of DevOps challenges. *Information and Software Technology*, *161*, 107244. https://doi.org/10.1016/j.infsof.2023.107244

Varshneya. (2021). Understanding design of microservices architecture at Netflix. *TechAhead*. https://www.techaheadcorp.com/blog/design-of-microservices-architecture-at-netflix/

Waseem, M., Liang, P., & Shahin, M. (2020). A Systematic Mapping Study on Microservices Architecture in DevOps. *Journal of Systems and Software*, *170*, 110798. https://doi.org/10.1016/j.jss.2020.110798

Wayner, P. (2023). *How to choose the right software architecture: The top 5 patterns*. TechBeacon. https://techbeacon.com/app-dev-testing/top-5-software-architecture-patterns-how-make-right-choice

Wickramasinghe, S. (2021). *The Role of Microservices in DevOps*. BMC Blogs. https://www.bmc.com/blogs/devops-microservices/

APPENDIX A

Interview 1 - transcript, 5th June 2024

Interviewer:
This concludes my framework. Of course, every tool listed here depends on the technology stack of the organisation using the framework but it serves as s starting point for a company trying to implement DevOps practices while using or migrating to microservices architecture.
I would just like to ask you the following three questions: if you would consider the proposed framework useful and why? Or please tell me if you completely disagree. If you have any criticism or recommendations towards the proposed framework? And if you would you consider implementing the proposed framework? Please clarify why/why not. Of course, feel free to add any other input or feedback to my framework.

Expert 1:
Let me just see the framework slide again please. From my part, according to my experience, as you mentioned correctly, DevOps is a culture and is influenced by the people in the team. In my case, we have some points in your proposed framework that are not sequential but are parallel. I am talking about “Performance monitoring” and “load testing”. When we are executing a deployment, we do indeed have a service testing that makes sure everything is completely working, but the load and performance monitoring is always happening in the background.

But of course, it is a culture, and the process may vary from company to company. Speaking of the framework per se, it is a hundred percent aligned with the culture in general.

I would also like to recommend Jira for the first three phases of the framework. I don’t know if you have worked with it so far.

Interviewer:
Absolutely, I’ve worked with it and it’s useful for tasks planning.

Expert 1:
Unit testing is correct, integration testing as well. This is used on a team level, right?

Interviewer:
Yes, from “microservices development” forward, it is all done by a single team.

Expert 1:
Other than this, I have nothing else to note. I think it looks great. This answers the first question and second questions regarding my recommendations.

Interviewer:
Thirdly I’d like to ask you if for example you were implementing DevOps and microservices for the first time in a company, if you find this framework useful.

Expert 1:
Absolutely, I would recommend it. In my case, I’ve worked with a company where I was a developer only, and the Ops part was assigned to a completely different team. The developer team participated

in the first stage (Planning Microservices Architecture) right until the deployment process. Everything else was assigned to the Ops team. This was a barrier between teams, which made it harder to communicate and treat occurring errors. Really limiting the communication and agility between the team. So yes, I think your proposed framework would be useful to implement in such a company and would increase the productivity. It is a very simple framework, in the sense that any person in the team, whether junior, mid-level or senior level can look at it and get to understand the responsibilities and lifecycle of a DevOps workflow.

Other than that, I have nothing else to add to any of the questions.

Interviewer:
That's exactly the goal of this framework, to provide people with a base for anyone to jump into the framework and start to get a grasp of each phase.

Regarding the segregation between Dev and Ops teams, I found that really interesting because that proves what I have read in my literature review, stating that it really increases productivity, that the whole team participates in all phases of the lifecycle, helping solve errors and increasing efficiency.

In your current company, does each developer participates in all stages of the DevOps lifecycle?

Expert 1:
Yes! When we are, for example delivering a change or migration in production, we plan the migration, which cluster to include, etc. We then assign it to a member, the normal development occurs, including unit testing. Then we start the CI/CD pipeline, with the integration testing, enabling continuous integration. Which is what I am currently doing and consists in validating if the environments are eligible for the artifacts, that the communication systems that work together work, enabling the deployment and testing it. So answering your question directly, we participate in every stage, yes.

Another important aspect is the on call 24/7. Which fits into the monitoring.

Interviewer:
So, like a prevention team, for any occurrences. Maybe rotating the team for each of them to be on prevention.

