Desmistificando Microsserviços e DevOps: Projetando Arquiteturas Efetivamente Escaláveis

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

[IF1004] - Seminários em SI 3
http://bit.ly/vcg-devops

ASSERT Lab
Advanced System and Software Engineering Research Technologies

Licença do material

Este Trabalho foi licenciado com uma Licença

Creative Commons - Atribuição-NãoComercial-Compartilhalgual 3.0 Não Adaptada

Mais informações visite

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

The Deployment Pipeline

Building and Testing

> Testing leads to failure, and failure leads to understanding.
> — Burt Rutan

Introduction

- Although architects like to focus on design and implementation, the infrastructure that is used to support the development and deployment process is important for a number of reasons

Introduction

- Team members can work on different versions of the system concurrently.
- Code developed by one team member does not overwrite the code developed by another team member by accident.
- Work is not lost if a team member suddenly leaves the team.
- Team members' code can be easily tested.
- Team members' code can be easily integrated with the code produced by other members of the same team.
- The code produced by one team can be easily integrated with code produced by other teams.
- An integrated version of the system can be easily deployed into various environments (e.g., testing, staging, and production).
- An integrated version of the system can be easily and fully "tested without affecting the production version of the system.
- A recently deployed new version of the system can be closely supervised.
- Older versions of the code are available in case a problem develops once the code has been placed into production.
- Code can be rolled back in the case of a problem.

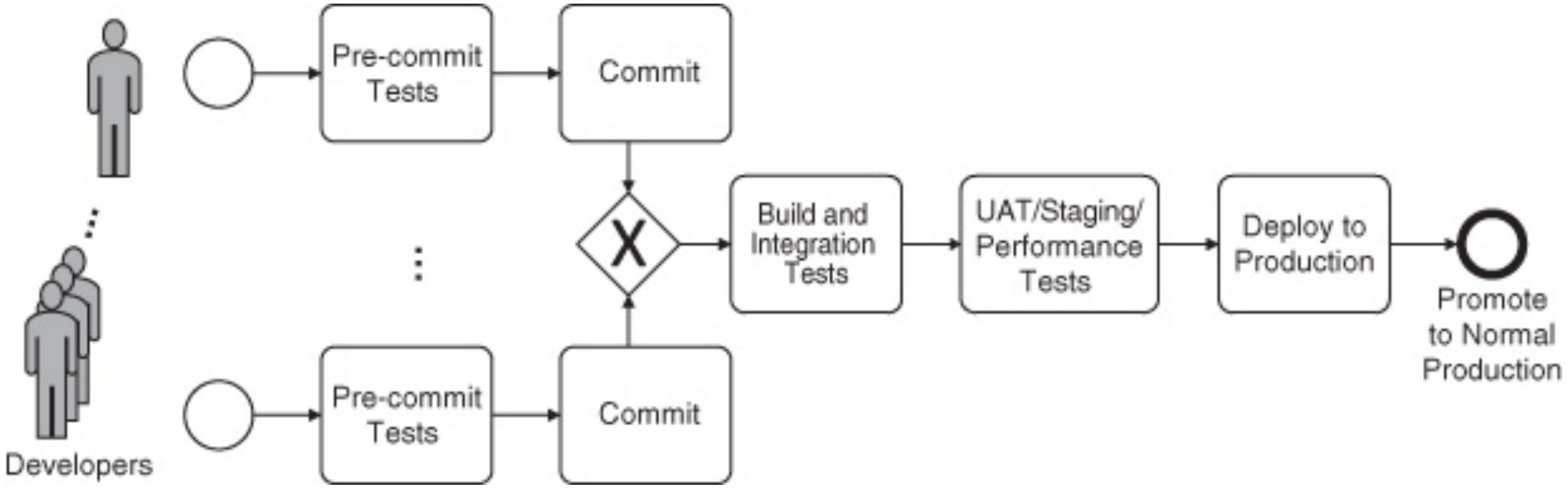
Question

- What is the most important reason why practicing architects should probably be concerned about the development and deployment infrastructure?
- Either they or the project managers are responsible for ensuring that the development infrastructure can meet the preceding requirements

Deployment pipeline

- None of the requirements are new, although the tools used to support these tasks have evolved and gained sophistication over the years

