Uma proposta para centralizar práticas DevOps com uso do gitlab

por

Jonnatas Matias Ribeiro Cabral

Trabalho de Graduação

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CIN - CENTRO DE INFORMÁTICA

GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

www.cin.ufpe.br

RECIFE, ABRIL DE 2017

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CIN - CENTRO DE INFORMÁTICA

GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Jonnatas Matias Ribeiro Cabral

UMA PROPOSTA PARA CENTRALIZAR PRÁTICAS DEVOPS COM USO DO GITLAB

Projeto de Graduação apresentado no Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Vinícius Cardoso Garcia

RECIFE, ABRIL DE 2017

“A menos que modifiquemos a nossa maneira de pensar, não seremos capazes de resolver os problemas causados pela forma como nos acostumamos a ver o mundo”.

Albert Einstein

Agradecimentos

Agradeço à minha família pelo amor, dedicação e paciência.

Agradeço aos meus amigos pelo incentivo e apoio durante toda a graduação.

Agradeço a minha namorada Jéssica que me apoiou nos dias que passei estudando pela noite. Ao meu Líder técnico Arthur, que me deu um tempo para pesquisar e colocar em prática alguns estudos que compõem esse trabalho.

Agradeço ao meu orientador Vinícius Garcia pela sua disponibilidade e por todo o apoio fornecido. Ao co-orientador Fish, que ajudou na ideia do projeto, além de dar aquela força.

Agradeço a todos os professores do CIn por todo o conhecimento e dedicação fornecidos durante o curso.

E a você leitor, que doará um pouco do seu tempo para ler a pesquisa.

Resumo

Embora os processos ágeis de entrega de sistemas tragam vários pontos positivos para as organizações, na prática, vários problemas são encontrados em sua execução. É necessário bastante esforço para que o caminho de um *release* até a disponibilização aos usuários seja todo automatizado. Não há um ferramenta robusta que possa ser usada ao longo do processo, permitindo assim, que cada organização ou projeto construa sua própria infraestrutura da sua maneira, utilizando o que conhece ou tem mais facilidade de colocar em prática. Essa realidade acarreta em problemas relacionados a manutenção, integração e aprendizado do uso de diferentes ferramentas.

Neste trabalho de graduação, propõe-se uma alternativa para as diferentes ambientes de entrega de sistemas existentes, sugerindo a utilização de uma única ferramenta proposta que é o *gitlab*, configurada na arquitetura de serviços da *AWS*. Facilitando o gerenciamento de releases disponibilizados em diferentes ambientes (estágio, homologação e produção) a partir de uma única visão proporcionada pela ferramenta.

Palavras-chave: *DevOps*, *infra-estrutura*, *release*, *Deploy*, *AWS* .

Abstract

Although DevOps brings several positives to organizations, in practice, several problems are encountered in its execution. It takes a lot of effort to get the road from a release to the availability to users all automated. There are a number of tools available to all parties to this process, allowing each organization or project to build its own infrastructure in its own way, using what it knows or is easier to put in practice. This reality leads to problems related to maintenance, integration and learning of the use of different tools.

In this undergraduate work, an alternative is proposed for the different existing infrastructures, suggesting the use of a single proposed tool that is emphgitlab, configured in emphAWS services architecture. Facilitating the management of releases made available in different environments (stage, homologation and production) from a single view provided by the tool.

Keywords: DevOps, release,infrastructure, gitlab.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Neste capítulo será apresentada a motivação para a realização deste trabalho, os objetivos esperados e como este trabalho está estruturado, a fim de facilitar o entendimento do leitor.

1.1 Contexto

A computação em nuvem (Cloud computing) é um modelo que permite acesso on-demand a uma rede de recursos computacionais configuráveis (por exemplo, redes, cpu, servidores, armazenamento, aplicativos e serviços) que pode ser rapidamente provisionado e liberado com o mínimo de esforço de gerenciamento ou interação do provedor de serviços [20]. Segundo o estudo [8] a computação em nuvem permite que os donos desses recursos de hardware possam oferecer esses ativos em forma de serviço e de maneira transparente a qualquer lugar do mundo que possua internet. Assim facilitando o acesso a servidores e diversos serviços disponíveis na web.

Recentemente, computação em nuvem tem sido considerada uma tecnologia do tipo must-to-have ao invés de should-to-have [14]. Computação em nuvem tem viabilizado flexibilidade da proposta pay-per-use contida nos modelos Software as a Service (SaaS), Platform as a Service (PaaS) e Infrastructure as a Service (IaaS). Além disso a alta disponibilidade (um dos fundamentos básico da proposta) [20] e as diversas plataformas presentes atualmente (PaaS) permite que organizações desenvolvam software com

maior velocidade e estabilidade (alta disponibilidade), que (i) aumenta a competitividade de organizações e (ii) proporciona qualidade do ponto de vista do cliente ou consumidor.

Embora pontos importantes tenham sido apresentados, estudos recentes revelam que implantações em cloud computing tem obtido um resultado inesperado [14]. Em outras palavras, o software não tem obtido a estabilidade esperada e a entrega do produto (software), na realidade, tem sido atrasada.

Esse fato está diretamente relacionado a divisão e barreira organizacional entre os times de desenvolvimento e operações [14]. Essa realidade pode ser relacionada a custos diretos à organização, ao qual pode-se citar a falta de compartilhamento de conhecimento entre os times. Por exemplo, o desenvolvedor não constrói aplicações que facilitem sua análise em tempo de execução (em produção), disponibilizando logs idiossincráticos (que apenas pessoas com o background de dev possa entender). Deste modo, possivelmente o time de operações, deve levar mais tempo (mais custo) analisando a causa de problema.

Visando quebrar tais barreiras, DevOps surge com intuito de aproximar os times de desenvolvimento e operação mantendo a visão ágil, ”DevOps é um conjunto de práticas destinada a reduzir o tempo entre o efetuar uma alteração em um sistema até que a alteração seja colocada em Produção, assegurando ao mesmo tempo qualidade ”. Tendo em vista que de uma pequena alteração no código até chegar em produção, deve passar por diversas etapas do processo de desenvolvimento, o conjunto dessas etapas é conhecido como pipeline.

Um pipeline de entrega de sistema é constituído de várias etapas, que são desenvolver o código, armazená-lo em uma ferramenta de configuração de código, revisar, testar, validar e colocar o código em produção. Como descrito no artigo DevOps and Its Practices IBM [10] existem diversas ferramentas para cada parte especificas desse pipeline e a integração desse código é “hand-tailored” algo feito a mão pelos desenvolvedores. Não há uma ferramenta que abrace todas as etapas desse processo de entrega de sistemas. Com isso gera um grande esforço para distribuição de conhecimento, manutenção, gerenciamento e visualização das ferramentas e práticas em uso.

1.2 Objetivos

1.2.1 Objetivos Gerais

O objetivo dessa pesquisa é propor um ambiente efetivo para o uso dos processos DevOps, propondo o uso de uma ferramenta que engloba uma maior parte dos processos de entrega contínua. Demonstrando os processos que nos dão base à construção de um ambiente de entrega contínua. Abordando ferramentas dispostas no mercado, e seus recursos. Por fim demonstrar como criar um ambiente de uso para a ferramenta escolhida, fazendo uso de uma aplicação web exemplo que percorra o fluxo proposto.

1.2.2 Objetivos Específicos

- Fazer um estudo acerca dos conceitos *DevOps*, e explorar dos processos de entrega contínua;
- Fazer um estudo sobre ferramentas de entrega contínua dispostas no mercado ;
- Implementar e analisar um ambiente de entrega contínua que use uma ferramenta que auxilie uma maior parte do processo. Observar os benefícios ao aderir tal ferramenta. E por fim demonstrar como utilizar o fluxo de entrega contínua de uma aplicação web de exemplo, usando a ferramenta proposta.

1.3 Organização do Trabalho

O presente trabalho está organizado em 5 capítulos, dos quais o primeiro é a introdução e os próximos 4 capítulos estão descritos abaixo:

- No Capítulo 2 é apresentado um conjunto de definições relevantes para o entendimento do trabalho;
- O capítulo 3 é responável pelo entendimento de como funciona os de *continuous Integration* e *Continuous Deployment* nas empresas e quais são os maiores problemas enfrentados devido a essa decisão;

- O capítulo 4 apresenta uma proposta para implantação de uma ferramenta alternativa para o auxílio no processo de CI e CD.
- O capítulo 5 apresenta as conclusões retiradas de toda a pesquisa.

Capítulo 2

Conceitos

Este capítulo é responsável por apresentar conceitos chave para o melhor entendimento deste trabalho, trazendo a definição dos conceitos e uma visão geral sobre os seguintes temas: DevOps , *Continuous Deployment e Deployment Pipeline*, *microservices*, *Django*,*Docker* e o *gitlab*.

2.1 DevOps

O DevOps trata de processos de desenvolvimento e entrega de sistemas com um foco no ágil. Ele integra eficientemente as áreas de desenvolvimento e operações, facilitando assim uma conexão fluída dessas áreas tradicionalmente separadas [12].

É uma mudança organizacional em que, em vez de grupos distribuídos desempenhando funções separadamente, equipes multifuncionais trabalham em entregas de recursos operacionais contínuos. Essa abordagem ajuda a fornecer valor de forma mais rápida e contínua, reduzindo problemas devido à falta de comunicação entre os membros da equipe acelerando a resolução de problemas.

