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

A demanda cada vez mais latente pela disponibilização rápida de serviços e aplicações, traz uma alta cobrança sobre a equipes de desenvolvimento e operacional quanto a quase imediata disponibilização de determinado software ao usuário final, aliado a isso a qualidade da aplicação também tem um lugar fundamental no processo. Nesse sentido, um serviço que realiza gestão da construção de uma aplicação, torna a morosidade do processo de deploy tradicional, suprida e reduz os possíveis conflitos entre as equipes de desenvolvimento e operações, a isso conferimos o termo DevOps.

Aliado a isso, o provisionamento da Infraestrutura como código e o uso de contêineres fazem com que todo o ambiente se torne replicável e escalável, assim a agilidade em todo o processo de deploy torne-se evidente e de simples monitoramento.

Palavras-chaves: Automação, DevOps, Modelo Ágil, Integração Contínua.

Lista de ilustrações

Lista de abreviaturas e siglas

DevOps - Development and Operational (Desenvolvimento e Operacional).

AWS - Amazon Web Services

SSH - Secure Shell

NETCONF - Network Configuration

CI - Continous Integracion

Sumário

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 uma arquitetura 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.

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, isso resulta na entrega ágil do produto trabalhado.

A estrutura desse trabalho está organizado da seguinte forma: Na seção 2 a justificativa. Na seção 3 a problemática que envolve o tema. Na seção 4 os objetivos geral e específicos. Na seção 5 será apresentado o referencial teórico utilizado para embasar o tema e a ideia proposta, de acordo com todas as ferramentas estudadas. Na seção 6 serão apresentados os métodos utilizados para compor o estudo, assim como as ferramentas utilizadas durante todo o processo. 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).

2 Justificativa

A visão DevOps traz consigo o ideal de produtividade pela automatização na entrega de um serviço ou aplicação. Aliado a essa proposta, a Integração Contínua (CI) dá suporte para que todo o processo de construção ou build de um código-fonte possa ser realizado de forma automatizada, desde a implantação do ambiente aos testes unitários e de integração, tal prática retira tanto da equipe de desenvolvimento quanto de infra o peso e tempo gasto no ato de disponibilizar essa aplicação para produção.

Fazer uso dessa abordagem possibilita aos times mencionados um retorno rápido acerca das alterações que estão continuamente sendo realizadas no código de um projeto, além de viabilizar uma forma mais barata de resolver problemas relacionados a falhas, quando identificadas.

Falamos então, não somente da mudança do processo em si, mas, mais pontualmente uma mudança cultural. A aplicação de uma visão DevOps traz melhorias frente a compilação do código, testes automatizados, empacotamento, criação de ambientes para teste e produção, configuração da infraestrutura, migração de dados, monitoramento, auditoria, segurança, deploy, entre outros. Algumas empresas que fizeram uso das práticas DevOps passaram a ter feedbacks significativos quanto a adaptação das mudanças do mercado, realizando diversos *deploys* por dia de forma segura.(SATO, 2014)

Em se tratando do cenário universitário, é fundamental que a instituição corra na mesma velocidade em que as novas tecnologias surgem, em se tratando de cursos de Tecnologia da Informação isso se torna latente. Novas e boas práticas devem ser experimentadas pelos alunos, a Integração Contínua pode ser aplicada largamente para fins de estudo e experimentos, além de entenderem a vantagem do uso da automatização, podem investir um tempo maior para o desenvolvimento da aplicação.

3 Problema

A abordagem tradicional torna o processo de deploy lento, estressante e caro, afinal o tempo tem valor. Uma das maiores problemáticas dessa forma de disponibilização é a dificuldade em permitir uma alteração de código e imediatamente depois a sua implementação, isso porque o tempo para realizar essa atualização não permite essa agilidade, a prática adotada geralmente reúne uma série de modificações ou novas funcionalidades e o deploy é feito ao final de um curso, seja semanal, quinzenal ou qualquer que seja a métrica.

Esse problema pode também ser conhecido como **"A Última Milha"**, ocorre na fase final do processo de desenvolvimento, após a validação dos requisitos e antes da implantação em produção. É exatamente nesse ponto onde são realizados os testes e homologação que o processo é notoriamente dispendioso, imagine gerir dezenas ou centenas de alterações no código onde a cobrança pela disponibilidade delas é imediata. A questão da última milha é mais notória quando adotamos uma visão macro do processo, passamos a perceber quanto tempo de se perde com etapas que podem ser automatizadas, e aliado a isso quanto recurso humano-financeiro poderia estar sendo aplicado em outra área. (SATO, 2014)

4 Objetivos

4.1 Objetivo Geral

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

4.2 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 para o ambiente proposto.

5 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)

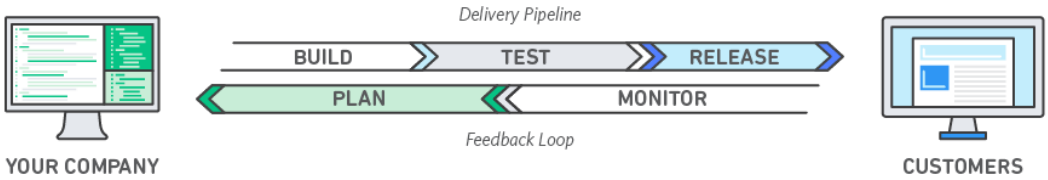

Figura 1 – Ciclo DevOps
Fonte: Amazon Web Services, Inc., 2018

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.

5.1 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.

