INTERFACE TECNOLÓGICA

O PROCESSO DE DEPLOY AUTOMÁTICO EM AMBIENTES DE COMPUTAÇÃO NA NUVEM: exemplo de utilização da plataforma Jenkins

THE PROCESS OF AUTOMATIC DEPLOYMENT IN CLOUD COMPUTING ENVIRONMENTS: example using the Jenkins platform

Klerison Emmanuel Lopes – klerisonpe@gmail.com
Ronaldo Ribeiro de Campos – ronaldo.campos@fatectq.edu.br
Erick Eduardo Petrucelli – erick.petrucelli@fatectq.edu.br
Gustavo Henrique Soriano – sorianox2013@gmail.com
Faculdade de Tecnologia de Taquaritinga (FATEC) – SP – Brasil

RESUMO

As características do processo de desenvolvimento de *software* vêm sendo aprimoradas há mais de duzentos anos. Desde seu início, com trechos de algoritmos escritos à mão e incorporações de melhorias como as realizadas por John Von Neumann no período pós segunda guerra mundial, novas tecnologias e métodos de programação estão evoluindo. Para acompanhar esta evolução também foi necessário a criação de plataformas de desenvolvimento que atendessem a demanda e a interpretação das novas linguagens. O mecanismo atual de desenvolvimento sugere que essa evolução passe de plataformas localizadas em ambientes internos para os ambientes integrados à tecnologia de computação na nuvem. Junto com essa tecnologia encontram-se diferentes ferramentas para suporte ao próprio desenvolvimento do software integrado à essa proposta. Este estudo aborda, por meio da revisão bibliográfica, diferentes ferramentas de apoio à atividade de desenvolvimento de software na nuvem. Além disso foi conduzido, por meio da metodologia descritiva, o exemplo de uso da ferramenta Jenkins. Como resultados principais destacam-se as possibilidades do uso das vantagens do ambiente local e na nuvem e agendamento de atividades.

Palavras-chave: Computação na nuvem. DevOps. Desenvolvimento. Deploy. Jenkins.

ABSTRACT

The characteristics of the software development process have been improved over two hundred years. Since its inception, with extracts from handwritten algorithms and incorporations of improvements such as those made by John Von Neumann in the post WWII period, new technologies and programming methods are evolving. To follow this evolution, it was also necessary to create development platforms that would meet the demand and interpretation of the new languages. The current development mechanism suggests that such evolution shifts from intranet-based platforms to cloud computing-embedded environments. Along with this technology are different tools to support the very development of software integrated with this proposal. This study approaches, through the bibliographic review, different tools to support the software development activity in the cloud. In addition, an

example of using the Jenkins tool was conducted using the descriptive methodology. The main results highlight the possibility of using the advantages of local and cloud environment and scheduling of activities.

Keywords: Cloud computing. DevOps. Development. Deploy. Jenkins.

1 INTRODUÇÃO

O desenvolvimento de *software* tem se tornado cada vez mais comum na sociedade moderna, pois desenvolver sistemas não é novidade, uma vez que o primeiro código feito pela humanidade, de acordo com Obregon, Costa Filho (2015), foram pequenos algoritmos escritos em 1842 pela condessa Ada Lovelace.

Porém na década de sessenta foi definido que para ser considerado um *software* é preciso conter instruções de linguagens de programação, armazenadas em memória eletrônica e executadas por microprocessadores, assim sendo o primeiro escrito por John Von Neumann afirma Macrae (2016). Isto está em crescente mudança e se adequando a novas abordagens, metodologias e técnicas. O novo mercado de *software* sugere que assim como as linguagens utilizadas na programação, o ambiente de desenvolvimento também evoluiu. Todo processo de desenvolvimento de *software* passa por etapas antes de chegar ao usuário final.

Cada etapa do processo não envolve somente o desenvolvedor e a linguagem, mas também o ambiente aonde os processos estão ocorrendo, de forma que um mesmo produto de *software* assume comportamentos diferentes de acordo com seu ambiente de desenvolvimento. Isso porque durante seu desenvolvimento um sistema precisa passar por uma série de testes, mudanças e adequações. Erros precisam serem tratados e cada ação do usuário precisa ter como resposta uma informação útil, cada *feedback* do sistema é importante para o usuário e sempre levando em consideração qual o nível de familiaridade que determinado usuário possui com o sistema.

