

**POLITECNICO DI BARI**

**BSc in COMPUTER AND AUTOMATION ENGINEERING**

**Thesis in: SOFTWARE ENGINEERING**

**INNOVATIVE TECHNIQUES FOR AGILE DEVELOPMENT: DEVOPS METHODOLOGY TO IMPROVE SOFTWARE PRODUCTION AND DELIVERY CYCLE**

|  |  |
| --- | --- |
| **Supervisor**  Prof.ssa Marina MONGIELLO  **Assistant Supervisor**  Dott. Ing. Francesco NOCERA Prof. Tommaso DI NOIA | **Candidate**  Davide GALLITELLI |

Academic Year 2015/2016

*“Soltanto una cosa  
 rende impossibile un sogno,  
la paura di fallire”*

*- Paulo Coelho*

*A chi sostiene le mie scelte,  
a chi gioisce delle mie vittorie,  
a chi mi conforta nelle sconfitte.  
  
Alla mia famiglia.*

# Abstract in Italian

Raggiungere il mercato ed il cliente in tempi quanto più brevi possibili rispetto alla concorrenza rappresenta un fattore di notevole importanza per vecchie e nuove aziende. Gli attori in gioco nella produzione e rilascio di un servizio, a partire dalla progettazione fino alla gestione dell’infrastruttura, sono molteplici ed eterogenei, rendendo complesso il processo di velocizzare tutte le fasi incluse nel ciclo di sviluppo del software.

Nel corso degli anni il processo software è passato da modelli di sviluppo *Waterfall* a *Agile*, introducendo concetti quali lo sviluppo iterativo, l’importanza del feedback dell’utenza finale e della velocità. La filosofia Agile ha permesso un migliore collegamento fra la prospettiva Business e quella Development (Dev), tralasciando però un importante settore -- le IT Operations (Ops) - responsabili di tutte le fasi successive allo sviluppo, quali rilascio, supporto e gestione dell’infrastruttura.

Tale staticità del settore Ops rappresenta un vero e proprio collo di bottiglia per le aziende che puntano ad una maggiore produttività e presenza sul mercato attraverso nuovi servizi o funzionalità.

In questa tesi si studia la metodologia DevOps, un nuovo movimento culturale basato sulla partecipazione di tutta la catena produttiva e degli attori in essa presenti al ciclo di vita del software, avvicinando realtà storicamente scarsamente collaborative, per l’appunto Dev ed Ops, rendendole partecipi del progetto in maniera continuativa e collaborativa. Ciò comporta una più elevata reattività alle richieste di mercato ed una riduzione del tempo di reazione ai feedback ricevuti da parte dei clienti. La metodologia DevOps permette anche il connubio fra stabilità e funzionalità, tradizionalmente in rapporto antitetico a causa degli inevitabili bug introdotti con l’aggiunta di nuove funzionalità ad un servizio già esistente.

La metodologia DevOps permette alle organizzazioni che la adottano di beneficiare di alcuni fondamentali vantaggi. In primo luogo, DevOps riduce drasticamente la durata dei cicli di produzione, risultando in più brevi tempi di rilascio sul mercato, sia di servizi che di funzionalità aggiuntive. Applicando i principi DevOps su tutta la linea produttiva, la qualità finale risulta migliorata su più livelli, con una più elevata disponibilità dei servizi ed un numero inferiore di falle. DevOps migliora l’efficacia dell’organizzazione, dando la possibilità di usufruire del tempo guadagnato evitando lavori ripetitivi o critici e riallocandolo in attività che risultino in valore aggiunto per l’azienda e per il cliente finale.

La struttura del lavoro di tesi svolto è da intendersi come una “guida” all’adozione della metodologia DevOps in un’azienda di medie-grandi dimensioni. In particolare, la prima fase consiste nel definire una base teorica della suddetta metodologia, analizzandone radici, letteratura e modelli esistenti. Ciò è particolarmente importante per il carattere culturale da introdurre nei team che intendano seguire un approccio DevOps alla produzione e rilascio del software. Successivamente, ci si sposta all’analisi di strumenti e prodotti che migliorino la produttività dell’azienda applicando i principi precedentemente descritti. La scelta di uno strumento rispetto alla rispettiva concorrenza è stata basata sulle menzioni di ricerche in Google tramite Google Trends, utilizzato come indicatore per discriminare le tecnologie più ricercate al momento della stesura della tesi nella pluralità di software sviluppati con simili obiettivi. Infine, si passano in analisi “casi aziendali” ed i risultati delle aziende che hanno deciso di adottare con successo un approccio DevOps per la loro catena produttiva. Nomi eccelsi quali Amazon, Facebook, e Netflix rientrano in questa categoria, e lo studio della loro casistica rappresenta una prova fondamentale della validità della metodologia DevOps e dei vantaggi che da essa si possono trarre.

Table Of Contents

[Abstract in Italian 6](#_Toc462170061)

[Table of Figures 11](#_Toc462170062)

[Chapter I: Introduction 14](#_Toc462170063)

[Chapter II: Definition of DevOps 16](#_Toc462170064)

[2.1. The roots of the movement 17](#_Toc462170065)

[2.1.1. The evolution of the Development process 17](#_Toc462170066)

[2.1.2. The static nature of IT Operations 21](#_Toc462170067)

[2.2. Challenges faced 23](#_Toc462170068)

[2.3. The DevOps philosophy 25](#_Toc462170069)

[2.3.1. A short history for a big movement 25](#_Toc462170070)

[2.3.2. DevOps Values and Principles 26](#_Toc462170071)

[2.4. Architectural components 34](#_Toc462170072)

[2.4.1. The need for a reference 34](#_Toc462170073)

[2.4.2. Culture Adoption 35](#_Toc462170074)

[2.4.3. End-to-End Value Production with Continuous Deployment 36](#_Toc462170075)

[2.4.4. The importance of continuous monitoring and feedback 41](#_Toc462170076)

[Chapter III: The DevOps tool rack 45](#_Toc462170077)

[3.1. Collaboration 47](#_Toc462170078)

[3.1.1. A Culture of Collaboration 47](#_Toc462170079)

[3.1.2. Source Code Management 50](#_Toc462170080)

[3.2. Continuous Deployment 52](#_Toc462170081)

[3.2.1. Continuous Integration 52](#_Toc462170082)

[3.2.2. Build Automation 52](#_Toc462170083)

[3.2.3. Automated Deployment 53](#_Toc462170084)

[3.3.4. Continuous Testing 57](#_Toc462170085)

[3.3. Building the Infrastructure 61](#_Toc462170086)

[3.3.1. Configuration Management 61](#_Toc462170087)

[3.3.2. Containers and VMs 63](#_Toc462170088)

[3.4. Monitoring & Logging 67](#_Toc462170089)

[3.5. Moving to the Clouds 69](#_Toc462170090)

[Chapter IV: Conclusions 74](#_Toc462170091)

[Bibliographic and Web References 78](#_Toc462170092)

# Table of Figures

[Figure II.1 - 2015 Gartner's Hype Cycle for Enterprise Architecture 16](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170129)

[Figure II.2 - Waterfall Development Life Cycle 18](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170130)

[Figure II.3 - Agile Development Life Cycle 20](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170131)

[Figure II.4 - The CALMS model as a flow 28](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170132)

[Figure II.5 - Automated deployment pipeline 30](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170133)

[Figure II.6 - The "First Way" of DevOps: Systems Thinking 32](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170134)

[Figure II.7 - The "Second Way" of DevOps: Amplify Feedback Loops 33](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170135)

[Figure II.8 - The "Third Way" of DevOps: Culture of Continual Experimentation and Learning 33](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170136)

[Figure II.9 - DevOps Reference Architecture 34](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170137)

[Figure II.10 - Cultural Change of Team's Organization 35](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170138)

[Figure II.11 - Delivery Pipeline Overview [9] 36](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170139)

[Figure II.12 - Evolution towards Continuous Deployment (from left to right: Waterfall methodology, Agile methodology, DevOps methodology) 37](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170140)

[Figure II.13 - Relative costs to fix software defects (Source: IBM Systems Sciences Institute) 39](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170141)

[Figure II.14 - Four Vectors of the DevOps Feedback System 43](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170142)

[Figure III.1 - DevOps Periodic Table of Tools (credits to XebiaLabs) 46](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170143)

[Figure III.2 - Google Trends for Collaboration Software 47](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170144)

[Figure III.3 - Slack Logo 48](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170145)

[Figure III.4 - Trello Logo 49](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170146)

[Figure III.5 - Google Trends for SCM Software 50](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170147)

[Figure III.6 - Git and GitHub's logos 51](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170148)

[Figure III.7 - Build to Deployment Automation chain 53](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170149)

[Figure III.8 - Google Trends for Continuous Integration 54](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170150)

[Figure III.9 - Jenkins Logo 55](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170151)

[Figure III.10 – Pyramidal pattern of Automated Testing and ice cream cone anti-pattern 57](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170152)

[Figure III.11 - Google Trends for Continuous Testing 59](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170153)

[Figure III.12 - Selenium Logo 59](#_Toc462170154)

[Figure III.13 - Google Trends for Configuration Management Tools 62](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170155)

[Figure III.14 - Ansible Logo 62](#_Toc462170156)

[Figure III.15 - Differences between Virtualization and Containerization 63](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170157)

[Figure III.16 - Google Trends for Containerization and Virtualization Tools 64](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170158)

[Figure III.17 - Docker Logo 65](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170159)

[Figure III.18 - Vagrant Logo 66](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170160)

[Figure III.19 - The Elastic Stack Structure for Logging and Monitoring 67](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170161)

[Figure III.20 - Google Trends for Cloud Computing Platform 70](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170162)

[Figure III.21 - Amazon Web Services' Logo 70](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170163)

[Figure III.22 – Example Computing and Networking Architecture of AWS 71](file:///C:\Users\david_000\Dropbox\Università\Tesi%20in%20Ingegneria%20del%20Software\DevOps.docx#_Toc462170164)

# Introduction

Today, every company in IT can be a software company. Software applications, including mobile apps, are becoming increasingly important for all businesses. Customers are expecting to get an increasing number of features, faster. The success of a company depends on its ability to offer new and exciting products to its customers, faster with respect to its competitors. Staying ahead of competition puts an increasing pressure on software developers to produce new features at an increasing speed.

The huge variety of IT companies on the market implies diversity and complexity in objectives and directives of a team. For many years a “silver bullet” for project management has been sought, although in vain. Many software development methods have emerged due to deficiencies in the existing methods and others have been created completely from the ground up to meet new development philosophies. The project’s characteristics generally led the team to converge on one development method or another. New software development practices such as Agile Software Development encourage small incremental changes which means developers can deliver code changes frequently - often several times in one day.

With these conventional methods however, a gap is formed within teams. Members have different goals, tools and procedures, and sometimes they might even be located in different geographic locations with limited communication. Although the development cycle can be as short as a few hours trying to accommodate ever-increasing requirements, the deployment process can be a bottleneck in many organizations, due to differences existing between development and production environments. This causes the clogging of the process of delivering the perfect solution to the customer, causing its dissatisfaction and possibly a loss of value for the company behind the product.

In a hyper-competitive economy, a transformation is in need, not just with the advancing development technologies, but also in the thought process of the whole project and the philosophy behind the teams working on it, in order to deliver the best product as fast as possible while listening to the feedback from the customers.

DevOps is the new way of integrating the two conflicting worlds of Development and IT Operations, removing the static layout of two opposing teams in order to improve collaboration and productivity through the automation of the workflow and continuous measurement of application performance. DevOps can be considered a movement, a culture that emphasizes collaboration and aims at creating an environment where developing, testing and releasing software can happen rapidly, frequently and more efficiently.

The goal of this work is to introduce the meaning and values of DevOps and its importance in software development, while explaining its origins, capabilities and toolchain. The first part goes through the DevOps’ theory and its roots, defining a possible reference architecture in a guide to DevOps adoption for organizations; the second one analyzes the main tools that can be used to follow the practices explained in the previous section; the third section includes stories of successful adoption of DevOps methodologies.

# Definition of DevOps

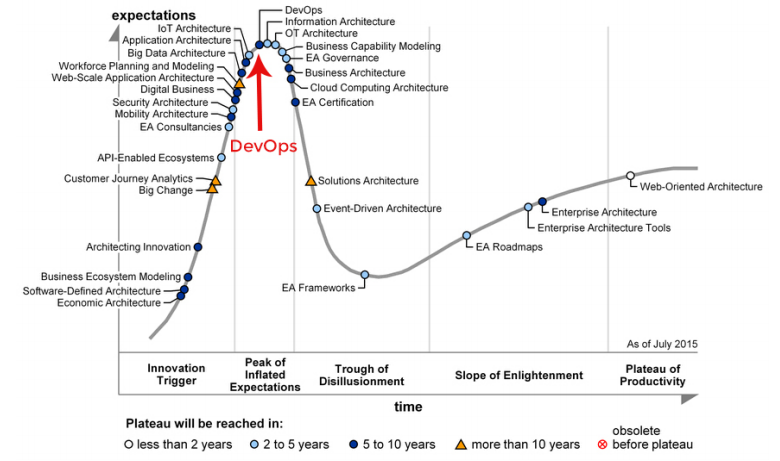
One of the biggest challenges in the adoption of DevOps culture is to understand what it is and what it means. The term itself has become a commonly used word, even though the current academic research is very limited. Following a DevOps approach for a project is known to have benefits, just like known are the many challenges involved in implementing it. Its popularity has been growing incredibly fast, as it was positioned in the Innovation Trigger Phase in 2014 Gartner’s Hype Cycle for Enterprise Architecture, while in last year’s cycle it was at the Peak of Inflated Expectations. A recent study by Rackspace shows that 55% percent of the 700 biggest IT companies have already adopted DevOps, while another 31% of them plan to do the same thing in the next 2 years [1]. There are some excellent examples of companies who have excelled and are at the leading edge of the DevOps movement – the often cited Etsy, Facebook and Netflix. Over this chapter, I will analyze the origins of this movement, and the values and principles lying at its core.