5.2 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:

5.2.1 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)

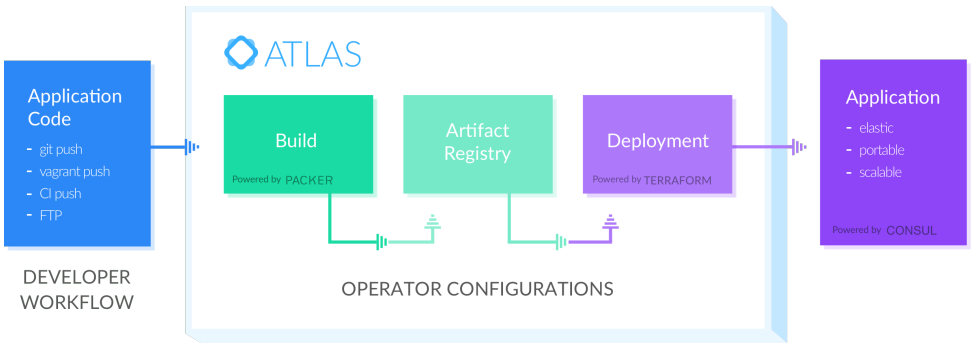

Figura 2 – Etapas do Atlas
Fonte: Hashicorp[3]

[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).

5.2.2 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.

5.2.3 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).

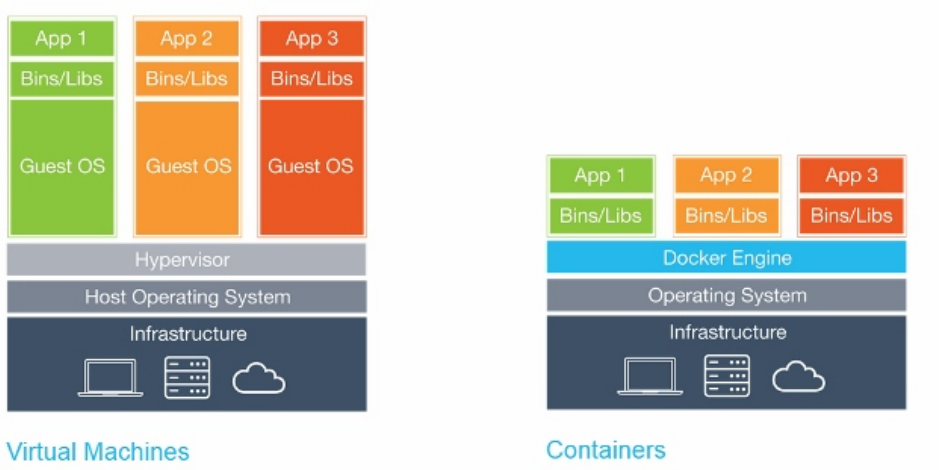

Figura 3 – Comparativo Docker x Máquinas Virtuais
Fonte: Bright Computing[6]

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)

5.2.3.1 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.

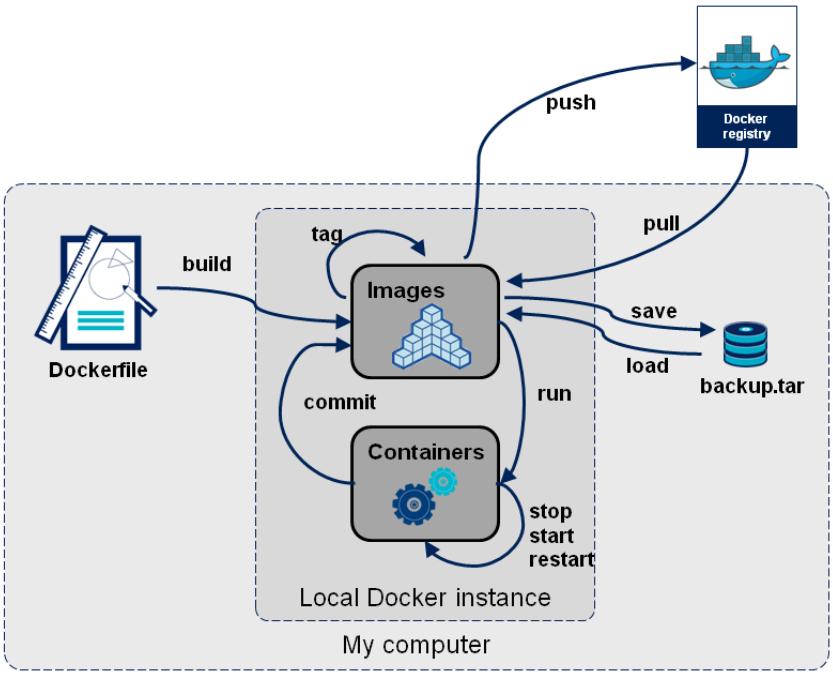

Figura 4 – Fluxo de disponibilização de Imagem Docker
Fonte: OCTO Technology[7]

5.2.4 Puppet

É uma ferramenta de código livre para gestão de configurações. A ideia central do Puppet[8] 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[9] (com suporte pago), e a Open Source Puppet[10] (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.

	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[11]

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

5.2.5 Ansible

O Ansible[12] 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[13].

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)

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[14] (RFC 6241), onde é possível o envio de comandos aos componentes e receber o retorno da aplicação.

5.2.5.1 Componentes

O Ansible é estruturado pela composição dos seguintes elementos:

- Playbooks[15]: arquivos de configuração, implementação e linguagem de orquestração do Ansible.
- Agentless[16]: É descartado o uso de agente nos servidores a serem monitorados.

[12] Disponível em https://www.ansible.com
[13] Disponível em: http://jsonapi.org/
[14] Disponível em: https://tools.ietf.org/html/rfc6241
[15] Disponível em https://docs.ansible.com/ansible/playbooks.html
[16] Disponível em https://dbruno.ansible.com/ansible/

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[17]: 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[18]: Armazena e controla informações sobre os grupos de hosts.

5.2.5.2 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.

5.2.6 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[19] 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.

