CENTRO UNIVERSITÁRIO 7 DE SETEMBRO

CENTRO DE CIÊNCIAS TECNOLÓGICAS

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

ISAC FABRICIO MAIA NETO

RETROFIT DO PROCESSO DE *BUILD* E *DEPLOY* DE APLICAÇÕES JAVA

FORTALEZA – CEARÁ

2023

ISAC FABRICIO MAIA NETO

RETROFIT DO PROCESSO DE *BUILD* E *DEPLOY* DE APLICAÇÕES JAVA

Monografia apresentada ao CENTRO UNIVERSITÁRIO 7 DE SETEMBRO como requisito parcial para obtenção do título de bacharel em Sistemas de Informação.

Orientador: Dr. Francisco Cristiano de França Júnior

FORTALEZA – CEARÁ

2023

ISAC FABRICIO MAIA NETO

RETROFIT DO PROCESSO DE *BUILD* E *DEPLOY* DE APLICAÇÕES JAVA

Monografia apresentada ao CENTRO UNIVERSITÁRIO 7 DE SETEMBRO como requisito parcial para obtenção do título de bacharel em Sistemas de Informação.

Aprovada em:

BANCA EXAMINADORA

Dr. Francisco Cristiano de França Júnior (Orientador)
CENTRO UNIVERSITÁRIO 7 DE SETEMBRO - UNI7

Prof. Me. Marcelo Bezerra de Alcântara
CENTRO UNIVERSITÁRIO 7 DE SETEMBRO - UNI7

Prof. Me. Alan Bessa Gomes Peixoto
CENTRO UNIVERSITÁRIO 7 DE SETEMBRO - UNI7

Dedico este trabalho primeiramente a Deus; a minha família, pelo incentivo e apoio; aos sinceros amigos, que confiaram em meu caráter humano; e em especial, a minha mãe, pela esperança depositada durante esta caminhada.

AGRADECIMENTOS

Neste momento de encerramento, não posso deixar de expressar minha profunda gratidão às pessoas que tornaram esta caminhada possível, tornando-a não apenas uma experiência acadêmica, mas também uma verdadeira caminhada de crescimento e aprendizado pessoal.

Em primeiro lugar, agradeço a Deus, que tem sido minha fonte de força, inspiração e orientação ao longo de toda a minha vida. Sua graça e misericórdia me sustentaram em todos os momentos, e por isso, sou eternamente grato.

Minha mãe, Valda Maia, sua dedicação incansável, amor incondicional e apoio constante foram os pilares que me sustentaram em todas as fases deste desafio acadêmico. Suas palavras de incentivo e o exemplo de determinação que sempre ofereceu foram uma fonte constante de inspiração.

À minha esposa, Alessandra Freitas, agradeço por seu amor, paciência e compreensão durante os momentos em que minha atenção estava inteiramente dedicada a esta monografia. Sua presença e incentivo constante tornaram essa jornada mais significativa e menos solitária.

Emanuel Maia, meu pequeno filho, você é a luz da minha vida. Seu sorriso e entusiasmo contagiantes foram a motivação que me impulsionou a superar desafios e a buscar a excelência neste trabalho. Cada conquista minha é uma expressão do meu amor e compromisso com o seu futuro.

Agradeço também a todos os amigos, familiares, colegas de estudo e a todos que, de alguma forma, contribuíram para que este projeto se tornasse realidade.

Por fim, agradeço a esta instituição de ensino, UNI7 (Centro Universitário 7 de Setembro), que proporcionou o ambiente e os recursos necessários para o desenvolvimento desta pesquisa.

Este trabalho é dedicado a todos vocês, e cada página é uma expressão da minha gratidão profunda e sincera.

Muito obrigado por fazerem parte desta jornada.

"... considerem motivo de grande alegria o fato de passarem por diversas provações, pois vocês sabem que a prova da sua fé produz perseverança. E a perseverança deve ter ação completa, a fim de que vocês sejam maduros e íntegros, sem lhes faltar coisa alguma."

(Tiago 1:2-4.)

RESUMO

Este trabalho visa apresentar de forma prática a implementação de *Continuous Integration* (CI) e *Continuous Delivery* (CD) para automatizar o processo de *build* e *deploy* de aplicação java no Sistema Fecomércio (Sesc e Senac) do estado do Ceará. Concluindo que com a utilização do *Azure Devops* para o desenvolvimento do CI e CD, utilizando o *Scrum* como metodologia ágil, é possível automatizar todo o processo de *build* e *deploy* e com isso evitar que as equipes de desenvolvimento e operações gastem tempo com atividades simples e repetitivas evitando eventuais erros manuais.

Palavras-chave: *DevOps*; *Scrum*; CI; CD;

ABSTRACT

This work aims to practically implement Continuous Integration CI and Continuous Delivery CD methodologies to automate the java application *build* and *deploy* process in the Federation of Commerce (Fecomércio System, Sesc and Senac) in the state of Ceará. Concluding that with the use of Azure Devops for CI and CD development, using Scrum as a methodology agile, it is possible to automate the entire build and deploy process and thus prevent teams from development and perations teams spend time on simple, repetitive activities, avoiding possible manual errors.

Keywords: *DevOps*; *Scrum*; CI; CD;

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CD	*Continuous Delivery*
CI	*Continuous Integration*
DASA	*DevOps Agile Skills Association*
GO	*Golang*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hyper Text Transfer Protocol Secure*
IAC	Infraestrutura como código
PaaS	*Platform as a Service*
POM	*Project Object Model*
RAM	Memória de Acesso Aleatório
SDLC	*Software Development Life Cycle*
SO	Sistema Operacional
TFVC	*Team Foundation Version Control*
TI	Tecnologia da Informação
UX	*User Experience*
VM	*Virtual Machines*

SUMÁRIO

1 INTRODUÇÃO

A grande quantidade de novas tecnologias e demandas por soluções tecnológicas, vem exigindo das empresas de Tecnologia da Informação (TI) soluções mais praticas e eficientes para lidar com um mercado cada vez mais competitivo (CAMÕES; SILVA, 2018). Por muito tempo as empresas de TI desenvolviam *software* de forma prescritivas, linear ou sequencial nos modelos de cascata, incremental e espirais. Esses modelos possuem prescrição de um conjunto de atividades, produtos e ferramentas de qualidade (PRESSMAN; MAXIM, 2021).

Com isso em mente desenvolver os projetos em menor tempo, pode influenciar o nível de concorrência das empresas. O *DevOps* vem com uma abordagem de desenvolvimento de *software* que tem como objetivo principal a comunicação, colaboração e integração entre as equipes de desenvolvimento de *software* e de a equipe de infraestrutura. A abordagem do *DevOps* tem como finalidade ampliar a quantidade de *deploy*, entregas de *software*, assim como ampliar a estabilidade e segurança dos sistemas em produção (SATO, 2014).

1.1 OBJETIVO

O trabalho tem como objetivo demonstrar e descrever a substituição das práticas tradicionais de *build* e *deploy*, no Sistema Fecomércio (Sesc e Senac) do estado do Ceará. A abordagem adotada concentra-se na aplicação prática da ferramenta *Azure DevOps* para implementar os processos de CI e CD. Com o intuito de mitigar possíveis falhas manuais e otimizar significativamente os procedimentos de construção e implementação, proporcionando uma maior agilidade ao desenvolvimento.

1.2 OBJETIVOS ESPECÍFICOS

a) Descrever o arquivo *Dockerfile*.
b) Descrever em etapas o *Azure Pipelines* de CI.
c) Descrever em etapas *Azure Pipelines* de CD.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os assuntos relevantes para o desenvolvimento deste trabalho. Serão dissertada os aspecto da cultura *pipeline*, *DevOps*, *Azure Devops*, CI, CD, *Docker*, IAC e *Scrum*.

2.1 *PIPELINE*

De acordo com RedHat (2022a), o *Pipeline* é formado por uma série de etapas que devem ser realizadas para a disponibilização de uma nova versão de um *software*.

2.2 *DEVOPS*

Faber (2020) utiliza o termo "Muro de confusão"para descrever as dificuldades entre as equipes de desenvolvimento e operações. Em que a equipe de desenvolvimento busca mudanças, diferente da equipe de operações que busca a estabilidade.

