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
9 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

9 de maio de 2016

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

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.

24f. : 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, 9 de maio de 2016:

Prof. Clayton Escouper das Chagas
Orientador, M. Sc., do IME

Prof. Humberto
Convidado, M. c., do IME

Prof. Chorem
Convidado, D. c., do IME

Rio de Janeiro

9 de maio de 2016

Sumário

Resumo

Resumo em pt

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

Abstract

Abstract in English

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 claboração e cominicação 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. Essas 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 [1]. 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 [2], 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 [3]. Nesse cenário, surgiu uma tendência de tentar criar estruturas automatizadas que pudessem facilitar a integração desses processos de desenvolvimento de sistemas [1], englobando todas as fazes do processo de desenvolvimento de softwares e sistemas.

A partir desse momento, os administradores não mas 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 [4], 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), Núvem, IaaS(Infrastructure as a Service), PaaS(Plataform as a Service), BI, Monitoring, SMC, Gerencia de repositórios, Configuração e Provisionamento, Release Managiment, Logging, Build, Testing, Conteinerization, 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

2.1 Modelo do cardoso!!!!!

2.2 Modelo baseado no livro Caixa de Ferramentas DevOps

2.2.1 Ferramentas básicas: SSH, Git

2.2.1.1 SSH

É o protocolo utilizado para conectar-se de forma segura em servidores [8]. 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. O Ansible, ferramenta que será usada mais afrente, só precisa de acesso por SSH.

Na maioria dos provedores, será encontrada uma opção para adicionar uma chave SSH pública. Para criar uma chave RSA local segue-se os seguintes passos:

1. Digite o comando a seguir:

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

2. Quando for perguntado sobre uma frase digite enter duas vezes.
3. Serão criados dois arquivos:

```
id_rsa_devops e id_rsa_devops.pub
```

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

2.2.1.2 Git

O Git é um sistema de controle de versão gratuita. Em termos práticos, sua função é controlar mudanças em repositórios de dados [9]. Essa ferramenta controla repositórios de dados, pois, além de código-fonte de programas, também é possível controlar repositórios para a constução de livros, e qualquer outra atividade que em que várias pessoas trabalham ao paralelamente na sua construção.

Para instalar essa ferramenta, serão seguidos os seguintes passos:

1. Para linux baseado em debian:

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

2. Para linux CentOS/RH

```
sudo yum install git
```

3. Para MacOSX com homebrew

```
brew install git
```

Após instalar o Git, execute os seguintes comandos para configurar um nome de usuário e um email:

```
git config --global user.name "Seu nome"
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 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 começar a utilizalo de forma gratuita, deve-se acessar o site “github.com” e criar uma conta. Neste ponto, será necessária a chave “ssh”, que está em:

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

Agora, será 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, basta seguir os paços do site clicando no botão de novo repositório. Para esse projeto, será dado o nome de projeto-simples. Agora, será criado um diretório local e, posteriormente, será feita a associação desse diretório local com o repositório remoto criado anteriormente no GitHub. Para isso, deve-se seguir os passos listados abaixo:

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

```
mkdir projeto-simples
```

2. Entre nesse diretório com:

```
cd projeto-simples
```

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

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

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

```
#!/bin/sh
Echo "Itens unicos"
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. Agora seguiremos os passos a seguir para realizar essa sincronização:

1. Para iniciar o repositório Git no diretório local:

```
git init
```

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

```
git add .
```

3. Para criar uma mensagem, descrevendo as modificações:

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

4. Para vincular o repositório local ao repositório remoto:

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

5. Para enviar as alterações:

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

Após isso, se a página do repositório remoto for atualizada, os arquivos locais estarão lá. Será feito agora um pequeno resumo dos comandos do Git:

- Esse comando Inicia um repositório Git no diretório atual.

```
Git init
```

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

```
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.

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

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

```
Git remote ...
```

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

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

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

```
Git clone
```

- Mostra o estado atual do repositório e das mudanças.

```
Git status
```

Agora, o diretório local será modificado e sincronizado com o repositório no Github. Para isso, será criado um arquivo com o nome de portas.sh, com o seguinte código:

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

Para adicionar essa modificação ao repositório no Github:

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

2.2.2 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). Para executar as máquinas virtuais, foi escolhido o VirtualBox(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 especifica (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 invísivel, possibilitando se preocupar apenas com o código. Funciona como uma máquina virtual reduzida e portável. 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, 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 e testou-se a criação de máquinas virtuais.

Para instalar o Vagrant, foi acessado 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 :
                       192  .168.33.21
11         end
12         config.vm.provider     virtualbox     do |v|
13                 v.customize [    modifyvm    , :id,    --
                      memory ,     1024    ]
14         end
15 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 m quina virtual
```

Assim, resumindo os comandos, temos:

1. Para subir a máquina virtual:

```
1  vagrant up
```

2. Para se conectar à máquina virtual:

```
vagrant ssh
```

3. Para destruir a máquina virtual:

```
vagrant destroy
```

4. Para desativar a máquina virtual:

```
vagrant halt
```

5. Para executar somente o provisionamento:

```
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:

```
#!/bin/bash

echo   Atualizando   repositorio
sudo apt-get update
echo   Instalando   o  nginx
sudo apt-get -y install nginx
```

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

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION =    2

Vagrant.configure( VAGRANTFILE_API_VERSION ) do |config|

        config.vm.define     testvm     do |testvm|
                testvm.vm.box =   ubuntu  /trusty64
                testvm.vm.network : private_network, ip :   192
                   .168.33.21
                testvm.vm.provision     shell    , path:
                     webserver   . s h
        end
        config.vm.provider     virtualbox     do |v|
                v.customize [    modifyvm    , :id,    --memory ,
                       1024    ]
        end
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:

```
#!/bin/bash

echo   Atualizando   repositorio
sudo apt-get update
echo   Instalando   o  nginx
sudo apt-get -y install nginx
echo   instalando    PHP
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.

2.2.3 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 sevidores 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 lingua-

gem 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) 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.

2.2.4 Instalando Wordpress em uma máquina

2.3 Modelo do Zalla!!!!!

3 Cronograma do Rocha

Atividades	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out
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			
Analise de performance em cloud com new Relic				X	X			
Docker				X	X			
Em produção					X	X	X	

4 Cronograma do 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		

5 Conclusão

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.