FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Towards DevOps: Practices and Patterns from the Portuguese Startup Scene

Carlos Manuel da Costa Martins Teixeira

DISSERTATION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. Hugo Sereno Ferreira

Co-Supervisor: Tiago Boldt Sousa

June 21, 2016

Towards DevOps:
Practices and Patterns from the Portuguese Startup Scene

Carlos Manuel da Costa Martins Teixeira

Mestrado Integrado em Engenharia Informática e Computação

June 21, 2016

Abstract

The DevOps movement aims to reduce the articulation problems and the friction that exists between Developers and Operators by creating a new culture of collaboration between the two as well as introducing new practices and methodologies.
With initial developments going back as far as 2008/2009, ideas regarding the DevOps movement and what it means are still evolving and, eight years later, are still topics of discussion even within the community. Combining that with the fact that some studies are showing a significant number of companies adopting DevOps and an equal amount looking into doing the same it becomes worrying that DevOps literature is still largely based on personal opinions rather than scientific or academic literature.
With the objective of filling that vacuum, we looked into some of the Start-up companies operating in Portugal. We talked with 25 of them and tried to understand what challenges did they face and what solutions did they adopt to face them. With the knowledge gathered from this interviews we compiled 13 patterns that can serve as both an overview of some of the DevOps practices and as a basis on which future studies can build. Additionally we have also developed a DevOps state-of-the-art on which we define what is DevOps and what are the main ideals and goals that DevOps advocates.

Contents

List of Figures

List of Tables

Abbreviations

ADT	Abstract Data Type
NIST	National Institute of Standards and Technology
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
IoT	Internet of Things
API	Application Interface
CAMS	Culture, Automation, Measurement, Sharing
CI	Continuous Integration
SCM	Source Control Management

Chapter 1

Introduction

The name DevOps comes from joining the words 'Development' and 'Operations'. With some of the original work that triggered the rise of the DevOps movement being traced back as far as 2008, we have been able to find multiple accounts and histories of personal opinions regarding the subject. This amount of information has, however, not been the subject of scientific and academic analysis meaning that DevOps is still surrounded by uncertainty.

Context

Developing and maintaining/operating software are often seen as disjoint tasks and responsibilities. This pattern has been observed in several organizations like the ones described in **??** and project management techniques like Waterfall **??**. Nevertheless, this was not always the case, and, in the beginning, the same person that developed the software was also the person that operated it [14].
As a result of this separation, two separated departments often exist. This two departments, Development and Operations, are usually not able to efficiently articulate which causes friction between the two as well as a bottle neck for businesses.

Problem

The DevOps movement spans throughout a vast set of areas and tries to change both the technical and cultural aspects related with software development and with the software operations.
Looking at the current state of DevOps, we can see that the movement popularity was and is rising. We can find, with a simple Google search, numerous blog posts about experiences and opinions regarding Devops and its adoption. Not only that, but there are also a growing number of *Devops ready* tools that promise to simplify and empower companies with the benefits of Devops.
Contrastingly, a similar search on *Scopus* will yield close to 200 results which is a much smaller list

of results when compared with other terms like *SCRUM* (more than 2000 results) or *Waterfall* (more than 700 results).
This lack of academic literature and study means that Devops understanding is still mostly based on opinions and personal experiences rather than scientific, peer reviewed literature. Consequently, adopting and practicing DevOps is still a surrounded with uncertainties.

Motivation

Devops represents a new way to look at the entire software pipeline. From development to delivery and maintenance, Devops represents an advantage for both teams and businesses by creating a more efficient, agile and collaborative way of working. DevOps also reduces the software time to market which provides a competitive advantage for those that are able to practice it.
Being such a strong driver of positive changes, we believed that studying Devops, its values and common practices, will enable broader adoption further strengthening the movement and both practitioners and businesses.

Goals

The main goal of this thesis is to increase the existent knowledge regarding DevOps in order to enable teams and companies that want to adopt DevOps with the required understanding of common pitfalls and solutions related with DevOps. At the same time further studies of the DevOps movement will be able to build upon this study by extending the current concepts or by having a base from which to search new ones.

Outline

This thesis documents a field study that tried to identify near Portuguese start-ups common practices and methodologies related with DevOps.
Chapter 2 introduces first the Cloud and then DevOps. This serves as, respectively, an introduction to some key concepts needed to understand this document and a base on which we based some of our study. The following concepts, Patterns and a characterization of the Portuguese start-up scene are serve as justification for some of the choices described later in the document.
The following chapter , describes how the study was made including the methodology used, what information was extracted, the filters applied to achieve the final sample and how the extracted data was extracted and compiled.
Chapter 4 shows the results of the field study. In this chapter, thirteen patterns are described as well as the relations existing between them.

Chapter 5 shows the approach of used to validate those patterns as well as the results of that validation. Finally,chapter 6 identifies the main contributions of this thesis and suggests some points that can be developed in future works.

Chapter 2

State of The Art

DevOps is usually associated with different types of technologies and practices. Effectively applying DevOps means not only to understand the cultural aspects but also the technological aspects that enabled some of its practices. In this chapter an introduction to cloud computing will be presented providing the needed context for the following section where a brief set of events involved in the rise of DevOps movement will be presented. After this, a small characterization of the Portuguese start-up scene will be presented in order to justify some of the choices presented in the next chapter 3.1.2.

Cloud Computing

Paraphrasing [3], imagine wanting to use electricity. Rather than building an entire grid and a power plant, one only needs to connect to the public network.In the common scenario, charges are calculated based on the usage amount and no special knowledge of how the network setup is needed.

Clouds follows exactly the same rationale, rather than having to build/purchase the computing resources, one needs only to connect to a provider and the resources are available to be used. Charges, like in the electric grid, are calculated based on the usage and clients do not need to know how the resources are managed or setup.

Definition

In [15], NIST[1] defines Cloud computing as *a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

NIST also defines the following essential characteristics of cloud computing:

[1] National Institute of Standarts and Technologie

- **On-demand self-service** : A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
- **Broad network access** : Users must be allowed to access resources through standard mechanism.
- **Resource pooling** : A multi-tenant model should be used in order to serve multiple users. Resources are allocated dynamically meaning that users do not know were, physically, the allocated resources are [9]
- **Rapid elasticity** : Resources can be elastically allocated or deallocate. This should be possible to be done automatically [15]
- **Measured service** : The usage of resources should be measured providing transparency in the provider-client relation.