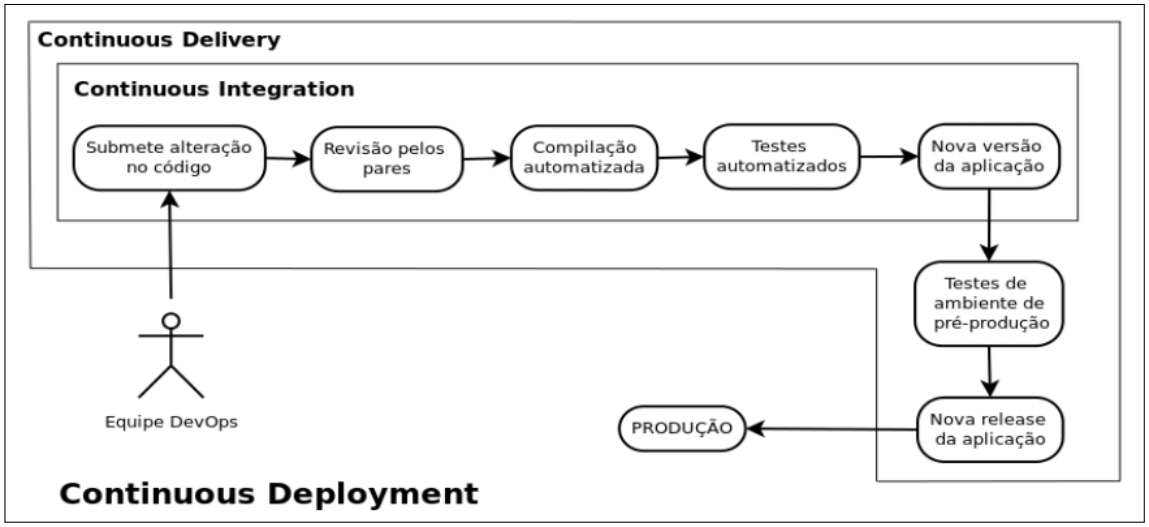
A equipe de desenvolvimento tem uma mentalidade voltada para mudanças, buscando mudanças no sistema, como a implementação de novas funcionalidades ou alteração no visual do sistema. Entretanto a equipe de operações busca estabilidade do sistema já que problemas na estabilidade e disponibilidade do sistema acabam sendo de responsabilidade da equipe de operações já o sistema está funcionando na máquina dos desenvolvedores (CAMÕES; SILVA, 2018). O *DevOps* tem como objetivo ultrapassar o "Muro de Confusão"e integrar os times. Debois (2008) apresenta o termo *DevOps*, que é a junção da palavra desenvolvimento com operações. Propondo um novo procedimento que tem como objetivo de automatizar os processos e integrar as equipes de desenvolvimento e operações.

Faria (2017) apresenta os três pipelines que viabilizam o processo de automação de algumas etapas da equipe de desenvolvimento e de operações, buscando a integração do *DevOps*. São eles CI, CD e *Continuous Deployment*. Na Figura 1 pode-se observar o esquemático da relação entre os três pipelines.

Faber (2020) descreve os seis princípios, formulado pelo *DevOps Agile Skills Association* (DASA), que cobrem a maioria das definições de *Devops*, são eles:

1. Ação centrada no cliente.
2. Criar com o fim em mente.
3. Responsabilidade de ponta a ponta.

Figura 1 – Pipelines que fazem a integração no *DevOps*



Fonte**:** (FARIA, 2017)

4. Equipes autônomas multifuncionais.
5. Melhoria continua.
6. Automatizar tudo o que puder.

2.2.1 Ação centrada no cliente

Beck *et al.* (2001) fala sobre desenvolver pequenas partes do *software* que agreguem valor para o cliente. O *Devops* compartilha do mesmo princípio, colocando o cliente como centro das operações. A equipes de *Devops* trabalham pensando em agregar valor diretamente para o cliente.

Tendo o cliente como parte principal do processo é decisivo ter a opinião dele. Com isso em mente ter um curto ciclo de *feedback* com o cliente é crucial.

2.2.2 Criar com o fim em mente

Com a separação das equipes os profissionais não trabalhavam com todo o ciclo de vida de desenvolvimento de *software*, do inglês *Software Development Life Cycle* (SDLC), em mente. Depois que o desenvolvimento criou o *software* e os testes foram realizados com sucesso a equipe finaliza suas atividades. Em seguida a equipe de operações faz a publicação em produção. Com o *Devops* todo o processo do SDLC fica como responsabilidade de todas as equipes devendo assim investir em um processo colaborativo com um serviço funcional.

2.2.3 Responsabilidade de ponta a ponta

Como todos são responsável pelo SDLC, a responsabilidade mantém as equipes satisfeitas com o trabalho que realizam. Como Herzberg (2017) apresenta na teoria de dois fatores a diferença entre motivadores e fatores de higiene para a satisfação no trabalho. No *Devops* a responsabilidade é do inicio ao fim do SDLC.

2.2.4 Equipes autônomas multifuncionais

Para os profissionais poderem atuar em todas as partes do SDLC é importante que eles possuam um um conhecimento no modelo T que é descrito por Guest (1991). O modelo T é representado por duas barras em que a vertical representa a profundidade do conhecimento específico e a barra horizontal representa o conhecimento em outras áreas em que o profissional não não é especialista.

Dessa forma é possível que o profissional resolva problemas simples fora da sua especialização. Entretanto só é possível com a presença de um profissional especialista para compartilhar experiência com os outros membros.

2.2.5 Melhoria continua

Kim, Behr e Spafford (2014) apresenta o *The Second Way*, que consiste na implementação de um fluxo constatante de *feedback*. Os *feedback* tem como objetivo a melhoria constante dos serviços da equipe. Com o monitoramento, as equipes devem estar preparadas para reagir rapidamente a testes com falhas ou alterações de desempenho.

2.2.6 Automatizar tudo o que puder

A automação de processos repetitivos é fundamental para mitigar erros e tornar o SDLC mais rápido. Com isso a equipe acaba tendo tempo livre para se concentrar na entrega de qualidade do serviço. Apesar de existir diversas ferramentas de automação no mercado com o nome *DevOps*, o *DevOps* não é somente a ferramenta. O *DevOps* não inicia com a ferramenta, e sim com a mudança de pensamento da equipe (FABER, 2020).

2.3 *AZURE DEVOPS*

De acordo com Vuppalapati *et al.* (2020) o *Azure DevOps* é um mecanismo multiplataforma de automação baseada em script desenvolvido pela *Microsoft*. A Docs (2022) fornece o *Azure DevOps* para que as equipes planejem o trabalho, colaborem no desenvolvimento de código, testem e implantem sistemas. Com o *Azure DevOps* é possível trabalhar na nuvem ou localmente.

No *Azure DevOps* existem cinco serviços integrados que podem ser utilizados de acordo com a necessidade, são eles:

1. *Azure Repos*
2. *Azure Pipelines*
    - *Azure Agents*
3. *Azure Boards*
4. *Azure Test Plans*
5. *Azure Artifacts*

2.3.1 *Azure Repos*

Proporciona o controle da versão do projeto com o repositório *Git* ou o controle de origem do código com o *Team Foundation Version Control* (TFVC). Com o *Azure Repos* é possível procurar alterações no código ao longo do tempo (DOCS, 2022).

2.3.2 *Azure Pipelines*

Proporciona controle da compilação e lançamento do projeto em ambiente de homologação ou produção. Realizando o CI e o CD. Com o *Azure Pipelines* é possivel compilar, testar e implantar aplicativos em *Python, PHP, Java, C/C++, Ruby, .NET, Android* e *iOS*. É possível implantar o código em diversos destinos. Podendo ser em *Virtual Machines* (VM), containers, plataforma locais, nuvem e *Platform as a Service* (PaaS) ou diretamente na loja de aplicativos (DOCS, 2022).

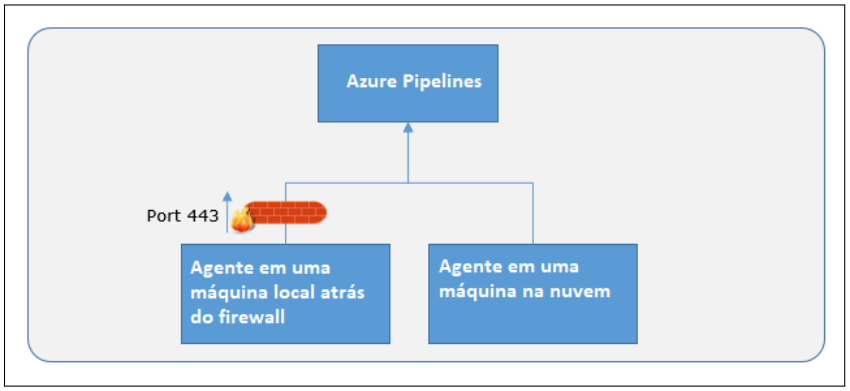
2.3.2.1 *Azure Agents*

Para possibilitar o CI e CD, o *Azure DevOps* disponibiliza o agente. O agente é uma infraestrutura de computação com *software* que é responsável por executar as tarefas do pipeline.

