

**MINISTÉRIO DA DEFESA  
EXÉRCITO BRASILEIRO  
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA  
INSTITUTO MILITAR DE ENGENHARIA  
ENGENHARIA DE COMPUTAÇÃO - SE/8**

Luan Ferreira Cardoso, Ricardo Sollon Zalla, Venicius Gonçalves da  
Rocha Junior

**DevOps: aproximando a área de desenvolvimento da  
operacional**

Rio de Janeiro  
11 de maio de 2016

Luan Ferreira Cardoso, Ricardo Sollon Zalla, Venicius Gonçalves da Rocha  
Junior

## **DevOps: aproximando a área de desenvolvimento da operacional**

Trabalho apresentado ao Curso de Engenharia de Computação do Instituto Militar de Engenharia como Verificação Especial do Projeto de Fim de Curso.

Instituto Militar de Engenharia

Orientador: Clayton Escouper das Chagas

Rio de Janeiro

11 de maio de 2016

c2016

Instituto Militar de Engenharia  
Praça General Tibúrcio, 80 - Praia Vermelha  
Rio de Janeiro - RJ CEP: 22290-270

Este exemplar é de propriedade do Instituto Militar de Engenharia, que poderá incluí-lo em base de dados, armazenar em computador, microfilmар ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade dos autores e do orientador.

Cardoso, Luan; Zalla, Ricardo e Gonçalves, Venicius  
S586d DevOps: aproximando a área de desenvolvimento da operacional / Luan  
Ferreira Cardoso, Ricardo Sollon Zalla, Venicius Gonçalves da Rocha Junior.  
- Rio de Janeiro: Instituto Militar de Engenharia, 2016.

43f. : il., graf., tab. : -cm.

Projeto de Fim de Curso - Instituto Militar de Engenharia  
Orientador: Clayton Escouper das Chagas.

1 - DevOps 2 - Desenvolvimento e Operação

CDU ????.???.

Luan Ferreira Cardoso, Ricardo Sollon Zalla, Venicius Gonçalves da Rocha  
Junior

## **DevOps: aproximando a área de desenvolvimento da operacional**

Trabalho apresentado ao Curso de Engenharia de Computação do Instituto Militar de Engenharia como Verificação Especial do Projeto de Fim de Curso.

Trabalho aprovado. Rio de Janeiro, 11 de maio de 2016:

---

**Prof. Clayton Escouper das Chagas**  
Orientador, M. Sc.

---

**Prof. Ricardo Choren Noya**  
Convidado, D. c.

---

**Prof. Humberto Henriques de Arruda**  
Convidado, M. c.

Rio de Janeiro

11 de maio de 2016

# Sumário

	<b>Lista de ilustrações</b>	<b>6</b>
	<b>Lista de tabelas</b>	<b>7</b>
	<b>Lista de abreviaturas e siglas</b>	<b>8</b>
	<b>Lista de símbolos</b>	<b>9</b>
<b>1</b>	<b>INTRODUÇÃO</b>	<b>12</b>
1.1	Motivação	12
1.2	Objetivo	12
1.3	Justificativa	13
<b>2</b>	<b>METODOLOGIA</b>	<b>14</b>
<b>3</b>	<b>FERRAMENTAS</b>	<b>15</b>
3.1	Banco de Dados	15
3.1.1	MySQL	15
3.2	Gerenciamento de configurações de software - SCM	15
3.2.1	Git	15
3.2.2	GitHub	15
3.3	Build	15
3.3.1	Maven	15
3.4	Entrega Contínua - CI	16
3.4.1	Jenkins	16
3.5	Deployment	16
3.5.1	SSH	16
3.6	Provisionamento	16
3.6.1	Puppet	16
3.6.2	Ansible	16
3.6.3	Vagrant	16
3.7	Monitoramento	17
3.7.1	Nagios	17
3.8	Serviços Cloud	17
3.8.1	Amazon Web Services	17
3.9	Containers	17
3.9.1	Docker	17

<b>4</b>	<b>ARQUITETURAS</b>	<b>19</b>
<b>4.1</b>	<b>Primeira Arquitetura Customizada</b>	<b>19</b>
4.1.1	Aplicação	19
4.1.2	Ambiente de Produção	21
4.1.3	Build e Deploy da Aplicação	21
<b>4.2</b>	<b>Segunda Arquitetura Customizada</b>	<b>23</b>
4.2.1	Criação de chave pública SSH	23
4.2.2	Configuração do controle de versão com Git e GitHub	24
4.2.3	Vagrant e Virtualbox	27
4.2.4	Ansible	31
4.2.5	Instalando Wordpress em uma máquina	34
<b>4.3</b>	<b>Arquitetura proprietária</b>	<b>34</b>
4.3.1	Introdução ao Bluemix	34
4.3.2	Devops e Bluemix	35
4.3.3	Serviços de DevOps disponíveis	36
4.3.4	Serviços	37
4.3.5	Características	38
4.3.6	Requisitos Mínimos	39
4.3.7	Bluemix na prática	40
4.3.8	Prova de conceito	40
<b>5</b>	<b>CRONOGRAMA</b>	<b>41</b>
<b>5.1</b>	<b>Luan Ferreira Cardoso</b>	<b>41</b>
<b>6</b>	<b>CONCLUSÃO PARCIAL</b>	<b>42</b>
	<b>Referências</b>	<b>43</b>

# Lista de ilustrações

Figura 1 – Uma prévia da da página inicial da loja virtual . . . . .	20
Figura 2 – Ambiente de produção da loja virtual . . . . .	21
Figura 3 – Configuração declarada no VagrantFile . . . . .	22
Figura 4 – Estrutura do Bluemix . . . . .	35
Figura 5 – <i>Roadmap</i> da experiência de DevOps no Bluemix . . . . .	37
Figura 6 – Tela para adicionar ou ligar serviço no Bluemix . . . . .	38
Figura 7 – <i>Pipeline</i> no Bluemix . . . . .	39

## Lista de tabelas



# Lista de abreviaturas e siglas

Fig.	Figura
------	--------

# Lista de símbolos

Γ	Letra grega Gamas
---	-------------------

# Resumo

Este trabalho está focado na análise e comparação de três modelos de estruturas DevOps encontrados na atualidade. Em uma primeira abordagem, o sistema da IBM Blue Mix será estudado, analisado, e terá suas características catalogadas para a futura comparação. Em uma segunda abordagem, será criado e testado o modelo encontrado no livro Caixa de Ferramentas DevOps: um guia para a construção, administração e arquitetura de sistemas modernos. Esse modelo usa para criar sua estrutura DevOps as seguintes ferramentas: Linux, SSH, Git, Vagrant, Ansible, exemplo da instalação de wordpress em uma máquina virtual, proxy reverso, Cassandra e EC2, Métricas e monitoração, Análise de performance em cloud com New Relic, Docker e uma última parte que fala de técnicas para se colocar software em produção. Nessa fase, os resultados também serão catalogados para as comparações e análises. Em uma terceira fase, será abordado o modelo encontrado no livro DevOps na pratica: entrega de software confiável e automatizada. Esse livro aborda os seguintes temas para a construção do seu ambiente DevOps: introdução às idéias pregadas pelas técnicas DevOps, uma parte falando sobre a fase de produção, uma parte falando sobre monitoramento, uma parte falando sobre infraestrutura como código, uma parte falando sobre Puppet, uma parte falando sobre integração contínua, Pipeline de entrega e uma parte que fala de tópicos avançados. Após essas implementações e catalogações, será feita uma comparação dos três modelos analisando suas características e elencando pontos positivos e negativos.