Delivery methods

In regards to their accessibility, it is common to identify three main categories of clouds [24] :

- **Public Clouds** : Public clouds are a pool of resources hosted by cloud providers who rent them to the general public. This resources can be accessed over the Internet and are shared among users.
- **Private Clouds** : Private clouds are usually administered and used by the same organization. Alternatively a third party can also be hired to manage the resources. The main difference between public and private clouds is the usage of the resources. Private clouds resources are only used by one company as opposed to public clouds were resources are shared.
- **Hybrid Clouds** : Hybrid clouds combine both the private and public concepts. When using an hybrid clouds approach, infrastructure is divided by the two types of clouds meaning that some modules may be hosted in the private space and others on the public one.
- **Virtual Private Cloud** : Virtual private clouds are an alternative to private clouds. This type of cloud are essentially a public cloud that *leverages virtual private network (VPN) technology.*.[24] allowing users to combine characteristics of both public clouds and private clouds.

Service Levels

In terms of service levels cloud computing can be classified in regard to the provided abstraction. The main categories are the following [22] :

- **SaaS** - Software as a Service (SaaS) gives users access to a platform usually through a web client. Without the need to download or install software the user is able to use the provided software instantly and virtually everywhere.
  Applications of this model include messaging software, email services, collaborative platforms, etc.

- **PaaS** - Platform as a Service (PaaS) allows it's users to quickly deploy applications with little to no configuration. In this type of platform environments are usually setup previously or configurable. PaaS users should nevertheless expect only to be able to deploy applications or software supported by the provider.

- **IaaS** - Infrastructure as a Service (IaaS) represents the lowest abstraction made available by cloud providers. In this model the user is able to configure and access a machine directly without constraints. This machine, usually a virtual server managed by the provider, can be configured and maintained by the user.
  This model is used when applications are complex and therefore need complex configurations.

Benefits

The main advantages of cloud computing for its users can be summarized as following:

- **Monetary Efficiency** - Cloud providers allow users to keep their resources to the needed minimum. By allowing users to quickly and easily increase/decrease the allocated resources amount and billing clients only for the resources used, cloud providers are good way to save money and spend only the needed amount [9, 15].

- **Scalability** - usually through a public API of some kind most cloud providers allow for the quick increase or reduction of resources [15]. This enables businesses to quickly go from zero to millions of users with minimum overhead. Additionally because processes related with the management and configuration of cloud servers can be automated it is usually possible to manage large systems with small teams [14].

- **Maintainability** - Cloud Providers are responsible for the maintenance of all the hardware and infrastructure aspects. Cloud computing users therefore do not need to worry about updating the hardware or maintaining the physical infrastructure. This enables users to focus their resources in improving their product rather than improving the structure that supports it [9].

DevOps

In *Why DevOps Is Like An Onion* [5], Dave Sayers chooses to use the analogy of peeling an onion, with each layer representing a different concept, in order to describe DevOps. Further developing upon his initial idea, in this section, we will try to introduce DevOps using that same analogy.

The cultural layer

Looking at some of the first DevOps efforts we see that a lot of them aimed to create a better articulation between Developers and Operations [7] [1].
Being two distinct departments with different work methodologies and objectives, this meant that some cultural changes had to happen in order for the two departments to be able to create a common understanding of each other worlds [1].
The main cultural values associated with DevOps are, as a result, values that promote cooperation and communication. These are some of the ones that we identified:

- Respect [6] [1]
- Trust [10]
- Collaboration [6]
- Sharing [23]

The methodology layer

The DevOps movement does not define a specific methodology mostly since it belives that this choice should be made by the company/organization in order to better accomodate their needs. Nevertheless, when we look for instance at CAMS2.2.6 or at some of the practices like *Continuous Deployment* it becomes clear that, altough a methodology is not defined, the one chosed should enable an iterative and continuous improvement focused approach.

The practices layer

Broadly speaking, DevOps practices gravitate around the automation of processes. From the setup of environments on the developers machine, up to the deployment phase, the capacity to automate repetitive tasks is a key feature of DevOps. Common practices associated with the DevOps movement include:

- **Continuous Integration** - Regular integration of software helps discover risks associated with the integration with the integration of the software. [2]

- **Continuous Deployment** - Deploying often means that less functionalities are deployed each time. This makes deployments more manageable and in turn safer [1].
- **Continuous Monitoring** - Continuously monitoring infrastructure and applications allows for the early detection of bugs [**?**]. It can also be a way to identify areas that can be improved [23]
- **Defining infrastructure as code** - By defining infrastructure as code it is possible to increase reliability and consistency [14].
- **Using feature Flags** - Feature flags are special flags that toggle features on and off. Using this, can improve error handling (if a feature is faulty, it can be turned off. Feature flags also enable more complex schemes of operation where certain functionalities are launched but are not displayed or are displayed just to certain users. Once it is observeved that the new functionality works, the flag can be turned on and the new funtionality is available [3].
- **Others** - As we will see in 2.2.6 there can be a great number of practices that can be considered DevOps practices. This means that we would not be able to include them all here.

The tool layer

Tools allow for some practices to be more effective. In this section we will present some of the existent categories of tools. This categories are presented in [13] and are currently being maintained and increased by the DevOps community:

- Source Control Management (SCM)
- Continuous Integration (CI)
- Deployment management
- Cloud platform and infrastructure
- Monitoring
- Repository Management
- Infrastructure Provisioning
- Release management
- Logging
- Security
- Build
- Testing
- Containerization
- Collaboration
- Database Management

The full onion

As it was stated by Paul Hammon and John Allspawn at their *10+ Deploys Per Day: Dev and Ops Cooperation at Flickr* presentation, tools and processes alone are not enough and the cultural change should be the first step to take towards DevOps.
Because of this, the outermost represents the most important layer and without *peeling* it you will not be able to access the inner layers.

A DevOps definition

DevOps vastness can be seen in the enormous ammounts of tools categories identified in 2.2.4 related with DevOps. As one might expect from this vastness of areas that DevOps touches, there are still difficulties to properly define DevOps. In order to reduce this indefinition, we will use throughout this thesis a conjugation of two definitions for DevOps.

Figure 2.1: The full onion [5]

The first definition is from *DevOps: A Software Architect's Perspective* [3] and it states that:

DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality.

We chose this definition because it allows to easily classify something as being DevOps or not i.e. one would only have to ask himself if a practice or cultural aspect will allow for the reduction of time since commiting a change until that change is in production, if it does, then it is DevOps. Nevertheless, we find this definition to be a bit empty in the sense that by only reading it one would not be aware of the aspects that are associated with Devops. As a workaround for this problem we use a second definition based on the CAMS acronym.

