CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Engenharia de Software

DevOps e Ambientes de Execução de Software

André Marcelino de Souza Neves
andreneves3@gmail.com

Leonam Teixeira de Vasconcelos
leonam.teixeira.vasconcelos@gmail.com

Maycon Arthuso Carvalho
maycon.arth.carvalho@gmail.com

Professora: Deisymar Botega Tavares

Timóteo
2018

Sumário

1	**INTRODUÇÃO**	**3**
1.1	**Contextualização**	**3**
2	**SOBRE DEVOPS**	**5**
2.1	**Atores de sistema de um modelo DevOps genérico**	**5**
2.2	**Práticas**	**7**
2.2.1	Controle de versão	7
2.2.2	Containers	7
2.3	**Ferramentas de DevOps**	**8**
2.3.1	Docker	8
2.3.2	Kubernates	8
2.3.3	Jenkins	8
2.3.4	CI/CD	10
2.3.5	Gitlab CI/CD	11
2.4	**Vantagens**	**12**
2.5	**Desvantagens**	**12**
2.6	**Ponto de vista**	**12**
	REFERÊNCIAS	**13**

1 Introdução

Basicamente, o DevOps é um modelo que busca aproximar as equipes de desenvolvimento das equipes que controlam as infraestruturas de implantação (operação), de forma a simplificar processos, integrar equipes, obter maior qualidade com as entregas e oferecer respostas mais rápidas aos clientes (BARBOSA, 2019). Esse último ponto é o que mais auxilia a prática das metodologias ágeis, nas quais o *feedback* e comunicação constante com o cliente é um dos principais pilares.

1.1 Contextualização

Durante as aula de história, quando é estudado a revolução industrial, o fordismo e diversas lógicas industriais, foi visto que existia uma demanda por produtos manufaturados, e foi necessária uma espécie de esteira de montagem para agilizar o processo. Essa mesma ideia foi trazida para a engenharia de software com o modelo cascata, no qual, consistia em um modelo sequencial que continha muitos problemas (GUIMARÃES, 2019), (DELFINO, 2018).

Um dos problemas da tecnologia, é o seu dinamismo avassalador, que faz com que os requisitos sejam alterados a todo o tempo, precisando fazer constantes alterações no projeto. Outra dificuldade é que o cliente dificilmente sabe o que quer quando pede um software, e consequentemente, torna-se necessário alterar o projeto inúmeras vezes até que o usuário final esteja satisfeito com o resultado (DELFINO, 2018), (GUIMARÃES, 2019).

Com isso, muitos profissionais da área de desenvolvimento de software vêm percebendo a crescente necessidade de entregar softwares com maior qualidade e em menor tempo, e por isso recorrem aos métodos ágeis como o Scrum, para viabilizar estes objetivos (Sergio, 2015).

O desenvolvimento de software de qualidade, sempre foi e sempre será um grande desafio. A qualidade está relacionada com o atendimento dos requisitos solicitados pelo cliente, além da preocupação com os requisitos não funcionais, que são de grande importância para o software produzido, e cada vez esses requisitos vem crescendo em nível de importância (Sergio, 2015).

Além de buscar um bom processo de desenvolvimento, um outro ponto muito importante a ser considerado na construção de um software é a infraestrutura. Isso é o conjunto de software, hardware e redes que suporta o funcionamento do software produzido. Ter uma boa infraestrutura é essencial para cumprir os requisitos não funcionais com sucesso.

O custo de infraestrutura nunca foi pequeno ou irrelevante, logo deve ser considerado algo de grande importância. Os responsáveis pela manutenção da infraestrutura, os chamados *sysadmin* – administradores de sistema – possuem diversos desafios entre manter e evoluir a disponibilidade da infraestrutura de forma a torná-la cada vez mais confiável e com custos altamente vantajosos (Sergio, 2015).

Nesse contexto, os setores de desenvolvimento e operações precisam trabalhar em conjunto. No entanto, as equipes das duas áreas trabalham com contextos, de certa forma, bem distintos. Enquanto o time de desenvolvimento trabalha com soluções criativas que gerem valor, e é responsável por provisionar formas inteligentes de implementar as funcionalidades, o time de operações tem como

foco a estabilidade e pleno funcionamento do sistema (GUIMARÃES, 2019).