Para que um sistema seja implementado e chegue até seu usuário final, metodologias mais clássicas sugerem que o ambiente de desenvolvimento seja a própria máquina do desenvolvedor que também pode ser descrita como ambiente local, ou seja, todo processo de desenvolvimento ocorre diretamente no computador dos programadores responsáveis. Antes de qualquer processo de codificação, ferramentas, dependências que são pacotes oficias ou de terceiros que são essenciais para o ideal funcionamento de aplicações e adequações precisam ser realizadas na máquina do desenvolvedor, o que torna muito comum a prática de manter a

máquina sempre com ferramentas instaladas e a disposição de desenvolvimentos futuros. Esta abordagem é a mais comum e ainda possui muita força em relação a como os desenvolvedores se organizam para desenvolver seus produtos.

A ideia de manter um ambiente local sempre atualizado e preparado para o desenvolvimento de sistemas é bem aceita pela comunidade atual, mas traz como principal desvantagem a mesma ideia de que a maior dependência ainda nos dias atuais, é o desafio de manter um sistema operacional estável, bem configurado, o que além de demandar esforços, ainda sim envolvem perigos e fatores a qual todos estão sujeitos na sua rotina de utilização de sistemas, a segurança.

Outro fator importante quanto ao desenvolvimento local, é a responsabilidade de manter versões compatíveis com os sistemas já implementados, isso é comum e ocorre que dependendo da plataforma de implementação do sistema, ou até mesmo a linguagem de

programação, alguns recursos só funcionam com determinadas versões, o que torna a manutenção do ambiente mais desafiadora e comprometida. No caso de aplicações web que interagem diretamente com o sistema operacional no servidor, o desafio é ainda mais crescente, isso porque o *software* funcionando local, mesmo que devidamente configurado, não significa que terá o mesmo comportamento no servidor.

Boa parte dos desenvolvedores de sistemas de informação, sugerem que utilizar o mesmo sistema operacional do servidor, pode ser uma forma de evitar problemas inesperados. Tecnologias como Docker, vem para entre outros fatores, resolver problemas como o apresentado anteriormente, tendo em vista que uma vez que o sistema é configurado local tendo como base um container Docker, o próprio Docker neste mesmo container e configuração, garantem a funcionalidade, não dependendo assim, diretamente do sistema operacional do ambiente local, para com o ambiente servidor, ou ambiente de produção, como é comum ser chamado.

A forma tradicional de manter a própria máquina local sempre atualizada e configurada, como os programadores comumente se organizam para resolver os problemas, tem funcionado muito bem, e levando em consideração tecnologias como ditas anteriormente, foi possível chegar mais próximo de uma solução viável e menos problemática.

O objetivo deste estudo foi apresentar um panorama das possibilidades de uso de ambientes em nuvem para realização de deploy. Para o desenvolvimento do estudo foi realizada a revisão bibliográfica em conjunto com a aplicação da metodologia descritiva por meio de um exemplo de uso da ferramenta Jenkins.

A estrutura deste trabalho foi elaborada apresentando-se a introdução, um capítulo abordando a descrição de plataformas em nuvem para deploy, seguido de um capítulo relatando a metodologia DevOps. Na sequencia, apresenta-se o exemplo de uso, baseado no servidor Jenkins. Por fim, seguem-se as discussões e conclusão do estudo.