Figure II.1 - 2015 Gartner's Hype Cycle for Enterprise Architecture

## 2.1. The roots of the movement

### 2.1.1. The evolution of the Development process

In the world of software development, the quest for the optimization of project management has always been a very important point. Developers, engineers, and managers have been concerned since the 1960s with maximizing software’s quality while reducing costs and delivery times, trying to come up with “best practices” for writing code as well as other management issues, even on a human relationship level.

By the early ‘80s, software engineer became a profession, besides computer scientist and traditional engineer. As described by Moore’s Law, computers have kept updating and making obsolete older machines with an alarming rate for software engineers and developers, which had to adapt to new technologies, with consequent increase in costs and risks of bugs in the code. Teams of software engineers searched the key for successful development of software project, but in vain.

Software companies were functionally separated organizations, with no cross-departmental integration between development, the software engineering side, and operations, those responsible for building and keeping alive the infrastructure.

This was the period of the first generation of software development methodologies, the *Waterfall Model*, a sequential non-iterative design process, recognized as the oldest standardized way of software engineering. Herbert D. Benington firstly presented it on June 1956 [2], although it was formally described only in 1970 by Winston Royce. In Royce’s original waterfall model [3], six steps are to be followed in the following order:

* System and software requirements, with the writing of a comprehensive product requirements document
* Analysis, and its deriving models, schemas, and business rules
* Design, to build the software architecture
* Implementation, the actual development and integration of the software
* Verification, the systematic discovery of bugs and defects
* Maintenance, deployment of the complete system and its monitoring

The Waterfall Model obliges to move from one phase to the next only upon completion of the previous one. In addition, great importance is given to the early stages of the process. In his 1956 paper, Benington explains how the first steps could take up to 53% of the engineering process, while coding 7% and leaving the rest up for testing. This implies that the Waterfall Model would be very strict about changes in specifications as, upon completion of the analysis and design stages, any change would require a new analysis phase and unexpected rise in costs. Moreover, the application has to be completely coded in order to move to the testing phase, making tracing errors difficult and expensive in terms of time-to-market and value.

Figure II.2 - Waterfall Development Life Cycle

Over the years, engineers realized the problems with the Waterfall Model and started to search for new methods and ideas. They adopted concepts from the automotive industry, such as the Toyota Production System (TPS), applying the *Lean* mindset to software development. Lean was designed to reduce waste and improve efficiency, by eliminating those tasks that proved to be repetitive and not to bring any kind of additional value to the project [4]. Of course, value is seen from the customer point of view: for this reason, waste can mean excess documentation, features not explicitly asked by the customer, or excess development time. Lean thinking faced the problem of Waterfall’s static nature with the approach of deciding as late as possible when faced with options, in order to make decisions based on evolving facts rather than assumptions and hypothesis, hence including the capabilities for change. Another Lean key value is to deliver as fast as possible, generating a faster feedback loop as a better mean for improving the product.

Lean software development can be considered the basis for the second generation of software development method, *Agile programming*. The *Agile Manifesto* [5] gave birth to a new movement, with its values and principles to guide a people-centric approach to software development.

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

|  |  |  |
| --- | --- | --- |
| *Kent Beck* | *Mike Beedle* | *Arie van Bennekum* |
| *Alistair Cockburn* | *Ward Cunningham* | *Martin Fowler* |
| *James Grenning* | *Jim Highsmith* | *Andrew Hunt* |
| *Ron Jeffries* | *Jon Kern* | *Brian Marick* |
| *Robert C. Martin* | *Steve Mellor* | *Ken Schwaber* |
| *Jeff Sutherland* | *Dave Thomas* |  |

Agile main concept is to let the developers build software incrementally, in smaller batches of code that are integrated in to the main project constantly, and obtain the customer feedback as a form of collaboration with the customer itself. With this iterative process, the developers build the software exactly the way the customers want it, without spending time in unnecessary features and solving bugs as soon as customers find them. In addition, as requirements keep changing during the development process, continuous integration becomes extremely useful for fully adopting the previously mentioned “decide as late as possible” philosophy from Lean thinking.

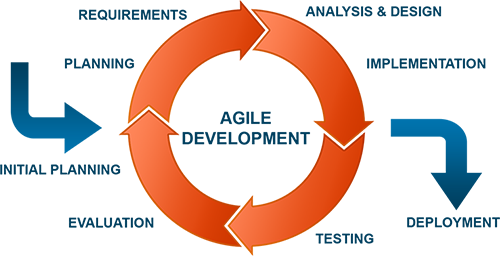
Agile software development is a philosophy rather than a standard set of practices. There are several detailed processes that apply Agile methods, the most popular among them being Scrum and Kanban. Scrum allows the organization of the workflow through *sprints*, which are short development period that typically lasts between one and four weeks in which features are coded, tested and integrated into the existing product. In order to track progress and create the plan for the next 24 hours, Scrum teams hold daily sessions called “Scrums” in which they decide the *item* for the next sprint, which is the objective they need to pursue. Kanban focuses on the visualization of the workflow through cards attached to a board (the term *Kanban* in Japanese stands for “signal board”) with different columns for different stages of the process: a typical board has stages such as Backlog, To-Do, In Progress and Completed. Both Scrum items and Kanban cards are called *stories*, which are prioritized according to their perceived value at each point of time, allowing agility of the team. The main differences between Scrum and Kanban are summarized in the following table.

Figure II.3 - Agile Development Life Cycle

|  |  |
| --- | --- |
| Scrum | Kanban |
| Work in sprints, of 1-4 weeks. Each sprint should end with a shippable product. | No fixed-length sprints, teams pull tasks from a prioritized backlog of things to be done. |
| Products is released on a particular cadence, determined by sprints’ length. | Releases occur continuously, or whenever the product is shippable. |
| Teams have no specified roles, low hierarchy. Everyone is a “marketer”. | Team members can be specialized and pull tasks related to their area of expertise. |
| Regular meetings, daily sessions, sprint reviews and sprint retrospectives. | No standardized regular meetings, emphasis on continually improving processes. |

Table II.1 - Differences between Scrum and Kanban

### 2.1.2. The static nature of IT Operations

IT Operations are the set of all processes and services provisioned by an IT staff for the functioning of the infrastructure and the production environment. They are responsible for maintaining and assuring the uptime of said environment, administrating a constantly growing number of servers as the company’s services scale. It could be said that the less changes are introduced, the easier is ensuring the availability of the production environment and its services. Operations teams focus on achieving *stability* over *agility* to reduce service disruptions, and it is their responsibility to resolve any failure or outage of service.

IT Operations Management tasks can be divided, as defined by ITIL (Information Technology Infrastructure Library), into two main areas, each with his own set of related activities [6]. *IT Operations Control* comprehend functions concerned with the monitoring and control of the virtual IT infrastructure, such as console management to monitor the systems, job scheduling for triggering automatically integrations, cleaning operations and other batch jobs, backup and restore management and any other maintenance activities. *Facility management* takes care of the physical IT infrastructure such as server rooms and data centers, providing the physical capabilities for networking, scaling and disaster recovering. This second area of functions is slowly getting integrated in the first one, as cloud computing and service virtualization are replacing traditional horizontal scalability. This concept will be analyzed later in this work.

As Development kept evolving and adopting new technologies, it would be predictable for Operations to have changed with times. IT Operations have been scaling infrastructures for decades, both vertically and horizontally, respectively upgrading the existing hardware and setting up new components. However, modern tools are not exactly designed for fast scalable infrastructures and are not as effective when administrating large volume of servers. Generally, servers, storage, network as well as application environments are built and configured in a semi-automatically, but some of the configurations and the deployment of the applications still is manual. This affects the delivery time, as code from Development usually needs some work on it to fit the production environment because of smaller or bigger differences from the development one. Hence, code deployments need to be scheduled, resulting in a slowed application life cycle and discontent of both Development and the customer.

## 2.2. Challenges faced

From the current state of the roles division in the IT world, four aspects need to be improved by the new philosophy of application development and delivery.

##### The wall of confusion and the Blame Game

Having Development and Operations separated in their own teams led to misunderstanding between the two, resulting in extra work for both teams and costs. At times, teams are even reluctant in sharing knowledge and a common goal because of the fear of losing power, reputation and influence. Misunderstandings and time pressure always lead to the “Blame Game” [7]. A common scenario in an IT company is the following. Developers release a feature that worked well in their test environment, however it fails to run in production. They blame Operations for poor implementation and out-of-date infrastructure, while Ops point their fingers towards Development for bad design. Often the root cause of this problem is a difference between configuration or some other minor detail.

##### Misalignment of goals

Development teams have adopted Agile methodologies to respond fast to changing requirements and technologies, while increasing collaboration and transparencies within the team. Their main goal is to deliver an increasing numbers of features at an increasing speed. On the other hand, Operations have been left out, as they need to deliver the features produced by Development with an additional nonfunctional requirement of service stability. Operations are reluctant to deploy at the pace of Development to ensure stability, widening the gap between completed functions and planned release dates. In a nutshell, while Development is expected to bring in as much *change* as possible, Operations is tasked with assuring *stability*, leading to conflict.

##### Contemporary feedback

Lean thinking introduced the concept of eliminating waste, meaning that features not directly needed by the customer should not be implemented. Sometimes decisions are based on out-of-date or incomplete data due to a lack of contemporary feedback. There is a need to implement a working feedback loop between development and the customer, to base feature development decisions on.

##### Management of Production Environment and Migration

The setting up and deployment of new environments require a lot of configuration and introduce complexity and risks of failures. The complexity of running environments makes impossible the decommission of systems no longer in use, due to their role in a fragile network where their removal can cause catastrophic consequences.

## 2.3. The DevOps philosophy

DevOps was born to address specifically these inefficiencies of modern IT industry. Although technology has been constantly innovating, the delivery of features and solutions has been lagging behind due to inefficient project management, missing deadlines, causing service outages, hence costing more than anticipated. The division of the work environment and of the roles caused a lot of “finger-pointing” between Development and Operations teams, one accusing the other for bugs and downtimes as if they were “throwing” the code from one side of a fictional wall to another, and the blame for any faults that come with it.

DevOps was formed out of a fundamental need and it is based on a simple yet powerful philosophy: business works best when efforts are coordinated and when they are based on collaboration. It promotes a set of processes and methods for a better communication and collaboration between the three traditional sections of an IT company, which are Development, Quality Assurance, and IT Operations.

### 2.3.1. A short history for a big movement

The roots of DevOps can be traced back to Belgium, 2007, where a consultant named Patrick Debois had to work for the public administration for a data migration. As he was in charge for testing, he had to struggle between the various Dev and Ops groups that were working on the project. In 2008, at an Agile Conference in Toronto he brought up this problem, stating that the conflict between these team have been causing delays and underperformance in the projects. As a result, the Agile System Administration Group was formed, although the problem still did not have much relevance in the Agile community.

In June 2009 at the O’Reilly Velocity conference in San Jose, the first DevOps speech was given by Flickr employees, John Allspaw and Paul Hammond, titled “10+ deploys per Day: Dev and Ops Cooperation at Flickr”. Patrick and other IT professionals were passionate about the speech, and were inspired to organize a conference called “Devopsdays”, later the same year in Belgium. Since then the movement started to be referred as “DevOps”, also thanks to the circulating hashtag on Twitter, *#Devops*, with the social network as the main source for information and place for conversation. Follow-ups to the Devopsdays conference were held the next year in Sidney and in Mountain View. From this moment in time, the culture of DevOps started spreading online, with professionals sharing their view on the state of IT and on the changes that were in need in their blogs and pages. People even started making their own song parodies and music videos about the topic, but most importantly realized the deficiencies of the tools that they had at their disposal at the time.

This led the DevOps community to develop a whole new generation of tools to formalize the new practices and ideas that they were preaching. In 2011 the first tool to create and configure virtual deployment environment, called Vagrant, was born. Only two years later, DevOps had started to formulate a coherent and standardized practices, with the birth of the first literature on the topic such as “What is DevOps” [8], “Implementing Lean Software Development” [4], “DevOps for Dummies” [9], and even a business fiction novel called “The Phoenix Project” [10]. All of them have been used as reference for this work.

If the first objective was that of tearing down the wall between Development and Operations, the movement has now expanded to include the whole development life cycle. This includes "automation from the beginning of the process through the deployment of the solution in a system of incremental builds" [11].

Currently, organizations have started reaping the benefits of this new approach, with several well-known organizations such as Netflix, Facebook, and Amazon.

### 2.3.2. DevOps Values and Principles

#### 2.3.2.1. Business value

Any organization is always reluctant to changing the “usual way” for doing business.

For a company to change technology, methodology, or to simply adopt a new approach, it is always hard and usually requires investments. This shift has to be driven by a business need, and pointed towards a business value. DevOps is proven to have a huge business value, giving the companies adopting it an edge over competitors.

Every organization wants to create new applications or services to solve business problems, whether they want to address internal business problems or to help their customers or end-users. Whatever the case, the organization is looking for a way to make what they build *better*.

“Better” can mean many things. For a startup, it could mean a higher rate of features release, without quality being the highest priority. For a financial institution, for example, it could mean minimizing downtimes and outages of service. Another organization may have ensuring 100% reliability of their systems as priority. It is necessary for all actors of a development process to share the same objective and to follow the same direction, so that business requirements are defined before starting the solution development.

DevOps applies agile and lean principles across the entire software supply chain, enabling a business to maximize the speed of its delivery, and to enhance the product based on the feedback received from customers. Faster time to value, increased speed of changes and enhanced customer experience are the three areas of ROI (Return on investment) of DevOps. Through the principles that will be analyzed later, DevOps also allow a better stability of the operating environments, and the reduction of service outages due to errors in code and/or configurations. More deploys and more uptime, means faster time-to-market and continual improvement, which translate in more value.