De certa forma, a inovação trazida pelo time de desenvolvimento pode significar mais trabalho e complexidade para o time de operações, pois terão que suportar novos recursos de infraestrutura. Consequentemente, a partir disso surgem problemas para que o time de operações mantenha corretamente o software desenvolvido, seja por falta de conhecimento sobre a infraestrutura exigida pelos novos recursos implementados pelos desenvolvedores, ou seja pelo abuso dos desenvolvedores no tocante à infraestrutura que suas implementações exigem, o que surge como consequência da falta de noção a respeito do funcionamento da infraestrutura disponível.

Quando existe esse desalinhamento, falhas são geradas, atrasos são comuns, retrabalhos acontecem e o produto final perde qualidade. E além dos desgaste das equipes, o cliente é o principal afetado por essa falta de sincronia (BARBOSA, 2019).

Nesse cenário, um termo tem se popularizado na área de tecnologia, o DevOps, que é a junção dos termos "development"e "operations", que tem como objetivo integrar o trabalho dessas equipes. A implantação do DevOps começa com o entendimento da cultura proposta, de forma que o operacional tenha mais conhecimento sobre o software em si, e que o desenvolvimento entenda sobre a infraestrutura operacional. Com isso, a comunicação é essencial (GUIMARÃES, 2019).

2 Sobre DevOps

A comunicação é um dos grandes alicerces do desenvolvimento ágil, pois é a forma que as equipes encontram para constantemente se atualizarem das frequentes mudanças de requisitos e adaptações de planejamento (COIS; YANKEL; CONNELL, 2014). Ela faz com que os riscos de um escopo tão mutável seja reduzido e o resultado do projeto seja aprimorado. Ter mecanismos eficientes de comunicação é essencial para alavancarmos o planejamento e o desenvolvimento e times modernos utilizam para isso ferramentas que garantem velocidade e robustez no fluxo de informação entre os membros (COIS; YANKEL; CONNELL, 2014).

Porém, segundo (COIS; YANKEL; CONNELL, 2014) produtividade e qualidade são diretamente afetadas pela comunicação entre os membros da equipe de uma forma que em falta a mesma causa redução de qualidade e em excesso causa queda de produtividade no time. Contudo, deve haver um equilíbrio de comunicação para que possamos obter o máximo de desempenho no desenvolvimento de software.

Nesta abordagem, o DevOps nos traz um benefício: o uso da automação de alguns processos de comunicação, que é minimizar os requisitos de comunicação humana enquanto maximiza-se a troca de informação e comunicação entre atores do sistema (COIS; YANKEL; CONNELL, 2014). Atores do sistema consistem em softwares responsáveis por automatizar algumas tarefas que não exigem (ou exigem pouca) interação humana, como a comunicação entre tais atores não afeta a produtividade do time (já que são máquinas e não trabalhadores humanos), a equipe de atores humanos pode se empenhar em realizar trabalhos e resolver problemas específicos que exigem da cognição e criatividade humana.

2.1 Atores de sistema de um modelo DevOps genérico

As categorias abaixo foram totalmente baseadas no trabalho de (COIS; YANKEL; CONNELL, 2014).

- Sistema de controle de versão:

  Armazena o histórico do código fonte e todo o material necessário para o mesmo ser executado;

- Sistema de controle de tarefas:

  Gerenciador de tarefas, seus status e os responsáveis pelas mesmas;

- Sistema de integração contínua:

  Sistemas que compilam, constroem e testam código fonte para produzir uma aplicação funcional associados à CI (Continuous Integration) – continuamente construir e testar aplicações à toda nova modificação no código fonte;

- Sistema de documentação:

  Algum método de criar, armazenar, compartilhar e apresentar documentações do projeto;

- Sistema de revisão de código:

  Sistema que faz a verificação de qualidade e correção do código fonte ou facilita a revisão do código por atores humanos;

- Sistema de monitoramento:

  Sistema para monitorar o estado e a funcionalidade dos outros sistemas;

- Ambiente de integração:

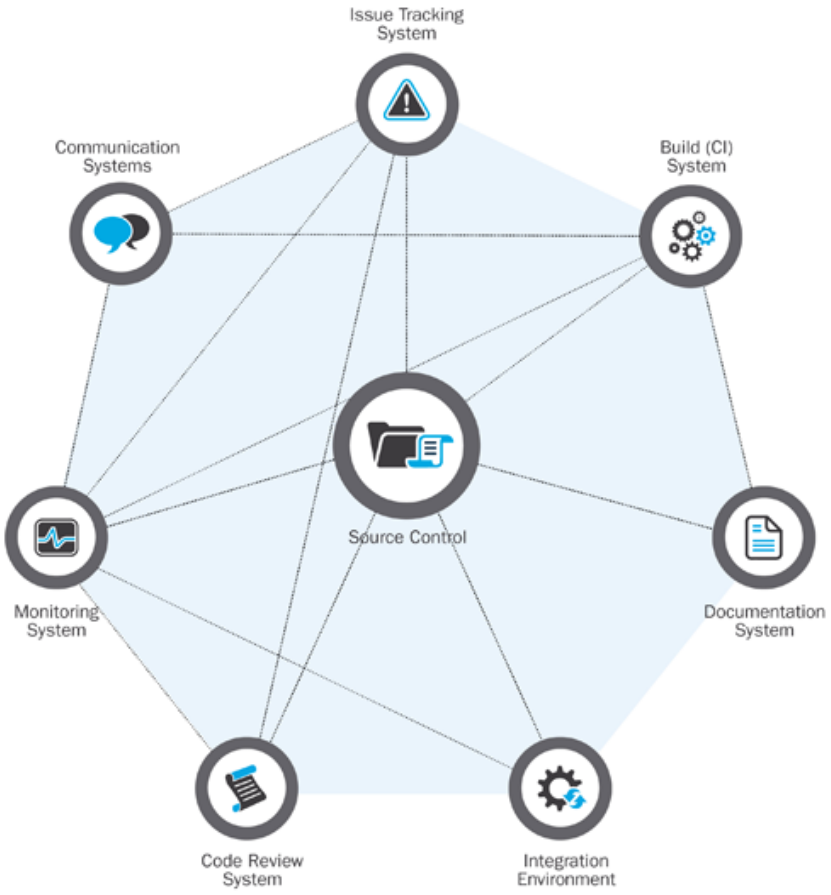
  O ambiente onde tanto os sistemas de DevOps quanto os produtos de software produzidos serão operados. Frequentemente utilizando, uma infraestrutura virtual;

- Sistemas de comunicação:

  Sistema para compartilhamento de conhecimento entre atores humanos ou entre atores humanos e atores do sistema.

A imagem (1) abaixo ilustra os items acima, destacando os canais de comunicação entre os membros da equipe.

Figura 1 – Diagrama ilustrando um modelo genérico de DevOps.



Fonte: Carnegie Mellon University [1].

[1] <https://insights.sei.cmu.edu/sei_blog/2014/06/a-generalized-model-for-automated-devops.html>

2.2 Práticas

2.2.1 Controle de versão

Um sistema de controle de versão agrega ferramentas para gerenciar diferentes versões de objetos de software que são produzidos durante o processo de desenvolvimento (PRESSMAN, 2005), e são relacionados com quatro principais capacidades:

- Um repositório de projeto, que armazena todo conteúdo relevante.
- Um controlador de versão, que armazena todas as versões de objetos ou permite a reconstrução de qualquer versão através de diferenciação.
- Funcionalidade que permite a construção de qualquer versão específica do software.
- Um gerenciador de tarefas ou issues que permite os desenvolvedores recordarem e localizarem o status de tarefas relacionadas com cada arquivo ou código fonte.