Expert 1:
Yes exactly! We have one member on prevention each week. But I'm not sure how this would fit into your framework. It's just a responsibility we have in our team.
Apart from the things I mentioned so far, I don't think I have anything further to add. Do you have any other questions?

Interviewer:
No, that is all. You were very helpful. Thank you so much for your time.

Expert 1:
Not at all, thank you as well. Anything else you need let me know.

APPENDIX B

Interview 2 - transcript, 7th June 2024

Interviewer:
Do you think this framework makes sense? And do you have any recommendations or criticisms?

Expert 2:
I think the framework makes sense. All the tests and deployments being incorporated in the CI/CD pipeline makes total sense.

Interviewer: Do you think “Load testing” and “Performance Monitoring” shouldn’t be phases in the framework?

Expert 2:
I think they should be phases in the framework, yes. They are steps we individually have to execute and think of and ideally there are incorporated in the pipeline, everything should be as automated as possible. In every project I’ve participated the performance and load testing are a part of the process and of course, the more automated they are, the better.

Interviewer:
Thirdly, if you for example were moving to a new company that doesn’t use DevOps, do you think this framework would be helpful and would you consider implementing it?

Expert 2:
Yes, it would definitely be helpful. Nowadays, if a company is trying to implement continuous and agile development, without a clear framework like yours, with all the detailed steps, it would be in trouble. Developing in monolithic architecture and without follwing good practices is not a good idea at all. It’s hard to maintain, manage, everything is more difficult to work with since it’s all so tightly coupled together. In order to change a component, you need to change the whole project. With microservices that doesn’t happen, obviously there has to be an integration but changes in microservices are much easier to manage.

Interviewer:
Does everyone in your team work in each step of the product life cycle?

Expert 2:
In my current project we have an actual DevOps team, which is my yeam. We have a dedicated team to DevOps lifecycle, and we support the developer team and work very closely together.

So, we deliver the CI/CD to the developers, and they just have to work with it and commit their code using the CI/CD. For example, I use Jenkins for various steps in your framework.

In my previous team everyone had a specific task: we had a scrum master, the developers, the QAs, and the DevOps. The project owner informed the scrum master what were the requirements, the scrum master discussed the requirements with the developer team and built user stories. In the CI/CD the developers transmitted the user stories to the QA team, they conducted the testing and near the end of the cycle, the developers executed the performance tests.

Interviewer: So, in your current team, you have a specific DevOps team and provide the cycle to the development team?

Expert 2:
Exactly! I work in a DevOps team, we specifically work in the pipeline process and we facilitate the development team with the pipeline and the tools to execute it. Whether it is Jenkins, Prometheus, etc.

Interviewer:
How many people are in your team?

Expert 2:
Five, in the DevOps team.

Interviewer:
And the developers?

Expert 2:
That really depends on the project. Some teams have ten, others have fifteen.

Interviewer:
Okay, great. I don’t know if you have anything else to add or any other questions.

Expert 2:
Not from my side, I hope I helped you and if you need anything else let me know.

APPENDIX C

Interview 3 - transcript, 21st June 2024

Expert 3:
First of all, let add a few comments to this circle framework. I think it is interesting that it is a circle because it means that it is iterative. Some of these processes I don't think are repeated after every cycle. More specifically the "Planning Microservices Architecture", there are indeed some cycles where it is executed but it is not executed in every cycle.
Is the integration testing and the Integration in environments in the pipeline? Because most of the times they are performed by the same platform.

In the .Net world, we now have something called Aspire, previously known as TYE, which very closely follows your theoretical framework. It has the advantage that it clearly guides the user to each step, in a very transparent way. You can even breakpoint inside your local machine, since when you deploy to an environment, your code is exactly the same because the Aspire tool does everything for you on the background, all the hooks and integration between microservices. Which after all, is the most challenging part of microservices development: making sure everything is connected together and working. Because each microservice is always changing and you have to guarantee they work well together.
Another advantage is that it eliminates the need for extra platforms to execute the steps in a framework like yours.

Interviewer:
I haven't heard of Aspire but it seems very interesting.

Expert 3:
It is a very recent tool, it is still in preview but I think it will be made core with the new .Net version coming out in September.

Interviewer:
My framework is a series of steps, indeed some tools may conduct one or more phases of my framework at a time. For instance, Aspire would work very well with a .Net working company, but for a company that doesn't work with .Net other tools are still needed.

Expert 3:
Yes absolutely, Aspire is very useful for .Net but it only works with .Net, so for other technology stacks, other tools are needed.