[17] Disponível em https://docs.ansible.com/ansible/modules.html
[18] Disponível em https://dbruno.ansible.com/ansible/
[19] Disponível em https://www.github.com
[20] Disponível em: https://www.cloudturbine.com/using-github-and-git/

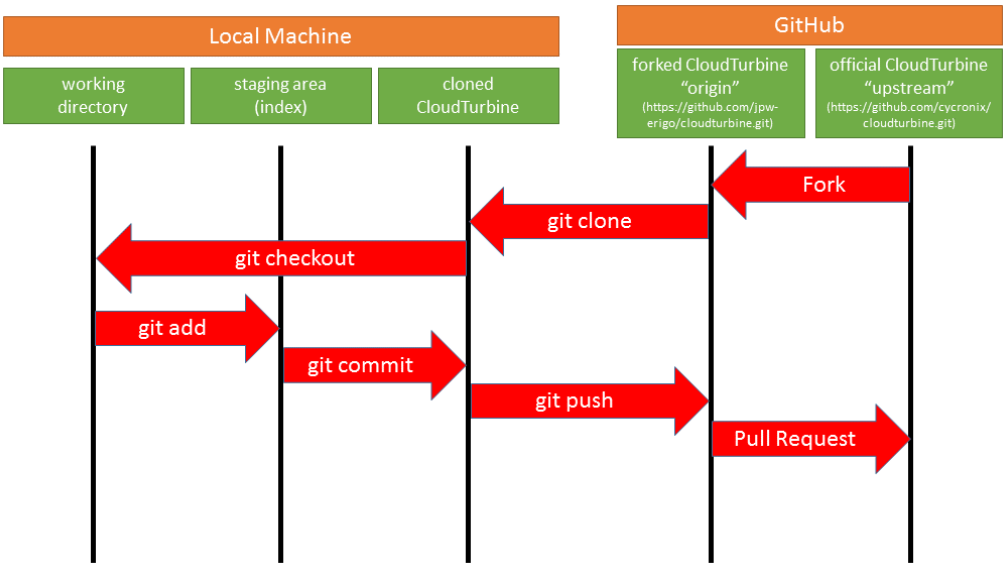

Figura 5 – Fluxo commit e request
Fonte: Cloud Turbine[20]

5.2.7 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[21] atua em um conceito de integração contínua, onde o objetivo é agilizar tarefas que tradicionalmente demandam um tempo de excecução mais 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.

[21] Disponível em https://www.jenkins.io

5.2.7.1 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.

5.2.8 Sonar

O SonarQube é uma plataforma de análise da qualidade do código, com ele é possível avaliar por meio de dados estatísticos o desempenho de determinada aplicação quanto aos testes realizados utilizando diversas métricas.

A qualidade do código é verificada através de alguns eixos, sendo esses, a arquitetura, duplicações de código, complexidade, regras de codificação, comentários, testes unitários e erros em potencial.

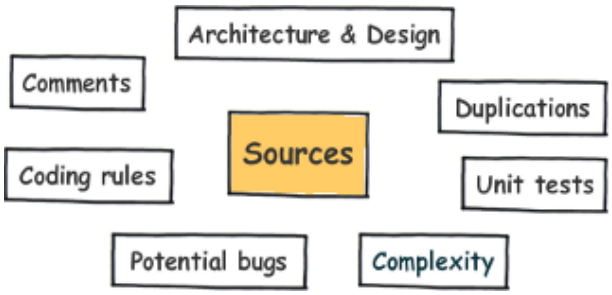

Figura 6 – 7 Eixos de Análise do Sonar
Fonte: Blog Gabriel Amorim[22]

O Sonar tem compatibilidade com diversas linguagens, e plugins para vários serviços como Jenkins, por exemplo. Assim, é possível através da execução de uma determinada tarefa, definir métricas de análise, monitorar todo o processo e como retorno disponibilizar dados estatísticos referente a cada teste executado.

Vantagens:

- Fácil acompanhamento da evolução do código;

[22] Disponível em: http://blog.gabrielamorim.com/analisando-a-qualidade-do-codigo-com-o-sonar/

- Representação dos dados através de gráficos;
- Uso simples da ferramenta.

5.2.9 Kubernetes

O Kubernetes é um sistema destinado à orquestração de contêineres. É perfeitamente associável a ferramentas como Docker, Rocket, Jenkins, Chef, Puppet e Ansible. O seu objetivo é retirar o peso de execução de determinado contêiner em uma única instância do serviço, para isso o Kubernetes cria nós ou slaves diferentes e distribui a execução da tarefa entre esses nós. A figura 7 demonstra a arquitetura comum do Kubernetes.

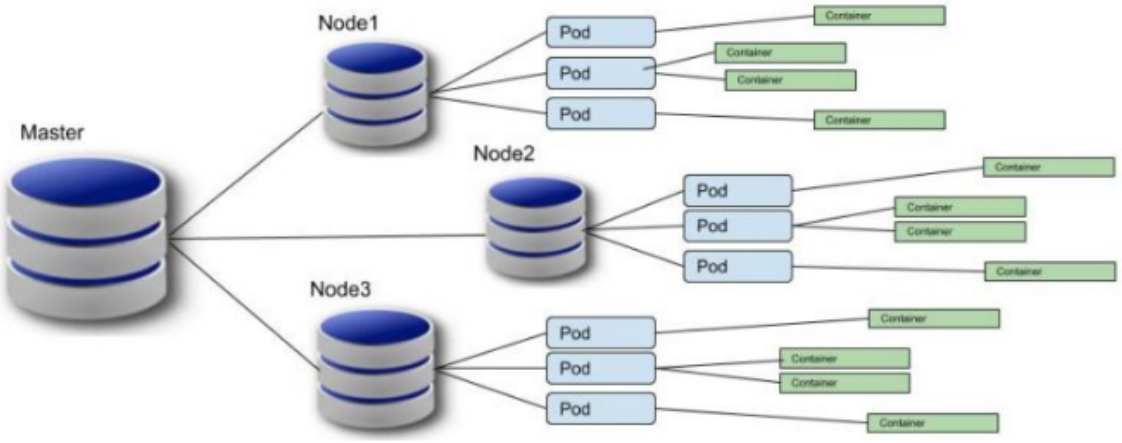

Figura 7 – Arquitetura do Kubernetes
Fonte: King Host[23]