#### 2.3.2.2. Understanding DevOps principles

DevOps’ aim is to reduce the gap between the three antagonizing entities of any IT company, in particular between Development, working with an agile mindset, and Operations, which wants stability. Although with different tools and mindset, everyone has the same goal – the happiness of the customer. DevOps brings forth the change of mindset necessary for the two to work together, while thinking more alike and sharing responsibilities. It connects development and production environment by ensuring that the development changes are being tested and deployed in a way that is efficient and does not interrupt ongoing operations.

It can be said that DevOps is a software development approach that synthesizes development and operations to enable *agility*, as a critical element to reduce complexity, aid error diagnosis and eliminate conflicts and narrow-mindedness. Gartner refers to DevOps as “IT service delivery approach rooted in agile philosophy with an emphasis on business outcomes, not business orthodoxy”.

This lack of orthodoxy led to the actual lack of an official standard for a DevOps approach, which is now just a set of guiding principles, making it flexible and non-rigid in its interpretation and implementation.

DevOps can be described through the CALMS-model [12]

* **C for Culture**: the quest for building a collaborative culture with the focus on people, in order to reduce blame and build productive teams
* **A for Automation**: automating the deployment process and infrastructure building, reducing bottlenecks and increasing speed and efficiency via continuous delivery and the use of Infrastructure as Code
* **L for Lean:** applying Lean principles to software development, such as Kanban boards or Scrum sprints, to eliminate waste, incentive continuous improvement and limit work in progress by working on fast cycles of small batches of code
* **M for Measurement:** using new tools to adopt an iterative process to monitor and improve code and operations
* **S for Sharing:** sharing knowledge across teams to increase collaboration, preventing fragmentation of systems and practices

These areas also describe DevOps as a “flow” [13].

Figure II.4 - The CALMS model as a flow

##### Culture

At its core, DevOps is a philosophy. Although the tangible part of DevOps consists of tools and processes for continuous delivery and integration, an essential part is changing the organizational culture from a collection of groups into a collaborative bigger team. Ensuring that each individual is part of the entire solution life cycle covering development and operations is one key concept. This can be achieved involving Operations personnel in the design and transition process of an application, breaking the “wall of confusion”. They should attend necessary planning meetings of project teams in order to share their insights and knowledge already early on in the process. The organization of the project should shift from project- and responsibility-oriented teams into a bigger product team [14], as everybody is now responsible for the entire life cycle of an application or service, in a “from-development-to-operations” supply chain. Such a product team might consist of developers, operators and quality assurance personnel. People taking part in this delivery chain must share similar objectives and targets, and valued for their potential impact on the solution life cycle.

Collaboration and the introduction of new tools to the workflow decrease the need for control over the teams, as the shift to autonomous teams involves building trust between managers and personnel, along with changing the way risk is managed. Building a DevOps culture requires the leaders of the organization to work with their teams to create an environment and culture of collaboration and sharing. Leaders must remove any self-imposed barriers to cooperation. Sometimes, building a DevOps culture requires people to change. Adopting DevOps, is not just about adopting a product or a process. It is about undergoing Transformational Change [9].

##### Automation

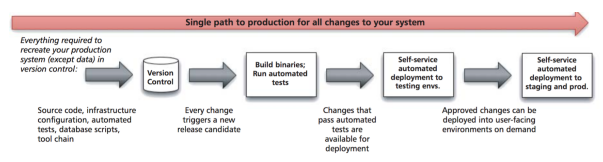
Once the culture has been spread, it is the moment to move to the adoption of new tools. Automation is essential to create processes that are iterative, frequent, repeatable, and reliable, for this reason the organization must create a *delivery pipeline* that allows for continuous and automated deployment and testing. The goal of the new automated processes is to achieve shorter life cycles and rapid feedback loop, all of this while removing manual, error-prone tasks from the production pipeline and allowing people to concentrate on the quality of the product. Between steps of the pipeline, automated tests are executed on each change on the whole product, to find and solve potential bugs. Once the change passes all tests, it moves to the next step in the pipeline, which ends with the actual deployment in the production environment. A more detailed view on the delivery pipeline and its automation will be analyzed in later sections of this work.

Figure . - Automated deployment pipeline

##### Lean

The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency.

As this quote from Bill Gates states, the automation of a process is meaningless if the process itself is inefficient. Lean thinking is fundamental in a DevOps environment, where Work-in-Progress needs to be as limited as possible, while changes should be small and continuous the enable agility. Therefore, Scrum and Kanban can be continued to be applied as they were previously. Automation keeps waste to a minimum and reduces repetitive tasks; productivity is increased and work results more enjoyable.

##### Measurement

Measurement is defined as “monitoring high-level business metrics such as revenue or end-to-end transactions per unit time” [14]. It affects the way people work, as visualizing metrics shows how well the team is doing at any given point in time. Moreover, it helps discovering bottlenecks in the process which could slow down the production line. Organizations typically are good at monitoring applications and systems in production because they have tools that capture production systems’ metrics in real time. However, the DevOps approach requires automated testing to be done early and often in the life cycle to monitor functional and non-functional characteristics of the application. In the key of collaboration and sharing, these metrics should be in a format understandable and usable by all business stakeholders, among which team members, managers, and customers [9].

##### Sharing

The final component of the CALMS Model is Sharing. Sharing is at the same time key to and effect of the new culture of collaboration. Introducing version control in every aspect of the production pipeline and sharing code repositories across team helps aligning teams’ knowledge about the project. The team should share successes, responsibilities and pain, as it is in the whole team’s interest to make deployment a better experience for everybody.

A common deployment pipeline and a shared monitoring systems for all application instances further increase intra-team transparency and collaboration, on every level of the system.

#### 2.3.2.3. The three ways

Gene Kim, one of the authors of the “DevOps Cookbook” [15] and the novel “The Phoenix Project: A novel about IT, DevOps, and Helping your Business Win” [10], introduce a new model to describe the values and philosophies that frame the processes, procedures, practices of DevOps, as well as the prescriptive steps, called “The Three Ways” [16].

The *first way*, “Systems Thinking”, refers to the emphasis DevOps poses on the entire system, as opposed to the function-based separation of traditional organization, of course that being Development and IT Operations. It begins with the identification of requirements by the business or IT, the building of the related solution or feature by Development, and its transition into IT Operations to deliver the service to the customer. Managing this flow allows for global optimization and bottleneck reduction.

To make the first way effective, a few steps are needed [17]:

* Increase “velocity” by accelerating each of the process components in the pipeline
* Decrease “variation” by eliminating wasteful or time consuming sub processes in the pipeline
* Visualize and understand better the global flow – seeing the system

Figure II.6 - The "First Way" of DevOps: Systems Thinking

The *second way*, “Amplify Feedback Loops”, is about creating “right-to-left” feedback loops. As new requirements, bug fixes or completely new services are deployed, the process can improve only through the feedback received by customers and end-users, so that the team can make the necessary corrections fast and continuously. Following Lean Thinking principles, the earlier a defect is discovered, the less costly it will be to the final cost of the service delivery. As for the “First Way”, velocity of corrections and variation of infrastructure play a key role in tightening the feedback loop. A later section will go through the steps, or “vectors”, to enhance the feedback loop.

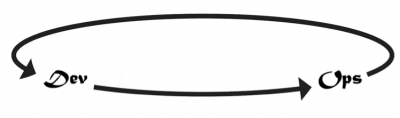
The *third way*, “Culture of Continual Experimentation and Learning”, is self-explanatory. DevOps is all about the creation of a new culture of taking risks and learning from failures while understanding that repetition and practice is the prerequisite to mastery [16].

Figure II.7 - The "Second Way" of DevOps: Amplify Feedback Loops

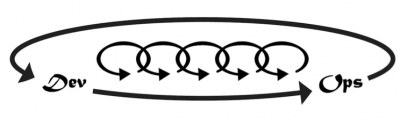
Experimentation means going out of one’s comfort zone, trying new things that could potentially help the process and the product or sometimes put it in danger. But it’s only through this process, and recovering from the possible issues generated that the team itself can improve and master the skills needed. A team adopting a DevOps approach should include time allocated for the improvement of daily work, rituals that reward the team for taking risks, occasional faults in the system to enhance the skill to recover from disasters.

Figure II.8 - The "Third Way" of DevOps: Culture of Continual Experimentation and Learning

## 2.4. Architectural components

### 2.4.1. The need for a reference

Although DevOps is born as a movement and a culture of collaboration, a set of rules and practices is needed for organizations to follow in order to improve teams and processes, and through them the value of the service offered itself. Organizations are generally aware of the DevOps methodology and its value, but they either don’t know how to start or have failed at to adopt it and reap its fruits. This could have been caused, among other things, due to the absence of a *reference architecture*, which a template of proven solution by using a set of preferred methods and capabilities. It helps practitioners to access and use the guidelines, directives, and other material needed to architect or design a platform according to the chosen methodology.

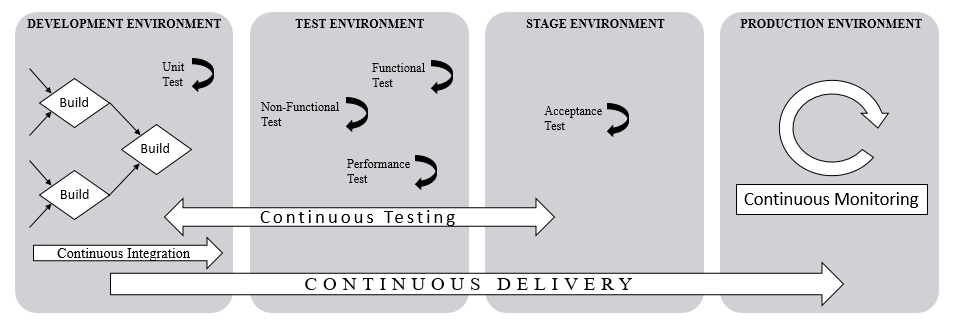
As stated in previous chapters, DevOps lacks an orthodoxy for rules and practices, which makes almost impossible to create a reference architecture. However, having used the two models, the CALMS and the Three Ways, as reference to grasp the main aspects of this philosophy, it is possible to determine a preferred course of action for an organization willing to adopt DevOps as development methodology.

Figure . - DevOps Reference Architecture

In this work, the above diagram is proposed as a reference for businesses to follow in order to complement the previously described DevOps culture, and achieve the goals discussed as business value. This chapter’s goal is to guide step-by-step the reader, ideally a Chief Information Technology Officer, that wishes to adopt DevOps to improve the production line and gain market share.

### 2.4.2. Culture Adoption

In the CALMS model, as well as with the “Third Way” of the Three Ways Model, one of the peculiarities of DevOps is undoubtedly its focus on the necessity of a new culture. This should be the first step in the adoption of DevOps, the underlying element of the diagram of reference architecture. Success in the adoption stage lies in understanding and acknowledging that the business side must lead the transition, joining a team first-hand in which collaboration is the main value. [18]

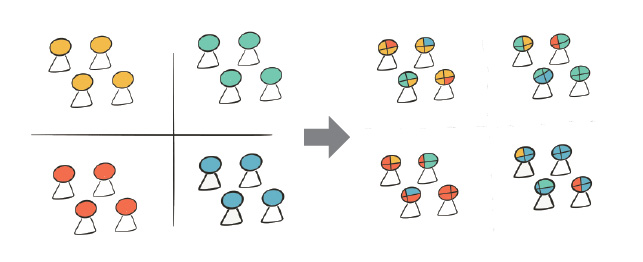
In the “Business Value” section of the “DevOps Values and Principles” chapter of this work, emphasis was put on the reluctance to change of organizations’ upper levels, unless driven by a business need towards a business goal. It is important to establish a common set of goals and shared objectives between all stakeholders of the production process, including development, operators, testers, managers and marketers. Teams should not be segregated in their “siloes”, according to their functions and role in the delivery process, but be integrated in a new “production team”, covering all steps in the production process. This means taking down the traditional “wall” between Dev and Ops, in the name of collaboration and value.

Figure II.10 - Cultural Change of Team's Organization

### 2.4.3. End-to-End Value Production with Continuous Deployment

Once the new teams are formed, and goals and requirements are established, it is time to move concerns towards production. There are three main processes in an organization regarding production: development, deployment, and feedback. Every organization has his own approach to each of these processes, and their definition and interaction define the architecture of the software life cycle.

The delivery pipeline is the core of the team itself, as it describes the steps that are to be followed to make the team more productive and efficient. Pipelining the process of delivery means dividing it in consecutive stages:

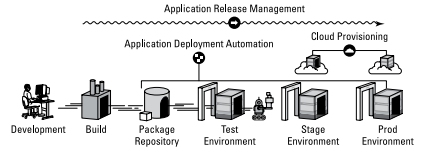
1. **Plan:** building a roadmap of the whole process of production and defining the backlog queue
2. **Code:** development of the solution and its *check in* in version control
3. **Test:** tests for software quality, like Unit, Integration, Functional and Cross-Functional tests
4. **Release:** release of the software to the production environment where it produces value
5. **Monitor:** monitoring of the production environment for bugs through log analysis and feedback collection from customers

Figure II.11 - Delivery Pipeline Overview [9]

An efficient pipeline should therefore integrate quality tests that provide actions for the team, through quick and effective feedback, it should require as minimal as possible manual interaction, and the application should be tested manually only when it is necessary and after is has passed all the automated stages in an environment as similar to production as possible.

DevOps practices allow the team to use a robust, end-to-end, automated pipeline to commit often while assuring quality. [19] Defining and visualizing the pipeline is key for the “First Way” of DevOps, explained in previous sections, because it allows to see systems as whole, knowing the capabilities and limits of the entire production process in order to improve the final product quality.