O agente executa uma tarefa por vez e pode ser instalado em uma VM, container ou utilizar o agente hospedado pela *Microsoft*. No agente hospedado pela *Microsoft*, tem a manutenção e atualizações feitas automaticamente. Sempre que é executado o pipeline, uma nova máquina virtual é criada para o trabalho e em seguida é descartada. Com isso qualquer alteração feita, como a verificação de algum código, não estará disponível para a próxima vez que executar o pipeline. Os agentes auto-hospedados são gerenciados e configurados pelo próprio usuário. No agente auto-hospedado o usuário possui mais liberdade para instalar *software* dependentes fundamentais para a compilação e implantação. O agente auto-hospedado pode ser instalado em máquinas *Windows, macOS, Linux* ou containers. Os caches e as configurações persistem de execução para execução, o que pode aumentar a velocidade. De acordo com a documentação da *Microsoft*, é possivel instalar mais de um agente por computador, entretanto não é recomendado. A *Microsoft* só recomenda instalar um agente por maquina para não afetar negativamente o resultado dos pipelines (DOCS, 2022).

A comunicação do agente com o *Azure Pipelines* é feita para que o agente saiba qual trabalho ele deve executar e responder com os *logs* e os status de trabalho. Por isso a comunicação sempre se inicia pelo agente. Todas a mensagens feita pelo agente são feitas por *Hypertext Transfer Protocol* (HTTP) ou *Hyper Text Transfer Protocol Secure* (HTTPS), o que vai mudar de acordo com a configuração do agente. Esse modelo de configuração *pull* permite que o agente seja configurado em diferentes topologias, conforme motra a Figura 2.

Figura 2 – Comunicação do agente com o *Azure Pipelines*.



Fonte**:** (DOCS, 2022), modificado pelo autor.

A comunicação entre o agente e o *Azure pipelines* segue o seguinte padrão:

1. O usuário faz o registro do agente, para isso ele precisa ser o administrador do *pool* de agentes. A identificação do administrador do *pool* de agentes só é necessária apenas no momento em que o registro é feito e não persiste no agente, nem é usada em qualquer

comunicação extra entre o *Azure pipelines* e o agente. Com registro terminado, o agente baixa o *token* de autenticação do ouvinte e o usa para a fila de trabalho.

2. O agente fica na espera de um solicitação de trabalho do *Azure pipelines* usando a pesquisa longa do HTTP, Quando existe um novo trabalho, o agente utiliza o *token* de autenticação para pegar o trabalho.
3. Com a conclusão do trabalho, o agente descarta o *token* especifico do trabalho e volta a verificar uma nova solicitação de trabalho utilizando o *token* principal.

A comunicação entre o *Azure pipelines* e o agente é protegida por criptografia assimétrica. Cada agente possui um par de chaves publica-privada. A chave publica é trocada com o servidor durante o registro. Com isso o servidor utiliza a chave pública para criptografar o conteúdo do material. Para conseguir ler os dados o agente descriptografa o conteúdo do trabalho utilizando a sua chave privada.

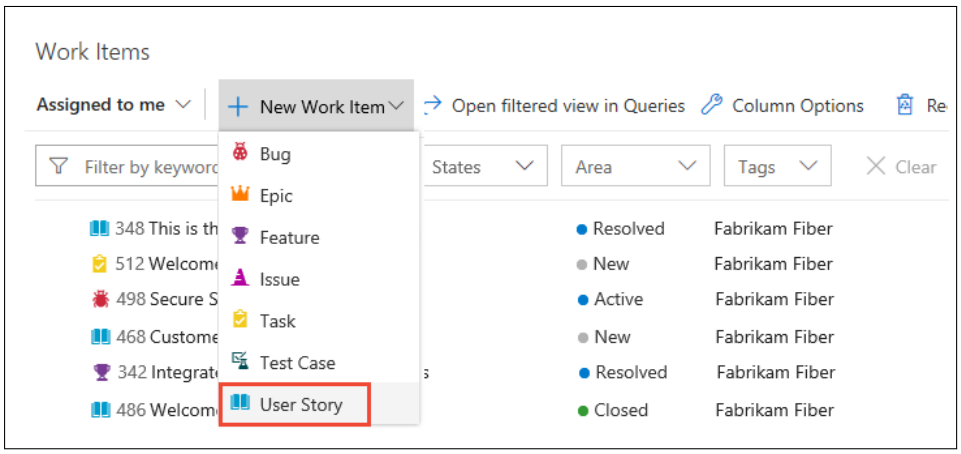
2.3.3 *Azure Boards*

Exibe ferramentas ágeis para dar suporte ao planejamento do projeto, rastreamento de trabalho e defeitos de código usando métodos *kanban* e *Scrum*. Com o *Azure Boards* é possível acompanhar o trabalho, problemas e defeitos no código do projeto. Podendo filtrar o histórico de um colaborador específico, *Bugs*, recursos e épicos. Cada item no *Azure Boards* possui campos para histórico, *links*, anexos e discussão. Na parte de controle de *Deployment , Development* e *Related Work* possuem suporte para rastreamento de quando o código é disponibilizado ou alterado e as relações entre os itens de trabalho. É disponibilizado um *hub* de itens de trabalho para localizar os itens de trabalhos relacionados ao usuário. Na Figura 3 é possível visualizar a lista de itens disponibilizados pelo *Azure Boards* para filtrar e facilitar a visualização do usuário (DOCS, 2022).

2.3.4 *Azure Test Plans*

Exibe ferramentas para testar os projetos, com testes manuais, testes unitarios, testes contínuos, testes manuais planejados, testes de aceitação do usuário, testes exploratórios e coleta de *feedback* das partes interessadas. No teste de aceitação do usuário, são realizados por usuários responsáveis por verificar se o produto atende aos requisitos do cliente. No testes exploratório, são realizados por pessoas da equipe de desenvolvedore, *User Experience* (UX) e proprietários do produto. Explorando o software sem usar planos de testes ou suítes de teste. No *feedback*

Figura 3 – *Hub* de itens de trabalho.



Fonte**:** (DOCS, 2022)

das partes interessadas, são testes realizados por pessoas de *marketing* e vendas. Os testes automatizados são integrados ao *Azure Pipelines* para possibilitar o CI e o CD. Com isso é possível adicionar planos de testes aos *pipelines* de CI e o CD. Os *pipelines* podem capturar e publicar os resultados dos teste, revisando por meio de relatórios de progresso integrados e relatórios de teste de *pipelines*. Na rastreabilidade, testes e defeitos são automaticamente relacionados aos requisitos e compilação que estão sendo testados, o que possibilita ajudar a rastrear a qualidade de requisitos. Com isso é possível executar testes no quadro *kanban* ou usar o *hub* de planos de teste para definir planos de teste e suítes de teste. No relatório e análise é apresentado gráficos de rastreamento configuráveis, *widgets* específicos de teste que podem ser adicionados nos painéis e relatórios integrados, como relatório de progresso, relatório de resultados de teste de *pipelines* e o serviço *analytics* (DOCS, 2022).

2.3.5 *Azure Artifacts*

Permite que a equipe compartilhe entre os membros, os artefatos de *build* gerado nos pipelines de compilação. Como é integrado com os *pipelines* o usuário pode automatizar a publicação do artefado (DOCS, 2022).

2.4 CI

De acordo com Pinto e Terra (2015), o CI é a utilização de práticas de desenvolvimento de sistemas, que automatizem o processo de testes e compilação do sistema. De acordo com Berg (2012) o obejtivo do CI é facilitar a detecção de erros e problemas antes da publicação

do sistema, evitando custos com o reparo.

Segundo Duvall, Matyas e Glover (2007) os servidores de CI podem ser configurados para assim que tenha atualização no repositório, o processo de *build* seja iniciado, resultando no artefato pronto para a publicação do sistema.

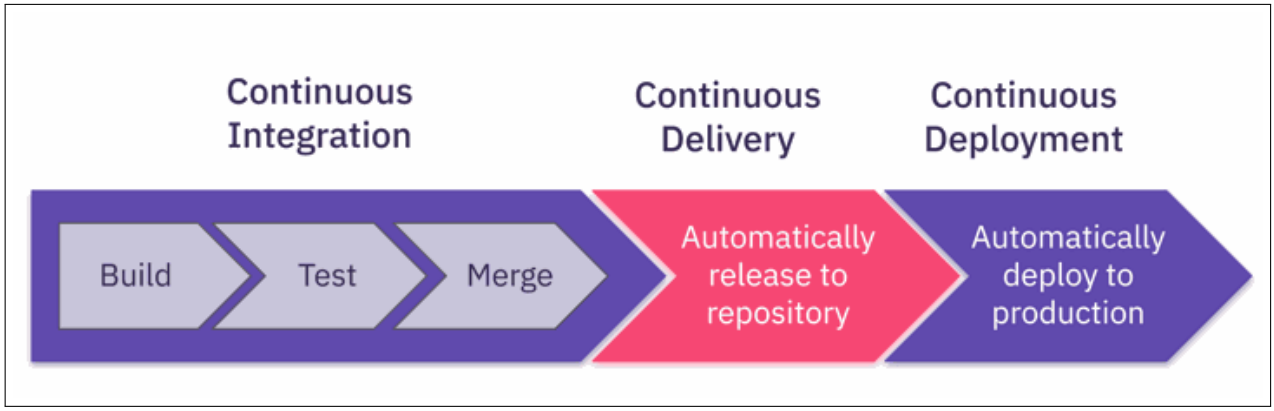
2.5 CD

De acordo com Humble e Farley (2014), o CD tem como objetivo reduzir os custos da entrega de um sistema. Com isso em mente é utilizado *scripts* e ferramentas de automação para garantir que o sistema está pronto para ser disponibilizado em produção a qualquer momento.

