CURSO DE SISTEMAS DE INFORMAÇÃO

QUALIFICAÇÃO DE AMBIENTE DE ENTREGA DE APLICAÇÃO EXTERNA UTILIZANDO CONCEITO DEVOPS APLICADO AO CONTEXTO DE UMA UNIVERSIDADE PÚBLICA

MATHEUS JOSÉ ALVES SILVA SANTOS

Palmas

2018

CURSO DE SISTEMAS DE INFORMAÇÃO

QUALIFICAÇÃO DE AMBIENTE DE ENTREGA DE APLICAÇÃO EXTERNA UTILIZANDO CONCEITO DEVOPS APLICADO AO CONTEXTO DE UMA UNIVERSIDADE PÚBLICA

MATHEUS JOSÉ ALVES SILVA SANTOS

Projeto apresentado como requisito para aprovação na disciplina de Projeto de Conclusão de Curso de Sistemas de Informações da Universidade Estadual do Tocantins- UNITINS, sob a orientação do professor Me. Douglas Chagas da Silva.

Palmas

2018

1. INFORMAÇÕES DO ACADÊMICO:

Nome: Matheus José Alves Silva Santos Matrícula: 2015101100100005

Período VIII

E-mail: matheusetf@gmail.com Telefone: (63) 99285-7729

2. INFORMAÇÕES DO PCC:

Professor Orientador: Douglas Chagas da Silva

Início das atividades: 13/08/2018 Término das atividades: 20/12/2018

Total de horas semanais dedicadas ao estágio supervisionado: 20 horas

Área de realização do estágio: Infraestrutura Ágil

Data: ____/____/____

3. PARECER DO PROFESSOR ORIENTADOR:

Aprovado ( ) Reprovado ( ) Nota: _____

OBSERVAÇÕES:

DATA: ____/____/____

_____

Professor Orientador

4. PARECER DO COORDENADOR DE ESTÁGIO:

OBSERVAÇÕES:

DATA: ____/____/____

_____

Coord. de Estágio Supervisionado

Palmas

2018

Este trabalho é dedicado à minha família, pelo apoio incondicional.

Agradecimentos

À Deus, pela graça de permitir o meu crescimento no campo de estudo em que me sinto feliz e realizado.

Aos meus pais e meus irmãos que sempre apostaram em minha capacidade de crescimento e superar as barreiras que a vida impõe.

À minha esposa que tem sido compreensiva nos momentos de dificuldades, e companheira nos desafios que surgem.

Aos meus amigos e colegas de universidade, que sempre contribuíram para o desenvolvimento do meu conhecimento e pelas parcerias nos projetos e estudos.

"Não só isso, mas também nos gloriamos nas tribulações, porque sabemos que a tribulação produz perseverança; a perseverança, um caráter aprovado; e o caráter aprovado, esperança. E a esperança não nos decepciona, porque Deus derramou seu amor em nossos corações, por meio do Espírito Santo que Ele nos concedeu."

(Bíblia Sagrada, Romanos 5:3-5)

Resumo

Este projeto apresentará um estudo sobre diversas ferramentas de automação da entrega de software, visando estabelecer uma proposta de cenário que viabilize esse processo dentro de uma cultura DevOps. O cenário deve funcionar como um provedor de serviços sobre plataforma ágil.

Palavras-chaves: Automação, DevOps, Modelo Ágil.

Lista de ilustrações

Figura 1 – Ciclo DevOps 12
Figura 2 – Etapas do Atlas 14
Figura 3 – Comparativo Docker x Máquinas Virtuais 17
Figura 4 – Fluxo commit e request 20
Figura 5 – Representação da Arquitetura Proposta 24

Lista de abreviaturas e siglas

DevOps - Development and Operational (Desenvolvimento e Operacional).

AWS - Amazon Web Services

SSH - Secure Shell

NETCONF - Network Configuration

Sumário

1 INTRODUÇÃO 10
1.1 Objetivos 11
1.1.1 Objetivo Geral 11
1.1.2 Objetivos Específicos 11

2 REFERENCIAL TEÓRICO 12
2.1 Infraestrutura como Código 13
2.2 Principais Ferramentas e Recursos Tecnológicos Disponíveis 14
2.2.1 Atlas 14
2.2.2 Chef 15
2.2.3 Docker 15
2.2.3.1 Vantagens: 16
2.2.4 Puppet 17
2.2.5 Ansible 18
2.2.5.1 Componentes 19
2.2.5.2 Vantagens 19
2.2.6 GitHub 20
2.2.7 Jenkins 20
2.2.7.1 Vantagens 21

3 METODOLOGIA 22
3.1 Materiais Utilizados 22
3.2 Arquitetura e Pipeline 23
3.3 Métricas 25

4 RESULTADOS 26

5 CONCLUSÃO 27

REFERÊNCIAS 28

A SCRIPTS DE CONFIGURAÇÃO DE AMBIENTE 30
A.1 Instalação do Docker 30
A.2 Jenkins 30
A.3 Ansible 36

1 Introdução

As mudanças econômicas refletem demandas de mercado. Isso têm impacto fundamental na forma como os recursos são disponibilizados e os serviços são exigidos. Nesse sentido, a tecnologia da informação, e mais especificamente nesse estudo, a entrega de um produto ou serviço deve dar um suporte a altura das cobranças de mercado.

O cenário atual que envolve os processos de desenvolvimento e infraestrutura estão ficando cada vez mais defasados. Tradicionalmente o processo de *deploy*[1] e gestão de entrega de soluções, tende a tornar a rotina de produção e disponibilização de serviços demasiadamente morosa. Em termos atuais, o usuário final e o próprio mercado demandam agilidade e rápida resposta.

A concepção de um modelo de entrega contínua que permita ganho na produtividade e fácil detecção de erros no processo de *deploy*, tem sido cada vez mais requisitado no meio da gestão da tecnologia de informação aliado a uma cultura *DevOps*[2]. Tomando isso como base, é importante entendermos que os prazos de entrega estão cada vez mais apertados, o que influi no aumento da carga de trabalho, refletindo diretamente no desempenho do profissional, tanto de desenvolvimento quanto de infraestrutura.

O presente projeto tem como foco a formulação de um modelo que permita essa entrega contínua e automatizada no que diz respeito o *deploy* de aplicações em tempo consideravelmente reduzido. Passa-se antes pelo estudo das diversas ferramentas disponíveis no mercado de software, a fim de levantar as possibilidades de aplicação nesse modelo. Sendo assim, serão apresentadas as ferramentas eleitas como mais viáveis para o projeto.

É vital então percebermos a real vantagem no uso do modelo baseado na visão DevOps, a redução na carga de trabalho e a comunicação massiva entre os times de infraestrutura e desenvolvimento dentro de um mesmo projeto, e a entrega ágil do produto trabalhado.