Deployment pipeline



Continuous Integration, Delivery & Deployment

- One way to define continuous integration is to have automatic triggers between one phase and the next, up to integration tests
  - That is, if the build is successful then integration tests are triggered. If not, the developer responsible for the failure is notified
- Continuous delivery is defined as having automated triggers as far as the staging system
  - This is the box labeled UAT (user acceptance test)/staging/performance tests
- Continuous deployment means that the next to last step (i.e., deployment into the production system) is automated as well
  - Once a service is deployed into production it is closely monitored for a period and then it is promoted into normal production
  - At this final stage, monitoring and testing still exist but the service is no different from other services in this regard

Then…

- We use the deployment pipeline as an organizing theme for this lectures
- Then we discuss crosscutting concerns of the different steps, followed by sections on the pre-commit stage, build and integration testing, UAT/staging/performance tests, production, and post-production
- Before moving to that discussion, however, we discuss the movement of a system through the pipeline

Moving a System Through the Deployment Pipeline

Moving a System Through the Deployment Pipeline

- Committed code moves through the steps shown in Figure, but the code does not move of its own volition
- Rather, it is moved by tools controlled by their programs (scripts) or by developer/operator commands. Two aspects of this movement are of interest:
  1. Traceability
  2. The environment associated with each step of the pipeline

Traceability

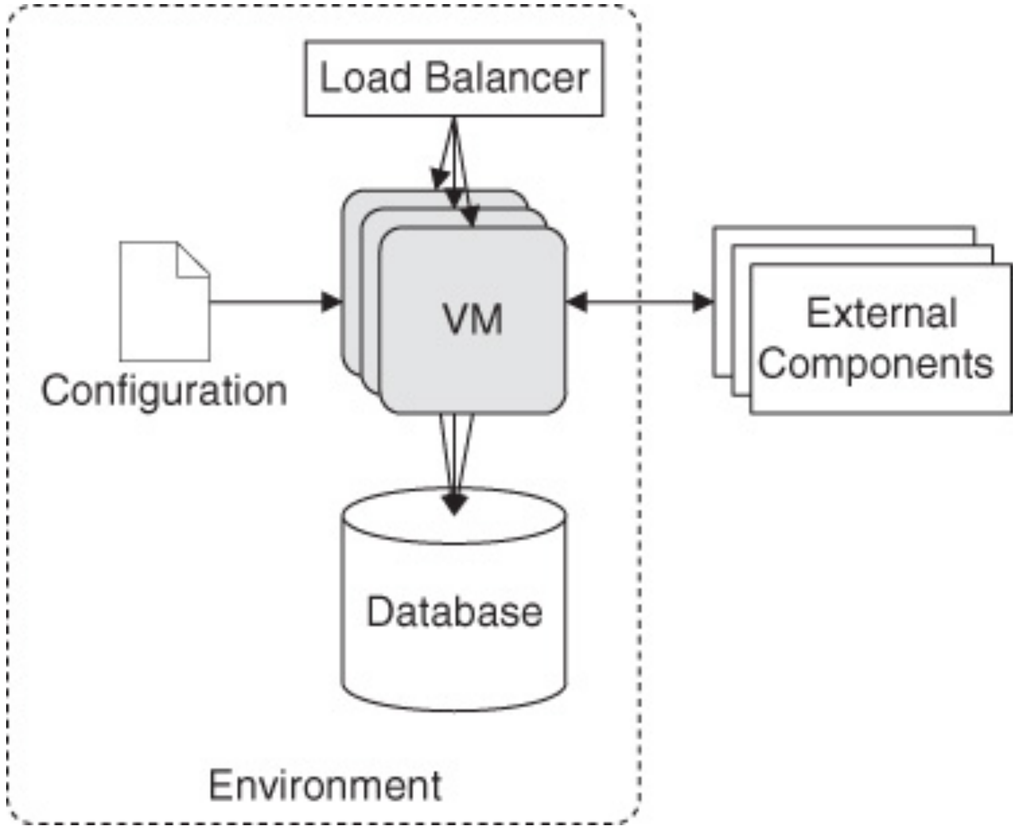
- Traceability means that, for any system in production, it is possible to determine exactly how it came to be in production
  - This means keeping track not only of source code but also of all the commands to all the tools that acted on the elements of the system
- Individual commands are difficult to trace. For this reason, controlling tools by scripts is far better than controlling tools by commands
  - Infrastructure as Code

Traceability

- A complication to the requirement to keep everything in version control is the treatment of third-party software such as Java libraries
- Such libraries can be bulky and can consume a lot of storage space
- Libraries also change, so you must find a mechanism to ensure you include the correct version of third-party software in a build, without having multiple copies of the same version of the library on the servers running your system
- Software project management tools like Apache Maven can go a long way to managing the complexities of library usage.