Instituições como *Google*[24] revelaram que utilizam contêineres em serviços como *Gmail* e *Google Docs*, nesses são mais de 2 bilhões de implantações por semana, essa dinâmica é gerida pelo Borg, e essa mesma ferramenta foi precursora do Kubernetes. Em 2015 foi doada para a *"Native Computing Foundation"* e *"Linux Foundation"*, sendo a partir disso um projeto open source.(TRUCCO, 2018)

A sua arquitetura pode ser definida como:

- **Master:** é a instância central que provê uma determinada aplicação com configurações específicas.
- **Cluster:** conjunto dos componentes. Na figura 7 percebemos a criação de um cluster.
- **Node:** Uma máquina ativa gerenciada pelo master, pode também ser mencionada como minions ou slaves.

[23] Disponível em: https://king.host/blog/2018/05/introducao-ao-kubernetes/
[24] Disponível em: www.google.com.br

- **Pods:** menores unidades implantadas que podem ser criadas, escaladas e manuseadas. É uma coleção lógica de contêineres que pertencem a uma aplicação.

O principal motivo que atrai desenvolvedores para o uso do Kubernetes é viabilizar aos clusters abranger alocações em clouds públicas, privadas ou híbridas, e mais especificamente ainda, quando essa arquitetura envolve uma gama elevada de contêineres, o que exige que a ferramenta seja obrigatoriamente auto-escalável e proporcione alta disponibilidade.

5.2.9.1 MiniKube

O manuseio do Kubernetes é mais complexo do que seu concorrente direto *Docker Swarm*[25], demanda mais conhecimento de redes e sistemas distribuídos do arquiteto projetista do ambiente.

Pensando nisso a comunidade criou o Minikube[26], que basicamente é uma Toolkit que reúne uma série de soluções que facilitam muito a implantação do Kubernetes, isso tornou a aplicabilidade mais abrangente, sendo possível o uso em plataformas Windows, Linux e Mac. Nesse caso, o Minikube simula um nó com tudo o que é preciso já instalado e configurado.

Vantagens:

- Fácil manipulação de contêineres;
- Escalonamento da aplicação;
- Volumes persistentes;
- Auto regeneração;
- Abstração de complexidade.

Desvantagens:

- Pouca documentação disponível para iniciantes;
- Curva de aprendizado longa;

[25] Disponível em: https://docs.docker.com/engine/swarm/
[26] Disponível em: https://kubernetes.io/docs/setup/minikube/

6 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 que dessem suporte às necessidades da arquitetura proposta.

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

- 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.

6.1 Materiais Utilizados

Tomando por base os critérios mencionados no item 6, passou-se a definição das ferramentas, sendo essas:

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 17.12.0-ce
GitHub	Serviço Web: www.github.com
SonarQube	Plugin Jenkins Versão 2.8.1
Jenkins	Versão 2.11.2
Kubernetes	Plugin Jenkins - Minikube Versão 1.11

Tabela 2 – Ferramentas utilizadas

- VirutalBox: Criação de máquinas virtuais para testes. Serão construídas máquinas virtuais para execução dos hosts clientes e servidores;
- GitHub: Versionamento e repositório. Esse serviço receberá os códigos das aplicações dos desenvolvedores;
- 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 serviços. O Docker será executado em uma máquina virtual com Sistema Operacional Ubuntu Linux 64 bits.
- Kubernetes: Recurso utilizado para criar nós ou slaves, destinados a dividir o peso de compilação, evitando a sobrecarga no nó master do Jenkins, utilizar-se-á versão Minikube.
- Sonar: Plugin de análise estatística do build, com ele será possível coletar resultado das métricas definidas para resultados.

6.2 Arquitetura

Para o ritmo do processo de deploy será considerada a arquitetura representada na figura 8, que envolve desde a requisição do ambiente por meio de formulário até o provisionamento e entrega. Toda a arquitetura é gerida e orquestrada pelo Ansible desde uma nova mudança no repositório do GitHub até o envio para produção.

Considerando o modelo apresentado na figura 8, pontuaremos cada etapa a fim de detalhar o fluxo da disponibilização ao usuário final. O simbolo de infinito encontrado