The CAMS acronym [23] defines DevOps as being a **C**ultural movement where **A**utomation and continuous **M**easurement of processes and people are promoted and where the later can serve as a input for the **S**haring of problems and new ideas. As new ideas appear to solve the indentified problems, new measurements can be made to further identify problems furthering improving the overall process.

DevOps Benefits

In a recent study [8] some of the DevOps benefits were identified:

- DevOps projects are believed to accelerate in 15%-20% the ability to delivery of capabilities to the client
- Adopting DevOps allows business to practice Continuous Delivery.
- The average cost of a critical application failure per hour is $500,000 to $1 million (DevOps can help reduce application failures).
- The average cost percentage (per year) of a single application's development, testing, deployment, and operations life cycle considered wasteful and unnecessary is 25% (DevOps can help automate some repetitive tasks)

Patterns

Some previous progress has already been made regarding the identification of DevOps related patterns. We will summarize this progress by listing the identified patterns and briefly describing them:

- **Store Big Files in Cloud Storages**[**?**] - Instead of creating and managing a system to store large files, or storing them in database columns, store them in a Cloud Storage[2]
- **Queue based solution to process asynchronous jobs**[**?**] - When there are tasks that take a long time to complete but users still expect a quick response, create a new Job instance in a queueing service and then have a service performing those tasks. When finished, post the result of that Job somewhere acessible to the user and notify the user that the task is done.
- **Prefer PaaS over IaaS**[**?**] - For non technology companies, PaaS is usually preferable because it will give them lot of functionalities without the need for configuration. This will allow them to simply focus on their core business.
- **Load Balancing Application Server with memcached user sessions**[**?**] - Use a load balancer in front of your application servers. This severs will handle sessions using memcached which means that if a application server goes down or if a new application server is needed, it will be able to use the user session.
- **Email delivery**[**?**] - Rather than implementing your own SMTP solution, use cloud mail delivery services which provicde REST API's to send emails.
- **Logging**[**?**] - Having multiple servers you need a way to consolidate your application logs. In order to do so, you should use a cloud based log service.
- **Realtime User Monitoring (RUM)**[**?**] - Monitor user behaviour in order to find possible bugs or errors.
- **The isolation by containerization pattern** [18] - "Use a container to package the applications and its dependencies and deploy the service within it".
- **The discovery by local reverse proxy pattern** [18] - "Configure a service port for each service, which is routed locally from each server to the proper destination using a reverse proxy."
- **The orchestration by resource offering pattern** [18] - "Orchestrate services in a cluster based on each host's resource-offering announcements."

[2] A storage system provided by a cloud provider.

The Portuguese startup scene

Motivated by a recent investment in innovation and entrepreneurship, Portugal startups have been growing their position in the global startup scene [4].

A study from 2015 [19] in which the Portuguese startup scene was analyzed, revealed that there were already 40 technology scaleups [3] operating in Portugal at the time. The same study stated that this startups were able to raise a large portion of the received investment from international investors indicating, therefore, that the reach and scale of this startups was broader than just the national arena. Additionally, it is also indicated in the study that Porto and Lisbon are the main centers of innovation, encompassing 70% of the total of existing scaleups. In addition to the scaleups identified other smaller scale startups exist. Some of this startups are currently being incubated in incubators around the country. UPTEC [4] and Startup Lisboa, both business incubators. This incubators had, at the time of this study more than 300 companies [21, 20] under their wing.

A pattern language

Pattern languages have been used in the software development area in the past for describing common architectural solutions [12], recurring software design choices [11] and even describing pattern languages themselves [16].

Patterns represent a way to share recurring solutions to a problem [16]. As a result, pattern languages help reduce re-discovery and re-invention of concepts and functionalities as well as *avoiding common pitfalls that are learned from experience* [17].

[3] Scaleups are companies that raised more than $1M funding (since foundation) and had at least one funding event in the last five-year period

[4] Science and Technology Park of University of Porto

Chapter 3

Towards DevOps

In this chapter we will look at the approach used in order to solve the problem described in 1.2. We will start by looking at how we defined the sample from which we extracted information and then how we handle that data in order to produce the final results.

Methodology

In **??** we saw that the DevOps movement emerged in the software development[1] community. Having this factor into consideration and knowing that there is still a significant lack of scientific information regarding the subject, we choose to look for a solution within the development community. We believed, before conducting this study, that not only would they be able to provide us that information, but because they were using this techniques/tools in their daily activities, it would serve also an extra layer of assurance.

Collecting the information

Taking into consideration the vastness of topics that DevOps **??** touches, we knew that it would not be hard to create a form that covered that extensively. Instead, the chosen approach was to do go for a more exploratory approach. This idea lead to the creation of a script rather than a form that would guide the interviews. This script had 5 major sections:

- **Product** - The product section would try to understand, first what did the company do and secondly if there were any kind of special requirements that would influence what followed.
- **Team Management** - Team sizes, interactions, project management techniques would be analyzed here.

[1] software development should be seen in this context as both the development, maintenance and any other tasks related with the creation of software

- **Software delivery pipeline** - In this section, we identified if teams did Continuous Integration, how did they handled the creation of environments for each of the pipeline states and what teams did what in each state.
- **Infrastructure Management** - We tried to capture how the companies handled their infrastructer. Did they use the cloud? Which processes did they automate?
- **Monitoring & Error Handling** - With this section we aimed to understand if the companies were monitoring their infrastructure, how did they do it and, when errors were detected, how were they responding.

Defining a sample

As it was seen in 2.3, Portugal as a rich startup community. Startups have strict contraints regarding the ammount of resources they have at their disposal and have, as a result, aditional incentive to automate as many tasks as they can. Being small companies(in terms of staff) communication and cultural aspects are usually guided towards cooperation as this is a key factor in allowing small teams to handle large projects. Finally, the fact that startups have as their objective to scale further highligh the need for automation and cooperation. This conjugation of factors meant the mindset of startups was aligned with the DevOps one and startups are therefore a good place to look for information that can be directly linked to DevOps.

Having more than 300 startup companies 2.3 from which to choose, there needed to be a way to reduce the size of the sample. With that objective we listed and attempted to identify if a startup was doing software development or not. To do so we looked to, when available,at the company web page and tried to determine if the company had staff members working on software development task. When we could not find a team page, we also looked at the Linkedin[2] companie profile and did the same thing. Companies that had software developers or software related productswould go to the next round. We were able to reduce the sample to 155 companies.

Because the ammount of time we had was limited, it would not be possible to interview those 155 companies. We choose, in this phase, to prioritize which companies were better or worst for our study. To do so we created a compound evaluation metric that would allow us to rank companies. We created 5 metrics to do so:

- **Cloud Usage** - We created three possible values for this metric. If a company used cloud services, we would give the company 2 points. If we were not sure if a company was using cloud services, we would give it 1 point. If we know the company was not using cloud services, we

[2] www.linkedin.com

would give it 0 points. With this metric we attempted to distinguish between, for instance, companies that were developing hardware solutions from those that were developing more software oriented solutions.