2.6 *CONTINUOUS DEPLOYMENT*

O *Continuous Deployment* se concentra na implantação real, enquanto o CD foca na estrategia de lançamento. Um paralelo seria "apertar um botão"para disponibilizar as alterações em produção. O "apertar do botão"é o CD. Já o *Continuous Deployment* é a disponibilização do sistema em produção, ficando disponível para o cliente (RAVI, 2021). Na Figura 4 pode-se observar a relação entre o CI, CD e o *Continuous Deployment*.

Figura 4 – Etapas do processo de *Continuous Deployment*.



Fonte**:** (WILKES, 2022)

2.7 *DOCKER*

Docker é uma empresa que desenvolve, assegura e divulga o *software* de mesmo nome *Docker*. O sistema é *open source* podendo ser utilizado na versão *Enterprise* ou *Community*. Ele é utilizado para gerenciamento de contêineres. O projeto do *Docker* foi desenvolvido dentro

do *Google*, que posteriormente disponibilizou o código fonte na linguagem *Golang* (GO). Hoje diverasas empresas utilizam o *Docker*, como a *Netflix*, *Google*, *Amazon* e *Spotify* (RAD; BHATTI; AHMADI, 2017).

2.7.1 Containers

De acordo com Rad, Bhatti e Ahmadi (2017) o container é uma forma de isolar um ambiente dentro de um dos três sistemas operacionais desde que sejam de 64 bits. São eles o *Windows*, *Linux* ou *Mac OS*.

Segundo Potdar *et al.* (2020) a conteinerização a tecnologia que combina o sistema, as suas dependências e as bibliotecas do sistema organizadas para construir na forma de um contêiner. Com o objetivo disponibilizar micro serviços, sem ter que instalar todo um Sistema Operacional (SO) para o novo serviço. Com isso é mais rápido subir um novo servidor *web*.

O container compartilha o *kernel*, bibliotecas e binários do SO que a executa, enquanto a máquina virtual tem seu próprio *kernel*, bibliotecas e binários (RAD; BHATTI; AHMADI, 2017).

2.7.2 *Docker Images*

Segundo o Docker (2022) a imagem é o conjunto ordenado de modificações no sistema raiz, variáveis e parâmetros utilizados em tempo de execução do container. Normalmente a imagem possui a união de sistemas de arquivos em camadas empilhados uns sobre os outros. A imagem é o conjunto de instruções para se criar o container.

Existe duas formas de criar imagens *Docker*. O primeiro método é o "confirmar uma alteração" no qual é feita uma imagem com base em um modelo de somente leitura. O modelo fundamenta-se em imagem de base, que podem ser de qualquer SO leve como Ubuntu 14.04 LTS ou Fedora 20. O segundo método é conhecido como o método automatizado de criar uma imagem no qual é feito um arquivo *Docker* que possui todas as instruções e regras para criar a imagem *Docker*. Ao executar o comando *docker build* no terminal, a imagem é criada com as dependências mencionadas no arquivo *Docker* (POTDAR *et al.*, 2020).

2.7.3 *Dockerfile*

O *Dockerfile* é um documento de texto que possui todas as instruções que geralmente são executadas manualmente para criar uma imagem *Docker*. Com o *Dockerfile* o *Docker* pode

Tabela 1 – VM vs Container

	VM	Conatiner
Nível de processo de isolamento	*Hardware*	SO
SO	Separado	Compartilhado
Tempo de inicialização	Longo	Curto
Uso de recursos	Em maior quantidade	Em menor quantidade
Imagens pré-construídas	Difícil de encontrar e gerenciar	Já disponível para servidor doméstico
Imagens pré-configuradas personalizadas	Difícil de construir	Fácil de construir
Tamanho	Maior porque eles contêm todo o SO por baixo	Menor com apenas mecanismos docker no SO *host*
Mobilidade	Fácil de mudar para um novo SO *host*	Destruído e recriado em vez de se mover
Hora de criação	Mais tempo	Em segundos

Fonte**:** (POTDAR *et al.*, 2020), modificado pelo autor.

criar as imagens automaticamente (DOCKER, 2022).

2.7.4 *Docker Registries*

O *Docker Registries* é um repositório de imagens *Docker*. O *Docker* disponibiliza dois tipos de registros o público e o privado. Nos registros do tipo público qualquer um pode extrair ou enviar imagens. O controle do tipo de registro é feito com o *Docker Hub*, as imagens podem ser separadas entre pública ou privada (RAD; BHATTI; AHMADI, 2017).

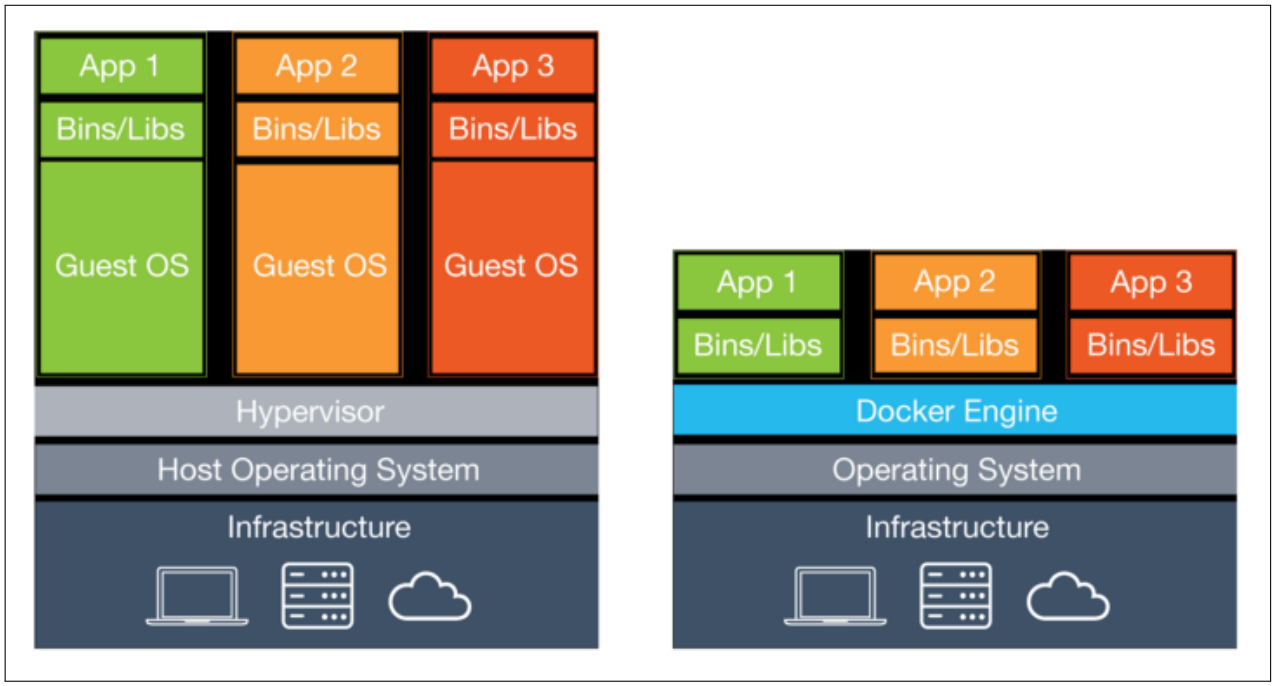
2.7.5 Arquitetura do container

Diferente da VM o container utiliza o *kernel* do SO em que o container esta disponível. Dessa forna não é preciso instalar um SO completo apenas para o container como é preciso na VM. O *Docker* desenvolveu a *Docker engine* para conseguir abstrair o SO e conseguir utilizar as bibliotecas e binários já existentes da maquina que está o container. Na Figura 5 pode-se obeservar as diferenças entre a arquitetura da VM e do container (POTDAR *et al.*, 2020).

O *Docker* cria processos, isolando em cada processo as dependências da aplicação para rodar os containers. Com isso o *Docker* garante que estejam instalados no container apenas as bibliotecas para o projeto funcionar. Assim não é preciso instalar um SO novo em cima de um já existente como é feito na VM (POTDAR *et al.*, 2020).

A arquitetura de contêineres é projetada para aproveitar as bibliotecas do sistema no qual o contêiner está implantado, o que resulta em várias vantagens ao utilizar essa abordagem. Na Tabela 1, é possível analisar as distinções entre Máquinas Virtuais (VM) e contêineres.