A estrutura desse trabalho está organizado da seguinte forma: primeiramente são traçados os objetivos. Na seção 2 será apresentado o referencial teórico utilizado para embasar o tema e a ideia proposta, de acordo com todas as ferramentas estudadas, a fim de utilização posterior em um cenário de testes. Na seção 3 serão apresentados os métodos utilizados para compor o estudo, assim como as ferramentas utilizadas durante todo o processo. Na seção 4, são apresentados os resultados referente ao projeto. Por fim, menciona-se as referências utilizadas no referido estudo.

[1] Publicação de um determinado software ou serviço para uso.
[2] Development and Operational (Desenvolvimento e Operacional).

1.1 Objetivos

1.1.1 Objetivo Geral

Configurar e construir um cenário de automatização de deploys considerando a Universidade Estadual do Tocantins - UNITINS, como ambiente macro.

1.1.2 Objetivos Específicos

- Realizar estudos das principais ferramentas disponíveis;
- Definir as métricas utilizadas para avaliar o desempenho do cenário proposto;
- Apresentar as configurações de cada ferramenta adotada.

2 Referencial Teórico

O conceito de DevOps pode ser entendido como o nivelamento entre as equipes de desenvolvimento e operações no que tange suas interações, mantendo funções específicas, porém alinhando suas demandas referentes às responsabilidades e processos, visando a disponibilização de um produto ou funcionalidade de forma rápida e confiável.(GARTNER, 2018a)

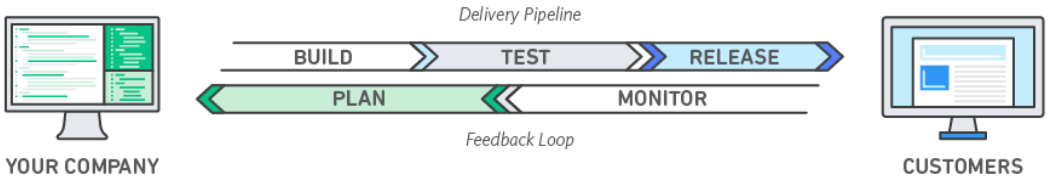

Figura 1 – Ciclo DevOps
Fonte: Amazon Web Services, Inc., 2018

As implementações nesse tipo de ambiente fazem uso de ferramentas de automação com o intuito de dinamizar cada vez mais a infraestrutura e torná-la mais programável, de forma que reflita na melhoria contínua da comunicação e integração entre desenvolvedores e administradores de infraestrutura, transformando o cenário tradicional de isolamento entre essas duas equipes em um ambiente participativo e colaborativo.(COSTA, 2018a)

O objetivo do DevOps é gerar em toda a equipe envolvida na produção de um software, uma cultura que vise o aumento do fluxo de trabalho (maior frequência de deploys) e em paralelo a isso, dar mais robustez no desenvolvimento da aplicação. Esse conceito representa muito mais do que simplesmente o uso de ferramentas de automação, é importante observarmos que trata-se de uma quebra de paradigmas e uma mudança na cultura no negócio com uma nova forma de produção.(SATO, 2014)

A agilidade no processo de deploys citado anteriormente, aluz à uma necessidade de amparo para que essa entrega rápida de fato aconteça, e mais do que isso, a demanda por parte dos clientes ou usuários de determinada aplicação, requer cada vez mais velocidade no uso de determinada funcionalidade. Analogamente à revolução industrial do século XX, a mudança na forma de produção de produtos naturalmente está sendo absorvida pela tecnologia, assim, a concepção desse produto deve acompanhar a demanda externa. Nesse sentido, a adaptabilidade e mudança da arquitetura funcional no desenvolvimento de serviços tecnológicos são necessidades relevantes, a cultura DevOps é uma mudança importante nesse ponto.(IBM, 2018)

Segundo uma pesquisa realizada pelo Gartner Group sobre DevOps, em 2015, somente 29% das organizações pesquisadas tem esse modelo atuante em produção. É

evidenciado ainda que apenas 42% desses, tem a atuação do DevOps em aplicações móveis. De acordo com o mesmo grupo o DevOps evoluiria de uma estratégia de nicho para uma estratégia comum sendo empregada por 25% das organizações do Global 2000[1].(GARTNER, 2018b)

O uso da cultura DevOps deve ser absorvida na necessidade de versionamento contínuo de determinada aplicação, ou seja, a frequente execução de deploys. Nisso, é importante observarmos que a rápida proliferação de software requer atualizações operando na mesma medida ágil e demandada pela competitividade de mercado, visto que a velocidade no atendimento da expectativa dos clientes diferencia a empresa em relação às demais atuantes no mercado.

A principal vantagem no uso de DevOps é a melhora evidente nos processos e automatização das tarefas, otimizando o tempo e reduzindo os ciclos de desenvolvimento. Por se tratar de uma interação entre as equipes de desenvolvedores e operacionais, dizemos que é um sistema bimodal de trabalho.(SATO, 2014)

O monitoramento de métricas e registro de logs é um aspecto relevante. Leva-se em conta que os serviços devem estar disponíveis 24 horas por dias e durante os 7 dias da semana, acarretando em uma massiva análise de dados e logs gerados pelo sistema, portanto, a rotina na observância desses elementos deve ser constante. É possível, inclusive, a criação de alertas que apontem situações e permitam a gerência proativa dos serviços.

Outro benefício fundamental do DevOps é o aumento na comunicação e colaboração que envolve todos os personagens da empresa. É importantíssimo a definição de normas que permitam um maior compartilhamento de informações, ou que permita a proliferação da comunicação, seja qual for o método ou tecnologia, desde que agregue valor. Além disso, a diminuição de ruídos na comunicação e conflitos entre as equipes melhora o ambiente e tende a produzir efeitos positivos ao fim do processo.(GAEA, 2018)

Podemos ainda elencar ganhos em maior estabilidade e melhor desempenho, e tão importante quanto, a redução considerável de custos de trabalho, visto que a diminuição de tempo de produção e menor esforço afeta diretamente o custo financeiro estimado em um projeto.

2.1 Infraestrutura como Código

Falar em infraestrutura com código é exatamente absorver o entendimento do tratar a estrutura de TI como um software, programável. Com isso é possível o uso de práticas que envolvam o controle por versionamento, testes automatizados, entrega contínua, entre

[1] Forbes Global 2000 é uma classificação anual das 2.000 empresas públicas do mundo pela revista Forbes. O ranking é baseado em quatro critérios: vendas, lucro, ativos e valor de mercado. A lista é publicada desde 2003. Fonte: Forbes

diversos outros recursos por meio de scripts específicos.

O uso de práticas anteriores com métodos de gerenciamento de infraestutura foram válidos e deram uma base poderosa para a concepção de novas tecnologias. Atualmente, há uma constante demanda de softwares cada vez mais dinâmicos e um alto índice de deploys, inclusive simultâneos, levando à necessidade de concepção de uma infraestrutura que acompanhe o ritmo de complexidade desses novos sistemas.