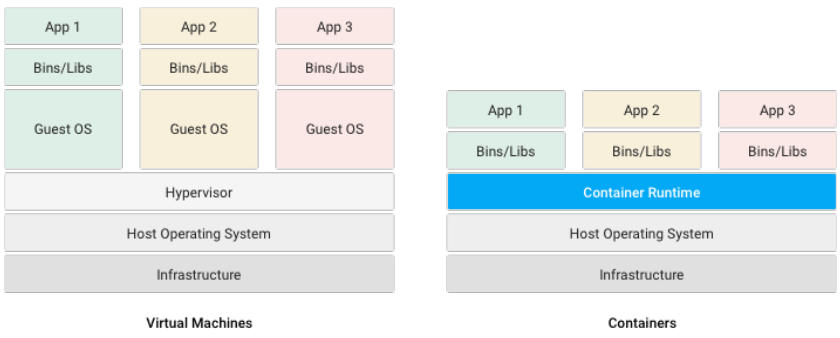
Alguns sistemas de controle de versão são **Git**, **Mercurial** e **SVN**.

2.2.2 Containers

Um container consiste em uma unidade padrão de software que encapsula logicamente o ambiente de execução de um software (código fonte e suas dependências) para que as aplicações executem rápido e de forma consistente em cada ambiente computacional (Docker, 2019; Cloud google, 2018). A utilização de containers permite que a equipe de produção desenvolva mais rapidamente e façam implantações eficientes com responsabilidades bem separadas por parte da equipe – desenvolvedores focados em aplicações e dependências e funcionários de TI focados em implantação e gerenciamento sem se preocuparem com detalhes do software – (Cloud google, 2018).

De forma divergente a uma execução virtualizada, onde temos que virtualizar todo o hardware de uma máquina, um container opera com uma virtualização a nível de sistema operacional o que implica em uma execução em container é muito mais leve e eficiente. Todos os containers executados em uma máquina compartilham o kernel do sistema operacional nativo daquele computador, inicializando muito mais rápido e utilizando uma fração da memória requerida por uma execução em uma máquina virtual dedicada (Cloud google, 2018). A figura (2) ilustra tal diferença.

Figura 2 – Diagrama comparando a arquitetura de container e a arquitetura de máquina virtual.



Fonte: Google Cloud [2].

Alguns benefícios de execução de softwares em containers são listados abaixo de acordo com o site oficial da Google Cloud (Cloud google, 2018):

- Ambiente consistente:

  containers permitem que os desenvolvedores criem ambientes que não irão sofrer interferências de outras aplicações, podendo ser personalizados com as dependência da aplicação específica que será executada.

- Portabilidade:

  o uso de container permite que uma aplicação execute de forma idêntica em diferentes plataformas, já que há um encapsulamento do ambiente de execução da mesma.

- Isolamento:

  containers provêem versões lógicas de processador, memória armazenamento e rede, que são exclusivas logicamente àquele container e nenhum outro container com outra aplicação irá interferir.

2.3 Ferramentas de DevOps

2.3.1 Docker

Docker é uma tecnologia de containers de código aberto lançada em 2013 que impulsionou a aplicação de containers no mundo de desenvolvimento de software, principalmente nos ambientes Linux, que com o sucesso que obteve trouxe parcerias com a Microsoft, criando-se assim o recurso para ambientes Windows. O foco dos containers Docker é a separação de dependências de aplicações de requisitos de infraestrutura, desassociando assim responsabilidades de desenvolvedres e operadores de sistema (Docker, 2019).

2.3.2 Kubernates

Kubernetes, ou assim conhecido como *"k8s"*, é um software de código aberto que provê o ferramental necessário para coordenar e gerenciar containers de uma aplicação, verificando o estado do container, reiniciando-os caso necessário e fazendo adaptações de escalabilidade à medida que os serviços são executados. Também são utilizados para agrupar vários containers que fazem parte de uma aplicação em unidades lógicas que facilitam o gerenciamento (Cloud google, 2018; Kubernetes, 2019).