- **SaaS/PaaS offering** - Having the same point attribution schema as the *Cloud Usage* metric we believed that if a company had a SaaS/PaaS product, it would need to have some sort of automation put into practice as it would need to be able to scale if there was a sudden increase in clients.

- **Company Size** - We create four possible values for each metric. Companies could have 0,1,2,3 points if they had respectively less than 5 members, between 5 and 15 memebrs, more than fifteen members or more than fifteen members and several teams.

- **Subjective Appreciation** - This metric would reflect the overall opinion of the company that we developed when searching for information for the other metrics. Some common factors that influenced this metric were for instance the fact that some companies had no software developers, or the company website was down.

This ranking is not supposed to be seen as a precise way to accurately compare companies i.e. the second company maybe more relevant than the second one, but rather as a way to prioritize them i.e.the first company surely is more intersting to study then the last one.

In the end of the study, we contacted 60 comapanies (of the remaining 155) from which we were able to interview 25.

Processing the data

After conducting the interviews the next task was to analyze and process the collected information.

We started by creating a set of concepts/techniques that were observed. This concepts/techniques were then analyzed and we extracted their frequency in the data set. It became clear at this point, that, because of the size of the sample (25 companies), it would be difficult for most observations to be statistically significant as some practices were only identified one or two times. This lead us to the next option that would be to identify the practices that were relevant to our study and that embodied the DevOps mindeset. Using this criteria we were able to identify a total of 13 practices.

Chapter 4

Patterns from the Portuguese Startup Scene

The following patterns represent the result of the carried out interviews.

Team orchestration

Context

The environment were you are in can evolve at a fast pace. In order to be effective, you want your teams to be able to adapt and respond quickly to those changes.

Problem

How do you orchestrate your team(s) so that they can handle new challenges and deliver results in a sustained manner ?

Forces

- Specialized teams are faster at answering specific problems related with their speciality.
- Specialized teams will not be able to deliver a final product if it requires skills beyond their speciality.
- Specialized teams must combine their efforts with other specialized teams to deliver the final result.
- Objectives for specialized teams may conflict with other specialized teams (e.g. the team responsible for the performance of an application may not agree with the user experience team when the later wants more content in a specific page).

- Multidisciplinary teams can deliver a final product if they have all specializations represented.
- To be effective, members of multidisciplinary teams must be able to work together.
- It may not be possible to have all expertise constantly working in parallel in the same team.

Solution

Multidisciplinary teams are able to articulate and communicate in order to deliver a product that takes into account several constraints and requirements specified by each of its members specialization. Additionally, teams that have representatives of the expertise needed to deliver the final product are able to deliver it
In terms of **size**, these teams should not go beyond 10 elements. This constraint comes from the need that multidisciplinary teams have to be able to communicate effectively. Team members should be seen as equals and no hierarchical structure should be imposed inside the team.
Occasionally, due to changes on requirements, some team members can be shared with other teams.

Communication

Context

Each member of a multidisciplinary teams have **different knowledge** and **background** that differ from each other.
Professionals working in this kind of environment will sometimes disagree as they guide their work by different constraints and goals. **Facilitating** the resolution of this discussions and promoting the **sharing** of **knowledge/view** points is, therefore, key and solutions must be found to promote this.

Problem

What kind of approach should you adopt to promote communication and facilitate it?

Forces

- You want to allow people to communicate easily.
- When concentrated on a task some people do not like to be interrupted.
- Some information (e.g. links and code) may not be easy to share verbally.
- Sometimes you want the content of the communication to be made available.

Solution

- **Direct Communication** is efective in handling day to day problems (e.g. solving doubts, giving advice, asking for help). Having the teams physically working together is a great way of promoting this type of communication.
- **Chat tools** allow you to share links, files and code quickly. Sometimes chat tools are also useful if you need to speak with someone and you do not want to disturb him.
- **Emails** can be used for sharing information that is not urgent(e.g. scheduling a reunion for next week). Additionally, emails can be used if your communication needs to be stored like when speaking with an outside provider or with a client.

Related Patterns

This pattern is related to both the *Continuous Integration* and the *Alerting* in the sense that the same channels defined here can be used to send the messages those systems generate.

Version Control Organization

Context

As more people are working on the same team and contributing to the same product it becomes increasingly difficult to manage and synchronize those contributions. Tools like Git, SVN and Mercury are helpful on dealing with this kind of problem.

You have chosen to use Git either because you believed it was the best fit to your project and you have the need to know:

- What is the code in each of my environments (e.g. production, development) ?
- What was the code developed for a specific feature ?

Problem

How do you setup your version control branching strategy so that you can infer valuable information about your current state and past events ?

Forces

- Having too many branches may be complicated to manage or cause confusion.
- Having too few branches may make you loose valuable information.

Solution

There are several ways you can manage branching. The main ones are:

- **GitFlow** specifies that at any given time two branches should be active. This branches are the **master** and **develop** ones. The code present in the master only contains shippable code. The *develop* branch contains the most recent working version of your code. This branch should not contain non working code but it may contain, for instance, features that have not been through a QA process. Adding to this two branches there are additional branches called *feature branches*. This branches represent a new feature under development and there should be one *feature branch* per feature. When a feature is implemented it should be merged into the *develop* branch. If that feature and the previous ones are considered production ready then the *develop* branch should be merged into the *master* one.

  Finally, if at some time you need to create an hotfix, you can do it by creating a new *hotfix* branch with the content of the *master* branch, applying the changes and merging it back into the *master* branch.

- **Feature Branches** can be seen as a subset of the **Gitflow** strategy. Instead of having a *develop* branch, this strategy only uses the *master* and *feature* branches The *master* branch holds tested and functioning code, the *feature* branches (one for each feature) hold the code of the correspondant feature. When a feature is ready and tested it is merged from the *feature* branch into the *master* branch.

Related Patterns

It is possible to use the *Code Review* pattern even without using any sort of version control. Using one, will nevertheless help organize the process.
In the same way, *Error Handling* can also be improved by using the correct *Version Control Organization* as it will allow you to for instance to traceback what was the last live version or what was the code introduced in an hotfix.

Cloud

Context

Your company and/or your product needs to acquire computing resources in order to perform tasks like:

- performing **large complex operations**.
- supporting a **website** or a **web platform**.

- any other kind of **computing task**

This resources should be accessible and configurable and you believe that you do not need to physically connect to them it in order to control them.