Segundo a Hewlett Packard[2], em seu site oficial, a infraestrutura como código elimina a necessidade de criar diversos ambientes de produção de forma manual, isolada e separadamente, e/ou atuailzações de hardware e sofware. Toda essa dinâmica pode ser feita através de scripts contidos no mesmo conjunto de códigos, trazendo velocidade, economia e otimização de tempo. Nesse contexto, a infraestrutura como código traça uma linha tênue entre o código que executa a aplicação e o código que configura um ambiente, tornando um ambiente característico do DevOps.

2.2 Principais Ferramentas e Recursos Tecnológicos Disponíveis

Apesar do conceito DevOps ser recente, a gama de ferramentas que contribuem para implantação dessa cultura já se mostram bastante diversificadas. Dentre as mais comuns podemos apresentar:

2.2.1 Atlas

É uma ferramenta disponibilizada pela Hashicorp, que tem a função de unificar projetos open source para o manejo de aplicações finalizadas no desenvolvimento para a produção em qualquer que seja a infraestrutura. As etapas do Atlas seguem cinco passos (vide figura 2), isso independe da tecnologia utilizada, sejam máquinas virtuais ou contêineres, as etapas se mantêm as mesmas. (HASHIMOTO, 2018)

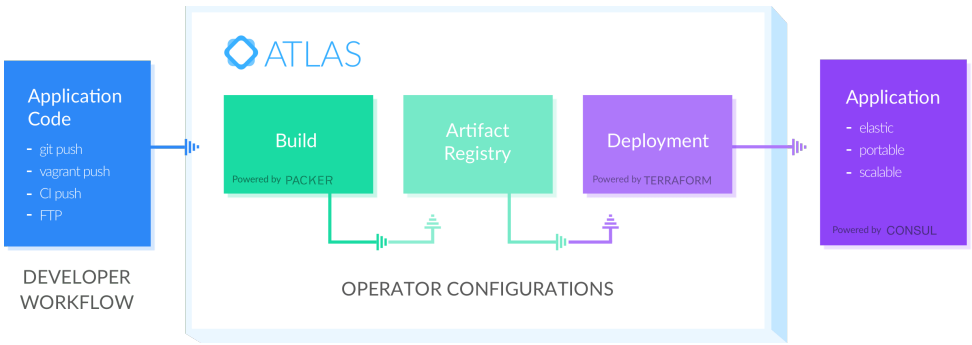

Figura 2 – Etapas do Atlas
Fonte: Hashicorp[3]

[2] Disponível em: https://www.hpe.com/br/pt
[3] Disponível em: https://www.hashicorp.com/blog/atlas-announcement

O Atlas não é um software de caixa preta, ou seja, é possível acesso a serviços que contribuem para tanto, como Vagrant (gerencia ambientes de desenvolvimento), Packer (construção de artefatos), Terraform (implantação de Infraestrutura) e Consul (monitora os serviços em tempo real).

2.2.2 Chef

É um framework destinado à automatização para sistemas e infraestrutura em nuvem. O Chef[4] constrói, entrega e administra fazendo uso de scripts replicáveis.

O Chef tira a carga dos administradores de sistemas que focam no gerenciamento projetado para servidores autônomos, ele permite executar centenas de instâncias de servidor em um tempo imensamente menor se comparado ao uso comum em deploys. Para gerenciar esse tipo de configuração, o Chef, transforma a infraestrutura em código, deixando o processo mais flexível, legível pelos analistas e testável, possibilitando assim a gerência de recursos tanto localmente quanto na nuvem

Em sua topologia o Chef tem três principais componentes: o Servidor Chef, as Estações e o Nós. O maior atrativo dessa ferramenta é o uso de "cookbooks"ou receitas, são ditas configurações plugáveis, que envolvem todas as instalações e parâmetros necessários para atender determinada aplicação no servidor ou máquina. Assim como as receitas convencionais definem uma estrutura sequencial que deve ser seguida a fim de tornar o produto final reflexo de uma ideia original, o Chef mantém um conceito semelhante a isso, onde define-se um estado desejado do sistema, desenvolvendo um código de configuração, então processa-se esse código, une ao processo os dados do nó em questão e garante que o estado concebido seja correspondente ao estado do sistema. (GARZOTTO; MEGALE, 2006)

O Chef pode ser executado em várias plataformas, como Windows, distribuições Linux, FreeBSD, Solaris, AIX, Cisco IO e Nexus. E ainda suporta plataformas em nuvem,como Amazon Web Services (AWS), Google Cloud Platform, OpenStack, IBM Bluemix, HPE Cloud, Microsoft Azure e VMware vRealize.

2.2.3 Docker

O Docker[5] é uma plataforma de automação que implanta as aplicações em espaços isolados chamados de *contêineres*, possibilitando executar as aplicações de forma mais ágil. O objetvo é criar múltiplos ambientes dentro de um mesmo servidor, acessíveis externamente.

[4] Disponível em https://www.chef.io
[5] Disponível em https://www.docker.com/

Segundo Matthias e Kane, no livro Primeiros Passos com Docker, essa ferramenta constrói a arquitetura do software de modo a tornar o armazenamento ainda mais robusto. A essência no Docker é acomodar os serviços ou aplicações em contêineres atômicos ou descartáveis. Em todo o processo de desenvolvimento, um contêiner pode ser excluído e ser restaurado se assim surgir a necessidade, isso torna a dinâmica de entrega e testes extremamente flexível.(MATTHIAS KARL; KANE, 2016)

O Docker é uma plataforma Open Source, que não pode ser confundido com um ambiente tradicional de virtualização. No Docker fazemos uso de recurso isolados que utilizam bibliotecas do kernel em comum, isso porque é presente nessa ferramenta o Linux Containers (LXC).

O mesmo permite empacotar a aplicação ou um ambiente em um contêiner e movê-lo para qualquer outro host que possua o Docker instalado, tornando assim uma ferramenta portável. A grande vantagem disso é que não há necessidade de reconfigurar todo o ambiente novamente, visto que todo ele é movido, reduzindo acentuadamente o tempo de deploy de uma infraestrutura.

A proposta do Docker é exatamente essa. Não há a necessidade de subir várias máquinas virtuais, tão somente precisa-se de uma máquina, e será possível executar várias aplicações sem conflitos entre elas. Todas as dependências, bibliotecas e recursos necessários especificamente a cada software, serão disponibilizados no contêiner. Dessa forma, não é necessário instalar novamente todos os serviços em cada ambiente, diminui-se o uso de recursos e mantém-se as configurações da aplicação isoladas de outros softwares, evitando conflitos. (SCAMPINI, 2018)

Essa plataforma não pode ser confundida com um ambiente virtualizado, visto que em cenários que utilizam máquinas virtuais há a presença de uma camada intermediária de sistema operacional entre o host e as aplicações, no Docker essa é desnecessária pois ele não utiliza kernel (vide figura 3), tornando independente quanto a nível de disco, memória e processamento.(MOUAT, 2016)

A infraestrutura no Docker também é replicável, é possível criar imagens predefinidas e disponibilizá-las em ambientes de desenvolvimento, teste, homologação e produção para aplicações.(MATTHIAS KARL; KANE, 2016)

2.2.3.1 Vantagens:

Podemos elencar alguns ganhos consideráveis na utilização do Docker (SCAMPINI, 2018) :

- Empacotamento de software otimizando o uso das habilidades dos desenvolvedores;
- Empacotamento de aplicação de software com todos os arquivos e dependências

necessárias para determinada aplicação.