**Palavras-chave:** DevOps, desenvolvimento, operação, ambientes.

# Abstract

This work is focused on analysis and comparison of three DevOps models of structures found today. In a first approach, the IBM Blue Mix system will be studied, analyzed, and will have its characteristics cataloged for future comparison. In a second approach, the model found in the book *Caixa de Ferramentas DevOps - um guia para a construção, administração e arquitetura de sistemas modernos* will be created and tested. This model uses to create its DevOps structure the following tools: Linux, SSH, Git, Vagrant, Ansible, wordpress installation example a virtual machine, reverse proxy, Cassandra and EC2, and Metrics monitoring, cloud-performance analysis with New Relic, Docker and a last part that talks about techniques to put software in production. In this phase, results will also be cataloged for comparisons and analysis. In a third phase, the model found in the book *DevOps na pratica - entrega de software confiável e automatizada* will be created and tested. This book covers the following topics for the construction of its DevOps environment: introduction to the ideas preached by technical DevOps, a part talking about the production phase, a part talking about monitoring, a part talking about infrastructure as code, a part talking about Puppet, a part talking about continuous integration, delivery Pipeline and a part that speaks of advanced topics. After these implementations and catalogations, the three models will be compared and analysed and the positives and negatives points will be pointed.

**Keywords:** DevOps, development, operation, environment.

# 1 Introdução

## 1.1 Motivação

Quando uma organização precisa de servidores e computadores, ou precisa desenvolver um software e liberá-lo para os usuários, ou ainda precisa de mais colaboração e comunicação entre as equipes devido a peculiaridades de alguns projetos, surge a necessidade de instalação e configuração de sistemas operacionais, programas e serviços que entrarão em operação ao final do projeto. Essa situação, aparentemente simples do ponto de vista de um usuário comum que instala os programas convencionais de que precisa, se transforma em uma tarefa de configuração complexa e inviável de ser feita para organizações com um número de servidores e computadores muito elevado <sup>[1]</sup>. Essa demanda por ativos computacionais pode variar muito dependendo do serviço oferecido pela organização, pode crescer dia a dia ou apresentar picos sob uma demanda específica, e para se otimizar a relação entre custo benefício, e para possibilitar uma entrega contínua e confiável <sup>[2]</sup>, se faz necessária a capacidade de automatizar o processo de desenvolvimento e implantação de tais sistemas computacionais quando for necessário.

Assim, o processo de instalação dos sistemas operacionais e dos aplicativos se torna árduo e envolve tarefas trabalhosas e repetitivas para os administradores e desenvolvedores <sup>[3]</sup>. Nesse cenário, surgiu uma tendência de tentar criar estruturas automatizadas que pudessem facilitar a integração desses processos de desenvolvimento de sistemas <sup>[1]</sup>, englobando todas as fases do processo de desenvolvimento de softwares e sistemas.

A partir desse momento, os administradores não mais ficaram responsáveis por configurar e instalar sistemas de softwares, e passaram a investir seu tempo no desenvolvimento de ferramentas que automatizem todos os passos do processo. Nesse contexto, uma área chamada DevOps <sup>[4]</sup>, que trata da integração de operação com desenvolvimento de sistemas vem se apresentando e se fortalecendo, a medida em que as demandas por estruturas de sistemas cada vez mais flexíveis e com menor custo vem crescendo.

## 1.2 Objetivo

Com o constante crescimento da comunidade DevOps, o suporte e o número de ferramentas e alternativas disponíveis estão aumentando constantemente. Assim, já é possível encontrar diversas ferramentas de DevOps, incluindo artigos, scripts e softwares, que po-

dem ser reusadas para automatizar o processo de colocar um software em produção como é possível observar em [5] e [6].

Esse trabalho tem como objetivo o desenvolvimento e a comparação de estruturas de DevOps. Cada estrutura dessa deverá conter uma das ferramentas usadas em cada fase do processo de desenvolvimento e implantação de software: Bancos de dados, Integração contínua, Colocar em produção ( deployment), Nuvem, IaaS( Infrastructure as a Service), PaaS( Plataforma as a Service), BI, Monitoring, SMC, Gerencia de repositórios, Configuração e Provisionamento, Release Managment, Logging, Build, Testing, Containerization, Colaboration, Security.

Assim, serão desenvolvidas estruturas dessas completas e funcionais, e serão feitas comparações com o objetivo de tentar determinar um parâmetro que possa ser útil na determinação de qual dessas estruturas se deve usar.

### 1.3 Justificativa

Para mostrar a importância desse trabalho, é possível citar alguns casos de sucesso da implementação da metodologia DevOps e analisar as melhorias que essa nova abordagem trouxe para essas organização.

Inicialmente, pode-se citar o grupo empresarial WOTIF GROUP que atua no comércio de viagens com uma plataforma na internet, segundo [7]. Em 2013 e 2014, a organização reorganizou o seu processo de liberação de softwares, reduzindo o tempo médio de liberação de software de semanas para horas, ratificando a importância dessa nova metodologia. Em resumo, uma das principais dificuldades encontradas pela empresa era que seus diversos departamentos de engenharia queriam colaborar nas fases de desenvolvimento de infraestrutura, de teste e de colocar em produção, mas não conseguiam encontrar uma maneira de fazer isso. Assim essa organização conseguiu resolver seus problemas utilizando as técnicas de DevOps e criando uma cadeia de ferramentas que atendeu às suas expectativas.

## 2 Metodologia

## 3 Ferramentas

### 3.1 Banco de Dados

#### 3.1.1 MySQL

É um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Linguagem de Consulta Estruturada) como interface. É atualmente um dos bancos mais populares com mais de 10 milhões de instalações pelo mundo.

### 3.2 Gerenciamento de configurações de software - SCM

#### 3.2.1 Git

É um sistema de controle de versão distribuído e um sistema de gerenciamento de código fonte e de dados com ênfase em velocidade <sup>[9]</sup>. Cada diretório de trabalho do Git é um repositório com um histórico completo e habilidade total de acompanhamento de revisões, não dependente de acesso a uma rede ou a um servidor central.

#### 3.2.2 GitHub

É um serviço de Web Hosting compartilhado para projetos que usam o controle de versionamento Git. Esse oferece todas as funcionalidades do sistema de controle de revisão e gerenciamento de código (SCM) Git com algumas funcionalidades adicionais.

### 3.3 Build

#### 3.3.1 Maven

Apache Maven, ou simplesmente Maven, é uma ferramenta de automação de compilação utilizada primariamente em projetos Java. O Maven utiliza um arquivo XML para descrever o projeto de software sendo construído, suas dependências sobre módulos e componentes externos, a ordem de compilação, diretórios e plug-ins necessários.



## 3.4 Entrega Contínua - CI

### 3.4.1 Jenkins

Jenkins é uma ferramenta open source de integração contínua escrita em Java. Jenkins conta com serviços de integração contínua para desenvolvimento de software. É um sistema de arquitetura servidor rodando em um container servlet tal qual Apache Tomcat. Esse suporta ferramentas SCM incluindo Git e pode ser executado em projetos utilizando Apache Ant e Apache Maven.

## 3.5 Deployment

### 3.5.1 SSH

Parte da suíte de protocolos TCP/IP que torna segura a administração remota de servidores do tipo Unix <sup>[8]</sup>. O SSH possui as mesmas funcionalidades do TELNET com vantagem da criptografia na conexão entre cliente e o servidor. Por meio de criptografia ele cria um canal seguro entre duas máquinas. Também pode ser utilizado para executar comandos remotamente sem entrar no shell da máquina.

## 3.6 Provisionamento

### 3.6.1 Puppet

Utilitário para gerenciamento de configuração de código livre que roda em muitos sistemas Unix compatíveis bem como em Microsoft Windows. Inclui sua própria linguagem declarativa para descrever a configuração do sistema.

### 3.6.2 Ansible

Plataforma de software livre para configuração e gerenciamento de computadores, combina deployment de software multi nós, execução de tarefas ad hoc e gerenciamento de configurações.

### 3.6.3 Vagrant

Vagrant é um software de computador que cria e configura ambientes de desenvolvimento virtuais. Pode ser visto como um software que está em um nível acima dos softwares de virtualização tradicionais, tais como VirtualBox, VMware, KVM e Linux Containers.

## 3.7 Monitoramento

### 3.7.1 Nagios

Aplicação de monitoramento de rede de código aberto distribuída sob a licença GPL. Pode monitorar tanto hosts quanto serviços, alertando quando ocorrerem problemas e também quando os problemas são resolvidos.