2 AMBIENTES DE DESENVOLVIMENTO

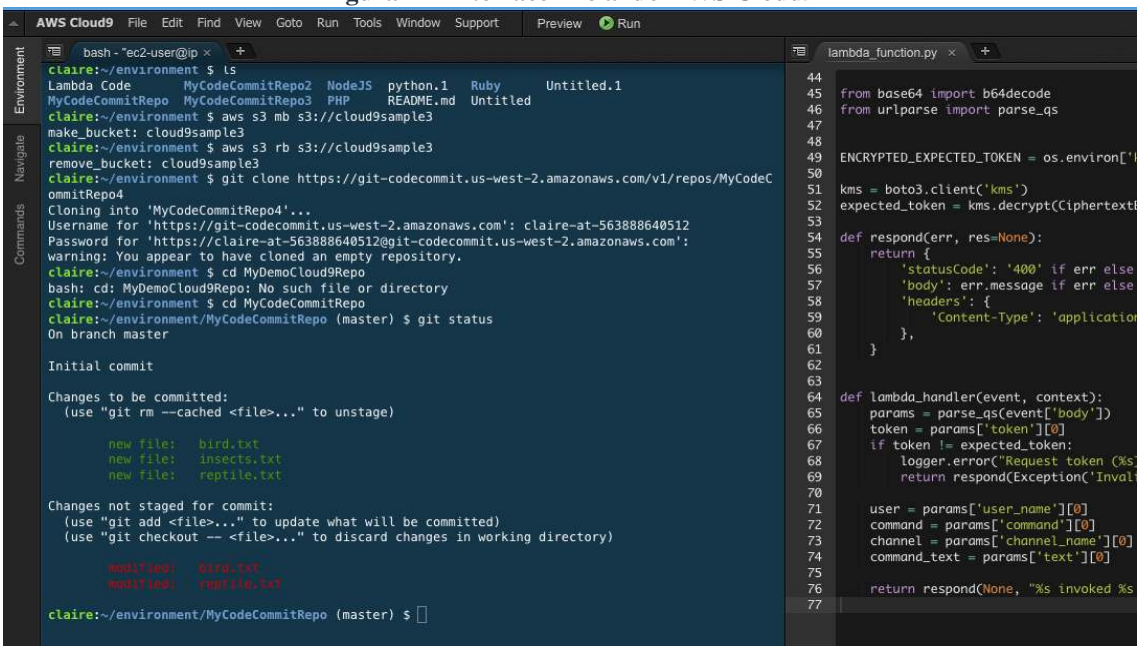
Dentre muitas plataformas disponíveis, como DigitalOcean e GoogleCloud atualmente na nuvem, em sua grande maioria fazem uso do recurso de *deploy* que é o ato de publicar uma nova versão de um *software* diretamente no servidor, facilitando e agilizando o desenvolvimento do programador, porém além de serem melhor aproveitadas elas geram alguns problemas em questão de problemas que possam a ocorrer com a conexão do computador com o servidor onde se está trabalhando. A seguir será apresentado alguns destes ambientes de desenvolvimento.

2.1 Amazon Cloud9

O AWS Cloud9 é um ambiente de desenvolvimento integrado (IDE) que permite escrever, executar e depurar código usando apenas um navegador. O ambiente inclui um editor de código, um depurador e um terminal. A ferramenta está disponível na nuvem, é flexível e pode ser utilizada juntamente com Docker, que em poucas linhas de instruções de código já estará disponível para utilização (AMAZON, 2019).

Com Cloud9 o desenvolvedor poderá depurar aplicativos tendo como dependência apenas o browser. Vale acrescentar que a IDE é compatível com as principais linguagens de programação web, como JavaScript, Python e PHP. A Figura 1 demonstra a interface inicial do Cloud9.

Figura 1 – Interface inicial do AWS Cloud9



Fonte: Amazon Web Service (2019)

Ele possui uma funcionalidade que é permitida a codificação do mesmo projeto em várias localidades simultâneas, e vários programadores mexendo no mesmo código.

2.1.1 Pontos Negativos

Por ser uma plataforma disponível apenas na web conforme diz Amazon (2019), ocorre que podem ter imprevistos diante de conexões ruins ou até mesmo a perda total dela, outro fator negativo para a IDE é o ponto onde vários programadores possuem acesso ao mesmo código e ao mesmo tempo, pois se houver discrepâncias entre ambos pode ocorrer erros de logica e sintaxes. Se houver problemas no endereçamento IP da instancia também é gerado um problema, onde a equipe de desenvolvimento perde o total acesso dela.