Problem

Owning computing resources is essential or at least advantageous to you or your business so the question is how do you acquire and maintain computing resources in a efficient way ?

Forces

- Acquiring hardware may require significant upfront costs.
- Depending on the ammount of hardware you have to manage, a person, team or department may be needed to maintain it.
- Different services may provide different levels of customization/control.
- You may want to scale the amount of allocated resources to match your needs.
- Applications may have very specific needs both in terms of harware and environment where they run.

Solution

Solutions for this problem can be seen as belonging to three categories:

- **Purchasing** and maintaining your own **hardware** allows you to have full control over your infrastructure. You can control, for instance, in which machine does a specific applications run, how that machine is configured, etc. This option represents therefore the **highest level of customization** and **control**.
  On the **downside**, this options usually means that you **have to purchase hardware** and that you either **acquire more resources than what you need** or you risk **not having enough resources** to answer increasing computing needs. Additionally you will have to **create** and **support** a team or department to **manage the infrastructure**.

- When using **IaaS** there is no need to purchase anything upfront. In this **pay-as-you-go** model you only pay for what you consume and you are able to **increase/decrease** the size and/or number of **resources** you are using. With this model the responsability for **maintaining and setting up infrastructure** is shifted to the cloud provider.

IaaS providers usually allow you to have some degree of customization like choosing the operating system and resources available(CPU cores, memory, etc) but lower level configurations will not be available. As a matter of fact, most cloud providers use virtual machines to run their clients applications meaning that you will not be able to tweak network configurations or choose exactly wich machine runs what. IaaS **reduces** therefore the **level of control** in comparisson to hosting your own infrastructure.

- **PaaS** follows the same **pay-as-you-go** model as as IaaS meaning that you can also **increase/decrease** the size and/or number of **resources** you use.
  PaaS represents the **smallest** level of **customization** but at the same time allows you to use already **pre-configured environments** in which you can run your applications.

Related Patterns

When using the *Cloud*, choosing for instance a PaaS alternative may, depending on your choice, prevent you from being able to fully reproduce the production environment. Different levels of support exist, as well, for different *Reproducible Environments* techniques. Some providers may, for instance, create pre-programed container ready environments while others may not allow you to run your own Virtual Machine.

Reproducibe Environments

Context

When you have several environments (e.g. production,staging,devevlopment) or multiples instalations/instances of your software it is desirable to be able to guarantee that all instances work the same way. With this goal in mind you have identified that the environments where your instances run is a key factor when trying to antecipate how does the software behaves.
This consistency is important because it will allow you to have reproducibility and will give you some guarantees when you desire to increase the number of instances of your software.

Problem

How do you guarantee that the environment where you setup your application is consistent across instances?

Forces

- Having a complete copy of your environment (OS's, libraries, etc) may create large files that may be hard to move around.

- You may want to have several running instances of different environments in the same machine.
- Some of your dependencies may be fetched from external providers.
- You may want to update or change the environment.
- Depending on your choice for *Cloud* you may have more or less access to your environment settings.

Solution

- Using **scripts** usually means **describing you environment** in the form of a text **file**. This **file** is then **executed/interpreted** inside an **environment** in order to create the desired state.
  Because scripts do not contain the dependencies you need, you usually have to **rely** on **external providers**. If for instance one provider shuts down your script will not be able to complete.
  In case you need to **update** your setup, depending on the change and the tool you use you may need to **run** the script **again**, run only the part you modified or reset the machine and run everything again.
  Because scripts are just text files they often represent the most efficient alternative in terms of memory.

- Using **virtual machines** an environment can be created by creating an image of the operative system with all dependencies installed. This image can then be replicated across different projects. If the need to change dependencies arise a new image can be created.
- **Containers** 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.

Related Patterns

Having chosen to have environment consistency across your instances means that the when creating new instances(*Deploying new instances*) and building your software (*Continuous Integration*) the same choices should be used.

Deploying new instances

Context

You have decided to increase your computing resources **horizontally** in order to increase your hability to handle a bigger load of tasks. Depending on what type of *Cloud* you choose to use new resources were allocated but you still need to have your application running on those resources.

Problem

How do you deploy your application in a reproducible and consistent way ?

Forces

- Deployments must be reliable.
- Deployments should not waist time.
- Deployments should correctly articulate with your environment setup method.
- Deployments should be easy to manage.

Solution

In order have an efficient deploy both in terms of reliability and speed you should have some sort of reproducibility. Depending on your choice for setting up environments (*Reproducible Environments*) there are different ways you can manage the deploy:

- If you have chosen *Containers* you can simply pull the container from a container registry and run it in your new instance. With this approach you will have a high degree of certainty that your instance will behave as you predict. Because containers are generally lightweight you will be able to download them fairly quickly. You will, nevertheless be dependant on you container registry service.
- If you have chosen *Virtual Machine* you can create and image (AWS lets you crete a Virtual Machine using their services) and then deploy it to all your instances.
- If you have chosen *Scripts* you can simply run the script in your target machine.

Related Patterns

Depending on your choice for *Reproducible Environments*, you should use the appropriate choice to deploy your code. You should be aware of the cost of this choice because if you want to scale horizontally, your performance maybe tied to the velocity with which you can provision new environments.

Scalling

Context

Having a 1:1 ratio between your needs and your resources may be easy to achieve if your needs are fixed in time. If, however, your needs fluctuate as a result of, for instance, new users accessing your application you would want to be able to **increase** or **decrease** (in case users numbers drop) the **resource** allocated.

Problem

What strategy do you choose to increase your computing power ?

Forces

- Costs are a factor.
- You want to change the allocated resources quantity without having to stop the existing application(s).
- Your application may have need to keep state.
- You may not have an upper limit for the amount of resources you will be using.

Solution

Usually if your are using the *Cloud* you can easily allocate new machines or increase the CPU cores, RAM, Disk Space, etc of your current machine(s). This two options represent the two existing approaches to scale your computing resources. The first one (increasing the number of machines) is usually refered to as **Horizontal Scalling** and second approach (increasing the resources of each machine) is usually refered to as **Vertical Scalling**.

Horizontal Scalling usually is the **cheaper** option and allows for **no downtime** when upgrading (the existing machine can be put into production while the old one is running). This approach will also allow you to **scale** virtually **without** a **limit**. On the downside, this approach will force to have some considerations in mind concerning state keeping. If you have a need to keep sessions, for instance, and you are storing them in the machine, the new machines will not have access to that.

Vertical Scalling is usually more expensive and depending on your *Cloud* provider may have associated downtime. **Verical Scalling** also has a maximum amount of resources you can allocate to a single machine. On the upside if you scale vertically(and have only one instance) you can keep the state of your application inside your machine.