- Utilização de artefatos empacotados que possibilitem a passagem pelo teste e produção sem necessidade de recompilação.
- Uso de softwares sem onerar recursos demasiados, visto que o contêiner é apenas um processo que se comunica diretamente com kernel do Linux.

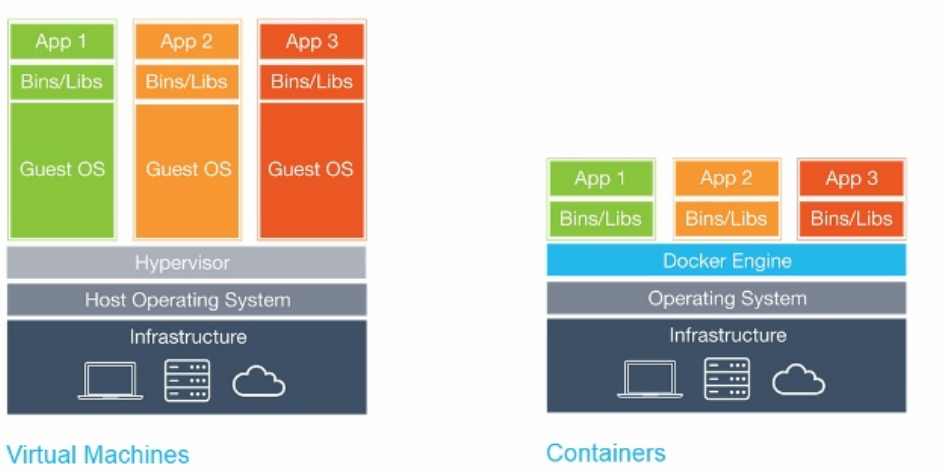

Figura 3 – Comparativo Docker x Máquinas Virtuais
Fonte: Bright Computing[6]

2.2.4 Puppet

É uma ferramenta de código livre para gestão de configurações. A ideia central do Puppet[7] é a administração de diversas máquinas físicas ou virtuais, onde a configuração é centralizada em um único nó e então distribuídas por diversos nós na rede, assim, gerenciando configurações, automatizando instalação de pacotes e facilitando o estabelecimento de normas e auditoria.(WALBERG, 2008)

A ferramenta está disponível em duas versões: Puppet Enterprise[8] (com suporte pago), e a Open Source Puppet[9] (código aberto). É uma ferramenta bem utilizada pela comunidade e por isso existem muitos módulos desenvolvidos, empresas como McAfee e Nasa fazem uso dela.

O Puppet utiliza SSH para a conexão aos hosts, esse é um ponto positivo se olharmos pela ótica que em alguns cenários não é possível a instalação de agentes ou em situações onde o agente consome uma fatia considerável de memória e cpu.

[7] Disponível em https://www.puppet.com/
[8] Disponível em: https://puppet.com/products/puppet-enterprise
[9] Disponível em: https://puppet.com/download-open-source-puppet
[10] Disponível em: https://www.digitalocean.com/

	Ansible	Puppet	Chef
Linguagem do Script	YAML	Custom DSL baseado em Ruby	Ruby
Infraestrutura	Máquina controladora aplica configuração em nós via SSH	Puppet Master sincroniza configuração em nós Puppet	Chef Workstation empurra configuração para o servidor Chef onde os nós Chef serão atualizados
Softwares especiaiados requeridos para nós	Não	Sim	Sim
Fornece ponto de controle centralizado	Não, qualquer computador pode ser controlado	Sim, via Puppet Master	Sim, via servidor Chef
Terminologia de Scripts	Playbook/Roles	Manifests/ Módulos	Receitas/ Cookbooks
Ordem de Execução das Tarefas	Sequencial	Não sequencial	Sequencial

Tabela 1 – Comparativo entre Ferramentas de Gerenciamento de Configurações
Fonte: Digital Ocean[10]

2.2.5 Ansible

O Ansible[11] foi criado em 2012, por Michael DeHann, basicamente essa ferramenta gerencia configurações e orquestra tarefas. Nesse sentido ele implementa módulos para nós sobre SSH. Os módulos são distribuídos nos nós temporariamente que realizam a comunicação com a máquina de controle (assim com em ferramentas concorrentes) por meio de um protocolo JSON[12].

O diferencial do Ansible em relação as demais ferramentas que se propõe ao mesmo objetivo, é que ele atua com uma arquitetura cliente-servidor, sem agente, isso que dizer que os nós não são necessários para a instalação dos daemons para comunicação com a máquina controle. Isso resulta em diminuição de carga na rede.

Os objetivos mais notáveis do Ansible é tornar a experiência do usuário muito mais simples e fácil, e investir massivamente na segurança e confiabilidade utilizando o OpenSSH como condutor de dados. É uma linguagem construída tendo como parâmetro a auditabilidade, ou seja, possibilitar ao analista o acompanhamento de todas as etapas, rotinas e dados circulados dentro de uma arquitetura definida.(COSTA, 2018b)

[11] Disponível em https://www.ansible.com
[12] Disponível em: http://jsonapi.org/

Nessa ferramenta ainda encontramos o uso de um arquivo de inventário, denominado "hosts", que definem quais nós serão gerenciados, que é simplesmente um arquivo de texto que lista os nós individualmente ou agrupados, ou até um arquivo executável que constrói um inventário de hosts. É uma opção de alta confiabilidade e segurança, e isso se fundamenta pelo uso do Secure Shell. Possui ainda fácil usabilidade, no entanto, não deixa a desejar em qualquer aspecto se comparado a soluções concorrentes.

Na forma tradicional de trabalho o Ansible faz o upload do código que deve ser executado nas estações clientes, é então executado, retorna o resultado da execução e após isso é removido dos clientes. Quando usamos a ótica DevOps esse tipo de fluxo é modificado, fazendo uso do protocolo NETCONF[13] (RFC 6241), onde é possível o envio de comandos aos componentes e receber o retorno da aplicação.