## 3.8 Serviços Cloud

### 3.8.1 Amazon Web Services

Plataforma de serviços em nuvem segura oferecendo poder computacional, armazenamento de banco de dados, distribuição de conteúdo e outras funcionalidades para ajudar as empresas em seu dimensionamento e crescimento.

## 3.9 Containers

É um novo modelo de virtualização que trabalha no nível de sistema operacional, ou seja, ao contrário da máquina virtual, um container não tem visão de uma máquina inteira, ele é apenas um processo em execução em um kernel compartilhado entre todos os outros containers.

Eles utilizam o namespace para prover o devido isolamento de memória RAM, processamento, disco e acesso a rede, ou seja, mesmo compartilhamento o mesmo kernel, esse processo em execução tem a visão de estar usando um sistema operacional dedicado.

Assim, os containers são leves por não consumir muitos recursos do sistema uma vez que por usar o mesmo kernel eles podem executar com mais eficiência que uma máquina virtual. Enquanto objetos de software virtuais que incluem todos os elementos que um app precisa para executar, o container tem benefícios de isolamento e alocação de recurso, sendo mais portátil e eficiente que uma máquina virtual. Dessa forma, eles ajudam a construir aplicativos de alta qualidade mais rapidamente.

### 3.9.1 Docker

Como solução inovadora, o Docker traz diversos serviços e novas facilidades que deixam esse modelo muito mais atrativo. Um deles é a criação do conceito de “imagens”, que podem ser descritas como definições estáticas de como os containers devem ser no momento da sua inicialização. São como fotografias de um ambiente. Uma vez instanciadas,

colocadas em execução, elas assumem a função de containers, ou seja, saem da abstração de definição e se transformam em processos em execução, dentro de um contexto isolado, que enxergam um sistema operacional dedicado pra si, mas na verdade compartilham o mesmo kernel.

Junto a facilidade de uso dos containers, o Docker agregou o conceito de nuvem, que dispõe de serviço um para carregar e “baixar” imagens Dockers, ou seja, se trata de uma aplicação web que disponibiliza um repositório de ambientes prontos, onde viabilizou um alto nível de compartilhamento de ambientes.

Com o uso do serviço de nuvem do Docker, percebe-se que a adoção do modelo de containers ultrapassa a questão técnica e adentra nos assuntos de processo, gerência e atualização do ambiente, tornando possível compartilhar facilmente as mudanças e viabilizar uma gestão centralizada das definições de ambiente.

Utilizando a nuvem Docker, é possível disponibilizar ambientes de teste mais leves, acelerando drasticamente o potencial de velocidade com que problemas em ambientes integrados são resolvidos.

Assim, tem-se o Docker como um importante projeto open-source que automatiza o deploy de aplicações dentro de containers de softwares por meio de uma camada adicional de abstração e automação em nível da virtualização do sistema operacional. O uso de características de isolamento de recursos do kernel do Linux (cgroups e namespaces do kernel) permitem a containers independentes rodar dentro de uma instância particular do Linux, evitando o overhead de iniciar e gerenciar máquinas virtuais.

## 4 Arquiteturas

### 4.1 Primeira Arquitetura Customizada

#### 4.1.1 Aplicação

Como o foco do trabalho não é no desenvolvimento em si, mas sim nas práticas Devops que auxiliam o build, deploy e operação de uma aplicação em produção, foi utilizada uma aplicação fictícia baseada em um projeto de código aberto. O projeto será uma loja virtual sobre a plataforma Broadleaf Commerce

<(http://www.broadleafcommerce.org/)>.

O projeto pode ser acessado no seguinte repositório do Github:

<http://github.com/dtsato/loja-virtual/devops/>

Serão utilizadas as funcionalidades padrão de um site de compras online como Submarino ou a Amazon:

- Navegação e busca no catálogo de
- produtos;
- Páginas de produto com nome, descrição, preço, fotos e produtos relacionados;
- Carrinho de compras;
- Processo de checkout customizável incluindo códigos promocionais, dados de cobrança e de entrega;
- Cadastro de usuário;
- Histórico de compras;
- Ferramentas de administração para: catálogo de produtos, promoções, preços, taxas de frete e páginas com conteúdo customizado

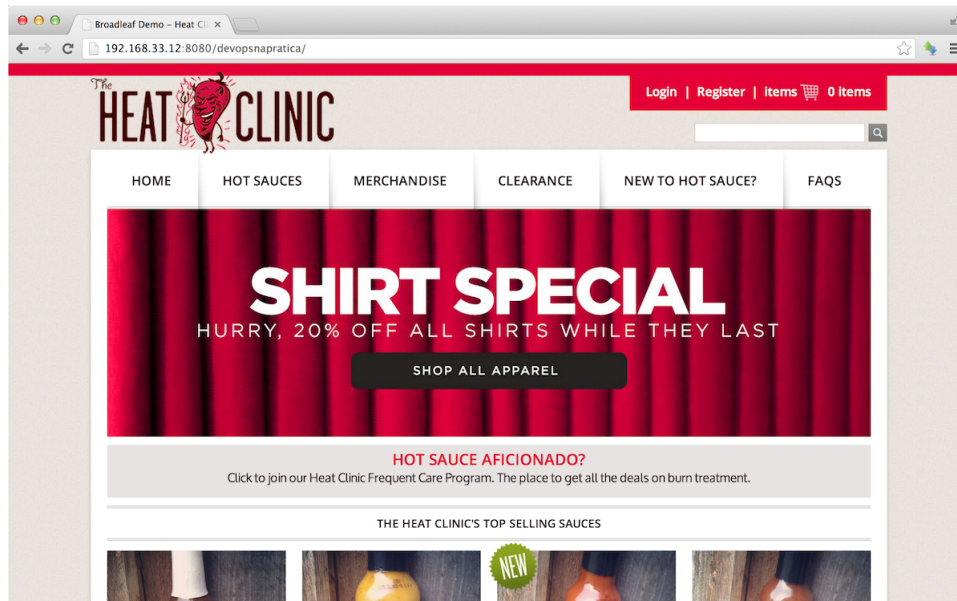


Figura 1 – Uma prévia da da página inicial da loja virtual

As tecnologias utilizadas na aplicação da loja virtual foram:

- **Java:** a aplicação é escrita em Java (<<http://java.oracle.com/>>), compatível com a versão Java SE 6 e superiores.
- **Spring:** o Spring (<<http://www.springframework.org/>>) é um framework de desenvolvimento de aplicações corporativas popular na comunidade Java que oferece diversos componentes como: injeção de dependências, gerenciamento de transações, segurança, um framework de MVC, dentre outros.
- **JPA e Hibernate:** o JPA é a API de persistência do Java e o Hibernate (<<http://www.hibernate.org/>>) é a implementação mais famosa da JPA na comunidade Java, oferecendo recursos para realizar o mapeamento objeto-relacional (object-relational mapping ou ORM) entre objetos Java e as tabelas no banco de dados.
- **Google Web Toolkit:** o GWT (<<http://developers.google.com/web-toolkit/>>) é um framework desenvolvido pelo Google para facilitar a criação de interfaces ricas que rodam no browser. O GWT permite que o desenvolvedor escreva código Java que é então compilado para Javascript. A loja virtual utiliza o GWT para implementar a interface gráfica das ferramentas de administração.
- **Apache Solr:** o Solr (<<http://lucene.apache.org/solr/>>) é um servidor de pesquisa que permite a indexação do catálogo de produtos da loja virtual e oferece uma API eficiente e flexível para efetuar consultas de texto em todo o catálogo.

- **Tomcat:** o Tomcat (<http://tomcat.apache.org/>) é um servidor que implementa as tecnologias web Java Servlet e JavaServer Pages do Java EE. Apesar do Broadleaf Commerce rodar em servidores de aplicação alternativos como Jetty, GlassFish ou JBoss utilizaremos o Tomcat por ser uma escolha comum em diversas empresas rodando aplicações web Java.