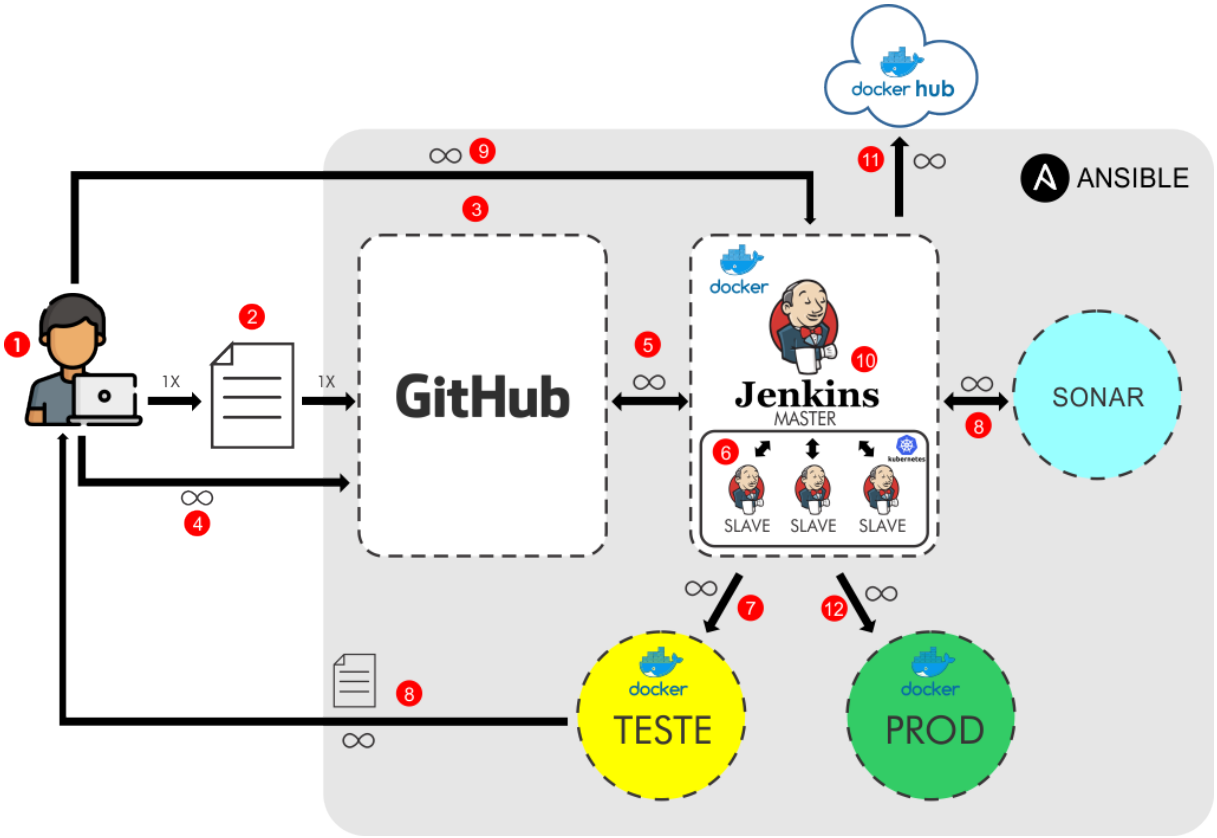

Figura 8 – Representação da Arquitetura Proposta
Fonte: Acervo próprio

no na mesma figura, indica que o fluxo nesse ponto é contínuo e se repete a cada nova interação.

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 ambiente através de formulário web, que contenha as informações pertinentes, tais como link do repositório, dependências do projeto e informações de acesso à aplicação.
3. A requisição é recebida pelo provedor que verifica os dados de solicitação, avalia a disponibilidade de repositório;
4. O desenvolvedor envia os commits e o GitHub gera uma notificação de alteração ao Jenkins;
5. Jenkins recebe a notificação (Git Fetch – Git Merge – Dry Run – Commit Check);
6. O Jenkins cria instâncias diferentes para construir o projeto, para isso, faz uso do Kubernetes;
7. O Jenkins incia o build e realiza testes automatizados;

8. O gestor é notificado quanto aos logs e status do build;
9. O gestor homologa ou não o build;
10. Jenkins gera novo release e nova tag de versão;
11. O Ansible informa ao Docker para construir uma nova imagem da aplicação e a envia para um repositório no Docker Hub[1]
12. Estando as alterações aprovadas o Jenkins através do Ansible realiza deploy da aplicação em um contêiner Docker destinado a produção;

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.

6.3 Descrição do Serviço de Integração Contínua

A fim de criar uma plataforma que possibilitasse a construção e configuração de um ambiente replicável, auto escalável e ágil, foi utilizado o conceito da infraestrutura como código, ou *"as-a-code"*. Nesse sentido, vislumbrou-se um serviço que reunisse as ferramentas Docker, Jenkins e Kubernetes sob orquestração do Ansible para realizar a execução do processo de deploy bem como o provisionamento do ambiente com a instalação dos serviços, dependências e configurações com a chamada do arquivo Dockerfile que faz a instalação das dependências para o ambiente Jenkins, bem como as chaves de acesso, instalação de plugins, busca por jobs, e criação dos devidos contêineres Docker.

O ponto de partida é a criação do *Formulário de Solicitação de Ambiente de Desenvolvimento*, no qual o usuário sinalizará a pretensão de alocação de sua aplicação em servidor web local. Os dados informados nesse formulário gerarão um arquivo *.json* que será lido pela API do Jenkins a fim de se criar um novo Job para esse repositório, esse ponto será mais detalhado na subseção 3.3.1.

Os arquivos fundamentais desse projeto são primeiramente o playbook (cd.yml) do Ansible, ele é responsável por fazer a instalação do Docker e baixar a imagem Jenkins, e então iniciar o processo de configuração do ambiente chamando o Dockerfile mencionado

[1] Disponível em: https://hub.docker.com/

anteriormente. É necessário criar uma conta no Docker Hub[2], esse será o repositório onde as imagens Docker serão exportadas e servirá como controle de versionamento da imagens[3].

A imagem do Jenkins foi construída com o objetivo da mesma ser replicável e auto-escalável, isso quer dizer que a mesma possui configurações genéricas e, portanto, aplicável em qualquer tarefa a ser executada, sendo os *Playbooks* Ansible de cada serviço responsável por suprir as dependências de cada projeto não disponíveis no Jenkins.

O arquivo Dockerfile traz o script utilizado para construir as imagens Docker e mapear os volumes externos utilizados pelo Jenkins, como arquivos de configuração e base de dados, além disso esse arquivo inicia os contêineres dos serviços com o comando "run". O arquivo *plugins.txt*[4] mapeado para um desses volumes traz uma lista de plugins que serão instalados em tempo de execução no build da imagem, assim o Ansible somente instalará plugins mencionados no formulário e que não conste nessa lista, nesse caso é realizado um novo build da imagem para atualizar a lista de plugins e/ou demais configurações.