Figura 5 – Comparativo entre a arquitetura baseada em *Hypervisor* e a arquitetura baseada em *Container*



Fonte**:** (ROCHA, 2016)

2.8 IAC

De acordo com a RedHat (2022b) a IAC é o gerenciamento e provisionameto da infraestrutura por meio de códigos, como em *softwares*. As especificações e configurações da infraestrutura são salvas em arquivos que podem ser executados de forma automática. Com isso é assegurado o provisionamento do mesmo ambiente todas as vezes que o arquivo for executado. Tratar a infraestrutura como se fosse um código possibilita fazer a separação em módulos, que podem ser executados de forma automática. Com isso é evitado que os profissionais tenham que gerenciar manualmente os servidores, sistemas operacionais e outros componentes de infraestruturas.

Existe duas formas de utilizar a IAC, uma pela abordagem declarativa e outra pela abordagem imperativa. Na abordagem declarativa define o estado desejado do sistema, incluindo as propriedades necessárias, os recursos que ele precisa ter e uma ferramenta de IAC para configurar. A abordagem declarativa também conserva uma lista de estado dos objetos do sistema, facilitando o gerenciamento da desativação da infraestrutura. Na abordagem imperativa define as instruções específicas e necessárias para atingir a configuração planejada. Em seguida os comando são executados na ordem configurada. Nas ferramentas que usam a declarativa guardam de forma automática a infraestrutura planejada. Caso seja alterado o estado desejado, a

ferramenta declarativa aplica as alterações automaticamente. Na imperativa exige que o usuário conheça as alterações que deverão ser aplicadas. Normalmente as ferramentas de IAC funcionam nas duas abordagens. A utilização do IAC é fundamental para possibilitar as automações do CI/CD. Pois a automação tende a falhar quando a configuração do ambiente é diferente, o que não acontece com a utilização do IAC (REDHAT, 2022b).

Com a migração do provisionamento da forma manual, que é um método caro e demorado, da infraestrutura do *hardware* físico em *datatacenters* para virtualização, *cloud computing* e containers. O numero de componentes da infraestrutura aumentou, um maior numero de aplicações são disponibilizadas em produção frequentemente o que força que a infraestrutura seja flexível para suportar as frequentes alterações. Com a competitividade do mercado atual fica inviável não utilizar praticas de IAC para fazer o gerenciamento e escalonamento da infraestrutura. Dentre os benefícios da utilização de IAC, são eles:

1. Aumento na velocidade das implantações
2. Redução de custos
3. Redução de erros
4. Melhoria na consistência da infraestrutura
5. Eliminação de desvios de configuração

2.9 *SCRUM*

Segundo SCHWABER e SUTHERLAND (2017) o *Scrum* é uma metodologia estrutural para gerenciamento de trabalhos em produtos complexos, utilizada desde 1990. No *Scrum* é proposta a eficácia relativa de suas práticas de gerenciamento de produto e técnicas de trabalho. O ponto chave do *Scrum* é a possibilidade de melhoria dos processos de acordo com as necessidades do projeto e equipe. Os pilares empíricos do *Scrum*, são: transparência, inspeção e adaptação.

Na transparência, considera-se que as atividades relevantes do projeto devem estar sempre disponíveis para os responsáveis do projeto. Na inspeção, o artefato do *Scrum* deve ser inspecionado para se observar variações indesejadas que podem atrapalhar no resultado final. As inspeções não devem ser frequentes a ponto de atrapalhar o objetivo do projeto. As inspeções deve ser realizada de forma diligente por inspetores especializados no trabalho a se verificar. Na adaptação, é feito ajustes no processos para que o objetivo do projeto seja alcançado, de forma mais rápida possível para minimizar desvios no objetivo. A adaptação é feita com base no

resultado da inspeção (SCHWABER; SUTHERLAND, 2017).

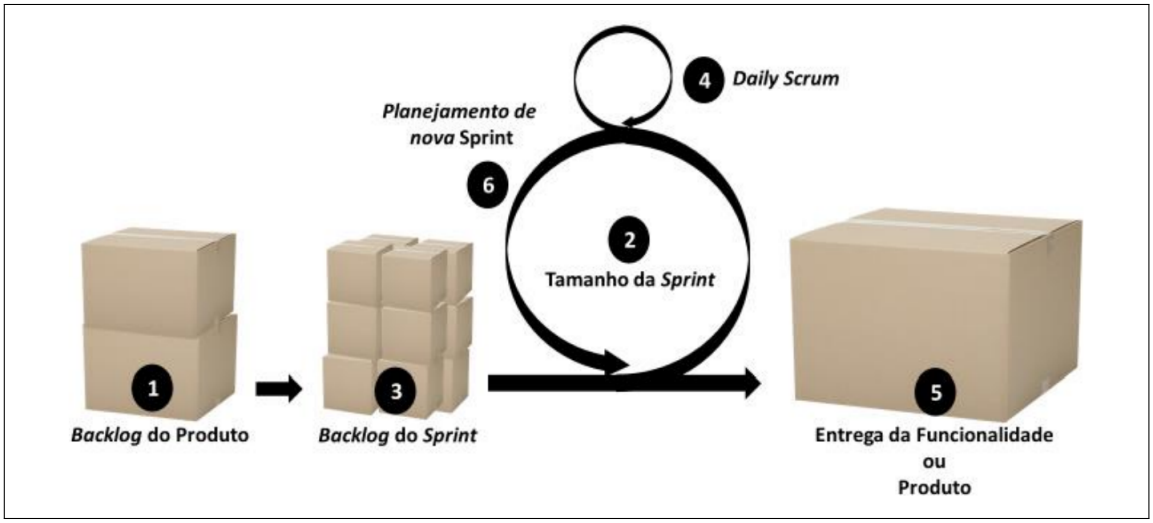
Como é apresentado em STOPA e RACHID (2019), no cerne do *Scrum* o trabalho é realizado com equipes pequenas altamente flexíveis e adaptativas. As equipes são conhecidas como *Times Scrum*, em que são compostas pelos *Scrum Master*, *Product Owner* e o Time. O *Product Owner* é o responsável por maximizar o trabalho da equipe e o valor do produto desenvolvido. O *Scrum Master* por garantir que as regras de boas práticas da metodologia sejam aplicadas durante o projeto. O time são os profissionais capacitados para realizarem o projeto.

O *Scrum* é composto por seis etapas que podem ser adaptadas de acordo com a equipe e projeto, São elas:

1. *Backlog* do produto
2. Tamanho da *sprint*
3. *Backlog* da *sprint*
4. *Daily scrum*
5. Entrega da funcionalidade ou produto
6. Planejamento da nova *Sprint*

O processo do *Scrum* se inicia com *backlog* do produto, em que são levantados todos os requisitos e a lista do que precisa ser desenvolvido. Por meio de reuniões com os clientes são levantadas todas as funcionalidades e necessidades do projeto em ordem de prioridade. Com o *backlog* do produto terminado, se inicia as reuniões de planejamento da *Sprint*. No *backlog Sprint* tem como objetivo organizar as demandas por ordem de prioridade e definir o tempo de duração que cada ciclo vai levar para realizar uma entrega, com um determinado número de demandas realizadas. O ciclo de entrega costuma ser um ciclo curto de uma ou duas semanas, mas mudando de acordo com a equipe e projeto. Com todas demandas organizadas, e o tempo de cada entrega definido é iniciado a *sprint*. A *sprint* é o período de tempo de execução das atividades definidas no *backlog* da *Sprint*, no qual uma parte do produto é entregue. No final da *sprint* os defeitos e demandas não entregues são adicionados no *backlog* do produto e em seguida a uma reunião de revisão da *sprint*. Na revisão da *sprint* a equipe revisa os acertos e erros, para ressaltar as lições aprendidas. Durante a *sprint* são realizadas reuniões diárias e curtas, em que a equipe conversa sobre as atividades realizadas e problemas encontrados. Na Figura 6 é apresentado o fluxo da metodologia *srum*.

