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 26, 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 26, 2016

Contents

Chapter 1

Patterns

Team Orchestration

Context

When you have a project that you want to execute you will need to assign one or several teams and/or persons to that project. Because of you project complexity you have to allocate people with different expertises.

Problem

How do you ochestrate your team(s) in order to maximize you chances of success ?

Forces

- Teams should share objectives and be able to have their objectives aligned with the project objectives.
- Knowledge should spread across project intervinients.
- It may not be possible to have all expertises working in parallel in the same project.
- Teams should be able to colaborate.

Solution

A team should be looked as a unit that can, on their own, produce the results and functionalities necessary for the project. This means that teams should usually be multidisciplinary and capable of communicating efficiently.

Teams should also not have an imposed hierarchical structure and should be allowed to organize themselves.

In terms of size teams should not exceed the 9 elements, this small size allows teams to communicate better and to be able to self-organize.

Sometimes, due to lack of work some members of a team can accumulate more than one project at a time.

Version Control Organization

Context

You are using Git as a VCS tool and your team, having multiple people working on the same project and pushing code to the same repository, runs into situtations like:

- What is the code currently in production ?
- 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 to many branches may be complicated to manage.
- Having to few branches may make you loose valuable information.

Solution

There are several ways you can manage your VCS 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.

- **Gitlab Flow**
- **Github Flow**

Cloud

Context

In order to have computing power you have to **acquire computing resources**. This resources can be used to perform a number of functions:

- 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 there should not be a need to physically connect to 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 over operative system
- Sometimes due to a variation in your needs you may want to scale both up or down the ammount of resources you allocated.
- 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**.

  In 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

Because you have **several instances** of your software, you want to maintain *consistency* across all of them. Consistency is important because it will allow to have reproducibility and will give you some guarantees that you can setup new instances of your software.

Forces

- Having a complete copy of your environment (OS's, libraries, etc) may create large images.
- 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.

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.

Problem

How do you deploy your application ?

Forces

- You want to have guarantees that a deploy will be successfull.
- You want deployments to be reliable.
- You want to be able to deploy new instances as fast as possible.

Solution

In order have an efficient deploy both in terms of reliability and speed you should have some sort of reproducibility and depending on your choice for setting up your environment (*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

Having several people collaborating into the same project can be challenging. If a developer, unaware that introduced an error, pushes code it into the team repository a long time may go by before an error is even identified. After that the error needs to be identified and, because the code that introduced the error was pushed a long time ago, it may no seem obvious where the error is.

Problem

How do you detect early if some error has been introduced into your code?

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.

Communication