Na raiz do projeto estão os arquivos de configuração do Jenkins. Nesses, estarão os parâmetros necessários para iniciar o serviço do Jenkins sem a necessidade de reconfigurar toda a ferramenta ao reinicio de cada instância. Essa forma de configuração torna o serviço reaproveitável, ou seja, tanto é possível criar um padrão inicial definido para demais projetos do Jenkins, quanto utilizá-lo para execução dos mais diversos tipos de jobs, desde o mais básico ao mais complexo, alocando somente o recurso necessário para cada iteração, dessa forma não se desperdiça recurso computacional para os builds mais simples e utilizam-se instâncias diferentes do Jenkins através de *slaves* orquestrados pelo Kubernetes para construções que demandam mais poder computacional.

6.3.1 Pipeline do Build

Para início do processo de build do código, é necessário antes de tudo que esteja disponível um repositório de versionamento de códigos, definido aqui com a plataforma do GitHub. Essa sinalização será concebida por meio de um *Formulário de Solicitação de Ambiente de Desenvolvimento*. Esse formulário será disponibilizado por um endereço web, onde é premissa que o usuário já possua um repositório para o projeto no GitHub, e preencha os demais campos solicitados do formulário. É requerido desde o nome do responsável pelo projeto até as dependências necessária para execução do mesmo. Essa busca pelos Jobs são feitas pela API do Jenkins semelhantemente ao REST.

Nesse sentido a API pode ser usada para:

[2] hub.docker.com

[3] Nesse caso o repositório fará versionamento da imagem do serviço, não do código-fonte, esse será feito diretamente no GitHub

[4] vide Apêndice A

- Recuperar informações de Jenkins para consumo programático;
- Desencadear uma nova compilação;
- Criar ou copiar Jobs.

Figura 9 – Formulário Web para Solicitação de Provisionamento de Ambiente

O Pipeline de um projeto baseia-se num fluxo iniciado na alteração do estado do código-fonte em um repositório até a disponibilização do software em produção. Esse processo tem por natureza um fluxo contínuo da cadeia de comandos da Integração Contínua. Iniciando, o Jenkins é responsável por monitorar o repositório GitHub a fim de verificar alteração no estado do código do projeto utilizando recurso de *Push Notification*

já próprio da ferramenta, ou seja, se houveram *commits* no mesmo, o Jenkins atualiza a versão do projeto de acordo com o número de compilação, dessa forma garantisse uma assinatura única da construção durante o processo de implantação. Após atualizar a versão do código partiremos para a construção do mesmo.

Tendo sucesso na compilação, partiremos para a construção do projeto. Após o Jenkins identificar uma alteração no código, o mesmo clona o projeto do repositório, e escalona diferentes estâncias desse mesmo serviço, fazendo o papel de *slaves* e o *master* faz o papel de orquestrador, esse processo retira o peso e custo computacional, se o projeto fosse compilado somente em uma instância master do Jenkins, assim o build é distribuído entre os *slaves* e ao final do processo realiza o *merge* do projeto, esse processo é gerenciado pelo *Kubernetes*, utilizando sua versão *Minikube*.

Parte-se então para os testes da aplicação, nesse ponto o fluxo funciona de forma que testes automatizados sejam aplicados e na incidência de erros o Jenkins dispara uma notificação via email para o usuário, ou gestor da equipe de desenvolvimento, informando a causa da falha e o respectivo log, caso contrário se os testes lograrem sucesso o Ansible recebe o build do Jenkins e realiza o depósito em um ambiente de homologação com Docker. Simultaneamente a isso o gestor do projeto é notificado sobre o sucesso da construção e solicita que o mesmo acesse esse ambiente para a devida homologação. Sendo aprovada as alterações realizadas, o gestor libera a nova versão para ambiente de produção, então o Jenkins recebe a confirmação e sinaliza ao Ansible que solicitará ao Docker a construção de uma nova imagem da aplicação e exportará a mesma para o Docker Hub onde ficará armazena em repositório na nuvem (é também possível fazer essa guarda e versinamento em repositório local) e então é realizado o deploy da imagem em ambiente de produção e iniciado um contêiner do mesma, a partir dai o software já está disponível para acesso pelo usuário final, caso ocorra erro no deploy o gestor é notificado. Esse fluxo pode ser mais claramente entendido na representação da figura 10.

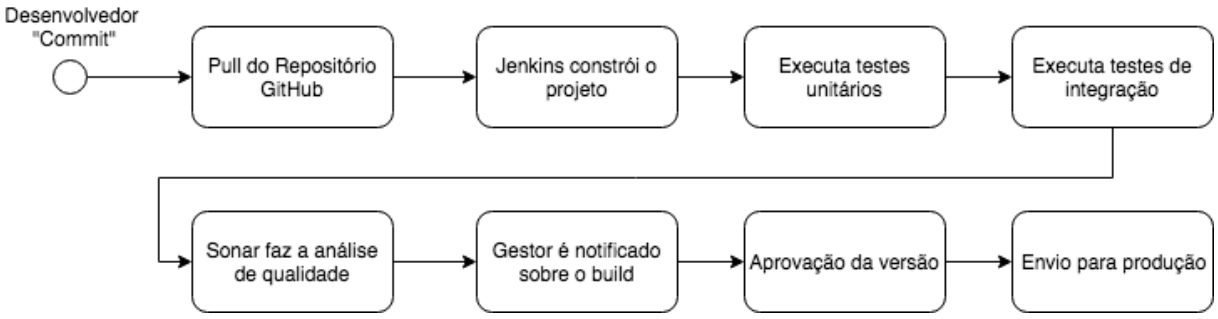

Figura 10 – Pipeline do Build

Durante todo o processo o SonarQube realiza o monitoramento e coleta dos resultados obtidos em todas as etapas do deploy, ao fim do processo é possível verificar nesse ambiente as estatísticas, tomando por base as métricas definidas. Isso dará uma visão mais próxima no que tange o desempenho e qualidade do deploy da aplicação.