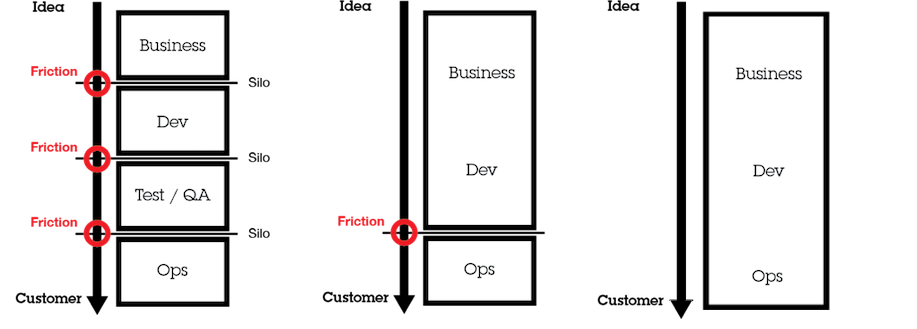
*Continuous Deployment* (CD) is at the core of DevOps adoption. It is a software strategy that enables organizations to deliver new features to users as fast and efficiently as possible. Jez Humble define CD as “the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way” [20].It can also be considered as an evolution of Waterfall and Agile software strategies, in order to have a frictionless delivery of the feature or service requested to the customer. [21]

Figure II.12 - Evolution towards Continuous Deployment (from left to right: Waterfall methodology, Agile methodology, DevOps methodology)

Continuous Deployment is a practice by which software is built and deployed so that it can be released into production at any given time. It allows a conspicuous shortening of various cycle times in the development and operations process, as well as costs and risks reduction. An efficient CD pipeline should include a few steps to guarantee that the software would be deployable throughout its life cycle, with team prioritizing this aspect over working on new features, along with automation whenever possible. These desired features can be achieved with *configuration management*, *continuous integration*, and *continuous testing*. The tools that can enable these will be studied in a later section of this work.

#### 2.4.3.1. Continuous Integration

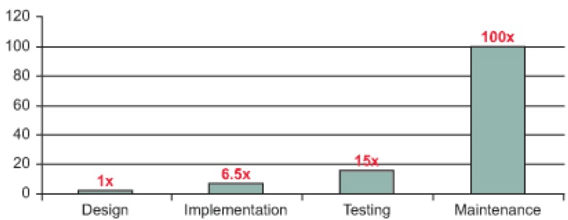
The development process reached his peak with Agile practices at the beginning of this century. It allowed to face fast-changing requirements by developing and releasing application functionalities in small proportional batches, avoiding useless excessive ahead planning or documentation. Agile focuses on small iterations and cooperation with the customer throughout the project, whose feedback is fundamental to plan changes and sprints.

The introduction of *Continuous Integration* tools and practices is key to the development, as it helps preventing integration problems, sometimes referred to as “integration hell”. With CI is possible to integrate, build and test code automatically within the development environment. Source code is versioned in a single place, called *repository*, and shared through team members with version control software. They can obtain, “*check out*”, the latest or previous versions of the code from the repository, and integrate, “*commit*”, the modules they have been working on after the automated phase of tests. CI means having an automated process build *continuously* as developers commit code. A later automated process is supposed to install it, “*deploy*”, on test or production system, in order to save time and reduce errors due to manual operations in compiling or installing the introduced changes. Continuous integration and deployment automation are key component of a well-functioning continuous delivery pipeline.

#### 2.4.3.2. Continuous testing

In this stage, the new version of an application is rigorously tested to ensure that it meets all desired system qualities and that all requirements are fulfilled. Traditionally, extensive use was made of manual inspection of code changes and manual testing in order to demonstrate the correctness of the system. This type of testing was normally done in a phase following “dev complete”, which means that tests were run only once the process of development had reached its final stages. However, regression tests proved to be bottlenecks to fast software release, causing developers to get feedback weeks or months after testing of the code. Moreover, manual tests and inspection are prone to errors, as people tend to fail at performing repetitive tasks. Finally, relative costs to fix software defects at later stages in the development production line are higher, as shown by the following diagram [22].

Figure II.13 - Relative costs to fix software defects (Source: IBM Systems Sciences Institute)

In DevOps, desired characteristics of testing are *continuity* and *left shift*.

The goal of *continuous testing* is to provide fast and continuous feedback regarding the level of business risk in the latest build or release candidate [23]. It includes both functional testing, such as Unit, API, integration and system tests, and non-functional testing, with practices like static code analysis, security and performance tests, etc.

Tests should be designed to provide earliest possible detection (or prevention) of the risks that are most critical for the organization releasing the software. By *left shifting* these phases, which means moving them to the left of the delivery pipeline, bugs and issues can be found soon after they are introduced in order to avoid them from getting further in the software development life cycle. Being continuous, it also implies that software need to pass a test phase to move further in the delivery pipeline. This ensures that quality is built-in as early in the life cycle as possible. For teams practicing continuous delivery, tests are commonly executed many times a day, every time that the application is updated into the version control system [24].

The use of production-like environment can ensure better accuracy and consistency of the tests on the system. The practices of *configuration management* fit perfectly with this need, as will be explained more in detail in the next segment.

#### 2.4.3.3. Configuration Management

Software can be released frequently and reliably only through “*jidoka*”, a word from the Toyota tradition sometimes translated as “automation with a human touch”. This means that computers should perform simple, repetitive tasks, such as build, deployment, regression testing, and infrastructure provisioning, so that humans can focus on problem-solving, as if humans and machines were to complement each other [20].

And that is where *Configuration Management* ties in. It is defined by Puppet Labs as “the process of standardizing resource configurations and enforcing their state across IT infrastructure in an automated yet agile manner”. In a nutshell, server configuration management, also referred to as IT Automation, is a solution for turning infrastructure administration into a codebase, describing all processes necessary for deploying a server in an asset of provisioning scripts that can be versioned and easily reused. Version control is applied to everything required to perform these processes – source code, test and deployment scripts, infrastructure and application configuration information, dependencies management. The main problem lies with infrastructure version control, as traditionally it is not something that can be automated due to its manual processes.

However, with the virtualization of servers and with the spread of cloud computing, *Infrastructure as Code* (IAC) has come to life. IAC is a type of IT infrastructure that can be automatically managed and provisioned through code, rather than using manual processes. IAC is sometimes referred to as *programmable infrastructure*, as it uses higher level or descriptive language to “code” an adaptive provisioning and deployment process. [25] This is one step further from writing scripts, as it involves tested and proven software development practices, and it is different from infrastructure automation, which just involves replication of steps multiple times on several servers. IAC allows developers to engage in tasks that were usually Operations concerns, because of the not very steep learning curve of the related languages and tools and the easy APIs provided by server hosting sites.

The ability to use version control on infrastructure code implies that it becomes easy to track all the changes in infrastructure environment and to roll back a previously working configuration in case of problems. Because most tools provide virtualization of servers similar to those in the production environment, it becomes quicker to test software and the issues related to the deployment to different machines tend to disappear.

Despise its usefulness, IAC requires a lot of planning before the actual configuration, such as the adoption of the right tools and learning the related descriptive language; it also causes bad configurations to get duplicated on all the servers if not checked before deployment.

Although the use of configuration management typically requires more initial planning and effort than manual system administration, many benefits are provided to any level of server infrastructure. CM tools can automate most, if not all, the provisioning process of a new server when needed. Every repetitive task related to server deployment can be performed fast and accurately, and can even be automatically triggered if monitoring tools have been setup to do so. Moreover, quick provisioning also means quick recovery from critical events, as a replacement server can be automatically deployed whenever another goes offline for whatever reason. Documentation of provisioning scripts implies the possibility for version control of the server environment, along with tools and workflows normally used for software source code.

In conclusion, configuration management can drastically improve the integrity of servers over time by providing a framework for automating processes and keeping track of changes made to the system environment. Implementation strategies and tools will be analyzed in a later section.

### 2.4.4. The importance of continuous monitoring and feedback

As organizations deploy more and more rapidly changes to their production environment, whether it is a bug fix, a new requirement or a completely new service, the importance of rapid feedback grows drastically, as the best way to keep improving the product itself and thus customers’ satisfaction. Development can adjust its project plans or priorities, production may enhance the environment, and business can modify the release plans.

*Continuous monitoring* provides data and metrics to the production team and to the other stakeholders about the application at different stages of the delivery cycle, as well as the production environment itself. Operations teams should use tools that can monitor application performance and issues. Where necessary, it may be required for Ops to work with Development to build self-monitoring or analytics gathering capabilities right into the application being built, enabling true continuous *end-to-end* monitoring.

A tight feedback loop is essential for the software delivery process. If testing processes and feedback analysis take longer cycles, the organization has to keep working on the next sprint, resulting in rework if the previous code had bugs. It provides rapid response to problems or new requirements, encouraging faster experimentation and tolerance for failure. Fast feedback also helps connect the production team to user experiences, while fostering collaboration between previous groups within the team.

Feedback loops in smaller organizations tend to be relatively simple, as it only needs for interested counterparts to talk to each other or the programming of alerts in the monitoring system. However, larger organizations encounter many difficulties due to the multiplicity of teams and the volume of events [26].

In larger organization, we could analyze two different “actors”: a “front office”, including the production environment and the customers, and a “back office”, with the development environment and a support/development group. Therefore, each “actor” is made of two components, a “machine” or “environment”, with the product and the code, and a “person” or “group”, using or making the product. It would be possible to connect these four elements with four “vectors”, four lines indicating the feedback direction and the elements each relates to [26]. An illustration of the mentioned system would be the following.

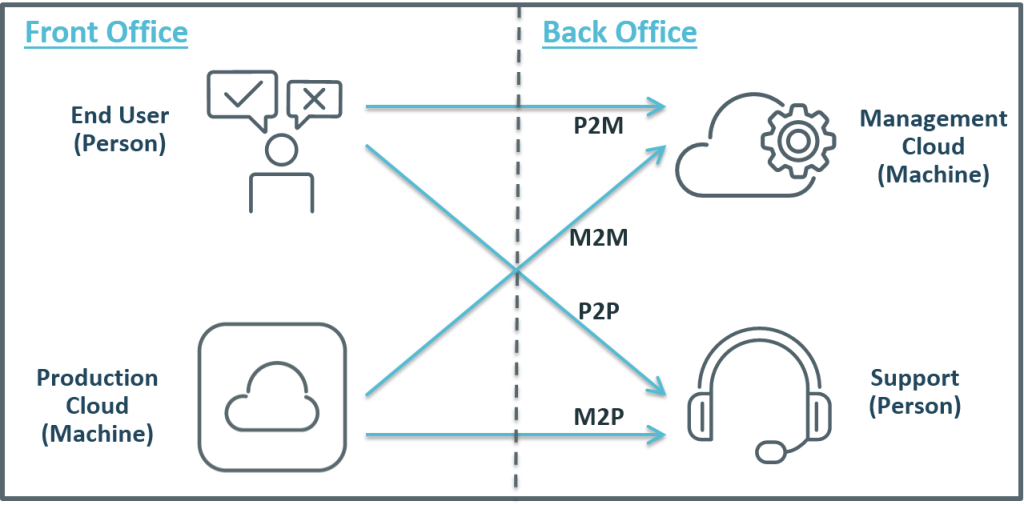
These four vectors are:

Figure II.14 - Four Vectors of the DevOps Feedback System

* **Machine to Person (M2P):**

A machine process such as performance monitoring picks up an error situation currently unresolvable and forwards it to a person of the production team. This helps the team solving issues that cannot be solved via intelligent orchestration – release rollback, capacity increase, configuration modification, service restart – and that have appeared only in the production environment.

* **Person to Person (P2P):** end-users talk directly to someone in IT, most commonly reallocated Operators, and communicate bugs and features requested. It can help identify defects and opportunities for improvement, and can even be used as basis for future release or sprint planning.
* **Machine to Machine (M2M):** this includes automated problem resolution, such as detecting bottlenecks both in capacity and in features implemented, but also includes data mining useful to execute more realistic tests and new QA environments, it helps to reduce human and process errors and to enable scalability as early as possible in the software development life cycle.
* **Person to Machine (P2M): t**hrough ratings of application, online complaint, and self-service portals for help, P2M feedback can trigger automated resolution for known problems, or loop the found problem to someone in the production team.

These four vectors can be used as an effective starting point, connecting stakeholders and giving value to user experience. It is not to be forgotten that an IT organization’s purpose should be to get value by building solutions to the final user’s problems. The happier the customer, the bigger the value for the company.

This feedback mechanism enables a key feature of DevOps culture, *continuous improvement* [27]. As the application is continually delivered, customers can use the application and provide a feedback useful to improve both the application itself and the environment it is delivered on, in the next iteration. Applications’ features can be enhanced, added or removed, based on the feedback, while the environment can be scaled or re-configured if the performance it allows is not up to what decided in Service Level Agreements (SLAs). Fast feedback also allows an organization to improve the delivery process itself, by applying Lean principles (*Kaizen* is a key word here) to reduce wastes and repetitions, and eliminate steps in which there is no value production. This can be achieved by mapping the delivery pipeline and identifying its bottlenecks. Those are then prioritized and the most critical are to be addressed as soon as possible through DevOps principles and capabilities. Most commonly, an “improvement team” is set to point out these limitations by observing the whole process, to then communicate them to the management team.

# The DevOps tool rack

Cultural and process changes, as described in the previous chapter, are fundamental to DevOps success. Developing and implementing a culture of collaboration and continuous improvement helps organizations to get faster innovation, accelerated time to market, improved deployment quality, better operational efficiency, and more time to focus on core business goals. A fast and tight feedback loop allows the customer to feel closer to the development process, resulting in its increased satisfaction.

DevOps is about people and processes as much as – if not more than – tools. “Software-based tools are not the core of DevOps, but merely DevOps enablers”. Supposedly “must-have” technologies are designed to support those definitive aspects of DevOps: collaboration, breaking down silos, bringing Dev and Ops together, agile development, continuous delivery and automation. For this reason, in the vast landscape of DevOps tools, there is no fixed perfect solution as a combination of software that can help a team achieve the best results. “What matters is the teams are empowered to choose their own tools” [28].

This chapter can be thought as a more technical view on the “Architectural Component” section of the previous chapter, a companion guide to the theoretical principles discussed in the mentioned section. The tools presented are by no means to be thought as the best in their category: Google Trends and the Stack Overflow forum were used as reference to see which technologies share most followers in the community among those presented in the “DevOps Periodic Table of Tools” by XebiaLabs.