2.3.3 Jenkins

Jenkins é uma ferramenta de automação de código aberto escrita em Java com plugins criados para fins de Integração Contínua. O Jenkins é usado para criar, testar e entregar, ou implantar, projetos

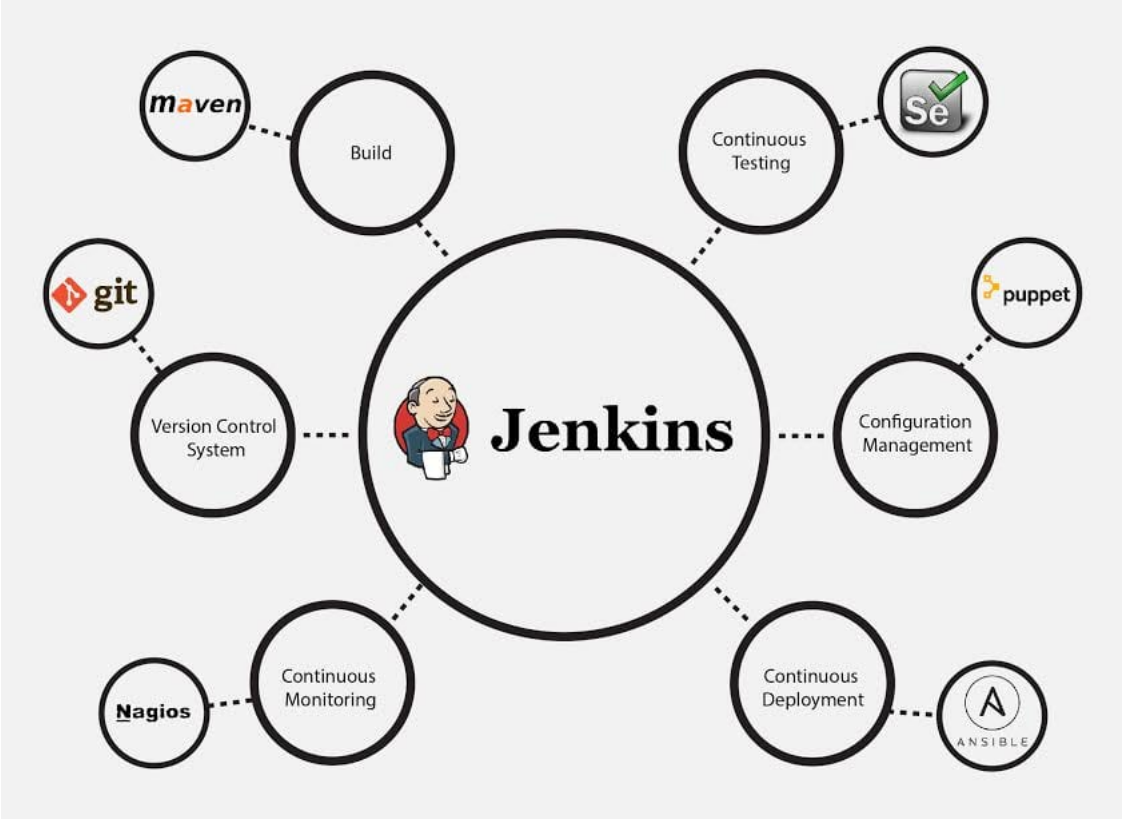
[2] <https://cloud.google.com/containers/>

de software continuamente, facilitando para os desenvolvedores a integração de alterações no projeto e a obtenção de uma nova compilação pelos usuários (Saurabh, 2019).

Jenkins integra processos de ciclo de vida de desenvolvimento de todos os tipos, incluindo construção, documento, teste, pacote, estágio, implantação, análise estática e muito mais (Saurabh, 2019).

Jenkins alcança a integração contínua com a ajuda de plugins. Plugins permitem a integração de vários estágios do DevOps. A figura 3 representa o Jenkins interagindo com vários estágios do DevOps (Saurabh, 2019).

Figura 3 – Jenkins está integrando vários estágios do DevOps



Fonte: Edureka [3].