Figura 6 – Ritos do scrum



Fonte**:** (STOPA; RACHID, 2019)

2.10 APACHE MAVEN

Mavem é uma palavra *Yiddish*, que significa acumulador de conhecimento. O maven foi desenvolvido com o objetivo de facilitar a publicação de informações do projeto e compartilhar os JARs de um projeto java. O resultado final foi uma ferramenta que torna possível construir e gerenciar qualquer projeto java (MAVEN, 2022). Na documentação do Maven (2022) é possível observar as áreas de preocupação do maven, são elas:

- Facilitando o processo de construção
  - O maven não elimina a necessidade de conhecer os utensílios implícitos.
- Fornecendo um sistema de construção uniforme
  - O maven constrói o projeto usando o modelo de *Project Object Model* (POM) e o conjunto de *plugins*.
- Fornecendo informações de projeto de qualidade
  - *Log* de alterações criado diretamente do controle de origem
  - Fontes de referência cruzada
  - Listas de discussão gerenciadas pelo projeto
  - Dependências utilizadas no projeto
  - Relatórios de teste unitário
- Incentivando melhores práticas de desenvolvimento

3 METODOLOGIA

Este capítulo é divido em dois tópicos, o Proposito e o Procedimento prático. No Proposito é apresentado a descrição do processo para realizar a automação de *build* e *deploy*. No Procedimento prático é descrito, em subtópicos, as fases do procedimento pratico, que são:

- *Hardware* e *Software* utilizados.
- Configuração do *Token* de autenticação.
- Configuração do *pool* de agente.
- Desenvolvimento do arquivo *Dockerfile*.
- Configuração do *pipeline* CI.
- Configuração do *pipeline* CD.

3.1 PROPÓSITO

Este trabalho visa descrever de forma prática a automação do processo de *build* e *deploy*, aplicada no Sistema Fecomércio (Sesc e Senac) do estado do Ceará. Para isso desenvolveu-se uma VM para ser utilizado como um servidor *docker* responsáveis por fazer o *build* das aplicações. A ferramenta *Azure DevOps* é utilizada para fazer toda a parte de gerenciamento e controle, quando o repositório do projeto na ferramenta identifica um novo *commite* ela aciona o agente que instalado e configurado no container que fica no servidor *docker*. Com processo de *build* finalizado o *Azure DevOps* pega o artefato de *build*, que ate então esta no container, e salva finalizando o processe de *build* automático. Com isso o processo de *deploy* automático fica aguardando a permissão do usuário que foi configurado como responsável para poder disponibilizar o artefato de *build* em ambiente de homologação e tudo dando certo seguir para a publicação em ambiente de produção.

De forma prática foi configurado a VM como servidor *docker*, a ferramenta de *pipelines* e *Releases* do *Azure DevOps*.

3.2 PROCEDIMENTO PRÁTICO

3.2.1 *Hardware* e *Software*

Antes de iniciar a configuração do *pipeline* de CI/CD foi instalado o *docker* em uma VM *Windows*. O *docker* utilizado foi o da versão é a 19.03.19. A VM *Windows* foi com

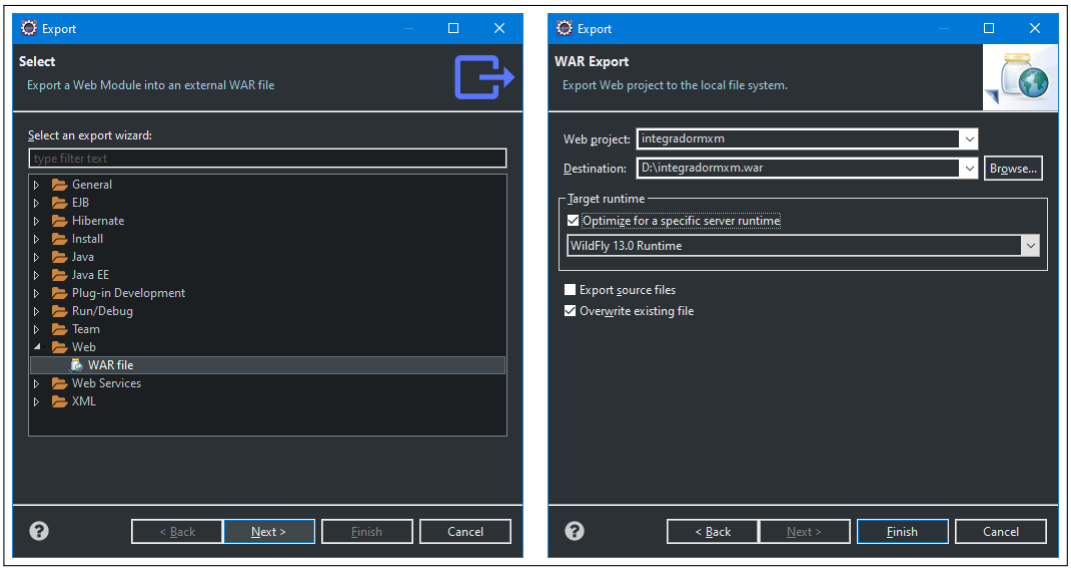
Microsoft Windows Server 2019 datacenter com 9 GB de Memória de Acesso Aleatório (RAM) e com o processador intel *Xenon* 2.60 GHz 4 *core*.

Para o desenvolvimento dos *pipelines* foi configurado um containers, dentro da VM, com o servidor *wildfly-13.0.0*, *maven 3.8.4* e o java na versão 8. Com isso facilita subir outros containers com outros tipos de servidores ou para outros tipos de aplicações.

3.2.2 Processo de *Build* e *Deploy* manual

O processo de *Build* e *Deploy* manual consiste em uma serie de etapas que iniciam com o *build* do projeto. Operando o Eclipse como *IDE* utiliza-se a opção de exportar o *WAR* do projeto. Na Figura 7 pode-se observar como a configuração para exportar o *WAR* do projeto é feita gerando o artefato de *build*.

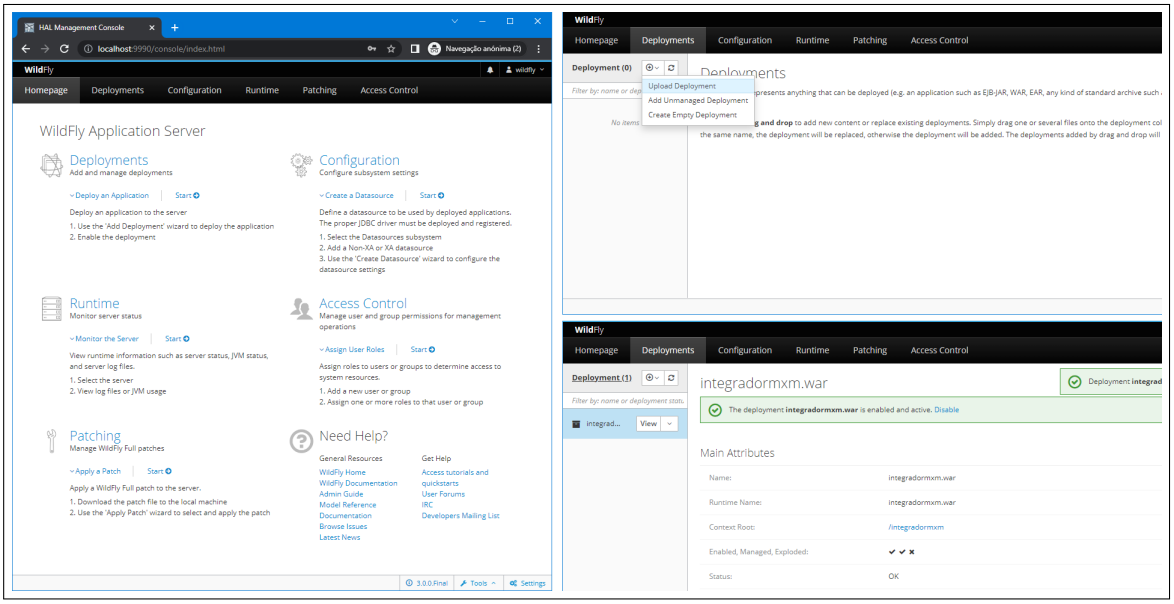
Figura 7 – Exportar o *WAR* do projeto



Fonte**:** O autor

O próximo passo é adicionar o artefato de *build* do projeto no servidor. Para isso utiliza-se a interface gráfica do servidor para subir o artefato de *build*. Na Figura 8 pode-se observar os passos para adicionar o artefato de *build* no servidor. Para realizar todo o processo de *build* e *deploy* são aproximadamente 12 etapas que vão desde a exportação do *WAR* do projeto, passando pelo *login* no painel de administração do servidor até selecionar o arquivo *WAR* para subir no servidor.

Figura 8 – Adição do artefato de *build* no servidor



Fonte**:** O autor

3.2.3 Configuração do *Token* de autenticação

Para o desenvolvimento dos *pipelines* é preciso configurar o *token* de autenticação, para possibilitar a comunicação do container com *Azure DevOps*. Na configuração do *token* é informado que ele tem permissão de gerenciar e ler o *pool* de agente. Nele também é informado a data de validade do *token*.