The Environment

- An executing system can be viewed as a collection of executing code, an environment, configuration, systems outside of the environment with which the primary system interacts, and data
- As the system moves through the deployment pipeline, these items work together to generate the desired behavior or information



Pre-commit

- The code is the module of the system on which the developer is working
- In Lecture 2, we discussed reducing coordination among teams. Pre-commit requires coordination within a team
- The environment is typically a laptop or a desktop, the external systems are stubbed out or mocked, and only limited data is used for testing
- Read-only external systems, for example, an RSS feed, can be accessed during the pre-commit stage.
- Configuration parameters should reflect the environment and also control the debugging level

Build and integration testing

- The environment is usually a continuous integration server
- The code is compiled, and the component is built and baked into a VM image
  - This VM image does not change in subsequent steps of the pipeline
- During integration testing, a set of test data forms a test database
- The configuration parameters connect the built system with an integration testing environment

UAT/staging/performance testing

- The environment is as close to production as possible
- Automated acceptance tests are run, and stress testing is performed through the use of artificially generated workloads
- The database should have some subset of actual production data in it
- Configuration parameters connect the tested system with the larger test environment
- Access to the production database should not be allowed from the staging environment

Production

- The production environment should access the live database and have sufficient resources to adequately handle its workload
- Configuration parameters connect the system with the production environment

A longer list of environments form Wikipedia

- Local: Developer's laptop/desktop/workstation
- Development: Development server, a.k.a. sandbox
- Integration: Continuous integration (CI) build target, or for developer testing of side effects
- Test/QA: For functional, performance testing, quality assurance, etc.
- UAT: User acceptance testing
- Stage/Pre-production: Mirror of production environment
- Production/Live: Serves end-users/clients

Crosscutting Aspects of Testing

Test harnesses

- A test harness is a collection of software and test data configured to test a program unit by running it under varying conditions and monitoring its behaviour and output
- Test harnesses are essential in order to automate tests
- A critical feature of a test harness is that it generates a report
- In particular it should, at a minimum, identify which tests failed

Negative tests

- Most tests follow the “happy path” and check if the system behaves as expected when all assumptions about the environment hold and the user performs actions in the right order with the right inputs
- It is also important to test if the system behaves in a defined way when these assumptions are not met
  - Tests that follow this purpose are collectively called negative tests.
- The common expectation is that the application should degrade or fail gracefully (i.e., only degrade the functionality as necessitated by the actual problem), and, if failure is unavoidable, provide meaningful error messages and exit in a controlled manner.

Regression testing

- Is the core reason for maintaining and rerunning tests after they first passed
- Another use of regression testing is to ensure that any fixed bugs are not reintroduced later on
- When fixing a bug in test-driven development, it is good practice to amend the test suite by adding a test that reproduces the bug
  - In the current version, this test should fail
  - After the bug has been fixed, the test should pass
- It is possible to automate the regression test creation: Failures detected at later points in the deployment pipeline (e.g., during staging testing) can be automatically recorded and added as new tests into unit or integration testing

Traceability of errors

- If a bug occurs in production, you want to be able to find out quickly which version of the source code is running, so that you can inspect and reproduce the bug
  - The first option to enable traceability is to associate identifying tags to the packaged application, such as the commit ID of various pieces of software and scripts that specify the provenance
  - Another option is to have an external configuration management system that contains the provenance of each machine in production

Small components

- We mentioned in Lecture 2 that small teams mean small components
- In Lecture 5, we discussed microservices as a manifestation of small components
  - It is also the case that small components are easier to test individually
  - A small component has fewer paths through it and likely has fewer interfaces and parameters
- These consequences of smallness mean that small components are easier to test, with fewer test cases necessary
- However, as mentioned in Lecture 2, the smallness also introduces additional challenges in integration and requires end-to-end tests due to the involvement of more components