#### 4.1.2 Ambiente de Produção

Usuários acessarão a loja virtual em uma instância do Tomcat no servidor web. O servidor web rodará todas as bibliotecas e frameworks Java utilizados pela loja virtual, inclusive uma instância embutida do Solr. Por fim, a aplicação web utilizará o MySQL, rodando em um servidor de banco de dados separado. Esta é uma arquitetura web de duas camadas comumente utilizada por diversas aplicações no mundo real.

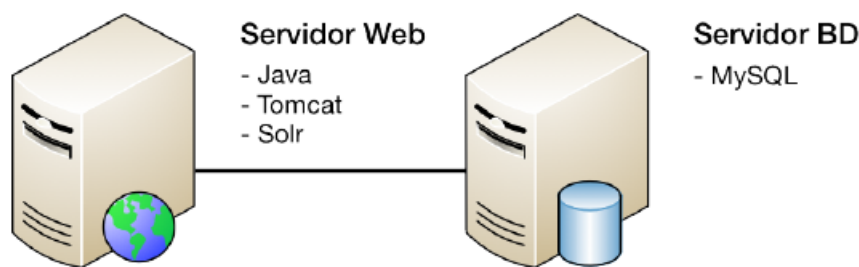


Figura 2 – Ambiente de produção da loja virtual

Será utilizado a ferramenta VirtualBox para instanciação dos dois servidores chamados no presente trabalho de servidor web e servidor bd em um ambiente Linux Ubuntu 12.04 LTS de 32 bits. Para gerenciamento da dos dois servidores foi utilizado o Vagrant. O Vagrant é uma ferramenta para o building completo em ambientes de desenvolvimento. O Vagrant file foi configurado da seguinte forma:

#### 4.1.3 Build e Deploy da Aplicação

Nessa etapa será baixado e compilado o código, rodar os testes, empacotar a aplicação e colocar a loja virtual no ar. Esse processo de compilação, teste e empacotamento é conhecido como build. As etapas do processo de build também podem incluir gerenciamento de dependências, rodar ferramentas de análise estática do código, cobertura de código, geração de documentação, dentre outros. A principal função do processo de build é gerar um ou mais artefatos, com versões específicas, que se tornam potenciais candidatos para release em produção. Para realizar o build foi preciso instalar as seguinte ferramentas:

```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "hashicorp/precise32"

  config.vm.define :db do |db_config|
    db_config.vm.hostname = "db"
    db_config.vm.network :private_network,
                        :ip => "192.168.33.10"
  end

  config.vm.define :web do |web_config|
    web_config.vm.hostname = "web"
    web_config.vm.network :private_network,
                        :ip => "192.168.33.12"
  end
end
```

Figura 3 – Configuração declarada no VagrantFile

- Git (<<http://git-scm.com>>) para controle de versão
- Maven (<<http://maven.apache.org/>>) para executar o build
- JDK (Java Development Kit) para compilação do código Java

As etapas na primeira execução do Maven serão:

- **Sub-projetos:** o projeto está organizado em 4 módulos: core possui as configurações principais, extensões e customizações da loja virtual; site e a aplicação web da loja virtual; admin e uma aplicação web restrita para administração e gerenciamento da loja virtual; por fim, combined e um módulo que agrega as duas aplicações web e o core em um único artefato. Cada módulo atua como um subprojeto que o Maven precisa realizar o build separadamente.
- **Resolução de dependências:** o Maven irá resolver todas as dependências do projeto e baixar as bibliotecas e frameworks necessários para compilar e rodar cada módulo da loja virtual.

- **Compilação:** o código Java e GWT precisa ser compilado para execução. Isso toma um bom tempo na primeira vez. O compilador Java é inteligente o suficiente para recompilar apenas o necessário em builds subsequentes.
- **Testes automatizados:** em módulos que definem testes automatizados, o Maven irá executá-los e gerar relatórios de quais testes passaram e quais falharam.
- **Empacotamento de artefatos:** por fim, o módulo core será empacotado. Como um arquivo .jar enquanto os módulos web (site, admin e combined) serão empacotados como um arquivo .war. Ambos os arquivos jar e war são equivalentes a arquivos zip que você baixa regularmente da internet, no entanto a estrutura interna dos pacotes é bem definida e padronizada pelo Java: o jar contém classes compiladas e serve como biblioteca enquanto o war contém classes, bibliotecas, arquivos estáticos e de configuração necessários para rodar uma aplicação web em um container Java como o Tomcat.

As etapas futuras para o projeto serão:

- Configuração e implementação de um Servidor de Monitoramento utilizando o Nagios (<<http://www.nagios.org/>>).
- Configuração de ferramenta de gerenciamento de configurações para automatizar o processo de provisionamento, configuração e deploy da loja virtual utilizando o Puppet.
- Configuração e estabelecimento de um sistema de controle de versões utilizando o Git (<<http://git-scm.com/>>)
- Configuração e estabelecimento do provisionamento do repositório de pacotes utilizando o Reprero (<<http://mirrorer.alioth.debian.org/>>)
- Realização de Deploy na Nuvem

## 4.2 Segunda Arquitetura Customizada

### 4.2.1 Criação de chave pública SSH

Primeiramente, essa arquitetura se inicia com a configuração de chaves públicas. Na maioria dos provedores, será encontrada uma opção para adicionar uma chave SSH pública. Para criar uma chave RSA local seguiu-se os seguintes passos:



1. Digite o comando a seguir:

```
1 ssh-keygen -t rsa
```

2. Quando for perguntado sobre uma frase digite enter duas vezes.

3. Serão criados dois arquivos:

```
1 id_rsa_devops e id_rsa_devops.pub
```

4. Os arquivos com extensão .pub são os únicos que devem ser copiados para outras máquinas e representam as chaves públicas.

### 4.2.2 Configuração do controle de versão com Git e GitHub

Para gerenciar o versionamento, foi escolhida a ferramenta Git e seguiu-se os seguintes passos para realizar sua instalação:

1. Para linux baseado em debian:

```
1 sudo apt-get install git-core
```

2. Para linux CentOS/RH

```
1 sudo yum install git
```

3. Para MacOSX com homebrew

```
1 brew install git
```

Após a instalação do Git, foram executados os seguintes comandos para configurar um nome de usuário e um email, que serão usados toda vez em que for necessário se conectar ao repositório remoto:

```
1 git config --global user.name "Seu nome"
2 git config --global user.email "seu@email.com.br"
```

Nesse momento o ambiente está com o Git instalado e com nome e e-mail configurados. Será adotado agora um sistema para controlar remotamente os repositórios usados para construir o modelo de DevOps proposto.

O “Github” é um site que fornece um serviço de controle de repositórios remotos de “Git” sem custo, caso o projeto seja aberto, e com custo caso o projeto seja privados.

Para utiliza-lo de forma gratuita, foi acessado o site “github.com”, e foi criada uma conta. Neste ponto, será necessária a chave “ssh” que foi criada anteriormente, que está em:

```
1 ~/.ssh/id_rsa.pub.
```

A partir desse ponto, foi criado um novo repositório que posteriormente será sincronizado com o repositório local que estará em cada máquina que estiver trabalhando no projeto. Para criar esse repositório, foram seguidos os passos do site clicando no botão de novo repositório. Para esse projeto, será dado o nome de projeto-simples. Assim, será criado um diretório local e, posteriormente, será feita a associação desse diretório local com o repositório remoto criado anteriormente no GitHub. Para isso, seguiu-se os passos listados abaixo:

1. Para criar o diretório local foi digitado na linha de comando:

```
1 mkdir projeto-simples
```

2. Entrou-se nesse diretório com o comando:

```
1 cd projeto-simples
```

3. Com um editor de textos, foi criado um arquivo de exemplo, com nome de README.md, para poder ser sincronizado com o repositório remoto criado, explicando a função do projeto. Foi colocado o seguinte texto no arquivo:

```
1 # README do meu projeto-simples
```