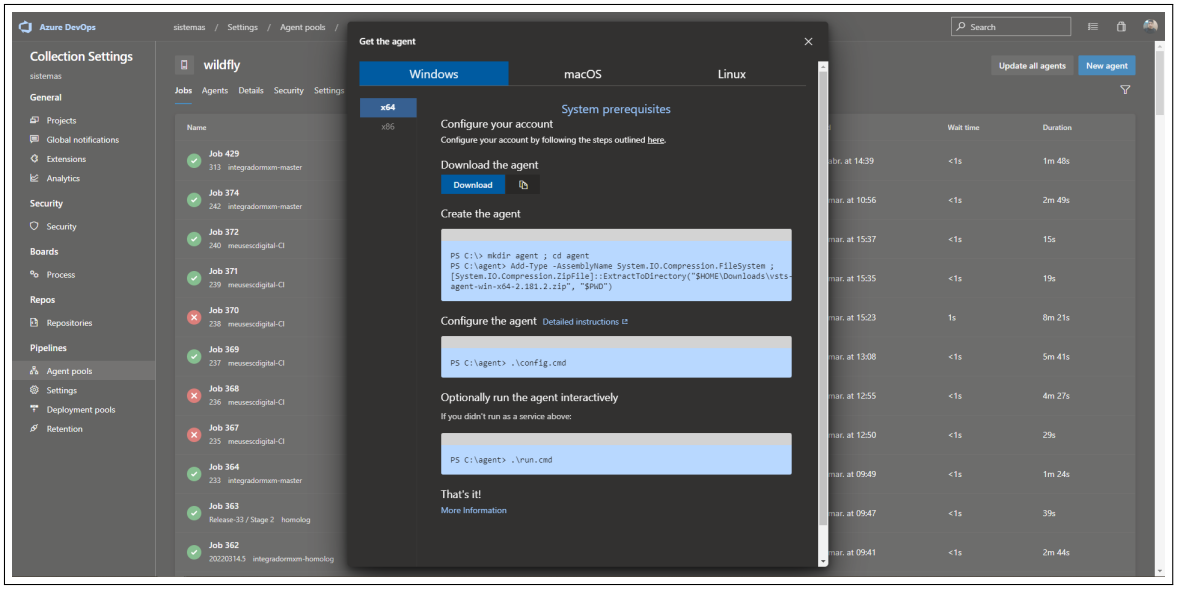
3.2.4 Configuração do *pool* de agente

Depois da criação e configuração do *token* é criado o *pool* de agentes, que faz o gerenciamento de todos os agentes adicionados ao *pool*. Com o agente criado é baixado o arquivo para instalação do agente na VM ou container que vai executar o *pipeline*. Na página do *Azure DevOps*, em que é baixado os arquivos do agente também possui as instruções para instalar e configurar o agente. Na Figura 9 é apresentado a página do *Azure DevOps* que disponibiliza as informações para baixar, instalar e configurar o agente.

3.2.5 Desenvolvimento do arquivo *Dockerfile*

Para criar as imagens e com ela subir o container, configurou-se um arquivo *dockerfile*. A configuração do *dockerfile* é feita para criar uma imagem, que quando é executada, ela inicia o processo de criar um novo container. No processo de subir um novo container, ele copia todos os

Figura 9 – Instruções para instalar o agente



Fonte**:** O autor

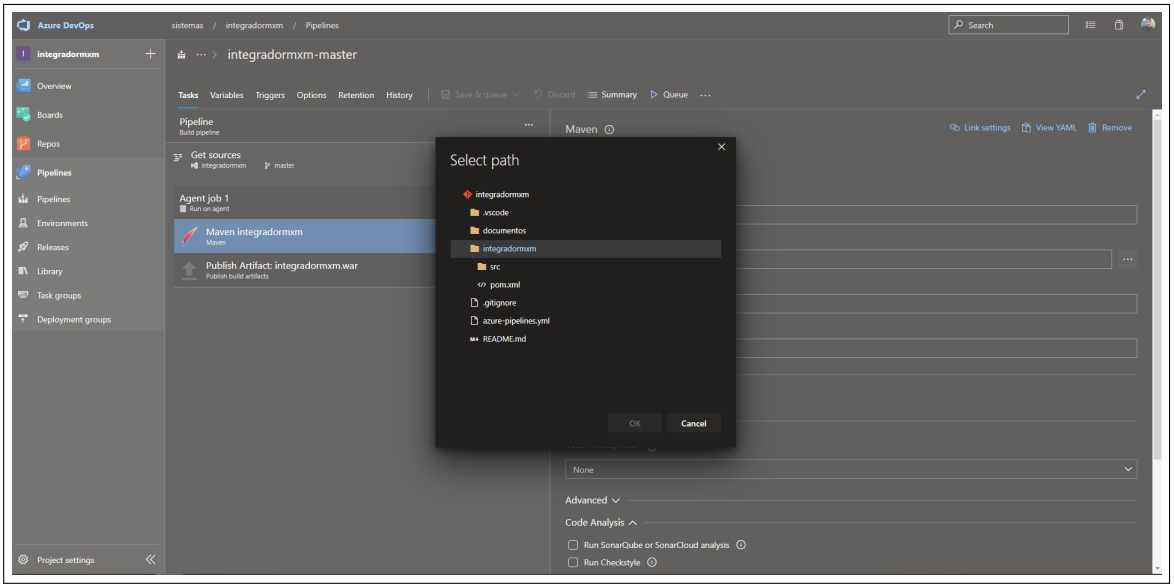
arquivos que precisar para rodar a aplicação e para se comunicar com o *Azure DevOps*. Ainda no processo de subir um novo container, depois de ter todos os arquivos copiados para dentro do container, é executado o comando para configurar as variáveis de ambiente. Para finalizar a criação do novo container é executado o comando para iniciar o servidor *wildfly-13.0.0*. Para criar uma nova imagem é utilizada o comando *docker build* e para subir um novo container com base na imagem criada é utilizado o comando *docker run* junto com o *token* de acesso e nomo do *pool* de agentes que esse container vai utilizar.

3.2.6 Configuração do *pipeline* CI

Com o container *online* e conectado com o *Azure DevOps* basta configurar o *pipeline*, informando em qual *pool* de agentes ele vai executar as instruções do CI. Nas instruções do *pipeline* CI é configurado a forma como o projeto faz o *build* e aonde é salvo o resultado do *build*. Na Figura 10 pode-se observar o local em que o arquivo com as instruções do *Maven* do projeto é salvo e que é utilizado para fazer o *build*.

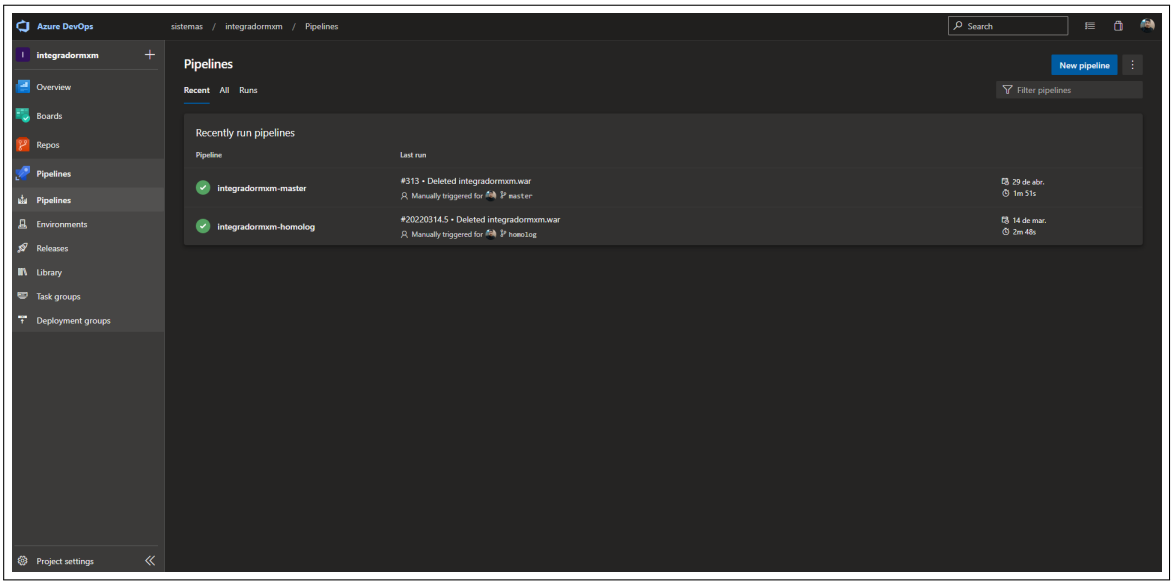
Com o *pipeline* de CI configurado ele faz o *build* do projeto sempre que identifica um novo *commit*. O resultado do *build* é exibido na tela principal de *pipelines*. Na Figura 11 pode-se observara o resultado dos teste de *build* do projeto.

Figura 10 – Local do arquivo com informções de *build* do projeto.



Fonte**:** O autor.

Figura 11 – Resultado de *build* do projeto.