2.2.5.1 Componentes

O Ansible é estruturado pela composição dos seguintes elementos:

- Playbooks[14]: arquivos de configuração, implementação e linguagem de orquestração do Ansible.
- Agentless[15]: É descartado o uso de agente nos servidores a serem monitorados. Isso deve-se à utilização de OpenSSH para definir o estado atual do ambiente, adaptando-se se acaso estiver em desconformidade com a configuração no playbook.
- Módulos[16]: São as tarefas executadas de fato. Os módulos são ditos como "plugins de tarefas"e por isso são eles que realizam as atividades pertinentes.
- Inventário[17]: Armazena e controla informações sobre os grupos de hosts.

2.2.5.2 Vantagens

Podemos apontar alguns ganhos consideráveis na utilização do Ansible (JUNIOR; CARDOSO; ZALLA, 2016) :

- Não há a necessidade na instalação de agentes nos servidores a serem gerenciados;
- Gerencia paralela e simultaneamente de forma orquestrada;
- Simples configuração e bem estruturado;
- Desenvolvimento em diversas linguagens.

[13] Disponível em: https://tools.ietf.org/html/rfc6241
[14] Disponível em https://docs.ansible.com/ansible/playbooks.html
[15] Disponível em https://dbruno.ansible.com/ansible/
[16] Disponível em https://docs.ansible.com/ansible/modules.html
[17] Disponível em https://dbruno.ansible.com/ansible/

2.2.6 GitHub

É um dos serviços web de repositórios mais difundidos. Com essa ferramenta é possível hospedar projetos e aplicações, trabalhando com controle de versionamento.

O Github[18] funciona por meio de repositórios, que se dividem em pastas e dentro destas, subpastas que comportam os arquivos de diversas extensões e linguagens. Ainda oferece a possibilidade de contribuir com um código mesmo que não seja membro original do projeto, desde que permitido. Além de permitir o acesso múltiplo por diversos desenvolvedores de um mesma empresa, permitindo que vários programadores trabalhem simultaneamente em uma mesma aplicação ou arquivo.

Essa ferramenta funciona através de comandos "git", que gera ações de upload, request, clone, update, remoção, merge e rollbacks (retorno a versões anteriores do código). O git é estruturado em árvores, e nessa estrutura encontramos os Branchs que são ponteiros que apontam para um determinado commit.

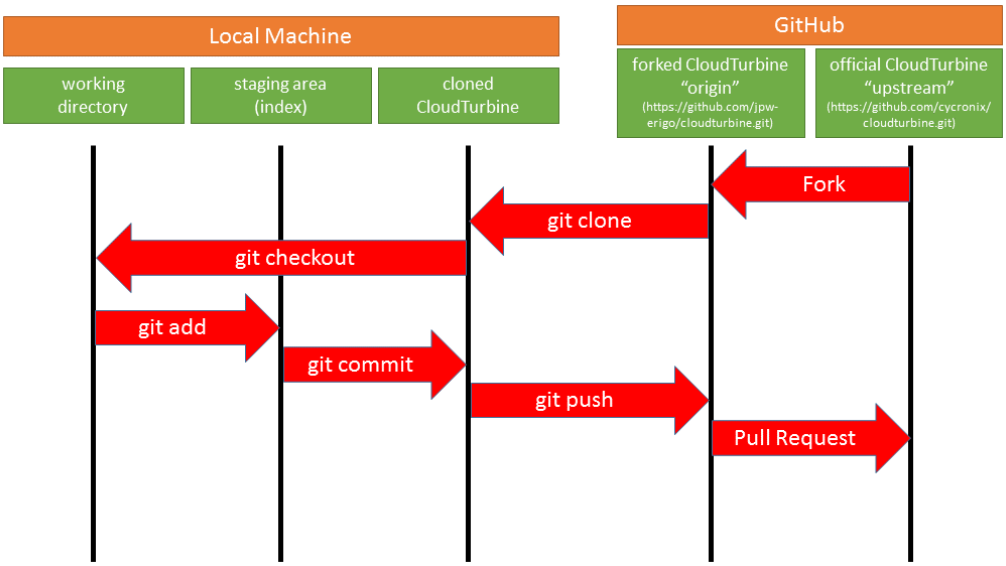

Figura 4 – Fluxo commit e request
Fonte: Cloud Turbine[19]

2.2.7 Jenkins

Essa ferramenta multiplataforma funciona como um servidor destinado a integração contínua que automatiza a execução de tarefas, possui código aberto e permite ao usuário total liberdade em sua operação.

Basicamente o Jenkins[20] atua em um conceito de integração contínua, onde o objetivo é agilizar tarefas que tradicionalmente demandam um tempo de excecução mais

[18] Disponível em https://www.github.com
[19] Disponível em: https://www.cloudturbine.com/using-github-and-git/
[20] Disponível em https://www.jenkins.io

prolongado como compilação do projeto e execução de testes automatizados. Cada interação é analisada por um build automatizado a fim de detectar possíveis erros de integração, isso permite que testes sejam realizados reduzindo problemas de updates e tornando o software mais coeso.(ATALAY, 2018)

O Jenkins pode ser integrado ao Git, SVN, CVS, Maven, entre outros. É importante ressaltar que um ponto forte do Jenkins é a difusão entre a comunidade. Ele ainda possui mais de 1000 plugins disponíveis e utilizado por várias empresas de desenvolvimento de softwares.

Um recurso interessante é o uso de Forks (recurso do GitHub), que permitem a criação de cópias de um determinado projeto e trabalha nesse sem a preocupação de afetar em algum ponto a aplicação original. É possivel ainda, adicionar testes de desempenho e balanceamento na integração contínua, permitindo a análise de risco e a reduzir eventuais quedas de performance no momento em que um novo recurso é adicionado ou a correção de um erro presente.

2.2.7.1 Vantagens

- Builds periódicos;
- Testes automatizados;
- Possibilita análise de código;
- Identificar erros mais cedo;
- Fácil de operar e configurar;
- Comunidade ativa;
- Integra outras ferramentas por meio de plugins nativos.

3 Metodologia

No âmbito desse estudo, é proposta uma arquitetura para implantação de hospedagem de aplicações externas. Assim, o levantamento de informações consistiu em avaliar um conjunto de requisitos essenciais para apoiar o modelo apresentado.

Para a eleição das ferramentas escolhidas foi levado em conta, a sua disponibilização de código open source, a flexibilidade no uso e confiabilidade nos resultados, as suas vantagens e desvantagens, a aceitação junto a comunidade científica e a documentação disponível, seja por meio de livros ou artigos publicados ou por contribuições junto a comunidade web.

O processo de escolha das ferramentas teve como ponto inicial estar em consonância com atuabilidade do mercado de TI e ter continua contribuição da comunidade web. Sendo assim, o primeiro critério a ser considerado foi como a ferramenta daria suporte às diversas linguagens de programações disponíveis. Outro fator relevante está atrelado a maneira como o produto pode ser adquirido, mais diretamente, ao custo necessário para fazer uso da ferramenta. A premissa é que as mesmas deveriam possuir versão open source (gratuita) que dessem suporte às necessidades da arquitetura proposta.