Em um cenário em que o código fonte completo do aplicativo foi criado e, em seguida, implantado no servidor de teste para ser testado, parece uma maneira perfeita de desenvolver um software. Porém esse processo tem muitas falhas, como (Saurabh, 2019):

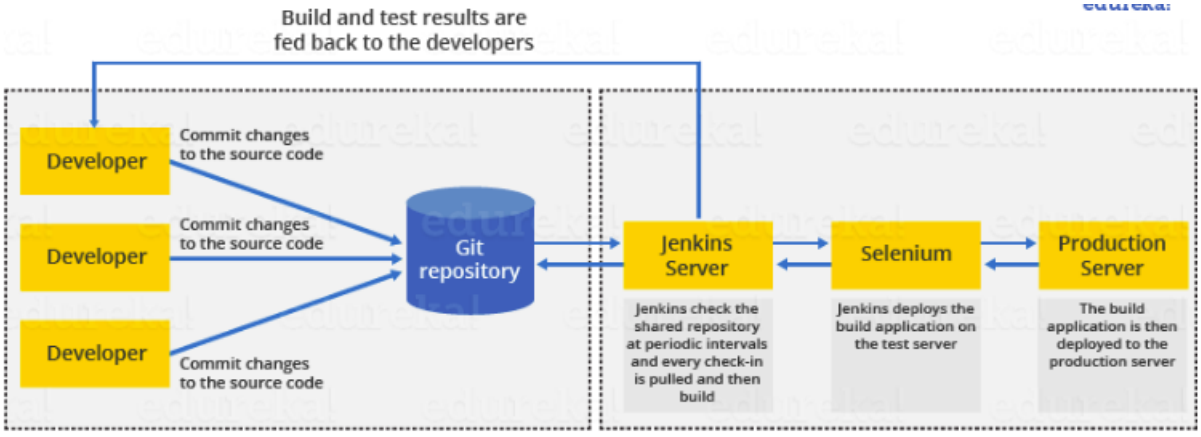
- Os desenvolvedores precisam esperar até que o software completo seja desenvolvido para os resultados do teste.
- Os resultados do teste mostrem vários erros.
- Retarda o processo de entrega do software.

[3] <https://www.edureka.co/blog/what-is-jenkins/>

- Faltava feedback contínuo referente a problemas de codificação ou arquitetura, falhas de compilação, status de teste e upload de arquivos.

Dados os problemas citados, não apenas o processo de entrega do software ficou mais lento, mas também a qualidade do software diminuiu. Com isso, surgiu a necessidade de existir um sistema que os desenvolvedores pudessem adicionar continuamente uma construção e testar toas as suas alterações feitas no código-fonte. O Jenkins é uma ferramenta criada para esse fim (Saurabh, 2019). Ela possui o seguinte fluxograma representado na figura 4.

Figura 4 – Fluxo genérico da CI usando Jenkins



2.3.4 CI/CD

CI/CD é um método para entregar aplicações com frequência aos clientes. Para isso é aplicada automação nas etapas de desenvolvimento de aplicações. Os principais conceitos atribuídos a esse método são a integração, entrega e implantação contínuas.

O CI/CD aplica monitoramento e automação contínuos a todo ciclo de vida dos softwares, incluindo as etapas de teste e integração, entrega e implantação. Juntas, essas práticas são chamadas de *"pipeline de CI/CD"*.

O acrônimo CI se refere à Integração contínua, que é o processo de automação para desenvolvedores. A CI é bem sucedida quando tem novas mudanças no código que são desenvolvidas, testadas e consolidadas em um repositório compartilhado. Vários testes automatizados, geralmente de unidade e integração, são feitos para garantir que as mudanças não corromperam a aplicação. Basicamente, tudo é testado, incluindo classes, funções e diferentes módulos que formam toda a aplicação. Em caso de conflito entre os códigos novos e existentes, a CI facilita a correção desses bugs com rapidez e frequência.

O CD se refere a entrega contínua, ou implantação contínua. Ela representa as mudanças feitas pelo desenvolvedor em uma aplicação, que são automaticamente testadas contra bugs e carregadas em um repositório ou conteiner. O objetivo da entrega contínua é garantir uma base de códigos que esteja sempre pronta para implantação em um ambiente de produção.

Outro significado para CD, se refere ao lançamento automático das mudanças feitas por um desenvolvedor à produção, onde podem ser usados pelos cliente. Assim, a equipe de operações não fica sobrecarregada, por não precisar implementar manualmente.

A parte final do CI/CD é a implantação contínua, que é um complemento da entrega contínua, que automatiza o lançamento de compilações prontas para produção em um repositório de códigos. A implantação contínua automatiza o lançamento de uma aplicação para a produção.