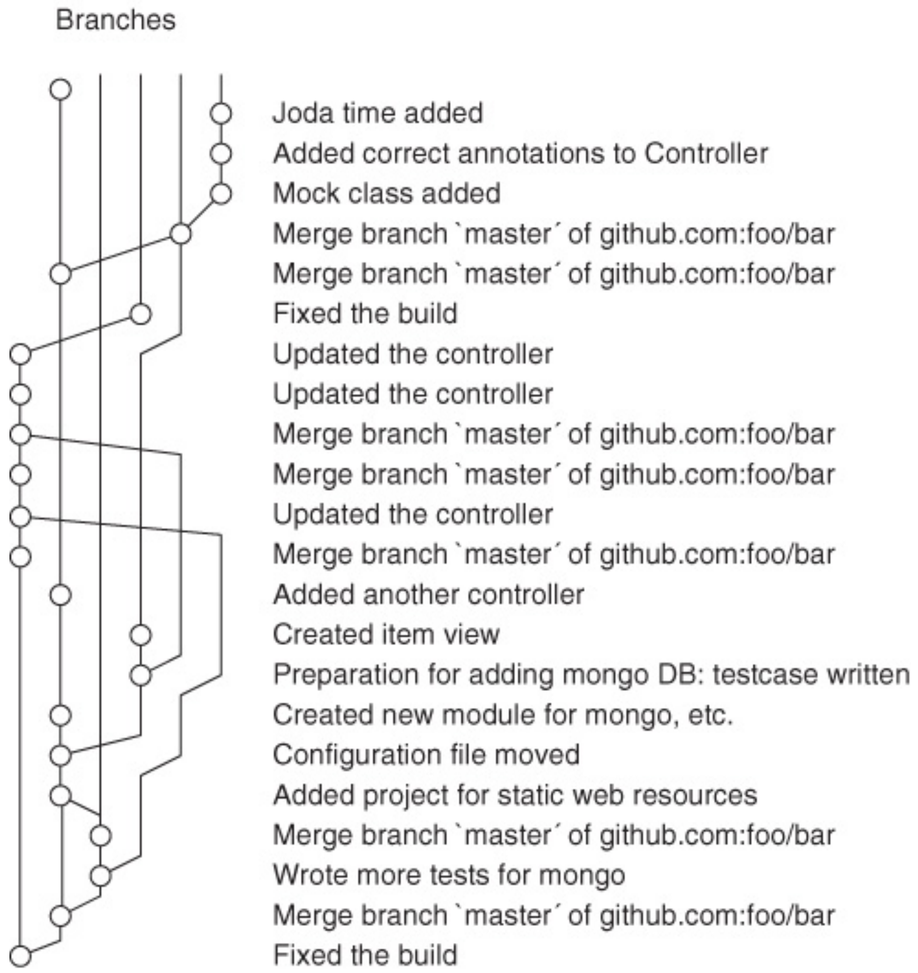
Environment tear down

- Once an environment is no longer being used for a specific purpose, such as staging, it should be dismantled
  - Freeing resources associated with the environment is one rationale for tearing down an environment; avoiding unintended interactions with resources is another
- The case study in Lecture 13 makes tear down an explicit portion of the process
  - It is easy to lose track of resources after their purpose has been achieved.
- Every VM must be patched for security purposes, and unused and untracked resources provide a possible attack surface from a malicious user
- We discuss an example of an exploitation of an unused VM in Lecture 9

Development and Pre-commit Testing

Version Control and Branching

- You may have too many branches and lose track of which branch you should be working on for a particular task.
  - Short-lived tasks should not create a new branch
- Merging two branches can be difficult
  - Different branches evolve concurrently, and often developers touch many different parts of the code.



Feature Toggles

- A feature toggle (also called a feature flag or a feature switch) is an “if” statement around immature code
- A new feature that is not ready for testing or production is disabled in the source code itself, for example, by setting a global Boolean variable
  - Common practice places the switches for features into configuration
- The switch is toggled in production (i.e., the feature is turned on) only once the feature is ready to be released and has successfully passed all necessary tests
- When there are many feature toggles, managing them becomes complicated

Configuration Parameters

- A configuration parameter is an externally settable variable that changes the behavior of a system
- The number of configuration parameters should be kept at a manageable level
- One decision to make about configuration parameters is whether the values should be the same in the different steps of the deployment pipeline
  - Values are the same in multiple environments
  - Values are different depending on the environment
  - Values must be kept confidential