6.4 Pseudo-códigos dos Algoritmos Implementados

Afim de organização os playbooks foram agrupados por categorias, sendo o playbook cd.yml o responsável pela chamada dos demais. Cada Playbook é constituído por diversas funções que identificam o tipo da dependência ou serviço, sua versão, se existe repositório específico ou se o Ansible fará a gestão deste, e a definição da tag que identifica a qual serviço essa função pertence.

O pseudo-código 1 faz menção a execução do playbook inicial do Ansible, nele será solicitado a execução de cada um dos demais playbooks que dizem respeito aos serviços necessário para o ambiente:

Algoritmo 1: PROVISIONAMENTO DE AMBIENTE - Funcionalidade: Instalação de Serviços e Dependências (Playbook Ansible)

```
Entrada: Playbook (cd.yml)
Saída: Chamada por cada playbook
início
    Var
        host: server_name          /* Define local de provisionamento */
        user: user                 /* Define usuário remoto */
        become: boolean            /* Define escalonamento de privilégios */
    faça
        execute o playbook java
        execute o playbook docker
        execute o playbook registry
        execute o playbook jenkins
    fim
fim
```

O Pseudo-código 2 trata da execução de cada playbook de serviço, onde verifica-se cada função que trata as dependências exigidas para que esse serviço possa funcionar:

Algoritmo 2: PROVISIONAMENTO DE AMBIENTE – Funcionalidade: Execução dos Playbooks de cada serviço

```
Entrada: Playbook de cada serviço
Saída: Instalação dos serviços utilizando configurações genéricas
início
    para cada função faça
        defina o nome da função          /* Informa o nome da função */
        defina o nome do serviço         /* Informa qual serviço deve ser baixado */
        se possuir repositório específico faça  /* Condição para baixar serviço de repositório específico */
            repositório = url
        fim
        define a tag que identifica a função  /* Informa a qual serviço pertence a função*/
    fim
    execute todas as funções
fim
```

O Pseudo-código 3 busca por Jobs em um diretório mapeado pelo contêiner Docker de cada serviço, nesse diretório serão armazenados os arquivos .json gerados com o preenchimento do formulário de solicitação de ambiente de desenvolvimento:

Algoritmo 3: JOBS – Funcionalidade: Busca por Job

```
Entrada: Arquivo Json
Saída: Job
inicio
    var
        diretorio_de_jobs: string        /* Define diretório (volume) onde os arquivos json serão armazenados
        jobs: lista                      /* Lista de jobs (em formato json)
    se jobs não estiver vazio faça
        para cada json faça                  /* Popula jobs no Jenkins
            leia arquivo json            /* Verifica cada Json no repositório
            novoJob = adicioneBuild(json)    /* Adiciona Job ao Jenkins, ainda não implementado
        fim
    fim
fim
```

O Pseudo-código 4 trata da execução dos Jobs no Jenkins, onde passa pela busca de jobs no diretório, baixa o código do repositório git informado no formulário, gera uma nova versão do código e se autorizado pelo gestor, executa o contêiner:

Algoritmo 4: ANSIBLE – JENKINS – DOCKER – Funcionalidade: Build de Job

```
Entrada: Playbook
Saída: Criação do Job e execução do contêiner
inicio
    para cada novo Job faça
        Executa Dockerfile                  /* Para provisão das dependências do projeto
        Defina diretório de alocação do Job /* Informa o diretório onde os jobs estão armazenados
        Baixe o código fonte do repositório /* Busca pelo código-fonte no repositório informado
        Compile o código                    /* Compila código baixado
        se não houver erros faça
            construa imagem Docker da aplicação  /* Constrói imagem Docker da aplicação
            faça upload da nova imagem      /* Realiza upload da imagem para repositório Docker
            solicite revisão do gestor      /* Aguarda homologação do gestor
            se autorizado faça
                Baixe nova imagem Docker    /* Baixa Imagem no servidor de produção
                Execute o contêiner         /* Iniciar contêiner no servidor de produção
            fim
        fim
        senão faça
            envie email ao responsável do projeto  /* Notifica o responsável quanto aos erros ocorridos (Logs)
        fim
    fim
fim
```

6.5 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 3:

MÉTRICA	DESCRIÇÃO
Tempo de Deploy	Critério utilizado para avaliar o tempo gasto para uma atualização de aplicação entrar em produção
Taxa de erros dos Jobs.	Ocorrência de erros retornados durante a execução
Desempenho frente a múltiplos deploys	Medição do desempenho obtido frente à um estresse de deploys simultâneos
Recurso computacional consumido utilizando diferentes instâncias de Slaves.	Quantidade de recursos de memória e processamento utilizados entre os slaves para concluir a tarefa iniciada.

Tabela 3 – Métricas Definidas

Os resultados das métricas definidas são importantes pois com isso é possível ter noção muito próxima da "saúde"do código, pontos de fragilidade, potenciais melhorias, eventuais de melhorias nos recursos físicos e previsão de tempo de entrega de uma funcionalidade. Esses pontos dão suporte para tomadas de decisões por parte do gestor do projeto. Em se tratando do SonarQube a própria ferramenta já traz uma série de métricas pré-definidas e genéricas que podem ser aplicadas aos mais diversos tipos de jobs, e nesse ponto é importante ressaltar que é possível a criação de scripts específicos de testes de acordo com o tipo de tarefa que está sendo executada no build, no entanto o foco aqui não foi a implementação dos testes, mas sim a provisão do ambiente de automatização.