2.2 Deploy

O termo deploy, no contexto de desenvolvimento de software, significa a ação de disponibilizar seu software na nuvem ou em qualquer ambiente que possa ser utilizado pelos seus usuarios. O deploy é realizado em uma série de etapas sequenciais, assim como uma receita de bolo, com o propósito de atualizar um sistema, ou disponibilizar um sistema completamente novo [17]. As etapas de um deploy *vão depender da arquitetura do sistema e da infra-estrutura utilizada*.

2.3 Integração contínua

A integração contínua foi escrita pela primeira vez no livro de Kent Beck, Extreme Programming Explained (publicado pela primeira vez em 1999). A idéia por trás da integração contínua era que, se a integração da sua base de código é bom, por que não fazê-lo o tempo todo? No contexto de integração, ”todo o tempo”significa cada vez que alguém efetua qualquer alteração ao sistema de controle de versão [17].

As boas práticas de CI exigem que todos os desenvolvedores enviem seu trabalho para um repositório de código comum em uma base diária. Além do que, após cada integração, o sistema deve ser reconstruído e testado, para garantir que o sistema ainda esteja funcional após cada alteração [13]. Assim os desenvolvedores podem continuar seu trabalho em cima das alterações feitas e aprovadas.

As organizações que aderem à prática de integração contínua de forma eficaz são capazes de desenvolver um software muito mais rápido, e com menos bugs, do que os times que não utilizam. Os bugs são detectados muito mais cedo no ciclo de entrega, quanto mais rápido essa detecção menos custoso é para solucioná-lo, representando uma significativa economia de custos e tempo. Por isso, essa prática é considerada essencial, e para os paradigmas enfrentados pelas organizações hoje em dia talvez seja uma prática tão importante quanto usar o controle de versão no desenvolvimento.

2.4 Entrega contínua

A entrega contínua (CD) é uma disciplina de desenvolvimento de software em que estuda como o software pode ser disponibilizado para a produção a qualquer momento [15]. O Continuous Deployment é uma extensão do CD em que cada alteração é construída, testada e implantada automaticamente na produção. Assim, em contraste com CD, não há etapas ou decisões manuais entre um commit de desenvolvedor e uma implantação de produção. A motivação para automatizar a implantação na produção é Obter feedback mais rápido do uso da produção para corrigir defeitos ficam caros com o mais tardar da detecção.

A diferença entre CI e CD é mais destacada na Fig. 2.2, onde se mostra que, embora o CI seja constituído por apenas uma única gase, o CD consiste em vários estágios que verificam se o software está em condições de ser disponibilizado para uso [15].

2.5 Deployment Pipeline

Em um nível abstrato, ”o *Deployment Pipeline* é uma execução automatizada do todo o seu processo para ,entregar o software nas mãos de seus usuários, a partir de uma ferramenta de controle de versão”. [17].

Toda mudança em seu software passa por um processo complexo no corpo do *Deployment Pipeline* até ser aprovada. Esse processo envolve a construção do software (build), seguido de progressos destas compilaçõe através de múltiplos estágios de testes e implantação. Este, por sua vez, requer colaboração entre muitos indivíduos, e talvez equipes. O pipeline de implantação modela este processo em uma ferramenta de gerenciamento de *releases* e integração contínua é o que permite controlar o progresso de cada mudança à medida que se move do controle de versão para vários conjuntos de testes e implementações para liberar aos usuários.

2.6 Microservices

A arquitetura monolítica é um padrão comumente usado para o desenvolvimento de aplicações de grandes empresas."Esse padrão funciona razoavelmente bem para pequenas aplicações, pois o desenvolvimento, testes e implantação de pequenas aplicações monolíticas é relativamente simples"[21]. No entanto, para aplicações mais complexas, acabam dificultando a utilização de uma entrega contínua, devido a uma grande quantidade de componentes que estão dentro da aplicação. Para aplicações desse porte é interessante dividi-la em um conjunto de serviços.

A arquitetura de microservices tem muitas vantagens, por exemplo, serviços individuais são mais fáceis de entender e podem ser desenvolvidos e implantados de forma independente. Adotar novas tecnologias e frameworks torna-se mais fácil, pois a adoção pode ser aplicada em um serviço de cada vez Ao escolher o uso de uma arquitetura de microservices A aplicação é decomposta em diversos serviços de front end, que implementam diferentes partes da interface do usuário e múltiplos serviços de back end [19].

2.7 Contêineres

Usando contêineres para receber seu projeto, tudo o que é necessário para fazer um pedaço de sistema executado é empacotado em contêineres isolados. Ao contrário das máquinas virtuais, os contêineres não agrupam um sistema operacional completo, apenas são necessárias bibliotecas e configurações necessárias para que o trabalho do software seja necessário. Isso faz com que sistemas eficientes, leves e autônomos garantam que o software sempre seja o mesmo, independentemente de onde ele é implantado [2].

A ideia por trás do uso de contêineres é permitir unificar o deploy de diferentes tipos de projetos, ao fazer o uso dessa tecnologia. Permitindo assim que esses contêineres funcionem em diferentes tipos de sistemas operacionais.

2.7.1 Docker

O Docker surgiu como uma das grandes forças para o DevOps, é uma ferramenta que permite a criação de microservices e facilita a configuração de diversos ambientes de desenvolvimento [7]. Assim como na sua descrição:

O Docker é a principal plataforma de contêineres de software no mercado. Os desenvolvedores usam o Docker para eliminar os problemas da famosa frase ”funciona na minha máquina”ao colaborar no código com colegas de trabalho. Os operadores usam o Docker para executar e gerenciar aplicativos lado a lado em recipientes isolados para obter uma melhor densidade de computação. As empresas usam o Docker para criar pipelines de entrega de software ágil para enviar novos recursos de forma mais rápida, segura e com confiança para aplicativos Linux e Windows Server [2].

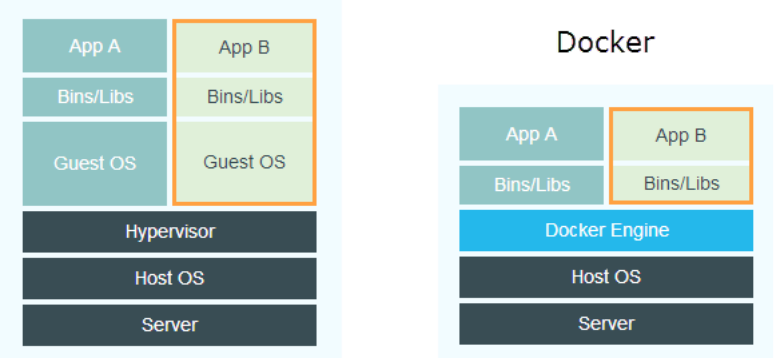

Figura 2.1: Arquitetura do docker, evidenciando o compartilhamento de recursos.

2.8 Django

O Django é um framework web que será usado na execução do ambiente proposto na pesquisa. ”O Django é um framework Python Web de alto nível que incentiva o desenvolvimento rápido e o design limpo e pragmático”[3]. A idea do framework é facilitar o desenvolvimento de aplicações web, servindo diversas funcionalidades básicas que esse tipo de projeto exige, tais como autenticação de usuário e formulários, além de criar uma interface com alguns bancos de dados, com isso o desenvolvedor não precisa se preocupar em entender consultas em SQL, basta ter conhecimento da ferramenta e da linguagem python.

2.9 GitLab

GitLab é uma ferramenta que unifica issues, revisão de código, CI e CD em uma UI única [4]. A ferramenta possui continuous integration Integrada e continuous deployment para testar, construir e implantar seu código. Você pode monitorar facilmente o progresso de seus testes e criar pipelines. Permitindo a disponibilização do seu projeto com a confiança de que seu código foi testado em vários ambientes.

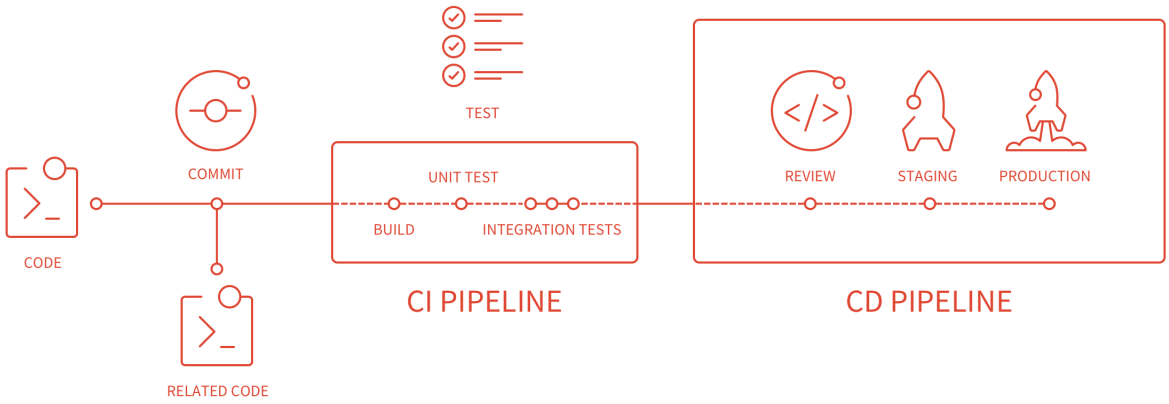

Figura 2.2: Abrangência da arquitetura do gitlab, evidenciando o englobamento dos pipelines de CI e CD.