Testing During Development and Pre-commit Tests

- Test-driven development
  - A virtue of this practice is that happy or sunny day path tests are created for all of the code
- Unit tests
  - A common practice is to write the code in a way that complicated but required artifacts (such as database connections) form an input to a class—unit tests can provide mock versions of these artifacts, which require less overhead and run faster

Build and Integration Testing

Build scripts

- The build and integration tests are performed by a continuous integration (CI) server
- The input to this server should be scripts that can be invoked by a single command (“build")
- This practice ensures that the build is repeatable and traceable

Packaging

- Runtime-specific packages
- Operating system packages
- VM images
- Lightweight containers
- There are two dominant strategies for applying changes in an application when using VM images or lightweight containers:
  - Heavily baked images cannot be changed at runtime: immutable servers
  - Lightly baked images are fairly similar to heavily baked images, with the exception that certain changes to the instances are allowed at runtime

Continuous Integration and Build Status

- Once building is set up as a script callable as a single command, continuous integration can be done as follows:
  - The CI server gets notified of new commits or checks periodically for them
  - When a new commit is detected, the CI server retrieves it
  - The CI server runs the build scripts
  - If the build is successful, the CI server runs the automated tests —as described previously and in the next section
  - The CI server provides results from its activities to the development team (e.g., via an internal web page or e-mail)

Integration Testing

- Integration testing is the step in which the built executable artifact is tested
- The environment includes connections to external services, such as a surrogate database
- Including other services requires mechanisms to distinguish between production and test requests
  - This distinction can be achieved by providing mock services
- As with all of the tests we discussed, integration tests are executed by a test harness, and the results of the tests are recorded and reported

UAT/Staging/ Performance Testing

UAT/Staging/Performance Testing

- Staging is the last step of the deployment pipeline prior to deploying the system into production
  - User acceptance tests (UATs) are tests where prospective users work with a current revision of the system through its UI and test it, either according to a test script or in an exploratory fashion
  - Automated acceptance tests are the automated version of repetitive UATs
  - Smoke tests are a subset of the automated acceptance tests that are used to quickly analyze if a new commit breaks some of the core functions of the application
  - Nonfunctional tests test aspects such as performance, security, capacity, and availability

Production

Early Release Testing

- The most traditional approach is a beta release
- Canary testing is a method of deploying the new version to a few servers first, to see how they perform
- A/B testing is similar to canary testing, except that the tests are intended to determine which version performs better in terms of certain business-level key performance indicators

Error Detection

- Even systems that have passed all of their tests may still have errors
  - Techniques used to determine nonfunctional errors include monitoring of the system for indications of poor behavior
  - This can consist of monitoring the timing of the response to user requests, the queue lengths, and so forth
- Enabling the diagnosis of errors is one of the reasons for the emphasis on using automated tools that maintain histories of their activities
- Once the error is diagnosed and repaired, the cause of the error can be made one of the regression tests for future releases

Live Testing

- Another form of testing after the system has been placed in production is to actually perturb the running system
- Netflix has a set of test tools called the Simian Arm
  - For example, the Chaos Monkey kills active VMs at random
  - The Latency Monkey injects delays into messages

Incidents

Incidents

- No matter how well you test or organize a deployment, errors will exist once a system gets into production
- Understanding potential causes of post-deployment errors helps to more quickly diagnose problems

Summary

- An architect involved in a DevOps project should ensure the following:
  - The various tools and environments are set up to enable their activities to be traceable and repeatable
  - Configuration parameters should be organized based on whether they will change for different environments and on their confidentiality
  - Each step in the deployment pipeline has a collection of automated tests with an appropriate test harness
  - Feature toggles are removed when the code they toggle has been placed into production and been judged to be successfully deployed

For Further Reading

- For a more detailed discussion of many of the issues covered in this lecture, see the book: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation
- The Simian Army is defined and discussed in: J. Kojo, V. Asokan, G. Campbell, and A. Tull. “Nicobar: Dynamic Scripting Library for Java,” February 10, 2015, http://techblog.netflix.com

Homework #4

- Reading the post of Carl Caum, discuss the main issues, differences and benefits between continuous delivery and continuous deployment
- Reading the article of Paul Hammant, discussing branch versus trunk-based approaches, relate in your opinion the main advantages and disadvantages of each approach.
- Reading this article, explain, in your words, how are the factors that I should consider to decide between heavily baked and lightly baked images.