Dessa forma, alguns requisitos foram considerados na escolha, os pontos citados a seguir nortearam o estudo:

- Extensibilidade: tomando como fato as rápidas mudanças no cenário tecnológico, a ferramenta deveria poder dar suporte a diferentes linguagens e integrações;
- Usabilidade: a ferramenta deveria ser de fácil compreensão e fornecer uma boa experiência ao usuário;
- Segurança: é essencial que critérios de segurança sejam inerentes à ferramenta, definição de papéis e usuários.

3.1 Materiais Utilizados

Tomando por base os critérios mencionados no item 3, passou-se a definição das ferramentas, sendo essas:

- VirutalBox: Criação de máquinas virtuais para testes. Serão construídas máquinas virtuais para execução dos hosts clientes e servidores;

DESCRIÇÃO	ESPECIFICAÇÃO
Máquina Física	Mac OS High Sierra 10.13.4 / i5 2.4Ghz 10 GB RAM 1333 MHz DDR3
Oracle VirtualBox	Versão 5.2.8
Docker	Versão
Ansible	Versão
GitHub	Serviço Web: www.github.com

Tabela 2 – Ferramentas utilizadas

- GitHub: Versionamento e repositório. Esse serviço receberá os códigos das aplicações dos desenvolvedores;
- Ansible: Gerenciamento de configuração e orquestração da rotina. Essa ferramenta será responsável por administrar as rotinas dentro de todo o processo de deploy. Será executado em uma máquina virtual com Sistema Operacional Ubuntu Linux 64 bits.
- Jenkins: Automatização da execução de tarefas e testes de implantação. O Jenkins será executado em uma máquina virtual com Sistema Operacional Ubuntu Linux 64 bits.
- Docker: Contêineres para armazenamento das aplicações e testes. O Docker será executado em uma máquina virtual com Sistema Operacional Ubuntu Linux 64 bits.

3.2 Arquitetura e Pipeline

Para desenvolvimento do processo de deploy utilizar-se-á a seguinte arquitetura, que envolve desde a referência de repositório Git até o provisionamento e entrega.

Considerando o modelo apresentado na figura 5, temos como sequenciamento mais detalhado, as etapas a seguir, absorvendo desde o início de uma requisição até a disponibilização da aplicação em servidor de produção.

1. No ponto de partida, é pre-requisito a existência de conteúdo codificado em linguagem de programação, que permita o versionamento do projeto.
2. O desenvolvedor faz a requisição de um novo repositório através de um formulário, que contenha as informações pertinentes, tais como, chave ssh, membros do projeto e nome do mesmo;
3. A requisição é recebida pelo provedor que verifica os dados de solicitação, avalia a disponibilidade de repositório;
4. O provedor valida a disponibilidade e cria um novo repositório no GitHub;

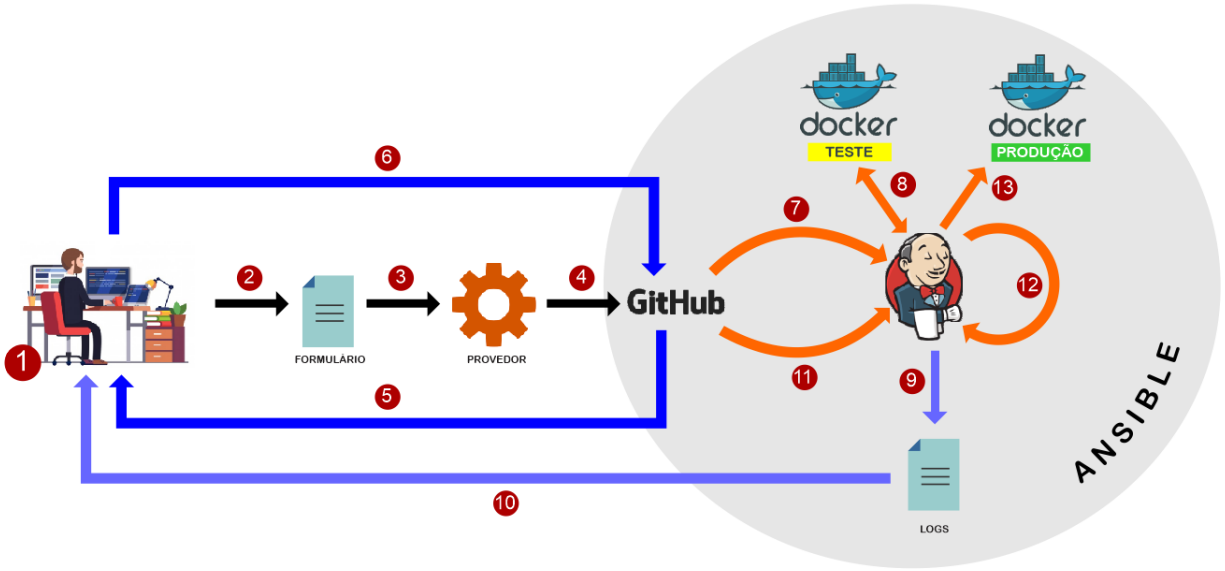

Figura 5 – Representação da Arquitetura Proposta
Fonte: Acervo próprio

5. O link do repositório é retornado ao usuário juntamente com sua chave de autenticação;
6. O desenvolvedor envia os commits e o GitHub gera uma notificação de alteração ao Jenkins;
7. Jenkins recebe a notificação (Git Fetch – Git Merge – Templates Ansible – Dry Run – Commit Check);
8. O Jenkins realiza teste jundo ao contêiner Docker destinado para tanto;
9. Jenkins gera um histórico (Notificação aos Administradores);
10. Administrador analisa os logs de testes;
11. GitHub gera nova notificação ao Jenkins;
12. Jenkins gera novo release e gera nova tag de versão;
13. Estando as alterações aprovadas o Jenkins realiza deploy da aplicação em um contêiner Docker destinado a produção;

Analogamente, um pipeline para essa arquitetura pode ser representado pelo seguinte fluxo: .

.

O processo inicia com a chamada de uma nova requisição do código-fonte do aplicativo do repositório Github em questão. Após isso faz-se a atualização da versão do projeto levando-se em conta o número de compilação, assim tem-se uma identificação digital específica para essa implantação. Após a atualização da versão do projeto, o Jenkins inicia a construção do código fonte utilizando o Maven.