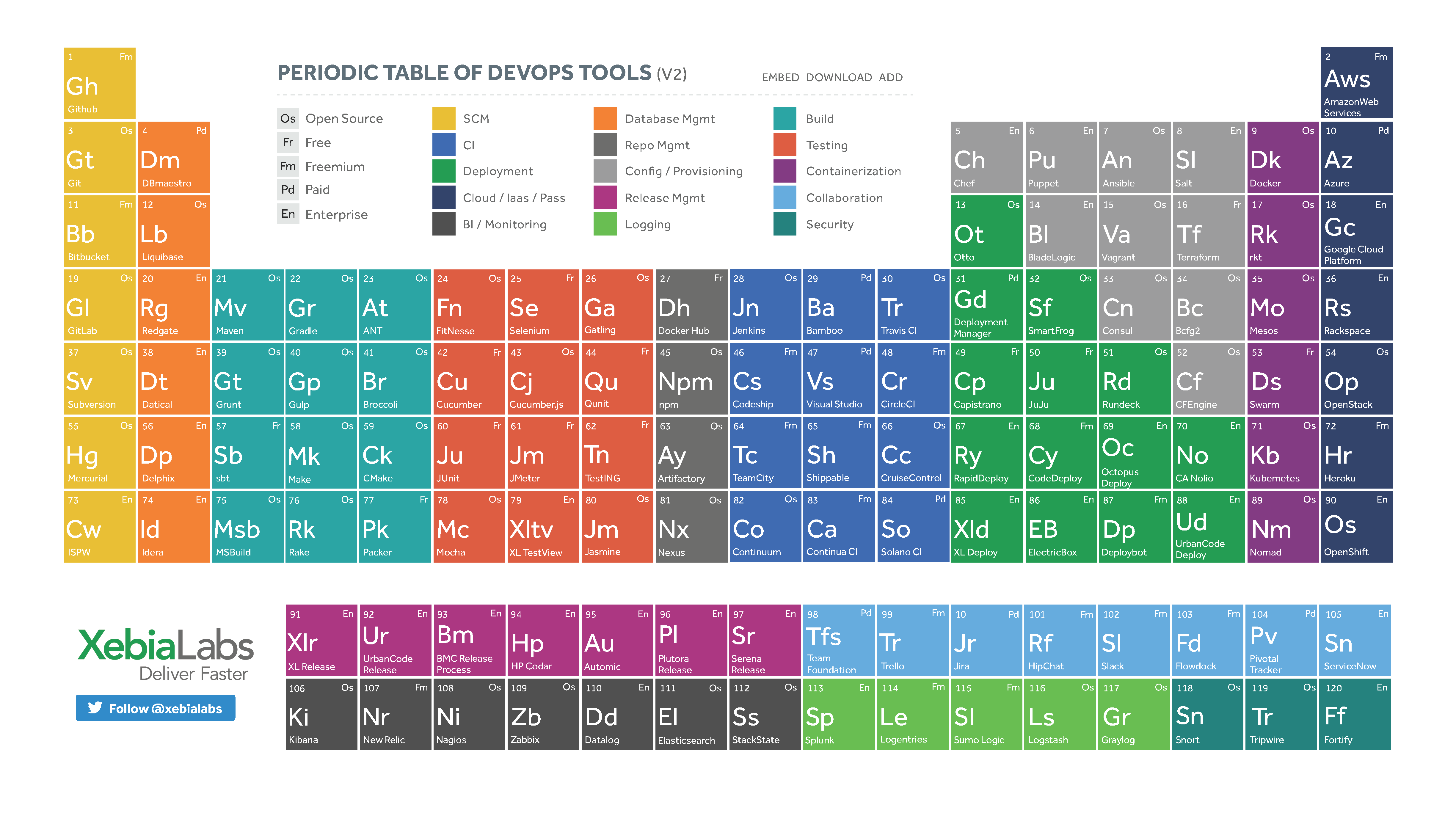


Figure III.1 - DevOps Periodic Table of Tools (credits to XebiaLabs)

## 3.1. Collaboration

### 3.1.1. A Culture of Collaboration

A definition of the DevOps philosophy was born from the C.A.L.M.S.-model [12], as described previously in this work. The first letter of the acronym stands for *Culture*, meaning the sharing of values and team collaboration.

Agile methodology, in its Manifesto, already shifted attention towards “individuals over processes” and “customer collaboration over contract negotiation”. DevOps took this even further, proposing a change in team’s organization and build by breaking down the siloes where their members have worked.

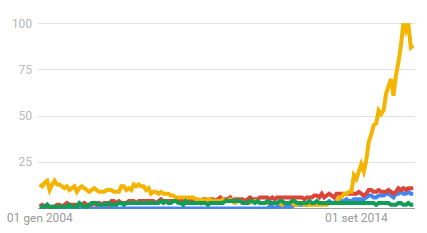
Technology helps uniting members of a team under the same flag, improving their means of communication and their sense of belonging in the team, in the name of collaboration. Two tools will be analyzed in this section: Slack, the most used app for team communication, and Trello, as the best app for workflow visualization.

Figure III.2 - Google Trends for Collaboration Software

#### ***Slack***

Figure . - Slack Logo

Slack is one of the fastest growing start-ups, as can be seen from the Google Trends graph in Figure 3.3. It is now valuated at 3.6 billion and it is one of the most used communication means among team members, changing deeply the way they cooperate.

Slack helps with the cluttering of communication systems by converging it all in one place, and segmenting it in various *channels* for various topics. Users can be assigned to as many channels as needed, according to the desired visibility of the topic itself. Therefore, a worker can join both the company channel to keep up with news and information regarding the organization itself, and its own team’s channel to discuss sprints, share code snippets and files, etc., all in one place

However, chat communication is barely the surface of what Slack is capable of. Slack can be integrated with many web services that are being used every day. GitHub’s plugin notifies commits, Trello’s shows instantly cards as they are being created/edited/completed. Other services integrated in Slack are Dropbox, Google Drive, Heroku, IFTTT, Jenkins CI, Travis CI, New Relic. Most of the tools that will be mentioned in later sections of this work also have their integrations in Slack. For example, Jenkins CI plugin notifies the channel whether a build was successful or not, while New Relic plugin takes application performance monitoring to a whole new level by notifying channel when an alert is triggered.

Last but definitely not least, Slack portal is available from every platform, including mobile, so that no important notification or updates are missed ever.

#### Trello

Figure . - Trello Logo

Trello is the visual collaboration platform released by FogCreek Software and one of the best project management tools, increasing team productivity exponentially when used correctly.

Trello can be defined as the digital version of a Kanban board, as the application allows team members to visualize the workflow and keep track of work activities. The management of a project starts with the creation of a board, and the typical Kanban columns: Backlog, To-Do, In Progress and Done. Each column can be filled with cards, the digital version to Kanban stories, to which details can be added to better define the story. Links, images, files can be attached to a card and be accessed by every user who is given access to the board.

An important feature of Trello’s card system is the “drag & drop” within and through columns. Cards can be rearranged vertically in the same column, placing them in a priority order as they will be completed in roughly top-down order by the development team. When a team or one of its members starts or has finished working on a story, a developer will move the related card to the appropriate column, from left to right.

In conclusion, it is easy to understand the huge role that Trello can have in production teams. It focuses the team’s attention to smaller tasks, it provides a method to visualize the progress of the whole group and identify bottlenecks, it shows whether a task is blocked and communicates why to the rest of the group so that problems can be identified and solved.

### 3.1.2. Source Code Management

The Source Code Management (SCM), also known as Revision Control System (RCS), is a software implementation of revision control that automates the storing, retrieval, logging, identification, and merging of revisions. SCM is extremely useful for large groups of developers that are working on a shared project in order to implement best *continuous integration*, which sometimes use SCM software to also maintain documentation and configuration files. Data can be retrieved by *checking out*, modified locally in a “working copy”, and the *checked in* or *committed*. SCM systems are often centralized, with a single authoritative data store, the *repository*, as check-ins and check-outs are done with reference to this central repository.

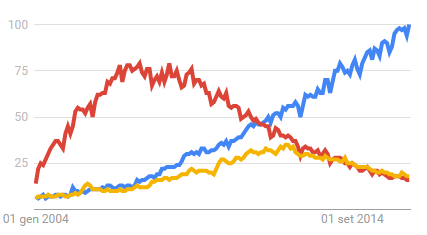
RCS for source code have been used for a long time, first released in 1982 by Walter F. Tichy and is currently maintained by the GNU Project. However, a DevOps peculiarity is the adoption of the same RCS for infrastructure, thanks to *Infrastructure as Code*. This allows for automatic provisioning and deployment.

Figure III.5 - Google Trends for SCM Software

#### GIT & GitHub

Figure III.6 - Git and GitHub's logos

GIT uses a peer-to-peer network of repositories to enable distributed maintenance of digital content. It was built in 2005, as the community working on the Linux kernel needed a VCS that could satisfy their needs. This new VCS should support distributed workflows similar to those of other VCS, offer safeguards against content corruption, and be highly performant. For this reason, the adopted model was that of a distributed VCS, which offers great benefits to the workflow, among which the ability for collaborators to work offline and commit incrementally, the possibility to publish in different branches at whenever the developer felt like sharing his/her work.

GIT is now being used as the main SCM globally. Its adoption has been slightly impacted by the lack of IDE integration, differently from other SCM tools, which made GIT less portable and scarier for new users, not accustomed to the huge variety of commands available and the difficult understanding of some of the error messages.

A web-based repository hosting service, GitHub, offers all of the distributed revision control and source code management functionalities of GIT, as well as its own features like web-based and desktop cross-platform GUIs, access control, bug tracking, feature requests and wikis for every project. Hosting projects on GitHub’s cloud service is offered in the form of either free and public or paid and private repositories, and can be considered as an important step in the possible full virtualization of the architecture of a company. As of April 2016, GitHub reports having more than 14 million users and more than 35 million repositories, making it the largest host of source code in the world.

## **3.2. Continuous Deployment**

### 3.2.1. Continuous Integration

Continuous Integration tools help in supporting the required process for software solution production/factory. This process can encompass checking out the source code from a SCM, compile it, run automate testing, package building and deployment through the related tools. It also reports whether a build was successful or not. Said build process can be triggered either by having the CI system periodically test the SCM for changes, or by having the SCM notify the CI system when new changes are checked in.

Continuous Integration forces developers and teams to integrate their individual work as early as possible, exposing issues and conflicts with a system wide or application wide “Integration build”, preferably on a daily basis [9]. As these issues are found in early stages, no delays are left at the end of the sprint, thus resulting in a shorter cycle and faster time-to-market.

### 3.2.2. Build Automation

Build automation is the process of automating the creation of a software build by compiling source code into binary code and packaging it. Its origins lie in the shell itself, where a script would invoke the compiler, evolving then to more mature forms such as Makefiles, to build automation utilities like Rake, MS Build, Apache Maven, Gradle, etc.

As building incorporated compiling, linking and packaging the code into an executable form, tools allowing the automation of simple, repeatable tasks and doing so in the most efficient way possible through a specific set of tasks are key to the DevOps philosophy, saving time and reducing the risk for human error.

Build automation is considered one of the first steps in moving towards Continuous Delivery and therefore DevOps methodology [29].

### 3.2.3. Automated Deployment

Figure III.7 - Build to Deployment Automation chain

Producing executable code is just the first step of the delivery pipeline. Deployment automation plays a fundamental role in this chain, as it actually means “making an application available to end users” [30]. This process can be broken down in multiple steps, including installing the application, configuring resources and middleware components, starting/stopping components, and, where necessary, configuring the application for different environments. Many build automation tools are not programmed to handle middleware systems, and have no knowledge of concepts such as deployment packages, target environments or customizations [31]. An efficient deployment automation tool should satisfy a different set of requirements, according to Partington:

* Support for concepts central to deployment, like packages, environments, bindings, etc.
* Support for common middleware systems with the possibility to extend the support
* Support for common deployment scenarios with the possibility to customize them
* Support for different roles: developers deliver software, administrators set up environments, ops deploy applications, etc.
* Scalability to large environments
* Cross-platform support: complex middleware environments can span multiple OS and versions

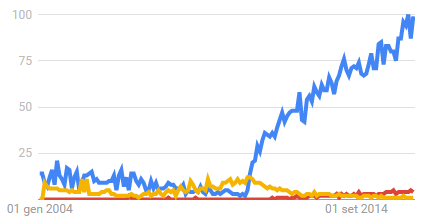


Figure III.8 - Google Trends for Continuous Integration

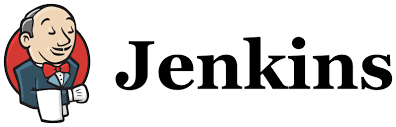
Jenkins

Figure . - Jenkins Logo

Jenkins is one the most known and used CI tools in modern development. As a Java-based Continuous Build system, it is a powerful platform that makes it relatively easy to automate build testing and integration, as well as providing good support for providing Continuous Delivery.

As a CI server, Jenkins acts as the middle man between SCM and Build servers, by checking for updates in SCM branch and triggering build with an automation script. Both the origin SCM server and build options and targets are options selected during the setup of a Jenkins project, while further functionalities can be introduced with plugins. Natively, it can interface with shell scripts, Apache Ant or Maven, Gradle, and other build tools.

Once a Jenkins project strong point is successfully created, the settings are used in order to automate all future builds. As an update is found in the branch, Jenkins automatically checks out the code and starts the build process in an executor, which can also be run in “slave build servers” in order to take advantage of existing distributed architecture.

Jenkins’ strong point is in his reporting capability. It keeps track of build status, letting know the team whether the last build was successful or not, and also giving an average of the build trend. These reports can be greatly enhanced with the use of pre-build plugins, covering unit tests, bug finding, style checking, and others. It can also be implemented in the previously mentioned service for communication, Slack. A plugin is available for the latter, with a bot that can report the status of the last build directly in a Slack channel.

