Towards DevOps: Practices and Patterns from the Portuguese Startup Scene

Carlos Manuel da Costa Martins Teixeira

DISSERTATION PLANNING

U.PORTO

FEUP **FACULDADE DE ENGENHARIA**
UNIVERSIDADE DO PORTO

Towards DevOps:
Practices and Patterns from the Portuguese Startup Scene

Carlos Manuel da Costa Martins Teixeira

Mestrado Integrado em Engenharia Informática e Computação

June 18, 2016

Abstract

Software projects have been increasing both in complexity and in size across the years.

As more businesses and opportunities arise, so did the challenges and the need for better and more robust practices and methodologies. In 2001, with the writing of Manifesto for Agile Software Development and the subsequent movement we would see new ideas that tried to solve some of the known problems with software development. This new set of ideas and methodologies would, nevetheless do nothing to change the everlasting conflict between developers and operations.

As years passed and more companies moved online complexity grew and an extra pressure would be put both on the operations professionals and developers. Attempts to heal this divide would eventually culminate in the appearance of DevOps.

Formed by combining the word Development with the word Operations, the "DevOps" is a movement that aims to change the way companies work. An amalgam of both cultura changes, new technologies and new practices, DevOps aims to change the way companies and teams structure themselves in order to promote a healthier work environment and at the same time increase the value delivered by teams.

Interpretations of the movement and what it means are still topics of discussion and written academic literature on the subject is still scarce.

In this article we study DevOps, first by using the existing literature as the basis to create a DevOps state-of-the-art and then by analyzing the current practices associated with DevOps in the real world.

The analysis, done by interviewing and observing 25 Portuguese startups is the basis from which we then extract a set of patterns related with DevOps and its values.

Contents

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

Chapter 1

Introduction

Altough DevOps ideas are not recent (DevOps ideals can be traced back as far as 2008) academic and peer reviewed literature about the subject is scarce.

Having notice that, we aim, in this thesis, to study DevOps and the way it is being applied near the industry. With this study, we aim to enable those that want to adopt DevOps and those who want to study DevOps with a starting point on which to build.

Context

Developing and maintaining/operating software are often disjoint tasks and responsabilities. This pattern has been observed in several organizations and project management techniques. 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 [8].

As a result of this separation, two separated departments were born. This two departments, Development and Operations, would not be able to efficiently articulate preventing businesses from beeing as efficient as they could be.

Devops appeared within this context with the objective of solving this lack of articulation between those that developed and those that maintained/operated.

Problem

The DevOps movement spans throughout a vast set of areas and tries to change both techniques and culture. This variety means, nonetheless that Devops can offer an integrated approach that takes into consideration several aspects of the software pipeline but, at the same time, makes it hard to fully study and understand.

Looking at the current state of DevOps, we can see that the movement popularity was and is rising. We can find, with a simple search, numerous blog posts made both by individuals and companies about their experiences and opinions regarding Devops an 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 simmilar search on a search engines like "Engineering Village" or "Scopus" will yield a much smaller list of results.

This lack of academic literature and study means that Devops understandment is still mostly based on opinions and personal experiences. Consequently, adopting Devops is not trivial.

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. From a more business centric perspective, DevOps also reduces the software time to market which provides a competitive advantage for those that are able to practice it.

With that in mind, we believed that studying Devops, its values and common practices, will allow for a broader adoption and further strengthening of the movement.

Goals

The main goal of this thesis is to increase the existent knowledge regarding DevOps. With this increment, teams and companies that want to adopt DevOps will be able to better understand 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.

Contributions

Taking into account the goals defined for this thesis, we defined the contributions of this thesis to be:

- A comprehensive description of the current state of the art regarding the DevOps movement.

- Set of patterns and good practices concerning DevOps.

Outline

This report was made as preparation for the dissertation.

Chapter 2

State of The Art

In this chapter we will first look at what is the Cloud and Cloud computing. This initial section will provide the needed context for what follows where we will describe DevOps based on some of the work already done.

Cloud Computing

Paraphrasing [4], imagine wanting to use electricity. Rather than building an entire grid and a power plant, one only needs to connect to the public network. Charges are calculated based on the usage ammount and no special knowledge of how the network setup is needed.

Clouds follows the exact 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 [9], 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 essencial characteristics of cloud computing:

- **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.

---

[1]National Institute of Standarts and Technologie

- **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 [6]

- **Rapid elasticity** : Resources can be elastically allocated or deallocate. This should be possible to be done automatically [9]

- **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 [15] :

- **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.*.[15] 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 regards to the provided abstraction. The main categories are the following [14] :

- **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 pre-setup 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 it's 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 [6, 9].

- **Scalability** - usually through a public API of some kind most cloud providers allow for the quick increase or reduction of resources [9]. 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 [8].

- **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 [6].

DevOps

Developers & Operations

First it is important to understand who were the interested parties and how they were related.

In software development there are two responsabilities that fall on two distinct departments: Operations and Developers.

Operations and their members are charged with the responsibilitie of operating the companies computing infrastructure. From changing faulty hardware to configuring new machines or monitoring incoming traffic, Operations have, as their goal, to keep the applications they were operating running and doing so correctly [1].

Developers, develop and build the application. The result of their work is then handed over to Operators who would then be resposible for running the application [7].

Problem

In *Agile infrastructure and operations: How infra-gile are you?* [4], Patrick Debois discusses and describes three different attempts to apply Agile methodologies to infrastructure and operations teams. In the three cases, companies had identified that the lack of articulation between the two departments was causing problems like failure to meet deadlines, outages or a lack of consistency across environments which lead, directly or indirectly to the loss of value for the respective companies.

As the article progresses, the cultural and technological changes that happened are described and, in the end a list of patterns is identified. Some of those patterns are as follow:

- " Operations know very well that changes introduce incidents. Therefore they think their job is to minimize change to the production environment. Still they serve the same customer the project does. Therefore the project should have a view on the demands the customer puts on the operations."

- " Development and infrastructure need to be seen as whole and not two separate projects. This is especially difficult in large enterprises where both belong to different entities."

- " Technical skills need to be complemented with an agile mindset. Similar to the technical, to speed things up have experienced agile project managers guide the process."

- " Non-agile infrastructure tends grow old changes are hard to execute in these environments. This can be seen as technical debt."

- " Every change will eventually require a management buy-in to allow it to persist."

Solution

In 2009, John Allspaw and Paul Hammond gave a talk at the *Velocity Conference* named *10+ Deploys Per Day: Dev and Ops Cooperation at Flickr* in San Jose where they discussed a new culture emerging at Flickr [2]. In this culture, John and Paul describe the way operations and developers objectives are changed from respectively keeping the site stable and fast and creating new functinalities to being enablers for the business. As the business evolve, changes are needed and both developers and operations should create the environment where those changes are possible without representing a risk for the business.

The talk then continues to describe the changes that happen at Flickr that enabled this change:

- **Automation** - By automating deployment processes (for staging or production environments), consistency across the environments improved decreasing the chance of errors in production.

---

[2] www.flickr.com

- **Deploying Frequently** - By enabling deploys to be made often(due to the automation of some of the deployment processes), this meant that less functionalities would be deployed each time. This made deployments more manageable and in turn safer.

- **Using Feature Flags** - Feature flags are special flags that toggle features on and off. Using this, improved error handling because if a feature was faulty, one could just turn it off rather than having to fix it right away. Feature flags would also enable more complex schemes of operation where certain functionalities would be launched but would not be displayed in the final site, once it was observer that the new functionality would not hurt performance or cause some kind of error, the flag could be turned on and the new funtionality would be available.

- **Version Control** - All code was centralized in a single repository. This meant that there was more transparency across instances.

- **Shared metrics** - Developers and operations would have access to metrics related with the site. Having this, developers could know how certain features would affect the infrastructure and this would in turn influence their decisions.

- **Communication increase & Cultural changes** - Developers and operations would communicate and a culture of respect and understanding between the two departments would be promoted. This change would allow trust to exist between the two departments meaning that developers and operations would take each other points of views, problems and objectives into consideration. Additional rules like *No finger pointing* would also promote a culture of collaboration and mutual aid between the two .

In the end, John and Paul both highlight the importance of the cultural aspects of the change. In their opinion if the cultural aspects are not met, the tooling or new practices will not be enough.

The DevOps Movement

In the problem section 2.2.2 we saw how the separation of departments was causing problems to both the businesses and professionals. This problem, often seen ans unsovable, would soon after be view as solvable after all. In 2.2.3 a new way of doing things started to emmerge. This new ideas would eventually fuel a community that would discuss them and enventually DevOps emmerged.

In the several studies that we read, we found several definitions for DevOps. Some or quite large and broad. In this thesis, we will use the one from *DevOps: A Software Architect's Perspective* [1] which is:

"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 find this definition to be empty in the sense that by only reading it one would not be aware of the aspects that are associated with Devops. On the other hand, it does allow us to easily

classify something as being DevOps or not i.e. we only have to ask ourselves if a practice or cultural aspect will allow for the reduction of time since commiting a change until that change is in normal production, if it does, then it is DevOps.

As we saw in 2.2.3, there are several ways in which DevOps can influence professionals and the work they do, but we have not seen what are the benefits for businesses. In *DevOps and the Cost of Downtime: Fortune 1000 Best Practice Metrics Quantified* [5], DevOps is viewed from a more business centric perspective and the following benefits are 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**[3] - Instead of creating and managing a system to store large files, or storing them in database columns, store them in a Cloud Storage[3]

- **Queue based solution to process asynchronous jobs**[3] - 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**[3] - 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**[3] - 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.

---

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

- **Email delivery**[3] - Rather than implementing your own SMTP solution, use cloud mail delivery services which provicde REST API's to send emails.

- **Logging**[3] - 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)**[3] - Monitor user behaviour in order to find possible bugs or errors.

- **The isolation by containerization pattern** [10] - "Use a container to package the applications and its dependencies and deploy the service within it".

- **The discovery by local reverse proxy pattern** [10] - "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** [10] - "Orchestrate services in a cluster based on each host's resource-offering announcements."

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 [2].

A study from 2015 [11] in which the Portuguese startup scene was analyzed, revealed that there were already 40 technology scaleups [4] 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 [5] and Startup Lisboa, both business incubators. This incubators had, at the time of this study more than 300 companies [13, 12] under their wing.

---

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

[5]Science and Technology Park of University of Porto

Chapter 3

Studying the Portuguese Startup Scene

Methodology

Both the companies and the individuals that have been involved in the adoption of DevOps are the best sources regarding the subject. Even though most of the knowledge they hold is not formalized, it represents a tested set of solutions to identified problems.

In order to collect this knowledge individuals and companies must be given a way to share their experiences and knowledge.

This collection of data will initially be made through online surveys. These will serve the purpose of selecting companies in which to focus.

When chosen, provided that they authorize it, companies will be asked to in a detailed way :

- Why did they felt that adopting DevOps was necessary.

- What initial concerns did they have when adopting DevOps.

- What were the obstacles that they have to overcome and how they they overcome them.

- What technologies do they use.

The gathered data from all companies will then be analyzed, compiled and formalized in the form of patterns

Solution Perspective

Collecting data from different companies will provide us with useful information on how to successfully adopt DevOps and what are the common pitfalls and traps, however, if not treated, this information can feel clumsy and disconnected. That is why, in addition to the collected data, common problems and solutions will be represented as patterns.

Granularity

In it's lowest level, granularity for the solutions presented could go as far as to specify what tools to use and how to configure and set them up. This would nonetheless be counterproductive as different companies have different requirements and different requirements require different tools. Consequently, the final solution will try to be tool agnostic and only present the situation and the types of solutions available with the hope to help the broader audience it can reach.

Why patterns?

In the software development world there are several examples of different domains that were studied and where patterns were identified. Architecture, software development and Scrum are just some examples of that.

Patterns are by definition recurring and, in that sense, identifying them means identifying situations that will frequently happen. This allows us to prevent common pitfalls and be prepared with solutions in case we can not prevent them.

When identified, patterns allow us to have a common language while referring to domains of knowledge. Instead of having to constantly describe a situation or solution one can simply refer the pattern and, provided that there is a common knowledge of the pattern, communication can be greatly simplified.

Additionally patterns are also useful when reasoning about a subject. They allow us to abstract away the complexity of a situation and better visualize the overall solution as a set of pattern instead of an agglomerate of situations.

Lastly, although they are tied to a context, patterns are within that context generic, meaning that they are not tied to a specific moment in time or space. This makes them even more useful as they can be reused.

Domains

As DevOps spreads throughout different areas of knowledge both the cultural and technological aspects need to be considered.

In the cultural side, information about aspects like how to redistribute operations teams or how to improve communication between peers will ,as well as other problems, be gathered and discussed. This will hopefully allow companies and their members to better accept the adoption of DevOps.

In the technological side, ways to improve the infrastructure elasticity among others will also be addressed.

Company selection and sorting

The script

Chapter 4

Patterns

Team Orchestration

Context

Budgets, customer needs, users numbers and other types constraints evolve at a fast pace. In order to be effective in this kind of environment, you want your teams to be able to adapt and respond quickly to those changes.

Problem

How do you ochestrate 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.

- 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 specialities represented.

- Members of multidisciplinary teams must be able to work together.

- It may not be possible to have all expertises constantly working in parallel in the same team.

Solution

Having multidisciplinary teams is usually and advantage. 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 specialities. Additionally, teams that have representatives of the expertises needed to deliver the final product are able to deliver it.

In terms of size, this 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 team have different knowledge and backgroud that differ from each other.

Professionals working in this kind of environment will sometimes disagree as they guide their work by different contraints 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 informatiom (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.

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 you 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.

15

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.

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.

Reproducible 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 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.

- **Containers** (containers)

- 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.

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*

- If you have chosen *Scripts*

- If you have chosen *Manual*

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.

Jobs

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.

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).

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.

Deployment Flow

Context

Problem

Forces

Solution

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.
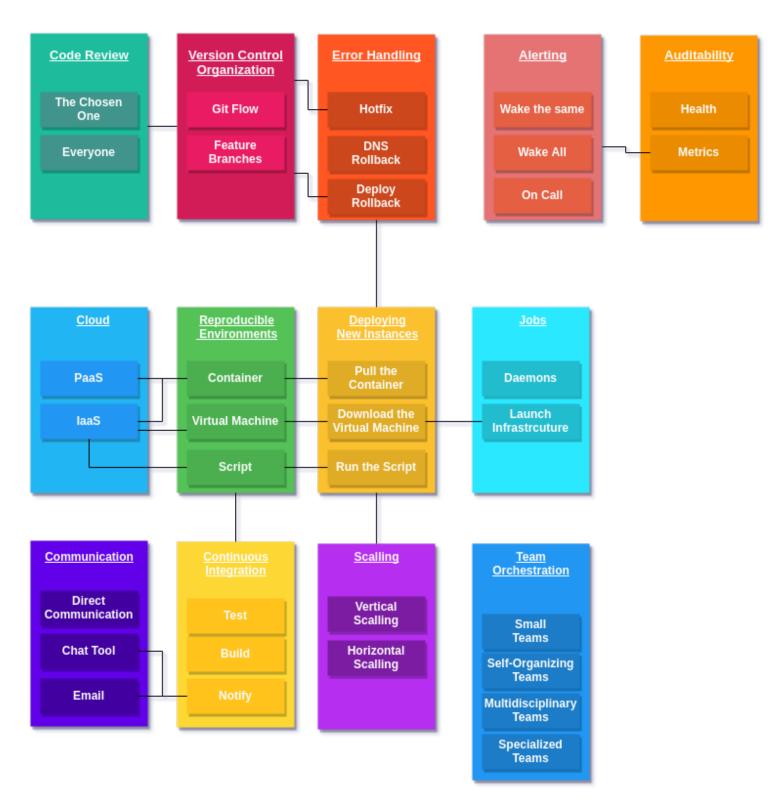
Figure 4.1: Pattern Map

Bibliography

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

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

[3] D. Cukier. DevOps patterns to scale web applications using cloud services. *Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity - SPLASH '13*, (Figure 2):143–152, 2013.

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

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

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

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

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

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

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

[11] Startup Europe Partnership. SEP MONITOR PORTUGAL RISING: MAPPING ICT SCALEUPS. 2015.

[12] Startup Lisboa. Startups.

[13] Uptec. Companies.

[14] 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.

[15] 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.