Towards DevOps: Practices and Patterns from the Portuguese Startup Scene

Carlos Teixeira - ei11145@fe.up.pt, Tiago Sousa - tbs@fe.up.pt, Hugo Ferreira - hugosf@fe.up.pt

Faculty of Engineering of the University of Porto

Abstract. As more businesses and opportunities arouse, so did the challenges and the need for better and more robust practices and methodologies. Initial software development methodologies like Waterfall would prove to be inefficient in dealing with the management or Software Products and would create a clear separation of responsibilities between developers and operations. This separation would persist across the 2001 Manifesto for Agile Software Development and the subsequent movement and methodologies that emerged.

As years passed and complexity grew, extra pressure would be put both on the operations professionals and the developers. This pressures would eventually highlight the flaws of the siloed approach where the lack of cooperation, caused by mismatching objectives and low communication, would lead to inefficiencies as well as an atmosphere of tension between the two departments. Businesses would, as a result, struggle to deliver value in a fast and sustained manner.

Attempts to solve this problem would eventually lead to the appearance of DevOps.

Formed by combining the word Development with the word Operations, the "DevOps" movement can be traced back as far as 2008 making it almost 8 years old (as of the writing of this article). Interpretations of the movement and what it means are, nevertheless, still topics of discussion and written literature on the subject is still scarce. In this article we demystify DevOps by analyzing the current practices associated with DevOps in the real world. We do this by showing the results of an interviewing process carried out across 25 Portuguese startups were a set of patterns related with DevOps and its values was identified. Some of those patterns(13 in total) are, subsequently, validated by watching the effects of their application in a real world situation.

1 Introduction

Traditional organizations and organizational patterns have two distinct departments. The development department, that is responsible for developing new software and wants to see it in production[1] and the Operations department that is charged with the responsibility of managing the running application[1].

When new features are pushed by the Development department, bugs may be introduced. Being incentivized to keep the application stabel it would not take long until operations professionals started to see the increments developed by developers as harmfull. This would lead to friction between the two departments [1].

As this problem grew bigger attempts were made to reduce it and eventually the DevOps movement was born.

1.1 Motivation

Being a movement that aims to solve the divide between development and operations, DevOps is, as a result, a way to reduce the inefficiencies of those departments. The new way brought by DevOps to look at the entire software pipeline, from development to delivery and maintenance, represents an advantage for teams in the sense that it creates a more efficient, agile and collaborative way of working.

Furthermore DevOps, by taking advantage of newer technologies and enforcing automation practices, aims to reduce the time to market of new features while reducing deployment errors [1] which can directly be translated into a competitive advantage for businesses that aim to be able to respond quickly to changes in their environment [3].

1.2 Outline

Within this article you will find the results of a survey conducted across 25 portuguese startups. We will begin by refering the way the information was extracted. Then, a small summary of the observed patterns will be presented followed by three fully expanded ones. Finally, we describe the validation results and end with the conclusions of the study.

2 Towards DevOps

2.1 Introduction

The study of DevOps is still in its infancy and there are still few studies that identify practices related with it. Having little literature written about the subject and taking into consideration that the movement was born in the industry the only option was to find answers near those that have been the main engine of DevOps.

The target chosen were Portuguese Startups. Having limited resouces, startups rely on their teams and team members to succeed. Because startups are usually small, this means that often, members have to assume different roles and be able articulate seamlessly. Furthermore, startups must be prepared to grow quickly which, with a limited number of members, means that automation techinques must be put into practice. This set of characteristics is a indication of a DevOps mentality within startups making them an excelent sample from which to extract information.

2.2 Extracting the information

In order to identify which practices companies startups were adopting, we choose to interview them. We elaborated an interview script with exploratory characteristics and ended up interviewing 25 companies.

The elaborated script would define the company approach to the following concepts:

- Team orchestration
- Software development practices
- Deployment process
- Infrastructure management
- Monitoring and error handling

2.3 Patterns Summary

We have identified, throughout the study, a total of 13 patterns. In this article we will briefly describe 10 of those patterns and then fully describe the remaining 3. The full description of the 13 patterns will be found in the Master Thesis "Towards DevOps: Practices and Patterns from the Portuguese Startup Scene".

Cloud
Cloud allows the elimination of upfront costs and enables businesses with the possibility to quickly increase or decrease the allocated resources. The pay-as-you-go model also means that you only pay for what you use.