4. Esse projeto será apenas um shell script que conta itens únicos no diretório etc. Assim, será criado o seguinte script com o nome de itens\_unicos.sh:

```
1 #!/bin/sh
2 Echo "Itens unicos"
3 Ls /etc | cut -d" -f 1 | sort | uniq | wc -l
```

Nesse momento, já foi criado o projeto simples que será sincronizado com o repositório remoto no github. Foram seguidos os passos a seguir para realizar essa sincronização:

1. Para iniciar o repositório Git no diretório local:

```
1 git init
```

2. Para adicionar todos os arquivos modificados ao conjunto de modificações que serão enviadas para serem sincronizadas com o repositório remoto:

```
1 git add .
```

3. Para criar uma mensagem, descrevendo as modificações:

```
1 git commit -m "mensagem descrevendo a altera o "
```

4. Para vincular o repositório local ao repositório remoto:

```
1 git remote add origin git@github.com:veniciusgrjr/projeto-  
    simples.git
```

5. Para enviar as alterações:

```
1 git push -u origin master
```

Após isso, a página do repositório remoto foi atualizada e os arquivos locais puderam ser visualizados remotamente. Foi feito agora um pequeno resumo dos comandos do Git:

- Esse comando Inicia um repositório Git no diretório atual.

```
1 Git init
```

- Adiciona um ou mais arquivos para serem enviados (commit) ao repositório.

```
1 Git add.
```

- Confirma as mudanças e cria um commit com uma mensaApós isso, faça reload da página do repositório, e os arquivos locais estarão lá.

```
1 Git commit -m "mensagem"
```

- Esse comando adiciona um “remote” ao repositório atual, chamado origin. Você poderia trabalhar sem ter um remote, não é mandatório.

```
1 Git remote ...
```

- Envia (push) as modificações para o repositório remoto. O parametro -u só é necessário na primeira execução.

```
1 Git push -u origin master
```

- Cria uma cópia o repositório dado pela URL para a máquina local em que foi digitado.

```
1 Git clone
```

- Mostra o estado atual do repositório e das mudanças.

```
1 Git status
```

Agora, para testar a ferramenta de sincronização, o diretório local será modificado e sincronizado com o repositório no Github. Para isso, foi criado um arquivo com o nome de `portas.sh`, com o seguinte código:

```
1 #!/bin/sh
2 echo "Lista de porta 80 no netstat"
3 netstat -an | grep 80
```

Para adicionar essa modificação ao repositório no Github, foram executados os seguintes comandos:

```
1 Git add portas.sh
2 Git commit -m "add portas.sh"
3 Git push origin master
```

### 4.2.3 Vagrant e Virtualbox

Uma máquina virtual parada é uma imagem de um disco de metadados que descrevem sua configuração: processador, memória, discos e conexões externas. A mesma máquina em execução é um processo que depende de um scheduler( agendador de processos) para coordenar o uso dos recursos locais. Para gerenciar máquinas virtuais, pode-se usar as interfaces das aplicações VirtualBox, Parallels ou VMW, ou pode-se utilizar bibliotecas e sistemas que abstraem as diferenças entre essas plataformas, com interface consistente para criar, executar, parar e modificar uma máquina virtual. Para criar e gerenciar os ambientes de máquinas virtuais locais, foi escolhido o Vagrant ([www.vagrantup.com](http://www.vagrantup.com)). Para executar as máquinas virtuais, foi escolhido o VirtualBox([www.virtualbox.org](http://www.virtualbox.org) ).

O Vagrant gerencia e abstrai provedores de máquinas virtuais locais e públicos( VMWare, VirtualBox, Amazon AWS, DigitalOcean, entre outros). Ele tem uma linguagem específica (DSL) que descreve o ambiente e suas máquinas. Além disso, ele fornece interface para os sistemas de gerenciamento de configuração mais comuns como Chef, Puppet, CFEngine e Ansible. Ele é um software que cria e configura ambientes virtuais de desenvolvimento. Possui interface de linha de comando simples para subir e interagir com esses ambientes virtuais. Essa ferramenta ajuda na criação da infraestrutura para o projeto, usando para

isso uma máquina virtual. Nesse momento surge um questionamento: será preciso uma máquina virtual para cada projeto, isso não complicaria ainda mais o projeto? A resposta é não. O Vagrant deixa muita coisa invisível, possibilitando se preocupar apenas com o código. Funciona como uma máquina virtual reduzida e portátil. Para cada projeto é possível deixar um ambiente rodando PHP 4, outro PHP 5, outro Debian, outro em CentOS...

Para instalar o Virtualbox, basta acessar o site [www.virtualbox.org](http://www.virtualbox.org), e seguir os passos de instalação. Após isso, para testar se tudo está funcionando, foi feito o download de uma ISO do ubuntu em [www.ubuntu.com/download/server](http://www.ubuntu.com/download/server) e testou-se a criação de máquinas virtuais.

Para instalar o Vagrant, foi acessado [www.vagrantup.com](http://www.vagrantup.com) e, foram seguidos os passos da instalação. Para criar uma máquina virtual usando o Vagrant seguiu-se os paços abaixo:

1. Foi criado um diretório chamado testvm.
2. Foi criado um arquivo chamado vagrantfile e digitado nele:

```
1 # -*- mode: ruby -*-
2 # vi: set ft=ruby :
3
4 VAGRANTFILE_API_VERSION = 2
5
6 Vagrant.configure( VAGRANTFILE_API_VERSION ) do |config|
7
8     config.vm.define testvm do |testvm|
9         testvm.vm.box = ubuntu / trusty64
10        testvm.vm.network : private_network, ip :
11            192 .168.33.21
12    end
13    config.vm.provider virtualbox do |v|
14        v.customize [ modifyvm , :id, --
15            memory , 1024 ]
16    end
17 end
```

3. no diretório desse arquivo criado,para subir o servidor digitou-se:

```
1 vagrant up
```

4. Para testar, foi feita uma conexão com a máquina virtual criada digitando-se:

```
1 vagrant ssh
```

5. É importante notar que dentro da máquina o seu diretório local foi mapeado como um mount point dentro da máquina virtual, acessando o diretório vagrant digitando `cd /vagrant` é possível observar isso. É possível criar este ponto com outro nome ou deixar de criá-lo de acordo com a configuração do Vagrantfile.

6. Para destruir a máquina virtual criada, basta digitar no terminal:

```
1 vagrant destroy
2 //seguido de y, quando for perguntado se pode realmente
   destruir a máquina virtual
```

Assim, resumindo os comandos, temos:

1. Para subir a máquina virtual:

```
1 vagrant up
```

2. Para se conectar à máquina virtual:

```
1 vagrant ssh
```

3. Para destruir a máquina virtual:

```
1 vagrant destroy
```

4. Para desativar a máquina virtual:

```
1 vagrant halt
```

5. Para executar somente o provisionamento:

```
1 vagrant provision
```

Usar os processos fornecidos pelo VirtualBox, por exemplo, para criar máquinas virtuais se torna um processo demorado e difícil de ser replicado. Usando esse Vagrantfile, é possível subir( `up`) a mesma máquina várias vezes, em ocasiões distintas sem usar a interface do VirtualBox. É possível versionar este arquivo que descreve a máquina virtual junto com seu código e quando mudar a configuração de memória, por exemplo, esta mudança estará no histórico junto às mudanças de código.