2.10 Considerações Finais

Este capítulo apresentou os conceitos básicos que serão utilizados no trabalho. Inicialmente, foram apresentadas noções de DevOps, Continuous Integration, Continuous Integration, Deployment Pipeline, Docker, Microservices e o gitlab. Embora o que foi apresentado até aqui seja de grande importância, ainda é necessário dar um foco maior nos conceitos de CI e CD. O próximo capítulo conta com os conceitos da arquitetura do Gitlab, e apresenta técnicas de implementação, citando algumas ferramentas do mercado, a fim de sintetizar os conhecimentos existentes na área.

Capítulo 3

Ambiente de Entrega Contínua

Neste capítulo falaremos sobre os processos que compõem o ciclo de entrega contínua. O objetivo desta etapa é demonstrar a necessidade de cada passo dentro de todo o processo. Explorando benefícios, desafios enfrentados para criar ambiente de entrega contínua e por fim demonstrar algumas ferramentas que auxiliam a execução dos processos.

3.1 A Adoção do Processo de Entrega Contínua

Para a adoção dos processos de entrega contínua é necessário a definição de um conjunto de ações, as quais, a organização de desenvolvimento de software deve executar para incorporar esses processos no seu ambiente de desenvolvimento [16]. O conjunto real de ações depende do ponto de partida da organização e de algumas características da aplicação, assim a adoção pode ser diferente de caso a caso.

É necessário entender o seu ambiente de desenvolvimento e buscar fontes que cubram suas necessidades, algumas fontes fornecem ações específicas que precisam ser feitas durante a adoção A adoção mais provável requer interação e ações específicas de caso, por isso as implementações de CD diferem entre os profissionais e organizações[15].

Para a construção de um ambiente de entrega contínua, é necessário um alto nível de automação de atividades, para que assim exista capacidade de executar entregas

efetivas em curtos espaços de tempo. Para isso, algumas práticas e ferramentas são necessárias. Vamos abordar os processos, e ferramentas citadas anteriormente, nas seções a seguintes.

3.2 Pipeline

Definir um pipeline é uma etapa inicial e importante para a construção de um processo de entrega contínua. Um pipeline é basicamente a definição de um conjuntos de tarefas que devem ser executadas quando houver alguma alteração a ser integrada no código da aplicação.

Os pipelines dentro de ambientes de entrega contínua, foca em definir esses passos que agem de uma maneira incremental, ou seja, cada passo tem um status, e os passos seguintes sempre são executados se o status da atividade atual for executado com sucesso. Na figura 3.1, temos um exemplo de pipeline.

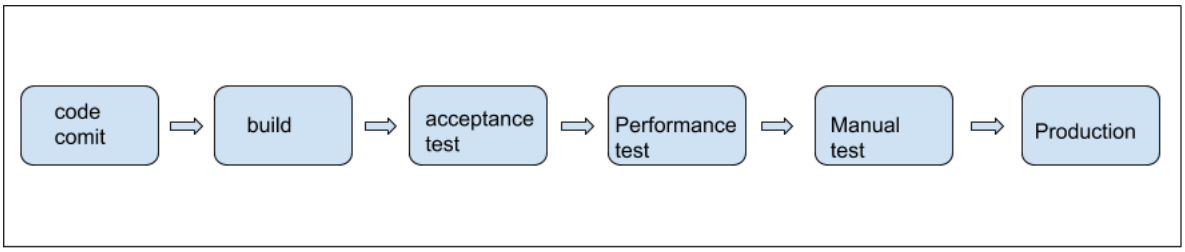

Figura 3.1: Corpo do pipeline descrito no paragrafo acima.

3.3 The commit stage (Etapa de validação das mudanças no código)

Esta seção abordará os processos que envolvem a etapa de ”commit stage”, essa etapa compõe atividades como : Compilação de código, execução de uma suíte de testes, análise da qualidade do código e a preparação dos releases e dependências que serão enviados para ambientes de uso [15], o cumprimento de todas essas etapas tem como

objetivo a criação de um release candidato a ser testado em ambientes próximos ao do usuário, e em consequência o de produção.

Essa etapa é importante para o entendimento da grande necessidade de automatizar alguns processo no desenvolvimento de sistemas, além de mencionar a importância da adoção das devidas práticas de desenvolvimento.

3.3.1 Controle de Versão

Esta etapa é de bastante importância para todo o processo de entrega contínua, não só pela escolha da sua ferramenta mas pela importância dos processos contido nesta etapa. Alguns dos processos desta etapa são práticas de clonagem do código, ramificações e integração de alterações no código [9].

Todos esses processos focam a qualidade de entrega de código e práticas que definem os requisitos necessários para que uma alteração seja integrada ao linha principal de desenvolvimento. Exemplo desses requisitos podem ser padrões de código, sucesso dos testes automáticos, além das condições de integração de código.

Hoje em dia, existem algumas ferramentas muito utilizadas que são o git e o SVN, devido a essas ferramentas diversos serviços surgiram para facilitar o gerenciamento dos processos citados, isso faz com que todo o time esteja acompanhando tudo que está acontecendo dentro do código, relacionado a alterações e integrações, existem vários serviços com essas funcionalidades tais como github, bitbucket. Alguns novos serviços já se preocupam em abranger tanto a etapa de integração quanto a etapa de deploy, exemplos como: Buddy e gitlab.

3.3.2 Build

”Build”, no contexto do desenvolvimento de software, se da a uma versão *”compilada”* de um sistema ou parte dele que contém um conjunto de recursos que poderão integrar um produto final.

”A construção de automação é um ponto fundamental para os processos de entrega

contínua"[17]. Isso quer dizer que se no fluxo de entrega de um projeto ainda existe aquelas atividades manuais como: execução de testes feitas em ambiente de desenvolvimento ou configuração de ambientes de testes feitas por um desenvolvedor, é necessário que uma mudança de pensamento da equipe para que exista uma dedicação de tempo maior a automação dessas e outras atividades manuais.

Neste ambiente de automação, não necessariamente é preciso alocar todo o tempo de um desenvolvedor para concluir essas atividades, mas sim, é preciso um pouco de tempo para que essa necessidade seja suprida, mas para isso existem diversas ferramentas as quais dão suporte e facilitam a conclusão dessa automação.

As ferramentas de build, possuem alguns objetivos em comum alguns deles são: compilar o código da sua aplicação, gerenciar as dependências necessárias, tais como bibliotecas de código, além de executar testes e implantar a devida aplicação em diferentes ambientes [12].

Geralmente as ferramentas estão divididas entre as que tem como objetivo a construção da aplicação, exemplo do Make, Maven, Gradle entre outras, e as que tem o objetivo de implantar a aplicação, tais como Puppet, chef e ansible. Todas essas ferramentas estão disponíveis na internet.

3.4 Testes

Os testes de aceitação tem como principal requisito atender os requisitos dos usuários, e alguns testes precisam ser executados em ambientes próximos ao de produção [17], porém a execução dessa etapa não é feita só por uma ferramenta, é necessário que exista configuração da aplicação em um ambiente de teste, já que a execução desses testes geralmente simulam a interação do usuário com a aplicação.

Existem algumas atividades fundamentais nesta etapa, elas são: Configurar ambiente de testes, isso inclui alocar máquina virtual, checar compatibilidade de sistemas operacionais e instalar dependências, depois de executar esses passos, e garantir que a máquina possui os requisitos necessários para a aplicação, o próximo passo seria construir sua aplicação no ambiente. lembrando que todos esses passos devem ser automatizados com auxílio de uma ferramenta de build, como descrito na seção 3.3.2.

3.5 Ambientes

O ambientes são a ponta final do percusso de entrega contínua, neles devem ser garantido a execução de todos os requisitos técnicos e os requisitos do usuário, para o funcionamento da aplicação. Comumente são usados os ambientes de estágio e homologação como ambientes de teste, o requisito para a entrega a produção é o sucesso nessas etapas. Sempre tendo em mente que o deploy para esses ambiente deve levar apenas o clique de um botão [11] tendo em vista assim alcançar um ambiente de entrega contínua.

3.6 Trabalho em equipe

O trabalho em equipe é o ponto fundamental de todo o processo, é por isso que o DevOps é tratados por muitos como uma cultura [12]. O pensamento DevOps deve estar contido em cada atividade executada no processo de desenvolvimento, é por isso que o DevOps precisa de pessoas multi-diciplinares, pessoas essas que entendem todo o processo, e estão dispostas a realidade de que seu trabalho só tem valor quando é entregue ao usuário final.

3.7 Desafios

Após abordar sobre os métodos e práticas para implantar um processo de entrega contínua (CD). Iremos falar sobre dificuldades, pois colocar o processo em prática pode ser um desafio, tanto relacionados às pessoas envolvidas quanto nas ferramentas e práticas utilizadas.

Os desafios técnicos serão o foco dessa pesquisa, essa seção abrirá caminhos para questões que possam ser solucionadas em um ambiente de entrega contínua, relacionadas a parte técnica do processo.