3.3 Métricas

A definição de mecanismos que forneçam e possibilitem a análise estatística de todo o processo é fundamental dentro deste projeto. Nesse sentido, o uso de métricas é necessário para que se tenha uma visão próxima de cada processo de deploy. Serão adotadas nesse projeto as seguintes métricas expostas na tabela ??:

MÉTRICA	DESCRIÇÃO
Tempo de deploy	Critério utilizado para avaliar o tempo gasto para uma atualização de aplicação entrar em produção
Quantidade de linhas de código adicionadas	Critério utilizado para definir o tamanho da atualização enviada.
Relação tempo x custo	Critério utilizado para mensurar o valor monetário da unidade de tempo gasto pela produtividade
Desempenho frente a múltiplos deploys	Medição do desempenho obtido frente à um estresse de deploys simultâneos
Memória utilizada	Quantidade de memória reservada destinada ao processo de deploy

Tabela 3 – Métricas adotadas

4 Resultados

Após análise das ferramentas estudadas e posteriormente eleitas as que melhores se adequaram ao presente estudo, partiu-se para a elaboração e concepção do modelo proposto para entrega de aplicação externa com base em uma visão DevOps.

A figura 5 representa a proposta de modelo de arquitetura para aplicação externa pretendida nesse estudo. O processo que dará início à sequencia de testes e deploy, começará com a requisição do usuário (desenvolvedor) para adquirir um repositório com intuito de armazenar uma determinada aplicação web. Isso quer dizer, que o programador terá acesso a um formulário de requisição (via sistema web) para pretensão de repositório, será retornado ao mesmo o link desse repositório a fim de que o mesmo possa hospedar o seu código. Os códigos e alterações serão enviados ao GitHub. Depois de enviar o código, o Jenkins receberá continuamente os códigos e executará os testes. Sendo os testes realizados com sucesso e tendo retorno positivo, sem erros, o Jenkins realizará o deploy no servidor através do Docker.

5 Conclusão

Este trabalho concentrou-se na apresentação dos conceitos da visão DevOps e analogia com o modelo tradicional de entrega de um software, onde ao decorrer do estudo analisou-se diversas ferramentas disponíveis no mercado e elegeram-se as que mais se adequavam à necessidade da implementação de um modelo de entrega para aplicação externa com o conceito de deploy automatizado.

O deploy automatizado fornece ao desenvolvedor e profissional de TI um ambiente simplificado para entrega de uma aplicação externa, tornando o gestor de tecnologia um provedor de serviço.

O presente modelo tem como foco contribuir para o aprimoramento no processo de entrega de software. Nesse sentido o cenário atual ainda lida com o processo de deploy baseado no modelo tradicional "cascata". Com o estudo realizado nesse projeto é possível otimizar o planejamento do processo da entrega da aplicação para produção utilizando recursos automatizados e gratuitos (open source) e uma visão DevOps. É possível economizar não só tempo como recursos humanos e financeiros que são fundamentais para a conclusão do processo.

Para continuidade deste projeto, objetiva-se a implementação de um cenário funcional utilizando máquinas virtuais, como expõe-se a seguir:

1. Implementar o cenário apresentado neste projeto;
2. Validar o modelo considerando diversas situações;
3. Implementar o modelo visando ambientes críticos e cenários de risco.
4. Implantar a solução afim de gerar um estudo de caso, visando verificar a aceitação da mesma.

Referências

ATALAY, A. *Implantação Contínua via GitLab, Jenkins, Docker e Slack*. [S.l.]: Disponível em: <https://www.infoq.com/br/news/2013/04/ansible1.1>. Acesso em 14 de fevereiro, 2018.

COSTA, B. L. *DevOps, quem usa? como adotar? o que usar?* [S.l.]: Disponível em: <https://www.primeinf.com.br/devops-quem-usa-como-adotar-o-que-usar/>. Acesso em 13 de fevereiro, 2018.

COSTA, M. *Ansible: uma nova opção para gerenciamento de configuração*. [S.l.]: Disponível em: <https://www.infoq.com/br/news/2013/04/ansible1.1>. Acesso em 14 de fevereiro, 2018.

GAEA. *4 Motivos para Usar a Cultura DevOps*. [S.l.]: Disponível em: <https://gaea.com.br/4-motivos-para-usar-a-cultura-devops/#>. Acesso em: 13 de fevereiro, 2018.

GARTNER, G. *Glossário de Ti: DEVOPS*. [S.l.]: Disponível em: <https://www.gartner.com/
it-glossary/devops>. Acesso em 13 de fevereiro, 2018.

GARTNER, G. *Newsroom: Gartner Says By 2016, DevOps Will Evolve From a Niche to a Mainstream Strategy Employed by 25 Percent of Global 2000 Organizations*. [S.l.]: Disponível em: <https://www.gartner.com/newsroom/id/2999017>. Acesso em 13 de fevereiro, 2018.

GARZOTTO, F.; MEGALE, L. Chef: a user centered perspective for cultural heritage enterprise frameworks. In: ACM. *Proceedings of the working conference on Advanced visual interfaces*. [S.l.], 2006. p. 293–301.

HASHIMOTO, M. *Atlas*. [S.l.]: Disponível em: <https://www.hashicorp.com/blog/atlas-announcement>. Acesso em 14 de fevereiro, 2018.

IBM. *O que é DevOps?* [S.l.]: Disponível em: <https://www.ibm.com/developerworks/
community/blogs/rationalbrasil/entry/o_que_devops?lang=en>. Acesso em: 12 de fevereiro, 2018.

JUNIOR, V. et al. Devops: Aproximando a área de desenvolvimento da operacional. 2016.

MATTHIAS KARL; KANE, S. *Primeiros Passos com Docker: Usando Contêineres em Produção*. [S.l.]: Editora Novatec, 2016.

MOUAT, A. *Usando Docker*. [S.l.]: Editora Novatec, 2016.

SATO, D. *DevOps na prática: entrega de software confiável e automatizada*. [S.l.]: Editora Casa do Código, 2014.

SCAMPINI, R. *Docker para desenvolvedores - Que vantagem eu teria com docker?* [S.l.]: Disponível em: <https://medium.com/thdesenvolvedores/docker-para-desenvolvedores-que-vantagem-eu-teria-com-docker-ee8eb77cfe8d> . Acesso em: 20 de fevereiro, 2018.

WALBERG, S. Automate system administration tasks with puppet. *Linux Journal*, Belltown Media, v. 2008, n. 176, p. 5, 2008.