With the Pipeline plugin, built-in into Jenkins, users can implement the project’s entire build/test/deploy pipeline in a Jenkins configuration file, that can be stored and versioned with classical SCM tools such as GIT. A Jenkins pipeline is a Groovy script containing instructions for Jenkins to follow about what to do when the pipeline is run. This script contains information about the whole deployment process, starting from the GIT repository from which check out, the build target, and the tests to run to check whether the build was successful. Once completed the writing of the script, the whole pipeline can be visualized through Pipeline StageView, and run. Jenkins waits for an executor to be free in order to run the workflow, and executes it returning status if programmed to do so. If so desired, a Docker plugin can provide containers that can be run for building or testing the application being deployed.

Jenkins CI is currently being used by many well-known companies worldwide, such as NASA, Apache, Netflix, LinkedIn, Mozilla Foundation, Etsy, and many others. It comes packaged as a WAR file, which can be directly implemented in any servlet container like Tomcat or Glassfish, or it can be run with its own pre-packaged servlet, for a lighter implementation. Packages exists for most common OS available on the market, but can also be used through cloud hosting platform that provide a Jenkins CI instance, like Cloudbees and ShiningPanda.

### 3.3.4. Continuous Testing

DevOps puts great attention to testing, as it gets left-shifted in the delivery pipeline and becomes continuous all along it. DevOps brings testing into the mainstream of development processes, avoiding the problems created by having testing left to the end of the development life cycle, among which release delays and quality issues.

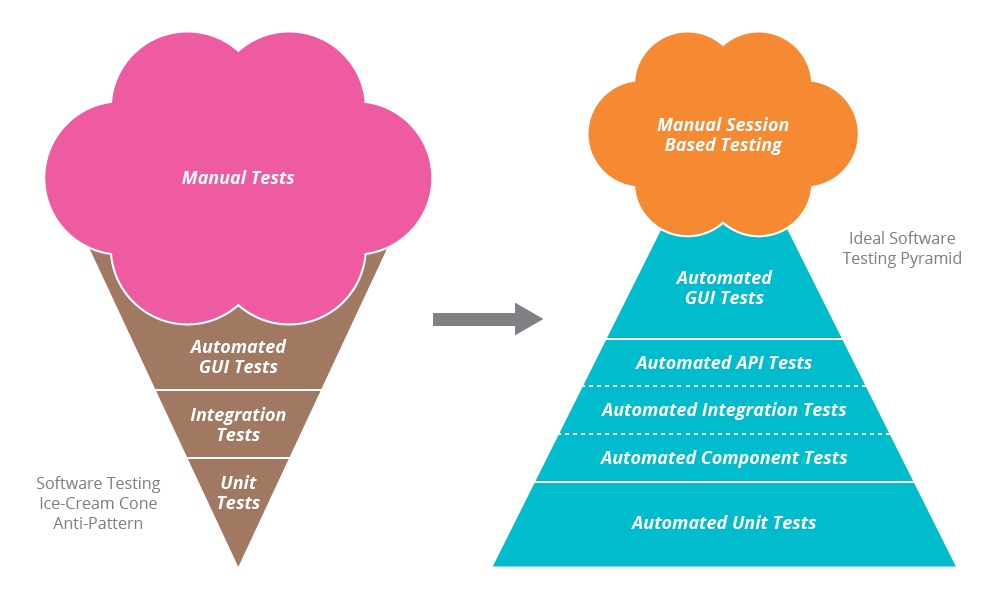
The core philosophy of CT processes is to test every change made to the application under test as early as possible. Many organizations are using tools that can enable automated integration, build and delivery processes, but still rely on manual tests and bug finding. Test orchestration and automation becomes increasingly necessary as the number of services and codebase grow, due to the proportional risk of errors.

Figure III.10 – Pyramidal pattern of Automated Testing and ice cream cone anti-pattern

The automated test pyramid in the figure above, an adaptation of the original “pyramid of automated testing” [32], can be used to follow useful guide lines to determine which tests need to be automated with higher priority. It can be considered as the evolution to the four squares of Agile testing, pointing out which, when and how tests should be done. The pyramid is structured in three layers, where bottom layer tests are cheaper and faster to execute, while moving up costs and duration of the tests increase. A desired situation is to have many more low-level unit tests than high-level end-to-end tests running through a GUI.

Tests from the bottom layer, namely Unit tests, are extremely fast to execute, and are supposed to run them after every build. This gives the team immediate feedback when regressions occur, as codebase keeps growing and evolving. Those tests are made before code integration to trunk (SCM) and during continuous integration to make sure merges are done the desired way.

Common methods of regression testing include re-running previously completed tests and checking whether program behavior has changed and whether previously fixed bugs have re-emerged. Such tests are critical although sometimes can last quite long, which is why they are executed in smaller batches of code, testing single components or units.

Acceptance tests are also quite prone to being automated. These automated component tests, or automated API tests, are used to test business logic without involving the user interface. Testers provide data as inputs to the system and check whether the result are as expected, checking for issues in controls and conditions. These tests are slower than unit tests, although much faster and more reliable than UI tests which are long, brittle and maintenance to their automation is needed any time changes are introduced in the UI.

A balanced CT process keeps the automation at high levels among the first two layers of tests, providing more time for developers to work on new features and components instead of manually testing those already built. A pattern to be definitely avoided, or anti-pattern, is the “ice cream cone” pattern, an inverted pyramid. Automating excessively GUI tests means also needing much more maintenance to those tests as the UI changes, resulting in excessive overhead work, in opposition to DevOps philosophy. Moreover, due to their relative ease to be written and automated, the automation of Unit tests provides the best ROI (Return On Investments) for a team.

Moreover, nonfunctional tests are to be included in order not to lose the efficiencies gained in the overall CT process. However, new challenges are to be faced when integrating nonfunctional tests, which are the availability and scalability of dedicated servers to generate the appropriate user load for load tests and the on-demand utilization of monitoring tools to identify bottlenecks.

Through successful adoption of the CT model based on continuous automated testing, a North American insurance carrier sustained its fast-paced continuous software delivery by improving test environment availability by up to 98% from 70%, reduced deployment time by 60%, saving 800 hours per release [33].

#### 

Figure III.11 - Google Trends for Continuous Testing

#### Selenium



Figure III.12 - Selenium Logo

Selenium is a browser automation tool, commonly used for writing end-to-end tests of web applications. It automates the control of a browser so that repetitive tasks can be reproduced without human intervention.

Selenium is a suite of three tools. The first of them, *Selenium IDE*, is an integrated development environment for Selenium scripts. It is implemented as a Firefox extension, and allows to record, edit, and debug tests. However, it is not only a recording tools, as it also allows for manual editing and writing of scripts, with autocomplete support. At times however, the record/playback paradigm can be limiting and not suitable for every user. Therefore, the second tool of the suite, *Selenium WebDriver*, provides APIs in a variety of languages to allow for more control and the application of standard software development practices, such as version control and unit testing. WebDriver is designed to accurately simulate the way that a user will interact with a web application. JavaScript is used to write a series of events happening in the application being tested, as if a real user were to perform the same interaction. This “events-based” approach can imply difficulties, due to each browser firing different events with slightly different values: where possible, WebDriver fires OS level events, not generated by the browser and therefore compatible with every browser as long as the OS is the same. The last piece of the suite is *Selenium Grid*, which makes it possible to use the Selenium APIs to control browser instances distributed over a grid of machines, allowing tests to run concurrently taking advantage of existing distributed infrastructure.

By using Selenium, testers are given the full strength of any programming language in a library that can easily be implemented into projects to start automating test cases. An interesting example found on Quora [34] shows how easy it is to automate with C# launching the browser and visiting a defined webpage:

IWebDriver driver = new FirefoxDriver(); //to open the browser  
driver.navigate().To(http://www.example.com); //to visit the webpage

## 3.3. Building the Infrastructure

### 3.3.1. Configuration Management

The proliferation of virtualization coupled with the availability of cloud computing has led to a significant growth of the number of servers (or containers) that need to be managed by Operators. Data centers and servers are spread across the globe, while racks of physical servers are becoming more and more obsolete because of their costs.

In many cases, large groups of servers are identical, running identical application and services, and can either be virtualized frameworks within the organization, or cloud instances in remote data centers, or both. Data center orchestration and configuration management tools are designed to administer and control more easily infrastructure of any size and number of servers from one central location, in order to enable *agility* at IT Operations level.

Although very different between each other, CM tools share many characteristics and concepts. Most of them use a slave/master or node/agent model, where a central controller directs the configurations of the nodes, based on a series of instructions defined in the provisioning scripts. Those scripts are written in languages very similar to conventional programming languages, although simplified. CM tools are based on the concept of *idempotency*. The objective is that after each provisioning run the system reaches (or keeps) the desired state, even if run multiple times. This behavior is not necessarily applied in all cases, and can be disabled when so desired.

#### Ansible

Figure III.13 - Google Trends for Configuration Management Tools

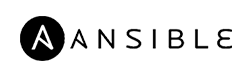


Figure III.14 - Ansible Logo

Ansible works by configuring client machines from a computer with Ansible components installed and configured. It communicates over SSH channels in order to retrieve information from remote machines, issue commands, and copy files. For this reason, an Ansible system does not require any additional software to be installed on the client computers, as any server with an SSH port exposed can be configured and managed via Ansible, regardless of what stage it is at in its life cycle.

Ansible takes on a modular approach, it can be extended to use functionalities of the main system in order to deal with specific scenarios. Modules can be written in any language and communicate with the main Ansible component in standard JSON.

Configuration files are mainly written in the YAML data serialization format due to its expressive nature and its similarity to popular markup languages. Ansible can interact with clients through either command line tools or through its configuration scripts called *Playbooks*.

### 3.3.2. Containers and VMs

With the huge growth in virtualization and cloud computing, there has also been a correspondent increase in the average number of virtual machines (VM) that IT Operators have to manage. Manually creating a full VM in modern virtualizers, like VMWare and Hyper-V, is for an operator a tedious task, starting with the snapshot of the entire machine configuration, to then replicate it to another machine. As this implies waste of space and time, it does not tie well in with DevOps principles.

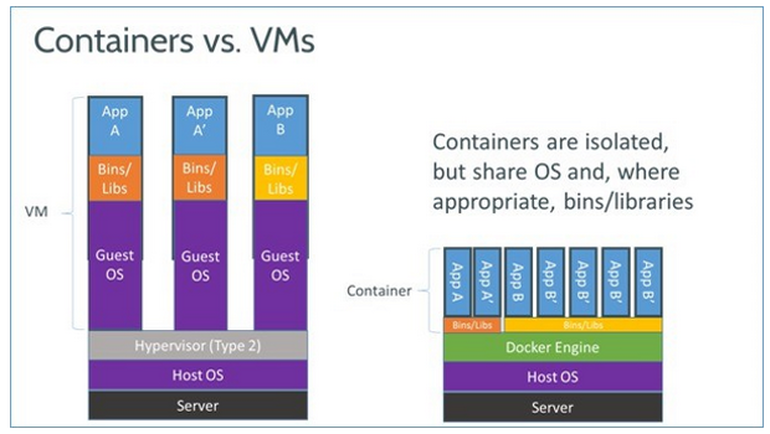
However, based on virtualization technology, an improved model was born. Instead of creating an image of the whole machine to then replicate it, *containerization* allows an operator to share the OS platform where possible, and assign to each *container* the appropriate libraries and applications to run. Doing so, setting up new containers, instead of VMs, becomes extremely faster and more efficient about the use of server resources.

Figure III.15 - Differences between Virtualization and Containerization

Containerization systems make use of LXC, *LinuX Containers*, a userspace interface for the Linux kernel containment features. Its goal is to create an environment as close as possible to a standard Linux installation, without the need for a separate kernel. It does so by sharing the kernel among the containers that are also running on the host machine. Those shared parts of the OS are read-only, while containers’ own space is writable.

Containerization provides a number of benefits. First, the use of resources of the same server becomes much more efficient, as many containers can be installed where just one VM was present thanks to avoiding the redundancy of the same system and kernel being installed repeatedly. Moreover, by sharing kernel and resources, containers can boot much more rapidly than a full VM. Containers can also be run on Amazon Web Services and Azure public clouds.

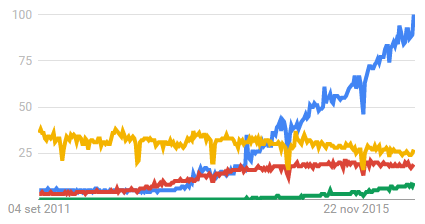
However, virtualization provides an extremely sophisticated management infrastructure built into products such as VMWare’s vCenter or Microsoft’s System Center Virtual Machine Manager through years of experience, with functions and capabilities that are not available yet to modern containerization technology. Google, in collaboration with IBM, Red Hat, CoreOS and Microsoft are working on the open-source Kubernetes Management System, which is supposed to meet quickly the more advanced needs of bigger enterprises [35].

Figure III.16 - Google Trends for Containerization and Virtualization Tools

#### Docker

Figure . - Docker Logo

Docker, previously called dotCloud and open-sourced in 2013, is a Linux-only virtual environment (VE) tool. Docker is written in the lightweight Go language, and it uses helper scripts to create containers as lightweight machines. It builds on LXC (LinuX Containers), which uses the *cgroups* functionality to enable creation and running of multiple isolated Linux Virtual Environments on a single control host, sharing the same kernel and OS, if so desired. Docker allows the user to snapshot the OS and apps into an image, that can be easily deployed on other Docker hosts. It adds the following features to *cgroups*’ ability, according to Docker’s FAQ:

* Portable deployment across machines, but Docker is needed to run the images containing all the bundled applications
* Versioning, including VCS-like capabilities for tracking successive versions of a container, inspecting differences between versions, committing new versions, rolling back, etc.
* Component reusability, by creating an image from any container with the necessary components already installed, and use it as a base for new containers; this is extremely useful in any scenario requiring new environments, such as deployment, testing, and for scalability purposes
* Automatic build, as it includes a tool for developers to automatically assemble a container from their source code, with full control over application dependencies, build tools, packaging, etc., so that developers can use any build tools regardless of the configuration of the machines
* Shared library, in the form of a public registry where the community has already uploaded containers ready to be used

In conclusion, Docker is extremely useful when different scenarios for testing or debugging software are needed. It does so quickly, efficiently and with more than sufficient isolation from the other sandboxed images concurrently running.