Uma solução robusta, abrangente e ainda altamente personalizável para CD ainda não existe [11]. Então, fica aberto o desenvolvimento da sua própria solução. Quando estamos construindo o caminho para ter um ambiente de CD, usamos muitas ferramentas

e tecnologias diferentes como parte dessa construção. Evitar as falhas de serviços terceiros é um desafio. Lidar com aplicativos que não são acessíveis ao CD (por exemplo, aplicativos grandes monolíticos) também são desafiadores. Existe uma grande quantidade de desafios a serem citados dentro de um ambiente de CD, porém nessa pesquisa vamos focar nos desafios voltados a parte técnica do processo, tais como implantação, escalabilidade, suporte a novas tecnologias e implantação;

3.8 Ferramentas

Vamos falar sobre algumas ferramentas bastantes usadas pelas organizações que trabalham com CD no processo de desenvolvimento, no intuito de esclarecer o funcionamento de algumas ferramentas que já fazem parte do mercado a algum tempo, e as novas ferramentas que tem uma abordagem mais recentes e comumente usam containers para prover certas do dependências no nosso pipeline de CD.

Existem alguns serviços ofertados no mercado. Algumas ferramentas como o Travisci disponibilizam uma versão gratuita para desenvolvimento de projetos open source. Outras ferramentas como codeship, apenas disponibilizam versões pagas. E também temos o gitlab e o jenkins que fornecem versões “self-hosted” onde você configura a ferramenta proposta na sua própria infraestrutura. nesta seção vamos falar um pouco sobre estas ferramentas, e analisar o que elas oferecem dentro do processo de entrega contínua.

3.8.1 Jenkins

Vamos abordar alguns pontos de uma das ferramentas mais usadas do mercado, o Jenkins. A ideia desse trecho é mostrar seus recursos e o que ela oferece dentro do processo de entrega contínua.

O jenkins é uma ferramenta desenvolvida em Java, que auxilia os processos de integração e entrega contínua [5], ele é uma ferramenta que tem suporte para sistemas operacionais tais como: Windows, Mac OS X e Unix. Outro ponto importante é que a ferramenta é ”self-hosted”ela precisa ser instalada e configurada na sua infraestrutura.

O Jenkins é uma ferramenta que está a bastante tempo sendo utilizada por diversas equipes que focam processos de entrega contínua. Demonstrando assim que mesmo com surgimento de novas ferramentas, o Jenkins continua sendo bastante efetivo no cenário. Um dos problemas da ferramenta é que ela veio bem antes do surgimento das ferramentas de contêineres, com isso a ferramenta não possui recursos suficientes para projetos que utilizam contêineres em sua execução.

3.8.2 Travis-CI

Nesta seção mostraremos alguns recursos oferecidos pelo serviço do Travis-CI. O Travis-CI é um serviço que se encaixa na etapa final da entrega contínua, ele constrói seu projeto e efetua o deploy .

Alguns recursos do Travis-CI são 3.2:

- Visualização de testes enquanto executam;
- Integrações com serviços de comunicação: Slack, HipChat, e-mails;
- Disponibilização uma máquina virtual limpa para cada compilação;
- Execução testes em paralelo;
- Suporte a Linux e Mac (e iOS);
- Suporte ao Docker.

Custo básico para uso da ferramenta, está disponível no seu site [6] como visualizado na figura.

3.8.3 CodeShip

O CodeShip é mais um serviço que aborda algumas etapas do processo de entrega contínua, assim como o Travis-CI ele se encaixa no fim do processo, que é construir e executar testes, e fazer o deploy do projeto em ambientes de estágio, e produção.

Figura 3.2: Precificação de serviços do Travis-ci.[6]

- Servidor CI que escala sua necessidade.
- Deploys automáticos assim que os testes são executados com sucesso
- Execução de testes em paralelo
- Suporte ao Docker em alguns planos

Os custos para utilização desta ferramenta depende das necessidades, e grandeza do projeto. Na figura 3.3 contém o preço disponibilizado pelo site do CodeShip.

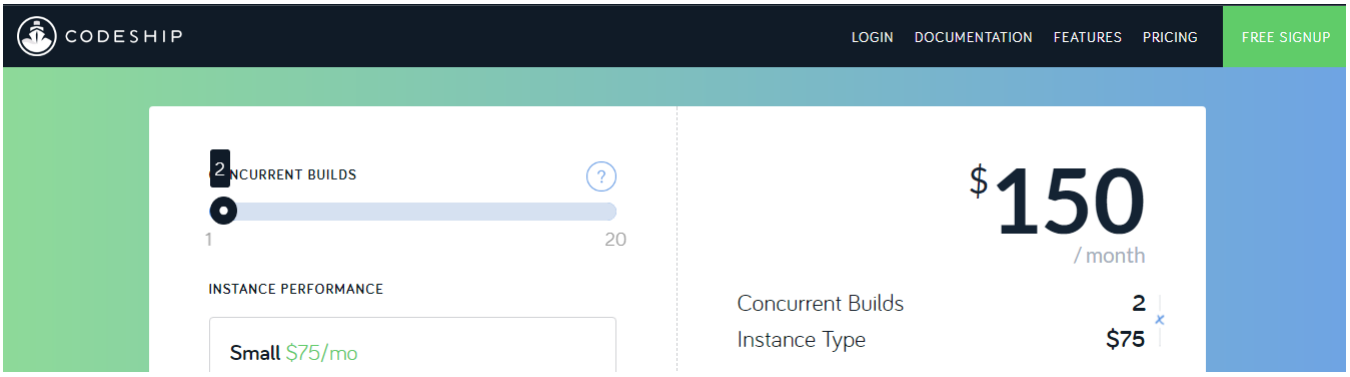

Figura 3.3: Precificação de serviços do CodeShip.[1]

3.8.4 GitLab-ce (Community edition)

Neste seção abordaremos um pouco sobre a ferramenta gitlab-ce que se encontra disponível para a comunidade. Mostrar vantagens e desvantagens. E falar um pouco da arquitetura da ferramenta, e seus recursos. Todas as informações deste capítulo foram tiradas do site do gitlab, e de sua documentação [4].

O gitlab é uma ferramenta dividida em micro serviços, permitindo assim que ela seja usada para coisas mais simples como: usar uma ferramenta de versionamento de código, ou até ser utilizada em projetos com uma complexidade maior, fazendo sua utilização para o *Continuous Integration* e *Deployment Pipelines*. A ferramenta tem integrações para CI e CD para testar, construir e disponibilizar sistemas.

A lista a seguir demonstra alguns serviços disponíveis no Gitlab-CE.

- Ferramentas integradas Integrado: GitLab CI é parte de GitLab.
- Interface: GitLab CI oferece boas interfaces para o desenvolvedor que são, familiares a ferramentas do mesmo tipo, fácil de usar. [4]
- Escalabilidade: Os testes rodam distribuídos em máquinas separadas do gitlab server, e o desenvolvedor pode adicionar quantas forem necessárias.
- Resultados e feedbacks rápidos: Cada build de um projeto pode ser dividido em múltiplos *jobs* que serão processados paralelamente pelas máquinas de teste.
- Continuous Delivery (CD):Múltiplos estágios, deploys manual, diversos ambientes e variáveis.

A próxima lista demonstra algumas características da ferramenta.Algumas destas características importantes são:

- Multi-plataforma: você pode executar compilações no Unix, Windows, OSX e em qualquer outra plataforma que suporte o Go.
- Versátil: os scripts de compilação são orientados por linha de comando e funcionam com Java, PHP, Ruby, C e qualquer outra linguagem.

- Estável: suas compilações são executadas em uma máquina diferente do GitLab.
- *Builds* paralelos: o GitLab divide divisões sobre várias máquinas, para execução rápida.
- Registro em tempo real: um link na solicitação de mesclagem leva você ao registro de compilação atual que atualiza dinamicamente.
- Testes em versão: um arquivo .gitlab-ci.yml que contém seus testes, permitindo que todos contribuam com mudanças e assegurem que cada ramo obtenha os testes que ele precisa.
- Pipeline: você pode definir vários trabalhos por etapa e você pode desencadear outras compilações.
- Autoscaling: você pode escalar automaticamente para cima e para baixo suas VM's para garantir que suas compilações sejam processadas imediatamente e minimizar custos.
- Construa artefatos: capacidade de enviar binários e outros artefatos de compilação para o GitLab e navegar e fazer o download deles.
- Suporte do Docker: Capacidade de usar imagens Docker personalizadas, serviços de spin up como parte do teste, criar novas imagens do Docker.

3.8.5 Analise das Ferramentas

Na figura 3.4, faremos uma comparação entre os recursos oferecidos pelas ferramentas citadas anteriormente, com adição da ferramenta do gitlab-ce. E quais etapas do processo de entrega continua essas ferramentas dão suporte.

Como visto na imagem 3.4, as três ferramentas oferecem suporte para mesmas etapas do processo, evidenciando a falta da etapa de *code commit* em todas.

Uma das importâncias do gitlab é que a ferramenta possui uma ferramenta de controle de versão, que proporciona uma visualização da etapa de code commit junto as etapas de build, testes e deploy. Com isso, no momento em que um desenvolvedor vai avaliar uma solicitação de integração no código, a ferramenta proporciona informações sobre

Ferramentas	Controle de versão	Build	Execução de Testes unitários	Execução Testes de Aceitação	Execução Testes manuais	Deploy
Travis-CI		x	x	x		x
CodeShip		x	x	x		x
Jenkins		x	x	x		x
Gitlab-ce	x	x	x	x		x

Figura 3.4: Visualização das da cobertura de atapas do processo de CD proporcionadas pelas ferramentas descritas.[6], [1], [4]

a construção do projeto, informações sobre a execução da sua suíte de testes, e se a ferramenta foi capaz de enviar o projeto a algum ambiente de testes, todas essas visões junto a analise de código.