A Scripts de Configuração de Ambiente

A.1 Instalação do Docker

É recomendado a atualização dos do repositório do sistema:

```
$ sudo apt-get update
```

Instalação de pacotes extras:

```
$ sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual
```

Permitindo a instalação a partir de repositórios:

```
$ sudo apt-get install apt-transport-https ca-certificates curl software-properties
```

Adicionando repositório do Docker:

```
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubunt
```

Atualizando o repositório adicionado:

```
$ sudo apt-get update
```

Instalando o Docker CE (Communit Edition):

```
$ sudo apt-get install docker-ce
```

```
CRIAR REDE INTERNA PARA TESTES
docker network create {attachable {subnet=192.168.0.0/24 cluster
```

A.2 Jenkins

```
//plugins.txt

workflow-api:2.26
jira:2.5
```

```
ssh-agent:1.15
dashboard-view:2.9.11
pipeline-model-extensions:1.2.7
git-server:1.7
blueocean-rest-impl:1.4.2
timestamper:1.8.9
blueocean-config:1.4.2
external-monitor-job:1.7
cobertura:1.12
workflow-support:2.18
build-name-setter:1.6.9
gitlab-hook:1.4.2
saferestart:0.3
build-pipeline-plugin:1.5.8
jenkins-design-language:1.4.2
jdk-tool:1.1
active-directory:2.6
resource-disposer:0.8
ansible:1.0
blueocean-git-pipeline:1.4.2
parameterized-trigger:2.35.2
handy-uri-templates-2-api:2.1.6-1.0
blueocean-dashboard:1.4.2
ldap:1.20
subversion:2.10.5
config-file-provider:2.18
gradle:1.28
sse-gateway:1.15
publish-over-ssh:1.19.1
blueocean-pipeline-scm-api:1.4.2
pam-auth:1.3
jquery:1.12.4-0
docker-commons:1.11
docker-workflow:1.15.1
blueocean-rest:1.4.2
apache-httpcomponents-client-4-api:4.5.3-2.1
command-launcher:1.2
antisamy-markup-formatter:1.5
workflow-cps-global-lib:2.9
```

```
artifactdeployer:1.2
webhook-step:1.3
github:1.29.0
workflow-job:2.17
jackson2-api:2.8.11.1
token-macro:2.4
ruby-runtime:0.12
pipeline-stage-tags-metadata:1.2.7
ant:1.8
gitlab-plugin:1.5.5
git-client:2.7.1
gitlab-merge-request-jenkins:2.0.0
windows-slaves:1.3.1
workflow-step-api:2.14
email-ext:2.62
checkstyle:3.50
nodejs:1.2.5
pipeline-stage-view:2.10
branch-api:2.0.18
mailer:1.21
blueocean:1.4.2
momentjs:1.1.1
conditional-buildstep:1.3.6
scm-api:2.2.6
cloudbees-bitbucket-branch-source:2.2.10
blueocean-i18n:1.4.2
blueocean-personalization:1.4.2
script-security:1.43
pipeline-rest-api:2.10
blueocean-events:1.4.2
javadoc:1.4
handlebars:1.1.1
envinject-api:1.5
build-timeout:1.19
embeddable-build-status:1.9
ssh-slaves:1.26
display-url-api:2.2.0
workflow-scm-step:2.6
authentication-tokens:1.3
```

```
jenkins-multijob-plugin:1.29
credentials-binding:1.16
workflow-basic-steps:2.6
mercurial:2.3
blueocean-pipeline-api-impl:1.4.2
github-api:1.90
pipeline-graph-analysis:1.6
bouncycastle-api:2.16.2
jquery-detached:1.2.1
blueocean-autofavorite:1.2.2
copyartifact:1.39.1
analysis-core:1.95
variant:1.1
ssh-credentials:1.13
workflow-cps:2.45
pipeline-model-declarative-agent:1.1.1
workflow-multibranch:2.17
envinject:2.1.5
git:3.8.0
structs:1.14
pipeline-utility-steps:2.0.2
blueocean-core-js:1.4.2
blueocean-github-pipeline:1.4.2
favorite:2.3.1
pipeline-model-api:1.2.7
blueocean-display-url:2.2.0
sonar-quality-gates:1.3.0
rebuild:1.28
matrix-auth:2.2
blueocean-pipeline-editor:1.4.2
matrix-project:1.12
toolenv:1.1
publish-over:0.21
pipeline-github-lib:1.0
workflow-aggregator:2.5
blueocean-jwt:1.4.2
ws-cleanup:0.34
credentials:2.1.16
blueocean-web:1.4.2
```

```
pipeline-build-step:2.7
pipeline-milestone-step:1.3.1
plain-credentials:1.4
maven-plugin:3.1.2
blueocean-bitbucket-pipeline:1.4.2
pubsub-light:1.12
blueocean-jira:1.4.2
durable-task:1.22
jsch:0.1.54.2
htmlpublisher:1.15
pipeline-stage-step:2.3
mapdb-api:1.0.9.0
ssh:2.5
ace-editor:1.1
github-branch-source:2.3.3
cloudbees-folder:6.4
role-strategy:2.7.0
run-condition:1.0
junit:1.24
pipeline-model-definition:1.2.7
workflow-durable-task-step:2.19
pipeline-input-step:2.8
built-on-column:1.1
blueocean-commons:1.4.2
thinBackup:1.9
kubernetes:1.5


//Dockerfile

FROM jenkins/jenkins:2.112
USER root
RUN apt-get update && apt-get install -y make git openjdk-8-jdk
RUN mkdir /srv/backup && chown jenkins:jenkins /srv/backup
USER jenkins
RUN echo 2.112 > /usr/share/jenkins/ref/jenkins.install.UpgradeWizard.state
RUN echo 2.112 > /usr/share/jenkins/ref/jenkins.install.InstallUtil.lastExecVersion
COPY plugins.txt /usr/share/jenkins/ref/plugins.txt
RUN /usr/local/bin/install-plugins.sh < /usr/share/jenkins/ref/plugins.txt
```

```
ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64
COPY --chown=jenkins:jenkins config_jenkins /var/jenkins_home
```

Executar conteiner:

```
\\setup-and-run.sh

#!/usr/bin/env bash

dockerhub_user=matheusjose

jenkins_port=8080
image_name=missao-devops-jenkins
image_version=2.2.2
container_name=md-jenkins

docker pull jenkins:2.112

docker stop ${container_name}

docker build --no-cache -t ${dockerhub_user}/${image_name}:${image_version} .

if [ ! -d m2deps ]; then
mkdir m2deps
fi
if [ -d jobs ]; then
rm -rf jobs
fi
if [ ! -d jobs ]; then
mkdir jobs
fi

docker run -p ${jenkins_port}:8080 \
-v `pwd`/jobs:/var/jenkins_home/jobs/ \
-v `pwd`/m2deps:/var/jenkins_home/.m2/repository/ \
--rm --name ${container_name} \
${dockerhub_user}/${image_name}:${image_version}
```

A.3 Ansible

Instalção do Ansible:

```
$ apt-get install software-properties-common
$ apt-add-repository ppa:ansible/ansible
$ apt-get update
$ apt-get install ansible
```

Definição de inventário de hosts a serem gerenciados pelo Ansible:

```
cat /etc/ansible/hosts

[webserver]
app-01.webserver.com.br
app-02.webserver.com.br

[database]
db-01.databaseserver.com.br

[ansible]
ansiblecontrol-01.server.com.br ansible_connection=local
```

Playbook de automatização do Jenkins

```
---
- name: Deploy Jenkins CI
hosts: jenkins_server
remote_user: jenkins
become: yes

roles:
- geerlingguy.repo-epel
- geerlingguy.jenkins
- geerlingguy.git
- tecris.maven
- geerlingguy.ansible

roles:
- jenkins-keys-config
```