Both approaches can be combined in order to accomodate your needs.

Continuous Integration

Context

There are several people contributing code to your application.

Problem

Having several people collaborating into the same project can be challenging. If a developer, unaware that is introducing an error, pushes code it into the team repository a long time may pass before the error is detected. Once detected, the error cause must identified and, because the code that introduced the error was pushed a long time ago, it may not seem obvious where the error is.

Forces

- Running your entire test suit before pushing code may take to much time.
- It can be challenging to setup an environment simillar to the production one in your local machine.
- Your environment may need to be different from the production one(you may need some extra tools to aid you developing).
- Your environment may be subject to bias (ex: case where you may manually set an environment variable that you code uses).

Solution

Use (or develop) an automatic continuous integration(CI) system. This system should detect when code is pushed to your repository and then run the following steps:

- **Build** . Building your software consists in, depending on your choice for *Reproducible Environments*, building your environment, then fetching all required dependencies and finally compiling the code(if needed).
- **Test** your build. When your build is successfull you should run your test suite against that build in order to check if everything is running according to plan.
- **Notify** If any of the previous steps fails you should notify the developer that checked the code and any other people to whom the build integration status is relevant.

Related Patterns

The *Continuous Integration* pattern can use the *Communication* defined channels to send its messages. The test and build steps of this pattern, shoud be done in a environment equal to the production and development one and should follow the chosen *Reproducible Environment* strategy.

Job Scheduling

Context

Sometimes there are tasks that, due their complexity may take a long time to finish. Cases may also exist when you have tasks that you want to schedule for later(e.g. maintenance tasks may be runned at a time when your application is under).

Both this problems can be solved by scheluding jobs to be run when possible or later.

Problem

How do you setup your infrastructure to handle this cases?

Forces

- Having a fixed set of resources for dealing with scheduling tasks may not be cost effective.
- Your load may vary during the day.

Solution

You may launch new instances of your infrastructure to handle each of you tasks or batches of tasks. Each new instance receives the desired tasks and does the needed computation. When the task as been computed the new piece of infrastructure should be shutdown.
Alternatively you can have a set of daemons running alongside your applications that handle this tasks. Tasks are generally fed through a queing system altough you can also store them in a database.

Related Patterns

When you are launching new pieces of infratructure to handle your jobs the effectiveness of that technique may be dependant on how fast you can *Deploy new instances*.

Auditability

Context

As applications grow identifying potential problems within your infrastructures will become increasingly difficult. If you have several machines and/or different possible points of failure you can not predict or assume that everything will always go without incident and you will therefore have to be prepared. Building a robust system may seem enough but is not. When problems appear (and they will appear) being able to identify them ,where and why they appear is essential for the resolution of those problems.

Problem

What metrics should you extract and what should you do with them?

Forces

- Extracting too many metrics may cluter your hability to effective analyse them.
- You may want to keep an history of how your system behaved.
- You want to have information about the current state of your service.

Solution

Monitoring your application health can be done by using your own or external tools. Some cloud providers even provide you with a health view that tells you if your machines are healthy and running. Identifying some key indicators and some metrics is also important, by defining thresholds for each metric you can setup different levels of alerts for your teams. This way you can tackle problems as soon as they happen. Additionaly some indicators can also trigger automatic responses (e.g. if a platform is taking to long to answer requests you may launch new resources to distribute the traffic).

Related Patterns

You must be monitoring some metrics in order to create alerts. This mean that the *Alerting* pattern will only exist if there are metrics being monitored.

Alerting

Context

You have defined a set of metrics for checking the health of your application. For some of those metrics when values reach a certain level a solution can not be automated (e.g. server repetitive failures). You still would want an immediate responese to that alert in order to make sure your services will not go down or in order to put them back on.

Problem

Who are you going to call?

Forces

- People may not be available to answer alerts or may be unreacheable.
- Alerting everyone may solve your problem quickly but may not be needed.

Solution

Notifying can follow three main strategy. The first one is to wake everyone up. This approach is wastefull and as you usually do not need your entire team to solve the problem. The second one is always notifying the same person. This person should preferably be someone capable of diagnose the origin of your problem and then solve it or contact someone that can. The third option is two have a system were the responsability of handling errors rotates among the team members. Alerts can be sent using email, calling people, sending an sms and/or sending a notification to the person you want to notify.

Error Handling

Context

It is important not only to be able to detect errors(*Auditability*, *Alerting*) but also to be able to respond to them in a proportional way.
As someone involved in a team developing a product, being able to find the best way to handle a crysis may prove to be fundamental.

Problem

What alternatives do you have to handle an error ?

Forces

- Errors have different degrees of impact.
- The cause of an error may not be easy to find.
- You may have a working backup of your application.
- Sometimes you can not use a previous backup (e.g. when you have removed a column from your database).

Solution

Handling and error is a delicate task. There are several things that need to be taken into account.

If an error as a direct and significant impact in your applicaion (e.g. there is an error that allows people to login without checking the users passwords) you would want to respond as quickly as possible. In this types of cases you can **rollback** to an older version of your software. The rolling back effectiveness is nevertheless constrained by the speed with which you can do it and by the fact that you may not be able to do it. Strategies for rolling back you application can be of two types:

- **Deploying the previous version**: You can order your system to deploy a version of your software that you know works.
- **Keeping a backup** : You can keep a backup of your application/infrastructure and if an error is detected you can switch the DNS servers to point to your old infrastructure.

When you can not roll back and/or the error you detected does not have a substancial impact you can try to find and fix the problem. Depending on your choice for *Version Control Organization* you can create a newer **hotfix** branch and work on that. In the end, when you have found and fixed the problem merging that branch with master and deploying the version will have fixed your problem.