3.9 Considerações Finais

Nesta seção falamos sobre os processos de CI e CD, focando nos processos usados para obter um ambiente desejado. Logo após, descrevemos uma linha de raciocínio entre os processos e os tipos de ferramentas que devem ser adotadas para automação do processo. Mostramos alguns desafios relevantes dentro da utilização desses processos. Além da citação do forte surgimento do docker e microservices em ambientes que executam CI e CD. Por fim demonstramos alguns serviços de ferramentas ofertados ao mercado, os quais possuem suporte aos ambientes mais atuais, destacando seus recursos e seus custos de acordo com uma necessidade mínima.

Capítulo 4

Construção do ambiente com a ferramenta gitlab-ce

Neste capítulo será proposto o uso da ferramenta do gitlab para a construção de um ambiente DevOps, para que exista uma centralização no acompanhamento dos processos. Assim demonstraremos a utilização da ferramenta do gitlab, sitada no capítulo 3 desta pesquisa. Temos alguns objetivos a serem alcançados neste capítulo, que são:

- Abordar a configuração da ferramenta em um ambiente de produção.
- Utilizar um exemplo de aplicação web para percorrer o fluxo de entrega contínua.
- Demonstrar as conclusões sobre o uso da ferramenta

4.1 Arquitetura

O GitLab CI é uma parte do GitLab, uma aplicação web com uma API que armazena seu estado em um banco de dados. Ele gerencia projetos / compilações e fornece uma interface de usuário agradável, além de todos os recursos do GitLab.

O GitLab Runner é uma aplicação que processa compilações. Ele pode ser implantado separadamente em uma máquina diferente da do gitlab server e funciona com o GitLab CI através de uma API.

Para executar nossos testes, precisaremos de pelo menos uma instância do GitLab e um GitLab Runner.

4.2 Configurando as ferramentas

Nesta seção iremos implementar um tutorial exemplo de como construir um ambiente de entrega contínua usando o gitlab-ce (Community Edition) a versão *open source* da ferramenta. Iremos usar também utilizar os serviços da AWS. Lembrando que essa abordagem cobre processos de CI e CD citados na seção 3 desse trabalho, o ambiente proposto está na fig 4.1.

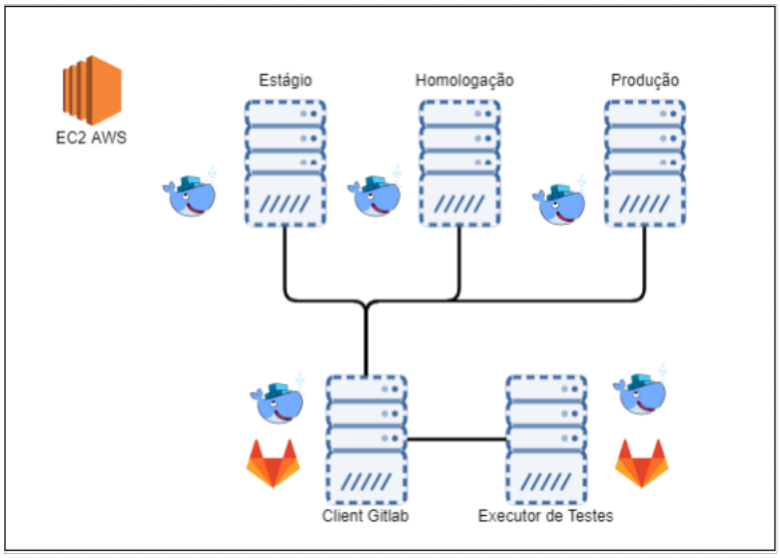

Figura 4.1: Ambiente proposto para o uso do gitlab.

Assim como numa receita de bolo, primeiro iremos fazer uma lista de tudo que precisaremos:

- Acesso a máquinas virtuais no AWS EC2.
- Ter o serviço do docker instalado em todas as máquinas que serão usadas. A AWS permite que você escolha uma máquina que já possua o docker instalado, esse passo será mostrado a mais a frente.
- Como mostrado na imagem, precisaremos de uma máquina para o servidor do gitlab, nela teremos nosso processo de commit stage, facilitando revisões e análises de erros dos commits que estarão elegíveis para o deploy.
- Outra máquina virtual terá a função de processar os builds do projeto, criando o ambiente específico e executando sua switch de testes. Para isso o gitlab disponibiliza um instalador e um token para que o servidor do gitlab use o processamento de uma outra, na execução de build e teste do projeto.
- Por fim precisaremos da máquina o qual receberá o deploy do projeto.

4.2.1 Configurando Gitlab server

Antes de instalar nosso servidor do gitlab, O primeiro passo é configurar uma VM no EC2 da AWS, as especificações da máquina estão na imagem 4.2

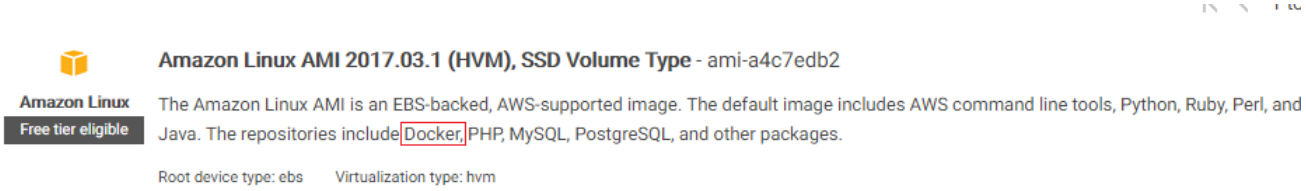

Figura 4.2: Exemplo de VM's que tem o *Docker* pre-instalado.

A única configuração específica na criação da nossa VM é a abertura de algumas portas, figura 4.3. Pois por padrão as VMs da AWS são criadas com todas suas portas fechadas para acesso externo.

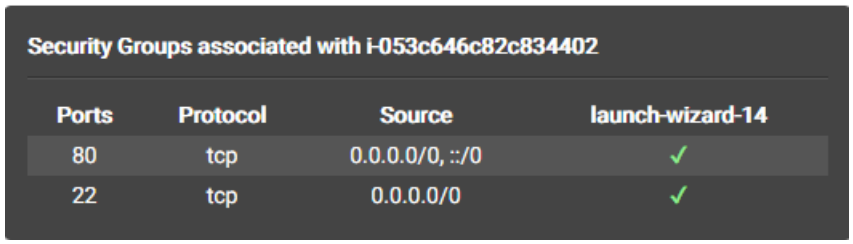

Figura 4.3: Configurações das portas ao acesso externo.

- Depois de iniciar a VM partiremos para a instalação do gitlab-ce server. Se você estiver utilizando o CentOS, os comandos abaixo também abrirão acesso HTTP e SSH no firewall do sistema.

```
$ sudo apt-get install curl openssh-server ca-certificates
   postfix
```

- Adicionando pacote do gitlab server e instalando.

```
$ curl -sS $ https://packages.gitlab.com/install/repositories/
   gitlab/gitlab-ce/script.deb.$ sh | sudo bash
$ sudo apt-get install gitlab-ce
```

- Configurando e iniciando o gitlab server

```
sudo gitlab-ctl reconfigure
```

- Navegue até o nome do host e faça o login. Na sua primeira visita, você será redirecionado para uma tela de redefinição de senha para fornecer a senha da conta de administrador inicial. Digite a senha desejada e você será redirecionado de volta para a tela de login.O nome de usuário da conta padrão é root. Forneça a senha que você criou anteriormente e faça o login. Após o login, você pode alterar o nome de usuário se desejar.

Ao finalizar essa etapa teremos nosso servidor do gitlab, onde já podemos utilizar a ferramenta de controle de versão, gerenciamento de atividades, visualização de todo o processo de entrega do sistema.

4.2.2 Configurando Gitlab Runner

Nesta seção iremos configurar a VM o qual processará os testes, e terá a função de construir e enviar a aplicação para deploy.