2.3.5 Gitlab CI/CD

https://docs.gitlab.com/ee/ci/introduction/

https://about.gitlab.com/blog/2019/10/18/better-devops-with-gitlab-ci-cd/

https://www.concrete.com.br/2018/02/01/conheca-o-gitlab-ci/

Referências:

(LOURENÇO, 2018)

(GitLab Docs, 2019)

(BUCHANAN, 2019)

Conclusão

2.4 Vantagens

- Entrega rápida de atualizações de software para o usuário final
- Processos automatizados de teste de novos releases do software. Com isso, evita erros técnicos dos processos de compilação, testes e implantação, muito comum quando há necessidade de troca de funcionários em uma empresa, ou quando o técnico de infraestrutura não é experiente.
- Com uma entrega mais ágil, evita problemas que acontecem quando uma equipe de desenvolvimento trabalha por muito tempo em um caminho diferente do qual o software deve seguir. Esse ponto é favorecido pelo fato de o cliente sempre ter contato com novas atualizações.
- Quando utilizado *containers*, padroniza e isola o ambiente de execução, de forma a evitar conflitos com outras aplicações ou incompatibilidades com sistemas

2.5 Desvantagens

A Curva de aprendizado é uma desvantagem, pois é preciso estudar e entender como funcionam as ferramentas de automação oferecidas.

2.6 Ponto de vista

O devops é extremamente importante para startups que precisam reduzir seus gastos

Referências

BARBOSA, D. A. M. *Importância de DevOps para as organizações - Blog da Cedro*. 2019. Disponível em: <https://blog.cedrotech.com/importancia-de-devops-para-as-organizacoes/>. Citado nas páginas 3 e 4.

BUCHANAN, C. *Unlock better DevOps with GitLab CI/CD | GitLab*. 2019. Disponível em: <https://about.gitlab.com/blog/2019/10/18/better-devops-with-gitlab-ci-cd/>. Citado na página 11.

Cloud google. *What are Containers and their benefits | Google Cloud*. 2018. Disponível em: <https://cloud.google.com/containers/>. Citado nas páginas 7 e 8.

COIS, C. A.; YANKEL, J.; CONNELL, A. Modern DevOps: Optimizing software development through effective system interactions. In: *2014 IEEE International Professional Communication Conference (IPCC)*. IEEE, 2014. p. 1–7. ISBN 978-1-4799-3749-3. Disponível em: <http://ieeexplore.ieee.org/document/7020388/>. Citado na página 5.

DELFINO, D. *O que é DevOps? Chegou a hora de entender esse conceito inovador!* 2018. Disponível em: <https://vertigo.com.br/o-que-e-devops/>. Citado na página 3.

Docker. *What is a Container? | Docker*. 2019. Disponível em: <https://www.docker.com/resources/what-container>. Citado nas páginas 7 e 8.

GitLab Docs. *GitLab CI/CD | GitLab*. 2019. Disponível em: <https://docs.gitlab.com/ee/ci/>. Citado na página 11.

GUIMARÃES, A. L. *O que é e qual é a importância do DevOps?* 2019. Disponível em: <https://configr.com/blog/o-que-e-devops/>. Citado nas páginas 3 e 4.

Kubernetes. *Orquestração de contêiner pronto para produção - Kubernetes*. 2019. Disponível em: <https://kubernetes.io/pt/>. Citado na página 8.

LOURENÇO, L. *Concrete | Conheça o Gitlab CI*. 2018. Disponível em: <https://www.concrete.com.br/2018/02/01/conheca-o-gitlab-ci/>. Citado na página 11.

PRESSMAN, R. S. *Software engineering : a practitioner's approach*. [S.l.]: McGraw-Hill, 2005. 880 p. ISBN 007301933X. Citado na página 7.

Saurabh. *O que é Jenkins? | Jenkins para integração contínua | Edureka*. 2019. Disponível em: <https://www.edureka.co/blog/what-is-jenkins/>. Citado nas páginas 9 e 10.

Sergio. *DevOps: Conceitos e características*. 2015. Disponível em: <https://www.devmedia.com.br/devops-conceitos-e-caracteristicas/32030>. Citado na página 3.