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.

TRUCCO, C. *Tudo o que você precisa saber sobre Kubernetes – Parte 01*. [S.l.]: Disponível em: <https://imasters.com.br/desenvolvimento/tudo-o-que-voce-precisa-saber-sobre-kubernetes-parte-01>, 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 Formulário

Classe que acessa API do Jenkins:

```
import org.dom4j.io.SAXReader;
import org.dom4j.Document;
import org.dom4j.Element;
import java.net.URL;
import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.util.List;

/**
* Classe que acessa API Jenkins
*/
public class Main {
        public static void main(String[] args) throws Exception {

                // seta a url do projeto
                URL url = new URL("http://jenkinsserver/api/json");

                // lancando url a documento job
                Document jobDocument = new SAXReader().read(url);

                // Busca lista de jobs
                for( Element job : (List<Element>)jobDocument.
                                        getRootElement().elements("job")) {
                        System.out.println(String.format("Name:%s\tStatus:%s",
                                job.elementText("name"), job.elementText("color")));
                }
        }
}
```

A.2 Ansible

Script para instalação do Ansible (setupansible.sh):

```
#!/bin/bash

echo "Instalando o Ansible..."
apt-get install -y software-properties-common
apt-add-repository ppa:ansible/ansible
apt-get update
apt-get install -y ansible
cp /vagrant/ansible/ansible.cfg /etc/ansible/ansible.cfg
```

A.2.1 Diretório - hosts -

Arquivo hosts:

```
[books-service]
<ip_do_web_server> ansible_ssh_private_key_file = <diretorio_da_private_key>/private_key
```

A.2.2 Playbooks do diretório - roles

Playbook Ansible chamando outros playbooks:

```
- hosts: localhost
        remote_user: admin
        become: yes
        roles:
                - java
                - docker
                - registry
                - jenkins
```

Playbook docker:

```
- name: Add Docker repository and update apt cache
        apt_repository:
                repo: deb https://apt.dockerproject.org/repo ubuntu-trusty main
                update_cache: yes
                state: present
        tags: [docker]

- name: Docker is present
        apt:
                name: docker-engine
                state: latest
                force: yes
        tags: [docker]

- name: Python-pip is present
        apt: name=python-pip state=present
        tags: [docker]

- name: Debian docker-py is present
        pip:
                name: docker-py
                version: 1.10.1
                state: present
        tags: [docker]

- name: Files are present
        copy:
                src: docker
                dest: /etc/default/docker
                register: copy_result
        tags: [docker]

- name: Docker service is restarted
        service:
```

```
            name: docker
            state: restarted
        when: copy_result|changed
        tags: [docker]
```

Playbook Java (necessário para o Jenkins):

```
- name: Package are present
        apt:
            name=openjdk-8-jdk
            state=present
            with_items: packages
        tags: [java]
```

Playbook Jenkins:

```
- name: Directories are present
        file:
            path="{{ item }}"
            state=directory
            mode=0777
            with_items: "{{ directories }}"
        tags: [jenkins]

- name: Config files are present
        copy:
            src='{{ item }}'
            dest='{{ jenkins_directory }}/{{ item }}'
            mode=0777
            with_items: "{{ configs }}"
        tags: [jenkins]

- name: Plugins are present
        copy:
            src='{{ item }}'
            dest='{{ jenkins_directory }}/plugins/{{ item }}'
            mode=0777
            with_items: "{{ plugins }}"
        tags: [jenkins]
```

A.3 Scripts Jenkins

A.3.1 Script de Configuração de Ambiente

```
default_credentials_id = "jenkins_rsa"

images {
        masterImageName = "matheusjose/code-jenkins"
}

registry {
        registryURL = "http://localhost:5000"
        registryCredentials = " "
}
```

```
global {
        numExecutorsOnMaster = 1
        jenkinsRootUrl = ""
        jenkinsAdminEmail = "Jenkins Admin <matheusetf@gmail.com>"
        scmQuietPeriod = 3
        scmCheckoutRetryCount = 3
        git {
                name = "Jenkins Git User"
                email = "matheusetf@gmail.com"
        }
        variables {
                default_credentials = "${default_credentials_id}"
                default_credentials_back = "credentials.base.credentialsId"
                default_repo = "https://github.com/mjassdev/TesteDockerJenkins.git"
                default_branch = "master"
                default_registry_url = "${registry.registryURL}"
                default_registry_credId  = "${registry.registryCredentials}"
                utility_slave = "master"
                utility_slave_old = "utility-slave"
                master_image_name = "${images.masterImageName}"
        }
        smtp {
                enabled = false
                port = "25"
                host = "1.1.1.1"
                reply_to_address = "matheusetf@gmail.com"
                authentication {
                        enabled = true
                        login = "jenkins"
                        passwordFile = "/var/jenkins_home/.ssh/.smtp_password"
                }
        }
}
```

A.4 Docker

Dockerfile:

```
FROM jenkins/jenkins:2.112

LABEL Author="Matheus Jose Alves S. Santos"

ARG master_image_version="v.2.2.2"
ENV master_image_version $master_image_version

USER jenkins

# Plugins Install
COPY plugins.txt /usr/share/jenkins/ref/plugins.txt
RUN /usr/local/bin/install-plugins.sh < /usr/share/jenkins/ref/plugins.txt

# Auto Setup Scripts

COPY src/main/resources/*.properties /var/jenkins_home/config/
```

```
COPY src/main/resources/initials/*.file /var/jenkins_home/config/initials/

# Para configuracoes de Seguranca
COPY .ssh/* /var/jenkins_home/.ssh/
```

A.5 Plugins

Arquivo *plugins.txt*:

```
kubernetes:1.5
workflow-api:2.26
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
pipeline-maven:3.5.5
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
tasks:4.52
```