- Baixar o instalador do Gitlab Runner de acordo com a versão da sua VM:

```
$ sudo wget -O /usr/local/bin/gitlab-runner https://gitlab-ci-
   multi-runner-downloads.s3.amazonaws.com/latest/binaries/
   gitlab-ci-multi-runner-linux-386
$ sudo wget -O /usr/local/bin/gitlab-runner https://gitlab-ci-
   multi-runner-downloads.s3.amazonaws.com/latest/binaries/
   gitlab-ci-multi-runner-linux-amd64
$ sudo wget -O /usr/local/bin/gitlab-runner https://gitlab-ci-
   multi-runner-downloads.s3.amazonaws.com/latest/binaries/
   gitlab-ci-multi-runner-linux-arm
```

- É preciso dar permissões de execução:

```
$ sudo chmod +x /usr/local/bin/gitlab-runner
```

- Criar um usuário para o GitLab CI:

```
sudo useradd --comment 'GitLab Runner TCC' --create-home gitlab
   -runner --shell /bin/bash
```

Para registrar o Runner ao cliente Gitlab server, algumas etapas são necessárias, estas etapas são descritas a seguir:

Pre-requisitos: Antes de registrar nosso Runner, precisamos de:

- Ter o gitlab server instalado em uma VM
- Obter o token de compartilhamento fornecido via Gitlab, esse token é encontrado na página de pipeline CI/CD no gitlab server. A demonstração dessa etapa se encontra na figura 4.4:

Para registrar o runner instalado ao gitlab server precisaremos de:

- Rodar o seguinte comando:

```
$ sudo gitlab-runner register
```

- inserir a url do seu gitlab server:

```
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab
   .com )
https://gitlab.com
```

- Inserir o token que adquirimos para registrar nosso Runner:

```
Please enter the gitlab-ci token for this runner
xxxTOKENxxx
```

- Adicionar uma descrição ao Runner, devido a possibilidade te ter vários com propósitos:

```
Please enter the gitlab-ci description for this runner
[hostame] my-runner
```

- Adicionar uma tag, com essa tag você pode diferenciar os projetos que serão executados neste runner:

```
Please enter the gitlab-ci tags for this runner (comma
   separated):
my-tag,another-tag
```

- Escolher se você permite que projetos sem flags sejam executados (defaults to false):

```
Whether to run untagged jobs [true/false]:
[false]: true
```

Escolher qual programa será o executor do seu projeto, esse passo é bastante importante devido ao uso do docker, que será nosso Runner executor:

```
Please enter the executor: ssh, docker+machine, docker-ssh+
   machine, kubernetes, docker, parallels, virtualbox, docker-
   ssh, shell:
docker
```

Ao final desse passo, teremos uma infraestrutura capaz de ter um controle de versão do nosso codigo, ambiente para revisões de codigo, servidor de CI, e interface para implantar um pipeline de deploy. Todo esse processo será descrito em apenas um arquivo, que será abordado na próxima seção deste trabalho.

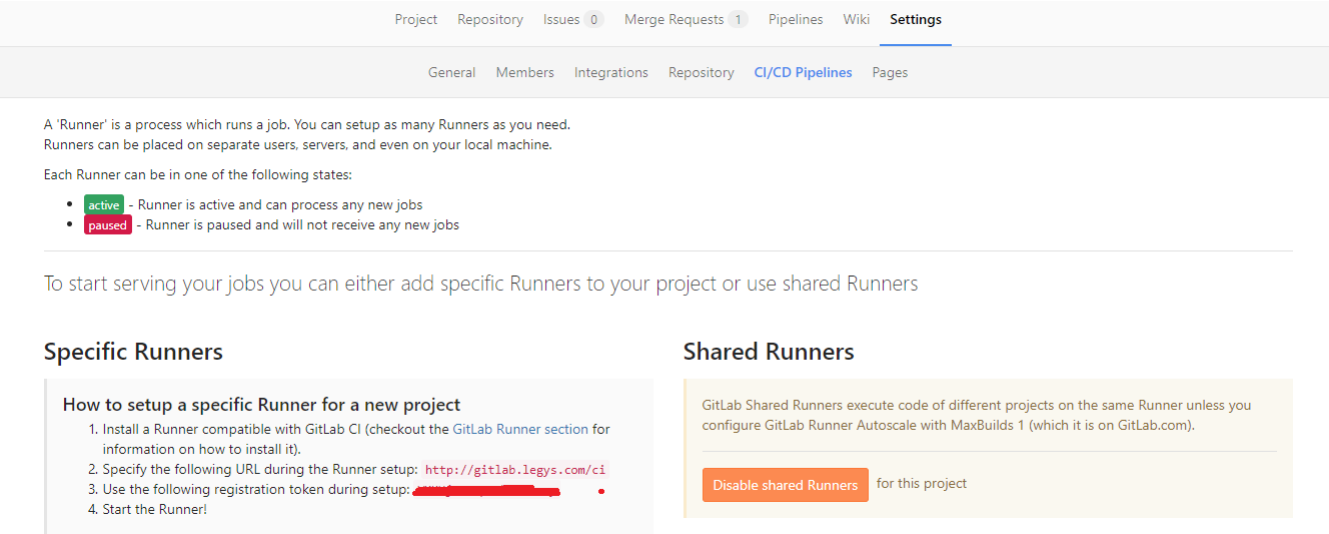

Figura 4.4: Obtenção de token para o GitLab Runner.

4.2.3 Descrevendo o Pipeline em um Único Arquivo

A partir da versão 7.12, o GitLab CI usa um arquivo YAML (.gitlab-ci.yml) para a configuração do projeto. É colocado na raiz do seu repositório e contém definições de como seu projeto deve ser construído [4].

O arquivo YAML define um conjunto de tarefas com restrições indicando quando eles devem ser executados. As tarefas são definidos como elementos de nível superior com um nome e sempre devem conter pelo menos uma linha de script.

Iremos dividir nosso pipeline em duas partes, a primeira contem nosso pipeline de CI e a segunda nossa parte de CD. Lembrando que as duas images são partes de um único arquivo 'gitlab-ci.yml'.

```
1  image: docker:latest
2
3  services:
4  - docker:dind
5
6  stages:
7  - test
8  - release
9  - deploy
10
11 before_script:
12 - apk add --update python py-pip python-dev && pip install docker-compose# install docker compose
13 - docker version
14 - docker-compose version
15
16
17 variables:
18   CONTAINER_TEST_IMAGE: 00000000000.dkr.ecr.us-east-1.amazonaws.com/TCC:test
19
20 test:
21   stage: test
22   script:
23   - docker-compose build
24   - docker-compose up -d
25   - docker-compose exec  -T TCC py.test -s --cov
26
27 release:
28   stage: release
29   script:
30   - export COMPOSE_FILE=docker-prod.yml
31   - docker-compose build
32   - docker push $CONTAINER_TEST_IMAGE
33   only:
34   - master
35
```

Figura 4.5: Primeira parte do gitlab-ci.yml.

Iremos descrever cada etapa da primeira etapa do nosso arquivo fig 4.5.

- As especificações *Image* e *services* detalham para nosso Runner, qual serviço será usado para construir nosso projeto.
- A especificação *stages* detalha os passos para construção do nosso projeto, o qual ficará visível na interface do gitlab. Assim como na imagem abaixo:
- before_script: Como a construção do projeto será executada dentro de um container docker, é preciso instalar algumas dependências adicionais, como a de exemplo na imagem.
- variables: Aqui colocamos todas as variáveis de ambientes utilizadas na aplicação, como repositório destino de releases. Em cada passo de estágio como 'test, release e deploy' os scripts para execução de cada tarefa é passado ao arquivo.

Partimos agora para descrição da segunda parte do nosso arquivo. Iremos descrever as etapas dos nossos ambientes de estágio e produção.

```
36 deploy:
37   stage: deploy
38   script:
39   - export COMPOSE_FILE=docker-prod.yml
40   - docker-compose up
41   only:
42   - master
43
44 -deploy to stage:
45   stage: deploy
46   script: make deploy
47   environment:
48     name: stage
49     url: https://stag.example.com
50
51 deploy to production:
52   stage: deploy
53   script: git push production HEAD:master
54   environment:
55     name: production
56     url: https://prod.example.com
```

Figura 4.6: Exibição dos ambientes implementados no gitlab-ci.yml.

Nesta etapa (fig 4.6)a tarefa de deploy é única para qualquer ambiente. O que precisamos notar é a descrição do atributo *enviroment* e a url destino do ambiente que receberá o *deploy*.

4.3 Executando Fluxo da Ferramenta

Nesta seção iremos demonstrar como funciona a ferramenta, ao ser usada por uma aplicação web de exemplo, todo o projeto está disponível na referência [18]. Os requisitos técnicos para a execução do pipeline são:

- Docker;
- Python;
- Django.
- Make

os passos a serem mostrados na execução da ferramenta são:

- Estrutura do projeto;
- Arquivo de configuração de build do projeto;
- Arquivo de configuração Docker
- Arquivo de configuração do pipeline, gitlab-ci.yml
- Etapa de commit stage;
- Etapa de execução de testes;
- Etapa de deploy;

O primeiro passo é demonstrar a estrutura do projeto. O projeto é uma aplicação Django, onde temos um arquivo de teste, e nosso arquivo de Build será construído em um *Makefile*, além disso temos o arquivo *Dockerfile* para configuração docker, que descreve como a aplicação é construída dentro de um container. A estrutura do projeto é demonstrada na figura 4.7.