Related Patterns

When you choose to rollback of to do an hotfix, you will be limited by the time it takes you to change the running version on your machines. If this change includes changing the environment you will need to update it. In order to do so, you will probably have to use the same method you defined for *Deploying New Instances*.
Additionally, rollbacks may depend on the fact that you can detect what your last working software version was. This information can be store in your version control system if you used tags for instance. Information about the hotfix (if you chose to do one), can also be stored in you version control system(*Version Control Organization*)

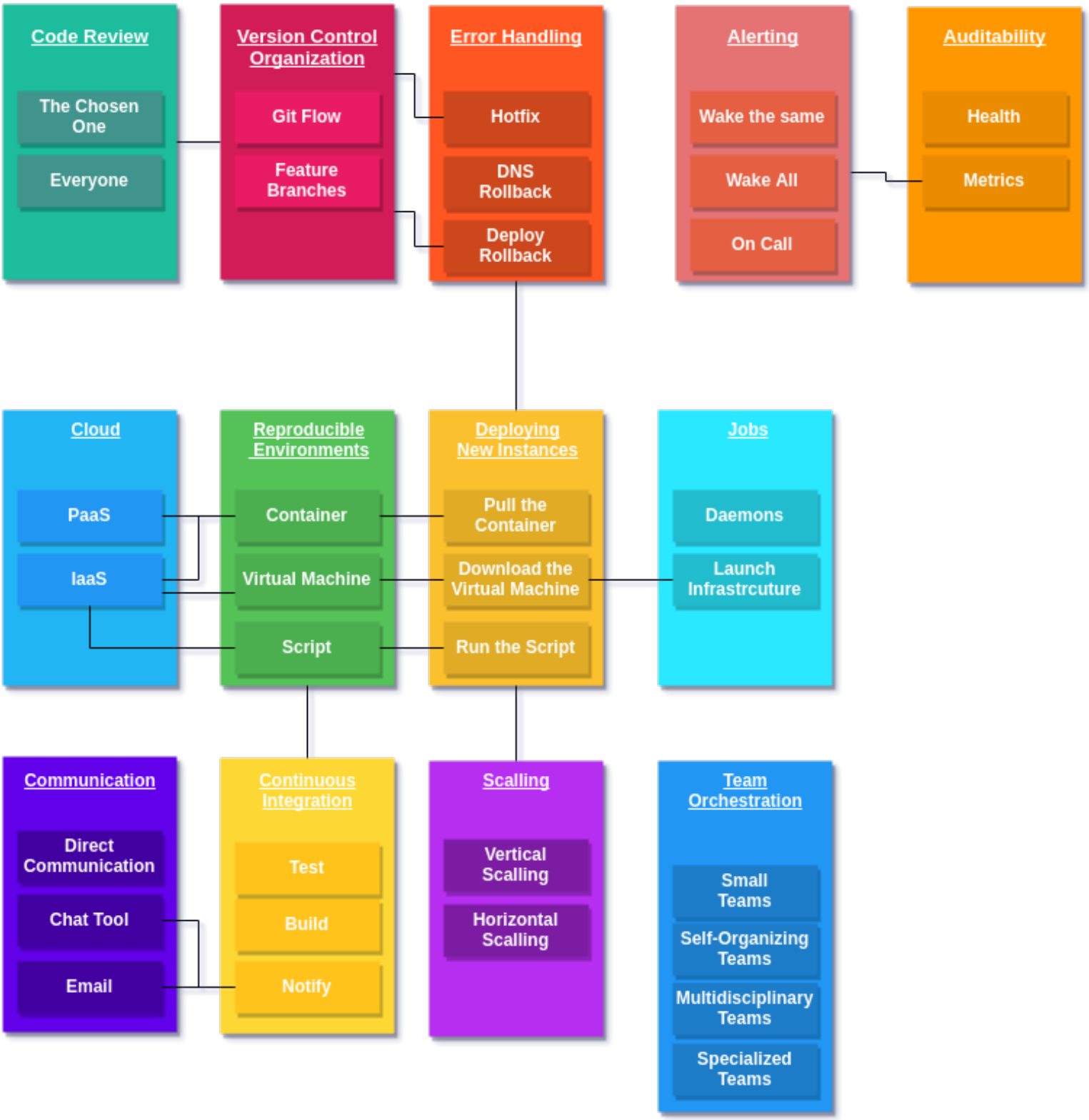

Figure 4.1: Pattern Map

Chapter 5

Validation

In this chapter we describe the validation of some of the collected patterns at Ventureoak [1]. In the beggining, we describe the Ventureoak core business and the methodology used to validate the identified patterns.
We then show, for each pattern, what were the measured changes.

Ventureoak

VentureOak is a startup currently operating at UPTEC, Porto that started its activity in 2014.
Having more than 20 employees, Ventureoak focus is on developing software products for other companies and in offering consultancy solutions both in the product idealization and developing phases.
At Ventureoak projects have usually a short duration - 3 to 6 months - and they usually target the web market.
Several projects are developed concurrently at Ventureoak by teams of 2 to 6 elements. This teams are self organizing and are usually made up of only developers.
Operations at Ventureoak are mostly managed by one of the employees altough access is provided and used by the other members of the company. Usually, infrastruture is handled manually.

Methodology

In order to validate the collected patterns, there were two approaches used.
The first was to take advantage of some of the changes that were already ocurring at Ventureoak. This changes, like the *Code Review* pattern were already being implemented when we arrived and in order to take advantage of that we tried to find, near the team what did they feel changed when this

[1] www.ventureoak.com

techniques were applied.
The second approach used consisted in applying the patterns and then measuring what changed.

Application

Pattern Validation

Cloud

Before

The cloud pattern was already being used at Ventureoak. Ventureoak prefered to use the IaaS alternative which allowed them to fully control their infrastructure. This, however forced Ventureoak to handle their own infrastructure monitoring setup. At the time this study was made that meant that a New Relic instance was installed in the used servers. Elasticity was not a priority and so there was no automatic way of scalling.

Implementation

Due to the choices made for *Reproducible Environments* and *Deploying new instances* as well as the objectives for *Continuous Integration* , manually handling the deployment process was not desirable. As a result, we choose to use a PaaS alternative instead and used AWS Elastic Beanstalk.

After

Because of the usage of AWS Elastic Beanstalk, we were able to have, without having to spend time implementing it:

- Automatic restart in case of error was already implemented.
- Load balancing was enabled from the start.
- Automatic scalling could also be configured from the start.
- Automatic deployment management.
- Automatic rollbacks.

Code Review

A code review process was already underway at Ventureoak. This process was conducted as a way for members to share opinions and help each other grow faster.

When we talked with Jorge Meireles, the Ventureoak Head of Development, he confessed that the process was not being adopted by the entire team and that the efectiveness of the process was therefore not being fully potentiated. Nonetheless, Jorge told us that by reviewing the code produced by the team he was able to reduce code duplication and have a broader view of the project.

Team Orchestration

Teams at Ventureoak are usually small (between two and six members). This meant that communication was facilitated and because this teams were usually multidisciplinar, members were able to be always active and always contributing to the final product.
When new projects were started, this also meant that no special members could be easily allocated.