Additionally, some cloud providers offer pre-configured tools with their services(e.g. load balancers, monitoring tools, ...).

Team orchestration

A multidisciplinary approach is usually desirable for teams to be able to face all challenges autonomously. In order for this approach to be possible, teams should be as independant as possible and be able to self-organize and communicate effectively.

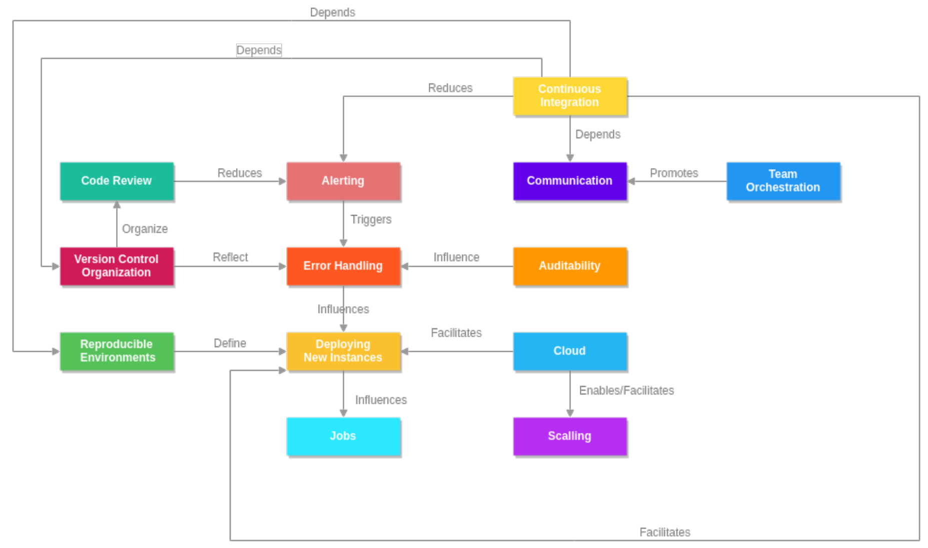

Fig. 1. Pattern map

Auditability
Continuously monitoring your infrastructure is crucial in order to give your customers a pleasant experience. For some, this means monitoring the health of each instance and guaranteeing that users can have access to the application while others have concerns related with the response time for each user request. Being able to define this metrics and for each metric, a set of desired values will help you react faster to errors or even to detect them before they become a real problem.

Alerting
When a specific metric is not in the set of desired values you would want that situation solved as quicly as possible. Notifying someone that can evaluate and fix the error is usually the way to go. As the person you notify is going to be the first responder it is important to choose wisely. Solutions for this problem include notifying all of your team members, a specific team member, a subset of your team or having a rotation schedule where different team members are notified in each day.

Error handling
When an error is detected, it is important to know how to respond. There are three main approaches for tackling erros and they are as follow:

– Fixing the problem and applying the newer version.
– Deploying and older, working version.

- Keeping a backup of an older working version and, when an error is detected, redirecting the traffic to that version.

Scalling
Scalling your application will help you handle traffic as your business grows. Usually, it is prefereable to scale horizontally because it is often cheaper and because there are virtually no limits to how much you can scale. Nevertheless, this approach forces you to take special considerations like having some way of distributing traffic across your instances, keeping track of several instances health and so on. Scalling vertically, by adding resources(e.g. RAM, CPU, Disk memory), on the other hand, is often easier but limitations exist to how much you can scale. Prices are also higher than scalling horizontally.

Deploying new instances
This pattern is deeply connected to the way environments are setup. Depending on the choices made for that particular problem there are three main ways to deploying new instances:

- Downloading the container.
- Downloading the virtual machine.
- Running a script inside the target instance.

Jobs
Sometimes, there are task/jobs that are continuosly running or that only run from time to time. This tasks can be handled by keeping a daemon running on some instance(s) of by launching a new instance to handle them.

Code Review
Code Reviews are a process by which all new increments to the project must be reviewed by someone other than the person that developed that increment. Usually the code reviewer is any member of a team altough some approaches prefer to define the code reviewer as a senior or as someone with a deeper understanding of a specific module. Depending on which approach you use, code reviews can help reduce code duplication, increase coding standarts and allow for a better dissemination of knowledge across the team.