O Vagrant e sua configuração utiliza Ruby. Existem versões distintas de APIs e nesse trabalho será utilizada a versão 2. Assim, dentro de uma instância da configuração definimos máquinas e suas características, em um bloco de código Ruby. O Vagrant implementa o conceito de provisionador com drivers para quase todos os sistemas de gerenciamento de configuração existentes. Estes sistemas vão desde receitas simples para instalar pacotes até agentes que verificam a integridade de arquivos de configuração e variáveis do sistema. Vamos utilizar um driver de provisionamento do vagrant que permite que um arquivo com comandos do shell seja executado logo após a criação da máquina virtual. Assim será criado o arquivo `webserver.sh` no mesmo diretório e com o seguinte código:

```
1 #!/bin/bash
2
3 echo    Atualizando    repositório
4 sudo apt-get update
5 echo    Instalando    o nginx
6 sudo apt-get -y install nginx
```

Agora o `vagrantfile` será editado para que o provisionamento seja ativado. ele deve ficar com o exposto abaixo:

```
1 # -*- mode: ruby -*-
2 # vi: set ft=ruby :
3
4 VAGRANTFILE_API_VERSION = 2
5
6 Vagrant.configure( VAGRANTFILE_API_VERSION ) do |config|
7
8     config.vm.define testvm do |testvm|
9         testvm.vm.box = ubuntu / trusty64
10        testvm.vm.network : private_network, ip : 192
11            .168.33.21
12        testvm.vm.provision shell , path:
13            webserver . sh
14    end
15    config.vm.provider virtualbox do |v|
16        v.customize [ modifyvm , :id, --memory ,
17            1024 ]
18    end
19 end
```

Agora a máquina que está rodando será destruída com `vagrant destroy` e criada novamente com `vagrant up`. Observando a saída do terminal, e fazendo o teste no browser, percebemos que o `nginx` foi instalado com sucesso sem a necessidade de se conectar por `ssh` para executar o comando que instala o `nginx` na máquina virtual.

Vamos instalar, agora, o `PHP5` na máquina virtual. Para isso, será modificado o arquivo `webserver.sh` de acordo com o código a seguir:

```
1 #!/bin/bash
2
3 echo    Atualizando    r e p o s i t o r i o
4 sudo apt-get update
5 echo    Instalando    o  n g i n x
6 sudo apt-get -y install nginx
7 echo    instalando    P H P
8 sudo apt-get install -y php5-fpm
```

Nesse ponto, não foi preciso destruir e recriar a máquina para executar novamente o provisionamento. Foi aproveitado o fato de que o gerenciador de pacotes não instalará um pacote duas vezes. Assim, é possível executar o mesmo script, que só o `PHP` seria instalado. O `Vagrant` oferece um comando que ativa somente a fase do provisionamento, `vagrant provision`. Assim Executando `vagrant provision` no terminal, observa-se a execução do script. Note que os comandos do `Vagrant` só funcionaram dentro do diretório em que o `vagrantfile` está. Com o comando `ls -larth` é possível ver que foi criado um diretório chamado `.vagrant` que contém todos os dados do ciclo de vida e provisionamento da máquina virtual.

#### 4.2.4 Ansible

É um sistema de automação de configuração feito em `python`, que permite descrever procedimentos em arquivos no formato `YAML` que são reproduzidos utilizando `SSH` em máquinas remotas. Existem outras ferramentas desta categoria que executam localmente e que fornecem servidores para o gerenciamento de dados remotamente, como `CHEF`, `Puppet` e `CFEngine`. Vamos utilizar o `Ansible` localmente sem seu servidor de dados, o `Ansible Tower`. O `Ansible` vem com bibliotecas completas para quase todas as tarefas além de uma linguagem de template. Além das tarefas dentro de um servidor, o `Ansible` fornece módulos para o provisionamento de máquinas e aplicações remotas. Provisionar é o jargão para instalar e configurar itens de infraestrutura e plataforma como máquinas, bancos de dados e balanceadores de carga.



Sistemas como o Ansible implementam tarefas com uma característica importante: idempotência. A idempotência é uma propriedade que, aplicada ao gerenciamento de configuração, garante que as operações terão o mesmo resultado independentemente do momento em que serão aplicadas. A criação de uma máquina utilizando este conjunto de configurações sempre terá o resultado previsível.

Para instalar no ubuntu, utilizou-se os seguintes comandos:

```
1 sudo apt-get install software-properties-common
2 sudo apt-add-repository ppa:ansible/ansible
3 sudo apt-get update
4 sudo apt-get install ansible
```

Para testar se foi instalado corretamente, foi digitado no terminal `ansible-playbook -h`, para verificar se aparece a ajuda do Ansible.

Para configurar o Ansible, foi criado no diretório `testvm` o arquivo com o código a seguir, com o nome de `webserver.yml`:

```
1 - hosts: all
2   sudo: True
3   user: vagrant
4   tasks:
5     - name: "Atualiza pacotes"
6       shell: sudo apt-get update
7     - name: "Instala o nginx"
8       shell: sudo apt-get -y install nginx
```

Esse formato de arquivo funciona como um dicionário no formato: chave: valor. Agora, temos que reconfigurar o Vagrant, para isso o arquivo `vagrantfile` deve ficar do modo a seguir:

```
1 # -*- mode: ruby -*-
2 # vi: set ft=ruby :
3
4 VAGRANTFILE_API_VERSION = 2
5
6 Vagrant.configure( VAGRANTFILE_API_VERSION ) do |config|
7
8     config.vm.define testvm do |testvm|
9         testvm.vm.box = ubuntu / trusty64
```

```

10         testvm.vm.network : private_network, ip : 192
           .168.33.21
11         testvm.vm.provision    ansible    do |ansible|
12             ansible.playbook =  webserver . yml
13             ansible.verbose =   vvv
14         end
15     end
16     config.vm.provider    virtualbox    do |v|
17         v.customize [    modifyvm    , :id,    --memory ,
           1024    ]
18     end
19 end

```

A definição de máquina virtual de testvm tem um atributo que define um provisionador. No capítulo anterior, foi utilizado um shell script para instalar o nginx. Com essa mudança, o Vagrant utilizará o Ansible como provisionador com o playbook webserver.yml. Usou-se o atributo verbose do Ansible com o valor vvv para indicar que todas as mensagens devem ser enviadas para o console, e assim podemos observar os comandos sendo executados.

Toda task é uma função de um módulo de Ansible. Os módulos que vêm na instalação padrão são chamados de Core Modules. É possível construir módulos utilizando a linguagem Python e estender o Ansible. Agora o playbook precisa ser refatorado, para utilizar um módulo core chamado apt\_module ([http://docs.ansible.com/apt\\_module.html](http://docs.ansible.com/apt_module.html)) em vez do módulo shell. O módulo shell é útil para automatizar muitas tarefas, mas o Ansible possui módulos especializados que cuidam da consistência da tarefa e criam estados para manter a idempotência. Assim, o arquivo webserver.yml deve ficar desse modo:

```

1 - hosts: all
2   sudo: True
3   user: vagrant
4   tasks:
5     - name: "Atualiza pacotes e instala nginx"
6       apt: name=nginx state=latest update_cache=yes
7       install_recommends=yes

```

Note que duas tarefas foram reduzidas a uma que utiliza o módulo apt e diz para o Ansible instalar o nginx na última versão presente no repositório. Um dos atributos desta tarefa está indicando que um update no cache do gerenciador de pacotes deve ser executado. Outro atributo indica que as dependências recomendadas devem ser instaladas

automaticamente. Note que, utilizando o módulo shell, foi preciso pedir explicitamente pela atualização do repositório e também adicionar o parâmetro `-y` ao comando `apt-get install` para evitar que a task ficasse esperando uma entrada do usuário.

O Ansible também fornece um módulo para o gerenciador de pacotes yum (<http://docs.ansible.com>) e diretivas condicionais que podem ser usadas para detectar o sistema operacional e distribuições de Linux. Veja como declarar a mesma task para instalar nginx para duas famílias de distribuições de Linux utilizando o condicional “when” e variáveis internas do Ansible:

```
1 - name: Install the nginx packages
2   yum: name={{ item }} state=present
3     with_items: redhat_pkg
4     when: ansible_os_family == "RedHat"
5 - name: Install the nginx packages
6   apt: name={{ item }} state=present    update_cache=yes
7     with_items: ubuntu_pkg
8     environment: env
9     when: ansible_os_family == "Debian"
```

Para testar o novo `webserver.yml`, basta digitar `vagrant up` ou apenas `vagrant provision` caso sua máquina virtual ainda esteja sendo executada. Execute o provisionamento em seguida para notar a diferença entre a primeira e a segunda execução. Esta é uma maneira simples de verificar a idempotência do playbook: na segunda rodada, o atributo `changed` deve ser 0.

