

# Desmistificando Microserviços e DevOps: Projetando Arquiteturas Efetivamente Escaláveis

Prof. Vinicius Cardoso Garcia  
vcg@cin.ufpe.br :: @vinicius3w :: assertlab.com

[IF1004] - Seminários em SI 3  
<https://github.com/vinicius3w/if1004-DevOps>

# Licença do material

Este Trabalho foi licenciado com uma Licença

Creative Commons - Atribuição-NãoComercial-CompartilhaIgual 3.0 Não Adaptada



Mais informações visite

<http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt>







# Crosscutting Concerns





# Other Ilities

“As a child my family’s menu consisted of two choices: take it or leave it.  
— Buddy Hackett”

# Introduction

- We already discussed the **major functionalities** of the **continuous deployment pipeline**, such as build, test, and deployment
- There are other DevOps operations that **resemble a process-like pipeline** such as error detection, diagnosis, and recovery



# Introduction

- “ility”: quality concerns other than those that focus on the basic functionalities and their correctness
  - How well are these functionalities in your pipeline performing?
  - Can you precisely repeat your DevOps operations when needed?
  - How much time has passed between a business concept and its final release?
  - How can different tools in your pipeline interoperate?



# List of the ilities and their primary quality concerns

| Ilities          | Quality Concerns   |
|------------------|--|
| Repeatability    | The degree to which repeating the same operation is possible   |
| Performance      | The time and resources required to execute a DevOps operation  |
| Reliability      | The degree to which the DevOps pipeline and individual pieces of software within it maintain their services for defined periods of time        |
| Recoverability   | The degree to which a failed DevOps operation can be brought back to a desired state with minimal impacts to the application being operated on |
| Interoperability | The degree to which different DevOps tools can usefully exchange information via interfaces in a particular context                            |
| Testability      | The ease with which the DevOps operation software can be made to demonstrate its faults through testing  |
| Modifiability    | The amount of effort required to change the DevOps software, processes, or the operation environment of an application                         |

We consider the ility issues of the DevOps pipeline from two different perspectives:  
product and process





# Repeatability

- Repeatability is the **degree** to which a process can be **repeated** for a **different application or branch**.
  - It can be **measured** by counting the **number of failures and successes** of that process.
  - If a process that was **previously successful now fails**, this failure is an indication that the process is **not repeatable in some different context**.
- Two activities are key to achieving repeatability: **definition and enforcement of processes** and **maintaining version control over all artifacts**.





# Repeatability

- Measuring repeatability depends on being able to identify that two executions of an operation are executing the same process.
- One means for doing this is to examine traces of the process to ensure they have performed the same steps in the same order.
  - In other words, we are equating repeatability and traceability



# Defining and Enforcing Process at the Appropriate Level

- There will be important **tradeoffs** between **defining** and **enforcing** repeatable actions to **improve** quality and allowing for leeway to enable desired creative activities.
- Process enforcement is a matter both of **automation** and of **social** processes.
  - **Automating** a process will enforce certain actions and certain gates.
  - **Social** processes such as wearing a hat that says “I Broke the Build” also **educate** and **encourage** team members to conform to particular processes.



# Defining and Enforcing Process at the Appropriate Level

- In order to achieve repeatability, processes must enforce selected practices.
- The choice of which practices to enforce is a portion of the tradeoffs that go into designing a DevOps process.
- The goal is to have some defined and repeatable best practices around the development workflow at an appropriate level.
- Defining an operation through scripts and automation tools does not inherently achieve repeatability





# Version Control Everything

- **Single tool** vs Multiple tools
  - Scripts change over time
    - Any script or code that changes the infrastructure or environment should be version controlled
- A script often takes some parameters to run, and these parameters operate on a particular environment and make changes to that particular environment



# Version Control Everything

- **Deployment tools.** Maintaining traceability of the higher-level process could be accomplished by a deployment tool since it is the last stage of placing an instance in production.
- **Configuration management database (CMDB).** The configuration parameters are stored in a database. This database can also record accesses, so that tools and scripts that access configuration information will be known later.
- **Tagging data items.** Each script manipulates some entity. Tagging the entity will lead to the final deployed version having traceability information.



# Performance

- Performance is characterized by the **amount** of **useful work** it accomplishes and the **time and resources used** to **accomplish** that work
- Can be measured by the response time to a given piece of work





# Measuring the Important Things

- Before you can improve the performance of your pipeline, you should first measure it!
- At a high level, the performance of interest is the time between a business concept and its successful deployment.