Reproducible Environments

Environments were setup manually and in the developers machine. This meant that often, problems would arise everytime a developer changed the version of a dependency like the gulp version or nodejs version. This problem would usually be quickly solved once identified. In production environments, however, it had happen that a service was deployed and the required dependenies were not installed meaning that the application did not work correctly.
Also, newcomers, usually took, depending on their previous experience, one or two days to fully install all of the needed dependencies.
With the release of PHP7, a new problem was also starting to appear. Developers would want to use this new, more performant version but not all projects could be upgraded. For some developers, this meant that they would have to have in their machines two distinct installations which sometimes generated some errors.

Implementation

In order to provide the team with a way to easily reproduce their environments we used containerization. This approach was used because we would need to run several services at the same time and because we would need different configurations for each service.
This was done by using Docker which provided a way to describe the environment through a Dockerfile.

Results

With this approach we were able to give the team a way to share their environment. Everytime a change was made, they would just have to push the new file that descibed the environment and, when other team members would pull that file, the environment would be easily updated.
For newcomers, setting up a project was now a matter of minutes. They would just have to install

Docker and then running a command in the shell, the entire environment would be created. Additionally, developers could now have different services running different PHP versions without needing to manage the extra complexity of having two distinct versions running on the same machine

Continuous Integration

Ventureoak had no stagin environment setup. This environment would house different projects and would create

Implementation

In order to implement a Continuous Integration pipeline we begun by choosing a CI tool. The choice fell upon GoCD. The creation of the pipeline took around 2 weeks. The time it took was mostly due to the fact that the pipeline included and automated deploy to a test environment in AWS which took around a week to fully automate.
The setup phase also took advantage from the *Reproducible Environments* choice and by using containers we were able to setup the system and easily install it in a different machine.
The CI pipeline would run on every push to the project repository and would have three phases. The first would create/update the needed configurations in the AWS Elastic Beanstalk service .The second would consist in building an container image and pushing that container to a registry. The third and last would be to trigger a deployment in the AWS Elastic Beanstalk Service.

Results

Before the creation of the CI solution a normal deployment could take between 5 and 20 minutes depending on the need to install newer dependencies or run some assets minimization tasks. This deploy would be managed by a developer and done by manually accessing the server and updating the code. This task would be done every time an external clients or internal project manager wanted to check if a feature was correctly implemented. Sometimes, developers forgot to clean the server cache and/or compile some assets and would leave the application in a inconsistent state.
By implementing the CI pattern we were able to remove the need for developers to manually handle the deployment and to enabled project managers to choose what to deploy and when to deploy. This automated deployment would usually take 5 minutes if the build and push step was already previously done and around 15 to 20 minutes if the build and push process had to be done.

Deploying new instances

The deployment process followed the *Reproducible Environments* approach. We would not be able to measure any indicator of success for the application of this pattern.

Scalling

Taking advantage of the automatic scalling options provided by the choice for *Cloud* we were able to setup a scalable application without the need for extra work.

Auditability, Alerting, Error Handling

Having only deployed to a stagin environment, we were not able to identify metrics or situations related with this patterns.

Chapter 6

Conclusions

Contributions

As of this moment, DevOps is still evolving and as more people join the discussion more perspectives and experiences will become part of the movement. Regardless, we believe that the initial work made in this thesis towards a more structured way of presenting and talking about DevOps will help reduce the barriers for new adopters and help the movement grow by providing a common dialect for discussing DevOps. This work can also be seen as a starting point for further investigation.

Future Work

Specializing the identified patterns

While pursuing the study of DevOps we kept a high level of granularity with the objective of capturing a wider view of the movement and its practices. We believe that having done this work, each pattern should be further analyzed in a more precise way.

DevOps monitoring

As refered before, the DevOps movement is still evolving. We believed that, by monitoring the DevOps movement it will be possible to identify more practices being employed by companies and individuals.

Further validation

The initial results observed hint that DevOps and its practices can bring benefits to both companies and individuals. In this thesis we were not able to fully validate all of the identified patterns and we also did not observe long term effects of this developments in both teams and companies. Validation

was also done with only one team and one company which is insufficient to prove that those benefits can be reproduced in other teams or organizations.

Bibliography

[1] J. Allspaw and P. Hammond. 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr.

[2] S. S. And and B. Coyne. *DevOps For Dummies*, volume 53. John Wiley & Sons, Inc., 2015.

[3] L. Bass, I. Weber, and L. Zhu. *DevOps: A Software Architect's Perspective*. 1 edition, 2015.

[4] A. Coleman. Portugal Discovers Its Spirit Of Entrepreneurial Adventure, 2015.

[5] Dave Sayers. Why DevOps Is Like An Onion, 2013.

[6] J. Davis and K. Daniels. *Effective DevOps*, volume 1. 2015.

[7] P. Debois. Agile infrastructure and operations: How infra-gile are you? *Proceedings - Agile 2008 Conference*, pages 202–207, 2008.

[8] S. Elliot. DevOps and the Cost of Downtime: Fortune 1000 Best Practice Metrics Quantified. *IDC Insight*, (December), 2015.

[9] G. Garrison, S. Kim, and R. L. Wakefield. Success factors for deploying cloud computing. *Communications of the ACM*, 55(9):62, sep 2012.

[10] M. Hüttermann. *DevOps for Developers*. Apress, Berkeley, CA, 2012.

[11] R. Johnson, E. Gamma, R. Helm, and J. Vlissides. Design patterns: Elements of reusable object-oriented software. *Boston, Massachusetts: Addison-Wesley*, 1995.

[12] M. Kircher and P. Jain. *Pattern-Oriented Software Architecture, Patterns for Resource Management*, volume 3. John Wiley & Sons, 2013.

[13] X. Labs. Periodic Table Of DevOps Tools.

[14] M. Loukides. *What is DevOps?* "O'Reilly Media, Inc.", 2012.

[15] P. Mell and T. Grance. The NIST definition of cloud computing. *NIST Special Publication*, 145:7, 2011.

[16] G. Meszaros and J. Doble. A pattern language for pattern writing. *Pattern languages of program design*, pages 1–36, 1997.

[17] D. C. Schmidt. Using design patterns to develop reuseable object-oriented communication software. *Communications of the ACM Special Issue on Object-Oriented Experiences*, 38(10):65–74, 1995.

[18] T. B. Sousa, F. Correia, and H. S. Ferreira. DevOps Patterns for Software Orchestration on Public and Private Clouds. page 11, 2015.

[19] Startup Europe Partnership. SEP MONITOR PORTUGAL RISING: MAPPING ICT SCALE-UPS. 2015.

[20] Startup Lisboa. Startups.

[21] Uptec. Companies.

[22] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds. *ACM SIGCOMM Computer Communication Review*, 39(1):50, dec 2008.

[23] J. Willis. What Devops Means to Me, 2010.

[24] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.