#### Vagrant

Figure . - Vagrant Logo

Vagrant, an open-source product released in 2010, is best described as a VM manager. It allows to script and package the VM configurations and the provisioning setup. It is designed to run on almost any VM tool, such as VMWare, VirtualBox, AWS, etc., by installing the provided plugins where needed. Therefore, any common OS platform is supported if it can run any of the above mentioned virtualization tools.

Although Vagrant creates full-fledged VMs, those are still lighter than standard, but most importantly it provides a reproducible way to do so automatically. It guarantees resources at hardware level by pre-allocating them, like any other VM. Vagrant can be integrated with configuration management tools such as Puppet and Chef to provision customized VM setups and configurations.

Vagrant is the go-to virtualization tool when full completely isolated VMs are required.

## 3.4. Monitoring & Logging

As production teams keep delivering features and services, a crucial part of the software development life cycle is to provide fast feedback to developers on how applications are performing in production.

As there is no shortage of tools providing feedback, a distinction has to be made. Metrics monitoring utilities, such as Prometheus, New Relic, Datadog and Sysdig, create customized metrics to suit an organization’s specific applications. Log monitoring tools, including Sumo Logic, Loggly, and Splunk, analyze logs from servers and customers, and offer reports, dashboards and alerting as well as advanced querying to the production team.

However, fundamental to the production team is a way to extract useful information from the data supplied by DevOps monitoring tools. DevOps values monitoring and analytics tools, such as those applying Data Science to log analysis, that can provide intelligence beyond simple status, and can correlate IT performance to business metrics.

#### The Elastic Stack

Figure III.19 - The Elastic Stack Structure for Logging and Monitoring

Elasticsearch is a general purpose full-text search engine used for log analysis when combined with tools for log retrieval and pre-processing – Logstash – caching/buffering – Redis – and visualization – Kibana. Previously, Elastic stack was known as ELK stack, from the acronym of the mentioned tools. Elasticsearch is not a hosted solution but a running software, therefore it needs the traditional hardware acquisition and deployment. It is perfect for analyzing text along with JSON metadata to later perform searches via REST interfaces. It runs well in the cloud and it is built for distributed and high-availability deployments. Some providers even offer hosted Elasticsearch environments.

Integration with different data providers is achieved via Logstash, a dynamic data collection tool for centralizing, analyzing, parsing and forwarding logs, with a large number of built-in integrations with most common log formats and a vast plugin ecosystem. It works in collaboration with Kibana, an analytics and visualization platform that builds on Elasticsearch to give a better understanding of data. It provides filters and search queries to find specific log messages, and shows in a histogram the number of logs received versus time, matched by the filter selected. It even allows to create and modify customized visualization, ranging from *vertical bars* to *pie charts* and *data tables*, that can be shared with other users who have access to the same Kibana instance.

Usually, the Elastic stack is used along with data shippers that send various types of server data to an Elasticsearch instance, allowing to gather information about CPU, memory, and process activity on servers. The Elastic stack website, used as reference for this section, advises Beats as data shipper, but any other alternative such as Prometheus or Statsd can be used.

The Elastic Stack is currently among the most community-backed solutions on Stack Overflow about monitoring and log analysis. Being open-source and provided an already existing hardware or cloud platform to host it, it is basically free. The organization behind Elastic Stack also offers “Elasticsearch-as-a-Service”, a hosted solution by the name of Elastic Cloud, providing REST API to interface with the provided service. It integrates perfectly with custom DevOps workflow considering the large number of customizable metrics parsed from analyzed logs.

An interesting tutorial series has been posted in DigitalOcean Community webpage [36] about centralized logging with the Elastic stack on Ubuntu 14.04.

## 3.5. Moving to the Clouds

The expandability of traditional IT infrastructure is a nightmare. Facing an increasing number of services and related customers, organizations have to scale their physical servers, by buying new ones or upgrading their racks. This process means the investment of a larger budget, and huge delays due to installation and configuration of the new machines to match them to the production environment, without mentioning the risk for outages of service or networking during these operations. Moreover, differences between production environment, as provided by industrial-grade server rack, and development environment, which means developers’ local machines, can introduce significant quality issues in the services provided.

The evolution from traditional IT infrastructure into ubiquitous cloud infrastructure during the 21st century addresses these problems. Operators are provided multi-location redundancy for better stability, and the capability to scale an application or a service rapidly enabling agility. Cloud enables servers to be accessed from anywhere over the Internet, allowing operators to provision new server in minutes when needed, or even set automated tools to respond to determined alarms when traffic cannot be handled by the current configuration. Server virtualization, splitting the resources of one physical machine into multiple virtual servers “sandboxed”, works perfectly with Infrastructure as Code, as it allows fast provisioning for better environment management and cost reduction by reducing the need for permanent, static test environments as traditional racks were.

DevOps without Cloud Computing means not taking advantage of all of its benefits. Many huge businesses are making enormous investments in cloud platform, adopting it and providing some components of it “as-a-Service”. Among the others, Amazon with Amazon Web Services, Microsoft with Microsoft Azure, Google with Google Cloud Platform are the biggest actors investing in this sector. But an honorable mention is also Openstack, an open-source IaaS (Infrastructure as a Service) cloud computing project, born from a collaboration between Rackspace Cloud and NASA.

#### 

Figure III.20 - Google Trends for Cloud Computing Platform

#### Amazon Web Services

Figure III.21 - Amazon Web Services' Logo

Amazon Web Services (AWS) provides on-demand computing resources and services in the cloud. It is currently the cloud-hosting solution having the biggest market share and revenues, leaving behind giants like IBM BlueMix, Microsoft Azure and Google Cloud Platform. It can be used for a large number of solutions, ranging from simple data storage, to application or website hosting, from high-performance parallel computing to real-time services for IOT.

Amazon services are available worldwide thanks to the many data centers in different areas of the world, providing resources closer to where they are needed the most. Each *region*, area of the world where Amazon data centers are, contains multiple distinct locations called *Availability Zones*, which allow protection from outages if resources were places in just a single location.

AWS can be configured to meet the needs of the project that is to be hosted. It can provide full IaaS capabilities, giving engineers the complete management over the cloud infrastructure, so that they can provision virtual server, set up firewalls, configure Internet access, and scale the infrastructure itself on demand.

EC2 is the fundamental block around which the AWS is structured. It provides remote operations on virtual machine on Amazon’s infrastructure, where a single VM is called *instance*, differing one another in available resources. A *micro-instance* is the only free instance provisioned by EC2, with the least resources (only 613 MB of memory) and the lowest priority. An EC2 instance is not scalable by itself, as it needs other services such as *Auto Scaling*, which enables automatic scaling of virtual server based on changes in load, and *Elastic Load Balancing*, which distributes incoming network traffic across multiple EC2 instances.

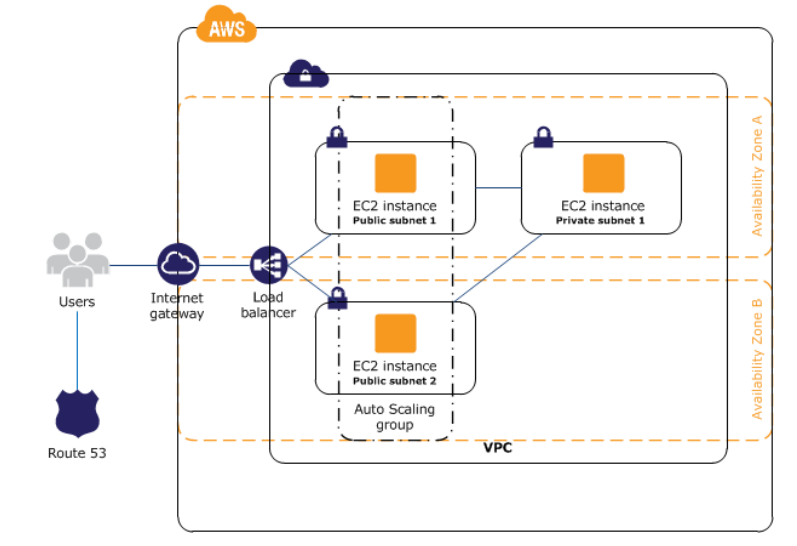
An example infrastructure would be the one in the following diagram. Amazon *Route 53* provides routing of domain name to AWS-hosted infrastructure, in which the load balancer distributes traffic among the EC2 public instances adjusting accordingly to the Auto Scaling group based on the request of resources.

Figure III.22 – Example Computing and Networking Architecture of AWS

AWS also provide storage and Content Delivery services. Among those, Amazon *Simple Storage Service* (S3) is a scalable data storage service based on containers called *buckets* with concurrent access and security restrictions, and it can be used in a wide range of scenarios, from backing up data, to large file storage, to static website hosting. Amazon *CloudFront* helps creating a content delivery network (CDN) of data centers around the world called *edge locations*, so that a user trying to access files part of a CloudFront distribution is always routed towards the data center closest to the user that has it stored in its cache, key factor for improving website speed, especially those providing multimedia. Amazon *Elastic Block Store* (EBS) provides block-level storage volumes available across EC2 instances, useful for data changing frequently and that must persist, such as the primary storage for a database or file system. EBS even provides the ability to load incremental backups called *snapshots* of data to S3.

Other services provided by Amazon platform are AWS *Identity and Access Management*, for managing user access to AWS resources, Amazon *RDS* and *DynamoDB*, respectively relational and NoSQL database services, along with analytics services for AWS such as Amazon EMR, for data processing with Hadoop, and Amazon Machine Learning, to obtain predictions for applications through patterns in data analyzed with ML technology.

Amazon is one of the leading companies in web marketing, as well as in technology adoption. It is one of the first among big companies to have adopted DevOps fully for their production line. They are also providing a suite of services for a AWS-based DevOps approach. It can even be considered like a complete end-to-end infrastructure, completely hosed in Amazon’s cloud.

As a Version Control System, AWS *CodeCommit* is a fully managed source control service for hosting scalable private Git repositories that integrates with the existing Git tools. It integrates perfectly with AWS *CodePipeline*, a Continuous Delivery service for fast and reliable application updates; it builds and tests code whenever changes in the SCM are found, according to customized models and processes configured by the user. AWS *CodeDeploy* completes the Continuous Deployment pipeline, by automatically integrating and deploying code in the specified instances, both EC2 and locally executed ones.

Configuration Management is fully supported by Amazon’s Cloud, providing tools such as AWS *OpsWorks*, an IaC-oriented service that helps configuring and operating applications using Chef by defining application architecture and the resources needed by each component, and AWS *CloudFormation* is an easy way to create and manage a collection of related AWS resources, provisioning and updating them predictably.

Amazon *CloudWatch* and AWS *CloudTrail* complete this “AWS DevOps stack”, by providing monitoring and collection of log files about applications and API usage, triggering alarms and reacting automatically upon changes in AWS resource changes.

Amazon also provides PaaS functionalities with AWS *Elastic Beanstalk*, an easy-to-use service for deploying and scaling web applications and services developed with many of the most common development languages, such as Java, .Net, Node.js, Python, Ruby, and Go, on Apache, Nginx, or IIS servers. AWS Elastic Beanstalk manages automatically implementation, provisioning and auto-scaling to the application, without having to calibrate manually anything at infrastructure-level. This service is not to be confused with the Elastic Stack, nor with any of its components, as they share nothing but the similarity in the name.

A walkthrough of setting up a completely automated DevOps deployment pipeline is available in AWS Documentation webpage. AWS webpage is also rich in Reference Architecture for common use cases, such as static or dynamic webpage, e-commerce marketing or applications hosting, as well as study cases of companies that have chosen AWS as their cloud-hosted solutions and the benefits gained.

Amazon Web Services are generally not free, and are billed based on the “pay-for-what-you-use” model. However, AWS provides a free usage tier that expires after 12 months from sing up, with restrictions and quotas on usage that will be billed at normal rates for anything exceeding said quotas.

# Conclusions

In this work much has been written about what DevOps is, from how it proposes to change organization from a cultural level, building an environment with collaboration and sharing as primary values and focusing on communication and continuous innovation and improvement, to the necessary steps to take advantage of its principles in order to shorten development cycles, increase deployment frequency, and achieve faster time-to-market. Tools have been defined as “enablers” of the DevOps approach, making possible the adoption of previously discussed principles at a technological level, and an ideal model of a DevOps-based architecture for software development and delivery has been proposed.

From the concepts introduced and analyzed in this work, applying DevOps methodology to large organizations working on projects involving cooperation of several teams is supposed to have a big impact, and bring quantifiable technical and business benefits. An example has already been given in the “Continuous Testing” section of the “The DevOps Tool Rack” chapter, in which a North-American insurance company improved noticeably its production life cycle and feedback quality, and reaped the fruit of adopting the DevOps philosophy.

Many other companies can exemplify the possibilities and the benefits of DevOps.

Amazon moved its business from dedicated server farms, expensive and difficult to manage when facing traffic spikes such as during the Christmas shopping season, to its own cloud service, Amazon Web Services. Adopting containerization technology applied to cloud hosting, it allowed engineers to scale capacity up or down incrementally and based on the actual traffic of the portal. Not only did this reduce costs on server capacity and the difficulties encountered in scalability, but it also enabled the transition towards a continuous deployment process, so that developers can deploy code to whenever needed. Within a year of Amazon’s move to AWS, engineers were deploying code every 11.7 seconds on average [37].