Version Control Organization
With several people contributing to the same project it becomes difficult to manage those contributions. Extracting information, like what is the current version currently in production or what feature are ready for the next release may, furthermore, be of interest. Being able to organize you version control system in order for it to give you this information will help you better handle erros, plan releases, etc.

2.4 The Communication Pattern

Context

Teams will often have members with different degrees of knowledge not only because there may be less experienced members but, if teams are multidisciplinary, different members have different areas of specialization. In this kind of environment one would want to facilitate knowledge sharing in order for the team to be able to work together.

Problem

What kind of communication channels should the team have at their disposal to effectively communicate?

Forces

- Comunication should be as straightforward as possible.
- Some pieces of information(e.g. links, files, code snippets,..) must be shared.
- Sometimes you want information to be persisted and available(e.g. communication with external clients)
- Some informations may be optional or may not need to be read immediately.

Solution

There is no sinlge solution that can solve the communication on its own. We have found that most companies use a combination of different channels in order to facilitate communication

Direct Communication is effective in handling day to day communications (e.g. sharing opinions, ask questions). Teams that used this channel of communication would usually work side by side.

Chat is usually used as a way to share links and files. Usually chat tools can also be integrated with external tools allowing them to deliver additional information(*Continuous delivery*, *Alerting*).

Finally, email can be used to handle more formal communications as well as communication that does not need to be read immediately like information about a team event or the scheduling of a meeting.

2.5 Reproducible Environments Patterns

Context

Each team member will have one working machine were he/she works. Production, test and/or stagin servers will also add to this list o machines that run your application.

Initially you can keep your machine environments (libraries, packages, etc) synchronizes manually but, as new developers join the team and more servers are needed, upgrading them becomes increasingly difficult.

Problem
How do you guarantee that all instances of your application share the same running environment?

Forces

- Creating a full copy of your environment(OS, libraries, ...) may create large files that can be hard to move around.
- Depending on external providers may prove to be a security risk.
- Fetching all the dependencies each time an environment is setup takes a long time.
- Upgrades should be easy to achieve.
- Cloud usually restrict the level of access to their infrastructure.
- There should be consistency across setups.
- Sometimes it is positive that multiple environments can coexist in the same machine.

Solution
Guaranteeing environment consistency and reproducibility should start by describing the environment. Usually, this description takes the form of a script like a bash or ruby. This script can then be used in the creation of an environment that can be shared in a number of different ways.

Using only a script means that you probably will have an easy time passing it around because of its usually small size. In order for scripts to be more than just a simple set of commands, some tools exist that take care of extra dependendencies and also that across runs the same version of each package is always installed. Nevertheless, when you want to setup a new machine/environment you will probably have to fetch all the packages/dependencies again.

Virtual Machines are often setup with the help of scripts. Onde the virtual machine is setup it can be shared as a whole meaning that every time that VM is put to run, the same environmet will be observed. Problems with using this approach are usually concerned with the size of virtual machines as they carry the entire operative system with them. Additionally because VM do not share resources with other VM's running in the same host it may be difficult for several VM's to coexist.

Cloud providers do not always let you run you VM in their infrastructure and sometimes, the ones that do only let you use certain VM's.

Containers are a lightweight alternative to Virtual Machines. The setup process is pretty similar to VM's but the generated representation/image is much smaller. This decrease in size comes from the fact that containers share resources with the host machine and even among containers. By doing so, it is usually possible to have multiple containers running in the same host.

Some cloud providers already have options were they support containers natively.

2.6 Continuous Integration Pattern

Context

When projects grow and several developers contribute continuously to a single project, chances of someone introducing and error increase. When introduced, errors may only be detected far into the future which can add an additional layer of difficulty in the debug process.

Detecting errors as soon as they are introduced is, as a result, a welcomed ability.

Furthermore, having a history of the changes made together with information about the impact of those changes also proves to be an advantageous because it allows the acceleration of release processes.

Problem

How to setup a system that allows developers to receive quick feedback regarding their development?

Solution

Continuous Integration is a technique by which every time an increment to the project occurs an integration process for the resultant application is triggered. From SaaS offers to desktop applications, several options exist that allow companies and individuals to implement a Continuous Integration pipeline.