# Measuring the Important Things

- Measure the **different types of errors** that occur and the **reasons** behind them
- A recent empirical study from **Google** shows that the top build errors are related to **dependency issues** representing **52.68% (C++)** or **64.71% (Java)** of all build errors.
- **Understanding** these errors can guide your efforts for **improvement**.



# Measuring the Important Things

- For various reasons it is **not always best** to enforce best practices **proactively and mechanically**. Instead, **monitoring compliance** provides an **indication** of problems.
- Not all deviations are **problems**, some may be **justified**.
- On the other hand, **multiple compliance deviations** can indicate a **need to improve** a process.





# Improving Resource Utilization

- Moving all the above environments to the cloud, thereby switching off machines that are currently underutilized and only paying for what you use.
- Using containers rather than virtual machines (VMs).



# Reliability

- Software **fails**.
- **Reliability** refers to the **capability** of the overall DevOps pipeline and its individual pieces to **maintain service**.
- The DevOps pipeline can be seen as a **distributed system of systems** dealing with various **distributed services**.



# Improving the reliability of the pipeline

- Understanding the Reliability Characteristics of Different Services
  - Use a wrapper around the original service to improve reliability, employing standard fault-tolerant mechanisms.
  - Use local mirrors of the remote services.
- Detecting and Repairing Errors Early





# Recoverability

- The goal of recoverability is to **enable easy recovery** after a **failure**, whether the cause is an internal or external system, or human operators.
  - Include extensive exception handlings in your operation logic.
  - Build in support for external monitoring or recovery systems.
  - Design your software with operators as first-class stakeholders.



# Interoperability

- Interoperability refers to the **degree** to which **different tools** can **usefully exchange information** via **interfaces** in a particular context.
- Paying Attention to Interoperation of Interfaces
- Understanding Existing Data Models



# Testability

- Testability concerns the **effort required** for the software to **demonstrate** its **faults** through **testing**.
- However, a DevOps pipeline poses additional challenges...
- the difficulty of testing infrastructure outcomes [infrastructure-as-code]





# Modifiability

- Modifiability is a measure of the **ability** to **make changes** in **existing software**.
- Change either the interactions with one of the stages in the pipeline or the conditions for moving from one stage to the next.



# Summary

- The take-aways are:
  - quality comes from the beginning and not from the end
  - in as an architecture

| Qualities        | Techniques to Achieve this Quality  |
|------------------|---|
| Repeatability    | Maintain traces of activities. Version control everything. Use a CMDB to maintain parameters. Enforce where necessary.  |
| Performance      | Measure to determine bottlenecks in processes. Tear down an environment when it is not used. Perform as many operations as possible in the cloud, where resources can be freed if not used.                             |
| Reliability      | Identify failure rates of different services. Mirror services with high failure rates. Detect failures as soon as possible through tools whose job it is to monitor components for execution times outside of the norm. |
| Recoverability   | Build in exception handling in scripts. Provide information for monitoring services. Ensure that appropriate diagnostics are generated to enable faster debugging.  |
| Interoperability | Select tools with stable interfaces and flexible scripting facilities. Ensure data models of various phases of the pipeline are consistent.   |
| Testability      | Use unit and integration test scripts for specialized tools. Coordinate test cases with monitoring rules.   |
| Modifiability    | Modularize scripts based on expected changes to the tools. Encapsulate operations actions into small modules that are loosely coupled with each other.  |

ilities and  
sidered  
dding them



# For Further Reading

- Forilities in software architecture: L. Bass, P. Clements, and R. Kazman. [Software Architecture in Practice](#), 3rd Edition. Addison-Wesley, 2013.
- For common build errors, you can read the Google study published in "[Programmers' Build Errors: A Case Study \(at Google\)](#)"
- “For more about testability and test-driven development for infrastructure code, "[Test-Driven Infrastructure with Chef](#)”





# Homework #6

- Students should read the article "[Programmers' build errors: a case study \(at google\)](#)", and post their understanding of the work in a MD file. Submit in our Slack team, the MD file (LOGIN-HW6.md) containing the following:
  - Your identification (name and CIn email address)
  - Your understanding of the work, a brief resume, strong and positive points, weak points, etc.
- Due D+5 (Sunday, 9/04), 17:00.