2.1.2 Pontos Positivos

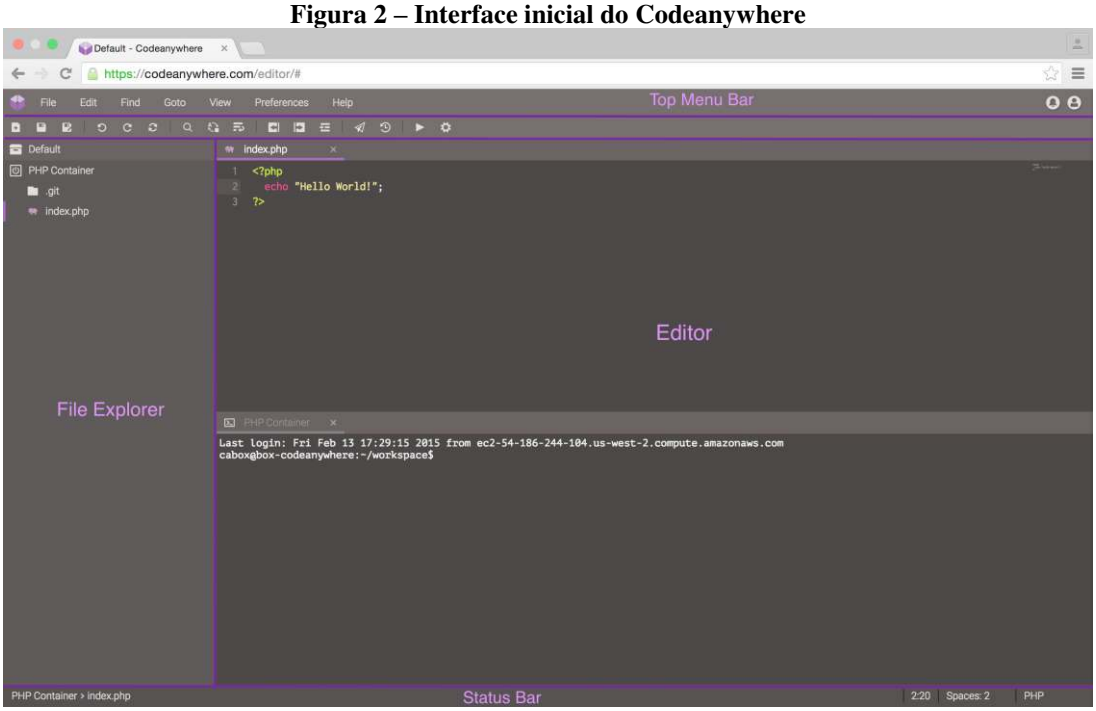
Como pontos positivos nos requisitos de avanço de desenvolvimento, pois não é necessário fazer o uso de *deploy* já que o desenvolvimento está feito diretamente no servidor. Também é sucessível apresentar uma funcionalidade que poucos ambientes de desenvolvimento possuem, segundo Amazon (2019) um editor de fotografias, assim ao se desenvolver uma página web é permitido que se edite algumas imagens se necessário.

2.2 Codeanywhere

O Codeanywhere, de acordo com a Codeanywhere (2019), é um *software* que atua em Cross Plataform Cloud IDE, ou seja um *software* com os mesmos recursos da sua versão desktop, porém com recursos adicionais que só são disponíveis em seu aplicativo na nuvem.

A principal proposta do Codeanywhere é a possibilidade de codificar diretamente na nuvem, sem necessidade de instalar ferramentas e dependências locais, e funciona independentemente de plataformas e dispositivos, basta tão somente uma conexão com Internet. Pode-se acessar devidos projetos e códigos de um smartphone até uma TV Smart. A Figura 2 representa a tela inicial do Codeanywhere.

Figura 2 – Interface inicial do Codeanywhere



Fonte: Codeanywhere (2019)

A IDE pode ser utilizada em uma versão gratuita ou em planos mensais com algumas variações nos valores, onde cada valor oferecido conta com mais ou menos recursos que estarão disponíveis na ferramenta (CODEANYWHERE, 2019).

2.2.1 Pontos Negativos

Assim como outras plataformas de desenvolvimento em nuvem é extremamente negativo a visão de apenas ter acesso ao ambiente se tivermos uma conexão com a Internet, pois se houver perda de dados ou problemas na rede não é possível realizar o acesso e assim manter os projetos em andamentos.

2.2.2 Pontos Positivos

Porém de acordo com a Codeanywhere (2019), diferentemente de outras IDES como Cloud9, ele possui uma versão local que pode ser instalada e utilizada em modo *offline*, assim se ocorrer problemas de conexão ou perda de dados ainda sim é possível continuar o projeto, mas essa versão contém algumas restrições de ferramentas que só é disponível na versão *online*.

3 DEVOPS

De acordo com Kim *et al* (2018), DevOps é um movimento cultural que vem para acabar com alguns problemas que muitas empresas possuem em seus departamentos de TI. Realizar a entrega de um *software* em produção de longe é uma das tarefas mais complexas a qual os programadores tem dificuldade, principalmente por terem que realizar testes entre equipes de desenvolvimento. O DevOps vem com a proposta de realizar um foco em automação, colaboração compartilhamento de ferramentas e de conhecimento, mostrando aos engenheiros de *software* que é possível realizar uma interação entre as equipes.