#### 4.2.5 Instalando Wordpress em uma máquina

### 4.3 Arquitetura proprietária

Como caso de estudo de uma arquitetura proprietária que automatiza os processos de DevOps, foi escolhido o IBM Bluemix.

#### 4.3.1 Introdução ao Bluemix

A IBM desenvolveu o IBM Bluemix com a ideia de possibilitar a construção de aplicativos sem se preocupar com a infraestrutura, deixando esta a cargo do Bluemix. Assim, construído com base nos projetos open source mais populares do mundo, o IBM Bluemix é uma plataforma em nuvem que permite desenvolvedores construir e executar os apps e serviços mais modernos em tendência no mercado.

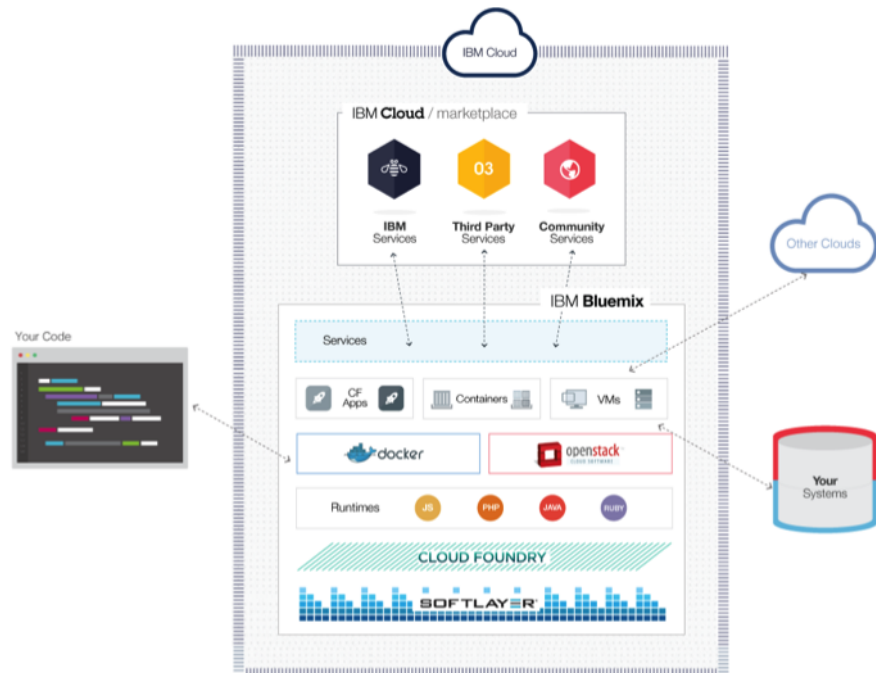


Figura 4 – Estrutura do Bluemix

O Bluemix também é o mais recente produto de nuvem da IBM. Ele permite às organizações e aos desenvolvedores uma maneira rápida e fácil de criar, realizar o deploy e gerenciar aplicações na nuvem. Ele é uma implementação da IBM *Open Cloud Architecture* baseada no *Cloud Foundry*, uma plataforma como serviço (mais conhecido como *PaaS – Platform as a Service*). Um de seus maiores diferenciais é entregar serviços de nível empresarial que permitem integrar facilmente as aplicações em nuvem sem precisar se preocupar com questões de instalação e configuração das mesmas, otimizando dimensões de custo e produtividade.

#### 4.3.2 Devops e Bluemix

É o serviço premium de DevOps da Plataforma de nuvem da IBM, que promove um mecanismo de adoção sem fricção e incremental dos serviços de DevOps para o Bluemix, tornando as fases de planejamento ágil e integrado, codificação, *Building* e *deploying* mais fáceis.

Enquanto experiência integrada para o desenvolvedor, a plataforma oferece:

- Solução na nuvem para desenvolver aplicações
- Integra gerenciamento de tarefas, planejamento ágil, controle de recursos

- Possibilidade de uso tanto das ferramentas pessoais do desenvolvedor ou da Web IDE
- Escalável, seguro e pronto para aplicações empresariais: roda na infraestrutura do *SoftLayer*

A ideia geral é que o Bluemix traga as ferramentas automatizadas de infraestrutura e o desenvolvedor se preocupe exclusivamente com o código.

### 4.3.3 Serviços de DevOps disponíveis

Enquanto principais funcionalidades que fazem do Bluemix o serviço de DevOps na plataforma de nuvem da IBM, são explicitados como recursos de DevOps disponíveis na plataforma:

- Acesso fácil: com repositório Git e a Web IDE que já vem construída, é possível começar a codificação dos apps e serviços instantaneamente
- Versatilidade de ferramentas: além da Web IDE que é embutida, é possível usar o Eclipse, Visual Studio ou a ferramenta de escolha do desenvolvedor
- *Build & deploy*: automaticamente construir e realizar o deploy das aplicações para o Bluemix
- Trabalho em equipe: compartilhar o trabalho e colaborar através de ferramentas especializadas para o desenvolvimento ágil

Por exemplo, em um projeto público ou privado no Bluemix, é possível facilmente convidar novos membros ao time, acessar o código de qualquer lugar, construir colaborativamente desde o início, escolher quem vê o projeto e como ele se engaja com comunidades externas.

Nos projetos públicos, é fácil ter acesso e compartilhar trabalho com uma audiência mais ampla. Nos privados, somente é possível compartilhar com a equipe do projeto.

O desenvolvimento ágil na nuvem também é facilitado com os serviços de DevOps do Bluemix por funcionalidades como:

- Suporte aos processos do desenvolvimento ágil embutido

- Ítems de trabalho para planejar e gerenciar atividades dos projetos
- Ferramentas ágeis para o *backlog* do produto, releases e sprints
- Dashboard de gráficos para o status do projeto

Além disso, é possível escolher onde será feito o desenvolvimento dos serviços:

- *Browser*, através da IDE embutida no próprio Bluemix
- Desenvolvimento local, através da integração com o Eclipse ou Visual Studio
- *Jazz Source Control*, que também tem suporte embutido no Bluemix
- Repositório git
- GitHub

A experiência de DevOps Bluemix, portanto, pode ser resumida através do seguinte diagrama:

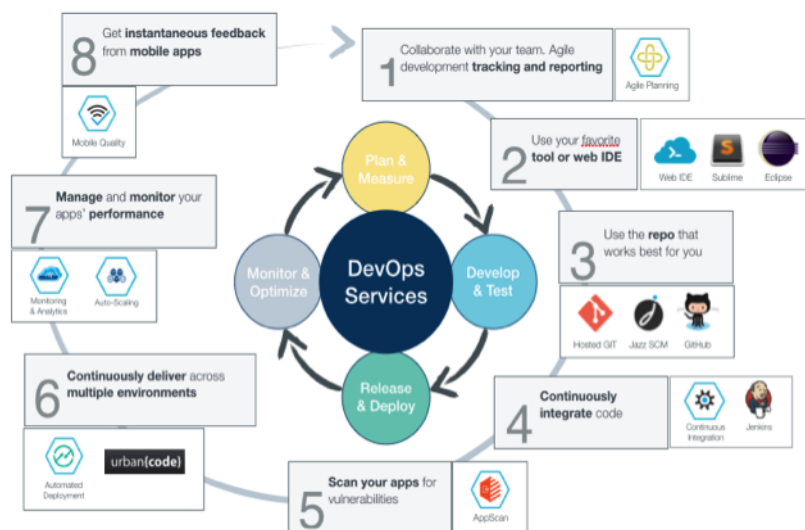


Figura 5 – Roadmap da experiência de DevOps no Bluemix

#### 4.3.4 Serviços

O conceito de serviços no Bluemix possui um caráter interessante e peculiar. Ele fornece serviços que podem ser usados pelas aplicações sem a necessidade de gerenciar a configuração e operação desses serviços.