Netflix has proven to be a real entertainment game-changer in the last years, providing its streaming service basically in every house in the US and actually spreading very fast in Europe. What Netflix is not really known for is the creation of the *Simian Army*, a suite of automated tools developed by a group of hundreds of developers that stress test Netflix’s infrastructure allowing the company to identify and resolve vulnerabilities before the deployment of features to customers [38]. Netflix has continued its commitment to automation and open source, and last year it was awarded with the JAX Special Jury Award, as “the rate at which [Netflix] has adopted new technologies and implemented them into its DevOps approach is setting new standards in IT” [39].

Even Facebook, to reduce outages of service and customer dissatisfaction, has recently migrated its entire infrastructure and back-end IT to the Chef configuration management platform [40], even making some of its *cookbooks* (Chef’s equivalent to Ansible’s playbooks) available to the public. This way development and delivery life cycles are greatly accelerated, so that nobody misses that important Facebook post.

Downtimes also impacted Etsy’s sales to millions of users through its online marketplace, risking driving them to competitors. Once changed technical management team, Etsy could transition from its waterfall model, and its four hour full-site deployments twice weekly, to a fully automated pipeline based on the continuous delivery process [41], resulting in more than 50 deployments a day with fewer disruptions of the service [42]. For this reason, Etsy has been made a model for the adoption of the DevOps framework with its astounding 60 million monthly visits and 1.5 billion page views per month, even though it does not have a DevOps group among its teams.

Other examples can be made, such as Target, Walmart, Adobe and Sony Pictures Entertainment; each and every one of those companies have benefited greatly from the transition to a DevOps approach, living proofs of the success that this methodology enables at every level, whether the company is large or small, young or old [43].

Puppet Labs’ annual “State of DevOps Report” is a great tool for measuring the evolution and effectiveness of DevOps over time. It highlights the trends in the DevOps movement and provides valuable insights on organizations that have already embraced DevOps, as well as numbers on money and time saved thanks to the adoption.

Some interesting conclusions can be extracted from reviewing said report [44]. DevOps is key to accelerating deployment, while preserving reliability and stability. Code deployment of organization adopting DevOps happens up to 30 times more frequently than competitors, while having 60 times fewer failures and recover up to 168 times faster. Continuous deployment enables better software to be delivered more consistently, while DevOps automated tools improve organizational performance, streamlining routine operations. The collaborative and sharing environment also helps with tech burnout phenomena, thanks to a more respectful and supportive work experience fostered by DevOps culture.

Moreover, putting aside the great numbers and values of cost savings and efficiency, there is another true return on investment earned with the adoption of DevOps. Saving time for tedious work means more time for engineers, which can be reallocated to other productive activities, adding further value to a company. The proofs of this are in every example proposed earlier in this section: Amazon and Facebook are investing in new sectors far from their comfort zone, while Netflix decided to spread in Europe and face the television monopoly over dining room entertainment. Those forward-thinking companies know that reinvesting the human time, creativity and energy saved by reducing rework and downtime result in improvements to the organization’s performance, revenues, profitability and other measurable outcomes.

In conclusion, the road to the adoption of DevOps is not straight, nor easy by any means. DevOps is a complex concept that lacks of an orthodoxy, which means references and models to follow, but it is rich in principles and values. There is no path to follow that can lead directly to the great benefits DevOps promises, but a multitude of smaller branches, which make implementations different from company to company. DevOps can even be met with skepticism by older organizations in the beginning, due to the necessary reformation of the existing culture and processes feeling like a gargantuan task, sure to be time-consuming even though promising of many benefits. However, if the main challenges faced when adopting DevOps were to be overcome, DevOps implementation is sure to give maximum benefit, and the company transitioning to come on top over its competitors.

# Bibliographic and Web References

[1] Jackson, Chris. DevOps Adoption. *RackSpace.* [Online] October 2014. https://www.rackspace.co.uk/sites/default/files/devops-automation-report.pdf.

[2] United States, Navy Mathematical Computing Advisory Panel. *Symposium on advanced programming methods for digital computers.* Washington D.C. : Office of Naval Research, Dept. of the Navy, 1956.

[3] Royce, Winston. Managing the Development of Large Software Systems. *University of Maryland - Department of Computer Science.* [Online] 1970. http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf.

[4] Poppendieck, Mary and Poppendieck, Tom. *Implementing Lean Software Development.* s.l. : Addison Wesley Longman Inc Div Pearons Suite 300, 2006.

[5] Beck, kent, et al. Manifesto for Agile Software Development. 2001.

[6] ITIL Glossary and Abbreviations - English. *ITIL | AXELOS Global Best Practices.* [Online] July 2011. https://goo.gl/m1C4uG.

[7] Hüttermann, Michael. *DevOps for Developers.* s.l. : Apress, 2012.

[8] Loukides, Mike. *What is DevOps?* s.l. : O'Reilly, 2012.

[9] Sharma, Sanjeev and Coyne, Bernie. *DevOps for Dummies, 2nd IBM Limited Edition.* s.l. : John Wiley & Sons, Inc., 2015.

[10] Kim, Gene, Behr, Kevin and Spafford, George. *The Phoenix Project.* s.l. : IT Revolution Press, 2015.

[11] Dodson, Jessica. A Brief and Incomplete History of Devops. *Sonatype blog.* [Online] July 29, 2013. http://blog.sonatype.com/2013/07/a-brief-and-incomplete-history-of-devops/.

[12] Humble, Jez. There is No Such Thing as a “Devops Team”. *Thoughtworks blog.* [Online] October 22, 2012. https://www.thoughtworks.com/insights/blog/there-no-such-thing-devops-team.

[13] Riley, Chris. How to Keep CALMS and Release More! *Blog Logentries.* [Online] October 24, 2014. https://blog.logentries.com/2014/10/how-to-keep-calms-and-release-more/.

[14] *Why enterprises must adopt devops to enable continuous delivery.* Humble, Jez and Molesky, Joanne. Cutter IT Journal : s.n., 2011, Cutter IT Journal.

[15] Kim, Gene, et al. *DevOps Cookbook.* 2013.

[16] Kim, Gene. The Three Ways: The Principles Underpinning DevOps. *IT Revolution Press.* [Online] 2014. http://itrevolution.com/the-three-ways-principles-underpinning-devops/.

[17] Willis, John. Docker and the Three Ways of DevOps. *Docker Blog.* [Online] May 26, 2015. https://blog.docker.com/2015/05/docker-three-ways-devops/.

[18] Withers, Stephen. How to Kick-Start Your DevOps Adoption. *DevOps.com : Where the world meets DevOps.* [Online] March 25, 2016. http://devops.com/2016/03/25/kickstart-devops-adoption/.

[19] Edwards, Dibbe. IBM Rational DevOps Transformation. *IBM.com.* [Online] 2015. https://www.ibm.com/devops/method/experience/culture/rational\_devops\_transformation.

[20] Humble, Jez and Farley, David. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* s.l. : Addison-Wesley Professional, 2010.

[21] Gajda, Kim and Kuczarski, Laura. Evolving from Waterfall to Continuous Delivery. *IBM.com.* [Online] 2015. https://www.ibm.com/devops/method/content/culture/practice\_evolve\_to\_continuous\_delivery/.

[22] Edwards, Dibbe. DevOps: Shift left with continuous testing by using automation and virtualization. *IBM Bluemix Garage Method.* [Online] https://www.ibm.com/devops/method/experience/deliver/dibbe\_edwards\_devops\_shift\_left/.

[23] Philipp-Edmonds, Cameron. The Relationship between Risk and Continuous Testing: An Interview with Wayne Ariola. *StickyMinds.* [Online] December 2015. http://www-stickyminds.com.

[24] Marvin, Rob. *Testing in a Continuous Delivery World.* s.l. : SD TImes, 2014.

[25] Rouse, Margaret. Infrastructure as Code (IAC). *TechTarget.* [Online] October 2015. http://searchitoperations.techtarget.com/definition/Infrastructure-as-Code-IAC.

[26] Mann, Andi. Four key vectors for closing the DevOps feedback loop. *DevOps.com.* [Online] September 29, 2014. http://devops.com/2014/09/29/four-key-vectors-closing-devops-feedback-loop/.

[27] Sharma, Sanjeev. Undestanding DevOps - Part 7: Continuous Improvement. *SDArchitect.* [Online] October 17, 2014. https://sdarchitect.wordpress.com/2014/10/17/understanding-devops-part-7-continuous-improvement/.

[28] Bloomberg, Jason. Spread the DevOps Virus in Your Organization. *Intellyx: Cortex Newsletter.* [Online] February 29, 2016. https://intellyx.com/2016/02/29/spread-the-devops-virus-in-your-organization-part-one/.

[29] Bashan, Shmuel and Bellagio, David. *Work Item Management with IBM Ration ClearQuest and Jazz: A customization Guide.* s.l. : IBM Press, 2011.

[30] Loghem, Robert Van. So What Is A Deployment Really? *Xebia Blog.* [Online] July 8, 2009. http://blog.xebia.com/so-what-is-a-deployment-really/.

[31] Partington, Vincent. Deployment Automation vs. Build Automation. *DZone.* [Online] October 13, 2010. https://dzone.com/articles/deployment-automation-vs-build.

[32] Cohn, Mike. Test Automation: Let Service Be Your Middle Man. *ScrumAlliance.* [Online] December 2014. https://www.scrumalliance.org/community/spotlight/mike-cohn/december-2014/test-automation-let-service-be-your-middle-man/.

[33] Cognizant. Jumpstarting DevOps with Continuous Testing. *Cognizant.com.* [Online] December 2015. https://www.cognizant.com/whitepapers/Jumpstarting-DevOps-with-Continuous-Testing-codex1719.pdf.

[34] Dubey, Pankaj. How does the Selenium WebDriver work? *Quora.* [Online] May 2, 2016. https://www.quora.com/How-does-the-Selenium-WebDriver-work/answer/Pankaj-Dubey-41?srid=uiVGW.

[35] Rubens, Paul. The Benefits of Docker vs. Server Virtualization. *ServerWatch.* [Online] August 19, 2014. http://www.serverwatch.com/server-trends/the-benefits-of-docker-vs.-server-virtualization.html.

[36] Anicas, Mitchell. Centralized Logging with ELK Stack (Elasticsearch, Logstash, and Kibana) on Ubuntu 14.04. *DigitalOcean.* [Online] March 15, 2015. https://www.digitalocean.com/community/tutorials/how-to-install-elasticsearch-logstash-and-kibana-elk-stack-on-ubuntu-14.04/.

[37] Lawton, George. How Amazon Made The Leap To A DevOps Culture. *Service Virtualization Today.* [Online] September 5, 2013. http://servicevirtualization.com/how-amazon-made-the-leap-to-a-devops-culture.

[38] Bradley, Tony. Netflix, the Simian Army, and the culture of freedom and responsibility. *Devops.com : Where the world meets DevOps.* [Online] March 27, 2014. http://devops.com/2014/03/27/netflix-the-simian-army-and-the-culture-of-freedom-and-responsibility/.

[39] Hamilton, Coman. Jax Award winners announced at JAX 2015 Conference! *JAXenter.* [Online] April 22, 2015. https://jaxenter.com/jax-awards-winners-announced-at-jax-2015-conference-116358.html.

[40] Courtemanche, Meredith. Facebook completes Chef Devops migration, shares lessons. *TechTarget SearchITOperations.* [Online] May 2015. http://searchitoperations.techtarget.com/feature/Facebook-completes-Chef-Devops-migration-shares-lessons.

[41] Dix, John. How Etsy makes DevOps work. *NetworkWorld from IDG.* [Online] February 19, 2015. http://www.networkworld.com/article/2886672/software/how-etsy-makes-devops-work.html.

[42] Miranda, Joao. How Etsy Deploys More Than 50 Times a Day. *InfoQueue.* [Online] March 17, 2014. https://www.infoq.com/news/2014/03/etsy-deploy-times-a-day.

[43] Null, Cristopher. 10 companies killing at DevOps. *TechBeacon.* [Online] September 25, 2015. http://techbeacon.com/10-companies-killing-it-at-devops.

[44] PuppetLabs. 2016 State of DevOps Report. *Puppet.com.* [Online] 2016. https://puppet.com/resources/white-paper/2016-state-of-devops-report.

[45] Brizeno, Marcos. Kickstart CD with deployment automation. *ThoughtWorks.* [Online] January 17, 2014. https://www.thoughtworks.com/insights/blog/kickstart-cd-deployment-automation.

[46] RIley, Chris. 6 Challenges Facing DevOps and Operations Teams in 2015. *Logentries Blog.* [Online] October 14, 2014. https://blog.logentries.com/2014/10/6-challenges-facing-Devops-and-operations-teams-in-2015/.

[47] XebiaLabs. PERIODIC TABLE OF DEVOPS TOOLS (v2). *XebiaLabs: Deliver Faster.* [Online] https://xebialabs.com/periodic-table-of-devops-tools/.

[48] UpGuard. Docker vs. VMWare: How Do They Stack Up? *UpGuard - Turn Risk into Resilience: Discover. Control. Fortify.* [Online] https://www.upguard.com/articles/docker-vs.-vmware-how-do-they-stack-up.

[49] Seshachala, Sudhi. Docker VS VMs. *DevOps.com: Where the world meets DevOps.* [Online] November 24, 2014. http://devops.com/2014/11/24/docker-vs-vms/.

[50] Daniels, Katherine. Putting the Dev in Devops: Bringing Software Engineering to Operations Infrastructure Tooling. *Code as Craft, Etsy's Engineering Blog.* [Online] February 22, 2016. https://codeascraft.com/2016/02/22/putting-the-dev-in-devops-bringing-software-engineering-to-operations-infrastructure-tooling/.

[51] Barrano, Emanuele. DevOps: Development featuring Operations. *MokaByte.* [Online] September 2015. http://www.mokabyte.it/2015/09/devops-2/.

[52] Docker. Frequently Asked Questions (FAQ). *Docker.com.* [Online] https://docs.docker.com/engine/faq/.

[53] Chapman, David. Introduction to DevOps on AWS. *Amazon Web Services.* [Online] December 2014. https://d0.awsstatic.com/whitepapers/AWS\_DevOps.pdf.