O termo é um jargão muito comum nos meios da tecnologia de informação com ênfase no desenvolvimento de *software*, pois se trata no processo de migração do sistema partindo do ambiente local para o ambiente de produção, ou seja, o ambiente onde o sistema estará disponível comercialmente. Antes do *deploy*, é comum um sistema passar por uma bateria de testes e integração contínua, de forma que a evolução e detecção de problema e melhorias está diretamente ligada com esta etapa tão importante quanto a arte de codificar um sistema.

Todo o processo de migração entre ambientes pode ser automatizado do desenvolvimento ao produto final, através de integração contínua, sem que haja muitas interferências humanas. Para tal, surgem metodologias como DevOps, que implementam

recursos e abordagens com ênfase na infraestrutura e autonomia do ciclo de vida de *software*, desde o momento de sua concepção, manutenção até sua evolução e contínua integração, bem como contínuo desenvolvimento (DAVIS; DANIELS, 2016).

O processo de *deploy* também está ligado à boa integração de recursos de infraestrutura, o que normalmente envolve novas equipes, tecnologias e procedimentos, que com o passar do tempo e de acordo com as metodologias tradicionais, trouxe novos problemas e desafios entre o time de desenvolvimento e a equipe responsável por definir a infraestrutura. Como envolve mais de uma equipe, quando surge um problema de adequação do sistema, a fase de migração pode deixar a desejar, devido a configurações ou recursos tecnológicos, que em ocasiões comuns faz com que as equipes se divirjam, não sabendo de onde se origina o problema na integração do sistema, se na parte programática e lógica do código ou nas ferramentas que demandam configuração do ambiente, que compete ao time de infraestrutura no servidor com foco em aplicações *Web* (HTTERMANN, 2012).

DevOps vem para resolver esse problema, pois sua principal meta é a melhoria contínua da integração de ambiente de *software*, envolvendo equipes que compões partes distintas do sistema, pois a metodologia organiza a integração, coordena, define recursos e tecnologias que auxiliam e reduzem os desafios já mencionados.

4 EXEMPLO DE APLICAÇÃO

Neste presente estudo de caso foi realizado a construção de uma aplicação em uma Máquina virtual disponibilizada na AWS, contendo acesso a plataforma Jenkins, um *software* livre que possibilita a integração continua para um *software* que está em desenvolvimento.

> A integração contínua é uma prática de desenvolvimento de software de DevOps em que os desenvolvedores, com frequência, juntam suas alterações de código em um repositório central. Depois disso, criações e testes são executados. Geralmente, a integração contínua se refere ao estágio de criação ou integração do processo de lançamento de software, além de originar um componente de automação (ex.: uma CI ou serviço de criação) e um componente cultural (ex.: aprender a integrar com frequência). Os principais objetivos da integração contínua são encontrar e investigar bugs mais rapidamente, melhorar a qualidade do software e reduzir o tempo que leva para validar e lançar novas atualizações de software. (Jenkins, Amazon Web Service. 2019)

Conforme a Amazon Web Service(2019) ele vem para resolver alguns gargalos que as equipes de programadores encontram durante o processo de produção de um *software*, pois se pretende o aceleramento do ciclo de vida do desenvolvimento do *software*. O Jenkins traz algumas vantagens nos processos de integrar complicações, implantações e testes em variados ambientes. Ele é o modelo perfeito para metodologias ágeis de desenvolvimento e operações como o DevOps. Ele foi escolhido por ser a plataforma onde o autor tem a disponibilidade de uso para a demonstração.

Para o versionamento de códigos e *backup* das alterações foi utilizado o GitHub, uma plataforma de hospedagem de códigos fonte com controle de versão, ele tem a sua principal função além de hospedagem de códigos a contribuição, onde qualquer programador pode ter acesso a outros projetos e assim contribuírem com programadores alheios.

> A principal diferença entre o Git e qualquer outro VCS (Subversion e amigos incluídos) é a forma como o Git pensa sobre seus dados. Conceitualmente, a maioria dos outros sistemas armazena informações como uma lista de alterações baseadas em arquivos. Esses outros sistemas (CVS, Subversion, Perforce, Bazaar e assim por diante) pensam nas informações que armazenam como um conjunto de arquivos e as alterações feitas em cada arquivo ao longo do tempo (isso é comumente descrito como controle de versão baseado em delta). (traduzido de CHACON; STRAUB, 2014, p. 32).