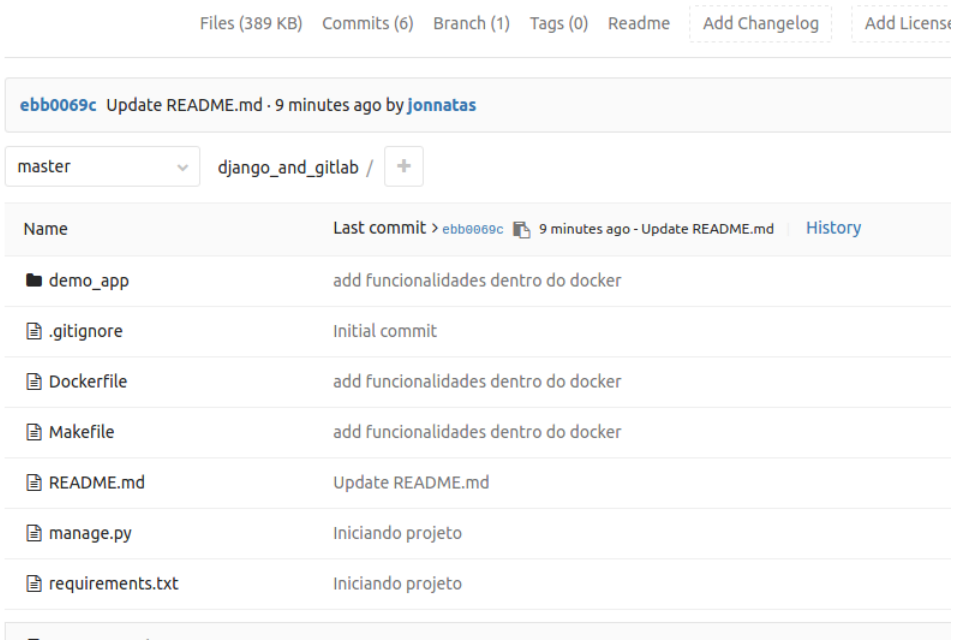

Figura 4.7: Exibição dos ambientes implementados no gitlab-ci.yml.

Makefile, nele descrevemos nossos scripts de execução, onde iremos construir o projeto, executar nossos testes, e efetuar o deploy. Arquivo descrito a seguir:

```
MANAGE_PY=python manage.py
PROJECT_NAME='django_and_gitlab'
dev: $(eval SETTINGS:=$(SETTINGS_DEV))
prod: $(eval SETTINGS:=$(SETTINGS_PROD))
build:
    @docker build  -t  demo_app_gitlab .
runserver:
    @docker run -it  -p 8080:8080 demo_app_gitlab
test:
    @docker run -it demo_app_gitlab $(MANAGE_PY) test
deploy: build runserver
```

Nesta etapa, mostraremos como configurar o *Dockerfile*, este arquivo descreve os passos necessários para gerar seus container.

```
FROM python:3.6
RUN mkdir current/
COPY . current/
COPY requirements.txt current/
```

```
WORKDIR current/
RUN pip install -r requirements.txt
RUN python manage.py migrate --noinput
CMD python manage.py runserver 0.0.0.0:8080
```

O arquivo descreve a linguagem necessária para executar o projeto, copia o projeto para dentro da imagem do container, instala as dependências, e por fim, o *CMD* descreve que quando o container for colocado em execução, ele irá executar a aplicação django.

Demonstrado os pontos anteriores, agora possuímos os passos necessários para entrar no nosso pipeline de entrega contínua, pois podemos efetuar um commit da aplicação, efetuar o build do projeto, executar os testes, e se todo o fluxo for executado com sucesso, a próxima etapa é o envio para um ambiente de testes e consequentemente de produção.

O próximo arquivo é o gitlab-ci.yml que descreve por quais etapas uma alteração deve percorrer até ser entregue ao ambiente de produção.

```
image: docker:latest
services:
- docker:dind
stages:
- test
- staging
- production
variables:
  CONTAINER_RELEASE_IMAGE: demo_app_gitlab
test:
  stage: test
  script:
  - make build
  - make test
staging:
  stage: staging
  environment:
   name: staging
   url: https://54.82.134.57
  script:
```

```
  - make deploy
  only:
  - master
deploy:
  stage: production
  environment:
   name: production
   url: https://54.82.132.11
  script:
  - make deploy
  only:
  - master
```

Todas essas etapas do *gitlab-ci.yml* foram descritas na seção 4.2.3, onde descrevemos os detalhes de cada etapa do arquivo.

Com todos esses arquivos descritos, o gatilho para a execução do pipeline é efetuar um commit no repositório e acompanhar a execução de todos os passos descritos no *gitlab-ci.yml*.

Para a etapa de commit stage, a ferramenta proporciona um ambiente de revisão de código e relatório da execução dos testes.Como podemos visualizar na figura 4.8.

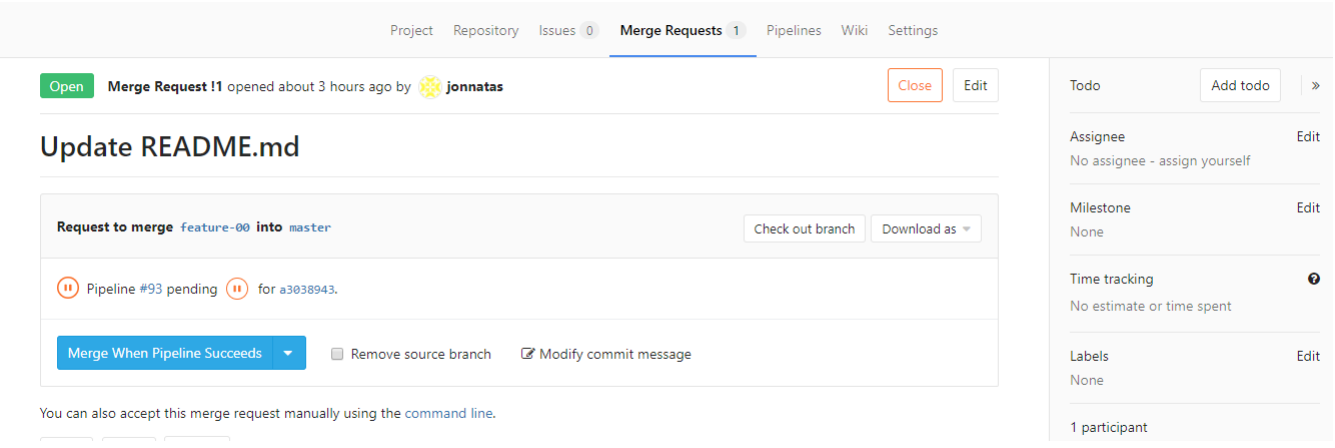

Figura 4.8: Tela para analise de alterações de codigo

A primeira visão de fluxo na ferramenta pode ser visualizada na pagina de pipelines, como na figura 4.9.

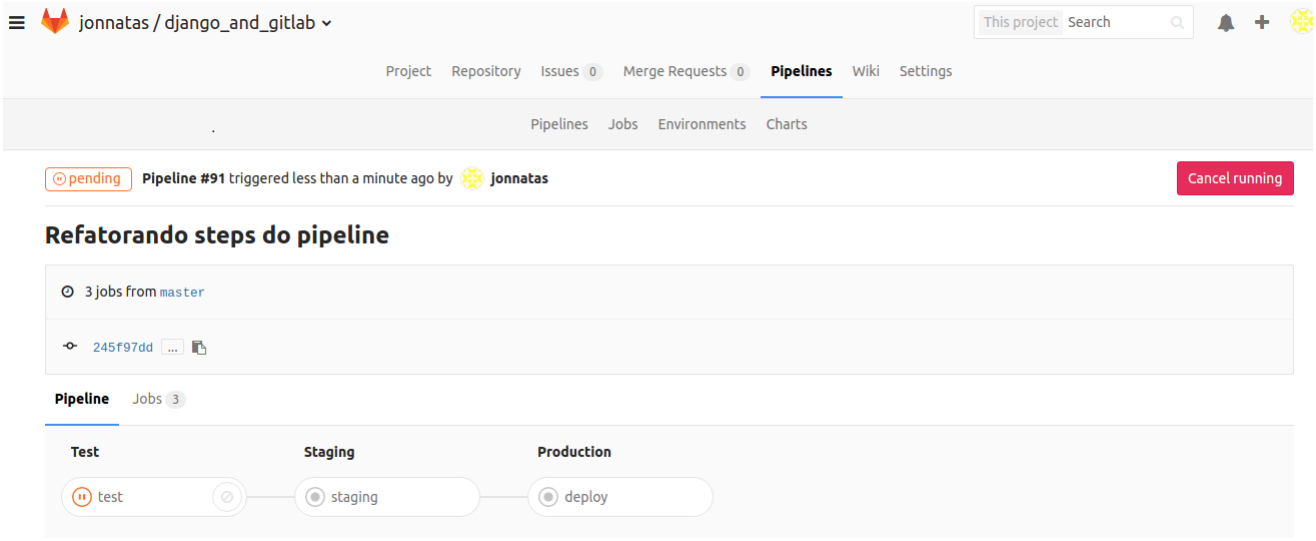

Figura 4.9: Exibição inicial da execução do pipeline

Uma nova visão pode ser observada ao clicar em uma das etapas em execução, assim teremos um acompanhamento da construção e execução de testes do projeto. Esse etapa pode ser vista na figura 4.10.