Continuous integrations implementations usually divide their pipeline into three main sections:

- A build is a materialization of your application. Depending on your choice of technologies it can assume the form of a container, or a binary,or a conjugation of both, or a virtual machine and so on. It is very important that, in this phase, the build reflects the environment where it will run (e.g. the build should be compiled with the same java version existent in the servers)(see *Reproducible Environments*) in order for the following processes to be valid.
- The existent test suite should be run at this point. Unitary, Integration and any other type of test available should run at this point. All of the tests should be run using the previous build in order to ensure that the build was correct. Additional metrics like performance, linters, code coverage can also be integrated into this step.
- When the tests are over it is important to notify the team of this event and of the result. The notification process can take advantage of the *Comunication* pattern described above.

3 Validation

We wanted to know if the indetified patterns were able to bring value to the teams that practice them or to the company where they were practiced. In order to find out, we implemented some of them in a real world scenario and measured the changes that followed. The company were we implemented this patterns was Ventureoak [1] This were the results of that implementation:

3.1 Cloud

We choose to use a PaaS (AWS Elasticbeanstalk). We were able to take advantage of the many existing services for monitoring and alerting as well as the pre configured options. Before this, if the team wanted to start a new server/project they would have to manually configure the server (install dependencies, libraries,...) and then install the tools needed for alerting, monitoring , load balancing,etc. This process would take around three hours to complete manually, with the help of the PaaS platform we were able to reduce the time needed to setup a new environment to around 20 minutes.

Another benefit of using the cloud that we identified was that we were able to turn off machines when we did not need them. This way we did not have the cost of maintaining machines during weekends and holidays.

3.2 Code Review

Practicing code reviews allowed team members to share comments about each others code and with that, members were able to share knowledge and ideas. Additioanlly, because members of the team would see each others code, they would be able to sometimes reuse that code reducing the ammount of code duplication.

3.3 Reproducible Environments

We choose to use containers in order to create a reproducible environment for each team member. This environment was then shared across the team. With this, we observed a decrease in the time it took for members to setup their projects(from hours depending on the complexity of the project to just a couple of minutes) and we also able to have different libraries for each project(e.g. developers would be able to use PHP7 for one project and PHP5.6 for other projects without having to change the configuration of their machines).

[1] Ventureoak (www.ventureoak.com) is a portuguese software development company with 19 employees.

3.4 Deploying new instances

The deployment on new instances followed the choice made for *Reproducible Environments*. By choosing to pull a pre-built container from a registry, we were able to provide developers with the option to have, in their machines, an environment equal to the production one. This gave them the possibility to, if needed, pull the image used in the production to their personal computers and test the project in that environment.

3.5 Continuous Integration

The CI approach, provided the Project Managers/QA with the tools needed to individually check each feature as soon as they were ready. Doing so before would require the developer to demo the new functionality in their personal machine or it would require manual deployment. In the first approach the developer would have to stop their work for the ammount of time needed for him/her to demo the functionality, in the second approach the developer the developer would have to take around 10 minutes to perform the deploy.

With this CI approach we were able to give Project Manageres/QA a third option in which they could deploy the developed functionality to a development environment. This deployment was automatic so no one had to stop what they were doing. The deployment would take around 5 minutes to deploy (if the built step was ready) and around 20 minutes if there was a need to run the entire pipeline.

3.6 Auditability

Because deploys were being made to a staging environment, we were not able to fully test the Auditability pattern. This was due to the fact that monitoring this environment would not bring value to the company (e.g. if the environment was unhealthy there was no real danger).

4 Conclusion

The DevOps movement allowed Developers and Operations to collaborate in ways that were not possible before. With the help of newer, more robust technologies and techniques, this collaboration has enabled projects with bigger dimensions to be handled by smaller teams.

Startups, beeing force to scale and adapt fast but having restrictions on the ammount of resources they can spend, have also adopted this new culture in order to go around this limitation.

DevOps is therefore, an increasingly important topic with applications and repercussions both in the startup community and in the more industrial environments.

With this work, we believed that we were able to create a comprehensive representation of DevOps and the areas it touches. Further investigation and specification is, nevertheless, needed, in order to create a rich and complete pattern language for DevOps that can help formalize and strengthen the movement.

References

1. Bass, L., Weber, I., Zhu, L.: DevOps: A Software Architect's Perspective. 1 edn. (2015)
2. Debois, P.: Agile infrastructure and operations: How infra-gile are you? Proceedings - Agile 2008 Conference pp. 202–207 (2008)
3. Wettinger, J., Breitenb, U., Leymann, F.: Standards-based DevOps Automation and Integration Using TOSCA. Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014 pp. 59–68 (2014)