4.1 Construção do Servidor Jenkins em uma Instância Amazon EC2

O Jenkins é um programa autônomo baseado em Java, pronto para funcionar em *out-of-the-box*, ou seja, o ato de funcionar após ou sem a necessidade de uma instalação ou configurações adicionais, com pacotes para Windows, Mac OS X e outros sistemas operacionais semelhantes a Unix. Como um servidor de automação extensível, o Jenkins pode ser usado como um servidor de CI (*continuous integration*, ou integração continua) simples ou transformado no *hub* de entrega contínua para qualquer projeto. Para se iniciar a construção do servidor foram utilizados recursos da Amazon Elastic Compute Cloud (EC2) com a configuração de *hardware* simples, contendo Ubuntu 18.04, 1GB de memória RAM e 25GB de Armazenamento.

Para efetivamente estar com o ambiente preparado foi necessário configurar alguns pré-requisitos, primeiro foi criado um grupo de segurança para fazer o controle de portas da instancia, assim liberando o acesso a porta 8080 que é de uso padrão para o Jenkins. Logo

após a liberação da porta foi feito a instalação do Java pois é de uso necessário uso de alguns recursos desta ferramenta. Para isso foi executado o comando: `sudo apt install openjdk-8-jdk`. Executamos um outro comando para verificar se foi instalado corretamente e também verificamos se a versão está correta, a saída que obtida foi:

- `openjdk version "1.8.0_191" OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.18.04.1-b12) OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)`

Após toda essa base instalada foi definido de fato iniciar a instalação do Jenkins em nossa instância, primeiramente foi adicionado o repositório de dados da aplicação através do WGET que é um programa que permite o download de dados da web, então executamos `sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'`, e então atualizamos as listas de pacotes com `sudo apt upgrade && sudo apt update –y`. Depois de termos feito esses dois passos, então foi iniciado a instalação do programa com `sudo apt install jenkins`.

Então, foram executados alguns testes para a verificação e iniciação do Jenkins no sistema, com o comando `sudo systemctl enable jenkins` para que o programa seja iniciado na inicialização do sistema e `sudo systemctl start jenkins` para iniciar de fato. Para testar, foi utilizado `systemctl status jenkins`, retornando o seguinte:

- `jenkins.service - LSB: Start Jenkins at boot time Loaded: loaded (/etc/init.d/jenkins; generated) Active: active (exited) since Wed 2018-08-22 13:03:08 PDT; 2min 16s ago Docs: man:systemd-sysv-generator(8) Tasks: 0 (limit: 2319) CGroup: /system.slice/jenkins.service`

O Jenkins só é acessado pelo IP do servidor e por sua respectiva porta que foi liberada, neste caso foi usado a porta padrão 8080. Para fazer um pequeno teste foi preciso utilizar uma máquina terceira e digitar em qualquer navegador de Internet: http://18.244.168.212:8080.

Depois de instalado e testado foi realizado as devidas configurações, por padrão o nome de usuário e senha são *admin:admin*, para a alteração da senha root do *software* foi preciso ir até as configurações e localizar o campo "*Password*" é assim pode ser feito a alteração necessária. Também foi preciso alterar o caminho do Java para *usr/lib/jvm*, dentro da aplicação, em sua aba *Manage Jenkins > Global Tool Configuration > JDK*.

Então foi possível criar um arquivo teste para verificar se está tudo corretamente instalado e configurado dentro destas IDE a partir das seguintes ações: 1 – Criar novo item; 2

Digitar um nome para o novo item; 3 – Digitar dentro do arquivo `echo "Bem-vindo Ao Jenkins Teste"`; 4 – Salve o arquivo; 5 – Execute o arquivo; 6 – A saída mostrada no browser é exatamente o que foi escrito no arquivo "Bem-vindo ao Jenkins Teste".

Depois de todo esse passo a passo de como fazer as instalações de dependências e realizar configurações, foram executadas as configurações no GitHub para que pudesse ser feito o versionamento para o servidor, realizando assim o *deploy* automático.