Another challenge we have has to do with the amount of technologies and tools that requires a series of workshops and courses for our developers. They are currently reluctant to learning so many tools and claim that "I already know jest, now I have to learn Postman, Github Actions and all these other tools" that they start to abstain themselves from doing so and pass the responsibility to someone else. Which becomes a problem for us, because people are refraining from some responsibilities and this way can't be aware of the impact of their code, quality assurance is diminished because they don't properly conduct QA tests thinking that someone will eventually do them. This introduces more errors in our code and efficiency.

If it is a minor mistake and someone later in the pipeline is able to quickly fix it, fine, however, most of the times it is not so simple, and a mistake ends up slowing the whole team deployment.

When the developer knows, even if on the surface, the entire deployment process, he himself is able to validate his code and build, eliminating lots of errors. So, when the team is able to master your framework from start to finish, or at least understand in which phase he is working on, it makes the entire process much easier and the team is much more productive.

Interviewer:
That's what I noticed from the literature review and other interviews. That there is indeed a steep learning curve when implementing DevOps methodologies and tools in the deployment process. DevOps increases the productivity drastically, but it requires a big initial effort from the team and each member individually in learning the process and the tools.

Expert 3:
Exactly. There is indeed a learning curve that we are constantly trying to reduce in our organisation.

Expert 3:
Your framework is what I'd consider to be a modern development framework. It's not different to what we currently do. I could change the names of some of the tools and this would be basically what we currently follow.
Maybe change the name of the planning Microservices Architecture. (sprints, occur in every cycle).

Interviewer:
Maybe change it to a name that referrers to project planning as well as sprint planning.

Expert 3:
Something we do is every two or three months we have an architecture review meeting where we discuss new possible tools and evaluate our current tools. We interact with the team and the team tells us what went wrong on a specific deployment, why did the build fail, what major issues they're having, and we try to all come up with possible improvements for our CI/CD pipeline.
But any changes that come from these meetings take time to be implemented. Firstly, we must carefully consider the impact of the tools: will they make our time easier? Will they be easier to be used and learned by our team?
We don't want the team to change tools after every sprint, that would be chaotic. Then, after around 6 months of careful consideration, they begin to be used by our teams.

Interviewer:
Does everyone in your team participate in the entire process of the framework?

Expert 3:
We have very small teams; our biggest team is around 15 people. We prefer to keep teams small.

Expert 4:
But not everyone in the team participates in all the processes in the framework described. We try to have everyone be aware of what's going on, as we mentioned previously. We have some people just defining the architecture, others are just participating in the integrations tests, the developers write

code as well as unit tests, we have a QA team that create the performance tests, some people specialise in the CI/CD.

Expert 3:
If you ask if we have very specialised people, we try not to. We try to not have everyone implement each phase, but we want everyone to know what each phase does. Because if they don't, we will have longer build times, unnecessary errors, security breaches, bad product performance.
For example, if a microservice is slow, deconstruct the process and understand if it is a caching problem, if it is a database problem, or a transaction problem. There are multiple things that can be done to improve the perception of performance, and if the team doesn't have a good perception of what is happening, they will not be able to improve their performance. We don't want only senior members to understand the process, it is normal that they have more knowledge than the rest, but it is important that junior and mid members have some knowledge on performance monitoring as well.

Expert 3:
Regarding the integration testing, we have that working directly on an environment. We normally have three environments: dev, staging and production, and the development environment already has the integration tests. For instance, if we run one of those tests that replicates browser interactions, data insertion, it gives us complete feedback on if the pieces are working together and we can figure out if a specific microservice is faulty. This is important because if a microservice is in the middle of the system, it can, sometimes, break 70% of the product.
We simply find it easier to run integration tests in the dev environment, which replicates the production environment, that on a testing machine trying to replicate the production environment and running the integration tests there.

Interviewer:
So maybe in your case, the "Integration Testing" would be situated between "Microservices Incorporation" and "Service Testing".

Expert 3:
Yes, that would make more sense in our case. Every commit uploads the feature to the Kubernetes Dev environment. And we aren't going as far as I've seen some teams do, which is for each feature branch, creating a specific DNS branch for that request. So, on every pull-request, a specific environment is created and the team member testing the pull request can test the changes in the feature branch, a replication of Dev and if the tests succeed, a different build is uploaded to the actual Dev environment. This helps maintain the Dev environment stable. But this is a case that happens in much more mature teams.

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa