FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Towards DevOps: Practices and Patterns from the Portuguese Startup Scene

Carlos Manuel da Costa Martins Teixeira

DISSERTATION PLANNING

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. Hugo Sereno Ferreira

Co-Supervisor: Tiago Boldt Sousa

May 31, 2016

Towards DevOps:
Practices and Patterns from the Portuguese Startup Scene

Carlos Manuel da Costa Martins Teixeira

Mestrado Integrado em Engenharia Informática e Computação

May 31, 2016

Abstract

Software and its development have been increasing both in complexity and in size. As more businesses were moving first to the Internet and then to the Cloud, new technologies and opportunities started to emerge. As the environment were they acted changed, most practices and mindesets stayed the same causing a imbalance between the external and internal environment of companies. In the midst of this confusion new ideas started to appear with the objective of restoring the balance between the environment and those who operated eventually culminating in the rise of DevOps.

But what is DevOps? Formed by combining the word Development with the word Operations, the word "DevOps" has been around for sometime. Uses and appearances have been seen in different contexts and with different meanings making it a topic of discussion and discord. Being mostly refered to as a movement aiming to conciliate Software Operations and Software Developer professionals, there are those who see it as no more than a set of tools or a new job position. This mismatch of interpretations and overall lack of understanding of DevOps often means that companies and professionals lack the knowledge to fully take advantage of the benefits that DevOps brings.

In this thesis we demystify and formalize DevOps, first by using the existing literature as a basis to create a DevOps state-of-the-art and then by analysing the current practices associated with DevOps in the real world. The analysis, done by interviewing and observing some startups from Portugal is then the basis from wich we then extract a set of patterns related with DevOps and its values. Some of the 13 identified patterns are then validated by watching the effects of the DevOps culture in real people in real world situations.

In the end we conclude that even though the collected information allows us to have some knoledge and grasp of DevOps and its broad scope, due to that same broad scope more work and investigation is needed in order to fully understand the movement and its consequences.

Contents

Chapter 1

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.

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.

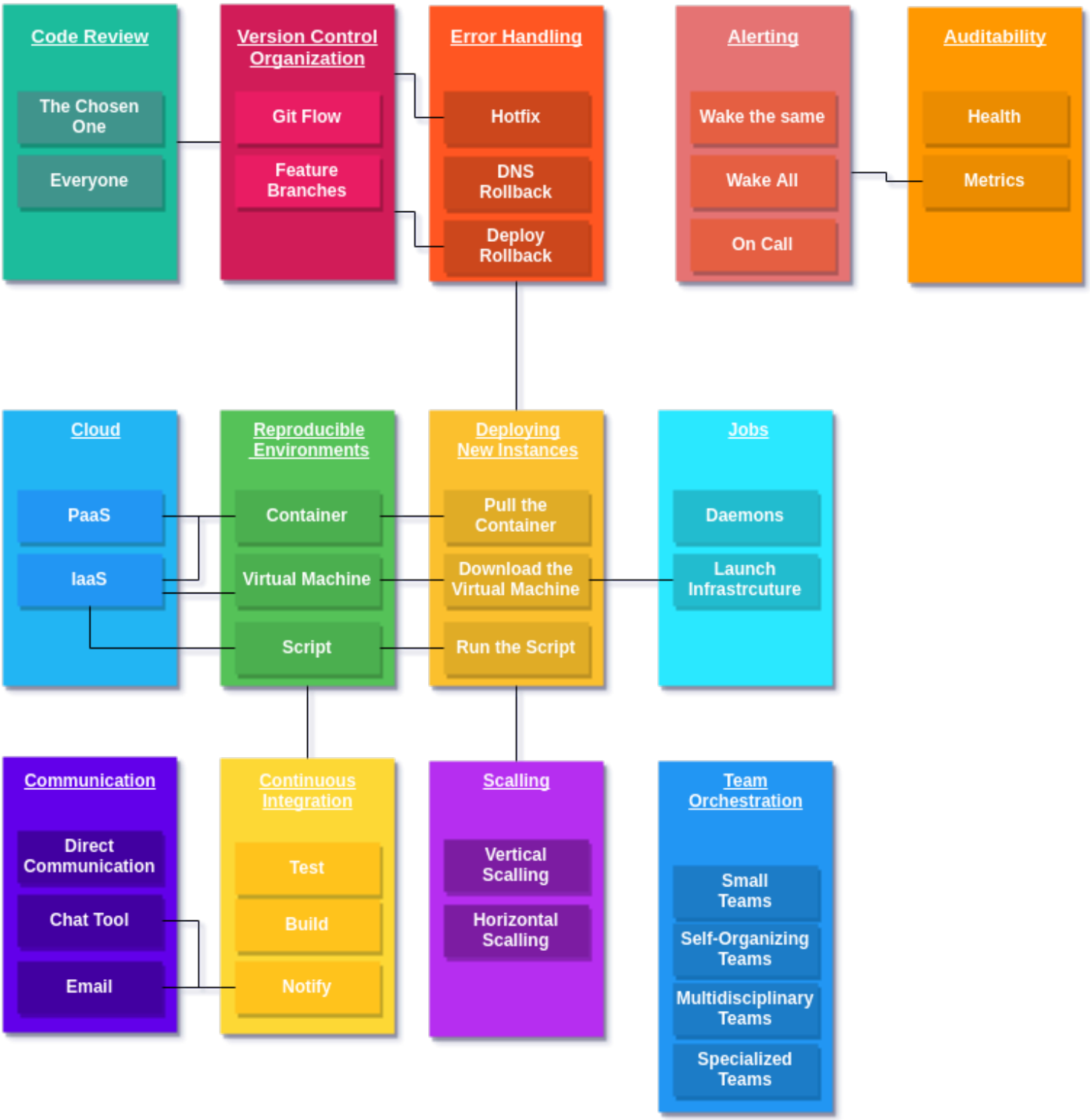

Figure 1.1: Pattern Map