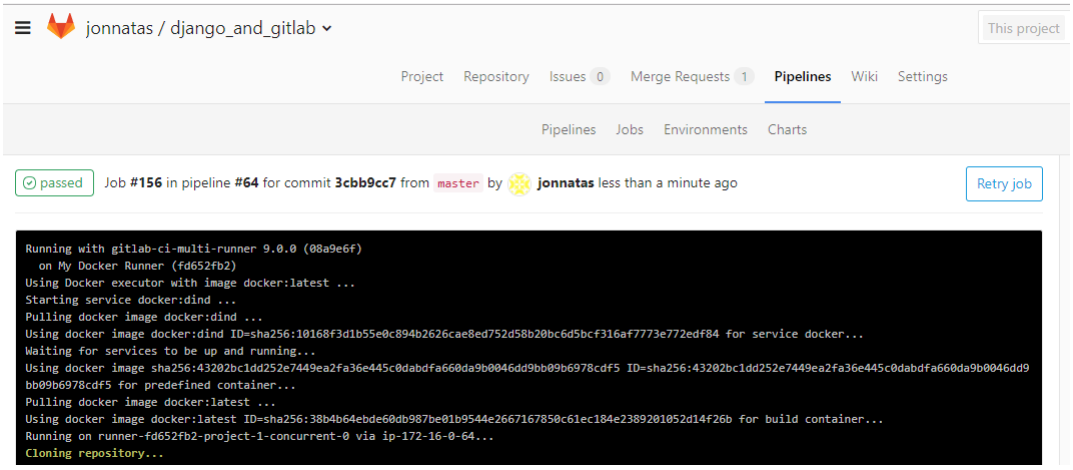

Figura 4.10: Imagem de acompanhamento de execução

Por fim da execução de todas as etapas, teremos a visão da finalização do pipeline descrito. Essa etapa pode ser vista na figura 4.11.

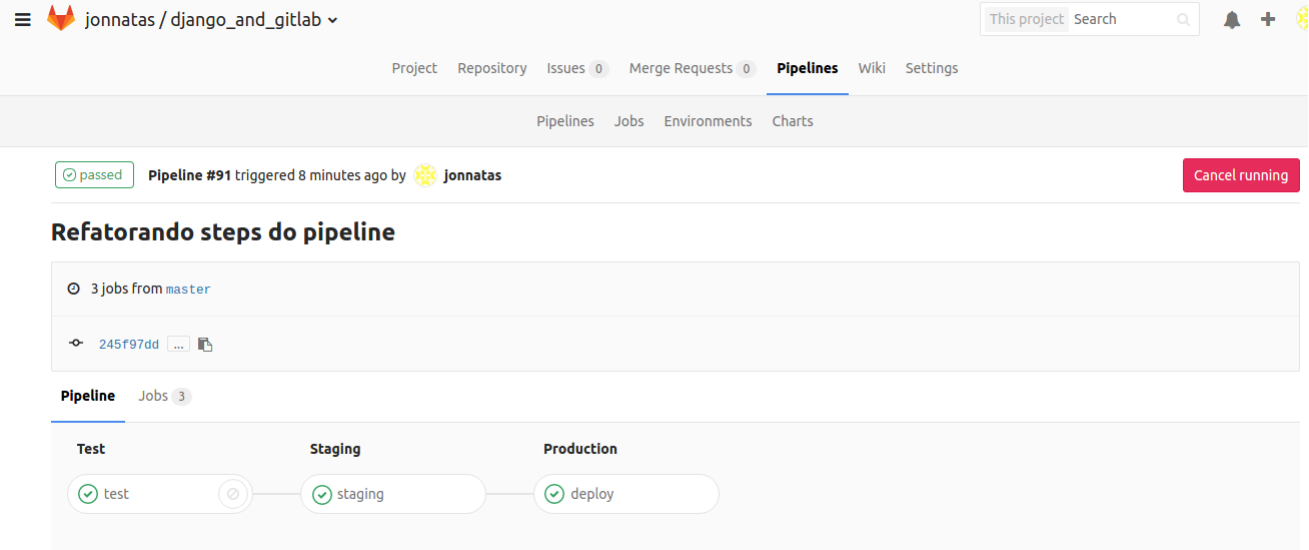

Figura 4.11: Exibição inicial da execução do pipeline

4.4 Considerações Finais

Nesta seção descrevemos um tutorial base para configuração de um ambiente de entrega contínua, criação de um pipeline genérico de CI e CD para uma aplicação web, usando o gitlab, Make e docker, e o django. Podemos conhecer um pouco sobre o gitlab Runner, ferramenta o qual podemos distribuir o processamento dos nossos testes, e escalar assim que necessário.

Assim, ao usar esse o ambiente descrito, podemos nos beneficiar da centralização da execução do pipeline de entrega contínua, tendo em vista que o versionamento e a revisão de código (commit stage), estão na ferramenta, assim como os relatórios e todo o acompanhamento das etapas de execução de testes e deploy da aplicação. Com isso demonstramos uma alternativa para os ambientes efetivos de entrega contínua.

Outro ponto interessante é demonstrar que a o custo de uso da ferramenta é viável com as tabelas.

Relação de preços comparado aos serviços citados na seção 3.

Tabela 4.1: Custos

serviço	concorrencia	preço
Travis-ci	2	$129
CodeShip	2	$150

Tabela 4.2: Custos do ambiente com gitlab

serviço	VMs	preço
gitlab-Server	1	$16.56
gitlab-Runner	2	$16.56
total		49.68

Os custos citados acima são referentes aos encontrados no site da AWS no momento da pesquisa. Segue a tabela abaixo:

	vCPU	ECU	Memória (GiB)	Armazenamento da instância (GB)	Uso do Linux/UNIX
Uso geral – Geração atual					
t2.nano	1	Variável	0.5	Somente EBS	$0.0059 por hora
t2.micro	1	Variável	1	Somente EBS	$0.012 por hora
t2.small	1	Variável	2	Somente EBS	$0.023 por hora
t2.medium	2	Variável	4	Somente EBS	$0.047 por hora
t2.large	2	Variável	8	Somente EBS	$0.094 por hora
t2.xlarge	4	Variável	16	Somente EBS	$0.188 por hora
t2.2xlarge	8	Variável	32	Somente EBS	$0.376 por hora

Figura 4.12: Preços das VMs do serviço EC2 da AWS.

Capítulo 5

Conclusão

Devido ao grande desenvolvimento de projetos fazendo utilização da Computação em Nuvem, a necessidade de ter agilidade na entrega de sistemas é foco de vários estudos. O surgimento do DevOps deixa mais evidente a necessidade de processos e ferramentas ágeis. Assim como a utilização do CI, CD , *Docker* e *microservices* , (todos citados nesta pesquisa). Este trabalho mostrou os benefícios provindos da utilização de uma biblioteca *open source do gitlab*, o *gitlab-ce* (community edition) como por exemplo uma maior centralização de recursos no gitlab e AWS, compartilhamento de recursos ao usar o *gitlab Runner*, facilidade na implementação e redução de custos com serviços de terceiros.

O objetivo principal desta pesquisa foi propor uma implementação do gitlab-ce para cobrir os processos de CI e CD, utilizando o *Docker* como provedor de dependências da aplicação. O intuito foi mostrar a ferramenta de uma forma que os desenvolvedores achea tão simples e usual, que a tenham como uma alternativa para construção de ambientes de entrega contínua.

Referências Bibliográficas

[1] Codeship. URL `https://codeship.com/`.

[2] Docker. URL `https://www.docker.com/what-docker`.

[3] Django project. URL `https://www.djangoproject.com/`.

[4] Gitlab community edition. URL `https://gitlab.com/gitlab-org/gitlab-ce`.

[5] Jenkins. URL `https://jenkins.io/`.

[6] Travis-ci. URL `https://travis-ci.com/`.

[7] Charles Anderson. Docker. 2017.

[8] Estrella Julio Cezar Ferreira Carlos Henrique Gomes Filho Dionisio Machado Leite Nakamura Luis Hideo Vasconcelos Reiff-Marganiec Stephan Santana Marcos José Santana Regina Helena Carlucc Batista, Bruno Guazzelli. *Performance Evaluation of Resource Management in Cloud Computing Environments*. 2015.

[9] Christian Bird Tamara Marshall-Keim Foutse Khomh Kim Moir Mozilla Bram Adams, Stephany Bellomo. The practice and future of release engineering. *IEE Software*, 2015.

[10] George Champlin-Scharff. Devops and its practices. *IBM*, 2016.

[11] Lianping Chen. Continuous delivery: Huge benefits, but challenges too. *IEEE software*, 2015.

[12] Josune Hernantes Christof Ebert, Gorka Gallardo and Nicolas Serran. Devops. *IEEE*, 2016.

[13] Juha Itkonena Casper Lassenius b Eero Laukkanena. Problems, causes and solutions when adopting continuous delivery. *ELSEVIER*, 2016.

[14] EMA. Devops, continuous delivery, and the cloud journey: an evolutionary model. *IEEE*, 2016.

[15] M. Fowler. Continuous delivery. 2013.

[16] AybükeAurum G.G. Claps, RichardBerntsson Svensson. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Tech*, 2015.

[17] D. HUMBLE, J.; FARLEY. Continuous delivery: Reliable software releases through build, test, and deployment automation. *ISBN*, 2010.

[18] JonnatasCabral. Django and gitlab. URL `https://github.com/JonnatasCabral/django_and_gitlab`.

[19] Alberto Lluch Lafuente Manuel Mazzara Fabrizio Montesi Ruslan Mustafin Larisa Safina Nicola Dragoni, Saverio Giallorenzo. Microservices: yesterday, today, and tomorrow. *Cornell University*, 2017.

[20] Timothy Grance Peter Mell. The nist definition of cloud computing. *IBM*, 2011.

[21] Chris Richardson. Microservices: Decomposição de aplicações para implantação e escalabilidade. *INFOQ*, 2014.