A conexão com o GitHub foi feita através do *plug-in* chamado GitHub Authentication Plugin. Com este recurso disponível, após a instalação, só é preciso passar o endereço local do repositório, por exemplo: https://github.com/nomeUsuario/nomeRepositorio. Também é preciso entrar com o usuário e senha para autenticar. Assim é feita a configuração para trabalhar com a integração contínua, através do Jenkins e GitHub.

5 RESULTADOS E DISCUSSÃO

De acordo com todas as características encontradas e analisadas, foi possível elaborar o Quadro 1, contendo um comparativo entre todas as plataformas apresentadas, destacando características semelhantes entre as IDES.

Quadro 1 - Comparativo entre plataformas de desenvolvimento

	Gratuito para uso	Possui versão local, sem a necessidade de Internet	Somente *online*, para computação em nuvem	Codificação em tempo real com outros programadores	Oferece *deploy* automatizado	Código aberto
Amazon Cloud 9			X	X		
CodeAnywhere		X		X		
Jenkins	X	X			X	X

Fonte: adaptado de Amazon Cloud9 (2019); CodeAnywhere(2019); Amazon Jenkins (2019).

6 CONCLUSÃO

Devido à vasta possibilidade de utilização de ferramentas disponíveis na nuvem, a evolução das ferramentas de infraestrutura e a forma como são empregadas na rotina de profissionais de TI, as mudanças situadas no meio informacional afetaram diretamente a forma como é feita a integração de sistemas em nuvem, bem como a forma de automatizar o

processo de *deploy*, passando por processos de integração que continuamente submetem o software a testes automatizados, pré-análise e, por fim, a etapa de produção.

Assim é possível concluir que com as inovações veio o grande surgimento de tecnologias, oportunidades e um novo foco para o profissional de infra, mas também novos desafios, novos testes de segurança, bem como o dinamismo das ferramentas em nuvem e a forma como elas se alteram constantemente.

O desenvolvimento de um sistema de qualidade sempre será um enorme desafio para os programadores, com isso foi apresentado a metodologia DevOps, que veio com a proposta de integrar os processos de criação e unir as equipes para se obter um resultado claro e com agilidade. Essa prática é possível graças a computação em nuvem, permitindo a interação contínua com ferramentas de agrupamento de equipes como o Amazon Cloud9, porém não se é muito confiável pois sem conexão com a rede essa ferramenta fica totalmente inutilizada. A integração continua vem para os recursos chaves em DevOps para agilizar ainda mais o desenvolvimento dos *softwares*, com o Jenkins isso é possível a cada nova atualização lançada no repositório de origem do sistema.

REFERÊNCIAS

AWS Cloud9 – **AWS Cloud9** Disponível em: <https://aws.amazon.com/pt/cloud9/>. Acesso em: 13 mar. 2019.

CHACON, Scott; STRAUB, Ben. **Pro Git**: Everything you need to know about git. 2. ed. New York: Apress, 2014. 419 p.

CODEANYWHERE. **CodeAnywhere**. Disponível em: <https://docs.codeanywhere.com/#addons>. Acesso em: 17 mar. 2019.

CONFIGURE um servidor de compilações Jenkins. 2019. Disponível em: <https://aws.amazon.com/pt/getting-started/projects/setup-jenkins-build-server/>. Acesso em: 11 abr. 2019.

DAVIS, Jennifer; DANIELS, Ryan. **Effective DevOps**: Building a Culture of Collaboration, Affinity, and Tooling at Scale. [s.l]: O'reilly Media, 2016. 410 p.

HTTERMANN, Michael. **DevOps for Developers**. [s.l]: Apress, 2012. 196 p.

KIM, Gene; HUMBLE, Jez; DEBOIS, Patrick; WILLIS, John. **Manual de Devops. Como Obter Agilidade, Confiabilidade e Segurança em Organizações Tecnológicas.** Rio de Janeiro: Alta Books, 2018. 464 p.

MACRAE, Norman. **JOHN VON NEUMANN.** [canada]: Plunkett Lake Press, 2016.

OBREGON, Rosane de F. A.; COSTA FILHO, Nilson Sá. **DESENVOLVIMENTO DE** SOFTWARE **BASEADO EM REALIDADE AUMENTADA PARA PROCESSOS DE APRENDIZAGEM.** São Luis: Conahpa, 2015. 11 p.

SATO, Danilo. **DevOps na prática: Entrega de software confiável e automatizada.** Casa do Código, 2014. 257 p.