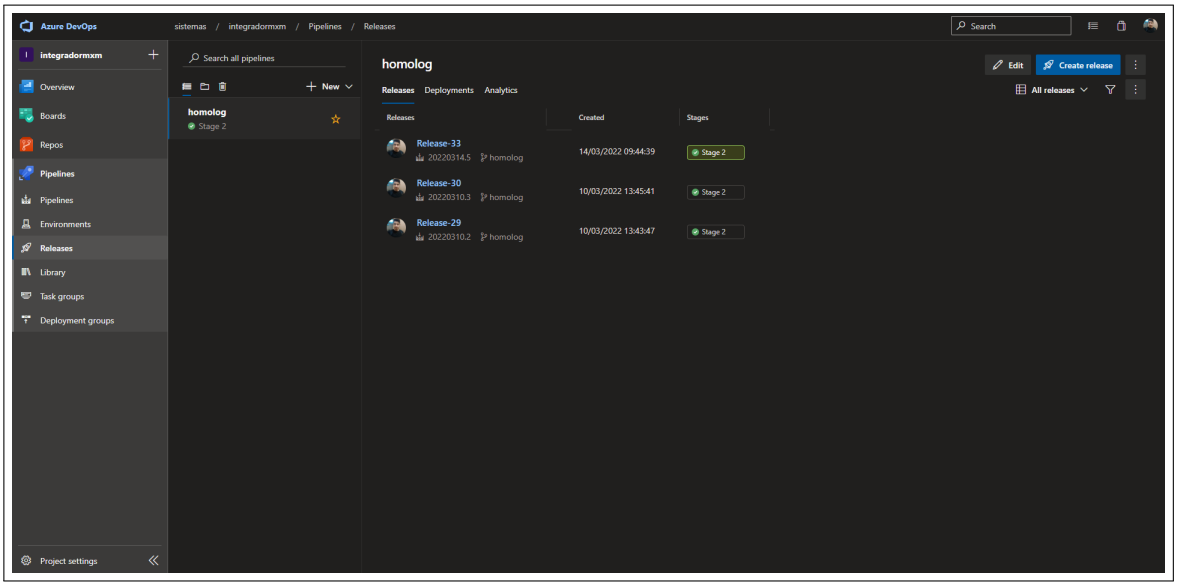
Fonte**:** O autor.

3.2.7 Configuração do *pipeline* CD

A configuração do *pipeline* de CD consiste em definir as pessoas que possuem permissão para liberar a publicação, quando a publicação sera feita e o local aonde o projeto vai ser publicado. Depois que o *build* do projeto é realizado o *Azure DevOps* salva o arquivo para publicar o projeto e fica aguardando a liberação das pessoas autorizadas para fazer a publicação no local indicado. Assim como no *build* o *pipeline* de CD também possui uma tela em que o resultado da publicação é exibida, mostrado se deu certo, errado ou se está aguardando liberação

para publicação. Na Figura 12 pode-se observar os resultado dos testes de publicação do projeto.

Figura 12 – Resultado de publicação do projeto.



Fonte**:** O autor.

4 CONCLUSÃO

Conclui-se que o projeto foi bem sucedido, pois com a utilização do *pipeline* de CI e CD, com as boas praticas da cultura *DevOps* foi possível automatizar o processo de *build* e *deploy* da aplicação. O processo automatizado é iniciado sempre que um novo *commit* é identificado na *branch*, finalizando com a publicação de uma versão estável da aplicação.

Com o processo de *build* e *deploy* automatizado a equipe de desenvolvimento e operações conseguem evitar de fazer aproximadamente 12 etapas, que vão desde a exportação do *WAR* do projeto, passando pelo *login* no painel de administração do servidor até selecionar o arquivo *WAR* para subir no servidor. Etapas que tornam a entrega do *software* lenta e passível de erros.

Com as melhorias descritas aplicadas, o projeto obteve em modo geral, ganhos na performance de operação, analise e identificação visual de eventuais erros no meio do processo de *build* e *deploy*. Com isso, evitando que a equipe de desenvolvimento e operações perdessem tempo com processos repetitivos e voltassem os olhares em atividades complexas como o desenvolvimento de novas aplicações ou correções de eventuais erros.

REFERÊNCIAS

BECK, K.; BEEDLE, M.; BENNEKUM, A. V.; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R. *et al.* Manifesto for agile software development. Snowbird, UT, 2001.

BERG, A. **Jenkins Continuous Integration Cookbook**. [S.l.]: Packt Publishing Ltd, 2012.

CAMÕES, R. J. da S.; SILVA, J. A. da. Desenvolvimento e operações utilizando ansible como principal ferramenta de implantação: Estudo de caso no tribunal de justiça do distrito federal (tjdft). **TECNOLOGIAS EM PROJEÇÃO**, v. 9, n. 2, p. 36–52, 2018.

DEBOIS, P. Agile infrastructure and operations: how infra-gile are you? In: IEEE. **Agile 2008 Conference**. [S.l.], 2008. p. 202–207.

DOCKER. **Glossary | Docker Documentation**. 2022. <https://docs.docker.com/glossary/>. (Acessado em 08/16/2022).

DOCS, M. **Documentação técnica da Microsoft**. 2022. <https://docs.microsoft.com/pt-br/?view=azure-devops>. (Acessado em 08/11/2022).

DUVALL, P. M.; MATYAS, S.; GLOVER, A. **Continuous integration: improving software quality and reducing risk**. [S.l.]: Pearson Education, 2007.

FABER, F. Testing in devops. In: **The Future of Software Quality Assurance**. [S.l.]: Springer, Cham, 2020. p. 27–38.

FARIA, M. R. Devops para infraestrutura de redes. 2017.

GUEST, D. The hunt is on for the renaissance man of computing. **The Independent**, v. 17, n. 9, 1991.

HERZBERG, F. **Motivation to work**. [S.l.]: Routledge, 2017.

HUMBLE, J.; FARLEY, D. **Entrega contínua: como entregar software de forma rápida e confiável**. [S.l.]: Porto Alegre: Bookman, 2014.

KIM, G.; BEHR, K.; SPAFFORD, K. **The phoenix project: A novel about IT, DevOps, and helping your business win**. [S.l.]: IT Revolution, 2014.

MAVEN. **Maven – Introduction**. 2022. <https://maven.apache.org/what-is-maven.html>. (Acessado em 08/19/2022).

PINTO, A. F.; TERRA, R. Processo de conformidade arquitetural em integração contínua. **2nd Latin-American School on Software Engineering (ELA-ES)**, p. 1–12, 2015.

POTDAR, A. M.; NARAYAN, D.; KENGOND, S.; MULLA, M. M. Performance evaluation of docker container and virtual machine. **Procedia Computer Science**, Elsevier, v. 171, p. 1419–1428, 2020.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software-9**. [S.l.]: McGraw Hill Brasil, 2021.

RAD, B. B.; BHATTI, H. J.; AHMADI, M. An introduction to docker and analysis of its performance. **International Journal of Computer Science and Network Security (IJCSNS)**, International Journal of Computer Science and Network Security, v. 17, n. 3, p. 228, 2017.

RAVI. **Continuous Delivery vs. Continuous Deployment: What's the Difference? | Harness**. 2021. <https://harness.io/blog/continuous-delivery-vs-continuous-deployment#:~:text=Continuous%20Delivery%20is%20the%20automation,to%20get%20changes%20into%20production.> (Acessado em 09/22/2022).

REDHAT. **O que são pipelines de CI/CD?** 2022. <https://www.redhat.com/pt-br/topics/devops/what-cicd-pipeline>. (Acessado em 08/25/2022).

REDHAT. **O que é infraestrutura como código (IaC)?** 2022. <https://www.redhat.com/pt-br/topics/automation/what-is-infrastructure-as-code-iac>. (Acessado em 08/18/2022).

ROCHA, V. H. **Docker — O que é Docker e como começar | by Victor Hugo Rocha | Training Center | Medium**. 2016. <https://medium.com/trainingcenter/docker-o-que-%C3%A9-docker-e-como-come%C3%A7ar-58e04bdcb043>. (Acessado em 09/23/2022).

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

SCHWABER, K.; SUTHERLAND, J. Guia do scrum–um guia definitivo para o scrum: As regras do jogo. 2013. **http://www. scrumguides. org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR. pdf>. Citado**, v. 3, p. 49, 2017.

STOPA, G. R.; RACHID, C. L. Scrum: Metodologia ágil como ferramenta de gerenciamento de projetos. **CES Revista**, v. 33, n. 1, p. 302–323, 2019.

VUPPALAPATI, C.; ILAPAKURTI, A.; CHILLARA, K.; KEDARI, S.; MAMIDI, V. Automating tiny ml intelligent sensors devops using microsoft azure. In: IEEE. **2020 ieee international conference on big data (big data)**. [S.l.], 2020. p. 2375–2384.

WILKES, A. **A Crash Course on Continuous Delivery | Launchable, Inc.** 2022. <https://www.launchableinc.com/a-crash-course-on-continuous-delivery>. (Acessado em 09/23/2022).