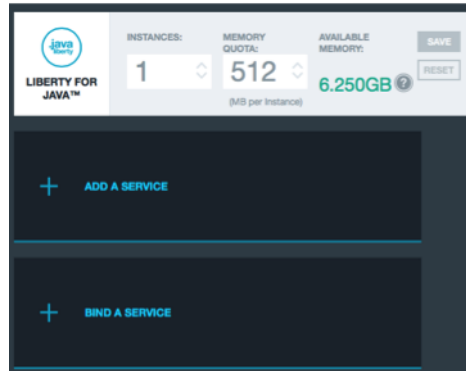


Figura 6 – Tela para adicionar ou ligar serviço no Bluemix

Os serviços disponíveis são listados em um catálogo na Web UI e também podem ser obtidos usando o comando:

```
1 cf marketplace
```

Para ligar um serviço a uma aplicação que desejar usá-lo, o comando é:

```
1 cf bs
```

Depois de ligar o serviço à aplicação, o Bluemix adicionará detalhes sobre o serviço a uma variável do ambiente que será parseada pelas aplicações.

Um fato interessante no contexto dos serviços no Bluemix é que além de consumir serviços, também é possível criar novos. Enquanto os serviços privados são disponíveis exclusivamente para a organização do usuário, os públicos podem ser adicionados na *IBM Cloud Marketplace* e se tornar um recurso adicional de receita, o que pode ser feito através do estreitamento de relações com a IBM. Para isso, basta que os serviços rodem ou sejam implantados na *SoftLayer* (plataforma de nuvem da IBM), ou se integrem com um dos serviços de plataforma premium da IBM.

### 4.3.5 Características

Para efetivamente entregar o valor prometido, o Bluemix foi concebido de forma a preservar pontos que estejam adequados aos desafios de negócio das empresas e às expectativas dos desenvolvedores. Assim, se destacam como pontos fortes inerentes ao desenvolvimento e ao uso da plataforma:

- *Browser*, através da IDE embutida no próprio Bluemix
- Design centrado no usuário

- Princípios e práticas alinhadas às metodologias ágeis
- Processos e ferramentas de DevOps
- Arquitetura para a nuvem

Nesse contexto, o Bluemix possui um forte viés para a importância do desenvolvimento dirigido a testes (não há código que seja escrito se não for para passar em um teste). Os serviços de DevOps no Bluemix podem ser configurados para automaticamente rodar testes e realizar o deploy no Bluemix se o código passar nos testes, obedecendo a um pipeline de DevOps.

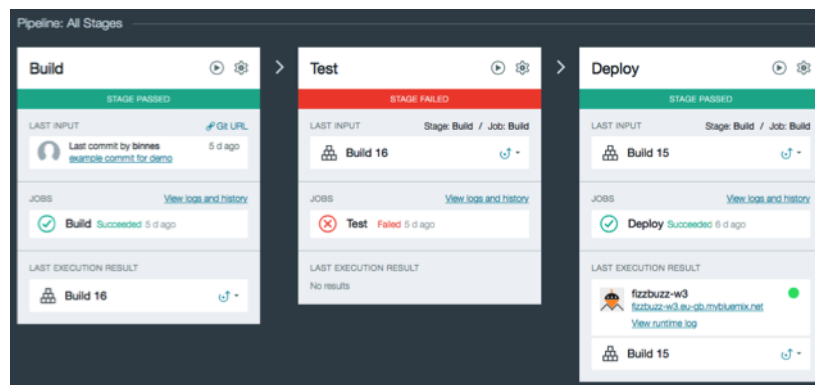


Figura 7 – Pipeline no Bluemix

Assim, são benefícios diferenciados inerentes às funcionalidades do Bluemix:

- Contruir e rodar apps: uso de poderosas tecnologias open source em tempo de execução, de containers e máquinas virtuais para potencializar os apps e serviços
- Acessar dados e aplicações de qualquer lugar: transformação de dados não minerados em informação de valor para o consumo e uso em aplicações de produção
- Usar modelos de *deployment* híbridos e flexíveis: o Bluemix se apresenta como uma plataforma única para várias necessidades, consistente entre nuvens públicas, dedicadas e baseadas em condições. É possível ser iniciado de qualquer lugar e facilmente expandir a estratégia com o passar do tempo

#### 4.3.6 Requisitos Mínimos

Para o sucesso máximo na execução do Bluemix, são requisitos mínimos do software de *Browser*:



- *Chrome*: versão atualizada do sistema operacional
- *Firefox*: versão atualizada do sistema operacional e ESR 38
- *Internet Explorer*: versão 10 ou 11
- *Safari*: versão mais atualizada do *Browser*
- Interface da linha de comando do *Cloud Foundry*: versão 6.5.1 ou posterior

### 4.3.7 Bluemix na prática

### 4.3.8 Prova de conceito

O Bluemix permite diversas possibilidades de desenvolvimento de aplicações. Como proposta de estudo e para fins de normalização de uma instância de projeto, visando comparar com outras alternativas de serviços de DevOps, conforme objetivo do trabalho, serão desenvolvidos no contexto do Bluemix, com documentação e estudo de métricas nos pontos aplicáveis, os seguintes trabalhos:

1. Criação de um app Java usando o IBM Bluemix e seus serviços de DevOps
2. Métricas para localizar e realizar o fork de um app java
3. Configurar Builds automáticos e deploys pelo Bluemix
4. Editar um app Java pelo Eclipse e configurar um trigger para deploy no Bluemix
5. Vulnerabilidades em caso de falhas do desenvolvedor
6. Convidar outros usuários para ajudar no desenvolvimento do app
7. Gerenciamento do desenvolvimento do app
8. Planejamento dos sprints no app

## 5 Cronograma

Atividades	Mar-1	Mar-2	Abr-1	Abr-2	Mai-1	Mai-2	Jun-1	Jun-2	Jul-1	Jul-2	Ago-1	Ago-2	Set-1	Set-2
Linux, SSH, Git	X													
Vagrant	X	X												
Ansible		X	X											
Instalando Wordpress			X											
Proxy reverso			X	X										
Cassandra e EC2			X	X										
Métricas de monitoração				X	X									
Análise de performance em cloud com new Relic				X	X									
Docker				X	X									
Em produção					X	X	X							
Testes integrados									X	X				
Ajustes											X	X		

### 5.1 Luan Ferreira Cardoso

Atividades	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out
Pesquisa inicial	X							
Configurando ambiente de produção		X						
Configuração build e deploy		X						
Monitoramento			X					
Sistema de gerenciamento de configuração			X					
Sistema de controle de versões				X				
Provisionamento de repositório de pacotes					X			
Deploy na nuvem						X		
Testes integrados							X	

## 6 Conclusão Parcial

Texto Conclusão

# Referências

- 1 HUMBLE, J.; MOLESKY, J. Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, v. 24, n. 8, p. 6, 2011.
- 2 HUMBLE, J.; FARLEY, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. [S.l.]: Pearson Education, 2010.
- 3 HTTERMANN, M. *DevOps for developers*. [S.l.]: Apress, 2012.
- 4 LOUKIDES, M. *What is DevOps?* [S.l.]: " O'Reilly Media, Inc.", 2012.
- 5 NELSON-SMITH, S. *Test-Driven Infrastructure with Chef: Bring Behavior-Driven Development to Infrastructure as Code*. [S.l.]: " O'Reilly Media, Inc.", 2013.
- 6 SABHARWAL, N.; WADHWA, M. *Automation through Chef Opscode: A Hands-on Approach to Chef*. [S.l.]: Apress, 2014.
- 7 CALLANAN, M.; SPILLANE, A. Devops: Making it easy to do the right thing. IEEE.
- 8 BARRET, D. J.; SILVERMAN, R. E.; BYRNIS, R. G. *SSH, The Secure Shell: The definitive Guide*. [S.l.]: " O'Reilly Media, Inc.", 2005.
- 9 LOELIGER, J.; MCCULLOUGH, M. *Version Control with Git: Powerful tools and techniques for collaborative software development*. [S.l.]: " O'Reilly Media, Inc.", 2012.